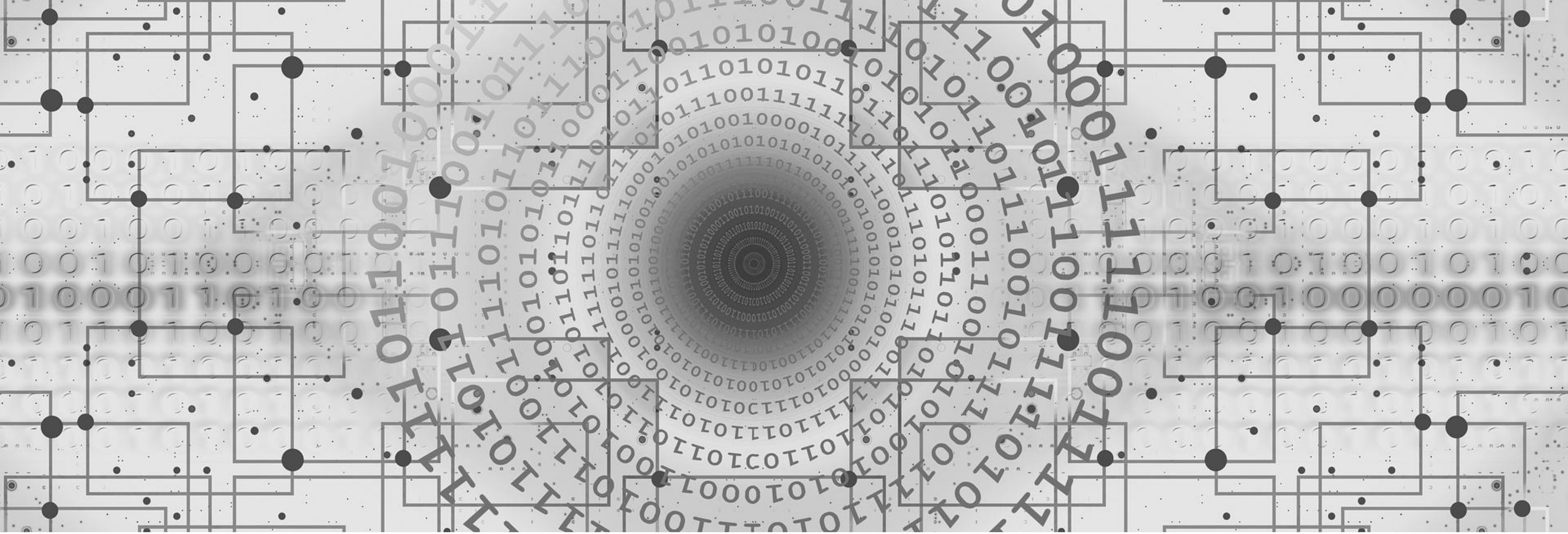


Homework-4 LFSR and Barrel Shifter

ECE-111 Advanced Digital Design Project

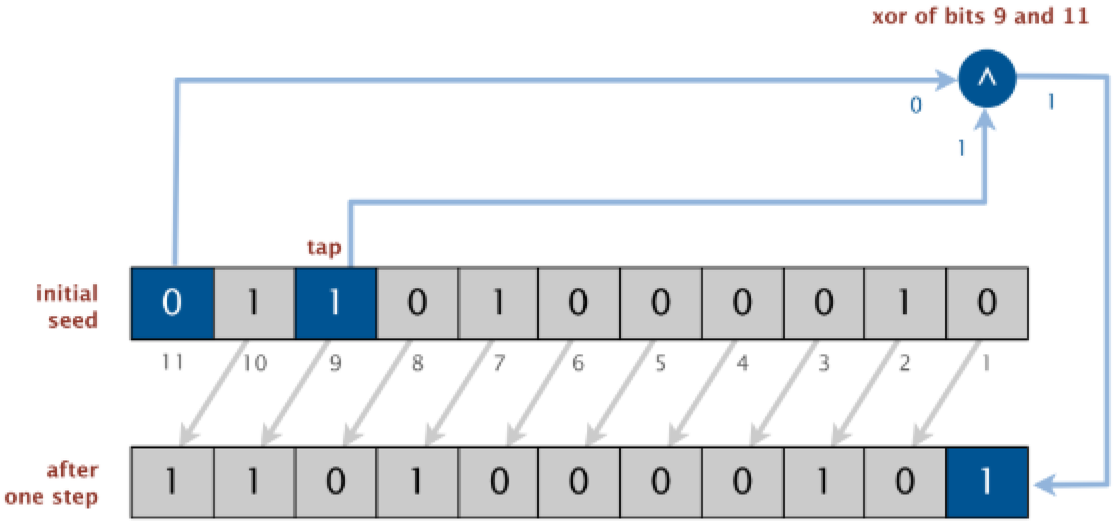
Winter 2022

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

Homework-4a : Linear Feedback Shift Register Design Concepts and Requirements

Linear Feedback Shift Register (LFSR)

- **An LFSR is type of a shift register which performs following mentioned steps:**
  - Does exclusive-or (xor or xnor) of bits at given tap positions in shift register
  - Shifts the bits one position to the left
  - Replaces the vacated bit with the output of exclusive-or (xor) function



- **There are three important parameters of LSFR :**
  - Number of bits *n*,
  - Initial *seed* (the sequence of bits that initializes the register)
  - Tap position to perform exclusive-or

Linear Feedback Shift Register (LFSR)

- **LFSR is a n-bit counter exhibiting *pseudo-random* behavior.**
  - Random numbers from LSFR are generated based on initial values loaded to LFSR (known as a seed value)
  - For each initial seed value there is a deterministic pattern of counting
  - It requires very less hardware
    - Built from simple shift-registers with a small number of xor gates.
    - N-bit LSFR would require N number of flip-flops
    - N-bit LSFR will generate deterministic 2^N-1 data patterns
  - Can perform very fast

- **Some of the applications of LFSR :**
  - Built in self testing (BIST)
  - Psuedo-random number generation
  - Counters
  - error checking and correction
  - Data compression
  - Wireless communication

Linear Feedback Shift Register (LFSR)

- □ **The LFSR is maximal-length if and only if the corresponding feedback polynomial is primitive. This means that the following conditions are necessary (but not sufficient):**
  - The number of taps is even
  - The set of taps is setwise co-prime. i.e., there must be no divisor other than 1 common to all taps.

- □ **Table of primitive polynomials from which maximum-length LFSRs can be constructed are given below and in the references.**

Bits(n)	Feedback Polynomial	Max Length / Period ($2^N – 1$)
2	$x^2 + x^1 + 1$	3
3	$x^3 + x^2 + 1$	7
4	$x^4 + x^3 + 1$	15
5	$x^5 + x^3 + 1$	31
6	$x^6 + x^5 + 1$	63
7	$x^7 + x^6 + 1$	127
8	$x^8 + x^6 + x^5 + x^4 + 1$	255

LFSR polynomial $x^3 + x^2 + 1$ indicates that the feedback is provided through output of 'xor' gate whose inputs are connected to positions 3 and 2 of LFSR

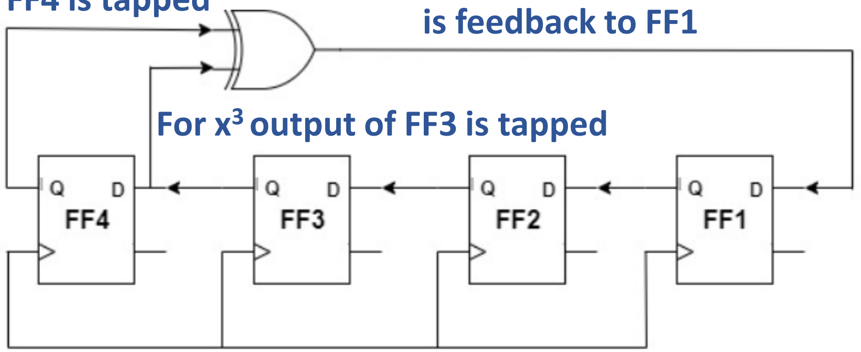
Linear Feedback Shift Register (LFSR)

❑ Example of 4-bit LFSR pseudo random generation:

- Circuit counts through *(2^4-1)=15* different non-zero bit patterns.
- Polynomial for maximum-length 4-bit LSFR is : $x^4 + x^3 + 1$
- To implement this polynomial, output of FF4 and FF3 are taped and exclusive-or using its current values before left-shifting
- LSB bits 0 to 2 are left-sifted by one position
- Vacated LSB bit-0 is replaced with output of exclusive-or

For x^4 output of FF4 is tapped

For x^0 = 1, output of xor gate is feedback to FF1



For x^3 output of FF3 is tapped

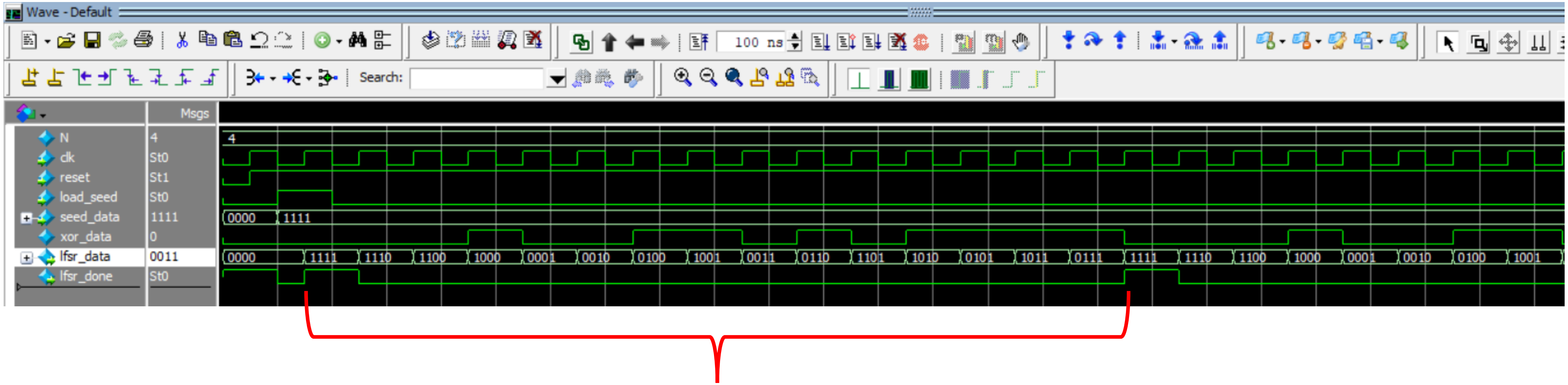
4-bit LFSR Polynomial : $x^4 + x^3 + 1$

Clock Cycle	FF4	FF3	FF2	FF1	Output of XOR
1	1	1	1	1	0
2	1	1	1	0	0
3	1	1	0	0	0
4	1	0	0	0	1
5	0	0	0	1	0
6	0	0	1	0	0
7	0	1	0	0	1
8	1	0	0	1	1
9	0	0	1	1	0
10	0	1	1	0	1
11	1	1	0	1	0
12	1	0	1	0	1
13	0	1	0	1	1
14	1	0	1	1	1
15	0	1	1	1	1
16	1	1	1	1	0
17	1	1	1	0	0

4-bit deterministic max pattern repeats for initial see value after every 15 cycles

Linear Feedback Shift Register (LFSR)

❑ 4-bit LSFR Simulation Result



Wave - Default

Search:

100 ns

	Msgs
N	4
clk	St0
reset	St1
load_seed	St0
seed_data	1111
xor_data	0
lfsr_data	0011
lfsr_done	St0

4-bit pseudo random pattern repeating after 15 cycles

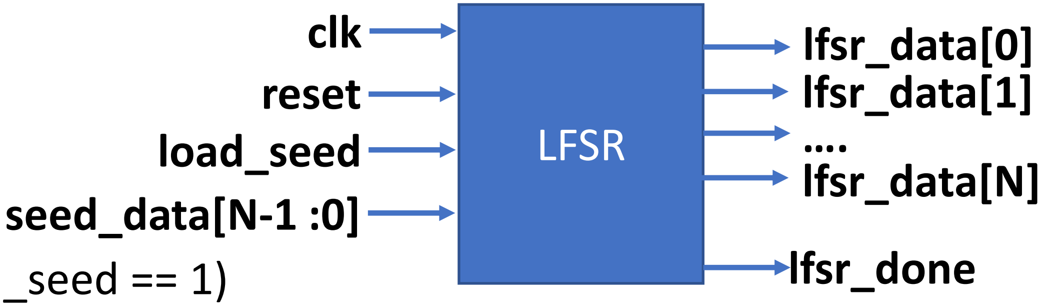
Homework Assignment-4a : LFSR

❑ Develop SystemVerilog RTL model for N-bit Linear Feedback Shift Register (LFSR) :

- Value N can be either 2,3,4,5,6,7,8
- In same SystemVerilog module implement max-length LSFR pseudo-random generation polynomial to support each of the above mentioned N values
- Use **xor** implementation for LFSR SystemVerilog RTL model development
- Synthesize LFSR and run simulation using LSFR testbench provided
  - Testbench provided for 4-bit LSFR. In testbench file parameter N can be set to some other value say 5 and run simulation
- Review synthesis results (resource usage and RTL netlist/schematic)
- Review input and output signals in simulation waveform.
- Assume below mentioned primary port names and SystemVerilog RTL module name **lfsr.**

❑ Primary Ports for LFSR module

- Input clk (clock)
- Input reset (asynchronous reset / negedge signal)
- Input load_seed (synchronous and active high signal)
- Input seed_data(initial seed data gets loaded when load_seed == 1)
- Output lfsr_data (output of each Flipflop on shift register)
- Output lfsr_done (once N-bit battern generation is completed and cycle before repeatition of pattern start, generate 1 cycle pulse for lfsr_done



Homework Assignment-4a : LFSR

❑ **Submit report which should include :**

- SystemVerilog design code
- Synthesis resource usage and schematic generated from RTL netlist viewer
- Simulation snapshot and explain simulation result to confirm RTL model developed works as a LFSR
- Post-Mapping schematic is optional to submit.
- Explanation of FPGA resource usage in report is not required.

Note :

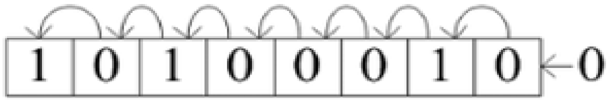
- Lab4 folder includes full testbench code and partial design template code for LFSR register
- It is not mandatory to use design template code provided in lab folder. Student can implement their design module from scratch without referring to template code as long as primary port list is matching with in previous slide. This is to ensure testbench is compatible with design.
- For learning purpose, student can change the stimulus in initial block in testbench file.

Homework-4b : Barrel Shifter Design Concepts and Requirements

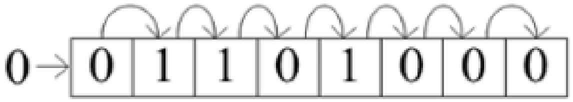
Barrel Shifter

- ❑ **Performs shift or rotate operations by specified number of bits in same clock cycle**
  - Purely combinational circuit and result of operation is available within same clock cycle
  - Similar to Bi-directional universal shift register, however universal shift register shifts one place in one cycle whereas barrel shifter can shift many places in same cycle and the shift register is a sequential logic
- ❑ **Barrel shifters can perform below mentioned operations :**
  - **Logical Shift Right (LSR)**, **Logical Shift Left (LSL)**, **Rotate Left (ROL)** and **Rotate Right (ROR)** operations

When shifting left by 1, the most-significant bit is lost, and a 0 is inserted on least-significant bit



Shift Logical Left

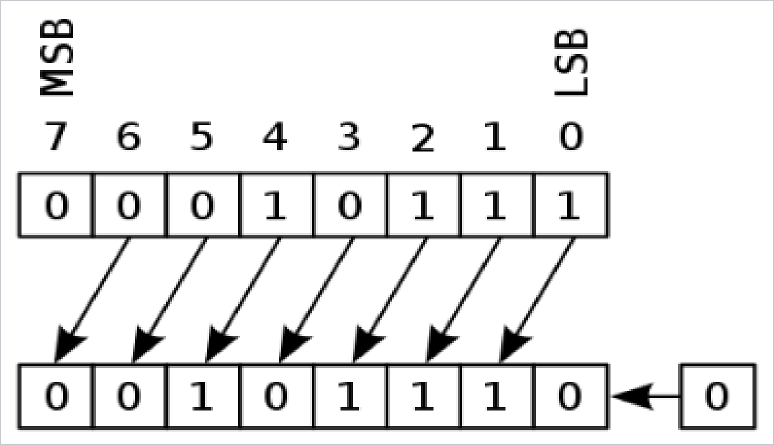


Shift Logical Right

When shifting right by 1, the least-significant bit is lost, and a 0 is inserted on most-significant bit

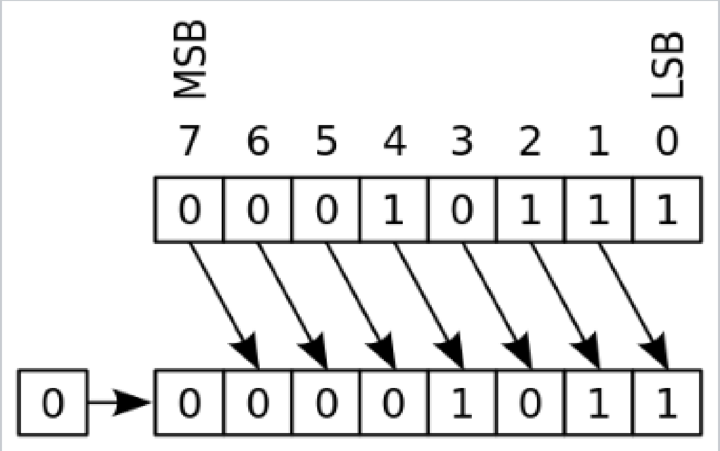
Binary : 8'b00010111 (decimal value : 23)

After Logical Shift Left by '1', New Binary : 8'b00101110 (New decimal value : 56 Value doubled after LSL by 1)



Logical Shift Left By 1 Position (LSL>>1)

LSL by '1' is same as multiply by 2
LSL by '2' is same as multiply by 4
LSL by '3' is same as multiply by 8



Binary : 8'b00010111 (decimal value : 23)

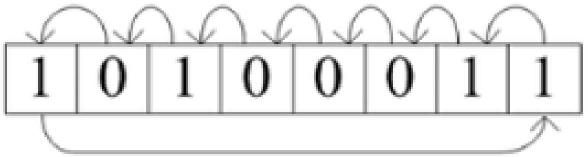
After Logical Shift Right by '1', New Binary : 8'b0001011 (New decimal value : 11 Value halved after LSR by 1)

Logical Shift Right By 1 Position (LSR >>1)

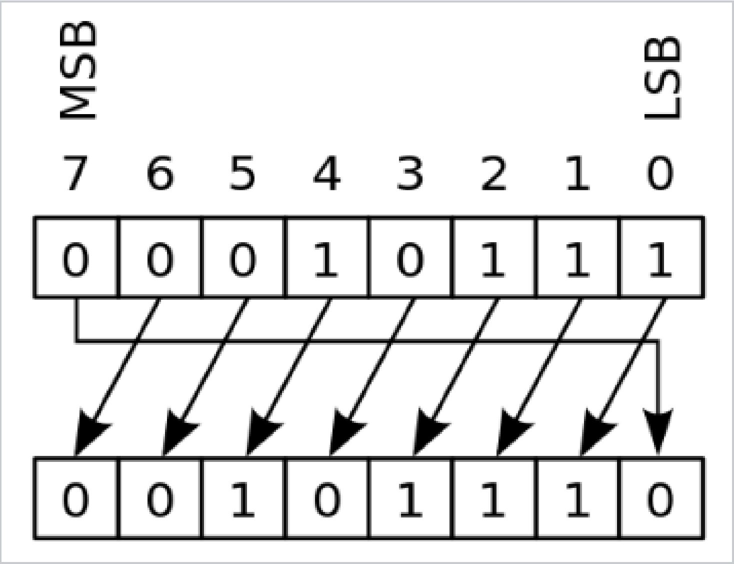
LSR by '1' is same as divide by 2
LSR by '2' is same as divide by 4

Note : MSB = Most Significant Bit, **LSB** = Least Significant Bit

Barrel Shifter

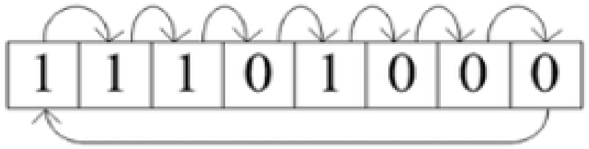


Rotate Left

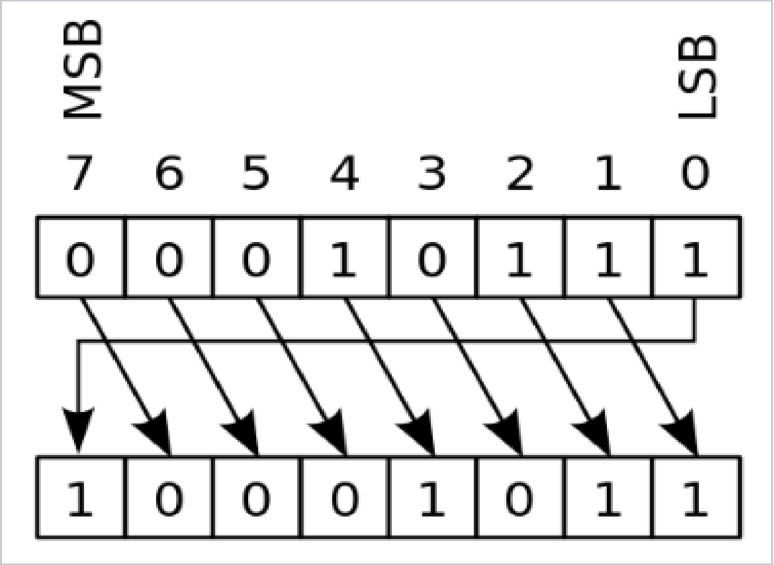


Rotate Left (ROL) By 1 Position

When rotating left by 1, MSB is copied to LSB and all bits are shifted left by ‘1’ position. No bits are lost in rotate operation



Rotate Right



Rotate Right (ROR) By 1 Position

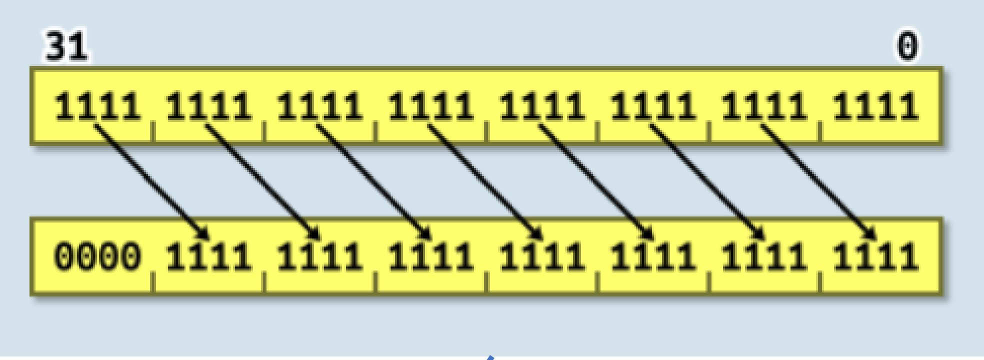
When rotating right by 1, LSB is copied to MSB and all bits are shifted right by ‘1’ position. No bits are lost in rotate operation

Barrel Shifter

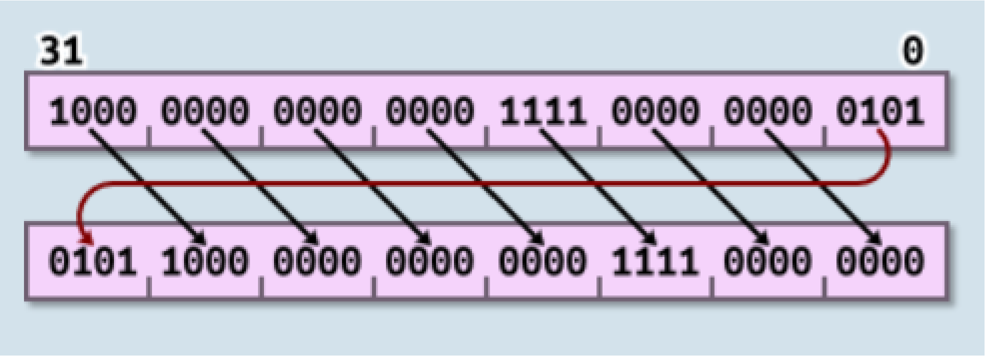
- ❑ **Shift and Rotate can move input data bits by any number of bit positions**
  - 4-bit barrel left shift can shift by 0, 1, 2 or 3 bit positions
  - 8-bit barrel left shift can shift by 0, 1, 2, 3, 5, 6, 7 bit positions
    - **Shifting an 8-bit number by 8 bit positions is pointless since all the bits will be lost**

- ❑ **Typically designed as a natural size (2, 4, 8, 16, 32 …) barrel shifters**
  - See below example right shift and rotate by 4 bit positions in a 32 bit number

Logical Shift Right (LSR) By 4 Position



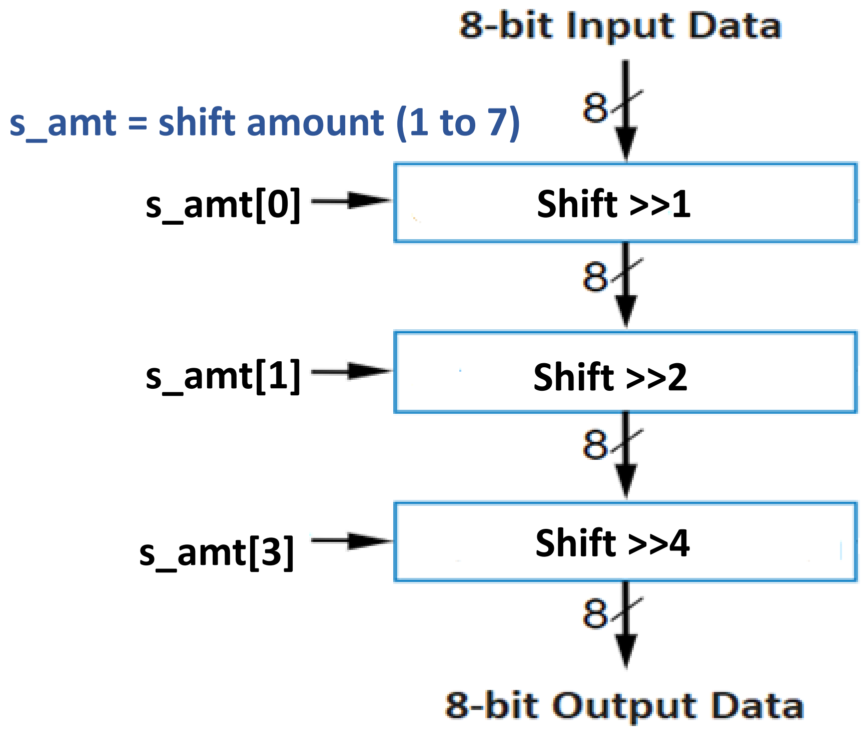
Right Rotate (ROR) by 4



LSR by ‘4’ bit position is same as dividing number by 16
In above mentioned example : 32’b1111_1111_1111_1111_1111_1111_1111_1111 decimal equivalent value is 4294967295
Performing LSR by ‘4’ on above mentioned value binary equivalent value is 32’b0000_1111_1111_1111_1111_1111_1111_111
which is decimal equivalent value 268435455 (4294967295 divide by 16 = 268435455)

8-bit Barrel Shifter

- **8-bit barrel shifter can be designed using 8x1 multiplexor but lot of wires**
  - More efficient design is two use cascaded 2x1 multiplexers
  - Number of 2x1 multiplexer required for N-bit input barrel shifter is : **$N \times \log_2 N$**
    - To implement 8-bit Barrel shifter, **24** 2x1 multiplexer are required (8 x $\log_2 8$ = 24)
  - Chain 3 rows of shifters and each shifter will either shift by 4, 2 or 1 bit positions respectively
  - Each shifter row has **8** 2x1 mux cascaded
  - Can achieve any amount of shift (1 to 7) by enabling correct combinations of these chained shifts. Example:
    - Shifting by 3-bit positions can be achieved by setting **s_amt[0]=1, s_amt[1]=1 and s_amt[2]=0**
    - Shifting by 7-bit positions can be achieved by setting **s_amt[0]=1, s_amt[1]=1 and s_amt[2]=1**



8-bit Input Data

s_amt = shift amount (1 to 7)

s_amt[0]

Shift >>1

s_amt[1]

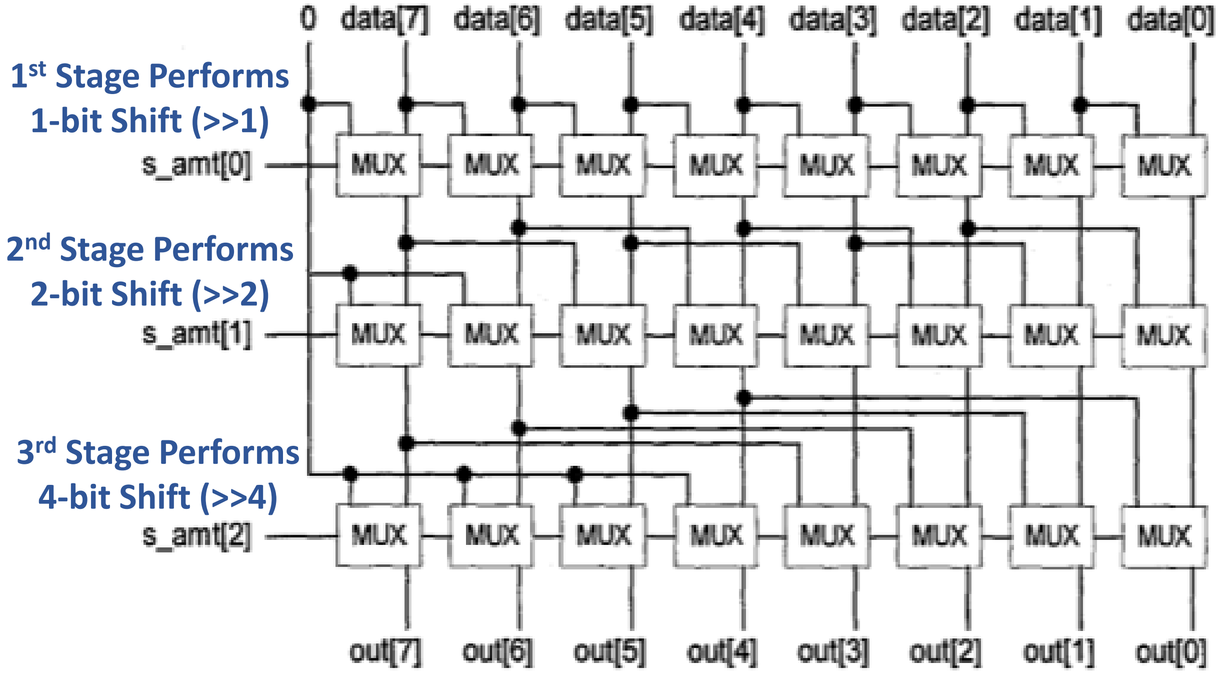
Shift >>2

s_amt[3]

Shift >>4

8-bit Output Data

Barrel Shifter using 2x1 MUX to perform logical shift right operation



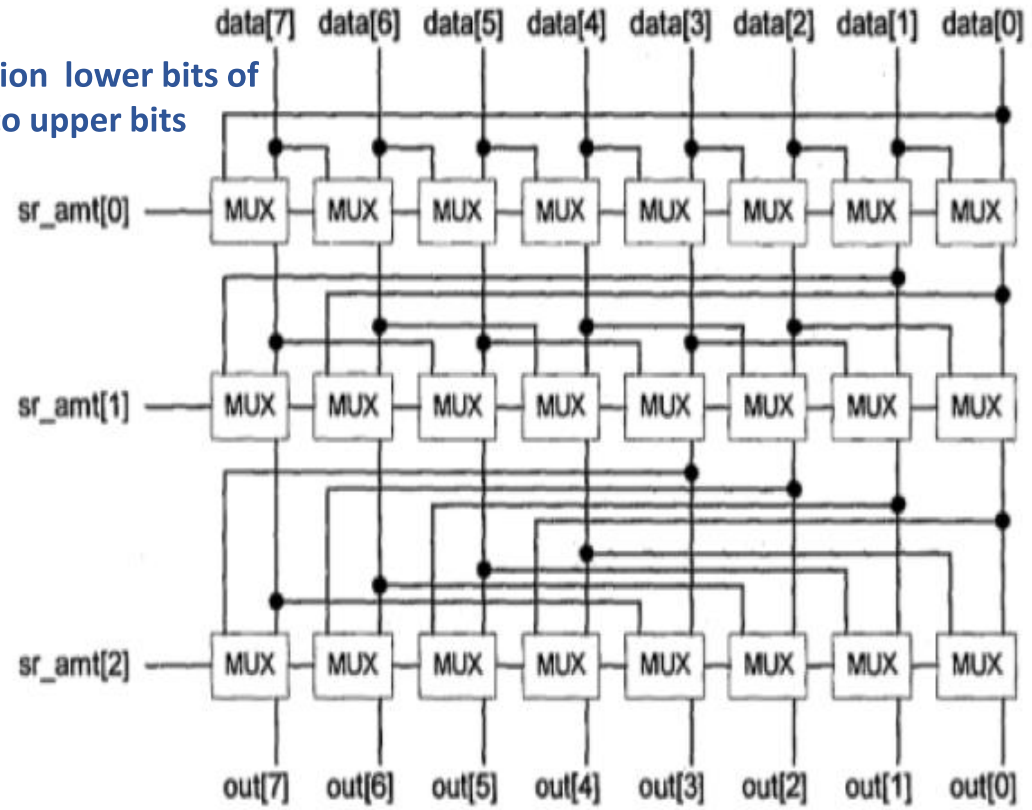
1st Stage Performs 1-bit Shift (>>1)

2nd Stage Performs 2-bit Shift (>>2)

3rd Stage Performs 4-bit Shift (>>4)

8-bit Barrel Shifter

- **A right rotator is very similar to a logical right shifter.**
  - Since all of the input bits are routed to the output, there is no longer a need for interconnect lines carrying the zero tied signal.
- **Interconnect lines need to be inserted to enable routing of the low order bits from each row to high order region so that rotate can occur**
  - Longer the interconnect lines of the rotator can increase both area and delay



For Rotate operation lower bits of input are copied to upper bits

1st Stage Performs Rotate by 1=bit

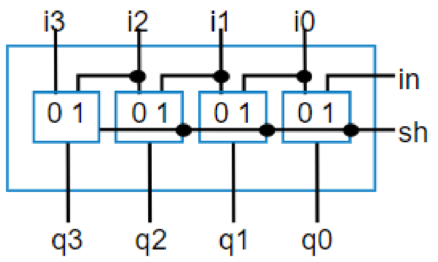
2nd Stage Performs Rotate by 2-bits

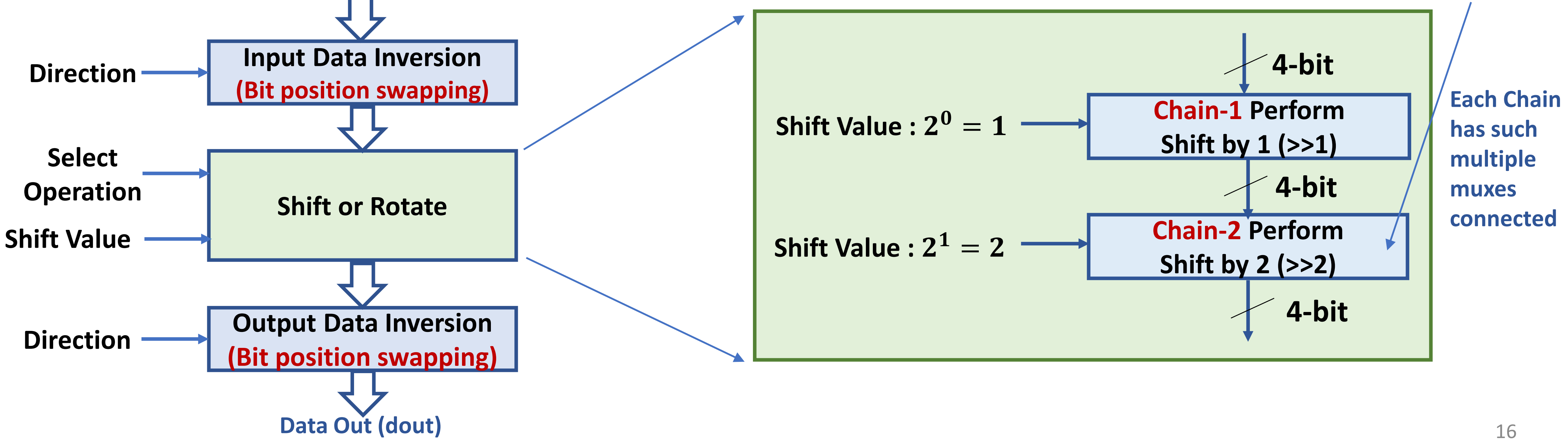
3rd Stage Performs Rotate by 3-bits

Barrel Shifter using 2x1 MUX to perform right rotate operation

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 x $\log_2 4$ = 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)**
- ❑ **Selection of shift vs rotate operation is done using a select bit**
- ❑ **Left shift/rotate operation is implemented through inversion of the input and output data**
  - Right shift/rotate operation does not require data bit inversion if muxes are wired for right shift/rotate





4-Bit Barrel Shifter Logical Right Shift and Rotate Right

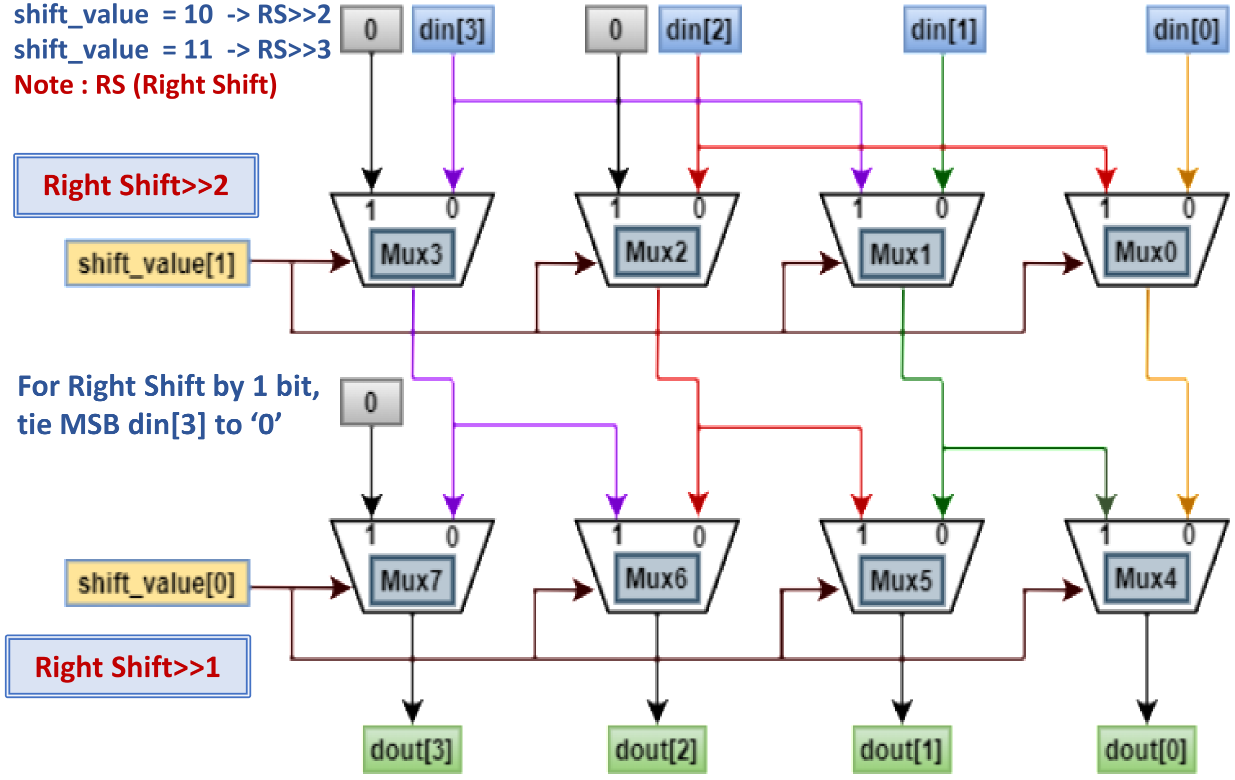
- ❑ For Logical Shift Right and Rotate Right operations, input and output data bit inversions is not required !

Barrel Shifter using 2x1 MUX to perform logical shift right operation

Shift Right by 1, 2, 3 bit position

shift_value = 00 -> No Shift
shift_value = 01 -> RS>>1
shift_value = 10 -> RS>>2
shift_value = 11 -> RS>>3
Note : RS (Right Shift)

For Right Shift by 2 bits, tie upper two din[3] and din[0] bit to '0'



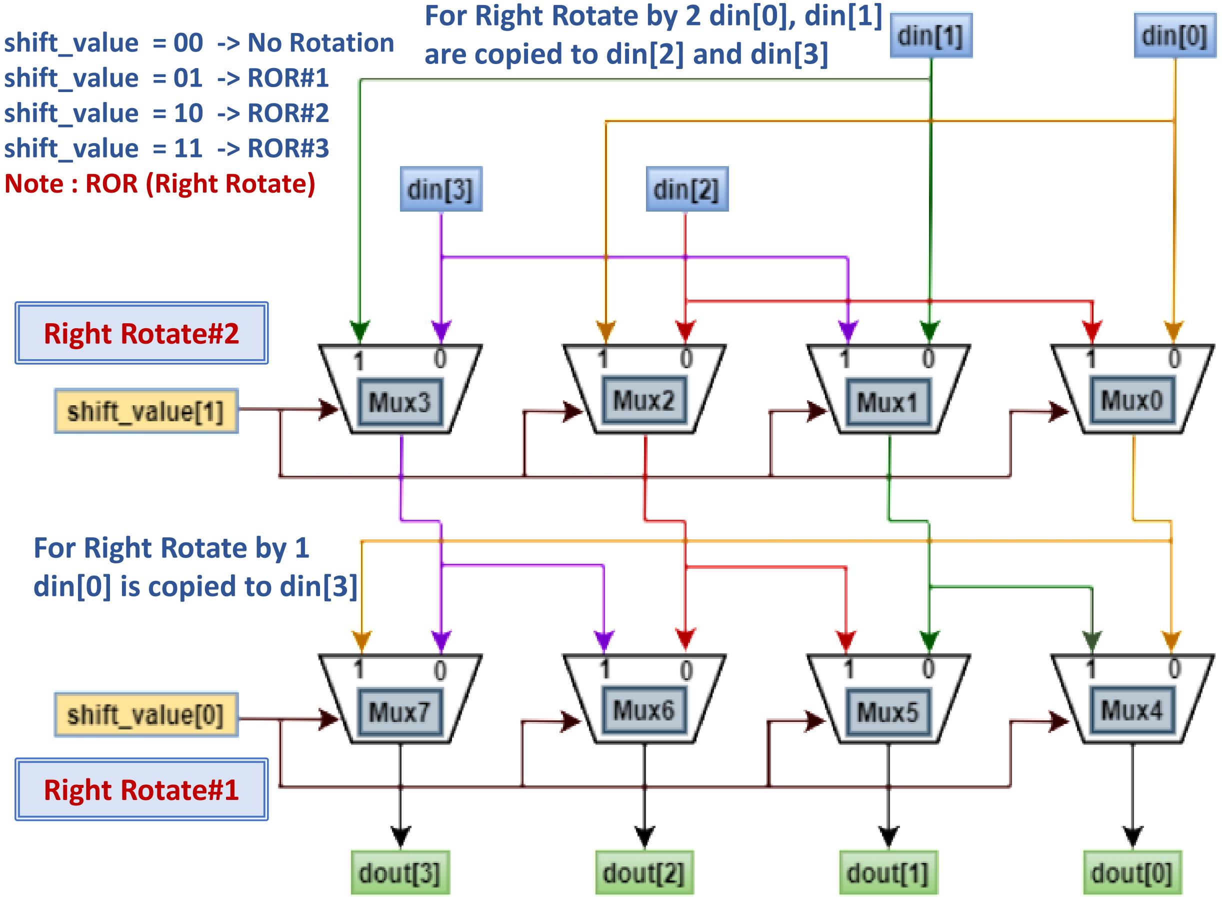
For Right Shift by 1 bit, tie MSB din[3] to '0'

Barrel Shifter using 2x1 MUX to perform rotate right operation

Rotate Right by 1, 2, 3 bit position

shift_value = 00 -> No Rotation
shift_value = 01 -> ROR#1
shift_value = 10 -> ROR#2
shift_value = 11 -> ROR#3
Note : ROR (Right Rotate)

For Right Rotate by 2 din[0], din[1] are copied to din[2] and din[3]



For Right Rotate by 1 din[0] is copied to din[3]

4-Bit Barrel Shifter Left Shift and Rotate Left

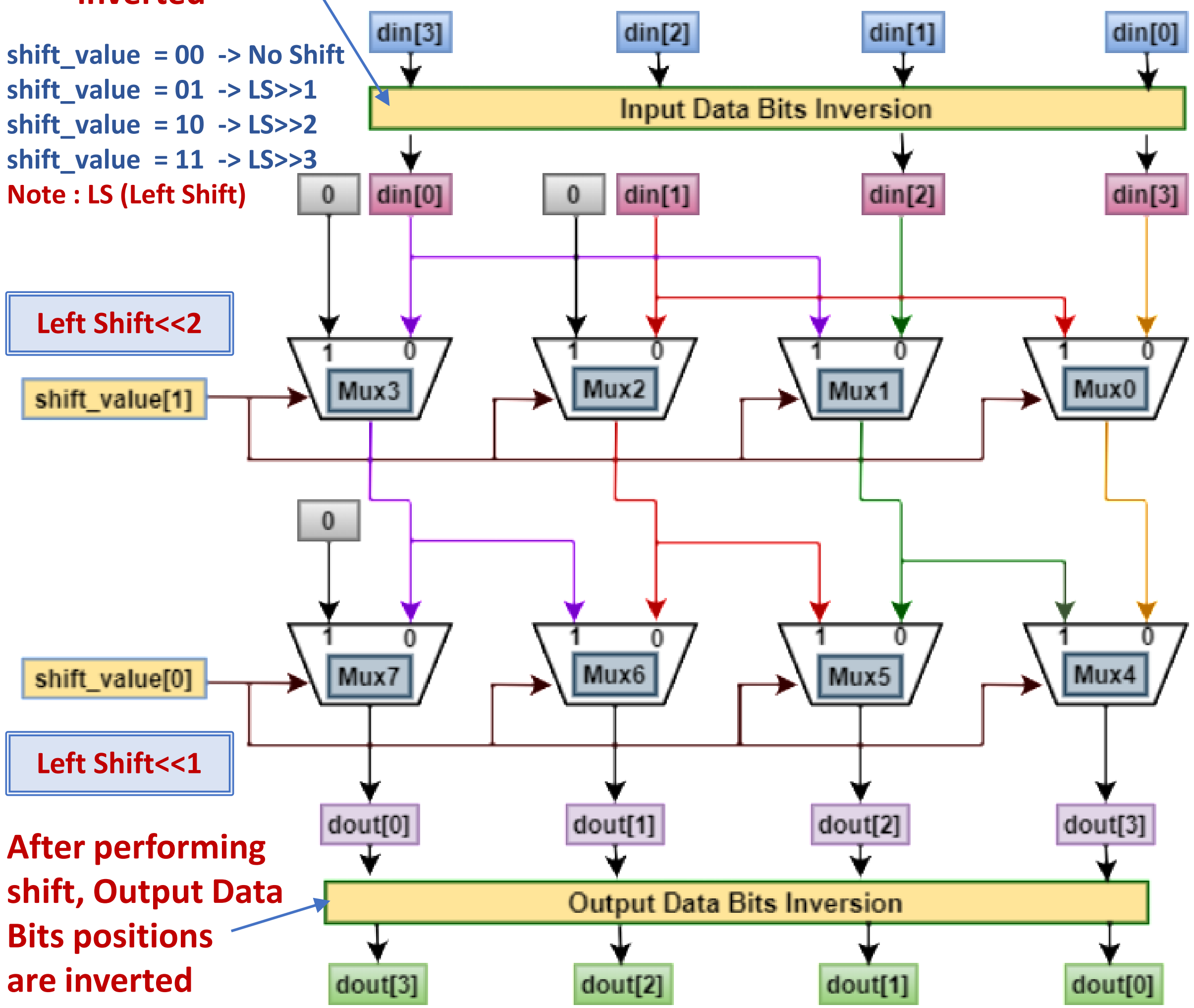
- ❑ For Logical Shift Left and Rotate Left operations, input and output data bit inversions is required !

Barrel Shifter using 2x1 MUX to perform logical shift left operation

Shift Left by 1, 2, 3 bit position

Before performing shift, Input Data Bits positions are inverted

shift_value = 00 -> No Shift
shift_value = 01 -> LS>>1
shift_value = 10 -> LS>>2
shift_value = 11 -> LS>>3
Note : LS (Left Shift)



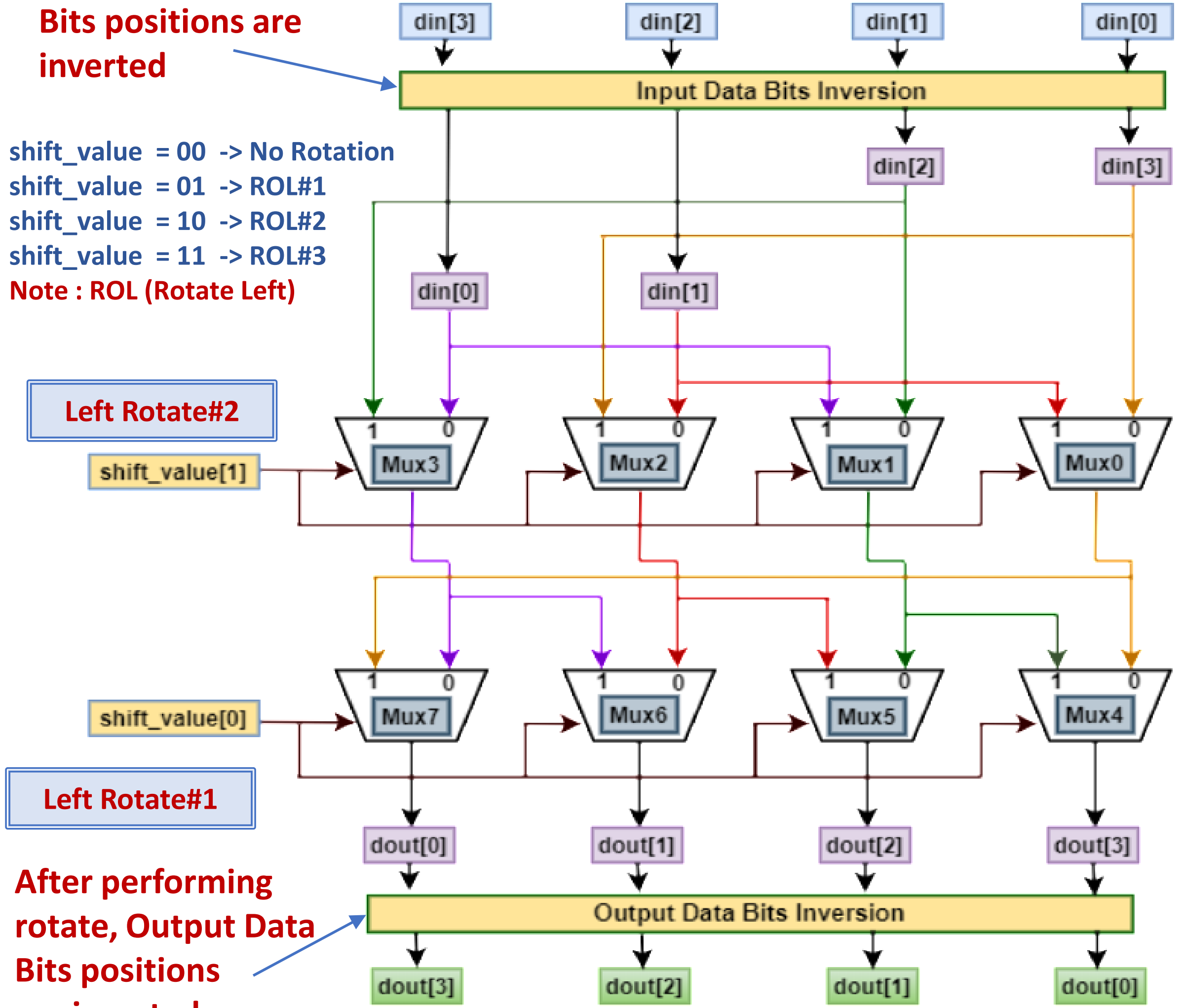
After performing shift, Output Data Bits positions are inverted

Barrel Shifter using 2x1 MUX to perform rotate left operation

Rotate left by 1, 2, 3 bit position

Before performing rotate, Input Data Bits positions are inverted

shift_value = 00 -> No Rotation
shift_value = 01 -> ROL#1
shift_value = 10 -> ROL#2
shift_value = 11 -> ROL#3
Note : ROL (Rotate Left)

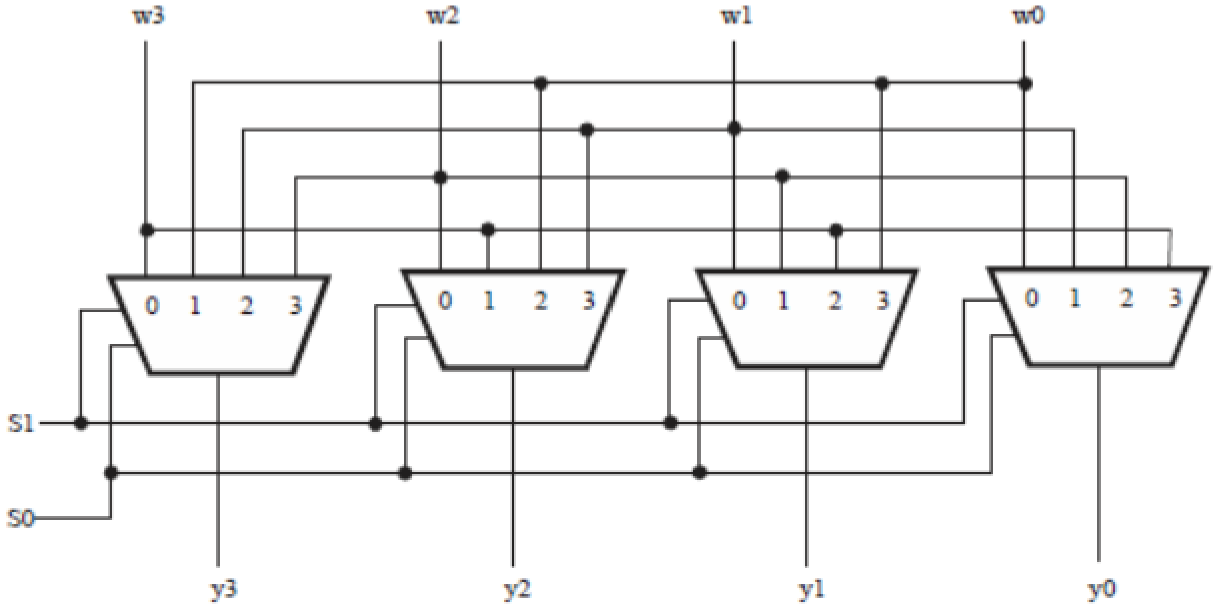


After performing rotate, Output Data Bits positions are inverted

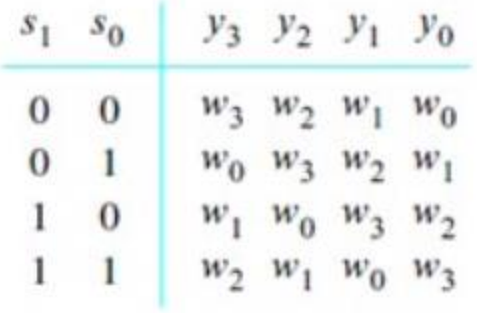
4-bit Barrel Shifter

- **Barrel Shifter can also be designed using 4x1 Multiplexers**
  - Truth Table and Circuit Diagram using 4x1 Multiplexer is shown below

Barrel Shifter using 4x1 MUX



Truth Table



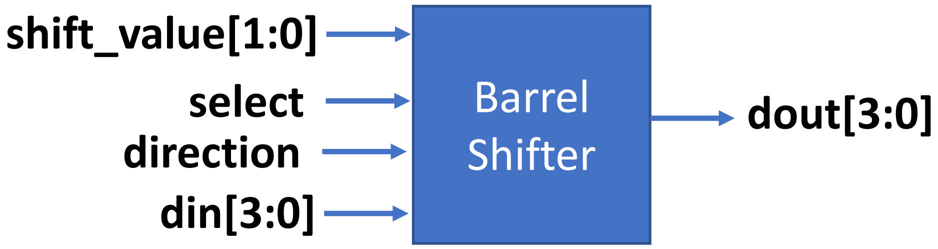
s_1	s_0	y_3	y_2	y_1	y_0
0	0	w_3	w_2	w_1	w_0
0	1	w_0	w_3	w_2	w_1
1	0	w_1	w_0	w_3	w_2
1	1	w_2	w_1	w_0	w_3

Note : For Barrel Shifter RTL Modeling homework, use 2x1 Mux instead of 4x1 Mux

Homework Assignment-4b : Barrel Shifter

❑ Develop SystemVerilog RTL model for 4-bit Barrel Shifter using 2x1 Mux behavioral model

- Same Barrel Shifter RTL model should support logical shift left, logical shift right, rotate left, rotate right 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**



shift_value[1:0]

select

direction

din[3:0]

Barrel Shifter

dout[3:0]

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

❑ 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

Homework Assignment-4b : Barrel Shifter

❑ **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 Assignment-4b : Barrel Shifter

❑ **Submit report which should include :**

- SystemVerilog design code
- Synthesis resource usage and schematic generated from RTL netlist viewer
- Simulation snapshot and explain simulation result to confirm RTL model developed works as a Barrel Shifter
- Post-Mapping schematic is optional to submit.
- Explanation of FPGA resource usage in report is not required.

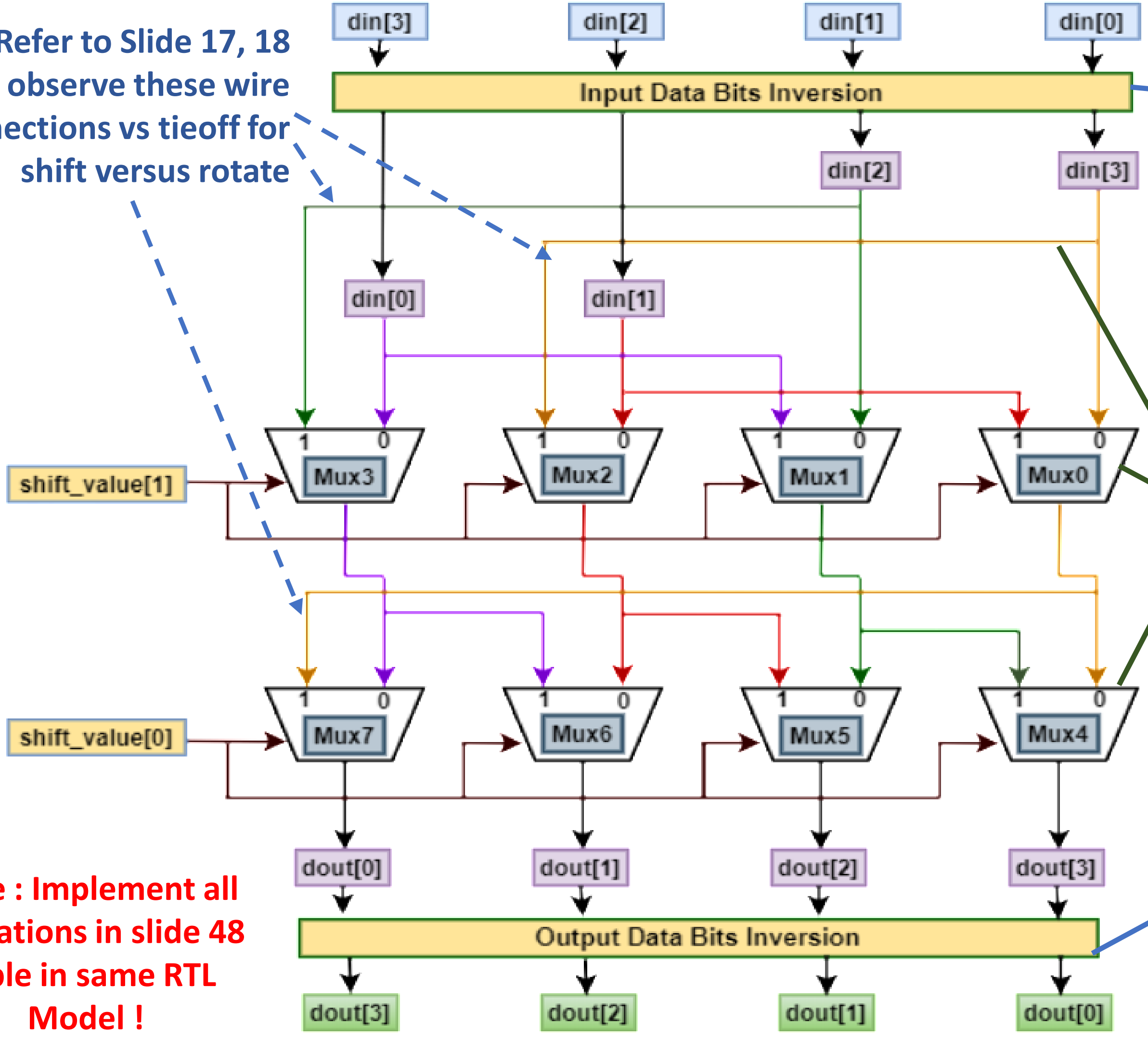
Note :

- Lab4 folder includes full testbench code and partial design template code for Barrel Shifter
- It is not mandatory to use design template code provided in lab folder. Student can implement their design module from scratch without referring to template code as long as primary port list is matching with in previous slide. This is to ensure testbench is compatible with design.
- For learning purpose, student can change the stimulus in initial block in testbench file.

Homework Assignment-4b : Barrel Shifter (Hints)

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

Refer to Slide 17, 18 and observe these wire connections vs tieoff for shift versus rotate



Have a separate always@ procedural block to store input din data to a local variable and do the swapping of input din bits based on direction 0 or 1 (0: right, 1: left)
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 17/18, 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 data out dout bits based on direction == 0 or 1
If direction == 0, then do not perform bit inversions
If direction == 1, then perform bit inversions

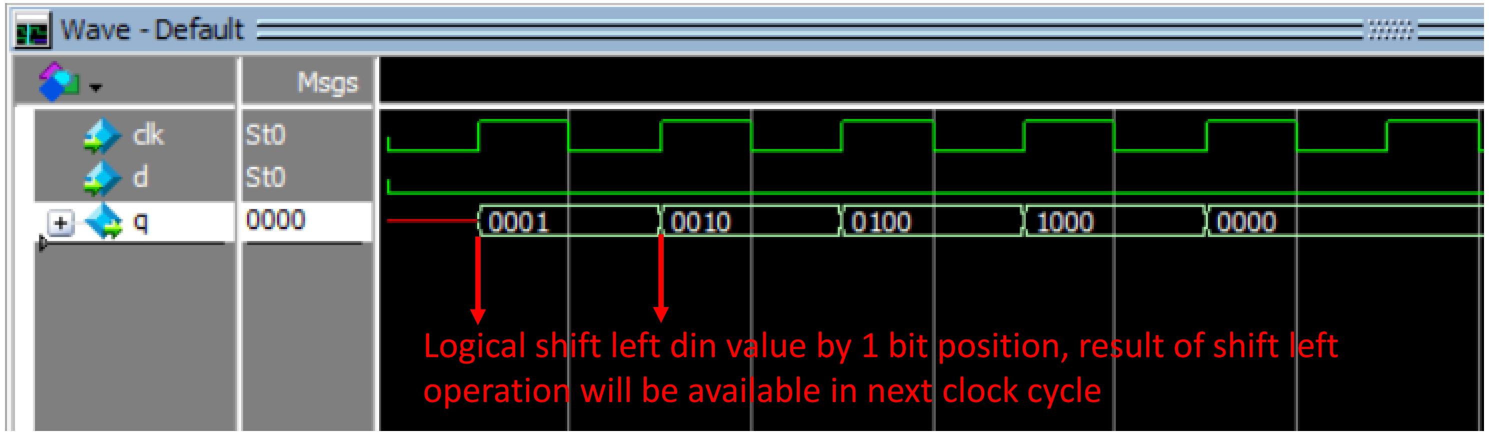
dout[0] = t_dout[3];
dout[1] = t_dout[2];
dout[2] = t_dout[1];
dout[3] = t_dout[0];

Note : Implement all operations in slide 48 table in same RTL Model !

Sequential Shift Register vs Barrel Shifter

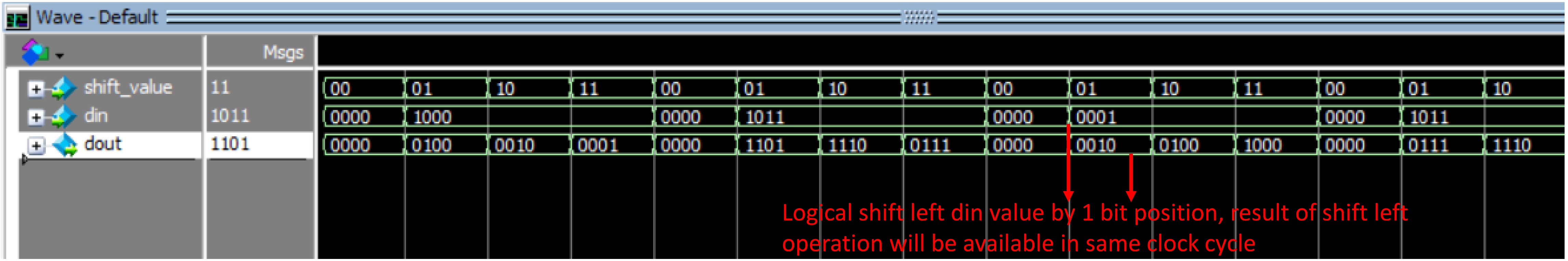
- ❑ **In Sequential** shift register result of shifting by ‘1’ bit position will be available in next clock cycle whereas result of Right Shift by ‘1’ using Barrel shifter will be available in same clock cycle

LSL<<1 using Sequential Shift Register



Logical shift left din value by 1 bit position, result of shift left operation will be available in next clock cycle

LSL<<1 using Barrel Shifter

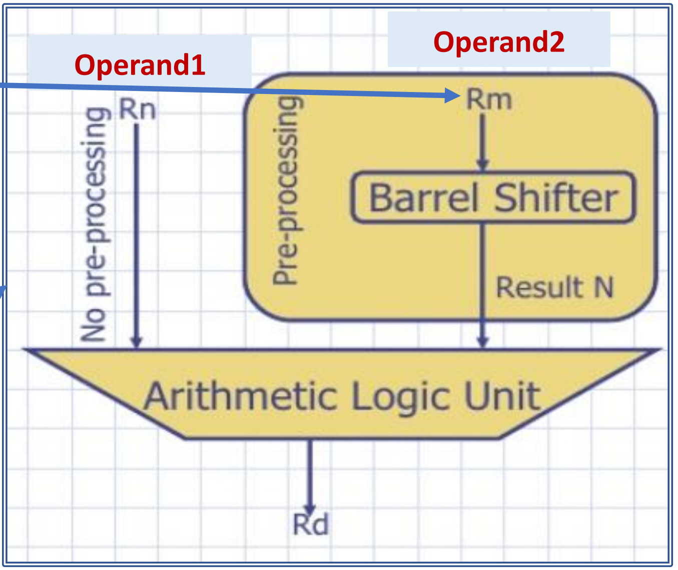


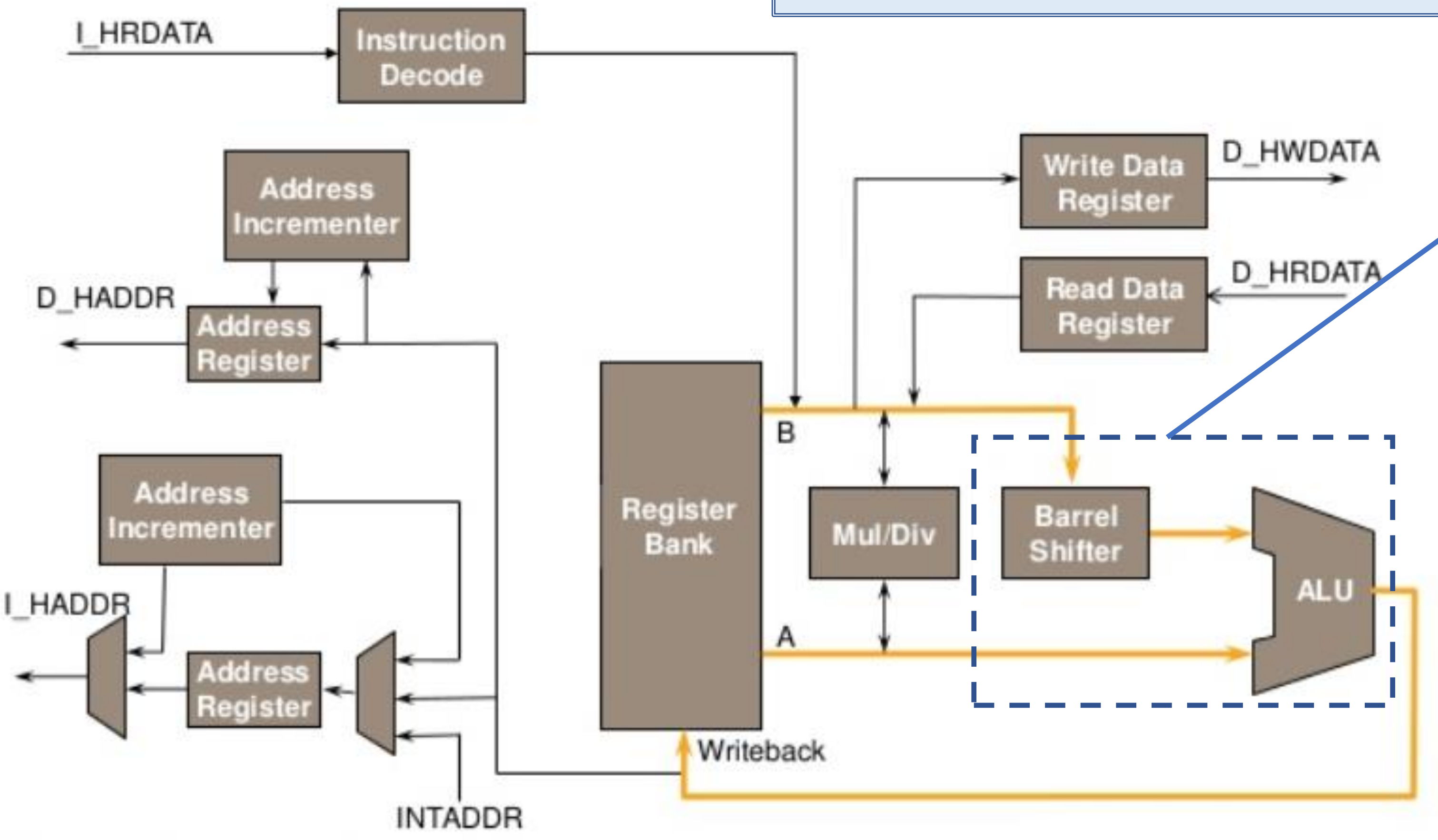
Logical shift left din value by 1 bit position, result of shift left operation will be available in same clock cycle

Application of Barrel Shifter

- ❑ **Barrel Shifter is used in most modern CPU's to pre-process instruction operand before execution of the instruction**

Left or right shift/rotate the 32 bit binary pattern in one of the source registers (Rm) by a specific number of positions before it enters the ALU





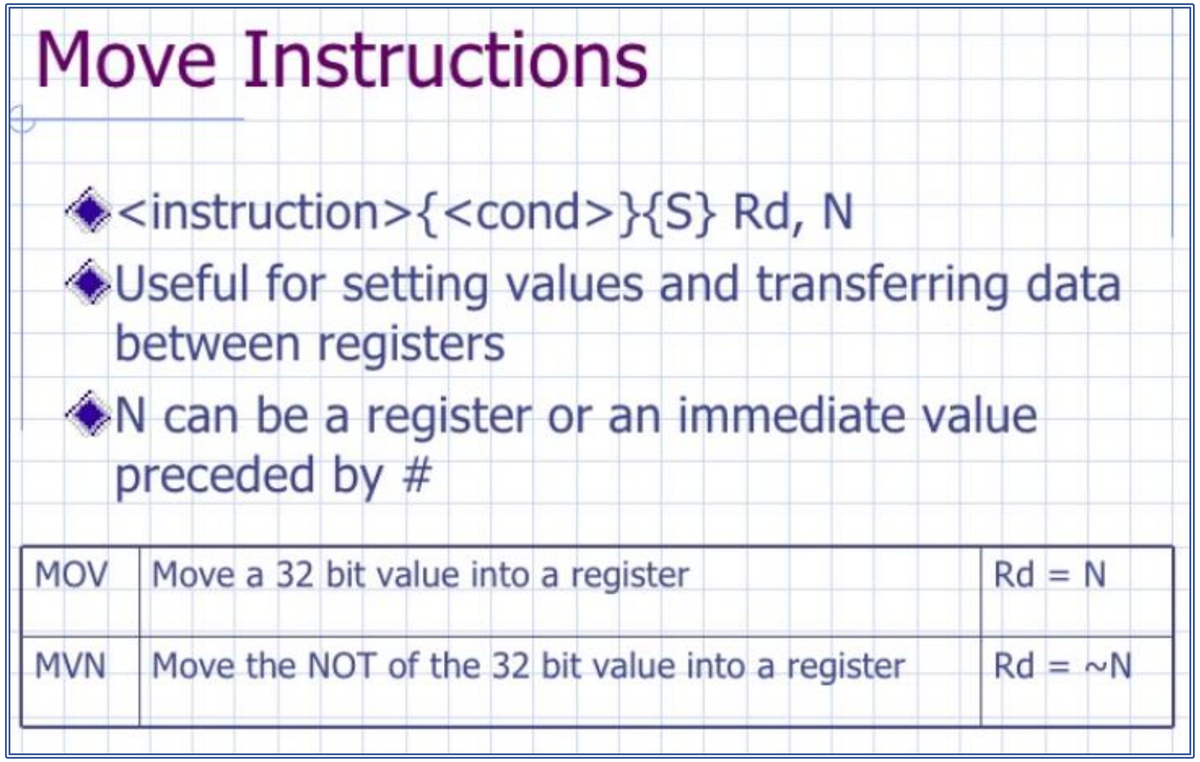
Barrel Shifter usage in ARM Cortex-M3 CPU Datapath

Example

- ❑ **MOV r0, r0, LSL #1**
  - Multiply **r0** by 2
- ❑ **MOV r1, r1, LSR #2**
  - Divide **r1** by 4
- ❑ **MOV r3, r3, ROR #16**
  - Swap the top and bottom halves of **r3**.
- ❑ **ADD r5, r4, r4, LSL #4**
  - Multiply **r4** by 16 and then add with **r4**

Application of Barrel Shifter

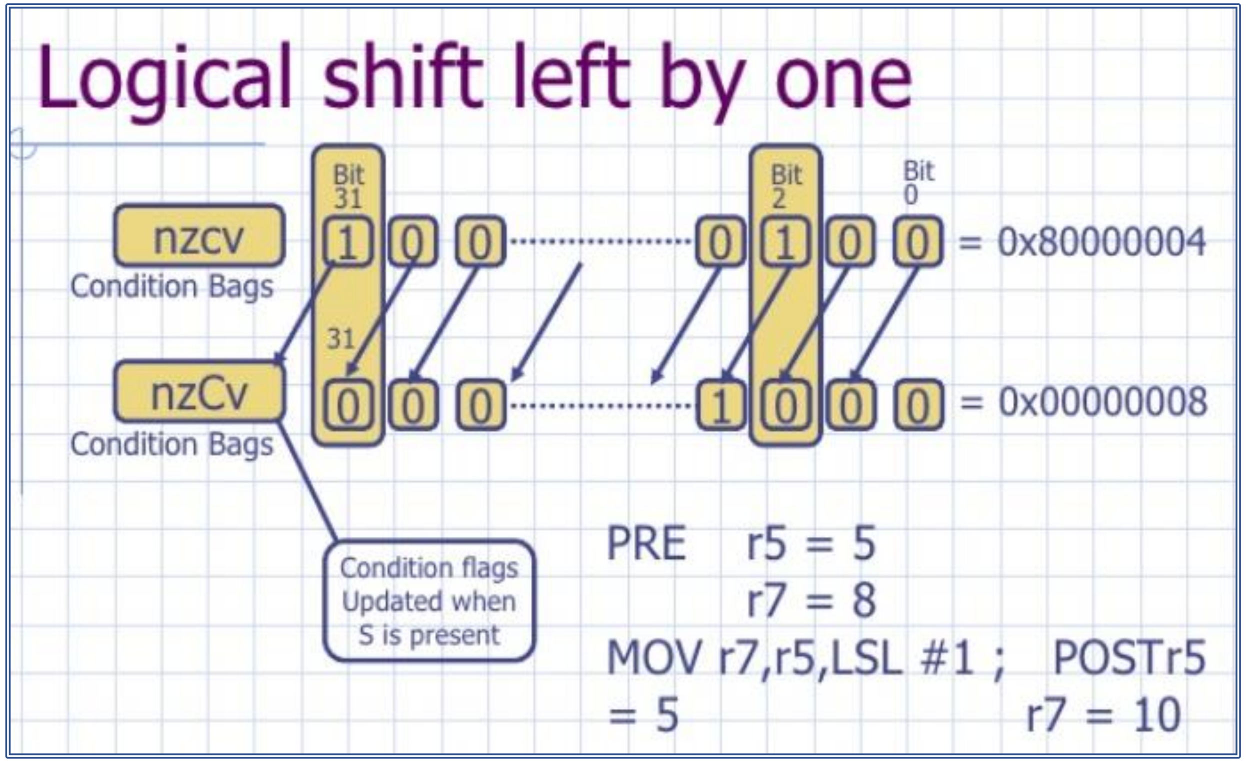
- ❑ **Example of “move” instructions execution with pre-processing of a operand using Barrel Shifter**



Move Instructions

- <instruction>{<cond>}{S} Rd, N
- Useful for setting values and transferring data between registers
- N can be a register or an immediate value preceded by #

MOV	Move a 32 bit value into a register	Rd = N
MVN	Move the NOT of the 32 bit value into a register	Rd = ~N



Logical shift left by one

nzcv
Condition Bags
Bit 31
Bit 2
Bit 0
1 0 0 .... 0 1 0 0 = 0x80000004
31
nzCv
Condition Bags
0 0 0 .... 1 0 0 0 = 0x00000008
Condition flags Updated when S is present

```
PRE    r5 = 5
       r7 = 8
MOV r7,r5,LSL #1 ;   POSTr5
= 5                  r7 = 10
```