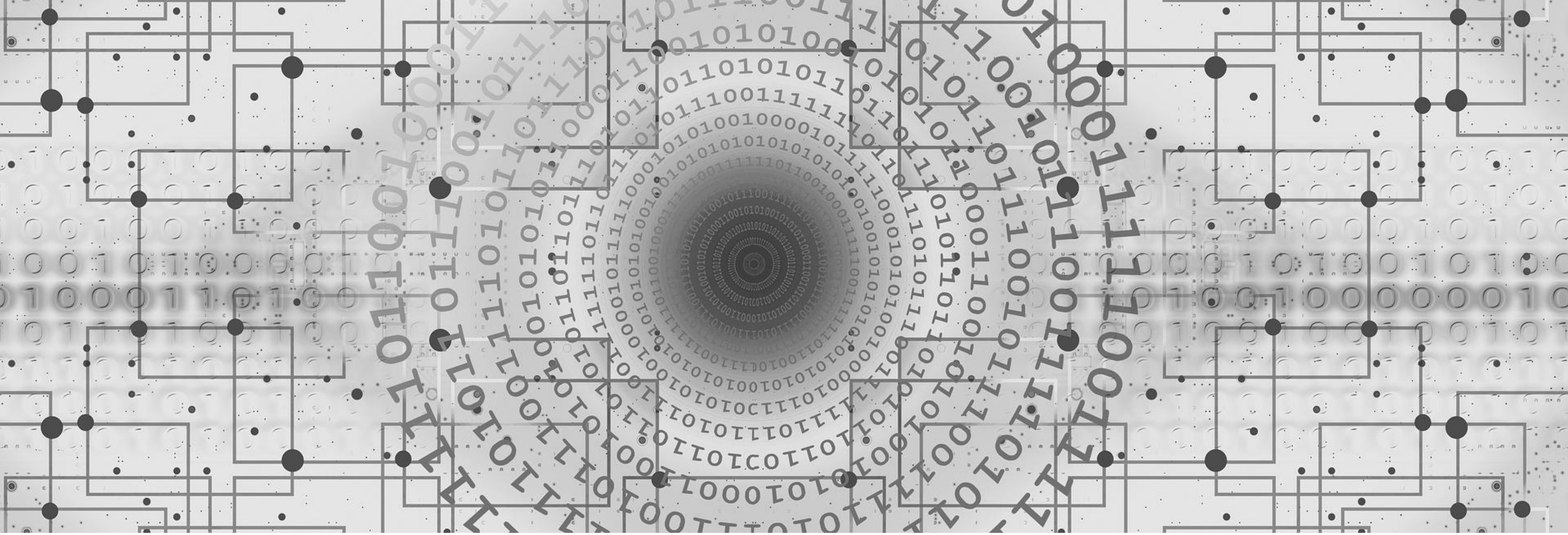


Homework-3 : Johnson Counter & Universal Shift Register

ECE-111 Advanced Digital Design Project

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

Outline

- **Learn concepts of Shift Register**
  - What is a Shift Register
  - Operation of a Shift Register
  - Types of Shift Register
  - What is Universal Shift Register

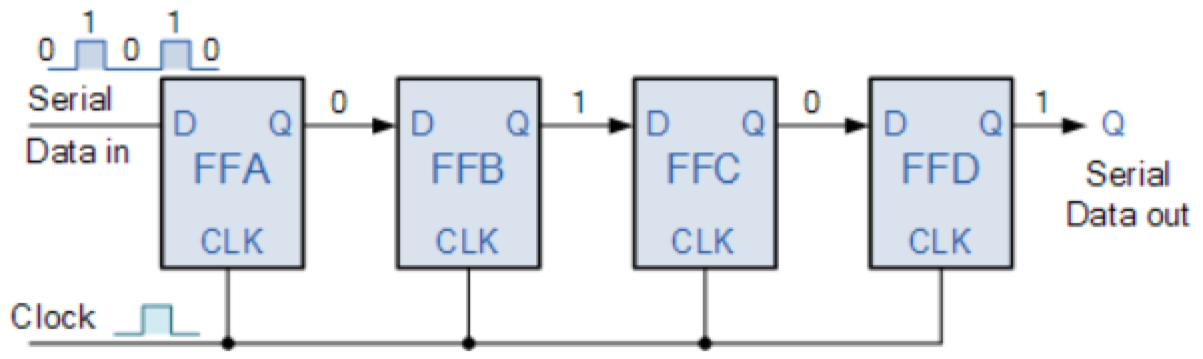
- **Application of Shift Register : Ring Counter and Johnson Counter**
  - Overview of Ring Counter
  - SystemVerilog Implementation of Ring Counter using non-blocking and blocking assignment statement
  - Simulation Results of Ring Counter
  - Overview of Johnson Counter

- **Homework-3a : Johnson Counter Requirements**
- **Homework-3b : Universal Shift Register Requirements**

Shift Register

❑ What is a Shift Register

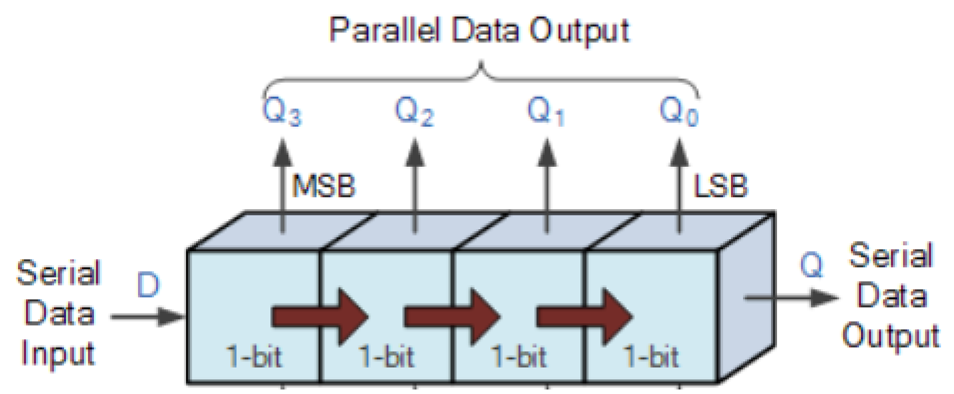
- A register capable of shifting its binary information either from right to left or left to right is called a shift register.



- As shift register typically consists of several single bit "D-Type Flip-Flop", connected together in a serial type daisy-chain arrangement so that the output from one Flip-Flop becomes the input of the next Flip-Flop and so on.

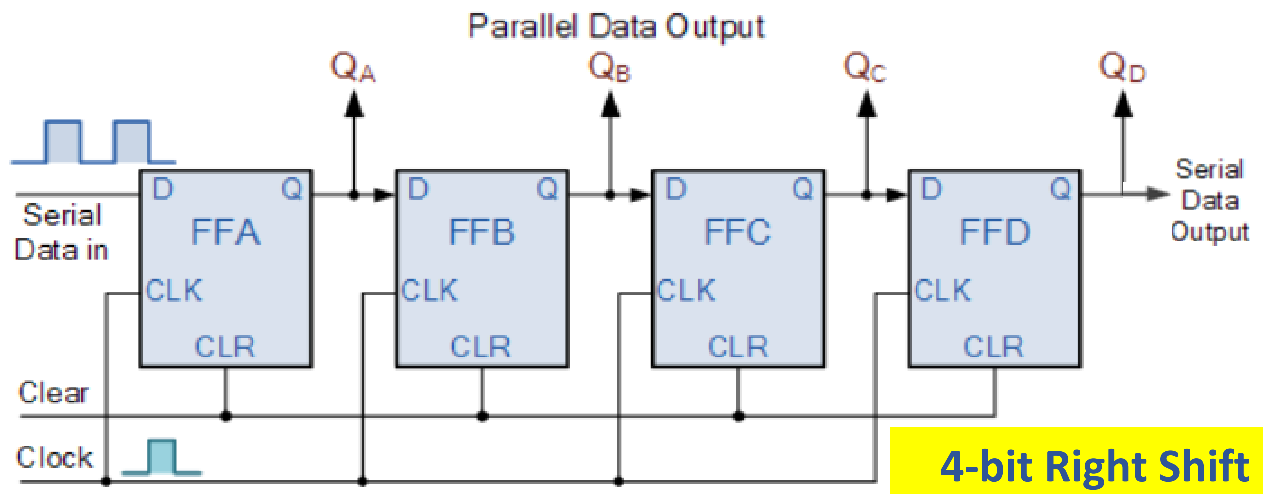
- Shift Register typically is used for data storage or for movement of data and are therefore commonly used inside calculators or computers :
  - To store data such as two binary numbers before they are added together, or
  - To convert the data from either a serial to parallel or parallel to serial format

Operation of a 4-bit Shift Register

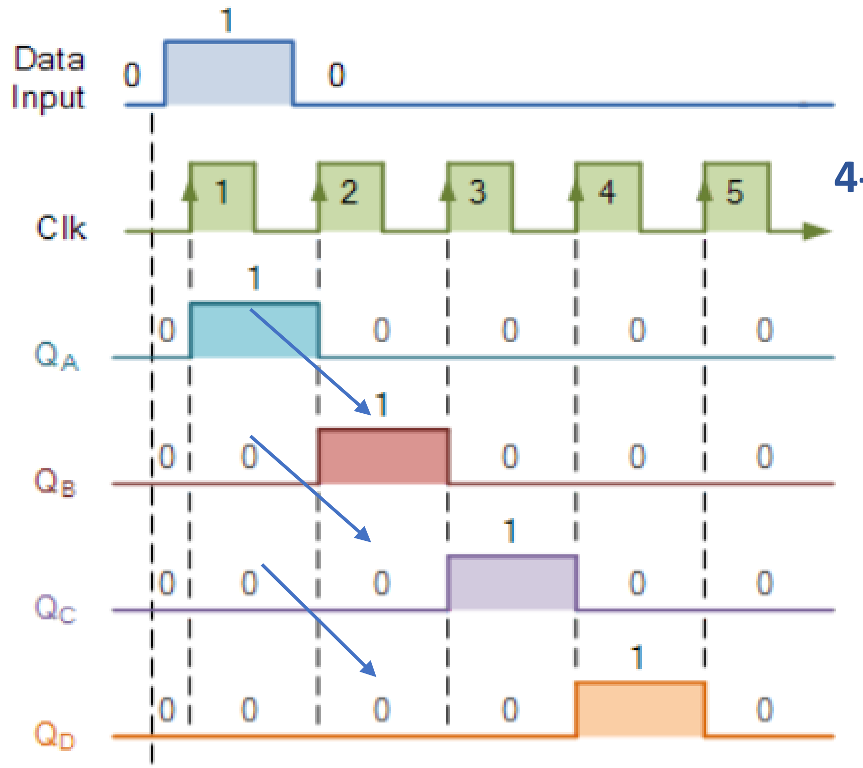


Serial Data In Serial Data Out
(4-bit Right Shift Register)

Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0



4-bit Right Shift Register



Stored Word in 4-bit Shift Register

1000

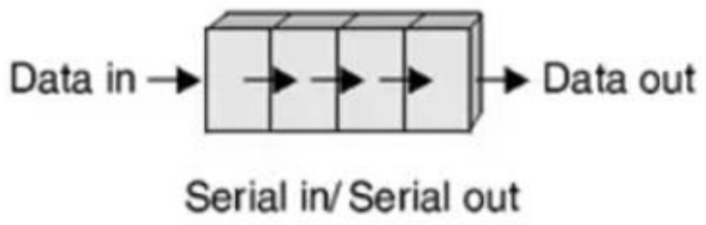
0100

0010

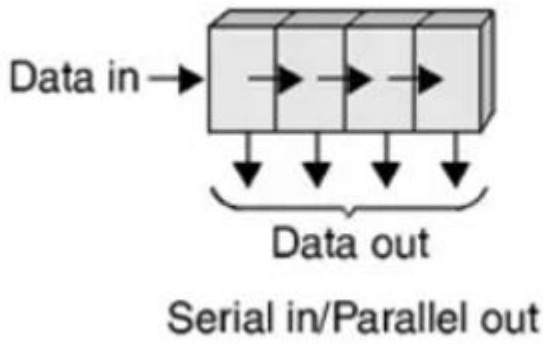
0001

Types of Shift Register

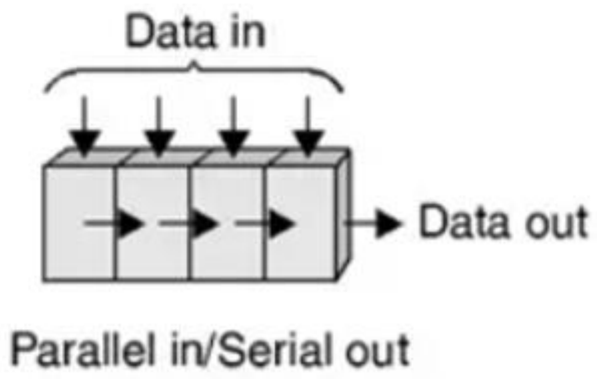
- **Shift Registers are classified under four types**
  - **Serial In / Serial Out (SISO)** : One bit of data can be moved in and out of the register serially at a time
    - serial data can be shifted in and shifted out using either Right Shift or Left Shift operation (**SISO-L** and **SISO-R**)
  - **Serial In / Parallel Out (SIPO)** : One bit of can be loaded serially but the stored data can be taken out of the register parallelly
    - serial data can be shifted in using either Right Shift or Left Shift operation (**SIPO-L** and **SIPO-R**)
  - **Parallel In / Serial Out (PISO)** : The data can be loaded simultaneously but can be removed only as one bit at a time during each clock pulse
    - serial data can be shifted out using either Right Shift or Left Shift operation (**PISO-L** and **PISO-R**)
  - **Parallel In / Parallel Out (PIPO)** : The data can be loaded and read out into the register simultaneously



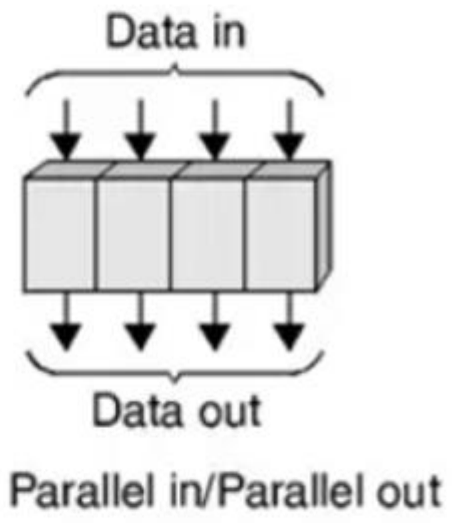
SISO with Right Shift



SIPO with Right Shift



PISO with Right Shift



PIPO

Universal Shift Register

- **What is a Universal Shift Register**
  - A Universal shift register is a register which has both the **right shift** and **left shift** with both **serial** and **parallel** load capabilities
  - It is a **Bi-directional** Shift Register since it is capable of shifting data in both left and right direction
  - Universal shift registers are used as memory elements in computers.
  - It can perform either of the following mentioned shift operation :
    - **PIPO :** Parallel In Parallel Out
    - **SIPO-L :** Left Shift Serial In Parallel Out
    - **SIPO-R :** Right Shift Serial In Parallel Out
    - **PISO-L :** Parallel In Left Shift Serial Out
    - **PISO-R :** Parallel In Right Shift Serial Out
    - **SISO-L :** Left Shift Serial In Left Shift Serial Out
    - **SISO-R :** Right Shift Serial In Right Shift Serial

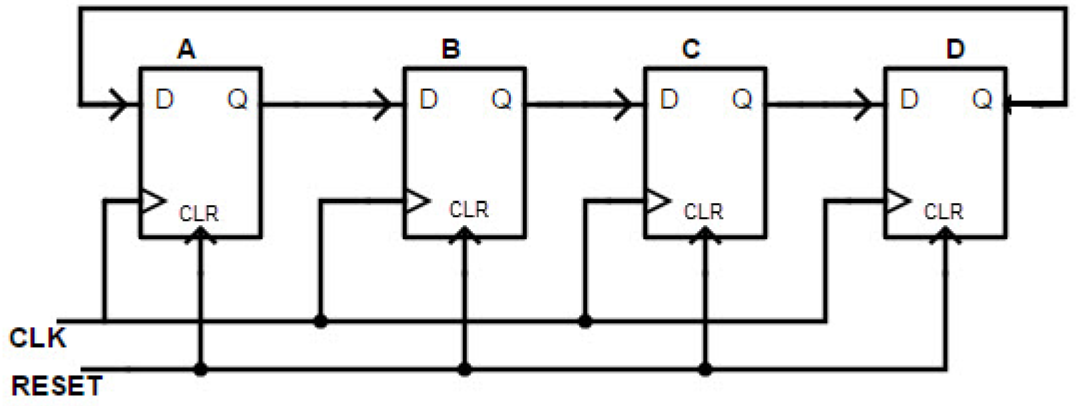
Universal Shift Register

- **PISO :** When performing **PISO-R** and **PISO-L** operation, once parallel value is loaded in shift register, and as serial data out shifting starts, then end of each clock cycle, value 0 is entered in vacated register bit position.
  - **Example :** 4-bit **PISO-R** operation, say starting value is **4'b1100** loaded in shift register
    - Clock-1 : 1100 // Initial value loaded in shift register
    - Clock-2 : 0110 // Upon Right Shifting, shift register bit position '3', value 0 is entered
    - Clock-3 : 0011 // Upon Right Shifting, shift register bit position '2', value 0 is entered
    - Clock-4 : 0001 // Upon Right Shifting, shift register bit position '1', value 0 is entered
    - Clock-5 : 0000 // Upon Right Shifting, shift register bit position '0', value 0 is entered
  - **Example :** 4-bit **PISO-L** operation, say starting value is **4'b1111** loaded in shift register
    - Clock-1 : 1111 // Initial value loaded in shift register
    - Clock-2 : 1110 // Upon Left Shifting, shift register bit position '0', value 0 is entered
    - Clock-3 : 1100 // Upon Left Shifting, shift register bit position '1', value 0 is entered
    - Clock-4 : 1000 // Upon Left Shifting, shift register bit position '2', value 0 is entered
    - Clock-5 : 0000 // Upon Left Shifting, shift register bit position '3', value 0 is entered

Application of Shift Register : Ring Counter

❑ What is a Ring Counter

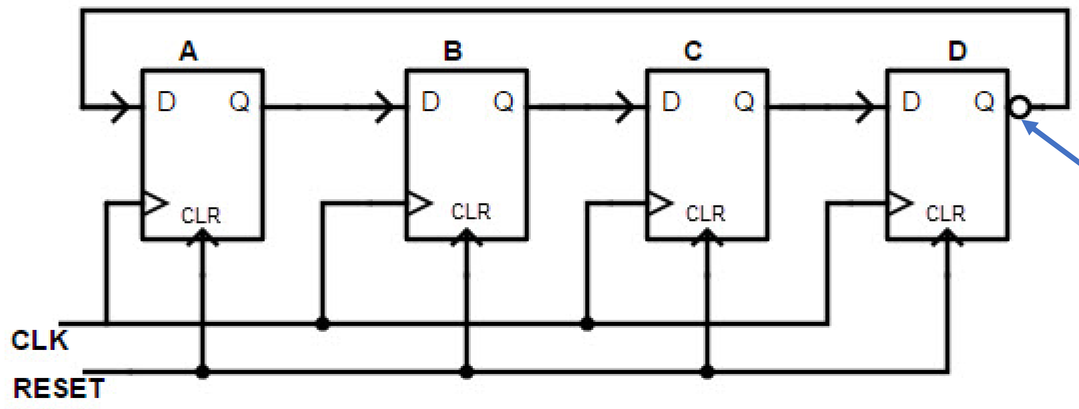
- Ring counter is an application of a Shift Register.
- A N-bit **ring counter** is composed of N-bit shift register, with the output of the last flip-flop fed to the input of the first flip-flop, making a "circular" or "ring" structure.



- There are two types of ring counters:
  - o **Straight ring counter** : Also known as a **one-hot counter (only 1-bit changes at a time in a count value)**, connects the output of the last shift register to the first shift register input and circulates a single one (or zero) bit around the ring.
    - o N-bit straight ring counter has N-states and N number of flipflops

Application of Shift Register : Johnson Counter

- **What is a Johnson Counter**
  - It is another type of ring counter also known as **twisted ring counter**
  - In Johnson Counter, complement of the output of the last stage register (i.e. flipflop) is fed back to the input of the first register (i.e. flipflop)
    - It circulates a stream of ones followed by zeros around the ring.
    - **N-bit** Johnson counter has **2N-**states and it requires **N** number of flipflops



Output of last flipflop is Inverted before feeding back to First Flip-Flop

Ring Counter vs Binary Counter

- **Ring Counter vs Binary Counter**
  - A **ring counter** of N-bits has only N valid states instead of 2^N. This makes them inefficient in terms of state usage and hardware resources.
  - A **binary counter** of N-bit has 2^N states.
  - Binary counter requires adder cricuit which is more complex than ring counter
  - Binary counter has higher propagation delay as the number of bits increases, whereas the propagation delay of a ring counter will be nearly constant regardless of the number of bits in the code

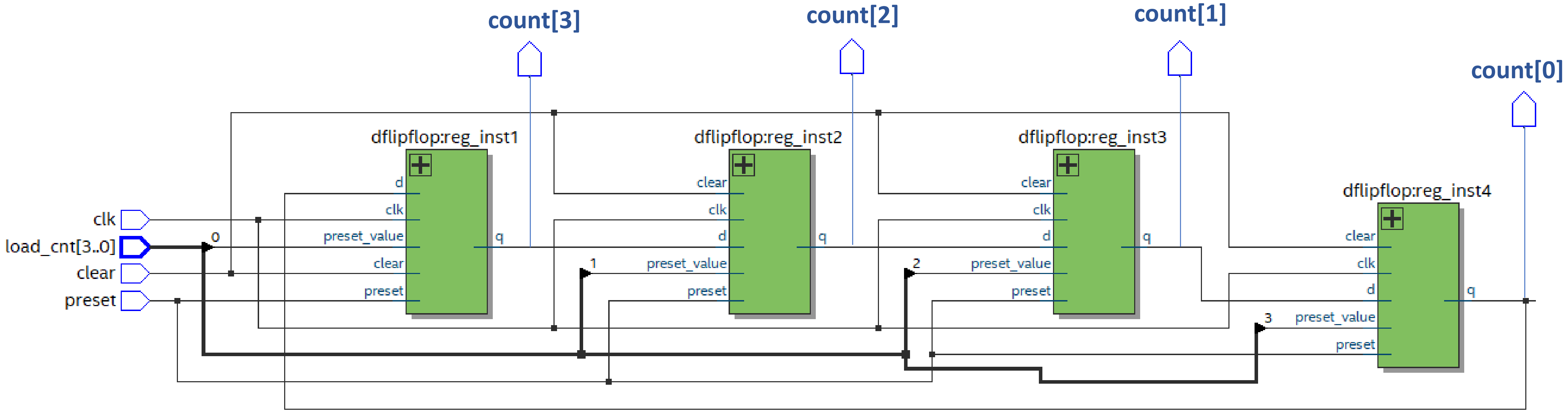
4-bit Ring Counter Truth Table and Block Diagram

4-bit Ring Counter with initial load count value = 4’b1000

N-bit Ring Counter has N states

Straight Ring Counter					
Clock Cycle	State	count[3]	count[2]	count[1]	count[0]
1	0	1	0	0	0
2	1	0	1	0	0
3	2	0	0	1	0
4	3	0	0	0	1
5	0	1	0	0	0
6	1	0	1	0	0
7	2	0	0	1	0
8	3	0	0	0	1
9	0	1	0	0	0

4-bit pattern repeats every 4 cycles



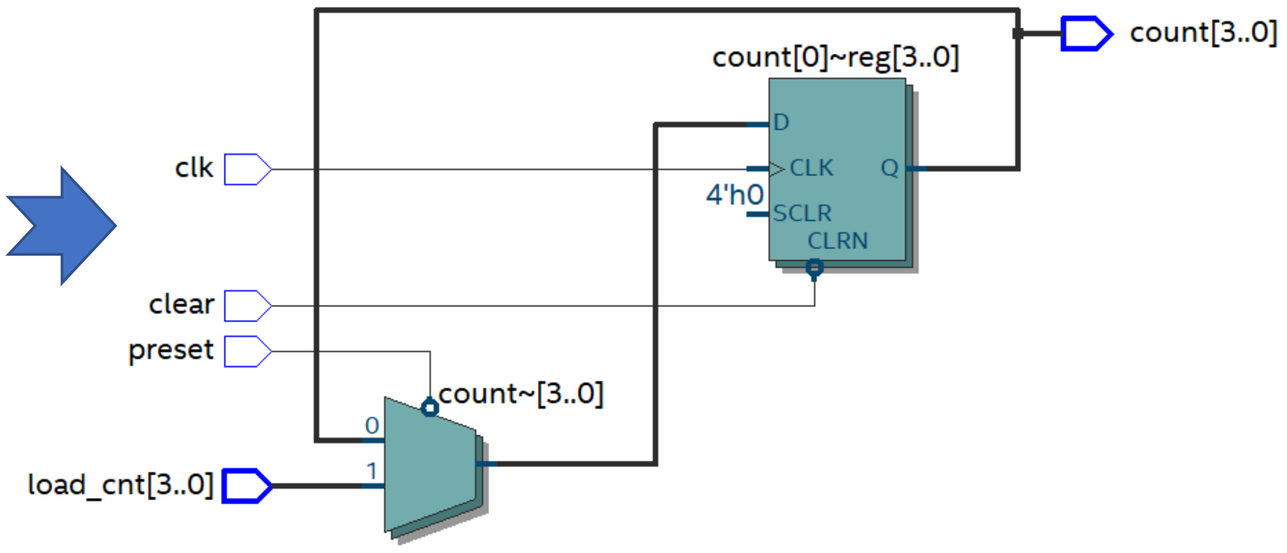
4-bit Ring Counter Using Non-Blocking Assignment

```
module ring_counter (
  input logic clk, clear, preset,
  input logic[3:0] load_cnt,
  output logic[3:0] count
);
 always@(posedge clk or negedge clear) begin
   if(!clear) count <= 4'b0000;
   else if(!preset) count <= load_cnt;
   else begin
     count <= count >> 1;
     count[3] <= count[0];
   end
  end
endmodule: ring_counter
```

Each clock cycle count value is right shifted and Count vector LSB which is output of last flipflop in shift register assigned to input of first fliflop

Can also be specified as

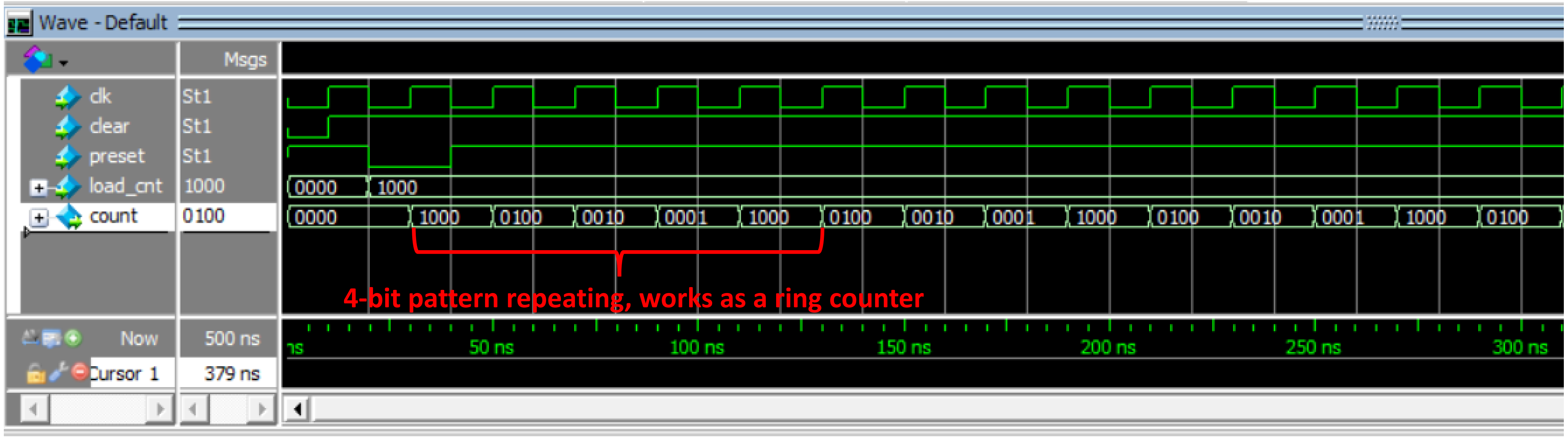
```
count[3] <= count[0];
count[2] <= count[3];
count[1] <= count[2];
count[0] <= count[1];
```



Both in Simulation and Synthesis it works as a Ring Counter

4-bit Ring Counter Using Non-Blocking Assignment

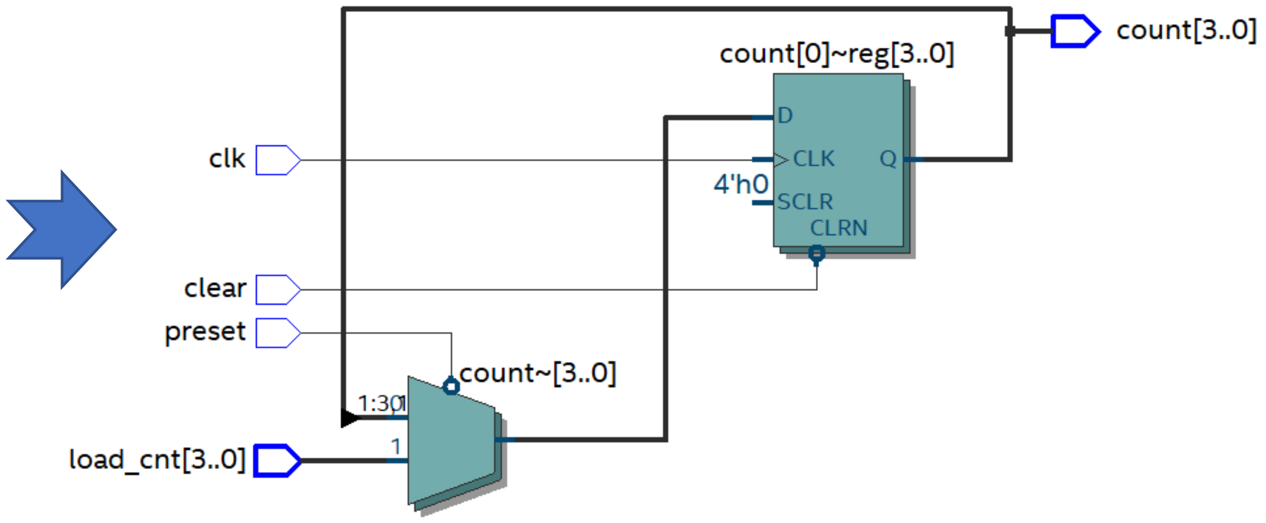
Simulation Result



4-bit Ring Counter Using Blocking Assignment

```
module incorrect_ring_counter (
  input logic clk, clear, preset,
  input logic[3:0] load_cnt,
  output logic[3:0] count
);
 always@(posedge clk or negedge clear) begin
   if(!clear) count = 4'b0000;
   else if(!preset) count = load_cnt;
   else begin
     count = count >> 1;
     count[3] = count[0];
   end
  end
endmodule: ring_counter
```

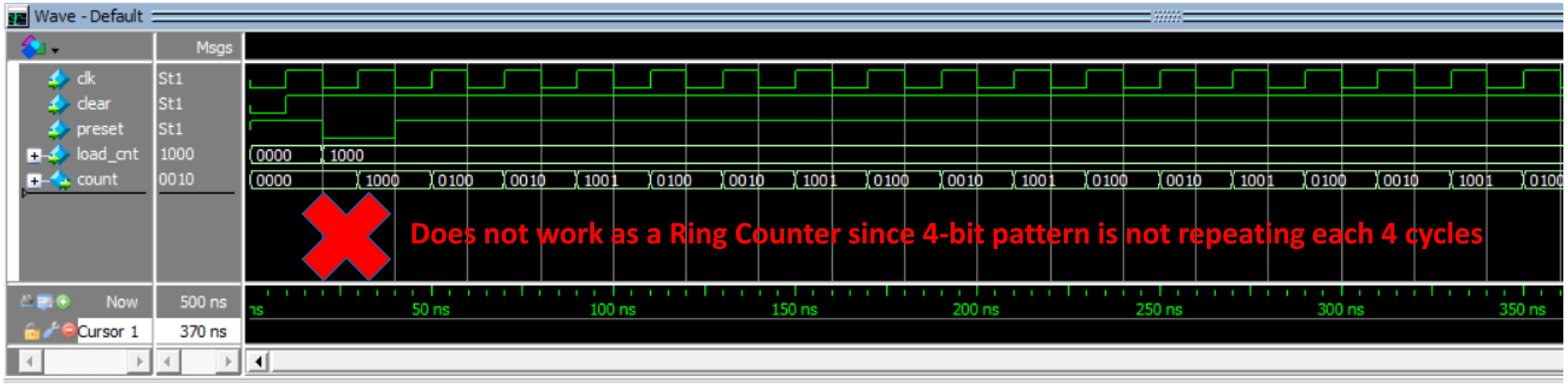
Replacing Non-blocking assignment with blocking assignment, RTL code does not work as a Ring counter



Simulation and Synthesis mis-match if non-blocking assignment used !!

4-bit Ring Counter Using Blocking Assignment

Simulation Result

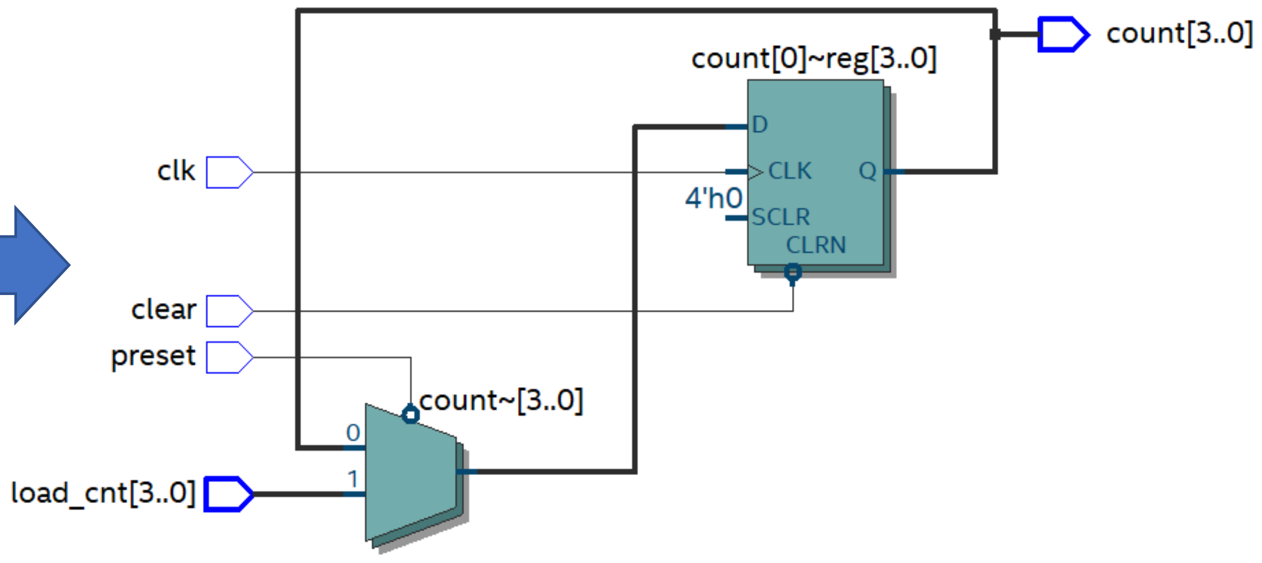


Does not work as a Ring Counter since 4-bit pattern is not repeating each 4 cycles

4-bit Ring Counter Using Blocking Assignment : Alternate Implementation

```
module ring_counter (
  input logic clk, clear, preset,
  input logic[3:0] load_cnt,
  output logic[3:0] count
);
 always@(posedge clk or negedge clear) begin
   if(!clear) count = 4'b0000;
   else if(!preset) count = load_cnt;
   else begin
     count = {count[0], count[3:1]};
   end
  end
endmodule: ring_counter
```

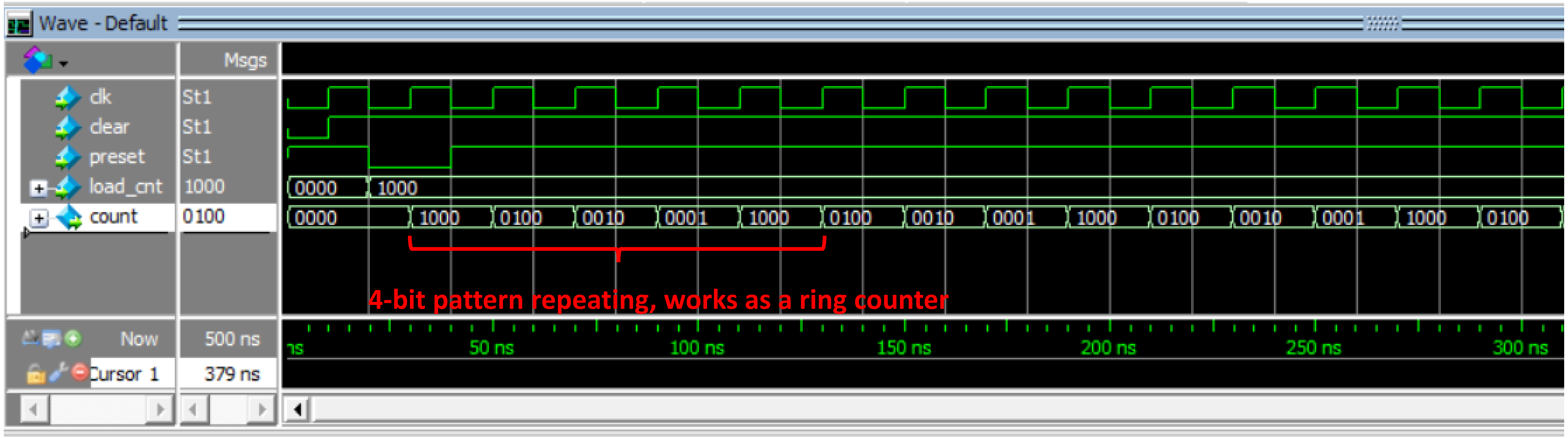
Each clock cycle count value is right shifted and Count vector LSB which is output of last flipflop in shift register is assigned to input of first flipflop



Both in Simulation and Synthesis it works as a Ring Counter

4-bit Ring Counter Using Blocking Assignment : Alternate Implementation

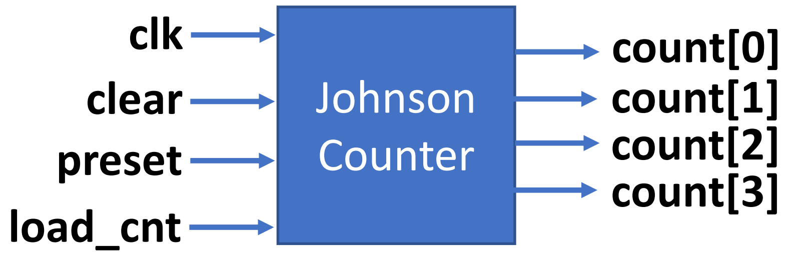
Simulation Result



Homework Assignment-3a : Johnson Counter

❑ Develop Synthesizable SystemVerilog Model for a 4-bit Johnson Counter

- Synthesize Johnson counter
- Run simulation using Johnson counter testbench provided
- Review synthesis results (resource usage and RTL netlist/schematic)
- Review input and output signals in simulation waveform.
- Create SystemVerilog module name as **johnson_counter**
- **Use non-blocking assignment statement when implementing johnson_counter**



4-bit Johnson Counter with initial load count value = 4'b0000

Inverted count[0] value

Johnson Counter					
Clock Cycle	State	count[3]	count[2]	count[1]	count[0]
1	0	0	0	0	0
2	1	1	0	0	0
3	2	1	1	0	0
4	3	1	1	1	0
5	4	1	1	1	1
6	5	0	1	1	1
7	6	0	0	1	1
8	7	0	0	0	1
9	0	0	0	0	0

4-bit pattern repeats every 8 cycles

N-bit Johnson Counter has 2N states

❑ Declare below mentioned Primary Ports for Johnson Counter

- Input **clk** : clock
- Input **clear** : asynchronous reset / negedge signal
- Input **preset** : synchronous and active low signal
- Input **load_cnt** : Initial count value. Count Value initialized when **!preset**
- Output **count[3:0]** : output 4-bit count value

Homework Assignment-3a : Johnson Counter

- **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 Johnson counter
  - Post-Mapping schematic is optional to submit.
  - Explanation of FPGA resource usage in report is not required.

Note :

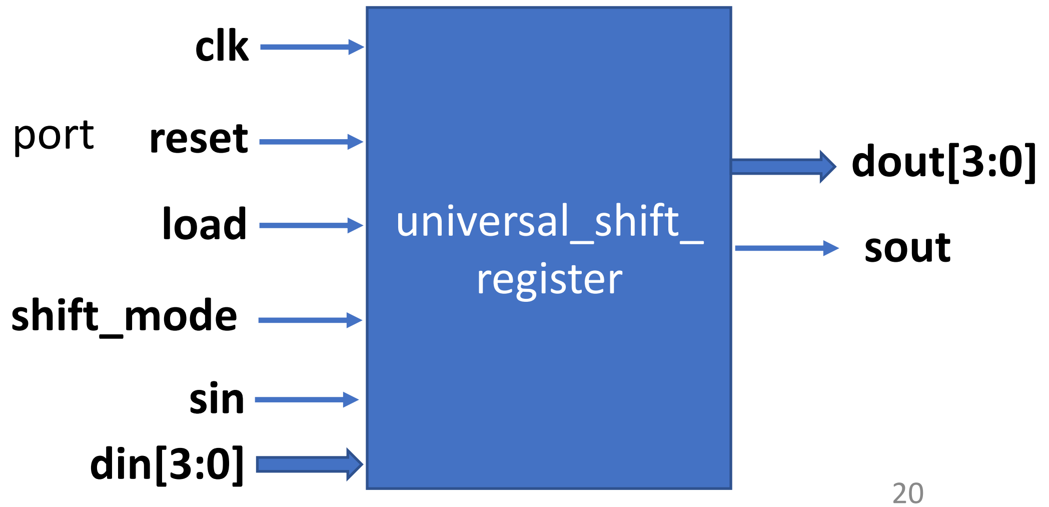
- Ring Counter SystemVerilog code provided in this document, can be used as a reference to implement Johnson counter
- Lab3 folder includes full testbench code and partial design template code
- It is not mandatory to use design template code provided in lab folder. Student can implement their design 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.
- Initial **load_cnt** signal value is set to 4'b0000 in testbench code provided.
- For learning purpose, student can update johnson_counter_testbench.sv code including initial **load_cnt** value to any other value of choice or any other stimulus under initial block, and more

Homework Assignment-3b : Universal Shift Register

- **Develop Synthesizable SystemVerilog model of a 4-bit Universal Shift Register**
  - Universal Shift Register should perform below mentioned 6 operations :
    - **PIPO, SIPO-L, SIPO-R, PISO-L, PISO-R, SISO-L, SISO-R**
  - Create module with name "universal_shift_register"
  - When implementing shift register for each mode of operation, use non-blocking assignment statements in **case** expression body under clocked always block. See example below :

```
shift_reg[0] <= sin;
shift_reg[1] <= shift_reg[0];
shift_reg[2] <= shift_reg[1];
shift_reg[3] <= shift_reg[2];
```

  - Declare below mentioned primary Ports of Universal Shift Register
    - Input **clk** : clock
    - Input **reset** : asynchronous reset / posedge signal
    - Input **load** : when set to '1' load parallel data through din port into 4-bit shift register
    - Input **din[3:0]** : 4-bit parallel data input
    - Output **dout[3:0]** : 4-bit parallel data output
    - Input **sin** : 1-bit serial data in
    - Output **sout** : 1-bit serial data out
    - Input **shift_mode** : selects type of shift operation



Homework Assignment-3b : Universal Shift Register

- **Implement shift operation based on table below**
  - In the table below, last two columns indicates for each shift operation through which input port data enters in universal shift register and through which output port data exits universal shift register
  - For PIPO, PISO-L and PISO-R, when input load signal is driven to '1' then all bits of shift register are loaded simultaneously through din[3:0] input port

shift_mode	Operation	Value of Input "load"	Input data port	Output data port
3'b000	PIPO (Parallel In Parallel Out)	1	din[3:0]	dout[3:0]
3'b001	SIPO-L (Left Shift Serial In Parallel Out)	0	sin	dout[3:0]
3'b010	SIPO-R (Right Shift Serial In Parallel Out)	0	sin	dout[3:0]
3'b011	PISO-L (Parallel In Left Shift Serial Out)	1	din[3:0]	sout
3'b100	PISO-R (Parallel In Right Shift Serial Out)	1	din[3:0]	sout
3'b101	SISO-L (Left Shift Serial In Left Shift Serial Out)	0	sin	sout
3'b110	SISO-R (Right Shift Serial In Right Shift Serial Out)	0	sin	sout

Homework Assignment-3b : Universal Shift Register

- **When implementing each shift operation in SystemVerilog code refer to table below on which bit position of internal variable shift_reg data enters and exits**

shift_mode	Operation	Internal Shift Reg assignment	Output dout and sout assignment
3'b000	PIPO (Parallel In Parallel Out)	N/A	dout[3:0] = din[3:0] sout = 0
3'b001	SIPO-L (Left Shift Serial In Parallel Out)	shift_reg[0] = sin	dout[3:0] = shift_reg[3:0] sout = 0
3'b010	SIPO-R (Right Shift Serial In Parallel Out)	shift_reg[3] = sin	dout[3:0] = shift_reg[3:0] sout = 0

3'b011	PISO-L (Parallel In Left Shift Serial Out)	shift_reg[0] = 1'b0	sout = shift_reg[3]
			dout = 0

3'b100	PISO-R (Parallel In Right Shift Serial Out)	shift_reg[3] = 1'b0	sout = shift_reg[0]
			dout = 0

3'b101	SISO-L (Left Shift Serial In Left Shift Serial Out)	shift_reg[0] = sin	sout = shift_reg[3]
			dout = 0

3'b110	SISO-R (Right Shift Serial In Right Shift Serial Out)	shift_reg[3] = sin	sout = shift_reg[0]
			dout = 0

3'b111	Default Case (No Shifting Operation)	N/A	sout = 0, dout = 0

Homework Assignment-3b : Universal Shift Register

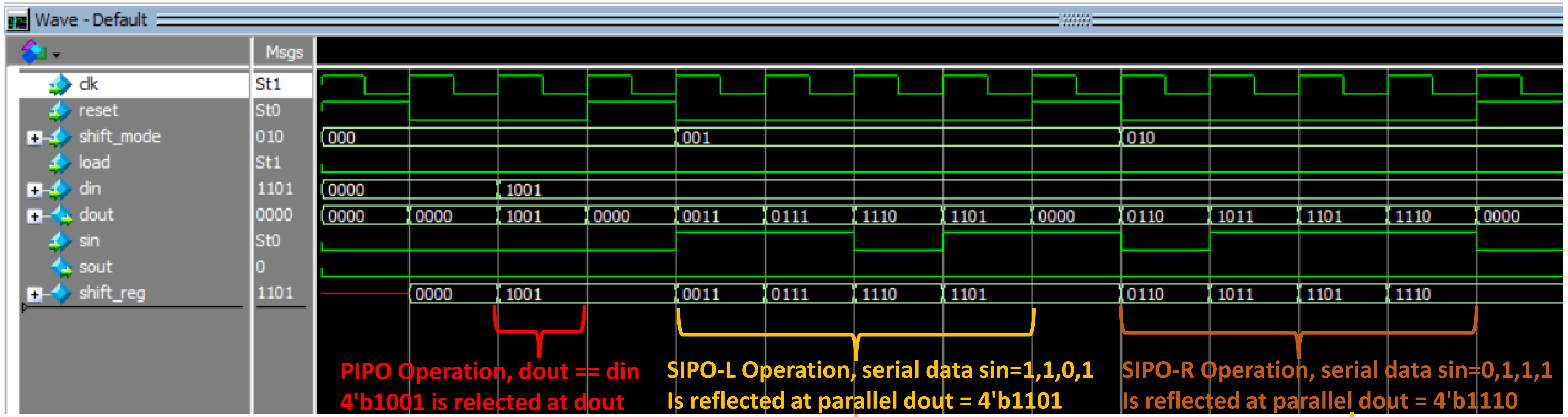
- **Submit report which should include :**
  - SystemVerilog design code
  - Synthesis resource usage and schematic generated from RTL netlist viewer
  - Simulation snapshot and explain simulation result to confirm RTL model developed works as a Universal Shift Register
  - Post-Mapping schematic is optional to submit.
  - Explanation of FPGA resource usage in report is not required.

Note :

- Lab3 folder includes full testbench code and partial design template code for universal shift 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 Assignment-3b : Universal Shift Register

❑ Reference Simulation Waveform for PIPO, SIPO-L, SIPO-R:



PIPO Operation, dout == din
4'b1001 is relected at dout

SIPO-L Operation, serial data sin=1,1,0,1
Is reflected at parallel dout = 4'b1101

SIPO-R Operation, serial data sin=0,1,1,1
Is reflected at parallel dout = 4'b1110

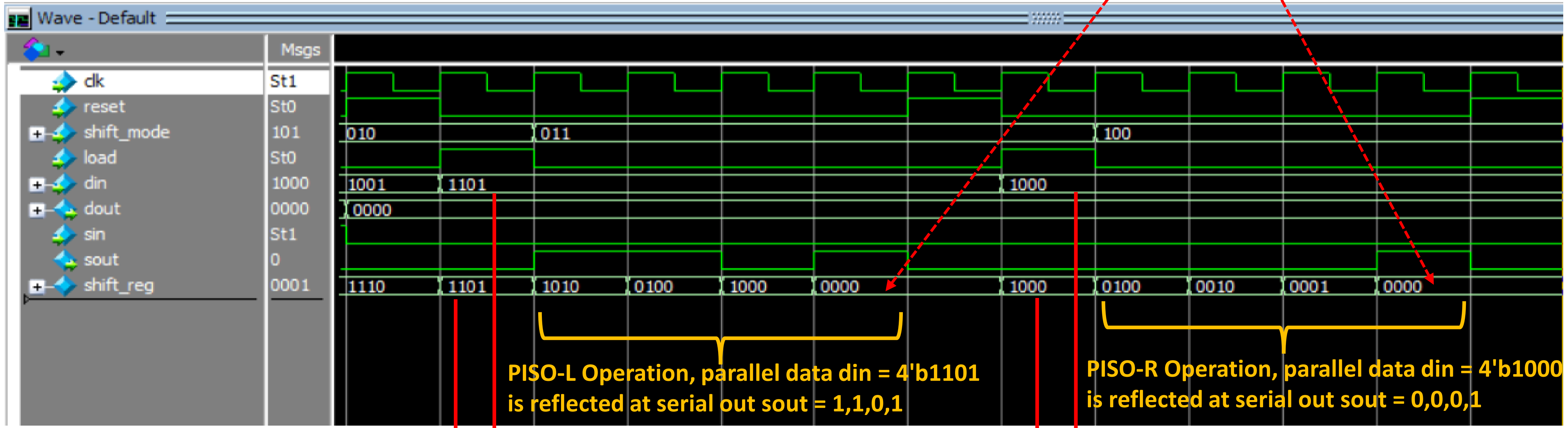
Note : In SIPO-L since this is Left shift operation, serial data enters shift register through shift_reg[0]
shift_reg[0] <= sin
And hence after shifting in all 4 serial data bits, internal shift_reg variable and external parallel dout port will reflect data in the same order it entered
i.e. sin= 1,1,01 and dout = 4'b1101

Note : In SIPO-R since this is Right shift operation, serial data enters shift register through shift_reg[3]
shift_reg[3] <= sin
And hence after shifting in all 4 serial data bits, internal shift_reg variable and external parallel dout port will reflect data in the reverse order it is entered
i.e. sin= 0,1,1,1 and dout = 4'b1110

Homework Assignment-3b : Universal Shift Register

❑ **Reference Simulation Waveform for PISO-L, PISO-R:**

When all bits of internal shift_reg are shifted out it reflects 4'b0000 value



PISO-L Operation, parallel data din = 4'b1101 is reflected at serial out sout = 1,1,0,1

PISO-R Operation, parallel data din = 4'b1000 is reflected at serial out sout = 0,0,0,1

In case of PISO-L Operation, first when load == '1' then parallel data 4'b1101 is loaded to internal shift_reg variable through din port.
And then Left Shift operation is performed to serially shift out 1-bit data at a time at each clk posedge on to sout port.

Note : Since this is Left shift operation, upon left shift each clk cycle, value '0' is entered through shift_reg[0] register.
shift_reg[0]<=1'b0;
And, sout gets value from shift_reg[3].
sout<=shift_reg[3];

In case of PISO-R Operation, first when load == '1' then parallel data 4'b1000 is loaded to internal shift_reg variable through din port.
And then Right Shift operation is performed to serially shift out 1-bit data at a time at each clk posedge on to sout port.

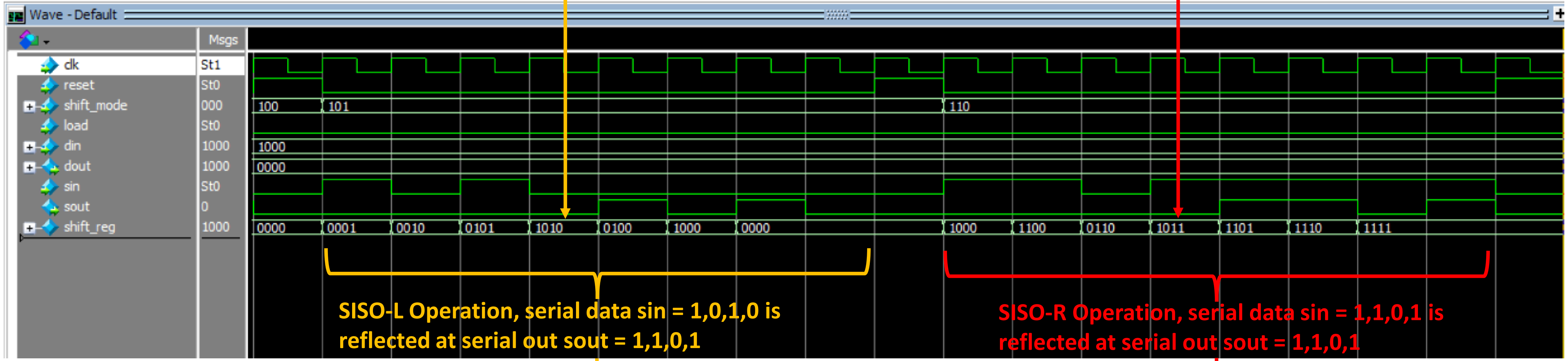
Note : Since this is Right shift operation, upon right shift each clk cycle, value '0' is entered through shift_reg[3] register.
shift_reg[3]<=1'b0;
And, sout gets value from shift_reg[0].
sout<=shift_reg[0];

Homework Assignment-3b : Universal Shift Register

❑ Reference Simulation Waveform for SISO-L, SISO-R:

SISO-L Operation, after all serial data sin = 1,0,1,0 is shifted in, then internal shift_reg relects same value shift_reg=4'b1010 similar to order in which data entered through sin

SISO-R Operation, after all serial data sin = 1,1,0,1 is shifted in, then internal shift_reg relects value shift_reg=4'b1011 which is in reverse order in which data entered through sin



SISO-L Operation, serial data sin = 1,0,1,0 is reflected at serial out sout = 1,1,0,1

SISO-R Operation, serial data sin = 1,1,0,1 is reflected at serial out sout = 1,1,0,1

Note : Since this is Left shift operation, serial data sin enters into shift_reg[0] and serial data exits through shift_reg[3]
shift_reg[0] <= sin;
sout <= shift_reg[3];

Note : Since this is Right shift operation, serial data sin enters into shift_reg[3] and serial data exits through shift_reg[0]
shift_reg[3] <= sin;
sout <= shift_reg[0];