Habib University
School of Science & Engineering

Course:	Digital System Design
Semester	Fall 2024
Assignment #	1
Due Date :	Feb. 16, 2024
Instructor :	Syed Arsalan Jawed
Total Marks :	100

Instructions for Students

1. Print the following question paper and provide your answer within the empty spaces given after each question.
2. Keep your answers concise and to the point. Provide neat and tidy calculations and plots.
3. Make reasonable assumptions related to concepts and associated parameters motivated by your text and reference book.
4. Support your answer with equations and simulation where asked and needed.

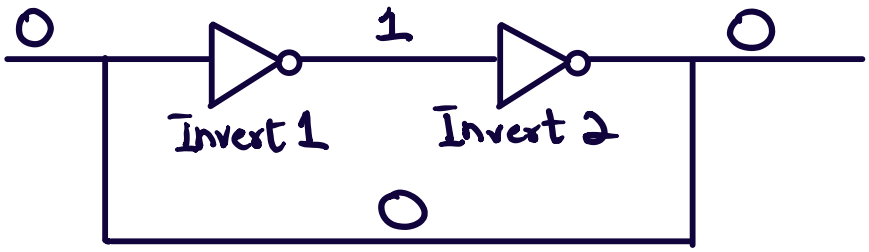
Name: Huzaifah Tariq Ahmed

(ha07151)

CE Batch of 2025

Question # 1 : [CLO-1]

Two digital inverters are connected in the back-to-back; input of invert1 is connected to output of inverter2 and output of inverter1 is connected to input of inverter2. You may assume that the input of the first inverter is 0 and the output of the second inverter is 0 as well. Briefly comment what functionality this configuration is achieving and suggest a change in the circuit to provide a trigger to make the circuit change its output state from 0 to 1(VDD) where a low value of trigger should toggle that circuit state and a high value should not affect the existing state. [10]



This configuration of two cascaded inverters, achieves positive feedback. It is useful where we want to retain the previous state, instead of flipping the state.

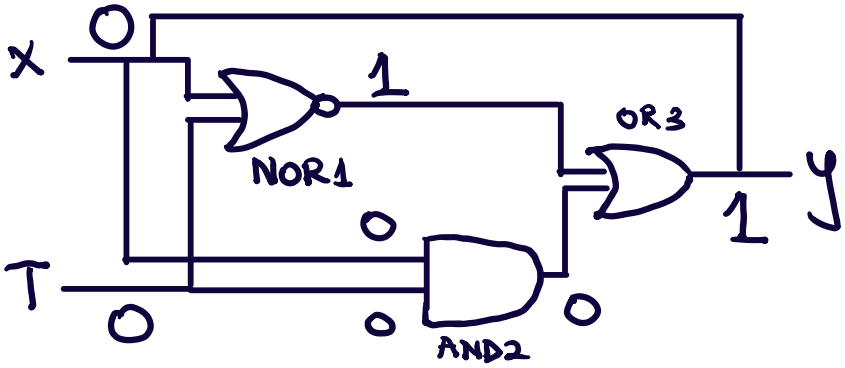
Input (x)	Trigger (T)	Output (y)
0	0	1
0	1	0
1	0	0
1	1	1

K-Map

x \ T	0	1
0	1	
1		1

$y = \bar{x}\cdot\bar{T} + x\cdot T$

$y = \overline{x+T} + x\cdot T$



Question # 2 : [CLO-1]

Two flip-flops are connected in series with the output of the first flip-flop directly connected to the input of the second. The clock to Q delay of flip-flops is 1nsec. The setup time is 1nsec. The hold time is 1nsec. Calculate the maximum frequency this digital circuit can operate at. [10]

With 2 flip flops connected in series to each other, we need to find the minimum clock time of our configuration, in order to find the maximum frequency.

For the two flip flops in series, t will be equal to;

$$\Rightarrow t = \text{Setup time} + \text{C2Q Delay (1st FF)} + \text{C2Q Delay (2nd FF)}$$

$$= 1\text{ns} + 1\text{ns} + 1\text{ns} = 3\text{ns}$$

$$\Rightarrow \text{Max Freq} = \frac{1}{t}$$

$$= \frac{1}{3 \times 10^{-9}}$$

$$= 333.3 \times 10^{6}$$

$$= 333 \text{ MHz}$$

Question # 3 : [CLO-2]

Q3. Draw a 2:1 Multiplexer using AND and OR gates then convert it into a logic circuit that uses only NAND and NOR gates to reduce the overall propagation delay. Comment on why a NAND/NOR implementation will reduce the t_P. [10]

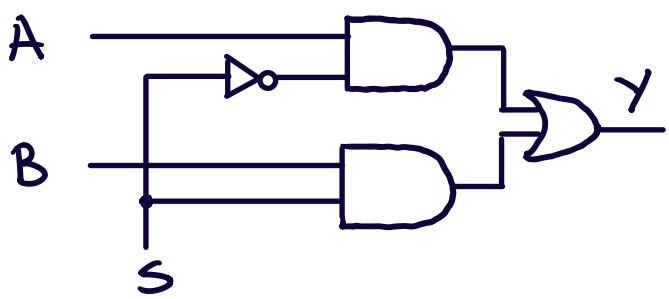
A	B	S	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

K-Map

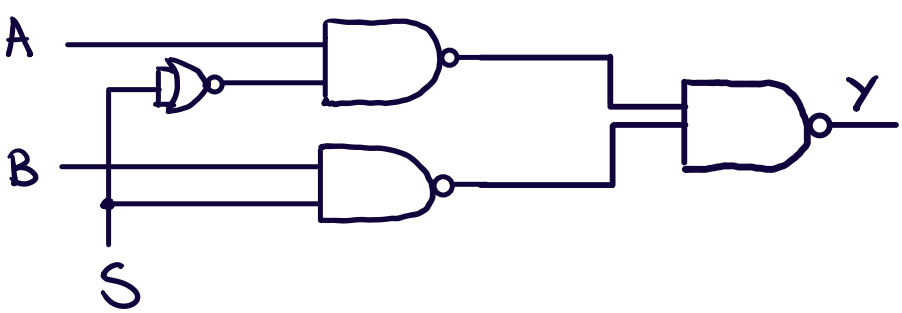
A \ BS	00	01	11	10
0			1	
1	1		1	1

$Y = BS + A\bar{S}$

Using AND/OR Gates



Using NAND/NOR Gates

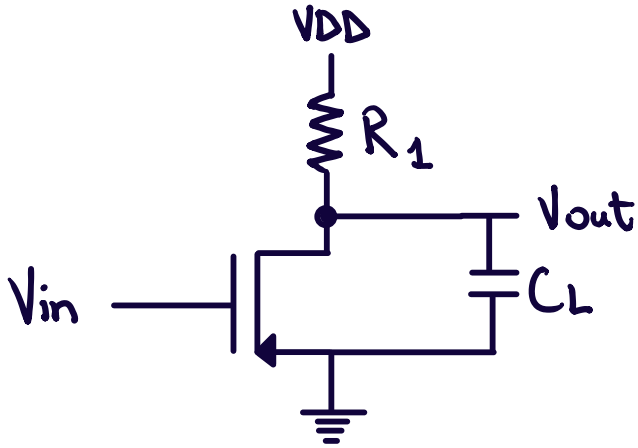


Reducing propagation delay in digital circuits is achieved through a NAND/NOR implementation, utilizing fewer MOSFETs compared to AND gates. NAND gates (4 MOSFETs) and inverters (2 MOSFETs) are basic components, while AND gates, composed of 6 MOSFETs due to their inherent inverting behavior, introduce more internal resistances and capacitances, leading to increased RC-delays. Similarly, NOR gates benefit from lower MOSFET counts, aiding in minimizing propagation delays.

Question # 4 : [CLO-2]

You have an NMOS and a resistor, there is no PMOS. Implement a digital inverter and calculate the expressions for t_{PLH} and t_{PHL}. If kn=100uA/V2 and W/L=1 and VDD=5V and VTH=0.5V, calculate exact values of these propagation delays assuming you are driving a capacitive load of a typical digital oscilloscope probe (You may chose to ignore resistive component of the probe) .

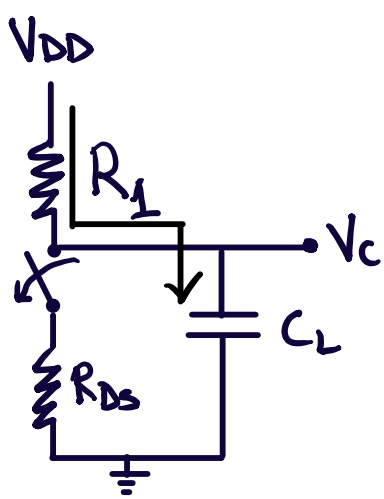
[20]



* If input is low, NMOS Channel will not be formed, and current doesn't flow through it. The resistor will pull up the output to make it high

We can model our Nmos with a resistor R_{DS} (Signifying NMOS Channel Resistance) and a switch (Actual Function of NMOS)

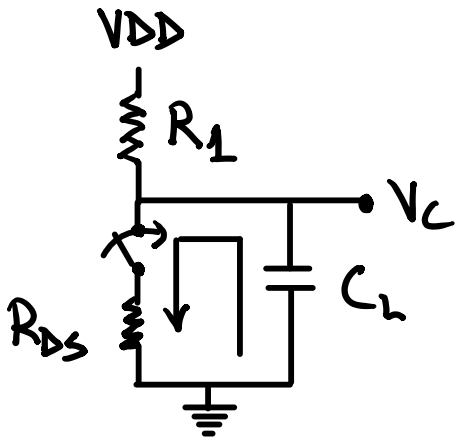
For t_{PLH}



With no nmos channel, Current passes through R_1, and C_L and then goes to ground. So the time delay in this case is simply

$$t_{PLH} = R_1 C_L \ln(2)$$

For t_{PHL}



When input goes to high, nmos channel get's formed, and now the pull up circuit gets disconnected. We see Capacitor discharge, so the time delay in this case will no longer involve R_1, and it will simply be;

$$t_{PHL} = R_{on} C_L \ln(2)$$

Now we have information to calculate R_{on}.

$$R_{on} = \frac{1}{k_n \left(\frac{W}{L}\right)(V_{GS} - V_{TH})} = \frac{1}{(100\times10^{-6})(1)(5-0.5)}$$

$$= 2.22\times10^{3}$$

$$= 2.22\ K\Omega \quad ; \quad V_{GS} = V_{DD}$$

C_L can be taken to be a standard value of 14 pF.

Hence,

$$t_{PHL} = (2.22\times10^{3})(14\times10^{-12})\ln(2)$$
$$= 21.56\ ns.$$

Now, we don't have sufficient information, to calculate R_1, and use it to find t_{PLH}. Therefore we will take help of the relationship b/w t_{PLH} and t_{PHL}.

$$t_{PLH} = t_{PHL} = 21.56\,ns \quad ; R_1 = R_{on}$$

Question # 5 : [CLO-2, CLO-3, CLO-4]

Write Verilog code of a module that implements a 16-bit shift register using cyclic constructs that proper infer flipflops and avoid latches, with and without for-loop constructs. The complete code should be of less than 15 lines. Attach simulation snaps of this code from Xilinx Vivado software.

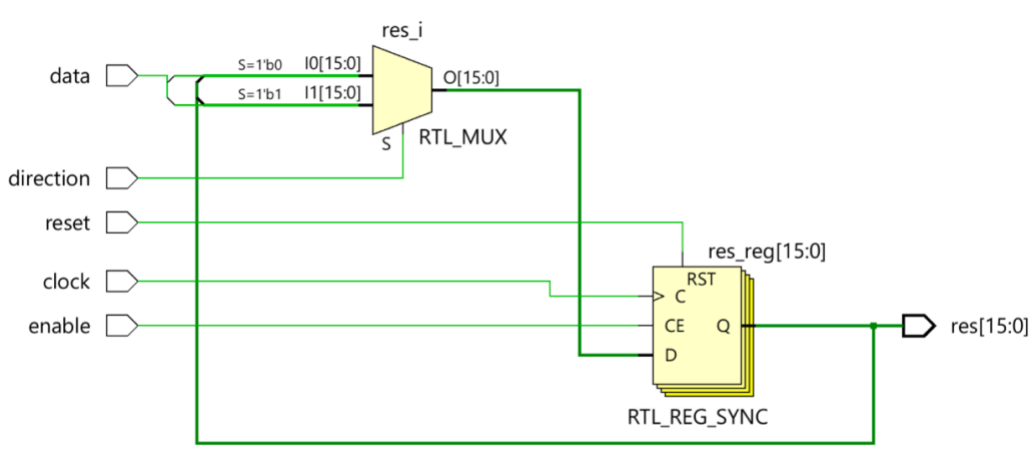
[20]

Without For Loop:

Code:

```
`timescale 1ns / 1ps

module sixteen_bit_shift_register (
    data,           // Serial Data
    clock,
    enable,         // only shift if enable is 1
    reset,          // reset output to 0
    direction,      // specifies direction of shift (left or right)
    res);           // output

    input data,
          clock,
          enable,
          reset,
          direction;

    output reg [15:0] res;

  always @ (posedge clock)
        if (reset)
            res <= 16'b0;
        else begin
          if (enable)
            case (direction)
              0 : res <= {res[14:0],data};  // Shift Left
              1 : res <= {data, res[15:1]}; // Shift Right
            endcase
          else
            res <= res;
        end
endmodule
```

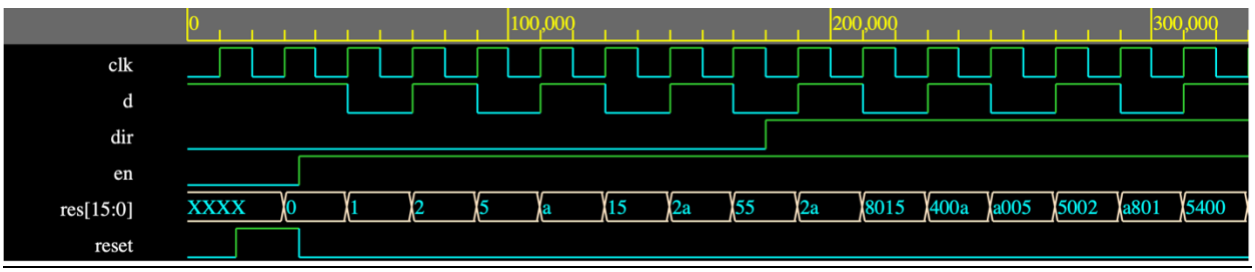
Schematic:



Log Output:

```
reset=0 data=1, enable=0, direction=0, result=xxxxxxxxxxxxxxxx
reset=1 data=1, enable=0, direction=0, result=xxxxxxxxxxxxxxxx
reset=1 data=1, enable=0, direction=0, result=0000000000000000
reset=0 data=1, enable=1, direction=0, result=0000000000000000
reset=0 data=0, enable=1, direction=0, result=0000000000000001
reset=0 data=1, enable=1, direction=0, result=0000000000000010
reset=0 data=0, enable=1, direction=0, result=0000000000000101
reset=0 data=1, enable=1, direction=0, result=0000000000001010
reset=0 data=0, enable=1, direction=0, result=0000000000010101
reset=0 data=1, enable=1, direction=0, result=0000000000101010
reset=0 data=0, enable=1, direction=0, result=0000000001010101
reset=0 data=0, enable=1, direction=1, result=0000000001010101
reset=0 data=1, enable=1, direction=1, result=0000000000101010
reset=0 data=0, enable=1, direction=1, result=1000000000010101
reset=0 data=1, enable=1, direction=1, result=0100000000001010
reset=0 data=0, enable=1, direction=1, result=1010000000000101
reset=0 data=1, enable=1, direction=1, result=0101000000000010
reset=0 data=0, enable=1, direction=1, result=1010100000000001
reset=0 data=1, enable=1, direction=1, result=0101010000000000
reset=0 data=1, enable=1, direction=1, result=1010101000000000
```

Simulation:



Test Bench:

```
1 `timescale 1ns / 1ps
2
3 `include "16_bit_shift_register.v"
4
5 module tb_sixteen_bit_shift_register;
6
7
8     reg d, clk, en, reset, dir;
9     wire [15:0] res;
10
11    // Instantiate 16 Bit Shift Register
12    sixteen_bit_shift_register sbsr (
13        .data(d),
14        .clock(clk),
15        .enable(en),
16        .reset(reset),
17        .direction(dir),
18        .res(res)
19    );
20
21    // Clock generation
22    always #10 clk = ~clk; // Generate a clock with a period of 20ns
23
24    // Initialize variables to default values at time 0
25    initial begin
26        $dumpfile("dump.vcd");
27        $dumpvars;
28        clk <= 0;
29        en <= 0;
30        dir <= 0;
31        reset <= 0;
32        d <= 1;
```

```
        // Reset the counters
        #15 reset = 1;
        #20 reset = 0;
            en = 1;

        // For 7 clocks, drive alternate values to data pin
        repeat (7) @ (posedge clk)
            d <= ~d;

        //Shift direction and drive alternate value to data pin for another 7 clocks
        #10 dir <= 1;
        repeat (7) @ (posedge clk)
            d <= ~d;

        //Drive nothing for next 7 clocks, allow shift register to simply shift based on dir
        repeat (7) @ (posedge clk)

        // Finish the simulation
        $finish;
    end

    // Monitor values of these variables and print them into the logfile for debug
   initial
     $monitor ("reset=%0b data=%b, enable=%0b, direction=%0b, result=%b", reset, d, en, dir, res);

endmodule
```

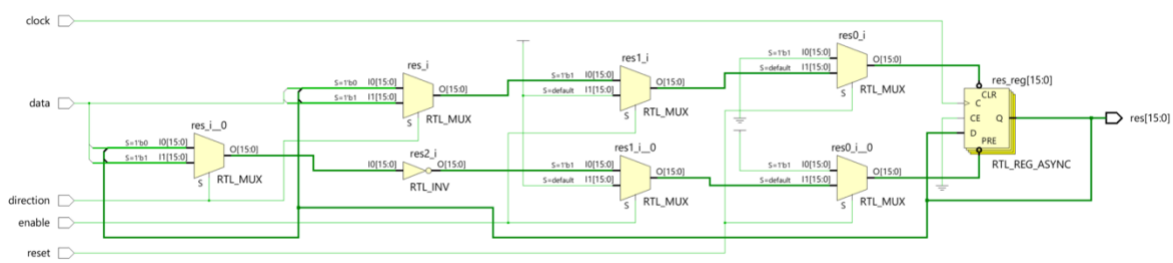
With For Loop:

Code:

```
`timescale 1ns / 1ps

module sixteen_bit_shift_register_with_for_loop (
    data,           // Serial Data
    clock,
    enable,         // only shift if enable is 1
    reset,          // reset output to 0
    direction,      // specifies direction of shift (left or right)
    res);           // output

    integer i;

    input data,
          clock,
          enable,
          reset,
          direction;

    output reg [15:0] res;


```

```
  always @ (posedge clock or posedge reset or posedge enable)
    begin
          if (reset)
              res <= 16'b0;
          else begin
            if (enable)
            begin
              case (direction)
                0 : begin
                    for (i = 14; i >= 0; i = i - 1) // Shift Left
                        res[i + 1] <= res[i];
                    res[0] <= data;
                    end
                1 : begin
                    for (i = 0; i < 15; i = i + 1)  // Shift Right
                        res[i] <= res[i + 1];
                    res[15] <= data;
                    end
              endcase
            end
            else
              res <= res;
          end
    end
endmodule
```

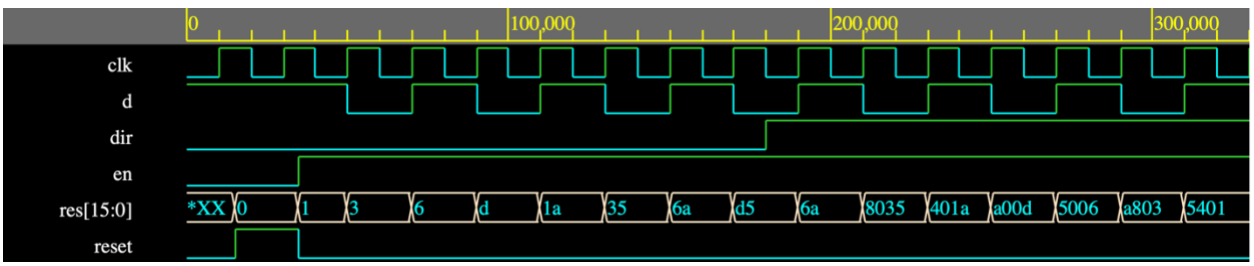
Schematic:



Log Output:

```
# KERNEL: reset=0 data=1, enable=0, direction=0, result=xxxxxxxxxxxxxxxx
# KERNEL: reset=1 data=1, enable=0, direction=0, result=0000000000000000
# KERNEL: reset=0 data=1, enable=1, direction=0, result=0000000000000001
# KERNEL: reset=0 data=0, enable=1, direction=0, result=0000000000000011
# KERNEL: reset=0 data=1, enable=1, direction=0, result=0000000000000110
# KERNEL: reset=0 data=0, enable=1, direction=0, result=0000000000001101
# KERNEL: reset=0 data=1, enable=1, direction=0, result=0000000000011010
# KERNEL: reset=0 data=0, enable=1, direction=0, result=0000000000110101
# KERNEL: reset=0 data=1, enable=1, direction=0, result=0000000001101010
# KERNEL: reset=0 data=0, enable=1, direction=0, result=0000000011010101
# KERNEL: reset=0 data=0, enable=1, direction=1, result=0000000011010101
# KERNEL: reset=0 data=1, enable=1, direction=1, result=0000000001101010
# KERNEL: reset=0 data=0, enable=1, direction=1, result=1000000000110101
# KERNEL: reset=0 data=1, enable=1, direction=1, result=0100000000011010
# KERNEL: reset=0 data=0, enable=1, direction=1, result=1010000000001101
# KERNEL: reset=0 data=1, enable=1, direction=1, result=0101000000000110
# KERNEL: reset=0 data=0, enable=1, direction=1, result=1010100000000011
# KERNEL: reset=0 data=1, enable=1, direction=1, result=0101010000000001
```

Simulation:



Test Bench:

```
1 `timescale 1ns / 1ps
2
3 `include "16_bit_shift_register_with_for_loop.v"
4
5 module tb_sixteen_bit_shift_register_with_for_loop;
6
7
8     reg d, clk, en, reset, dir;
9     wire [15:0] res;
10
11    // Instantiate 16 Bit Shift Register
12    sixteen_bit_shift_register_with_for_loop sbsrwfl (
13        .data(d),
14        .clock(clk),
15        .enable(en),
16        .reset(reset),
17        .direction(dir),
18        .res(res)
19    );
20
21    // Clock generation
22    always #10 clk = ~clk; // Generate a clock with a period of 20ns
23
```

```
    // Initialize variables to default values at time 0
    initial begin
        $dumpfile("dump.vcd");
        $dumpvars;
        clk <= 0;
        en <= 0;
        dir <= 0;
        reset <= 0;
        d <= 1;

        // Reset the counters
        #15 reset = 1;
        #20 reset = 0;
            en = 1;

        // For 7 clocks, drive alternate values to data pin
        repeat (7) @ (posedge clk)
            d <= ~d;

        //Shift direction and drive alternate value to data pin for another 7 clocks
        #10 dir <= 1;
        repeat (7) @ (posedge clk)
            d <= ~d;

        //Drive nothing for next 7 clocks, allow shift register to simply shift based on dir
        repeat (7) @ (posedge clk)

        // Finish the simulation
        $finish;
    end

    // Monitor values of these variables and print them into the logfile for debug
   initial
     $monitor ("reset=%0b data=%b, enable=%0b, direction=%0b, result=%b", reset, d, en, dir, res);

endmodule
```

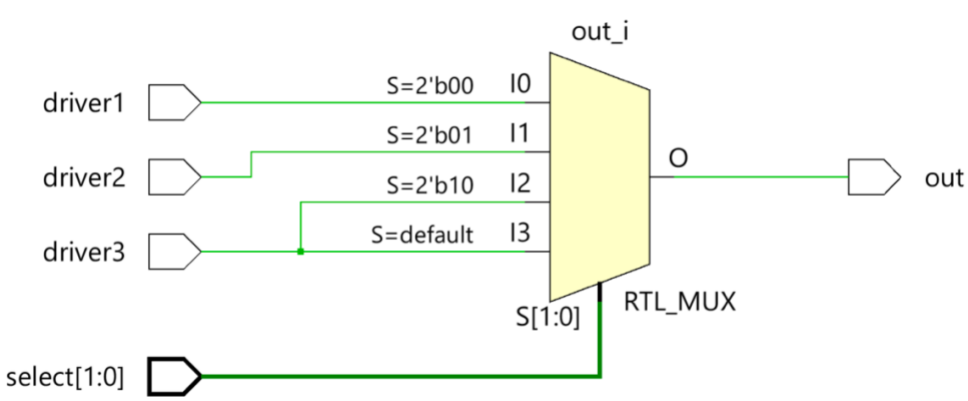
Question # 6 : [CLO-2, CLO-3,CLO-4]

Write Verilog code of a module that contains four drivers that drive a single shared 1-bit output OUT in a half-duplex manner. When the select is 00, input of driver1 appears on OUT, when select is 01, input of driver2 appears on OUT, when select is 10, input of driver3 appears on OUT, otherwise input of driver3 appears on OUT. Attach simulation snaps of this code from Xilinx Vivado software. [20]

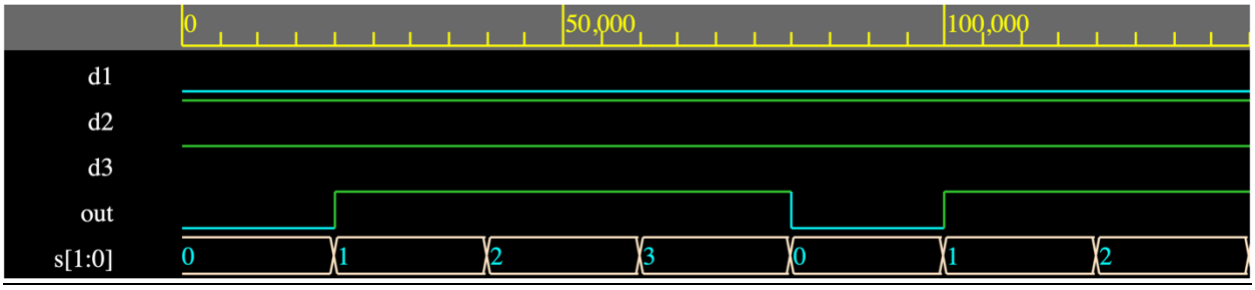
Code:

```
`timescale 1ns/1ps

module four_to_one_mux(driver1,
                       driver2,
                       driver3,
                       select,
                       out);

    input driver1,
          driver2,
          driver3;

    input [1:0] select;

    output reg out;

    always @ (*)
    begin
      case (select)
        2'b00 : out <= driver1;
        2'b01 : out <= driver2;
        2'b10 : out <= driver3;
        default : out <= driver3;
      endcase
    end
endmodule
```

Schematic:



Simulation:



Test Bench:

```
1 `timescale 1ns/1ps
2
3 `include "4_to_1_MUX.v"
4
5 module tb_four_to_one_MUX;
6
7   reg d1,d2,d3;
8
9   reg [1:0] s;
10
11  wire out;
12
13  // Instantiate 4 to 1 MUX
14  four_to_one_mux ftom (
15    .driver1(d1),
16    .driver2(d2),
17    .driver3(d3),
18    .select(s),
19    .out(out)
20    );
21
22  initial begin
23    $dumpfile("dump.vcd");
24    $dumpvars;
25
26    //initialize variables
27    s = 2'b00;
28    d1 = 1'b0;
29    d2 = 1'b1;
30    d3 = 1'b1;
31
```

```
    #20 s = 2'b01;
    #20 s = 2'b10;
    #20 s = 2'b11;
    #20 s = 2'b00;
    #20 s = 2'b01;
    #20 s = 2'b10;
    #20 s = 2'b11;

    // Ending the simulation!
    $finish;
  end

  // Below command is for monitoring the output
  always @(out) $display("OUTPUT = %b", out);

endmodule
```

Question # 7 : [CLO-1]

Express difference between blocking and non-blocking assignments of Verilog HDL. Write a code in Verilog and synthesize it in Xilinx Vivado to show the effect of using the wrong assignment with the wrong construct. [10]

Non-Blocking v/s Blocking Assignments:

Blocking and non-blocking assignments are utilized in procedural blocks for register value assignments, with different behaviors. In combinational blocks, blocking assignments, akin to high-level languages, evaluate and store expressions sequentially. Conversely, non-blocking assignments, common in sequential circuits, execute subsequent instructions concurrently without waiting for assignment completion, beneficial for simultaneous register updates in a clock cycle. In the case of a shift register, non-blocking assignments accurately model its behavior, revealing four D flip-flops in the schematic. In contrast, blocking assignments alter the shift register's behavior, resulting in an incorrect schematic showing only one D flip-flop due to their sequential nature.

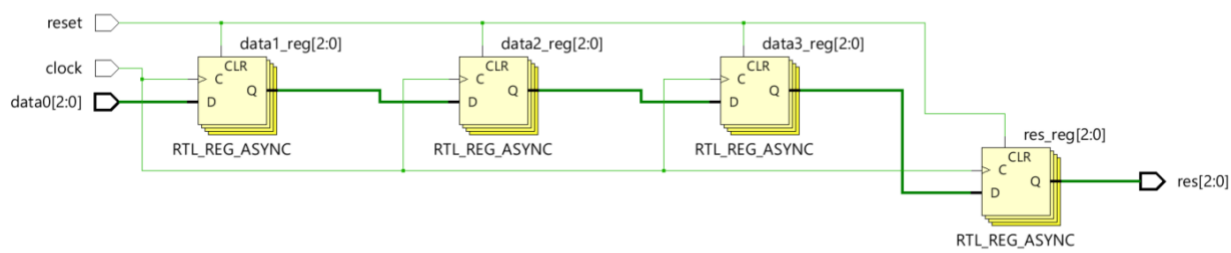
Non-Blocking 3-Bit Shift Register:

Code:

```
`timescale 1ns / 1ps

module three_bit_shift_register_non_blocking (
    data0,
    clock,
    reset,          // reset output to 0
    res);           // output

  input [2:0] data0;

  input clock,reset;

  reg [2:0] data1,
            data2,
            data3;

  output reg [2:0] res;

  always @ (posedge clock or posedge reset)
    begin
```

```
        if (reset)
        begin
            data1 <= 0;
            data2 <= 0;
            data3 <= 0;
            res <= 0;
        end
        else begin
            data1 <= data0;
            data2 <= data1;
            data3 <= data2;
            res <= data3;
        end
    end
endmodule
```

Schematic:



Blocking 3-Bit Shift Register:

Code:

```
`timescale 1ns / 1ps

module three_bit_shift_register_blocking (
    data0,
    clock,
    reset,          // reset output to 0
    res);           // output

  input [2:0] data0;

  input clock,reset;

  reg [2:0] data1,
            data2,
            data3;
```

```
  output reg [2:0] res;

  always @ (posedge clock or posedge reset)
    begin
          if (reset)
          begin
              data1 = 0;
              data2 = 0;
              data3 = 0;
              res = 0;
          end
          else begin
              data1 = data0;
              data2 = data1;
              data3 = data2;
              res = data3;
          end
    end
endmodule
```

Schematic: