ECS 409/609
Assignment 3
Sequential Based Assignment Solutions

Kunwar Arpit Singh

Submission Deadline: September 14c, 2025

For code repository of this assignment: **Verilog Assignments**

---

1. Problem 1

Problem statement

Implement a verilog module of an SR latch with asynchronous enable and reset.

Verilog source (Sequential)

```
module sr_latch(input S, input R, input en, input reset, output reg Q, output reg Q_n);
    always @(S, R, en, reset) begin
        if (reset) begin
            Q <=1'b0;
            Q_n <= 1'b1;
        end
        else if (en) begin
            case ({S, R})
                2'b00: begin
                    Q <= Q;
                    Q_n <= Q_n;
                end
                2'b01: begin
                    Q <= 1'b0;
                    Q_n <=1'b1;
                end
                2'b10: begin
                    Q <= 1'b1;
                    Q_n <=1'b0;
                end
                2'b11: begin
                    Q <= 1'bx;
                    Q_n <= 1'bx;
                end
            endcase
        end
```

```
    end
endmodule
```

Testbench

```
module sr_latch_tb;
    reg S;
    reg R;
    reg en;
    reg reset;
    wire Q;
    wire Q_n;
    reg [8*24-1:0] name;
    sr_latch dut (.S(S), .R(R), .en(en), .reset(reset), .Q(Q), .Q_n(Q_n));

    initial begin
        name = " Kunwar Arpit Singh 22185";
        $display("Kunwar Arpit Singh");
        $dumpfile("assign3_problem1_sr_latch.vcd");
        $dumpvars(1, sr_latch_tb);
        $display("Kunwar Arpit Singh");

        $display("| reset | en | S | R | Q | Q_n |");
        $display("---------------------------------");

        reset = 1'b1; S = 1'b0; R = 1'b0; en = 1'b0; #10;
        $display("|   %b   |  %b  | %b | %b | %b  |  %b  |",reset, en, S, R, Q,
            Q_n);
        reset = 1'b0; S = 1'b0; R = 1'b0; en = 1'b0; #10;
        $display("|   %b   |  %b  | %b | %b | %b  |  %b  |",reset, en, S, R, Q,
            Q_n);
        reset = 1'b0; S = 1'b0; R = 1'b0; en = 1'b1; #10;
        $display("|   %b   |  %b  | %b | %b | %b  |  %b  |",reset, en, S, R, Q,
            Q_n);
        reset = 1'b0; S = 1'b0; R = 1'b1; en = 1'b1; #10;
        $display("|   %b   |  %b  | %b | %b | %b  |  %b  |",reset, en, S, R, Q,
            Q_n);
        reset = 1'b0; S = 1'b0; R = 1'b0; en = 1'b1; #10;
        $display("|   %b   |  %b  | %b | %b | %b  |  %b  |",reset, en, S, R, Q,
            Q_n);
        reset = 1'b0; S = 1'b1; R = 1'b0; en = 1'b1; #10;
        $display("|   %b   |  %b  | %b | %b | %b  |  %b  |",reset, en, S, R, Q,
            Q_n);
        reset = 1'b0; S = 1'b0; R = 1'b0; en = 1'b1; #10;
        $display("|   %b   |  %b  | %b | %b | %b  |  %b  |",reset, en, S, R, Q,
            Q_n);
        reset = 1'b0; S = 1'b1; R = 1'b1; en = 1'b1; #10;
        $display("|   %b   |  %b  | %b | %b | %b  |  %b  |",reset, en, S, R, Q,
            Q_n);
        $finish;
```

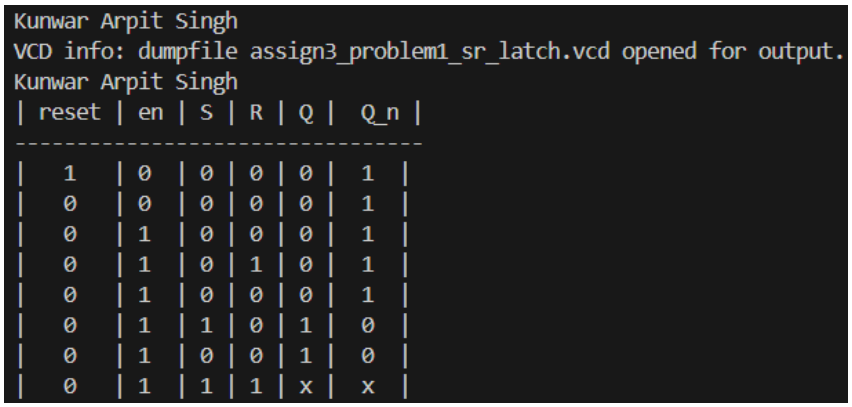
```
    end
endmodule
```

Output in Terminal

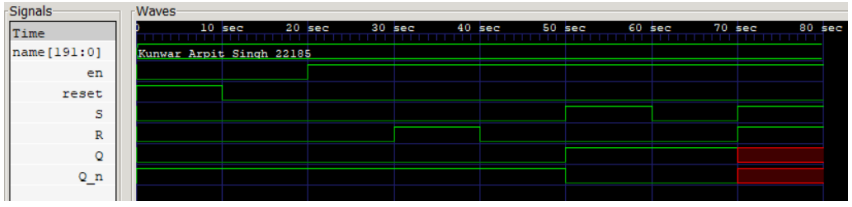
```
Kunwar Arpit Singh
VCD info: dumpfile assign3_problem1_sr_latch.vcd opened for output.
Kunwar Arpit Singh
| reset | en | S | R | Q | Q_n |
---------------------------------
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | x | x |
```

Output Screenshots

Terminal output (SR Latch)



GTKWave waveform (SR Latch)



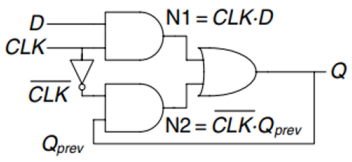
2. Problem 2

Problem statement

Implement the Verilog module of an improved version of D latch as shown in figure 1. Specify delays of 1 ns to each gate. With your simulator, show that the latch operates correctly.

CLK	D	Q_{prev}	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$Q = CLK \cdot D + \overline{CLK} \cdot Q_{prev}$



Verilog source (Sequential)

```
module d_latch_improved(input D, input CLK, output reg Q);
    wire CLK_n;
    wire D_and_CLK;
    wire Qprev_and_CLK_n;

    not #1 U1(CLK_n, CLK);
    and #1 U2(D_and_CLK, D, CLK);
    and #1 U3(Qprev_and_CLK_n, Q, CLK_n);

    always @(*) begin
        #1 Q = D_and_CLK | Qprev_and_CLK_n;
    end
    initial Q = 0;
endmodule
```

Testbench

```
module d_latch_improved_tb;
    reg D, CLK;
    wire Q;
    reg [8*24-1:0] name;

    d_latch_improved dut (.D(D), .CLK(CLK), .Q(Q) );

    initial begin
        name = "␣Kunwar␣Arpit␣Singh␣22185";
        $dumpfile("assign3_problem2_d_latch.vcd");
        $dumpvars(1, d_latch_improved_tb);
        $display("Kunwar␣Arpit␣Singh");
        CLK = 1; D = 0; #10;
        CLK = 0; #10;
```

```
15
16          CLK = 1; D = 1; #10;
17          CLK = 0; #10;
18
19          CLK = 1; D = 0; #10;
20          CLK = 0; D = 1; #10;
21
22          CLK = 1; D = 1; #10;
23          CLK = 0; #10;
24
25          CLK = 1; D = 0; #10;
26
27          CLK = 0; D = 1; #10;
28          CLK = 1; D = 0; #10;
29
30          CLK = 0; D = 0; #10;
31          CLK = 1; D = 1; #10;
32
33          CLK = 0; D = 1; #10;
34          CLK = 1; D = 1; #10;
35          $finish;
36      end
37
38      initial begin
39          $display("| CLK | D | Q |");
40          $monitor("| %b | %b | %b |", CLK, D, Q);
41      end
42  endmodule
```

Output in Terminal

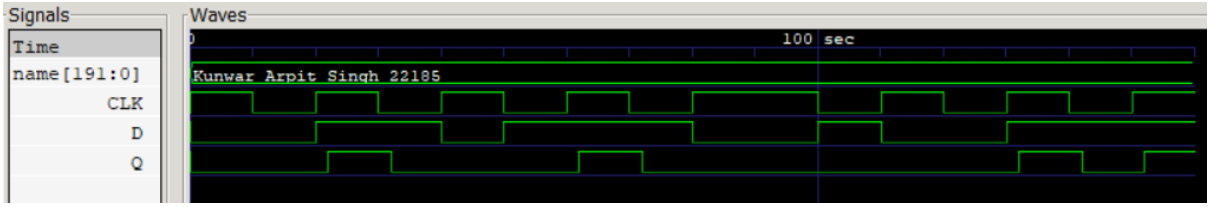
```
1  Kunwar Arpit Singh
2  VCD info: dumpfile assign2_problem6_wallace_tree_multiplier_4_bit.vcd opened
       for output.
3  Kunwar Arpit Singh
4  | A(B) | A(D)| B(B) | B(D) | Product(B) | Product(D) |
5  | 0001 | 1 | 0001 | 1 | 00000001 | 1 |
6  | 0101 | 5 | 1010 | 10 | 00110010 | 50 |
7  | 1111 | 15 | 1111 | 15 | 11100001 | 225 |
8  | 1001 | 9 | 0111 | 7 | 00111111 | 63 |
9  | 1100 | 12 | 0011 | 3 | 00100100 | 36 |
10 | 0100 | 4 | 1111 | 15 | 00111100 | 60 |
11 | 0000 | 0 | 1111 | 15 | 00000000 | 0 |
```

Output Screenshots

Terminal output (D Latch)

```
VCD info: dumpfile assign3_problem2_d_latch.vcd opened for output.
Kunwar Arpit Singh
| CLK | D | Q |
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |
```

GTKWave waveform (D Latch)



3. Problem 3

Problem statement

Design a verilog module of counter that counts from 7 to 0 (e.g., 7, 6, 5, 4, 3, 2, 1, 0).

Verilog source (Sequential)

```
module binary_counter(input CLK, input RESET, output reg [2:0] COUNT);
    always @(posedge CLK, posedge RESET) begin
        if (RESET) begin
            COUNT <= 3'b111;
        end
        else begin
            if (COUNT == 3'b000) begin
                COUNT <= 3'b111;
            end else begin
                COUNT <= COUNT - 1;
            end
        end
    end
endmodule
```

Testbench

```
module binary_counter_tb;
    reg CLK;
    reg RESET;
    wire [2:0] COUNT;
    reg [8*24-1:0] name;

    binary_counter dut (.CLK(CLK), .RESET(RESET), .COUNT(COUNT));

    initial begin
        CLK = 0;
        forever #5 CLK = ~CLK;
    end

    initial begin
        name = "␣Kunwar␣Arpit␣Singh␣22185";

        $dumpfile("assign3_problem3_counter.vcd");
        $dumpvars(1, binary_counter_tb);

        $display("Kunwar␣Arpit␣Singh");
        $display("|␣Reset␣|␣Clock␣|␣Count␣|");
        $display("----------------------");

        $monitor("|␣%b␣|␣%b␣|␣%b␣(%d)␣|", RESET, CLK, COUNT, COUNT);
        RESET = 1; #15;
```

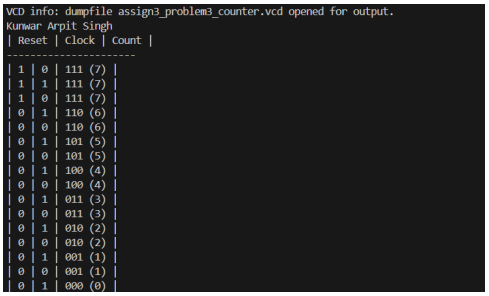
```
        RESET = 0; #60;
        $finish;
    end
endmodule
```

Output in Terminal

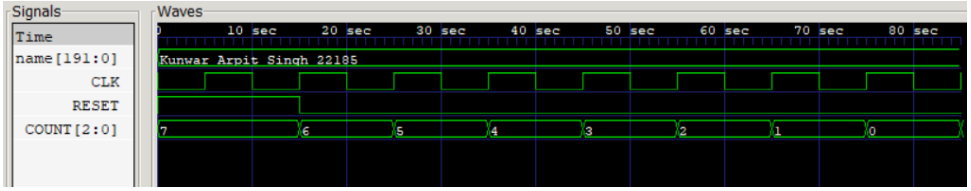
```
VCD info: dumpfile assign3_problem3_counter.vcd opened for output.
Kunwar Arpit Singh
| Reset | Clock | Count |
----------------------
| 1 | 0 | 111 (7) |
| 1 | 1 | 111 (7) |
| 1 | 0 | 111 (7) |
| 0 | 1 | 110 (6) |
| 0 | 0 | 110 (6) |
| 0 | 1 | 101 (5) |
| 0 | 0 | 101 (5) |
| 0 | 1 | 100 (4) |
| 0 | 0 | 100 (4) |
| 0 | 1 | 011 (3) |
| 0 | 0 | 011 (3) |
| 0 | 1 | 010 (2) |
| 0 | 0 | 010 (2) |
| 0 | 1 | 001 (1) |
| 0 | 0 | 001 (1) |
| 0 | 1 | 000 (0) |
```

Output Screenshots

Terminal output (Counter)



GTKWave waveform (Counter)



4. Problem 4

Problem statement

Implement a verilog module of UP/DOWN modulo 8 Gray Code Counter as shown in table 2, adding an Input UP. If UP = 1, the counter advances sequentially to the next number. Otherwise, UP = 0, the counter stays with the old value.

Verilog source (Sequential)

```
module gray_counter(input CLK, input RESET, input UP, output reg [2:0] GRAYout
    );

    always @(posedge CLK or posedge RESET) begin
        if (RESET) begin
            GRAYout <= 3'b000;
        end else begin
            if (UP) begin
                case (GRAYout)
                    3'b000: GRAYout <= 3'b001;
                    3'b001: GRAYout <= 3'b011;
                    3'b011: GRAYout <= 3'b010;
                    3'b010: GRAYout <= 3'b110;
                    3'b110: GRAYout <= 3'b111;
                    3'b111: GRAYout <= 3'b101;
                    3'b101: GRAYout <= 3'b100;
                    3'b100: GRAYout <= 3'b000;
                    default: GRAYout <= 3'b000;
                endcase
            end
        end
    end
endmodule
```

Testbench

```
module gray_counter_tb;
    reg CLK;
    reg RESET;
    reg UP;
    wire [2:0] GRAYout;

    gray_counter dut (.CLK(CLK), .RESET(RESET), .UP(UP), .GRAYout(GRAYout));

    initial begin
        CLK = 0;
        forever #5 CLK = ~CLK;
    end

```

```
    initial begin
        $dumpfile("assign3_problem4_gray_code_counter.vcd");
        $dumpvars(1, gray_counter_tb);

        $display("Kunwar Arpit Singh");
        $display("| Reset | UP | Gray Output |");
        $display("----------------------------");
        $monitor("|   %b   |  %b |    %b (%d)   |", RESET, UP, GRAYout, GRAYout
            );

        RESET = 1; UP = 0; #20;
        RESET = 0; #5;

        UP = 1; #70;

        UP = 0; #40;

        UP = 1; #30;
        $finish;
    end
endmodule
```

Output in Terminal

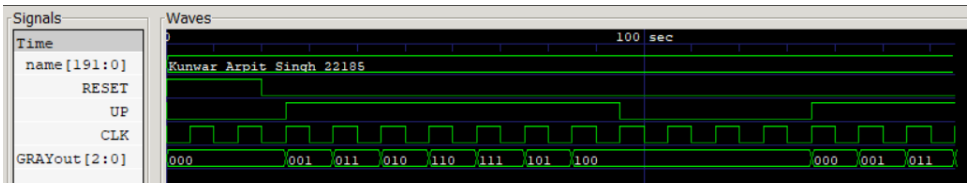
```
VCD info: dumpfile assign3_problem4_gray_code_counter.vcd opened for output.
Kunwar Arpit Singh
| Reset | UP | Gray Output |
----------------------------
| 1 | 0 | 000 (0) |
| 0 | 0 | 000 (0) |
| 0 | 1 | 001 (1) |
| 0 | 1 | 011 (3) |
| 0 | 1 | 010 (2) |
| 0 | 1 | 110 (6) |
| 0 | 1 | 111 (7) |
| 0 | 1 | 101 (5) |
| 0 | 1 | 100 (4) |
| 0 | 0 | 100 (4) |
| 0 | 1 | 000 (0) |
| 0 | 1 | 001 (1) |
| 0 | 1 | 011 (3) |
| 0 | 1 | 010 (2) |
```

Output Screenshots

Terminal output (Gray Code Counter)

```
VCD info: dumpfile assign3_problem4_gray_code_counter.vcd opened for output.
Kunwar Arpit Singh
| Reset | UP | Gray Output |
---------------------------
|   1   |  0 |    000 (0)  |
|   0   |  0 |    000 (0)  |
|   0   |  1 |    001 (1)  |
|   0   |  1 |    011 (3)  |
|   0   |  1 |    010 (2)  |
|   0   |  1 |    110 (6)  |
|   0   |  1 |    111 (7)  |
|   0   |  1 |    101 (5)  |
|   0   |  1 |    100 (4)  |
|   0   |  0 |    100 (4)  |
|   0   |  1 |    000 (0)  |
|   0   |  1 |    001 (1)  |
|   0   |  1 |    011 (3)  |
|   0   |  1 |    010 (2)  |
```

GTKWave waveform (Gray Code Counter)



5. Problem 5

Problem statement

Implement a verilog module of N-bit bidirectional shift registers using D ip op. (Hint: Use Parameterized Module to implement N).

Number	Gray code		
0	0	0	0
1	0	0	1
2	0	1	1
3	0	1	0
4	1	1	0
5	1	1	1
6	1	0	1
7	1	0	0

Verilog source (Sequential)

```
1  module shift_register #(parameter N = 8)(input clk, reset, input shift_en,
       load_en, input direction, input serial_in, input [N-1:0] parallel_in,
       output reg [N-1:0] parallel_out);
2
3      always @(posedge clk, posedge reset) begin
4          if (reset) begin
5              parallel_out <= {N{1'b0}};
6          end else if (load_en) begin
7              parallel_out <= parallel_in;
8          end else if (shift_en) begin
9              if (direction) begin
10                 parallel_out <= {serial_in, parallel_out[N-1:1]};
11             end else begin
12                 parallel_out <= {parallel_out[N-2:0], serial_in};
13             end
14         end
15     end
16
```

```
endmodule
```

Testbench

```
module shift_register_tb;
    localparam N = 8;
    reg clk;
    reg reset;
    reg shift_en, load_en, direction, serial_in;
    reg [N-1:0] parallel_in;
    wire [N-1:0] parallel_out;
    reg [8*24-1:0] name;
    shift_register #(.N(N)) dut (.clk(clk), .reset(reset), .shift_en(shift_en),
         .load_en(load_en), .direction(direction), .serial_in(serial_in), .
        parallel_in(parallel_in), .parallel_out(parallel_out)
    );

    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin
        name = " Kunwar Arpit Singh 22185";
        $dumpfile("assign3_problem5_bidirectional_shift_register.vcd");
        $dumpvars(1, shift_register_tb);
        $display("--- N-Bit Shift Register Simulation (N=%0d) ---", N);
        $display("Kunwar Arpit Singh");
        $display("| Reset | Load_en | Shift_en | Direction | Serial_in | 
            Parallel_out |");
        reset = 1; #20; reset = 0;
        $monitor("%b | %b | %b | %b | %b | %b", reset, load_en, shift_en,
            direction, serial_in, parallel_out);

        load_en = 1; parallel_in = 8'b10110101;
        #10;
        load_en = 0;
        #10;

        shift_en = 1; direction = 1;
        serial_in = 1;
        #10;
        serial_in = 0;
        #10;
        serial_in = 1;
        #10;

        direction = 0;
        serial_in = 0;
        #10;
```

```
        serial_in = 1;
        #10;
        serial_in = 0;
        #10;
        shift_en = 0;

        $finish;
    end
endmodule
```

Output in Terminal

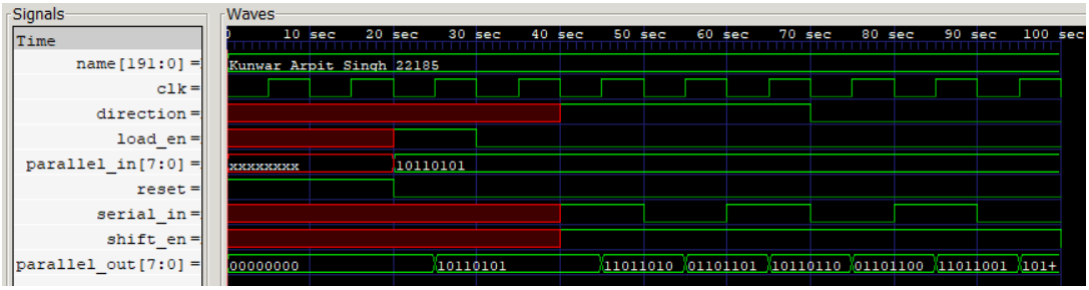
```
VCD info: dumpfile assign3_problem5_bidirectional_shift_register.vcd opened
    for output.
--- N-Bit Shift Register Simulation (N=8) ---
Kunwar Arpit Singh
| Reset | Load_en | Shift_en | Direction | Serial_in | Parallel_out |
0 | 1 | x | x | x | 00000000
0 | 1 | x | x | x | 10110101
0 | 0 | x | x | x | 10110101
0 | 0 | 1 | 1 | 1 | 10110101
0 | 0 | 1 | 1 | 1 | 11011010
0 | 0 | 1 | 1 | 0 | 11011010
0 | 0 | 1 | 1 | 0 | 01101101
0 | 0 | 1 | 1 | 1 | 01101101
0 | 0 | 1 | 1 | 1 | 10110110
0 | 0 | 1 | 0 | 0 | 10110110
0 | 0 | 1 | 0 | 0 | 01101100
0 | 0 | 1 | 0 | 1 | 01101100
0 | 0 | 1 | 0 | 1 | 11011001
0 | 0 | 1 | 0 | 0 | 11011001
0 | 0 | 1 | 0 | 0 | 10110010
0 | 0 | 0 | 0 | 0 | 10110010
```

Output Screenshots

Terminal output (Bidirectional Shift Register)

```
VCD info: dumpfile assign3_problem5_bidirectional_shift_register.vcd opened for output.
--- N-Bit Shift Register Simulation (N=8) ---
Kunwar Arpit Singh
| Reset | Load_en | Shift_en | Direction | Serial_in | Parallel_out |
0 | 1 | x | x | x | 00000000
0 | 1 | x | x | x | 10110101
0 | 0 | x | x | x | 10110101
0 | 0 | 1 | 1 | 1 | 10110101
0 | 0 | 1 | 1 | 1 | 11011010
0 | 0 | 1 | 1 | 0 | 11011010
0 | 0 | 1 | 1 | 0 | 01101101
0 | 0 | 1 | 1 | 1 | 01101101
0 | 0 | 1 | 1 | 1 | 10110110
0 | 0 | 1 | 0 | 0 | 10110110
0 | 0 | 1 | 0 | 0 | 01101100
0 | 0 | 1 | 0 | 1 | 01101100
0 | 0 | 1 | 0 | 1 | 11011001
0 | 0 | 1 | 0 | 0 | 11011001
0 | 0 | 1 | 0 | 0 | 10110010
0 | 0 | 0 | 0 | 0 | 10110010
```

GTKWave waveform (Bidirectional Shift Register)



6. Problem 6

Problem statement

Implement a single port memory of address space 128 and the addressability of 16-bit in Verilog. The memory has four inputs: clock, write enable, write address register, and read address register, and two outputs: read and write data register.

Verilog source (Behavioral)

```
module simple_dual_port_ram #(parameter DATA_WIDTH = 16, parameter ADDR_WIDTH
    = 7)( input clk, input we, input [ADDR_WIDTH-1:0] waddr, input [ADDR_WIDTH
    -1:0] raddr, input [DATA_WIDTH-1:0] wdata, output reg [DATA_WIDTH-1:0]
    rdata);

    localparam DEPTH = 1 << ADDR_WIDTH;

    reg [DATA_WIDTH-1:0] memory [0:DEPTH-1];

    always @(posedge clk) begin
        if (we) begin
            memory[waddr] <= wdata;
        end
        rdata <= memory[raddr];
    end

endmodule
```

Testbench

```
module simple_dual_port_ram_tb;

    localparam DATA_WIDTH = 16;
    localparam ADDR_WIDTH = 7;

    reg clk;
    reg we;
    reg [ADDR_WIDTH-1:0] waddr;
    reg [ADDR_WIDTH-1:0] raddr;
    reg [DATA_WIDTH-1:0] wdata;
    wire [DATA_WIDTH-1:0] rdata;
    reg [8*24-1:0] name;
    simple_dual_port_ram #(.DATA_WIDTH(DATA_WIDTH), .ADDR_WIDTH(ADDR_WIDTH))
        dut (.clk(clk), .we(we), .waddr(waddr), .raddr(raddr), .wdata(wdata), .
        rdata(rdata));

    initial begin
        clk = 0;
        forever #5 clk = ~clk; end
```

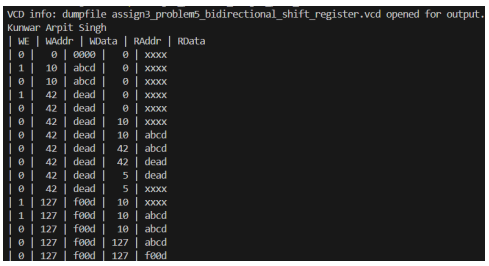
```
18
19      initial begin
20          name = "␣Kunwar␣Arpit␣Singh␣22185";
21          $dumpfile("assign3_problem5_bidirectional_shift_register.vcd");
22          $dumpvars(1, simple_dual_port_ram_tb);
23          $display("Kunwar␣Arpit␣Singh");
24          $display("|␣WE␣|␣WAddr␣|␣WData␣|␣RAddr␣|␣RData");
25          $monitor("|␣%b␣|␣%d␣|␣%h␣|␣%d␣|␣%h", we, waddr, wdata, raddr, rdata);
26
27          we = 0;
28          waddr = 0;
29          raddr = 0;
30          wdata = 0;
31          #10;
32
33          we = 1;
34          waddr = 10;
35          wdata = 16'hABCD;
36          #10;
37          we = 0;
38          #10;
39
40          we = 1;
41          waddr = 42;
42          wdata = 16'hDEAD;
43          #10;
44          we = 0;
45          #10;
46
47          raddr = 10;
48          #10;
49          raddr = 42;
50          #10;
51          raddr = 5;
52          #10;
53          we = 1;
54          waddr = 127;
55          wdata = 16'hF00D;
56          raddr = 10;
57          #10;
58          we = 0;
59          #10;
60
61          raddr = 127;
62          #10;
63
64          $finish;
65      end
66
67  endmodule
```

Output in Terminal

```
VCD info: dumpfile assign3_problem5_bidirectional_shift_register.vcd opened for output.
Kunwar Arpit Singh
| WE | WAddr | WData | RAddr | RData
| 0 | 0 | 0000 | 0 | xxxx
| 1 | 10 | abcd | 0 | xxxx
| 0 | 10 | abcd | 0 | xxxx
| 1 | 42 | dead | 0 | xxxx
| 0 | 42 | dead | 0 | xxxx
| 0 | 42 | dead | 10 | xxxx
| 0 | 42 | dead | 10 | abcd
| 0 | 42 | dead | 42 | abcd
| 0 | 42 | dead | 42 | dead
| 0 | 42 | dead | 5 | dead
| 0 | 42 | dead | 5 | xxxx
| 1 | 127 | f00d | 10 | xxxx
| 1 | 127 | f00d | 10 | abcd
| 0 | 127 | f00d | 10 | abcd
| 0 | 127 | f00d | 127 | abcd
| 0 | 127 | f00d | 127 | f00d
```

Output Screenshots

Terminal output (Single Port RAM)



GTKWave waveform (Single Port RAM)