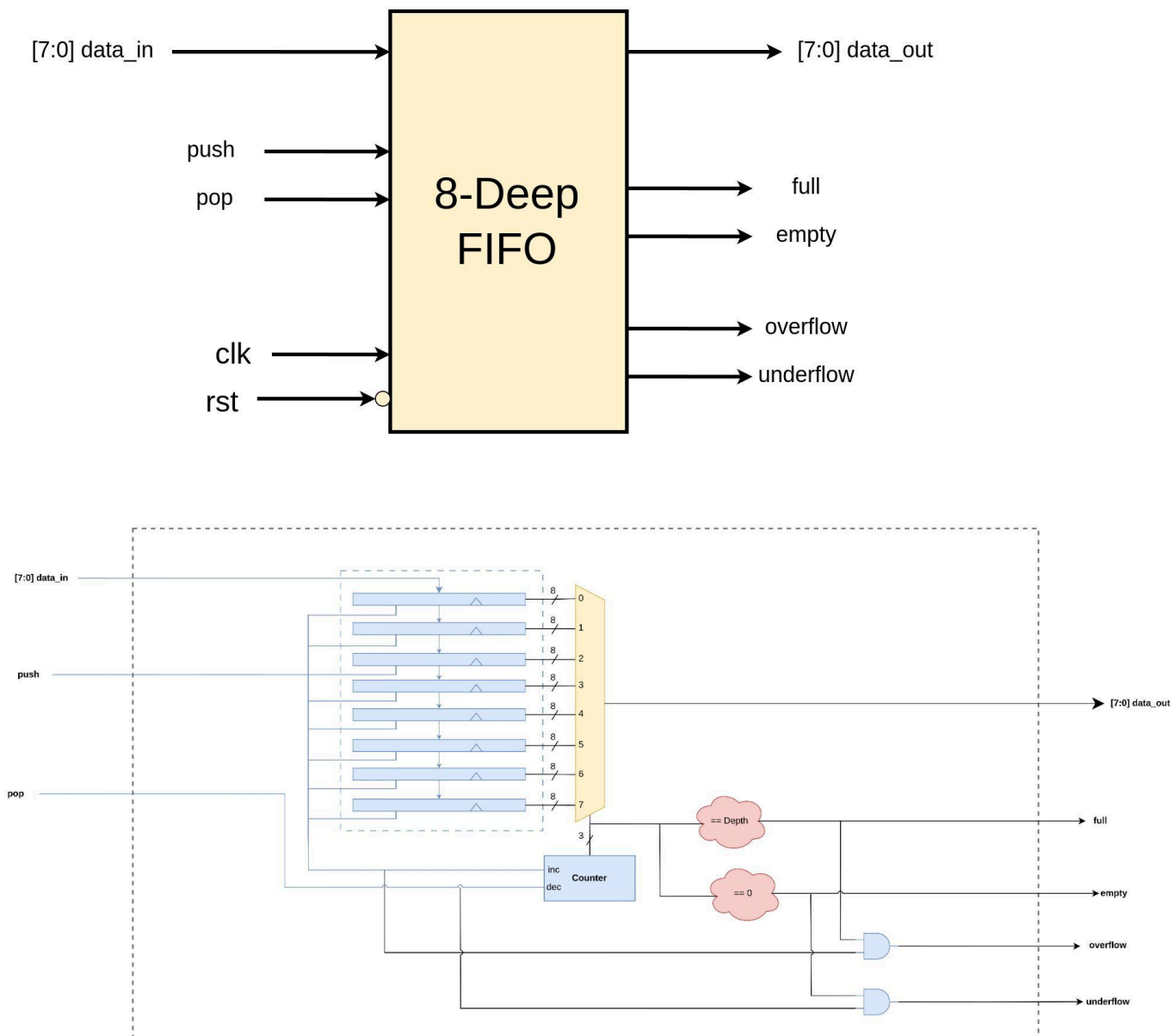
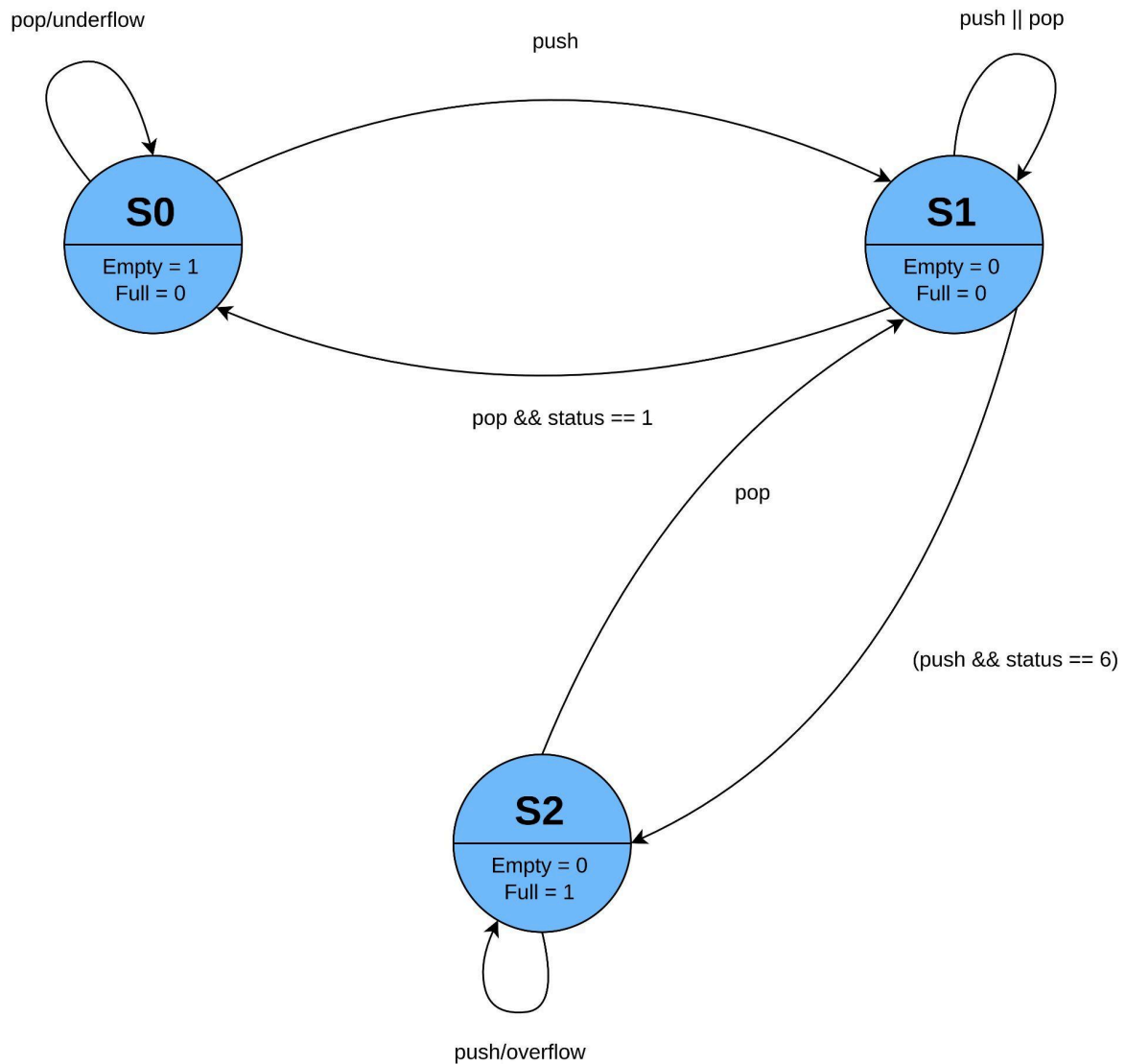


**Module: R3: DLD + DSD****Section: Sequential Circuits Task: Final Project****Design Problem****FIFO Design****➤ Question: Design an Asynchronous active low reset FIFO:****1. Schematic Diagram:****2. FSM Diagram:**



### 3. Verilog Code:

```

module fifo #(parameter DATA_SIZE = 8, parameter ADDRESS_SIZE = 3)
    (input clk, rst, push, pop,
     input [DATA_SIZE-1:0] data_in,
     output reg [DATA_SIZE-1:0] data_out,
     output reg full, empty,
     output reg overflow, underflow);

    localparam ADDRESS_DEPTH = 2**ADDRESS_SIZE;
    reg [DATA_SIZE-1:0] memory [ADDRESS_DEPTH-1:0];
    reg [ADDRESS_SIZE-1:0] read_ptr, write_ptr;
  
```

```

    reg [ADDRESS_SIZE:0] status; // Extra bit for distinguishing
full and empty

    //Empty & Full Logic
    assign empty = (status == 0);
    assign full = (status == ADDRESS_DEPTH);

    // Overflow & Underflow Logic
    assign overflow = (push && full);
    assign underflow = (pop && empty);

    // Sequential logic for write and pointer updates
    always @(posedge clk or negedge rst) begin
        if (!rst) begin
            // Reset logic
            read_ptr <= 0;
            write_ptr <= 0;
            status <= 0;
        end else begin
            if (push && pop) begin
                // If both push and pop occur at the same time
                if (!full && !empty)
                    begin
                        memory[write_ptr] <= data_in; // Perform write
                        write_ptr <= write_ptr + 1;
                        read_ptr <= read_ptr + 1;
                    end
                else if (full) begin
                    read_ptr <= read_ptr + 1;
                    status <= status - 1;
                end
                else if (empty) begin
                    memory[write_ptr] <= data_in;
                    write_ptr <= write_ptr + 1;
                    status <= status + 1;
                end
            end else if (push && !full) begin
                memory[write_ptr] <= data_in;
                write_ptr <= write_ptr + 1;
                status <= status + 1;
            end else if (pop && !empty) begin

```

```

        read_ptr <= read_ptr + 1;
        status <= status - 1;
    end
end
end

// Combinational logic for immediate read
always @(*) begin
    if (push && pop) begin
        if (!full && !empty)
            begin
                data_out = memory[read_ptr]; // Perform read
            end
        else if (full) begin
            data_out = memory[read_ptr];
        end
    end
    else if (pop && !empty ) begin
        data_out = memory[read_ptr]; // Perform read
    immediately
    end else begin
        data_out = 'bx; // X value when pop is not active or
FIFO is empty
    end
end
end

endmodule

```

#### 4. Testbench:

```

module tb_fifo;

    // Parameters
    parameter DATA_SIZE = 8;
    parameter ADDRESS_SIZE = 3;
    localparam ADDRESS_DEPTH = 2**ADDRESS_SIZE;

    // Signals
    reg clk, rst, push, pop;
    reg [DATA_SIZE-1:0] data_in;
    wire [DATA_SIZE-1:0] data_out;
    wire full, empty, overflow, underflow;

```

```
integer i;

// Instantiation
fifo #(DATA_SIZE, ADDRESS_SIZE) uut (
    .clk(clk),
    .rst(rst),
    .push(push),
    .pop(pop),
    .data_in(data_in),
    .data_out(data_out),
    .full(full),
    .empty(empty),
    .overflow(overflow),
    .underflow(underflow)
);

// Clock generation
always #5 clk = ~clk;

// Testbench logic
initial begin
    $dumpvars;

    // Initialize signals
    clk = 0;
    rst = 1;
    push = 0;
    pop = 0;
    data_in = 0;

    // Reset the FIFO
    @(posedge clk);
    rst = 0;
    @(posedge clk);
    rst = 1;
    @(posedge clk);

    // Test case: Push to full
    $display("Test case: Push to full");
    for (i = 0; i < ADDRESS_DEPTH; i = i + 1) begin
        @(posedge clk);
        push = 1;
    end
end
```

```

        data_in = i;
    end
    @(posedge clk);
    push = 0;
    @(negedge clk)
    if (!full) $display("Error: FIFO should be full");
    else $display("Passed");

    // Test case: Pop to empty
    $display("Test case: Pop to empty");
    for (i = 0; i < ADDRESS_DEPTH; i = i + 1) begin
        @(posedge clk);
        pop = 1;
    end
    @(posedge clk);
    pop = 0;
    @(negedge clk)
    if (!empty) $display("Error: FIFO should be empty");
    else $display("Passed");

    // Test case: Simultaneous push and pop
    for (i = 0; i < ADDRESS_DEPTH; i = i + 1) begin
        @(posedge clk);
        push = 1;
        pop = 1;
        data_in = i;
    end
    @(posedge clk);
    push = 0;
    pop = 0;

    // Test case: Overflow
    $display("Test case: Overflow");
    for (i = 0; i < ADDRESS_DEPTH; i = i + 1) begin
        @(posedge clk);
        push = 1;
        data_in = i;
    end
    @(posedge clk);
    push = 1;
    @(posedge clk);

```

```

        push = 0;
        if (!overflow) $display("Error: FIFO should have
overflowed");
        else $display("Passed");

        // Test case: Underflow
        $display("Test case: Underflow");
        for (i = 0; i < ADDRESS_DEPTH; i = i + 1) begin
            @(posedge clk);
            pop = 1;
        end
        @(posedge clk);
        pop = 1;
        @(posedge clk);
        pop = 0;
        if (!underflow) $display("Error: FIFO should have
underflowed");
        else $display("Passed\n");

        // End of test
        $display("Congratulations, All test cases are now
passing\n");
        $finish;
    end

endmodule

```

## 5. Output:

