Assignment 2

Ameer Ashraf Louly, G3

Q1:

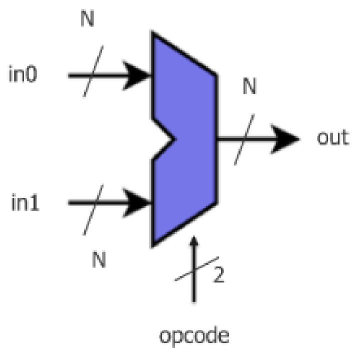
1) Implement a N-bit ALU with the following specifications

Parameters:

- N: defines the ports width (default is 4)
- OPCODE: defines the operation to be done (default is 0)

Ports:

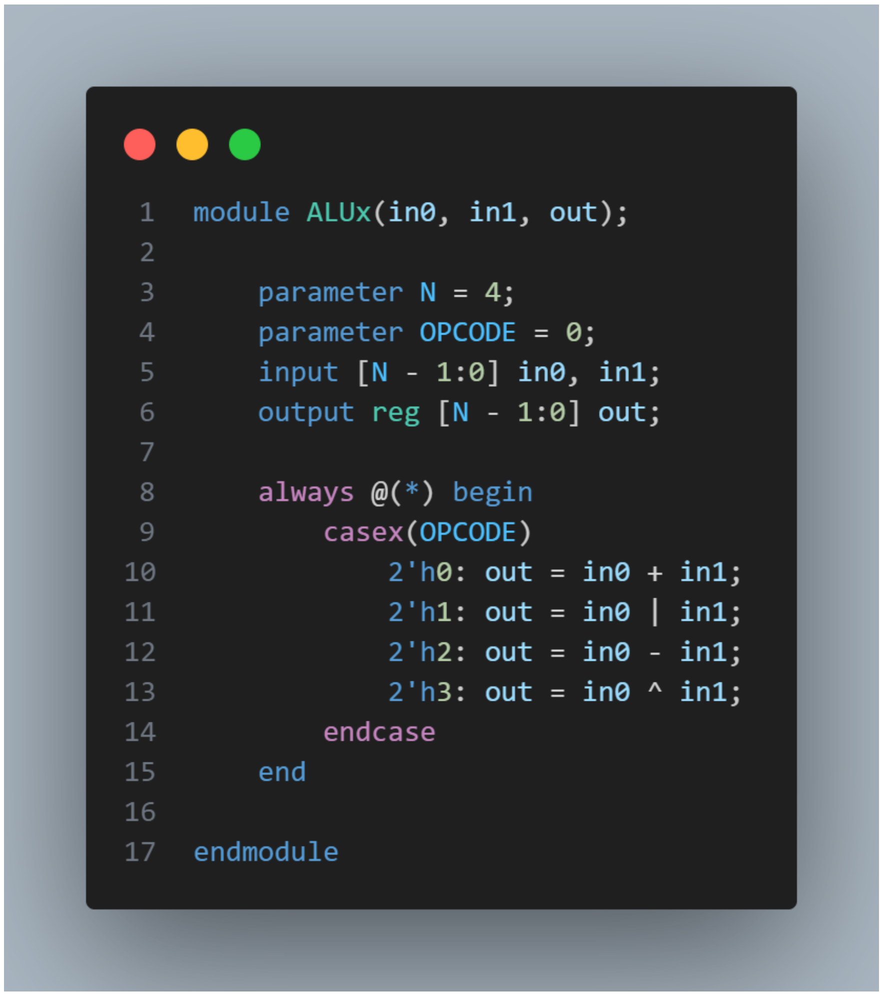
- In0 (Width: N)
- In1 (Width: N)
- Out (Width: N)



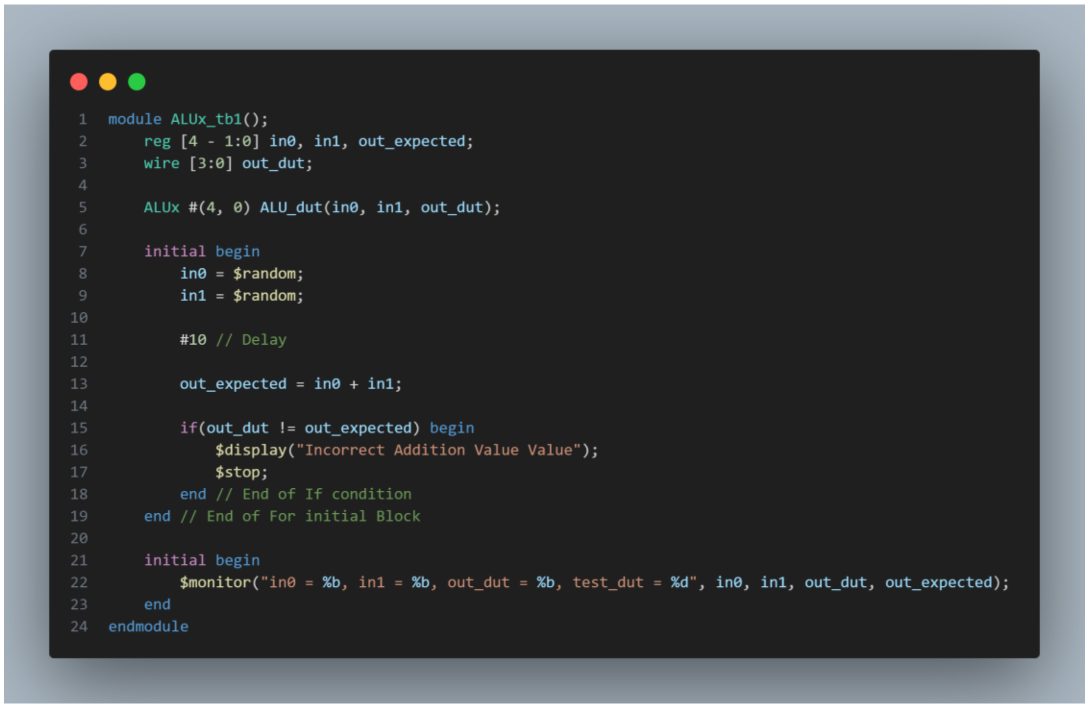
Inputs		Outputs
Opcode		Operation
0	0	Addition
0	1	OR
1	0	Subtraction (in0-in1)
1	1	XOR

After designing the ALU, create 4 testbenches where each testbench will verify different parameter configurations as follows:

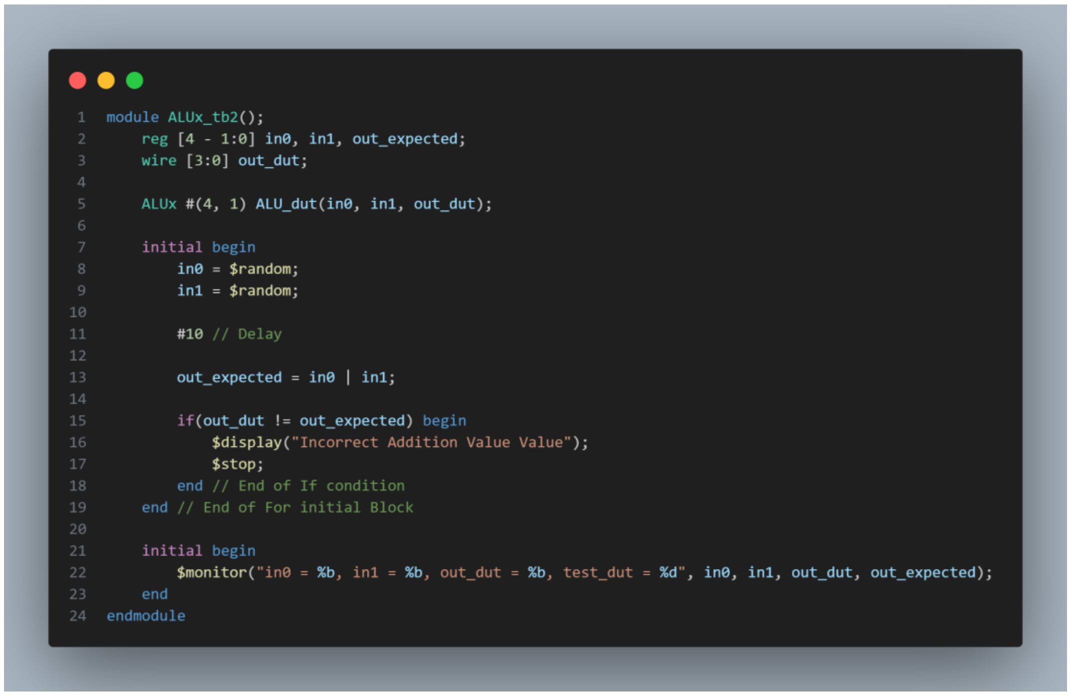
- Testbench 1: N=4 & OPCODE = 0
- Testbench 1: N=4 & OPCODE = 1
- Testbench 1: N=4 & OPCODE = 2
- Testbench 1: N=4 & OPCODE = 3

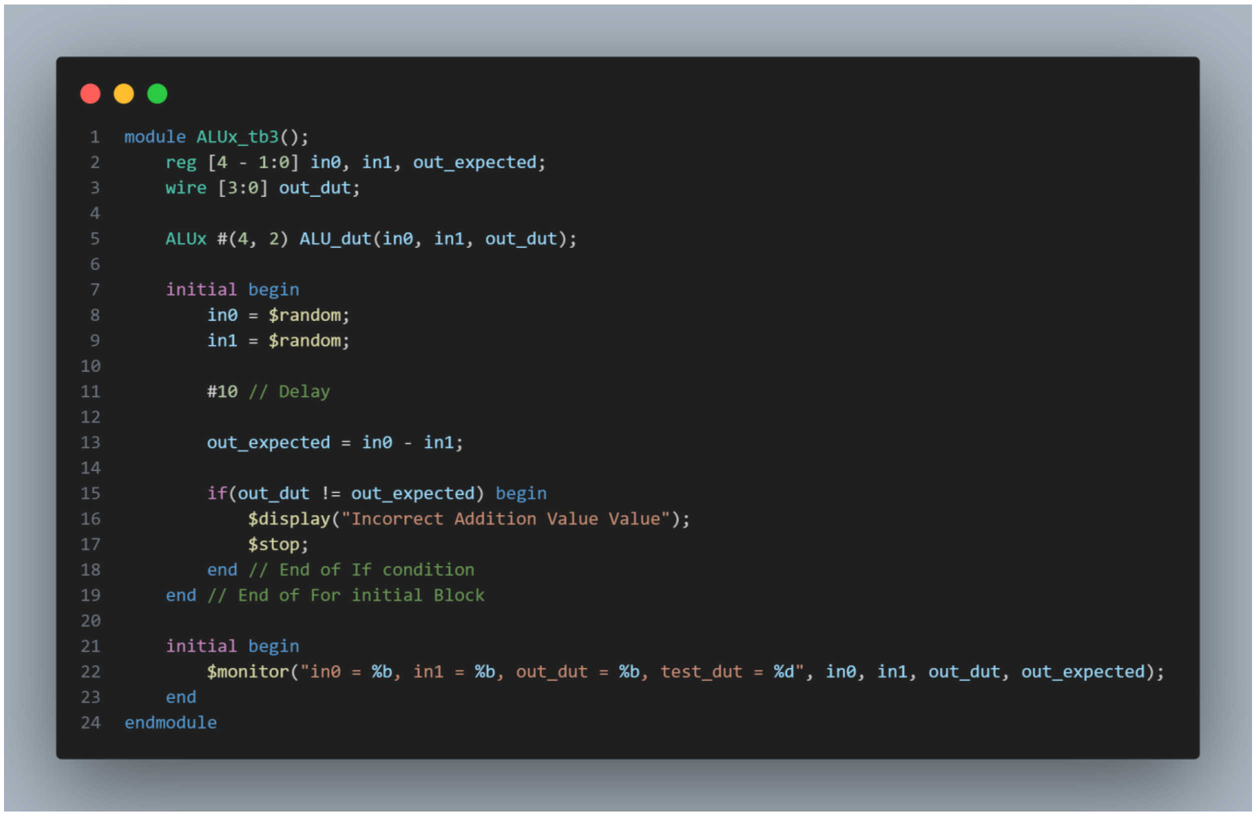


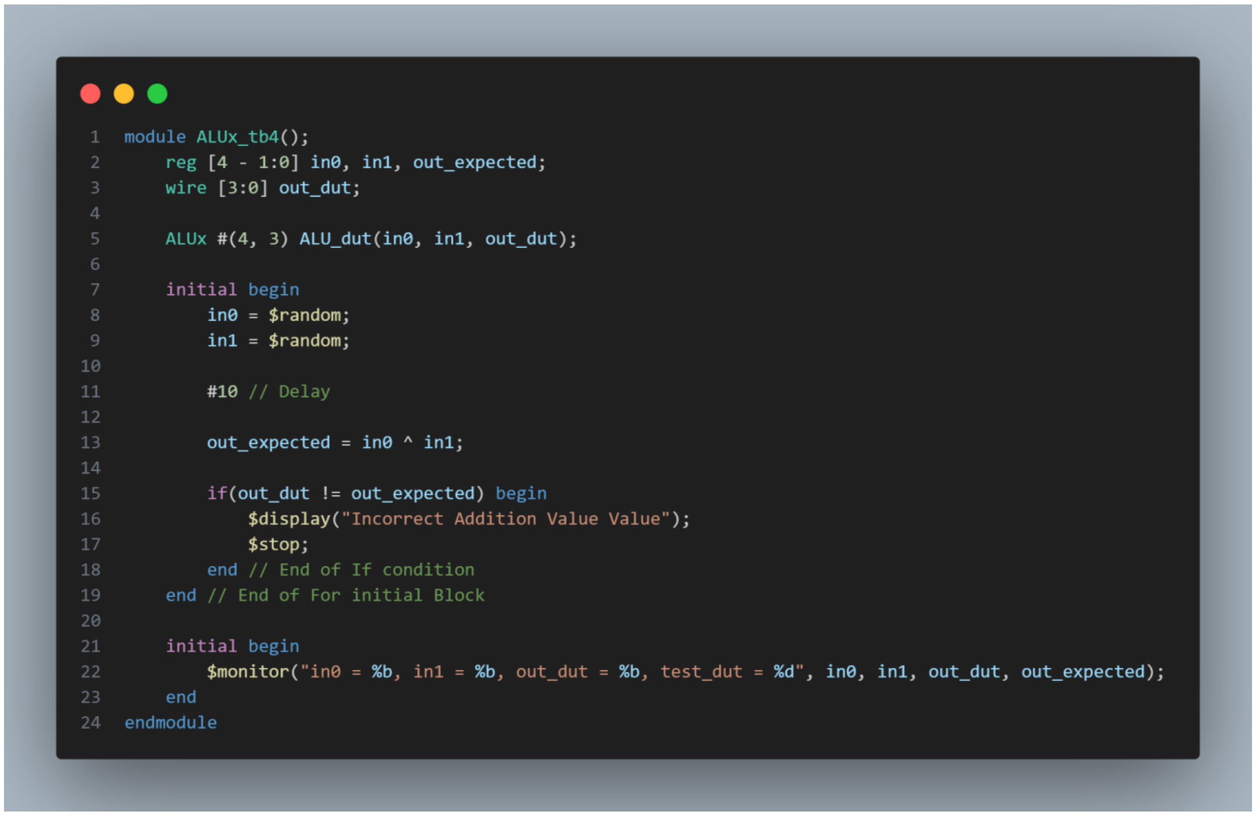
```verilog
module ALUx(in0, in1, out);

    parameter N = 4;
    parameter OPCODE = 0;
    input [N - 1:0] in0, in1;
    output reg [N - 1:0] out;

    always @(*) begin
        casex(OPCODE)
            2'h0: out = in0 + in1;
            2'h1: out = in0 | in1;
            2'h2: out = in0 - in1;
            2'h3: out = in0 ^ in1;
        endcase
    end

endmodule
```



```verilog
module ALUx_tb1();
    reg [4 - 1:0] in0, in1, out_expected;
    wire [3:0] out_dut;

    ALUx #(4, 0) ALU_dut(in0, in1, out_dut);

    initial begin
        in0 = $random;
        in1 = $random;

        #10 // Delay

        out_expected = in0 + in1;

        if(out_dut != out_expected) begin
            $display("Incorrect Addition Value Value");
            $stop;
        end // End of If condition
    end // End of For initial Block

    initial begin
        $monitor("in0 = %b, in1 = %b, out_dut = %b, test_dut = %d", in0, in1, out_dut, out_expected);
    end
endmodule
```


```verilog
module ALUx_tb2();
    reg [4 - 1:0] in0, in1, out_expected;
    wire [3:0] out_dut;

    ALUx #(4, 1) ALU_dut(in0, in1, out_dut);

    initial begin
        in0 = $random;
        in1 = $random;

        #10 // Delay

        out_expected = in0 | in1;

        if(out_dut != out_expected) begin
            $display("Incorrect Addition Value Value");
            $stop;
        end // End of If condition
    end // End of For initial Block

    initial begin
        $monitor("in0 = %b, in1 = %b, out_dut = %b, test_dut = %d", in0, in1, out_dut, out_expected);
    end
endmodule
```


```verilog
module ALUx_tb3();
    reg [4 - 1:0] in0, in1, out_expected;
    wire [3:0] out_dut;

    ALUx #(4, 2) ALU_dut(in0, in1, out_dut);

    initial begin
        in0 = $random;
        in1 = $random;

        #10 // Delay

        out_expected = in0 - in1;

        if(out_dut != out_expected) begin
            $display("Incorrect Addition Value Value");
            $stop;
        end // End of If condition
    end // End of For initial Block

    initial begin
        $monitor("in0 = %b, in1 = %b, out_dut = %b, test_dut = %d", in0, in1, out_dut, out_expected);
    end
endmodule
```



```
module ALUx_tb4();
    reg [4 - 1:0] in0, in1, out_expected;
    wire [3:0] out_dut;

    ALUx #(4, 3) ALU_dut(in0, in1, out_dut);

    initial begin
        in0 = $random;
        in1 = $random;

        #10 // Delay

        out_expected = in0 ^ in1;

        if(out_dut != out_expected) begin
            $display("Incorrect Addition Value Value");
            $stop;
        end // End of If condition
    end // End of For initial Block

    initial begin
        $monitor("in0 = %b, in1 = %b, out_dut = %b, test_dut = %d", in0, in1, out_dut, out_expected);
    end
endmodule
```

Q2:

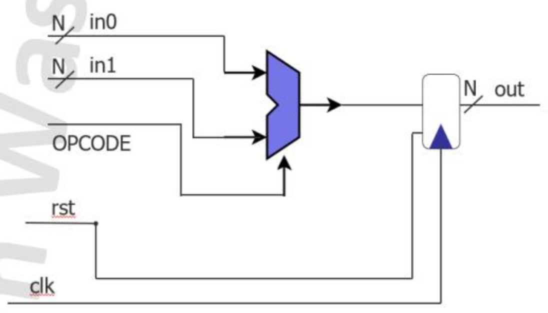
2) Modify on question 1 design so that the output “out” is evaluated with the positive edge of a register as follows

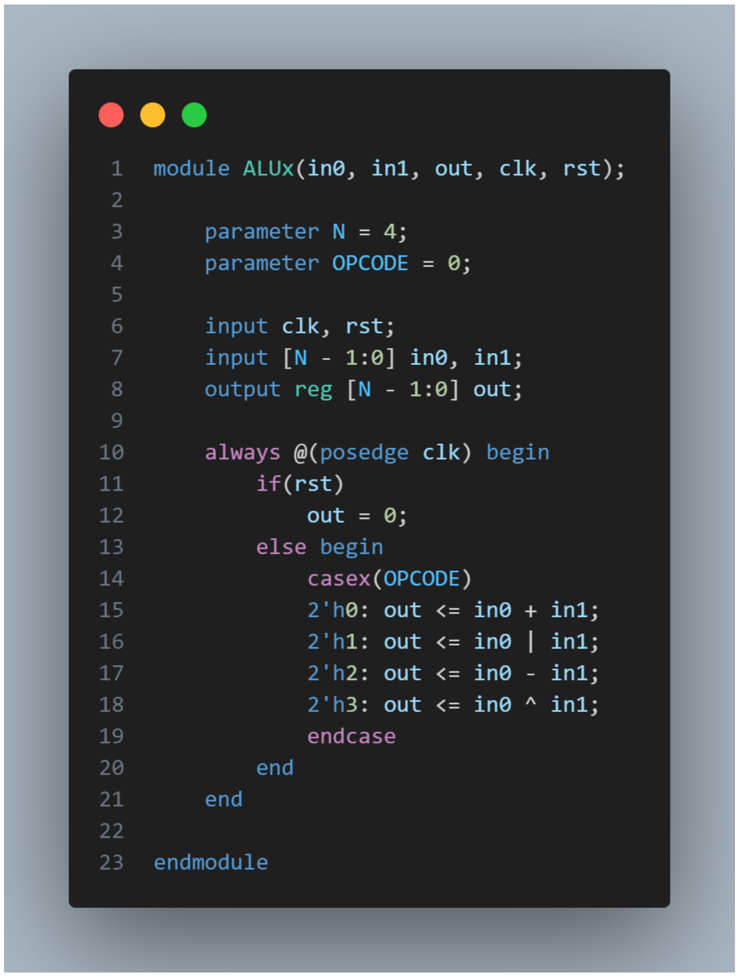
Parameters:

- N: defines the ports width (default is 4)
- OPCODE: defines the operation to be done (default is 0)

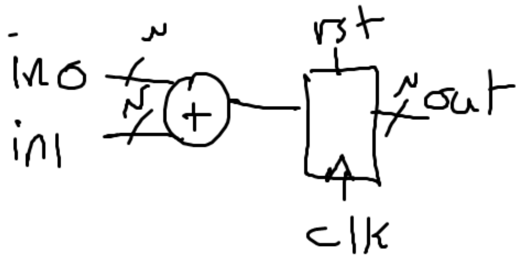
Ports:

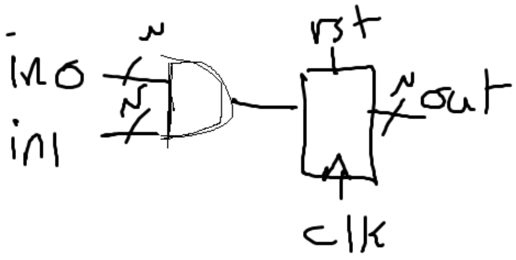
- In0 (Width: N)
- In1 (Width: N)
- Out (Width: N)
- Clk (new port added)
- Rst (new port added)
  - Active high synchronous reset

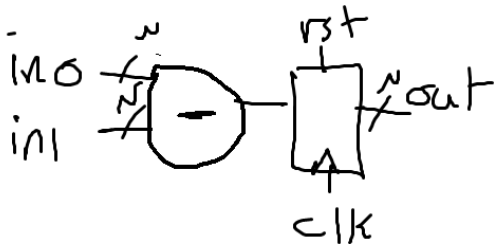


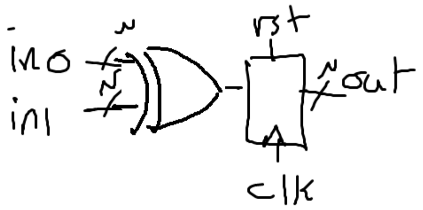


```
module ALUx(in0, in1, out, clk, rst);

    parameter N = 4;
    parameter OPCODE = 0;

    input clk, rst;
    input [N - 1:0] in0, in1;
    output reg [N - 1:0] out;

    always @(posedge clk) begin
        if(rst)
            out = 0;
        else begin
            casex(OPCODE)
                2'h0: out <= in0 + in1;
                2'h1: out <= in0 | in1;
                2'h2: out <= in0 - in1;
                2'h3: out <= in0 ^ in1;
            endcase
        end
    end

endmodule
```



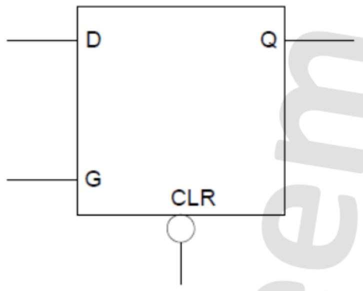






Q3:

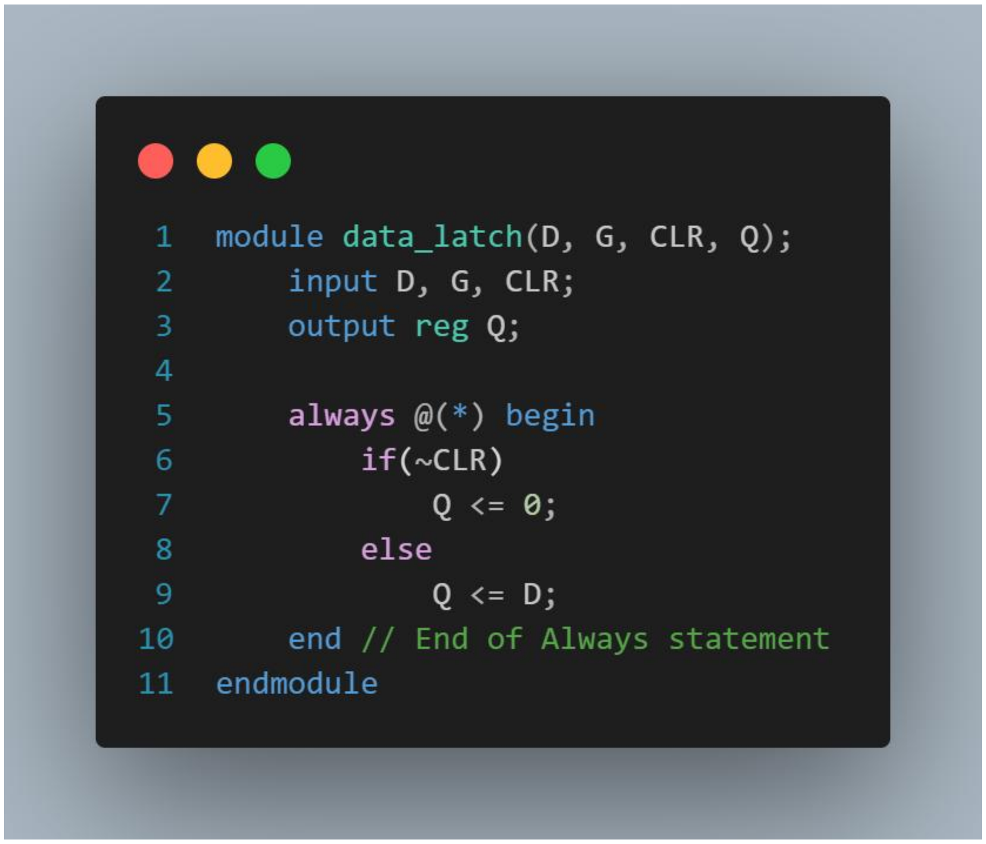
3) Implement Data Latch with active low Clear. No testbench is needed.



Input	Output
CLR, D, G	Q

Truth Table

CLR	G	D	Q
0	X	X	0
1	0	X	Q
1	1	D	D



```verilog
module data_latch(D, G, CLR, Q);
    input D, G, CLR;
    output reg Q;

    always @(*) begin
        if(~CLR)
            Q <= 0;
        else
            Q <= D;
    end // End of Always statement
endmodule
```

Q4:

4) Implement the following latch as specified below. No testbench is needed.

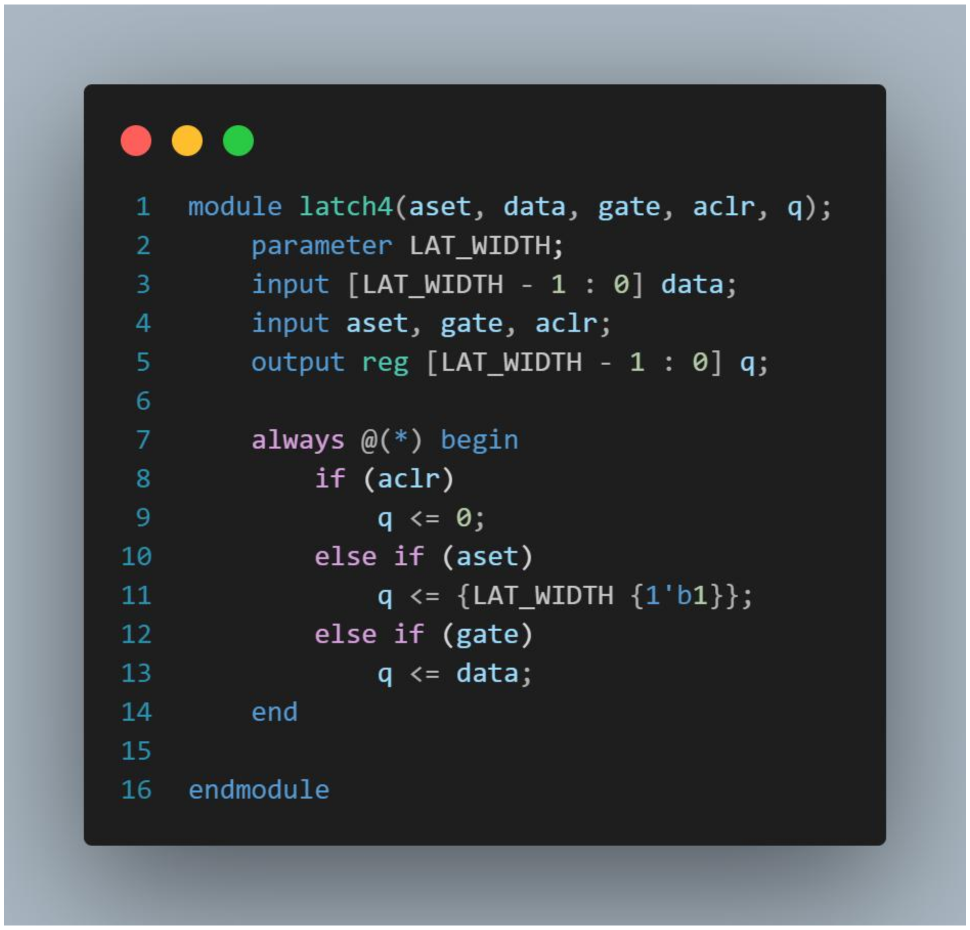
Parameters

LAT_WIDTH: Determine the width of input data and output q

Ports

Name	Type	Description
aset	Input	Asynchronous set input. Sets q[] output to 1.
data[]		Data Input to the D-type latch with width LAT_WIDTH
gate		Latch enable input
aclr		Asynchronous clear input. Sets q[] output to 0.
q[]	Output	Data output from the latch with with LAT_WIDTH

If both aset and aclr are both asserted, aclr is dominant. aset signals sets **all** output bits to 1.



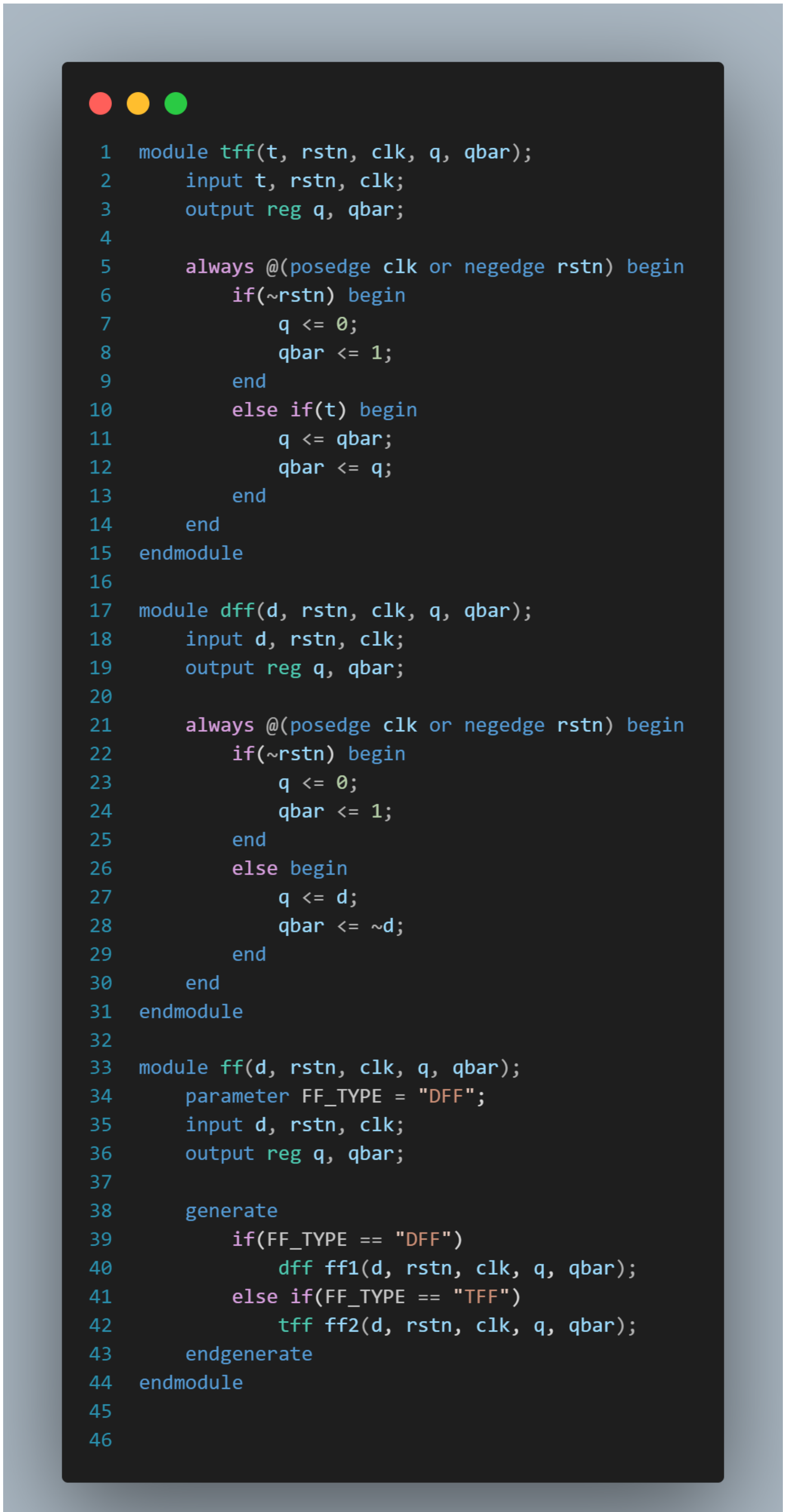
```verilog
module latch4(aset, data, gate, aclr, q);
    parameter LAT_WIDTH;
    input [LAT_WIDTH - 1 : 0] data;
    input aset, gate, aclr;
    output reg [LAT_WIDTH - 1 : 0] q;

    always @(*) begin
        if (aclr)
            q <= 0;
        else if (aset)
            q <= {LAT_WIDTH {1'b1}};
        else if (gate)
            q <= data;
    end

endmodule
```

Q5:

A. Implement T-type (toggle) Flipflop with active low asynchronous reset (forces q to 0 and forces qbar to 1). T-Flipflop has input t, when t input is high the outputs toggle else the output values do not change. No testbench is needed.
   - Inputs: t, rstn, clk
   - Outputs: q, qbar

B. Implement Asynchronous D Flip-Flop with Active low reset (forces q to 0 and forces qbar to 1). No testbench is needed.
   - Inputs: d, rstn, clk
   - Outputs: q, qbar

C. Implement a parameterized asynchronous FlipFlop with Active low reset with the following specifications.
   - Inputs: d, rstn , clk
   - Outputs: q, qbar
   - Parameter: FF_TYPE that can take two valid values, DFF or TFF. Default value = “DFF”. Design should act as DFF if FF_TYPE = “DFF” and act as TFF if FF_TYPE = “TFF”. When FF_TYPE equals “DFF”, d input acts as the data input “d”, and when FF_TYPE equals “TFF”, d input acts the toggle input “t”.
   - No testbench is needed.



```verilog
module tff(t, rstn, clk, q, qbar);
    input t, rstn, clk;
    output reg q, qbar;

    always @(posedge clk or negedge rstn) begin
        if(~rstn) begin
            q <= 0;
            qbar <= 1;
        end
        else if(t) begin
            q <= qbar;
            qbar <= q;
        end
    end
endmodule

module dff(d, rstn, clk, q, qbar);
    input d, rstn, clk;
    output reg q, qbar;

    always @(posedge clk or negedge rstn) begin
        if(~rstn) begin
            q <= 0;
            qbar <= 1;
        end
        else begin
            q <= d;
            qbar <= ~d;
        end
    end
endmodule

module ff(d, rstn, clk, q, qbar);
    parameter FF_TYPE = "DFF";
    input d, rstn, clk;
    output reg q, qbar;

    generate
        if(FF_TYPE == "DFF")
            dff ff1(d, rstn, clk, q, qbar);
        else if(FF_TYPE == "TFF")
            tff ff2(d, rstn, clk, q, qbar);
    endgenerate
endmodule


```