Anika Haque, 20283

San Francisco Bay University

EE461 Digital Design and HDL
Week#5 Sequential Logic

1. Lab Outlines:

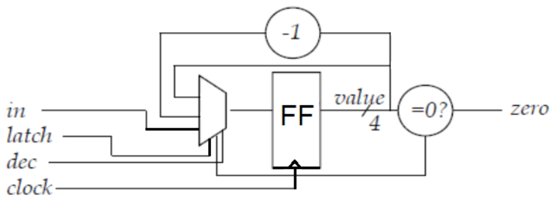
1. DFF & Latch modeling in Verilog
2. Sequential logic feedback
3. Not glue logic in the top level
4. Race condition again
5. Exercises

2. Lab Procedures

I. DFF & Latch modeling in Verilog

```
`timescale 1ns/100ps
module xor3(input D_i, clock_i, output Q_o);
       reg Q_o;

       always@(clock_i or D_i)
              if (clock_i) Q _o<=D_i;
endmodule

`timescale 1ns/100ps
module xor3 (input a_i, b_i, c_i, clock_i, output Q_o);
       reg Q_o;

       always@(posedge clock_i)
              if (a_i) Q_o< = c_i;
              else Q_o<= b_i;
endmodule
```

II. Sequential logic feedback

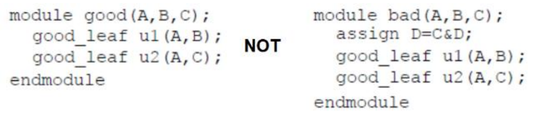


```
/* If latch is high, value will get 'in'; if dec is high and zero
isn't 0, value is to decrease 1; if value is 0, zero is set to 1.
*/
module counter (clock, in, latch, dec, zero);
       input clock;        /* clock */
       input [3:0] in;     /* starting count */
       input latch;        /* latch `in' when high */
       input dec;          /* decrement count when dec high */
       output zero;        /* high when count down to zero */

       reg [3:0] value;    /* current count value */

       always@(posedge clock) begin
              if (latch) value <= in;
              else if (dec && !zero) value <= value - 1'b1;
       end
```

```
        assign zero = (value == 4'b0);
    endmodule /* counter */
```

III. Not glue logic in the top level

- Generally, it is a good idea to only implement logic in the leaf cells of a hierarchical design, and not at a higher level.



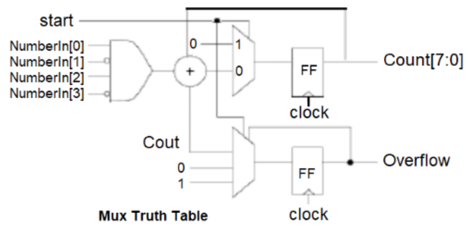
```
module good(A,B,C);                 module bad(A,B,C);
  good_leaf u1(A,B);        NOT       assign D=C&D;
  good_leaf u2(A,C);                  good_leaf u1(A,B);
endmodule                             good_leaf u2(A,C);
                                    endmodule
```

IV. Race condition Again

- Guideline 1: When modeling sequential logic, use nonblocking assignments.
- Guideline 2: When modeling combinational logic with an "always" block, use blocking assignments.
- Guideline 3: Do not mix blocking and nonblocking assignments in the same "always" block.
- Guideline 4: Do not make assignments to the same variable from more than one "always" block.
- Guideline 5: Do not make assignments using #0 delays.

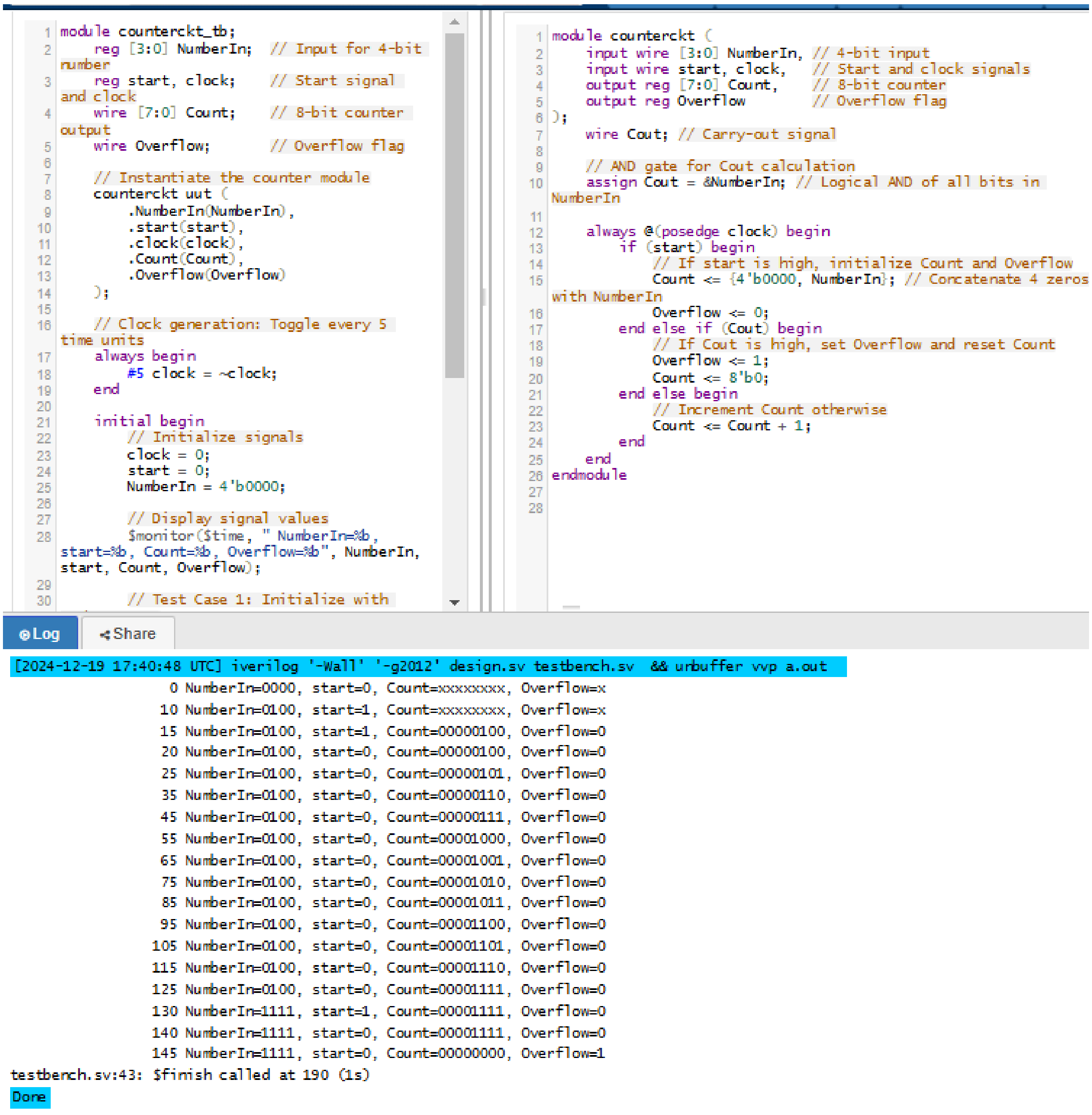
V Exercises

- Convert the following circuit into Verilog module and write the testbench to verify the design.



Mux Truth Table

Start	Overflow:		Output
0	0	:	Cout
0	1	:	1
1	0	:	0
1	1	:	0



```verilog
module counterckt_tb;
    reg [3:0] NumberIn;   // Input for 4-bit number
    reg start, clock;     // Start signal and clock
    wire [7:0] Count;     // 8-bit counter output
    wire Overflow;        // Overflow flag

    // Instantiate the counter module
    counterckt uut (
        .NumberIn(NumberIn),
        .start(start),
        .clock(clock),
        .Count(Count),
        .Overflow(Overflow)
    );

    // Clock generation: Toggle every 5 time units
    always begin
        #5 clock = ~clock;
    end

    initial begin
        // Initialize signals
        clock = 0;
        start = 0;
        NumberIn = 4'b0000;

        // Display signal values
        $monitor($time, " NumberIn=%b, start=%b, Count=%b, Overflow=%b", NumberIn, start, Count, Overflow);

        // Test Case 1: Initialize with
```

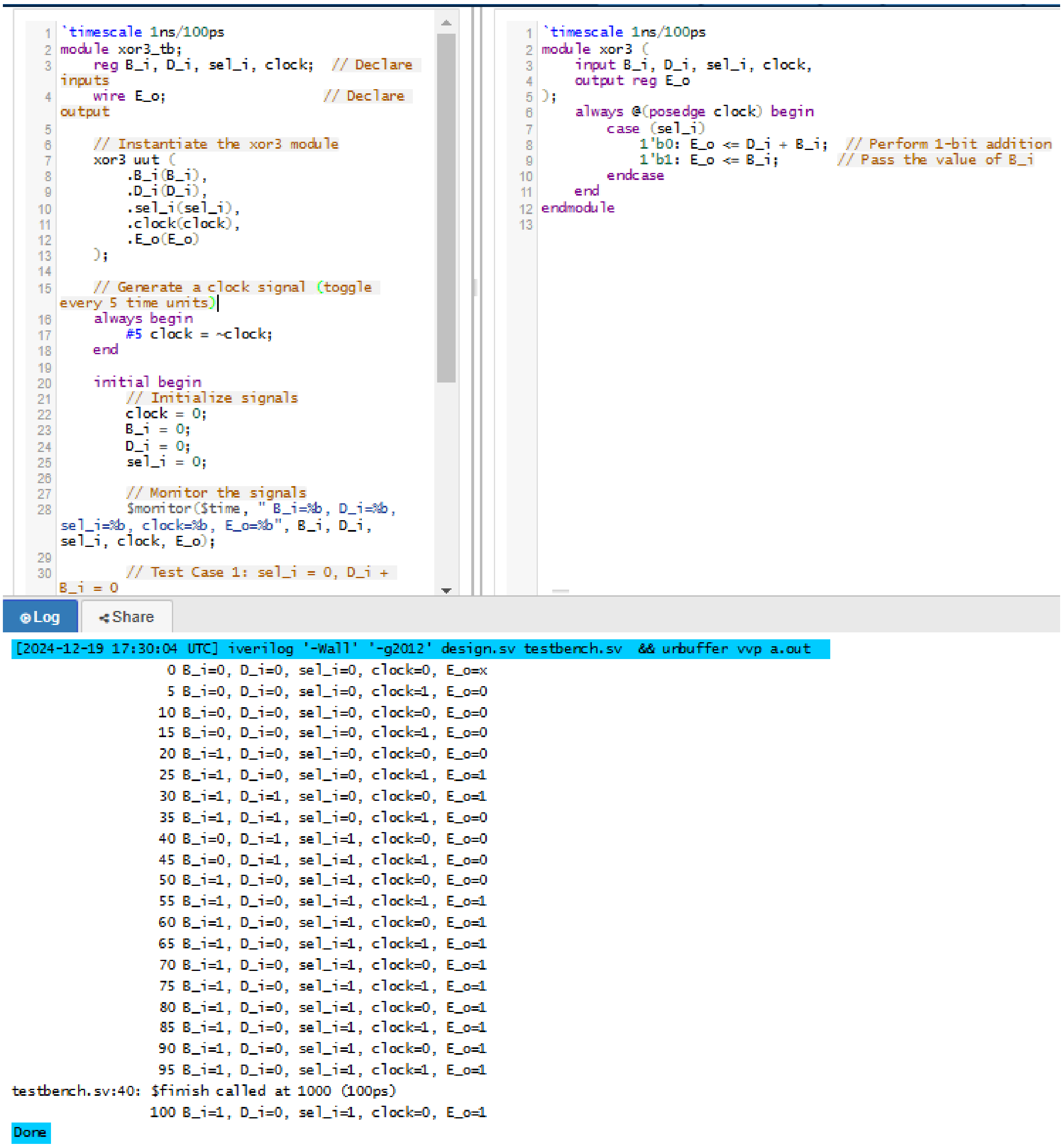
```verilog
module counterckt (
    input wire [3:0] NumberIn, // 4-bit input
    input wire start, clock,   // Start and clock signals
    output reg [7:0] Count,    // 8-bit counter
    output reg Overflow        // Overflow flag
);
    wire Cout; // Carry-out signal

    // AND gate for Cout calculation
    assign Cout = &NumberIn; // Logical AND of all bits in NumberIn

    always @(posedge clock) begin
        if (start) begin
            // If start is high, initialize Count and Overflow
            Count <= {4'b0000, NumberIn}; // Concatenate 4 zeros with NumberIn
            Overflow <= 0;
        end else if (Cout) begin
            // If Cout is high, set Overflow and reset Count
            Overflow <= 1;
            Count <= 8'b0;
        end else begin
            // Increment Count otherwise
            Count <= Count + 1;
        end
    end
endmodule
```

Log   Share

```
[2024-12-19 17:40:48 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
                   0 NumberIn=0000, start=0, Count=xxxxxxxx, Overflow=x
                  10 NumberIn=0100, start=1, Count=xxxxxxxx, Overflow=x
                  15 NumberIn=0100, start=1, Count=00000100, Overflow=0
                  20 NumberIn=0100, start=0, Count=00000100, Overflow=0
                  25 NumberIn=0100, start=0, Count=00000101, Overflow=0
                  35 NumberIn=0100, start=0, Count=00000110, Overflow=0
                  45 NumberIn=0100, start=0, Count=00000111, Overflow=0
                  55 NumberIn=0100, start=0, Count=00001000, Overflow=0
                  65 NumberIn=0100, start=0, Count=00001001, Overflow=0
                  75 NumberIn=0100, start=0, Count=00001010, Overflow=0
                  85 NumberIn=0100, start=0, Count=00001011, Overflow=0
                  95 NumberIn=0100, start=0, Count=00001100, Overflow=0
                 105 NumberIn=0100, start=0, Count=00001101, Overflow=0
                 115 NumberIn=0100, start=0, Count=00001110, Overflow=0
                 125 NumberIn=0100, start=0, Count=00001111, Overflow=0
                 130 NumberIn=1111, start=1, Count=00001111, Overflow=0
                 140 NumberIn=1111, start=0, Count=00001111, Overflow=0
                 145 NumberIn=1111, start=0, Count=00000000, Overflow=1
testbench.sv:43: $finish called at 190 (1s)
Done
```

- What does logic statement specify for the hardware in the module?

```
`timescale 1ns/100ps
   module xor3 (input B_i, D_i, sel_i, clock, output E_o);
              reg E_o;

              always@(posedge clock) begin
                     case(sel_i)
                            0: E_o <= D_i + B_i;
                            1: E_o <= B_i;
                     endcase
              end
   endmodule
```



```
`timescale 1ns/100ps
module xor3_tb;
    reg B_i, D_i, sel_i, clock;  // Declare inputs
    wire E_o;                   // Declare output

    // Instantiate the xor3 module
    xor3 uut (
        .B_i(B_i),
        .D_i(D_i),
        .sel_i(sel_i),
        .clock(clock),
        .E_o(E_o)
    );

    // Generate a clock signal (toggle every 5 time units)
    always begin
        #5 clock = ~clock;
    end

    initial begin
        // Initialize signals
        clock = 0;
        B_i = 0;
        D_i = 0;
        sel_i = 0;

        // Monitor the signals
        $monitor($time, " B_i=%b, D_i=%b, sel_i=%b, clock=%b, E_o=%b", B_i, D_i, sel_i, clock, E_o);

        // Test Case 1: sel_i = 0, D_i + B_i = 0
```

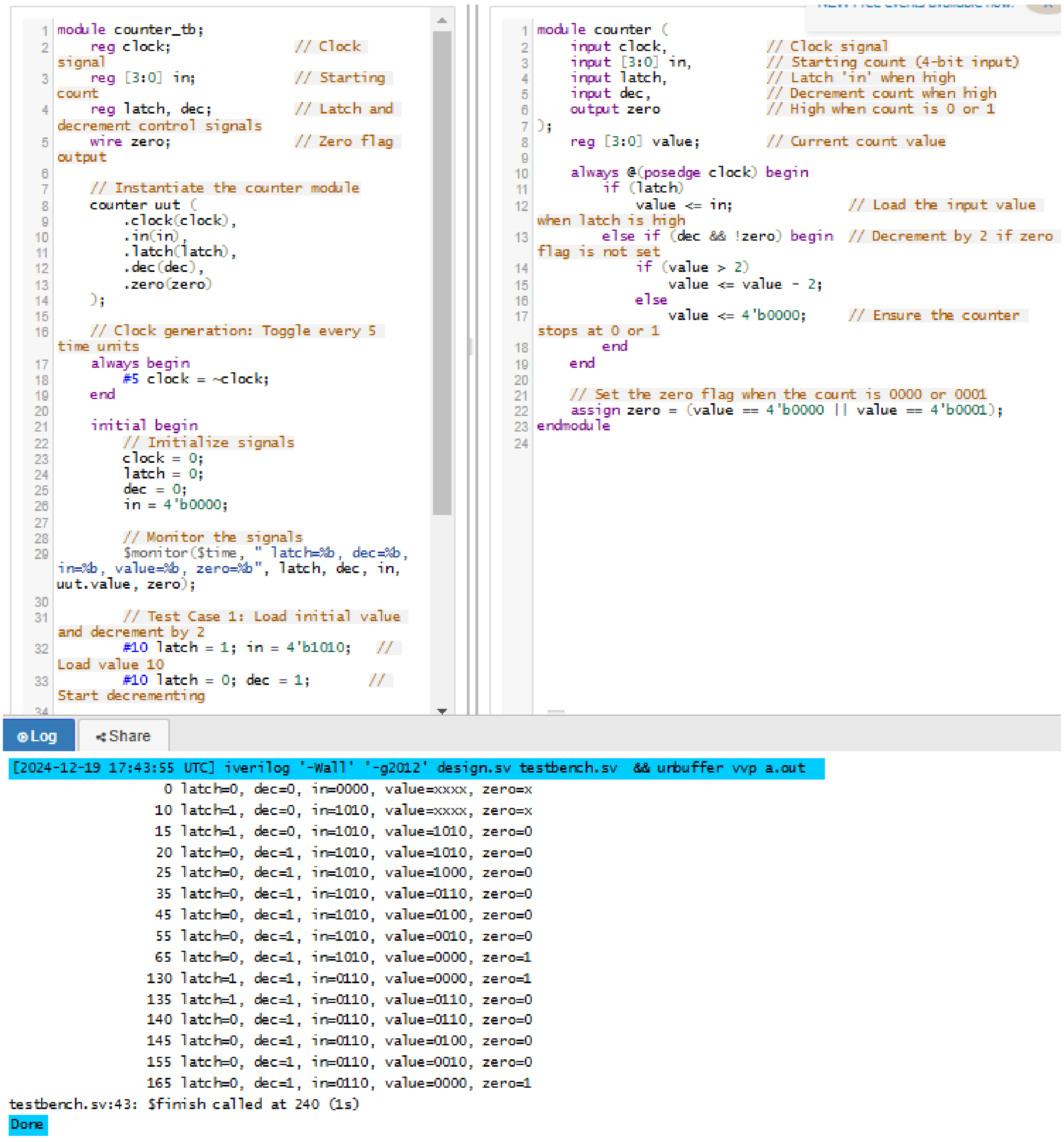
```
`timescale 1ns/100ps
module xor3 (
    input B_i, D_i, sel_i, clock,
    output reg E_o
);
    always @(posedge clock) begin
        case (sel_i)
            1'b0: E_o <= D_i + B_i;  // Perform 1-bit addition
            1'b1: E_o <= B_i;        // Pass the value of B_i
        endcase
    end
endmodule
```

```
[2024-12-19 17:30:04 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
                    0 B_i=0, D_i=0, sel_i=0, clock=0, E_o=x
                    5 B_i=0, D_i=0, sel_i=0, clock=1, E_o=0
                   10 B_i=0, D_i=0, sel_i=0, clock=0, E_o=0
                   15 B_i=0, D_i=0, sel_i=0, clock=1, E_o=0
                   20 B_i=1, D_i=0, sel_i=0, clock=0, E_o=0
                   25 B_i=1, D_i=0, sel_i=0, clock=1, E_o=1
                   30 B_i=1, D_i=1, sel_i=0, clock=0, E_o=1
                   35 B_i=1, D_i=1, sel_i=0, clock=1, E_o=0
                   40 B_i=0, D_i=1, sel_i=1, clock=0, E_o=0
                   45 B_i=0, D_i=1, sel_i=1, clock=1, E_o=0
                   50 B_i=1, D_i=0, sel_i=1, clock=0, E_o=0
                   55 B_i=1, D_i=0, sel_i=1, clock=1, E_o=1
                   60 B_i=1, D_i=0, sel_i=1, clock=0, E_o=1
                   65 B_i=1, D_i=0, sel_i=1, clock=1, E_o=1
                   70 B_i=1, D_i=0, sel_i=1, clock=0, E_o=1
                   75 B_i=1, D_i=0, sel_i=1, clock=1, E_o=1
                   80 B_i=1, D_i=0, sel_i=1, clock=0, E_o=1
                   85 B_i=1, D_i=0, sel_i=1, clock=1, E_o=1
                   90 B_i=1, D_i=0, sel_i=1, clock=0, E_o=1
                   95 B_i=1, D_i=0, sel_i=1, clock=1, E_o=1
testbench.sv:40: $finish called at 1000 (100ps)
                  100 B_i=1, D_i=0, sel_i=1, clock=0, E_o=1
Done
```

The logic specifies a multiplexer-based sequential circuit with a 1-bit full adder. The sel_i signal determines whether the output is the sum of D_i and B_i or simply B_i. It has the following hardware components:

1. 1-bit Full Adder: For the D_i + B_i operation.
2. 2-to-1 Multiplexer: To choose between D_i + B_i and B_i.
3. Flip-Flop: To store the value of E_o and ensure it updates on the rising clock edge.

- Modify design in **Lab Procedures II** so it is decremented by *two,* and then stops, setting zero flag high, when it reaches 0000 or 0001. After that, verify the design in the testbench.



```verilog
module counter_tb;
    reg clock;              // Clock signal
    reg [3:0] in;           // Starting count
    reg latch, dec;         // Latch and decrement control signals
    wire zero;              // Zero flag output

    // Instantiate the counter module
    counter uut (
        .clock(clock),
        .in(in),
        .latch(latch),
        .dec(dec),
        .zero(zero)
    );

    // Clock generation: Toggle every 5 time units
    always begin
        #5 clock = ~clock;
    end

    initial begin
        // Initialize signals
        clock = 0;
        latch = 0;
        dec = 0;
        in = 4'b0000;

        // Monitor the signals
        $monitor($time, " latch=%b, dec=%b, in=%b, value=%b, zero=%b", latch, dec, in, uut.value, zero);

        // Test Case 1: Load initial value and decrement by 2
        #10 latch = 1; in = 4'b1010;    // Load value 10
        #10 latch = 0; dec = 1;         // Start decrementing
```

```verilog
module counter (
    input clock,            // Clock signal
    input [3:0] in,         // Starting count (4-bit input)
    input latch,            // Latch 'in' when high
    input dec,              // Decrement count when high
    output zero             // High when count is 0 or 1
);
    reg [3:0] value;        // Current count value

    always @(posedge clock) begin
        if (latch)
            value <= in;                // Load the input value when latch is high
        else if (dec && !zero) begin    // Decrement by 2 if zero flag is not set
            if (value > 2)
                value <= value - 2;
            else
                value <= 4'b0000;       // Ensure the counter stops at 0 or 1
        end
    end

    // Set the zero flag when the count is 0000 or 0001
    assign zero = (value == 4'b0000 || value == 4'b0001);
endmodule
```

```
[2024-12-19 17:43:55 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
                   0 latch=0, dec=0, in=0000, value=xxxx, zero=x
                  10 latch=1, dec=0, in=1010, value=xxxx, zero=x
                  15 latch=1, dec=0, in=1010, value=1010, zero=0
                  20 latch=0, dec=1, in=1010, value=1010, zero=0
                  25 latch=0, dec=1, in=1010, value=1000, zero=0
                  35 latch=0, dec=1, in=1010, value=0110, zero=0
                  45 latch=0, dec=1, in=1010, value=0100, zero=0
                  55 latch=0, dec=1, in=1010, value=0010, zero=0
                  65 latch=0, dec=1, in=1010, value=0000, zero=1
                 130 latch=1, dec=1, in=0110, value=0000, zero=1
                 135 latch=1, dec=1, in=0110, value=0110, zero=0
                 140 latch=0, dec=1, in=0110, value=0110, zero=0
                 145 latch=0, dec=1, in=0110, value=0100, zero=0
                 155 latch=0, dec=1, in=0110, value=0010, zero=0
                 165 latch=0, dec=1, in=0110, value=0000, zero=1
testbench.sv:43: $finish called at 240 (1s)
Done
```

MODULE

```
module counter (
    input clock,            // Clock signal
    input [3:0] in,         // Starting count (4-bit input)
    input latch,            // Latch 'in' when high
    input dec,              // Decrement count when high
    output zero             // High when count is 0 or 1
);
    reg [3:0] value;        // Current count value

    always @(posedge clock) begin
        if (latch)
            value <= in;              // Load the input value when latch
is high
        else if (dec && !zero) begin  // Decrement by 2 if zero flag is
not set
            if (value > 2)
                value <= value - 2;
            else
                value <= 4'b0000;     // Ensure the counter stops at 0
or 1
        end
    end

    // Set the zero flag when the count is 0000 or 0001
    assign zero = (value == 4'b0000 || value == 4'b0001);
endmodule
```

TESTBENCH

```
module counter_tb;
    reg clock;               // Clock signal
    reg [3:0] in;            // Starting count
    reg latch, dec;          // Latch and decrement control signals
    wire zero;               // Zero flag output

    // Instantiate the counter module
    counter uut (
        .clock(clock),
        .in(in),
        .latch(latch),
        .dec(dec),
        .zero(zero)
    );

    // Clock generation: Toggle every 5 time units
    always begin
        #5 clock = ~clock;
    end

    initial begin
        // Initialize signals
        clock = 0;
        latch = 0;
        dec = 0;
        in = 4'b0000;

        // Monitor the signals
        $monitor($time, " latch=%b, dec=%b, in=%b, value=%b, zero=%b",
latch, dec, in, uut.value, zero);

        // Test Case 1: Load initial value and decrement by 2
        #10 latch = 1; in = 4'b1010;   // Load value 10
        #10 latch = 0; dec = 1;       // Start decrementing

        // Test Case 2: Decrement to 0 or 1
        #100;                         // Let it decrement by 2 each
clock cycle

        // Test Case 3: Reload value
        #10 latch = 1; in = 4'b0110;  // Load value 6
        #10 latch = 0; dec = 1;       // Start decrementing again

        // End simulation
        #100 $finish;
    end
endmodule
```