Ryan Sullivan

U73687595

21 November 2019

EC311

Lab 2

Objective:

Create a counter with both manual and automatic counting functions with a universal reset and a seven-segment display.

Methodology:

Create multiple submodules that could be put together to create the desired product:

Counter:

The first iteration of the counter module that could only store up to one byte (8-bits), it would add a single bit to the total stored value for every time an increment is registered at a clock positive edge and would reset on a positive reset.

Counter16:

A modified version of the first counter module the only difference being that it could now store up to two bytes (16-bits) of data.

Debouncer:

A module used to account for the inaccuracy of a button push, it would register a single button push only after adding up enough of its signals and would not continually send a positive signal to the counter, thus negating the problem with having such a fast clock with such slow button pushes.

Clock Divider:

There were two separate clock divider modules, one for a fast clock (1 kHz) which would be used to refresh the seven-segment display, and one slow clock (1 Hz) used for the input for the automatic counting function. Both the clock dividers would only change their signal (from 1 to 0 or vice versa) once a certain number of clock pulses from the onboard clock had been registered, this number varies from clock to clock.

Display Control:

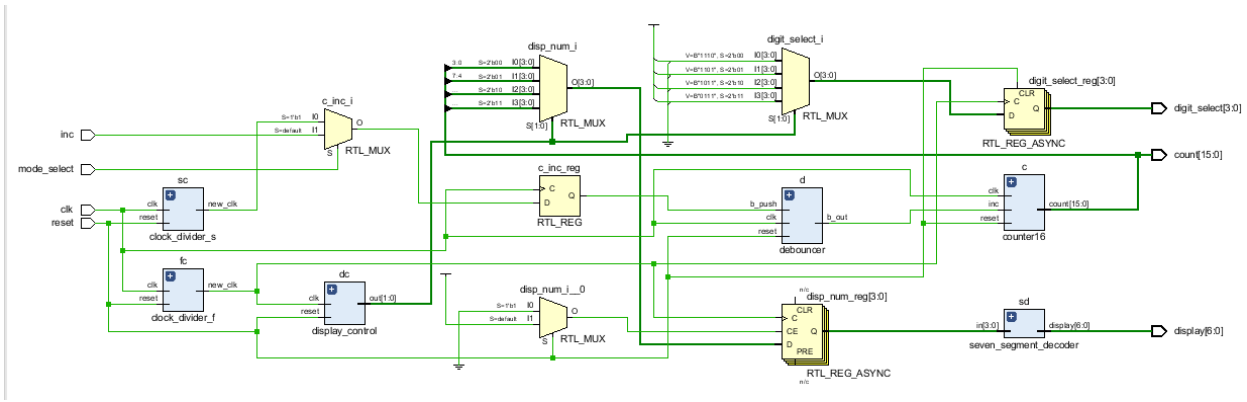
This was very simply a 2-bit counter that fed into 2:4 decoder, wherein the counter would register every clock pulse from the fast clock (1kHz) and increment by 1 bit, then the 2:4 decoder was fed into the seven-segment display to select a digit anode, refreshing each digit in sequence at a refresh rate of 1 kHz.

Seven-Segment Decoder:

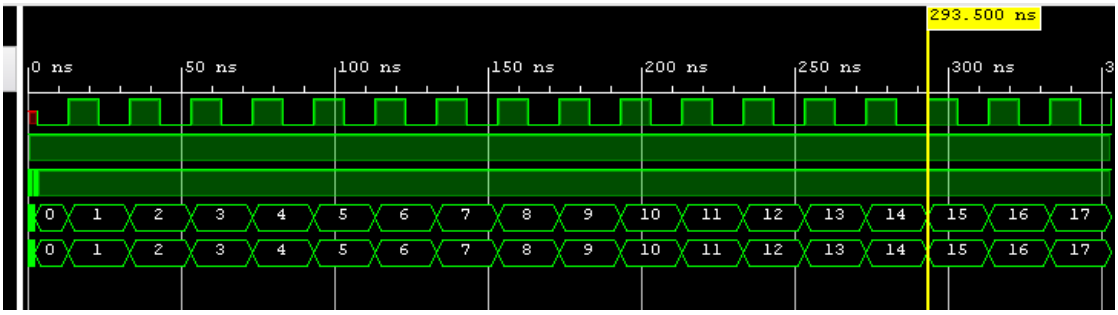
This was a 4:7 decoder that would take a 4-bit input and output a 7-bit bitstream that would be fed directly into the seven-segment display digits that would light up each segment in accordance with the 4-bit input (taken from the counter16 module in 4-bit chunks each corresponding to their own digit), which would translate the binary value into a more palatable hexadecimal value. The output is updated with every change to the input.

Observation:

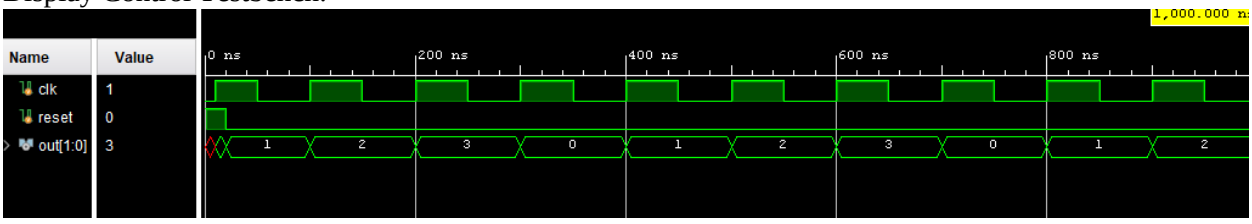
Elaborated Design:



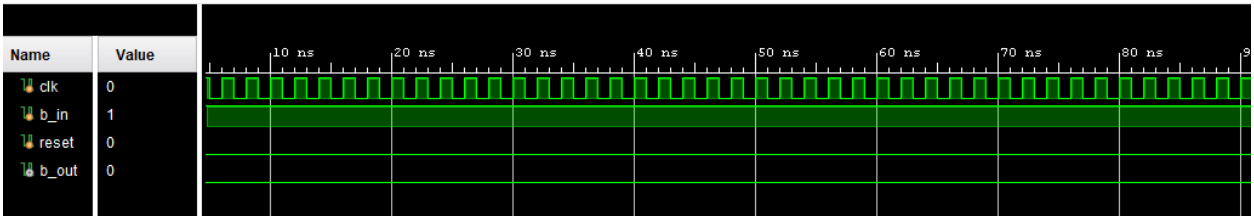
Counter Testbenches:



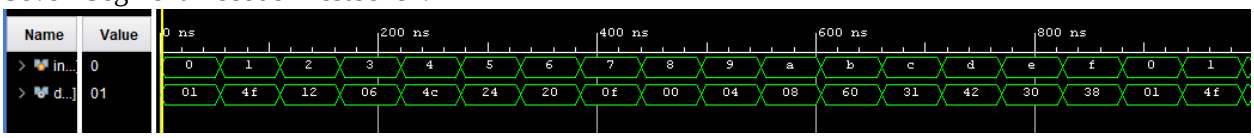
Display Control Testbench:



Debouncer Testbench:



Seven-Segment Decoder Testbench:



Constraints File:

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];


##Switches

#set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { swt[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
#set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { swt[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
#set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { swt[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
#set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { swt[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
#set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { swt[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
#set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { swt[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { swt[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { mode_select }]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]


## LEDs

set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { count[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { count[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { count[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { count[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { count[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { count[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { count[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { count[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { count[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { count[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { count[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { count[11] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { count[12] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { count[13] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { count[14] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { count[15] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]
```

```
##7 segment display

set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { display[6] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { display[5] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { display[4] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { display[3] }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { display[2] }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { display[1] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { display[0] }]; #IO_L4P_T0_D04_14 Sch=cg

#set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { DP }]; #IO_L19N_T3_A21_VREF_15 Sch=dp

set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { digit_select[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { digit_select[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { digit_select[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { digit_select[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
#set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { AN[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
#set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { AN[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
#set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { AN[6] }]; #IO_L23P_T3_35 Sch=an[6]
#set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { AN[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]


##Buttons

#set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { CPU_RESETN }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn

set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { inc }]; #IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { reset }]; #IO_L4N_T0_D05_14 Sch=btnu
#set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { BTNL }]; #IO_L12P_T1_MRCC_14 Sch=btnl
#set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { BTNR }]; #IO_L10N_T1_D15_14 Sch=btnr
#set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { BTND }]; #IO_L9N_T1_DQS_D13_14 Sch=btnd
```

Conclusion:

This lab has helped immensely in improving my understanding of Verilog and the workings of an FPGA board. The lab also taught me that a modular design is extremely efficient and the best in terms of testing and fixing any bugs that arise in project creation. Finally working with a physical board has improved my overall coding skillset and has better prepared me for the professional landscape in the future.

Code:

```
module top(
    input clk,
    input reset,
    input inc,
    output [7:0] count
    );

    wire b_out;

    debouncer d(
        .clk(clk),
        .reset(reset),
        .b_push(inc),
        .b_out(b_out)
    );
    counter c(
        .clk(clk),
        .reset(reset),
        .inc(b_out),
        .count(count)
    );
endmodule
```

```
module top2(
    input clk,
    input reset,
    input mode_select,
    input inc,
    output [15:0] count,
    output reg [3:0] digit_select,
    output [6:0] display
    );

    wire b_out, f_clk, s_clk;
    wire [1:0] dig_sel;
    reg [3:0] disp_num;
    reg c_inc;

    clock_divider_f fc(
        .clk(clk),
        .reset(reset),
        .new_clk(f_clk)
    );
    clock_divider_s sc(
        .clk(clk),
        .reset(reset),
        .new_clk(s_clk)
    );
    seven_segment_decoder sd(
        .in(disp_num),
        .display(display)
    );
    display_control dc(
        .clk(f_clk),
        .reset(reset),
        .out(dig_sel)
```

```
);
debouncer d(
    .clk(clk),
    .reset(reset),
    .b_push(c_inc),
    .b_out(b_out)
);
counter16 c(
    .clk(clk),
    .reset(reset),
    .inc(b_out),
    .count(count)
);

always@(posedge clk) begin
    if(mode_select) begin
        c_inc = s_clk;
    end else begin
        c_inc = inc;
    end
end
always@(posedge f_clk, negedge reset) begin
    if(reset) begin
        digit_select <= 4'b0000;
    end else if(f_clk) begin
        case(dig_sel)
            2'b00: begin
                digit_select <= 4'b1110;
                disp_num <= count[3:0];
                end
            2'b01: begin
                digit_select <= 4'b1101;
                disp_num <= count[7:4];
```

```
                    end
                2'b10: begin
                    digit_select <= 4'b1011;
                    disp_num <= count[11:8];
                    end
                2'b11: begin
                    digit_select <= 4'b0111;
                    disp_num <= count[15:12];
                    end
            endcase
        end
    end
endmodule
```

```
module counter(
    input clk,
    input reset,
    input inc,
    output reg [7:0] count
    );

    always @(posedge clk, negedge reset) begin
        if(reset) begin
            count <= 8'b00000000;
        end else if(clk) begin
            if(inc) begin
                count <= count + 8'b00000001;
            end
        end
    end
endmodule
```

```
module debouncer(
    input clk,
    input reset,
    input b_push,
    output reg b_out
    );
    parameter MAX = 1600000;
    reg [31:0] count;

    always @(posedge clk) begin
        if(reset)
            count <= 0;
        else if(b_push)
            count <= count +1;
        else
            count <= 0;
    end

    always @(posedge clk) begin
        if(reset)
            b_out <= 1'b0;
        else if(count == MAX)
            b_out <= 1'b1;
        else
            b_out <= 1'b0;
    end
endmodule
```

```
module counter16(
    input clk,
    input reset,
    input inc,
    output reg [15:0] count
    );

    always @(posedge clk, negedge reset) begin
        if(reset) begin
            count <= 16'b0000000000000000;
        end else if(clk) begin
            if(inc) begin
                count <= count + 16'b0000000000000001;
            end
        end
    end
endmodule
```

```
module clock_divider_f(
    input clk,
    input reset,
    output reg new_clk
    );
    reg [31:0] count;
    parameter slow_clk = 50000000;
    parameter fast_clk = 50000;

    always @(posedge clk, negedge reset) begin
        if(reset) begin
            count <= 32'b0;
            new_clk <= 1'b0;
        end
        else if(count == fast_clk -1)begin
            count <= 32'b0;
            new_clk <= ~new_clk;
        end
        else begin
            count <= count + 32'b1;
            new_clk <= new_clk;
        end
    end
endmodule
```

```
module clock_divider_s(
    input clk,
    input reset,
    output reg new_clk
    );
    reg [31:0] count;
    parameter slow_clk = 50000000;
    parameter fast_clk = 50000;

    always @(posedge clk, negedge reset) begin
        if(reset) begin
            count <= 32'b0;
            new_clk <= 1'b0;
        end
        else if(count == slow_clk -1)begin
            count <= 32'b0;
            new_clk <= ~new_clk;
        end
        else begin
            count <= count + 32'b1;
            new_clk <= new_clk;
        end
    end
endmodule
```

```
module seven_segment_decoder(
        input [3:0] in,
        output reg [6:0] display
    );
     always @(in)begin
            case(in)
                4'b0000: display <= 7'b0000001;
                4'b0001: display <= 7'b1001111;
                4'b0010: display <= 7'b0010010;
                4'b0011: display <= 7'b0000110;
                4'b0100: display <= 7'b1001100;
                4'b0101: display <= 7'b0100100;
                4'b0110: display <= 7'b0100000;
                4'b0111: display <= 7'b0001111;
                4'b1000: display <= 7'b0000000;
                4'b1001: display <= 7'b0000100;
                4'b1010: display <= 7'b0001000;
                4'b1011: display <= 7'b1100000;
                4'b1100: display <= 7'b0110001;
                4'b1101: display <= 7'b1000010;
                4'b1110: display <= 7'b0110000;
                4'b1111: display <= 7'b0111000;
            endcase
        end
endmodule
```

```
module display_control(
    input clk,
    input reset,
    output reg [1:0] out
    );


    always @(posedge clk, negedge reset) begin
        if(reset) begin
            out <= 2'b00;
        end else if(clk) begin
            out <= out + 2'b01;
        end
    end
endmodule
```

Testbenches:

```
module seven_seg_test();
    reg[3:0] in;
    wire[6:0] display;
    seven_segment_decoder s(
        .in(in),
        .display(display)
    );

    initial begin
        in <= 4'b0000;
    end
    always begin
        #55 in <= in + 4'b1;
    end
endmodule
```

```
module deb_test();
    reg clk, b_in, reset;
    wire b_out;

    debouncer d(
        .clk(clk),
        .b_push(b_in),
        .reset(reset),
        .b_out(b_out)
    );

    initial begin
        b_in <= 1'b1;
        reset <= 1'b1;
        #1 reset = 1'b0;
        clk <= 1'b0;
        forever begin
        #1 clk = ~clk;
        end
        #1;
    end
endmodule
```

```
module count_test();
    reg clk, inc, reset;
    wire [7:0] count;
    wire [15:0] count2;

    counter c(
        .clk(clk),
        .inc(inc),
        .reset(reset),
        .count(count)
    );
    counter16 c2(
        .clk(clk),
        .inc(inc),
        .reset(reset),
        .count(count2)
    );

    initial begin
        inc <= 1'b1;
        #1 reset <= 1'b1;
        #1 reset <= 1'b0;
        #1 reset <= 1'b1;
        clk <= 1'b0;
        forever begin
        #10 clk = ~clk;
        end
    end
endmodule
```

```
module display_control(
    input clk,
    input reset,
    output reg [1:0] out
    );


    always @(posedge clk, negedge reset) begin
        if(reset) begin
            out <= 2'b00;
        end else if(clk) begin
            out <= out + 2'b01;
        end
    end
endmodule
```