Lab 5 BCD Up/Down Counter on 7-Segment Display

Course: ECE 3300

Date: 7/20/2025

Authors: Mariam, Anish C

Introduction

In this lab, we designed and implemented a two-digit binary-coded Decimal up/down counter on a Nexys A7 FPGA board using verilog HDL . The counter's output was displayed on a 7-segment display using time-multiblexing. User interaction was enabled through pushbuttons to control the counting direction and reset, and through switches to control the clock speed. This lab integrates several digital design principles such as modular coding, clock division, cascading counters, and multiplexed display control.

Objective

- Design a two-digit up/down BCD counter using verilog HDL.
- Implement a 32-bit free-running counter and a 32-to-1 multiplexer to divide the clock.
- Control counter direction using BTN1 and reset using BTN0.
- Display the counter output on a dual-digit 7-segment display using scan multiplexing.
- Simulate, synthesize and program the design onto a Nexys A7 FPGA board.
- Test all hardware functionality including speed selection, up/dowm count, and reset behavior.

Parts/ Applications

- Nexys A7 FPGA
- Vivado Design suit
- Verilog HDL
- Slide Switches
- LEDS
- 7-segments

Theory

The counter operates using a divided clock signal derived from a 100MHz board clock. The clock is divided using a 32-bit free running counter and a 32-to-1 multiplexer, with speed selection controlled via five slide switches(sw[4:0]). The core of the design consists of two cascaded BCD counter:

- **Units digit counter**: increment or decrements based on the direction signal(BTN1) and rolls over from 9 to 0 or underflows from 0 to 9
- **Tens digit counter:** Triggered by overflow or underflow from the unit's digit counter.

Both counters share the reset(BTN0) and direction (BTN1) signal.

The 7-segment display is driven by a seg7_scan module that time-multiplexes the two digits using control signal AN[1:0]. The segments SEG[6:0] display the appropriate digit values in BCD. Additionally,LED indicators show the current BCD value for both digits and reflect the switch settings for debugging.

The entire system was simulated using Testbenches for each module and verified using behavioral and timing simulations before being synthesized and programmed on the hardware.

Design Overview: Code

Clock_divider.v

This Clock_divide module creates a 32-bit free-running counter by incrementing a register on every positive clock edge. It serves as a basis for generating slower clock signals by dividing the 100 MHz onboard clock.

```
module clock_divider(
input clk,
    input rst,
    output reg [31:0] count
);
    always @(posedge clk or posedge rst) begin
        if (rst)
            count <= 0;
        else
            count <= count + 1;
    end
endmodule
```

Mux32x1.v

The mux32x1 module implements a 32-to-1 multiplexer. It selects one of the 32 bits from the count signal based on the 5-bit sel input from the switches. The selected bit becomes the divided clock output. This effectively allows the user to control the clock speed dynamically, enabling fast or slow counting.

sw[4:0]	Selected bit	Approx frequency	Behavior
00000	count[0]	50 MHz	Too fast to Observed
00010	count[2]	12.5 MHz	Very fast
01000	count[19]	~95Hz	Human observable
10000	count[24]	~3Hz	Very slow
11111	count[29]	~0.095 Hz	Extremely slow

```
module mux32x1(

  input [31:0] count,
    input [4:0] sel,
    output clk_out
);
    assign clk_out = count[sel];
endmodule
```

Bcd_up_down_counter.v

The bcd_up_down_counter module combines two bcd_counter instances to form a two-digit counter units and tens, the units counter increments or decrements based on the dir signal. When it overflows or underflows, it triggers the tens counter. This setup mimics how decimal counting works and enables full-range BCD counting from 00 to 99 and vice versa.

```
module bcd_up_down_counter(
    input clk,
    input rst_n,
    input dir,
    output [3:0] units,
    output [3:0] tens
);
    wire en = 1'b1;
    wire en_tens;

    // The first Digit
    bcd_counter UNIT (
        .clk(clk),
        .rst(rst_n),
        .en_in(en),
        .upd(dir),
        .op(units),
        .en_out(en_tens)
    );

    // Second Digit
    bcd_counter TENS (
        .clk(clk),
        .rst(rst_n),
        .en_in(en_tens),
        .upd(dir),
        .op(tens),
        .en_out() // unused
    );
endmodule
```

```
module bcd_counter(
input clk,
    input rst,
    input en_in,
    input upd,
    output reg [3:0] op, // 4 Bit Output
    output reg en_out
);
    always @(posedge clk or posedge rst)
    begin
        // When reset is pressed everything is 0
        if (rst)
            begin
                op <= 0;
                en_out <= 0;
            end
        // ????
        else
            if (en_in)
                begin
                    en_out <= 0;
                    if (upd)
                        begin
                            if (op == 9)
                                begin
                                    op <= 0;
                                    en_out <= 1;
                                end
                            else
                                begin
                                    op <= op + 1;
                                end
                        end
                    // ?????
                    else
                        begin
                            if (op == 0)
                                begin
                                    op <= 9;
                                    en_out <= 1;
                                end
                            else
                                begin
                                    op <= op - 1;
                                end
                        end
                end
        end
endmodule
```

Seg7_scan.v

The seg7_scan module handles the 7-segment display using time-multiplexing. It alternates between displaying the units and tens digits at a high frequency using a refresh clock, marking it as if both digits are always lit. It also includes a lookup function (decode_7seg) that converts 4-bit BCD values into 7 segments encoding patterns.

```
module seg7_scan(
 input clk,
    input [3:0] digit0,
    input [3:0] digit1,
    output reg [6:0] seg,// Change to CNODE
    output reg [7:0] an // CHANGE TO AN
);
    reg [19:0] refresh_counter = 0;
    reg refresh_clk = 0;
    reg toggle = 0;

    initial begin
        seg = 7'b1111111;
        an  = 8'b11111111;
    end

    always @(posedge clk) begin
        refresh_counter <= refresh_counter + 1;
        refresh_clk <= refresh_counter[19];
    end

    always @(posedge refresh_clk) begin
        toggle <= ~toggle;
        if (toggle) begin
            an <= 8'b11111110; // AN[0]
            seg <= decode_7seg(digit0);
        end else begin
            an <= 8'b11111101; // AN[1]
            seg <= decode_7seg(digit1);
        end
    end

    function [6:0] decode_7seg;
        input [3:0] digit;
        case (digit)
            4'd0: decode_7seg = 7'b1000000;
            4'd1: decode_7seg = 7'b1111001;
            4'd2: decode_7seg = 7'b0100100;
            4'd3: decode_7seg = 7'b0110000;
            4'd4: decode_7seg = 7'b0011001;
            4'd5: decode_7seg = 7'b0010010;
            4'd6: decode_7seg = 7'b0000010;
            4'd7: decode_7seg = 7'b1111000;
            4'd8: decode_7seg = 7'b0000000;
            4'd9: decode_7seg = 7'b0010000;
            default: decode_7seg = 7'b1111111;
        endcase
    endfunction
endmodule
```

Top_lab5.v

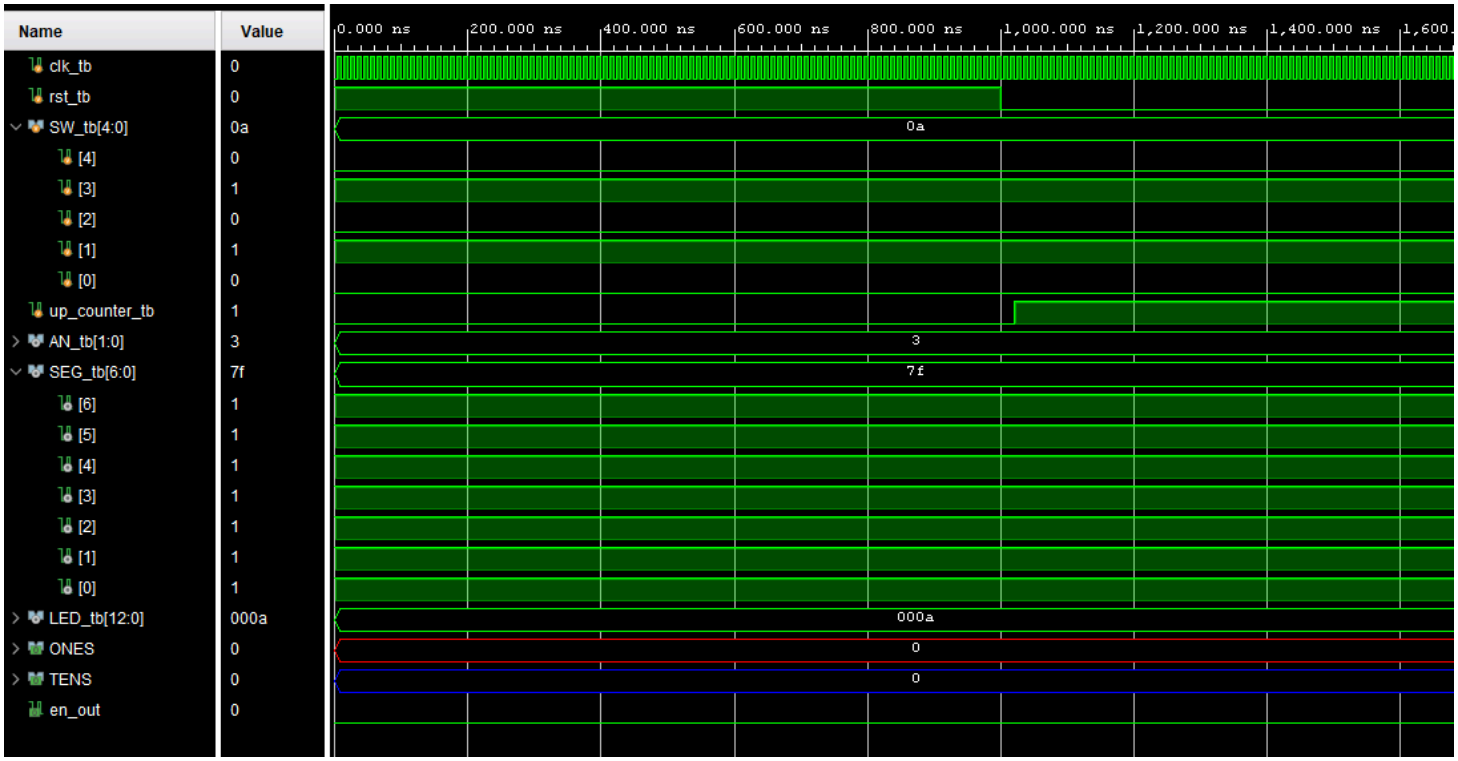
The top_lab5 module is the top level design that integrates all submodules. It connects input controls and output to build the complete system. It demonstrates modular design principles by combining the clock divider, multiplexer, counter, and display log into a unified and functional FPGA project.

```
module top_lab5(

    input clk,
    input [4:0] sw,
    input btn0, // This is the reset
    input btn1,
    output [6:0] seg,
    output [7:0] an,
    output [12:0] led
);
    wire [31:0] count;
    wire clk_div;
    wire [3:0] units, tens;

    assign led[4:0] = sw;
    assign led[8:5] = units;
    assign led[12:9] = tens;

    clock_divider CD (
        .clk(clk),
        .rst(btn0),
        .count(count)
    );

    mux32x1 MUX (
        .count(count),
        .sel(sw),
        .clk_out(clk_div)
    );

    bcd_up_down_counter COUNTER (
        .clk(clk_div),
        .rst_n(btn0),
        .dir(btn1),
        .units(units),
        .tens(tens)
    );

    seg7_scan DISPLAY (
        .clk(clk),
        .digit0(units),
        .digit1(tens),
        .seg(seg),
        .an(an)
    );
endmodule
```

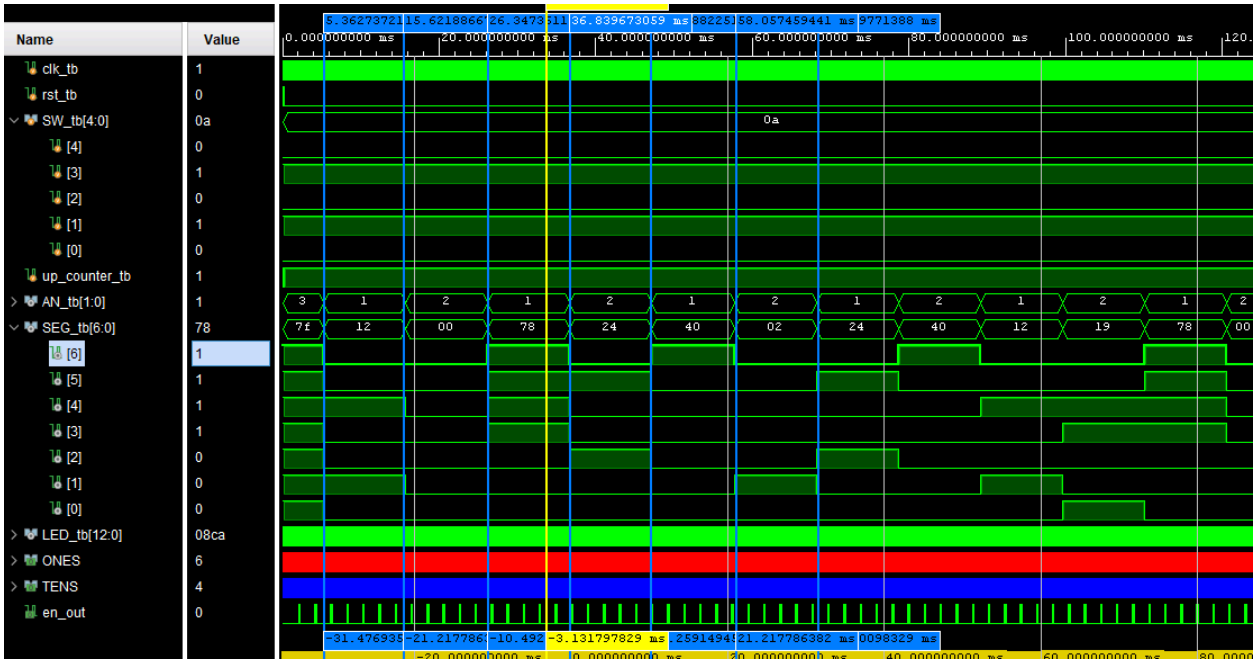
TestBench: (Code)

```
module top_lab5_tb();

    reg clk_tb, rst_tb;
    reg [4:0] SW_tb;
    reg up_counter_tb;
    wire [1:0] AN_tb;
    wire [6:0] SEG_tb;
    wire [12:0] LED_tb;


    top_lab5 DUT(
        .clk(clk_tb),
        .sw(SW_tb),
        .btn0(rst_tb),
        .btn1(up_counter_tb),
        .seg(SEG_tb),
        .an(AN_tb),
        .led(LED_tb)
    );

    always
        begin: clock_gen
            #5 clk_tb = ~clk_tb;
        end

    initial
        begin
            clk_tb = 0;
            SW_tb = 5'b01010;
            rst_tb = 1;
            up_counter_tb = 0;

            // Run with Reset high for some time
            #1000;

            rst_tb = 0;
            #20
            up_counter_tb = 1;

            #10000;
            $stop;

        end
```

Simulation: (Graphs)



Reset Switches From High to Low and Up Counter is High

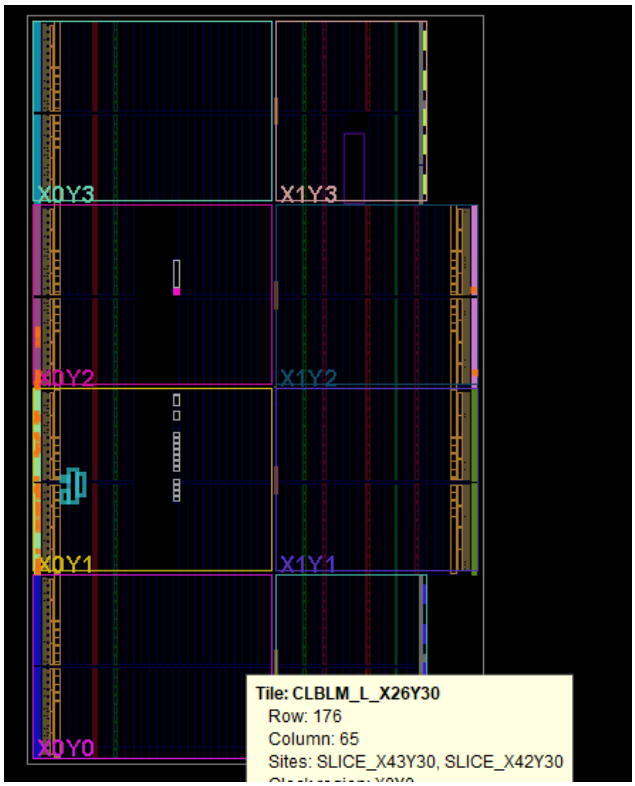


We can see the digits switches every 10ms since the switch is ar 4’b01010.

Nexys-A7-100T-Master.xdc

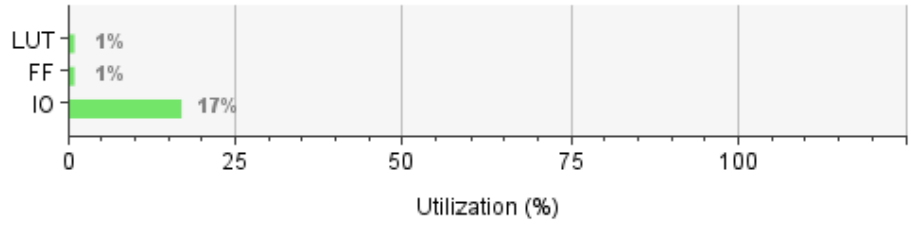
```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk}];

##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { sw[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]


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


##7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports {  seg[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports {  seg[1] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports {  seg[2] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports {  seg[3] }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports {  seg[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports {  seg[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports {  seg[6] }]; #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 { an[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { an[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { an[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { an[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 N17   IOSTANDARD LVCMOS33 } [get_ports { btn0 }]; #IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { btnl }]; #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
```

Implementation summary:



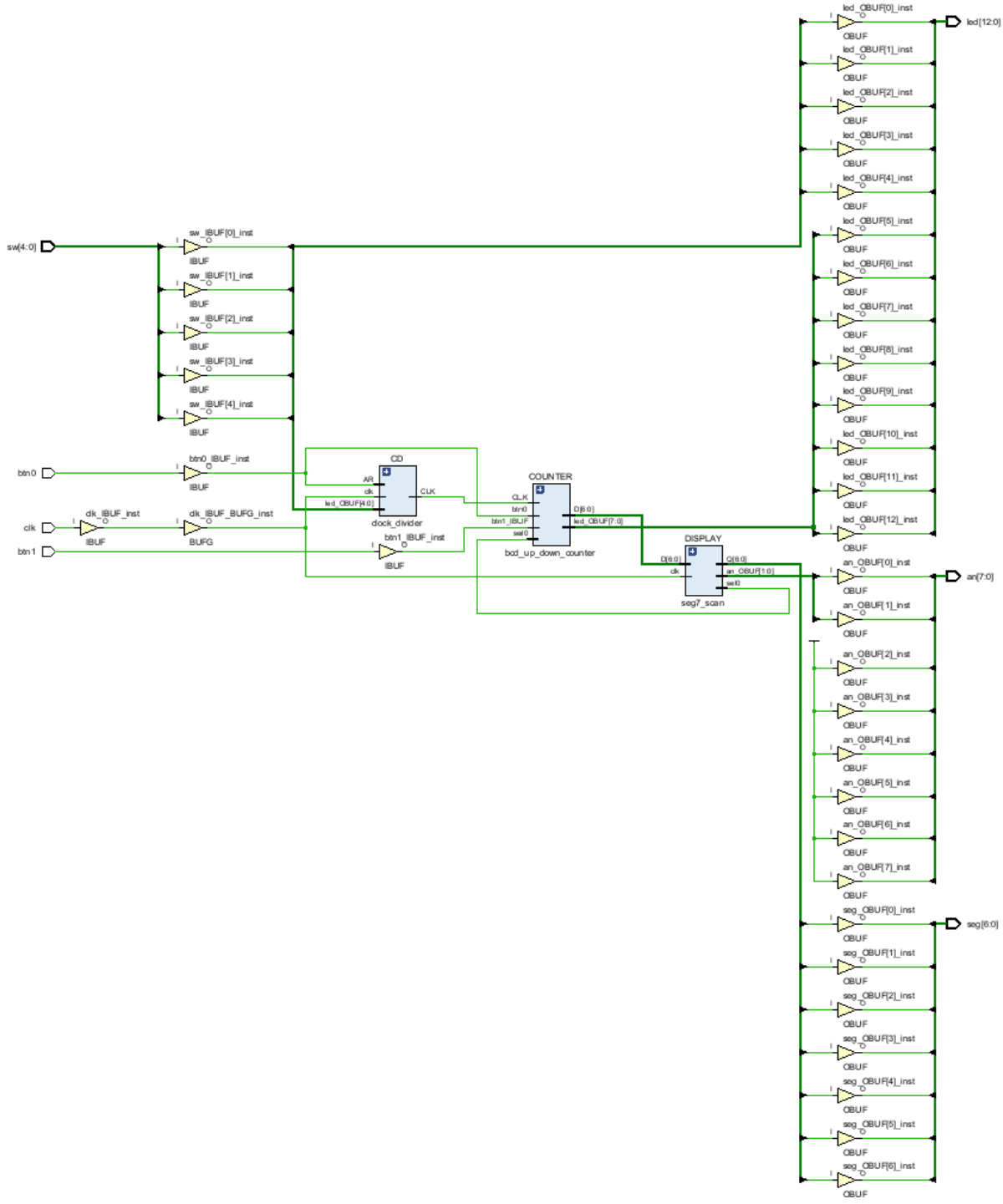
Utilization Report:

Resource	Utilization	Available	Utilization %
LUT	27	63400	0.04
FF	72	126800	0.06
IO	36	210	17.14



In my design, the overall FPGA resource utilization was extremely low, indicating efficient implementation. The logic required only 27 look-up tables (LUTs) and 72 flip-flops (FFs), which is just 0.04% and 0.06% of the available resources, respectively. This confirms that the counter logic, clock divider, and display control modules were lightweight and well-optimized. However, the IO utilization was higher at 17.14%, with 36 out of 210 available pins used. This is expected, as the design interfaces with multiple external components including switches, buttons, LEDs, and a dual-digit 7-segment display. Overall, my design fits well within the capacity of the Nexys A7 FPGA and leaves ample room for future expansion or feature upgrades.

Implementation Schematic (Synthesis):



Conclusion

This lab reinforced key concepts in digital design, including modular verilog programming, clock management, cascading counters, and hardware multiplexing. By successfully implementing a BCD up/down counter with user control and dynamic display, we gained valuable experience in interfacing input/output devices with an FPGA. The completed design demonstrates a scalable and interactive approach to counter-based digital systems, forming a strong foundation for more advanced sequential circuit designs.

Contributions:

Mariam : Top.v Code, Synthesis, Lab Report, XDC File

Anish: Testbench, Verification, Lab Report, Vivado Utilization

Youtube Link