ECE3300L.E01

Lab 6: Dual BCD Up/Down Counters. ALU, and Control Display on 7-Segment

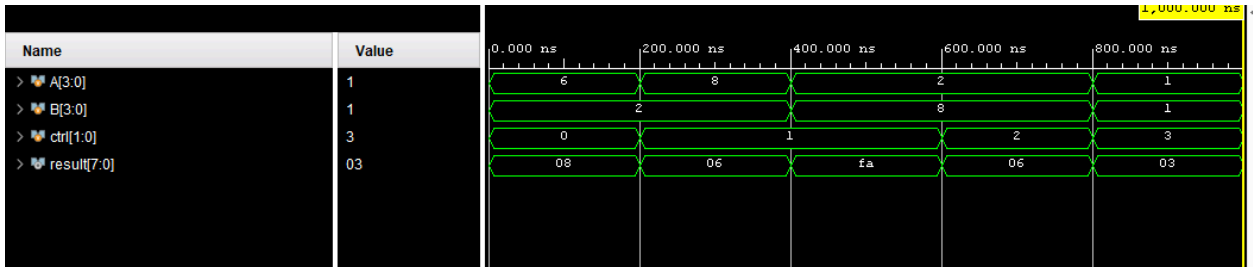
Priyanka Ravinder and Raj Gokidi

07/25/2025

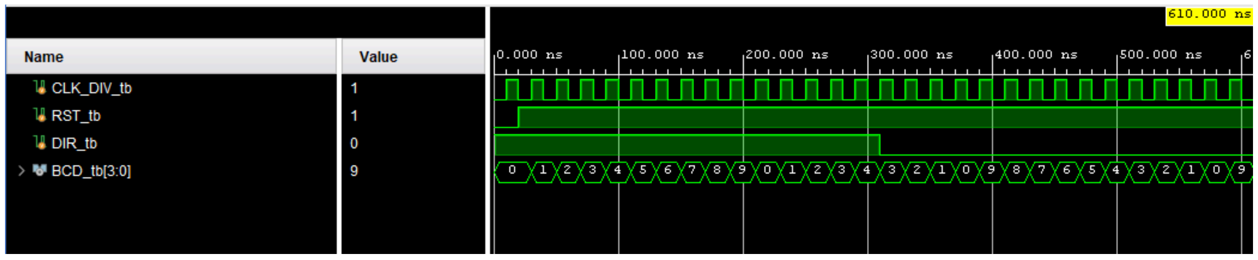
Introduction:

The purpose of this experiment was to write a program which is able to both implement and simulate arithmetic operations using a 4-bit ALU, convert binary outputs to BCD and show the results on a 7-segment display. The program should be able to perform all these operations with the use of control inputs. Once the results are converted to BCD, it is then shown on the 7-segment display via multiplexing. After the program was implemented and tested using the board, we were also able to simulate it in Vivado and capture the waveforms. The lab ultimately lets us explore arithmetic, display controls and BCD conversions using the Nexys A7-100T FPGA board.

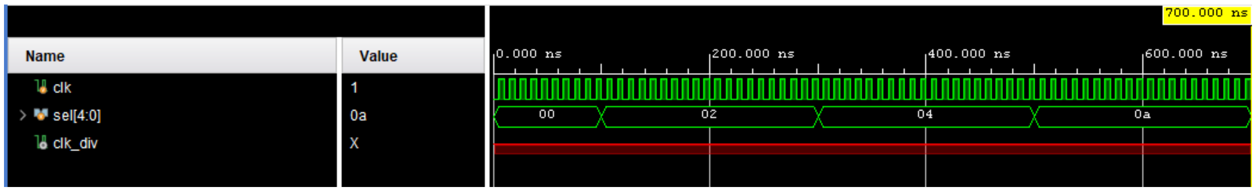
Waveforms:



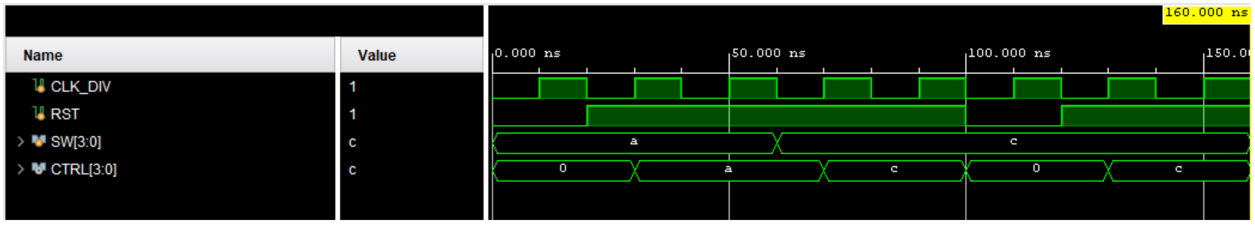
alu_tb waveform



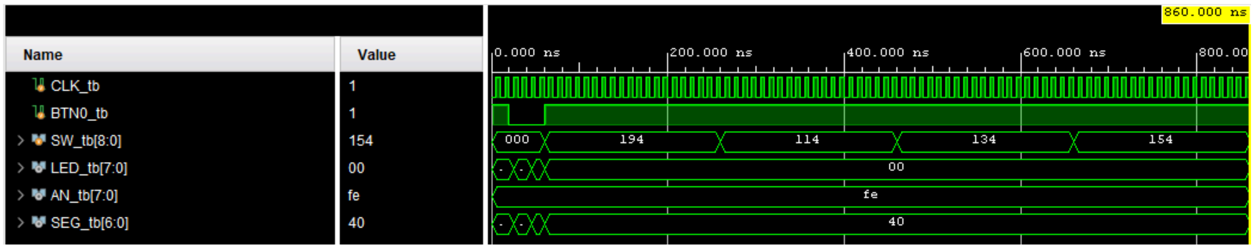
Bcd_counter_tb



Clock_divider_tb



Control_decoder_tb



top_lab6_tb

<u>Code:</u>

<u>alu.v</u>

module alu(

input [3:0] A,

input [3:0] B,

input [1:0] ctrl,

output reg [7:0] result

);

always @(*) begin

case (ctrl)

```
2'b00: result = A+B;

2'b01: result = A-B;

default: result = 0;

endcase

end

endmodule
```

<u>Bcdcounter.v</u>

```
`timescale 1ns / 1ps

module bcd_counter(

   input clk_div,//divided clock

   input BTN0,// active low reset

   input dir_bit, // direction bit

   output reg [3:0] BCD // 4 bit BCD

   );

   always @(posedge clk_div or negedge BTN0) begin

      if (!BTN0) begin

      //resetbcd value

         BCD <= 4'd0;

      end else begin

         if (dir_bit) begin

            if (BCD == 4'd9) begin

               BCD <= 4'd0;
```

```
                end else begin
                    BCD <= BCD + 4'd1;
                end
            end else begin
                if (BCD == 4'd0) begin
                    BCD <= 4'd9;
                end else begin
                    BCD <= BCD - 4'd1;
                end
            end
        end
    end
endmodule
```

<u>clockdivider.v</u>

```
module clock_divider
(
input clk,
input BTN0,
input [4:0] sel,
output reg [31:0] cnt,
output clk_div
);
```

```
always @(posedge clk or negedge BTN0)
begin
if (!BTN0)
cnt <= 32'b0;
else
cnt <= cnt + 1;
end
assign clk_div = cnt[sel];


endmodule
```

Seg7scan.v

```
`timescale 1ns / 1ps
module seg7_scan(
   input clk,                     // 100 MHz board clock
   input BTN0,                    // Active-low reset
   input [3:0] digit0,            // lower nibble result
   input [3:0] digit1,            //upper nibble result
   input [3:0] digit2,            // control nibble
   output reg [6:0] SEG,          // Segment lines a-g
   output reg [3:0] AN            // Anode lines (active low)
   );
   reg [15:0] refresh_counter = 0;
   always @(posedge clk or negedge BTN0) begin
```

```
 if (!BTN0)
 refresh_counter <= 16'd0;
 else
 refresh_counter <= refresh_counter + 1;
 end
 wire [1:0] sel = refresh_counter[15:14];
reg [3:0] current_digit;
always @(*) begin
case (sel)
2'd0: current_digit = digit0;
2'd1: current_digit = digit1;
2'd2: current_digit = digit2;
default: current_digit = 4'd0;
endcase
end
    always @(*) begin
     case (sel)
         2'd0: AN = 4'b1110;  // enable digit0
         2'd1: AN = 4'b1101;  // enable digit1
         2'd2: AN = 4'b1011;
         default: AN = 4'b1111;
     endcase
 end
```

```
    always @(*) begin
        case (current_digit)
4'h0: SEG = 7'b1000000;
4'h1: SEG = 7'b1111001;
4'h2: SEG = 7'b0100100;
4'h3: SEG = 7'b0110000;
4'h4: SEG = 7'b0011001;
4'h5: SEG = 7'b0010010;
4'h6: SEG = 7'b0000010;
4'h7: SEG = 7'b1111000;
4'h8: SEG = 7'b0000000;
4'h9: SEG = 7'b0010000;
4'hA: SEG = 7'b0001000;
4'hB: SEG = 7'b0000011;
4'hC: SEG = 7'b1000110;
4'hD: SEG = 7'b0100001;
4'hE: SEG = 7'b0000110;
4'hF: SEG = 7'b0001110;
default: SEG = 7'b1111111;
        endcase
    end
endmodule
```

Top.v

```
module top_lab6(

input CLK,

input BTN0,

input  [8:0]  SW,

output [7:0]  LED,

output [7:0] AN,

output [6:0] SEG

);

wire [31:0] cnt;

wire clk_out;

clock_divider u_div(

.clk(CLK),

.BTN0(!BTN0),

.sel(SW[4:0]),

.clk_div(clk_out),

.cnt(cnt)

);

 wire [3:0] unit_bcd, tens_bcd, control_display;

 wire roll;

  bcd_counter bcd_count_ones(

  .clk_div(clk_out),
  .BTN0(!BTN0),
  .dir_bit(SW[7]),
  .BCD(unit_bcd)
  );
  bcd_counter bcd_count_tens(
  .clk_div(clk_out),
  .BTN0(!BTN0),
  .dir_bit(SW[8]),
  .BCD(tens_bcd)
  );
wire [7:0] alu_result;
wire [3:0] ctrl_nibble = SW[8:5];
alu alu1(
.A(unit_bcd),
.B(tens_bcd),
 .ctrl({SW[6], SW[5]}),
.result(alu_result)
);
control_decoder dec(
 .clk_div(clk_out),
 .BTN0(!BTN0),
.SW(SW[8:5]),
.ctrl_nibble(control_nibble)
```

```
);
wire [3:0] scan_AN;
   seg7_scan u_scan (
   .clk(CLK),
   .BTN0(!BTN0),
   .digit0(alu_result[3:0]),
   .digit1(alu_result[7:4]),
   .digit2(ctrl_nibble),
   .SEG(SEG),
   .AN(scan_AN)
   );
 assign LED[3:0] = unit_bcd;
 assign LED[7:4] = tens_bcd;
 assign AN = { 5'b11111, scan_AN[2:0] };
endmodule
```

<u>lab6.xdc</u>

```
## 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]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { SW[5]
}]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { SW[6]
}]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7]
}]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS33 } [get_ports { SW[8] }];
#IO_L24N_T3_34 Sch=sw[8]
## 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]
##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 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 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 { BTN0
}]; #IO_L9P_T1_DQS_14 Sch=btnc
```

<u>Video Link:</u>

https://youtu.be/HgjVjz3XO6Q

<u>Contributions:</u>

Priyanka Ravinder: Code and Report

Raj Gokidi: Code and Report

<u>Conclusion:</u>

Overall, this lab solidified our knowledge of how to design and implement 4-bit signed operations. Using Verilog code, we were able to create the ALU which was able to properly perform all of the required arithmetic operations, and the final output was converted from binary to BCD. The SEGDRIVE module was responsible for displaying on the 7-segment display using multiplexing. The simulation also confirmed that the program was functioning correctly, thus displaying the correct waveforms.