DESIGN OF (ALU)

ARITHMETIC LOGIC UNT

Mansoura University
ECE Department

TEAM MEMBERS

AMR LABIB ISSA

HASSAN MUHAMMED JUHA

ELSAYED ASHRAF

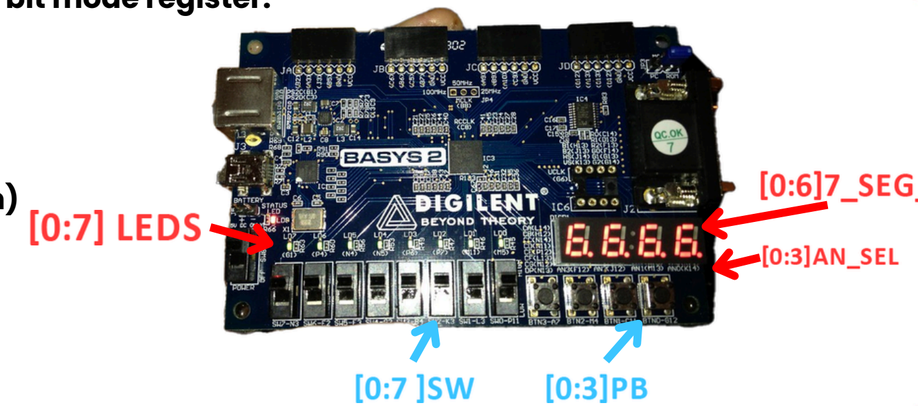
AGENDA

- Overview of the project
- Schematic of the project
- clock divider Block
- Debouncing Block
- ALU block
- Binary to BCD Block
- Implementation of 7 segment display on FPGA
- Anode Select Block
- Digit Control Block
- 7-segment decoder Block
- Module ALU TOP

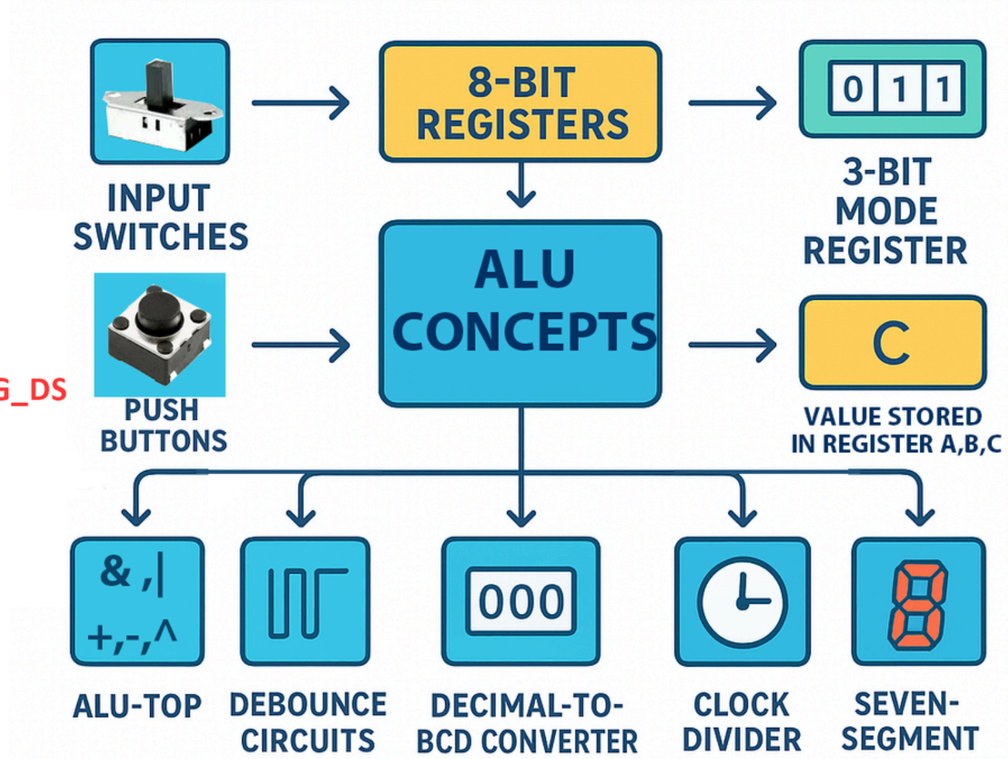
OVERVIEW OF THE PROJECT

- Design and implement an 8-bit Arithmetic Logic Unit (ALU).
- Data Handling:
  - Operands loaded into 8-bit registers A and B via switches and push buttons.
  - Results stored in register C.
  - Operation selected using a 3-bit mode register.

- Supported Operations (6 total):
  a. Addition
  b. Subtraction
  c. Two's complement (negation)
  d. Bitwise AND
  e. Bitwise OR
  f. Bitwise XOR

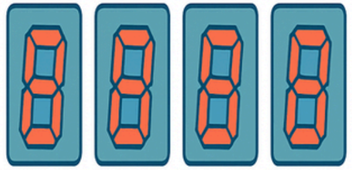


- Output Display:
  - Result shown in decimal form on seven-segment displays.
- System Modules:
  - ALU_TOP (main control)
  - Debounce circuits (stable inputs)
  - Decimal-to-BCD converter
  - Clock divider
  - Seven-segment multiplexer/decoder
- Outcome:
- A fully functional 8-bit ALU system demonstrating:
  - Digital design principles
  - Modular implementation
  - Hardware-based arithmetic & logic processing



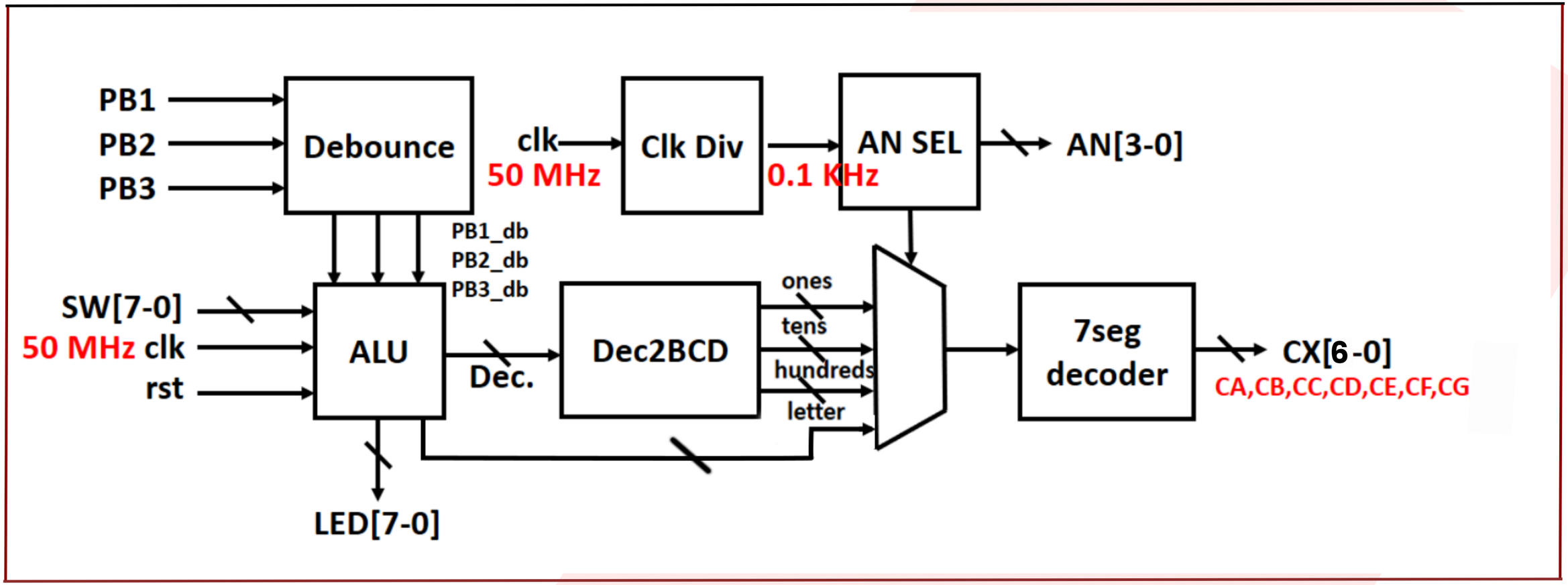
PROJECT CONSEPTS

8IMPLEMENTATION OF AN 8-BIT ALU
*SUPPORTING SIX DIFFERENT OPERATIONS
* ABILITY TO INPUT VALUES THROUGH SWITCHES.
* CONTROL OF OPERATION SELECTION VIA A MODE REGISTER.
* CLEAR DISPLAY OF A, B, AND C VALUES ON THE SEVEN-SEGMENT DISPLAY.

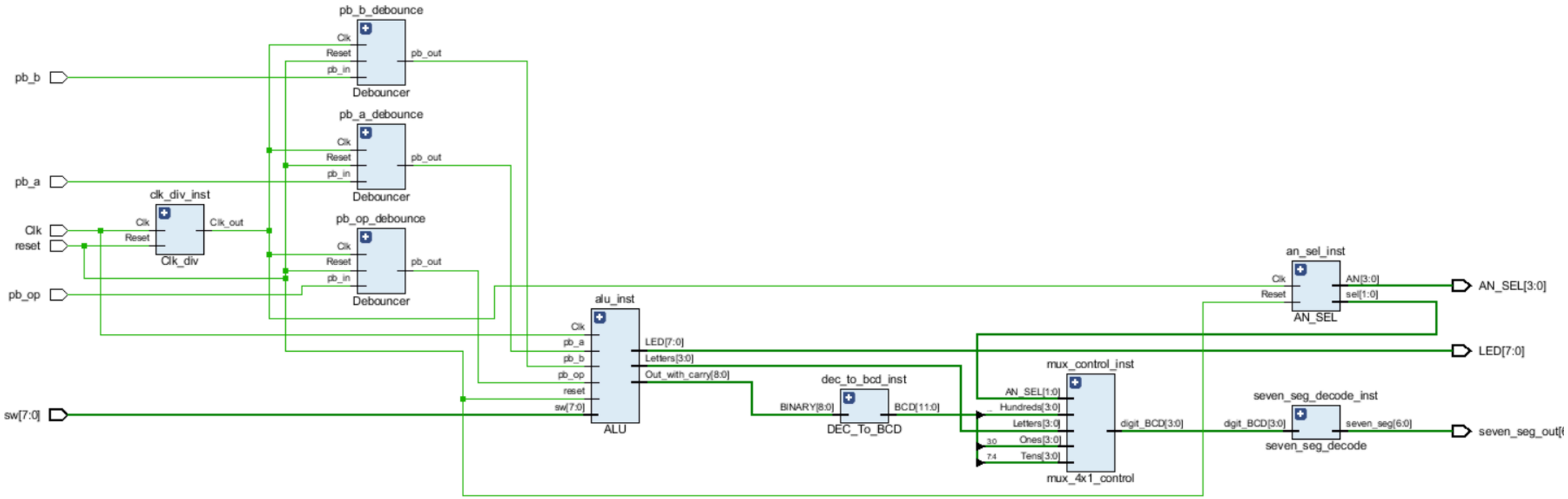


OUTCOME

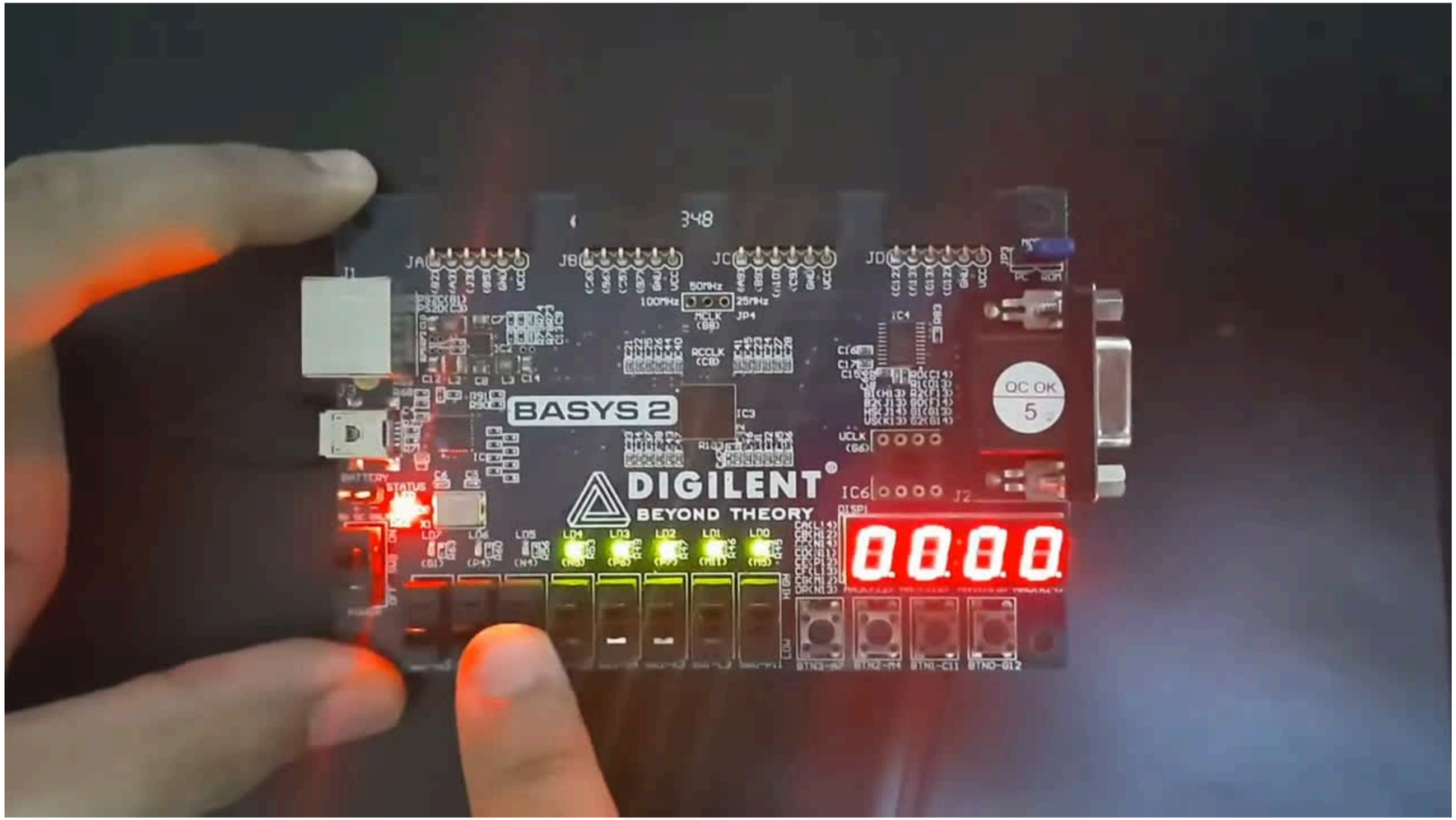
A BLOCK DIAGRAM OF THE PROJECT



A SCHEMATIC OF THE PROJECT



OVERVIEW OF THE PROJECT



CLOCK DIVIDER

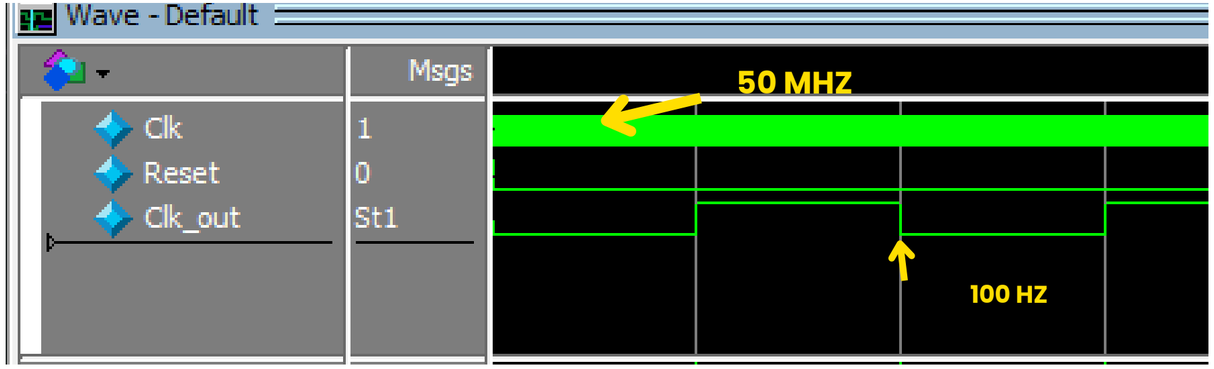
A clock divider is a digital circuit used to reduce the frequency of a high-speed clock signal and generate a slower clock output.

- It works by counting the input clock pulses using an internal counter.
- Once the counter reaches a predefined value, the circuit toggles the output clock and resets the counter.
- Repeating this process produces an output clock with a lower frequency but the same duty cycle (approximately 50%).
- A reset input ensures the divider can be initialized to a known state at any time.

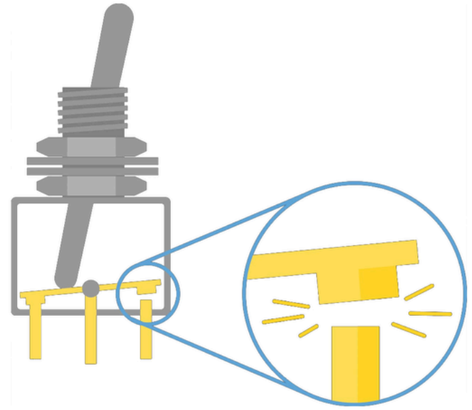
This logic is widely used in timing control, digital systems, and embedded applications where slower clock signals are required from a high-frequency source.

```
# Time: 0 ns, Clk_out: 0
# Time: 5000010 ns, Clk_out: 1
# Time: 10000010 ns, Clk_out: 0
# Time: 15000010 ns, Clk_out: 1
# Time: 20000010 ns, Clk_out: 0
# Time: 25000010 ns, Clk_out: 1
# Time: 30000010 ns, Clk_out: 0
# Time: 35000010 ns, Clk_out: 1
# Time: 40000010 ns, Clk_out: 0
# Time: 45000010 ns, Clk_out: 1
# Time: 50000010 ns, Clk_out: 0
# Time: 55000010 ns, Clk_out: 1
# Time: 60000010 ns, Clk_out: 0
# Time: 65000010 ns, Clk_out: 1
# Time: 70000010 ns, Clk_out: 0
# Break key hit
```

```
1  module Clk_div #(parameter counter_div = 25'd249_999) (Clk,Reset,Clk_out);
2      input Clk, Reset;
3      output reg Clk_out;
4
5      reg [24:0] counter=0; // 25-bit counter for division
6
7      always @(posedge Clk or posedge Reset) begin
8          if (Reset) begin
9              counter <= 0;
10             Clk_out <= 0;
11         end else if (counter < counter_div ) begin // Adjust this value for desired frequency
12                 counter <= counter + 1; // Increment counter
13             end else begin
14                 Clk_out <= ~Clk_out; // Toggle output clock
15                 counter <= 0; // Reset counter
16             end
17         end
18
19 endmodule //Clk_div
```



DEBOUNCING BLOCK

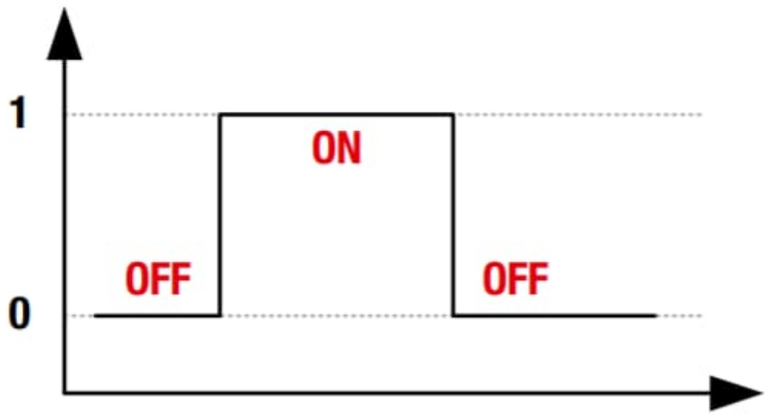


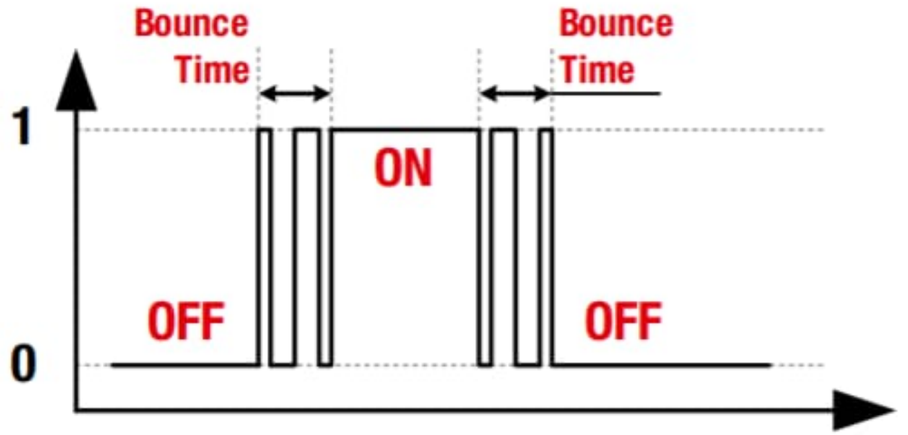
Bouncing occurs when two metal contacts in an electronic device generate multiple signals as they close or open. Debouncing is a hardware or software technique that ensures only a single signal is acted upon for each opening or closing of a contact.

- **Ideal switch**

VS

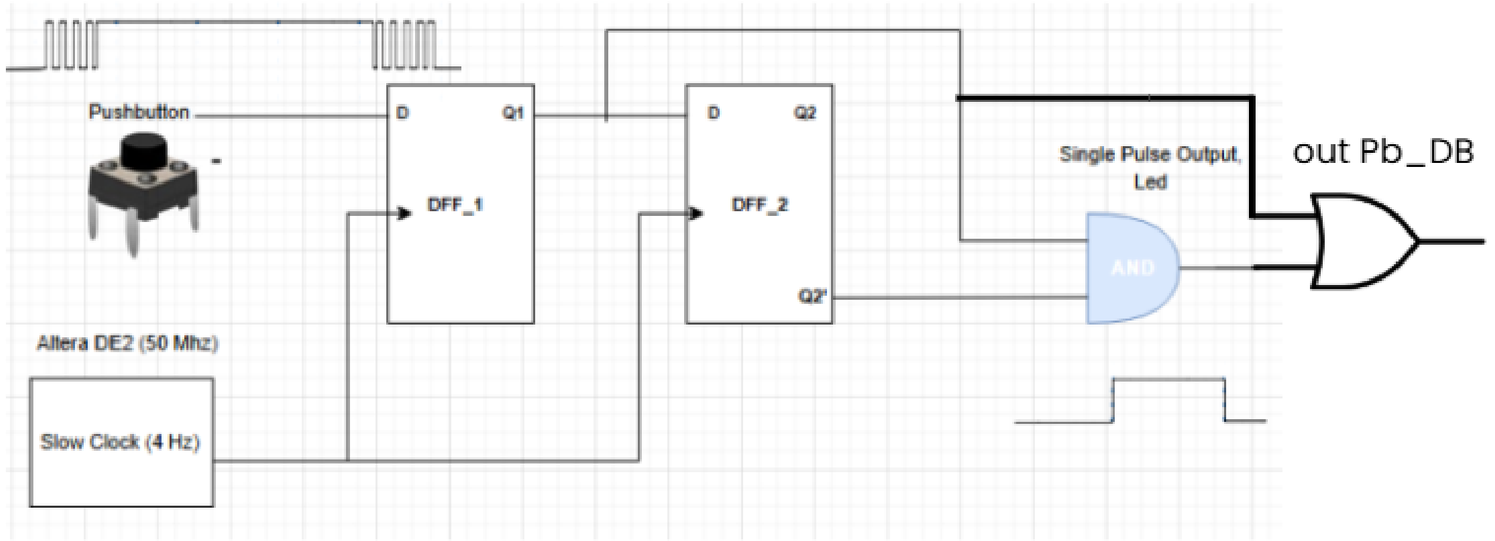
- **Bouncing switch**





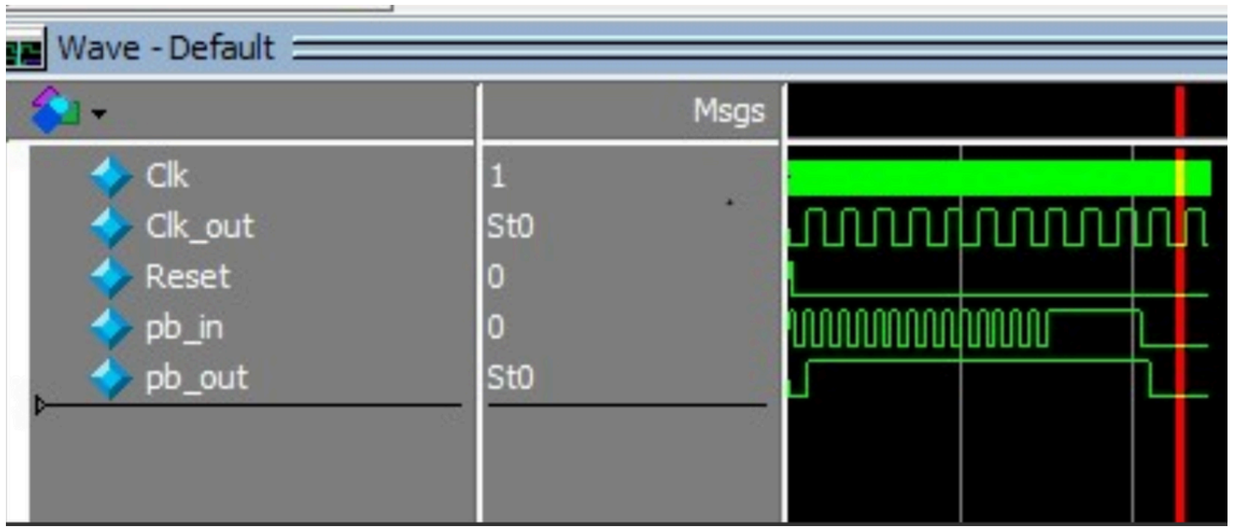
DEBOUNCING MODULE :

```verilog
module D_FF (Clk, Reset, D, Q, Qn);
input Clk, Reset, D;
output reg Q, Qn;
always @(posedge Clk or posedge Reset) begin
    if (Reset) begin
        Q <= 0; // Reset Q to 0
        Qn <= 1; // Reset Qn to 1
    end else begin
        Q <= D; // Set Q to D on clock edge
        Qn <= ~D; // Set Qn to the inverse of D
    end
end
endmodule //D_FF
```



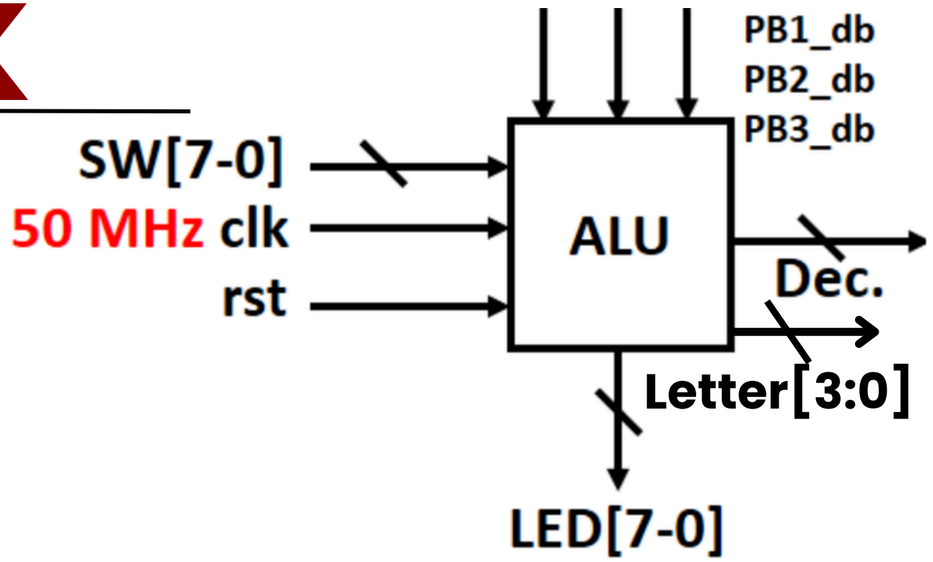

```verilog
module Debouncer (pb_in,Clk,Reset,pb_out);
input pb_in,Clk,Reset;
output pb_out; // Initialize output to 0
wire Q1, Qn2,Q1_and_Qn2; // Outputs from D flip-flops
// Instantiate the clock divider to generate a slower clock for debouncing
D_FF D_FF_1 (.Clk(Clk), .Reset(Reset), .D(pb_in), .Q(Q1), .Qn());
D_FF D_FF_2 (.Clk(Clk), .Reset(Reset), .D(Q1), .Q(), .Qn(Qn2));
and And1(Q1_and_Q2, Q1, Qn2); // AND gate to combine outputs of D flip-flops
or  Or1(pb_out,Q1,Q1_and_Q2);
endmodule //Debouncer

```



ALU BLOCK

An ALU (Arithmetic Logic Unit) is a digital circuit designed to perform arithmetic and bitwise logical operations on binary values. It operates on binary numbers at the bit level for logical procedures and on entire numbers for arithmetic operations.



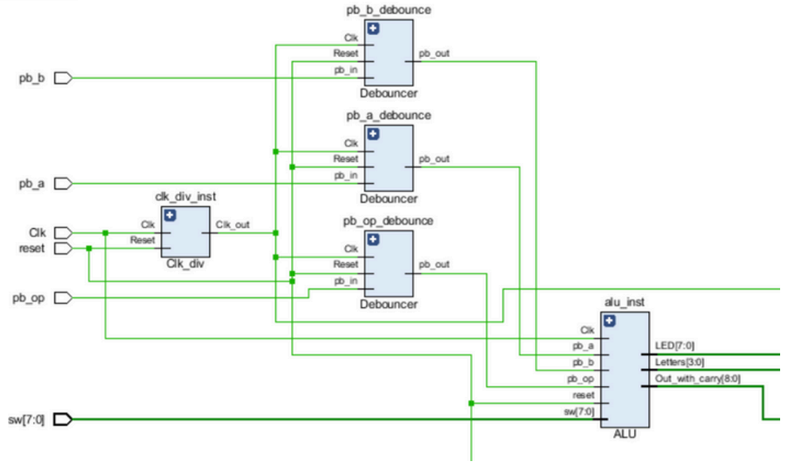
The registers are clocked by the global clock and the loading of these registers is controlled by the push buttons where :

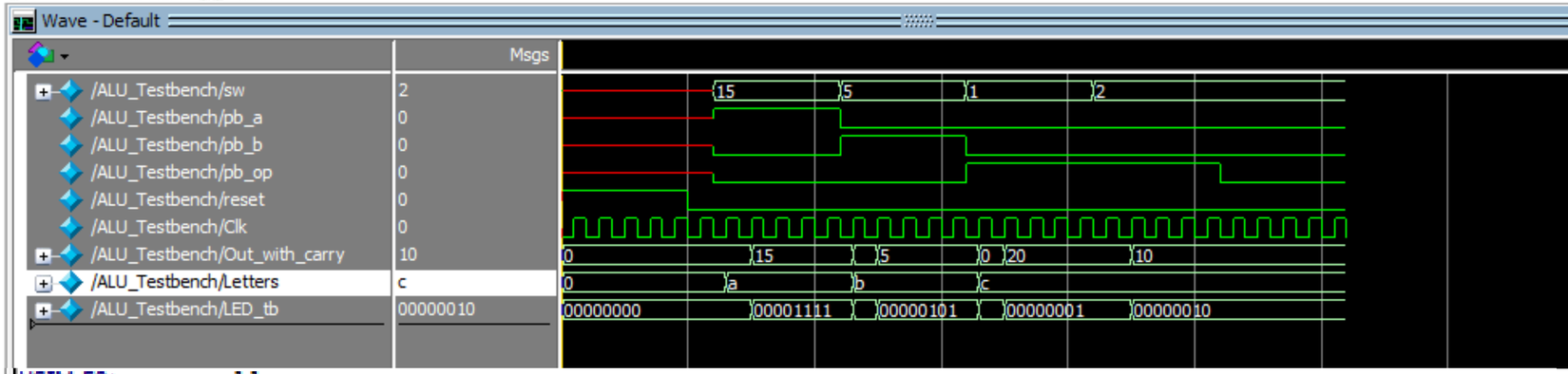
- “Push button 0” is for register A
- “Push button 1” is for register B
- “Push button 2” is for register C
- "MODE" register form SW2, SW1, SW0 controls the function of ALU

ALU Control Signal Code	Operation
001	Addition
010	Subtraction
011	Negation of A
100	Bit-wise AND
101	Bit-wise OR
110	Bit-wise XOR

ALU BLOCK

```
module ALU (
input [7:0]sw,
input pb_a, pb_b, pb_op, reset, Clk,
output reg [8:0]Out_with_carry,
output reg [3:0] Letters,
output reg [7:0] LED
);
reg [7:0] A, B;
reg [2:0] OP_SEL;
always @(posedge Clk or posedge reset) begin
    if (reset) begin
        A <= 8'b0;
        B <= 8'b0;
        OP_SEL <= 3'b0;
        LED <= 8'b0;
        Letters <= 4'd0;
        Out_with_carry <= 9'b0;
    end
    else if (pb_a) begin
        A <= sw;
        LED <=A;
        Out_with_carry <= {1'b0, A };
        Letters <= 4'hA;
    end
    else if (pb_b) begin
        B <=sw;
        LED <=B;
        Out_with_carry <= {1'b0, B };
        Letters <= 4'hB;
    end
    else if (pb_op) begin
      OP_SEL <=sw[2:0];
      LED <= {5'b0,OP_SEL};
      Letters <= 4'hC;
     case (OP_SEL)
        3'b001 : Out_with_carry = A + B;
        3'b010 : Out_with_carry = {1'b0,A - B};
        3'b011 : Out_with_carry = {1'b0, ~A + 1};
        3'b101 : Out_with_carry = {1'b0, A | B};
        3'b110 : Out_with_carry = {1'b0, A ^ B};
    default : Out_with_carry = 9'b0;
    endcase
    end
end
endmodule //ALU
```



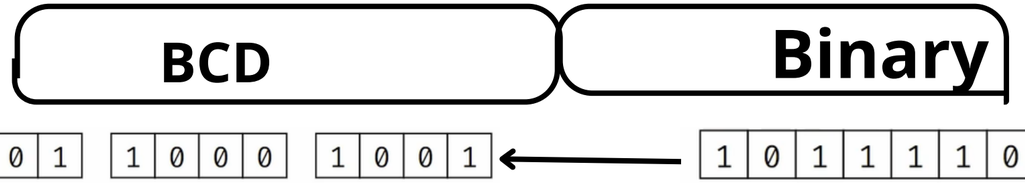



```
VSIM 59> run -all
#  A=  15 ,LED_A:  15 , Out with Carry =  15
#  B=   5 ,LED_B:   5 , Out with Carry =   5
#  Operation:   1 ,LED_OP:   1 , Out with Carry =  20
#  Operation:   2 ,LED_OP:   2 , Out with Carry =  10
# Break in Module ALU_Testbench at C:/Users/mstfy/Desktop/ALU_Project/ALU_Testbench.v line 48
VSIM 60> run

VSIM 61>
```

BINARY TO BCD

BCD is often employed when decimal numbers need to be represented directly in hardware, as each 4-bit BCD digit maps directly to a decimal digit.

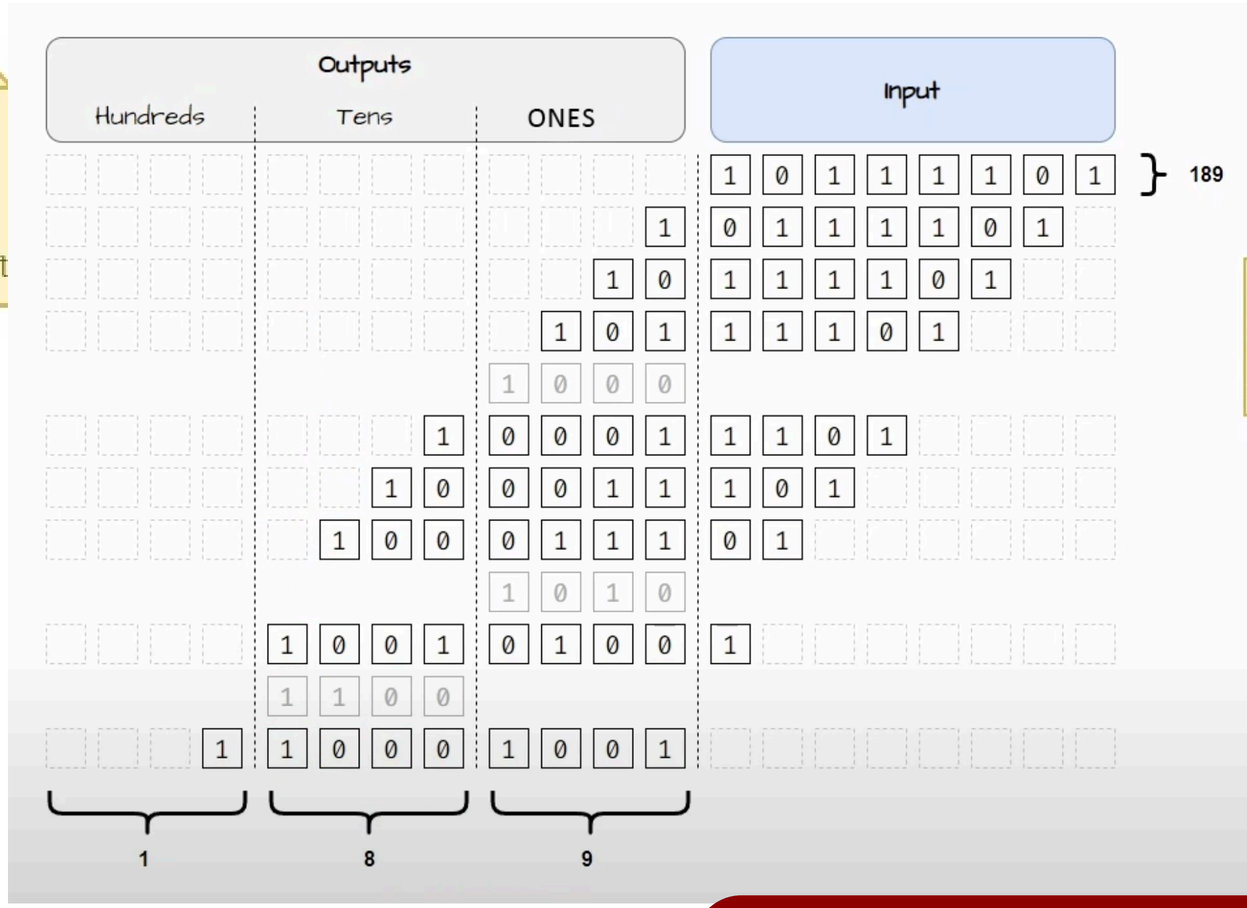



```
1  module DEC_To_BCD(input [8:0]BINARY,output reg [11:0]BCD);
2  integer i;
3  always @(*) begin
4    BCD = 12'b0;
5    for (i=8;i>=0;i=i-1)begin
6      BCD = {BCD[10:0],BINARY[i]};
7    if (i>0)begin
8    if (BCD[3:0]>4) BCD[3:0] = BCD[3:0] + 3;
9    if (BCD[7:4]>4) BCD[7:4] = BCD[7:4] + 3;
10   if (BCD[11:8]>4) BCD[11:8] = BCD[11:8] + 3;
11
12     end
13   end
14
15 end
16 endmodule //DEC_To_BCD
```

DOUBLE DABBLE

1. If any digit is bigger or equal than 5 then add 3

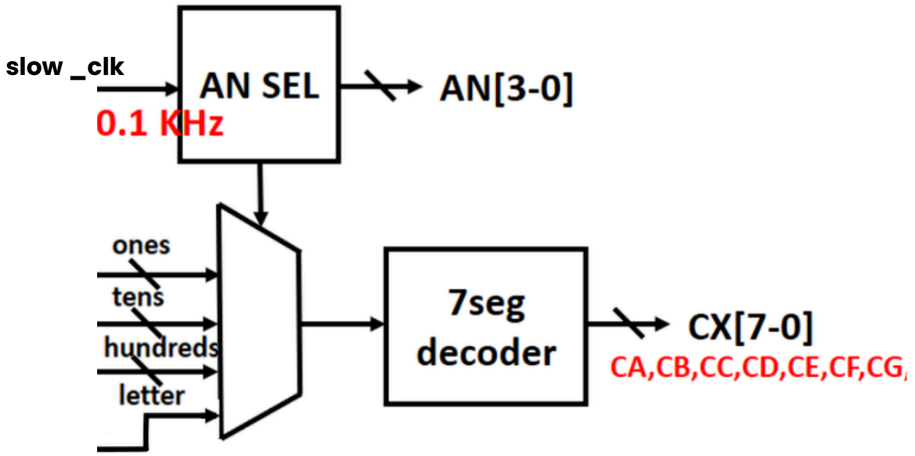
2. Shift all bits one to the left

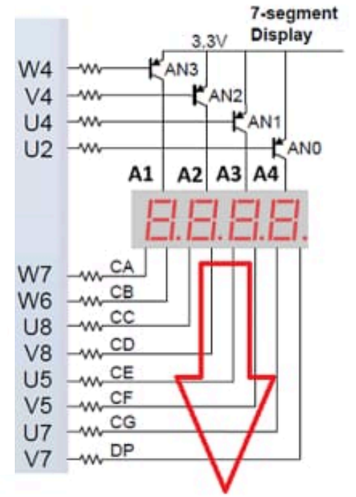


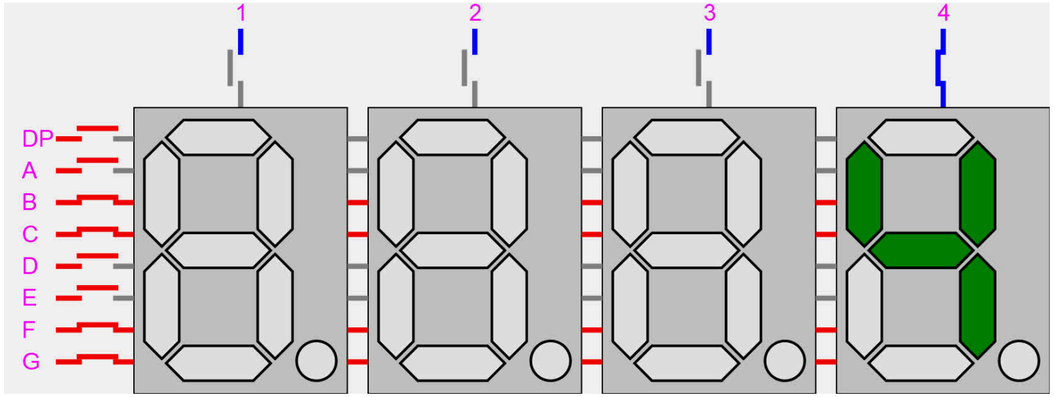

```
VSIM 34> run -all
# binary_num = 292  bcd_num = 0010_1001_0010
# binary_num = 129  bcd_num = 0001_0010_1001
# binary_num =   9  bcd_num = 0000_0000_1001
# binary_num =  99  bcd_num = 0000_1001_1001
# binary_num = 269  bcd_num = 0010_0110_1001
# binary_num = 397  bcd_num = 0011_1001_0111
# binary_num = 101  bcd_num = 0001_0000_0001
# binary_num =  18  bcd_num = 0000_0001_1000
# binary_num = 257  bcd_num = 0010_0101_0111
# binary_num = 269  bcd_num = 0010_0110_1001
# binary_num = 374  bcd_num = 0011_0111_0100
# binary_num = 317  bcd_num = 0011_0001_0111
# binary_num = 493  bcd_num = 0100_1001_0011
# binary_num = 396  bcd_num = 0011_1001_0110
# binary_num = 505  bcd_num = 0101_0000_0101
# binary_num = 198  bcd_num = 0001_1001_1000
# binary_num = 197  bcd_num = 0001_1001_0111
# binary_num = 170  bcd_num = 0001_0111_0000
# binary_num = 485  bcd_num = 0100_1000_0101
# binary_num = 119  bcd_num = 0001_0001_1001
# binary_num =  18  bcd_num = 0000_0001_1000
# binary_num = 399  bcd_num = 0011_1001_1001
# binary_num = 498  bcd_num = 0100_1001_1000
# binary_num = 206  bcd_num = 0010_0000_0110
# binary_num = 232  bcd_num = 0010_0011_0010
# binary_num = 197  bcd_num = 0001_1001_0111
# binary_num = 348  bcd_num = 0011_0100_1000
# binary_num = 189  bcd_num = 0001_1000_1001
# binary_num =  45  bcd_num = 0000_0100_0101
# binary_num = 101  bcd_num = 0001_0000_0001
# Break in Module DEC_To_BCD_testbench at C:/Users/mstfy/Desktop/ALU_Project/DEC_To_BCD_testbench.v line 18
```

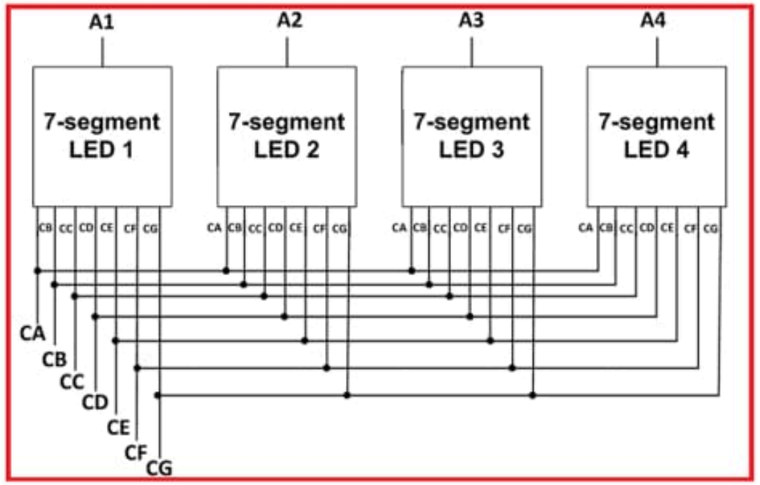
IMPLEMENTATION OF 7-SEGMENT DISPLAY ON FPGA

The diagram shows out approach to implement it. At a particular instant only one of the four 7 segment displays will be active and so on. The frequency for updating will be so fast that we won't be able to notice the switching and it will appear as if it's glowing continuously.







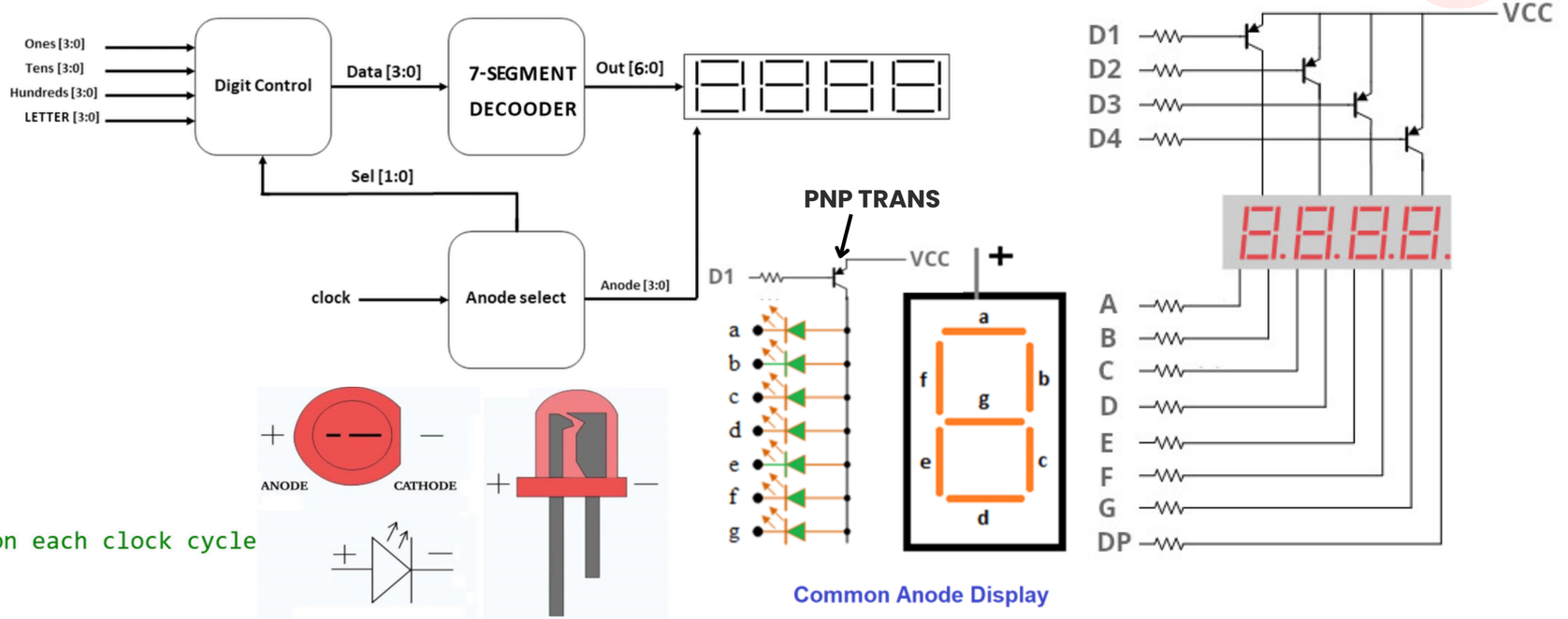


ANODE SELECTOR

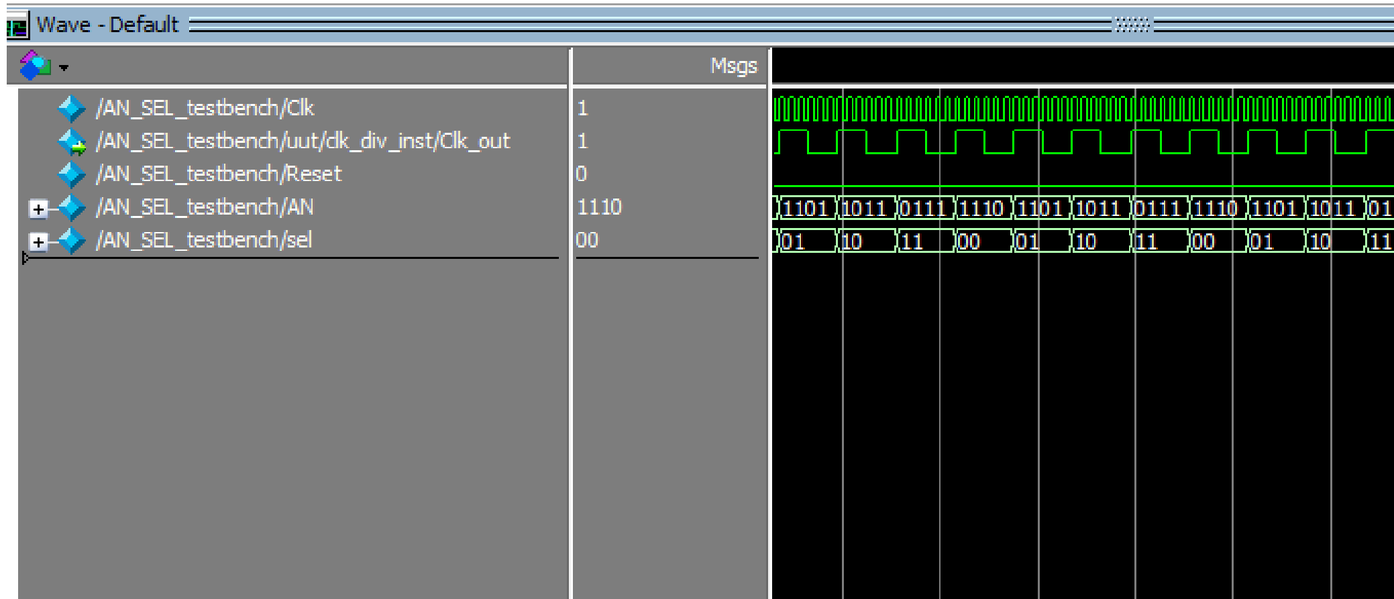
```
module AN_SEL (Clk,Reset,AN,sel);
input Clk, Reset;
output reg [3:0] AN;
output reg [1:0] sel; // 2-bit selector input

always @(posedge Clk or posedge Reset) begin
    if (Reset) begin
        sel <= 2'b00; // Reset selector to 0
    end
    else begin
        sel <= sel + 1; // Increment selector on each clock cycle
    end
    end
    always @(*) begin
        case (sel)
            2'b00: AN = 4'b1110; // Enable first 7_sig for Ones
            2'b01: AN = 4'b1101; // Enable second 7_sig for Tens
            2'b10: AN = 4'b1011; // Enable third 7_sig for Hundreds
            2'b11: AN = 4'b0111; // Enable fourth 7_sig for Letters
            default: AN = 4'b1111; // Disable all 7_seg
        endcase
    end

endmodule //AN_SEL
```

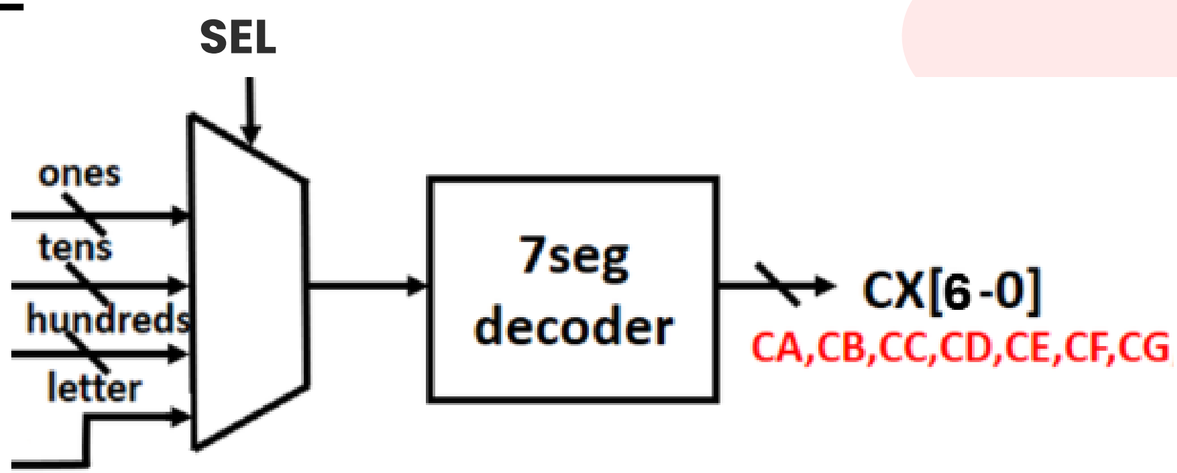


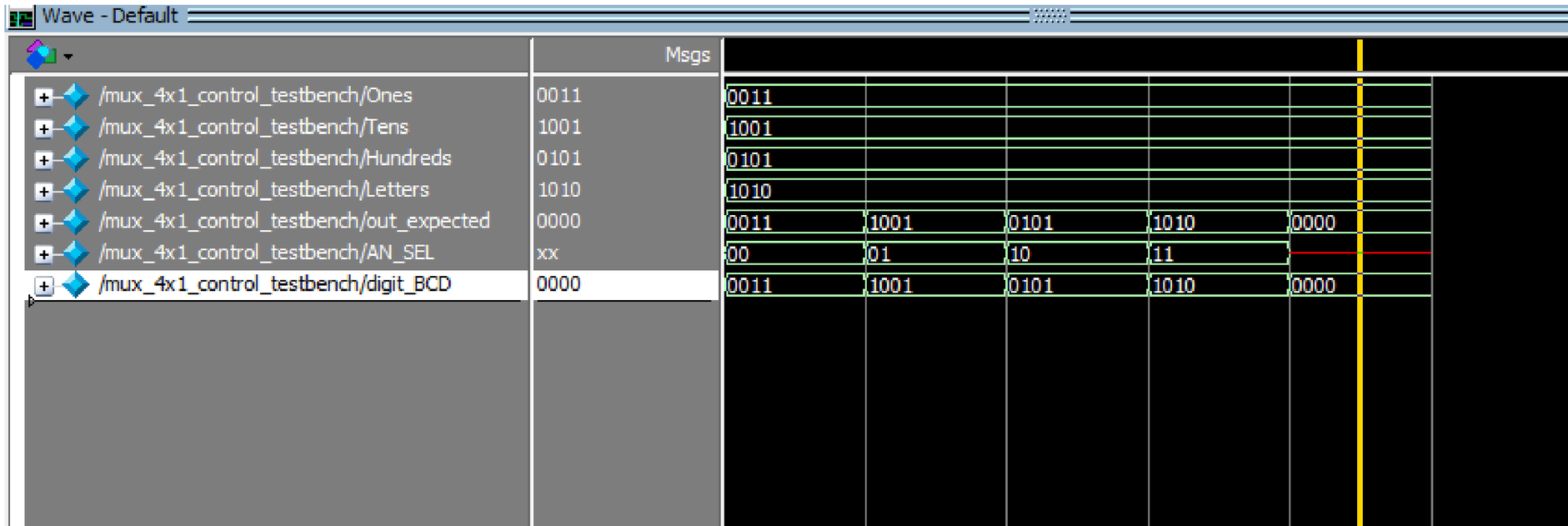
Common Anode Display



MUX 4x1 Control

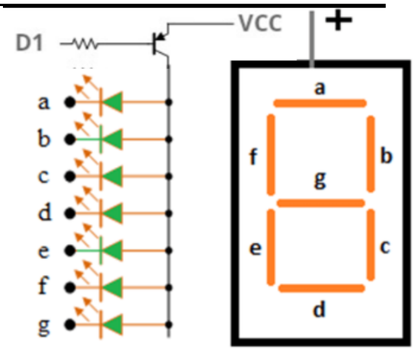
```
1  module mux_4x1_control (Ones, Tens, Hundreds, Letters, AN_SEL,digit_BCD);
2  input [3:0] Ones, Tens, Hundreds, Letters;
3  input [1:0] AN_SEL; // 2-bit AN_SEL input
4  output reg [3:0] digit_BCD; // 4-bit output for BCD
5  always @(*) begin
6      case (AN_SEL)
7          2'b00: digit_BCD = Ones;       // Select Ones
8          2'b01: digit_BCD = Tens;       // Select Tens
9          2'b10: digit_BCD = Hundreds;    // Select Hundreds
10         2'b11: digit_BCD = Letters;     // Select Letters
11         default: digit_BCD = 4'b0000;  // Default case to avoid latches
12     endcase
13 end
14
15 endmodule //mux_4x1_control
```



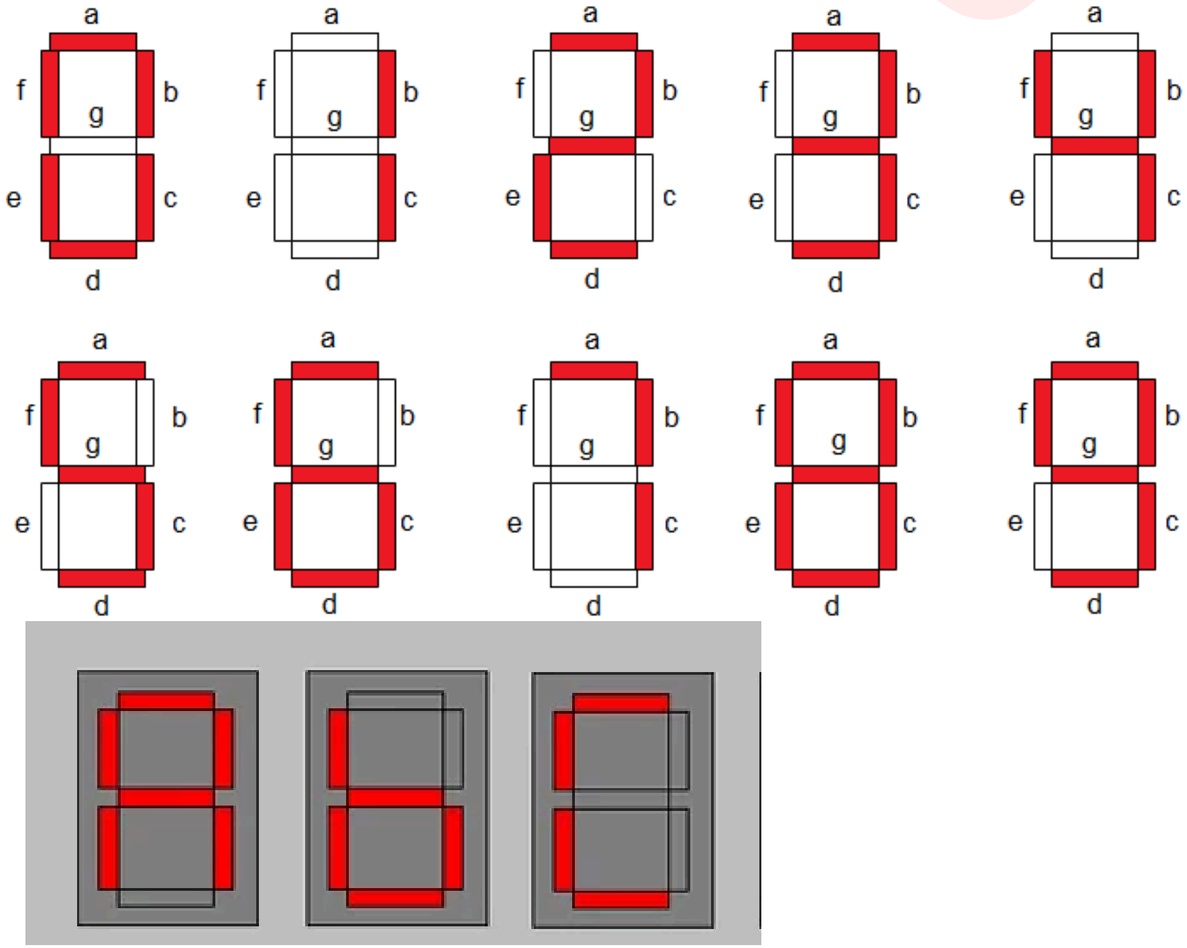


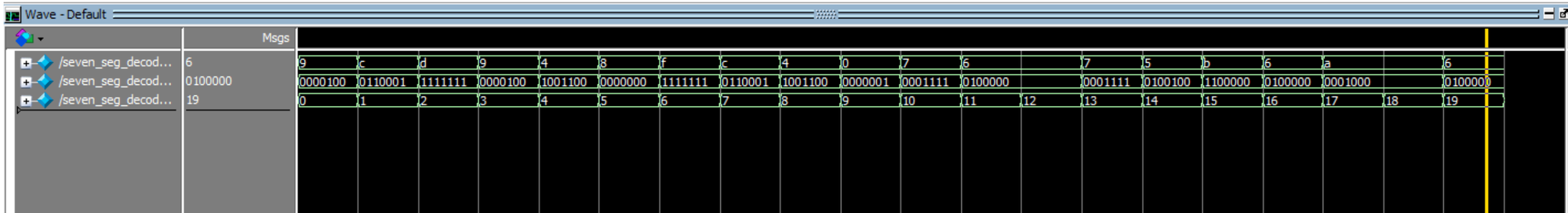
7 seg decooder

```
1  module seven_seg_decode (digit_BCD,seven_seg);
2   input [3:0] digit_BCD;
3   output reg [6:0] seven_seg;
4       always @(*) begin
5           case (digit_BCD)
6               4'd0: seven_seg = 7'b0000001;
7               4'd1: seven_seg = 7'b1001111;
8               4'd2: seven_seg = 7'b0010010;
9               4'd3: seven_seg = 7'b0000110;
10              4'd4: seven_seg = 7'b1001100;
11              4'd5: seven_seg = 7'b0100100;
12              4'd6: seven_seg = 7'b0100000;
13              4'd7: seven_seg = 7'b0001111;
14              4'd8: seven_seg = 7'b0000000;
15              4'd9: seven_seg = 7'b0000100;
16              4'hA: seven_seg = 7'b0001000;
17              4'hB: seven_seg = 7'b1100000;
18              4'hC: seven_seg = 7'b0110001;
19            default: seven_seg = 7'b1111111;
20          endcase
21      end
22  endmodule //seven_seg_decode
```

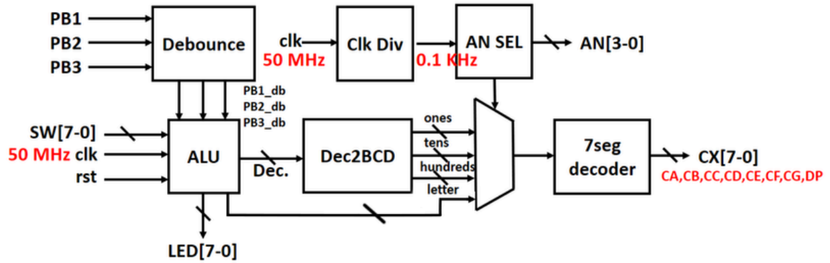


Common Anode Display





TOP Module




```
module ALU_TOP (pb_a, pb_b, pb_op, reset,Clk,sw,LED,AN_SEL,seven_seg_out);

input pb_a, pb_b, pb_op, reset, Clk;
input [7:0] sw;

output  [3:0] AN_SEL;
output  [6:0] seven_seg_out;
output  [7:0] LED;

wire [11:0] BCD_out;
wire [8:0] Out_with_carry;
wire [3:0] seven_seg_in;
wire [3:0] Letters;
wire [1:0] sel_in;
wire pb_a_debounced, pb_b_debounced, pb_op_debounced,slow_clk;


// Clk Divider for slower clock signal
Clk_div #(.counter_div(25'd4)) clk_div_inst (.Clk(Clk), .Reset(reset), .Clk_out(slow_clk));

//Instatiate debounce modules for buttons and switches

Debouncer pb_a_debounce (.Clk(slow_clk), .Reset(reset), .pb_in(pb_a), .pb_out(pb_a_debounced));
Debouncer pb_b_debounce (.Clk(slow_clk), .Reset(reset), .pb_in(pb_b), .pb_out(pb_b_debounced));
Debouncer pb_op_debounce(.Clk(slow_clk), .Reset(reset), .pb_in(pb_op), .pb_out(pb_op_debounced));

//Instantiate ALU module
ALU alu_inst (
    .sw(sw),
    .pb_a(pb_a_debounced),
    .pb_b(pb_b_debounced),
    .pb_op(pb_op_debounced),
    .reset(reset),
    .Clk(Clk),
    .Out_with_carry(Out_with_carry),
    .Letters(Letters),
    .LED(LED)
);

// Instantiate DEC_To_BCD module
DEC_To_BCD dec_to_bcd_inst (.BINARY(Out_with_carry),.BCD(BCD_out));

// Instantiate AN_SEL module
AN_SEL an_sel_inst(.Clk(slow_clk), .Reset(reset), .AN(AN_SEL), .sel(sel_in));

// Instantiate mux_control model
mux_4x1_control mux_control_inst (
    .Ones(BCD_out[3:0]),
    .Tens(BCD_out[7:4]),
    .Hundreds(BCD_out[11:8]),
    .Letters(Letters),
    .AN_SEL(sel_in),
    .digit_BCD(seven_seg_in)
);

// Instantiate seven_seg_decode module
seven_seg_decode seven_seg_decode_inst (
    .digit_BCD(seven_seg_in),
    .seven_seg(seven_seg_out)
);

endmodule //ALU_TOP
```

TOP Module TB

```
module ALU_TOP_testbench ();

    reg pb_a_tb, pb_b_tb, pb_op_tb, reset_tb, Clk_tb
    reg [7:0] sw_tb;

    wire [7:0] LED_tb;
    wire [3:0] AN_SEL_tb;
    wire [6:0] seven_seg_out_tb;
    integer i;

    // Instantiate DUT
    ALU_TOP dut (
        .pb_a(pb_a_tb),
        .pb_b(pb_b_tb),
        .pb_op(pb_op_tb),
        .sw(sw_tb),
        .reset(reset_tb),
        .Clk(Clk_tb),
        .AN_SEL(AN_SEL_tb),
        .LED(LED_tb),
        .seven_seg_out(seven_seg_out_tb)
    );

    // Clock generation
  initial begin
  Clk_tb = 0;
  forever #10 Clk_tb = ~Clk_tb;// 50 MHz clock
end
```

```
initial begin

     reset_tb = 1;
   #200 reset_tb = 0;
    // Test case 1: Set A
    pb_b_tb = 0; pb_op_tb = 0;
    sw_tb = 8'b00001111; // Set A to 15
    for (i =0 ;i<15 ;i=i+1 ) begin
    pb_a_tb=1;
    #20;
    pb_a_tb=~pb_a_tb;
    #20;
   end
    pb_a_tb=1;
   #550;
    pb_a_tb = 0;
     // Test case 2: Set B
    #500 ;
    sw_tb= 8'b00000101; // Set B to 5
    #20;
    for (i =0 ;i<15 ;i=i+1 ) begin
    pb_b_tb=1;
    #20;
    pb_b_tb=~pb_b_tb;
    #20;
   end
   pb_b_tb=1;
   #500;
    pb_b_tb = 0;
```

```
        pb_b_tb = 0;
        // Test case 3: Perform addition
        #500
        sw_tb= 8'b000000001; // Set operation to additio
        #20;
          for (i =0 ;i<15 ;i=i+1 ) begin
        pb_op_tb=1;
        #20;
        pb_op_tb=~pb_op_tb;
        #20;
       end
       pb_op_tb=1;
       #550;
       pb_op_tb = 0;
        #2000
        $stop;
    end

endmodule
```

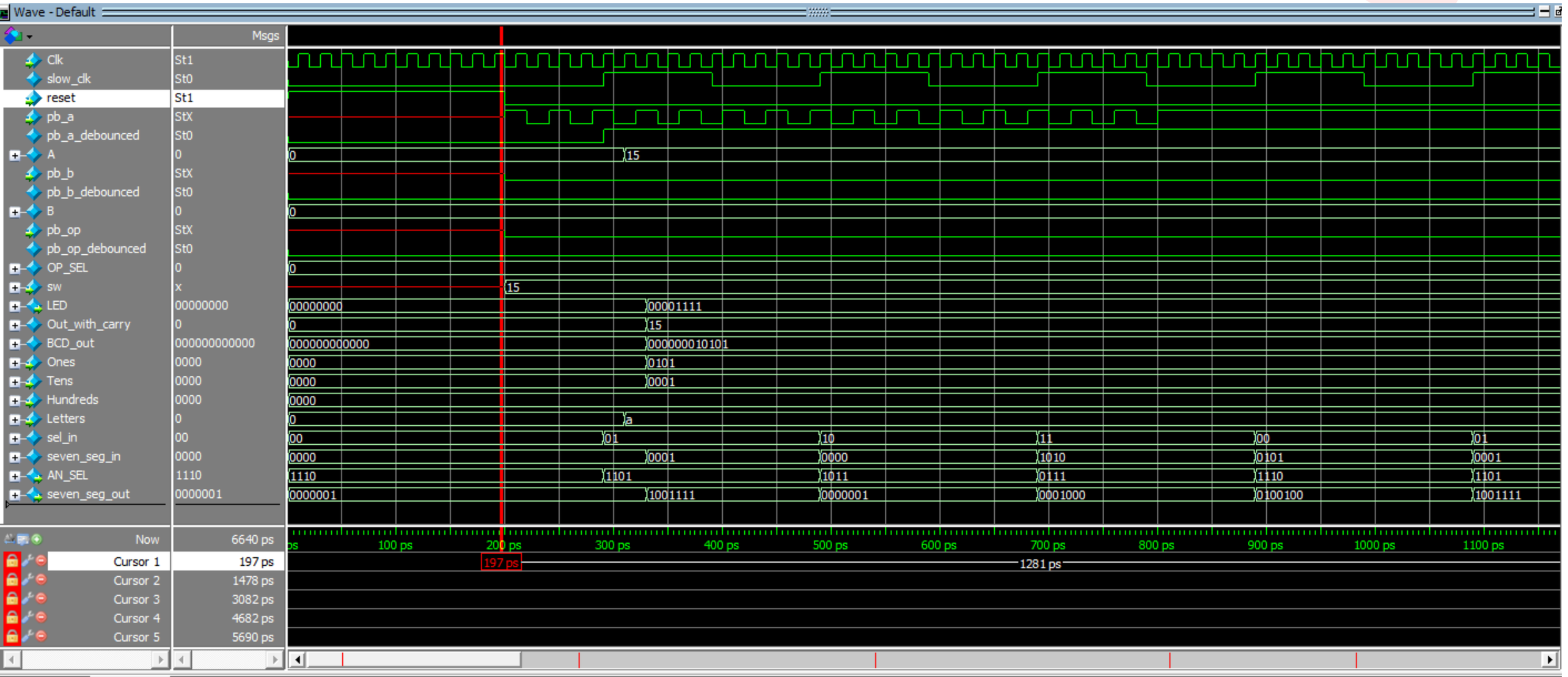
TOP Module TB

Do file

```
vlib work
vlog ALU.v
vlog Clk_div.v
vlog D_FF.v
vlog Debouncer.v
vlog DEC_To_BCD.v
vlog AN_SEL.v
vlog mux_4x1_control.v
vlog seven_seg_decode.v
vlog ALU_TOP.v
vlog ALU_TOP_testbench.v
vsim -voptargs=+acc work.ALU_TOP_testbench
add wave *
run -all
#quit -sim
```

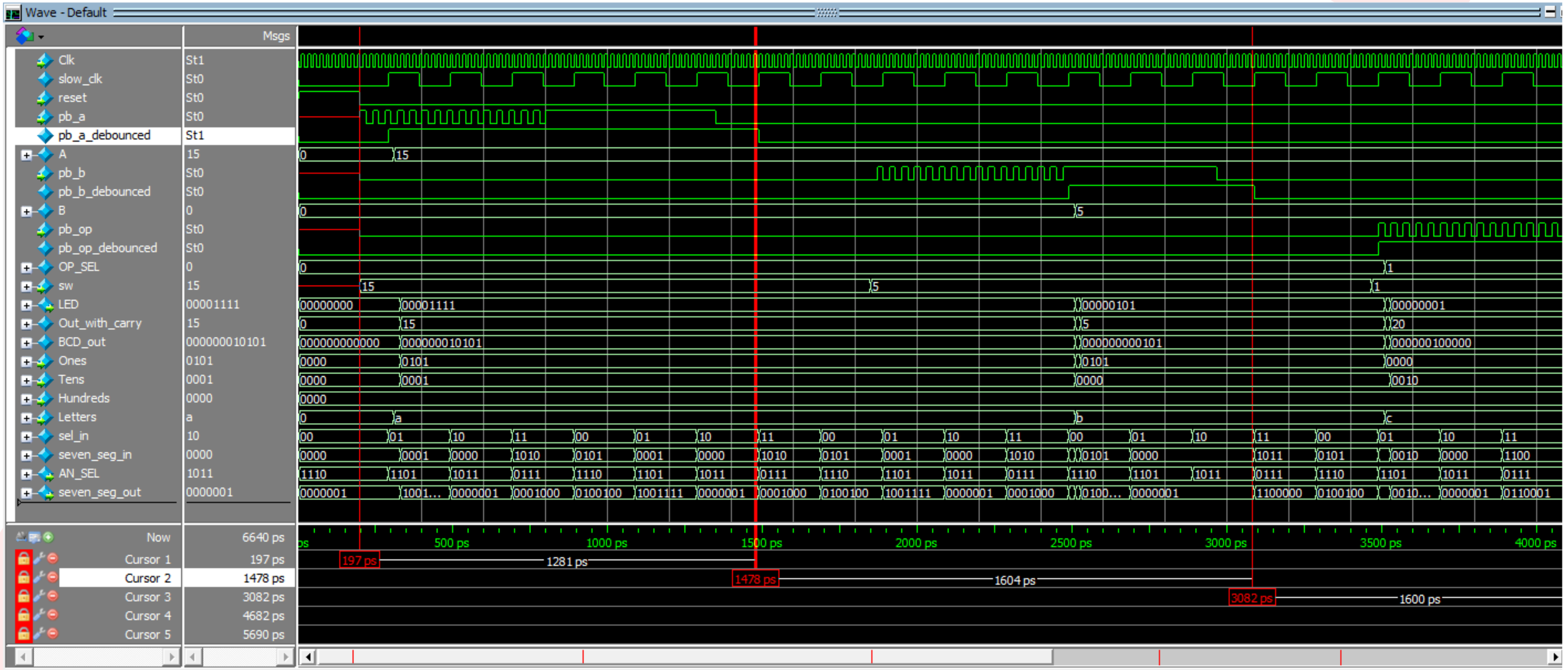
TOP Module TB

Reset On



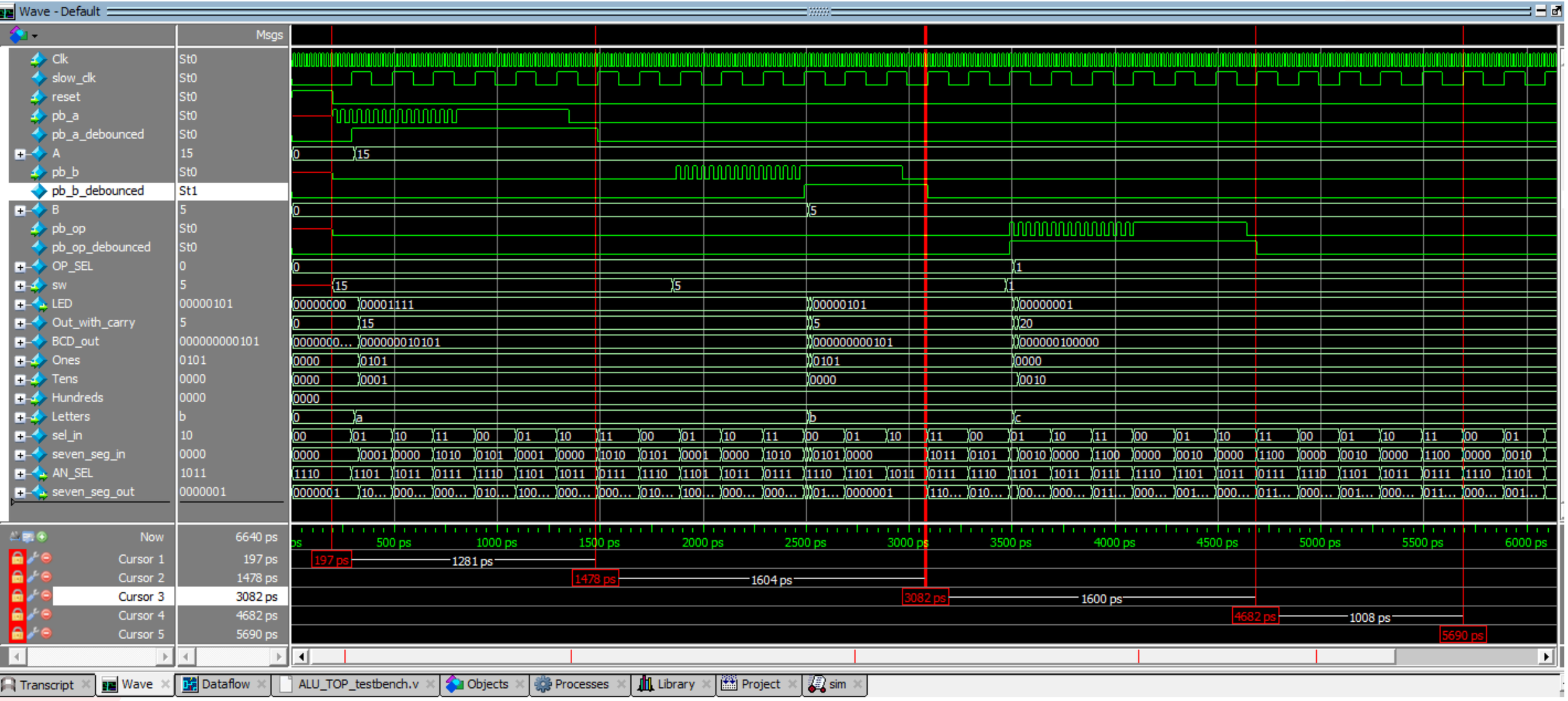
TOP Module TB

Pb_A On



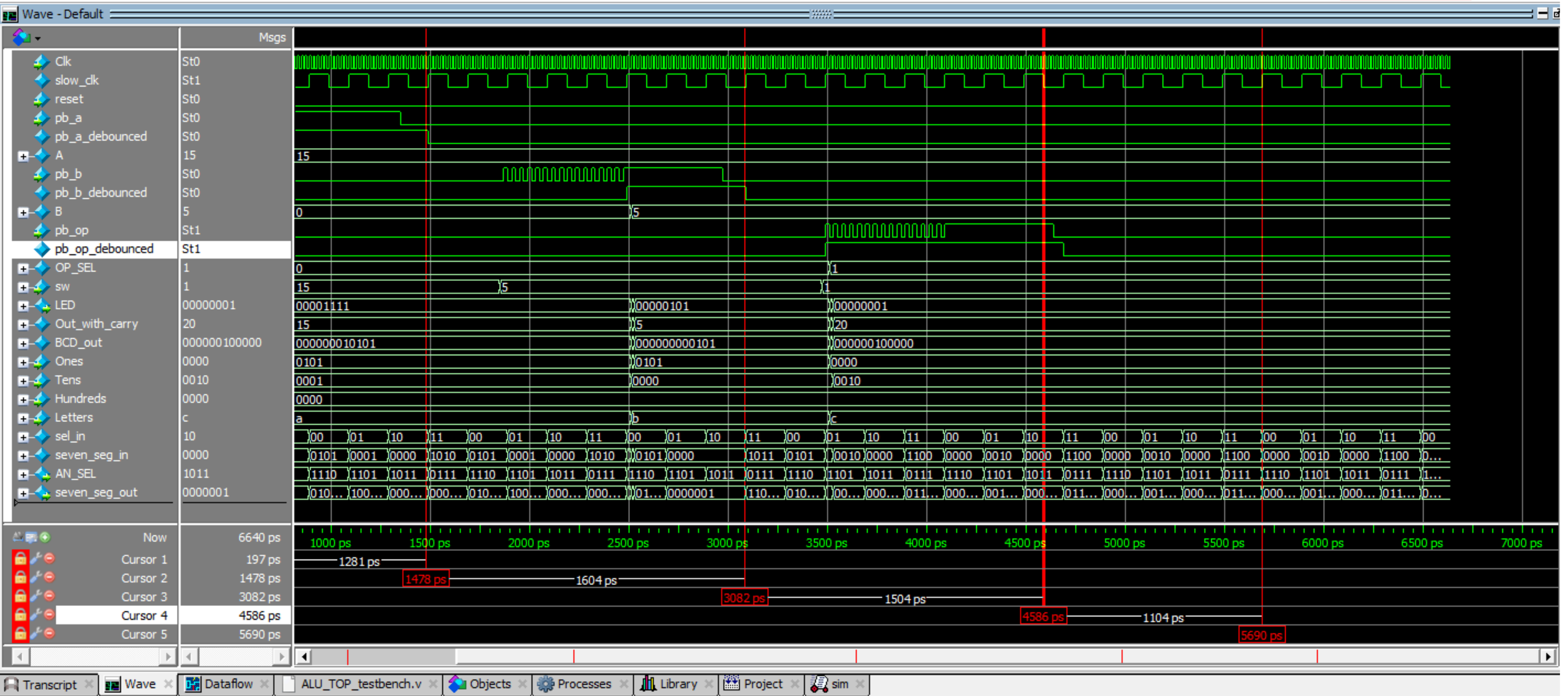
TOP Module TB

Pb_B On



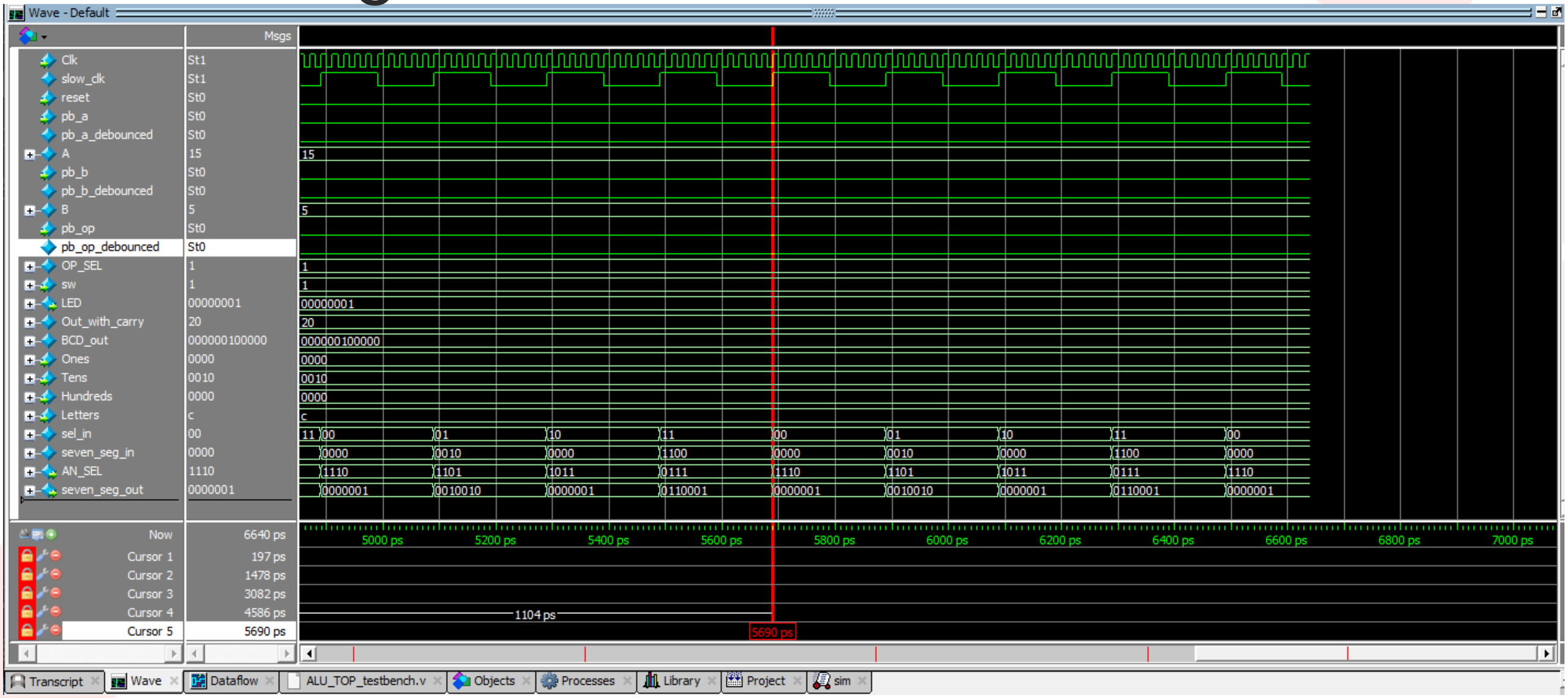
TOP Module TB

Pb_OP On



TOP Module TB

seven_seg_out



Conclusion

The full project, including all source files, is available on GitHub: ALU_Project

Scan QR

THANK YOU