DIGITAL DESIGN

LAB6   COMBINATORIAL CIRCUIT

2022 SUMMER TERM

LAB6

- Verilog
  - Non-blocking assignment , RHS , LHS
  - Intra-statement delay vs Inter-statement delay
  - Sequential block vs Parallel block

- Combinatorial circuit
  - 1bit full adder , 2bits full adder
  - Lighting 7 segment digital tube

- Practice

PRACTICE 1 IN LAB5

```verilog
module testswap( );
    reg temp=0, in2, in1, clock;
    initial begin
        clock = 1'b0;
        forever #50 clock = ~clock;
    end
    initial $display("temp <= in1;     in1 <= in2;     in2 <= temp;");
    always@(posedge clock)
    begin
        $display("=================================================================");
        $display($time, "(display)Before swap:\tin1 = %d, in2 = %d, temp = %d", in1, in2, temp);
        $strobe($time, "(strobe )After swap: \tin1 = %d, in2 = %d, temp = %d", in1, in2, temp);
        temp <= in1;     in1 <= in2;     in2 <= temp;
        $display($time, "(display)After swap: \tin1 = %d, in2 = %d, temp = %d", in1, in2, temp);
    end

    initial begin
        {in1, in2 } = 2'b00;
        repeat(3)
            #100 {in1, in2} = {in1, in2} + 1;
        #100 $finish(0);
    end
endmodule
```

```
temp <= in1;     in1 <= in2;     in2 <= temp;

        50(display)Before swap:   in1 = 0, in2 = 0, temp = 0
        50(display)After swap:    in1 = 0, in2 = 0, temp = 0
        50(strobe )After swap:    in1 = 0, in2 = 0, temp = 0

        150(display)Before swap:  in1 = 0, in2 = 1, temp = 0
        150(display)After swap:   in1 = 0, in2 = 1, temp = 0
        150(strobe )After swap:   in1 = 1, in2 = 0, temp = 0

        250(display)Before swap:  in1 = 1, in2 = 1, temp = 0
        250(display)After swap:   in1 = 1, in2 = 1, temp = 0
        250(strobe )After swap:   in1 = 1, in2 = 0, temp = 1

        350(display)Before swap:  in1 = 1, in2 = 1, temp = 1
        350(display)After swap:   in1 = 1, in2 = 1, temp = 1
        350(strobe )After swap:   in1 = 1, in2 = 1, temp = 1
$finish called at time : 400 ns
```

RHS calculate execute at the very beginning, $strobe $monitor execute at end, LHS just before the $strobe and $monitor

INTRA-STATEMENT DELAY VS INTER-STATEMENT DELAY

- **Intrastatement delay**: happened in a statement

  - such as : A = **#10** 1'b0; //get RHS, delay 10 ,then assign to A

- **Interstatement delay:** happened before the execution of the statement

  - such as**:** **#10** A=1'b0;   //delay 10, get RHS ,then assignment to A
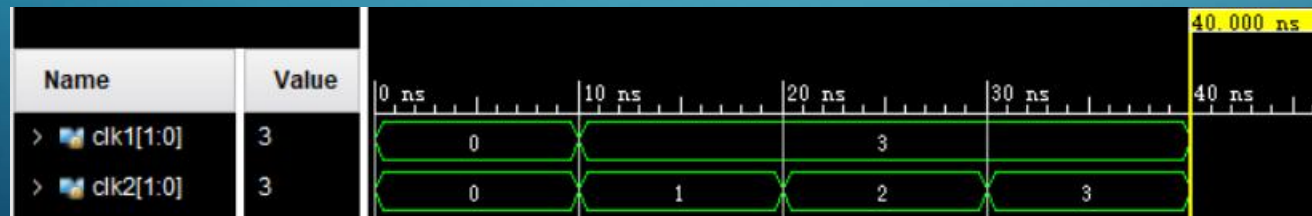

```
module test_delay():
    reg [1:0]clk1, clk2:
    initial
    {clk1, clk2} = 4'h0:

    initial
    begin
        #10 clk2 <=2'd1:
        #10 clk2 <=2'd2:
        #10 clk2 <=2'd3:
    end

    initial
        begin
            clk1 <= #10 2'd1:
            clk1 <= #10 2'd2:
            clk1 <= #10 2'd3:
        end

    initial
        #40 $finish(1):
endmodule
```

SEQUENTIAL BLOCK VS PARALLEL BLOCK

```verilog
module block();
    reg [1:0] x, y, z;
    initial
        begin
            #10 x = 2'd0;
            #10 x = 2'd1;
            #10 x = 2'd2;
            #10 x = 2'd3;
        end

    initial
        fork
            #10 y = 2'd0;
            #10 y = 2'd1;
            #10 y = 2'd2;
            #10 y = 2'd3;
        join

    initial
        fork
            #10 z = 2'd0;
            #20 z = 2'd1;
            #30 z = 2'd2;
            #40 z = 2'd3;
        join

    initial
        #50 $finish(1);
endmodule
```
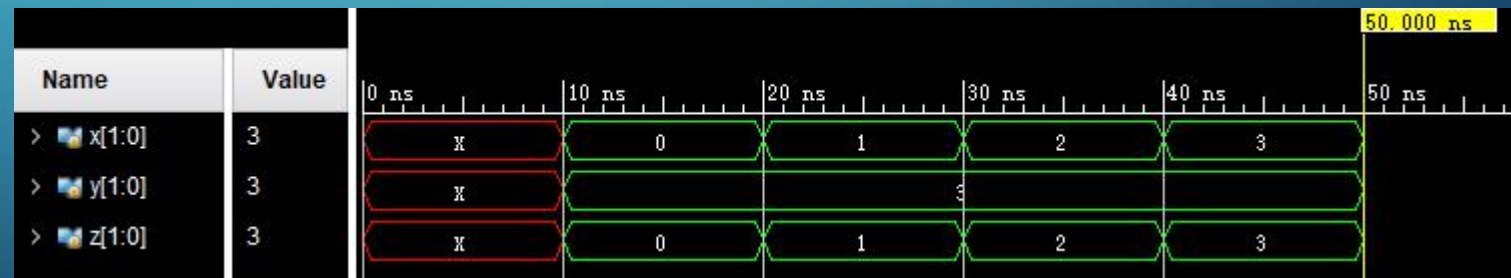
- In one module all the block executes at the same time(time 0)
- Sequential block(begin … end): ------synthesizable
  - all the statements (except unblock assignment) in one sequential block executes with the order of writing.
- Parallel block(fork … join): ------Not synthesizable
  - all the statement in one parallel block executes at same time

# 1BIT FULL ADDER VS MULTI-BITS FULL ADDER

```verilog
module full_add_1b(a, b, cin, sum, cout);
input  a, b, cin;
output  sum, cout;
assign {cout, sum}=a+b+cin;
endmodule
```

```verilog
module full_add_2b(a, b, cin, sum, cout);
input [1:0]a, b;
input cin;
output[1:0]  sum;
output cout;

wire cout1;
full_add_1b u0(a[0], b[0], cin, sum[0], cout1);
full_add_1b u1(a[1], b[1], cout1, sum[1], cout);
endmodule
```
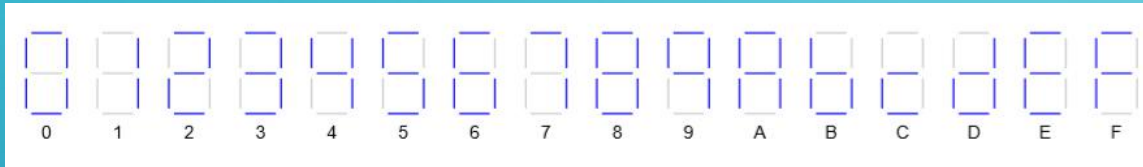
```verilog
module full_add_sm(    );
reg scin;
reg [1:0]sa, sb;
wire [1:0]ssum;
wire scout;
full_add_2b u2(sa, sb, scin, ssum, scout);
initial
{scin, sa, sb} = 5'd0;

initial
begin
    repeat(31)
    #10 {scin, sa, sb} ={scin, sa, sb} +1;
end
endmodule
```
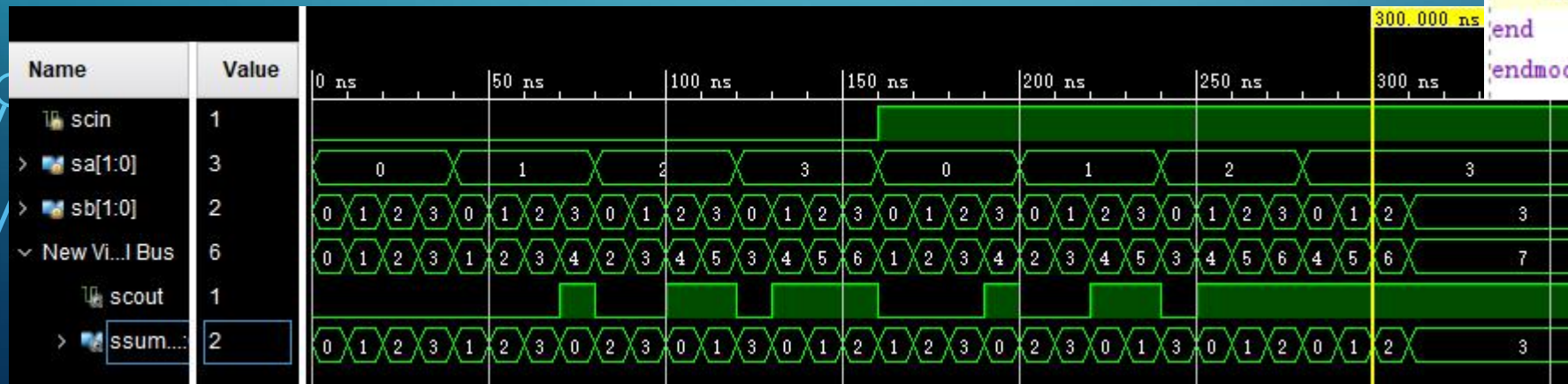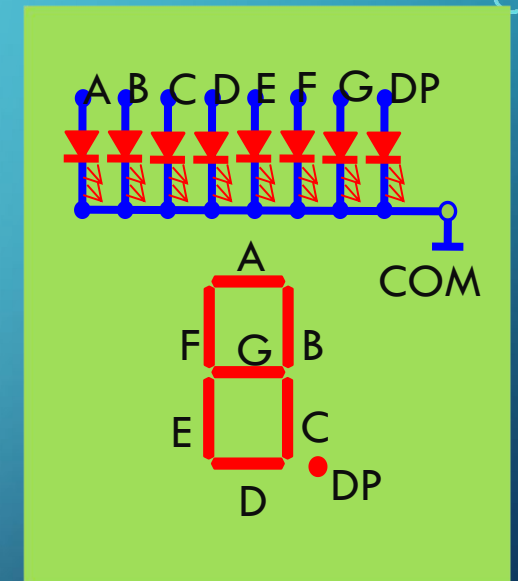
LIGHTING A 7-SEG-TUBE

common cathodes

| DN0				DN1			
DK1	DK2	DK3	DK4	DK5	DK6	DK7	DK8
CA0 CB0 CC0 CD0 CE0 CF0 CG0 DP0				CA1 CB1 CC1 CD1 CE1 CF1 CG1 DP1			

LIGHTING A 7-SEG-TUBE

- There is an 4-bits width binary number , show its hexadecimal number.

- Using EGO1 develop board

- 4*dial switch are the inputs while 8*7 seg-tubes are the outputs

LIGHTING A 7-SEG-TUBE (DESIGN)

```
assign seg_en = 8'hff;
always @ *
begin
    case(sw)    //A B C D E F G DP
        4'h0:  seg_out = 8'b1111_1100;   //0
        4'h1:  seg_out = 8'b0110_0000;   //1
        4'h2:  seg_out = 8'b1101_1010;   //2
        4'h3:  seg_out = 8'b1111_0010;   //3
        4'h4:  seg_out = 8'b0110_0110;   //4
        4'h5:  seg_out = 8'b1011_0110;   //5
        4'h6:  seg_out = 8'b1011_1110;   //6
        4'h7:  seg_out = 8'b1110_0000;   //7

        4'h8:  seg_out = 8'b1111_1110;   //8
        4'h9:  seg_out = 8'b1110_0110;   //9
        4'ha:  seg_out = 8'b1110_1110;   //a
        4'hb:  seg_out = 8'b0011_1110;   //b
        4'hc:  seg_out = 8'b0001_1010;   //c
        4'hd:  seg_out = 8'b0111_1010;   //d
        4'he:  seg_out = 8'b1001_1110;   //e
        4'hf:  seg_out = 8'b1000_1110;   //f

        default: seg_out = 8'b0000_0000;   //all disabled
    endcase
end
```
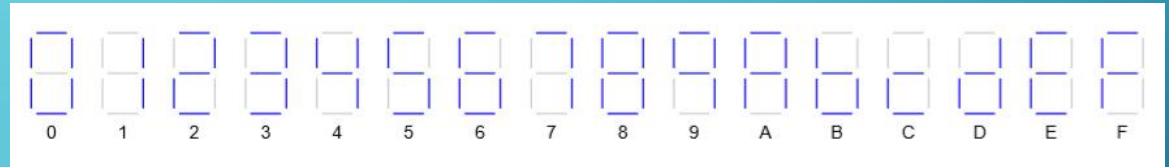

- Eight 7-seg-tubes, each one has a enable pin, while it's 1 it enables the corresponding tube, while it's 0 it disables the tube.

- While 'seg_en' is 8'hff, it means enable all the tubes.

- From DP to CA ~CG, all are positive enable.

LIGHTING A 7-SEG-TUBE (CONSTRAINTS FILE)

- I/O Planning: choose LVCMOS33*

- Use SW7, SW6, SW5, SW4 as sw[3:0] input pins

- Select pins on eight 7-seg tubes
  - Use DK1, DK2, DK3, DK4 as seg_en[7:0] output pins

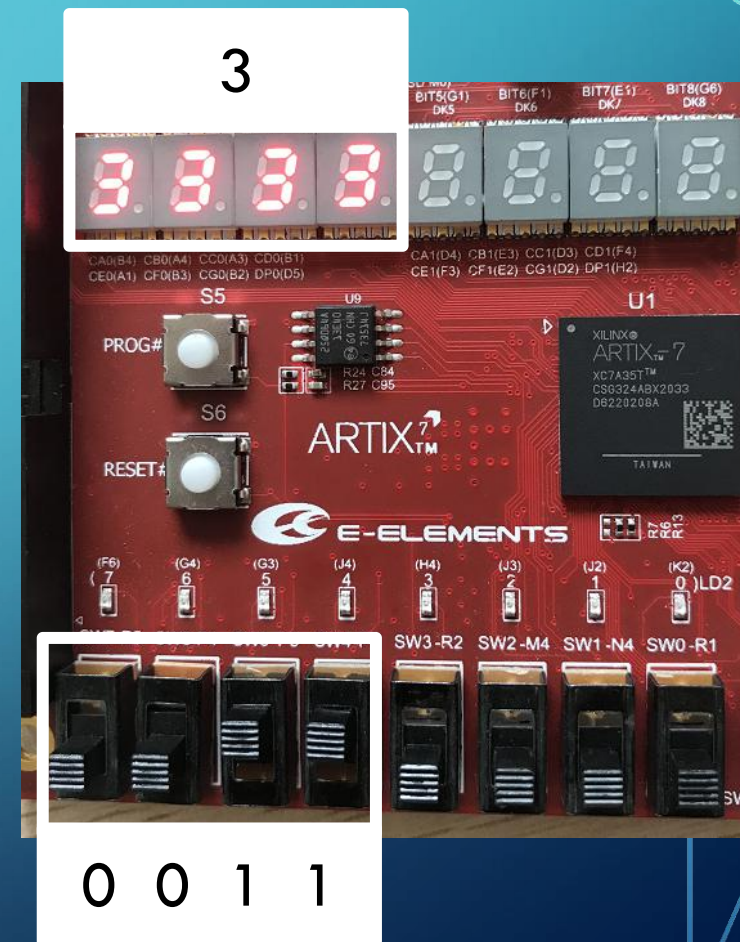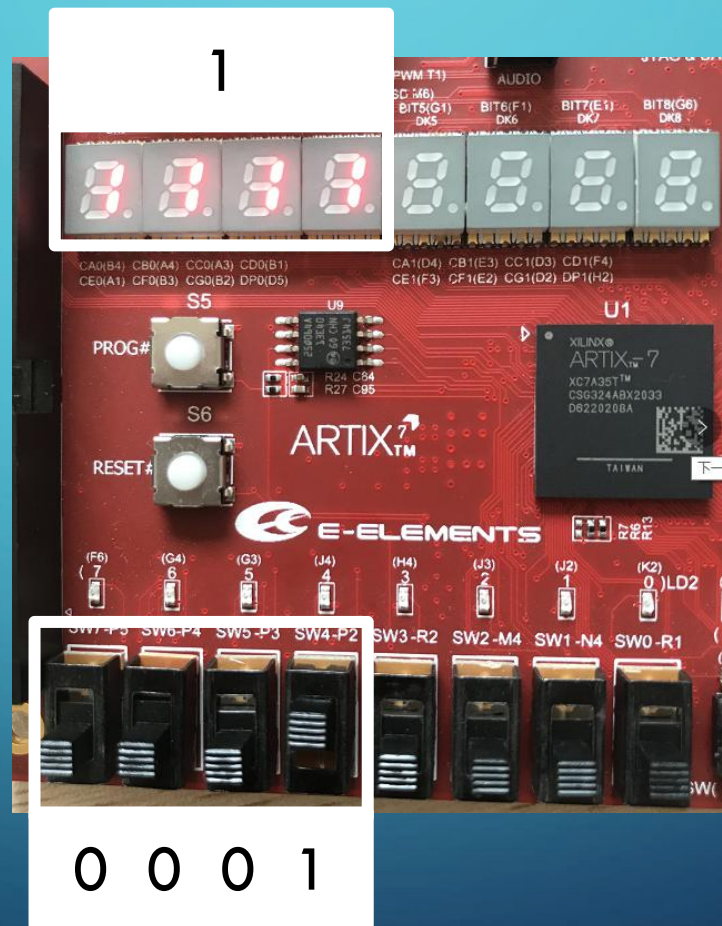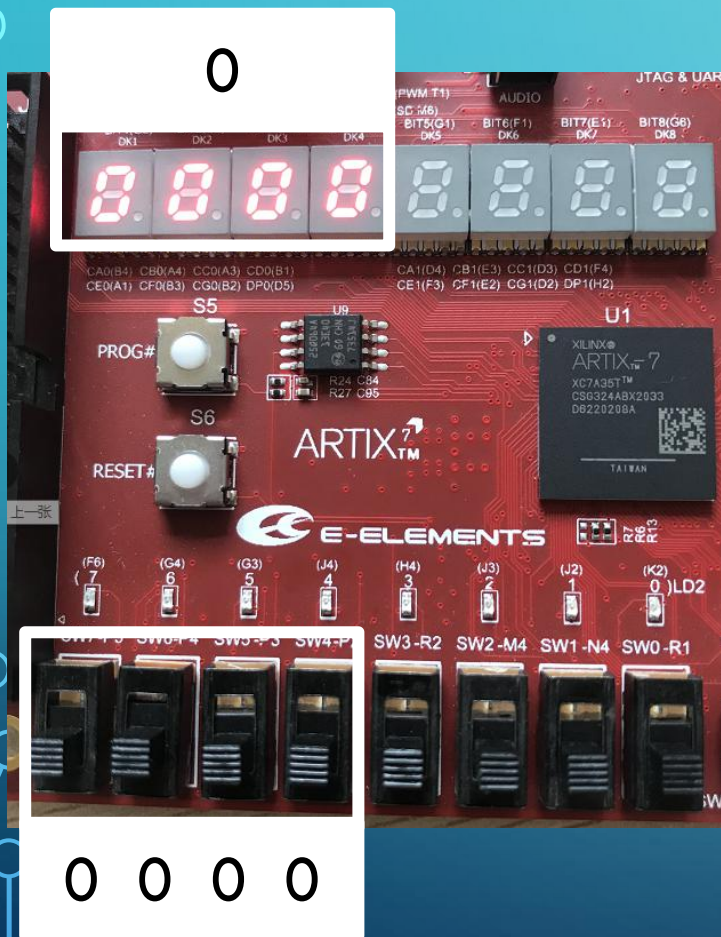- Use CA0, CB0, CC0, CD0, CE0, CF0, CG0, DP0 as seg_out[7:0] output pins

LVCMOS33* ▼

sw (4)	
sw[3]	P5
sw[2]	P4
sw[1]	P3
sw[0]	P2

seg_en (8)	
seg_en[7]	G2
seg_en[6]	C2
seg_en[5]	C1
seg_en[4]	H1
seg_en[3]	G1
seg_en[2]	F1
seg_en[1]	E1
seg_en[0]	G6

seg_out (8)	
seg_out[7]	B4
seg_out[6]	A4
seg_out[5]	A3
seg_out[4]	B1
seg_out[3]	A1
seg_out[2]	B3
seg_out[1]	B2
seg_out[0]	D5

LIGHTING A 7-SEG-TUBE (TESTING RESULT)

PRACTICE 1

- In the 7-seg tub demo, the other 4 tubes are not drived, please complement the demo to control them with the switches.
    - Do the design and verify the function of your design.
    - Create the constrain file, do the synthesis and implementation, generate the bitstream file and program the device, then test on the EGO1 develop board.
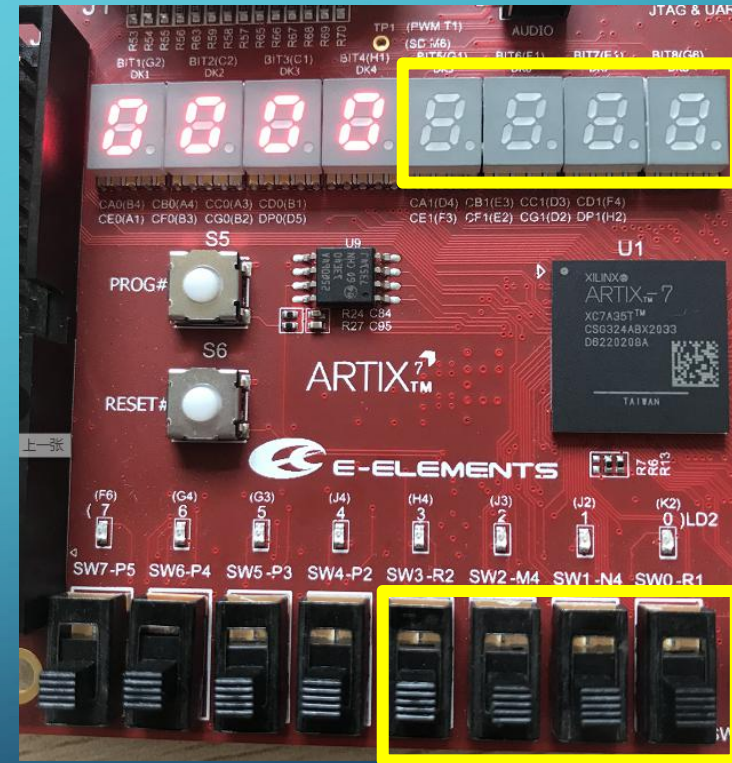
PRACTICE 2

- There are sixteen wards, which are numbered from 0 to F respectively, among which #0 ward has the lowest priority, #F has the highest priority(Priority increases as the number increases). Each room has an call bell, it could be turn on or turn off. In the main control room there is a display screen which shows the ID of the room whose bell is on with highest priority of all the bell on room.

- Please write a circuit to implement this function and test.
  - Do the design and verify the function of your design.
  - Create the constrain file, do the synthetic and implementation, generate the bitstream file and program the device, then test on the EGO1 develop board.