Digital Logic

Lab5 Combinatorial circuit: structur design based on sub-modules, IOs:7-seg tub

Lab5

- IOs on EGO1
  - input： switch
  - output: led, **7-seg tube**
- Verilog
  - behavioral description: case, **casex, casez**
  - **structure design**
    - based on primitve gates
    - **based on sub-modules**
- Practice

Recall: Behavioral modeling if/else, case

An **always** block can include **a sensitivity list** in which any of these signals change will trigger the always block execution.

- The data type of the assigned object MUST be reg
- 'if else' and 'case' could ONLY be used as part of 'initial' or 'always'
- 'if else' VS conditional operator VS 'case'

```
reg o1, o2, o3;
always @(*)
begin
    if(p == q)
        {o1, o2, o3} = 3'b100;
    else if (p < q)
        {o1, o2, o3} = 3'b010;
    else
        {o1, o2, o3} = 3'b001;
end
```

```
reg o1, o2, o3;
always @*
    {o1, o2, o3} = (p==q) ? 3'b100 : (p<q) ? 3'b010 : 3'b001;
```

```
reg o1, o2, o3;
always @(p, q)
begin
    $display("{p, q} = %d", {p, q});
    case({p, q})
        4'b0000, 4'b0101, 4'b1010, 4'b1111:
            {o1, o2, o3} = 3'b100;
        4'b0001, 4'b0010, 4'b0011, 4'b0110, 4'b0111, 4'b1011:
            {o1, o2, o3} = 3'b010;
        default:
            {o1, o2, o3} = 3'b001;
    endcase
end
```

Behavioral modeling: case

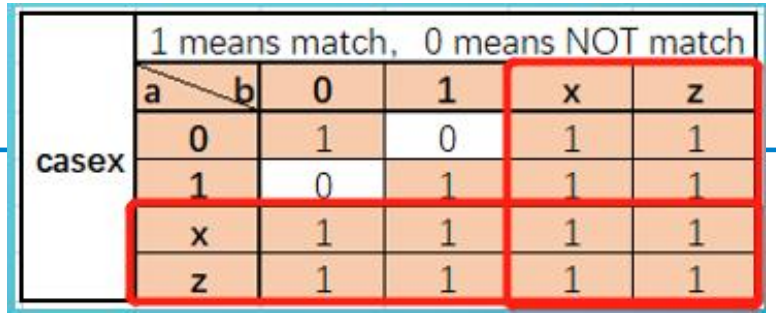
p		q		o1(p==q)	o2(p<q)	o3(p>q)
0	0	0	0	1		
0	0	0	1		1	
0	0	1	0		1	
0	0	1	1		1	
0	1	0	0			1
0	1	0	1	1		
0	1	1	0		1	
0	1	1	1		1	
1	0	0	0			1
1	0	0	1			1
1	0	1	0	1		
1	0	1	1		1	
1	1	0	0			1
1	1	0	1			1
1	1	1	0			1
1	1	1	1	1		

truth table for 2-bit comparator

case	1 means match, 0 means NOT match				
	a \ b	0	1	x	z
	0	1	0	0	0
	1	0	1	0	0
	x	0	0	1	0
	z	0	0	0	1

```
reg o1, o2, o3;
  always @(p, q)
    begin
      $display("{p, q} = %d", {p,q});
      case({p,q})
        4'b0000, 4'b0101, 4'b1010, 4'b1111:
            {o1, o2, o3} = 3'b100;
        4'b0001, 4'b0010, 4'b0011, 4'b0110, 4'b0111, 4'b1011:
            {o1, o2, o3} = 3'b010;
        default:
            {o1, o2, o3} = 3'b001;
      endcase
    end
```

Behavioral modeling: casex




```
reg o1, o2, o3;
always @(p, q)
  begin
    $display("{p, q} = %d", {p, q});
    case({p, q})
      4'b0000, 4'b0101, 4'b1010, 4'b1111:
          {o1, o2, o3} = 3'b100;
      4'b0001, 4'b0010, 4'b0011, 4'b0110, 4'b0111, 4'b1011:
          {o1, o2, o3} = 3'b010;
      default:
          {o1, o2, o3} = 3'b001;
    endcase
  end
```

```
casex({p,q})
      4'b0000,4'b0101,4'b1010,4'b1111:
            {o1, o2, o3} = 3'b100;
      4'b0001, 4'b001x, 4'b011x, 4'b1011:
            {o1, o2, o3} = 3'b010;
      defalut:   {o1, o2, o3} = 3'b001;
endcase
```

```
casex({p,q})
      4'b0000,4'b0101,4'b1010,4'b1111:
            {o1, o2, o3} = 3'b100;
      4'b0001, 4'b001z, 4'b011z, 4'b1011:
            {o1, o2, o3} = 3'b010;
      defalut:   {o1, o2, o3} = 3'b001;
endcase
```

Behavioral modeling: casz

```
reg o1, o2, o3;
  always @(p, q)
  begin
    $display("{p, q} = %d", {p,q});
    case({p,q})
      4'b0000, 4'b0101, 4'b1010, 4'b1111:
          {o1, o2, o3} = 3'b100;
      4'b0001, 4'b0010, 4'b0011, 4'b0110, 4'b0111, 4'b1011:
          {o1, o2, o3} = 3'b010;
      default:
          {o1, o2, o3} = 3'b001;
    endcase
  end
```

casez

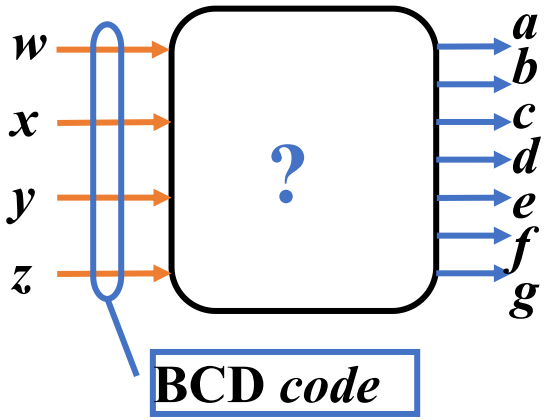
a \ b	0	1	x	z
0	1	0	0	1
1	0	1	0	1
x	0	0	1	1
z	1	1	1	1

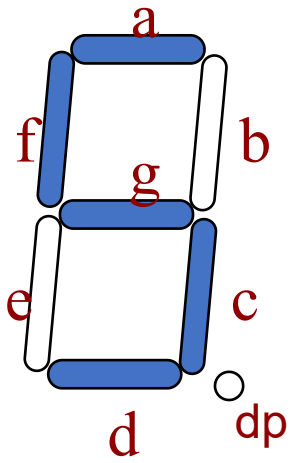
1 means match, 0 means NOT match

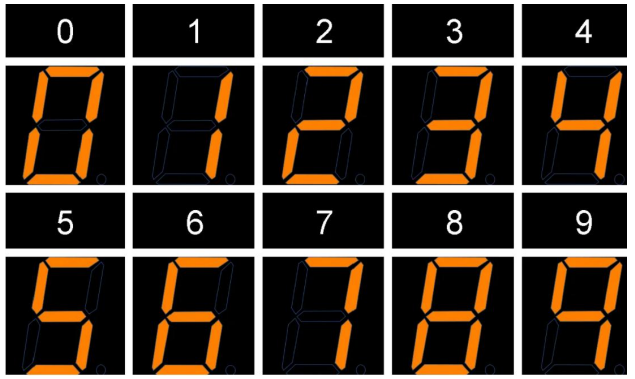
```
casez({p,q})
      4'b0000,4'b0101,4'b1010,4'b1111:
          {o1, o2, o3} = 3'b100;
      4'b0001, 4'b001z, 4'b011z, 4'b1011:
          {o1, o2, o3} = 3'b010;
      defalut:  {o1, o2, o3} = 3'b001;
endcase
```

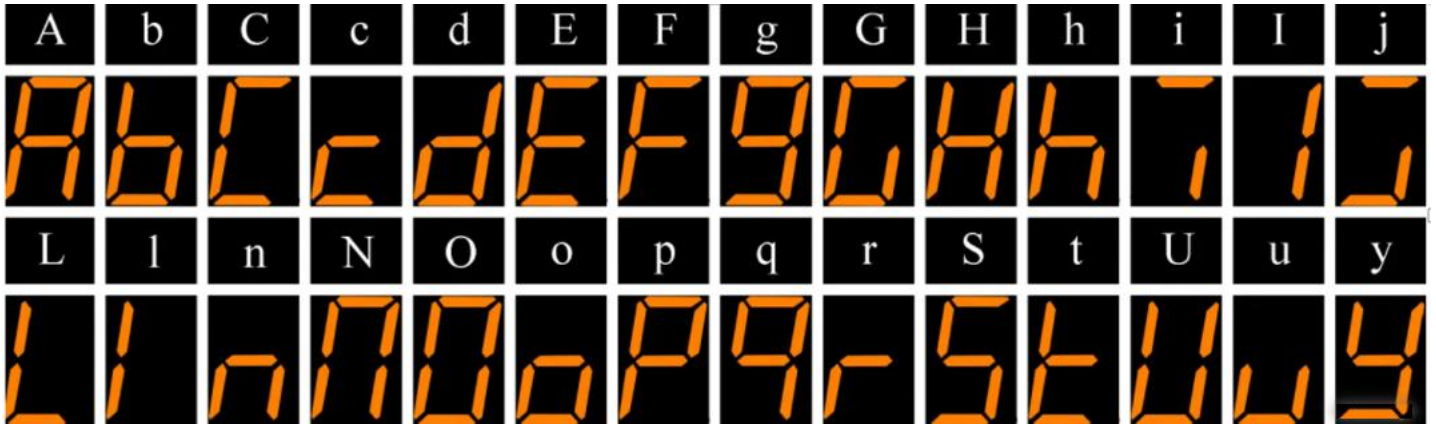
7-Segment Display Decoder

BCD Input	7-Segment Decoder
w x y z	a b c d e f g
0 0 0 0	1 1 1 1 1 1 0
0 0 0 1	0 1 1 0 0 0 0
0 0 1 0	1 1 0 1 1 0 1
0 0 1 1	1 1 1 1 0 0 1
0 1 0 0	0 1 1 0 0 1 1
0 1 0 1	1 0 1 1 0 1 1
0 1 1 0	1 0 1 1 1 1 1
0 1 1 1	1 1 1 0 0 0 0
1 0 0 0	1 1 1 1 1 1 1
1 0 0 1	1 1 1 1 0 1 1
1 0 1 0	x x x x x x x
1 0 1 1	x x x x x x x
1 1 0 0	x x x x x x x
1 1 0 1	x x x x x x x
1 1 1 0	x x x x x x x
1 1 1 1	x x x x x x x





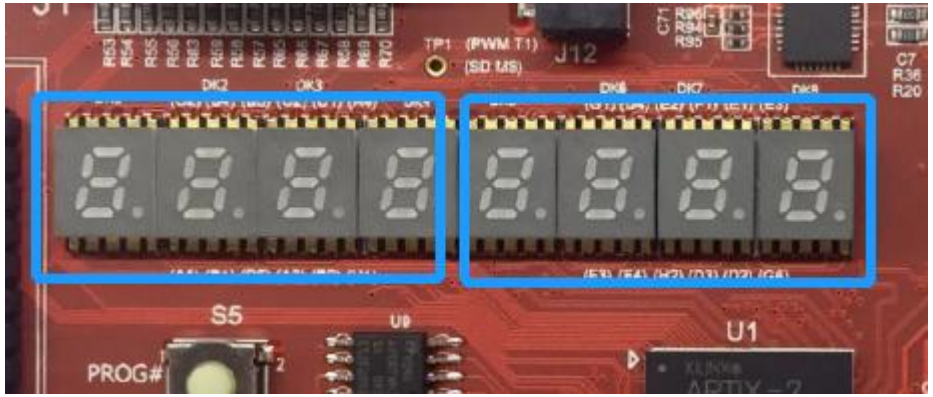


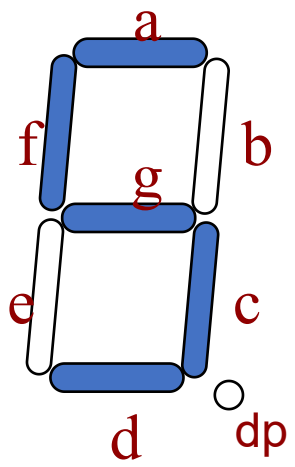


7-Segment Display Decoder on EGO1

The 8 seven-segment tubes are divided into two groups, with each group corresponding to pins A0-G0 and A1-G1 respectively.

- Each individual 7-segment display is controlled by a chip selection signal, and the corresponding display can only work when the chip selection signal is set to 1.
  - For example, if you want the leftmost display to show “5”
    - 1) set the corresponding chip selection signal to 1
    - 2) set the segments of a, c, d, f, and g to high level, and set b, e and dp to low level.





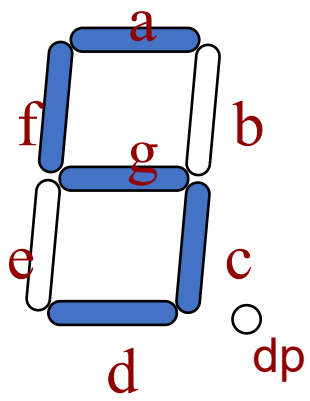
constraint:

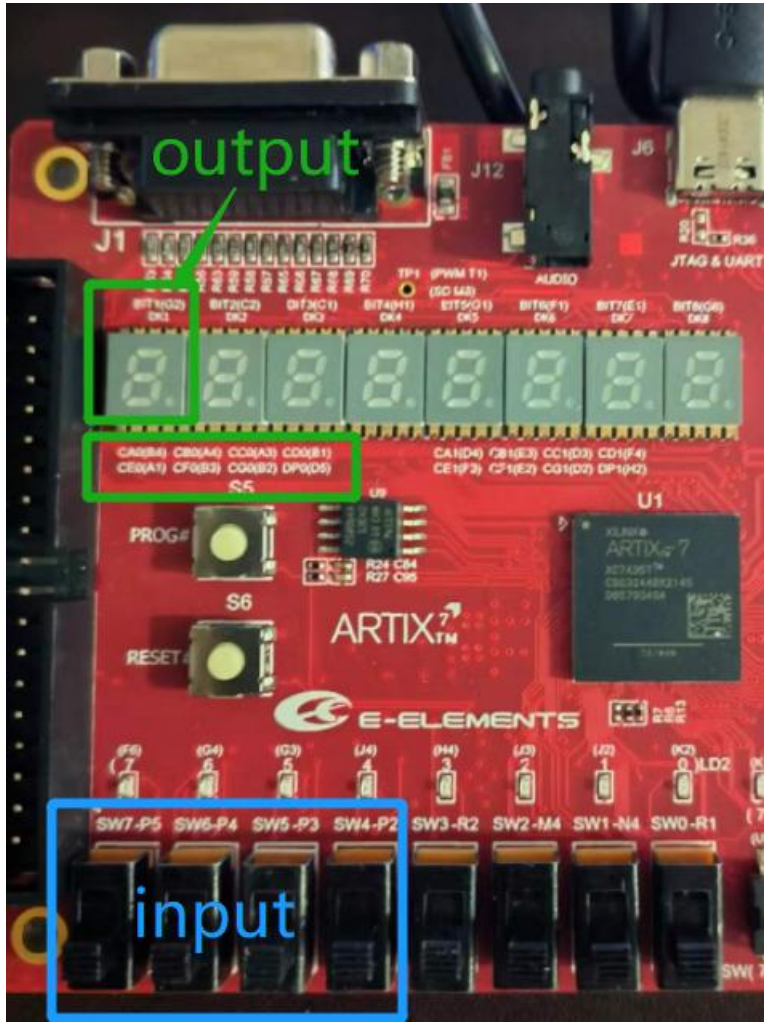
名称	原理图标号	FPGA IO PIN
A0	CA0	B4
B0	CB0	A4
C0	CC0	A3
D0	CD0	B1
E0	CE0	A1
F0	CF0	B3
G0	CG0	B2
DP0	DP0	D5
A1	CA1	D4
B1	CB1	E3
C1	CC1	D3
D1	CD1	F4
E1	CE1	F3
F1	CF1	E2
G1	CG1	D2
DP1	DP1	H2
DN0_K1	BIT1	G2
DN0_K2	BIT2	C2
DN0_K3	BIT3	C1
DN0_K4	BIT4	H1
DN1_K1	BIT5	G1
DN1_K2	BIT6	F1
DN1_K3	BIT7	E1
DN1_K4	BIT8	G6

lab5_demo1：BCD2Decimal_show(1)

Design a circuit with the function of **displaying the decimal number corresponding to the current input on a 7-segment tube when the input is a 4-bit BCD code**. **When the input is not a BCD code, display the letter E on the 7-segment tube.**

```
module lab5_demo1(
        input [3:0] in_b4,
        output tub_sel,
        output reg [7:0]  tub_control
    );
    assign tub_sel = 1'b1;

    // tub_control : abcb_efg"dot"
endmodule
```



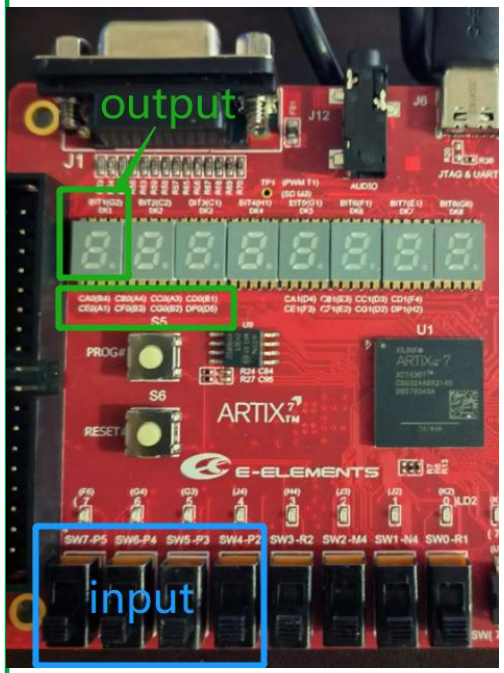


Tips: in this design

1. **tub_sel** is used to make the selected 7-segment tube work or not;
2. **tub_control** is used to control the appearance of the 7-segment tube display: tub_control[7] : a , tub_control[6] : b , tub_control[5] : c .... tub_control[2] : f , tub_control[1] : g, tub_control[0] : dot point(on the right bottom of the 7-segment tub".

lab5_demo1：BCD2Decimal_show(2)

- To determine using which input devices and output devices, then build the constraint file(.xdc)
  - Here we choose using the 7-seg-tub on the left top as the output, and the switches on the left buttom as the input



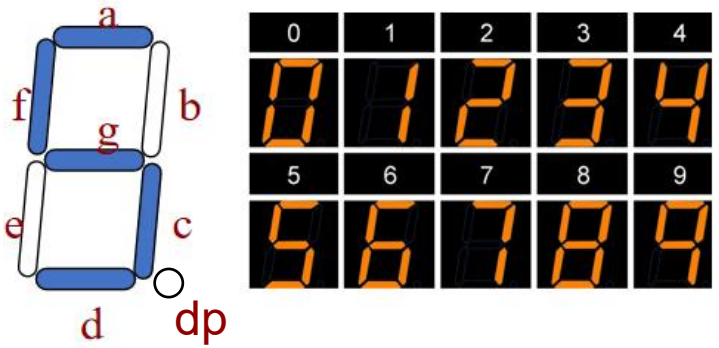
名称	原理图标号	FPGA IO PIN
A0	CA0	B4
B0	CB0	A4
C0	CC0	A3
D0	CD0	B1
E0	CE0	A1
F0	CF0	B3
G0	CG0	B2
DP0	DP0	D5
A1	CA1	D4
B1	CB1	E3
C1	CC1	D3
D1	CD1	F4
E1	CE1	F3
F1	CF1	E2
G1	CG1	D2
DP1	DP1	H2
DN0_K1	BIT1	G2
DN0_K2	BIT2	C2
DN0_K3	BIT3	C1
DN0_K4	BIT4	H1
DN1_K1	BIT5	G1
DN1_K2	BIT6	F1
DN1_K3	BIT7	E1
DN1_K4	BIT8	G6

PACKAGE_PIN part of the constraint file：

```
set_property PACKAGE_PIN G2 [get_ports tub_sel]

set_property PACKAGE_PIN B4 [get_ports {tub_control[7]}]
set_property PACKAGE_PIN A4 [get_ports {tub_control[6]}]
set_property PACKAGE_PIN A3 [get_ports {tub_control[5]}]
set_property PACKAGE_PIN B1 [get_ports {tub_control[4]}]
set_property PACKAGE_PIN A1 [get_ports {tub_control[3]}]
set_property PACKAGE_PIN B3 [get_ports {tub_control[2]}]
set_property PACKAGE_PIN B2 [get_ports {tub_control[1]}]
set_property PACKAGE_PIN D5 [get_ports {tub_control[0]}]

set_property PACKAGE_PIN P5 [get_ports {in_b4[3]}]
set_property PACKAGE_PIN P4 [get_ports {in_b4[2]}]
set_property PACKAGE_PIN P3 [get_ports {in_b4[1]}]
set_property PACKAGE_PIN P2 [get_ports {in_b4[0]}]
```

Tips: the defination of input and output ports of the module could be found on the previous page.

lab5_demo1: BCD2Decimal_show(3



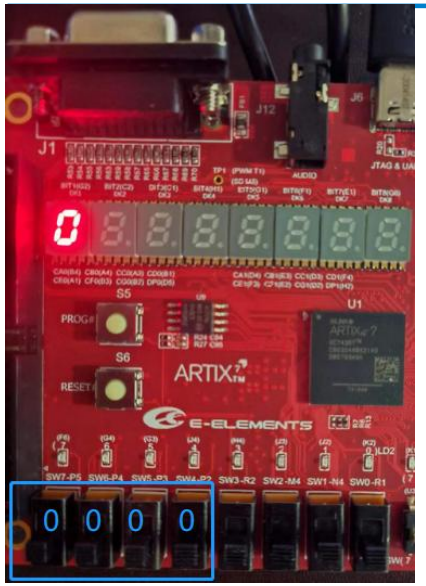

```
module lab5_demo1(
        input [3:0] in_b4,
        output tub_sel,
        output reg [7:0]  tub_control
    );
    assign tub_sel = 1'b1;

    // tub_control : abcb_efg"dot"

endmodule
```

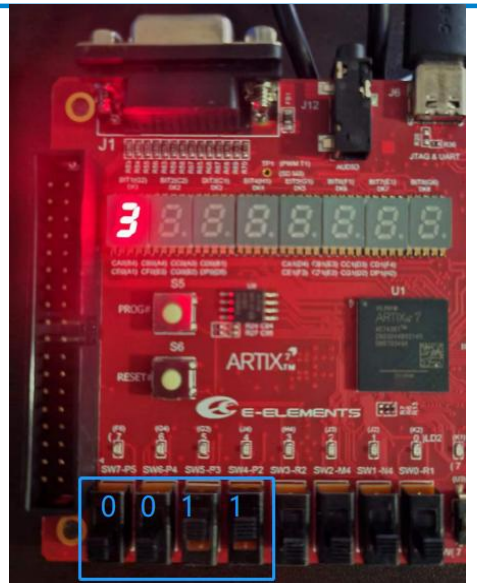
```
//tub_control : abcb_efg"dot"
    always @ * begin
        case(in_b4)
            4'b0000: tub_control = 8'b1111_1100;  //"0" : abcdef_ _
            4'b0001: tub_control = 8'b0110_0000; //"1":  _bc_ _ _ _ _ _
            4'b0010: tub_control = 8'b1101_1010; //"2": ab_de_g_
            4'b0011: tub_control = 8'b1111_0010;  //"3":  abcd_ _ g _
            4'b0100: tub_control = 8'b0110_0110; //"4": _bc _ _fg_
            4'b0101: tub_control = 8'b1011_0110;  //"5": a_cd_fg_
            4'b0110: tub_control = 8'b1011_1110; //"6": a_cdefg_
            4'b0111: tub_control = 8'b1110_0000; //"7": abc_ _ _ _ _
            4'b1000: tub_control = 8'b1111_1110; //"8": abcdefg_
            4'b1001: tub_control = 8'b1110_0110; //"9": abc_ _ fg_
            default:
                tub_control = 8'b1001_1110;   //"E": a_ _ defg_
        endcase
    end
```

名称	原理图标号	FPGA IO PIN
A0	CA0	B4
B0	CB0	A4
C0	CC0	A3
D0	CD0	B1
E0	CE0	A1
F0	CF0	B3
G0	CG0	B2
DP0	DP0	D5
A1	CA1	D4
B1	CB1	E3
C1	CC1	D3
D1	CD1	F4
E1	CE1	F3
F1	CF1	E2
G1	CG1	D2
DP1	DP1	H2
DN0_K1	BIT1	G2
DN0_K2	BIT2	C2
DN0_K3	BIT3	C1
DN0_K4	BIT4	H1
DN1_K1	BIT5	G1
DN1_K2	BIT6	F1
DN1_K3	BIT7	E1
DN1_K4	BIT8	G6

```
set_property PACKAGE_PIN B4 [get_ports {tub_control[7]}]
set_property PACKAGE_PIN A4 [get_ports {tub_control[6]}]
set_property PACKAGE_PIN A3 [get_ports {tub_control[5]}]
set_property PACKAGE_PIN B1 [get_ports {tub_control[4]}]
set_property PACKAGE_PIN A1 [get_ports {tub_control[3]}]
set_property PACKAGE_PIN B3 [get_ports {tub_control[2]}]
set_property PACKAGE_PIN B2 [get_ports {tub_control[1]}]
set_property PACKAGE_PIN D5 [get_ports {tub_control[0]}
```

lab5_demo1: BCD2Decimal_show(4)





Q1: Change the PACKAGE_PIN on tub_control part of the constraints file from the left bottom to the right bottom on this page, re-generate bitstream and re-program-device. while test the circuit on the EGO1, the in_b4 is 4'b0011, what's the tube display?

Q2: To make the tube on the righ hand of the EGO1 to show the output, how to make changes? on design file or on contraint file?

```
set_property PACKAGE_PIN B4 [get_ports {tub_control[7]}]
set_property PACKAGE_PIN A4 [get_ports {tub_control[6]}]
set_property PACKAGE_PIN A3 [get_ports {tub_control[5]}]
set_property PACKAGE_PIN B1 [get_ports {tub_control[4]}]
set_property PACKAGE_PIN A1 [get_ports {tub_control[3]}]
set_property PACKAGE_PIN B3 [get_ports {tub_control[2]}]
set_property PACKAGE_PIN B2 [get_ports {tub_control[1]}]
set_property PACKAGE_PIN D5 [get_ports {tub_control[0]}
```

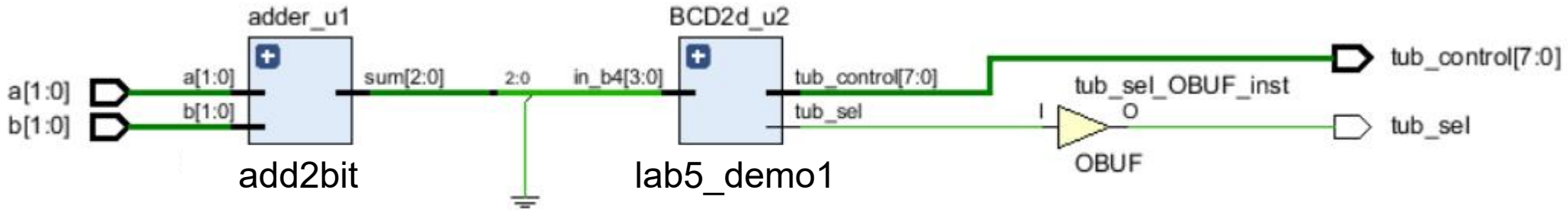
```
set_property PACKAGE_PIN B4 [get_ports {tub_control[0]}]
set_property PACKAGE_PIN A4 [get_ports {tub_control[1]}]
set_property PACKAGE_PIN A3 [get_ports {tub_control[2]}]
set_property PACKAGE_PIN B1 [get_ports {tub_control[3]}]
set_property PACKAGE_PIN A1 [get_ports {tub_control[4]}]
set_property PACKAGE_PIN B3 [get_ports {tub_control[5]}]
set_property PACKAGE_PIN B2 [get_ports {tub_control[6]}]
set_property PACKAGE_PIN D5 [get_ports {tub_control[7]}
```

lab5_demo2: adder_show_sum_in_decimal(1)

Goal: Design a circuit which get the addition of two two-bit unsigned numbers, and display the binary representation of the result on the LED and the decimal representation of the result on the 7-segment tube.

Analysis:

1) Get the addition of two two-bit unsigned numbers. The inputs are a and b which are both 2 bits, the output is “sum” which is 3 bits.
   - “add2bit” could be reused as a sub module in this design.
2) Display the binary representation of sum on the LED and the decimal representation of sum on the 7-segment tube.
   - “lab5_demo1” could be reused as a sub module in this design.
   - The bitwidth of the output “sum” of “lab3_practic_add2bit” is 3, while the bitwidth of the input “in_b4” is 4, how to convert “sum” to “in_b4” and keep their values consistent ?
3) How to make the constraints file of lab5_demo2? What’s the relationship of these 3 module’s constraints file?

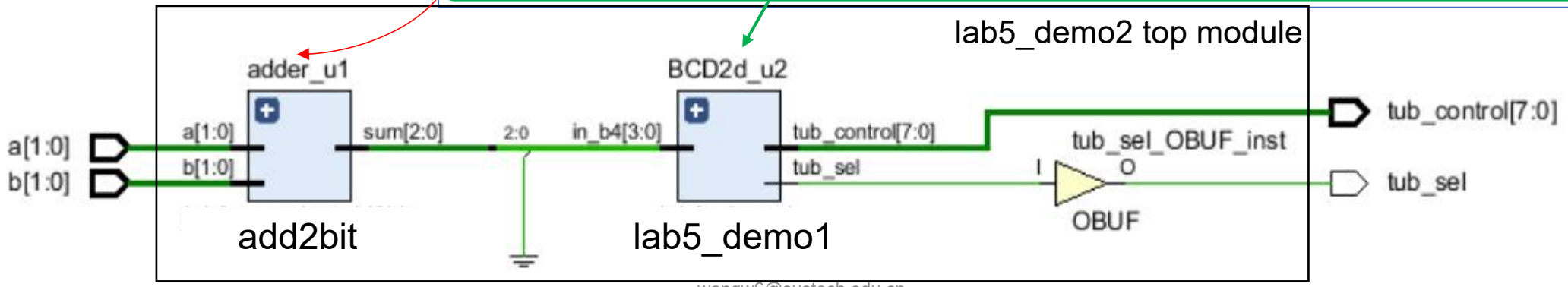


lab5_demo2: adder_show_sum_in_decimal(2)

- **Hierarchical design** (structural design by using sub-modules)
  - The design approach breaks down a complex digital circuit into smaller modules. These modules are designed independently and then interconnected to form the complete circuit.

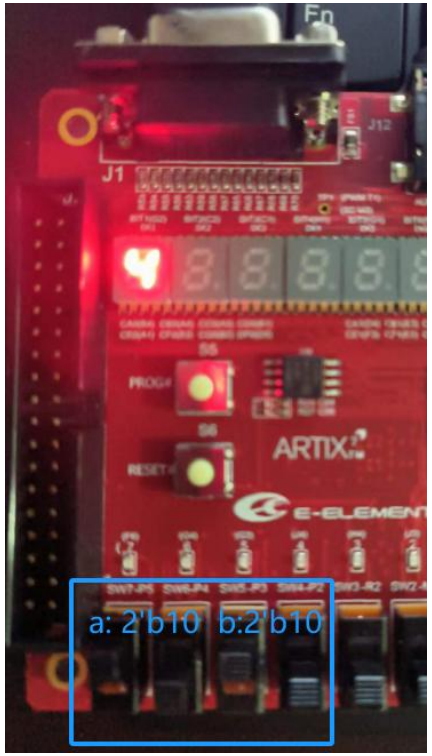
```
module lab5_demo2(
        input [1:0] a, b,
        output tub_sel,
        output [7:0] tub_control
    );
    /*hierarchical design details*/
endmodule
```

```
wire [2:0] sum;
wire [3:0] in_b4;

assign in_b4 = {1'b0, sum};
/*add2bit (input [1:0] a,b, output [2:0] sum );*/
add2bit  adder_u1(.a(a),  .b(b),  .sum( /* complete code here*/) );

lab5_demo1 BCD2d_u2(.in_b4( /* complete code here*/),  .tub_sel(tub_sel),
                                                    . tub_control(tub_control) );
```



lab5_demo2: adder_show_sum_in_decimal(3)

Design a circuit which get the addition of two two-bit unsigned numbers, and display the binary representation of the result on the LED and the decimal representation of the result on the 7-segment tube.



tc1: 2 + 2 = 4

tc2: 2+3 = 5

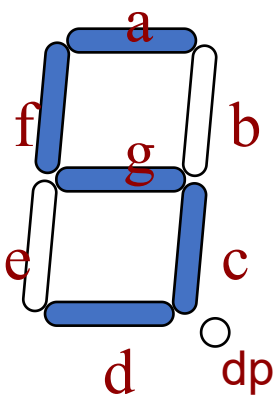
tc3: 3 + 3 = 6

Practice1

- Today we continue the display part of the Rock-Paper-Scissors
  - The input variables for the comparator would be two binary numbers representing the choices of the two players.
  - Now we want to display the hand gesture played by the two player, whether it is scissors, rock, or paper.
    - if it's scissors, the 7-segment display shows an 'S'
    - if it's rock, the 7-segment display shows an 'r'
    - if it's paper, the 7-segment display shows an 'P'
    - otherwise, the 7-segment display shows an 'E'

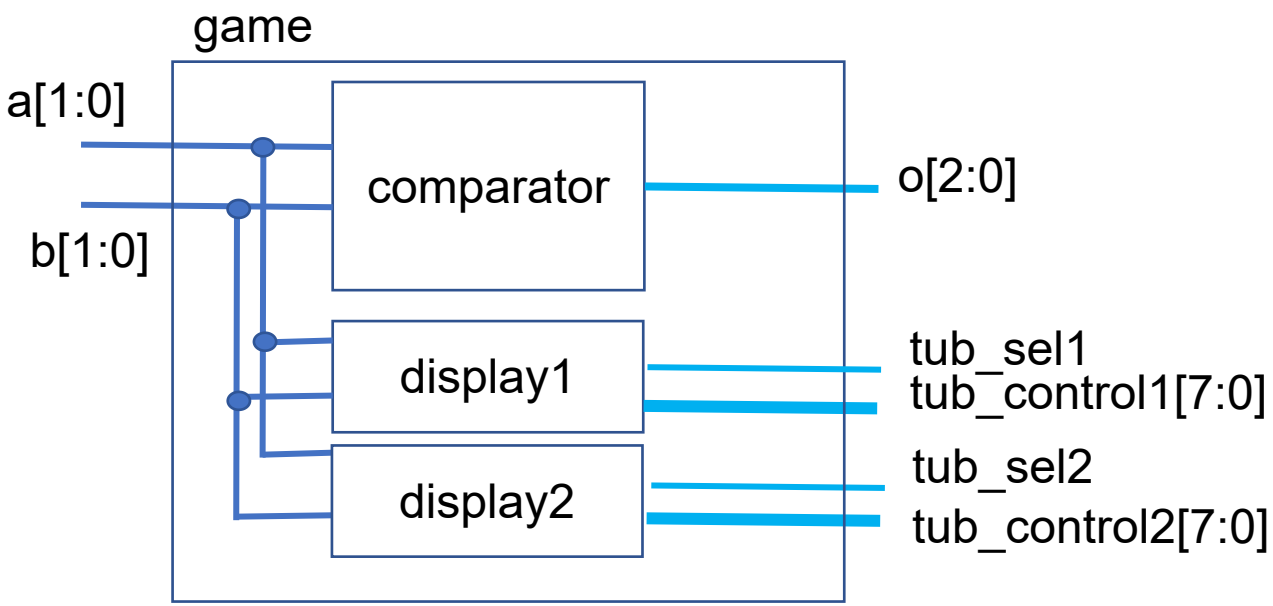
- Implement your **display_decoder** module
  1. Design using behavioral style with 'case'
  2. Simulate your design with testbench.

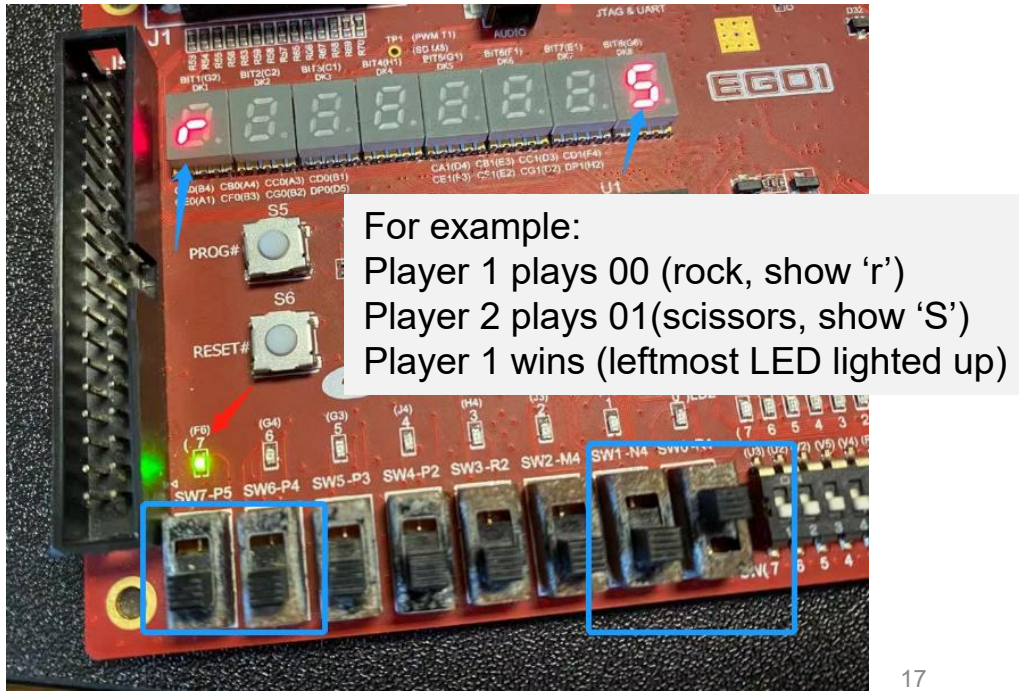
```
module display_decoder(
  input [TBD:0] in,
  output tub_sel,
  output reg [7:0] tub_control
  // Complete your design
endmodule
```



Practice2:
Rock-Paper-Scissors game (v2.0)

- Design a **game** using **Hierarchical design.**
  1. Create "**game**" module as top level module in a new file
  2. Instantiate 3 sub-modules: 1 using **comparator** and 2 using **display_decoder**
  3. Generate the bitstream and test your game on board.
  4. ATTENTION: binding the output using wire type!!!
  5. To be continued, so please keep your source files.





For example:
Player 1 plays 00 (rock, show 'r')
Player 2 plays 01(scissors, show 'S')
Player 1 wins (leftmost LED lighted up)

Practice3(optional)

- 3-1: Design a circuit which get the addition of two three-bit unsigned numbers, and display the decimal representation of the result on the 7-segment tubes.
  - Tips: It's suggested to use two 7-segment tubes:
    - for example:
      the sum of unsigned number 3'b100 (4) and 3'b110 (6) is ten(10)
      using the tube on the left to show "1", using the tube on the right to show "0".

- 3-2: Design a circuit which get the addition of two three-bit signed numbers, and display the decimal representation of the result on the 7-segment tubes.
  - Tips: It's suggested to use two 7-segment tubes:
    - for example:
      the sum of signed number 3'b101(-3) and 3'b110(-2) is negative five(-5)
      using the tube on the left to show the negative flag "-", using the tube on the right to show "5".