數位系統技術

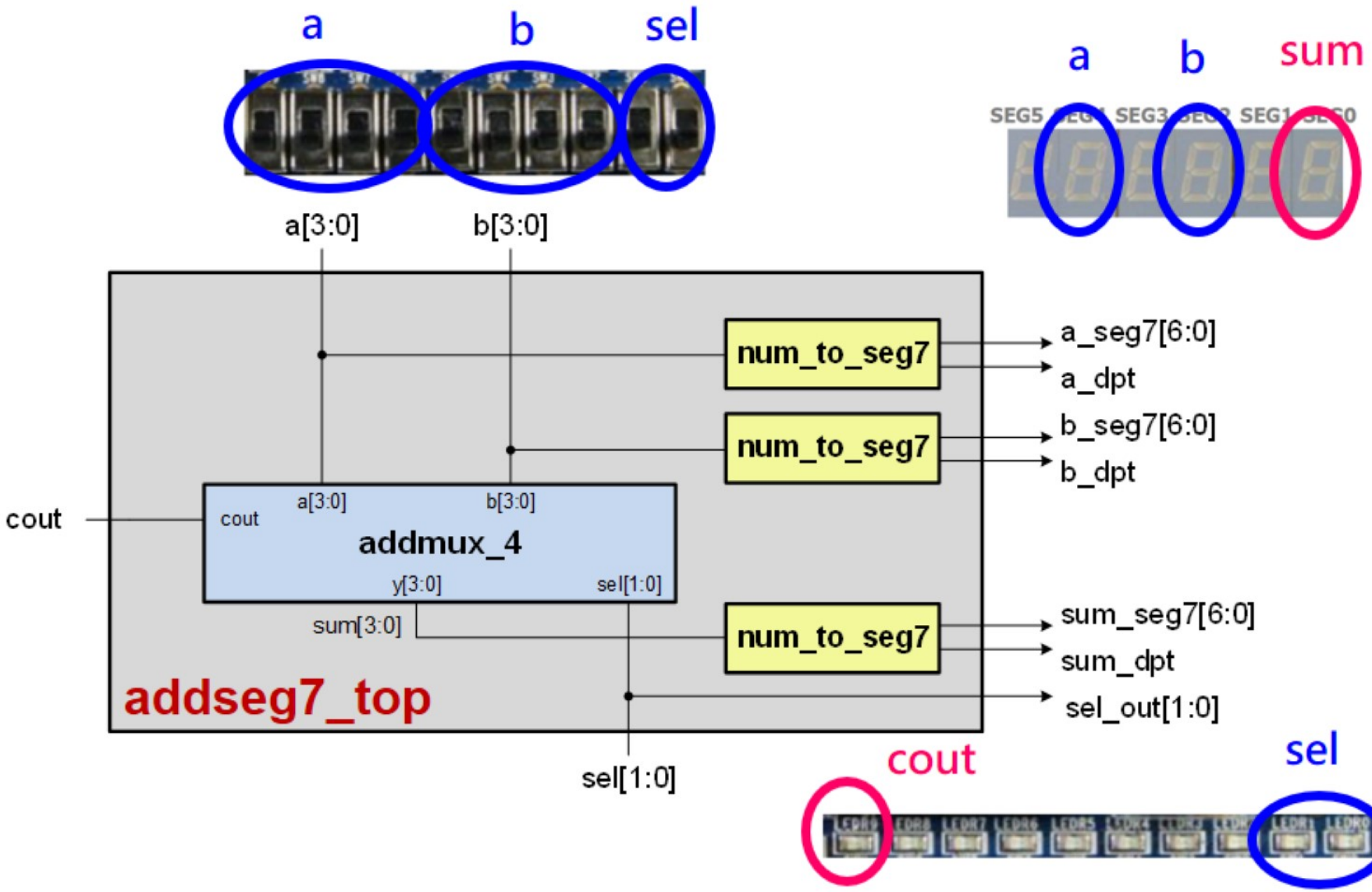
7-Segment Display

Ren-Der Chen (陳仁德)
Department of Computer Science and
Information Engineering
National Changhua University of Education
E-mail: rdchen@cc.ncue.edu.tw
Spring, 2025

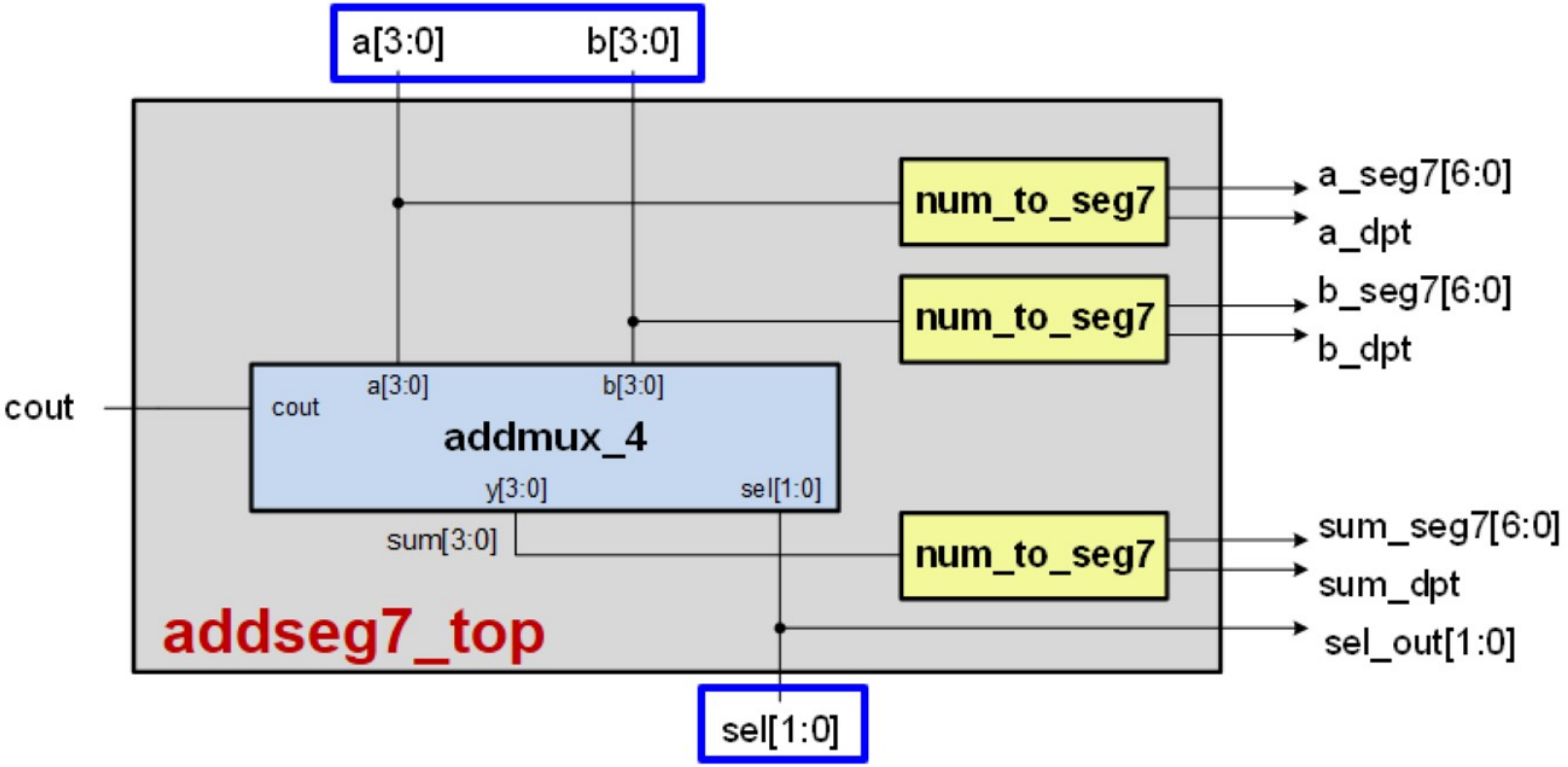
實驗內容

- 使用七段顯示器 (7-segment display) 顯示數字。
- 設計一個電路，將4-bit adder/subtractor的運算結果呈現於七段顯示器上。
- 將所設計之電路燒錄至FPGA晶片中，並利用實驗板週邊進行電路功能驗證。

addseg7_top 電路方塊圖 (1/3)



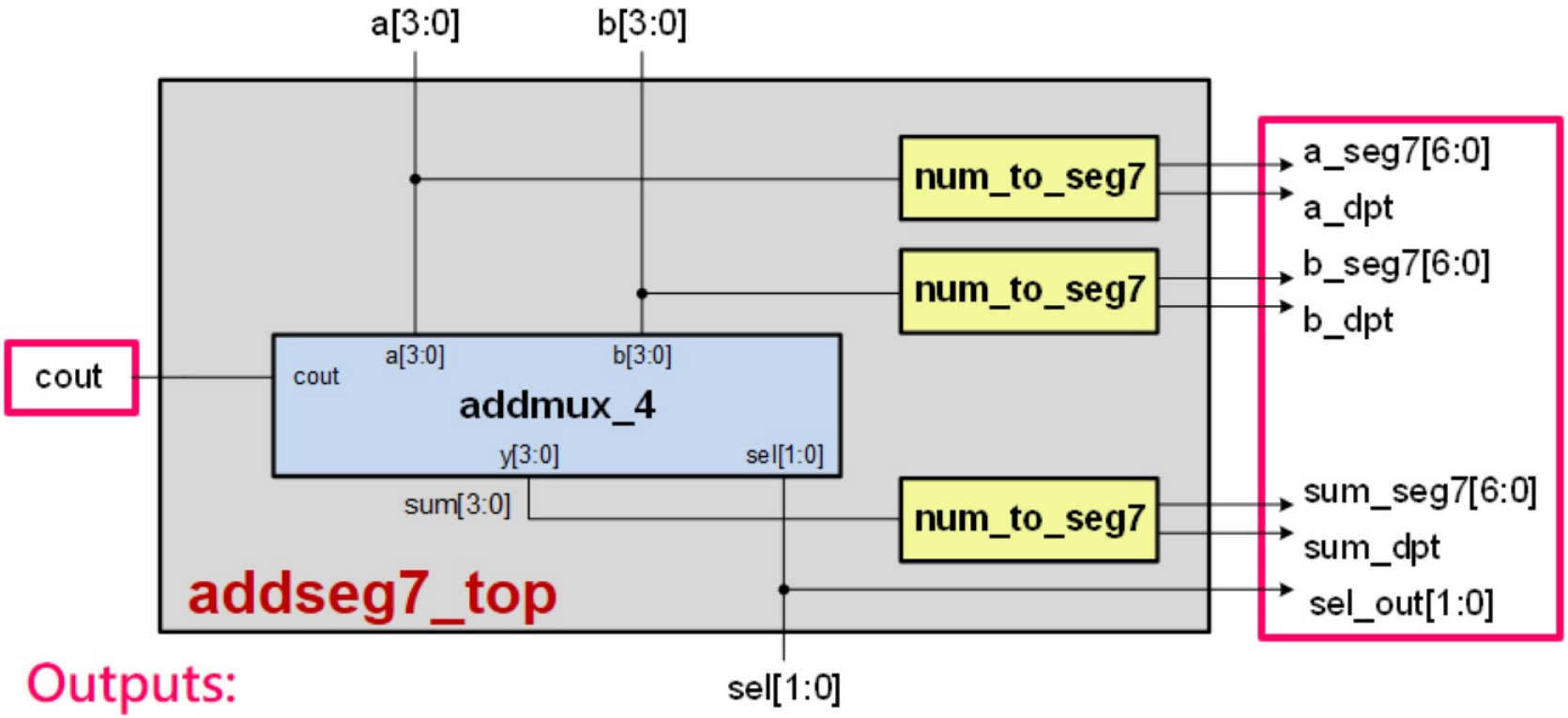
addseg7_top 電路方塊圖 (2/3)



Inputs:

1. a[3:0], b[3:0]: 要運算之數字
2. sel[1:0]: 選擇輸出為a, b, a+b, or a-b (四選一)

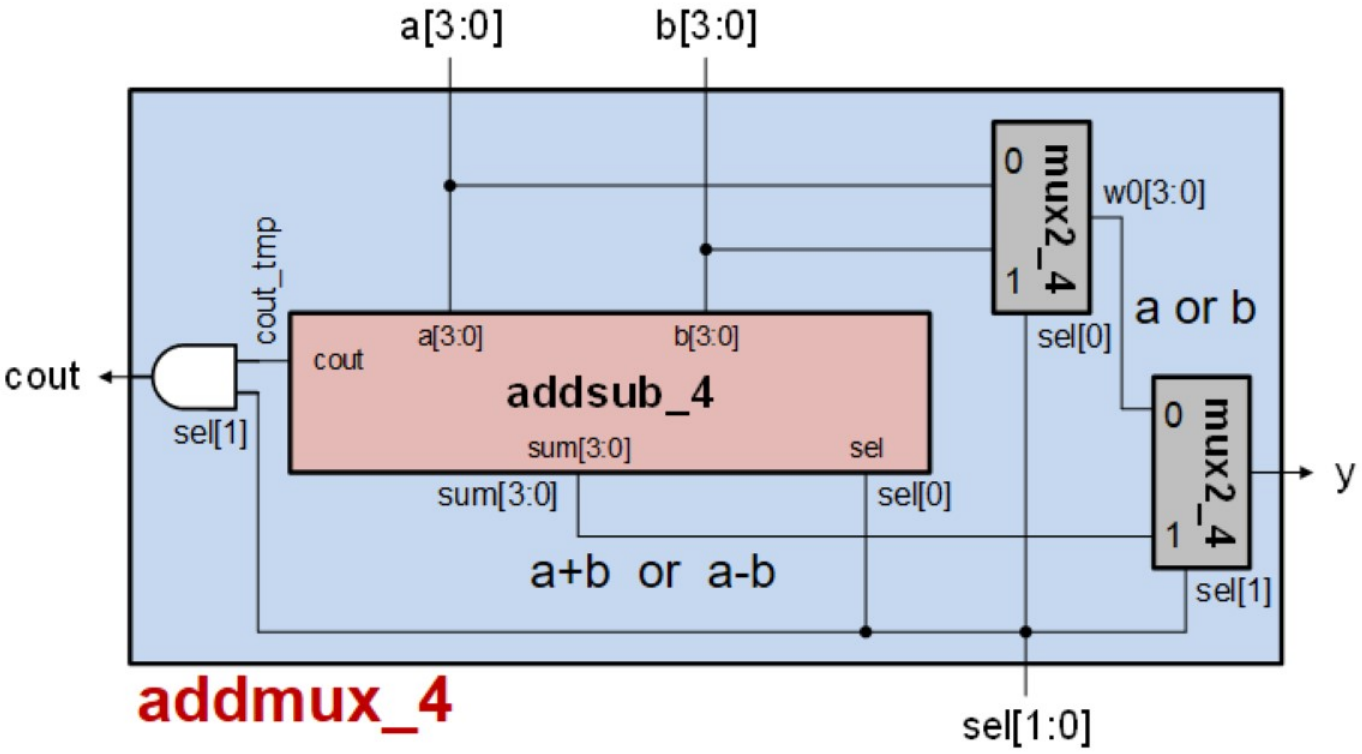
addseg7_top 電路方塊圖 (3/3)



Outputs:

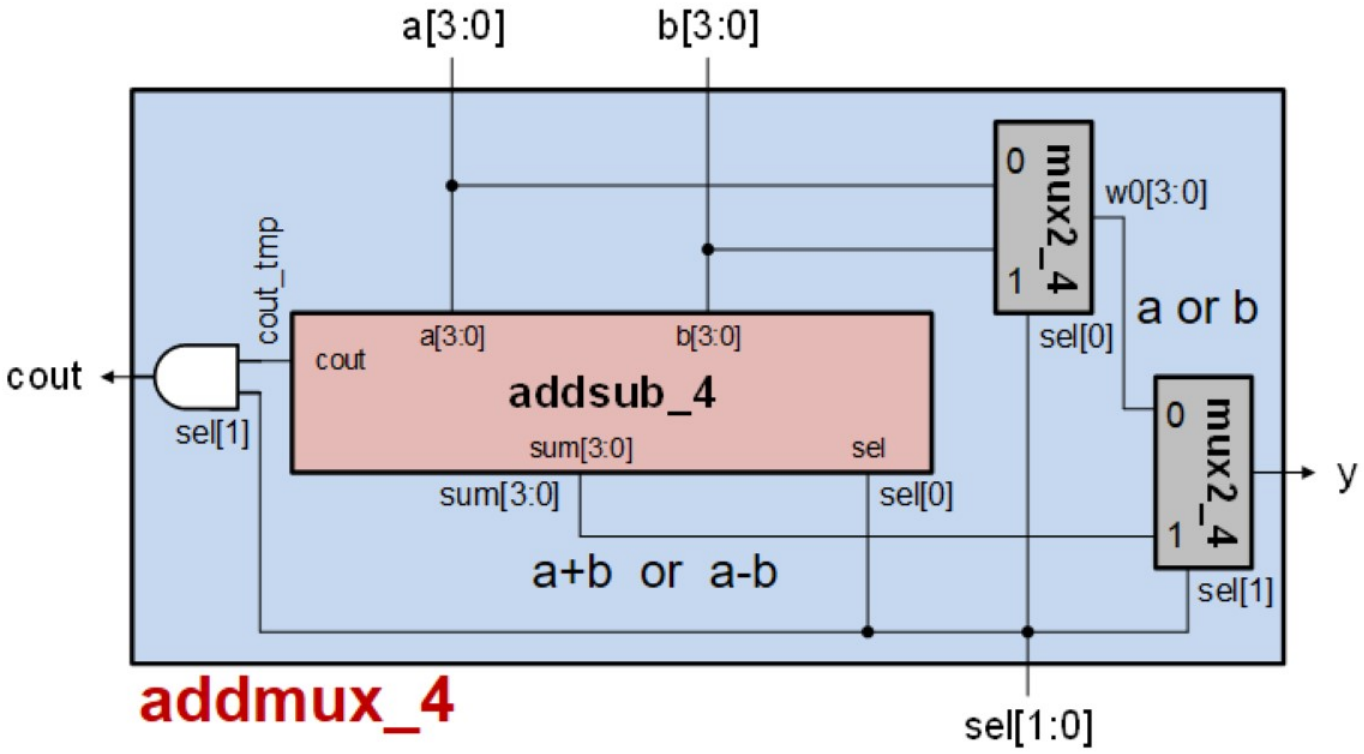
1. **cout:** 接至LED9, 燈亮表示發生overflow/underflow
2. **a_seg7[6:0], b_seg7[6:0], sum_seg7[6:0]:** 分別送至SEG4, SEG2,SEG0之信號a, b, sum (a, b, a+b, or a-b)之編碼
3. **a_dpt, b_dpt, sum_dpt:** 分別送至SEG4, SEG2,SEG0小數點之符號位元, 亮表示負數, 暗表示正數

addmux_4 電路方塊圖 (1/2)



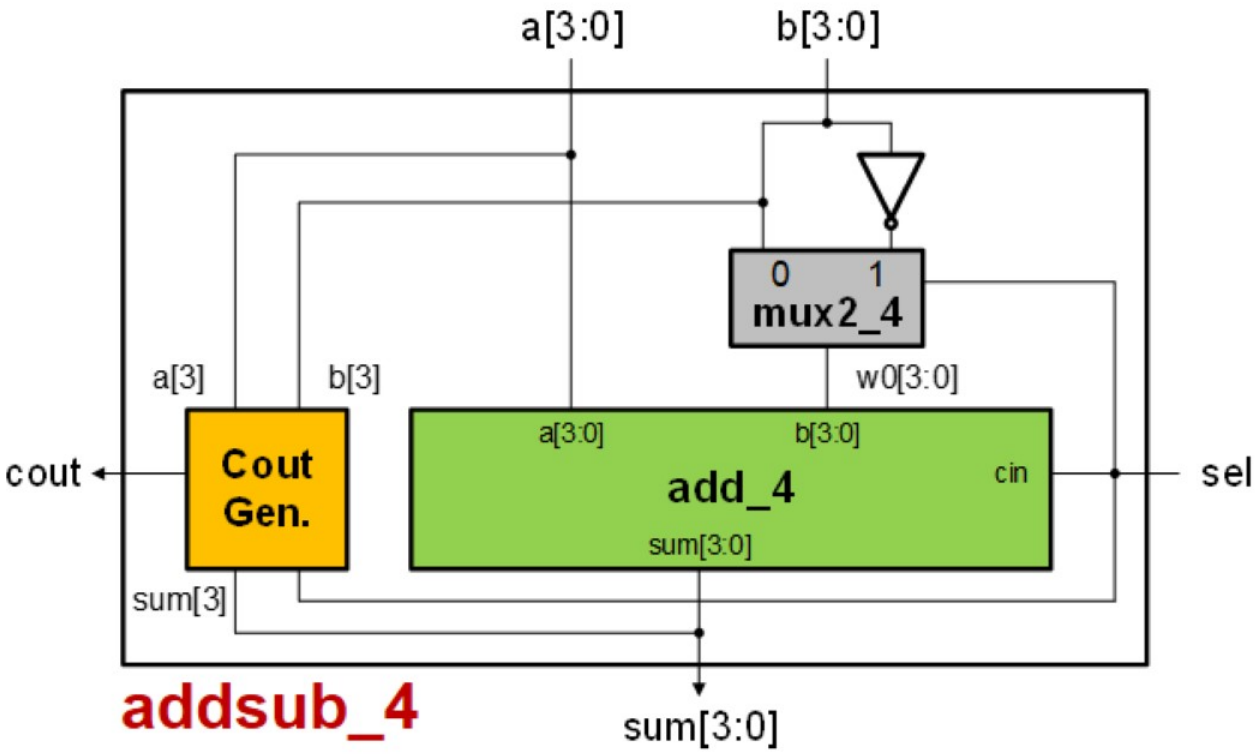
1. sel[1:0] = 00, y = a
2. sel[1:0] = 01, y = b
3. sel[1:0] = 10, y = a+b, cout為overflow/underflow偵測
4. sel[1:0] = 11, y = a-b, cout為overflow/underflow偵測

addmux_4 電路方塊圖 (2/2)

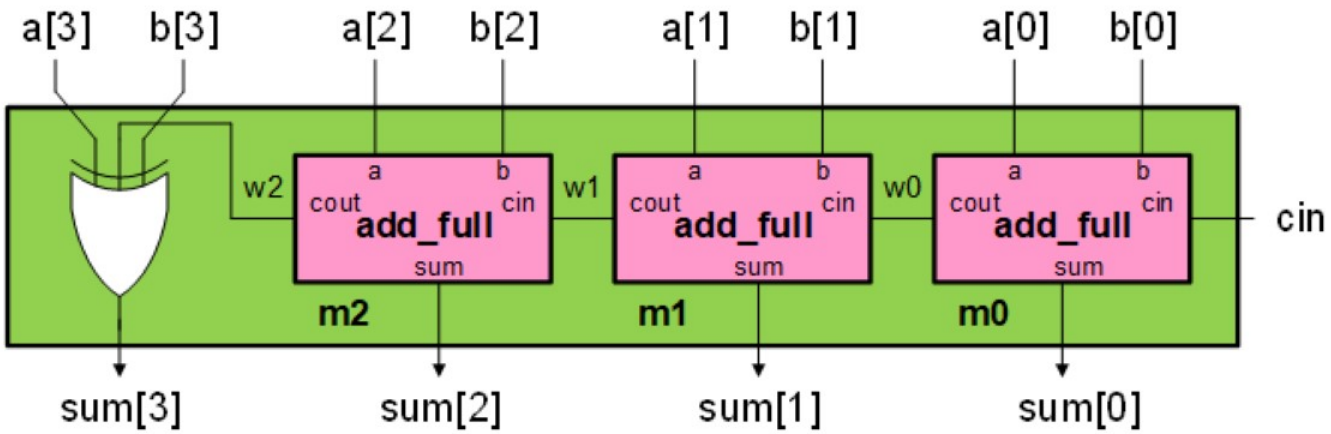


1. if sel[1]=0, y = a or b, cout = 0 (不會有overflow/underflow問題)
2. if sel[1]=1, y = a+b or a-b, cout = cout_tmp (將實際運算後的cout傳出)
3. if sel[0]=0, 將信號a傳出 or 進行加法運算
4. if sel[0]=1, 將信號b傳出 or 進行減法運算

addsub_4 電路方塊圖



add_4 電路方塊圖



add_4

Design 0: addsub_4

- Create New Project - addseg7_top
- 利用前次實驗所設計之4-bit adder/subtractor，將所有設計檔案加入至project中。
  - add_half.v
  - add_full.v
  - add_4.v
  - addsub_4.v

4-Bit Signed Number

- 4-bit signed number可表示之值的範圍為**-8 ~ 7**, 第一個位元為符號位元, 0為正, 1為負。
- Overflow: 兩個4-bit number之運算結果大於7。
- Underflow: 兩個4-bit number之運算結果小於-8。

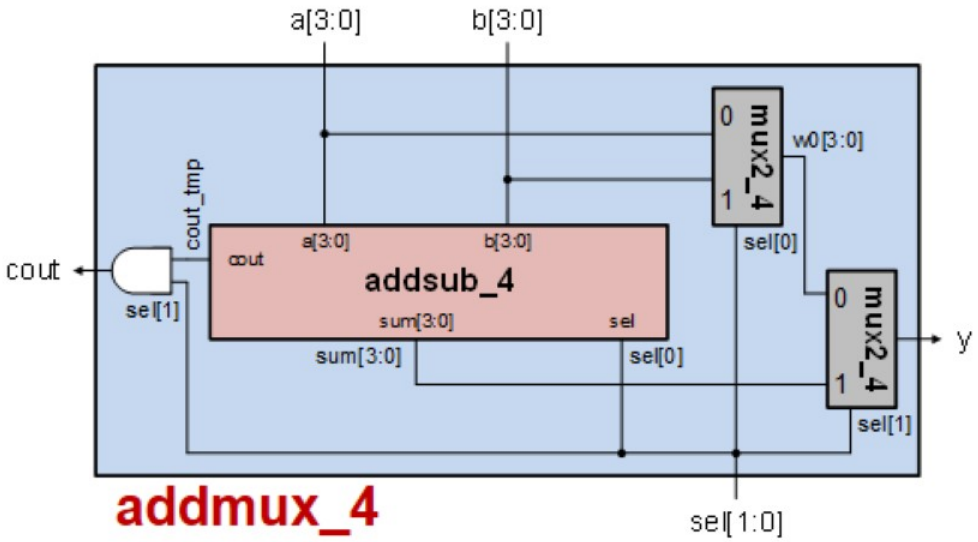
0000 -> 0
0001 -> 1, 1111 -> -1
0010 -> 2, 1110 -> -2
0011 -> 3, 1101 -> -3
0100 -> 4, 1100 -> -4
0101 -> 5, 1011 -> -5
0110 -> 6, 1010 -> -6
0111 -> **7**, 1001 -> -7
1000 -> **-8**

Design 1: addmux_4 (1/2)

- 利用addsub_4，設計addmux_4，使其輸出可選擇a, b, a+b, or a-b。
- 將addmux_4.v加入至project中。
- 完成functional simulation，驗證行為之正確。

Design 2: addmux_4 (2/2)

```
module addmux_4(a, b, sel, cout, y);
    input   [3:0] a,b;
    input   [1:0] sel;
    output  cout;
    output  [3:0] y;
```

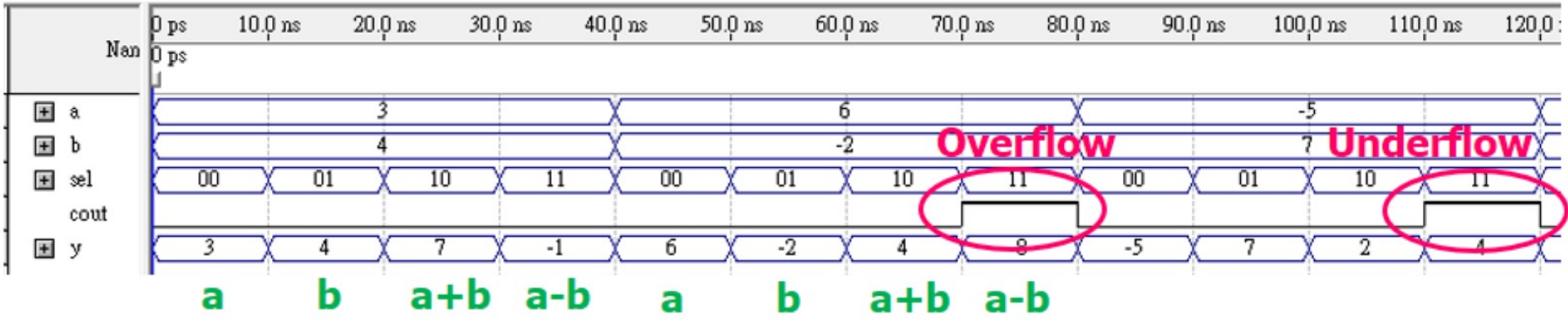



```
    addsub_4 a0
```

```
    assign w0   =
    assign y    =
    assign cout =
endmodule
```

// ex. assign y = (x==1) ? a : b;

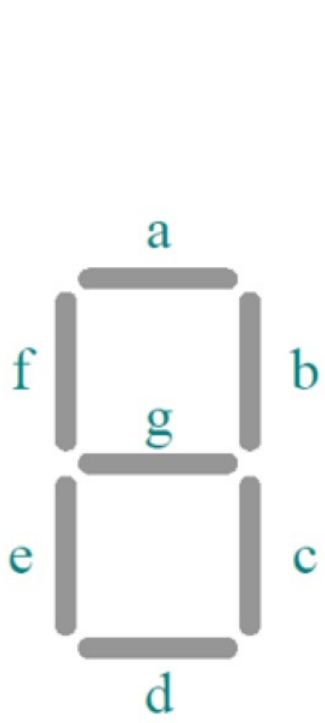
Functional Simulation for “addmux_4”

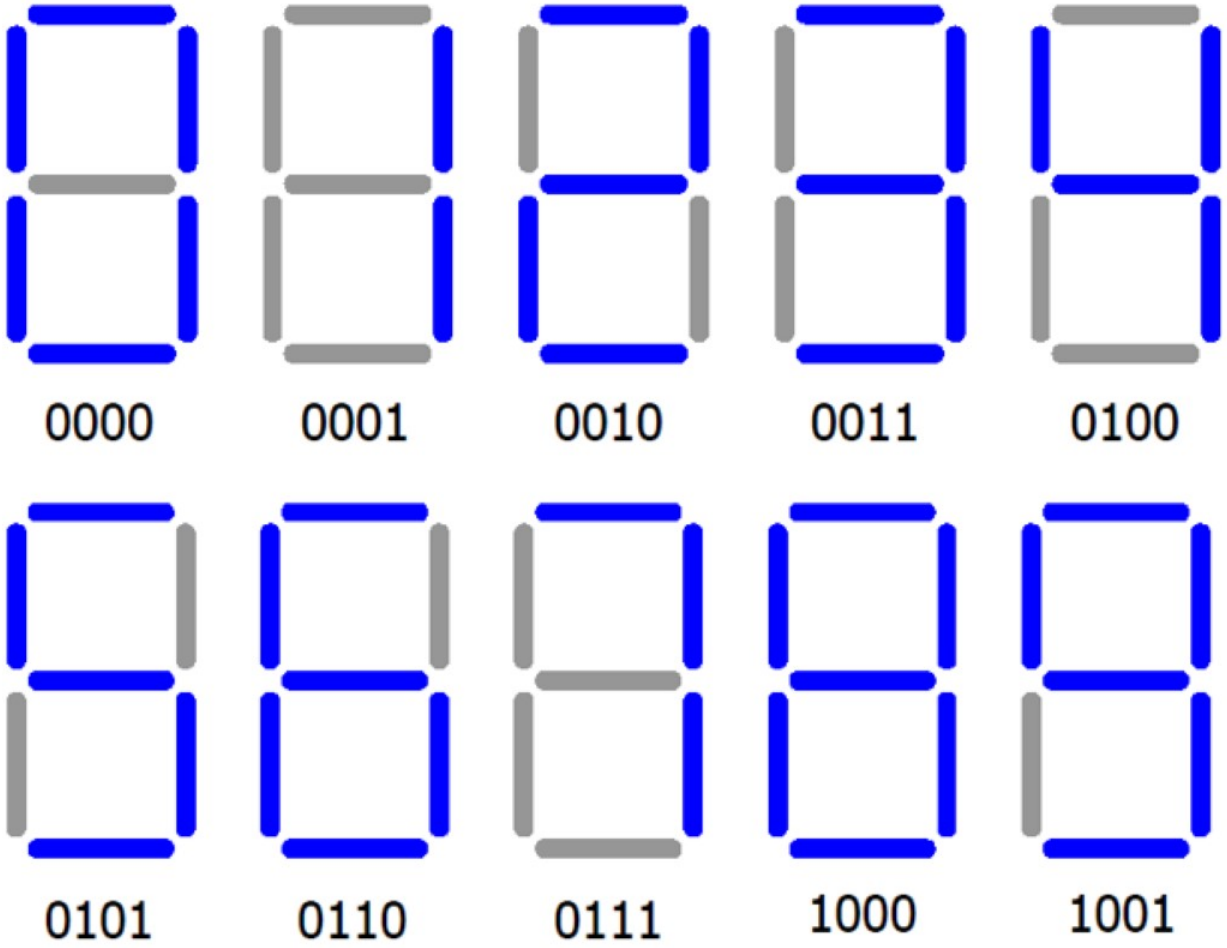


記錄實驗結果

1. sel[1:0] = 00, y = a
2. sel[1:0] = 01, y = b
3. sel[1:0] = 10, y = a+b, cout為overflow/underflow偵測
4. sel[1:0] = 11, y = a-b, cout為overflow/underflow偵測

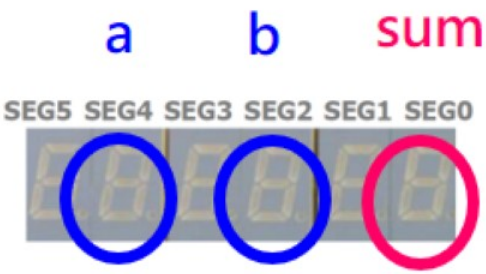
7-Segment Display (1/2)

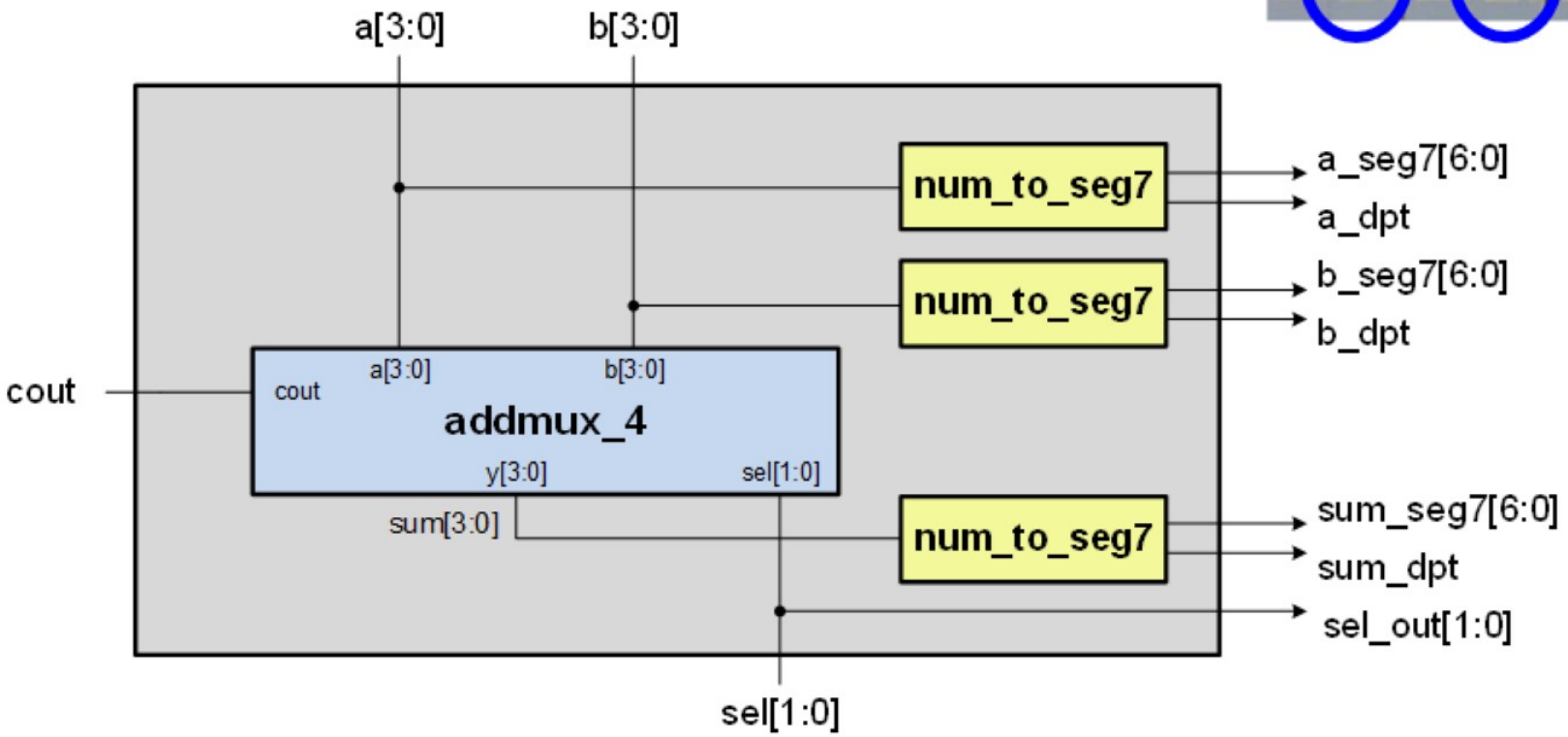




7-Segment Display (2/2)

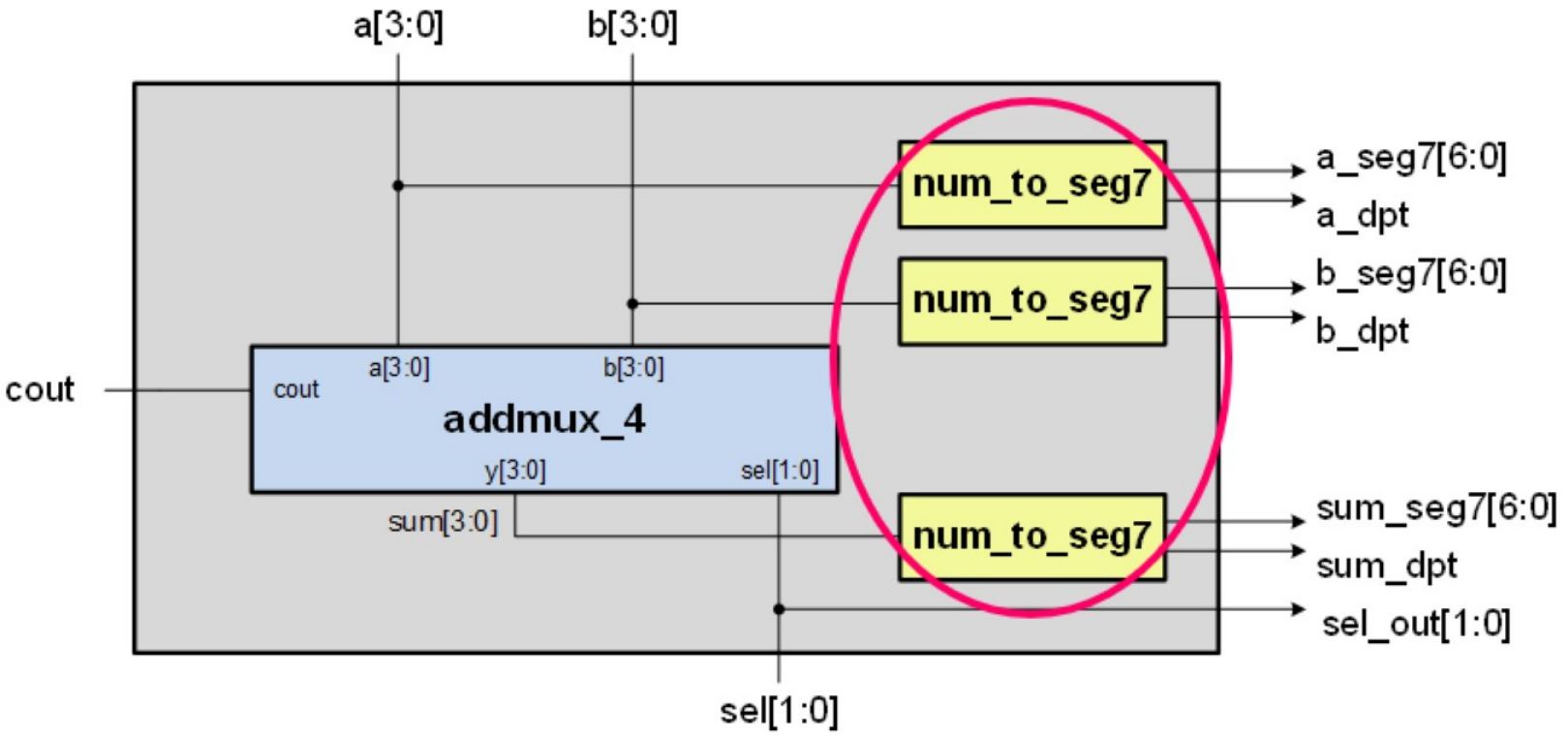
- a_seg7[6:0]之值為SEG4之a, b, …, g各信號之值
- a_dpt之值為SEG4小數點之值
- 0: 亮, 1: 暗





Design 2: num_to_seg7 (1/3)

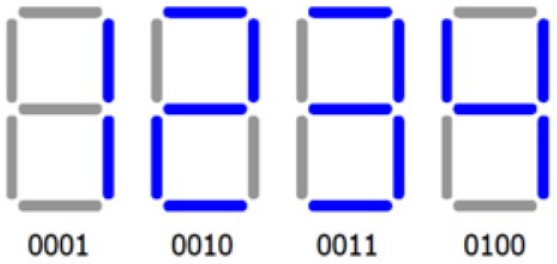
- 設計num_to_seg7.v並加入至project中，將a, b, sum之值轉成對應seg7之信號。
- 小數點亮表示對應數字為負數，否則為正數。

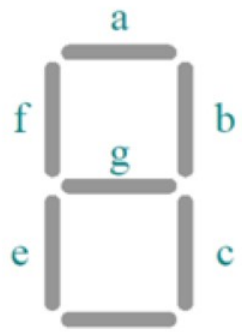


Design 2: num_to_seg7 (2/3)

```
module num_to_seg7(num, seg7, dpt);
    input       [3:0] num;
    output      [6:0] seg7;
    output      dpt;
    reg         [6:0] seg7;
    reg         dpt;

    always @(num)
        case(num)                          //  gfedcba, dpt
            4'b0000:{seg7, dpt} = {7'b1000000, 1'b1}; // 0
            4'b0001:{seg7, dpt} =                     // 1
            4'b0010:{seg7, dpt} =                     // 2
            4'b0011:{seg7, dpt} =                     // 3
            4'b0100:{seg7, dpt} =                     // 4
            4'b0101:{seg7, dpt} = {7'b0010010, 1'b1}; // 5
            4'b0110:{seg7, dpt} = {7'b0000010, 1'b1}; // 6
            4'b0111:{seg7, dpt} = {7'b1111000, 1'b1}; // 7
```





正數, dpt=1, led: off

Design 2: num_to_seg7 (3/3)

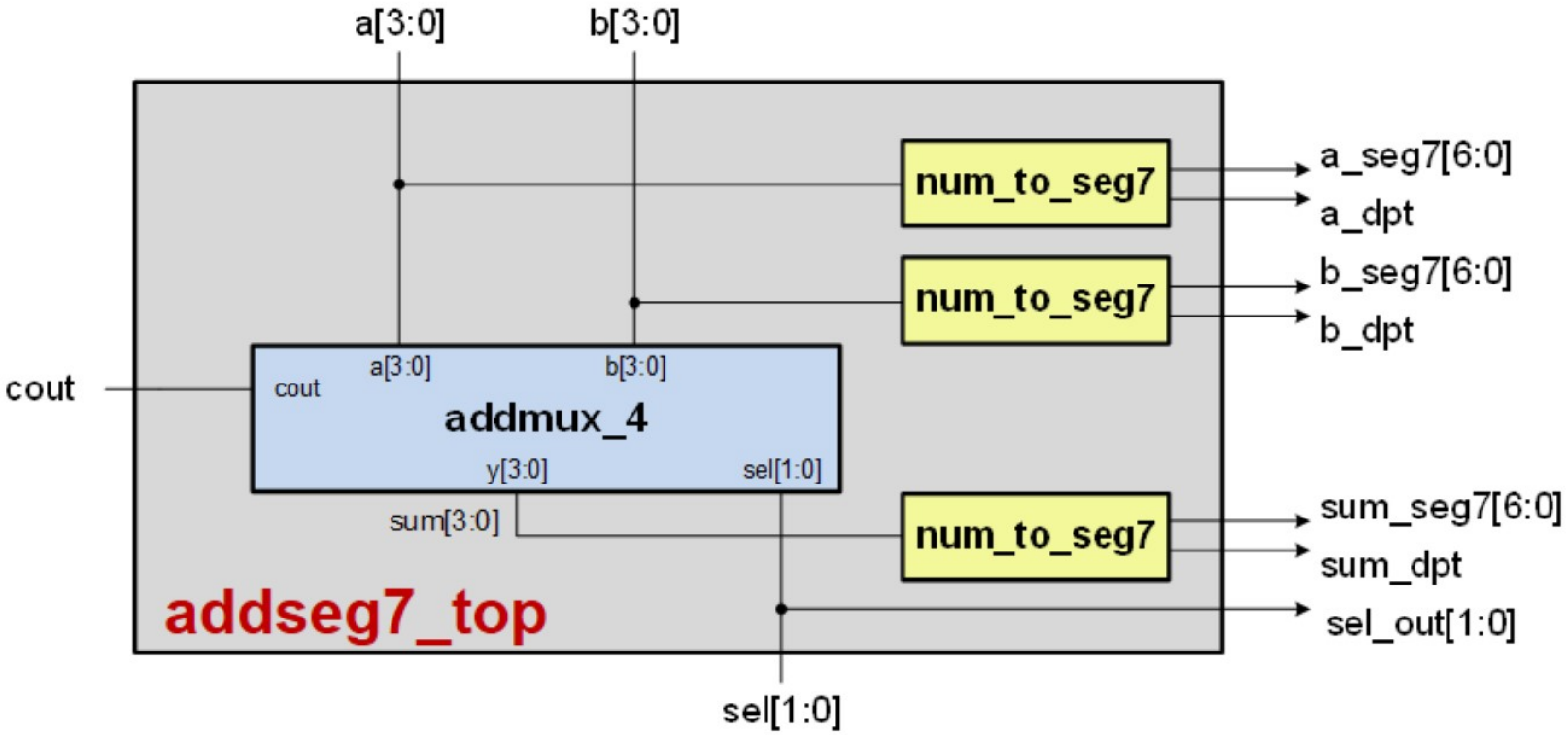
負數, dpt=0, led: on

```
            4'b1000:{seg7, dpt} = {7'b0000000, 1'b0}; // -8
            4'b1001:{seg7, dpt} = {7'b1111000, 1'b0}; // -7
            4'b1010:{seg7, dpt} = {7'b0000010, 1'b0}; // -6
            4'b1011:{seg7, dpt} = {7'b0010010, 1'b0}; // -5
            4'b1100:{seg7, dpt} =                     // -4
            4'b1101:{seg7, dpt} =                     // -3
            4'b1110:{seg7, dpt} =                     // -2
            4'b1111:{seg7, dpt} =                     // -1

            default:{seg7, dpt} = {7'b1111111, 1'b1};
        endcase
endmodule
```

Design 3: addseg7_top (1/3)

- 設計addseg7_top.v並加入至project中，整合所有電路模組。

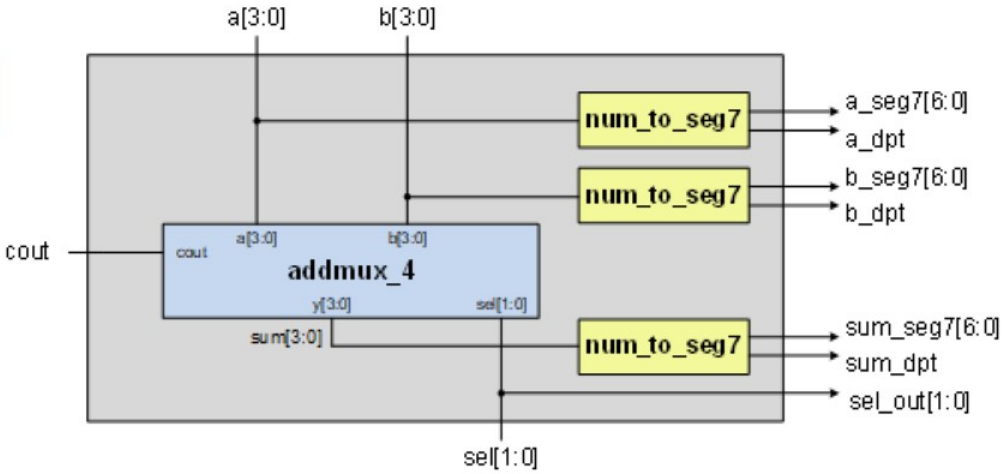


Design 3: addseg7_top (2/3)

```
module addseg7_top(a, b, sel,
a_seg7, a_dpt, b_seg7, b_dpt, sel_out,
cout, sum_seg7, sum_dpt);

    input
    input
    output
    output
    output
    output

    wire
```

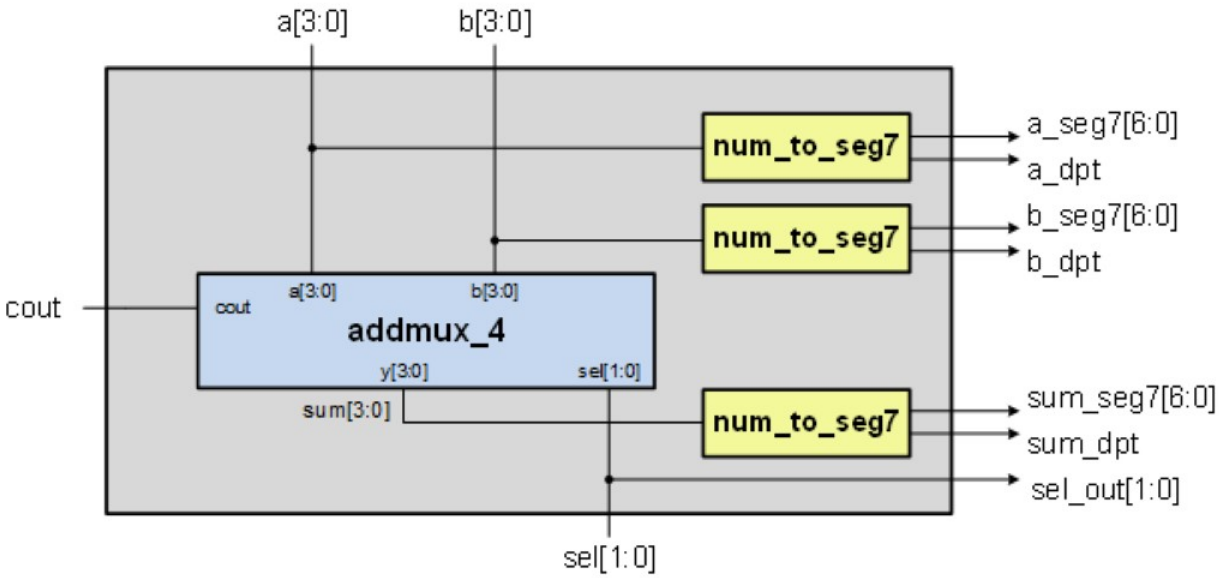


Design 3: addseg7_top (3/3)

```
num_to_seg7   m0
num_to_seg7   m1
assign sel_out

addmux_4      m2
num_to_seg7   m3

endmodule
```



Pin Assignments

a b sel

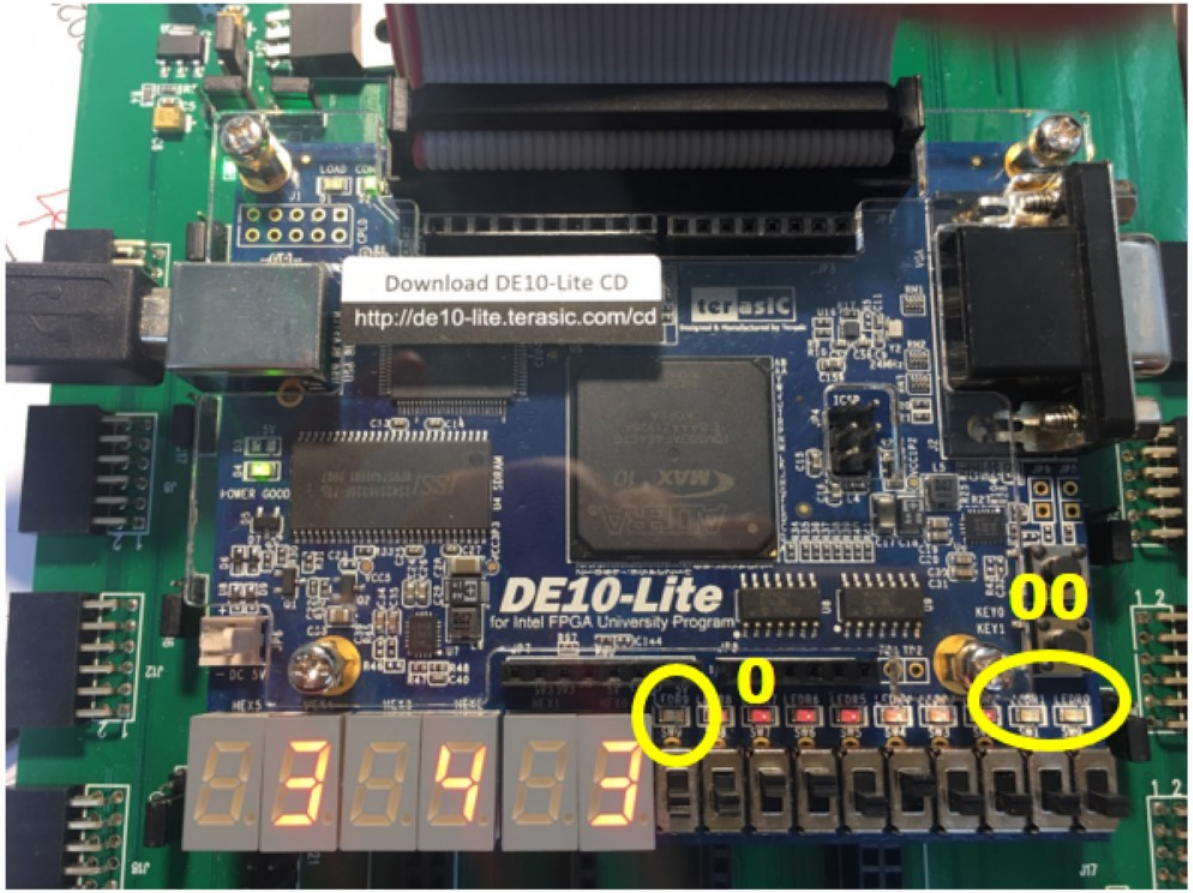
a_seg7 a_dpt b_seg7 b_dpt sum_seg7 sum_dpt

SEG5 SEG4 SEG3 SEG2 SEG1 SEG0

cout sel_out

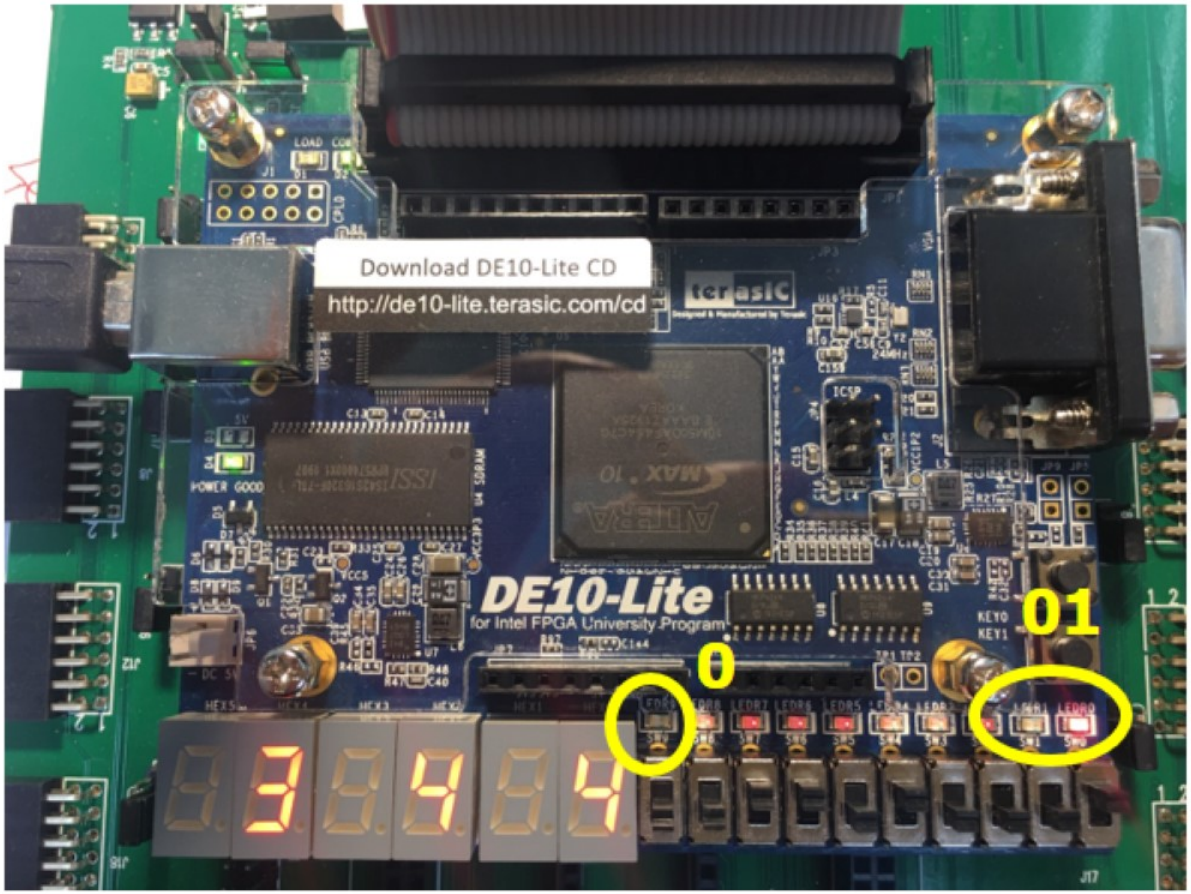
Verification 1

- a=3, b=4, sel=00
- sum = a = 3



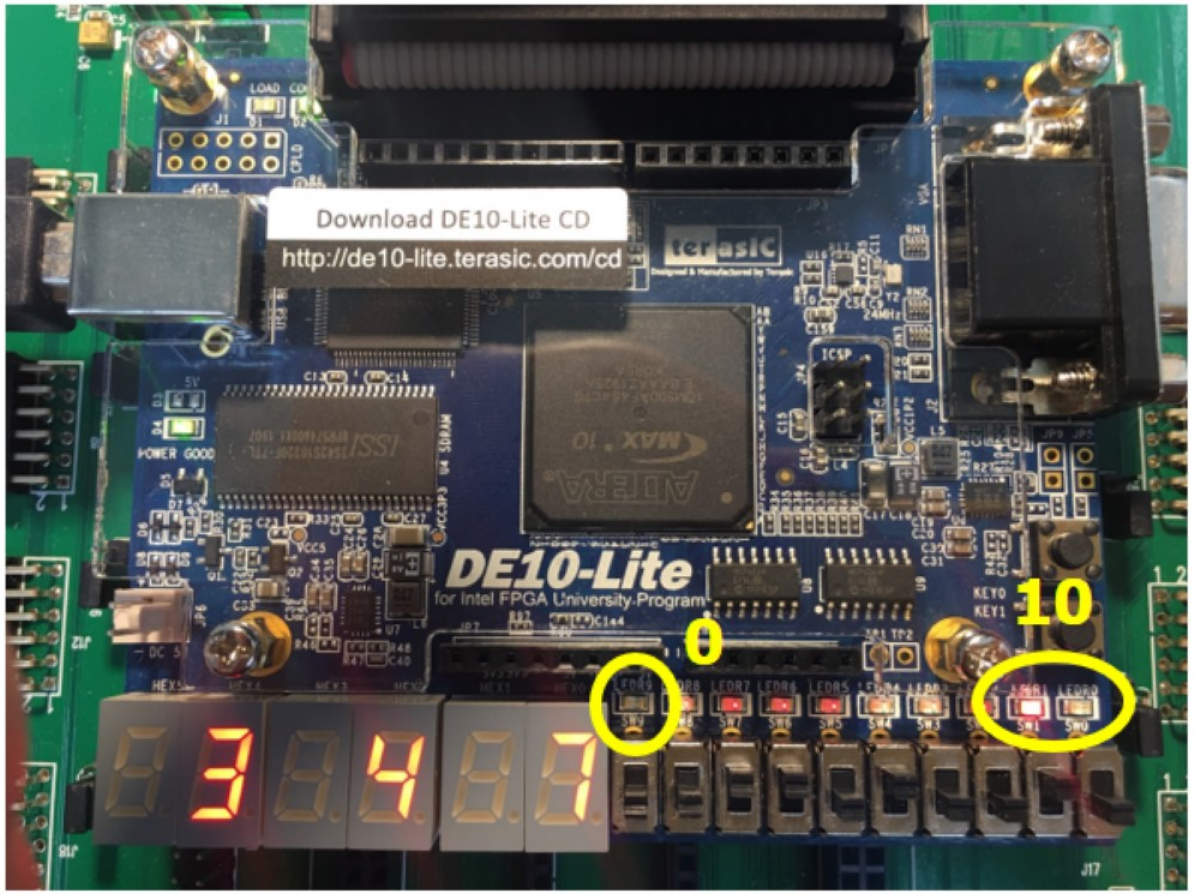
Verification 2

- a=3, b=4, sel=01
- sum = b = 4



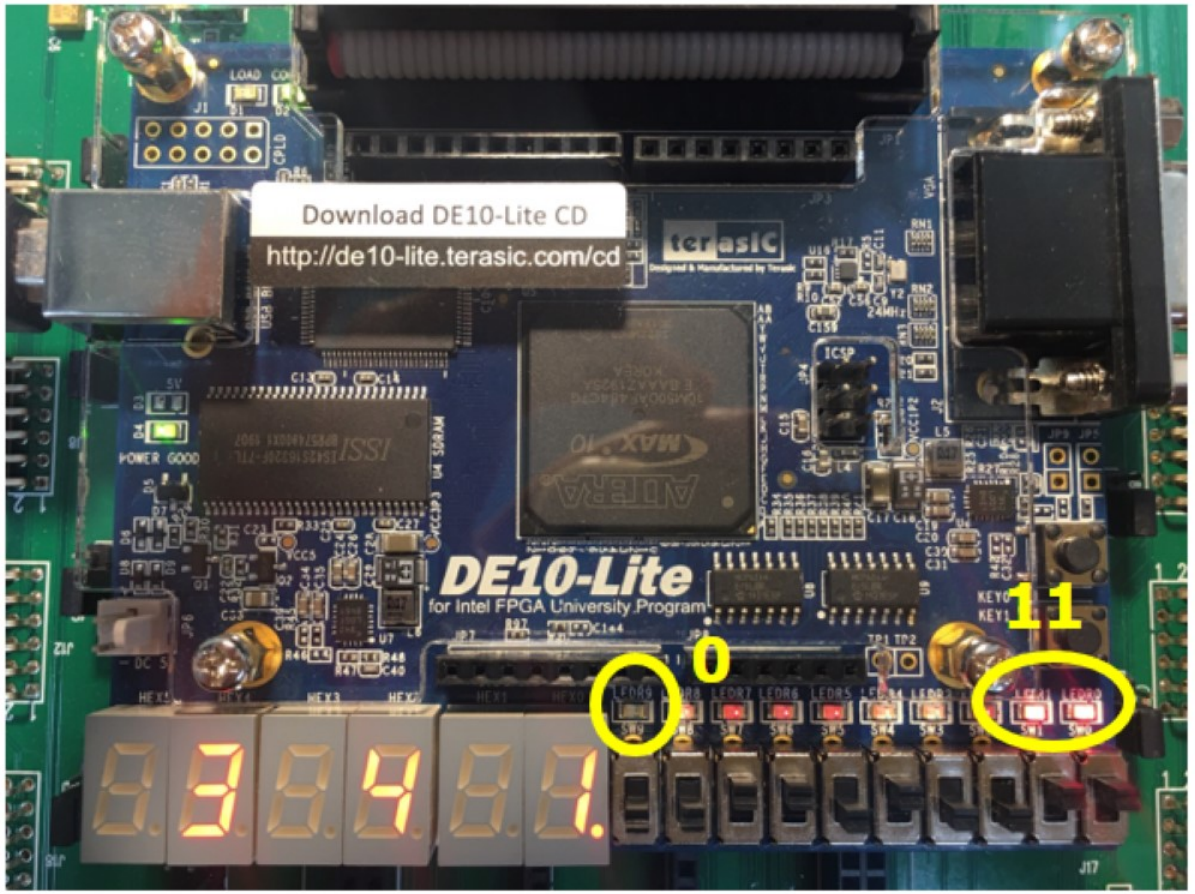
Verification 3

- a=3, b=4, sel=10
- sum = a+b = 7



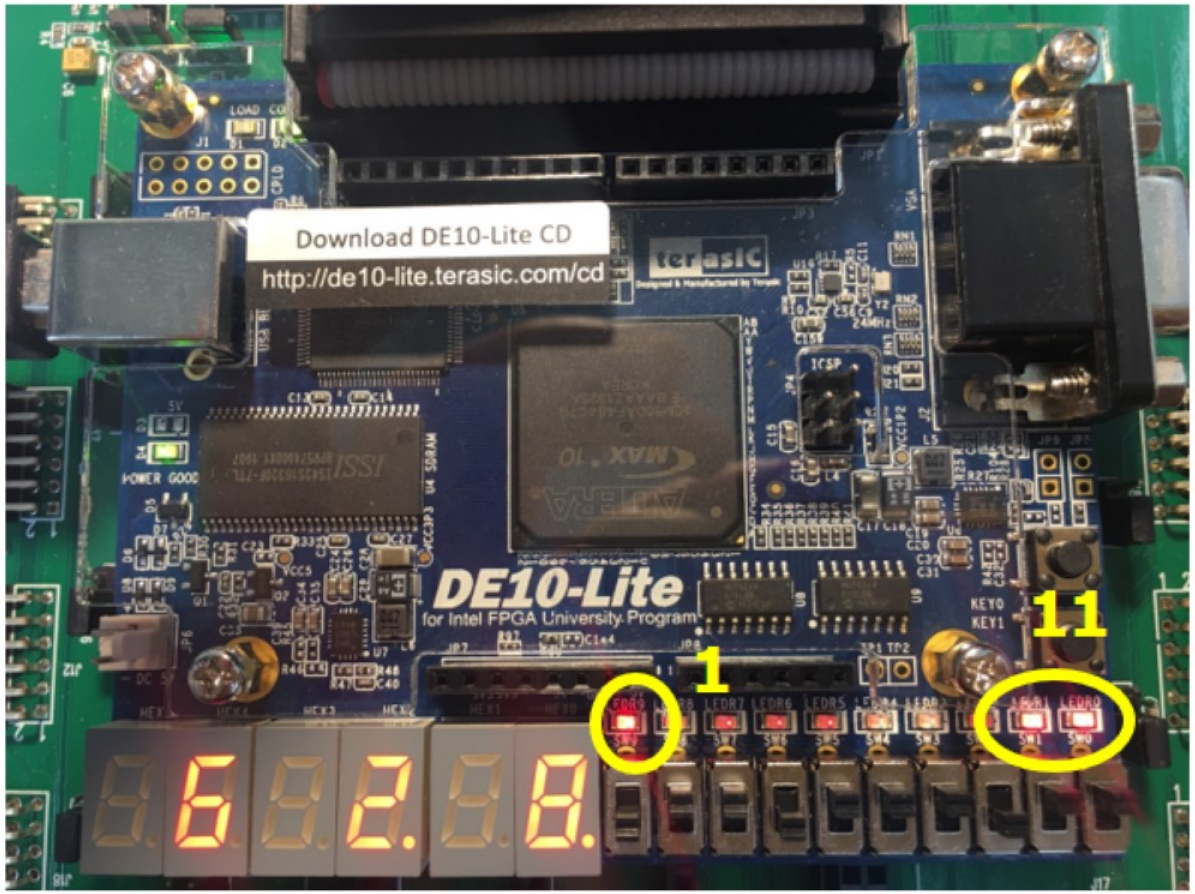
Verification 4

- a=3, b=4, sel=11
- sum = a-b = -1



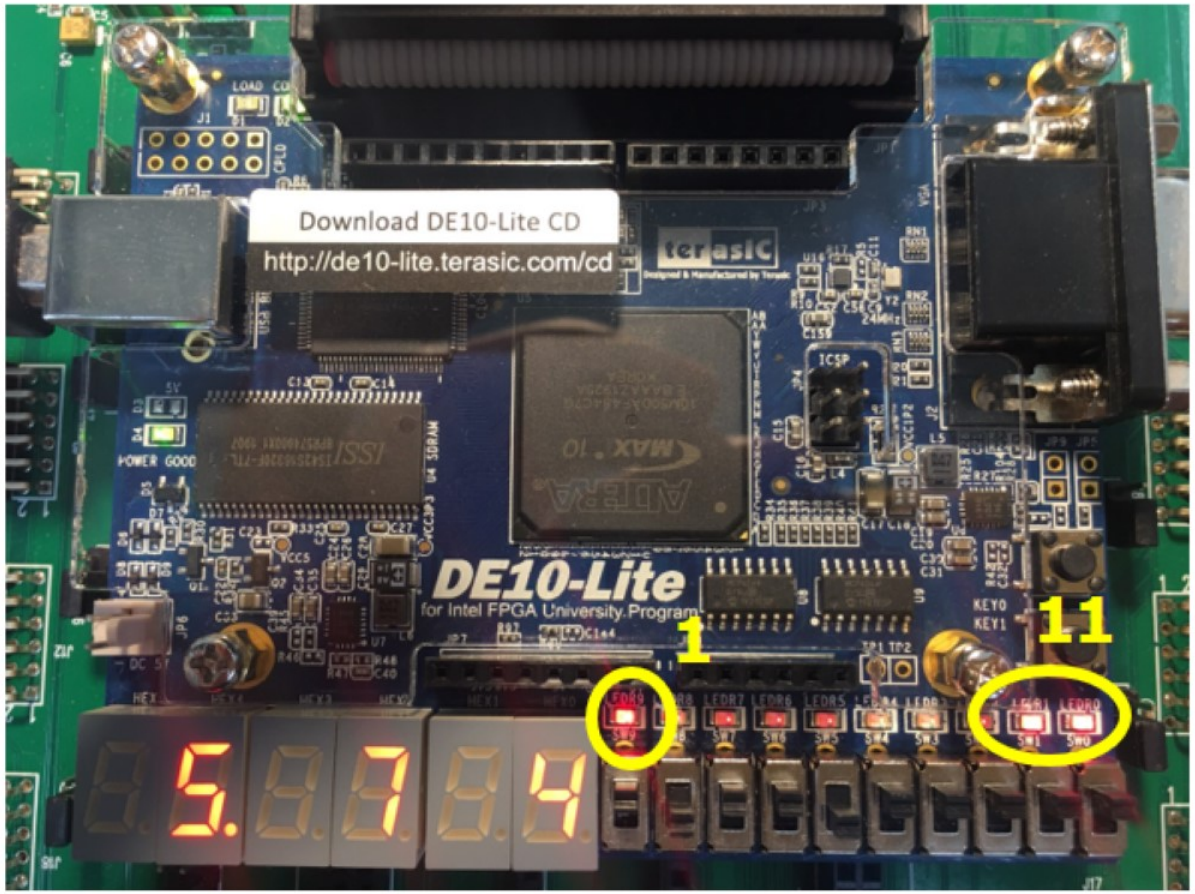
Verification 5

- a=6, b=-2, sel=11
- sum = a-b = 8 **(overflow)**



Verification 6

- a=-5, b=7, sel=11
- sum = a-b = -12 **(underflow)**



File List

- Verilog files
  - add_half.v
  - add_full.v
  - add_4.v
  - addsub_4.v
  - addmux_4.v
  - num_to_seg7.v
  - addseg7_top.v
- Waveform files
  - addmux_4.vwf

實驗結果驗收

- 請老師或助教驗收

  1. addmux_4 波形圖

  2. addseg7_top 電路於實驗板之行為