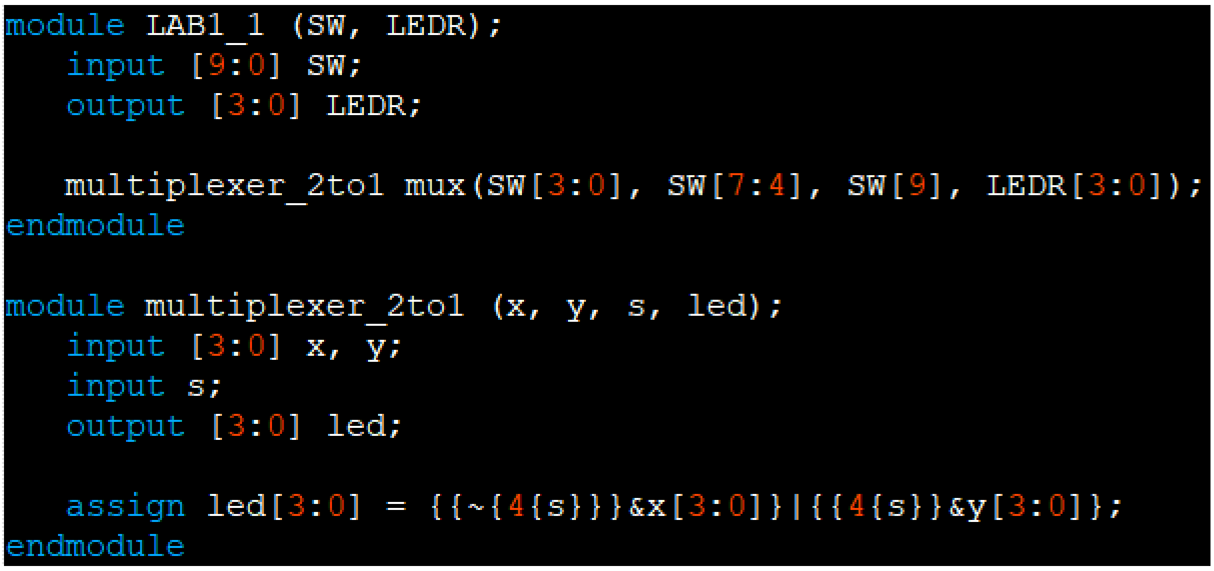
題目：Multiplexers, Number and Displays

繳交日期：2023/07/09

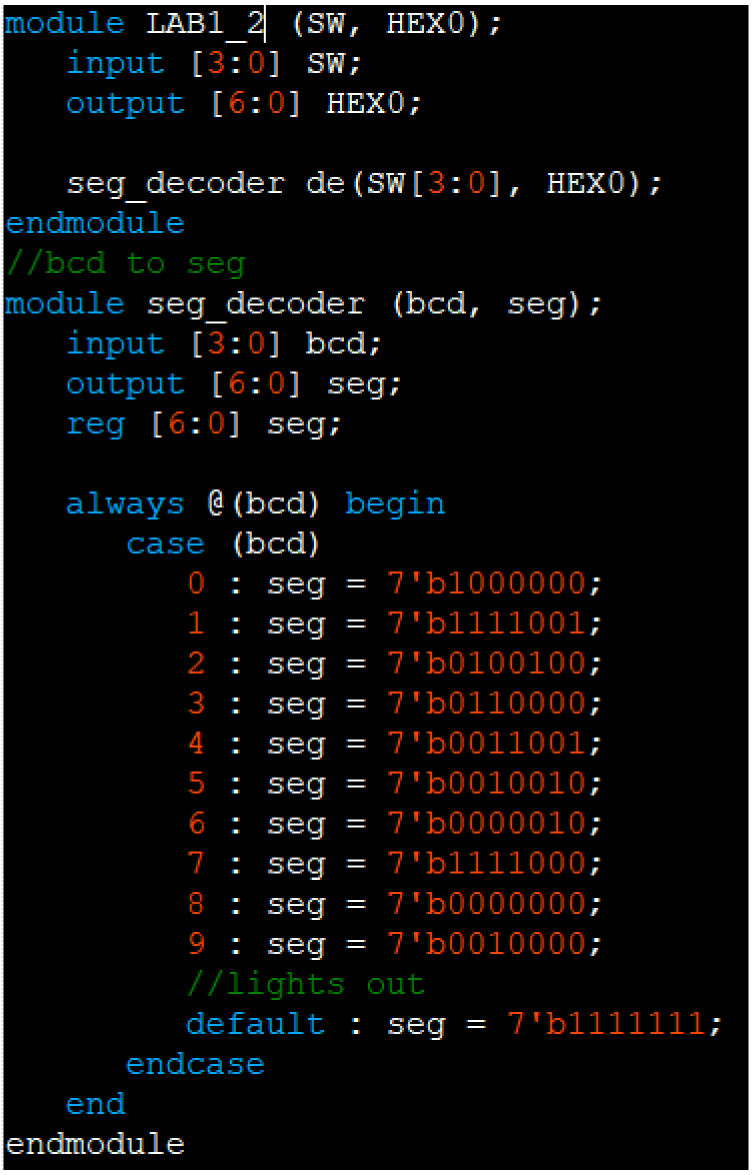
1. 實驗程式碼

◆ **lab1_1→2-to1 multiplexer**



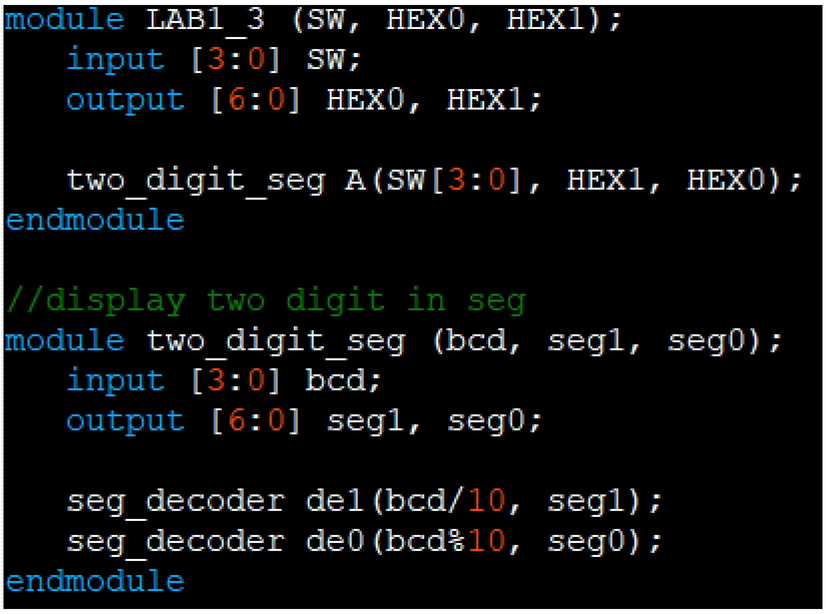
```
module LAB1_1 (SW, LEDR);
   input [9:0] SW;
   output [3:0] LEDR;

   multiplexer_2to1 mux(SW[3:0], SW[7:4], SW[9], LEDR[3:0]);
endmodule

module multiplexer_2to1 (x, y, s, led);
   input [3:0] x, y;
   input s;
   output [3:0] led;

   assign led[3:0] = {{~{4{s}}}&x[3:0]}|{{4{s}}&y[3:0]};
endmodule
```

◆ **lab1_2→7-segment displays**

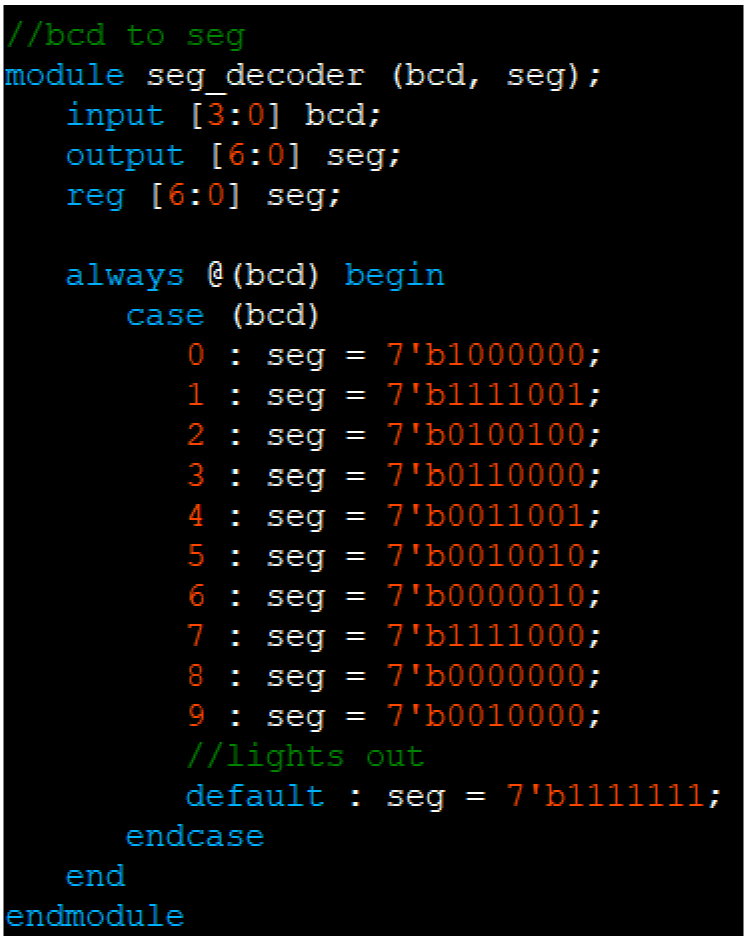


```
module LAB1_2 (SW, HEX0);
   input [3:0] SW;
   output [6:0] HEX0;

   seg_decoder de(SW[3:0], HEX0);
endmodule
//bcd to seg
module seg_decoder (bcd, seg);
   input [3:0] bcd;
   output [6:0] seg;
   reg [6:0] seg;

   always @(bcd) begin
      case (bcd)
         0 : seg = 7'b1000000;
         1 : seg = 7'b1111001;
         2 : seg = 7'b0100100;
         3 : seg = 7'b0110000;
         4 : seg = 7'b0011001;
         5 : seg = 7'b0010010;
         6 : seg = 7'b0000010;
         7 : seg = 7'b1111000;
         8 : seg = 7'b0000000;
         9 : seg = 7'b0010000;
         //lights out
         default : seg = 7'b1111111;
      endcase
   end
endmodule
```

- **lab1_3→Binary coded decimal**

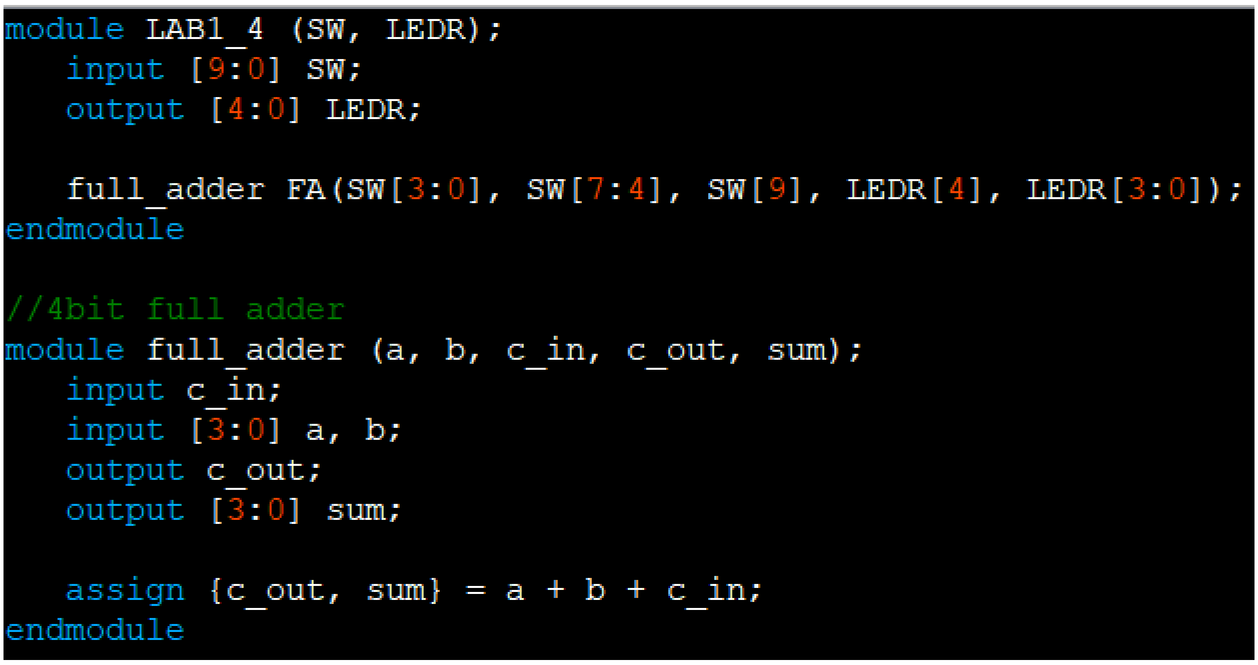


```
module LAB1_3 (SW, HEX0, HEX1);
   input [3:0] SW;
   output [6:0] HEX0, HEX1;

   two_digit_seg A(SW[3:0], HEX1, HEX0);
endmodule

//display two digit in seg
module two_digit_seg (bcd, seg1, seg0);
   input [3:0] bcd;
   output [6:0] seg1, seg0;

   seg_decoder de1(bcd/10, seg1);
   seg_decoder de0(bcd%10, seg0);
endmodule
```



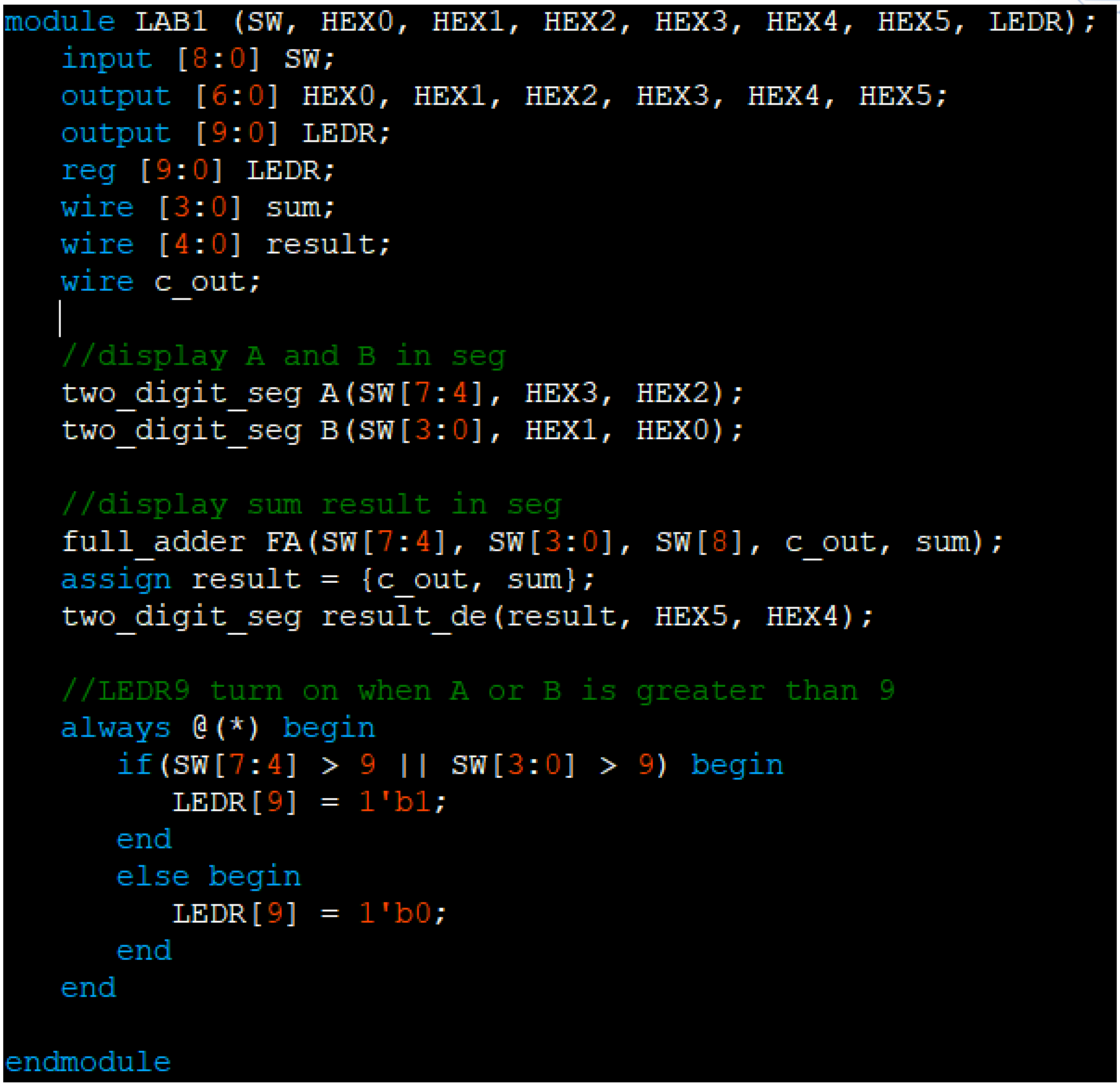
```
//bcd to seg
module seg_decoder (bcd, seg);
   input [3:0] bcd;
   output [6:0] seg;
   reg [6:0] seg;

   always @(bcd) begin
      case (bcd)
         0 : seg = 7'b1000000;
         1 : seg = 7'b1111001;
         2 : seg = 7'b0100100;
         3 : seg = 7'b0110000;
         4 : seg = 7'b0011001;
         5 : seg = 7'b0010010;
         6 : seg = 7'b0000010;
         7 : seg = 7'b1111000;
         8 : seg = 7'b0000000;
         9 : seg = 7'b0010000;
         //lights out
         default : seg = 7'b1111111;
      endcase
   end
endmodule
```

- **lab1_4→Full adder**



```
module LAB1_4 (SW, LEDR);
   input [9:0] SW;
   output [4:0] LEDR;

   full_adder FA(SW[3:0], SW[7:4], SW[9], LEDR[4], LEDR[3:0]);
endmodule

//4bit full adder
module full_adder (a, b, c_in, c_out, sum);
   input c_in;
   input [3:0] a, b;
   output c_out;
   output [3:0] sum;

   assign {c_out, sum} = a + b + c_in;
endmodule
```

- **lab1_5→Add two 1-digit BCD numbers**



```
module LAB1 (SW, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, LEDR);
   input [8:0] SW;
   output [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   output [9:0] LEDR;
   reg [9:0] LEDR;
   wire [3:0] sum;
   wire [4:0] result;
   wire c_out;

   //display A and B in seg
   two_digit_seg A(SW[7:4], HEX3, HEX2);
   two_digit_seg B(SW[3:0], HEX1, HEX0);

   //display sum result in seg
   full_adder FA(SW[7:4], SW[3:0], SW[8], c_out, sum);
   assign result = {c_out, sum};
   two_digit_seg result_de(result, HEX5, HEX4);

   //LEDR9 turn on when A or B is greater than 9
   always @(*) begin
      if(SW[7:4] > 9 || SW[3:0] > 9) begin
         LEDR[9] = 1'b1;
      end
      else begin
         LEDR[9] = 1'b0;
      end
   end

endmodule
```

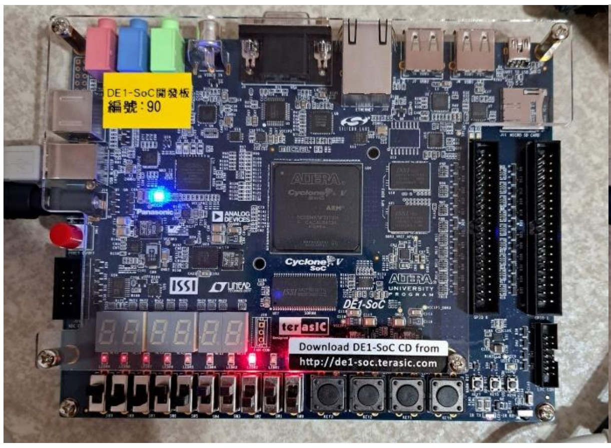
```
//bcd to seg
module seg_decoder (bcd, seg);
   input [3:0] bcd;
   output [6:0] seg;
   reg [6:0] seg;

   always @(bcd) begin
      case (bcd)
         0 : seg = 7'b1000000;
         1 : seg = 7'b1111001;
         2 : seg = 7'b0100100;
         3 : seg = 7'b0110000;
         4 : seg = 7'b0011001;
         5 : seg = 7'b0010010;
         6 : seg = 7'b0000010;
         7 : seg = 7'b1111000;
         8 : seg = 7'b0000000;
         9 : seg = 7'b0010000;
         //lights out
         default : seg = 7'b1111111;
      endcase
   end
endmodule
```

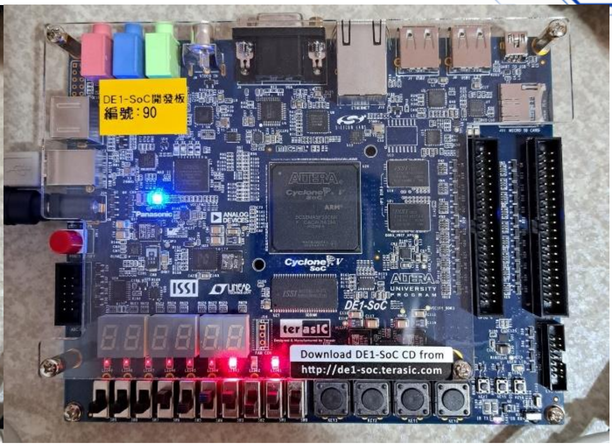
```
//display two digit in seg
module two_digit_seg (bcd, seg1, seg0);
   input [4:0] bcd;
   output [6:0] seg1, seg0;

   seg_decoder de1(bcd/10, seg1);
   seg_decoder de0(bcd%10, seg0);
endmodule
```

```
//4bit full adder
module full_adder (a, b, c_in, c_out, sum);
   input c_in;
   input [3:0] a, b;
   output c_out;
   output [3:0] sum;

   assign {c_out, sum} = a + b + c_in;
endmodule
```

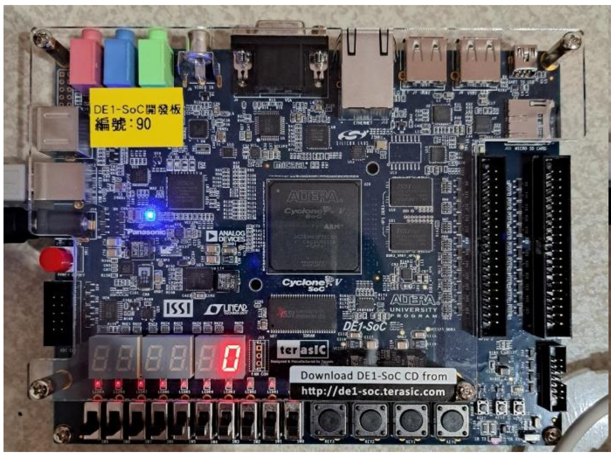
2. 實驗結果

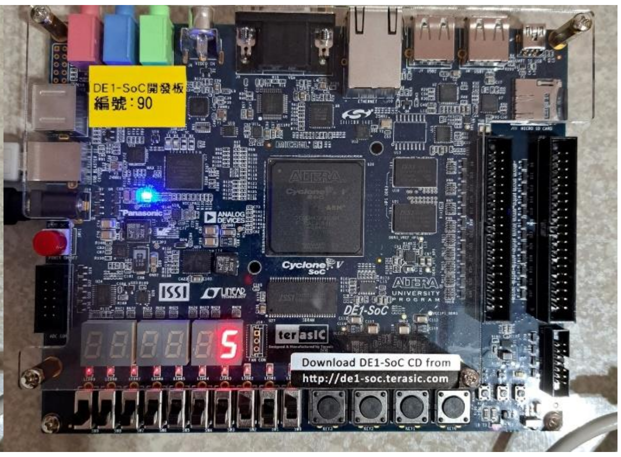
- **lab1_1**

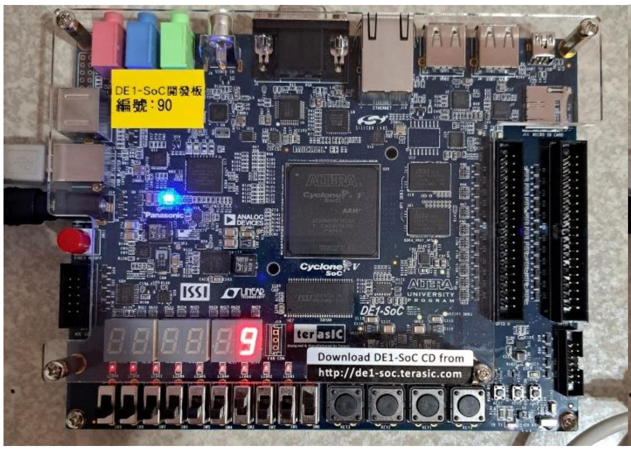

DE1-SoC開發板
編號：90
Download DE1-SoC CD from
http://de1-soc.terasic.com

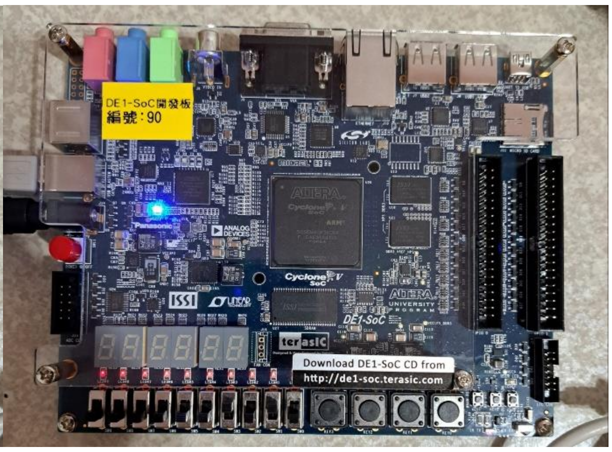

DE1-SoC開發板
編號：90
Download DE1-SoC CD from
http://de1-soc.terasic.com

- **lab1_2**

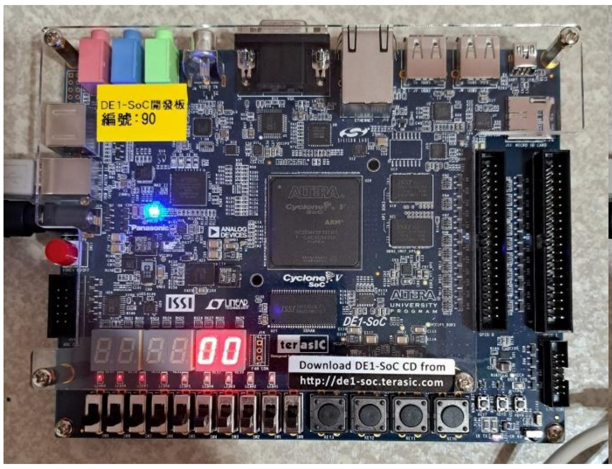

DE1-SoC開發板
編號：90
Download DE1-SoC CD from
http://de1-soc.terasic.com

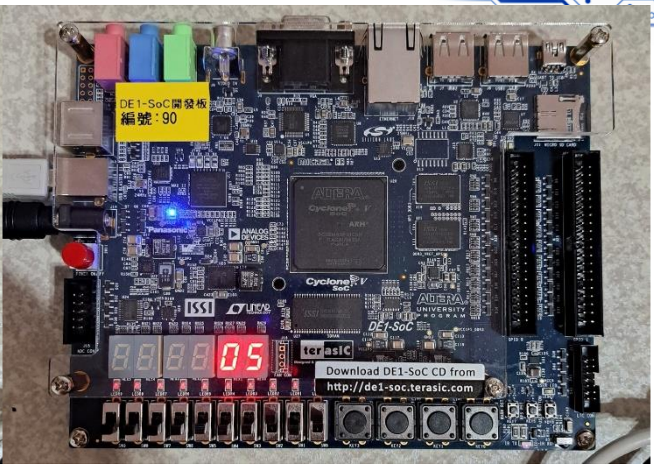

DE1-SoC開發板
編號：90
Download DE1-SoC CD from
http://de1-soc.terasic.com

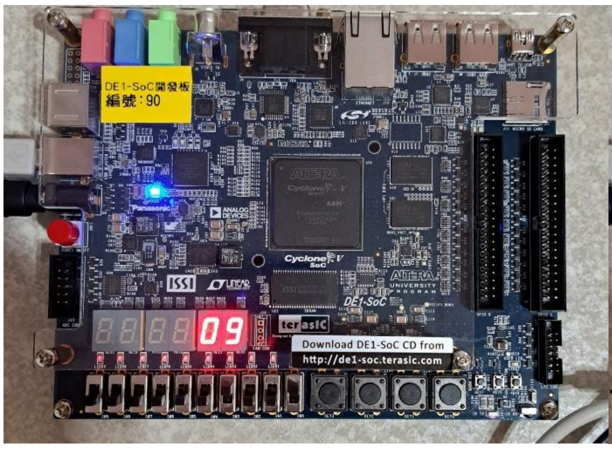

DE1-SoC開發板
編號：90
Download DE1-SoC CD from
http://de1-soc.terasic.com

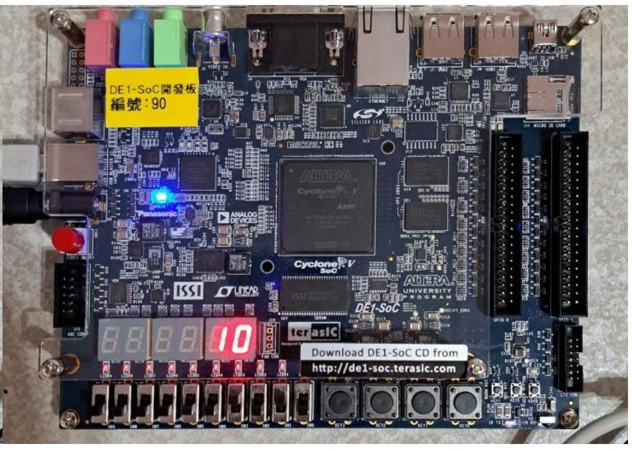

DE1-SoC開發板
編號：90
Download DE1-SoC CD from
http://de1-soc.terasic.com

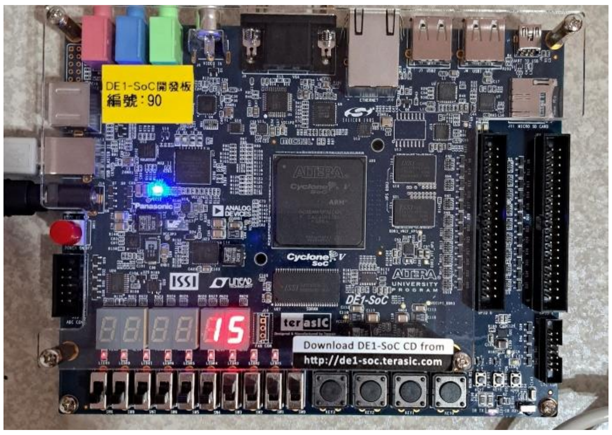
- **lab1_3**


DE1-SoC開發板
編號：90
00

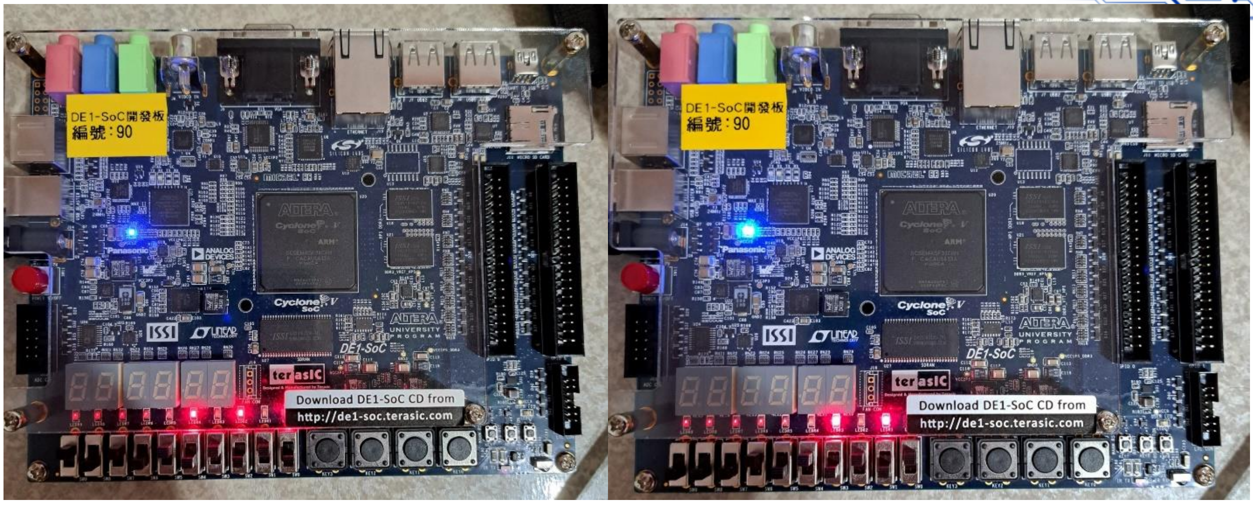

DE1-SoC開發板
編號：90
05


DE1-SoC開發板
編號：90
09

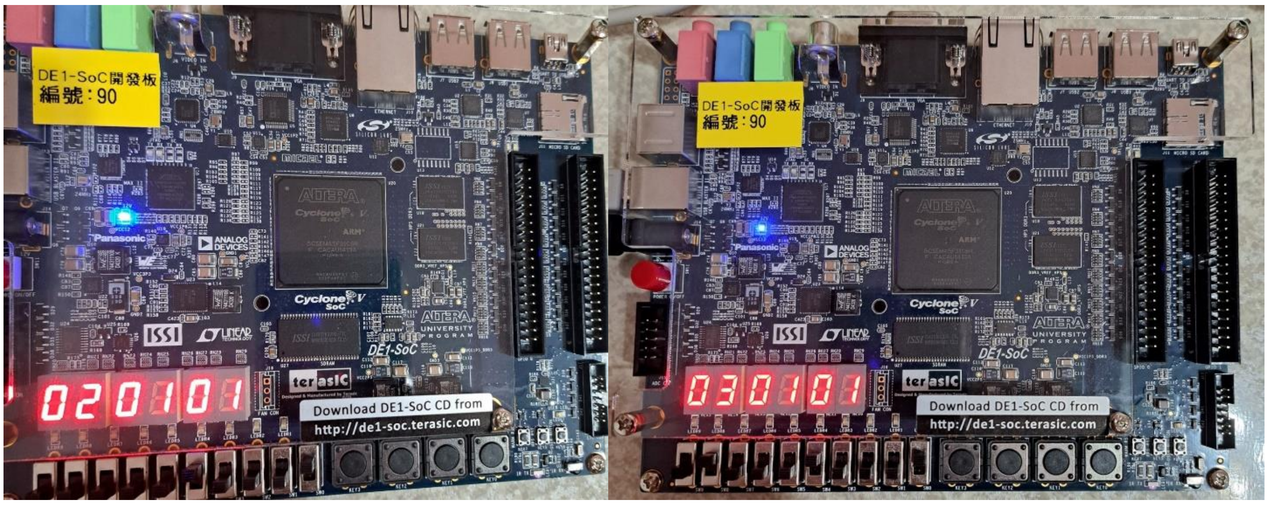

DE1-SoC開發板
編號：90
10

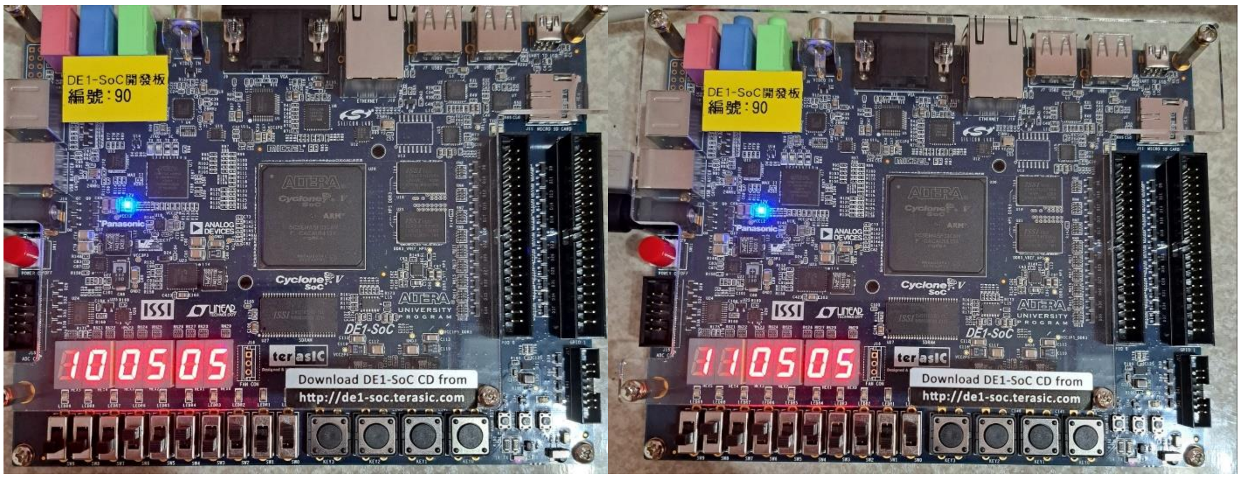

DE1-SoC開發板
編號：90
15

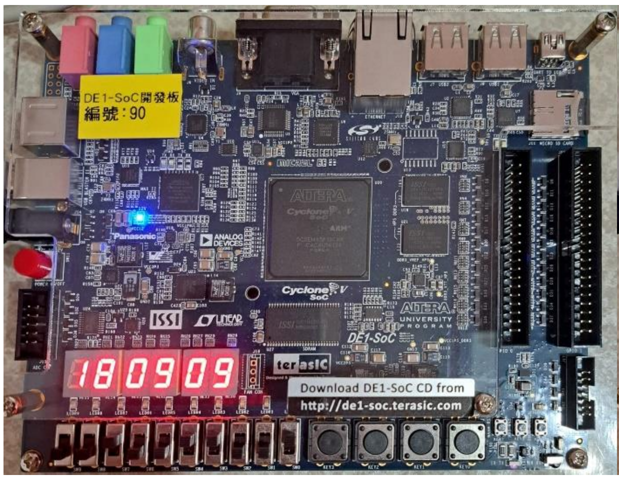
◆ **lab1_4**

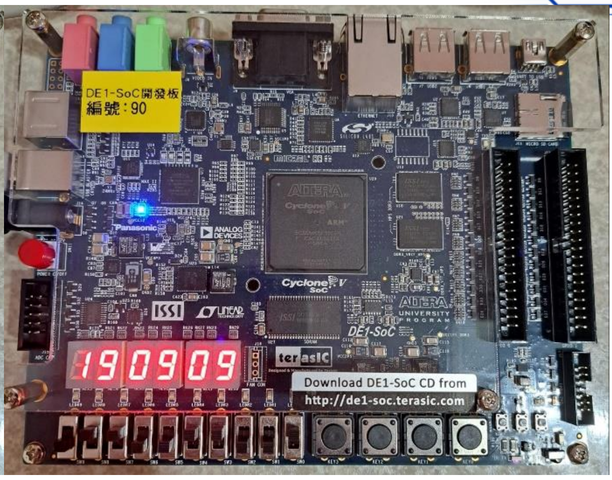

DE1-SoC開發板
編號：90
Download DE1-SoC CD from
http://de1-soc.terasic.com

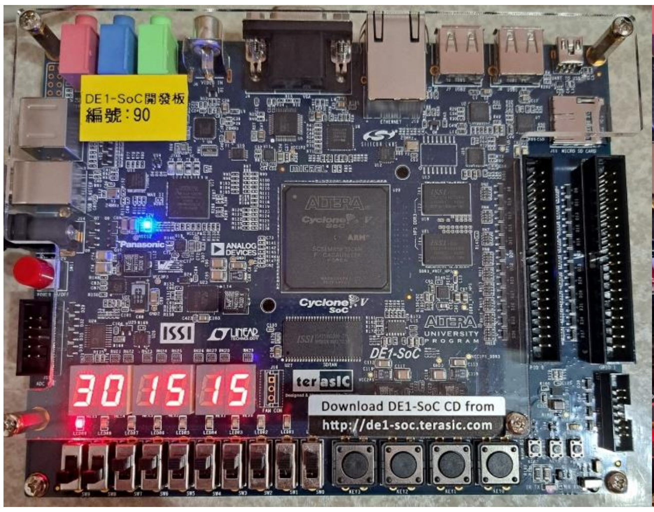
◆ lab1_5

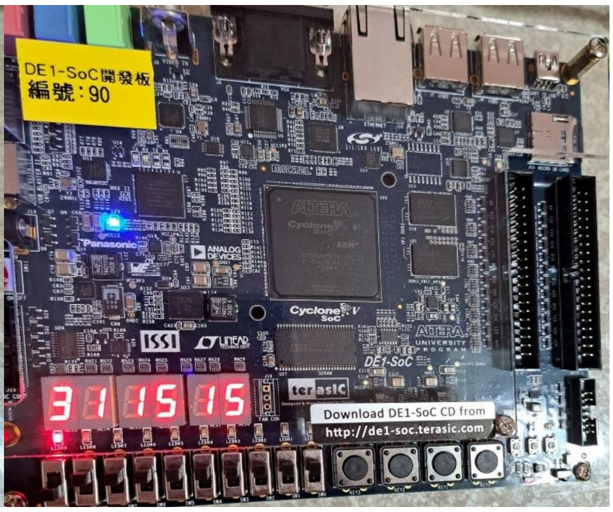

DE1-SoC開發板
編號：90
020101
030101
Download DE1-SoC CD from
http://de1-soc.terasic.com

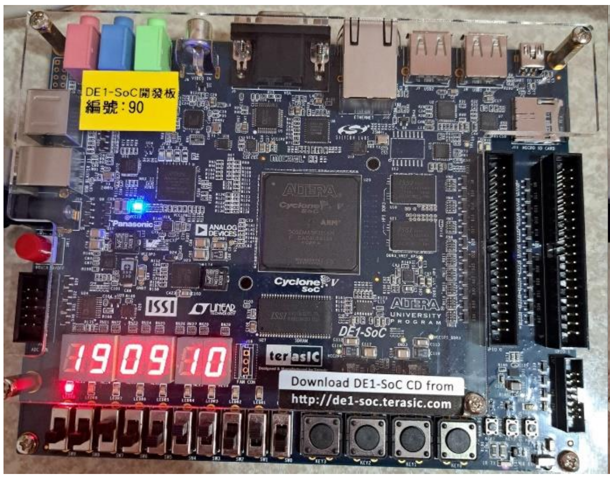

DE1-SoC開發板
編號：90
100505
110505
Download DE1-SoC CD from
http://de1-soc.terasic.com

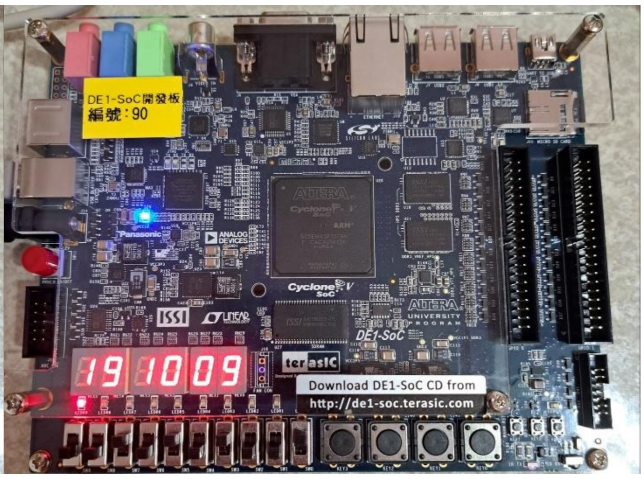






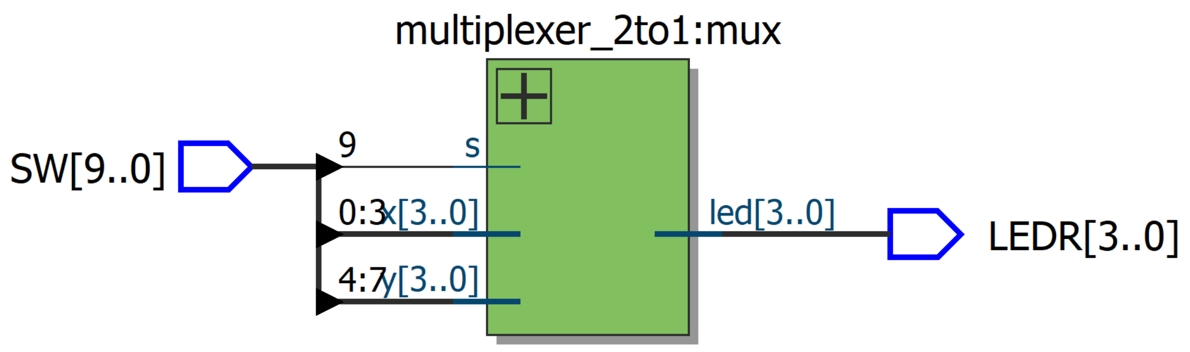


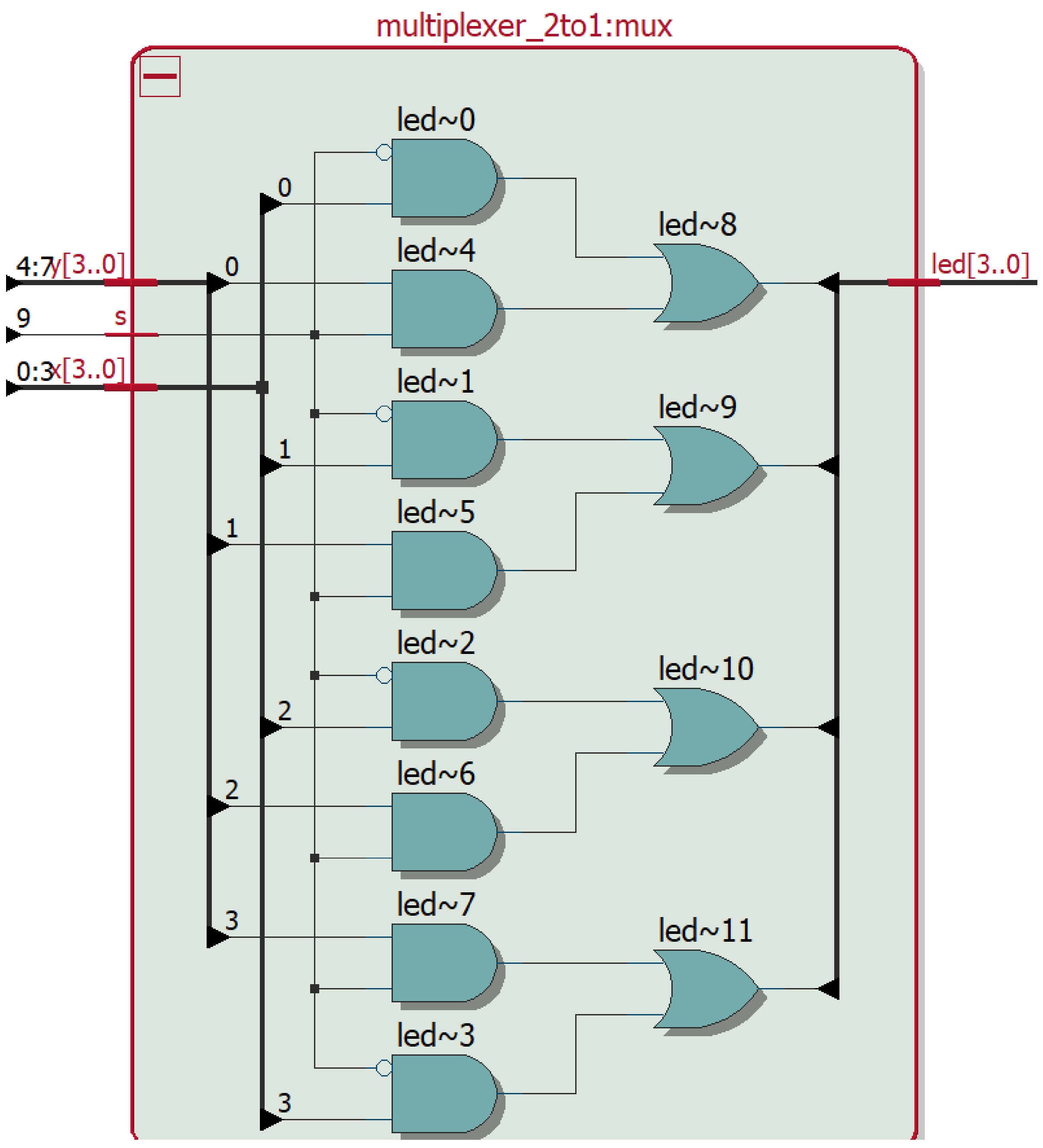




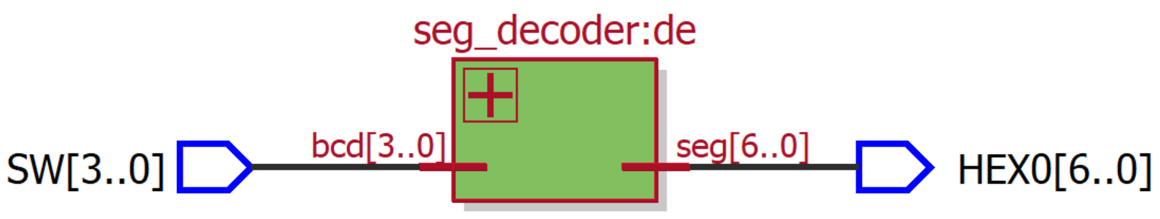
3. **RTL 佈局**

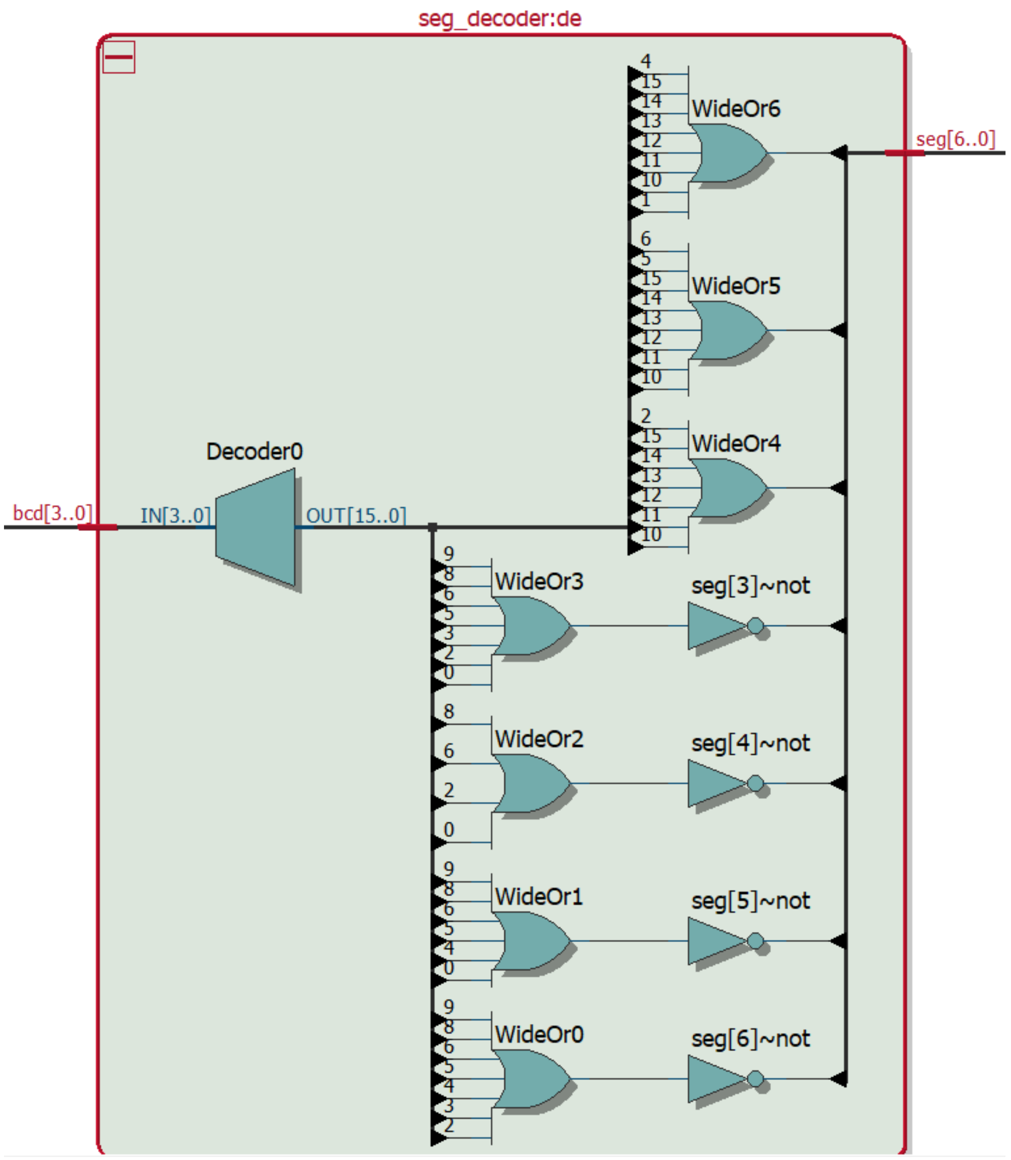
- **lab1_1**


multiplexer_2to1:mux
SW[9..0]
9
s
0:3
x[3..0]
4:7
y[3..0]
led[3..0]
LEDR[3..0]

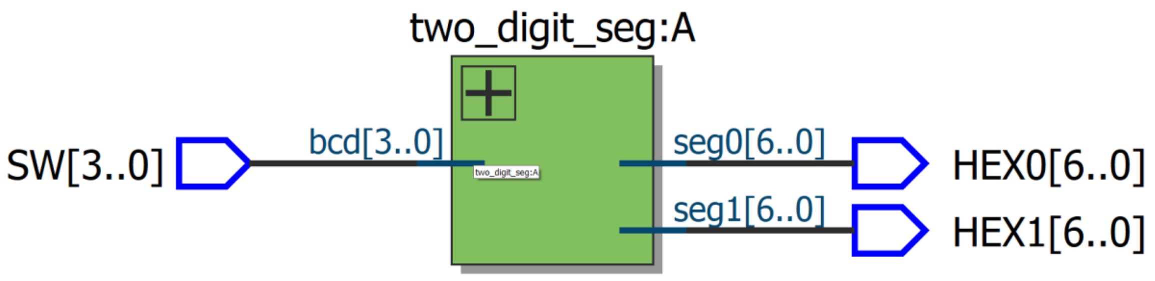

multiplexer_2to1:mux
4:7
y[3..0]
9
s
0:3
x[3..0]
led~0
led~4
led~1
led~5
led~2
led~6
led~7
led~3
led~8
led~9
led~10
led~11
led[3..0]
0
1
2
3

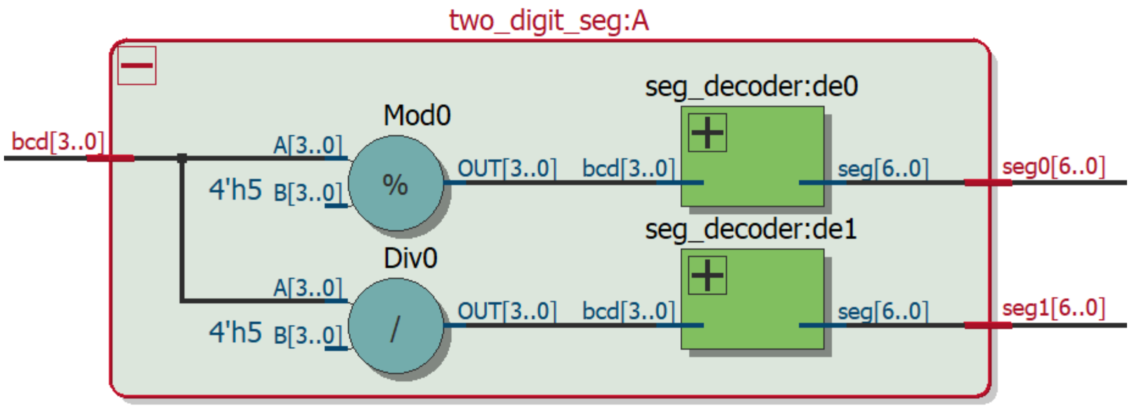
- **lab1_2**

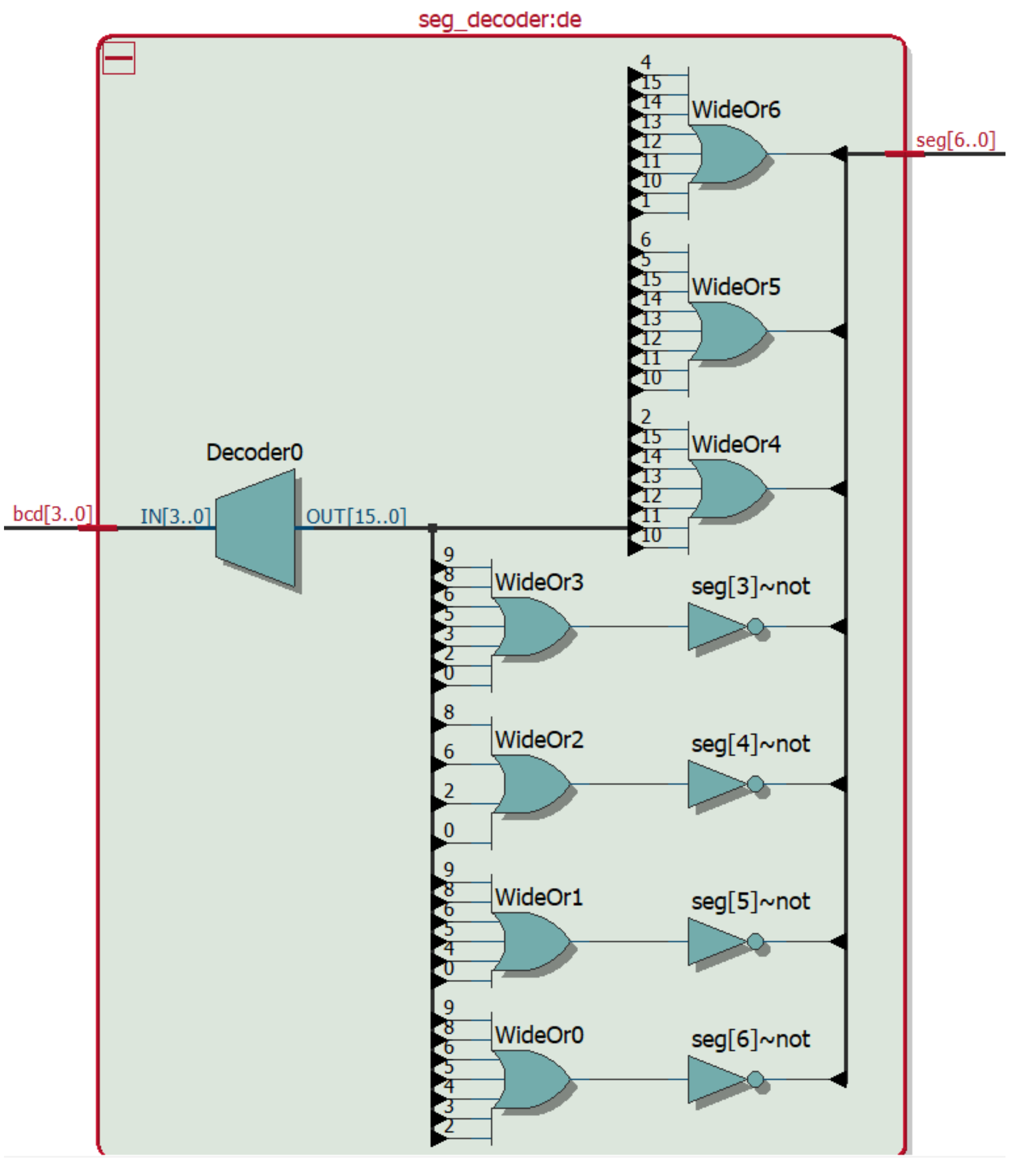

seg_decoder:de
SW[3..0]
bcd[3..0]
seg[6..0]
HEX0[6..0]


seg_decoder:de
bcd[3..0]
IN[3..0]
Decoder0
OUT[15..0]
WideOr6
WideOr5
WideOr4
WideOr3
WideOr2
WideOr1
WideOr0
seg[3]~not
seg[4]~not
seg[5]~not
seg[6]~not
seg[6..0]

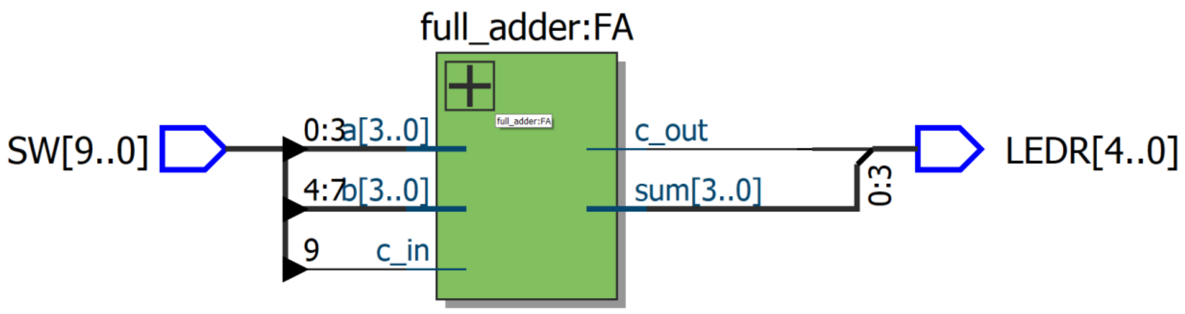
- **lab1_3**

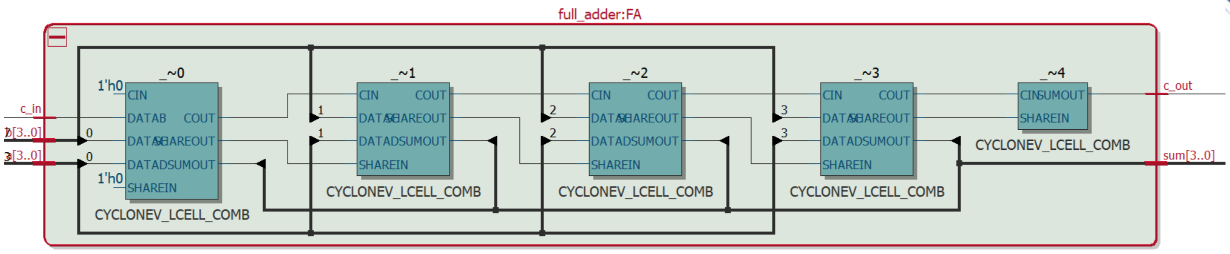

two_digit_seg:A
SW[3..0]
bcd[3..0]
seg0[6..0]
seg1[6..0]
HEX0[6..0]
HEX1[6..0]


two_digit_seg:A
bcd[3..0]
Mod0
A[3..0]
4'h5
B[3..0]
%
OUT[3..0]
seg_decoder:de0
bcd[3..0]
seg[6..0]
seg0[6..0]
Div0
A[3..0]
4'h5
B[3..0]
/
OUT[3..0]
seg_decoder:de1
bcd[3..0]
seg[6..0]
seg1[6..0]

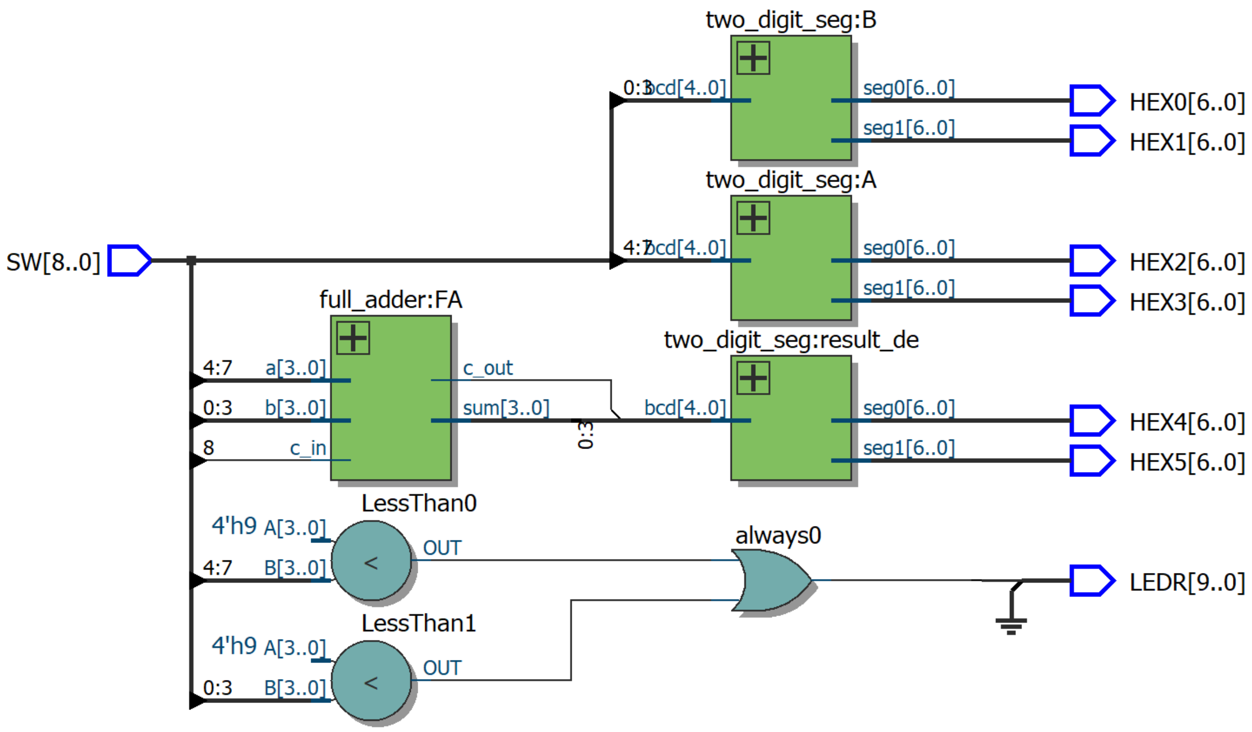

seg_decoder:de
bcd[3..0]
IN[3..0]
Decoder0
OUT[15..0]
WideOr6
WideOr5
WideOr4
WideOr3
WideOr2
WideOr1
WideOr0
seg[3]~not
seg[4]~not
seg[5]~not
seg[6]~not
seg[6..0]

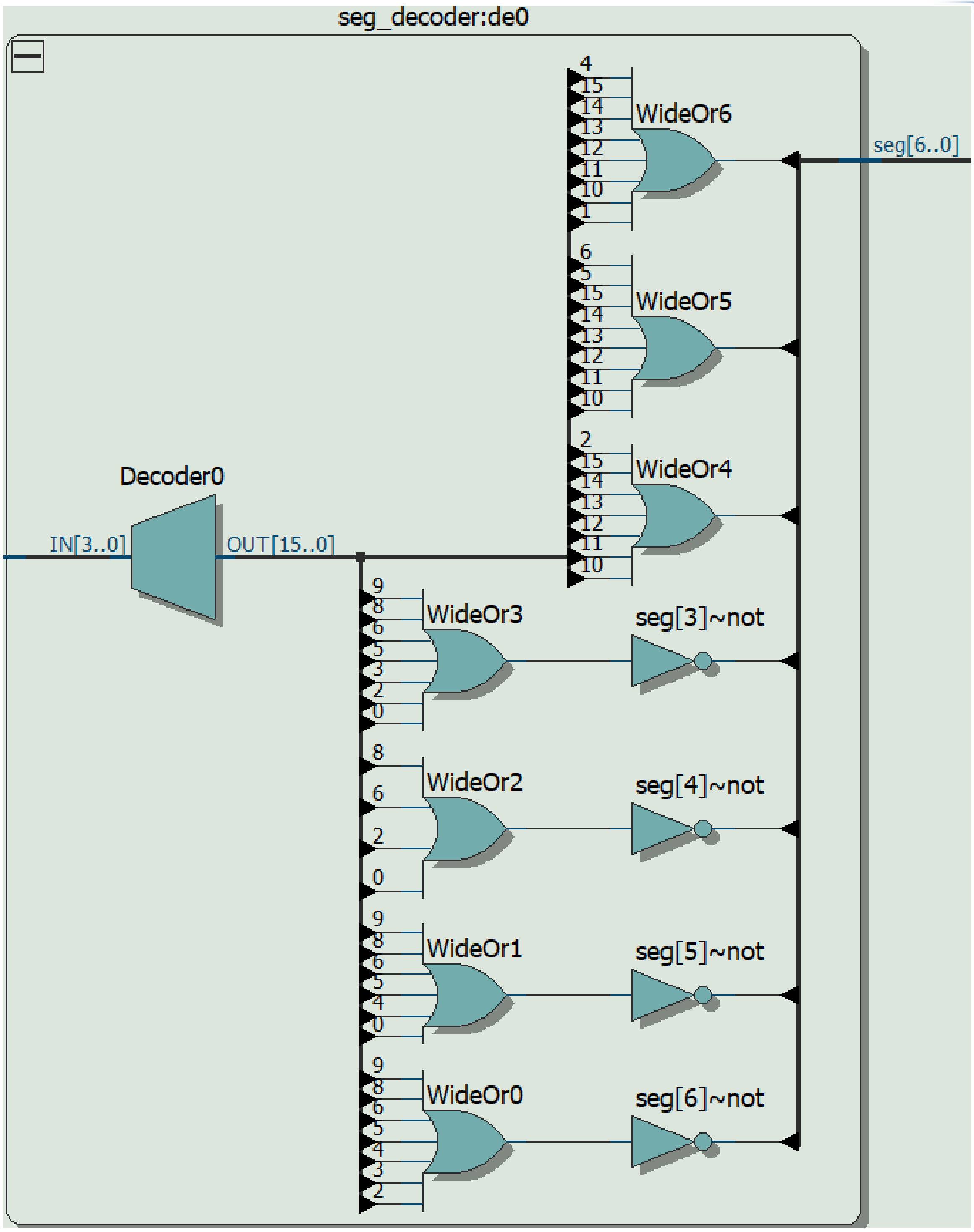
◆ **lab1_4**

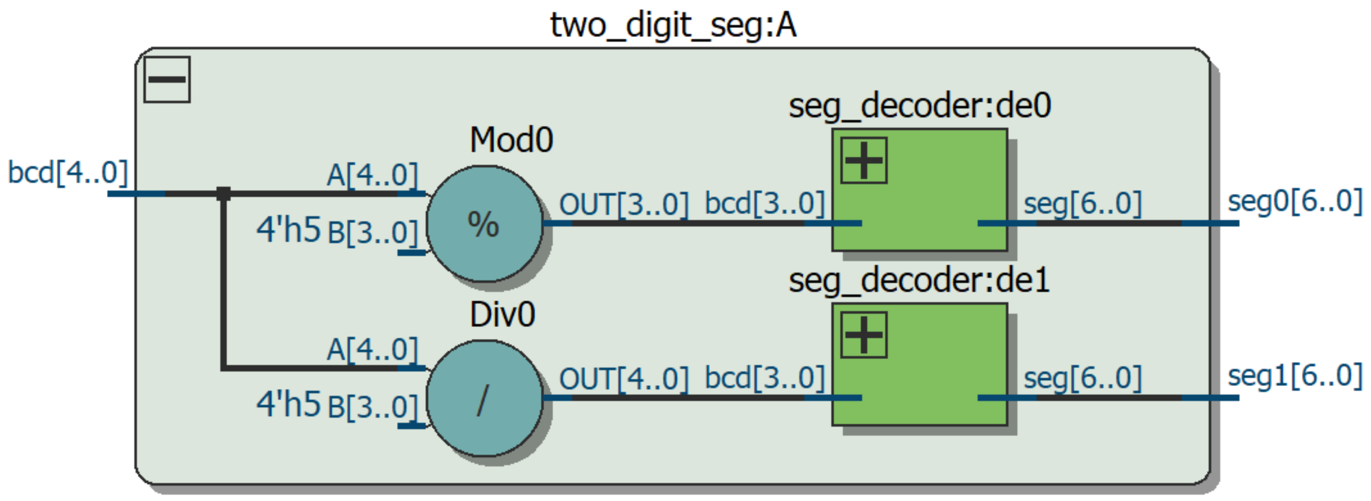

full_adder:FA
SW[9..0]
0:3
a[3..0]
4:7
b[3..0]
9
c_in
c_out
sum[3..0]
0:3
LEDR[4..0]

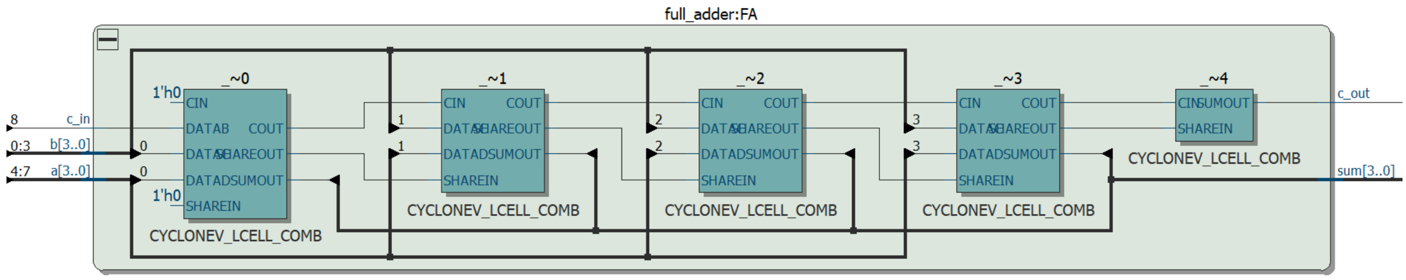

full_adder:FA
c_in
b[3..0]
a[3..0]
~0
~1
~2
~3
~4
1'h0
CIN
DATAB
DATAD
SHAREIN
COUT
SHAREOUT
SUMOUT
CYCLONEV_LCELL_COMB
c_out
sum[3..0]

- **lab1_5**


two_digit_seg:B
two_digit_seg:A
two_digit_seg:result_de
full_adder:FA
SW[8..0]
bcd[4..0]
seg0[6..0]
seg1[6..0]
HEX0[6..0]
HEX1[6..0]
HEX2[6..0]
HEX3[6..0]
HEX4[6..0]
HEX5[6..0]
0:3
4:7
8
a[3..0]
b[3..0]
c_in
c_out
sum[3..0]
LessThan0
LessThan1
4'h9
A[3..0]
B[3..0]
OUT
always0
LEDR[9..0]


seg_decoder:de0
Decoder0
IN[3..0]
OUT[15..0]
WideOr6
WideOr5
WideOr4
WideOr3
WideOr2
WideOr1
WideOr0
seg[3]~not
seg[4]~not
seg[5]~not
seg[6]~not
seg[6..0]





4. 問題與討論

- reg 和 wire 類似 C/C++的變數，但是有位寬，預設是一位元。input, output 和 inout 預設是 wire，要改成 reg 需要另外宣告。

- begin end 相當於 C/C++的{}，可執行多行指令，內部只能放 reg。

- 賦值(連接線路)時，wire 要使用 assign，而 reg 不用。

- 一般來說，只有在時序電路(always block 的敏感列表中是 clock 形式)，reg 是 register，可以重複賦值，不然還是和 wire 一樣是 net(一條導線)，不能重複賦值。

- always block：@後面放列表訊號，這些訊號改變才會觸發 block，沒有@代表會一直執行 block

  *代表 block 內所有輸入訊號
  敏感列表只要沒有 clock 就是組合邏輯電路

- assign 訊號只要右端訊號改變就會執行該語句的計算，always block 只要敏感變量訊號改變就會執行該 block

- 不能在 always block 裡面初始化 module

- {}內可以串接不同變數，串接順序由右至左為 LSB 到 MSB，在 full adder 有使用此技巧