邏輯系統實習

實驗六

虛擬儀器(一) + Verilog語法介紹(三)：行為層次-組合電路

國立成功大學 電機系

2016

大綱

- Verilog中的四種描述層次
- 程序區塊
- 時序控制
  - 準位觸發
- 程序指定
  - 阻礙指定
- 條件敘述與多路徑分支
- 文字巨集與參數
- 資料處理層次與行為層次之比較
- 數位電路設計流程
- VeriInstrument
- 基礎題(一)
  - 4bit簡易算數邏輯單元
- 基礎題(二)
  - 4bit加法器與七段顯示器
- 挑戰題
  - 5bit簡易算數邏輯單元與七段顯示器
- 實驗結報繳交

Verilog中的四種描述層次

- 行為或演算法層次 (lab6~)
  - 在這個層次中，只需要考慮模組的功能或演算法，不需要考慮硬體方面的詳細電路是如何。
- 資料處理層次 (lab5)
  - 在這個層次中，只需要指名資料處理的方式，像是資料如何在暫存器中儲存與傳送以及資料在設計裡處理的方式。
- 邏輯閘層次 (lab4)
  - 在這個層次中，模組是邏輯閘連接而成，如同以前用邏輯閘描繪電路一樣。
- 低階交換層次 (x)
  - 在這個層次中，線路是由開關與儲存點構成，需要知道電晶體的元件特性。

越上層設計層次越高

程序區塊 (1/2)

- 程序區塊是行為層次中的基本用法，包含initial與always兩種區塊。
  - 所有的行為層次語法皆必須寫在initial或always區塊中。

```
module testbench();
   reg [3:0]in1, in2;
   reg carry_in;
   wire [3:0]sum;
   wire carry_out;

   adder_4bit x1(in1, in2, carry_in, sum, carry_out);

   initial begin
         in1=0;  in2=1;  carry_in=0;
      #1 in1=1;  in2=2;  carry_in=1;
      #1 in1=2;  in2=3;  carry_in=0;
      #1 in1=3;  in2=4;  carry_in=1;
      #1 in1=4;  in2=5;  carry_in=0;
      #1 in1=5;  in2=6;  carry_in=1;
      #1 in1=6;  in2=7;  carry_in=0;
      #1 in1=7;  in2=8;  carry_in=1;
      #1 in1=8;  in2=9;  carry_in=0;
      #1 in1=9;  in2=10; carry_in=1;
      #1 in1=10; in2=11; carry_in=0;
      #1 in1=11; in2=12; carry_in=1;
      #1 in1=12; in2=13; carry_in=0;
      #1 in1=13; in2=14; carry_in=1;
      #1 in1=14; in2=15; carry_in=0;
      #1 $finish;
   end

endmodule
```

testbench

initial區塊啟始於模擬時間零，僅執行一次，通常用於初始化、監控邏輯值變化、顯示波形等

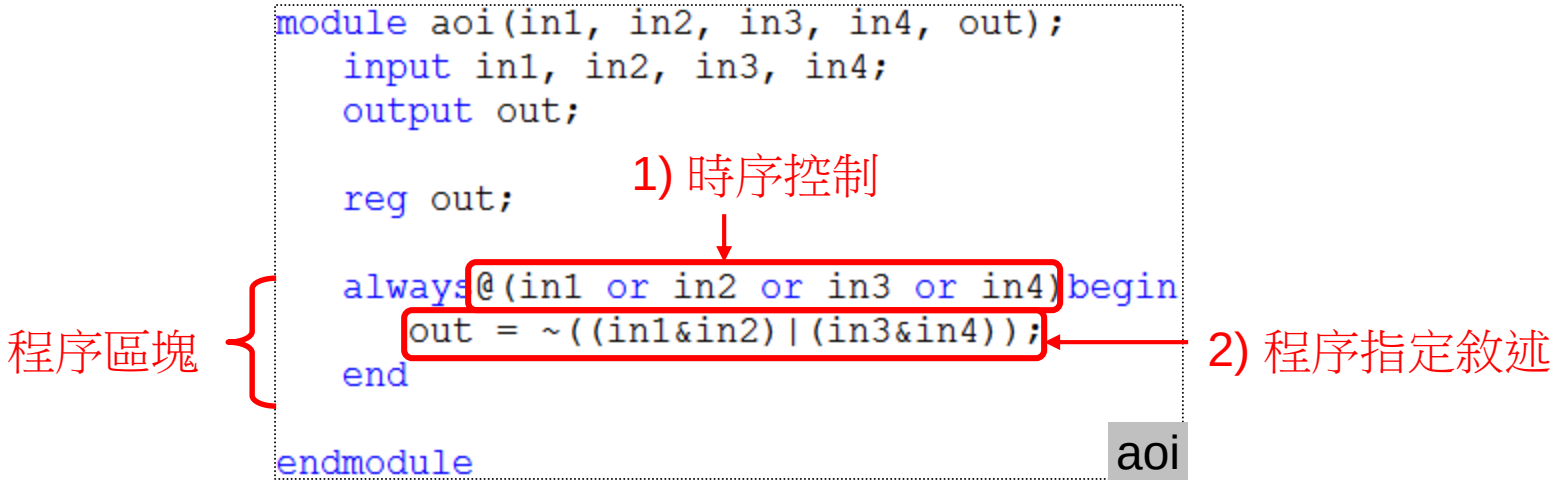
always區塊起始於模擬時間零，以迴圈的形式持續重複執行，通常用於描述持續重複工作的邏輯電路

```
module adder_4bit(in1, in2, carry_in, sum, carry_out);
   input [3:0]in1, in2;
   input carry_in;
   output [3:0]sum;
   output carry_out;

   reg [3:0]sum;
   reg carry_out;

   always@(in1 or in2 or carry_in)begin
      {carry_out, sum} = in1 + in2 + carry_in;
   end

endmodule
```

adder_4bit

程序區塊 (2/2)

- 程序區塊包含兩個要素：
  - 1) 時序控制
  - 2) 程序指定敘述



時序控制 (1/2)

- 時序控制是用來設定某程序敘述在某時間被執行，共有三種方式。
  - 1) 簡易延遲
  - 2) 準位觸發
  - 3) 邊緣觸發 (lab7)

```
module alu_4bit(sel, in1, in2, result);
   input [2:0]sel;
   input [3:0]in1, in2;
   output [3:0]result;

   reg [3:0]result;

   always@(sel or in1 or in2)begin
      case(sel)
         3'b000: result=in1+in2;
         3'b001: result=in1-in2;
         3'b010: result=in1&in2;
         3'b011: result=in1|in2;
         3'b100: result=in1^in2;
         3'b101: result=in1>>in2;
         3'b110: result=in1<<in2;
         3'b111: result=in1>in2;
         default: result=4'b0;
      endcase
   end

endmodule
```

2) 準位觸發

alu_4bit

```
module testbench();
   reg clk;
   reg [2:0]sel;
   reg [3:0]in1, in2;
   wire [3:0]result;

   system X1(clk, sel, in1, in2, result);

   initial clk=1'b1;
   always #5 clk=~clk;

   initial begin
            sel=3'b000; in1=9; in2=3;
      #10   sel=3'b001; in1=9; in2=3;
      #10   sel=3'b010; in1=9; in2=3;
      #10   sel=3'b011; in1=9; in2=3;
      #10   sel=3'b100; in1=9; in2=3;
      #10   sel=3'b101; in1=9; in2=3;
      #10   sel=3'b110; in1=9; in2=3;
      #10   sel=3'b111; in1=9; in2=3;
      #10   $finish;
   end

endmodule
```

1) 簡易延遲

testbench

時序控制 (2/2)
準位觸發

- 準位觸發又稱為非同步觸發，在使用always區塊描述組合電路時，需要將所有的觸發訊號列在感測列表中。
- 當訊號很多時，可以使用@(*)語法自動將所有可能的觸發訊號加入感測列表中。

```
module alu_4bit(sel, in1, in2, result);
   input [2:0]sel;
   input [3:0]in1, in2;
   output [3:0]result;

   reg [3:0]result;

   always@(sel or in1 or in2)begin
      case(sel)
         3'b000: result=in1+in2;
         3'b001: result=in1-in2;
         3'b010: result=in1&in2;
         3'b011: result=in1|in2;
         3'b100: result=in1^in2;
         3'b101: result=in1>>in2;
         3'b110: result=in1<<in2;
         3'b111: result=in1>in2;
         default: result=4'b0;
      endcase
   end

endmodule
```

alu_4bit

必須將所有可能的觸發訊號一一列出

如果觸發訊號填寫不完全，可能會造成(合成前)模擬與(合成後)驗證結果不相同

```
module alu_4bit(sel, in1, in2, result);
   input [2:0]sel;
   input [3:0]in1, in2;
   output [3:0]result;

   reg [3:0]result;

   always@(*)begin
      case(sel)
         3'b000: result=in1+in2;
         3'b001: result=in1-in2;
         3'b010: result=in1&in2;
         3'b011: result=in1|in2;
         3'b100: result=in1^in2;
         3'b101: result=in1>>in2;
         3'b110: result=in1<<in2;
         3'b111: result=in1>in2;
         default: result=4'b0;
      endcase
   end

endmodule
```

alu_4bit

或使用@(*) 自動加入可能的觸發訊號

程序指定
阻礙指定

- 程序指定可以用來更新暫存器等變數。被指定的變數值將被保持到下次被更新為止。
  - 在程序指定的左邊必須是暫存器(reg)或其他變數等等；而在敘述右邊的運算元可以是任意運算式。
- 程序指定有兩種形態：
  - 1) 阻礙指定
  - 2) 無阻礙指定 (lab7)

阻礙指定(=)敘述會依照其在程序區塊中的位置依序執行

```
module alu_4bit(sel, in1, in2, result);
   input [2:0]sel;
   input [3:0]in1, in2;
   output [3:0]result;

   reg [3:0]result;

   always@(*)begin
      case(sel)
         3'b000: result=in1+in2;
         3'b001: result=in1-in2;
         3'b010: result=in1&in2;
         3'b011: result=in1|in2;
         3'b100: result=in1^in2;
         3'b101: result=in1>>in2;
         3'b110: result=in1<<in2;
         3'b111: result=in1>in2;
         default: result=4'b0;
      endcase
   end

endmodule
```

alu_4bit

條件敘述與多路徑分支 (1/5)

```
module mux_4to1(in1, in2, in3, in4, sel, out);
   input in1, in2, in3, in4;
   input [1:0]sel;
   output out;

   reg out;

   always@(sel or in1 or in2 or in3 or in4)begin
      if(sel==2'b00)out=in1;
      else if(sel==2'b01)out=in2;
      else if(sel==2'b10)out=in3;
      else out=in4;
   end

endmodule
```

mux_4to1

簡單少位元的if-else敘述通常會如同條件運算子(?:)一般合成出多工器電路

但如果是複雜的if-else敘述時，則不一定會合成出大型的多工器電路

簡單與與大型的case敘述通常皆會合成出多工器電路

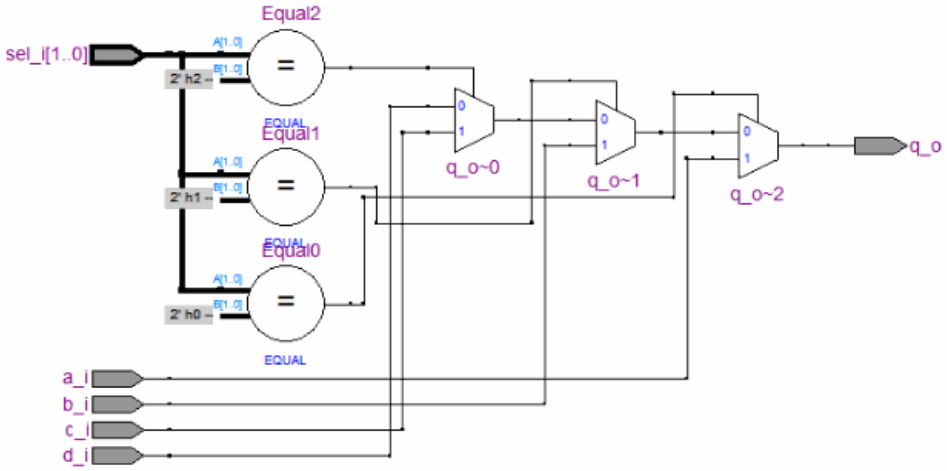
```
module mux_4to1(in1, in2, in3, in4, sel, out);
   input in1, in2, in3, in4;
   input [1:0]sel;
   output out;

   reg out;

   always@(sel or in1 or in2 or in3 or in4)begin
      case(sel)
         2'b00: out=in1;
         2'b01: out=in2;
         2'b10: out=in3;
         2'b11: out=in4;
         default:out=1'b0;
      endcase
   end

endmodule
```

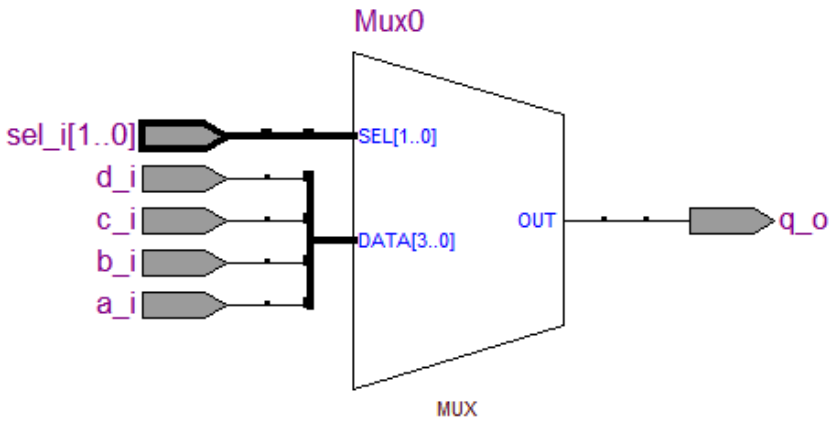
mux_4to1

條件敘述與多路徑分支 (2/5)

```
module mux_4to1(in1, in2, in3, in4, sel, out);
   input in1, in2, in3, in4;
   input [1:0]sel;
   output out;

   reg out;

   always@(sel or in1 or in2 or in3 or in4)begin
      if(sel==2'b00)out=in1;
      else if(sel==2'b01)out=in2;
      else if(sel==2'b10)out=in3;
      else out=in4;
   end

endmodule
```

```
module mux_4to1(in1, in2, in3, in4, sel, out);
   input in1, in2, in3, in4;
   input [1:0]sel;
   output out;

   reg out;

   always@(sel or in1 or in2 or in3 or in4)begin
      case(sel)
         2'b00: out=in1;
         2'b01: out=in2;
         2'b10: out=in3;
         2'b11: out=in4;
         default:out=1'b0;
      endcase
   end

endmodule
```





條件敘述與多路徑分支 (3/5)

```
module mux_4to1(in1, in2, in3, in4, sel, out);
   input in1, in2, in3, in4;
   input [1:0]sel;
   output out;

   reg out;

   always@(sel or in1 or in2 or in3 or in4)begin
      if(sel==2'b00)out=in1;
      else if(sel==2'b01)out=in2;
      else if(sel==2'b10)out=in3;
      else out=in4;
   end

endmodule
```

mux_4to1

if-else敘述記得要有個else

case敘述記得要有default

```
module mux_4to1(in1, in2, in3, in4, sel, out);
   input in1, in2, in3, in4;
   input [1:0]sel;
   output out;

   reg out;

   always@(sel or in1 or in2 or in3 or in4)begin
      case(sel)
         2'b00: out=in1;
         2'b01: out=in2;
         2'b10: out=in3;
         2'b11: out=in4;
         default:out=1'b0;
      endcase
   end

endmodule
```

mux_4to1

這兩種敘述都必須要做完整的定義，才能使合成的工作正常運作

因為如果定義不完整，合成軟體也許會合成出latch而不是多工器

條件敘述與多路徑分支 (4/5)

```
module example(in1, in2, sel, out1, out2);
   input in1, in2, sel;
   output out1, out2;

   reg out1, out2;

   always@(*)begin
      if(sel==1'b1)begin
         out1=in1&in2;
         //out2=??
      end
      else begin
         //out1=??
         out2=in1|in2;
      end
   end

endmodule
```

example

WARNING : Found 1-bit latch for signal <out2>

WARNING : Found 1-bit latch for signal <out1>

務必將每一項敘述補齊，例如補上 1'b0或1'bx (don't care)等

通常建議補上1'bx，因為可以讓合成軟體替你做最佳化，而減少電路面積

```
module example(in1, in2, sel, out1, out2);
   input in1, in2, sel;
   output out1, out2;

   reg out1, out2;

   always@(*)begin
      if(sel==1'b1)begin
         out1=in1&in2;
         out2=1'b0;
      end
      else begin
         out1=1'b0;
         out2=in1|in2;
      end
   end

endmodule
```

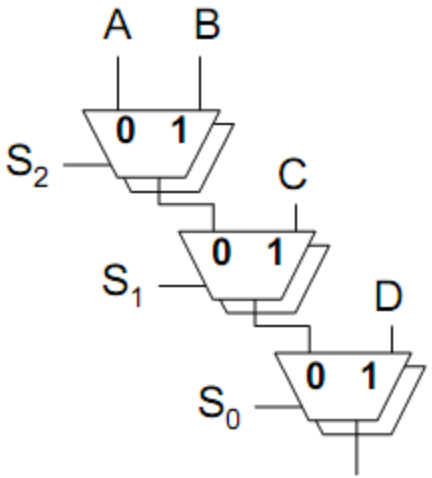
example

條件敘述與多路徑分支 (5/5)

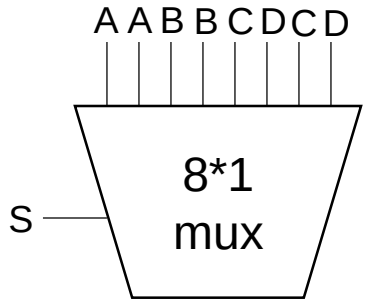
- 使用條件運算子(?:)與if-else、case語法的不同：
  - 無論是使用if-else或case的語法都會合成出多工器電路，但是如果你就是需要多工器這樣的結構，那最好就是使用條件運算子。
  - 因為如果使用if-else或case的敘述，雖然可攜性較佳，但有可能會多合成出一些多餘的邏輯閘。
  - 而使用條件運算子則是會犧牲掉可攜性，而且通常需要撰寫較長的程式碼。

Priority mux v.s. Parallel mux

- **Priority mux**
  - if (S0 == 1'b1)
    - Out=D
  - else if(S1== 1'b1)
    - Out = C;
  - else if(S2 == 1'b1)
    - Out = B;
  - else
    - Out = A;



- **Parallel mux**
  - Case({S0,S1,S2})
  - 3'b000=A;
  - 3'b001=A;
  - 3'b010=B;
  - 3'b011=B;
  - 3'b100=C
  - 3'b101=D;
  - 3'b110=C;
  - 3'b111=D;
  - endcase



文字巨集與參數

```
module alu_4bit(sel, in1, in2, result);
   input [2:0]sel;
   input [3:0]in1, in2;
   output [3:0]result;

   `define ADD 3'b000
   `define SUB 3'b001
   `define AND 3'b010
   `define OR  3'b011
   `define XOR 3'b100
   `define SHR 3'b101
   `define SHL 3'b110
   `define CMP 3'b111

   reg [3:0]result;

   always@(*)begin
      case(sel)
         `ADD: result=in1+in2;
         `SUB: result=in1-in2;
         `AND: result=in1&in2;
         `OR : result=in1|in2;
         `XOR: result=in1^in2;
         `SHR: result=in1>>in2;
         `SHL: result=in1<<in2;
         `CMP: result=in1>in2;
         default: result=4'b0;
      endcase
   end

endmodule
```

alu_4bit

文字巨集(`defne)可以用來定義文字巨集，如同C程式語言中的#define

參數(parameter)可以用來定義固定的常數

```
module alu_4bit(sel, in1, in2, result);
   input [2:0]sel;
   input [3:0]in1, in2;
   output [3:0]result;

   parameter ADD = 3'b000,
             SUB = 3'b001,
             AND = 3'b010,
             OR  = 3'b011,
             XOR = 3'b100,
             SHR = 3'b101,
             SHL = 3'b110,
             CMP = 3'b111;

   reg [3:0]result;

   always@(*)begin
      case(sel)
         ADD: result=in1+in2;
         SUB: result=in1-in2;
         AND: result=in1&in2;
         OR : result=in1|in2;
         XOR: result=in1^in2;
         SHR: result=in1>>in2;
         SHL: result=in1<<in2;
         CMP: result=in1>in2;
         default: result=4'b0;
      endcase
   end

endmodule
```

alu_4bit

資料處理層次與行為層次之比較

行為層次

```
module mux_4to1(in1, in2, in3, in4, sel, out);
   input in1, in2, in3, in4;
   input [1:0]sel;
   output out;

   reg out;

   always@(sel or in1 or in2 or in3 or in4)begin
      case(sel)
         2'b00: out=in1;
         2'b01: out=in2;
         2'b10: out=in3;
         2'b11: out=in4;
         default:out=1'b0;
      endcase
   end
endmodule
```

等號左側為暫存器(reg)

mux_4to1

資料處理層次

```
module mux_4to1(in1, in2, in3, in4, sel, out);
   input in1, in2, in3, in4;
   input [1:0]sel;
   output out;

   assign out = sel[1] ? (sel[0] ? in4 : in3) : (sel[0] ? in2 : in1);

endmodule
```

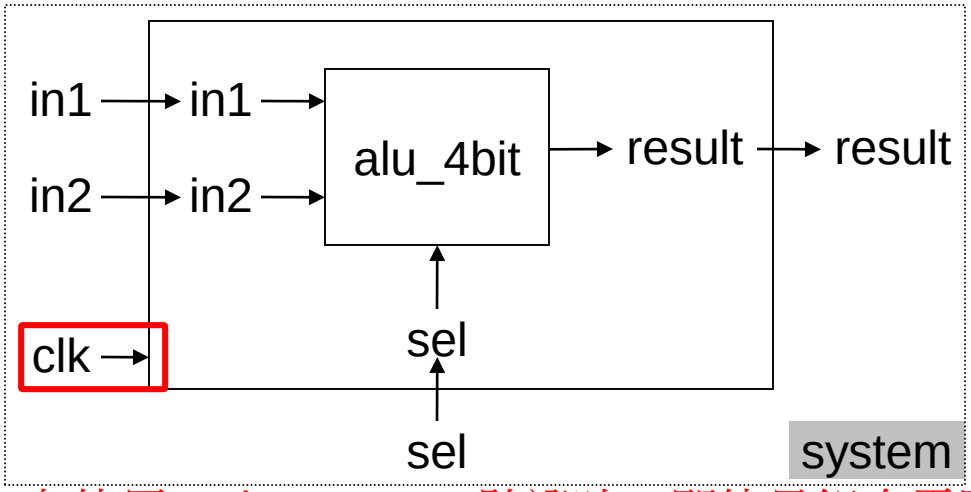
等號左側為接線(wire)

mux_4to1

數位電路設計流程

- 1) 設計 (lab4 - Xilinx ISE)
- 2) 編譯 (lab4 - Xilinx ISE)
- 3) (合成前)模擬 (lab4 - ISim)
- 4) (合成後)驗證 (lab5 - VeriComm)
- 5) (合成後)驗證 (lab6 - VeriInstrument)

VeriInstrument (1/13)



在使用VeriInstrument驗證時，即使是組合電路也需要加上clock，所以我們使用一個最上層模組(system)將原本的組合電路設計給包覆起來

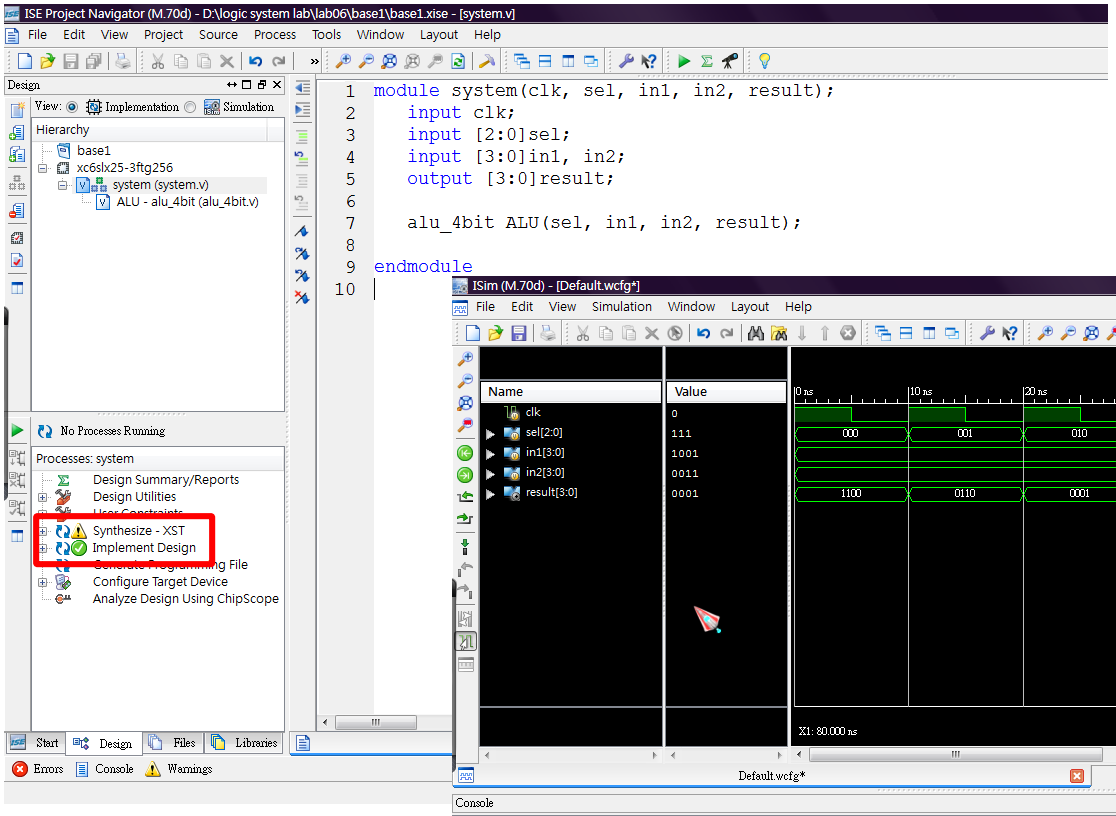
```
module system(clk, sel, in1, in2, result);
   input clk;
   input [2:0]sel;
   input [3:0]in1, in2;
   output [3:0]result;

   alu_4bit ALU(sel, in1, in2, result);

endmodule
```

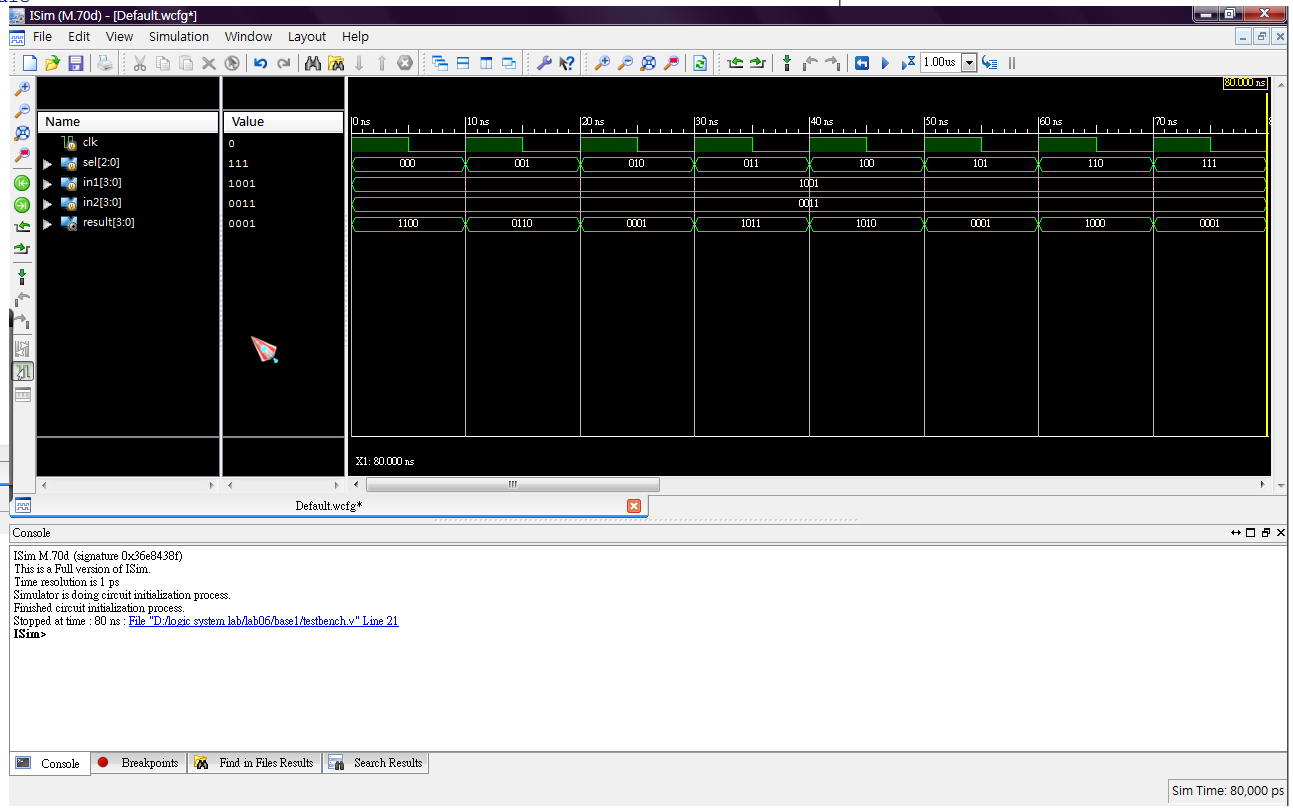
system

```
module alu_4bit(sel, in1, in2, result);
   input [2:0]sel;
   input [3:0]in1, in2;
   output [3:0]result;

   parameter ADD = 3'b000,
             SUB = 3'b001,
             AND = 3'b010,
             OR  = 3'b011,
             XOR = 3'b100,
             SHR = 3'b101,
             SHL = 3'b110,
             CMP = 3'b111;

   reg [3:0]result;

   always@(*)begin
      case(sel)
         ADD: result=in1+in2;
         SUB: result=in1-in2;
         AND: result=in1&in2;
         OR : result=in1|in2;
         XOR: result=in1^in2;
         SHR: result=in1>>in2;
         SHL: result=in1<<in2;
         CMP: result=in1>in2;
         default: result=4'b0;
      endcase
   end

endmodule
```

alu_4bit

VeriInstrument (2/13)





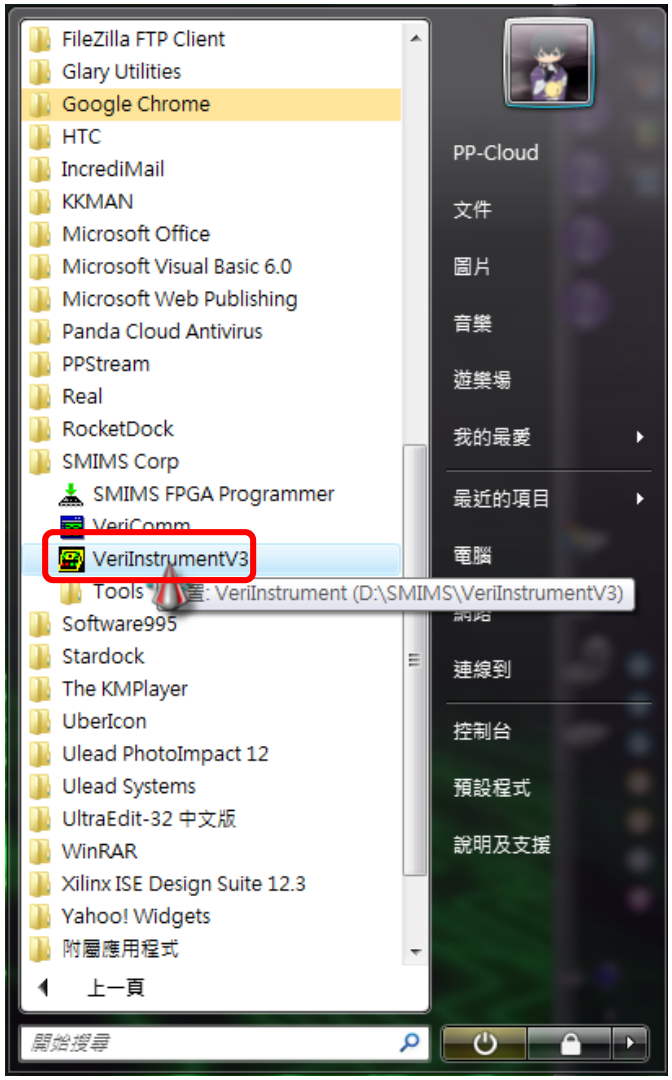
接下來的動作是假設您已經完成了設計、編譯與模擬的步驟，以及已經將FPGA板接上電腦並且已經打開開關

編譯時會顯示一個warning，是關於clk浮接的問題，請忽略之

VeriInstrument (3/13)

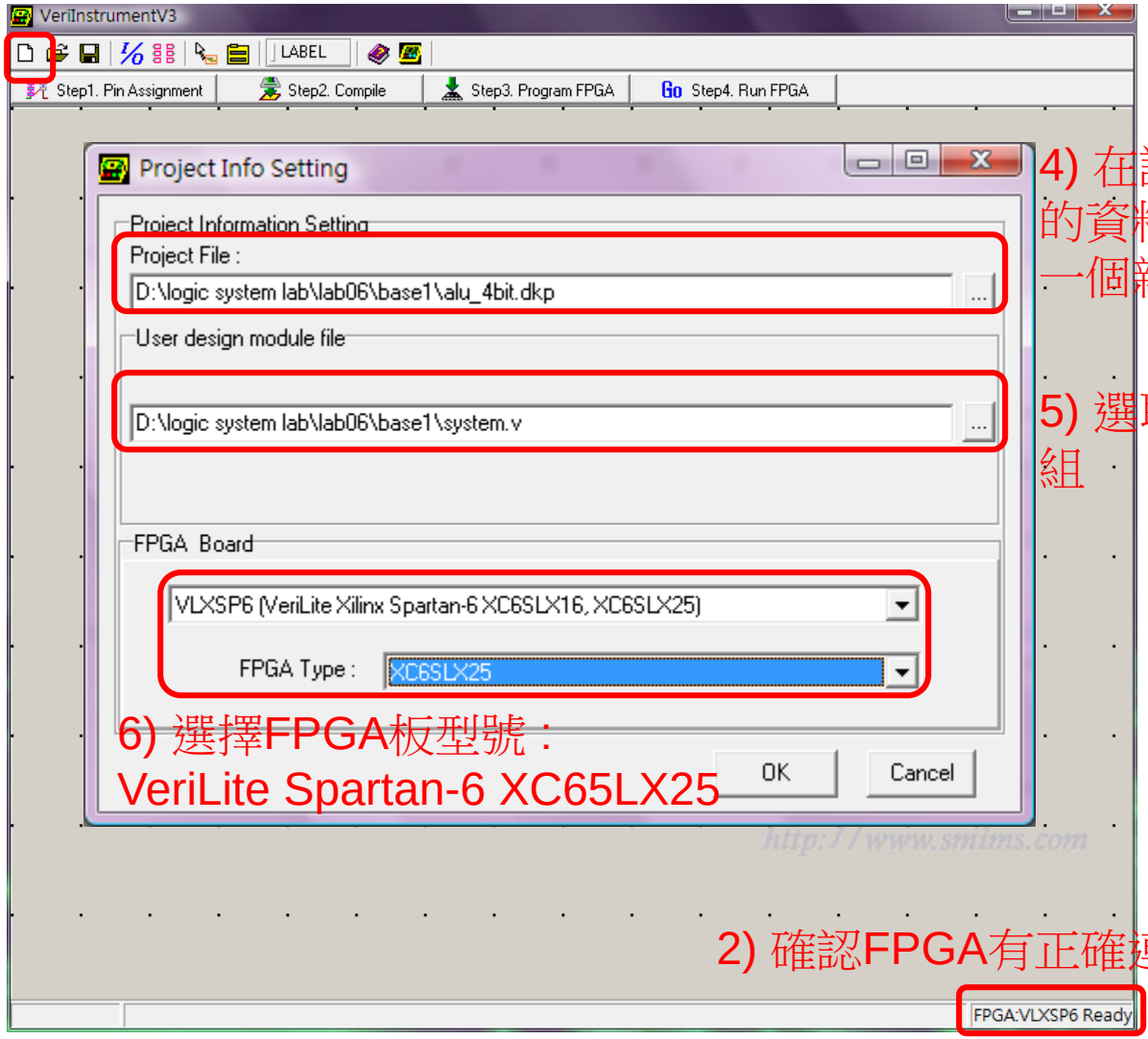
- SMIMS VeriInstrument為初學者提供一個有趣的數位邏輯設計界面。
- 經由拖放與執行虛擬元件，我們可以結合出不同的硬體架構。

1) 開始 > SMIMS Corp > VeriInstrumentV3



VeriInstrument (4/13)

3) 開新專案



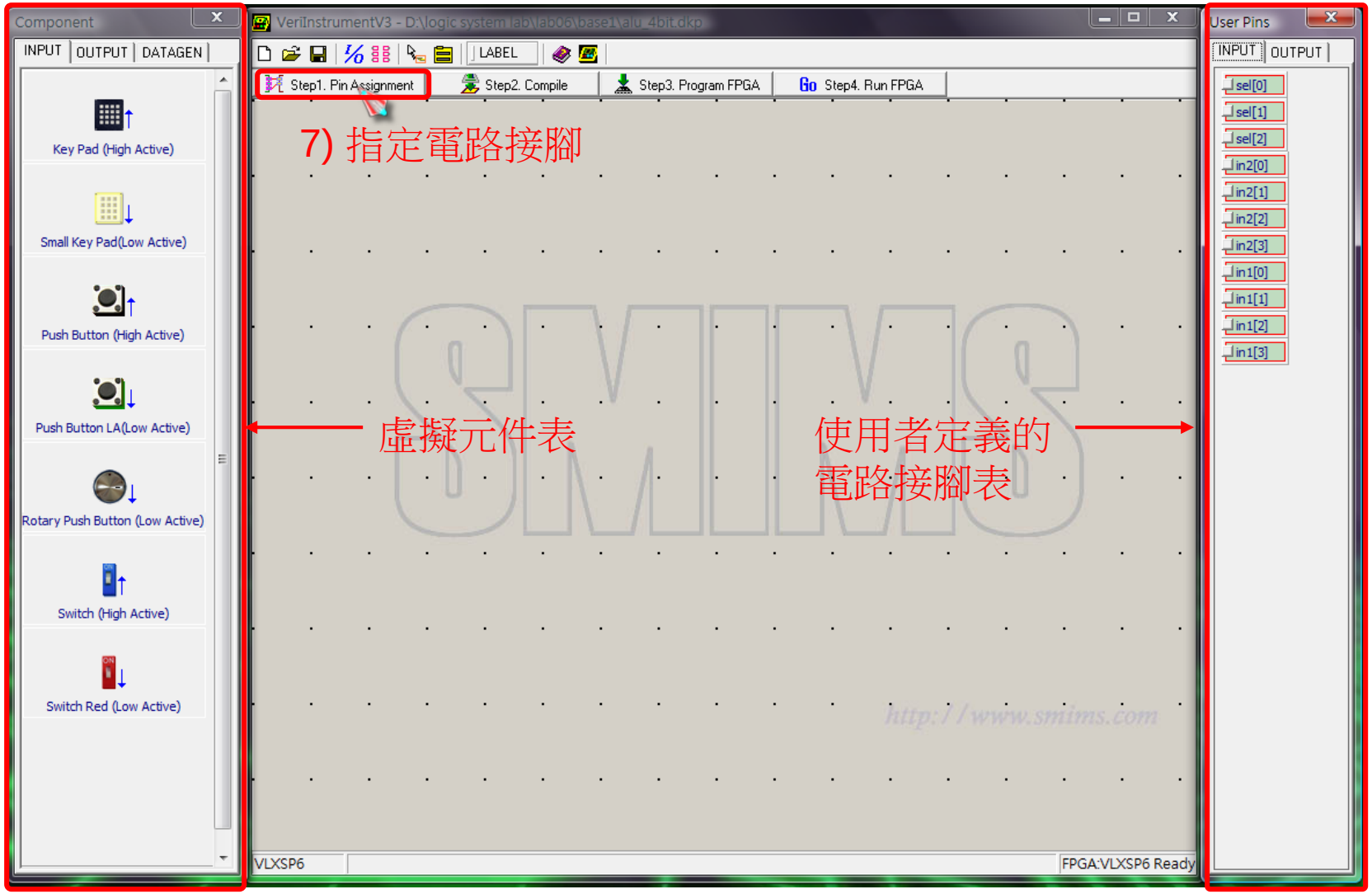
4) 在該次實驗課的資料夾下建立一個新專案

5) 選取最上層模組

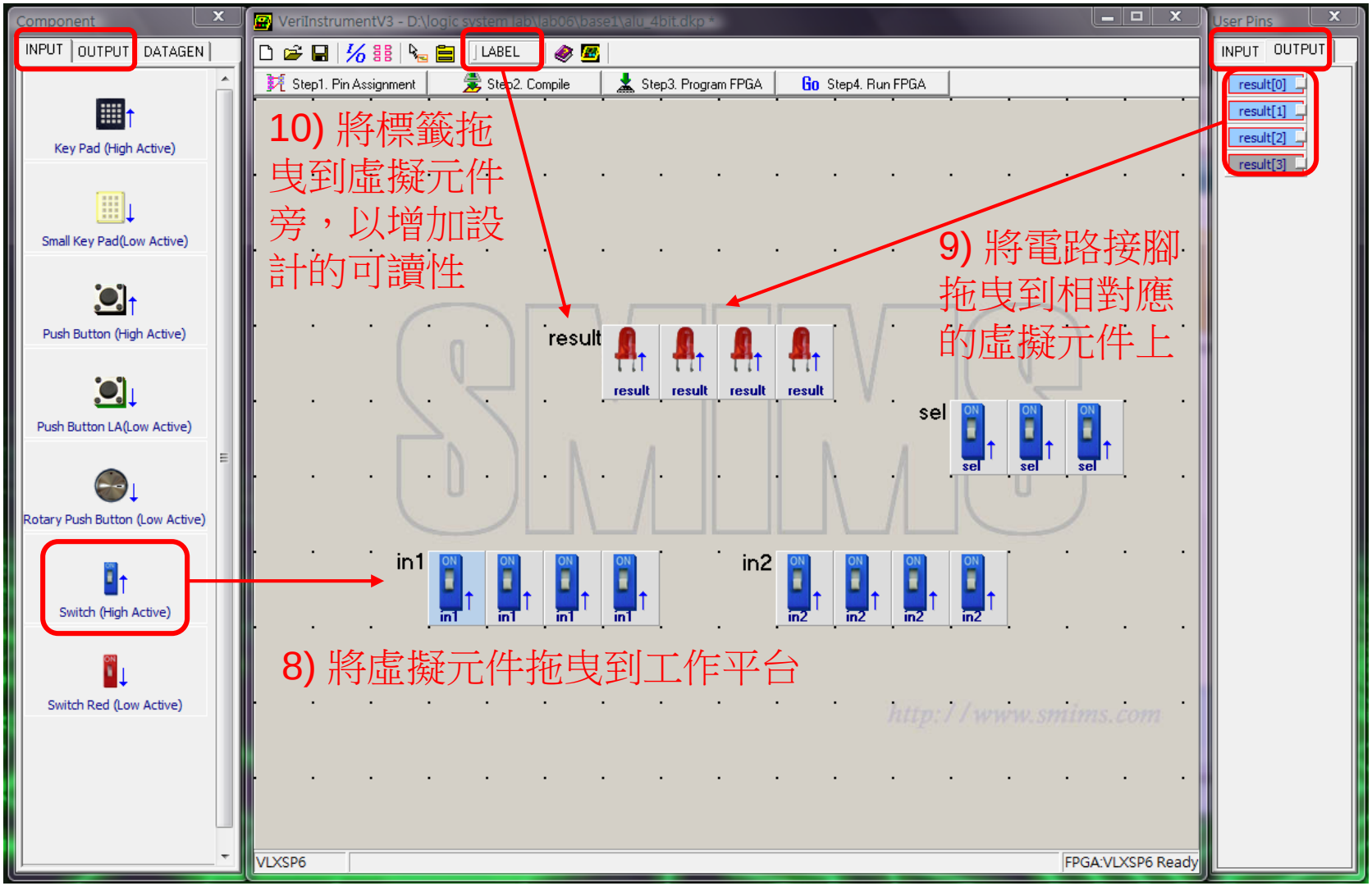
6) 選擇FPGA板型號：
VeriLite Spartan-6 XC65LX25

2) 確認FPGA有正確連接上電腦

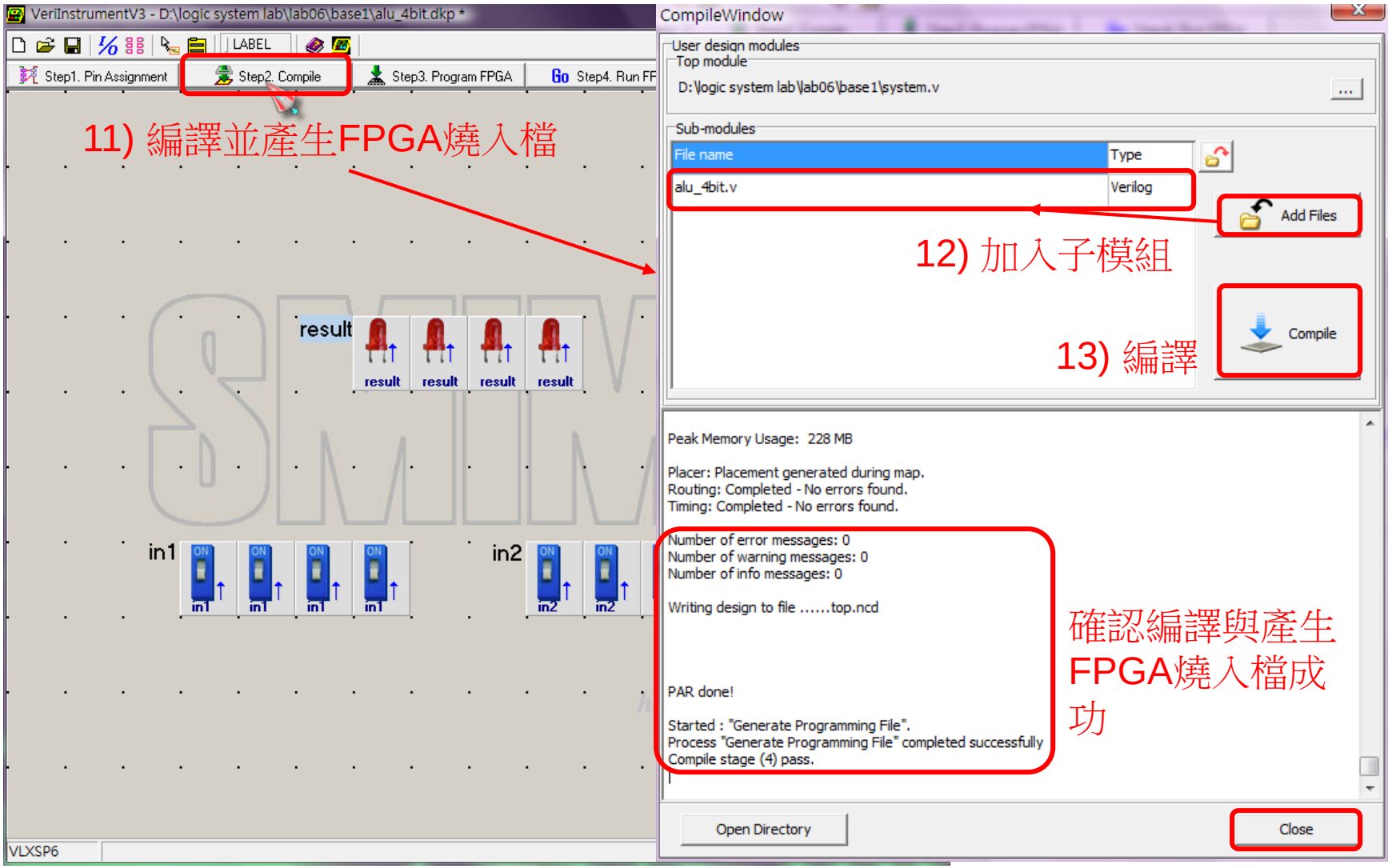
VeriInstrument (5/13)



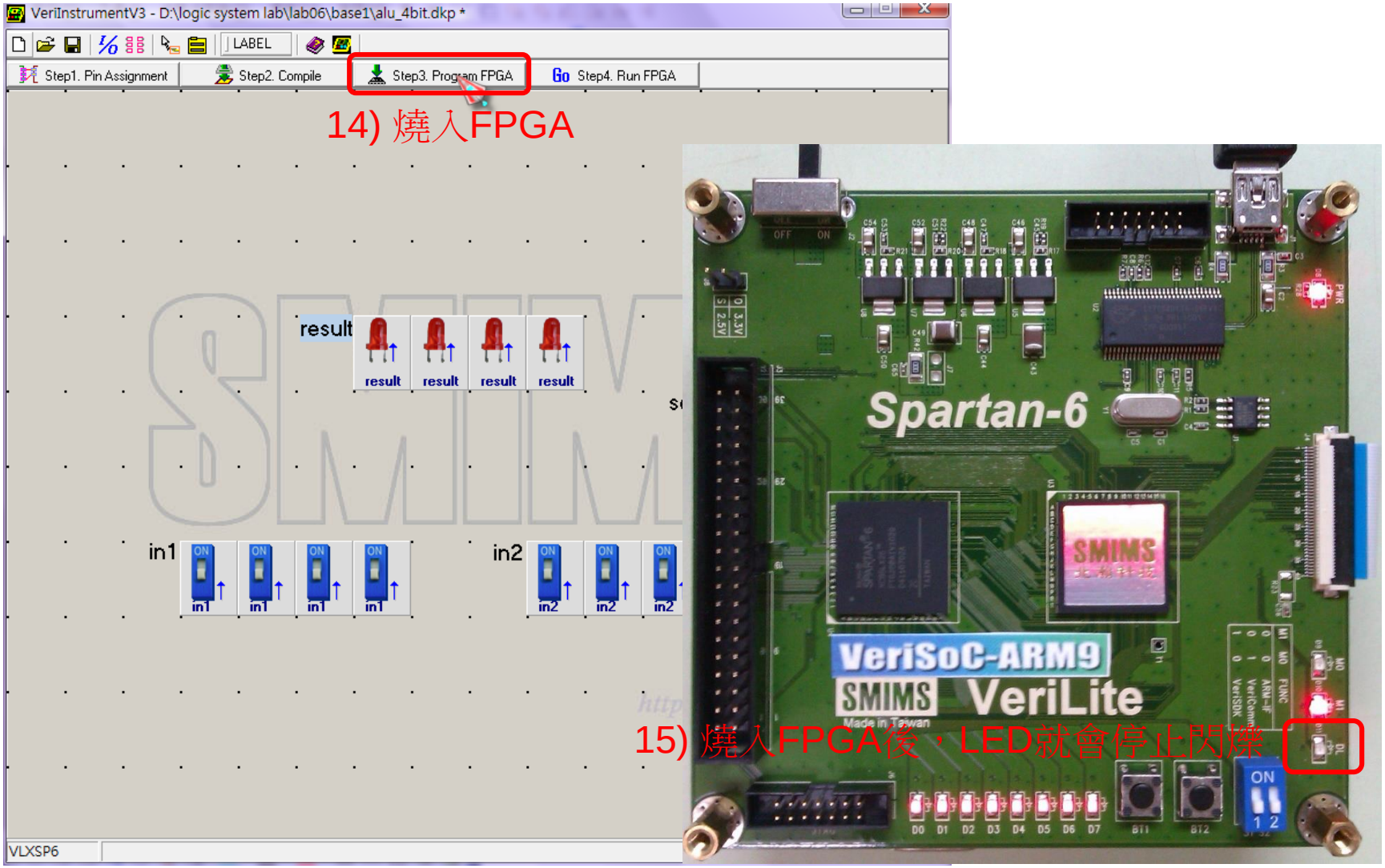
VeriInstrument (6/13)



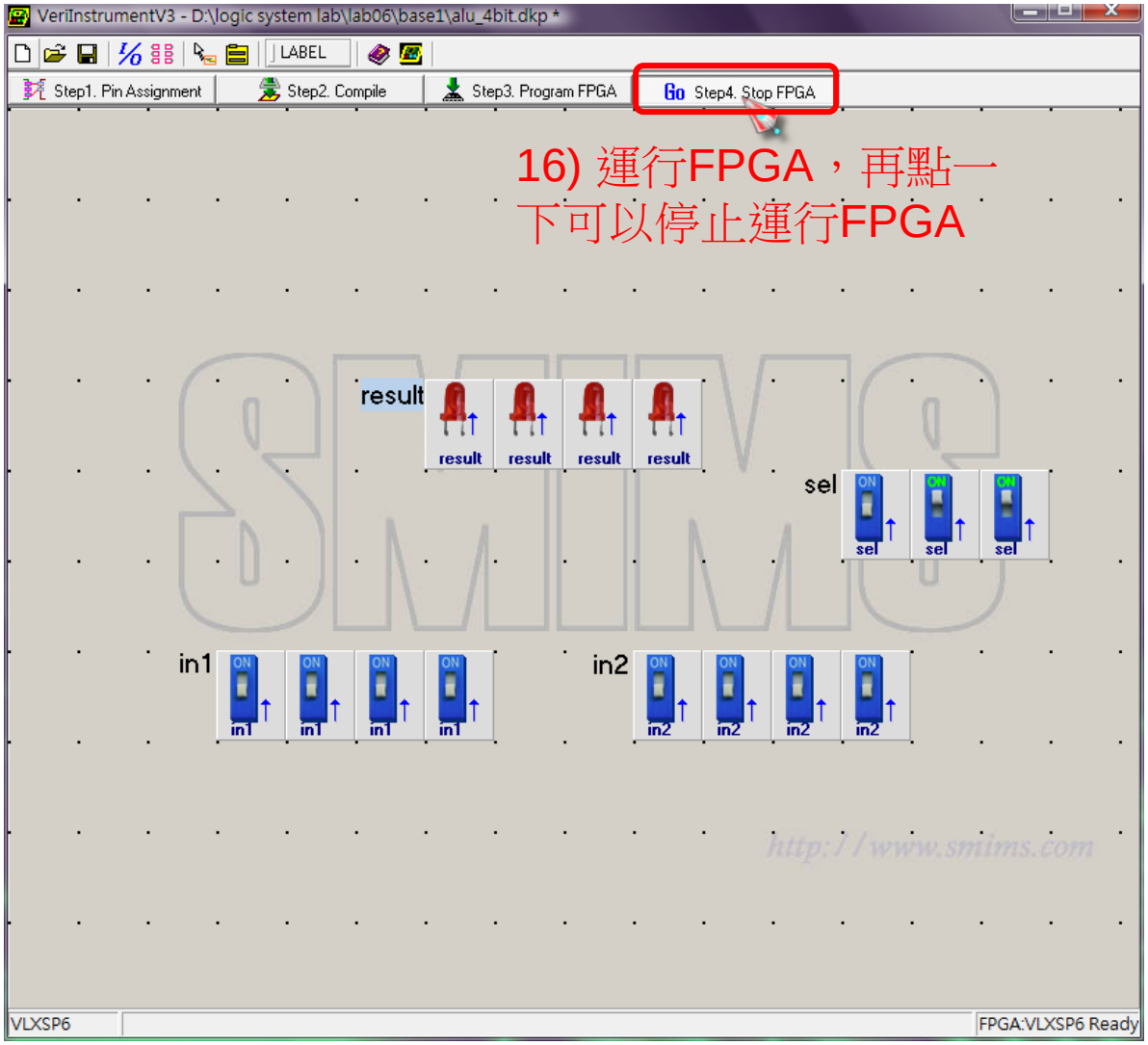
VeriInstrument (7/13)



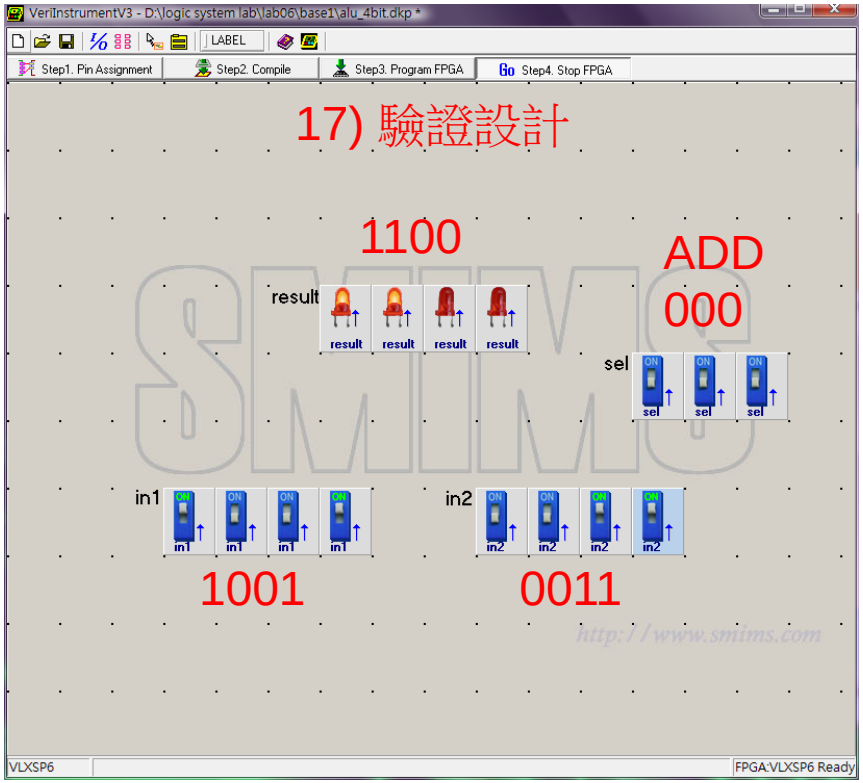
VeriInstrument (8/13)

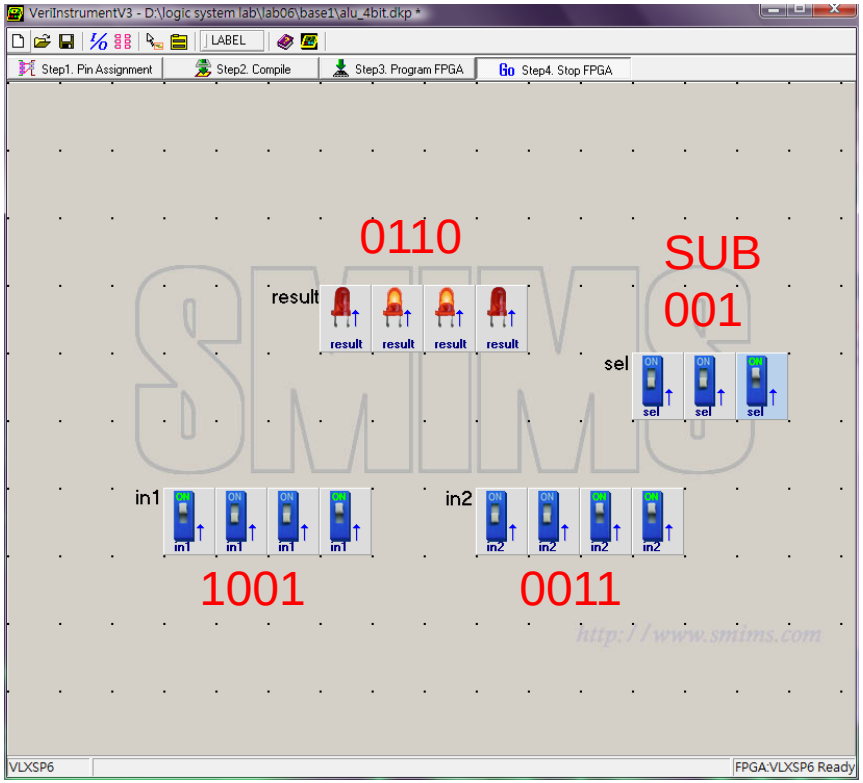


VeriInstrument (9/13)

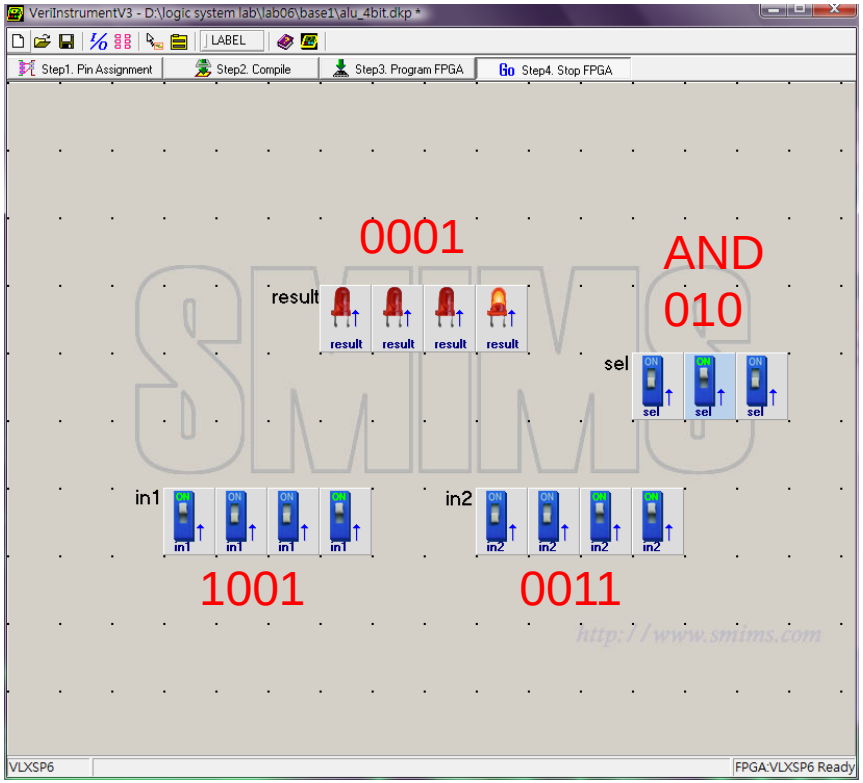


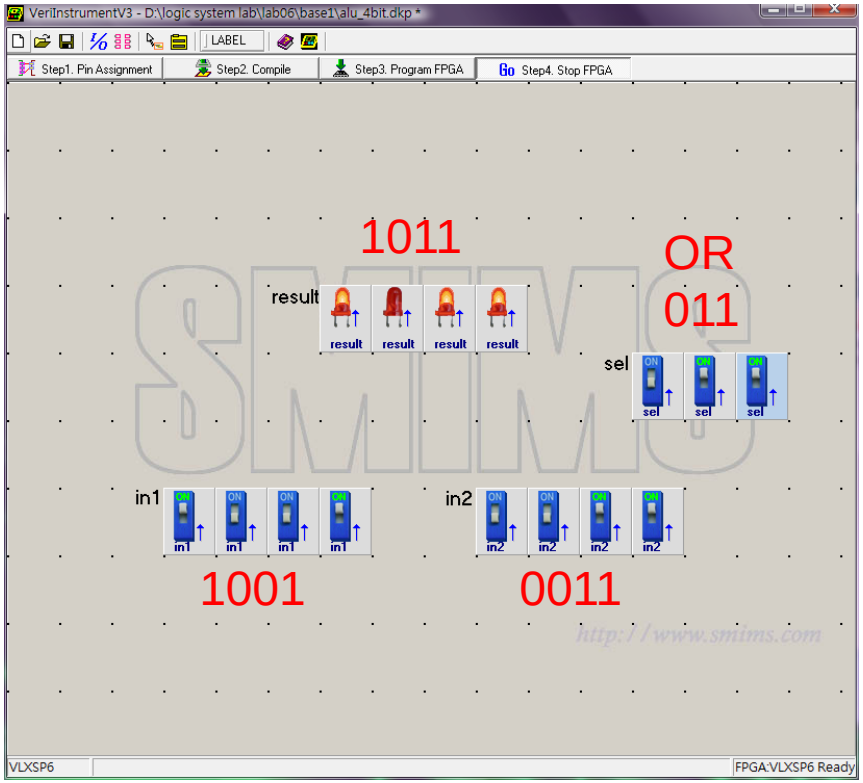
VeriInstrument (10/13)



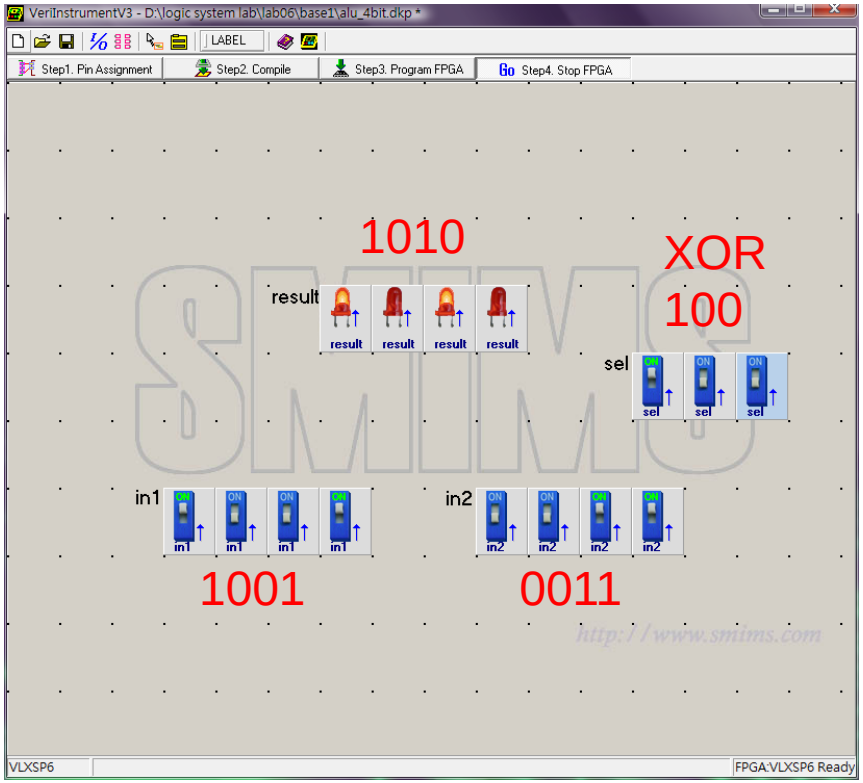


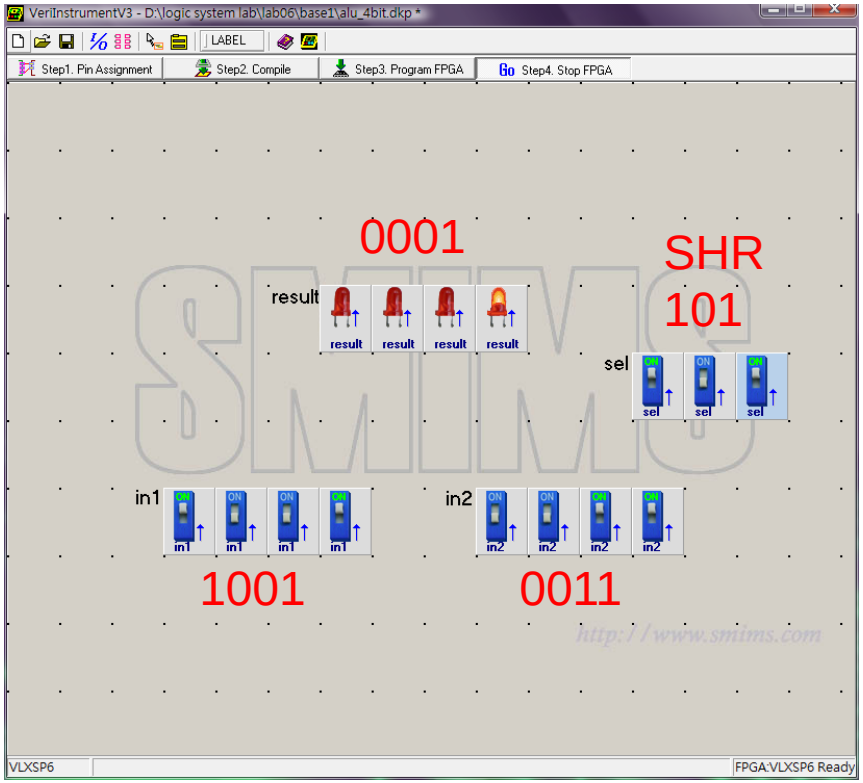
VeriInstrument (11/13)



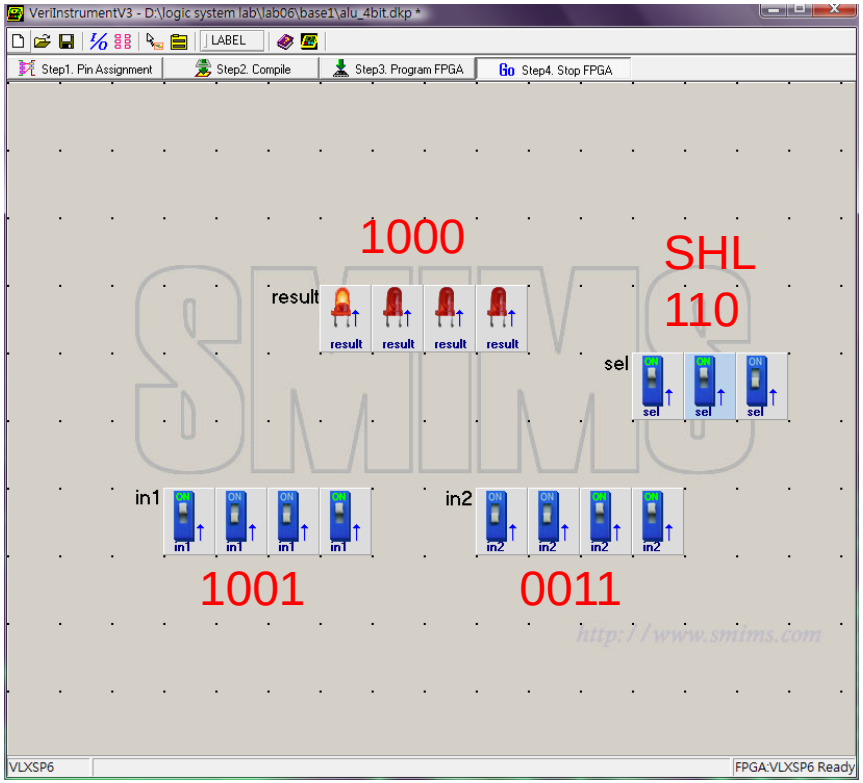


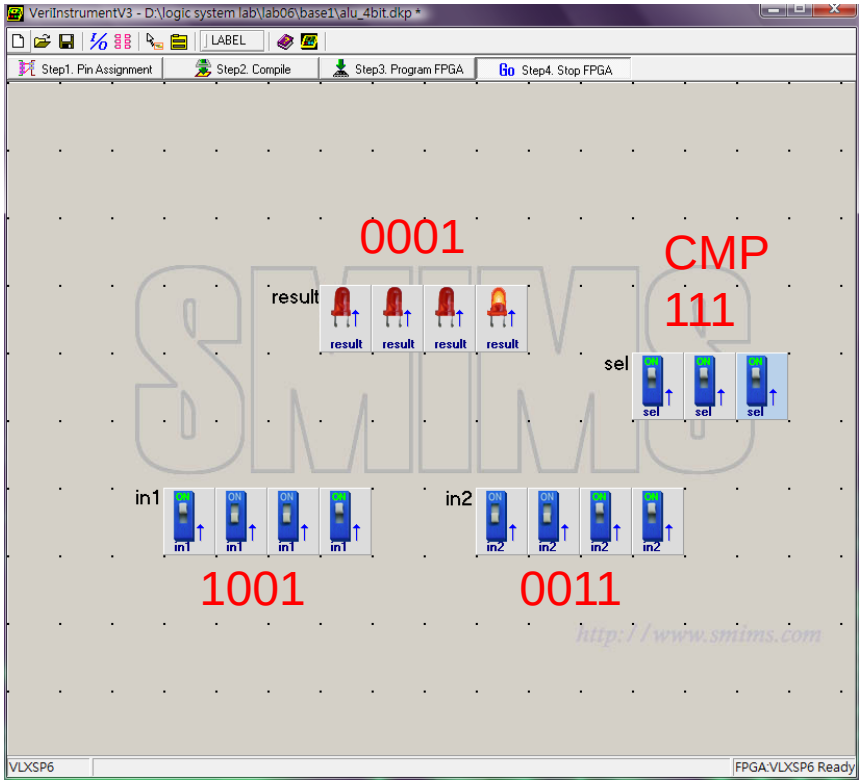
VeriInstrument (12/13)





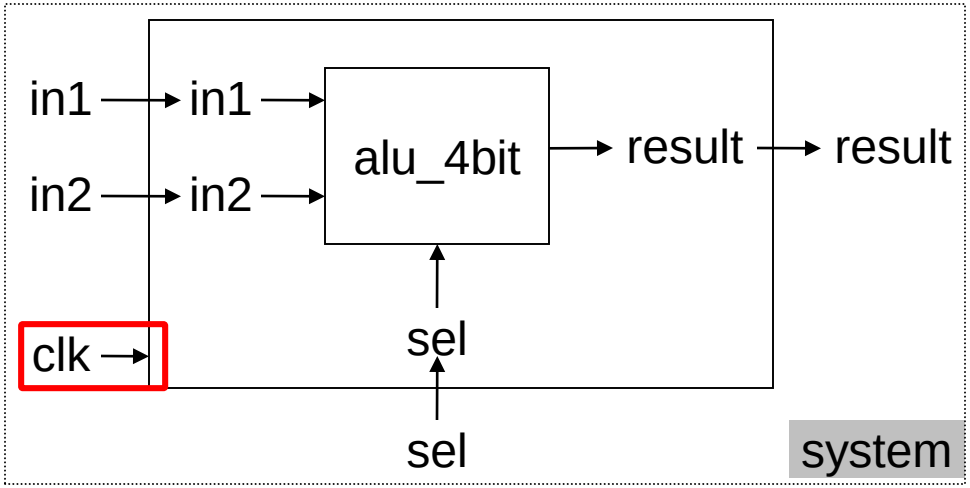
VeriInstrument (13/13)





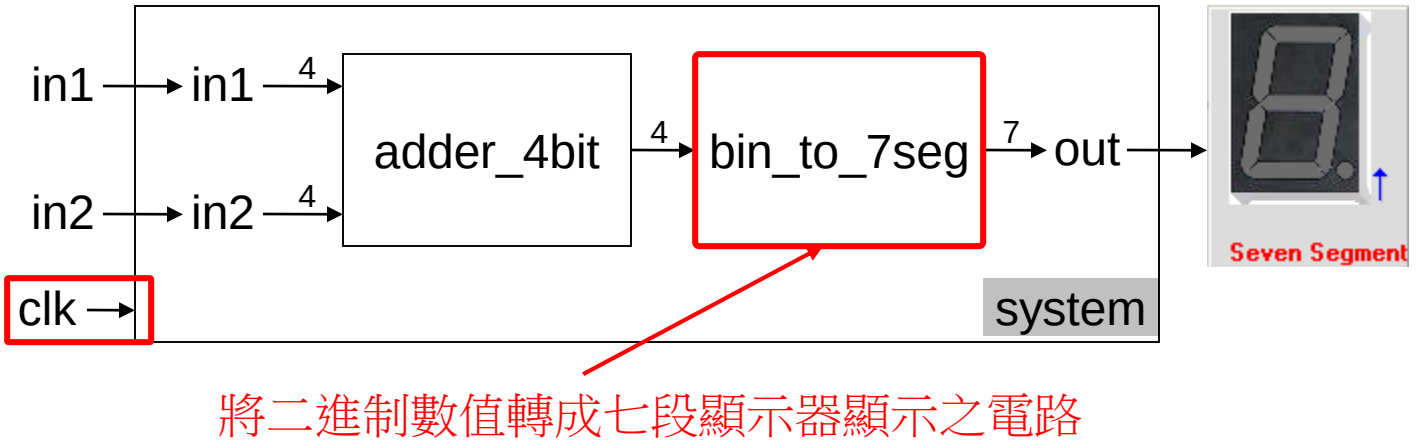
基礎題(一)
4bit簡易算數邏輯單元

- 請參考alu_4bit.v的範例原始碼與操作流程。
  - 設計時，請寫出4bit簡易算數邏輯單元模組，並且確認編譯後沒有error產生。
  - 模擬時，請自行撰寫testbench.v，並觀察波形結果或主控台的螢幕顯示結果是否如期望一般。
  - 驗證時，請觀察虛擬儀器之運作是否如期望一般。



基礎題 (二)
4bit加法器與七段顯示器 (1/3)

- 請寫出以下的電路。
  - 設計時，請寫出以下的電路模組，並且確認編譯後沒有error產生。
  - (模擬時，請自行撰寫testbench.v，並觀察波形結果或主控台的螢幕顯示結果是否如期望一般。)
  - 驗證時，請觀察虛擬儀器之運作是否如期望一般。



基礎題 (二)

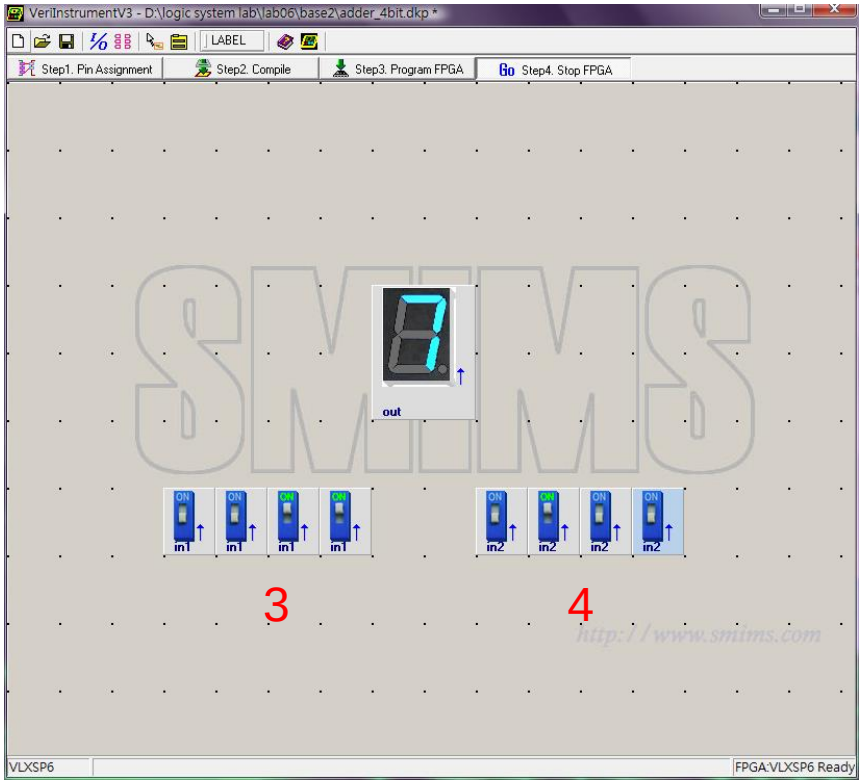
4bit加法器與七段顯示器 (2/3)

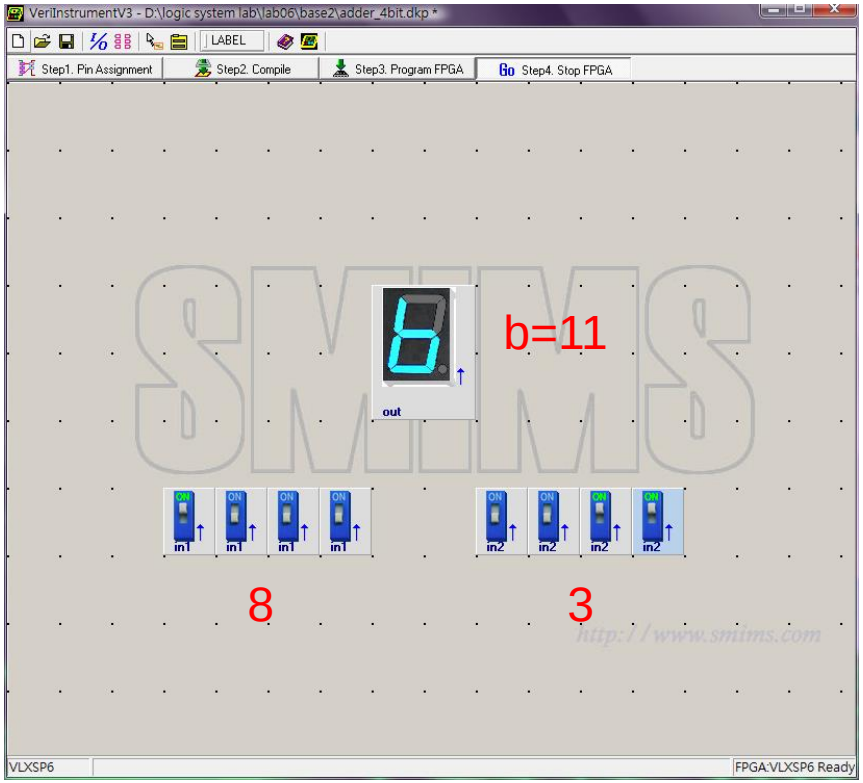

```
module bin_to_7seg(bin, out);
   input [3:0]bin;
   output [6:0]out;

   reg [6:0]out;

   always@(bin)begin
      case(bin)
         4'b0000: out=7'b0111111;
         4'b0001: out=7'b0000110;
         4'b0010: out=7'b1011011;
         4'b0011: out=7'b1001111;
         4'b0100: out=7'b1100110;
         4'b0101: out=7'b1101101;
         4'b0110: out=7'b1111101;
         4'b0111: out=7'b0000111;
         4'b1000: out=7'b1111111;
         4'b1001: out=7'b1101111;
         4'b1010: out=7'b1110111;
         4'b1011: out=7'b1111100;
         4'b1100: out=7'b0111001;
         4'b1101: out=7'b1011110;
         4'b1110: out=7'b1111001;
         4'b1111: out=7'b1110001;
         default: out=7'b0000000;
      endcase
   end

endmodule
```

bin_to_7seg

基礎題 (二)

4bit加法器與七段顯示器 (3/3)





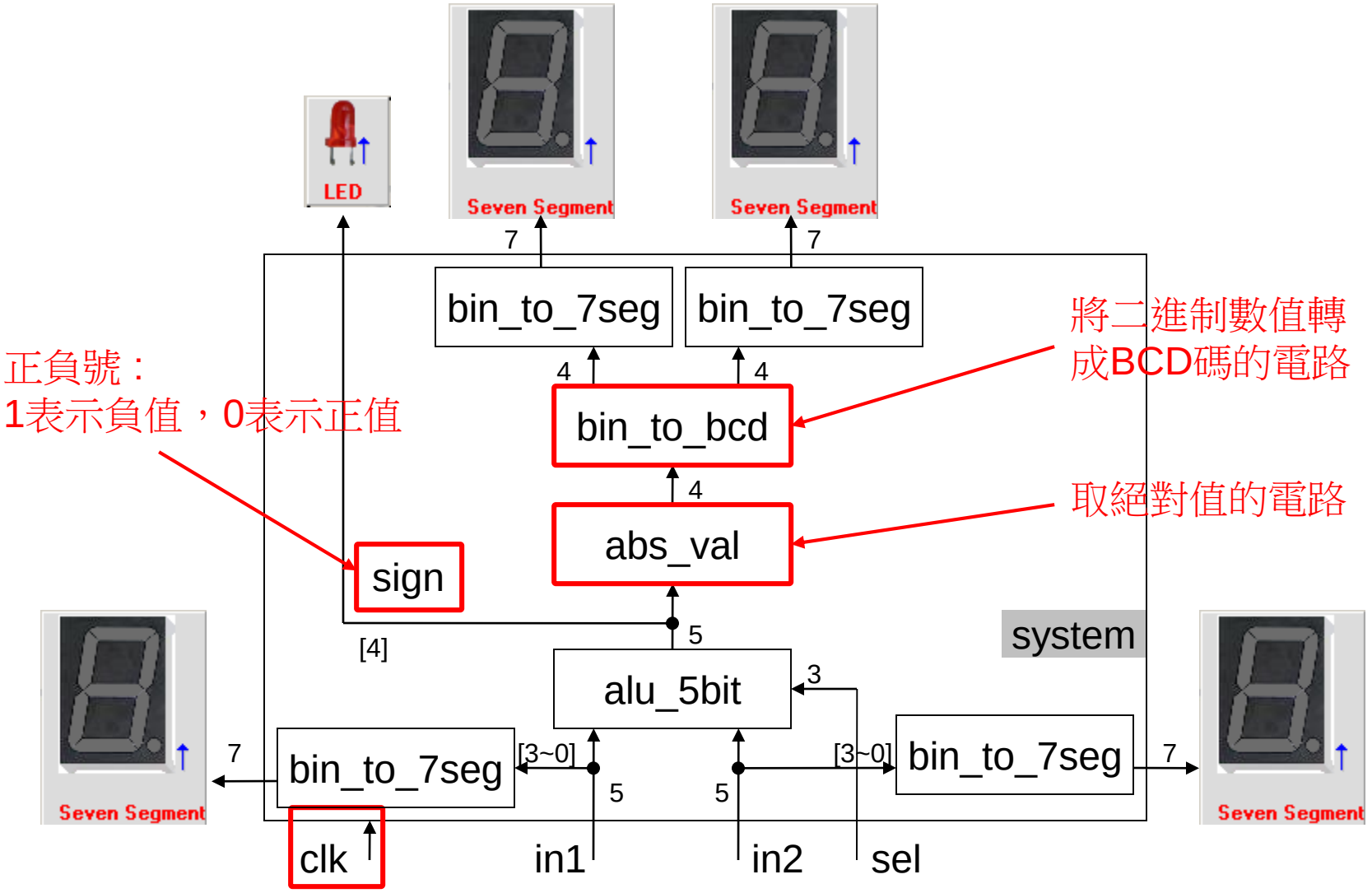
挑戰題

5bit簡易算數邏輯單元與七段顯示器 (1/5)

- 請寫出下一頁的電路。
  - 設計時，請寫出下一頁的電路模組，並且確認編譯後沒有error產生。
  - (模擬時，請自行撰寫testbench.v，並觀察波形結果或主控台的螢幕顯示結果是否如期望一般。)
  - 驗證時，請觀察虛擬儀器之運作是否如期望一般。

挑戰題

5bit簡易算數邏輯單元與七段顯示器 (2/5)



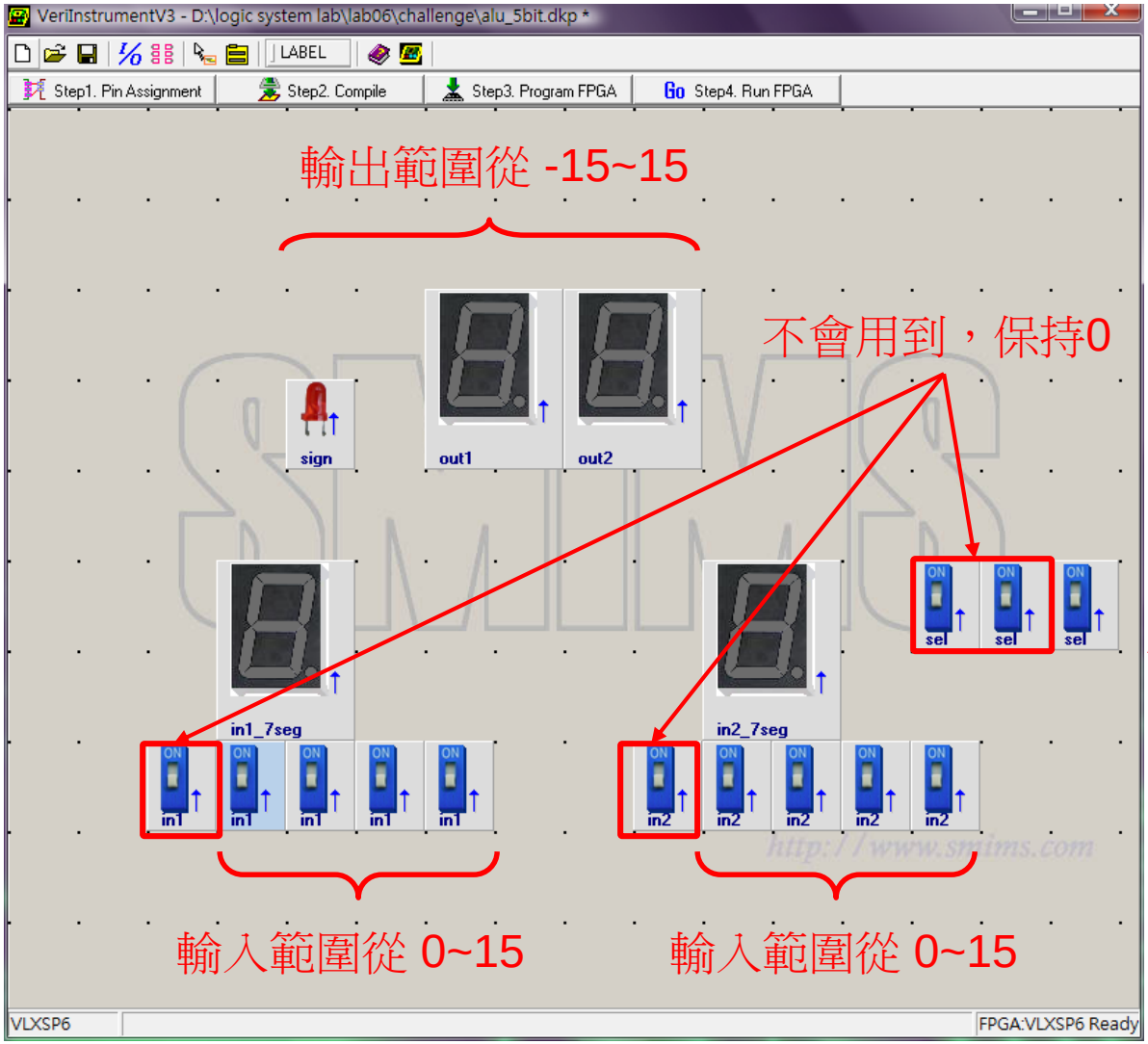
挑戰題

5bit簡易算數邏輯單元與七段顯示器 (3/5)

- 有號數二進位數值的解讀：
  - 若MSB為1則為負數，MSB為0為正數。
  - ex : a=5'b00110 -> a=+6。
  - ex : b=5'b10011 -> b= -(5'b01100+1) -> b=-13。
- 有號數二進位正負值互換(二補數)：
  - 將每一個bit做01互換後，再加1。
  - ex : b=5'b10011 -> b= -(5'b01100+1) -> b=-13。
- abs_val模組：
  - 輸入：5bits的有號數二進位數值。
  - 輸出：4bits的無號數二進位數值。
  - 功能：將輸入取絕對值後再輸出。
- bin_to_bcd模組：
  - 輸入：4bits的無號數二進位數值。
  - 輸出：兩個4bits的無號數二進位數值，分別代表十位數與個位數。
  - 功能：將二進制數值轉成二進制編碼的十進制數值後再輸出。

挑戰題

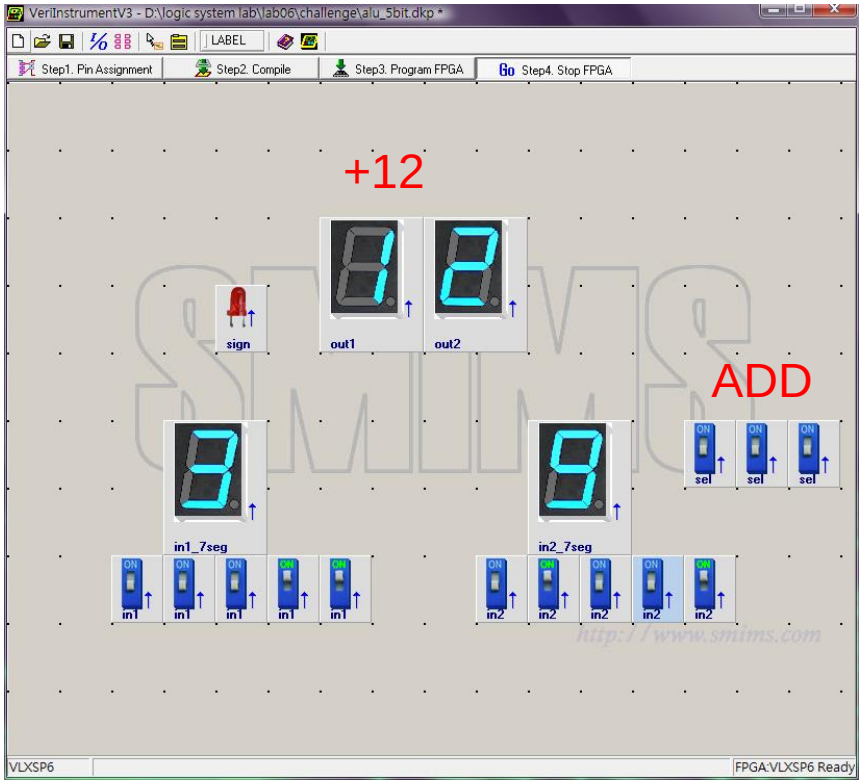
5bit簡易算數邏輯單元與七段顯示器 (4/5)

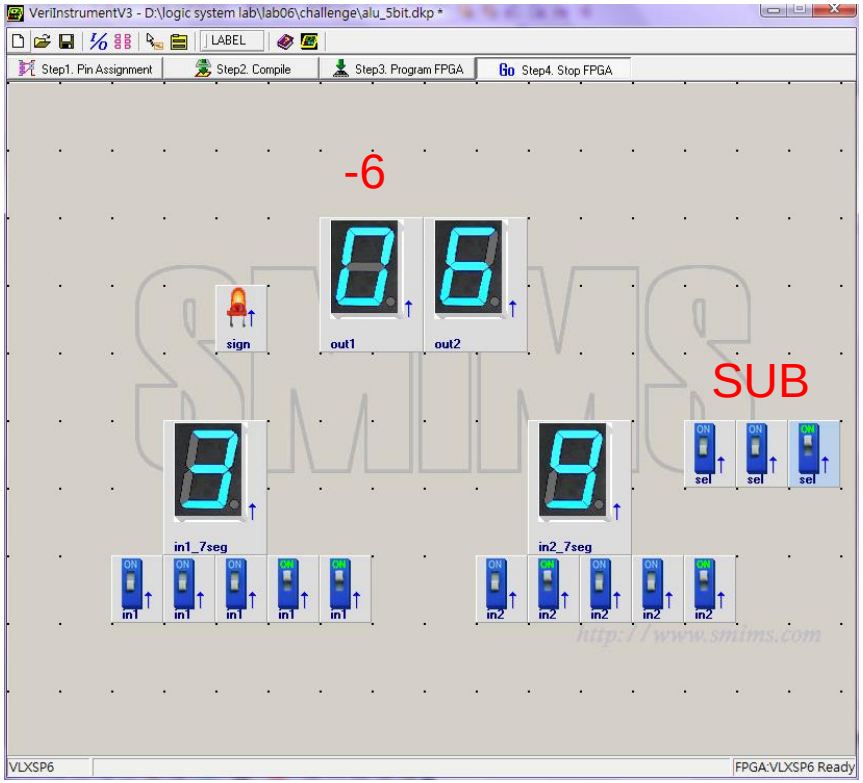


只會使用到ADD與SUB的功能

挑戰題

5bit簡易算數邏輯單元與七段顯示器 (5/5)





實驗結報繳交

- 基礎題 (一)
  - 請附上原始碼、模擬結果擷圖、驗證結果擷圖與解釋。
- 基礎題 (二)
  - 請附上原始碼、(模擬結果擷圖)、驗證結果擷圖與解釋。
- 挑戰題
  - 請附上原始碼、(模擬結果擷圖)、驗證結果擷圖與解釋。
- 各自之心得報告