



邏輯系統實習

實驗五之二

VeriLite FPGA之介紹與操作 + Verilog語法介紹(二)：資料流層次

國立成功大學 電機系

2016

大綱

- **Verilog**中的四種描述層次
- 持續指定
- 運算子的種類與符號
- 運算子的優先順序
- 邏輯最佳化
- 邏輯閘層次與資料處理層次的比較
 - 四對一多工器
 - 4bit加法器
- 數位邏輯科技
- **FPGA**介紹
- 數位電路設計流程
- **VeriLite FPGA**開發板
- **SMIMS** 埠指定檔自動產生程式
 - 產生.ucf檔
- **Xilinx ISE**
 - 產生.bit檔
- **SMIMS VeriComm**
 - **FPGA**燒入
 - **FPGA**驗證
- 基礎題 (一)
 - 4bit加法器
- 基礎題 (二)
 - 4bit簡易算數邏輯單元
- 挑戰題
 - 4bit前瞻式進位加法器
- 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。

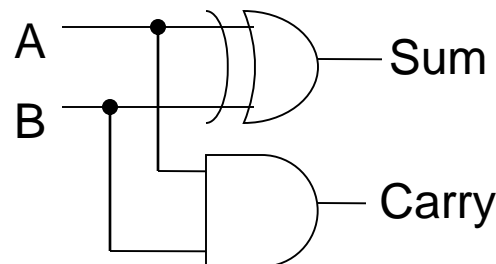
越上層設計層次越高

持續指定

- 持續指定是資料處理層次中，對接線指定其邏輯值的基本用法。
 - 在指定敘述(=)的左邊必須是接線(wire)，不能是暫存器(reg)；而在敘述右邊的運算元可以是接線(wire)或是暫存器(reg)。
 - 持續指定永遠處於活動狀態，當敘述符號(=)右邊的運算元的值發生變化時，其左邊指定的接線的值也會相對應的改變。
- 持續指定包含正規的持續指定與隱含式的持續指定，兩者有相同的效果。
 - 正規的持續指定：如右上圖。
 - 隱含式的持續指定：如右下圖。

```
module halfadder(in1, in2, sum, carry);  
    input in1, in2;  
    output sum, carry;  
  
    wire sum, carry;  
  
    assign sum = in1^in2;  
    assign carry = in1&in2;  
  
endmodule
```

halfadder



```
module halfadder(in1, in2, sum, carry);  
    input in1, in2;  
    output sum, carry;  
  
    wire sum = in1^in2;  
    wire carry = in1&in2;  
  
endmodule
```

halfadder

運算子的種類與符號 (1/4)

種類	符號	運算功能	需要運算元數目
算數運算	*	乘法	2
	/	除法	2
	+	加法	2
	-	減法	2
	%	取餘數	2
邏輯運算	!	邏輯的NOT	1
	&&	邏輯的AND	2
		邏輯的OR	2
比較運算	>	大於	2
	<	小於	2
	>=	大於或等於	2
	<=	小於或等於	2
相等運算	==	等於	2
	!=	不等於	2
條件運算	?:	做條件運算	3

種類	符號	運算功能	需要運算元數目
位元運算	~	位元的NOT	1
	&	位元的AND	2
		位元的OR	2
	^	位元的XOR	2
	~^	位元的XNOR	2
化簡運算	&	化簡的位元AND	1
	~&	化簡的位元NAND	1
		化簡的位元OR	1
	~	化簡的位元NOR	1
	^	化簡的位元XOR	1
	~^	化簡的位元XNOR	1
移位運算	>>	向右移位	2
	<<	向左移位	2
連結運算	{}	連結	任意數目
	{}}	複製	任意數目

運算子的種類與符號 (2/4)

■ 算數運算子

- ex : `a = 4'b0111; b = 4'b0100;`
- `a+b`的結果會是`4'b1011`。

■ 邏輯運算子

- ex : `a = 3; b = 0;`
- `(a==3)&&(b==0)`的結果會是`true`。
- `a&&b`的結果會是`false`，相當於`1&&0`。

■ 比較運算子

- ex : `a = 4; b = 3;`
- `a>b`的結果會是`true`。

■ 相等運算子

- ex : `a = 4'b0011; b = 4'b0100;`
- `a==b`的結果會是`false`。

運算子的種類與符號 (3/4)

■ 位元運算子

- ex : $a = 4'b1010;$ $b = 4'b1101;$
- $a \& b$ 的結果會是 $4'b1000$ 。
- $a \wedge b$ 的結果會是 $4'b0111$ 。

■ 化簡運算子

- ex : $x = 4'b1010;$
- $\&x$ 的結果會是 $1'b0$ ，相當於 $1 \& 0 \& 1 \& 0$ 。
- $\wedge x$ 的結果會是 $1'b0$ ，相當於 $1 \wedge 0 \wedge 1 \wedge 0$ 。

■ 移位運算子

- ex : $x = 4'b1100;$
- $x \gg 2$ 的結果會是 $4'b0011$ ，高位元缺項補0。
- $x \ll 1$ 的結果會是 $4'b1000$ 。

運算子的種類與符號 (4/4)

■ 連結運算子

- 連結運算子可以將不同位元連結成單一個運算元。
- ex : `a = 1'b1; b = 2'b00;`
- `c = {a, b, 3'b001}`，c的結果會是6'b100001。

■ 複製運算子

- 複製運算是將一個運算元複製指定的次數。
- ex : `a = 1'b1; b = 2'b00;`
- `c = { {4{a}}, {2{b}} }`，c的結果會是8'b11110000。

■ 條件運算子

- 條件運算子的格式如 `<條件運算式> ? <真值運算式> : <假值運算式>`。
- 首先計算<條件運算式>，若結果為**true**，則回傳<真值運算式>的計算結果；否則回傳<假值運算式>的計算結果。
- ex : `assign out = sel ? in2 : in1;` //二對一多工器
- ex : `assign out = sel[1] ? (sel[0] ? in4 : in3) : (sel[0] ? in2 : in1);` //四對一多工器

運算子的優先順序

- 雖然各種運算有其優先順序，但使用括號()能使運算式更加容易的被了解。

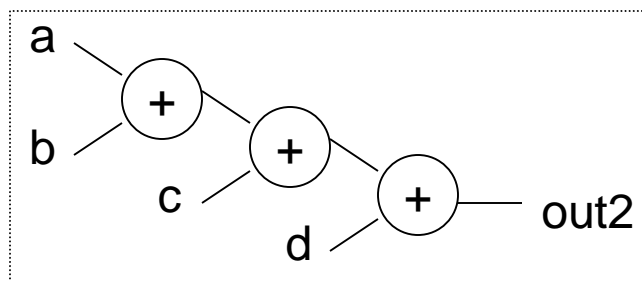
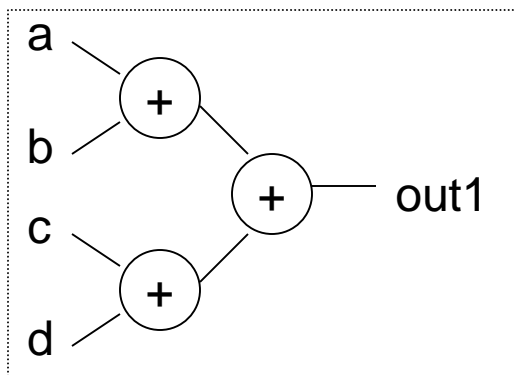
種類	運算子	優先順序
一位元運算、乘法、除法、取餘數	+ - ! ~ * / %	最高
加法、減法、移位	+ - << >>	
比較運算、相等運算	< <= > >= == !=	
化簡運算	& ~& ^ ~^ ~	
邏輯運算	&&	
條件運算	?:	最低

邏輯最佳化

- 設計工程師可以利用括號()來增加程式的可閱讀性，但同時也可以使用括號來做一些指定最佳化的工作。

- ex : assign out1 = (a + b) + (c + d); //如左下圖

- ex : assign out2 = (((a + b) + c) + d); //如右下圖



- 乘法、除法、取餘數的設計通常需要極大的面積，在設計可攜性與最佳化兩方面就必須有所取捨。
 - 可攜性的設計，通常合成出來的電路比較慢、面積也比較大。
 - ex : assign out = a*b;
 - 最佳化的設計，通常可以做出比較快的電路，但犧牲了可攜性。
 - ex : 自己設計一個乘法器。

邏輯閘層次與資料處理層次的比較 (1/2)

四對一多工器

```
module mux_2to1(in1, in2, sel, out);  
  input in1, in2, sel;  
  output out;  
  
  wire a_out, b_out, c_out;  
  
  not a(a_out, sel);  
  and b(b_out, in1, a_out);  
  and c(c_out, in2, sel);  
  or d(out, b_out, c_out);  
  
endmodule
```

mux_2to1

```
module mux_4to1_v2(in1, in2, in3, in4, sel, out);  
  input in1, in2, in3, in4;  
  input [1:0]sel;  
  output out;  
  
  wire a_out, b_out, c_out;  
  
  mux_2to1 a(.in1(in1), .in2(in2), .sel(sel[0]), .out(a_out));  
  mux_2to1 b(.in1(in3), .in2(in4), .sel(sel[0]), .out(b_out));  
  mux_2to1 c(.in1(a_out), .in2(b_out), .sel(sel[1]), .out(out));  
  
endmodule
```

endmodule

mux_4to1

邏輯閘層次

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
```

mux_4to1

邏輯閘層次與資料處理層次的比較 (2/2)

4bit加法器

```
module fulladder(in1, in2, in3, sum, carry);  
    input in1, in2, in3;  
    output sum, carry;  
  
    wire a_out, b_out, d_out;  
  
    xor a(a_out, in1, in2);  
    and b(b_out, in1, in2);  
    xor c(sum, a_out, in3);  
    and d(d_out, a_out, in3);  
    or e(carry, b_out, d_out);  
  
endmodule
```

fulladder

```
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;  
  
    wire [2:0]carry;  
  
    fulladder fa0(in1[0], in2[0], carry_in, sum[0], carry[0]);  
    fulladder fa1(in1[1], in2[1], carry[0], sum[1], carry[1]);  
    fulladder fa2(in1[2], in2[2], carry[1], sum[2], carry[2]);  
    fulladder fa3(in1[3], in2[3], carry[2], sum[3], carry_out);  
  
endmodule
```

adder_4bit

adder_4bit

邏輯閘層次

資料處理層次

```
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;  
  
    assign {carry_out, sum} = in1 + in2 + carry_in;  
  
endmodule
```

adder_4bit

數位邏輯科技

- 數位邏輯 (digital logic)
 - 標準邏輯晶片 (standard logic)
 - ex : TTL
 - ex : CMOS
 - 可程式邏輯設計 (programmable logic design)
 - 可程式邏輯裝置 (PLD)
 - 現場可程式邏輯閘陣列 (FPGA)
 - 複雜可程式邏輯裝置 (CPLD)
 - 部分客製化設計 (semi-custom design)
 - 平台式設計 (platform based design)
 - 核心式設計 (core based design)
 - 閘陣列設計 (gate array design)
 - 標準晶包設計 (standard cell design)
 - 完全客製化設計 (full-custom design)

FPGA介紹

- 現場可程式邏輯閘陣列(Field Programmable Gate Array)，是一個可供使用者設定的邏輯閘元件。
- 目前以硬體描述語言(Verilog或VHDL)所完成的電路設計，可以燒錄至FPGA上進行測試，是現代IC設計驗證的主流。
- 這些可編輯元件可以被用來實現基本的邏輯閘電路(例如：AND、OR、XOR、NOT)或者更複雜的功能(例如：解碼器或數學運算)。
- 系統設計師可以根據需要，通過可編輯的線路把FPGA內部的邏輯區塊連接起來。

數位電路設計流程 (1/2)

- 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)

數位電路設計流程 (2/2)

ISE Project Navigator (M.70d) - D:\logic system lab\lab05\adder\adder.xise - [adder_4bit.v]

File Edit View Project Source Process Tools Window Layout Help

Design

View: Implementation Simulation

Hierarchy

- adder
 - xc6slx25-3ftg256
 - adder_4bit (adder_4bit.v)

```
1 module adder_4bit(in1, in2, carry_in, sum, carry_out);
2   input [3:0]in1, in2;
3   input carry_in;
4   output [3:0]sum;
5   output carry_out;
6
7   assign {carry_out, sum} = in1 + in2 + carry_in;
8
9 endmodule
10
```

No Processes Running

Processes: adder_4bit

- Design Summary/Reports
- Design Utilities
- User Constraints
- Synthesize - XST
- Implement Design
- Generate Programming File
- Configure Target Device
- Analyze Design Using ChipScope

Start Design Files Libraries

Errors Warnings Console

ISim (M.70d) - [Default.wcfg*]

File Edit View Simulation Window Layout Help

Simulation

1.00ns

Name	Value	0 ns	2 ns	4 ns	6 ns	8 ns	10 ns	12 ns	14 ns
carry_in	0	0	1	2	3	4	5	6	7
in1[3:0]	14	0	1	2	3	4	5	6	7
in2[3:0]	15	1	2	3	4	5	6	7	8
carry_out	1	1	4	5	8	9	12	13	0
sum[3:0]	13	1	4	5	8	9	12	13	0

Console

ISim M.70d (signature 0x36e8438f)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
Stopped at time : 15 ns : File "D:\logic system lab\lab05\adder\testbench.v" Line 25
ISim>

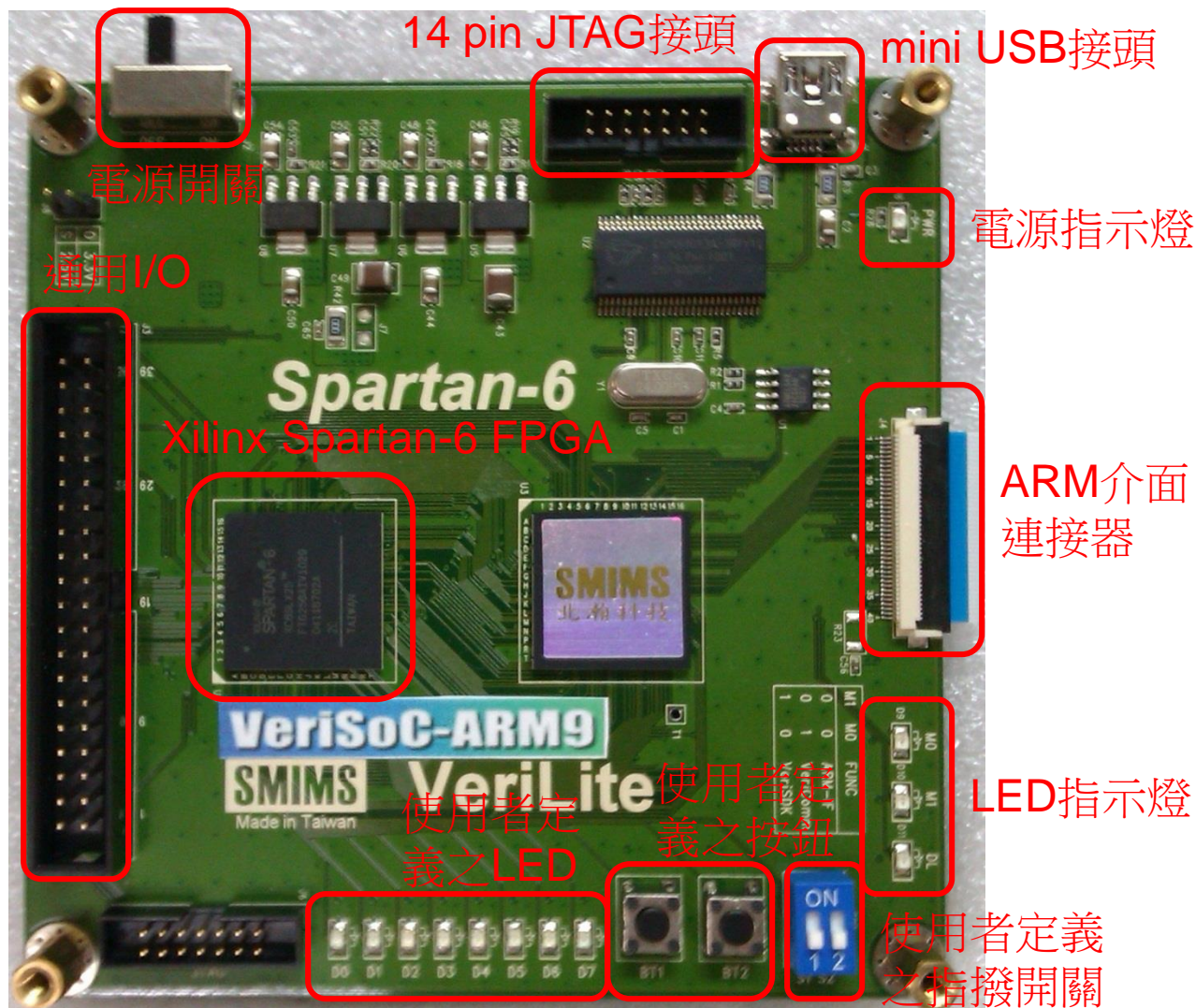
Console Breakpoints Find in Files Results Search Results

Sim Time: 15,000 ps

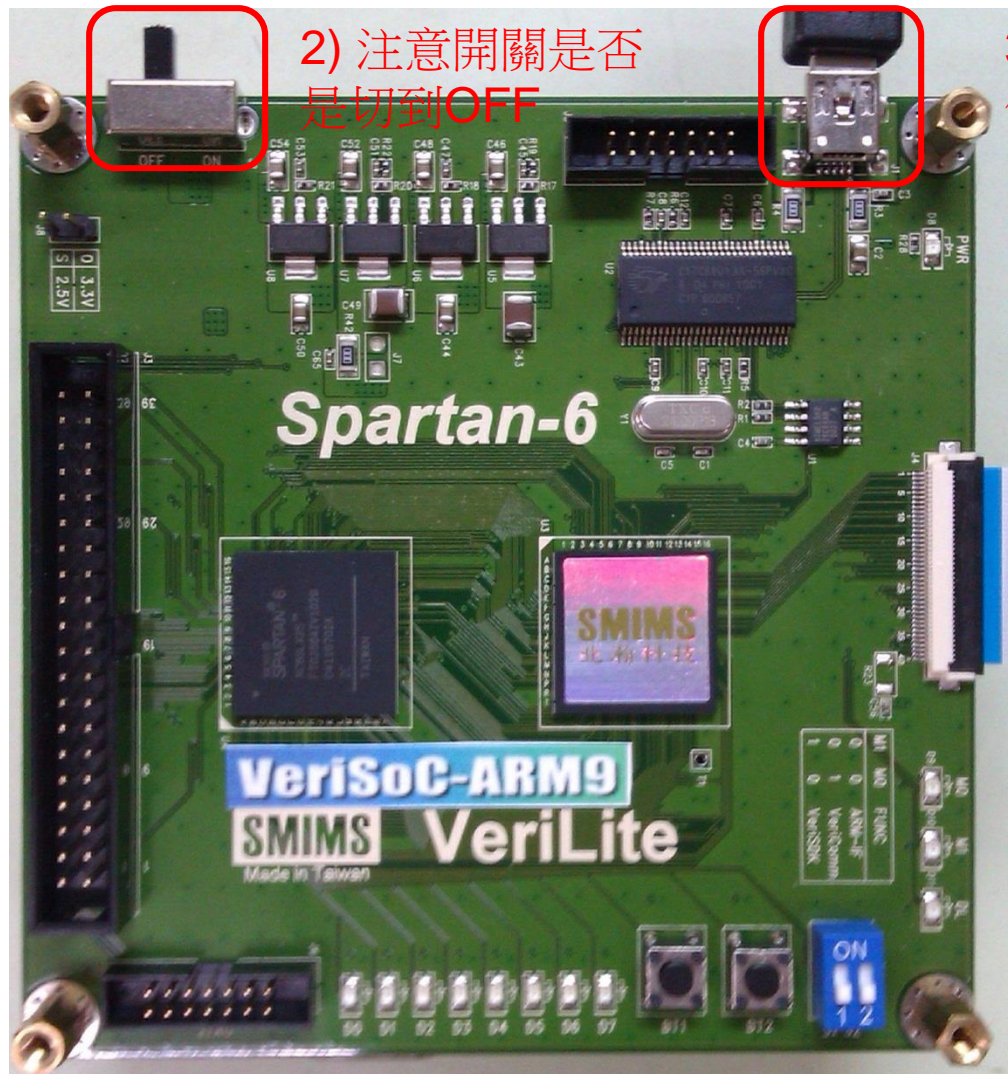
接下來的動作是假設您已經完成了設計、編譯與模擬的步驟

VeriLite FPGA開發板 (1/3)

1) 小心的將
FPGA開發板拿
出來

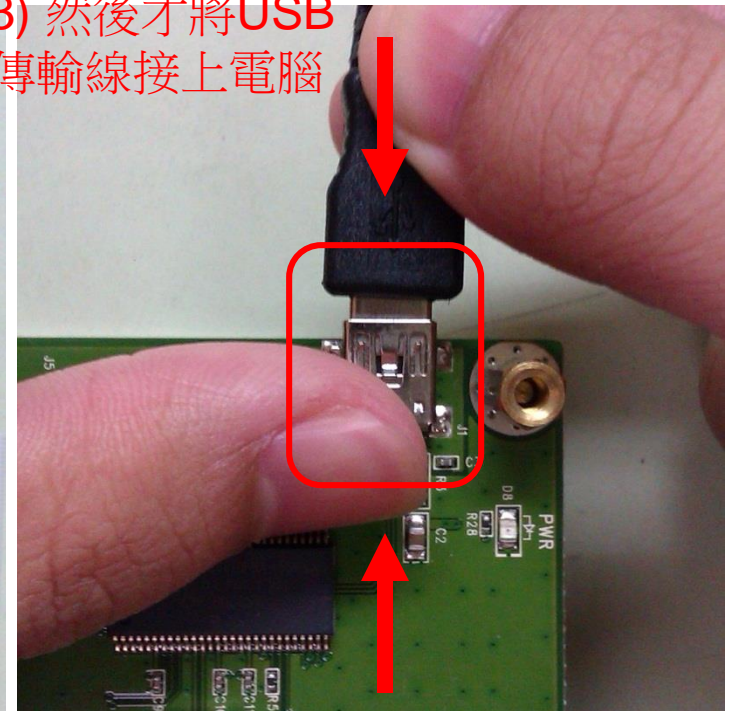


VeriLite FPGA開發板 (2/3)



2) 注意開關是否
是切到OFF

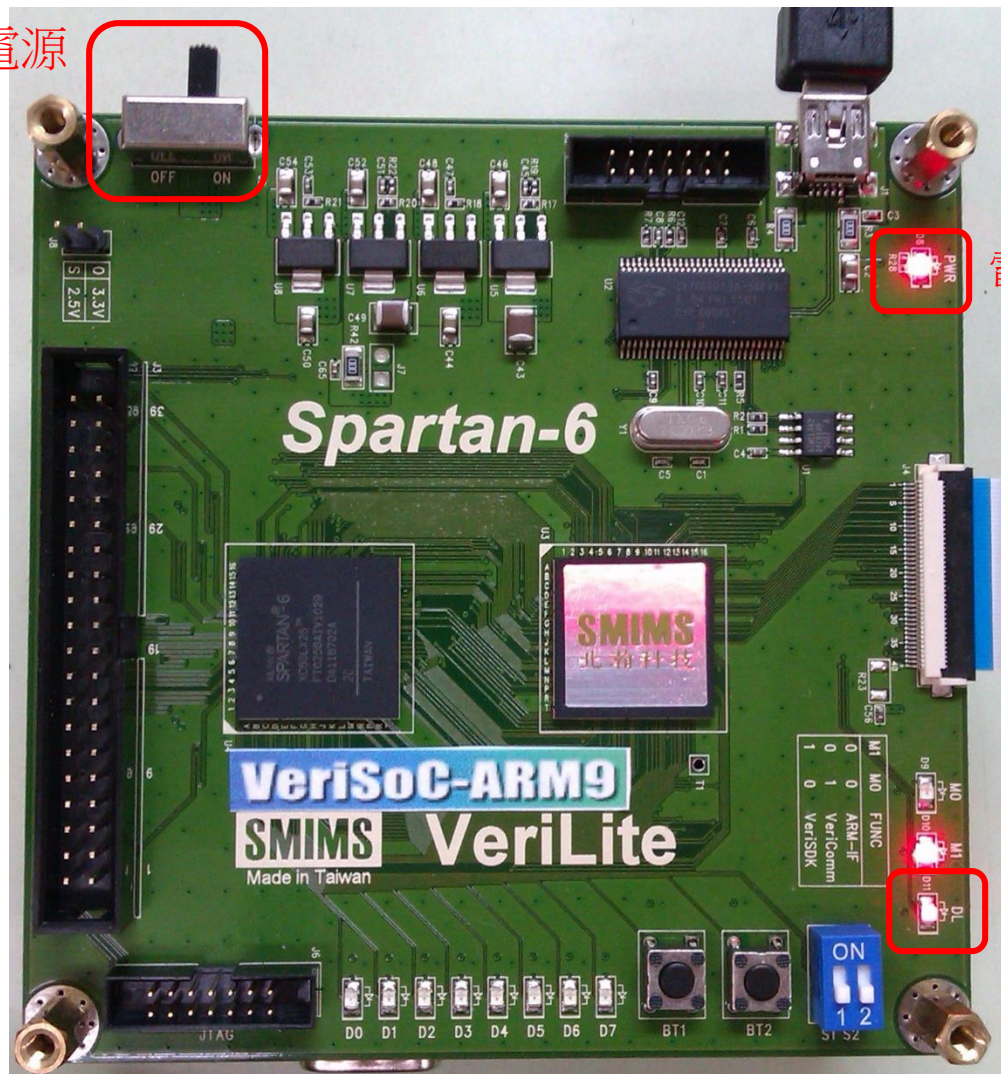
3) 然後才將USB
傳輸線接上電腦



4) 注意!! 將USB傳輸線接上mini
USB接頭時，請用另一隻手壓住
接頭後方，以免接頭脫落；移除
USB傳輸線時請用另一隻手壓住
mini USB接頭後再拔除

VeriLite FPGA開發板 (3/3)

5) 開啟電源



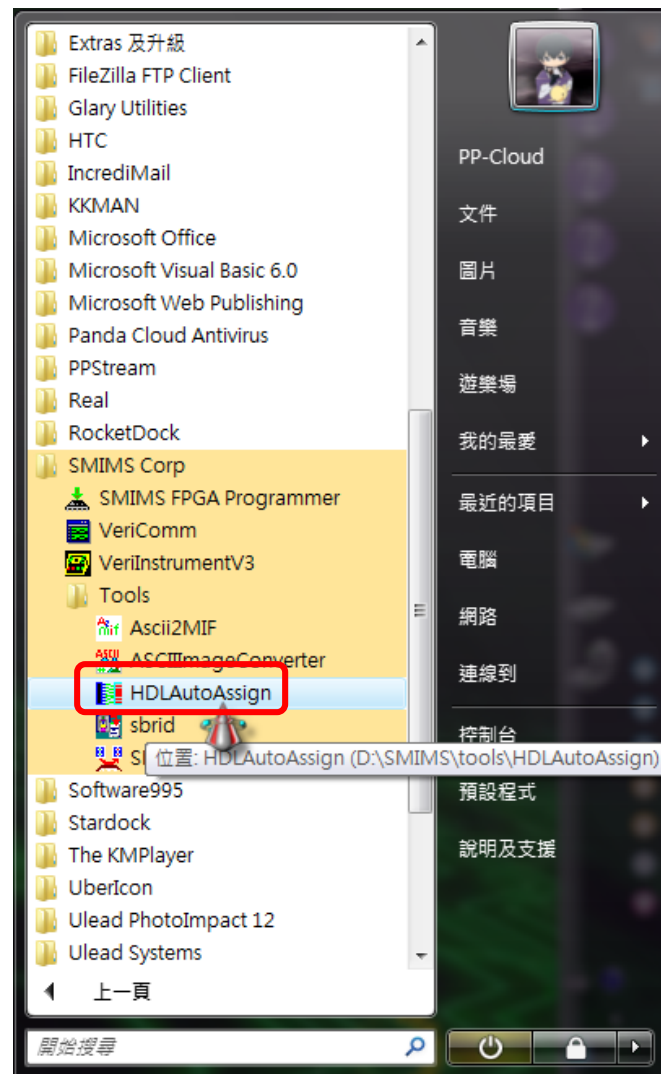
電源指示燈亮

LED閃爍，代表
原始碼尚未燒入
FPGA中

SMIMS 埠指定檔自動產生程式 產生.ucf檔 (1/5)

- 使用Xilinx ISE軟體時，經常需要去指定電路的I/O訊號腳位，但是當訊號一多時，這就是個費時的工程了，而且經常會輸入錯誤。
- 北瀚科技(SMIMS)提供一個小程式幫您解決這個困擾，其會自動產生VeriComm / VeriInstrument所需要的電路接腳對應檔。

1) 開始 > SMIMS Corp >
Tools > HDLAutoAssign



SMIMS 埠指定檔自動產生程式 產生.ucf檔 (2/5)

HDL Auto Assign Pin - SMIMS Corp.

adder_4bit.v()
VLXSP6(XC6SLX25)

VeriComm Mode Assignment Generate

User design module file

2) 選取設計中的最上層模組

D:\logic system lab\lab05\adder\adder_4bit.v

FPGA Board

VLXSP6 (VeriLite Xilinx Spartan-6 XC6SLX16, XC6SLX25)

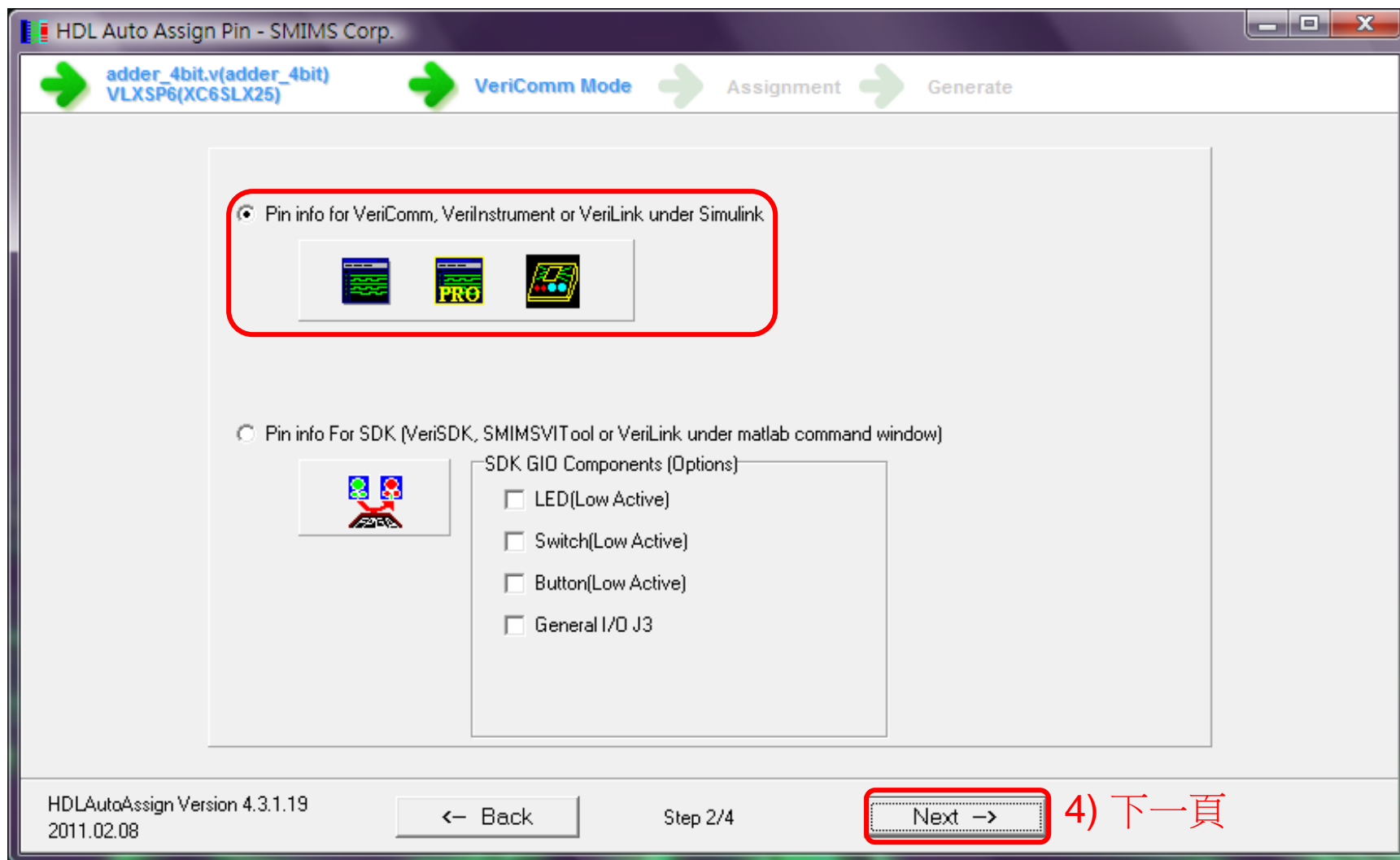
FPGA Type : XC6SLX25

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

HDLAutoAssign Version 4.3.1.19
2011.02.08

Back Step 1/4 Next

SMIMS 埠指定檔自動產生程式 產生.ucf檔 (3/5)



SMIMS 埠指定檔自動產生程式

產生.ucf檔 (4/5)

HDL Auto Assign Pin - SMIMS Corp.

add_4bit.v(add_4bit) VLXSP6(XC6SLX25) VeriComm Mode Assignment Generate

add_4bit

Name	Width	Description
+ in1	4	input bus
+ in2	4	input bus
carry_in	1	input
+ sum	4	output bus
carry_out	1	output

VeriComm Mode IO

Name	Width	Description
+ F1...	4	input bus
+ H2...	4	input bus
J1	1	Input[8]
+ N4...	4	output bus
R5	1	Output[4]
H4	1	Clock source
K5	1	Input[9]
K6	1	Input[10]
J4	1	Input[11]
K2	1	Input[12]
K1	1	Input[13]
L4	1	Input[14]
L5	1	Input[15]
L3	1	Input[16]
L1	1	Input[17]

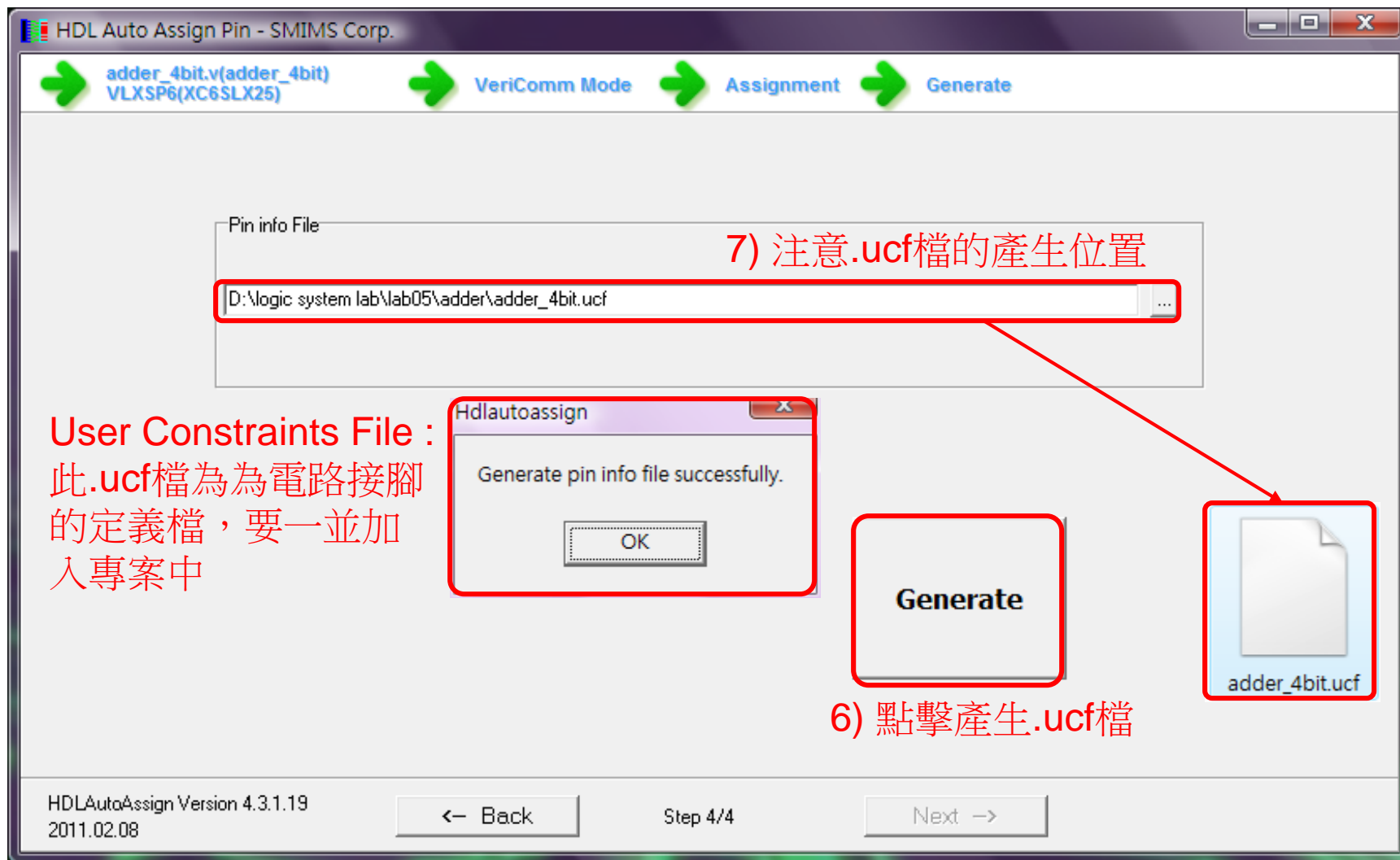
HDLAutoAssign Version 4.3.1.19
2011.02.08

← Back Step 3/4 Next →

5) 下一頁

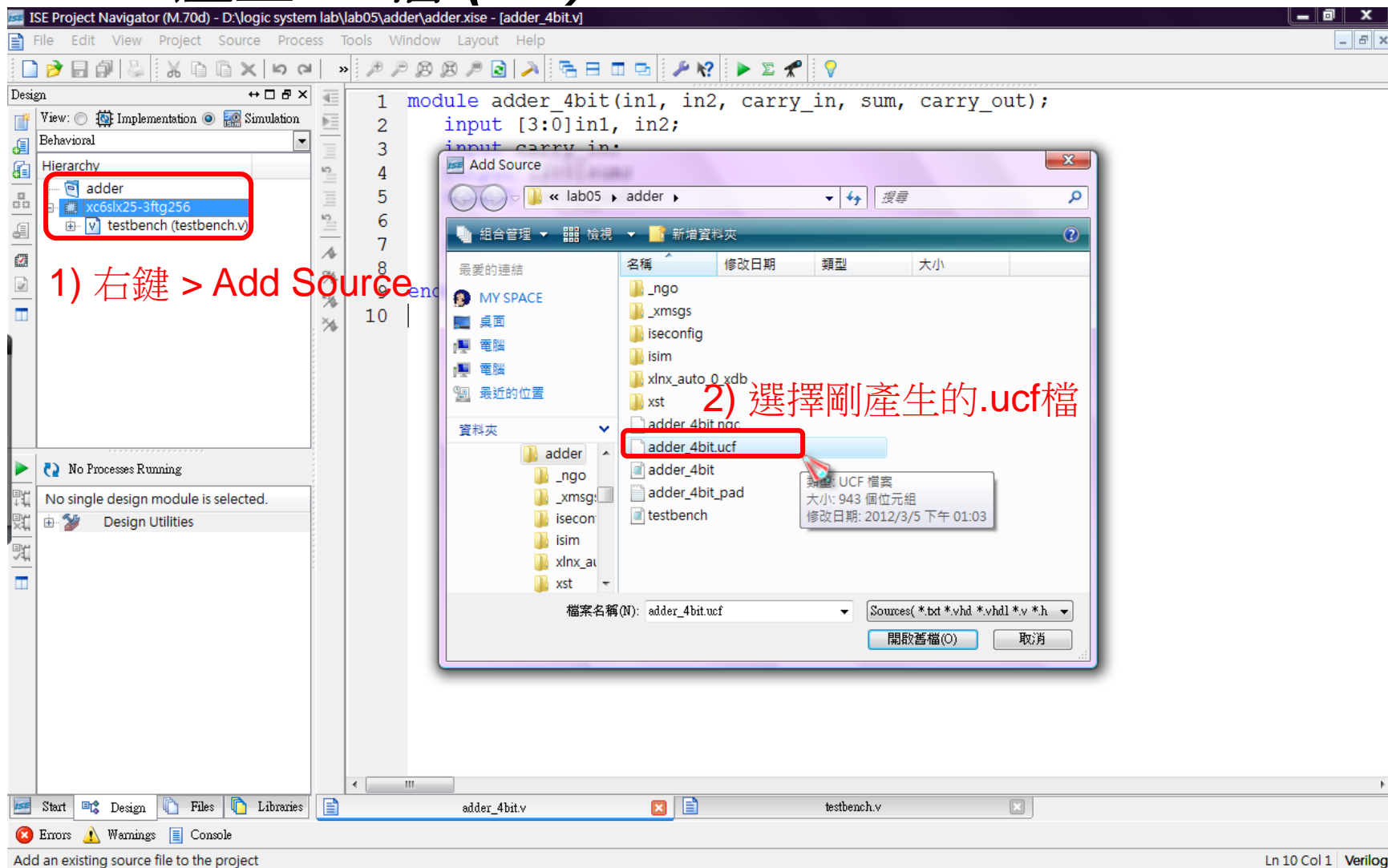
SMIMS 埠指定檔自動產生程式

產生.ucf檔 (5/5)



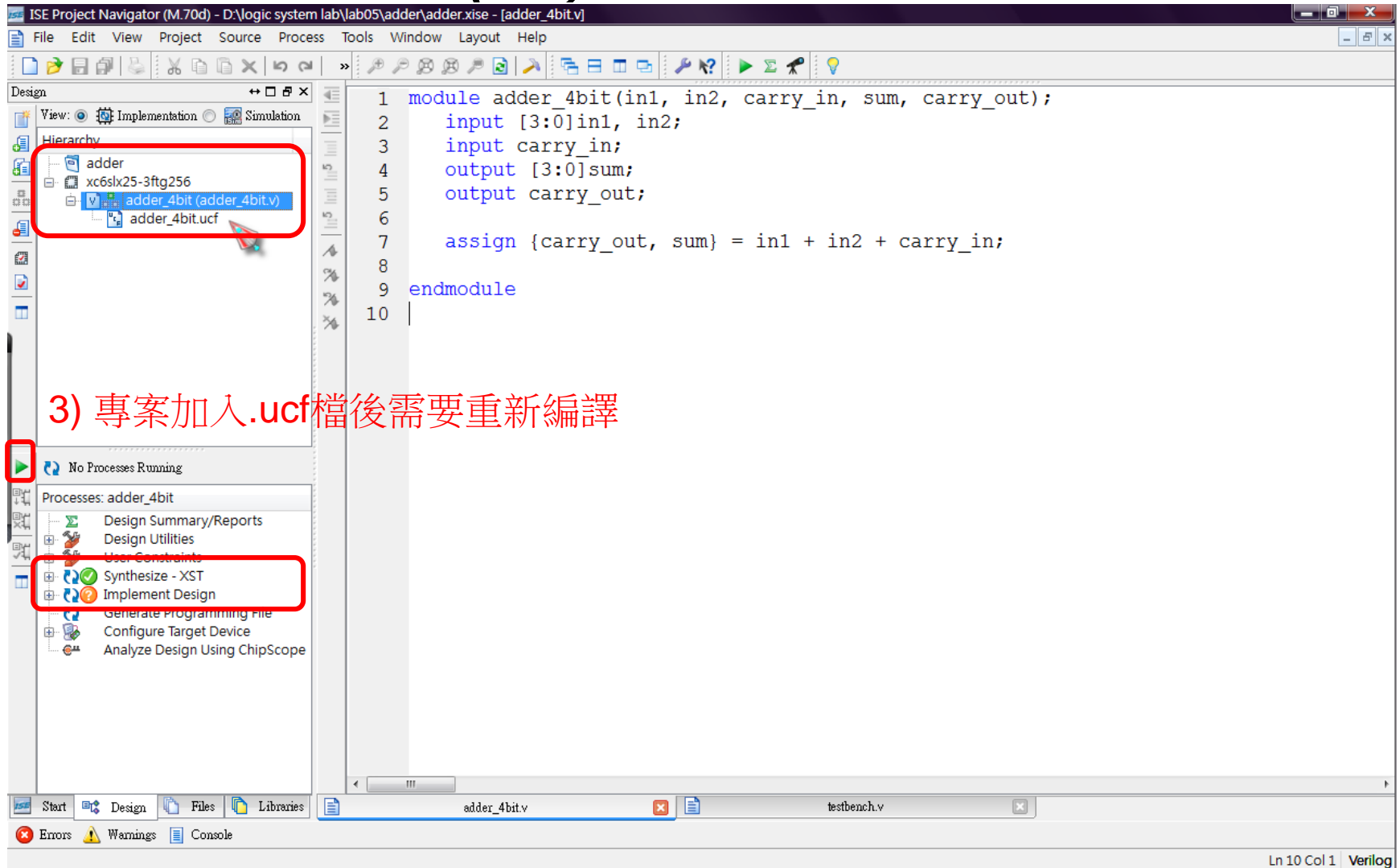
Xilinx ISE

產生.bit檔 (1/4)



Xilinx ISE

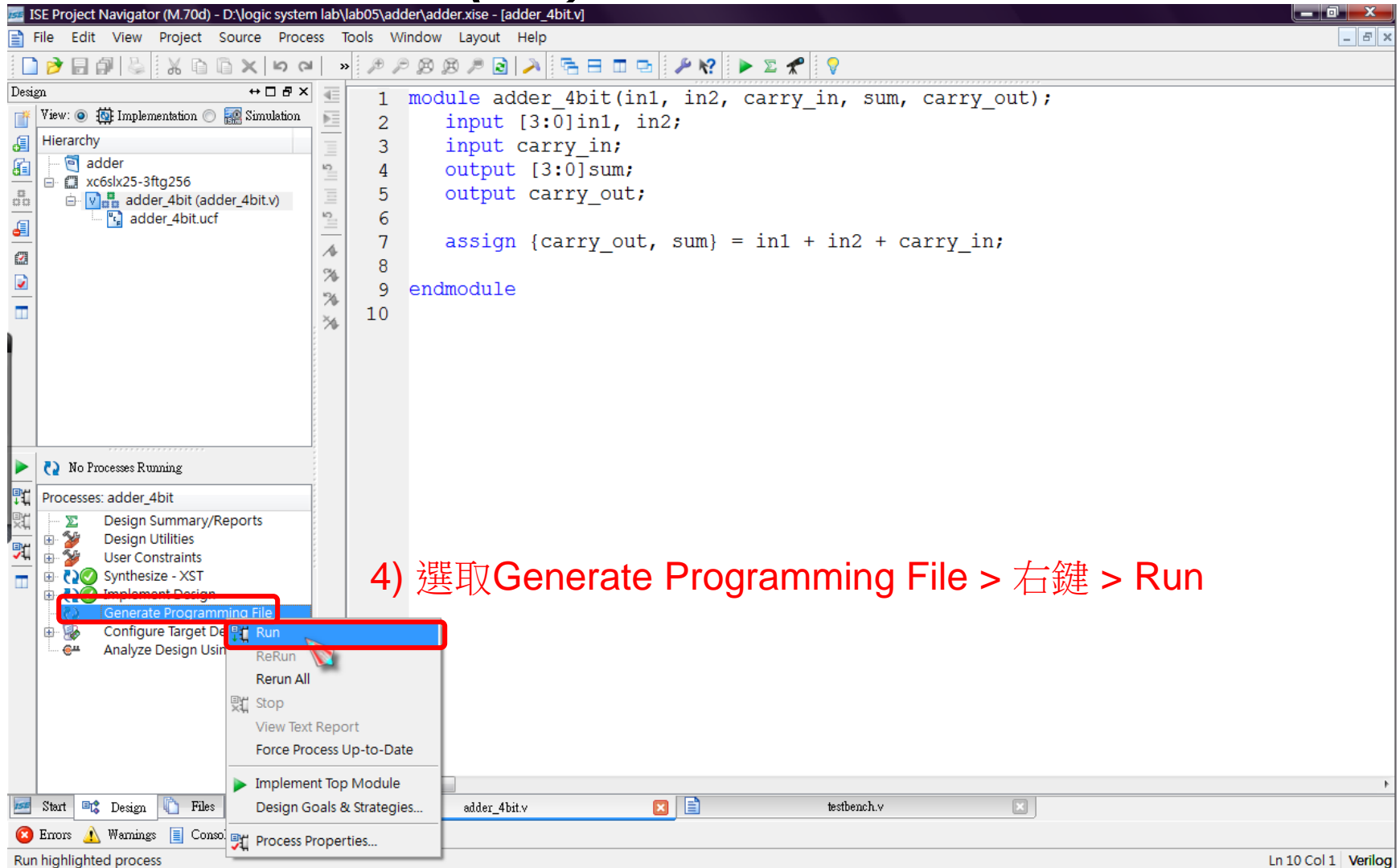
產生.bit檔 (2/4)



Ln 10 Col 1 Verilog

Xilinx ISE

產生.bit檔 (3/4)



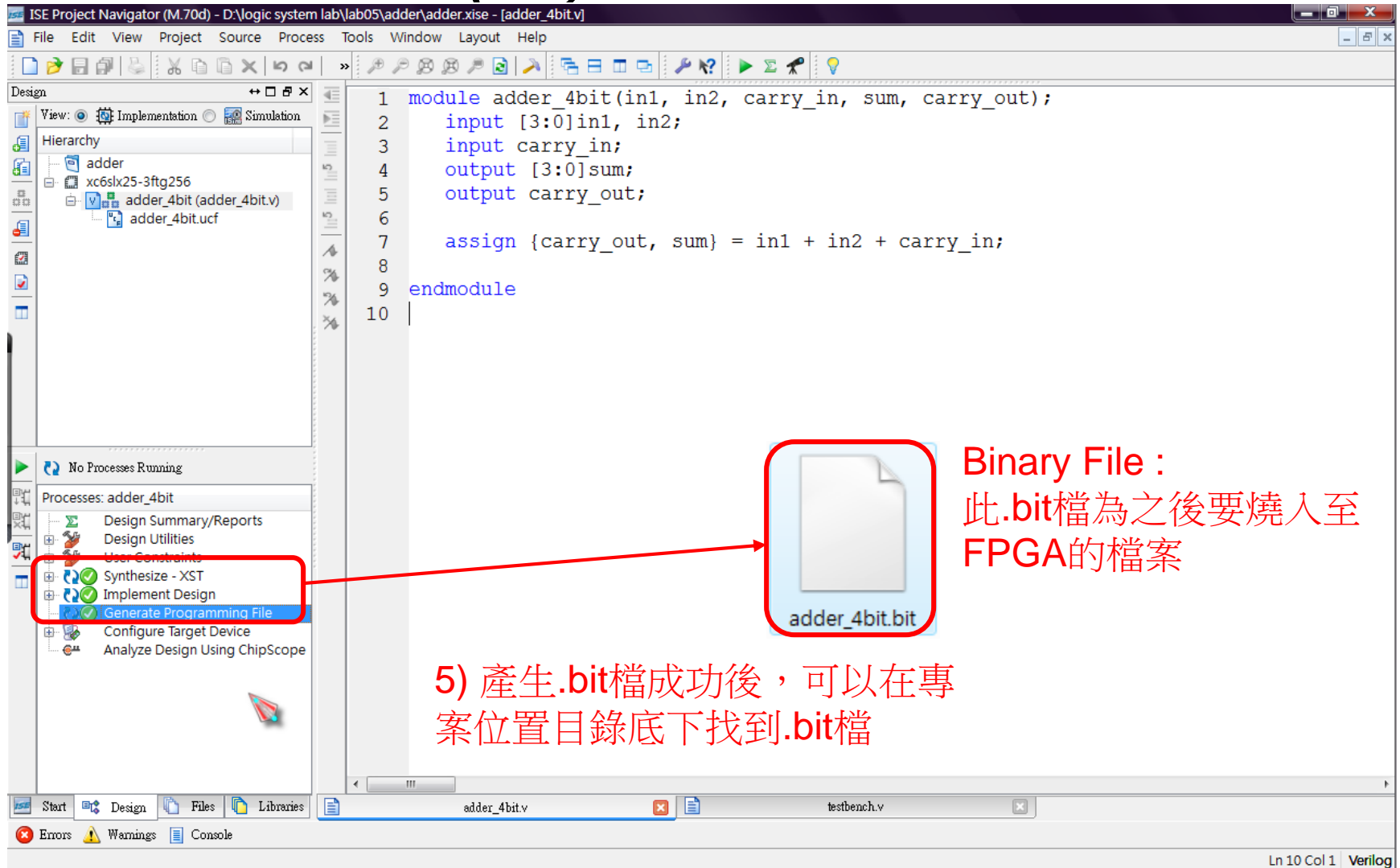
The screenshot displays the Xilinx ISE Project Navigator interface. The 'Design' tab is active, showing a hierarchy of files: 'adder' (xc6slx25-3ftg256), 'adder_4bit (adder_4bit.v)', and 'adder_4bit.ucf'. The 'Processes' pane on the left lists various tasks, with 'Generate Programming File' highlighted. A context menu is open over this option, showing 'Run' as the selected action. The main editor window displays the Verilog code for a 4-bit adder module.

```
1 module adder_4bit(in1, in2, carry_in, sum, carry_out);
2     input [3:0]in1, in2;
3     input carry_in;
4     output [3:0]sum;
5     output carry_out;
6
7     assign {carry_out, sum} = in1 + in2 + carry_in;
8
9 endmodule
10
```

4) 選取Generate Programming File > 右鍵 > Run

Xilinx ISE

產生.bit檔 (4/4)



The screenshot shows the Xilinx ISE Project Navigator interface. The 'Processes' pane on the left lists the steps for the 'adder_4bit' project. The 'Generate Programming File' step is highlighted with a red box. A red arrow points from this step to a file icon labeled 'adder_4bit.bit'.

Binary File :
此.bit檔為之後要燒入至FPGA的檔案

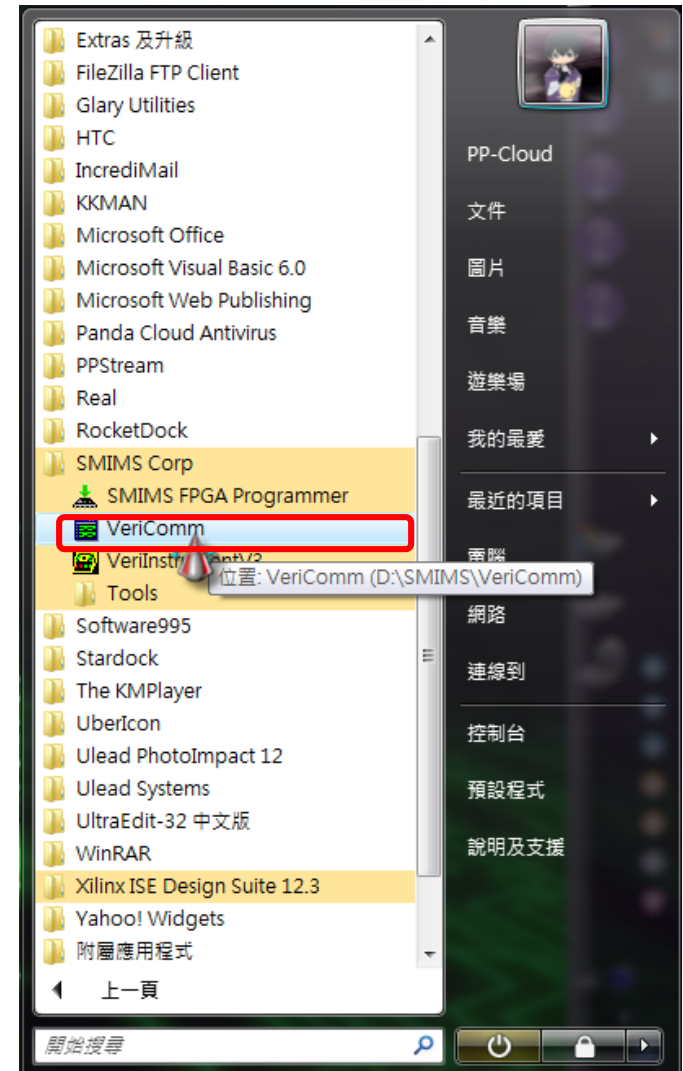
5) 產生.bit檔成功後，可以在專案位置目錄底下找到.bit檔

```
1 module adder_4bit(in1, in2, carry_in, sum, carry_out);
2     input [3:0]in1, in2;
3     input carry_in;
4     output [3:0]sum;
5     output carry_out;
6
7     assign {carry_out, sum} = in1 + in2 + carry_in;
8
9 endmodule
10
```

SMIMS VeriComm (1/3)

- SMIMS VeriComm提供容易使用的驗證環境，能快速並簡單的對SMIMS FPGA平台上的硬體描述語言做除錯與分析。

1) 開始 > SMIMS Corp > VeriComm



SMIMS VeriComm (2/3)

FPGA燒入 (1/2)

Wave Form **FPGA Programmer** Altera Programmer

3) 選取FPGA Programmer

4) 選取先前所產生的.bit檔

5) 將.bit檔燒入至FPGA

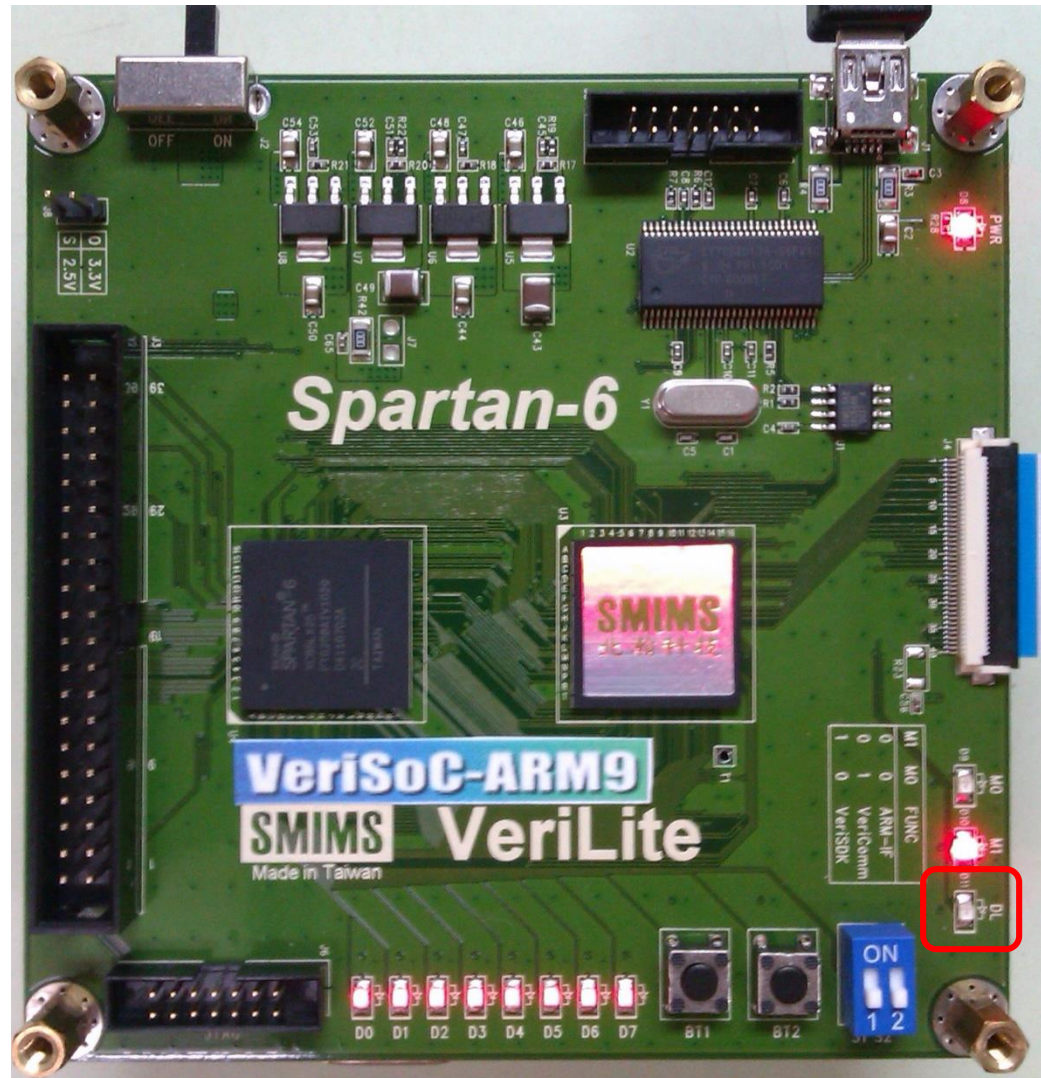
6) 確認燒入成功

2) 確認FPGA有正確連接上電腦

FPGA:Ready

SMIMS VeriComm (2/3)

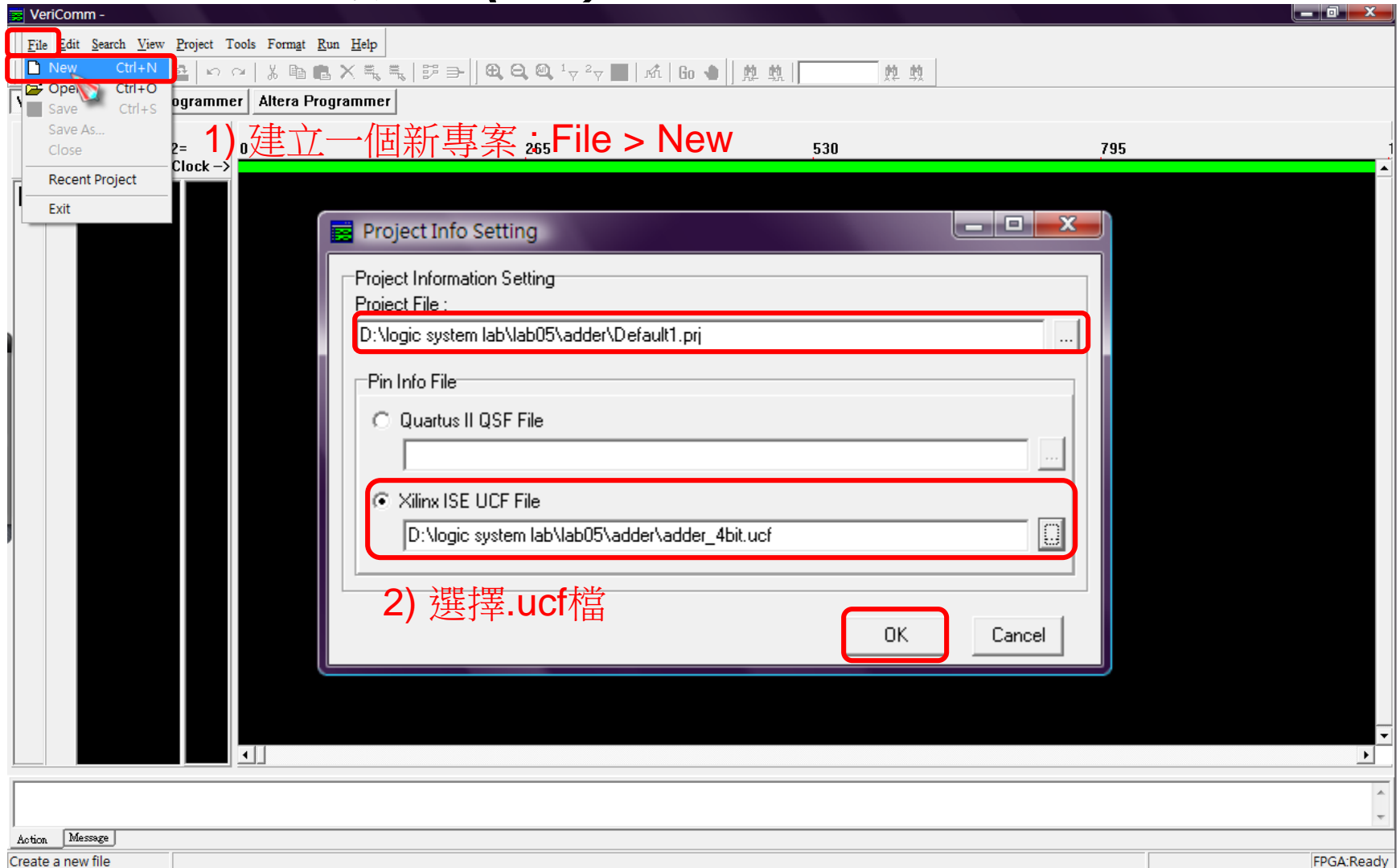
FPGA燒入 (2/2)



7) 將.bit檔燒入FPGA後，LED就會停止閃爍

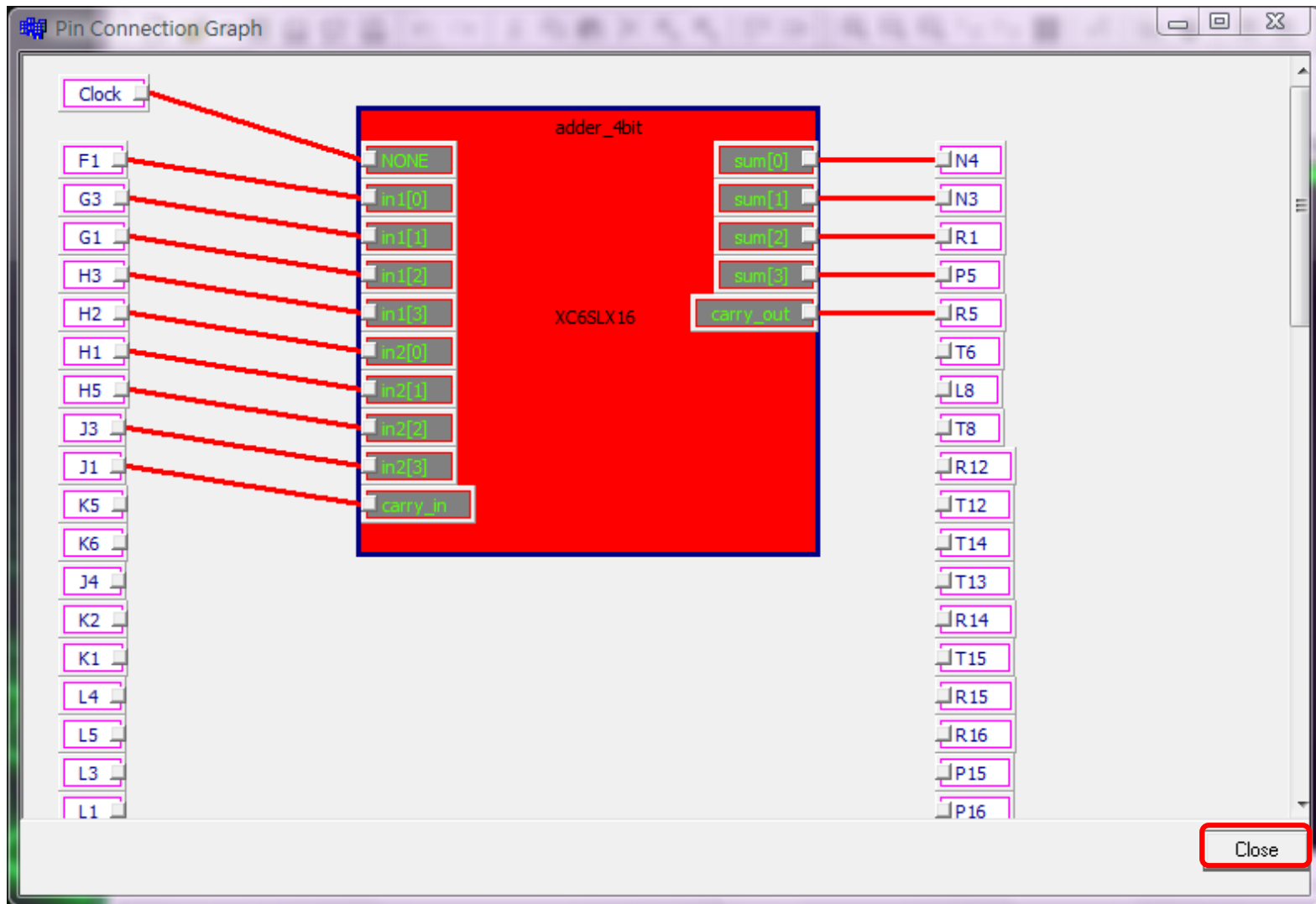
SMIMS VeriComm (3/3)

FPGA驗證 (1/7)



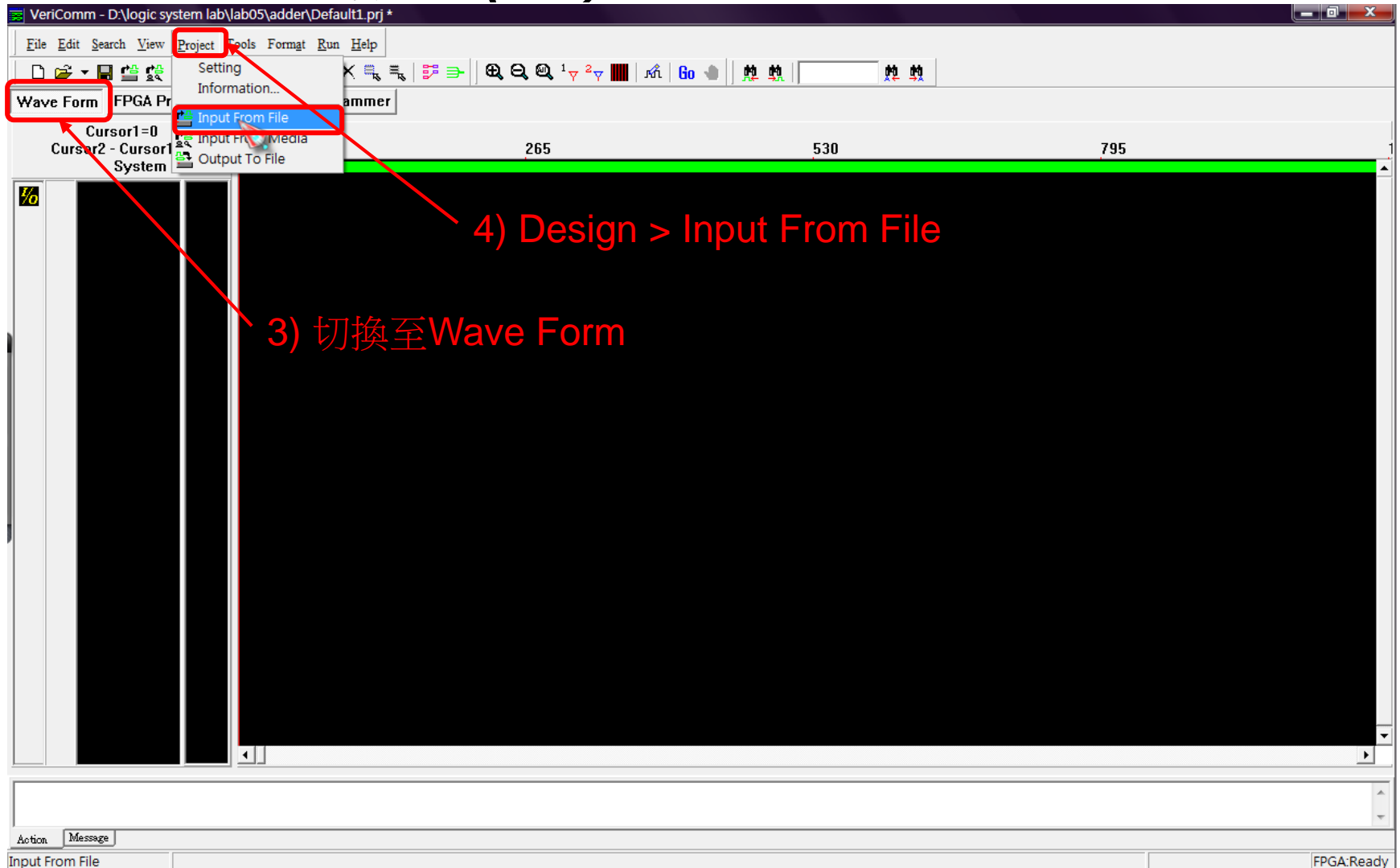
SMIMS VeriComm (3/3)

FPGA驗證 (2/7)



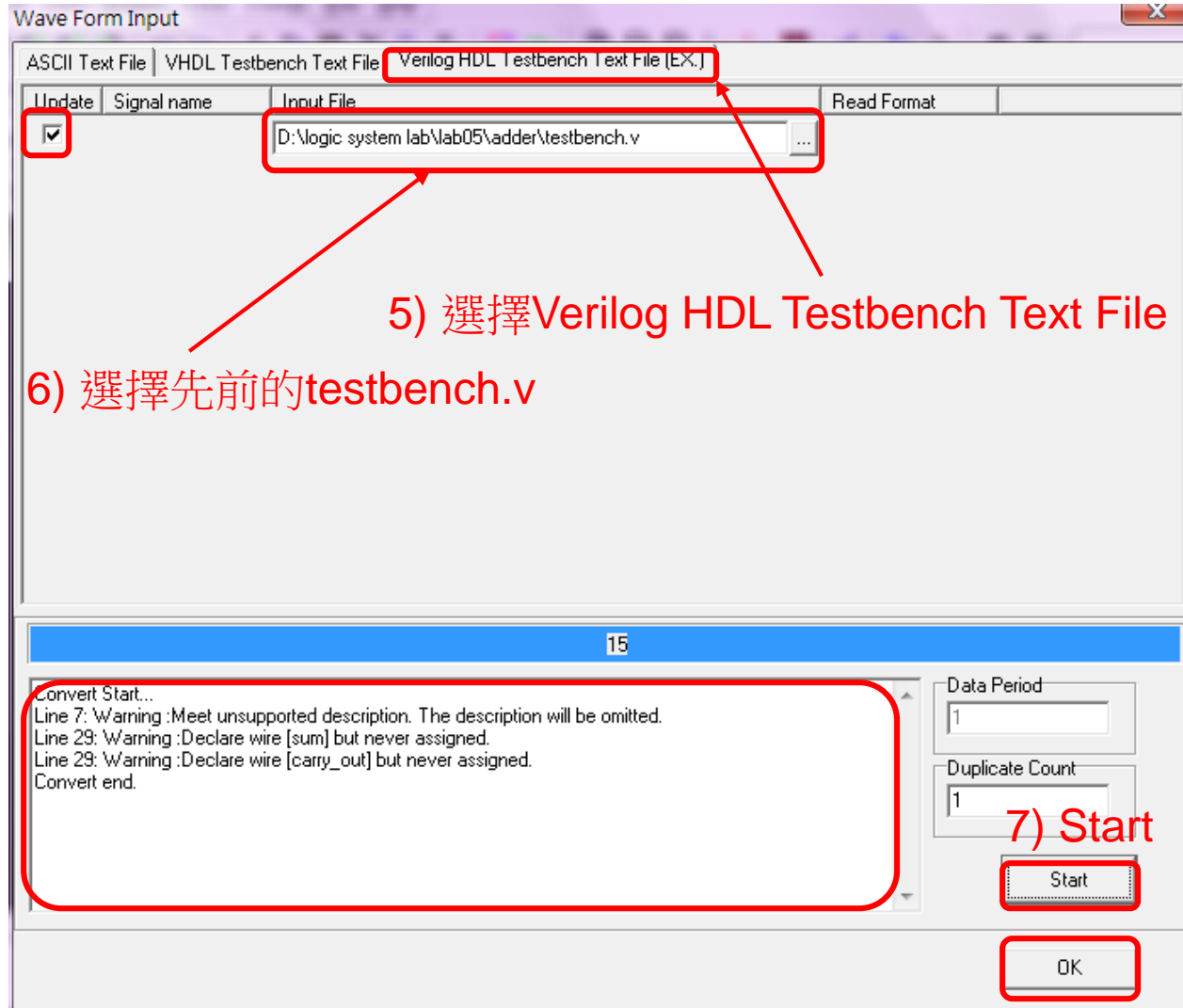
SMIMS VeriComm (3/3)

FPGA驗證 (3/7)



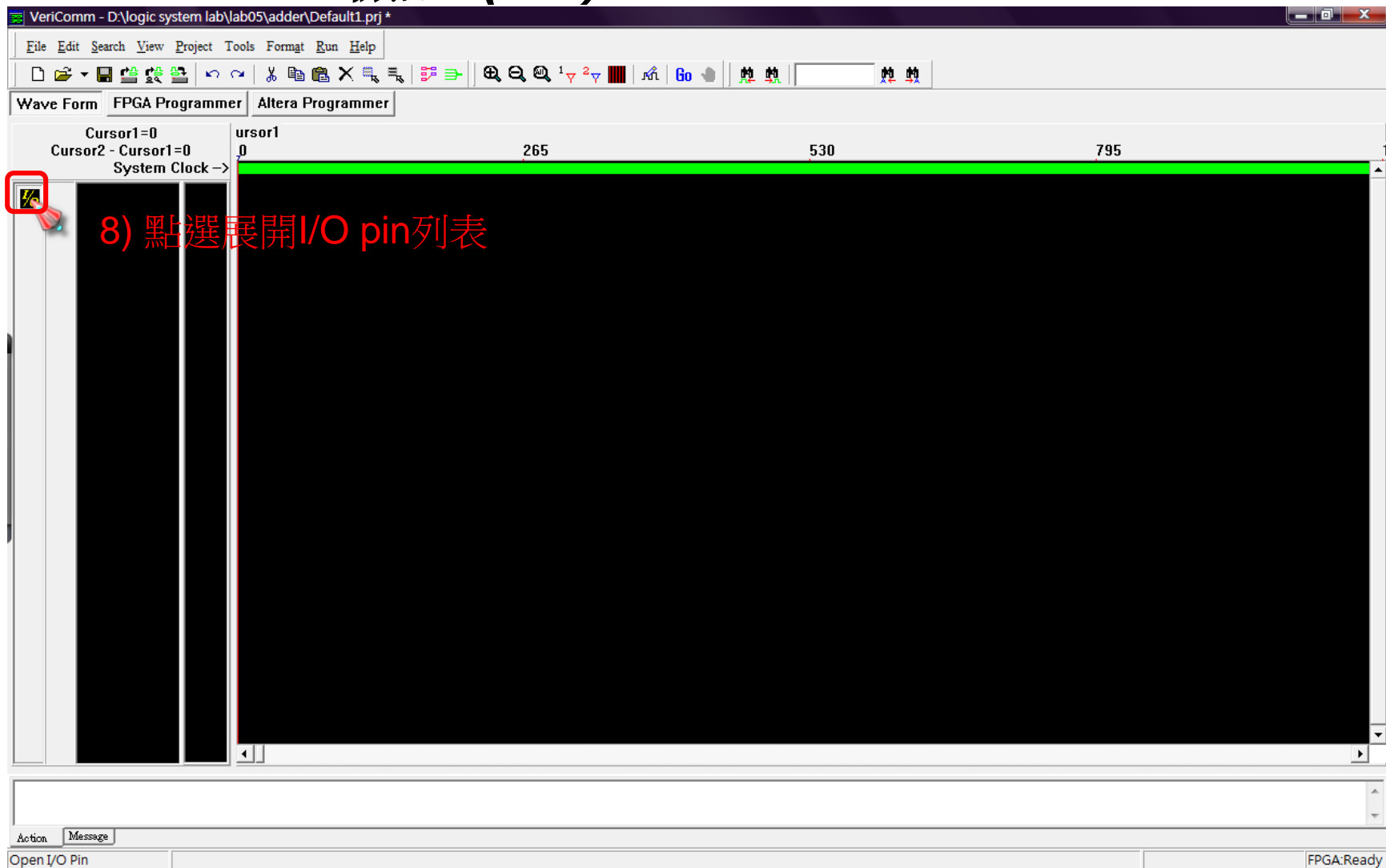
SMIMS VeriComm (3/3)

FPGA驗證 (4/7)



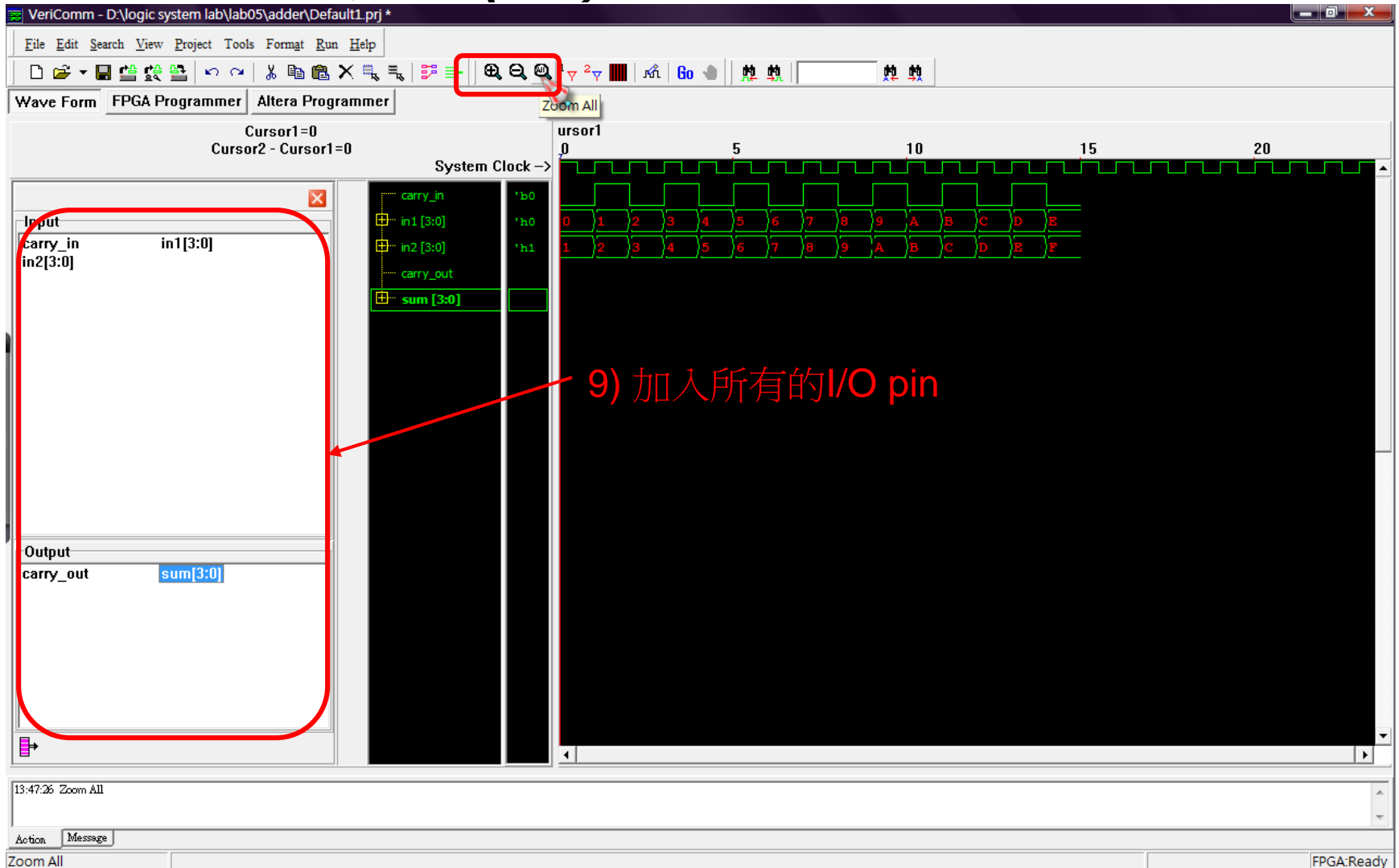
SMIMS VeriComm (3/3)

FPGA驗證 (5/7)



SMIMS VeriComm (3/3)

FPGA驗證 (6/7)



SMIMS VeriComm (3/3)

FPGA驗證 (7/7)

VeriComm - D:\logic system lab\lab05\adder\Default1.prj *

File Edit Search View Project Tools **Format** Run Help

Radix Binary Hex Decimal Signed Decimal **Go**

Wave Form FPGA Programmer Altera Program

Cursor1=0
Cursor2 - Cursor1=0
System Clock ->

10) 開始驗證

11) 選取所有的pin腳，將顯示格式切換為十進制，以方便驗證

Signal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
carry_in	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
in1 [3:0]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
in2 [3:0]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	15
carry_out	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
sum [3:0]	1	4	5	8	9	12	13	0	1	4	5	8	9	12	13	13

13:49:41 Multiple Select carry_out
13:49:41 Multiple Select sum [3:0]

Action Message

Sending data complete

FPGA:Ready

基礎題 (一)

4bit加法器

- 請參考adder_4bit.v的範例原始碼與操作流程。
 - 設計時，請利用Verilog資料處理層次語法寫出4bit加法器模組，並且確認編譯後沒有error或warning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，請觀察波形結果是否如期望一般。

資料處理層次

```
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;  
  
    assign {carry_out, sum} = in1 + in2 + carry_in;  
  
endmodule
```

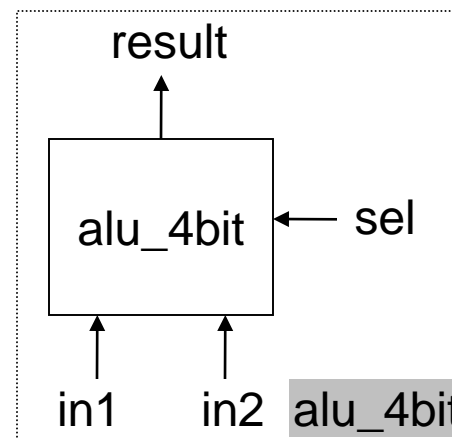
adder_4bit

基礎題 (二)

4bit簡易算數邏輯單元

- 請寫出4bit簡易算數邏輯單元(ALU)。
 - 設計時，請利用Verilog資料處理層次語法寫出4bit簡易算數邏輯單元模組，並且確認編譯後沒有error或warning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，請觀察波形結果是否如期望一般。

sel	result
000	$in1 + in2$
001	$in1 - in2$
010	$in1 \& in2$
011	$in1 in2$
100	$in1 \wedge in2$
101	$in1 \gg in2$
110	$in1 \ll in2$
111	$in1 > in2$



挑戰題

4bit前瞻式進位加法器

- 請寫出4bit前瞻式進位加法器(carry lookahead adder)。
 - 設計時，請利用Verilog資料處理層次語法寫出4bit前瞻式進位加法器模組，並且確認編譯後沒有error或warning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，請觀察波形結果是否如期望一般。

carry propagate : $p_i = a_i \oplus b_i$, $i = 0 \sim 3$

carry generate : $g_i = a_i \bullet b_i$, $i = 0 \sim 3$

sum : $s_i = p_i \oplus c_i$, $i = 0 \sim 3$

carry : $c_{i+1} = g_i + (p_i \bullet c_i)$, $i = 0 \sim 3$

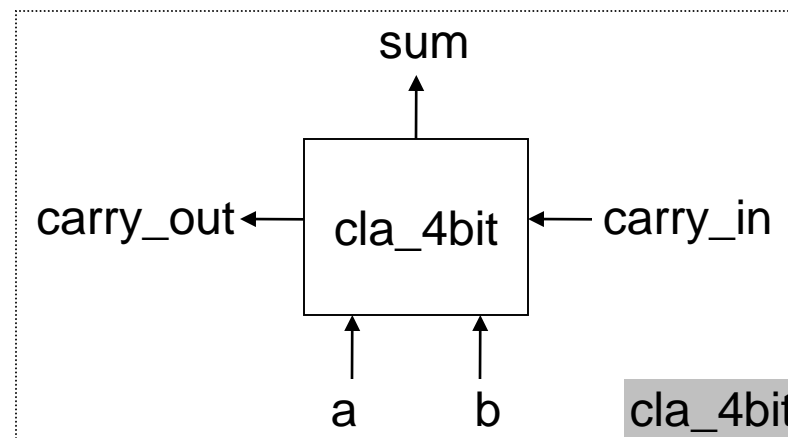
$c_0 = \text{carry_in}$

$c_1 = g_0 + (p_0 \bullet c_0)$

$c_2 = g_1 + (p_1 \bullet c_1) = g_1 + (p_1 \bullet g_0) + (p_1 \bullet p_0 \bullet c_0)$

$c_3 = g_2 + (p_2 \bullet c_2) = g_2 + (p_2 \bullet g_1) + (p_2 \bullet p_1 \bullet g_0) + (p_2 \bullet p_1 \bullet p_0 \bullet c_0)$

$c_4 = g_3 + (p_3 \bullet c_3) = g_3 + (p_3 \bullet g_2) + (p_3 \bullet p_2 \bullet g_1) + (p_3 \bullet p_2 \bullet p_1 \bullet g_0) + (p_3 \bullet p_2 \bullet p_1 \bullet p_0 \bullet c_0) = \text{carry_out}$



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