Computer Architecture and System Laboratory

Laboratory 5

Verilog 基本介紹

Department of Electrical Engineering
National Cheng Kung University

實驗目的

- 了解硬體描述語言的概念
- 學習利用Verilog設計階層式的模組並且驗證

使用器材

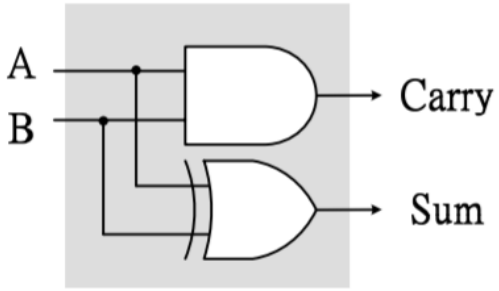
- 桌上型電腦
- Xilinx FPGA 板

Verilog基本介紹

HDL

➢硬體描述語言 HDL (Hardware Description Language)

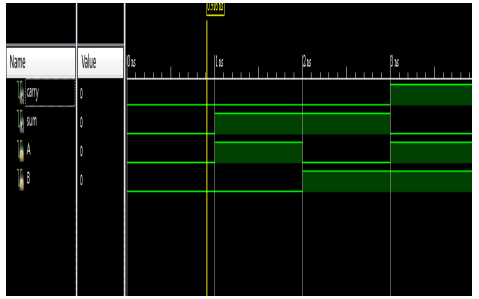
- 設計者可以利用這類語言來描述自己設計的硬體電路並且在電腦上模擬



Design

Carry = A AND B
Sum = A XOR B

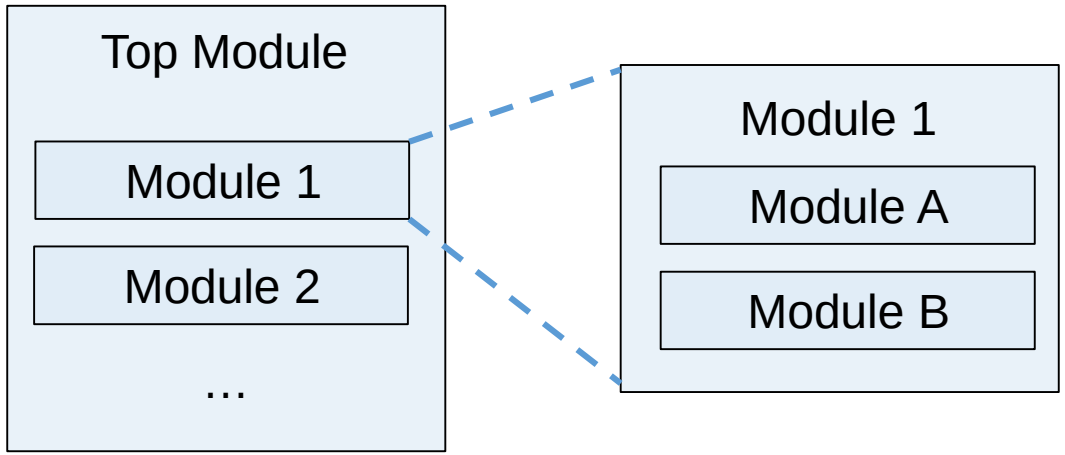
HDL



Simulation

Verilog

- Verilog是一種硬體描述語言
  - 在1995年被接納為IEEE標準
- 以階層式(Hierarchy)的模組(Module)來描述硬體電路
  - 大型的電路模組可以由小塊的模組組合而成



Verilog - 資料型態

➢Verilog主要利用兩種資料型態模擬邏輯電路

- 連接線 (Net) : 用於連接接點

ex: wire, input, output

- 暫存器 (Register)：用於儲存資料

ex: reg, output reg

wire

➢資料總共有四種狀態

0	邏輯0
1	邏輯1
x	未知的值
z	高阻抗 或 浮接

Verilog – 向量

➢宣告變數時，可以左側中括號([ ])設定該變數的寬度

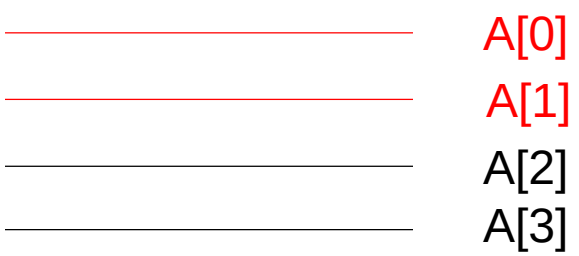
- 此類資料型態稱為向量

[#high：#low] 變數名稱

//該向量的寬度為
//　#high - #low - 1

wire [3:0] A; //宣告4bit連接線

A[1:0] //選取A的其中兩個位元



Verilog – 陣列

➢宣告變數時，可以利用右側中括號([ ])產生陣列

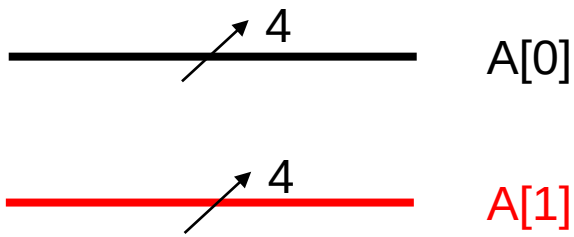
- 最多一維陣列

變數名稱 [#low : #high]

//該陣列的長度為
// #high - #low - 1

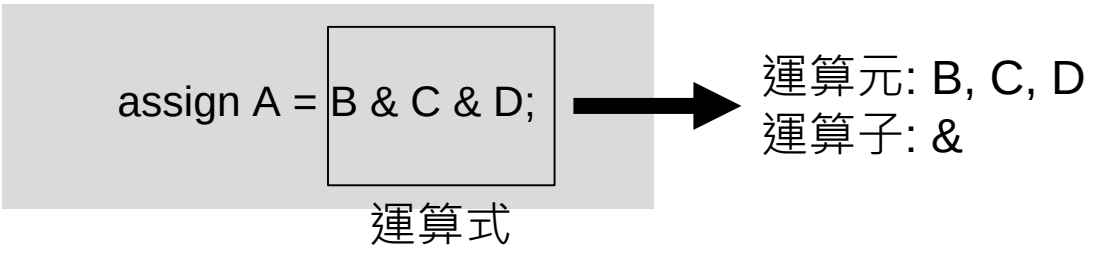
wire [3:0] A [0:1]; //宣告兩條4bit連接線

A[1] //選取其中一條4bit連接線



Verilog – 運算式

- 在資料處理模式中對於一條導線與其他調導向相對應關係陳述統稱為**運算式**(Expression)
- 運算式包含有運算子(Operator)與運算元兩部分
- **運算元**的資料型態包括常數、連接線、暫存器等等
- **運算子**作用在運算元上以得到想要的結果



Verilog - 運算子

➢Verilog支援許多種運算子，下面列出幾種

運算子種類	符號	運算功能	例子
位元運算符號	~	NOT	~A
	&	AND	A & B
	|	OR	A | B
	^	XOR	A ^ B
算數運算符號	+	加法	A + B
邏輯運算符號	&&	邏輯上的”AND”	A && B
連結符號	{}	連結	{A, B}

Verilog – 數字格式

<size>’<base format><number>

<size>是以十進位來表示數字位數(bits)。

<base format> 是用以定義此數為**十進位**(‘d或’D)，
十六進位(‘h或’H)，**二進位**(‘b或’B)，
八進位(‘o或’O)。

<number> 大小寫皆可(0~9, a~f, A~F, x ,X, z, X)，
x是代表不確定的值(unknown)， z是
代表高阻抗。

```
1’b0        // 這是一1-bit二進位數
4’b1111     // 這是一4-bit二進位數
12’habc     // 這是一12-bit十六進位數
32’dz       // 這是一32-bit高阻抗
```

Verilog – 模組(module) (1/3)

- 設計者可以將硬體電路抽象化為一塊模組
  - 簡化設計的複雜度
  - 方便重複利用

```
module 模組名稱(  I/O ports );

    宣告 I/O ports

    宣告資料型態

    內部電路敘述

endmodule
```

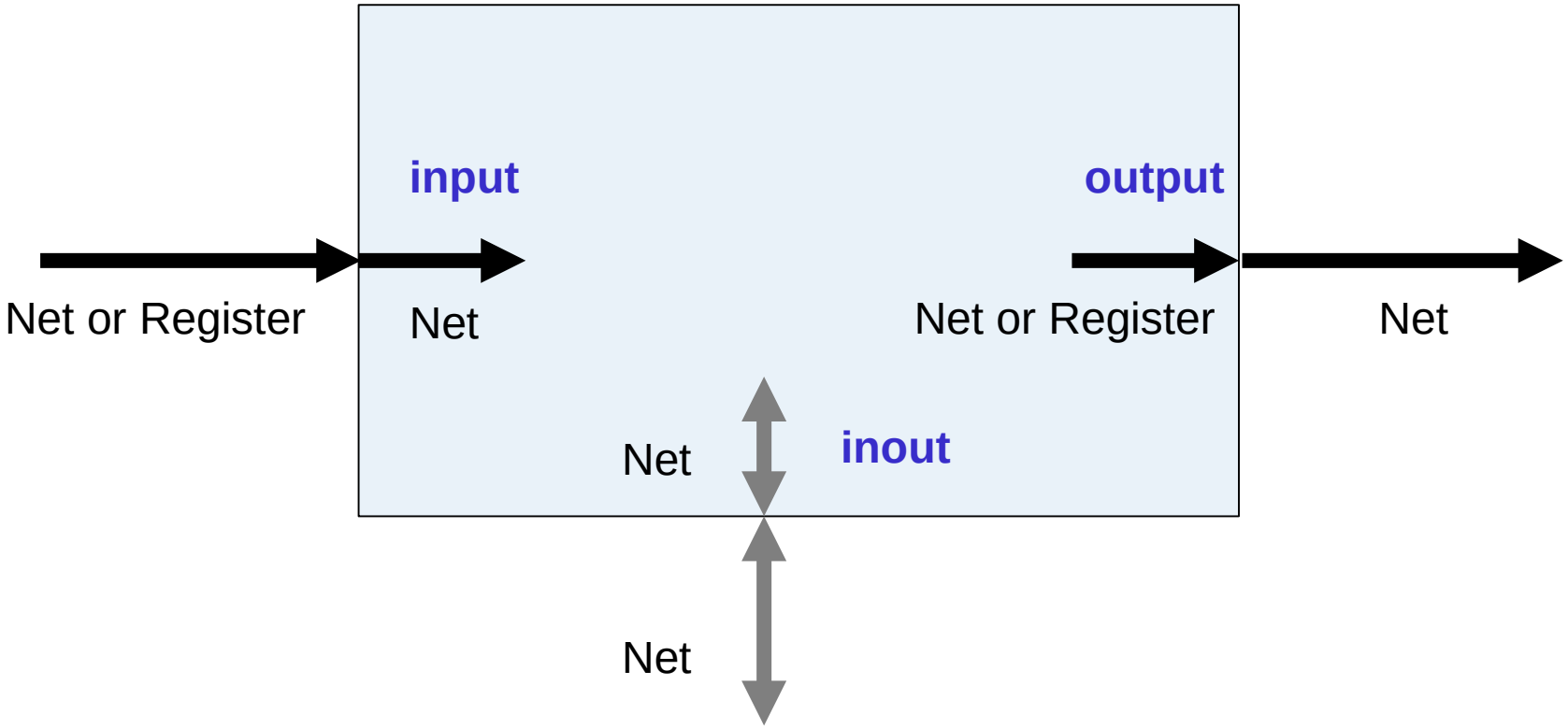
```
module circuit0(A, B, C);

    input A, B;
    output C;
    wire D;

    assign D = (A & B) | (~A & ~B);
    assign C = D | ~D;
endmodule
```

Verilog – 模組(module) (2/3)

- I/O ports 的相連規定



Verilog – 模組(module) (3/3)

- 新增一個模組

模組名稱 代稱(I/O ports);

```
circuit0 c0(A,B,C);

//或是

circuit0 c0(  .A(A),
              .B(B),
              .C(C)  );
```

實作題

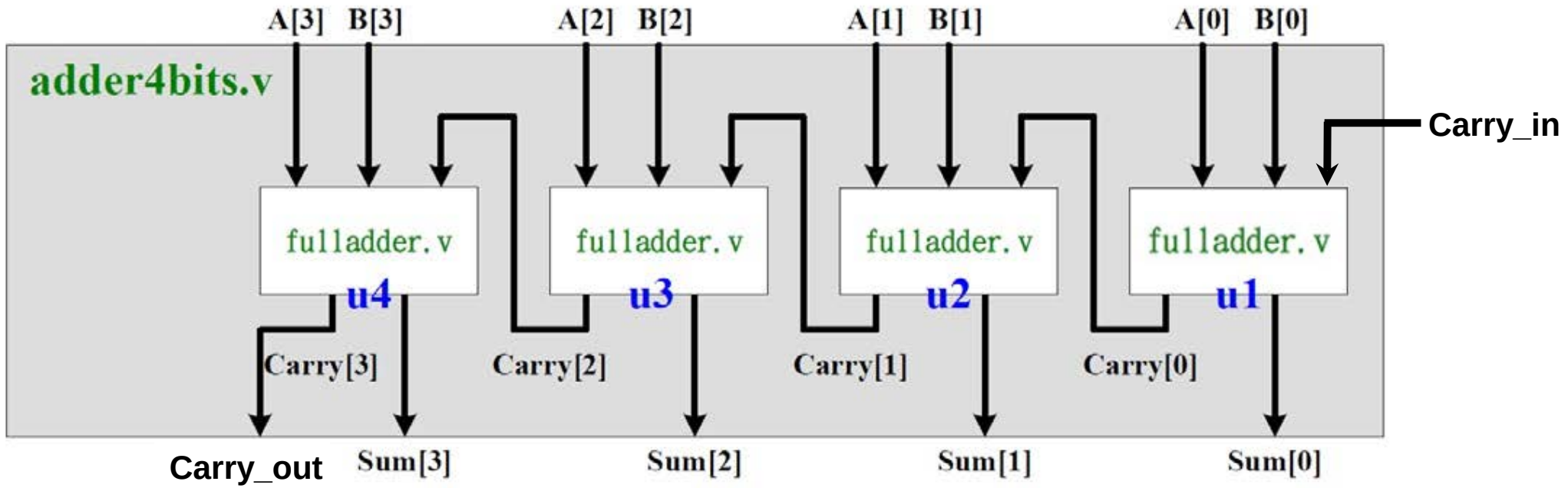
本次實作分為兩個基本實作與一個挑戰實作

實作題(一): 4-bit加法器(1/7)

- 實作4-bit ripple-carry adder漣波進位加法器並且驗證

實作題(一): 4-bit加法器(2/7)

- 我們可以利用全加器組出該電路



實作題(一): 4-bit加法器(3/7)

步驟:

1. 建立 **C:\logiclab\<學號>\lab5_1** 資料夾。
2. 依上面投影片所示，建立一個 project。
3. 在ISE開啟新的Verilog檔案，並把名稱設為**adder_4bits.v**，儲存在上述資料夾。

實作題(一): 4-bit加法器(4/7)

4. 利用全加器組合出4-bit加法器

請同學先鍵入下列程式碼，並且將空白處填完後，再按一次存檔

```
module adder4bits(A,B,Carry_in,Sum,Carry_out);

    input [3:0]     A,B;
    input   Carry_in;
    output [3:0]    Sum;
    output          Carry_out;

    wire [3:0]      Carry;

    fulladder u1(A[0],B[0],Carry_in,Carry[0],Sum[0]);
    fulladder u2(A[1],B[1],Carry[0],Carry[1],Sum[1]);
    fulladder u3(A[2],B[2],Carry[1],Carry[2],Sum[2]);
    fulladder u4(                                   );

    assign Carry_out = Carry[3];

endmodule
```

實作題(一): 4-bit加法器(5/7)

5. 在ISE再開啟新的Verilog檔案，並把名稱設為**testbench.v**，儲存在上述資料夾。

6. 鍵入右邊程式碼，再按一次存檔

```
module testbench();

    reg [3:0]a,b,
    reg cin;
    wire [3:0]sum;
    wire cout;

    adder4bits n1(a,b,cin,sum,cout);

    initial begin
        a = 3; b = 5;   cin = 0;
    #1  a =10; b = 2;   cin = 0;
    #1  a =15; b = 15;  cin = 1;
    #1  a = 0; b = 0;   cin = 0;
    #1 $finish;
    end

endmodule
```

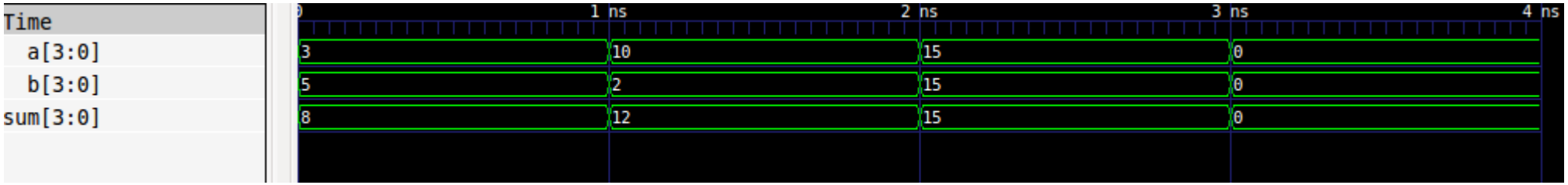
實作題(一): 4-bit加法器(6/7)

7. 依投影片所示，將**adder4bits.v**和**testbench.v**都加入project。
8. 依投影片所示，將上一個實驗的**fulladder.v**加入project 。
9. 依投影片所示開始編譯檔案，Compiler後會顯示是否出現Errors、Warnings，若有Error發生，可直接用滑鼠點錯誤訊息兩 下即可直接跳至錯誤處。

實作題(一): 4-bit加法器(7/7)

9. 驗證結果是否符合預期。

依投影片所示，執行電路模擬並觀察波形，如下圖。

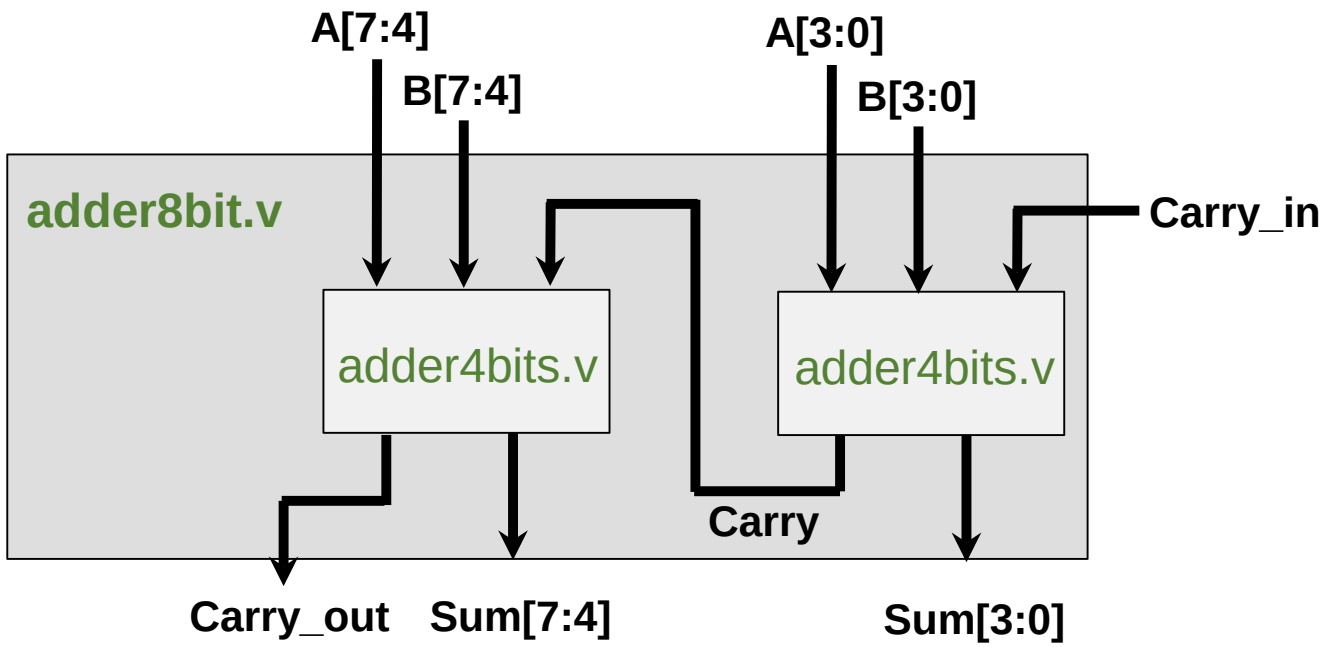


實作題(二): 8-bit加法器(1/7)

- 實作8-bit ripple-carry adder漣波進位加法器並且驗證

實作題(二): 8-bit加法器(2/7)

- 我們可以利用4-bit ripple-carry adder快速地組出該電路



實作題(二): 8-bit加法器(3/7)

步驟

1. 建立 **C:\logiclab\<學號>\lab5_2** 資料夾。
2. 依投影片所示，建立一個project。
3. 在ISE開啟新的Verilog檔案，並把名稱設為**adder8bits.v**， 儲存在先前lab5_2的資料夾。

實作題(二): 8-bit加法器(4/7)

4. 利用4-bit加法器組合出8-bit ripple-carry adder加法器

請同學先鍵入下列程式碼，並且將空白處填完後，再按一次存檔

```
module adder8bits(A,B,Carry_in,Sum,Carry_out);

    input [7:0] A,B
    input   Carry_in;
    output [7:0] Sum;
    output Carry_out;

    wire Carry;

    adder4bits u1(                    );
    adder4bits u2(                    );

endmodule
```

實作題(二): 8-bit加法器(5/7)

4. 在ISE再開啟新的Verilog檔案，並把名稱設為**testbench.v**， 儲存在上述資料夾。

5. 鍵入右邊程式碼，再按一次存檔。

```
module testbench();

    reg [7:0]a,b,
    reg cin;
    wire [7:0]sum;
    wire cout;

    add8bits n1(a,b,cin,sum,cout);

    initial begin
        a =15; b =16;   cin = 1;
    #1  a =25; b =30;   cin = 0;
    #1  a =31; b =12;   cin = 1;
    #1  a = 0; b = 0;   cin = 0;
    #1 $finish;
    end

endmodule
```

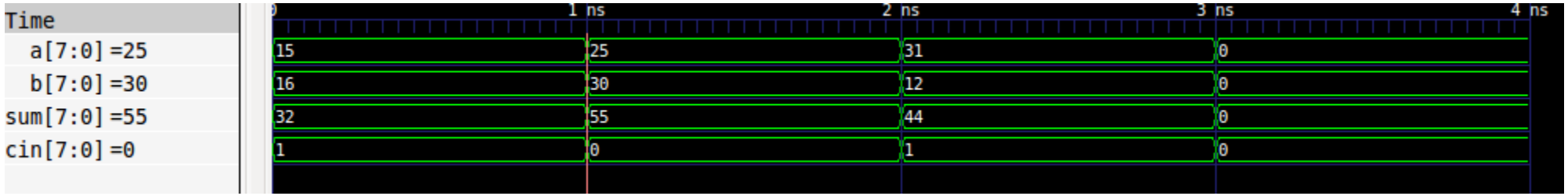
實作題(二): 8-bit加法器(6/7)

6. 依投影片所示，將**adder8bits.v** 和**testbench.v**都加入project。
7. 依投影片所示，將上一個實驗的**adder4bits.v**與**fulladder.v**都加入project 。
8. 依投影片所示開始編譯檔案，Compiler後會顯示是否出現Errors、Warnings，若有Error發生，可直接用滑鼠點錯誤訊息兩 下即可直接跳至錯誤處。

實作題(二): 8-bit加法器(7/7)

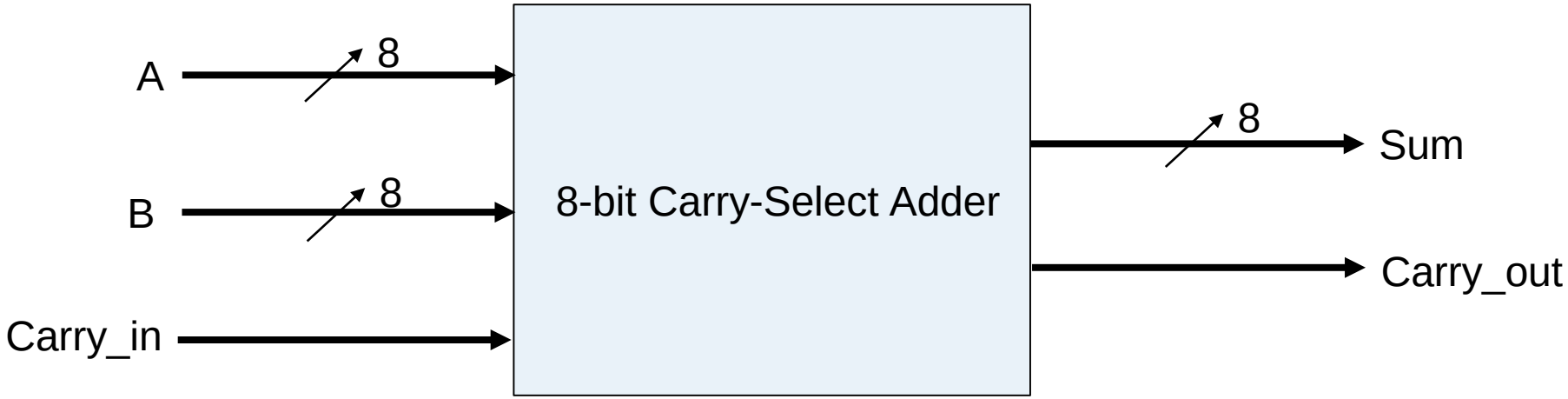
8. 驗證結果是否符合預期。

依投影片所示，執行電路模擬並觀察波形，如下圖。



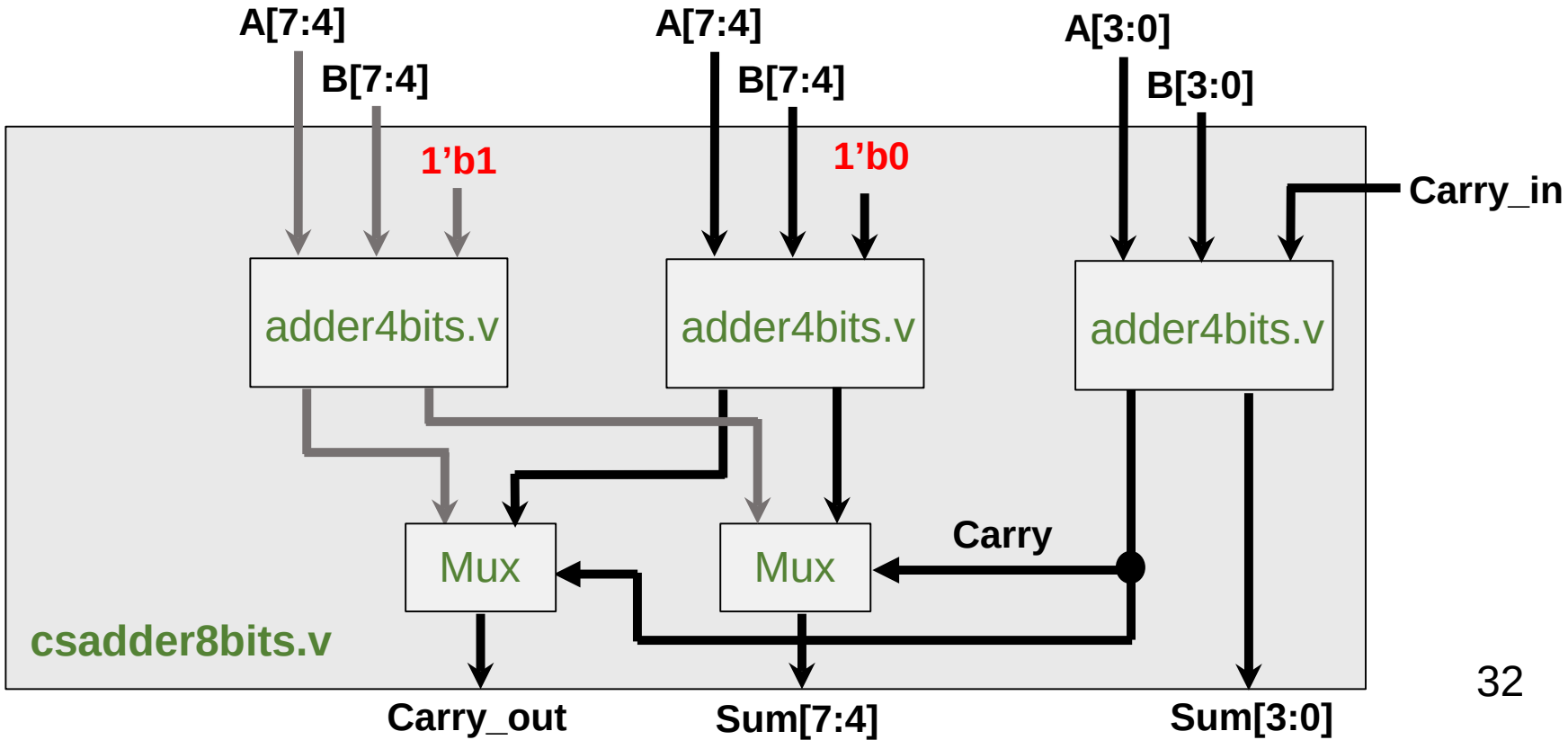
挑戰題(一): 8-bit CSA(1/4)

- 實作8 bit carry-select adder
  - 優點: 運算速度比8 bit ripple-carry adder更快
  - 缺點: 需要使用更多的邏輯閘



挑戰題(一): 8-bit CSA(2/4)

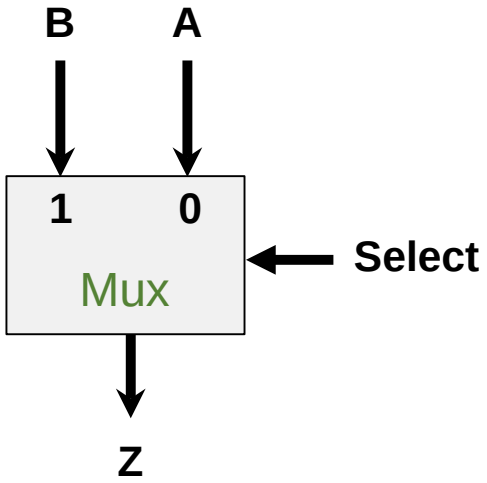
- 該加法器在計算低四位時，高四位預先把所有可能的結果算完。等到低四位算完後，直接選取對應的高四位，進而達到加速運算的效果



挑戰題(一): 8-bit CSA(3/4)

- 多工器(Mux Multiplexer)或稱資料選擇器
  - 用於從多個訊號中選擇一個訊號
  - Verilog中可以使用三元運算子 “?:” 當作多工器
    - 以下式為例，當 Select 為 0 時，z會選到 A
    - 反之，當Select為1時，z會選到B

```
assign z = Select ? B : A;
```



挑戰題(一): 8-bit CSA(4/4)

步驟

1. 建立 **C:\logiclab\<學號>\lab5_3** 資料夾。
2. 請使用ISE將上圖用Verilog實現。
3. 參考實作(二)，設計一個testbench。
4. 將檔案Compile及Simulation，並檢查模擬解果是否符合預期。

Verilog參考資料

- Verilog 硬體描述語言(Verilog HDL)第二版
  原著:Samir Palnitkar
  原出版社: Prentice Hall
  編譯: 黃英叡, 黃稚存, 張銓淵, 江文啟
  全華科技圖書

- Verilog HDL: A Guide to Digital Design and Synthesis," 2nd ed. by Samir Palnitkar, Prentice Hall, 2003. ISBN: 0-13-044911-3

- http://www.asic-world.com/verilog