

FPGA

FPGA

數位設計實驗

課程須知

- 一、本實驗課採分組方式進行，兩人一組，同共完成當週實驗，並一起撰寫一份報告。
- 二、每次上課前點名，缺席扣 1 次分數，遲到扣 0.5 次分數，出席成績佔 12%
- 三、每次實驗完成後，均需請助教檢查並登記，實驗佔 20%
- 四、每個實驗完成，均須繳交一份報告佔 18%（不含期中及期末報告），報告內容包含：
 1. 題目名稱
 2. 功能說明：有哪些輸入訊號，有哪些輸出訊號，整個電路的功能為何？
 3. 硬體的架構圖，可參考講義給的架構圖來修改，模組與模組之間的連結必須標出名稱，且使用的名稱必須和程式碼所使用的名稱相同。輸入、輸出腳位必須標示硬體所連結的 **pin number**。並且利用硬體架構圖來簡單說明電路設計的想法。
 4. 程式碼
 5. 程式註解：每一行都須要說明這一行的功用，如果單獨一行無法說明功能時，則對一個區塊做說明。宣告的部份也是每一個變數單獨宣告，並說明這個變數的用途
 6. 如果有心得討論更好
- 五、實驗 7 與實驗 11 分別須繳交一份期中、期末報告，報告內容如上，再加上心得與問題討論，心得的部份，需分別各自撰寫自己的心得。各佔 10%，共 20%。
- 六、學期中時會抽籤決定期末專題的題目。
- 七、學期末有期末考，每個人獨自寫自己的試卷，不可參考任何資料，佔 30%，考試內容為：
 1. 依題目撰寫一個小程序
 2. 題目給一個語法有錯的程式，找出其中的錯誤並改正
 3. 題目給一個漏寫幾行的程式，依題目功能補上正確程式碼

目錄

課程須知.....	1
目錄.....	2
實驗 1 Verilog 簡介.....	4
一、 Verilog 概述.....	4
二、 建立專案.....	5
三、 實作.....	11
實驗 2 全加器.....	12
一、 模組化設計.....	12
二、 匯流排.....	13
三、 實作.....	14
實驗 3 無號數乘法器.....	16
一、 連結運算子.....	16
二、 數值表示法.....	16
三、 實作.....	17
實驗 4 算術邏輯單元.....	19
一、 Data Flow Model	19
二、 Behavioral Model.....	19
三、 條件敘述.....	21
四、 實作.....	23
實驗 5 除頻器.....	26
一、 Blocking 與 Non Blocking	26
二、 消除彈跳.....	26
三、 模擬.....	27
四、 實作.....	28
實驗 6 跑馬燈.....	30
一、 實作.....	30
實驗 7 計時器.....	32
一、 task	32
二、 七段顯示器掃描電路.....	32
三、 實作.....	33
實驗 8 有限狀態機.....	35
一、 Mealy-type FSM	35
二、 Moore-type FSM.....	36
三、 實作.....	36
實驗 9 以 Gate Level 設計有限狀態機.....	38
一、 以 Gate Level 設計流程	38
二、 FSM 化簡.....	39

三、實作.....	41
實驗 10 自動販賣機.....	42
實驗 11 期末專題.....	43
一、題目一：貪吃蛇.....	43
二、題目二：二十一點.....	43
三、題目三：擲骰子.....	44

實驗1 Verilog 簡介

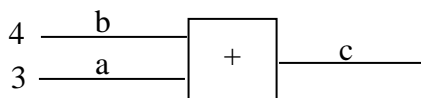
一、Verilog 概述

Verilog 是一種硬體描述語言，它用來描述一個邏輯電路中，不同的元件是如何連線。雖然它的語法與 C 語言類似，但不像 C 語言一樣，在執行時是一行一行的執行下來，前面的程式碼先執行，後面的程式碼後執行。例如

```
b=4;  
a=3;  
c=a+b;
```

```
c=a+b;  
b=4;  
a=3;
```

這兩種寫法，在 C 語言中是不相同的。但是在 Verilog 中，這兩種寫法是相同的，同樣都表示



雖然結果都相同，但習慣上還是使用左邊的寫法，比較能理解整個電路的運作。既然是描述一個邏輯電路，自然會使用到一些邏輯閘。在 Verilog 的程式語法中，邏輯閘是以函數的名稱、輸出與輸入來描述，例如，輸入為 x_1 、 x_2 ，輸出為 y 的兩輸入 AND 邏輯閘，其 Verilog 語法寫成 `and(y,x1,x2)`。而比較複雜的邏輯電路可以用模組（module）的方式來寫，類似 C 語言中的副函式。模組裡面可以再使用模組，而最外層的模組即類似於 C 語言中的主程式。使用模組的好處是具有結構化，以及模組可重複使用。一個模組包含了定義此電路的敘述，包括輸入與輸出的埠(port)。如圖 1-1 所示，左邊的邏輯電路可由右邊程式碼描繪出來。第一行為一個模組的開始，**module** 為保留字宣告模組的開始，**example1** 表示該模組的名稱，而括號內的 x_1 、 x_2 、 x_3 、 f 則為此模組的 I/O Port，記得最後要加分號。第二、三行為宣告哪些 I/O Port 是輸入或輸出，**input** 與 **output** 均為保留字，後面接的名稱，例如 x_1 、 x_2 ，必需列在 **module name** 後的括弧中。第四行宣告所使用的 g 、 k 、 h 為 **wire** 的型態，可省略不宣告。第五行至第九行為模組的電路描述，當模組的描述結束後，則要以 **endmodule** 作為模組結束的描述，但不要加分號，如最後一行。要注意的是，我們可以用如 C 語言的方式，「//」雙斜線來作單行註解，/* 及*/來作區塊的註解，以增加程式的可讀性。

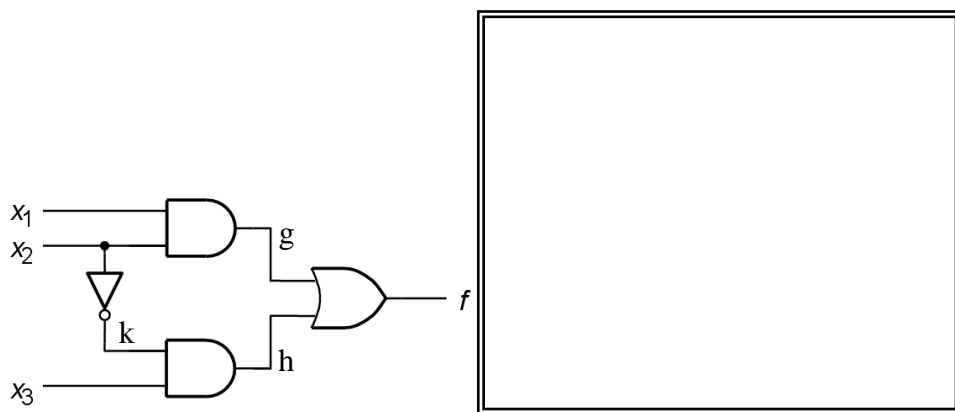


圖 1-1 模組範例

當程式完成後，必須將程式燒錄至 FPGA 晶片上。此時，要對模組的每個 I/O Port 指定

對應的晶片腳位。如圖 1-2 所示，當模組燒錄至 FPGA 晶片內時，會將 Module 的 x1 接到晶片的第 47 腳，x2 接到晶片的第 65 腳，x3 接到晶片的第 79 腳，f 接到晶片的第 7 腳。當訊號由晶片外部經 47、65、79 這三個輸入腳位傳到模組時，經過我們設計的模組電路，產生輸出訊號。然後輸出訊號會經 7 這個輸出腳位傳送到晶片外部。至於要使用哪些腳位，需與晶片外部的電路互相配合。由於這門課是使用教學板，晶片外部的電路已固定接線，所以在使用上需參考教學板上的腳位設定。教學板的腳位請參考講義最後的附錄 1。

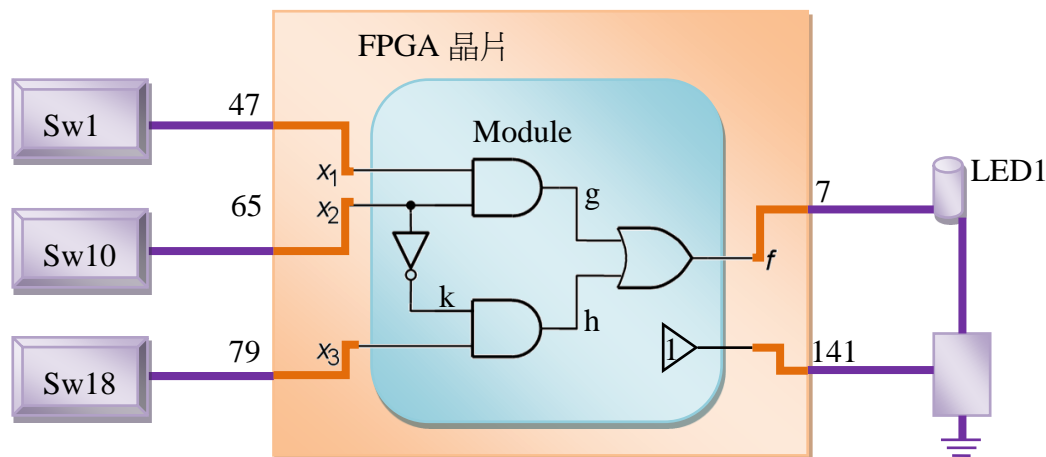
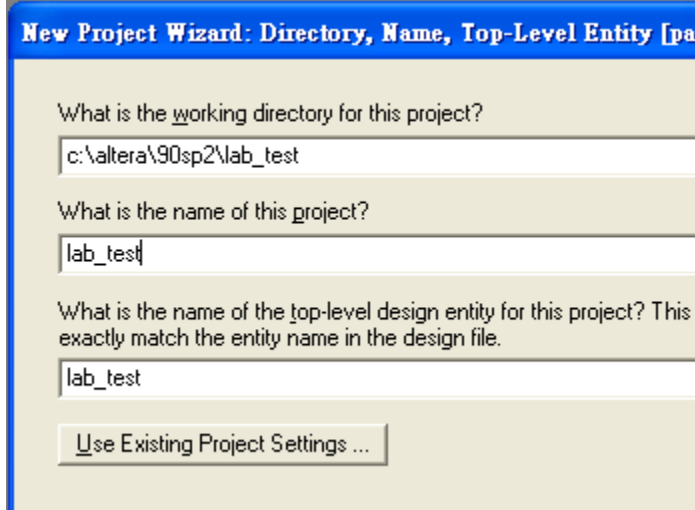
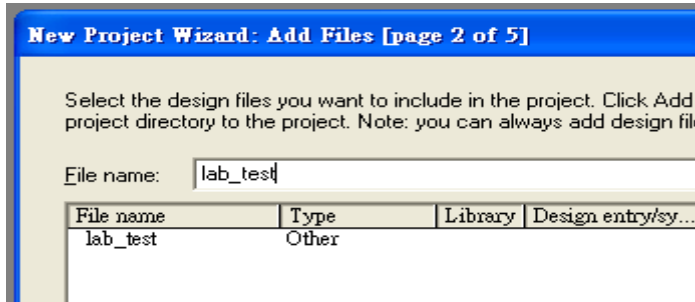
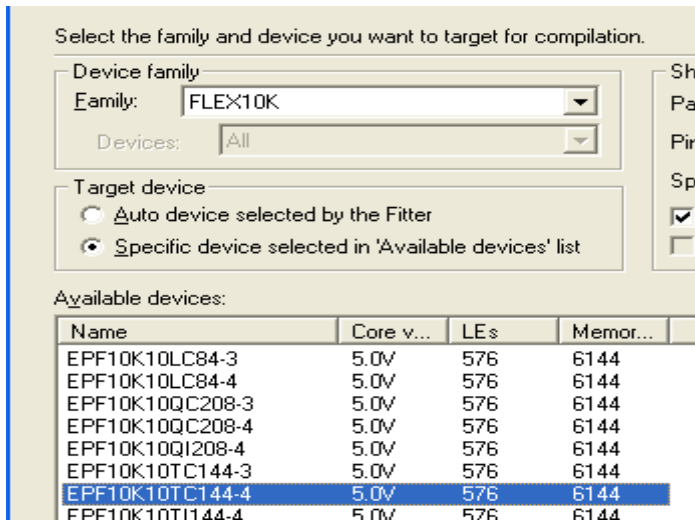
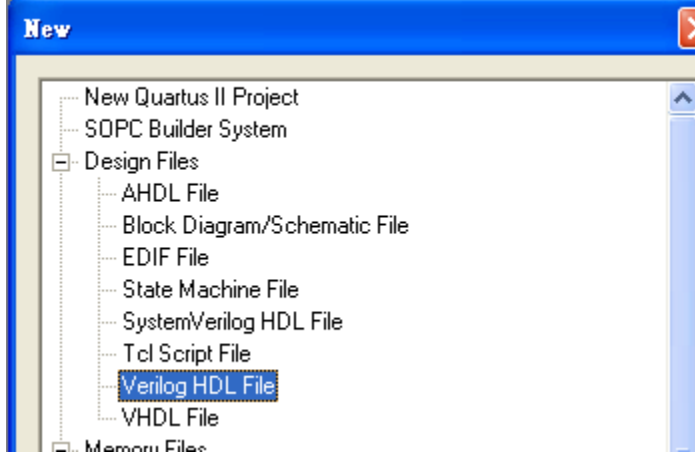


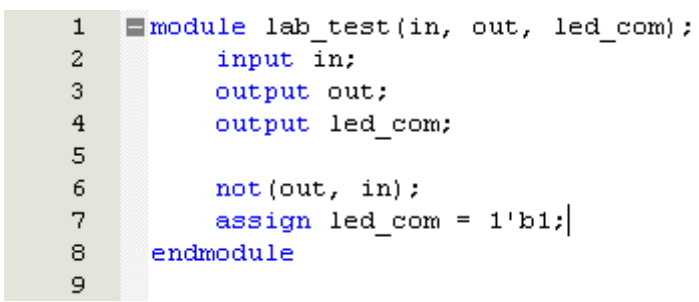
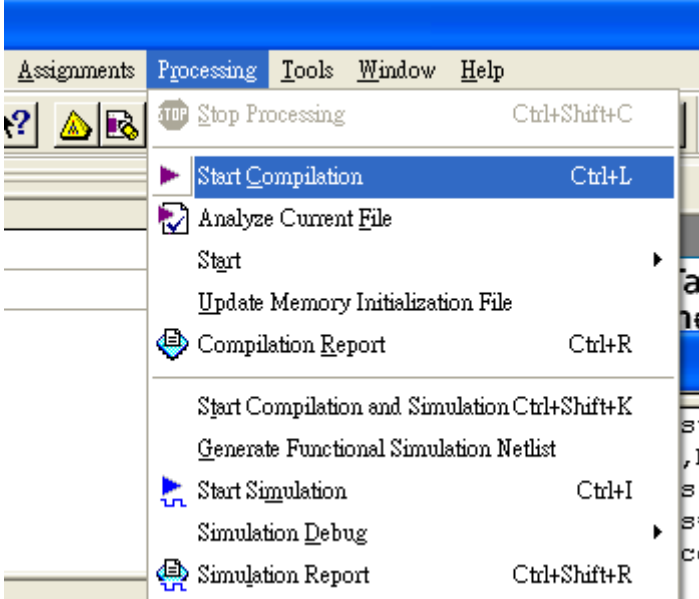
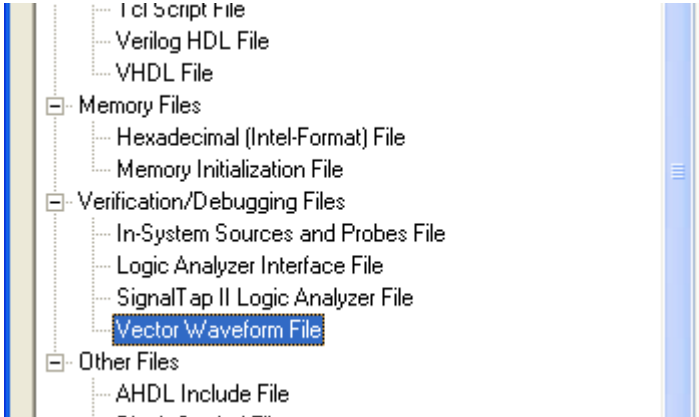
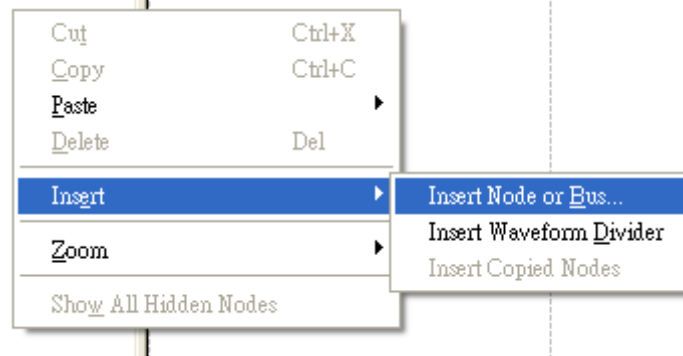
圖 1-2 晶片腳位連結示意圖

二、建立專案

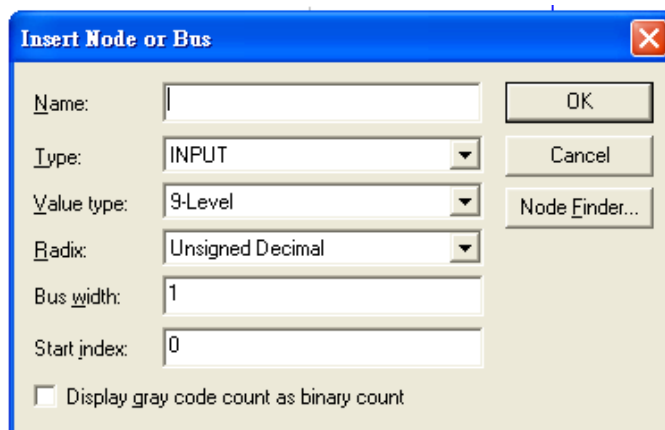
接下來將一步一步指示如何建立一個專案，撰寫程式，執行模擬，燒錄程式。以後的課程都將按照這些步驟來完成。

<p>Step1. 在主選單上按「File」/「New Project Wizard」，會出現一個新增專案的對話窗。</p>	
--	--

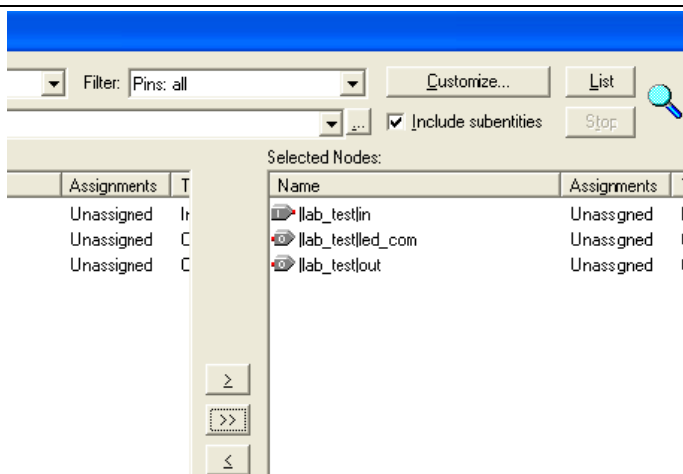
<p>Step2.分別輸入專案資料夾、專案名稱，最上層模組名稱，建議三者名稱均相同，例如 lab_test。注意！最上層模組名稱不可用數字當開頭，其餘字母可英文字、數字、底線混用，與一般變數的命名規則相似。</p>	
<p>Step3.這步驟可省略不填，或是在「File name」填入上一步所填的名稱。填完後記的要按「Add」才真的有加入。</p>	
<p>Step4.Family 選擇 Device family 名稱 FLEX10K，然後 Available device 選 EPF10K10TC144-4。接下來的設定都是按下一步。也可在此直接按 Finish 來結束設定。</p>	
<p>Step5.在主選單上按「File」/「New」，此時會出現一個對話窗，選擇「Verilog HDL File」。然後就會出現一空白文件，可以開始寫程式了。</p>	

<p>Step6.在文件上編寫 Verilog 語言。注意,最上層的 module name 必需和 Step2. 所輸入的名稱相同。程式寫完後記得存檔。</p>	 <pre> 1 module lab_test(in, out, led_com); 2 input in; 3 output out; 4 output led_com; 5 6 not(out, in); 7 assign led_com = 1'b1; 8 endmodule 9 </pre>
<p>Step7. 在選單按上「Processing」/「Start Compilation」。如果有錯誤訊息,則修正程式碼,如果只有警告訊息,則可忽略。至此,完成程式的編輯工作,接下來設定模擬波形。</p>	
<p>Step8.在選單上按「File」/「New」,此時會出現一個對話窗,選擇「Vector Waveform File」。然後就會出現設定波形的文件。</p>	
<p>Step9. 在左欄空白處點右鍵,會彈出選單,選擇「Insert」/「Insert Node or Bus」。會出現一個對話窗。</p>	

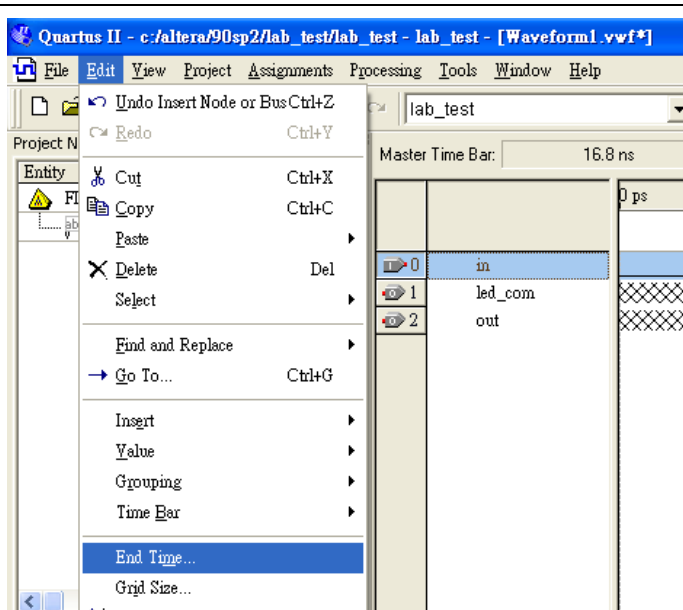
Step10.按下「Node Finder」後，會再出現另一個對話窗。



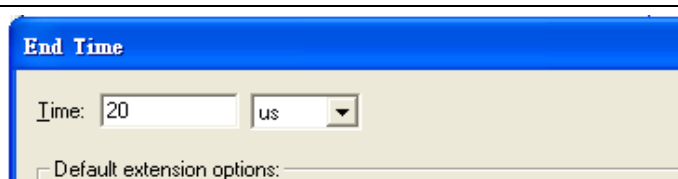
Step11.按下「List」後會在左邊欄列出所有的 I/O Port。再按下「>>」鍵會將所有的 I/O Port 選取至右邊欄。然後按「OK」回到波形視窗。



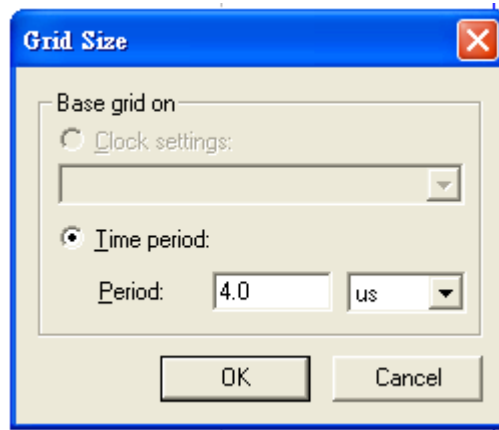
Step12. 在選單上按「Edit」/「End Time」，會出現設定模擬終止時間的對話窗。



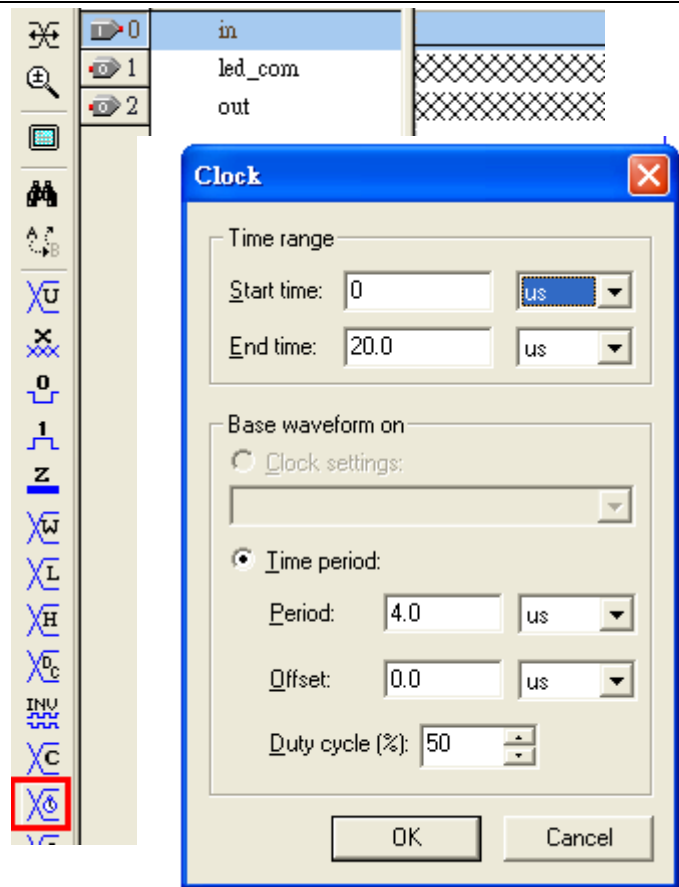
Step13.單位請務必選擇 us，然後設定適當的 End Time。再按「OK」。



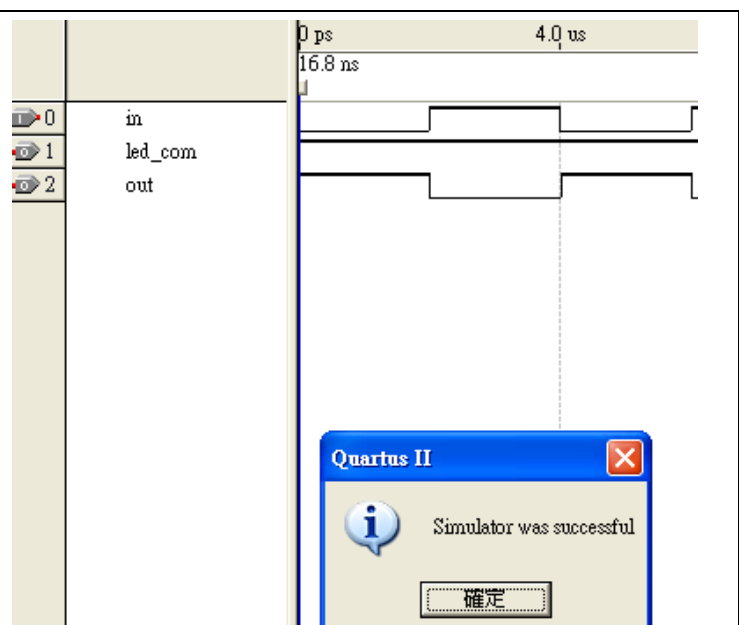
Step14.如 Step12.一樣，在選單上按「Edit」/「Grid Size」，會出現設定 Grid Size 的對話窗。單位請務必選擇 us，然後設定適當的 Time period。再按「OK」。注意，如果選 ns 的話，在設定波形時，可能會因輸入波形變化太快，而又因邏輯閘的延遲時間，導致模擬結果錯誤。



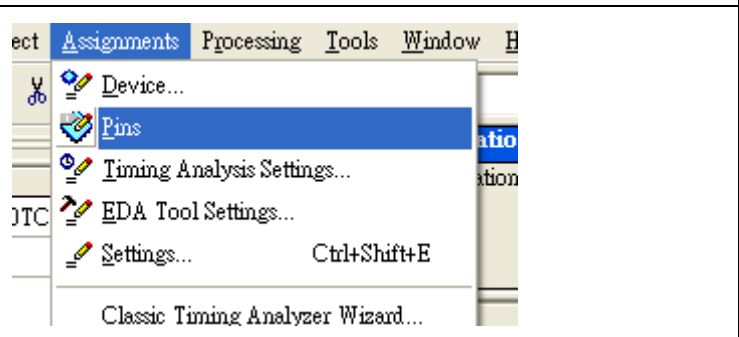
Step15.點選要設定的 Input Port，例如 in。然後在工具列上選擇適當的波形，例如，按下紅色框框的按鈕即為設成 Clock。按不同的按鈕，如需要進一步設定，則會出現對話窗做設定。如設定 Clock 視窗，輸入適當的 Clock 開始及結束時間，以及適當的週期。注意，週期若太小，可能會因訊號變化太快，再因邏輯閘的延遲時間，導致模擬結果錯誤。注意！當所有的 Input Port 都設定好波形後，必須先將此波形檔存在專案資料夾內，檔案名稱與專案名稱須相同。




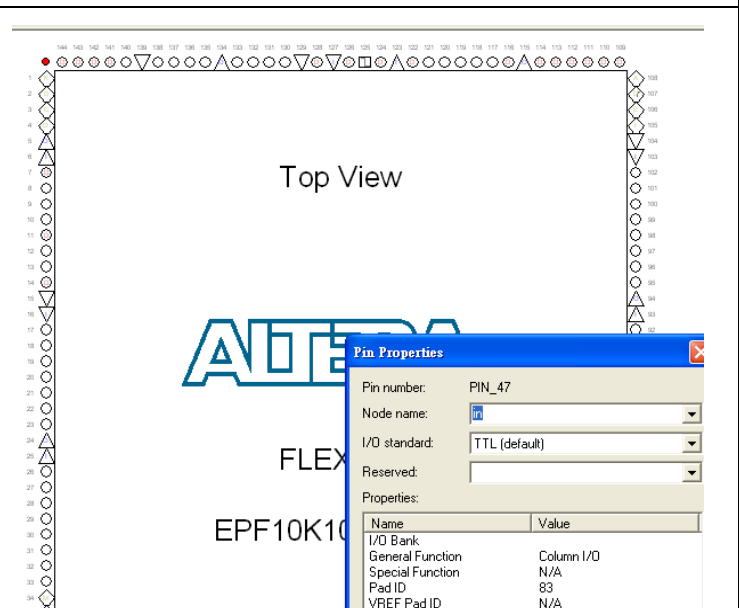
Step16. 如 Step7. 一樣，在選單按上「Processing」/「Start Simulation」開始模擬。檢視模擬結果，如果有誤，則回到 Step6.修改程式，直到模擬正確。至此則完成模擬結果，接下來要設定 I/O Port 連結到實體腳位，並將程式燒錄到 FPGA 教學板上。



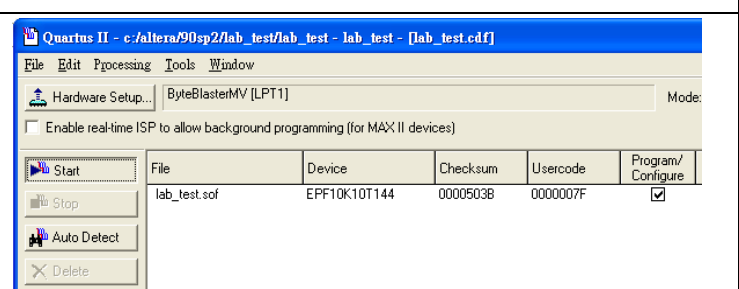
Step17. 在選單按上「Assignments」/「Pins」，會出現設定腳位的檔案。



Step18. 選擇要設定的 Pin 腳，點選兩下，會出現對話窗。在「Node name」選擇對應的 I/O Port 的名稱。依此方式將所有的 I/O Port 設定至相應的腳位。再將此設定腳位的視窗關閉。然後再依 Step7. 重新在選單按上「Processing」/「Start Compilation」一次。最後在工具列上點選 ，會出現燒錄的視窗。



Step19. 點選「Start」即開始燒錄，燒錄完成後，即可在教學板上驗證程式是否正確。如果在「Hardware Setup」沒有出現「ByteBlasterMV[LPT1]」的文字，則按「Hardware Setup」去選擇正確的選項。



通常為了減少錯誤的發生，建議在 Step2.輸入**專案名稱**時，不要去更改它自動設定的**最上層模組名稱**。然後在寫程式時，把**最上層的模組名稱**跟**檔案名稱**都設成與專案名稱一樣的名字。然後建立波形檔後儲存時，**波形檔名稱**也設成跟專案名稱相同。

三、實作

以下有個需求，有三開關要來控制房間內的同一光源是否要點亮或熄滅，其真值表如表 1-1 所示，以 Sum of Products 方式所設計之電路如圖 1-3 所示。請參考圖 1-1 之範例程式，完成圖 1-3 之電路，並以波形模擬器做檢查是否正確。程式撰寫與模擬的流程請參考上述建立專案之步驟，模擬參數之設定參考值為 End Time=8us, Grid Size=1us, Time Range=8us。而輸入波形設定用「Overwrite Clock」，x1~x3 的「Starting Value」皆為 0，x1 的週期為 2us，x2 的週期為 4us，x3 的週期為 8us。模擬正確後，將該程式燒錄至 LP2900，腳位設定請參考附錄一。x1-x3 用 Data switch 來設定，f 用 LED 當輸出，記得要設定 LED 的共陰接腳，即晶片的第 141 腳位的值為 1。完成後請助教檢查確認其狀態運作是否符合真值表。

表 1-1 $f = x1 \oplus x2 \oplus x3$ 真值表

x1	x2	x3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

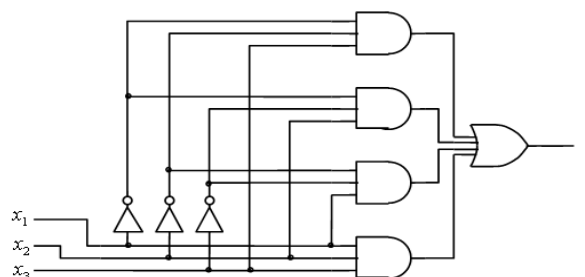


圖 1-3 Sum of Products realization

實驗2 全加器

在設計電路時，經常會有重複出現的電路區塊，例如設計 4 位元全加器時，每個位元的全加器視為一個區塊，則此區塊內的電路其實是相同的。此時，可將這重複出現的區塊設為一個模組。如此在設計更大電路時，即可重複使用這些小模組，以便簡化程式，同時也可以對模組再利用。此外，利用模組的方式，也可以讓程式看起來更結構化，方便除錯。使用模組時，程式的部分看起來就像呼叫副程式一樣。對應到電路的部分，就是在大電路中插入一個區塊。本實驗即利用模組化的設計方式來設計一個 4 位元的全加器。

一、模組化設計

圖 2-1 即為模組化設計之示意圖，藍色為較小的模組，名稱為 M1，有兩個輸入腳，一個輸出腳，橘色為最上層模組，名稱為 M2，有三個輸入腳，一個輸出腳。在 M2 這個模組裡，放了兩個 M1 模組，分別命名為 b1 與 b2。b1 的 out 經由 a 這條導線連結到 b2 的 in1，x1 接到 b1 的 in1，x2 接到 b1 的 in2，x3 接到 b2 的 in2，f 接到 b2 的 out。這樣的電路以 Verilog 來描述即可寫成圖 2-1 右邊的程式碼。b1 跟 b2 分別是 M2 這個模組在使用 M1 模組時，對這個模組給的名稱，也可省略不寫。使用模組時，輸出入埠的對應方式有兩種，一種是指定輸出入埠的方式，也就是括號裡的參數必須指定模組定義的名稱，順序未必要跟模組定義的順序相同。另一種是依模組定義之順序來連接，括號裡的參數順序，必須和模組定義的順序相同，也就是說，它會由左至右一條一條的對應連接。以圖 2-1 為例，b1 這個區塊為使用指定輸出入埠的方式，x1 跟 x2 分別會連接到 b1 的 in1 跟 in2，順序不必依定義 M1 時的順序先寫 out，再寫 in1 跟 in2。b2 這個區塊則依模組定義的順序來寫，所以可省略 in1，in2，out 這些定義的名稱，f 會自動連到 out，a 跟 x3 分別會自動連到 in1 跟 in2。由於 M2 為最上層晶片，所以它的 I/O 腳位必須指定連結到 FPGA 晶片的腳位。在此圖例中，x1、x2、x3 分別連到第 47、48、49 三個腳位，可用按鈕開關來當輸入訊號，f 連到第 7 腳位可用 LED 來當輸出腳位。注意！這裡省略了 LED 的共陰腳沒設定，實際操作時必須設定，否則 LED 不會亮。

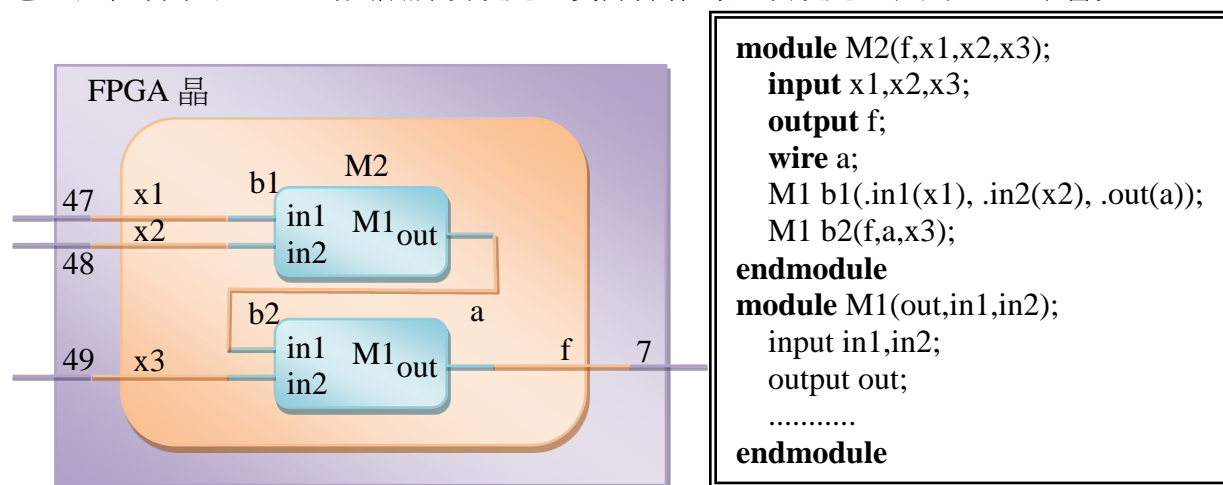


圖 2-1 模組化設計示意圖

不同模組可以寫在同一個檔案內，也可以一個模組寫成一個檔案，只要確定所有用到的檔案都有放進專案裡，且最上層的模組名稱與專案設定的最上層模組名稱相同即可。可參看 0，建立專案的 step2。亦可如圖 2-2 所示，先按「Assignments」/「Settings...」，出現 Settings

對話視窗。在 Settings 對話視窗左邊點選「Files」，確認所有用到的檔案都有加在右邊的列表裡。然後在 Settings 對話視窗左邊點選「General」，然後在右邊的「Top-level entity：」欄位輸入最上層模組的模組名稱。

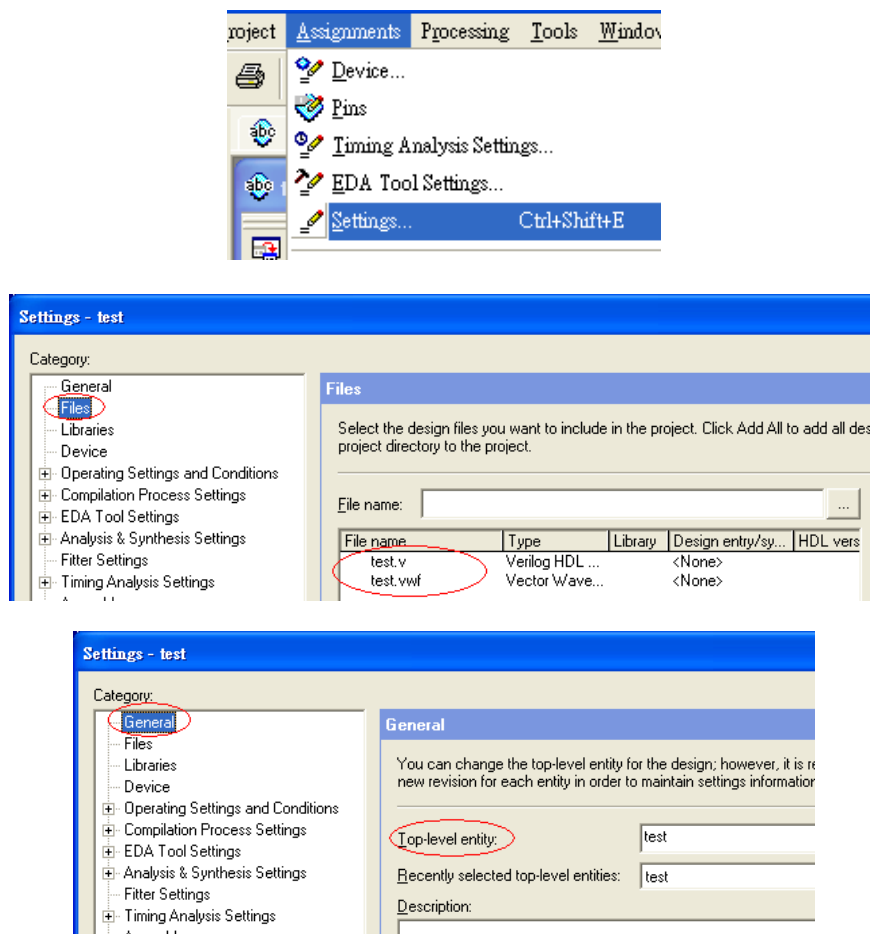


圖 2-2 設定最上層模組

二、匯流排

在之前的例子中，用來連結不同的邏輯閘或模組之間，所使用內部的連線，我們都宣告為 **wire**。如果這些連線有很多條，且功能相似，例如八位元記憶體的資料線就有八條連線，為了方便描述，我們把這些功能相似的連線合起來看，稱為匯流排。在計算機裡面就有所謂的資料匯流排、位址匯流排。在 Verilog 的語法中，也有匯流排的方式來描述，首先看看宣告的方式：

`type [m:n] bus_name;`

`type` 就是這個匯流排的类型，例如 **output**，**input**，**wire**...等，`m` 跟 `n` 為匯流排編號的最大值與最小值，`bus_name` 就是匯流排名稱，可以放多個名稱，用逗號隔開。例如

wire [5:3] a, b;

表示有兩個匯流排 a 跟 b，每個匯流排各有 3 條線，分別為 a[3]，a[4]，a[5]，b[3]，b[4]，b[5]，在習慣上，n 設為 0。切記不要寫成

wire [4:0]a,[4:0]b;

當宣告 **module** 時使用匯流排做為輸出與輸入時，在 **module** 的括弧內只要寫匯流排名稱，

而不需要給編號數目，在 **input** 與 **output** 後的宣告才給匯流排編號數目。在使用上，如果只寫匯流排名稱，表示使用整個匯流排的所有編號，如果要指定匯流排的單一條線，可用方刮號裡面放匯流排的某個編號。如果要指定其中的某幾條線，則在方刮號裡放結束編號與起始編號，兩編號中用冒號隔開，例如圖 2-3 所示。

```
module bus_example(a,b,out);
  input [7:0]a,b;    //宣告 a,b 為兩個輸入匯流排，各有 8 條線，編號從 0 到 7
  output [7:0]out;    //宣告 out 為 8 位元的輸出匯流排，wire 可省略不寫
  wire [7:0]c;
  assign c=a;
  assign out[0]=c[0]; //將 c[0]的值指定給 out[0]，也就是將 c[0]跟 out[0]連起來
  assign out[4:1]=b[7:4] //指定 out[4]=b[7], out [3]=b[6], out [2]=b[5], out [1]=b[4]
endmodule

module test(in,out);
  input [15:0]in;
  output [7:0]out;
  bus_example(in[7:0],in[15:8],out);
endmodule
```

圖 2-3 匯流排使用範例

三、實作

請利用 Gate Level 的方式，先完成一個一位元的全加器模組，然後再利用二個全加器，合成一個二位元加法器，最後再用二個二位元加法器，合成一個三層模組之四位元加法器，如圖 2-4。描述 4bits 與 2bits 加法器的 I/O Port，以及使用模組時，請使用匯流排的方式來表示。FPGA 的腳位請參考附錄一。在設定模擬的波形檔時，請設定適當的 End Time 及 Grid Size，然後將兩個 4bits 輸入分別群組起來，輸出的 cout 與 sum 群組起來，以方便觀察結果。注意 cout 與 sum 群組時，是在最高位元。輸入波形可以隨機設定的方式來產生。

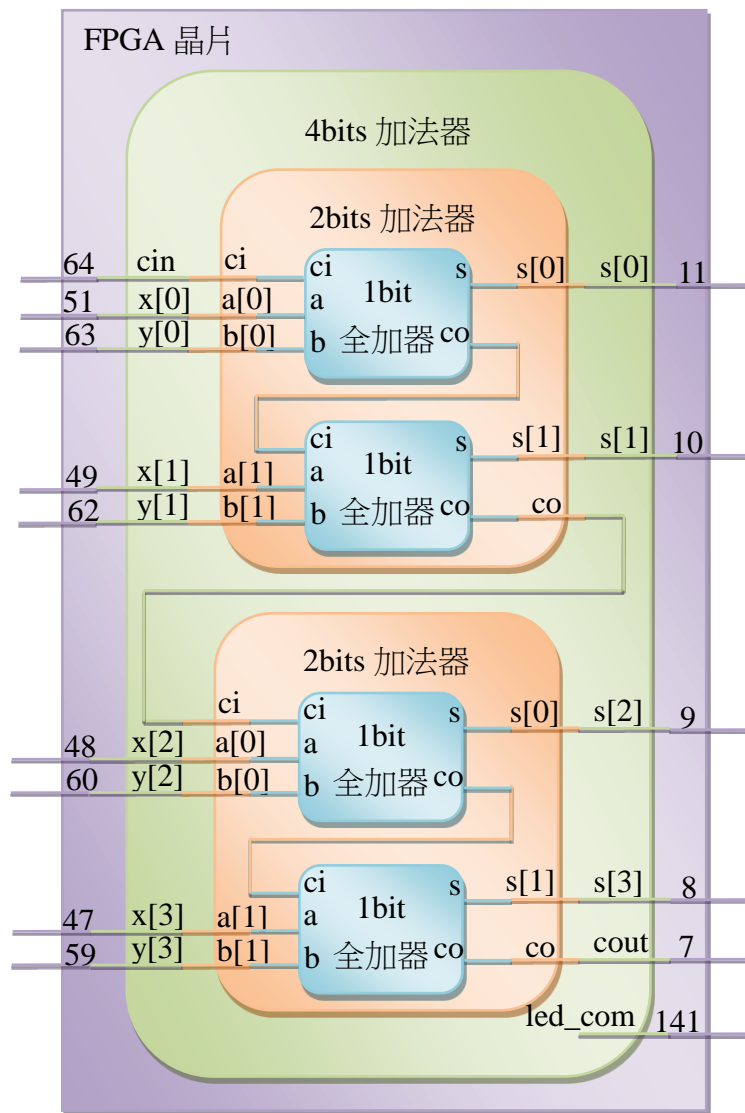


圖 2-4 三層模組之四位元全加器

實驗3 無號數乘法器

本實驗旨在利用實驗 2 所教授的內容，實作出一無號數之乘法器，以便同學能有更深刻的瞭解。此外，本實驗將介紹連結運算子的使用方法。

一、連結運算子

在實驗 2 中我們介紹了匯流排的用法，本實驗所要介紹的連結運算子即為專門處理匯流排的運算子。所謂的連結運算子，就是可以指定某些連線或是某些匯流排的某部分，合成一個匯流排指定給另一個匯流排，如圖 3-1 為連結運算子的例子。左邊在第四行中表示 $c[7:4]=a[3:0]$, $c[3]=x$, $c[2:1]=b[1:0]$, $c[0]=y$ 。右邊第三行為呼叫模組時，使用連結運算子的例子

<pre>wire [3:0] a,b; wire [7:0] c; wire x,y; assign c={a[3:0],x,b[1:0],y};</pre>	<pre>wire [3:0] a,b; wire x,y; test({a,b},{x,y}); module test(a,b); input [7:0] a; output [1:0]b;</pre>
--	---

圖 3-1 連結運算子範例

二、數值表示法

在 Verilog 中表示數值，必須指出是幾位元，以幾進制來表示這個數字，整個數字格式如下

$[sign]<size>'<base><value>$

sign 表示正負號，如果是正，則省略不寫，如果是負，則寫上-。*size* 表示這個數字是幾位元的數字，需要注意的是，如果數值所需的位元數大於 *size* 數，則此數值無法正確呈現，但不會有錯誤訊息。所以 *size* 要設成表示數值所需的位元數，或是更大。*base* 為數值所採用的進制，b 表示二進制，o 表示八進制，d 表示十進制，h 表示十六進制。*value* 則是以 *base* 的進制系統所呈現的數值，數值的表示中，除了進制所允許的數字外，還可以有 x, z, _。x 表示未知數。在三態輸出中，除了 0 與 1 之外，還有一種狀態為高阻抗，對電路來說，就是斷路的意思，z 就是設成高阻抗。_通常用在二進制，多位元的情形，只是用來區隔，增加可讀性。在之前的實驗中，每次都有一行指令，`assign led_com=1'b1;`就是指定 led_com 這條線的值是 1bit 的 1。底下是幾個數值表示的範例。

1'b1	1bits，二進制，值為 1
- 8'd3	8bits，十進制，值為-3
12'hz	12bits，十六進制，值為 z，代表每 bit 都接到高阻抗
6'bx	6bits，二進制，值為 x，代表每 bit 都是未知
8'b10	8bits，二進制，值為 2，也就是 0000 0010 ₂
8'b11	8bits，二進制，值為 3，也就是 0000 0011 ₂

12'b1111_0010_1011	12bits，二進制，值為 3883，也就是 $1111\ 0010\ 1011_2$ ，位元如果太多時，可以用底線區隔來增加可讀性，不影響數值
12'h13x	12bits，十六進制，以二進制來看，表示 0001 0011 xxxx

三、實作

請參考下述之計算式，完成一個 4bits 乘 4bits 的無號數乘法器。由計算式可以看出，整個乘法器是由一些 AND 閘與全加器所組合而成。

$$\begin{array}{r}
 \begin{array}{cccc}
 & A3 & A2 & A1 & A0 \\
 \times & B3 & B2 & B1 & B0 \\
 \hline
 & A3B0 & A2B0 & A1B0 & A0B0 \\
 A3B1 & A2B1 & A1B1 & A0B1 & \\
 \hline
 & A3B2 & A2B2 & A1B2 & A0B2 \\
 + & A3B3 & A2B3 & A1B3 & A0B3 \\
 \hline
 P7 & P6 & P5 & P4 & P3 & P2 & P1 & P0
 \end{array}
 \end{array}$$

在此提供兩種方法來完成本電路，一種是將電路分解成一群 AND 邏輯閘跟 1bit 全加器模組。觀察上述之計算式，可以得到如圖 3-2 之電路圖。如果再仔細觀察這些電路，會發現此電路可分成橘色及藍色兩種模組，差別在於橘色模組的兩個輸入都是連到 AND 閘，而藍色模組只有一個連到 AND 閘，另一個輸入則為另一個模組的進位。於是我們可以改寫 1bit 全加器，將 AND 邏輯閘併入到 1bit 全加器裡面。

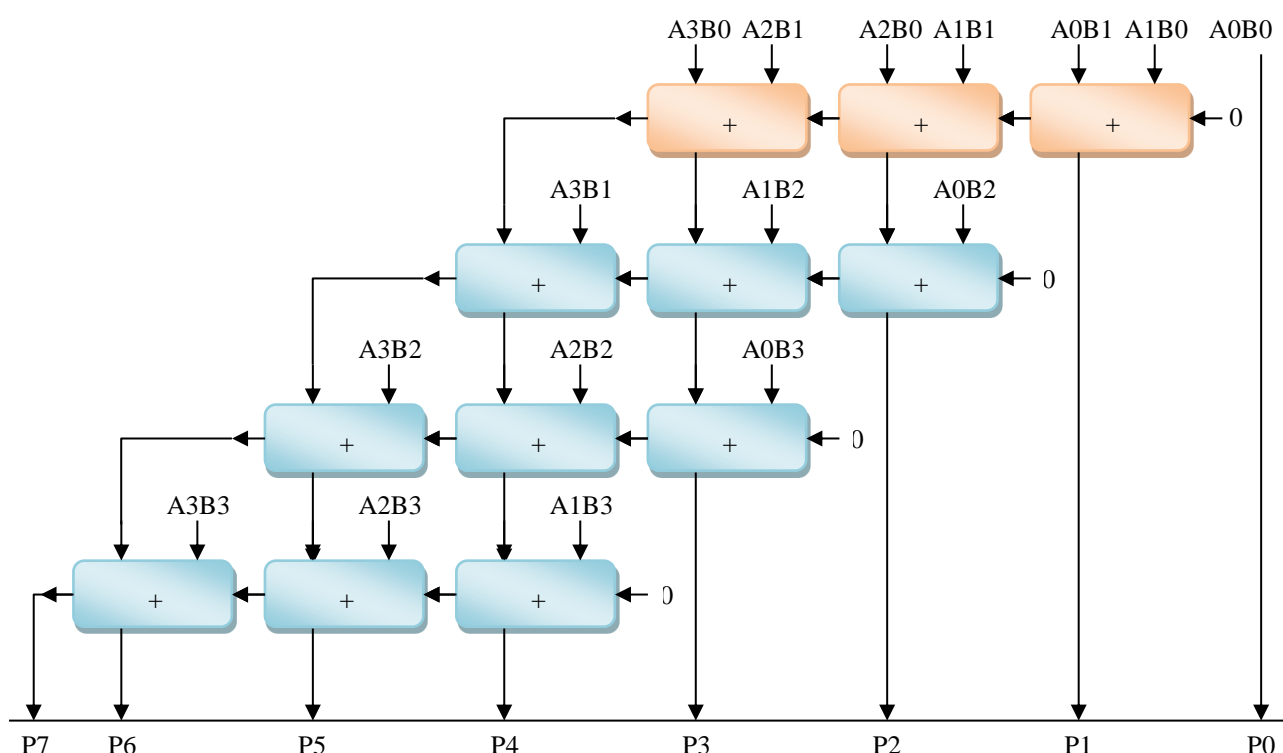


圖 3-2 使用 1bit 加法器之乘法電路

另一種做法則是利用 4bits 全加器來做，而 4bits 全加器又可再如實驗 2 由四個 1bit 全加

器來完成。觀察上述計算式，可得到如圖 3-3 之電路圖，在設定全加器之輸入、輸出腳位時，可配合連結運算子來描述會比較方便。需注意的是，此 4bits 加法器與實驗 2 之加法器略有不同，它沒有 carry in 的 input。模擬時請如實驗 2 之方式將兩組輸入分別群組，並以隨機方式設定波形。輸出也是自行群組起來，同時為增加可讀性，請將群組的 format 設成 unsigned decimal。

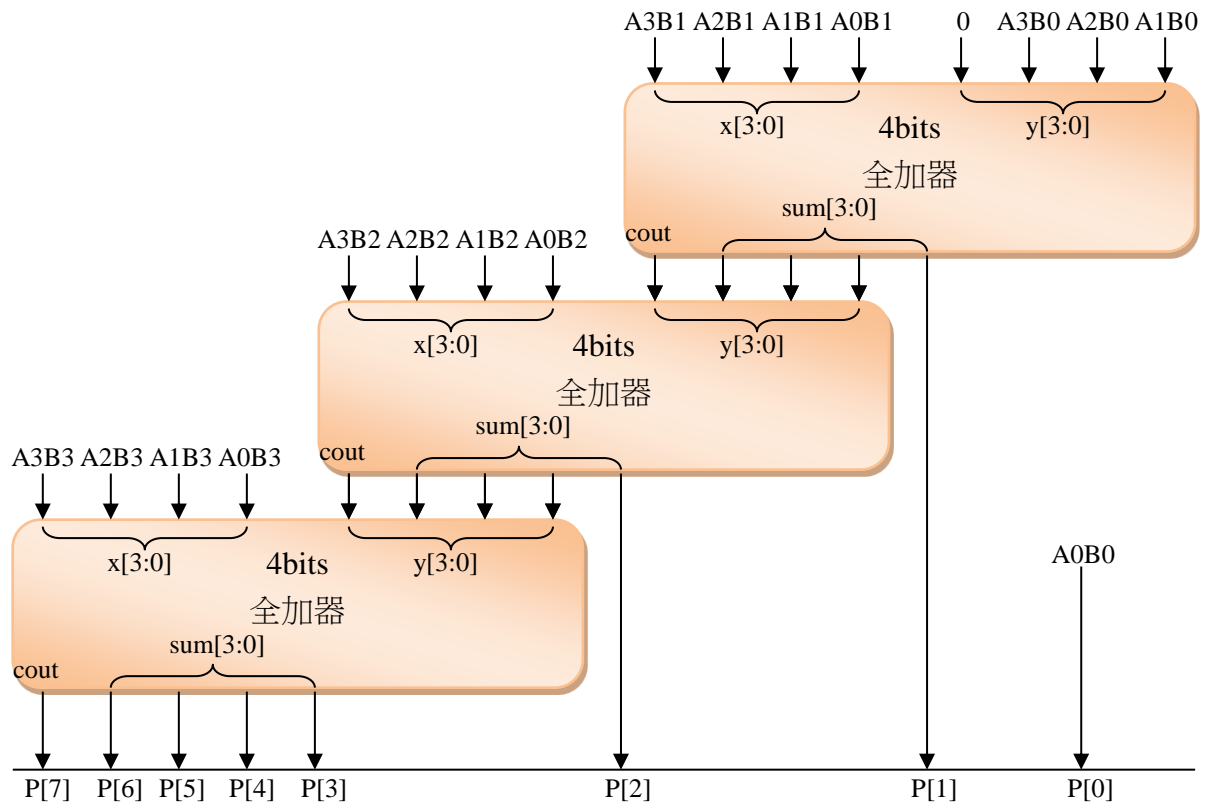


圖 3-3 使用 4bits 加法器之乘法電路

實驗4 算術邏輯單元

之前的實驗已介紹 Gate Level Model 的方式來設計電路，本實驗將介紹另二種方式，即 Data Flow Model 與 Behavioral Model 的方式來設計電路。同時也介紹如何使用條件敘述語法，並利用條件判斷來實現算術邏輯單元。也就是說，根據使用者所選擇的輸入，來完成不同的運算式。運算子的符號請參考表 4-1。

一、Data Flow Model

在 0 介紹的方式是屬於 Gate Level Model，也就是描述邏輯閘之間如何連接。本實驗再介紹第二種較簡便的描述方法，就是 Data Flow Model，也就是描述資料在電路中如何傳送的過程。這種模型的描述方式與一般的 C 語言的語法很相似。

以圖 1-1 的電路為例，若以 Data Flow Model 來寫，可寫成如圖 4-1 之程式碼。其中除了第五行與之前不同外，其餘的與圖 1-1 的 Gate Level Model 的寫法一樣。在第五行中，用一些運算子的表示方式來取代邏輯閘，除了這幾個運算子外，Verilog 還提供了許多運算子，如表 4-1。這些運算子的使用方法，基本上與 C 語言類似，詳細使用方式，請參考課本，在此不再舉例說明。使用 Data Flow Model 的描述方式時，必須注意要用 **assign** 這個保留字，來描述將等號右邊的結果指定給等號左邊的線路名稱。

```
module example(x1,x2,x3,f,led);  
  input x1,x2,x3;  
  output f,led;  
  assign f=(x1&x2)|(-x2&x3);  
  assign led=1'b1;  
endmodule
```

圖 4-1 以 Data Flow Model 撰寫圖 1-1 之電路圖

表 4-1 運算子列表

運算子型態	運算子	說明	運算子型態	運算子	說明
算術運算子	+、-、*、/ %	算術加、減、乘、除 取餘數	位元邏輯運算子	~、&、 、^ ^~或~^	not、and、or、xor nxor
邏輯運算子	!、&&、	not、and、or	移位運算子	<<、>>	左移、右移
關係運算子	>、>= <、<= ==、!= ===、!==	大於、大於或等於 小於、小於或等於 等於、不等於 狀況等於、狀況不等於	精簡邏輯運算子	&、~& 、~ ^ ^~或~^	and、nand or、nor xor xnor
條件運算子	?:	條件指定	連結運算子	{}	連結

二、Behavioral Model

Behavioral Model 的描述方式，與 Data Flow Model 的描述方式除了幾個差異點之外，其

餘都一樣。第一個差異點是使用了 **always** 這個關鍵字，也就是說，在 **always** 這個範圍內的敘述，都是屬於 Behavioral Model。而不在 **always** 這個範圍內的敘述，則是屬於 Data Flow Model 的方式。再以圖 1-1 之例，這個電路圖若是以 Behavioral Model 是描述，則如圖 4-2 所示。我們先看第五行，在 **always** 這個保留字後，先打上@這個符號，意思表示當@後面括弧內的條件成立時，底下的 Behavioral Model 的敘述會被執行。在這個範例中，@後面括弧內的條件是 **x1 or x2 or x3**，表示 **x1** 或 **x2** 或 **x3** 的值，至少有一個改變。這種條件的表示方式是一種準位觸發，也就是說它的值由 0 變成 1 或是由 1 變成 0 時條件成立，當這三個值其中至少有一個改變了，**f** 的值就要重新計算。此外，條件也可以採用邊緣觸法來表示，可分成正緣觸發跟負緣觸發。正緣觸發用 **posedge** 這個保留字，當訊號由 0 上升至 1 時條件成立。負緣觸發用 **negedge** 這個保留字，當訊號由 1 變 0 時條件成立，必須注意準位觸發與邊緣觸發不能混用。當條件成立後，**always** 底下這一行會被執行，也就是如第六行所寫的。如果 **always** 的範圍超過一行的話，就需要用 **begin** 和 **end** 這兩個保留字來設定。在這裡又有一個與 Data Flow Model 差異的地方，就是沒有 **assign** 這個保留字。也就是說，Behavioral Model 的描述方式是不使用 **assign** 這個保留字的。因為 **always** 的範圍只有緊接著的下一行，也就是第六行，所以第七行就屬於 Data Flow Model 的描述方式，必需使用了 **assign** 這個保留字。第三個差異在第四行，使用 **reg** 這個保留字來定義 **f**。**reg** 這個保留字是一種抽象的資料儲存元件，它可以保留一個數值直到下一次指定新值為止。另外必須注意的是，在 **always** 裡的敘述，以 Blocking 的方式來指定時，其先後順序是有影響的，也就是說，先寫 **a=b**;再寫 **b=c**;與先寫 **b=c**;再寫 **a=b**;是有差異的，詳細的說明於實驗 5 再介紹。總結這幾個差異如下：

1. 用 **always** 保留字。
 2. **always** 裡的敘述不用 **assign**
 3. **always** 裡，等號左邊的要宣告成 **reg**
 4. **always** 裡的敘述，其順序必須正確
 5. **always** 裡不可以使用 **module**，只能用 **task**
- task** 將在下次實驗介紹

```
module example(x1,x2,x3,f,led);  
  input x1,x2,x3;  
  output f,led;  
  reg f;  
  always@(x1 or x2 or x3)  
  begin  
    f=(x1&x2)|(-x2&x3);  
    g=x1 |x2;  
    assign led=1'b1;  
  endmodule
```

圖 4-2 以 Behavioral Model 撰寫圖 1-1 之電路圖

三、條件敘述

所謂的條件敘述，就是當某個條件成立的時候，某個電路會被執行，或是當某個條件成立的時候，某條件會被指定成某個值。例如多工器，選擇線選擇了某條輸入訊號，指定給輸出線就是用條件敘述來完成。在此我們介紹三種條件敘述，第一種是條件運算子，這種運算子可用於 Data Flow Model 及 Behavioral Model。可以把它想像成一個 2 對一的多工器，也就是說有兩個輸入訊號 x_1 與 x_2 ，根據 s 的值來決定輸出 y ，如果 $s=1$ ， $y=x_1$ ，否則 $y=x_2$ 。寫法為 $y=s?x_1:x_2$ ；如果是 4 對 1 的多工器，則可使用巢狀的方式來寫： $y=s_0?(s_1?x_1:x_2):(s_1?x_3:x_4)$ ，其電路圖如圖 4-3 所示。

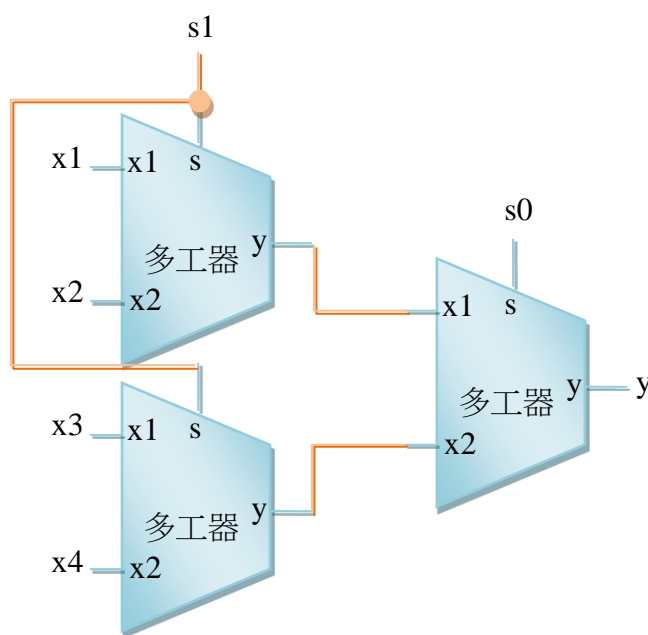


圖 4-3 巢狀之條件運算子示意圖

第二種與第三種分別是 **if** 條件敘述和 **case** 選擇敘述，這兩種都必須使用 Behavioral Model，而不可以寫在 Data Flow Model，也就是說，要寫在 **always** 裡面，其語法分別圖 4-4 與圖 4-5 所示。在圖 4-4 中，條件表示式的結果為 **true** 或 **false**，當結果為 **true** 時，會進行運算式 1 的電路，反之會執行運算式 2 的電路。如果沒有運算式 2，則 **else** 省略不寫。此外，運算式 1 與運算式 2 只能為一行敘述，如果要描述的電路超過一行的話，要用 **begin** 及 **end** 來設定。

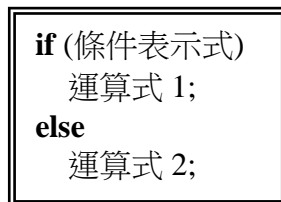


圖 4-4 **if** 的語法

case 的使用方式與 **if** 相似，在圖 4-5 中，值計算式會計算出一個值，然後根據計算出來的值與值 1 至值 n 依序做比對，如果比對的值與值 j 相同，則會進行運算式 j 的電路，否則會進行運算式 $n+1$ 的電路。如果沒有運算式 $n+1$ ，則 **default** 省略不寫。運算式如果超過一行，必須用 **begin** 及 **end** 來設定。使用 **case** 必須要注意值計算式結果的位元數是否與底下條列的

值位元數相同。

```
case (值計算式)
  值 1:begin
            運算式 1;
  值 2:運算式 2;
  .....
  值 n:運算式 n;
default:運算式 n+1
```

圖 4-5 case 的語法

在此我們以圖 4-6 來說明其使用之方法，第一行至第四行如果不懂，請參考以前講義。第六行的 **always** 即表示底下的部分是以 Behavioral Model 來描述，所以等號的左邊必須定義成 **reg**，如第五行所寫。**always** 後面的括號內表示，當 a、b、s 三個訊號如果其準位有改變，也就是說值有改變，那麼某個等號左邊的線當值也會改變，在此即是 out 會跟著改變。第七行先判斷 s 的值為 true 或 false。因為 s 只有 1bit，所以如果 s 為 1 表示為 true，則做加法，否則 s 為 0 表示 false，做 and 運算。

```
module add_and(a,b,s,out);
  input [1:0] a,b;
  input s;
  output [6:0] out;
  reg [6:0] out;
  always@(a or b or s)
    if (s)
      case (a+b) //abcd_efg
        3'd0:out=7'b1111_110;
        3'd1:out=7'b0110_000;
        3'd2:out=7'b1101_101;
        3'd3:out=7'b1111_001;
        3'd4:out=7'b0110_011;
        3'd5:out=7'b1011_011;
        3'd6:out=7'b1011_111;
        default:out=7'b0011_111;
      endcase
    else
      case (a[0]&b[0]);
        1'b0:out=7'b1000111;
        1'b1:out=7'b0001111;
      endcase;
  endmodule
```

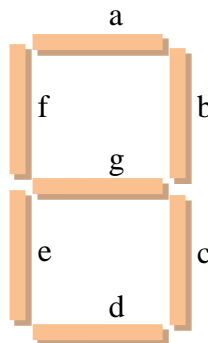


圖 4-6 if 及 case 之範例

第八行至第十七行為 **if** 條件成立時要做的電路，第八行是一個 **case** 敘述，括號裡面的計算式值是 $a+b$ ，因為 a 、 b 均為 2bits，加起來後最多有 3bits，所以接下來的條例值都是以 3bits 來比對。第九行至第十五行是 $a+b$ 算出來後的可能值，為 0 至 6，但因為 3bits 總共有 0 至 7 的可能，所以我們在第十六行加上 **default**，表示如果值不為 0 至 6 的其他情形。這幾行的 out 輸出為七段顯示器的輸出，值的設定請參考七段顯示器的編號。第十九行至第二十二行為 **if** 條件不成立時要做的電路，也就是做 $a[0]\&b[0]$ 。由於第十九行的 **case** 括號裡的計算結果為 1bit，且底下的條例值為 0 跟 1，已把所有情形條列出來，所以就不需要再寫 **default** 敘述了。












四、實作

撰寫一個具有兩輸入 A 與 B （每一輸入有 4bits），並以 S （2bits）當做選擇功能訊號的 ALU，其可選擇的功能如表 4-2 所示，詳細說明如後。所有輸出以 5 個 LED 及 1 個七段顯示器來呈現。 $S=00$ 時用四個 LED 顯示每一位元的運算結果，並以七段顯示器分別顯示 A ，表示此時執行 AND 運算。 $S=01$ 至 11 時，以七段顯示器顯示計算結果的個位數，以 5 個 LED 採二進制顯示十位數及百位數。七段顯示器的符號設定如表 4-3，其腳位請參考附錄 1。

表 4-2 ALU 功能表

S[1:0]		功能敘述	輸出訊號
0	0	$A \& B$	七段顯示器顯示 A ，用 5 個 LED 顯示結果
0	1	$A+B$	七段顯示器顯示個位數，用 5 個 LED 以二進制顯示十位數
1	0	$A \ll 1$ (乘 2)	七段顯示器顯示個位數，用 5 個 LED 以二進制顯示十位數
1	1	$A * B$	七段顯示器顯示個位數，用 5 個 LED 以二進制顯示十位數

表 4-3 七段顯示器之符號設定

0	1	2	3	4	5	6	7	8	9	A
										

七段顯示器之解碼電路如圖 4-7 所示，六個七段顯示器之 ABCDEFG 七個輸入腳位分別都接在一起，當對 ABCDEFG 七個腳位設定值時，六個七段顯示器同時收到相同的訊號，此時還必需設定 DE1-DE3 的值，透過 3 對 8 解碼器來指定哪個七段顯示器要亮，未指定到的七段顯示器雖然同樣收到 ABCDEFG，但依然不顯示。

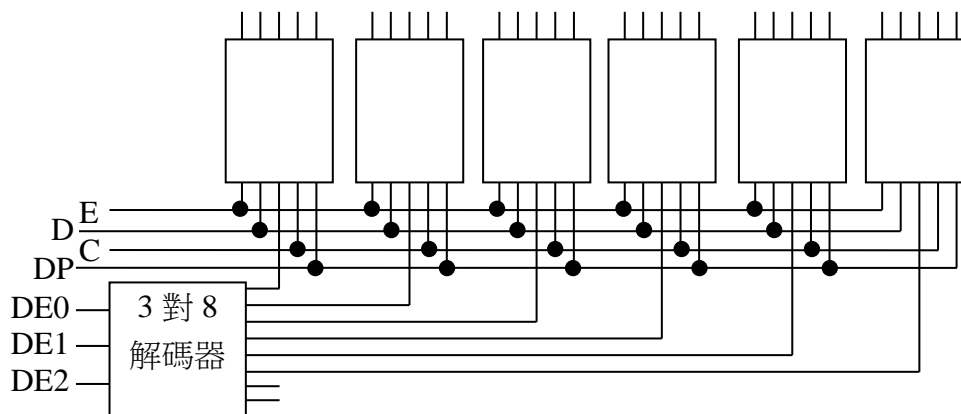
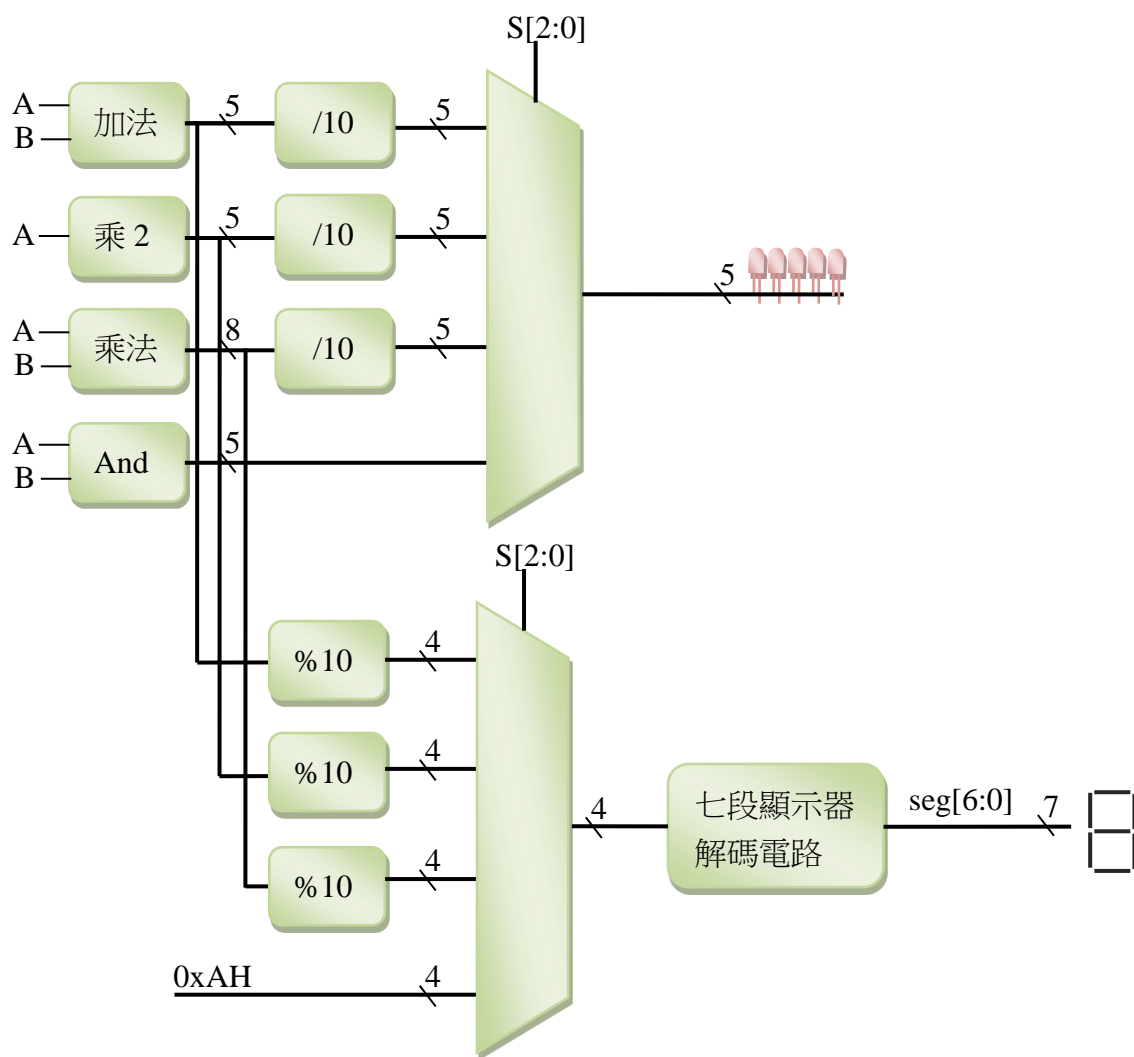


圖 4-7 七段顯示器之解碼電路

此次實作之方法要求如下：先分別求出四個運算結果，然後利用三元運算子(?:)指定選擇

線所選擇的運算結果的個位數(算數運算時)或 A(邏輯運算時)。再利用另一個三元運算子指定選擇線所選擇的運算結果的十位、百位數或邏輯運算結果給 5 個 LED。整個 ALU 之架構可



參考

圖 4-8。

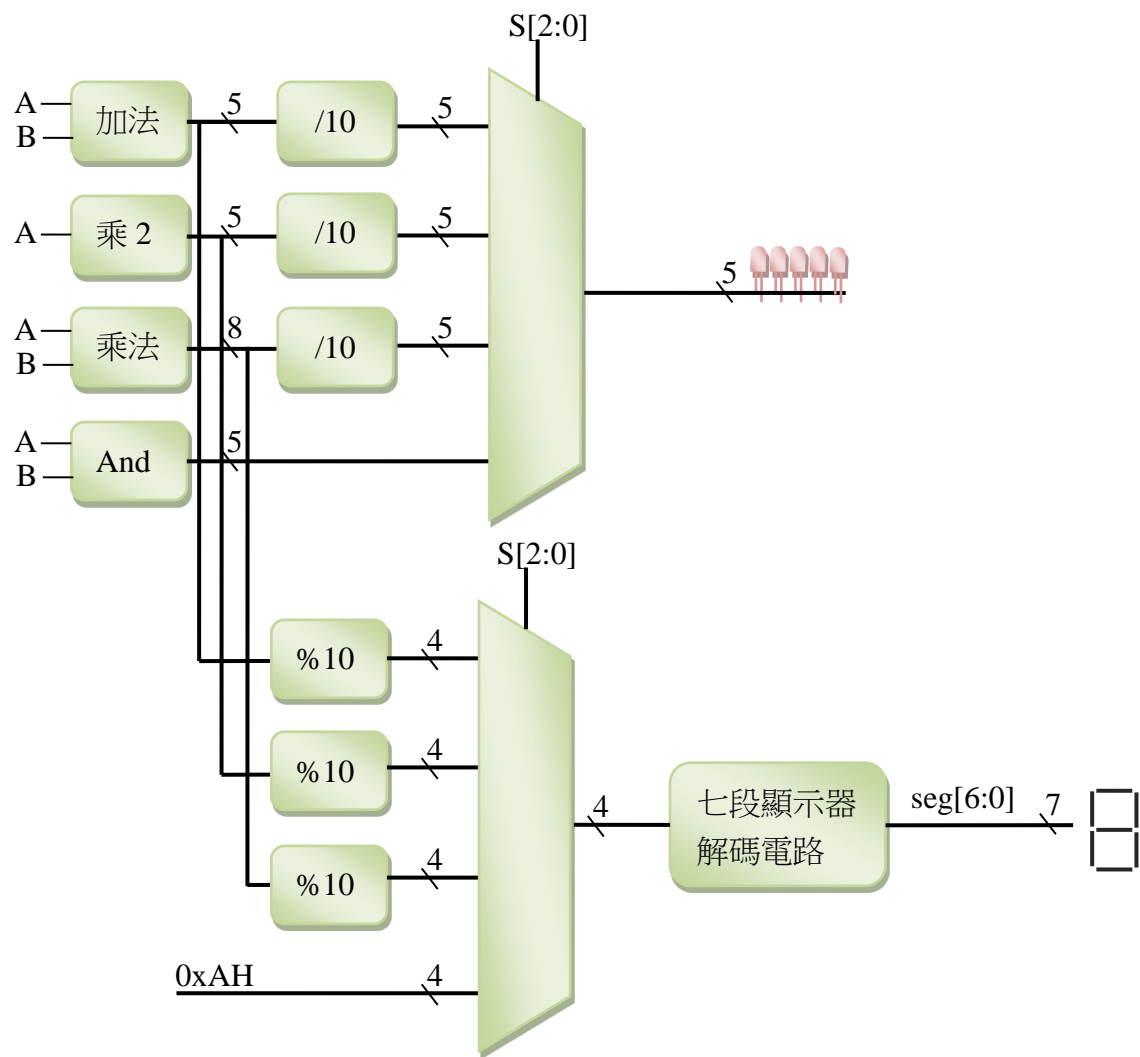


圖 4-8 ALU 示意圖

實驗5 除頻器

在上一次實驗中，已介紹過 Behavioral Model，本實驗再進一步介紹，在 Behavioral Model 兩種指定值的方式：Blocking 與 Non Blocking。

一、Blocking 與 Non Blocking

在 Behavioral Model 中，指定運算結果的方式有兩種，分別為 Blocking 與 Non Blocking。這兩者的差異為在同一區塊中的敘述是否同時執行。Blocking 的指定語法用「=」來指定運算結果，在同一區塊中 Blocking 的指定方式會讓前一行的敘述執行完之後，再接著執行下一行的敘述，就如同程式碼一樣。如圖 5-1(a)所示為 Blocking 的指定方式，會先將 b 的值指定給 a，然後再將 a 的值指定給 b，結果 a 和 b 的值會等於原先 b 的值。Non Blocking 的指定語法用「<=」來指定運算結果，在同一區塊中 Non Blocking 的指定方式會讓所有的敘述同時執行。如圖 5-1(b)所示為 Non Blocking 的指定方式，同時會將 b 的值指定給 a，以及將 a 的值指定給 b，結果 a 和 b 的值會互相對調。

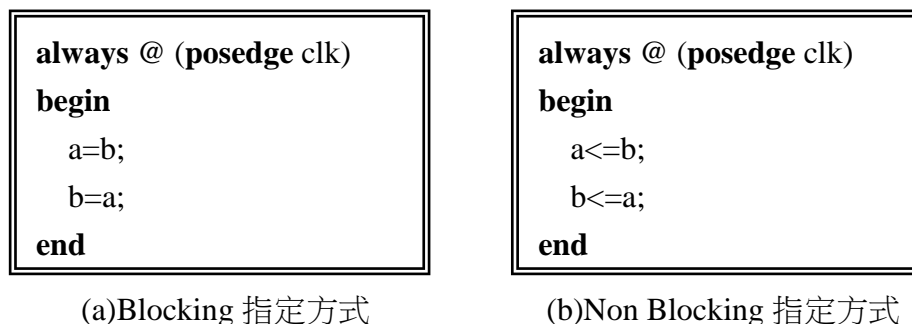


圖 5-1 Blocking 與 Non Blocking 之指定方式

二、消除彈跳

對於機械式的開關，先天上就必然存在著彈跳現象。所謂的彈跳現象，就是當開關按下去，會有極短暫的時間，電路會呈現若通若不通，斷斷續續的物理現象，結果導致輸出在 0 與 1 之間跳動。同樣地，當開關放開時，也會同樣有彈跳現象發生，如圖所示。

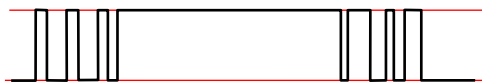


圖 5-2開關之彈跳現象

一般來說，此彈跳現象約持續 20ms，由於非常短暫，對於組合電路來說，只會導致輸出結果短暫的時間內不穩定。而對於順序電路來說，有時會產生錯誤的結果，最明顯的例子為計次電路。當開關被按下時，由於彈跳現象，電路會解讀成按了好幾下而導致計數錯誤。解決彈跳現象的方法大致上可分成軟體與硬體來解決。軟體的解決方法通常是偵測到開關狀態改變時，延長一段時間後，才可再偵測開關狀態。而硬體可使用電容或正反器來處理。底下為利用位移的方式來消除彈跳現象。因為一般的彈跳現象最多約 20ms，若以 1kHz 的頻率來偵測開關訊號，則大約 20 個 clock 後即可穩定。假定教學板上開關的彈跳現象在 7ms 之內，只需要偵測 7 個 clock 內，開關的訊號是否一致即可。然而為簡化電路，利用一個 AND 邏輯

聞來判斷這 5 個訊號是否同為 1，若是連續 5 次偵測開關的狀態都為 1，即表示開關已按下且已達穩定狀態，此時將輸出設為 1。如果其中有 1 次以上為 0，表示此時不穩定，還處於彈跳現象中，或開關已放開且已達穩定狀態，此時將輸出設為 0，如此即可消除彈跳現象。若是開關的彈跳時間較長，則此程式必需多加位移暫存器，判斷更多的取樣點。也可以將取樣頻率調低。有一點需注意的是，彈跳現象愈長，則切換開關的時間需更長，也就是說切換開關的頻率不能太快。

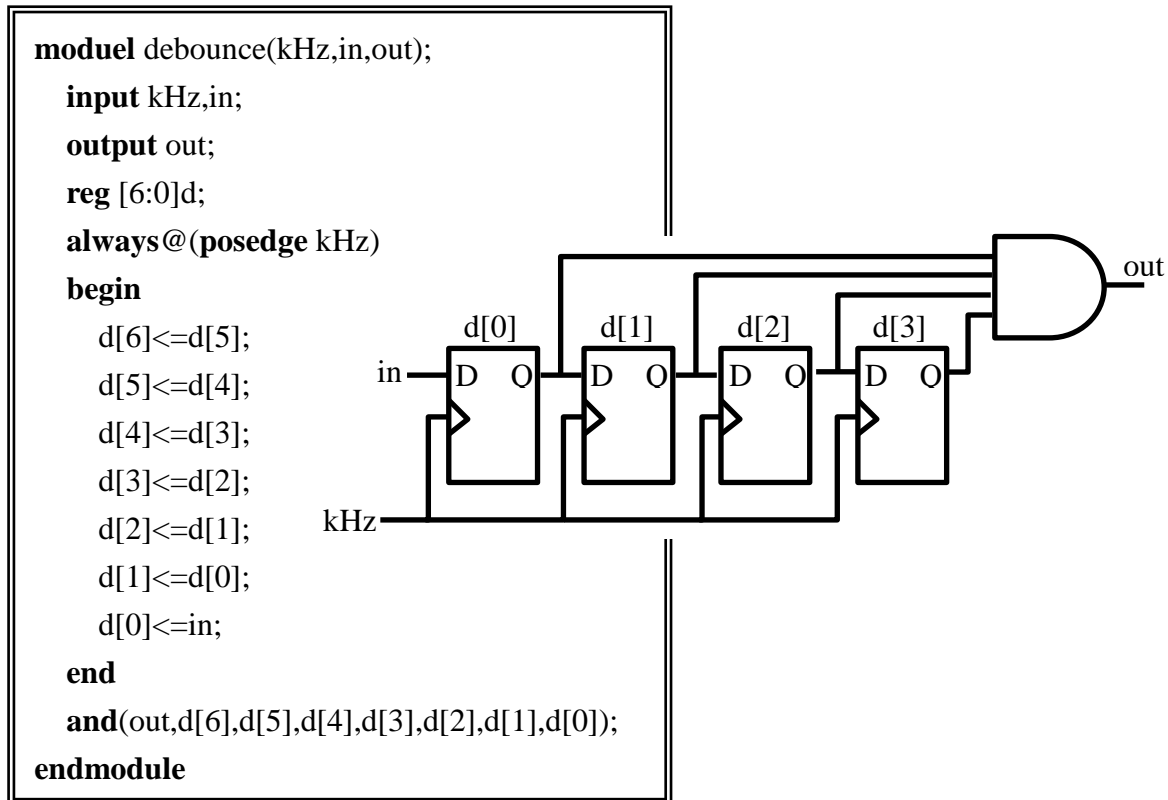


圖 5-3 消除彈跳之模組

三、模擬

請依圖 5-5 之程式碼撰寫編譯後，按圖 5-4 之波形圖設定輸入波形，觀察其輸出波形。再將程式碼改為 Non Blocking 之方式，再觀察其模擬後的輸出波形，並和 Blocking 方式比較其結果差異。

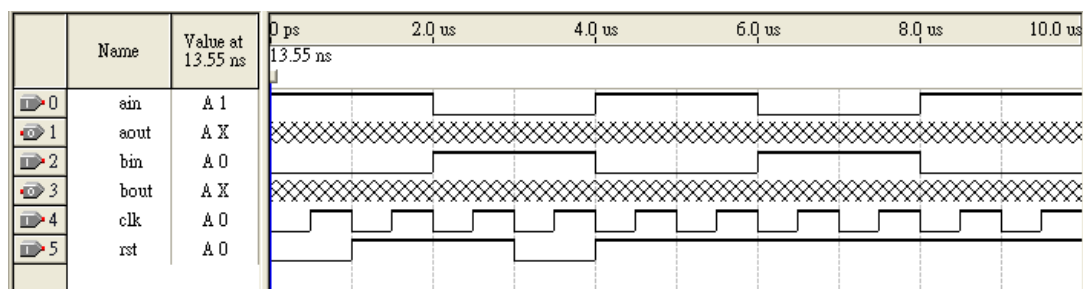


圖 5-4 輸入波形圖

```

module blocking(ain,bin,clk,rst,aout,bout);
input ain,bin,clk,rst;
output reg aout,bout;
always @ (posedge clk or negedge rst)
begin
    if (~rst)
        begin
            aout=ain;
            bout=bin;
        end
    else
        begin
            aout=bout;
            bout=aout;
        end
    end
endmodule

```

圖 5-5 Blocking 之模擬範例程式

四、實作

設計一個 3 位元計數器電路，每當 PS3 按鈕按 1 下時，計數器累加 1。先不考慮消除開關的彈跳現象，觀察計數器是否能正常計數。再來將開關加上消除彈跳的電路，如圖 5-3，觀察計數器是否能正常計數。

由於消除彈跳現象，需要 1kHz 的方波，而教學板只提供由 pin55 輸入的 10MHz 方波。因此必須先將 10MHz 的方波除頻成 1kHz 的方波。圖 5-6 為頻率除 8 之除頻器，in 為輸入之 clock，out 為輸出之 clock，counter 設為輸出是觀察用，實際使用時可不用設成輸出。設計的概念是當輸入的 in 為正緣觸發時，counter 累加 1，當 counter 從 0 累加到 3 時，表示 in 已經經過 4 個 clock，此時令 out 反相，同時也令 counter 歸零。如此經過 8 個 clock 後，out 再次反相。模擬結果如圖 5-7，當 in 經過 8 個 clock 後，out 完成 1 個 clock，如此即為除 8 之除頻器。在設計除頻器時，由於 counter 要累加計數，必須要注意 counter 的位元數要夠多。否則無法數到預計的數字。

首先計算利用 10MHz 產生 1kHz 需要除頻多少，然後修改圖 5-6 之程式碼，使之產生 1kHz 之 clock。然後再依將此 clock 給消除彈跳電路使用整個實作之架構圖如圖 5-8 所示。

```

module div8(in,out,counter);
input in;
output reg out;
output reg [2:0]counter;
always@(posedge in)
if (counter==3)
begin
    counter<=0;
    out<=~out;
end
else
    counter<=counter+1;
endmodule

```

圖 5-6 頻率除 8 之除頻器

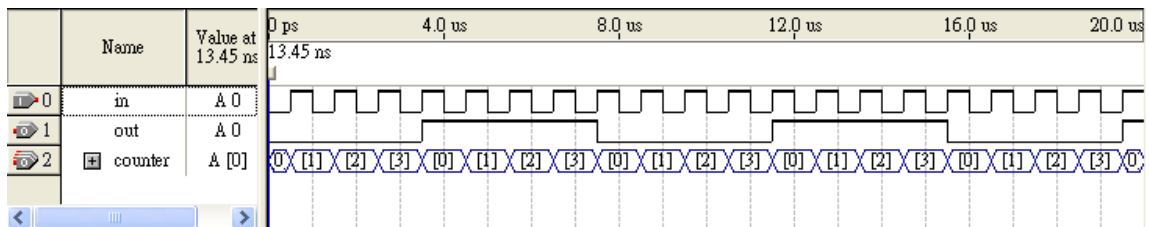


圖 5-7 頻率除 8 之除頻器模擬結果



圖 5-8 三位元計數器架構圖

實驗6 跑馬燈

一、實作

利用 8x8 點矩陣 LED 來做跑馬燈，先設計一個 8x8 的圖案(或是更大，例如 8*12)，然後利用 1Hz clock 讓圖案在 8x8 點矩陣 LED 上循環顯示，造成動畫的效果。另外再利用兩個開關來設定跑馬燈是否要環循移動及要移動的方向。如圖 6-1 所示，先設計一個圖案，如左圖，然後依 1Hz clock 依序將圖案變成中間圖、右圖，並依些循環顯示。當依反向循環顯示時，則會由右到左顯示。

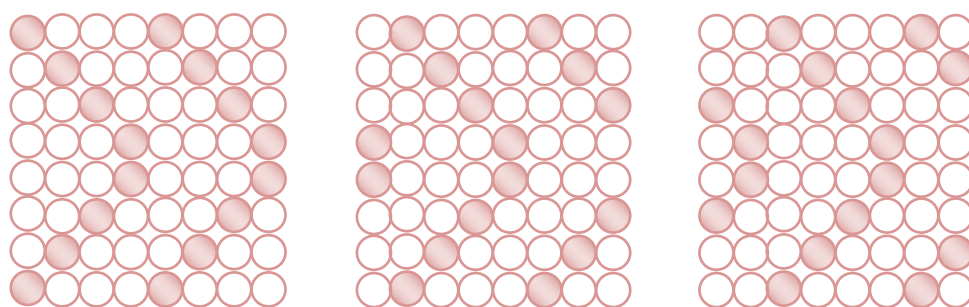


圖 6-1 8x8 點矩陣 LED 設計範例

8x8 點矩陣 LCD 的腳位設定請參考附錄 1。使用方式為設定 row1-row8 中的某個腳位為 1，其餘為 0，例如 row=8'b0000_0100。接著設定 col1-col8 中某個腳位為 1，其餘為 0，例如 col=8'b0000_1000。這樣第 3 列、第 4 行的 LED 燈會亮，其餘不顯示。如果同時要顯示若干個 LED 燈，則必須讓這些要顯示的 LED 燈輪流亮。例如要讓對角上的 8 顆 LED 都亮起來，就必須依序如下設定。首先令 row=1、col=1，此時左上角 LED 亮，其次令 row=2、col=2，此時換對角線上的第二顆 LED 亮，第一顆 LED 熄滅。再令 row=4、col=4，換只亮對角線上第三顆 LED 燈，依此類推令 row 及 col 為 8、16、32、64、128，會依序只亮對角線上第四、五、六、七、八顆 LED 燈。雖然每次只亮一顆 LED 燈，但只要輪流的頻率很快，則因視覺暫留的原因，看起來好像八顆 LED 都同時亮起來。基於這樣的原理，如果要讓設計好的一個 8x8 點圖案要顯示出來，可以用 1kHz 的頻率來做輪流亮的頻率。以圖 6-1 中的左圖為例，首先令 row=1，然後令 col 為第一列上要亮的點為 1，col=8'b1000_1000。經 1ms 後，令 row=2、col=8'b0100_0100，再 1ms 後，令 row=4、col=8'b0010_0010，依此類推。經過 8ms 後又回到顯示第一列的圖案。

整個系統可分成三部分，第一部分為除頻器，這部分電路可參考實驗 5。將教學板上的 10MHz 除頻成 1kHz 以及 10Hz。其中 1kHz 是用來給 8x8 點矩陣 LCD 顯示資料用，而 10Hz 則用來設定切換圖案，也就是說每 0.1 秒換一個圖案。第二部分為計數器，將 10Hz 當做此電路的 clock 做正緣觸發，計數現在要顯示第幾種圖案。當 Enable 設為 1 時開始計數，反之則暫停計數。當 Dir 設為 1 時，計數器遞增計數，當數到最後一個圖案時，要將計數器歸零。當 Dir 設為 0 時，計數器遞減計數，當數到 0 時，要將計數器設為最後一個圖案。第三部分為解碼器，也就是將計數器目前的值解碼成相對應的圖案，並以 1kHz 來輪流顯示相應圖案的 8 列資料，輸出到教學板上的 8x8 點矩陣 LED。

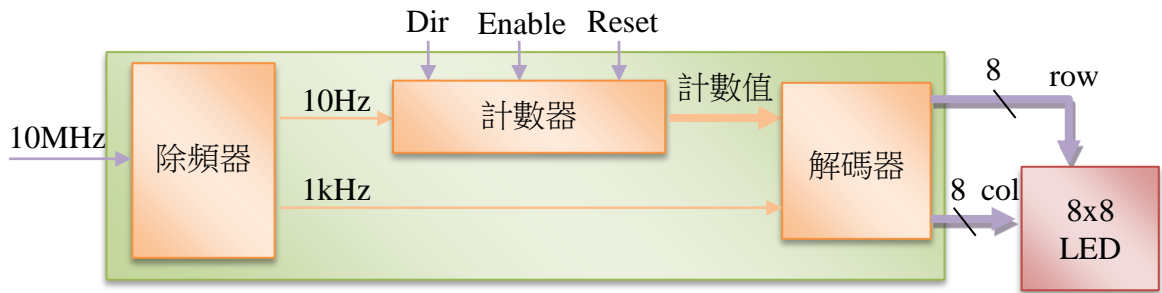


圖 6-2 跑馬燈電路結構圖

本實驗的 Reset 採非同步方式，也就是隨時按 Reset，計數值立刻回到 0，而不是等 Clock 正緣觸發時再歸零，非同步的 Reset 程式參考圖 6-3。

```
always@(posedge clock or negedge reset)
begin
    if (~reset)
        counter=0;
    else
        begin
            .....
        end
    end
```

圖 6-3 非同步之 Reset 設計範例

實驗7 計時器

首先介紹 **task** 之使用方式，然後實作一計時器，以 1Hz 的頻率，從 00:00:00 計時到 11:59:59。按 **reset** 可以重新計時，按 **enable** 可以開始計時，取消 **enable** 則暫停計時。此外，為了方便測試計時到 11:59:59，請再加一個加速鍵按鈕，不按時以 1Hz 計時，按下加速鍵時以 100Hz 計時。以六個七段顯示器分別表示時、分、秒的個位數與十位數。

一、task

在實驗 2 中提到為了能夠模組化設計，可將某個功能寫成一個 **module**，以後要使用此功能時，可直接使用這個 **module**，而不需要再重覆寫一次程式碼。然而，在實驗 4 中卻說 **always** 裡面的敘述，不能使用 **module**。此時，若要能夠保持模組化的設計概念，進而重覆使用已經寫好的程式碼，就必需使用 **task** 方式。

task 的使用方式如圖 7-1 所示，**task** 需寫在 **module** 的區塊裡。**task** 後面只寫 **task** 的名稱，然後分號，而不像 **module** 那樣還要寫 **input** 或 **output** 的腳位。整個區塊由 **task** 附加名稱開始，到 **endtask** 結束。**task** 的 **input** 與 **output** 寫在 **task** 底下，呼叫時，則按宣告的順序來指定，例如第 8 行呼叫 **show** 這個 **task**，第一個參數 **counter** 就對應到第 12 行的 **num**，第二個參數 **leds** 就對應到第 13 行的 **out**。**task** 裡面宣告成 **reg** 的資料在每次呼叫 **task** 時，都會重設，所以第一次呼叫 **task** 時，裡面的運算結果，並不能儲存給下次呼叫 **task** 時來使用。如果有這樣的需求，則需將運算結果當作 **output** 來傳回，並於下次呼叫時連進來。此外，由於 **task** 本身已是 Behavioral Model，所有裡面不可以再使用 **always**，但也不可以用 **assign**。

<pre>module increase(clk,leds); input clk; output [6:0]leds; reg [1:0]counter; always @ (posedge clk) begin counter=counter+1; show(counter,leds); end</pre>	<pre>task show; input [2:0]num; output reg [6:0]out; case (num) 3'd0:out=7'b1111_110; 3'd1:out=7'b0110_000; 3'd2:out=7'b1101_101; 3'd3:out=7'b1111_001; endcase endtask endmodule</pre>
---	--

圖 7-1 task 之使用範例

二、七段顯示器掃描電路

在教學板上，有六顆七段顯示器，每一個七段顯示器都有 8 個輸入資料腳位，如果各自擁有獨立的輸入腳位，則需要 48 個腳位，這對於晶片有限的腳位數來說，過於浪費。所以一

一般在設計上，會將六顆七段顯示器的輸入腳位分別接在一起，也就是每個七段顯示器的 a 腳位都接一起，每個 b 腳位也都接一起。那麼當訊號從晶片輸出到七段顯示器時，會導致每一顆七段顯示器的輸出都一樣。為了解決這個問題，就必須使用掃描電路。也就是說，以共陽極七段顯示器為例，在一段時間內，只讓某一個要亮的七段顯示器的共陽極腳接設為 1，其他的七段顯示器共陽極接腳設為 0，這樣雖然每一個七段顯示器都接收同樣的輸入訊號，但只有我們要它亮的七段顯示器會亮，其它的都不會亮。教學板上的 DE0-DE2 就是用來選擇哪一個七段顯示器要亮。這樣所需的腳位只要 8 個資料線再加上 3 個選擇線，共 11 個輸出腳位，如圖 4-7 所示。如果我們要讓六顆七段顯示器全都顯示不同的數值，例如 876543，可以在第 1ms 的時間內讓第一顆亮 8，從 1ms 至 2ms 的時間內讓第二顆亮 7，從 2ms 至 3ms 時讓第三顆亮 6。依此類推，讓六顆七段顯示器輪流亮 1ms 的時間。這樣因視覺暫留的原因，看起來就像六顆全亮，呈現 876543，結果如圖 7-2 所示。

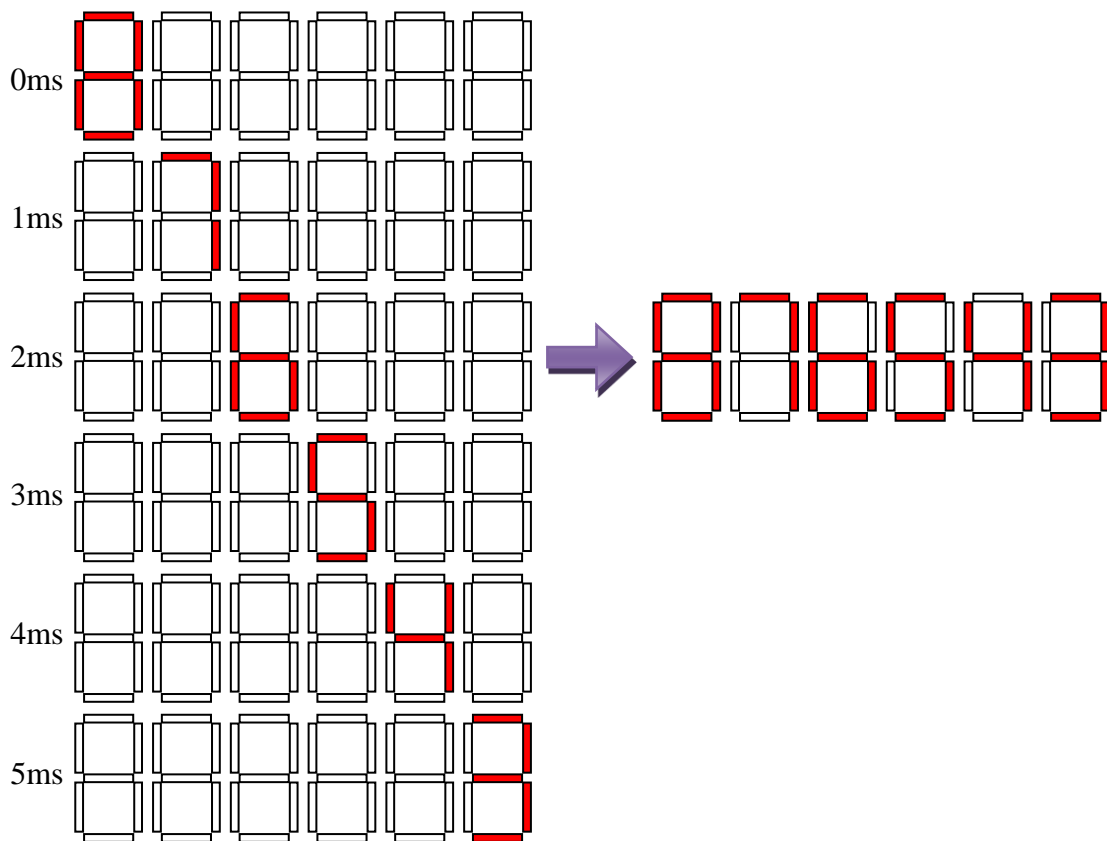


圖 7-2 七段顯示器掃描顯示示意圖

再來只要在正確的時間設定正確的輸出訊號給對應的七段顯示器，同時設定對應的 DE0-DE2，就可以讓六顆七段顯示器分別亮各自要顯示的數字。此種掃描電路同樣可以應用在點矩陣的 LED 陣列或鍵盤上。

三、實作

整個電路架構如圖 7-3 所示，請先完成綠色方塊的掃描電路。先將 10MHz 除頻成 1kHz，用這個頻率來當七段顯示器的掃描頻率。用這個 1kHz 頻率來對計數器重複計數 0-5，表示要顯示第幾個七段顯示器。然後將計數的結果(DE0-DE2)當做多工器選擇線來選擇 HHMMSS 的

哪個數字要送到解碼器，同時這 DE0-DE2 也輸出到教學板上選擇相對應的七段顯示器。在此先將 HHMMSS 指定成 012345 這六個數字，而不用計時器的值，如此可與計時器的結果做區隔。看看是否能在教學板上的七段顯示器正確顯示 012345，如果能正確顯示，再完成藍色方塊的計時電路。

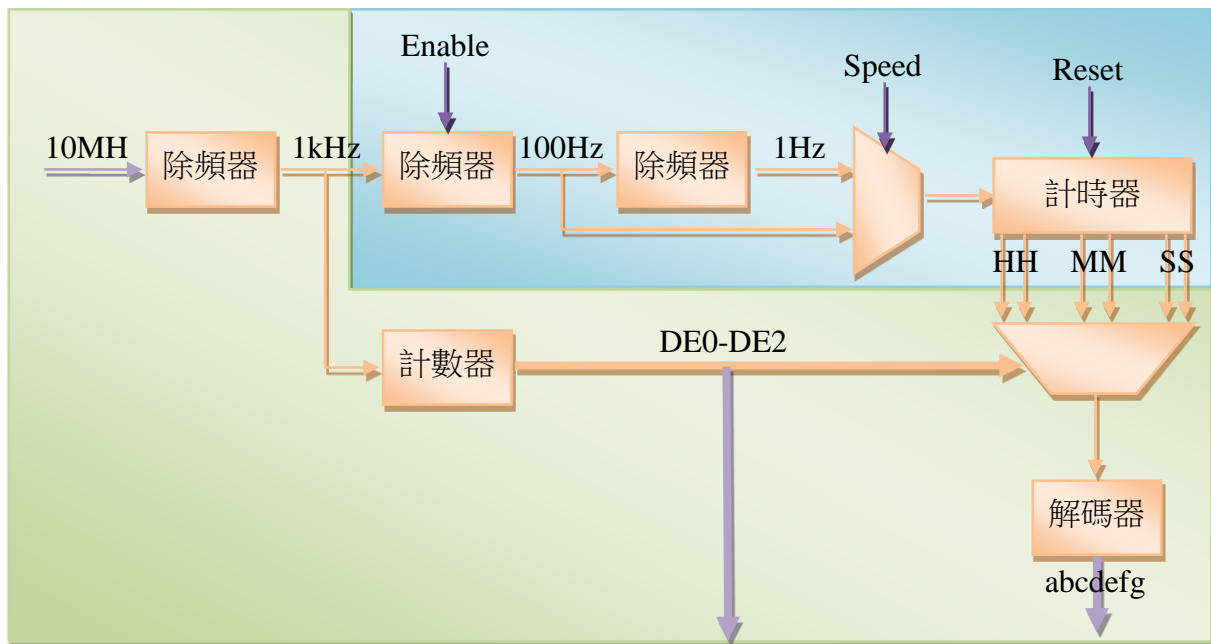


圖 7-3 計時器電路結構圖

實驗8 有限狀態機

有限狀態機的設計可分成兩種，一是 Moore-type FSM，其輸出只與所處的狀態有關。另一種是 Mealy-type FSM，其輸出除了與狀態有關外，還與輸入有關，也就是說其輸出是由所處狀態與輸入來決定。底下以同一問題為例，來說明如何用 Verilog 設計 FSM。

一、Mealy-type FSM

舉例來說，要設計一個電路，當輸入連續為 1 時，輸出為 1，其餘輸出為 0，則根據 Mealy-type FSM 的方式可繪出狀態圖

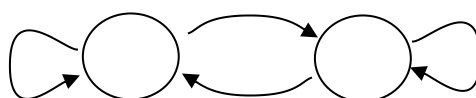


圖 8-1 以 Mealy-type 所繪之狀態圖

將狀態圖改寫成狀態表

表 8-1 以 Mealy-type 所製之狀態表

Current State	Next State		Output	
	w=0	w=1	w=0	w=1
s0	s0	s1	0	0
s1	s0	s1	0	1

Verilog code 可如下方式撰寫

<pre> module mealy(ck,w,z,rst); input ck,w,rst; output reg z; reg y,Y; parameter s0=1'b0,s1=1'b1; always@(y,w) case (y) s0:Y=w?s1:s0; s1:Y=w?s1:s0; default: Y=s0; endcase </pre>	<pre> always@(posedge ck , negedge rst) if (~rst) begin y<=s0; z<=0; end else begin y<=Y; z<=(y==s1 && w==1); end endmodule </pre>
--	--

圖 8-2 以 Mealy-type 所寫之 Verilog code

在此程式第五行中 **parameter** 用來將一代號定義為一個固定的數值，以方便程式中用此代號來表示此數值，增加程式的可讀性。

二、Moore-type FSM

同上述的問題，若根據 Moore-type FSM 的方式可繪出狀態圖

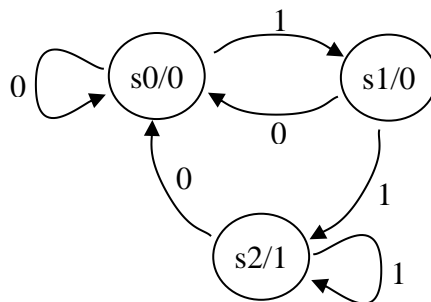


圖 8-3 以 Moore-type 所繪之狀態圖

將狀態圖改寫成狀態表

表 8-2 以 Moore-type 所製之狀態表

Current State	Next State		Output
	w=0	w=1	
s0	s0	s1	0
s1	s0	s2	0
s2	s0	s2	1

Verilog code 可如下方式撰寫

<pre> moduel moore(ck,w,z,rst); input ck,w,rst; output reg z; reg [1:0]y,Y; parameter s0=2'b00,s1=2'b01,s2=2'b10; always@(y,w) case (y) s0:Y=w?s1:s0; s1:Y=w?s2:s0; s2:Y=w?s2:s0; default: Y=s0; endcase </pre>	<pre> always@(posedge ck , negedge rst) if (~rst) y<=s0; else y<=Y; assign z<=(y==s2); endmoduel </pre>
--	---

圖 8-4 以 Moore-type 所寫之 Verilog code

三、實作

請設計一電路，當連續四個輸入為 1001 或 1111 時，輸出為 1，其餘情況輸出為 0，設計完成後，請先以模擬器驗證電路的正確性，驗證時輸入與輸出的樣本如下

Clk	t0	t1	t2	t3	t4	t5	t5	t7	t8	t9	t10	t11	t12	t13	t14	t15	t16	t17
W	0	1	0	1	1	1	1	0	0	1	1	0	0	1	1	1	1	1
Z	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	1

模擬正確後，以 SW1 當輸入，PS3 當 Clock，在教學板上實作出來。由於 SW1 只是組合電路的一部份，不需加上消除彈跳電路，但 PS3 做為 Clock，務必加上消除彈跳電路，確保按一下，只輸入一次訊號。

由於題目要求是判斷連續四個輸入的值，在繪製狀態圖時，可先將四個輸入的所有情形以樹狀圖的方式列出來，然後再對最底下一層的狀態，依據題目需求，根據輸入將狀態轉移至下一個狀態，由於此種窮舉方式會隨著問題的複雜而導致狀態數目暴增，需要大量的化簡工作。好處是先將所有可能情形列出，比較不會有遺漏考慮的問題。如圖 8-5 所示。

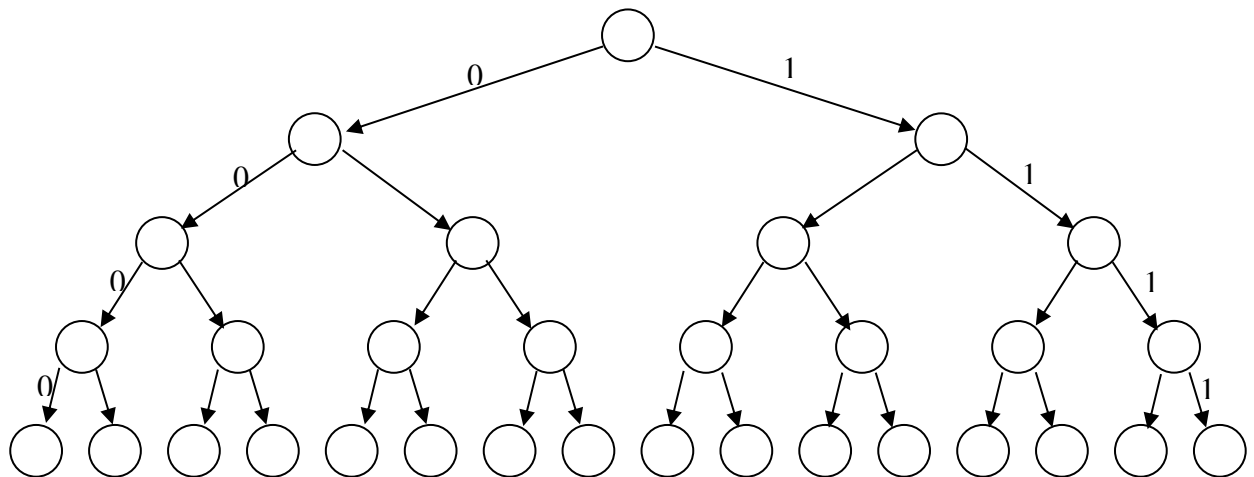


圖 8-5 實作狀態圖

圖 8-6 為實作之架構圖，從外部的 10MHz 經除頻器後得到 1kHz 的時脈。以此時脈用來消除 PS3 的 Clock 訊號上的彈跳現象，得到 clk 的訊號。當 PS3 按一次的時候，可令 clk 的值從 0 變為 1，再變為 0 各一次，無彈跳現象。將 clk 訊號當 FSM 的時脈來用，可使得 PS3 按一次，只會讓 W 的值輸入一次。Reset 請設計為非同步歸零，Z 為輸出訊號。

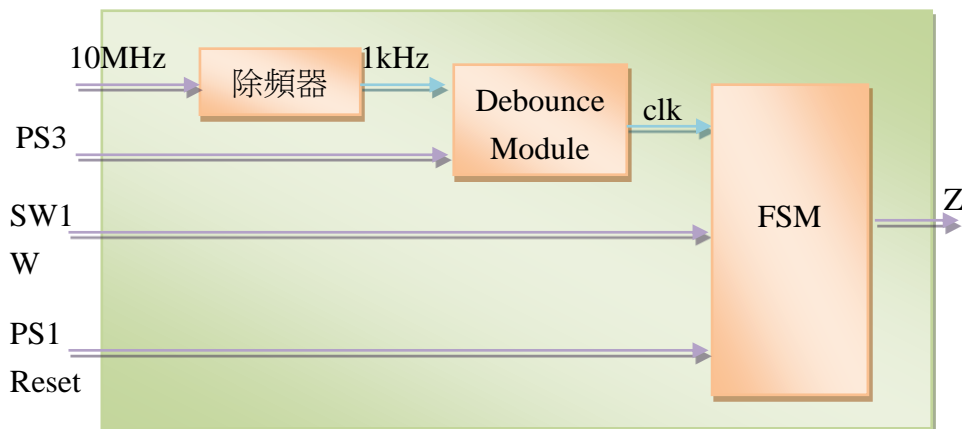


圖 8-6 實作之架構圖

實驗9 以 Gate Level 設計有限狀態機

實驗 8 中是以 Behavioral Model 的方式來設計有限狀態機。此種方式雖然可以快速設計出電路圖，然而從其合成出來的電路中可以發現，其選擇下一個狀態的方式是先經過一個 case，判斷其目前狀態為何，此時需用一個多工器來完成這個部分。判斷所處的狀態後，再依輸入決據其下一個狀態為何，此時又需用一個多工器來完成這個部份。然而每一個多工器都需兩個邏輯閘的延遲時間，所以共需四個邏輯閘的延遲時間。

本次實驗是以 Gate Level 的方式來設計有限狀態機，其決定下一個狀態的方式，是直接根據目前狀態與輸入訊號，經過一組合電路來決定。而這個組合電路可經由卡諾圖化簡成 SOP 或 POS 的形式，如此只需兩個邏輯閘的延遲時間。這樣的設計好處是可以有更快的運算速度，而缺點則是需要較長的時間撰寫 code。

一、以 Gate Level 設計流程

FSM 架構可以如圖 9-1 來表示，左邊的組合電路為根據目前狀態以及輸入訊號，求得出下個狀態。中間的正反器為儲存目前的狀態值，以 Clock 來同步觸發更新目前狀態值。右邊的組合電路則根據目前狀態（或再考慮 Input）求得輸出訊號。

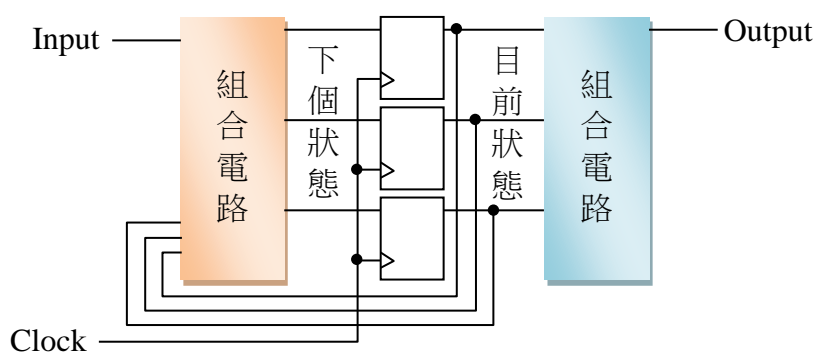


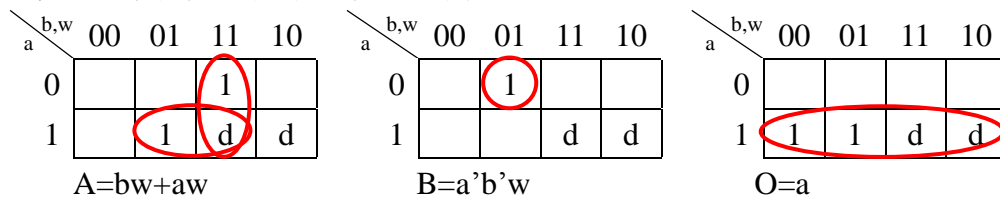
圖 9-1 FSM 架構圖

以實驗 8 所述之 Moore-type FSM 為例，輸入為連續兩個 1 時，輸出 1，其餘則輸出 0，由表 8-2 可知共有 3 個狀態，故狀態的編碼需要 2 位元，即圖 9-1 中的正反器需要 2 個，分別標為 y1、y0。則表 8-2 可改寫成表 9-1。

表 9-1 真值表

Current State		Input(w)	Next State		Output(O)
a	b		A	B	
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	1
1	0	1	1	0	1
1	1	0	d	d	d
1	1	1	d	d	d

再依據卡諾圖來化簡求得下個狀態與輸出的邏輯組合電路



最後按圖 9-1 之架構圖將上述化簡後的電路組合出最後完整的電路，如圖 9-2，Verilog 程式碼則如圖 9-3。

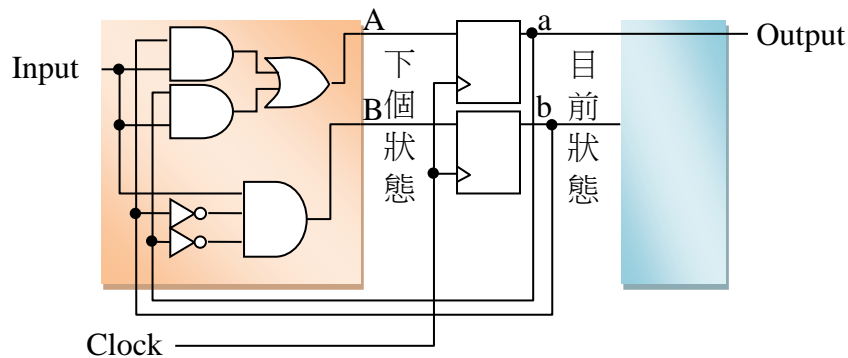


圖 9-2 以 Gate Level 完成之 FSM 架構圖

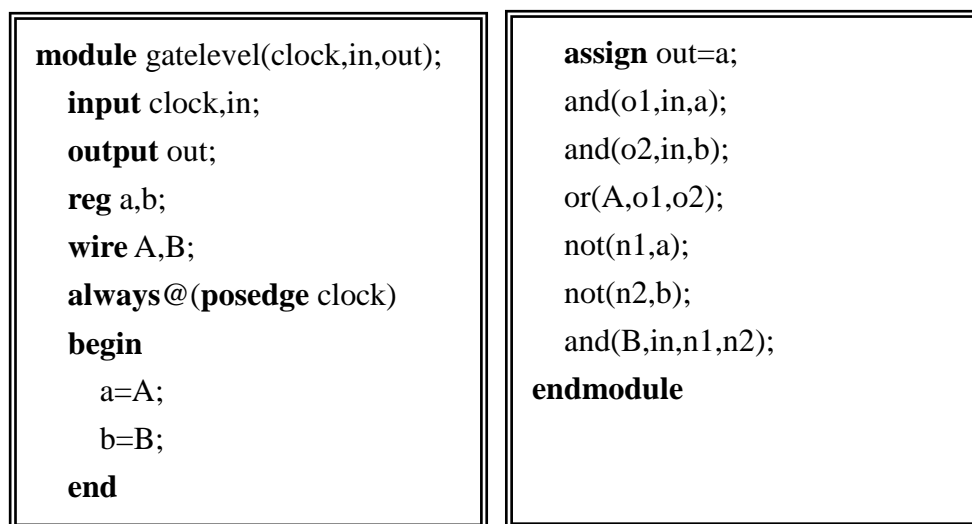


圖 9-3以Gate Level完成之FSM程式碼

二、FSM 化簡

由上述方式可知，以 Gate Level 來完成時，如果狀態的數目愈多，則所需的正反器個數愈多，因此需要推導更多的組合電路。然而，如果能事先將 FSM 化到最簡的形式，則可大大減少所需的推導過程。

表 9-2 化簡前、後的狀態表

Current State	Next State		Output
	Input=0	Input=1	
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

(a)化簡前

Current State	Next State		Output
	Input=0	Input=1	
A	B	C	1
B	A	F	1
C	F	C	0
F	C	A	0

(b)化簡後

表 9-3 化簡梯形表

B	B=D C=F					
C	X	X				
D	C=G	D=B F=G	X			
E	X	X	E=C	X		
F	X	X	F=E E=D	X	F=E C=D	
G	X	X	E=G	X	C=G	E=F D=G
	A	B	C	D	E	F

如表 9-2 所示，左邊表(a)為化簡前的狀態表，我們可依此表格畫出如表 9-3 之梯形表格。從表 9-2(a)的第一列開始看，狀態 A 與狀態 B 的輸出相同，故有可能可以化簡，其條件為 $B=D$ 且 $C=F$ ，所以在表 9-3 的第 B 列 A 欄裡填上其成立時的條件。然後再檢查 A 與 C，其輸出相異，故不可能化簡。再檢查 A 與 D，有可能化簡，在第 D 列 A 欄裡填上條件。依此將狀態 A 與其他狀態一一比較。再來將狀態 B 與 C、D、E、F、G 等狀態比較，並將有可能化簡的狀態分別填入相應列的 B 欄。而因為 A 與 B 相同的可能性已經在檢查狀態 A 時列出條件，故不需再檢查。同理，剩下的狀態 C、D、E、F 依次與之後的狀態相比即可，最後可得表 9-3 之結果。表中畫 X 的即表示相對應的兩個狀態不可能化簡成一個狀態，例如第 C 列第 B 欄畫一個 X，表示狀態 B 與 C 不可能化簡。接下來依序檢查每一個條件是否有不成立的地方。首先看第 B 列 A 欄，第一個條件是 $B=D$ ，而目前 B 與 D 並未畫 X，故還有可能成立。第二個條件 $C=F$ ，依然未畫 X，有可能成立。再來看 D 列 A 欄，條件是 $C=G$ ，有可能成立。依此方式一一檢查，再來看第 F 列 E 欄，第二個條件 $C=D$ ，然而在第 D 列 C 欄是畫 X，所以條件不成立，故狀態 E 必定與狀態 E 不同，可將第 F 列 E 欄畫 X。這樣的結果又會導致之前的判斷改變，如第 F 列 C 欄的條件裡，就有 $F=E$ 的條件，因此第 F 列 C 欄也因此要畫 X。如此從第一個條件檢查到最後一個條件，如果中途有某個條件不成立，而將某欄某列新畫上

X，則必需再重新檢查一次，直到從第一個條件至最後一個條件都沒有不成立的情形產生，此時即可得到最後的化簡結果。然後在表 9-2 中，將相同的狀態刪掉多餘的狀態，只保留一個，並將 Next State 中，改成保留的狀態名稱，最後可得到表 9-2(b)的結果，最後對化簡後的狀態編碼。未化簡前有七個狀態，編碼時需要 3bits，而化簡後有四個狀態，只需要 2bits，可減少一個正反器。

三、實作

請撰寫一具有 2bits 同位檢查之電路，以相鄰的兩個輸入來判斷，如果有奇數個 1，則輸出 1，否則輸出 0。請參考實驗 8 之消除彈跳之電路來處理輸入的訊號。測試樣本如下所示

Input	0	0	1	1	0	1	1	1	0	0	0	1	1
Output	0	0	1	0	1	1	0	0	1	0	0	1	0

實驗10 自動販賣機

設計一報紙自動販賣機，可以投 5 元、10 元，分別用按 PS3(Pin124)與 PS4(Pin126)來輸入。用四個 LED 表示目前已投金額，亮一顆表示已投 5 元，亮二顆表示已投 10 元，依此類推，最多亮四顆表示已投 20 元。用另外一顆 LED 表示是否可以購買了，即當金額大於或等於 15 元時，表示可購買的 LED 會亮，否則是暗的。當可以購買時，按 PS2(Pin56)表示購買，剩下的金額依舊顯示在四顆 LED 上。Reset 或退幣按 PS1(Pin54)表示，按完後顯示金額的 LED 全暗，表示可購買的 LED 也是暗的。

由於 PS1-PS4 的彈跳現象很嚴重，所以請參考實驗 8 來消除彈跳現象。注意，PS1 與 PS2 按下按鈕時是負緣，放開按鈕時是正緣，而 PS3 與 PS4 相反，按下按鈕時是正緣，放開按鈕時是負緣。圖 10-1 是整個架構圖，對 FSM 而言，1kHz 是同步時脈，Reset 是非同步歸零，five、ten、buy 三個訊號為輸入訊號，已投金額是目前狀態，是否可購買是輸出。

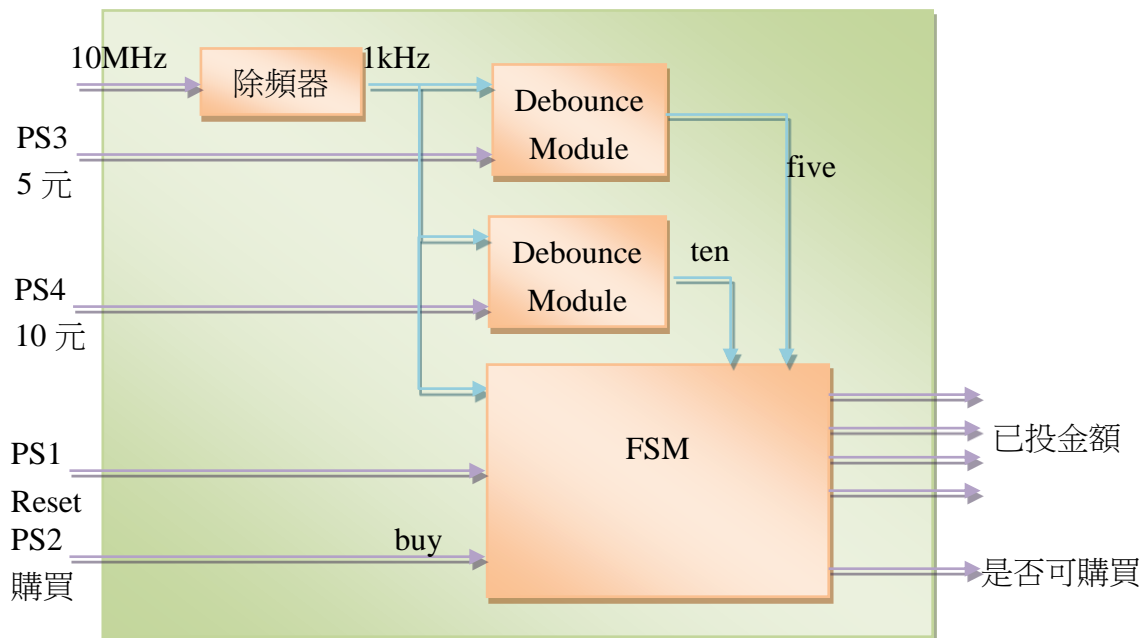


圖 10-1 自動販賣機架構圖

實驗11 期末專題

提示：亂數的產生可以用 10MHz 的時脈來計數一個 counter，此 counter 的值就在產生亂數的範圍內計數，例如要產生 1 至 8 的亂數，就讓 counter 的值重覆的從 1 累加到 8，再回到 1 重新累加。當需要亂數的時候，直接讀取 counter 的值即可。在某些情形下，會因計數頻率太快，導致抓取到的亂數，在使用上有問題，可適當的調慢計數的頻率。

一、題目一：貪吃蛇

用 8x8 點矩陣 LED 顯示器來顯示貪吃蛇，蛇的長度固定三個紅色 LED 燈，不會隨著吃到食物而變長，也不用考慮撞到自己或牆壁死掉的情形。移動的方向為上、下、左、右。當按鈕放開時則維持剛才的方向繼續前進。當蛇走到邊界時，可從另一邊出來。食物用一個綠色 LED 燈顯示，隨機產生位置後就固定到被吃掉為止。每吃到一個食物則加一分，同時再隨機產生下一個食物的位置。以七段顯示器顯示目前的得分。

要能夠隨機位置產生食物，不能分別隨機產生 row 的位置及 col 的位置，因為我們在同一時間只能產生一個隨機亂數，所以要隨機產生一個 0-63 之間的亂數，然後分別除 8 取商及餘數當作 row 及 col 的位置。

8x8 點矩陣 LED 顯示器如何顯示及腳位請參考實驗 6 及附錄 1。因為蛇跟食物分別採用兩種顏色來顯示，而紅色跟綠色的 LED 燈其 Row1 至 Row8 是共用的，也就是說如果要用兩種顏色的 LED 來顯示，當設定好要顯示列時，同時要分別設定紅色 LED 的 Col 的值以及綠色 LED 的 Col 的值。

提示：可設一個變數來記錄目前的方向，當沒有按上、下、左、右時，蛇就會依原來記錄的方向持續移動，如果有按下其中一個方向，就改變記錄方向的變數。

二、題目二：二十一點

兩個人玩二十一點，按 Reset 重新開始一局，首先兩人分別按一按鈕發底牌，然後按另一個按鈕加牌。每加一次牌，要將加牌的總數累加顯示出來，雙方根據已顯示的點數來決定是否繼續要牌，如果顯示的總數已大於 21 點，則不能再要牌，如果雙方都決定不再要牌時，再按另一顆按鈕決定誰贏，並同時以 12 顆 LED 顯示跑馬燈顯示誰贏或是平手。

每一個人有三顆七段顯示器來顯示資料，其中一顆用來顯示底牌以及最新發的牌，用一指撥開關來切換要顯示底牌或是最新發牌。另外兩顆顯示除了底牌外的所有牌點數總合。一個指撥開關切換要顯示底牌或最新要的牌。二個按鈕，一個要底牌，一個要加牌。整個系統額外有二個按鈕，一個 Reset 按鈕以及比結果的按鈕。當要底牌的按鈕按下得到底牌數字後，如果再按要底牌的按鈕，不能讓底牌數字變動。要牌時如果會有彈跳現象，導致一次要兩張牌，可參考實驗 8。

提示：產生發牌亂數的計數器，使用的計數頻率要低，太快會有問題。

三、題目三：擲骰子

兩個人分別擲三顆骰子，按底下所列遊戲規則論贏。

規則一：如果兩人的數字組合都相同，則平手。

規則二：三數目相同>二數目相同>均不相同。

規則三：如果雙方都是三數目相同，則數字大者贏。

規則四：如果雙方都是二數目相同，則相同的數字大者贏，如果相同的數字一樣大，則不相同的數字大者贏。

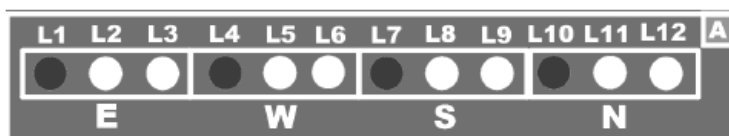
規則五：如果雙方都是三數字均不相同，則以總合加起來的數字大者贏，如果加起來數字一樣大，則平手。

首先按 **Reset** 重新開始一局，每個人的三個數字分別用三顆七段顯示器顯示。一開始都顯示 0。每個人都有三個按鈕分別對應擲三顆骰子，按一個按鈕則相對應的七段顯示器顯示擲出來的數字。當兩人共六顆骰子都擲完後，自動判定輸贏，並以 12 顆 LED 顯示跑馬燈顯示誰贏或是平手。當某個骰子擲出數字後，如果再按按鈕不能讓數字變動。

提示：判斷輸贏的方式，可將擲出的 3 個點數，轉成 1 個分數，然後比較兩人的分數即可。

附錄1 LP2900 使用各元件之對應腳位圖

Red-Yellow-Green LED



代碼	L1	L2	L3	L4	L5	L6	L7	L8
腳位	Pin7	Pin8	Pin9	Pin10	Pin11	Pin12	Pin13	Pin14
代碼	L9	L10	L11	L12	L1-L12 共同致能			
腳位	Pin17	Pin18	Pin19	Pin20	Pin141			

要使用 L1-L12 時，Pin141 必須設為 1。LED 相對應的腳位輸出 1 時 LED 會亮，反之，輸出 0 則 LED 熄滅。

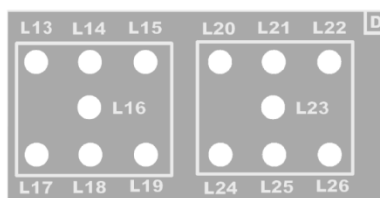
7-Segment Display



七段顯示器代碼	A	B	C	D	E	F	G	DP
腳位	Pin23	Pin26	Pin27	Pin28	Pin29	Pin30	Pin31	Pin32
七段顯示器選擇線	DE1	DE2	DE3					
腳位	Pin33	Pin36	Pin37					

六顆七段顯示器的 ABCDEFG 以及 DP 的腳位均共用。故使用時還需搭配 DE1-DE3 來設定要將數字顯示在那顆七段顯示器。七段顯示器其[DE3 DE2 DE1]的設定值由左至右分別為 000-101。例如要亮第二顆七段顯示器，設定 DE3=0，DE2=0，DE1=1。

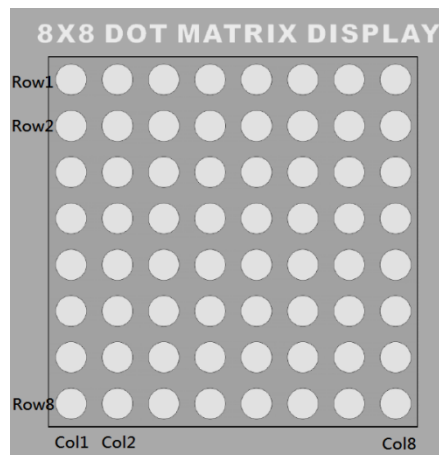
Electronic Dice



代碼	L13	L14	L15	L16	L17	L18	L19	L20
腳位	Pin7	Pin8	Pin9	Pin10	Pin11	Pin12	Pin13	Pin14
代碼	L21	L22	L23	L24	L25	L26	L13-L26 致能	
腳位	Pin17	Pin18	Pin19	Pin20	Pin21	Pin22	Pin142	

要使用 L13-L26 時，Pin142 必須設為 1。LED 相對應的腳位輸出 1 時 LED 會亮，反之，輸出 0 則 LED 熄滅。

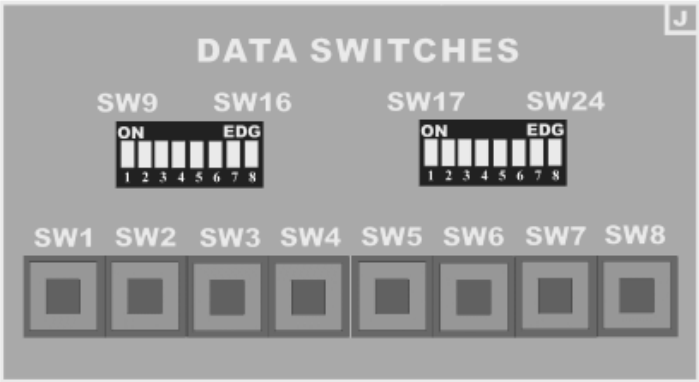
8x8 Dot Matrix LED Display



代碼	Row1	Row2	Row3	Row4	Row5	Row6	Row7	Row8
腳位	Pin88	Pin89	Pin90	Pin91	Pin92	Pin95	Pin96	Pin97
紅色 LED								
代碼	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8
腳位	Pin98	Pin99	Pin100	Pin101	Pin102	Pin109	Pin110	Pin111
綠色 LED								
代碼	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8
腳位	Pin112	Pin113	Pin114	Pin116	Pin117	Pin118	Pin119	Pin120

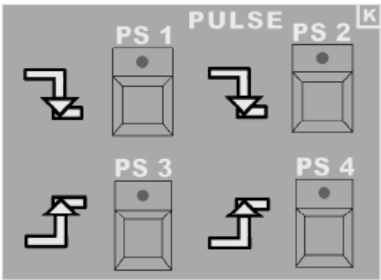
Row1-Row8 設為 1 的列表示該列的點要亮，同時再搭配 Col1-Col8 設為 1 的行表示該行的點要亮。所以 Row3=1，Col4=1 表示第 3 列，第 4 行的點要亮。由於一個點有綠色及紅色兩個 LED，所以如果第 3 列，第 4 行的點要亮紅色，則 Pin90=1，Pin101=1；如果第 3 列，第 4 行的點要亮綠色，則 Pin90=1，Pin116=1；如果第 3 列，第 4 行的點要亮黃色，則 Pin90=1，Pin101=1 且 Pin116=1。

Data Switches



代碼	SW1	SW2	SW3	SW4	SW5	SW6	SW7	SW8
腳位	Pin47	Pin48	Pin49	Pin51	Pin59	Pin60	Pin62	Pin63
代碼	SW9	SW10	SW11	SW12	SW13	SW14	SW15	SW16
腳位	Pin64	Pin65	Pin67	Pin68	Pin69	Pin70	Pin72	Pin73
代碼	SW17	SW18	SW19	SW20	SW21	SW22	SW23	SW24
腳位	Pin78	Pin79	Pin80	Pin81	Pin82	Pin83	Pin86	Pin87

Plus



代碼	PS1	PS2	PS3	PS4
腳位	Pin54	Pin56	Pin124	Pin126

Clock

代碼	10MHz
腳位	Pin55