EECS 2070 02 Digital Design Labs 2019 Lab 2
學號：107062314　姓名：陳柏均

0. 前言

這次 lab 主要要我們用 verilog 實作 counter, 我想這無疑是硬體之中很重要的一部分, 在上學期邏輯設計這堂課就有學過 counter 的運算方法以及其中的道理, 而這次的 lab 幫助我釐清許多觀念, 同時也更加熟悉此硬體語言的精髓。 剛開始還有點怕自己完全不會,但經過了許多努力即奮鬥,我終於能完成這次的 lab。

1. 實作過程

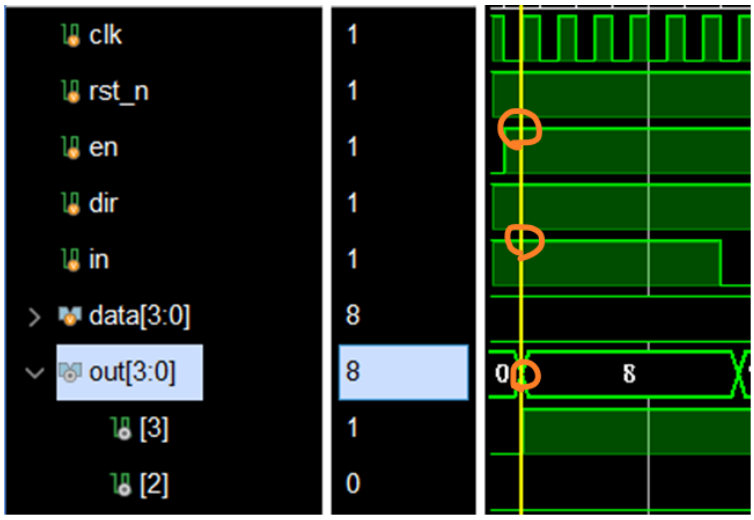
(i)

第一題我沒有花太多時間在打程式上面, 反而是查清楚 counter 的概念, 畢竟已經過了一段時間所以對它有些生疏。 這題我使用的方法主要是在 always block 裡面放 casex 並依照題目所示去判斷各種可能發生的狀況,然後再去給 out 值。

另外當我得知這題要我們寫 testbench 的時候我也是十分慌, 畢竟沒有自己打過 testbench, 非常怕出錯; testbench 是用來判斷我 counter 的 code 究竟寫得對不對, 而一開始我打算列出所有 input 的可能性讓 testbench 去判斷我的答案是否正確, 雖然這一個方法嚴謹, 然而同時也花了我許多時間, 起初認為這是這題 testbench 唯一的解法, 後來才發現並不是。

後來我改了 testbench 的寫法, 我從原本列出所有可能值變成隨意去拉動一個 input 的值,進而去判斷 output 以推論這個功能到底有沒有起作用, 如此一來既可以避免 testbench code 的過度冗長, 也可以快速從 wave 中間接得知自己是否有出錯。

舉個例子, 如圖, 當 en=1,in=1 時, 照題目要求, out 的值變成 data 的值 8, 而同理可用這樣的方法去測每一種功能是否正常。



(註:用 Vivado 的 wave 去看每一個功能的效果)

(ii)

第二題也就是 lab2_2，我認為是花我最多時間的，以前在畫 kmap 的時候其實就有悟出 Gray code 中的一些小重點，就是鄰近的兩組數字只會差一個 bit，這明顯跟普通的 counter 有所差別，因此多了不少挑戰性。

我認為這題主要有兩個重點，第一個如題目所說是先做一個 1 bit 的 gray code counter 再去用它做 2 bits 的。 而第二個我認為是 binary code 跟 gray code 的轉換，原本我還打算用 state 的改變去進行 counter 的前進，但後來上網查有尋找到兩個之間的關聯，因此後來決定用了比較能統一的方式，省去一些有共通點卻又不共同判斷的狀況。

最後當在接兩個 gray code counter 時，由於當前四位(gray[3:0])在進行往上跑的動作時，後四位必須保持不動，而由題目給的參數中可以運用 cout 這一個功能，當在做 2 bits 的 counter 進行連接時，不難發現只要將第一個 bit 的 cout 給第二個 bit 的 en 就可以達到我們所需要的目的。程式碼大致如下:

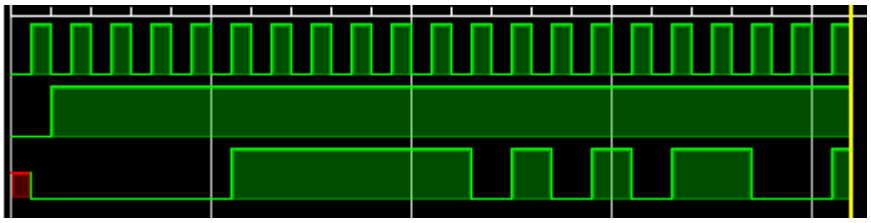
```
wire tmpcout;
counter_1 first(.clk(clk),.rst_n(rst_n),.en(en),.dir(dir),.gray(gray[3:0]),.cout(tmpcout));
counter_1 second(.clk(clk),.rst_n(rst_n),.en(tmpcout),.dir(dir),.gray(gray[7:4]),.cout(cout));
```

(註:將 gray code 第一個 bit 的 cout 給第二個 bit 的 en)

(iii)

最後是這次 lab 的 bonus 題，也就是 lab2_3，對於這一題題意的理解我看了助教給的網站許久，後來經過同學的解釋還有 demo 時助教的更加釐清，我才了解到這題的設計理念無疑是製造出一個簡單的亂數器。 此題我打的主要方法是看圖進行設計，將零到四位往後移動一個，並將第零位填上原本第零位和第五位的 xor。

這題也是要我們自己實作自己的 testbench 去進行測試,而我這題也只有對 rst_n 進行少許變動而去看 wave 的變化。畢竟當 rst_n 為零時, F 即為 6'b000001, 而經由 shift 跟 xor 即可以產生一個假的亂數,如下圖。



(註:out(最下面那個)便類似一種亂數產生器,一時間看不出什麼特別的規律)

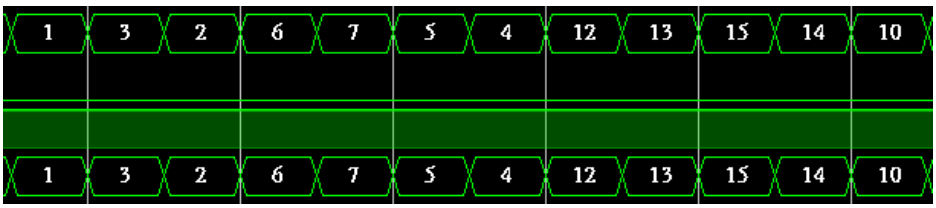
2. 學到的東西與遇到的困難

(i) 波形圖的方便性

以前在打 C 或 C++等軟體語言時，在 debug 時主要都是依賴編譯器的除錯功能或是用 printf, cout 這些功能進行 debug，但 verilog 有時比較難這樣子做;畢竟它是硬體語言，大部分$display 其實沒有那麼大的用處，因為硬體程式往往是一同執行，雖說還是有一些軟體的特性，但主要的思維仍有所差距。

但打程式時再怎麼厲害的工程師也會出錯，而當自己錯的時候不斷重複盯著自己所打出來的 code 也不是個辦法，效率也沒有很高，而 verilog 的主要 debug 工具我認為就是 wave，由這個功能可以快速看出自己到底是在哪一個地方出錯，才能針對自己開始錯誤的地方進行改正。

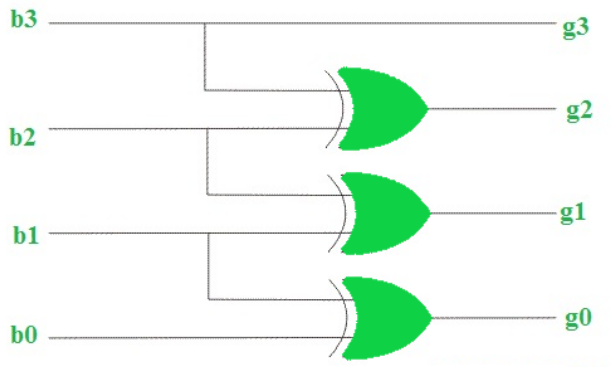
如在 lab2_2 中即可以將自己的 counter 跟 testbench 的數字進行快速比對，十分方便。



(註:由 wave 可看出自己的答案對應是否正確無誤)

(ii) Gray code 的換算

在 lab2_2 中，我不但學會了 Gray code 的特性，也得知了 Binary code 以及它的轉換方式。 我上網找到 Binary code 轉 Gray code 即是最前面的 bit 不動,後面的 bits 分別是兩兩 xor 之後的答案。如圖。



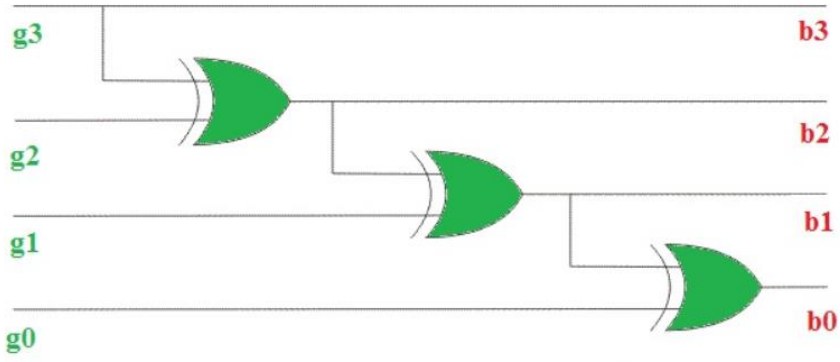
(註:Binary code 轉 Gray code 概念圖)

```
bnum=bnum+1;
g3=bnum[3]^bnum[2];
g2=bnum[2]^bnum[1];
g1=bnum[1]^bnum[0];
gray={bnum[3],g3,g2,g1};
```

(註:Binary code 轉 Gray code 的程式碼)

但一開始我一直認為兩種轉換是互通的，因而當我在進行 Gray code 轉 Binary code 時，我竟用了一模一樣的方法去接。我不斷的看 wave，即使當時看到我自己錯在哪，還是遲遲想不通為什麼答案竟是錯的；後來仔細用題目所給的表去進行換算，才赫然發現沒我原本想向那麼簡單。

由 Gray code 轉回 Binary code 的方法稍顯不同，比起前者，後者比較像是由高位一直進行串接，用此方法方能將值轉回 binary 的形式，因為對於我個人而言，我是採用 binary code 來進行加減(counter)的動作，然後再將其轉換回 Gray code，以此類推。



(註:Gray code 轉 Binary code 的概念圖)

```
bnum={before[3],
before[3]^before[2],
before[3]^before[2]^before[1],
before[3]^before[2]^before[1]^before[0]};
```

(註: Gray code 轉 Binary code 的程式碼)

(iii) rst_n 放在 always 中

同樣在 lab2_2 之中，曾經有出現一個錯誤我也是除錯了一段時間，那就是當我在寫 always 時，後面只想到要放 negedge clk，在執行程式的時候出現了不計其數的 x，即 don't care，檢查了許久都不知問題所在感到有點沮喪。後來非常仔細看 wave 才發現 rst_n 其實應該也要跟著放進去，如此一來在這一題才能發揮它應有的作用。

```
always@(negedge clk )
    begin
    bnum={before[3],befor
    if(rst_n==4'b0000) be
        gray=4'b0000;
        bnum=4'b0000;
```

(註:程式碼的考慮不周全)

```
[NOT_PASS_1] : GRAY1 :  x, GRAY0 :  x, num :          6, cout : x, num_c :          0
[NOT_PASS_1] : GRAY1 :  x, GRAY0 :  x, num :          5, cout : x, num_c :          0
[NOT_PASS_1] : GRAY1 :  x, GRAY0 :  x, num :          4, cout : x, num_c :          0
[NOT_PASS_1] : GRAY1 :  x, GRAY0 :  x, num :          3, cout : x, num_c :          0
[NOT_PASS_1] : GRAY1 :  x, GRAY0 :  x, num :          2, cout : x, num_c :          0
[NOT_PASS_1] : GRAY1 :  x, GRAY0 :  x, num :          1, cout : x, num_c :          0
[NOT_PASS_1] : GRAY1 :  x, GRAY0 :  x, num :          0, cout : x, num_c :          1
[NOT_PASS_1] : GRAY1 :  x, GRAY0 :  x, num :          0, cout : x, num_c :          0
[NOT_PASS_1] : GRAY1 :  x, GRAY0 :  x, num :          1, cout : x, num_c :          0
```

(註:思慮不充足而造成一連串 x 的出現)

3. 想對老師或助教說的話

這次出的題目我認為是很有挑戰性的，前兩題的概念更是環環相扣。助教出這樣的題目大概是要我們釐清 counter 的很多概念吧。我覺得第一題就直接銜接第二題是有幫助的；雖說我認為第二題的難度大幅度的提升，同時也花了我諸多時間在除錯上面，但我知道這也是學習的一部分。

第三題的題意我花了許久才理解，一開始不懂它的具體用意，也覺得它跟前兩題的關聯性好像沒到那麼大，但很高興能多增加一個知識，讓我理解到一個我從未聽聞過的亂數產生器方法，以前總是不懂亂數產生的方法，寫了這一題之後，我才稍稍體認到簡單的亂數產生方式。

經過這次的 lab2，我知道我的知識又多長進了一成，但不可否認的是我還有很多要學，未來的路上還有很多果實等著我去收穫。 我從一開始碰硬體語言必須每一個細節都去問別人，到如今我能自己打出來自己 debug，當然有時還需他人的幫助找出自己忽略的那些盲點。

不能不承認 lab 的過程中遭遇了無數挫折，甚至還有一次幾乎改了自己全部的程式碼；表面上好像非常浪費時間，但我相信那些錯誤也是我未來的借鏡，使我在以後減少錯誤率。困難與挑戰將會是眼前無可避免的高牆，但也唯有克服它，方能使自己更上一層樓。