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

1. 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。

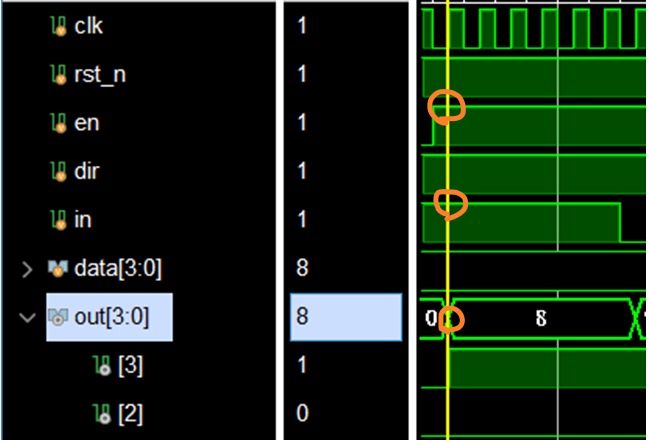
1. 實作過程

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。

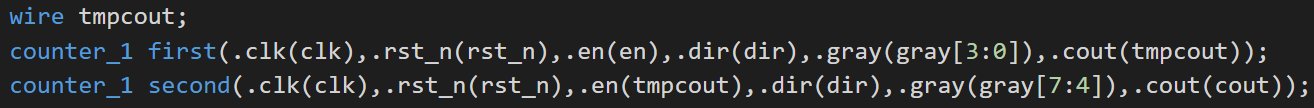


(註:用Vivado 的wave去看每一個功能的效果)

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:

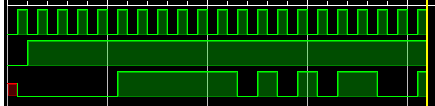


(註:將gray code 第一個bit的cout給第二個bit的en)

(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。



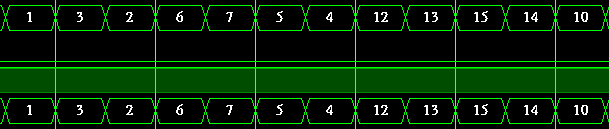
(註:out(最下面那個)便類似一種亂數產生器,一時間看不出什麼特別的規律)

1. 學到的東西與遇到的困難
2. 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。



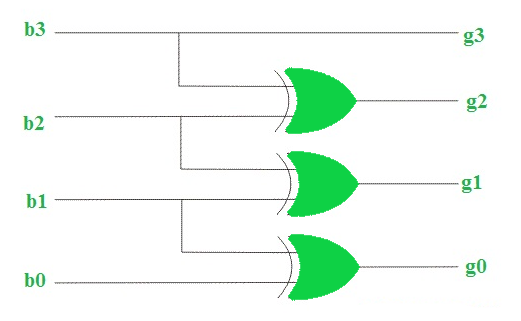
(註:由wave可看出自己的答案對應是否正確無誤)

1. Gray code的換算

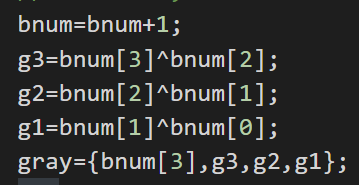
在lab2\_2中, 我不但學會了Gray code 的特性, 也得知了

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。



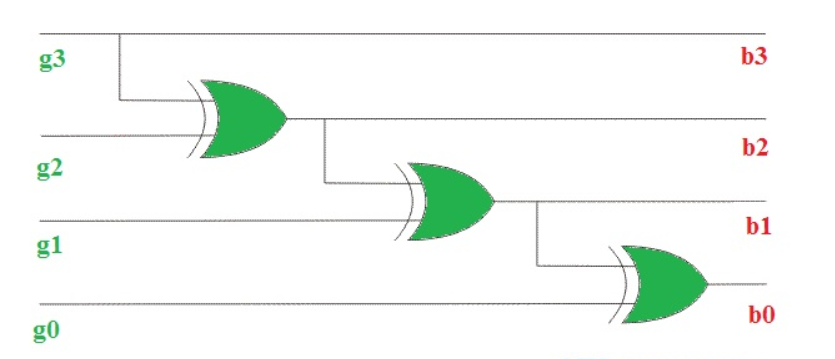
(註:Binary code轉Gray code概念圖)



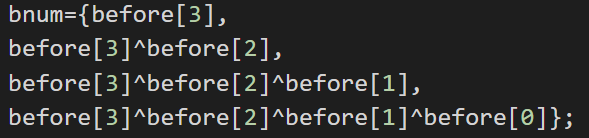
(註:Binary code轉Gray code的程式碼)

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。



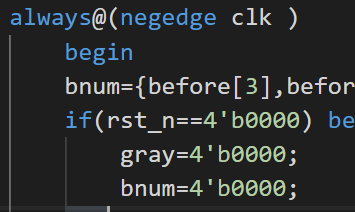
(註:Gray code 轉Binary code的概念圖)



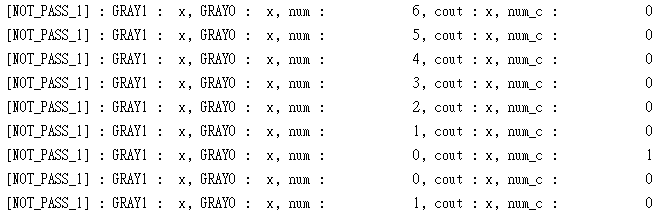
(註: Gray code轉Binary code的程式碼)

1. 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。



(註:程式碼的考慮不周全)



(註:思慮不充足而造成一連串x的出現)

1. 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。