EECS 2070 02 Digital Design Labs 2019 Lab 3
學號：107062314　　姓名：陳柏均

0. 前言

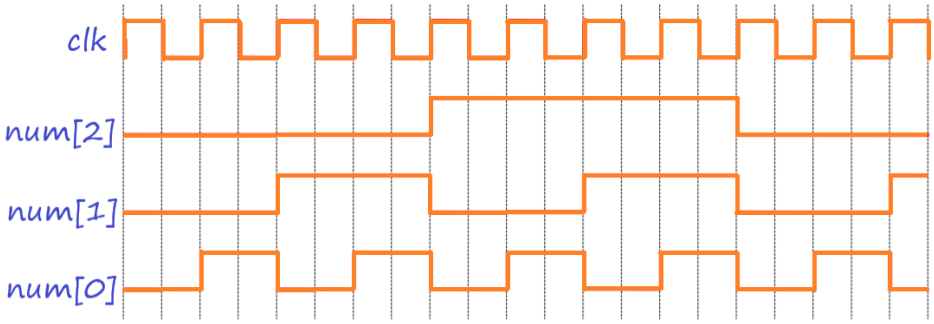
此次 lab3 為第一次把自己的想法燒在板子上，我覺得這次 lab 與前兩次明顯不同，不只要用到 FPGA 板，還多了一些十分有趣的概念。另外，這一次 lab 的 1,2 題我認為相對較簡單，而第 3 題在我看來就相對比較有挑戰性，使我經歷了許多阻礙，但同時也是此讓我學到了更多。

1. 實作過程

(i)

第一題要求我們做 clock divider，由於這部分課程的講義上有說明，加上附圖使我更加理解它的意義和道理。以前從來沒想過 clock divider 竟然是這樣做，我認為這方法既簡易又好懂。

雖說如果要我自己想出 clock divider 的實作方法的話，我應該要思考一段時間，但其中的道理是好理解的。我跟講義上打的方法一樣，使用 num 一直去累加的特性就可將想要的 clock 直接做出來。當想要去除以 2^n 的時候，就將 num 的第 n 個 bit 回傳, binary 的特性即可以讓 clk 跳起來(變成 1)的頻率降低。



(註:clock divider 的概念圖)

(ii)

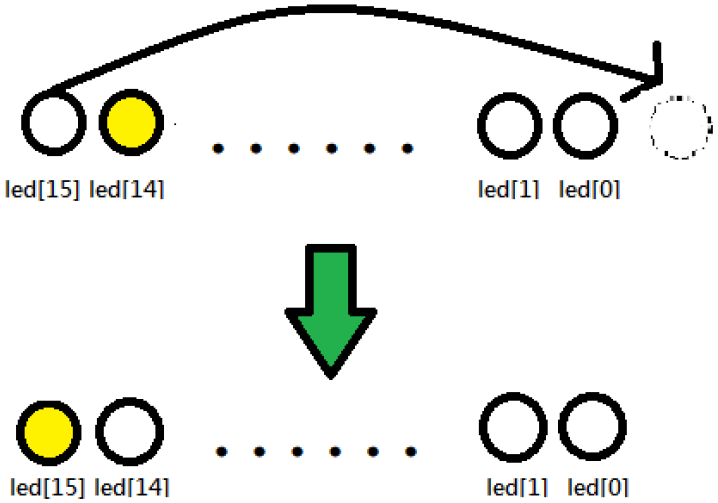
第二題是要運用第一題的 clock divider 使 LED 燈去移動，我使用了 always block 和 if else 去判斷各個輸入的狀況。而移動的部分我則是讓下一個 LED 的輸出用上一個 LED 直接去調整。

例如題目要求當 dir 為 1 時，便將 LED 由右往左移，我思考了一段時間才有靈感, LED 總共有 16 個 bits，而要做出類似移動的效果，我讓後來的 led

其 14 到 0 位為新的 15 到 1 位，而新 0 位的值即原本的第 15 位。

```
if(dir==1'b1)
begin
    led={led[14:0],led[15]};
end
```

(註:dir 為 1 時 led 的變化狀況)



(註:dir 為 1 時移動的示意圖)

(iii)

第三題出現了 Mr.1 跟 Mr.3 兩種 LED 燈之移動方式，這題我運用的方法跟上一題差不多，但我認為這一題的主要關鍵點在速度的問題, Mr.1 和 Mr.3 在改變方向時發生了速度的交換，這是我在這一題遇到最大的困難，因為我一開始想不到到底如何將一樣的速度指定給不同的人(兩組 LED)。

起初我沒多想就全部寫在一個 always block 之中，後來我是使用了 always block 並且以 dir 為觸發條件去寫，每當 dir 有所變動的時候，我就直接把 Mr.1 及 Mr.3 的速度分別給他們，這樣我就可以清楚明瞭的規劃我要寫的東西,較不會使自己搞混且對於硬體語言的執行也較佳。

```
always@(dir)
    begin
        if(dir==1'b1)
            begin
                mr1clk=bigclk;
                mr3clk=smallclk;
            end
        else
            begin
                mr1clk=smallclk;
                mr3clk=bigclk;
            end
    end
```

(註: dir 變動時將速度給 mr1clk 跟 m
mr3clk)

由於這題有兩組 led 在動，但在 FGPA 板上卻只有一種顯示方式，因此到最後我將自己所設的 mr1pos 跟 mr3pos 「or」起來，以得到我要的結果；製造出在板子上像有兩組的 led 燈同時在跑的現象，其實對於這個 FGPA 板而言，它只看 led 給的值去做亮暗燈的變化，而經由人的思維用 or 去呈現自己想要的效果我個人主張是十分有趣的。

(iv)

第四題也就是這一次 lab 的最後一題，我想無疑是四題之中最具有挑戰性的，但以我看來也是最耐人尋味的一題。 上題主要重點在速度的變化，而較少注重 Mr.1 以及 Mr.3 的相對關係；而這題明顯就有。 起初以為這題跟上題差不多，只要多加幾個判斷條件即可，但在實做之後，才發現非我原本想像的那樣。

第四題多出了碰撞這一個因素就增加了不少難度，助教所給的碰撞情形分析的確有幫助。一開始我使用的方法純粹是當 Mr.1 跟 Mr.3 有交集時(即兩個的 position 值 and 起來不為零)，就讓 Mr.1 的下一次的方向去改變，但使用此方法遇到了很多的問題，包括無碰撞即離開和穿越等等狀況。

後來去了助教時間，認知到了我的打法的可能錯誤性，因為這有牽涉到時間(clk) 的問題，我若沒有審慎去思考硬體的執行方式，即有可能出現就算我的想法是對的，跑出來的結果仍嚴重不如我的預期。 之後我便聽了助教的建議，使用較直接的辦法，每當出現碰撞的時候，我便直接改變它的位置，也就是說，如果原本 Mr.1 是由右往左，而當它碰到 Mr.3 的時候直接將其位置往右移，讓它馬上做出我想要執行的想法，避免有 delay 或是其它無法預期的情況。

而對於碰到中間的情況我則是在助教的引導下加上我自己原本的想法，利

用了 past 與 now 去記住上一次所發生的情況並判斷接下來該往哪裡去移動。這個部分我真的思考了許久才明白，我認為這非我輕易能理解的，但我相信多打幾遍之後，可以愈加熟悉，且在以後頭腦能動得更快更靈活。

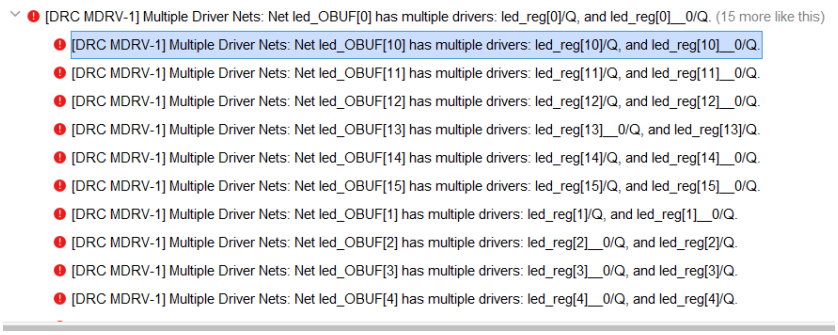
```
3'b011:
    begin
      nextmr3pos={mr3pos[0],mr3pos[15:1]};
      if((mr1pos & mr3pos)!=0)
       begin
          if(past==1'b1)
              begin
                  now=1'b0;
                  nextmr1pos={mr1pos[0],mr1pos[15:1]};
              end
          else
              begin
                  now=1'b1;
                  nextmr1pos={mr1pos[14:0],mr1pos[15]};
              end
       end
          else
              begin
                  now=1'b0;
                  nextmr1pos={mr1pos[0],mr1pos[15:1]};
              end
    end
```

(註:當 Mr.1 跟 Mr.3 發生碰撞時 ,用條件去給它下一個位置)

2. 學到的東西與遇到的困難

(i) always block 中不可出現重複給值

在打這次 lab 的時候，我仍有許多次不小心在不同的 always block 中給同一個東西值，這大概是因為在打軟體的時候總是會這樣操作，也認為理所當然，但當轉成硬體程式時，我卻常常無法注意到一些限制的細節，畢竟硬體的程式到最後必須在 FPGA 板子上呈現，它有許多的限制也是有它的道理在，我想，也只有多加去熟知它，才能有所進步。 希望能從降低錯誤到不會有這類的語法差錯，未來更加注重在硬體語言的思維操作上。



(註:對同一個 reg 但在不同的 always block 中給值而導致 Vivado 執行時出現錯誤的訊號)

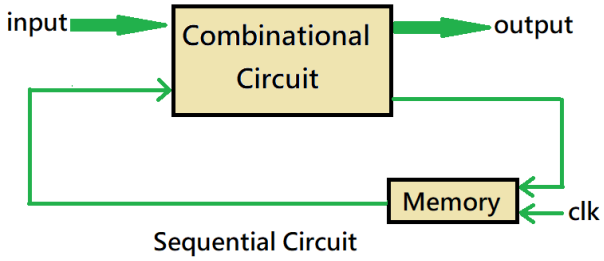
(ii) Combinational circuit 和 Sequential circuit

以前在邏輯設計的課時，便知道 combinational circuit 以及 sequential circuit 的主要差別，但每次在做 verilog 的時候，原本都將其

混在一起，想到什麼就打什麼，沒經過特別的細想，而如此一來常常形成了許多我不可解釋的現象發生，或是沒照我設想的方式去走。
雖說了解 combinational circuit 及 sequential circuit 的用法，但是其實我此次才真正理解到在 verilog 中這兩個分開的極大好處。

Combinational circuit 主要就是不斷的進行運算，輸出只和輸入有關，而 Sequential circuit 則會輸出則會看輸入以及當前電路的狀態。Sequential circuit 大多都是由 D flip-flop 作為 register 所組成，而 D flip-flop 每遇到 clk 跳動就會把輸入點的資料記錄下來，等待下一次 clk 跳動。

我認為網路上有一個比喻使我更加容易理解，那就是:
Combinational circuit 就像馬路，負責讓車流動；而 Sequential circuit 就像紅綠燈，負責管理何時能通行。



(註: Combinational Circuit 及 Sequential Circuit 簡易圖)

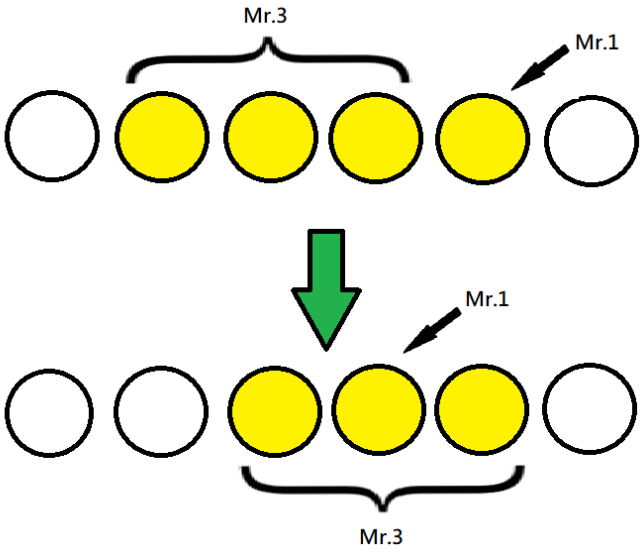
(iii) 碰撞的第二種狀況原因

在最後一題中，即 lab03_3 中，助教有給三種碰撞的方式，而一開始沒仔細思考時，完全不知會有 Mr.1 和 Mr.3 相交在中間的情況。因為單純以物理的角度來看，在中央碰撞是不可能的。但在此卻是能發生的。

而一旦細細地深入思考以及觀察 LED 燈在 FPGA 板上的移動，其實不難發現只有當 Mr.1 由右往左去碰撞 Mr.3 時，才有可能出現第二種狀況，由左往右則不會出現。Mr.3 的方向是往右，而 Mr.1 與它同向 Mr.1 速度較快看似從後面追 Mr.3 的感覺，因此不會出現重疊在中間。

當 Mr.1 由右往左時，由於兩個 clk 的差異，某些狀況會導致兩個同時到(Mr.3)中間的 LED 燈位置，而這就是第二種情況的發生的原因。我相信這個部分也是助教要我們去思考的地方，更是這一題的主要重

點。



(註:會出現第二種 Mr.1 和 Mr.3 重疊在中間的原因)

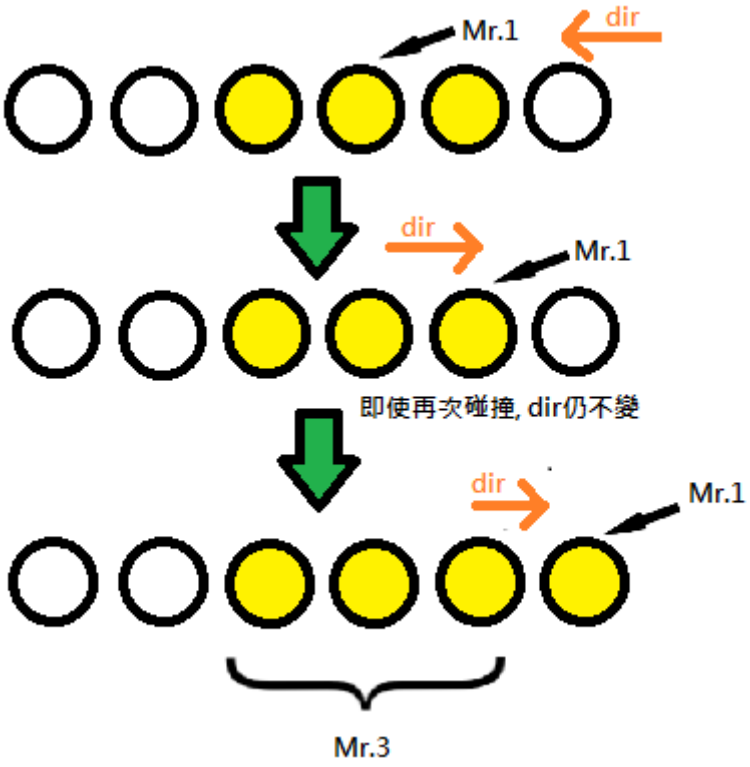
(iv) 第二種狀況之解決方式

如我上面所述，當我遇到第二種狀況時，我便直接指定 Mr.1 的下一個位置，但另外還有一個問題，那就是當碰到中間的那個當下，我又如何知道原本 Mr.1 是從何而來。

若不去仔細判斷，將有可能出現 Mr.1 卡在 Mr.3 之中的狀況，過一段時間才會因為 clock 之差讓 Mr.1 再次離開 Mr.3，在實做的過程中我便有遇到這樣的狀況；而這自然非題目所要我們達到的目標。

因為每次它都會去判斷如今的位置是否有交集。 雖說理論上只會從右邊碰撞，但為了 code 的嚴謹性以及對稱性，我在左右兩種情況都加了判斷。 我使用了兩個 reg，分別是 past 跟 now 去解決。

當現在有交集時((Mr.1pos & Mr.3pos)!=0)，我便先判斷 past 為多少才去指定它的下一個應在的地方；而 past 則是表示上一次是否有碰撞，而當這次又再次碰撞時，我便仍照原本的方向去跑。而這一個方法也是使用了 Flip-Flop 的概念, now 即 nextpast 的概念。



(註:在中間碰撞時的主要示意圖)

(v) 用大括號給值

在給 Mr.1 以及 Mr.3 位置值時，曾經遇到了一個小問題，以我個人習慣，後來在給長位數且高重複性的電路時，我會盡量不將全部的位數狀況寫出來，我偏好使用倍數的方法去表示。

而剛開始在使用的時候，出現了 syntax error，我想了許久後才發現這種方法必須在裡面再多加一組括號。 由於對這種語法的不夠通曉導致了錯誤，但我認為這樣的方法十分地好用，我也差不多熟悉了它的用法。 下次若有需要用到也會善加利用這一個功能，省去打出全部 bits 還得細細數的困擾跟麻煩。

```
{1'b1,15{1'b0}};
{3{1'b1},13{1'b0}};
```

```
={1'b1,{15{1'b0}}};
{{3{1'b1}},{13{1'b0}}};
```

(註:一開始沒加括號導致語法的錯誤)

3. 想對老師或助教說的話

這次的 lab3 是我目前碰到的 lab 之中最有趣的一個，除了題目的新鮮感之外，主要原因我猜應該是這次終於有接板子了。 以前(去年)就有看過學長姐們再做這個，但當時的我總是不能理解這個板子到底在幹嘛。

後頭才發現這一個小小的 FPGA 板子其實真的可以做許多事，更可以做出許多的遊戲。 它的功能遠遠超過我對它的既定印象(順帶一題，上網查後發現，其實它的價錢也遠遠超出我的想像)，剛開始看到這個板子覺得它外觀普普通通，好像也不能拿來幹嘛，但之後才明瞭我誤會它了。

我認為學 verilog 之後將 code 呈現在板子上，對於我個人而言成就感是具相當程度的，這次的 lab 我主要卡在第四題，而錯誤的主要原因除了題目較有挑戰性之外，一部分也是因為我 code 的規畫問題。 我認為在打硬體語言時，縝密及嚴謹的邏輯思考對於成功執行是充分且必要的。

另外，我也覺得與其看到題目便埋頭開始打 code，真的不如先仔細思索並將構思的做法寫在紙上。 這樣當後面遇到 bug 的時候，我堅信也能幫助自己有邏輯性的去找出自己所犯之錯。

這次的題目由於時間的關係多少帶給我一些壓力，尤其是最後一題我還去了助教時間詢問助教，也很感謝助教的細心引導及建議。雖說有一定的挑戰性，但我必須說我還是有從這次 lab 中學習新知識和得到許多樂趣的!

最後，今天(我打 report 這天)是國慶日，在這裡講個笑話好了:
一對夫妻在婚前約定好，家中小事情由妻子決定，大事情由老公決定。 某日老公輕聲細語地對老婆說:「老婆我有件小事想跟你商量」
老婆:「什麼小事?」
老公:「我想娶小老婆」
老婆:「什麼! 這種大事你居然好意思跟我說是小事!」
老公:「好! 你說這是大事，大事由我決定!!!」。

雖說等您看到時也許時間已過了，但還是祝老師及助教們國慶日快樂!