Lab4 Report

4-1. Binary 1Hz Adder

●Design Specification

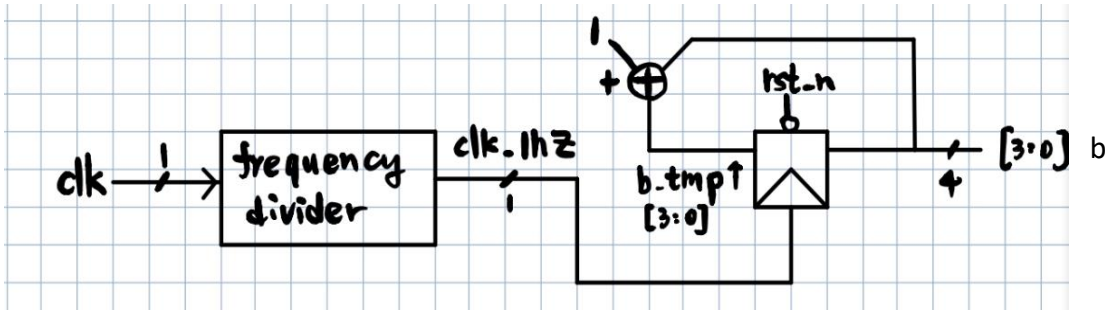
1. Input: clk、rst_n
2. Output: [3:0]b
3. Logic Function: b_tmp=b+1'b1;

●Design Implementation

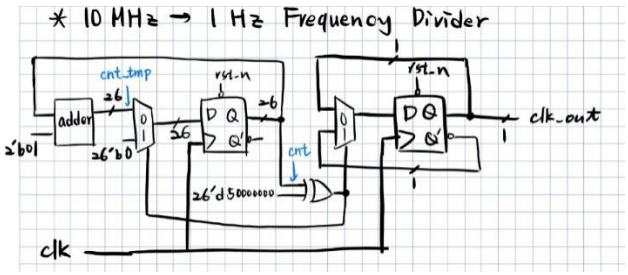
1.Outline

這個實驗主要是要用 LED 燈來顯示每一秒往上加的 Binary Adder。只要利用一個 b 持續加一，然後每次 b 變動的時候再讓 b_tmp 等於 b+1'b1，再把每個 b 接上相對應的 LED 燈即可。

2.Logic Diagram



a. Frequency divider 是要利用 XOR 的特性，當原本的 clk(100MHz)震盪 50M 次的時候讓會讓 XOR 的 output=1，讓後面的 T flip flop toggle，會變成整體頻率剛好是 1Hz 的 clk_1hz。



b. 將 clk_1Hz 接上 D flip flop，持續讓 b_tmp 每一秒加上 1，再等於 b，把每個 b 的 bit 分別接上 LED 燈(如下表)，即可完成。

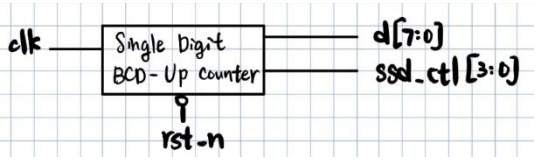
I/O	B3	B2	B1	B0	Rst_n
PIN	V19	U19	E19	U16	W15

●Discussion

我原本都很習慣再 always@(posedge clk_1hz or negedge rst_n)放入 if-else statement 來表示 up counter 的效果，感覺這樣比較直覺，但是再看馬教授第三章的講義時候才發現，我這樣寫其實不太好，因為這樣是把 up counter 埋進去 D flip flop 裡面了，如果之後程式比較複雜會很容易找不出 bug 在哪裡，所以要從簡單的題目就養成好習慣，把不同的功能分開寫，不要讓 code 看起來非常複雜。

4-2. BCD 1Hz UP Adder

●Design Specification



1. Input: clk、rst_n

2. Output: [7:0]d、[3:0] ssd_ctl;

3. Logic Function: b_tmp=b-1'b1;

(clk 是板子上原本的 Fcrystal; rst_n 是 reset; [7:0]d 是用來控制 ssd ; ssd_ctl 是用來控制各個 ssd 要亮不亮的)

●Design Implementation

1.Outline

這個實驗主要是要把 1Hz 的除頻器接上 BCD Up Counter 來做成一個 One Digit 的十進位上數器。

2.Logic Diagram

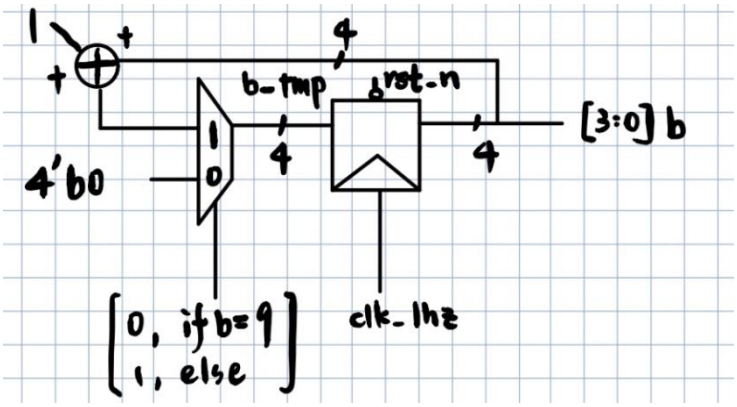
■BCD 1Hz UP Adder

各個 module 會在下面解釋。

a. frequency divider

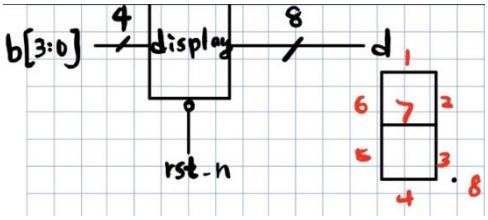
4-1 已經解釋過。

b. BCD up counter

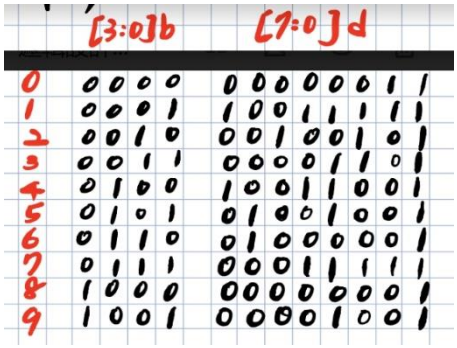


←左圖與之前 binary up adder 不同的地方是在 BCD 沒有是十進位，沒有 9 以上的數字，所以要再利用一個 MUX，而且條件是當 b==的時候要直接把 b_tmp 歸零。

c. display



←左圖是 display，效果是要把 4bits 的 b 轉成可以與七段顯示器一對一對應的 8bits 的 d，decoder 的表在下面，0 代表七段顯示器的亮，1 則代表暗。



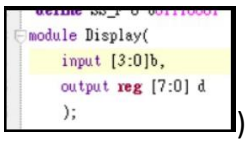
I/O	d[7]	d[6]	d[5]	d[4]	d[3]	d[2]	d[1]	d[0]	Clk	Rst_n
Pin	V14	U14	U15	W18	V19	U19	E19	U16	W5	R2

I/O	ssd_ctl[3]	ssd_ctl[2]	ssd_ctl[1]	ssd_ctl[0]
Pin	W4	V4	U4	U2

●Discussion

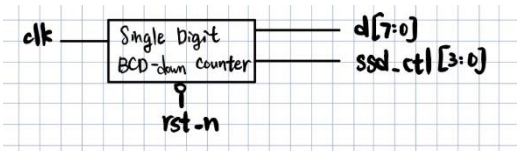
這個實驗我 debug 蠻久的，問題有兩個。

1. 我沒有好好把形式打出來(ex:4'b0000)，而是直接打出=0，所以發生問題了，但是這種問題 vivado 不會跟你說，所以非常難找出來，所以養成好習慣真的非常重要。

2. 這個也是一個壞習慣，就是沒有在每個 module 前面打上相對應的 bits 數(ex: 

)，我原本沒有打出[ ]裡面的 bits 數，所以顯示出來的一直是 8，而沒有變動，我想應該是他原本判定只有 1bits 才會有這樣的結果。

4-3. BCD 1Hz Down Adder



●Design Specification

1. Input: clk、rst_n

2. Output: [7:0]d、[3:0] ssd_ctl;

3. Logic Function: b_tmp=b-1'b1;

(clk 是板子上原本的 Fcrystal; rst_n 是 reset; [7:0]d 是用來控制 ssd ; ssd_ctl 是用來控制各個 ssd 要亮不亮的)

●Design Implementation

1.Outline

這個實驗主要是要把 1Hz 的除頻器接上 BCD Down Counter 來做成一個 One Digit 的十進位上數器。這個實驗與 4-2 非常相似，只有 if 條件的不同而已。

2. Logic Diagram

b

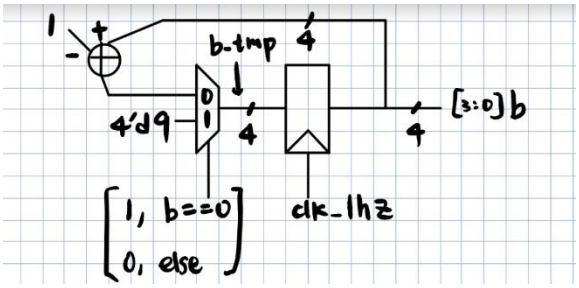
■BCD 1Hz Down Adder

各個 module 會在下面解釋。

a. frequency divider

4-1 已經解釋過。

b. BCD Down Counter



←大致與 4-2 的 BCD Up Counter 差不多，差別是判斷的條件不一樣，當 b==0 的時候 b_tmp 會被 mux 改成 9，造成一個 9~0，又從頭 9 開始倒數的循環。

c. display

4-2 已經解釋過。

I/O	d[7]	d[6]	d[5]	d[4]	d[3]	d[2]	d[1]	d[0]	Clk	Rst_n
Pin	V14	U14	U15	W18	V19	U19	E19	U16	W5	R2

I/O	ssd_ctl[3]	ssd_ctl[2]	ssd_ctl[1]	ssd_ctl[0]
Pin	W4	V4	U4	U2

●**Discussion**

這個實驗與 4-2 大致相同，所以稍微改一些程式碼即可。

4-4. 30s Count Down Counter (stop at 00)

●**Design Specification**

1. Input: clk、rst_n

2. Output: [7:0]d、[3:0]ssd_ctl;

3. Logic Function: b_tmp=b-1'b1;

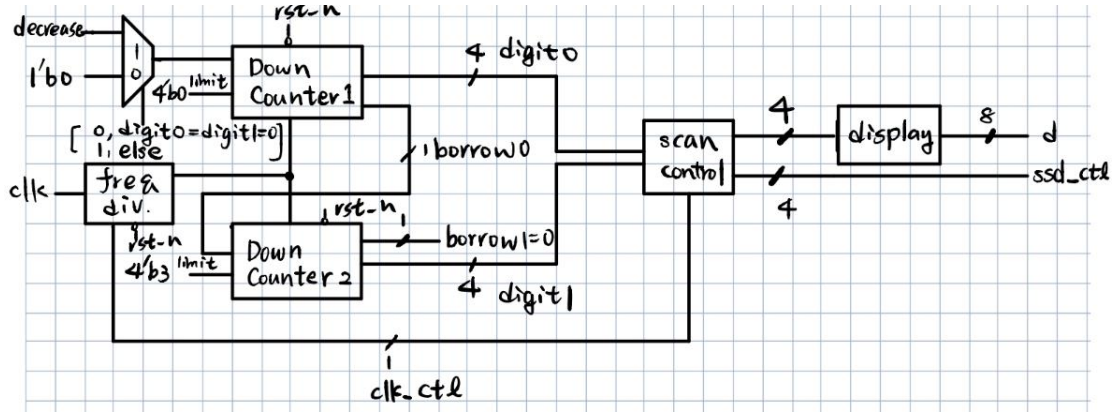
(clk 是板子上原本的 Fcrystal; rst_n 是 reset; [7:0]d 是用來控制 ssd ; ssd_ctl 是用來控制各個 ssd 要亮不亮的)

●**Design Implementation**

1. Outline

這個實驗要做一個 30 秒的倒數器，然後倒數到零的時候就停止了。需要用到 Frequency Divider、decoder counter、scan control、display 四個功能的來完成。

2. Logic Diagram



■**30s Count Down Counter (stop at 00)**

各個 module 會在下面解釋。

a. frequency divider

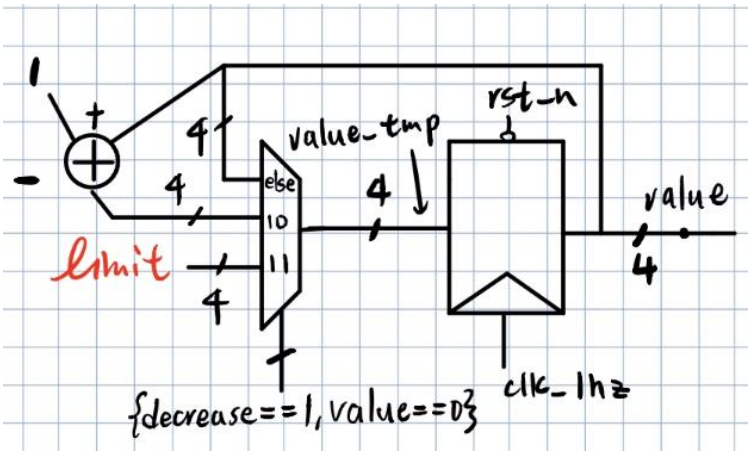
4-1 已經解釋過。

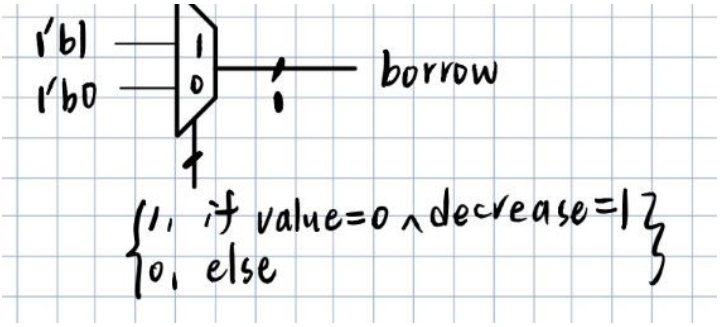
b. display

4-2 已經解釋過。

c. Down counter

這個 Down counter 跟前面的比較不一樣的地方是，他有不只一位數字，所以會需要利用 decrease 還有 value 當作 MUX 的判斷標準。



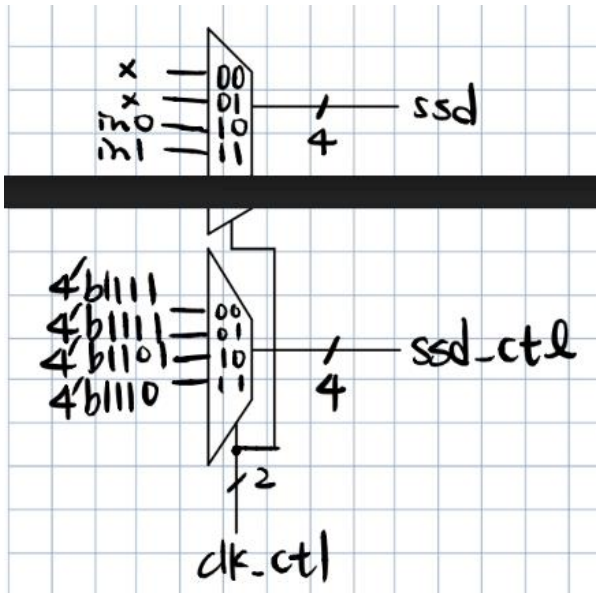


個位數的 Decrease 是 1，而十位數的 decrease 是接上個位數 borrow，而十位數的 borrow 則是 0，因為只有兩位數，沒有百位數可以借。

Limit 則是起始值，也就是十位數是 3，而個位數是 0。

左圖這個 MUX 則是 borrow 的判斷式，如果該位為零且可以向上一位數借的話，

d. Scan control



Scan controll 有兩個 output，兩個 mux 同樣都是利用 clk_ctl 當作判斷標準，上面那個 mux 是要輸出的 bcd 數字，前兩個是 x 的原因是因為該時間的七段顯示器並不會亮，因為題目只需要用到兩位數字。

I/O	d[7]	d[6]	d[5]	d[4]	d[3]	d[2]	d[1]	d[0]	Clk	Rst_n
Pin	V14	U14	U15	W18	V19	U19	E19	U16	W5	R2

I/O	ssd_ctl[3]	ssd_ctl[2]	ssd_ctl[1]	ssd_ctl[0]
Pin	W4	V4	U4	U2

●Discussion

這個題目對我來說並不簡單，又讓我發現很多自己常犯的錯誤。

1. 再用之前的 module 的時候我常常沒有看清楚 bits 數目，而犯了很多像是把 2bits 接到 3bits 的情況發生，但因為在不同的 module 所以不好檢查，所以一定在放入什麼新東西的時候都要看清楚是不是真的能夠成功運作。

2. 這個是讓我覺得最神奇的錯誤，我原本打成”wire b=1’b0”我跑了很多次都找不到錯誤，我還特地跑去問之前修過的同學說是不是 wire 都不能給定值，後來才發現打邏輯設計真的不能用打 c++語言的方式打，一定要用 assign 的才能訂定值，這個小錯誤真的讓我吃盡苦頭……

3. 還有一個是當我成功跑出來之後，當 30 倒數到 0 的時候，會從 99 開始，那時候我才發現不可以直接把 limit 的值直接分別定成 3 跟 0 就好，因為當計數器倒數到 0 的時候，他會跑回在 BCD DOWN COUNTER 的 limit 值裡面，簡單來說就是會從 99 再繼續倒數。我的解決辦法是在外面再加上一個 mux，功能是判斷如果當 digit0 和 digit1 同時等於 0 的時候，把 decrease 改成零，也就會停在 00 的數值，不會再重新倒數了

●Conclusion

我通常都在宿舍裡面打 verilog，但是室友剛好都跟我不同系，所以沒有接觸，大多都要自己想或是上網查資料來解惑。不過最近有一個之前修過邏輯設計實驗的朋友來我房間讀書，跟他聊天之後才發現自己花在邏輯設計的時間是非常長的，聽他說他們通常都一個晚上或是頂多兩個晚上可以把實驗還有結報完成，但是我通常都要花到兩天，而且很多上述的 bug 還都是他幫我看出來的，不然可能會需要花到更久 qq。不過有比較抓到打 verilog 的感覺了，可以感覺出每個語法的實際功能還有特定的打法，希望之後可以更順利完成實驗!