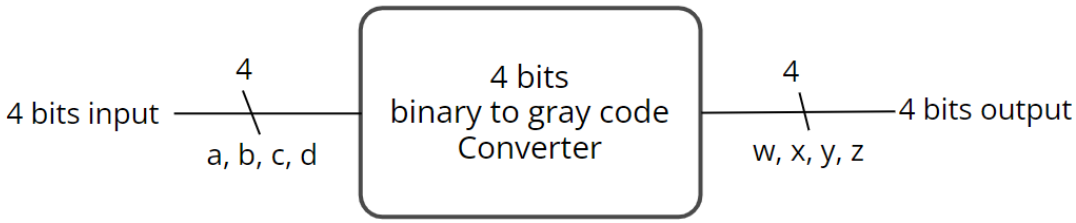
Lab 2: FPGA Emulation

1 For exp1 in lab1 (a signed binary-to-Gray-code converter), finish the FPGA emulation with the following parameters.

Design Specification

IO 輸出入設定:
Input: a, b, c, d 大小皆為 1bit (a 為 MSB)
Output: w, x, y, z 大小皆為 1bit (w 為 MSB)

Logic block



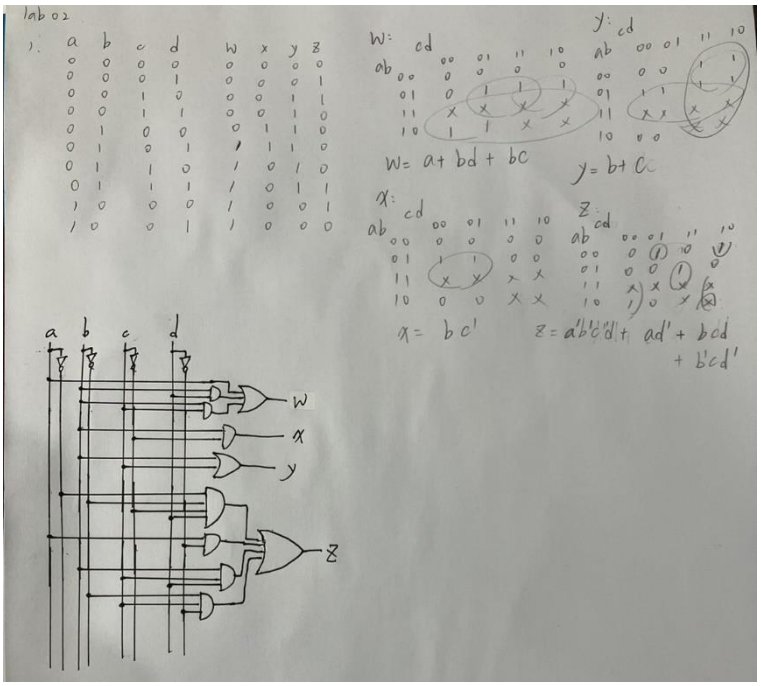
Design Implementation

a. Logic Func and truth table
   truth table 見右圖→
   abcd: 10~15 為 don't care condition

   w = a + bd + bc
   x = bc'
   y = b + c
   z = a'b'c'd + ad' + bcd + b'cd'

b. Logic diagram
   見右圖→

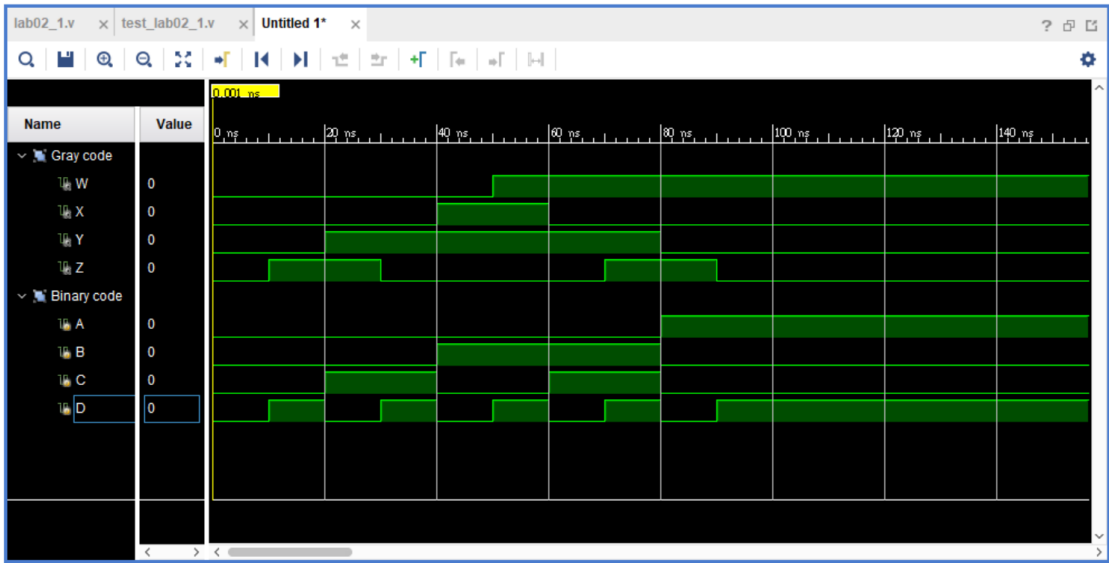
c. Code 請見附件



IO pin assignment

I/O	a	b	c	d	w	x	y	Z
pin	W17	W16	V16	V17	V19	U19	E19	U16

Discussion



程式碼和 testbench 和 lab01_1 一樣，用基本的組合邏輯和一些邏輯運算子操作，將 abcd 10~15 設為 don't care。唯有這次實際將程式灌進 FPGA 裡，並逐一為每個變數分配電路板腳位。腳位的是直接在 IO Planning 處理，雖然可能有點麻煩，對我覺得由 IO Planning 介面去調整較好，就不用怕打 xdc 檔時出現錯誤。

Conclusion

在本次實驗後，我學到了

- 如何分配腳位
- 如何將檔案實際在 FPGA 版上執行

接腳位的部分是我卡最久的地方，因為一直出現 UCIO-1 的錯誤，上網查詢後發現這段錯誤訊息是指邏輯端口沒有被指定具體的物理腳位。但我檢查了半天還是找不到我哪裡寫錯，所以後來改由 IO Planning 去設計便可以了。

Reference

https://docs.xilinx.com/v/u/2013.1-English/ug912-vivado-properties
由 AMD 提供的 UG912 寫了更多詳細的腳位分配的規則。像是 IOSTANDARD，文件也提供了在 VHDL 裡的寫法。往後若有更多關於腳位的問題也可以查看這篇函示庫。

2 Derive a 4-bit binary (i[3:0], i[3] as MSB) to 7-segment display decoder (SSD[7:0]), and also use four LEDs (d[3:0]) to monitor the 4-bit binary number.

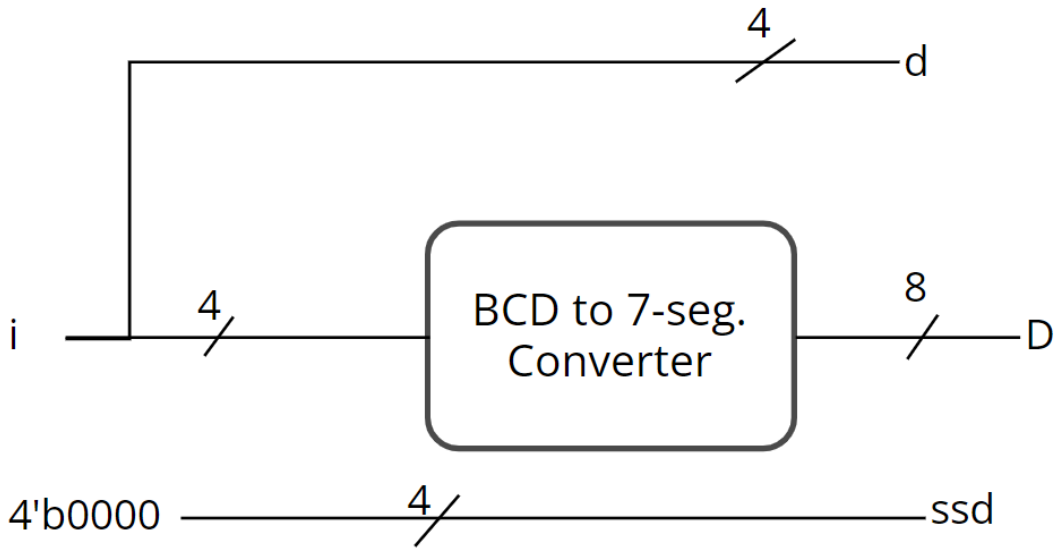
Design Specification

IO 輸出入設定

輸入: [3:0]i, 4bits (左邊 i[3]為 MSB)

輸出: [3:0]d, 4 bits(左邊 d[3]為 MSB), [7:0]D, 8bits(分別代表 7-segement 的 8 個位置), [3:0]ssd(控制 4 個七段顯示器是否顯示)

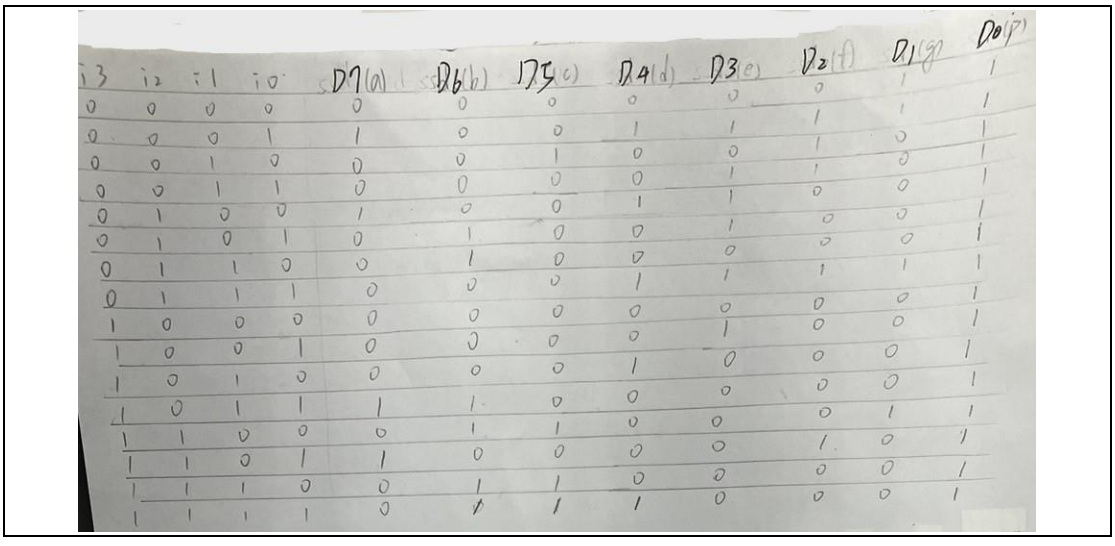
Logic diagram

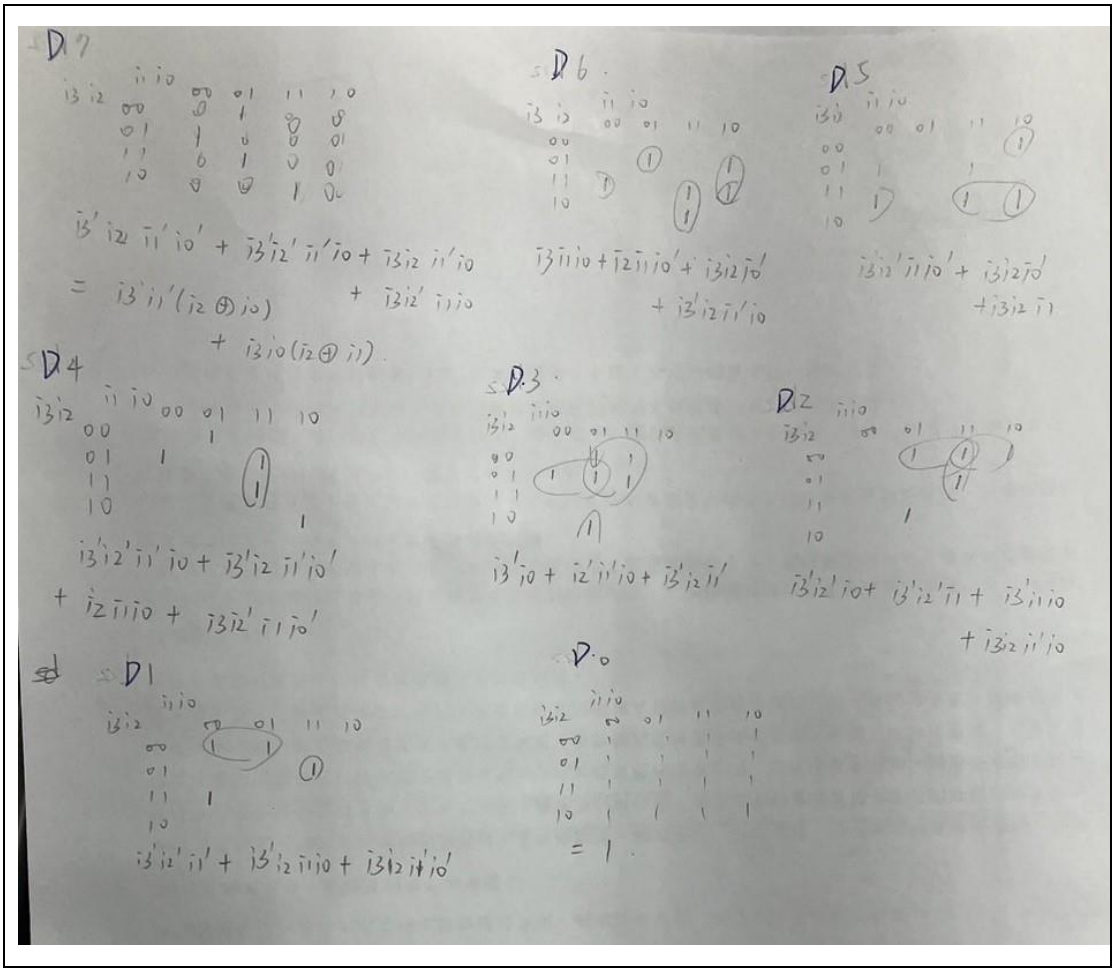


Design Implementation

a. truth table and logic func

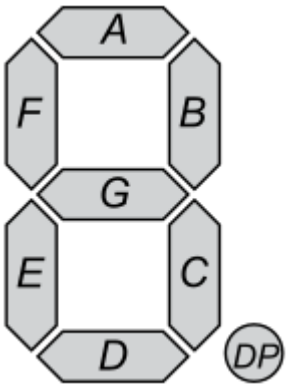
見下面兩張圖





D[8:0] 分別代表 7 段顯示器的 A~G 和 DP，由於 8 個 D outputs，若用卡諾圖進行簡化有點複雜，所以在程式裡我用類似 decoder 的形式，利用 case()為每種輸入指定特定輸出。另外 ssd 的控制，我預設為 0，所以四個燈都會亮。

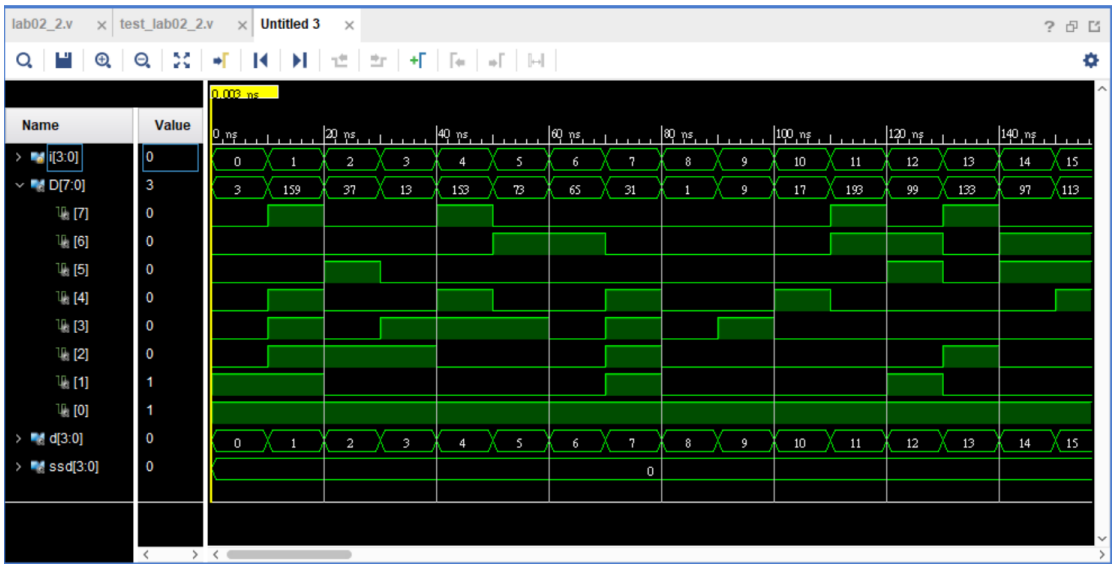
要注意的是，由於 7-seg 是 low actived，所以當 D[i]為 0 時，其對應的燈才會亮



Pin assignment

I/O	i3	i2	i1	i0	d3	d2	d1	d0
pin	W17	W16	V16	V17	V19	U19	E19	U16
I/O	D7	D6	D5	D4	D3	D2	D1	D0
pin	W7	W6	U8	V8	U5	V5	U7	V7
I/O	ssd3	ssd2	ssd1	ssd0				
pin	W4	V4	U4	U2				

Discussion



由於 ssd = 0, d = i 所以就沒有展開了，D 的結果和真值表上的設計都一樣。這裡要注意的是，雖然我們只用到 7 個燈號，但陣列設到 8 位，所以在七段顯示器中的點(腳位: DP)還是要設定。其餘就是用 case()處理，default 我設為 11111111，所以當出現未知錯誤時就不會顯示。但基本上應該不會有這樣的狀況，因為我把 4 bits 0000~1111 都設定好了。

另外可以改善的地方就是，我看到老師和其他同學都是先定義 `define SS_number = 8'bXXXXXXXX，這樣的優點我認為可以增強可讀性，下次可以試試看這樣的方法。

Conclusion

在這次實驗裡，我學到了

- 七段顯示器的使用方式
- Define 的使用方式

這次的實驗一開始看以為很複雜、很難。因為每個數字的顯示方式不一樣，以為要考慮很多東西。結果在實際開始設計的時候才發現跟我們剛學邏設時的題目一樣，簡單的組合邏輯，只要真值表沒寫錯就可以了。但在簡化的時候遇到了困難，因為有 8 個輸出，若全部都用 assign 做有點麻煩。後來想到了老師在上學期末提到的用 decoder 模擬真值表的方式寫，直接為每種輸入定義輸出。且剛好 Verilog 支援這種 case 或 if else 控制，使得這個方法快速了許多。

3 (Bonus) In the Bulls and Cows game, each of the two players writes a two-digit secret number in BCD. The digits must be all different. Then, in turns, the players try to guess their opponent's number and give the number of matches. If the matching digits are in their right positions, they are bulls, and if in different positions, they are cows. In this problem, we want to build the matching process to show the number of bulls and cows.

Design Specification

IO 輸出入設定

輸入: [7:0]s, 8bits(s[7:4], s[3:0], represent 2 BCD bits secret number),
[7:0]g, 8bits(g[7:4],g[3:0], represent 2 BCD bits guess number)

輸出: [2:0] led_b, 3bits (if 0 bulls: 001, 1 bulls: 010, 2 bulls: 100, else 000)
[2:0] led_c, 3bits (if 0 cows: 001, 1 cows: 010, 2 cows 100, else 000)

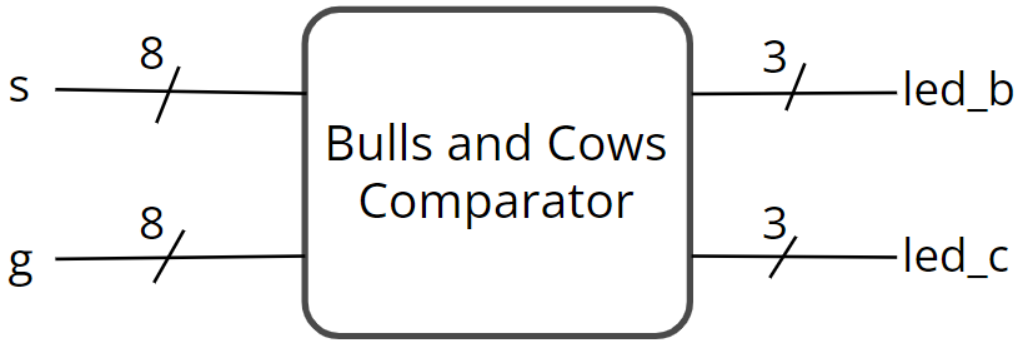
所以我整理了以下狀況:

O: correct num and place, W: wrong number, A: right number but wrong place

g[7:4]	g[3:0]	led_b[2:0]	led_c[2:0]
O	O	100	001
O	W	010	001
O	A	010	010
A	A	001	100
W	W	001	001
W	A	001	010

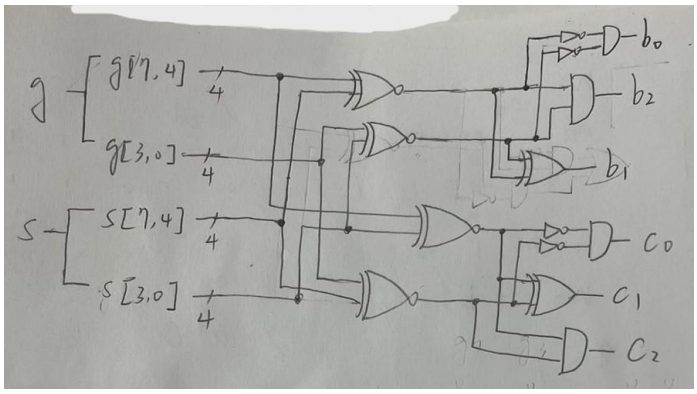
若 g 的兩位 BCD 一樣，例如: (9, 9) →無效輸入，led_b, led_c 輸出 000，s 亦然。

Logic Block



Design Implementation

a. Logic func and logic diagram



b, c 為我另外宣告的變數，用於統計 bulls 和 cows。

$b_0 = (g_{[7,4]} \oplus s_{[7,4]})(g_{[3,0]} \oplus s_{[3,0]})$

$b_1 = (g_{[7,4]} \oplus s_{[7,4]}) \oplus (g_{[3,0]} \oplus s_{[3,0]})$

$b_2 = (g_{[7,4]} \oplus s_{[7,4]})'(g_{[3,0]} \oplus s_{[3,0]})'$

$c_0 = (g_{[7,4]} \oplus s_{[3,0]})(g_{[3,0]} \oplus s_{[7,4]})$

$c_1 = (g_{[7,4]} \oplus s_{[3,0]}) \oplus (g_{[3,0]} \oplus s_{[7,4]})$

$c_2 = (g_{[7,4]} \oplus s_{[3,0]})'(g_{[3,0]} \oplus s_{[7,4]})'$

註: 在這裡若出現 2 個 bulls 時，b 會顯示 100

將 g[7:0]中的兩個 BCD g[7:4], g[3:0]分別和 s 的兩個 BCD 做比較，然後 led_b, led_c 再依據 b, c 決定輸出(case)。由於題目有規定 s 的 BCD 兩位必須為不同數字，因此我在程式裡最後有設一個 enable 判斷: 若 s 的兩位 BCD 相同或 g 的兩位 BCD 相同則 led_b, led_c 輸出為 000。也就是說只要輸入正常，led_b, led_c 至少各亮一燈，反之則是無效輸入。

pin assignment

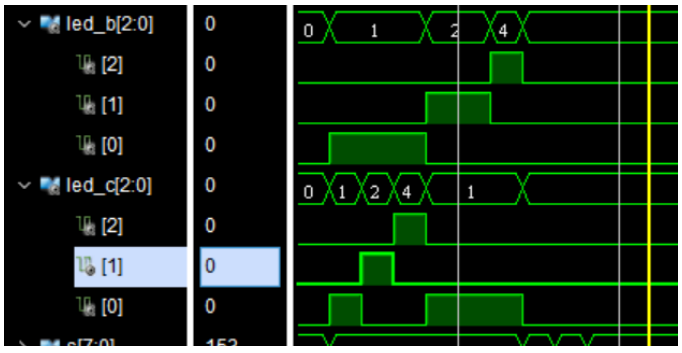
I/O	g7	g6	g5	g4	g3	g2	g1	g0
pin	R2	T1	U1	W2	R3	T2	T3	V2
I/O	s7	s6	s5	s4	s3	s2	s1	s0
pin	W13	W14	V15	W15	W17	W16	V16	V17
I/O	led_b2	led_b1	led_b0	led_c2	led_1	led_c0		
pin	U15	W18	V19	U19	E19	U16		

Discussion

```
lab02_3 U0(.s(s), .g(g), .led_b(led_b), .led_c(led_c));
initial
begin
s = 8'd0;
g = 8'd0;

#10 s[7:4] = 9;s[3:0] = 8; g[7:4] = 7; g[3:0] = 6;
#10 s[7:4] = 9;s[3:0] = 8; g[7:4] = 8; g[3:0] = 6;
#10 s[7:4] = 9;s[3:0] = 8; g[7:4] = 8; g[3:0] = 9;
#10 s[7:4] = 9;s[3:0] = 8; g[7:4] = 9; g[3:0] = 6;
#10 s[7:4] = 9;s[3:0] = 8; g[7:4] = 7; g[3:0] = 8;
#10 s[7:4] = 9;s[3:0] = 8; g[7:4] = 9; g[3:0] = 8;

#10 s[7:4] = 9;s[3:0] = 9; g[7:4] = 7; g[3:0] = 6;
#10 s[7:4] = 9;s[3:0] = 8; g[7:4] = 6; g[3:0] = 6;
#10 s[7:4] = 9;s[3:0] = 9; g[7:4] = 6; g[3:0] = 6;
end
```



▲測試測資及 led_b, led_ c 輸出

測試結果皆為正常，在遇到 invalid input 時不會顯示，符合我的預期。在實作時我是使用 if, else if 來處理判斷問題。並在最後另外寫一個 if else 來作為 enable。在程式執行時，我先宣告了兩個變數 b, c 用來統計 bulls 有幾個，cows 有幾個。最後才從 case(b), case(c)來決定 led_b, led_c 的輸出。後來的設計跟 Logic Diagram 有些不一樣，其實就是把 b 用 adder("+"運算子)來統計。我覺得這樣寫的優點就是彈性大，若之後改為 3 個 BCD 做判斷時，只要多加 b 對應到 led_b 的 case 就好，統計 bulls,cows 的方式就不用再改了。
另外 testbench 的部分，由於是用 for loop 進行迴圈，中途會出現某一位 4'd10 的現象，屬於本程式的小瑕疵。但整體運行上輸出出來的結果皆為正確。

Conclusion

這次實驗裡，我學到了

- FPGA 腳位的判斷

FPGA 腳位的判斷基本上可以從電路板上看到，但有些腳位寫得很近，常常會誤認，像 led 燈的腳位我就看成 switches 的腳位。
這次的題目有點開放，導致我剛看到題目時我點不知所措。尤其是各 3 個 bits 的 bulls, cows 輸出，讓我花了一些時間想如何顯示結果。還有在做 2 位 BCD 判斷，一開始不知道從何下手，一開始寫還會不小心重複判斷。最後從 guess number 一一做判斷。同時判斷順序也很重要，否則出來的結果會相差很多。

Reference

https://www.chipverify.com/verilog/verilog-if-else-if
https://www.kevnugent.com/2020/10/22/verilog-blogpost_002/
這兩篇提到了 if else 的實際電路圖如何操作，並和 case 做比較。可以看到 if else if 的電路圖實際上複雜很多。