**Lab 2: FPGA Emulation**

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

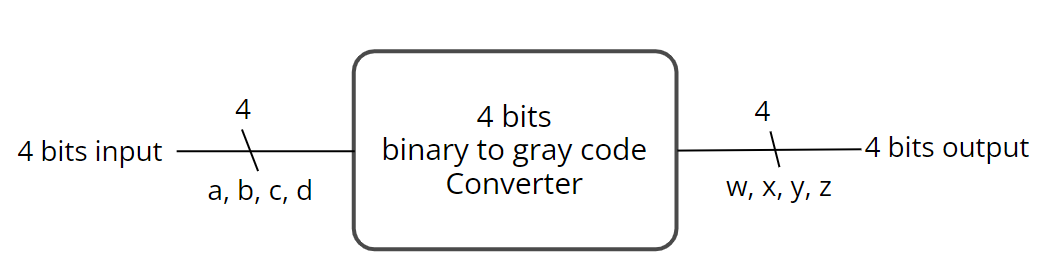
Design Specification

IO輸出入設定:

Input: a, b, c, d 大小皆為1bit (a為MSB)

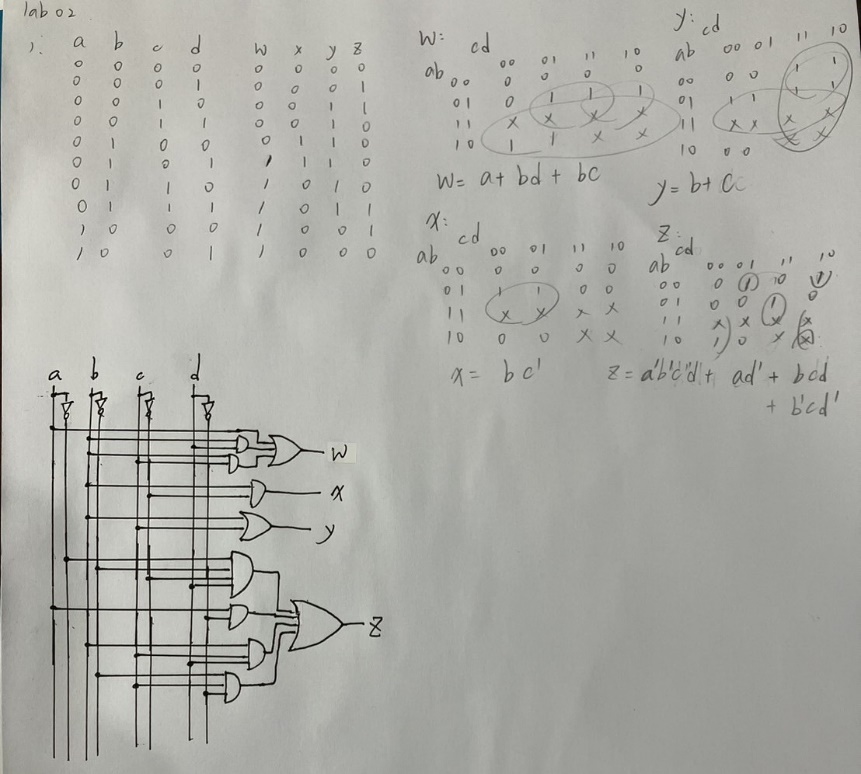
Output: w, x, y, z 大小皆為1bit (w為 MSB)

Logic block



Design Implementation

1. 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’

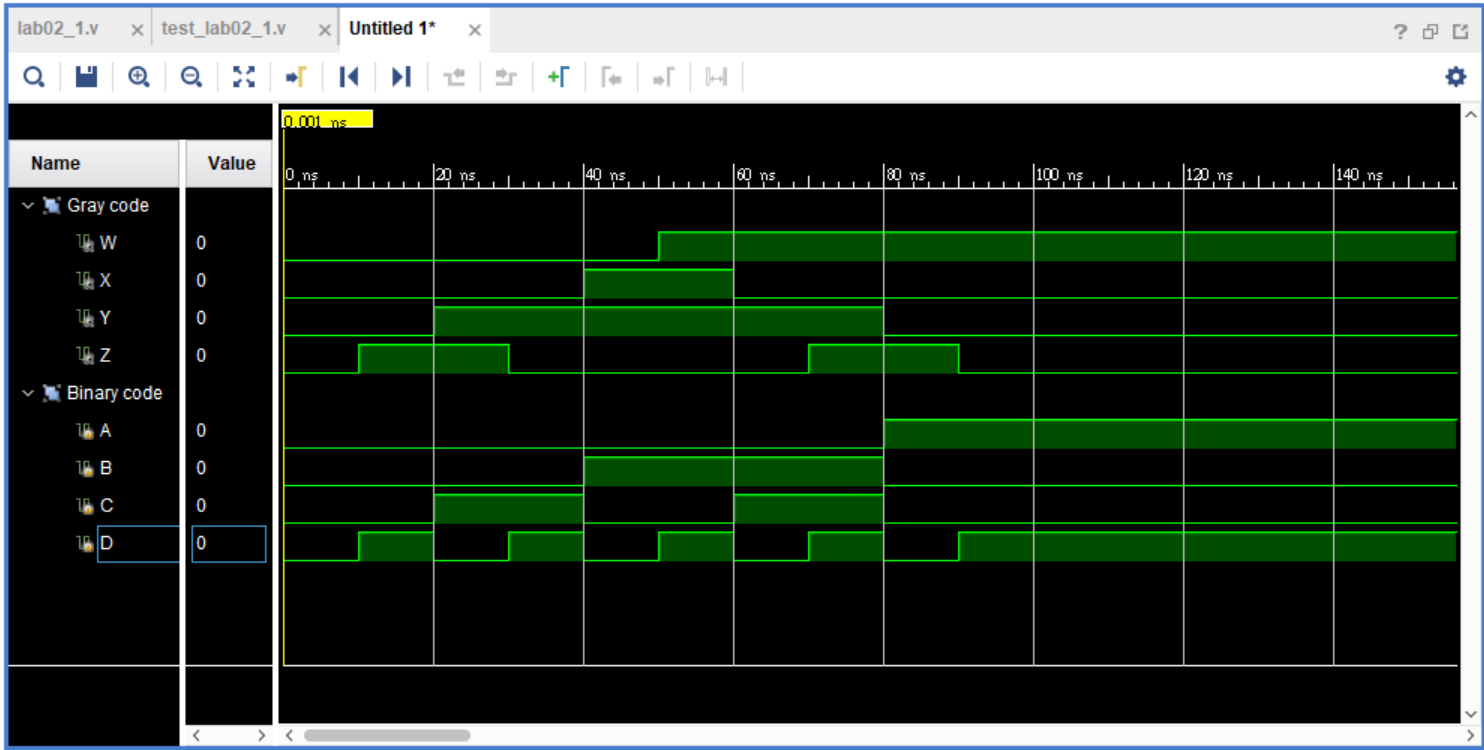
1. Logic diagram

見右圖🡪

1. 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.

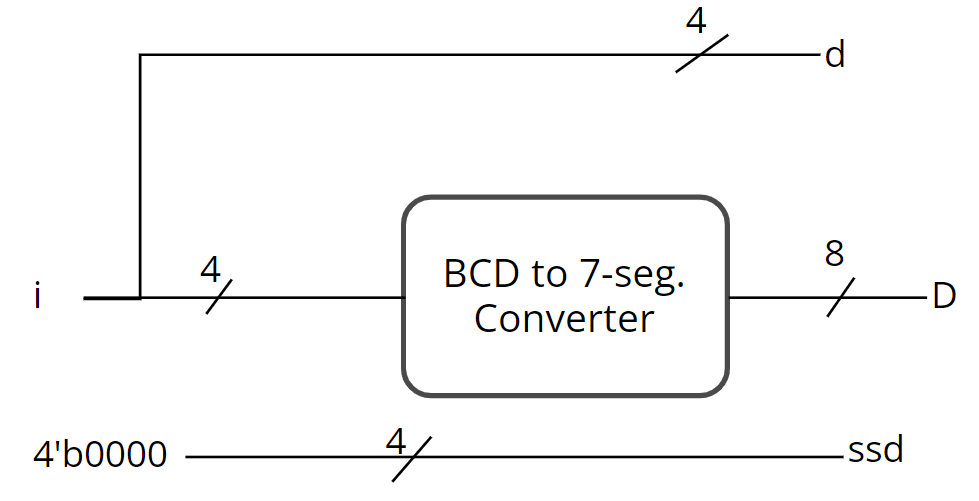
Design Specification

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

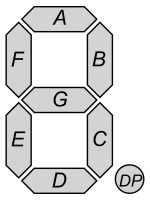


Design Implementation

1. truth table and logic func

見下面兩張圖

|  |
| --- |
|  |
|  |



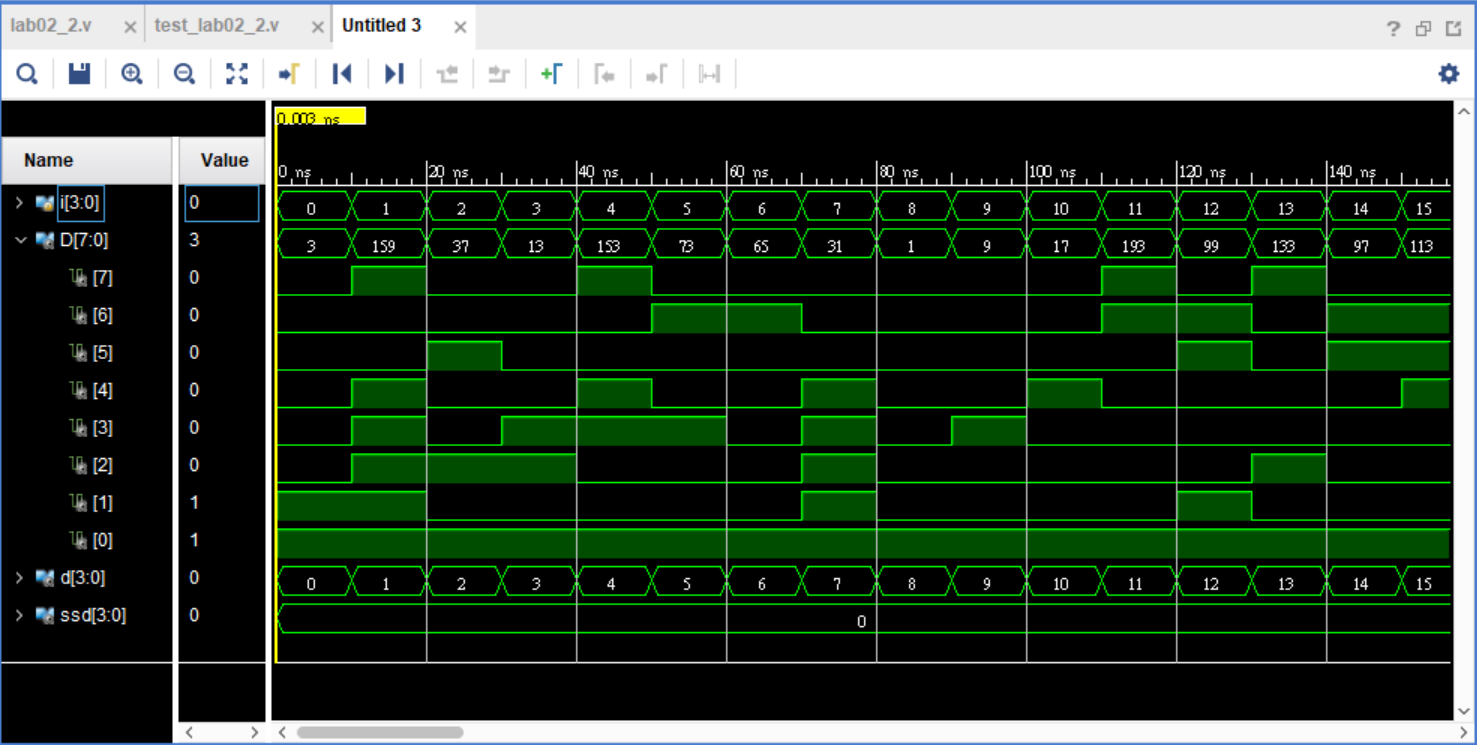
D[8:0] 分別代表7段顯示器的A~G和DP，由於8個D outputs，若用卡諾圖進行簡化有點複雜，所以在程式裡我用類似decoder的形式，利用case()為每種輸入指定特定輸出。另外ssd的控制，我預設為0，所以四個燈都會亮。

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)

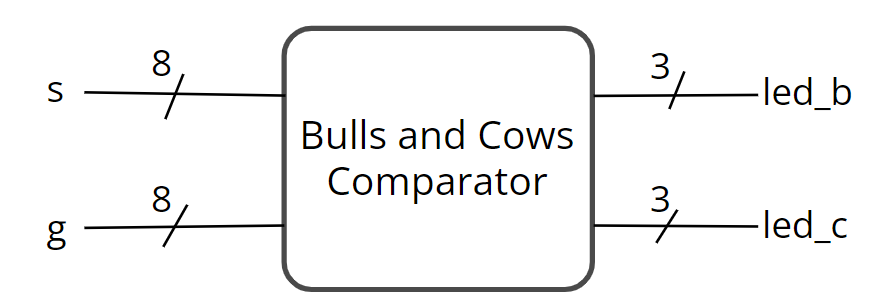
所以我整理了以下狀況:

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

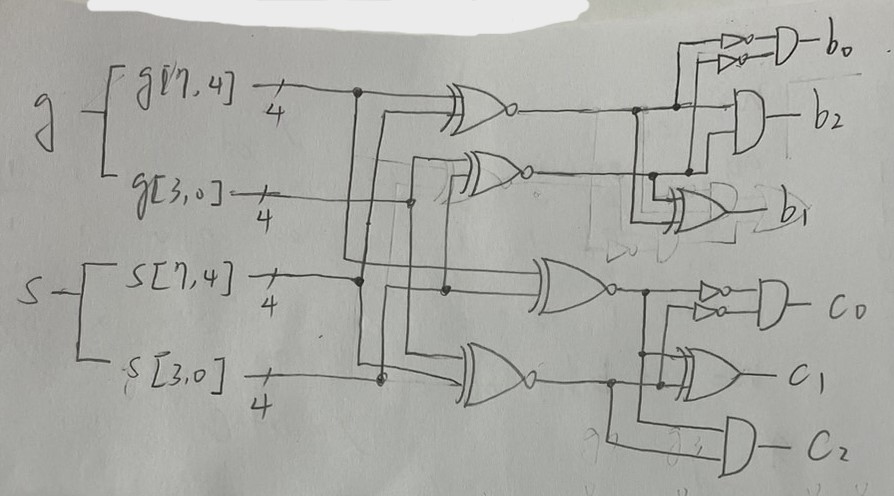
|  |  |  |  |
| --- | --- | --- | --- |
| g[7:4] | g[3:0] | led\_b[2:0] | led\_c[2:0] |
| O | O | 100 | 001 |
| O | W | 010 | 001 |
| O | A | 010 | 010 |
| A | A | 001 | 100 |
| W | W | 001 | 001 |
| W | A | 001 | 010 |

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

1. Logic func and logic diagram

b, c為我另外宣告的變數，用於統計bulls 和 cows。

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

|  |  |
| --- | --- |
|  |  |

▲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。