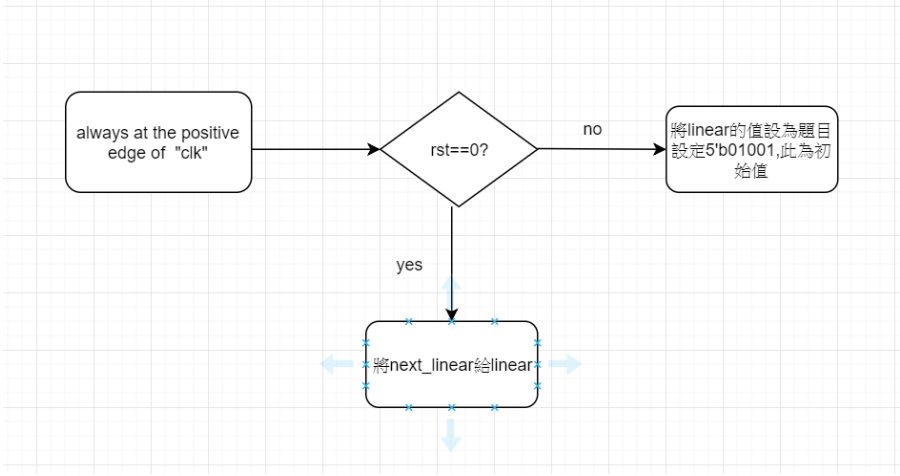
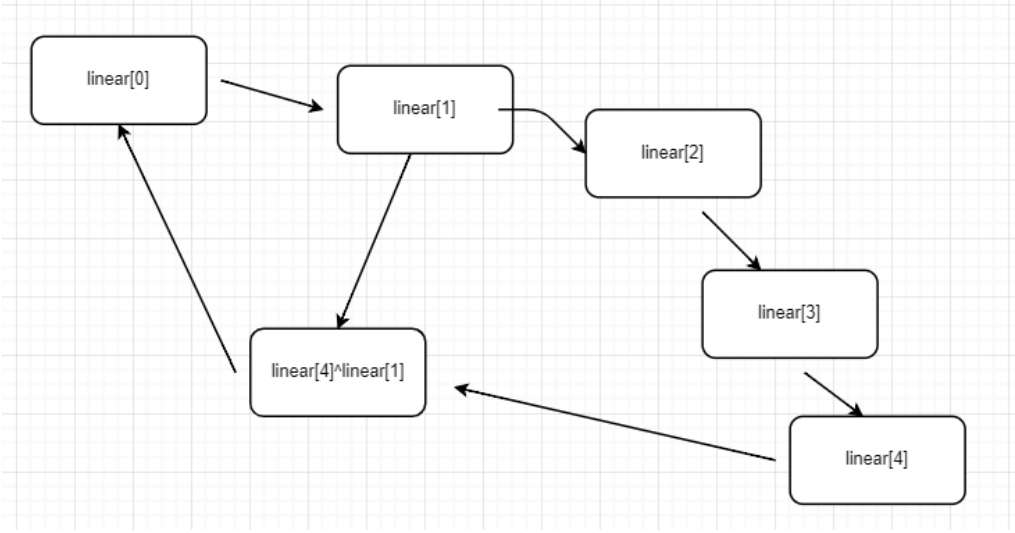
Lab3_Team6_Report

1. Linear-Feedback Shift Register (LFSR)
    a. Diagram+ Explanation
        i. Sequential Circuit:



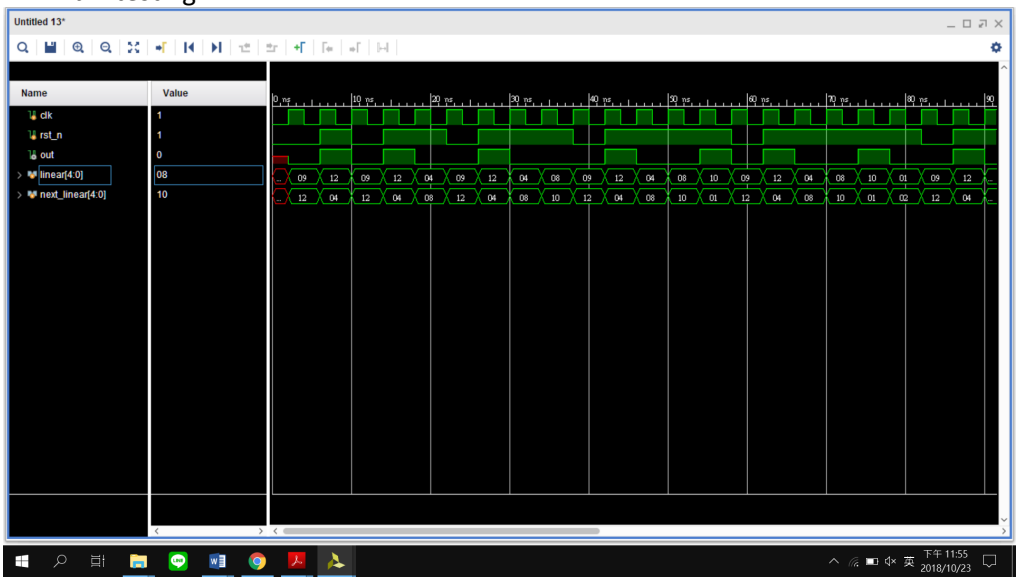
在 positive edge 的時候判斷 reset 是否為 0,若為 0 則設為初始值 01001,若為 1 則將 next_linear 給 linear(DFF).要注意的是這邊是用 non-blocking.

        ii. State transition diagram:



此處要注意的是這邊是 combinational circuit,因此是用 blocking 而不是 nonblocking.

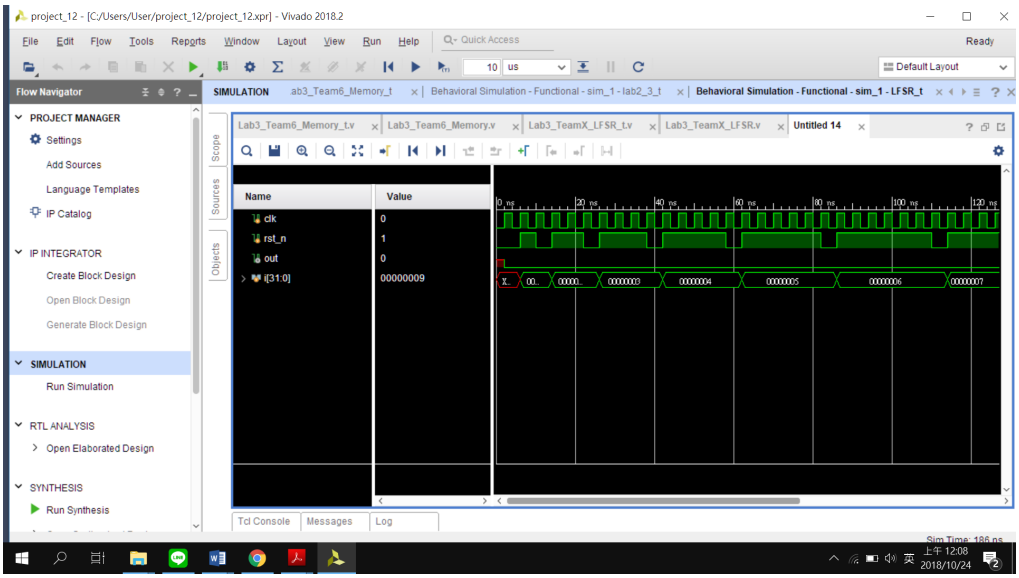
b. testing:



在我們設計的 testbench 中我們時常將 reset 置為零,隨時檢查及中斷整份程式.其中在後面我們有讓它過比較久後再中斷,確定它的連續性沒問題.而 reset 也都是在 positive edge 時立起來的.

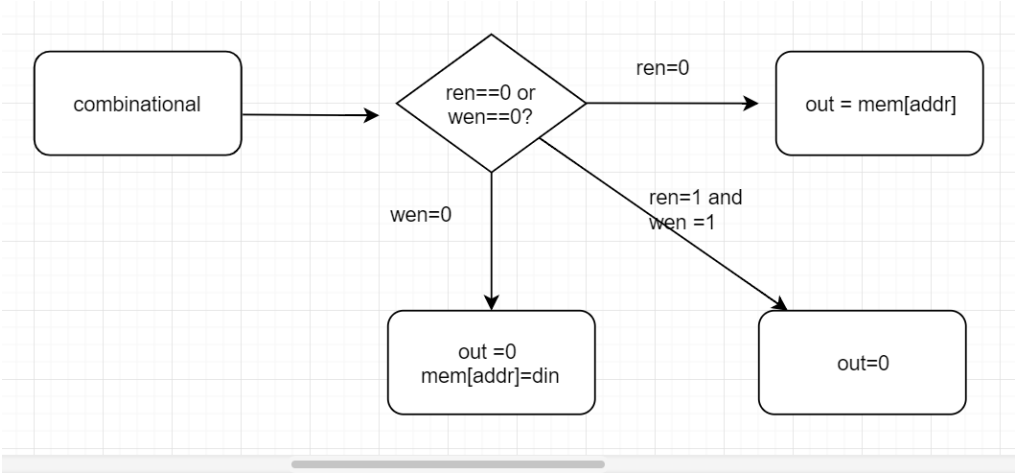
c. special situation:

如果我們在 reset=0 的情況下將 linear 設為 0,會 output 出什麼呢?先讓我們來看看 waveform:



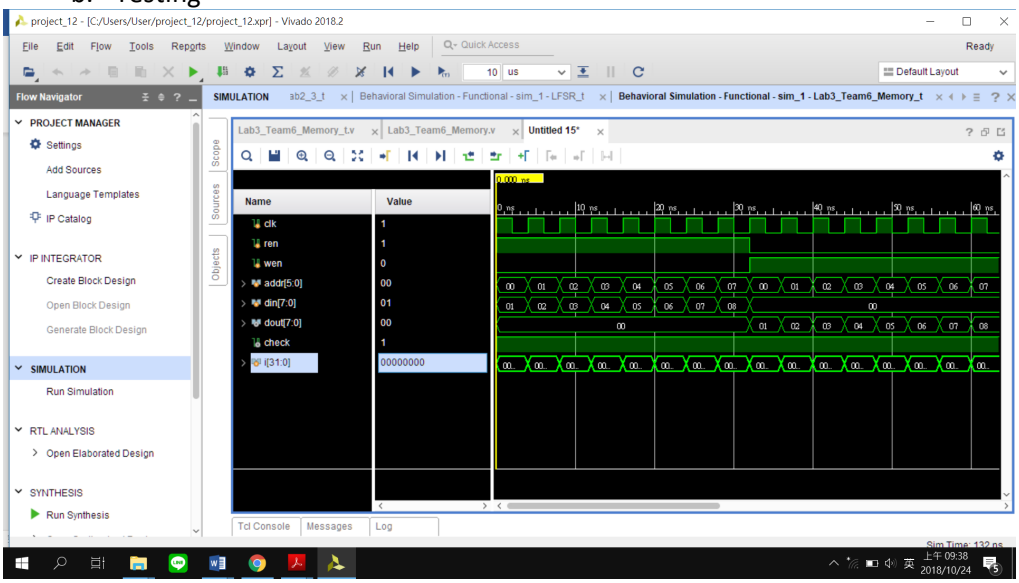
就結果來說它的 out 將永遠是 0,而經過推斷我們可以得知因為 0 和 0 的 xor 為 0,因此並沒有任何情況能產生出 1,因此所有的 output 將都為 0.

2. 64 x 8 memory array MEM
    a. Diagram+ Explanation
        i. combinational Circuit:



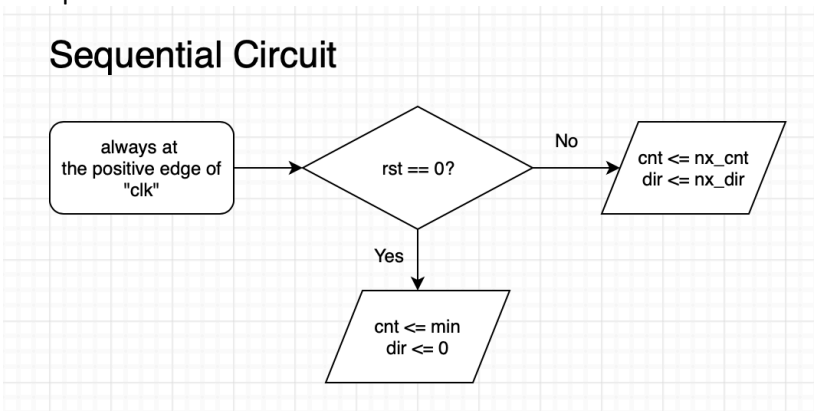
此題的 Wen 和 ren 不會同時為零(此為題目敘述),不過可以同時為 1.在 Ren 為零但 wen 為 1 時,我們將 Out 給到記憶體 mem,在 ren 為 1 但 wen 為 0 時將 out 設為零,並將已經存好的 mem 丟進 din 裡面.當兩者都為 1 的情況下將 out 設為 0.

    b. Testing



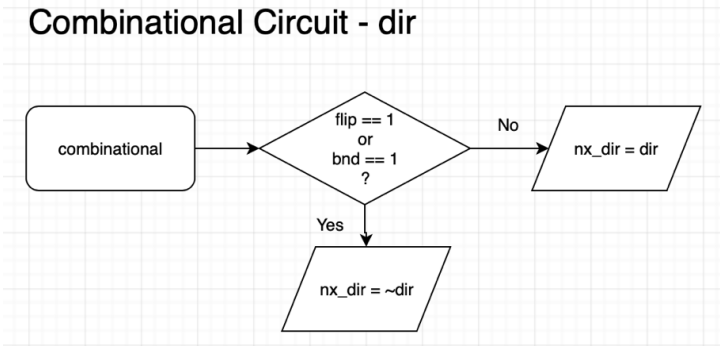
在 waveform 中我們可以很清楚的看出來 din 是在 ren=0 且 wen=1 的情況下才將值給 dout,而在 ren=1 且 wen=0 的情況下是將 dout 設為 0.而 check 則是偵測在 testbench 裡面的值和我們用.v 檔跑出來的值有沒有一樣,若一樣則將 check 設為 1.

3. Parameterized Ping-pong Counter
    a. Block Diagram + Explanation
        i. Sequential Circuit:



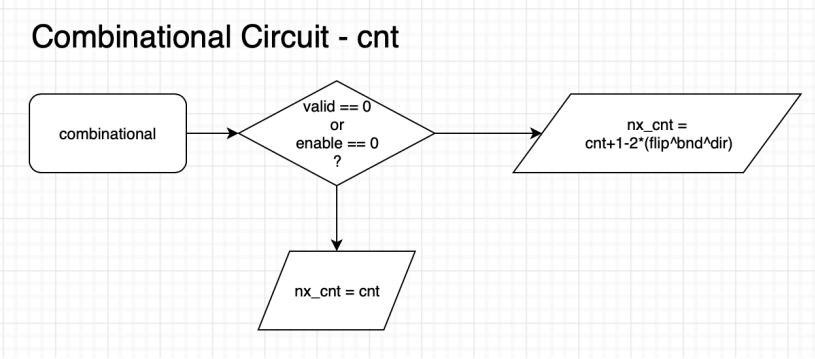
Synchronous reset，在 clk 的 positive edge 更新 cnt 和 dir，並且判斷 reset。

        ii. Combinational Circuit of dir:



需要改變方向（dir 改變）有兩種狀況：
一是 flip 被拉了起來，另一個是數到了邊緣該數回來了。值得注意的是，cnt 到了極值不代表一定是數了邊緣準備數回來，所以還需要判斷方向，例如：cnt 等於極小值而且 dir 等於 1。

iii. Combinational Circuit of cnt:

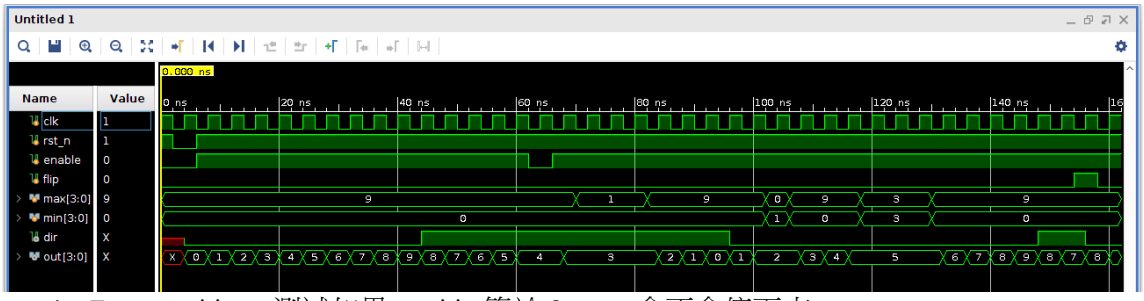


nx_cnt 可分成兩種狀況：停留在原本的值，或者增加減少。其中根據規定，當 enable 等於 0 或者 cnt 超出了極值的範圍，cnt 的值即不再變化。判定是否超出（信號 valid）的依據是 cnt 介在 max 和 min 之間，以及 min 小於 max。至於另一種情況，決定 cnt 增加還是減少的變數有 flip、bnd，以及 dir。思考的時候，我用表格列出所有狀況：

flip	bnd	dir	nx_cnt	flip^bnd^dir
0	0	0	cnt+1	0
0	0	1	cnt-1	1
0	1	0	cnt-1	1
0	1	1	cnt+1	0
1	0	0	cnt-1	1
1	0	1	cnt+1	0
1	1	0	無此可能	
1	1	1	無此可能	

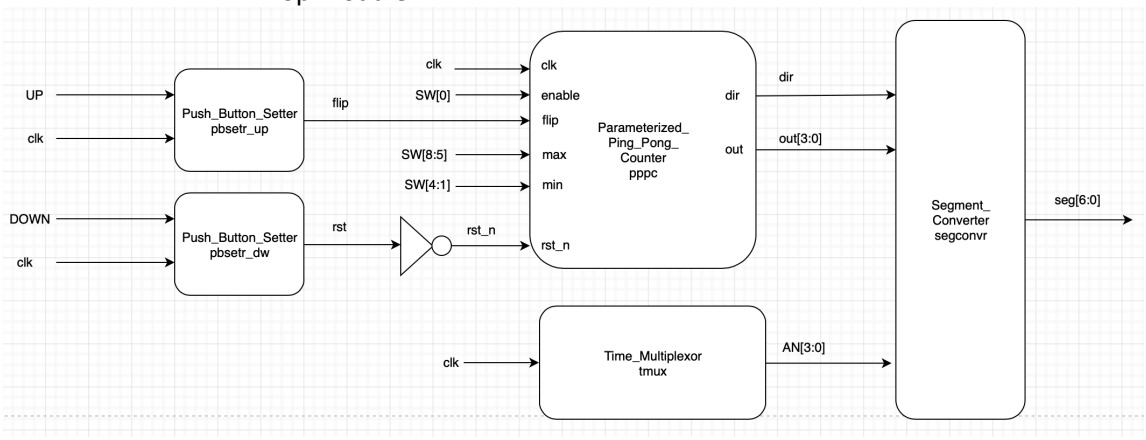
因為想要避免再用一個 if-else，所以我湊了一下信號的真值表，發現 exor 恰好可以將兩種狀況分開，再用數學加減法湊一下，便得到了那個算式。

b. Testing:

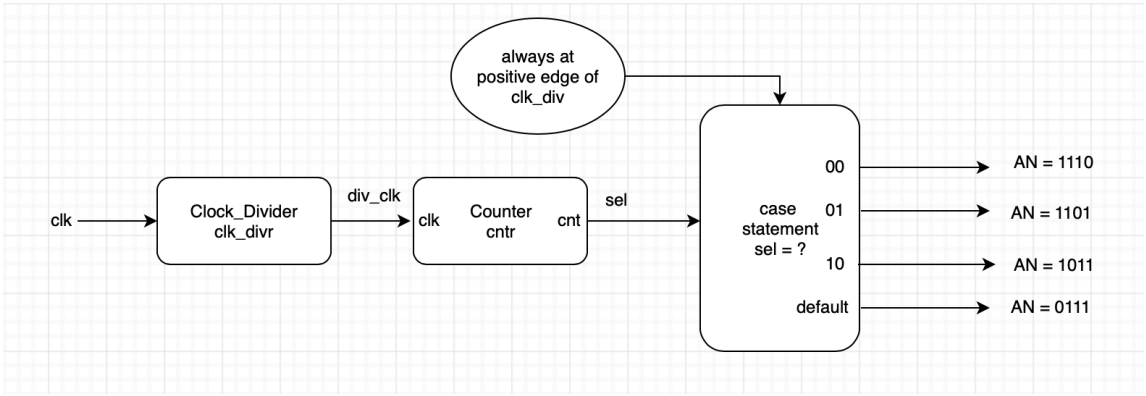


i. Test_enable： 測試如果 enable 等於 0，cnt 會不會停下來。
ii. Test_maxgtmin： 測試如果極值如果不合法，cnt 會不會停下來。
iii. Test_cntinrange： 測試如果 cnt 不在極值之間，是否會停下來。
iv. Test_flip：測試 flip 之後，是否會反過來數。

4. Parameterized Ping-pong Counter FPGA
    a. Block Diagram + Explanation
        i. Top module



        ii. Push Button Setter：
        將 push button 信號依規定轉成 debounced one pulse 信號。
        由一個 debounce module 和一個 one pulse module 接連而成。
        iii. Parameterized Ping-pong Counter：即第三題之 module。
        iv. Time Multiplexor：依規定以一定頻率切換 AN 信號，使用到了 Clock Divider 除頻以及 Counter 切換。



        v. Segment Converter：將 ping-pong counter 算出來的值轉換成 7-segment display port 的格式。我使用的是 data-level techniques，以下說明各個信號的意義。
        每個在板子上的數字是在其相對應的 AN 等於零的時候會亮起，又每個數字的每個 segment 在不同的輸入（信號 out 和 dir）時各自要亮起，因此我用 dig[3:0]記錄每四個數字，dig[i][6:0]則紀錄當 AN[i]對應的數字亮時，各個 segment 的亮暗情形。
        所以 seg[i]的值等於所有「dig[j][i]而且 AN[[j]等於 0」or 起來的結果。
        Decoder 將 out 轉換成 one-hot 信號，以單純化 dig 的 assignment。
        推導所使用的表格：

dig[0]	seg[0]	seg[1]	seg[2]	seg[3]	seg[4]	seg[5]	seg[6]
0	0	0	0	0	1	1	1
1	1	0	0	1	0	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	0	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
10	0	0	0	0	0	0	1
11	1	0	0	1	1	1	1
12	0	0	1	0	0	1	0
13	0	0	0	0	1	1	0
14	1	0	0	1	0	0	0
15	0	1	0	0	1	0	0

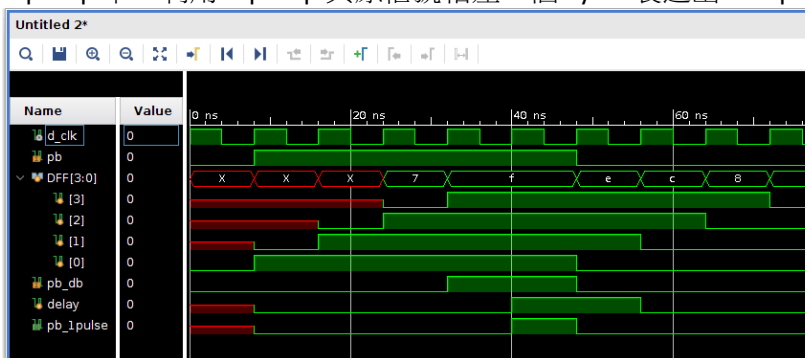
dig[1]	seg[0]	seg[1]	seg[2]	seg[3]	seg[4]	seg[5]	seg[6]
0~9	0	0	0	0	0	0	1
10~15	1	0	0	1	1	1	1

dig[2]/dig[3]	seg[0]	seg[1]	seg[2]	seg[3]	seg[4]	seg[5]	seg[6]
dir = 0;	0	0	1	1	1	0	1
Dir = 1;	1	1	0	0	0	1	1

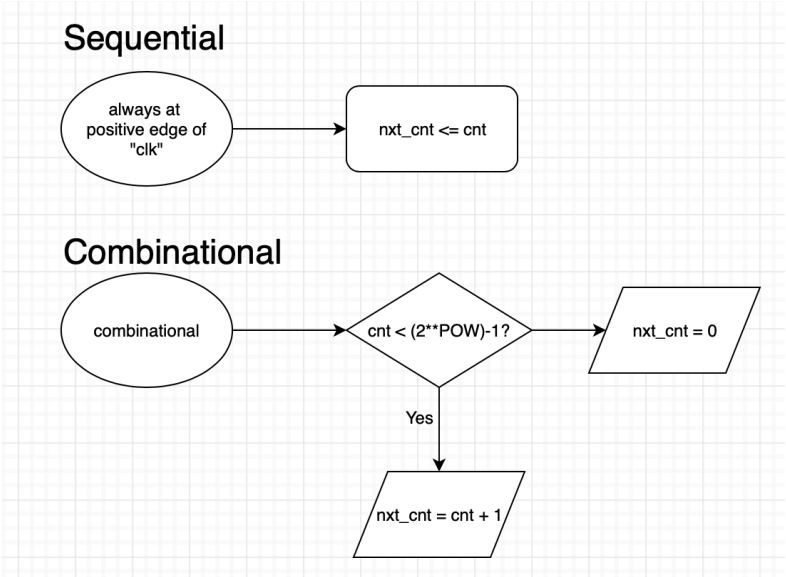
vi. Clock Divider：使用了一個 counter 計算走到第幾個 cycle。
輸出的 o_clk 是用「cnt 是否走超過一半」在做判斷的。

vii. Debounce：使用一個 clock divider 除頻，再將輸入信號放進連續四個 flip-flop 裡面，待信號夠穩定（即四個 flip-flop 都儲存 1）再輸出 1。

viii. One Pulse：使用一個 clock divider 除頻，再將輸入信號儲存在一個 flip-flop 中。利用 flip-flop 與原信號相差一個 cycle 製造出 one pulse。



ix. Counter：



x. Decoder4X16:
bin 輸入是多少，就將 16'h0001 右移幾位，即為輸出。

xi. 關於 clock cycle：雖然幾乎每個 module 用的 clock 頻率都一樣，但當初在設計時，為了讓設計更有彈性，因此各個 module 獨立產生自己的 clock。但經過幾次試驗，因為 one pulse 需維持的時間至少為 ping-pong counter 的一個 cycle，而 debounce 亦須至少維持一個 one pulse 的 cycle，所以最後才將它們的 clock 頻率調成一樣的。

b. Testing:
   i. 接上板子，依照第三題的 test bench 操作，觀察是否有預期的結果。
   ii. 決定 counting clock 頻率的過程（2 的指數）：30（太慢）->20（太快）->25（不夠快）->23（太快）->24
   iii. AN 切換的 clock 頻率是 2^17。

5. What I've learned:
   a. 越來越熟悉如何 debug，開始會猜測出現 unknown 的原因。
   b. 更熟悉 7-segment display 的運用，但希望能想到比現在更好的做法。
   c. 學會如何加 clock。
   d. 學會幾個 verilog 的 coding 技巧，如 passing parameter to a module 和 task。
   e. 成功地將各種操作拆解成一個個 module。
   f. 學會更多 vim 便利的技巧，如同時編輯兩個檔案，或對數行做同樣編輯。