**Lab 04**

105060016 謝承儒

1. **Lab04-1 : 8-bit Shifter**
2. **Design Specification :**
3. Input : clk //輸入的頻率(100MHz)

rst //當=1時，使板子暫停運作

1. Output : [7:0]q //為這次移位的值，當rst=1時，其值為01010101
2. Wire : clk\_d //除頻後的頻率(1Hz)，用來驅動Shifte

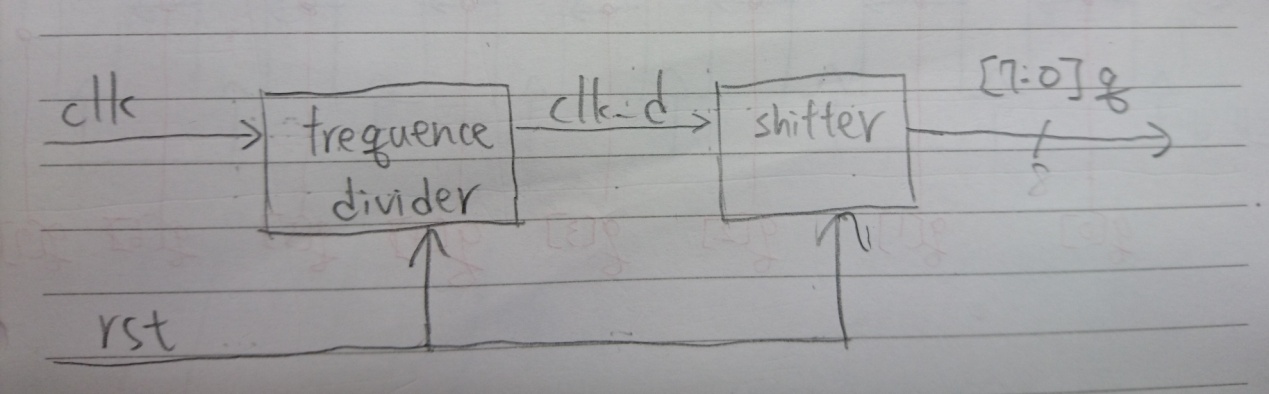


圖1 Lab04-1的區塊圖

1. **Design Implementation :**
2. Logic diagram :

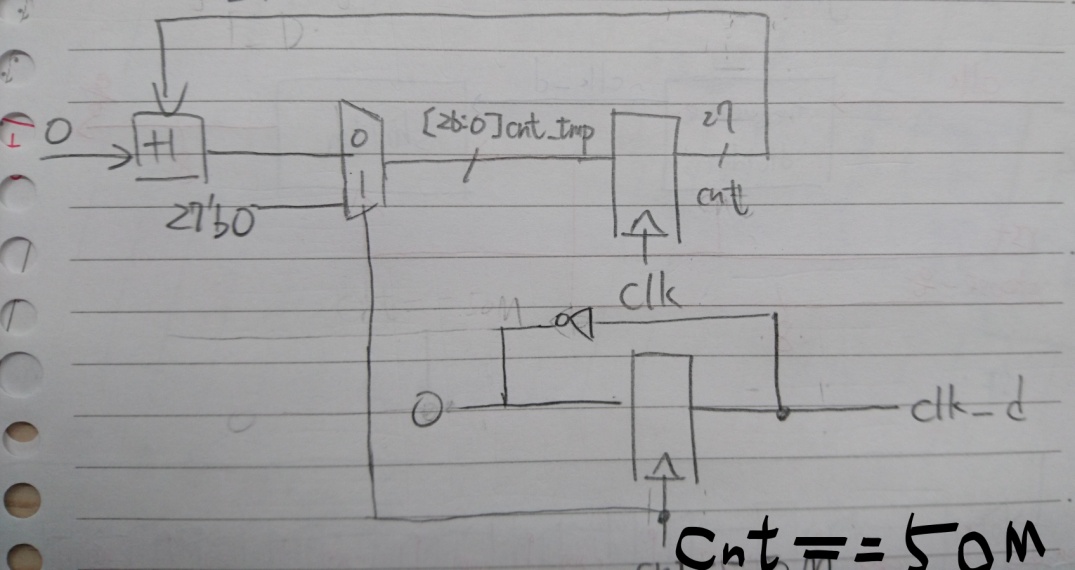


圖2 Lab04-1的除頻器(100MHz->1Hz)

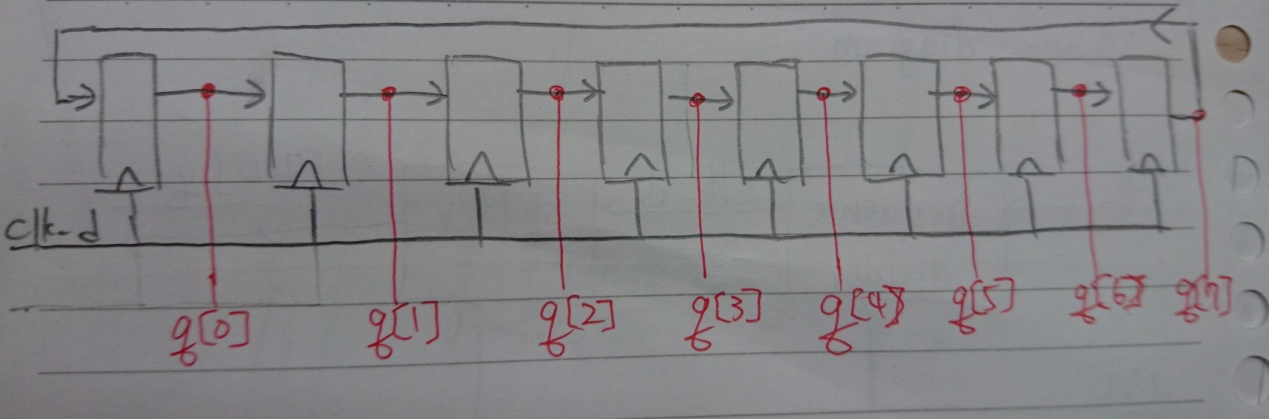


圖3 Lab04-1的8-bit移位器

1. Pin assignment :
2. Input :
3. clk = W5
4. rst = R2
5. Output :
6. q[0] = U16
7. q[1] = E19
8. q[2] = U19
9. q[3] = V19
10. q[4] = W18
11. q[5] = U15
12. q[6] = U14
13. q[7] = V14
14. **Discussion :**
15. 思考過程 :

這實驗室要做1Hz的8-bit Shifter。

首先，利用上次Lab03做出來的成果，設計一個Counter以clk驅動，每當它數到50M時，就將其歸零並把clk\_d =~clk\_d，如此clk\_d便為1Hz。

接著把clk\_d輸入到Shifter，每次clk\_d posedge時，就將q裡面的每一位往下一位送。Code裡使用”<=”便可以讓每位數Shift動作同時執行，若使用”=”便有可能把其中的值改變。

1. 過程中的Bug :

沒有遇到甚麼Bug。

1. **Lab04-2 : 8-bit Shifter(可手動輸入)**
2. **Design Specification :**
   1. Input : clk //輸入的頻率(100MHz)

rst //當=1時，使板子暫停運作

[7:0] choose\_q //當rst=1時，q是其值，可以手動輸入

* 1. Output : [7:0]q //為這次移位的值，當rst=1時可以改變
  2. Wire : clk\_d //除頻後的頻率(1Hz)，用來驅動Shifter

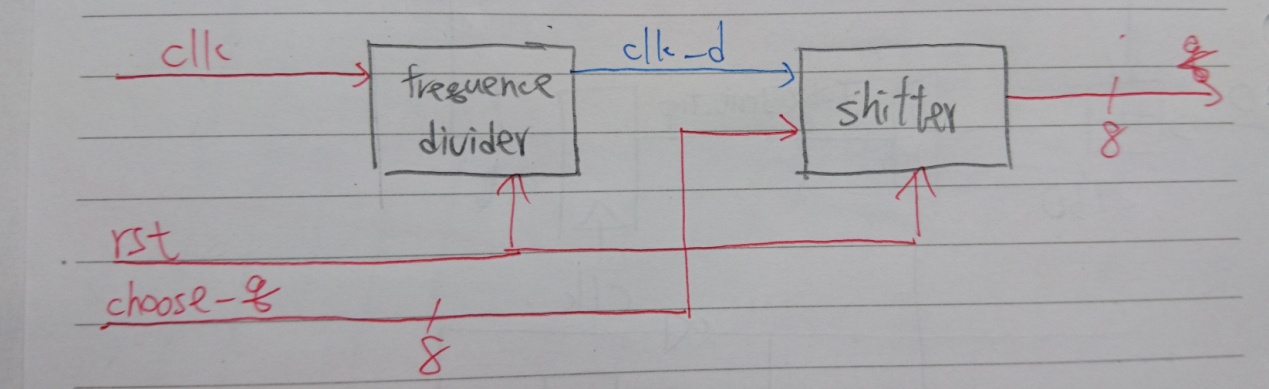


圖4 Lab04-2的區塊圖

1. **Design Implementation :**
   1. Logic diagram :

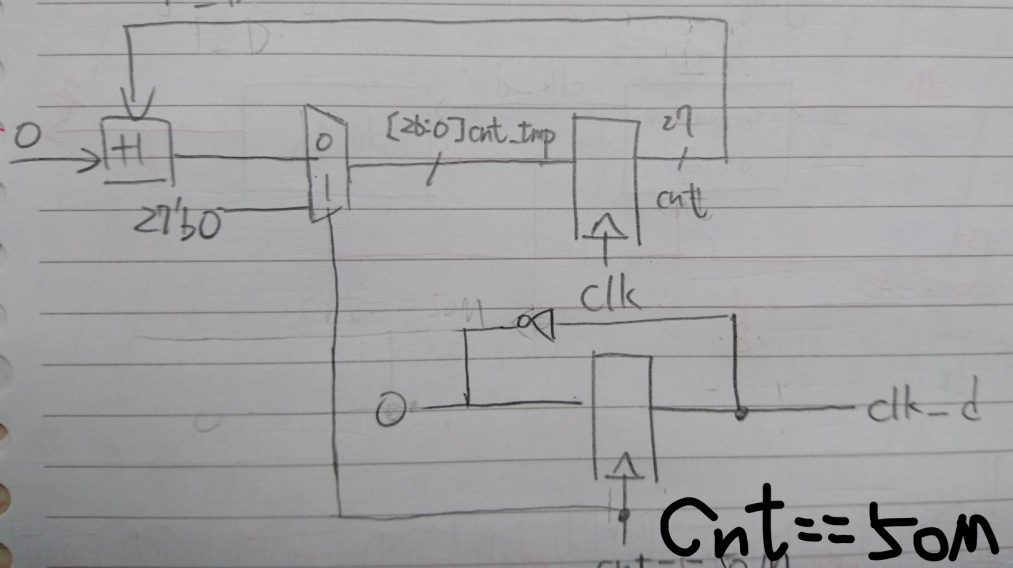


圖5 Lab04-2的除頻器(100MHz->1Hz)

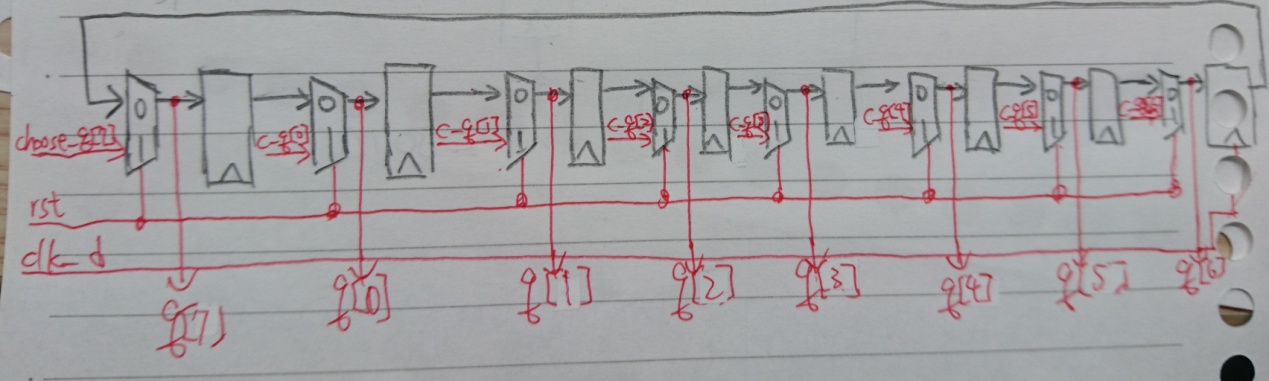


圖6 Lab04-2的8-bit 移位器

* 1. Pin assignment :
     1. Input :
        1. clk : W5
        2. rst : R2
        3. choose\_q[0] = V17
        4. choose\_q[1] = V16
        5. choose\_q[2] = W16
        6. choose\_q[3] = W17
        7. choose\_q[4] = W15
        8. choose\_q[5] = V15
        9. choose\_q[6] = W14
        10. choose\_q[7] = W13
     2. Output :
        1. q[0] = E19
        2. q[1] = U16
        3. q[2] = U19
        4. q[3] = V19
        5. q[4] = W18
        6. q[5] = U15
        7. q[6] = U14
        8. q[7] = V14

1. **Discussion :**
   1. 思考過程 :

這實驗要做的是當rst=1時可以手動輸入新的值([7:0]choose\_q)來取代本來的值([7:0]q)。

除頻器可以利用前個實驗Lab04-1的結果。而Shifter也跟Lab04-1差異不大，只需在rst=1時，將choose\_q輸入給q即可。

* 1. 過程中的Bug :

本來這題要做出真正的隨機(Random)，可以直接從除頻器裡的Counter輸出現在的值，再輸入給q。其值是由板子上的震盪器(W5)中得到的，因此變化極快，每一次rst開關後我們都很難控制得到的值，達到近似隨機的效果。

1. **Lab04-3 : ntHUEE的跑馬燈**
2. **Design Specification :**
3. Input : clk\_crystal //板子輸入的頻率(100MHz)

rst //讓板子暫停運作

1. Output : [7:0] display //七段顯示器的顯示碼

[3:0] display\_c //用來決定display放入哪一位

* 1. Wire : clk\_d //除頻後的頻率(1Hz)，用來驅動Shifter

[1:0] ssd\_ctl\_en //用來決定此次處理的是SSD的哪一位

[2:0] D0,D1,D2,D3//為Shifter輸出的4個3-bit碼

[7:0] Display0, Display1, Display2, Display3

//為由D0~3解碼出來的顯示

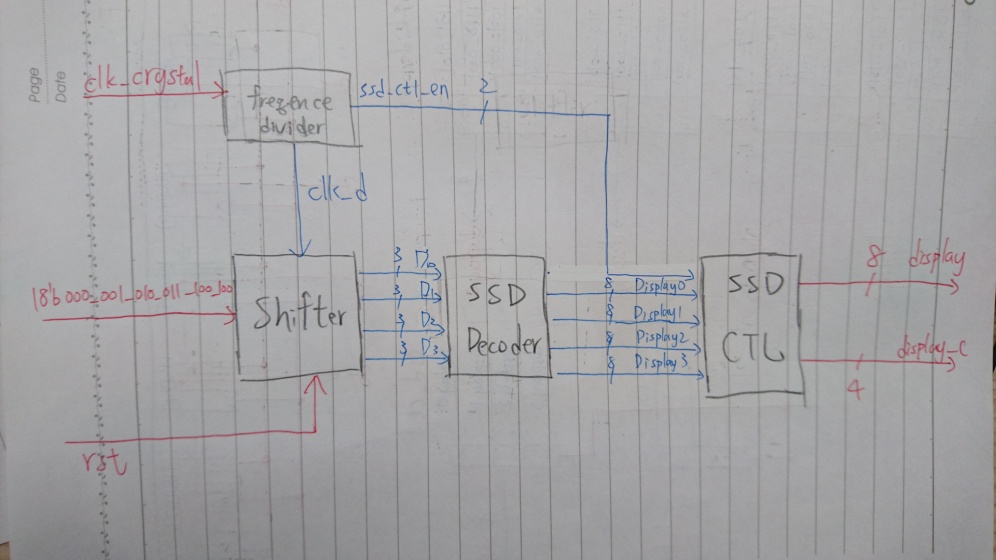


圖7 Lab04-3的區塊圖

1. **Design Implementation :**
2. Logic function :
3. ssd\_ctl\_en = cnt[20:19];
4. D0 = nthuee[8:6]
5. D1 = nthuee[11:9]
6. D2 = nthuee[14:12]
7. D3 = nthuee[17:15]
8. Logic diagram :

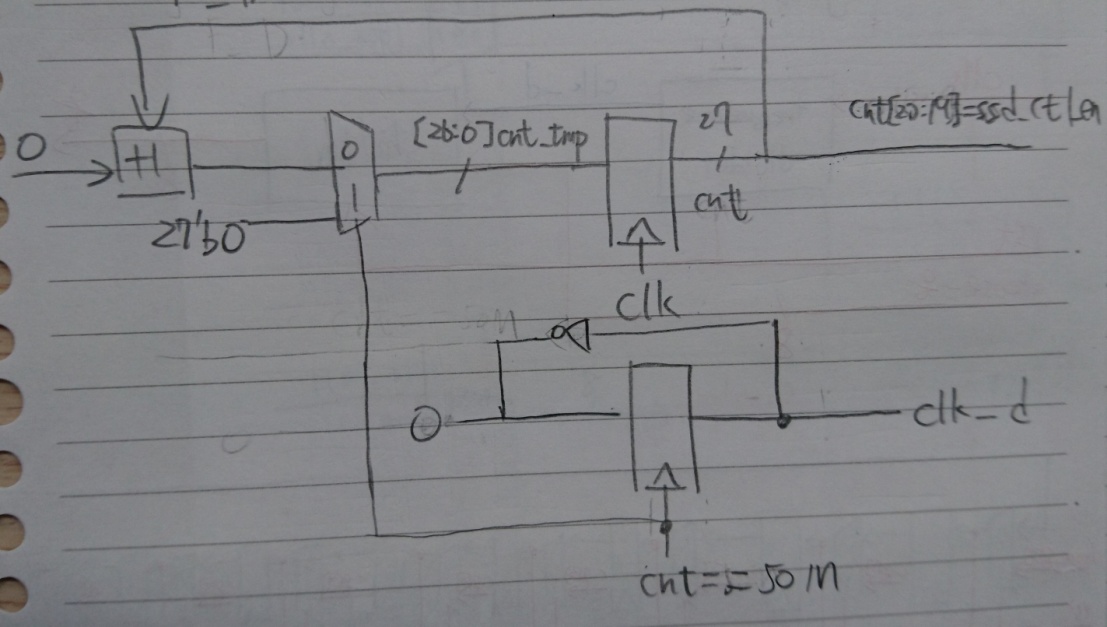


圖8 Lab04-3除頻器

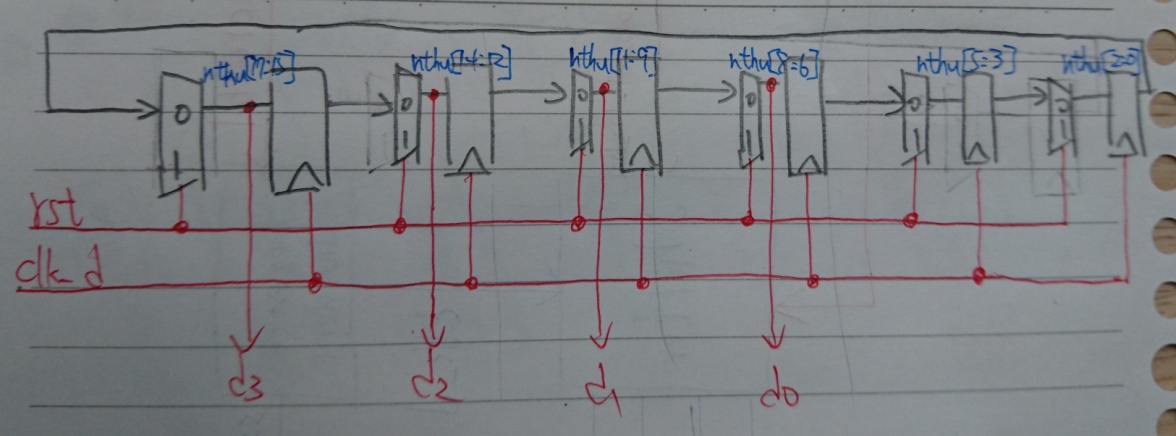


圖9 Lab04-3 Shifter

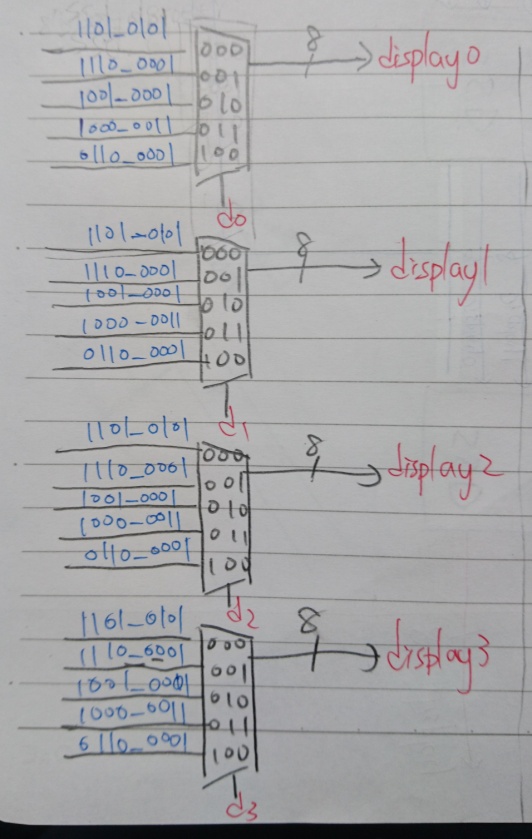


圖10 Lab04-3 3->8 SSD(七段顯示器) Decoder

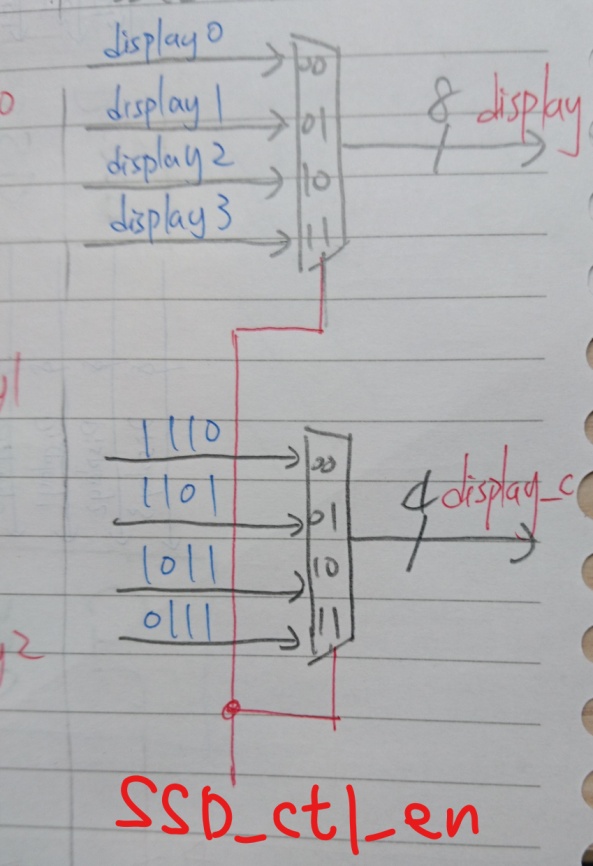


圖11 Lab04-3 SSD Control

1. Pin assignment :
   * 1. Input :
        1. clk\_crystal = W5
        2. rst = V17
     2. Output :
        1. display[0] = V7
        2. display[1] = U7
        3. display[2] = V5
        4. display[3] = U5
        5. display[4] = V8
        6. display[5] = U8
        7. display[6] = W6
        8. display[7] = W7
        9. display\_c[0] = U2
        10. display\_c[1] = U4
        11. display\_c[2] = V4
        12. display\_c[3] = W4
2. **Discussion :**
3. 思考過程 :

這次要做的是頻率為1Hz的ntHUEE的跑馬燈。按照圖7的區塊圖來解釋。

1. **100MHz->1Hz的除頻器**

用輸入的clk(clk\_crystal)來驅動27-bit的Counter(其值為[26:0]cnt)。在cnt=50M時，讓clk\_d = ~clk\_d且cnt歸零，達到1Hz的效果。除此之外，將其中的兩位(ssd\_ctl\_en)輸出，作為待會SSD\_Control的其中一個Input，此次選擇cnt[20:19]。

1. **一次移3-bit的Shifter**

將a的1Hz的clk\_d輸入來驅動Shifter，其中移動的值稱為[17:0]nthuee，初始值為000\_001\_010\_011\_100\_100，每3-bit代表1個字母(000為n、001為t、010為H、011為U、100為E)。因此，每次移3-bit就代表移1個字母，再將前4組3-bit碼輸出(D0~D3)作為下個SSD Decoder的Input。

1. **SSD Decoder**

將b處得到的4組3-bit碼輸入，作為多工器的依據(在code裡可用case得到相同效果)，得出4組8-bit的顯示碼(Display0~Display3)，將它們輸入到下個SSD Control。

1. **SSD Control**

把在a處得到的2-bit的ssd\_ctl\_en輸入，因為2-bit可以得到4種不同的結果(00、01、10、11)，可以作為選擇SSD位數的依據，而且因為其變化速度極快，讓本來是依序做的動作，看起來像是同時並行的。

當ssd\_ctl\_en=00時，便將display0輸出(display)，並將display\_c輸出為1110。如此顯示器上便只有最後一位可以改變，保留其他三個的字母。其他ssd\_ctl\_en的情況也是一樣。

1. 遇到的Bug :

中間常常把變數的寬度宣告錯誤，像是本來要8-bit的display，卻宣告成[8:0]display，導致顯示出來的結果不符合預期。除此之外，還打錯名稱，像是ssd\_ctl\_en用到時打成ssd\_ctl，結果輸出的值沒有改變，可是這類的bug不會導致無法編譯，因為系統很”好意”的幫你宣告，讓code能順利編譯。

1. **Conclusion :**

這次實驗因為一堆小bug而出錯，但這類bug又不會導致編譯錯誤或是成為critical warning，讓debug的時間大幅增加，真是讓人感到無力。

1. **Reference :**
   1. 老師給的實驗講義

讓我知道code怎麼打。

* 1. 上學期的邏輯設計講義

讓我知道Shiter的原理