Lab 04

105060016 謝承儒

一.Lab04-1 : 8-bit Shifter

1. Design Specification :

A. Input : clk　　//輸入的頻率(100MHz)
　　rst　　//當=1 時，使板子暫停運作
B. Output : [7:0]q　　//為這次移位的值，當 rst=1 時，其值為 01010101
C. Wire : clk_d　　//除頻後的頻率(1Hz)，用來驅動 Shifte

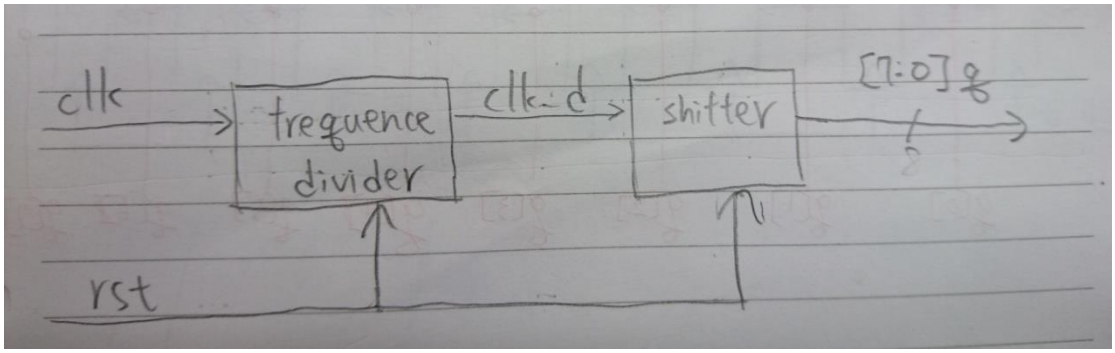

圖 1 Lab04-1 的區塊圖

2. Design Implementation :

A. Logic diagram :

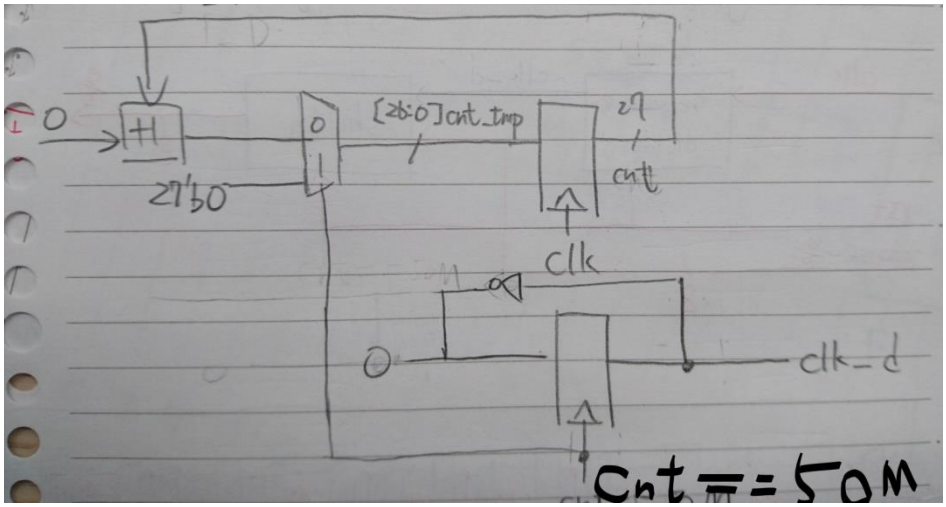

圖 2 Lab04-1 的除頻器(100MHz->1Hz)

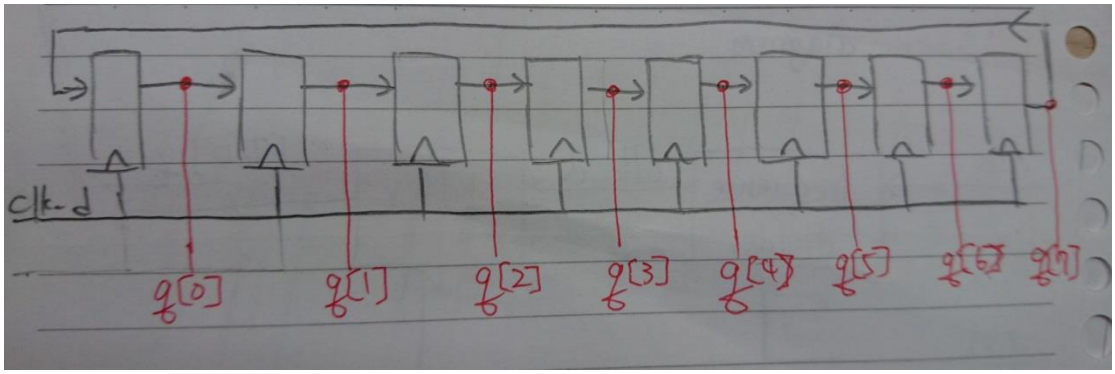

圖 3 Lab04-1 的 8-bit 移位器

B. Pin assignment :
   a. Input :
      1) clk = W5
      2) rst = R2
   b. Output :
      1) q[0] = U16
      2) q[1] = E19
      3) q[2] = U19
      4) q[3] = V19
      5) q[4] = W18
      6) q[5] = U15
      7) q[6] = U14
      8) q[7] = V14

3. Discussion :

A. 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。

B. 過程中的 Bug :

沒有遇到甚麼 Bug。

二.Lab04-2 : 8-bit Shifter(可手動輸入)

1. Design Specification :

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

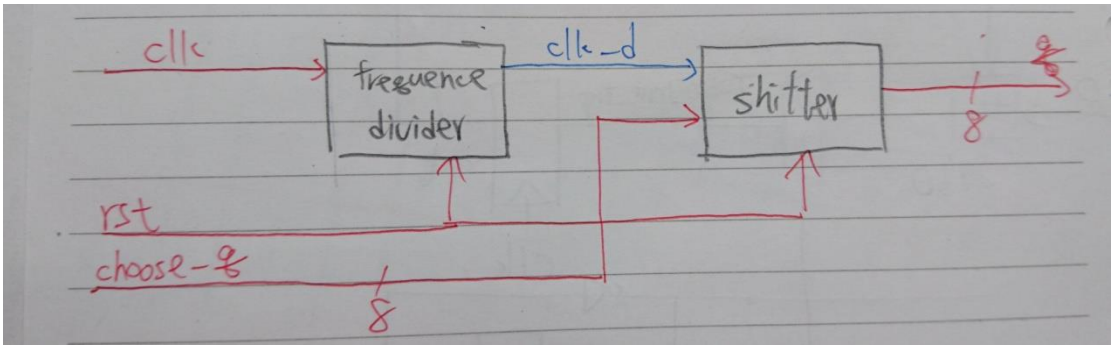

圖 4 Lab04-2 的區塊圖

2. Design Implementation :

A. Logic diagram :

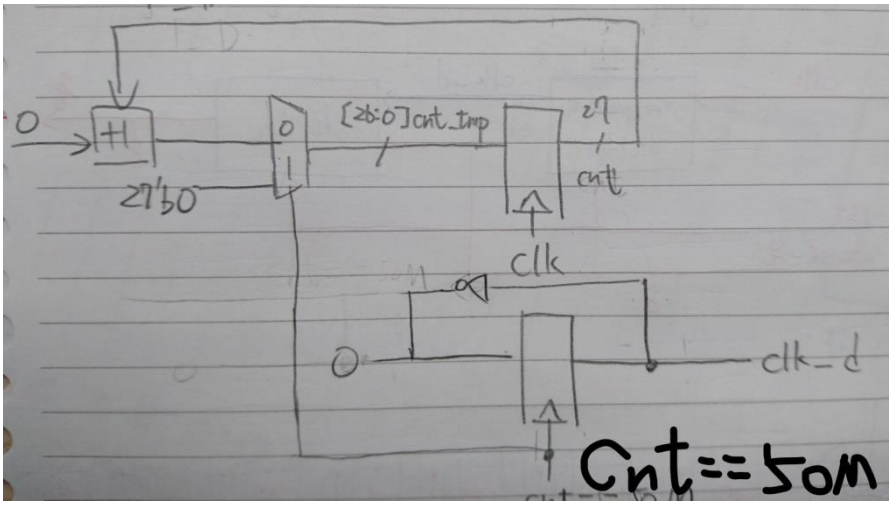

圖 5 Lab04-2 的除頻器(100MHz->1Hz)

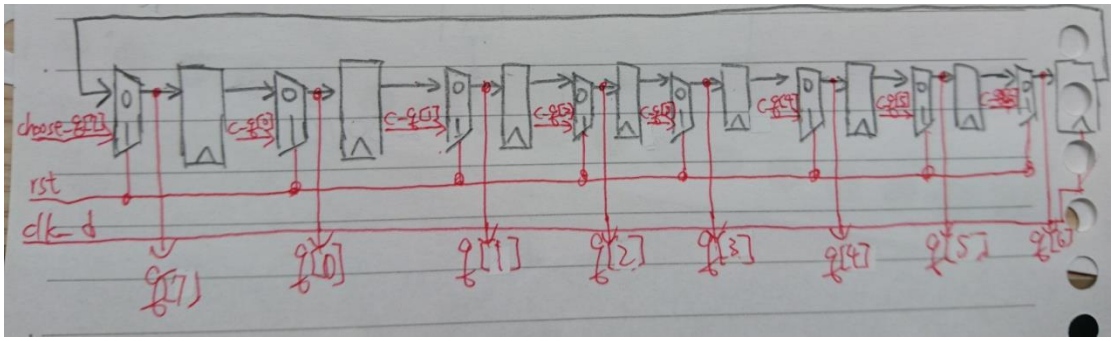

圖 6 Lab04-2 的 8-bit 移位器

B. Pin assignment :

a. 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

b. 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

3. Discussion :

A. 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。

B. 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。

三.Lab04-3 : ntHUEE 的跑馬燈

1. Design Specification :

A. Input : clk_crystal　　//板子輸入的頻率(100MHz)
　　rst　　//讓板子暫停運作
B. 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
C. 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

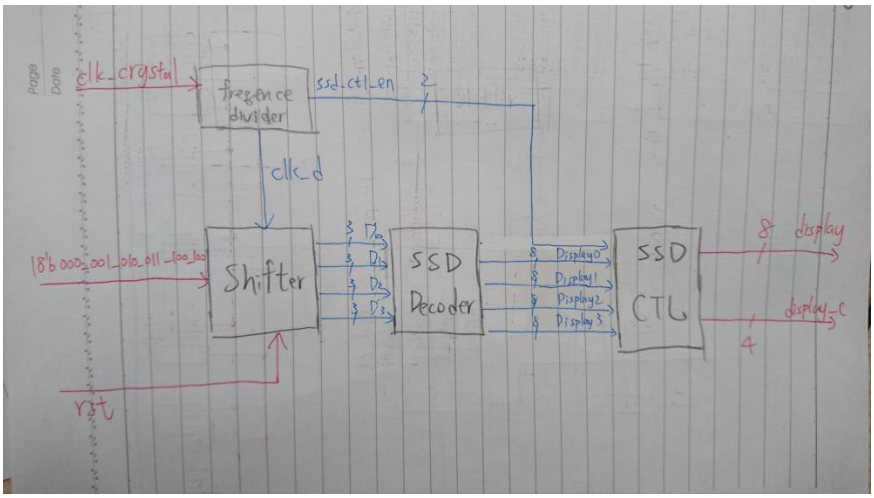

圖 7 Lab04-3 的區塊圖

2. Design Implementation :

A. Logic function :
   a. ssd_ctl_en = cnt[20:19];
   b. D0 = nthuee[8:6]
   c. D1 = nthuee[11:9]
   d. D2 = nthuee[14:12]
   e. D3 = nthuee[17:15]

B. Logic diagram :

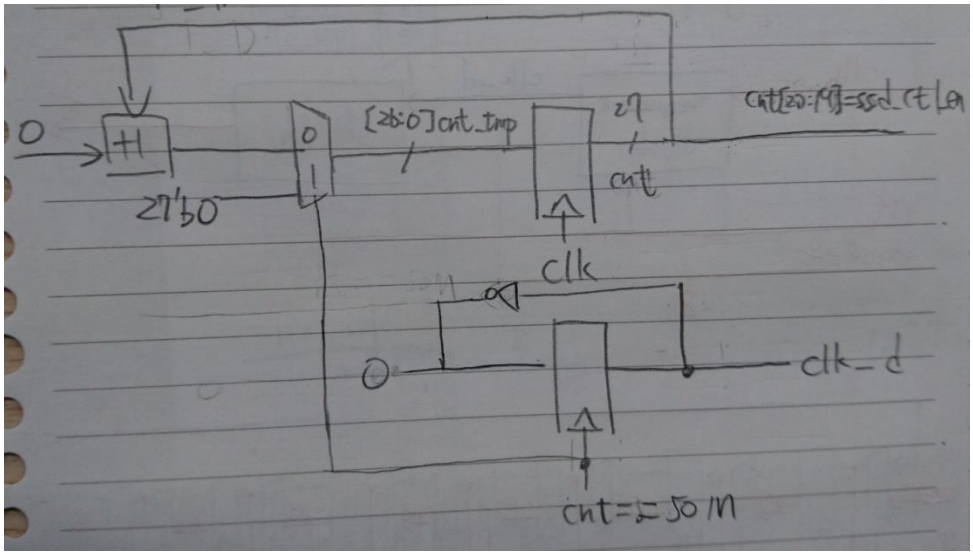

圖 8 Lab04-3 除頻器

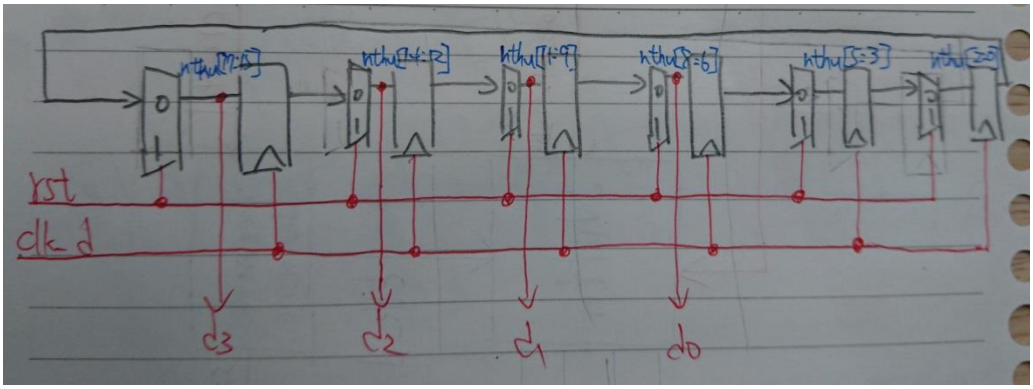

圖 9 Lab04-3 Shifter

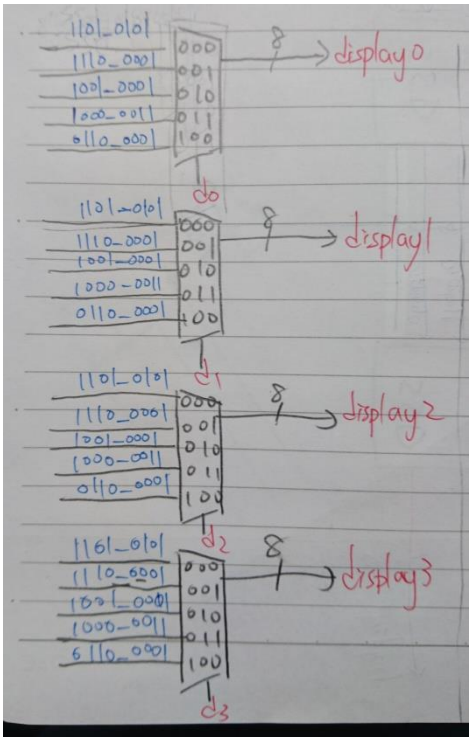

圖 10 Lab04-3 3->8 SSD(七段顯示器) Decoder

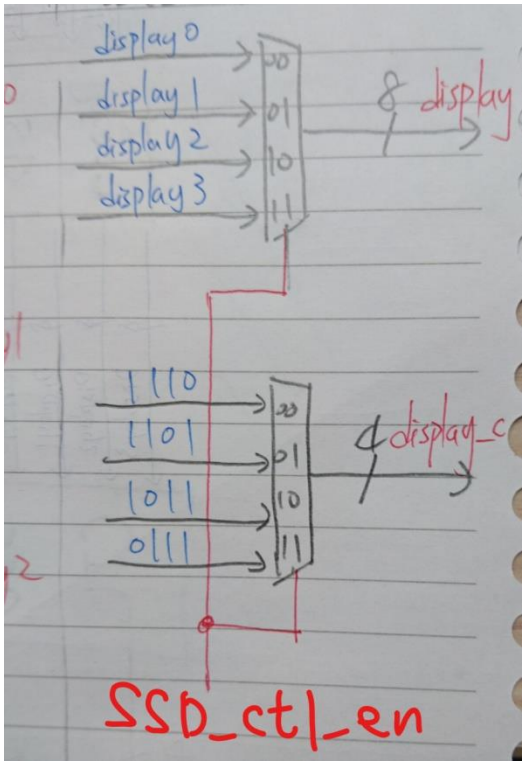

圖 11 Lab04-3 SSD Control

C. Pin assignment :

a. Input :

1) clk_crystal = W5
2) rst = V17

b. 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

3. Discussion :

A. 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。

a. 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]。

b. 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。

c. 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。

d. 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。

B. 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。

四.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。

五.Reference :

1. 老師給的實驗講義
   讓我知道 code 怎麼打。
2. 上學期的邏輯設計講義
   讓我知道 Shiter 的原理