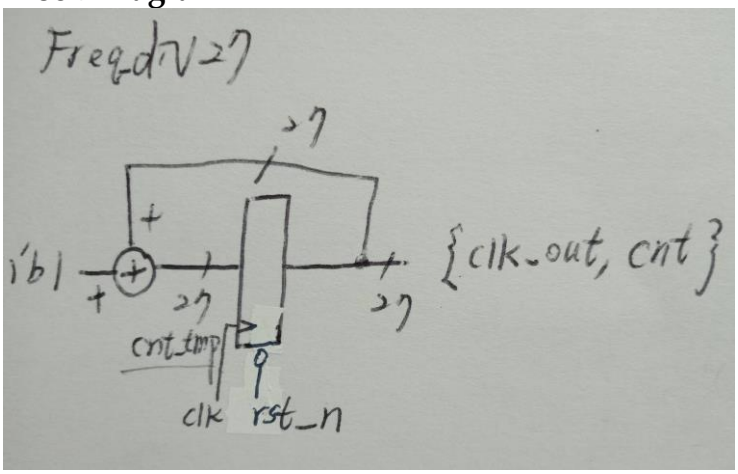
1.
Design Specification

- ✓ **Input / Output**

  For a 1/ (2^27) frequency divider:

  Input: clk, rst_n.
  Output: clk_out.

- ✓ **Block Diagram**



Design Implementation

- ✓ **Function Descriptions**

  此功能為作一個 1/(2^27)的除頻器，使原本為 100MHZ 的石英震盪器除頻為接近 1HZ 的頻率輸出。
  此功能設計的方法為作一個從 0 數到 2^27-1 的 counter 累加器, 取 27bit 數中的最高位做為 clk 的輸出(clk_out)，就可以得到 1/(2^27)的除頻器。Counter 的做法為使 DFF 中的 reset(rst_n)設定初始值為 0,然後每次的數字為前一次數字加一，直到數到 2^27-1 又會歸零(忽略 overflow)。

- ✓ **Logic Functions**

  *Combinational logics
  cnt_tmp = {clk_out, cnt} + 1'b1;
  ＊Sequential logics (for DFFs)
  if (~rst_n), {clk_out, cnt} <=27'd0;
  else {clk_out, cnt} <= cnt_tmp;

✓ **I/O pin**

I/O	clk	rst_n	clk_out
LOC	W15	V17	U16

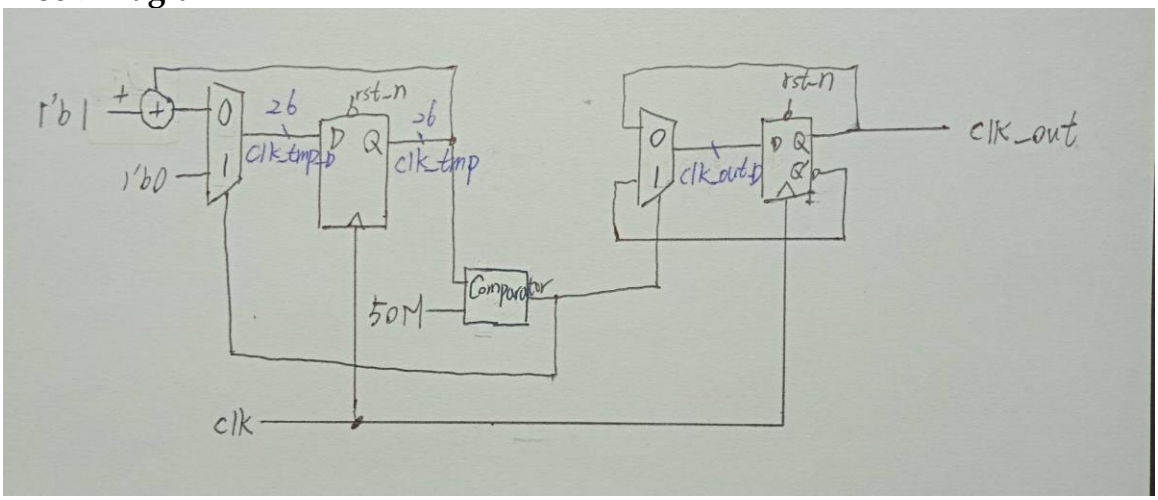
2.
Design Specification

✓ **Input / Output**

For a 1hz frequency divider:

Input: clk, rst_n.
Output: clk_out.

✓ **Block Diagram**



Design Implementation

✓ **Function Descriptions**

此功能為作一個輸出為 1hz 的除頻器。與上題不同的是，此除頻器要輸出確切為 1hz 的頻率。

我的做法為，作一個從 0 數到 50M 的累加器，輸出的結果用 if 來作 comparator 與 mux 的功能，判斷是否數到 50M 了。若是，則把累加器歸零開始數然後讓 toggle flip flop toggle 一次，若還沒數到 50M，則累加器繼續加 1，然後 toggle flip flop 維持原值。這樣一來，toggle flip flop 輸出的頻率就為 1hz。

✓ **Logic Functions**

*Combinational logics
if (clk_tmp == 50000000)
clk_tmp_D = 1'b0

```
    clk_out_D = ~clk_out
    else
    clk_tmp_D = clk_tmp + 1'b1
    clk_out_D = clk_out
     ＊Sequential logics (for DFFs)
    if (~rst_n)
    clk_tmp <= 1'b0;
  clk_out <= 1'b0;

 else
      clk_tmp <= clk_tmp_D;
      clk_out <= clk_out_D;
```

✓ **I/O pin**

I/O	clk	rst_n	clk_out
LOC	W15	V17	U16

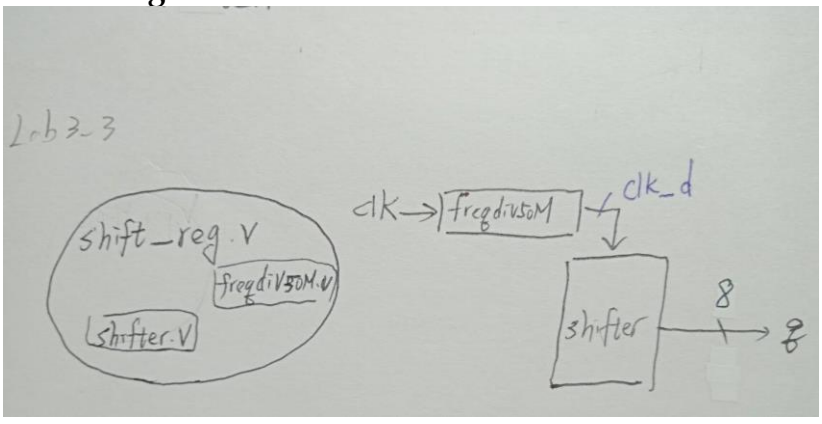
3.
Design Specification

✓ **Input / Output**

For a 8 bit shift register:

Input: clk, rst_n.
Output: q[7:0].

✓ **Block Diagram**



Design Implementation

✓ **Function Descriptions**

此題為 prelab 2 的延伸題，不同的地方在於 shifter 的 clock 輸入應先被除頻為 1hz。因此本功能為以 1s shift 一次，並且把最後一個 DFF 的輸出當作第一個 DFF 的輸　入的 ring counter，由 reset 時輸進初始值。
本題做法為，作一個除頻器輸出 1hz 的 clock 當作 shifter 的 clock input，除頻器的做法與上題相同，而 shifter 的做法為，讓 T_tmp 的最低位為上次 T_tmp 的最高位，其餘的的位數都等於上次的位數減一。

✓ **Logic Functions**

*Combinational logics
Frequency divider 與上題同。
Shifter:
T_tmp [7:0] = {T[6:0], T[7]};

✓ **I/O pin**

I/O	clk	rst_n						
LOC	W15	V17						
I/O	q[7]	q[6]	q[5]	q[4]	q[3]	q[2]	q[1]	q[0]
LOC	V14	U14	U15	W18	V19	U19	E19	U16

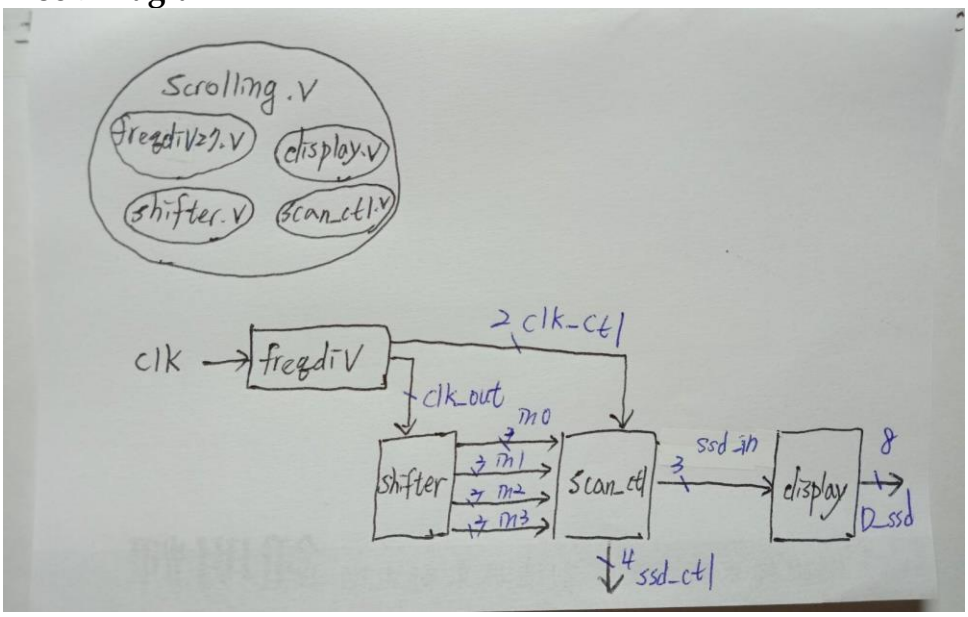
4.
Design Specification

✓ **Input / Output**

For a shift register on a 7-segment display:

Input: clk, rst_n.
output D_ssd [7:0], ssd_ctl [3:0].

✓ **Block Diagram**



Design Implementation

- ✓ **Function Descriptions**

  此功能為讓 shift register 展現在七段顯示器上。除頻器輸出兩種訊號，第一種訊號 1 bit 輸出接近 1hz 的 clock 控制 shifter, 使 NTHUEE 在 shifter 裡 shift，而 shifter 輸出四種訊號，告訴 scan_ctl 這四個訊號(字母)需同時顯現在七段顯示器上。除頻器輸出的第二種訊號輸出的交替頻率大概是使七段顯示器交替顯示字可以是人產生視覺暫留的頻率，因此這個訊號 4 bit 輸出當作 scan_ctl 的 mux 功能用 case 作出來，00->輸出第一個字 01->第二 10->第三 11->第四。接下來 scan_ctl 會輸出現在要輸出的字到 display 中，用 mux 查表把他轉成控制七段顯示器的 8 bit 訊號。

- ✓ **Logic Functions**

  *Combinational logics

  Frequency divider 與上題同。

  Shifter:

  (in DFF)

  in0 <= in1;
  in1 <= in2;
  in2 <= in3;
  in3 <= in4;
  in4 <= in5;
  in5 <= in0;

  Scan_ctl:

  (mux)

  Case(clk_ctl)
  00, ssd_ctl = 4'b0111, ssd_in = in0
  01, ssd_ctl = 4'b1011, ssd_in = in1
  10, ssd_ctl = 4'b1101, ssd_in = in2
  11, ssd_ctl = 4'b1110, ssd_in = in3

  Display:

  (查表)

  000->11010101
  001->11100001
  010->10010001
  011->10000011
  100->01100001

- ✓ **I/O pin**

I/O	clk	rst_n	ssd_ctl[3]	ssd_ctl[2]	ssd_ctl[1]	ssd_ctl[0]		
LOC	W15	V17	U2	U4	V4	W4		
I/O	D_ssd[7]	D_ssd[6]	D_ssd[5]	D_ssd[4]	D_ssd[3]	D_ssd[2]	D_ssd[1]	D_ssd[0]
LOC	W7	W6	U8	V8	U5	V5	U7	V7

Discussion

這次的實驗幾乎都會用到除頻器，實驗 3-1 作出來的除頻器的確比 3-2 較快因為 2^27>100M，並不是真的除到 1hz。利用 counter 數到 50M 再 toggle 一次的確可以得到 1hz 因為 50M*2 = 100M。
這次在作第四題跑馬燈時，找錯找了好久。不知道為什麼 NTUHUEE 一值只停留在 N。這時我就想既然 shifter 看起來沒有問題，問題一定在除頻器出來的頻率有問題了。果真，在 code 裡面因為多打了一個位數導致控制七段顯示器的 clock 沒有正常出現。

Conclusion

這次實驗學到如何使用 counter 作出除頻器，以及製作 shift register 與利用視覺暫留使七段顯示器看起來同時顯示不同字母。
第四個實驗第一次用到那麼多的小 module，這時流程圖就很重要，才能看懂輸入輸出有沒有接上。