Lab 3 Report

107061218 謝霖泳

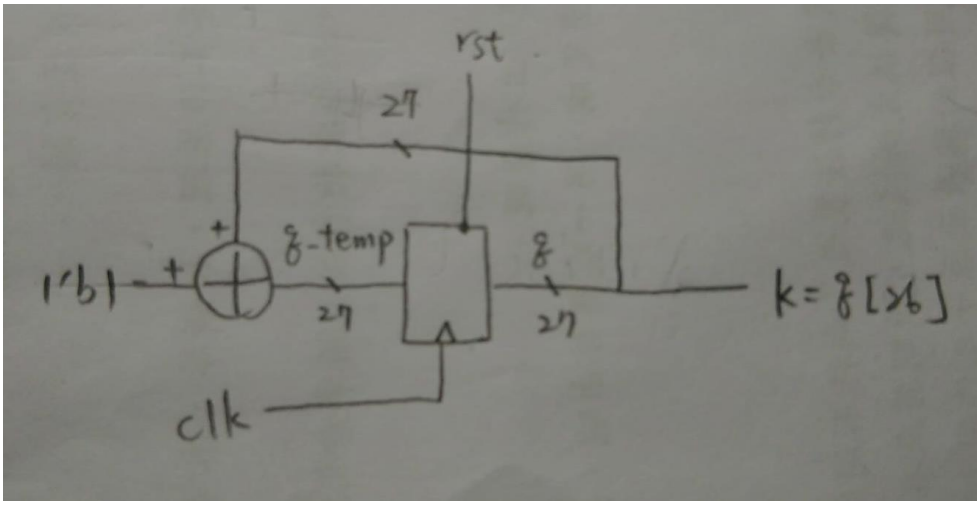
1.

- Design Specification

  For a frequency divider using MSB:

  Input: clk, rst

  Output: k



- Design Implementation

  一個 counter 他每往左邊一位，變動的頻率都是右邊的一半，以 3-bit 為例：

0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

  可發現每往左一位，變動頻率減半。

  因此，我們利用這個概念，可以製作出一個頻率接近 1Hz 的 clock。在 FPGA 板的 W5 腳位，是一個頻率為 100MHz 的 clock，將這個頻率做為 LSB 的 clock，再來，每往左一個 bit，頻率就會減半。因此，為了讓除頻完的頻率接近 1Hz，我們設計了一個 27-bit 的 binary adder，其 MSB 的變動頻率為$\frac{100M}{2^{27}}$，約為 0.75Hz，即達到我們要求接近 1Hz 的除頻效果。

在設計中，我們可以看到一個加法器的構造，q_temp 經過前面的加法器運算完後就是現在的 q 再加 1'b1，當 clock 的 positive edge trigger 來的時候，q 的值就更新為剛剛已經+1 的 q_temp，達到加法器的效果。

因此，我們將 MSB，也就是 q [26]作為輸出 k，k 的變化頻率就會接近 1Hz 了。再來，將這個輸出 k，也就是 q [26]的腳位設到最右邊的 LED 燈 U16，即可看到該 LED 燈不斷閃爍。

此外，我也設計了一個 rst，即 low active reset，當它為 0 時，q 的值會被歸零，做為一個當機時的處理方法。

IO pin assignment:

clk	rst	k
W5	V17	U16

- Discussion

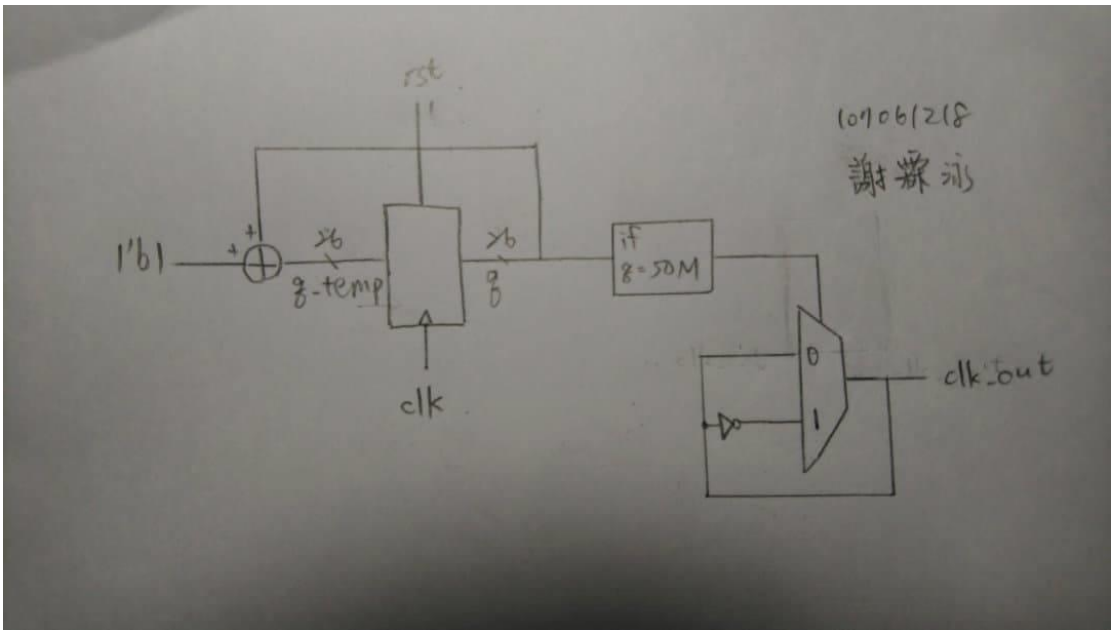
  因為 k 的變動頻率如上面所計算，會略小於 1，因此，週期略大於 1，實測結果無誤，的確會比 1 秒在久一點點。

2.

- Design Specification

  Input: clk, rst

  Output: clk_out



- Design Implementation

設置一個 26-bit 的加法器，若加完的結果為 50M，則輸出的 clk_out 則 toggle。因為晶體震盪的原始頻率為 100MHz，所以，只要加到 50M 的時候讓輸出 toggle，便可製作出一個變化頻率恰為 1Hz 的 clk。

IO pin assignment:

clk	rst	clk_out
W5	V17	U16

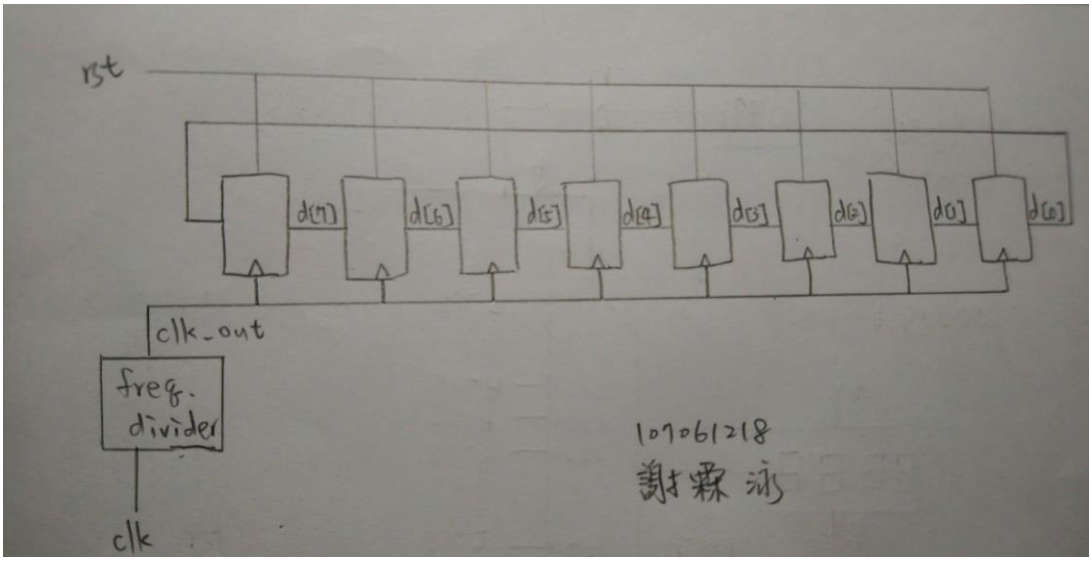
- Discussion

將 output x 的腳位設在最右邊的 LED 燈 U16，program on the device 之後即可觀察到亮+暗剛好為 1 秒鐘，較前一小提閃爍的速度略快一些，肉眼即可察覺這兩題的閃爍速度差異，因為前一提的週期其實略大於 1 秒，因此這一題會閃爍的比較快。我也用碼表實測閃爍 15 次的時間，相當接近 15 秒，可說是相當準確，幾乎沒有誤差。

3.

- Design Specification

For a 8-bit shift register:
Input: clk, rst
Output: [7:0] d



- Design Implementation

由板子提供的原始頻率 100MHz 先經過一個除頻器，得到恰為 1Hz 的頻率，作為這 8 個 FF 的 clock，每當 clock positive edge trigger 時，所有的

bit 就向右移一位，最後一個 FF 的 output 就拉回來最前面作為第一個 FF 的 input，以達到 ring counter 的效果。

當 rst=0 時，8-bit 的 d 就會如同題目要求，被預設為 8'b01010101。我將 output d 的腳位設在 FPGA 板上最右邊的 8 個 LED 燈。

IO pin assignment:

d [7]	d [6]	d [5]	d [4]	d [3]	d [2]	d [1]	d [0]
V14	U14	U15	W18	V19	U19	E19	U16

clk	rst
W5	V17

- Discussion

當 rst=1 時，8 個 LED 燈將會交替閃爍，因為 8-bit 輸出 d 是受除頻器出來的 1Hz 的 clock 所控制，因此，亮暗變化的周期也恰好就是 1 秒。

當 rst=0 時，就回到預設的 01010101，因此，LED 燈由左到右就會依序呈現出暗亮暗亮暗亮暗亮的效果。

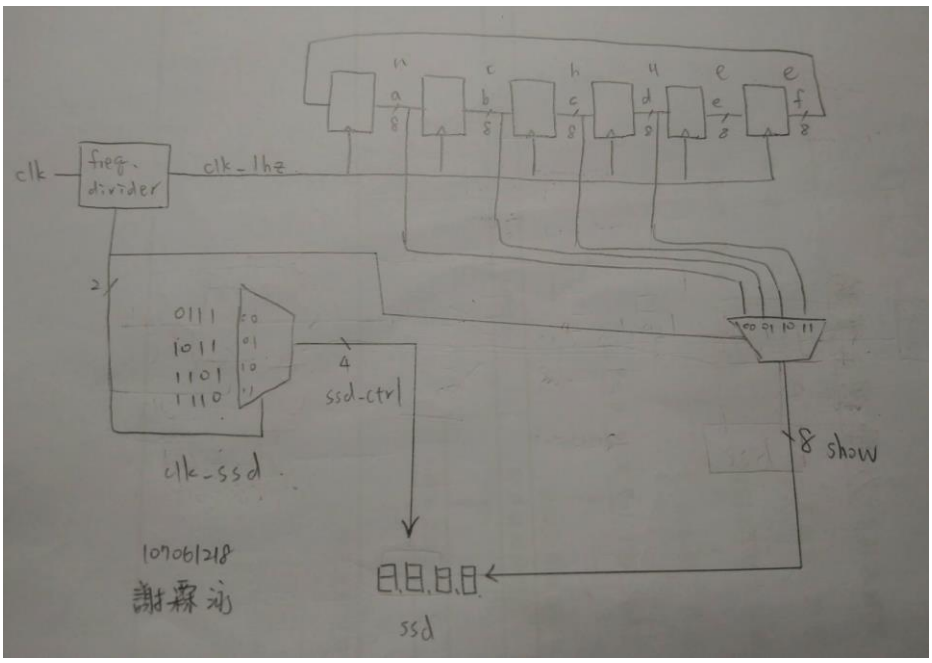
4.

- Design Specification

For a 7-segment display for nthuee:

Input: clk, rst

Output: [7:0] show, [3:0] ssd_ctrl



- Design Implementation

設置 6 個 FF，分別儲存 n、t、H、U、E、E 的 8-bit ssd decoder 的值，取前 4 個字出來顯示。而這 6 個 FF 的 clock 為原始 100MHz 經過 27-bit MSB 加法器所做出之除頻器所得到的大約 1Hz，也就是說，每經過約 1 秒，字就會往左移 1 個單位，也就是利用 shift register 做出跑馬燈的效果。

上段說明如何選取要顯示哪 4 個字，而至於如何成功顯示，則是利用人眼「視覺暫留」的特性，意即當頻率變動很快時，人眼無法觀察出其差異。所以，利用剛剛的 27-bit MSB 除頻器，將第 16、17 個 bit 輸出作為 [1:0] clk_ssd 視覺暫留的工具，當 clk_ssd=2'b00 時，則 ssd_ctrl = 4'b0111，也就是亮第 1 個位數；當 clk_ssd=2'b01 時，則 ssd_ctrl = 4'b1011，也就是亮第 2 個位數；當 clk_ssd=2'b10 時，則 ssd_ctrl = 4'b1101，也就是亮第 3 個位數；當 clk_ssd=2'b11 時，則 ssd_ctrl = 4'b1110，也就是亮第 4 個位數。

此外，[1:0] clk_ssd 也做為要亮哪一個字的 MUX 的 selection，在途中也就是右邊的 MUX，當 clk_ssd=2'b00 時亮第 1 字；當 clk_ssd=2'b01 時亮第 2 字；當 clk_ssd=2'b10 時亮第 3 字；當 clk_ssd=2'b11 時亮第 4 字。

綜上所述，當 clk_ssd=2'b00 時，第 1 個位數會亮第 1 個字；當 clk_ssd=2'b01 時，第 2 個位數會亮第 2 個字；當 clk_ssd=2'b10 時，第 3 個位數會亮第 3 個字；當 clk_ssd=2'b11 時，第 4 個位數會亮第 4 個字。因為其變化頻率極高，所以人眼產生視覺暫留，就會產生 4 個字同時顯示的錯覺。

IO pin assignment:

show [7]	show [6]	show [5]	show [4]	show [3]	show [2]	show [1]	show [0]
V7	U7	V5	U5	V8	U8	W6	W7

ssd_ctrl [3]	ssd_ctrl [2]	ssd_ctrl [1]	ssd_ctrl [0]	clk	rst
W4	V4	U4	U2	W5	V17

- Discussion

在設計出來的跑馬燈中，我額外觀察到了一個現象，就是 rst=0 時，跑馬燈停止運作，僅依 rst 部分的設定在最左邊的位數顯示"n"。之後 rst 變為 1 時，跑馬燈開始運作，這時候肉眼即可觀察到一個有趣現象，就是字的亮度稍微變低了，推測其可能原因為跑馬燈沒運作時，該位數穩定持續顯示，但跑馬燈開始運作後，是每個位數間極快速的變換造成視覺暫留，

因為是每個 digit 間快速變換，不是穩定持續顯示，所以才會感覺亮度才會變暗。

5. conclusion

這次的實驗讓我了解到除頻器的做法。除此之外，我發現在 verilog code 中若引用其他 project 的 module，code 會顯得較為清楚明瞭，不會寫一堆東西全部擠在同一個 module 裡面，藉由呼叫 module 的方法也可以讓 code 的 readability 增加。