Lab3: Counters and Shift Registers I

107061112 王昊文

Experiments

1. Frequency Divider: Construct a 27-bit synchronous binary counter. Use the MSB of the counter, we can get a frequency divider which provides a 1/227 frequency output (f_{out}) of the original clock ($f_{crystal}$, 100MHz). Construct a frequency divider of this kind.

Design Specification

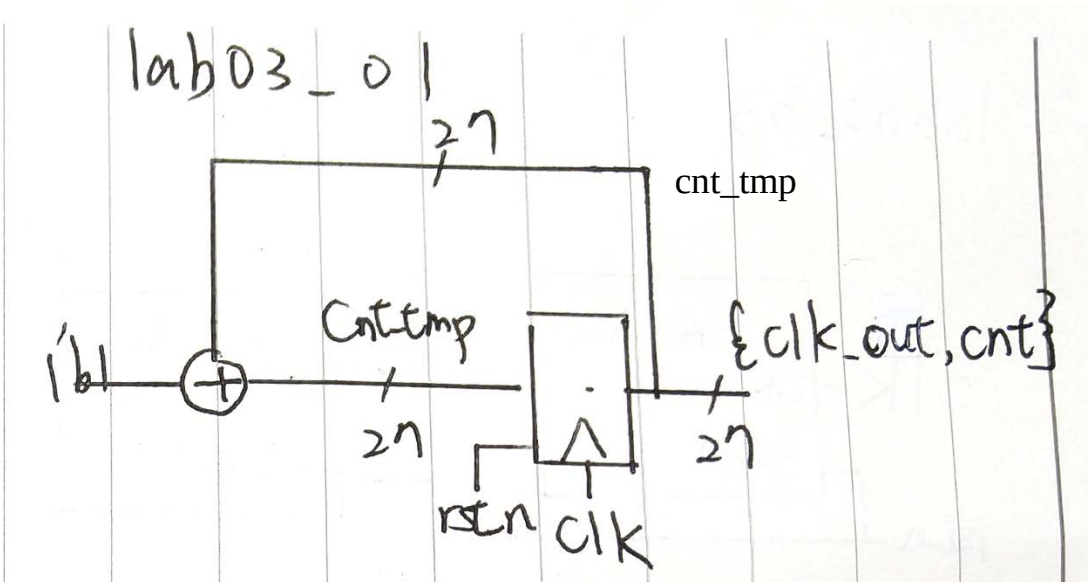
✓ I/O

Input clk // global clock input

Input rst_n // active low reset

Output clk_out // divided output

✓ Logic Block:



Cnt_tmp通過up-counter，將cnt_tmp的MSB當作clk_out輸出。

Design Implementation

✓ Logic Function:

For the flip flop,

If rst_n = 0, {clk_out, cnt} <= `FREQ_DIV_BIT'd0

else {clk_out,cnt} <= cnt_tmp

into the counter, every clock pulse + 1

✓ IO Pin

Input		Output
clk	*rst_n*	*clk_out*
W5	R2	U16

Discussion

這次的lab接觸到Sequential logic的部分，整體難度開始上升。一開始還不清楚 non-blocking assignment的意義，實際應用在這個例子終究很清楚了。在做這個lab之前還以為除頻是一件很困難的事，如果僅用之前的logic gate可能很難拼湊出來，但是透過Verilog的某些語法，把count的第一個MSB當作新的clock輸出即可。

2. Frequency Divider: Use a count-for-50M counter and some glue logics to construct a 1 Hz clock frequency. Construct a frequency divider of this kind.

Design Specification

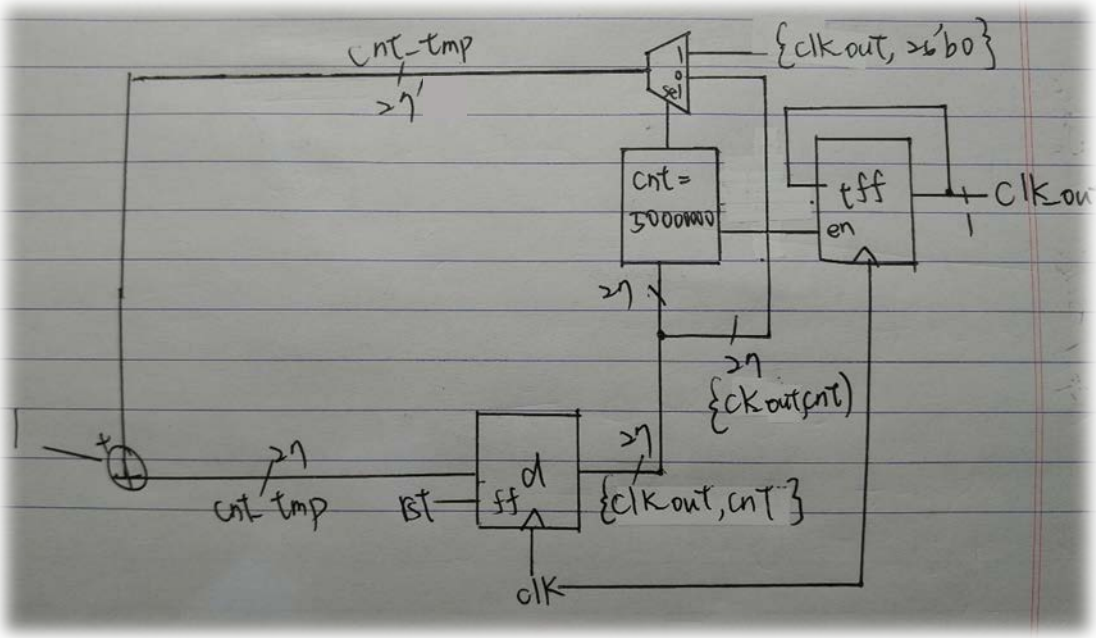
✓ I/O

Input clk // global clock input

Input rst_n // active low reset

Output clk_out // divided output

✓ Logic Block



與前一題類似，多了一個判斷條件cnt是否達到50000000，是則clk_out會進行toggle，mux輸進原值，否則tff不會運作，然後將cnt_tmp值改為0。

Design Implementation

✓ Logic

Still uses a up counter, but adding an additional comparator and a mux.

If (cnt == 50000000)

clk_out toggle, cnt_tmp = {clk_out, cnt}

else cnt_tmp = {clk_out, 26'b0}.

✓ IO Pin

Input		Output
clk	*rst_n*	*clk_out*
W5	R2	U16

Discussion

在這次的實驗中遇到了non-blocking跟blocking assignment的問題。剛開始做的時候發現怎麼做結果都不對，某些assignment，放在non-blocking assignment會出問題，這個lab中讓我好好研究了不同assignment之間的差異。

3. Implement pre-lab2 with clock frequency of 1 Hz.

Design Specification

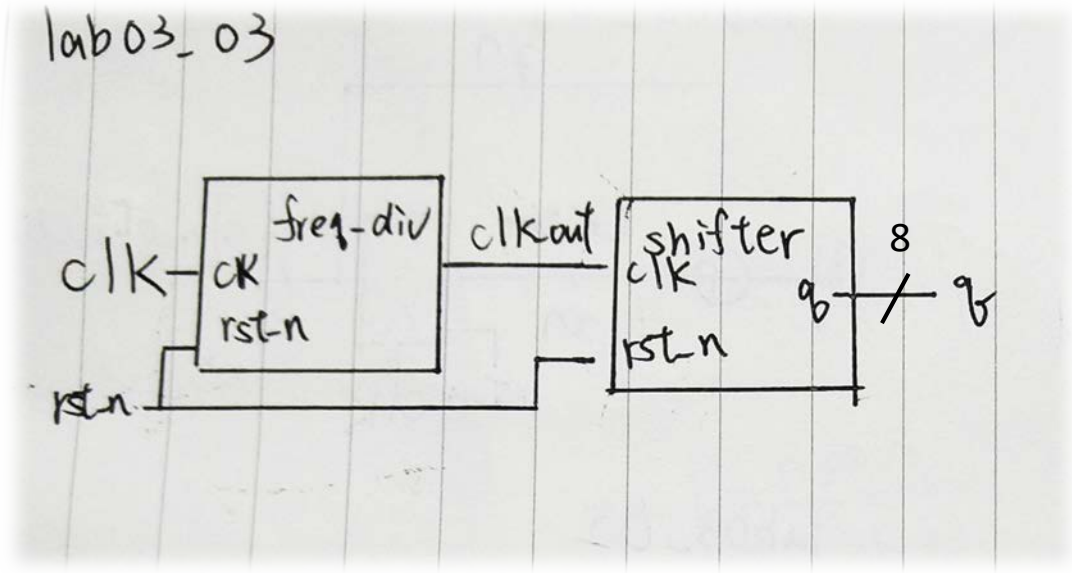
✓ I/O

Input clk // global clock input

Input rst_n // active low reset

Output [7:0]q // shift registers

✓ Logic Block:



將除頻後的Clk送入Pre-Lab的Shifter中。

Design Implementation

✓ Logic

Shifter from pre-lab and the divided clock from exp. 2.

✓ IO Pin

Input	
clk	*rst_n*
W5	R2

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

Discussion

基本上只要把前面的lab皆起來即可。但是從這裡開始一個project裡面的module逐漸變多，接線的時候研究了好一會兒才全部接對。這提算是為後面接線做練習吧！接線的時候發現了logic block的重要性，畫的好的logic block可以大幅減少接線的錯誤率。

4. Use the idea from pre-lab2. We can do something on the seven-segment display. Assume we have the pattern of E, H, N, T, U for seven-segment display as shown below. Try to implement the scrolling pre-stored pattern NTHUEE with the four seven-segment displays.

Design Specification

✓ I/O

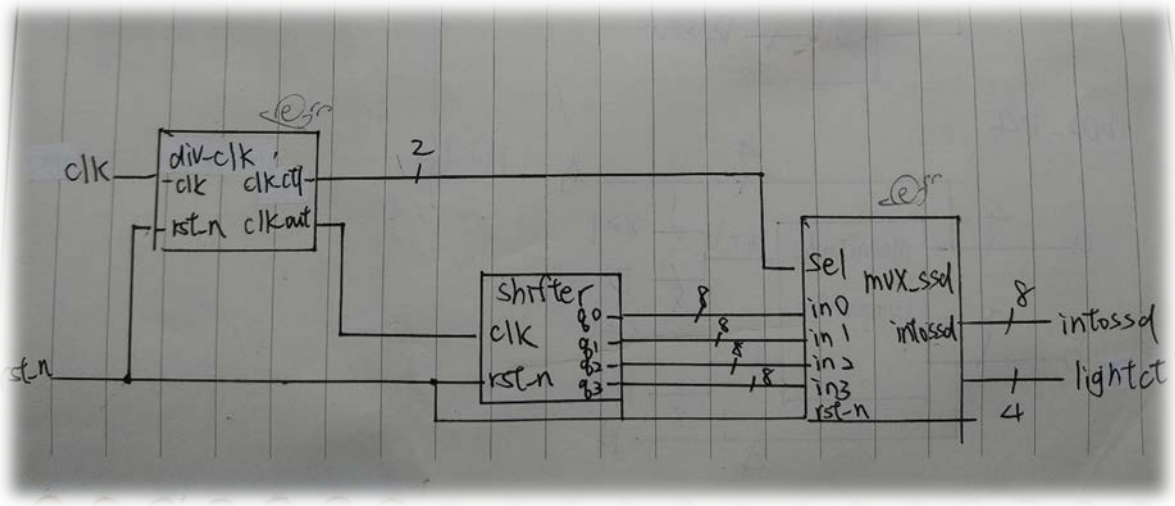
Input clk // global clock input

Input rst_n // active low reset

Output [7:0]ssd // control what to display

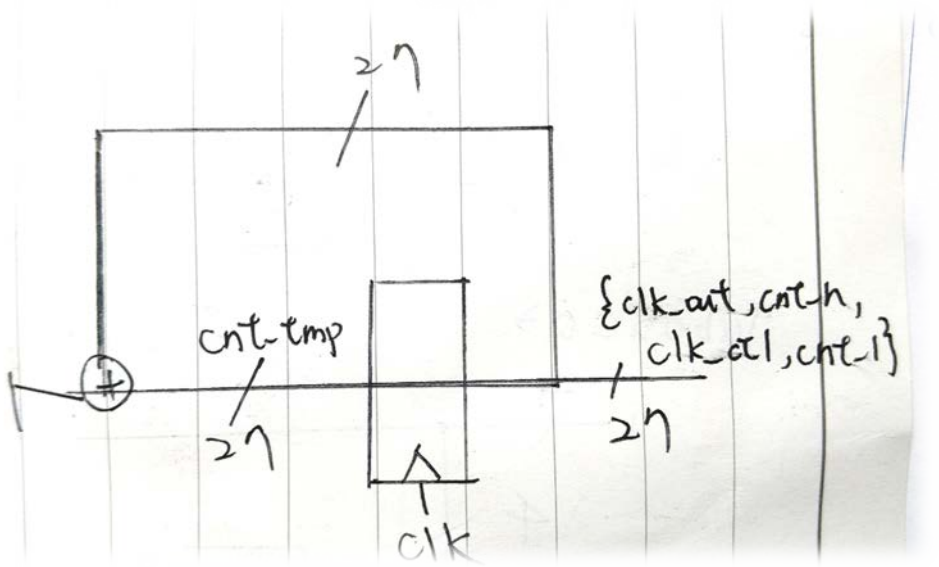
Output [3:0]lightctl // control which segment

✓ Logic Block:



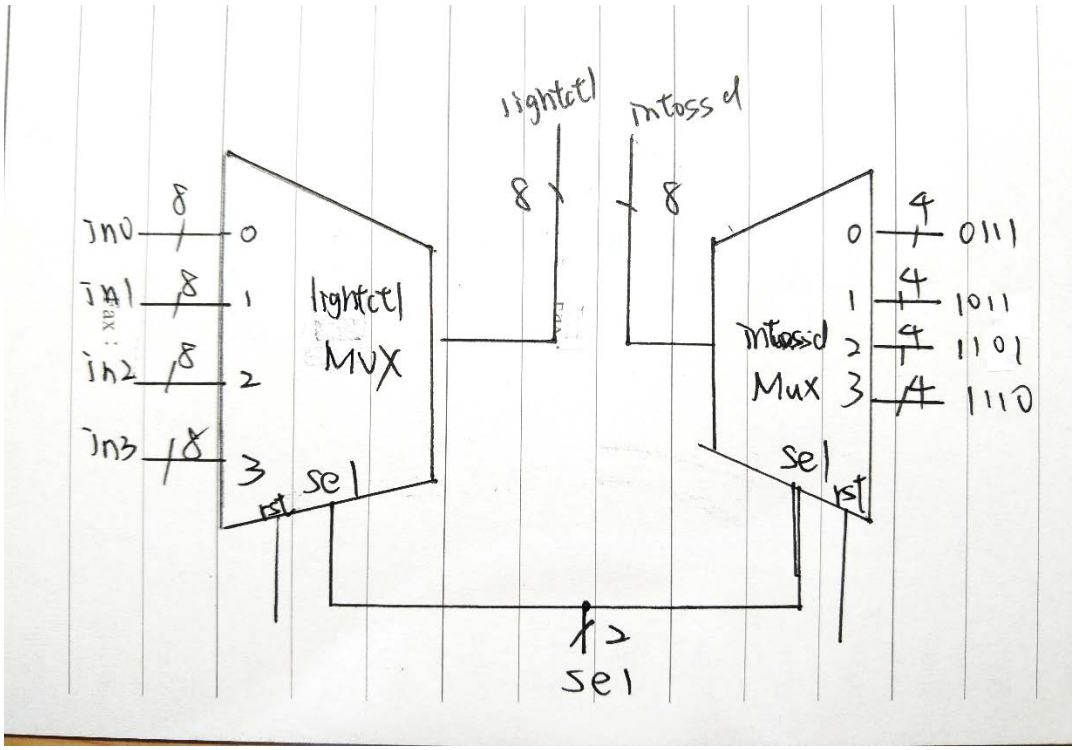
首先將除頻後的clk送入shifter中，shifter將預載的segment資料開始做shift的動作。將結果送入mux_ssd中，功能是將剛剛div_clk中的clk_ctl當作選擇邏輯，在一個clock cycle之內去選擇要輸出的字，最後將An接上lightctl，七段顯示器接上intossd。

以下為Frequency Divider



這次一樣使用除頻過的clk，但是需要使用中間的Clk_ctl作為之後mux_ssd的select。

以下為mux_ssd



將clk_ctl的訊號送進來，當作選擇顯示的數字與燈。當rst訊號送進來時燈顯示0000。

Design Implementation

✓ Logic Function:

For the clock

if (~rst_n)

{clk_out, cnt_h, clk_ctl, cnt_l}<=`FREQ_DIV_BIT'd0

else

{clk_out,cnt_h, clk_ctl, cnt_l}<=cnt_tmp;

To produce the clk_ctl signal.

Shifter from pre-lab。

✓ IO Pin

✓ Input	
clk	*rst_n*
W5	R2

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

output lightctl			
lightctl[0]	lightctl[1]	lightctl[2]	lightctl[3]
W4	V4	U4	V7

Discussion

這一題難度開始變的相當複雜。剛開始完全沒有頭緒該如讓顯示器顯示不同的數字。後來是看完老師的ppt有一頁，我一直思考那個clk_ctl到底是什麼用處，後來才想到透過製造一個更高頻率的clk，使的他可以在一個clock cycle之內變換燈的顯示。這題非常具挑戰性，但是看到他成功顯示NTHUEE的成就感也更高。

透過這個clock的小技巧，終於瞭解如何再版子上顯示不同的字。

Conclusion

這次的lab難度明顯提升，帶入了clock的應用。相信這應該只是正式踏入這塊領域的開端吧！期待下一個lab的挑戰。