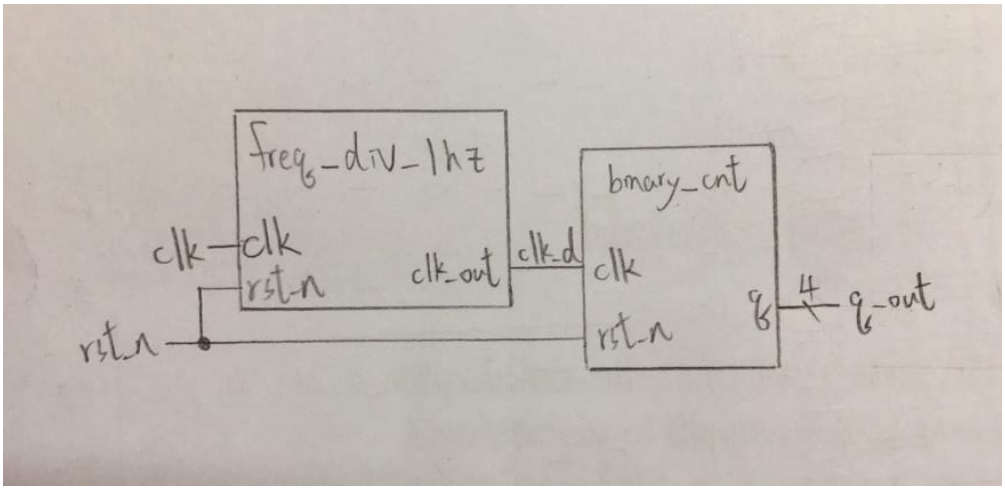
(1) Construct a 4-bit synchronous binary up counter with the 1-Hz clock frequency from lab2 and use 4 LEDs for display.

IO:

Input: clk, rst_n

Output: [3:0]q_out

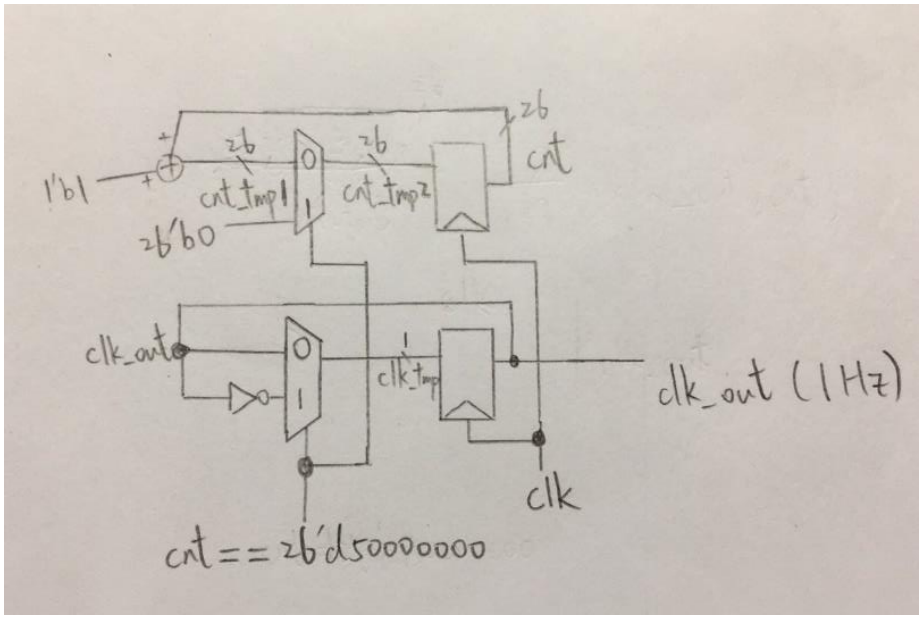
Block diagram:



這個實驗需要兩個子 module 組成，一個是除頻器，另一個是 binary up counter。

Logic diagram:

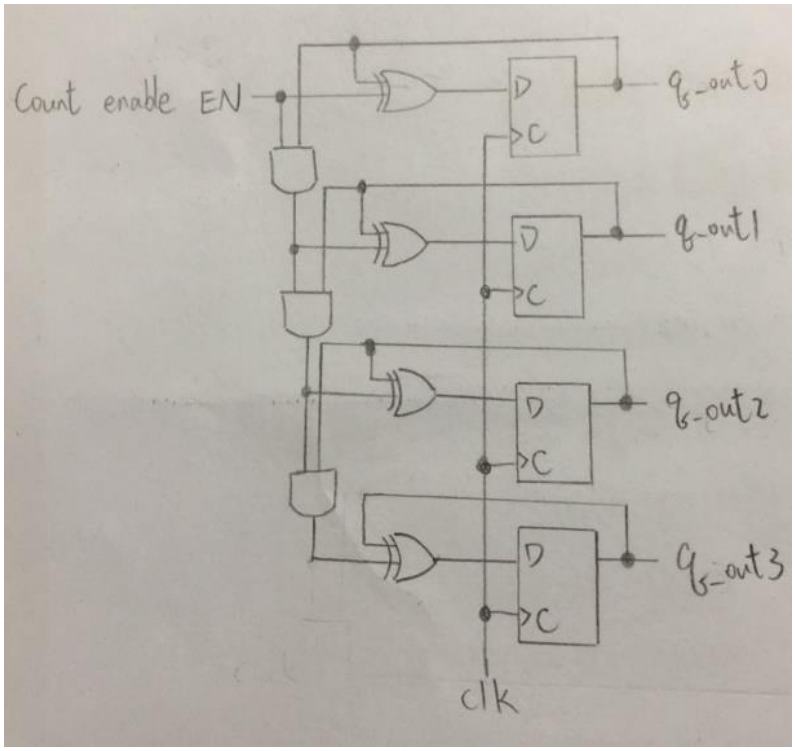
<u>100MHz to 1Hz frequency divider</u>:



由於 FPGA 板上的頻率為 100MHz，建構理念是設計一個上限為 50M 的 binary

up counter，在達到 50M 之前，每個 clk 都讓 counter 加上 1，而當 counter 數到 50M 時，便將輸出值(clk_out) toggle 一次，也將 counter 歸零，如此下來，輸出值(clk_out)在等於 0 和等於 1 的時間各自都是 50M 次的 clk，也就是說，clk_out 的值 0.5 秒會 toggle 一次，因此輸出值的頻率將會等於 100MHz/(50M*2) = 1Hz。

4-bit synchronous binary up counter:



用四個 DFF 組成，在每個 1Hz 的 clock 觸發後，output (4bit 的 q_out) 的值會等於 上一個 clock 的 q_out 值 + 1。

將上述子 module 用 top module 連接起來，可以完成 4-bit synchronous binary up counter with the 1-Hz clock frequency 的功能。

I/O pin assignment:

I/O	**pin**
q_out[3]	V19
q_out[2]	U19
q_out[1]	E19
q_out[0]	U16
clk	W5
rst_n	R2

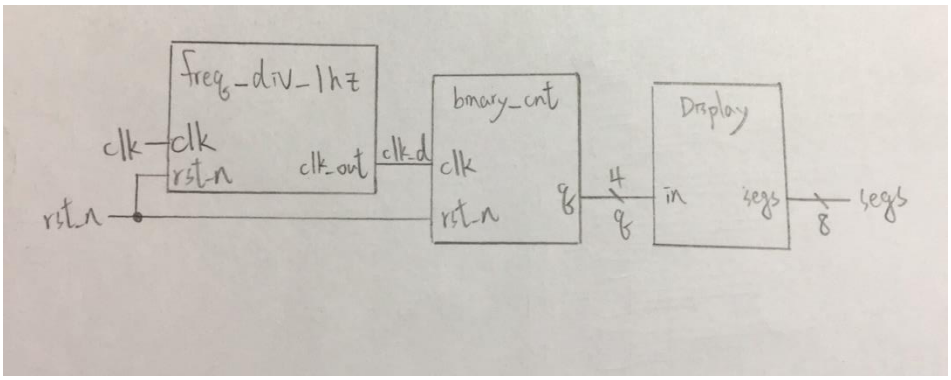
(2) Combine the 4-bit synchronous binary up counter from exp1 with a binary-to-sevensegment-display decoder (from lab2-exp3) to display the binary counting in 7-segment display.

IO:

Input: clk, rst_n

Output: [7:0]segs, [3:0]ssd_ctl

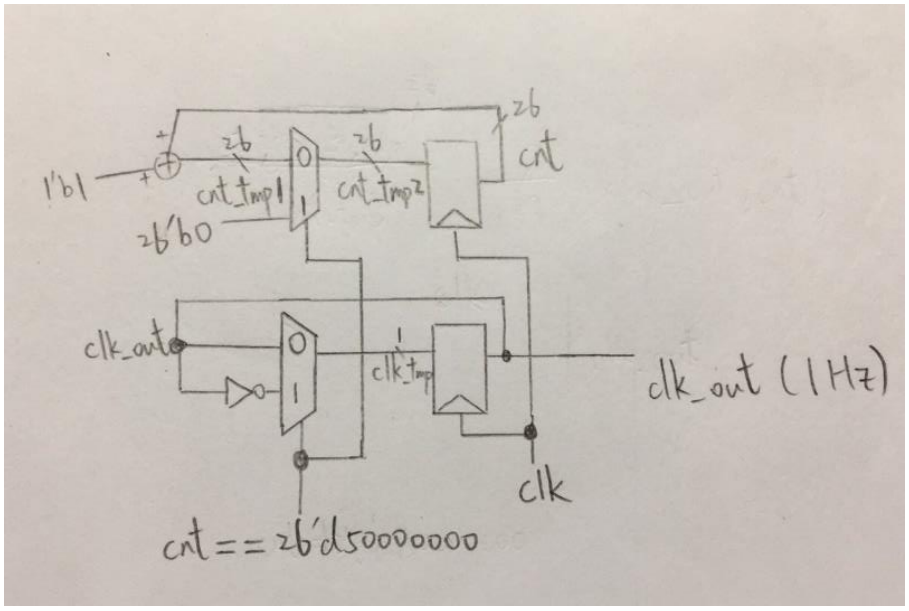
Block diagram:



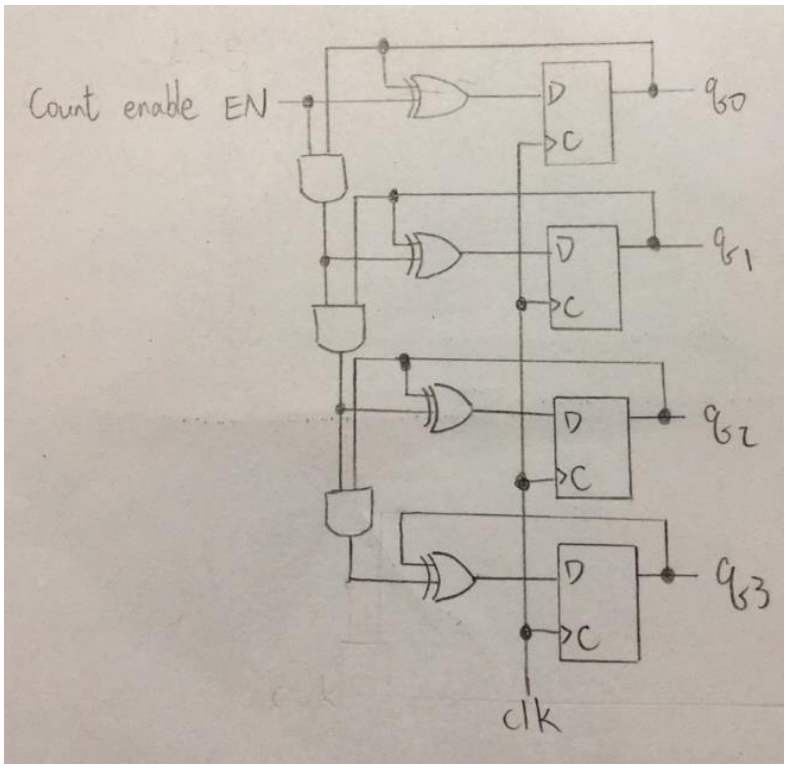
並將[3:0]ssd_ctl 設為 4’b1110，使得七段顯示器只有顯示個位數。

Logic diagram:

<u>100MHz to 1Hz frequency divider</u>:



4-bit synchronous binary up counter:



Binary to 7-segment display decoder:

in = 0 → segs = 8'b0000_0011
in = 1 → segs = 8'b1001_1111
in = 2 → segs = 8'b0010_0101
in = 3 → segs = 8'b0000_1101
in = 4 → segs = 8'b1001_1001
in = 5 → segs = 8'b0100_1001
in = 6 → segs = 8'b0100_0001
in = 7 → segs = 8'b0001_1111
in = 8 → segs = 8'b0000_0001
in = 9 → segs = 8'b0000_1001
in = 10 → segs = 8'b0001_0001
in = 11 → segs = 8'b1100_0001
in = 12 → segs = 8'b0110_0011
in = 13 → segs = 8'b1000_0101
in = 14 → segs = 8'b0110_0001
in = 15 → segs = 8'b0111_0001

Discussion:

本實驗是在上一小題的基礎上再多增加一個 binary to 7-segment display 的

decoder，利用多工器將每個 binary 的 output 解碼成七段顯示器的顯示法，再用 top module 將 binary 的 output 連接至 display decoder 的 input，即可完成一個 4-bit synchronous binary up counter with a binary-to-sevensegment-display decoder。

I/O pin assignment:

I/O	pin
segs[7]	W7
segs[6]	W6
segs[5]	U8
segs[4]	V8
segs[3]	U5
segs[2]	V5
segs[1]	U7
segs[0]	V7
clk	W5
rst_n	R2
ssd_ctl[3]	W4
ssd_ctl[2]	V4
ssd_ctl[1]	U4
ssd_ctl[0]	U2

(3) Construct a single digit BCD up counter with the divided clock as the clock frequency and display on the seven-segment display.

3.1 Construct a BCD up counter.

3.2 Construct a BCD-to-seven-segment display decoder.

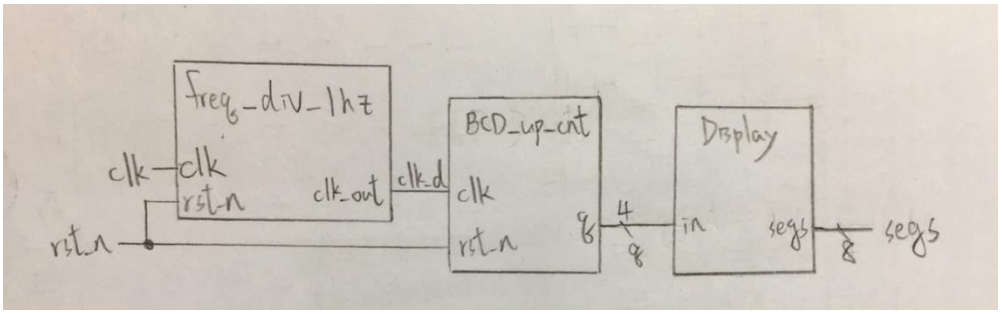
3.3 Combine the above two together.

IO:

Input: clk, rst_n

Output: [7:0]segs, [3:0]ssd_ctl

Block diagram:

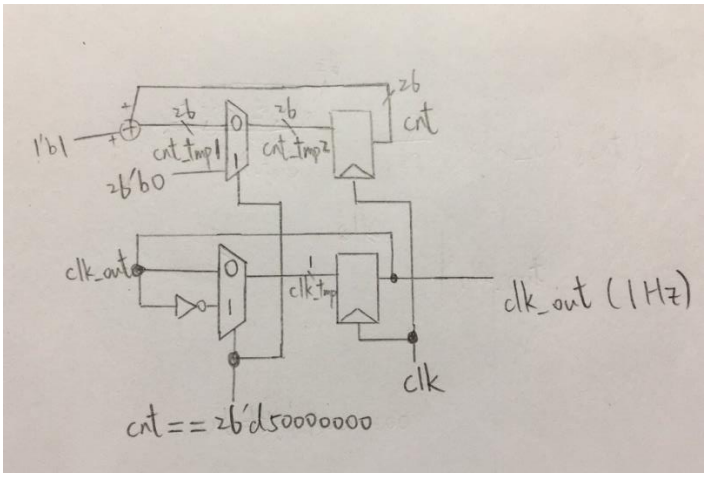


並將[3:0]ssd_ctl 設為 4’b1110，使得七段顯示器只有顯示個位數。

這個實驗需要三個子 module 組成，分別為除頻器、 BCD up counter 和 display decoder。

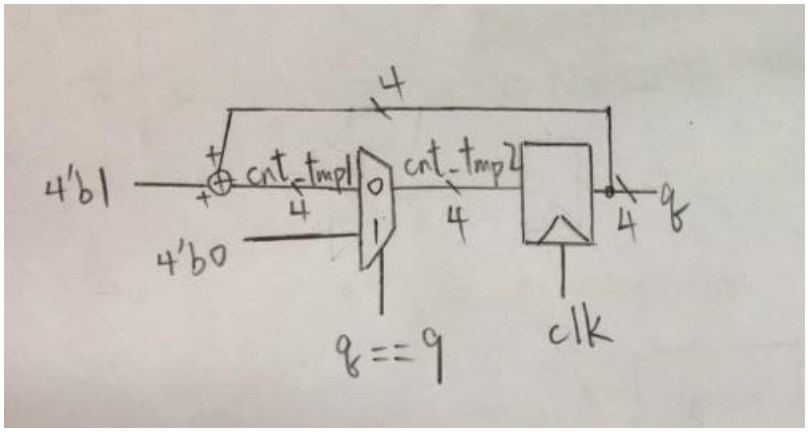
Logic diagram:

<u>100MHz to 1Hz frequency divider</u>:



由於 FPGA 板上的頻率為 100MHz，建構理念是設計一個上限為 50M 的 binary up counter，在達到 50M 之前，每個 clk 都讓 counter 加上 1，而當 counter 數到 50M 時，便將輸出值(clk_out) toggle 一次，也將 counter 歸零，如此下來，輸出值(clk_out)在等於 0 和等於 1 的時間各自都是 50M 次的 clk，也就是說，clk_out 的值 0.5 秒會 toggle 一次，因此輸出值的頻率將會等於 100MHz/(50M*2) = 1Hz。

BCD up counter:



用四個 DFF 和一個多工器組成，每次 cnt_tmp1 的值會+1，利用多工器選擇，當輸出(q) ≠ 9 時 cnt_tmp2 會等於 cnt_tmp1，當輸出(q) = 9 時 cnt_tmp2 會輸入 0，在每個 1Hz 的 clock 的 positive edge 來時，輸出(q)的值會等於 cnt_tmp2。

BCD-to-seven-segment display decoder:

in = 0 → segs = 8'b0000_0011
in = 1 → segs = 8'b1001_1111
in = 2 → segs = 8'b0010_0101
in = 3 → segs = 8'b0000_1101
in = 4 → segs = 8'b1001_1001
in = 5 → segs = 8'b0100_1001
in = 6 → segs = 8'b0100_0001
in = 7 → segs = 8'b0001_1111
in = 8 → segs = 8'b0000_0001
in = 9 → segs = 8'b0000_1001

將 BCD up counter 的輸出(q)轉換成七段顯示器的表示法。

結合上述的 module 可以完成一個 single digit BCD up counter with the divided clock as the clock frequency and display on the seven-segment display 的功能。

I/O pin assignment:

I/O	pin
segs[7]	W7
segs[6]	W6
segs[5]	U8

segs[4]	V8
segs[3]	U5
segs[2]	V5
segs[1]	U7
segs[0]	V7
clk	W5
rst_n	R2
ssd_ctl[3]	W4
ssd_ctl[2]	V4
ssd_ctl[1]	U4
ssd_ctl[0]	U2

(4) Construct a single digit BCD down counter with the divided clock as the clock frequency and display on the seven-segment display.

4.1 Construct a BCD down counter.

4.2 Construct a BCD-to-seven-segment display decoder.

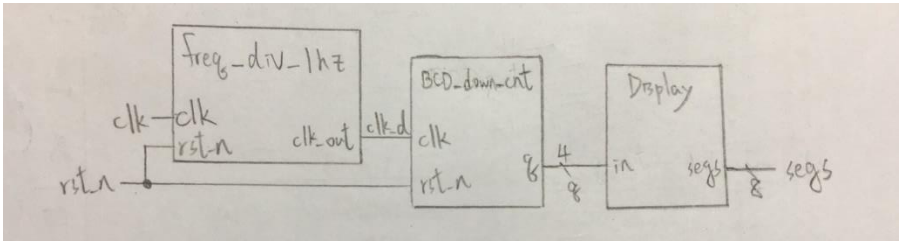
4.3 Combine the above two together.

IO:

Input: clk, rst_n

Output: [7:0]segs, [3:0]ssd_ctl

Block diagram:

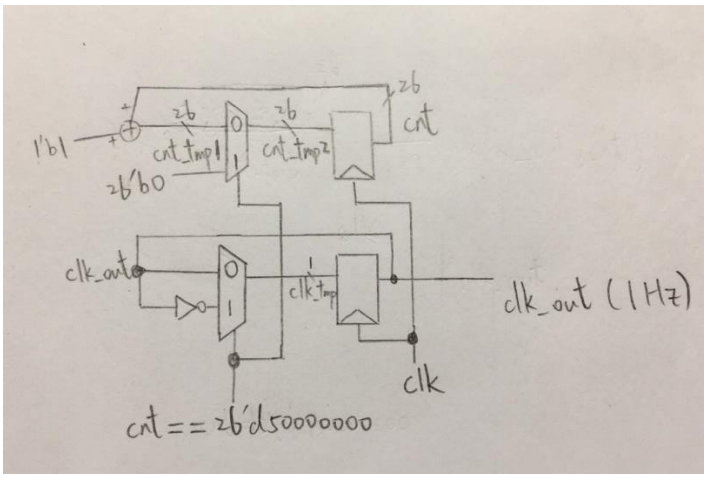


並將[3:0]ssd_ctl 設為 4’b1110，使得七段顯示器只有顯示個位數。

這個實驗需要三個子 module 組成，分別為除頻器、 BCD down counter 和 display decoder。

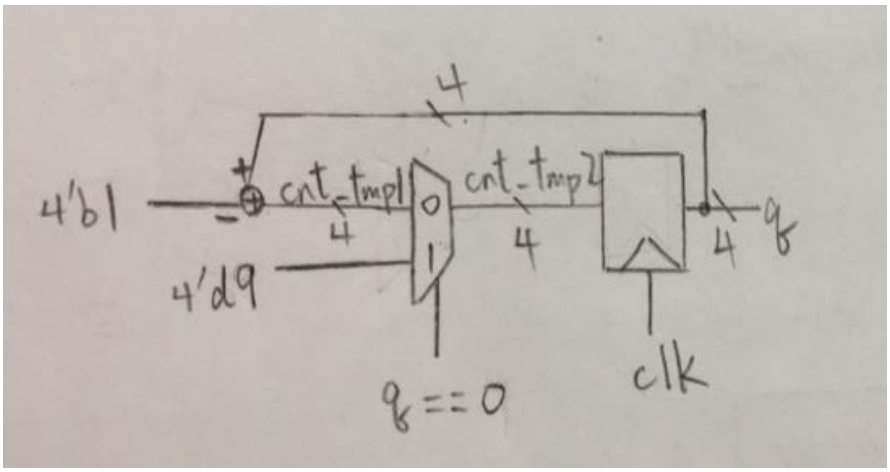
Logic diagram:

100MHz to 1Hz frequency divider:



與第三小題相同。

BCD down counter:



將 cnt_tmp1 每次減 1，並將 MUX 的判斷改成判斷輸出 q 是否等於 0，若等於 0，cnt_tmp2 會等於 9；若不等於 0，則 cnt_tmp2 的值會等於 cnt_tmp1，而在每個 1Hz 的 clock 的 positive edge 來時，輸出(q)的值會等於 cnt_tmp2。

BCD-to-seven-segment display decoder:

in = 0 → segs = 8'b0000_0011

in = 1 → segs = 8'b1001_1111

in = 2 → segs = 8'b0010_0101

in = 3 → segs = 8'b0000_1101

in = 4 → segs = 8'b1001_1001

in = 5 → segs = 8'b0100_1001

in = 6 → segs = 8'b0100_0001

in = 7 → segs = 8'b0001_1111

in = 8 → segs = 8'b0000_0001
in = 9 → segs = 8'b0000_1001

結合以上 module 可以完成一個 single digit BCD down counter with the divided clock as the clock frequency and display on the seven-segment display 的功能。

I/O pin assignment:

I/O	pin
segs[7]	W7
segs[6]	W6
segs[5]	U8
segs[4]	V8
segs[3]	U5
segs[2]	V5
segs[1]	U7
segs[0]	V7
clk	W5
rst_n	R2
ssd_ctl[3]	W4
ssd_ctl[2]	V4
ssd_ctl[1]	U4
ssd_ctl[0]	U2

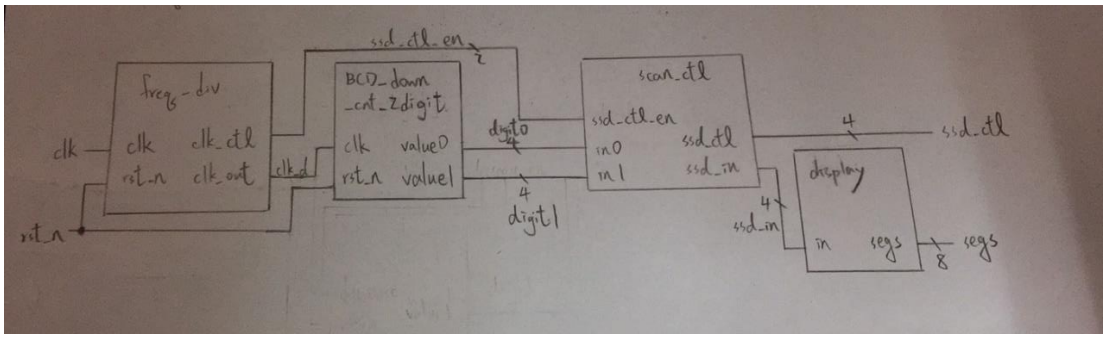
(5) Construct a 30-second count down timer (stop at 00).

IO:

Input: clk, rst_n

Output: [3:0]ssd_ctl, [7:0]segs

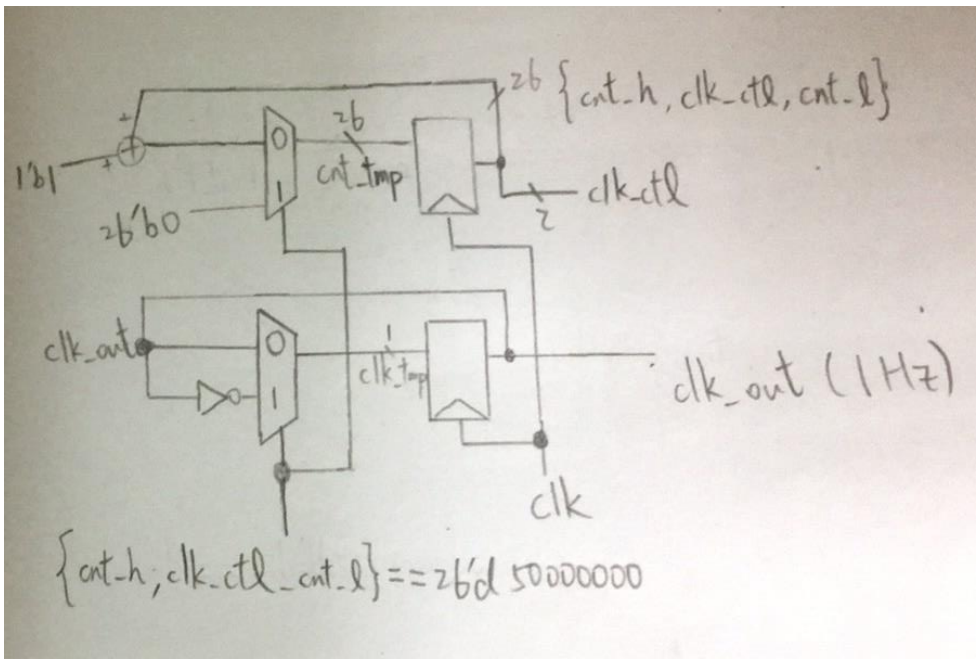
Block diagram:



本實驗需要四個子 module 組合而成，分別為除頻器、二位數的 BCD down counter、scan control 以及 display decoder。

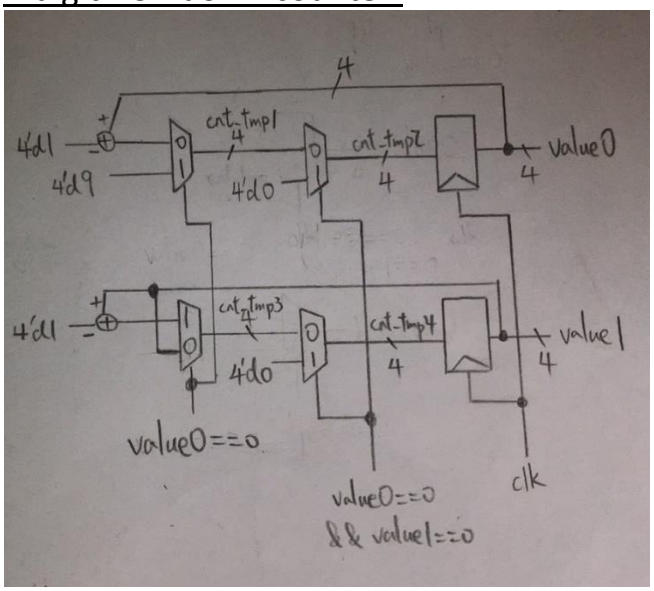
Logic diagram:

100MHz to 1Hz frequency divider with 2-bit clock control:



跟前幾小題不同的地方是這題的除頻器需要另外將上限為 50M 的 counter 中的第 16、17bit 抓出來當作 scan control 的控制項，如此可以藉由視覺暫留達成 4 個七段顯示器分別顯示各自數字的效果。

2-digit BCD down counter:



需要製作一個 30→0 的倒數計時器。

利用多工器選擇 cnt_tmp1、cnt_tmp3 的值，當個位數(value0)≠0 時，代表個位數不需要重置成 9→cnt_tmp1 = 個位數(value0)減一；個位數不需要借位，十位數(value1)維持原本的值→cnt_tmp3 = value1；當個位數 = 0 時，代表個位數下一次從 9 開始→cnt_tmp1 = 9；個位數需要借位，十位數減一→cnt_tmp3 =

value1 – 1。
第二個多工器判斷倒數是否結束(value1 = value0 = 0)，若未結束→cnt_tmp2 = cnt_tmp1、cnt_tmp4 = cnt_tmp3；若已結束→cnt_tmp2 = 0、cnt_tmp4 = 0。
在每個 1Hz 的 clock 的 positive edge 來時，輸出 value0、value1 的值分別會等於 cnt_tmp2、cnt_tmp4。

Scan control:

ssd_ctl_en = 00 → ssd_ctl = 0111
ssd_ctl_en = 01 → ssd_ctl = 1011
ssd_ctl_en = 10 → ssd_ctl = 1101
ssd_ctl_en = 11 → ssd_ctl = 1110

四者輪流顯示，而且要小心的地方是 ssd_ctl 為 low active，因此 ssd_ctl_en = 00 時→顯示第一個七段顯示器(全暗)；ssd_ctl_en = 01 時→顯示第二個七段顯示器(全暗)；ssd_ctl_en = 10 時→顯示第三個七段顯示器(十位數 value1)；ssd_ctl_en = 11 時→顯示第四個七段顯示器(個位數 value0)。

BCD-to-seven-segment display decoder:

in = 0 → segs = 8'b0000_0011
in = 1 → segs = 8'b1001_1111
in = 2 → segs = 8'b0010_0101
in = 3 → segs = 8'b0000_1101
in = 4 → segs = 8'b1001_1001
in = 5 → segs = 8'b0100_1001
in = 6 → segs = 8'b0100_0001
in = 7 → segs = 8'b0001_1111
in = 8 → segs = 8'b0000_0001
in = 9 → segs = 8'b0000_1001
default 為 segs = 8’b1111_1111(全暗)

結合所有子 module，可以完成一個 30 秒的倒數計時器。

I/O pin assignment:

I/O	pin
segs[7]	W7
segs[6]	W6
segs[5]	U8

segs[4]	V8
segs[3]	U5
segs[2]	V5
segs[1]	U7
segs[0]	V7
clk	W5
rst_n	R2
ssd_ctl[3]	W4
ssd_ctl[2]	V4
ssd_ctl[1]	U4
ssd_ctl[0]	U2

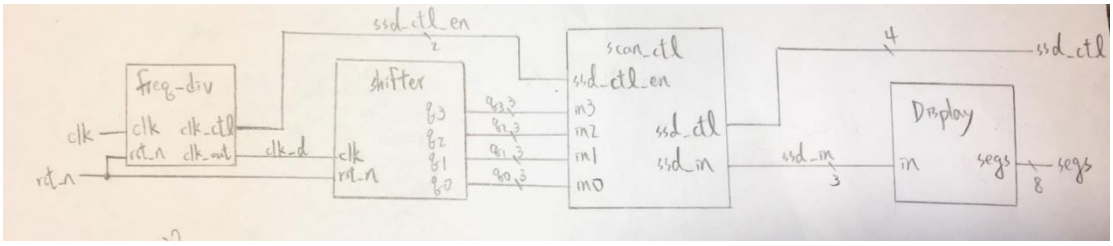
(6) Implement the scrolling pre-stored pattern NTHUEE with the four seven-segment displays.

IO:

Input: clk, rst_n

Output: [3:0]ssd_ctl, [7:0]segs

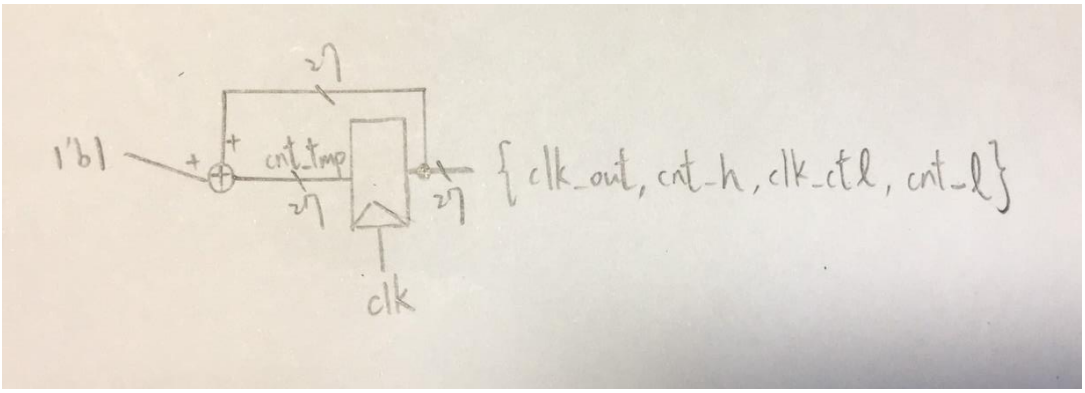
Block diagram:



本實驗需要用到四個子 module，分別為除頻器、4bit 輸出的 shift register、scan control 以及 display decoder。

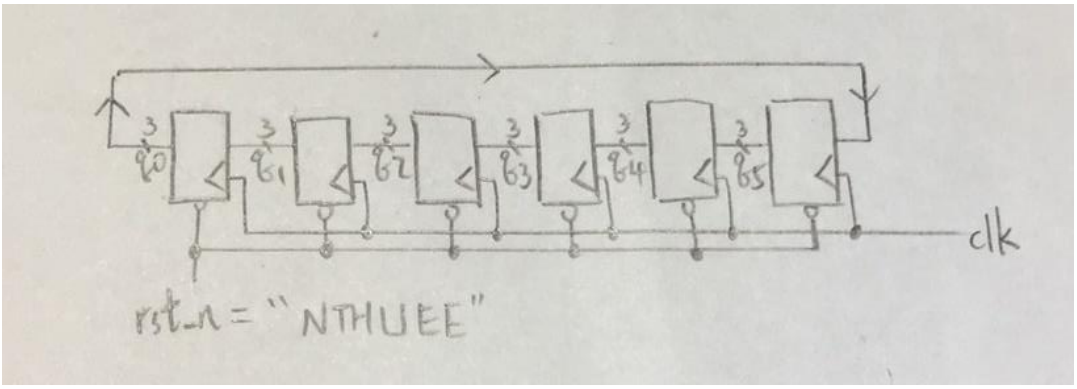
Logic diagram:

Frequency Divider with 1/2^{27} original frequency:



由於原本在 FPGA 板上的頻率為 100MHz，因此透過一個 27bit 的 binary up counter，並將此 27bit 的最高位數(clk_out)當作輸出，便可將頻率除到接近 1Hz (100MHz / 2^{27}≒0.745Hz)。另外再將 counter 中的第 16、17bit 抓出來當作 scan control 的控制項。

4bit 輸出的 shift register:



雖然有六組 3bit 的 DFF，但總共只有四個七段顯示器，因此將前四組 flip flop 的輸出(q3～q0)當作要顯示出來的英文字母。將每個英文字母配對到一個數字，當 reset 的時候，讓六組 flip flop 的輸出按照順序存入"NTHUEE"相對應的數字，並在每一次 clk 來時讓各組 flip flop 的值順時針移動，再把前四組 flip flop 的輸出抓出來，便可達到跑馬燈的效果。

display decoder:

in = 0 → segs = 8'b1101_0101 (N)

in = 1 → segs = 8'b1110_0001 (T)

in = 2 → segs = 8'b1001_0001 (H)

in = 3 → segs = 8'b1000_0011 (U)

in = 4 → segs = 8'b0110_0001 (E)

將 q0～q3 的數字轉換成所代表的英文字母，進而顯示在七段顯示器上。

scan control:

ssd_ctl_en = 00 → ssd_ctl = 0111
ssd_ctl_en = 01 → ssd_ctl = 1011
ssd_ctl_en = 10 → ssd_ctl = 1101
ssd_ctl_en = 11 → ssd_ctl = 1110

四者輪流顯示，ssd_ctl_en = 00 時→第一個七段顯示器顯示 q0；ssd_ctl_en = 01 時→第二個七段顯示器顯示 q1；ssd_ctl_en = 10 時→第三個七段顯示器顯示 q2；ssd_ctl_en = 11 時→第四個七段顯示器顯示 q3。

結合所有 module，可以完成一個 NTHUEE 的跑馬燈。

I/O pin assignment:

I/O	**pin**
segs[7]	W7
segs[6]	W6
segs[5]	U8
segs[4]	V8
segs[3]	U5
segs[2]	V5
segs[1]	U7
segs[0]	V7
clk	W5
rst_n	R2
ssd_ctl[3]	W4
ssd_ctl[2]	V4
ssd_ctl[1]	U4
ssd_ctl[0]	U2

Conclusion:

這次實驗較之前的複雜了許多，除此之外，從 lab3 移到 lab4 的最後一題也花了我不少時間，但畢竟花了那麼多的時間，也讓我更加熟練 verilog 的基本語法以及多個 module 的寫法。

在這次實驗之前，我並不是很熟悉 top module 的連接方式，透過這次的 lab，我可以算是很好的練習過了，畢竟有好幾題複雜的題目，如果只寫一個

module 肯定會很撩亂，邏輯、功能也會不夠清晰，此外，像是除頻器這些功能都是很多小題重複用到的，單獨寫一個 module 也會比較好利用。

這次的題目很多都具有連貫性，有些之前做出來的 module 可以拿到下一題去用，做起來會比較方便。不過由於電路越來越複雜，所以花費的時間比起上次也多了許多。

經過幾次的實驗後，我發現先畫出圖果然還是重要的，思緒也會比較清楚，而不是把程式打好才再想圖怎麼畫。

雖然以後的 lab 題目只會越來越難，但還是希望以後可以更有效率的完成各個實驗。