**Lab 3: Counters and Shift Registers I**

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 (fout) of the original clock (fcrystal, 100MHz). Construct a frequency divider of this kind.

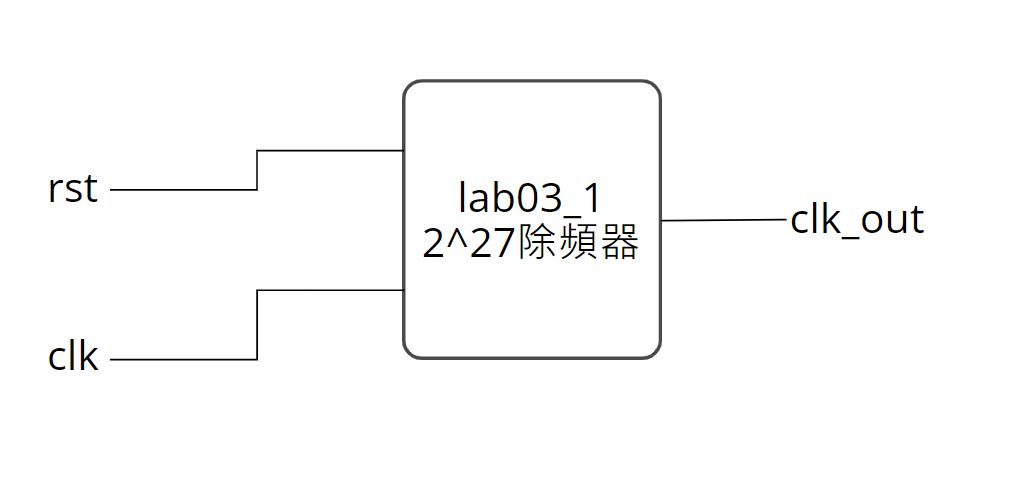
**Design Specification**

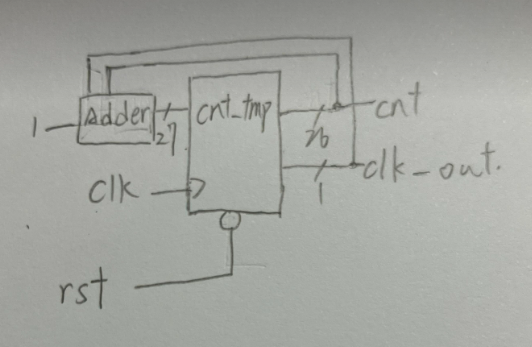
IO設定

輸入: clk (1 bit), rst(1 bit)

輸出: clk\_out (1 bit)

**Logic block**



**Design Implementation**

Logic diagram

見右圖🡪

Logic Func.

cnt\_tmp = (clk\_out, cnt) + 1

~rst🡪 clk\_out, cnt = 0

先用一個cnt\_tmp來記錄當前的計數器的數字，輸出再傳給clk\_out, cnt。clk\_out用於輸出時脈，而cnt是紀錄最高位以外的所有位數，和clk\_out作為下個cnt\_tmp的輸入。

Pin assignment

|  |  |  |  |
| --- | --- | --- | --- |
| IO | clk | rst | clk\_out |
| Pin | W5 | V17 | U16 |

**Discussion**

這次的實驗我覺得最難的地方在於如何確定自己的做出來的程式是對的。我想到的方法是藉由手機錄影，紀錄十次亮燈後所需的時間，就能得出平均閃一次的頻率。而出來的結果離1Hz有點差距，但這是因為100M直接除2^27本來就不是1，所以和1Hz相比還是有點差距。

另外在做正反器時，有想到略過cnt\_tmp，直接設{clk\_out, cnt} = {clk\_out, cnt} + 1。而經過上網查詢侯這種方式也的確行得通。但感覺上跟上學期學的有點不一樣，還是有點不直觀。

**Conclusion**

在這次實驗，我學到了:

* 正反器的使用

第一次接觸正反器，看過老師示範後也覺得很直觀。因此這一小題很快就寫出來了。此外，我發現vivado好像不建議一次同時開啟兩個project。因為在做sim.或syn.的時候vivado很常會搞混，這一點需要注意一下。

**References**

<https://www.chipverify.com/verilog/verilog-4-bit-counter>

這連結有我剛才提到的直接進行加減，不需要經過cnt\_tmp的寫法。

PS: 由於我的筆電跑不完10000多的testbench，所以沒有附上testbench

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.

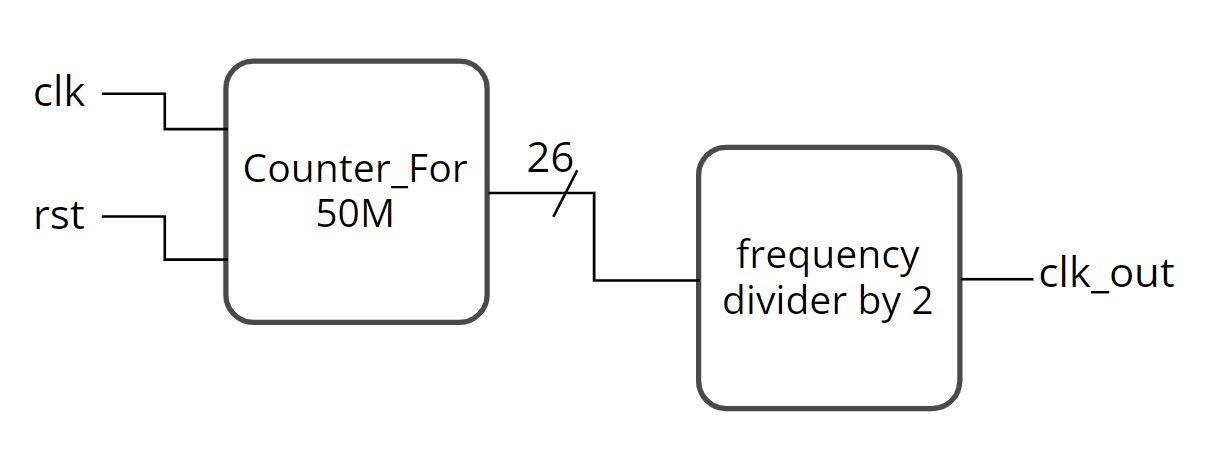
**Design Specification**

IO輸出入設定

輸入: clk(1 bit), rst (1 bit)

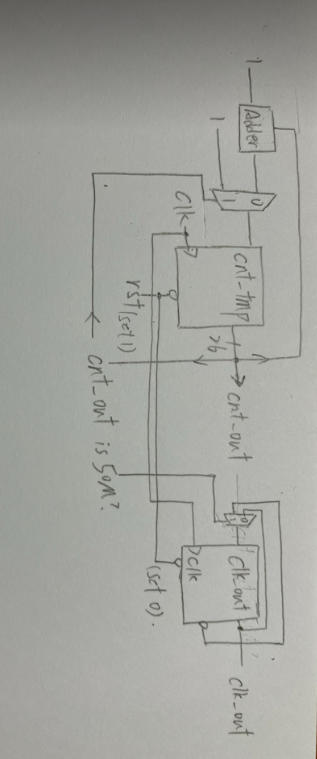
輸出: clk\_out(1 bit)

Logic Block



**Design Implementation**

Logic diagram



先建立一個CounterFor50M的計數器可以數到50M，內容大致上和前面一樣，只是是26bits，且碰到50M(binary: 10111110101111000010000000)就重設。

而後面再接一個2的除頻器。只需要1 bit的計數器便能達成，所以這個計數器就設成clk\_out，不過這裡的rst還是0。

Pin Assignment

|  |  |  |  |
| --- | --- | --- | --- |
| IO | clk | rst | clk\_out |
| Pin | W5 | V17 | U16 |

**Discussion**

在這裡的1Hz clk相比前面就明顯準確許多。檢驗的方法同上，數十次後取平均一次的周期(頻率)。在Counter\_For\_50M實做1Hz的部分，我一開始以為他已經除50M了，所以我只需要在除2就好。但在這裡有個小細節，就是在初始50M Counter的時候，初始值應該設為1(但後面的clk\_out初始值還是設為0)，否則會變成除50M+1的除頻。還有在處理2的除頻時，只需要一位計數器就好，不用到兩位。另外，除2的除頻器我時脈還是接原本的100M，因為我是在正反器裡面判斷50M時做翻轉。當然，也可以用50M的判斷當成clk2Hz的時脈，在接clk2Hz的時脈也可。

**Conclusion**

在這個小題，我學到了除2頻的除頻器以外的除頻方法，以及top module的應用。在這一小題，我覺得不要用top module，直接寫在一起或許也是一個很不錯的方式。且top module在接子module的輸出時，似乎得用wire資料型態區接，少了reg的彈性。

**References**

無

3 Consider a 4-bit synchronous binary up counter (q3q2q1q0).

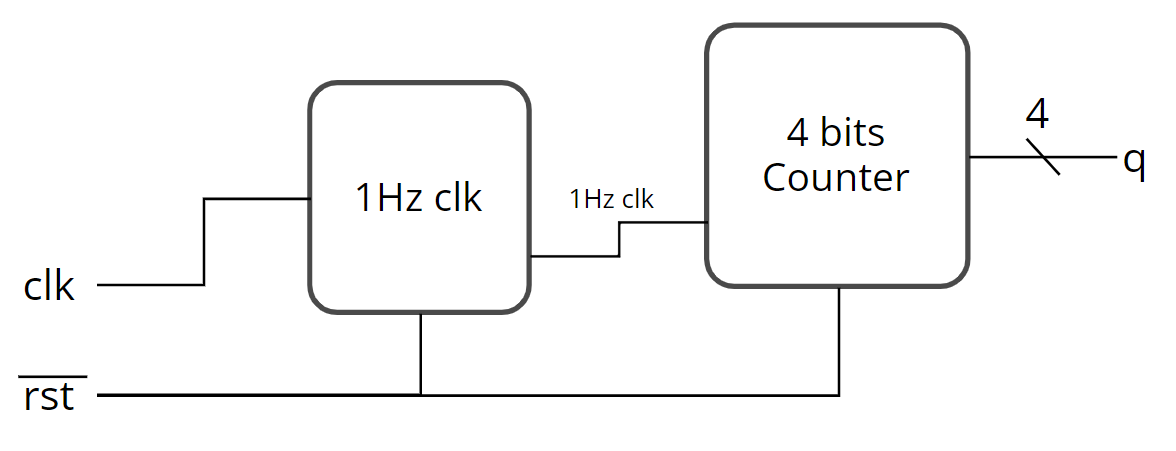
**Design Specification**

IO輸出入設定

輸入: clk(1 bit), rst(1 bit)

輸出: [3:0]q (4 bits)

Logic Block



**Design Implementation**

Logic diagram

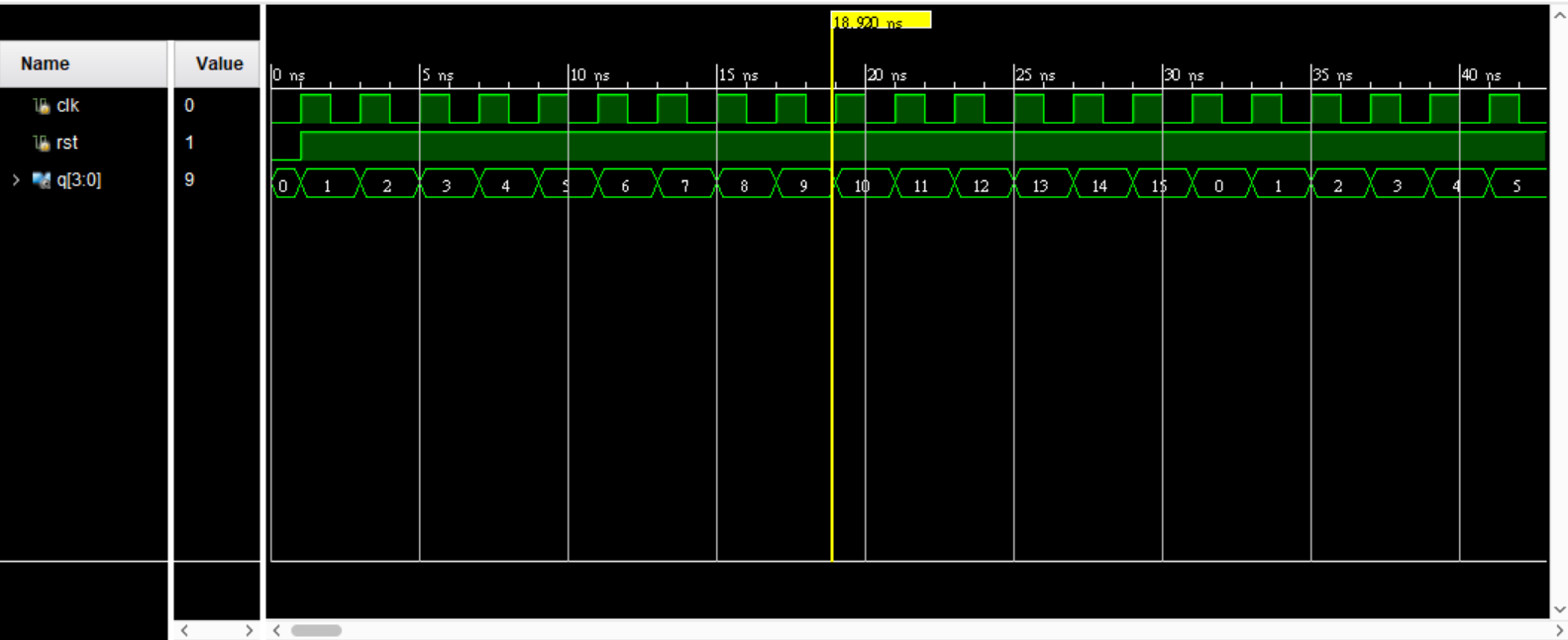
|  |  |
| --- | --- |
|  |  |
| Whole structure | 4 bits counter |

由於要用到1Hz的時脈，所以我直接把前面第二小題搬過來，所以這題我們只要解決4 bits的Counter。所以就照前面除頻器的方式寫計數器的模型，cnt\_tmp作為儲存當前的計數器數到的數字，q(4 bits)作為輸出，而後經加一後傳回cnt\_tmp。

Pin Assignment

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| IO | clk | rst | q3 | q2 | q1 | q0 |
| Pin | W5 | V17 | V19 | U19 | E19 | U16 |

**Discussion**



這裡testbench用的時間clk是我自己設定的，因為1Hz的時間testbench跑不完。四位計數器正確地運行，並無出現其他錯誤。這次就真的得用到top module，模組化的執行較容易找出錯誤在哪裡。

在實做的過程中，我發現盡量將所有功能寫完再合在一起，較不會出現錯誤。例如: 應該把Counter和1Hz clk分開做，最後再用top module串起來。若直接在Counter裡呼叫1Hz clk，反而會因為某些因素導致q會亂跳，在這裡我推測是我在設定時間的部分不一致，所以出現時域上的錯誤。在實做的時候一直因為1Hz clk沒搞定，導致後面q出現一連串的錯誤。加上q設定的時間的問題，讓q的結果跟亂數一樣。

**Conclusion**

在這個小題裡，我更瞭解了top module的操作，對計數器的寫法更是變的熟悉。另外，我也學會了clk在testbench裡的寫法，跟一般的always 有些差異，還有rst的操作。

**References**

無

4 Cascade eight DFFs together as a shift register. Connect the output of the last DFF to the input of the first DFF as a ringer counter. Let the initial value of DFF output after reset be 1101110. Construct the Verilog RTL representation for the logics with verification. Also, Implement the ring counter with clock frequency of 1 Hz. The I/O pins can be assigned by yourself.

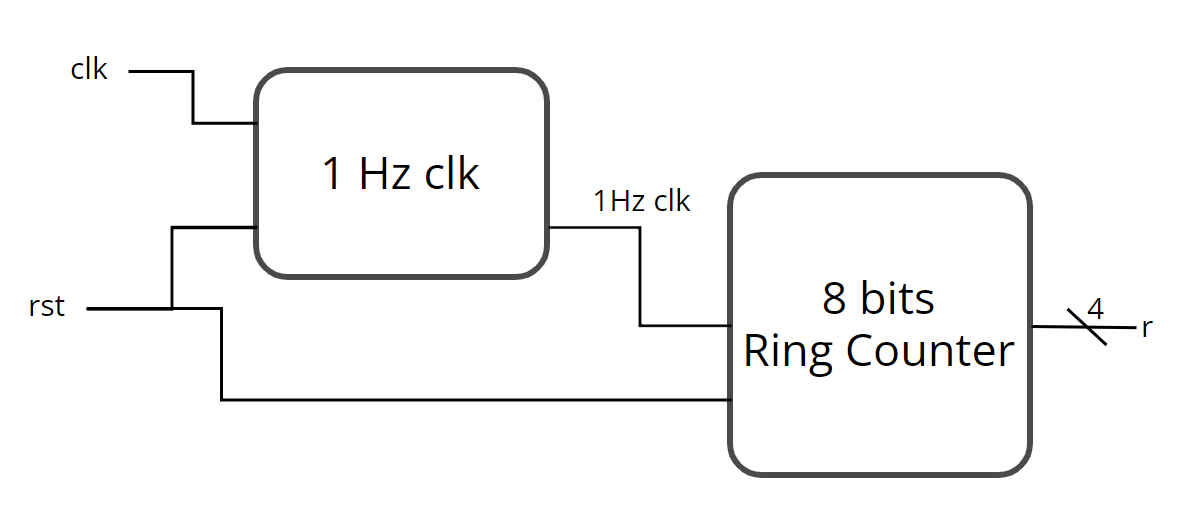
**Design Specification**

IO輸出入設定:

輸入: clk (1 bit), rst (1 bit)

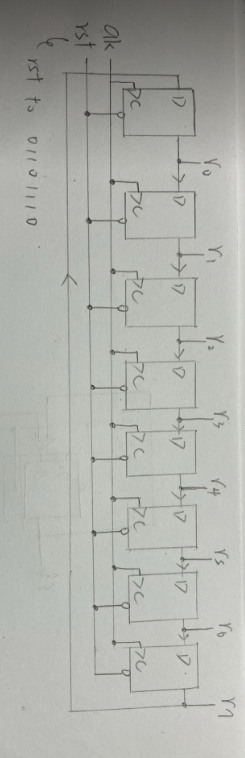
輸出: [8:0]r (8 bits)

Logic Block



**Design Implementation**

Logic Diagram



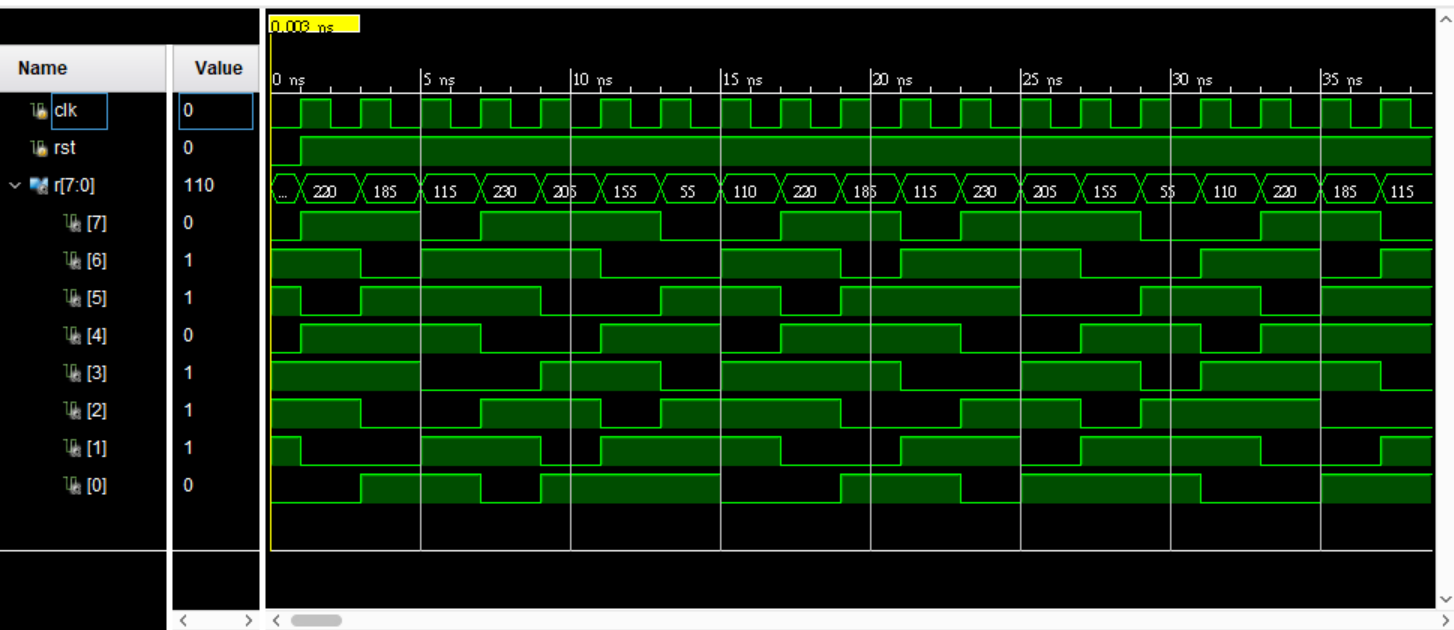
一樣把前面第二小題的1Hz時脈搬過來，只需要做ring counter。所以建立8 bits的r作為輸出，且每一位承接前面暫存器傳過來的數值，而最後一位傳到第一位，且傳送時皆是同時傳送。當rst’啟動時，就回到預設01101110。

Pin assignment

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| IO | clk | rst | r0 | r1 | r2 | r3 | r4 | r5 | r6 | r7 |
| Pin | W5 | V17 | U16 | E19 | U19 | V19 | W18 | U15 | U14 | V14 |

PS: rst設定: 01101110

**Discussion**



這張是8bits Ring Counter的testbench，時間一樣是自己設定的。

從波形圖就可以很確定Ring Counter是沒有寫錯的。這一題我覺得跟上一題很像，只是把4 bits up Counter改成 Ring Counter。整體上除了Ring Counter在傳送的細節不要寫錯，大致上沒有問題。時脈clk大致上跟前面一樣直接套用第二小題的結果使用。

寫成top module的形式最大的好處就是只要變數在模塊間的設定沒錯，clk時域的設定就不用考慮太多。只是這次發現在實作時呼叫模塊，呼叫的名稱記得要不一樣，例如: U0, U1…。在打程式時沒注意到，debug很久才發現這樣的問題。

**Conclusion**

在這一小題裡，我學到了如何寫Ring Counter。整體上的概念跟上學期學到的一模一樣。我覺得核心跟上一題一樣，只要top module寫得好基本上實做起來就沒什麼難度。

**References**

無

5 Use the idea from 4. 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 “NTHUEE2023” with the four seven-segment displays.

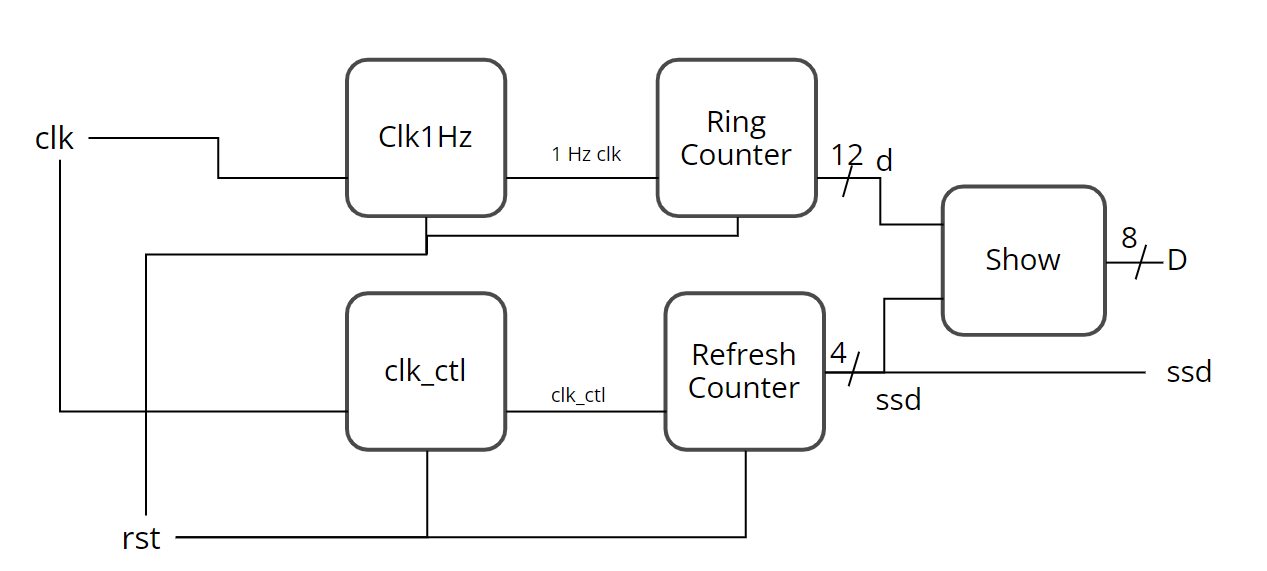
**Design Specification**

IO輸出入設定:

輸入: clk (1 bit), rst (1 bit)

輸出: [3:0]ssd( 4 bits), [7:0] D(8 bits)

Logic Block



**Design Implementation**

Clk1Hz跟第二題一樣輸出1Hz 的時脈。而clk\_ctl是用於控制七段顯示器的輸出頻率，實脈控制在100M\*2^-19左右，剛好可以在人的視覺暫留的範圍內。

RingCounter的功用是用於在顯示器上依序顯示NTHUEE2023，其中共有10位，所以它是一個10 bits Ring Counter。且N, T, H, U, E, 0, 2, 3共8種輸出，所以我用3bits來依上面順序編號他們。所以嚴格上這個Ring Counter是30bits，但每次是3 bits在做傳送。接著再把10位中的前四位字元共12bits設為d，作為當前應顯示的字元。

而Refresh Counter是用於在四個七段顯示器上面顯示不同字元。所以連的時脈是較快的clk\_ctl。輸出ssd其依序是0111🡪1011🡪1101🡪1110，共4 bits的 Ring Counter，控制哪時候四個燈哪個燈亮。

最後Show是針對d進行調整，前面講過d裡面的字元是3bits代表，所以Show一個任務是將d的3bits轉為七段顯示器的D 8bits。另一個是決定在ssd為哪個燈號時應亮哪個字串。

**Discussion**

在Ring Counter裡(控制字元輸出的)，目前是先用3 bits在轉換成8bits。一開始我是想跳過轉換的步驟，直接設為七段顯示器的值，同時我又想一次進行傳送，不想一個一個慢慢打3bits的傳送，所以選了二維陣列。但我發現二維陣列在always進行賦值或其他操作時會出現錯誤。由於不太瞭解其機制，所以我只好重新慢慢打。接著是顯示四個不同字元的七段顯示器。我參考了討論區的文章，並藉此得到啟發，瞭解如何實際處裡這樣的問題。最後一樣將各個模塊組合，寫成top module的形式。雖然有在yt上看到別人只寫2個時脈的模塊後直接在top module寫scrolling，但我為了保險起見，我還是統一處裡。

**Conclusion**

在這一小題，我學到了如何一次顯示四種不同字元的方法。但我覺得我目前使用的時脈仍有瑕疵，可能可以再clk\_ctl調高頻率來使七段顯示器看起來更自然。同時我也更熟悉使用七段顯示器和Ring Counter這兩個寫法。

**References**

<https://www.fpga4student.com/2017/09/seven-segment-led-display-controller-basys3-fpga.html>

這個是討論區有同學分享的，非常實用。唯一我個人的小缺點是他把所有東西寫在一起，有點複雜。