Lab 3 report

組員：110062221李品萱

110062213唐翊雯

1. **4-bit Ping-Pong Counter**

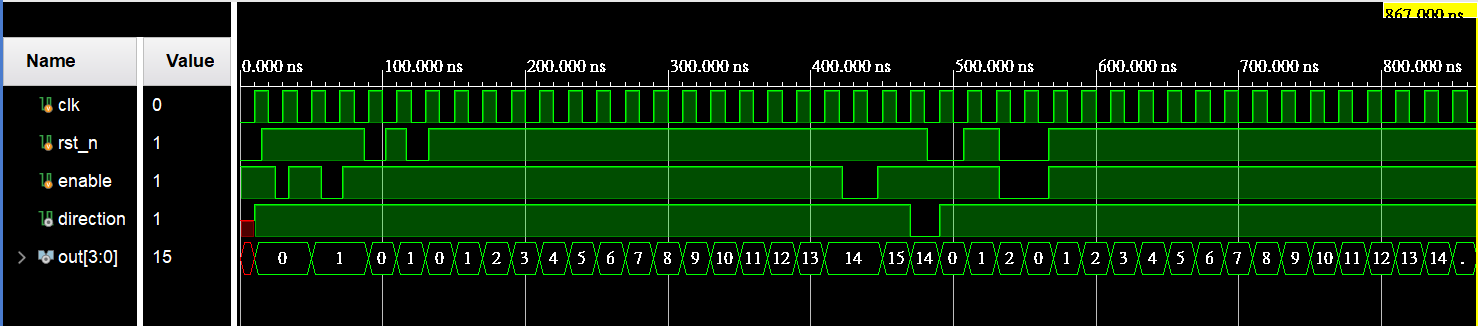
如下圖，這題我們需要實作一個Ping-Pong Counter。

一張含有 文字 的圖片

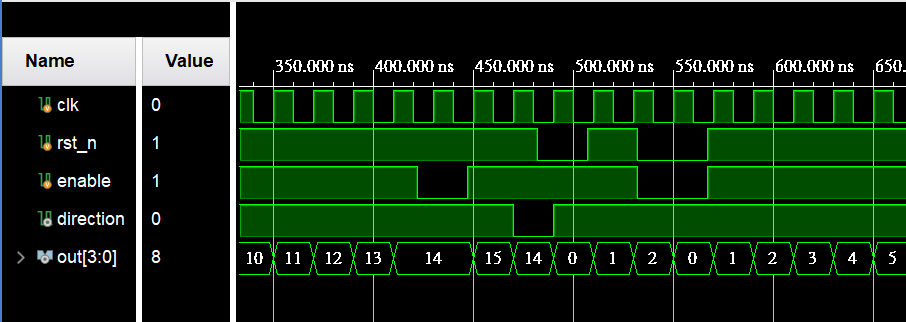
自動產生的描述

首先在combinational block的部份我們先用next\_out及next\_direction接下一次clock起來的時候要傳過Flip Flop的值。為了讓這個Counter counting up時能做到Spec中數到15後將direction從1變為0及counting down數到0後將direction從0變為1，我們觀察到只有Counting up的out在0到14的區間內或Counting down的out為0時，會使next\_direction = 1’b1，next\_out = out+1’b1，因此我們用if-else去做判斷，而它synthesize的結果即為Mux。Mux由左而右是priority由低至高，最左邊的Mux如上所述，而接下來這個Mux則會藉由enable判斷是否要hold原值或繼續傳next\_out及next\_direction；rst\_n則會判斷是否要將next\_dirextion設為1及next\_out設為0。在以上的combinational block做完之後，在每次clock起來時這個值就會通過Flip Flop，出來的值即為out與direction。簡單來說，combinational block會負責處理各個條件下next\_direction及next\_out的值，而sequential block則單純負責配合clock處理最後的out及direction。

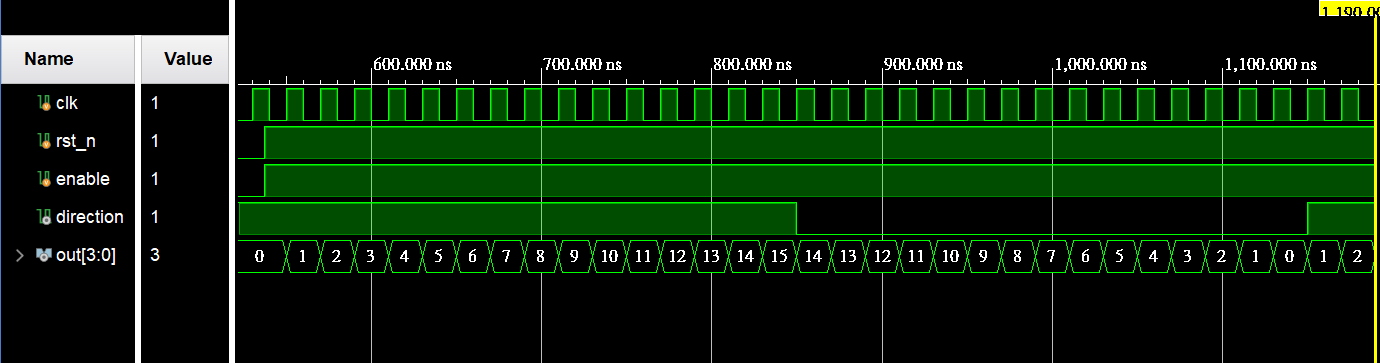
在下面的波型圖中我們可以看到，在clk起來時才會trigger到rst\_n為0，在此之前都是Unknow的狀態，並將我們的counter設定初始值，在out為1時會造成hold value的情形是因為我們的(rst\_n, enable) = (1, 0)的結果。



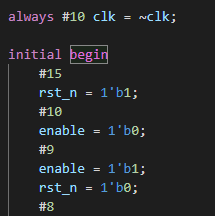
接下來我們在Counter運算過程中進行測試(rst\_n, enable)為(1, 0), (0, 1), (0, 0)的情況，如下圖。在out為14後經過一個clock，enable為0會被trigger到，此時如預期的會使counter hold value，而後由於已經數到15，direction會變為0並開始往下數；接下來我們讓rst\_n為0，clock起來時Counter也確實使out回到0，direction回到1；至於(0, 0)的情況則是由於rst\_n已經是0，且它的優先權是最高的，因此便會如waveform顯示的直接reset。



下圖我們測試從0數到15再數回0的狀況，確認我們設計的ping-pong counter可以正確地執行count up, count down及轉換direction的動作。



testbench的設計我們每10個單位時間（配合timescale即為10ns）會將clock反向，藉此產生clock的週期變化，其他input主要則是使用延遲一段時間去raise不同信號確認以上所述的各種情況，部分如下圖。



1. **First-In First Out (FIFO) Queue**

這題要設計一個 FIFO 的環狀 Queue，支援三種操作：寫入新的元素、讀取並pop掉最早寫進的值、初始化整個queue。我們另外開了三個reg：head、rear、ct，來記錄這個queue的資訊。head會記錄當前最早寫進來的元素在queue中的位置；rear會記錄下一個元素被寫進來時應該要儲存的位置，意即當前queue中最後一個元素的下一個位置；ct則是當前queue中的元素數量。

對於寫入新的元素這個操作，我們的實作方法如下：首先檢查queue是否已滿，如果，代表元素的數量恰等於queue的size，這時將無法再寫入元素，須將error設為1’b1。否則代表queue還未滿，這時我們先將要放進來的元素存在queue[rear]的位置，再更新rear，讓，維持rear的定義（下一個元素被寫進來時應該要儲存的位置）。最後將error設為0、並更新ct，讓（queue中元素的數量增加1）。

而對於讀取並pop掉最早寫進的值這個操作，我們的做法如下：首先檢查queue是否為空，如果，代表元素的數量為0，這時將無法讀取元素，須將error設為1’b1。否則代表queue非空，這時我們先將queue[head] 的值傳給dout，再更新head，讓，維持head的定義（當前最早寫進來的元素在queue中的位置）。最後將error設為0、並更新ct，讓（queue中元素的數量減少1）。

對於初始化整個queue這個操作，我們將head、rear與ct三者的值都設為0，即完成清空queue、對其初始化的工作。

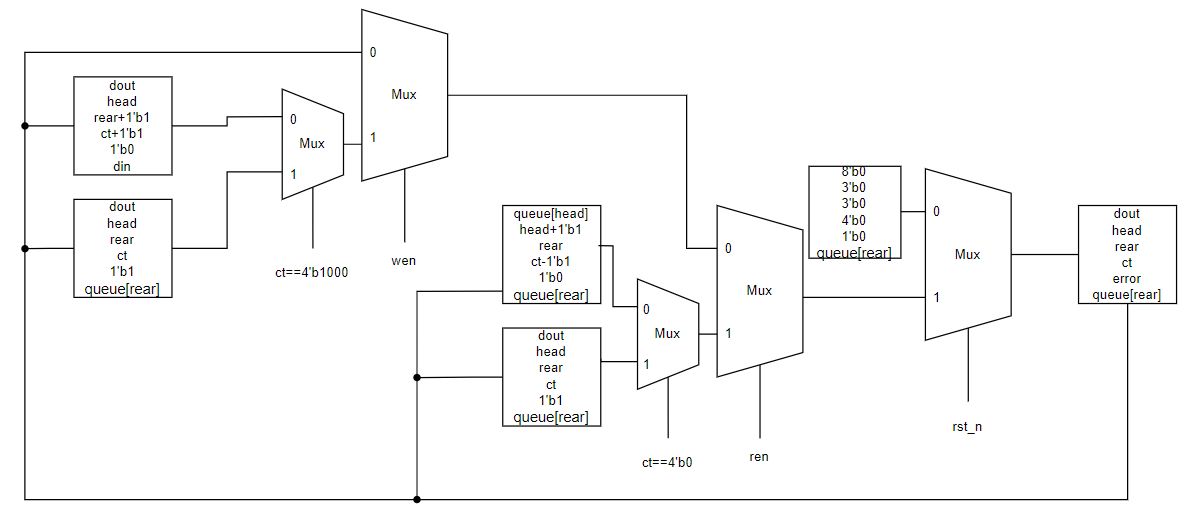
現在，根據題目給的input，總共會有三種情況，分別對應到這三種不同的操作：

在這個條件下，根據題目要求，我們執行上述提到初始化queue的操作，並且將dout與error都設為0。

在這個條件下，根據題目，我們執行上述提到「讀取並pop掉最早寫進的值」的操作。

在這個條件下，根據題目，我們執行上述提到「寫入新的元素」的操作。

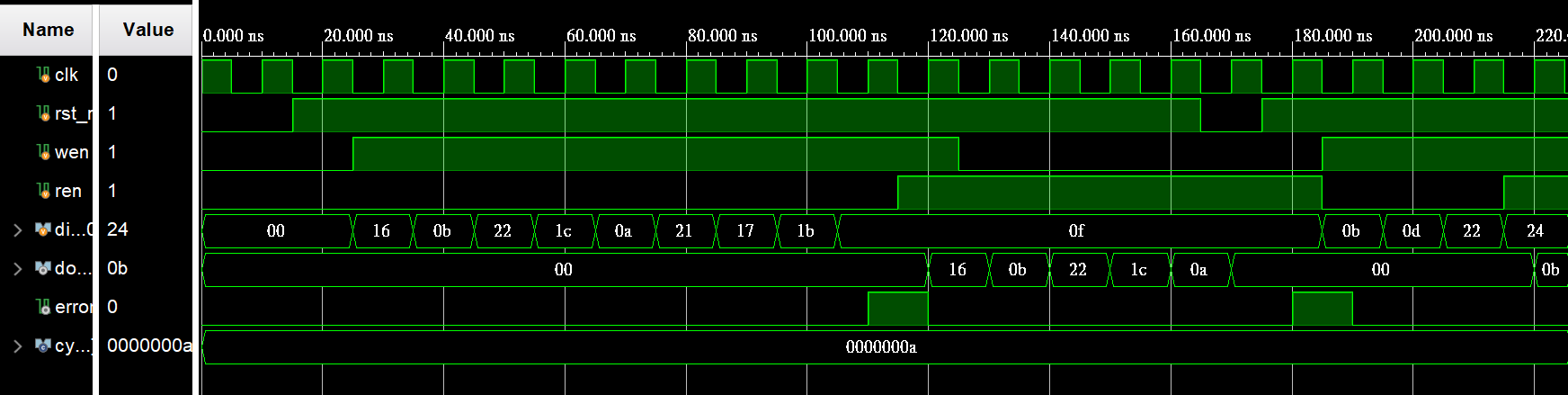
上述的所有實作我們都放在always block裡面並用posedge clk triggered，並且使用non-blocking assignment的方法來完成，block diagram如下：



Testbench的部分，除了最基本的讀取與寫入外，主要要測試的重點有下列幾項：

* reset後是否有確實將queue清空
* queue為空但仍收到讀取指令時，是否做好error handle
* queue已滿但仍收到寫入指令時，是否做好error handle
* ren與wen同時為1’b0時，是否執行read operation

因此我們在設計testbench時，調整參數使以上四種情況都出現，以檢查code的正確性：在將rst\_n設為1後，我們先write進9個數字，確認queue已滿但仍收到寫入指令時，error會被設為1且queue中的元素不會被更動。接著我們做讀取的動作，確認讀取的順序遵循FIFO。接下來我們做reset，將rst\_n設為0再設為1並立即做讀取，檢查reset後是否有確實將queue清空以及queue為空但仍收到讀取指令時，是否將error確實設為1’b1並維持queue不被更動。最後再對queue做一些讀寫操作確認reset後的queue確實沒有問題，最後的waveform如下：



1. **Multi-Bank Memory**

這題要求我們使用basic question 2實作的module來做出Multi-Bank Memory。我們最一開始直接使用16個basic question 2的Memory module作為此題module的16個sub-bank，並將他們由4’b0000~4’b1111做編號，而並沒有另外分bank。然而，這樣做會造成讀取的index含有變數，無法好好切分sequential circuit與combinatial circuit。因此後來我們更改成先開4個bank的module，每一個bank的module裡面再使用4個basic question 2的Memory module作為sub-bank。

根據ren以及raddr的值，要決定是否做讀取操作以及要對哪一個bank做讀取。這裡分成兩個情況：

此情況下不需要做讀取操作，我們將renb設為4’b0，使所有的bank都不做讀取。

此情況下須對編號為raddr[10:9] 的bank做讀取操作，此時我們應將renb[raddr[10:9]] 設為1’b1，而renb其他位置的值設為1’b0。此項操作類似於decoder，我們可以利用位移運算來達成：

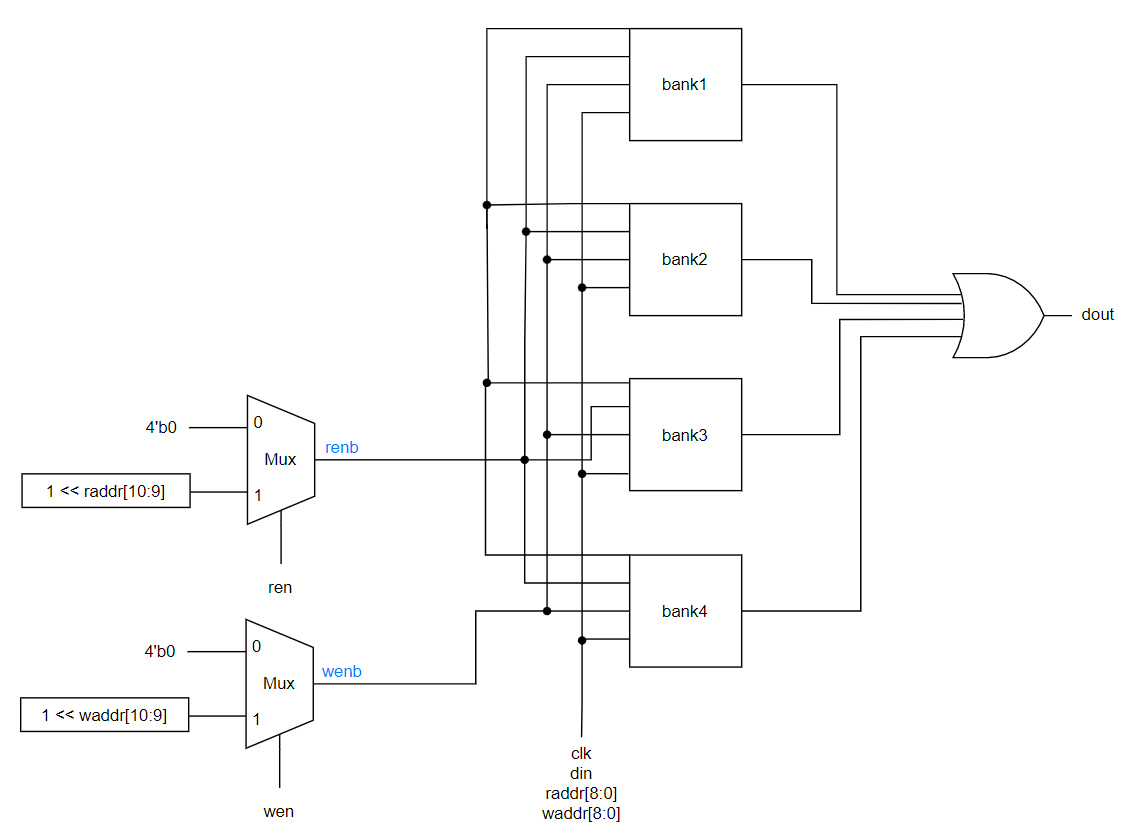
讓，即可達成前述要求。

與前一段類似，根據wen以及waddr的值，要決定是否做寫入操作以及要對哪一個bank做寫入。一樣分成兩個情況：

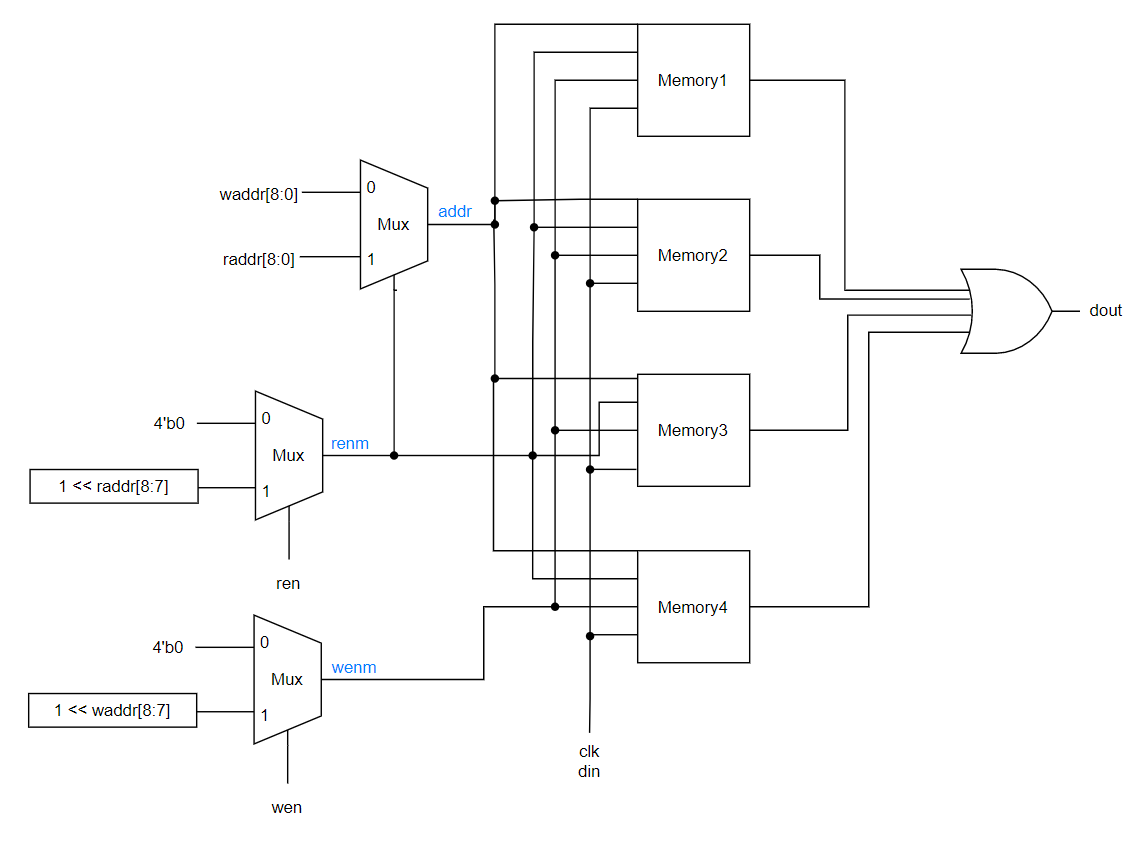
此情況下不需要做寫入操作，我們將wenb設為4’b0，使所有bank都不做寫入。

根據我們前面的定義，此情況下須對編號為waddr[10:9] 的bank做讀取操作，此時我們應將wenb[waddr[10:9]] 設為1’b1，而wenb其他位置的值設為1’b0。一樣可以利用位移運算來達成：讓，即可達成前述要求。

最後，dout的值即為所有bank的output取bitwise or。以上操作都是用combinatial circuit實作，block diagram如下：



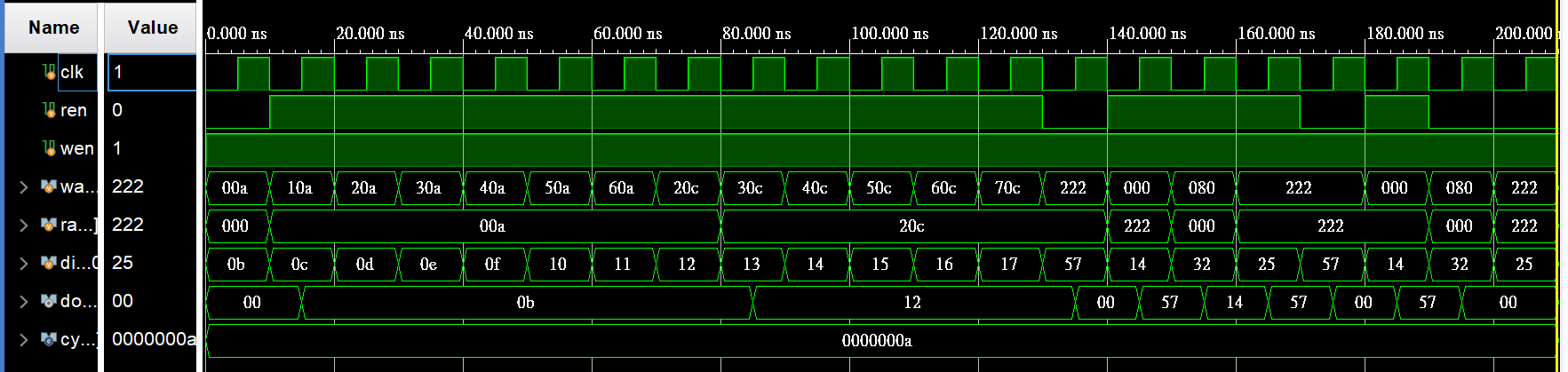
而每個bank裡面的實作與前述類似，差別在於對每個memory module而言，不再需要給raddr、waddr兩個位址，而是根據ren以及wen的值決定要傳哪一個address進去。由於讀取較寫入有較大的優先序，因此我們以ren作為判斷，若該memory的，我們便讓addr = raddr[6:0]，否則讓addr = waddr[6:0]。Block diagram如下：



testbench的部分，除了最基本的操作，我們還測試了各種情況，包含：

* 對同一個sub-bank同時進行讀取與寫入
* 在raddr==waddr的狀況下只寫入不讀取，確認是否寫入成功
* 對同一個位置重複寫入，確認其紀錄的值是否正確

而最終的波形圖如下：

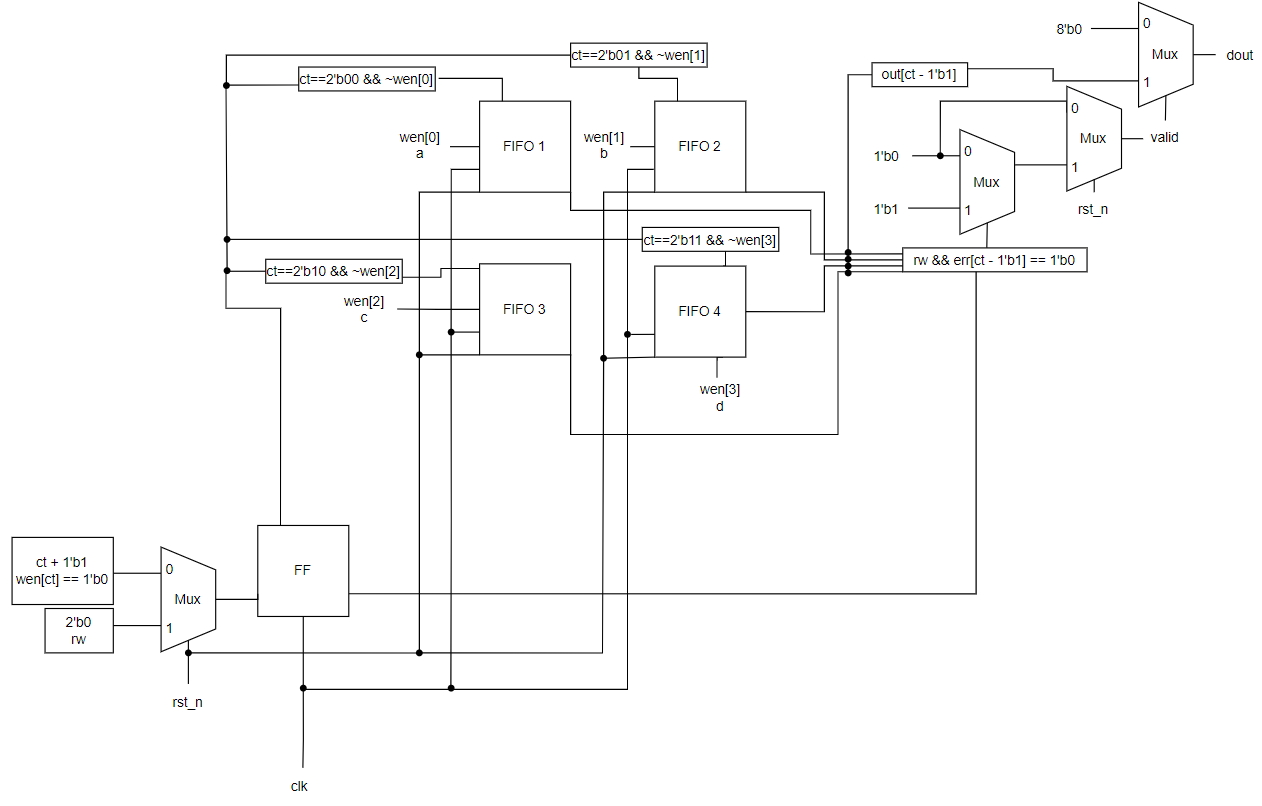


1. **Round-Robin FIFO Arbiter**

這題要沿用advance 2的module，實作Round-Robin FIFO Arbiter。根據題目要求，我們開了4個queue的module，並分別幫它們多開了wire接它們的output（dout與error），和多開了reg來作為這四個queue的ren。

我們另外開了一個變數ct，來記錄當前讀取到第幾個queue。我們將ct 宣告為2個bits的reg，這樣在它數完第3個queue（0-indexed）之後再加一，會恰好溢位並回到第0個queue，恰符合題目要求。而維護ct的方法如下：先判斷rst\_n是否為1’b0，如果是的話代表要reset，ct必須重新設為2’b0。否則的話，讓ct = ct + 1’b1，往下一個queue移動。此外，我們另外維護一個等一下會用到的reg，記錄前一刻是否有在正在讀取的那個queue中，做寫入的動作。我們將此變數命名為rw，它會與combinatial circuit中valid的值有關。這段操作我們使用sequential的寫法，寫在always block內，並用posedge clk triggered，讓ct能在clk變動時改值，達到題目的要求。

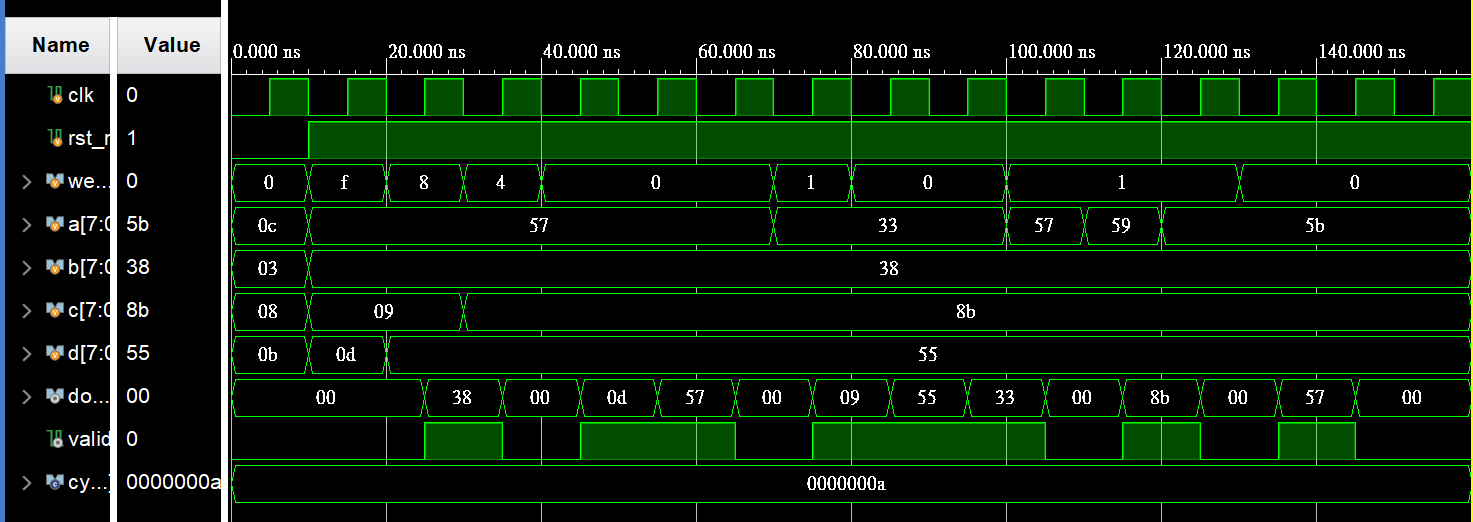
接著combinatial circuit的部分，我們寫了另一個always block，對ren、vaild以及dout作更新。我們將ren宣告成4個bits的reg，分別代表四個queue的ren。因為題目要求若同時對同一個queue進行讀取和寫入操作時，應只做寫入而不做讀取，因此這裡ren[i] 的值應該要設為ct == i && ~wen[i]。而valid的部分，主要與三件事有關：rst\_n是否為0、前一刻是否有寫入（有寫入便不能讀取），以及前一刻若有讀取是否讀取成功。這時判斷就需要用到在sequential circuit中更新的rw，以得知前一刻是否有寫入。因此考慮valid的值時，我們先考慮rst\_n是否為0，若為0則valid為1’b0。若rst\_n不為0，則valid的值為，這邊注意ct要減一，因為是判斷前一刻的讀取是否成功。最後dout的部分，如果valid為1’b0，則dout為8’b0，否則dout的值為out[ct – 1’b1]，一樣因為讀的是前一刻的值所以ct要減一，整體的block diagram如下：



Testbench的部分，主要需測試的重點有下列幾項：

* 要讀取的queue為空時，是否做好error handle
* 同時寫入與讀取queue時，是否只做寫入操作
* 讀取順序是否正確

我們讓以上這幾種狀況都在testbench中出現，以驗證code的正確性，waveform如下圖：



1. **4-bit Parameterized Ping-Pong Counter**

這一題與第一題非常相似，小差別在於多了flip、max、min等control signal，如下圖，block diagram的hold如下所示。

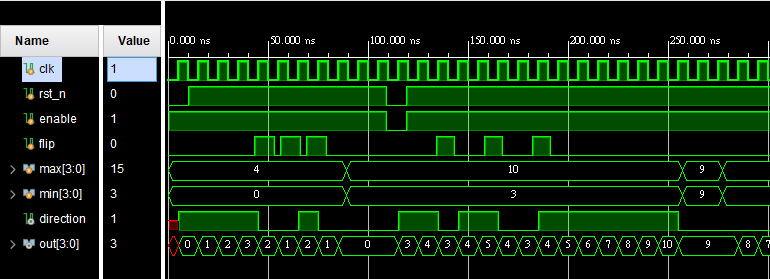


一張含有 文字 的圖片

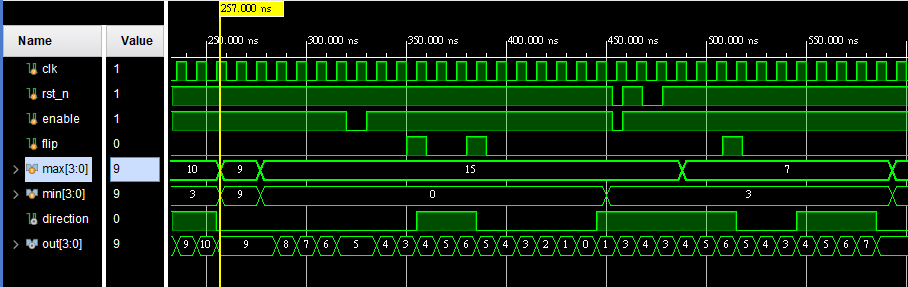
自動產生的描述

與第一題相同的Mux這邊就不贅述，我們可以先看以flip為reset signal的Mux，flip為1時會將direction反向，out數的方向也會與原本相反；而hold我們將這題新增的條件直接以assign的方式接起來去判斷是否要維持當前的值。經過途中這些Mux之後我們就會得到最後的值，等clock起來便會將值送出。

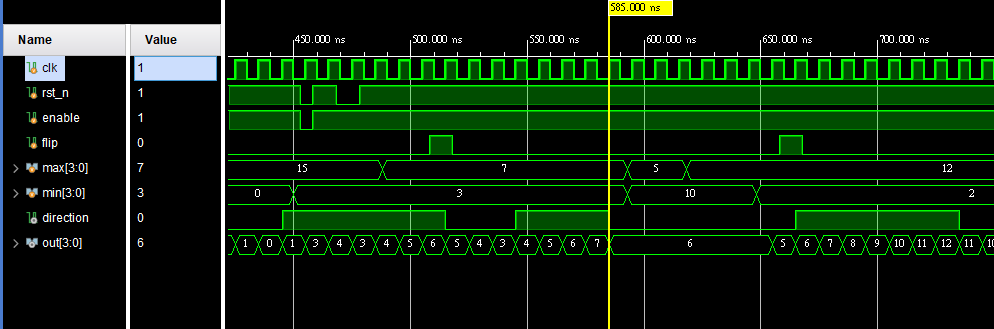
下面的波型圖中，我們先將(max, min)設為(4, 0)，做出與spec相同的waveform初步確認我們的設計應該是正確的，這邊我們也設計了連續flip的狀況，能看到direction及out都有做出相對的改變。接下來我們將(max, min)設為(3, 10)，剛trigger到新的值的時候，由於out<min，因此out會維持它原先的值一直到rst\_n為0時，direction為1，out被設為min，它才符合繼續count的條件。



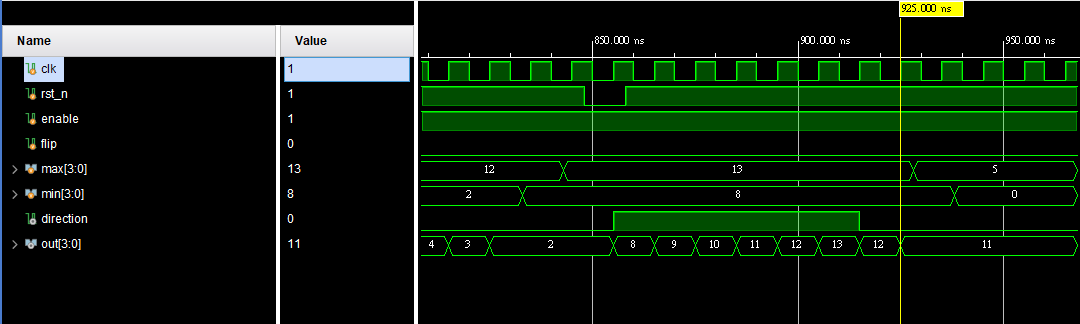
接下來我們測min==max==output的情況，counter此時會維持其原值(out and direction)，之後我們也繼續測flip及enable，皆符合預期。而由於前面我們都是同時改動max及min，這邊我們試了先改動min的case，也都有出現正常的結果。



接下來我們測max<min，此時counter會維持其當前值。



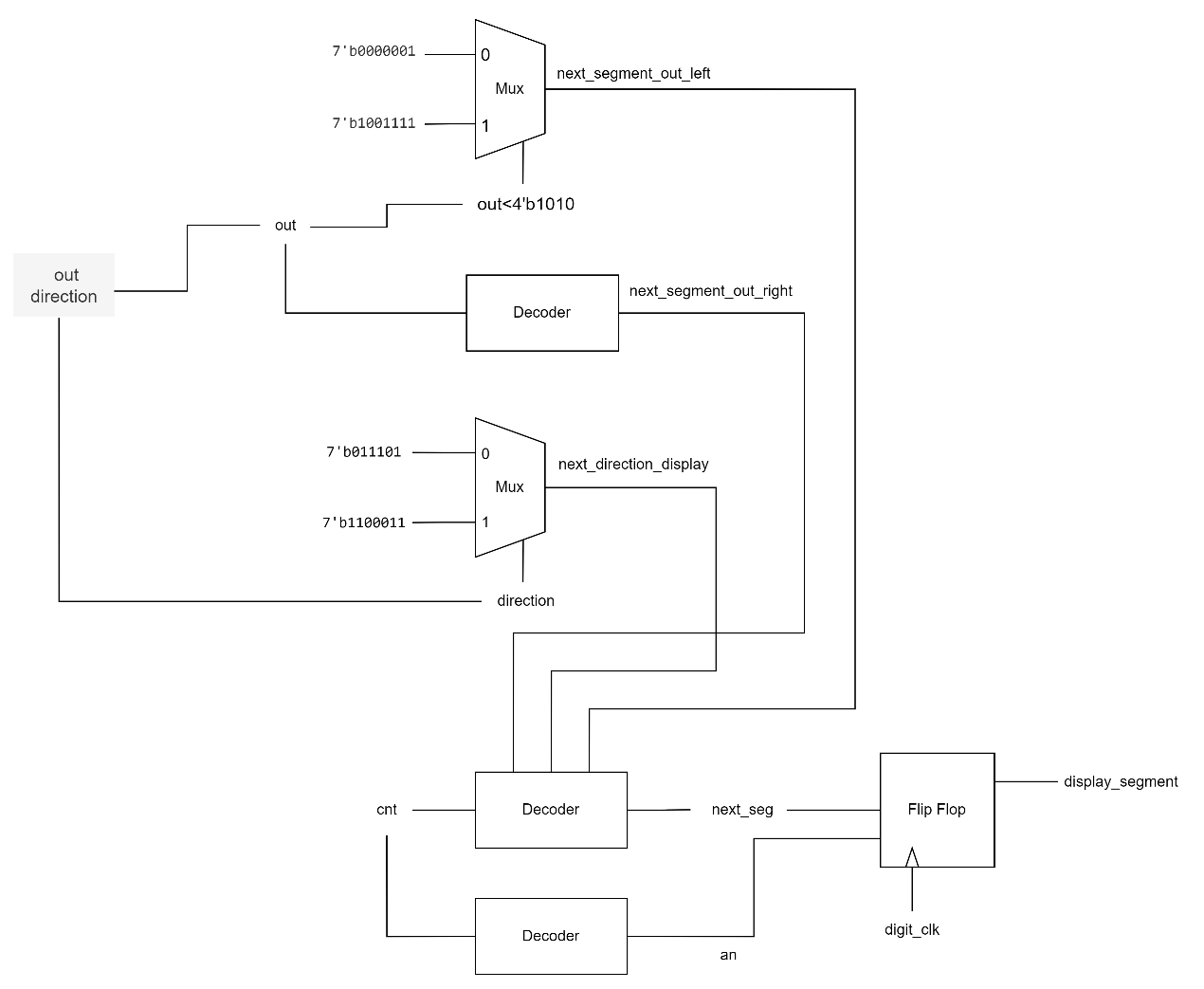
下圖中我們測out>max，此時counter亦會維持當前值。



這個testbench我們的目的是測試其是否符合spec的要求，因此除了檢查flip及enable是否能正常運作之外，我們主要對特殊狀況都進行測試，稍嫌冗長但至少是完整的。

1. **4-bit Parameterized Ping-Pong Counter on FPGA**

如下圖，這題我們將上一題實作的呈現於FPGA上，由於上一題已說明過counter的out及direction，這邊我們針對display的部分進行說明。首先，我們用Mux及Decoder為out對應到它要在seven segment顯示的值，Mux處理的是十位顯示的值及direction，Decoder則處理個位數的值。而後，如同上課提過的，seven segment一次只能顯示一個digit，因此我們用cnt決定它顯示的先後順序，而此處我們用的clock是除頻過的digit\_clock，在每個positive edge digit\_clock，我們的display\_segment就會收到next\_seg的值，即為它要顯示的值。



* **Clock divider**

由於板子給的clock變化很快，為了方便肉眼觀察，我們需要為clock做除頻，我們的作法是將除頻過的clock送進負責計算out及direction的module。除頻的部分上課時教授給的建議值是clk，但我們在操作時發現如果要同時滿足觀察跟reset及flip bottom的話，clk會較為剛好，如下圖。我們運用的是在basic lab做過的Clock Divider加以衍伸，用cnt數我們要在第幾個clock時將對應的divided\_clk做更動。

一張含有 文字 的圖片

自動產生的描述

後來在與另一班同學討論到除頻時，發現我們使用了不同的除頻方式，因此以下我們也用這個方法做出了相應的設計，code如下圖。與前一種做法不同的是，這邊我們直接指定divided\_clk = cnt[23]，這個作法的原理是，假設以一個3bits的count來說，每次加一結果如下。

* 000 -> 001 -> 010 -> 011 -> 100 -> 101 -> 110 -> 111

我們可以觀察到count [0]每次都會反向，count [1]每兩次反向，count [2]每四次反向，若配合clock則會使count [0]產生週期為clk的方波，count [1]產生週期為clk的方波，count [2]產生週期為clk的方波，由此可知count[n]會產生週期為 clk的方波，因此cnt[23]即為週期clk的方波，digit\_clk = cnt\_1[15]也是同樣原理。

一張含有 文字 的圖片

自動產生的描述

這兩個做法會產生不同的clock waveform，第一種作法是做出clock high的寬度與原本的clock signal相同的clock，第二種作法則會產生類似放大版的clock信號，區間為1與區間為0的寬度相同；但這兩種做法都能產生正確的結果，因為我們只要divide過的clock為high的瞬間，所以只要這個瞬間是正確的，就能使後續觸發的信號正常運作。

* **Debounce and One pulse**

另外，我們還需要對bottom做debounce及one pulse，這邊上課時都有講解過，所以可以直接實做出來。One pulse Circuit首先會將debounce過的結果做延遲一個clock及反向，而後將其與debounce的結果and起來送入DFF，即會得到結果。這麼做的原因在於bottom可能被按著不只一個clk，但我們只想要按下去時產生一個clk的變化。在design時需要注意的是， onepulse module用的clock必須是除頻過後的clock，因為我們這邊onepulse出來的結果必須跟控制display的clock一樣寬，否則可能會trigger不到這個信號，這也是為什麼我們需要配合bottom去調整除頻的數值。Debounce的部分則是為了去除push bottom產生的glitches，送出乾淨的信號，因此我們用四個DFF過濾掉它，即可完成debounce。

這題我們沿用上一題的module並稍作修改，如：在push reset bottom時它會給出1，但我們要的是reset為0時reset，因此我們先做not再送進上一題的module。對於這題我們分不同module處理debounce、one pulse、clock divider、計算out and direction及FPGA display，在display的部分則寫好7’b對應亮哪個Pin再將其接好，即可display 4-bit Parameterized Ping-Pong Counter。

1. **Summary**

這次的Lab我們接觸了Clock divider、One pulse、debounce等新的東西，也學到如何養成好的coding style。Basic Lab的時候還不太清楚自己在幹嘛，只是按照spec的指示給條件，在進一步寫到advance時才比較有概念，但由於接觸的不多，這次我們在實作過程也出現比前幾次都要多的問題，除了需要更仔細的思考combinational 與sequential如何符合我們期待的設計之外，也遇到vivado不同的報錯及警告並藉由網路資料解決它，這個過程也使我們去思考自己寫出來的code實際上會接出怎樣的電路。

在Ping-Pong Counter的部分，我們一開始的寫法是將rst\_n及enable放在sequential block內，但會出現enable要hold值時，我們不太確定是否可以寫out <= out這種code，或者直接將else後面空白，但也不確定這樣會產生什麼樣的電路；詢問助教後，我們得到直接將它們放在combinational block裡面用next值來接的建議，真的大夢初醒，這兩個其實跟其他控制next\_out，next\_direction的信號是同樣的概念，因此後來我們的sequential block就只有短短四行，只對out及direction做傳遞值的動作，非常感謝助教。

在FIFO Queue的部分，我們學到如何實作一個circular queue。實作過程中需小心的思考如何取值刪值、何時移動head，何時移動rear、如何知道queue為空以及為滿等等，思考並學習到許多實作上的細節。

而Multi-Bank Memory 以及Round-Robin FIFO Arbiter的部分，最難的是要思考如何好好切割combinatial circuit以及sequential circuit。如果沒有處理好的話，會導致clk延遲、在非posedge的時後改值等問題。雖然此時output看起來大致正確，修正起來卻是浩大的工程，這也提醒了我們先畫圖的重要性。

FPGA的部分則讓我們進一步思考clock在整個過程的運作，也加入了push bottom帶來的新觀念，一開始我們的bottom在one pulse沒有正確的設定clock因此一直沒有反應，成功解決後對此也有了更好的理解。

從gate-level到這次的modeling techniques，我們對整個電路有更完整的了解，這次主要是要對combinational及sequential有較為清楚的概念及養成良好的coding習慣，我們也確實在這方面有所收穫，不過或許是慢慢接近期中了，因此近期綜合結果如下圖。



1. **Contributions**
2. **4-bit Ping-Pong Counter** By 唐翊雯
3. **First-In First Out (FIFO) Queue** By 李品萱
4. **Multi-Bank Memory** By 李品萱
5. **Round-Robin FIFO Arbiter** By 李品萱
6. **4-bit Parameterized Ping-Pong Counter** By 唐翊雯
7. **4-bit Parameterized Ping-Pong Counter on FPGA** By 唐翊雯