

第七組

黎佑廷 105030009

郭家瑋 105030015

lab3

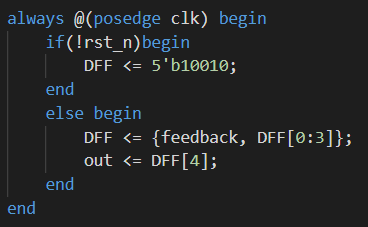
Verilog Question 1:

Linear-Feedback Shift Register (LFSR)

DFF[1] 與DFF[4]進行XOR運算後會回授到DFF[0]上，我稱這個回授為feedback，它所對應的code如下:



如果reset訊號是0的話，我將DFF[0:4] 設為5’b10010。此外，在posedge clk時將feedback訊號傳給DFF[0]，DFF[0:3]傳給DFF[1:4]，它們所對應的code如下:



這題我所使用的驗證方法是利用simulation的波形圖去看訊號有沒有順利從DFF[0]傳到DFF[4]。

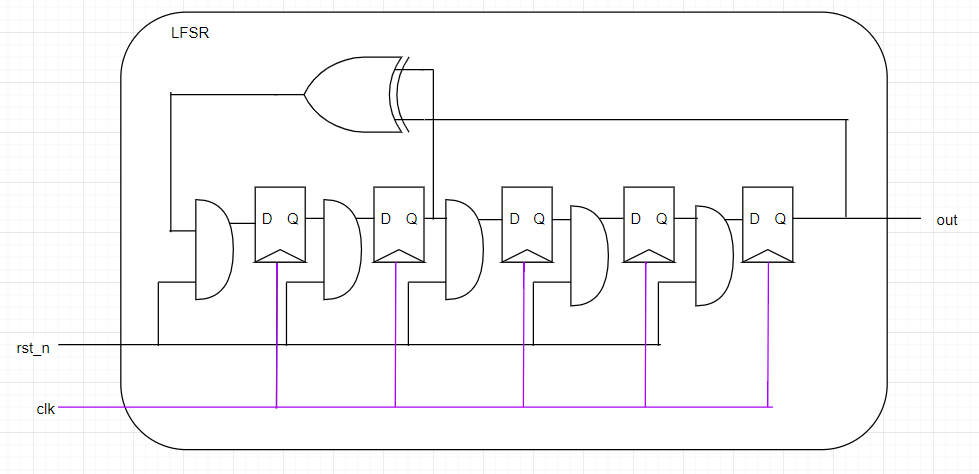
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| DFF[0] | DFF[1] | DFF[2] | DFF[3] | DFF[4] | output |
| 1 | 0 | 0 | 1 | 0 | x |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 |

由上面的表格可以知道output會是x0100100001

對照波形圖可以發現結果(out)跟我們預測的一樣:



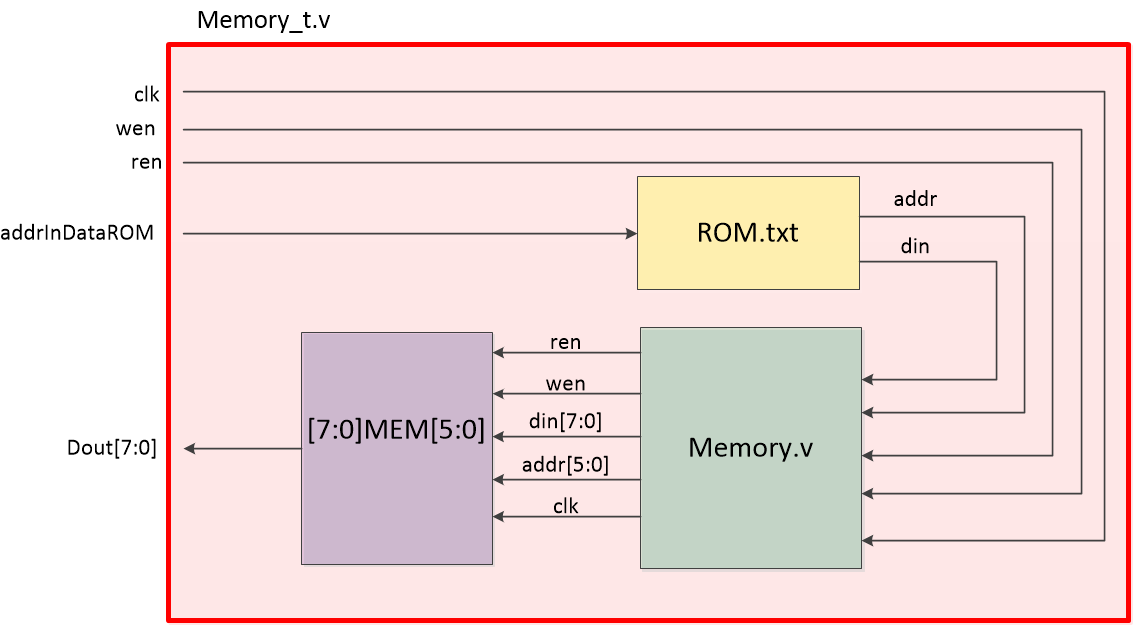
Block Diagram:



Verilog Question 2

64 x 8 memory array MEM

* **Block diagram**



* 說明：

在Memory\_t.v裡會產生clk與wen、ren、addrInDataROM的訊號。讓addrInDataROM從第一筆依序跑到最後一筆資料，以從ROM.txt裡取出addr與din的訊號。在ROM.txt裡面，同一筆資料(同一個時間點吐出的addr與din)我分成兩行寫，第一行是addr，第二行是din。換言之，

always @ (negedge clk) begin

If ((addrInDataROM \* 2 + 1) <= maxSizeInDataROM) begin

addr = dataROM [2 \* addrInDataROM];

din = dataROM [2 \* addrInDataROM+1’b1];

addrInDataROM = (addrInDataROM+1’b1);

end

end

於是Memory.v取得clk、wen、ren、addr、din的資料送給MEM的二維陣列。當ren=1’b1且wen=1’b0代表有資料要從Memory寫入MEM，此時讓Memory.v送出要寫的數值din與要寫入的位置，並讓dout輸出為8’b0。

當ren=1’b0，代表想要從MEM裡面讀資料，因此Memory.v送出想要讀的位置給MEM，MEM送出數值給dout。

當兩種情況都不符合，讓dout輸出8’d0。

換言之，

always@(\*) begin

if (!ren) begin

dout=Mem[addrp];

end

else if ((!wen)&&ren) begin

Mem[addrp] = din;

Dout = 8’b0;

end

else begin

dout = 8’b0;

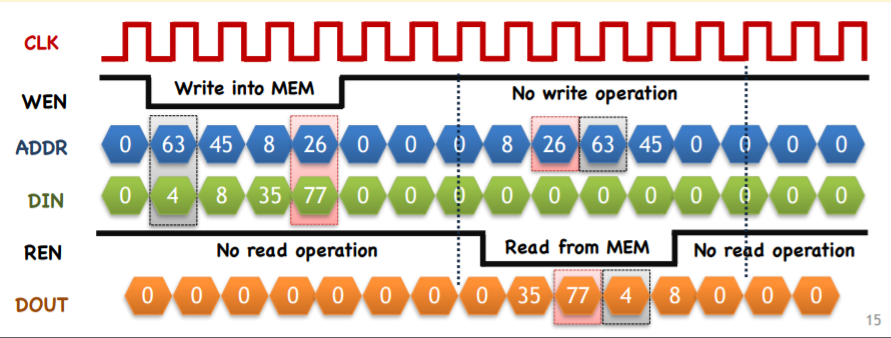
end

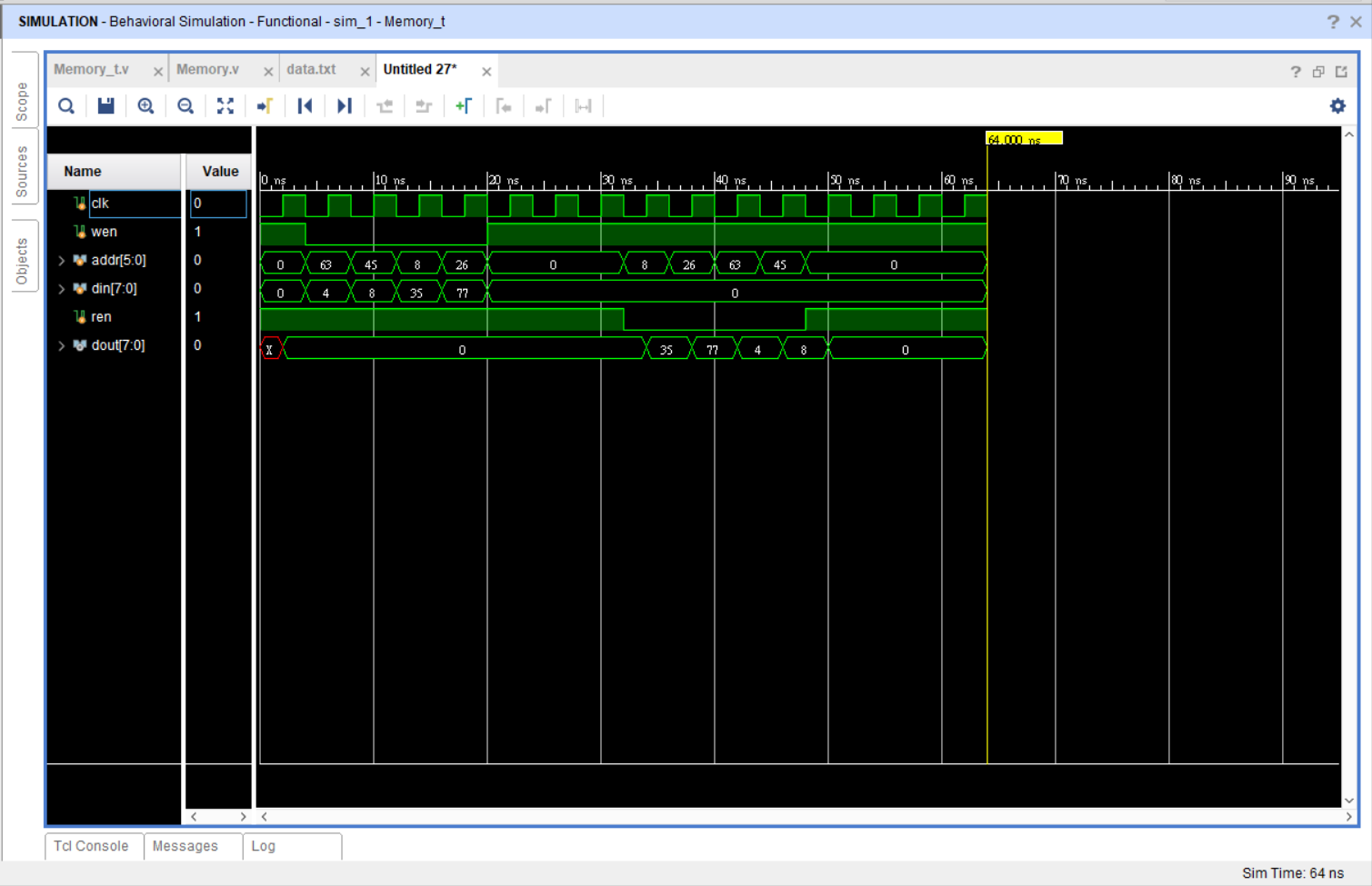
end

在testbench是negedge觸發，在.v檔是posedge觸發，以使.v檔讀到的資料已經是已達到穩定狀態的數值。

* **Testbench**

Testbench因為想要從簡，所以並沒有另外再造一個ROM與wen、ren的測資，而是照著老師ppt波形圖的方式，在對的時間把wen、ren升起來，ROM.txt的資料也是老師的addr跟din。於是我比較我的dout、波形圖與老師的，確認相同。

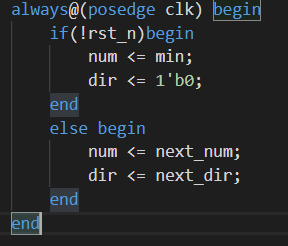




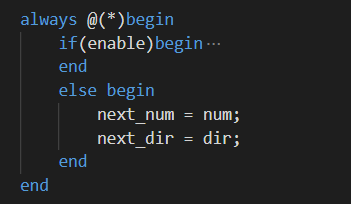
Verilog Question 3

Parameterized Ping Pong Counter

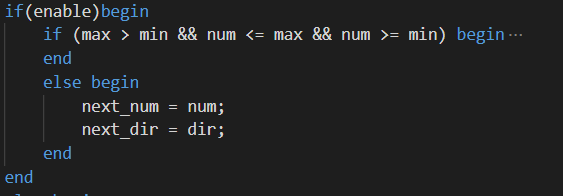
如果rst\_n == 0，按照spec要求，我將num設成min，dir 設成 0，以便它從min開始往上數, code如下(next\_num以及next\_dir在combinational block有進一步的運算):



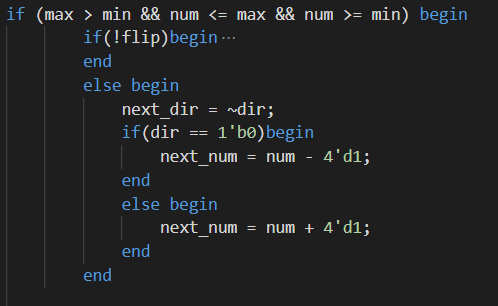
至於combinational block，讓我們從enable看起，如果enable == 0的話我們就不改變num以及dir，如下圖:



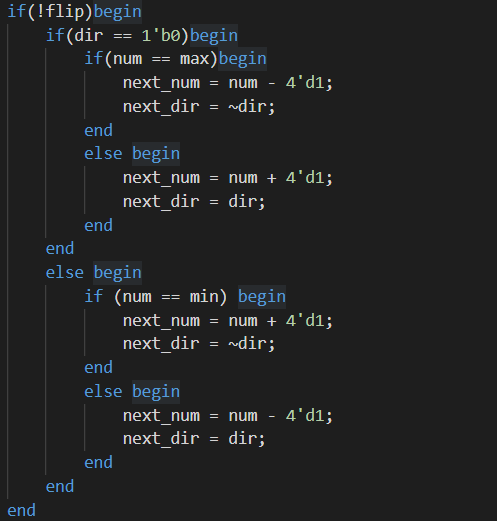
如果enable = 1，第一步是先進行max與min的相關檢查，如果檢查不通過就不改變num以及dir的值，如下圖:



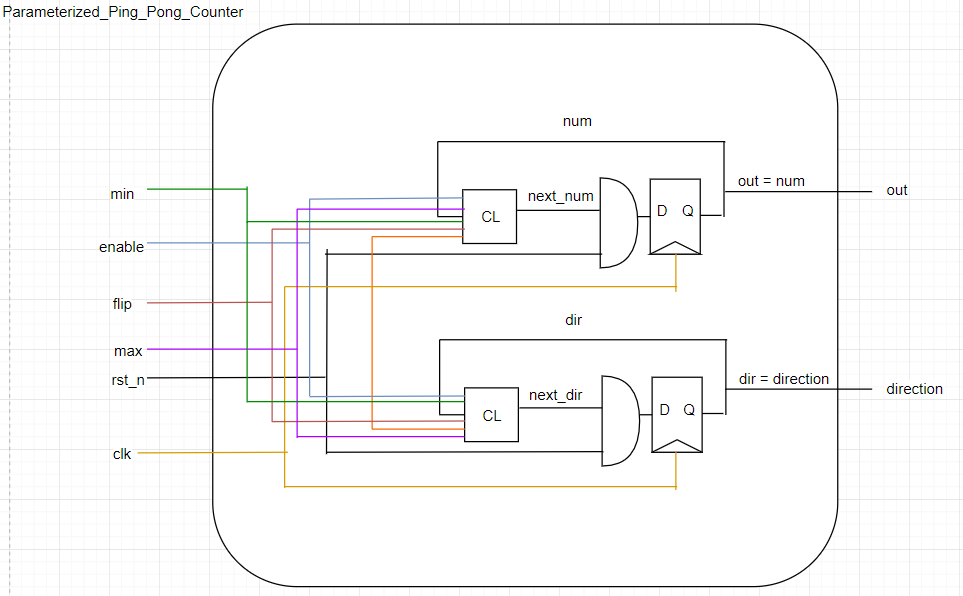
如果max以及min的相關檢查通過，我的下一步是檢查是否有flip訊號，如果有，那就將dir變換方向，此外，如果flip == 1的時候dir是0那麼下一個num應該要減一，如果dir是1那麼下一個num就加一，如下圖:



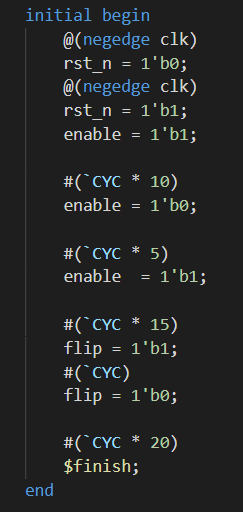
如果沒有偵測到flip訊號，那麼原本dir是0的話就繼續加1直到num == max時進行反向操作，dir是1的話就繼續減1直到num == min時進行反向操作，如下圖:

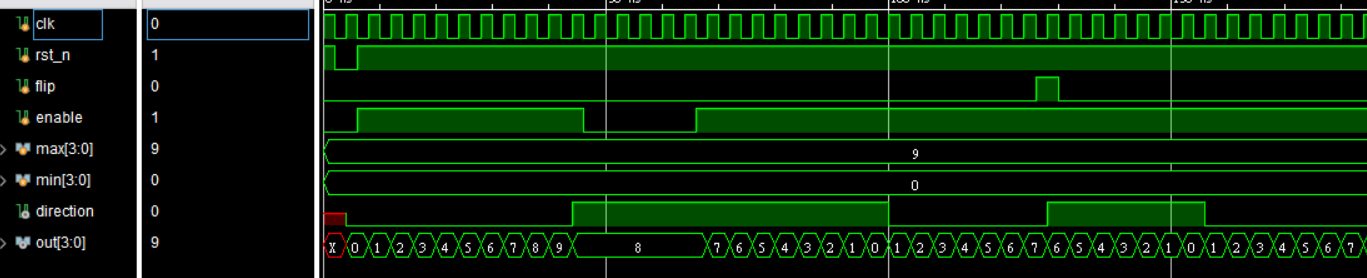


Block Diagram:



Testbench的寫法是在不同的時間給予不一樣的input，觀看波形圖的正確與否，如下圖:





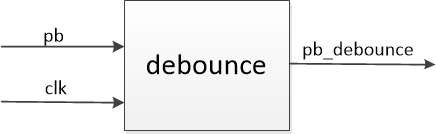
Verilog Question 4

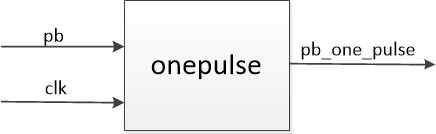
FPGA

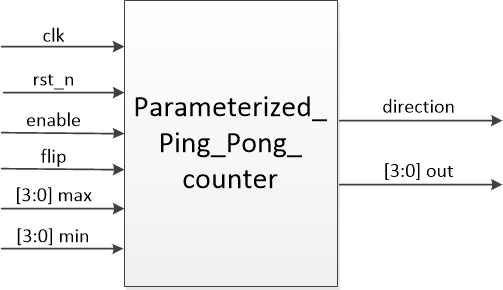
* **Block diagram**

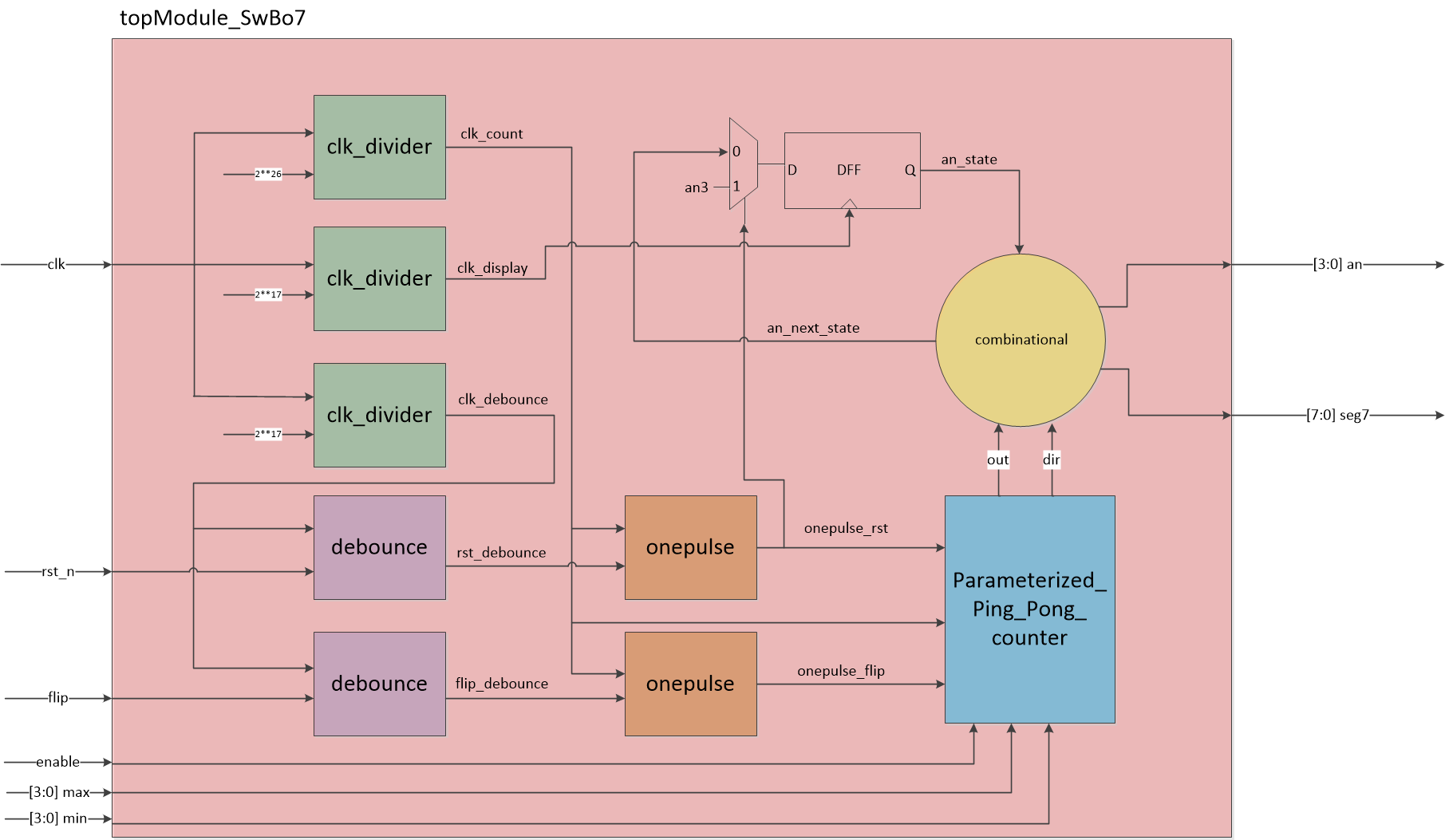
在top module：topModule\_SwBo7中包含了以下小module，為了簡化整體的block diagram，先放上小module本身的port，在整體的block diagram就省略小module的port。











1

4

2

3

* 說明：

先講clock的設定。在Basys 3 的內建clock是100MHz，也就是週期是10ns。在不同的情況下不同module會需要用不同clock 週期才能是整體正常運作，於是我用clock divider做出三個不同週期的clock

clk\_count：用於counter數數字的快慢，就是在FPGA上數字跑動的快慢。我將內建clock週期乘以2^25，週期約0.3秒。

clk\_display：用於讓四個七段顯示器分別通電的時間週期。我將內建clock乘以2^16，週期約0.3ms。若四個七段顯示器通電時間太短則來不及通電，無法顯示正確的數字，若太長則無法造成明顯的視覺暫留。

clk\_onepulse：因為手指按下按鈕的時間可能會稍長，造成按鈕訊號對後面的邏輯可能會有大於1個cycle的觸發，造成像是rst按了兩次獲flip按了兩次，但實際上只想要一次。照著講義上的方法，運用一個DFF延後訊號，並原始訊號與延後訊號做一些邏輯處裡即可以產生只有一個cycle的訊號，即onepulse\_rst、onepulse\_flip。這裡的DFF 的clk我使用clk\_count，約為0.3秒。使用clk\_count的原因是，若clk週期大於count，產生的onepulse有可能覆蓋到兩個clk\_count，就是只壓下按鈕一次卻flip兩次；若clk週期小於count，有可能產生的onepulse沒辦法覆蓋到clk\_count的posedge，造成沒有flip反應。因此若壓下按鈕的時間短於0.3秒有可能不在posedge 被讀入訊號。按壓按鈕按0.3秒以上可以確定會被讀入訊號。

clk設定好後，處裡按鈕的去除雜訊，按鈕包括rst與flip。debounce module包含四個DFF相連，DFF的clk都使用BASYS內建clk，即100MHz。當四個DFF output都是1代表準確的按下，輸出rst\_debounce、flip\_debounce。

Parameterized\_ping\_pong\_counter module 在前面的Parameterized\_ping\_pong\_counter.v已經解釋，因此不贅述。唯一稍加改變的是在初始時，我改成if(rst\_n)而不使用if(!rst\_n)。這是因為之前使用初始的訊號是0時初始，但在FPGA裡的按鈕是按下為1，因此直接用if(rst\_n)。這邊會送出out與dir。

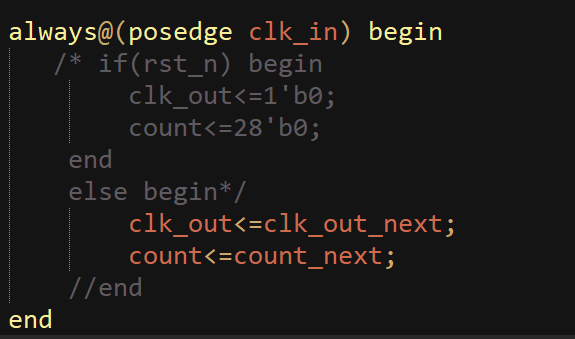
(這會簡化code，但若要符合spec則在呼叫Parameterized\_ping\_pong\_counter module時，要代入!onepulse\_rst，並且在Parameterized\_ping\_pong\_counter module裡要用if(!onepulse\_rst)來初始)。

這部分主要處裡如何將dir與out顯示在FPGA上。因為七段顯示器有四個，需要分別通電而不能同時，因此我利用an\_state依序跑現在要顯示哪個七段顯示器，有點像finite state machine，用來初始的訊號是onepulse\_rst(同樣使用onepulse\_rst=1’b1觸發初始)，也是壓下rst的訊號經過雜訊與單一凸波的處裡，這裡DFF用的clk是clk\_display，因為有關顯示。假設現在的an\_state是an3，也是最左的七段顯示器，只要考慮dir就能決定最左的七段顯示器要顯示上半或下半、輸出an是4’0111，並且指定an\_next\_state是an2。an2也相同。若an\_state是an1要考慮out的數值是多少，決定要不要開每一個小段的七段顯示器、輸出an是4’1101，並且指定an\_next\_state是an0。

* 討論：

這次遇到的比較難處裡的bug是我按下rst後顯示器完全不能跑。後來發現是因為我在clk\_divider中用來初始output clk的信號是rst，parameterized\_ping\_pong\_counter中用來初始counter的也是rst。因此，我若按著rst，會讓三種clk都維持在被初始的值，因此parameterized\_ping\_pong\_counter當然什麼posedge也沒讀到，所以完全進不去counter的module。

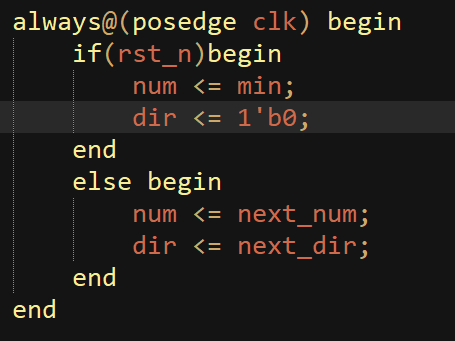
矛盾的情況下我試了不要初始clk\_divider。如圖。



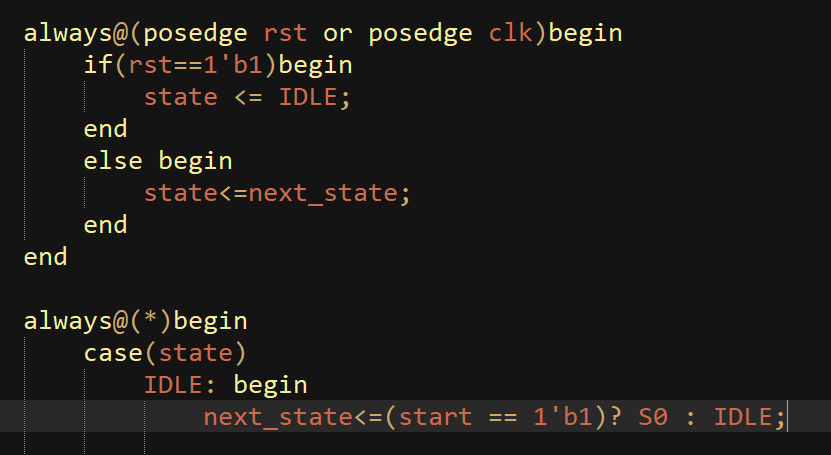
原本我預期這樣會使clk\_out、count都是unknown，因為unknown是reg的default，再用unknown不斷持續跑進DFF，應該全部都維持unknown，所以應該也不能跑。但意外的結果是FPGA完全能正常運作了。

這讓我想到是不是跑simulation的時候若不給初值直接進DFF，reg會顯示X，但在實際的FPGA上，它還是有內建的default值且不是X，有可能是亂數。但因為clk頻率太高，就算一開始不是自己想要的初始值做開始，output的clock還是能按照理想的情況實現，因為肉眼完全看不出來。但多方嘗試後，我發現若不給初始值，FPGA的default是0而不是亂數。這是simulation與FPGA在初始的差異。

不過這樣改後，變成我讓bitstream灌進FPGA後，就算我不按rst它也會馬上開始數，雖然在跑的時候按下rst是可以從min rst的。因為就算沒有按下rst它還是把num初始default為0、dir初始為0於是也可以進去不是rst的block完成一個正常的循環，而不是unknown循環。如圖。



這雖然解決了問題，但要是我想要一個被灌入bitstream後不會自動開始跑，而是要用例如start按紐去觸發呢。我想到的方法是finite state machine，不過可能會需要rst跟start兩個訊號。例如



這樣的話，雖然FPGA default給state值是0，會進入IDLE的block，但要是沒有按下start，next\_state還會是IDLE，於是就達到我想要的效果了，這可能一種可以把迴圈困住而不讓default進入DFF迴圈的方法。

不過若像這題的spec不是同時有start與reset，FSM也沒用。目前沒有想到其他方法是可以只用一個按鈕就實現的方法。

另外的問題是我在決定debounce DFF用的clock時，參考了網路上的資料。資料說，通常，按鍵抖動會產生10—20ms的毛刺，因此要做的實際上就是在20 ms中採樣一次，當檢測到按鍵下降沿的時候，就認定按下，其他狀態忽略。因此我用了四個DFF，每個需要約5ms的clock cycle，因此我將BASYS 的內建10ns clock，擴充約2^9(=512) 倍作為debounce DFF用的clock cycle。不過後來助教說用10ns clock就好，但意思應該不是要做500多個DFF 來debounce，但這樣還是成功跑起來了，這邊我還搞不太清楚。

最後，在demo時發現當按下flip後，數值不會在當下的下個數字就flip，而是中間會多一個cycle才flip，因為跑過wave，所以應該不是parameterized\_ping\_pong\_counter的問題，應該是clock數字設定上的問題，因為有時候會馬上flip，有時候會等待一個cycle，不一定能每次成功。這也是需要再思考的問題。

心得(by郭家偉)：

這次時間花在debug上很久，尤其是思考上述的問題，不過我覺得這反而讓我進步很多，不論是在思考上或是更清楚DFF的模式。雖然期中考爛了，但也不管了，反而更重視每次練習，覺得每次練習都是進步的感覺很棒。

心得(by黎佑廷)

這次的第三題我原本的寫法是flip起來後經過一些延遲counter才有變化，後來經過仔細思考才debug出來，在FPGA的部分， 我覺得最困難的部分就是clock，而且每調一次clock就要等它重新燒一次板子，實在是耗時耗力，跟家偉討論的過程中我也越來越了解clock，知道怎樣功能的module該用什麼樣的clock，雖然花了很多時間與精力研究，但是我覺得很值得，學到了很多。

**成員:**

黎佑廷: 負責第1、3題以及整理report

郭家瑋: 負責第2題以及fpga