LAB3

第七組

黎佑廷 105030009

郭家瑋 105030015

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:

```
assign feedback = DFF[1] ^ DFF[4];
```

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:

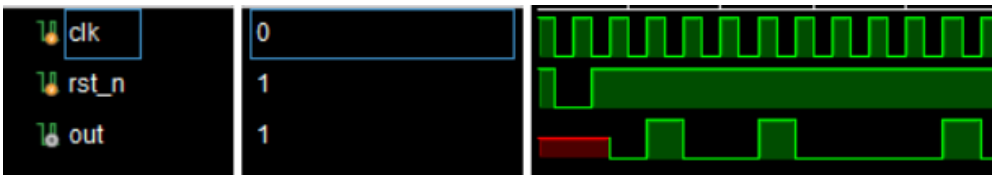
```
always @(posedge clk) begin
    if(!rst_n)begin
        DFF <= 5'b10010;
    end
    else begin
        DFF <= {feedback, DFF[0:3]};
        out <= DFF[4];
    end
end
```

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]。

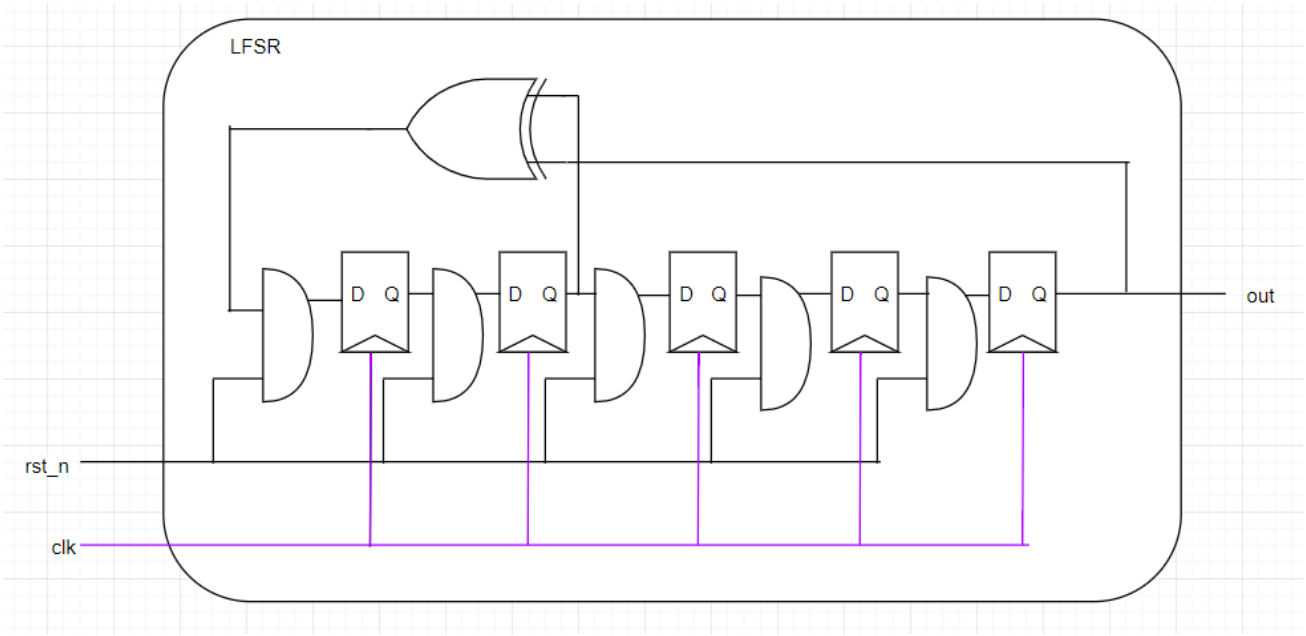
DFF[0]	DFF[1]	DFF[2]	DFF[3]	DFF[4]	output
1	0	0	1	0	x
0	1	0	0	1	0
0	0	1	0	0	1
0	0	0	1	0	0
0	0	0	0	1	0
1	0	0	0	0	1
0	1	0	0	0	0
1	0	1	0	0	0
0	1	0	1	0	0
1	0	1	0	1	0
1	1	0	1	0	1

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:



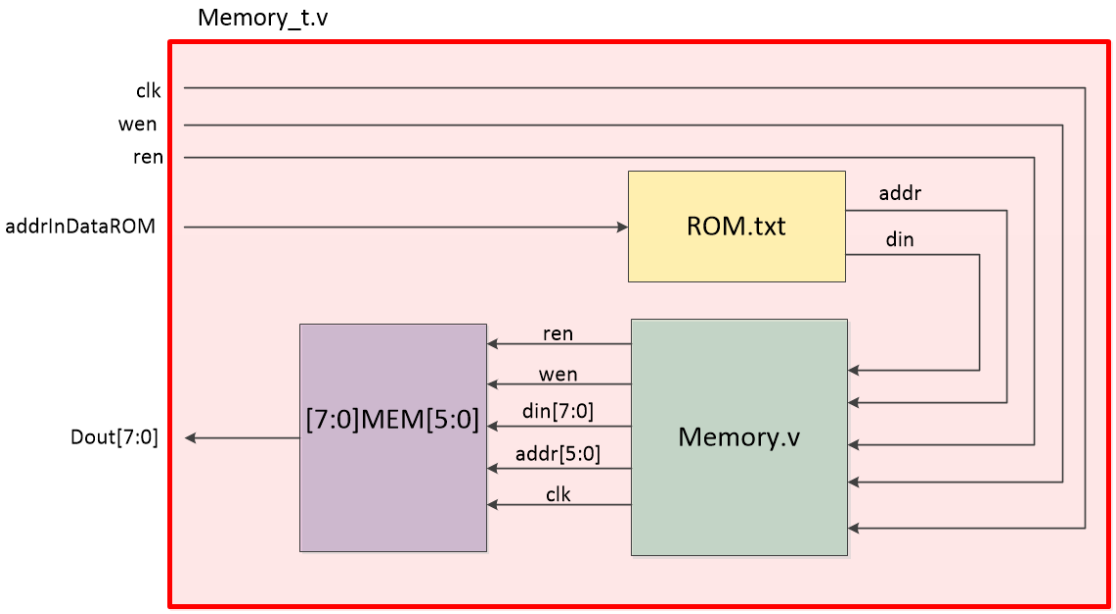
Block Diagram:



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。

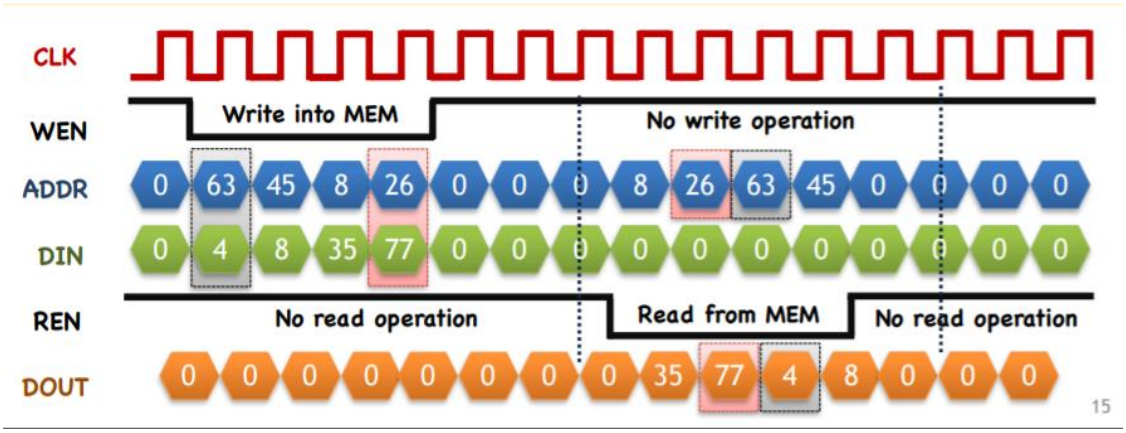
換言之，

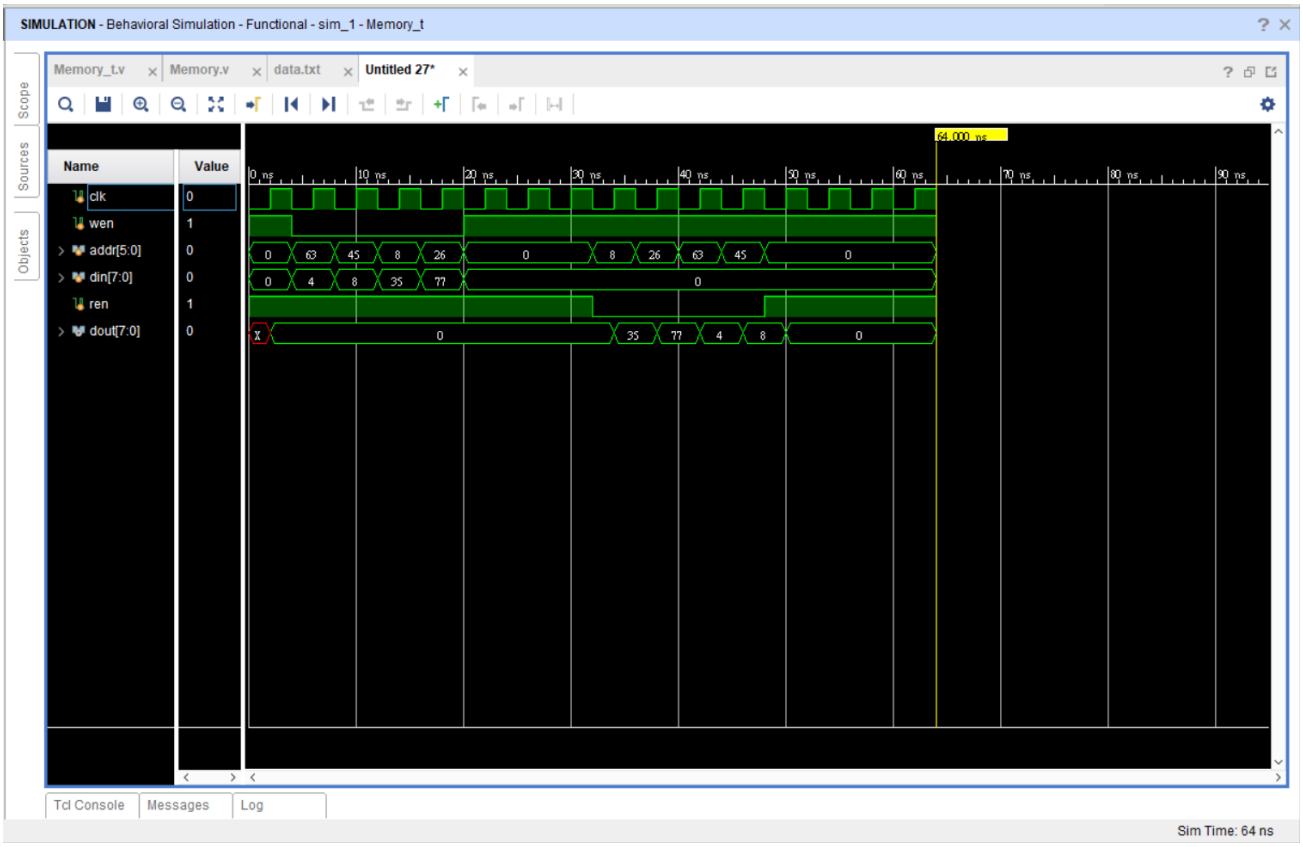
```
always@(*) begin
   if (!ren) begin
      dout=Mem[addrp];
     end
     else if ((!wen)&&ren) begin
        Mem[addrp] = din;
        Dout = 8’b0;
     end
     else begin
        dout = 8’b0;
     end
end
```

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。


CLK
WEN
Write into MEM
No write operation
ADDR
0 63 45 8 26 0 0 0 8 26 63 45 0 0 0 0
DIN
0 4 8 35 77 0 0 0 0 0 0 0 0 0 0 0
REN
No read operation
Read from MEM
No read operation
DOUT
0 0 0 0 0 0 0 0 35 77 4 8 0 0 0
15


SIMULATION - Behavioral Simulation - Functional - sim_1 - Memory_t
Memory_t.v
Memory.v
data.txt
Untitled 27*
Scope
Sources
Objects
Name
Value
clk
wen
addr[5:0]
din[7:0]
ren
dout[7:0]
Tcl Console
Messages
Log
Sim Time: 64 ns

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):

```
always@(posedge clk) begin
    if(!rst_n)begin
        num <= min;
        dir <= 1'b0;
    end
    else begin
        num <= next_num;
        dir <= next_dir;
    end
end
```

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:

```
always @(*)begin
    if(enable)begin···
    end
    else begin
        next_num = num;
        next_dir = dir;
    end
end
```

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:

```
if(enable)begin
    if (max > min && num <= max && num >= min) begin···
    end
    else begin
        next_num = num;
        next_dir = dir;
    end
end
```

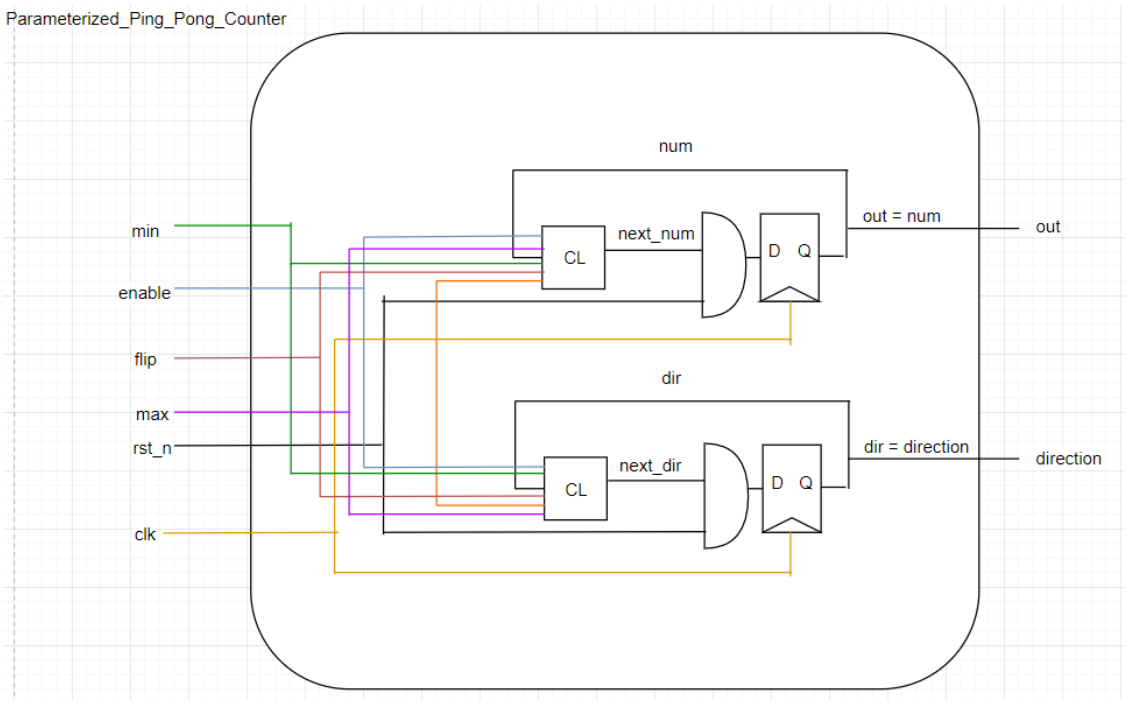
如果 max 以及 min 的相關檢查通過，我的下一步是檢查是否有 flip 訊號，如果有，那就將 dir 變換方向，此外，如果 flip == 1 的時候 dir 是 0 那麼下一個 num 應該要減一，如果 dir 是 1 那麼下一個 num 就加一，如下圖:

```
if (max > min && num <= max && num >= min) begin
        if(!flip)begin···
        end
        else begin
            next_dir = ~dir;
            if(dir == 1'b0)begin
                next_num = num - 4'd1;
            end
            else begin
                next_num = num + 4'd1;
            end
        end
```

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:

```
if(!flip)begin
    if(dir == 1'b0)begin
        if(num == max)begin
            next_num = num - 4'd1;
            next_dir = ~dir;
        end
        else begin
            next_num = num + 4'd1;
            next_dir = dir;
        end
    end
    else begin
        if (num == min) begin
            next_num = num + 4'd1;
            next_dir = ~dir;
        end
        else begin
            next_num = num - 4'd1;
            next_dir = dir;
        end
    end
end
```

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:

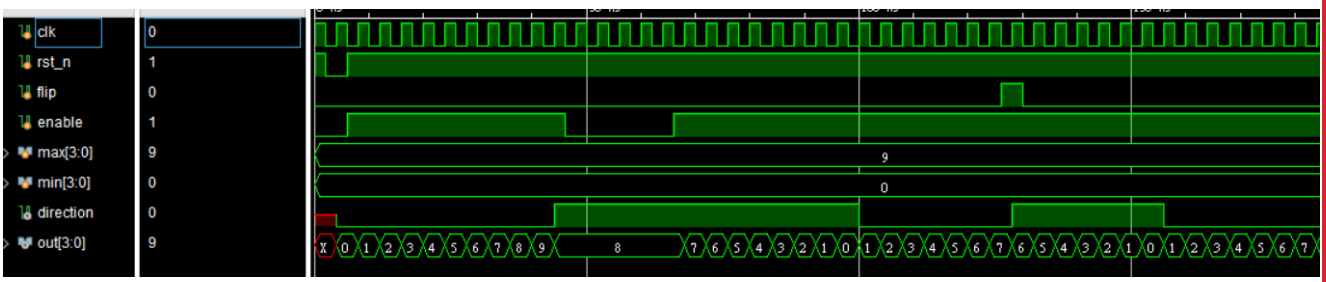
```
initial begin
    @(negedge clk)
    rst_n = 1'b0;
    @(negedge clk)
    rst_n = 1'b1;
    enable = 1'b1;

    #(`CYC * 10)
    enable = 1'b0;

    #(`CYC * 5)
    enable  = 1'b1;

    #(`CYC * 15)
    flip = 1'b1;
    #(`CYC)
    flip = 1'b0;

    #(`CYC * 20)
    $finish;
end
```

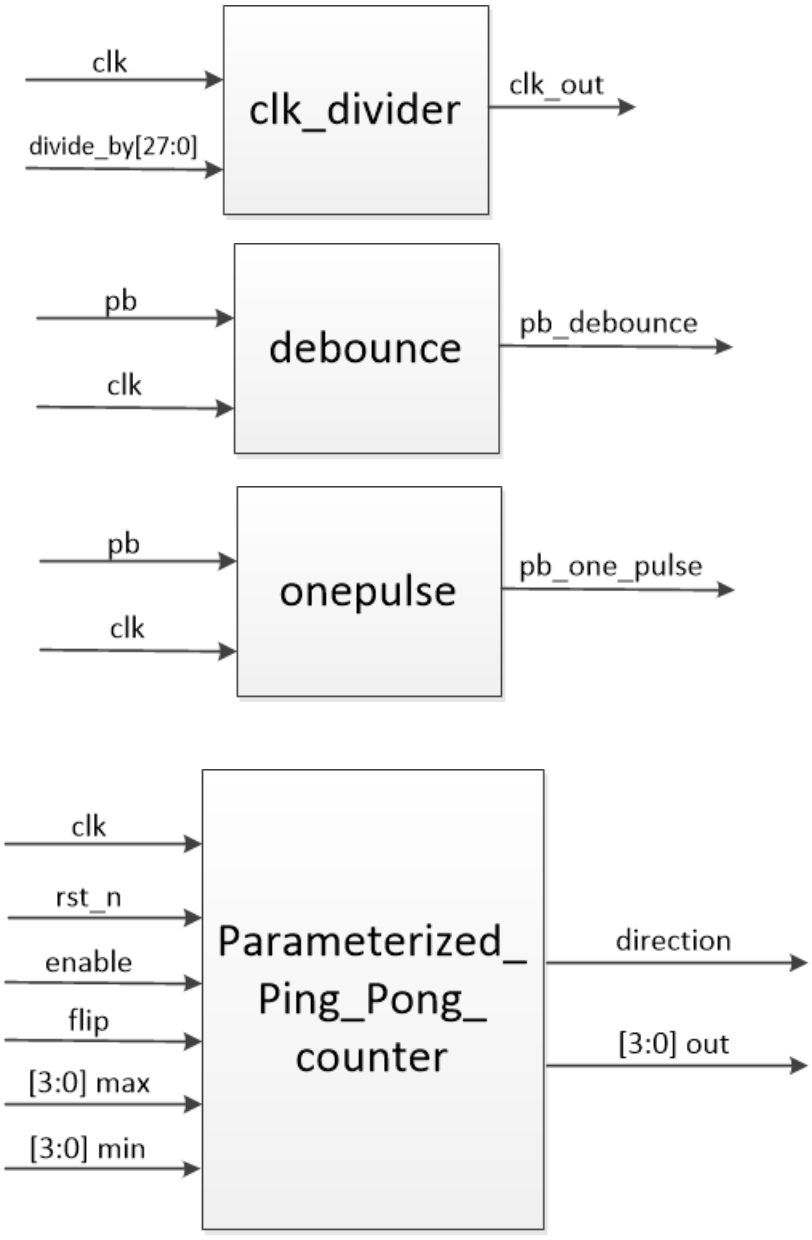


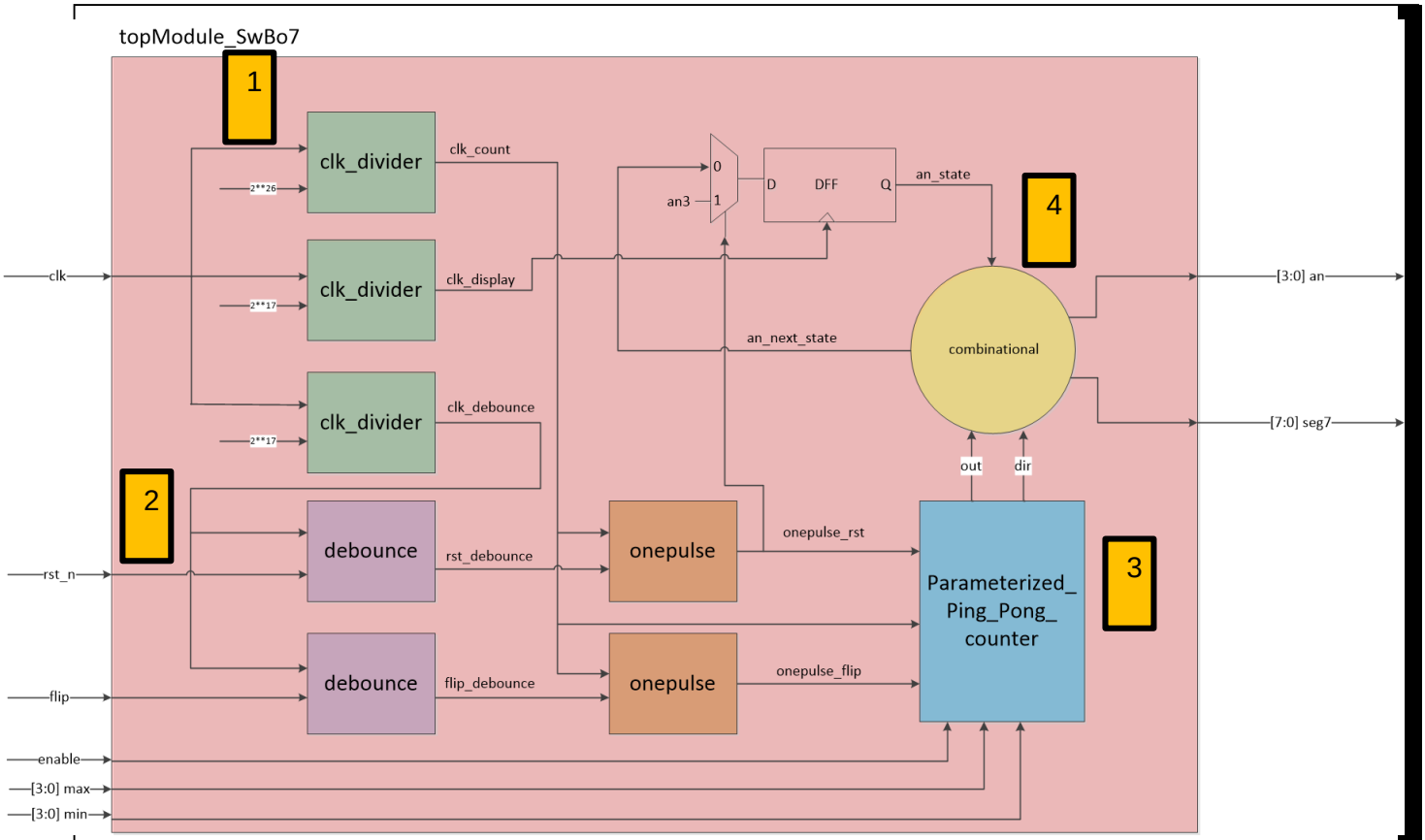
VERILOG QUESTION 4

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。





✓ 說明：

①

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。

②

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。

③

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)。

④

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。

```
always@(posedge clk_in) begin
    /* if(rst_n) begin
        clk_out<=1'b0;
        count<=28'b0;
    end
    else begin*/
        clk_out<=clk_out_next;
        count<=count_next;
    //end
end
```

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。

```
always@(posedge clk) begin
    if(rst_n)begin
        num <= min;
        dir <= 1'b0;
    end
    else begin
        num <= next_num;
        dir <= next_dir;
    end
end
```

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

```
always@(posedge rst or posedge clk)begin
    if(rst==1'b1)begin
        state <= IDLE;
    end
    else begin
        state<=next_state;
    end
end

always@(*)begin
    case(state)
        IDLE: begin
            next_state<=(start == 1'b1)? S0 : IDLE;
```

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