EECS 2070 02 Digital Design Labs 2019

Lab 2

學號：107062107　姓名：王領崧

1.前言

lab 分成兩個部分，實作基本的 4-bit counter 和 8-bitsgray code counter，搭配各種參數決定加減，或者是 hold 住原本的數值等等功能。8-bit gray code counter，需要先刻出 4-bit gray code counter，再把它連接起來。Bonus 的部分，利用 left shift bit 與頭尾 bit 的 XOR，實作偽亂數產生器 LFSR。

2.實作過程

lab2_1

lab2_1，把程式碼都包在同一個 always block 裡面，在 posedge clk 以及 negedge reset 的時候觸發，接著就依照 spec 中的架構，利用巢狀 if-else，依序從 en, dir 判斷下去。

```
always @(posedge clk or negedge rst_n) begin
    if (rst_n == 1'b0) begin
        out = 4'b0000;
    end
    else begin // rst_n == 1
        if (en == 1'b1) begin // enable
            if (in == 1'b0) begin
                if (dir == 1'b1) begin // count UP
                    out = (out == 4'b1111) ? 4'b0000 : out + 1;
                end
                else begin // count DOWN
                    out = (out == 4'b0000) ? 4'b1111 : out - 1;
                end
            end
            else begin // assign data into out
                out = data;
            end
        end
        else begin // disable
            out = out;
        end
    end
```

lab2_1_t

testbench 部分，先把所有的參數初始化，接著延遲 3ns 的時間，將 rst_n 變為 0，讓 lab2_1 的 counter 歸零，延遲 4ns 後，把 rst_n 拉回 1 直到程式執行結束。後面開始檢測各個參數，從 en 開始，接著是累加、扣除的功能，這邊延遲的時間比較長，用來確認進位的正確性，最後再將 in 拉起來，給入一個特定的 data 值，觀察 out 有無正確地接收。

```
lab2_1 counter(.clk(clk), .rst_n(rst_n), .en(en), .dir(dir), .in(in), .data(data), .out(out));

initial begin
    clk = 0;
    rst_n = 1;
    en = 0;
    dir = 1;
    in = 0;
    data = 0;
    #3
    rst_n = 0;
    #4
    rst_n = 1;
    #20
    en = 1;
    #500
    dir = 0;
    #500
    in = 1;
    data = 4'b0111;
    #20
    $finish;
end

always begin
    #5 clk = ~clk;
end
```

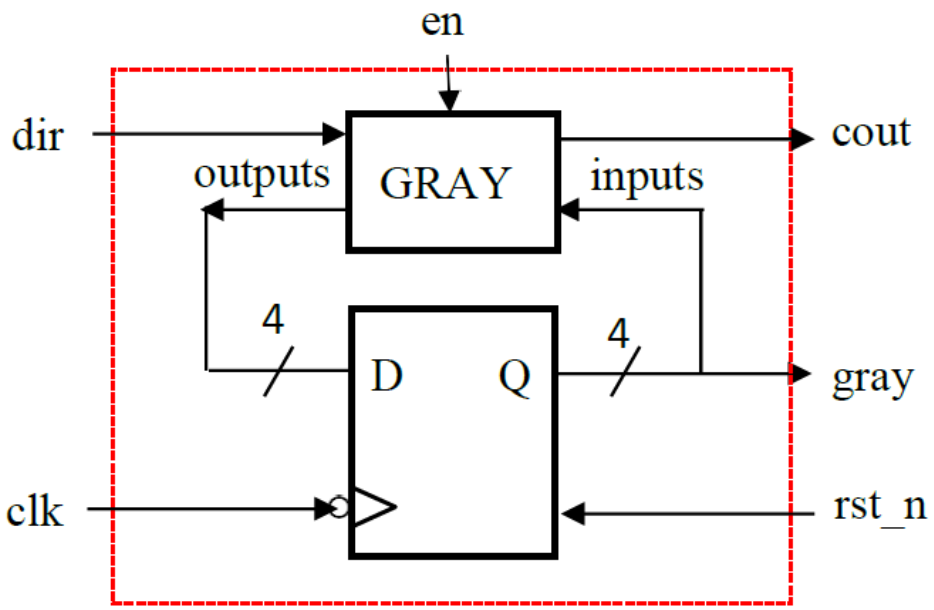
lab2_2

lab2_2，實作 4-bit gray code counter 的部分，一樣也是採用 always block 的方式，在 negedge clk 以及 negedge reset 的時候觸發。gray code 計數是利用 binary 轉 gray，因此額外宣告一個 reg (bin_count)，在每一個 cycle，先累加或是扣除，再進行轉換。cout 的部分，因為左邊 4-bit 要在右邊 4-bit 需要進位的時候才要執行，所以左邊 module 的 en 就是由右邊 module 的 cout 來驅動的，而左邊 module 的 cout 就是整個大 module 的 cout。output cout 是利用 assign 的方式，當在 15 且要累加，以及當在

0 要扣除時，才拉為 1，否則為 0。

```
module lab2_2(clk, rst_n, en, dir, gray, cout);
    input clk, rst_n, en, dir;
    output [7:0] gray;
    output cout;
    wire tmp_cout;

    one_digit_graycode odg0(.clk(clk), .rst_n(rst_n), .en(en), .dir(dir), .gray(gray[3:0]), .cout(tmp_c
    one_digit_graycode odg1(.clk(clk), .rst_n(rst_n), .en(tmp_cout), .dir(dir), .gray(gray[7:4]), .cout

endmodule

module one_digit_graycode(clk, rst_n, en, dir, gray, cout);
    input clk, rst_n, en, dir;
    reg [3:0] bin_count;
    output reg [3:0] gray;
    output cout;

    always @(negedge clk, negedge rst_n) begin
        if (rst_n == 1'b0) begin
            gray = 4'b0000;
            bin_count = 4'b0000;
```

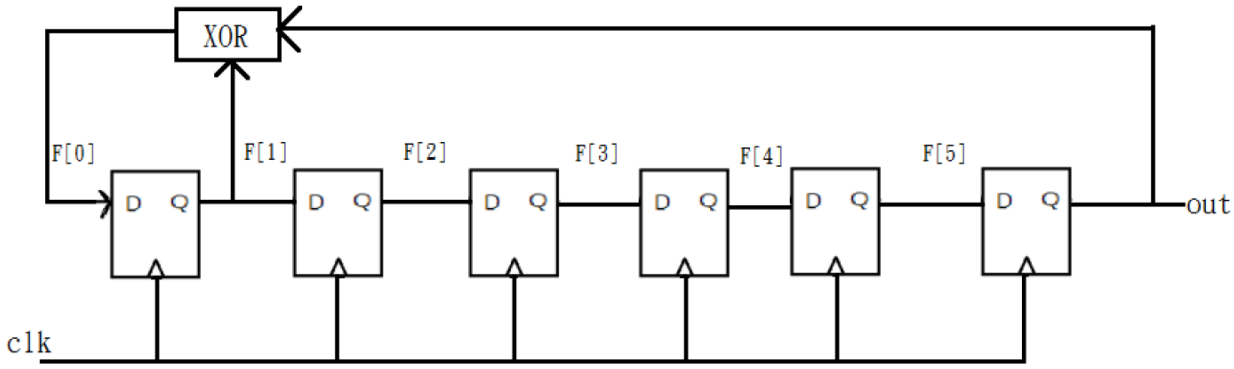
```
        else begin // rst_n == 1
            if (en == 1'b1) begin // enable
                if (dir == 1'b1) begin // count UP
                    bin_count = bin_count + 1'b1;
                    gray = {bin_count[3], bin_count[3]^bin_count[2], bin_count[2]^bin_count[1], bin_cou
                end
                else begin // count DOWN
                    bin_count = bin_count - 1'b1;
                    gray = {bin_count[3], bin_count[3]^bin_count[2], bin_count[2]^bin_count[1], bin_cou
                end
            end
            else begin // disable
                gray = gray;
                bin_count = bin_count;
            end
        end
    end
    assign cout = ((gray==4'b1000 && dir==1'b1 && en==1) || (gray==4'b0000 && dir==1'b0 && en==1)) ? 1'

endmodule
```



lab2_3

lab2_3，完全仿照講義上所繪製的電路圖下去實作，left shift 的部分，採用大括號的方式，串接[4:0]與 XOR 的結果，以達到左移的功能。

```
module lab2_3 (clk, rst_n, out);
    input clk, rst_n;
    output reg out;
    reg [5:0] F;

    always @(posedge clk or negedge rst_n) begin
        if (rst_n == 1'b0) begin
            // reset
            F = 6'b000001;
        end
        else begin
            F = {F[4:0], F[0]^F[5]};
            out = F[5];
        end
    end
endmodule
```



lab2_3_t

testbench 的部分，一開始也是讓 rst_n 短暫的變為 0，進行重製，接著創造出與 lab2_3 module 相同的一個 LFSR，並設有一變數 pass 來檢驗所得到的 out 之正確性，如果正確則 display pass，否則為 wrong。

```
initial begin
    clk = 0;
    rst_n = 1;
    pass = 1;
    #3
    rst_n = 0;
    F = 6'b000001;
    #4
    rst_n = 1;
    #1000
    if (pass)
        $display("---- Pass ----");
    else
        $display("---- Wrong ----");
    $finish;
end
```

```
always begin
    #5 clk = ~clk;
    if (out != F[5]) begin
      pass = 0;
    end
end

always @(posedge clk) begin
   if (rst_n == 1) begin
       F = {F[4:0], F[0]^F[5]};
    end
```

3.學習到的東西與困難

這次利用兩種不同的 counter，複習了 sequential circuit 的用法，也從網路上學習到，利用 always block 中給予 posedge clk 和 negedge rst_n 參數的方式，取代以前在邏設 lab 中多寫一個 DFF module 來執行，不過助教還是有提醒盡量把 combinational 和 sequential 分兩個 block 來寫，一方面不容易錯，另一方面合成電路上也比較不會有問題。Bonus 的部份，程式還蠻簡單的，當作是學習到一種偽亂數表產生的方式吧~

4.想對老師和助教說的話

Demo 的時間助教真的辛苦了，不但讓大家有多次 demo 的機會，也會給予我不錯的提點建議。