Lab 4 FPGA-The 1A2B game

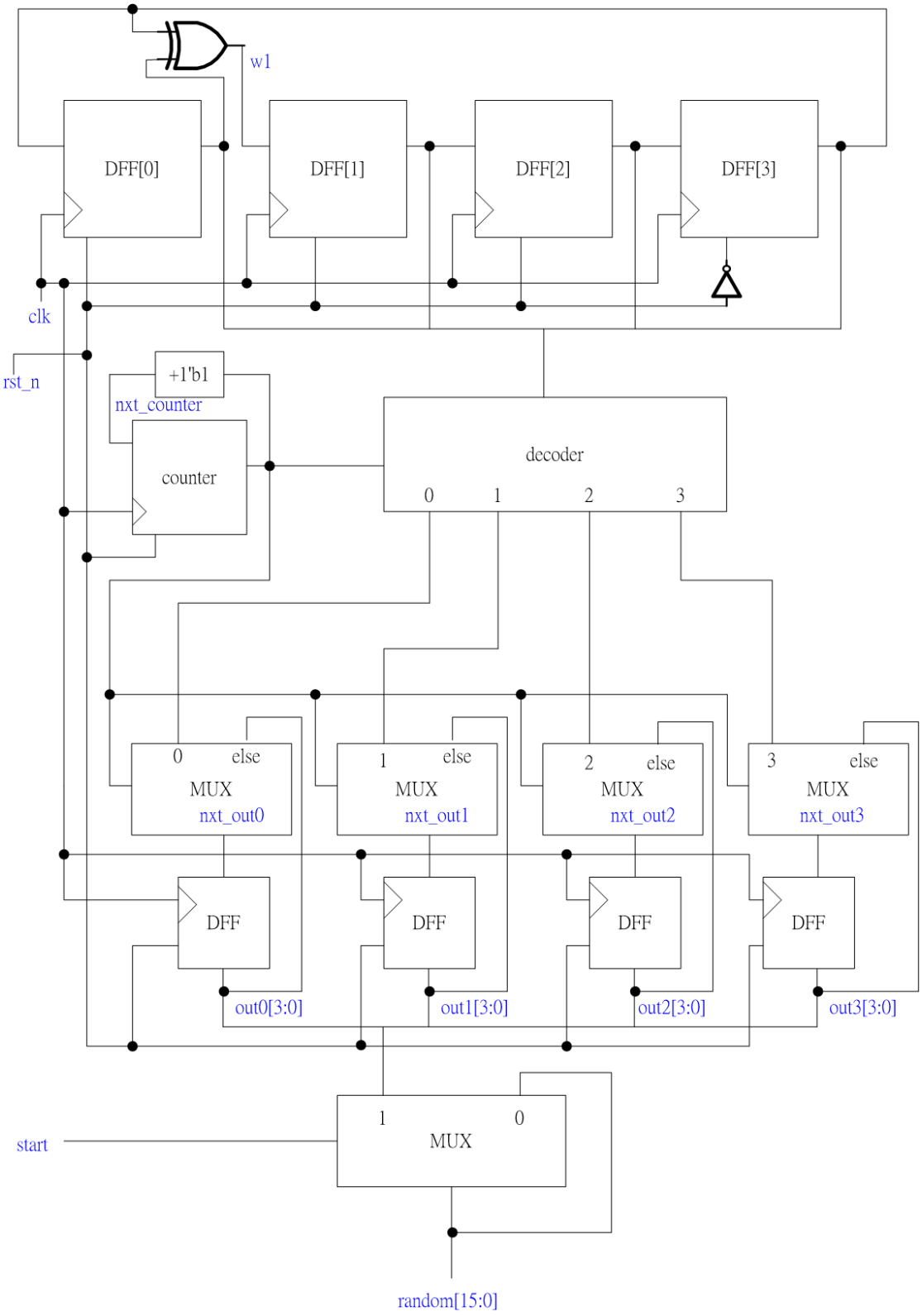
實驗報告

組長:劉奇泓 109033135

組員:洪聖祥 109062315

- Random generator (using LFSR)

Block diagram:



Random generator 的 input 有 clk, rst_n, start(可以切換 output 的值)，output 是 random[15:0]，在 module 內部有 out0~out3 將 output 分開成 4*4bits，以及一個 counter 來選擇 LFSR 的輸出，LFSR 則使用 4-bits 的 DFF，並將 w1 (DFF[0]⊕ DFF[3])連接到 DFF[1]來形成一個 LFSR。

在 posedge clk sequential always block 中，rst_n = 0 時會將 LFSR 內的 DFF[3:0]初始為 4’ b1000，並將 counter 以及 out0~out3 都初始成 0。

```
always @ (posedge clk) begin
if (rst_n == 1'b0) begin
    DFF <= 4'b1000;
    counter <= 2'd0;
    out0 <=  4'd0;
    out1 <=  4'd0;
    out2 <=  4'd0;
    out3 <=  4'd0;
    end
```

反之，會執行 LFSR 的動作，並將 counter 和 out0~out3 輸入下個 cycle 的值；在這之中如果 start = 1 時會將 out0~out3 接到 output random[15:0]。

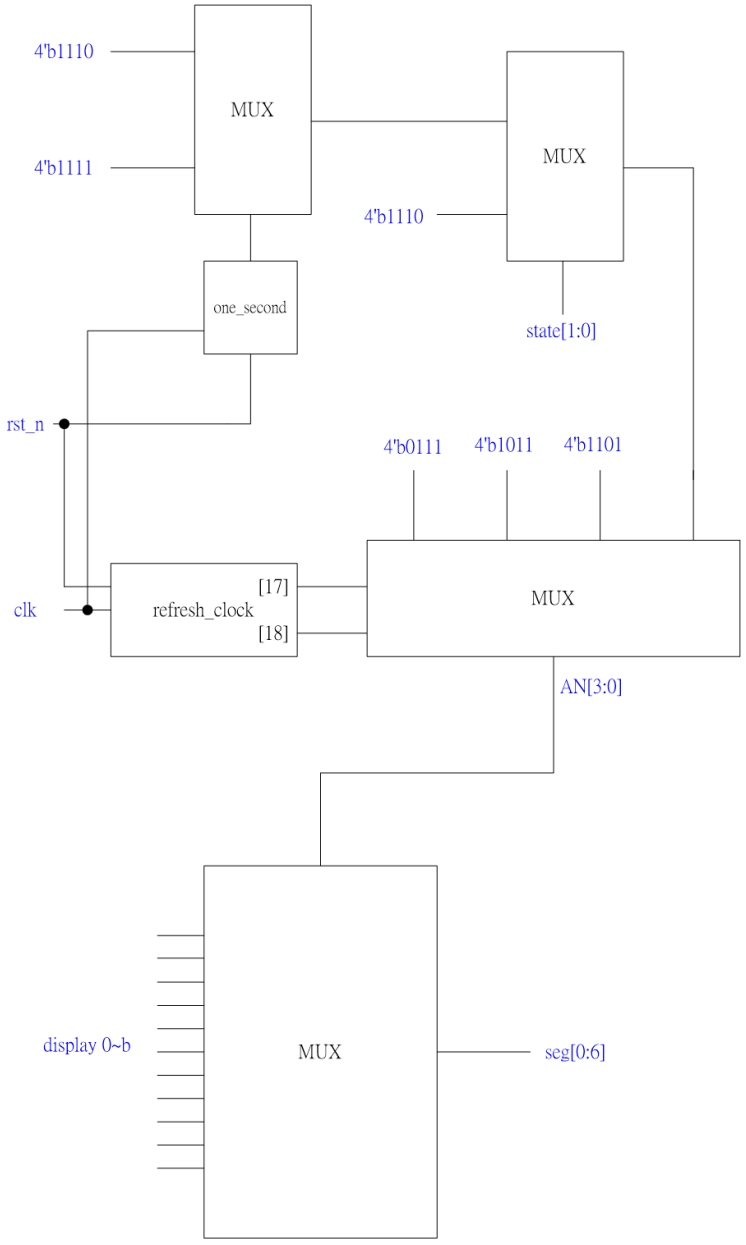
```
else begin
    DFF[0] <= DFF[3];
    DFF[1] <= w1;
    DFF[2] <= DFF[1];
    DFF[3] <= DFF[2];
    out3 <= nxt_out3;
    out2 <= nxt_out2;
    out1 <= nxt_out1;
    out0 <= nxt_out0;
    counter <= nxt_counter;
    if(start) begin
        random[15:12] <= out3;
        random[11:8] <= out2;
        random[7:4] <= out1;
        random[3:0] <= out0;
        end
    else random <= random;
    end
end
```

在 combinational always block 中，我們用 counter 為 case 來選擇要進行的動作。以 counter == 2’b00 為例，此時我們不更動 out1~out3 的值並判斷 LFSR 的 DFF 的值，在 DFF 大於 9 並且不等於 out1~out3 的數字時，out0 會在下個 cycle 改變成 DFF 的值然後將 counter + 1，如果沒有滿足條件就會不更動 out0 以及 counter 的值，一直到 LFSR 產生符合條件的亂數。以此類推我們就能把 LFSR 產生的，0~9 之間且不重複的亂數，輪流給 out0~out3 了。

```
always @ (*) begin
case(counter)
2'b00 : begin
    nxt_out1 = out1;
    nxt_out2 = out2;
    nxt_out3 = out3;
    if(DFF <= 4'd9) begin
        if(DFF != out1)begin
            if(DFF != out2) begin
                if(DFF != out3) begin
                nxt_out0 = DFF;
                nxt_counter = counter + 1'b1;
                end
                else nxt_out0 = out0;
            end
            else nxt_out0 = out0;
        end
        else nxt_out0 = out0;
    end
    else nxt_out0 = out0;
end
```

- Seven segment

Block diagram:



Seven segment 的 input 有 clk, rst_n, inoutput 有 AN[3:0], seg[0:6]，module 中有 one_clock 提供時間間隔為一 sec 的 clock，以及 refresh_clock 提供 AN 更新頻率。

我們使用在 lab 3 fpga 題製作的 one_second 再稍作改動，讓 output 訊號能以 0.5 秒為間隔 complement 一次。

```
always @ (posedge clk) begin
if(rst_n == 1'b0) begin
    counter <= 27'd0;
    second <= 1'b0;
    end
else begin
    counter <= nxt_counter;
    second <= nxt_second;
    end
end

always @ (*) begin
if(counter == 27'd49999999) begin
    nxt_second = ~second;
    nxt_counter = 27'd0;
    end
else begin
    nxt_second = second;
    nxt_counter = counter + 27'd1;
    end
```

refresh_clock 使用的也是在 lab3 製作過的，因此不再詳述，但是在選擇 AN 的值時必須再加上 second 的條件，在 state == S1 時如果 refresh_clock[18:17] == 2’b00 且 second == 1 時 AN = 4’b1110，second == 0 時 AN = 4’b1111，如此便能讓最右邊的顯示器以固定頻率產生閃爍的效果，而在 state == S0 及 S2 時顯示器不會閃爍，照著原本的方式用 refresh_clock 輪替顯示。

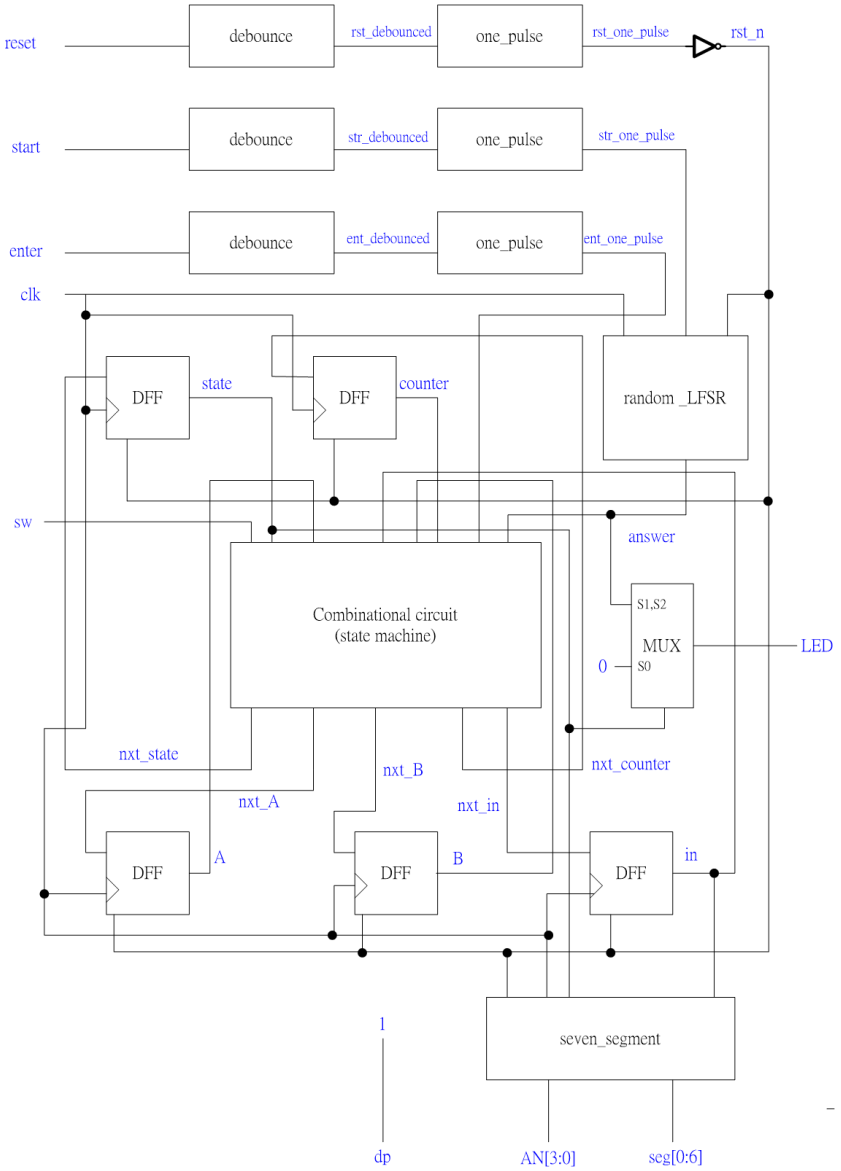
```
case(state)
S0, S2: begin
case(activate_AN)
2'b00 : AN = 4'b1110;
2'b01 : AN = 4'b1101;
2'b10 : AN = 4'b1011;
2'b11 : AN = 4'b0111;
endcase
end
S1: begin
case(activate_AN)
2'b00 : begin
    if(second) AN = 4'b1110;
    else AN = 4'b1111;
    end
2'b01 : AN = 4'b1101;
2'b10 : AN = 4'b1011;
2'b11 : AN = 4'b0111;
endcase
end
```

根據 AN 的值，我們可以再用 AN 判斷要把 in 的哪串 4bit 當成 MUX 的 selection bits，最後再用 in 的其中 4bits 選擇 seg[0:6]的值，以 AN == 4’b0111 為例:

```
always @ (*) begin
if(AN == 4'b0111) begin
    if(in[15:12] == 4'd0) seg = 7'b0000001;
    else if(in[15:12] == 4'd1) seg = 7'b1001111;
    else if(in[15:12] == 4'd2) seg = 7'b0010010;
    else if(in[15:12] == 4'd3) seg = 7'b0000110;
    else if(in[15:12] == 4'd4) seg = 7'b1001100;
    else if(in[15:12] == 4'd5) seg = 7'b0100100;
    else if(in[15:12] == 4'd6) seg = 7'b0100000;
    else if(in[15:12] == 4'd7) seg = 7'b0001101;
    else if(in[15:12] == 4'd8) seg = 7'b0000000;
    else if(in[15:12] == 4'd9) seg = 7'b0000100;
    else if(in[15:12] == 4'd10) seg = 7'b0001000;
    else if(in[15:12] == 4'd11) seg = 7'b1100000;
    else seg = 7'b0000001;
    end
```

- 1A2B

Block diagram:



1A2B 的 input 有 clk、sw、reset、start、enter，output 有 dp、AN[3:0]、seg[0:6]，LED[15:0]，在 module 內部有三組 debounce、one_pulse 可以處理 reset、start 及 enter 的訊號，rst_one_pulse 再經過一個 not gate 變成 rst_n；有一個 random_LFSR 產生亂數，然後將 state、counter、answer、sw、A、B 輸入到 state machine 中處理並得到下個 cycle 的 input；有一個 seven_segment 提供七段顯示器需要的訊號 AN[3:0]及 seg[0:6]，dp 因為不需要小數點所以設為 1，最後用一個 MUX 選擇 LED[15:0] 的值，如果 state == S0 時 LED = 0，state = S1 或 S2 時 LED 會顯示遊戲的答案 answer。

```
debounce d1 (reset, clk, rst_debounced);
debounce d2 (start, clk, str_debounced);
debounce d3 (enter, clk, ent_debounced);
one_pulse o1 (rst_debounced, clk, rst_one_pulse);
one_pulse o2 (str_debounced, clk, str_one_pulse);
one_pulse o3 (ent_debounced, clk, ent_one_pulse);
random_LFSR r1 (clk, rst_n, str_one_pulse, answer);
seven_segment s1 (clk, rst_n, in, state, AN, seg);

assign dp = 1'b1;
assign rst_n = ~rst_one_pulse;
assign LED = (state == S0) ? 16'd0 : answer;
```

在 posedge clk sequential always block 中，rst_n == 0 時會將 state 初始成 S0，counter、A、B 都初始成 0，反之會改變成下個 cycle 的值。

```
always @ (posedge clk) begin
if (rst_n == 1'b0) begin
    state <= S0;
    counter <= 2'd0;
    A <= 4'd0;
    B <= 4'd0;
    end
else begin
    state <= nxt_state;
    counter <= nxt_counter;
    in <= nxt_in;
    A <= nxt_A;
    B <= nxt_B;
    end
end
```

在 combiantional always block 中我們根據 state 為 case 選擇要執行的動作，也就是一個 state machine。在 state == S0 時我們將 nxt_in 設成{4'h1, 4'ha, 4'h2, 4'hb}，讓七段顯示器在這個 state 時顯示”1A2b”，讓 A、B 保持不變；在 start 被按下時會讓下個 state 變為 S1 並讓 counter 歸零，反之就讓 state 停留在 S0。

```
always @ (*) begin
case(state)
S0: begin
    nxt_in = {4'h1, 4'ha, 4'h2, 4'hb};
    nxt_A = A;
    nxt_B = B;
    if(str_one_pulse) begin
        nxt_state = S1;
        nxt_counter = 2'd0;
        end
    else nxt_state = S0;
    end
```

state == S1 時可以用 sw 輸入四個 0~9 的數字，我們用一個 counter 來確認輸入的是第幾個位數， counter == 2’b00 時，我們讓 nxt_in 的前 3 個數字為 0，最後一個數字為正在輸入的 sw；enter 按下後會將 counter + 1、state 停留在 S1、把輸入的數字暫存在 tmp_in[15:12]，並判斷如果 sw 跟亂數產生的答案 answer[15:12]一樣的話把 A + 1，如果跟其他三個數字一樣的話把

B + 1，如果都沒有的話就不改變 A、B。

```
2'b00: begin
    nxt_in = {12'd0, sw};
    if(ent_one_pulse) begin
        nxt_counter = counter + 2'b01;
        nxt_state = S1;
        tmp_in[15:12] = sw;
        if(sw == answer[15:12]) begin
            nxt_A = A + 4'd1;
            nxt_B = B;
            end
        else if(sw == answer[11:8]) begin
            nxt_A = A;
            nxt_B = B + 4'd1;
            end
        else if(sw == answer[7:4]) begin
            nxt_A = A;
            nxt_B = B + 4'd1;
            end
        else if(sw == answer[3:0]) begin
            nxt_A = A;
            nxt_B = B + 4'd1;
```

Counter == 2’b01、2’b10 時的動作也類似，以 2’b01 為例，差別在 nxt_in 的右邊第二個數字會顯示剛剛輸入的 tmp_in[15:12]，以及判斷 A + 1 時要比較 answer[11:8]。

```
2'b01: begin
    nxt_in = {8'd0, tmp_in[15:12], sw};
    if(ent_one_pulse) begin
        nxt_counter = counter + 2'b01;
        nxt_state = S1;
        tmp_in[11:8] = sw;
        if(sw == answer[11:8]) begin
            nxt_A = A + 4'd1;
            nxt_B = B;
            end
        else if(sw == answer[15:12]) begin
            nxt_A = A;
            nxt_B = B + 4'd1;
            end
        else if(sw == answer[7:4]) begin
            nxt_A = A;
            nxt_B = B + 4'd1;
            end
        else if(sw == answer[3:0]) begin
            nxt_A = A;
            nxt_B = B + 4'd1;
```

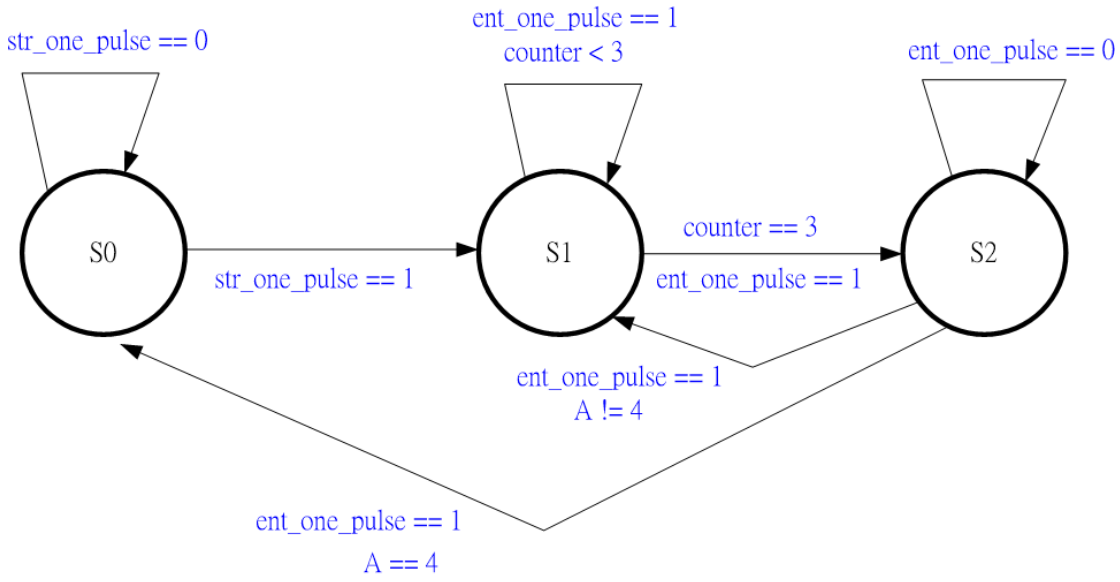
Counter == 2’b11、enter 按下時，下個 state 換到 S2。

```
default: begin
    nxt_in = {tmp_in[15:4], sw};
    if(ent_one_pulse) begin
        nxt_counter = counter + 2'b01;
        nxt_state = S2;
        tmp_in[3:0] = sw;
        if(sw == answer[3:0]) begin
            nxt_A = A + 4'd1;
            nxt_B = B;
            end
        else if(sw == answer[15:12]) begin
            nxt_A = A;
            nxt_B = B + 4'd1;
            end
        else if(sw == answer[11:8]) begin
            nxt_A = A;
            nxt_B = B + 4'd1;
            end
        else if(sw == answer[7:4]) begin
            nxt_A = A;
            nxt_B = B + 4'd1;
```

State == S2 時，七段顯示器的左邊數來第一個數字會顯示 A 的數量，第三個數字顯示 B 的數量，如果 enter 按下時將 A、B、tmp_in 都歸零，如果這時是 4A0B 的情況就猜對了，state 回到 S0，如果沒猜對的話 state 會回到 S1，再輸入一組四位數字。

```
S2: begin
    nxt_in = {A, 4'ha, B, 4'hb};
    if(ent_one_pulse) begin
        tmp_in = 16'd0;
        nxt_A = 4'd0;
        nxt_B = 4'd0;
        if(A == 4'd4) nxt_state = S0;
        else nxt_state = S1;
        end
    else begin
        nxt_state = S2;
        nxt_A = A;
        nxt_B = B;
        end
    end
endcase
end
```

State diagram:



- **心得**

劉奇泓:這次的 fpga 題我學到了 LFSR 的運作，並學會了如何用硬體的角度來製作亂數產生器，在寫得過程中有一度發現明明用

testbench 測試都沒有問題但是燒到 fpga 上都得不到想要的結果，一直到上課的時候聽到教授講課才發現是 multi-assignment 的問題，也讓我知道之後在做 fpga 的 module 時要更注意 sequential 和 combinational 的訊號，還要把每個條件都列清楚，才能讓 fpga 跑得順利。希望這些經驗都能累積起來，在下次的 lab 提升效率，使我有能力處理更多難關。