

**11010EECS207001Logic Design Lab**

**LAB4 FPGA : 1A2B Gmae**

**TEAM9**

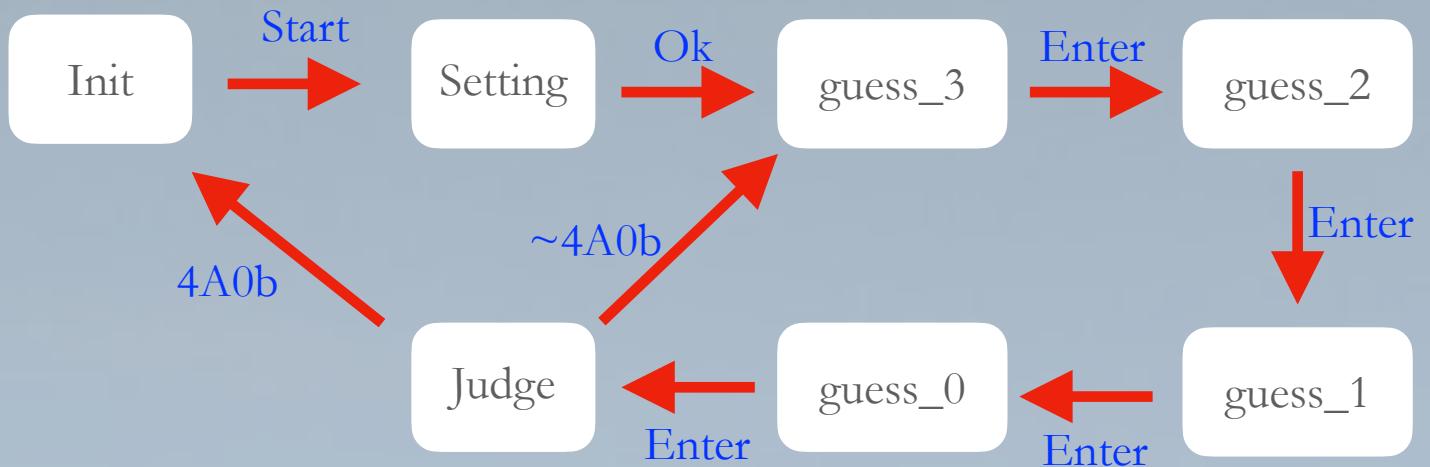
**組長 : 108062213 顏浩昀  
組員 : 106062304 黃鈺舒**

**Prof. Chun-Yi Lee**

**2021.11.11**

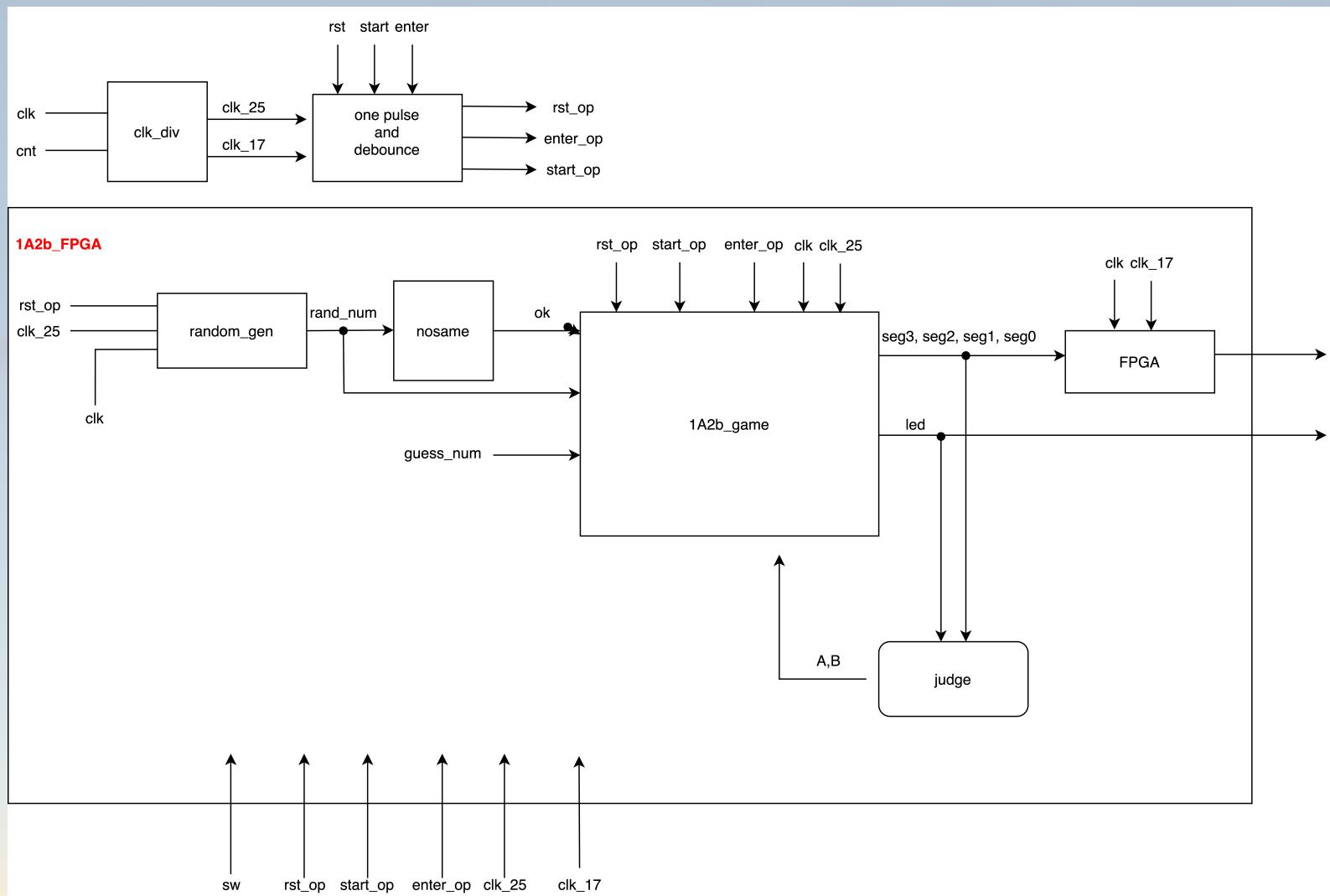
# I. Diagram

## 1. State Transition Diagram



## 2. Logic Design Circuit

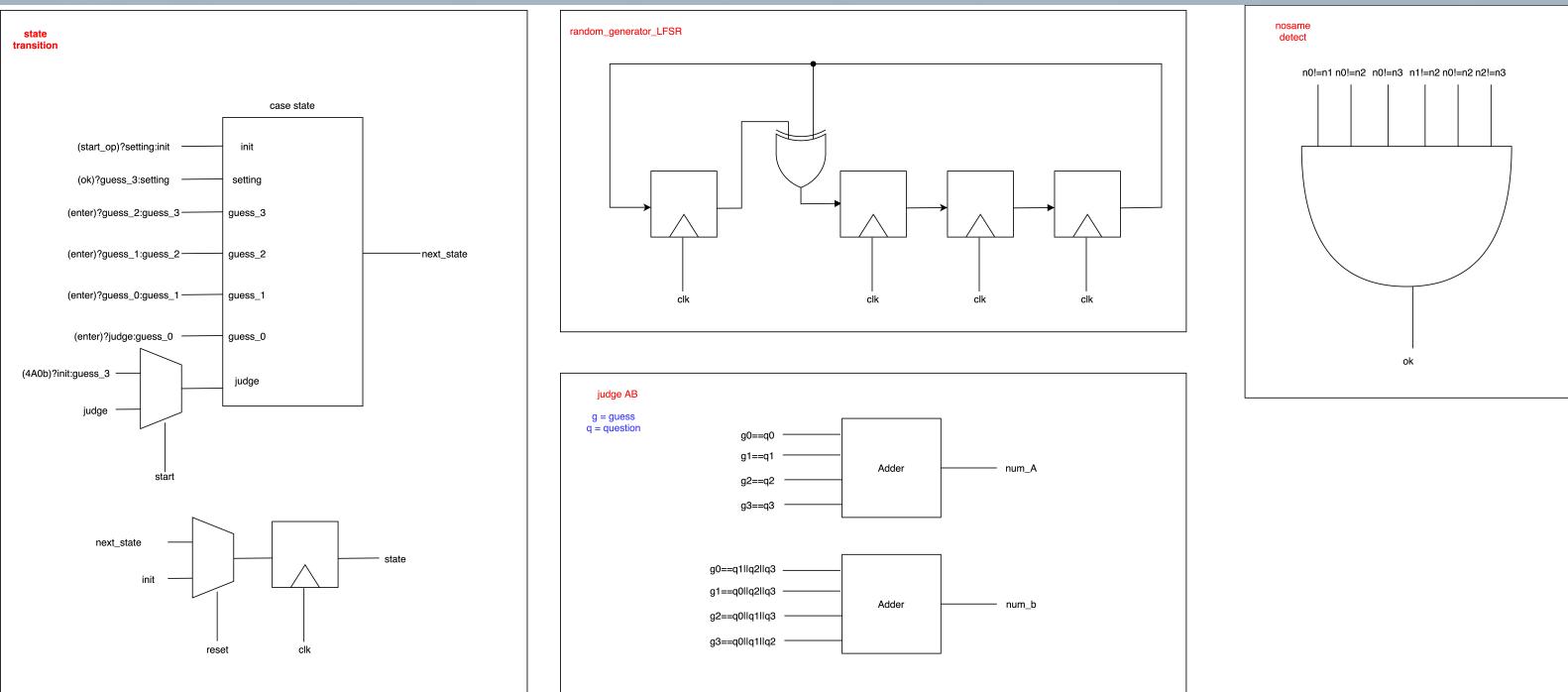
### a) Architecture



# I. Diagram

## 2. Logic Design Circuit

### b) partial circuit design



## II. Implement

這次的FPGA用到需多的子Module，以下將會一個一個分開解釋：

1. AB\_game\_FPGA (Top module)：在top module中負責FPGA輸入及輸出，以及將FPGA的輸入傳入AB\_game module中進行運算。在這個module中比較特別的是控制7-segment輸出的部分，因為在guess階段猜測位的輸出要進行適當的閃爍，因此我們使用一個變數去count(count\_final\_seg)何時亮、何時為暗。在display時，我們本來就需要一個count(cnt)去控制an的輸出，接著我們在要輸出an[0]時去判斷count\_final\_seg即可。（下方擷取count and display控制部分）

```
if(clk_17) begin
    if(state != 3'b0 && state != 3'd1 && state != 3'd6) begin //flash
        cnt <= cnt + 2'b1;
        count_final_seg <= count_final_seg + 8'b1;
    end
    else begin //not flash
        count_final_seg <= 8'b0;
        cnt <= cnt + 2'b1;
    end
end

always@(*) begin
    case (cnt)
        0:begin
            seg = seg3_7;
            an = 4'b0111;
        end
        1: begin
            seg = seg2_7;
            an = 4'b1011;
        end
        2: begin
            seg = seg1_7;
            an = 4'b1101;
        end
        3: begin
            seg = seg0_7;
            if(count_final_seg > 8'b01111111) begin
                an = 4'b1111;
            end
            else begin
                an = 4'b1110;
            end
        end
        default: begin
            seg = seg0_7;
            an = 4'b1111;
        end
    endcase
end
```

2. AB\_gmae：這是這次FPGA中最主要的module，首先我們先設定了七個state以及會用到三種按鍵。當按下reset時，會進到state init，這時的7-segment會被設定為1A2b，16顆LED都不會亮。在這個時候按下start就會進入state setting。state setting會將random\_generator作出的亂數傳入LED燈號中，並且在確定四位數字皆不相同後進入guess state。guess state總共分為四個階段，分別代表猜測四個位置的狀態。當到猜測到最後一位數字時，按下enter則會進入jg(judge) state，並將本輪結果(xAxb)顯示於7-segment，並根據結果判斷下一個state。如果結果為4A0b，next\_state為init，並清空LED，如果不是則會回到guess state。（下方擷取部分程式碼）

```
init: begin
    if(start) begin
        next_state = setting;
        next_seg3 = seg3;
        next_seg2 = seg2;
        next_seg1 = seg1;
        next_seg0 = seg0;
        next_led = led;
        next_A = num_A;
        next_B = num_B;
    end
    else begin
        next_state = state;
        next_seg3 = seg3;
        next_seg2 = seg2;
        next_seg1 = seg1;
        next_seg0 = seg0;
        next_led = led;
        next_A = num_A;
        next_B = num_B;
    end
end

setting:begin
    if(ok) begin
        next_state = guess_3;
        next_seg3 = 4'd0;
        next_seg2 = 4'd0;
        next_seg1 = 4'd0;
        next_seg0 = guess_num;
        next_led = random_num;
        next_A = num_A;
        next_B = num_B;
    end
    else begin
        next_state = state;
        next_seg3 = seg3;
        next_seg2 = seg2;
        next_seg1 = seg1;
        next_seg0 = seg0;
        next_led = led;
        next_A = num_A;
        next_B = num_B;
    end
end

guess_0:begin
    if(enter) begin
        next_state = jg;
        next_seg3 = num_A;
        next_seg2 = 4'd10;
        next_seg1 = num_B;
        next_seg0 = 4'd11;
        next_led = led;
        next_A = tmp_A;
        next_B = tmp_B;
    end
    else begin
        next_state = state;
        next_seg3 = seg3;
        next_seg2 = seg2;
        next_seg1 = seg1;
        next_seg0 = seg0;
        next_led = led;
        next_A = num_A;
        next_B = num_B;
    end
end

jg:begin
    if(num_A==3'd4 && num_B==3'd0) begin
        if(enter) begin
            next_state = init;
            next_seg0 = 4'd11;
            next_seg1 = 4'd2;
            next_seg2 = 4'd10;
            next_seg3 = 4'd1;
            next_led = 16'b0;
            next_A = num_A;
            next_B = num_B;
        end
        else begin
            next_state = state;
            next_seg3 = num_A;
            next_seg2 = 4'd10;
            next_seg1 = num_B;
            next_seg0 = 4'd11;
            next_led = led;
            next_A = num_A;
            next_B = num_B;
        end
    end
end
```

## II. Implement

3. Judge module：這個module是負責判斷猜測的結果。會比較玩家猜測值與與題目，並得出a,b兩個output，並在state guess\_0 (final bit)時將值給next\_A, next\_B，在clock trigger時便會將值給予num\_A, num\_B，並用在state jg時判斷下一個state。

(下圖為實作方式)

```
assign {n3,n2,n1,n0} = num;
assign {q3,q2,q1,q0} = question;

assign a = (n0==q0)+(n1==q1)+(n2==q2)+(n3==q3);
assign b = (n0==q1)+(n0==q2)+(n0==q3) + (n1==q0)+(n1==q2)+(n1==q3) + (n2==q0)+(n2==q1)+(n2==q3) + (n3==q0)+(n3==q1)+(n3==q2);
```

4. Random\_generator module：這個module負責製作亂數作為遊戲謎題。因為每位數字是分開產生，因此分為4bit一組做相同的事情。根據spec要求使用LFSR製作亂數，先在reset時將值設為各自的值，每個clock更改值的規範則是將bit3->bit0, bit1->bit2, bit2->bit3, bit0^bit3->bit1。然後因為數字範圍是0~9，因此只要製作出來的數 $\geq 10$ ，我們全數設為0。這部分比較困難的部分在於變為0之後會影響combinational部分的判斷，因此多使用一個reg去維護數字的產生。

(下方程式碼呈現部分實作，並表示每一位數之初始值)

```
if(clk_17) begin
    if(rst) begin
        random_num[15:12] <= 4'b0001;
        random_num[11:8] <= 4'b0010;
        random_num[7:4] <= 4'b0100;
        random_num[3:0] <= 4'b1000;
        num <= 16'b0001_0010_0100_1000;
    end
    else begin
        num <= next_num;
    end
end

if(next_num[15:12] > 4'd9) begin
    random_num[15:12] <= 4'b0;
end
else begin
    random_num[15:12] <= next_num[15:12];
end
```

```
always@(*) begin
    next_num[15:14] = num[14:13];
    next_num[13] = num[15]^num[12];
    next_num[12] = num[15];
```

5. Nosame module：這個部分是要判斷前面做出來的亂數中有沒有重複的數字，如果沒有才能進入guess state，否則就需要使用另一個亂數作為題目。實現方式是將這個module output做為state setting進入下一個state的判斷標準。

(下圖為module實作，state setting part請見前頁範例圖片)

```
output ok;
assign ok = (num0!=num1) & (num0!=num2) & (num0!=num3) & (num1!=num2) &(num1!=num3) & (num2!=num3) ;
```

6. 剩餘的module是一one pulse, debounce, clock divider這些基本實作，不在這邊多加贅述。

# What do we learn in the Lab

在這次FPGA實作中，遇到的一個最後也沒能完全解決的問題就是亂數產生器。受到Shift有一定規律影響，又加上不能有重複數字出現，以及讓10以上的數字都變0的因素，可以產生的題目序列並不多。在完成後的這幾天思考的解決辦法覺得改變10以上的數的值或許是比較容易的解決方法（不要全部設為0）。因此想請問老師在產生亂數這方面會有什麼改善的建議呢？

另外這次比較難做但有成功解決的是閃爍的燈號，一開始想要用兩種頻率取控制燈號，卻忘記其實四個7-segment是同一個輸出導致失敗。後來想到讓最後一顆燈一次亮一次暗，但那樣是看不出閃爍的。但也因為這個想法，才更近一步想到可以用count的方式紀錄並判斷亮暗。

最後是這次的clock divider換成只會產生一個clock cycle方波的寫法（上次是直接把clock週期延長），這樣才能全部的module都使用system clock trigger，我想這次這麼多按鈕、燈號的狀況下能這麼順利完成也是因為使用正確的coding style。

# Cooperation

108062213 顏浩昀：state transition diagram繪製、電路圖繪製、FPGA 實作、Report製作