EECS 2070 02 Digital Design Labs 2019

Lab 6

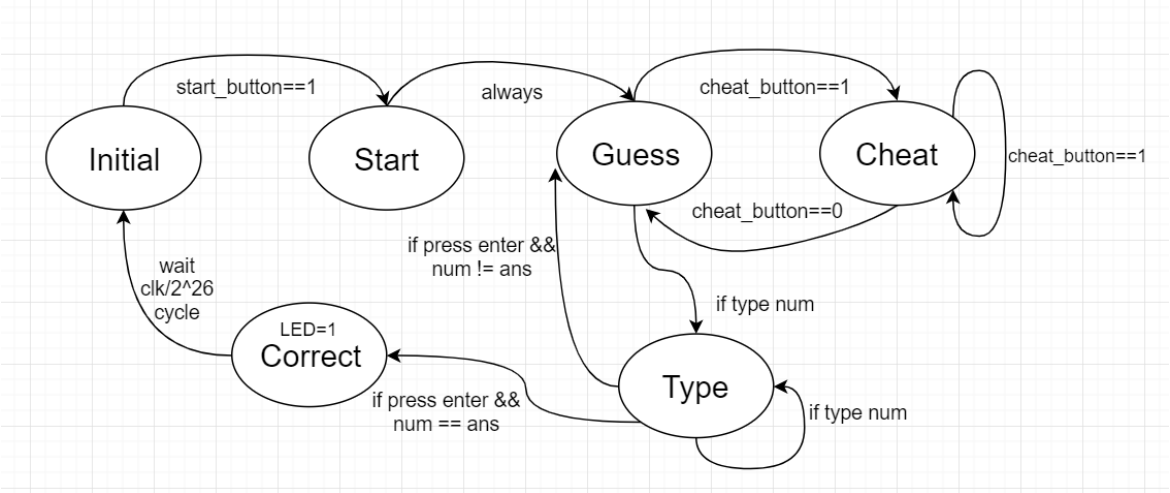
學號：107062107　姓名：王領崧

1.前言

此次 lab 實作達文西密碼，利用鍵盤不斷地輸入猜測的答案，並且搭配按鈕來完成重置、開始、作弊的功能。當輸入到正確答案時，所有的 LED 燈會亮一個(Clk/2^25)的 cycle，之後回到初始的狀態

2.實作過程

State diagram



`Initial state

這次程式使用了 4 種 register 來記錄，分別是 state、left(猜數字的左界)、right(猜數字的右界)、nums(於 7 段顯示器顯示的數字)。在 Initial state 時，將 next_left 設為 00，next_right 設為 99(顯示器上的 99，非十進位的 99)，如果 start_button 被按下的話，next_num 為 0099、next_state 變為 Guess，反之 next_num 為----、next_state 維持在 initial。

```
`Initial : begin
    next_state = `Initial;
    next_nums = {4{4'd10}};
    next_left = 8'd0;
    next_right = {2{4'd9}};
    if (start_de == 1) begin
        next_nums = {{2{4'd0}}, {2{4'd9}}};
        next_state = `Start;
    end
end
```

`Start state

Start state 中，next_num、next_left、next_right 皆等於上一個 cycle 的值，並且將 next_state 設為 Guess。

```
`Start : begin
    next_nums = nums;
    next_state = `Guess;
end
```

`Cheat state

因為 cheat state 為顯示正確答案，與猜數字的左界右界無關，所以 next_left 跟 next_right 皆等於 left、right。另外當 cheat_button 仍被按壓時，next_state 維持在 cheat，next_num 維持在答案，反之的話，next_state 回到 Guess state，next_num 等於{left ,right}。

```
`Cheat : begin
    next_state = `Cheat;
    next_nums = {random, {2{4'd11}}};
    if (cheat_de == 0) begin
        next_state = `Guess;
        next_nums = {left, right};
    end
end
```

`Guess state

Guess state 需要判斷兩種情況，第一種是 cheat_button 被按下去時，next_left 與 next_right 一樣維持原本的值，next_state 變成 cheat，next_num 則變成{答案,--}，第二種是當數字鍵按下去時，表示使用者要開始猜測答案，所以 next_state 變成 Type，next_num

則變成{empty,剛剛輸入的數字}，next_left 與 next_right 一樣維持原本的值。

```
`Guess : begin
    next_state = state;
    next_nums = nums;
    if (cheat_de == 1) begin
        next_state = `Cheat;
        next_nums = {random, {2{4'd11}}};
    end
    if (been_ready && key_down[last_change] == 1'b1) begin
        if (key_num <= 4'd9 && key_num >= 4'd0) begin // pr
            next_nums = {{3{4'd11}}, key_num};
            next_state = `Type;
        end
    end
end
```

`Type state

此 state 主要目的是接受從數字的輸入，分成兩種情形來做判斷。第一種是如果使用者按下的是數字的話，就將 nums 中最右邊的位數進行左移的動作，並將新的數字指派至最右邊的位數，next_state 仍然維持 Type state。第二種是如果按下的是 enter 鍵的話，先判斷 nums 右邊兩位(使用者的答案)是否等於終極密碼，如果是的話，將 next_nums 設定為{ans,ans}，next_state 設定為 Correct ; 如果猜的答案不是終極密碼的話，則判斷是否在範圍之間，如果不是的話，表示是一個無效的輸入，next_left、next_right 不變，next_state 回至 Guess 的 state，如果猜測是在範圍之間且大於終極密碼的話，將 next_right 設為此次的猜測結果，反之如果小於終極密碼的話，則是將 next_left 設為此次的猜測結果，next_state 回到 Guess 中，等待使用者下次猜答案。

```
`Type : begin
    next_state = state;
    next_nums = nums;
    if (been_ready && key_down[last_change] == 1'b1) begin
        if (key_num <= 4'd9 && key_num >= 4'd0) begin // press number (1~9)
            next_nums = {{2{4'd11}}, nums[3:0], key_num};
            next_state = `Type;
        end
        else if (key_num == 4'd10) begin
            if (nums[3:0] == 4'd11 || nums[7:4] == 4'd11) begin
                next_nums = {left, right};
                next_state = `Guess;
            end
            else begin
                if (nums[7:0] == random) begin
                    next_nums = {random, random};
                    next_state = `Correct;
                    next_LED = 16'b1111111111111111;
                end
                else if (nums[7:0] >= right || nums[7:0] <= left) begin
                    next_nums = {left, right};
                    next_state = `Guess;
                end
                else begin
                    if (nums[7:0] > random) begin
                        next_left = left;
                        next_right = nums[7:0];
                        next_nums = {left, nums[7:0]};
                        next_state = `Guess;
                    end
                    else begin
                        next_left = nums[7:0];
                        next_right = right;
                        next_nums = {nums[7:0], right};
                        next_state = `Guess;
                    end
                end
```

`Correct state

因為 LED 需要亮 clk/2^25 的時間，所以我額外使用一個以 clk/2^15 trigger 的 counter，當在 correct state 的時候，進行累加的動作，否則的話，數值持續為 0，在主要的 state transtition 中，判斷 counter 如果是 1024(2^10)的話，表示已在這個 correct state 達到所需要的時間，則將 next_num 設定為----，next_state 設定為 initial，反之的話，代表持續的時間尚未達到，所以 next_state、next_num 以及 next_LED 都維持原本的狀態。

```
`Correct : begin
    next_state = state;
    next_nums = nums;
    next_LED = 16'b1111111111111111;
    if (count == 12'd1024) begin
        next_LED = 16'd0;
        next_nums = {4{4'd10}};
        next_state = `Initial;
    end
end
```

```
always @(posedge clk_div15) begin
    case (state)
        `Correct : begin
            count <= count + 12'd1;
        end
        default begin
            count <= 12'd0;
        end
    endcase
end
```

亂數產生器

亂數產生器使用簡單 counter 的方式，在 1~98 之間不斷的循環，當程式執行至 start state 時，random 就來接收目前 counter 的數值作為答案，如果是在別的 state 的話，就維持原本數值。

```
always @(posedge clk) begin
    case (state)
        `Start : begin
            random[7:4] = cnt / 7'd10;
            random[3:0] = cnt % 7'd10;
        end
        default begin
            random = random;
        end
    endcase
end
```

```
always @(posedge clk) begin
    if (cnt == 7'd98) begin
        cnt <= 7'd1;
    end
    else begin
        cnt <= cnt + 7'd1;
    end
end
```

3.學習到的東西與困難

此次 lab 中使用的亂數產生器，一開始是使用老師教的 LFSR，但是實作後發現要做到 1~98 的循環，還要將數字進行%98+1 有點麻煩，而且害怕取餘數的時間會太長，所以後面就想到了利用 counter 的方式來模擬亂數器，因為每次執行時間的間隔都不同，取出來的值也就無法預測，達到模擬亂數的效果。

在七段顯示器中，因為之前都是使用自己寫的，會將數字取餘數來分成四個位，但是這一次是使用助教提供的，而助教所使用的方法是直接分成四組四個 bit 來判斷，導致一開始寫 lab 的過程中，明明都是正確的，但就是在數字顯示上會發生錯誤。這次 lab 中使用助教的七段顯示器後，發現其實比較好用，也比較簡明，因為相對於還要一位一位的取餘數，分開賦予每一位的數值，一方面不會互相干擾，另一方面是乾淨清楚，數值賦予多少就是多少，能減少 bug 的發生。