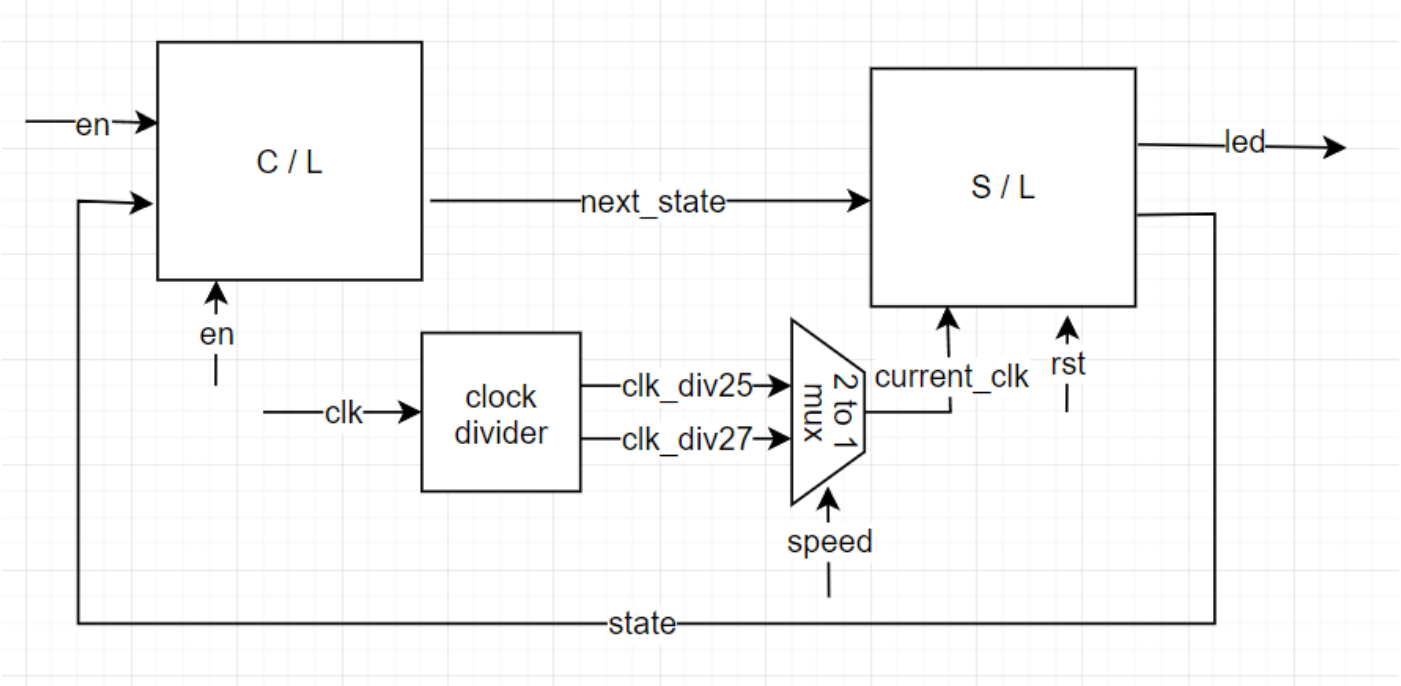
Lab 3

學號: 109062173 姓名: 葉昱揚

A. Lab Implementation

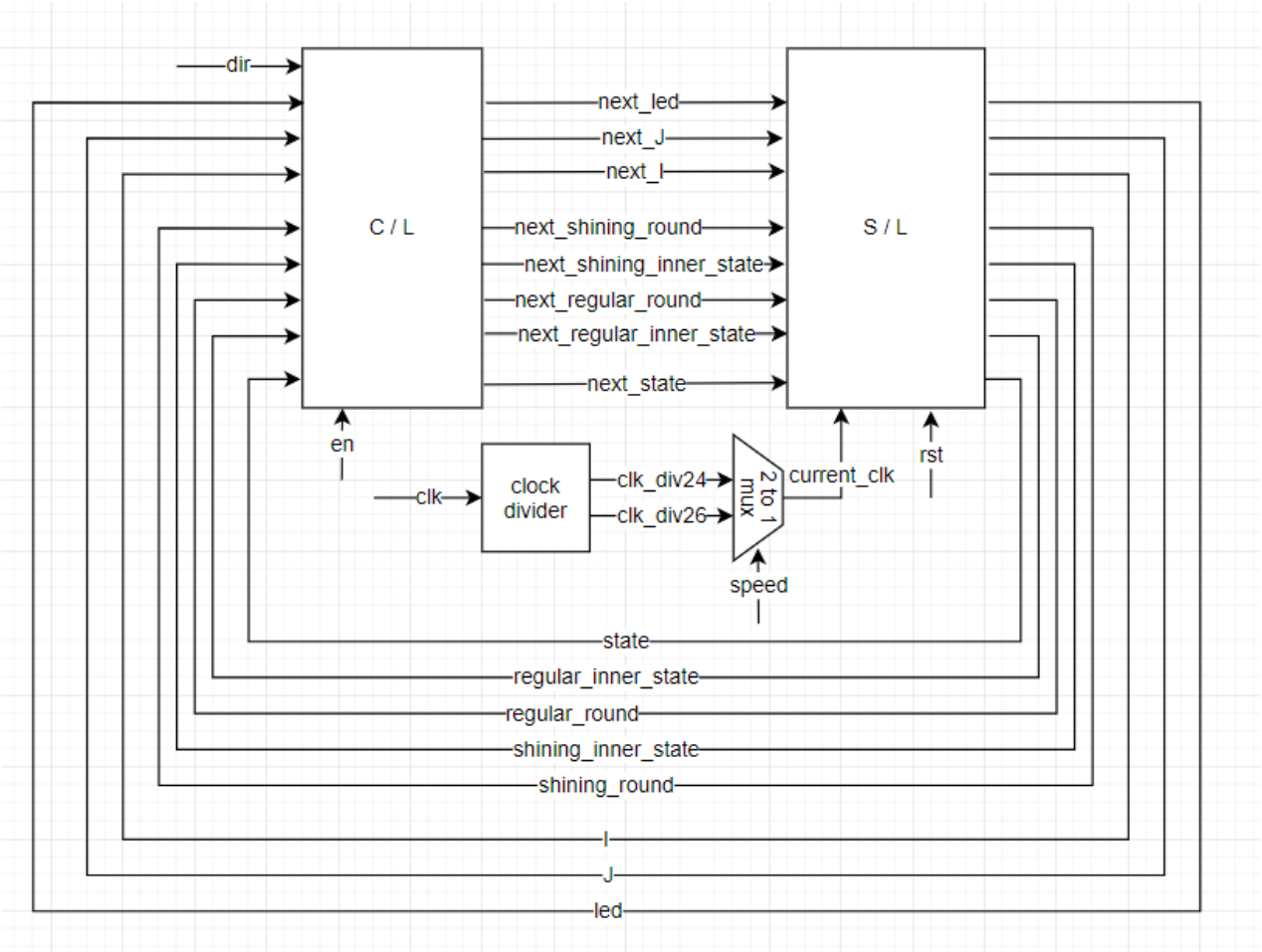
1. Block diagram

* Lab3_1



Lab3-1 是要讓 16 個 led 依照特定的順序亮起，不同的階段會亮起不同的燈。
因此這邊設計一個 state 用來表示現在要〝亮起/熄滅〞的燈光是哪幾顆。
另外，題目設計了兩個頻率不同的 clk，分別為 clk_div25、clk_div27，我用了 2 to 1 mux 依據 speed 的值來決定使用哪一個 clk 來 trigger S/L。

*Lab3_2



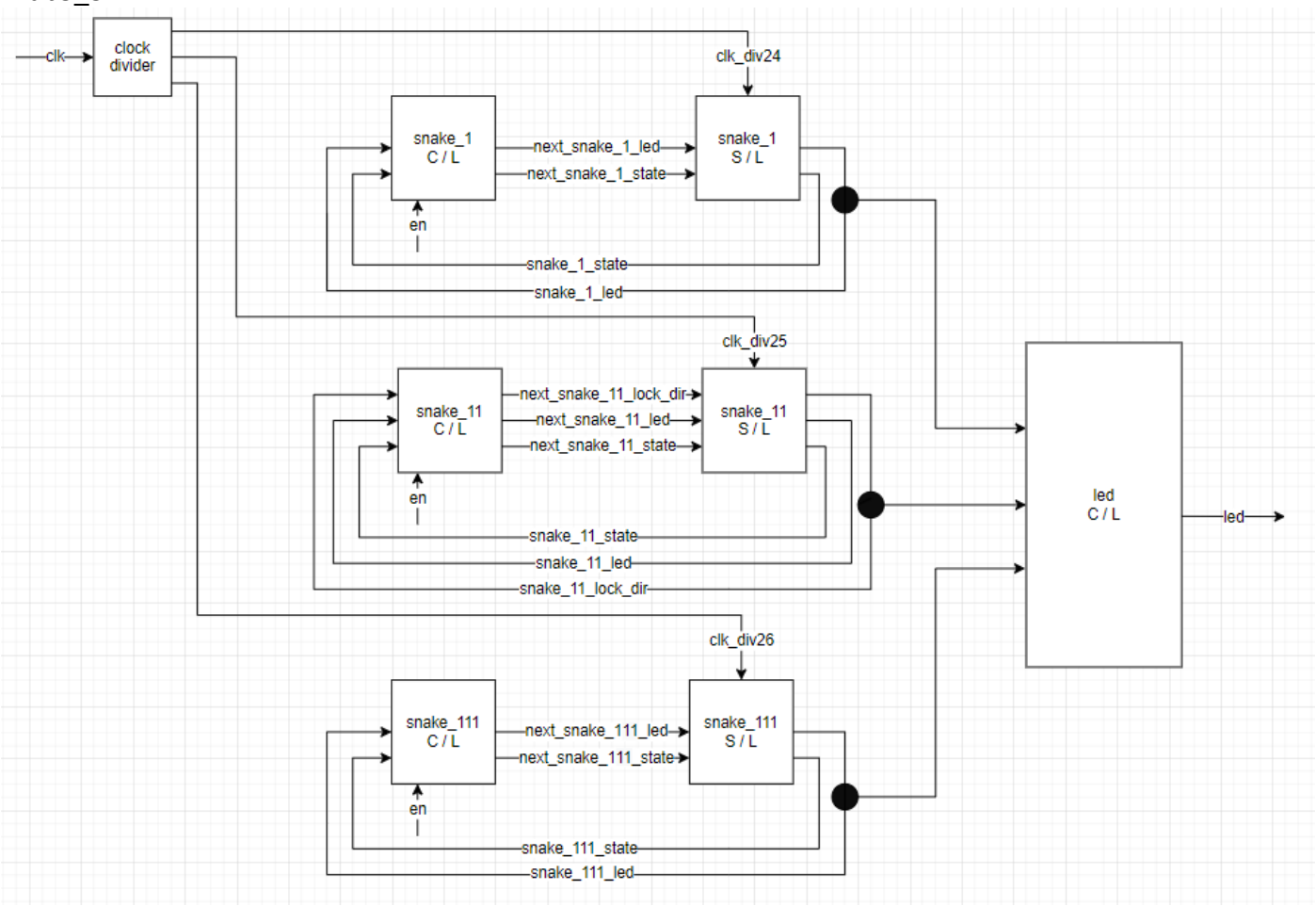
Lab3-2 有三個 STATE，分別為 REGULAR、ESCAPE、SHINING。
除此之外，用了 2 to 1 mux 依據 speed 的值來控制使用 clk_div24 / clk_div26 來 trigger S/L。

REGULAR STATE:
與 lab3-1 一樣，照順序亮起 led 燈，但需要重複三次。
因此我這邊用了 regular_round 記錄總共亮起了幾次； regular_inner_state 則與 lab3-1 的 state 一樣，是用來表示現在要〝亮起/熄滅〞哪幾顆 led。

ESCAPE STATE:
這個 STATE 的重點是紀錄兩顆 led 的位置。
因此這邊用了 I、J 紀錄那兩個位置，並依據 dir 的值，決定下一次需要〝亮起/熄滅〞的兩顆 led 位置，並用 next_I、next_J 紀錄。

SHINING:
這個 STATE 需〝亮起、熄滅〞全部的 led 燈 5 次。
因此這邊用了 shining_round 紀錄總共執行了〝亮起、熄滅〞幾次；shining_inner_state 紀錄現在需要〝亮起〞or〝熄滅〞。

*Lab3_3



Lab3-3 有三條蛇，分別為 snake_1、snake_11、snake_111。
因為 trigger 的 clk 頻率不同，因此設計它們有自己的 S/L，並用自己的 clk 來 trigger。

這邊的重點是 trigger S/L 的 clk 頻率不同，為了讓三條蛇看起來在板子的 led 上跑動，三條蛇必須擁有〝自己的 led〞: snake_1 有 snake_1_led、snake_11 有 snake_11_led、snake_111 有 snake_111_led。每次被 clk trigger 的時候，它們會先判定自己的 state，決定下一次要往左 or 往右走，並更新自己的 led，最後在更新到板子上的 led。如此才能在同一個 led 上讓三條不同 clk trigger 的蛇跑動。

2. Partial code screenshot with the explanation

*Lab3_1

```
case(state)
    0: begin
        led <= 15'b0;
    end
    1: begin
        led[15] <= 1;
        led[11] <= 1;
        led[7] <= 1;
        led[3] <= 1;
    end
    2: begin
        led[14] <= 1;
        led[10] <= 1;
        led[6] <= 1;
        led[2] <= 1;
    end
    3: begin
        led[13] <= 1;
        led[9] <= 1;
        led[5] <= 1;
        led[1] <= 1;
    end
    4: begin
        led[12] <= 1;
        led[8] <= 1;
        led[4] <= 1;
        led[0] <= 1;
    end
    5: begin
        led <= 15'b0;
    end
endcase
```

Lab3-1 唯一個重點是判斷需要〝亮起〞哪幾顆燈。
因此如上圖的 code 所示，我用 state 紀錄現在要亮的燈號，並讓那些燈號亮起/熄滅。

*Lab3_2

```
/* FSM transition */
always@(*) begin
    next_state = state;
    if(en) begin
        case(state)
            REGULAR: begin
                if(regular_round == 3'd3) next_state = ESCAPE;
                else next_state = REGULAR;
            end

            ESCAPE: begin
                if((dir==0) && (led == ~ALL_LED_SHINING)) next_state = SHINING;
                else if((dir==1) && (led ==  ALL_LED_SHINING)) next_state = REGULAR;
                else next_state = ESCAPE;
            end

            SHINING: begin
                if(shining_round == 4'd4) next_state = REGULAR;
                else next_state = SHINING;
            end
        endcase
    end
end
```

上圖是 FSM transition 的 code。
在 lab3-2 裡面有 3 個 STATE。我在這邊用了 regular_round、{dir、led}、shining_round 幾個值來決定下一次會在哪個 STATE。

```
/* next_led */
always@(*) begin
    next_led = led;
    if(en) begin

        case(state)
            REGULAR: begin
                case(regular_inner_state)
                    0: begin
                        next_led = ~ALL_LED_SHINING;
                    end
                    1: begin
                        next_led[15] = 1; next_led[11] = 1; next_led[7] = 1; next_led[3] = 1;
                    end
                    2: begin
                        next_led[14] = 1; next_led[10] = 1; next_led[6] = 1; next_led[2] = 1;
                    end
                    3: begin
                        next_led[13] = 1; next_led[9] = 1; next_led[5] = 1; next_led[1] = 1;
                    end
                    4: begin
                        next_led[12] = 1; next_led[8] = 1; next_led[4] = 1; next_led[0] = 1;
                    end
                endcase
                if(first_in_escape_mode) begin
                    next_led = ALL_LED_SHINING;
                end
            end

            ESCAPE: begin
                case(dir)
                    0:begin
                        next_led[i] = 0; next_led[j] = 0;
                    end
                    1:begin
                        next_led[i] = 1; next_led[j] = 1;
                    end
                endcase
            end

            SHINING: begin
                case(shining_inner_state)
                    0: next_led = ALL_LED_SHINING;
                    1: next_led = ~ALL_LED_SHINING;
                endcase
            end

        endcase
    end
end
```

上圖是針對 led 燈需要〝亮起/熄滅的操作〞。
在 REGULAR STATE，會 regular_inner_state 來判斷現在要亮起哪幾顆 led 燈
在 ESCAPE STATE，I、J 是要〝亮起/熄滅〞的位置。因此這邊的操作很簡單，判斷 dir，並讓 I、J 紀錄的那兩個位置〝亮起/熄滅〞。
在 SHINING STATE，當 shining_inner_state 為 0 的時候讓燈全亮 ； 當 shining_inner_state 為 1 的時候讓燈全滅。

*Lab3_3

Lab3-3 三條蛇的操作非常相似，因此這邊拿出 snake_11 的 partial code 來解釋。

```
always@(*) begin

    if(en) begin
        case(snake_11_state)
            LOCK:begin
                next_snake_11_state = snake_11_lock_dir;
            end

            /* collision with snake_111 */
            MOVE_RIGHT:begin
                next_snake_11_lock_dir = MOVE_RIGHT;
                next_snake_11_state = (((snake_11_led >> 1) & snake_111_led) && ((snake_11_led << 1) & snake_1_led)) ? LOCK :
                                      (snake_11_led == {14'b0,2'b11})       ? MOVE_LEFT :
                                      ((snake_11_led >> 1) & snake_111_led) ? MOVE_LEFT :
                                                                              MOVE_RIGHT ;
            end

            /* collision with snake_1 */
            MOVE_LEFT:begin
                next_snake_11_lock_dir = MOVE_LEFT;
                next_snake_11_state = (((snake_11_led >> 1) & snake_111_led) && ((snake_11_led << 1) & snake_1_led)) ? LOCK :
                                      (snake_11_led == {2'b11,14'b0})     ? MOVE_RIGHT :
                                      ((snake_11_led << 1) & snake_1_led) ? MOVE_RIGHT :
                                                                            MOVE_LEFT ;
            end

            default:begin
                next_snake_11_state = LOCK;
            end
        endcase
    end

    else begin
        next_snake_11_state = snake_11_state;
    end

end
```

上圖是 snake_11 的行走方向判斷，FSM 的圖片放在下方。

Lab3-3 有一個重點就是蛇該往左邊走/右邊走。所以我設計了 snake_11_state，每次行走時判斷是 MOVE_RIGHT、MOVE_LEFT、LOCK 何者，並對應到以下操作:

LOCK:

原地停留一個 clock cycle，下一次行走方向是 snake_11_lock_dir。snake_11_lock_dir 是個特殊的變數，它在 MOVE_RIGHT、MOVE_LEFT 擁有不同的值，是為了幫助判斷 snake_11 被 LOCK 住之前是往哪個方向走，LOCK 結束後可以繼續朝那個方向移動。

MOVE_RIGHT:

1.如果往右走會撞到 snake_111，往左走會撞到 snake_11 -> LOCK
2.如果往右走是牆壁 -> MOVE_LEFT
3.如果往右走是 snake_111 -> MOVE_LEFT
4.以上皆非 -> MOVE_RIGHT

MOVE_LEFT:

1.如果往右走會撞到 snake_111，往左走會撞到 snake_11 -> LOCK
2.如果往左走是牆壁 -> MOVE_RIGHT
3.如果往左走是 snake_1 -> MOVE_RIGHT
4.以上皆非 -> MOVE_LEFT

若當前 en 不為 true，則保持原來方向不動。

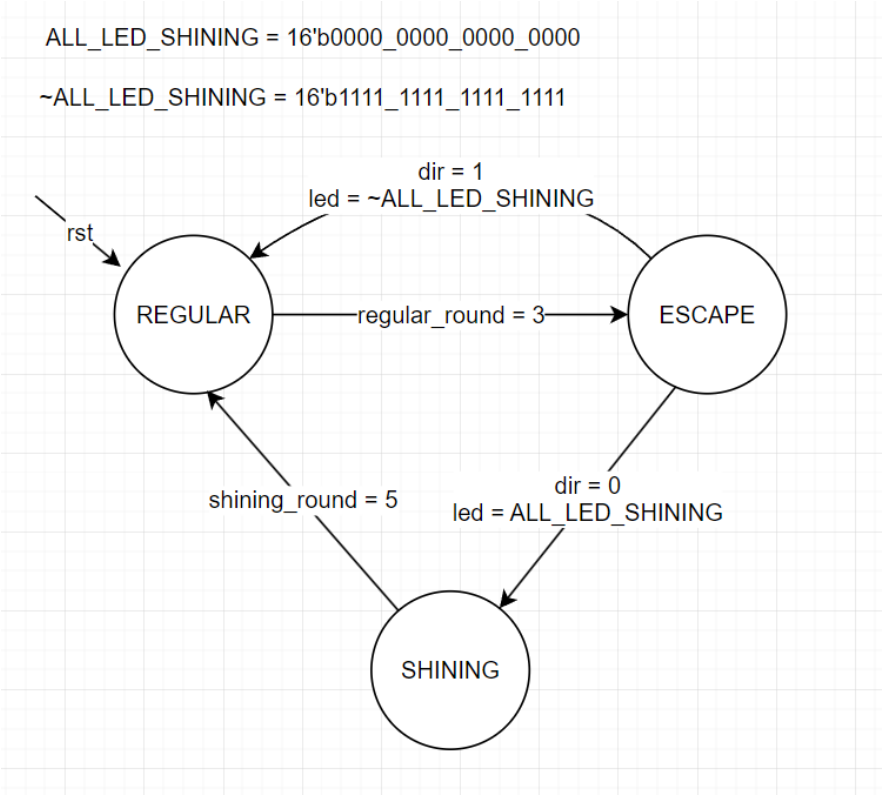
```
always@(*) begin
    if(en) begin
        /* snake 1 */
        next_snake_11_led = (next_snake_11_state == MOVE_RIGHT) ? snake_11_led >> 1 :
                            (next_snake_11_state == MOVE_LEFT)  ? snake_11_led << 1 :
                                                                  snake_11_led;
    end
    else begin
        /* snake 1 */
        next_snake_11_led = snake_11_led;

    end
end
```

上圖是 snake_11 移動的 code。
可以簡單判斷下次的移動方向 next_snake_11_state，並直接 shift snake_11 的 led 完成移動。

3. Finite state machine (FSM) with an explanation.

*Lab3_2



REGULAR:

Rst 後進入這個 STATE。
用 regular_round 記錄次數。當照順序全部的燈〝3 次〞後可以進入 ESCAPE STATE。

ESCAPE:

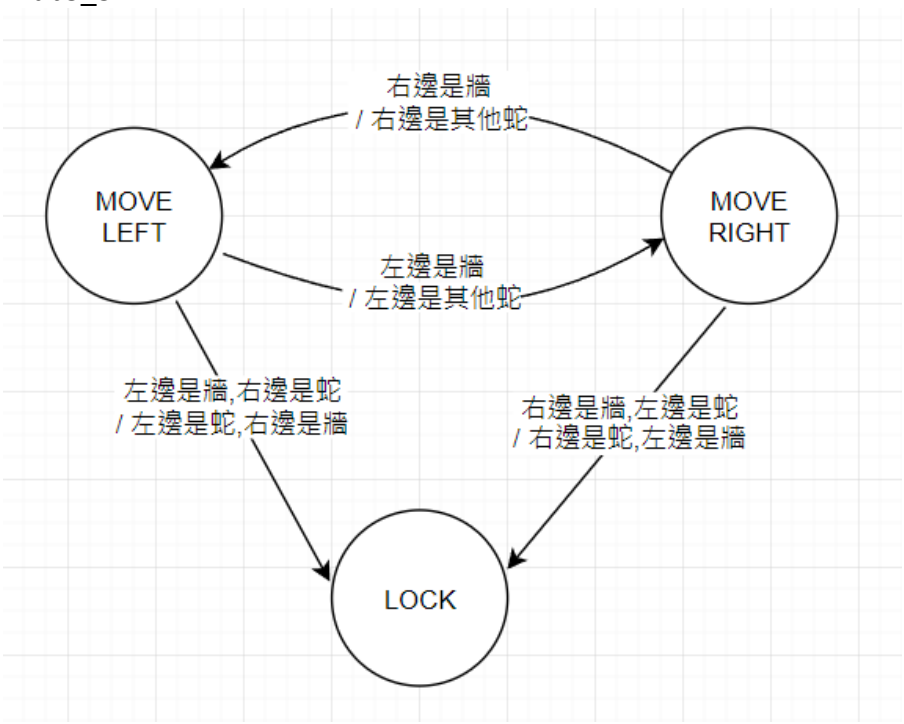
判斷〝dir 的值〞和〝led 燈全亮 or 全滅〞。

若 dir 為 0 且燈全暗，進入 SHINING STATE ；若 dir 為 1 且燈全亮，回到 REGULAR STATE

SHINING:

這個 STATE 需要亮起、熄滅 led 燈 5 次。設計 shining_round 用來紀錄次數，當數字為 5 的時候回到 REGULAR STATE。

*Lab3_3



我把方向判斷設計成 FSM 的形式，共有三個 STATE : MOVE_LEFT、MOVE_RIGHT 和 LOCK。
因為在 partial code 的地方解釋過了，這便就不多做贅述 ！

B. Questions and Discussions

A)

Code 的寫法如下 ：

```
always@(posedge current_clk,posedge rst) begin

    if(rst) begin
        led <= 15'b0;
        state <= 3'b000;
    end

    else begin
        state <= next_state;
        // led <= ....
    end

end
```

這就是 asynchronous reset 的寫法。

不用在意現在是否為 posedge clk，只要接收到 rst signal，立刻 reset 電路。其他賦值、改變 led 的 code 都放在 else 裡面，當 rst 時，保證它能有最高的優先度，其他 code 都不會跑動。

B)
這個問題我想到的解法是當 A 蛇移動時，幫助 B 蛇、C 蛇判斷是否會撞到 A 蛇。
也就是在 A 蛇要移動的時候，如果撞到其他蛇，順便幫其他蛇改變方向。
同理 B 蛇要移動的時候，如果相撞，也幫 A 蛇、C 蛇改變方向。不是 A、B、C 只有在自己移動的階段去判斷方向，讓他們在其他蛇移動的階段也可以判斷相撞、改變方向。

原本的 code 如下圖所示:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*                                                  snake_1                                                       */

reg [1:0] snake_1_lock_dir , next_snake_1_lock_dir;
always@(posedge clk_div24,posedge rst) begin
    if(rst) begin
        snake_1_state <= LOCK;
        snake_1_led <= 16'b1000000000000000;
        snake_1_lock_dir <= MOVE_RIGHT;
    end

    else begin

        if(en) begin
            snake_1_state <= next_snake_1_state;
            snake_1_led <= next_snake_1_led;
            snake_1_lock_dir <= next_snake_1_lock_dir;
        end
    end
end
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

更動後的 code 約略會像下面的樣子:

```
// ... previous code

always @(posedge clk_div24, posedge rst) begin
    if (rst) begin
        snake_1_state <= LOCK;
        snake_1_led <= 16'b1000000000000000;
        snake_1_lock_dir <= MOVE_RIGHT;
    end else begin
        // Continuous Collision Detection
        if (en) begin
            // if snake_1 and snake_11 have a collision
            if ((snake_1_led & snake_11_led)) begin
                // Collision detected, change directions of both snakes
                next_snake_1_state = MOVE_LEFT;
                next_snake_11_state = MOVE_RIGHT;
            end
        end
        // update snake states and positions
        snake_1_state <= next_snake_1_state;
        snake_11_state <= next_snake_11_state;
    end
end
```

C. Problem Encountered

最難的地方是 lab3-3 沒辦訪把三條蛇串在一起。因為它們用不同的 clk trigger，設計最初，我只想出下圖的寫法 ：

```
always@(posedge clk_div24,posedge clk_div25,posedge clk_div26,posedge rst) begin
    if(rst) begin
        // initialize something here
    end

    else begin

        if(clk_div24)begin
            // snake_1 <= .....
        end

        if(clk_div25)begin
            // snake_11 <= .....
        end

        if(clk_div26)begin
            // snake_111 <= .....
        end

    end
end
```

我當初對一個 always block 用三個不同頻率的 clk 來 trigger，很天真地以為不同的 clk 偵測到 posedge 時就進入對應的 if statement，結果是三條蛇完全跑不起來，就算跑起來了也是在到處亂跑，頭尾分離。

後來寫了非常非常久，查過很多資料才發現當一個 always block 用多個 clk edge trigger 的時候它會不知道要用誰來 trigger，所以蛇蛇才跑得很奇怪。

其中的另一個難題是如果三條蛇要在一個 led 上奔跑，那三條蛇必須放在同一個 always block 裡面，這樣他們才能夠一起對同一個 led 改變數值。如果切成多個 always block，並在裡面對同一個 led 賦值會產生 multiple driven 的 error，code 類似下方左圖:

```
always@(posedge clk_div24,posedge rst) begin
    if(rst) begin
        // initialize something here
    end

    else begin

        // led <= snake_1 ;
        // led <= .....;

    end
end

always@(posedge clk_div25,posedge rst) begin
    if(rst) begin
        // initialize something here
    end

    else begin

        // led <= snake_11;
        // led <= .....;

    end
end

always@(posedge clk_div26,posedge rst) begin
    if(rst) begin
        // initialize something here
    end

    else begin

        // led <= snake_111;
        // led <= .....;

    end
end
```

```
always@(posedge clk_div24,posedge rst) begin
    if(rst) begin
        // initialize something here
    end

    else begin

        snake_1_led <= next_snake_1_led;

    end
end

always@(posedge clk_div25,posedge rst) begin
    if(rst) begin
        // initialize something here
    end

    else begin

        snake_11_led <= next_snake_11_led;

    end
end

always@(posedge clk_div26,posedge rst) begin
    if(rst) begin
        // initialize something here
    end

    else begin

        snake_111_led <= next_snake_111_led;
    end
end

assign led = snake_1_led + snake_11_led + snake_111_led;
```

為了解決 multiple driven 的問題，才終於想出要給每條蛇設計自己 led 燈，它們會在自己的 led 燈上奔跑，跑完確定位置後才會更新到真正板子的 led 燈上，寫法形似上方右圖。

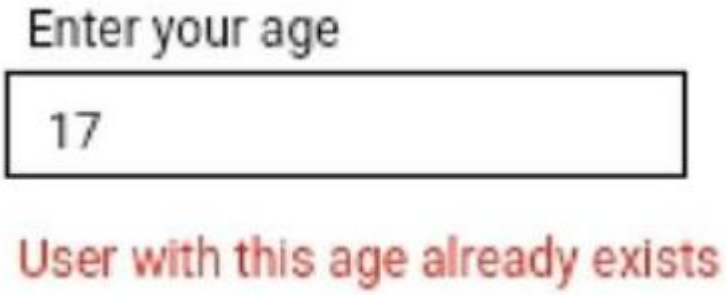
我的心得重點是，若有多個不同 clk 的東西要執行，並在同一個 reg 上顯像，他們需要先在自己的 reg 上跑完，再更新到共同的 reg 上才行。

D. Suggestions

個人認為本次 Lab 偏難，耗時為: Lab3-1 (2 小時) ; Lab3-2 (約 1 天) ;Lab3-3 (約 3 天) 。

Lab3-3 是三條蛇跑來跑去，很早就把兩條蛇互撞的情況處理完，但是處理很久三條蛇撞一起的情況，且 3-3 和 3-1、3-2 關聯性沒那麼大，也許可以把 3-3 再拆分成兩條蛇的題目和三條蛇的題目。

最後附上本次迷因，非常感謝助教對所有人本次 LAB 的幫忙 ！


Enter your age
17
User with this age already exists