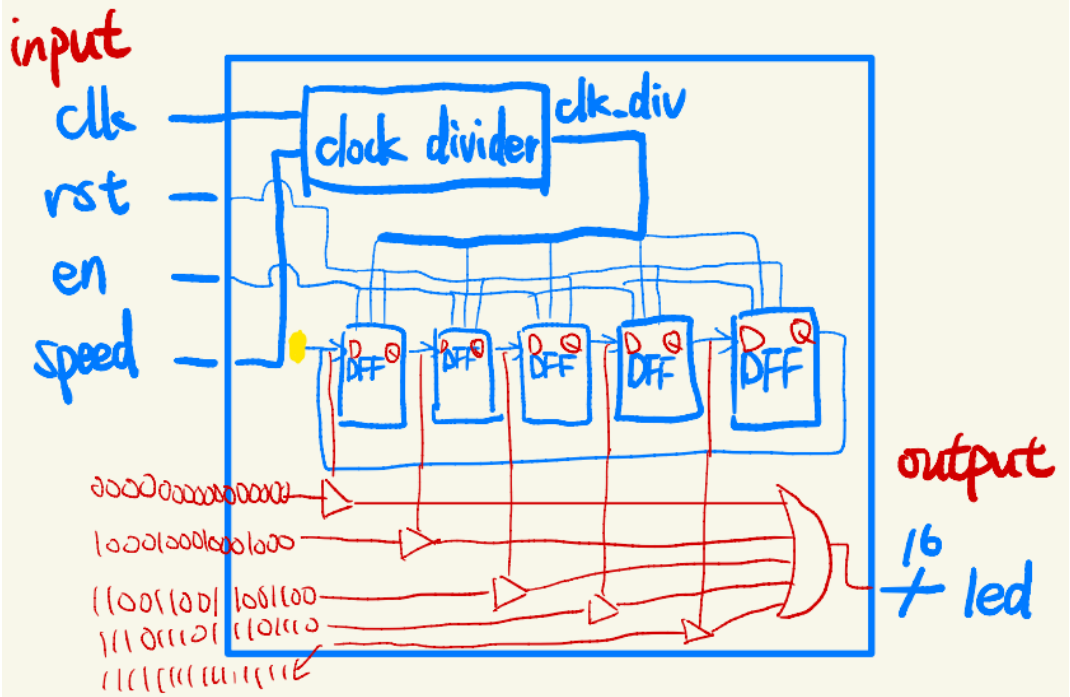
Lab 3

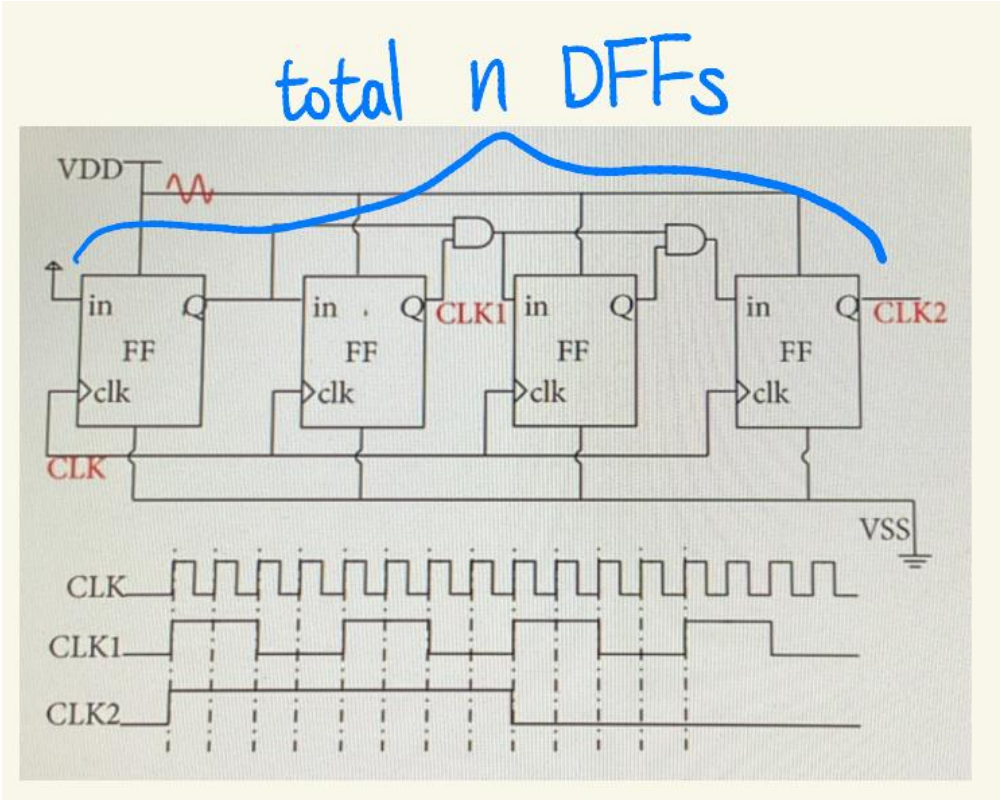
學號: 109021115 姓名: 吳嘉濬

A. Lab Implementation

以下是 Lab3_1 的 block diagram：



D flip flop 可以暫存資料 1 個 cycle，循環串聯連續 5 個 D flip flops 可以使資料每 5 個 cycle 輪回一圈。利用這個技巧，我們可以設計出每 5 個 cycle 一個循環的 machine。



至於 clock divider 的部分，如果要設計一個頻率為原 clock 頻率/(2^n)的新 clock，可以利用上圖

的方法去設計，注意總共只需要 n 個 DFFs 就可以了。

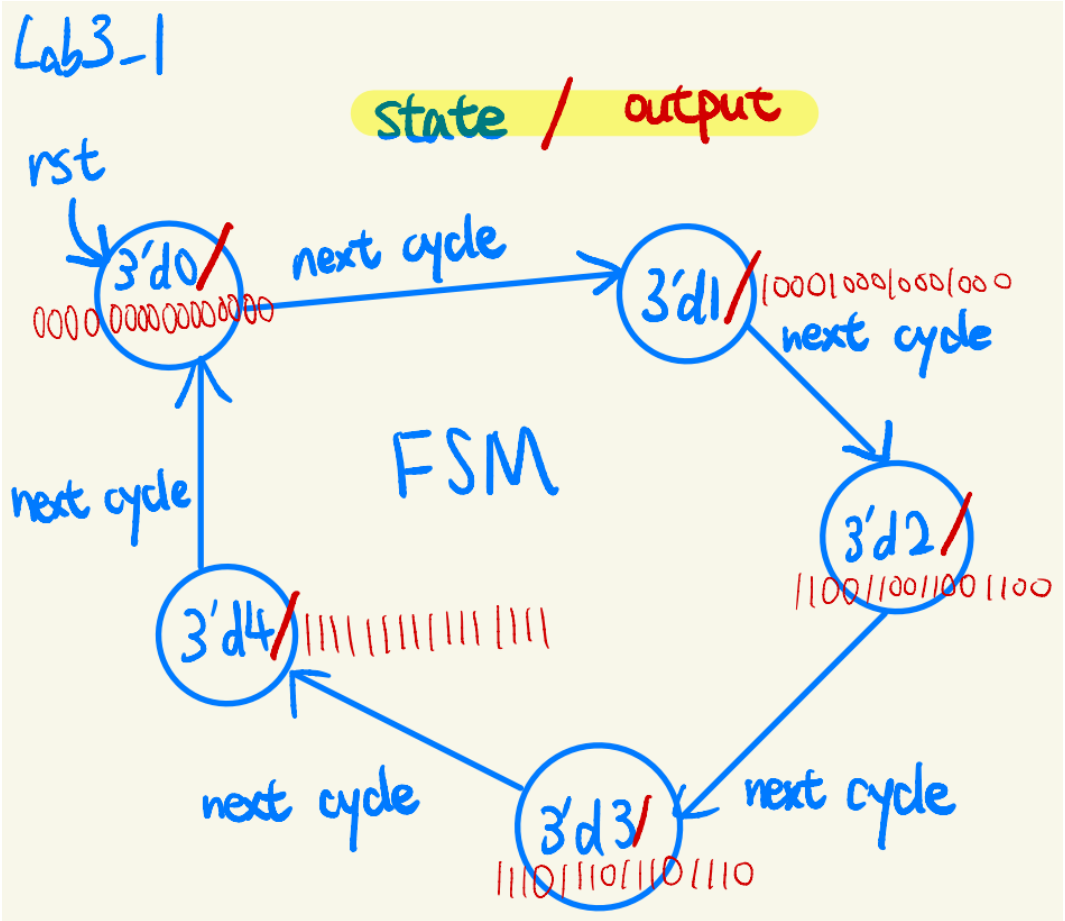
Clock divider 對應的 code 為：

```
module clock_divider(clk,clk_div);
    input clk;
    output clk_div;
    parameter n = 25;
    reg[n-1:0]num;
    wire[n-1:0]next_num;
    always@(posedge clk)begin
        num <= next_num;
    end
    assign next_num = num + 1;
    assign clk_div = num[n-1];
endmodule
```

以下是利用#(XX)的語法去改變 clock divider module 的 parameter n：

```
clock_divider #(25) div1(.clk(clk),.clk_div(clk_div25));
clock_divider #(27) div2(.clk(clk),.clk_div(clk_div27));
```

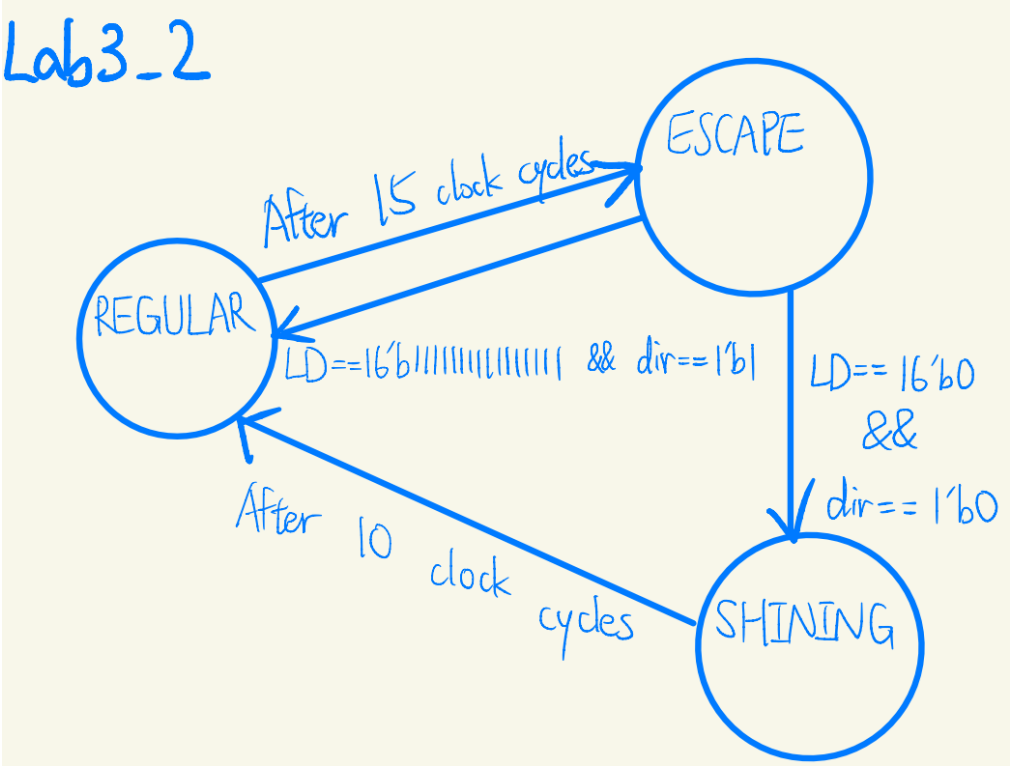
以下是 Lab3_1 的 FSM：



圖為 Moore FSM。依照題義，可以直覺寫出 lab3_1 的 FSM。
在算出 reg LD 後，assign 到 wire led，加上 constraint file 把 FPGA 的 pins map 到 Verilog I/O ports，即可讓 FPGA 板上的 led 登發亮。
利用以上 clock divider 的技術加上 FSM，可以成功完成 lab3_1。

Lab3_2 是 Lab3_1 的延伸，我將針對新增的內容作詳細說明。

以下為 Lab3_2 的 FSM：



根據題義可以畫出以上的 FSM，並進而判斷出 next_LD。

```
case(state)
REGULAR: begin
    if(offset_cnt==5'd14) next_state = ESCAPE;
    else next_state = REGULAR;
    if(offset_cnt==5'd14) next_LD=16'b1111111111111111;
    else if(LD==16'b0000000000000000) next_LD=16'b1000100010001000;
    else if(LD==16'b1000100010001000) next_LD=16'b1100110011001100;
    else if(LD==16'b1100110011001100) next_LD=16'b1110111011101110;
    else if(LD==16'b1110111011101110) next_LD=16'b1111111111111111;
    else if(LD==16'b1111111111111111) next_LD=16'b0000000000000000;
end
ESCAPE: begin
    if(LD==16'b0000000000000000 && dir==1'b0) begin
        next_state = SHINING;
        next_LD = 16'b0000000000000000;
    end else if(LD==16'b1111111111111111 && dir==1'b1) begin
        next_state = REGULAR;
        next_LD = 16'b0000000000000000;
    end else if(dir==1'b0) begin
        next_state = ESCAPE;
        next_LD = LD>>2;
    end else begin
        next_state = ESCAPE;
        next_LD = (LD<<2)+2'b11;
    end
end
```

上圖為在 REGULAR mode 和 ESCAPE mode 下判斷 next_LD 的 kernel code。不同於 Lab3_1，我直接以連續 if … else if …來賦予 next_LD 的值，而非細分成 5 個 state；ESCAPE mode 的部分，考慮不同的 LD 和 dir 決定下個 cycle 的 LD 和 state。

Lab3_3 不同於 Lab3_1、Lab3_2，他運用到了 3 個不同頻率的 clock 去影響著 3 條不同的蛇的移

動，其中牽涉到對碰撞的反應。

以下為 Lab3_3 的 kernel code：

```
always@(posedge clk_div1 or posedge rst)begin
    if(rst) begin
        dir1<=1'b0;  //right
        pos1<=4'd15;
    end else if(!en) begin
        dir1<=dir1;
        pos1<=pos1;
    end else begin
        dir1<=next_dir1;
        pos1<=next_pos1;
    end
end
```

```
always@(*) begin
    if(dir11==1'b1) begin  //default
        next_pos11=pos11+1'b1;
        next_dir11=dir11;
    end else if(dir11==1'b0) begin
        next_pos11=pos11-1'b1;
        next_dir11=dir11;
    end

    if(pos1-pos11<=1 && pos11-pos111<=2) begin
        next_pos11=pos11;  //stay
        next_dir11=dir11;  //keep direction
    end else if(pos1-pos11<=1) begin
        next_pos11=pos11-1'b1;
        next_dir11=1'b0;  //right
    end else if(pos11-pos111<=2) begin
        next_pos11=pos11+1'b1;
        next_dir11=1'b1;  //left
    end
end
```

左圖是 Snake1 在"自己的 cycle"更新的資料，分別是 dir1 和 pos1，即方向和位置，在這裡，我們不像前面 lab 直接存 LD，因為這會使判斷 boundary condition 以及更新更複雜，於是我改存"位置"，詳細說明請見下面 C part "Problem Encountered"。

右圖是 Snake11 判斷自己 next_pos11 和 next_dir11 的地方。第一個 if statement 是 default 的情況，即沒有碰到 boundary condition 時一般採取的反應；第二個 if statement 是考慮遇到 boundary condition 時採取的應對，從 code 由上到下，依序是判斷：同時碰撞 Snake1 和 Snake111 時(即被夾在中間不能動彈)、只有碰撞 Snake1 時、只有碰撞 Snake111 時。根據遇到的 boundary condition，會 output 出不同的 next_pos11 和 next_dir11。

利用先前所提及實作 clock divider 的方法，加上 3 條蛇各自對應於左上方和右上方圖片的 code，就能成功實作 Lab3_3。

B. Questions and Discussions

A. In lab3_1, rst has the highest priority than any other signal (in most hardware designs, it's true as well). How do you implement that?

```
always@(posedge clk_div or posedge rst)begin
    if(rst) begin  //led all light off
        state<=3'd0;
    end else if(!en) begin  //unchange
        state<=state;
    end else begin  //keep changing
        state<=next_state;
    end
end
```

在這個 always block 當中，rst 是 asynchronous positive reset，即 rst 一旦被 trigger(set)，會立即執行 always block(不用等到下一個 clk_div 的 posedge)；而在 always block 當中，第一個被優先判斷的就是 rst signal，所以可以達到 rst 有 highest priority 的效果。

B. In lab3_3, we simplify the direction policies of snakes to only apply at the moment the snake is going to move. What if the policies affect every time snakes meet each other? (Simply explain what you are going to change in your code).

在原本的 code 當中，我讓 snake 被 trigger 的當下，也就是 snake 對應的 clk 產生 posedge 時，再去看有沒有碰到其他蛇(或是牆壁)，做出相對應的判斷(下一個位置、下一個方向)。

至於如何達成 Question B 的效果，我們可以先有以下觀察：此特別情況發生在 Snake1 和 Snake11 相撞，或是 Snake11 和 Snake111 相撞。

我們可以在頻率高的蛇相撞的當下，去 mark 尚未被 trigger、頻率較低的蛇。當頻率較低的蛇被 trigger 時，判斷是否被 mark 過，有的話就改變方向，並判斷出相對應的下一個位置。

C. Problem Encountered

一開始在設計 lab3_3 時，以為可以比照 lab3_1 和 lab3_2，直接算出每個 cycle 哪些 led 燈要亮，然後在 module 的末端 assign led = LD;，如下圖：

```
    endcase
end

assign led = LD;

endmodule
```

結果在實作時發現，3 條蛇能跑出的 led output 很複雜，如果先算出每條蛇的 reg LD，利用”|” or operation 去組合，最後再 assign 到 led，會發現每條蛇的 LD 很難被直接計算出來。於是我改變儲存的資料，instead of 存每條蛇會讓哪些 led 燈亮，我選擇去存它們的”位置”，我把”蛇的位置”定義為每條蛇最左邊的那個 led 位置，接著，我只要在最終 assign led 時，做一些修正，就能正確 output 三條蛇的樣子，即：

```
end

assign led = (1'b1<<pos1 | 2'b11<<(pos11-1'd1) | 3'b111<<(pos111-2'd2));

endmodule
```

Snake1 對應 1’b1<<pos1；Snake11 對應 2’b11<<(pos11-1’d1)；Snake111 對應 3’b111<<(pos111-2’d2)，我把這三項用”|” or operation 去取聯集，即能得到想要的 output。

D. Suggestions

謝謝助教在這次 TA time 幫我解惑。

這次 lab3 是第一次用 FPGA 板做 lab，聽助教說以後的 lab 都是直接觀察 FPGA 板而非看 waveform，這是不是意味著 testbench 的製作並非必要了呢，雖然我覺得製作 testbench 還蠻助於觀察個別訊號並 debug 就是了。

剛剛看了一下 lab4 感覺又是個兩三天跑不掉的作業量 QQ，希望在實作時可以 debug 順利。