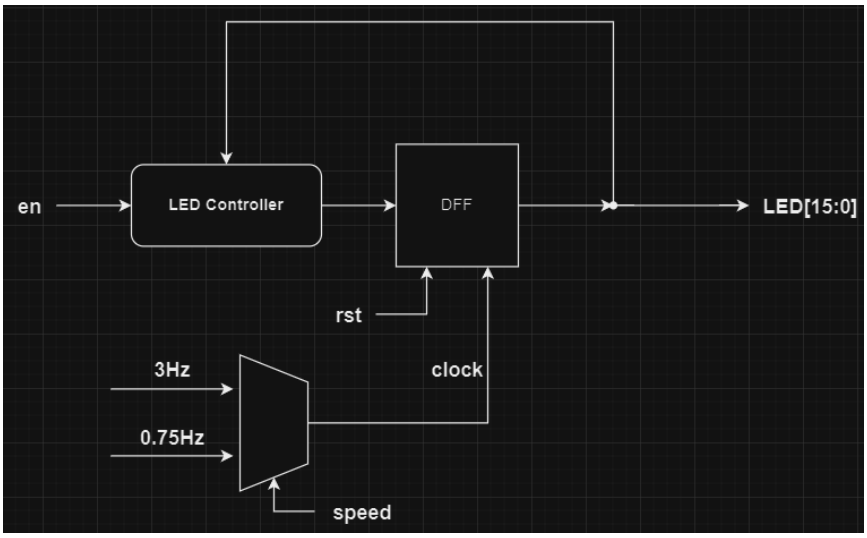
Lab 3

A. Lab Implementation

Lab3_1 Block Diagram



Lab3_1 用了一個簡單的 sequential circuit 的架構去實現，比較特別的部分是 clock 的部分是利用了 clock divider 來自定 clock 的頻率，clock divider 的實現是參考題目附件提供的。這裏是根據題目要求設置了一個快，一個慢的 clock 去 run LEDs，至於 LED Controller 的部分就是根據 LED 的狀態去決定下一個 edge 的狀態爲何。另外一個比較要注意的重點是 asynchronous reset，所做到的就是在 always block 的 sensitivity list 上表示 rst signal。

下圖的 code 就是 LED Controller 的實踐，先是檢測 rst signal，再來是檢測 en signal，en 被 triggered 的時候，LED 就會根據當下的 clock 在每一次的 edge triggered 顯示對應的圖案，否則，LED 會保持當下的狀態直到 en 被 triggered。

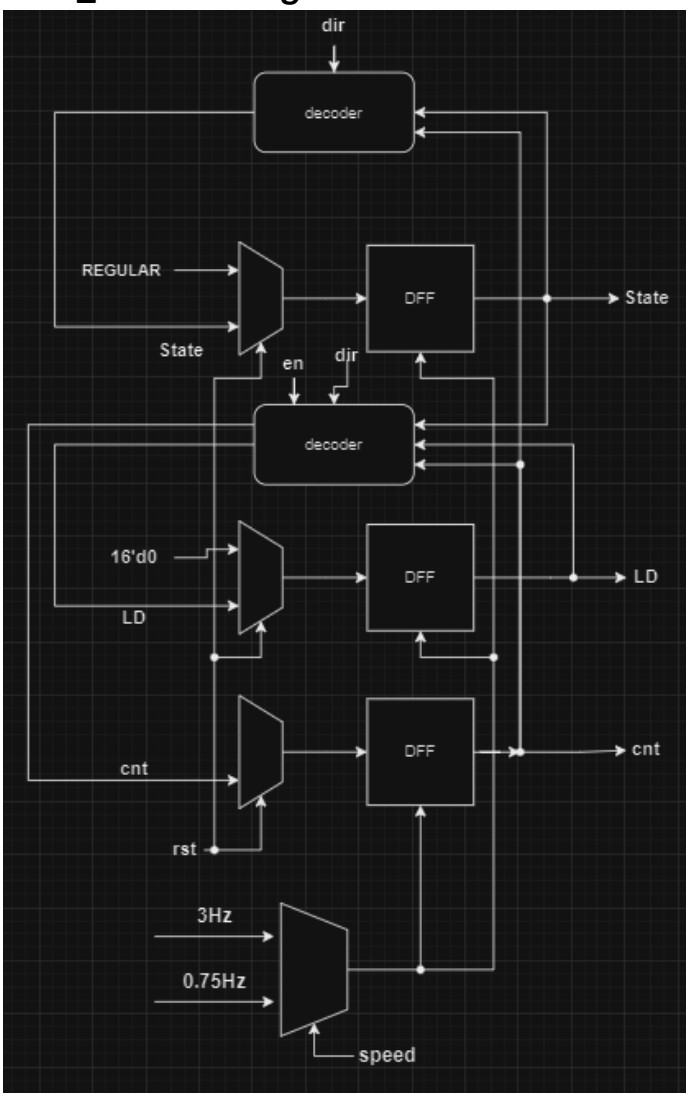
```
always @(posedge clk_div_now or posedge rst) begin
    if(rst)
        LD <= 16'b0;
    else begin
        if(en) begin
            if(LD == 16'hffff)
                LD <= 16'b0;
            else if(LD == 16'd0)
                LD <= 16'b1000100010001000;
            else if(clk_div_now)
                LD <= {1'b1, LD[15:13], 1'b1, LD[11:9], 1'b1, LD[7:5], 1'b1, LD[3:1]};
        end
        else
            LD <= LD;
    end
end
```

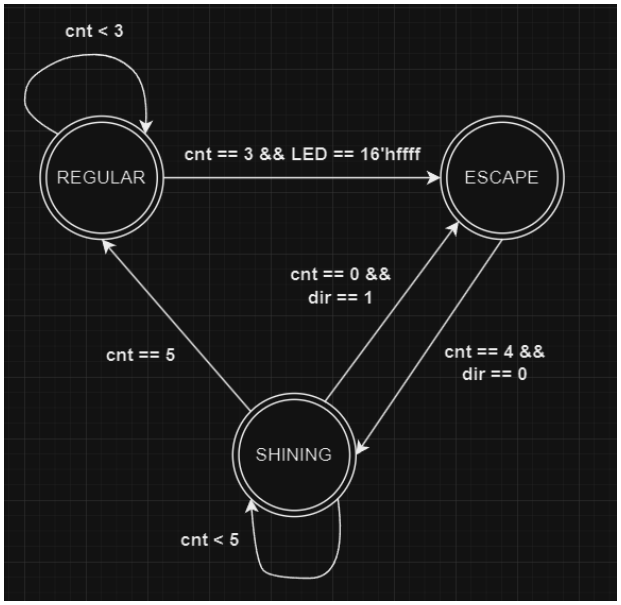
```
module clock_divider
    #(parameter n = 25)(
    input clk,
    output clk_div
    );

    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
```

```
clock_divider #(27) slow(.clk(clk), .clk_div(clk_div_s));
clock_divider #(25) fast(.clk(clk), .clk_div(clk_div_f));

assign clk_div_now = (speed) ? clk_div_f : clk_div_s;
```

上圖以及左圖是 clock divider 以及 module assign 當下 clock 的方式（根據 speed signal 來決定），lab3 的 clock divider 皆是如此。

Lab3_2 Block Diagram





Lab3_2 的設計是利用了 FSM 的方式，首先我設定了三個 state，在每一個 state 達成條件了之後便會跑到下一個 state 去執行，根據題目的要求，我設計了一個變數 cnt 來記錄在當下 state 所執行的動作次數，一旦達標，他就會發生 state transition。

在 REGULAR 狀態裏，當他完成三次的 loop 時候并且確保 LEDs 是全部亮的，就會進入下一個 state。在 ESCAPE 的時候，題目要求是根據 dir signal 來決定方向並依照 clock edge 依序兩兩熄滅/點亮 LEDs，因爲總共有 16 個 LED，所以我利用 cnt==8/0 和 dir signa 來決定他下一個 state 的去向。最後在 SHINING 狀態，當他完成五次的 loop，就會回去 REGULAR。

比較重要的部分是在於如何去計算他完成 cnt 的次數，下圖就是我去實現的部分。

```
else if(LD == 16'b0) begin
    LD <= 16'b1000100010001000;
    cnt <= cnt + 1'b1;
end
```

REGULAR state 的計數是當 LED 全部熄滅時，下一次的動作就會 cnt 加一。LED 的動作方式與 Lab3_1 一樣。

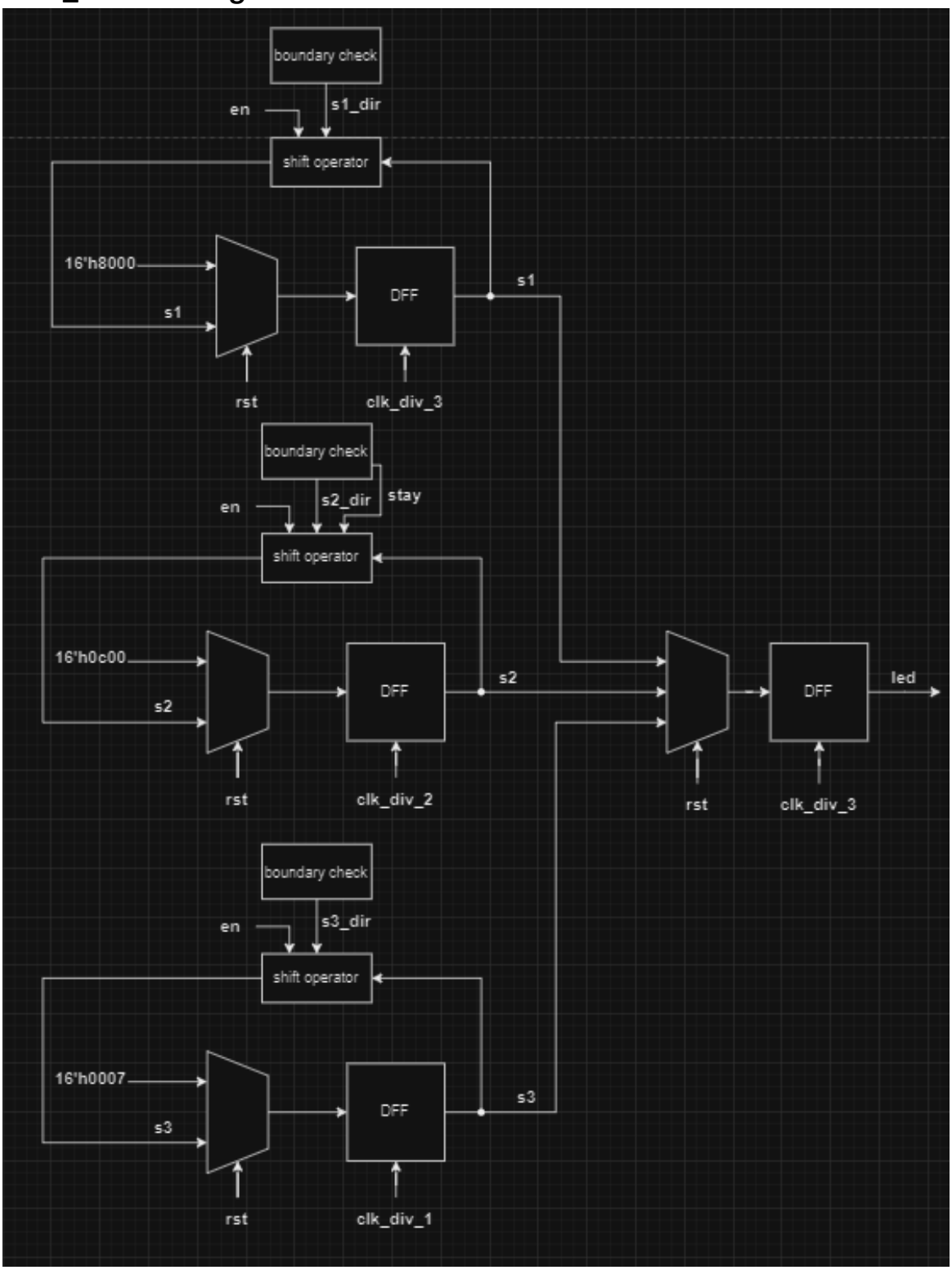
```
if(!dir) begin
    if(cnt == 4'd8) begin
        LD <= 16'd0;
        cnt <= 4'd0;
    end
    else begin
        LD <= {2'b0, LD[15:2]};
        cnt <= cnt + 1'b1;
    end
end
else begin
    if(cnt == 4'd0) begin
        LD <= 16'd0;
        cnt <= 4'd0;
    end
    else begin
        LD <= {LD[13:0], 2'b11};
        cnt <= cnt - 1'b1;
    end
end
```

ESCAPE state 的計數是依據 dir signal 的值，因爲 LEDs 初始是全亮的，如果 dir 是向右熄滅，cnt 就會 increment，如果是向左點亮，那就會 decrement。

```
if(LD == 16'd0)
    LD <= 16'hffff;
else begin
    LD <= 16'd0;
    cnt <= cnt + 1'b1;
end
```

SHINING state 是當 LEDs 全熄滅時就會 cnt 加一。

Lab3_3 Block Diagram



Lab3_3 的是設計使用了三個不同的 clock，我把三個個別的 snake 用 s1，s2 和 s3 去記錄他們個別的狀態，每一個 snake 也都有自己的 clock 去 trigger。

每一個 snake 都會根據他們自身的 dir signal（我用 s1_dir，s2dir，s3_dir）去決定他會往哪一個方向做 shift operation，dir signal 則是會依據他們個別的 boundaries checking 去決定。比較特別的有 snake11，因爲他會有被兩邊夾到的情況出現，所以我用了 stay 這個 signal 去讓我的 snake11 決定說要不要逗留一個 clock cycle。最後顯示的部分，我是用了最高頻率的 clock 去做 trigger，將 s1，s2 和 s3 的狀態 OR 起來，并顯示到 LEDs。

```
always @(posedge clk_div_3 or posedge rst) begin
    if(rst) begin
        s1_dir <= 1'b1;
    end
    else begin
        if(s1[14] == 1'b1)
            s1_dir <= 1'b1;
        else if(((s1 >> 2) & s2) && (s1_dir == 1'b1))
            s1_dir <= 1'b0;
        else
            s1_dir <= s1_dir;
    end
end
```

這個是我 snake1 檢查 boundary 的部分，snake11 和 snake111 用的方式也是一樣。我會根據用 snake 現在的狀態去做 AND operation，要是有重疊的地方出現就會讓 if-else statement 不成立並讓 dir signal 變換，再來如果是 LED[15]/[0]的部分則是直接檢測當下的狀態是否為 1，若是則 dir signal 變換。

```
always @(posedge clk_div_3 or posedge rst) begin
    if(rst)
        LD <= 16'h8c07;
    else begin
        LD <= s1 | s2 | s3;
    end
end
```

這一 part 就是用最高頻率的 clock 來 trigger LEDs 顯示的部分，每一次的 trigger 都會把三個 snake 的狀態 OR 起來。

B. Questions and Discussions

a. 我用的方法就是在 always block 的 sensitivity list 裏面加上 posedge rst，讓這個 block 除了會在 clock 的 posedge trigger，也會在 rst posedge trigger，所以只要 rst 有被 triggered 到，他的優先序就會比其他 input signal 高。

b. 如果要讓每個 snake 去 react 每一次的碰撞，那樣的話就必須使用最高頻率的 clock 來作爲每一個 snake 的 clock signal，只有這樣，在每一次的碰撞檢測才會去更新每一條 snake 碰撞的狀態，而不是需要先確定那條蛇是否已經被 triggered，這樣的方式就可以確保讓他在碰撞之後就馬上可以對 direction signal 作變換。

C. Problem Encountered

我所遇到的問題就是 Lab3_3 中的碰撞檢測，我無法做到及時檢測並更新我的 direction signal，我的 snake 有時會在碰撞之後，延遲一個 cycle 才做出變換 direction 的部分，而有時候則會提早做變換的動作，我用的解決方法是提早去判斷說如果下一次的 shift operation 會不會讓兩條 snake 之間接觸到，若是會，則它會在下一次的 clock cycle 去做 direction 變換，從而達到碰撞后就變向的行爲。

```
if(s1[14] == 1'b1)
    s1_dir <= 1'b1;
else if(((s1 >> 2) & s2) && (s1_dir == 1'b1))
    s1_dir <= 1'b0;
else
    s1_dir <= s1_dir;
```

D. Suggestions

對於 lab 有一個小小的提議是，若後面的 lab 有像這次 lab3_3 在碰撞的時候處理 policy 的部分會比較 tricky/challenging 的話，可以在 lab 結束之後提供一個解題思路/解法推薦等等的東西。:>