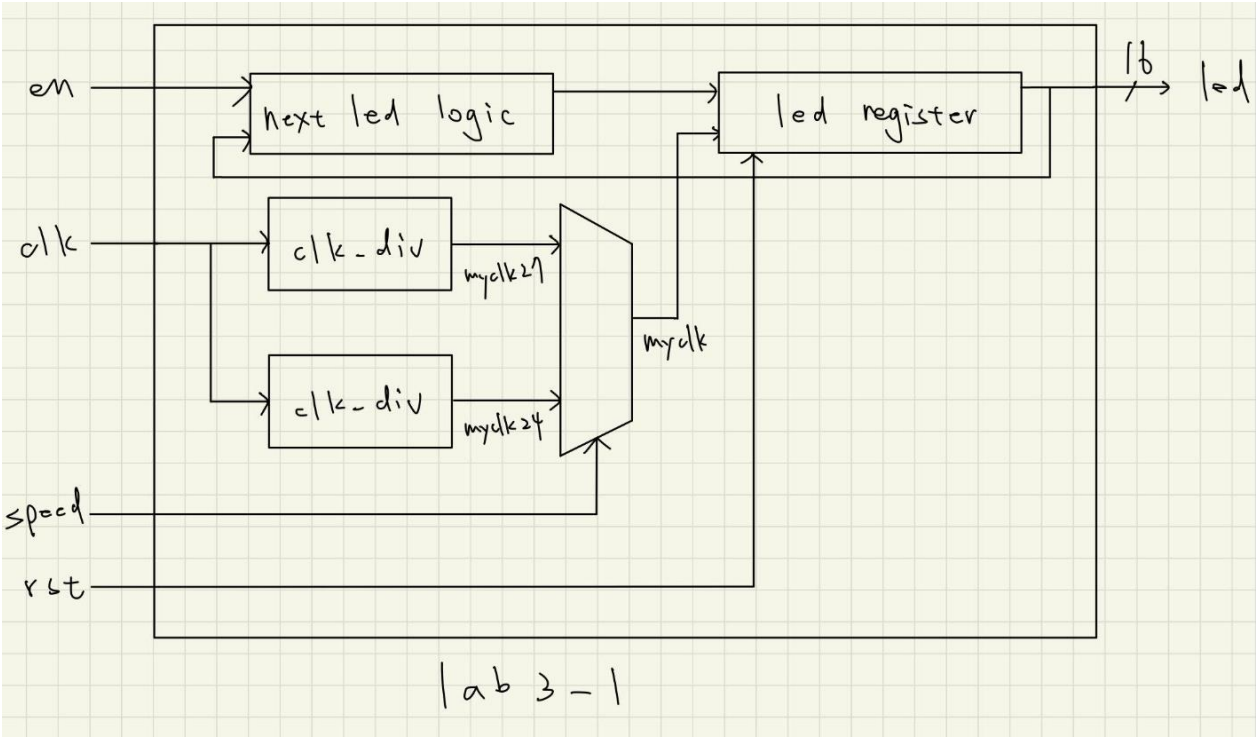
Lab 2

學號: 110062131 姓名: 馬毓昇

A. Lab Implementation

Lab3_1: 可以用 clock_divider 調整速率的 LED 跑馬燈。



Lab3_1 segment:

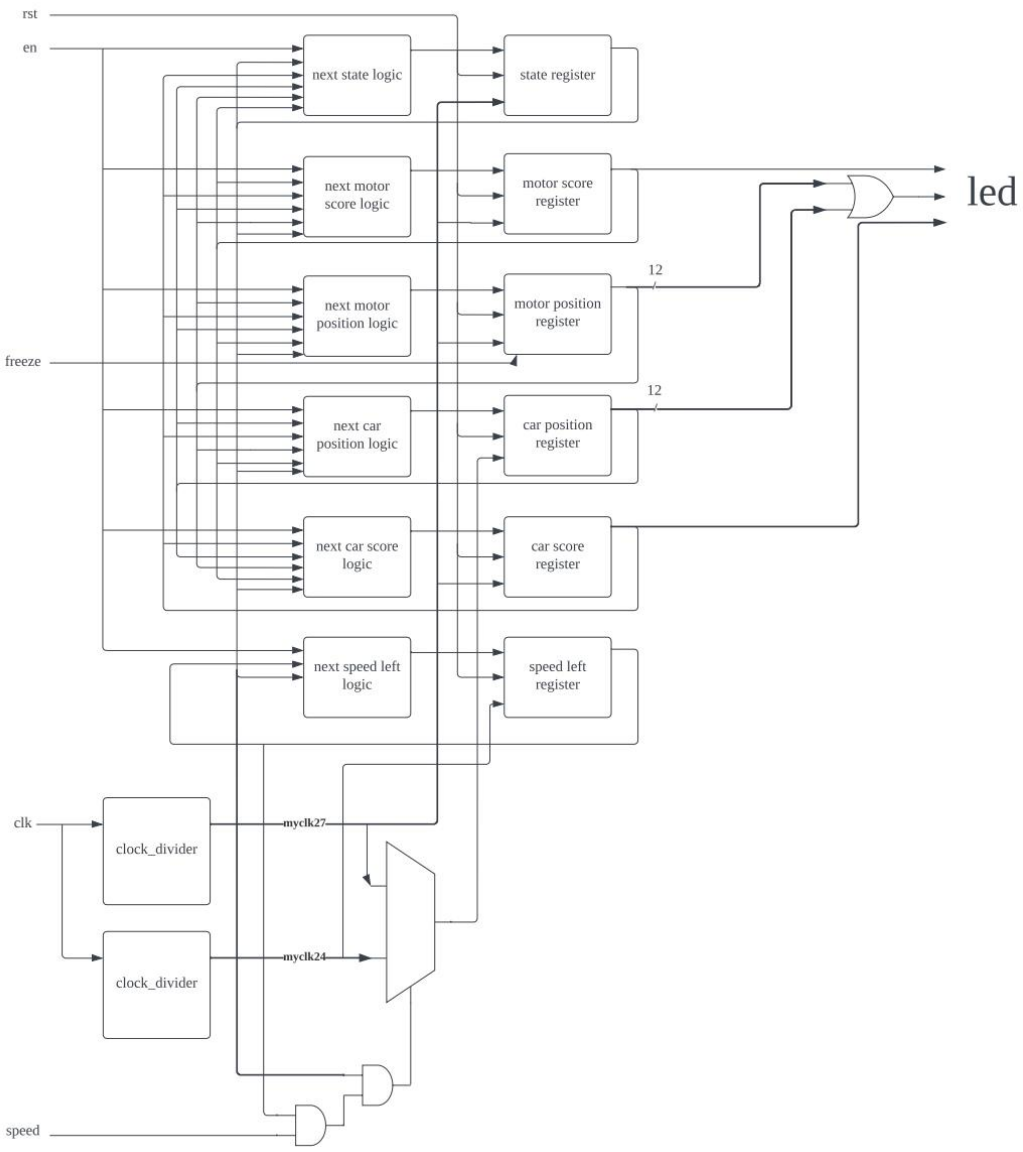
1. clock_divider

```
55  module clock_divider #(parameter n = 25)
56  (
57      input clk,
58      output clk_div
59  );
60
61  reg [n-1:0] num = 0;
62  wire [n-1:0] next_num;
63
64  always @(posedge clk) begin
65      num <= next_num;
66  end
67
68  assign next_num = num + 1;
69  assign clk_div = num[n-1];
70
71  endmodule
```

2. 用三元運算子來改變 clk 的頻率，也就是 led 燈的更新速度

```
28  assign myclk = speed ? myclk27 : myclk24;
29
30  always @(posedge myclk or posedge rst) begin
31      if(rst) begin
32          led <= 16'b1000000000000000;
33      end
34      else begin
35          led <= next_led;
36      end
37  end
38
39  always @* begin
40      if(en == 0) begin
41          next_led = led;
42      end
43      else begin
44          if(led == 16'b0000000000000001) begin
45              next_led = 16'b1000000000000000;
46          end
47          else begin
48              next_led = led >> 1;
49          end
50      end
51  end
```

Lab3_2: 一個 LED 賽車遊戲，配有記分板顯示以及 freeze 跟 speed up 兩個特殊功能。



Lab3_2 segment:

1. 7 states，INIT_WAIT 做 dummy state 來實現勝利後新比賽一開始的等待

```
27  parameter INIT = 3'd0;
28  parameter RACING = 3'd1;
29  parameter M_F = 3'd2;
30  parameter M_W = 3'd3;
31  parameter C_F = 3'd4;
32  parameter C_W = 3'd5;
33  parameter INIT_WAIT = 3'd6;
34  reg [2:0] state = 3'd0, next_state;
```

2. 用 concat 的方式實作 led 接線

```
49  assign led = {c_score, c_pos[13:2] | m_pos[13:2], m_score};
```

3. 車子移動：使用位元運算子”>>”來模擬車子往右走(以 motor 為例，car 以此類推)

```
// motor pos
RACING : begin
    if(freeze == 1) begin
        m_next_pos = m_pos;
    end
    else if(m_pos[2] != 1'b1 && c_pos[2] != 1'b1) begin
        m_next_pos = m_pos >> 1;
    end
    else begin
        m_next_pos = m_pos;
    end
end
```

4. freeze 功能：freeze==1 時讓 car position 不變(見上圖)

5. speed 功能：全部的更新(state, motor position, motor score…)都用 myclk27 來更新。只有 car position 與 speed_left 用 myclk，myclk 則是用三元運算子依 speed_left 狀態(小於 6 大於 0 時，共 5 個 cycle)來決定要接入 myclk24 或 myclk27，來達成加速的效果。(state == RACING 說明只有在賽跑時才能提速，其他狀態(INIT, FINISH, WIN)car position 皆只能用 myclk27 更新，如此一來便不會影響到其他狀態的時序)

```
184 assign myclk =
185     ((speed_left < 6) && (state == RACING) && (speed_left > 0)) ? myclk24 : myclk27;
```

```
212 // car pos
213 always @(posedge myclk or posedge rst) begin
214     if(rst) begin
215         c_pos <= 16'b0011000000000000;
216     end
217     else begin
218         c_pos <= c_next_pos;
219     end
220 end
```

6. speed_left 用 myclk24 來算，初始化為 6，到 0 停止，中間共經過 5 個 cycle。RACING 時每 speed 一 cycle 減一，如果回到 speed == 0，依舊把 speed_left 減到 0 為止。可以實現 speed 只要是 HIGH 一次就一定會加速滿 5 個 cycle(不論加速中是否把 speed 變回 LOW)與一回合只能用一次的效果。

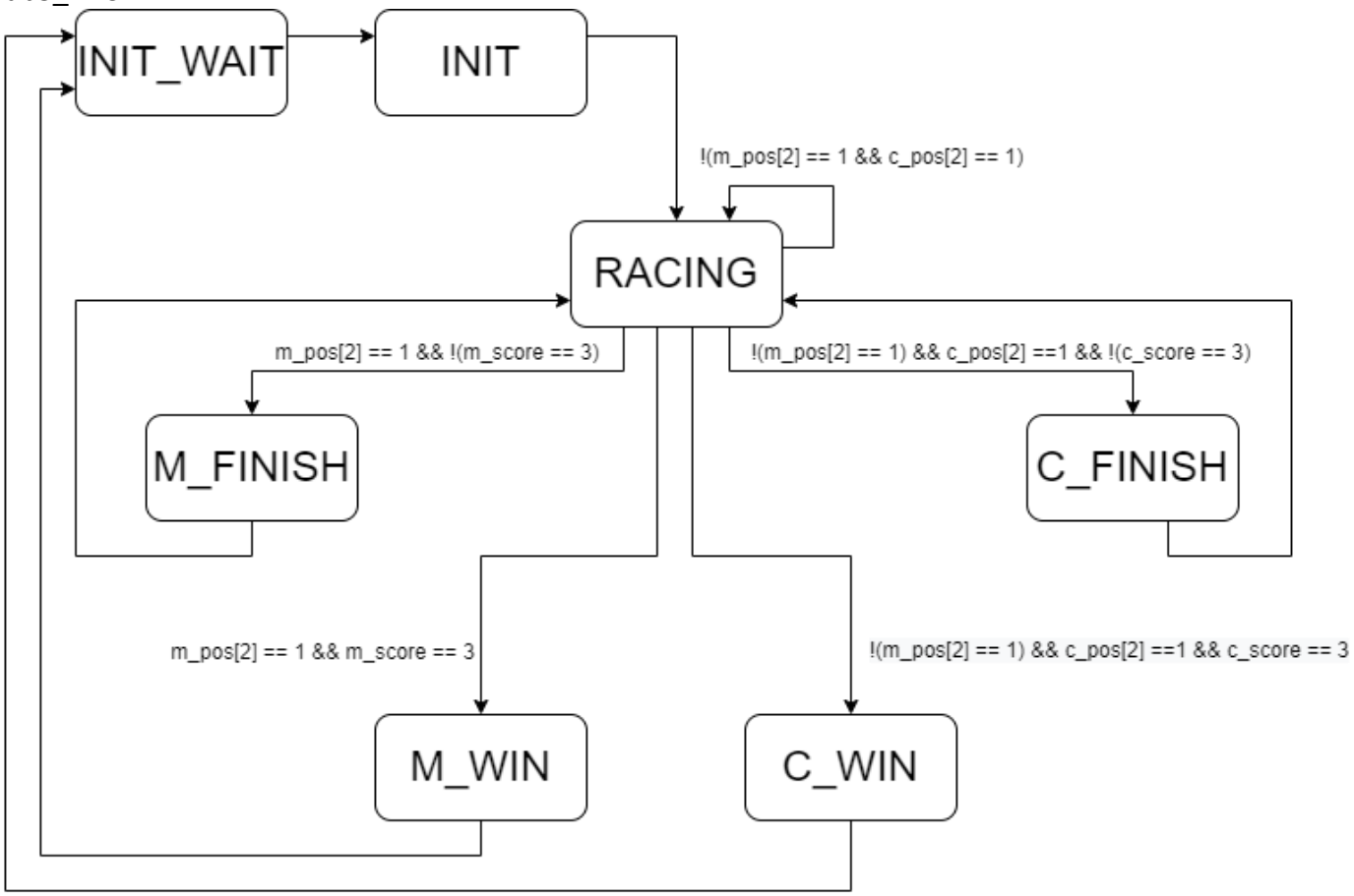
```
// speed left
INIT, INIT_WAIT : begin
    speed_left <= 3'd6;
end
RACING : begin
    if(en == 0) begin
        speed_left <= speed_left;
    end
    else if(speed == 1 && speed_left > 0) begin
        speed_left <= speed_left - 1;
    end
    else if(speed_left != 3'd6 && speed_left > 0) begin
        speed_left <= speed_left - 1;
    end
end
```

7. 算分，簡單的 counter，car score 進 car finish 時加一，motor 同理

```
// motor score
case(state)
    INIT : begin
        m_next_score = 2'b00;
    end
    RACING : begin
        m_next_score = m_score;
    end
    M_F : begin
        m_next_score = m_score + 1;
    end
    C_F : begin
        m_next_score = m_score;
    end
    M_W : begin
        m_next_score = 2'b11;
    end
    C_W : begin

        m_next_score = 2'b00;
    end
endcase
```

Lab3_2 FSM:



1. INIT -> RACING:
   進 INIT 後的下個 cycle 直接進 RACING。
2. RACING -> FINISH(M_F, C_F):
   當各自到達終點(摩托車燈到 led[2]or 車子燈到 led[2])而自己又還沒贏 3 場就進 FINISH。
3. FINISH(M_F. C_F) -> RACING:
   進 FINISH 後的下個 cycle 直接進 RACING。
4. RACING -> WIN(M_W, C_W):
   當達成各自勝利條件(摩托車到終點且已贏 3 場 or 車子到終點且已贏 3 場)就進 WIN。
5. WIN(M_W, C_W) -> INIT_WAIT:
   進 WIN 後的下個 cycle 直接進 INIT_WAIT。
6. INIT_WAIT -> INIT:
   進 INIT_WAIT 後的下個 cycle 直接進 INIT。5, 6 搭配實現勝利後新比賽的等待。

B. Que stions and Discussions

A: 做一個 counter(叫 freeze_left)存 freeze 剩餘的 cycle 數，每一輪初始化 freeze_left = 3，比賽中 freeze == 1 時就每一 cycle 把 freeze_left 減一，如果 freeze_left <= 0 就覆蓋掉 freeze 的影響。

```
reg [1:0] freeze_left;

// freeze left
always @(posedge myclk27 or posedge rst) begin
    if(rst) begin
        freeze_left <= 2'd3;
    end
    else begin
        case(state)
            INIT, M_F, C_F, M_W, C_W : begin
                freeze_left <= 2'd3;
            end
            RACING : begin
                if(freeze == 1 && freeze_left > 0) begin
                    freeze_left <= freeze_left - 1;
                end
            end
        endcase
    end
end
// motor pos update
RACING : begin
    if(freeze == 1 && freeze_left > 0) begin
        m_next_pos = m_pos;
    end
    else if(m_pos[2] != 1'b1 && c_pos[2] != 1'b1) begin
        m_next_pos = m_pos >> 1;
    end
    else begin
        m_next_pos = m_pos;
    end
end
```

B: 我是做 method 2，method 1 我會用 myclk24 + delay_counter 的方式設計，讓 delay_counter 以 myclk24 更新，而其他部分(led control、car、motor)則會在 myclk24 && delay_counter == 3'b000 時更新。至於車子加速的部分，則是把 delay_counter 的判斷拿掉就可以了。

delay_counter

```
49  reg [2:0] delay_counter;
50  reg [2:0] next_delay_counter;
```

myclk24

```
183 // delay_counter
184 always @(posedge myclk24 or posedge rst) begin
185     if(rst) begin
186         delay_counter <= 0;
187     end
188     else begin
189         delay_counter <= next_delay_counter;
190     end
191 end
192
193 always @* begin
194     if(en == 0) begin
195         next_delay_counter = delay_counter;
196     end
197     else begin
198         next_delay_counter = (delay_counter + 1) % 8;
199     end
200 end
```

如果 delay_counter==0 就更新，不然就 hold 住

車子加速把 delay_counter 判斷拿掉

```
// car pos
RACING : begin
    if(speed == 0 || speed_ok > 3'd4) begin
        if(m_pos[2] != 1'b1 && c_pos[2] != 1'b1 && delay_counter == 3'b000) begin
            c_next_pos = c_pos >> 1;
        end
        else begin
            c_next_pos = c_pos;
        end
    end
    else begin
        if(m_pos[2] != 1'b1 && c_pos[2] != 1'b1) begin
            c_next_pos = c_pos >> 1;
        end
        else begin
            c_next_pos = c_pos;
        end
    end
end
```

C. Problem Encountered

1. 最初一畫完 FSM 圖就直接把整份 code 打出來，但做出來的燈號呈現一個混亂的狀態，根本無從 debug。後來把整份 code 都刪掉，從最簡單的幾個功能開始實作：單純的 motor 由左而右、motor 到終點分數加一與復位、motor 得到 4 分進勝利狀態。確保上述這三個功能與燈號是正常的之後，再一步步補上 car 的部分與 freeze 和最難的 speed。依照這樣的步調慢慢實作花不到幾個小時做成功了，反觀我前一天一股腦地做，花了一整晚都沒做出來，算是學到了一個經驗。

2. FSM clocking 一開始是嘗試做 method 1 的，但發現會有兩個問題：第一，無論如何 car 的動作都對不上 motor，我想應該是因為我的實作方式是只有 car 跑 myclk24，其餘全部還是跑 myclk27，因此 delay_counter 更新(依據 myclk24)出來的慢 8 倍的 myclk27 都會比實際上的 myclk27 還要再慢幾個 myclk24，所以應該要整份 code 都用同樣的 myclk24 + delay_counter 邏輯才對。

3. 延續上面，第二，en=0 再 en=1 會很明顯看得出來 car 相對 motor 的時序變調了，這問題很明顯就出在我原本的做法 delay_counter 是用 myclk24 來更新的，相較於比較慢的 myclk27，如果在 myclk27 是 HIGH 或 LOW 的過程中把 en 回復到 1，delay_counter(依據 myclk24)就會偷跑更新，解決辦法應該也是跟第一一樣，整份 code 都要用 myclk24 + delay_counter 的邏輯才行。

原本實作的錯誤方式，除了 car 與 delay_counter 都是用 myclk27，後來改做 method 2 了：

delay_counter

```
49  reg [2:0] delay_counter;
50  reg [2:0] next_delay_counter;
```

myclk27

```
105 // motor pos
106 always @(posedge myclk27 or posedge rst) begin
107     if(rst) begin
108         m_pos <= 16'b0000010000000000;
109     end
110     else begin
111         m_pos <= m_next_pos;
112     end
113 end
```

myclk24

```
// delay_counter
always @(posedge myclk24 or posedge rst) begin
    if(rst) begin
        delay_counter <= 0;
    end
    else begin
        delay_counter <= next_delay_counter;
    end
end

always @* begin
    if(en == 0) begin
        next_delay_counter = delay_counter;
    end
    else begin
        next_delay_counter = (delay_counter + 1) % 8;
    end
end
```

如果 delay_counter==0 就更新，不然就 hold 住

```
RACING : begin
    if(speed == 0 || speed_ok > 3'd4) begin
        if(m_pos[2] != 1'b1 &&
           c_pos[2] != 1'b1 &&
           delay_counter == 3'b000) begin
            c_next_pos = c_pos >> 1;
        end
        else begin
            c_next_pos = c_pos;
        end
    end
```

D. Suggestions

這次 lab 沒有 practice 太難了，希望以後 lab 都能盡量有 practice code 讓我們參考。