

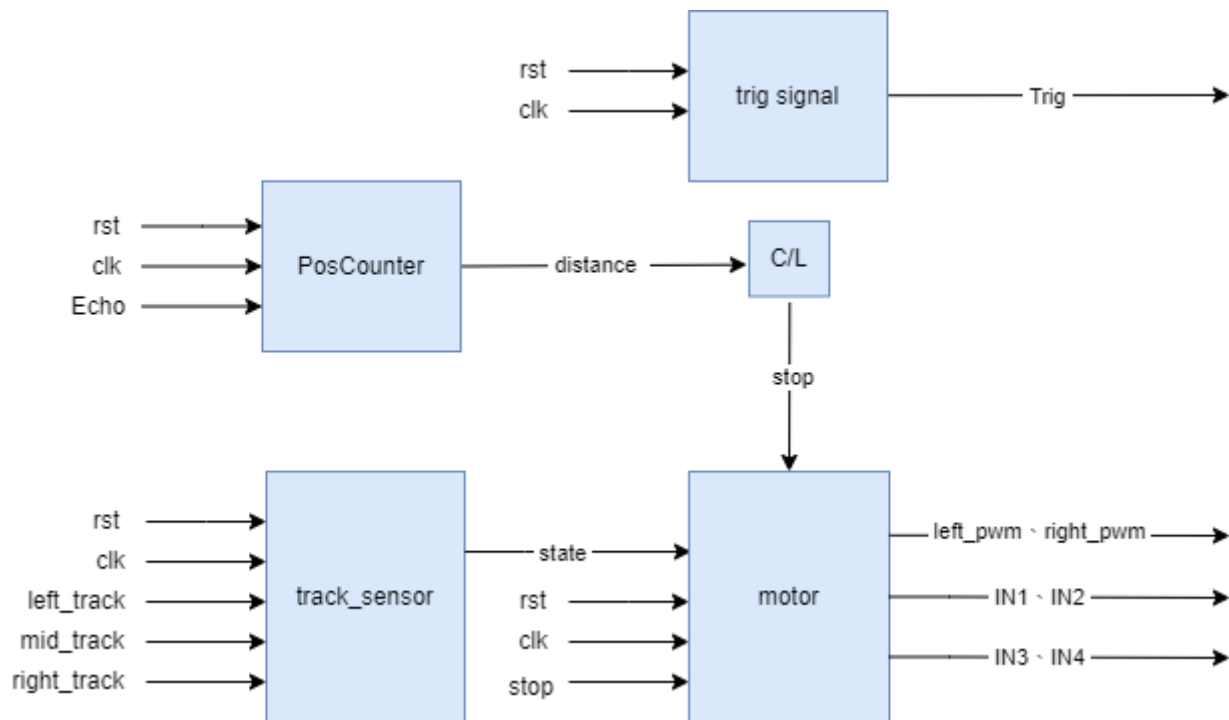
Lab 8

學號: 109062173

姓名: 葉昱揚

A. Lab Implementation

1. Block Diagram of the design with explanation (10%)



(圖 1) Lab8 Block Diagram of the design

上圖為 lab8 block diagram，使用到的 module 為 track_sensor、motor、PosCounter、trig_signal。

track_sensor 接收三個 input，left_track、mid_track、right_track，分別代表左中右 IR sensor 感應到的顏色。並依據顏色不同設計不同的 state。

motor 接收 track_sensor 設計好的 state，並依據 state 做出相對應的處理。不同的 state 會讓左右馬達轉向、轉速不同。

trig_signal 的 output trig 會傳給 ultrasonic sensor，使其正確運作。

PosCounter 的 echo 讓我們知道前方是否有障礙物擋在前面。這邊會利用 echo 計算障礙物與車子距離 distance，經過 C/L 後變成 stop signal，控制車子是否需要停住。

2. Partial code screenshot with the explanation (35%)

a. PWM calculation

```
1 // speed state
2 parameter speed_stop    = 10'd0;
3 parameter speed_fast    = 10'd750;
4 parameter speed_fast_2  = 10'd765;
```

(圖 2) speed design

本次 Lab 中我只設計了兩種速度模式: stop 和 fast。speed_fast_2 本質上與 speed_fast 相同。因為其中一邊馬達比較沒力，若兩邊馬達速度一樣，車子無法順利直直往前進，會偏向某一邊。因此讓較弱邊的馬達速度快一點，以達到往前進的效果。

```
1 //generate PWM by input frequency & duty cycle
2 module PWM_gen (
3     input wire clk,
4     input wire reset,
5     input [31:0] freq,
6     input [9:0] duty,
7     output reg PWM
8 );
9     wire [31:0] count_max = 100_000_000 / freq;
10    wire [31:0] count_duty = count_max * duty / 1024;
11    reg [31:0] count;
12
13    always @(posedge clk, posedge reset) begin
14        if (reset) begin
15            count <= 0;
16            PWM <= 0;
17        end else if (count < count_max) begin
18
19            count <= count + 1;
20            // TODO: set <PWM> accordingly
21            if(count < count_duty) begin
22                PWM <= 1'b1;
23            end
24            else begin
25                PWM <= 1'b0;
26            end
27        end else begin
28            count <= 0;
29            PWM <= 0;
30        end
31    end
32 end
33 endmodule
```

(圖 3) PWM_gen

圖 2 的 speed 是 PWM_gen 的 duty input signal。duty 會控制 counter 數數的上限。如上圖 line21~line26，在數到上限之前，output PWM 為 1，超過上限後為 0。也就是說，如果 duty cycle 越大，output PWM 為 1 的時間越長，馬達轉動越有力。

b. Tracker sensor reading



```

1 // state
2 parameter car_forward = 2'd0;
3 parameter car_stop    = 2'd1;
4 parameter car_left    = 2'd2;
5 parameter car_right   = 2'd3;
6
7 //dir
8 parameter dir_straight = 2'd0;
9 parameter dir_stop    = 2'd1;
10 parameter dir_left    = 2'd2;
11 parameter dir_right   = 2'd3;

```

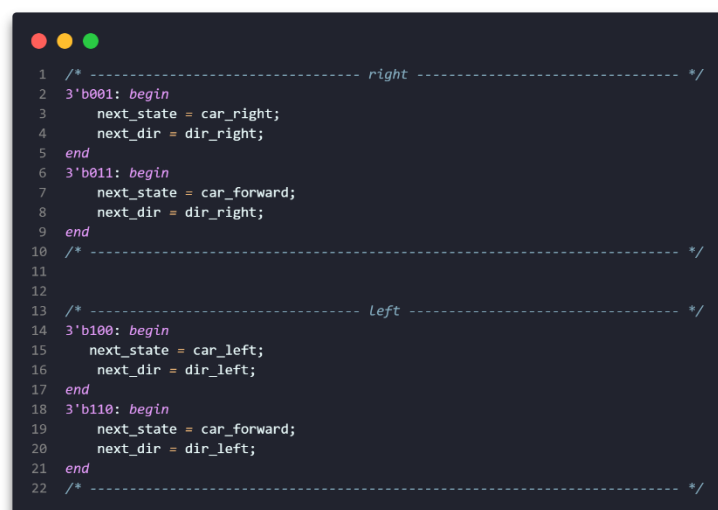
(圖 4) 車子的行為和轉向

車子有四種狀態：

往前 car_forward、停住 car_stop、左轉 car_left、右轉 car_right
 這幾種狀態會 output 給 motor module，讓馬達做處對應的動作

有四個轉向狀態：

往前 dir_straight、停住 dir_stop、左轉 dir_left、右轉 dir_right
 設計轉向狀態是因為賽到中需要左轉或右轉，我需要利用 3 個 IR sensor 給我的資訊，
 預測未來轉向哪個方向，預測資訊即是使用 dir，以四個狀態紀錄。



```

1 /* ----- right ----- */
2 3'b001: begin
3   next_state = car_right;
4   next_dir   = dir_right;
5 end
6 3'b011: begin
7   next_state = car_forward;
8   next_dir   = dir_right;
9 end
10 /* ----- */
11
12 /* ----- left ----- */
13 3'b100: begin
14   next_state = car_left;
15   next_dir   = dir_left;
16 end
17 3'b110: begin
18   next_state = car_forward;
19   next_dir   = dir_left;
20 end
21 end
22 /* ----- */

```

(圖 5) 左轉、右轉

我的設計比較特殊，車子放在軌道內，當有一顆 IR sensor 壓在黑線上時會前進：

圖 5 的 line2~line5，input 為 001(黑黑白)時，車子的 state 為 car_right 往右走，dir 為右轉。
 圖 5 的 line6~line9，input 為 011(黑白白)時，車子的 state 為 car_forward 往前走，dir 為右轉。
 因為只有在車子順時鐘繞地圖時才會讀到 011，每逢直角以外的轉彎時必為右轉。

圖 5 的 line14~line17，input 為 100(白黑黑)時，車子的 state 為 car_left 往左走，dir 為左轉。
 圖 5 的 line18~line21，input 為 110(白白黑)時，車子的 state 為 car_forward 往前走，dir 為左轉。
 因為只有在車子逆時鐘繞地圖時才會讀到 110，每逢直角以外的轉彎時必為左轉。

```

1  /* ----- 直角轉彎 ----- */
2  3'b111: begin
3      next_dir = dir;
4      if(dir==dir_right) begin
5          next_state = car_left;
6          next_dir = dir_left;
7      end
8      else if(dir==dir_left) begin
9          next_state = car_right;
10         next_dir = dir_right;
11     end
12     else next_state = state;
13 end
14 /* ----- */

```

(圖 6) 直角轉彎

直角轉彎只會發生在 111 (白白白)。這邊需要用 dir 來判斷要往左直角 or 右直角轉彎

當 dir 為 dir_right 時，車子會左直角轉彎。

因為 dir==dir_right，代表車子在順時鐘繞地圖，而順時鐘繞地圖只會遇到左直角轉彎

當 dir 為 dir_left 時，車子會右直角轉彎。

因為 dir==dir_left，代表車子在逆時鐘繞地圖，而逆時鐘繞地圖只會遇到左直角轉彎

```

1  default begin
2      if(dir==dir_left) begin
3          next_state = car_left;
4          next_dir = dir_left;
5      end
6
7      else if(dir==dir_right) begin
8          next_state = car_right;
9          next_dir = dir_right;
10     end
11     else begin
12         next_state = state;
13         next_dir = dir;
14     end
15 end

```

(圖 7) 其餘情況

其餘其況會根據 dir 來決定車子的行動。

如上圖，dir_right、dir_left、dir_straight、dir_stop 可以輕鬆決定車子的行動。

c. Movement policy and corresponding motor control

```

1  always@(*) begin
2      if(stop) begin
3          {next_motorB_speed, next_motorA_speed} = {speed_stop, speed_stop};
4      end
5      else begin
6          case(mode)
7              car_left    : { next_motorB_speed, next_motorA_speed } = { speed_stop , speed_fast };
8              car_right   : { next_motorB_speed, next_motorA_speed } = { speed_fast , speed_stop };
9              car_forward : { next_motorB_speed, next_motorA_speed } = { speed_fast_2 , speed_fast };
10             car_stop    : { next_motorB_speed, next_motorA_speed } = { speed_stop , speed_stop };
11             default: begin
12                 next_motorB_speed = speed_stop;
13                 next_motorA_speed = speed_stop;
14             end
15         endcase
16     end
17 end

```

(圖 8) 馬達速度

motorB_speed 為左馬達 duty 、motorA_speed 為右馬達 duty。

若前方有障礙物，stop signal 為 1，兩顆馬達都不轉動。

若左轉(car_left) ，左馬達停住、右馬達轉動。

若右轉(car_right) ，左馬達轉動、右馬達停住。

若前進(car_forward)，左馬達轉動、右馬達轉動。

若停止(car_stop) ，左馬達停住、右馬達停住。

```

1  always @(*) begin
2      if(stop) begin
3          {motorB,motorA} = {motor_off , motor_off};
4      end
5      else begin
6          case(mode)
7              car_left    : { motorB, motorA } = { motor_off , A_forward };
8              car_right   : { motorB, motorA } = { B_forward , motor_off };
9              car_forward : { motorB, motorA } = { B_forward , A_forward };
10             car_stop    : { motorB, motorA } = { motor_off , motor_off };
11             default     : { motorB, motorA } = { motor_off , motor_off };
12         endcase
13     end
14 end
15 end
16 end

```

(圖 9) 輪胎轉向

motorB 為左輪胎轉向 、motorA 為右輪胎轉向。

若前方有障礙物，stop signal 為 1，兩顆輪胎都不轉動。

若左轉(car_left) ，左輪胎停住 、右輪胎向前轉。

若右轉(car_right) ，左輪胎向前轉、右輪胎停住 。

若前進(car_forward)，左輪胎向前轉、右輪胎向前轉。

若停止(car_stop) ，左輪胎停住 、右輪胎停住 。

d. Sonic sensor control (pulse, echo)

```

1 // count 10us to set <trig> high and wait for 100ms, then set <trig> back to low
2 always @(*) begin
3     next_trig = trig;
4     next_count = count + 1;
5     // TODO: set <next_trig> and <next_count> to let the sensor work properly
6
7     // 1s : 10^8 cycle
8     // 10s : 10^9 cycles
9     // 10us: 10^3 cycles
10    if(count==10*3-1) begin
11        next_trig = 0;
12    end
13    // 100s : 10^10 cycles
14    // 100ms: 10^7 cycles
15    if(count==10*7-1) begin
16        next_count = 0;
17        next_trig = 1;
18    end
19 end

```

(圖 10) trig pulse

TODO: count 10us to set <trig> high and wait for 100ms, then set <trig> back to low

1s : 10^8 cycle

10s : 10^9 cycles

10us: 10^3 cycles

在 count 數到 10^3 的時候 (10us) 設定 trig 為 0。

100s : 10^{10} cycles

100ms: 10^7 cycles

在 count 數到 10^7 的時候 (100ms) 設定 trig 為 1 並重製 count。

```

1 assign stop = (distance <= 21'd30) ? 1'b1 : 1'b0;

```

(圖 11) stop signal

當偵測到車子前方 30 cm 有障礙物時，會傳出 stop 訊號，而 distance 的單位為 cm，故為 $dis \leq 21'd30$ 。

```

1 // TODO: trace the code and calculate the distance, output it to <distance_count>
2 assign distance_count = (distance_register / 21'd58);

```

(圖 12) 障礙物距離計算

空氣中的音速約為 342 (m/s)，換算過後得知每 29.1us 可移動 1cm。

假設 Echo 發射、回來的時間為 t ，單趟時間為 $\frac{t}{2}$ 。

如果前方有障礙物，障礙物與車子距離 = 時間 * 音速 = $\frac{t}{2} * \frac{1}{29.1} = \frac{t}{58}$ cm。

B. Questions and Discussions (40%)

A. How does the PWM_gen module generate the pulse for the motors? Please explain the implementation in the PWM_gen module. What will happen if the duty cycle is extremely high or extremely low? Will that affect the car's performance on the track (e.g., speed, smoothness, or anything else)? If yes, please explain how.

```

1  PWM_gen pwm_0 (
2      .clk(clk),
3      .reset(reset),
4      .freq(32'd25000),
5      .duty(duty),
6      .PWM(pmod_1)
7  );
8
9  module PWM_gen (
10     input wire clk,
11     input wire reset,
12     input [31:0] freq,
13     input [9:0] duty,
14     output reg PWM
15 );
16     wire [31:0] count_max = 100_000_000 / freq;
17     wire [31:0] count_duty = count_max * duty / 1024;
18     reg [31:0] count;
19
20     always @(posedge clk, posedge reset) begin
21         if (reset) begin
22             count <= 0;
23             PWM <= 0;
24         end else if (count < count_max) begin
25
26             count <= count + 1;
27             // TODO: set <PWM> accordingly
28             if(count < count_duty) begin
29                 PWM <= 1'b1;
30             end
31             else begin
32                 PWM <= 1'b0;
33             end
34         end else begin
35             count <= 0;
36             PWM <= 0;
37         end
38     end
39 end
40 endmodule

```

(圖 13) PWM_gen module

count_max 表示 counter 的最大值，根據給定的 freq = 25000 計算而得。

count_duty 表示達到 duty cycle 的 counter 值。

由於 count_max 為固定值，count_duty 的值由 duty signal 主導

always block 之中，counter 只要沒數到 count_duty，就 output PWM 為 1。反之 output PWM 為 0。

換個說法， PWM signal 持續為 1 的時間，由 duty 決定。

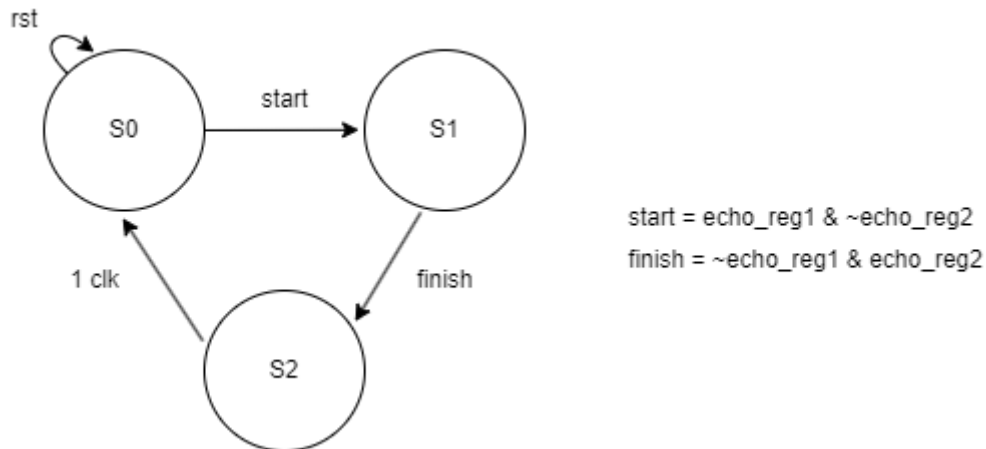
duty 大，PWM signal 持續為 1 的時間長。

duty 小，PWM signal 持續為 1 的時間短。

如果 duty cycle 非常高，馬達可能以接近最大速度運行，這可能導致電池供電不穩定，處於隨時會斷電的情況，或是馬達可能會受到過度加熱、電池過熱等等。

如果 duty cycle 非常低，最值觀的情況是馬達幾乎停止運動，導致車輛無法移動或以非常緩慢的速度行駛。

B. How does the state transition in the PosCounter module work? Draw a state diagram and explain how it works. You should describe the concept of events instead of listing the signals.



(圖 14) PosCounter state diagram

PosCounter 內部為 S0、S1、S2 狀態輪轉。

在 S1 時，counter 從 0 開始計數。跳到 S2 時，這個 counter 會被用來計算與障礙物的距離。

每經過一個 cycle，echo_reg1 接收 echo signal，echo_reg2 接收 echo_reg1。

start 為 high 時，echo_reg1 為 high，echo_reg2 為 low。

echo_reg2 接收的是 **cycle time=T-1** 的 echo_reg1 信號。

echo_reg1 為 high，echo_reg2 為 low，代表 **cycle time=T** 時，echo_reg1 接到第一個 echo signal，而 echo_reg2 還沒接到任何信號，這代表最初接收到 echo signal 的瞬間。

finish 為 high 時，echo_reg1 為 low，echo_reg2 為 high。

echo_reg1 為 low，代表 **cycle time=T** 時，echo signal 已經全部傳遞完畢。

echo_reg2 接受到 **cycle time=T-1** 的 echo_reg1 high signal。

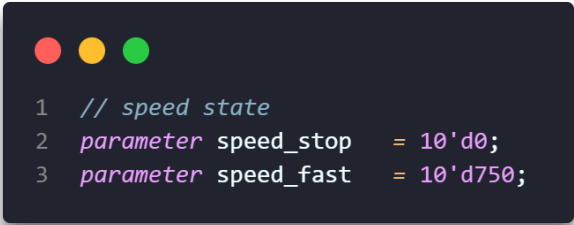
這代表 echo signal 已經全部傳遞完畢的瞬間。

換句話說，state S1 的 counter 在計數 Echo signal 持續了多久的時間，我們再去利用 counter 計算與障礙物的距離。

C. Problem Encountered (10%)

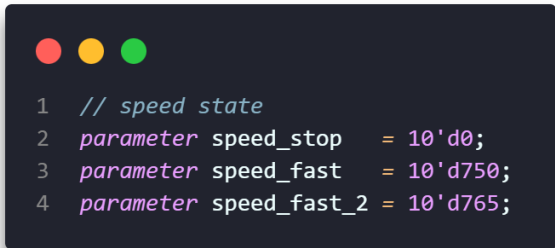
這次 lab 遇到的兩個問題是左邊馬達比右邊馬達無力。

1. 左邊馬達比右邊馬達無力



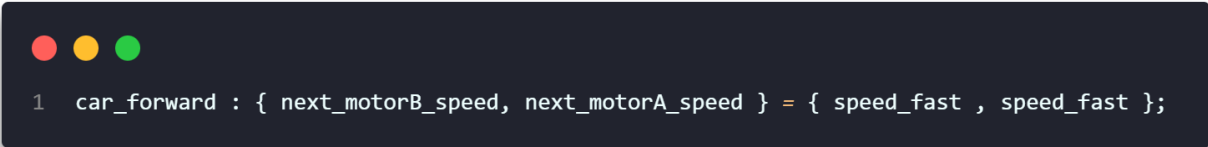
```
1 // speed state
2 parameter speed_stop = 10'd0;
3 parameter speed_fast = 10'd750;
```

(圖 14) 最初速度



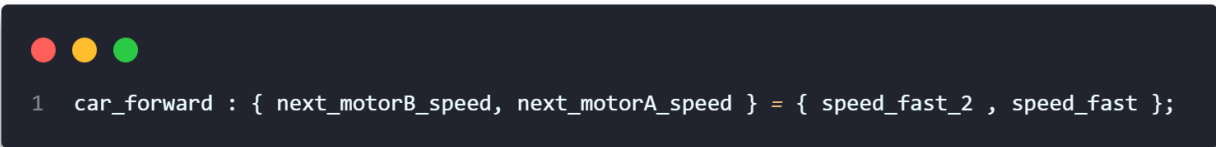
```
1 // speed state
2 parameter speed_stop = 10'd0;
3 parameter speed_fast = 10'd750;
4 parameter speed_fast_2 = 10'd765;
```

(圖 15) 最終速度



```
1 car_forward : { next_motorB_speed, next_motorA_speed } = { speed_fast , speed_fast };
```

(圖 16) 最初速度 assignment



```
1 car_forward : { next_motorB_speed, next_motorA_speed } = { speed_fast_2 , speed_fast };
```

(圖 17) 最終速度 assignment

如標題所說，左邊馬達比右邊馬達無力，起初的設置是圖 14 和圖 16。

我預想中這樣的設計會讓車子直直往前，但因為左邊馬達無力，右邊馬達比較強，右邊輪胎轉速更快，導致車子往右前方行進。

想了非常非常久一直以為 code 寫錯，浪費一堆時間才想到馬達本身可能有問題。

最後設置成圖 15、圖 17 的樣子，讓左邊無力的馬達轉更快，就可以解決兩邊輪胎轉速不同的問題。

左邊 765、右邊 750 是實際花費時間慢慢測出的數字，

過程中，左邊轉太快反而會讓車子變成往左前方走，或是 duty cycle 調太大導致電池斷掉等等。比其他同學花了額外時間解決硬體上的問題。

D. Task Delegation and Suggestions (5%)

不太喜歡當天才公布進階賽道，原因為：

1. 人流多，賽道少。
2. advanced 佔 40% demo 分數，比重太多。

排隊都要等非常久的時間，即使排到了，車子死在某個轉彎處又要重新排很久的隊伍。

即使立刻 de 完 bug 也要排隊，lab demo 兩小時時間大概 50% 的時間都在排隊，給人 “不是不會解 bug，而是 bug 解好後沒有賽道讓我們確定是否解成功的情況”，這樣排隊浪費時機的情況 advanced 又佔 40%，覺得 40% 太多了。

以下為想到的解法：

1. Advanced 比重調低。
2. 當天給更多張 advanced 賽道疏散排隊人潮。
3. Demo 前預先給 advanced 的賽道圖片，讓我們腦中模擬車子的移動情況，demo 當天再給實際賽道。

最後是感謝 TA，幾乎每天都會有人約時間換車子的零件，每天跑來跑去辛苦了。

車子的狀況真的太多太多了。在 dream lab 遇到其他同學，和他們聊天後才知道他們的車子問題也很多。

希望能夠做好車子零件的紀錄，不要讓學弟妹們拿到硬體壞掉的車子，debug 一整天才知道不是 code 寫錯而是硬體本來就壞掉。