Lab6_Team32_Report_Car

組別：Team32

組長：蘇勇誠（108062373）

組員：張晏瑄（108062273）

Design

共有四個主要 module，top 負責將所有訊號串接在一起，motor module 負責控制馬達的速度，而 tracker_sensor 則會根據車車 sensor 感應到黑白比例去調整其方向（左轉、右轉、直線），最後是 sonic ，偵測是否有障礙物在 40 cm 內，若有，則停止車車。

Detail

top

- 將 rst 的按鈕做 debounce, onepulse 傳至各個 module。
- 將 state 切分為 7 個來因應賽道不同的彎曲程度，而衡量標準則是由 tracker_sensor 偵測到賽道的黑白比例，分別為 `go_straight`, `turn_left`, `turn_strong_left`, `turn_peak_left`, `turn_right`, `turn_strong_right`, `turn_peak_right`, `stop_going`。top module 的 state 是處理輪子要向前、向後轉或是靜止。

```
parameter motor_off = 2'b00;
parameter forward = 2'b10;
parameter backward = 2'b01;
```

state (from tracker_sensor module):

當 `stop` 和 `EN_stop` 皆為 1 時，車子為靜止，此時左輪和右輪皆為 motor_off，車子靜止。

`go_straight`: 當要前進時，兩個輪子皆為 `forward`。

`turn_left`: 當要左轉時，兩個輪子皆為 `forward` 。

`turn_strong_left`: 當要程度普通的左轉時，將左輪靜止，右輪向前轉，即可達到此效果。

`turn_peak_left`: 當要程度較劇的左轉時，將左輪往後轉，右輪向前轉，即可達到此效果。

`turn_right`: 當要右轉時，兩個輪子皆為 `forward` 。

`turn_strong_right`: 當要程度普通的右轉時，將右輪靜止，左輪向前轉，即可達到此效果。

`turn_peak_right`: 當要程度較劇的右轉時，將右輪往後轉，左輪向前轉，即可達到此效果。

`stop_going`: 此時兩個輪子皆設為 `motor_off`。

motor

經過很多次嘗試及和其他組別討論後，設計出轉速，傳給 `motor_pwm` 的數字越大，則馬力越強，並搭配 top 所控制的輪子轉向，即可達成 7 個 state 要得轉向及轉速。

測試時在 `turn_strong_left` 和 `turn_strong_right` 皆出了小錯，原本為兩個都設 slow_motor，後來則搭配 slow_e_motor 及 fasr_motor，即可順利轉向。

```
parameter stop_motor = 10'd0;
parameter slow_motor = 10'd512;
parameter fast_motor = 10'd1023;
parameter slow_e_motor = 10'd100;
```

`go_straight`: 左右馬達皆為 fast_motor，且兩個輪子皆向前轉

`turn_left`: 左馬達為 slow_motor，右馬達為 fast_motor，且兩個輪子皆向前轉

`turn_strong_left`: 左馬達為 slow_e_motor，右馬達為 fast_motor，且左輪靜止，右輪向前轉

`turn_peak_left`: 左右馬達皆為 fast_motor，且左輪往後轉，右輪往前轉

`turn_right`: 左馬達為 fast_motor，右馬達為 slow_motor，且兩個輪子皆向前轉

`turn_strong_right`: 左馬達為 fast_motor，右馬達為 slow_e_motor，且左輪向前，右輪靜止

`turn_peak_right`: 左右馬達皆為 fast_motor，且左輪往前轉，右輪往轉

`stop_going`: 停止時，左右馬達皆為 stop_motor，左右輪皆靜止。

- **PWM_gen**:

傳入 `duty` (`10'd0`~`10'd1023`，即 `left_motor`, `right_motor` 之值)，若 `duty` 值越大，則 `count_duty` 值亦越大，其 output `PWM` 為 1 的比例越多，則馬力越強，反之亦然。

```
wire [31:0] count_max = 32'd100_000_000 / freq;
wire [31:0] count_duty = count_max * duty / 32'd1024;
reg [31:0] count;

always @(posedge clk, posedge reset) begin
    if (reset) begin
        count <= 32'b0;
        PWM <= 1'b0;
    end else if (count < count_max) begin
        count <= count + 32'd1;
        if(count < count_duty)
            PWM <= 1'b1;
        else
            PWM <= 1'b0;
    end else begin
        count <= 32'b0;
        PWM <= 1'b0;
    end
end
```

sonic_top

當偵測到車車前方 40 cm有障礙物時，會傳出 `stop` 訊號，而 dis 是以 0.01 cm 為單位，故為 `dis <= 21'd4000`

```
assign stop = (dis <= 21'd4000) ? 1'b1 :1'b0;
```

- **PosCounter:**

空氣中聲速大約為 343.2 m/s (20 °C)，換算成聲音傳播 1 cm 所需時間，及 1/343.2 = 29.1 μs。

由開始發射超音波至接收到回音的時間差 t，可以計算出超音波「來回」走了多少距離 = 時間 * 速率 = (t/2) * (1/29.1) = t/58.1 (cm)，而因為是以 0.01 cm 為單位，故還要再 *100。

`start` 則表示有 posedge

`finish` 則表示有 negedge

```
assign distance_count = distance_register * 21'd100 / 21'd58;
assign start = echo_reg1 & ~echo_reg2;
assign finish = ~echo_reg1 & echo_reg2;
```

- **TrigSignal**: 將 trig 設為 `1'b1` 10 μs，且 trig 週期為 1 sec。此為產生超音波的訊號。

```
always @(*) begin
    next_trig = trig;
    next_count = count + 1'b1;
    if(count == 24'd999)
        next_trig = 1'b0;
    else if(count == 24'd9999999) begin
        next_trig = 1'b1;
        next_count = 24'd0;
    end
end
```

- **div**: 將原本 100MHz 的 clk 轉為1MHz 的 out_clk。用於傳入 PosCounter。

```
always @(posedge clk) begin
    if(cnt < 7'd50) begin
        cnt <= cnt + 1'b1;
        out_clk <= 1'b1;
    end
    else if(cnt < 7'd100) begin
      cnt <= cnt + 1'b1;
      out_clk <= 1'b0;
    end
    else if(cnt == 7'd100) begin
        cnt <= 7'b0;
        out_clk <= 1'b1;
    end
    else begin
        cnt <= 7'b0;
        out_clk <= 1'b1;
    end
  end
```

tracker_sensor

根據 sensor 傳入的 left_signal, mid_signal, right_signal 將 state 設定為 7 種其中之一，若賽道為白，則訊號為 1，為黑則為 0。而由於當賽道為黑時，需要判斷此時該向右還是

向左回到正軌，故還有一個紀錄 direction 的 reg。

```
reg [1:0] direction, next_direction;

parameter left = 2'd0;
parameter right = 2'd1;
parameter straight = 2'd2;
```

{left_signal, mid_signal, right_signal}:

`000`: 用前一個state的direction判斷要左轉還是右轉。當前一個state `direction` 為左時，state設為 `turn_peak_left`，為右則為 `turn_peak_right`。

```
3'b000: begin
        next_direction = direction;
        if(direction == left)
            next_state = turn_peak_left;
        else if(direction == right)
            next_state = turn_peak_right;
        else
            next_state = state;
        end
```

`001`: 此時最左邊兩個 sensor 偵測到黑色，車車應該往右回到正軌，且偏離軌道有些嚴重，故將 state 設為 `turn_strong_right`， `direction` 則為右（表示要右轉）。

```
3'b001: begin
    next_state = turn_strong_right;
    next_direction = right;
end
```

`011`: 此時最左邊的 sensor 偵測到黑色，車車應該往右回到正軌，而偏離軌道程度普通，將 state 設為 `turn_right` 即可， `direction` 為右（表示要右轉）。

```
3'b011: begin
    next_state = turn_right;
    next_direction = right;
end
```

`100`: 此時最右邊兩個 sensor 偵測到黑色，車車應該往左回到正軌，偏離軌道有些嚴重，故將 state 設為 `turn_strong_left`，下個 `direction` 則為左（表示要左轉）。

```
3'b100: begin
    next_state = turn_strong_left;
    next_direction = left;
end
```

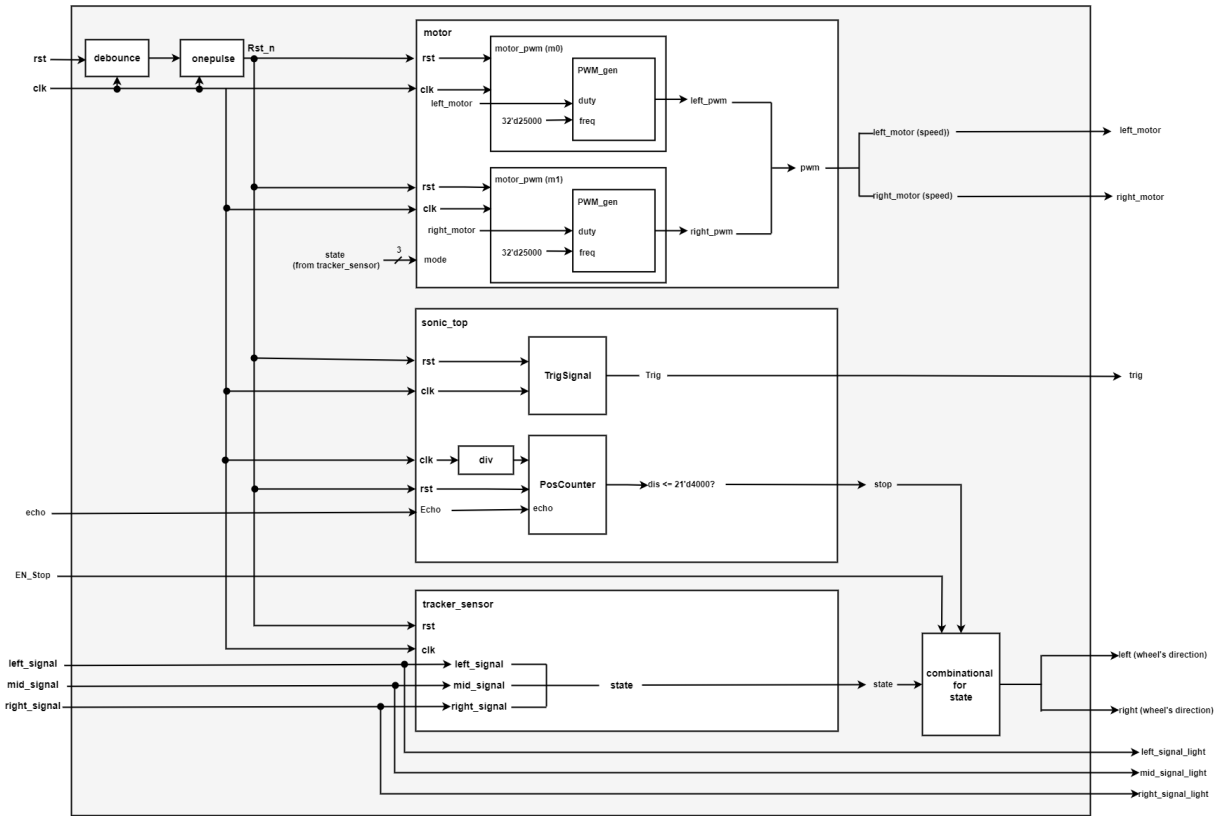
`110`: 此時最右邊的 sensor 偵測到黑色，車車應該往左回到正軌，而偏離軌道程度普通，將 state 設為 `turn_left` 即可，`direction` 為左（表示要左轉）。

```
3'b110: begin
    next_state = turn_left;
    next_direction = left;
end
```

`111`: 此時三個 sensor 都偵測白色，車車正在軌道上，直線前進即可。

```
default: begin
   next_state = go_straight;
   next_direction = straight;
end
```

Block Diagram



Learning

1. 有了 chip2chip 的經驗，為避免搞混線的顏色，同樣用 excel 檔做紀錄

motor						sensor					sonic			
ENA	綠	JB3	B15	right_motor		R	紫	JA2	L2		vcc	橘		
IN1	藍	JB2	A16	right[1]		C	綠	JA3	J2		trig	紅	JXAC2	L3
IN2	紫	JB1	A14	right[0]		L	藍	JA4	G2		echo	咖啡	JXAC1	J3
IN3	灰	JB9	C15	left[1]		GND	黃	JA5			gnd	黑		
IN4	白	JB8	A17	left[0]		VCC	橘	JA6						
ENB	黑	JB7	A15	left_motor										
GND														

2. 學習到硬體模組的應用
3. 抓 bug 時有用 FPGA 的 LED 燈顯示 state 及 sensor 的訊號，來偵測是否為硬體出錯還是 code 寫錯
4. 很多時候花了超多時間找 bug，結果最後原來是電池沒電
5. motor 的轉速調配可以藉由詢問其他組的經驗以及不斷的 try and error 得出

分工

- 蘇勇誠（108062373）

  motor module

  sonic module

  tracker_sensor module
- 張晏瑄（108062273）

  report

  .xdc 接線