group 37 : Final project
是食物者為俊傑

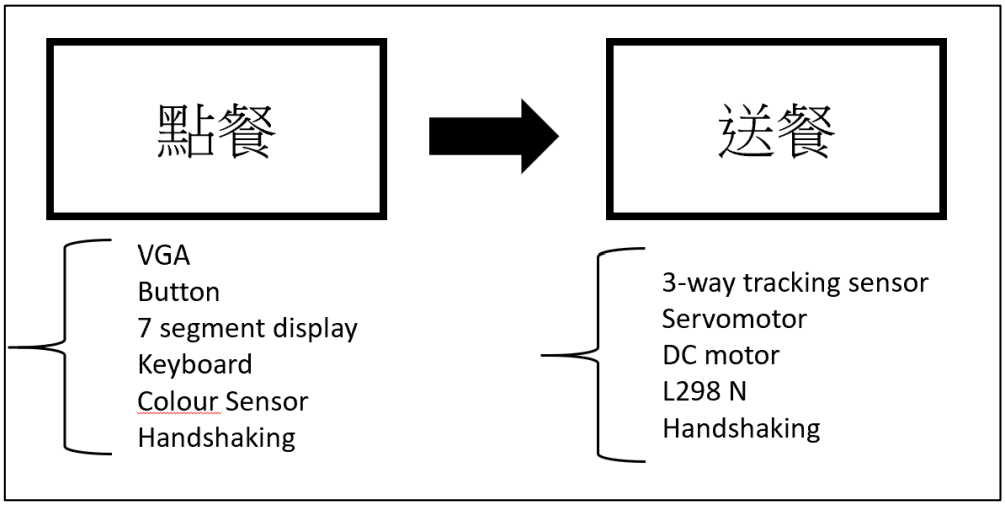
112062139 劉韋呈

112062117 王政傑

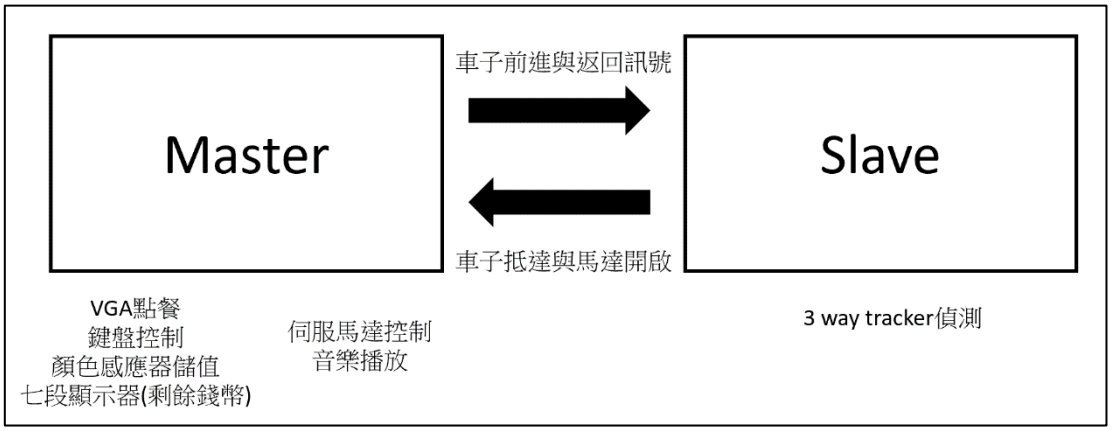
A. 硬體設計概念

本次 project 模擬自動點餐機，先透過 master 端呈現音樂與點餐畫面，透過鍵盤選取餐點。此外，利用 TCS3200 顏色感應器可以獲取當前物體的 RGB，當按下右邊的 Button 時，會根據相應假鈔的顏色，增加對應的金額，並顯示於七段顯示器。

接著，進入準備狀態，master 端 handshaking 位子給 slave 端，並抬起接受訊號，車子利用 3-way tracking sensor 保持直線運動，並且當接受全黑的訊號時，表示跨過一個站點。當抵達目的地時，slave 端抬起抵達訊號，此時 master 端操控伺服馬達轉動，物體掉落於車上的漏斗後，master 端抬起回去的訊號，車子到回原點後抬起結束訊號，slave 短暫進入 final 狀態後，回到 initial 狀態。



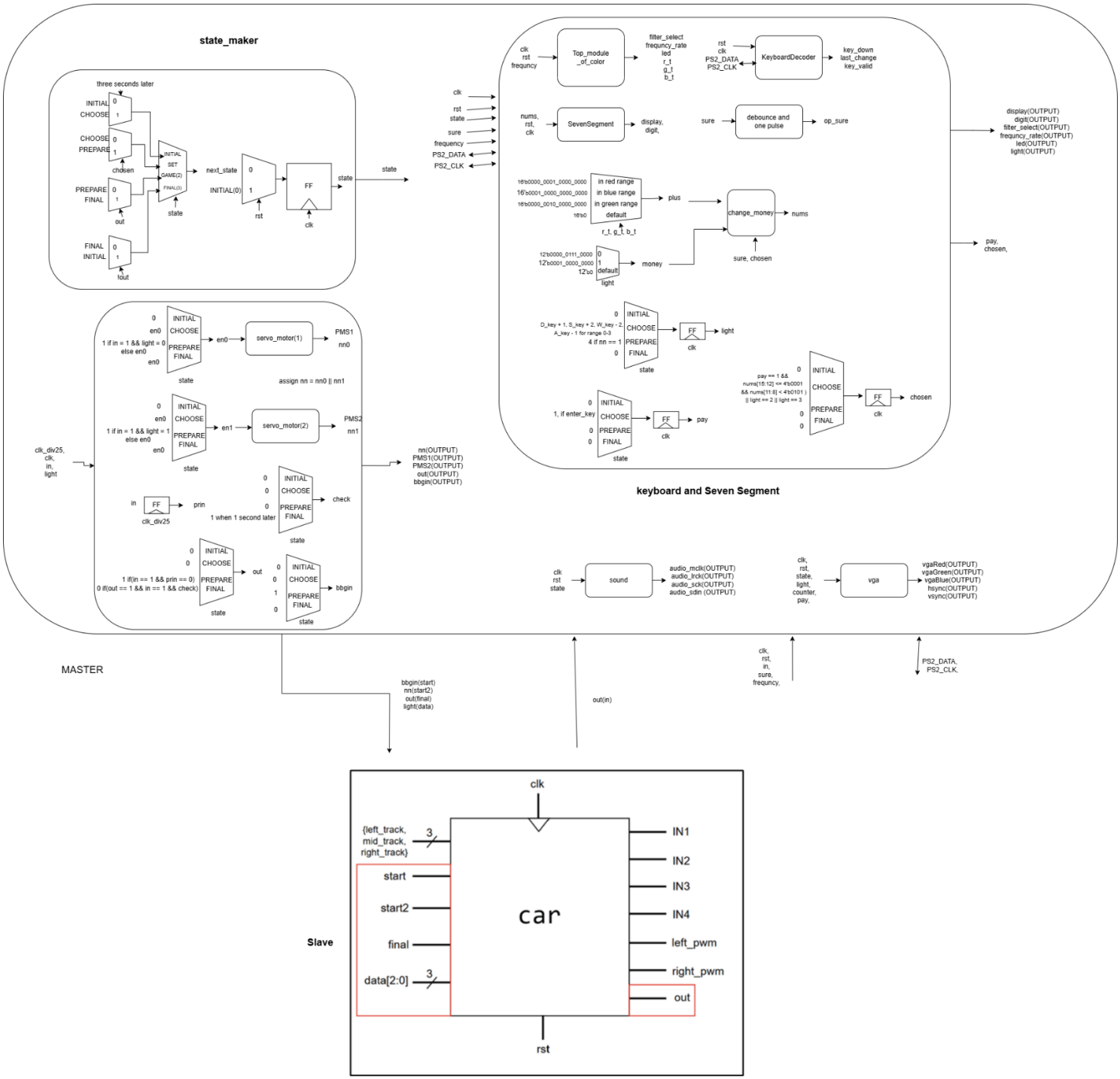
專題器材與技巧關聯圖



主從關係示意圖

B. 架構細節與方塊圖

a. Block diagram



b. VGA 與七段顯示器

七段顯示器會顯示目前玩家擁有的金額，在 CHOOSE state 時，你可以選擇你要點的餐點，螢幕上會有四張圖，用 light 來記錄目前你所選的餐點，被選的圖上會有淡淡的藍色，表示他正在被選，當確定要選的東西時，按下 enter 後會跑到付費介面，你可以按 N 表示不要付費，按 enter 會進入 PREPARE state 並付費，七段顯示器的值會改變。

c. 顏色感應器儲值

先進行白平衡，找出各種顏色的光強度上限，並控制在 256 以下。ready 變數代表白平衡已經完成。

```
if(r_counter<255)
begin
    r_counter=r_counter+1;
    filter_select=2'b00;
end
else if(r_counter==255)
begin
    r_counter=r_counter+1;
    R_time=counter;
end
...
if(r_counter>255&&g_counter>255&&b_counter<255)
begin
    filter_select=2'b10;
    b_counter=b_counter+1;
end
else if(b_counter==255)
begin
    b_counter=b_counter+10;
    B_time=counter-R_time-G_time-1;
end

assign ready=(r_counter>255&&g_counter>255&&b_counter>255);
```

白平衡程式碼

frequncy 變數來源於顏色感應器，透過接受不一樣的光強度，會產出對應的頻率。而 filter_select 代表遮色片的模式，當為 2’b00 時，此時只有紅光通過，2’b10 時，則只有藍光通過，最後當 2’b11 時，為綠光通過。借助這種方法，可以有效得知 RGB 數值。

```
always@(posedge frequncy&& ready) begin
    if(counter==0)
    begin
        r_counter=0;
        g_counter=0;
        b_counter=0;
        reset=0;
    end
    if(counter<r_time)
    begin
        filter_select = 2'b00;
        r_counter = r_counter+1;
    end
    else if(counter>=r_time&&counter<r_time+g_time)
    begin
        filter_select = 2'b11;
        g_counter = g_counter+1;
    end
    else if(counter>=r_time+g_time&&counter<r_time+g_time+b_time)
    begin
        filter_select = 2'b10;
        b_counter = b_counter+1;
    end
    else if(counter>=r_time+g_time+b_time)
    begin
        red=r_counter-1;
        green=g_counter-1;
        blue=b_counter-1;
        reset=1;
    end
end
```

顏色判斷程式碼

d. 伺服馬達控制

當進入 PREPARE state 後，bbgin(slave 的 start) = 1 代表車子可以開始動了，如果 slave 傳入的 in(slave 的 out) = 1，根據 light 的值改變哪個 en，其對應的馬達轉 360 度，我用的方法是讓伺服馬達一直轉，大概估算一下轉完一圈的時間，時間到了就停下，轉完後輸出 nn(slave 的 start2)給 slave，代表車子可以倒回原點。等到倒回原點後，slave 會傳入 in(slave 的 out)=1，如果 prin(0.25 秒前的 in)也為 0，out 變為 1，輸出給 slave 的同時，也利用 out == 1，使進入 FINAL state。

e. 音樂控制

在 note_gen module 中，原本波型為方波圖，透過描點的方式，將波型轉成類似鋼琴的波型，呈現的效果更加平滑順耳。

```
always @* begin
   if (note_div_left == 22'd1)
    audio_right = 16'h0000;
else begin
    if (clk_cnt >= 0 && clk_cnt < note_div_left / 20)
        audio_right = 16'h0100;
    else if (clk_cnt >= note_div_left / 20 && clk_cnt < note_div_left * 2 / 20)
        audio_right = 16'h0350;
   ...
    else if (clk_cnt >= note_div_left * 18 / 20 && clk_cnt < note_div_left * 19 / 20)
        audio_right = 16'h0200;
    else if (clk_cnt >= note_div_left * 19 / 20 && clk_cnt <= note_div_left)
        audio_right = 16'h0100;
end
end
```

震幅控制程式碼

利用 beat module，根據不同狀態來控制歌曲的時間長度，當播完之後，再次循環播放。

```
module beat(
    input clk,
    input rst,
    input [1:0] state,
    output reg [11:0] ibeat
);
    wire [8:0] LEN;
    reg [11:0] next_ibeat;
    reg [1:0] last_state;
    assign LEN=(state==0)?0:
                (state==1)?272:
                (state==2)?242:400;
    always @(posedge clk, posedge rst) begin
        if (rst) begin
            ibeat <= 0;
            last_state <= 0;
        end else begin
            if (state != last_state) begin
                ibeat <= 0;
            end else begin
                ibeat <= next_ibeat;
            end
            last_state <= state;
        end
    end
    always @* begin
        next_ibeat = (ibeat + 1 < LEN) ? (ibeat + 1) : 0;
    end
endmodule
```

beat module 程式碼

利用 sheet module，將網路上的五線譜轉成音調，然後以四分音符為佔十六個為基礎，去分配節奏於程式碼中，並同音調連音時要隔一個 silence。此次專題總共三首曲子，每首曲大致有八個小節，右聲道處理高音部，而左聲道則處理低音部。

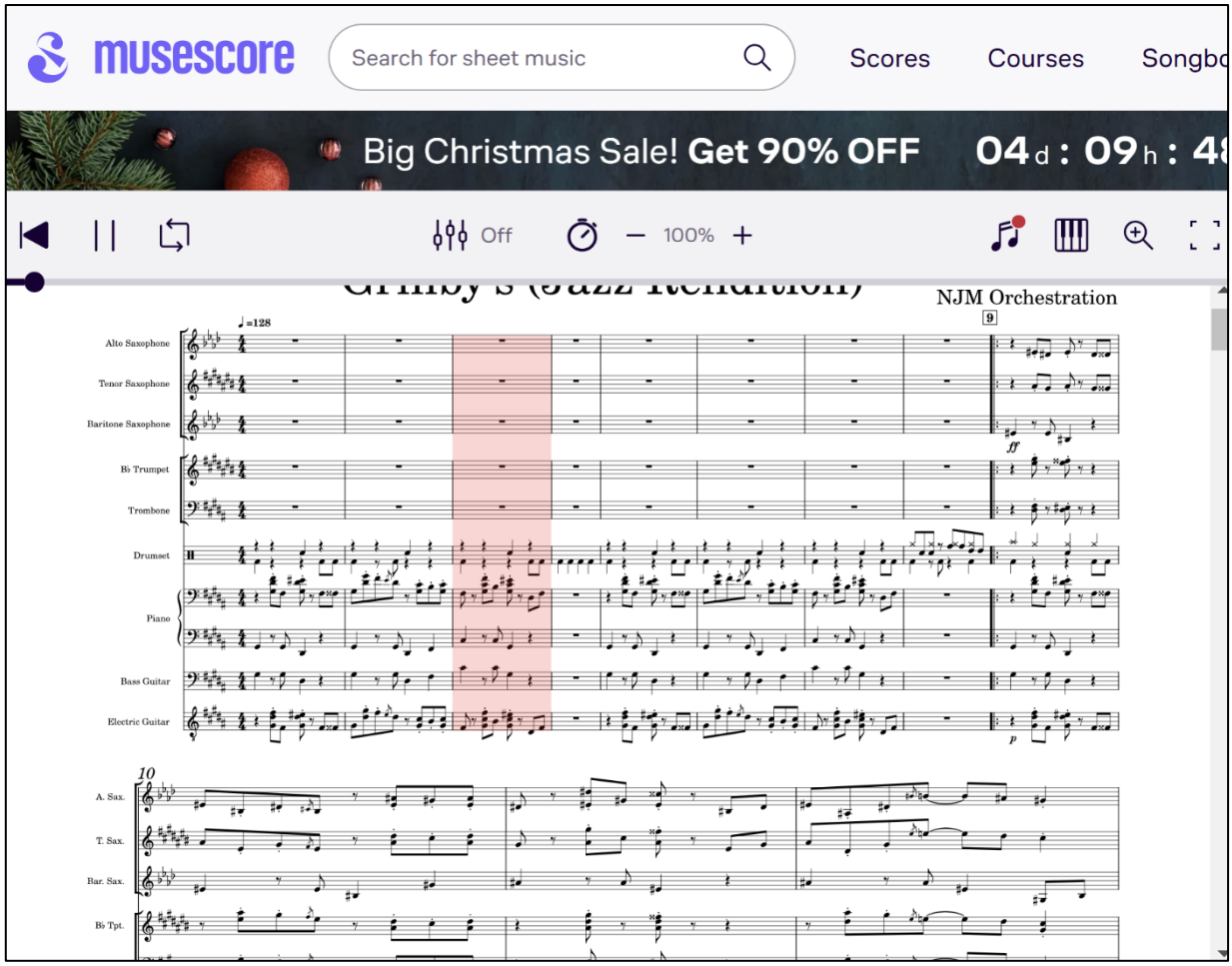
```
module sheet(
    input [11:0] ibeatNum,
    input [1:0] state,
    output reg [31:0] toneL,
    output reg [31:0] toneR
);

    always @* begin
        if(state==0) begin
            toneR = `silence;
        end
        else if(state==1) begin
            case(ibeatNum)
                // --- sheet1(1/4) ---
                12'd0: toneR = `silence;  12'd1: toneR = `silence;
                12'd2: toneR = `silence;  12'd3: toneR = `silence;
                12'd4: toneR = `silence;  12'd5: toneR = `silence;
                12'd6: toneR = `silence;  12'd7: toneR = `silence;
                12'd8: toneR = `silence;  12'd9: toneR = `silence;
                12'd10: toneR = `silence; 12'd11: toneR = `silence;
                12'd12: toneR = `silence; 12'd13: toneR = `silence;
                12'd14: toneR = `silence; 12'd15: toneR = `silence;

                12'd16: toneR = `fau;     12'd17: toneR = `fau;
                12'd18: toneR = `fau;     12'd19: toneR = `fau;
                12'd20: toneR = `fau;     12'd21: toneR = `fau;
                12'd22: toneR = `fau;     12'd23: toneR = `fau;
                ...
```

sheet module 程式碼

五線譜皆來自 musescore 這個網站，選擇較為簡化的譜來實做。



f. 自走車控制

自走車在 master 端傳遞位子並接收到 start 或 start2 的訊號後，透過 3-way tracking sensor 來直進，並且當收到全黑訊號時，會記錄過一個站點。抵達目的地後，輸出 out 值用於後續的馬達控制，或是進入 final 狀態。

在 motor module 中，根據不同情況設定不同的轉速差異，以維持自走車的直線前進。因為初始情況下，黑色直線軌道會位在車子的左側，故通過讓車子往右偏的方式，並在檢測到全白的情況後，偏回左邊，以此達到車頭保持向前的狀態。

可以注意到，Move 情況的輪胎轉速偏向右側，也是為了校正車頭而設計。但是因為倒退走的情況很難掌握，所以依舊維持兩輪轉速一致的情況。此外，IN 值為 2’b10 是往前轉，而 2’b01 則是往後轉。

```
case (mode)
    3'b000: begin  //Stop
        left_motor <= 10'd0;
        right_motor <= 10'd0;
        l_IN_t <= 2'b00;
        r_IN_t <= 2'b00;
    end
    3'b001: begin //Move
        left_motor <= 10'd700;
        right_motor <= 10'd695;
        l_IN_t <= 2'b10;
        r_IN_t <= 2'b10;
    end
    3'b010: begin//Back
        left_motor <= 10'd695;
        right_motor <= 10'd695;
        l_IN_t <= 2'b01;
        r_IN_t <= 2'b01;
    end
    3'b011: begin//Right_M
        left_motor <= 10'd710;
        right_motor <= 10'd680;
        l_IN_t <= 2'b10;
        r_IN_t <= 2'b10;
    end
    3'b100: begin//Left_M
        left_motor <= 10'd685;
        right_motor <= 10'd705;
        l_IN_t <= 2'b10;
        r_IN_t <= 2'b10;
    end
    3'b101: begin//Right_B
        left_motor <= 10'd685;
        right_motor <= 10'd700;
        l_IN_t <= 2'b01;
        r_IN_t <= 2'b01;
    end
    3'b110: begin//Left_B
        left_motor <= 10'd700;
        right_motor <= 10'd685;
        l_IN_t <= 2'b01;
        r_IN_t <= 2'b01;
    end
    default: begin//Move
        left_motor <= 10'd690;
        right_motor <= 10'd690;
        l_IN_t <= 2'b10;
        r_IN_t <= 2'b10;
        LED<=3'b001;
    end
```

轉速控制程式碼

在 tracker_sensor module 中，start 代表接受前進訊號，而 start2 代表接收返回訊號。當偵測到位置訊號改變後，根據地板黑線判斷方向，當偵測到三黑代表過一站。而 init 變數是避免將起始三黑訊號算入一站，而 leave 變數是避免重複將一站誤認成多站。

```
if(start==1 || start2==1 ) aim<=data;
else aim<=aim;

if(aim!=where) begin
        init<=1;
        if(aim==0 || aim==1) begin
            if({left_track, mid_track, right_track}==3'b011) begin
                state<=Right_M;
                pre<=0;
                leave<=1;
            end
            else if({left_track, mid_track, right_track}==3'b111) begin
                if(pre==0) state<=Left_M;
                else state<=Right_M;
                leave<=1;
            end
            else if({left_track, mid_track, right_track}==3'b000) begin
                if(leave==0) state<=Move;
                else begin
                    leave<=0;
                    if(where==4) where<=0;
                    else where<=1;
                end
            end
        end
    else if(aim==4) begin
        out<=0;
        if(where==0 || where==1) begin
            if({left_track, mid_track, right_track}==3'b011) begin
                state<=Right_B;
                pre<=0;
                leave<=1;
            end
            else if({left_track, mid_track, right_track}==3'b111) begin
                if(pre==0) begin
                    state<=Left_B;
                end
                else begin
                    state<=Right_B;
                end
            end
            else if({left_track, mid_track, right_track}==3'b000) begin
                if(leave==0) state<=Back;
                else begin
                    leave<=0;
                    if(where==1) where<=0;
                    else if(where==0) where<=4;
                end
            end
        end
    end
end
else begin
    state<=Stop;
end
```

自走車前進程式碼(1)

當當前位置與目標位置一致時，傳送抵達訊號。並且，當 master 端傳送 final 訊號時，代表已經抵達終點，經過一秒後，初始化所用的變數。

```
else begin
    ...
    if(init==0) begin
        out<=0;
    end
    else begin
        if(final==0) out<=1;
        else begin
            counter<=counter+1;
            if(counter[26]==1 && counter[25:0]==0) begin
                out<=0;
                init<=0;
                where<=4;
                counter<=0;
            end
            else  out<=1;
        end
    end
end
```

自走車前進程式碼(2)

C. 實作完成度

本次專題大致有三個部分未實作。
第一，我們將藍芽控制改成直接用 handshaking，因為藍芽在操作上有一定難度。
第二，將食物掉落改成物品掉落，因為可能會造成不必要的浪費。
第三，我們將超聲波感應器刪除，因為容器固定於車子上，所以不用實作拿走容器的判斷。
總結，我們認為有 80%的完成度。

D. 難易度說明

困難的地方在於我們動用了所有課堂學到的技巧，所以相互配合起來有點吃力，加上大量實體物件需要手工製作。

VGA 控制需要思考容量上的問題，而音樂則需要思考波形方面的設計，以及理解並將音符放置於程式碼中，需要很多時間。自走車方面，可能光線等因素，3-way tracking sensor 有時候會失靈，需要不斷設計新的地圖。

最後，伺服馬達與顏色感應器皆屬於課堂外的技巧，所以蒐集了很多資料去理解。

E. 困難與解決

a. VGA 與七段顯示器

如果載太多圖 Bram 會有過載的問題，因此我有的畫面要用刻的，在不同 h_cnt 和 v_cnt 輸入對應的顏色，使得螢幕顯示出你想要的圖案或字。

b. 顏色感應器儲值

原本使用 tcs34725 顏色感應器，但發現 I^2C 通訊難以實作於 FPGA 板上，後面就改成 TCS3200。此外，一開始沒有白平衡，導致數值極其不穩定。

c. 伺服馬達控制

因為 FPGA 的接孔的電流會變小，因此如果伺服馬達的 GND 和 VCC 接 FPGA 的孔時，馬達轉不起來，因此我們使用了 Pmod con3，這使我們可以用變壓器接 Pmod con3 的 GND 和 VCC 來供電。

d. 音樂控制

方波音聽起來很刺耳，將波形後改善很多。在找譜方面，需要找到簡化但是不偏離原曲的譜，並不容易，甚至有去找除了鋼琴以外的譜。

e. 自走車控制

由於自走車整體的構造，所以倒退走花了非常多時間，解決方式是改短倒退的移動路徑，這樣直線前進時，可以再度修正回來。

F. 心得討論

這個專題可謂是整學期課程的總結，透過不同技巧的搭配，有時會出現意想不到的效果。雖然中間製作的過程很辛苦，但看到 demo 能順利進行，成就感滿滿。

若想延伸此專題，或許能將紙板改成木板比較牢固，然後如果能優化密封性，或許能改成液體的流動控制等。

G. 詳細分工

a. 王政傑

- **VGA 控制(master 端)**
- **伺服馬達控制**
- **七段顯示器**

b. 劉韋呈

- **自走車控制(slave 端)**
- **顏色感應器控制**
- **聲音控制**

H. 課程外部分

a. 伺服馬達

我把伺服馬達的脈衝寬度固定，讓馬達一直轉，我們用計數器來計算甚麼時候馬達要停，這樣能使伺服馬達轉到我們想要的角度。

以下是轉 360 度後停止的 code:

```
module servo_motor_two(
    input clk,
    input en,
    output reg PMS,
    output reg nn
);
    reg [29:0] timer;
    reg [25:0] counter;
    reg start;
    always @(posedge clk ) begin
        if (!en) begin
            PMS <= 0;
            counter <= 0;
            timer<=0;
            start<=0;
            nn <= 0;
        end else if(nn) begin

        end else begin
            if(start==0) begin
                if(timer == 89478485) begin
                    counter<=0;
                    timer<=0;
                    start<=1;
                    PMS<=0;
                    nn <= 1;
                end
                else begin
                    timer<=timer+1;
                if (counter < 50000) begin // 高電平 (1.5ms)
                            PMS <= 1;
                             counter <= counter + 1;
                        end else if (counter < 2000000) begin
                            PMS <= 0;
                            counter <= counter + 1;
                        end else begin
                            counter <= 0; // 重置計數器
                        end
                    end

                end
                else begin
                    if(timer[27]==1 && timer[26:0]==0) begin
                    start<=0;
                    timer<=0;

                    end
                    else timer<=timer+1;
                end

            end
        end
    end
endmodule
```

伺服馬達控制程式碼

b. 顏色感應器

在顏色感應器的控制上，遇到很多困難，最後在一個 github 上找到相關的例子，所以有 RGB 部分程式碼參考於這個網站，屬於此專題未完整實作的部分。

Github: https://github.com/zhangyiwen599/VGA_game