# Final Project report

組員：110062221 李品萱
110062213 唐翊雯

## Peatank

目錄

## I. Introduction

在這學期的 final project 中，我們基於 ”Peatank”，也就是結合植物大戰殭屍中的 peashooter 加上一些坦克的元素，做出一個可被遙控的裝置，外觀如下圖。

在圖中 peashooter（綠色豌豆頭）的部分，我們實際上改良了發球機的操作完成他每次能夠發射一顆子彈，此外，他的頭也是可以水平旋轉的，這邊我們用了減速馬達讓他能以慢且穩定的速度旋轉，而在頭的後側有一個白色的柱狀體，作為儲球槽的功能使用，方便操作者將他的子彈藉由儲球槽送到球道中；而由於我們要做出能轉動的頭，我們需要將平台架高，為了添加坦克的成分，我們也用了不織布呈現迷彩，並配合各個訊號的接線做整線的動作。最後我們可以藉由板子遙控 peatank，讓他做出八個方向的移動、速度模式切換、轉動砲台及發射子彈等動作。

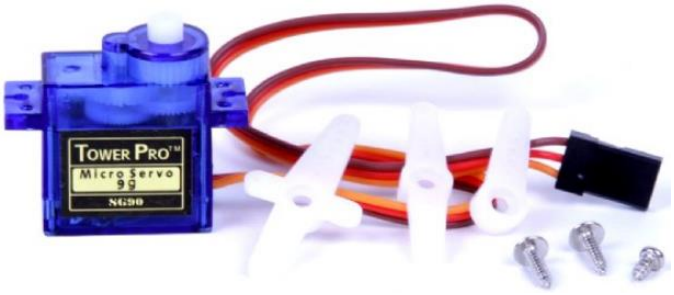
## II. Motivation

在發想主題的過程中，一開始我們想到的是遊戲機台，例如推硬幣、娃娃

機、打地鼠等等，或者發球機、遙控蜘蛛之類比較特別的設計。最後，我們把主意打到了植物大戰殭屍的豌豆(Peashooter)身上。結合之前想到的發射東西、遙控，及遊戲體驗，我們希望做出一個可以在玩家的控制下行進及發射子彈的豌豆，結合坦克的特徵做出 Peatank。

## III. Material

### A. Servo motor SG90




我們使用了兩個 **Tower Pro Micro Servo SG90 9g** 來實作控制豌豆一次發射一個子彈，它是一個小型伺服系統，重量 9 克，可以推動約 35 公斤的重量。藉由特定週期波及一個週期波內的 low、high 比例旋轉特定角度，由 SG90 內的馬達及齒輪帶動。

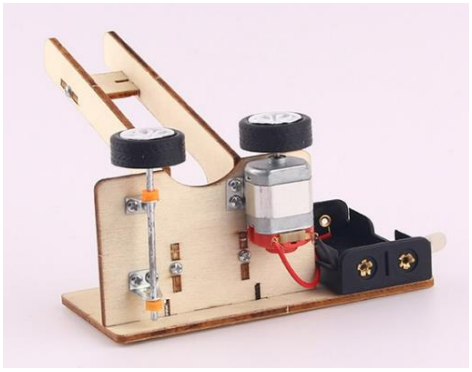
不同種類的伺服馬達會有不同的最大轉動角度，而我們使用的 SG90 最大轉動角度為 180，可以控制其正轉逆轉，其三條線對應訊號如下。

| | |
|---|---|
| **紅線** | 供電 |
| **橘線** | 訊號 |
| **褐線** | 接地 |

### B. tt 減速機

tt 減速機可以讓我們實現以穩定速度及足夠的扭力讓豌豆的頭正常轉動，tt 馬達由兩軸做軸心轉動，利用曲柄帶動足部支點作圓周運動，它可由不同的減速比區分，我們使用的是減速比 1:220 的 tt 馬達減速機，減速比值越小轉速越慢，扭力越大，可滿足我們的需求。

## C. 自動發球機



自動發球機由木板、馬達及電池組成，向下的軌道可以讓球盡量地接近發球口，而發球口由馬達及輪子控制，當通電時馬達就會帶動輪子快速旋轉，此時一旦球碰到輪子就會被帶動飛出，實測效果非常良好。

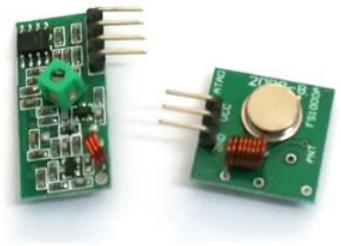
## D. HC-05 藍芽模組

HC-05 是主從一體的藍芽接發器，為專用於無線通訊的 Serial port protocol module，用 UART 來 implement serial port，可設定主機從機。我們用兩個 HC-05 對接實作。主機向從機建立連線，支援藍牙 2.0 + EDR 規範，預設 Baud Rate 通常為 38400，傳輸距離可達 10M（無阻

隔），有六個 pin 腳，對應功用如下。

| | |
|---|---|
| **VCC** | 供電 5V 或 3.3V |
| **GND** | 接地 |
| **TXD** | 傳送 serial data |
| **RXD** | 接收 serial data |
| **STATE** | 檢查模組是否連接 |
| **EN** | 讓模組處於 AT 模式，以進行後續操作 |

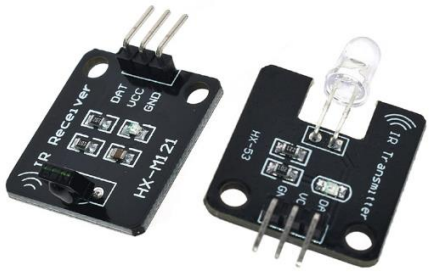
## E. rf-link 433



頻率 433 的無線接發器，接受供電 3-12V，電壓越高訊號越穩定，傳輸距離越遠，訊號範圍 40 至 100 公尺，亦可由焊天線的方式增加傳輸距離，接腳對應功能如下。

| | |
|---|---|
| **DATA** | 接收 / 傳送訊號 |
| **5V** | 供電 |
| **GND** | 接地 |

## F. ir transmitter




以紅外線形式傳輸的無線接發的模組，須遵守其 protocol 傳送訊號，其接腳對應功能如下。

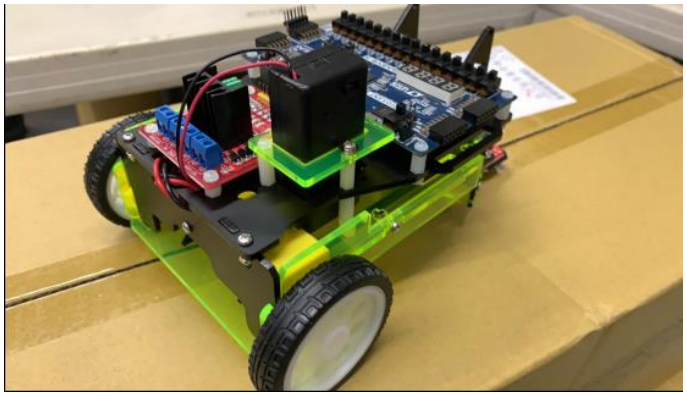
| | |
|---|---|
| **DAT** | 接收 / 傳送訊號 |
| **VCC** | 供電 |
| **GND** | 接地 |

## G. KY-023 Joystick

俗稱蘑菇頭，其十字搖杆為一個雙向的 10K 電阻器，隨搖桿方向不同（x、y 軸），電阻值會發生 0 到 5V 的變化，輸出為類比訊號，而 z 軸則是檢測其是否被按下，其接腳對應功能如下。

| | |
|---|---|
| **GND** | 接地 |
| **VCC** | 供電 |
| **VRX** | X 軸輸出 |
| **VRY** | Y 軸輸出 |
| **SW** | 按鈕 |

### H. Car (from Lab 6)



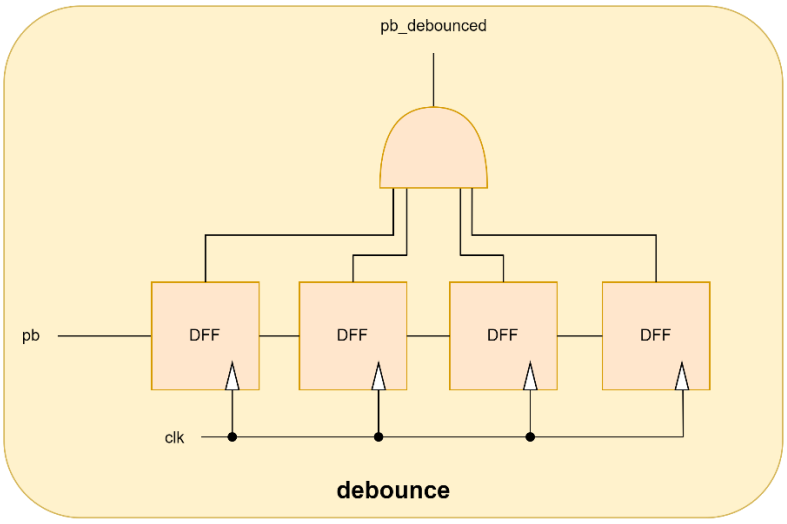
## IV. System specification

我們的 code 主要分成兩大部分：作為 master 的 remoteCtrl 與作為 slave 的 car，分別負責遠端遙控車子與實際接在車子上控制車子，以下我們會分別說明兩部分 code 的實作。

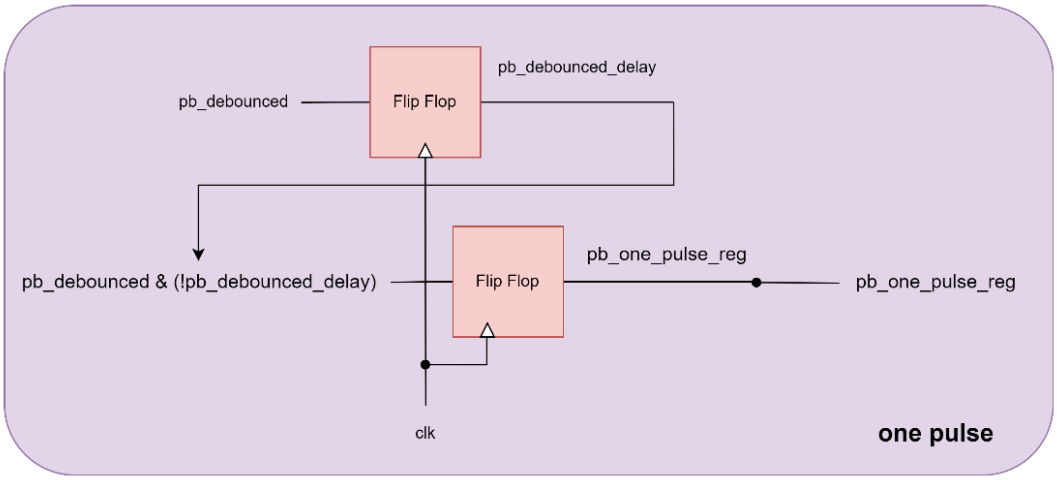
### A. Utility

首先我們將一些 utility module 整理成 utility.v，各 module 如下。

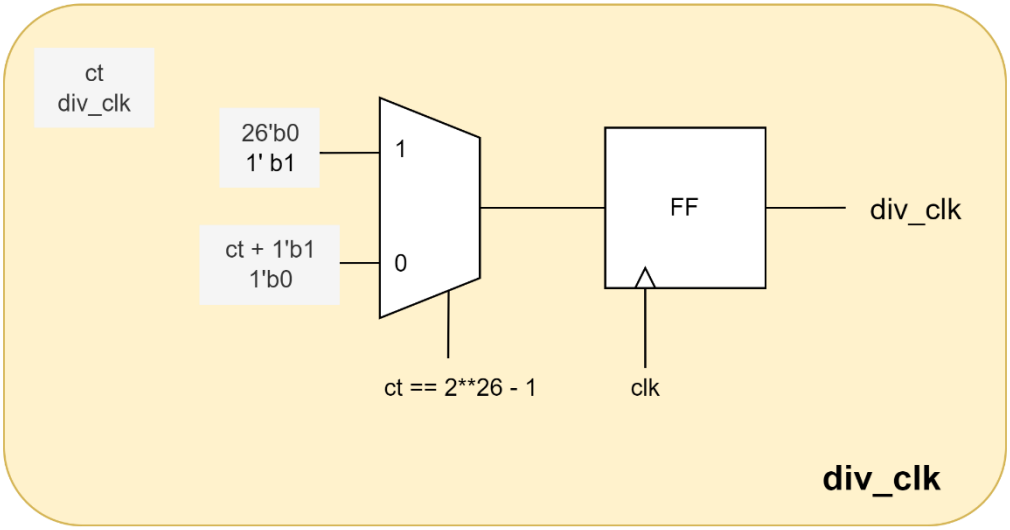
- **debounce**



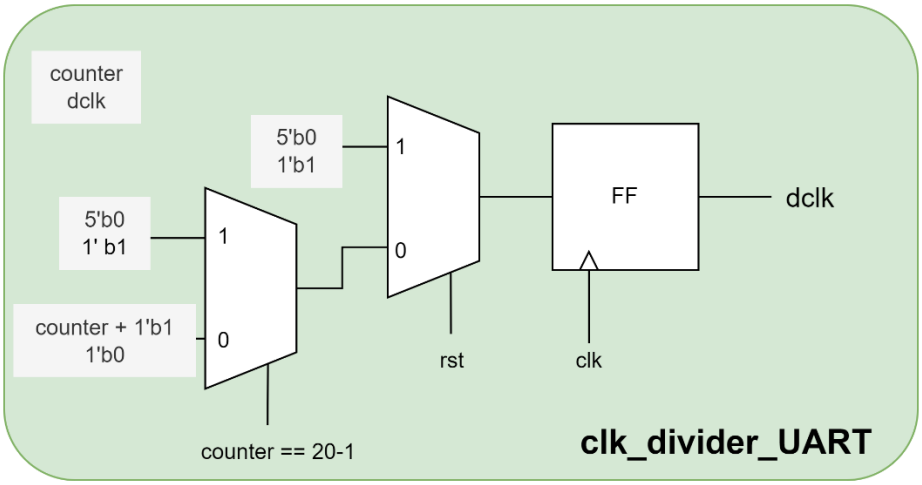

- **onepulse**



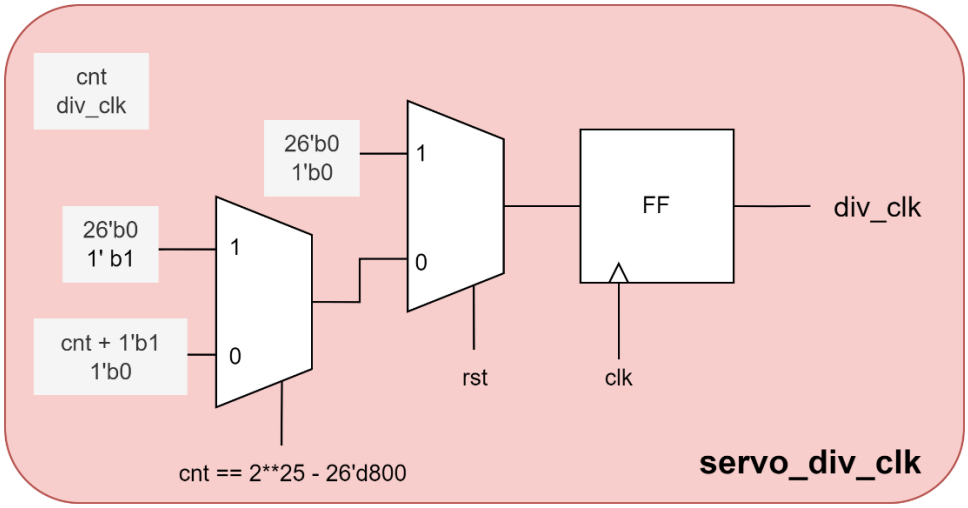

- **div_clk**（用於 joystick）




- **clk_divider_UART**（用於藍芽的 remote ctrl）



- **servo_div_clk**（伺服馬達用的 dclk）

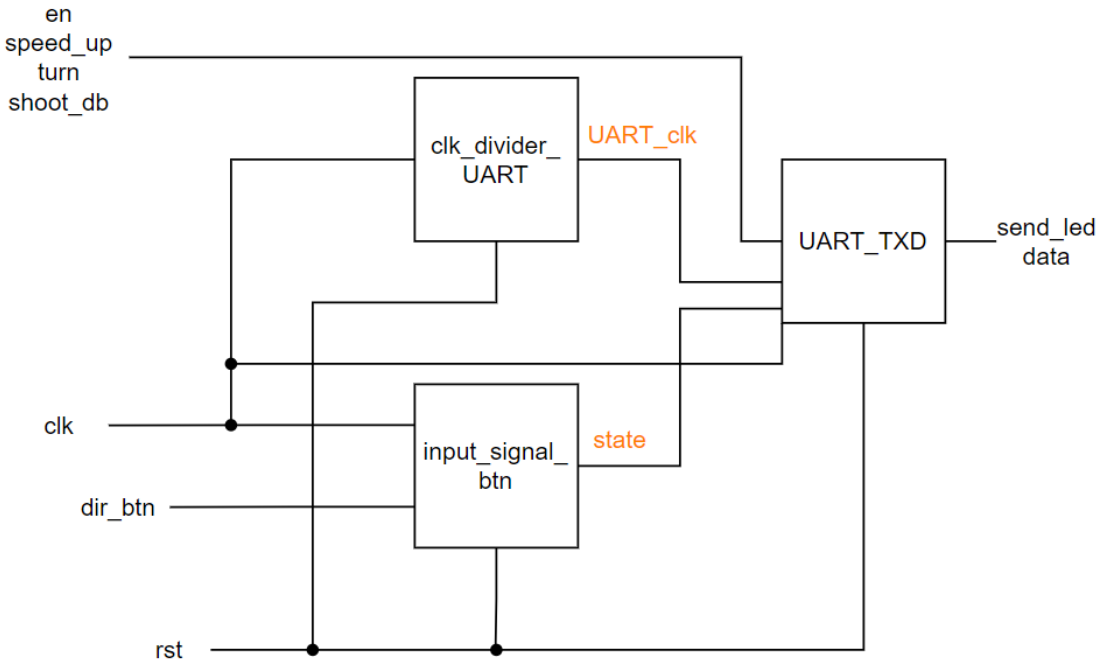



## B. remote control

首先，這塊板子必須實作以下功能：

- 接收 input 訊號並將其轉換為要發射出去的訊號
- 將要傳送的訊號，使用 HC-05 的 protocol 傳送出去

而 top module 的 block diagram 如下，其中 input_signal_btn 是負責轉換 input 訊號；而 clk_divider_UART 與 UART_TXD 兩個 module 則是負責傳送的部分。



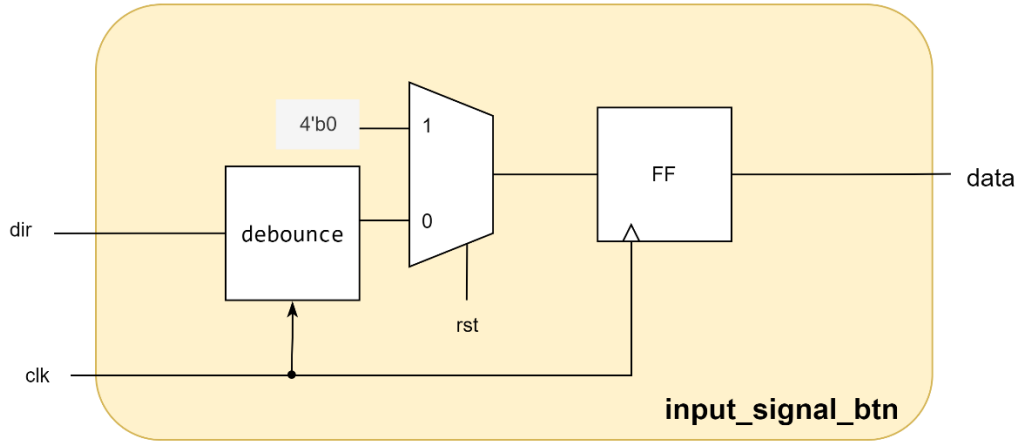
## i. dealing with input signal

我們要接收的 input 訊號有：

- 當前是否要傳輸訊號 (en, switch)
- 當前速度（為慢或快其中一種） (speed_up, switch)
- 砲台旋轉（往左轉、往右轉分別用一個 switch 控制） (turn, switches)
- 發射砲彈 (shoot, button)
- 車子移動方向（前後左右，共四個按鈕） (dir_btn, buttons)

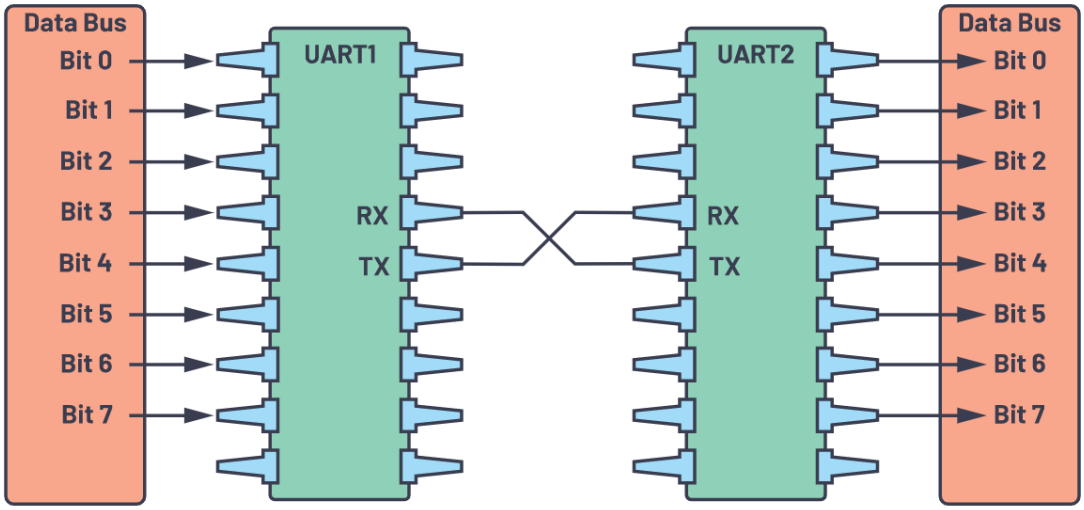
其中 en、speed_up 不須經由特別的轉換。shoot 只要經過 debounce 的處理便也可以直接傳輸（這邊不做 onepulse 是因為會造成時間太短而來不及傳輸出去的問題，因此到接收段那裡才做 onepulse 的處理）。dir_btn 的部分，也只要先做 debounce 便能傳輸。

input_signal_btn 主要只是將 dir_btn 做 debounce 與使其在 reset 時特別設成 4’ b0 而已。特別包成一個 module，原因在於我們原本希望能使用其他東西作為方向的 input，而使用者可以透過 switch 來決定使用什麼東西來操作，因此使用這樣寫法以便於維護與增加新功能，之後會在 problem 這一部分多加介紹。這個 module 的 block diagram 如下：




## ii. send signal

我們要用藍芽 HC-05 來進行資料的傳輸。HC-05 是一種 UART (Universal Asynchronous Receiver/Transmitter)，而 UART 是一個普遍的、用於硬體的 device-to-device communication protocol，屬於非同步的 transmission，可以擁有至多一個 master 與一個 slave。每個 UART 都擁有 Tx (Transmitter) 與 Rx (Receiver) 兩個 signals，而兩個 UART 傳輸的 block diagram 大致如下：




其中左邊為 master，右邊為 slave，兩個 UART 要有相同的 speed 才能互相連線，且 master 要主動找到 slave 去建立連線。為了讓兩個 HC-05 能在供電時自動配對，我們使用了 arduino 的幫助來對它們做設定。

以下是我們使用的 code，這份 code 可以一次對一個 HC-05 進行設定。

在 setup 函式中，我們首先對電腦的 monitor 與藍芽建立連線，兩者的連線速率分別為 9600 bps 與 38400 bps。與 HC-05 連線時，要先按住上面的黑色按鈕再對它供電，這時 HC-05 會緩慢的閃爍 LED 燈（約每兩秒閃一次，一次閃兩秒），表示成功進入 AT 模式，可以繼續做接下來的設定。

接著在 loop 函式中，我們先檢查是否成功連線到電腦的 monitor，如果成功便將輸入讀進來。接著檢查 HC-05 這邊的連線是否成功，如果成功的話便將剛剛讀到的輸入送給 HC-05。

```
#include <SoftwareSerial.h>
// 98D3:31:F69434

SoftwareSerial BT(11, 10); // Rx(HC-05's Tx), Tx(HC-05's Rx)
char val;  // store the input

void setup() {
  Serial.begin(9600);   // connect with serial monitor
  Serial.println("BT is ready!");

  BT.begin(38400);
}

void loop() {
  if (Serial.available()) {
    val = Serial.read();
    BT.print(val);
  }

  if (BT.available()) {
    val = BT.read();
    Serial.print(val);
  }
}
```

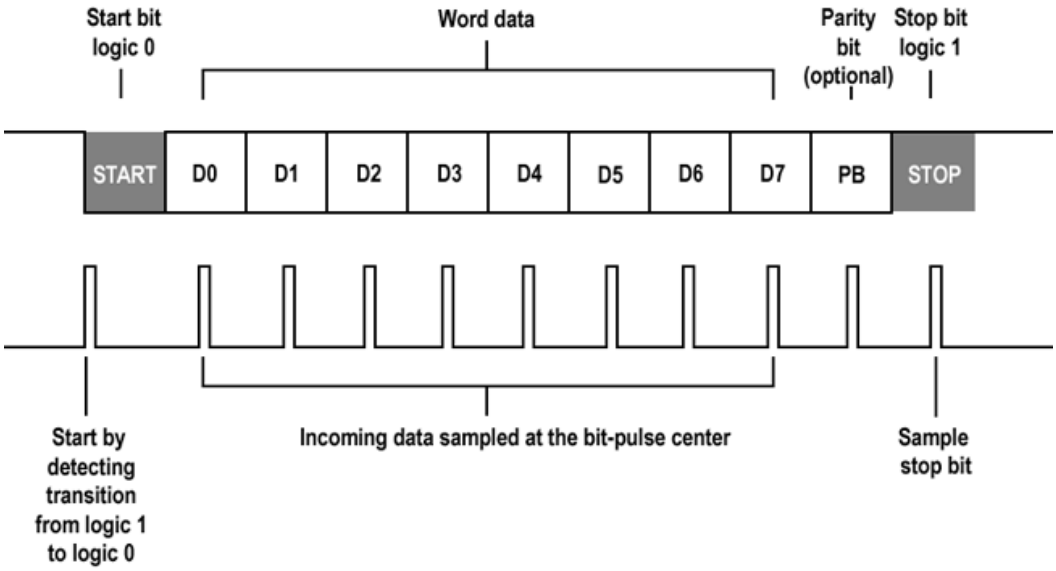
我們可以使用 monitor，對 HC-05 發送 AT 指令，以對它進行設定或是詢問其當前的狀態。我們先設定作為 slave 的 HC-05，對它發送 AT 並得到 OK 的 response，確認連線成功。接下來輸入 AT+ROLE=0，將此模組的 role 設成 slave。再來我們輸入 AT+UART=9600, 0, 0，這項輸入的 3 個

parameters 分別為 baud rate, stop bit 與 parity bit。最後輸入 AT+ADDR?，詢問此模組的 address 並得到其 address 為 98D3:31:F69434，至此 slave 的設定便完成了。

設定 master 時，我們一樣先建立連線、發送 AT 並得到 OK 的 response，確認連線成功。接下來輸入 AT+ROLE=1，將此模組的 role 設成 master。再來我們輸入 AT+UART=9600, 0, 0，與 slave 的設定相同。最後輸入 AT+BIND= 98D3,31,F69434，讓 master 知道他要去找誰做連線，這邊的 address 便是剛剛詢問 slave 得到的 address，特別注意輸入時要將冒號變成逗號。至此，master 的設定也完成了。

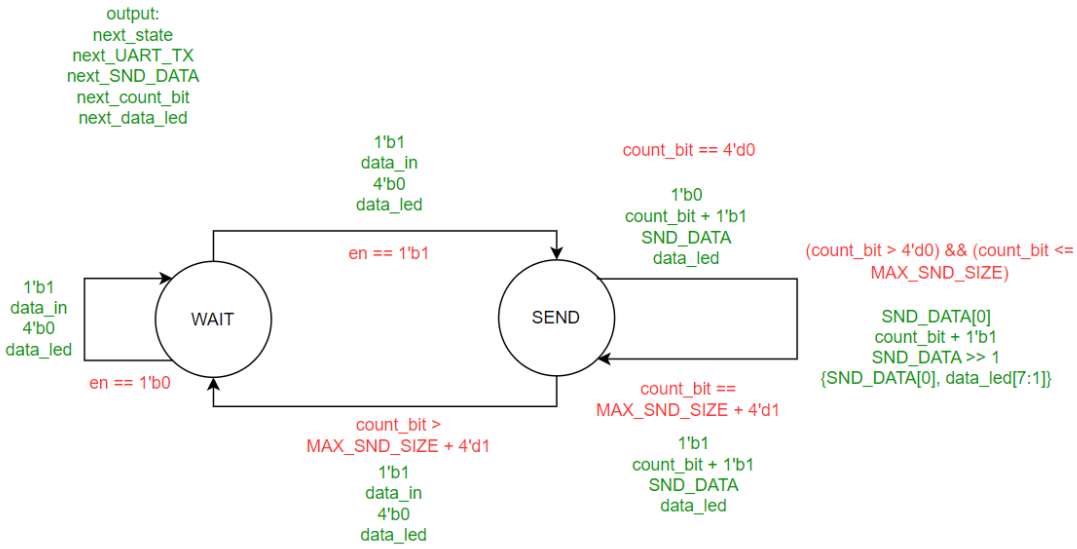
接著我們對兩個藍芽模組進行供電，供電後的模組閃燈方式變為每隔一段時間快閃兩次，表示成功配對。

Data transmission 的部分，要將 data 包成一個 packet 進行傳輸。Packet 中要包含：start bit(1 bit), data(5 ~ 9bits), parity bits(0 ~ 1 bit), stop bits (1 ~ 2bits)，如下圖。而我們自己使用的 packet 為：start bit(1 bit, low), data(8 bits), stop bit(1 bit, high)。




remoteCtrl 這邊負責的是傳輸端的部分，在未傳輸任何資料時，我們讓傳輸端不斷發送連續的 high 訊號，而要傳送時在傳送整個 packet 的資料。負責這部分的 module 是 clk_divider_UART 與 UART_TXD，前者為後者提供符合 HC-05 頻率的 divided clock，後者則負責傳輸。clk_divider_UART 與之前寫過的 divided clock 只有時間長短上的差別，其 block diagram 在 utility 中已提供。

UART_TXD 負責主要傳輸的部分，其實作較為複雜。我們將整體分為兩個 state：WAIT 與 SEND。WAIT 是在等待使用者作出發射訊號的指令（也就是將控制 en 的 switch 設為 1），SEND 便是在做傳輸。state diagram 如下：




在 WAIT state 的時候，如果 en 為 0，則繼續待在 WAIT state 並維護下一個 clock 的 output：輸出的 data (next_UART_TX) 設為 1’ b1、要傳輸的 8 bits data (next_SND_DATA) 設為 data_in、用來計數的 reg (next_count_bit) 設為 4’ d0、記錄當前 reciever 應接收到的資料 (next_data_led) 設為 data_led。而當 en 為 1 時，將下一個 state 設為 send，而其他 output 都與 en 為 0 時相同。

在 SEND state 的時候，先後傳輸 1’ b0(start bit)、SND_DATA(data)、1’ b1(stop bit)。在傳 data 這一部分的時候，我們從最低位開始傳，因此我們不斷讓 next_UART_TX = SND_DATA[0]，並在每次傳輸完一個 bit 的 data 時，讓 SND_DATA 右移 1 bit，這麼一來便可以照順序從最低位傳到最高位。data_led 的部分也是不斷右移，並每次將 SND_DATA[0]，也就是正要傳輸的資料，放到最左邊。整個 data 都傳輸完後，將 next_UART_TX 設為 1’ b1 作為 stop bit，全部都傳輸完後才再回到 WAIT state。這部分的 code 是用 combinational 的方式實作。

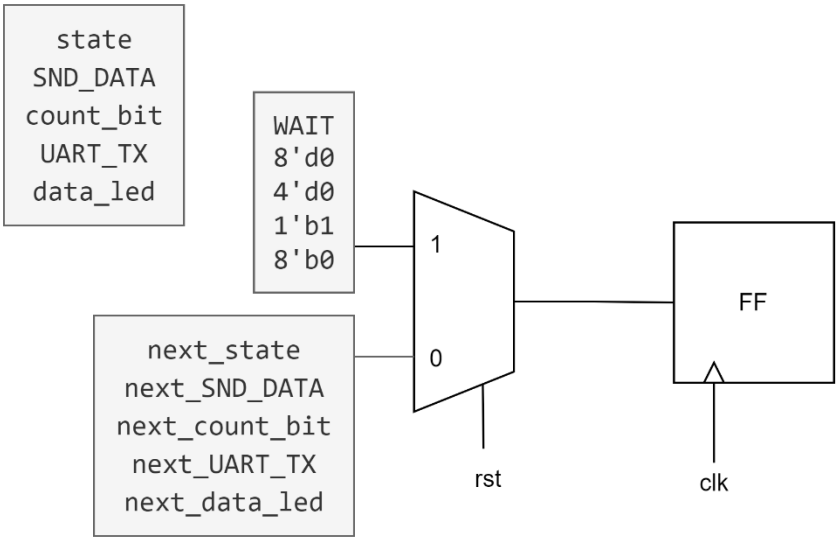
Sequential 的部分，我們維護 state、SND_DATA、count_bit、UART_TX、

data_led 的值，將剛剛在 combinational circuit 中跑出來的結果傳入其中

如下圖。

```
always@(posedge clk) begin
    if(rst) begin
        state <= WAIT;
        SND_DATA <= 8'd0;
        count_bit <= 4'd0;
        UART_TX <= 1'b1;
        data_led <= 8'b0;
    end else begin
        state <= next_state;
        SND_DATA <= next_SND_DATA;
        count_bit <= next_count_bit;
        UART_TX <= next_UART_TX;
        data_led <= next_data_led;
    end
end
```

這個 module 整體的 block diagram 如下：



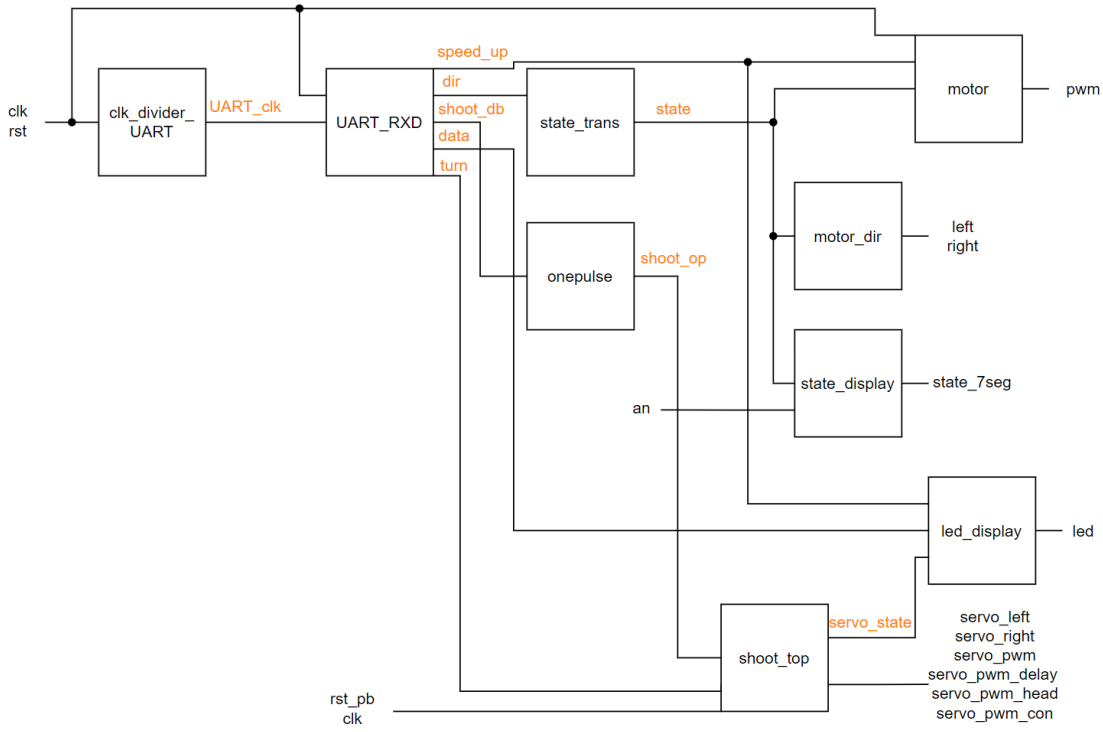

至此我們實作出了所有 remote control 應具備的功能。

## C. car

- 接在車子上的 FPGA 板必須實作以下功能：
- 接收傳輸端給的資料
- 根據接收到的資料決定當前車子的 state

- 依照當前 state 控制車子的 motor
- 依照接收到的資料控制發球機發射
- 依照接收到的資料控制發球機左/右轉動
- (debug 用) 在板子上 output 出當前各種資訊

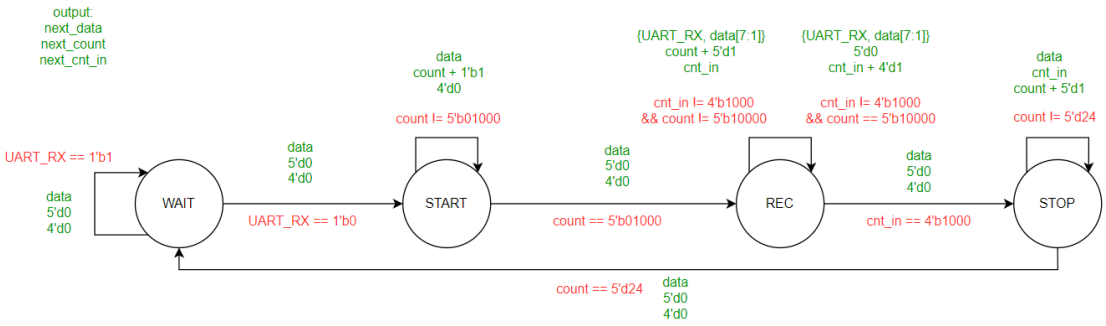
Top module 的 block diagram 如下，其中 clk_divider_UART 負責接收傳輸端給的資料；state_trans 負責決定當前的 state；motor 與 motor_dir 控制車子的 motor；shoot_top 控制有關發球機的所有功能；state_display 與 led_display 在板子上顯示各種資訊。




### i. receive signal

所有的 module 都必須依賴 receive 的 data 來做事。前面提到我們使用藍芽 HC-05 來進行資料的傳輸，這邊要實作的是接收端，也就是 slave 的部分。我們一樣將整個過程分為數個 state，分別為 WAIT、START、REC 與 STOP。這邊的 state transition 要處理的 output 有：next_data

（接收到的 8 bit data）、next_count（計數用的，算每個 bit 是否完整傳完了）、next_cnt_in（計數用的，算當前接收的是 data 的第幾個 bit）。state diagram 如下：




在 WAIT 的部分，如果收到 low 訊號，則表示收到 start bit，next state 要改為 START；反之則留在 WAIT state 繼續等待。不論 next state 為何，我們都將 next_data 設為 data、next_count 及 next_cnt_in 皆設為 0。

在 START 這個部分，我們完整的將 start bit 收完。start bit 會持續 8 個 divided clock，目的是讓接下來的 data 接收不會在 edge 讀值，因此等一下的 data，每一 bit 的資料會持續 16 個 divided clock。發射端沒有處理各個部分是因為恰好被 divided clock 處理掉了，兩端使用的 divided clock 是不一樣的，就是為了處理這件事情。在還沒數滿 8 bits 前，next_state 依舊為 START，next_data、next_cnt_in 分別設為 data 與 0。next_count 則是在 divided clock 為 1’ b1 時增加為 count + 1’ b1。數滿 8 個 bits 時，next_state 便進到 REC，next_data、next_count 及 next_cnt_in 則分別設為 data、0、0。

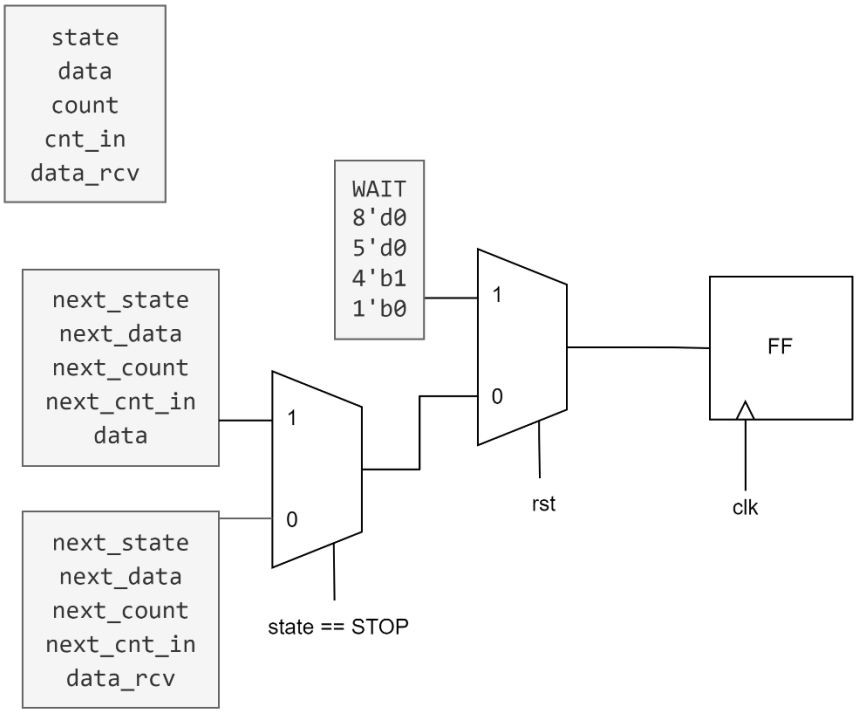
而 REC 便是主要接收 data 的 state。每一 bit 都要完整數完 16 個 divided clock 才算完成。接收的時候由於是由最低位讀到最高位，我

們每次讓 next_data 右移並把新讀到的資料放到最左邊。讀完完整 8 個 bit，也就是 cnt_in == 4’ d8 時，data 便接收完畢，next_state 要設為 STOP，反之 next_state 設為 REC 繼續讀值。

STOP 與 START 很像，接收完 stop bit 再回到 WAIT state 即可，這邊會接收 24 個 divided clock 的 stop bit。

最後 Sequential 的部分，一樣維護 state、data、count、cnt_in、data_rcv（與 data 不同的是，data_rcv 在 REC state 不會改值，並作為 output 接給其他 module）五個值，將剛剛在 combinational circuit 中跑出來的結果傳入其中。

這個 module 整體的 block diagram 如下：




## ii. state transition

我們要將接收到的資料(dir)，轉換成我們設計的 9 個 state 的其中一

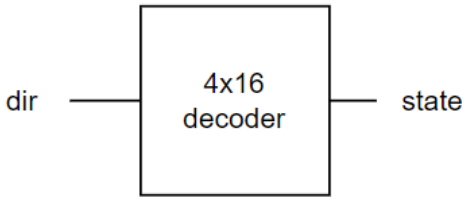
個：stop、go_straight、go_left、go_right、go_back、left_front、right_front、left_back、right_back，分別為停止不動與分別往八方位移動。

同一時間操作者按下的按鈕，共有 2^4=16 種組合，而這 16 種組合與 state 的對應關係如下表：

| direction {front, back, left, right} | state |
|---|---|
| **0000** | stop |
| **0001** | go_right |
| **0010** | go_left |
| **0011** | stop |
| **0100** | go_back |
| **0101** | right_back |
| **0110** | left_back |
| **0111** | go_back |
| **1000** | go_straight |
| **1001** | right_front |
| **1010** | left_front |
| **1011** | go_straight |
| **1100** | stop |
| **1101** | go_right |
| **1110** | go_left |
| **1111** | stop |
| **default** | stop |

概念是若只按一個按鈕，便往該方向前進；若同時按下前後其中一個按鈕與左右其中一個按鈕，便往兩方向合起來的方向前進，例如同時按下左與按下前，便是往左前，state 為 left_front，依此類推；若同時按下前後，或同時按下左右，則相互抵消，視為兩個按鈕都沒有按。

我們在 state_trans 中實作了一個 decoder 來完成上述的功能，其 block diagram 如下：




### iii. motor control

左右兩個 motor 的 output signal 各有三個：pwm(1 bit)、spinning direction(2 bits)。我們要根據 state 來控制 spinning direction 與 speed，給予車子正確的 signal，以讓車子依照我們希望的結果行動。State 與 spinning direction、speed 的對應如下表：

| State | State number | direction (left / right) | speed (left/right) |
|---|---|---|---|
| stop | 0 | 11 / 11 | 0 / 0 |
| go_straight | 1 | 10 / 10 | normal / normal |
| go_left | 2 | 01 / 10 | normal / normal |
| go_right | 3 | 10 / 01 | normal / normal |
| go_back | 4 | 01 / 01 | normal / normal |
| left_front | 5 | 10 / 10 | turn_in / normal |
| right_front | 6 | 10 / 10 | normal / turn_in |
| left_back | 7 | 01 / 01 | turn_in / normal |
| right_back | 8 | 01 / 01 | normal / turn_in |

其中 speed 的 normal 與 turn_in 各為一個特定速度（稍後會說明）、direction 則是對應到下表：

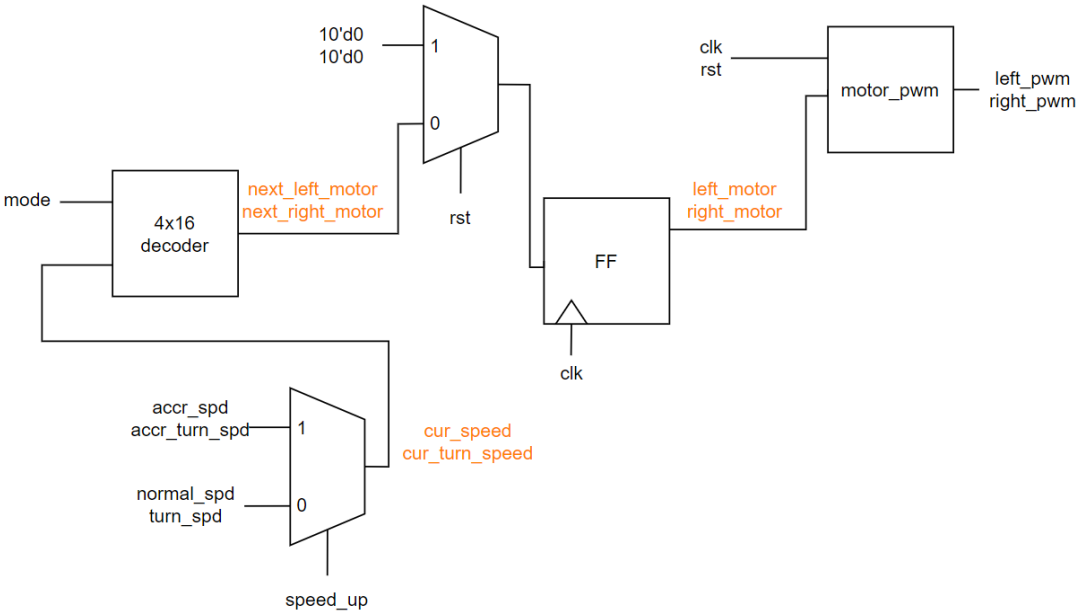
IN1 and IN2 pins control the spinning direction of the motor A, while IN3 and IN4 control motor B.

| Input1 | Input2 | Spinning Direction |
|---|---|---|
| Low(0) | Low(0) | Motor OFF |
| High(1) | Low(0) | Forward |
| Low(0) | High(1) | Backward |
| High(1) | High(1) | Motor OFF |

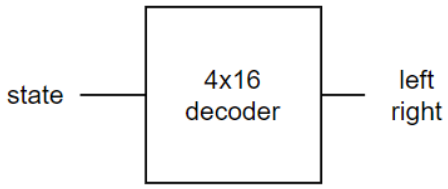
motor 這個 module 負責的是給出 pwm。Generate pwm 的部分在之前

的 lab 就已經有 template 了，會用到 motor_pwm 與 PWM_gen 兩個 module。

我們要做的就是決定 speed，我們用兩個 reg：cur_speed 與 cur_turn_speed 代表上表的 normal 與 turn_in，其代表意義為一般輪子的速度，與轉彎時內輪的速度。首先我們要決定好這兩個 reg 的值：在 speed_up 為 1’ b0 時，要分別將它們設為 normal_spd(10’ d800) 與 turn_spd(10’ d600)；而在 speed_up 為 1’ b1 時，要分別將它們設為 accr_spd(10’ d1023) 與 accr_turn_spd(10’ d723)。這部分使用 sequential 的方式實作。接著我們就只要使用 decoder、上面表格的資訊與剛剛決定好的兩個 reg，根據 state 決定兩邊 motor 的 speed。這個 module 整體的 block diagram 如下：

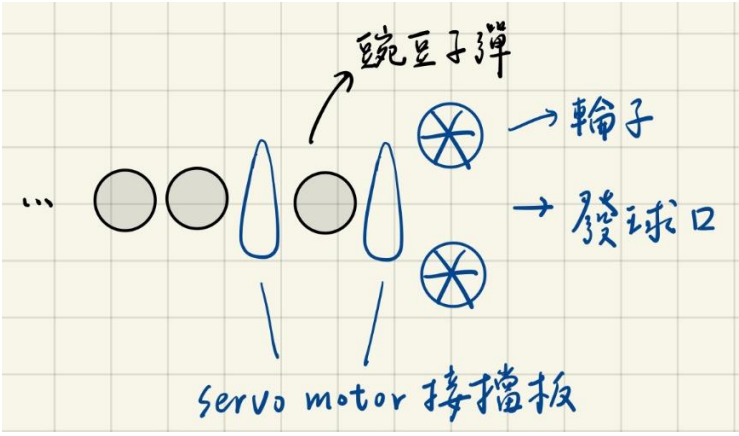



motor_dir 這個 module 負責的是根據 state 決定 spinning direction，我們使用一個 decoder 來實作，block diagram 如下：



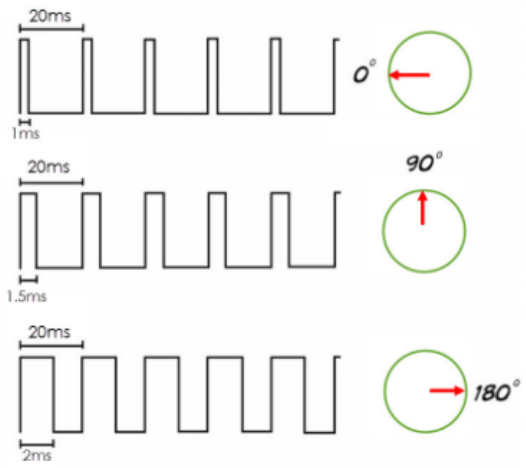

### iv. shoot

控制射擊的部份我們採用發球機加上兩個 servo motor，由於發球機會一次將所有在軌道上的球發出，因此我們需要一些操作讓它一次發射一顆球並具有儲球的功能，示意圖如下。




我們讓上圖中靠近發球口的擋板會先動（水平 -> 垂直），此時原本被它擋住的豌豆子彈便可以往前滾動進而射出，然後我們製造時間差讓另一個檔板也動，此時下一顆球就會往前滾到進到預備發射的位置，在這樣的循環之下就可以達成我們的目標。

要讓 SG90 轉動，就要給它正確的 pwm，首先，無論任何角度，它需要的 period 皆為 20ms，而若要使其轉 90 度，就要在 20ms 之中給它 1.5ms 的 high 訊號（Duty cycle = 1.5/20），波形如下圖。

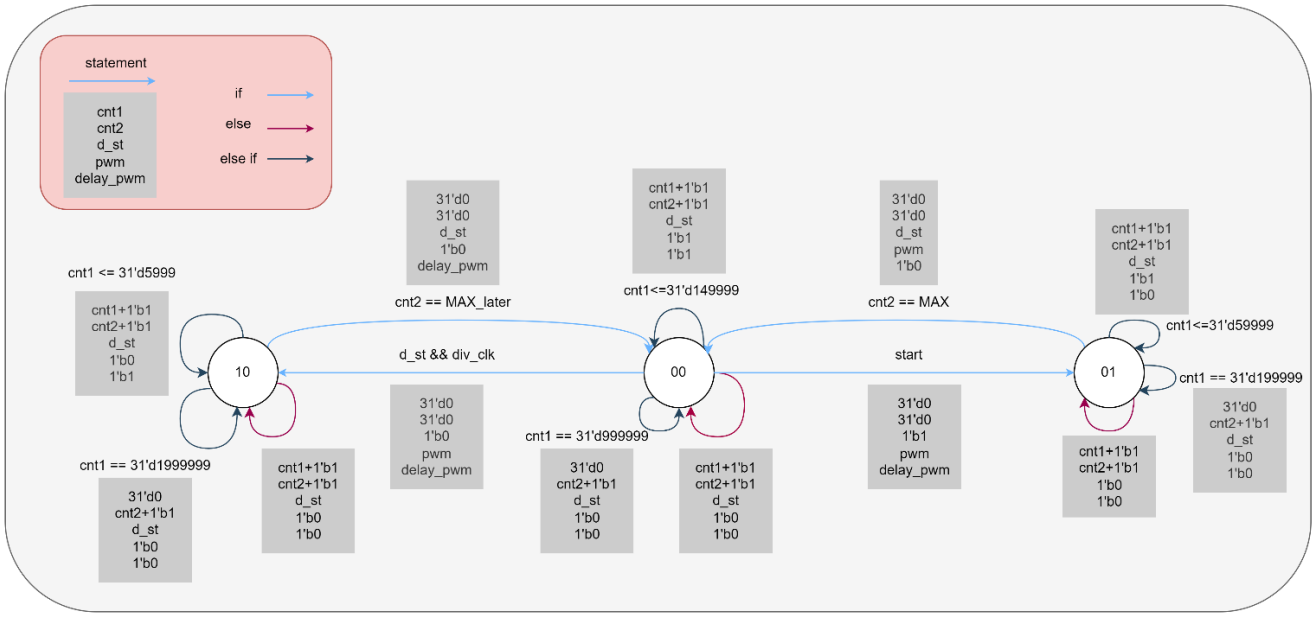



考慮到我們需要轉 90 度就自動回到原位的擋板，又要控制兩個 servo motor，因此我們用 FSM 的方式實作，state 對應到的 Motor behavior 如下圖。

| State | Motor behavior |
|---|---|
| 2'b00 | 回到原位 |
| 2'b01 | 第一個擋板轉 90 度 |
| 2'b10 | 第二個檔板轉 90 度 |

在實作過程中，我們讓他逆時針轉動，即在 state 為 2'b00 時，servo motor 為 90 度的狀態，而 state 為 2'b01 時為 0 度的狀態，確認他可以正常地轉動再復位後，我們發現不知為何依照上方的波型圖設計似乎沒有轉完整的 90 度，因此我們嘗試去改動他的 Duty cycle 讓他符合我們的需求；而為了實現第一個轉完第二個接著轉，我們在按下射擊（start）時，記 d_st <= 1'b1，再判斷 d_st 為 1 且 div_clk 為 1 時進到 2'b10 的 state，此時便完成初步設計。

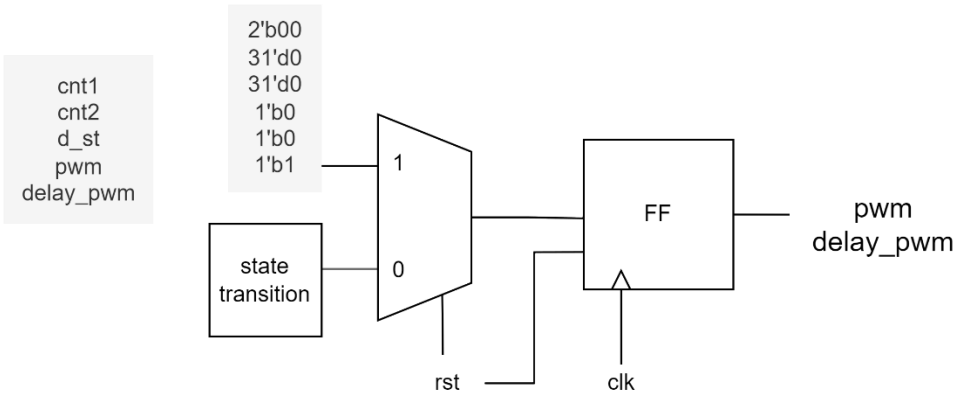
而後在實際測試時，發現如果直接讓他轉 90 馬上復位，對豌豆子彈來說會太快，因此我們去修改兩個伺服馬達各自的 MAX 值，讓他在轉好 90 度時停一下再復位，完成射擊部分的完整設計。控制 servo motor 的 state transition 如下圖。




以第一個轉的馬達為例，code 如下圖。

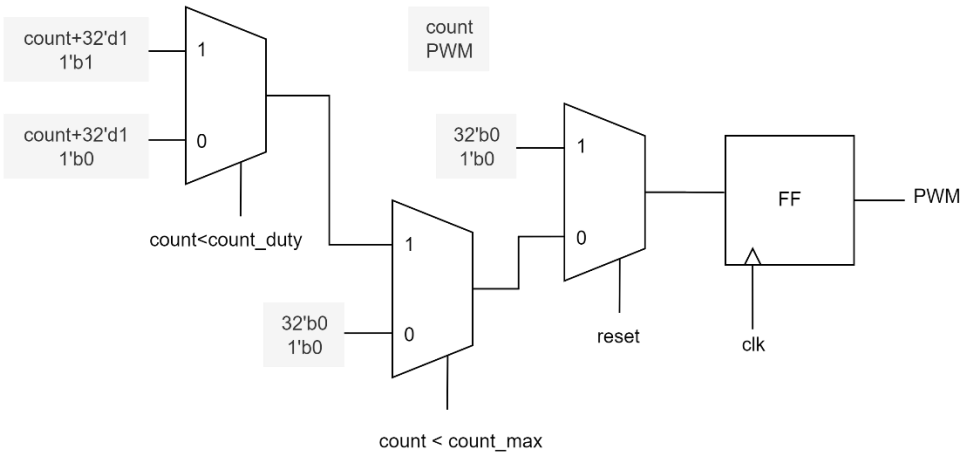
```
2'b01:begin
    d_st <= d_st;
    if(cnt2 == MAX)begin // back to original posotion
        state <= 2'b00;
        cnt1 <= 31'd0;
        cnt2 <= 31'd0;
        pwm <= pwm;
        delay_pwm <= 1'b0;
    end else if(cnt1 <= 31'd59999)begin // ctrl duty cyc
        state <= state;
        cnt1 <= cnt1 + 1'b1;
        cnt2 <= cnt2 + 1'b1;
        pwm <= 1'b1;
        delay_pwm <= 1'b0;
    end else if(cnt1 == 31'd1999999)begin // ctrl period
        state <= state;
        cnt1 <= 31'd0;
        cnt2 <= cnt2 + 1'b1;
        pwm <= 1'b0;
        delay_pwm <= 1'b0;
    end else begin // continue
        state <= state;
        cnt1 <= cnt1 + 1'b1;
        cnt2 <= cnt2 + 1'b1;
        pwm <= 1'b0;
        delay_pwm <= 1'b0;
    end
end
```

射擊部分的 block diagram 如下圖。



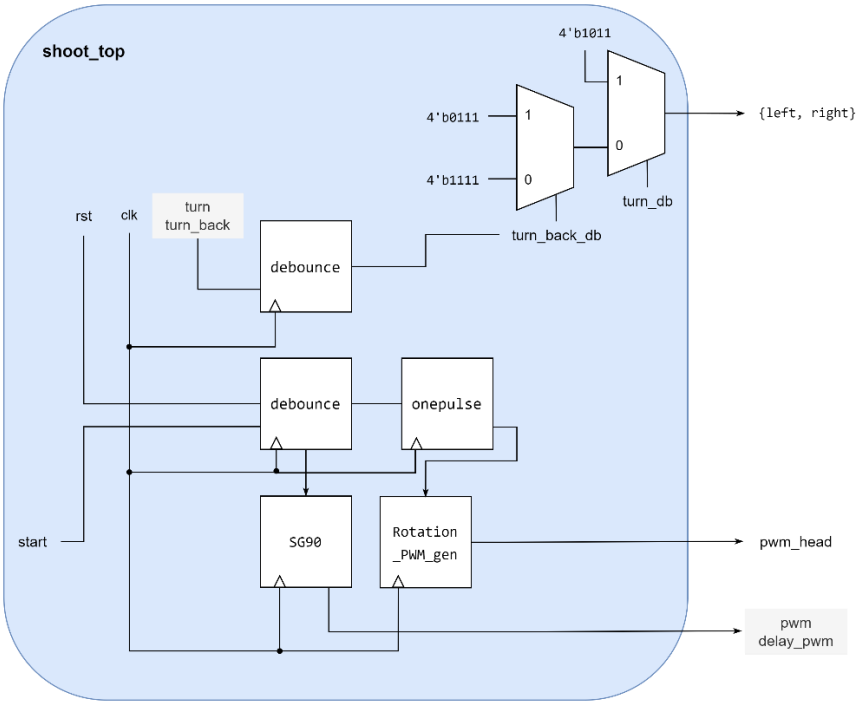

## v. head rotate

要使豌豆的頭可以轉動，我們模仿坦克砲台的轉動模式。一開始我們沒有多想只用了單純的玩具馬達，但發現它在轉動時無法馬上停止，且轉速也太快，因此我們改用減速馬達來操作，這邊我們用的原本是車車上的減速馬達，但它可以給的最低轉速仍然太快，Lab 6 在做車車就有發現它大概 duty 給 600 就跑不動，這次我們測出來 duty 大概 750 馬達就會一下能轉一下不能轉，不太穩定，中間也有想到給它低一點的電壓或許就可以轉慢一點，但仍然到相似的速度就會不穩定，想了一下推測是馬達內齒輪尺數不夠，沒辦法帶動的那麼精確，因此我們最後是用減速比 1:220 的減速馬達，並 modify Lab 6 的 PWM_gen 控制減速馬達，也成功在 duty 為 850 的狀況下達成我們的需求。這部分的 block diagram 如下圖。



```
parameter duty = 32'd850; // decide motor speed
wire [31:0] count_max = 32'd100_000_000 / 32'd25000;
wire [31:0] count_duty = count_max * duty / 32'd1024;
```

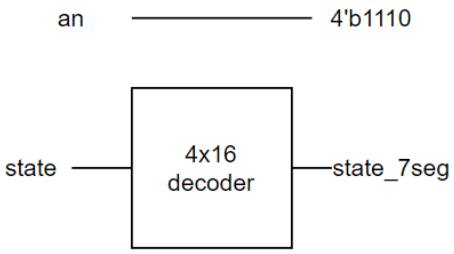
而為了達成可以控制左轉或右轉，我們用 Lab 6 做過的控制輪子前進後退的方式來操作，因此整個豌豆砲台的部份（包含射擊及砲台旋轉）的 block diagram 如下。




## vi. current signal display

這部分只是為了 debug 方便，我們將一些重要資訊，例如當前的 state 與 speed_up 等等，利用 7-segment display 與 LED 燈顯示出來，block diagram 如下。

state_display:




led_display:

in_led ———————— led

至此我們實作出了所有 car 應具備的功能。

# V. Problems

## A. Remote control

為了實現遠端遙控的功能，我們先後嘗試了許多種作法，經過多次實驗失敗與嘗試後，才使用藍芽模組成功建立連線。

### i. Rf link module

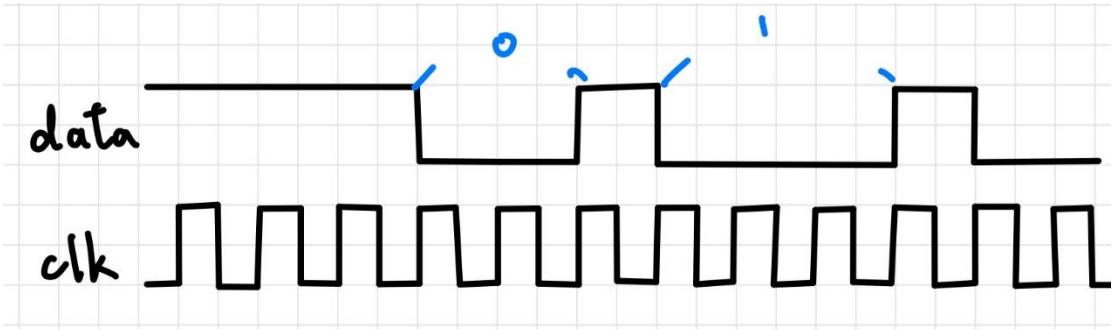
最一開始的時候，我們嘗試使用的是 Rf link 315Mhz module，來做無線傳輸。這個模組傳輸的方式很單純，當接收端發射 high 的時候，接收端就會收到 high 的訊號；而當接收端發射 low 的時候，接收端也就會收到 low。為了延長接收與發射的距離，我們特別另外買了天線，焊接在模組上面。

使用這個 module，要利用單一針腳做 8 bits 的資料傳輸，並且兩塊板子的 clock 並不同步。因此若是直接傳輸 data 會有問題：由於我們沒辦法得知發射端何時開始傳送，接收的板子會無法識別當前收到的 bit 是哪一個 data 的資料。為此，我們設計了兩種 protocol 來嘗試接收與發射資料：

(1)

第一種方法，我們傳輸的方式如下：以 low 訊號的連續長度來表示 0 與 1。傳輸時，若想要傳輸 0，則發送連續 2 個 clk 的 low 訊號；若想要傳輸 1，則發送連續 6 個 clk 的 low 訊號。接收的時候，若接收到小於 4 個連續的 low 訊號，則判斷接收到的是 0；若接收到大於等於 4 個連續的 low 訊號，則判斷接收到的是 1。

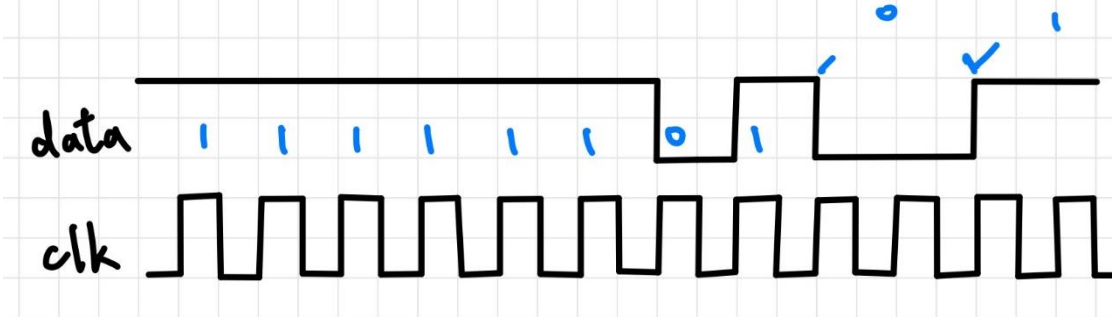
而在最一開始，我們發送連續的 high 訊號表示尚未開始傳輸。對於接收端，接收到第一個 low 訊號才是資料傳輸的開始。傳送的方式如下圖所示：




這個方法在我們使用實體的杜邦線傳輸時會是正確的。然而，這個方法沒有任何的容錯，更改任何一個 bit，尤其是 high 的部分，都會造成傳輸錯誤，因此這個方案最後並沒有被採納。

(2)

第二種方法，我們直接傳輸 data 本身的資料（data 為 1 就傳送 1、data 為 0 就傳送 0）。而在最一開始、還未開始傳送資料的時候，我們讓發射端一直傳送連續的 high 訊號。一旦要開始傳輸，我們便發送 11111101，共 8 個 bits，作為開始傳輸的訊號，隨後便開始傳送資料。而對於接收端，在還未開始傳輸時，它便會不斷偵測是否接收到 11111101，也就是開始傳輸的訊號。一旦接收之後，便讓 counter 開始計算，以辨別當前接收到的是誰的 data。傳送的方式如下圖所示：



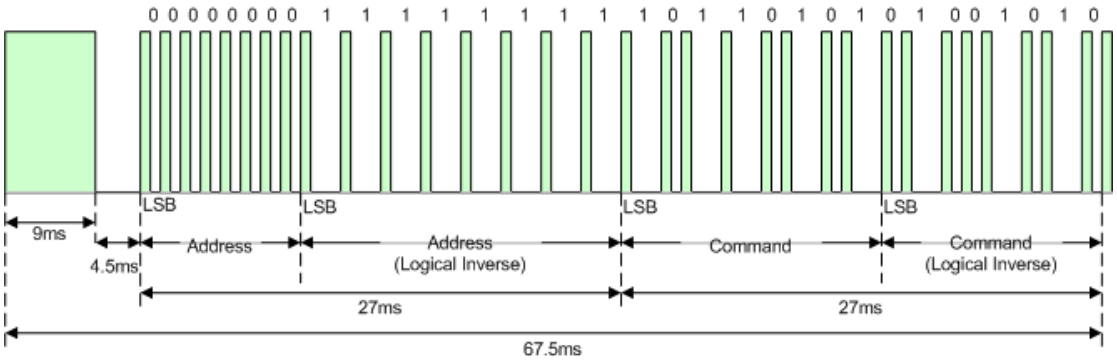
同樣，這個方法在我們使用實體的杜邦線傳輸時會是正確的。然而由於訊號過於不穩定，依舊常常造成傳輸錯誤，因此這個方案最後也沒有被採用。

總之，由於 Rf link module 實在太不穩定，易被干擾。尤其在模組有位移時，經常造成傳輸錯誤，這樣便沒辦法使用此模組傳送資料給車子上的 FPGA 板，不符合我們的需求。因此，我們最後放棄此模組，並開始尋找其他替代方案。

## ii. IR transmitter

我們嘗試的第二個模組是 38kHz 紅外線接收/發射模組。雖然此模組要求傳輸時不能有遮蔽物干擾且可傳輸距離較短，但經過評估後我們認為即使如此，這個模組仍能達成我們的目標，因此我們決定嘗試看看。

紅外線模組有自己定義好的 protocol，大致長這樣：




會先有 9ms 的 high 接 4.5ms 的 low 作為開頭，接著是 receiving device 的 8-bit address、address 的 logical inverse、8-bit command、command 的 logical inverse。最後是 562.5μs 的 high 作為結束信號。

然而在實際測試的時候，訊號的接收並不穩定。在非常短的距離內

（≤15cm）且接收端對準發射端的情況下，接收與傳輸是穩定的。然而當上述任一條件不滿足，傳輸失敗的機率便大大的提高。傳輸距離過短的限制不滿足我們遙控車子所需要的條件，因此最後我們也放棄使用該模組，轉而嘗試藍芽模組。

## B. Joystick

許多遙控車的遙控器都是使用搖桿（蘑菇頭），來控制車子的方向。因此我們一開始也希望可以使用蘑菇頭來遙控車子。蘑菇頭的輸出接腳有三個：VRx、VRy、sw，其中 VRx 與 VRy 代表的是搖桿當前位置的 x 座標及 y 座標，sw 則是搖桿是否被按下。處理蘑菇頭最為棘手的便是 VRx 與 VRy 的處理，由於兩者代表的是座標位置，因此使用的是類比訊號。Basys3 最左下的 Pmod connector 能夠對類比訊號做處理，且官方就有提供 sample code，code 的功能是輸入兩個類比訊號（電壓值），輸出它們的電壓差。

然而，雖然有已經實作好的 code，我們還是面臨許多問題。第一個問題是，這份 code 只能處理 output 介於 0V 至 1V 的情況，因此我們要將蘑菇頭原本的輸出電壓：0V~5V，轉換至 0V~1V 內。經過多次的嘗試，我們在電路中對蘑菇頭的供電反著串聯一個 1.5V 的電池，降低輸入電壓，成功解決這個問題。

第二個問題是，不知道官方的 code 是否有誤，但輸入與輸出間的對應關係與官方說明並不相符。不同 pair 的輸出值理應不互相干擾，然而我們與別組都有遇到訊號會互相干擾的問題。經過多次嘗試，我們找到一個恰好不會被干擾的接法，成功得到我們要的輸出。

最後一個問題，也是我們沒有解決的問題，便是類比轉換的這個過程太久，導致我們在不同 clock 要拿出不同 data（VRx 或 VRy）時，轉換的過程會有一點點延遲，使得 x 座標吃到 VRy、y 座標吃到 VRx 的值。此項誤差會導致車子在行進時走走停停，而在經過多次嘗試後我們也沒有精準的得到延遲的 clock 數，因此沒有將這部分實作出來。

## VI. Conclusion

終於來到最後的 Final Project，一開始以為做起來難度不會太高感覺也蠻好玩的，唯一需要擔心的就是我們會不會把它玩壞，結果除了我快要把它玩壞之外完全猜錯。

從一開始只有一個自動發球機到可以把它整個組起來、可以遠端遙控，我們的確花了不少時間，需要的設計除了 code 還要找出適當的硬體組合出我們的需求；遇到的各種挫折有時候也讓人很迷惘，這邊先謝謝 dream lab，真的是個通宵的好地方。這份 project 除了讓我們將整個學期學到的知識運用上去之外，也讓我們對一些電子產品及模組有更多了解，從一開始拿到硬體設備無從下手，到藉由各種 data sheet 及查到的資料能夠完全了解進而運用它，其中最讓人頭痛的是硬體好難 debug，需要檢查 code、接線是否正確、如果有什麼東西沒反應是在哪個過程出錯等等，甚至花好多時間結果確認出硬體是壞的，或者有些地方跟我們一開始的認知不一樣，例如藍芽模組的 Tx 是接 Rx，Rx 接 Dx，或者有些硬體似乎與 Data sheet 的說明有些落差，有些地方仍然是我們抱著試試看的心理設計出來的，不少時間我們也會懷疑自己的 code 是不是一隻用頭飛的鴿子（如下圖）。


這次我們嘗試了很多東西，對他們了解之後進行應用，這樣基本上從 0 開始的過程到成功做出想要的東西的確很有成就感，第一次能夠讓車子在我們的遙控下移動、轉頭、發射，甚至是很酷的邊跑邊發射，那種興奮真的很讓人印象深刻（結果它被我們測一測就玩到快壞了）。

最後，謝謝這學期教授與助教的教導及協助，讓我們認識及學習到了許多東西，也謝謝與我們討論問題的同學，讓我們可以一步步完成這學期的每個 Lab 及最後的 Final Project。

## VII. Reference


- https://www.engineersgarage.com/servo-motor-sg90-9g-with-89c51-microcontroller/
- https://www.varitron.com.tw/faq_show.asp?seq=38&title=%E6%B8%9B%E9%80%9F%E6%A9%9F%E9%A6%AC%E9%81%94%E7%9A%84%E7%94%A8%E9%80%94

- https://www.youtube.com/watch?v=vOWejHNixhI&t=3s
- https://zh.wikipedia.org/zh-tw/UART
- https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html
- https://www.youtube.com/watch?v=EEtvbv3itYM&ab_channel=%E9%BB%83%E7%B4%B9%E5%B3%AF
- https://www.teachmemicro.com/hc-05-bluetooth-command-list/
- https://techdocs.altium.com/display/FPGA/NEC+Infrared+Transmission+Protocol