Final Project report

組員：110062221李品萱

110062213唐翊雯

**Peatank**

**目錄**

* **Introduction**
* **Motivation**
* **Material**
* **System specification**
* **remote control** 
  + - **dealing with input signal**
    - **send signal**
* **car**
  + - **receive signal**
    - **state transition**
    - **motor control**
    - **shoot**
    - **head rotate**
    - **current signal display**
* **Problems**
* **Conclusion**
* **Reference**

1. **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。

一張含有 文字, 室內, 牆, 凌亂 的圖片

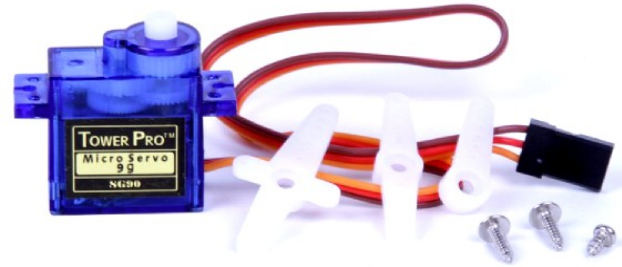
自動產生的描述

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。

1. **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。

1. **Material**
2. **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 |

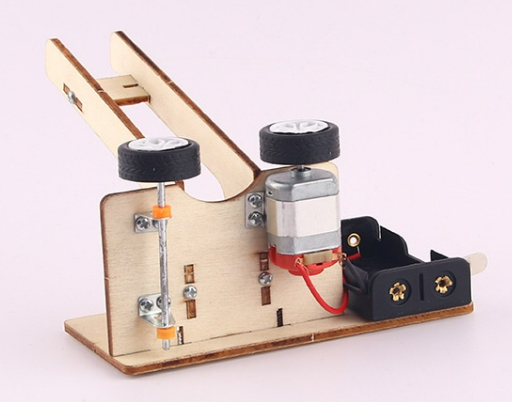
1. **tt減速機**

一張含有 黃色 的圖片

自動產生的描述

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。

1. **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。

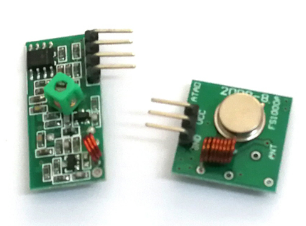
1. **HC-05藍芽模組**

****

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 |

1. **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 |

1. **ir transmitter**

**一張含有 文字, 電子用品, 電子 的圖片

自動產生的描述**

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 |

1. **KY-023 Joystick**

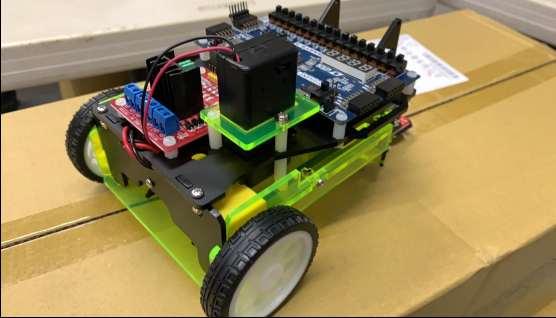
**一張含有 黑色, 室內 的圖片

自動產生的描述**

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 |

1. **Car (from Lab 6)**

****

1. **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。

1. **Utility**

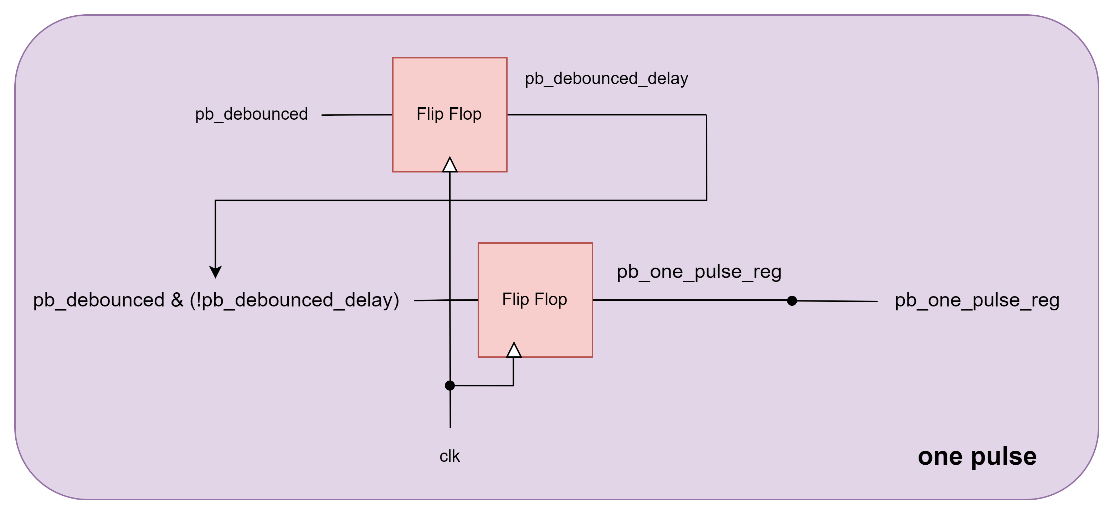
首先我們將一些utility module整理成utility.v，各module如下。

* **debounce**

一張含有 文字, 時鐘 的圖片

自動產生的描述

* **onepulse**



* **div\_clk**（用於joystick）

**一張含有 文字, 時鐘 的圖片

自動產生的描述**

* **clk\_divider\_UART**（用於藍芽的remote ctrl）

**一張含有 文字, 時鐘 的圖片

自動產生的描述**

* **servo\_div\_clk**（伺服馬達用的dclk）

**一張含有 文字, 時鐘 的圖片

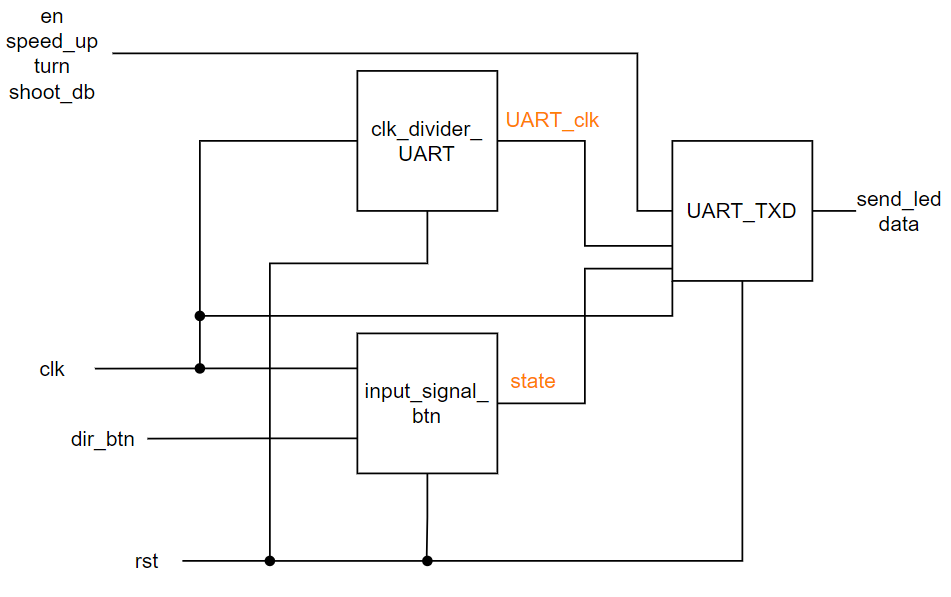
自動產生的描述**

1. **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。



* + 1. **dealing with input signal**

我們要接收的input訊號有：

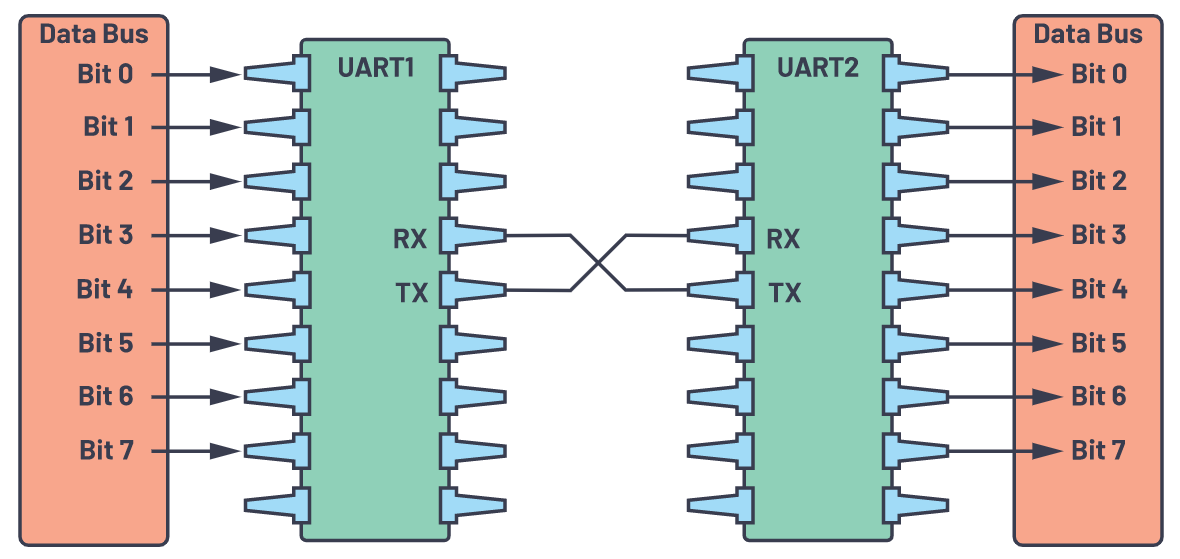
* 當前是否要傳輸訊號 (en, switch)
* 當前速度（為慢或快其中一種） (speed\_up, switch)
* 砲台旋轉（往左轉、往右轉分別用一個switch控制） (turn, switches)
* 發射砲彈 (shoot, button)
* 車子移動方向（前後左右，共四個按鈕） (dir\_btn, buttons)

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：

// add diagram

* + 1. **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。

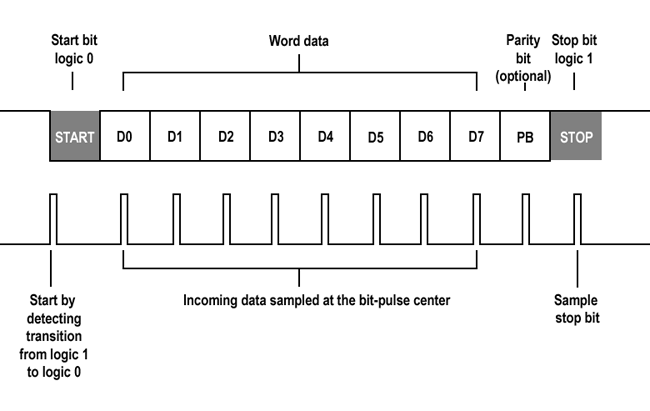


我們可以使用monitor，對HC-05發送AT指令，以對它進行設定或是詢問其當前的狀態。我們先設定作為slave的HC-05，對它發送AT並得到OK的response，確認連線成功。接下來輸入AT+ROLE=0，將此模組的role設成slave。再來我們輸入AT+UART=9600, 0, 0，這項輸入的3個parameters分別為baud rate, stop bit與parity bit。最後輸入AT+ADDR?，詢問此模組的address並得到其address為98D3:31:F69434，至此slave的設定便完成了。

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如下：

// add diagram

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：

// add diagram

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的方式寫的，block\_diagram如下：

// add diagram

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，這部分的block diagram如下：

// add diagram

這個module整體的block diagram如下：

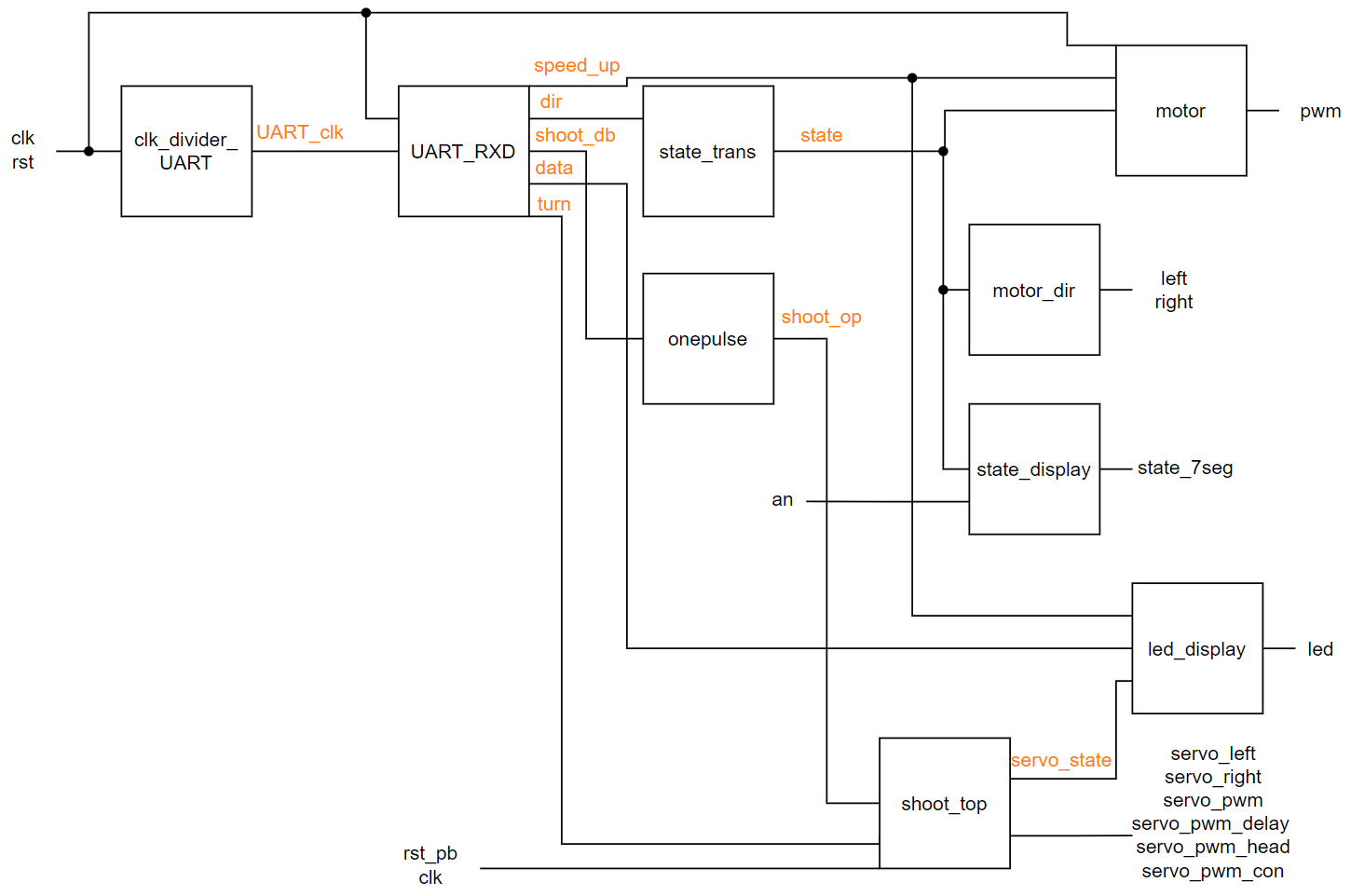
// add diagram

至此我們實作出了所有remote control應具備的功能。

1. **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。



1. **state transition**

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：

// add diagram

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。

整體的block diagram如下：

// add diagram

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，這部分的block diagram如下：

// add diagram

這個module整體的block diagram如下：

// add diagram

1. **state transition**

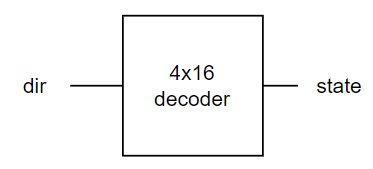
我們要將接收到的資料(dir)，轉換成我們設計的9個state的其中一個：stop、go\_straight、go\_left、go\_right、go\_back、left\_front、right\_front、left\_back、right\_back，分別為停止不動與分別往八方位移動。

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：

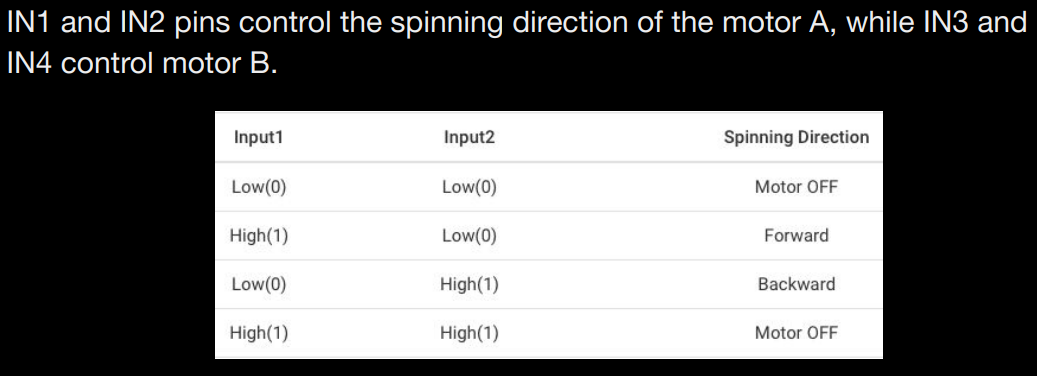


1. **motor control**

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 |

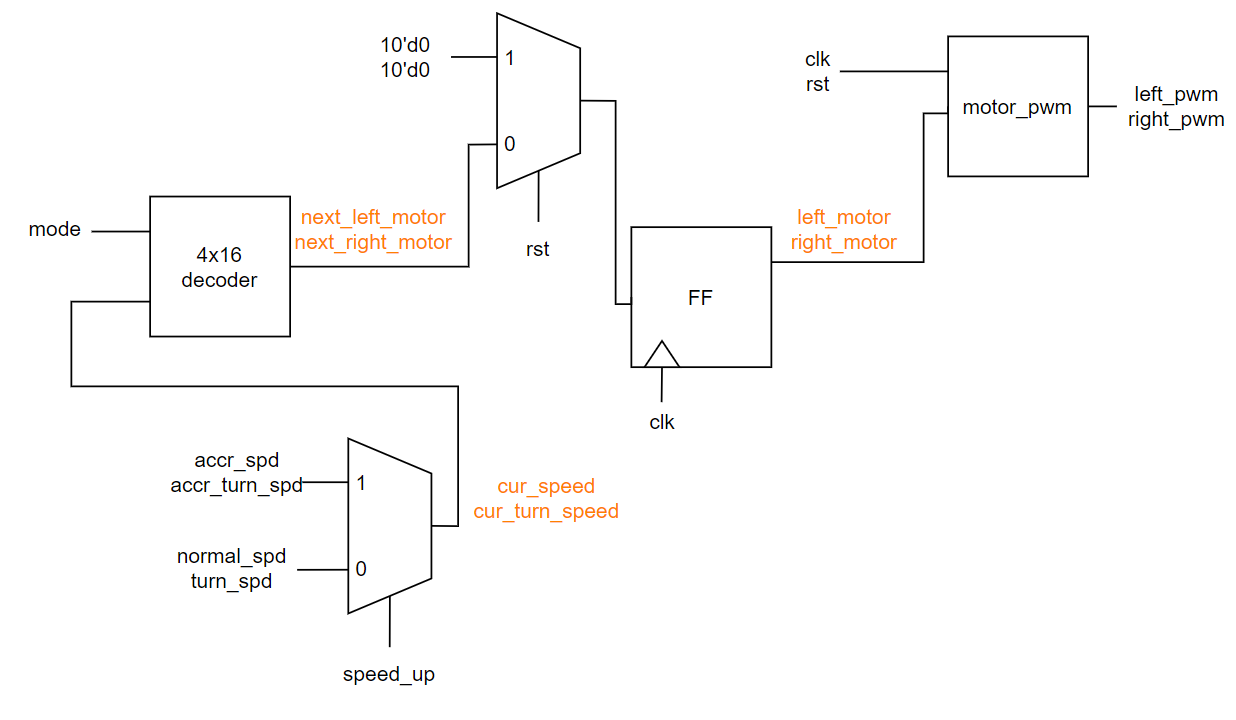
其中speed的normal與turn\_in各為一個特定速度（稍後會說明）、direction則是對應到下表：



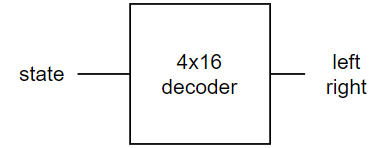
motor這個module負責的是給出pwm。Generate pwm的部分在之前的lab就已經有template了，會用到motor\_pwm與PWM\_gen兩個module，它們的block diagram如下：

// add diagram

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：



1. **shoot**

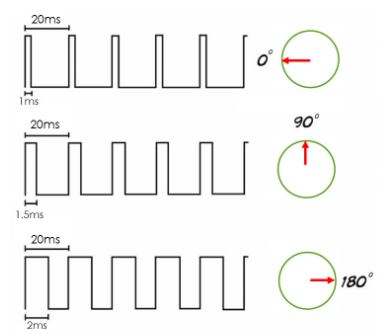
控制射擊的部份我們採用發球機加上兩個servo motor，由於發球機會一次將所有在軌道上的球發出，因此我們需要一些操作讓它一次發射一顆球並具有儲球的功能，示意圖如下。

一張含有 文字, 白板 的圖片

自動產生的描述

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。

一張含有 文字, 時鐘 的圖片

自動產生的描述

以第一個轉的馬達為例，code如下圖。

一張含有 文字 的圖片

自動產生的描述

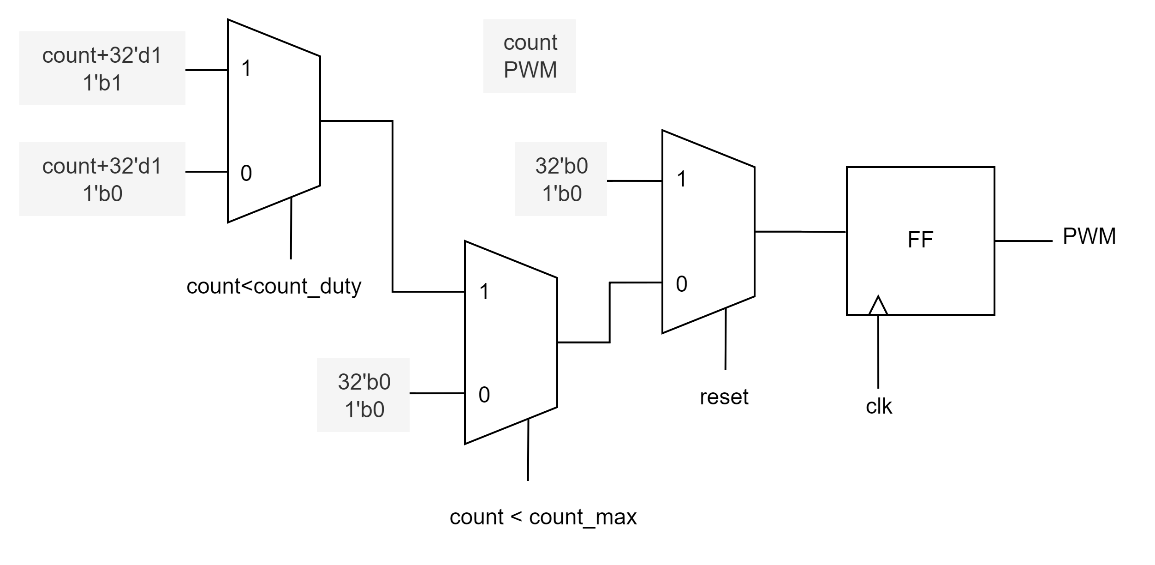
射擊部分的block diagram如下圖。

一張含有 文字 的圖片

自動產生的描述

1. **head rotate**

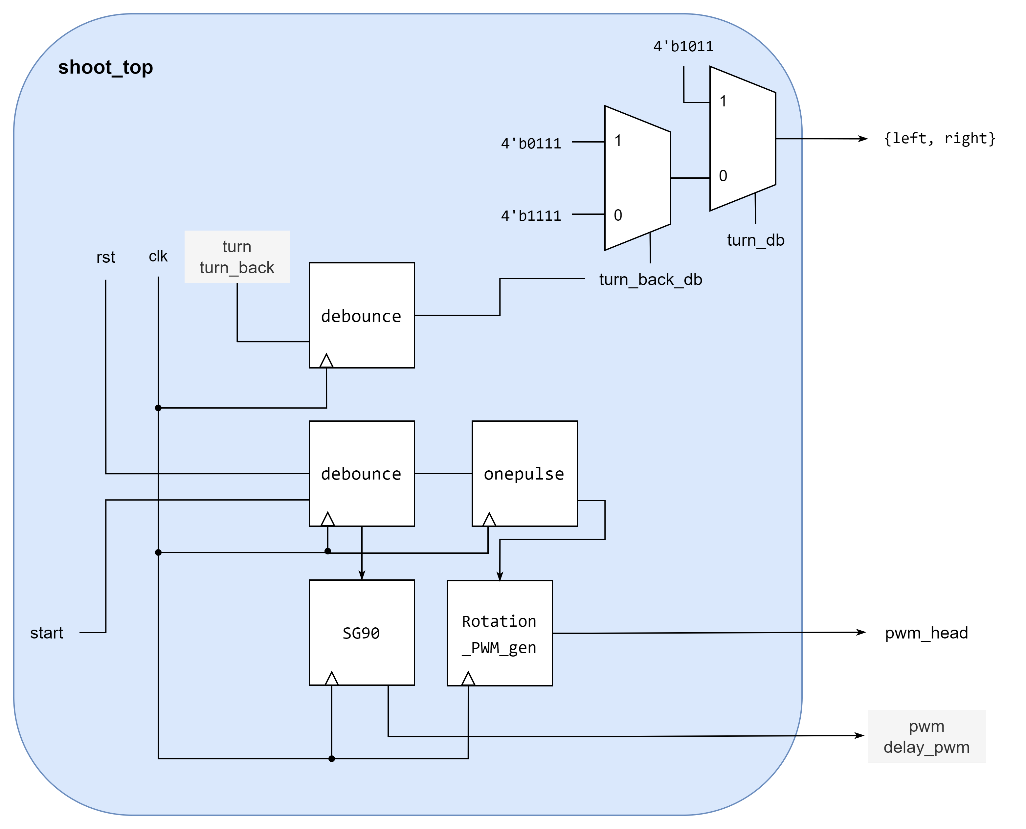
要使豌豆的頭可以轉動，我們模仿坦克砲台的轉動模式。一開始我們沒有多想只用了單純的玩具馬達，但發現它在轉動時無法馬上停止，且轉速也太快，因此我們改用減速馬達來操作，這邊我們用的原本是車車上的減速馬達，但它可以給的最低轉速仍然太快，Lab 6在做車車就有發現它大概duty 給600就跑不動，這次我們測出來duty大概750馬達就會一下能轉一下不能轉，不太穩定，中間也有想到給它低一點的電壓或許就可以轉慢一點，但仍然到相似的速度就會不穩定，想了一下推測是馬達內齒輪尺數不夠，沒辦法帶動的那麼精確，因此我們最後是用減速比1:220的減速馬達，並modify Lab 6的PWM\_gen控制減速馬達，也成功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



一張含有 文字 的圖片

自動產生的描述

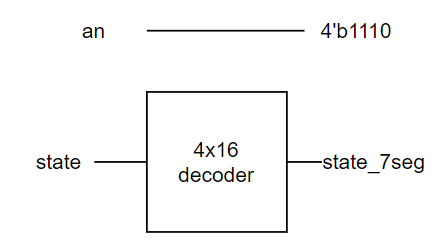
而為了達成可以控制左轉或右轉，我們用Lab 6做過的控制輪子前進後退的方式來操作，因此整個豌豆砲台的部份（包含射擊及砲台旋轉）的block diagram如下。



1. **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:



至此我們實作出了所有car應具備的功能。

1. **Problems**
2. **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。

1. **Rf link module**

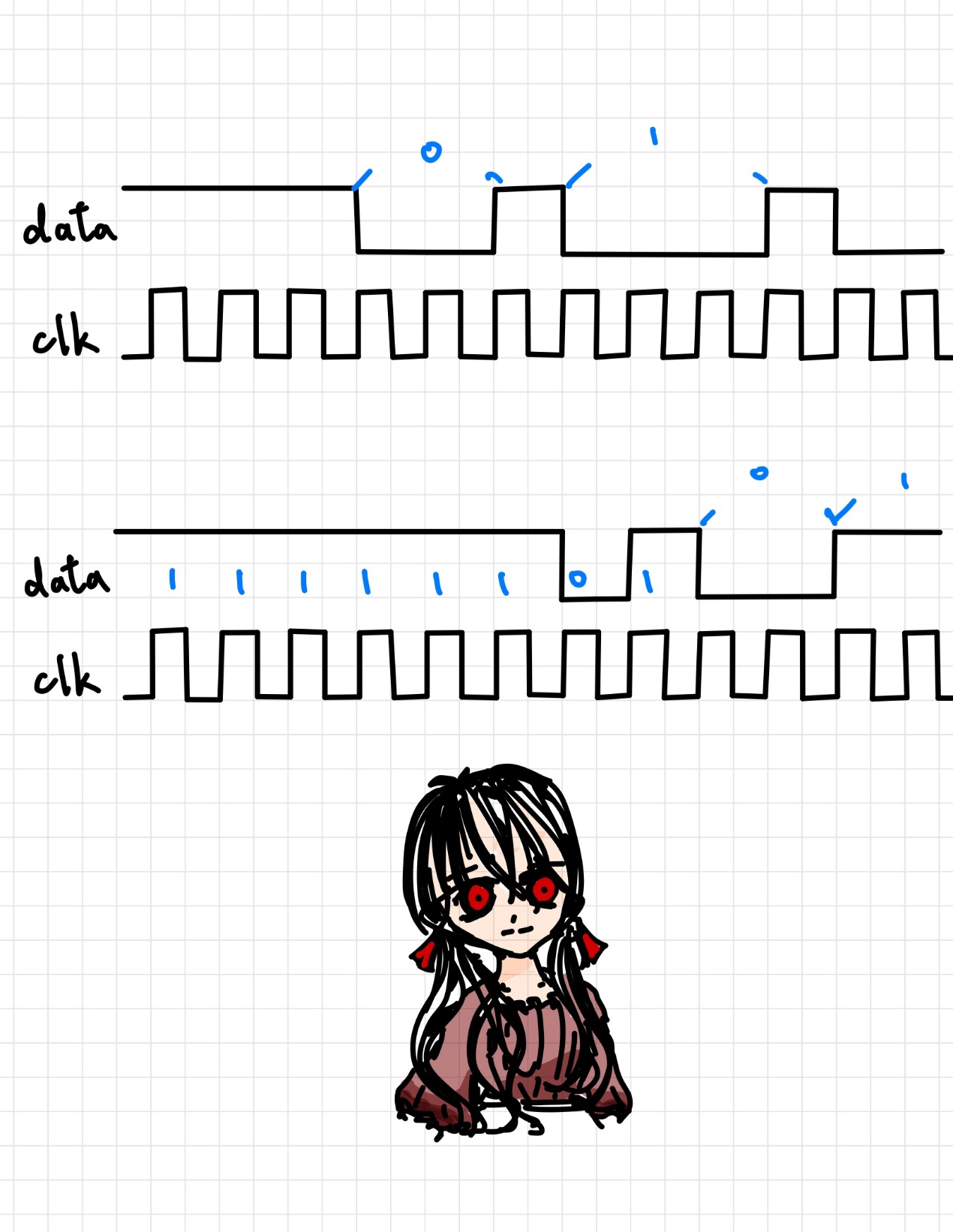
最一開始的時候，我們嘗試使用的是Rf link 315Mhz module，來做無線傳輸。這個模組傳輸的方式很單純，當接收端發射high的時候，接收端就會收到high的訊號；而當接收端發射low的時候，接收端也就會收到low。為了延長接收與發射的距離，我們特別另外買了天線，焊接在模組上面。

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。

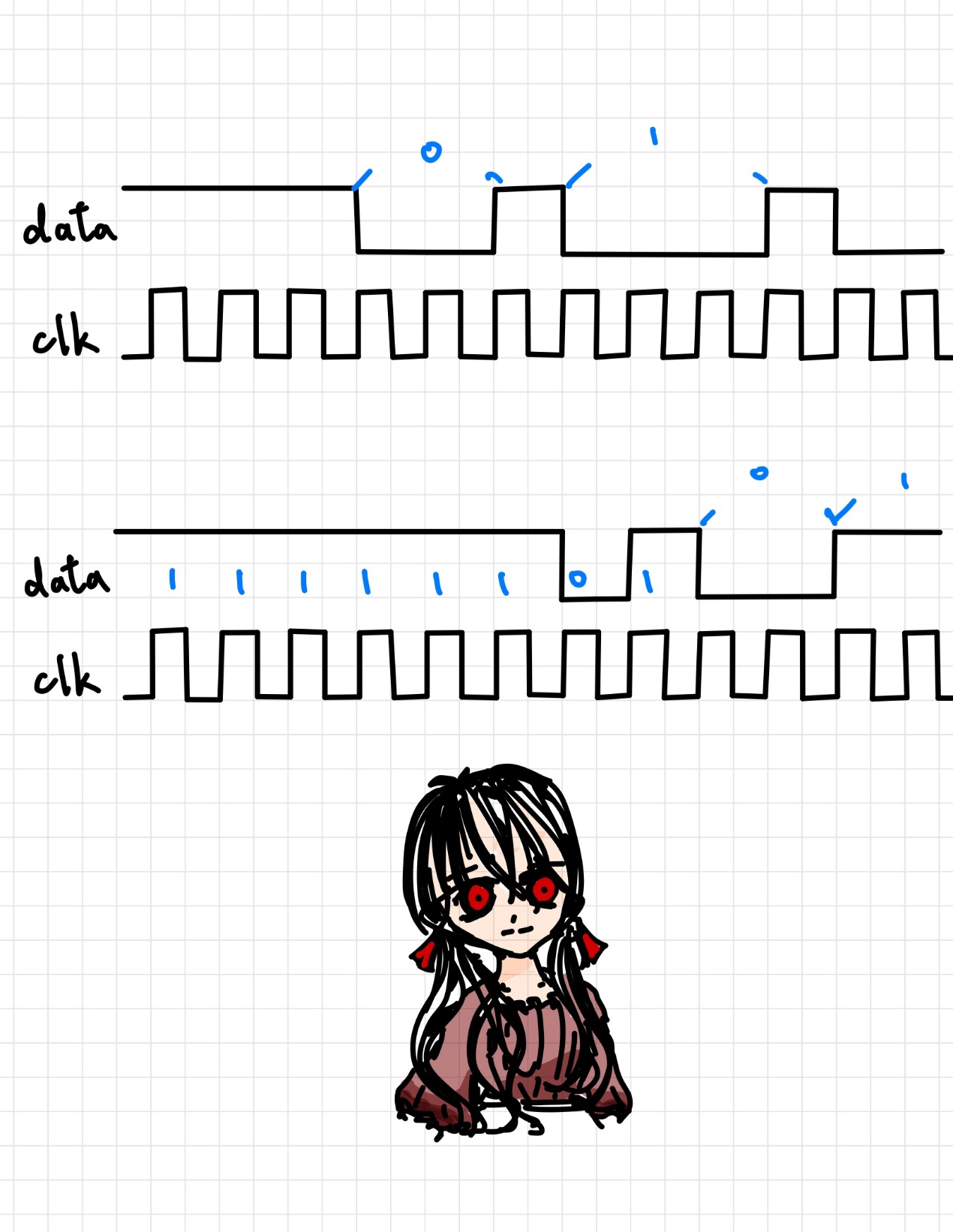
而在最一開始，我們發送連續的high訊號表示尚未開始傳輸。對於接收端，接收到第一個low訊號才是資料傳輸的開始。傳送的方式如下圖所示：



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：



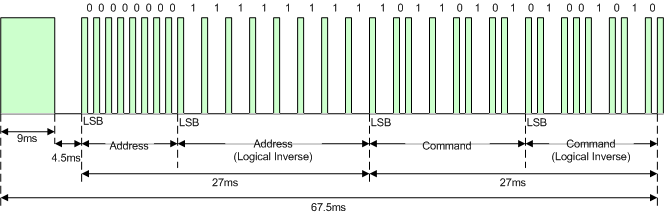
同樣，這個方法在我們使用實體的杜邦線傳輸時會是正確的。然而由於訊號過於不穩定，依舊常常造成傳輸錯誤，因此這個方案最後也沒有被採用。

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。

1. **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。

1. **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。

1. **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>