Final Project Report - Bartender

107062240 林柏均
107062318 李俊逸

Introduction

作品外觀

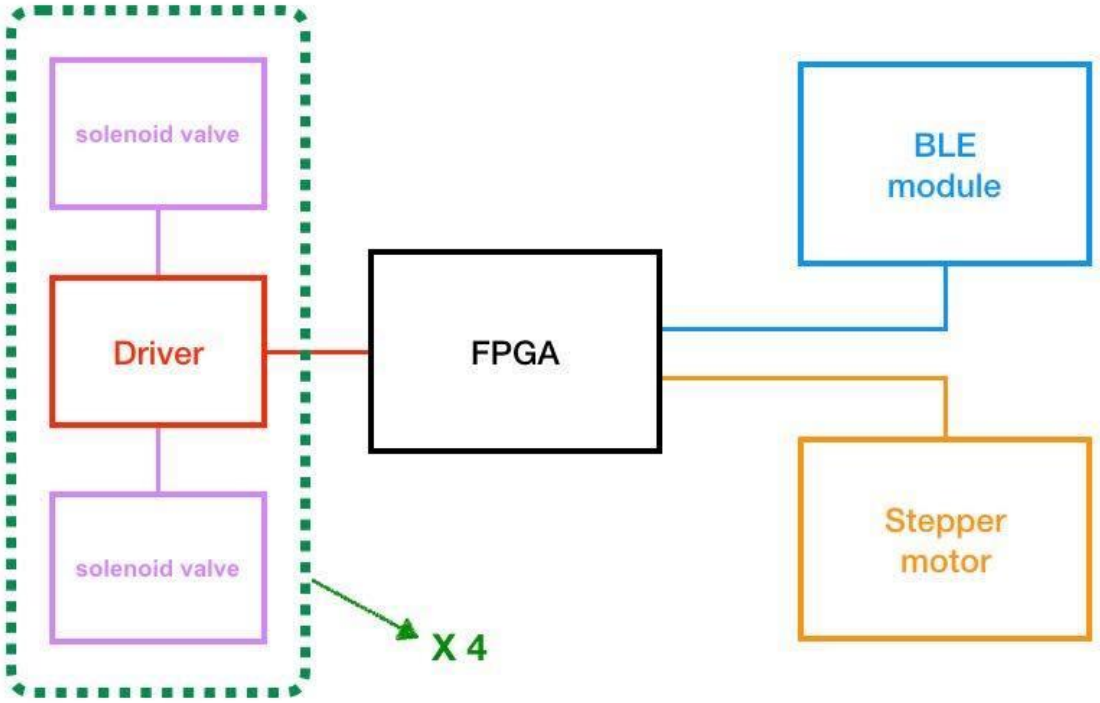
Bartender 為一款自動調酒機，藉由 app 經由藍牙與其通訊，有鑒於人為調酒時，倒酒的低效率與不精確性，我們希望以科學的方式，藉由機器精準提供標準化比例的調酒，讓每一杯調酒有最適當、完美的口感。此外我們藉人性化的手機 app 介面，提供使用者多種調酒選項，用最簡單的方式享受與酒吧相同的滋味。

Motivation

起初我們這組想要嘗試製作咖啡沖泡機，但礙於製作過程非常繁雜且需花費大量金錢，遂而取消此想法，經多次討論，決定製作飲料機。起初只想做出類似奶茶或是奶綠等簡單飲料組合，但既然 Final Project 是展示自己成果的地方，何不呈現一些會吸引大家目光的東西呢？所以最後產生調酒機的想法。我們發現，在國外網站也發現了調酒機是很少見的，僅僅看到 Barsys 這款由印度人 Akshet Tewari 製作的機器，但此款機器只能放上 5 款基酒，我們認為應該超越他，最後決定以 8 款不同的酒與飲料為基本配置，並藉由手機介面來操作我們的機器。

來自印度德里的 Akshet Tewari 與團隊共同開發出「自動調酒機 Barsys」，內建 2000 種以上的酒單資源，馬丁尼 (Martini)、長島冰茶 (Long Island Iced Tea) 和柯孟波單 (Cosmopolitan) 等應有盡有，而且做好一杯調酒只需要約 30 秒。雖然一次最多僅能放上 5 瓶基酒，但已足夠應付個人和派對使用。

Overview



以 FPGA 為中心，連接藍芽模組、步進馬達及馬達驅動板、4 塊 L298N（原是馬達驅動板，我們用以驅動電磁閥），每塊 L298N 連接 2 顆電磁閥。

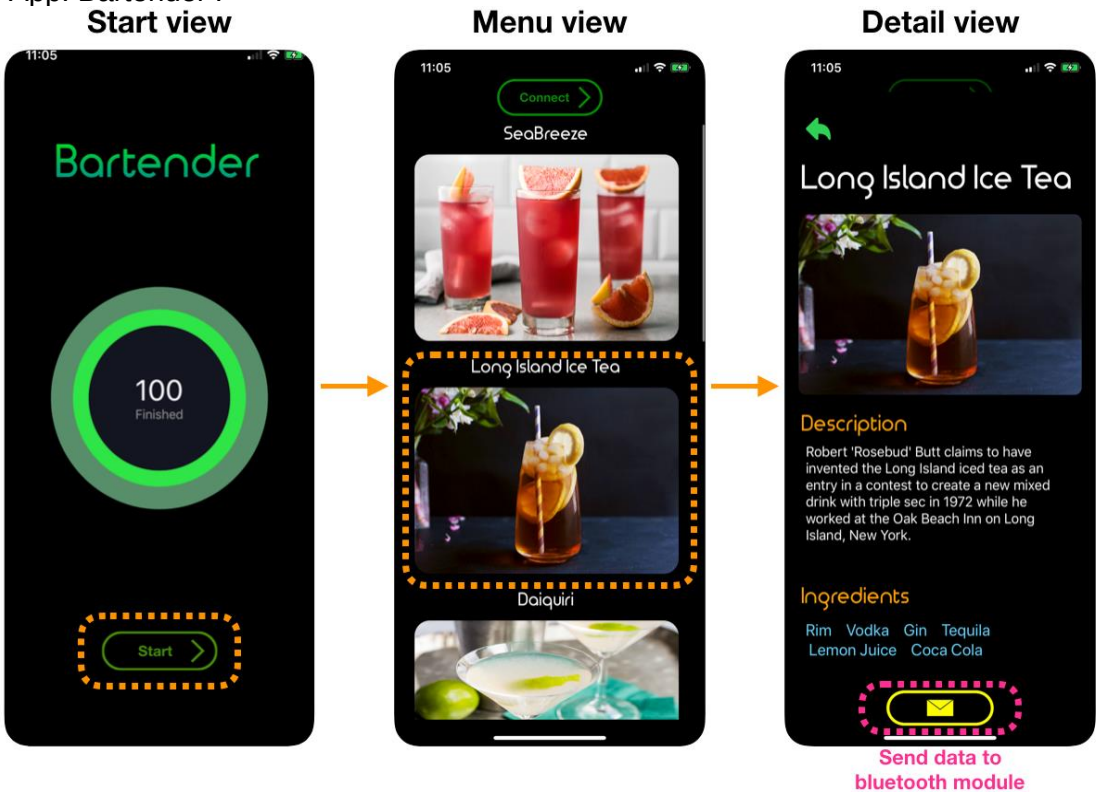
此裝置運作方式為：1. 將杯子放在轉盤上，手機連接藍芽並使用 APP 點選調酒，藍芽傳入 FPGA 訊號後，上面的成分會依照比例加入杯子，每加完一種成分後，連接馬達的轉盤會將杯子送到下個成分出口，轉完一圈即完成調酒。

How App connects with Bartender

這次我們藉由 app 透過藍芽與調酒機做連結，以下為其概念圖:

以下我們將對 app 及 Bluetooth 的連線做詳盡的解釋：

(a) App: Bartender :



為了讓 Demo 過程更吸睛，我們用 app 的方式來呈現，此處撰寫 app 的語言為 swift，以下將介紹此 app 的三個頁面：

(1) Start view:

此為剛進入 app 時的初始畫面，在此畫面時會將存於 database 的資料提出，並做好 pre-load 的動作，完成讀取資料後，下方的進入鈕即會顯示。

(2) Menu view:

此頁面會顯示所有可選取的酒單，諸如：長島冰茶、琴通寧、自由古巴等七種不同的調幾款項，起初我們的設定為三十種，但是後來經過一連串的討論後，心想這麼多調酒可以選是很好，但是在 Demo 的時候卻無法一一展示出來，所以最後決定做七種調酒就好，在 Demo 時就以最複雜的長島冰茶作為 Demo 的酒款。而此處使用者點選任一選項，即可進入 Detail view 的頁面。

(3) Detail view:

此頁面為介紹各款酒的資訊，其中包含其照片、歷史背景及出處、內容物以及傳輸按鈕，使用者可以藉由此頁面獲得關於此款酒的資訊，當決定好要此種酒後，按下傳輸鍵，此時會秀出 Confirmed view，此畫面會詢問使用者確定要選取此款調酒，如確定，則

app 會經由以下程式傳輸資料到 Bluetooth：

```
var Data:[UInt8] = [0x44,0xFF]
Data[1] = UInt8(BartendingNumber)

DispatchQueue.main.asyncAfter(deadline: .now() + 1.0, execute: { self.resetBartendingDetailView()
    if BLE.sharedInstance().activePeripheral?.state == CBPeripheralState.connected {
        print("bleDidConnectToPeripheral - send DATA to device!")
        BLE.sharedInstance().writeWithResponse(data: self.dataWithHexstring(bytes: Data) as Data, UUID: "FFF2")
    }
})
```

Data 為我們要傳輸的資料，而在 swift 中的 BLE delegate 傳輸資料的方式為 16 進位法，而 Data 會傳輸兩個字元，第一個字固定為 D，第二個才是要傳輸的第幾款酒的編號，此種做法可以在傳輸資料時有一個字元可以判斷進入的訊號是資料與否。

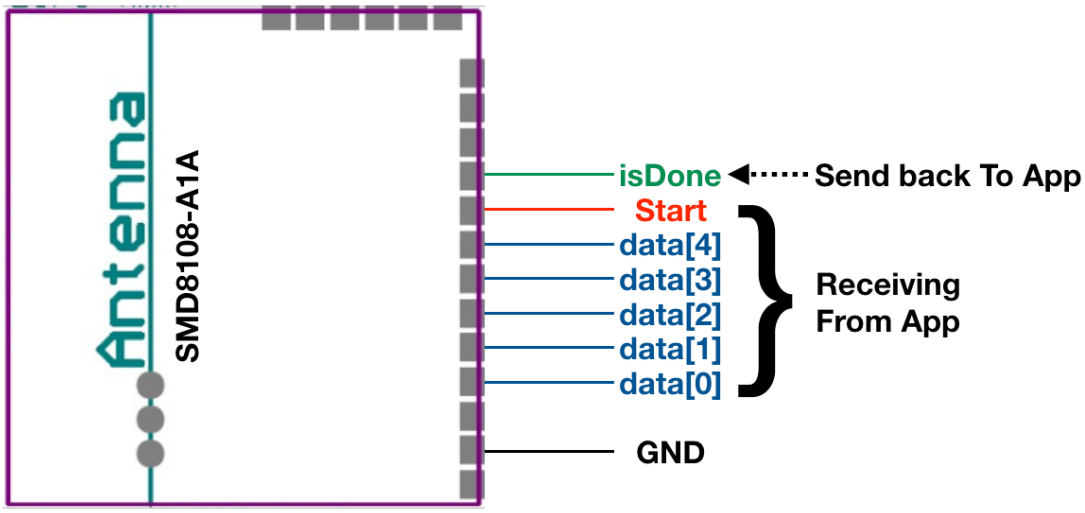
```
if tick_count == 0 {
    let Data: [UInt8] = [0x43, 0x59]

    if BLE.sharedInstance().activePeripheral?.state == CBPeripheralState.connected {
        print("bleDidConnectToPeripheral - send DATA to device!")
        BLE.sharedInstance().writeWithResponse(data: self.dataWithHexstring(bytes: Data) as Data, UUID: "FFF2")
    }

    if isDone {
        MenuTable.isUserInteractionEnabled = true
        isDone = false
        tickerTimer.invalidate()
    }
}
```

此處為 ticker function 中的一小段，由前面的敘述中我們提到我們有個 pin 腳為接收 FPGA 是否已完成調酒的訊號，而此段即是 app 端我們每半秒就發送一次訊號詢問 FPGA 是否已完成製作，如果完成了，則在此段程式前面有個會改變 isDone 的機制，如果調酒製作完成，則 if loop 中的參數就會被 reset 並回到可在製作的模式中。

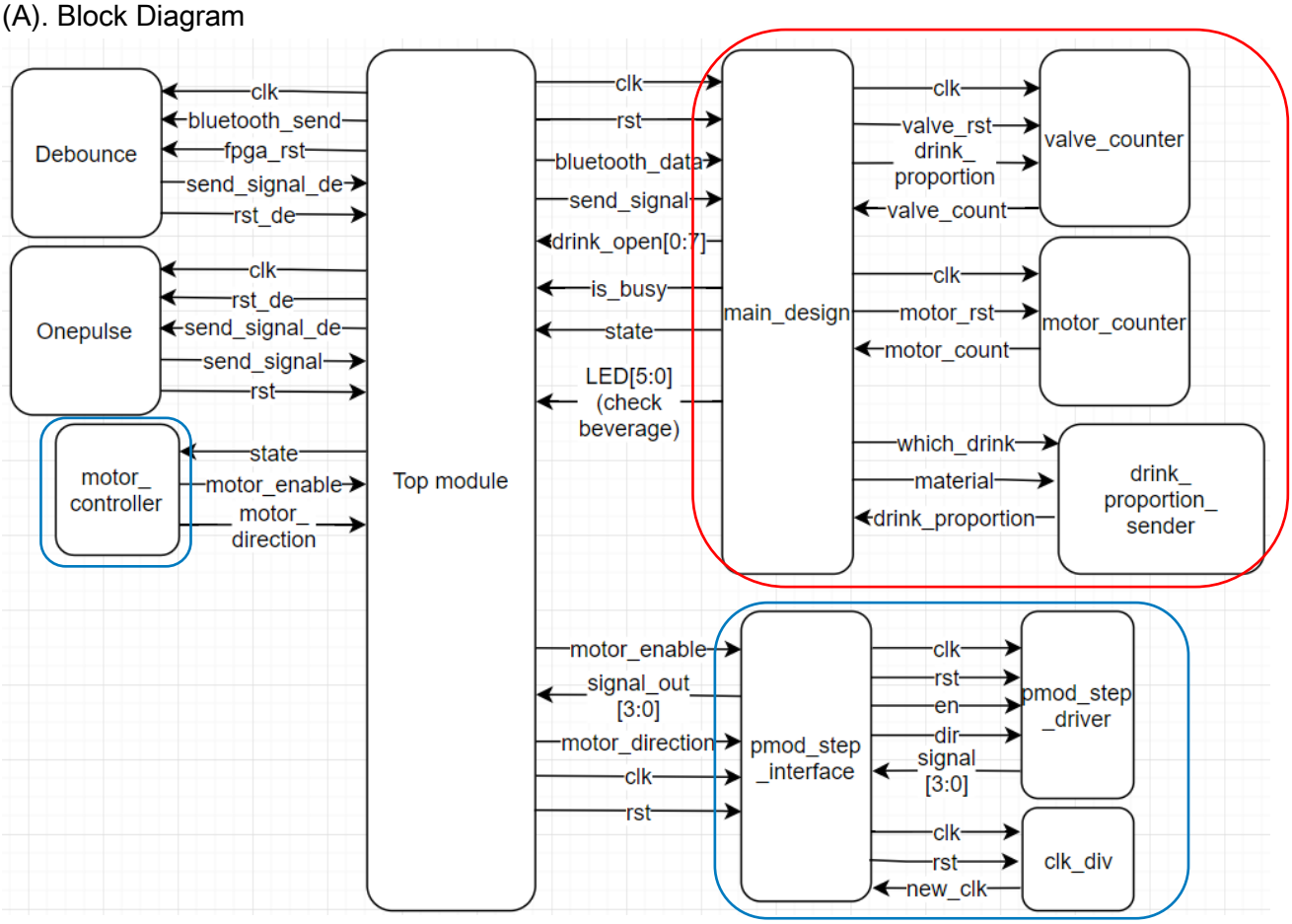
(b) Bluetooth connection:



這次我們用到的藍芽模板為 SMD8108-A1A，其好處為已將藍牙鑲嵌在可接線的板子上，其麻煩之處為要自己用 C 語言去編寫好每個 pin 腳的輸出與輸入，以我們的例子為例，運用到七個 pin 攪作輸出輸入，其中有六個輸出端，分別為五個資料 pin 以及一個 start pin，剩下的一個為 isDone pin，這個訊號會每 0.5 秒對此模板傳輸訊號，當其傳輸的值為 0 時，則代表機器仍然在製作調酒，此時手機端不論做任何傳輸的動作對此模板都沒有任何效果；相反地，只要此值變為 1 時，即代表調酒已製作完成，代表我們可繼續製作其他酒款。

How FPGA, Stepper motor, Valves work

(A). Block Diagram

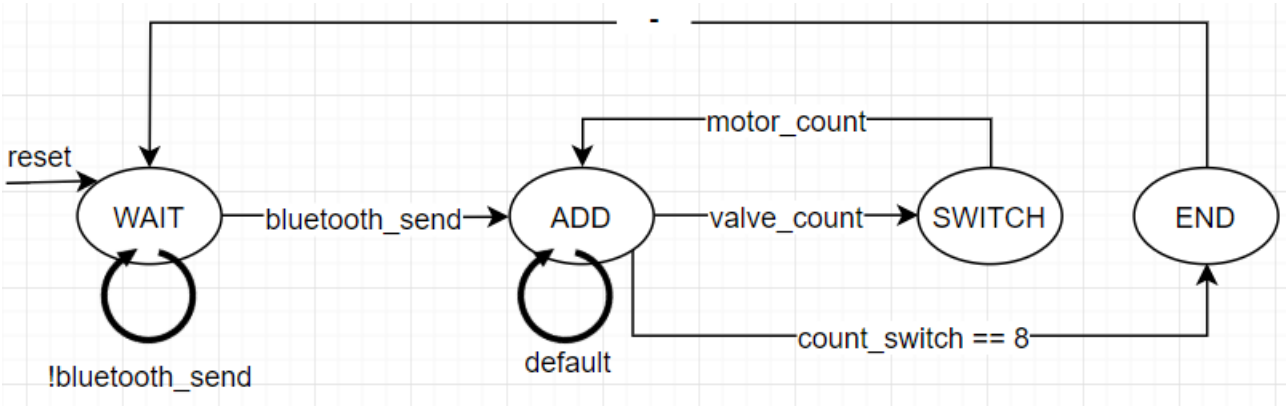


FPGA 需與藍芽、馬達、電磁閥三種硬體相連，接下來會一一解說。左上角是負責處理藍芽傳過來的”start”，也就是 verilog code 裡的”bluetooth_send”、以及”rst”，由 FPGA 的中間按鈕作為輸入，一按下按鈕就回到等待狀態（可參閱 B 部分），方便使用者點錯反悔時使用。

藍色方框是控制步進馬達用的，其中 pmod_step_interface, pmod_step_driver 以及 clk_div 是從網路上抓下來的（資料來源附於 report 最後一頁），運作原理會在待會詳細解說。

紅色方框是主要的設計，負責控制狀態、電磁閥開關、馬達、接受藍芽資料，詳細也會在下面解釋。

(B). State Transition Diagram



這是 Bartender 的運作流程，首先在 WAIT 狀態等待藍芽傳送資料，若 bluetooth_send 拉成 1，進入 ADD 狀態，並根據刻在 code 的比例加第一種酒，此時會送出訊號給驅動板，驅動板會讓一號電磁閥通電，閘門就開啟，加完後（valve_count 算加酒時間）斷電，並進入 SWITCH 狀態，此時馬達會轉 45 度，轉到後（motor_counter 算轉到目的地時間）會回到 ADD 狀態，共重複八次，直到最後一次（count_switch 自 0 開始，每次從 ADD 到 SWITCH 加 1，加到 8 就代表不用再加酒）SWITCH 轉到 ADD 時，代表製作完畢進入 END 狀態，並在 10ns 後自動回到 WAIT 狀態。

(C). Detailed Description

// 以下會順序地詳細解釋各設備的操作

(1). FPGA 如何接收藍芽信號

```
WAIT: begin
    if(send_signal) begin
        next_state = ADD;
        next_count_switch = 4'd0;
        next_which_drink = input_drink;
    end else begin
        next_state = WAIT;
        next_count_switch = 4'd0;
        next_which_drink = 5'd31;
    end
end
```

當藍芽將”bluetooth_send” (藍芽 code 裡叫”start”)拉成 1，會立刻抓取 input_drink(也就是藍芽傳入的 bluetooth_data)，並且持續維持飲料種類的值。(沒飲料的 default 值是 31)

(2). 怎麼控制電磁閥

```
ADD: begin
    next_which_drink = which_drink;
    if(count_switch == 8) begin
        next_state = END;
    end else begin
        if(valve_count) begin
            next_state = SWITCH;
        end else begin
            next_state = ADD;
        end
    end
    next_count_switch = (state == ADD && next_state == SWITCH)? count_switch + 1: count_switch;
end
```

```
assign count = (i == n * proportion * 10000000)? 1: 0;
```

(proportion 為當前材料所需的百分比)

```
valve_counter #(10) valve_counter_0
```

(n = 10，也就是開 1 秒電磁閥)

```
assign drink_open[0] = (state == ADD && count_switch == 0)? 1: 0;
```

(drink_open[ ] 是控制電磁閥的信號)

電磁閥控制解釋：

在 ADD 狀態時，控制電磁閥開關的 drink_open 會依照 count_switch 的數字而開啟特定電磁閥，直到 valve_count 拉成 1。valve_counter 一開始進入 ADD 狀態時會被 reset，之後不斷地數，數到一定的數(見圖二條件判斷)後，valve_count 即拉成 1，轉入 SWITCH 狀態，關閉電磁閥，並開始轉動馬達。

(3). 怎麼控制步進馬達

```
clock_div clock_Div(
    .clk(clk),
    .rst(rst),
    .new_clk(new_clk_net)
    );
assign motor_enable = (state == SWITCH)? 1: 0;
assign motor_direction = 1;
```

```
module pmod_step_interface(
    input clk,
    input rst,
    input direction,
    input en,
    output [3:0] signal_out
    );
```

```
SWITCH: begin
    next_which_drink = which_drink;
    next_count_switch = count_switch;
    if(motor_count) begin
        next_state = ADD;
    end else begin
        next_state = SWITCH;
    end
end
```

馬達控制解釋：

只要傳入想要的”direction”, ”enable”, “new_clk_net”，就可以控制馬達”順時針或逆時針轉”、”動或不動”、”轉多快”，最後 counter 控制 SWITCH 狀態的時間，就可以正確地調到我想要的速度以及位置。

馬達信號補充：

```
begin
    if (present_state == sig4)
        signal = 4'b1000;
    else if (present_state == sig3)
        signal = 4'b0100;
    else if (present_state == sig2)
        signal = 4'b0010;
    else if (present_state == sig1)
        signal = 4'b0001;
    else
        signal = 4'b0000;
```

步進馬達利用四個電磁感應推動，需要四個信號輪流拉成 1 藉以使馬達轉動。就像上面 code 寫的那樣。

```
// The constant that defines the clock speed.
// Since the system clock is 100MHZ,
// define_speed = 100MHz/(2*desired_clock_frequency)
localparam define_speed = 26'd500000;
```

這邊是 clk_div 裡的常數，此常數與馬達轉速成反比，此速度下，步進馬達每 2.6 秒轉 45 度。

(4). FPGA 工作時如何阻隔藍芽信號

```
assign is_busy = (state == END || state == WAIT)? 0: 1;
```

飲料機還在加酒時，is_busy 拉成 1 給藍芽，藍芽會阻隔 APP 來的信號。

Experiment Result

經裁切木塊、鑽洞、接線、加對電壓、安排複雜的線路、FPGA 與藍芽對接傳輸資料、嘗試用 FPGA 驅動電磁閥等等困難後，我們成功如期做出一台調酒機，外型幾乎符合我們當初的規劃（馬達要放中間所以柱子改接旁邊），能用藍芽傳輸資料給 FPGA，能任意地調整酒的比例（需調整 verilog code，未來希望可以在 APP 上實現），這台機器完成度很高，算是達成當初的目標了。

Future Goal

起初我們對於本次的規劃其實是希望可以做到完全客製化的調酒機，簡單來說，使用者可以藉由手機面板調控各自喜歡的口味及比例，然而我們之所以沒有做到此步，是因為在傳輸資料上的困難度不易克服，以下兩種方式是我們相信在未來我們可行的方法：

(1) 對於八種不同的基酒及飲料，我們可以規劃 8 個 pin 腳，每次 FPGA 就依序由 1 號酒開始接收 6 bits 的 sequence 資料(i.e. 010010 代表 1 號酒佔總量 18%)一直接收到 8 號酒才開始製作。

(2) 一號方法的缺點為會佔掉大量的 pin 腳，為了解決此一問題，我們可以將上述的編碼做成一長串的 0 or 1 sequence，接下來藉由 FPGA 開始接收資料；然而此種方法的缺點在於大量的資料很可能在藍牙的傳輸中有所遺漏，所以得在 App 端及 FPGA 端做好良好的溝通以避免資料的遺漏。

以上為軟體方面的加強，而硬體方面，希望可以加速調酒的製作時間，以及各種基酒及添加物的精準性，由於考量到步進馬達失步的問題，無法做到高速旋轉到指定地點，再者，電磁閥的流量過小，進而花費了許多時間將添加物輸出，以上提到的不足，希望在以後有機會可以有所改進超過市面上的調酒機。

Conclusion

對於這次進行 final project 的過程，我們收穫良多；第一，過往我們的 project 多以個人操作為主，而此次的進行方式卻大為不同，需要兩個人經過多次的溝通、討論及傾聽才得以完成，很高興在討論的過程中，沒有產生情緒的衝突，而是以理性的方式提出各自的觀點，最終產生出最後的計畫，我想這就是教授希望我們能藉由本次的 project 學習到的經驗之一。

第二，學習將本學期課程中習得的知識加以應用在我們的機器上，不論是 clock, state transition diagram, button implementation 等，在我們的專題中都扮演著許多重要的角色，很多時候我們僅僅是將課堂上習得的知識用於對付期中期末考等等偏向考試方面的地方，然而專題的目的就是希望我們藉由手作以及自行上網查詢等，將課程上學習到的知識加以應用，畢竟在未來我們的工作方面就多以應用方面居多，藉由親自規劃及嘗試讓我們了解到將知識應用到現實生活的重要性。

第三，由於我們這次的機器用到大量的木工及接線等步驟，在起初動手時並沒有做到詳盡的規劃，進而導致期間浪費了許多的材料及時間，效率明顯不佳，不過經過一次又一次的改進，我們學習到了許多經驗；未來不論是在其他專題又或是產品的發展上，事前的規劃扮演著極其重要的角色，如果能在事前就做到良好的規劃，就可以避免許多金錢及時間的浪費。

最後我們想提到的是，有時候對於一個產品的產生的初期會有許多的疑慮，但是如果我們在未動手實踐前就給予自己悲觀的想法，那這個產品終將不會實現，美國著名的人際關係學大師 戴爾·卡耐基就曾說過： 多數人都擁有自己不了解的能力和機會，都有可能做到未曾夢想的事情。 很高興我們從來都沒有懷疑過自己是否能完成這項專題，但是我們一步一步去克服遇到的困難，沮喪是成功道路上的寒風，唯有堅定信念挺過一次又一次的嚴寒，我們才能迎接代表成功的暖陽。

資料來源：

步進馬達 code：https://www.instructables.com/id/How-to-Control-a-Stepper-Motor-With-an-FPGA/