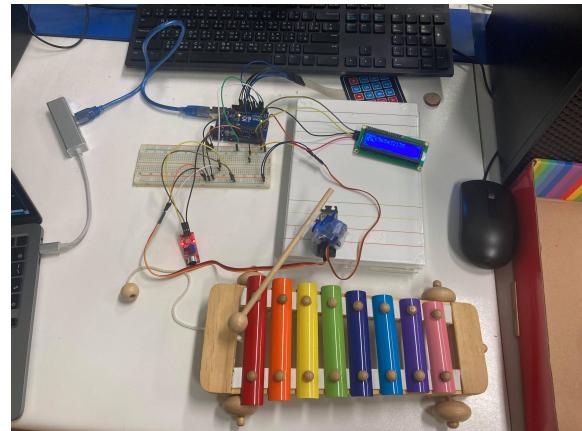
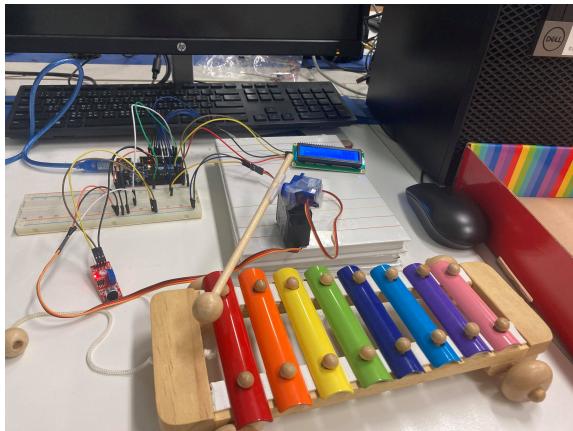




# Arduino Xylophone Report

2020 Embedded System Final Project

107062130 陳奕君



[Goal](#)

[Components](#)

[LCD](#)

[Keypad](#)

[麦克風模組](#)

[伺服馬達](#)

[Implementation](#)

[Global Variables](#)

[Setup](#)

[Servo Function](#)

[ControlCenter](#)

[SongTask](#)

[PlayTask](#)

[RecordTask](#)

[ReplayTask](#)

[FFTTask](#)

[FFT Replay Task](#)

[Difficulties](#)

[Program Flow Chart](#)

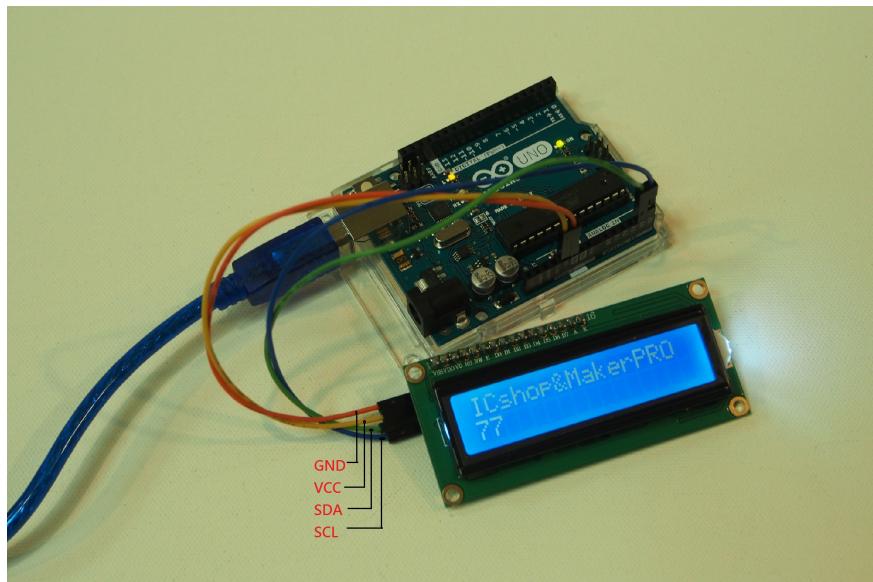
## Goal

用 Arduino 實現一台可以自動敲歌、跟人互動的鐵琴，有以下 5 種模式：

- 播歌模式：可以自己選擇想聽的歌，arduino 會自動敲打樂曲，LCD 顯示現在是什麼歌和敲什麼音
- 演奏模式：可以用按鍵選擇要彈什麼音，LCD 會顯示音名（簡譜記號）
- 錄音模式 1：功能跟玩樂模式一樣，差別在會記住你彈過什麼音
- 錄音模式 2：人敲打鐵琴，arduino 聽音辨認現在在敲什麼音，並記住
- 回放模式 1：可以彈奏出剛剛錄起來的音樂

## Components

### LCD



LCD 及 pin 腳位示意圖。圖片來源：<https://makerpro.cc/2017/02/how-arduino-use-i2c-to-control-lcd-module/>

透過  
LiquidCrystal\_I2C  
函式庫來控制 LCD  
螢幕上要顯示什麼。  
由於課堂上有教過因  
此這裏不贅述  
function 功能，僅  
列出我所使用的  
function:

```
lcd.clear() ,  
lcd.print() ,  
lcd.setCursor() ,  
lcd.backlight()
```

### Keypad



ArduinoGetStarted.com

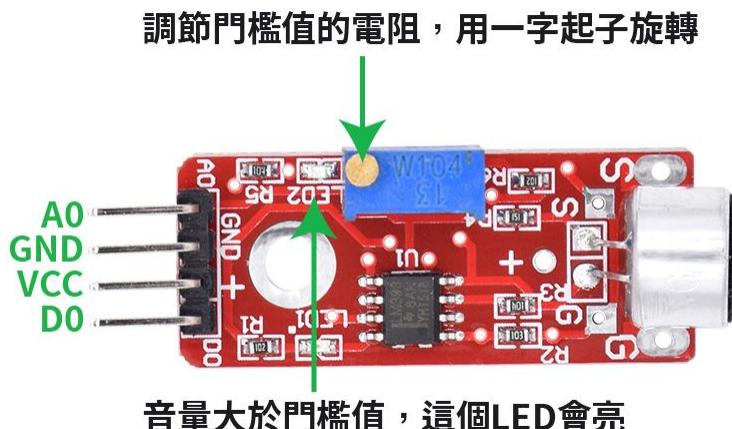


Keypad 及 pin 腳位示意圖，我使用的是右邊的 4\*4 版本。圖片來源：

<https://arduinogetstarted.com/tutorials/arduino-keypad>

透過 Keypad.h 函式庫來建立 keypad instance - myKeypad, 要先設定好 colPins, rowPins 和 keymap 來確定使用的 pin 腳和吃進來的 key 對應到哪個字元，並使用 `myKeypad.getKey()` 來得知現在使用者按了哪個鍵

## 麥克風模組

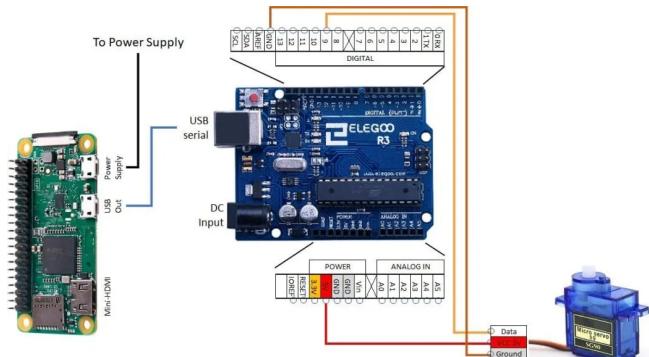


此麥克風模組可接收環境的聲音，並透過 A0 腳位輸入給 arduino 知道，除了聲音大小外也能使用 arduinoFFT.h 的一些功能來得知現在環境最大聲音的頻率。

麥克風模組及 pin 腳位示意圖。圖片來源：

<https://blog.jmaker.com.tw/arduino-sound/>

## 伺服馬達



SG90 servo 及腳位示意圖。圖片來源：

<https://peppe80.com/how-to-remote-control-a-servo-motor-sg90-using-raspberry-pi-zero-w-with-python-and-arduino/>



MG995 servo 及腳位示意圖。圖片來源：

<https://components101.com/motors/mg995-servo-motor-datasheet>

利用 Servo.h 函式庫的幫助，就能很輕鬆的指定哪個馬達要轉幾度

使用 `Servo servo_name` 宣告一個伺服馬達 instance

`servo_name.attach(pin)` 可以指定這個馬達的 PWM 在哪個 pin

`servo_name.write(angle)` 可以指定此馬達要轉幾度

## Implementation

### Arduino 與 Peripheral 腳位對照表

Aa	Arduino pin	Component	Component - pin	備註
13		Keypad	R1	
12		Keypad	R2	
11		Keypad	R3	
10		Keypad	R4	
9		Keypad	C1	
8		Keypad	C2	
7		Keypad	C3	
6		Keypad	C4	
5				
4		Servo SG90	PWM	
3		Servo MG995	PWM	

Arduino pin	Component	Component - pin	備註
2	Button		
A0	Microphone	A0	
A4	LCD	SDA	
A5	LCD	SCL	

接下來會分成一個一個功能、模式講解整份 code。由於記憶體問題，我寫了兩份 Code，這個問題會在 difficulty 內詳細敘述，以下將會合併在一起講，並附註這是寫在 Code1 還是 Code2。

## Global Variables

```
// song
const int8_t star[48] = {1, 1, 5, 5, 6, 6, 5
const int8_t bee[64] = {5, 3, 3, 0, 4, 2, 2,
int8_t record[16];
int8_t record_id;

char key;
int8_t angle = 10;
int8_t pos_x;
int8_t mode = 1;
uint8_t i = 0;
```

(Part of) Code 1 global variables

最上面兩個陣列是用來紀錄歌曲的，1 - 7 是音，0 就是不演奏

`record` 是錄音時儲存音的陣列，`record_id` 紀錄現在的 record 大小

`key` 用來儲存 keypad 吃進來的訊號

`angle` 是轉 SG90 角度的迴圈裡用的，為了不要影響其他迴圈

`pos_x` 是用來記錄 lcd 顯示的位置

`mode` 是紀錄現在是什麼模式

`i` 是拿來迴圈用的，為了節省記憶體因此拿到 global 用

```
#define SAMPLES 128           // Must be a base 2 number. Max 128 for Arduino Uno.
#define SAMPLING_FREQUENCY 2048 // Must be 2 times the highest expected frequency.
```

```
arduinoFFT FFT = arduinoFFT();

unsigned int samplingPeriod;
unsigned long microSeconds;

double vReal[SAMPLES]; //create vector of size SAMPLES to hold real values
double vImag[SAMPLES]; //create vector of size SAMPLES to hold imaginary values
```

(Part of) Code2 global variables

如果要使用 arduinoFFT 的 function 的話，需要先宣告一個 instance (`FFT`)

`samplingPeriod` 用來紀錄每次隔多久要 sample 一次

`microSeconds` 則是用來記住一開始 sample 時的時間，這樣才能配合 `micros()` 來算間隔了多久

`vReal` 是存我們在每次 sample 時從 A0 接收到的最大聲音的頻率

`vImag` 則都是 0，

## Setup

```
void setup() {
    Serial.begin(9600);

    servo_SG90.attach(4);
    servo_MG995.attach(3);
    servo_MG995.write(0);

    lcd.init();
    lcd.backlight();
    pos_x = 0;
    record_id = 0;
    testTask();
}
```

Code1 setup

成品有兩個伺服馬達，一個是 MG995，負責控制敲級棒的旋轉角度，另一個是 SG90，負責控制敲擊棒的上下敲打。會先指定 MG995 轉到 0 度位置是因為我以 0 度為最低音位置基準點，加上如果不先給定角度，在剛啟動時會因為電壓不穩造成馬達左右胡亂轉而把棒子甩出去，如果有在 setup 時設好角度就不會發生這個情況。

再來重新把 lcd 初始化、亮背光、設定初始位置為 0。

給 `record_id` 初始值，一開始沒有錄音所以是 0。

最一開始的模式不在正式模式裡，算是測試用，會一直敲打 Do 基準音，可以在這時看角度有沒有對，調整鐵琴位置。

```
void setup() {
    Serial.begin(9600); //Baud rate for the Serial Monitor

    pinMode(buttonPin, INPUT);

    servo_SG90.attach(4);
    servo_MG995.attach(3);
    servo_MG995.write(0);

    record_id = 0;
    samplingPeriod = round(1000000 * (1.0 / SAMPLING_FREQUENCY)); //Period in microseconds
    for (int i = 0; i < 16; i++) {
        play(1, 100);
    }
}
```

Code2 setup

這裡只比 Code1 多了設定 `samplingPeriod` 的步驟，是用來決定每隔多久要 sample 一次，前面設定 sample 頻率是 2048，因此週期就是  $1 / 2048$ ，最後再乘  $10^6$  讓單位從秒變微秒，取 round 便整數。

下面是讓它敲基準音 Do 16 次，方便我們調整鐵琴角度，敲出最好聽的聲音。

## Servo Function

`change(int8_t note)`

```
// MG995
void change(int8_t note) {
    if (note == 1) {
        servo_MG995.write(0);
    } else if (note == 2) {
        servo_MG995.write(22);
    } else if (note == 3) {
        servo_MG995.write(40);
    } else if (note == 4) {
        servo_MG995.write(54);
    } else if (note == 5) {
        servo_MG995.write(64);
    } else if (note == 6) {
        servo_MG995.write(80);
    } else if (note == 7) {
        servo_MG995.write(98);
    } else if (note == 8) {
        servo_MG995.write(111);
    }
}
```

用來控制 Servo MG995 的轉動角度，只要 call `change(音)` 就能讓 MG995 自動轉到對應的音的角度。

數字 1 - 8 是音符 Do, Re, Mi, Fa, Sol, La, Si, Do 的簡譜記號，我以角度 0 為 Do 的基準點，再分別去量其他音的角度求得的。

`knock()`

```
// SG90
void knock() {
    servo_SG90.write(82);
    for(angle = 82; angle >= 71; angle--) {
        servo_SG90.write(angle);
        delay(10);
    }
    for(angle = 71; angle <= 82; angle++) {
        servo_SG90.write(angle);
        delay(10);
    }
}
```

`knock` 是用來控制 Servo SG90 的 function，決定棒子上下敲打的角度與速度。

每次呼叫 `knock()`，就等於是讓棒子往下敲打一下並回原位，會搭配上面的 `change()` 讓 arduino 做到自動敲擊鐵琴，而且包成這兩個 function 也能在寫 code 時更直覺，不用去想角度問題。

`play(int8_t note, int d_time)`

`note` 指定現在要敲什麼音，如果是 0 就是不敲。

```

// play xylophone
void play(int8_t note, int d_time) {
    if (note == 0) {
        delay(200 + d_time);
        return;
    }
    change(note);
    delay(200);
    knock();
    lcd.setCursor(pos_x, 0);
    if (pos_x == 15) {
        pos_x = 0;
        lcd.clear();
    } else {
        pos_x += 1;
    }
    lcd.print(note);
    delay(d_time);
}

```

`d_time` 指定敲完這個音要等多久才會有下個指令，用來調整現在是一拍、半拍、兩拍或四拍

會在轉動 MG995 跟 SG90 間延遲 2 ms 是因為如果不 `delay` 的話在轉動棒子的同時棒子也會下降，造成轉動過程中經過的音都會被敲到，因此有延遲 2 ms。

除了敲音以外還會在 `lcd` 上顯示目前彈的是什麼音，如果沒敲滿 15 音的話，音符顯示會一路由左往右，敲滿 15 音的話會清空音符列，再從最左邊開始 `print`。

使用 `play` 可以更方便的指定要敲什麼音、幾拍，就跟在演奏樂譜是一樣的。

## ControlCenter

此 function 是在吃 `keypad` 讀進來的 `key` 並且根據 `key value` 來轉換模式。

前面的 `switch(key)` 就是在看吃進來的是哪個 `key`，並轉換模式  
+ 用 `lcd` 印出現在處於哪個模式下。

`key '0'` 是測試模式，對應  
`mode = 0`

`key 'A'` 是歌曲模式，對應  
`mode = 1`

`key 'B'` 是歌曲模式，對應  
`mode = 2`

`key 'C'` 是歌曲模式，對應  
`mode = 3`

`key 'D'` 是歌曲模式，對應  
`mode = 4`

後面的 `switch(mode)` 是根據現在是什麼 `mode` 進行什麼 task，雖然可以一起寫在

`switch(key)` 裡，但這樣寫邏輯比較清楚，也可以任意加上想操作的 key 跟模式。後面會再詳細介紹各個 task。

```
void controlCenter() {
    key = myKeypad.getKey();

    // test
    switch(key) {
        case '0':
            mode = 0;
            lcd.clear();
            lcd.print("TEST");
            delay(1000);
            lcd.clear();
            break;
        case 'A':
            mode = 1;
            pos_x = 0;
            lcd.clear();
            lcd.print("SONG");
            delay(1000);
            lcd.clear();
            break;
        case 'B':
            mode = 2;
            pos_x = 0;
            lcd.clear();
            lcd.print("PLAY");
            delay(1000);
            lcd.clear();
            break;
        case 'C':
            mode = 3;
            record_id = 0;
            pos_x = 0;
            lcd.clear();
            lcd.print("RECORD");
            delay(1000);
            lcd.clear();
            break;
        case 'D':
            mode = 4;
            pos_x = 0;
            lcd.clear();
            lcd.print("REPLAY");
            delay(1000);
            lcd.clear();
            break;
        default: break;
    }

    switch(mode) {
        case 0: testTask(); break;
        case 1: songTask(); break;
        case 2: playTask(); break;
        case 3: recordTask(); break;
        case 4: replayTask(); break;
        default: break;
    }
}
```

## SongTask

```
void songTask() {
    if (key == '1') {
        pos_x = 0;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("STAR");
        delay(1000);
        lcd.clear();
        for (i = 0; i < 48; i++) {
            play(star[i], 100);
        }
    } else if (key == '2') {
        pos_x = 0;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("BEE");
        delay(1000);
        lcd.clear();
        for (i = 0; i < 64; i++) {
            play(bee[i], 100);
        }
    } else if (key == '3') {
        pos_x = 0;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("LITTLE WORLD");
        delay(1000);
        lcd.clear();
        for (i = 0; i < 80; i++) {
            play(story[i], 100);
        }
    }
    controlCenter();
}
```

播歌模式，對應 `mode = 1`。

按 `'A'` 進入這個模式後可以透過 keypad 選擇三首歌，每首歌我會先讓他們顯示名字 1 秒，後來再去演奏。由於譜已有事先寫好，因此只要簡單用個迴圈 call `play()` 就好，這裡一拍是 0.1 秒，但由於轉動馬達本身也會有延遲，一拍實際上會有 0.5 秒，就樂理來說就是一分鐘有 120 拍。

每次偵測完 `key` 後就要呼叫一次 `controlCenter()`，確認有沒有被換模式，為了保持樂曲完整性，在演奏中途無法突然換模式。

## PlayTask

演奏模式，對應 `mode = 2`。

這裡也是根據 keypad 回傳什麼就彈什麼音，拍子是  $0.2 + 0.4$

= 0.6 秒一拍，但由於實際上在吃 key 時會有延遲，因此會拍子時常會更久一些。

```
void playTask() {
    switch(key) {
        case '1': play(1, 200); break;
        case '2': play(2, 200); break;
        case '3': play(3, 200); break;
        case '4': play(4, 200); break;
        case '5': play(5, 200); break;
        case '6': play(6, 200); break;
        case '7': play(7, 200); break;
        case '8': play(8, 200); break;
        default: break;
    }
    controlCenter();
}
```

## RecordTask

錄音模式，對應 mode = 3 。

DEBUG 是用來印出資訊的 function，前面的參數是表示 type 的， i 是整數， s 是字串，而且後面的參數是可無限增加的，可以用一行 code 代表很多的 Serial.println() 。

後面則是跟演奏模式很像，差別只在要限制錄音的長度（最多只有 16 格）跟把演奏的音寫進陣列裡。錄音也是採用 function 的形式寫，可以增加 code readability 和 maintainability。

```
void recording(int8_t note) {
    record[record_id] = note;
    record_id += 1;
}

void recordTask() {
    DEBUG("i", record_id);
    if (record_id < 16) {
        switch(key) {
            case '1':
                play(1, 200);
                recording(1);
                break;
            case '2':
                play(2, 200);
                recording(2);
                break;
            case '3':
                play(3, 200);
                recording(3);
                break;
            case '4':
                play(4, 200);
                recording(4);
                break;
            case '5':
                play(5, 200);
                recording(5);
                break;
            case '6':
                play(6, 200);
                recording(6);
                break;
            case '7':
                play(7, 200);
                recording(7);
                break;
            case '8':
                play(8, 200);
                recording(8);
                break;
            default: break;
        }
    }
    controlCenter();
}
```

## ReplayTask

```
void replayTask() {
    if (record_id == 0) {
        lcd.clear();
        lcd.print("NO RECORD");
        delay(1000);
    } else if (key == '#') {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("start music");
        delay(1000);
        lcd.clear();
        for (i = 0; i < record_id; i++) {
            DEBUG("ii", i, record[i]);
            play(record[i], 200);
            controlCenter();
        }
        delay(1000);
    }
    controlCenter();
}
```

重播模式，對應 `mode = 4`。

會先檢查 `record_id` 確定有先錄音過，如果沒有錄音 lcd 會顯示 "NO RECORD"，有的話則是按下 `#` 鍵就能重播一次。

原本是希望能在重播時切換模式，但因為迴圈的 `i` 都是共用的，如果在播歌曲的一半跳到 `ControlCenter` 的話 `i` 的值就會被改掉，因此改成寫在條件判斷的外面。

## FFTTask

```

if (buttonState == HIGH) {
    /*Sample SAMPLES times/
    for(int i = 0; i < SAMPLES; i++) {
        microSeconds = micros();

        // Reads the value from analog pin 0 (A0), quantize it and save it as a real term.
        vReal[i] = analogRead(0);
        //Makes imaginary term 0 always
        vImag[i] = 0;

        /*remaining wait time between samples if necessary*/
        while(micros() < (microSeconds + samplingPeriod)) {};
    }

    /*Perform FFT on samples*/
    FFT.Windowing(vReal, SAMPLES, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
    FFT.Compute(vReal, vImag, SAMPLES, FFT_FORWARD);
    FFT.ComplexToMagnitude(vReal, vImag, SAMPLES);

    /*Find peak frequency and print peak*/
    double peak = FFT.MajorPeak(vReal, SAMPLES, SAMPLING_FREQUENCY);
    //      Serial.println(peak);      //Print out the most dominant frequency.

    /*Script stops here. Hardware reset required.*/
    if (record_id < 16) {
        if ((peak >= 46) && (peak <= 51)) {
            DEBUG("sii", "record ", record_id, 1);
            record[record_id] = 1;
            record_id += 1;
            delay(100);
        } else if ((peak >= 174) && (peak <= 178)) {
            DEBUG("sii", "record ", record_id, 2);
            record[record_id] = 2;
            record_id += 1;
            delay(100);
        } else if ((peak >= 468) && (peak <= 472)) {
            DEBUG("sii", "record ", record_id, 3);
            record[record_id] = 3;
            record_id += 1;
            delay(100);
        } else if ((peak >= 678) && (peak <= 682)) {
            DEBUG("sii", "record ", record_id, 4);
            record[record_id] = 4;
            record_id += 1;
            delay(100);
        } else if ((peak >= 993) && (peak <= 997)) {
            DEBUG("sii", "record ", record_id, 5);
            record[record_id] = 5;
            record_id += 1;
            delay(100);
        }
    }
    delay(200);
}

```

FFT Task 是透過麥克風模組採取環境中的聲音和 arduinoFFT 來算出環境中最大聲的聲音的頻率。

Fast Fourier Transform (FFT) 的概念為任何波形都能靠 sin 波形組成，把越多 sin 波疊加就能越接近我們想模仿的波形，因此 sample 的次數才設為 Arduino 可接受的最大值，128，這樣測出來的頻率才是最準的。

再來根據 Nyquist-Shannon Sampling Theorem，如果我們想採樣一個波形，那 sampling frequency 必須至少是此波形頻率的兩倍，這裡 Arduino 可接受最大值為 2048，也就是我們最多只能採樣出頻率為 1024 以下的波形，就鐵琴來說是超過 Sol 的音（不包含 Sol）都無法被辨識。

首先我們先 sample 環境聲音 128 次，`vReal` 是用來存從 A0 讀進的最大聲音的頻率，`vImag` 存的是基準值，這裡我們設基準值為 0。

接下來的 `FFT.Windowing` 是用來計算 weighting factor 的，我們選擇的是 `FFT_WIN_TYP_HAMMING` 的計算方法，公式為  $weighingFactor = 0.54 - (0.46 * \cos(2\pi * ratio))$ ；會用一個 i 從 0 跑到 SAMPLES / 2 的迴圈，分別對每個 `vReal[i]` 乘上 weightingFactor（因為我們是選擇 FFT\_FORWARD，如果不是的話會是除上 weightingFactor）

接下來的 `FFT.Compute` 是計算頻率的主函式，會利用複數計算的方式來更改 `vReal` 和 `vImag` 的值，再透過 `FFT.ComplexToMagnitude` 把複數值變回一般數字，詳細可見原始碼 <https://github.com/kosme/arduinoFFT/blob/master/src/arduinoFFT.cpp>

最後透過 `FFT.MajorPeak` 來得到最大的頻率。

之後就是先透過 `peak` 值看來敲各個音時是什麼頻率，由於環境音有很多不確定因素，因此是娶一個範圍當作某個音，這裏特意 print 出偵測到什麼音是為了方便查看到底有沒有錄到和是不是錄到環境雜音，隔 0.2 - 0.3 秒後再做一次 FFT 偵測。

## FFT Replay Task

```
else {
    for (int8_t j = 0; j < 5; j++) {
        for (int8_t i = 0; i < record_id; i++) {
            DEBUG("sii", "replay ", i, record[i]);
            play(record[i], 200);
        }
        delay(3000);
    }
    record_id = 0;
}
```

如果 pin2 輸入為低電壓的話，就會來到播歌模式，如果沒有錄音的話什麼都不會播，如果有錄音的話則會重複把歌曲播 5 次，每次間隔 3 秒，播完即把歌曲刪除。

# Difficulties

記憶體不足

```
Done uploading.  
Sketch uses 8040 bytes (24%) of program storage space. Maximum is 32256 bytes.  
Global variables use 1303 bytes (63%) of dynamic memory, leaving 745 bytes for local variables. M
```

此為單純只使用 FFTTask 跟 FFT Replay Task 的記憶體用量，如果加上前面 4 個模式的話會出現警告，因為 Global variables 的用量已經達到 95%，跑 FFT 跑到一半就會自動自己 reset，我嘗試了以下幾種方式來減省記憶體：

1. 各個 global variable 都盡量縮小記憶體 ex: `int` → `int8_t`
2. 把 array 變成 global variable，不用變動的就加 const
3. 把迴圈變數變成 global variable
4. 善用 DEFINE

最後把記憶體省下了 50 多 bytes，卻還是有 warning，而且跑 FFT 變得很不穩定，因此最後才會變成拆成兩份 code 的模式。

# Program Flow Chart

