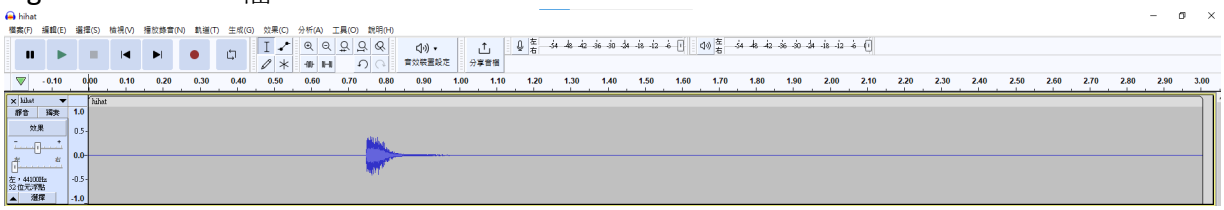
Final Report

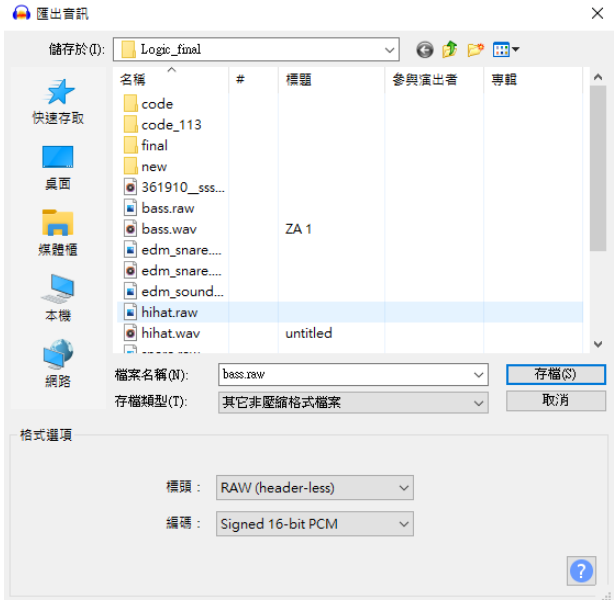
Team No: 04	Team Name: Group04
Project Title: while i == music	
Name: 邱煒甯	ID: 108072244
Name: 劉祥暉	ID: 109072142

A. Lab Implementation

取樣步驟解析:

1. 首先先到 FreeSound 去找到想要的素材，素材有幾個挑選重點:
   (1) 盡量找每個波取樣點的數量比較平均的
   (2) 取樣的素材的長度不能太長，因為可能會造成 memory 不足。
   (3) 音高低(頻率低)的素材會有比較高的機率取樣後的音質變差。
2. 把取樣的素材丟到 Audacity(或是其他音樂分析軟體分析)，分析這邊要處理音量(波幅大小)，因為這部份很難利用 code 去調整。再來是要去看每一個波的取樣點平不平均，若是很不平均(有的波很多取樣點，有的波很少)，可以有兩種處理方法，一是選擇不處理，因為素材經過壓縮，聲音就會變質。二是選擇去壓縮比較多取樣點的波，使每個波的取樣點變平均。這兩種處理建議都試試看，不見得哪個方法會比較好。最後，處理好後就輸出成 Signed 16-bit raw 檔。





3. 下一步是用 python 去處理，因為 raw 裡面的資料是雙聲道的資料，但考量到 memory 可能不足，可以只取單聲道就好(這邊是取左聲道)。

```
with open('./Logic_final/tom_louder.raw', 'rb') as f:
    data = f.read()

str_init=data.hex()
str_final=""
str_list=[]
count=0
for i in range(len(str_init)):
    if(count%8==3 and i%4==3):
        str_final=""
#         print(str_init[i-3:i+1])
        str_final+=str_init[i-1]
        str_final+=str_init[i]
        str_final+=str_init[i-3]
        str_final+=str_init[i-2]
        str_list.append(str_final)
#         print(str_final[i-3:i+1])
    count+=1
print(len(str_list))
# print(len(str_list))
change=[]
sample=[]
order=1
for i in str_list:
    change.append(twosComplement_hex(i))
    sample.append(order)
    order+=1
plt.figure(figsize=(20, 20), dpi=100)
plt.scatter(sample, change)
plt.show()
path = 'output.txt'
with open(path, 'w') as f:
    for i in str_list:
        f.write(i)
        f.write(',\n')
```

最後輸出成 txt 檔。最後一步是把 txt 檔手動改成 coe 檔，更改的方法是貼上以下標頭
memory_initialization_radix=16;
memory_initialization_vector=
然後再把最後一個逗號改成分號。最後再改附檔名(改成 coe)。

4. 以上就是取樣前置作業的處理流程。接下來將會進入到 verilog 的部分，這邊主要的重點會 在如何計算 blk_mem_gen 的 addr 值，首先有幾個變數值可以先行定義(以下以小鼓為例):

(1) Tleft=使用的 clk 頻率/聲音的頻率 (e.g Tleft=100M/236)

```
assign Tleft_A=25000000/210;
```

(2) sound_addr_A=(clk_cnt*一個波大概多少個取樣點/Tleft_A+cnt_A*一個波大概多少個取樣點)%取樣素材的總取樣點;

```
assign sound_addr_A=(clk_cnt*64/Tleft_A+cnt_A*64)%4423;
```

(3) clk_cnt 為 counter 其中之一，clk_cnt 的值的範圍為[0,Tleft-1]。
(4) cnt_A 則是從 0 開始每當 clk_cnt 數到 Tleft-1 且音頻沒有變化，則 cnt_A 就可以加 1。

```
always @(posedge track_iter_cnt[1] or posedge reset)
    if (reset == 1'b1)
        begin
            clk_cnt <= 22'd0;
            cnt_A <= 29'd0;
            flag_A <= 0;
        end
    else
        begin
            clk_cnt <= clk_cnt_next;
            cnt_A <= next_cnt_A;
            flag_A <= next_flag_A;
        end

always @* begin
    clk_cnt_next=clk_cnt;
    next_cnt_A=cnt_A;
    if(play_or_not_o)next_flag_A=1;
    else next_flag_A=flag_A;
    if(flag_A)begin
        if (clk_cnt == Tleft_A-1)begin
            clk_cnt_next = 22'd0;
            next_flag_A=flag_A;
            next_cnt_A=cnt_A+1;
            if(cnt_A==29'd68)begin
                next_cnt_A=29'd0;
                next_flag_A=0;
            end
            else next_cnt_A=cnt_A+1;
        end
        else begin
            clk_cnt_next = clk_cnt + 1'b1;
            next_cnt_A=cnt_A;
            next_flag_A=flag_A;
        end
    end
end
```

5. 在講解完 address 後，接下來就是講我們撥放音樂的機制。
首先講解 FIXED MODE 的播放方式，這裡我們會有以下變數:
track_vec 是一個[15:0] 的一個變數，意思是在該個拍點上該樂器要不要發出聲音。舉例來說，若 track_vec=16’b0000000000000111，意思就是在 loop 中的前三個拍點會發出該樂器的聲音。
再來是 track_iter，意思是現在放到 16 個拍點中的哪個拍點。
play_or_not 意思是當下這個拍點要不要放出聲音
play_or_not=(!count)?track_vec[track_iter]:0;
這裡的 count 是用來降頻用，也是為了要生成 onepulse 訊號去讓樂器只發出一次聲響。

```
wire play_or_not,play_or_not_d,play_or_not_o;
reg count,next_count;
assign play_or_not=(!count)?track_vec[track_iter]:0;
debounce play_or_not_debounce(.clk(clk),.pb(play_or_not),.pb_debounced(play_or_not_d));
OnePulse play_or_not_onepulse(.clock(track_iter_cnt[1]),.signal(play_or_not_d),.signal_single_pulse(play_or_not_o));

always @(posedge track_iter_cnt[23], posedge reset) begin
    if (reset) begin
        count <= 0;
    end else begin
        count <= count+1;
    end
end
```

6. 最後講解 Free Mode 的播放方式，這裡會有幾個點和前面的 FIXED MODE 不同:

(1) 因為是旋律樂器，音頻會根據使用者的按鍵而變化。

(2) 每次並不一定會完整放完一個取樣聲音，若是使用者在一個完整取樣聲音放完前就按下一個按鍵，那就要提前結束並從頭再發出新按鍵頻率的聲音。

```
always @(posedge cnt[3] or posedge reset)
    if (reset == 1'b1)
        begin
            clk_cnt <= 22'd0;
            cnt_A <= 29'd0;
            flag_A <= 0;
            old_freq <= `silence1;
        end
    else
        begin
            clk_cnt <= clk_cnt_next;
            cnt_A <= next_cnt_A;
            flag_A <= next_flag_A;
            old_freq <= next_old_freq;
        end
```

```
always @* begin
    next_old_freq=freq;
    clk_cnt_next=clk_cnt;
    next_cnt_A=cnt_A;
    next_flag_A=flag_A;
    if(freq!=`silence1)begin
        if(old_freq!=freq)next_flag_A=1;
    end
    else next_flag_A=0;
    if(flag_A)begin
        if (clk_cnt == Tleft_A-1)begin
            clk_cnt_next = 22'd0;
            next_flag_A=flag_A;
            if(old_freq!=freq)begin
                next_cnt_A=29'd0;
            end else begin
                next_cnt_A=cnt_A+1;
                if(cnt_A==29'd50)begin
                    next_cnt_A=29'd0;
                    next_flag_A=0;
                end
            end
        end
        else begin
            if(old_freq!=freq)begin
                next_cnt_A=29'd0;
                clk_cnt_next=22'd0;
            end else begin
                clk_cnt_next = clk_cnt + 1'b1;
                next_cnt_A=cnt_A;
                next_flag_A=flag_A;
            end
        end
    end
end
```

因此實作上就會有些許的不同，像是原本的產生聲音是利用產生 One_pulse 訊號去控制，但現在是判斷頻率有沒有變，如果有變，那就要放出一次聲音。

Loop 系統與操作介面:

1. FSM: 當 switch[15] == 1, state = FREE; 當 switch[15] == 0, state = FIXED。

```
//state machine
always @(posedge clk, posedge rst) begin
    if (rst) begin
        state <= FIXED;
    end
    else begin
        state <= next_state;
    end
end

always @(*) begin
    if (sw[15]) next_state = FREE;
    else next_state = FIXED;
end
```

2. 使用變數 track_change, track_change_free 紀錄現在 FIXED state, FREE state 分別在哪一軌; 使用變數 track_select 紀錄現在在 FIXED state 時哪一格，都有設定保護機制，當你在此

state 時不會動到另一個 state 的變數值。

3. 共有三個 16 格的一維陣列: track0, track1, track2，分別代表 FIXED mode 三個音軌的節奏 pattern，此格值為 1 代表這一格要發出聲音，為 0 代表不用。判斷機制為按下空白鍵時將當前格數值從 0 改為 1，1 改為 0。

```
always @(*) begin
    next_track0 = track0;
    if (track_change == 0 && state == FIXED) begin
        if (key_down[KEY_CODES[15]] && last_down != KEY_CODES[15]) begin
            next_track0[track_select] = ~track0[track_select];
        end
    end
end
```

4. 設置兩變數 track_iter_cnt 和 track_iter，track_iter_cnt 每個 clk 會+1，加到 2^25 時會歸零，track_iter 就會根據 track_iter_cnt 判斷現在 loop 系統走到哪一格。

5. 將 track 陣列, track_iter_cnt, track_iter 傳入先前音樂取樣步驟設置好的 module，並設置好要輸出到哪個音軌。(以小鼓模組為例)

```
track_iterator_edm_snare track_iterator_edm_snare(
    .clk(clk),
    .reset(rst),
    .track_vec(track2),
    .track_iter_cnt(track_iter_cnt),
    .track_iter(track_iter),
    .volume(3'd1),
    .audio(audio_in_left_C)
);
```

6. 將輸出的 audio_in 傳給 speaker_control module，即可輸出音訊。

```
speaker_control sc3(
    .clk(clk),
    .rst(rst),
    .audio_in_left(audio_in_left_C),
    .audio_in_right(audio_in_right_C),
    .audio_mclk(audio_mclk_C),
    .audio_lrck(audio_lrck_C),
    .audio_sck(audio_sck_C),
    .audio_sdin(audio_sdin_C)
);
```

7. record 機制判斷:
如果 record_cnt(專門為 record 設置的 counter，數到 UP_BOUND 時代表在一個 loop 的最後一個 clock)數到 UP_BOUND，且 record 鍵在這個 loop 有被按下，則下一個 loop 開始錄音，若正在錄音，則下一個 loop 開始停止錄音。

```
always @(*) begin
    if (!recording && key_down[KEY_CODES[0]] && state == FREE) next_record_press = 1;
    else if (record_cnt == UP_BOUND) next_record_press = 0;
    else next_record_press = record_press;
end
```

```
always @(*) begin
    if (record_cnt == UP_BOUND && record_press) begin
        next_recording = 1;
    end
    else if (record_cnt == UP_BOUND && recording) begin
        next_recording = 0;
    end
    else next_recording = recording;
end
```

8. music_record 陣列用來記錄錄音當下 keyboard 按的 frequency。

```
if (recording) begin
    if (track_change_free == 0) begin
        music_record_0[record_id] <= next_freq_kb;
    end
    else begin
        music_record_1[record_id] <= next_freq_kb;
    end
end
```

9. 根據此音軌是否已有錄音判斷要輸出的 frequency 為何。

```
assign freq_vga_0 = (has_record_0 ? freq_record_0 : freq_kb);
assign freq_vga_1 = (has_record_1 ? freq_record_1 : freq_kb);
```

並將該 frequency 傳入音樂取樣部分的 module，設置好輸出音軌再交給 speaker_control 即可輸出音訊。

```
track_iterator__edm_sound edm_sound2(
    .clk(clk),
    .reset(rst),
    .volume(3'd1),
    .freq(freq_vga_1),
    .audio(audio_in_left_B)
);
```

10. VGA:

傳入 track 陣列，上面三行就會將陣列內容呈現在螢幕。截取部分 code 舉例，當我在該軌的 v_cnt 時，h_cnt 對應到的格子 track 值為 1 就畫紅色，否則白色。

```
else if (h_cnt < 38)
    if (track_change == 2 && track_select == 0) {vgaRed, vgaGreen, vgaBlue} = 12'h0f0;
    else if (track2[0]) {vgaRed, vgaGreen, vgaBlue} = 12'hf00;
    else {vgaRed, vgaGreen, vgaBlue} = 12'hfff;
else if (h_cnt < 78)
    if (track_change == 2 && track_select == 1) {vgaRed, vgaGreen, vgaBlue} = 12'h0f0;
    else if (track2[1]) {vgaRed, vgaGreen, vgaBlue} = 12'hf00;
```

傳入 FREE 音軌 frequency，下面兩行會根據不同頻率呈現不同顏色。截取部分 code 舉例，color 會根據 frequency 給予不同值，pixel_gen module 再將 color 值轉換為 RGB 輸出給 vga。

```
case (freq_vga_1)
    `c>>1: next_color_1 = 1;
    `d>>1: next_color_1 = 2;
    `e>>1: next_color_1 = 3;
    `f>>1: next_color_1 = 4;
    `g>>1: next_color_1 = 5;
    `a>>1: next_color_1 = 6;
    `b>>1: next_color_1 = 7;
    `silence: next_color_1 = 0;
endcase
```

```
case (color_1)
     0: {vgaRed, vgaGreen, vgaBlue} = 12'hfff;
     1: {vgaRed, vgaGreen, vgaBlue} = 12'hf00;
     2: {vgaRed, vgaGreen, vgaBlue} = 12'hf80;
     3: {vgaRed, vgaGreen, vgaBlue} = 12'hff0;
     4: {vgaRed, vgaGreen, vgaBlue} = 12'h0f0;
     5: {vgaRed, vgaGreen, vgaBlue} = 12'h00f;
     6: {vgaRed, vgaGreen, vgaBlue} = 12'h0ff;
     7: {vgaRed, vgaGreen, vgaBlue} = 12'hf0f;
endcase
```

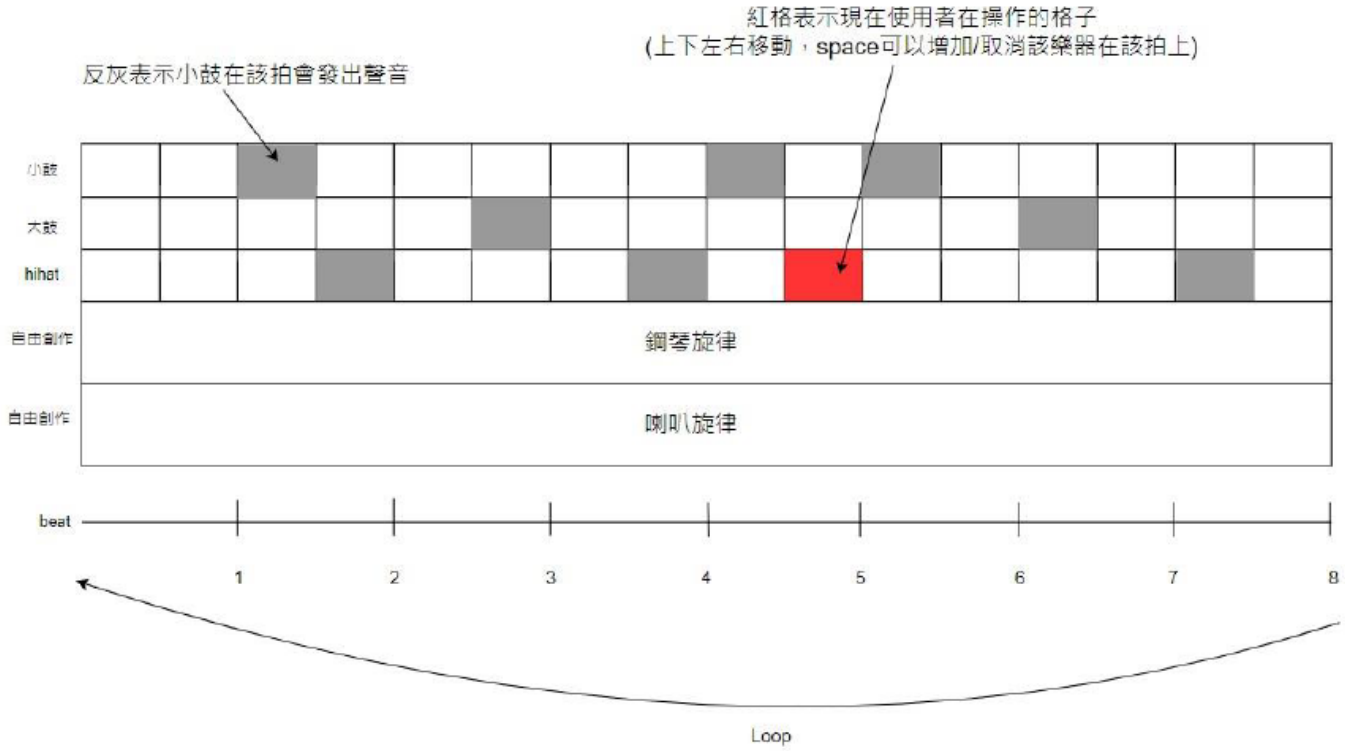
B. Problem Encountered

取樣部分碰到的困難主要是一開始真的沒什麼方向，所以花了很多時間去找尋資料，包含前面所提到的取樣素材的各種前置作業等等，都是在不斷的嘗試之後才整理出上述所提到的完整步驟。再來就是對於 blk_mem_gen 的 address 的計算，像是一開始就因為 clk 的頻率太高而造成音訊有許多雜音，後來經過除頻，並修正 addr 的計算方法，雜訊的部分也就有顯著的改善了。

C. 實作完成度/難易度說明/分工

1. 實作完成度:

Prposal 預期完成內容全數都有達成，另外加上電吉他音色(我們認為與鼓的聲音較適配)。附上 proposal 中的預期完成圖，與作出結果完全相符。



▲ 示意圖

另外我們也額外加入了現在播放到哪裡的指示線，是因為經過我們討論後，我們覺得這樣的設計可以更提升使用者的體驗。最後還有讓自由創作軌道隨著音符變色等等，目的都是為了還原真正的鼓機或是取樣機那種繽紛的介面感。

分工:

邱煒甯: 音樂取樣與音色 module 實作。

劉祥暉: Loop 系統與操作介面實作。

D. 心得

這次的 final project 雖然工程浩大，過程當中也碰到許多問題，但都在經過和組員互相討論，花時間去 debug 下，最後還是完成了當初 proposal 預想中的目標，在看到成品的當下是很有成就感的。過程當中，對 blk_mem、vga、Pmod 的操作更加的熟悉了，很開心能有這次的機會做出自己喜歡的作品，還能在過程當中也學習到各種知識，算是一舉兩得。最後總結一下這學期的心得，我們一路從不會寫 verilog，一直到操作各種設備（e.g. switch、button、LED、seven-segment、鍵盤、vga、pmod、car motor），雖說實作中碰到各種挫折，但每次的 lab 都能讓我們對該設備的熟悉度大幅提升，就這樣一路累積到最後，運用自己所學並做出一個 final project，真的是非常充實，也謝謝老師、助教們這一學期的指導，真的是好課值得一生推啊！