

**EECS207001 Project Report****Team : 25****Student ID : 109062108      Name : 陳寬宸****Student ID : 109062135      Name : 陳家輝**

---

**ML - Music Light****\*大綱\*****1. 前言****2. 動機****3. 材料簡介****4. Sound Sensor 介紹****5. LED Cube 介紹****6. Project Demo 影片****7. 結語****8. 參考資料**

## I. 前言

從 final exam 結束至今已然一個多月，在這段期間裡，我們從開始探索 project 要做什麼，查找資料和討論可行性，完成 proposal，到如今已經實作出一個完成度頗高的 project，這一路走來的過程，除了遇到很多技術上的困難，邊準備其他科的期末考邊實作 project 也是一項不小的挑戰。所幸，我們做到了。

現在在此一筆一字的寫下此份 report，不僅是為了完成作業亦或是介紹 project 細節，更是記錄我們這段時間以來的心路歷程。至此，在此份 report 內我們將以自己實作的 code 輔以網上資料搭配邏輯圖去介紹 project 的內容細節，期許可以讓看到這份 report 的讀者能更加了解 Music Light 這個可以和聲音互動的裝置是如何實作出來的。

## II. 動機

在討論 project 要做什麼的時候，其實我們的方案一變再變，從遊戲軟體到生活中會出現的硬體，例如：推箱子遊戲、停車場、彈珠檯等等。正巧已經十二月中旬，街道上甚至學校都已經佈置成聖誕節的氣氛，晚上走到哪裡都閃閃發亮的，我們便萌生了一個念頭，那便是做一個可以顯示很多不同圖案的 LED cube。而後又覺得只顯示圖案有點單調，於是我們決定將音樂融合進來，做一個可以根據音樂而有變化的聲音互動 Music Light Cube。

## III. 材料簡介

此次 project 裡我們主要運用到的材料有兩個——Waveshare Sound Sensor Module 以及 Arduino 4x4x4 RGB LED Cube Module。以下將針對兩者的產品細節做一些簡介

### i. Waveshare Sound Sensor Module



產品圖-LM386

此款 Sound Sensor 是由 Waveshare 出產，是一種音頻集成功率放大器，其因為自身功耗低、更新內鍾增益可調整，總諧波失真小，所以我們選擇此款 sound sensor 作為我們聲音的探測工具。

而 LM386 電壓增益內置為 20，但在 1 pin 腳和 8 pin 腳之間增加一只外接電阻和電容便可以將電壓增益為 200 以內的任意值。

圖中圓形凸起處為偵測聲音的麥克風，而外接的四個 pin 腳由上至下分別為 DOUT：數字量輸出；AOUT：模擬量輸出；GND：電源接地；VCC：電源正，範圍為 3.3V~5.5V。

利用這個 Sound sensor，我們可以檢測周圍環境聲音的有無，甚至可以判斷聲音強度的大小。

## ii. Arduino 4x4x4 RGB LED Cube Module



產品圖-WS2812B

此款 4x4x4 RGB LED Cube Module 可以多樣化隨意控制，由 64 顆彩色 LED 組合而成的 3D 模組，並且可以支援多個串接，可以同時將兩片或多片模組拼接起來。

雖然本模組是 For Arduino，但模組運用大同小異，所以也可以運用到 FPGA 上面。其 pin 腳總共有四個，Date in：數據輸入訊號；Data Out：數據輸出訊號；GND：電源接地；VCC：電源正，範圍 3.3V~5V。

利用這個模組，我們可以將輸入進來的聲音，根據聲音大小和節奏的快慢，將這些數值以 LED Cube 做圖案變化顯示。

## IV. Sound Sensor 介紹

在實作 project 的第一步，我們需要先了解 Sound Sensor 是如何運作。首先我們先將 Sound Sensor 連上 FPGA，然後寫了一個簡單的測試 module 去觀察 Sound Sensor 的運作原理。

```
assign LED_D = Dout;
assign LED_A = Aout;
```

在這個 module 裡我們先行測試的是是否接受的到聲音，我們將 Sound Sensor 輸入進來的 AOUT 和 DOUT 分別給到兩個 wire，並將這兩個 wire 輸出到 FPGA 的 LED 上，觀察是否可以亮燈。



測試的結果如我們所設想的一般，只要有聲音響起的時候，Sound Sensor 會偵測到，並將訊號給到 FPGA，使得 FPGA 的 LED 亮起。而在測試的過程中，我們發現，當 Sound Sensor 偵測到聲音時，其實他本身的 module 就會在元件上顯示亮燈，其亮燈的位置正是在 DAT。透過這個測試，我們順利的抓到聲音訊號並可以將之運用到 LED Cube 裡，然而我們並不滿足於此，我們想要更進一步知道聲音的大小，這樣才能做出更多變化



為了要抓到聲音的大小，我們在這個簡單的 module 裡多增加了一些東西，把 AOUT 的值抓進 FPGA 裡，並先假設一個數值做判斷，利用七段顯示器看看能不能知道聲音輸入進來的值為多少。

```

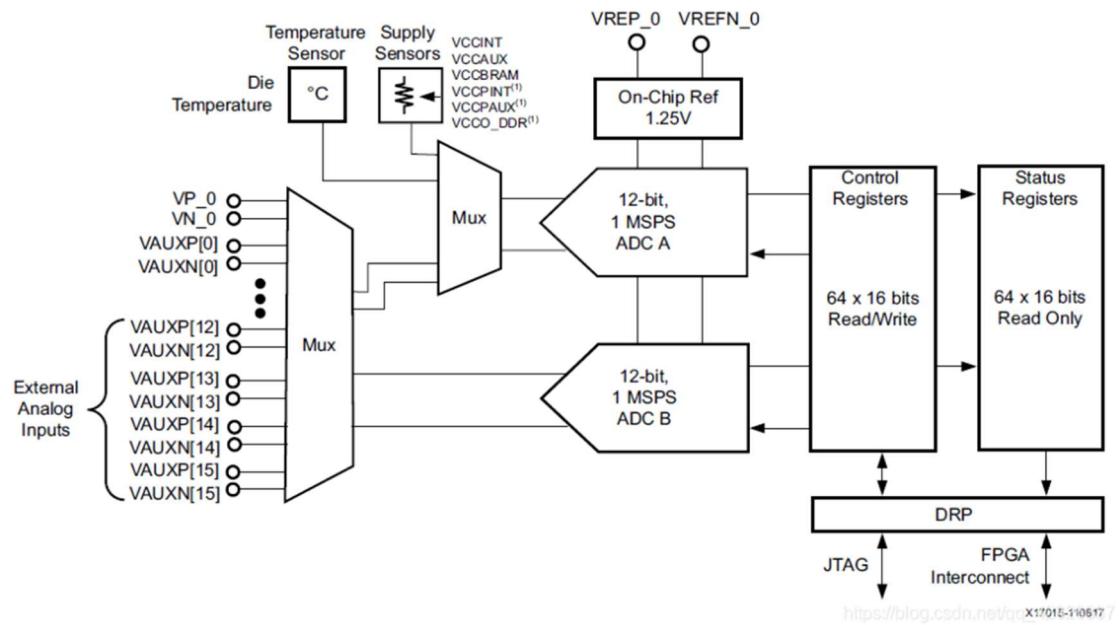
if(Aout >= 200) begin
    seg_3 = 7'b1111111;
    seg_2 = 7'b0010010;
    seg_1 = 7'b0000001;
    seg_0 = 7'b0000001;
end else if (Aout >= 150) begin
    seg_3 = 7'b1111111;
    seg_2 = 7'b1001111;
    seg_1 = 7'b0100100;
    seg_0 = 7'b0000001;
end else if(Aout >= 100) begin
    seg_3 = 7'b1111111;
    seg_2 = 7'b1001111;
    seg_1 = 7'b0000001;
    seg_0 = 7'b0000001;
end else if(Aout >= 50) begin
    seg_3 = 7'b1111111;
    seg_2 = 7'b1111111;
    seg_1 = 7'b0100100;
    seg_0 = 7'b0000001;
end else begin
    seg_3 = 7'b1111111;
    seg_2 = 7'b1111111;
    seg_1 = 7'b1111111;
    seg_0 = 7'b0000001;
end

```

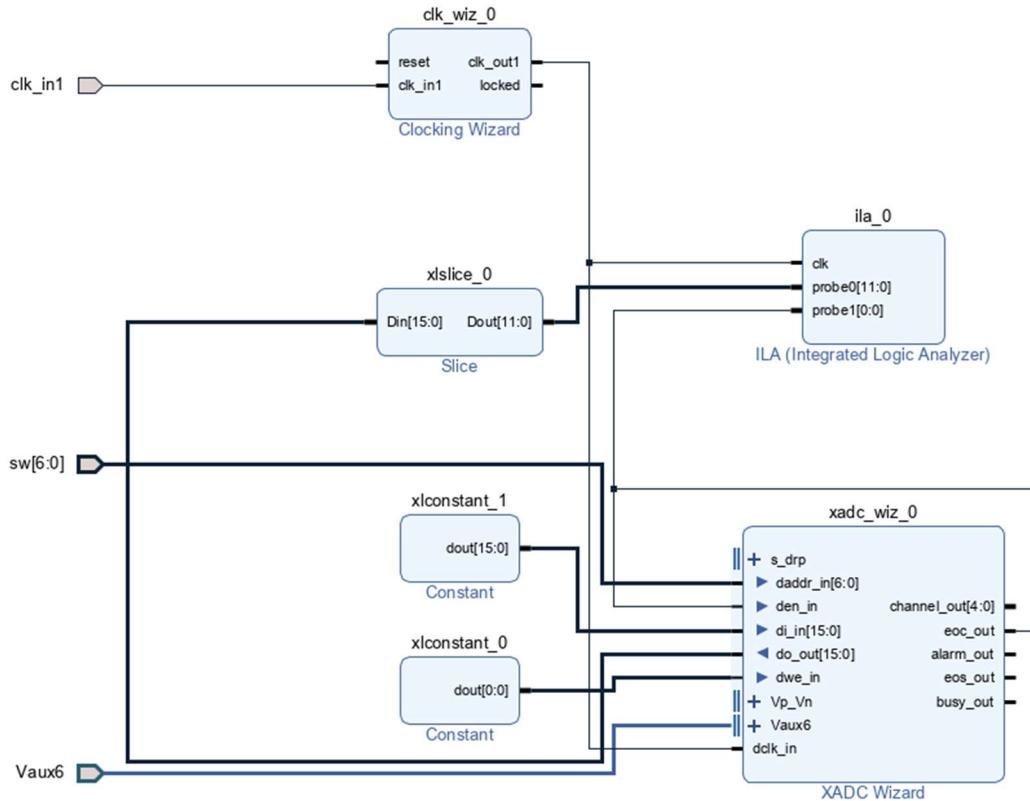
然而，此次測試並不如我們所願，其在七段顯示器上顯示不出數值，而且在我們多次更改判斷數值之後也仍然無法知道聲音輸入的值為何。所以我們上

Google 搜尋相關內容，這才知道如果要知道聲音大小，那麼就必須要將輸入進來為模擬訊號的聲音，把模擬訊號轉換成數位訊號，也就是所謂的模數轉換，這樣才能抓到我們想要的數值並且拿來運用。

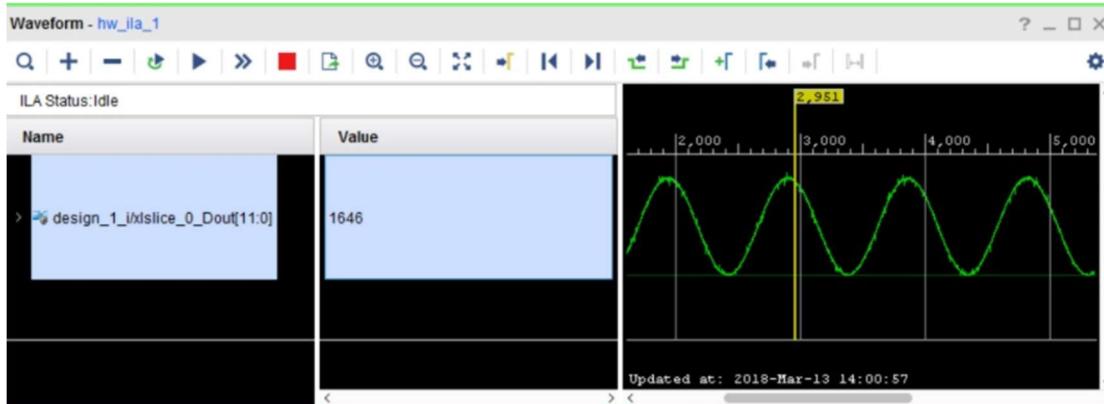
隨後我們便開始研究模數轉換，然而因為這個部分並不屬於上課內容，要實作起來難度頗高。在一開始還不了解模數轉換的時候，即便查了很多網站也是對其不甚了解，看著一堆又一堆的數學公式以及原理介紹，感覺已經超出了能力所及的範圍，而我們甚至為了要更了解模數轉換該如何運用，我們開始閱讀起 XILINX 給出的 XADC 使用手冊及系列指引。然而那一份份七八十頁的全英手冊也是讓人頭昏腦漲，即便硬著頭皮讀到與我們需要的東西相關的，在要實作的時候也僅僅是知道這個東西是在幹嘛，也無法實作出想要的東西。



在這段煎熬的時間裡，我們便一直往復在使用手冊以及 Google 網站裡徘徊，但是我們的努力也不算白費，在某次搜尋中，我們有找到一個較為完整的模數轉換教程，按照那個教程再搭配上我們所讀到的知識，一步一步利用 Block Design 將我們想要的東西給刻出來。



在此處我們使用 Vivado 自帶的 IP Catalog 裡的 xadc 做模數轉換，依照網站的教程將 I/O port 稍微做更改，並且多設置一個 ila 去做模擬量的觀察。



做到這裡我們已然可以產生一個較為像樣的模擬量，但是這依舊存在一個缺點——這個模擬量只有每當我們按下次 trigger 它才會跑出來一次，然而聲音的輸入應當要是無時無刻都在偵測，而不是等我們人工按一次 trigger 才會偵測一次，所以我們還需要繼續修改。

在修改的過程中，我們仍然繼續查找相關資料，期待可以有更多資源可以幫助我們實作出我們需要的。而就在此時，我們找到一個開源 code，同樣也是對於模數轉換的，於是我們利用這個開源 code，將輸入從 Single Channel 改成 Channel Sequencer，然後將 XADC Pmod 所連結的 4 個 differential analog pair 啟用，隨後將 Sound Sensor 的 AOUT 利用 Vaux 通道輸入進 FPGA，而後再運用

XADC 採集模擬訊號，並可以使用 XDAC PMOD 連接器訪問 Basys 3 XADC 模擬輸入，從而可以訪問 XADC channel 6,7,14,15。

	Basic	ADC Setup	Alarms	Channel Sequencer	Summary
VCCINT				<input type="checkbox"/>	
VCCAUX				<input type="checkbox"/>	
VCCBRAM				<input type="checkbox"/>	
VP/VN				<input type="checkbox"/>	
VREFP				<input type="checkbox"/>	
VREFN				<input type="checkbox"/>	
vauxp0/vauxn0				<input type="checkbox"/>	
vauxp1/vauxn1				<input type="checkbox"/>	
vauxp2/vauxn2				<input type="checkbox"/>	
vauxp3/vauxn3				<input type="checkbox"/>	
vauxp4/vauxn4				<input type="checkbox"/>	
vauxp5/vauxn5				<input type="checkbox"/>	
vauxp6/vauxn6				<input checked="" type="checkbox"/>	
vauxp7/vauxn7				<input checked="" type="checkbox"/>	
vauxp8/vauxn8				<input type="checkbox"/>	
vauxp9/vauxn9				<input type="checkbox"/>	
vauxp10/vauxn10				<input type="checkbox"/>	
vauxp11/vauxn11				<input type="checkbox"/>	
vauxp12/vauxn12				<input type="checkbox"/>	
vauxp13/vauxn13				<input type="checkbox"/>	
vauxp14/vauxn14				<input checked="" type="checkbox"/>	
vauxp15/vauxn15				<input checked="" type="checkbox"/>	

再來因為我們的模擬訊號為一直在變化，所以我們的 Interface Options 要選擇 DRP，而後因為我們不需要偵測硬體狀態發出警報，所以我們將 alarms 的地方全部勾掉，其他的設定則維持默認。

Interface Options

---

AXI4Lite  DRP  None

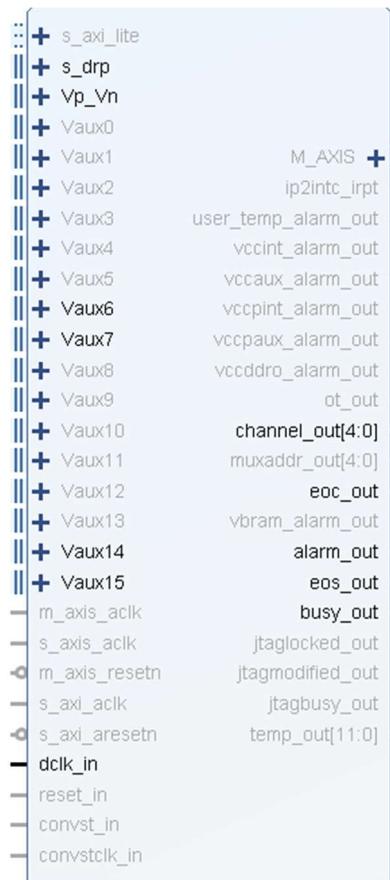
Startup Channel Selection

---

Simultaneous Selection  
 Independent ADC  
 Single Channel  
 Channel Sequencer

<input type="checkbox"/> Over Temperature Alarm (°C)	<input type="checkbox"/> User Temperature Alarm (°C)
Trigger <input type="text" value="125.0"/> [-40.0 - 125.0]	Trigger <input type="text" value="85.0"/> [-40.0 - 125.0]
Reset <input type="text" value="70.0"/> [-40.0 - 125.0]	Reset <input type="text" value="60.0"/> [-40.0 - 125.0]
<input type="checkbox"/> VCCINT Alarm (Volts)	
Lower <input type="text" value="0.97"/> [0.0 - 1.05]	Lower <input type="text" value="1.75"/> [0.0 - 1.89]
Upper <input type="text" value="1.03"/> [0.0 - 1.05]	Upper <input type="text" value="1.89"/> [0.0 - 1.89]
<input type="checkbox"/> VCCBRAM Alarm (Volts)	
Lower <input type="text" value="0.95"/> [0.0 - 1.05]	
Upper <input type="text" value="1.05"/> [0.0 - 1.05]	

完成 XADC 這個 block 的設定之後，所生成出來的 IP 便是我們所需的模數轉換的 Block。隨後我們再將轉換完的模擬訊號傳到 module 裡，做一個 binary to decimal 的 convert，最後得到的 out 值為 b2d\_out，即為我們聲音模擬訊號的差分值。



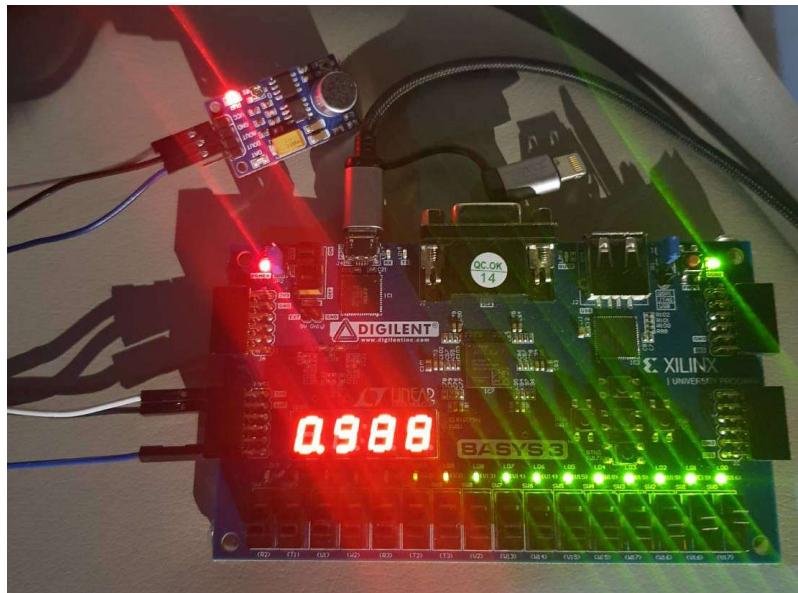
而為了要更好的觀察我們 module 的正確性以及聲音靈敏度及模數轉換後的值，我們又寫了一個 seven segment 將 b2d\_out 的值給到 sseg\_data 並顯示在七段顯示器上，而後抓 data 的後 3 個 bit 作為亮 led 的值，這樣 FPGA 上的 led 會隨著值得大小做一個聲量升降的效果

```

    always @ (posedge(CLK100MHZ)) begin
        if(ready == 1'b1) begin
            case (data[15:12])
                1: led <= 16'b11;
                2: led <= 16'b111;
                3: led <= 16'b1111;
                4: led <= 16'b11111;
                5: led <= 16'b111111;
                6: led <= 16'b1111111;
                7: led <= 16'b11111111;
                8: led <= 16'b111111111;
                9: led <= 16'b1111111111;
                10: led <= 16'b11111111111;
                11: led <= 16'b111111111111;
                12: led <= 16'b1111111111111;
                13: led <= 16'b11111111111111;
                14: led <= 16'b111111111111111;
                15: led <= 16'b111111111111111;
            default: led <= 16'b1;
            endcase
        end
    end

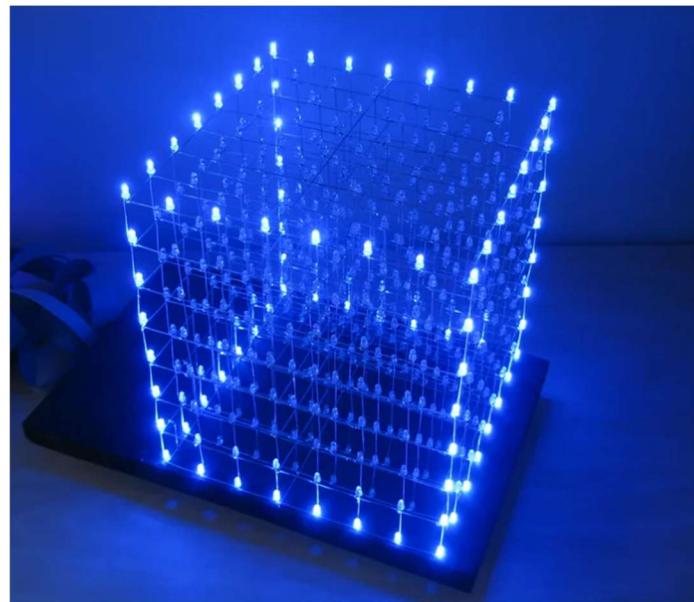
```

至此，我們便完成了第一部分，抓取聲音大小進行模數轉換，並準備將數位值給到 LED Cube 去做運用



## V. LED Cube 介紹

在做這份 project 的一開始，也就是還在撰寫 proposal 的時候，我們在查找資料的過程中，其實最初挑選到的是 8x8x8 的 LED Cube，因為我們有找到一個網站，裡面詳細的介紹了如何從零開始手工打造自己的 LED Cube。然而因為 8x8x8 的 LED Cube 規模較為龐大，而且手工焊接共 512 個 LED 形成 Cube 也是很花時間，在閱讀完前半部分的資料後，我們便放棄了這個念頭，轉而使用 4x4x4 的 LED Cube 並且串聯兩個形成 8x4x4。雖然沒能實作 8x8x8 的 LED Cube，但是在閱讀那份資料的過程也著實幫助了我們在之後實作 project 的路上少了一些彎路。



開始實作 LED Cube 之前，我們首先閱讀了 WS2812B-LED 的使用手冊，得知這個模組幾個重要的資訊——①其顏色為 GRB 而非 RGB；②每個 LED 燈的顏色由 24bit 組成；③LED 模組無法辨識 24bit 裡每 1bit 的 0 或 1，需要利用不同佔空比的 pwm 表示。

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

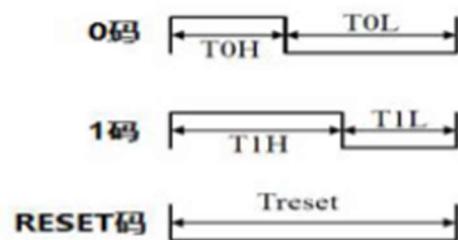
要讓 pwm 做數據的傳輸，需要一些時間，透過查找網站資料並搭配 pwm 相關的 datasheet，得知 pwm 要傳輸 0 和 1 的時候，對應到的時序波形圖不同，輸入的碼型也隨之不同，因此這裡要求的高低電平時間也有所規範。

### 数据传输时间

TOH	0 码, 高电平时间	220ns~380ns
T1H	1 码, 高电平时间	580ns~1.6μs
T0L	0 码, 低电平时间	580ns~1.6μs
T1L	1 码, 低电平时间	220ns~420ns
RES	帧单位, 低电平时间	280μs 以上

### 时序波形图

输入码型:



根據網站資料及閱讀到的知識，我們先開一個 module 去實作 pwm，首先要定義的便是 T0H, T0L, T1H 的值。因為網站資料裡給的是 ns，但我們在利用 verilog 計時的時候，是運用 counter 去數 clk 數，數到一定的 clk 數才做反應。因此我們查詢了 FPGA 的 clk 頻率，得知為 100MHz 後，再去做乘法便可以得出我們需要跑幾個 clk 才可以達到所需的時間範圍。

```
//clk 100MHz
parameter T0H = 8'd30; //300ns*100MHz
parameter T0L = 8'd100; //1000ns*100MHz
parameter T1H = 8'd100; //1000ns*100MHz
```

隨後便是利用 counter 去計算 clk 數，當 counter 達到 T0H + T0L 的時候，代表我們的這個 bit 做完了，便可以繼續做下一個 bit。

```
//cnt
always @(*) begin
    if(start==1'b0) next_cnt = 8'd1;
    else next_cnt = (cnt == T0H + T0L ? 8'd1 : cnt+1'b1);
end

//bit
always @(*) begin
    if(cnt == T0H + T0L) next_bit = (bit==5'd23 ? 5'd0 : bit+1'b1);
    else next_bit = bit;
end
```

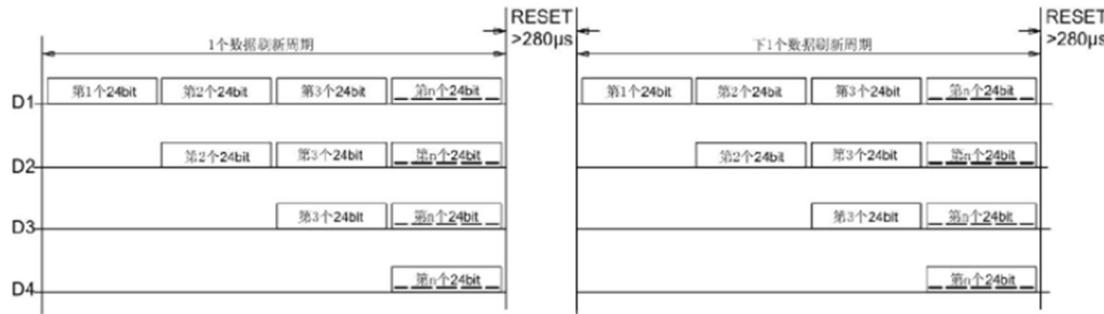
因為一個 LED 燈的顏色有 24 個 bit，所以我們要做完一種顏色需要重複循環 24 次，直到做完 24 次之後，代表一個顏色做完，done 才會拉起來，此時 start 會歸零。而在上一層的 module 接收到 done 才會指派下一個 LED 燈給 pwm 繼續轉換，start 此時才回又拉回 1。

```
//done
assign done = start & (bit == 5'd23) & (cnt == T0H + T0L);
```

而在這裡值得注意的是，GRB 的 bit 是由高到低，所以我們在做轉換的時候也要考慮這一點，於是便有了以下的 code。

```
//pwm
always @(*) begin
    if(!start) next_pwm = 1'b0;
    else
        if(GRB[23-bit] == 1'b0) next_pwm = (cnt <= T0H); //0 Hight Power
        else next_pwm = (cnt <= T1H); //1 Hight Power
end
```

這樣即為一個 LED 燈，而因為我們的 LED Cube 規模為 8x4x4，意即在上層 module 裡，每做一次顏色刷新就要做 128 次 pwm，做完這麼多次 pwm 之後，我們需要給他一個復位時間，這樣才可以重新下一個輪迴。



再來我們需要控制每一個 LED 燈應該要亮什麼顏色。在這個部分，因為 LED 為三維空間，如果我們展平成一維變數會需要透過一定的計算很是複雜，而且在後續的維護也很不好用，在查找過很多的相關資料卻沒有太多的建設性資訊。就在苦苦思求這樣的三維空間要如何表示的時候，我們想到了 lab 曾經實作過的 memory，其就是利用二維數組去表示一維。於是我們便大膽的設計了一個四維數組去控制我們的三維 LED Cube，而因為這方面沒有任何的相關資料，我們在實作的時候也是戰戰兢兢，但若是這個想法得以實現，那麼在後續維護 LED Cube 的時候就會變得非常容易

```
reg [23:0] G [7:0] [3:0] [3:0];
```

在第一維其所代表的是顏色，所以有 24 個 bit，後面三維則是分別代表 x, y, z。而因為我們的 LED Cube 亮燈並不會是同時亮燈，正如前面所述，他會有一個延遲時間，但是對於我們人的肉眼來說，那個延遲時間極短，無法辨識得出先後順序，所以看起來是同時亮暗，然而 LED 却是一個接一個的去做亮暗，也就是說，我們會需要一個變數承接 128 的數值，這樣才能讓每一個 LED 跑過一次，而在此處，我們便開了一個 7bit 的 number，讓這個 number 可以使 pwm 跑 128 次，刷新每一個 LED 的顏色。

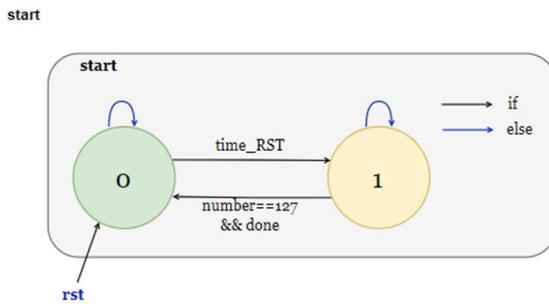
```
reg [6:0] number, next_number; //2^7 = 128
```

有了這個 number 之後，我們便可以完成上面所述的 pwm 的一個輪迴，當每次接收到 pwm 的 done 訊號拉起時，代表一個燈號做好了，則 number 會加 1，直到 number 做到 127 的時候，意即所有的 LED 都刷新過了一次，則此時便要進入 RST 的復位時間，等待這個時間過了我們才去做下一個輪迴。

```
//number
always @(*) next_number = number + done;

//start
always @(*) begin
    if(number == 7'd127 && done) next_start = 1'b0; //to RST
    else if(time_RST) next_start = 1'b1; //restart
    else next_start = start;
end

parameter RST = 15'd30000; //300μs*100Mhz
```



而後我們需要利用這個 number 去決定此時為哪一個 LED 燈要去刷新。在此處，我們設計了一個 to\_xyz 的 module 將 number 的一維轉成三維，而這個轉換的過程也算簡單，經過一點簡單的計算便可以完成。

```

assign x = number/16;
assign y = (number/4)%4;
assign z = number%4;
  
```

然而這裡有一個小小的问题，也就是在此處的運算裡摻雜著 / 和 %，在 verilog 裡面不是一個好的 coding 方式，於是我們上網搜尋了一些轉換的資料，並且加以運算，便將其改成了一個較好的形式，而這樣我們便可以再之後 project 的實作中，更好的維護我們的四維數組，也就是 LED Cube。

```

assign x = (number>>3'd4); //number/16
assign y = ((number>>3'd2)&2'b11); // (number/4)%4
assign z = (number&2'b11); //number%4
  
```

因為之後在控制 LED Cube 的亮暗以及做一些圖形的變換時，我們需要用到不同 LED 燈的延遲時間，所以我們需要一個 count 的 module 幫我們做出不同時間的值。在此處我們首先先找出要延遲一秒需要幾個 clk，這個運算在之前便已做過，只需要將 clk 的 pinlv 乘上要的時間便可以得到變數的值。

```
`define t1 35'd100000000 //1s
```

其他時間以此類推。隨後我們便寫一個 count 的 module，在裡面定義 35bit 的 count 變數去算經過了多少個 clk。35bit 的 clk 計數器換算過來約等於五分鐘也就是 300 秒，相信足以應付我們後面所需的延遲時間。

```
reg [34:0] count, next_count;
```

我們將需要延遲的時間以 parameter t 的形式傳進來當成我們要計數的 clk 數。

```

`define t248 35'd24800000000
`define t234 35'd23300000000
`define t196 35'd19500000000
`define t179 35'd17800000000
`define t159 35'd15800000000
`define t140 35'd14000000000
`define t122 35'd12100000000
`define t83 35'd8300000000 //83s
`define t66 35'd6500000000 //66s
`define t9 35'd900000000 //9s
`define t3 35'd300000000 //3s
`define t1 35'd100000000 //1s
`define t06 35'd60000000 //0.6s
`define t05 35'd50000000 //0.5s
`define t04 35'd40000000 //0.4s
`define t01 35'd10000000 //0.1s
`define t0099 35'd9900000 //0.099s
`define t0098 35'd9800000 //0.098s
`define t0097 35'd9700000 //0.097s
`define t0096 35'd9600000 //0.096s
`define t0095 35'd9500000 //0.095s
`define t0094 35'd9400000 //0.094s
`define t0093 35'd9300000 //0.093s
`define t005 35'd5000000 //0.05s
`define t001 35'd1000000 //0.01s
`define t005 35'd500000 //0.005s

```

```

module COUNT #(parameter t = 2)(
    COUNT #(`t1) count_1(clk, rst, time_1);

```

會這麼 define 的原因是因為我們發現這麼寫 code 會比較好看，而且也更能精確的抓出這個 count 是在數多久。而因為我們需要不同的時間，所以在上層的 module 裡會有一堆 count module 的實作以及 wire 接線。

```

COUNT #(`t1) count_1(clk, rst, time_1);
COUNT #(`t01) count_2(clk, rst, time_01);
COUNT #(`RST) count_3(clk, rst | (start), time_RST);
COUNT #(`t001) count_4(clk, rst, time_001);
COUNT #(`t3) count_5(clk, rst, time_3);
COUNT #(`t005) count_6(clk, rst, time_005);
COUNT #(`t05) count_7(clk, rst, time_05);
COUNT #(`t04) count_8(clk, rst, time_04);
COUNT #(`t06) count_9(clk, rst, time_06);
COUNT #(`t0093) count_0093(clk, rst, time_0093);
COUNT #(`t0094) count_10(clk, rst, time_0094);
COUNT #(`t0095) count_11(clk, rst, time_0095);
COUNT #(`t0096) count_12(clk, rst, time_0096);
COUNT #(`t0097) count_13(clk, rst, time_0097);
COUNT #(`t0098) count_14(clk, rst, time_0098);
COUNT #(`t0099) count_15(clk, rst, time_0099);
COUNT #(`t0005) count_16(clk, rst, time_0005);
COUNT #(`t248) count_17(clk, rst, time_248);
COUNT #(`t234) count_18(clk, rst, time_234);
COUNT #(`t196) count_19(clk, rst, time_196);
COUNT #(`t179) count_20(clk, rst, time_179);
COUNT #(`t159) count_21(clk, rst, time_159);
COUNT #(`t122) count_22(clk, rst, time_122);
COUNT #(`t83) count_23(clk, rst, time_83);
COUNT #(`t66) count_24(clk, rst, time_66);
COUNT #(`t140) count_25(clk, rst, time_140);
COUNT #(`t9) count_26(clk, rst, time_9);

```

在做完這些基礎的設定之後，我們總算要將腳步邁入設計 LED Cube 的燈型以及圖案。首先要先定義我們所需的顏色，此時我們有找到一個好用的網站，可以讓我們選取喜歡的顏色並且轉換為 24bit 的對應。

## 挑選圖片上的顏色



然而網站轉換出來的顏色色碼為 RGB，而我們的顏色為 GRB，需要注意到這一點才不會導致顏色怪怪的。之後我們挑選幾個喜歡的顏色將其定義在 module 裡。

```
`define whit 24'hffff00 //white
`define dark 24'h000000
`define red 24'h8ef591
`define ored 24'h8efd01
`define oran 24'h45ff00
`define yora 24'h8ffe01
`define yell 24'hccfb00
`define ygre 24'hff007f
`define gree 24'hb52450
`define bgre 24'h8b1e7c
`define blue 24'h6541af
`define bvio 24'h4275ff
`define viol 24'h1f7aa0
`define rvio 24'h31abaa
```

隨後我們再設計一個 combinational block 讓其可以變換顏色，3 秒後轉換為下一個顏色，以達到我們預期的七彩轉換的效果。

```
//mode1color
always @(*) begin
    if(time_3) next_mode1color = (mode1color==4'd11 ? 4'd0 : mode1color + 4'd1);
    else next_mode1color = mode1color;
end

//mode1GRB
always @(*) begin
    case(mode1color)
        4'd0: mode1GRB = `red;
        4'd1: mode1GRB = `ored;
        4'd2: mode1GRB = `oran;
        4'd3: mode1GRB = `yora;
        4'd4: mode1GRB = `yell;
        4'd5: mode1GRB = `ygre;
        4'd6: mode1GRB = `gree;
        4'd7: mode1GRB = `bgre;
        4'd8: mode1GRB = `blue;
        4'd9: mode1GRB = `bvio;
        4'd10: mode1GRB = `viol;
        4'd11: mode1GRB = `rvio;
        default: mode1GRB = `red;
    endcase
end
```

再來我們設計一個 led 的 module，其將 clk，rst 和我們 Sound Sensor 傳出來 16bit 的值以及一個 3bit 的 sel\_mode 用來之後不同模式之間的切換，而 output 則為先前的 pwm，之後便可以開始設計我們的 mode 了。

```
module LED(
    input clk,
    input rst,
    input [2:0] sel_mode,
    input [15:0] voice,
    output pwm
);
```

### 追尋你的點點滴滴模式：

在這個 Mode 裡我們本想要測試 LED Cube 每一個燈是否可以正常的亮起並且顏色是否正確，但後來發現效果還不錯便加以改進成為一個模式。在這個模式裡，我們讓每一個 LED 依序亮起，亮完一個便換下一個並且前一個變暗，所以我們利用一個 sequential block 幫助我們完成這件事

```
else if(mode==3'd7) begin //test
    G[mode7x2][mode7y2][mode7z2] <= mode1GRB;
    G[mode7x1][mode7y1][mode7z1] <= `dark;
end
```

值得一提的是，我們在這裡會使用 sequential block 的原因是因為發現這種形式的應用對於四維數組來說比較好維護。而後因為 sequential block 是由 clk 驅動，意即我們上面的 code 會使 LED 燈一閃而逝，肉眼根本無法捕捉，所以我們在下面利用一個 combinational block 將其做一個 0.05 秒時間延遲。

```
//mode7state
always @(*) begin
    if(mode==3'd7) next_mode7state = mode7state + time_005;
    else next_mode7state = 3'b0;
end
```

### 聽到你的聲音便心動模式：

在這個模式裡我們設計的是——當聲音超過一定範圍值，也就是 Sound Sensor 接收到聲音時，我們便讓 LED Cube 顯示一顆愛心，而如果沒有接收到聲音時即維持 LED Cube 為全暗。

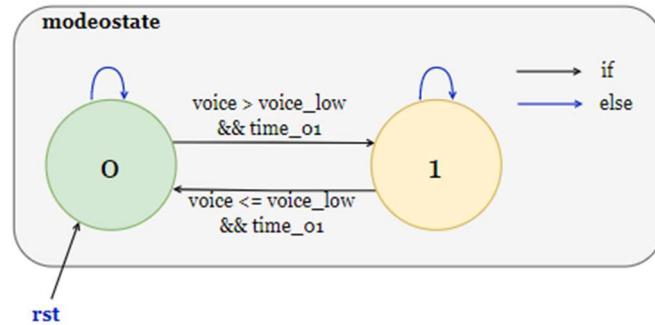
為了實作這個模式，我們不能像上面測試模式一般只針對兩個燈去做賦值，我們需要以整個 LED Cube 為整體去做改變。於是我們將 Cube 切成四個面，每一個面皆為 xz 平面，而後再稍加排版一下 code，便可以讓 LED Cube 可以以一個平面的方式呈現在我們的 code 裡，隨後再去賦值即可。

```
//y0
G[0][0][0] <= 'dark; G[1][0][0] <= 'dark; G[2][0][0] <= 'dark; G[3][0][0] <= 'dark; G[4][0][0] <= 'dark; G[5][0][0] <= 'dark; G[6][0][0] <= 'dark; G[7][0][0] <= 'dark;
G[0][0][1] <= 'dark; G[1][0][1] <= 'dark; G[2][0][1] <= 'dark; G[3][0][1] <= 'dark; G[4][0][1] <= 'dark; G[5][0][1] <= 'dark; G[6][0][1] <= 'dark; G[7][0][1] <= 'dark;
G[0][0][2] <= 'dark; G[1][0][2] <= 'dark; G[2][0][2] <= 'dark; G[3][0][2] <= 'dark; G[4][0][2] <= 'dark; G[5][0][2] <= 'dark; G[6][0][2] <= 'dark; G[7][0][2] <= 'dark;
G[0][1][0] <= 'dark; G[1][1][0] <= 'dark; G[2][1][0] <= 'dark; G[3][1][0] <= 'dark; G[4][1][0] <= 'dark; G[5][1][0] <= 'dark; G[6][1][0] <= 'dark; G[7][1][0] <= 'dark;
G[0][1][1] <= 'dark; G[1][1][1] <= 'dark; G[2][1][1] <= 'dark; G[3][1][1] <= 'dark; G[4][1][1] <= 'dark; G[5][1][1] <= 'dark; G[6][1][1] <= 'dark; G[7][1][1] <= 'dark;
G[0][1][2] <= 'dark; G[1][1][2] <= 'dark; G[2][1][2] <= 'dark; G[3][1][2] <= 'dark; G[4][1][2] <= 'dark; G[5][1][2] <= 'dark; G[6][1][2] <= 'dark; G[7][1][2] <= 'dark;
G[0][2][0] <= 'dark; G[1][2][0] <= 'dark; G[2][2][0] <= 'dark; G[3][2][0] <= 'dark; G[4][2][0] <= 'dark; G[5][2][0] <= 'dark; G[6][2][0] <= 'dark; G[7][2][0] <= 'dark;
G[0][2][1] <= 'dark; G[1][2][1] <= 'dark; G[2][2][1] <= 'dark; G[3][2][1] <= 'dark; G[4][2][1] <= 'dark; G[5][2][1] <= 'dark; G[6][2][1] <= 'dark; G[7][2][1] <= 'dark;
G[0][2][2] <= 'dark; G[1][2][2] <= 'dark; G[2][2][2] <= 'dark; G[3][2][2] <= 'dark; G[4][2][2] <= 'dark; G[5][2][2] <= 'dark; G[6][2][2] <= 'dark; G[7][2][2] <= 'dark;
G[0][3][0] <= 'dark; G[1][3][0] <= 'dark; G[2][3][0] <= 'dark; G[3][3][0] <= 'dark; G[4][3][0] <= 'dark; G[5][3][0] <= 'dark; G[6][3][0] <= 'dark; G[7][3][0] <= 'dark;
G[0][3][1] <= 'dark; G[1][3][1] <= 'dark; G[2][3][1] <= 'dark; G[3][3][1] <= 'dark; G[4][3][1] <= 'dark; G[5][3][1] <= 'dark; G[6][3][1] <= 'dark; G[7][3][1] <= 'dark;
G[0][3][2] <= 'dark; G[1][3][2] <= 'dark; G[2][3][2] <= 'dark; G[3][3][2] <= 'dark; G[4][3][2] <= 'dark; G[5][3][2] <= 'dark; G[6][3][2] <= 'dark; G[7][3][2] <= 'dark;
G[0][4][0] <= 'dark; G[1][4][0] <= 'dark; G[2][4][0] <= 'dark; G[3][4][0] <= 'dark; G[4][4][0] <= 'dark; G[5][4][0] <= 'dark; G[6][4][0] <= 'dark; G[7][4][0] <= 'dark;
G[0][4][1] <= 'dark; G[1][4][1] <= 'dark; G[2][4][1] <= 'dark; G[3][4][1] <= 'dark; G[4][4][1] <= 'dark; G[5][4][1] <= 'dark; G[6][4][1] <= 'dark; G[7][4][1] <= 'dark;
G[0][4][2] <= 'dark; G[1][4][2] <= 'dark; G[2][4][2] <= 'dark; G[3][4][2] <= 'dark; G[4][4][2] <= 'dark; G[5][4][2] <= 'dark; G[6][4][2] <= 'dark; G[7][4][2] <= 'dark;
G[0][5][0] <= 'dark; G[1][5][0] <= 'dark; G[2][5][0] <= 'dark; G[3][5][0] <= 'dark; G[4][5][0] <= 'dark; G[5][5][0] <= 'dark; G[6][5][0] <= 'dark; G[7][5][0] <= 'dark;
G[0][5][1] <= 'dark; G[1][5][1] <= 'dark; G[2][5][1] <= 'dark; G[3][5][1] <= 'dark; G[4][5][1] <= 'dark; G[5][5][1] <= 'dark; G[6][5][1] <= 'dark; G[7][5][1] <= 'dark;
G[0][5][2] <= 'dark; G[1][5][2] <= 'dark; G[2][5][2] <= 'dark; G[3][5][2] <= 'dark; G[4][5][2] <= 'dark; G[5][5][2] <= 'dark; G[6][5][2] <= 'dark; G[7][5][2] <= 'dark;
G[0][6][0] <= 'dark; G[1][6][0] <= 'dark; G[2][6][0] <= 'dark; G[3][6][0] <= 'dark; G[4][6][0] <= 'dark; G[5][6][0] <= 'dark; G[6][6][0] <= 'dark; G[7][6][0] <= 'dark;
G[0][6][1] <= 'dark; G[1][6][1] <= 'dark; G[2][6][1] <= 'dark; G[3][6][1] <= 'dark; G[4][6][1] <= 'dark; G[5][6][1] <= 'dark; G[6][6][1] <= 'dark; G[7][6][1] <= 'dark;
G[0][6][2] <= 'dark; G[1][6][2] <= 'dark; G[2][6][2] <= 'dark; G[3][6][2] <= 'dark; G[4][6][2] <= 'dark; G[5][6][2] <= 'dark; G[6][6][2] <= 'dark; G[7][6][2] <= 'dark;
G[0][7][0] <= 'dark; G[1][7][0] <= 'dark; G[2][7][0] <= 'dark; G[3][7][0] <= 'dark; G[4][7][0] <= 'dark; G[5][7][0] <= 'dark; G[6][7][0] <= 'dark; G[7][7][0] <= 'dark;
```

而後我們便將傳進來的聲音值去做判斷，但因為我們的四位模組也就是 LED Cube 是由 clk 所維護，如果判斷到聲音，LED 的亮起只有一個 clk，肉眼無法捕捉得到，所以我們在此也需要進行時間的延遲，保證其亮起和變暗不會太過迅速。

```
//mode0state
always @(*) begin
    case(mode0state)
        1'b0: next_mode0state = (voice > `voice_low && time_01);
        1'b1: next_mode0state = (voice <= `voice_low && time_01 ? 1'b0: 1'b1);
        default: next_mode0state = 1'b0;
    endcase
end
```

mode0state



這裡我們做了一個小巧思，也就是當 Sound Sensor 沒有偵測到聲音的時候，它仍然會等 0.1 秒才會變暗，這樣便不會瞬間閃一下然後讓觀看者眼睛很刺。隨後我們便運用我們的空間感在 LED Cube 找一個愛心出來，並把其對應的位置賦值給顏色。

```

//y0
G[0][0][3] <- 'dark'; G[1][0][3] <- 'dark'; G[2][0][3] <- 'dark'; G[3][0][3] <- 'dark'; G[4][0][3] <- 'dark'; G[5][0][3] <- 'dark'; G[6][0][3] <- 'dark'; G[7][0][3] <- 'dark';
G[0][0][2] <- 'dark'; G[1][0][2] <- 'dark'; G[2][0][2] <- 'dark'; G[3][0][2] <- 'dark'; G[4][0][2] <- 'dark'; G[5][0][2] <- 'dark'; G[6][0][2] <- 'dark'; G[7][0][2] <- 'dark';
G[0][0][1] <- 'dark'; G[1][0][1] <- 'dark'; G[2][0][1] <- 'dark'; G[3][0][1] <- 'dark'; G[4][0][1] <- 'dark'; G[5][0][1] <- 'dark'; G[6][0][1] <- 'dark'; G[7][0][1] <- 'dark';
G[0][0][0] <- 'dark'; G[1][0][0] <- 'dark'; G[2][0][0] <- 'dark'; G[3][0][0] <- 'dark'; G[4][0][0] <- 'dark'; G[5][0][0] <- 'dark'; G[6][0][0] <- 'dark'; G[7][0][0] <- 'dark';
//y1
G[0][1][3] <- 'dark'; G[1][1][3] <- 'dark'; G[2][1][3] <- 'dark'; G[3][1][3] <- 'dark'; G[4][1][3] <- 'dark'; G[5][1][3] <- 'dark'; G[6][1][3] <- 'dark'; G[7][1][3] <- 'dark';
G[0][1][2] <- 'dark'; G[1][1][2] <- 'dark'; G[2][1][2] <- mode1GRB; G[3][1][2] <- mode1GRB; G[4][1][2] <- mode1GRB; G[5][1][2] <- 'dark'; G[6][1][2] <- 'dark'; G[7][1][2] <- 'dark';
G[0][1][1] <- 'dark'; G[1][1][1] <- 'dark'; G[2][1][1] <- 'dark'; G[3][1][1] <- mode1GRB; G[4][1][1] <- mode1GRB; G[5][1][1] <- 'dark'; G[6][1][1] <- 'dark'; G[7][1][1] <- 'dark';
G[0][1][0] <- 'dark'; G[1][1][0] <- 'dark'; G[2][1][0] <- 'dark'; G[3][1][0] <- mode1GRB; G[4][1][0] <- mode1GRB; G[5][1][0] <- 'dark'; G[6][1][0] <- 'dark'; G[7][1][0] <- 'dark';
//y2
G[0][2][3] <- 'dark'; G[1][2][3] <- 'dark'; G[2][2][3] <- mode1GRB; G[3][2][3] <- 'dark'; G[4][2][3] <- mode1GRB; G[5][2][3] <- 'dark'; G[6][2][3] <- 'dark'; G[7][2][3] <- 'dark';
G[0][2][2] <- 'dark'; G[1][2][2] <- mode1GRB; G[2][2][2] <- mode1GRB; G[3][2][2] <- mode1GRB; G[4][2][2] <- mode1GRB; G[5][2][2] <- mode1GRB; G[6][2][2] <- 'dark'; G[7][2][2] <- 'dark';
G[0][2][1] <- 'dark'; G[1][2][1] <- mode1GRB; G[2][2][1] <- mode1GRB; G[3][2][1] <- mode1GRB; G[4][2][1] <- mode1GRB; G[5][2][1] <- mode1GRB; G[6][2][1] <- 'dark'; G[7][2][1] <- 'dark';
G[0][2][0] <- 'dark'; G[1][2][0] <- 'dark'; G[2][2][0] <- 'dark'; G[3][2][0] <- mode1GRB; G[4][2][0] <- mode1GRB; G[5][2][0] <- 'dark'; G[6][2][0] <- 'dark'; G[7][2][0] <- 'dark';
//y3
G[0][3][3] <- 'dark'; G[1][3][3] <- mode1GRB; G[2][3][3] <- mode1GRB; G[3][3][3] <- 'dark'; G[4][3][3] <- mode1GRB; G[5][3][3] <- mode1GRB; G[6][3][3] <- 'dark'; G[7][3][3] <- 'dark';
G[0][3][2] <- 'dark'; G[1][3][2] <- mode1GRB; G[2][3][2] <- mode1GRB; G[3][3][2] <- mode1GRB; G[4][3][2] <- mode1GRB; G[5][3][2] <- mode1GRB; G[6][3][2] <- mode1GRB; G[7][3][2] <- 'dark';
G[0][3][1] <- 'dark'; G[1][3][1] <- mode1GRB; G[2][3][1] <- mode1GRB; G[3][3][1] <- mode1GRB; G[4][3][1] <- mode1GRB; G[5][3][1] <- mode1GRB; G[6][3][1] <- 'dark'; G[7][3][1] <- 'dark';
G[0][3][0] <- 'dark'; G[1][3][0] <- 'dark'; G[2][3][0] <- 'dark'; G[3][3][0] <- mode1GRB; G[4][3][0] <- mode1GRB; G[5][3][0] <- 'dark'; G[6][3][0] <- 'dark'; G[7][3][0] <- 'dark';
  
```

### 點點星光為你閃爍模式：

在這個模式裡我們將實作每一個 yz 平面的點沿著 x 軸橫向移動，然後將其速度做不同的落差，便可以形成一點一點的光芒在做交錯變換的感覺。

為了實作出這個更為複雜的模式，如果我們依舊按照愛心模式一個一個去賦值的話，整份 code 會變得超級長。所以我們想出再用一個四維模組包在原本的四維模組裡，這樣便可以節省很多 code。

```

//y0
mode1x[0][0][3] <= 0;
mode1x[0][0][2] <= 0;
mode1x[0][0][1] <= 0;
mode1x[0][0][0] <= 0;
//y1
mode1x[0][1][3] <= 0;
mode1x[0][1][2] <= 0;
mode1x[0][1][1] <= 0;
mode1x[0][1][0] <= 0;
//y2
mode1x[0][2][3] <= 0;
mode1x[0][2][2] <= 0;
mode1x[0][2][1] <= 0;
mode1x[0][2][0] <= 0;
//y3
mode1x[0][3][3] <= 0;
mode1x[0][3][2] <= 0;
mode1x[0][3][1] <= 0;
mode1x[0][3][0] <= 0;
  
```

這個四維模組記下的便是每一個 LED 燈在 x, y, z 軸上的位置。而因為我們僅要讓 LED 燈在 x 軸上移動，y, z 軸是不用改變的，所以我們的 y 和 z 直接複製給對應位置即可。

```
//mode1y          //mode1z
always @(posedge clk) begin always @(posedge clk) begin
  //y0           //y0
  mode1y[0][0][3] <= 0; mode1z[0][0][3] <= 3;
  mode1y[0][0][2] <= 0; mode1z[0][0][2] <= 2;
  mode1y[0][0][1] <= 0; mode1z[0][0][1] <= 1;
  mode1y[0][0][0] <= 0; mode1z[0][0][0] <= 0;
  //y1           //y1z
  mode1y[0][1][3] <= 1; mode1z[0][1][3] <= 3;
  mode1y[0][1][2] <= 1; mode1z[0][1][2] <= 2;
  mode1y[0][1][1] <= 1; mode1z[0][1][1] <= 1;
  mode1y[0][1][0] <= 1; mode1z[0][1][0] <= 0;
  //y2           //y2z
  mode1y[0][2][3] <= 2; mode1z[0][2][3] <= 3;
  mode1y[0][2][2] <= 2; mode1z[0][2][2] <= 2;
  mode1y[0][2][1] <= 2; mode1z[0][2][1] <= 1;
  mode1y[0][2][0] <= 2; mode1z[0][2][0] <= 0;
  //y3           //y3
  mode1y[0][3][3] <= 3; mode1z[0][3][3] <= 3;
  mode1y[0][3][2] <= 3; mode1z[0][3][2] <= 2;
  mode1y[0][3][1] <= 3; mode1z[0][3][1] <= 1;
  mode1y[0][3][0] <= 3; mode1z[0][3][0] <= 0;
end           end
```

隨後為了要做出移動的效果，我們需要讓當前的 LED 燈在下一個 time 往左或往右移動一格，所以我們在此設計一個動量，讓 mode1x 的值可以根據動量決定要往右動一格或是往左。

```
//y0
mode1dx[0][0][3] <= (mode1dx[0][0][3] && mode1x[0][0][3] != 3'd7) || mode1x[0][0][3] == 3'd0;
mode1dx[0][0][2] <= (mode1dx[0][0][2] && mode1x[0][0][2] != 3'd7) || mode1x[0][0][2] == 3'd0;
mode1dx[0][0][1] <= (mode1dx[0][0][1] && mode1x[0][0][1] != 3'd7) || mode1x[0][0][1] == 3'd0;
mode1dx[0][0][0] <= (mode1dx[0][0][0] && mode1x[0][0][0] != 3'd7) || mode1x[0][0][0] == 3'd0;
//y1
mode1dx[0][1][3] <= (mode1dx[0][1][3] && mode1x[0][1][3] != 3'd7) || mode1x[0][1][3] == 3'd0;
mode1dx[0][1][2] <= (mode1dx[0][1][2] && mode1x[0][1][2] != 3'd7) || mode1x[0][1][2] == 3'd0;
mode1dx[0][1][1] <= (mode1dx[0][1][1] && mode1x[0][1][1] != 3'd7) || mode1x[0][1][1] == 3'd0;
mode1dx[0][1][0] <= (mode1dx[0][1][0] && mode1x[0][1][0] != 3'd7) || mode1x[0][1][0] == 3'd0;
//y2
mode1dx[0][2][3] <= (mode1dx[0][2][3] && mode1x[0][2][3] != 3'd7) || mode1x[0][2][3] == 3'd0;
mode1dx[0][2][2] <= (mode1dx[0][2][2] && mode1x[0][2][2] != 3'd7) || mode1x[0][2][2] == 3'd0;
mode1dx[0][2][1] <= (mode1dx[0][2][1] && mode1x[0][2][1] != 3'd7) || mode1x[0][2][1] == 3'd0;
mode1dx[0][2][0] <= (mode1dx[0][2][0] && mode1x[0][2][0] != 3'd7) || mode1x[0][2][0] == 3'd0;
//y3
mode1dx[0][3][3] <= (mode1dx[0][3][3] && mode1x[0][3][3] != 3'd7) || mode1x[0][3][3] == 3'd0;
mode1dx[0][3][2] <= (mode1dx[0][3][2] && mode1x[0][3][2] != 3'd7) || mode1x[0][3][2] == 3'd0;
mode1dx[0][3][1] <= (mode1dx[0][3][1] && mode1x[0][3][1] != 3'd7) || mode1x[0][3][1] == 3'd0;
mode1dx[0][3][0] <= (mode1dx[0][3][0] && mode1x[0][3][0] != 3'd7) || mode1x[0][3][0] == 3'd0;
```

當 dx 為 1 且當前位置不為 x 軸的最後一格或是當我們位於原點的時候，則會讓 LED 繼續沿著 x 軸往右移動。隨後我們為了要製作出變換效果，我們讓不同 LED 燈重新賦值的時間不一樣，也就是做出不同的時間延遲便可以達到效果。

```

if(time_001) begin
    mode1x[0][0][3] <= mode1x[0][0][3] + (mode1dx[0][0][3] ? 1 : -1);
end
if(time_0099) begin
    mode1x[0][0][2] <= mode1x[0][0][2] + (mode1dx[0][0][2] ? 1 : -1);
    mode1x[0][1][3] <= mode1x[0][1][3] + (mode1dx[0][1][3] ? 1 : -1);
end
if(time_0098) begin
    mode1x[0][0][1] <= mode1x[0][0][1] + (mode1dx[0][0][1] ? 1 : -1);
    mode1x[0][1][2] <= mode1x[0][1][2] + (mode1dx[0][1][2] ? 1 : -1);
    mode1x[0][2][3] <= mode1x[0][2][3] + (mode1dx[0][2][3] ? 1 : -1);
end
if(time_0097) begin
    mode1x[0][0][0] <= mode1x[0][0][0] + (mode1dx[0][0][0] ? 1 : -1);
    mode1x[0][1][1] <= mode1x[0][1][1] + (mode1dx[0][1][1] ? 1 : -1);
    mode1x[0][2][2] <= mode1x[0][2][2] + (mode1dx[0][2][2] ? 1 : -1);
    mode1x[0][3][3] <= mode1x[0][3][3] + (mode1dx[0][3][3] ? 1 : -1);
end
if(time_0096) begin
    mode1x[0][1][0] <= mode1x[0][1][0] + (mode1dx[0][1][0] ? 1 : -1);
    mode1x[0][2][1] <= mode1x[0][2][1] + (mode1dx[0][2][1] ? 1 : -1);
    mode1x[0][3][2] <= mode1x[0][3][2] + (mode1dx[0][3][2] ? 1 : -1);
end
if(time_0095) begin
    mode1x[0][2][0] <= mode1x[0][2][0] + (mode1dx[0][2][0] ? 1 : -1);
    mode1x[0][3][1] <= mode1x[0][3][1] + (mode1dx[0][3][1] ? 1 : -1);
end
if(time_0094) begin
    mode1x[0][3][0] <= mode1x[0][3][0] + (mode1dx[0][3][0] ? 1 : -1);
end

```

最後我們僅需把這個四維數組存的值放入到我們 LED Cube 的四維數組便可以時間 LED Cube 的沿著 x 軸變換顏色。

```

//y0
G[mode1x[0][0][3]][mode1y[0][0][3]][mode1z[0][0][3]] <= mode1GRB;
G[mode1x[0][0][2]][mode1y[0][0][2]][mode1z[0][0][2]] <= mode1GRB;
G[mode1x[0][0][1]][mode1y[0][0][1]][mode1z[0][0][1]] <= mode1GRB;
G[mode1x[0][0][0]][mode1y[0][0][0]][mode1z[0][0][0]] <= mode1GRB;
//y1
G[mode1x[0][1][3]][mode1y[0][1][3]][mode1z[0][1][3]] <= mode1GRB;
G[mode1x[0][1][2]][mode1y[0][1][2]][mode1z[0][1][2]] <= mode1GRB;
G[mode1x[0][1][1]][mode1y[0][1][1]][mode1z[0][1][1]] <= mode1GRB;
G[mode1x[0][1][0]][mode1y[0][1][0]][mode1z[0][1][0]] <= mode1GRB;
//y2
G[mode1x[0][2][3]][mode1y[0][2][3]][mode1z[0][2][3]] <= mode1GRB;
G[mode1x[0][2][2]][mode1y[0][2][2]][mode1z[0][2][2]] <= mode1GRB;
G[mode1x[0][2][1]][mode1y[0][2][1]][mode1z[0][2][1]] <= mode1GRB;
G[mode1x[0][2][0]][mode1y[0][2][0]][mode1z[0][2][0]] <= mode1GRB;
//y3
G[mode1x[0][3][3]][mode1y[0][3][3]][mode1z[0][3][3]] <= mode1GRB;
G[mode1x[0][3][2]][mode1y[0][3][2]][mode1z[0][3][2]] <= mode1GRB;
G[mode1x[0][3][1]][mode1y[0][3][1]][mode1z[0][3][1]] <= mode1GRB;
G[mode1x[0][3][0]][mode1y[0][3][0]][mode1z[0][3][0]] <= mode1GRB;

```

平靜的心因你而小鹿亂撞模式：

在這個模式裡，我們將實作一個類似於前一個模式的上下移動版本，也就是以 xy 平面沿著 z 軸移動，但做出的變動便是要設計時間差，在過程中會逐漸由初始平坦狀態變換到點點燈光皆位於不同切面閃爍靈動，到得最後，又漸漸恢復平坦，正如心動一般，再平靜的心也因你而小鹿亂撞。

在實作的時候，我們本來想說直接以上一個模式的模板下去改寫，但發現開這麼多四維數組會導致 look up table 爆掉，所以我們將不會動的 xy 平面以常數給值，並且將 z 軸的位置平展成一維，對應 32 格的 xy 平面。

```
mode6z[0] <= 3;
mode6z[1] <= 3;
mode6z[2] <= 3;
mode6z[3] <= 3;
mode6z[4] <= 3;
mode6z[5] <= 3;
mode6z[6] <= 3;
mode6z[7] <= 3;
mode6z[8] <= 3;
mode6z[9] <= 3;
mode6z[10] <= 3;
mode6z[11] <= 3;
mode6z[12] <= 3;
mode6z[13] <= 3;
mode6z[14] <= 3;
mode6z[15] <= 3;
mode6z[16] <= 3;
mode6z[17] <= 3;
mode6z[18] <= 3;
mode6z[19] <= 3;
mode6z[20] <= 3;
mode6z[21] <= 3;
mode6z[22] <= 3;
mode6z[23] <= 3;
mode6z[24] <= 3;
mode6z[25] <= 3;
mode6z[26] <= 3;
mode6z[27] <= 3;
mode6z[28] <= 3;
mode6z[29] <= 3;
mode6z[30] <= 3;
mode6z[31] <= 3;
```

此處將初始值設為 3 即代表一開始 xy 平面會在 z 軸的最上面。隨後移動的方式與上個模式相同，同樣運用動量去讓每一格 LED 燈決定要往上移動一格或往下。

```
//mod6dz
always @ (posedge clk) begin
    mode6dz[0] <= (mode6dz[0] && mode6z[0] != 2'd3) || mode6z[0] == 2'd0;
    mode6dz[1] <= (mode6dz[1] && mode6z[1] != 2'd3) || mode6z[1] == 2'd0;
    mode6dz[2] <= (mode6dz[2] && mode6z[2] != 2'd3) || mode6z[2] == 2'd0;
    mode6dz[3] <= (mode6dz[3] && mode6z[3] != 2'd3) || mode6z[3] == 2'd0;
    mode6dz[4] <= (mode6dz[4] && mode6z[4] != 2'd3) || mode6z[4] == 2'd0;
    mode6dz[5] <= (mode6dz[5] && mode6z[5] != 2'd3) || mode6z[5] == 2'd0;
    mode6dz[6] <= (mode6dz[6] && mode6z[6] != 2'd3) || mode6z[6] == 2'd0;
    mode6dz[7] <= (mode6dz[7] && mode6z[7] != 2'd3) || mode6z[7] == 2'd0;
    mode6dz[8] <= (mode6dz[8] && mode6z[8] != 2'd3) || mode6z[8] == 2'd0;
    mode6dz[9] <= (mode6dz[9] && mode6z[9] != 2'd3) || mode6z[9] == 2'd0;
    mode6dz[10] <= (mode6dz[10] && mode6z[10] != 2'd3) || mode6z[10] == 2'd0;
    mode6dz[11] <= (mode6dz[11] && mode6z[11] != 2'd3) || mode6z[11] == 2'd0;
    mode6dz[12] <= (mode6dz[12] && mode6z[12] != 2'd3) || mode6z[12] == 2'd0;
    mode6dz[13] <= (mode6dz[13] && mode6z[13] != 2'd3) || mode6z[13] == 2'd0;
    mode6dz[14] <= (mode6dz[14] && mode6z[14] != 2'd3) || mode6z[14] == 2'd0;
    mode6dz[15] <= (mode6dz[15] && mode6z[15] != 2'd3) || mode6z[15] == 2'd0;
    mode6dz[16] <= (mode6dz[16] && mode6z[16] != 2'd3) || mode6z[16] == 2'd0;
    mode6dz[17] <= (mode6dz[17] && mode6z[17] != 2'd3) || mode6z[17] == 2'd0;
    mode6dz[18] <= (mode6dz[18] && mode6z[18] != 2'd3) || mode6z[18] == 2'd0;
    mode6dz[19] <= (mode6dz[19] && mode6z[19] != 2'd3) || mode6z[19] == 2'd0;
    mode6dz[20] <= (mode6dz[20] && mode6z[20] != 2'd3) || mode6z[20] == 2'd0;
    mode6dz[21] <= (mode6dz[21] && mode6z[21] != 2'd3) || mode6z[21] == 2'd0;
    mode6dz[22] <= (mode6dz[22] && mode6z[22] != 2'd3) || mode6z[22] == 2'd0;
    mode6dz[23] <= (mode6dz[23] && mode6z[23] != 2'd3) || mode6z[23] == 2'd0;
    mode6dz[24] <= (mode6dz[24] && mode6z[24] != 2'd3) || mode6z[24] == 2'd0;
    mode6dz[25] <= (mode6dz[25] && mode6z[25] != 2'd3) || mode6z[25] == 2'd0;
    mode6dz[26] <= (mode6dz[26] && mode6z[26] != 2'd3) || mode6z[26] == 2'd0;
    mode6dz[27] <= (mode6dz[27] && mode6z[27] != 2'd3) || mode6z[27] == 2'd0;
    mode6dz[28] <= (mode6dz[28] && mode6z[28] != 2'd3) || mode6z[28] == 2'd0;
    mode6dz[29] <= (mode6dz[29] && mode6z[29] != 2'd3) || mode6z[29] == 2'd0;
    mode6dz[30] <= (mode6dz[30] && mode6z[30] != 2'd3) || mode6z[30] == 2'd0;
    mode6dz[31] <= (mode6dz[31] && mode6z[31] != 2'd3) || mode6z[31] == 2'd0;
end
```

最後再將時間差稍微設計一下以讓其移動方式到得最後可以恢復成原位，之後才再繼續下一次擺動。

```

if(time_01) begin
    mode6z[0] <= mode6z[0] + (mode6dz[0] ? 1 : -1);
    mode6z[8] <= mode6z[8] + (mode6dz[8] ? 1 : -1);
    mode6z[16] <= mode6z[16] + (mode6dz[16] ? 1 : -1);
    mode6z[24] <= mode6z[24] + (mode6dz[24] ? 1 : -1);
end
if(time_0099) begin
    mode6z[1] <= mode6z[1] + (mode6dz[1] ? 1 : -1);
    mode6z[9] <= mode6z[9] + (mode6dz[9] ? 1 : -1);
    mode6z[17] <= mode6z[17] + (mode6dz[17] ? 1 : -1);
    mode6z[25] <= mode6z[25] + (mode6dz[25] ? 1 : -1);
end
if(time_0098) begin
    mode6z[2] <= mode6z[2] + (mode6dz[2] ? 1 : -1);
    mode6z[10] <= mode6z[10] + (mode6dz[10] ? 1 : -1);
    mode6z[18] <= mode6z[18] + (mode6dz[18] ? 1 : -1);
    mode6z[26] <= mode6z[26] + (mode6dz[26] ? 1 : -1);
end
if(time_0097) begin
    mode6z[3] <= mode6z[3] + (mode6dz[3] ? 1 : -1);
    mode6z[11] <= mode6z[11] + (mode6dz[11] ? 1 : -1);
    mode6z[19] <= mode6z[19] + (mode6dz[19] ? 1 : -1);
    mode6z[27] <= mode6z[27] + (mode6dz[27] ? 1 : -1);
end
if(time_0096) begin
    mode6z[4] <= mode6z[4] + (mode6dz[4] ? 1 : -1);
    mode6z[12] <= mode6z[12] + (mode6dz[12] ? 1 : -1);
    mode6z[20] <= mode6z[20] + (mode6dz[20] ? 1 : -1);
    mode6z[28] <= mode6z[28] + (mode6dz[28] ? 1 : -1);
end
if(time_0095) begin
    mode6z[5] <= mode6z[5] + (mode6dz[5] ? 1 : -1);
    mode6z[13] <= mode6z[13] + (mode6dz[13] ? 1 : -1);
    mode6z[21] <= mode6z[21] + (mode6dz[21] ? 1 : -1);
    mode6z[29] <= mode6z[29] + (mode6dz[29] ? 1 : -1);
end
if(time_0094) begin
    mode6z[6] <= mode6z[6] + (mode6dz[6] ? 1 : -1);
    mode6z[14] <= mode6z[14] + (mode6dz[14] ? 1 : -1);
    mode6z[22] <= mode6z[22] + (mode6dz[22] ? 1 : -1);
    mode6z[30] <= mode6z[30] + (mode6dz[30] ? 1 : -1);
end
if(time_0093) begin
    mode6z[7] <= mode6z[7] + (mode6dz[7] ? 1 : -1);
    mode6z[15] <= mode6z[15] + (mode6dz[15] ? 1 : -1);
    mode6z[23] <= mode6z[23] + (mode6dz[23] ? 1 : -1);
    mode6z[31] <= mode6z[31] + (mode6dz[31] ? 1 : -1);
end

```

### 盛開之音樂生命樹模式：

在此模式中，我們將實作可以依據聲音大小做出變化的燈光效果，其反應模式類似音樂噴泉，只是衝到最頂端時音樂之樹會綻放開來，最後落下點點星光。

實作這個模式，我們將 LED Cube 以 xy 平面去做切分，也就是想像我們從上往下觀察這個 Cube，於是我們先初始化最底層中間的四個 LED 燈為恆亮。

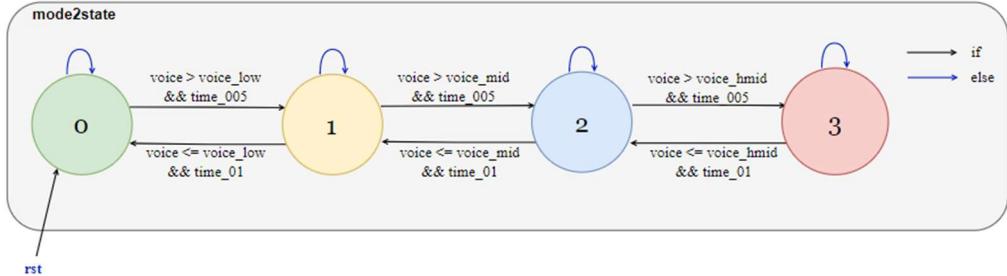
```
//z = 0
G[3][1][0] <= mode1GRB; G[4][1][0] <= mode1GRB;
G[3][2][0] <= mode1GRB; G[4][2][0] <= mode1GRB;
```

隨後我們根據 Sound Sensor 偵測的聲音大小，做出一個 finite state machine 判斷現在的音樂樹該成長到什麼程度。

```

case(mode2state)
  3'd0: next_mode2state = (voice > `voice_low && time_005) + mode2state;
  3'd1: next_mode2state = mode2state - (voice <= `voice_low && time_01) + (voice > `voice_mid && time_005);
  3'd2: next_mode2state = mode2state - (voice <= `voice_mid && time_01) + (voice > `voice_hmid && time_005);
  3'd3: next_mode2state = mode2state - (voice <= `voice_hmid && time_01);
  default: next_mode2state = 3'd0;
endcase
mode2state

```



在這裡我們做了一個小巧思，當聲音未達範圍標準時，我們延遲的時間為 0.1 秒，而達到標準往上時的延遲時間為 0.05，這樣呈現的效果便會是較容易往上成長，並不會馬上就掉下來。而後我們再根據 state 的轉換設計每一個 state 往上成長的音樂樹。

```

//z = 1
if(mode2state<3'd1) begin
  G[3][1][1] <= `dark; G[4][1][1] <= `dark;
  G[3][2][1] <= `dark; G[4][2][1] <= `dark;
end else begin
  G[3][1][1] <= mode1GRB; G[4][1][1] <= mode1GRB;
  G[3][2][1] <= mode1GRB; G[4][2][1] <= mode1GRB;
end
//z = 2
if(mode2state<3'd2) begin
  G[3][1][2] <= `dark; G[4][1][2] <= `dark;
  G[3][2][2] <= `dark; G[4][2][2] <= `dark;
end else begin
  G[3][1][2] <= mode1GRB; G[4][1][2] <= mode1GRB;
  G[3][2][2] <= mode1GRB; G[4][2][2] <= mode1GRB;
end
//z = 3
if(mode2state<3'd3) begin
  G[3][1][3] <= `dark; G[4][1][3] <= `dark;
  G[3][2][3] <= `dark; G[4][2][3] <= `dark;
end else begin
  G[3][1][3] <= mode1GRB; G[4][1][3] <= mode1GRB;
  G[3][2][3] <= mode1GRB; G[4][2][3] <= mode1GRB;
end

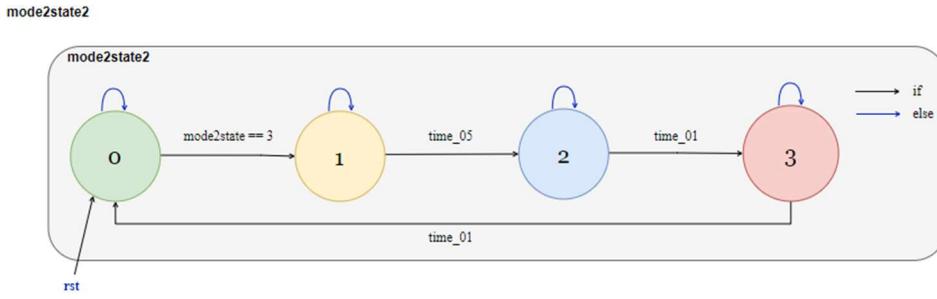
```

隨後當音樂樹成長到最高的時候，它便會盛放，於是我們需要再寫一個 finite state machine 去轉換盛放時的 LED 燈移動。

```

//mode2state2
always @(*) begin
  case(mode2state2)
    2'd0: next_mode2state2 = (mode2state==3'd3) + mode2state2;
    2'd1: next_mode2state2 = time_05 + mode2state2;
    2'd2: next_mode2state2 = time_01 + mode2state2;
    2'd3: next_mode2state2 = time_01 + mode2state2;
    default: next_mode2state2 = 0;
  endcase
end

```



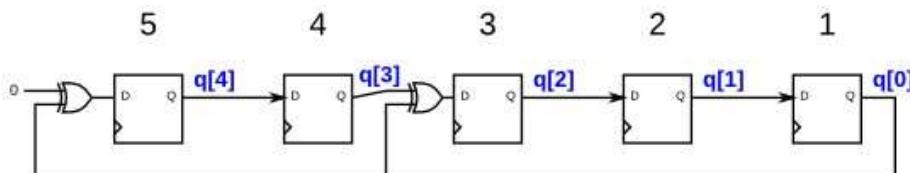
再來便是根據 state 去設計每一個 state 的音樂樹綻放會是何種圖案，而因為我們的四維數組以 xy 平面去切，在指定位置設值的時候，僅需從上往下看，並且將 code 稍作排版，便能在 code 裡呈現一個平面的 yz。

```

//z = 3 one
if(mode2state2==2'd1) begin
    G[2][0][3] <= mode1GRB; G[3][0][3] <= mode1GRB; G[4][0][3] <= mode1GRB; G[5][0][3] <= mode1GRB;
    G[2][1][3] <= mode1GRB; G[5][1][3] <= mode1GRB;
    G[2][2][3] <= mode1GRB; G[5][2][3] <= mode1GRB;
    G[2][3][3] <= mode1GRB; G[3][3][3] <= mode1GRB; G[4][3][3] <= mode1GRB; G[5][3][3] <= mode1GRB;
end else begin
    G[2][0][3] <= `dark; G[3][0][3] <= `dark; G[4][0][3] <= `dark; G[5][0][3] <= `dark;
    G[2][1][3] <= `dark; G[5][1][3] <= `dark;
    G[2][2][3] <= `dark; G[5][2][3] <= `dark;
    G[2][3][3] <= `dark; G[3][3][3] <= `dark; G[4][3][3] <= `dark; G[5][3][3] <= `dark;
end
//z = 3 two
if(mode2state2==2'd2) begin
    G[1][0][3] <= mode1GRB; G[6][0][3] <= mode1GRB;
    G[1][1][3] <= mode1GRB; G[6][1][3] <= mode1GRB;
    G[1][2][3] <= mode1GRB; G[6][2][3] <= mode1GRB;
    G[1][3][3] <= mode1GRB; G[6][3][3] <= mode1GRB;
end else begin
    G[1][0][3] <= `dark; G[6][0][3] <= `dark;
    G[1][1][3] <= `dark; G[6][1][3] <= `dark;
    G[1][2][3] <= `dark; G[6][2][3] <= `dark;
    G[1][3][3] <= `dark; G[6][3][3] <= `dark;
end
//z = 3 thr
if(mode2state2==2'd3) begin
    G[0][0][3] <= mode1GRB; G[7][0][3] <= mode1GRB;
    G[0][1][3] <= mode1GRB; G[7][1][3] <= mode1GRB;
    G[0][2][3] <= mode1GRB; G[7][2][3] <= mode1GRB;
    G[0][3][3] <= mode1GRB; G[7][3][3] <= mode1GRB;
end else begin
    G[0][0][3] <= `dark; G[7][0][3] <= `dark;
    G[0][1][3] <= `dark; G[7][1][3] <= `dark;
    G[0][2][3] <= `dark; G[7][2][3] <= `dark;
    G[0][3][3] <= `dark; G[7][3][3] <= `dark;
end

```

而後當音樂樹完全盛放之時，會隨機落下點點星光。在實作之前，我們需要先有一個隨機的 module 可供使用，所以我們上網查找了一下資料，並找到了一些可用的資訊。



Build this LFSR. The reset should reset the LFSR to 1.

根據資料圖，這樣可以實作出一個 5bit 的隨機數值，於是我們便按照著邏輯圖做出一個 module，並且將 module 的輸出值當成隨機值給到上層 module 裡面。

```
module LFSR(
    input clk,
    input rst,
    output reg [4:0] out
);

    wire [4:0] next_out;
    assign next_out = {out[0], out[4], out[0] ^ out[3], out[2], out[1]};

    always @(posedge clk) begin
        out <= (rst ? 5'd1 : next_out);
    end

endmodule
```

有了隨機值之後，我們便可以在最上層的指定範圍內隨機選擇一個點落下星光，而因為這裡產生的隨機值，我們並不能知道現在為哪一格被選到要降下，所以我們在降下的過程中都是以指定範圍內的所有點一起賦值上一格。

```
//z = 2 then
if(mode2state==2'd0 && mode2state==3'd3) begin
    case(rand)
        5'd1: G[0][3][2] <- mode2GRB; 5'd5: G[1][3][2] <- mode2GRB; 5'd9: G[2][3][2] <- mode2GRB; 5'd11: G[5][3][2] <- mode2GRB; 5'd13: G[6][3][2] <- mode2GRB; 5'd17: G[7][3][2] <- mode2GRB;
        5'd2: G[0][2][2] <- mode2GRB; 5'd6: G[1][2][2] <- mode2GRB; 5'd12: G[6][2][2] <- mode2GRB; 5'd14: G[7][2][2] <- mode2GRB;
        5'd3: G[0][1][2] <- mode2GRB; 5'd7: G[1][1][2] <- mode2GRB; 5'd15: G[6][1][2] <- mode2GRB; 5'd19: G[7][1][2] <- mode2GRB;
        5'd4: G[0][0][2] <- mode2GRB; 5'd8: G[1][0][2] <- mode2GRB; 5'd10: G[2][0][2] <- mode2GRB; 5'd12: G[5][0][2] <- mode2GRB; 5'd16: G[6][0][2] <- mode2GRB; 5'd20: G[7][0][2] <- mode2GRB;
        default: G[0][0][2] <- mode2GRB;
    endcase
end
if(time_05) begin
    //z = 2
    G[0][3][2] <- `dark; G[1][3][2] <- `dark; G[2][3][2] <- `dark; G[5][3][2] <- `dark; G[6][3][2] <- `dark; G[7][3][2] <- `dark;
    G[0][2][2] <- `dark; G[1][2][2] <- `dark; G[6][2][2] <- `dark; G[7][2][2] <- `dark;
    G[0][1][2] <- `dark; G[1][1][2] <- `dark; G[6][1][2] <- `dark; G[7][1][2] <- `dark;
    G[0][0][2] <- `dark; G[1][0][2] <- `dark; G[5][0][2] <- `dark; G[6][0][2] <- `dark; G[7][0][2] <- `dark;
    //z = 1
    G[0][3][1] <- G[0][3][2]; G[1][3][1] <- G[1][3][2]; G[2][3][1] <- G[2][3][2]; G[5][3][1] <- G[5][3][2]; G[6][3][1] <- G[6][3][2]; G[7][3][1] <- G[7][3][2];
    G[0][2][1] <- G[0][2][2]; G[1][2][1] <- G[1][2][2];
    G[0][1][1] <- G[0][1][2]; G[1][1][1] <- G[1][1][2];
    G[0][0][1] <- G[0][0][2]; G[1][0][1] <- G[1][0][2]; G[2][0][1] <- G[2][0][2]; G[5][0][1] <- G[5][0][2]; G[6][0][1] <- G[6][0][2]; G[7][0][1] <- G[7][0][2];
    //z = 0
    G[0][3][0] <- G[0][3][1]; G[1][3][0] <- G[1][3][1]; G[2][3][0] <- G[2][3][1]; G[5][3][0] <- G[5][3][1]; G[6][3][0] <- G[6][3][1]; G[7][3][0] <- G[7][3][1];
    G[0][2][0] <- G[0][2][1]; G[1][2][0] <- G[1][2][1];
    G[0][1][0] <- G[0][1][1]; G[1][1][0] <- G[1][1][1];
    G[0][0][0] <- G[0][0][1]; G[1][0][0] <- G[1][0][1]; G[2][0][0] <- G[2][0][1]; G[5][0][0] <- G[5][0][1]; G[6][0][0] <- G[6][0][1]; G[7][0][0] <- G[7][0][1];
end
```

而在此處的 mode2GRB 則為我們為了落下星光的顏色去做隨機，多新增的一個 always block。

```
//mode2GRB
always @(*) begin
    case(randcolor)
        5'd0: mode2GRB = `red;
        5'd1: mode2GRB = `ored;
        5'd2: mode2GRB = `oran;
        5'd3: mode2GRB = `yora;
        5'd4: mode2GRB = `yell;
        5'd5: mode2GRB = `ygre;
        5'd6: mode2GRB = `gree;
        5'd7: mode2GRB = `bgre;
        5'd8: mode2GRB = `blue;
        5'd9: mode2GRB = `bvio;
        5'd10: mode2GRB = `viol;
        5'd11: mode2GRB = `rvio;
        default: mode2GRB = `rvio;
    endcase
end
```

而後為了維持音樂樹的完整性，我們於最中間四格的往外八格皆設定為暗，這樣便可以具有較佳的觀賞性及效果。

```
//z = 2
| G[3][3][2] <= `dark; G[4][3][2] <= `dark;
G[2][2][2] <= `dark; G[2][1][2] <= `dark; G[5][2][2] <= `dark;
| G[3][0][2] <= `dark; G[4][0][2] <= `dark; G[5][1][2] <= `dark;
//z = 1
| G[3][3][1] <= `dark; G[4][3][1] <= `dark;
G[2][2][1] <= `dark; G[2][1][1] <= `dark; G[5][2][1] <= `dark;
| G[3][0][1] <= `dark; G[4][0][1] <= `dark; G[5][1][1] <= `dark;
// z = 0
| G[3][3][0] <= `dark; G[4][3][0] <= `dark;
G[2][2][0] <= `dark; G[2][1][0] <= `dark; G[5][2][0] <= `dark;
| G[3][0][0] <= `dark; G[4][0][0] <= `dark; G[5][1][0] <= `dark;
```

### 四季斜陽八方輝耀模式：

在這個模式裡，我們將實作類似於音樂噴泉的效果，但是會將基底變成斜面呈現，最後的效果會像是斜陽一般，將光芒輝耀到大地之上。

這個模式類似於上一個模式的基礎版，但由於將噴發面改成了斜面，所以空間感需要再重新建立。斜面的建立以  $yz$  平面的 16 個點切割成 1 2 3 4 3 2 1，依序為 1 到 7 的斜面，隨後根據聲音大小及基礎值設定，這七個斜面所能沿著  $x$  軸噴發的最常距離也有所不同。

```
//1
if(voice > 'voice_low) G[0][0][3] <= mode1GRB;
else G[0][0][3] <= `dark;
G[1][0][3] <= `dark; G[2][0][3] <= `dark; G[3][0][3] <= `dark; G[4][0][3] <= `dark; G[5][0][3] <= `dark; G[6][0][3] <= `dark; G[7][0][3] <= `dark;
//2
G[0][0][2] <= mode1GRB;
G[0][1][3] <= mode1GRB;
if(voice > 'voice_low) begin
  G[1][0][2] <= mode1GRB;
  G[1][1][3] <= mode1GRB;
end else begin
  G[1][0][2] <= `dark;
  G[1][1][3] <= `dark;
end
G[2][0][2] <= `dark; G[3][0][2] <= `dark; G[4][0][2] <= `dark; G[5][0][2] <= `dark; G[6][0][2] <= `dark; G[7][0][2] <= `dark;
G[2][1][3] <= `dark; G[3][1][3] <= `dark; G[4][1][3] <= `dark; G[5][1][3] <= `dark; G[6][1][3] <= `dark; G[7][1][3] <= `dark;
//3
G[0][0][1] <= mode1GRB; G[1][0][1] <= mode1GRB;
G[0][1][2] <= mode1GRB; G[1][1][2] <= mode1GRB;
G[0][2][3] <= mode1GRB; G[1][2][3] <= mode1GRB;
if(voice > 'voice_lmid) begin
  G[2][0][1] <= mode1GRB;
  G[2][1][2] <= mode1GRB;
  G[2][2][3] <= mode1GRB;
end else begin
  G[2][0][1] <= `dark;
  G[2][1][2] <= `dark;
  G[2][2][3] <= `dark;
end
G[3][0][1] <= `dark; G[4][0][1] <= `dark; G[5][0][1] <= `dark; G[6][0][1] <= `dark; G[7][0][1] <= `dark;
G[3][1][2] <= `dark; G[4][1][2] <= `dark; G[5][1][2] <= `dark; G[6][1][2] <= `dark; G[7][1][2] <= `dark;
G[3][2][3] <= `dark; G[4][2][3] <= `dark; G[5][2][3] <= `dark; G[6][2][3] <= `dark; G[7][2][3] <= `dark;
```

```

//4
G[0][0][0] <= mode1GRB; G[1][0][0] <= mode1GRB;
G[0][1][1] <= mode1GRB; G[1][1][1] <= mode1GRB;
G[0][2][2] <= mode1GRB; G[1][2][2] <= mode1GRB;
G[0][3][3] <= mode1GRB; G[1][3][3] <= mode1GRB;
if(voice > `voice_lmid) begin
    G[2][0][0] <= mode1GRB;
    G[2][1][1] <= mode1GRB;
    G[2][2][2] <= mode1GRB;
    G[2][3][3] <= mode1GRB;
end else begin
    G[2][0][0] <= `dark;
    G[2][1][1] <= `dark;
    G[2][2][2] <= `dark;
    G[2][3][3] <= `dark;
end
if(voice > `voice_mid) begin
    G[3][0][0] <= mode1GRB;
    G[3][1][1] <= mode1GRB;
    G[3][2][2] <= mode1GRB;
    G[3][3][3] <= mode1GRB;
end else begin
    G[3][0][0] <= `dark;
    G[3][1][1] <= `dark;
    G[3][2][2] <= `dark;
    G[3][3][3] <= `dark;
end
G[4][0][0] <= `dark; G[5][0][0] <= `dark; G[6][0][0] <= `dark; G[7][0][0] <= `dark;
G[4][1][1] <= `dark; G[5][1][1] <= `dark; G[6][1][1] <= `dark; G[7][1][1] <= `dark;
G[4][2][2] <= `dark; G[5][2][2] <= `dark; G[6][2][2] <= `dark; G[7][2][2] <= `dark;
G[4][3][3] <= `dark; G[5][3][3] <= `dark; G[6][3][3] <= `dark; G[7][3][3] <= `dark;

//5
G[0][1][0] <= mode1GRB; G[1][1][0] <= mode1GRB;
G[0][2][1] <= mode1GRB; G[1][2][1] <= mode1GRB;
G[0][3][2] <= mode1GRB; G[1][3][2] <= mode1GRB;
if(voice > `voice_lmid) begin
    G[2][1][0] <= mode1GRB;
    G[2][2][1] <= mode1GRB;
    G[2][3][2] <= mode1GRB;
end else begin
    G[2][1][0] <= `dark;
    G[2][2][1] <= `dark;
    G[2][3][2] <= `dark;
end
if(voice > `voice_mid) begin
    G[3][1][0] <= mode1GRB;
    G[3][2][1] <= mode1GRB;
    G[3][3][2] <= mode1GRB;
end else begin
    G[3][1][0] <= `dark;
    G[3][2][1] <= `dark;
    G[3][3][2] <= `dark;
end
if(voice > `voice_hmid) begin
    G[4][1][0] <= mode1GRB;
    G[4][2][1] <= mode1GRB;
    G[4][3][2] <= mode1GRB;
end else begin
    G[4][1][0] <= `dark;
    G[4][2][1] <= `dark;
    G[4][3][2] <= `dark;
end
G[5][1][0] <= `dark; G[6][1][0] <= `dark; G[7][1][0] <= `dark;
G[5][2][1] <= `dark; G[6][2][1] <= `dark; G[7][2][1] <= `dark;
G[5][3][2] <= `dark; G[6][3][2] <= `dark; G[7][3][2] <= `dark;

```

```

//6
G[0][2][0] <= mode1GRB; G[1][2][0] <= mode1GRB; G[2][2][0] <= mode1GRB;
G[0][3][1] <= mode1GRB; G[1][3][1] <= mode1GRB; G[2][3][1] <= mode1GRB;
if(voice > `voice_lmid) begin
    G[3][2][0] <= mode1GRB;
    G[3][3][1] <= mode1GRB;
end else begin
    G[3][2][0] <= `dark;
    G[3][3][1] <= `dark;
end
if(voice > `voice_mid) begin
    G[4][2][0] <= mode1GRB;
    G[4][3][1] <= mode1GRB;
end else begin
    G[4][2][0] <= `dark;
    G[4][3][1] <= `dark;
end
if(voice > `voice_hmid) begin
    G[5][2][0] <= mode1GRB;
    G[5][3][1] <= mode1GRB;
end else begin
    G[5][2][0] <= `dark;
    G[5][3][1] <= `dark;
end
G[6][2][0] <= `dark; G[7][2][0] <= `dark;
G[6][3][1] <= `dark; G[7][3][1] <= `dark;
//7
G[0][3][0] <= mode1GRB; G[1][3][0] <= mode1GRB; G[2][3][0] <= mode1GRB; G[3][3][0] <= mode1GRB;
if(voice > `voice_lmid) begin
    G[4][3][0] <= mode1GRB;
end else begin
    G[4][3][0] <= `dark;
end
if(voice > `voice_mid) begin
    G[5][3][0] <= mode1GRB;
end else begin
    G[5][3][0] <= `dark;
end
if(voice > `voice_hmid) begin
    G[6][3][0] <= mode1GRB;
end else begin
    G[6][3][0] <= `dark;
end
if(voice > `voice_high) begin
    G[7][3][0] <= mode1GRB;
end else begin
    G[7][3][0] <= `dark;
end

```

在設計這個模式的時候，我們並不是像是音樂樹那樣利用 finite state machine 做 state 的變換，從而讓 LED 燈有一個移動的感覺，而是直接根據聲音大小判斷便去賦值，這麼做也是為了實驗我們的想法能不能實現，所以算是一個還在實驗的模式，而在實際呈現出來的效果也不確實不如我們想像中的那樣，會隨著音樂大小往右噴發及往左復位的效果，而是會直接閃爍到對應聲音大小的位置，但其閃爍的效果透過七彩的斜面依然是呈現一個漂亮的結果。

### 六道輪迴聲不息模式：

在這個模式，我們將實作以一個平面沿著一條軸去做同時移動，並且平面移動為隨機產生，便能呈現立體三維變換六向的形式效果。

在設計平面前，我們先利用隨機值以及 finite state machine 去決定我們將以哪一個平面去做移動變換以及決定移動的方向為何。

```

5'd0: begin //initial state, all dark
    case(rand%6)
        3'd0: begin
            next_mode3state = 5'd1;
            next_mode3dir = 2'd2;
        end
        3'd1: begin
            next_mode3state = 5'd8;
            next_mode3dir = 2'd0;
        end
        3'd2: begin
            next_mode3state = 5'd9;
            next_mode3dir = 2'd2;
        end
        3'd3: begin
            next_mode3state = 5'd12;
            next_mode3dir = 2'd0;
        end
        3'd4: begin
            next_mode3state = 5'd13;
            next_mode3dir = 2'd2;
        end
        3'd5: begin
            next_mode3state = 5'd16;
            next_mode3dir = 2'd0;
        end
        default: begin
            next_mode3state = 5'd1;
            next_mode3dir = 2'd2;
        end
    endcase
end

```

因為我們是三個平面沿著三個軸移動，所以有 6 個方向 6 種狀態，因此我們讓 rand 值%6 以保證我們的值可以對應到每一個 state。而後我們便在下面利用 finite state machine 去定義為何者平面以及其移動的方向。從 1~8 為 yz 平面沿著 x 軸左右移動，9~12 為 xz 平面沿著 y 軸前後移動，13~16 為 xy 平面沿著 z 軸上下移動。

```

5'd1: begin //x0 time_01
    if(time_01) next_mode3state = (mode3dir==2'd0 ? 5'd0 : mode3state + 1'b1);
    else next_mode3state = mode3state;
end
5'd2: next_mode3state = (time_01 ? mode3state + mode3dir - 1'b1 : mode3state); //x1
5'd3: next_mode3state = (time_01 ? mode3state + mode3dir - 1'b1 : mode3state); //x2
5'd4: next_mode3state = (time_01 ? mode3state + mode3dir - 1'b1 : mode3state); //x3
5'd5: next_mode3state = (time_01 ? mode3state + mode3dir - 1'b1 : mode3state); //x4
5'd6: next_mode3state = (time_01 ? mode3state + mode3dir - 1'b1 : mode3state); //x5
5'd7: next_mode3state = (time_01 ? mode3state + mode3dir - 1'b1 : mode3state); //x6
5'd8: begin //x7
    if(time_01) next_mode3state = (mode3dir==2'd2 ? 5'd0 : mode3state - 1'b1);
    else next_mode3state = mode3state;
end

5'd9: begin //y0
    if(time_01) next_mode3state = (mode3dir==2'd0 ? 5'd0 : mode3state + 1'b1);
    else next_mode3state = mode3state;
end
5'd10: next_mode3state = (time_01 ? mode3state + mode3dir - 1'b1 : mode3state); //y1
5'd11: next_mode3state = (time_01 ? mode3state + mode3dir - 1'b1 : mode3state); //y2
5'd12: begin //y3
    if(time_01) next_mode3state = (mode3dir==2'd2 ? 5'd0 : mode3state - 1'b1);
    else next_mode3state = mode3state;
end

```

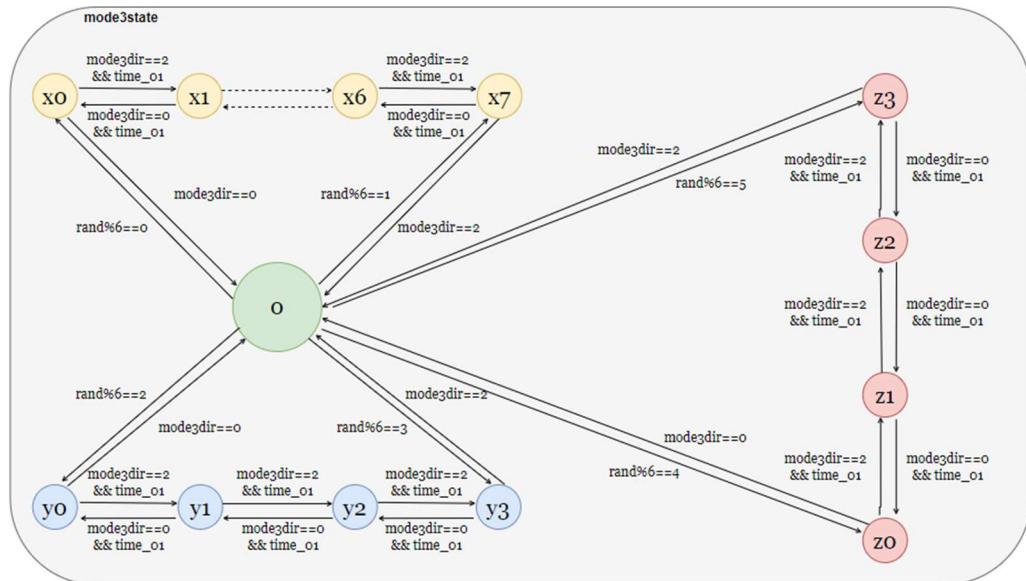
```

5'd13: begin //z0
    if(time_01) next_mode3state = (mode3dir==2'd0 ? 5'd0 : mode3state + 1'b1);
    else next_mode3state = mode3state;
end
5'd14: next_mode3state = (time_01 ? mode3state + mode3dir - 1'b1 : mode3state); //z1
5'd15: next_mode3state = (time_01 ? mode3state + mode3dir - 1'b1 : mode3state); //z2
5'd16: begin //z3
    if(time_01) next_mode3state = (mode3dir==2'd2 ? 5'd0 : mode3state - 1'b1);
    else next_mode3state = mode3state;
end

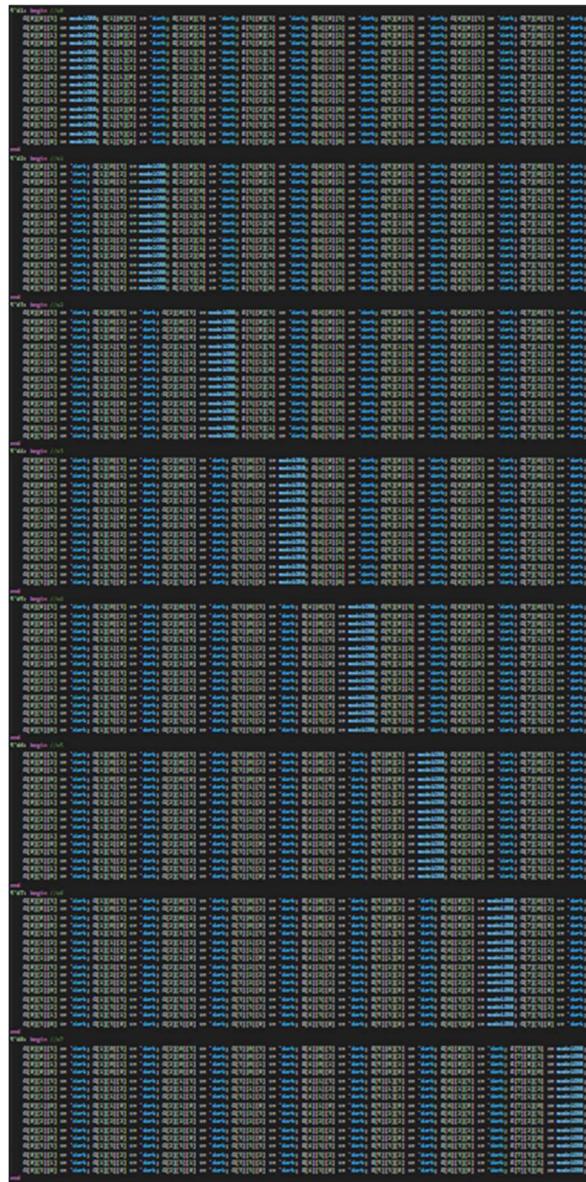
```

在此處我們將每個方向走到最旁邊那一面的時候，讓其方向也就是 dir 為 0 或為 2 是因為我們在設值的時候會多減一個 1，使其變成-1 或是 1，這樣便可以很好的知道下一個方向要為何。

mode3state



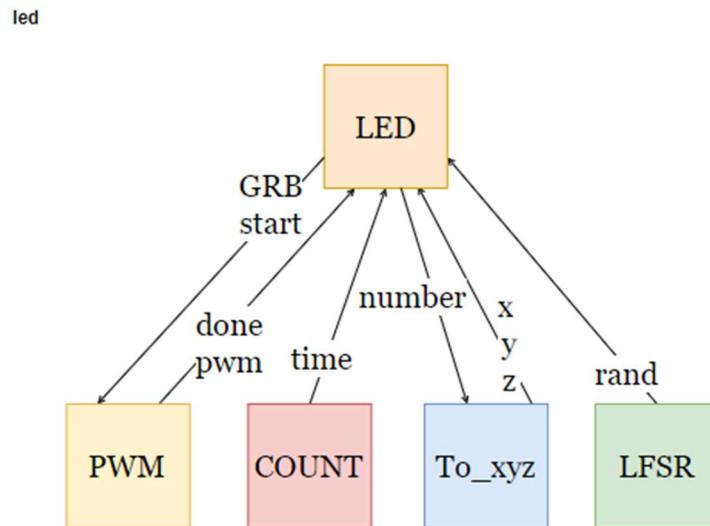
定義好 state 之後，接下來便是為每一個 state 設計該 state 的燈光效果。其實作方法類似前面，也是利用 LED Cube 的四維數組進行 code 排版，將其轉成平面圖呈現在我們眼前，隨後一個一個燈號去做設值，而因為這種排版在連續 coding 的過程中也可以顯示出一種動態的效果，更方便我們去維護。以 yz 平面沿著 x 軸左右移動為例，其呈現在 code 的效果如下圖。



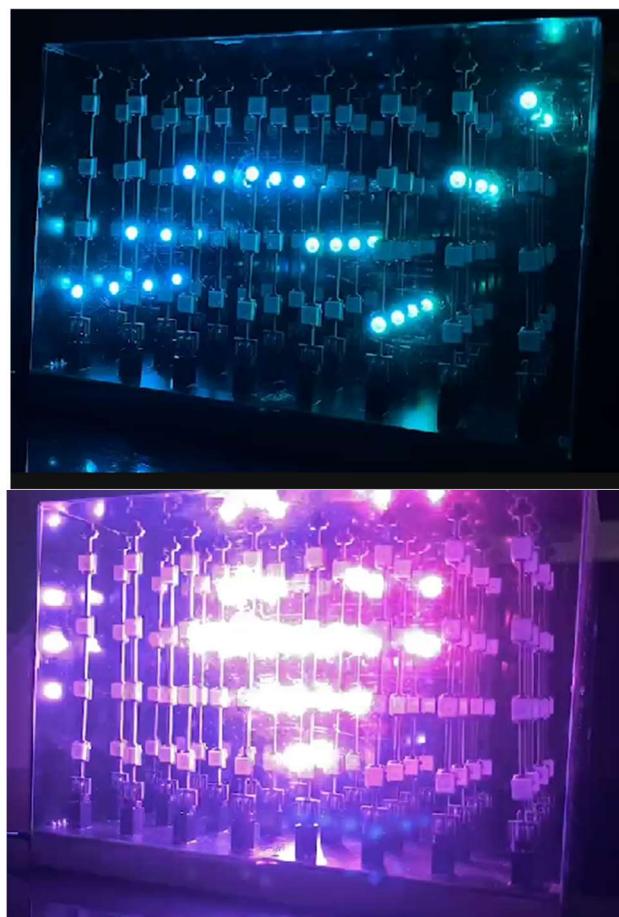
### 自定義歌曲適配模式：

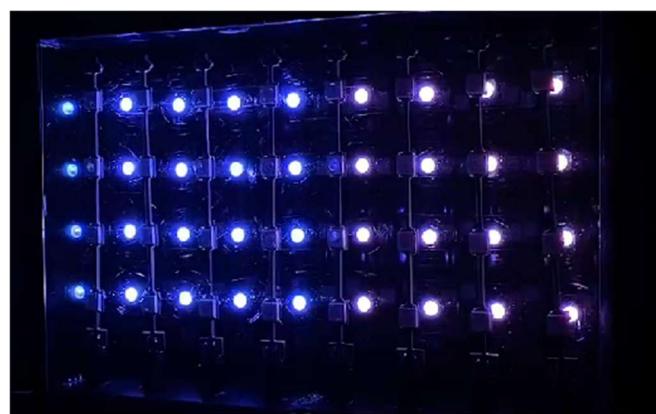
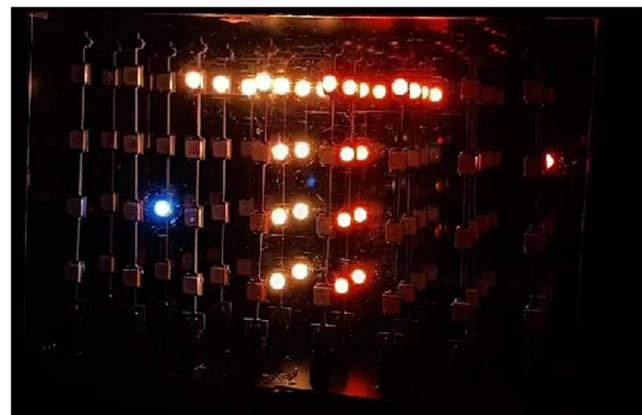
在這個模式裡，我們實作的是使用者可以依據不同音樂去呈現不同的燈光及變換效果。在此處可以自定義延遲時間，決定目前呈現模式要持續多久，並且也可以自定義要放入哪種音樂模式去搭配音樂做呈現。

```
case(mode4state)
 4'd0: begin
   mode = 3'd7;
   next_mode4state = (time_9 ? mode4state+1'b1 : mode4state);
 end
 4'd1: begin
   mode = 3'd6;
   next_mode4state = (time_66 ? mode4state+1'b1 : mode4state);
 end
 4'd2: begin
   mode = 3'd0;
   next_mode4state = (time_83 ? mode4state+1'b1 : mode4state);
 end
 4'd3: begin
   mode = 3'd2;
   next_mode4state = (time_122 ? mode4state+1'b1 : mode4state);
 end
 4'd4: begin
   mode = 3'd5;
   next_mode4state = (time_140 ? mode4state+1'b1 : mode4state);
 end
 4'd5: begin
   mode = 3'd7;
   next_mode4state = (time_159 ? mode4state+1'b1 : mode4state);
 end
 4'd6: begin
   mode = 3'd2;
   next_mode4state = (time_179 ? mode4state+1'b1 : mode4state);
 end
 4'd7: begin
   mode = 3'd1;
   next_mode4state = (time_196 ? mode4state+1'b1 : mode4state);
 end
 4'd8: begin
   mode = 3'd3;
   next_mode4state = (time_234 ? mode4state+1'b1 : mode4state);
 end
 4'd9: begin
   mode = 3'd7;
   next_mode4state = (mode7state==7'd127 ? mode4state+1'b1 : mode4state);
 end
 4'd10: begin
   mode = 3'd0;
   next_mode4state = mode4state;
 end
 default: begin
   mode = 3'd7;
   next_mode4state = 4'd0;
 end
```



## VI. Project Demo 呈現





## VII. 結語

歷時一個多月 project coding 和 testing，我們總算將一個完成度頗高的 Music Light 給實作出來，看著我們自己做出來的東西，隨著音樂的節奏及大小聲，實際的在眼前舞動著光芒，感覺這段時間以來的辛苦勞累，總算有了回報。即使這段過程中遭遇到種種的困難，從剛開始要利用 Sound Sensor 將聲音大小抓進來，經歷了一系列艱澀難懂的模數轉換及瀏覽資訊量過大的資料網站，再到需要用 LED Cube 呈現各種圖案轉換，利用四維數組以及不斷的計算 3D 圖案及賦值，最後在測試 LED Cube 時間歇出現的 bug 以及燈光效果不符預期，這些一個又一個的困難我們都一點一滴的解決，並且也一步一步的著實在向前的。直到 demo 的時候，看著同學們看到我們的作品驚歎不已的表情，我想，「成就感」，便是不斷付出的最好回饋。

回首這一學期的邏輯設計實驗課程，從一開始簡單的運用 D flip-flop 產生 simulation 觀察波形圖，再到利用 FPGA 設計 Finite state machine 並實作出 speaker、VGA、Keyboard 及車車等，最後依靠自己的力量將一個外人看起來已經頗厲害的作品完成，付出和努力得到的是無價的知識，是滿足的成就，更是寶貴的回憶。

如今在此寫下這份 report，意味著這學期的課程終將告一段落，但結束永遠不會是終點，而是另一個新的開始，這好比莫比烏斯環，在求學的路上是永無止境的，也期待著短短一學期的緣分並不會在此畫下句點，在往後的路程裡，希望還能見到各位助教以及教授。

最後，請允許我們僅以這份 report 的最後來表達我們的謝意——感謝教授這一學期以來的悉心教導，也感謝助教不厭其煩的解答我們的問題及課程上的幫助。

至此有緣終再相會  
祝新年快樂萬事如意

## VIII. 參考資料

XADC Wizzard v3.3 LogiCORE IP Product Guide

7 Series FPGAs and Zynq-7000 SoC XADC Dual 12-bit 1 MSPS Analog-to-Digital Converter

<https://ahmedmohamed45am.wixsite.com/fpgagate/single-post/2018/03/13/xilinx-xadc-part-1-analog-input>

<https://blog.csdn.net/weichen001122/article/details/105405017>

<https://digilent.com/reference/learn/programmable-logic/tutorials/basys-3-xadc/start>

[https://blog.csdn.net/weixin\\_41957211/article/details/106391890](https://blog.csdn.net/weixin_41957211/article/details/106391890)

[https://blog.csdn.net/as480133937/article/details/103439546?ops\\_request\\_mis\\_c=%257B%2522request%255Fid%2522%253A%2522164104143116781683949492%2522%252C%2522scm%2522%253A%252220140713.130102334.%2522%257D&request\\_id=164104143116781683949492&biz\\_id=0&utm\\_medium=distribute.pc\\_search\\_result.none-task-blog-2~all~top\\_positive~default-1-103439546.pc\\_search\\_em\\_sort&utm\\_term=pwm&spm=1018.2226.3001.4187](https://blog.csdn.net/as480133937/article/details/103439546?ops_request_mis_c=%257B%2522request%255Fid%2522%253A%2522164104143116781683949492%2522%252C%2522scm%2522%253A%252220140713.130102334.%2522%257D&request_id=164104143116781683949492&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-103439546.pc_search_em_sort&utm_term=pwm&spm=1018.2226.3001.4187)

<https://www.instructables.com/Led-Cube-8x8x8/>