Final Project Report: FPGA Musician

Team 11

一、介紹

這是一個針對熟悉音樂的人來設計的工具，其中包含調音器、節拍器、編曲器三種功能。

其中調音器和節拍器，在學習音樂的過程中都是不可或缺的。本工具實現了市面上電子節拍器、電子調音器的基本功能，也做得非常相似。

編曲器則是針對編曲、改編的需求來製作。據說奧地利作曲家舒伯特（Franz Seraphicus Peter Schubert）曾經在餐廳裡得到作曲的靈感，就即時在菜單背面完成了一首曲子，可見身邊隨時有寫曲的工具是非常重要的。如果寫曲過程中能夠隨時確認聽起來的效果那當然更好——這個編曲器就是結合了這兩個功能。

二、動機

在決定這個主題時，有兩個最主要的考量：

1. 合適性：

這是在 FPGA 上用 verilog 實作的專案，應該要選擇記憶體空間少、計算單純（僅有大量加減法，沒有三角函數、指數等等）的功能較適合。

2. 實用性：

即便是簡單的功能，我們也希望能做出確實有用的工具。

再加上我們有音樂相關的背景，因此決定製作和音樂有關的這三個工具。即使是在 FPGA 上，只有數百行程式碼的作品，也已經符合大部分市面上的需求。

三、功能說明

用 F1, F2, F3, F4 按鍵分別切換至調音器、節拍器、編曲器和靜音。FPGA 板上的 center button 是 reset，務必在使用前按下。

1. 調音器：

會持續播放一個音，可用+_按鍵上下調整半個音程，也可長按。可以播放的音有中央八度和上下各兩個八度，共五個八度，每個八度內有 C, C♯, D, D♯, E, F, F♯, G, G♯, A, A♯, B 共 12 個音，總共是 60 個音。

另外也可以直接指定特定的音高。用字母鍵盤上的 1~5 分別指定五個八度音域，英文 QWERASDFZXCV 按鍵分別對應 C, C♯, D, D♯, E, F, F♯, G, G♯, A, A♯, B，~按鍵是靜音。編曲器也使用同樣的按鍵設定。

2. 節拍器：

會依照指定的 BPM（beat per minute，是樂譜上最泛用的速度單位），用嗶聲來打節拍。可用+_按鍵上下調整一個 BPM，也可長按。

用字母鍵盤上的 1~3 可以設定拍號。拍號分別是 2/4 拍、3/4 拍、4/4 拍，對應 1、2、3。在指定拍號的第一拍上，嗶聲會高八度。

七段顯示器上會顯示 BPM。

3. 編曲器：

可以編入 128 個音符，並從任何位置播放試聽。播放時的速度和節拍器中設定的速度相同。

用{ }按鍵可以切換要修改的音符，七段顯示器上會顯示目前修改的是第幾個音符。用調音器裡的操作可以更改音符的音高，靜音按鍵（~）就是休止符。右側數字鍵盤的 1~8 可以設定音符的長度（拍值），分別是

1：附點二分音符（3 拍）
2：二分音符 2（2 拍）
3：附點四分音符（1.5 拍）
4：四分音符（1 拍）
5：八分音符（0.5 拍）
6：十六分音符（0.25 拍）
7：三連音（1/3 拍）
8：三連音（2/3 拍）

編曲過程中若有任何更動，包括更改音高、拍值、音符位置，都會重新播放在當前位置上的音符作為確認。

Enter 按鍵可以從當前位置開始播放或是停止播放。P 按鍵可以回到第一個音符。沒有編入音符的位置預設會是最低音的 C，四分音符。

在播放音樂的過程中，只有 Enter 鍵是有作用的。在重新按下 Enter 鍵之前都不會回到編曲模式。

四、詳細設計

以下分別敘述每個 module 的設計，較簡單的小 module 就只描述功能。

在此先說明，一個音符的音高（頻率）在這些 module 之間有三種表示方法：

1. One-hot {5-bit} * {13-bit}（也就是鍵盤輸入的原始訊號）
2. Binary {000, 001, ... 101} * {0000, 0001, ... 1100}
3. 真實頻率

在接下來的說明中會提到。

segment: 將輸入的 4-bit 數字轉成一個七段顯示器上的數字。

mod10: 將輸入的數字除以 10 的商和餘輸出，用連續減法，因此需要數個 cycle。

DisplayDigit: 將輸入的 13-bit 數字轉成一個七段顯示器上的四位數字（十進位）並輸出 FPGA 使用的 seg 和 an 訊號。需要使用 segment 和 mod10 這兩個 module。

debounce, onepulse: 用來處理鍵盤和 FPGA 上按鍵的訊號。

declock: 輸入一個 clock 訊號和一個數字 n，輸出除頻 2^(n+1)的 clock。

KeyboardDecoder: 課堂提供的 code，用來轉換鍵盤的 PS2 訊號。

Top:

有四種 state，會依據輸入鍵盤的 F1~F4 切換。有三個 state 分別代表使用三種功能的其中一種，一個 state 會取消所有輸出。

在 Top module 會將鍵盤輸入處理（KeyboardDecoder）後輸進三種功能 module，輸入時會加上一個當前模式的遮罩：modeX_on，也就是當 Top 在第 X 個模式時，這個 wire 的值才會是 1，其他模式時都會是 0。舉例來說，調音器的其中一個輸入是 13-bit 的 freq_in，另外有一個 mode1_on 在調音器模式才是 1，其他模式都是 0。那麼調音器的輸入就會是 freq_in & {12{mode1_on}}，這麼一來就能確保在其他模式下的輸入不會干擾到調音器的狀態。

而三種功能的輸出也會在 Top module 中整合，並輸出到 speaker。用一個簡單的 MUX，依據當前 state 來決定要送出來自哪一個 module 的輸出。

另外，Top module 也先準備好所有 module 需要的不同頻率的 clock，再送到各個 module。

speaker:

這個 module 幾乎是用課堂上提供的 audio sample code 來改編的。功能就是將輸入用 binary 表示的頻率轉換成實際頻率後，從 FPGA 原始的 clock 頻率來計算出需要的頻率，送進音效卡中處理。

雖然其中有設定 duty cycle 的功能，但在本作品中一律使用一半的 duty cycle。

Tuner:

這個 module 實際上是將 one-hot 表示的原始鍵盤訊號轉換成 binary 表示的 decoder，而 Top module 會再將 binary 訊號交給 speaker 輸出。

metronome:

這個 module 主要透過兩個 counter 來計算兩個項目：第一是兩個拍子之間需要隔多久，第二是每幾個拍子要有一個高音。在第一個 counter 時，設定只在計時的開頭一小段時間播放聲音，就能達到短促嗶聲的效果。輸入的 bpm 經過一些運算之後就能得到第一個 counter 應有的值，而第二個的值則是輸入的拍號直接決定。

Composer:

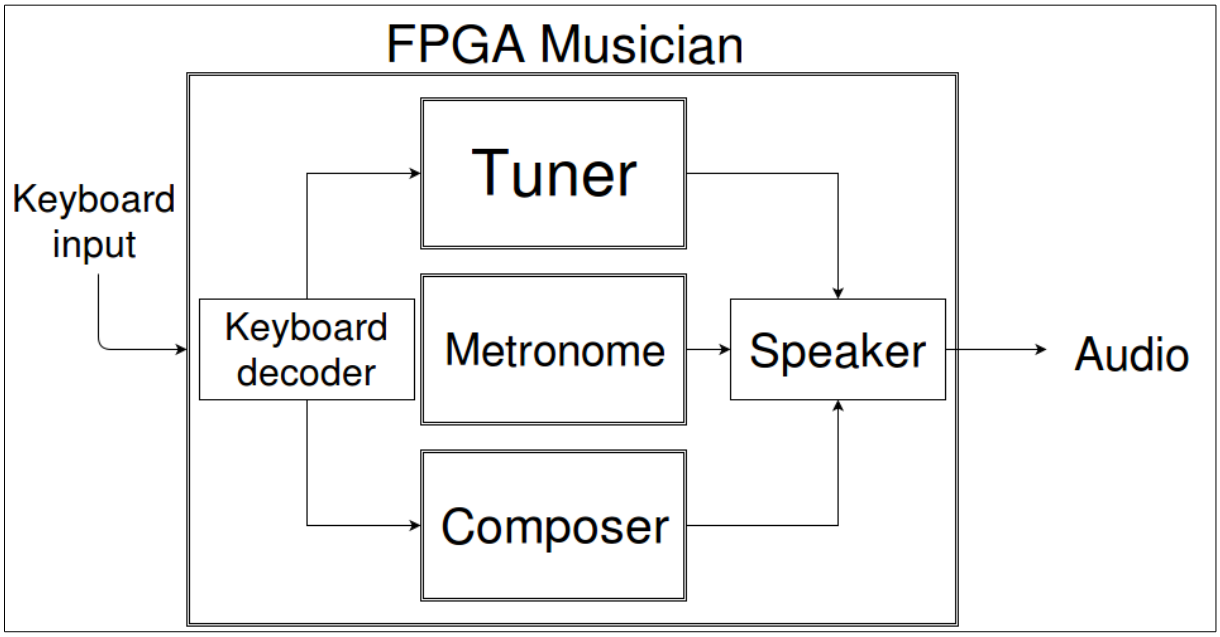
這個 module 最核心的部份就是三個 RAM 的讀寫。這三個 RAM 是 Lab 作業中練習過的虛擬 RAM，用 verilog 二維陣列的方式表達，分別用來存取 binary 表示的頻率還有 one-hot 表示的拍值。另外有一個 pos 記錄當前編輯的音符位置。

決定是讀或是寫的方式是透過一個 counter，當 counter 還在計時的時候，就是還在讀出訊息（也就是播放音符）。當 counter 固定在上限位置的時候就可以繼續寫入。而這個上限取決於不同音符在拍值記憶體中有多長的拍值。換句話說，設定 counter 為 0 就等於開始讀出 RAM 裡的資料。

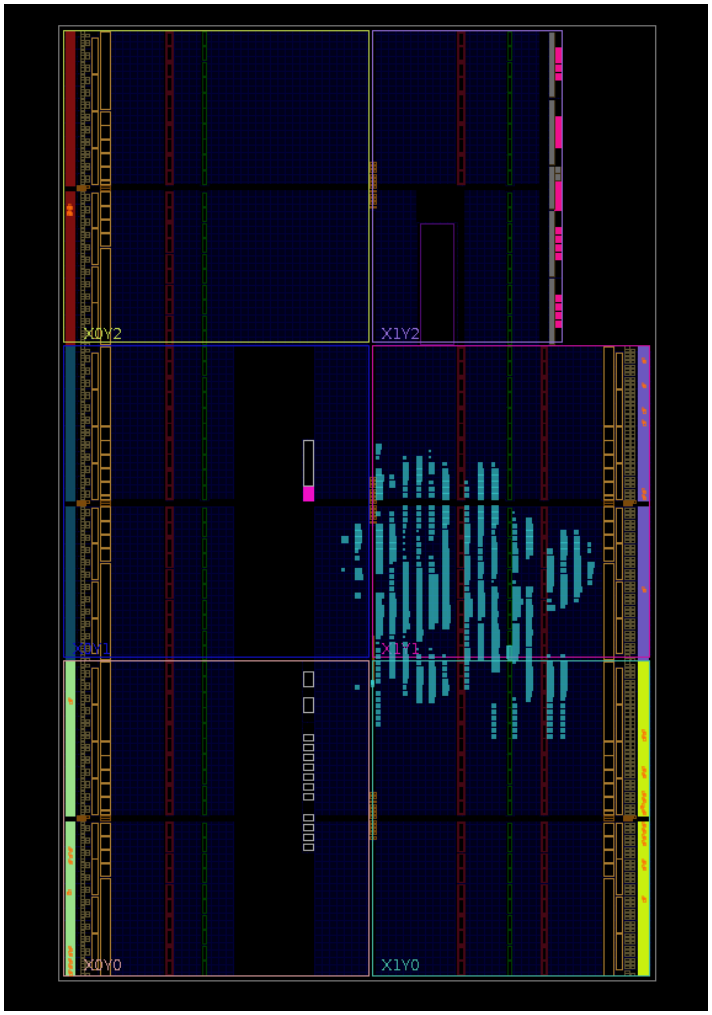
在這裡還設定了兩個 state，有不同的讀寫模式。

編曲模式時，會偵測鍵盤上有沒有任何輸入。一旦有輸入訊號，就做出相應的修改（寫入 RAM 或移動 pos），最後設定 counter 為 0，播放出 pos 位置的音符讓使用者確認。這一系列動作是瞬間完成的。

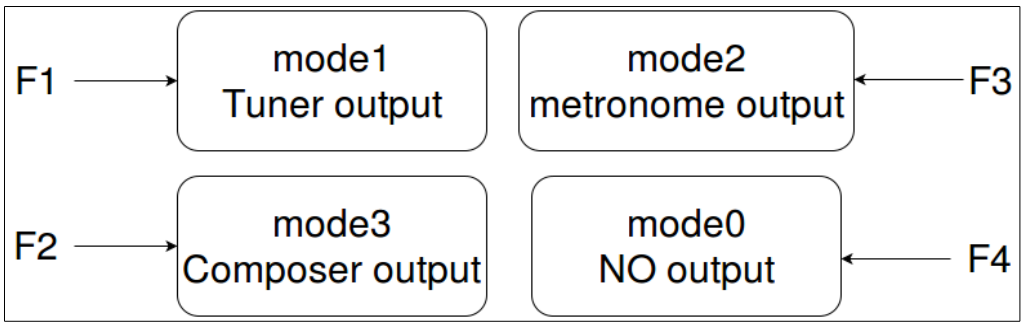
播放音樂模式比較簡單，當 counter 計時到上限後就會自動將 pos 增加，並將 counter 歸零，就能有自動向前播放的效果。



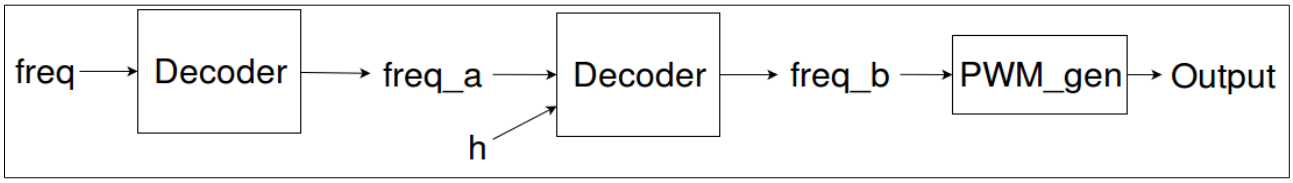
粗略的設計



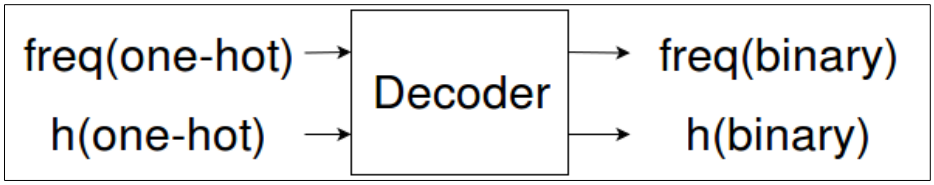
Vivado 中的配置



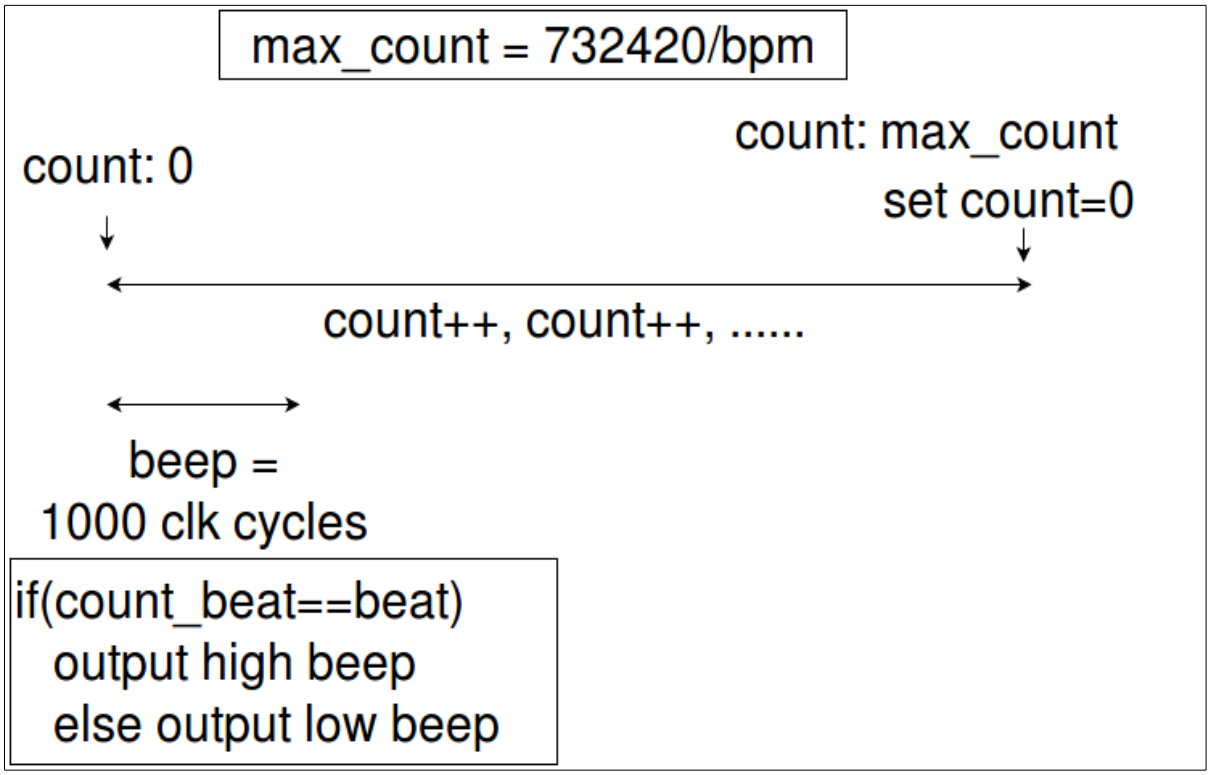
Top module 的 state transition diagram



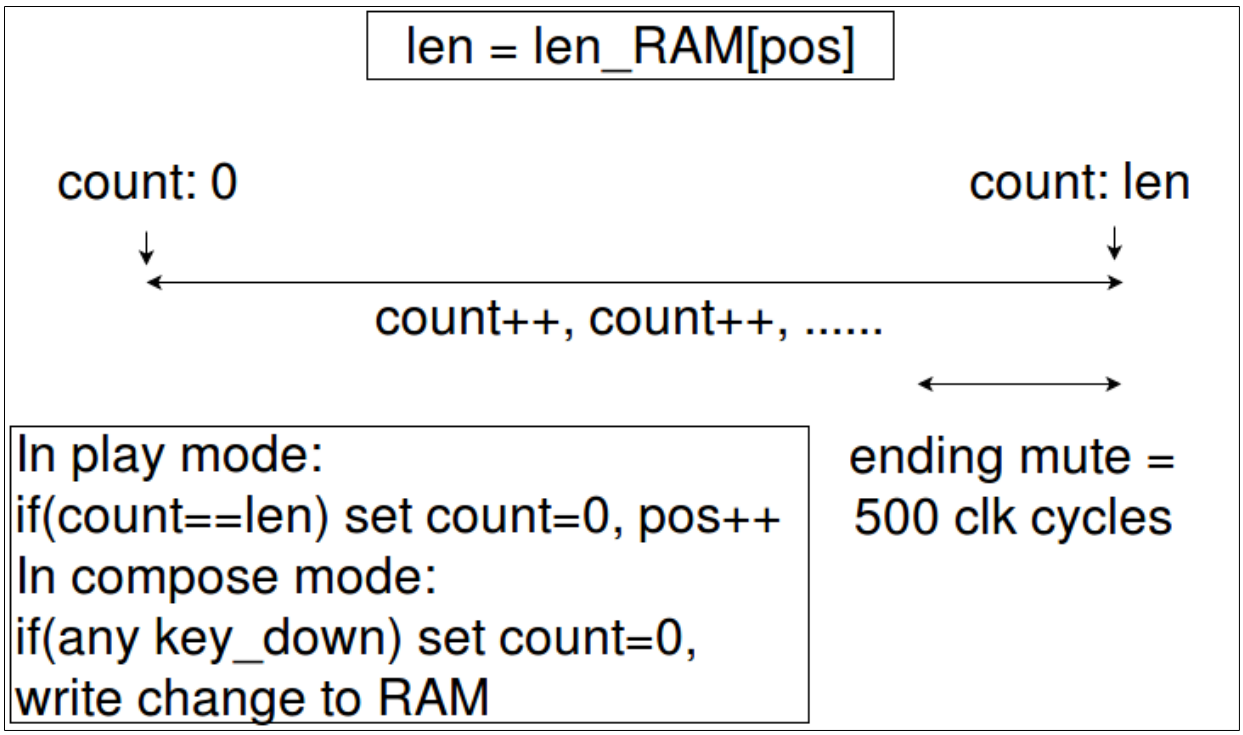
speaker 的流程（freq 和 h 是 binary 形式）



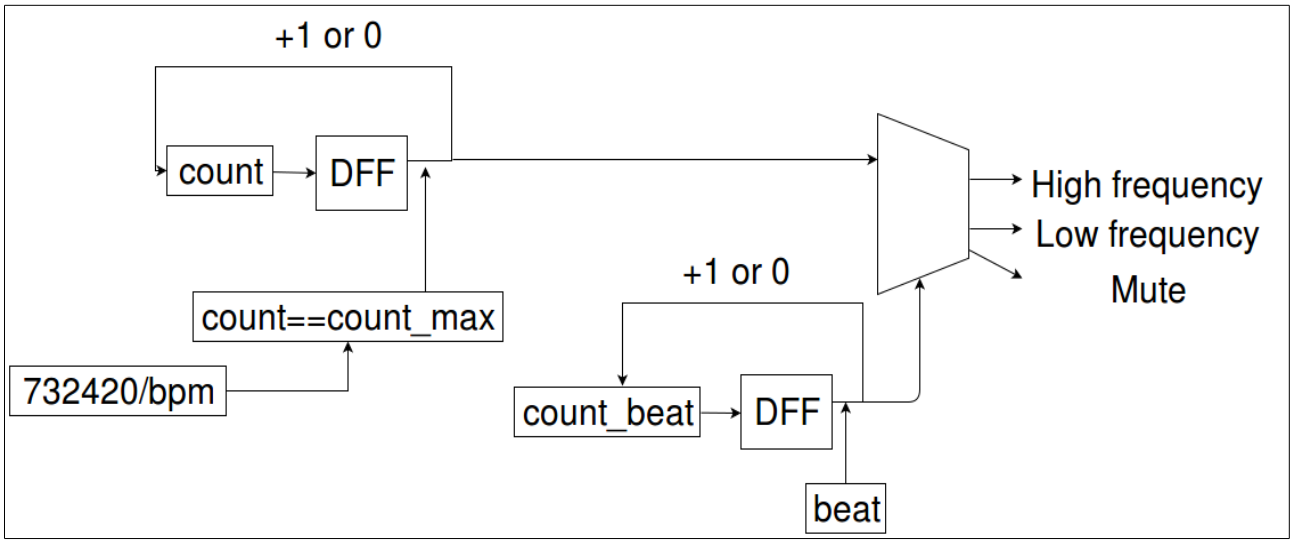
Tuner 調音器的流程



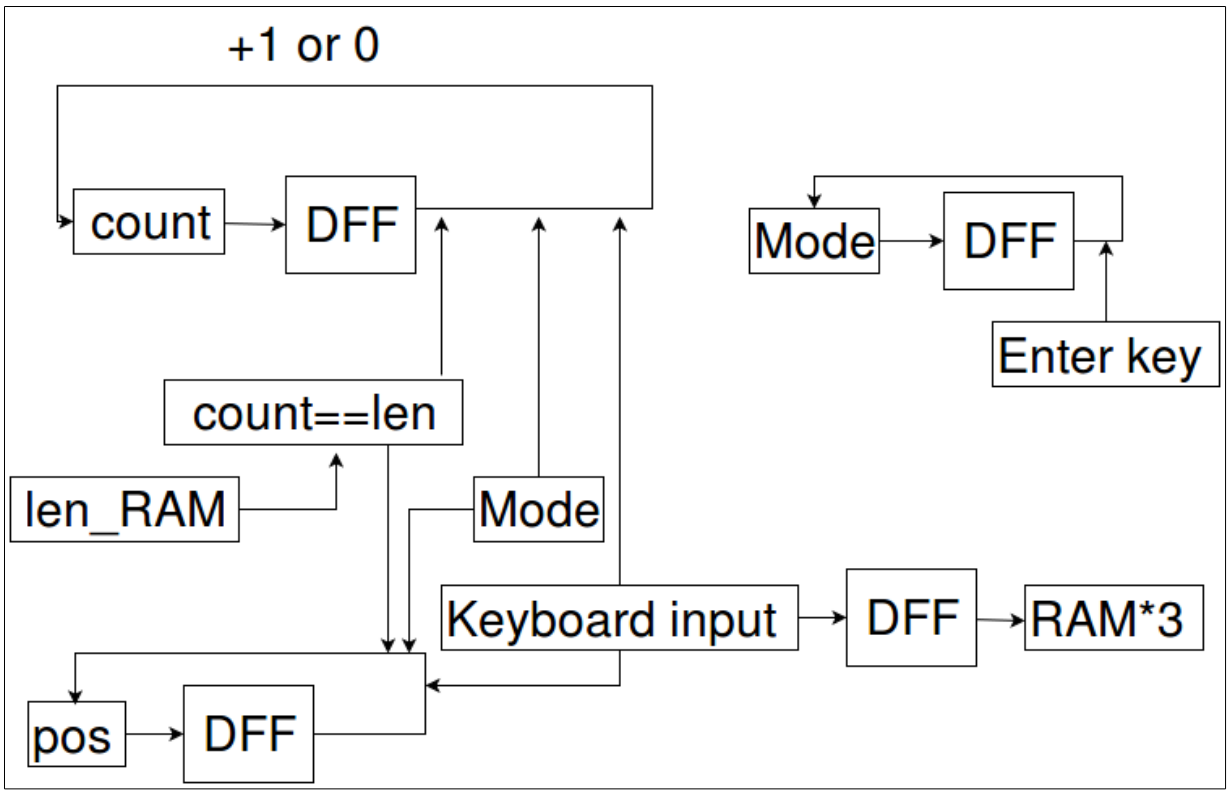
metronome 節拍器的圖解



編曲器的圖解



節拍器的架構



編曲器的架構

五、設計過程與困難

最初就決定要分成三個獨立的作品，再由一個 top module 整合輸入輸出。調音器和節拍器都很快速的做出來，但是在開始製作編曲器的時候一度因為構造有些龐大，不知從何開始。

思考後發現，可以在編曲器中放入一個調音器來輸出聲音，這樣一來編曲器的功能就簡單許多。

其中比較明顯的困難有幾個：

1. clock

不同 module 需要不同的 clock，經過多次測試之後才能決定。其中調音器的 clock 經過 2^23 的除頻，是為了在長按鍵盤時能緩慢改變音程，但缺點是短按按鍵時可能會沒有反應（沒有碰到 positive edge）。雖然知道有其他方法可以改用較小的除頻器（例如設定一個 counter，在倒數完之前不處理下一個輸入），但因為這不是很嚴重的問題就不特別費時處理。

而放在編曲器中的調音器 module 就使用較快的 clock，因此可以正常的即時處理鍵盤輸入。

2. dummy frequency

在課堂提供的範例中，靜音所使用的頻率是 20,000Hz。但是在 speaker 中會按照音高往左右 shift 最多 2 bits，這樣就可能到人耳聽得到的音域內。一開始

我將他改成 0Hz，以為應該沒有問題才對，卻發現音響會發出奇怪的爆裂聲。後來發現這樣會出現除以 0 的狀況，不知道編譯過程是怎麼處理的。最後改成 2,000,000Hz 之後才解決這個問題（不論往左或右 shift 2 bits 都不會溢位，最低也是 20,000Hz 是聽不到的）。

3. 設計邏輯

在設計方式中提到，是把 Tuner 當成 one-hot 到 binary 的 decoder，speaker 當成 binary 到實際頻率的 decoder。會有這樣的設計，是因為最初構想中 one-hot 比較符合鍵盤輸入，而 binary 可以進行某些運算，例如轉調等等。但實際成品中，binary 這部份是沒有必要的。雖然想移除這部份好讓程式更精簡，但需要的時間太多、不確定性也太高，因此就保留這部份，也當作預留其他設計的彈性。這是最初設計時遠見不足的一大敗筆，也是很寶貴的經驗。

六、心得與總結

過去練習的作業在這個 project 中有很大的幫助，包括七段顯示器、decoder 的實做這些基礎練習，還有像是設計 ping pong counter 時對 counter 的運用和模擬 RAM 的讀寫都非常重要。

另外也需要一些巧思，例如在將調音器定位成一個播放聲音用的 decoder 安插在編曲器中。如果沒有想到這一步而是在編曲器中又實做一個，想來會花掉成倍的時間。

可惜的是，因為缺乏組織較大規模程式的經驗，總是只想到目前著眼的小部份該怎麼設計，就造成前文提到的多餘的轉換步驟。可見在設計整體架構時應該用什麼編碼讓 module 之間溝通也是必須提前考量的。但也因為如此，這個 project 能有更大的擴充空間。

最後一提，在設計這樣大型的專案時，程式碼的分區和檔案分割真的非常重要。經過良好的分區後，在後續 debug 過程中，若確定 bug 的原因，需要修改的程式碼幾乎都在十行的範圍之內，減少許多翻找程式碼的時間。