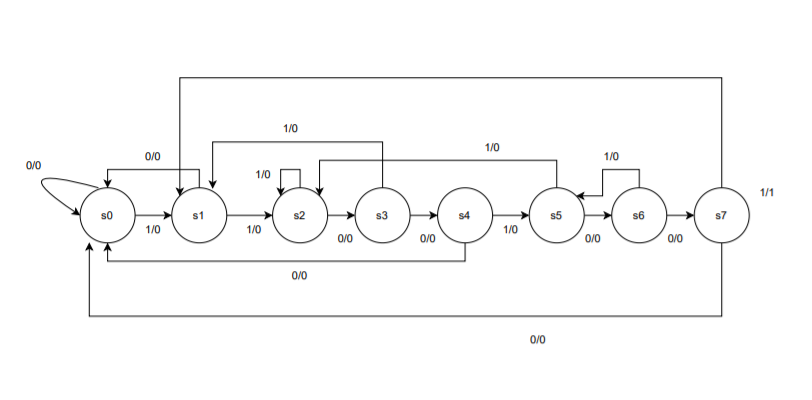
Lab 5 Team 20 Report

109062222 徐嘉徽

109062119 李佳栩

Advance\_1:

這是我們的state\_diagram。

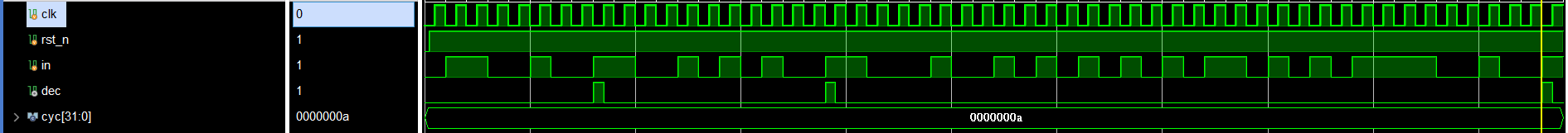


這題是上次lab的進階版，這題是一個mealy machine也就是output要跟著state改變，需要注意的地方就是當我state等於s2的時候，再接收到1是要回到s2，因為整串數字就是111，便從最右邊兩個1繼續讀取，相同的地方在s5和s6也有，因為讀進來可以是1100(10)+01，(10)可以很多次，所以s6如果input是1則回到s5。

Testbench:

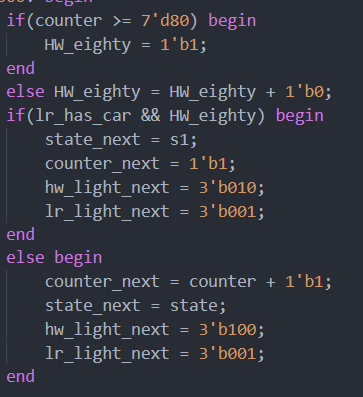
我測了6種測資，分別是11001001、110010101001、110001001 、10101010、1011010101、1111001001。

答案分別是1,1,0,0,0,1



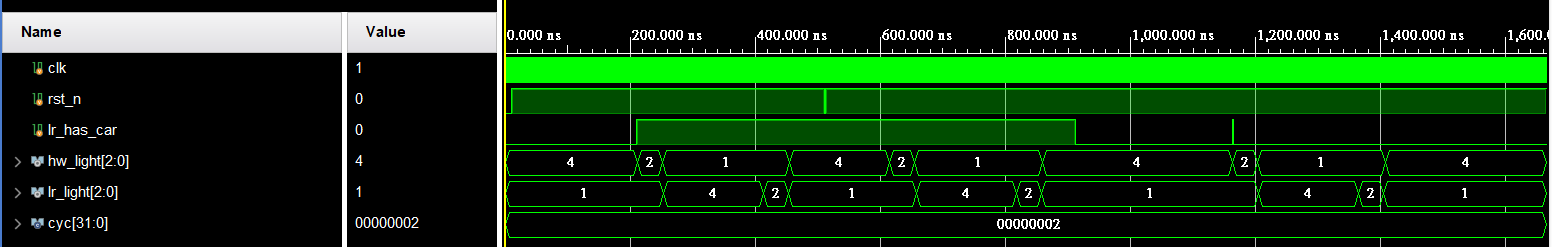
Advance\_2:

這題是要做紅綠燈，總共有6種，用FSM蠻好寫的，只要把每個state的狀態都寫好，也記得要寫default，一開始以為這題沒有甚麼陷阱，後來才發現如果等待的時間超過80秒很多的話，我用來記住時間的counter會爆掉，所以才改成

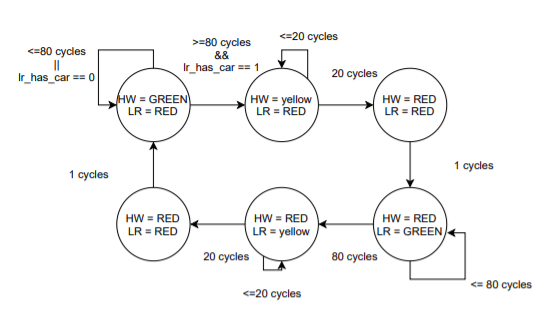


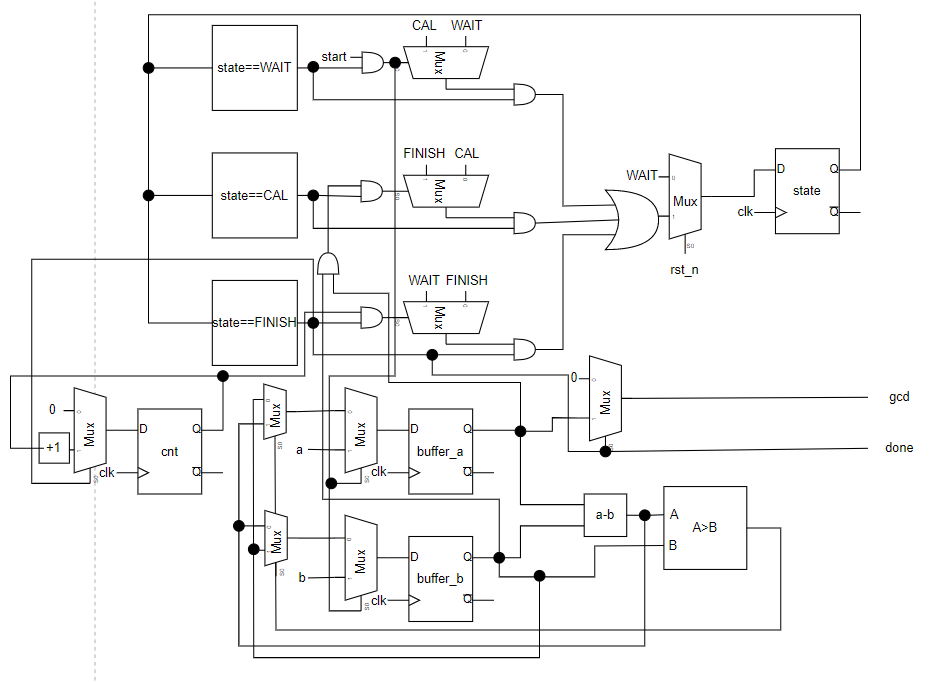
這樣即使overflow也沒關係。

Testbench:



前面就照著80秒、20秒、1秒、80秒、20秒、1秒來測試，也有測試reset，最後測試如果一開始綠燈的時間超過80秒且overflow的情況。

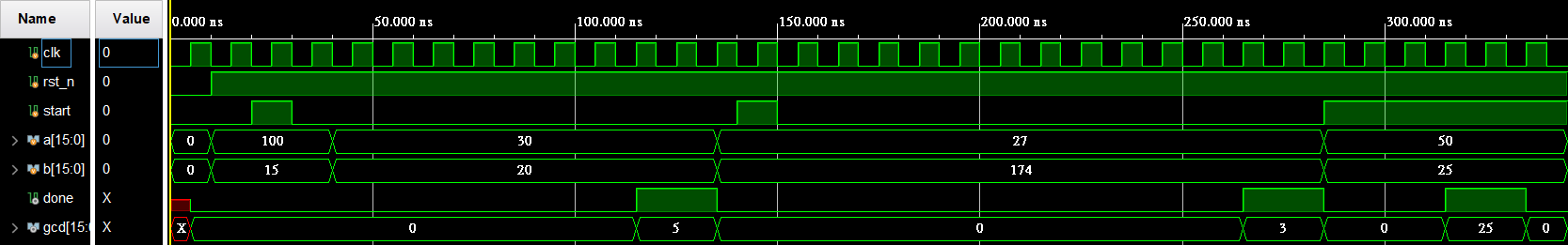


Advanced 3:

我們這題使用的state diagram如pdf上所附，所以就不再附上。首先，我們設定了3個state : WAIT、CAL、FINISH。當state==WAIT，代表state machine在初始狀態，gcd跟done這兩個output都是0，並且在start==1時轉換成下個state CAL，也就是next\_state=CAL。當state==CAL，便會執行找出最大公因數的功能，而為了完成此功能，我們利用輾轉相除法。由於每個clock cycle只能執行一次，所以設了5種16 bit的 reg : buffer\_a,next\_buffer\_a,buffer\_b,

next\_buffer\_b，以及一個運算元t。當state從WAIT變成CAL時，把a、b值存到next\_buffer\_a及next\_buffer\_b中，這樣下個cycle就會把buffer\_a跟buffer\_b更新成我們抓到的a跟b。接下來就每個cycle做一次輾轉相除

且如果buffer\_b==0時，就代表輾轉相除法做完並找出gcd，於是讓next\_state==FINISH，結束運算。當state==FINISH，代表我們已經找出最大公因數，所以讓output gcd=剛剛找出的答案，也就是buffer\_a，且讓done=1。由於題目要求要讓FINISH state維持兩個cycle，所以利用counter完成，讓counter在FINISH之外都為0，並在FINISH時每個cycle加1，數到1就讓next\_state=WAIT，就會讓FINISH在2個cycle後變成WAIT，完成題目的要求。



我們測試的測資有a=16'd100,b=16'd15、a=16'd27,b=16'd174、a=16’d50,b=16’d25。並且觀察一些當不是我們要偵測input時且input改變會不會出現錯誤，waveform如下:

Advanced 4 Bonus:

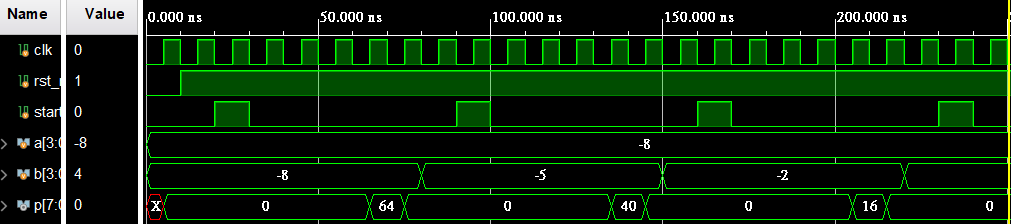
我們這題使用的state diagram如pdf上所附，所以就不再附上。首先，我們設定了3個state : WAIT、CAL、FINISH。當state==WAIT，代表state machine在初始狀態，output p=0，並且在start==1時轉換成下個state CAL，也就是next\_state=CAL。當state==CAL，便會執行我們的

Booth multiply，總共執行4個clk cycle，並在4個cycle後將state變成FINISH，也就是next\_state=FINISH。而booth multiply的方法為 : 我們利用一個reg[8:0] ans以及reg[3:0]cur\_a, cur\_b做運算。在WAIT變到CAL的時候，將input a跟b的值存到cur\_a,cur\_b中，並讓

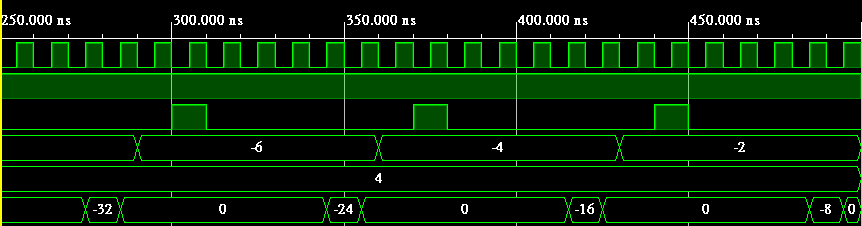
ans = {A,cur\_b,cur\_b-1}，且A的初始值為4’b0000，cur\_b-1的初始值也是0。接著便是連續判斷4次，總共有四種情況 : (1) cur\_b0 cur\_b-1 == 10 : A = A - cur\_a，並且將ans右移1 bit

(2) cur\_b,cur\_b-1 == 11 : 沒有操作，並且將ans右移1 bit (3) cur\_b,cur\_b-1 == 01 : A = A + cur\_a，並且將ans右移1 bit (4) cur\_b,cur\_b-1 == 00 : 沒有操作，並且將ans右移1 bit。每一個clk cycle分別判斷為哪種情況，並做出相對應的操作。當在state==CAL過了4個cycle，next\_state就會變為FINISH，並且如果cur\_a==-4’d8，output p= -ans[8:1]，其他情況則是p= ans[8:1]。並在一個cycle後state變回WAIT，完成我們的設計。

我們測試的測資有a=-8、b=-8,a=-8、b=-5,a=-8、b=-2,a=-8、b=4以下是固定a。



a=-6、b=4,a=-4、b=4,a=-2、b=4以下為固定b，waveform如下:



FPGA 1:

FPGA\_1

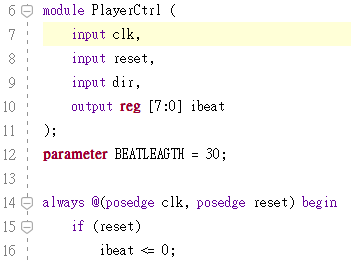
這題主要的作法是Trace code看出basic lab中的module分別能做哪些功能，再針對我們需要的部分改一下code，於是分成每個module講解:

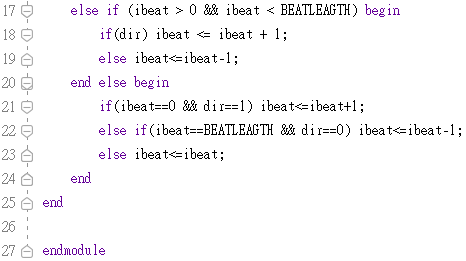
PlayerCtrl:

PlayerCtrl則是用來判斷現在撥放到譜的哪個位置以及決定要往上撥放還是往下撥放。

這個module有三個input clk,reset,dir，當dir==1時往上撥放，dir==0時往下撥放，且在每次posedge clk時判斷。最後當reset==1時，將撥放位置重置到Music的最前面，並把dir設為1。

Parameter BEATLEAGTH 代表我們譜中音符的數量，而output ibeat就是在Music中的input的ibeatNum。





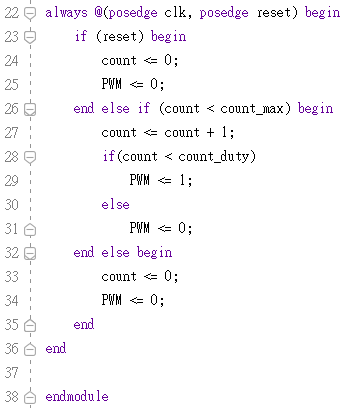
PWM\_gen:

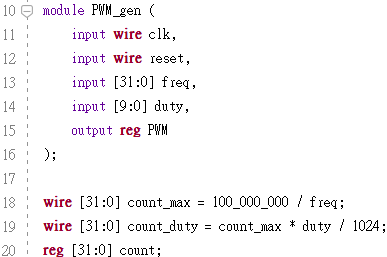
PWM\_gen的用途跟clock divider很像，能根據我們的需求產生對應的frequency。

根據input [31:0] freq來決定一秒要撥放幾個音符，例如我們題目需要兩種頻率 : 1秒一個音與0.5秒一個音(1秒兩個音)，則這兩個情形的freq分別等於1跟2。Count\_duty的作用則是讓產生的frequency圖形可以整理比較漂亮。

在我們的Top module中，總共使用了3個PWM\_gen : (1)Tone\_gen : 負責聲音輸出 (2) Fast\_gen :

產生1秒兩個音符的frequency (3) Slow\_gen : 產生1秒一個音符的frequency





Top :

Top便是將上述的module全部整合以後，做出題目的要求。

SampleDisplay :

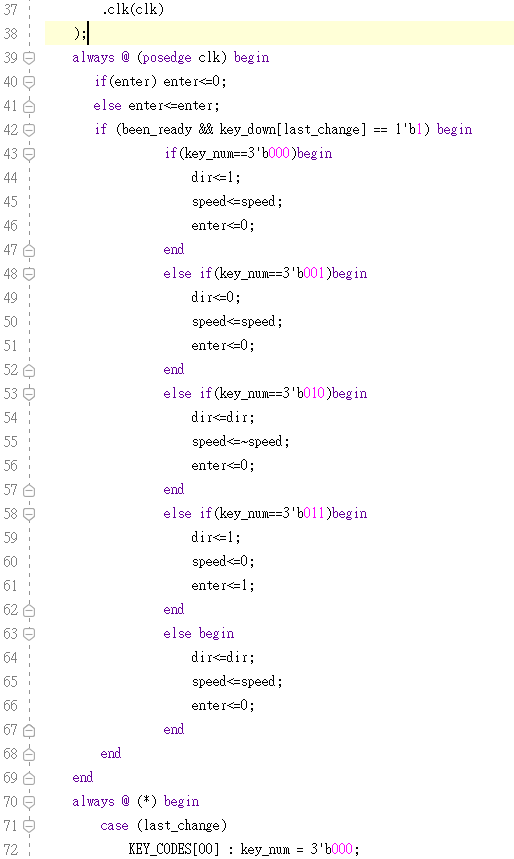
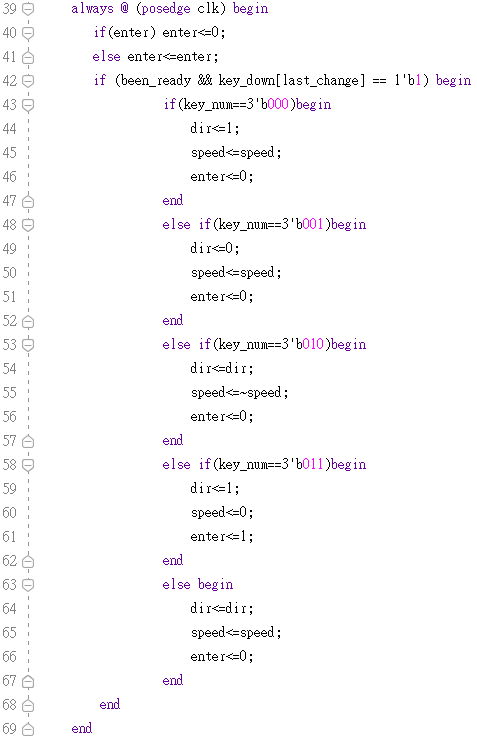
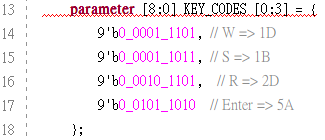
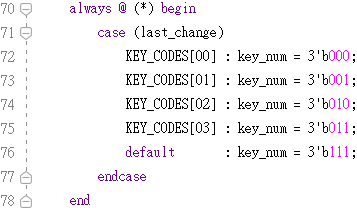
SampleDisplay則是用來偵測按下鍵盤的哪個鍵以及做出相對應的功能，在這題需要用的四個鍵為w、s、enter、r。首先，要先找出這四個鍵各自的MakeCode : W=>1D,S=>1B,R=>2D,ENTER=>5A

接著再分別給四個鍵各自的key\_num，從3’b000 ~ 3’b011。最後再決定按下每個鍵時會發生甚麼事，例如按下W會讓dir變成1。我將code放在下一頁

SampleDisplay :

SampleDisplay則是用來偵測按下鍵盤的哪個鍵以及做出相對應的功能，在這題需要用的四個鍵為w、s、enter、r。首先，要先找出這四個鍵各自的MakeCode : W=>1D,S=>1B,R=>2D,ENTER=>5A

接著再分別給四個鍵各自的key\_num，從3’b000 ~ 3’b011。最後再決定按下每個鍵時會發生甚麼事，例如按下W會讓dir變成1。我將code放在下面

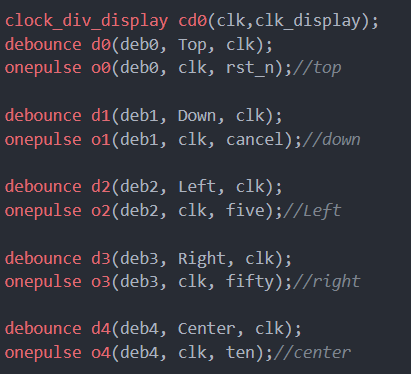


Top :

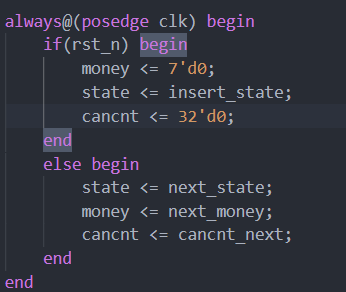
Top便是將上述的module全部整合以後，做出題目的要求。

FPGA\_2:

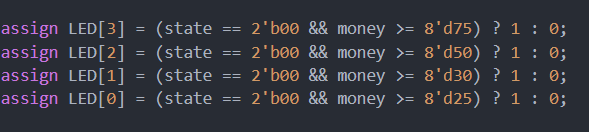
這題是要寫一個販賣機，因為連續兩次都不是我負責FPGA，所以這次花了很多時間在這題上面，原本我是想了三個state，分別為投幣、買、退錢，後來想想發現買可以跟投幣合併，因為最多只會買一罐，這樣比較方便，所以最後我只有買和退錢兩個state。



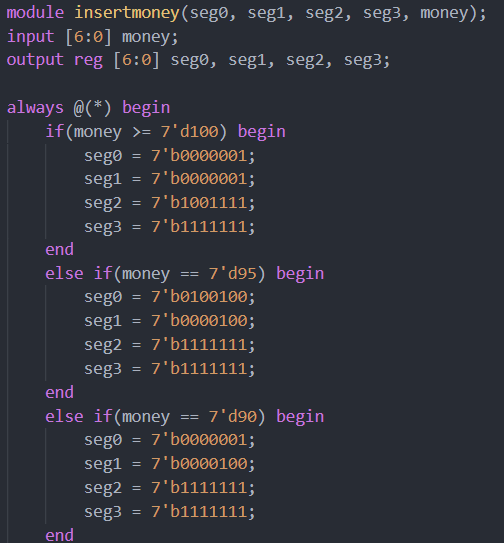
這邊就是負責clock divider和五個按鍵的onepulse 跟 debounce。



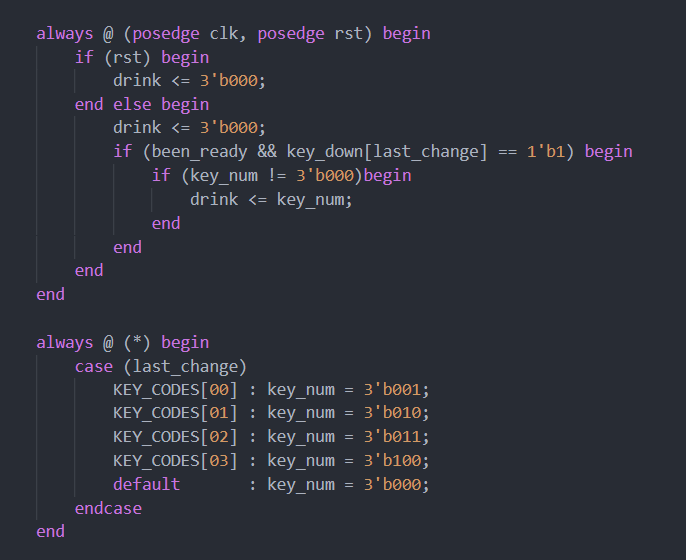
上面這個sequential circuits則是負責處理歸零跟state的轉換，money代表現在總共投了多少錢，cancnt則是在處理退錢時算秒數的counter。



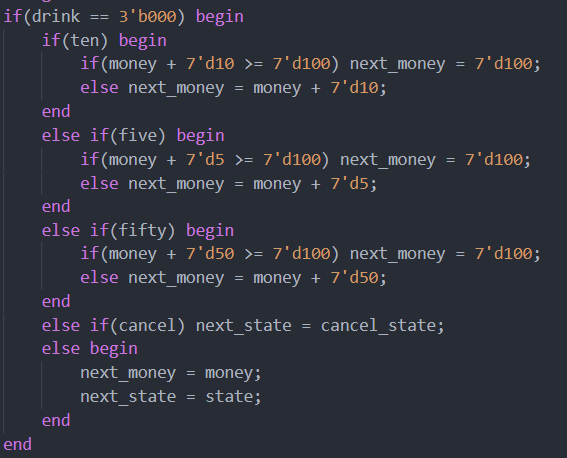
這是在處理LED顯示，必須在state在投幣的時候且錢需大於要買的飲料。



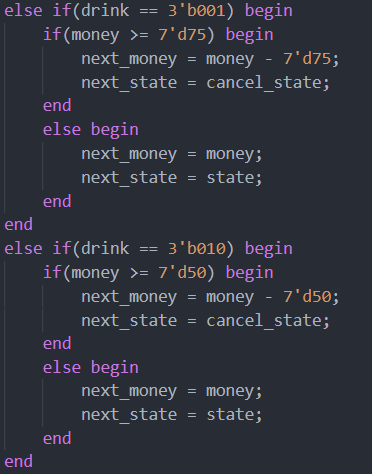
這個module則是在處理display數字的部分，因為沒想到更好的辦法所以選擇用窮舉的方式，以下就先省略。



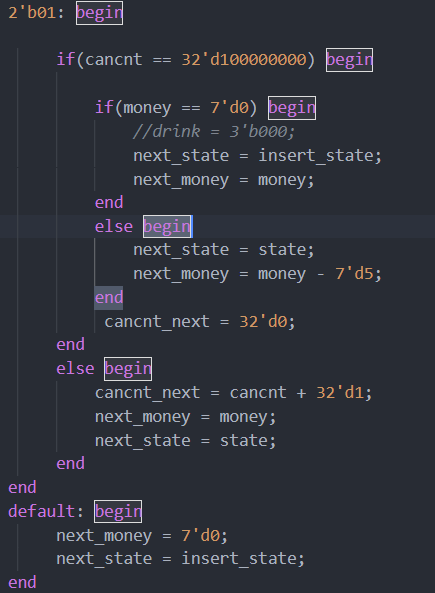
在處理鍵盤的部分我也是研究了一陣子，最後我用了一個drink當作我選擇的飲料，如果沒選就是3’b000，剩下就以此類推。



這邊就是在insert\_state的時候，因為還沒選飲料所以drink要等於3’000，在投每個錢的時候也要判斷有沒有超過100，另外就是cancel這個鍵要換state。



這邊就是處理買每個飲料的時候，要先判斷錢夠不夠，不夠就state維持，其他飲料也是如此。



這裡就是處理退錢，用一個counter倒數計時，這樣每一秒就會減5，當我的money等於0就轉回原本insert\_state。

分工: 徐嘉徽寫advanced3、4，FPGA 1

李佳栩寫advanced1、2，FPGA 2

心得:

徐嘉徽 :

這次的advanced question跟之前沒有甚麼太大的差別，所以寫起來沒有遇到太多問題，主要是第四題的booth multiplier要上網查很多資料才慢慢了解。比較麻煩的是FPGA 1要用到新教的鍵盤跟揚聲器，都要慢慢觀察之前basic lab的code去了解每個地方代表的功能是甚麼，才能改造成自己需要的功能，這次的FPGA是一個比較新奇的體驗 !

李佳栩:

這次是久久以來第一次寫FPGA，我其實遇到了很多問題，其中最大的問題還是coding style的問題，我常常case都忘記default，導致我的FPGA跑不出東西，然後寫太多if-else導致我的變數跑來跑去，這樣也沒辦法合成。我才了解到coding style的重要，另外我想提出我找到的bug，我發現在當我95塊錢然後要投入50塊的時候，我明明有寫money + 50要小於100，但還是會加上去，後來我才想到因為95+50等於145，已經超過我的money的bit(最多到128)，其實已經超過100了但是因為bit不夠所以錯了，最後我就把money多開一個bit就解決了，其實還有另一bug就是我的鍵盤只能夠買一次東西，後來發現是忘了default，才解決問題的，verilog的難度真的超乎我的想像，但我會加油的。