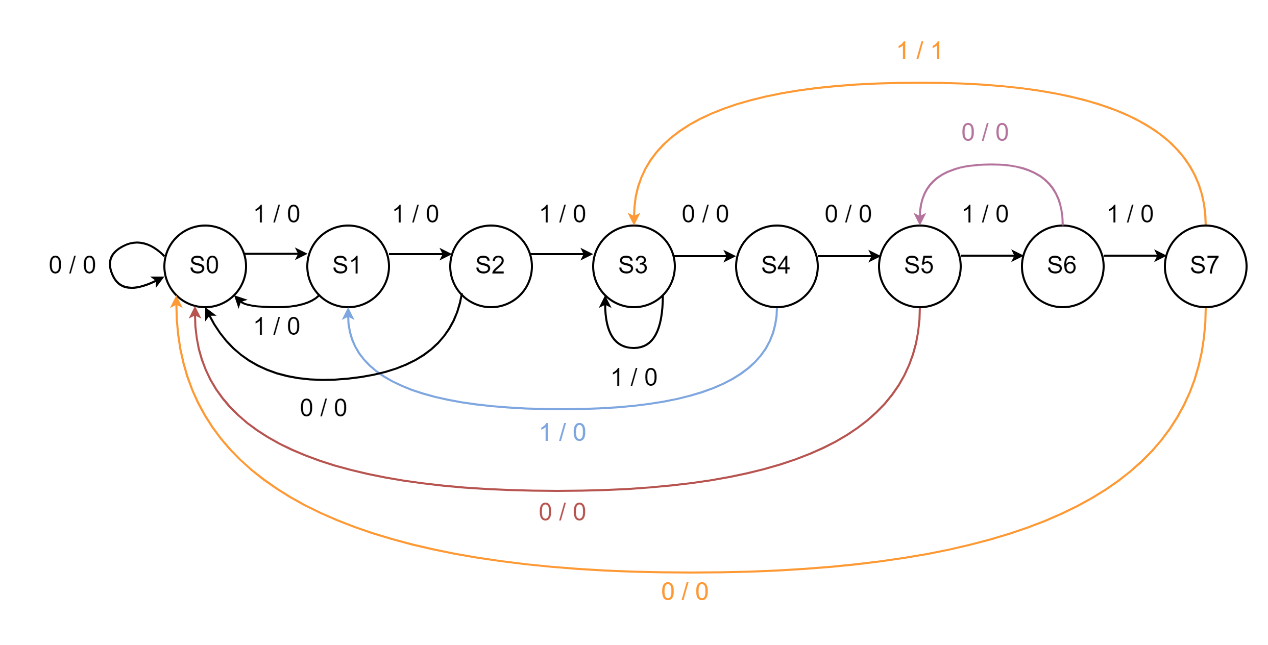
Lab 4 report

組員：110062221李品萱

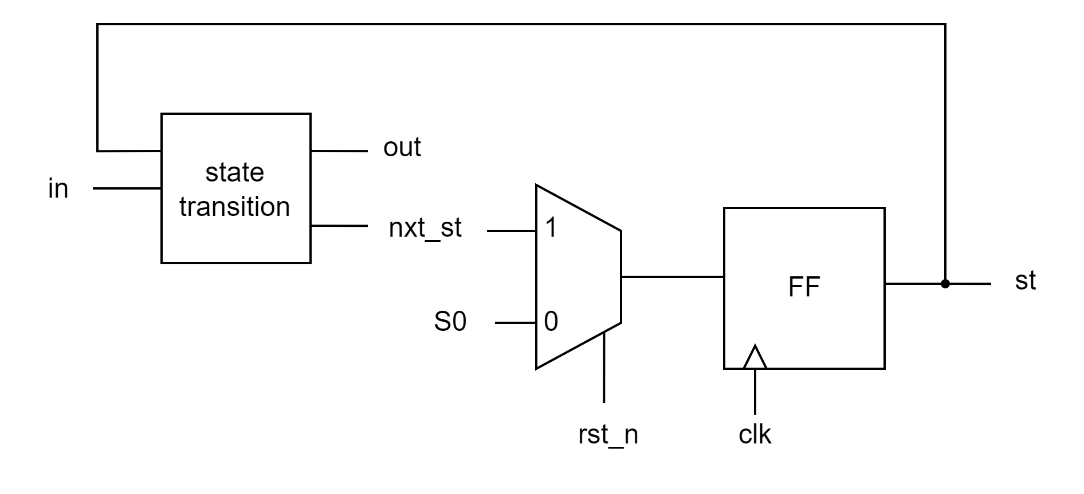
110062213唐翊雯

1. **Sliding window sequence detector**

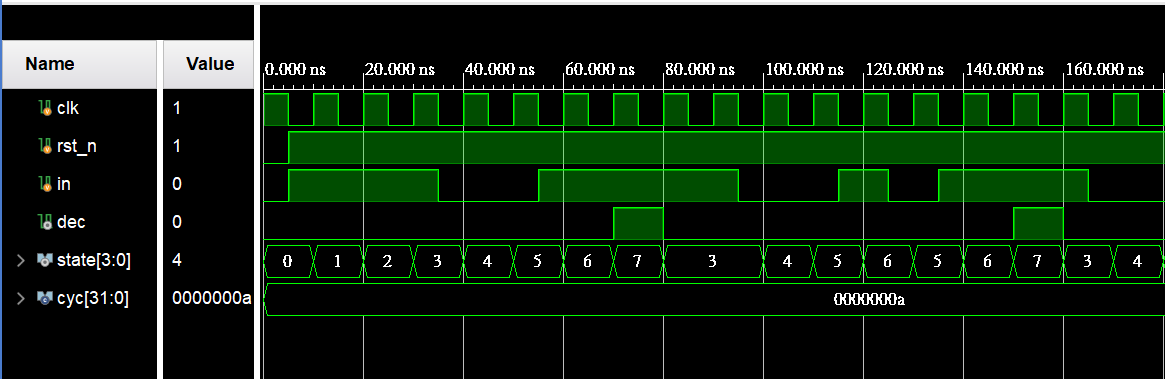
這題我們需要用Mealy machine作連續的detect，找到sequence 1110(01)+11，state transition diagram如下圖。S0到S2都是要收到1才能進入下一個state對應到sequence的前三個1，在這個階段如果收到0都不是合法的，會直接回到S0；但由於是用sliding widow的方式檢查，因此S3若收到1則會停留在S3，一直到收到0才會進下一個state；S4則是要收到0才合法，因此若收到1則會視為sequence已input一個1，因此進到S1；在S5及S6的部分，由於只能存在01且可以存在多個01，因此S5收到1進到S6，S6若收到0則回到S5，檢查下一個是否為1；S7的部分則代表前面已經收到1110(01)1，因此這邊若收到1則會output dec為1，收到0是不合法的，則回到S0。



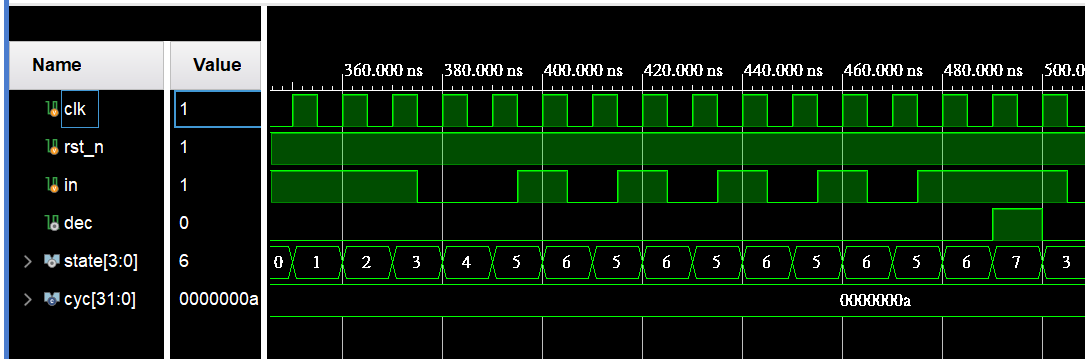
這題的block diagram如下。



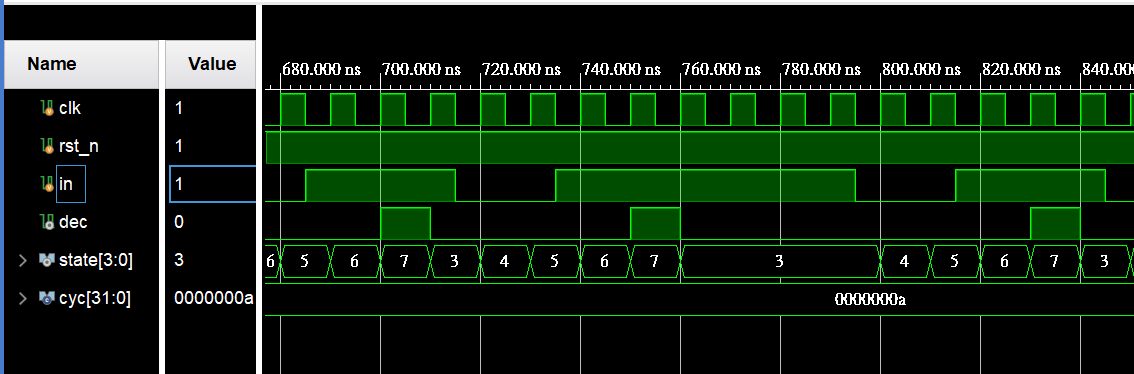
在testbench的部份，為了方便檢查我們將state也印出來，首先做出spec給的sample waveform，如下圖。



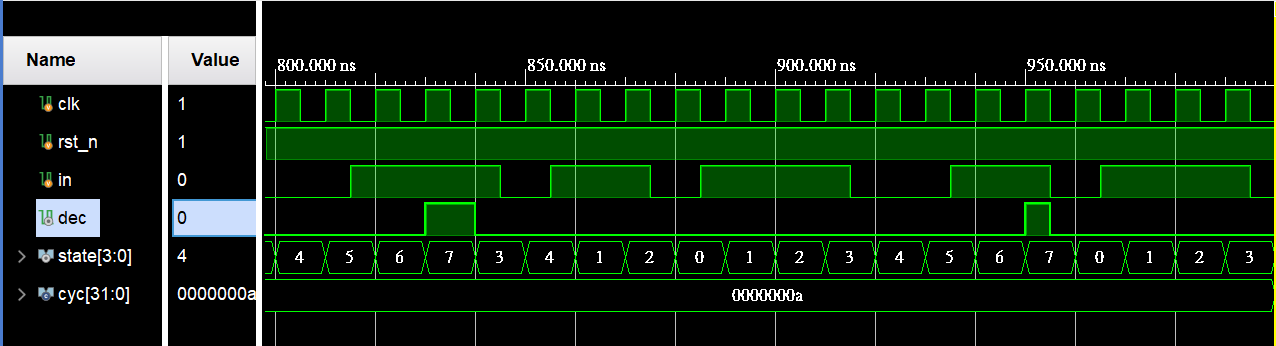
接下來我們測01在1110與11之間出現多次，確認符合預期結果。



接下來檢查我們dec為1之後的sequence開頭不給完整的三個1，及持續給1的狀況，依序是下圖中黃色線及藍色線處，皆符合state transition的結果。

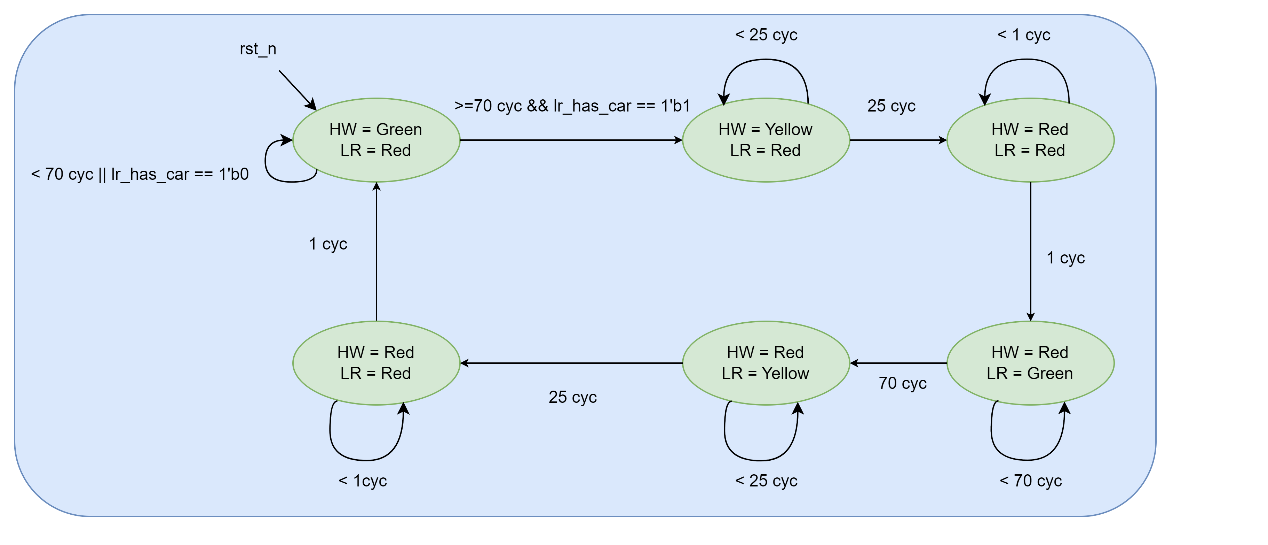


我們也測試不合法的輸入，皆符合預期結果，而在下圖橘色線處dec會產生一個clock的1是因為它在s6時讀到1，而後在clock起來，state變7時又讀到1，但在該clock low時input為0，不是合法的sequence，因此dec此時為0。

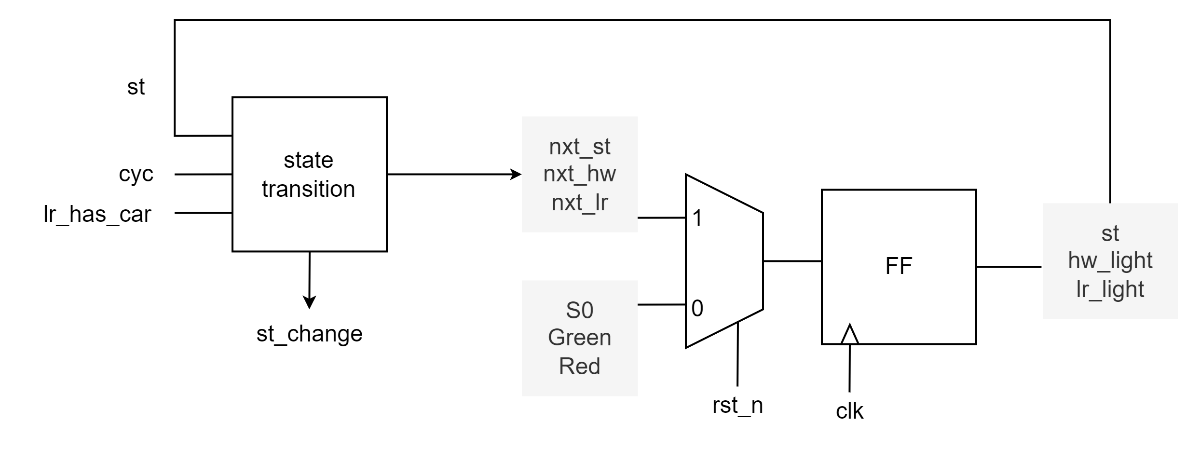


1. **Traffic light controller**

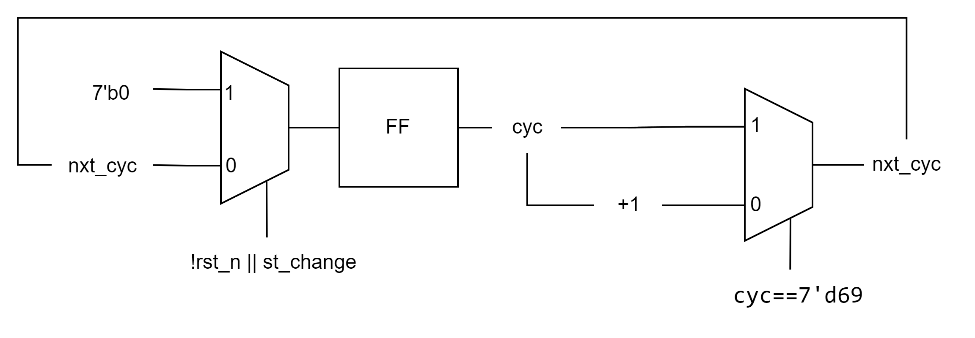
這題我們要做出一個紅綠燈，基本的state transition都在spec中有給了，我們基於spec再將state transition diagram處理得更完整，如下圖。

****

這題的block diagram如下圖，在這題的設計上，因為state transition是受clock cycle控制的，我們將燈號跟state一起變，並將st\_change作為FSM的output，也用來控制計算clock cycle，clock cycle控制的設計如下段所述。

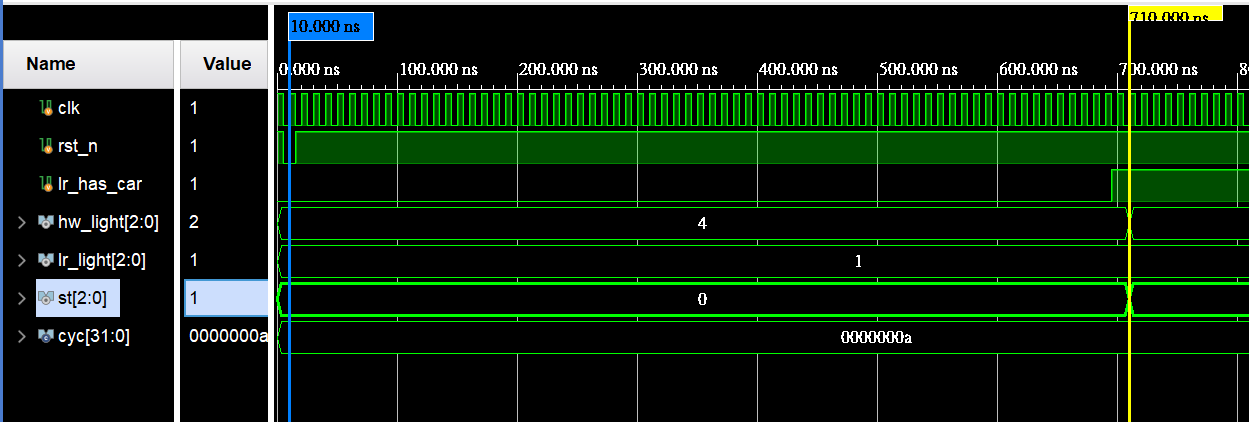


如下圖，在計算clock cycle上，我們讓state change發生時將cyc歸0，否則加1，因此只要state一變cyc就會開始數，並配合FSM的設計停留在原本的state直到cyc符合要求；又由於這題需要的clock cycle最大只到70，因此我們讓cyc是69時就不繼續加1，方便我們決定cyc的bits數。



在testbench的部分，我們一樣印出state方便觀察。

首先確認cycle數是對的，如下圖，黃色線部分是710ns，而我們是在藍色線，即10ns時reset的，因此可確認它經過了70個cyc。

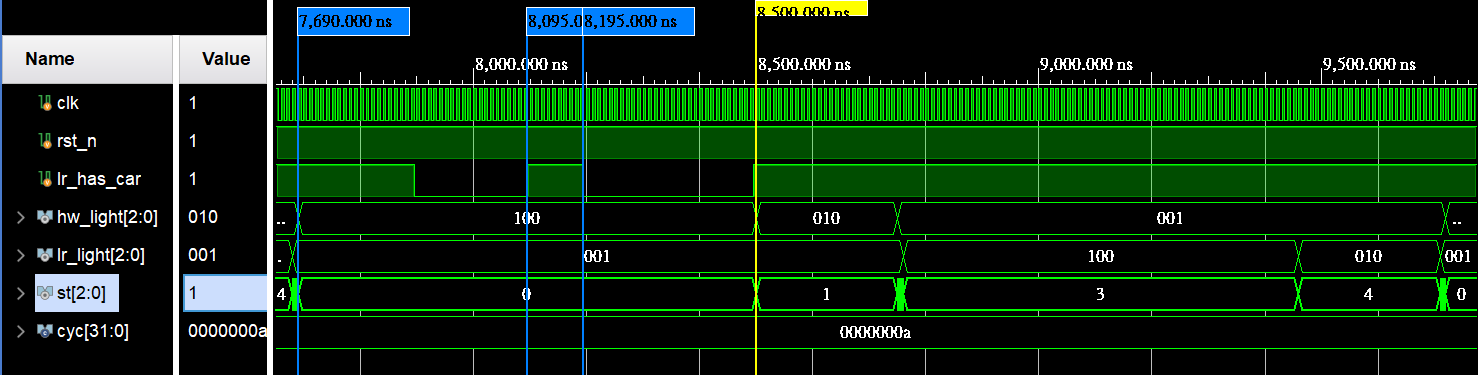


燈號變化的部份，我們按照spec，紅燈為001，黃燈為010，綠燈為100。首先我們測在70 cyc前lr\_has\_car就為1持續到70 cyc後，確認state transition及燈號都是對的，下圖我們也用藍色線標出各個state發生變化的時間，確認從state 0(HW=Green, LR = Red)到state 5(HW = Red, LR = Red)之間經過的cyc依序為(70, 25, 1, 70, 25, 1)

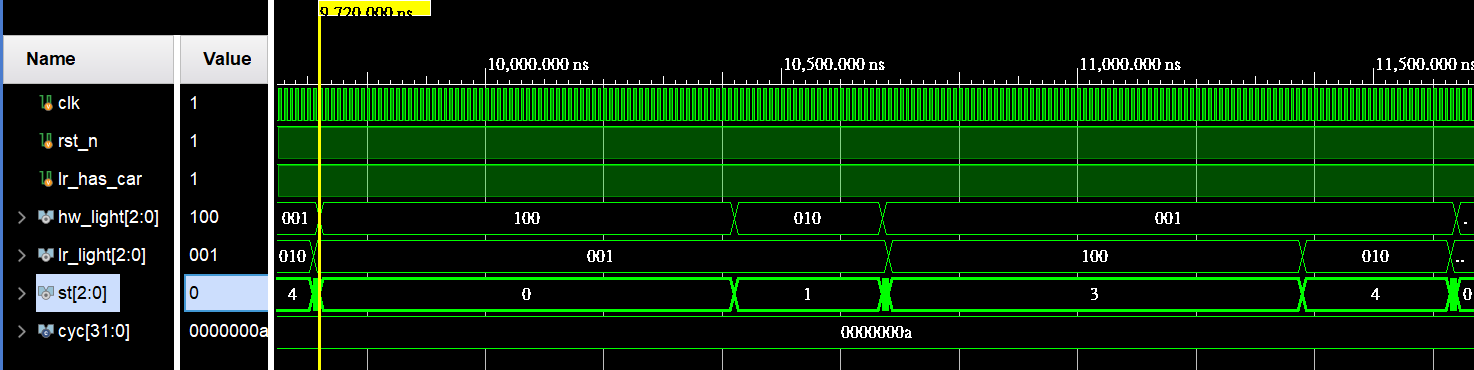
一張含有 文字, 計分板 的圖片

自動產生的描述

而後我們測在cyc未達70前lr\_has\_car為1，但在cyc不到70時lr\_has\_car就變0，如下圖，可以看到在大約經過50個cyc時，lr\_has\_car就有1到0的變化，但因為cyc未達70，因此不會進入下個state；而在cyc達70時，lr\_has\_car為0，一直到lr\_has\_car為1時，才進入下一個state（黃線處）。

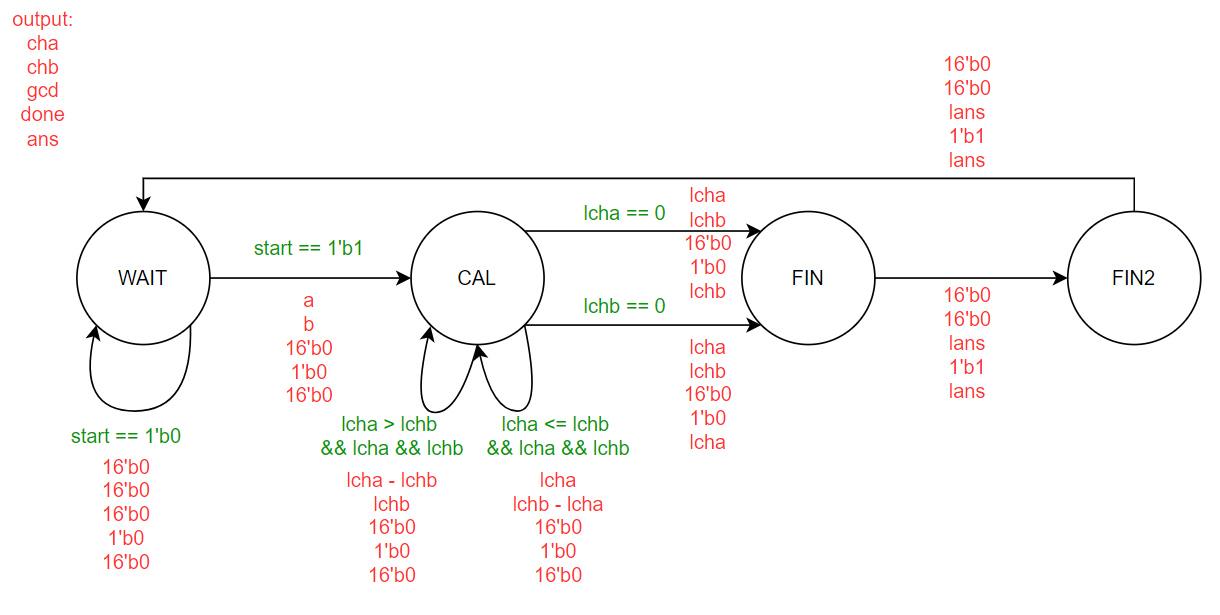


最後我們也對lr\_has\_car一直為1做確認，此情形義同在cyc為70之前lr\_has\_car就為1，如下圖。



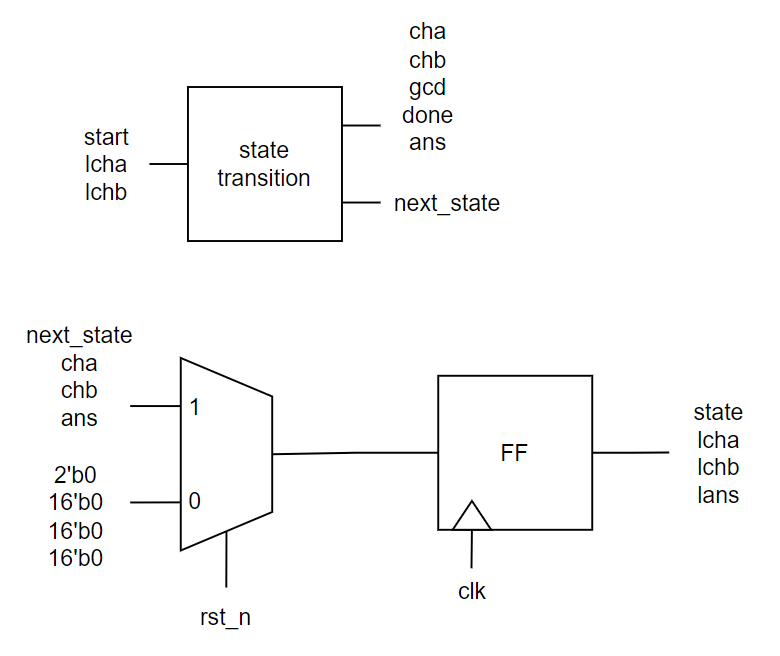
1. **Greatest common divisor**

這題我們要對於給定的兩個數，利用輾轉相除法找出最大公因數。我們依照給定的pseudo code來實作，其state diagram如下：

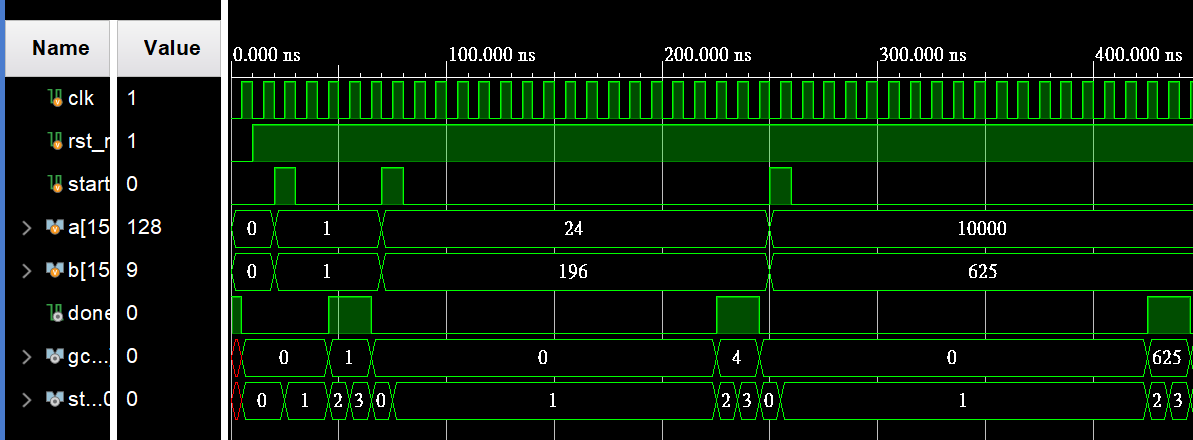


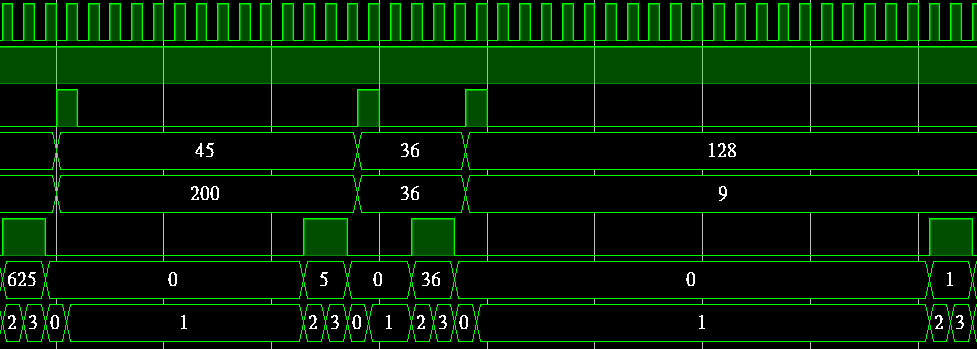
output的部分有cha（fetch出來的a）、chb（fetch出來的b），gcd，done跟ans（輾轉相除法的return值）。Input的部分有start、lcha（上一個clock cycle的cha）、lchb（上一個clock cycle的chb）。

sequential circuit的部分維護了state與前一個clk的cha、chb、ans的值，整體的block diagram如下：



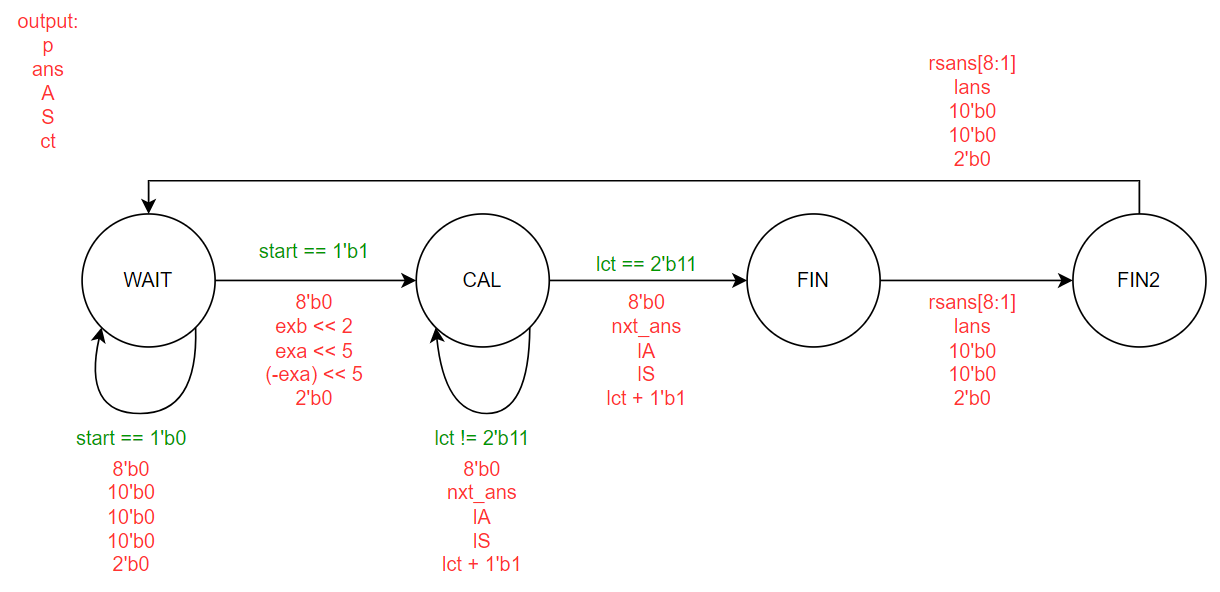
Testbench的部分，我們測了幾組數字，盡量列舉各種情況（e.g. a == b、a與b互質、a > b、b > a、gcd(a, b) == b…）以驗證其正確性。為了確認code有正常運作，我們將state印出來看，確定state的運作都是正確的。我們也事先推了每組數字應要在CAL這個state跑幾個clock cycle，以確認code有正常運作。



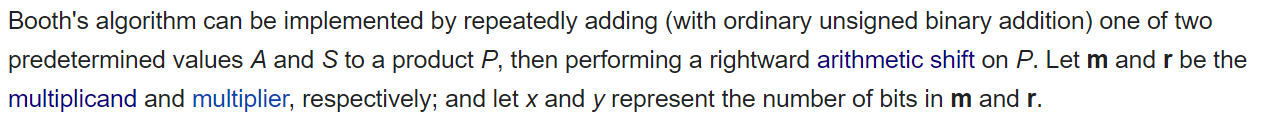


1. **Bonus: Booth multiplier**

這題要我們使用Booth Multiplier，算出給定兩個數字的乘積。根據spec給定的資訊以及作為參考的state diagram，我們做出的state diagram如下：

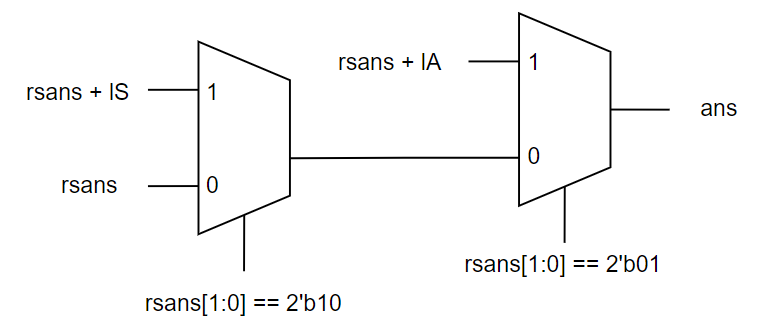


Output的部分有p、ans、A、S、ct。ct代表的意思是在CAL這個state中，已經做過幾次運算。而A跟S即是維基百科定義的A與S，而ans為維基百科定義的P：

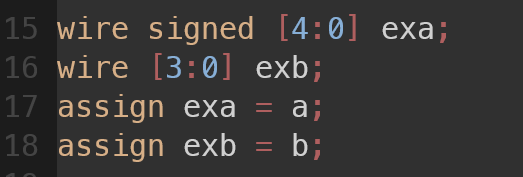


Input的部分則有start、lct（上一個clock cycle的ct）。

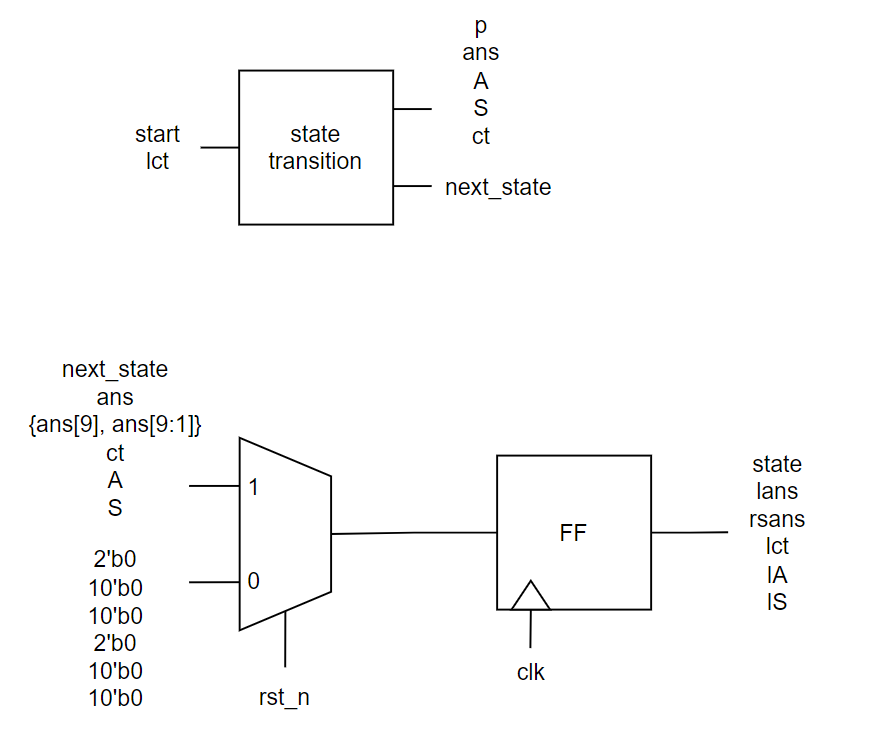
lct、lA、lS、lans分別代表前一個clk的ct、A、S、ans的值。rsans代表前一個clk的ans經過arithmetic right shift後的結果。在CAL這個state中，ans的輸出 (上圖以nxt\_ans表示) 較為複雜，沒有標註在state diagram上，其詳細的block diagram如下：



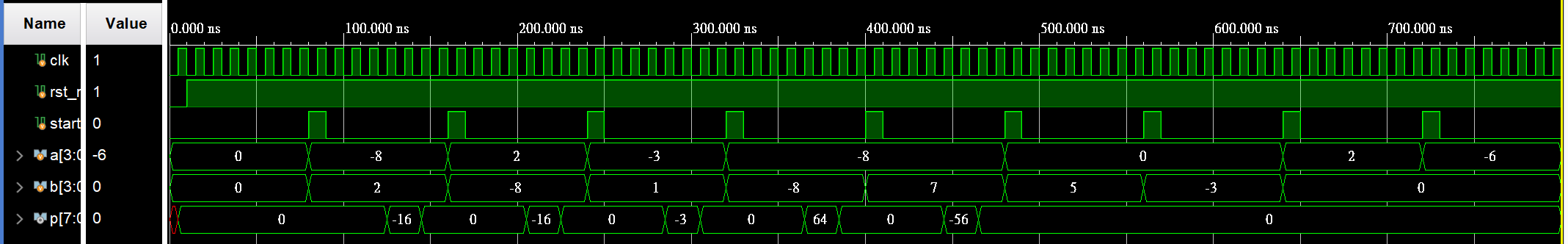
exa、exb則分別為input的a、b。但型態的部分，exa宣告成signed [4:0]，宣告成 [4:0] 是因為在計算S的值時需要用到 -a，這時若a為 -8，則 -a的值為8，無法被 [3:0] 存起來，因此多宣告1個bit來避免這個情況發生。宣告成signed則是因為從 [3:0] 轉成 [4:0] 要保持存的數字相同。Exb的部分，不使用signed宣告，這樣才不會在對ans初始化時改變ans的MSB。



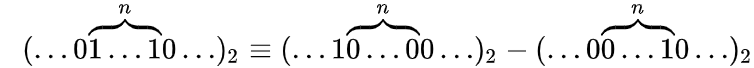
sequential circuit的部分維護了state與前一個clk的ans、ct、A、S、arithmetic right shift後的ans的值，整體的block diagram如下：



Testbench的部分，我們測了幾組數字，盡量列舉各種情況（e.g. a, b皆為正、a, b皆為負、a, b一正一負、a = 0、b = 0、a = -8、b = -8…），以驗證其正確性。



Booth Multiplier的原理是利用這個性質：



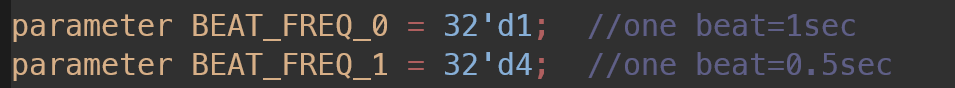
即一段連續的1可以被表示成兩數的差，其中這兩數皆為2的冪次。

因此當讀到01時，代表為一段連續的1的開始；讀到10時，代表為一段連續的1的結束。在這兩個時候要分別加上和減掉被乘數 \* 2^當前計算的次方。透過這樣的替換，可以比一般乘法器做更少的operation。

1. **FPGA: Mixed keyboard and audio modules together**

這題要把keyboard跟audio的module合在一起，並實作出spec上提到的功能。我們將basic提供的keyboard與musicbox的模板，做適當的修改並合在一起以完成這題。

Audio的部分，要修改的地方有要播出來的note以及更改note的速度。原本模板中的beatFreq為32’d8，對應到1 beat = 0.125 sec。根據題目要求，我們開了兩個parameter，32’d1與32’d4，分別對應到1 beat = 1 sec與1beat = 0.5 sec。

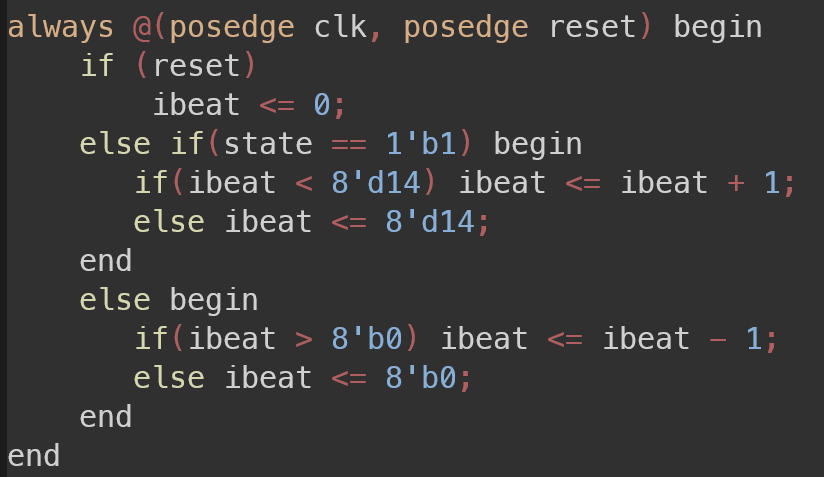


同時我們也要更改傳入module的freq，我們另外開了一個reg並傳入，這樣便能根據按下的鍵更改其值。

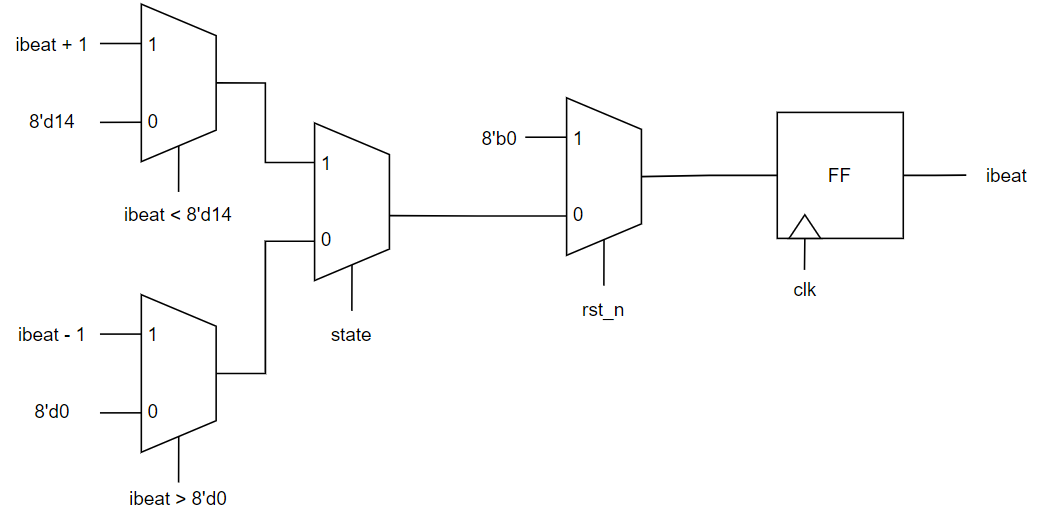
Note的部分，我們修改了Music、PlayerCtrl兩個module。PlayerCtrl的部分要修改總note數，只要把這項parameter改成題目給的15即可（總共有15個note）：



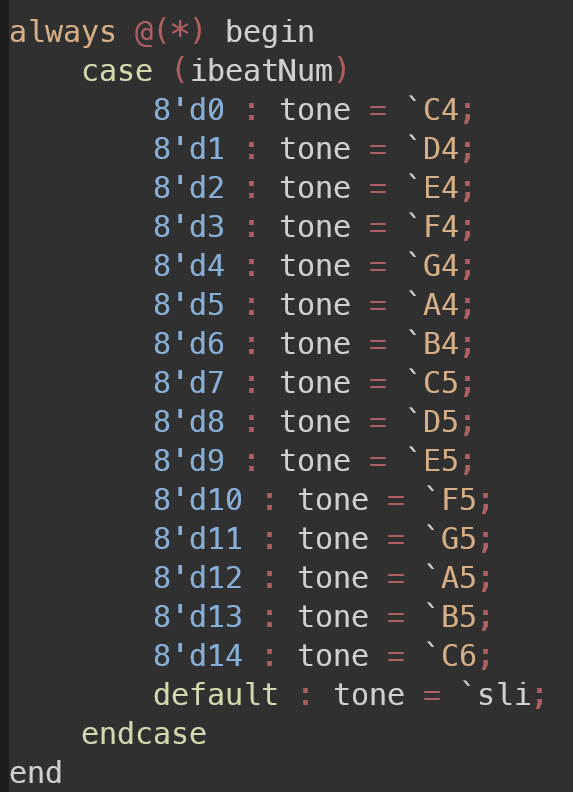
除此之外，由於題目要求 ”When it reaches C4 or C8, stay on the note”，以及note的變化要分為遞增及遞減，而不像原本的module只有遞增並且循環播放。因此我們對這邊的always block做了下列調整：



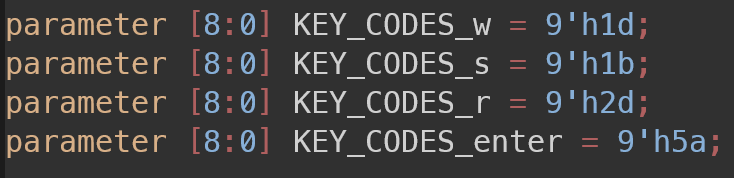
這部分的block diagram如下，其中state代表的是現在是遞增或遞減：



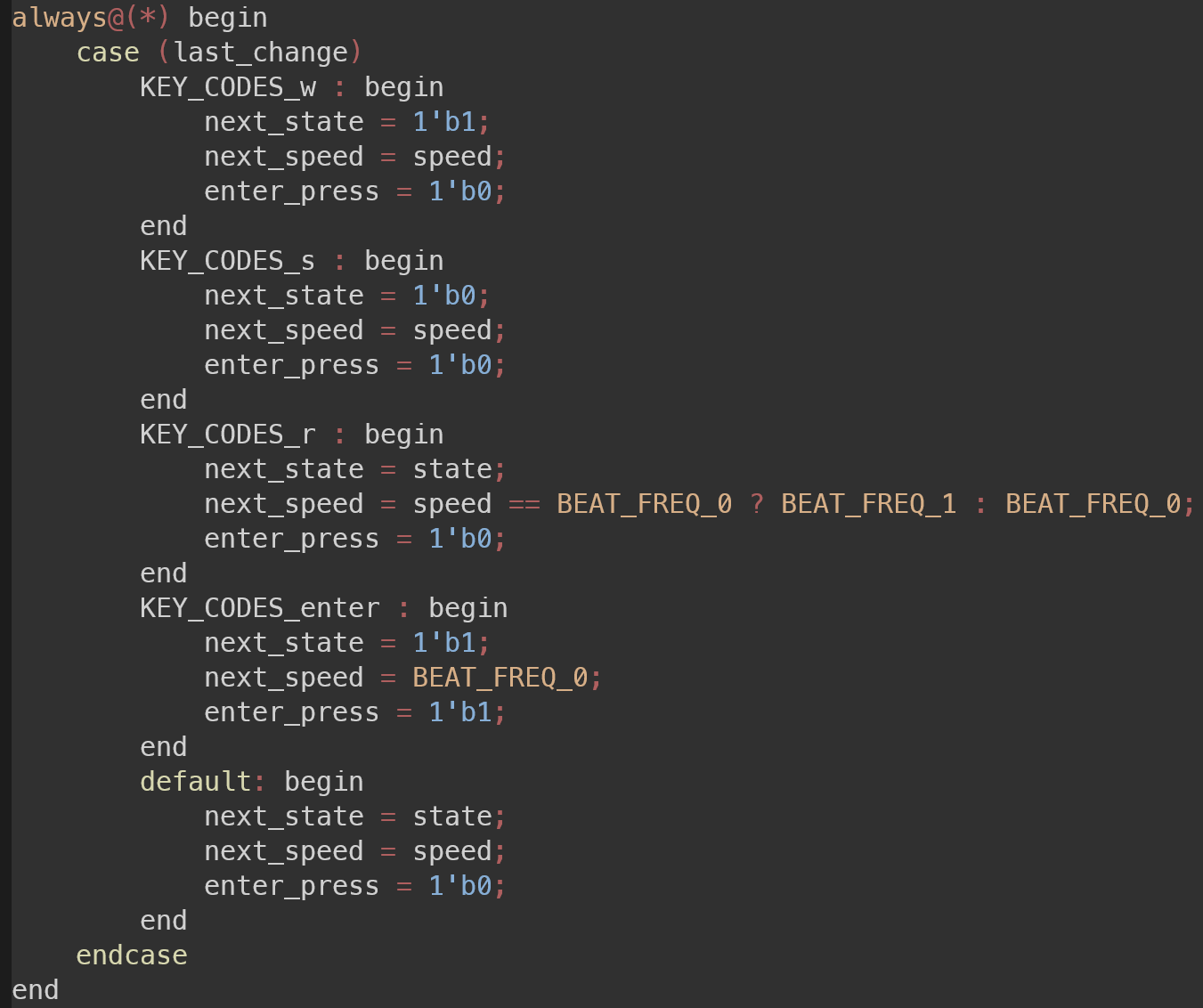
Music的部分，我們把原本小蘋果的旋律改掉，改成題目要求的C4~C6：



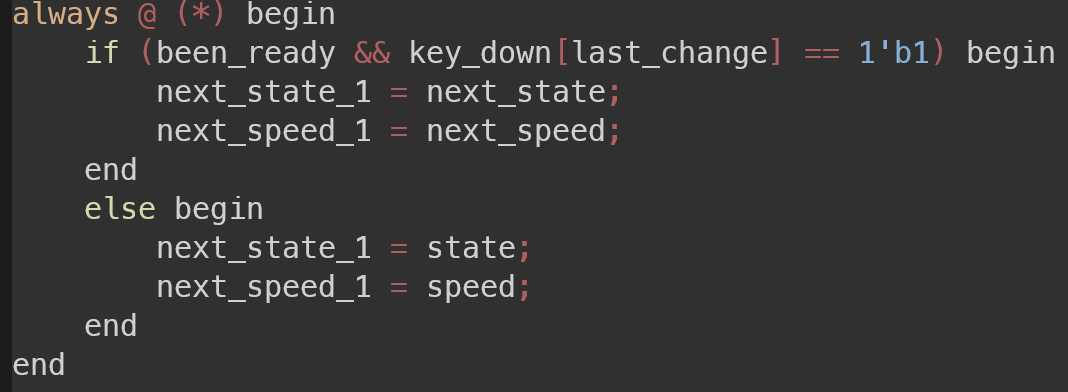
接著是Keyboard的部分，只需要對Top module做修改即可，首先我們會用到的鍵如下：

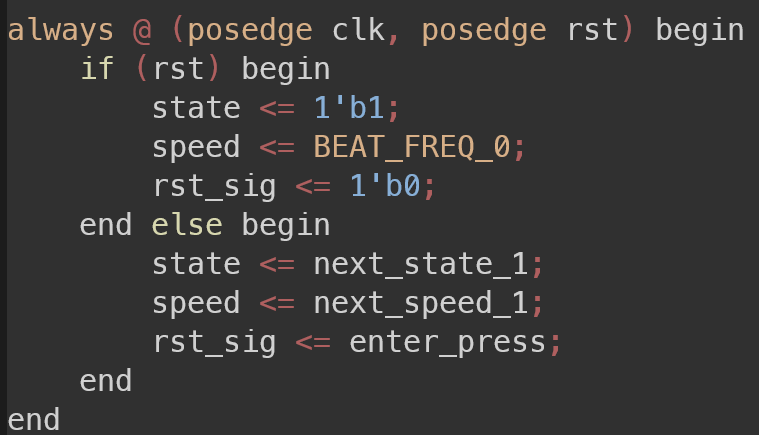


我們需要用鍵盤控制的reg有：state（現在為遞增或遞減）、speed（現在的速度為0.5 sec還是1 sec per note）與rst\_n。我們根據題目，在按鍵按下去時給予相對應的值：

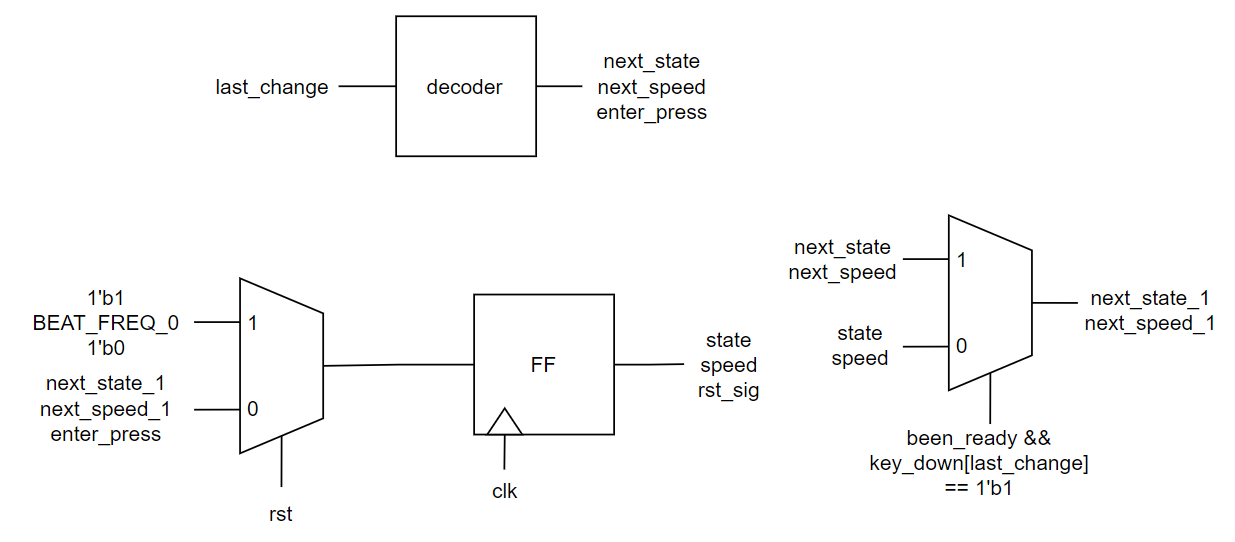


接著我們判斷last\_change時確認next\_state與 next\_speed的值，然後在sequential的部分，我們再把這些值傳給state、speed與rst\_n：





這些部分的block diagram如下：



將state與speed接回剛剛與audio有關的module後，即完成整份code。

1. **FPGA: Vending machine**

這次的FPGA我們需要用鍵盤跟板子實做一個販賣機，我們大致可分為以下部分：

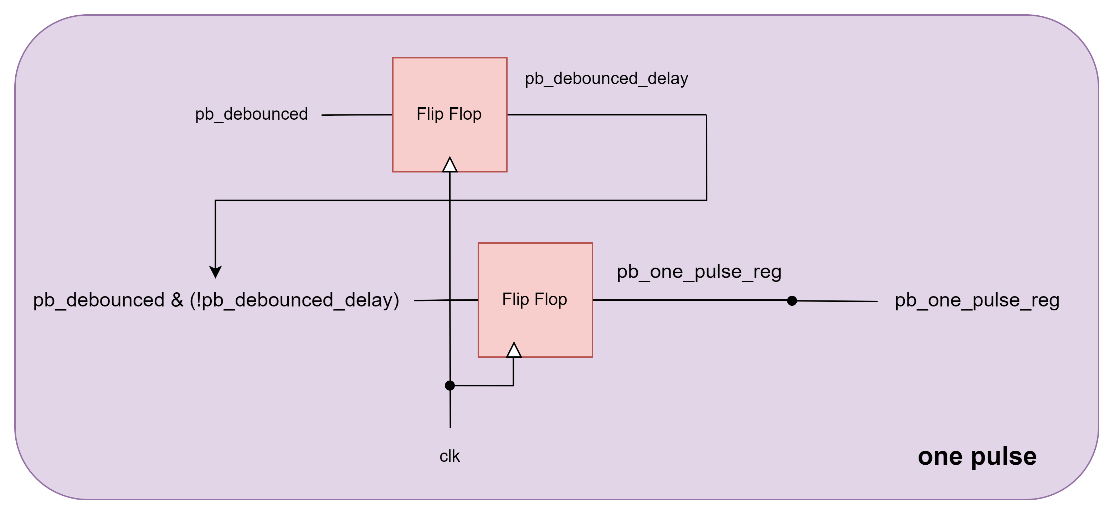
* 投錢
* 選擇可以買的飲料或取消購買
* 找錢
* 7-segment display

**投錢**

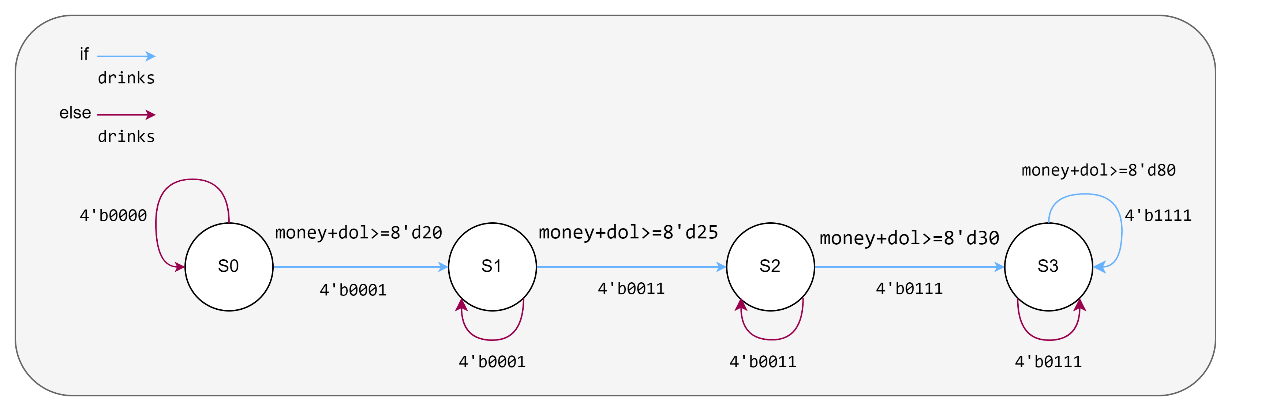
投錢是以按按鈕的方式，分別可以投5、10、50元，與前幾次lab一樣，我們需要對bottom做debounce跟one pulse的處理，分別如下兩張圖。

一張含有 文字, 時鐘 的圖片

自動產生的描述



由於投進的錢幣數值會決定可選擇的飲料種類，因此我們用FSM的方式處理，如下圖，dol代表這次投進的錢，money代表從開始投錢到現在累積的錢，drinks代表現在可以買的飲料種類，（0不能買，1可以買）。



**選擇可以買的飲料或取消購買**

接下來我們會用鍵盤的a, s, d, f鍵選擇要買什麼飲料，或藉由板子上的按鈕取消購買，按鈕的基本處理如前面所述，而鍵盤我們參考上課講義及basic的SampleDisplay檔案處理。

我們先將鍵盤的code對應出來，如下圖，每個parameter會是last\_change的判斷依據。

一張含有 文字 的圖片

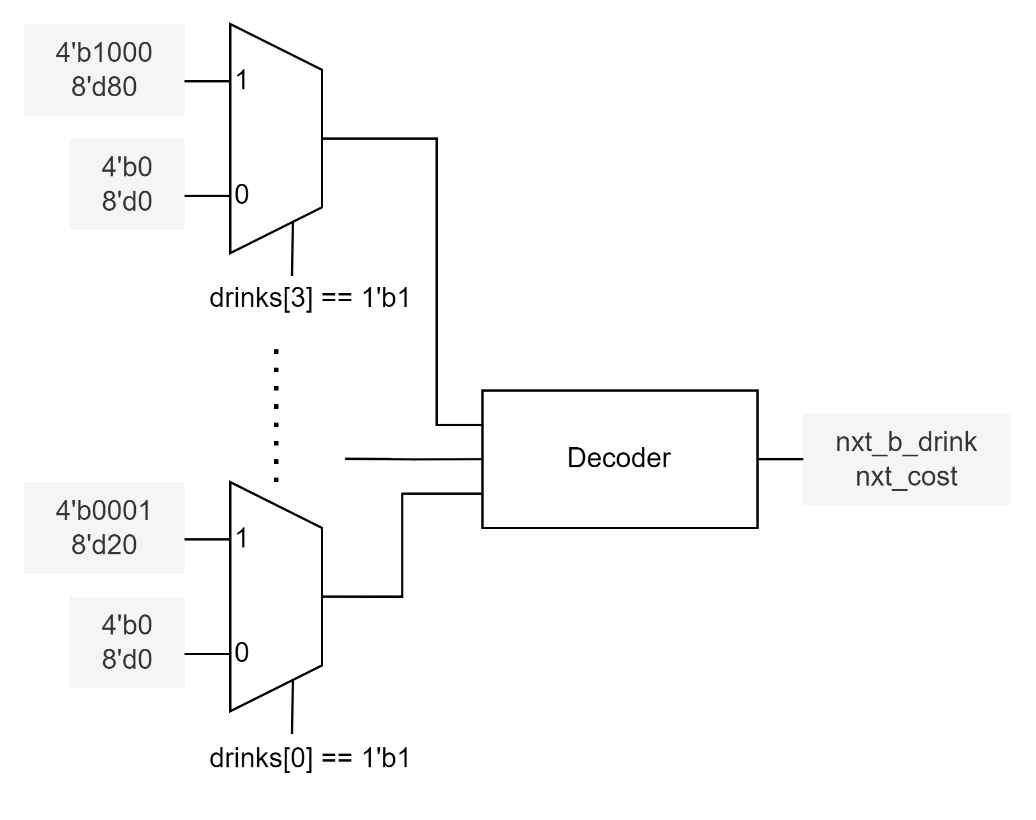
自動產生的描述

在判斷last\_change，即鍵盤按下什麼鍵時，我們也同時確認現在這個飲料是可以買的，以KEY\_CODE\_A舉例如下圖，這部份我們會得到買了什麼飲料及飲料所需的價錢。

一張含有 文字 的圖片

自動產生的描述

Block diagram如下圖。

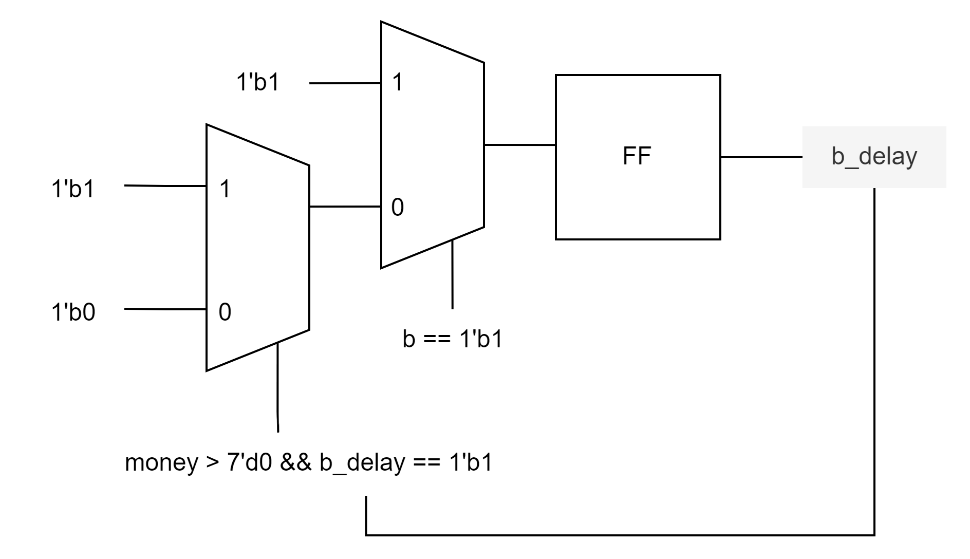


在判斷鍵盤有按下按鍵（即判斷key\_valid及key\_down[last\_change]）的部分，我們會多記一個b，b為1代表有成功買東西，0反之。因為b會拿來判斷現在是在投錢還是找錢，因此b如果是1的話，會一直維持1直到找過程結束，我們運用之前lab的ping pong counter中flip訊號delay的想法實現，設計如下，cancel也有做相同的處理。

一張含有 文字 的圖片

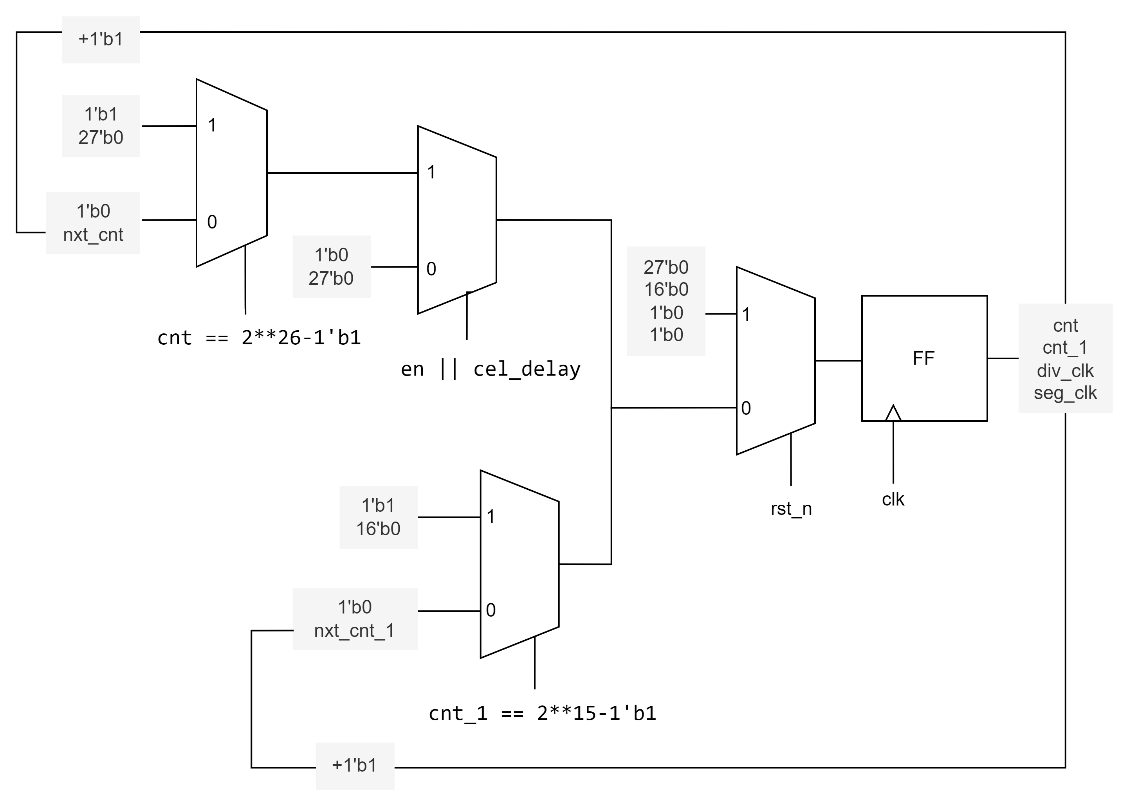
自動產生的描述

b\_delay的block diagram如下。

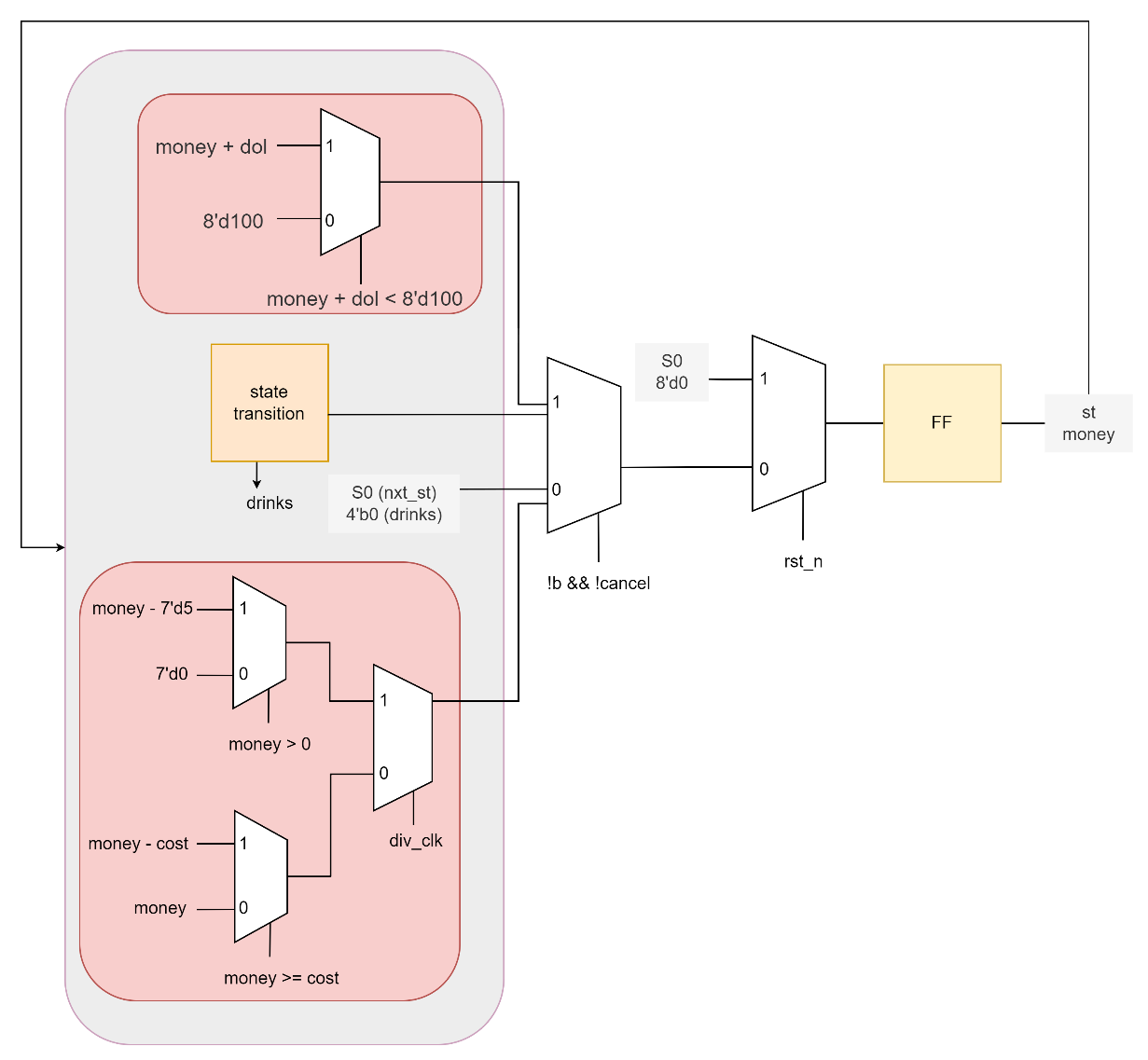


**找錢**

接下來是vending machine內部的運作部分，當b為1或cancel為1時（下圖的en||cel\_delay），會進入到找錢的環節，當這個環節被觸發後，state就會回到S0，drinks也會全部歸零，也就是代表什麼飲料可以買的LED燈會全部熄滅。如果是買飲料的話，會先顯示剩餘的錢再做每秒減五，為了做到這件事，我們讓負責做div\_clk的counter只有在b為1時才會做計算，這樣便可以讓我們顯示完剩餘的錢再做每秒減五；而cancel則是單純每秒減五，用clock divider控制，clock divider的block diagram如下圖，div\_clk控制每次退五元，seg\_clk控制7-segment的顯示。



而vending machine（不包含display）整體運作的block diagram如下。



**7-segment display**

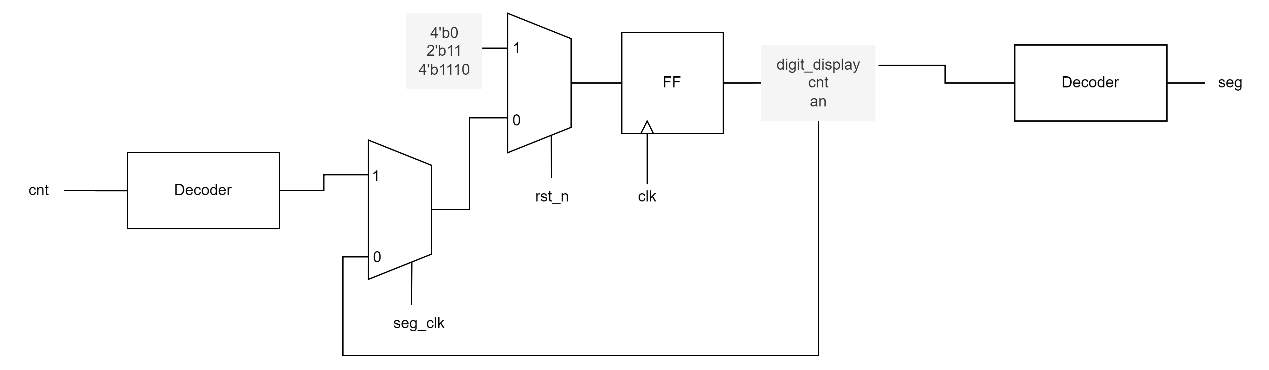
Spec中有提到不能prepend 0，因此在顯示上我們需要做一些處理。首先，在接各個數字的7-segment對應亮什麼時，我們多寫一個全都不亮的狀況，然後在輪流亮an時做判斷。

亮an[0]時，我們要亮的數值會是money - (money/7'd10)\*7'd10；

亮an[1]時，如果小於10，an[1]要全暗，否則亮money / 7'd100，但若money是100的話就不適用，而是直接亮0；

亮an[2]時，如果小於100，an[2]要全暗，否則亮money / 7'd100；

7-segment display的block diagram如下。



以上即為vending machine的實作呈現。

1. **Summary**

這次的Lab我們學習到如何運用聲音模組及鍵盤，這也讓FPGA的實作更有貼近生活的感覺，也讓我們對平常使用的鍵盤有更完整的理解，對音訊的處理也有了近一步的認識。

在advanced中，我們複習了FSM的使用，這次設計的mealy machine也比上次複雜，讓我們更了解mealy machine的運作。在紅綠燈的控制設計中，也了解到FSM在生活中的運用會是什麼樣子的呈現。

GCD 跟Booth Multiplier的部分其實很相似，我們學到如何好好運用FSM來處理問題。這兩題只要將state diagram定義完整便很容易實作，告訴我們先把state好好定義出來的重要性。

至於販賣機的實作部分，我們學到如何從0 design一個FSM，在不同的state transition的設計方式中，想到最適合的方式。一開始以為這一題會較為複雜，但實作下來發現除了一些細節要注意之外，沒有想像中的難，而最後的成果也的確作出了我們平常所知道的販賣機，非常有趣。

這次在跑vivado的時候，偶爾會遇到看不懂錯誤訊息的問題，花了不少時間，後來才發現大部分時候重開就好了，學到了一課：



1. **Contributions**

* **Code**

Sliding window sequence detector by唐翊雯

Traffic light controller by 唐翊雯

Greatest common divisor by 李品萱

Bonus: Booth multiplier by 李品萱

FPGA: Mixed keyboard and audio modules together by 李品萱

FPGA: Vending machine by 唐翊雯

* **Report**

各自描述負責的題目