Lab5 report

Team 34

110062238 潘茗脩

110062305 謝諺緯

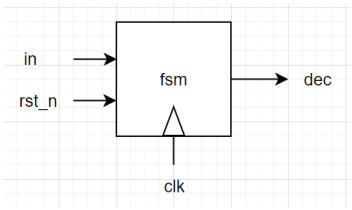
工作分配

潘茗脩(question 1, question 2, fpga demonstration 2)

謝諺緯(question 3, question 4, fpga demonstration 1)

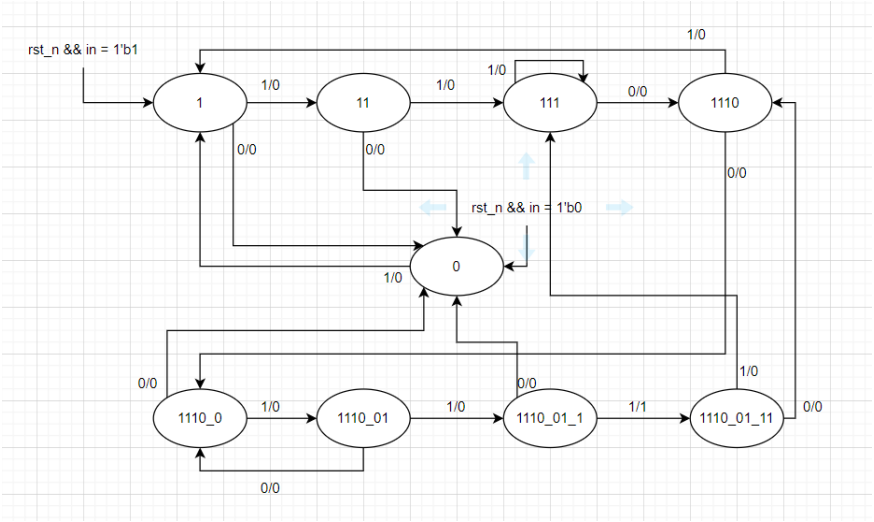
Question 1:

本題要求我們設計一個 sliding window sequence detector 並使用 mealy machine 設計。由上次的 lab 中，我們學到 mealy machine 的 output 是由 input 與 state 共同決定的。



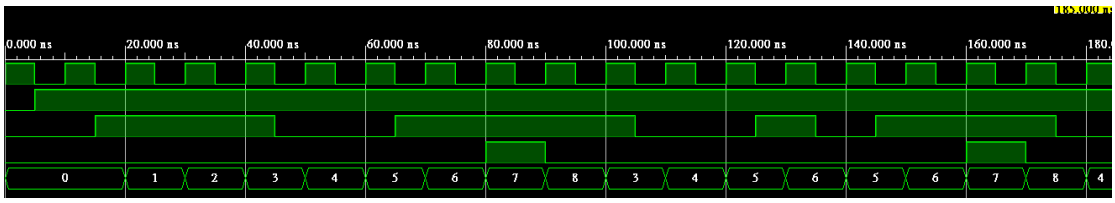
- Block diagram

本題畫出相對應的 fsm 即可解決，注意 1110(01)11 的(01)至少需要出現一次。解決方法便是當進入 1110_01 這個 state 時，如果 in = 1’b0，則回到 1110_0 state 即可實現重複(01)，以下是本題的 state diagram:

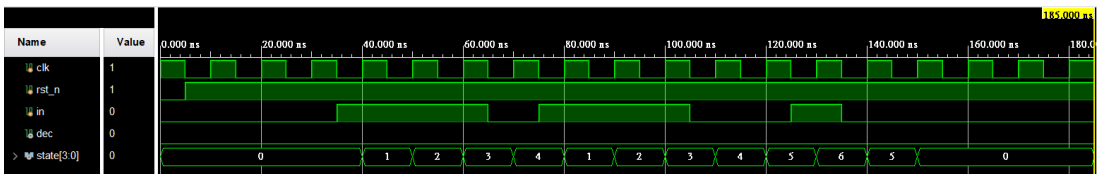


- Sliding window sequence detector

在測試的過程中，我們依照題目提供的 mismatch case 以及 match case 的波型圖來去設計 testbench，測試結果均與題目給的相符，完成測試。



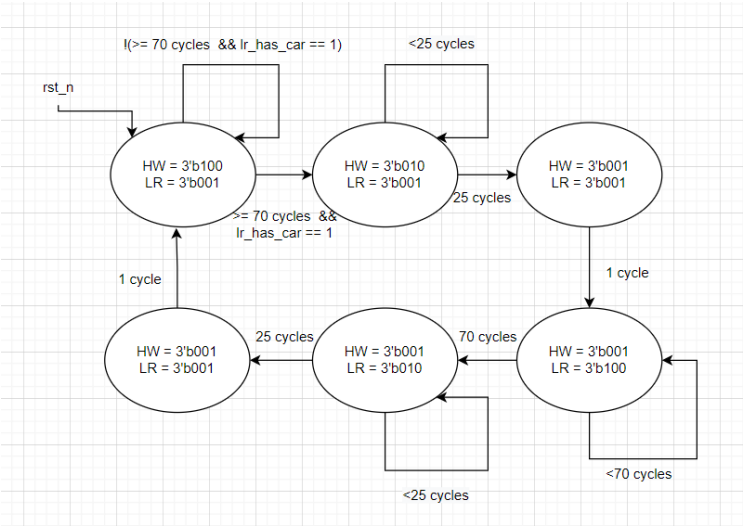
- Wave from (a match case)



- Wave form (a mismatch case)

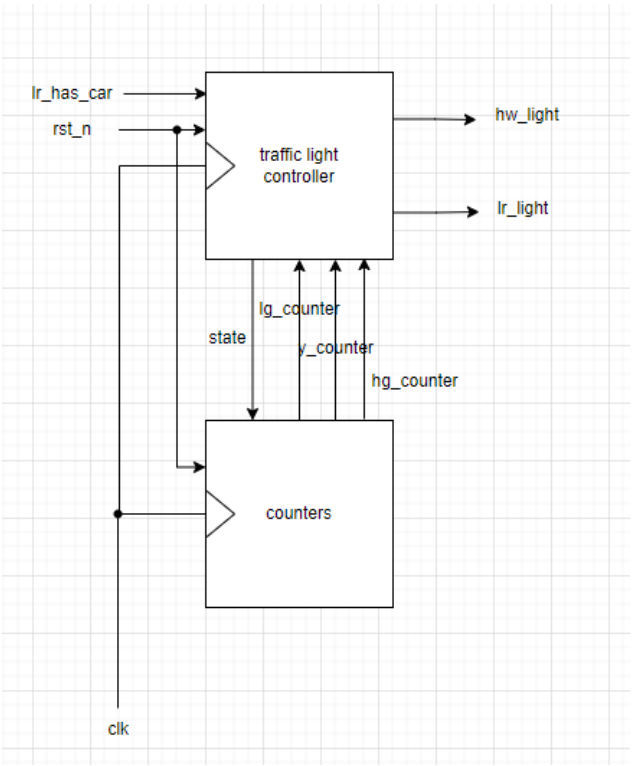
Question 2:

本題要求我們做出 traffic light controller，依照題目的要求畫出相對應的 state diagram 即可清楚如何設計。



- State diagram

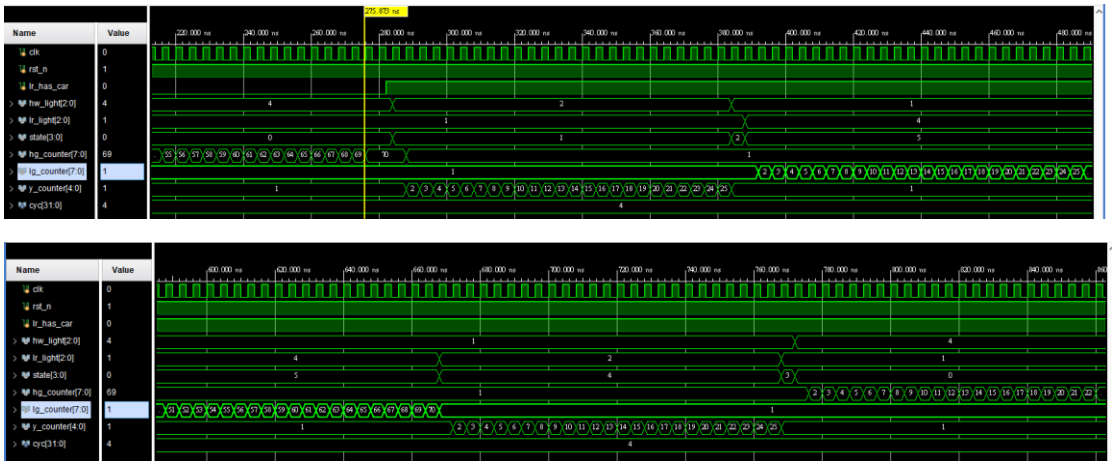
其中計算要經過幾個 cycles 我們可以用之前實現過的 counter 來實現，但要注意的是，只有當進入對應的 state 相應的 counters 才開始計時，否則其他的 state 維持 1(因為 1-70 才會是 70，0-70 會多數一個 cycles)。值得注意的是，在 HW = 3'b100_LR = 3'b001 這個 state 時，我們可以將 counter 大於>=70 的值設為 70，這樣就可以持續觸發進入下一個 state 的其中一個條件。



- Block diagram

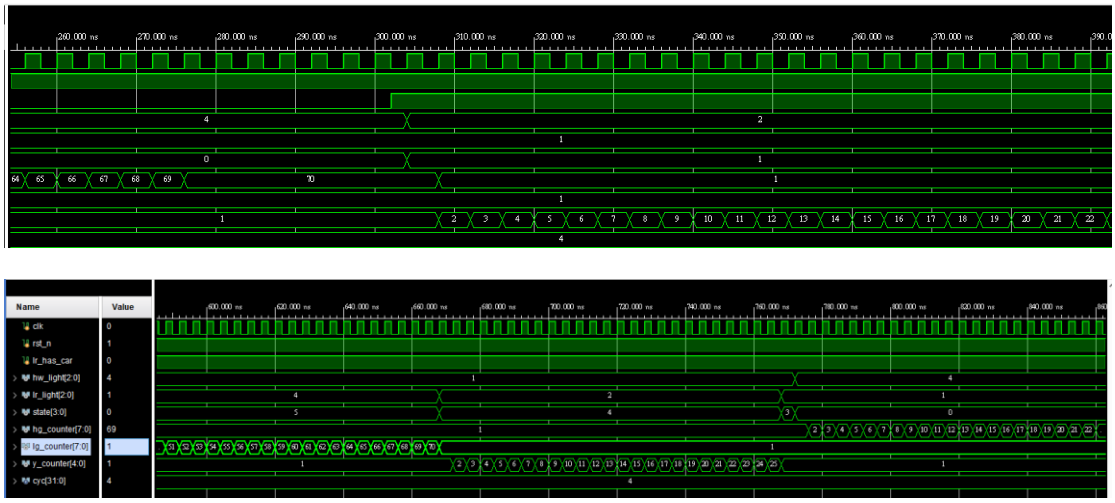
在測試的過程中，我們依照題目的要求來進行測試

1. 我們先測試 HW = 3'b100_LR = 3'b001 這個 state 在 70 個 cycle 是否可以進到下一個 state，結果會，與題目相符。之後繼續跑下去並且將 counters 以及 state 等等印出來方便我們檢查，測試結果均為正確。



- Wave form (=70cycles && lr_has_car = 1)

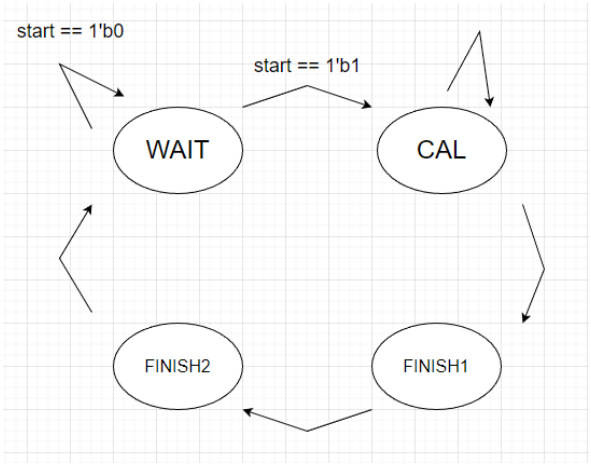
2. 我們再來測試 HW = 3'b100_LR = 3'b001 這個 state 在大於 70 個 cycle 但是 lr_has_car = 0 時會不會進到下一個 state，結果不會，與題目相符。之後繼續跑下去並且將 counters 以及 state 等等印出來方便我們檢查，測試結果均為正確。



- Wave form (>=70cycles && lr_has_car = 0)

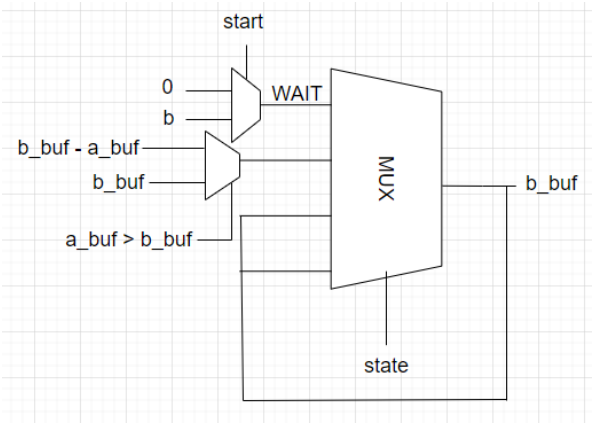
Question 3:

這題要我們運用輾轉相除法求出兩數的最大公因數，因為題目要求在完成運算後，需在展示成果的 FINISH 階段停留兩個 clock cycle，因此我們將 FINISH 這個 state 拆成兩個 state，state diagram 如下圖:

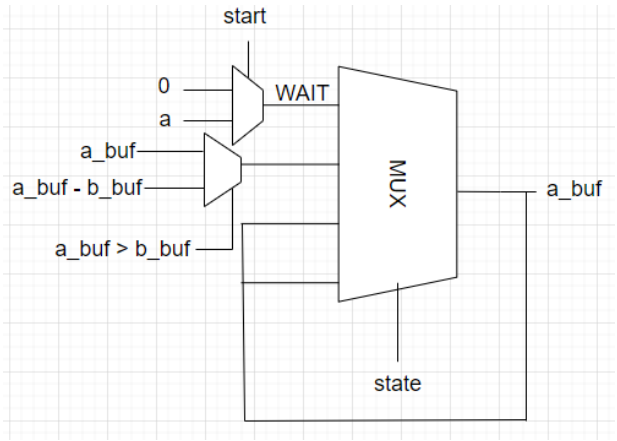


- State diagram

根據題目的運算模式，我們將減法的進行依照 clk 的時序來運作，並透過 if block 完成判斷，以下 Block diagram 中的 a_buf 以及 b_buf，分別為 a、b 在 CAL state 中存放值的 register，我們設計藉由 mux 的方式來完成運算。



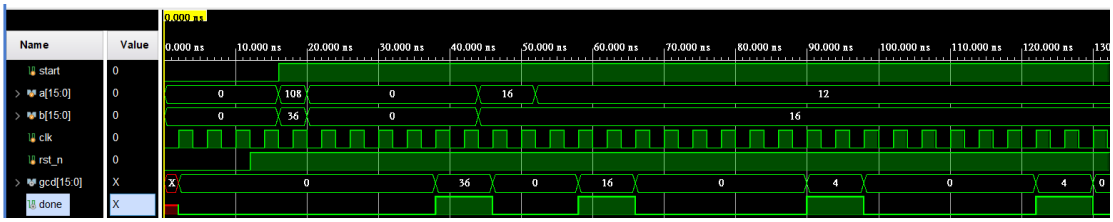
- Block diagram of a_buf



- Block diagram of b_buf

在測試中，我們將輸入 a、b 做出許多組合的變化，在成功得到正確答案

的同時，也可以完整的符合題目所要求的 clock cycle 條件，以下為部分 testbench 的波型圖:

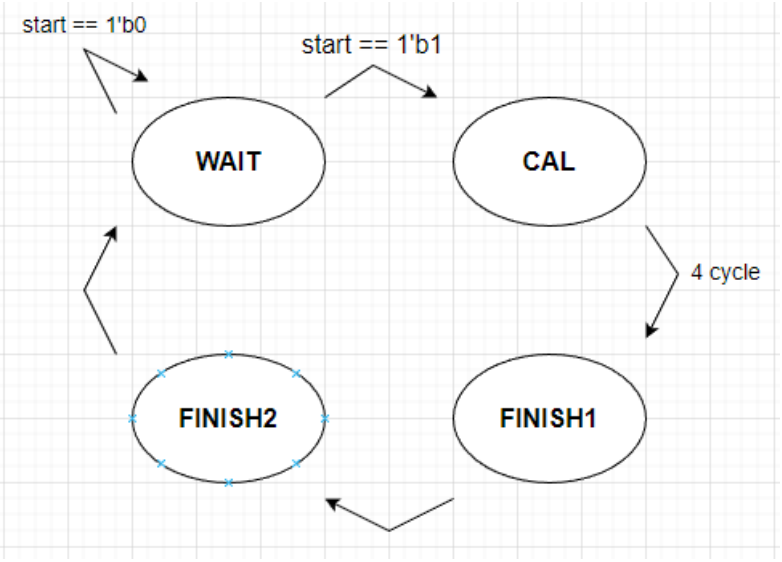


- 波形圖

Question 4:

本題的題目與一種特殊乘法算法有關，該算法名為 booth algorithm，此算法可有效提升二進位數之間乘法的效率，透過移位和加法，省去冗長的原先的進位乘法方式，需要條列出 n 項(若被除數為 n bit) 的繁複運算，透過一次判斷兩個 bit 運用二進位運算的特點，大幅降低完成所需的時間。

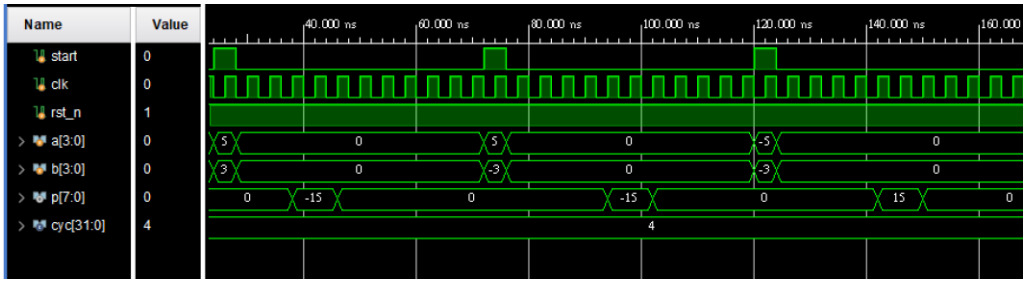
這題的 state diagram 與上一題蠻像的，需要兩個 clock cycle 來呈現運算的結果，下圖為 state diagram:



- State diagram

在實作方面，我們先將運算中會用到的 add、sub 算好，並在 clock edge 來的時候更新資訊，確認是否會推進到下一個 state。若當前的 state 為 CAL，便運用 case block，判斷現在運算到哪一個階段，因為這是兩個 4 bit 數的相乘，因此完成運算剛好需要四個 clock cycle，符合題目的要求。

在 testbench 的部分，由於此運算也可應付負數的輸入，因此我們設計了兩種正負的各種排列組合來測試我們的設計是否正確，以下為部分波型圖:



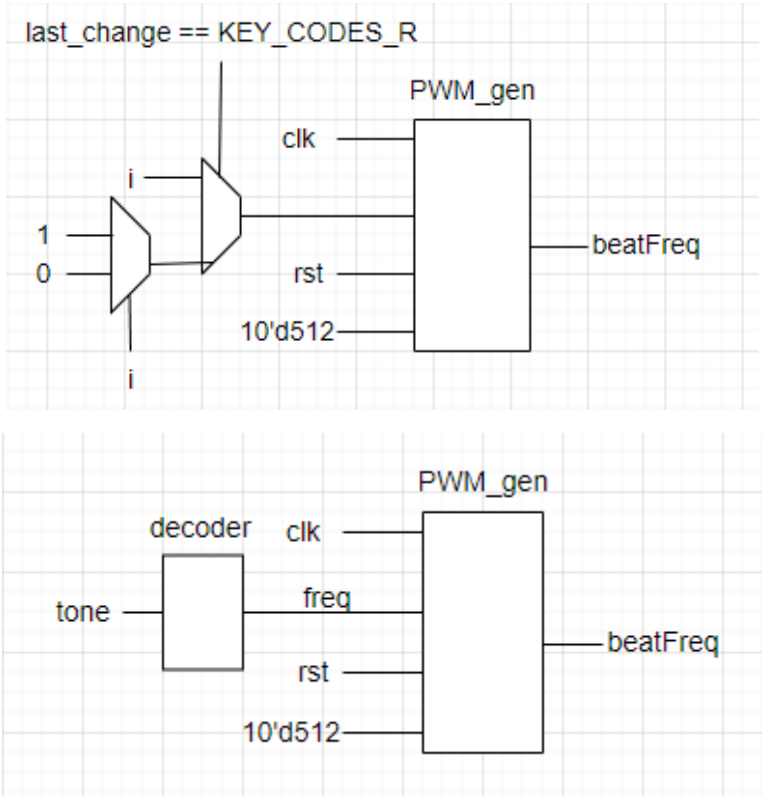
- 波形圖

FPGA Demonstration1:

這題相較於之前做過的題目較為新奇，因為除了 fpga 板的實作以外，還會運用到鍵盤輸入以及揚聲器的聲音輸出，讓設計的結果不再只是程式中的波型圖而已。

關於題目的描述，其實與先前做過的 counter 有著異曲同工之妙，皆要運用每一個 clk edge 來增加或減少 counter，不同之處這邊的 counter 是發聲的頻率，所以我們在設計方面運用了簡易的 6 bit counter，若 dir == 1'b1 則往上加，反之則向下減，並透過 decoder 將 counter 的值轉換為 c4 - c8 之間音調的頻率。

而在播放頻率 beatFreq 方面，我們仿造教授課堂中所提到的例子，運用兩個的 PWM_gen 此 module 來實作，其一用來製造 beatFreq，再者用來產生發出聲音的 pmod_1 訊號。下圖為兩個 PWM_gen module 的 block diagram:



- Block diagram of PWM_gen

在鍵盤輸入方面，我們使用了同樣是課堂中範例的 module，KeyboardDecoder，

並透過查表找到了我們所需的 Enter 鍵、W 鍵、S 鍵、R 鍵的 code 分別是 5A、1D、1B 以及 2D，再運用 if block 判斷是否有鍵被按下，且 last_change 的值為合，來得知接下來要進行何種動作。

FPGA Demonstration2:

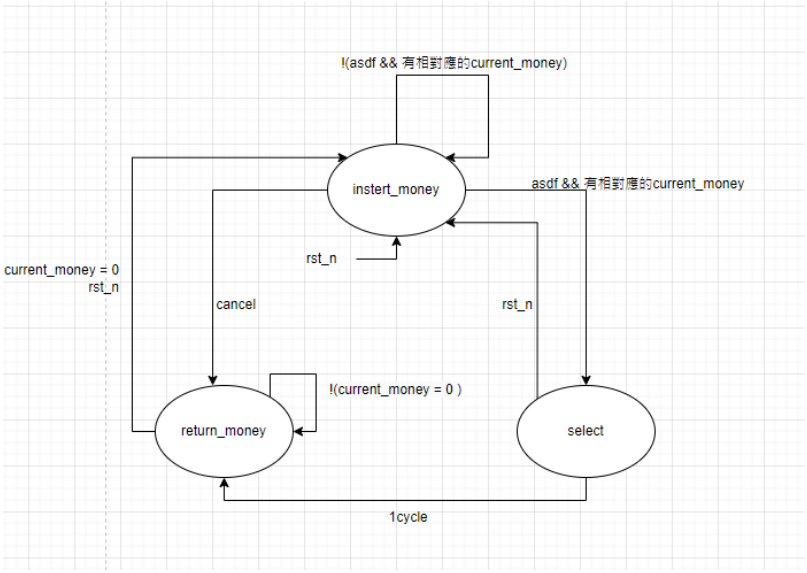
本題要求我們做出一個 vending machine，有三種錢幣，分別為 5、10、50 以及四種可以購買的飲料，分別是咖啡(80)、可樂(30)、茶(25)與水(20)。進行模式為先投錢，再選飲料最後找錢，或是先投錢，取消並且退錢。而本題保證一次只購買一種飲料。

這題大致可以區分成三種 state，分別為 insert_money、select、return_money，我們將退錢與找錢合併成一個 return_money 的 state，是因為它們的行為模式都相同。

在 insert_money 時，我們可以按按鈕持續投入 5, 10, 50 元，注意超過 100 投入的金額就為 100 元。並在投入錢且到達相對應的飲料金額時，對應的 LED 燈就要亮起。在此狀態中，我們等待 cancel 以及 a、s、d、f(購買飲料的按鈕)，以便進入 return_money 或是 select，要注意的是記得檢查是否購買的錢足夠。

在 select 時，我們將現有的金錢減去之前按的 a、s、d、f 所對應的飲料價格，然後進入 return_money。並且在此 state 中，LED 燈全部熄滅，因為不能再購買飲料了。

在 return_money 時，每隔一秒減少 5 塊錢，直到變成 0 後切換成 insert_money 。並且在此 state 中，LED 燈全部熄滅，因為退錢/找錢不能購買飲料。而每隔一秒減少 5 塊錢的實現與 advanced 2 的方法類似，就是設一個 1sec 的 counter，在其他 state 都設成 1，到了 return_money 的時候才開始，這樣就不會有在剛進這個 state 時的第一次扣錢時有不確定的時間。



- State diagram

而 reset、cancel、NT5、NT10、NT50 則可以用上個 lab 的 onepulse、

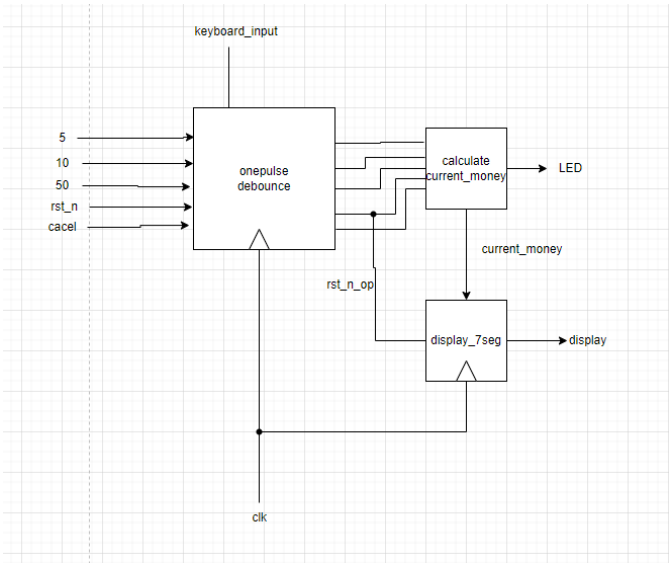
debounce 來產生訊號。顯示的方式亦然，使用一個 counter 數到 2^17 時將其視為一個 enable 訊號並且顯示。

而鍵盤的部分，首先我們需要先查到 asdf 的編碼，以及詳讀之後使用 sample code 中的 KeyboardDecoder 等的 module 以及 always blocked 裡面的內容，來進行讀取輸入。

```
parameter [8:0] KEY_CODES_a = 9'b0_0001_1100; // a => 1c
parameter [8:0] KEY_CODES_s = 9'b0_0001_1011; // s => 1b
parameter [8:0] KEY_CODES_d = 9'b0_0010_0011; // d => 23
parameter [8:0] KEY_CODES_f = 9'b0_0010_1011; // f => 2b
```

- Key_codes

七段顯示器中的部分與之前大致相同，稍微不一樣的是在當只有要輸出兩位數以及一位數時，其餘位數不用補零而是直接不顯示。所以百位數的部分就是如果是 1 就正常輸出，但當是 0 的時候就不輸出。而十位數顯示的部分則是當百位數字和十位數字是 0 時不顯示，個位數的部分正常顯示即可。



- Block diagram

Feedback

潘茗脩:

這次的 lab 讓我更了解了 verilog 與硬體之間的關係，在使用鍵盤與聲音原件上更加得心應手。不過麻煩的是，如果有 bug，常常不知道是 code 的問題還是硬體本身的問題，比如說按一下按鈕有時候會噴出兩個信號，或是七段顯示器顯示有問題。在 demo basic 時，我們的聲音一直出不來，多次嘗試接線後仍是以失敗告終，到最後重新換了一組，一次就成功了，所以這次經驗告訴我們要換一種思路，有時候不是自己的問題，是設備的問題。

而經過了上次與這次的 lab，我對寫 verilog 也愈發熟練，只要 codeing style 得當，可以以非常有效率的方式寫出 code，並且 debug。我也學到了除了要將 module 分開管理外，變數也可以拆成多個 always block 處理，這樣可

以避免寫出非常難以閱讀並且 debug 的 code。

謝諺緯:

這次的 lab 相當有趣也對我來說特別有意義，雖然之前的 lab 也能充分發揮 fpga 板的運算功力，達成許多的任務，但那些也都僅僅止於板子上面小面板的顯示，較看不出實際的功用。但在這次 lab 之後，終於來到了我滿心期待的周邊硬體實作環節，不管是連接鍵盤輸入數字，抑或是運用揚聲器撥放小蘋果歌曲，這次的實作為我打開了全新的一扇邏輯設計實驗的大門，找到更貼近生活的學習方式。

而在 verilog 方面，這次的練習也讓我更熟悉 state 之間的轉換，以及相對應的 clock cycle 要求，也更會注意到，可以將 always block 分成很多個陣隊需要的變數來撰寫，如此不僅可以更為明瞭，也能在 debug 時更加省力。