

第七組

黎佑廷 105030009

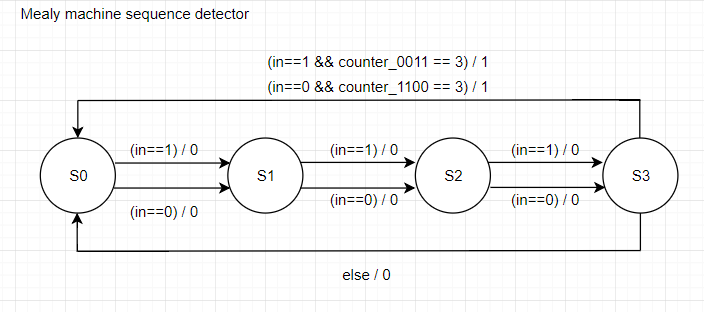
郭家瑋 105030015

lab5

Verilog Question 1:

Mealy machine sequence detector

* State-transition diagram:



當 state == S0時，不管怎樣都是輸出0，而且下一個state一定是S1，只是當in == 1時，我將counter\_1100 + 1，當in==0時，我將counter\_0011 + 1。

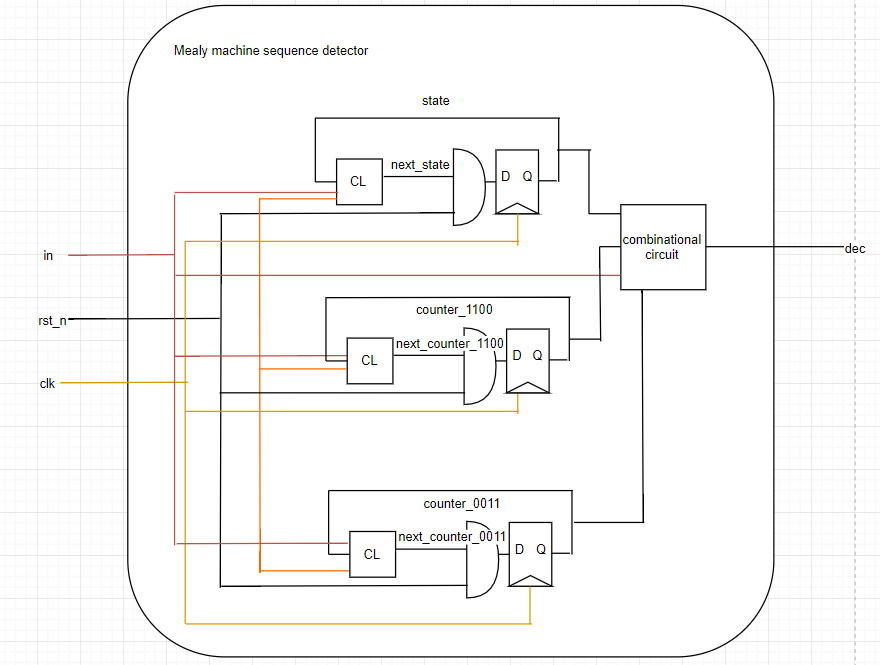
當state == S1 時，也是不管怎樣都輸出0，而且下一個state一定是S2，只是當in == 1時，我將counter\_1100 + 1，當 in == 0時，我將counter\_0011 + 1 。

當state == S2 時，也是不管怎樣都輸出0，而且下一個state一定是S3，只是當in == 0時，我將counter\_1100 + 1 ，當in == 1時，我將counter\_0011 + 1。

當state == S3 時，如果in == 1而且 counter\_0011 == 3，代表已經偵測到0011這個pattern，所以輸出1，如果in==0而且counter\_1100 == 3代表已經偵測到1100這個pattern，其他的狀況則是輸出0，下一個state回到S0以繼續偵測接下來的pattern。

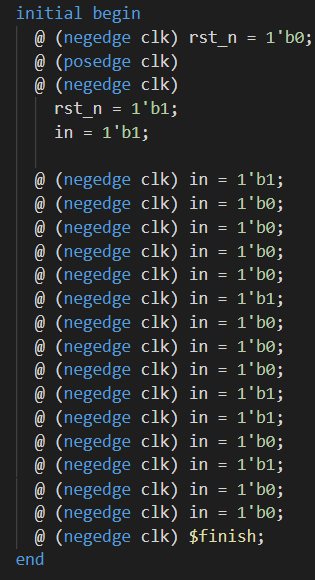
因為無論input為何，state的變換都是S0->S1->S2->S3所以每四個bit都會重新detect一次。

* Block Diagram



這題我所使用的驗證方法是給予跟老師投影片一樣的input，看看出來的waveform有沒有跟投影片上的一樣。

Code的部分如下(給予1001\_0010\_1001\_0100的input):



* Waveform



可以發現這個waveform跟老師投影片的一模一樣

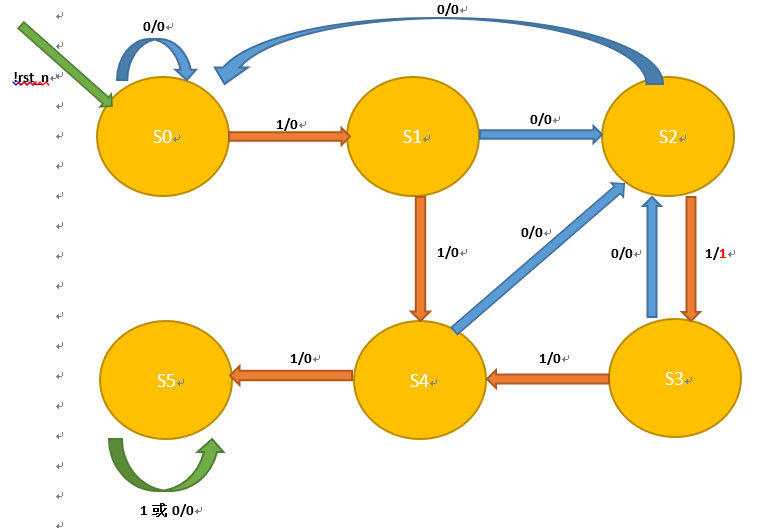


Verilog Question 2

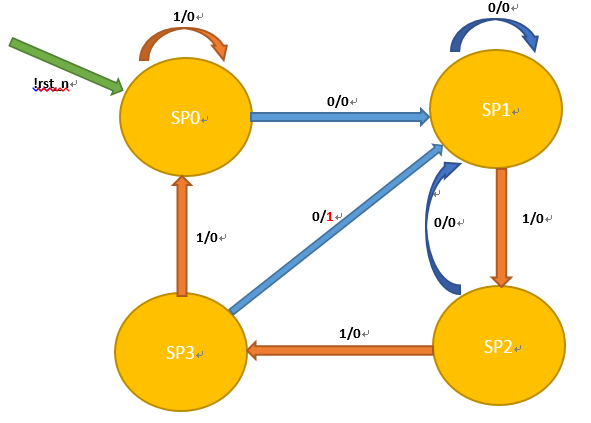
Traffic light controller

* State Transition Diagram

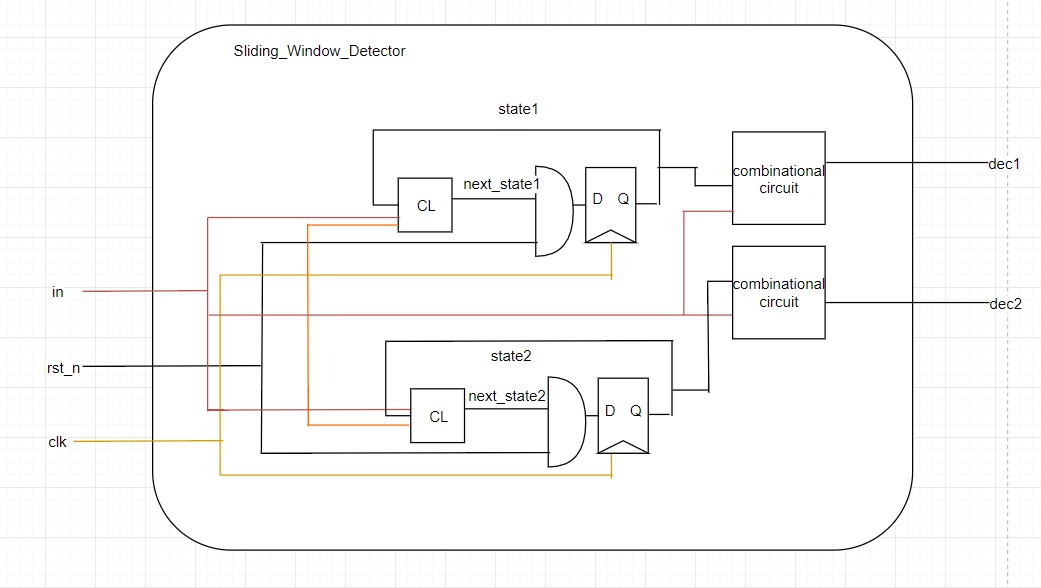
For Dec1



For Dec2



* Block\_diagram



Dec1：

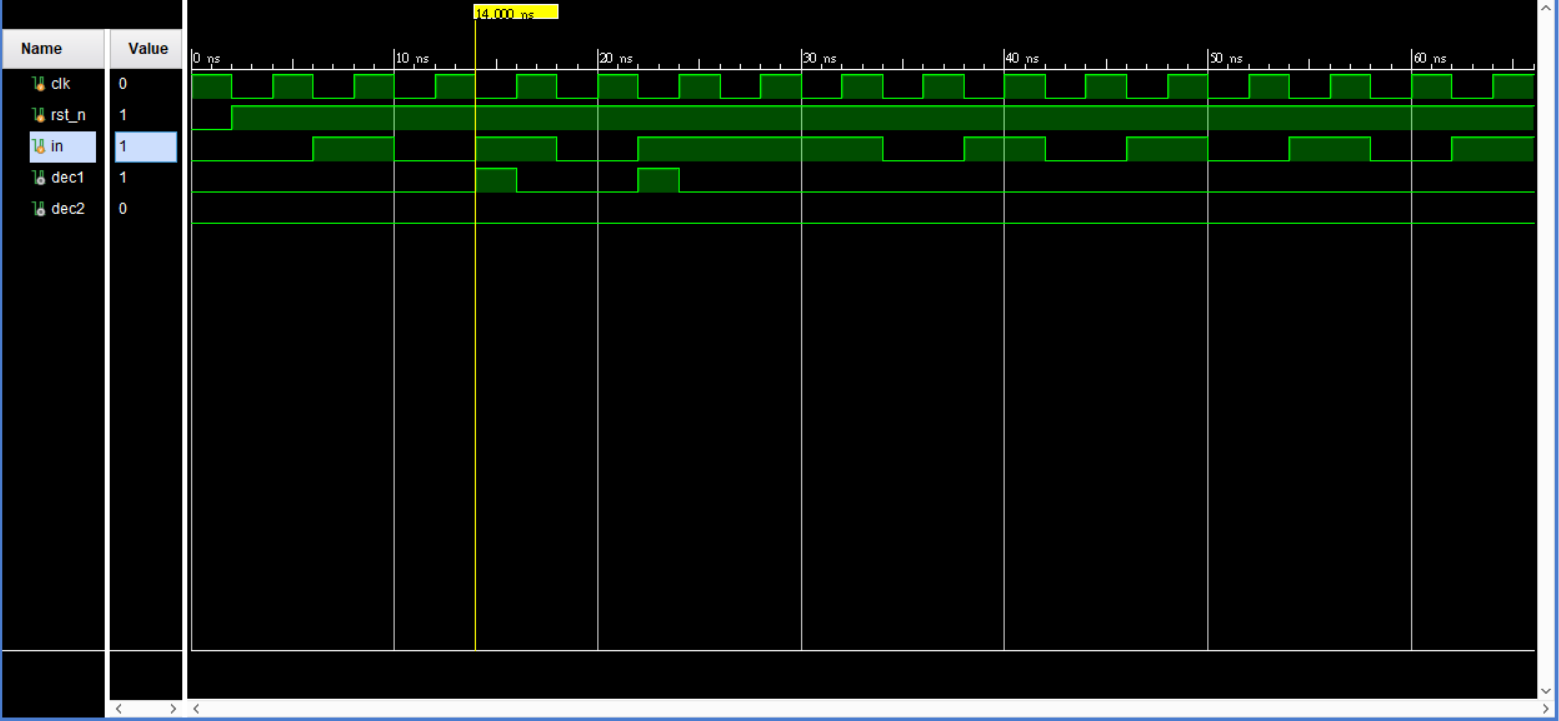
當接收到!RESET訊號會進入S0，此時要等待一個1才能繼續往下一個state，因此若in是0，next\_state會是S0，若in是1，next\_state會是S1。到達S1後，此時等待一個0才能往下一個state移動，因此若in是1，next\_state會進入S4，若in是0，next\_state會是S2。到達S2後，此時等待一個1即可完成一個101的訊號檢測並且output 1，因此若in為1，next\_state會是S3，並且output 1，若in為0，等於出現了「100」的訊號，此時要重新回到S0等待一個1。到達S3後，表示剛剛出現了一個「101」的訊號，因為訊號可以接連著上一次的pattern做檢測，例如「10101」會有兩次的訊號pattern檢測出來。因此，在S3，若in為0，要回到S2，因為到達S3代表剛剛出現過一個in=1，因此只需再等待「01」的出現即可output=1，若in為1，next\_state為S4。到達S4後，代表連續出現了兩個1，此時要判斷in是否為1，若in為1，代表連續出現了三個1，進入S5此後不管出現什麼訊號的pattern都要output 0，若是in=0，代表出現了「110」，此時只需再出現一個1即可output1，因此回到S2。

Dec2：

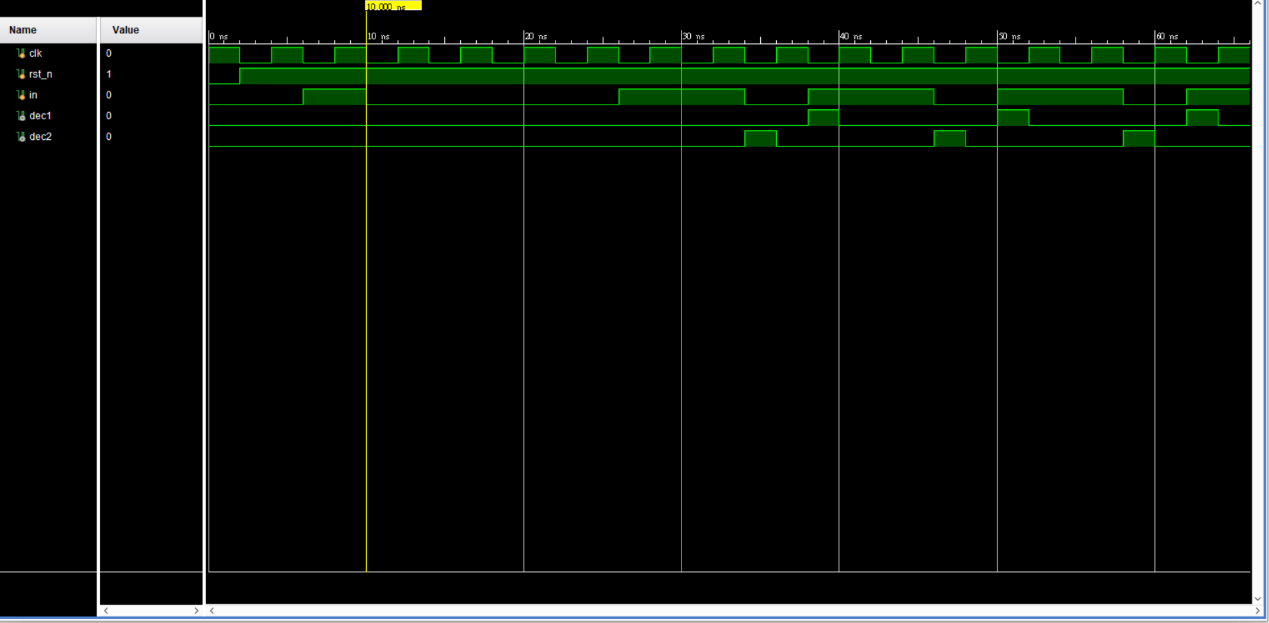
當接收到!RESET訊號會進入SP0，此時要等待一個0才能繼續往下一個state，因此若in=0，next\_state會進入SP1，若in=1，next\_state繼續在SP0。到達SP1後，此時需要一個1才能進入下一個state，因此若in=1，next\_state會是SP2，若in=0，next\_state會是SP1。到達SP2後，代表已有「01」，因此，此時需要1才能進入下一state，因此若in=1，next\_state是SP3，若in=0，next\_state會是SP1，要重新等待「0110」的第一個1。到達SP3後，代表已有「011」，此時只需一個0就可output 1，因此若in=0，next\_state會是SP1，已有一個0的情況，並且output 1，若in=1，要重新等待一個新的0。

Testbench寫法是模仿投影片中的input並且比較標準答案與我的波形圖是否相等。

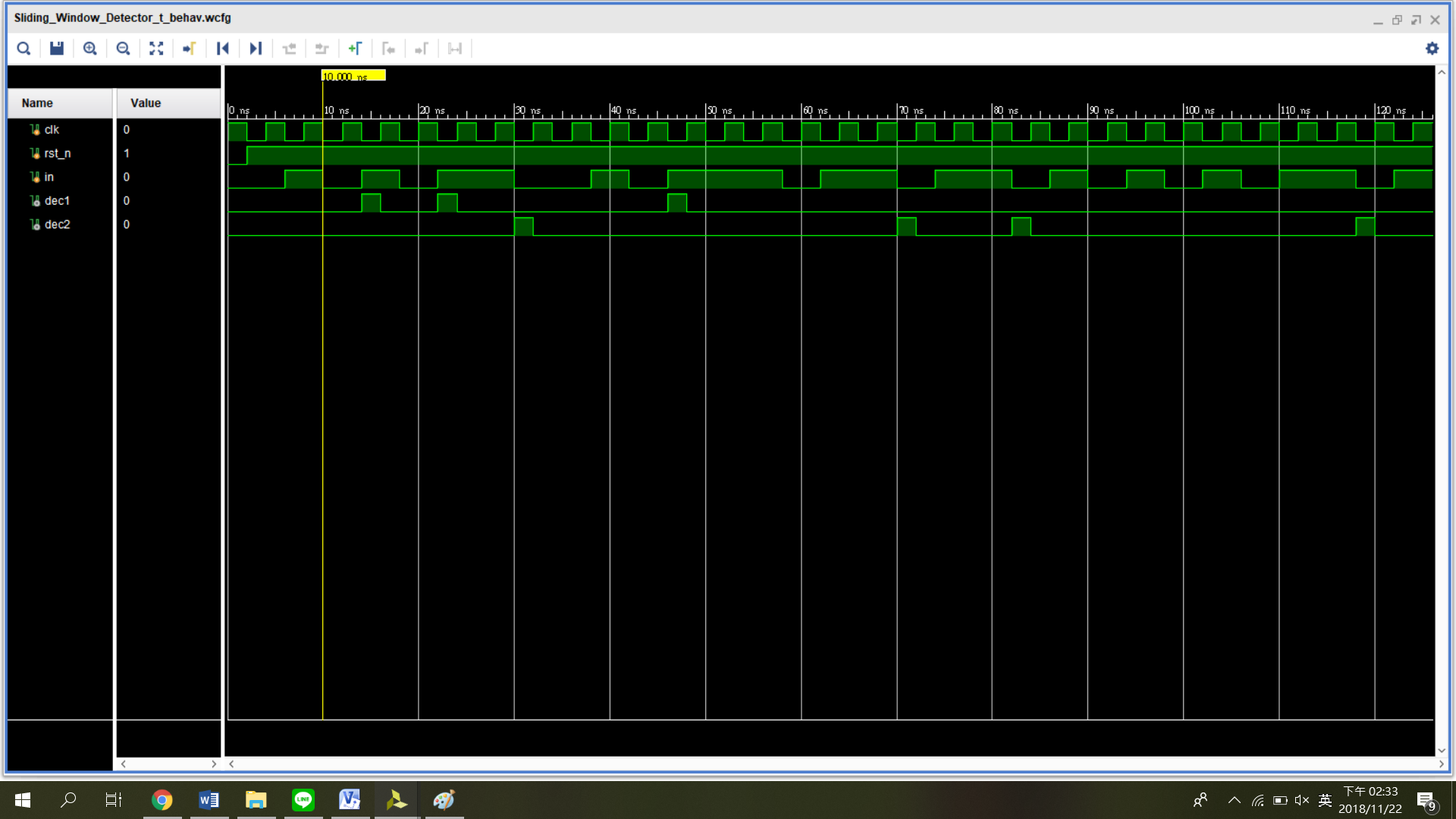
(檢測dec1)



(檢測dec2)



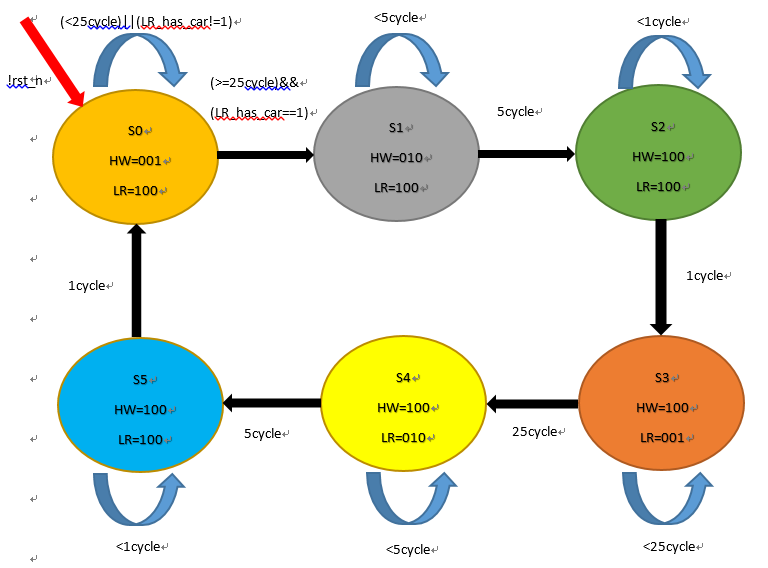
(general test)



Verilog Question 3

Traffic light controller

* State transition graph



這題實際的做法，若是像下方的方式實作，就會顯得太多state而造成不易看懂全貌：

共25cycle

共5cycle

共25cycle

共5cycle

這樣的做法會用到共61個state，不但寫起來很複雜難寫，並且也不易看清全貌。因為同樣的計數周期的功能運用多次，因此我把這樣的功能做成一個module，若沒有達到所要求的cycle次數，就讓done=0，繼續在同一個state。若達到所要求的cycle次數，就讓done=1，可以進入下一個state。這樣的實作方式，只需共6個state。

Transition graph如下：

**!done**

**!done**||(LR\_has\_car!=1)

!rst\_n

**done**&&

(LR\_has\_car==1)

**done**

**!done**

**done**

**done**

**!done**

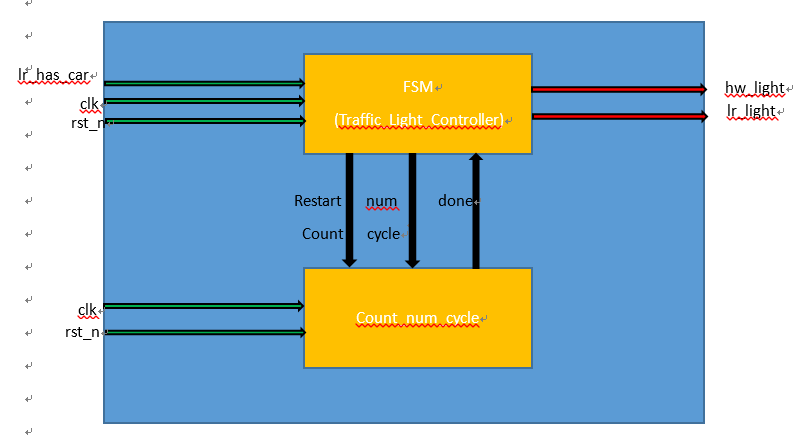
**!done**

**!done**

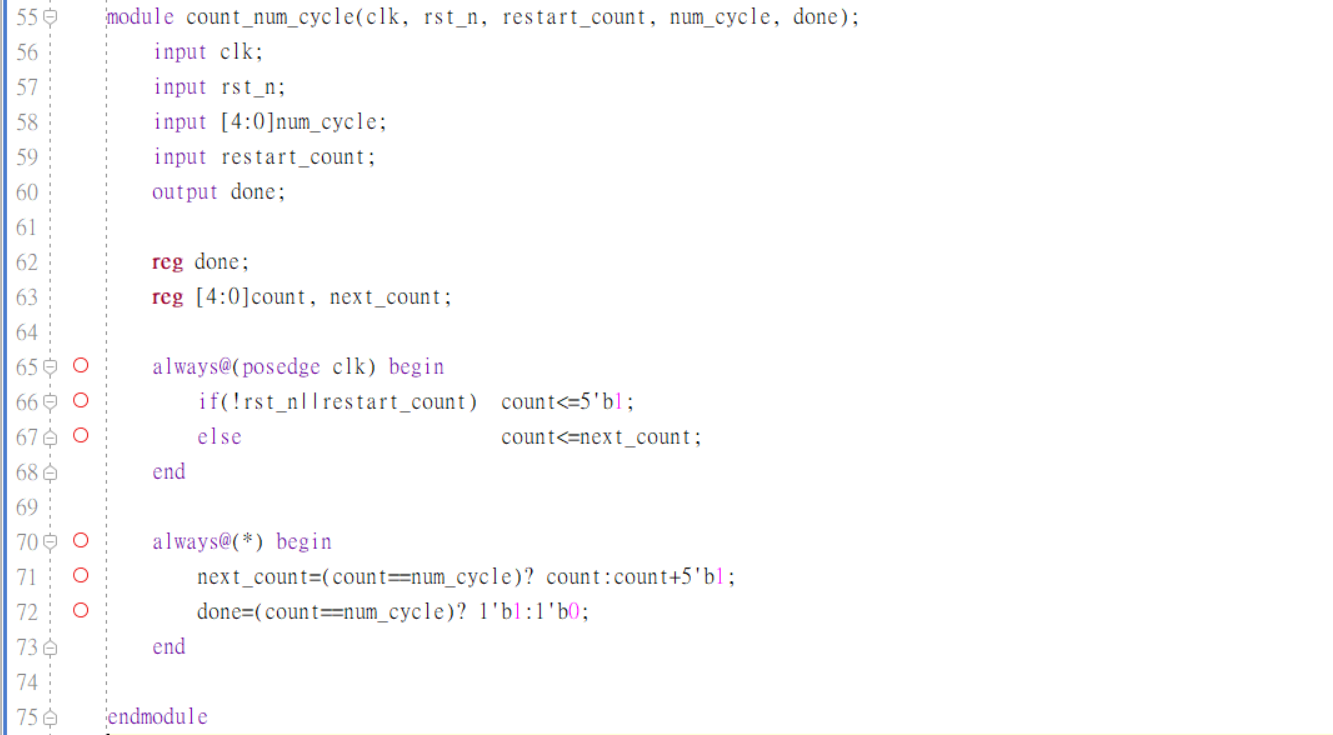
**done**

**done**

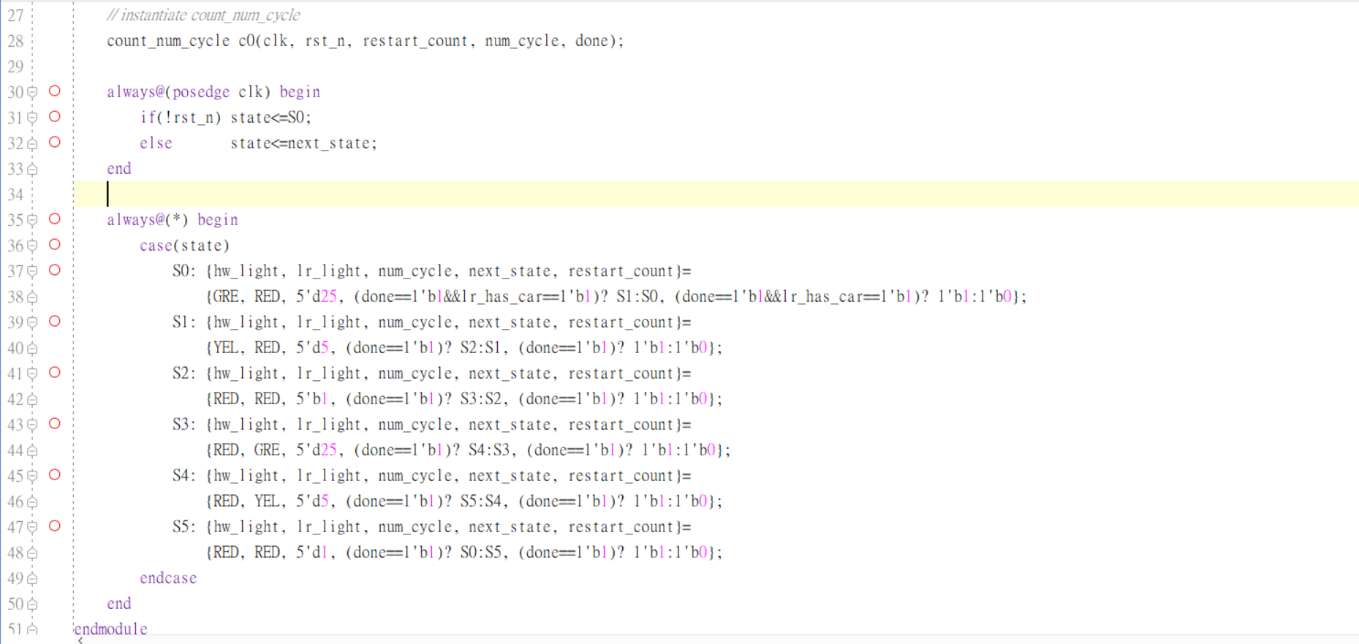
Block diagram如下：



關於計數cycle的module實作如下



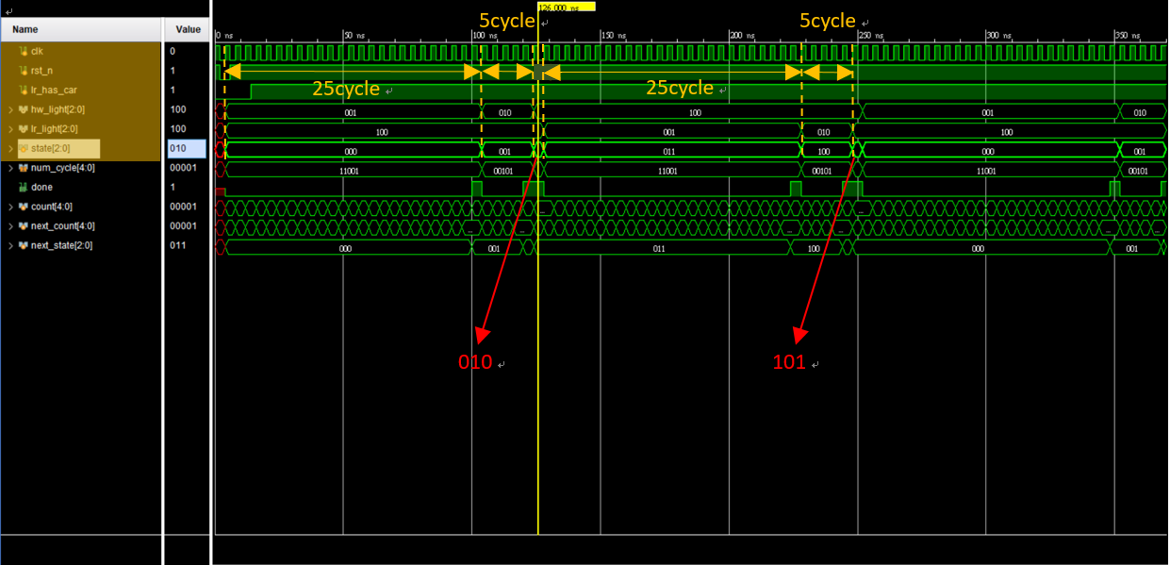
因此，state transition長的會像這樣。



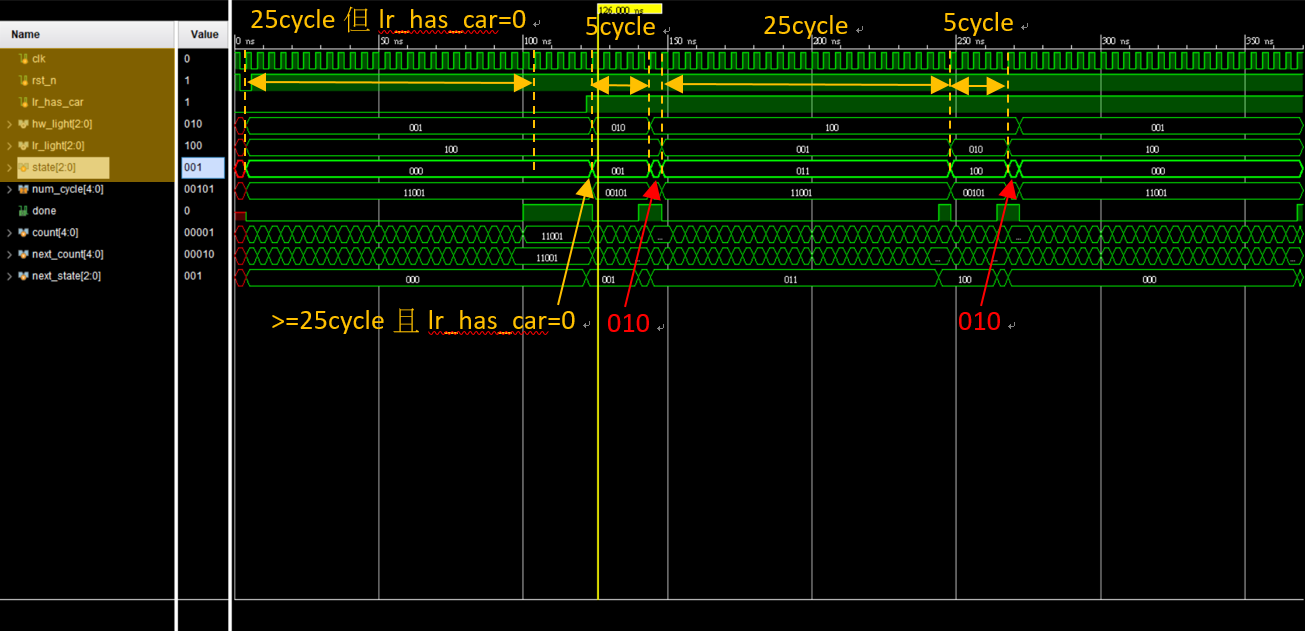
因為是Moore’s Machine，故output只決定於state，在不同的state就會給出該state的output (hw\_light、lr\_light)。因為在不同state需要的cycle數不同，因此會需要帶入num\_cycle給count\_num\_cycle。Next\_state會取決於是否done，在state S0中，除了要done外，還需要lr\_has\_car，才能決定next\_state。最後，因count\_num\_cycle在不同的state中計數完必須要歸零，才能在下一個state中重新計算，所以要給出restart\_count給count\_num\_cycle。restart\_count的判斷依與next\_state的判斷依據相同，若會進入下一個不同的state，就要將restart\_count設為1；若無，就設restart\_count=0。

testBench的寫法為模仿spec示範的波形圖來檢測state轉換與output

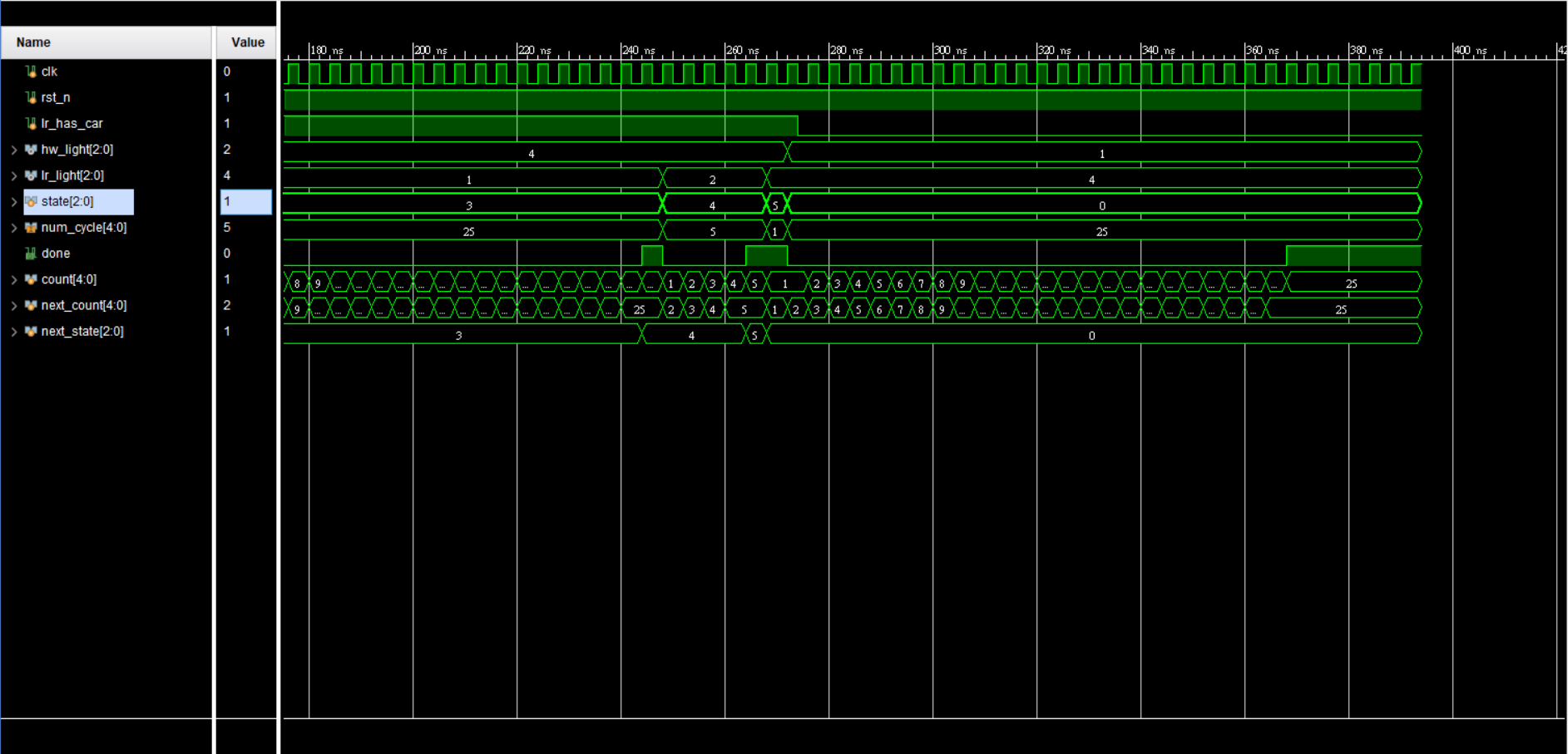
(test1) 一般情況



(test2) state0等了25cycle但lr\_has\_car=0情況



(test3) state0跑到state5後，回到state0但lr\_has\_car一直是0

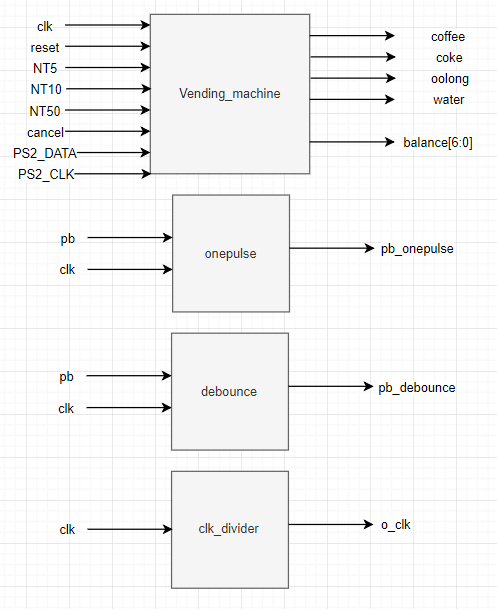


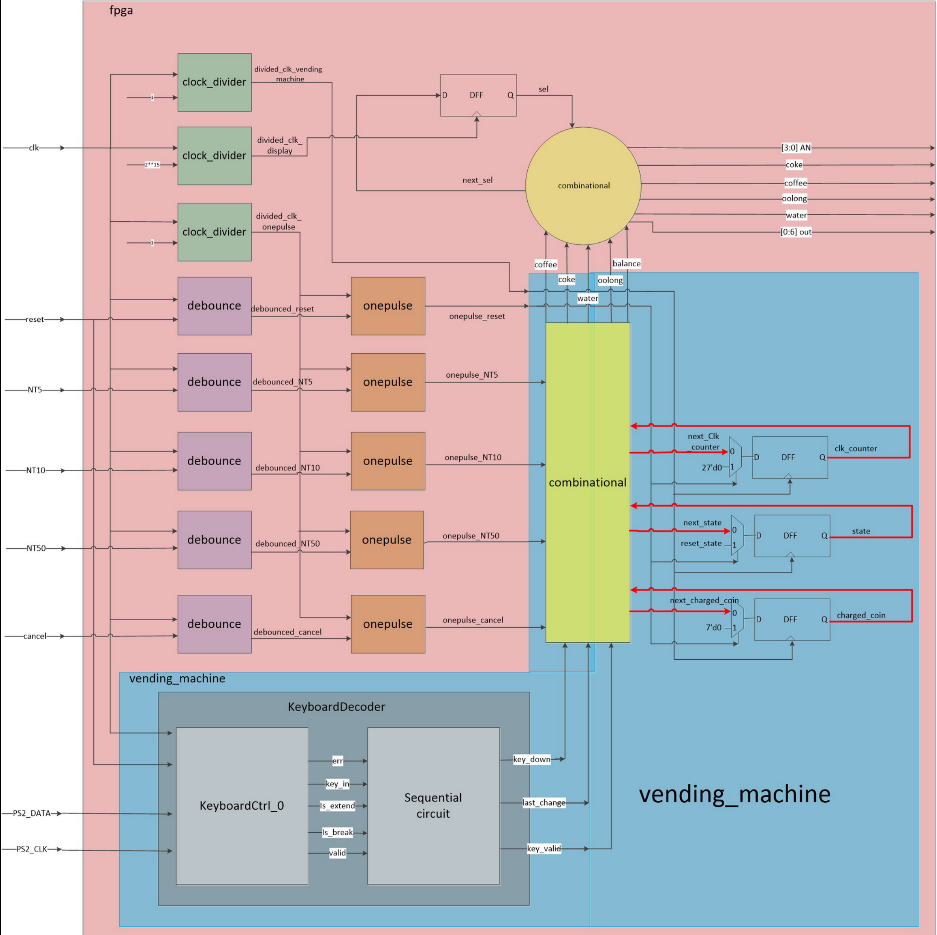
Verilog Question 4

FPGA

* Block diagram

在top module：fpga中包含了以下小module，為了簡化整體的block diagram，先放上小module本身的port，在整體的block diagram就省略小module的port。





* 說明

關於clock由於不同情況下會需要不一樣的clock來進行運算，所以我用clock\_divider做出了三個不一樣的clock:

divided\_clk\_vending\_machine : 用於vending\_machine這個module，因為需要接鍵盤，所子我將它除1，其實就是板子的clock，100MHZ(原本我為了能讓找錢的速度是一秒五塊，所以把頻率調成1HZ，但結果發現1HZ時按鍵盤沒反應，所以後來用了100MHZ，至於怎麼讓它一秒找一次錢，這個我在下面會詳述)。

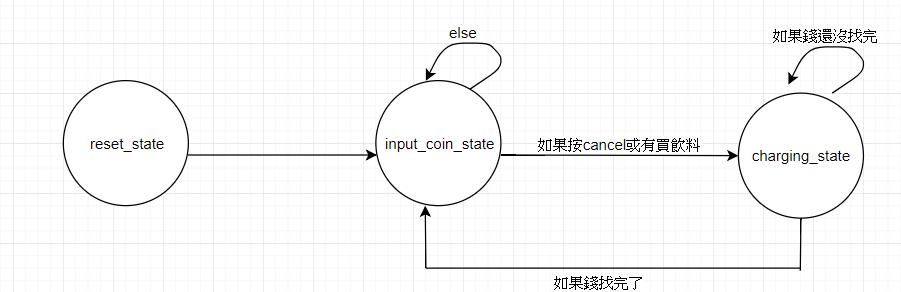
divided\_clk\_onepulse : 用於onepulse這個module上，它的頻率必須跟divided\_clk\_vending\_machine一樣，這樣才不會發生在vending\_machine在遇到posedge clk前onepulse的訊號就變回0的尷尬情形。

divided\_clk\_display : 用於讓四個七段顯示器分別通電的時間週期。我將內建clock乘以215。若四個七段顯示器通電時間太短則來不及通電，無法顯示正確的數字，若太長則無法造成明顯的視覺暫留。

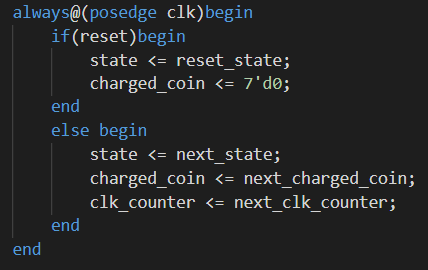
clk設定好後，處裡按鈕的去除雜訊，按鈕包括reset、cancel、NT5、NT10、NT50。debounce module包含四個DFF相連，DFF的clk都使用BASYS內建clk，即100MHz。當四個DFF output都是1代表準確的按下，輸出debounced\_reset、debounced\_cancel、NT5、NT10、NT50。

關於vending\_machine這個module，它的主要任務就是負責販賣機的相關運算(如找錢、算錢等)，並將output(coffee、coke、oolong、water、balance)傳給top module，詳細解釋如下:

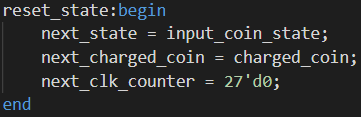
* State transition diagram



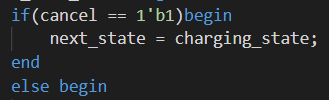
在偵測到reset訊號時，我將charged\_coin(這個變數是指買家投入的金額)設成0、state設給reset\_state，如果沒有偵測到reset訊號，那state、charged\_coin、clk\_counter就會分別等於conbinational block計算出來的next\_state、next\_charged\_coin、next\_clk\_counter(關於clk\_counter我在下面會詳述)，code如下圖所示:



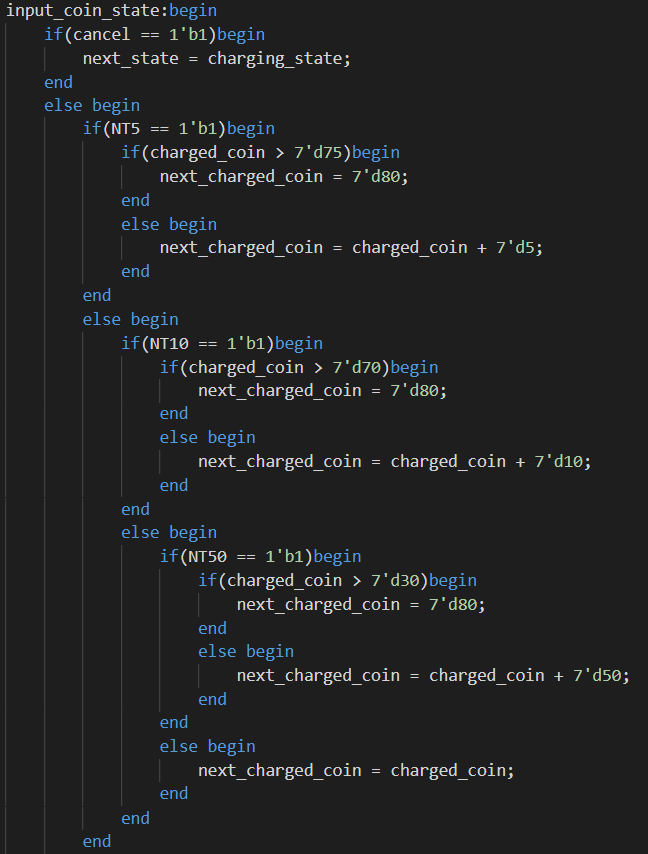
在reset\_state時，下一個state設給input\_coin\_state以等待start訊號，賣家投入金額(charged\_coin)保持不動，clk\_counter初始成0，code如下圖所示:



在input\_coin\_state時，主要的任務就是計算買家投入的金額，以及買飲料時所需要的運算(像是扣錢)，如果按下cancel按鍵，那下一個state就會進入charging\_state以進行找錢相關的運算，code如下圖所示:



如果cancel鍵沒有被按下，那我就計算買家目前投入了多少金額，並看看目前的金額是否足以買飲料，如果足夠就讓相對應的output變成1使LED燈發亮(例如買家投入55元，這時他可以買coffee、coke、oolong、water，所以這四個變數都會被設成1，以便讓LED燈發亮)，此外，input\_coin\_state也會對買家所買的飲料進行餘額運算(例如買家投入80元，買了55元的咖啡剩25元)，code如下所示:

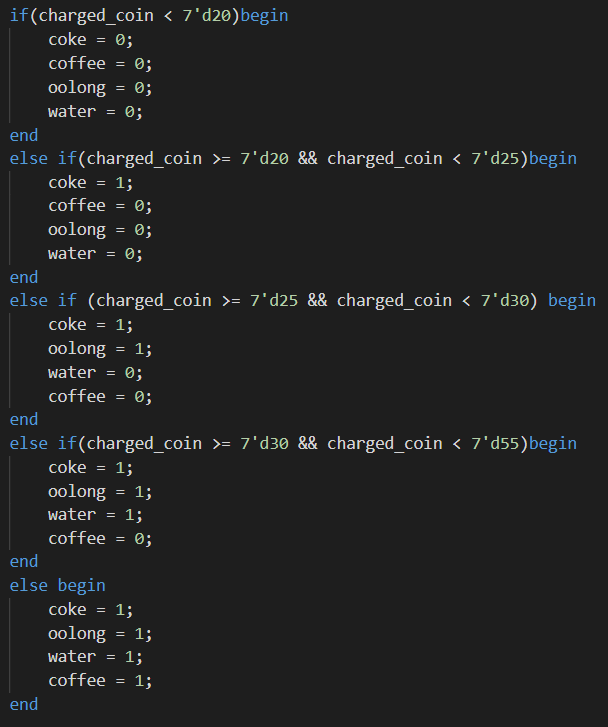


如果沒投錢，charged\_coin就保持不變

如果投入五十元，買家投入的金額(charged\_coin)就加50，且總金額不會超過80元

如果投入十元，買家投入的金額(charged\_coin)就加10，且總金額不會超過80元

如果投入五元，買家投入的金額(charged\_coin)就加5，且總金額不會超過80元



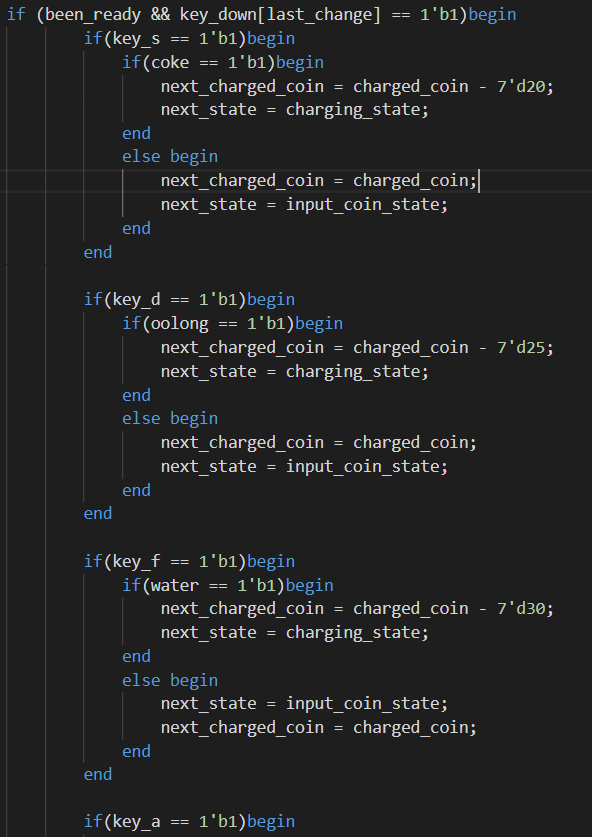
如果投入金額在大於55，那所有飲料都能買!

如果投入金額在30~54之間，可以買可樂、烏龍茶和水!

如果投入金額在25~29之間，可以買可樂以及烏龍茶!

如果投入金額在20~24之間，可以買可樂!

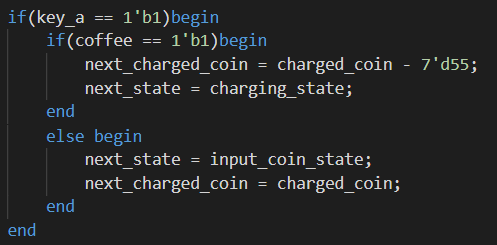
如果投入金額小於20，什麼飲料都沒辦法買



買水的相關運算

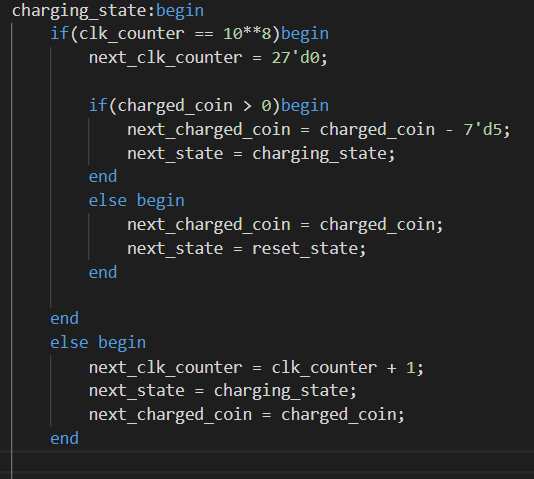
買烏龍茶的相關運算

如果買家按下S買可樂，先檢查他所投入金額是否足夠，如果夠的話就將charged\_coin減去可樂價錢，並且讓下一個state是charging\_state，以便進行找錢的相關運算; 如果錢不夠，那按S就什麼都不會發生



買咖啡的相關運算

Charging\_state的主要任務就是找錢，但因為一秒只找一次，而clk的頻率是100MHZ，所以我設了一個clk\_counter，它會在posedge時加1，如果加到10\*\*8，那就代表已經經過了一秒，此時賣販機就會找五元出去，所剩的金額也就扣五元，如果錢找完了那就會回到reset\_state重新開始，code如下圖所示:



這個部分主要是處理如何將買家所投入的金額以及coffee、coke、oolong、water四個LED燈正確的顯示在FPGA上，

* 心得(BY郭家瑋)

這次lab的pattern檢測不是很難，主要就是FSM。紅綠燈的問題也不算難，主要就是要應用factored FSM，否則會跑出一堆state。這次lab學系到比較多的主要在basic，學會了如何裝鍵盤與喇叭，雖然蠻複雜的，debug也因module眾多而困難，但聲音跑出來非常有成就感。

* 心得(BY黎佑廷)

我覺得這次lab的第一題不難，跟lab4的第一題很相似，所以寫起來比較沒那麼陌生，但debug花了許多的時間，因為當時也在想圖片裡的問題:



想了很久覺得很奇怪，後來去ilms上看時發現有人問了一樣的問題，下次在打之前應該先看看討論區才不會花太多時間debug，至於，fpga的部分，由於對鍵盤的使用還不太熟悉，所以花了很多時間debug。

**成員:**

黎佑廷: 負責第1題以及fpga

郭家瑋: 負責第2題以及第3題