**1. Implement key board using the left-hand-side keyboard (inside the black blocks).**

**1.1 Press 0/1/2/3/4/5/6/7/8/9 and show them in the seven-segment display. When a new number is pressed, the previous number is refreshed and overwritten.**

**1.2 Press a/s/m (addition/subtraction/multiplication) and show them in the seven-segment display as your own defined A/S/M pattern. When you press “Enter”, refresh (turn off) the seven-segment display.**

**1. Specification**

module LAB9\_1\_top(

output [7:0] cathode, // seven segment display

inout PS2\_DATA, // (PIC24) USB HID Host

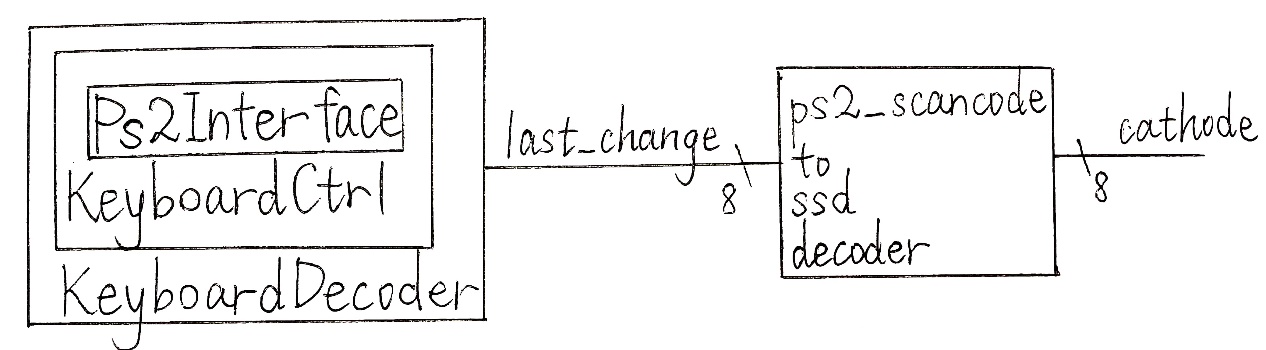
inout PS2\_CLK, // (PIC24) USB HID Host

input rst\_n, // active low reset

input clk // 100MHz global clock

);

**2. Block Diagram**

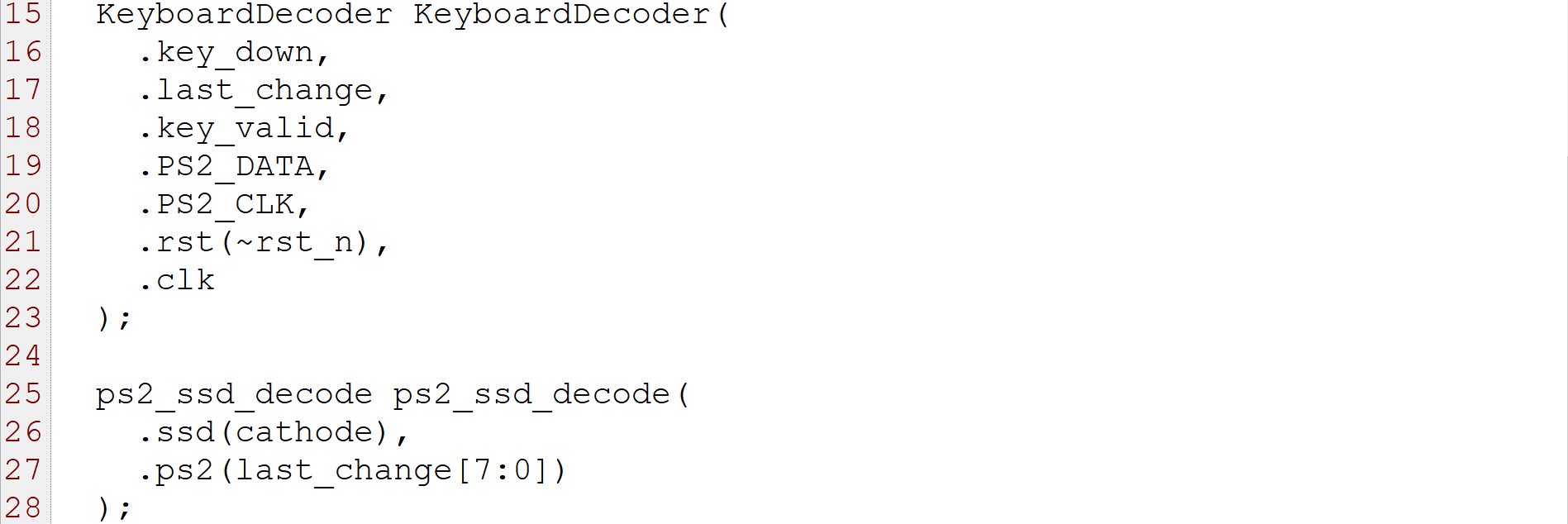


根據老師的課堂講解，KeyboardDecoder.v的其中一個output last\_change的訊號包含了被按下的按鍵的scan-code。因此，只要將last\_change的訊號連接到一個解碼器，將scan-code轉換成SSD即可。

**3. Finite state machine**

無

**4. Implement**

本題要讓鍵盤上的數字鍵被按下後，顯示對應的數字在七段顯示器上。檢查老師提供的KeyboardDecoder.v，發現可以使用其output last\_change來判斷是哪一個按鍵被按下。因為當一個按鍵被按下之後，last\_change的值就會更新成那個按鍵的scan-code，所以只要將last\_change連接到一個scan-code轉SSD的解碼器即可。下圖是LAB9\_1\_TOP.v的內容。

其中，解碼器ps2\_ssd\_decode.v的內容是case statement。將數字0到9與enter等可能的情況一一列出，構成look-up table。其code就只是把所有可能的情況列出來而已，詳參.v檔。

**5. Conclusion**

第一題很簡單，沒有遇到困難。我猜想老師會這麼安排第一題，旨是為了要確認每一位同學都能順利的引入老師提供的IP，並確保正常運作。

在操作的過程中，可以學習如何使用別人寫好的IP。透過本題，可以了解KeyboardDecoder.v的last\_change訊號代表的意義，並練習使用該訊號來取得被按下的按鍵等資訊。

**2. Implement a single digit decimal adder using the left-hand-side keyboard (inside the black blocks). Use the key board as the input and display the results on the 7-segment display (The first two digits are the addend/augend, and the last two digits are the sum).**

**1. Specification**

module LAB9\_2\_top(

output [7:0] cathode, // seven segment display

output [3:0] anode, // seven segment display

inout PS2\_DATA, // (PIC24) USB HID Host

inout PS2\_CLK, // (PIC24) USB HID Host

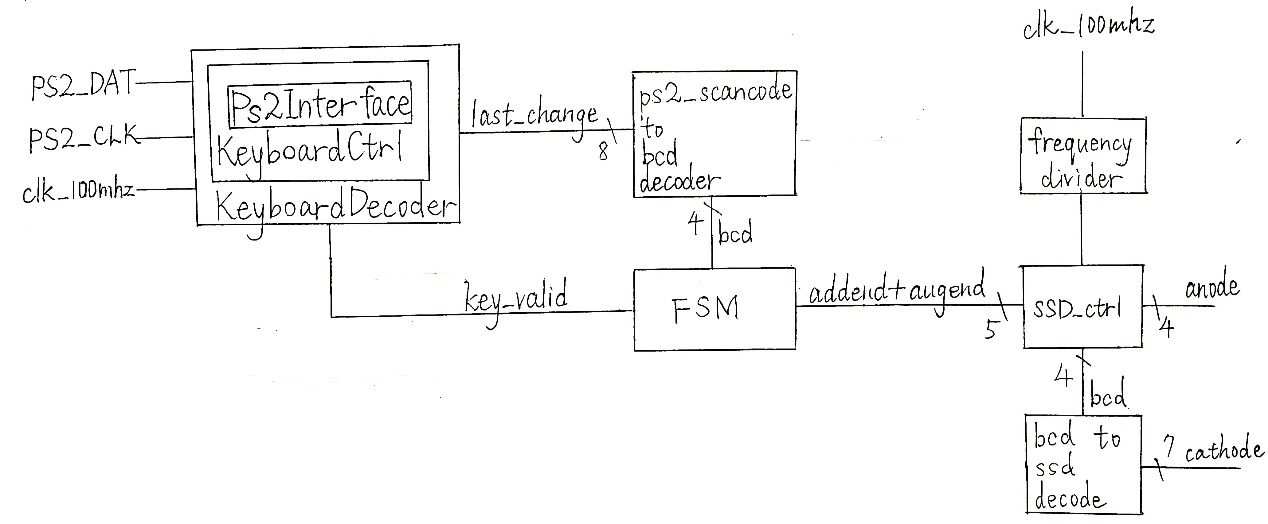
input rst\_n, // active low reset

input clk // 100MHz global clock

);

較第一題只多了anode，讓七段顯示器看起來能同時顯示四個不同的數字。

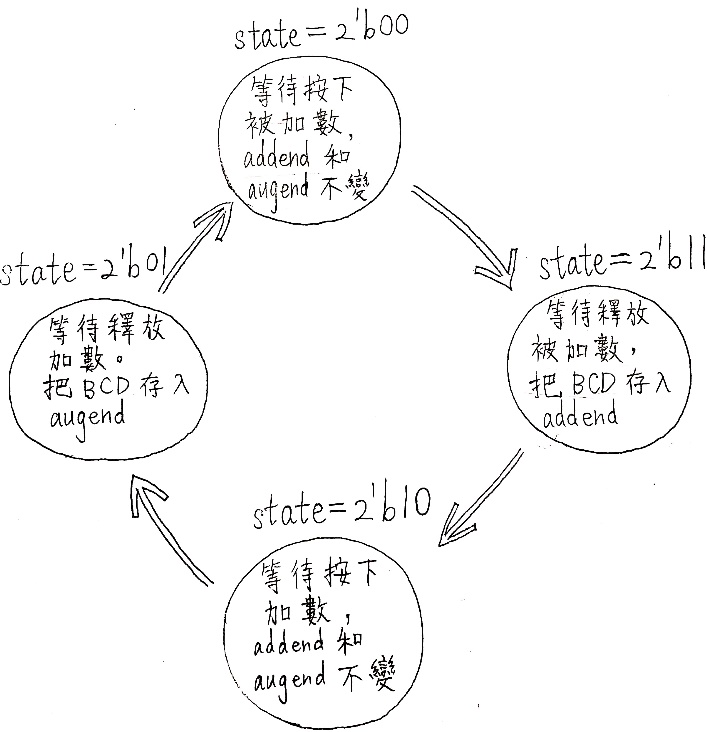
**2. Block Diagram**



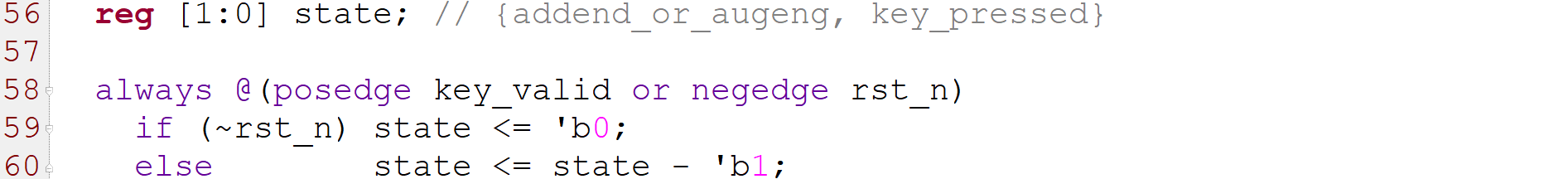
從block diabram可以看出，本題建立在上一題的基礎之上。上一題，last\_change直接使用look-up table轉換成七段顯示器cathode的樣式。但在本題，由於要將輸入的數字相加，因此需要先轉換成可以被加法器理解的BCD形式。addend與augend透過加法器相加後，再轉換成七段顯示器cathode的樣式。FSM的CLK連接key\_valid作為時脈。其功能主要為判斷當下的last\_change在轉換成BCD之後，應當存入addend、還是存入augend。

**3. Finite state machine**

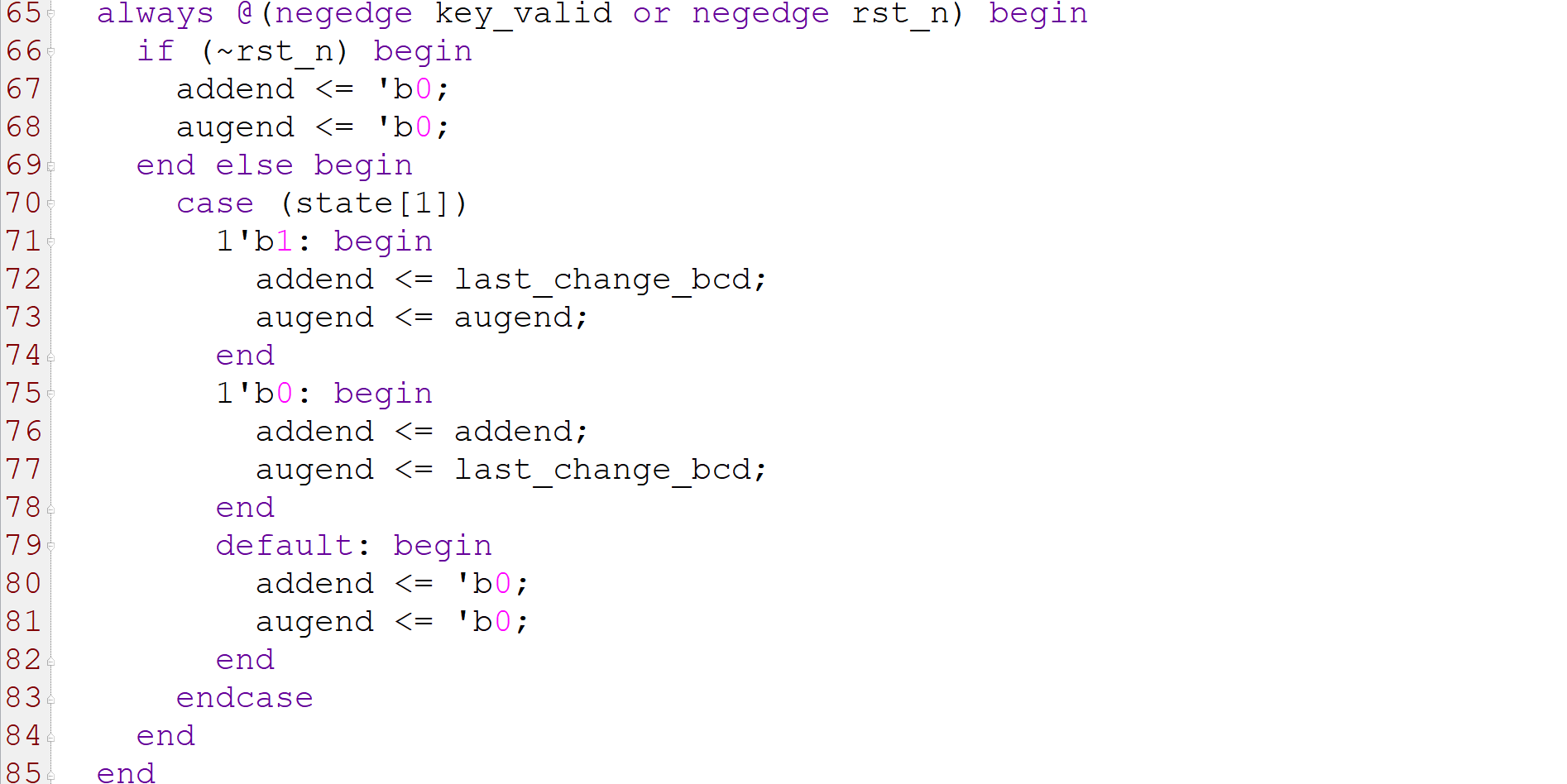
本題的FSM有四個state，可以表示成{被加數|加數, 按下|釋放}。如下圖。



此FSM的state transition為：按下被加數→釋放被加數→按下加數→釋放加數。可以使用一個二位元下數計數器來描述。

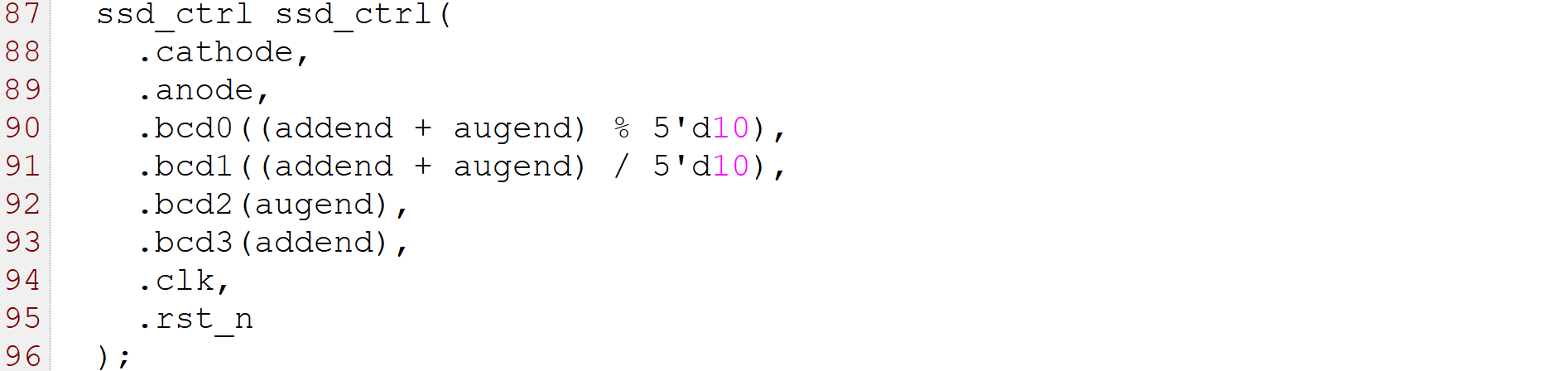


每當按下或釋放一個按鍵，key\_valid都會維持一個clk週期的1’b1。因此，使用key\_valid的falling edge作為Flip-flop的CLK，可以確保所有D都是已經更新後且穩定的資料。由於按下或釋放按鍵過了1/100M秒之後都會收到一次key\_valid的falling edge，所以只有在state從奇數變為偶數時才需要改變augend或addend的值。每按一次鍵盤，就依照順序將按下的數字（scan-code已經過look-up table轉換成BCD）存入加數addend、被加數augend。



**4. Implement**

接續上文的FSM，緊接著的是用來控制顯示器的模組。此段描述的是：將存放在FSM裡的addend與augend相加，再傳送到ssd\_ctrl.v，以在七段顯示器上顯示和。

 ssd\_ctrl.v的架構跟前幾次LAB中所使用的一樣，由除頻器、MUX、解碼器等組成，功能是將四個BCD轉換成SSD的樣式之後，以適當的頻率切換四個anode的enable，在四個七段顯示器上顯示出對應的cathode樣式。ssd\_ctrl.v的code跟上一次LAB的幾乎一模一樣，詳參.v檔。

**5. Conclusion**

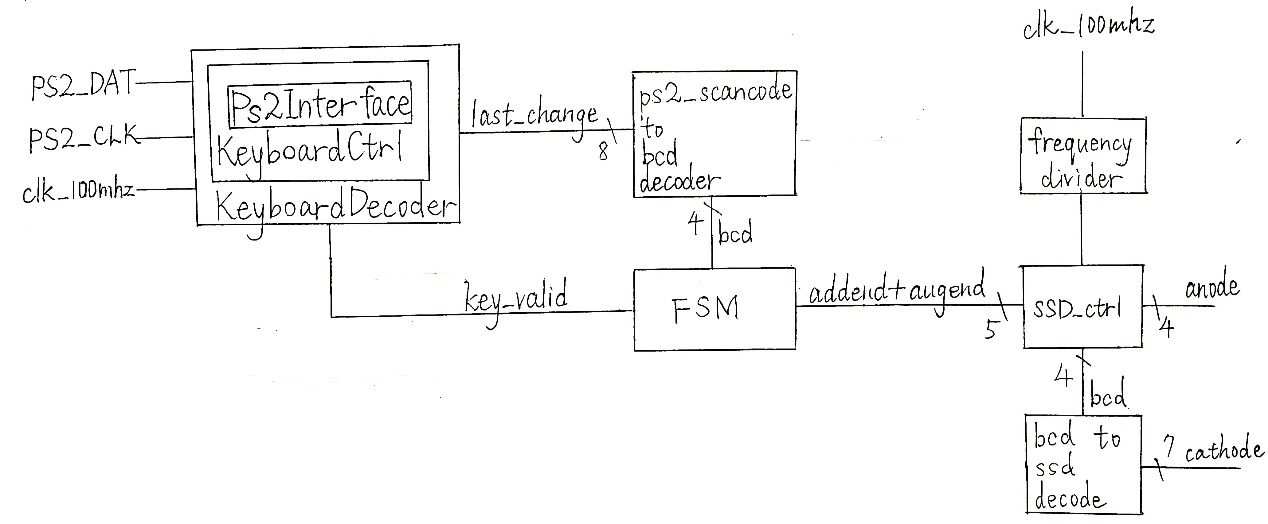
由於在本LAB中所使用的KeyboardDecoder.v等模組由老師提供，並非我自己親手架構，所以如果只靠老師在課堂上的講解內容，來理解各個output的含意與用法，是困難而吃力的。本題遭遇的最大困難在於，每一次按下或釋放鍵盤按鍵，key\_valid都會形成一個時長為1/100M秒鐘的脈衝訊號。因此，使用者按下並釋放一個按鍵的過程中，總共會產生兩個脈衝。在本題，只有第一個（因為按下按鍵而產生的）脈衝是有用的，第二個脈衝毫無用處，可以直接忽略。因此，本題如果要在建構FSM時，使用key\_valid來做為Flip-flop的時脈CLK的話，就必須要在「儲存addend」與「儲存augend」這兩個動作之間，多插入一個狀態，用來等待（跳過而不做任何事情）該按鍵被釋放。

透過本題，可以熟悉KeyboardDecoder.v的key\_valid訊號所代表的意涵，並練習使用key\_valid來做為FSM中Flip-flop的CLK。練習利用FSM的當下狀態來解讀數字的意涵，用state transisiton來將輸入一一存入加數與被加數中。結合前幾次LAB所建構出來的七段顯示器控制模組，反映使用者的輸入，並即時顯示兩個數字的和。

**3. Implement a two-digit decimal adder/subtractor/multiplier using the right-hand-side keyboard (inside the red block). You don’t need to show all inputs and outputs at the same time in the 7-segment display. You just need to show inputs when they are pressed and show the results after “Enter” is pressed.**

**1. Specification**

同第二題。

**2. Block Diagram**

**運算結果**

**7**

運算結果的bit width從一位數（4 bits，0~9）改成兩位數（7 bits，00~99），其餘同第二題。由於要輸入operator與enter，所以ps2-scancode-to-bcd decoder除了要把數字0~9的scan-code轉成bcd之外，我把a（加法）、s（減法）、m（乘法）、enter，這四個按鍵的scancode轉換成4’hA、4’hB、4’hC、4’hD。因為在原本的BCD中，4’hA、4’hB、4’hC、4’hD是unused，所以這麼做並不會影響到數字0~9的運作。本題的FSM有較大更動，上一題只需要四個state，依序存入addend、augend；本題則需要12個state，依序存入addend、augend的十位數、個位數，以及operator。

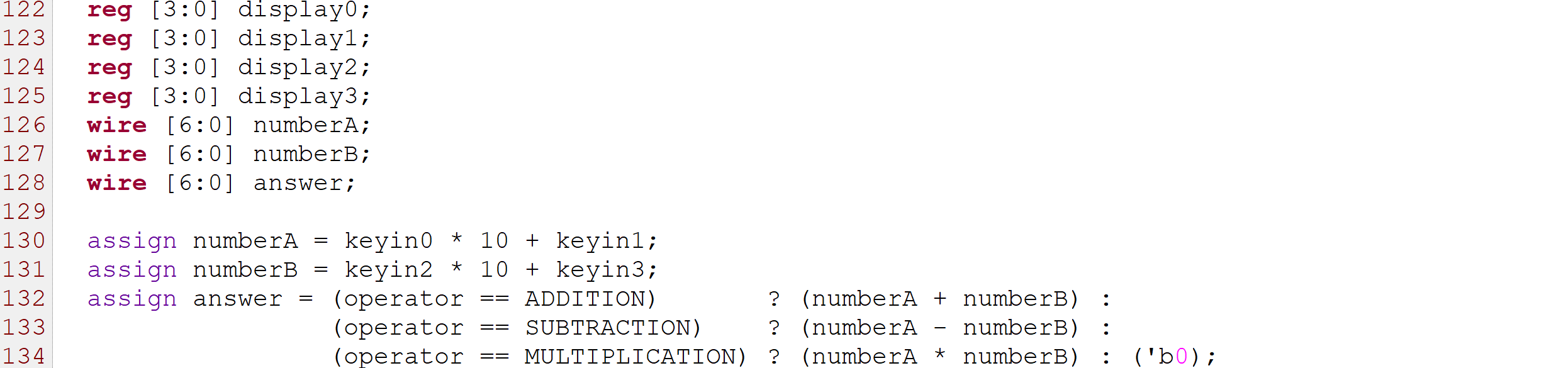
**3. Finite state machine**

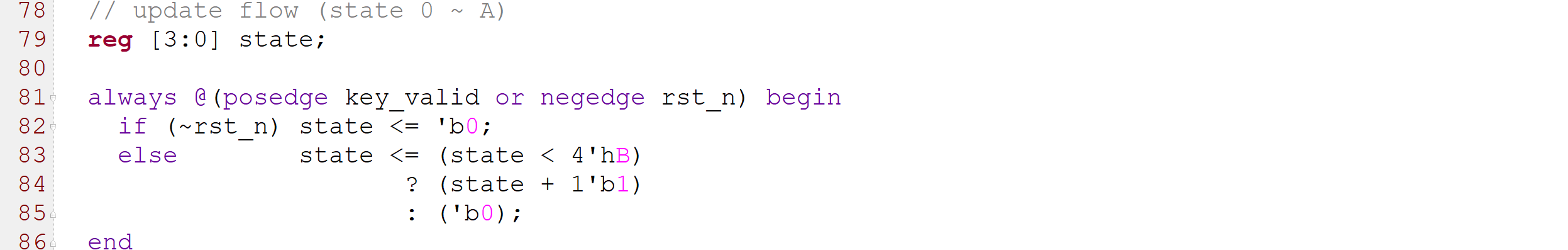
本題的FSM有12個狀態（state = 4’h0 ~ 4’hB）。其中，偶數的state都是等待按下按鍵，奇數的state都是等待釋放按鍵。因此，從偶數的state變成奇數的state的時候，需要將新的BCD存入對應的addend、augend的十位數、個位數（reg的名稱是keyin0 ~ keyin3）、operator。此外，雖然沒有在下圖中列出，但是為了要即顯示使用者輸入的數字，所以從偶數的state變成奇數的state的時候，也要將新的BCD顯示在七段顯示器上（存入display0 ~ display3），按下enter時則顯示運算結果。

keyin0 ~ keyin3儲存被加數和加數的十位數和個位數，operator則儲存要執行的運算。

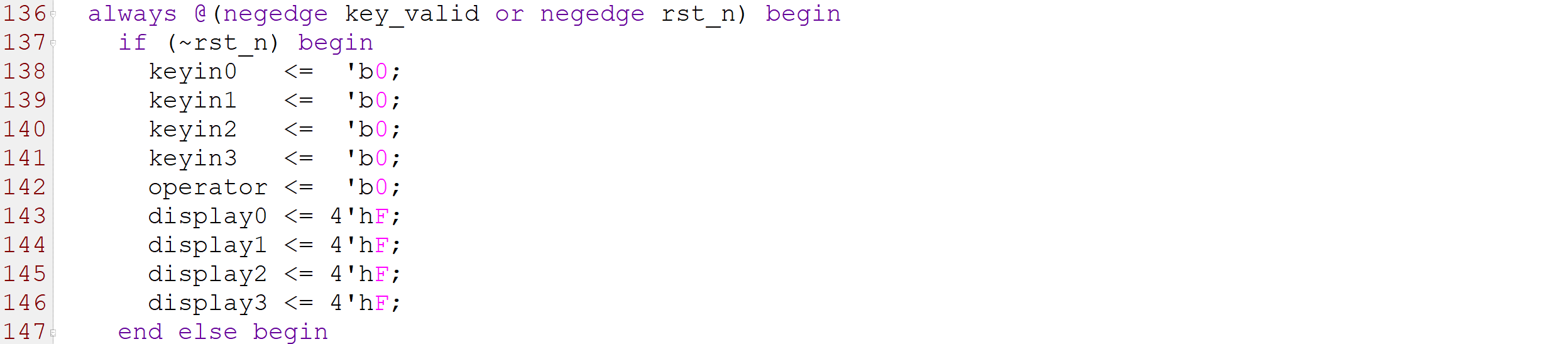


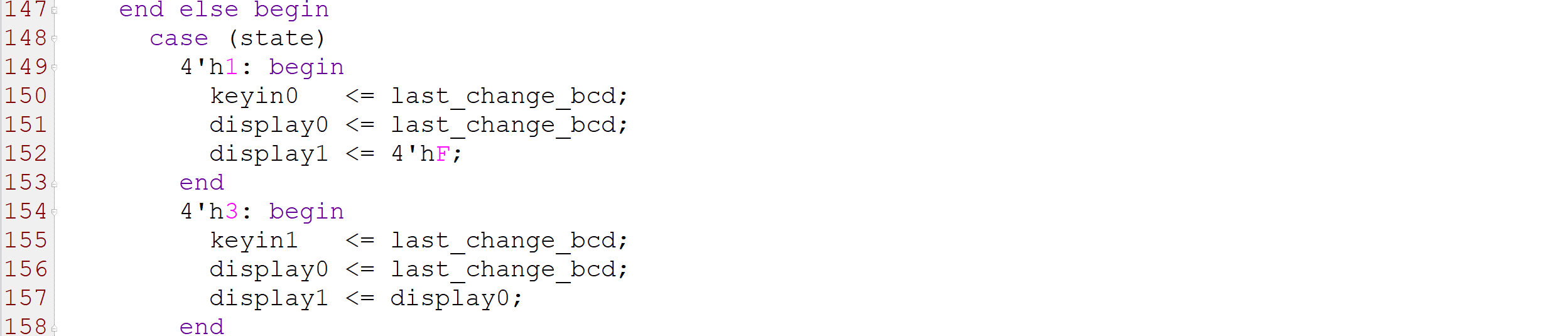
從偶數的state變成奇數的state時，會將BCD存入display0 ~ display3，以即時反應使用者的輸入。在state從4’hA變成4’hB時，則將運算結果answer存入display0 ~ display3，使七段顯示器顯示以這些資訊所計算出來的結果，如下。



FSM中，Flip-flop以key\_valid當作CLK，state transition可以用一個四位元上數計數器來描述，如下。

至於每一個state要做麼事情，另外用一個always block來描述。先描述重設時該做的事情，如下。



接下來一一列出所有state要做的事情。先依序將使用者的input存入被加數的十位數與個位數，如下。

然後將使用者的input存入operator，如下。



隨後再存入加數的十位數與個位數，如下。



上述狀態除了將使用者的input存入被加數或加數的十位數或個位數之外，同時也將BCD存入display0 ~ display3，以即時反應使用者的輸入於顯示器上。而按下enter時，則將上述數字的運算結果顯示於顯示器上，如下。



以上就是所有可能需要執行的動作。

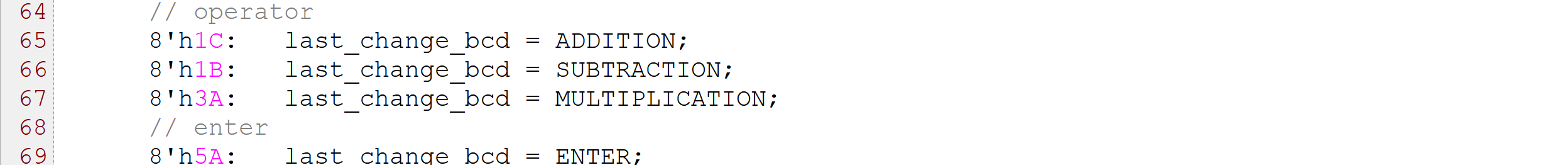
上述沒有提及的那些偶數的state則不做任何事情，如下。



**4. Implement**

首先，把KeyboardDecoder.v的last\_change用look-up table轉換成BCD，只要把所有0~9都列出來即可。同時，我特別將加減乘、enter編碼成4’hA、4’hB、4’hC、4’hD，如下。

****



經過FSM之後，只要再連接七段顯示器的控制模組即可，其作法為：先用除頻器產生適當的頻率（1/2^20），再用MUX依序enable四個anode，並將對應的display0 ~ display3傳送到BCD to SSD decoder。控制七段顯示器模組的部分跟上一次LAB一樣，只是reg的名稱換掉了而已，詳參.v檔。

**5. Conclusion**

這部分的FSM複雜了一點，我一開始還把控制keyin與display的描述拆開來寫，結果發現沒有比較簡單易懂，反而還讓我搞混十位數和個位數的輸入順序（例如，state1的時候，我誤將BCD存入了keyin0，而在state3的時候，將BCD存入keyin1，結果讓我的兩個數字的十位數跟個位數剛好倒過來，而且完全無法從顯示器察覺到異狀，因為display是接對的）。

本題可以學習到較複雜的FSM建構方法。我學期初時完全沒有想到我居然可以實作出一台計算機，我覺得我進步了許多。

**4. Implement the “Caps” control in the keyboard. When you press A-Z and a-z in the keyboard, the ASCII code of the pressed key (letter) is shown on 7-bit LEDs.**

**4.1 Press “Caps Lock” key to change the status of capital/lower case on the keyboard. Use a led to indicate the status of capital/lowercase in the keyboard and show the ASSCII code of the pressed key on 7-bit LEDs.**

**4.2 Implement the combinational keys. When you press “Shift” and the letter keys at the same time, 7-bit LEDs will show the ASCII code of the uppercase / lowercase of the pressed letter when the “Caps Lock” is at the lowercase / uppercase status.**

**1. Specification**

module LAB9\_4\_top(

output caps, // capital/lowercase

output reg [6:0] ascii, // pressed key

inout PS2\_DATA, // (PIC24) USB HID Host

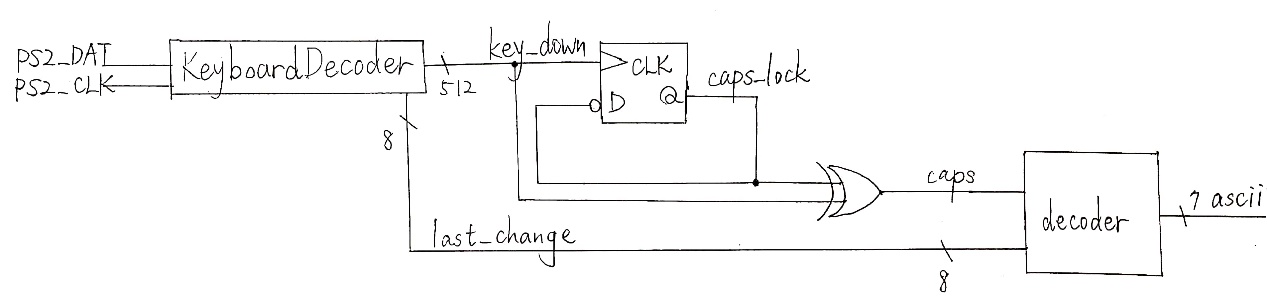
inout PS2\_CLK, // (PIC24) USB HID Host

input rst\_n, // active low reset

input clk // 100MHz global clock

);

**2. Block Diagram**

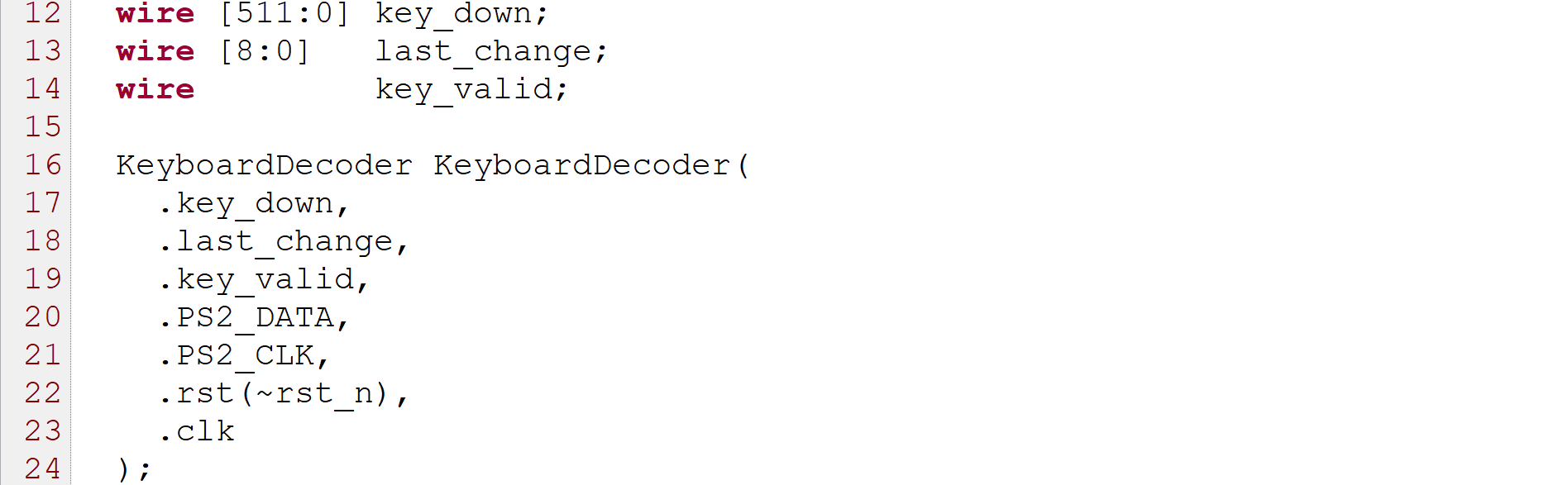
****

**3. Finite state machine**

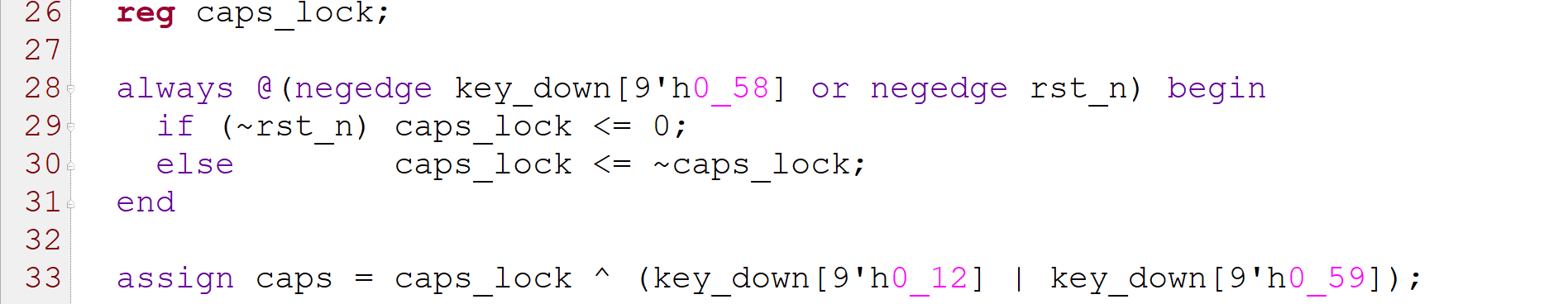
無

**4. Implement**

接好KeyboardDecoder.v。本題只需要使用其中的key\_down（用來判斷capslock與shift）、與last\_change（用來判斷按下的英文字母是a ~ z中的哪一個）。

****

判斷應轉換成大寫還是小寫。用T Flip-flop來儲存當下的caps\_lock，再跟shift做XOR，就可以知道要大寫還是小寫。其中9’h0\_12與9’h0\_59分別為左邊和右邊的shift；9’h0\_58則是鍵盤上的caps Lock按鍵scan-code。

****

最後，將判斷好的大小寫訊號caps傳送到MUX，就可以產生對應的ASCII code。內容只是把所有大寫和小寫的字母scan-code一一列出而已，詳參.v檔。

**5. Conclusion**

這題相對第三題來說單純很多，完全不需要使用到FSM就可以完成（其實理論上用來儲存caps的Flip-flop可以算是一種只有兩種state的FSM）。除了老師提供的KeyboardDecoder.v之外，都是應用之前LAB的觀念就好。