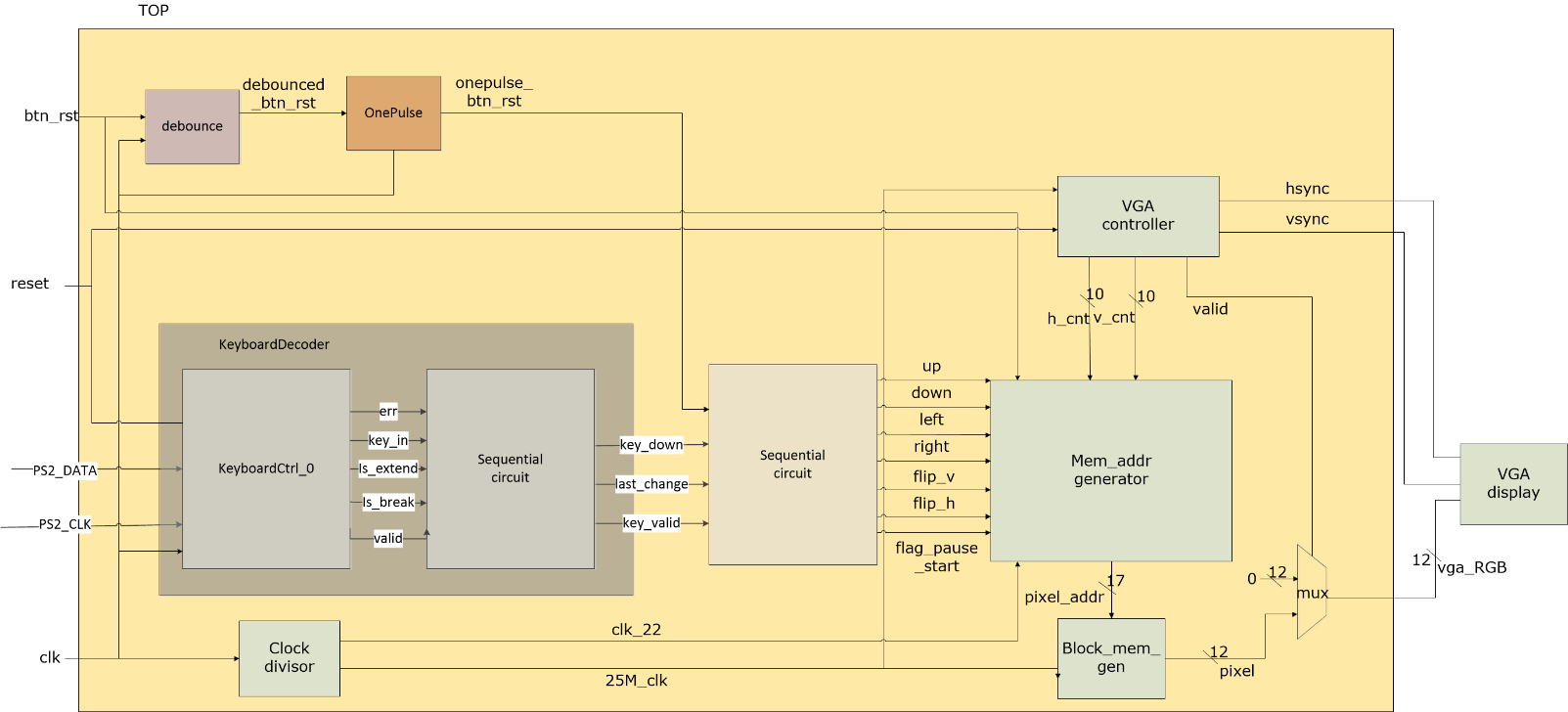
Lab 6: VGA and Mouse Modules

105030009 黎佑廷

105030015 郭家瑋

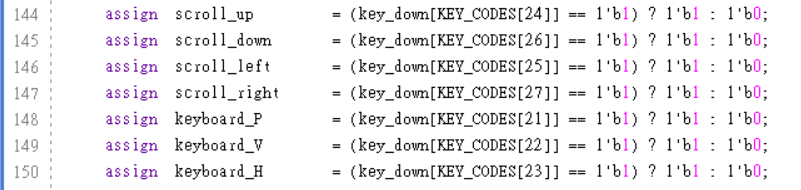
* **Block Diagram**



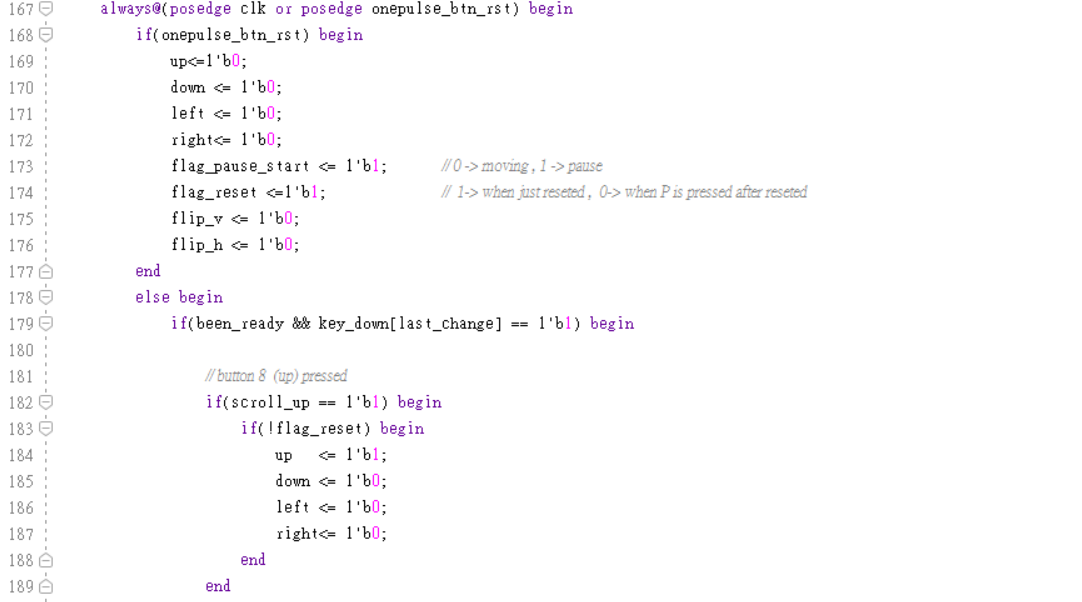
在Block Diagram中，top module的input有FPGA板的reset按鈕、重置Keyboard Decoder與VGA Controller的reset，但是這個reset並不用真的去按下一個reset的按鈕，Keyboard Decode與VGA Controller即會開始動作、鍵盤按鍵的WASD與PHV、最後是FPGA板內建的時脈。Output是控制升起或放下的hsync、vsync與每個pixel的RGB輸出給螢幕。

**Clk\_divisor**共產生兩種時脈。第一個是clk\_22，輸出頻率約為23Hz的時脈，是為Mem\_addr\_generator的input，這會影響到畫面的移動速度。第二個是25M\_clk，這是依據640\*840 pixel以60Hz的更新頻率來算出800\*525\*60=25M Hz (800與525包括了Visable area、Front Porch、Back Porch、Sync pulse的line與pixel)。這個25M\_clk是為Block\_mem\_gen的input，決定每讀一個pixel的時間。

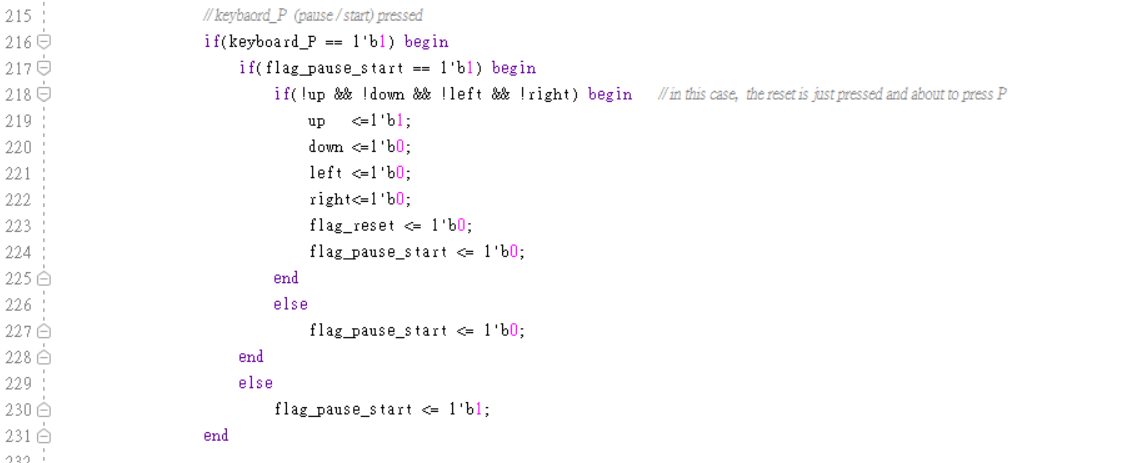
**KeyboardDecode**中，input是reset、鍵盤的PS2\_DATA、PS2\_CLK，經過IP產生的KeyboardCtrl\_0吐出err、key\_in、Is\_extend、Is\_break、valid訊號，經過一個Sequential circuit後會output Key\_down、last\_change、key\_valid，這會是在top module中判斷按下了哪些鍵的訊號。

FPGA上的reset按鍵經過debounce、Onepulse後，會和keyboardDecoder的output一起進入**top module**中的combinational 和sequential circuit。如下：

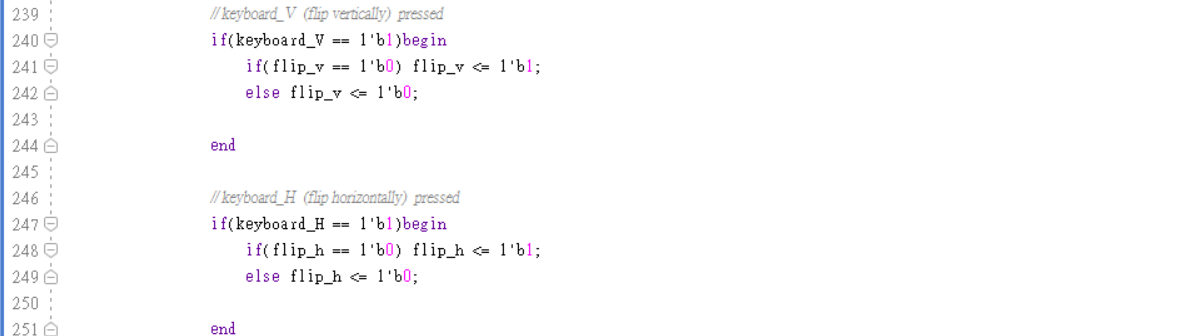
這邊利用key\_down來判斷是否按下鍵盤上按鍵。



**Up、down、left、right**是四個reg，紀錄現在是哪個方向的移動，並將那個方向設為1，其它為0。**Flag\_pause\_start**也是reg，紀錄現在螢幕的情況是移動還是暫停，移動則設為0，暫停為1。**Flip\_v與Flip\_h**是兩個功能類似布林函數的reg，紀錄現在有沒有被翻轉過。初始之為0，若按下水平或垂直的flip，則將flip\_h或flip\_v由0設為1，之後再次按下則由1變為0。**這七個訊號會output到Mem\_addr\_generator來決定每次要讀圖片記憶體的哪個address**。另外**flag\_reset**較為不重要，是為了處裡在按reset後，唯有按下P才能開始動的情況。因此，在if block中，在按下reset鍵後，因為要使畫面靜止，因此up、down、left、right都要設為0，flag\_pause\_start因為是在pause狀態，也設為1，flag\_reset在剛按下reset後設為1，等到按下P後才設變為0，並且解禁四個方向的給1。在else block中，若按下了W，scroll\_up會被assign 1，若此時已經按過了P，則flag\_reset會被設為0，因此可以解禁up的給1，並且設其它方向為0。在四個方向的實作方法皆同。

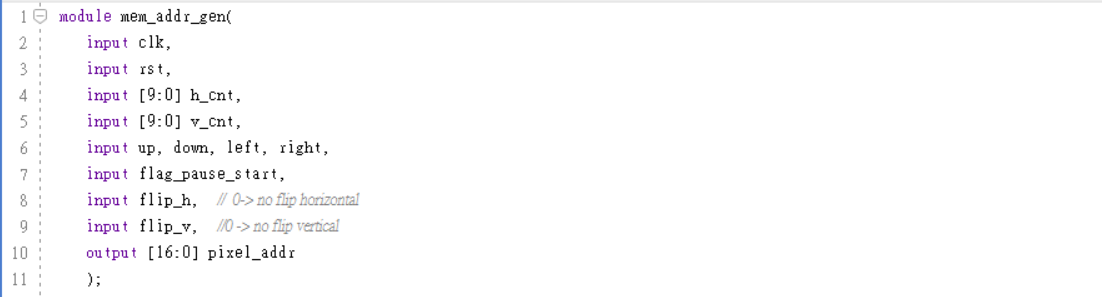


若是按下了P，則要先判斷按下P前的螢幕狀態，是因為按下reset後在等P才能開始移動，還是螢幕已經在移動，只是暫停或開始。因此，如果flag\_pause\_start==1，代表目前螢幕是停止狀況，如果恰巧又是up、down、left、right都是0，代表才剛剛按過reset，因此預設給up=1，其它為0，並且將flag\_reset設為0，代表P已經被按過了，可以解禁四個方向的按鍵控制，且將flag\_pause\_start設為0，代表現在在移動狀態。若不是在reset後按下P，則只要將flag\_pause\_start由1變0或0變1，代表一般的暫停或開始。

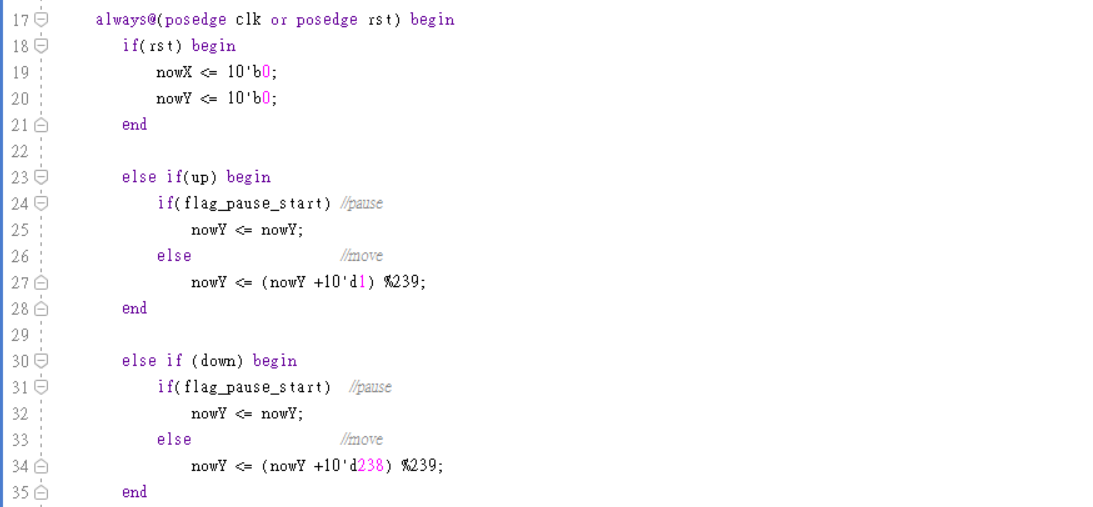


若是按下了V，代表鉛直的圖片要倒轉。因此，在這邊flip\_v、flip\_h類似布林函數功能，把flip\_v、flip\_h由1改0或0改1。

**Mem\_addr\_generator** module，是屬於最核心的module，決定在按下哪些鍵後，該如何將二維螢幕的每一個pixel RGB由一維的圖片記憶體中拿到。



這邊的input的clk，拿到的是23Hz的時脈，是控制畫面移動速度的時脈。input的rst必須要拿button reset的訊號，作為按下重置後，圖片能回到正中央的判斷依據。H\_cnt與v\_cnt是由VGA\_controller所output，給定現在在走到了螢幕上的鉛直與水平pixel數。Up、down、left、right、flag\_pause­\_start、flip\_h、flip\_v是調控H\_cnt與v\_cnt平移量的依據。最終會output記憶體位置給記憶體，讓記憶體可以output RGB。



這邊宣告nowX、nowY為reg，在button reset按下後重置為0，意義是加在h\_cnt與v\_cnt上的平移量，可以造成影像的移動。

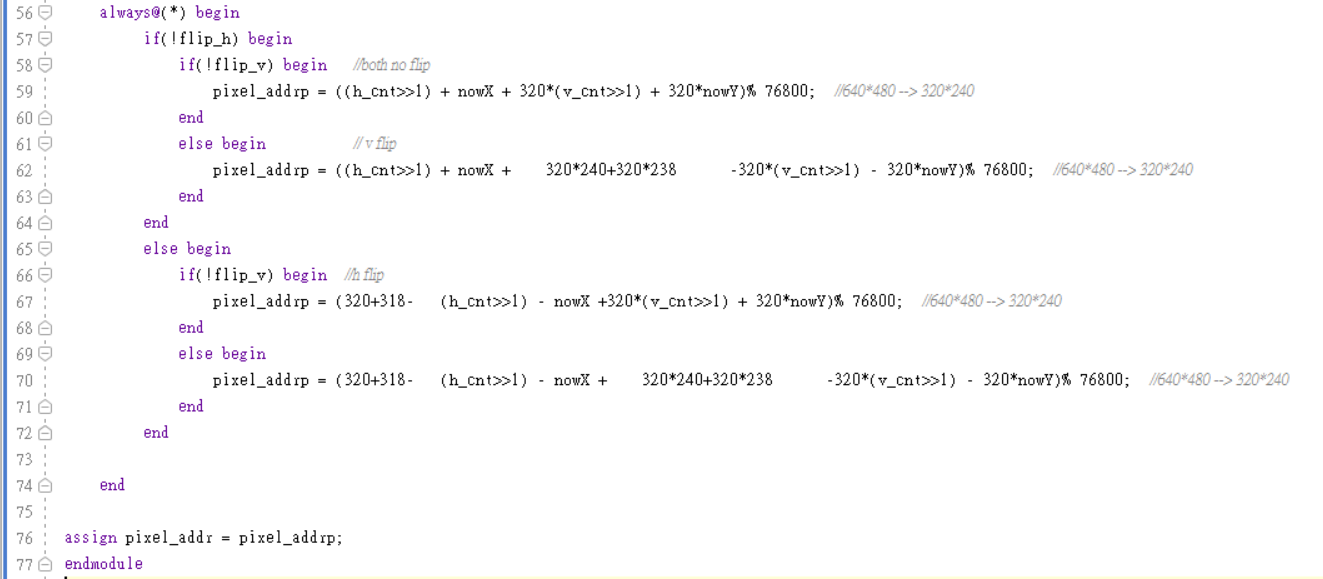
如果現在是up==1，但是pause的狀態，那意味垂直方向不會有速度，因此讓nowY<=nowY。如果現在是up==1且是在移動的狀態，那就讓nowY<=nowY+1，意義是在該pixel先讀到他下方pixel的記憶體address，並且在程式碼的最後會將nowY與cnt\_v相加。要做%239的原因是，不能讓nowY加至超過239，因為若nowY超過239，轉換為記憶體位置時，有可能已經超出了邊界，因此加至239後，要變成0重新開始。這也是畫面可以在往上捲時，上面消失的部分會從下方出現的原因。

如果現在是down==1，但是pause的狀態，那意味垂直方向不會有速度，因此讓nowY<=nowY。如果現在是down==1，且是在移動的狀態，那本應讓nowY<=nowY – 1，但考慮到減法容易會出現問題，關於signed或unsigned number的問題，所以改為nowY<=nowY+238，從圖形上來想，與減法意義相同。圖示如下。

0~319

0~239

若目前是up==1，且是移動狀態，假設現在mem\_addr\_generator的clk只跑了一個cycle，那麼圖上黑色pixel位置應該要讀到藍色pixel位置的記憶體，所以會顯示藍色。當h\_cnt、v\_cnt跑得很快，且所有的pixel都讀到他圖片上pixel下方的pixel對應到的記憶體位置，看起來就像是圖片上移了，當 mem\_addr\_generator的clk不斷跑動，就會形成圖片連續的上移。同理來處理右移，黑色的pixel要讀到紅色pixel的記憶體位置，所以將所有h\_cnt+319。下移則黑色的pixel要讀到綠色pixel的記憶體位置，所以將所有v\_cnt+239。左移則黑色的pixel要讀到黃色pixel的記憶體位置，所以將所有h\_cnt+1，這些在加在v\_cnt、h\_cnt之上的平移量用newX、newY紀錄。



**4**

**3**

**2**

**1**

用newX、newY處裡完圖片上下左右移的問題，接下來是要處裡newX、newY與h\_cnt、v\_cnt相結合並且給出最後記憶體的問題。這邊，因為vga\_controller module給出的h\_cnt與v\_cnt是以640\*480為大小，但我們的圖片只有320\*240，因此將兩者皆除以二。

在第一種情況，在水平方向與鉛直方向皆沒有翻轉，因此我們用將二維轉為一維的方式給出記憶體位置。如圖，藍色pixel的座標是(x,y)，因此，由左而右，由上而下，藍色pixel的一維位置是320\*x+y。最後要%76800 (76800=320\*240)也是因為可能超出記憶體。

在第二種情況，在垂直方向翻轉。如下圖，藍色的pixel要讀到黃色pixel位置的記憶體，因此，只要將y改成240-y即可。(該方向的最大值-本身值)

在第三種情況，在水平方向翻轉。如下圖，藍色的pixel要讀到紅色pixel位置的記憶體，因此，只要將x改成320-x即可。(該方向的最大值-本身值)

第四種情況，在水平方向與鉛直方向都翻轉，因此將第二種與第三種情況的效果綜合，x改成320-x、y改成240-y。

(0,0)

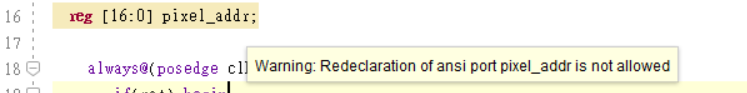
0~319

0~239

(x , y)

(320-x , y)

(x , 240-y)

最後，把記憶體位置先給一個buffer，在always外再assign給pixel\_addr的原因是似乎pixel\_addr不能宣告為reg。

最後，Block\_mem\_gen產生的RGB共12bit與VGA\_controller output的hsync、vsync一同output給VGA。不過Block\_mem\_gen的RGB要先經過mux，由VGA\_controller output 的valid判斷是否為顯示狀態來決定要output 0 還是RGB。

* **State Transition Diagram**

State transition的部分，因為主程式包括top module與核心的mem\_addr\_generator並無用到state transition，僅有keyboardDecoder的module用到state transition。下圖是keyboardDecoder的state transition diagram。

INIT

WAIT\_FOR\_SIGNAL

GET\_SIGNAL\_DOWN

WAIT\_FOR\_INVALID

Key\_in==8’hAA

Valid==0

Valid==1

Valid==1

Valid==0

Key<={been\_extend, been\_break, key\_in}

* 檢測方法似乎不太能用testbench，因為從testbench到VGA中也可能發生錯誤，故只用肉眼檢查VGA的運作是否正常。
* 心得：

(郭家瑋)

這次的lab其實不難，就是把鍵盤和VGA兜起來，再做個平移記憶體位置的處裡。不過寫起來還蠻有成就感的，相信這次lab會對final project很多幫助。

(黎佑廷)

我覺得這次的lab對於final project的幫助蠻大的，它讓我對於VGA的運作更加熟悉，相信經過lab6的訓練，final project會更加的順利。

* 工作分配：

共同完成code與report。