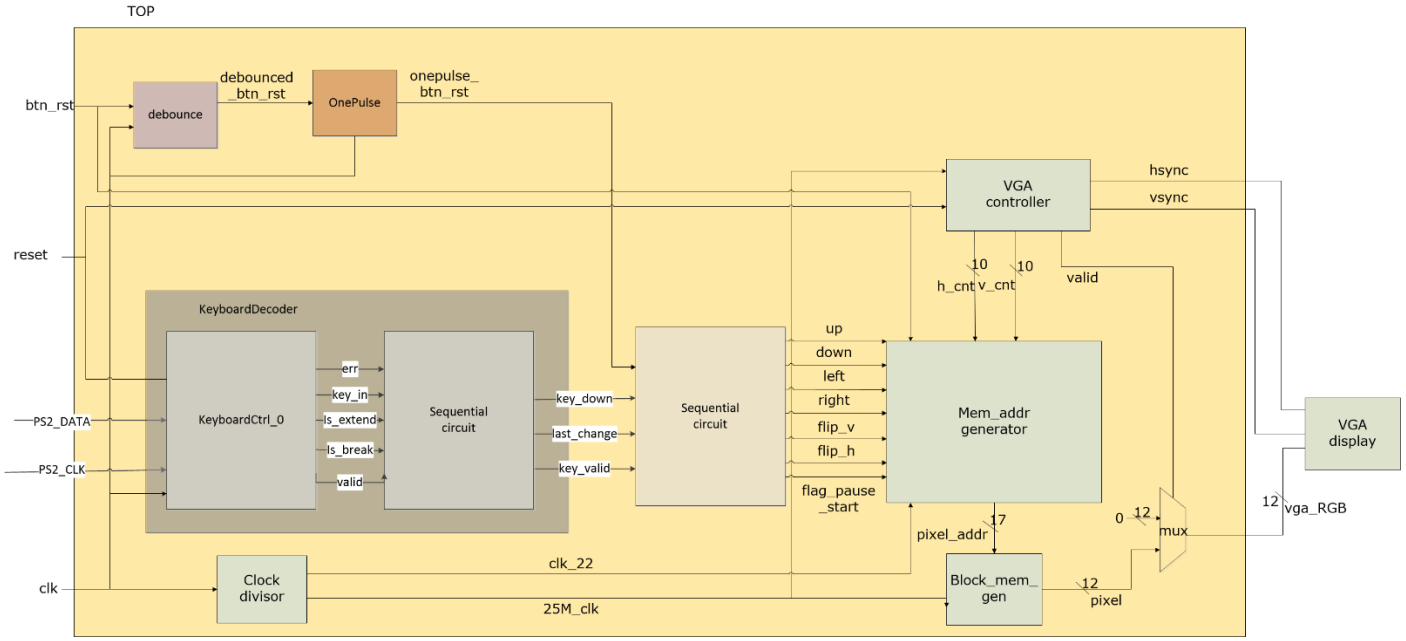
Lab 6: VGA and Mouse Modules

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、ls_extend、ls_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：

```
144     assign  scroll_up       = (key_down[KEY_CODES[24]] == 1'b1) ? 1'b1 : 1'b0;
145     assign  scroll_down     = (key_down[KEY_CODES[26]] == 1'b1) ? 1'b1 : 1'b0;
146     assign  scroll_left     = (key_down[KEY_CODES[25]] == 1'b1) ? 1'b1 : 1'b0;
147     assign  scroll_right    = (key_down[KEY_CODES[27]] == 1'b1) ? 1'b1 : 1'b0;
148     assign  keyboard_P      = (key_down[KEY_CODES[21]] == 1'b1) ? 1'b1 : 1'b0;
149     assign  keyboard_V      = (key_down[KEY_CODES[22]] == 1'b1) ? 1'b1 : 1'b0;
150     assign  keyboard_H      = (key_down[KEY_CODES[23]] == 1'b1) ? 1'b1 : 1'b0;
```

這邊利用 key_down 來判斷是否按下鍵盤上按鍵。

```
167 always@(posedge clk or posedge onepulse_btn_rst) begin
168     if(onepulse_btn_rst) begin
169         up<=1'b0;
170         down <= 1'b0;
171         left <= 1'b0;
172         right<= 1'b0;
173         flag_pause_start <= 1'b1;       //0 -> moving, 1 -> pause
174         flag_reset <=1'b1;              // 1-> when just reseted, 0-> when P is pressed after reseted
175         flip_v <= 1'b0;
176         flip_h <= 1'b0;
177     end
178     else begin
179         if(been_ready && key_down[last_change] == 1'b1) begin
180
181             //button 8 (up) pressed
182             if(scroll_up == 1'b1) begin
183                 if(!flag_reset) begin
184                     up   <= 1'b1;
185                     down <= 1'b0;
186                     left <= 1'b0;
187                     right<= 1'b0;
188                 end
189             end
```

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。

```
215                 //keybaord_P (pause / start) pressed
216                 if(keyboard_P == 1'b1) begin
217                     if(flag_pause_start == 1'b1) begin
218                         if(!up && !down && !left && !right) begin    //in this case, the reset is just pressed and about to press P
219                             up    <=1'b1;
220                             down <=1'b0;
221                             left <=1'b0;
222                             right<=1'b0;
223                             flag_reset <= 1'b0;
224                             flag_pause_start <= 1'b0;
225                         end
226                         else
227                             flag_pause_start <= 1'b0;
228                     end
229                     else
230                         flag_pause_start <= 1'b1;
231                 end
```

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。

```
239                 //keyboard_V (flip vertically) pressed
240                 if(keyboard_V == 1'b1)begin
241                     if(flip_v == 1'b0) flip_v <= 1'b1;
242                     else flip_v <= 1'b0;
243 
244                 end
245 
246                 //keyboard_H (flip horizontally) pressed
247                 if(keyboard_H == 1'b1)begin
248                     if(flip_h == 1'b0) flip_h <= 1'b1;
249                     else flip_h <= 1'b0;
250 
251                 end
```

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。

```
module mem_addr_gen(
    input clk,
    input rst,
    input [9:0] h_cnt,
    input [9:0] v_cnt,
    input up, down, left, right,
    input flag_pause_start,
    input flip_h,  // 0-> no flip horizontal
    input flip_v,  //0 -> no flip vertical
    output [16:0] pixel_addr
    );
```

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。

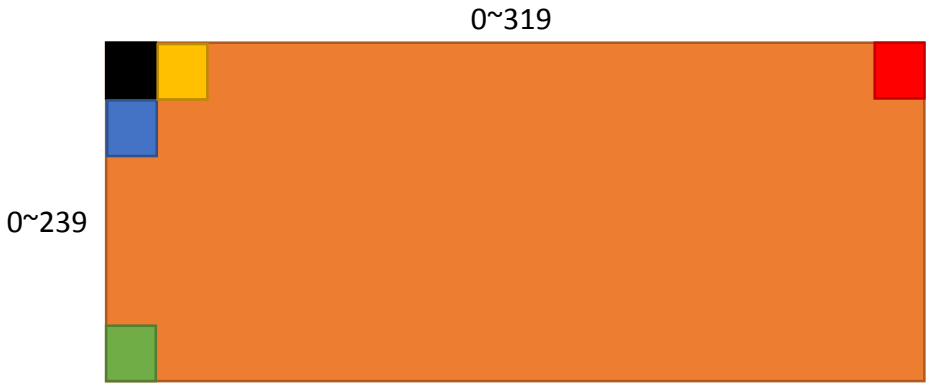
```
    always@(posedge clk or posedge rst) begin
        if(rst) begin
            nowX <= 10'b0;
            nowY <= 10'b0;
        end

        else if(up) begin
            if(flag_pause_start) //pause
                nowY <= nowY;
            else                 //move
                nowY <= (nowY +10'd1) %239;
        end

        else if (down) begin
            if(flag_pause_start)  //pause
                nowY <= nowY;
            else                  //move
                nowY <= (nowY +10'd238) %239;
        end
```

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。



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。

```
56      always@(*) begin
57          if(!flip_h) begin
58              if(!flip_v) begin    //both no flip
59      1           pixel_addrp = ((h_cnt>>1) + nowX + 320*(v_cnt>>1) + 320*nowY)% 76800;   //640*480 --> 320*240
60              end
61              else begin          // v flip
62      2           pixel_addrp = ((h_cnt>>1) + nowX +    320*240+320*238      -320*(v_cnt>>1) - 320*nowY)% 76800;   //640*480 --> 320*240
63              end
64          end
65          else begin
66              if(!flip_v) begin  //h flip
67      3           pixel_addrp = (320+318-   (h_cnt>>1) - nowX +320*(v_cnt>>1) + 320*nowY)% 76800;   //640*480 --> 320*240
68              end
69              else begin
70      4           pixel_addrp = (320+318-   (h_cnt>>1) - nowX +    320*240+320*238      -320*(v_cnt>>1) - 320*nowY)% 76800;   //640*480 --> 320*240
71              end
72          end
73
74      end
75
76  assign pixel_addr = pixel_addrp;
77  endmodule
```

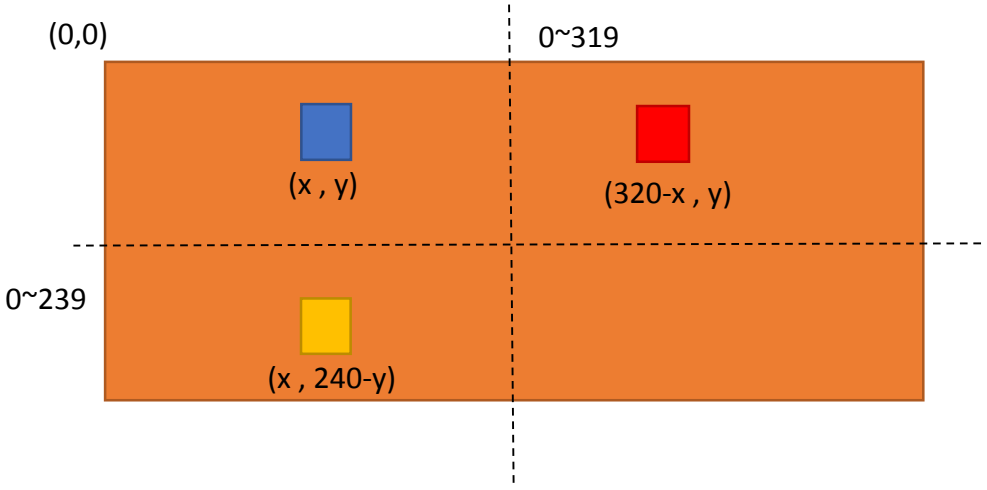
用 newX、newY 處裡完圖片上下左右移的問題，接下來是要處裡 newX、newY 與 h_cnt、v_cnt 相結合並且給出最後記憶體的問題。這邊，因為 vga_controller module 給出的 h_cnt 與 v_cnt 是以 640*480 為大小，但我們的圖片只有 320*240，因此將兩者皆除以二。

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。



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。

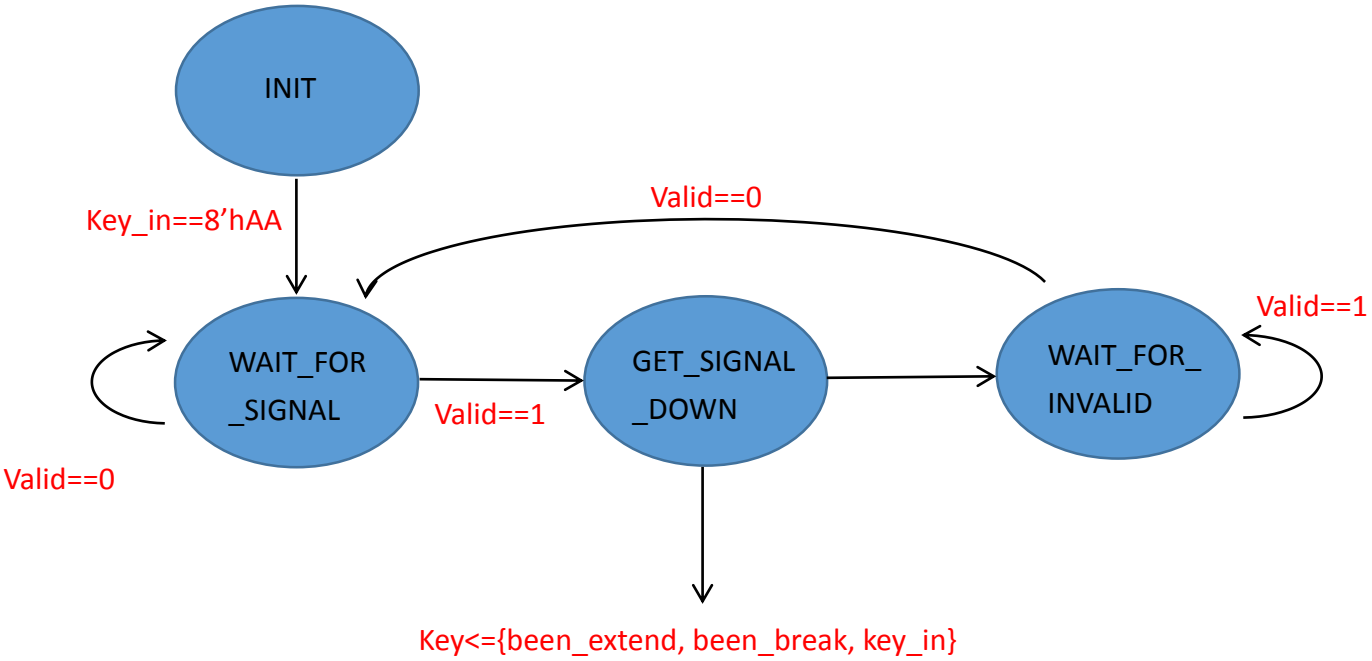
```
16  reg [16:0] pixel_addr;
17
18      always@(posedge cl
```

Warning: Redeclaration of ansi port pixel_addr is not allowed

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。



- ✓ 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。