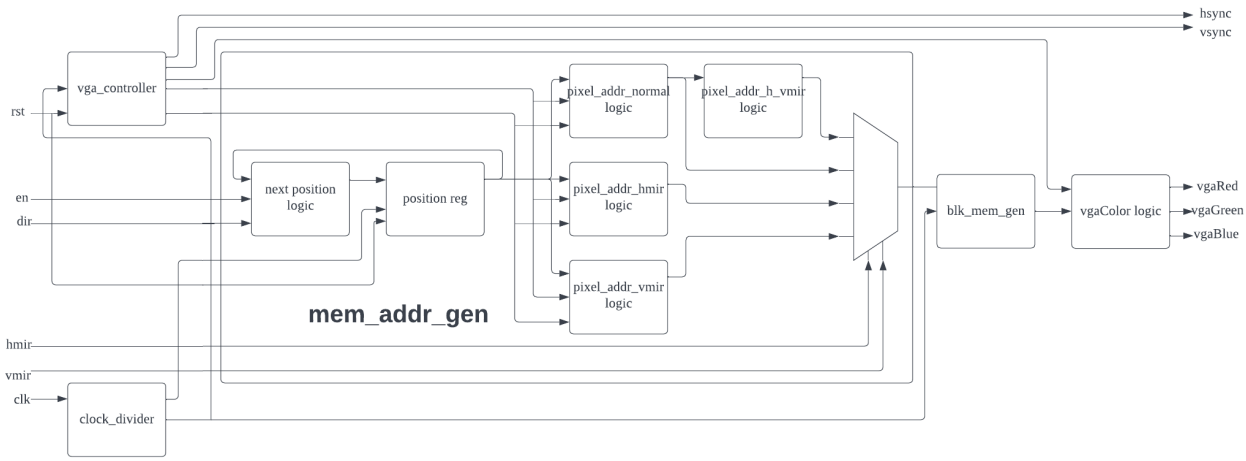
Lab 7

學號: 110062131 姓名: 馬毓昇

A. Lab Implementation

1. Lab7_1: Block diagram of the design with explanation



   a. Lab7_1 module 裡包含了 vga_controller、blk_mem_gen、mem_addr_gen 三個主要 module，其中 mem_addr_gen 是主要邏輯的地方，能透過數學運算出 v_cnt, h_cnt 對應到要顯示的 pixel 的 address 為何。
   b. 因為要設計滾動，所以在 mem_addr_gen 裡面有一個 position reg 當作 offset，最後用 mux 來決定要輸出哪種行為。

2. Lab7_1: Partial code screenshot with the explanation: you don't need to paste the entire code into the report. Just explain the kernel part.
   a. Lab7_1 module 裡面都跟 demo 的 template 長一樣，重點是在 mem_addr_gen 裡面。

```
assign {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel:12'h0;

clock_divider clk_wiz_0_inst(
    .clk(clk),
    .clk1(clk_25MHz),
    .clk22(clk_22)
);

my_mem_addr_gen mem_addr_gen_inst(
    .clk(clk_22),
    .rst(rst),
    .h_cnt(h_cnt),
    .v_cnt(v_cnt),
    .pixel_addr(pixel_addr),
    .dir(dir),
    .en(en),
    .vmir(vmir),
    .hmir(hmir)
);
```

```
blk_mem_gen_0 blk_mem_gen_0_inst(
    .clka(clk_25MHz),
    .wea(0),
    .addra(pixel_addr),
    .dina(data[11:0]),
    .douta(pixel)
);

my_vga_controller   vga_inst(
    .pclk(clk_25MHz),
    .reset(rst),
    .hsync(hsync),
    .vsync(vsync),
    .valid(valid),
    .h_cnt(h_cnt),
    .v_cnt(v_cnt)
);
```

b. Position: 當 en 被按的時候更新 position，不然 position 就不變。如果 dir 沒被按，position 就+1 形成往上滾動的效果，不然 position 就-1 形成往下滾動的效果。額外有用 if 做邊界判斷，如果超過 0~239 就要從重置 position。

```
if(en == 1'b1) begin
    if(dir == 1'b0) begin
        if(position < 239) begin
            next_position = position + 1;
        end
        else begin
            next_position = 0;
        end
    end
    else begin
        if(position > 0) begin
            next_position = position - 1;
        end
        else begin
            next_position = 239;
        end
    end
end
```

c. Pixel_addr：用一堆三元運算子來判斷要輸出哪個狀態，HV 都 mirror、H mirror、V mirror 或是正常(normal)。

Pixel_addr_normal: 把 h_cnt/2 + v_cnt/2*320 相當於把 2 的 h_cnt/2, v_cnt/2 座標轉換成一維的 address(h_cnt/2 橫向每多 1 代表要多 1 pixel，v_cnt/2 直向每多 1 代表要多一橫行也就是 320 pixels)，再加上 position*320(position 代表直向 offset，每多 1 就要多一橫行也就是 320 pixels)，最後模除 76800(640*480 共 76800 pixels，可以取得圖上的相對 address)，而因為 position 會一直更新，這樣就可以達到轉動的效果。

Pixel_addr_h_vmir: 用 76799 減去 pixel_addr_normal，就可以讓整張圖 pixel 順序 reverse，達成同時上下左右翻轉。

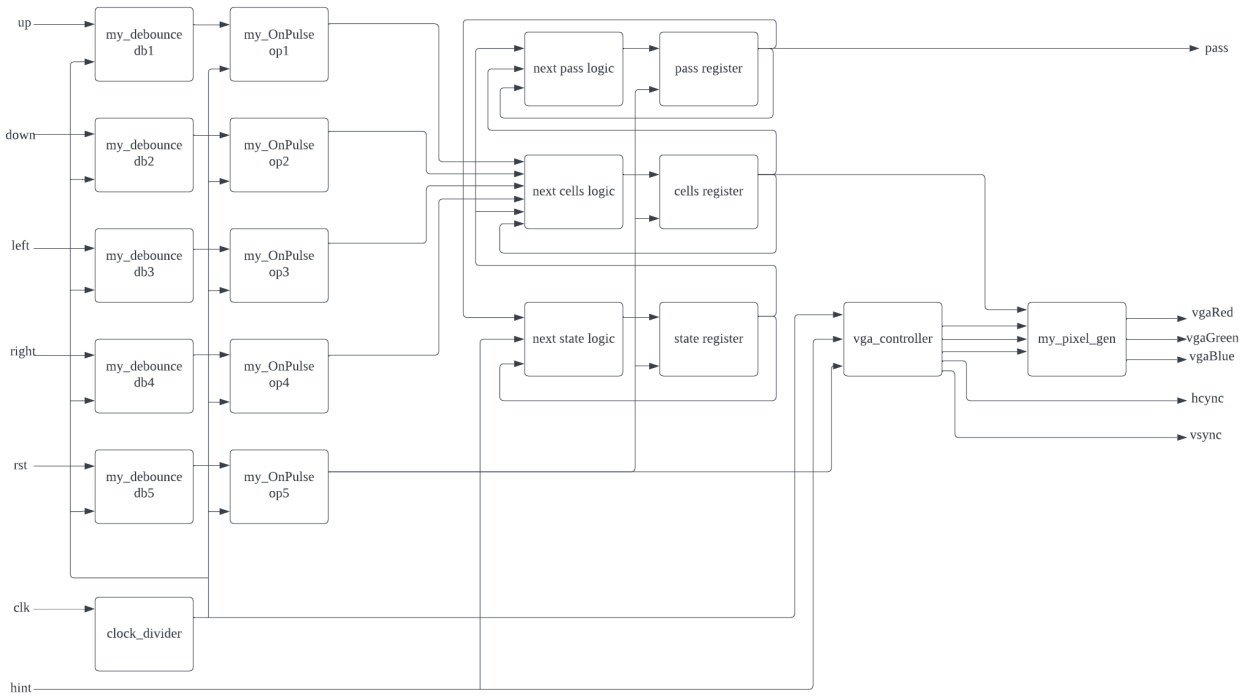
Pixel_addr_hmir: 用 319 減去 h_cnt/2 就可以讓每一橫行的 pixel 順序 reverse 過來，達成左右翻轉。

Pixel_addr_vmir: 用 239 減去 v_cnt/2 就可以讓每一直行的 pixel 順序 reverse 過來，比較不一樣的後面要減 position 而不是加，這樣圖片才能按照原本的滾動方向滾動。而因為是減，所以有可能會小減大然後溢位，模除就會有問題，因此在模除前要加 76800 來避免。

```
assign pixel_addr = hmir == 1'b1 && vmir == 1'b1 ? pixel_addr_h_vmir :
                    hmir == 1'b1 ? pixel_addr_hmir :
                    vmir == 1'b1 ? pixel_addr_vmir :
                    pixel_addr_normal;

assign pixel_addr_normal = ((h_cnt>>1)+320*(v_cnt>>1)+ position*320 )% 76800;  //640*480
assign pixel_addr_h_vmir = 76799 - pixel_addr_normal;
assign pixel_addr_hmir = (319-(h_cnt>>1)+320*(v_cnt>>1)+ position*320 )% 76800;
assign pixel_addr_vmir = ((h_cnt>>1)+320*(239-(v_cnt>>1))- position*320 + 76800)% 76800;
```

1. Lab7_2: Block diagram of the design with explanation



   a. up、down、left、right、rst 有用 debounce 跟 onepulse 處理
   b. FSM 部分有 state 作狀態變化，pass 判斷結束了沒，cells 為記錄背後的 puzzle(0~15)的 register
   c. vga_controller 控制 vga，my_pixel_gen 接受 vga_controller 的 v_cnt 跟 h_cnt 還有 cells 的訊號來判斷要輸出哪個 addr 的 pixel。

2. Lab7_2: Partial code screenshot with the explanation: you don't need to paste the entire code into the report. Just explain the kernel part
   a. state： 本題我有用 state 來作，分別作了 INIT、GAME、HINT、PASS 四個 state。當 hint 被按起時，state 會從 GAME 跳到 HINT，反之則會從 HINT 跳回 GAME。如果 pass 了就進到 PASS。

```
43  parameter INIT = 2'd0;
44  parameter GAME = 2'd1;
45  parameter HINT = 2'd2;
46  parameter PASS = 2'd3;
47  reg [1:0] state = INIT;
48  reg [1:0] next_state;
```

```
case(state)
    INIT : begin
        next_state = GAME;
    end
    GAME : begin
        if(hint == 1'b1) begin
            next_state = HINT;
        end
        else if(pass == 1'b1) begin
            next_state = PASS;
        end
    end
    HINT : begin
        if(hint == 1'b0) begin
            next_state = GAME;
        end
    end
endcase
```

b. puzzle 邏輯：如果 up, down, left, right 被按起，就找出 cell 是 0 的位置，並判斷反方向是不是 valid 的 cell，如果是就兩者交換。

```
if(up_1pulse) begin
    for(i=0; i<16; i=i+1) begin
        if(cells[i*4 +: 4] == 4'd0) begin
            j = i;
        end
    end
    if(j >= 0 && j <= 11) begin
        //j = i+4;
        next_cells[(j+4)*4 +: 4] = 4'd0;
        next_cells[j*4 +: 4] = cells[(j+4)*4 +: 4];
    end
end
```

c. pass: 我有做一個 pass_cells 來表示答案，過程中就一直檢查 cells 跟 pass_cells 有沒有完全一樣，有的話 pass 就會變 1。

```
27  reg [0:63] pass_cells = {
28      4'd1,  4'd2,  4'd3,  4'd4,
29      4'd5,  4'd6,  4'd7, 4'd8,
30      4'd9, 4'd10,  4'd11, 4'd12,
31      4'd13, 4'd14, 4'd15, 4'd0
32  };
```

```
GAME : begin
    next_pass = 1'b1;
    for(k=0; k<16; k=k+1) begin
        if(cells[k*4 +: 4] != pass_cells[k*4 +: 4]) begin
            next_pass = 1'b0;
        end
    end
end
```

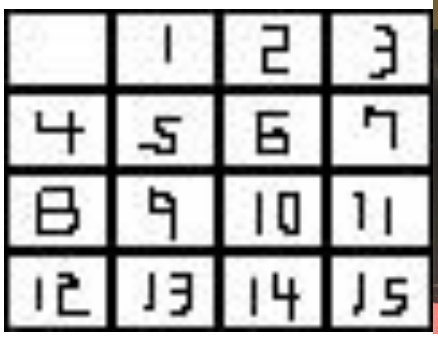
c. my_pixel_gen： 核心 module，會依據 h_cnt、v_cnt、cells 的模樣的輸出 pixel。

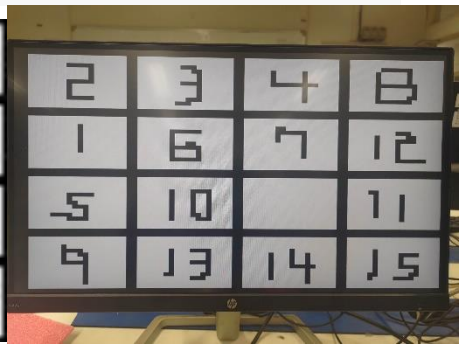
```
my_vga_controller   vga_inst(
  .pclk(clk_25MHz),
  .reset(rst_1pulse),
  .hsync(hsync),
  .vsync(vsync),
  .valid(valid),
  .h_cnt(h_cnt),
  .v_cnt(v_cnt)
);
```

在這個 module 裡，我有放圖片的 bitmap，大小是 80*60=4800 bits，然後還有一個 reg 標記出”每個 cell 數字對應到的小 bitmap”的 index 起始點是多少，小 bitmap 就是把大 bitmap 切成 4x4 每個是 20*15 pixels。

```
parameter [11:0] color [0:4799] = {
12'h000,
12'h000,
12'h000,
12'h000,
12'h000,
12'h000,
12'h000,
12'h000,
12'h000,
```

```
parameter [11:0] position [0:15] = {
    12'd0, 12'd20, 12'd40, 12'd60,
    12'd1200, 12'd1220, 12'd1240, 12'd1260,
    12'd2400, 12'd2420, 12'd2440, 12'd2460,
    12'd3600, 12'd3620, 12'd3640, 12'd3660
};
```

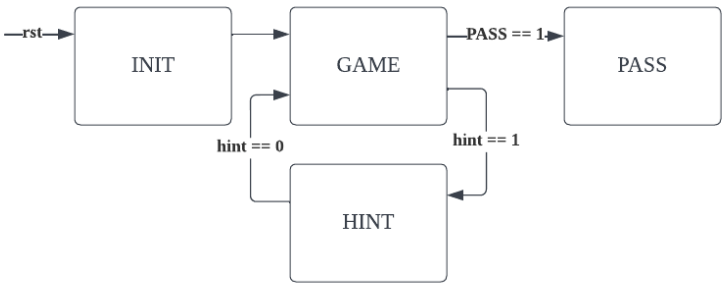




至於我算出 v_cnt、h_cnt 要輸出的 pixel 的方式是：
1. 先用 nested if-else statement 判斷出這個 v_cnt、h_cnt 對應到的是 cell[x : x+3] 現在是甚麼值(value: 0~15)
2. position[value]取出這個值對應到 bitmap 上的起點
3. ((v_cnt>>3)%15)*80 + (h_cnt>>3)%20 取出目前 v_cnt、h_cnt 相對於小 bitmap 起點的相對 address 差
4. 兩者相加：position[cells[0:3]] + ((v_cnt>>3)%15)*80 + (h_cnt>>3)%20 即是 pixel address color[address] 便是 pixel

```
else if(v_cnt < 120) begin
    if(h_cnt < 160)
        {vgaRed, vgaGreen, vgaBlue} =
            color[position[cells[0:3]] + ((v_cnt>>3)%15)*80 + (h_cnt>>3)%20];
    else if(h_cnt < 320)
        {vgaRed, vgaGreen, vgaBlue} =
            color[position[cells[4:7]] + ((v_cnt>>3)%15)*80 + (h_cnt>>3)%20];
    else if(h_cnt < 480)
        {vgaRed, vgaGreen, vgaBlue} =
            color[position[cells[8:11]] + ((v_cnt>>3)%15)*80 + (h_cnt>>3)%20];
    else if(h_cnt < 640)
        {vgaRed, vgaGreen, vgaBlue} =
            color[position[cells[12:15]] + ((v_cnt>>3)%15)*80 + (h_cnt>>3)%20];
    else
        {vgaRed, vgaGreen, vgaBlue} = 12'h000;
end
else if(v_cnt < 240) begin
    if(h_cnt < 160)
        {vgaRed, vgaGreen, vgaBlue} =
            color[position[cells[16:19]] + ((v_cnt>>3)%15)*80 + (h_cnt>>3)%20];
    else if(h_cnt < 320)
        {vgaRed, vgaGreen, vgaBlue} =
            color[position[cells[20:23]] + ((v_cnt>>3)%15)*80 + (h_cnt>>3)%20];
    else if(h_cnt < 480)
        {vgaRed, vgaGreen, vgaBlue} =
            color[position[cells[24:27]] + ((v_cnt>>3)%15)*80 + (h_cnt>>3)%20];
    else if(h_cnt < 640)
        {vgaRed, vgaGreen, vgaBlue} =
            color[position[cells[28:31]] + ((v_cnt>>3)%15)*80 + (h_cnt>>3)%20];
    else
        {vgaRed, vgaGreen, vgaBlue} = 12'h000;
end
```

3. lab7_2 FSM:
1. -> INIT: rst 被按
2. INTI -> GAME: 進 INIT 下個 cycle 就進 GAME
3. GAME -> HINT: hint 被按
4. INIT->GAME: hint 沒被按
5. GAME->PASS: 贏了



B. Questions and Discussions

A: wire pixel_addr = ((h_cnt>>1+position)%320 + 320*(v_cnt>>1))，讓向左捲動時，保持 position 遞增，向右捲動時，則保持 position 遞減，position 在 0~319 之間更新。(h_cnt>>1 + position)可以讓 pixel 水平移動，模除 320 則是找出 pixel 在這橫行的正確相對位置，最後在正常的補上 320*(v_cnt>>1)，這樣一來就可以達成左右捲動的效果。

B: 可以用 reg [3:0] puzzle [0:3][0:3] 的方式來做出存 puzzle 狀態的 4x4 大小的 register(0~15)。

C:
1. 如果圖片的顏色很少的話(黑白遊戲)，可以用 Run-length encoding algorithm 來存圖以節省空間。例如：有一黑白顏色列 0001100000110000，0 代表白 1 代表黑，可以不用{12'hFFF, 12'hFFF, …}來存，改用{4'd3, 4'd2, 4'd5, 4'd2, 4'd4}來存，就可以把 12*16=192 個 bits 降到 20 個 bits。
2. 如果是要做遊戲背景的話，可以選用幾何圖形重複的方式來製作，這樣的話，只需要儲存一個圖形的模樣，然後再依據 h_vnt、v_cnt 算出要輸出哪個 pixel 的 address 就好。假設一個 640x480 的畫面被分割成 4x4 個圖案，每一個圖案就只佔 160x120，那麼實際上就只需要存 19200*12=230400 bits 而不用到 3686400 bits，便可以用更小的空間輸出同樣解析度的圖片。

C. Problem Encountered

1. 第一個問題是我在做 lab7_1 時遇到的，就是我在弄上下顛倒的時候要減 position 才能達到遞增時向上滾動、遞減時向下滾動。但只減的話，就會觀察到說滾動的時候會破圖。想了想才發覺是因為負數取模會壞掉，解決辦法就是在取模前加回來就可以了。

```
assign pixel_addr_vmir = ((h_cnt>>1)+320*(239-(v_cnt>>1))- position*320 + 76800)% 76800;
// assign pixel_addr_vmir = ((h_cnt>>1)+320*(239-(v_cnt>>1))- position*320)% 76800;
```

2. 第二個問題是我在做 lab7_2 時遇到的，就是怎麼算都算不好數字對應到的圖片 address 要怎麼取，後來發現問題是出在我定義每個數字所對應的圖片的最左上那點座標找錯了，最後就用 reg 暴力記左上座標，直接省去運算。

```
280  parameter [11:0] position [0:15] = {
281      12'd0, 12'd20, 12'd40, 12'd60,
282      12'd1200, 12'd1220, 12'd1240, 12'd1260,
283      12'd2400, 12'd2420, 12'd2440, 12'd2460,
284      12'd3600, 12'd3620, 12'd3640, 12'd3660
285  };
```

3. systemverilog 不支援 cell[i : i+3]兩邊有一邊不是 constant 的用法，查了才知道要改成 cell [i +: 4]，學到一個新語法。

D. Suggestions

這次的 lab 實作出來的 15puzzle 遊戲感覺完成度比上次的 1A2B 還要高，說實在我很喜歡作這類型很像遊戲的 lab，希望之後可以多一點~