Lab 7

學號: 109000168 姓名: 許媄香

1. 實作過程
   - lab7_1

     Here we need to design a VGA controller that can scroll up or down an image and also could change the image into negative film.

     Also, when doing lab7_1, I used the code from the lab7 sample given by the TA. Since the sample demo is kind of similar to lab7_1, I decided to use that code and modify it a bit to meet the lab7_1 requirement.

     This lab has some requirements to do.
     - **rst**

       rst: the positive-edge-triggered reset, the VGA display will show the image at the origin position after triggered.

       I just simply reset the **position** reg to '0' because **position** play a role in changing the location of the picture, which later will be used in assigning **pixel_addr**.

```
assign pixel_addr = ((h_cnt>>1)+320*(v_cnt>>1)+ position*320 )% 76800;

always @ (posedge clk or posedge rst) begin
    if(rst) position <= 0;
```

     - **en**

       If **en** == 1'b0: hold the image on the screen unchanged.

       If **en** == 1'b1:

       - If **dir**== 1'b0:
         - the image scrolls up at the frequency of 100MHz divided by 2^{22}.
       - If **dir**== 1'b1:
         - the image scrolls down at the frequency of 100MHz divided by 2^{22}.

       When **en** signal is 0, the position will remain the same (not updated). But when the signal is '1', it will check the **dir** signal. If **dir** is (scroll up), **position** will be subtracted by one when **position** at that time is bigger than '0', otherwise just make **position** equal to '239' (bottom line of pixel in the picture because the original picture has the height 239). And it will the opposite when **dir** signal is '0' (scroll down). Just add **position** by one when the **position** now is not bigger than '239'. Otherwise, make the **position** become '0'.

```
else begin
    if(en) begin
        if(!dir) begin
            if(position < 239) position <= position + 1;
            else position <= 0;
        end else begin
            if(position > 0) position <= position - 1;
            else position <= 239;
        end
    end
end
```

- **nf**

  If nf==1'b0: show the original colors of the image.

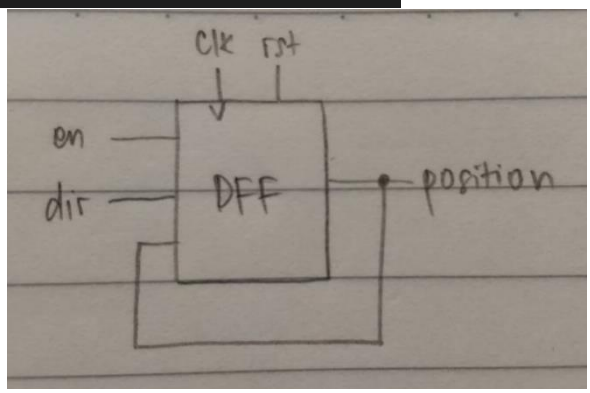
  If nf==1'b1: change the image into negative film (which means the colors reversed into their complementary colors).

  nf signal decide the color for the image, which mean it related to RGB. The color of the image will change to its complementary colors only when **nf** signal is '1'. Therefore, I design my code as below.

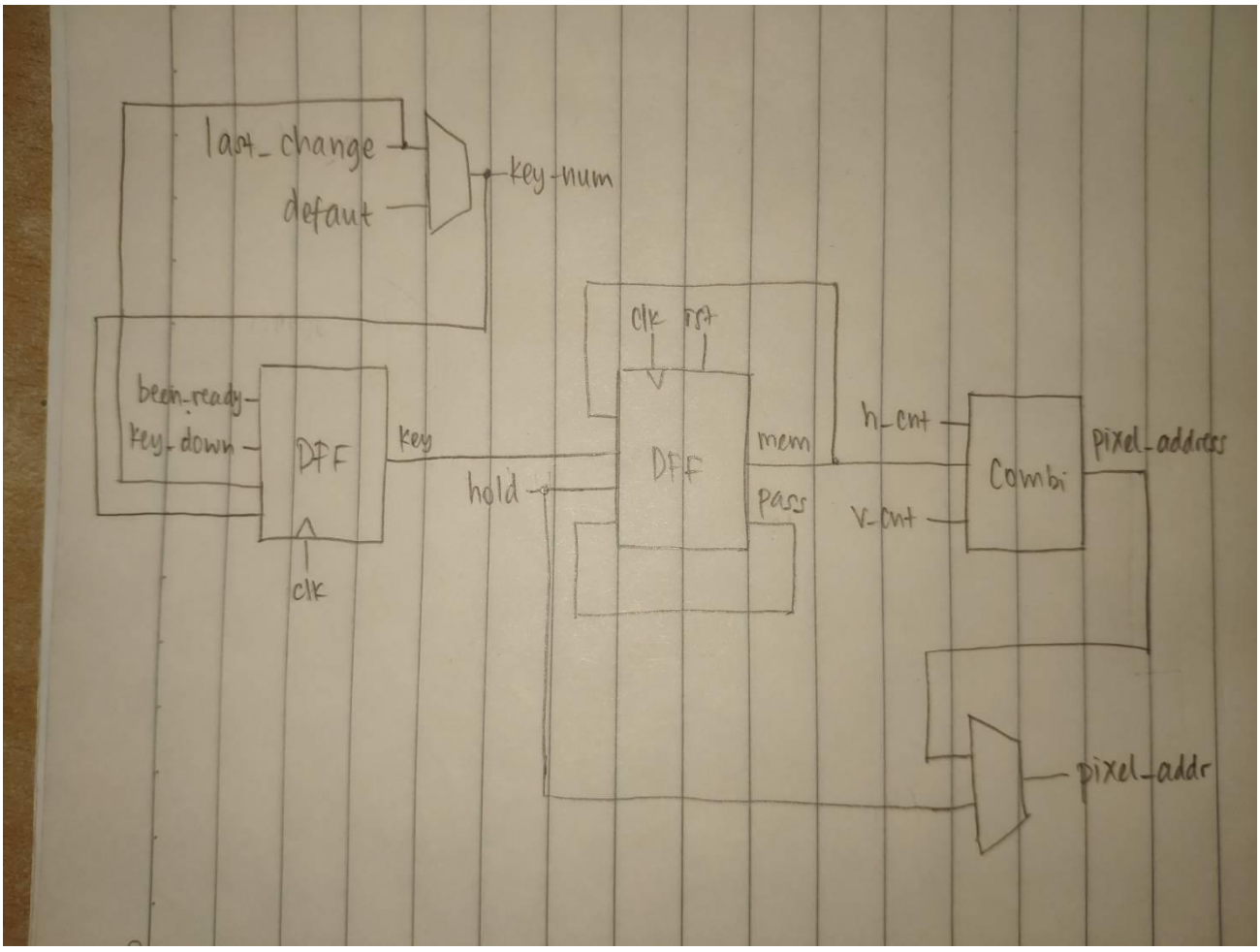
```
assign pixel_neg = 12'hFFF - pixel;

always @(*) begin
    if(nf) {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel_neg:12'h0;
    else {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel:12'h0;
end
```

To do the negative film, I subtract 12-bit FFF hexadecimal by **pixel** (the original RGB value of the image) because as it mentioned above, negative film is the color complement from the original color. Then I store the RGB value for negative color in **pixel_neg** reg beforehand. Then I assign the color of the image, **{vgaRed, vgaGreen, vgaBlue}** as **pixel_neg** when **nf** signal is '1'.

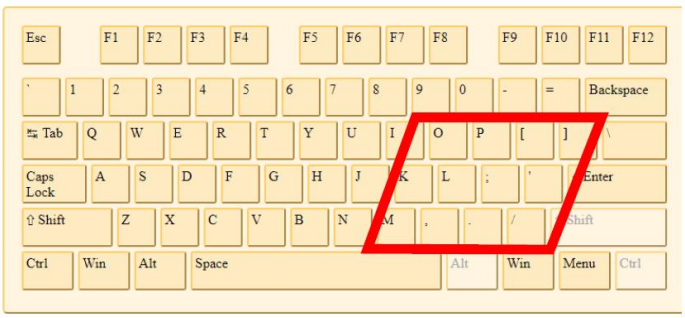


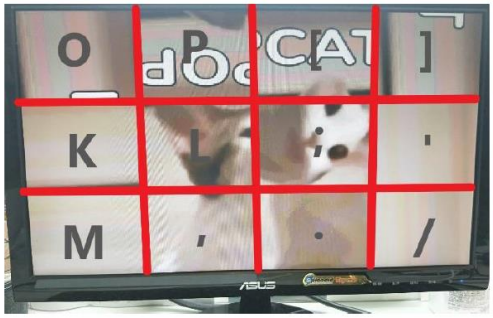
- lab7_2



The goal of this lab is to

Design a VGA game controller that divides your image into 4*3 blocks. You can rotate each block by the keyboard. The block turns 90° when you push the corresponding button. The following figures show the key mapping.





To divide the image become 12 pieces, I make 12-bit reg where every bit can store 2-bit reg (inspirated from **KEY_CODES** parameter in previous lab). I make it 2-bit because it shows 4 different types of binary code. Since rotating the image 90° counterclockwise is the same as rotating it 270° clockwise.

- 2'b00 : original position
- 2'b01 : rotate 90° clockwise (from original picture)
- 2'b10 : rotate 180° clockwise (from original picture)
- 2'b11 : rotate 270° clockwise (from original picture)

Because the initial image shows different position (rotation) of each piece of picture, I just randomly initial the rotation position of 12 pieces of picture.

```
reg [1:0] mem [0:11] = {
    2'b01, 2'b11, 2'b11, 2'b10,
    2'b00, 2'b01, 2'b10, 2'b01,
    2'b10, 2'b00, 2'b10, 2'b11
};
```

- **rst**

  rst: The VGA display will show the image at the origin position after the positive-edge-triggered reset. You may set the initial direction of each block as you like.

  The direction of each block is depending on **mem** value that I have explained above. And each of the mem corresponding to each block, shown as below.

```
always@(posedge clk or posedge rst) begin
    if(rst) begin
        mem[0] <= 2'b01;
        mem[1] <= 2'b11;
        mem[2] <= 2'b11;
        mem[3] <= 2'b10;
        mem[4] <= 2'b00;
        mem[5] <= 2'b01;
        mem[6] <= 2'b10;
        mem[7] <= 2'b01;
        mem[8] <= 2'b10;
        mem[9] <= 2'b00;
        mem[10] <= 2'b10;
        mem[11] <= 2'b11;
    end else begin
```

mem[0]	mem[1]	mem[2]	mem[3]
mem[4]	mem[5]	mem[6]	mem[7]
mem[8]	mem[9]	mem[10]	mem[11]

- **hold**

  If **hold** == 1'b1: The screen shows the correct image as *the hint mode*. The block can not be rotated in this mode.

  If **hold** == 1'b0: The screen goes back to *the game mode*.

  - Press the key directly:

    Turn the corresponding block 90° clockwise.

  - Press the key with the shift button:

    Turn the corresponding block 90° counterclockwise. The left-shift and right-shift buttons should both work.

  Because when **hold** button is pressed, it will just show the original picture for a short time (until the button released), then I just assign **pixel_addr** as the normal equation which I copy it copy and modify it a bit from the sample code. But when button is released, it will assign to **pixel_address** reg.

```
assign pixel_addr = (hold) ? ((h_cnt>>1)+320*(v_cnt>>1)) % 76800 : pixel_address;
```

- **pixel_address**

  pixel_address change according to the coordinate of the pixel, which are **h_cnt** and **v_cnt**. Because we need to make the picture twice bigger, each pixel that has been read will be use for 4 pixels. Like making 2x2 pixel become 4x4 pixel, so the same pixel will be used for 2x2 times it that area, therefore, to make it easy, we can just directly shift one bit to the right or divided it by 2 and make it as set, (**h_cnt >>1**) and (**v_cnt>>1**).

0,0	0,1
1,0	1,1

→

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,1	3,2

Every (**h_cn, v_cnt**) in 4x4 table will read the coordinate of ((**h_cnt >>1**), (**v_cnt>>1**)) from 2x2 picture (original size). For this reason, I just use (**h_cnt >>1**) and (**v_cnt>>1**) directly in my code to make it easy.

```
always@(*) begin
    pixel_address = ((h_cnt>>1)+320*(v_cnt>>1)) % 76800;
    pos = 2'd0;
    flip = 2'd0;
    if ((v_cnt>>1) < 80) begin
        if ((h_cnt>>1) > 240) pos = 2'd3;
        else if ((h_cnt>>1) > 160) pos = 2'd2;
        else if((h_cnt>>1) > 80) pos = 2'd1;

        if((h_cnt>>1) < 80) begin
            pos = 2'd0;
            flip = mem[0];
        end else if ((h_cnt>>1) < 160) begin
            pos = 2'd1;
            flip = mem[1];
        end else if ((h_cnt>>1) < 240) begin
            pos = 2'd2;
            flip = mem[2];
        end else begin
            pos = 2'd3;
            flip = mem[3];
        end

        if((flip == 2'b01)) pixel_address = (((v_cnt>>1) + 80 * pos) + 320 * (((240 * (pos+1))-1) - (h_cnt>>1))) % 25600;
        else if(flip == 2'b10) pixel_address = (320 * (79 + (2*80*pos) - (v_cnt>>1)) + (79 + (2*80*pos) - (h_cnt>>1))) % 25600;
        else if(flip == 2'b11) pixel_address = ((80 * (pos+1) - 1) - ((v_cnt>>1)) + 320 * ((h_cnt>>1) + 80)) % 25600;
```

To compute **pixel_address** according to its location, I found a formula to do it. I compute it manually by hand found those formula. So, it's kind of hard to explain how it works but it's correct when the coordinate is input there.

80	flip = mem[0] pos = 0	flip = mem[1] pos = 1	flip = mem[2] pos = 2	flip = mem[3] pos = 3
	flip = mem[4] pos = 0	flip = mem[5] pos = 1	flip = mem[6] pos = 2	flip = mem[7] pos = 3
160	flip = mem[8] pos = 0	flip = mem[9] pos = 1	flip = mem[10] pos = 2	flip = mem[11] pos = 3
240				
	80	160	240	320

To use the formula, I need to assign some reg to keep in track the block position that look like the table above. The block size is 80x80 and the horizontal size is 320. So, I need to make make 4 if else case to get **pos** and **flip** value. The reason I mod it by 25600 is because the total pixel in a line of 1x4 block is equal to 80*320 = 25600 pixel and also by doing the mod, I could use the same formula for the line of block, not only for **mem[0 to 3]**, but all.

Then for the next line of block, it's the same but I just simply modify **pixel_address** part a bit. For the second line of block where 80 <= (**v_cnt >> 1**) < 160, I use same formula as before but added 25600. Actually it's the same way as changing **position** in the lab7_1 before.

```
if(flip == 2'b01) pixel_address = 25600 + (((((v_cnt>>1) % 80) + 80 * pos) + 320 * (((240 * (pos+1))-1) - (h_cnt>>1))) % 25600);
else if(flip == 2'b10) pixel_address = 25600 + ((320 * (79 + (2*80*pos) - ((v_cnt>>1) % 80)) + (79 + (2*80*pos) - (h_cnt>>1))) % 25600);
else if(flip == 2'b11) pixel_address = 25600 + ((((80 * (pos+1) - 1) - ((v_cnt>>1) % 80) + 320 * ((h_cnt>>1) + 80)) % 25600));
```

For 160 <= (**v_cnt >> 1**) < 240, I added it by 51200.

```
if(flip == 2'b01) pixel_address = 51200 + (((((v_cnt>>1) % 80) + 80 * pos) + 320 * (((240 * (pos+1))-1) - (h_cnt>>1))) % 25600);
else if(flip == 2'b10) pixel_address = 51200 + ((320 * (79 + (2*80*pos) - ((v_cnt>>1) % 80)) + (79 + (2*80*pos) - (h_cnt>>1))) % 25600);
else if(flip == 2'b11) pixel_address = 51200 + (((80 * (pos+1) - 1) - ((v_cnt>>1) % 80) + 320 * ((h_cnt>>1) + 80)) % 25600);
```

- Keyboard button

First, I define the hexadecimal of the keys that will be used according to the code in lecture note.

```
parameter [8:0] LEFT_SHIFT_CODES  = 9'b0_0001_0010;
parameter [8:0] RIGHT_SHIFT_CODES = 9'b0_0101_1001;
parameter [8:0] KEY_CODES [0:11] = {
    9'h44,  // O => 44
    9'h4D,  // P => 4D
    9'h54,  // [ => 54
    9'h5B,  // ] => 5B
    9'h42,  // K => 42
    9'h4B,  // L => 4B
    9'h4C,  // ; => 4C
    9'h52,  // ' => 52
    9'h3A,  // M => 3A
    9'h41,  // , => 41
    9'h49,  // . => 49
    9'h4A   // / => 4A
};
```

shift_down reg to check whether left shift or right shift button is pressed or not.

```
assign shift_down = (key_down[LEFT_SHIFT_CODES] == 1'b1 || key_down[RIGHT_SHIFT_CODES] == 1'b1) ? 1'b1 : 1'b0;
```

key to check the others button ('o', 'p', '[' , ']', 'k', 'l', ';' , "'", 'm', ',', '.', '/'). And it depends on the changes of **key_num.**

Same as the previous lab, **key_num** value is based on **last_change**. If key other than ('o', 'p', '[' , ']', 'k', 'l', ';' , "'", 'm', ',', '.', '/') is pressed, **key_num** will be the default value and **key** won't be updated.

```
always @ (*) begin
    case (last_change)
        KEY_CODES[00] : key_num = 4'b0000;
        KEY_CODES[01] : key_num = 4'b0001;
        KEY_CODES[02] : key_num = 4'b0010;
        KEY_CODES[03] : key_num = 4'b0011;
        KEY_CODES[04] : key_num = 4'b0100;
        KEY_CODES[05] : key_num = 4'b0101;
        KEY_CODES[06] : key_num = 4'b0110;
        KEY_CODES[07] : key_num = 4'b0111;
        KEY_CODES[08] : key_num = 4'b1000;
        KEY_CODES[09] : key_num = 4'b1001;
        KEY_CODES[10] : key_num = 4'b1010;
        KEY_CODES[11] : key_num = 4'b1011;
        default       : key_num = 4'b1111;
    endcase
end
```

```
always @(posedge clk) begin
    if (been_ready && key_down[last_change] == 1'b1) begin
        if (key_num != 4'b1111)begin
            key <= key_num;
        end else key <= 4'b1111;
    end else key <= 4'b1111;
end
```

Then, to track the rotation position, I code it like below.

```
if(key != 4'b1111 && !pass && !hold) begin
    if(shift) mem[key] <= mem[key] - 2'b01;
    else mem[key] <= mem[key] + 2'b01;
end
```

As I mention before, **mem** store 4 type of rotation and I update it base on which key is pressed and add or subtract **mem** by one according to the **shift** button. If **shift** is '1', the image will rotate counter clockwise, that's why I subtract it by one. And when **shift** key is not pressed, the image rotate clockwise, and **mem** will be added by one.

- Pass

  When all the blocks rotate to the correct direction, set the pass signal to 1'b1 to indicate that the mission is accomplished. Also, disable the block rotation so that no block can be rotated further.

  If all blocks rotate to the correct position, it means that all of the **mem** value is '0'. So, I make a condition when all 12 **mem** is '0', pass signal will be '1' and **mem** won't be updated when if the **pass** signal is '1'.

```
if(key != 4'b1111 && !pass && !hold) begin
    if(shift) mem[key] <= mem[key] - 2'b01;
    else mem[key] <= mem[key] + 2'b01;
end
if(mem[0] == 2'b00 && mem[1] == 2'b00 && mem[2] == 2'b00 && mem[3] == 2'b00 && mem[4] == 2'b00 &&
mem[5] == 2'b00 && mem[6] == 2'b00 && mem[7] == 2'b00 &&  mem[8] == 2'b00 && mem[9] == 2'b00 &&
mem[10] == 2'b00 && mem[11] == 2'b00) pass <= 1'b1;
else pass <= 1'b0;
```

2. 學到的東西與遇到的困難

The hardest part for this lab rotating the image in lab7_2. It's really time consuming to figure it out how to rotate. I do a lot of trial and error while doing it and every time I change my code, I need to spend a couple of minutes to generate the bitstream. But the problem was, I kept repeating it again and again, change code and then generate bitstream.

First, I try to rotate the whole picture (640x480 pixel). And I figured it out how to rotate it. However, it didn't work when I apply to all smaller piece of picture (only few pieces of picture block works). Therefore, I change my plan and try to find a way to rotate the picture piece by piece.

Then I started it by counting it by hand. To do it, of course I need to count it using a small pixel as sample. I try to rotate a small picture, 8x4 size picture. And I try to rotate 2x2 size in the picture. Since it's a small size picture, I just simply draw the table contain the pixel address and do some math counting

to gain the pixel address (rotated picture). After spending a lot of my time, I finally figured how to rotate a line of block in the picture. And I just simply use the same method to implement it for the next line of block and adjust or modify it a bit.

3. 想對老師或助教說的話