Logic Design Lab 9

Experiment 1

VGA displaying functions.

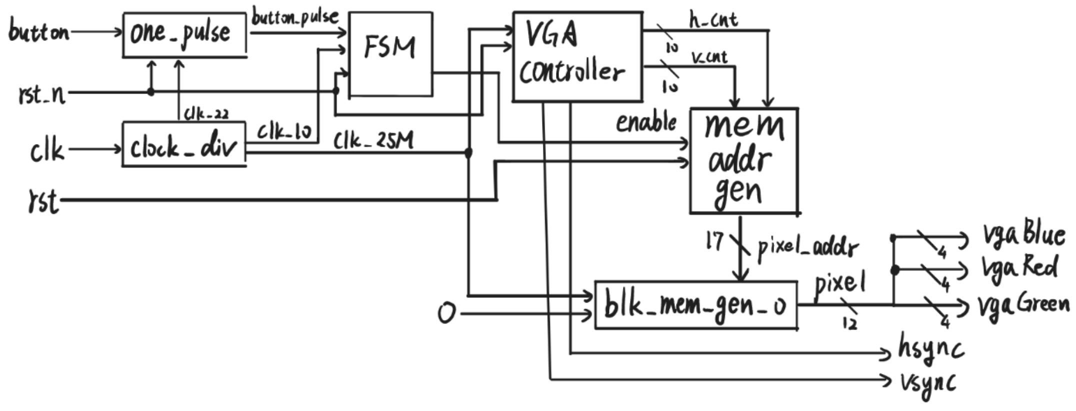
1.1 Inputs of the VGA controller are clk, reset, en and outputs of the VGA controller are hsync, vsync, vga_red[3:0], vga_green[3:0], vga_blue[3:0].

1.2 At the beginning or when reset (button) is pressed, the VGA display shows the image (e.g. amumu.jpg). The VGA image stay still until en (button) is pressed.

1.3 Pressing odd times en button to start/resume scrolling. Pressing even times en button to pause scrolling. Counter for en press is reset to zero when reset is pressed.

Design Specification：

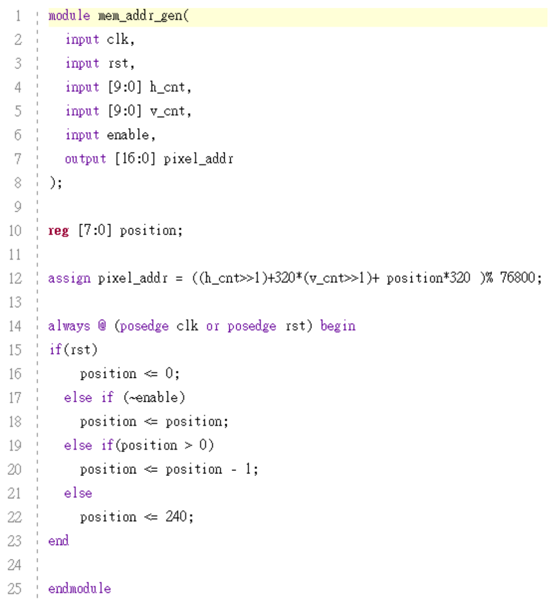
Input: button, rst(reset button), clk, rst_n.
Output: hsync, vsync, vgaBlue, vgaRed, vgaGreen.
Block diagram:



Design Implementation：

1. One pulse, clock divisor：
   We have implemented in the previous labs. Skip.
2. VGA controller：
   Following the reference code on the handout.

3. Memory address generator：

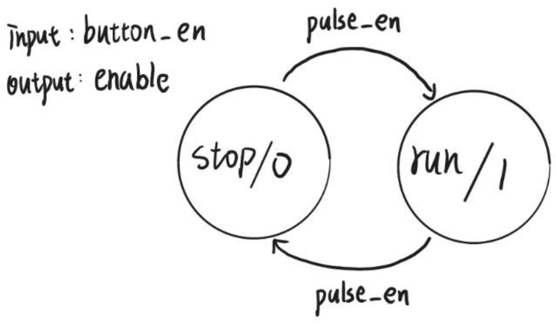


```
module mem_addr_gen(
  input clk,
  input rst,
  input [9:0] h_cnt,
  input [9:0] v_cnt,
  input enable,
  output [16:0] pixel_addr
);

reg [7:0] position;

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

always @ (posedge clk or posedge rst) begin
if(rst)
    position <= 0;
  else if (~enable)
    position <= position;
  else if(position > 0)
    position <= position - 1;
  else
    position <= 240;
end

endmodule
```

4. Memory：

   We use the IP of the memory from the Vivado.

5. FSM：



Discussion：

1. Blk_mem_gen：

   First, we follow the process on the handout to construct memory IP on the Vivado. And, we move on to use the code (PicTrans) provided by Tas to help us to translate the document form from jpg to coe. Finally, we can store the information of the picture in the memory.

2. Memory address generator：

   We set the position following the clock(22Hz) and use this parameter to generate the real pixel address on the screen.

Conclusion：

Although this experiment only requires us to get familiar to what VGA functions and add the FSM as well as pulse function to this problem, it still took me a lot of time to have better insight of the knowledge behind the VGA display.

Besides, the latter experiment is a little bit complicated, we cannot solve lab9_2 without completing this experiment first.

Experiment 2

Calculator display.

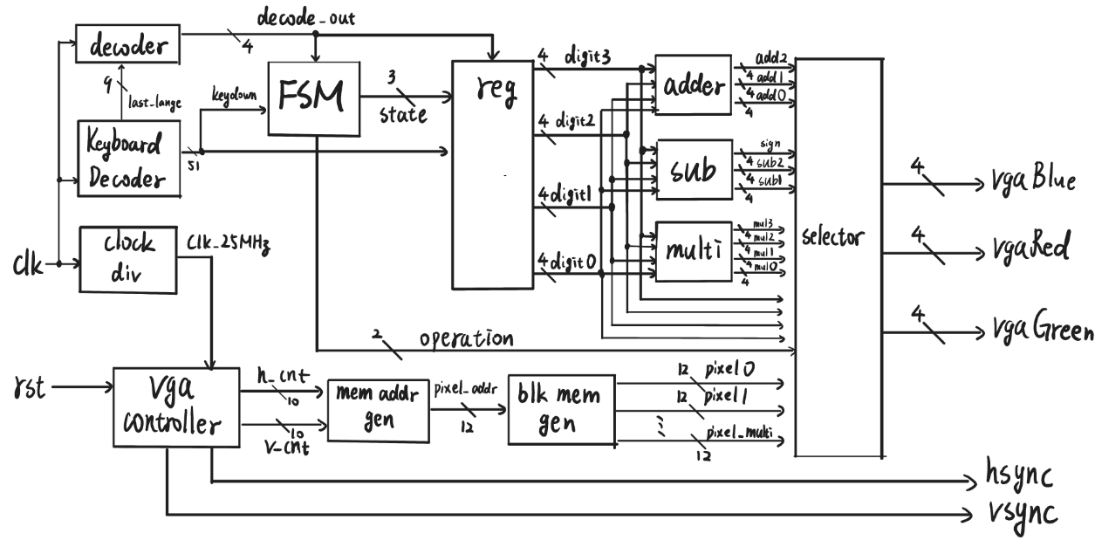
Combine the key board controller and VGA displaying controller to design a calculator with 2-digit addition/subtraction/multiplication. The display function should be the same as usual calculator or APP in the smartphone.

Design Specification：

Input: ps2_clk, ps2_data, clk, rst.

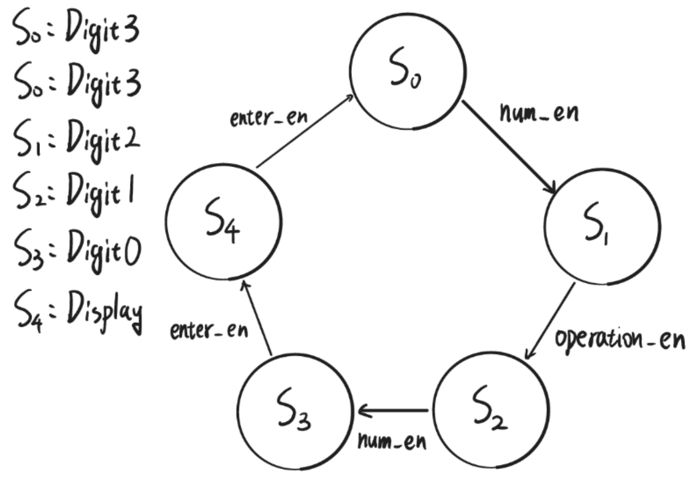
Output: [7:0] seg, [3:0] ssd, [1:0] state.

Block diagram:



Design Implementation：

1. FSM：



2. Register (partial)：

```
else if(state_ctrl == `STAT_DIGIT0)
    if(in == `KEY_ADD | in == `KEY_SUBTRACT | in == `KEY_MULTIPLY | in ==`KEY_ENTER)
        begin
            digit3_tmp = digit3;
            digit2_tmp = digit2;
            digit1_tmp = digit1;
            digit0_tmp = digit0;
        end
    else
        begin
            digit3_tmp = digit3;
            digit2_tmp = digit2;
            digit1_tmp = digit1;
            digit0_tmp = in;
        end
```

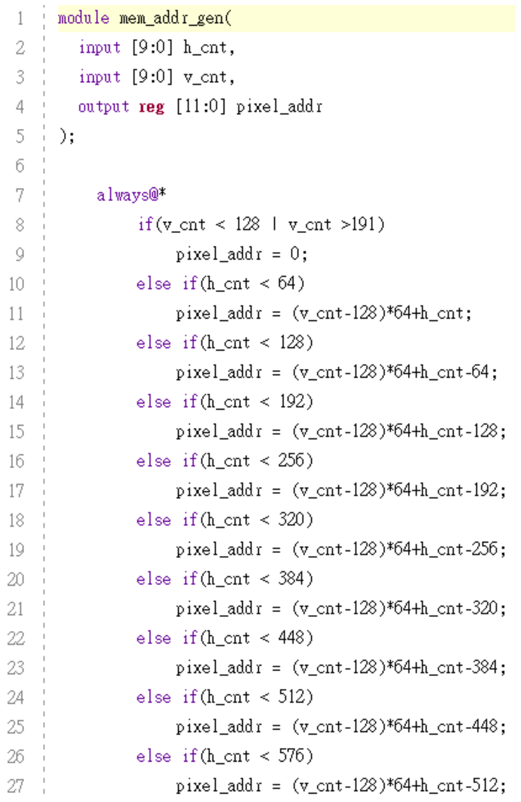
3. Addition, subtraction, multiplication：

   Similar to the function we implemented in lab8_3.

4. Selector (partial)：

```
if(v_cnt<128 | v_cnt>255)
    {vgaRed, vgaGreen, vgaBlue} = 12'h0;
else if(h_cnt<64)
    case(digit3)
        4'd0: {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel0:12'h0;
        4'd1: {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel1:12'h0;
        4'd2: {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel2:12'h0;
        4'd3: {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel3:12'h0;
        4'd4: {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel4:12'h0;
        4'd5: {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel5:12'h0;
        4'd6: {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel6:12'h0;
        4'd7: {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel7:12'h0;
        4'd8: {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel8:12'h0;
        4'd9: {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel9:12'h0;
        default: {vgaRed, vgaGreen, vgaBlue} = 12'h0;
    endcase
```

5. Memory address generator (partial)：



```
module mem_addr_gen(
  input [9:0] h_cnt,
  input [9:0] v_cnt,
  output reg [11:0] pixel_addr
);

    always@*
        if(v_cnt < 128 | v_cnt >191)
            pixel_addr = 0;
        else if(h_cnt < 64)
            pixel_addr = (v_cnt-128)*64+h_cnt;
        else if(h_cnt < 128)
            pixel_addr = (v_cnt-128)*64+h_cnt-64;
        else if(h_cnt < 192)
            pixel_addr = (v_cnt-128)*64+h_cnt-128;
        else if(h_cnt < 256)
            pixel_addr = (v_cnt-128)*64+h_cnt-192;
        else if(h_cnt < 320)
            pixel_addr = (v_cnt-128)*64+h_cnt-256;
        else if(h_cnt < 384)
            pixel_addr = (v_cnt-128)*64+h_cnt-320;
        else if(h_cnt < 448)
            pixel_addr = (v_cnt-128)*64+h_cnt-384;
        else if(h_cnt < 512)
            pixel_addr = (v_cnt-128)*64+h_cnt-448;
        else if(h_cnt < 576)
            pixel_addr = (v_cnt-128)*64+h_cnt-512;
```

Discussion：

1. Enable control：a simple combination circuit to control FSM.

2. Register：
   We follow the present state to decide what digit should be written on or be hold.

3. Selector：
   This is the most complicated parts of the whole experiment. we need to consider the present state, and the position on the screen to assign different values of the pixel including red, blue, and green with memory IPs.

4. Memory address generator：
   We divide the screen to 10 parts (8 digits, 1 operator, 1 base) to let the pattern show on the screen following the way we want.

Conclusion：

This experiment takes a lot of time to finish that, because the VGA control itself is more complicated than we have learned in the previous labs. First, I cannot do a good job at the part of selector (mentioned in discussion) so I decided to browse the old lab reports from our predecessor. Then, it dawned on me that I know how to correct my code. However, before I find the answer, this problem took me about 8 hours.

Besides, since we need to show all information on the screen, we need to be more careful to design the FSM, operation, and clear function to make output directly.

Experiment 3

TETRIS element generator.

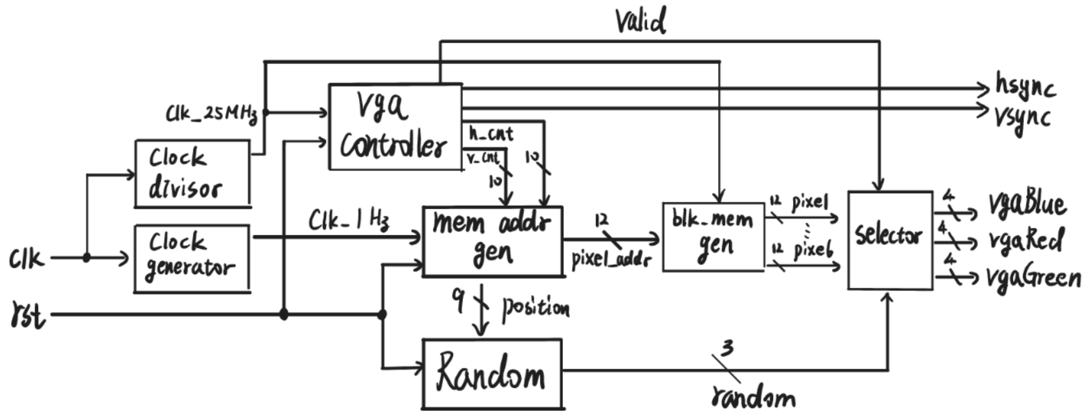
3.1 Generate basic elements of TETRIC (as follows) randomly in the VGA monitor, and plot each of them in the center of the first row of the display, which is a 10 x 20 (WxH) square 2D playing space.

3.2 Each generated basic elements moves down by the step of a square at the speed of1Hz. Finally, they disappear below the playing space. When a basic element disappears, a new basic element is generated again and fall down again repeatedly.

3.3 (Bonus) The same function of 3.1 and 3.2 are designed except that basic elements are stacked up until they are higher than the height of the playing space

Design Specification：

Input: ps2_clk, ps2_data, clk, rst.
Output: [7:0] seg, [3:0] ssd, [3:0] state.
Block diagram:



Design Implementation：

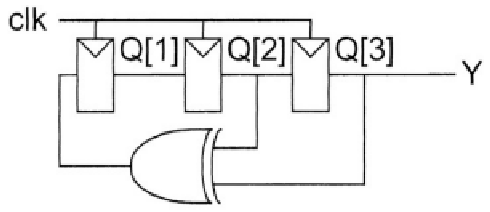
1. Memory address generation (partial)：

```verilog
reg [8:0] position_next;

always@*
   if(h_cnt<288 | h_cnt>351)
     pixel_addr = 0;
   else if(v_cnt<0+position | v_cnt>64+position)
     pixel_addr = 0;
   else pixel_addr = h_cnt-288 + (v_cnt-position)*64;

always@*
   if(rst)
     position_next = 0;
   else if(position < 416)
     position_next = position + 32;
   else
     position_next = 0;


always@(posedge clk or posedge rst)
     if(rst)
         position <= 8'b0;
     else position <= position_next;

endmodule
```

2. Selector (partial)：

```verilog
always@*
    if(h_cnt<192 | h_cnt>447)
        {vgaRed , vgaGreen , vgaBlue} = 12'h0;
    else if(random == 3'd0)
        if((v_cnt<position | v_cnt>position+64)&valid)
            {vgaRed , vgaGreen , vgaBlue} = 12'hfff;
        else if((h_cnt<288 | h_cnt>351)&valid)
            {vgaRed , vgaGreen , vgaBlue} = 12'hfff;
        else
            {vgaRed , vgaGreen , vgaBlue} = pixel0;
    else if(random == 3'd1)
        if((v_cnt<position | v_cnt>position+64)&valid)
            {vgaRed , vgaGreen , vgaBlue} = 12'hfff;
        else if((h_cnt<288 | h_cnt>351)&valid)
            {vgaRed , vgaGreen , vgaBlue} = 12'hfff;
        else
            {vgaRed , vgaGreen , vgaBlue} = pixel1;
    else if(random == 3'd2)
        if((v_cnt<position | v_cnt>position+64)&valid)
            {vgaRed , vgaGreen , vgaBlue} = 12'hfff;
        else if((h_cnt<288 | h_cnt>351)&valid)
            {vgaRed , vgaGreen , vgaBlue} = 12'hfff;
        else
            {vgaRed , vgaGreen , vgaBlue} = pixel2;
```

3. Random：



(Copied from the handout)

Discussion：

1. Memory address generation：
   Similar to lab9_1, we let the pattern move from the top of the screen to the bottom of the screen. Besides, when the pattern moves to the bottom of the screen, we regenerate the cycle again.

2. Selector：
   The core idea of the selector is the same as lab9_2. We need to rewrite the code to fit the condition we want.

3. Random：
   We use Linear Feedback Shift Register to generate our random function. However, this random is Pseudo randomness, but we can add more shift register to the LFSR to create more random functions.

Conclusion：

This experiment is the last lab of this course. After completing lab9_2, we get more familiar with what VGA works and it helped us to finish the experiment with more ease. However, I don’t do the bonus question because I heard of that this question may take us more time than lab9_2 and I have some assignments and tests during the week. Thus, I decided to give up finishing the bonus and save time to prepare for other subjects and final project.

Reference：

1. The handout and the assignment of logic design
   To review some basic concept of the combination circuit and the problem bulls and cows.
2. The handout of logic design lab
   To program our FPGA board by following the method on the handout of logic design lab.