CS207 Digital Design Project Report

The group leader: 傅伟堡

Group member: 蔡于飞 程千帆 崔俞崧

Operating instructions:

Three switches (Y9-task1, W9-task2, Y7-task3) correspond to three modes.
In task1 (Y9) mode: a switch (Y8) controls whether the ribbon rolls or not.
In task2 (W9) mode: one switch (Y8) controls the screensaver, four switches control four different resolutions (AB8-640 *480, AA8-600 *800, V8-1024 *768, V9-1280 *720), five switches control the movement of the image (P5-up, P1-left, P2-down, R1-right, P4-center), and one switch (W4) controls the stretching of the image.
In task3 (Y7) mode: a switch (Y8) control screen saver features, four switch control of four different image/video size (AB8 - 384 * 384, AA8 - 270 * 384, V8 - 200 * 200, V9 - 50 * 50), five switch to control the movement of image/video (P5 - up, P1 - left, P2 - down, R1 - right, P4 - center), a switch (W4) control image/video stretching operation.

Part 1-Development plan & Execution record

- Task allocation
  - [ ] To output 640*480 color strip image **(傅伟堡)**
  - [ ] Use existing files to display images **(崔俞崧)**
  - [ ] Use Uart port for real time images' transmission **(傅伟堡、蔡于飞)**
  - [ ] Analyse Universal Asynchronous Receiver/Transmitter and use Python and Java to send UART signals to board **(蔡于飞)**
  - [ ] To implement the function of resolution switching **(程千帆、崔俞崧)**
  - [ ] Extending resolution switching function to Task3 **(蔡于飞)**
  - [ ] Graphical interface design **(程千帆)**
  - [ ] To implement the function of color bar rolling **(程千帆)**
  - [ ] The implementation of images' free movement and a key center **(傅伟堡)**
  - [ ] Make the images and video to fill the screen **(程千帆、傅伟堡)**

- [ ] Draw frames from videos and convert them to .coe or bitstream files **(蔡于飞、傅伟堡)**
- [ ] To implement the function of screen saver **(程千帆)**
- [ ] Use Uart port for real time videos' transmission **(傅伟堡、蔡于飞)**
- [ ] Develop Java top front end - "VGA CU" **(蔡于飞)**
- [ ] To simulate the children module **(崔俞崧、程千帆)**
- [ ] Record the process of project and write report **(崔俞崧、傅伟堡)**

- Record
  - [x] On December 8, eight 640 * 480 colored ribbon images were successfully displayed.
  - [x] On December 9-10, the image cannot be output on the display screen, or the image appears misaligned and the image scrolls after output.
  - [x] On December 11, the coe file image was successfully output to the display screen, and the image moving function was completed.
  - [x] On December 12, the Python script was completed so that the file formats could be converted to each other.
  - [x] December 12-15, encountered difficulties in Verilog code implementation, no progress.
  - [x] On December 16, the resolution switching and automatic image movement functions were successfully implemented.
  - [x] On December 17, the incoming video file was successfully implemented and displayed.
  - [x] On December 18, the real-time transmission and display from the Uart port, the stretching function of the image, and the scrolling function of eight color ribbons were successfully implemented.
  - [x] December 19-20, complete the java front-end console and integrate all modules.

Part 2-Design

Demand Analysis

1. Identified system functions

In this project, our team used FPGA board to realize a simple VGA graphics card, and expanded and realized some other functions based on FPGA in addition to the basic tasks required.

- *Basic Functions*

(1) Output color strip images with resolution of 640*480 to the display using the FPGA chip on the development board and the VGA output interface.

(2) Use the provided image file to display the image on the screen successfully.

(3)

A: Input text from the serial port to the FPGA board using the onboard USB-UART port and the text output can be displayed on the screen instantly.

B: Use the onboard USB-UART serial port to input pictures from the serial port to the FPGA board and display them on the screen in time.

(4) On the basis of the above functions, the switch between different resolutions of VGA is realized. The keys on the development board are used to control the resolution switch, and the current system resolution is displayed on the digital tube.

- *Expansion Functions*

(1) Through the onboard USB-UART serial port, input video from the serial port to the FPGA board, and be able to see smooth video on the screen.

(2) Screen saver. When a picture or video is entered through the serial port, it can be used as a screensaver, that is, the picture or video can be moved freely on the screen, imitating the screen saver state entered when the computer has not been operated for a long time.

(3) Picture movement function. After a picture is entered, the position of the picture on the screen can be moved by the up, down, left and right buttons on the development board, and the image can be centered by the center button in the middle.

(4) Stretch tile function. The image can automatically adapt to the screen size, when the image size is not the same as the computer screen size, the image can adapt to the current screen size, automatically fill the entire screen.

(5) Front-end control platform. Our group realized the front-end control platform by integrating Verilog code with Java and python. Through this platform, the control of the development board can be realized. No matter the color stripe image mentioned above, or the information sent by serial port, or the resolution switch, or the realization of the screen saver function, can be operated by the front control platform, greatly reducing the operation difficulty of the VGA video card.

2. Input device

Host:

(1) Hardware platform: X86

(2) Operating system: Windows 10/Windows7

The FPGA board:

Xilinx ARTIX chip, MINISYS board.

3. Output device

Display with VGA interface

4. Port specification

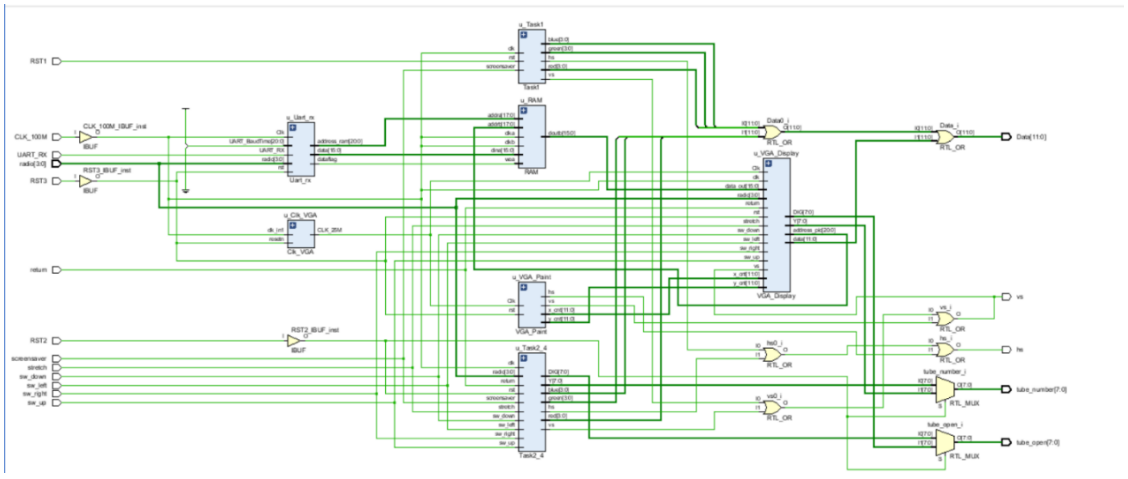
Computer (Windows system) input port: COM serial port

FPGA board receiving port: UART port

The sending port of FPGA board: VGA

The monitor receiving port: VGA parent

- **System Structure Diagram**



Definition of functions of each sub-module and interface between modules

- *1. Clk_VGA module*

  The module inputs clock signal with FPGA crystal oscillator of 100MHz, performs clock frequency division operation, and outputs clock frequency for use by other modules. The output interface of this module functions to provide clock frequency after frequency division to other modules.

- *2. Uart_rx module*

This module is bound to the UART_RX interface to receive the information transmitted by the computer, parse the information through the UART protocol, and output the parsed results to the RAM module. There is an interface between the module and the RAM interface, which is used to make the RAM module read the results after UART_rx module is parsed.

- *3. RAM module*

  This module calls a block of RAM, receives the output of the Uart_rx module, and stores the result in RAM. The module has an interface with the VGA_Display module, which is used to make the VGA_Display module read the access information in the RAM module.

- *4. VGA_paint module*

  The module receives the clock signal and outputs the information representing the pixel position on the screen at any time when the clock signal changes. At the same time output the count information to the VGA_Display module.

- *5. VGA_Display module*

  This module receives the information of RAM module and VGA_paint module, processes the information, and outputs the RGB signal after processing, then displays the RGB signal to the display.

- *6. Task1 module*

  The module input clock signal, through processing, output to the display resolution of 640*480 color strip.

- *7. Task2_4 module*

  The module inputs clock signals and .coe files, and outputs pictures to the display. On this basis, the resolution can be changed, that is, the resolution of the system can be adjusted by different switches.

- **System Execution Process**

  Data flow analysis is used here:

  The execution process of the system is as follows:

  First of all, determine which submodule to execute next based on the values of RST1, RST2, and RST3.

  - Execute task1 when RST1 is valid

    Here the data has input clock signal and screen saver signal.

    First, the input CLK clock signal is processed by frequency division, from 100MHz to 25MHz. Then, row timing and field timing are generated through the CLK signal after the frequency division. The specific method is to use a counter to count and judge when the CLK signal changes, so as

to generate continuously changing row timing and field timing. After generating the row sequence and field sequence, the position of the current pixel on the screen is known through the row sequence and field sequence of the current moment, and then the screen is divided into 8 parts. If the screensaver signal is invalid, you only need to judge the position of the pixel, and each time the pixel is in a different position, you can successfully display a different color. When the screensaver signal is valid, a new counter is created at the same time that the pixel position is determined. This counter can be added from zero to the maximum, then reset and then increase. Once the pixel location is known, it needs to be added to the data in the counter to move the pixel. And so can realize the function that the color block on the display scrolls ceaselessly, namely screen saver function.

- Execute task2 when RST2 is valid

  First of all, through radio, to judge the display resolution of the system, when the input resolution is not the same, process different clock frequency division, the specific way of frequency division and, by contrast, that corresponds to different resolution right clock frequency, thus dividing the CLK signal into different frequency, at the same time, the input of different resolution, will correspond to get the corresponding information on seven segment digital tube. Then, corresponding row timing and field timing are generated with task1.

  Later, when the picture control signals are all invalid, the initial picture is displayed. The image information is initially stored in ROM. After that, the counter is created, and driven by the CLK signal, the counter is incremented, and the corresponding data in the ROM is retrieved by obtaining the memory address of the ROM, which is the color displayed on the screen at the current time. From this, you can fully display the picture in the ROM to the monitor.

  When the picture control information is effective, the initial function needs to be processed. The five inputs, up, down, left, right, and center, control the manual movement of the image. For example, when the upward data is valid, a counter will be generated, which will increase with time. At this point, the position of the pixel is the initial position and the counter data will be added in the vertical direction, so that the upward movement of the picture can be realized. A screensaver that, when valid, also generates a counter, except that the counter is processed before adding to the pixel position. The processing method is as follows: judge the position of the image. If the image is not located at the boundary, the counter data will be increased horizontally and vertically to achieve the oblique movement of

the image. After touching the edge, take the right boundary as an example. And finally, a stretch. When the data is valid, the current resolution information and the screen size are calculated to obtain the ratio of horizontal and vertical stretching, and then the position of each original pixel should be multiplied by the ratio, thus achieving the function of horizontal and vertical stretching.

- Execute task3 when the RST3 signal is valid

  At this point, the CLK signal is the same as task2 signal. Radio data can be used to divide the CLK signal at different frequencies, so as to display at different resolutions. When uart_rx data is entered, it is first processed by the uart_rx module. The specific processing method is that, when the falling edge of uart_rx data is detected, it marks the beginning of data transmission, and a counter and a serial-parallel converter are created. At any time, the change counter keeps increasing to judge whether the input data reaches 8 bits. The input crossing data is converted into parallel data through the converter. When the input data is judged to be terminated, the parallel data is transmitted to the RAM module as the output signal of Uart_rx module for storage. At the same time, the clock signal is input into the VGA_Paint module, which changes over time and outputs the current pixel position on the screen in a similar way to task1. The location data is then entered into the VGA_Display module along with the stored data in RAM. The VGA_Display module is implemented as follows: first, the descending edge of the field is obtained by using the row timing and field timing obtained after the frequency division. Then the clock changes at any time to gradually read the information stored in RAM, and the clock signal is used to monitor the specific location of the current pixel. At this point, after obtaining the location of the current pixel and the color information of the current pixel (read in RAM), the current data can be output. The clock is constantly changing, the pixel position is constantly refreshing, the final result is a slowly emerging picture. When uart_rx data is continuously input here, a picture will be refreshed after it is fully displayed and the next picture will be displayed. Therefore, the video display function can be successfully realized by inputting the information of many frames of a video image here.

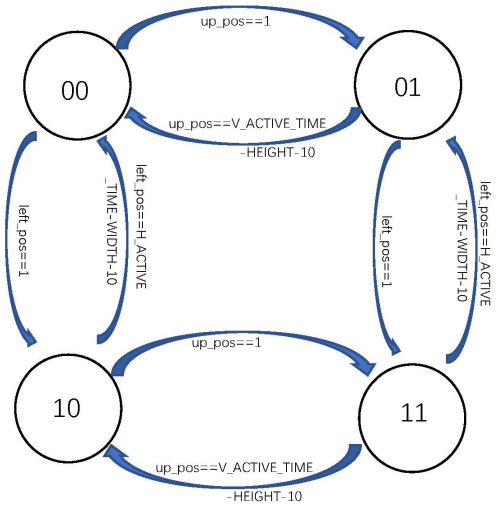
  The module also has image control information. The way to achieve image control here is similar to task2, please refer to the way to achieve task2.

The state transition diagram of screen saver

State transition diagram

This state transition diagram is the screen saver's state transition diagram. There are four states in total, namely: the picture moves to the left and up, corresponding to 00; The picture moves to the left and down, corresponding to 01; The picture moves up to the right, which corresponds to 10; The picture moves down to the right, which is 11.

The variable here corresponds to the variable in the program. The judgment condition is to judge whether the position of the image is located at the edge of the screen. If so, the corresponding operation is performed according to the judgment statement.



The pseudo code of VGA Time Sequence

```
//VGA Time Sequence Pseudo Code
int x,y;
byte screen[x][y][12];//Resolution ratio is x*y.
Field Blanking Interval;
while(hasNextSignal())
{
   Row Blanking Interval;
   for(int i=0;i<x;++i)
   {
```

```
        for(int j=0;j<y;++j)
        {
            screen[x][y]=nextSignal();//Each signal contains 16 bytes in
which the first 12 bytes are valid.
        }
        Row Blanking Interval;
    }
    Field Blanking Interval;
 }
```

- **Submodule Code**
  - **Top module**

```
`timescale 1ns / 1ps

module Top(
input CLK_100M,
input RST1,
input RST2,
input RST3,
input UART_RX,
input [3:0]radio,
input sw_left,
input sw_right,
input sw_up,
input sw_down,
input return,
input screensaver,
input stretch,
output hs,
output vs,
output [11:0] Data,
output [7:0] tube_open,
output [7:0] tube_number
);
wire CLK_25M;
wire rx_done;
wire [11:0] x_cnt;
wire [11:0] y_cnt;
wire [15:0] uart_data;
wire [15:0] data_out;
wire [20:0] address_pic;
wire [20:0] address_ram;
wire hs1,hs2,hs3;
wire vs1,vs2,vs3;
wire [11:0]Data1;
wire [11:0]Data2;
wire [11:0]Data3;
wire [7:0]tube_open1,tube_number1,tube_open2,tube_number2;
wire rst2;

Task1
u_Task1(CLK_100M,RST1,Data1[7:4],Data1[3:0],Data1[11:8],hs1,vs1,screens
aver);
```

```
Task2_4
u_Task2_4(CLK_100M,RST2,radio,sw_left,sw_right,sw_up,sw_down,return,scr
eensaver,stretch,Data2[7:4],Data2[3:0],Data2[11:8],hs2,vs2,tube_open1,t
ube_number1);
Clk_VGA u_Clk_VGA(CLK_25M,RST3,CLK_100M);
VGA_Paint u_VGA_Paint(CLK_25M,RST3,x_cnt,y_cnt,hs3,vs3);
VGA_Display
u_VGA_Display(CLK_100M,CLK_25M,RST3,x_cnt,y_cnt,data_out,sw_left,sw_rig
ht,sw_up,sw_down,return,vs,radio,stretch,address_pic,Data3,tube_open2,t
ube_number2);
Uart_rx
u_Uart_rx(CLK_100M,RST3,16'd103,UART_RX,radio,address_ram,uart_data,rx_
done);
RAM
u_RAM(CLK_100M,rx_done,address_ram,uart_data,CLK_100M,address_pic,data_
out);

assign vs = vs1|vs2|vs3;
assign hs = hs1|hs2|hs3;
assign Data = Data1|Data2|Data3;
assign rst2 = RST2;
assign tube_open = (rst2)? tube_open1 : tube_open2;
assign tube_number = (rst2)? tube_number1 : tube_number2;
endmodule
```

- ## VGA_Paint module

```
`timescale 1ns / 1ps
module VGA_Paint(
input Clk,
input rst,
output [11:0] x_cnt,
output [11:0] y_cnt,
output hs,
output vs
);
//640*480行时序参数
parameter
            H_SYNC_PULSE = 96,
            H_BACK_PORCH = 48,
            H_ACTIVE_TIME = 640,
            H_FRONT_PORCH = 16,
            H_LINE_PERIOD = 800;
//640*480场时序参数
parameter
            V_SYNC_PULSE = 2,
            V_BACK_PORCH = 33,
            V_ACTIVE_TIME = 480,
            V_FRONT_PORCH = 10,
            V_FRAME_PERIOD = 525;
reg [11:0] h_cnt=0;
reg [11:0] v_cnt=0;
wire active_flag;
always @(posedge Clk or negedge rst) begin
    if(!rst)
        h_cnt <= 0;
    else if(h_cnt == H_LINE_PERIOD-1'b1)
```

```
        h_cnt <= 0;
    else h_cnt <=  h_cnt + 1;
end
assign hs = (h_cnt<H_SYNC_PULSE)?1'b0:1'b1;
always @(posedge Clk or negedge rst) begin
    if(!rst)
        v_cnt <= 0;
    else if(v_cnt == V_FRAME_PERIOD - 1'b1)
        v_cnt <= 0;
    else if(h_cnt == H_LINE_PERIOD - 1'b1)
        v_cnt <= v_cnt + 1;
    else v_cnt <= v_cnt;
end
assign vs = (v_cnt<V_SYNC_PULSE)?1'b0:1'b1;
assign active_flag = (h_cnt>=(H_SYNC_PULSE+H_BACK_PORCH))&&
                     (h_cnt<=
(H_SYNC_PULSE+H_BACK_PORCH+H_ACTIVE_TIME))&&
                     (v_cnt>=(V_SYNC_PULSE+V_BACK_PORCH))&&
                     (v_cnt<=
(V_SYNC_PULSE+V_BACK_PORCH+V_ACTIVE_TIME));
assign x_cnt = active_flag?(h_cnt - (H_SYNC_PULSE + H_BACK_PORCH)):0;
assign y_cnt = active_flag?(v_cnt - (V_SYNC_PULSE + V_BACK_PORCH)):0;
endmodule
```

VGA_Display module

```
`timescale 1ns / 1ps
module VGA_Display(
input clk,
input Clk,                          //系统时钟
input rst,                          //复位信号
input [11:0] x_cnt,                 //在显示屏中行的位置
input [11:0] y_cnt,                 //在显示屏中列的位置
input [15:0] data_out,              //RGB数据流
input sw_left,                      //左移动
input sw_right,                     //右移动
input sw_up,                        //上移动
input sw_down,                      //下移动
input return,                       //复原，中间
input vs,                           //场时序
input [3:0] radio,                  //分辨率选择
input stretch,                      //填充
output reg [20:0] address_pic,      //在图片中的地址
output reg [11:0] data,              //12位RGB数据
output [7:0] DIG,                   //控制7段数码管是否亮
output [7:0] Y                      //7段数码管数值
);
reg pic_enable;
reg data_enable;
//640*480行时序参数
parameter
            H_SYNC_PULSE = 96,
            H_BACK_PORCH = 48,
            H_ACTIVE_TIME = 640,
            H_FRONT_PORCH = 16,
            H_LINE_PERIOD = 800;
//640*480场时序参数
```

```
parameter
            V_SYNC_PULSE = 2,
            V_BACK_PORCH = 33,
            V_ACTIVE_TIME = 480,
            V_FRONT_PORCH = 10,
            V_FRAME_PERIOD = 525;

reg[10:0]X_PIXEL,Y_PIXEL;             //图片行和列大小
wire neg_vs;                          //监测下降沿
reg vs_temp1,vs_temp2;                //临时变量，用于监测下降沿
reg[10:0]H_START,H_END,V_START,V_END;    //图片的四个顶点信息，用于绘制有效区域

reg clkout;
reg [31:0] cnt;
reg [2:0] scan_cnt;
reg [6:0] Y_r;
reg [7:0] DIG_r;
assign Y = {1'b1,(~Y_r[6:0])};
assign DIG = ~DIG_r;
parameter period = 200000;
    // frequency division :
always @( posedge clk or negedge rst) begin
    if(!rst) begin
        cnt <= 0;
        clkout <= 0;
    end
    else begin
        if (cnt == (period >> 1) -1) begin
            clkout <= ~clkout;
            cnt <= 0;
        end
        else
            cnt <= cnt+1;
    end
end

//change scan_cnt based on clkout
always @(posedge clkout or negedge rst) begin
    if(!rst)
        scan_cnt <= 0;
    else  begin
        scan_cnt <= scan_cnt + 1;
        if(scan_cnt == 3'd7)
            scan_cnt <= 0;
    end
end
//控制数码管是否亮
always @(scan_cnt) begin
    if( !rst ) DIG_r = 8'b0000_0000;
    else
        case( radio )
                4'b0010, 4'b0100: case( scan_cnt )
                                3'b000 :DIG_r = 8'b0000_0001;
                                3'b001 :DIG_r = 8'b0000_0010;
                                3'b010 :DIG_r = 8'b0000_0100;
                                3'b011 :DIG_r = 8'b0000_0000;
                                3'b100 :DIG_r = 8'b0001_0000;
```

```
                                3'b101 :DIG_r = 8'b0010_0000;
                                3'b110 :DIG_r = 8'b0100_0000;
                                3'b111 :DIG_r = 8'b0000_0000;
                                default : DIG_r = 8'b0000_0000;
                            endcase
                    4'b1000: case( scan_cnt )
                                3'b000 :DIG_r = 8'b0000_0001;
                                3'b001 :DIG_r = 8'b0000_0010;
                                3'b010 :DIG_r = 8'b0000_0000;
                                3'b011 :DIG_r = 8'b0000_0000;
                                3'b100 :DIG_r = 8'b0001_0000;
                                3'b101 :DIG_r = 8'b0010_0000;
                                3'b110 :DIG_r = 8'b0000_0000;
                                3'b111 :DIG_r = 8'b0000_0000;
                                default : DIG_r = 8'b0000_0000;
                             endcase
                    default: case( scan_cnt )
                                3'b000 :DIG_r = 8'b0000_0001;
                                3'b001 :DIG_r = 8'b0000_0010;
                                3'b010 :DIG_r = 8'b0000_0100;
                                3'b011 :DIG_r = 8'b0000_0000;
                                3'b100 :DIG_r = 8'b0001_0000;
                                3'b101 :DIG_r = 8'b0010_0000;
                                3'b110 :DIG_r = 8'b0100_0000;
                                3'b111 :DIG_r = 8'b0000_0000;
                                default : DIG_r = 8'b0000_0000;
                            endcase
            endcase
end
//控制数码管的数字
always @ ( scan_cnt ) begin
    if(rst)
    case( radio )
                    4'b0010: case ( scan_cnt ) // 384 * 270
                                0: Y_r = 7'b1100110; // 4
                                1: Y_r = 7'b1111111; // 8
                                2: Y_r = 7'b1001111; // 3
                                3: Y_r = 7'b0000000; // space
                                4: Y_r = 7'b0111111; // 0
                                5: Y_r = 7'b0000111; // 7
                                6: Y_r = 7'b1011011; // 2
                                7: Y_r = 7'b0000000; // space
                            endcase
                    4'b0100: case ( scan_cnt ) // 200 * 200
                                0: Y_r = 7'b0111111; // 0
                                1: Y_r = 7'b0111111; // 0
                                2: Y_r = 7'b1011011; // 2
                                3: Y_r = 7'b0000000; // space
                                4: Y_r = 7'b0111111; // 0
                                5: Y_r = 7'b0111111; // 0
                                6: Y_r = 7'b1011011; // 2
                                7: Y_r = 7'b0000000; // 1
                            endcase
                    4'b1000: case ( scan_cnt ) // 50 * 50
                                0: Y_r = 7'b0111111; // 0
                                1: Y_r = 7'b1101101; // 5
                                2: Y_r = 7'b0000000; // space
                                3: Y_r = 7'b0000000; // space
```

```
                            4: Y_r = 7'b0111111; // 0
                            5: Y_r = 7'b1101101; // 5
                            6: Y_r = 7'b0000000; // space
                            7: Y_r = 7'b0000000; // space
                        endcase
                default: case ( scan_cnt ) // 384 * 384
                            0: Y_r = 7'b1100110; // 4
                            1: Y_r = 7'b1111111; // 8
                            2: Y_r = 7'b1001111; // 3
                            3: Y_r = 7'b0000000; // space
                            4: Y_r = 7'b1100110; // 4
                            5: Y_r = 7'b1111111; // 8
                            6: Y_r = 7'b1001111; // 3
                            7: Y_r = 7'b0000000; // space
                        endcase
    endcase
end
//切换分辨率
always @(posedge Clk) begin
    case ( radio )
    4'b0010 : begin
            //270*384
            X_PIXEL<=270;
            Y_PIXEL<=384;
            end
    4'b0100 : begin
            //200*200
            X_PIXEL<=200;
            Y_PIXEL<=200;
            end
    4'b1000 : begin
            //50*50
            X_PIXEL<=50;
            Y_PIXEL<=50;
            end
     default : begin
            //384*384
            X_PIXEL<=384;
            Y_PIXEL<=384;
            end
     endcase
end
//更改四个顶点的坐标信息
always @(posedge Clk or negedge rst) begin
    if(!rst) begin
        H_START<=(H_ACTIVE_TIME-X_PIXEL)/2;
        H_END<=(H_ACTIVE_TIME+X_PIXEL)/2;
        V_START<=(V_ACTIVE_TIME-Y_PIXEL)/2;
        V_END<=(V_ACTIVE_TIME+Y_PIXEL)/2;
    end
    else if(!neg_vs) begin
        H_START<=H_START;
        H_END<=H_END;
        V_START<=V_START;
        V_END<=V_END;
    end
    else if(return) begin
        H_START<=(H_ACTIVE_TIME-X_PIXEL)/2;
```

```
        H_END<=(H_ACTIVE_TIME+X_PIXEL)/2;
        V_START<=(V_ACTIVE_TIME-Y_PIXEL)/2;
        V_END<=(V_ACTIVE_TIME+Y_PIXEL)/2;
    end
    else if(sw_left) begin
        if(H_START>0) begin
            H_START<=H_START-1;
            H_END<=H_END-1;
        end
    end
    else if(sw_right) begin
        if(H_END<H_ACTIVE_TIME-1) begin
            H_START<=H_START+1;
            H_END<=H_END+1;
        end
    end
    else if(sw_up) begin
        if(V_START>0) begin
            V_START<=V_START-1;
            V_END<=V_END-1;
        end
    end
    else if(sw_down) begin
        if(V_END<V_ACTIVE_TIME-1) begin
            V_START<=V_START+1;
            V_END<=V_END+1;
        end
    end
end

//获得场的下降沿
always @(posedge Clk or negedge rst) begin
    if(!rst) begin
            vs_temp1 <= 0;
            vs_temp2 <= 0;
        end
    else begin
            vs_temp2 <= vs_temp1;
            vs_temp1 <= vs;
        end
end
assign neg_vs = (~vs_temp1)&vs_temp2;
//地址信息
always @(posedge Clk or negedge rst) begin
    if(!rst)
        address_pic <= 19'd0;
    else if(address_pic == X_PIXEL*Y_PIXEL-1)
        address_pic <= 19'd0;
    else if(stretch) begin
            if(x_cnt>=0&& x_cnt<H_ACTIVE_TIME && y_cnt>0
&&y_cnt<=V_ACTIVE_TIME)
                address_pic <= Y_PIXEL * y_cnt / V_ACTIVE_TIME *
X_PIXEL + Y_PIXEL * (x_cnt+1) / H_ACTIVE_TIME;
            else address_pic <= address_pic;
            end
    else if(x_cnt>=H_START&& x_cnt<H_END && y_cnt>V_START
&&y_cnt<=V_END)
            address_pic <= address_pic + 1;
```

```
    else address_pic <= address_pic;
end

//监测绘画区域
always @(posedge Clk or negedge rst) begin
    if(!rst)
        pic_enable <= 1'b0;
    else if(stretch)
        if(x_cnt>=0&& x_cnt<H_ACTIVE_TIME && y_cnt>0
&&y_cnt<=V_ACTIVE_TIME)
            pic_enable <= 1'b1;
        else pic_enable <= 1'b0;
    else if(x_cnt>=H_START && x_cnt<H_END &&y_cnt>
V_START&&y_cnt<=V_END)
        pic_enable <= 1'b1;
    else pic_enable <= 1'b0;
end
//监测数据流有效区域
always @(posedge Clk or negedge rst) begin
    if(!rst)
        data_enable <= 1'b0;
    else if(stretch)
        if(x_cnt>=0&& x_cnt<H_ACTIVE_TIME && y_cnt>0
&&y_cnt<=V_ACTIVE_TIME)
            data_enable <= 1'b1;
        else data_enable <= 1'b0;
    else if(x_cnt>=H_START&& x_cnt<H_END &&
y_cnt>V_START&&y_cnt<=V_END)
        data_enable <= 1'b1;
    else data_enable <= 1'b0;
end
//输出的数据
always @(posedge Clk or negedge rst) begin
    if(!rst)
        data <= 12'd0;
    else if(data_enable==1'b1&&pic_enable==1'b1)
        data <= {data_out[15:12],data_out[7:0]};
    else data <= 12'd0;
end
endmodule
```

- ## Task1 module

```
`timescale 1ns / 1ps
module Task1(clk,rst,red,green,blue,hs,vs,screensaver);
input clk;   //时钟
input rst;   //复位信号
input screensaver;
output reg [3:0] red;   //红色分量
output reg [3:0] green;   //绿色分量
output reg [3:0] blue;   //蓝色分量
output hs;   //行同步信号
output vs;   //场同步信号

//640*480行时序参数
parameter
            H_SYNC_PULSE = 96,
```

```
            H_BACK_PORCH = 48,
            H_ACTIVE_TIME = 640,
            H_FRONT_PORCH = 16,
            H_LINE_PERIOD = 800;
//640*480场时序参数
parameter
            V_SYNC_PULSE = 2,
            V_BACK_PORCH = 33,
            V_ACTIVE_TIME = 480,
            V_FRONT_PORCH = 10,
            V_FRAME_PERIOD = 525;
parameter
            BAR_WIDTH = H_ACTIVE_TIME/8,
            clk_divide_cnt = 4;

reg [11:0] h_cnt=0;   //行时序计数器
reg [11:0] v_cnt=0;   //列时序计数器
reg [25:0]clk_cnt=0;
reg [9:0] hs_cnt;
reg clk_25M=0;
reg vs_temp1,vs_temp2;
wire active_flag;
wire neg_vs;

///产生25MHz的像素时钟
always @(posedge clk or negedge rst) begin
    if(!rst) begin
            clk_cnt <= 0;
            clk_25M <= 1'b0;
    end
    else if(clk_cnt==(clk_divide_cnt>>1)-1) begin
            clk_cnt <= 0;
            clk_25M <= ~clk_25M;
    end
    else clk_cnt <= clk_cnt + 1;
end

///产生行时序
always @(posedge clk_25M or negedge rst) begin
    if(!rst)
        h_cnt <= 12'd0;
    else if(h_cnt == H_LINE_PERIOD - 1'b1)
        h_cnt <= 12'd0;
    else h_cnt <= h_cnt + 1'b1;
end
assign hs = (h_cnt<H_SYNC_PULSE)? 1'b0:1'b1;

///产生场时序
always @(posedge clk_25M or negedge rst) begin
    if(!rst)
        v_cnt <= 12'd0;
    else if(v_cnt == V_FRAME_PERIOD - 1'b1)
        v_cnt <= 12'd0;
    else if(h_cnt == H_LINE_PERIOD - 1'b1)
        v_cnt <= v_cnt + 1'b1;
    else v_cnt <= v_cnt;
end
assign vs = (v_cnt<V_SYNC_PULSE)?1'b0:1'b1;
```

```
always @(posedge clk_25M or negedge rst) begin
    if(!rst) begin
            vs_temp1 <= 0;
            vs_temp2 <= 0;
    end
    else begin
            vs_temp2 <= vs_temp1;
            vs_temp1 <= vs;
    end
end
assign neg_vs = (~vs_temp1)&vs_temp2;

always @(posedge clk_25M or negedge screensaver or negedge rst) begin
    if(!screensaver||!rst) hs_cnt <= 0;
    else begin
            if(neg_vs) begin
                    if(hs_cnt==H_ACTIVE_TIME - 1) hs_cnt<=0;
                    else hs_cnt <= hs_cnt + 1;
            end
    end
end

assign active_flag = (h_cnt>=(H_SYNC_PULSE+H_BACK_PORCH))&&
                     (h_cnt<=
(H_SYNC_PULSE+H_BACK_PORCH+H_ACTIVE_TIME))&&
                     (v_cnt>=(V_SYNC_PULSE+V_BACK_PORCH))&&
                     (v_cnt<=
(V_SYNC_PULSE+V_BACK_PORCH+V_ACTIVE_TIME));

///把显示器分为8个纵列，每个纵列的宽度为80
always @(posedge clk_25M or negedge rst) begin
    if(!rst) begin
            red <= 4'b0000;
            green <= 4'b0000;
            blue <= 4'b0000;
    end
    else if(active_flag) begin
            if((h_cnt + hs_cnt-H_SYNC_PULSE -
H_BACK_PORCH)%H_ACTIVE_TIME <(BAR_WIDTH)) begin
                    red <= 4'b1111;
                    green <= 4'b0000;
                    blue <= 4'b0000;
            end
            else if((h_cnt + hs_cnt-H_SYNC_PULSE -
H_BACK_PORCH)%H_ACTIVE_TIME <(2*BAR_WIDTH)) begin
                    red <= 4'b0000;
                    green <= 4'b1111;
                    blue <= 4'b0000;
            end
            else if((h_cnt + hs_cnt-H_SYNC_PULSE -
H_BACK_PORCH)%H_ACTIVE_TIME <(3*BAR_WIDTH)) begin
                    red <= 4'b0000;
                    green <= 4'b0000;
                    blue <= 4'b1111;
            end
            else if((h_cnt + hs_cnt-H_SYNC_PULSE -
H_BACK_PORCH)%H_ACTIVE_TIME <(4*BAR_WIDTH)) begin
```

```
                    red <= 4'b1111;
                    green <= 4'b1111;
                    blue <= 4'b1111;
            end
            else if((h_cnt + hs_cnt-H_SYNC_PULSE -
H_BACK_PORCH)%H_ACTIVE_TIME <(5*BAR_WIDTH)) begin
                    red <= 4'b0000;
                    green <= 4'b0000;
                    blue <= 4'b0000;
            end
            else if((h_cnt + hs_cnt-H_SYNC_PULSE -
H_BACK_PORCH)%H_ACTIVE_TIME <(6*BAR_WIDTH)) begin
                    red <= 4'b1111;
                    green <= 4'b1111;
                    blue <= 4'b0000;
            end
            else if((h_cnt + hs_cnt-H_SYNC_PULSE -
H_BACK_PORCH)%H_ACTIVE_TIME <(7*BAR_WIDTH)) begin
                    red <= 4'b1111;
                    green <= 4'b0000;
                    blue <= 4'b1111;
            end
            else if((h_cnt + hs_cnt-H_SYNC_PULSE -
H_BACK_PORCH)%H_ACTIVE_TIME <(8*BAR_WIDTH)) begin
                    red <= 4'b0000;
                    green <= 4'b1111;
                    blue <= 4'b1111;
            end
    end
    else begin
            red <= 4'b0000;
            green <= 4'b0000;
            blue <= 4'b0000;
    end
end
endmodule
```

- ## Task2_4 module

```
`timescale 1ns / 1ps
module Task2_4(
input clk,   //时钟
input rst,   //复位信号
input [3:0] radio,    //控制分辨率信号
input sw_left,sw_right,sw_up,sw_down,   //向左\向右\向上\向下移动
input return,   //图片归位
input screensaver,   //屏保信号
input stretch,
output reg [3:0] red,   //红色分量
output reg [3:0] green,   //绿色分量
output reg [3:0] blue,   //蓝色分量
output hs,   //行同步信号
output vs,   //场同步信号
output [7:0] DIG, //控制7段数码管是否亮
output [7:0] Y  //7段数码管数值
);
reg [11:0] h_cnt=0;   //行时序计数器
```

```
reg [11:0] v_cnt=0;   //列时序计数器
reg [11:0] up_pos=0;   //图片上端位置
reg [11:0] left_pos=0;   //图片左方位置
reg [19:0] address;   //地址
wire [11:0] data;   //读取coe数据
reg [25:0]clk_cnt=0;
wire clk_out;
wire clk_25M;
wire clk_40M;
wire clk_65M;
wire clk_75M;
reg vs_temp1,vs_temp2;
reg right, down;
wire neg_vs;
wire active_flag;
wire locked;
reg clkclk;


//640*480行时序参数
reg [11:0]
            H_SYNC_PULSE = 96,
            H_BACK_PORCH = 48,
            H_ACTIVE_TIME = 640,
            H_FRONT_PORCH = 16,
            H_LINE_PERIOD = 800;
//640*480场时序参数
reg [11:0]
            V_SYNC_PULSE = 2,
            V_BACK_PORCH = 33,
            V_ACTIVE_TIME = 480,
            V_FRONT_PORCH = 10,
            V_FRAME_PERIOD = 525;
//图片和时钟参数
reg [11:0]
            WIDTH = 384,
            HEIGHT = 384;
parameter
            PIXEL = 147456,
            clk_divide_cnt = 4;

reg clkout;
reg [31:0] cnt;
reg [2:0] scan_cnt;
reg [6:0] Y_r;
reg [7:0] DIG_r;
assign Y = {1'b1,(~Y_r[6:0])};
assign DIG = ~DIG_r;
parameter period = 200000;
    // frequency division :
always @( posedge clk or negedge rst) begin
    if(!rst) begin
        cnt <= 0;
        clkout <= 0;
    end
    else begin
        if (cnt == (period >> 1) -1) begin
            clkout <= ~clkout;
```

```
            cnt <= 0;
        end
        else
            cnt <= cnt+1;
    end
end

    //change scan_cnt based on clkout
always @(posedge clkout or negedge rst) begin
    if(!rst)
        scan_cnt <= 0;
    else  begin
        scan_cnt <= scan_cnt + 1;
        if(scan_cnt == 3'd7)
            scan_cnt <= 0;
    end
end

    //select tube
always @(scan_cnt) begin
    if( !rst ) DIG_r = 8'b0000_0000;
    else case( radio )
        4'b0010: case( scan_cnt )
                    3'b000 :DIG_r = 8'b0000_0001;
                    3'b001 :DIG_r = 8'b0000_0010;
                    3'b010 :DIG_r = 8'b0000_0100;
                    3'b011 :DIG_r = 8'b0000_0000;
                    3'b100 :DIG_r = 8'b0001_0000;
                    3'b101 :DIG_r = 8'b0010_0000;
                    3'b110 :DIG_r = 8'b0100_0000;
                    3'b111 :DIG_r = 8'b0000_0000;
                    default : DIG_r = 8'b0000_0000;
                endcase
        4'b0100, 4'b1000: case( scan_cnt )
                    3'b000 :DIG_r = 8'b0000_0001;
                    3'b001 :DIG_r = 8'b0000_0010;
                    3'b010 :DIG_r = 8'b0000_0100;
                    3'b011 :DIG_r = 8'b0000_1000;
                    3'b100 :DIG_r = 8'b0001_0000;
                    3'b101 :DIG_r = 8'b0010_0000;
                    3'b110 :DIG_r = 8'b0100_0000;
                    3'b111 :DIG_r = 8'b1000_0000;
                    default : DIG_r = 8'b0000_0000;
                 endcase
        default: case( scan_cnt )
                    3'b000 :DIG_r = 8'b0000_0001;
                    3'b001 :DIG_r = 8'b0000_0010;
                    3'b010 :DIG_r = 8'b0000_0100;
                    3'b011 :DIG_r = 8'b0000_0000;
                    3'b100 :DIG_r = 8'b0001_0000;
                    3'b101 :DIG_r = 8'b0010_0000;
                    3'b110 :DIG_r = 8'b0100_0000;
                    3'b111 :DIG_r = 8'b0000_0000;
                    default : DIG_r = 8'b0000_0000;
                endcase
        endcase
end
```

```
always @ ( scan_cnt ) begin
    if(rst)
    case( radio )
        4'b0010: case ( scan_cnt ) // 800 * 600
                    0: Y_r = 7'b0111111; // 0
                    1: Y_r = 7'b0111111; // 0
                    2: Y_r = 7'b1111101; // 6
                    3: Y_r = 7'b0000000; // space
                    4: Y_r = 7'b0111111; // 0
                    5: Y_r = 7'b0111111; // 0
                    6: Y_r = 7'b1111111; // 8
                    7: Y_r = 7'b0000000; // space
                 endcase
        4'b0100: case ( scan_cnt ) // 1024 * 768
                    0: Y_r = 7'b1111111; // 8
                    1: Y_r = 7'b1111101; // 6
                    2: Y_r = 7'b0100111; // 7
                    3: Y_r = 7'b0000000; // space
                    4: Y_r = 7'b1100110; // 4
                    5: Y_r = 7'b1011011; // 2
                    6: Y_r = 7'b0111111; // 0
                    7: Y_r = 7'b0000110; // 1
                 endcase
        4'b1000: case ( scan_cnt ) // 1280 * 720
                    0: Y_r = 7'b0111111; // 0
                    1: Y_r = 7'b1011011; // 2
                    2: Y_r = 7'b0100111; // 7
                    3: Y_r = 7'b0000000; // space
                    4: Y_r = 7'b0111111; // 0
                    5: Y_r = 7'b1111111; // 8
                    6: Y_r = 7'b1011011; // 2
                    7: Y_r = 7'b0000110; // 1
                 endcase
        default: case ( scan_cnt ) // 640 * 480
                    0: Y_r = 7'b0111111; // 0
                    1: Y_r = 7'b1111111; // 8
                    2: Y_r = 7'b1100110; // 4
                    3: Y_r = 7'b0000000; // space
                    4: Y_r = 7'b0111111; // 0
                    5: Y_r = 7'b1100110; // 4
                    6: Y_r = 7'b1111101; // 6
                    7: Y_r = 7'b0000000; // space
                 endcase
    endcase
end

//调节分辨率
always @( radio ) begin
    case ( radio )
        4'b0010 : begin
                //600*800 行时序参数
                H_SYNC_PULSE <= 128;
                H_BACK_PORCH <= 88;
                H_ACTIVE_TIME <= 800;
                H_FRONT_PORCH <= 40;
                H_LINE_PERIOD = 1056;
                 //600*800 场时序参数
                V_SYNC_PULSE = 4;
```

```
                V_BACK_PORCH = 23;
                V_ACTIVE_TIME = 600;
                V_FRONT_PORCH = 1;
                V_FRAME_PERIOD = 623;
                end
        4'b0100 : begin
                //1024*768 行时序参数
                H_SYNC_PULSE <= 136;
                H_BACK_PORCH <= 160;
                H_ACTIVE_TIME <= 1024;
                H_FRONT_PORCH <= 24;
                H_LINE_PERIOD <= 1344;
                //1024*768 场时序参数
                V_SYNC_PULSE <= 6;
                V_BACK_PORCH <= 29;
                V_ACTIVE_TIME <= 768;
                V_FRONT_PORCH <= 3;
                V_FRAME_PERIOD <= 806;
                end
        4'b1000 : begin
                //1280*720 行时序参数
                H_SYNC_PULSE <= 40;
                H_BACK_PORCH <= 220;
                H_ACTIVE_TIME <= 1280;
                H_FRONT_PORCH <= 110;
                H_LINE_PERIOD <= 1650;
                //1280*720 场时序参数
                V_SYNC_PULSE <= 5;
                V_BACK_PORCH <= 20;
                V_ACTIVE_TIME <= 720;
                V_FRONT_PORCH <= 5;
                V_FRAME_PERIOD <= 750;
                end
         default : begin
                //640*480
                H_SYNC_PULSE <= 96;
                H_BACK_PORCH <= 48;
                H_ACTIVE_TIME <= 640;
                H_FRONT_PORCH <= 16;
                H_LINE_PERIOD <= 800;
                //640*480
                V_SYNC_PULSE <= 2;
                V_BACK_PORCH <= 33;
                V_ACTIVE_TIME <= 480;
                V_FRONT_PORCH <= 10;
                V_FRAME_PERIOD <= 525;
                end
    endcase
end

///产生行时序
always @(posedge clk_out or negedge rst) begin
    if(!rst)
        h_cnt <= 12'd0;
    else if(h_cnt == H_LINE_PERIOD - 1'b1)
        h_cnt <= 12'd0;
    else h_cnt <= h_cnt + 1'b1;
end
```

```
assign hs = (h_cnt<H_SYNC_PULSE)? 1'b0:1'b1;

///产生场时序
always @(posedge clk_out or negedge rst) begin
    if(!rst)
        v_cnt <= 12'd0;
    else if(v_cnt == V_FRAME_PERIOD - 1'b1)
        v_cnt <= 12'd0;
    else if(h_cnt == H_LINE_PERIOD - 1'b1)
        v_cnt <= v_cnt + 1'b1;
    else v_cnt <= v_cnt;
end
assign vs = (v_cnt<V_SYNC_PULSE)?1'b0:1'b1;

assign active_flag = (h_cnt>=(H_SYNC_PULSE+H_BACK_PORCH))&&
                     (h_cnt<=
(H_SYNC_PULSE+H_BACK_PORCH+H_ACTIVE_TIME))&&
                     (v_cnt>=(V_SYNC_PULSE+V_BACK_PORCH))&&
                     (v_cnt<=
(V_SYNC_PULSE+V_BACK_PORCH+V_ACTIVE_TIME));

///输出图片数据
always @(posedge clk_out or negedge rst) begin
    if(!rst) begin
            address <= 20'd0;
        end
    else if(active_flag)
        if(stretch) begin
               if(h_cnt>=(H_SYNC_PULSE+H_BACK_PORCH+left_pos)&&
                  h_cnt<=
(H_SYNC_PULSE+H_BACK_PORCH+left_pos+H_ACTIVE_TIME-1)&&
                  v_cnt>=(V_SYNC_PULSE+V_BACK_PORCH+up_pos)&&
                  v_cnt<=
(V_SYNC_PULSE+V_BACK_PORCH+up_pos+V_ACTIVE_TIME-1)) begin
                  red <= data[11:8];
                  green <= data[7:4];
                  blue <= data[3:0];
                  if(address >= PIXEL-1) address <= 0;
                  else address <= HEIGHT *(v_cnt-(V_SYNC_PULSE +
V_BACK_PORCH)) / V_ACTIVE_TIME * WIDTH + HEIGHT * (h_cnt -
(H_SYNC_PULSE + H_BACK_PORCH)+1)/H_ACTIVE_TIME;
                   end
               else begin
                    red <= 4'b0000;
                    green <= 4'b0000;
                    blue <= 4'b0000;
                    address <= address;
               end
            end
        else
           if(h_cnt>=(H_SYNC_PULSE+H_BACK_PORCH+left_pos)&&
              h_cnt<=(H_SYNC_PULSE+H_BACK_PORCH+left_pos+WIDTH-1)&&
              v_cnt>=(V_SYNC_PULSE+V_BACK_PORCH+up_pos)&&
              v_cnt<=(V_SYNC_PULSE+V_BACK_PORCH+up_pos+HEIGHT-1)) begin
                  red <= data[11:8];
                  green <= data[7:4];
                  blue <= data[3:0];
           if(address == PIXEL-1) address<= 0;
```

```
            else address <= address +1;
            end
            else begin
                   red <= 4'b0000;
                   green <= 4'b0000;
                   blue <= 4'b0000;
                   address <= address;
            end
            else begin
                  red <= 4'b0000;
                  green <= 4'b0000;
                  blue <= 4'b0000;
                  address <= address;
             end
end

//获得场扫描的下降沿
always @(posedge clk_out or negedge rst) begin
    if(!rst) begin
            vs_temp1 <= 0;
            vs_temp2 <= 0;
        end
    else begin
            vs_temp2 <= vs_temp1;
            vs_temp1 <= vs;
        end
end
assign neg_vs = (~vs_temp1)&vs_temp2;

//图片移动以及屏保
always@(posedge clk_out or negedge rst) begin
    if(!rst) begin
            left_pos <= 0;
            up_pos <= 0;
            right <= 1;
            down <= 1;
        end
    else if(neg_vs) begin
            if(screensaver) begin
                    if(left_pos==H_ACTIVE_TIME-WIDTH-10) right <= 0;
                    if(left_pos==1) right <= 1;
                    if(up_pos==V_ACTIVE_TIME-HEIGHT-10) down <= 0;
                    if(up_pos==1) down <= 1;
                    if(right) left_pos <= left_pos+1;
                    else left_pos <= left_pos-1;
                    if(down) up_pos <= up_pos+1;
                    else up_pos <= up_pos-1;
                end
            else begin
            if(sw_left) begin
                    if (left_pos>=1) left_pos <= left_pos-1;
                    else left_pos <= left_pos;
                    up_pos <= up_pos;
                end
            else if(sw_right) begin
                    if(left_pos<=H_ACTIVE_TIME-WIDTH-1) left_pos <=
left_pos+1;
                    else left_pos <= left_pos;
```

```
                    up_pos <= up_pos;
                end
            else if(sw_up) begin
                    left_pos <= left_pos;
                    if(up_pos>=1) up_pos <= up_pos-1;
                    else up_pos <= up_pos;
                end
            else if(sw_down) begin
                    left_pos <= left_pos;
                    if(up_pos<=V_ACTIVE_TIME-HEIGHT-1) up_pos <=
up_pos+1;
                    else up_pos <= up_pos;
                end
            else if(return) begin
                    left_pos <= (H_ACTIVE_TIME-WIDTH)/2;
                    up_pos <= (V_ACTIVE_TIME-HEIGHT)/2;
                end
            end
        end
end

clk_wiz_0 clkset(clk_25M, clk_40M, clk_65M, clk_75M, rst, clk);

assign clk_out = (radio==4'b0010) ? (clk_40M) : (radio==4'b0100) ?
clk_65M : (radio==4'b1000) ? clk_75M : clk_25M;

ROM uimage(clk_out,address,data);

endmodule
```

Uart_rx module

```
`timescale 1ns / 1ps
module Uart_rx(
input Clk,                          //系统时钟
input rst,                          //复位信号
input [20:0]UART_BaudTime,          //波特系数//BaudTime = 频率/波特mm
率-1;
input UART_RX,                      //Uart端口输入
input [3:0] radio,
output reg [20:0]address_ram,       //在RAM中的地址
output reg [15:0]data,              //16bit的颜色数据RGB0
output reg dataflag                 //判断是否读完数据
);
reg [1:0] rx_data;                  //消除亚稳态，检测下降沿
reg detect_negedge;                 //检测下降沿信号
reg start;                          //起始信号
reg sample;                         //采样信号
reg [4:0] bit_cnt=0;                //bit位计数器，每10个bit为一个周期（起始信
号、8bit读入、终止信号）
reg [20:0] time_cnt=0;              //时间计数器，记到波特系数/2时，触发采样信
号
reg [19:0] buffer_data;             //数据缓冲区，将串行转为并行
reg isfront=1;                      //标记读入的数据为前八位还是后八位
//消除亚稳态
reg[17:0]max_ind;
always @(posedge Clk) begin
```

```
    case(radio)
        4'b0010:max_ind<=103680;
        4'b0100:max_ind<=40000;
        4'b1000:max_ind<=2500;
        default:max_ind<=147456;
    endcase
end
always @(posedge Clk or negedge rst) begin
    if(!rst) rx_data <= 2'b11;
    else rx_data <= {rx_data[0],UART_RX};
end
//检测下降沿
always @(posedge Clk or negedge rst) begin
    if(!rst) detect_negedge <= 1'b0;
    else if(!start)
        detect_negedge <= rx_data[1]&&(~rx_data[0]);
end
//标记起始信号
always @(posedge Clk or negedge rst) begin
    if(!rst) start <= 1'b0;
    else if(!start&&detect_negedge) start <= 1;
    else if(bit_cnt==10) start<= 0;
end
//时间流计数器
always @(posedge Clk or negedge rst) begin
    if(!rst) time_cnt <= 21'd0;
    else if(time_cnt == UART_BaudTime || !start)
        time_cnt <=  21'd0;
    else time_cnt <= time_cnt+1;
end
//计数到波特系数一半时，最为稳定，此时标记采样信号
always @(posedge Clk or negedge rst) begin
    if(!rst) sample <= 0;
    else if(time_cnt==(UART_BaudTime>>1))
        sample <= 1'b1;
    else sample <= 1'b0;
end
//bit位计数器，当采样到10位时，标记为0
always @(posedge Clk or negedge rst) begin
    if(!rst) bit_cnt <= 0;
    else if(!sample && bit_cnt!=10)
        bit_cnt <= bit_cnt;
    else if(sample) bit_cnt <= bit_cnt + 1;
    else bit_cnt <= 0;
end
//每采样一个数据，向右移位。将串行转并行
always @(posedge Clk or negedge rst) begin
    if(!rst) buffer_data <= 0;
    else if(sample) buffer_data <= {rx_data[0],buffer_data[19:1]};
    else buffer_data <= buffer_data;
end
//每次bit计数器到10时，isfront取反表明下一个状态读的是前/后八位
always @(posedge Clk or negedge rst) begin
    if(!rst) isfront = 1;
    else if(bit_cnt==10) isfront = ~isfront;
end
//当后8位读完时，表明一组RGB已经读入。此时可将移位寄存器的并行数据赋值给RGD信号
always @(posedge Clk or negedge rst) begin
```

```
    if(!rst) data <= 0;
    else if(bit_cnt==10&&!isfront) data <=
{buffer_data[18:11],buffer_data[8:1]};
    else data <= data;
end
//一组RGB信号读完后，标记dataflag
always @(posedge Clk or negedge rst) begin
    if(!rst) dataflag <= 1'b0;
    else if(bit_cnt ==10&&!isfront) dataflag <= 1'b1;
    else dataflag <= 0;
end
//读完一组数据，同步RAM中的地址
always @(posedge Clk or negedge rst) begin
    if(!rst)
        address_ram <= 21'd0;
    else if(address_ram ==max_ind -1)
        address_ram <= 0;
    else if(dataflag)
        address_ram <= address_ram +1;
    else address_ram <= address_ram;
end
endmodule
```

- **Constraint File**

```
set_property IOSTANDARD LVCMOS33 [get_ports CLK_100M]
set_property IOSTANDARD LVCMOS33 [get_ports {Data[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Data[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Data[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Data[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Data[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Data[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Data[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Data[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Data[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Data[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Data[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Data[0]}]
set_property PACKAGE_PIN Y18 [get_ports CLK_100M]
set_property IOSTANDARD LVCMOS33 [get_ports hs]
set_property IOSTANDARD LVCMOS33 [get_ports RST1]
set_property IOSTANDARD LVCMOS33 [get_ports UART_RX]
set_property IOSTANDARD LVCMOS33 [get_ports vs]
set_property PACKAGE_PIN Y19 [get_ports UART_RX]
set_property PACKAGE_PIN Y9 [get_ports RST1]
set_property PACKAGE_PIN M21 [get_ports hs]
set_property PACKAGE_PIN L21 [get_ports vs]
set_property PACKAGE_PIN H17 [get_ports {Data[0]}]
set_property PACKAGE_PIN H18 [get_ports {Data[1]}]
set_property PACKAGE_PIN J22 [get_ports {Data[2]}]
set_property PACKAGE_PIN H22 [get_ports {Data[3]}]
set_property PACKAGE_PIN G17 [get_ports {Data[4]}]
set_property PACKAGE_PIN G18 [get_ports {Data[5]}]
set_property PACKAGE_PIN J15 [get_ports {Data[6]}]
set_property PACKAGE_PIN H15 [get_ports {Data[7]}]
set_property PACKAGE_PIN H20 [get_ports {Data[8]}]
set_property PACKAGE_PIN G20 [get_ports {Data[9]}]
```

```
set_property PACKAGE_PIN K21 [get_ports {Data[10]}]
set_property PACKAGE_PIN K22 [get_ports {Data[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[19]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[18]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[17]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[16]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[15]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[14]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[13]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[12]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {buffer_data[0]}]
set_property PACKAGE_PIN K17 [get_ports {buffer_data[0]}]
set_property PACKAGE_PIN L13 [get_ports {buffer_data[1]}]
set_property PACKAGE_PIN M13 [get_ports {buffer_data[2]}]
set_property PACKAGE_PIN K14 [get_ports {buffer_data[3]}]
set_property PACKAGE_PIN K13 [get_ports {buffer_data[4]}]
set_property PACKAGE_PIN M20 [get_ports {buffer_data[5]}]
set_property PACKAGE_PIN N20 [get_ports {buffer_data[6]}]
set_property PACKAGE_PIN N19 [get_ports {buffer_data[7]}]
set_property PACKAGE_PIN M17 [get_ports {buffer_data[8]}]
set_property PACKAGE_PIN M16 [get_ports {buffer_data[9]}]
set_property PACKAGE_PIN M15 [get_ports {buffer_data[10]}]
set_property PACKAGE_PIN K16 [get_ports {buffer_data[11]}]
set_property PACKAGE_PIN L16 [get_ports {buffer_data[12]}]
set_property PACKAGE_PIN L15 [get_ports {buffer_data[13]}]
set_property PACKAGE_PIN L14 [get_ports {buffer_data[14]}]
set_property PACKAGE_PIN J17 [get_ports {buffer_data[15]}]
set_property PACKAGE_PIN F21 [get_ports {buffer_data[16]}]
set_property PACKAGE_PIN G22 [get_ports {buffer_data[17]}]
set_property PACKAGE_PIN G21 [get_ports {buffer_data[18]}]
set_property PACKAGE_PIN D21 [get_ports {buffer_data[19]}]
set_property IOSTANDARD LVCMOS33 [get_ports sw_up]
set_property IOSTANDARD LVCMOS33 [get_ports sw_right]
set_property IOSTANDARD LVCMOS33 [get_ports sw_left]
set_property IOSTANDARD LVCMOS33 [get_ports sw_down]
set_property IOSTANDARD LVCMOS33 [get_ports screensaver]
set_property IOSTANDARD LVCMOS33 [get_ports RST3]
set_property IOSTANDARD LVCMOS33 [get_ports RST2]
set_property IOSTANDARD LVCMOS33 [get_ports return]
set_property PACKAGE_PIN W9 [get_ports RST2]
set_property PACKAGE_PIN Y7 [get_ports RST3]
set_property PACKAGE_PIN Y8 [get_ports screensaver]
set_property PACKAGE_PIN P2 [get_ports sw_down]
set_property PACKAGE_PIN P1 [get_ports sw_left]
set_property PACKAGE_PIN R1 [get_ports sw_right]
set_property PACKAGE_PIN P5 [get_ports sw_up]
set_property PACKAGE_PIN P4 [get_ports return]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {tube_open[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_open[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_open[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_open[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_open[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_open[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_open[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_open[0]}]
set_property PACKAGE_PIN A18 [get_ports {tube_open[7]}]
set_property PACKAGE_PIN A20 [get_ports {tube_open[6]}]
set_property PACKAGE_PIN B20 [get_ports {tube_open[5]}]
set_property PACKAGE_PIN E18 [get_ports {tube_open[4]}]
set_property PACKAGE_PIN F18 [get_ports {tube_open[3]}]
set_property PACKAGE_PIN D19 [get_ports {tube_open[2]}]
set_property PACKAGE_PIN E19 [get_ports {tube_open[1]}]
set_property PACKAGE_PIN C19 [get_ports {tube_open[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_number[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_number[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_number[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_number[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_number[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_number[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_number[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tube_number[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {stretch}]
set_property PACKAGE_PIN W4 [get_ports {stretch}]
set_property PACKAGE_PIN E13 [get_ports {tube_number[7]}]
set_property PACKAGE_PIN C15 [get_ports {tube_number[6]}]
set_property PACKAGE_PIN C14 [get_ports {tube_number[5]}]
set_property PACKAGE_PIN E17 [get_ports {tube_number[4]}]
set_property PACKAGE_PIN F16 [get_ports {tube_number[3]}]
set_property PACKAGE_PIN F14 [get_ports {tube_number[2]}]
set_property PACKAGE_PIN F13 [get_ports {tube_number[1]}]
set_property PACKAGE_PIN F15 [get_ports {tube_number[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {radio[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {radio[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {radio[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {radio[3]}]
set_property PACKAGE_PIN AB8 [get_ports {radio[0]}]
set_property PACKAGE_PIN AA8 [get_ports {radio[1]}]
set_property PACKAGE_PIN V8 [get_ports {radio[2]}]
set_property PACKAGE_PIN V9 [get_ports {radio[3]}]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets RST2_IBUF]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets RST3_IBUF]
```

Description of the constraint file

Input

- CLK_100M is connected to the Y18 pin, and input the clock frequency of 100MHz to the system.
- UART_RX, which is connected to the Y19 pin and input UART information to the system.
- RST1, 2, 3. The input is connected with the Y9, W9, and Y7 pins to provide the system with reset signals corresponding to the system reset of

task1, 2, and 3, respectively.

- Radio[3:0] this input is connected to the AB8,AA8,V8, and V9 pins, providing different resolutions for the system.
- Return, sw_left, sw_right, sw_up, sw_down are connected with P4,P1,R1,P5 and P2 pins to provide input information for the image movement function of the system.
- The Screensaver is connected to the Y8 pin to provide the system with the switch information for the Screensaver function.

Output

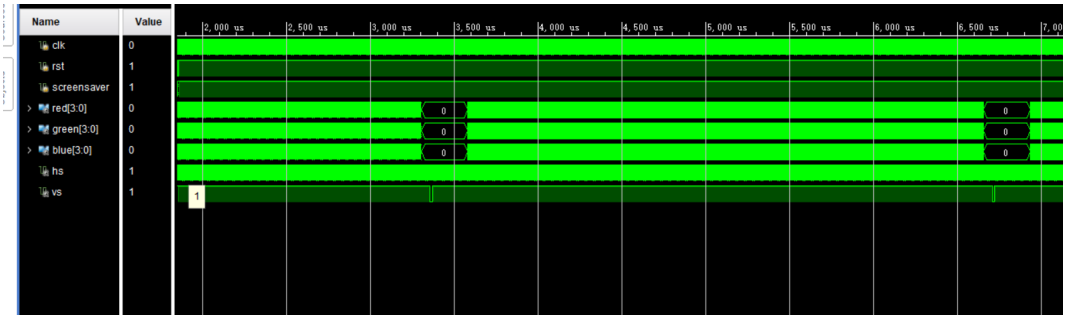
- Hs, vs are connected with M21 and L21 pins to provide the system with the information of the current pixel position when output.
- Data[11:0] is connected to the VGA module RGB pipe foot of the development board to provide the system with the color information of the current pixel when output.
- Tube_open[7:0] and tube_number[7:0] are connected to the 7-segment digital tube foot of the development board to provide the digital output of the digital tube for the system.

Part 3-Testing

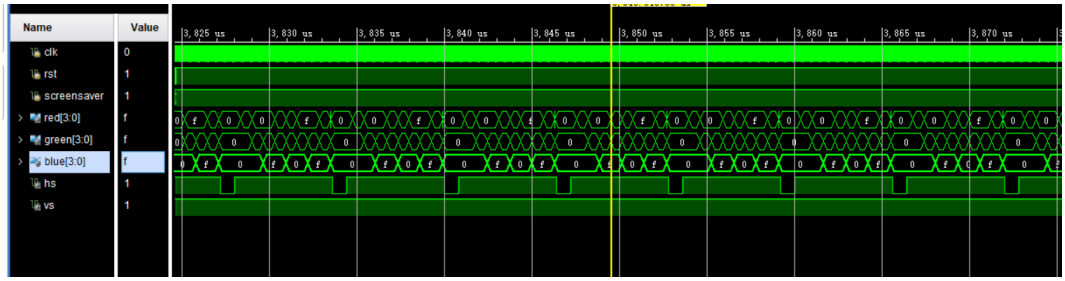
- **Task1 module**
  - *Testbench file*

```
`timescale 1ns / 1ps
module task1_tb(

    );
reg clk,rst,screensaver;
wire [3:0] red;
wire [3:0] green;
wire [3:0] blue;
wire hs;
wire vs;

Task1 u(clk,rst,red,green,blue,hs,vs,screensaver);

initial begin
   clk <= 0;
   rst <= 0;
   screensaver <= 0;
   #5 rst <= 1;
   #5 screensaver <= 1;
end

always #1 clk <= ~clk;

endmodule
```

- *Overall simulation result*



- *partial enlarged view*

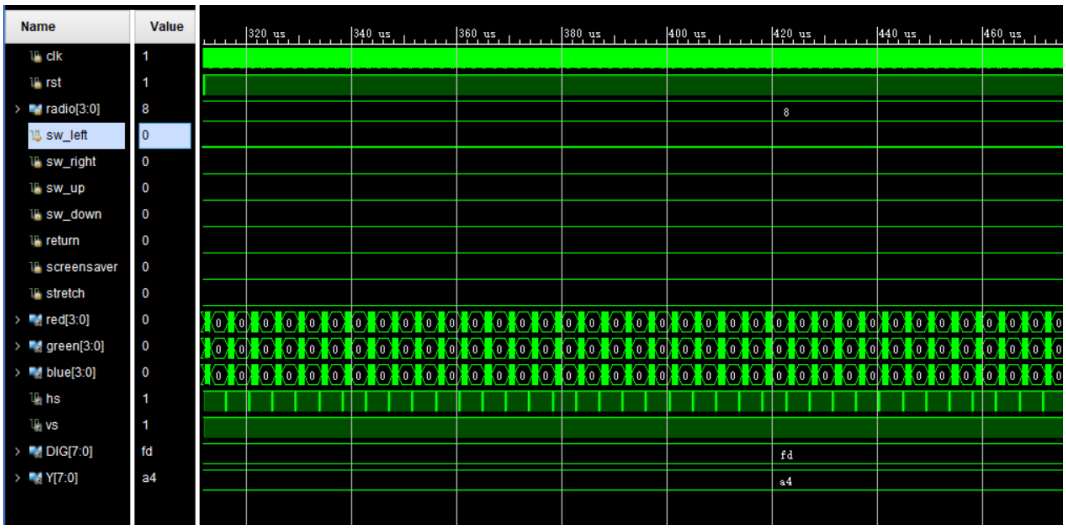


- **Task2 module**
  - *Testbench file*

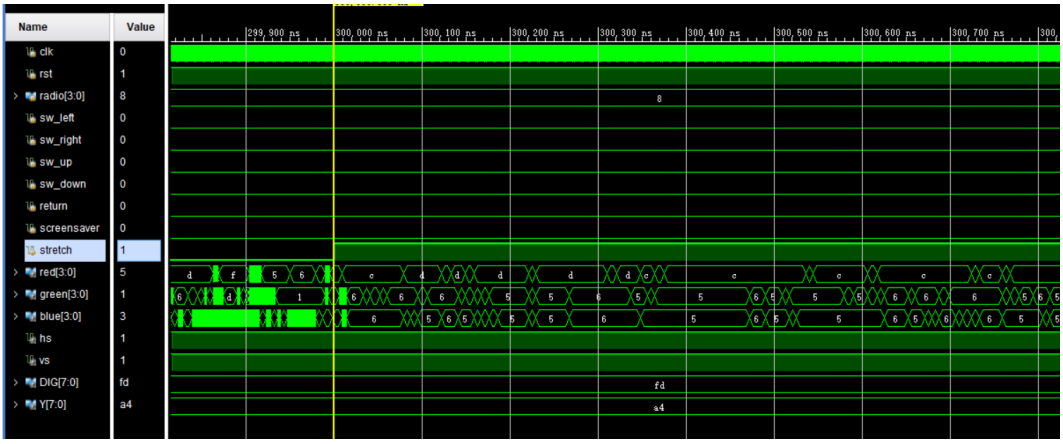
```
`timescale 1ns / 1ps

module task2_tb(

    );

  reg clk;
  reg rst;
  reg [3:0] radio;
  reg sw_left;
  reg sw_right;
  reg sw_up;
  reg sw_down;
  reg return;
  reg screensaver;
  reg stretch;
  wire [3:0] red;
  wire [3:0] green;
  wire [3:0] blue;
  wire hs;
  wire vs;
  wire [7:0] DIG;
  wire[7:0] Y;

  Task2_4
 u(clk,rst,radio,sw_left,sw_right,sw_up,sw_down,return,screensaver,stre
tch,red,green,blue,hs,vs,DIG,Y);

  initial
      begin
          clk <= 0;
          rst <= 0;
          radio <= 4'b1000;
```

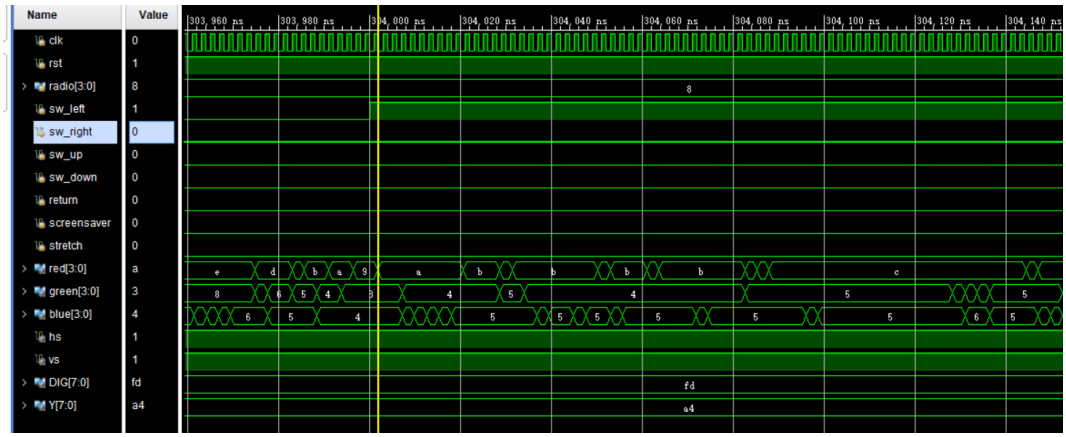
```
            sw_left <= 0;
            sw_right <= 0;
            sw_up <= 0;
            sw_down <= 0;
            return <= 0;
            screensaver <= 0;
            stretch <= 0;
            #5 rst <=1;
        end

    initial
      begin
          #300000
          stretch <= 1;
          #1000
          stretch <= 0;
      end

    initial
      begin
          #302000
          screensaver <= 1;
          #1000
          screensaver <= 0;
      end

    initial
        begin
            #304000
            sw_left <= 1;
            #1000
            sw_left <= 0;
        end


    always #1 clk <= ~clk;

    endmodule
```

- *VerilogOverall simulation result*



- *partial enlarged view*

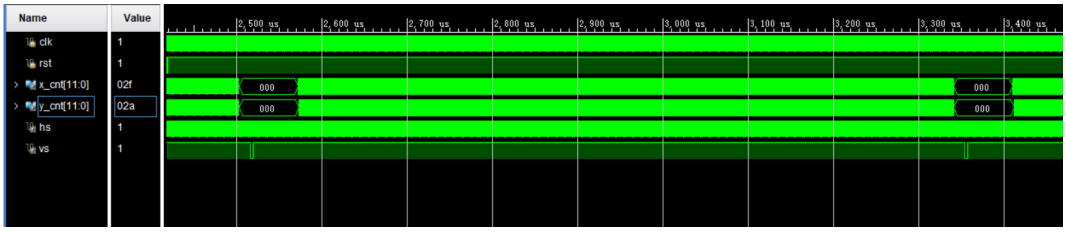




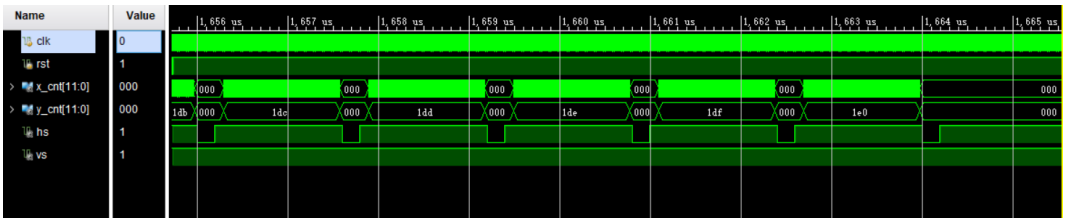
- **VGA_paint module**
  - *Testbench file*

```
`timescale 1ns / 1ps
module VGA_paint_tb(

    );

reg clk,rst;
wire [11:0]x_cnt;
wire [11:0]y_cnt;
wire hs,vs;

VGA_Paint u(clk,rst,x_cnt,y_cnt,hs,vs);

initial begin
      clk <= 0;
      rst <= 0;
      #5 rst <=1;
end

always #1 clk <= ~clk;

endmodule
```

  - *Overall simulation result*



- *partial enlarged view*

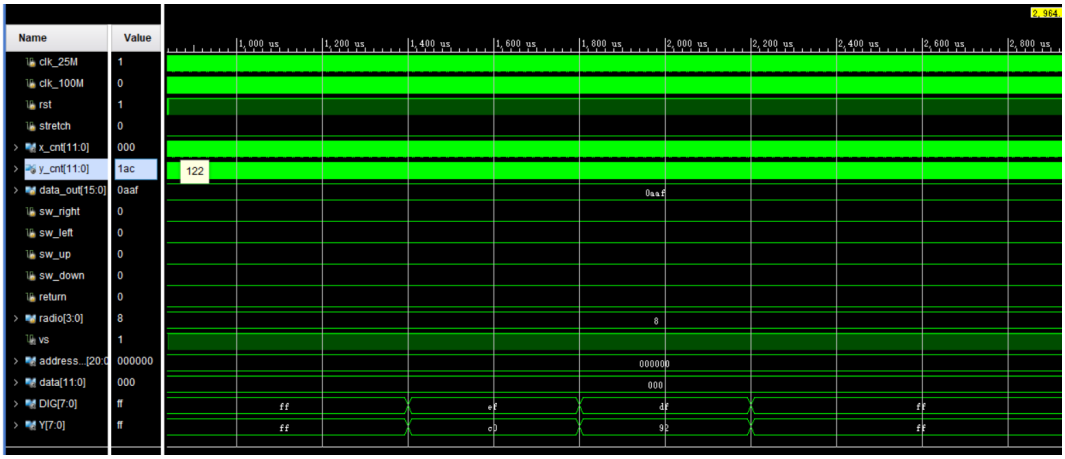


- **VGA_Display**
  - *Testbench file*

```
`timescale 1ns / 1ps
module VGA_display_tb(

    );
reg CLK_25M,CLK_100M,rst,stretch;
wire [11:0]x_cnt;
wire [11:0]y_cnt;
reg [15:0]data_out;
reg sw_right,sw_left,sw_up,sw_down,return;
reg [3:0] radio;
wire vs;
wire [20:0]address_pic;
wire [11:0]data;
wire [7:0] DIG;
wire [7:0] Y;

VGA_Paint
u_paint(.Clk(CLK_25M),.rst(rst),.x_cnt(x_cnt),.y_cnt(y_cnt),.vs(vs));
VGA_Display u_display(
CLK_100M,CLK_25M,rst,x_cnt,y_cnt,data_out,sw_left,sw_right,sw_up,sw_dow
n,return,vs,radio,stretch,address_pic,data,DIG,Y);

initial begin
      CLK_100M <=0;
      CLK_25M <= 0;
      rst <= 0;
      sw_left <= 0;
      sw_right <= 0;
      sw_up <= 0;
      sw_down <= 0;
      return <= 0;
      stretch <= 0;
      radio <= 4'b1000;
end

always #1 CLK_100M <= ~CLK_100M;
always #4 CLK_25M <= ~CLK_25M;
```

```
endmodule
```

- *Overall simulation result*



- *partial enlarged view*

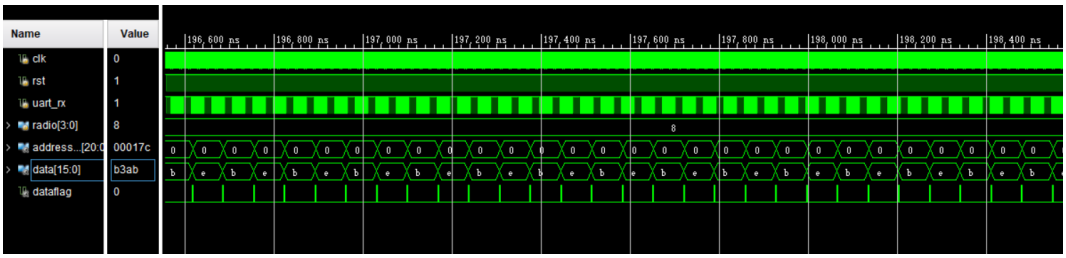
Uart_rx module

- *Testbench file*

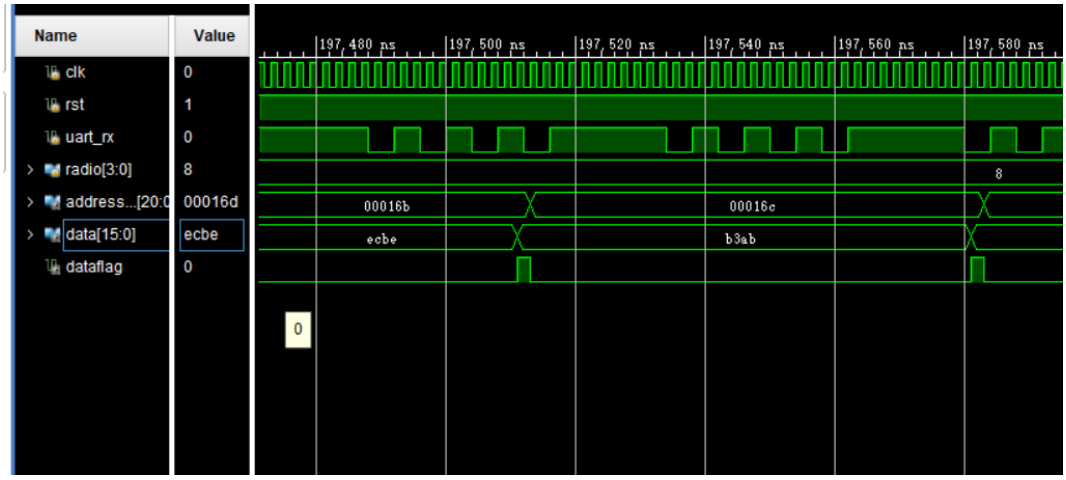
```
`timescale 1ns / 1ps
module Uart_rx_tb(

    );

reg clk,rst,uart_rx;
reg [3:0] radio;
wire [20:0]address_ram;
wire [15:0]data;
```

```
wire dataflag;

Uart_rx u(clk,rst,16'd1,uart_rx,radio,address_ram,data,dataflag);

initial begin
      clk <= 0;
      rst <= 0;
      radio <= 4'b1000;
      #5 rst <=1;
end

always #1 clk <= ~clk;

always begin
    uart_rx <= 1;
    #10
    uart_rx <= 0; //start
    #4
    uart_rx <= 1;
    #4
    uart_rx <= 0;
    #4
    uart_rx <= 1;
    #4
    uart_rx <= 0;
    #4
    uart_rx <= 1;
    #4
    uart_rx <= 0;
    #4
    uart_rx <= 1;
    #4
    uart_rx <= 1;
    #4
    uart_rx <= 1;//end
end

endmodule
```

- *Overall simulation result*



- *partial enlarged view*



Part 4-Summary

During the development process, our group encountered many problems. The solution of each problem took great efforts of our team members.

1. Summary of problems and solutions in the development process:

- *Initial understanding of the task*
  At the beginning, the first problem our team encountered was understanding the task and how to implement it. In the whole semester's study, we had not touched the input and output except the dial-code switch, indicator light and 7-segment digital tube. However, in the project of VGA video card, we were required to directly transfer data from the computer to the development board by using the UART port, and output images to the display by using the VGA port. All of these knowledge is new to us. Therefore, we are not very clear about the difficulty of each task and how to master VGA, and we have taken many detours during this period, such as unreasonable task division and the specific functions to be realized by each task.
- *Solution:*
  First of all, the requirements above for a description of the task is not very clear, therefore, we in the task description not clear direction question, the teacher and the teacher also answer the patience, at the same time sent a more detailed task description, it is also the task description, let us clear the specific content of each task, but also provide suggestion to adjust direction based on the description, found out the right direction.
- *First draft of task*
  After the initial period of confusion, we found the right direction and set about our task. At this point, however, there is a second problem: implementation difficulties. Before this, we had not touched the UART protocol, which input information directly from the computer to the

development board, so we did not know much about how to send information from the computer to the development board, and how to receive and store the information sent by the computer. On top of that, we also ran into code incompatibilities. Because of the issues involved in accessing coe files and the differences in naming and thinking habits of each person, the interfaces between different tasks do not perform as expected.

- *Solution:*
  Because using development board VGA graphics are popular and basic problems, so on the network also has very many people to share their experience, so we fully studying relevant tutorial on the network, through understanding the specific content of UART protocol, and research of the existing port debugging assistant, after fully understand personally developed port debugging assistant, solves the computer the question. At the same time, by studying the parallel conversion of strings and learning how to use the IP that calls RAM, we realized the function of storing information in RAM after the development board receives it and then calling RAM to store information. Finally, through timely communication within the team, we determined a set of common port naming rules and made clear and detailed comments, so that the docking work between different modules could finally proceed smoothly.
- *Final task draft*
  At this point, we have a full understanding of the task, and can cooperate with each other in a tacit way. The whole project is also gradually promoted to integration, improvement and finishing work. At this point, we carried out a detailed test of the implemented functions, and also discussed the direction of our future improvement. At this point, we found in the process of testing that when entering different information, the program does not achieve good compatibility, but will crash. Also, when we were writing the front-end integration project, we ran into problems with the output not meeting expectations.
- *Solution:*
  At this time, we find our thinking mistakes by communicating with other groups and asking them how to solve relevant problems. At the same time, I also found many people who encountered similar problems on the Internet and read their solutions to timely change our project. In the end, the perfect implementation of our project solved the compatibility problem and the output did not match expectations.

2. Describe the characteristics and optimization direction of the current system:

- *Current system features:*
  Our system has many features. First of all, the system for all independent research and development. From the implementation of the bottom Verilog code to the control system of the top front end, as well as the conversion of videos to pictures frame by frame, and the conversion of pictures to coe files were all manually implemented by our team members. Secondly, the system has realized the rich expansion function. For example, the system can input video information to play the video frame by frame based on the input picture. The system also realized the screen saver function, after the serial port input picture, can make the picture on the display according to the requirements of regular movement, in touch with the screen edge can rebound. Finally, the system realizes the front-end function. After integrating all the tasks and our extended functionality, we wrote the front-end controller in Java. The front-end program can directly input commands to the development board through the UI button, which can facilitate the issuing of commands to the development board and system testing.
- *Optimization direction:*
  At present, the system can be optimized and improved to make it more perfect. First of all, the video playback function of this system is not perfect, and it cannot achieve a high refresh rate, resulting in our video is not particularly smooth when playing. Here, the system can be improved. Secondly, the interactive function of the system needs to be improved. For example, games can be provided to control the input through the computer or development board, and the current effect can be instantly output on the display, bringing a better interactive experience for users. Finally, the system can be optimized in terms of sound. The current output of video information does not contain sound information, therefore, can consider through the buzzer to achieve the function of playing sound.