同济大学计算机系

数字逻辑课程实验报告

学　　号 2051493

姓　　名 张勋

专　　业 计算机科学与技术

授课老师 郭玉臣

目录

一、 实验内容

1. 项目内容介绍

   基于 VS1003B-MP3 模块、VGA 显示器和蓝牙模块，使用 verilog 语言，实现可以控制音量、切换歌曲、显示专辑封面、支持蓝牙操控的 mp3 播放器。

2. 项目使用设备介绍

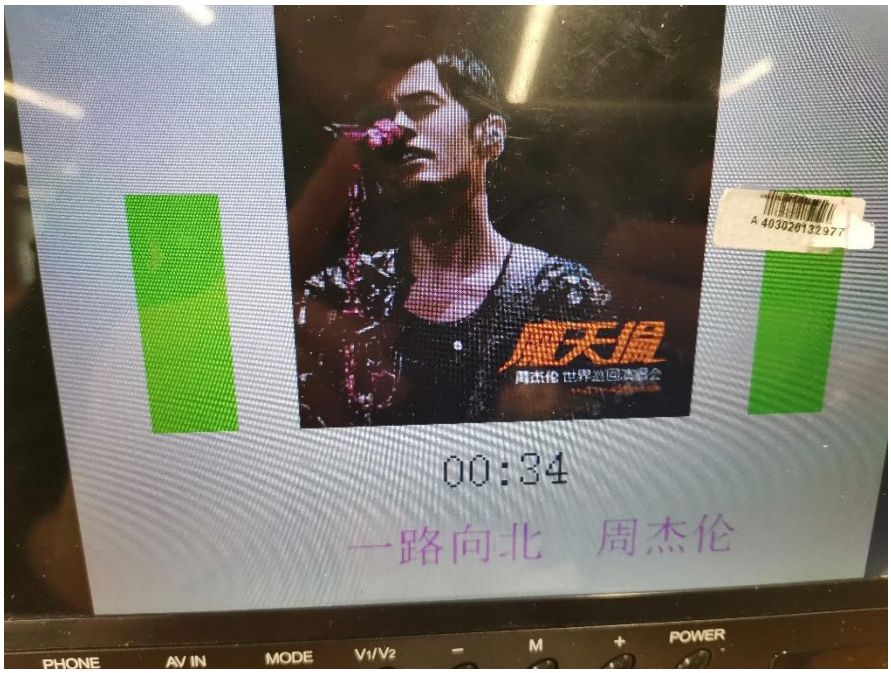
   1) NEXYS 4 DDR Atrix-7 开发板：由 Xilinx 公司开发出的一款现场可编程门阵列(FPGA)开发板

   2) 显示器：使用 VGA 接口，分辨率为 1024*768,刷新率为 60Hz，对应的时钟频率为 65Mhz

   3) VS1003B-MP3 模块：支持 WMA4.0/4.1/7/8/9 所有流文件、MP3 和 WAV 流

   4) 蓝牙模块：能够当做主机和从机，使用 uart 串口协议传输信息

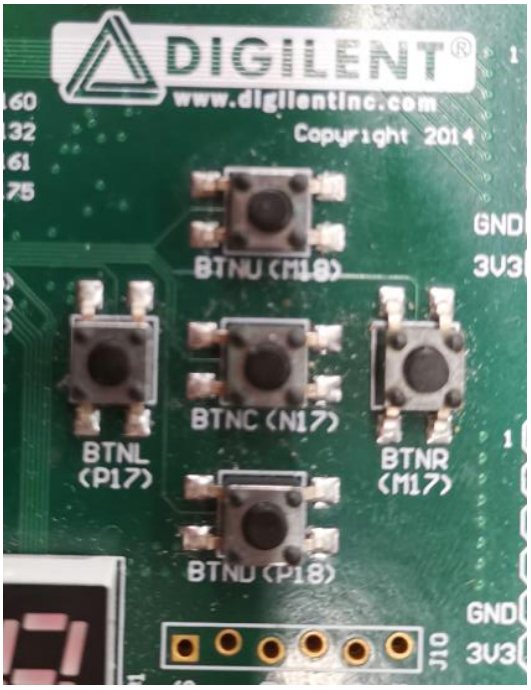
3. 界面说明



   中间区域为当前歌曲的图片，左右两个长条形显示当前的音量，图片下方为当前歌曲播放时间以及歌曲的信息

4. 操作说明

   支持蓝牙和开发板端进行操作

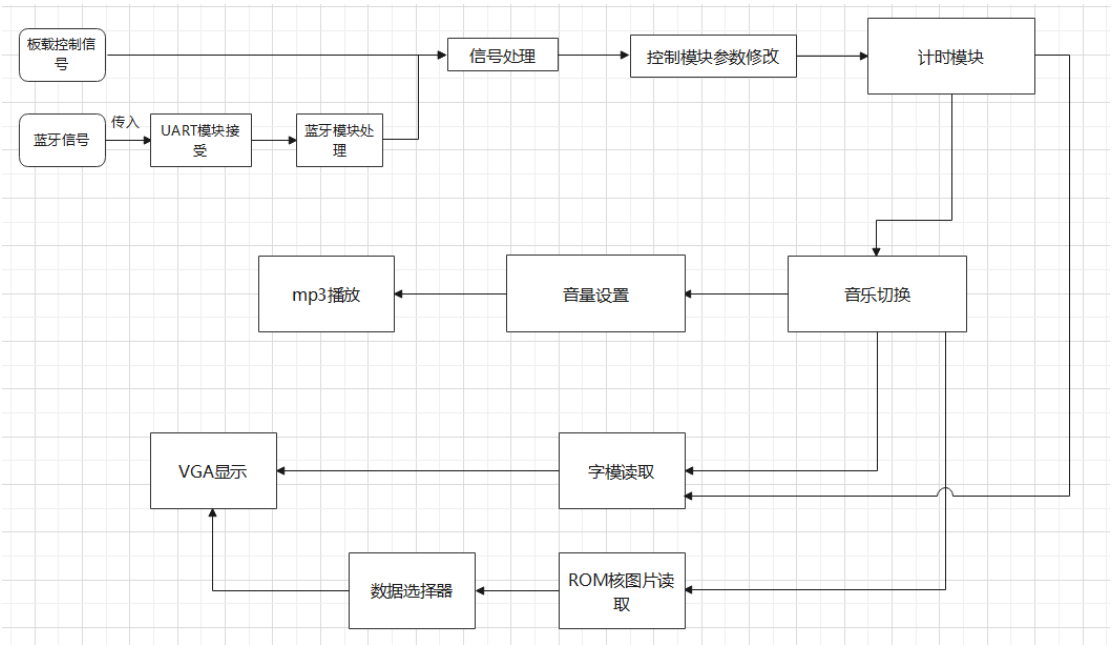


手机端连接蓝牙后，即可通过按钮进行音量的加减和音乐的切换，点击按钮后，开发板的 V11,V12,V14,V15 端口对应的灯会闪烁

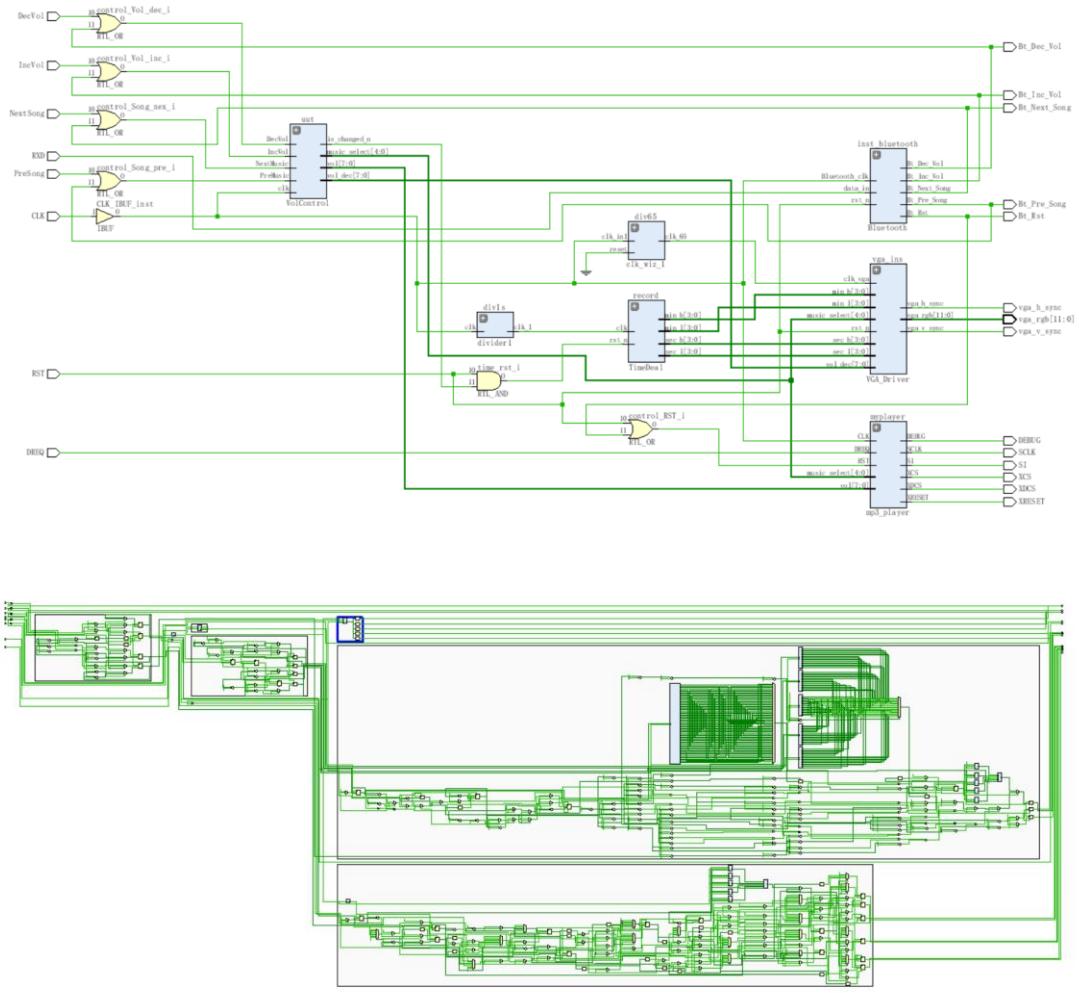
开发板端可通过 P17,M17 进行音量的加减，P18,M18 可进行音乐的切换，J15 为复位按键。

二、 项目总视图

设计框图如下



RTL 分析



项目整体按照先自顶向下再自底向上的方法进行设计，先思考整个项目需要完成什么，输入输出端大概有哪些，之后再进行模块化设计，最后将模块拼接优化。

各个模块的说明

1).分频模块(div):使用 vivado 自带的 IP 核，将 100MHZ 的输入时钟分频，提供给后序模块使用

2).数据 IP 核：使用 vivado 自带 IP 核，将音频数据和图片数据存入到 ROM 中，由于 VGA 输出图片相当于位图模式显示，占据较大的空间，此处选择的图片均为 240*240 大小，VGA 中每 4 个数据读取图片中的一个数据，相当于实现 4 倍的放大。

3).字模数据模块(char_set)：提供显示歌曲信息的字模，由于想要实现较高精度的显示，所以所以字符均采用 64 字宽*64 字高。

4).音乐播放模块(mp3_player)：mp3 模块数据输送符合 SPI 协议，MP3 模块中的针脚输入 DREQ 来判断是否能够继续获取数据，而 XCDS、XCS 等端口通过 MP3 模块的针脚输入

到 MP3 中，协调指令的传输和数据的传输。

5).显示器模块(VGA_Driver)：接受行信号、场信号和 RGB 值，按照 1024*768 的分辨率在显示器中显示图像，行信号、场信号主要用来协调显示器进行刷新、同步等操作。本模块实现了歌曲专辑图片的显示，音频调节的图形化显示以及歌曲灯的移动显示特效。

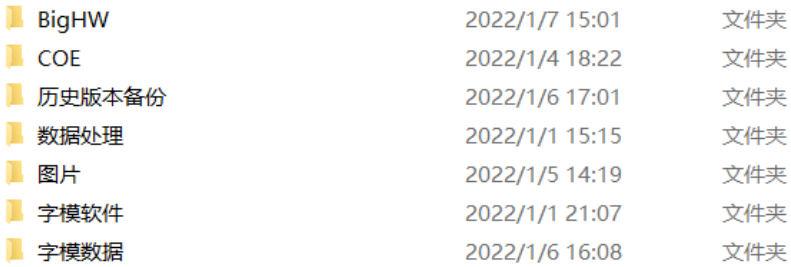
6).蓝牙模块(Bluetooth 和 UART)：通过实现 UART 协议，从手机端获取控制信号，获取的蓝牙和板载按键共同进行播放器的操作

7).音乐控制模块(MusicControl)：实现音量的调整和当前播放音乐的调整，输出端为 8 位音量值和 5 位信息值，音量值在播放模块作为 sdi 信息的一部分传入 mp3,5 位信息值(5 位中只含一位 1)作为 ip 核的使能端。

三、 版本管理

本项目在本地进行版本管理

最终项目文件夹包含以下内容



四、 子系统模块建模

1. 顶层模块 TOP

1） 描述

定义输入输出端口，调用、连接各个模块，同时对一些控制信号作基本的逻辑运算

2） 接口信号定义

接口名称	接口属性	接口描述
CLK	input	开发板系统时钟 100MHz
RST	input	复位信号与开始信号,低电平有效
IncVol	input	板载音量控制+，按钮开关右
DecVol,	input	板载音量控制-，按钮开关左
NextSong	input	板载音乐控制（下一首），按钮开关下
PreSong	input	板载音乐控制（上一首），按钮开关上
DREQ	Input	MP3 数据请求线，显示 VS1003 是否可以接受数据
RXD	Input	蓝牙传入信号
vga_h_sync	Output	显示器行同步信号
vga_v_sync	Output	显示器场同步信号
vga_rgb	Output	VGA 显示器 RGB 值
Bt_Inc_Vol	Output	蓝牙音量控制+
Bt_Dec_Vol	Output	蓝牙音量控制-
Bt_Rst	Output	蓝牙复位
Bt_Next_Song,	Output	蓝牙音乐控制（下一首）
Bt_Pre_Song	Output	蓝牙音乐控制（上一首）
XDCS	Output	数据片选，字节同步
XCS	Output	片选输入，低电平有效
SI	Output	声音传感器有效信号灯
SCLK	Output	SPI 总线时钟,12.288MHZ
XRESET	Output	复位引脚（硬件复位），低电平有效
DEBUG	Output	本接口用于调试，对应板载 J13 LED 灯

3） Verilog 代码

```
`timescale 1ns / 1ps

module TOP(
    input IncVol,
    input DecVol,
    input NextSong,
    input PreSong,

    input CLK,
    input RST,
```

```
    /***********mp3***********/
    input DREQ,
    output  XDCS,    //data
    output  XCS,     //cmd
    output  SI,
    output  SCLK,
    output  XRESET,


    /***********VGA***********/
    output vga_h_sync,
    output vga_v_sync,
    output [11:0]vga_rgb,

    /**********Bluetooth***********/
    input RXD,
    output Bt_Inc_Vol,
    output Bt_Dec_Vol,
    output Bt_Rst,
    output Bt_Next_Song,
    output Bt_Pre_Song,

    /***********调试按钮***********/
    output DEBUG

);
    //三个时钟
    wire clk_65;
    wire clk_12;
    wire clk_1s;

    //音量、歌曲控制
    wire[7:0]vol;
    wire[4:0]music_select;
    wire[7:0]vol_dec;
    wire is_changed_n;

    //时间，秒，分
    wire [3:0]sec_l;
    wire [3:0]sec_h;
    wire [3:0]min_l;
    wire [3:0]min_h;
```

```
    /***********时钟 IP 核*******/
    clk_wiz_1
div65(.clk_in1(CLK),.clk_65(clk_65),.clk_12(clk_12),.reset(0));

    /************蓝牙模块*********/
    Bluetooth inst_bluetooth(
        .Bluetooth_clk(CLK),
        .rst_n(RST),
        .data_in(RXD),
        .Bt_Dec_Vol(Bt_Dec_Vol),
        .Bt_Inc_Vol(Bt_Inc_Vol),
        .Bt_Rst(Bt_Rst),
        .Bt_Pre_Song(Bt_Pre_Song),
        .Bt_Next_Song(Bt_Next_Song)
    );

    /****板载控制与蓝牙控制处理****/
    wire control_RST=RST|Bt_Rst;
    wire control_Vol_inc=IncVol|Bt_Inc_Vol;
    wire control_Vol_dec=DecVol|Bt_Dec_Vol;
    wire control_Song_nex=NextSong|Bt_Next_Song;
    wire control_Song_pre=PreSong|Bt_Pre_Song;

    /**********控制模块***********/
    VolControl uut(
        .clk(CLK),
        .IncVol(control_Vol_inc),
        .DecVol(control_Vol_dec),
        .NextMusic(control_Song_nex),
        .PreMusic(control_Song_pre),
        .vol_dec(vol_dec),
        .is_changed_n(is_changed_n),
        .music_select(music_select),
        .vol(vol)
    );

    /**********音乐播放***********/
    mp3_player myplayer(
        .vol(vol),
        .music_select(music_select),
        .CLK(CLK),
        .DREQ(DREQ),
        .RST(control_RST),
        .XDCS(XDCS),
```

```
        .XCS(XCS),
        .SI(SI),
        .SCLK(SCLK),
        .XRESET(XRESET),
        .DEBUG(DEBUG)
    );

    /**********VGA 显示图片************/
   VGA_Driver vga_ins(
       .music_select(music_select),
       .clk_vga(clk_65),
       .rst_n(RST),
       .min_h(min_h),
       .min_l(min_l),
       .sec_h(sec_h),
       .sec_l(sec_l),
       .vga_rgb(vga_rgb),
       .vol_dec(vol_dec),
       .vga_h_sync(vga_h_sync),
       .vga_v_sync(vga_v_sync)
    );
    //分频器，时钟周期为 1s
    divider1 div1s(.clk(CLK),.clk_1(clk_1s));

    wire time_rst=RST&&is_changed_n;
    TimeDeal
record(.clk(clk_1s),.rst_n(time_rst),.sec_l(sec_l),.sec_h(sec_h)
,.min_l(min_l),.min_h(min_h));

endmodule
```

4） 过程介绍：TOP 模块中，先通过时钟 IP 核获取各个模块对应的时钟信号，然后先调用蓝牙模块，将蓝牙模块得到的控制信号逐个与板载的控制信号作逻辑或操作，这样就可以实现蓝牙和板载按钮的共同控制，之后将信号传入控制模块，修改对应的数据大小，之后再调用音乐播放模块和 VGA 显示模块

2. 音乐播放模块 mp3_player

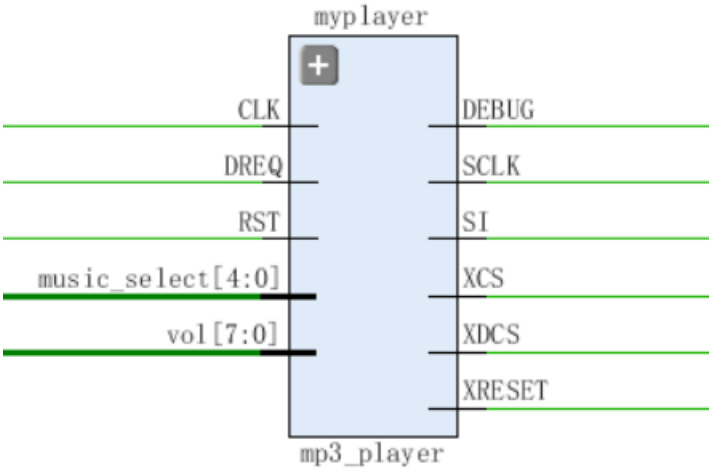
1) 描述:

与 VS1003 模块进行信号、数据传输，从 IP 核中获取音频数据，通过一个 5 选 1 数据选择器选择所需要的数据，同时能够根据传入的数据对当前音乐的音量进行修改

以及进行音乐的切换

2) 接口信号定义：

接口名称	接口属性	接口描述
Vol	input	8 位音量数据
RST	input	复位信号与开始信号,低电平有效
CLK	input	时钟信号
Music_select	input	5 位数据，其中仅有一位为 1，用来切换当前播放的音乐
DREQ	Input	MP3 数据请求线，显示 VS1003 是否可以接受数据
XDCS	Output	数据片选，字节同步
XCS	Output	片选输入，低电平有效
SI	Output	声音传感器有效信号灯
SCLK	Output	SPI 总线时钟
XRESET	Output	复位引脚（硬件复位），低电平有效
DEBUG	Output	本接口用于调试，对应板载 J13 LED 灯



3) Verilog 代码:

```
`timescale 1ns / 1ps

module selector51(
    input[15:0] data0,
    input[15:0] data1,
    input[15:0] data2,
    input[15:0] data3,
    input[15:0] data4,
```

```
    input[4:0]select_msg,
    output reg [15:0] oData
);
always @(*) begin
    if(select_msg==5'b00001)
        oData=data0;
    else if(select_msg==5'b00010)
        oData=data1;
    else if(select_msg==5'b00100)
        oData=data2;
    else if(select_msg==5'b01000)
        oData=data3;
    else if(select_msg==5'b10000)
        oData=data4;
end
endmodule


module mp3_player (
    input [7:0]vol,
    input [4:0]music_select,
    input CLK,
    input DREQ,
    input RST,
    input SO,

    output reg XDCS,    //data
    output reg XCS,     //cmd
    output reg SI,
    output reg SCLK,
    output reg XRESET,
    //add
    output reg DEBUG
);

    wire clk_div;
    //分频
    Clk_wiz div (.clk_in(CLK),.clk_12(clk_div));

    reg[3:0] STATE;
    wire [15:0]douta;
    reg  [15:0]addr=0;
    reg  [15:0]buffer;
    wire [15:0]RomData[4:0];
```

```
    //ip 核获取音频数据
    blk_mem_gen_0
music0(.clka(CLK),.ena(music_select[0]),.addra(addr),.douta(RomData
[0]));        //夜曲
    blk_mem_gen_1
music1(.clka(CLK),.ena(music_select[1]),.addra(addr),.douta(RomData
[1]));        //七里香
    blk_mem_gen_2
music2(.clka(CLK),.ena(music_select[2]),.addra(addr),.douta(RomData
[2]));        //安静
    blk_mem_gen_3
music3(.clka(CLK),.ena(music_select[3]),.addra(addr),.douta(RomData
[3]));        //轨迹
    blk_mem_gen_4
music4(.clka(CLK),.ena(music_select[4]),.addra(addr),.douta(RomData
[4]));        //一路向北

    //通过数据选择器，从 5 个数据中选择
    selector51 select_music(
        .data0(RomData[0]),
        .data1(RomData[1]),
        .data2(RomData[2]),
        .data3(RomData[3]),
        .data4(RomData[4]),
        .select_msg(music_select),
        .oData(douta)
    );

    parameter MAX_DELAY =16600 ;
    integer  delay;

    integer cmd_cnt;
    integer data_cnt;       //计数
    reg[31:0]sci_cmd;       //sci 命令
    reg[7:0]cur_vol;        //音量
    reg[4:0]cur_music;      //当前音乐

    /**********状态机定义***********/
    parameter WAITING  =4'd0 ;
    parameter H_RESET  =4'd1 ;
    parameter S_RESET  =4'd2 ;
    parameter SET_VOL  =4'd3;
    parameter LOAD_DATA=4'd4;
```

```
parameter PLAY     =4'd5 ;

assign music_data=douta;

always @(posedge clk_div) begin
    if(!RST)begin
        XRESET<=1'b0;
        XCS<=1'b1;
        XDCS<=1'b1;
        delay<=0;
        addr<=0;
        DEBUG<=1'b0;
        STATE<=WAITING;
        cur_vol<=vol;
        cur_music<=music_select;
    end
    else if(music_select!=cur_music)begin
        cur_music<=music_select;
        XRESET<=1'b0;
        XCS<=1'b1;
        XDCS<=1'b1;
        delay<=0;
        addr<=0;
        DEBUG<=1'b0;
        cur_vol<=vol;
        STATE<=WAITING;
    end
    else begin
        case(STATE)
            /**********等待状态*******/
            WAITING:begin
                SCLK<=0;
                if(delay==MAX_DELAY)begin
                    STATE<=H_RESET;
                    cmd_cnt<=0;
                    XRESET<=1'b1;
                    delay<=0;
                end
                else
                    delay<=delay+1;
            end
            /*********硬件复位********/
            H_RESET:begin
                cmd_cnt<=0;
```

```
            XCS<=1'b1;
            STATE<=S_RESET;
            sci_cmd<=32'h02000804;
            SCLK<=1'b0;
        end
        /********软复位**********/
        S_RESET:begin
            if(DREQ)begin
                if(SCLK)begin
                    if(cmd_cnt==32)begin
                        cmd_cnt<=0;
                        XCS<=1'b1;
                        STATE<=SET_VOL;
                        sci_cmd<={16'h020b,vol,vol};
                    end
                    else begin
                        XCS<=1'b0;
                        SI<=sci_cmd[31];
                        sci_cmd<={sci_cmd[30:0],sci_cmd[31]};
                        cmd_cnt<=cmd_cnt+1'b1;
                    end
                end
            end
            SCLK<=~SCLK;
        end

    /*********音量控制**********/
        SET_VOL:begin
            if(DREQ)begin
                if(SCLK)begin
                    if(cmd_cnt==32)begin
                        cmd_cnt<=0;
                        XCS<=1'b1;
                        STATE<=LOAD_DATA;
                    end
                    else begin
                        XCS<=0;
                        SI<=sci_cmd[31] ;
                        sci_cmd<={sci_cmd[30:0],sci_cmd[31]};
                        cmd_cnt<=cmd_cnt+1'b1;
                    end
                end
            end
            SCLK<=~SCLK;
```

```
                end
                /********数据装载，可跳转至音量控制*********/
                LOAD_DATA:begin
                    if(cur_vol!=vol)begin
                        cur_vol<=vol;
                        cmd_cnt<=0;
                        STATE<=SET_VOL;
                        sci_cmd<={16'h020b,vol,vol};
                        XCS<=1'b1;
                    end
                    else if(DREQ)begin
                        SCLK<=0;
                        STATE<=PLAY;
                        buffer<=douta;
                        data_cnt<=0;
                    end
                end
                /*******传输数据并播放*********/
                PLAY:begin
                    if(SCLK)begin
                        DEBUG<=1'b1;
                        if(data_cnt==16)begin
                            XDCS<=1'b1;
                            addr<=addr+1'b1;
                            STATE<=LOAD_DATA;
                        end
                        else begin
                            XDCS<=1'b0;
                            SI<=buffer[15];
                            buffer<={buffer[14:0],buffer[15]};
                            data_cnt<=data_cnt+1;
                        end
                    end
                    SCLK=~SCLK;
                end
            endcase
        end
    end
endmodule
```

4). 过程介绍：

VS1003 播放的流程主要是：等待播放信号—>进行硬件复位—>进行软件复位—>通过

sdi 指令设置音量等参数->传输数据，所以适合利用状态机进行实现。状态分别为 WATING、H_RESET、S_RESET、SET_VOL、LOAD_DATA、PLAY 等状态，WAITING 状态用来延时，若不进行延时，会错过一些信号，且 sci 有可能传输出错，导致一些意想不到的错误,具体说面见下图。当延时结束后，状态跳转为硬件复位，当硬件复位结束后，进入软件复位，此时需要处理指令的传输，所以在进行软件复位状态之前，需要先将指令赋值给 sci_cmd，此时再进行软件复位状态，一位一位的传输指令，此时需要注意，每次传输的前提条件是 DREQ 为高以及时钟信号 SCLK 为高，我在此处的操作为每次传输都将 XCS 置低，直到 32 位数据传输结束后，将 XCS 置为高。音量控制状态的注意点与软件复位相似，都是对指令的传输。

当设置完音量后，进入数据装载模块，起初设计时并没用该状态，但在后序实现音量修改功能时，需要在播放的同时进行修改，于是增加了本状态。在本状态中，若检测到了音量变化，则直接跳转至音量控制状态，先修改音量，此时需要注意将 XCS 拉高。若未检测到，则当 DREQ 为高时，将 ROM 中读取的数据传输到寄存器中并转到播放状态。最终的播放状态，需要对音频数据进行传输，同样也是一位一位的传输，当数据传输结束后，拉高 XDCS 并且修改从 ROM 获取数据的地址，不断进行循环，实现音频的播放。推荐做法为每次都检测一下 DREQ，一旦 DREQ 无效了，将当前字节数据传输结束后立刻停止，而不是每传输一整块数据后再进行检测，那样的情况下有时歌曲无法播放，有时会播放一段时间停止。此外，XDCS 在每次传输时拉低，传输结束后拉高电位。

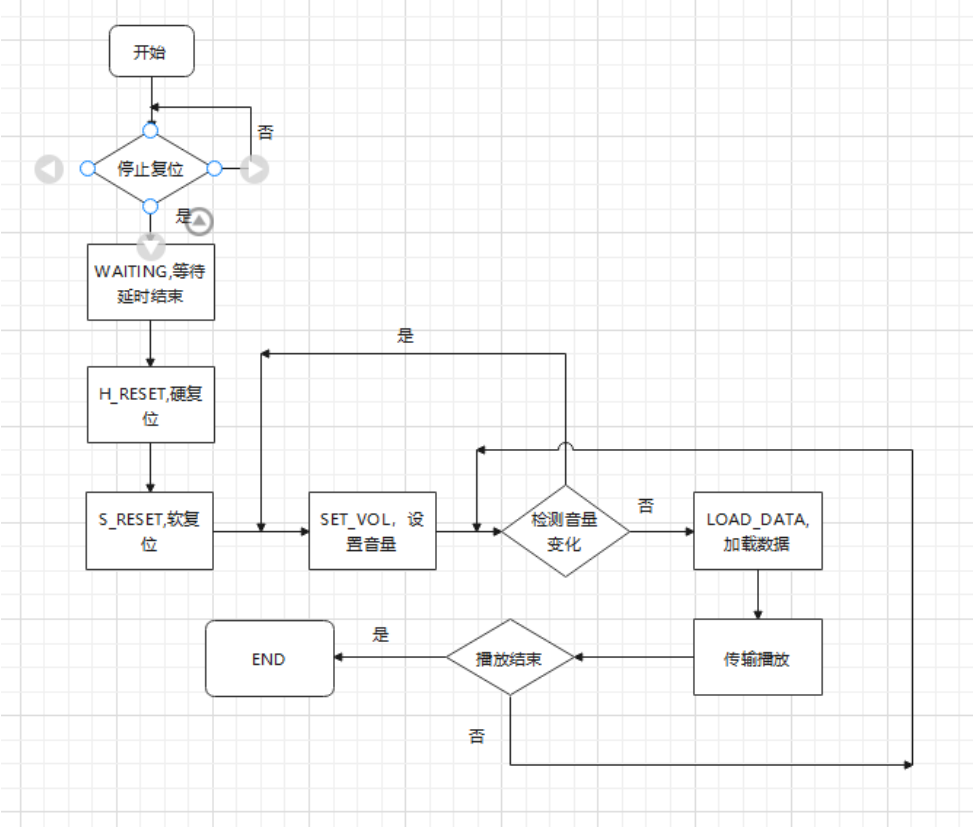
关于时钟的使用，本模块使用手动翻转的方法，在每次操作结束后，翻转 SCLK。

关于延时的说明：

There are cases when you still want to send SCI commands when DREQ is low. Because DREQ is shared between SDI and SCI, you can not determine if a SCI command has been executed if SDI is not ready to receive. In this case you need a long enough delay after every SCI command to make certain none of them is missed. The SCI Registers table in section 8.6 gives the worst-case handling time for each SCI register write.

- Max SCI read clock changed from CLKI/6 to CLKI/7.
- Typical connection diagram updated.
- SCI commands need a fixed delay if DREQ is low.
- AD_DIV documentation fixed.

5). 流程图：



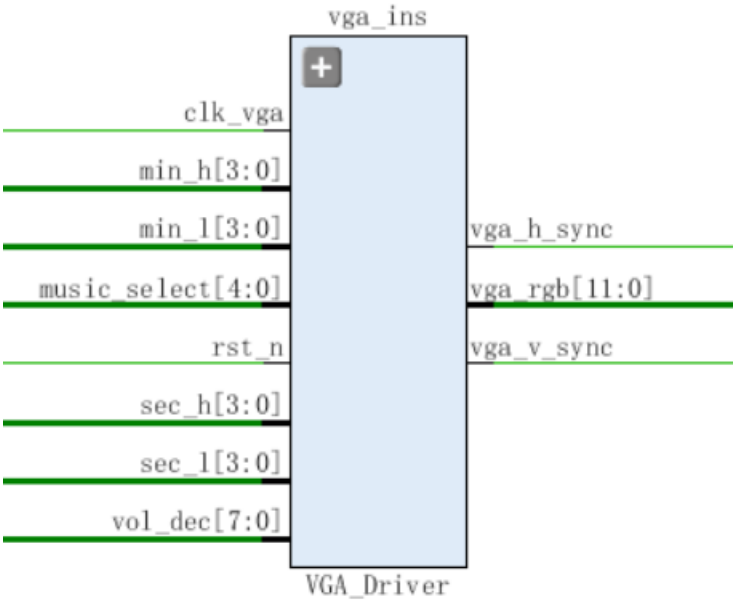
3. 显示模块(VGA_DRIVER)

1) 描述

利用分辨率 1024*768 的 vga 显示器，显示当前播放音乐的专辑封面、当前音乐的音量以及音乐的相关信息。音乐信息的显示实现了移动特效，音量支持动态显示。

2) 接口信号定义：

接口名称	接口属性	接口描述
Music_select	input	5 位音乐数据，决定当前播放的音乐
Clk_vga	input	Vga 显示时钟信号，1024*768 分辨率对应频率为 65MHZ
Rst_n	input	复位信号，低电平有效
Music_select	input	5 位数据，其中仅有一位为 1，用来切换当前播放的音乐
Vol_dec	Input	8 位音量数据
Vga_rgb	Output	Vga 显示的 rgb 值
Vga_h_sync	Output	Vga 行同步信号
Vga_h_sync	Output	Vga 场同步信号
Min_h	Input	秒的十位
Min_l	Input	秒的个位
Sec_h	Input	分钟的十位
Sec_l	input	分钟的个位



3) Verilog 代码

```
module VGA_Driver (
    input [4:0]music_select,
    input clk_vga,  //65MHZ
    input rst_n,
    input [7:0]vol_dec,
    output[11:0] vga_rgb,
    input[3:0]min_l,
    input[3:0]min_h,
    input[3:0]sec_l,
    input[3:0]sec_h,
    output reg vga_h_sync,    //行同步
    output reg vga_v_sync    //场同步
 );

parameter H_SYNC  =11'd136;     //行同步
parameter H_BACK  =11'd160;     //行显示后
parameter H_DISP  =11'd1024;    //行有效数
parameter H_FRONT =11'd24;      //行显示前
parameter H_TOTAL =11'd1344;     //总


parameter V_SYNC  =10'd6 ;
parameter V_BACK  =10'd29 ;
```

```
parameter V_DISP  =10'd768 ;
parameter V_FRONT =10'd3 ;
parameter V_TOTAL =10'd806 ;


/***********图片显示**********/
parameter Image_Begin_x =272 ;
parameter Image_Begin_y =30;
parameter IMAGE_H =480;
parameter IMAGE_V =480;

/***********字模显示**********/
integer   Char_Begin_x =50;
parameter Char_Begin_y =600;
parameter Char_Width=512;
parameter Char_Height=64;

/***********时间显示**********/
parameter Time_Begin_x =480;
parameter Time_Begin_y =510;
parameter Time_Width =160;
parameter Time_Height=64;

/********字模偏移量和数据*****/
wire [9:0]char_x;
wire [9:0]char_y;
wire [511:0]char_data[63:0];
wire char_ena;

/**********时间数据**********/
wire [9:0]time_x;
wire [9:0]time_y;
wire[39:0]time_data_sum[15:0];
wire[7:0]time_data_1[15:0];
wire[7:0]time_data_2[15:0];
wire[7:0]time_data_3[15:0];
wire[7:0]time_data_4[15:0];
wire[7:0]time_data_5[15:0];
wire time_ena;

/***********音量信息**********/
parameter Vol_Col_Begin_x1 =100;
parameter Vol_Col_Begin_x2 =824;
parameter Vol_Col_Width =100;
```

```
parameter Vol_Col_BASE =480 ;
parameter Vol_Per_Height=28;
wire vol_ena;


/*******字模数据设置********/
Char_Set inst_charset(
    .music_select(music_select),
    .rst_n(rst_n),
    .row0(char_data[0]),
    .row1(char_data[1]),
    .row2(char_data[2]),
    .row3(char_data[3]),
    .row4(char_data[4]),
    .row5(char_data[5]),
    .row6(char_data[6]),
    .row7(char_data[7]),
    .row8(char_data[8]),
    .row9(char_data[9]),
    .row10(char_data[10]),
    .row11(char_data[11]),
    .row12(char_data[12]),
    .row13(char_data[13]),
    .row14(char_data[14]),
    .row15(char_data[15]),
    .row16(char_data[16]),
    .row17(char_data[17]),
    .row18(char_data[18]),
    .row19(char_data[19]),
    .row20(char_data[20]),
    .row21(char_data[21]),
    .row22(char_data[22]),
    .row23(char_data[23]),
    .row24(char_data[24]),
    .row25(char_data[25]),
    .row26(char_data[26]),
    .row27(char_data[27]),
    .row28(char_data[28]),
    .row29(char_data[29]),
    .row30(char_data[30]),
    .row31(char_data[31]),
    .row32(char_data[32]),
    .row33(char_data[33]),
    .row34(char_data[34]),
```

```
    .row35(char_data[35]),
    .row36(char_data[36]),
    .row37(char_data[37]),
    .row38(char_data[38]),
    .row39(char_data[39]),
    .row40(char_data[40]),
    .row41(char_data[41]),
    .row42(char_data[42]),
    .row43(char_data[43]),
    .row44(char_data[44]),
    .row45(char_data[45]),
    .row46(char_data[46]),
    .row47(char_data[47]),
    .row48(char_data[48]),
    .row49(char_data[49]),
    .row50(char_data[50]),
    .row51(char_data[51]),
    .row52(char_data[52]),
    .row53(char_data[53]),
    .row54(char_data[54]),
    .row55(char_data[55]),
    .row56(char_data[56]),
    .row57(char_data[57]),
    .row58(char_data[58]),
    .row59(char_data[59]),
    .row60(char_data[60]),
    .row61(char_data[61]),
    .row62(char_data[62]),
    .row63(char_data[63])
);

/********时间数据读取********/
Time_Set data1(
    .num(sec_l),
    .rst_n(rst_n),
    .row0(time_data_1[0]),.row1(time_data_1[1]),
    .row2(time_data_1[2]),.row3(time_data_1[3]),
    .row4(time_data_1[4]),.row5(time_data_1[5]),
    .row6(time_data_1[6]),.row7(time_data_1[7]),
    .row8(time_data_1[8]),.row9(time_data_1[9]),
    .row10(time_data_1[10]),.row11(time_data_1[11]),
    .row12(time_data_1[12]),.row13(time_data_1[13]),
    .row14(time_data_1[14]),.row15(time_data_1[15])
);
```

```
Time_Set data2(
    .num(sec_h),
    .rst_n(rst_n),
    .row0(time_data_2[0]),.row1(time_data_2[1]),
    .row2(time_data_2[2]),.row3(time_data_2[3]),
    .row4(time_data_2[4]),.row5(time_data_2[5]),
    .row6(time_data_2[6]),.row7(time_data_2[7]),
    .row8(time_data_2[8]),.row9(time_data_2[9]),
    .row10(time_data_2[10]),.row11(time_data_2[11]),
    .row12(time_data_2[12]),.row13(time_data_2[13]),
    .row14(time_data_2[14]),.row15(time_data_2[15])
);
Time_Set data3(
    .num(4'b1010),
    .rst_n(rst_n),
    .row0(time_data_3[0]),.row1(time_data_3[1]),
    .row2(time_data_3[2]),.row3(time_data_3[3]),
    .row4(time_data_3[4]),.row5(time_data_3[5]),
    .row6(time_data_3[6]),.row7(time_data_3[7]),
    .row8(time_data_3[8]),.row9(time_data_3[9]),
    .row10(time_data_3[10]),.row11(time_data_3[11]),
    .row12(time_data_3[12]),.row13(time_data_3[13]),
    .row14(time_data_3[14]),.row15(time_data_3[15])
);
Time_Set data4(
    .num(min_l),
    .rst_n(rst_n),
    .row0(time_data_4[0]),.row1(time_data_4[1]),
    .row2(time_data_4[2]),.row3(time_data_4[3]),
    .row4(time_data_4[4]),.row5(time_data_4[5]),
    .row6(time_data_4[6]),.row7(time_data_4[7]),
    .row8(time_data_4[8]),.row9(time_data_4[9]),
    .row10(time_data_4[10]),.row11(time_data_4[11]),
    .row12(time_data_4[12]),.row13(time_data_4[13]),
    .row14(time_data_4[14]),.row15(time_data_4[15])
);
Time_Set data5(
    .num(min_h),
    .rst_n(rst_n),
    .row0(time_data_5[0]),.row1(time_data_5[1]),
    .row2(time_data_5[2]),.row3(time_data_5[3]),
    .row4(time_data_5[4]),.row5(time_data_5[5]),
    .row6(time_data_5[6]),.row7(time_data_5[7]),
    .row8(time_data_5[8]),.row9(time_data_5[9]),
```

```
    .row10(time_data_5[10]),.row11(time_data_5[11]),
    .row12(time_data_5[12]),.row13(time_data_5[13]),
    .row14(time_data_5[14]),.row15(time_data_5[15])
);

assign
time_data_sum[0]={time_data_5[0],time_data_4[0],time_data_3[0],t
ime_data_2[0],time_data_1[0]};
assign
time_data_sum[1]={time_data_5[1],time_data_4[1],time_data_3[1],t
ime_data_2[1],time_data_1[1]};
assign
time_data_sum[2]={time_data_5[2],time_data_4[2],time_data_3[2],t
ime_data_2[2],time_data_1[2]};
assign
time_data_sum[3]={time_data_5[3],time_data_4[3],time_data_3[3],t
ime_data_2[3],time_data_1[3]};
assign
time_data_sum[4]={time_data_5[4],time_data_4[4],time_data_3[4],t
ime_data_2[4],time_data_1[4]};
assign
time_data_sum[5]={time_data_5[5],time_data_4[5],time_data_3[5],t
ime_data_2[5],time_data_1[5]};
assign
time_data_sum[6]={time_data_5[6],time_data_4[6],time_data_3[6],t
ime_data_2[6],time_data_1[6]};
assign
time_data_sum[7]={time_data_5[7],time_data_4[7],time_data_3[7],t
ime_data_2[7],time_data_1[7]};
assign
time_data_sum[8]={time_data_5[8],time_data_4[8],time_data_3[8],t
ime_data_2[8],time_data_1[8]};
assign
time_data_sum[9]={time_data_5[9],time_data_4[9],time_data_3[9],t
ime_data_2[9],time_data_1[9]};
assign
time_data_sum[10]={time_data_5[10],time_data_4[10],time_data_3[1
0],time_data_2[10],time_data_1[10]};
assign
time_data_sum[11]={time_data_5[11],time_data_4[11],time_data_3[1
1],time_data_2[11],time_data_1[11]};
assign
time_data_sum[12]={time_data_5[12],time_data_4[12],time_data_3[1
2],time_data_2[12],time_data_1[12]};
```

```
assign
time_data_sum[13]={time_data_5[13],time_data_4[13],time_data_3[1
3],time_data_2[13],time_data_1[13]};
assign
time_data_sum[14]={time_data_5[14],time_data_4[14],time_data_3[1
4],time_data_2[14],time_data_1[14]};
assign
time_data_sum[15]={time_data_5[15],time_data_4[15],time_data_3[1
5],time_data_2[15],time_data_1[15]};

/***********行列计数************/
reg[10:0]h_cnt;
reg[10:0]v_cnt;
wire image_ena;

/***********Rom 取值************/
reg [18:0]ADR;
reg [4:0]cur_music;

/**********时间字模************/
assign time_ena=(h_cnt-H_SYNC-H_BACK>Time_Begin_x)&&(h_cnt-
H_SYNC-H_BACK-Time_Begin_x<Time_Width)&&(v_cnt-V_SYNC-
V_BACK>Time_Begin_y)&&(v_cnt-V_SYNC-V_BACK-
Time_Begin_y<Time_Height);
assign time_x=h_cnt-H_SYNC-H_BACK-Time_Begin_x;
assign time_y=v_cnt-V_SYNC-V_BACK-Time_Begin_y;

/**********音量设置***********/
assign
vol_ena=(((h_cnt>H_SYNC+H_BACK+Vol_Col_Begin_x1)&&(h_cnt<H_SYNC+
H_BACK+Vol_Col_Begin_x1+Vol_Col_Width))

||((h_cnt>H_SYNC+H_BACK+Vol_Col_Begin_x2)&&(h_cnt<H_SYNC+H_BACK+
Vol_Col_Begin_x2+Vol_Col_Width)))
       &&(v_cnt>V_SYNC+V_BACK+Vol_Col_BASE-
Vol_Per_Height*vol_dec)&&(v_cnt<V_SYNC+V_BACK+Vol_Col_BASE);

/*********字模数据相对偏移量************/
assign
char_ena=(h_cnt>H_SYNC+H_BACK+Char_Begin_x)&&(h_cnt<=H_SYNC+H_BA
CK+Char_Begin_x+Char_Width)&&

(v_cnt>V_SYNC+V_BACK+Char_Begin_y)&&(v_cnt<=V_SYNC+V_BACK+Char_B
egin_y+Char_Height);
```

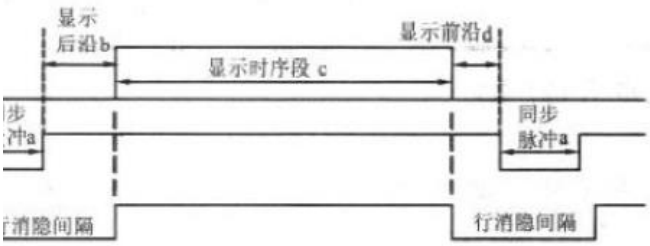
```
assign char_x= (h_cnt-H_SYNC-H_BACK-Char_Begin_x)>=0?(h_cnt-
H_SYNC-H_BACK-Char_Begin_x):(H_DISP+h_cnt-H_SYNC-H_BACK-
Char_Begin_x);
assign char_y= v_cnt-H_SYNC-H_BACK-Char_Begin_y;

always @(posedge clk_vga) begin
    if(!rst_n)begin
        cur_music<=music_select;
    end
    else if(cur_music!=music_select)begin
        cur_music<=music_select;
    end
end

always @(posedge clk_vga) begin
    if(!rst_n||cur_music!=music_select)
        h_cnt<=0;
    else if(h_cnt<H_TOTAL-1'b1)
        h_cnt<=h_cnt+1'b1;
    else
        h_cnt<=0;
end

always @(posedge clk_vga) begin
    if(!rst_n||cur_music!=music_select)
        v_cnt<=0;
    else if(h_cnt==H_TOTAL-1'b1)begin
        if(v_cnt<V_TOTAL-1'b1)
            v_cnt<=v_cnt+1'b1;
        else begin
            v_cnt<=0;
            if(Char_Begin_x+Char_Width>0)
                Char_Begin_x<=Char_Begin_x-1;
            else
                Char_Begin_x<=1000;
        end
    end
end

always @(posedge clk_vga ) begin
    if(!rst_n||cur_music!=music_select)
        vga_h_sync<=1'b1;
    else if(h_cnt>=0&&h_cnt<H_SYNC)
        vga_h_sync<=1'b0;
```

```
    else
        vga_h_sync<=1'b1;
end

always @(posedge clk_vga) begin
    if(!rst_n||cur_music!=music_select)
        vga_v_sync<=1'b1;
    else if(v_cnt>=0&&v_cnt<V_SYNC)
        vga_v_sync<=0;
    else
        vga_v_sync<=1'b1;
end


assign
image_ena=(h_cnt>=H_SYNC+H_BACK+Image_Begin_x)&&(h_cnt<H_SYNC+H_
BACK+Image_Begin_x+IMAGE_H)&&(v_cnt>=V_SYNC+V_BACK+Image_Begin_y
)&&(v_cnt<V_SYNC+V_BACK+Image_Begin_y+IMAGE_V)&&(rst_n);

always @(posedge clk_vga) begin
    if(!rst_n||cur_music!=music_select)
        ADR=0;
    else if(h_cnt==0&&v_cnt==0)
        //ADR=1;
        ADR=0;
    else if(image_ena)begin
        //ADR=(ADR+1);
        ADR=(v_cnt-V_SYNC-V_BACK-Image_Begin_y)/2*240+(h_cnt-
H_SYNC-H_BACK-Image_Begin_x)/2;
    end
    else if(ADR==(IMAGE_H*IMAGE_V)/4-1)
        ADR=0;
end

wire [11:0]image_data[4:0];
wire [11:0]douta;
reg [11:0]tmp_rgb;

//先图片，再音量，最后字模
always @(posedge clk_vga) begin
    if(image_ena)
        tmp_rgb<=douta;
```

```
    else
if(h_cnt<H_SYNC+H_BACK||h_cnt>H_SYNC+H_BACK+H_DISP||v_cnt<V_SYNC
+V_BACK||v_cnt>V_SYNC+V_BACK+V_DISP)
        tmp_rgb<=0;
    else if(time_ena&&time_data_sum[time_y/4][10'd39-time_x/4])
            tmp_rgb<=12'b0;
    else if(vol_ena)
        tmp_rgb<=12'b0000_1111_0000;
    else if(char_ena&&char_data[char_y][10'd511-char_x])
        tmp_rgb<=12'b1111_0000_1111;
    else
        tmp_rgb<=12'b1111_1111_1111;
end

assign vga_rgb=tmp_rgb;

/**********ROM 数据获取**************/
blk_mem_gen_0_pic
getData0(.clka(clk_vga),.ena(1),.addra(ADR),.douta(image_data[0]
));
blk_mem_gen_1_pic
getData1(.clka(clk_vga),.ena(1),.addra(ADR),.douta(image_data[1]
));
blk_mem_gen_2_pic
getData2(.clka(clk_vga),.ena(1),.addra(ADR),.douta(image_data[2]
));
blk_mem_gen_3_pic
getData3(.clka(clk_vga),.ena(1),.addra(ADR),.douta(image_data[3]
));
blk_mem_gen_4_pic
getData4(.clka(clk_vga),.ena(1),.addra(ADR),.douta(image_data[4]
));

/************5 选 1 数据选择器*********/
selector51 select_img(
        .data0(image_data[0]),
        .data1(image_data[1]),
        .data2(image_data[2]),
        .data3(image_data[3]),
        .data4(image_data[4]),
        .select_msg(music_select),
        .oData(douta)
    );
endmodule
```

4) 过程介绍

本模块首先要完成 vga 的基本显示。经过学习，可以发现 vga 的时序如下图所示。



VGA 的行时序

VGA 的场时序

Vga 在一次时序中有 4 个不同状态，在行扫描时，同步脉冲为数据行的结束标志，标志着本行数据的结束和下一行数据的开始标志，同步脉冲为低电平有效，即在此段需要将 vga_h_sync 拉低。同步脉冲结束后为显示后沿， 之后则为显示时序段，此时 vga 获取颜色数据，将其显示在屏幕中，板载 vga 接口所需的数据为 12bitRGB 值。在显示时序段结束后，为显示前沿。显示前沿和显示后沿不显示数据。场扫描的时序同行扫描。VGA 显示的逐行扫描时，从左上角开始，从左向右扫描，扫描完一行，就到下一行的最左边当所有行扫描结束后，形成一帧，并用场同步信号进行同步。

在解决了 VGA 的基本显示后，就可以开始进行图片的显示，利用 ROM 存放数据，每次从 ROM 中读取图片的 RGB 的信息。每读到一个信息，就将地址加一，直到图片数据读取结束。同时需要注意当前扫描到的位置是否是图片的有效位置，若不是，则不应该进行显示。

对于音量，通过柱形图来显示，柱形图的宽度以及单层高度自行选定，柱形图的最大高

度应该为 16 层（音量范围为 0x00-0xf0）。选定好柱形图的底层位置 base，之后根据输入的音量动态修改最高层的位置，即可实现音量的动态显示，同样的，需要注意音量显示的有效位置，当当前扫描位置无效时，不应该进行显示。

对于字模的显示，通过 char_set 模块进行数据的读取，读取到一个 512 像素宽，64 像素高的字模数据。之后设定好字模的显示行起点、列起点、显示长度、显示宽度，当扫描位置有效时，按照二维数组的方法对数据进行读取，读取时的坐标应该采用相对显示起点的偏移坐标。同时需要注意的是，verilog 中高位数据在前，所以应该要将显示宽度减去偏移坐标的 x 分量，才能读取到正确的字模数据，否则会出现左右逆置的现象。

对于时间字模的显示，大体上同 char_set 模块，只不过输入值修改为时间，操作都是相同的。

5) 调试过程

注意点 1：在 vga 扫描过程中，若当前不为显示时序段，则需要手动将 rgb 值置为 0，否则会出现无法正常显示的情况。在项目调试的过程中，试图修改背景颜色，但是遗漏了以上注意点，导致 vga 无法显示

注意点 2：图片数据通过 ROM 获取时，需要注意空间大小问题，在调试过程中成功显示了一张 440*440 像素的图片，但是若添加 2 张及以上的图片，若仍保持 440*440 像素不变，在综合时会显示空间不足。解决方法对图片大小进行修改。在本项目中，选择的图片均为 240*240 像素，在 vga 显示时通过多像素共用数据的方法实现 480*480 像素的显示，即 vga 显示过程中 2*2 的像素使用图片的一个数据。假定在 480*480 的图片中一个像素的坐标为(x,y)，则其对应使用的像素数据即为 240*240 像素图片中坐标为(x/2,y/2)的像素数据，转化为一维数组，对应的下标就为(y/2)*240+x/2。通过这样的“放大”，仍能获得较高清晰度的图片。对比见下图。

注意点 3：vga 接受的数据应该直接为每个像素的 RGB 值，所以选择的图片类型应该是 bmp 格式,同时应该过滤掉 bmp 文件的文件头信息，避免文件头信息对数据进行干扰。在获取数据时，我的方法是直接取 24 位 bmp 图片中每个数据的高四位组成图片。对应的 c++代码见下图。

对比图：

440*440 像素图片直接显示

240*240 像素等比放大

生成 RGB 数据的 c++代码

```
1.  void hdc_bitmap_image(const char* filename, const int point_x = 0, co
    nst int point_y = 0)
2.  {
3.      /* 通过带一参的构造函数方式，读取指定 bmp 的格式及所有点的颜色信息 */
4.      ofstream myout("picture.txt", ios::out );
5.      if (!myout.is_open())
6.          exit(-3);
7.      bitmap_image bmp(filename);
8.      /* 从上到下、从左到右显示整个图像 */
9.      for (int i = 0; i < bmp.height(); i++)
10.     {
11.         for (int j = 0; j < bmp.width(); j++)
12.         {
13.             unsigned int num = bmp.get_pixel(i, j);
14.             char buffer[30];
15.             string R, G, B;
16.             _itoa(num,buffer,2);
17.             string tmp = buffer;
18.             while (tmp.length() != 24)
19.                 tmp.insert(tmp.begin(), '0');
20.             for (int i = 0; i < 4; i++){
21.                 myout << tmp[i];
22.                 B += tmp[i];
23.             }
24.             for (int i = 8; i < 12; i++){
25.                 myout << tmp[i];
26.                 G += tmp[i];
```

```
            }
            for (int i = 16; i < 20; i++) {
                myout << tmp[i];
                R += tmp[i];
            }
            myout << ',' << endl;
            hdc_set_pencolor((unsigned char)atoi(R.c_str()),(unsigned
    char)atoi(G.c_str()), (unsigned char)atoi(B.c_str()));
            hdc_base_point(j,i);
        }
    }
    myout.close();
}
```

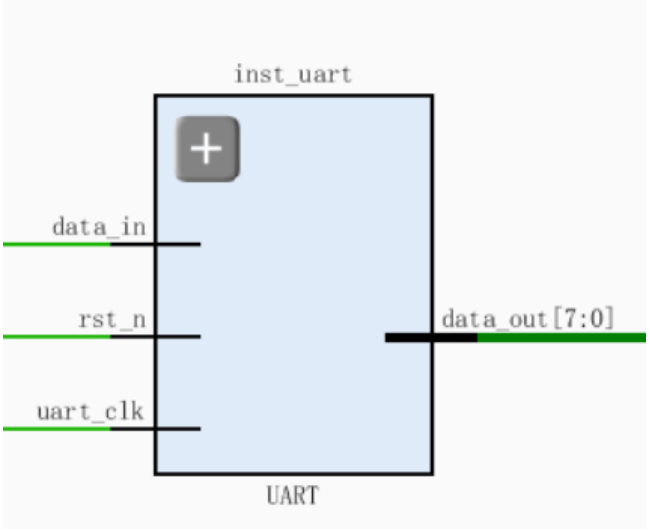
4. 蓝牙传输模块（UART）

1). 描述

实现 UART 传输协议，将蓝牙输入的信号通过 data_out 输出，输出数据为 8 位

2). 接口定义

接口名称	接口属性	接口描述
Uart_clk	input	时钟信号，上升沿有效，100MHZ
Rst_n	input	复位信号，低电平有效
Data_in	input	手机端输入的信号
Data_out	output	蓝牙接收端获取的 8 位数据
Vga_rgb	Output	Vga 显示的 rgb 值
Vga_h_sync	Output	Vga 行同步信号
Vga_h_sync	Output	Vga 场同步信号



3). Verilog 代码

```
module UART (
```

```
    input uart_clk,
    input rst_n,
    input data_in,
    output reg [7:0]data_out
);
parameter bps =10417 ;      //9600
reg[15:0] cnt0; //波特率计数
reg[3:0]  cnt1; //数据位计数

reg detect_edge_0,detect_edge_1,detect_edge_2;
wire posedge_state;
reg uart_state;

/*****************检测下降***********************/
always @(posedge uart_clk ) begin
    if(!rst_n)begin
        detect_edge_0<=1;
        detect_edge_1<=1;
        detect_edge_2<=1;
    end
    else begin
        detect_edge_0<=data_in;
        detect_edge_1<=detect_edge_0;
        detect_edge_2<=detect_edge_1;
    end
end

assign posedge_state=~detect_edge_1&detect_edge_2;//是否是上升沿

/*******************波特率计数******************/
always @(posedge uart_clk) begin
    if(!rst_n)
        cnt0<=0;
    else if(uart_state)begin
        if(cnt0==bps-1'b1)
            cnt0<=0;
        else
            cnt0<=cnt0+1'b1;
    end
end

/*******************数据计数***********************/
always @(posedge uart_clk ) begin
    if(!rst_n)
```

```
        cnt1<=0;
    else if(uart_state&&cnt0==bps-1'b1)begin
        if(cnt1==4'd8)
            cnt1<=0;
        else
            cnt1<=cnt1+1'b1;
    end
end

/**********状态调整***************/
always @ (posedge uart_clk)begin
    if(!rst_n)begin
        uart_state<=0;
    end
    else if(posedge_state)
        uart_state<=1;
    else if(uart_state&&cnt1==8&&cnt0==bps-1)
        uart_state<=0;
end

/************数据读取***********/
always @(posedge uart_clk) begin
    if(!rst_n)begin
        data_out<=0;
    end
    else if(uart_state&&cnt0==bps/2-1&&cnt1!=0)begin
        data_out[cnt1-1]<=data_in;
    end
end

endmodule
```

4). 过程介绍

Uart 串口通讯的原理是将待传输的数据一位一位地传输，输入数据通过 data_in，输出数据通过 data_out 传出。传输的格式规定为“起始位+八位数据+奇偶校验+停止”，空闲时始终为高点位，即 data_in 为 1，起始位规定为 0，所以要先判断是否进入了传输过程，也就是对 dataa_in 的下降沿进行检测，利用三个寄存器，第一个寄存器获取 data_in 的数据，另外两个寄存器获取相对于当前寄存器的上一个寄存器的值，通过这三个寄存器的值高低电位进行即可判断状态，当第二个寄存器为 0，第三个寄存器为 1 时，就代表传

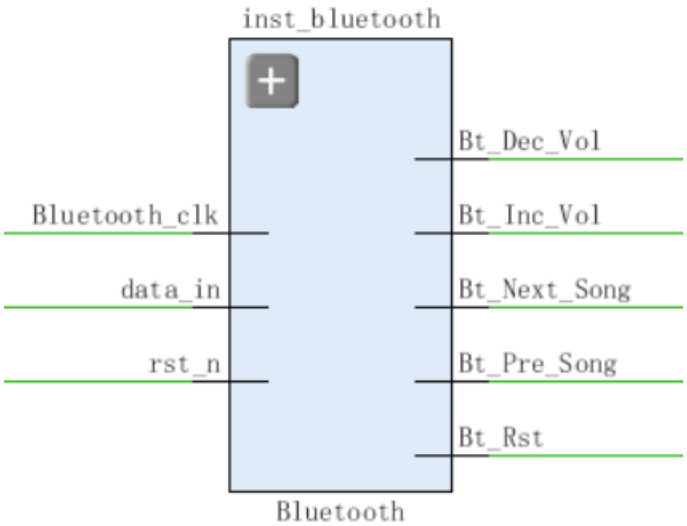
输开始，此时再将传输状态寄存器拉高，开始接受数据。接收数据要根据波特率进行计数。波特率为 9600，时钟为 100MHZ，周期为 10ns，所以计数的个数为 104166/10=10417。即每计数 10417，就传输一位数据，为了便于调试，设定在计数到达 10417/2-1 时进行传输。当 8 位数据传输结束后，再将传输状态寄存器拉低，等待下一次传输。

5. 蓝牙控制模块(Bluetooth)

1).描述

2).接口定义

接口名称	接口属性	接口描述
Bluetooth_clk	input	时钟信号，上升沿有效，100MHZ
Rst_n	input	复位信号，低电平有效
Data_in	input	手机端输入的信号
Bt_Inc_Vol	output	蓝牙音量+信号，高电平有效
Bt_Dec_Vol	Output	蓝牙音量-信号，高电平有效
Bt_Next_Song	Output	蓝牙歌曲切换上信号，高电平有效
Bt_Pre_Song	Output	蓝牙歌曲切换下信号，高电平有效
Bt_Rst	Output	蓝牙复位信号，高电平有效



4).verilog 代码

```
module Bluetooth (
    input Bluetooth_clk,
    input rst_n,
    input data_in,
    output Bt_Inc_Vol,
    output Bt_Dec_Vol,
    output Bt_Next_Song,
    output Bt_Pre_Song,
    output Bt_Rst
);

wire [7:0]data_out;
UART inst_uart(
    .uart_clk(Bluetooth_clk),
    .data_in(data_in),
    .rst_n(rst_n),
    .data_out(data_out)
    );

assign  Bt_Dec_Vol   = (data_out==8'hb0)?1'b1:1'b0;
assign  Bt_Inc_Vol   = (data_out==8'hb1)?1'b1:1'b0;
assign  Bt_Next_Song = (data_out==8'hb2)?1'b1:1'b0;
assign  Bt_Pre_Song  = (data_out==8'hb3)?1'b1:1'b0;
assign  Bt_Rst       = (data_out==8'hb4)?1'b1:1'b0;


endmodule
```

4).过程分析

在本模块中，实例化模块 UART，通过 UART 从蓝牙端获取信号。再将蓝牙端获取的信号转化为控制信号。信号对应关系如下图

指令数据	指令描述
8’hb0	音量-信号
8’hb1	音量+信号
8’hb2	下一首歌信号
8’hb3	上一首歌信号
8’hb4	复位信号

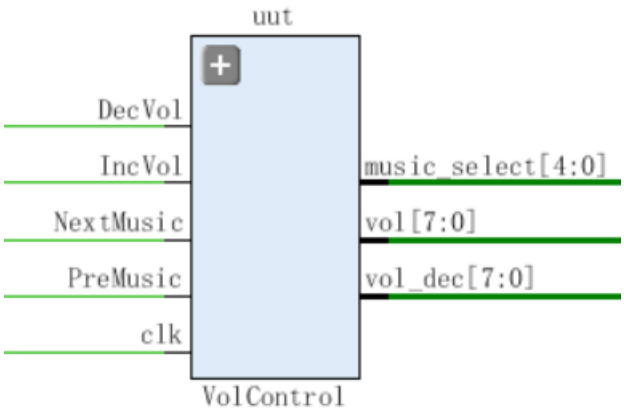
6. 控制信号处理模块(vol_control)

1). 描述

根据控制信号的值，修改各参数变量的值

2). 接口介绍

接口名称	接口属性	接口描述
Clk	input	时钟信号，100MHZ
IncVol	input	音量+信号（蓝牙+板载）
DecVol	Input	音量-信号（蓝牙+板载）
PreMusic	Input	上一首歌（蓝牙+板载）
NextMusic	Input	下一首歌（蓝牙+板载）
Vol	Output	音量数据
Vol_dec	Output	音量挡位(0-16)
Music_select	Output	5 位音乐播放信息



3). Verilog 代码

```
`timescale 1ns / 1ps

module VolControl(
    input clk,
    input IncVol,
    input DecVol,
    input PreMusic,
    input NextMusic,
    output [7:0]vol,
    output [7:0]vol_dec,
    output [4:0]music_select
    );

    //寄存器，保存上一个状态，用来检测变化
    reg [7:0]pre_vol=8'h60;
    reg [7:0]pre_vol_dec=8'd10;
    reg [4:0]cur_music=5'b01000;
```

```
    //需要有延时，否则会导致音量调整失效
    parameter delay_time = 10000000;
    integer  cnt_delay=0;

    always @(negedge clk) begin
        if(cnt_delay==delay_time)begin
            cnt_delay<=0;
            if(IncVol&&pre_vol>0)begin
                pre_vol<=pre_vol-8'h10;
                pre_vol_dec<=pre_vol_dec+8'd1;
            end
            else if(DecVol&&pre_vol<8'hf0)begin
                pre_vol<=pre_vol+8'h10;
                pre_vol_dec<=pre_vol_dec-8'd1;
            end

            //移位操作
            if(PreMusic)begin
                cur_music<={cur_music[0],cur_music[4:1]};
            end
            else if(NextMusic)begin
                cur_music<={cur_music[3:0],cur_music[4]};
            end
        end
        else
            cnt_delay<=cnt_delay+1;
    end

    assign vol=pre_vol;
    assign music_select=cur_music;
    assign vol_dec=pre_vol_dec;

endmodule
```

传入信号

```
/****板载控制与蓝牙控制处理****/
    wire control_RST=RST|Bt_Rst;
    wire control_Vol_inc=IncVol|Bt_Inc_Vol;
    wire control_Vol_dec=DecVol|Bt_Dec_Vol;
    wire control_Song_nex=NextSong|Bt_Next_Song;
    wire control_Song_pre=PreSong|Bt_Pre_Song;
```

4). 过程分析

音乐播放的控制通过一个移位操作实现，移位数据共 5 位，初始化为 5’b00001，当传入切换信号为高点位时，只需要将数据进行循环左移（右移）即可，将对应的数据作为 ROM 数据的使能端，即可实现音乐播放的切换。

音量的控制通过直接的加减操作实现，根据 vs1003 模块的说明，mp3 的音量通过指令的低 16 位赋值，[7:0]和[15:8]分别为左右声道的音量，在本项目，音量增加或减少的单次变化量为 0x10,这样整体的音量就分为了 16 个挡位，数值越低，音量越高。

由于播放、显示等操作在时钟的上升沿操作，所以控制模块整体应该在下降沿操作，这样可以避免时序混乱产生的错误。

5). 调试过程

注意点 1：控制模块同样也要加入信号，在最初实现时，并没有加入时钟信号，仅仅判断信号是否为高电位，导致无法控制音量，甚至一旦调整音量，音乐播放就停止了。解决方法是加入时钟，或者在对应按键信号的上升沿触发。但由于按键信号较多，且需要对同一个寄存器变量进行修改，多个 always 中出现了对同一个变量的赋值，导致综合出错，所以并没有选择第二种方法，而是加入了时钟。

注意点 2：模块中需要加入延时，避免按键时间响应较长，导致一次按键出现多次音量的加减或音乐的切换。

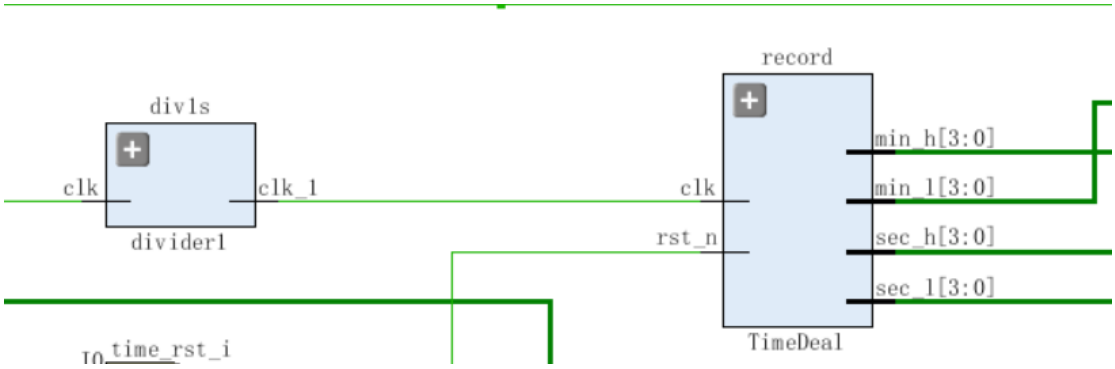
7. 计时模块(Time_Deal，包含一个分频器)

1), 描述

利用输入周期为 1s 的时钟，完成计数，返回分、秒的十位和个位

2). 接口定义

接口名称	接口属性	接口描述
Clk	input	时钟信号，1HZ
Rst_n	input	复位信号
Min_l	output	分钟的个位
Min_h	output	分钟的十位
Sec_l	output	秒的个位
Sec_h	output	秒的十位



3). Verilog 代码

```
module divider1 (
    input clk,
    output reg clk_1
);
parameter NUM_DIV =100000000 ;
reg [31:0]cnt;
always @(posedge clk) begin
    if(cnt<NUM_DIV/2-1)begin
        cnt<=cnt+1'b1;
        clk_1<=clk_1;
    end
    else begin
        cnt<=32'b0;
        clk_1<=~clk_1;
    end
end
endmodule

module TimeDeal  (
    input clk,//1s
    input rst_n,
    output [3:0]min_l,
    output [3:0]min_h,
    output [3:0]sec_l,
    output [3:0]sec_h
);

reg[3:0]dout1;
reg[3:0]dout2;
reg[3:0]dout3;
reg[3:0]dout4;
//秒的高、低位，分钟的高、低位
```

```
reg carry1,carry2,carry3;
//进位

//先处理秒的低位
always @(posedge clk or negedge rst_n)  begin
    if(!rst_n)
        dout1<=0;
    else begin
        if(dout1<9)begin
            dout1<=dout1+1'b1;
            carry1<=0;
        end
        else begin
            dout1<=0;
            carry1<=1'b1;
        end

    end
end

//再处理秒的高位
always @(posedge carry1 or negedge rst_n) begin
    if(!rst_n)
        dout2<=0;
    else begin
        if(dout2<5)begin
            dout2<=dout2+1'b1;
            carry2<=0;
        end
        else begin
            dout2<=0;
            carry2<=1'b1;
        end
    end
end

//再处理分钟的低位
always @(posedge carry2 or negedge rst_n) begin
    if(!rst_n)
        dout3<=0;
    else begin
        if(dout3<9)begin
            dout3<=dout3+1'b1;
            carry3<=0;
```

```
        end
        else begin
            dout3<=0;
            carry3<=1'b1;
        end
    end
end

//再处理分钟的低位
always @(posedge carry3 or negedge rst_n) begin
    if(!rst_n)
        dout4<=0;
    else begin
        if(dout4<6)begin
            dout4<=dout4+1'b1;
        end
        else begin
            dout4<=0;
        end
    end
end

assign sec_l=dout1;
assign sec_h=dout2;
assign min_l=dout3;
assign min_h=dout4;

endmodule
```

4). 过程介绍

板载时钟为 100MHZ,可以通过一个分频器将得到 1HZ 的时钟，根据这个时钟来计时。计时模块分为 4 个部分，分别为秒的个位、十位、分钟的个位、十位计时。对于秒的个位，应该在每次时钟的上升沿或复位的下降沿触发，具体实现即为一个模 10 计数器，满 10 就产生一个进位，秒的十位则在进位的上升沿或下降沿触发，为一个模 6 计数器，同样产生进位，分钟的个位与十位与秒类似，不过都是由进位来触发，这样就完成了支持异步复位的计时器。

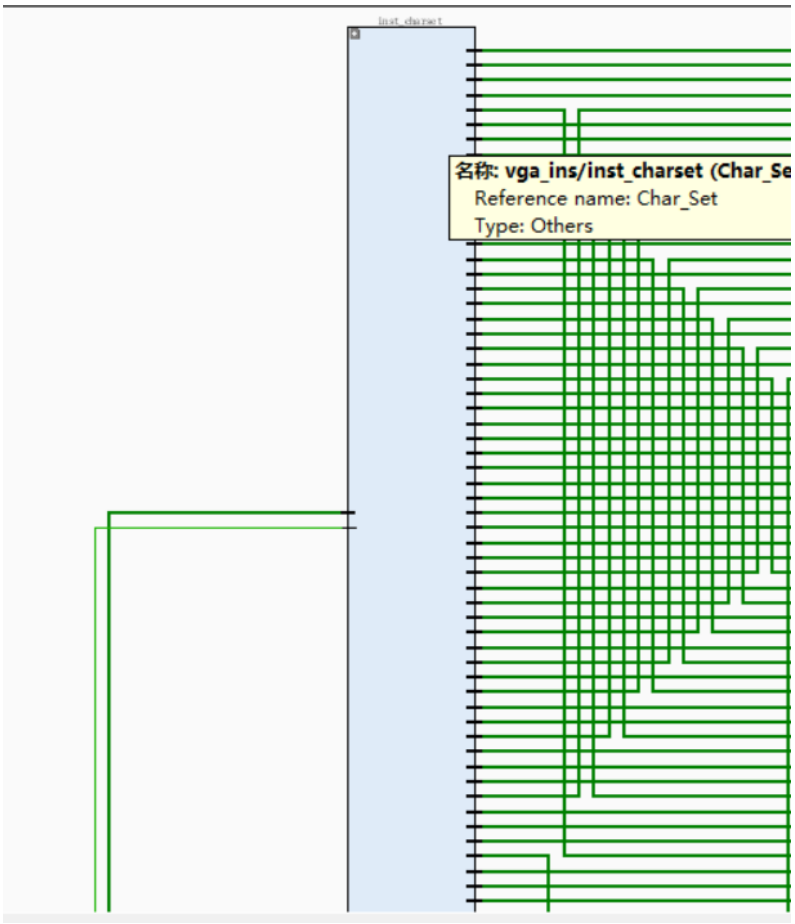
8. 字模文字模块(char_set)

1). 描述

根据当前播放的音乐，将对应歌曲信息的字模赋给输出接口，使得信息能够在 vga 显示器上显示

2). 接口定义

接口名称	接口属性	接口描述
Music_select	input	5 位音乐数据，决定当前播放的音乐
Rst_n	input	复位信号，低电平有效
Row0	Output	512 位数据，传输字模第 1 行数据
Row1	Output	512 位数据，传输字模第 2 行数据
…….	Output	512 位数据，传输字模第 n 行数据
Row63	Output	512 位数据，传输字模第 63 行数据



3). Verilog 代码

注：代码较长，仅显示部分

```
module Char_Set(
     input[4:0]music_select,
     input rst_n,
     output reg[511:0]row0,
     output reg[511:0]row1,
     output reg[511:0]row2,
     output reg[511:0]row3,
     output reg[511:0]row4,
    /*********************/
    /*
    大部分省略
    */
    /*********************/
     output reg[511:0]row62,
     output reg[511:0]row63
);
always@(*) begin
       case(music_select)
5'b00001:begin
row0<=512'h0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000;
row1<=512'h0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000;
/************************************/
大部分数据省略，共有 5 组，每组 64 行，一行中存储 512 位数据
/************************************/
end
```

4). 过程介绍

CharSet 模块主要存放的是字模数据，在最初设计时，计划使用字高*字宽为 16*16 的字模，利用 vga 显示中使用过的等比放大实现，结果发现文字显示效果并不是很好，于是选择使用字高*字宽为 64*64 的字模，每一首歌曲限定 8 个汉字信息，而这样的数据量极大，所以选择通过编写 c++程序，写出此 verilog 代码(即本模块代码均由程序生成)。

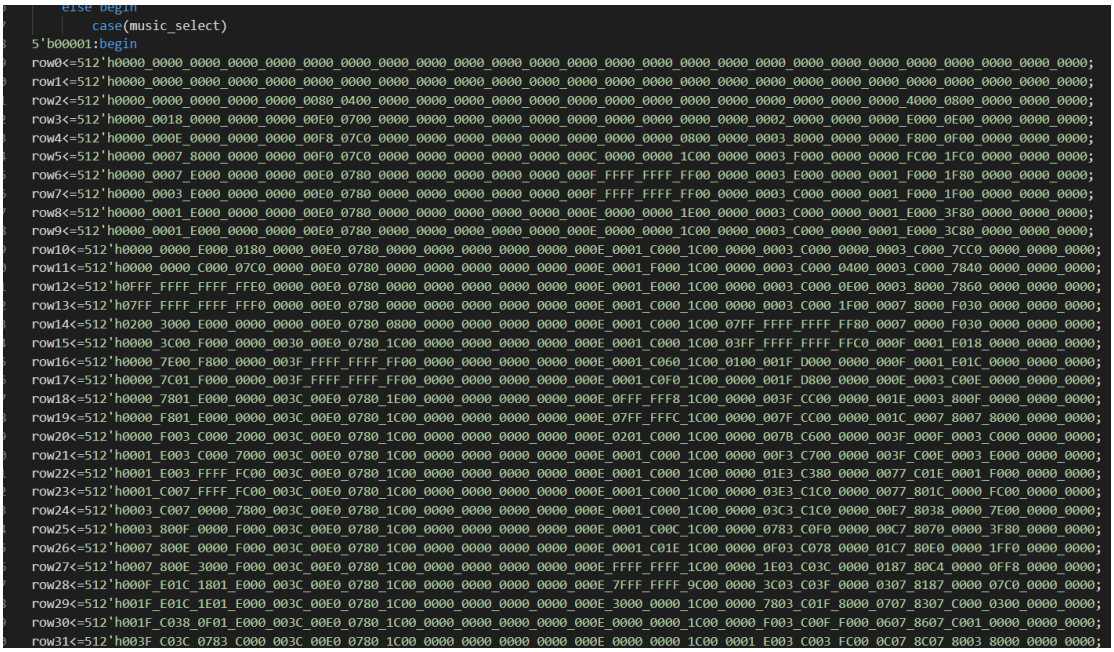
C++程序如下

```
#include<bits/stdc++.h>
using namespace std;
int main() {
    ofstream out("歌曲字模代码.v", ios::out);

    /***************模块定义部分*************/
    out << "module Char_Set(" << endl;
    out << "\t input[4:0]music_select," << endl;
    out << "\t input rst_n," << endl;
    for (int i = 0; i < 64; i++){
        out << "\t output reg[511:0]row" << i ;
        if(i!=63)
            out<<',';
        out<<endl;
    }
    out << ");" << endl;
    out << "always@(*) begin" << endl;
    out << "\tif(!rst_n)begin" << endl;
    for (int i = 0; i < 64; i++)
        out << "\trow" << i << "<=0;" << endl;
    out << "\tend" << endl;
    out << "\telse begin" << endl;

    /************case 语句部分*****************/
    out << "\t\tcase(music_select)" << endl;

    /***********case 共 5 种****************/
    int n;//定义该部分字数
    for (int m = 0; m < 5; m++) {
        ifstream in;
        if (m == 0) {
            n = 6;
            in.open("夜曲.TXT", ios::in | ios::binary);
            out << "5'b00001:begin" << endl;
        }
        else if (m == 1) {
            n = 7;
            in.open("七里香.TXT", ios::in | ios::binary);
            out << "5'b00010:begin" << endl;
        }
        else if (m == 2) {
            n = 6;
            in.open("安静.TXT", ios::in | ios::binary);
```

```
            out << "5'b00100:begin" << endl;
        }
        else if (m == 3) {
            n = 6;
            in.open("轨迹.TXT", ios::in | ios::binary);
            out << "5'b01000:begin" << endl;
        }
        else if (m == 4) {
            n = 8;
            in.open("一路向北.TXT", ios::in | ios::binary);
            out << "5'b10000:begin" << endl;
        }

        /***********数据输出部分***********/
        //最多 8 个汉字，字模中按 0xab 存储，即一次 8 位，64*64/8=512
        string buffer[8][512];

        for (int i = 0; i < n; i++) {
            for (int k = 0; k < 512; k++)
                in >> buffer[i][k];
        }
        for (int k = 0; k < 64; k++) {
            out << "row" << k << "<=512'h";
            for (int i = 0; i < 8; i++) {
                if (i >= n)
                    out << "0000";
                else {
                    for(int j=0;j<8;j++){

out<<buffer[i][k*8+j][2]<<buffer[i][k*8+j][3];
                        if((j+1)%2==0&&j!=7)
                            out<<'_';
                    }
                }
                if (i != 7)
                    out << '_';
                else
                    out << ";" << endl;
            }
        }
        out << "end" << endl;
        in.close();
    }
    out<<"endcase"<<endl;
```

```
    out<<"end"<<endl;
    out<<"end"<<endl;
    out<<"endmodule"<<endl;
    out.close();
}
```

代码截图如下



```
    else begin
        case(music_select)
5'b00001:begin
row0<=512'h0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000;
row1<=512'h0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000;
row2<=512'h0000_0000_0000_0000_0000_0080_0400_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_4000_0800_0000_0000_0000;
row3<=512'h0000_0018_0000_0000_0000_00E0_0700_0000_0000_0000_0000_0000_0000_0000_0000_0000_0002_0000_0000_0000_E000_0E00_0000_0000_0000;
row4<=512'h0000_000E_0000_0000_0000_00F8_07C0_0000_0000_0000_0000_0000_0000_0000_0800_0000_0003_8000_0000_0000_F800_0F00_0000_0000_0000;
row5<=512'h0000_0007_8000_0000_0000_00F0_07C0_0000_0000_0000_0000_0000_000C_0000_0000_1C00_0000_0003_F000_0000_0000_FC00_1FC0_0000_0000_0000;
row6<=512'h0000_0007_E000_0000_0000_00E0_0780_0000_0000_0000_0000_0000_000F_FFFF_FFFF_FF00_0000_0003_E000_0000_0001_F000_1F80_0000_0000_0000;
row7<=512'h0000_0003_E000_0000_0000_00E0_0780_0000_0000_0000_0000_0000_000F_FFFF_FFFF_FF00_0000_0003_C000_0000_0001_F000_1F00_0000_0000_0000;
row8<=512'h0000_0001_E000_0000_0000_00E0_0780_0000_0000_0000_0000_0000_000E_0000_0000_1E00_0000_0003_C000_0000_0001_E000_3F80_0000_0000_0000;
row9<=512'h0000_0001_E000_0000_0000_00E0_0780_0000_0000_0000_0000_0000_000E_0000_0000_1C00_0000_0003_C000_0000_0001_E000_3C80_0000_0000_0000;
row10<=512'h0000_0000_E000_0180_0000_00E0_0780_0000_0000_0000_0000_0000_000E_0001_C000_1C00_0000_0003_C000_0000_0003_C000_7CC0_0000_0000_0000;
row11<=512'h0000_0000_C000_07C0_0000_00E0_0780_0000_0000_0000_0000_0000_000E_0001_F000_1C00_0000_0003_C000_0400_0003_C000_7840_0000_0000_0000;
row12<=512'h0FFF_FFFF_FFFF_FFE0_0000_00E0_0780_0000_0000_0000_0000_0000_000E_0001_E000_1C00_0000_0003_C000_0E00_0003_8000_7860_0000_0000_0000;
row13<=512'h07FF_FFFF_FFFF_FFF0_0000_00E0_0780_0000_0000_0000_0000_0000_000E_0001_C000_1C00_0000_0003_C000_1F00_0007_8000_F030_0000_0000_0000;
row14<=512'h0200_3000_E000_0000_0000_00E0_0780_0800_0000_0000_0000_0000_000E_0001_C000_1C00_07FF_FFFF_FFFF_FF80_0007_0000_F030_0000_0000_0000;
row15<=512'h0000_3C00_F000_0000_0030_00E0_0780_1C00_0000_0000_0000_0000_000E_0001_C000_1C00_03FF_FFFF_FFFF_FFC0_000F_0001_E018_0000_0000_0000;
row16<=512'h0000_7E00_F800_0000_003F_FFFF_FFFF_FF00_0000_0000_0000_0000_000E_0001_C060_1C00_0100_001F_D000_0000_000F_0001_E01C_0000_0000_0000;
row17<=512'h0000_7C01_F000_0000_003F_FFFF_FFFF_FF00_0000_0000_0000_0000_000E_0001_C0F0_1C00_0000_001F_D800_0000_000E_0003_C00E_0000_0000_0000;
row18<=512'h0000_7801_E000_0000_003C_00E0_0780_1E00_0000_0000_0000_0000_000E_0FFF_FFF8_1C00_0000_003F_CC00_0000_001E_0003_800F_0000_0000_0000;
row19<=512'h0000_F801_E000_0000_003C_00E0_0780_1C00_0000_0000_0000_0000_000E_07FF_FFFC_1C00_0000_007F_CC00_0000_001C_0007_8007_8000_0000_0000;
row20<=512'h0000_F003_C000_2000_003C_00E0_0780_1C00_0000_0000_0000_0000_000E_0201_C000_1C00_0000_007B_C600_0000_003F_000F_0003_C000_0000_0000;
row21<=512'h0001_E003_C000_7000_003C_00E0_0780_1C00_0000_0000_0000_0000_000E_0001_C000_1C00_0000_00F3_C700_0000_003F_C00E_0003_E000_0000_0000;
row22<=512'h0001_E003_FFFF_FC00_003C_00E0_0780_1C00_0000_0000_0000_0000_000E_0001_C000_1C00_0000_01E3_C380_0000_0077_C01E_0001_F000_0000_0000;
row23<=512'h0001_C007_FFFF_FC00_003C_00E0_0780_1C00_0000_0000_0000_0000_000E_0001_C000_1C00_0000_03E3_C1C0_0000_0077_801C_0000_FC00_0000_0000;
row24<=512'h0003_C007_0000_7800_003C_00E0_0780_1C00_0000_0000_0000_0000_000E_0001_C000_1C00_0000_03C3_C1C0_0000_00E7_8038_0000_7E00_0000_0000;
row25<=512'h0003_800F_0000_F000_003C_00E0_0780_1C00_0000_0000_0000_0000_000E_0001_C00C_1C00_0000_0783_C0F0_0000_00C7_8070_0000_3F80_0000_0000;
row26<=512'h0007_800E_0000_F000_003C_00E0_0780_1C00_0000_0000_0000_0000_000E_0001_C01E_1C00_0000_0F03_C078_0000_01C7_80E0_0000_1FF0_0000_0000;
row27<=512'h0007_800E_3000_F000_003C_00E0_0780_1C00_0000_0000_0000_0000_000E_FFFF_FFFF_1C00_0000_1E03_C03C_0000_0187_80C4_0000_0FF8_0000_0000;
row28<=512'h000F_E01C_1801_E000_003C_00E0_0780_1C00_0000_0000_0000_0000_000E_7FFF_FFFF_9C00_0000_3C03_C03F_0000_0307_8187_0000_07C0_0000_0000;
row29<=512'h001F_E01C_1E01_E000_003C_00E0_0780_1C00_0000_0000_0000_0000_000E_3000_0000_1C00_0000_7803_C01F_8000_0707_8307_C000_0300_0000_0000;
row30<=512'h001F_C038_0F01_E000_003C_00E0_0780_1C00_0000_0000_0000_0000_000E_0000_0000_1C00_0000_F003_C00F_F000_0607_8607_C001_0000_0000_0000;
row31<=512'h003F_C03C_0783_C000_003C_00E0_0780_1C00_0000_0000_0000_0000_000E_0000_0000_1C00_0001_E003_C003_FC00_0C07_8C07_8003_8000_0000_0000;
```

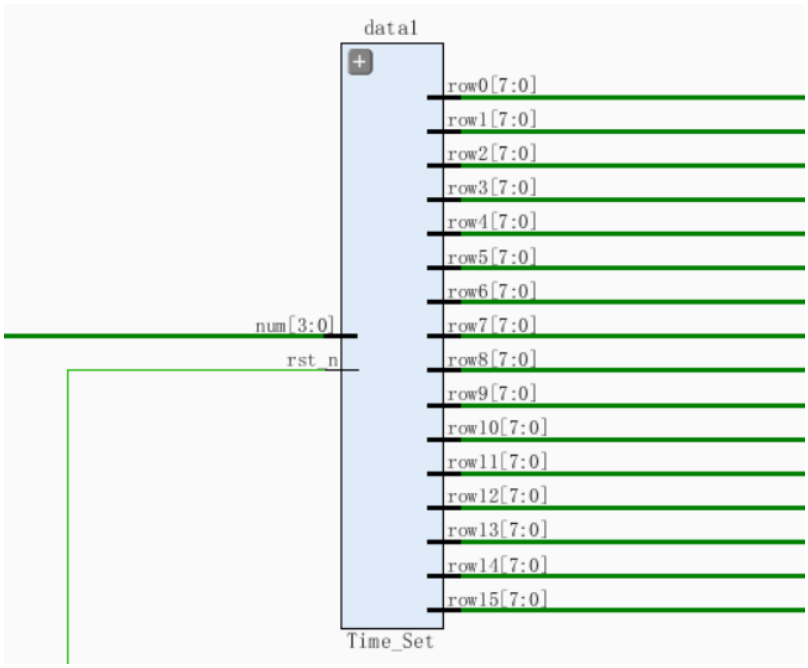
9. 字模数字模块(time_set)

1). 描述

根据输入的值，返回其对于阿拉伯数字的字模数据

2). 接口定义

接口名称	接口属性	接口描述
num	input	4 位数据，代表需要显示的数字
Rst_n	input	复位信号，低电平有效
Row0	Output	8 位数据，传输字模第 1 行数据
Row1	Output	8 位数据，传输字模第 2 行数据
…….	Output	8 位数据，传输字模第 n 行数据
Row15	Output	8 位数据，传输字模第 15 行数据



3). Verilog 代码

```
module Time_Set(
     input[3:0]num,
     input rst_n,
     output reg[7:0]row0,
     output reg[7:0]row1,
    /*大部分省略*/
     output reg[7:0]row15
);
always@(*) begin
    if(!rst_n)begin
    row0<=0;    row1<=0;    row2<=0;    row3<=0;
    row4<=0;    row5<=0;    row6<=0;    row7<=0;
    row8<=0;    row9<=0;    row10<=0;   row11<=0;
    row12<=0;   row13<=0;   row14<=0;   row15<=0;
    end
    else begin
        case(num)
        4'b0000:begin
        row0<=8'h00;        row1<=8'h00;        row2<=8'h00;
row3<=8'h18;
        row4<=8'h24;        row5<=8'h42;        row6<=8'h42;
row7<=8'h42;
        row8<=8'h42;        row9<=8'h42;        row10<=8'h42;
row11<=8'h42;
```

```
        row12<=8'h24;       row13<=8'h18;       row14<=8'h00;
row15<=8'h00;
        end
       /*大部分省略*/
        endcase
    end
end
endmodule
```

4). 过程介绍

根据传入值，通过一个 case 语句，分情况为端口赋值，特殊情况：当值为 4’b1010 时，代表为“:”赋值。由于显示的是数字与符号，精度不需要太高，所以选择字宽*字高为 16*16 的字模即可。由于字模数据较多，同样通过 c++程序实现.v 文件的编写。代码如下图所示。

```
#include<bits/stdc++.h>
using namespace std;
int main() {
    ofstream out("计时字模代码.v", ios::out);
    /*****/
    out << "module Time_Set(" << endl;
    out << "\t input[3:0]num," << endl;
    out << "\t input rst_n," << endl;
    for (int i = 0; i < 16; i++){
        out << "\t output reg[7:0]row" << i ;
        if(i!=15)
            out<<',';
        out<<endl;
    }
    out << ");" << endl;
    out << "always@(*) begin" << endl;
    out << "\tif(!rst_n)begin" << endl;
    for (int i = 0; i <16; i++){
        out << "\trow" << i << "<=0;" ;
        if((i+1)%4==0)
            out<<endl;

    }
    out << "\tend" << endl;
    out << "\telse begin" << endl;
```

```
    out << "\t\tcase(num)" << endl;
    /*****/
    for (int n = 0; n < 11; n++) {
        ifstream in;
        string addr="./16/";
        if(n==10)
            addr+="符号.TXT";
        else
            (addr+=char('0'+n))+=".TXT";

        in.open(addr.c_str(),ios::in);
        if(!in.is_open())
            system("pause");
        string buffer[16];
        for(int k=0;k<16;k++)
            in>>buffer[k];

        if(n==0)
            out<<"\t\t4'b0000:begin"<<endl;
        else if(n==1)
            out<<"\t\t4'b0001:begin"<<endl;
        else if(n==2)
            out<<"\t\t4'b0010:begin"<<endl;
        else if(n==3)
            out<<"\t\t4'b0011:begin"<<endl;
        else if(n==4)
            out<<"\t\t4'b0100:begin"<<endl;
        else if(n==5)
            out<<"\t\t4'b0101:begin"<<endl;
        else if(n==6)
            out<<"\t\t4'b0110:begin"<<endl;
        else if(n==7)
            out<<"\t\t4'b0111:begin"<<endl;
        else if(n==8)
            out<<"\t\t4'b1000:begin"<<endl;
        else if(n==9)
            out<<"\t\t4'b1001:begin"<<endl;
        else
            out<<"\t\t4'b1010:begin"<<endl;

        for (int k = 0; k < 16; k++) {
            out <<"\t\trow" << k << "<=8'h";
            out << buffer[k][2] << buffer[k][3];
            out << ";";
```

```
            if((k+1)%4==0)
                out<<endl;
        }
        out<< "\t\tend" << endl;
        in.close();
    }
    out<<"\t\tendcase"<<endl;
    out<<"\tend"<<endl;
    out<<"end"<<endl;
    out<<"endmodule"<<endl;
    out.close();
}
```

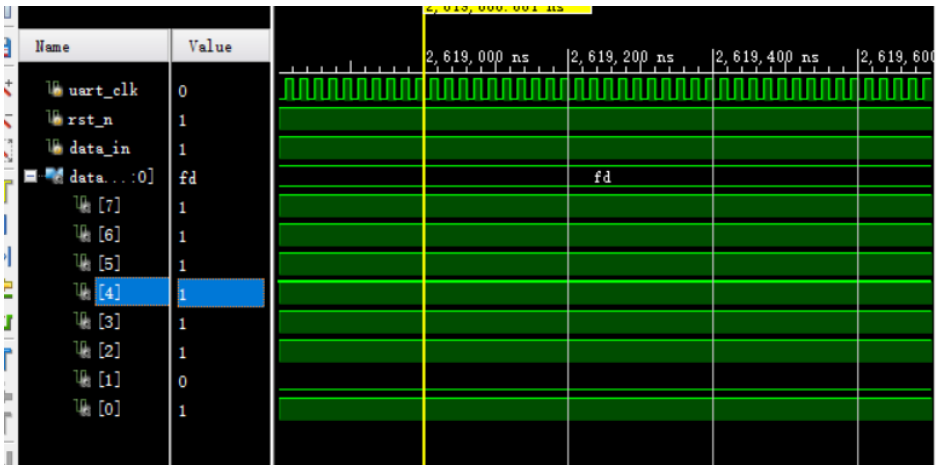
五、 测试模块建模

1. 蓝牙模块测试

测试代码：

```
module tb_UART;
    reg uart_clk;
    reg rst_n;
    reg data_in;
    wire [7:0]data_out;

    UART tb(
        .uart_clk(uart_clk),
        .rst_n(rst_n),
        .data_in(data_in),
        .data_out(data_out)
    );
    always #10 uart_clk=~uart_clk;
    initial begin
        uart_clk=0;
        rst_n=0;
        data_in=0;
        #100
        rst_n=1;
        data_in=1;      //空闲
        #104166
        data_in=0;      //开始传输
        #104166
        data_in=1;
        #104166
        data_in=1;
        #104166
        data_in=1;
```

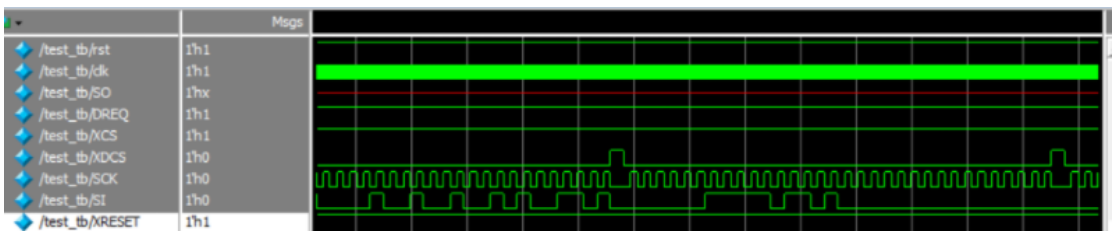
```
        #104166
        data_in=0;
        #104166
        data_in=1;
        #104166
        data_in=1;
        #104166
        data_in=1;
        #104166
        data_in=1;

        //receive 11110111
    end
endmodule
```



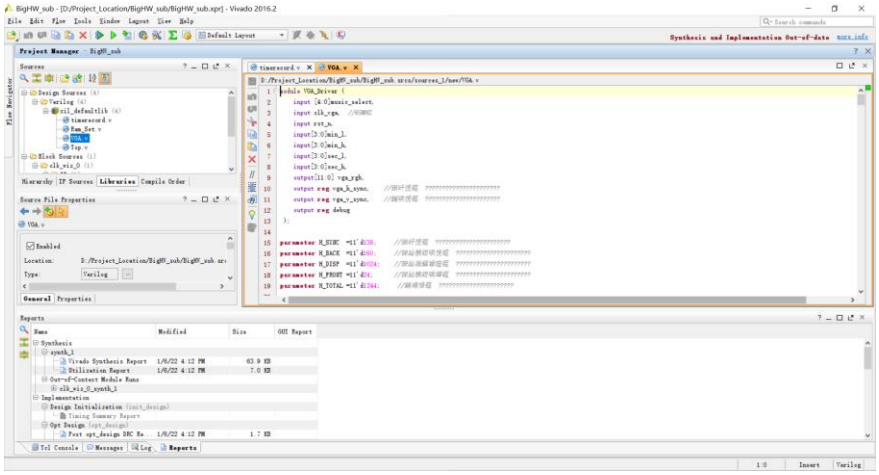
2). Mp3 模块测试建模

```
`timescale 1ns / 1ps

module test_mp3;
    reg [7:0]vol;
    reg [4:0]music_select;
    reg clk;
    reg DREQ;
    reg rst;
    wire SO;
    wire XDCS;    //data
    wire XCS;    //cmd
    wire SI;
    wire SCLK;
```

```
    wire XRESET;

    //add
    wire DEBUG;
    mp3_player(
        .vol(vol),
        .music_select(music_select),
        .CLK(clk),
        .SO(SO),
        .DREQ(DREQ),
        .RST(rst),
        .XDCS(XDCS),
        .XCS(XCS),
        .SI(SI),
        .SCLK(SCLK),
        .XRESET(XRESET)
    );
    always #10 CLK=~CLK;
    always #10 DREQ=~DREQ;
    initial begin
        RST=0;
        CLK=1;
        DREQ=0;
        music_select=5'b00010;
        vol=8'h60;
        #10
        RST=1;
        #10
        DREQ=1;
        #70000
        RST=0;

    end

endmodule
```

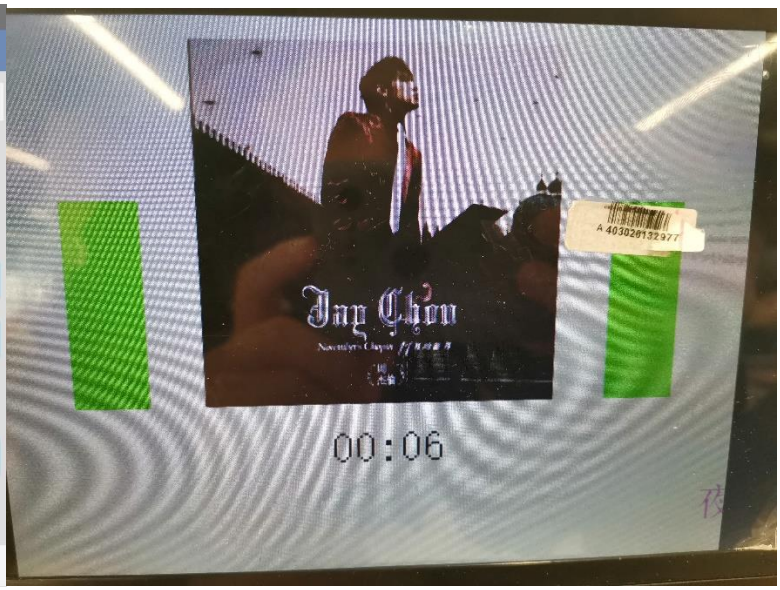


Vga 显示由于其特殊性，未进行 testbench 测试，而是通过另一个项目中进行单功能测试实现，逐次实现不同的显示功能并下板测试，最终项目截图如下：



六、 结果与总结

1.下板图片



2.总结

通过本次大作业，我对各类传输协议如 uart 协议、spi 协议有了初步的认识，自己通过 verilog 基本实现了这些协议。最终，能够将 vs1003 模块、蓝牙模块、vga 显示器、开发板联系起来，实现了这个 MP3 音乐播放器。整个播放器在下板后能够正常工作，能够实现既定的功能。在编写代码的过程中，我对于 verilog 的语言有了更深刻的认识，也终于摆脱了面向软件编程的习惯，更加注重时序的重要性，也对 verilog 的模块化开发有了切身的体验。与之前的小实验相比，显然这样的大作业能够极大地提高 verilog 代码的编写能力。

当然，整个项目仍然能够进一步开发，例如，若将音频、图片通过外设存储（例如 SD 卡），则可以存储更高质量的音乐，且可以通过 MP3 文件获取音频的音调高低等音频信息，从而绘制出音频波谱，这个功能在调试时曾经尝试过，但是由于音频文件 MIDI 文件，只能获取音频的音量，所以以失败告终。此外，在获得音频的基础上，也可以制作音乐游戏。

七、 心得体会

1. 课程收获

本学期开设的数字逻辑课程，让我初步接触了计算机中的硬件部分，在课程中，我学习了布尔代数、组合逻辑、时序逻辑等理论知识，了解各类门电路、寄存器、触发器的原理，为今后继续学习计算机底层相关的知识打下了坚实的基础。在此之后，还学习了各种设计理念，例如自顶向下的设计方法。这些理念让我能够在设计时，头脑更加清晰，设计出更加简单的电路。

当然，除了理论知识的学习，还需要亲身实践。在本学期的实验课上，我第一次了解并开始使用 verilog 语言进行编程。在一次又一次的小实验中，熟悉 verilog 语言，从最初理解不了 assign 和 always 块，到现如今能够写出大作业，这样进步让我感到喜悦。此外，实验课的实验内容基本都是一些底层模块的设计，这些实验也让我对计算机、处理器有切实的体会。

最终的大作业也让我明白，自己当前掌握的知识是远远不够的。通过查阅资料，我了解许多通信协议。尽管在使用 verilog 语言实现的过程中也遭遇了许多的 bug，但最终还是能够修复 bug，而最后成功下板后的喜悦也是之前任何一个实验难以带来的。整个过程中让我积累了 verilog 编程的许多经验，也让我更加明白学习需要钻研的精神。

2. 课程建议

1) 希望老师在小实验中夹杂一些中等规模的实验，为大作业打下基础。若仅凭

借之前小作业积累下的编程经验，在完成大作业时会有较大落差，也就是俗称的“造火箭”

2) 希望老师上课时能够继续与同学们进行互动，同时可以说一些方法、技术的实际应用，这样可以让我们对这类方法、技术有更深刻的认识。

3) 希望老师在平时课堂中就可以对某一些基础的通信协议进行讲解，因为在大作业的设计过程，几乎不可避免的就是一些基础通信协议的实现，若有了前期的教学，那么在期末时上手大作业会更加简单。

3. 数字芯片设计的认识与体会

自从中美贸易战开始以来，作为被美国卡脖子的技术，芯片成为了人们的焦点。在新时代下，中国的芯片行业在面临诸多挑战的同时，也拥有着此前从未拥有国的发展良机。

一方面，5G 时代下对芯片的性能提出了更高的要求，这使得国内现有的芯片很难满足高端设备的需求。同时代下，全球最先进的工艺制程达到了 5nm，而现如今，国内能够量产的最先进制程为 14nm，还停留在 5 年前的水平，与最新工艺有着 3 代的差距。而光刻机这样的核心设备，与国际最先进的水平差距也较大。在这样基础下，基于国产工艺的高端芯片的研发几乎无法实现，使得国内芯片企业在市场上同国际巨头的竞争几乎完全处于下风，只能够主攻其它行业，而这些行业较低的利润又进而使得企业无法拿出大量的资金投入到研发。

但另一方面，华为麒麟芯片的成功，也说明移动端处理器设计方面，我国与世界先进水平差距已经不大，麒麟芯片积累下来的经验，让我们明白，当硬件得到足够的支持，我们也能够设计出性能优越的处理器。这无疑为芯片行业打了一针强心剂。虽然落后，但我们仍有底气去追赶，去超越。随着国家大力发展芯片行业，对于行业投入越来越多的资金，我们可以期待，在不久的未来，通过芯片行业和国家的不断努力，能够解决光刻机这一问题，在那时，追赶甚至超越国际的顶尖水平便不在是空中楼阁。

除此以外，芯片的不断发展使得摩尔定律越来越难被遵循，芯片的性能的增幅减缓。随着制程的不断升级，同样的体积下，芯片中集成的晶体管数量已达到百亿级别，芯片已经快触碰到硅的物理极限。在如此小的空间集成如此多的电路，其产生的热量越来越难以处理，这里可以参考手机处理器制造商高通公司近两年推出的旗舰芯片，其发热量对于手机这类移动平台而言几乎无法承受，使得实际的体验的

大打折扣，从另一个角度上来说，这也为我国追赶先国际先进水平提供了时间。

而作为大学生，我们也应该努力学习先进知识，不断提高自己的专业水平，与此同时要牢记自己身上的使命感，投身于国家关键技术的研究，为国家复兴提供自己的力量。