



**计算机组成原理(汇编与接口)课程设计**

团队实验报告

|  |  |
| --- | --- |
| 学 院 | 计算机学院 |
| 专 业 | 计算机科学与技术 |
| 指导老师 |  |
| 组 长 | 周振翔 |
| 组 员 | 周振翔 张亦驰 穆新宇 董若扬 |
| 组长联系方式 |  |

二O二三 年 九 月

目 录

备注：团队与个人报告均使用此模板，红色字体供参考，请根据实际情况填写，黑色框架部分请勿修改。个人报告不需要分工介绍。

# 项目简述

我们小组的汇编项目实现了RISC-V架构五级流水线(IF, ID, IE, MA, WB)CPU。在此基础上我们实现了对蜂鸣器，拨片，LED，按键,，VGA外设的控制。并在我们小组实现的五级流水线CPU上运行了汇编代码编译成的机器码, 实现了通过拨片控制蜂鸣器播放不同的音乐, 通过LED指示正在播放的音乐的曲调，通过VGA输出图像信息，通过按键控制图像的位置。完成了通过自己实现的CPU控制外设的目标。

# 组员分工

小组分工见表2.1 小组分工。

|  |  |
| --- | --- |
| **组员** | **分工任务** |
| 周振翔 | * 掌握进度，组合汇编和计组任务，设计流水线CPU架构，协同DEBUG，综合上板 |
| 张亦驰 | * 协同设计流水线CPU，实现CPU模块，仿真DEBUG |
| 董若扬 | * 编写VGA驱动程序。编写按键信号控制逻辑。在此基础上编写测试汇编程序。 |
| 穆新宇 | * 编写蜂鸣器的接口Verilog程序 |

表2.1 小组分工

# 设计目的

集成接口控制器和处理器设计结果，形成自定义的计算机系统。设计并完成VGA控制器，蜂鸣器控制器，按键，拨片，LED灯控制器。成功在精工板上实现。测试后并形成IP核。

# 设计环境

|  |  |
| --- | --- |
| 操作系统 | Windows 11 |
| 编程语言 | Verilog asm |
| EDA工具 | Vivado 2020.2 |
| 汇编语言 | Riscv-32I |
| 汇编程序编辑器 | RARS 1.6 |

# 设计原理及内容

## CPU 的组织结构

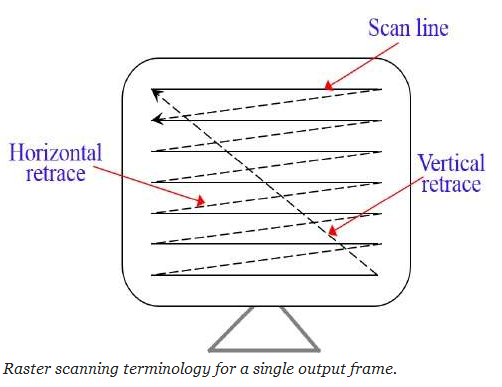
图示

描述已自动生成

CPU 对外部设备的控制是通过写入（SW）和读取（LW）外设中的数据存储器来实现的。将外部设备和 CPU 独立开来，将时钟信号 clk 和复位信号 rst 通过一个公共的顶层模块同时引线连接在 CPU 和外部设备上面。

## VGA显示的实现

VGA 显示器扫描方式从屏幕左上角一点开始，从左向右逐点扫描，每扫描完一行，电子束回到屏幕的左边下一行的起始位置，在这期间，CRT 对电子束进行消隐，每行结束时，用行同步信号进行同步；当扫描完所有的行，形成一帧，用场同步信号进行场同步，并使扫描回到屏幕左上方，同时进行场消隐，开始下一帧。完成一行扫描的时间称为水平扫描时间，其倒数称为行频率；完成一帧（整屏）扫描的时间称为垂直扫描时间，其倒数称为场频率，即屏幕的刷新频率，常见的有 60Hz，75Hz 等等，但标准的 VGA 显示的场频 60Hz。其扫描示意图如下图所示：



VGA 包含行时序与场时序两个部分，行时序包含：

Horizontal Sync Pulse

Horizontal Front Porch

Horizontal Front Active Video

Horizontal Back Porch

这四个参数，其时序图如下：

表格

描述已自动生成

相应的，场时序的参数类似，场时序图如下：

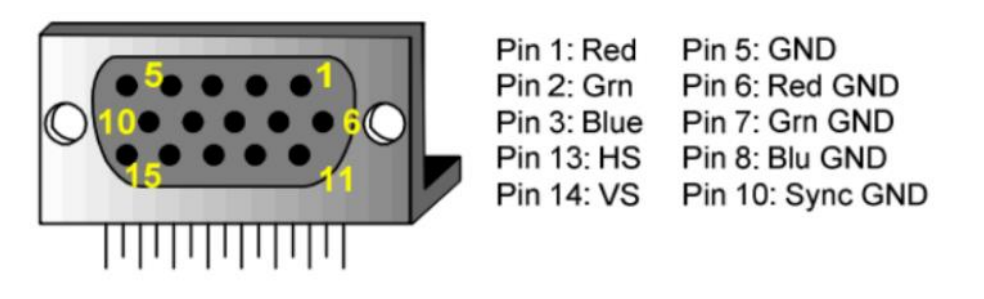
表格

描述已自动生成

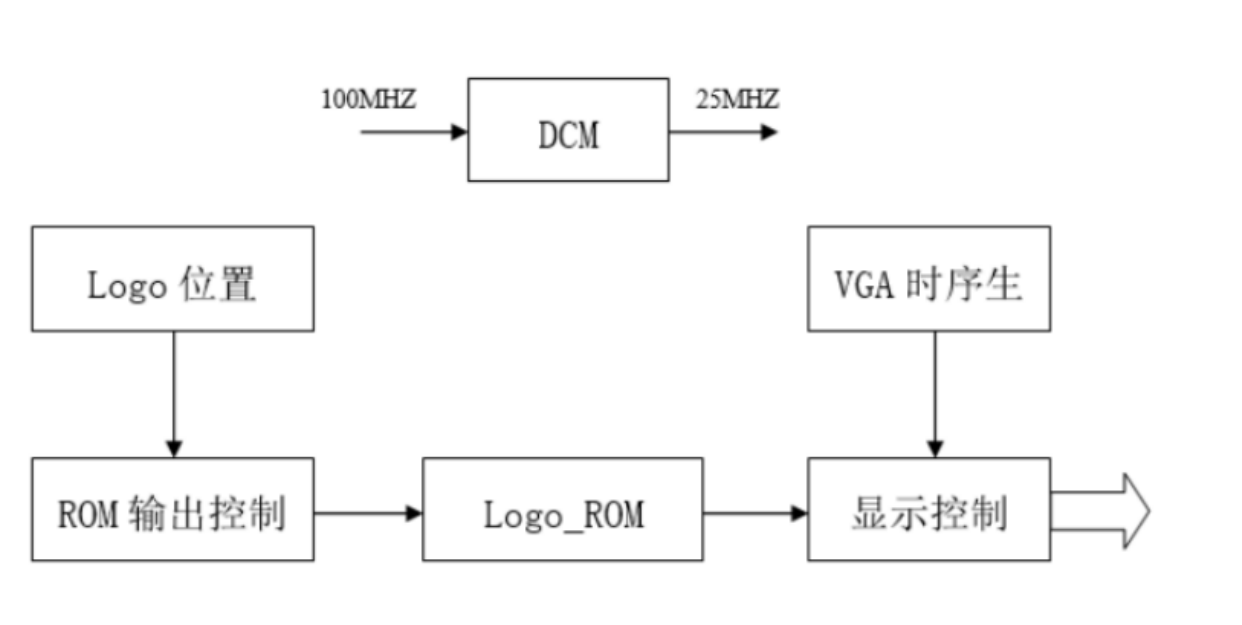
表格

描述已自动生成

## VGA硬件实现

VGA接口是显示器上应用最为广泛的接口类型，它是一种D型接口，上面共有15个针孔，信号连接如下图所示：

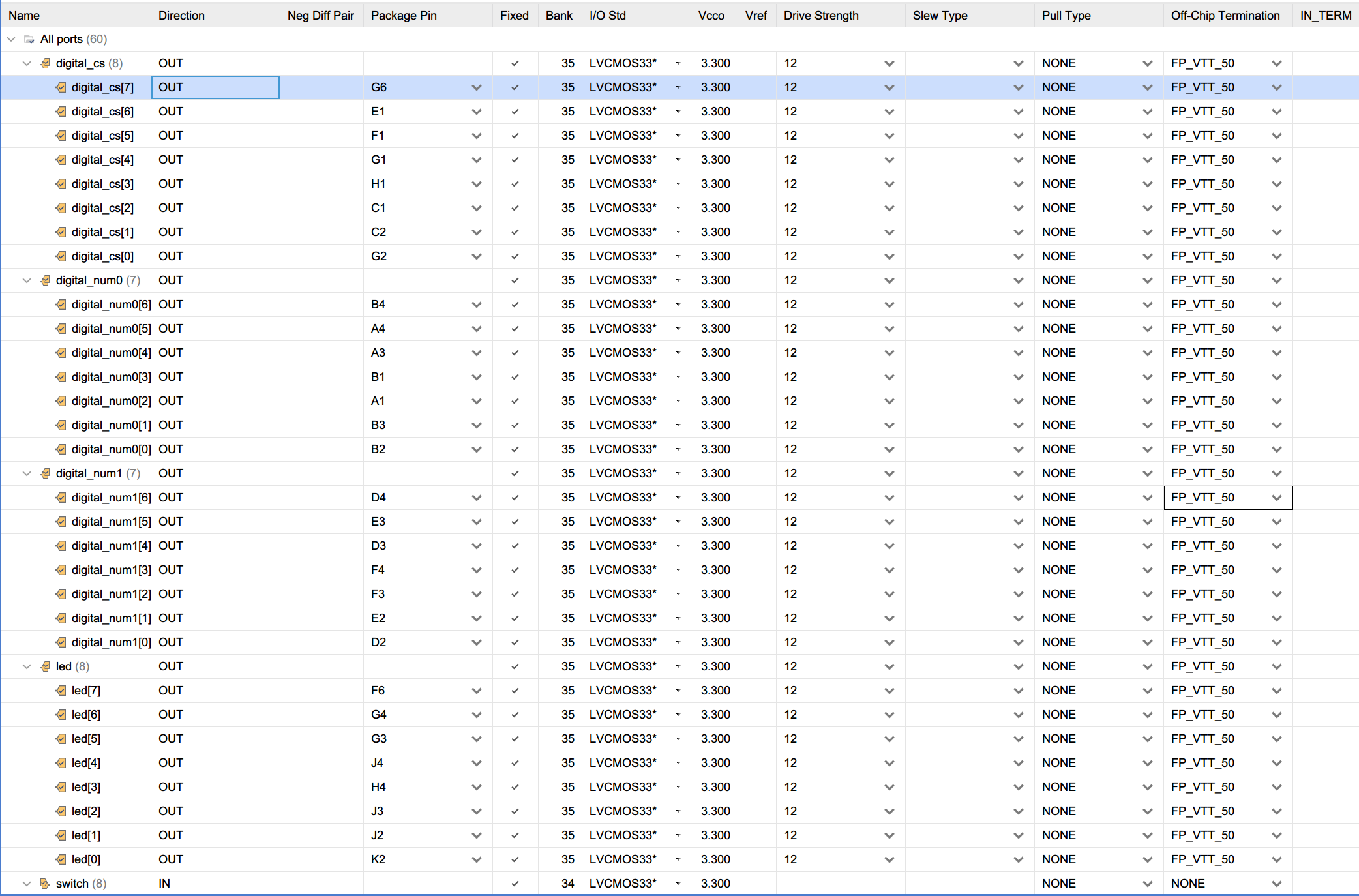
通用VGA显示卡系统主要由控制电路、显示缓存区和程序BIOS程序三个部分组成。控制电路主要完成时序发生、显示缓冲区数据操作、主时钟选择和D/A转换等功能；显示缓冲区提供显示数据缓存空间；视频BIOS作为控制程序固化在显示卡ROM中。本实验的系统架构如下图所示：



## 引脚约束

选择 “LVCOMS33”，将所有信号的电平标准设置 3.3V

分配各个信号在 FPGA 芯片上引脚的位置，各信号的具体位置可查看板卡的原理图。



电脑屏幕截图

中度可信度描述已自动生成

# 设计与实现

## 总体设计

三种思路，分别是：

1. 纯硬件

即完全通过verilog文件完成图像信号的处理和生成VGA信号.

优点: 容易实现复杂的运动.

缺点:

没有使用CPU.

图像信号唯一确定, 不可控制更改.

难度低.

1. CPU处理图像信号

由CPU运行编写的汇编代码编译成的机器码完成图像位置的处理, 再由VGA驱动输出信号.

优点: 使用了CPU, 能对CPU进行测试.

缺点:

1. 图像信号仍然唯一确定, 不可人为控制.
2. 用汇编实现复杂运动编写难度大, 且对CPU要求高.
3. CPU处理控制信号

由CPU运行汇编代码编译成的机器码完成对外部设备的轮询, 一旦检测到外设的输入, 就根据输入的控制信息对图像信号进行处理, 再由VGA驱动输出信号.

优点: 完成了使用RISC-V架构CPU对外部设备的控制.

缺点: 调试需要考虑各种环节: CPU设计是否正确? 汇编代码逻辑是否正确? 对外设的访问是否正确? 图像位置信息是否能正确存取……每次调试都要从生成比特流开始,费时费力.

## VGA

定义该模块从顶层模块接收时序信号, 生成VGA信号

定义水平前廊, 水平活动区域, 水平后廊, 垂直前廊, 垂直活动区域, 垂直后廊等参数.

// 定义模块 vga\_640x480

module vga\_640x480(pclk, reset, hsync, vsync, valid, h\_cnt, v\_cnt);

   input        pclk;                // 输入端口：像素时钟

   input        reset;               // 输入端口：复位信号

   output       hsync;               // 输出端口：水平同步信号

   output       vsync;               // 输出端口：垂直同步信号

   output       valid;               // 输出端口：有效信号

   output [9:0] h\_cnt;               // 输出端口：水平计数器

   output [9:0] v\_cnt;               // 输出端口：垂直计数器

   parameter    h\_frontporch = 96;   // 参数：水平前廊

   parameter    h\_active = 144;      // 参数：水平活动区域

   parameter    h\_backporch = 784;   // 参数：水平后廊

   parameter    h\_total = 800;       // 参数：水平总计数

   parameter    v\_frontporch = 2;    // 参数：垂直前廊

   parameter    v\_active = 35;       // 参数：垂直活动区域

   parameter    v\_backporch = 515;   // 参数：垂直后廊

   parameter    v\_total = 525;       // 参数：垂直总计数

   reg [9:0]    x\_cnt;               // 寄存器：水平计数器

   reg [9:0]    y\_cnt;               // 寄存器：垂直计数器

   wire         h\_valid;             // 连线：水平有效性信号

   wire         v\_valid;             // 连线：垂直有效性信号

使用时钟和复位信号控制的组合逻辑块，用于递增水平计数器

always @(posedge reset or posedge pclk)

      if (reset == 1'b1)

         x\_cnt <= 1;

      else

      begin

         if (x\_cnt == h\_total)

            x\_cnt <= 1;

         else

            x\_cnt <= x\_cnt + 1;

      end

使用时钟控制的组合逻辑块，用于递增垂直计数器

always @(posedge pclk)

      if (reset == 1'b1)

         y\_cnt <= 1;

      else

      begin

         if (y\_cnt == v\_total & x\_cnt == h\_total)

            y\_cnt <= 1;

         else if (x\_cnt == h\_total)

            y\_cnt <= y\_cnt + 1;

      end

使用条件运算符生成水平同步信号

   assign hsync = ((x\_cnt > h\_frontporch)) ? 1'b1 : 1'b0;

使用条件运算符生成垂直同步信号

   assign vsync = ((y\_cnt > v\_frontporch)) ? 1'b1 : 1'b0;

使用条件运算符生成水平有效性信号

   assign h\_valid = ((x\_cnt > h\_active) & (x\_cnt <= h\_backporch)) ? 1'b1 : 1'b0;

使用条件运算符生成垂直有效性信号

   assign v\_valid = ((y\_cnt > v\_active) & (y\_cnt <= v\_backporch)) ? 1'b1 : 1'b0;

使用条件运算符生成有效信号

有效信号是水平有效性信号和垂直有效性信号的逻辑与

   assign valid = ((h\_valid == 1'b1) & (v\_valid == 1'b1)) ? 1'b1 : 1'b0;

使用条件运算符生成水平计数器的值

如果水平有效性信号为 1，那么水平计数器的值为 x\_cnt - 144, 否则为 0.

这段代码的目的是根据 h\_valid 信号的状态来决定水平计数器的值。

如果活动区域内（h\_valid 为真），水平计数器将反映实际的计数值减去 144；

如果在活动区域之外（h\_valid 不为真），水平计数器将被置零。

这样可以方便地对活动区域内和外的像素进行不同的处理或控制。

   assign h\_cnt = ((h\_valid == 1'b1)) ? x\_cnt - 144 : {10{1'b0}};

使用条件运算符生成垂直计数器的值

如果垂直有效性信号为 1，那么垂直计数器的值为 y\_cnt - 35, 否则为 0.

   assign v\_cnt = ((v\_valid == 1'b1)) ? y\_cnt - 35 : {10{1'b0}};

判断当前输出区域是否是有效像素范围, 如果不是则输出纯白的信号.

判断当前输出区域是否是图像区域范围, 如果不是则输出纯黑信号.

如果是图片区域, 则从内存读取数据, 输出相应的信号.

//判断是否在logo区域内

assign logo\_area = ((v\_cnt >= logo\_y) & (v\_cnt <= logo\_y + logo\_hight - 1) & (h\_cnt >= logo\_x) & (h\_cnt <= logo\_x + logo\_length - 1)) ? 1'b1 : 1'b0;

always @(posedge pclk) begin: logo\_display

    if (rst\_n == 1'b1)

        vga\_data <= 12'b000000000000;

    else begin

        if (valid == 1'b1) begin

            if (logo\_area == 1'b1) begin

                rom\_addr <= rom\_addr + 14'b00000000000001;

                vga\_data <= douta;

            end

            else begin

                rom\_addr <= rom\_addr;

                vga\_data <= 12'b000000000000;

            end

        end

        else begin

            vga\_data <= 12'b111111111111;

            if (v\_cnt == 0)

                rom\_addr <= 14'b00000000000000;

        end

    end

end

## 按键

在confreg中对按键的行为进行定义

`define RST\_ENABLE 1'b1

`define DIGITAL\_NUM\_ADDR    16'h8000 // 0xbfaf\_8000

`define SWITCH\_ADDR         16'h8004 // 0xbfaf\_8004

`define LED\_ADDR            16'h8008 // 0xbfaf\_8008

`define MID\_BTN\_KEY\_ADDR    16'h800c // 0xbfaf\_800c

`define LEFT\_BTN\_KEY\_ADDR   16'h8010 // 0xbfaf\_8010

`define RIGHT\_BTN\_KEY\_ADDR  16'h8014 // 0xbfaf\_8014

`define UP\_BTN\_KEY\_ADDR     16'h8018 // 0xbfaf\_8018

`define DOWN\_BTN\_KEY\_ADDR   16'h801c // 0xbfaf\_801c

`define LOGO\_X\_ADDR         16'h8020 // 0xbfaf\_8020

`define LOGO\_Y\_ADDR         16'h8024 // 0xbfaf\_8024

以中间按钮为例，展示按钮的控制逻辑

/\*\*\*\*\*\*\*\*\*\*\* mid\_btn\_key \*\*\*\*\*\*\*\*\*\*\*/

reg mid\_btn\_key\_r;

assign mid\_btn\_key\_v = {31'd0, mid\_btn\_key\_r};

// eliminate jitter

reg         mid\_btn\_key\_flag;

reg [19:0]  mid\_btn\_key\_count;

wire mid\_btn\_key\_start = !mid\_btn\_key\_r && mid\_btn\_key;

wire mid\_btn\_key\_end   = mid\_btn\_key\_r && !mid\_btn\_key;

wire mid\_btn\_key\_sample= mid\_btn\_key\_count[19];

always @ (posedge clk) begin

    if (rst == `RST\_ENABLE) begin

        mid\_btn\_key\_flag <= 1'b0;

    end

    else if (mid\_btn\_key\_sample) begin

        mid\_btn\_key\_flag <= 1'b0;

    end

    else if (mid\_btn\_key\_start || mid\_btn\_key\_end) begin

        mid\_btn\_key\_flag <= 1'b1;

    end

    if (rst == `RST\_ENABLE || !mid\_btn\_key\_flag) begin

        mid\_btn\_key\_count <= 20'b0;

    end

    else begin

        mid\_btn\_key\_count <= mid\_btn\_key\_count + 1'b1;

    end

    if (rst == `RST\_ENABLE) begin

        mid\_btn\_key\_r <= 1'b0;

    end

    else if (mid\_btn\_key\_sample) begin

        mid\_btn\_key\_r <= mid\_btn\_key;

    end

end

这段代码中实现了对按键的行为, 状态定义. 标识了按键在按下和松开的行为特征, 并实现了复位操作和赋值操作, 还对按键进行了消抖处理. 功能非常完备.

## Confreg设计

首先, 我们定义了存放图像的坐标信息的地址

`define LOGO\_X\_ADDR         16'h8020 // 0xbfaf\_8020

`define LOGO\_Y\_ADDR         16'h8024 // 0xbfaf\_8024

我们让x\_logo\_v, y\_logo\_v作为直接进行操作的变量

当confreg模块收到读坐标数据请求时, 根据地址返回32位的坐标信息

assign confreg\_read\_data = get\_confreg\_read\_data(confreg\_addr);

function [31:0] get\_confreg\_read\_data(input [31:0] confreg\_addr);

    begin

        case(confreg\_addr[15:0])

            `DIGITAL\_NUM\_ADDR   :

                get\_confreg\_read\_data = digital\_num\_v;

            `SWITCH\_ADDR        :

                get\_confreg\_read\_data = switch\_v;

            `LED\_ADDR           :

                get\_confreg\_read\_data = {24'b0, led\_v};

            `MID\_BTN\_KEY\_ADDR   :

                get\_confreg\_read\_data = mid\_btn\_key\_v;

            `LEFT\_BTN\_KEY\_ADDR  :

                get\_confreg\_read\_data = left\_btn\_key\_v;

            `RIGHT\_BTN\_KEY\_ADDR :

                get\_confreg\_read\_data = right\_btn\_key\_v;

            `UP\_BTN\_KEY\_ADDR    :

                get\_confreg\_read\_data = up\_btn\_key\_v;

            `DOWN\_BTN\_KEY\_ADDR  :

                get\_confreg\_read\_data = down\_btn\_key\_v;

            `LOGO\_X\_ADDR        :

                get\_confreg\_read\_data = {22'b0, logo\_x\_v};

            `LOGO\_Y\_ADDR        :

                get\_confreg\_read\_data = {22'b0, logo\_y\_v};

            default:

                get\_confreg\_read\_data = 32'b0;

        endcase

    end

endfunction

当写使能信号为1, 地址为坐标预定义的地址时, 向存储坐标的器件中写入数据

assign logo\_x = logo\_x\_v;

wire write\_logo\_x;

assign write\_logo\_x = confreg\_wen & (confreg\_addr[15:0] == `LOGO\_X\_ADDR);

always @ (posedge clk) begin

    if (rst == `RST\_ENABLE) begin

        logo\_x\_v <= 10'b0;

    end

    else begin

        if (write\_logo\_x) begin

            logo\_x\_v <= confreg\_write\_data[9:0];

        end

    end

end

这样就实现了 RISC-V指令中的lw和sw所需要的逻辑了

**生成VGA信号**

顶层模块中定义的位置信息

wire [9:0]       logo\_x; //logo左上角x坐标

wire [9:0]       logo\_y; //logo左上角y坐标

通过confreg模块读取和更改位置信息

confreg confreg0(

            .clk(clk),

            .rst(~rstn),

            .confreg\_wen(confreg\_wen),

            .confreg\_write\_data(confreg\_wdata),

            .confreg\_addr(confreg\_addr),

            .confreg\_read\_data(confreg\_rdata),

            .digital\_num0(digital\_num0),

            .digital\_num1(digital\_num1),

            .digital\_cs(digital\_cs),

            .led(led),

            .switch(switch),

            .mid\_btn\_key(mid\_btn\_key),

            .left\_btn\_key(left\_btn\_key),

            .right\_btn\_key(right\_btn\_key),

            .up\_btn\_key(up\_btn\_key),

            .down\_btn\_key(down\_btn\_key),

            .logo\_x(logo\_x),

            .logo\_y(logo\_y)

        );

通过vga\_driver生成控制信号

vga\_640x480 u2 (

                .pclk(pclk),

                .reset(rst\_n),

                .hsync(hsync),

                .vsync(vsync),

                .valid(valid),

                .h\_cnt(h\_cnt),

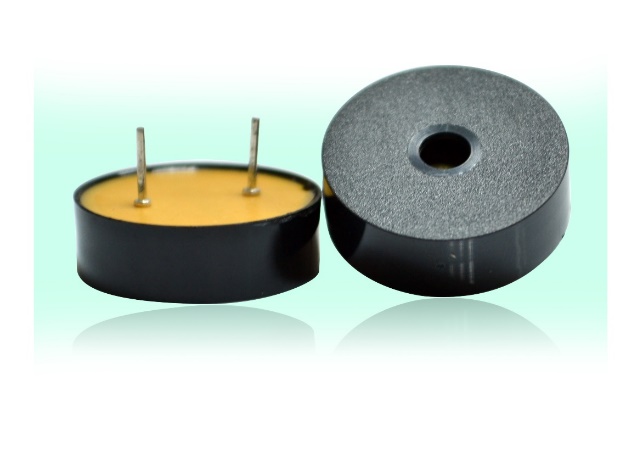
                .v\_cnt(v\_cnt)

            );

这样就实现了生成正确的图像信息和vga信号

## 蜂鸣器

### 6.5.1 无源蜂鸣器

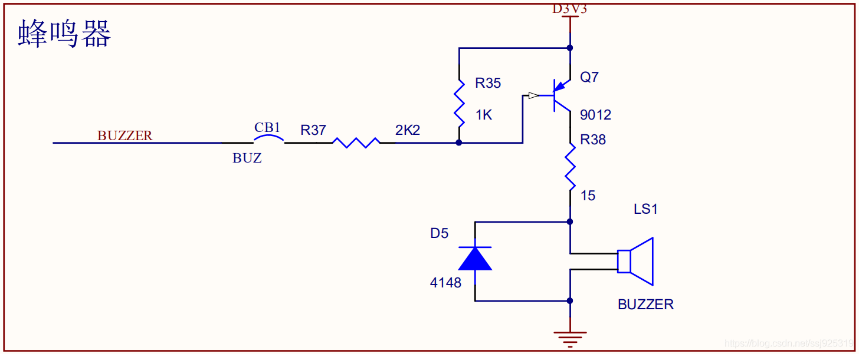


无源蜂鸣器广泛应用于各种电子设备中，例如报警器、电子闹钟、电子游戏、家电、汽车电子等。由于其简单性和易用性，它们成为对声音提示需求较简单的应用的理想选择.

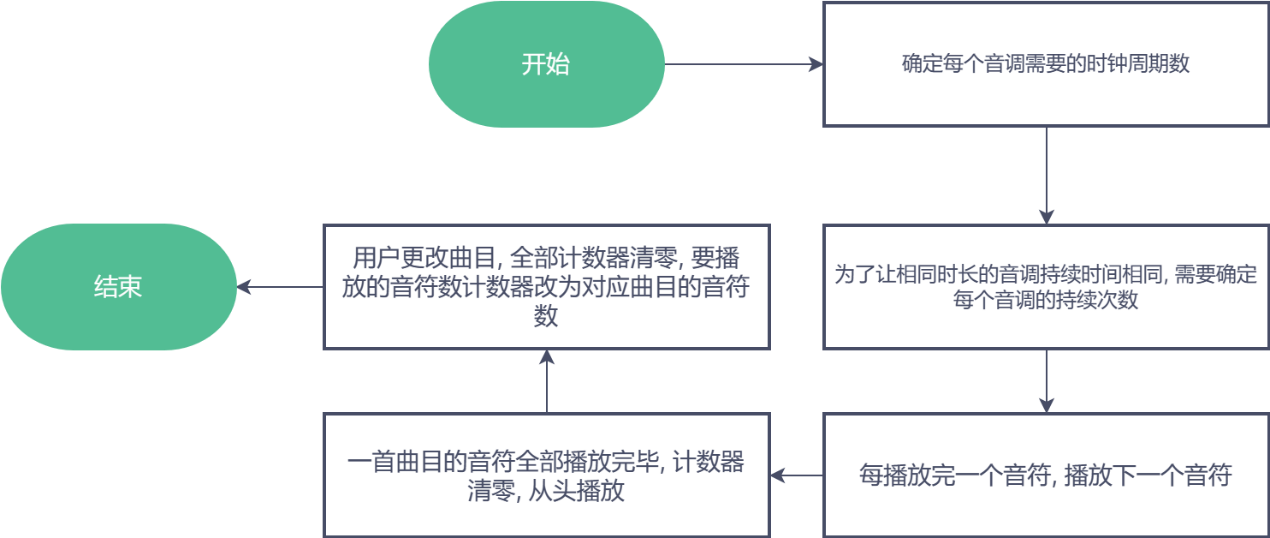
### 6.5.2 怎么响?

每个音调的周期(频率)都不同,所以每个时钟周期不同音调的重复次数都不同;

只需要往蜂鸣器的管脚输出连续的高低变化电平就可以让蜂鸣器发声, 控制重复次数就得到了不同的音调.



### 6.5.3 设计流程



### 6.5.4 代码介绍

#### 6.5.4.1 模块总览

`timescale 1ns / 1ps

module player(

    input               clk     ,       //时钟输入

    input               rst\_n   ,       //复位按键输入

    input           [7:0] song  ,       // 歌曲种类

    output  reg         buzzer          //驱动蜂鸣器

    );

模块比较简单,接受8位的歌曲种类作为输入, 将buzzer作为输出通过约束连接到蜂鸣器引脚上.

#### 6.5.4.2 音调

    //定义音符时序周期

    localparam

                        M0  = 98800,

                        M1  = 191200,

                        M2  = 170300,

                        M3  = 151700,

                        M4  = 143200,

                        M5  = 127500,

                        M6  = 113600,

                        M7  = 101200,

                        L1  = 381600,

                        L2  = 340100,

                        L3  = 303000,

                        L4  = 286500,

                        L5  = 255100,

                        L6  = 227200,

                        L7  = 202400,

                        H1  = 95500,

                        H2  = 85100,

                        H3  = 75800,

                        H4  = 75100,

                        H5  = 63700,

                        H6  = 56800,

                        H7  = 50800;

每个时钟周期(100MHz)相当于多少个音调的周期, 这里共定义了28中音调, 对应了高中低音和休止符.

#### 6.5.4.3 音符种类

    // 定义每个音符对应的常数

    localparam  L1\_E8 = 0,

                L1\_E4 = 1,

                L1\_E2 = 2,

                L1\_E3\_2 = 3,

                L1\_E1 = 4,

                L1\_E1\_2 = 5,

                L1\_E1\_4 = 6,

                L1\_E1\_8 = 7,

                L1\_E1\_16 = 8,

除了音调之外,每个音符还有”持续时间”的属性,这里为每个音调定义了10种持续时间(这样其实都没有完全覆盖音调的音长).

#### 6.5.4.4 歌曲选择

// 根据当前曲目更改音符个数

    always @(posedge clk or negedge rst\_n) begin

        if (!rst\_n) begin

            YINFU <= 48;

        end else begin

            case (flag)

                0: YINFU <= 48; // 小星星

                1: YINFU <= 36; // 两只老虎

                2: YINFU <= 203; // 云宫讯音

                3: YINFU <= 76; // 天空之城

                4: YINFU <= 29; // 任何邪恶

                5: YINFU <= 65; // 打上花火

                6 : YINFU <= 55; // 只因你太美

                default: YINFU <= 48; // 默认播放小星星

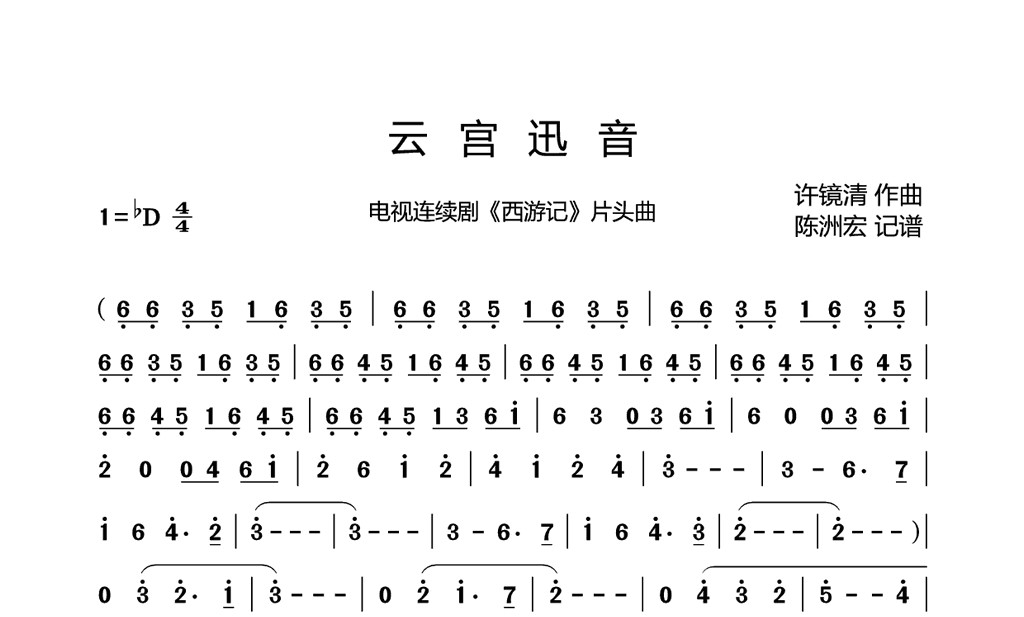
            endcase

        end

    end

一共预置了6首歌曲, 通过Switch按钮选择.

#### 6.5.4.5 简谱映射



            2: begin // 云宫迅音

                case(cnt2)

                    0 : begin pre\_set <= L6; note\_type <= L6\_E1\_2; end

                    1 : begin pre\_set <= L6; note\_type <= L6\_E1\_2; end

                    2 : begin pre\_set <= L3; note\_type <= L3\_E1\_2; end

                    3 : begin pre\_set <= L5; note\_type <= L5\_E1\_2; end

                    4 : begin pre\_set <= M1; note\_type <= M1\_E1\_2; end

                    5 : begin pre\_set <= L6; note\_type <= L6\_E1\_2; end

                    6 : begin pre\_set <= L3; note\_type <= L3\_E1\_2; end

                    7 : begin pre\_set <= L5; note\_type <= L5\_E1\_2; end

                    8 : begin pre\_set <= L6; note\_type <= L6\_E1\_2; end

                    9 : begin pre\_set <= L6; note\_type <= L6\_E1\_2; end

                    10 : begin pre\_set <= L3; note\_type <= L3\_E1\_2; end

                    11 : begin pre\_set <= L5; note\_type <= L5\_E1\_2; end

                    12 : begin pre\_set <= M1; note\_type <= M1\_E1\_2; end

                    13 : begin pre\_set <= L6; note\_type <= L6\_E1\_2; end

                    14 : begin pre\_set <= L3; note\_type <= L3\_E1\_2; end

将简谱映射到Verliog程序.

## 汇编程序

**地址定义&寄存器说明**

# DIGITAL\_NUM\_ADDR    16'h8000 // 0xbfaf\_8000//数码管

# SWITCH\_ADDR         16'h8004 // 0xbfaf\_8004//拨片开关

# LED\_ADDR            16'h8008 // 0xbfaf\_8008//LED

# MID\_BTN\_KEY\_ADDR    16'h800c // 0xbfaf\_800c//中间按键

# LEFT\_BTN\_KEY\_ADDR   16'h8010 // 0xbfaf\_8010//左边按键

# RIGHT\_BTN\_KEY\_ADDR  16'h8014 // 0xbfaf\_8014//右边按键

# UP\_BTN\_KEY\_ADDR     16'h8018 // 0xbfaf\_8018//上边按键

# DOWN\_BTN\_KEY\_ADDR   16'h801c // 0xbfaf\_801c//下边按键

# LOGO\_X\_ADDR         16'h8020 // 0xbfaf\_8020

# LOGO\_Y\_ADDR         16'h8024 // 0xbfaf\_8024

#a0 右边按钮地址    t0 右边按钮值

#a1 左边按钮地址    t1 左边按钮值

#a2 上边按钮地址    t2 上边按钮值

#a3 下边按钮地址    t3 下边按钮值

#a4 logo\_x 地址    t4 logo\_x值

#a5 logo\_y 地址    t5 logo\_y值

#t6 常量1

#s0 循环次数        s1 循环计数器

#s2 LED地址

#s3 switch地址      s4 switch值

**载入地址**

由于RISC-V 指令编码长度所限, 一条地址需要两条指令来完成.

Lui a0, 0xbfaf8 :用于将高5位设置为0xbfaf8, 这在项目中的含义是外设地址

Ori a0, a0, 0x14 :用于将地址的低位设置为0x14, 此时a0代表的地址即0xbfaf8014

即为预定义中的右边按钮的地址

    #地址

    lui a0, 0xbfaf8

    ori a0, a0, 0x14

    lui a1, 0xbfaf8

    ori a1, a1, 0x10

    lui a2, 0xbfaf8

    ori a2, a2, 0x18

    lui a3, 0xbfaf8

    ori a3, a3, 0x1c

    lui a4, 0xbfaf8

    ori a4, a4, 0x20

    lui a5, 0xbfaf8

    ori a5, a5, 0x24

    lui s2, 0xbfaf8

    ori s2, s2, 0x08

    lui s3, 0xbfaf8

    ori s3, s3, 0x04

**主要逻辑**

读取switch的值, 把值存进LED的地址中, 实现通过CPU控制LED的状态, 此功能还能指示当前播放的是第几首歌.

读取图像位置信息, 读取按钮的状态. 并根据按钮的值更改图像的位置信息.

(原点在左上角, 向右是X增大的方向, 向下是Y增大的方向, 据此修改不同范围键对应对坐标值的作用效果)

把值存回存放图像坐标的地址中

由于上板实际效果, 控制速度过快, 添加延时循环, 使其达到合理的速度.

(t6是通过或运算得到的常数1)

disp:

    #把拨片的值赋给LED

    lw s4, (s3)

    sw s4, (s2)

    #加载logo的x和y位置信息

    lw t4, (a4)

    lw t5, (a5)

    #加载按钮的值

    lw t0, (a0)

    lw t1, (a1)

    lw t2, (a2)

    lw t3, (a3)

    #对位置信息更改

    add t4, t4, t0

    sub t4, t4, t1

    add t5, t5, t2

    sub t5, t5, t3

    #存回logo的位置信息

    sw t4, (a4)

    sw t5, (a5)

    #循环计数器置0

    lui s1, 0

    lui s0, 0x00100 #循环次数

delay:

    ori t6, zero, 1 #常量1

    add s1, s1, t6

    beq s1, s0, disp

    jal delay

## 生成图像ROM初始化coe文件

主要要点:

1. 本项目使用的VGA是12位位宽的, R,G,B各4位, 所以使用的coe也只能是12位宽, 处理办法为每12位添加一个’,’

FFF,FFF,FFF,FFF,FFF,FFF,FFF,FFF,FFF,FFF,FFF,FFF,FFF,FFF,FFF,FFF,FFF;

1. Coe的开头申明这幅图片的编码进制和图片的基本信息.

;VGA Memory Map

;.COE file with hex coefficients

;Height: 144,Width: 144

memory\_initialization\_radix=16;

memory\_initialization\_vector=

def image\_to\_hex(image\_path, output\_path):  
 *# 打开图像文件* image = Image.open(image\_path)  
 *# 转换为RGB图像* image = image.convert('RGB')  
 *# 获取图像尺寸* width, height = image.size  
 *# 存储HEX数据* hex\_data = []  
 *# 遍历图像的像素* for y in range(height):  
 for x in range(width):  
 *# 获取像素值* r, g, b = image.getpixel((x, y))  
 *# 将RGB值转换为12位HEX格式并添加到列表中* hex\_value = format(((r >> 4) << 8) | ((g >> 4) << 4) | (b >> 4), '03X')  
 hex\_data.append(hex\_value)  
 *# 写入HEX数据到文件* with open(output\_path, 'w') as output\_file:  
 cnt = 0  
 for hex\_value in hex\_data:  
 output\_file.write(hex\_value + ',')  
 cnt += 1  
 if cnt % 32 == 0:  
 output\_file.write('\n')

# 测试

## 仿真测试用例、结果以及分析

测试用例主要是在RARS上观察寄存器的变化，判断汇编代码是否按预期工作。汇编小组没有在Vivado上进行仿真测试，每得到一版代码都直接进行上板测试，根据上板情况调整Verilog代码和汇编代码。

# 问题及解决方法

1. 新增访问外设的地址。

在confreg中定义地址，然后通过顶层模块对confreg的读写完成对外设的访问。

1. 把图片转成coe格式。

第一步转换成Hex格式，然后转换成12位位宽。由于RARS的标准是转换成32位位宽，与本项目中15个引脚的VGA接口不相符，故需写一个程序将Hex格式代码转成12位位宽的coe文件。格式形如“FFF,FFF;”并在顶部添加格式信息，例如编码采用16进制。

1. 测试外设时发现单周期CPU框架没有实现太多的指令。

尽可能使用and，sub，ori，lui指令写汇编进行外设测试。等待流水线CPU小组的成果。

1. 尝试更改VGA输出分辨率

由于板子产生的频率不一定准，于是最终选用640x480的分辨率，需要的时钟频率是25MHz，可以比较容易地从板子的100MHz得到。

1. 测试外设时发现汇编代码正确但上板却得不到想要的结果。

单周期CPU设计存在缺陷，不能很好地记住之前保存的数值，需要在使用时再次声明。这个问题在本小组开发的流水线CPU上不存在。

# 心得体会及总结

在这个实验中，我们的目标是集成接口控制器和处理器设计结果，形成一个自定义的计算机系统，并设计并完成了VGA控制器、蜂鸣器控制器、按键、拨片和LED灯控制器。最终，我们成功地在精工板上实现了这个系统，并进行了测试，形成了IP核。

在实验过程中，我学到了许多关于计算机系统设计和集成的知识。通过设计和实现接口控制器，学会了如何处理时序和数据传输的问题，以确保各个组件之间的正确连接和工作。在设计VGA控制器、蜂鸣器控制器、按键、拨片和LED灯控制器时，学到了如何应用数字电路设计的知识。通过在FPGA上实现这些控制器，我对数字电路的建模、电路设计和时序控制有了更深入的认识。在测试和验证阶段，学会了如何使用仿真工具来验证系统的功能和正确性。通过仔细设计测试用例并进行仿真，我能够发现和修复设计中的错误和缺陷。这个过程对于确保系统的可靠性和稳定性非常重要。

通过设计和实现集成接口控制器和处理器，加深了对计算机系统设计的理解，这些经验将对我的未来学习和工作有很大的帮助。

# 参考文献有价值的资源推荐

《RISC-V CPU 处理器设计》胡振波主编 人民邮电出版社