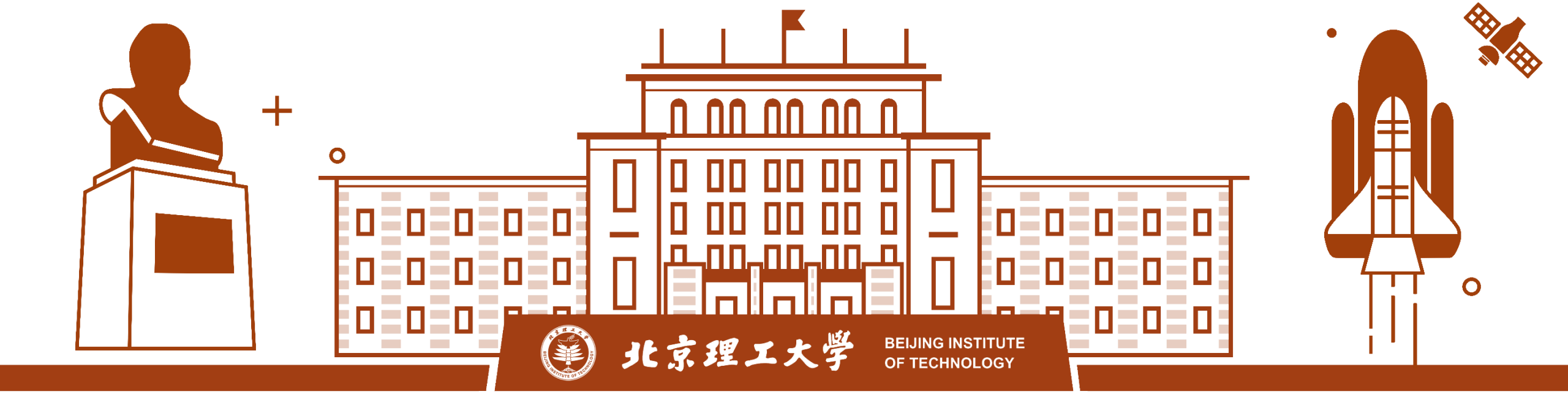


汇编与接口课程设计

答辩人：穆新宇 董若扬

时　间：2024/3/1

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

目录 / CONTENTS

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

PART ONE

概览

蜂鸣器

使用纯硬件的方式编写一个音乐播放器,并形成IP核

能够根据开发板上Switch的值切换不同的预置曲目

VGA/按键

编写一个VGA驱动，输出640×480像素的60Hz信号．编写汇编代码在CPU上运行机器码接收来自按键的控制信号，改变图像的位置信息，并将改变通过VGA信号送到显示器上.

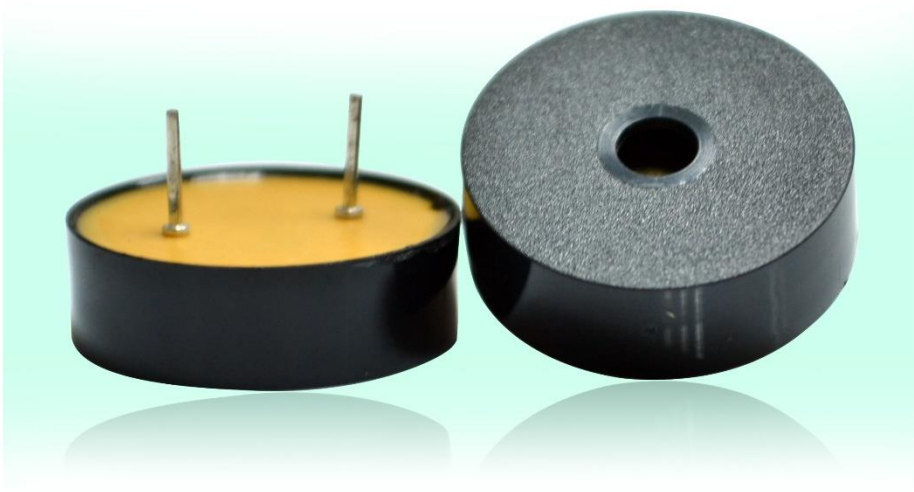
北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

PART TWO

蜂鸣器

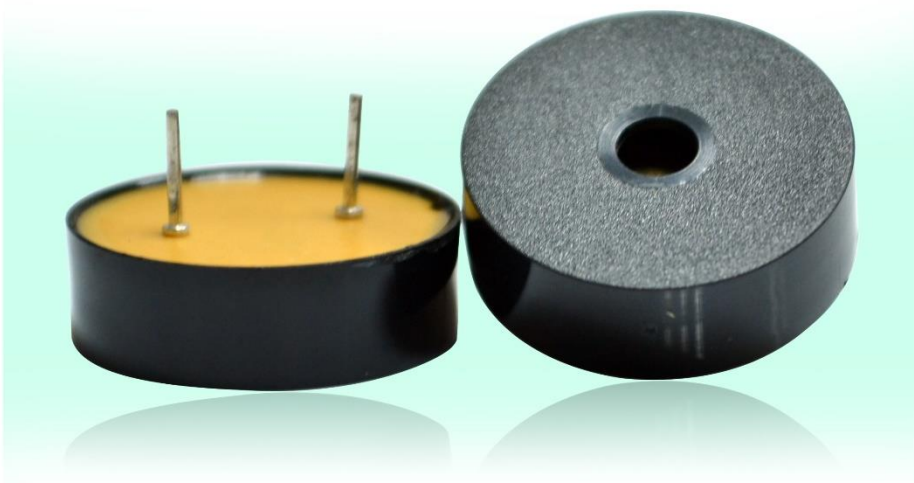
无源蜂鸣器

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。



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.



蜂鸣器

01 确定每个音调需要的时钟周期数

02 为了让相同时常的不同音调持续时间相同,需要确定每个音调持续次数

03 播放一个音符,每播放完一个音符换下一个

04 一首曲目的音符全部播放完毕,计数器清零,从头播放

05 如果用户更改曲目,全部计数器清零,更改音符数为该曲目的音符数

蜂鸣器


```
`timescale 1ns / 1ps
module player(
    input               clk     ,
    input               rst_n   ,
    input             [7:0] song  , // 歌曲种类

    output  reg         buzzer
    );
```

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

蜂鸣器


```
// 每个时钟周期(100MHz)相当于多少个音调的周期
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

蜂鸣器

```
// 定义音符种类
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,
            L2_E8 = 9,
```

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)

蜂鸣器


```
// 根据当前曲目更改音符个数[note n.音符]
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        note_cnt <= 48;
    end else begin
        case (flag)
            0: note_cnt <= 48; // 小星星
            1: note_cnt <= 36; // 两只老虎
            2: note_cnt <= 203; // 云宫讯音
            3: note_cnt <= 76; // 天空之城
            4: note_cnt <= 29; // 任何邪恶
            5: note_cnt <= 65; // 打上花火
            default: note_cnt <= 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

蜂鸣器


```
2: begin // 云宫讯首
    case(cnt2)
        0 : begin melody <= L6; note_type <= L6_E1_2; end
        1 : begin melody <= L6; note_type <= L6_E1_2; end
        2 : begin melody <= L3; note_type <= L3_E1_2; end
        3 : begin melody <= L5; note_type <= L5_E1_2; end
        4 : begin melody <= M1; note_type <= M1_E1_2; end
        5 : begin melody <= L6; note_type <= L6_E1_2; end
        6 : begin melody <= L3; note_type <= L3_E1_2; end
        7 : begin melody <= L5; note_type <= L5_E1_2; end
        8 : begin melody <= L6; note_type <= L6_E1_2; end
        9 : begin melody <= L6; note_type <= L6_E1_2; end
       10 : begin melody <= L3; note_type <= L3_E1_2; end
       11 : begin melody <= L5; note_type <= L5_E1_2; end
       12 : begin melody <= M1; note_type <= M1_E1_2; end
       13 : begin melody <= L6; note_type <= L6_E1_2; end
       14 : begin melody <= L3; note_type <= L3_E1_2; end
       15 : begin melody <= L5; note_type <= L5_E1_2; end
       16 : begin melody <= L6; note_type <= L6_E1_2; end
```

每个音符包括melody(音调)，note_type(持续时间)两个属性

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

PART THREE

VGA

VGA接口控制器路线选择

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

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:

1. 没有使用CPU.
2. 图像信号唯一确定，不可控制更改.
3. 难度低.

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.

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信号

```
// 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
```

vga_640x480

定义该模块从顶层模块接收时序信号，生成VGA信号

VGA 常用分辨率时序参数

显示模式	时钟 /MHz	行时序参数(单位：像素)					列时序参数(单位：行)				
		a	b	c	d	e	f	g	h	i	k
640x480@60Hz	25.175	96	48	640	16	800	2	33	480	10	525
800x600@60Hz	40	128	88	800	40	1056	4	23	600	1	623
1024x768@60Hz	65	136	160	1024	24	1344	6	29	768	3	806
1280x720@60Hz	74.25	40	220	1280	110	1650	5	20	720	5	750
1280x1024@60Hz	108	112	248	1280	48	1688	3	38	1024	1	1066
1920x1080@60Hz	148.5	44	148	1920	88	2200	5	36	1080	4	1125

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.

VGA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
```

display

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.

按键控制

学习了confreg中对按键的行为定义

```
// mid_btn_key_r : 用于记录中间按钮的状态
// mid_btn_key_v : 用于将中间按钮的状态输出到外部
// mid_btn_key_flag : 用于消除抖动
// mid_btn_key_count : 用于计数
// mid_btn_key_start : 用于检测按键按下
// mid_btn_key_end : 用于检测按键松开
// mid_btn_key_sample : 用于检测按键状态, 按下时为 1

/*********** mid_btn_key ***********/
reg mid_btn_key_r;
assign mid_btn_key_v = {31'd0, mid_btn_key_r};// 32 位数据。31 个最高位始终为 0, 最低位与按键的输入相关

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];
```

//////////

按键控制

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.

```
always @ (posedge clk) begin
    //如果重置已启用，则重置标志
    if (rst == `RST_ENABLE) begin
        mid_btn_key_flag <= 1'b0;
    end
    //否则，检查按钮是否已采样
    else if (mid_btn_key_sample) begin
        //如果对按钮进行了采样，则重置标记
        mid_btn_key_flag <= 1'b0;
    end
    //否则，检查按钮是否按下或松开
    else if (mid_btn_key_start || mid_btn_key_end) begin
        //如果按下或松开按钮，则启用标记
        mid_btn_key_flag <= 1'b1;
    end

    //如果复位已启用或标志已禁用，则复位计数器
    if (rst == `RST_ENABLE || !mid_btn_key_flag) begin
        mid_btn_key_count <= 20'b0;
    end
    //否则，计数器将递增
    else begin
        mid_btn_key_count <= mid_btn_key_count + 1'b1;
    end

    //如果启用复位或对按钮进行采样，则复位寄存器
    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
```

存放位置信息

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//图像X坐标存放地址
`define LOGO_Y_ADDR          16'h8024 // 0xbfaf_8024//图像Y坐标存放地址
```

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

```
1   function [31:0] get_confreg_read_data(input [31:0] confreg_addr);
2       begin
3           case(confreg_addr[15:0])
4               `LOGO_X_ADDR:
5                   get_confreg_read_data = {22'b0, logo_x_v};
6               `LOGO_Y_ADDR:
7                   get_confreg_read_data = {22'b0, logo_y_v};
8           endcase
9       end
10  endfunction
```

存放位置信息

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

```
1   assign logo_x = logo_x_v;
2
3   wire write_logo_x;
4   assign write_logo_x = confreg_wen & (confreg_addr[15:0] == `LOGO_X_ADDR);
5   always @ (posedge clk) begin
6       if (rst == `RST_ENABLE) begin
7           logo_x_v <= 10'b0;
8       end
9       else begin
10          if (write_logo_x) begin
11              logo_x_v <= confreg_write_data[9:0];
12          end
13      end
14  end
```

这样就实现了 RISC-V指令中的lw和sw所需要的逻辑了

```
lw t4, (a4)
sw t4, (a4)
```

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→

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

显示图像的方法与EES338 VGA示例代码一致，这里就不赘述了

```
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)
            );
endmodule
```

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

```
text
    lui a0, 0xbfaf8
    lui a1, 0xbfaf8
    lui a2, 0xbfaf8
    lui a3, 0xbfaf8
    lui a4, 0xbfaf8
    lui a5, 0xbfaf8
    lui s2, 0xbfaf8
    lui s3, 0xbfaf8
    ori a3, a3, 0x1c
    ori a4, a4, 0x20
    ori a5, a5, 0x24
    ori a0, a0, 0x14
    ori a1, a1, 0x10
    ori a2, a2, 0x18
    ori s2, s2, 0x08
    ori s3, s3, 0x04
```

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

汇编代码

主要逻辑

```
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
```

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)

```
    #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
```

由于乐学提供的示例代码single_cycle是一个较为简单的CPU，在编写汇编上板测试期间，出现添加延时循环后代码就无法工作的现象，经咨询助教，是由于t6的值没有被正确保留，所以放在了循环代码块内.

生成图像ROM初始化coe文件

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

Jpg转coe

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,
FFC,FFC,FEB,EB8,C86,832,942,FB9,
620,DA7,A53,A53,B63,FCA,FFC,FFC,
```

2. 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: 78,Width: 169

memory_initialization_radix=16;
memory_initialization_vector=
```

```python
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')
```

上板效果

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

PART FOUR

总结

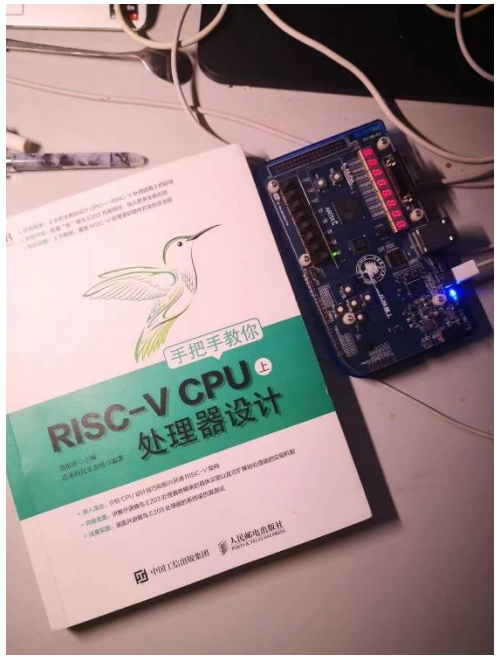
RISC-V Pipeline CPU

开源的RISC-V架构的光明的未来是我们小组选择RISC-V架构的原因. 我们小组实现了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.

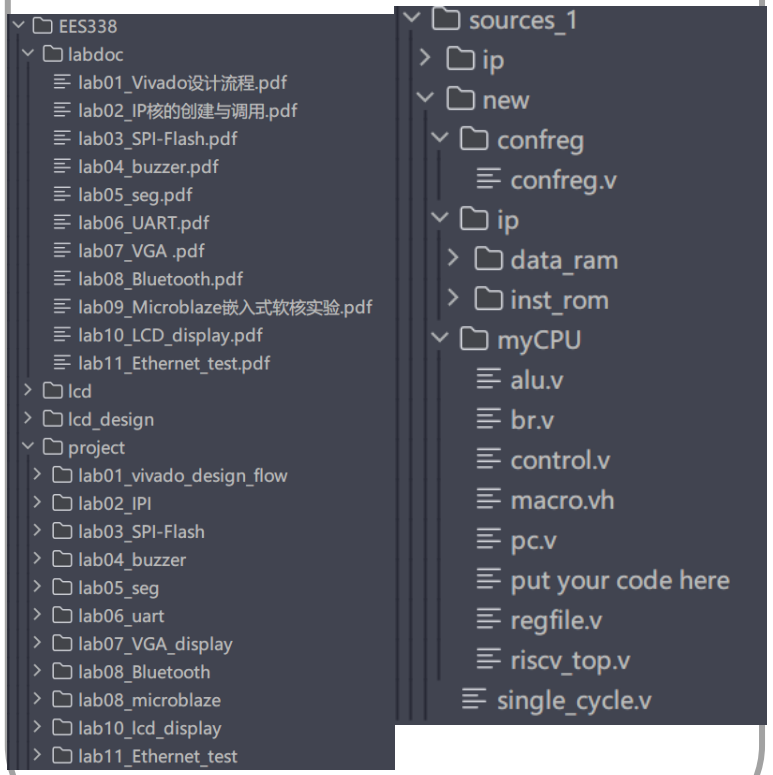
参考资料

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

《RISC-V 处理器设计》



EES338 教程和 single_cycle示例代码



助教答疑