同济大学计算机系

数字逻辑课程综合实验报告

学　　号 2352595

姓　　名 张嘉麟

专　　业 计算机科学与技术（精英班）

授课老师 郭玉臣

一、实验内容

本实验基于 Nexys-A7 电路板，利用 VGA、键盘、MP3 外设实现了两个数字系统。第一个实现了对室内温度的测量，以华氏度和摄氏度的形式在七段数码管上显示，并使用 MP3 播放音乐。第二个实现了通过 VGA 显示、键盘操作、MP3 放音乐的反弹箱小游戏。

系统设计涵盖以下主要功能模块：

1.VGA 显示模块

实现基于 VGA 协议的图像显示功能，包括像素时序生成、RGB 数据输出以及同步信号的控制。模块用于在显示器上绘制静态或动态图形，并验证时序的准确性。

2.PS/2 键盘模块

实现对 PS/2 键盘的通信，包括键盘扫描码的接收与解码、按键事件的识别和处理。模块支持单键、多键操作，以及键盘输入事件的实时响应。

3.MP3 播放模块

基于 VS1003 音频解码芯片，实现音频文件的加载、解码和播放功能。模块通过 SPI 协议与 MP3 解码芯片通信，并支持音频参数配置和播放控制。

4.温度传感器模块

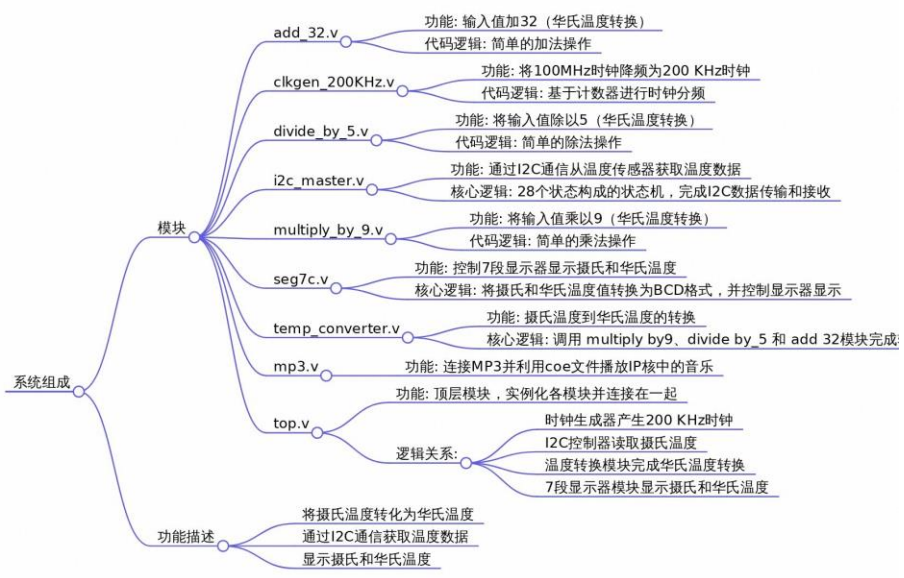
通过 I2C 接口与 ADT7420 温度传感器通信，完成温度数据的读取和转换，并实时监测环境温度。模块还实现温度上下限及临界值报警功能。

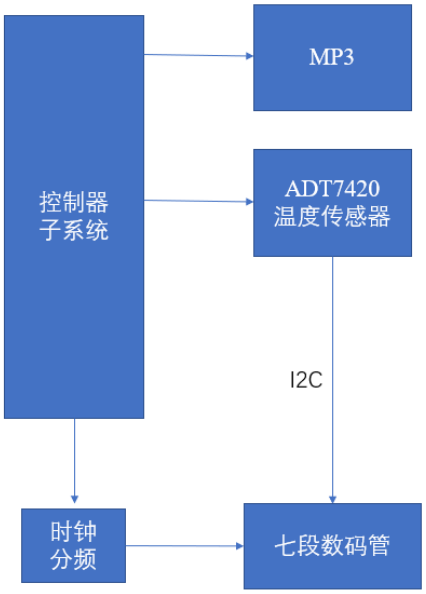
5.系统控制模块

实现对各子模块的统一管理，包括时序调度、模块间数据交互以及系统的整体功能协调。

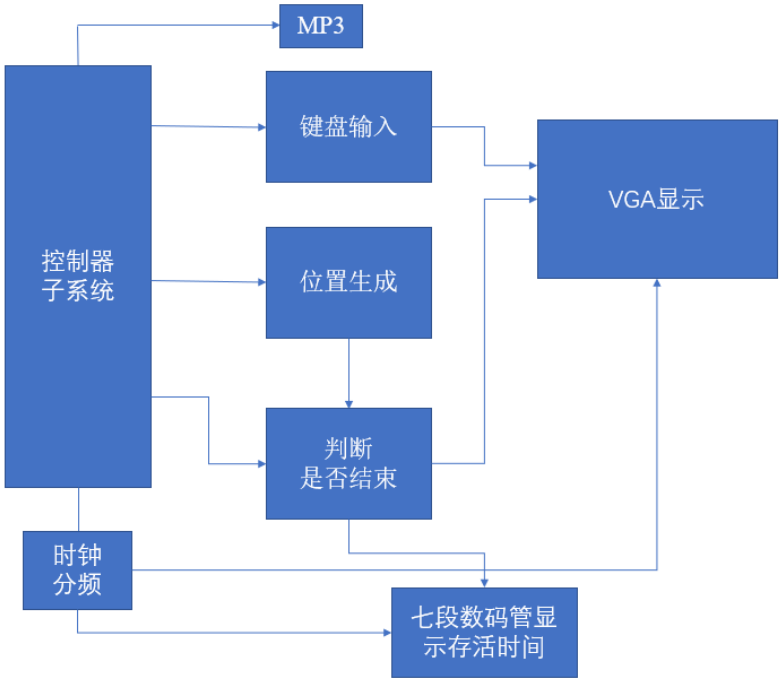
二、系统总框图

(一) MP3+温度传感器数字系统总框图



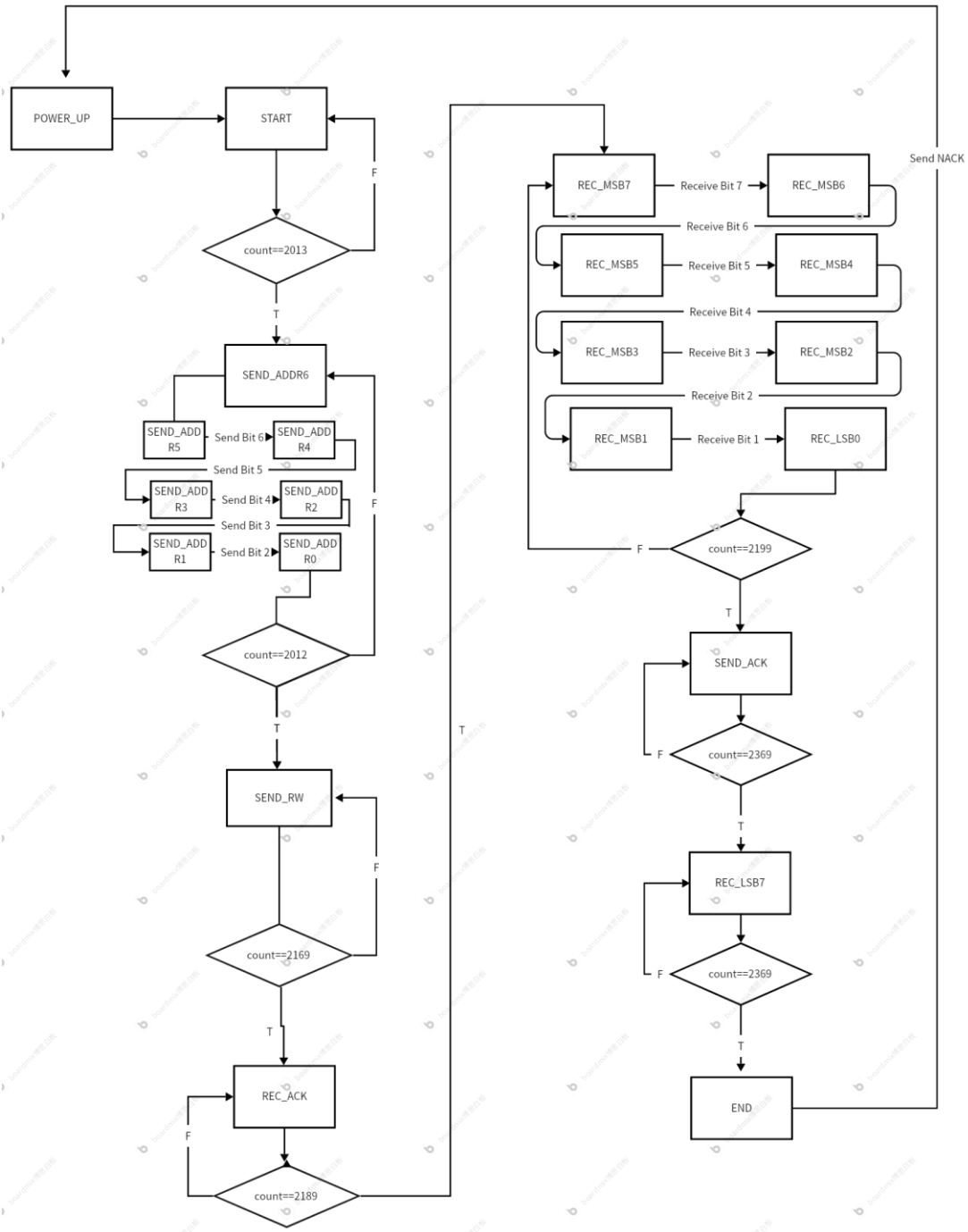


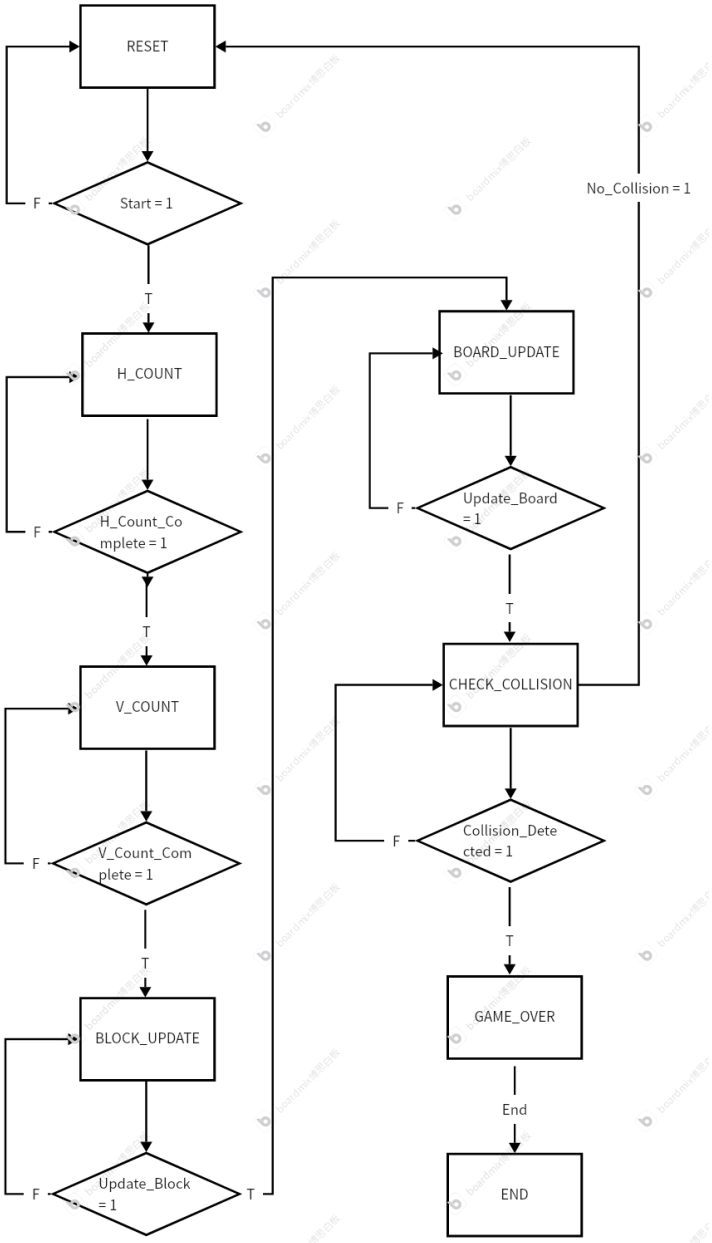
(二) VGA+MP3+键盘数字系统总框图



三、系统控制器设计

（要求画出所设计数字系统的 ASM 流程图，列出状态转移真值表。由状态转移真值表，求出系统控制器的次态激励函数表达式和控制命令逻辑表达式，并用 Logisim 画出系统控制器逻辑方案图。）


POWER_UP
START
count==2013
F
T
SEND_ADDR6
SEND_ADD R5
Send Bit 6
SEND_ADD R4
Send Bit 5
SEND_ADD R3
Send Bit 4
SEND_ADD R2
Send Bit 3
SEND_ADD R1
Send Bit 2
SEND_ADD R0
count==2012
SEND_RW
count==2169
REC_ACK
count==2189
REC_MSB7
Receive Bit 7
REC_MSB6
Receive Bit 6
REC_MSB5
Receive Bit 5
REC_MSB4
Receive Bit 4
REC_MSB3
Receive Bit 3
REC_MSB2
Receive Bit 2
REC_MSB1
Receive Bit 1
REC_LSB0
count==2199
SEND_ACK
count==2369
REC_LSB7
count==2369
END
Send NACK



1. 状态编码

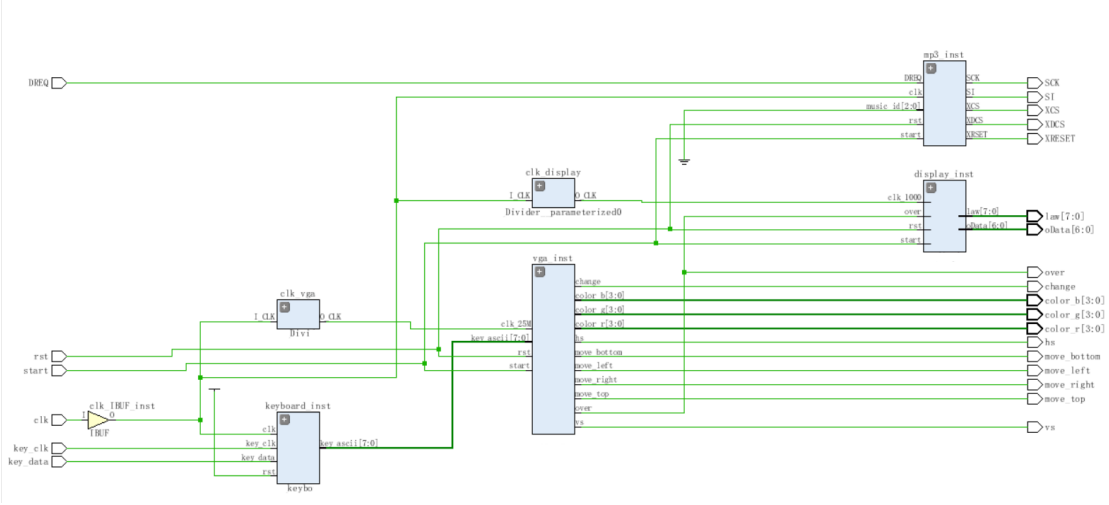
为了推导逻辑表达式，首先为状态机中的每个状态分配二进制编码。二进制编码表示 28 个状态：

状态	二进制编码
POWER_UP	00000
START	00001
SEND_ADDR6	00010
SEND_ADDR5	00011
SEND_ADDR4	00100

状态	二进制编码
SEND_ADDR3	00101
SEND_ADDR2	00110
SEND_ADDR1	00111
SEND_ADDR0	01000
SEND_RW	01001
REC_ACK	01010
REC_MSB7	01011
REC_MSB6	01100
REC_MSB5	01101
REC_MSB4	01110
REC_MSB3	01111
REC_MSB2	10000
REC_MSB1	10001
REC_MSB0	10010
SEND_ACK	10011
REC_LSB7	10100
REC_LSB6	10101
REC_LSB5	10110
REC_LSB4	10111
REC_LSB3	11000
REC_LSB2	11001
REC_LSB1	11010
REC_LSB0	11011
NACK	11100

2. 次态激励函数表达式

H_COUNT = RESET ∧ Start
V_COUNT = H_COUNT ∧ H_Count_Complete
BLOCK_UPDATE = V_COUNT ∧ V_Count_Complete
BOARD_UPDATE = BLOCK_UPDATE ∧ Update_Block
CHECK_COLLISION = BOARD_UPDATE ∧ Update_Board
GAME_OVER = CHECK_COLLISION ∧ Collision_Detected
RESET = CHECK_COLLISION ∧ No_Collision



四、子系统模块建模

（该部分要求对实验中的所有子系统模块进行描述，给出各子系统的功能框图及接口信号定义，并列出各模块建模，图文描述，描述清楚设计及实现的思路，不要在此处放 verilog 代码，所有的 verilog 代码在附录部分）

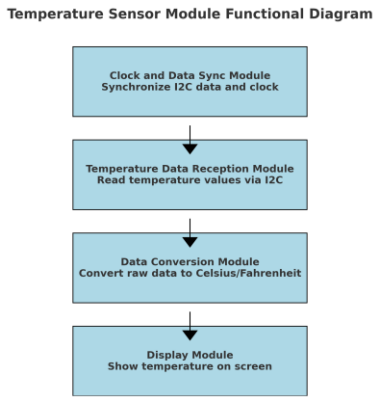
（一）温度传感器模块

1. 模块功能描述

Nexys 板载温度传感器模块（ADT7420）主要功能是通过 I2C 通信协议测量温度并提供报警信号。具体功能包括：

温度测量：提供最高 16 位分辨率的温度数据，精度可达 ±0.25°C。
温度阈值报警：通过 TMP_INT 和 TMP_CT 引脚实现温度上下限及临界阈值的报警输出。
I2C 通信：支持标准 I2C 接口协议，作为从设备与 FPGA 交互。

2. 温度传感器功能框图



3. 接口信号定义

ADT7420 模块的主要接口信号如下：

信号名称	功能描述	方向	FPGA 引脚
SCL	I2C 串行时钟信号	输入/输出	C14
SDA	I2C 串行数据信号	输入/输出	C15
TMP_INT	超温/低温报警信号	输出	D13
TMP_CT	临界超温报警信号	输出	B14

4. 设计与实现思路

(1) 温度测量

设计思路：

ADT7420 开机默认可直接作为温度传感器使用，无需配置即可读取温度数据。温度数据存储在温度寄存器中，以 16 位补码表示。通过读取两字节数据，并右移 3 位后乘以 0.0625，可得摄氏温度。

实现方法：

I2C 主机发送从设备地址 0x4B 和读指令。

连续读取两字节数据，将结果转换为摄氏温度。

(2) 温度阈值报警

设计思路：

通过 TMP_INT 和 TMP_CT 引脚输出超温、低温或临界温度报警信号。具体触发条件由 TLOW、THIGH 和 TCRIT 寄存器配置。

实现方法：

配置 TLOW（低温阈值）、THIGH（高温阈值）寄存器。

配置 TCRIT（临界温度阈值）寄存器。

使用 TMP_INT 和 TMP_CT 报警信号连接 FPGA，并在 FPGA 内设置上拉电阻。

(3) 数据传输与控制

设计思路：

通过 I2C 协议实现温度数据和配置寄存器的读写。支持 I2C 读写模式和地址自动递增功能。

实现方法：

发送写操作：主机写入寄存器地址及数据。

发送读操作：主机写入寄存器地址后，重启通信并发送读命令，读取数据。

5. 实验结果

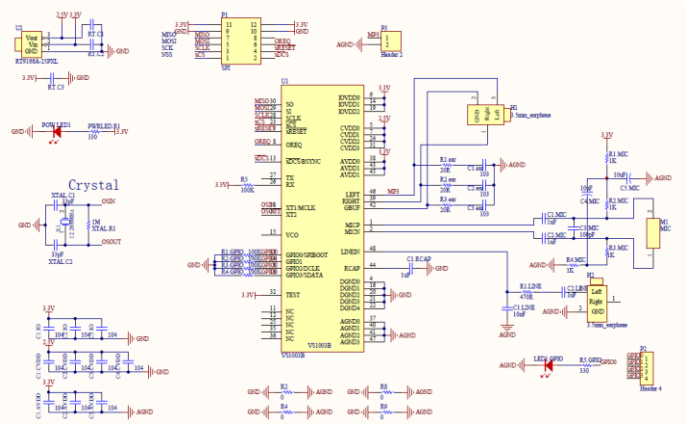
成功实现温度数据的实时读取，测量精度达 ±1°F。

超温和临界温度报警功能验证通过。

支持快速模式（无需配置）和自定义模式（配置寄存器）。

（二）MP3 模块

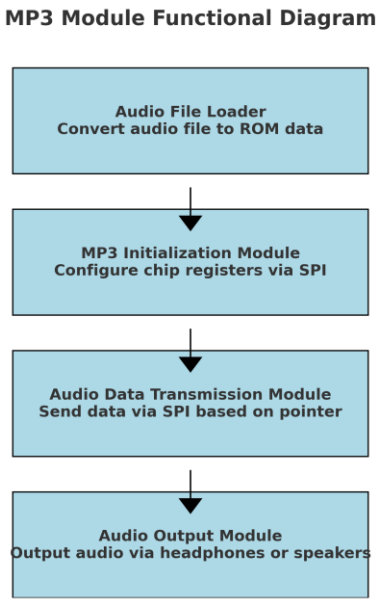
器件型号：VS1003B-MP3
内部电路原理图：



1. 模块功能描述

MP3 模块的主要功能是实现音频数据的加载、初始化及播放。具体功能包括：
音频文件加载：将音频文件转换为可加载的 ROM/RAM 数据。
音频参数初始化：通过 SPI 协议初始化 VS1003 芯片，设置播放模式、音效参数及音量大小。
音频数据传输：按照 SPI 协议传输音频数据至 MP3 芯片，完成音频播放。

2. MP3 功能框图



3. 接口信号定义

MP3 模块的主要接口信号如下：

信号名称	功能描述	方向	引脚位置	标准
clk	时钟输入	输入	E3	LVCMOS33
SCK	SPI 时钟信号	输出	J3	LVCMOS33
SI	SPI 数据输入信号	输出	J4	LVCMOS33
XCS	控制命令片选信号	输出	E7	LVCMOS33
XDCS	数据传输片选信号	输出	K1	LVCMOS33
XRESET	MP3 芯片复位信号	输出	F6	LVCMOS33
DREQ	数据请求信号	输入	J2	LVCMOS33
oData[6:0]	数码管显示信号	输出	多引脚定义	LVCMOS33

4. 设计与实现思路

(1) 音频文件加载模块

设计思路：

将音频文件通过工具转化为 .coe 格式。文件包含以 32 位为单位的音频数据，并存储在 RAM 或 ROM 中，地址指针用于读取音频数据。

实现方法：

使用 Vivado 中的 Block Memory Generator IP 核，将 .coe 文件加载到单端口 RAM/ROM。

地址指针 pos 负责按序读取音频数据。

(2) MP3 初始化模块

设计思路：

通过 SPI 协议完成 VS1003 的寄存器初始化，设置播放模式、音效参数（低音提升、高音调整）及左右声道音量。

实现方法：

使用 XCS 控制信号对寄存器进行初始化。

每次发送 32 位数据，按照 MSB 优先顺序传输。

(3) 音频数据传输模块

设计思路：

将音频数据按 SPI 协议从 RAM/ROM 中读取并传输到 MP3 芯片。传输状态由 DREQ 信号指示。

实现方法：

XDCS 信号控制数据传输。

每次传输 32 位数据，传输完成后 XDCS 置高。

(4) 音频信号输出模块

设计思路：

MP3 芯片通过耳机或扬声器输出音频信号。

实现方法：

将音频信号从 oData 输出，并连接耳机或扬声器。

确保信号的时钟同步性。

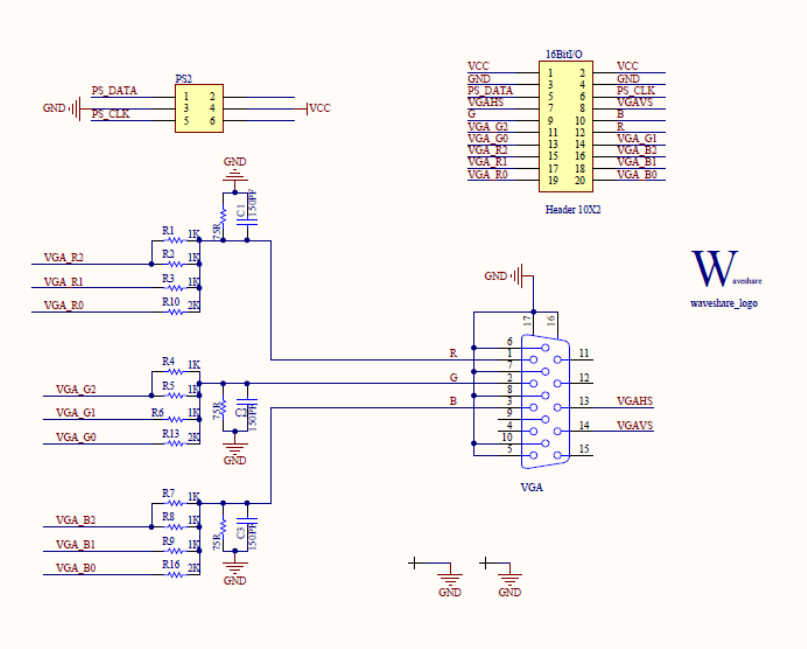
5. 实验结果

音频文件成功加载并播放。

实验验证了音频数据的 SPI 通信及播放功能。

（三）VGA 显示器模块

内部原理图如下：



1. 模块功能描述

VGA 显示模块的主要功能是通过 VGA 接口向显示器输出视频信号，实现图形显示。模块实现包括：

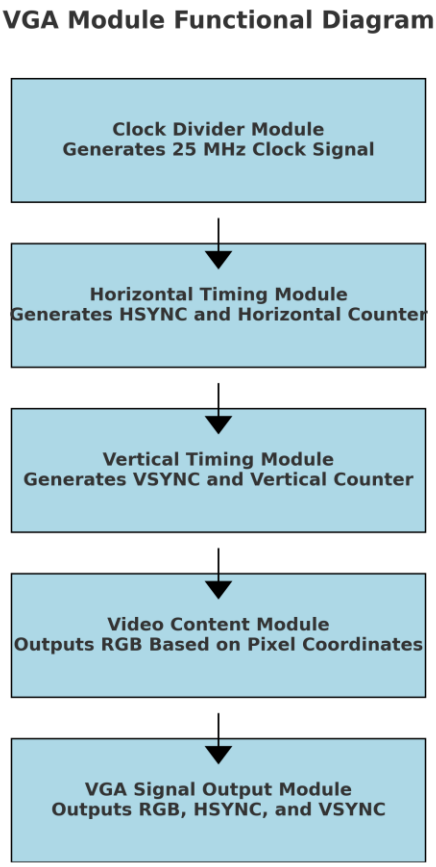
水平与垂直同步信号生成：生成 VGA 行同步（HSYNC）和场同步（VSYNC）信号。

像素数据输出：按照屏幕分辨率逐点输出 RGB 彩色数据。

有效显示区域控制：实现显示内容的定位和动态更新。

2. VGA 模块功能框图

VGA 模块的功能框图如下：



3. 接口信号定义

根据约束文件，VGA 模块的主要接口信号如下：

信号名称	功能描述	方向	引脚位置	标准
clk	输入时钟信号	输入	E3	LVCMOS33
color_r	红色分量信号	输出	多引脚定义	LVCMOS33
color_g	绿色分量信号	输出	多引脚定义	LVCMOS33
color_b	蓝色分量信号	输出	多引脚定义	LVCMOS33
hs	水平同步信号	输出	B11	LVCMOS33
vs	垂直同步信号	输出	B12	LVCMOS33

---

4. 设计与实现思路

(1) 时钟分频模块

设计思路：

通过主时钟信号分频生成 VGA 所需的 25 MHz 时钟信号，用于控制像素刷新速率。

实现方法：

使用分频器模块将主时钟信号（如 100 MHz）分频为 25 MHz。

(2) 行时序生成模块

设计思路：

生成水平同步信号（HSYNC）并控制水平像素计数，按行扫描完成一帧。
实现方法：
根据 VGA 标准行时序（例如 640×480 @ 60 Hz）：
行同步时间：3.8 μs
行消隐时间：1.9 μs
行视频有效时间：25.4 μs
行前肩时间：0.4 μs
(3) 场时序生成模块
设计思路：
生成垂直同步信号（VSYNC）并控制垂直行计数，按帧刷新显示内容。
实现方法：
根据 VGA 标准场时序：
场同步时间：63.5 μs
场消隐时间：31.7 μs
场视频有效时间：15.3 ms
场前肩时间：0.5 ms
(4) 视频内容生成模块
设计思路：
根据水平和垂直计数值计算当前像素坐标，输出对应的 RGB 数据。
实现方法：
在有效显示区域输出 RGB 信号：
红色分量由 `color_r` 输出。
绿色分量由 `color_g` 输出。
蓝色分量由 `color_b` 输出。
在非显示区域保持 RGB 信号为 0。

5. 实验结果

实现了 VGA 的 640×480 分辨率图形显示，刷新率为 60 Hz。
成功生成了 HSYNC 和 VSYNC 信号，与显示器同步。
显示器正确显示预设图像内容（如矩形、文字等）。

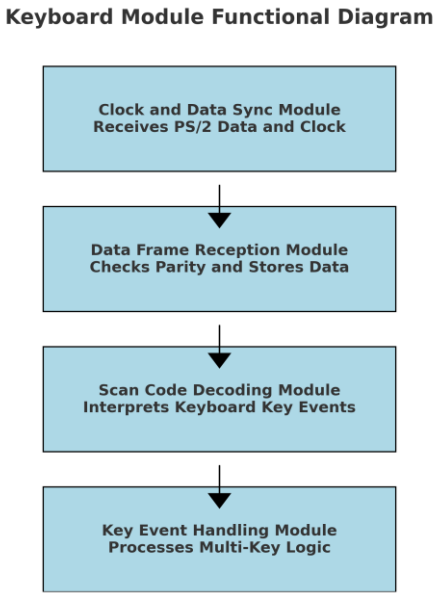
（四）键盘模块

1. 模块功能描述

PS/2 键盘模块的主要功能是通过 PS/2 接口实现键盘的扫描码接收与解码。具体功能包括：
键盘输入信号的采集：基于 PS/2 串行协议接收键盘扫描码。
扫描码解码：解析 Make Code（按键按下）、Break Code（按键松开）以及扩展键码。
按键状态处理：支持多按键同时操作，区分不同按键事件（按下与松开）。

2. 键盘模块功能框图

键盘模块的功能框图如下：



3. 接口信号定义

根据 PS/2 键盘接口的特点，模块的主要接口信号如下：

信号名称	功能描述	方向	引脚位置	标准
clk	时钟输入	输入	E3	LVCMOS33
key_clk	PS/2 键盘时钟信号	输入	F4	LVCMOS33
key_data	PS/2 键盘数据信号	输入	B2	LVCMOS33
rst	模块复位信号	输入	J15	LVCMOS33
key_ascii	键盘按键 ASCII 输出	输出	N/A	LVCMOS33

4. 设计与实现思路

(1) 时钟与数据同步模块

设计思路：

PS/2 数据传输基于时钟与数据同步工作，每发送 1 位数据对应 1 个时钟脉冲。

实现方法：

在 key_clk 信号的上升沿读取 key_data。

数据帧的每一位（起始位、数据位、校验位、停止位）依次采集。

(2) 数据帧接收模块

设计思路：

每帧 PS/2 数据包含起始位、8 位数据、1 位奇校验位和停止位。

实现方法：

使用状态机接收完整的 11 位数据帧。

校验奇偶性，确保数据传输无误。

(3) 扫描码解码模块

设计思路：

解析接收到的扫描码，区分按键事件（Make Code、Break Code、扩展键码）。

实现方法：

通过扫描码表对数据进行映射。
识别特殊扩展键（如功能键）和组合按键。
(4) 按键事件处理模块
设计思路：
处理多按键状态，支持按键的组合事件和连续输入。
实现方法：
使用寄存器存储按键状态。
在按键松开（Break Code）时清除对应状态。

5. 实验结果

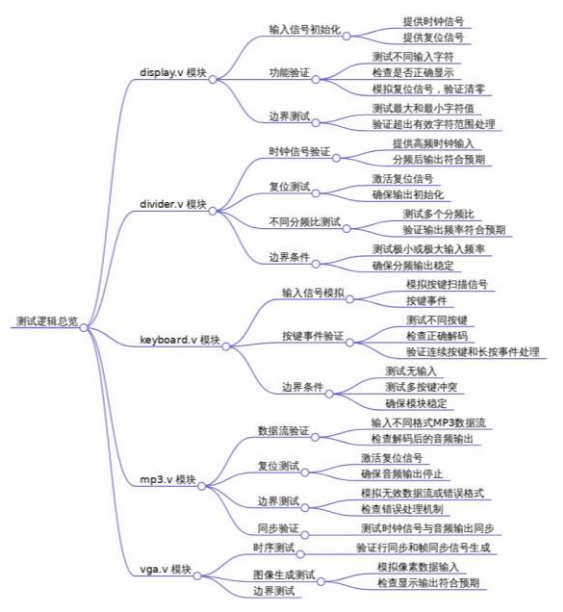
成功接收并解码 PS/2 键盘的扫描码。
支持常规按键和扩展键的识别（如 Shift、Delete）。
多按键同时操作无冲突，按键状态处理正确。

五、测试模块建模

（要求列写各建模模块的 test bench 模块测试逻辑，不要在此处放 verilog 代码，所有的 verilog 代码在附录部分）

1. MP3+温度传感器：

2. VGA+MP3+键盘:

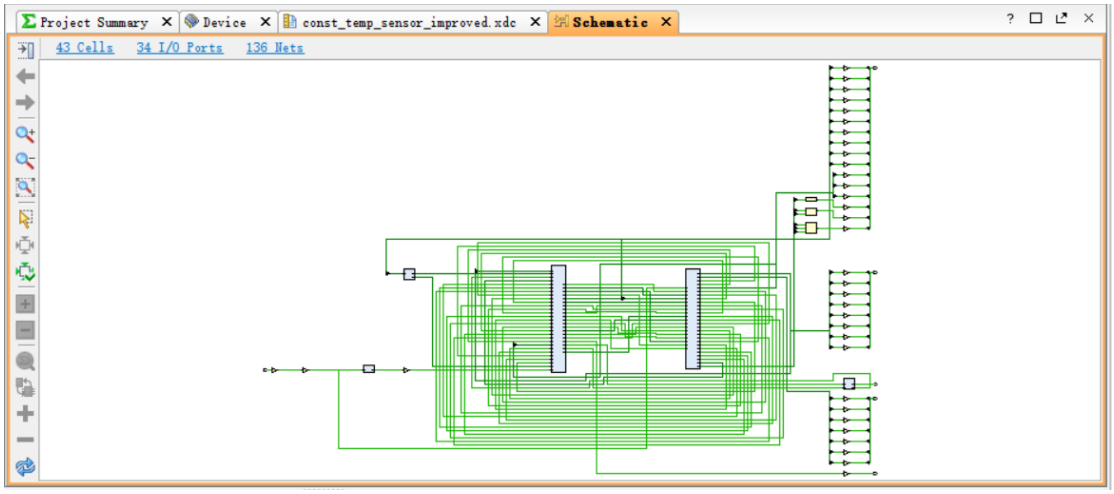


六、实验结果

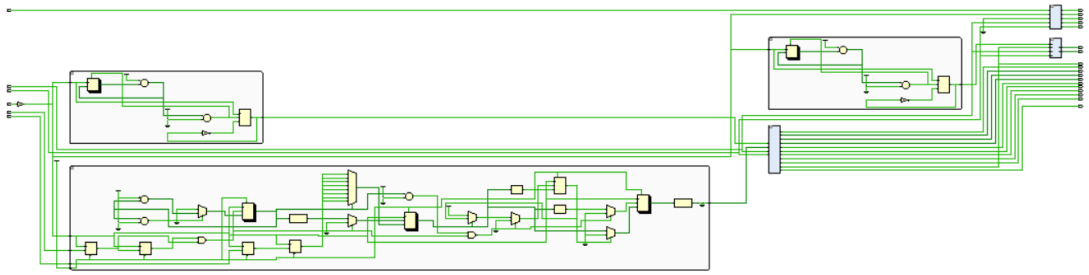
（该部分可截图说明，可包含 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图）

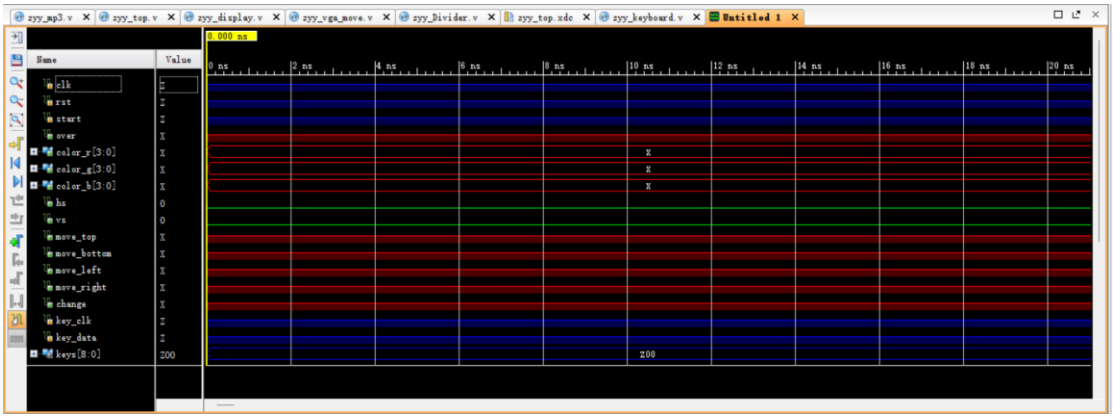
1.logisim 逻辑验证图

(1)MP3+温度传感器:

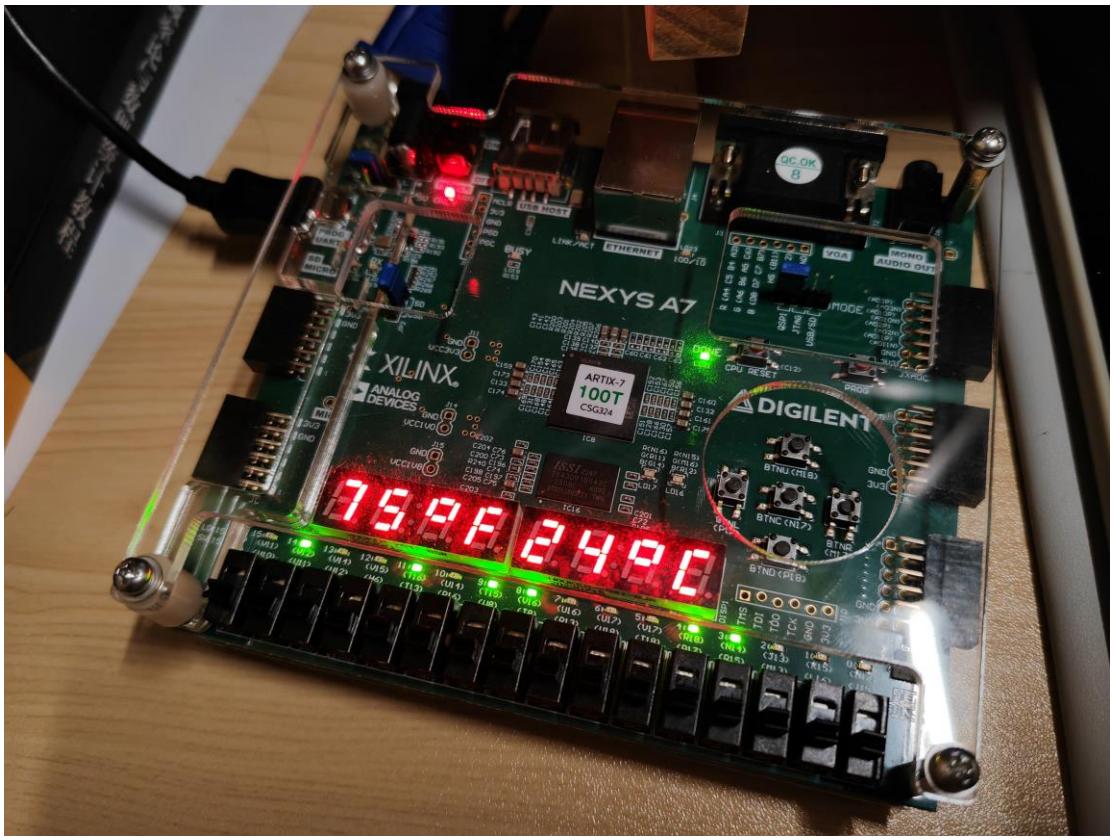


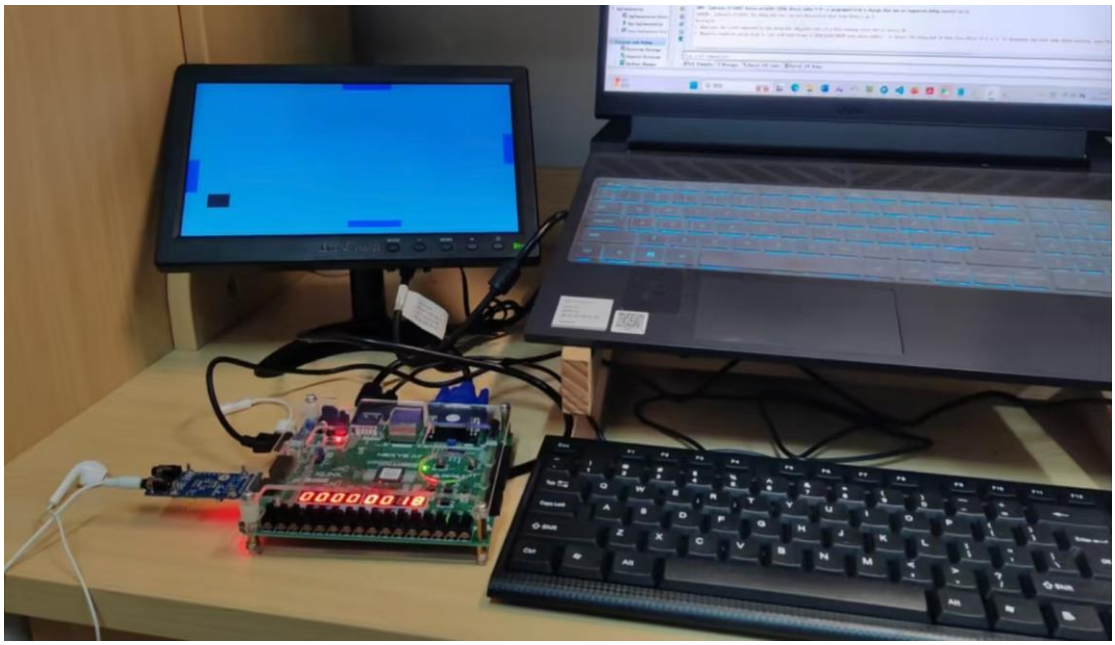
(2)VGA+MP3+键盘：





2.下板实验结果图





七、结论

八、心得体会及建议

九、附录

（该部分放 verilog 代码，包括所有设计文件和测试文件）

MP3+温度传感器系统

1. mp3.v

```
module mp3(clk,rst,start,music_id,XRSET,DREQ,XCS,XDCS,SI,SCK);
input clk;
input rst;
input start;
input [2:0] music_id;
// mp3
output reg XRSET;
input DREQ;
output reg XCS;                    // SCI
output reg XDCS;                    // SDI
output reg SI;
output reg SCK;                    // MP3
```

```
reg init = 0;
// 1M
wire clk_1M;
Divider #(.num(100)) clk_mp3(clk, clk_1M);

// IP
reg[11:0] addr;
wire [15: 0] get_data;
reg [15: 0] music_data;
blk_mem_gen_0 music_0 (
.clka(clk),
//.ena(1),
.addra({music_id, addr}),
.douta(get_data));

reg [3:0] condition = 0;
reg [63: 0] cmd = {32'h02000804, 32'h020B8080};


integer cnt = 0;
integer num = 0;

parameter DELAY = 1;
parameter PRE_CMD = 2;
parameter WRITE_CMD = 3;
parameter PRE_DATA = 4;
parameter WRITE_DATA = 5;
parameter DELAY_TIME = 500000;

always @(posedge clk_1M)
begin
    if(!rst || !init) begin
            init <= 1;
            XRSET <= 0;
            SCK <= 0;
            XCS <= 1;
            XDCS <= 1;
            condition <= DELAY;
            addr <= 0;
            cnt <= 0;
    end
    else begin
      if(start) begin
```

```
case (condition)
    DELAY: begin
        if(cnt == DELAY_TIME) begin
            cnt <= 0;
            condition <= PRE_CMD;
            XRSET <= 1;
        end
        else begin
            cnt <= cnt + 1;
        end
    end
    PRE_CMD: begin
        SCK <= 0;                                   // MP3
        if(num == 2) begin
            condition <= PRE_DATA;
            num <= 0;
        end
        else begin
            if(DREQ) begin
                cnt <= 0;
                condition <= WRITE_CMD;
            end
        end
    end
    WRITE_CMD: begin
        if(DREQ) begin
            if(clk) begin
                if(cnt == 32) begin
                    cnt <= 0;
                    XCS <= 1;
                    condition <= PRE_CMD;
                    num <= num + 1;
                end
                else begin
                    XCS <= 0;
                    SI <= cmd[63];
                    cmd <= {cmd[62: 0], cmd[63]};     // 鍛戒护绉讳綅
                    cnt <= cnt + 1;
                end
            end
            SCK <= ~SCK;
        end
    end
    PRE_DATA: begin
```

```
                if(DREQ) begin
                    SCK <= 0;
                    condition <= WRITE_DATA;
                    music_data <= get_data;
                    cnt <= 0;
                end
            end
            WRITE_DATA: begin
                if(SCK) begin
                    if(cnt == 16) begin
                        XDCS <= 1;
                        addr <= addr + 1;
                        cnt <= 0;
                        condition <= PRE_DATA;
                    end
                    else begin
                        XDCS <= 0;
                        SI <= music_data[15];
                        music_data <= {music_data[14:0],music_data[15]};
                        cnt <= cnt + 1;
                    end
                end
                 SCK <= ~SCK;
            end
            default: ;
        endcase
        end
      end
end

endmodule
```

2.top.v

```
`timescale 1ns / 1ps

module top(
    input           CLK100MHZ,          // nexys clk signal
    inout           TMP_SDA,            // i2c sda on temp sensor - bidirectional
    output          TMP_SCL,            // i2c scl on temp sensor
    output [6:0]    SEG,                // 7 segments of each display
    output [7:0]    AN,                 // 8 anodes of 8 displays
    output [15:0] LED                   // nexys leds = binary temp in deg C or deg F
    );
```

```
    wire w_200KHz;                  // 200kHz SCL
    wire [7:0] c_data;              // 8 bits of Celsius temperature data
    wire [7:0] f_data;              // 8 bits of Fahrenheit temperature data

    // Instantiate i2c master
    i2c_master i2cmaster(
        .clk_200KHz(w_200KHz),
        .temp_data(c_data),
        .SDA(TMP_SDA),
        .SCL(TMP_SCL)
    );

    // Instantiate 200kHz clock generator
    clkgen_200KHz clkgen(
        .clk_100MHz(CLK100MHZ),
        .clk_200KHz(w_200KHz)
    );

    seg7c segcontrol(
        .clk_100MHz(CLK100MHZ),
        .c_data(c_data),
        .f_data(f_data),
        .SEG(SEG),
        .AN(AN)
    );

    temp_converter tempconv(
        .c(c_data),
        .f(f_data)
    );

    // Set LED values for temperature data
    assign LED[15:8] = f_data;
    assign LED[7:0]  = c_data;

endmodule
```

3.temp_converter.v

```
`timescale 1ns / 1ps
module temp_converter(
    input [7:0] c,
```

```
    output [7:0] f
    );

    wire [15:0] p;
    wire [7:0] q;

    multiply_by_9 MULT9(.x(c), .y(p));
    divide_by_5 DIV5(.x(p), .y(q));
    add_32 ADD32(.x(q), .y(f));

endmodule
```

4. divide_by_5.v

```
module divide_by_5(
    input [15:0] x,
    output reg [7:0] y
    );

    always @(*) begin
        y = x / 5;
    end

endmodule
```

5. Divider.v

```
module Divider #(parameter num=100000000)
(
    input I_CLK,
    output reg O_CLK
);

reg [63:0] my_count;

initial
begin
O_CLK = 0;
my_count = 0;
end

always @ (posedge I_CLK)
```

```
begin
    my_count = my_count + 1;
    if(my_count >= num / 2) begin
        my_count = 0;
        O_CLK = ~O_CLK;
    end
end
endmodule
```

6. i2c_master.v

```
`timescale 1ns / 1ps

module i2c_master(
    input clk_200KHz,                   // i_clk
    inout SDA,                          // i2c standard interface signal
    output [7:0] temp_data,             // 8 bits binary representation of deg C
    output SCL                          // i2c standard interface signal - 10KHZ
    );

    wire SDA_dir;                       // SDA direction signal

    // *** GENERATE 10kHz SCL clock from 200kHz ***************************
    // 200 x 10^3 / 10 x 10^3 / 2 = 10
    reg [3:0] counter = 4'b0;    // count up to 9
    reg clk_reg = 1'b1;

    // Set value of i2c SCL signal to the sensor - 10kHz
    assign SCL = clk_reg;
    // ********************************************************************

    // Signal Declarations
    parameter [7:0] sensor_address_plus_read = 8'b1001_0111;// 0x97
    reg [7:0] tMSB = 8'b0;                                  // Temp data MSB
    reg [7:0] tLSB = 8'b0;                                  // Temp data LSB
    reg o_bit = 1'b1;                                       // output bit to SDA - starts
HIGH
    reg [11:0] count = 12'b0;                               // State Machine
Synchronizing Counter
    reg [7:0] temp_data_reg;                                // Temp data buffer register


    // State Declarations - need 28 states
    localparam [4:0] POWER_UP      = 5'h00,
```

```
                START          = 5'h01,
                SEND_ADDR6 = 5'h02,
                SEND_ADDR5 = 5'h03,
                SEND_ADDR4 = 5'h04,
                SEND_ADDR3 = 5'h05,
                SEND_ADDR2 = 5'h06,
                SEND_ADDR1 = 5'h07,
                SEND_ADDR0 = 5'h08,
                SEND_RW       = 5'h09,
                REC_ACK       = 5'h0A,
                REC_MSB7     = 5'h0B,
                REC_MSB6  = 5'h0C,
                REC_MSB5  = 5'h0D,
                REC_MSB4  = 5'h0E,
                REC_MSB3  = 5'h0F,
                REC_MSB2  = 5'h10,
                REC_MSB1  = 5'h11,
                REC_MSB0  = 5'h12,
                SEND_ACK      = 5'h13,
                REC_LSB7     = 5'h14,
                REC_LSB6   = 5'h15,
                REC_LSB5   = 5'h16,
                REC_LSB4   = 5'h17,
                REC_LSB3   = 5'h18,
                REC_LSB2   = 5'h19,
                REC_LSB1   = 5'h1A,
                REC_LSB0   = 5'h1B,
                NACK           = 5'h1C;

reg [4:0] state_reg = POWER_UP;                                    // state register

always @(posedge clk_200KHz) begin
        // Counters Logic
        if(counter == 9) begin
                counter <= 4'b0;
                clk_reg <= ~clk_reg;        // toggle reg
        end
        else
                counter <= counter + 1;

        count <= count + 1;

        // State Machine Logic
        case(state_reg)
```

```
POWER_UP       : begin
                    if(count == 12'd1999)
                        state_reg <= START;
end
START          : begin
                    if(count == 12'd2004)
                        o_bit <= 1'b0;              // send START condition
1/4 clock after SCL goes high
                    if(count == 12'd2013)
                        state_reg <= SEND_ADDR6;
end
SEND_ADDR6   : begin
                    o_bit <= sensor_address_plus_read[7];
                    if(count == 12'd2033)
                        state_reg <= SEND_ADDR5;
end
SEND_ADDR5   : begin
                    o_bit <= sensor_address_plus_read[6];
                    if(count == 12'd2053)
                        state_reg <= SEND_ADDR4;
end
SEND_ADDR4   : begin
                    o_bit <= sensor_address_plus_read[5];
                    if(count == 12'd2073)
                        state_reg <= SEND_ADDR3;
end
SEND_ADDR3   : begin
                    o_bit <= sensor_address_plus_read[4];
                    if(count == 12'd2093)
                        state_reg <= SEND_ADDR2;
end
SEND_ADDR2   : begin
                    o_bit <= sensor_address_plus_read[3];
                    if(count == 12'd2113)
                        state_reg <= SEND_ADDR1;
end
SEND_ADDR1   : begin
                    o_bit <= sensor_address_plus_read[2];
                    if(count == 12'd2133)
                        state_reg <= SEND_ADDR0;
end
SEND_ADDR0   : begin
                    o_bit <= sensor_address_plus_read[1];
                    if(count == 12'd2153)
```

```
                        state_reg <= SEND_RW;
end
SEND_RW         : begin
                    o_bit <= sensor_address_plus_read[0];
if(count == 12'd2169)
                        state_reg <= REC_ACK;
end
REC_ACK         : begin
                    if(count == 12'd2189)
                        state_reg <= REC_MSB7;
end
REC_MSB7        : begin
                    tMSB[7] <= i_bit;
                    if(count == 12'd2209)
                        state_reg <= REC_MSB6;

end
REC_MSB6        : begin
                    tMSB[6] <= i_bit;
                    if(count == 12'd2229)
                        state_reg <= REC_MSB5;

end
REC_MSB5        : begin
                    tMSB[5] <= i_bit;
                    if(count == 12'd2249)
                        state_reg <= REC_MSB4;

end
REC_MSB4        : begin
                    tMSB[4] <= i_bit;
                    if(count == 12'd2269)
                        state_reg <= REC_MSB3;

end
REC_MSB3        : begin
                    tMSB[3] <= i_bit;
                    if(count == 12'd2289)
                        state_reg <= REC_MSB2;

end
REC_MSB2        : begin
                    tMSB[2] <= i_bit;
                    if(count == 12'd2309)
```

```
                    state_reg <= REC_MSB1;

end
REC_MSB1        : begin
                    tMSB[1] <= i_bit;
                    if(count == 12'd2329)
                        state_reg <= REC_MSB0;

end
REC_MSB0        : begin
                    o_bit <= 1'b0;
                    tMSB[0] <= i_bit;
                    if(count == 12'd2349)
                        state_reg <= SEND_ACK;

end
SEND_ACK    : begin
                    if(count == 12'd2369)
                        state_reg <= REC_LSB7;
end
REC_LSB7        : begin
                    tLSB[7] <= i_bit;
                    if(count == 12'd2389)
                        state_reg <= REC_LSB6;
end
REC_LSB6        : begin
                    tLSB[6] <= i_bit;
                    if(count == 12'd2409)
                        state_reg <= REC_LSB5;
end
REC_LSB5        : begin
                    tLSB[5] <= i_bit;
                    if(count == 12'd2429)
                        state_reg <= REC_LSB4;
end
REC_LSB4        : begin
                    tLSB[4] <= i_bit;
                    if(count == 12'd2449)
                        state_reg <= REC_LSB3;
end
REC_LSB3        : begin
                    tLSB[3] <= i_bit;
                    if(count == 12'd2469)
                        state_reg <= REC_LSB2;
```

```
                end
                REC_LSB2        : begin
                                    tLSB[2] <= i_bit;
                                    if(count == 12'd2489)
                                        state_reg <= REC_LSB1;
                end
                REC_LSB1        : begin
                                    tLSB[1] <= i_bit;
                                    if(count == 12'd2509)
                                        state_reg <= REC_LSB0;
                end
                REC_LSB0        : begin
                                    o_bit <= 1'b1;
                                    tLSB[0] <= i_bit;
                                    if(count == 12'd2529)
                                        state_reg <= NACK;
                end
                NACK            : begin
                                    if(count == 12'd2559) begin
                                        count <= 12'd2000;
                                        state_reg <= START;
                                    end
                end
            endcase
    end

    // Buffer for temperature data
    always @(posedge clk_200KHz)
        if(state_reg == NACK)
            temp_data_reg <= { tMSB[6:0], tLSB[7] };


    // Control direction of SDA bidirectional inout signal
    assign SDA_dir = (state_reg == POWER_UP || state_reg == START || state_reg == SEND_ADDR6 || state_reg == SEND_ADDR5 ||
                      state_reg == SEND_ADDR4 || state_reg == SEND_ADDR3 || state_reg == SEND_ADDR2 || state_reg == SEND_ADDR1 ||
                      state_reg == SEND_ADDR0 || state_reg == SEND_RW || state_reg == SEND_ACK || state_reg == NACK) ? 1 : 0;
    // Set the value of SDA for output - from master to sensor
    assign SDA = SDA_dir ? o_bit : 1'bz;
    // Set value of input wire when SDA is used as an input - from sensor to master
    assign i_bit = SDA;
    // Outputted temperature data
```

```
    assign temp_data = temp_data_reg;

endmodule
```

7. multiply_by_9.v

```
module multiply_by_9(
    input [7:0] x,
    output reg [15:0] y
    );

    always @(*) begin
        y = x * 9;
    end
endmodule
```

8. seg7c.v

```
`timescale 1ns / 1ps
module seg7c(
    input clk_100MHz,                   // Nexys 4 DDR clock
    input [7:0] c_data,                 // Temp data from i2c master
    input [7:0] f_data,                 // Temp data from temp converter
    output reg [6:0] SEG,               // 7 Segments of Displays
    output reg [7:0] AN                 // 4 Anodes of 8 to display Temp C
    );

    // Binary to BCD conversion of temperature data
    wire [3:0] c_tens, c_ones;
    assign c_tens = c_data / 10;            // Tens value of C temp data
    assign c_ones = c_data % 10;            // Ones value of C temp data

    wire [3:0] f_tens, f_ones;
    assign f_tens = f_data / 10;            // Tens value of C temp data
    assign f_ones = f_data % 10;            // Ones value of C temp data

    // Parameters for segment patterns
    parameter ZERO  = 7'b000_0001;  // 0
    parameter ONE   = 7'b100_1111;  // 1
    parameter TWO   = 7'b001_0010;  // 2
    parameter THREE = 7'b000_0110;  // 3
    parameter FOUR  = 7'b100_1100;  // 4
    parameter FIVE  = 7'b010_0100;  // 5
```

```
    parameter SIX   = 7'b010_0000;   // 6
    parameter SEVEN = 7'b000_1111;   // 7
    parameter EIGHT = 7'b000_0000;   // 8
    parameter NINE  = 7'b000_0100;   // 9
    parameter DEG   = 7'b001_1100;   // degrees symbol
    parameter C     = 7'b011_0001;   // C
    parameter F     = 7'b011_1000;   // F

    // To select each digit in turn
    reg [2:0] anode_select;          // 2 bit counter for selecting each of 4 digits
    reg [16:0] anode_timer;          // counter for digit refresh

    // Logic for controlling digit select and digit timer
    always @(posedge clk_100MHz) begin
        // 1ms x 8 displays = 8ms refresh period
        if(anode_timer == 99_999) begin          // The period of 100MHz clock is 10ns (1/100,000,000 seconds)
            anode_timer <= 0;                    // 10ns x 100,000 = 1ms
            anode_select <=  anode_select + 1;
        end
        else
            anode_timer <=  anode_timer + 1;
    end

    // Logic for driving the 8 bit anode output based on digit select
    always @(anode_select) begin
        case(anode_select)
            3'o0 : AN = 8'b1111_1110;
            3'o1 : AN = 8'b1111_1101;
            3'o2 : AN = 8'b1111_1011;
            3'o3 : AN = 8'b1111_0111;
            3'o4 : AN = 8'b1110_1111;
            3'o5 : AN = 8'b1101_1111;
            3'o6 : AN = 8'b1011_1111;
            3'o7 : AN = 8'b0111_1111;
        endcase
    end

    always @*
        case(anode_select)
            3'o0 : SEG = C;      // Set to C for Celsuis

            3'o1 : SEG = DEG;    // Set to degrees symbol
```

```
3'o2 : begin            // C TEMPERATURE ONES DIGIT
                case(c_ones)
                    4'b0000 : SEG = ZERO;
                    4'b0001 : SEG = ONE;
                    4'b0010 : SEG = TWO;
                    4'b0011 : SEG = THREE;
                    4'b0100 : SEG = FOUR;
                    4'b0101 : SEG = FIVE;
                    4'b0110 : SEG = SIX;
                    4'b0111 : SEG = SEVEN;
                    4'b1000 : SEG = EIGHT;
                    4'b1001 : SEG = NINE;
                endcase
            end

3'o3 : begin            // C TEMPERATURE TENS DIGIT
                case(c_tens)
                    4'b0000 : SEG = ZERO;
                    4'b0001 : SEG = ONE;
                    4'b0010 : SEG = TWO;
                    4'b0011 : SEG = THREE;
                    4'b0100 : SEG = FOUR;
                    4'b0101 : SEG = FIVE;
                    4'b0110 : SEG = SIX;
                    4'b0111 : SEG = SEVEN;
                    4'b1000 : SEG = EIGHT;
                    4'b1001 : SEG = NINE;
                endcase
            end

3'o4 : SEG = F;       // Set to F for Fahrenheit

3'o5 : SEG = DEG;    // Set to degrees symbol

3'o6 : begin            // F TEMPERATURE ONES DIGIT
                case(f_ones)
                    4'b0000 : SEG = ZERO;
                    4'b0001 : SEG = ONE;
                    4'b0010 : SEG = TWO;
                    4'b0011 : SEG = THREE;
                    4'b0100 : SEG = FOUR;
                    4'b0101 : SEG = FIVE;
                    4'b0110 : SEG = SIX;
                    4'b0111 : SEG = SEVEN;
```

```
                        4'b1000 : SEG = EIGHT;
                        4'b1001 : SEG = NINE;
                    endcase
                end

            3'o7 : begin            // F TEMPERATURE TENS DIGIT
                    case(f_tens)
                        4'b0000 : SEG = ZERO;
                        4'b0001 : SEG = ONE;
                        4'b0010 : SEG = TWO;
                        4'b0011 : SEG = THREE;
                        4'b0100 : SEG = FOUR;
                        4'b0101 : SEG = FIVE;
                        4'b0110 : SEG = SIX;
                        4'b0111 : SEG = SEVEN;
                        4'b1000 : SEG = EIGHT;
                        4'b1001 : SEG = NINE;
                    endcase
                end
        endcase
endmodule
```

9.add_32.v

```
module add_32(
    input [7:0] x,
    output reg [7:0] y
    );

    always @(*) begin
        y = x + 32;
    end

endmodule
```

10. clkgen_200KHz.v

```
`timescale 1ns / 1ps
module clkgen_200KHz(
    input clk_100MHz, //input 100MHz clock signal
    output clk_200KHz //generated output 200KHz clock signal
    );

    // 100 x 10^6 / 200 x 10^3 / 2 = 250 <-- 8 bit counter
```

```
    reg [7:0] counter = 8'h00; //hexadecimal value - 8'b0 in binary - 0 value
    reg clk_reg = 1'b1; // 1 bit initialized as 1

    always @(posedge clk_100MHz) begin
        if(counter == 249) begin //if a cycle for the 200kHz clock is completed
            counter <= 8'h00; //reset counter
            clk_reg <= ~clk_reg; //inverting to high/low logic level for output
        end
        else
            counter <= counter + 1; //incrementing by 1
    end

    assign clk_200KHz = clk_reg; //assign the state of the output signal (1 or 0)

endmodule
```

(二) VGA+MP3+Keyboard 游戏系统

1.top.v

```
module
top(clk,rst,start,over,color_r,color_g,color_b,hs,vs,move_top,move_bottom,move_left,move_right,
change,key_clk,key_data,DREQ,XCS,XDCS,SCK,SI,XRESET,oData,law);
input clk;                          // 系统时钟 100M
input rst;                          // 低电平有效

// 游戏相关
input start;                        // 游戏开始
output over;                        // 游戏结束

// VGA 相关
output [3:0] color_r;               // 红色分量
output [3:0] color_g;               // 绿色分量
output [3:0] color_b;               // 蓝色分量
output hs;                          // 行同步
output vs;                          // 场同步
output move_top;                    // 当顶部木板移动时亮起
output move_bottom;                 // 当底部木板移动时亮起
output move_left;                   // 当左侧木板移动时亮起
output move_right;                  // 当右侧木板移动时亮起
output change;                      // 木块被反弹时亮起

//键盘相关
```

```
input     key_clk;                    // 键盘时钟
input     key_data;                   // 键盘输入数据
wire [8:0] keys;

// mp3 相关
input       DREQ;               //数据请求，高电平时可传输数据
output      XCS;                // SCI 传输读写指令
output      XDCS;               // SDI 传输数据
output      SCK;                // 时钟
output      SI;                 // 传入 mp3
output      XRESET;             // 硬件复位，低电平有效

// 数码管相关
output [6:0] oData;           // 显示时间
output [7:0] law;             // 片选数码管

wire clk_25M;                   // 25M 时钟
divider #(.num(4)) clk_vga(clk,clk_25M);
wire clk_1000;                  // 1000 时钟
divider #(.num(100000)) clk_display(clk,clk_1000);

// VGA
vga vga_inst(
      .clk_25M(clk_25M),
      .rst(rst),
      .start(start),
      .over(over),
      .key_ascii(keys),
      .color_r(color_r),
      .color_g(color_g),
      .color_b(color_b),
      .hs(hs),
      .vs(vs),
      .move_top(move_top),
      .move_bottom(move_bottom),
      .move_left(move_left),
      .move_right(move_right),
      .change(change)
);

//键盘
keyboard keyboard_inst(
      .clk(clk),
      .rst(1),
```

```
        .key_clk(key_clk),
        .key_data(key_data),
        .key_ascii(keys)
);

// mp3
mp3 mp3_inst(
        .clk(clk),
        .DREQ(DREQ),
        .rst(rst),
        .music_id(0),
        .XDCS(XDCS),
        .XCS(XCS),
        .XRSET(XRESET),
        .SI(SI),
        .SCK(SCK),
        .start(start)
        );

//  数码管
display display_inst(
        .clk_1000(clk_1000),
        .rst(rst),
        .start(start),
        .over(over),
        .oData(oData),
        .law(law)
        );

endmodule
```

2.mp3.v

```
module mp3(clk,rst,start,music_id,XRSET,DREQ,XCS,XDCS,SI,SCK);
input clk;
input rst;
input start;
input [2:0] music_id;
// mp3
output reg XRSET;
input DREQ;
output reg XCS;
output reg XDCS;
output reg SI;
```

```
output reg SCK;

reg init = 0;
wire clk_1M;
divider #(.num(100)) clk_mp3(clk, clk_1M);

reg[11:0] addr;
wire [15: 0] get_data;
reg [15: 0] music_data;
blk_mem_gen_0 music_0 (
.clka(clk),
//.ena(1),
.addra({music_id, addr}),
.douta(get_data));

reg [3:0] condition = 0;
reg [63: 0] cmd = {32'h02000804, 32'h020B8080};

// 变量
integer cnt = 0;
integer num = 0;


parameter DELAY = 1;
parameter PRE_CMD = 2;
parameter WRITE_CMD = 3;
parameter PRE_DATA = 4;
parameter WRITE_DATA = 5;
parameter DELAY_TIME = 500000;

always @(posedge clk_1M)
begin
    if(!rst || !init) begin
            init <= 1;
            XRSET <= 0;
            SCK <= 0;
            XCS <= 1;
            XDCS <= 1;
            condition <= DELAY;
            addr <= 0;
            cnt <= 0;
    end
    else begin
      if(start) begin
```

```
case (condition)
    DELAY: begin
        if(cnt == DELAY_TIME) begin
            cnt <= 0;
            condition <= PRE_CMD;
            XRSET <= 1;
        end
        else begin
            cnt <= cnt + 1;
        end
    end
    PRE_CMD: begin
        SCK <= 0;
        if(num == 2) begin
            condition <= PRE_DATA;
            num <= 0;
        end
        else begin
            if(DREQ) begin
                cnt <= 0;
                condition <= WRITE_CMD;
            end
        end
    end
    WRITE_CMD: begin
        if(DREQ) begin
            if(clk) begin
                if(cnt == 32) begin
                    cnt <= 0;
                    XCS <= 1;
                    condition <= PRE_CMD;
                    num <= num + 1;
                end
                else begin
                    XCS <= 0;
                    SI <= cmd[63];
                    cmd <= {cmd[62: 0], cmd[63]};
                    cnt <= cnt + 1;
                end
            end
            SCK <= ~SCK;
        end
    end
    PRE_DATA: begin
```

```
                if(DREQ) begin
                    SCK <= 0;
                    condition <= WRITE_DATA;
                    music_data <= get_data;
                    cnt <= 0;
                end
            end
            WRITE_DATA: begin
                if(SCK) begin
                    if(cnt == 16) begin
                        XDCS <= 1;
                        addr <= addr + 1;
                        cnt <= 0;
                        condition <= PRE_DATA;
                    end
                    else begin
                        XDCS <= 0;
                        SI <= music_data[15];
                        music_data <= {music_data[14:0],music_data[15]};
                        cnt <= cnt + 1;
                    end
                end
                 SCK <= ~SCK;
            end
            default: ;
        endcase
        end
      end
end

endmodule
```

3.keyboard.v

```
module keyboard (clk,rst,key_clk,key_data,key_ascii);
input clk;                          //系统时钟
input rst;                         //系统复位，低有效
input key_clk;                     //PS2 键盘时钟
input key_data;                       //PS2 键盘数据输入
output reg [7:0] key_ascii;       //按下键盘的 ASCII 码(特定键有效，其余均为 0)

reg [3:0] now_bit = 0;              //记录已获取的 bit 数
reg [7:0] store = 0;                //存储键盘数据输入
reg [7:0] key_info = 0;       //存储键盘数据信息
```

```
reg break = 0;                    //断码标志

reg   key_clk_new = 1'b1, key_clk_old = 1'b1;
reg   key_data_new = 1'b1, key_data_old = 1'b1;

//  对键盘时钟数据信号进行延时锁存
always @ (posedge clk or negedge rst)
begin
    if(!rst)
    begin
        key_clk_new <= 1'b1;
        key_clk_old <= 1'b1;
        key_data_new <= 1'b1;
        key_data_old <= 1'b1;
    end
    else
    begin
        //交替存储实现延时锁存
        key_clk_new <= key_clk;
        key_clk_old <= key_clk_new;
        key_data_new <= key_data;
        key_data_old <= key_data_new;
    end
end

wire key_clk_neg = key_clk_old & (~key_clk_new);

//PS2 键盘时钟信号下降沿读取数据
always @ (posedge clk or negedge rst)
begin
    if(!rst)
    begin
        now_bit <= 4'd0;
        store <= 8'd0;
    end
    else
    begin
        if(key_clk_neg)
        begin
            if(now_bit >= 4'd10) now_bit <= 4'd0;
            else now_bit <= now_bit + 1'b1;
            case (now_bit)
                0: ;                        //起始位
                // bit 数据位
```

```
                    1: store[0] <= key_data_old;
                    2: store[1] <= key_data_old;
                    3: store[2] <= key_data_old;
                    4: store[3] <= key_data_old;
                    5: store[4] <= key_data_old;
                    6: store[5] <= key_data_old;
                    7: store[6] <= key_data_old;
                    8: store[7] <= key_data_old;
                    9: ;                                //校验位
                    10:;                                //结束位
                    default: ;
                endcase
            end
        end
end

//判断断码
always @ (posedge clk or negedge rst)
begin
    if(!rst)
    begin
        break <= 0;             //重置标记
        key_info <= 0;
    end
    else
    begin
        if(now_bit == 4'd10 && key_clk_neg)                 //已经读取完整段数据，且处
于 PS2 键盘时钟信号下降沿
        begin
            if(store == 8'hf0) break <= 1;                  //断码（8'hf0）表示按键松开，下一
个数据为断码，设置断码标示为 1
            else if(!break) key_info <= store;      //不为断码
            else                                            //该段数据是断码，舍弃
            begin
                break <= 0;             //重置标记
                key_info <= 0;
            end
        end
    end
end


//将从键盘得到的数据转换为 ASCII 码
always @ (key_info)
```

```
begin
    case (key_info)
        8'h75: key_ascii <= 8'd1;           //上箭头
        8'h72: key_ascii <= 8'd2;           //下箭头
        8'h6b: key_ascii <= 8'd3;           //左箭头
        8'h74: key_ascii <= 8'd4;           //右箭头
        8'h1d: key_ascii <= 8'd5;       // w
        8'h1b: key_ascii <= 8'd6;       // s
        8'h1c: key_ascii <= 8'd7;       // a
        8'h23: key_ascii <= 8'd8;       // d
        default: key_ascii <= 0;
    endcase
end

endmodule
```

4.display.v

```
module BCD (game_time,data1,data2,data3,data4,data5,data6,data7,data8);
input [31:0] game_time;
output [3:0] data1;
output [3:0] data2;
output [3:0] data3;
output [3:0] data4;
output [3:0] data5;
output [3:0] data6;
output [3:0] data7;
output [3:0] data8;

reg [31:0]    bin;
reg [31:0]    tmp;
reg [31:0]    bcd;

always @(game_time) begin
    bin = game_time;
    tmp = 0;
    repeat (31)
    begin
        tmp[0] = bin[31];
        if (tmp[3:0] > 4) tmp[3:0] = tmp[3:0] + 4'd3;
        else tmp[3:0] = tmp[3:0];
        if (tmp[7:4] > 4) tmp[7:4] = tmp[7:4] + 4'd3;
        else tmp[7:4] = tmp[7:4];
        if (tmp[11:8] > 4) tmp[11:8] = tmp[11:8] + 4'd3;
```

```
            else tmp[11:8] = tmp[11:8];
            if (tmp[15:12] > 4) tmp[15:12] = tmp[15:12] + 4'd3;
            else tmp[15:12] = tmp[15:12];
            if (tmp[19:16] > 4) tmp[19:16] = tmp[19:16] + 4'd3;
            else tmp[19:16] = tmp[19:16];
            if (tmp[23:20] > 4) tmp[23:20] = tmp[23:20] + 4'd3;
            else tmp[23:20] = tmp[23:20];
            if (tmp[27:24] > 4) tmp[27:24] = tmp[27:24] + 4'd3;
            else tmp[27:24] = tmp[27:24];
            if (tmp[31:28] > 4) tmp[31:28] = tmp[31:28] + 4'd3;
            else tmp[31:28] = tmp[31:28];
            tmp = tmp << 1;
            bin = bin << 1;
        end
        tmp[0] = bin[31];
        bcd = tmp;
end
assign data1 = bcd[3:0];
assign data2 = bcd[7:4];
assign data3 = bcd[11:8];
assign data4 = bcd[15:12];
assign data5 = bcd[19:16];
assign data6 = bcd[23:20];
assign data7 = bcd[27:24];
assign data8 = bcd[31:28];
endmodule


module display (clk_1000,rst,start,over,oData,law);
input clk_1000;
input rst;
input start;                    //    Þ   ’
input over;                     //    Þ
output reg [6:0] oData;
output reg [7:0] law;                    //         Tw   ’

reg [31:0] game_time = 0;
integer cnt = 0;
integer time_update_cnt = 0;             // ﬁ       ¼
wire [3:0] store [7:0];                  //    泏 ɥ     Ƶ ÿhλ     ̥
//  ת    Ï BCD
BCD bcd(
        .game_time(game_time),
        .data1(store[0]),
```

```
        .data2(store[1]),
        .data3(store[2]),
        .data4(store[3]),
        .data5(store[4]),
        .data6(store[5]),
        .data7(store[6]),
        .data8(store[7])
        );

// Tw
always @(posedge clk_1000 or negedge rst) begin
        if(!rst) begin
                game_time <= 0;
        end
        else begin
                if(start) begin
                        if(over == 0) begin
                                if(time_update_cnt == 1000) begin
                                        game_time <= game_time + 1;
                                        time_update_cnt <= 0;
                                end
                                else time_update_cnt <= time_update_cnt + 1;
                        end
                        else begin
                                game_time <= game_time;
                                time_update_cnt <= 0;
                        end
                end
                else game_time <= 0;
                if(cnt == 8) cnt <= 0;
                else cnt <= cnt + 1;
                law <= 8'b1111_1111;
                law[cnt] <= 0;                                  //Tw
                case (store[cnt])
                        4'b0000: oData <= 7'b1000000;
                        4'b0001: oData <= 7'b1111001;
                        4'b0010: oData <= 7'b0100100;
                        4'b0011: oData <= 7'b0110000;
                        4'b0100: oData <= 7'b0011001;
                        4'b0101: oData <= 7'b0010010;
                        4'b0110: oData <= 7'b0000010;
                        4'b0111: oData <= 7'b1111000;
                        4'b1000: oData <= 7'b0000000;
                        4'b1001: oData <= 7'b0010000;
```

```
                default: oData <= 7'b1111111;
            endcase
        end
end

endmodule
```

5.divider.v

```
module divider #(parameter num=100000000)
(
    input I_CLK,
    output reg O_CLK
);

reg [63:0] my_count;

initial
begin
O_CLK = 0;
my_count = 0;
end

always @ (posedge I_CLK)
begin
    my_count = my_count + 1;
    if(my_count >= num / 2) begin
        my_count = 0;
        O_CLK = ~O_CLK;
    end
end
endmodule
```

6.vga.v

```
module vga
(clk_25M,rst,start,over,key_ascii,color_r,color_g,color_b,hs,vs,move_top,move_bottom,move_left
,move_right,change);
input clk_25M;                  // 25M 时钟
input rst;                      // 低电平有效
input start;                    // 游戏开始
output reg over;                // 游戏结束
input [7:0] key_ascii;          // 键盘按键
output reg [3:0] color_r;       // 红色分量
```

```
output reg [3:0] color_g;       // 绿色分量
output reg [3:0] color_b;       // 蓝色分量
output hs;                      // 行同步
output vs;                      // 场同步

output reg move_top;            // 当顶部木板移动时亮起
output reg move_bottom;         // 当底部木板移动时亮起
output reg move_left;           // 当左侧木板移动时亮起
output reg move_right;          // 当右侧木板移动时亮起
output reg change;              // 木块被反弹时亮起

// 常量参数
parameter HS_SYNC = 96;         // 同步区域
parameter HS_BACK = 48;         // 后沿区域
parameter HS_ACTIVE = 640;      // 有效显示区域
parameter HS_FRONT = 16;        // 前沿区域

parameter VS_SYNC = 2;          // 同步区域
parameter VS_BACK = 33;         // 后沿区域
parameter VS_ACTIVE = 480;      // 有效显示区域
parameter VS_FRONT = 10;        // 前沿区域

parameter TOTAL_ROWS = 525;     // 总行数
parameter TOTAL_COLS = 800;     // 总列数

wire law;                       // 处于有效显示区域标志，高电平有效
reg [11:0] h_cnt = 0;           // 列计数器
reg [11:0] v_cnt = 0;           // 行计数器

// 接入输出
assign hs = (h_cnt < HS_SYNC)? 0 : 1;
assign vs = (v_cnt < VS_SYNC)? 0 : 1;
assign law = (h_cnt >= (HS_SYNC + HS_BACK)) &&
             (h_cnt <= (HS_SYNC + HS_BACK + HS_ACTIVE)) &&
             (v_cnt >= (VS_SYNC + VS_BACK)) &&
             (v_cnt <= (VS_SYNC + VS_BACK + VS_ACTIVE));

// 移动相关参数
integer length_block = 40;      // 木块的长度
integer width_block = 40;       // 木块的宽度
integer X_block = 50;           // 木块中心的的 X 坐标
integer Y_block = 50;           // 木块中心的的 Y 坐标
integer length_board = 100;     // 木板的长度
integer width_board = 20;       // 木板的宽度
```

```
integer X_board_bottom = 320;                    // 底部木板中心的的 X 坐标
integer Y_board_bottom = 450;                    // 底部木板中心的的 Y 坐标
integer X_board_top = 320;                       // 顶部木板中心的的 X 坐标
integer Y_board_top = 10;                        // 顶部木板中心的的 Y 坐标
integer X_board_left = 10;                       // 左侧木板中心的的 X 坐标
integer Y_board_left = 240;                      // 左侧木板中心的的 Y 坐标
integer X_board_right = 630;                     // 右侧木板中心的的 X 坐标
integer Y_board_right = 240;                     // 右侧木板中心的的 Y 坐标
integer direction = 1;                           // 木块移动方向


// 行时序
always @(posedge clk_25M or negedge rst) begin
    if(!rst) begin
        h_cnt <= 0;
    end
    else if(start) begin
        if(h_cnt == TOTAL_COLS - 1) begin        // 显示完一行，重置
            h_cnt <= 0;
        end
        else begin
            h_cnt <= h_cnt + 1;
        end
    end
end

// 场时序
always @(posedge clk_25M or negedge rst) begin
    if(!rst) begin
        v_cnt <= 0;
        over <= 0;
        X_block <= 50;
        Y_block <= 50;
        X_board_bottom <= 320;
        Y_board_bottom <= 450;
        X_board_top <= 320;
        Y_board_top <= 10;
        X_board_left <= 10;
        Y_board_left <= 240;
        X_board_right <= 630;
        Y_board_right <= 240;
        direction <= 1;
        move_top <= 0;
        move_bottom <= 0;
```

```
            move_left <= 0;
            move_right <= 0;
            change <= 0;
        end
        else if(start) begin
            if(v_cnt == TOTAL_ROWS - 1) begin                // 显示完整个屏幕，重置更新
                v_cnt <= 0;
                // 重置灯
                move_top <= 0;
                move_bottom <= 0;
                move_left <= 0;
                move_right <= 0;
                change <= 0;
                // 进行木块的位置更新
               case (direction)
                    1: begin
                    X_block <= X_block + 1;
                    Y_block <= Y_block + 1;
                    end
                    2: begin
                    X_block <= X_block + 1;
                    Y_block <= Y_block - 1;
                    end
                    3: begin
                    X_block <= X_block - 1;
                    Y_block <= Y_block - 1;
                    end
                    4: begin
                    X_block <= X_block - 1;
                    Y_block <= Y_block + 1;
                    end
                    default: ;
                endcase
                // 木板位置更新
                if(key_ascii == 8'd3) begin                    // 底部左移
                    if(Y_block - Y_board_bottom <= (width_board + width_block) / 2 &&
                    Y_block - Y_board_bottom >= -(width_board + width_block) / 2 &&
                    X_board_bottom - 2 - X_block <= (length_board + length_block) / 2 &&
                    X_board_bottom - 2 - X_block > 0) begin
                        X_board_bottom <= X_board_bottom;
                    end
                    else if(X_board_bottom > length_board / 2 + 30) X_board_bottom <=
X_board_bottom - 2;
                    else ;
```

```
                move_bottom <= 1;
            end
            if(key_ascii == 8'd4) begin                    // 底部右移
                if(Y_block - Y_board_bottom <= (width_board + width_block) / 2 &&
                Y_block - Y_board_bottom >= -(width_board + width_block) / 2 &&
                X_block - X_board_bottom - 2 <= (length_board + length_block) / 2 &&
                X_block - X_board_bottom - 2 > 0) begin
                    X_board_bottom <= X_board_bottom;
                end
                else if(X_board_bottom < 640 - length_board / 2 - 30) X_board_bottom <=
X_board_bottom + 2;
                else ;
                move_bottom <= 1;
            end
            if(key_ascii == 8'd7) begin                    // 顶部左移
                if(Y_block - Y_board_top <= (width_board + width_block) / 2 &&
                Y_block - Y_board_top >= -(width_board + width_block) / 2 &&
                X_board_top - 2 - X_block <= (length_board + length_block) / 2 &&
                X_board_top - 2 - X_block > 0) begin
                    X_board_top <= X_board_top;
                end
                else if(X_board_top > length_board / 2 + 30) X_board_top <= X_board_top -
2;
                else ;
                move_top <= 1;
            end
            if(key_ascii == 8'd8) begin                    // 顶部右移
                if(Y_block - Y_board_top <= (width_board + width_block) / 2 &&
                Y_block - Y_board_top >= -(width_board + width_block) / 2 &&
                X_block - X_board_top - 2 <= (length_board + length_block) / 2 &&
                X_block - X_board_top - 2 > 0) begin
                    X_board_top <= X_board_top;
                end
                else if(X_board_top < 640 - length_board / 2 - 30) X_board_top <=
X_board_top + 2;
                else ;
                move_top <= 1;
            end
            if(key_ascii == 8'd1) begin                    // 左部上移
                if(X_block - X_board_left <= (width_board + width_block) / 2 &&
                X_block - X_board_left >= -(width_board + width_block) / 2 &&
                Y_board_left - 2 - Y_block <= (length_board + length_block) / 2 &&
                Y_board_left - 2 - Y_block > 0) begin
                    Y_board_left <= Y_board_left;
```

```
                end
                else if(Y_board_left > length_board / 2 + 10) Y_board_left <= Y_board_left -
2;
                else ;
                move_left <= 1;
            end
            if(key_ascii == 8'd2) begin                    // 左部下移
                if(X_block - X_board_left <= (width_board + width_block) / 2 &&
                X_block - X_board_left >= -(width_board + width_block) / 2 &&
                Y_block - Y_board_left - 2 <= (length_board + length_block) / 2 &&
                Y_block - Y_board_left - 2 > 0) begin
                    Y_board_left <= Y_board_left;
                end
                else if(Y_board_left < 460 - length_board / 2 - 10) Y_board_left <=
Y_board_left + 2;
                else ;
                move_left <= 1;
            end
            if(key_ascii == 8'd5) begin                    // 右部上移
                if(X_block - X_board_right <= (width_board + width_block) / 2 &&
                X_block - X_board_right >= -(width_board + width_block) / 2 &&
                Y_board_right - 2 - Y_block <= (length_board + length_block) / 2 &&
                Y_board_right - 2 - Y_block > 0) begin
                    Y_board_right <= Y_board_right;
                end
                else if(Y_board_right > length_board / 2 + 10) Y_board_right <=
Y_board_right - 2;
                else ;
                move_right <= 1;
            end
            if(key_ascii == 8'd6) begin                    // 右部下移
                if(X_block - X_board_right <= (width_board + width_block) / 2 &&
                X_block - X_board_right >= -(width_board + width_block) / 2 &&
                Y_block - Y_board_right - 2 <= (length_board + length_block) / 2 &&
                Y_block - Y_board_right - 2 > 0) begin
                    Y_board_right <= Y_board_right;
                end
                else if(Y_board_right < 460 - length_board / 2 - 10) Y_board_right <=
Y_board_right + 2;
                else ;
                move_right <= 1;
            end

            // 木块与木板接触，反弹
```

```
// 底部木板
if(Y_board_bottom - Y_block == (width_board + width_block) / 2 &&
X_board_bottom - X_block <= (length_board + length_block) / 2 &&
X_board_bottom - X_block >= -(length_board + length_block) / 2) begin
    if(direction == 4) direction <= 3;
    else if(direction == 1) direction <= 2;
    else ;
    change <= 1;
end
if(X_block - X_board_bottom == (length_board + length_block) / 2 &&
Y_block - Y_board_bottom <= (width_board + width_block) / 2 &&
Y_block - Y_board_bottom >= -(width_board + width_block) / 2) begin
    if(direction == 4) direction <= 1;
    else if(direction == 3) direction <= 2;
    else ;
    change <= 1;
end
if(X_board_bottom - X_block == (length_board + length_block) / 2 &&
Y_block - Y_board_bottom <= (width_board + width_block) / 2 &&
Y_block - Y_board_bottom >= -(width_board + width_block) / 2) begin
    if(direction == 2) direction <= 3;
    else if(direction == 1) direction <= 4;
    else ;
    change <= 1;
end
// 顶部木板
if(Y_block - Y_board_top == (width_board + width_block) / 2 &&
X_board_top - X_block <= (length_board + length_block) / 2 &&
X_board_top - X_block >= -(length_board + length_block) / 2) begin
    if(direction == 2) direction <= 1;
    else if(direction == 3) direction <= 4;
    else ;
    change <= 1;
end
if(X_block - X_board_top == (length_board + length_block) / 2 &&
Y_block - Y_board_top <= (width_board + width_block) / 2 &&
Y_block - Y_board_top >= -(width_board + width_block) / 2) begin
    if(direction == 4) direction <= 1;
    else if(direction == 3) direction <= 2;
    else ;
    change <= 1;
end
if(X_board_top - X_block == (length_board + length_block) / 2 &&
Y_block - Y_board_top <= (width_board + width_block) / 2 &&
```

```
Y_block - Y_board_top >= -(width_board + width_block) / 2) begin
        if(direction == 2) direction <= 3;
        else if(direction == 1) direction <= 4;
        else ;
        change <= 1;
end
// 左侧木板
if(X_block - X_board_left == (width_board + width_block) / 2 &&
Y_board_left - Y_block <= (length_board + length_block) / 2 &&
Y_board_left - Y_block >= -(length_board + length_block) / 2) begin
        if(direction == 4) direction <= 1;
        else if(direction == 3) direction <= 2;
        else ;
        change <= 1;
end
if(Y_board_left - Y_block == (length_board + length_block) / 2 &&
X_block - X_board_left <= (width_board + width_block) / 2 &&
X_block - X_board_left >= -(width_board + width_block) / 2) begin
        if(direction == 4) direction <= 3;
        else if(direction == 1) direction <= 2;
        else ;
        change <= 1;
end
if(Y_block - Y_board_left == (length_board + length_block) / 2 &&
X_block - X_board_left <= (width_board + width_block) / 2 &&
X_block - X_board_left >= -(width_board + width_block) / 2) begin
        if(direction == 2) direction <= 1;
        else if(direction == 3) direction <= 4;
        else ;
        change <= 1;
end
// 右侧木板
if(X_board_right - X_block == (width_board + width_block) / 2 &&
Y_board_right - Y_block <= (length_board + length_block) / 2 &&
Y_board_right - Y_block >= -(length_board + length_block) / 2) begin
        if(direction == 2) direction <= 3;
        else if(direction == 1) direction <= 4;
        else ;
        change <= 1;
end
if(Y_board_right - Y_block == (length_board + length_block) / 2 &&
X_block - X_board_right <= (width_board + width_block) / 2 &&
X_block - X_board_right >= -(width_board + width_block) / 2) begin
        if(direction == 4) direction <= 3;
```

```
                else if(direction == 1) direction <= 2;
                else ;
                change <= 1;
            end
            if(Y_block - Y_board_right == (length_board + length_block) / 2 &&
            X_block - X_board_right <= (width_board + width_block) / 2 &&
            X_block - X_board_right >= -(width_board + width_block) / 2) begin
                if(direction == 2) direction <= 1;
                else if(direction == 3) direction <= 4;
                else ;
                change <= 1;
            end

            // 判定结束
            if(X_block <= 0 || X_block >= 640 || Y_block <= 0 || Y_block >= 460) over <= 1;
        end
        else if(h_cnt == TOTAL_COLS - 1) begin          // 显示完一行，进行下一行显示
            v_cnt <= v_cnt + 1;
        end
        else begin
            v_cnt <= v_cnt;
        end
    end
    else ;
end

// 显示内容
always @(posedge clk_25M or negedge rst) begin
    if(!rst) begin
        color_r <= 4'b0000;
        color_g <= 4'b0000;
        color_b <= 4'b0000;
    end
    else if(law && start && over == 0) begin                    // 在有效显
示区域，显示内容
        // 显示木块
        if((h_cnt >= HS_SYNC + HS_BACK + X_block - length_block / 2) &&
        (h_cnt <= HS_SYNC + HS_BACK + X_block + length_block / 2) &&
        (v_cnt >= VS_SYNC + VS_BACK + Y_block - width_block / 2) &&
        (v_cnt <= VS_SYNC + VS_BACK + Y_block + width_block / 2)) begin
            // 黑色
            color_r <= 4'b0000;
            color_g <= 4'b0000;
            color_b <= 4'b0000;
```

```
end
// 显示木板
// 底部
else if((h_cnt >= HS_SYNC + HS_BACK + X_board_bottom - length_board / 2) &&
(h_cnt <= HS_SYNC + HS_BACK + X_board_bottom + length_board / 2) &&
(v_cnt >= VS_SYNC + VS_BACK + Y_board_bottom - width_board / 2) &&
(v_cnt <= VS_SYNC + VS_BACK + Y_board_bottom + width_board / 2)) begin
    // 蓝色
    color_r <= 4'b0000;
    color_g <= 4'b0000;
    color_b <= 4'b1111;
end
// 顶部
else if((h_cnt >= HS_SYNC + HS_BACK + X_board_top - length_board / 2) &&
(h_cnt <= HS_SYNC + HS_BACK + X_board_top + length_board / 2) &&
(v_cnt >= VS_SYNC + VS_BACK + Y_board_top - width_board / 2) &&
(v_cnt <= VS_SYNC + VS_BACK + Y_board_top + width_board / 2)) begin
    // 蓝色
    color_r <= 4'b0000;
    color_g <= 4'b0000;
    color_b <= 4'b1111;
end
// 左侧
else if((h_cnt >= HS_SYNC + HS_BACK + X_board_left - width_board / 2) &&
(h_cnt <= HS_SYNC + HS_BACK + X_board_left + width_board / 2) &&
(v_cnt >= VS_SYNC + VS_BACK + Y_board_left - length_board / 2) &&
(v_cnt <= VS_SYNC + VS_BACK + Y_board_left + length_board / 2)) begin
    // 蓝色
    color_r <= 4'b0000;
    color_g <= 4'b0000;
    color_b <= 4'b1111;
end
// 右侧
else if((h_cnt >= HS_SYNC + HS_BACK + X_board_right - width_board / 2) &&
(h_cnt <= HS_SYNC + HS_BACK + X_board_right + width_board / 2) &&
(v_cnt >= VS_SYNC + VS_BACK + Y_board_right - length_board / 2) &&
(v_cnt <= VS_SYNC + VS_BACK + Y_board_right + length_board / 2)) begin
    // 蓝色
    color_r <= 4'b0000;
    color_g <= 4'b0000;
    color_b <= 4'b1111;
end
// 显示背景
else begin
```

```
                // 白色
                color_r <= 4'b1111;
                color_g <= 4'b1111;
                color_b <= 4'b1111;
            end
        end
        else if(law && start && over) begin
            // 白色
            color_r <= 4'b1111;
            color_g <= 4'b1111;
            color_b <= 4'b1111;
        end
        else begin                                          // 未在有效显示区域则重置颜色
            color_r <= 4'b0000;
            color_g <= 4'b0000;
            color_b <= 4'b0000;
        end
    end

    endmodule
```