**同济大学计算机系**

**数字逻辑课程综合实验报告**

****

**学 号 1952339**

**姓 名 张馨月**

**专 业 计算机科学与技术**

**授课老师 郭玉臣**

1. 实验内容

利用NEXYS4 DDR开发板、VB1003B-MP3-Board、Blue-Tooth-Slave-UART-Board 和 Sound Sensor完成一款能够用蓝牙发送指令和接受反馈、控制播放音乐、调节音量大小的MP3播放器的制作。

七段数码管上前四位显示24小时制数字时钟，后四位显示当前播放曲目的相关信息。

蓝牙串口会在总复位开关下降沿返回最后一次接收的信息。

板上16个LED灯显示音量大小，全亮为最大音量，全灭指示音量设置错误

蓝牙指令：

01：播放前一首歌

02：播放后一首歌

03：进入音量增大调节模式，按板上P17来增大音量

04：进入音量减小调节模式，按板上P17来减小音量

05：单曲循环歌曲0

06：单曲循环歌曲1

07：单曲循环歌曲2

08：单曲循环歌曲3

09：单曲循环歌曲4

0a：单曲循环歌曲5

0b：单曲循环歌曲6

所有操作完成后都需要进行复位才能把指令应用到MP3中

1. MP3播放器数字系统总框图

（按由顶向下方法进行子系统的划分，给出包含各子系统相互关系及控制信号的总框图，并对各子系统功能及实现进行概述。具体可参考教材183页的相关描述方法。）

1. 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。）

1. 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，并列出各模块建模的verilog代码）

4.0 时钟分频

`timescale 1ns / 1ps

module Divider #(parameter T = 100000)(

input clk\_in,

output reg clk\_out = 0

);

integer tot = 0;

always@(posedge clk\_in) begin

if(tot < T/2-1) begin

tot <= tot + 1;

end

else begin

tot <= 0;

clk\_out = ~clk\_out;

end

end

endmodule

对系统时钟进行参数分频，默认为100000。

4.1蓝牙系统

4.1.1 波特率脉冲产生----clkGenerator模块

`timescale 1ns / 1ps

module clkGenerator(

input clk, //系统时钟输入（未分频）

input rst, //复位信号

input [15:0]BaudRate, //设定的通信波特率对应时钟换算后的脉冲数

output Tick //产生的对应波特率的脉冲

);

reg [15:0] baudratereg; //系统脉冲计数

always@(posedge clk or negedge rst)begin

if(!rst) baudratereg <= 16'b1; //复位重新计数

else if(Tick) baudratereg <= 16'b1;

else baudratereg <= baudratereg + 1'b1;

end

assign Tick = (baudratereg == BaudRate);

endmodule

对系统时钟脉冲进行对应波特率的转换，产生可用的波特率脉冲，用作接受和发送数据；

板子的频率是100MHz，此时波特率设为9600Hz，换算成波特率的脉冲宽度大概为104us，而板子时钟为10ns，按照蓝牙每16次发送一个bit内容的机制，我们需要产生的对应波特率脉冲为104000ns/16/10ns = 650个脉冲，所以在蓝牙顶层模块（bluetooth）中我将波特率脉冲数设置为650个。

4.1.2 蓝牙数据接收----UartReciever模块

module UartReciever(

input clk, //系统时钟（未分频）

input Tick, //波特率脉冲

input rst, //复位信号

input RXD, //当前接收位

output reg RXD\_OVER = 1'b0, //单次接收结束信号

output [7:0] RXD\_DATA //接收数据

);

parameter R\_IDLE = 1'b0, R\_READ = 1'b1; //状态机

reg [1:0] STATE, NEXT; //状态变量

reg read\_enable = 1'b0; //读入许可（高电平有效）

reg start\_bit = 1'b1; //读入开始标志（低电平有效）

reg [3:0] Bit = 4'b00000; //数据计数器

reg [3:0] rxd\_cnt = 4'b0000; //脉冲计数器

reg [7:0] data= 8'b00000000; //接收数据

always @ (posedge clk or negedge rst ) //状态机

begin

if (!rst) STATE <= R\_IDLE;

else STATE <= NEXT;

end

always @ (STATE or RXD or RXD\_OVER)

begin

case(STATE)

R\_IDLE: if(!RXD) NEXT = R\_READ;

else NEXT = R\_IDLE;

R\_READ: if(RXD\_OVER) NEXT = R\_IDLE;

else NEXT= R\_READ;

default NEXT = R\_IDLE;

endcase

end

always @ (STATE or RXD\_OVER)

begin

case (STATE)

R\_READ: begin

read\_enable <= 1'b1;

end

R\_IDLE: begin

read\_enable <= 1'b0;

end

endcase

end

always @ (posedge Tick)begin

if (read\_enable) begin

RXD\_OVER <= 1'b0;

rxd\_cnt <= rxd\_cnt+1;

if ((rxd\_cnt == 4'b1000) && (start\_bit))begin

start\_bit <= 1'b0;

rxd\_cnt <= 4'b0000;

end

if ((rxd\_cnt == 4'b1111) && (!start\_bit) && (Bit < 8))begin

Bit <= Bit+1;

data <= {RXD, data[7:1]};

rxd\_cnt <= 4'b0000;

end

if ((rxd\_cnt == 4'b1111) && (Bit == 8) && (RXD)) begin

Bit <= 4'b0000;

RXD\_OVER <= 1'b1;

rxd\_cnt <= 4'b0000;

start\_bit <= 1'b1;

end

end

end

assign RXD\_DATA = data;

endmodule

复位信号使状态机强制进入空闲状态，不接收任何信号，正常情况下转入下一状态。

当检测到开始信号，RXD为低电平时准备开始读入数据，否则仍保持空闲；当处于读入状态时，检测到读入结束信号，则下一状态为空闲，否则仍然是读入状态。（读入状态控制read\_anable为高电平）

在每次波特率脉冲上升沿进行数据读入，脉冲计数器计数；

探测到8个脉冲并且start\_bit无效：start\_bit在置有效，脉冲计数器复位；

探测到16个脉冲并且start\_bit有效：数据计数器计数，读入数据；

探测到16个脉冲并且RXD高电平，结束信号：全体复位

4.1.3 蓝牙数据发送----UartReply模块

`timescale 1ns / 1ps

module UartReply(

input clk, //系统时钟（未分频）

input Tick, //波特率脉冲

input rst, //复位信号

output reg TXD, //输出位

output reg TXD\_OVER = 1'b0, //输出结束标志

input [7:0] TXD\_DATA //输出数据

);

parameter T\_IDLE = 1'b0; //空闲状态

parameter T\_WRITE = 1'b1; //写

reg STATE, NEXT; //状态变量

reg write\_enable = 1'b0; //写入许可（低电平有效）

reg start\_flg = 1'b1; //开始标志（低电平有效）

reg stop\_flg = 1'b0; //结束标志（低电平有效）

reg [3:0] Bit = 4'b0000; //数据计数器

reg [3:0] txd\_cnt = 4'b0000; //脉冲计数器

reg [7:0] data = 8'b00000000; //数据地址

reg [1:0] R\_edge; //写入接脚除颤（高电平有效）

wire D\_edge; //写入接脚连接

always@(posedge clk or negedge rst)begin

if(!rst) STATE <= T\_IDLE;

else STATE <= NEXT;

end

always@(STATE or D\_edge or TXD\_DATA or TXD\_OVER)begin

case(STATE)

T\_IDLE: if(D\_edge) NEXT = T\_WRITE;

else NEXT = T\_IDLE;

T\_WRITE:if(TXD\_OVER)NEXT = T\_IDLE;

else NEXT = T\_WRITE;

default: NEXT = T\_IDLE;

endcase

end

always@(STATE)begin

case(STATE)

T\_WRITE:begin

write\_enable <= 1'b1;

end

T\_IDLE:begin

write\_enable <= 1'b0;

end

endcase

end

always@(posedge Tick)begin

if(!write\_enable)begin

TXD\_OVER = 1'b0;

start\_flg <= 1'b1;

stop\_flg <= 1'b0;

end

if(write\_enable)begin

txd\_cnt <= txd\_cnt + 1;

if(start\_flg && ! stop\_flg)begin

TXD <= 1'b0;

data <= TXD\_DATA;

end

if(txd\_cnt == 4'b1111 && start\_flg)begin

data <= {1'b0, data[7:1]};

TXD <= data[0];

start\_flg <= 1'b0;

end

if(txd\_cnt == 4'b1111 && !start\_flg && Bit < 7)begin

data <= {1'b0, data[7:1]};

Bit <= Bit + 1;

TXD <= data[0];

start\_flg <= 1'b0;

txd\_cnt <= 4'b0000;

end

if(txd\_cnt == 4'b1111 && Bit == 7 && !stop\_flg)begin

TXD <= 1'b1;

txd\_cnt <= 4'b0000;

stop\_flg <= 1'b1;

end

if(txd\_cnt == 4'b1111 && Bit == 7 && stop\_flg)begin

Bit <= 4'b0000;

TXD\_OVER <= 1'b1;

txd\_cnt <= 4'b0000;

end

end

end

always @ (posedge clk or negedge rst)begin

if(!rst) R\_edge <= 2'b00;

else R\_edge <={R\_edge[0], 1'b1};

end

assign D\_edge = !R\_edge[1] & R\_edge[0];

endmodule

空闲时：写入接脚有效时进入写状态；

写状态：写入结束标记有效进入空闲状态

检测到16个脉冲时并且start\_flg无效：start\_flg置有效，串出一位数据；

检测到16个脉冲并且start\_flg有效且数据未输出完：串出剩下7位数据；

检测到16个脉冲并且stop\_flg有效：输出高电平示意输出结束；

检测到16个脉冲并且stop\_flg无效：复位，等下一次输出

4.1.4 蓝牙控制与指令处理----bluetooth模块（蓝牙顶层模块）

`timescale 1ns / 1ps

module bluetooth(

input CLK, //系统时钟（未分频）

input RST, //复位信号

input UART\_RXD, //蓝牙数据接收位

output UART\_TXD, //蓝牙数据发送位

output reg UP, //音量增大模式（高电平有效）

output reg DOWN, //音量减小模式（高电平有效）

output [7:0] RXD\_DATA, //蓝牙接收数据

output reg [2:0] SW //切歌指令（与外部pre\_sw共同作用）

);

wire Tick; //波特率对应脉冲

wire [15:0] BaudRate; //设置波特率

assign BaudRate = 16'd650;

wire [7:0] rxd\_data; //蓝牙接收数据

wire rxd\_over, txd\_over; //接收、发送结束信号（高电平有效）

assign RXD\_DATA = rxd\_data; //传输数据

clkGenerator clkGenerator\_inst(CLK, RST, BaudRate, Tick);

//波特率对应脉冲发生器

UartReciever UartReciever\_inst(CLK, Tick, RST, UART\_RXD, rxd\_over, rxd\_data);

//接收数据

UartReply UartReply\_inst(CLK, Tick, RST, UART\_TXD, txd\_over, rxd\_data);

//发送数据反馈信息

wire sw\_enable = txd\_over & rxd\_over; //MP3模式修改使能信号

always@(posedge sw\_enable) begin //上升沿有效

if(rxd\_data != 8'h03 && rxd\_data != 8'h04)begin

UP <= 0;

DOWN <= 0;

end //保证不会重复修改音量

if(rxd\_data == 8'b0000\_0001) begin //向前列表循环模式

SW <= SW - 1;

end

else if(rxd\_data == 8'b0000\_0010)begin //向后列表循环模式

SW <= SW + 1;

end

else if(rxd\_data == 8'b0000\_0011)begin //音量增大模式

UP <= 1;

DOWN <= 0;

end

else if(rxd\_data == 8'b0000\_0100)begin //音量减小模式

DOWN <= 1;

UP <= 0;

end

else if(rxd\_data == 8'h05)begin //第0首歌单曲循环模式

SW <= 3'b000;;

end

else if(rxd\_data == 8'h06)begin //第1首歌单曲循环模式

SW <= 3'b001;

end

else if(rxd\_data == 8'h07)begin

SW <= 3'b010;

end

else if(rxd\_data == 8'h08)begin

SW <= 3'b011;

end

else if(rxd\_data == 8'h09)begin

SW <= 3'b100;

end

else if(rxd\_data == 8'h0a)begin

SW <= 3'b101;

end

else if(rxd\_data == 8'h0b)begin

SW <= 3'b110;

end

else begin

UP <= 0;

DOWN <= 0;

end

end

endmodule

利用产生的波特率脉冲使蓝牙对命令数据进行接收，并把接收到的命令数据发送回去进行反馈，方便调试；

当接收和发送都完成的时候进行歌曲的切换：由于蓝牙的接收和发送受总复位信号RST控制，使得每个RST上升沿进行一次接收，下降沿进行一次数据发送，所以命令在发送后，需要进行手动复位来达到目的，使指令有效被应用进模块中。

4.2 MP3系统

4.2.1 音量调节----VolSet模块

`timescale 1ns / 1ps

module VolSet( //声音大小设置

input VOL\_SW, //声音设置脉冲（按钮 下降沿有效）

input VOL\_RST, //音量复位信号（高电平有效）

input UP, //增大音量模式（高电平有效）

input DOWN, //减小音量模式（高电平有效）

output reg [15:0] VOL //音量数据

);

reg up; //增大音量使能信号（高电平有效）

reg down; //复位有效时二者均置空

always@(negedge VOL\_SW)begin //按钮下降沿进行数据更改

if(!VOL\_SW)begin //使能信号置有效

up <= UP;

down <= DOWN;

end

if(VOL\_RST)begin //音量复位

VOL <= 16'hf0f0;

up <= 0;

down <= 0;

end

if(up)begin //音量增大

VOL <= (VOL == 16'h0000) ? 16'h0000 : (VOL-16'h1010);

up <= 0;

end

else if(down)begin //音量减小

VOL <= (VOL == 16'hf0f0) ? 16'hf0f0 : (VOL+16'h1010);

down <= 0;

end

end

endmodule

将按钮状态作为边沿触发事件信号，下降沿有效；

复位操作需要首先拉音量复位信号VOL\_RST，然后给VOL\_SW一个脉冲下降沿，就可以完成复位，之后需要将音量复位信号拉低，否则之后的音量修改都将无效；

按钮按两次才能完成一次音量的修改，第一次是使能，让up、down这两个使能信号接收到UP、DOWN的指令状态，第二次是对音量进行修改，并马上把使能信号置空，达到只修改一次的目的；

进行音量修改需要处于UP、DOWN的音量修改模式中（由蓝牙提供模式状态），对于这两种模式下音量可以做到连续修改，但修改完成后需要使用RST总复位键将修改后的模式应用到MP3中。

4.2.2 音量编码----VolDecoder模块

`timescale 1ns / 1ps

module VolDecoder( //按照音量大小对灯光编码

input [15:0] Data\_in, //音量数据

output reg [15:0] Volcode //音量编码数据

);

always@(\*) begin

case(Data\_in)

16'h0000:begin

Volcode = 16'b1111111111111111; //最大音量

end

16'h1010:begin

Volcode = 16'b0111111111111111;

end

16'h2020:begin

Volcode = 16'b0011111111111111;

end

16'h3030:begin

Volcode = 16'b0001111111111111;

end

16'h4040:begin

Volcode = 16'b0000111111111111;

end

16'h5050:begin

Volcode = 16'b0000011111111111;

end

16'h6060:begin

Volcode = 16'b0000001111111111;

end

16'h7070:begin

Volcode = 16'b0000000111111111;

end

16'h8080:begin

Volcode = 16'b0000000011111111;

end

16'h9090:begin

Volcode = 16'b0000000001111111;

end

16'hA0A0:begin

Volcode = 16'b0000000000111111;

end

16'hB0B0:begin

Volcode = 16'b0000000000011111;

end

16'hC0C0:begin

Volcode = 16'b0000000000001111;

end

16'hD0D0:begin

Volcode = 16'b0000000000000111;

end

16'hE0E0:begin

Volcode = 16'b0000000000000011;

end

16'hF0F0:begin

Volcode = 16'b0000000000000001;

end

default: begin

;

end

endcase

end

endmodule

从0000（最大音量）使LED全亮，之后每次调节音量都依次对应亮或者灭一个灯，0F0F（最小音量）会保留一个LED灯，如果灯光全灭则代表Volume音量模块发生错误。

4.2.3 整体命令/音乐数据发送----MP3player模块（顶层模块）

`timescale 1ns / 1ps

module mp3play(

input CLK,

input RST,

output reg MP3\_RST = 1,

output reg MP3\_CS = 1,

output reg MP3\_DCS = 1,

output reg MP3\_MOSI = 0,

input MP3\_MISO,

output reg MP3\_SCLK = 0,

input MP3\_DREQ,

input VOL\_SW,

input VOL\_RST,

input UART\_RXD,

output UART\_TXD,

output [15:0] LED

);

/\*---clk divide---\*/

wire clk;

Divider #(100) Divider\_inst(CLK, clk);

/\*---order list---\*/

integer cnt = 0;

integer cmd\_seg\_cnt = 0;

parameter cmd\_cnt\_mx = 4;

//reg [127:0] cmd\_init = {32'h02000804, 32'h02000804, 32'h020BF0F0, 32'h02000800};

//reg [127:0] cmd = {32'h02000804, 32'h02000804, 32'h020BF0F0, 32'h02000800};

reg [31:0] reset\_cmd = 32'h02000804;

reg [31:0] vol\_cmd = 32'h020Bf0f0;

reg [31:0] set\_cmd = 32'h02000800;

reg [31:0] cur\_cmd = 32'h02000804;

wire [2:0] sw;

/\*---bluetooth server---\*/

wire bluetooth\_up;

wire bluetooth\_down;

wire [7:0] rxd\_data;

wire uart\_rxd;

bluetooth bluetooth\_inst(CLK, RST, UART\_RXD, UART\_TXD, bluetooth\_up, bluetooth\_down, rxd\_data, sw);

//assign LED = {6'b000000,{bluetooth\_down, bluetooth\_up}, rxd\_data[7:0]};

wire up;

wire down;

wire [15:0] vol;

wire [15:0] volcode;

assign up = bluetooth\_up;

assign down = bluetooth\_down;

VolSet VolSet\_inst(VOL\_SW, VOL\_RST, up, down, vol);

VolDecoder VolDecoder\_inst(vol, volcode);

//assign LED = {14'b00000000000000, dir[1:0]};

//assign LED = volcode;

//assign LED = {vol[15:4], VOL\_RST, VOL\_SW, up, down};

assign LED = volcode;

/\*---switch set---\*/

reg [2:0] pre\_sw = 0;

/\*wire [2:0] prev;

wire [2:0] next;

assign prev = bluetooth\_prev;

assign next = bluetooth\_next;

SW\_Set SW\_Set\_inst(clk, RST, prev, next, sw);\*/

/\*---data read---\*/

wire [31:0] data[6:0];

reg [31:0] dat;

reg [20:0] pos = 0;

blk\_mem\_gen\_0 music0(.clka(CLK), .wea(0), .addra(pos[16:0]), .dina(0), .douta(data[0]));

blk\_mem\_gen\_1 music1(.clka(CLK), .wea(0), .addra(pos[16:0]), .dina(0), .douta(data[1]));

blk\_mem\_gen\_2 music2(.clka(CLK), .wea(0), .addra(pos[16:0]), .dina(0), .douta(data[2]));

blk\_mem\_gen\_3 music3(.clka(CLK), .wea(0), .addra(pos[16:0]), .dina(0), .douta(data[3]));

blk\_mem\_gen\_4 music4(.clka(CLK), .wea(0), .addra(pos[16:0]), .dina(0), .douta(data[4]));

blk\_mem\_gen\_5 music5(.clka(CLK), .wea(0), .addra(pos[16:0]), .dina(0), .douta(data[5]));

blk\_mem\_gen\_6 music6(.clka(CLK), .wea(0), .addra(pos[16:0]), .dina(0), .douta(data[6]));

parameter INITIALIZE = 3'd0;

parameter SEND\_CMD = 3'd1;

parameter CHECK = 3'd2;

parameter DATA\_SEND = 3'd3;

parameter RSET\_OVER = 3'd4;

parameter VOL\_SET\_PRE = 3'd5;

parameter VOL\_SET = 3'd6;

reg [2:0] state = 0;

always@(posedge clk) begin

pre\_sw <= sw;

if(~RST || pre\_sw != sw) begin

MP3\_RST <= 0;

cmd\_seg\_cnt <= 0;

state <= RSET\_OVER;

cur\_cmd <= reset\_cmd;

MP3\_SCLK <= 0;

MP3\_CS <= 1;

MP3\_DCS <= 1;

cnt <= 0;

pos <= 0;

end

else begin

case(state)

INITIALIZE: begin

MP3\_SCLK <= 0;//总线时钟线清零

if(cmd\_seg\_cnt >= cmd\_cnt\_mx)//检查当前数据是否发送完毕，发送完毕进入CHECK状态准备发送下一波数据

state <= CHECK;

else if(MP3\_DREQ) begin//数据请求

MP3\_CS <= 0;//片选输入，低电平有效，置有效

cnt <= 1;//cmd发送一位

state <= SEND\_CMD;//切换到初始化之后的状态：命令发送（此处是各种寄存器的配置）

MP3\_MOSI <= cur\_cmd[31];//总线数据输出

cur\_cmd <= {cur\_cmd[30:0], cur\_cmd[31]};//串行数据输出

end

end

SEND\_CMD: begin//发送命令

if(MP3\_DREQ) begin//数据请求

if(MP3\_SCLK) begin//总线时钟线

if(cnt < 32) begin//串行发送32字节数据

cnt <= cnt+1;

MP3\_MOSI <= cur\_cmd[31];//从最高位发送

cur\_cmd <= {cur\_cmd[30:0], cur\_cmd[31]};

end

else begin//cnt == 32

MP3\_CS <= 1;//命令数据片选输入，低电平有效，置无效

cnt <= 0;

cmd\_seg\_cnt <= cmd\_seg\_cnt+1;//一段32位发送完毕

if(cmd\_seg\_cnt == 2)

cur\_cmd <= vol\_cmd;

else if(cmd\_seg\_cnt == 2)

cur\_cmd <= set\_cmd;

state <= INITIALIZE;//初始化

end

end

MP3\_SCLK <= ~MP3\_SCLK;//总线时钟计时

end

end

CHECK: begin//发送完一波（128位cmd）命令后，进行命令检查

if(vol[15:0] != vol\_cmd[15:0]) begin

state <= SEND\_CMD;

vol\_cmd <= {16'h020B, vol[15:0]}; //设置音量数据，计入下一命令中，32位，送入MP3

end

else if(MP3\_DREQ) begin//数据请求

MP3\_DCS <= 0;//音乐数据片选，字节同步，低电平有效，置有效（高电平中断传输， 强制standby空闲）

MP3\_SCLK <= 0;//总线时钟复位

state <= DATA\_SEND;//切换到下一状态，发送音乐数据

if(sw >= 0 && sw <= 6) begin//开关选择状态在第0~第6首歌中

dat <= {data[sw][30:0], data[sw][31]};//把对应曲目的数据扔进dat中

MP3\_MOSI <= data[sw][31];//串行发送到MP3

end

else begin

dat <= {data[0][30:0], data[0][31]};//指令非法则默认为第0首歌

MP3\_MOSI <= data[0][31];//穿行发送到MP3

end

cnt <= 1;//计数，当前音乐数据已经发送了一位

end

end

DATA\_SEND: begin//发送音乐数据

if(MP3\_SCLK) begin//总线时钟高电平

if(cnt < 32) begin//当前这一次串行发送16位数据

cnt <= cnt+1;

MP3\_MOSI <= dat[31];//串行发送（最高位串行）

dat <= {dat[30:0], dat[31]};//loop移动串行

end

else begin//cnt == 16本次音乐数据发送已结束

MP3\_DCS <= 1;//音乐数据片选，字节同步，低电平有效，置无效，强制拉高进入standby

pos <= pos+1;//传输进入下一行（每行16位音乐数据）

state <= CHECK;//进入命令检查模式

end

end

MP3\_SCLK <= ~MP3\_SCLK;//总线时钟计时

end

RSET\_OVER: begin//结束复位

if(cnt < 1000000)//等待1s

cnt <= cnt+1;

else begin

cnt <= 0;

state <= INITIALIZE;//进入初始化模式

MP3\_RST <= 1;//MP3复位，低电平有效，置无效

end

end

/\*VOL\_SET\_PRE: begin//音量修改准备模式

if(MP3\_DREQ) begin//数据请求

MP3\_CS <= 0;//命令片选输入，低电平有效，置有效（高电平结束当前操作，强制standby）

cnt <= 1;//当前发送了一位命令

state <= VOL\_SET;//进入音量设置模式

MP3\_MOSI <= vol\_cmd[31];//发送下一命令

vol\_cmd <= {vol\_cmd[30:0], vol\_cmd[31]};//串行发送下一命令

end

end

VOL\_SET: begin//音量设置模式

if(MP3\_DREQ) begin//数据请求

if(MP3\_SCLK) begin//总线时钟

if(cnt < 32) begin//串行发送32位命令

cnt <= cnt+1;

MP3\_MOSI <= vol\_cmd[31];//串行发送下一命令

vol\_cmd <= {vol\_cmd[30:0], vol\_cmd[31]};

end

else begin//cnt == 1下一命令发送完毕

MP3\_CS <= 1;//命令片选输入，低电平有效，置无效，强制结束当前操作进入standby模式

cnt <= 0;//命令发送计数器复位

state <= CHECK;//切换至CHECK状态

end

end

MP3\_SCLK <= ~MP3\_SCLK;//总线时钟计时

end

end\*/

endcase

end

end

endmodule

1. 测试模块建模

（要求列写各建模模块的test bench模块代码）

1. 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. 结论
2. 心得体会及建议