数电实验5：FPGA与Verilog项目进阶

# 一、实验目的

## 1、数字频率计

本实验使用Verilog语言在DSDB开发板的FPGA上实现频率检测。通过本实验掌握自主完成一个完整FPGA开发项目的经验，掌握程序调试排错的一般方法，为本课程项目设计及生物医学工程专业后续课程的学习打下坚实的基础。

## 2、数字密码锁

本实验使用Verilog语言在DSDB开发板的FPGA上实现一个带有电子密码锁的信息存取器，当且仅当拨动开关的输入与设定的密码相符时才能存取数据。

## 数字音频处理器

使用Verilog语言基于DSDB开发板的SSM2603音频编解码器实现了现实生活中常用的录音机、播放器功能。并将生物医学工程、电子信息领域常用的数字信号处理滤波技术融合进来，实现了音频去噪滤波器功能

## 数字音频录放机

通过一个带有音量调节及特定频率音频播放功能的录放机实验，进一步熟悉和掌握如何使用Verilog及FPGA设计实用的数字电路项目，为将来的项目设计打好基础。

# 二、实验原理

## 1、数字频率计

通过ELVIS开发板上的信号发生器产生指定频率的信号，用导线传送到IO端口，程序从IO端口读取频率信号计数。

## 2、数字密码锁

先做出顶层设计逻辑图，利用设计图进行模块化设计，最终连接成整体。

## 数字音频处理器

用I2C协议配置SSM2603音频编解码芯片的寄存器进行不同的功能设置，通过模块程序设计，实现数字音频处理器。

## 4、数字音频录放机

利用已编译好的audio\_recorder.xpr，），直接上传到FPGA，实现数字音频录放机的功能。

# 三、实验内容、结果与分析

## 1、数字频率计

### ①实验内容

1. 导入相关文件
2. 通过原本的.xdc文件和DSDB手册，查询信号输入端口
3. 编译原本文件，导入FPGA板，测试运行并理解代码
4. 修改程序实现量程为0~99999范围的频率测量，利用LED灯做MSB输出。

### ②实验结果

原本代码：导入后可以实现频率处于0~9999Hz的输入信号显示，但是忘记拍素材了

改进代码：见文件：数电实验5：FPGA与Verilog项目进阶 -> freq\_meter -> rst,mp4

改进后代码见：数电实验5：FPGA与Verilog项目进阶 -> freq\_meter -> \*.v&.xdc

### ③实验分析

整个系统的功能：将输入信号的频率通过FPGA板的数码管和LED灯显示出来，实现交互。

top.v：定义了各个模块和变量

freq\_meter.xdc：定义输入输出的管脚

freq\_count.v：实现频率计数，并将计算得到的count2传递给bit\_cacl.v

display.v：将计算得到的各位数字输出到数码管或LED灯

bit\_cacl.v：计算由freq\_count.v传递来的count2，将其转化为各位数字，并传递给display.v。具体的计算方法类似于取模，将余数进行下一位的运算。

修改思路：在老师给的代码的基础上，增加万位的计算和显示（实际上代码有提示，比如没有d1却有d2等，引导修改思路），具体的修改方面见文件中的注释。最终实现增加万位的计算，并将计算结果用LED灯进行二进制表示，剩下数字由数码管显示（比如10500则LED0亮，数码管显示500，59760则LED2和LED0亮，数码管显示9760等）。

SW[0]的作用：当SW[0]被按下时，count2才会开始计数；反应在代码上就是SW[0]=1,则rst\_n=1 , en在0和1之间变化，又load=~en，load会产生01变化，使clr不恒为0，count2才会计数，但在1s后clr会变化，故count2才能反应频率（每隔1s会清零）。

clk\_in的作用：查手册可知，AA12是JA的第二个输入口，所以clk\_in就是板子收到的电压信号。

## 2、数字密码锁

### ①实验内容

1. 导入相关文件
2. 编译原本文件，导入FPGA板，测试运行并理解代码

### ②实验结果

见文件：数电实验5：FPGA与Verilog项目进阶 -> lockram -> rst.mp4

### ③实验分析

#### lockram\_top

1. 数码管显示切换：通过计数器s控制数码管的切换，并通过an信号选择当前显示的数码管。根据显示模式（t\_flag）以及计数器s的值和输入信号sw/code的值，设置NUM的值，用于控制数码管的输出。
2. 密文记录与显示：根据输入信号record和mode的值，进行密文记录和密文显示。如果同时按下record和mode，则显示密文；否则，在编辑模式下，将输入的拨动开关值存储到code中，并显示在LED灯上。
3. 密码校验、修改和清零：根据输入信号judge和clr的值，进行密码校验、修改和清零操作。如果密码正确且没有按下clr，则进入密码编辑模式（j\_state=1），并将j\_stick置为1，保持长按状态。如果按下clr，则清空密码和密文。
4. 显示字符设置：根据输入信号的情况，设置NUM的值，用于控制数码管显示相应的字符。包括显示密码正确与否、编辑模式是否按下、显示模式切换等。
5. 跑马灯效果：在密码正确且不在密文记录状态下，通过计数器s的值设置LED灯的输出，实现跑马灯效果。

具体注释如下：

module lockram\_top(

input clk, //时钟

input clr, //清零功能

input mode, //显示模式切换按钮（十进制/十六进制），同时也是组合键的组成部分

input judge, //密码校验和修改按钮

input record, //密文记录（单独完成）和取出（配合mode）按钮

input[0:7]sw, //8个拨动开关

output[0:7]li, //LED灯

output[3:0]an, //数码管切换

output[6:0]a\_to\_g //数码管输出

);

//定义了一系列寄存器和信号线，用于存储计数器、密码、密文、显示模式状态等信息。

reg [4:0] NUM; //显示的数字（符号）

reg [25:0] s ; //计数器，作用包括控制显示的数码管切换

reg [7:0] key; //密码存储器

reg [7:0] code ; //密文存储器

reg t\_flag ; //记录显示模式

reg j\_state; //记录是否处在密码验证后的编辑模式

reg j\_stick; //记录judge是否在编辑模式按下，并始终保持长按

reg n\_flag; //记录数码管是显示拨动开关还是显示密文

reg lir\_flag; //记录LED灯是否处在跑马灯式的状态

reg c\_flag; //记录此前是否按了组合键

reg m\_flag; //记录此前是否按了mode键，用于反转状态

reg [7:0] lir ; //8个LED灯的输出

wire [11:0] temp;

wire [7:0] temp\_1;

assign li=lir; //赋值

assign temp\_1 = n\_flag?code:sw ; //确定输出密文还是拨动开关

//通过 assign 将 s[14:13] 的值与不同的常数进行比较，

//得到 an 数组的值，用于控制数码管的切换。

//扫描显示四个数字

assign an[0] = s[14:13]!=0;

assign an[1] = s[14:13]!=1;

assign an[2] = s[14:13]!=2;

assign an[3] = s[14:13]!=3;

always @( posedge clk ) //计数器计数

if(clk)

begin

s = s + 1;

if(s[25]) //满了就清零

s = 0;

end

//判断密码，修改密码，与清零，以状态机的形式给出

if(!judge) //如果judge没有按下，记录judge没有一直保持长按

j\_stick=0;

if(judge||clr) //长按judge

begin

n\_flag=0; //状态机状态改变

lir\_flag=1;

if(j\_stick&&judge) //密码正确则记录

key=sw;

else if(clr) //按clr则清空密码和密文

begin

key=0;

code=0;

end

if((sw==key)&&(!clr)) //密码正确但没清零

begin //进入下一个状态

j\_state=1;

j\_stick=1;

end

else

begin

j\_state=0; //密码错或者清零了，也进入另一个状态

lir=0;

end

end

if(!(judge||record||clr||mode))//如果没有按键按下，则显示相应字符

begin

if(t\_flag)

case(s[14:13]) //十进制模式直接输出，并在首位显示"D"表示十进制

3:NUM = 0; //首位显示D（长得像0）

2:NUM = temp[11:8]; //用第1到4位，下面依次类推

1:NUM = temp[7:4];

0:NUM = temp[3:0];

endcase

else

case(s[14:13]) //十六进制输出转换后的符号，并在首位显示"H"表示十六进制

3:NUM = 'h11; //显示H

2:NUM = 0; //用不到这位，用两位就够了

1:NUM = temp\_1[7:4];

0:NUM = temp\_1[3:0];

endcase

end

else if(clr) //clr按下时每一位都显示"-"，表示清零

NUM = 'h10;

else

begin

case(s[14:13]) //仅在第X位显示"-"，表示第X个按钮已按下

3:NUM = judge?'h10:'h12; judge按下则显示X，下面类似

2:NUM = record?'h10:'h12;

1:NUM = 'h12;

0:NUM = mode?'h10:'h12;

endcase

end

if(j\_state&&lir\_flag) //密码正确且不在密文记录的状态下显示跑马灯

begin

lir[0]=s[23:21]==0; //快速变化，得到跑马效果

lir[1]=s[23:21]==1;

lir[2]=s[23:21]==2;

lir[3]=s[23:21]==3;

lir[4]=s[23:21]==4;

lir[5]=s[23:21]==5;

lir[6]=s[23:21]==6;

lir[7]=s[23:21]==7;

end

end

//实例化

B8\_to\_Show C1(temp\_1, temp);

lockram\_sub A1(.NUM(NUM),.a\_to\_g(a\_to\_g));

endmodule

#### B8\_to\_Show

该模块的功能是将一个8位的二进制数转换为三个BCD（二进制码十进制）数，并将其输出。具体的注释如下：

module B8\_to\_Show(

input [7:0] B\_8, // 8位二进制数输入

output [11:0] D\_2 // 3个BCD十进制数输出

);

reg [4:0] c; //用于存放计算中间值

reg [3:0] d; //用于存放计算中间值

reg [7:0] q; //用于存放计算中间值

assign D\_2 = {d, c[3:0], q[3:0]}; // 将计算结果组合成三个BCD数并输出到top，//加工变为各位数传递给lockram\_sub

always @(\*)

begin

d = 0; // 初始化d值

c = B\_8[7:0] / 16; // 计算除以16的商

q = (B\_8[7:0] % 16) + (B\_8[7:0] / 16) \* 6; // 计算bcd码

repeat(4) // 循环4次

if (q / 16) // 如果商大于0

begin

c = c + q / 16; // c加上商

q = q % 16 + (q / 16) \* 6; // 计算bcd码

end

if (q >= 10) // 如果余数大于等于10

begin

c = c + 1; // c加1

q = q - 10; // q减去10

end

if (c >= 20) // 如果c大于等于20

begin

d = 2; // d赋值2

c = c - 20; // c减去20

end

else if (c >= 10) // 否则如果c大于等于10

begin

d = 1; // d赋值1

c = c - 10; // c减去10

end

end

endmodule

#### lockram\_sub

该模块的功能是根据输入的四位二进制数，生成对应的七段数码管的控制信号。具体注释如下：

module lockram\_sub(

input [4:0] NUM, // 输入四位二进制数

output reg [6:0] a\_to\_g // 输出对应的七段数码管控制信号

);

always @(\*)

case(NUM) //针对不同的输入而产生的对应输出

0:a\_to\_g=7'b1000000;

1:a\_to\_g=7'b1111001;

2:a\_to\_g=7'b0100100;

3:a\_to\_g=7'b0110000;

4:a\_to\_g=7'b0011001;

5:a\_to\_g=7'b0010010;

6:a\_to\_g=7'b0000010;

7:a\_to\_g=7'b1111000;

8:a\_to\_g=7'b0000000;

9:a\_to\_g=7'b0010000;

'hA: a\_to\_g=7'b0001000;

'hB: a\_to\_g=7'b0000011;

'hC: a\_to\_g=7'b1000110;

'hD: a\_to\_g=7'b0100001;

'hE: a\_to\_g=7'b0000110;

'hF: a\_to\_g=7'b0001110;

'h10: a\_to\_g=7'b0111111;//"-"符号

'h11: a\_to\_g=7'b0001001;//"H"符号

'h12: a\_to\_g=7'b1111111;//灭灯

default: a\_to\_g=7'b1000000;

endcase

endmodule

## 3、数字音频处理器

### ①实验内容

用MATLAB打开audio\_processor.m，并提前运行一下，计算好带有噪音的音频信号hn。将上传程序传入到FPGA上。上传后立刻切换到MATLAB运行命令play(hn)，观察实验现象。

### ②实验结果

见FPGA与Verilog项目进阶 ->Audio processor -> rst1和rst2

### ③实验分析

程序各模块功能分析如下：

(1) audio\_processor.v

\* 对clk\_24M时钟信号分频获得AC\_LRC=24.576MHz/256=48kHz；

\* 对clk\_24M时钟信号分频获得AC\_BCLK=24.576MHz/8=3.072MHz；

\* 模块例化ssm2603\_ctrl；

\* 模块例化debounce，对按钮进行简单的防抖处理；

\* 模块例化recorder。

\* t\_node连接外部端口，可以基于NI ELVISmx Instrument Launcher中的Oscilloscope和Dynamic Signal Analyzer软件，使用BNC信号连接线，面包板跳线进行测试。

(2) ssm2603\_ctrl.v

\* 模块例化ssm2603\_config，配置SSM2603的寄存器；

\* 模块例化audio\_receive，FPGA接收SSM2603的模数转换音频数据adc\_data；

\* 模块例化audio\_send，FPGA向SSM2603传送数模转换音频数据dac\_data。

(3) ssm2603\_config.v

\* 设置SSM2603 I2C器件地址为0011010、音频字长16等参数；

\* 模块例化i2c\_dri，调用IIC协议；

\* 模块例化i2c\_reg\_cfg，配置SSM2603的寄存器。

(4) i2c\_reg\_cfg.v

\* 使用I2C协议配置SSM2603寄存器

\* I2C写间隔控制（第1个和第2个寄存器配置时延时一段时间）；

\* 满足条件时触发I2C操作控制信号i2c\_exec；

\* 配置寄存器个数计数；

\* 配置寄存器完成信号；

\* 配置寄存器R0~R9的地址及其数据。

(5) i2c\_dri.v

\* 生成I2C的SCL的四倍频率的驱动时钟用于驱动i2c的操作

\* 定义三段式状态机（图8），描述时序状态转移，根据i2c\_exec和st\_done的值判断状态转移条件

\* 按照图4、图5的I2C时序图，设置时序信号scl和sda。完成I2C操作之后设置i2c\_done为1。

图8. I2C驱动模块状态跳转图

(6) audio\_receive.v

\* 按照图6，使用I2S音频传输标准接收音频信号；

\* 为了在AC\_LRC变化的第二个AC\_BCLK上升沿采集AC\_RECDAT,延迟打拍采集，对LRC信号进行边沿检测；

\* 16位音频采集数据计数；

\* 把采集到的音频数据临时存放在一个寄存器内；

\* 把临时数据传递给adc\_data,并使能rx\_done,表明一次采集完成。

(7) audio\_send.v

\* 按照图6，使用I2S音频传输标准发送音频信号；

\* 为了在AC\_LRC变化的第二个AUD\_BCLK上升沿采集AC\_PBDAT,延迟打拍采集；

\* 对LRC信号进行边沿检测；

\* 16位音频发送数据计数；

\* 把要发送的音频数据串行发送出去；

\* 16位数据计数完毕，设置完成信号。

(8) recorder.v

\* 模块例化mybram，访问自定义的block memory内存；

\* 模块例化fir31，调用滤波器，滤波器输入信号filterIn，输出信号filterOut。

\* 输入from\_ssm\_data

\* 输出to\_ssm\_data

\* 判断如果filter为真，则输出to\_ssm\_data为filterOut[31:16]，否则输出to\_ssm\_data为memOut；

\* 录音直到内存满了

(9) mybram.v

\* 如果we写使能信号we为真，将输入信号din保存到地址为addr的内存mem[addr] 中；

\* 否则将mem[addr]的内容读出，赋值给dout。

(10) fir31.v

\* 模块例化coeffs31，设置滤波器系数；

\* 输入信号x卷积滤波器，得到滤波之后的信号y。

(11) coeffs31.v

\* 为阶次order为31的滤波器系数赋值

(12) debounce.v

\* 对按键进行简单的防抖处理：如果按键输入在0.01秒内发生改变，说明按键有抖动，则重启0.01秒时钟等待；如果输入稳定0.01秒，则将按键的值输出。

## 数字音频录放机

### ①实验内容

用Vivado打开项目audio\_recorder目录中的audio\_recorder.xpr，已经编译好（无需重新编译，可少占用时间），直接上传到FPGA。用Windows自带的播放器打开FairyTale.mp3播放。

### ②实验结果及分析：

（1）通过UART串口通信控制SSM2603调节音量和静音的修改思路：

进行程序模块编程，分为以下三个模块：

UART串口通信模块： 首先，在FPGA中实现一个UART串口通信模块，该模块负责接收来自电脑键盘的指令。使用Verilog语言编写一个UART接收模块，通过FPGA的输入引脚接收来自电脑的串口数据。

指令解析模块： 在接收到UART串口数据后，编写Verilog代码来解析这些数据，以确定是什么样的控制指令。例如，可以定义一些特定的命令格式，用于调节音量、静音等功能。

SSM2603控制模块： 根据解析得到的指令，编写Verilog代码来控制SSM2603。对于音量调节，可以通过SPI（Serial Peripheral Interface）或I2C（Inter-Integrated Circuit）总线与SSM2603进行通信，设置相应的寄存器来调整音量。对于静音功能，可以控制相应的寄存器位。

1. 使用Verilog程序控制SSM2603在特定的时间播放音乐、语音、正弦波的修改思路：

使用一个计时器模块，结合状态机，来控制何时播放何种音频。具体实现为在Verilog中设计一个状态机和一个计数器模块，状态机会根据时间和自身状态决定是否播放音乐以及播放音乐的类别。模块如下：

计时器模块：在Verilog代码中，实现一个计时器模块，该模块会生成一个计时信号。使用这个信号来计算经过的时间，并在达到特定时间时触发相应的操作。

状态机：状态机中，引入一个状态，该状态表示当前是否需要播放音频。在特定的时间触发时，状态机会将状态设置为需要播放音频，并根据需要切换到不同的音频类型。

# 遇到的问题

问题：在实现数字频率机实验中，插线正确的情况下出现示数但示数不准确。

解决方法：在一一排除的方法下发现是示波器接口数据线存在问题，更换后成功解决问题。

# 反思与心得

本次实验是开展数电实验以来最具挑战性的一次实验，通过这次试验，我们更熟练的掌握如何使用Verilog及FPGA设计较复杂的数字电路项目，同时还实现了数字频率机、数字密码锁等非常有意思的实验。