2011-2012 学年第二学期

硬件课设实验报告

课题名称：基于 Xilinx 的 FPGA 的打地鼠游戏机设计

院　　系：电子与信息工程

专　　业：电子信息工程

小组成员：　电信 0904 杨　溢 U200913699

电信 0907 马　可 U200913766

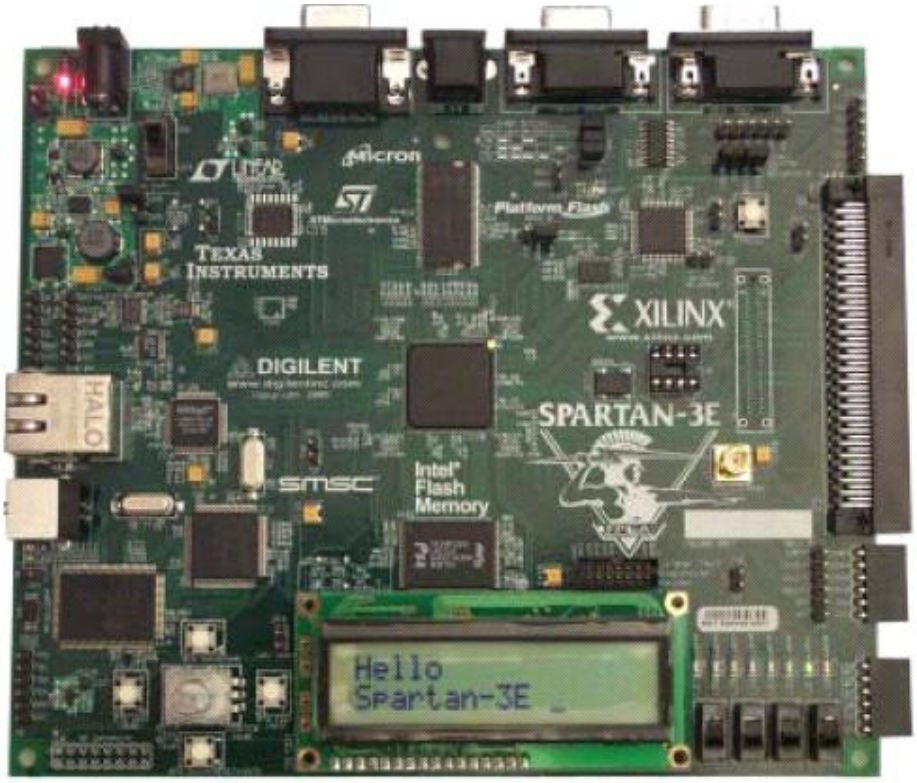
电信 0904 邓中凯 U200913691

指导老师：杨小献

一、课设要求

1.1. 项目描述

应用 Xilinx 大学计划 FPGA 实验板实现一个打地鼠游戏机，参与者根据系统中的亮灯提示控制相应按键，模拟打地鼠的实际情况。对按键控制越准确，说明参与者的身体越灵活，反应越快。由 3 个同学组成一个设计小组进行设计。



1.2. 任务要求：

1）基本功能：用 8 个灯作目标，与之对应有 8 个按键进行控制。每一次 8 个灯中随机出现一个灯处于“亮”的状态，在灯亮的时间内要求按到对应的按键，若按到则加 1 分，且灯熄灭；否则命数扣一条。

设定初始命数为 9，得分每达到 10 分便加 1 条命，失误一次扣一条命，命数为 0 时游戏结束。

游戏分四个难度级别，每个级别灯闪亮的速度不同，级别越高，速度越快，

灯亮的时间越短。得分每达 10 分，难度自动升一级。

设有暂停/继续和开始/停止功能，能记录和更新历史最高分数。控制 LCD 进行相应的显示。

每个难度级别对应产生不同的音乐，与灯闪的频率节奏相当。

2）扩展功能：①用 VGA 显示图案的随机闪动，②通过键盘不同按键对应相应的图案，③其它自行设计实现的本系统的扩充功能。

二、需求分析

在小组的开题讨论会，我们对课设要求进行了分析，总结有以下几个主要的部分：

■ 地鼠随机出现，需要产生随机数作为 LED 灯的信号，实现 LED 灯的随机闪亮。

■ 对应八只地鼠，需要 8 个按键，并在软件代码中比较按键状态和 LED 灯的状态，以判断是否打到地鼠。

■ 要播放背景音乐，需要一只蜂鸣器，实现音乐播放。

■ 要求设计四个游戏难度等级，则需要一个两位的二进制信号来控制频率，以产生不同的游戏速度。

■ 需要暂停/开始功能，则需要开发板上的拨动开关作为暂停控制。

三、整体设计及小组分工

经过进一步的讨论，我们小组决定采用一下方案来设计打地鼠游戏机，以满足课设要求。

硬件方面：

■ 用面包板作为外围扩展，面包板上应有 8 个 LED 灯、一个 3/8 译码器、8 个按键、一个蜂鸣器。其中 8 位按键信号占用开发板 8 个 I/O 接口，作为输入；用 3 个 I/O 接口，通过 3/8 译码器，译为 8 为 LED 电平信号，作为输出；一个 I/O 端口作为音乐信号的输出。这样，可以满足开发板只有 12 个可扩展 I/O 端口的限制。

■ 用 Sparten 3E 开发板上的 LCD 显示屏显示分数、生命数等玩家信息。LCD 相关信号的输出端口如 LED_en 等是开发板已经绑定的，不用扩展。

■ 用 Sparten 3E 开发板的 VGA 接口实现 VGA 扩展，外接至显示屏。与 LCD 相同，VGA 显示所需的行同步、列同步、RGB 色值等信号已经绑定至特定端口，只需向特定端口写相应数据即可，不用占用可扩展的 I/O 接口。

软件方面：

■ 设计以下几个模块

1. 顶层模块（WAMTop）

定义工程输出和输入端口，调用各个子模块完成各项功能。

2. 计分模块（ScoreCount）

积分模块是工程中最重要的模块，它应完成产生开始/暂停/复位控制、随机数、将三位随机数转换为 8 位 LED 电平信号、判断是否打地鼠正确并依据结果进行加减分数操作、依据玩家当前分数控制游戏难度、记录历史最高分等等主要功能。

3. LCD 显示模块（LcdDisplay）

LCD 显示模块是一个相对独立的模块，它从顶层模块中获取暂停/开始/复位信号，从分数计算模块中获取当前玩家信息，依据控制信号，输出历史最高分、玩家分数、玩家命数等信息。

4. 音乐播放模块（MusicPlayer）

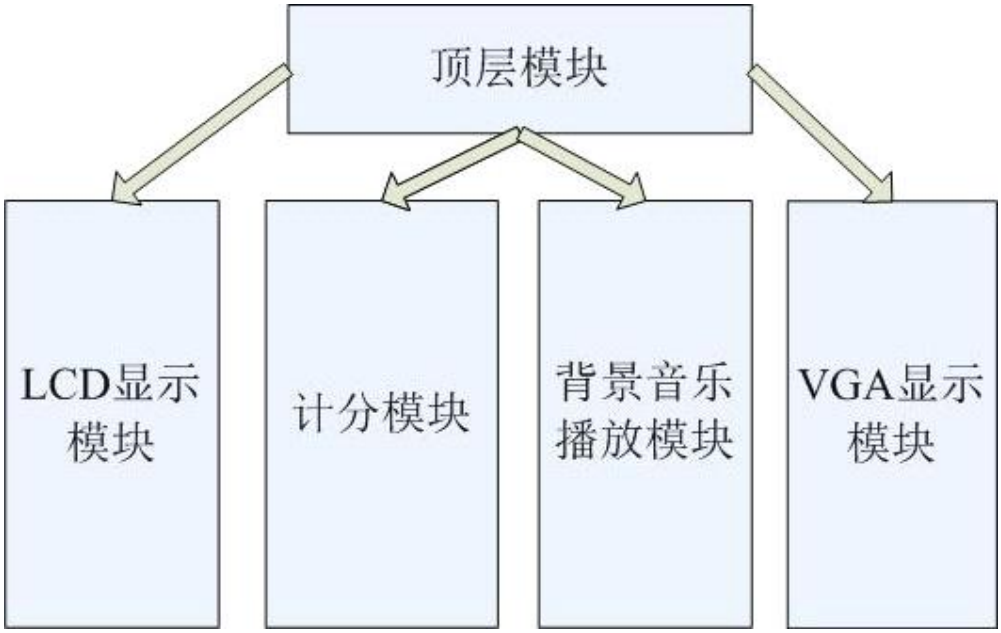
音乐播放模块也是一个相对独立的模块，只需要从计分模块中获取当前游戏难度，并依据游戏难度播放不同的音乐即可。

5. VGA 显示模块（VGADisplay）

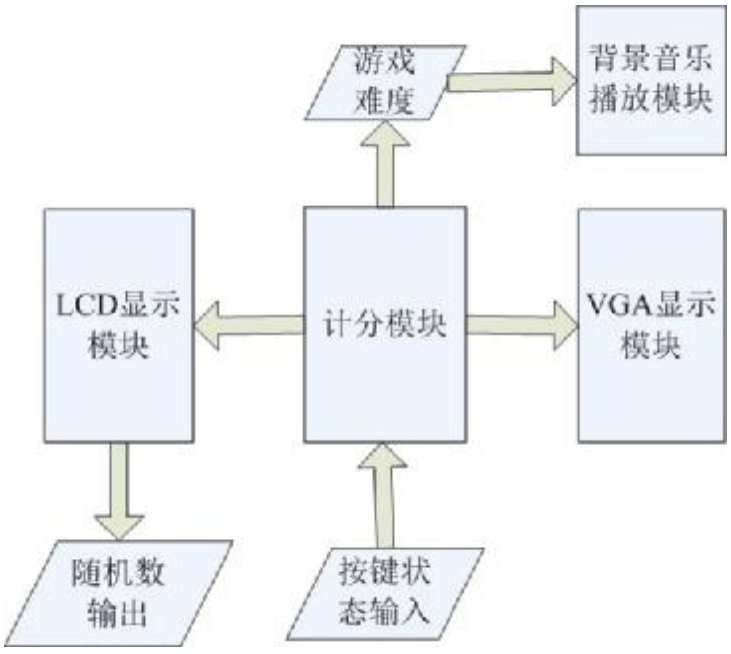
VGA 显示模块是本次实验的一个扩展，用于替代面包板上的 LED 灯，在显示器上显示地鼠在哪里出现即可。它从计分模块中获得当前游戏难度和随机数这两组信号，并根据不同的游戏难度，以不同的频率，在显示屏上显示“地鼠”，显示地鼠的位置由输入的随机数决定。

以上是我们设计出的主要的功能模块，其它模块如分频模块和 3/8 译码器模

块就不再列出，整体设计架构示意图如下：



子模块间关系图如下：



在整体设计完成之后，小组分工如下：

杨　溢：负责顶层模块、计分模块、LCD 显示模块、VGA 扩展；

马　可：负责音乐模块；

邓中凯：负责外围硬件电路（面包板）的设计和制作。

四、模块介绍

由于工程以软件代码设计为主，下面我逐个介绍各个模块的设计及其功能实现。

4.1 顶层模块（WAMTop）

在顶层模块中，我们定义了输入输出信号变量、各个子模块之间传递参数时用到的 wire 型变量，并调用各个主要子模块。

函数接口如下：

```
module WamTop(
    Clk50M,
    Reset,
    Button8Bit,
    LedOut,
    GamePause,
    GameStart,
    GameMusic,
    LcdOut,
    LcdRs,
    LcdEnable,
    LcdRw,
    hsync,
    vsync,
    vga_r,
    vga_g,
    vga_b
```

```
);
```

输入输出接口定义如下：

```
input Clk50M;
input Reset;
input GamePause;
input GameStart;
input [7:0]Button8Bit
output vga_r;
output vga_g;
output vga_b;
output hsync;
output vsync;
output [2:0]LedOut;
output [3:0]LcdOut;
output LcdRw;
output LcdRs;
output LcdEnable;
output GameMusic;
```

Wire 类型变量定义如下：

```
wire vga_r;
wire vga_g;
wire vga_b;
wire hsync;
wire vsync;
wire [3:0]PlayerScoreHun;
wire [3:0]PlayerScoreTen;
wire [3:0]PlayerScoreBit;
wire [3:0]MaxScoreHun;
wire [3:0]MaxScoreTen;
wire [3:0]MaxScoreBit;
wire [3:0]PlayerLifeHun;
wire [3:0]PlayerLifeTen;
wire [3:0]PlayerLifeBit;
wire Clk500;
wire GameOver;
wire [1:0]GameSpeed;
wire [2:0]Ledout;
```

函数调用如下：

```
FrequencyDid F1(Clk50M,Clk500);
ScoreCount S1(Reset,Clk500,GameStart,GamePause,Button8Bit,LedOut,GameSpeed,GameOver,MaxScoreHun,MaxScoreTen,
   MaxScoreBit,PlayerScoreHun,PlayerScoreTen,PlayerScoreBit,PlayerLifeHun,PlayerLifeTen,PlayerLifeBit,
   ButtonRight
    );
LcdDisplay L1(Reset,Clk500,PlayerLifeTen,PlayerLifeBit,PlayerScoreHun,PlayerScoreTen,PlayerScoreBit,
   MaxScoreHun,MaxScoreTen,MaxScoreBit,GameStart,GamePause,GameOver,LcdOut,LcdEnable,LcdRs,LcdRw);
MusicPlayer M1(Clk50M,Clk500,GameSpeed,GameMusic);
VGADisplay V1(Clk50M,Reset,hsync,vsync,vga_r,vga_g,vga_b,ButtonRight,LedOut);
```

4.2 计分模块（ScoreCount）

计分模块的函数接口如下：

```
module ScoreCount(
    Reset,
    Clk500,
    GameStart,
    GamePause,
    Button8Bit,

    LedOut,
    GameSpeed,
    GameOver,
    MaxScoreHun,
    MaxScoreTen,
    MaxScoreBit,
    PlayerScoreHun,
    PlayerScoreTen,
    PlayerScoreBit,
    PlayerLifeHun,
    PlayerLifeTen,
    PlayerLifeBit,
    ButtonRight
    );
```

变量定义如下：

```
43  input [7:0]ButtonSBit;
44  input Reset;
45  input Clk500;
46  input GameStart;
47  input GameMouse;
48  output [2:0]LedOut;
49  output [1:0]GameSpeed;
50  output GameOver;
51  output [3:0]MaxScoreHun;
52  output [3:0]MaxScoreTen;
53  output [3:0]MaxScoreBit;
54  output [3:0]PlayerScoreHun;
55  output [3:0]PlayerScoreTen;
56  output [3:0]PlayerScoreBit;
57  output [3:0]PlayerLifeHun;
58  output [3:0]PlayerLifeTen;
59  output [3:0]PlayerLifeBit;
60  output ButtonRight;
61
62  reg [31:0]RandomNum;
63  reg [31:0]RandomNumSeed;
64  reg [2:0]LedOut;
65  reg GameOver;
66  reg [1:0]GameSpeed;
67  reg [3:0]PlayerLifePlus;//每次多打到地鼠了10次，则将10的个位数加一
68  reg [3:0]MaxScoreHun,MaxScoreTen,MaxScoreBit,PlayerScoreHun,PlayerScoreTen,PlayerScoreBit,PlayerLifeHun,
69
70  wire [2:0]ButtonSBit;
71  reg [10:0]ClkCount;
72  reg [10:0]ClkCountTop;
```

在计分模块中，首先判断是否有复位信号，有复位信号，则进行对应处理：

```
always@(posedge Clk500 or negedge Reset)
begin
    if(~Reset)//复位时应有的操作
        begin
            LedOut <= 3'b000;
            GameSpeed <= 2'b00;
            GameOver <= 1'b1;
            PlayerScoreHun <= 4'b0000;
            PlayerScoreTen <= 4'b0000;
            PlayerScoreBit <= 4'b0000;
            PlayerLifeTen <=4'b0000;
            PlayerLifeBit <=4'b1001;
            PlayerLifePlus <= 4'b0000;
            MaxScoreHun <= 4'b0000;
            MaxScoreTen <= 4'b0000;
            MaxScoreBit <= 4'b0000;
            ClkCount <= 10'b000000000;
            RandomNumSeed <= 32'h00000003;
            ClkCountTop <= 10'b0111111111;
        End
```

然后判断是否有游戏开始信号，有则进行对应处理：

```
else if(!GameStart)
        begin
```

```
            if(LedOut == 3'b111)
                begin
                    LedOut <= 3'b000;
                end
            else
                begin
                    LedOut <= LedOut + 16'b1;
                end
                        LedOut <= 3'b000;
            GameSpeed <= 2'b00;
            GameOver <= 1'b1;
            PlayerScoreHun <= 4'b0000;
            PlayerScoreTen <= 4'b0000;
            PlayerScoreBit <= 4'b0000;
            PlayerLifeTen <=4'b0000;
            PlayerLifeBit <=4'b1001;
            PlayerLifePlus <= 4'b0000;
            ClkCount <= 10'b000000000;
            RandomNumSeed <= 32'h00000003;
            ClkCountTop <= 10'b0111111111;
        End
```

然后判断游戏是否已经结束，已经结束则进行频率极高的跑马灯，实现点亮所有 LED 的效果：

```
    else if(!GameOver)
        begin
            if(LedOut == 3'b111)
                begin
                    LedOut <= 3'b000;
                end
            else
                begin
                    LedOut <= LedOut + 16'b1;
                end
        End
```

接着判断是否在暂停，是则不进行任何操作。

若以上判断都为假，则游戏在正常进行，执行相应的操作。

首正常进行时，先根据当前分数确定对应的难度等级：

```
136             if(PlayerScoreHun >= 4'b0001)//先调整游戏难度
137                begin
138                   GameSpeed <= 2'b11;
139                end
140             else
141                begin
142                   case(PlayerScoreTen)
143                      4'b0000: GameSpeed <= 2'b00;
144                      4'b0001: GameSpeed <= 2'b01;
145                      4'b0010: GameSpeed <= 2'b10;
146                      4'b0011: GameSpeed <= 2'b11;
147                      default: GameSpeed <= 2'b11;
148                   endcase
149                end
150
151             case(GameSpeed)
152                2'b00: ClkCountTop <= 10'b1111111111;
153                2'b01: ClkCountTop <= 10'b0111111111;
154                2'b10: ClkCountTop <= 10'b1101111111;
155                2'b11: ClkCountTop <= 10'b1110111111;
156             endcase
```

其中，ClkCountTop 变量是用来控制计数器的顶端，借以控制游戏速度。

然后通过是随机数产生模块，我们采用线性同余法产生伪随机数，我们参阅了很多文章，用线性同余法产生伪随机数用 verilog 语言很好实现，同时，在应用到打地鼠游戏机上时，可以确保游戏过程中不重复。

```
166             RandomNum <= (RandomNumSeed<<20)+(RandomNumSeed<<19)+(RandomNumSeed<<16)+(RandomNumSeed<<14)
167             +(RandomNumSeed<<13)+(RandomNumSeed<<10)+(RandomNumSeed<<9)+(RandomNumSeed<<3)
168             +(RandomNumSeed<<2)+RandomNumSeed+1;
169
170             LedOut <= RandomNum[31:29];
171
172             RandomNumSeed <= RandomNum;
```

我们提取 32 位的随机数的前三位作为伪随机数输出，即为上图中的 LedOut。

然后读入按键的状态，并判断按键状态是否和随机数匹配，匹配则为打对地鼠，玩家的分数就加一，若不匹配，则为按错，玩家命数减一。

```
174                if(LedOut[2:0] == Button3Bit[2:0])//按对了的情况下
```

同时还设置了一个 PlayerLifePlus 变量，每当玩家答对，该变量加一，当该变量等于 5 时，玩家命数加一。这个操作的目的在于玩家连续打对五次时，奖励一条命。

以上就是计分模块的主要内容。

4.3 LCD 显示模块（LcdDisplay）

LCD 显示模块从计分模块获得玩家当前分数、当前生命值并显示。

4.3.1 LCD 显示原理

Sparten 3E 开发板上的 LCD 显示屏是 1602 显示屏，在参阅了网上的一些资料后，我们大致了解了 LCD 的显示原理：

◆ LCD 显示屏主要有 3 个控制信号和一组 4bit 的数据信号。通过三个控制信号（使能、写使能、读写控制）写 LCD 屏的控制字，并通过 4bit 的数据信号，输出要显示的内容。

◆ 向数据区内输出想要显示的字符的 ASCII 码，即可在 LCD 显示屏上显示对应的内容。由于 ASCII 码为 8bit 而数据信号只有 4bit，故需要两次写操作才能在 LCD 显示屏上显示一个字符。

◆ LCD 显示屏工作在 500HZ 频率，向数据区反复写要显示的内容并保持循环，使得 LCD 刷屏时保证输出稳定。

4.3.2 LCD 显示模块代码

首先是模块的接口定义：

```
module LcdDisplay(
    Reset,
    Clk500,
    PlayerLifeTen,
    PlayerLifeBit,
    PlayerScoreHun,
    PlayerScoreTen,
    PlayerScoreBit,
    MaxScoreHun,
    MaxScoreTen,
    MaxScoreBit,
    GameStart,
    GamePause,
    GameOver,
    LcdOut,
```

```
LcdEnable,
LcdRs,
LcdRw
);
```

然后是模块内部的变量定义：

```
41  input Reset;
42  input Clk500;
43  input GamePause;
44  input GameOver;
45  input GameStart;
46  input [3:0]MaxScoreHun;
47  input [3:0]MaxScoreTen;
48  input [3:0]MaxScoreBit;
49  input [3:0]PlayerScoreHun;
50  input [3:0]PlayerScoreTen;
51  input [3:0]PlayerScoreBit;
52  input [3:0]PlayerLifeTen;
53  input [3:0]PlayerLifeBit;
54
55  output [3:0]LcdOut;
56  reg [3:0]LcdOut;
57
58  output LcdEnable;
59  output LcdRs;
60  output LcdRw;
61  wire LcdEnable;
62  reg LcdRs;
63  reg LcdRw;
64  reg [6:0] LcdState;
65  reg [2:0] LcdCounter;                    //计数器，在初始化时计数实现等待
66  assign LcdEnable=Clk500;
```

可见，上图中的 LcdEnable 变量就是 LCD 的使能信号，LcdRs 是决定是在写控制命令还是在写显示数据，LcdRw 是写使能信号。

写数据时的循环是用 case 语句实现的，在 case 所在的 always 体中，每当时钟上升沿到来时，循环变量就加一，以进行到下一步操作，具体形式如下：

```
81          case(LcdState)
82             7'd0:                  //lcd屏幕初始化，0x33
83                begin
84                   LcdRs<=0;
85                   LcdOut<=4'b0011;
86                   LcdState<=LcdState+1;
87                end
88             7'd1:
89                begin
90                   if(LcdCounter!=3)
91                      begin
92                         LcdCounter<=LcdCounter+1;
93                      end
94                    else
95                      begin
96                         LcdCounter<=0;
97                         LcdOut<=4'b0011;
98                         LcdState<=LcdState+1;
99                      end
```

在 7'd14 之前，都是在写 LCD 显示的控制字，之后，从 14 至 78 期间，写要 LCD 显示屏要显示的内容：

```
161             7'd14:                  //如果over信号为高显示O，否则当pause为高显示P，否则显示S
162                begin
163                  LcdRs<=1;
164                     if(!GameOver)
165                        begin
166                           LcdOut<=4'b0100;
167                           LcdState<=LcdState+1;
168                        end
169                     else
170                        begin
171                           LcdOut<=4'b0101;
172                           LcdState<=LcdState+1;
173                        end
174                end
175            7'd15:
176               begin
177                  LcdRs<=1;
178                  if(!GameOver)
179                      begin
180                        LcdOut<=4'b1111;
181                        LcdState<=LcdState+1;
182                      end
```

在 case 函数体的最后，我们让循环控制变量 Lcd_State 回到 7'd12，重新开始写数据，以便在 LCD 刷屏时，保持输出稳定。

```
528             7'd72:              //第二行第十四个，
529                begin
530                   LcdRs<=1;
531                   LcdOut<=4'b0010;
532                   LcdState<=LcdState+1;
533                end
534             7'd73:
535                begin
536                   LcdRs<=1;
537                   LcdOut<=4'b0000;
538                   LcdState<=7'd12;
539                end
540          endcase
```

4.4 音乐播放模块（MusicPlayer）

音乐播放模块一直是困扰我们小组的一个主要问题，因为我们不知道如何用一个蜂鸣器去播放一首乐曲。

当马可把这个问题在小组会议上提出来时，我和邓中凯也都不能解决。在向电信系某老师咨询之后，他给了我们他的博客的网址，让我们去参阅他的文章和

代码。

在得到了老师的帮助之后，我们才得以了解音乐播放的原理，进一步地实现音乐播放模块。

4.4.1 音乐播放器的原理

众所周知，蜂鸣器只能依据输入电压播放单音，要想用蜂鸣器播放一首乐曲，就要在代码中预设乐谱，并通过不同的频率驱动蜂鸣器在不同的音阶发生。

声音的频谱范围约在几十到几千赫兹，若能利用程序来控制 FPGA 某个引脚输出一定频率的矩形波，接上扬声器就能发出相应频率的声音。乐曲中的每一音符对应着一个确定的频率，要想 FPGA 发出不同音符的音调，实际上只要控制它输出相应音符的频率即可。乐曲都是由一连串的音符组成，因此按照乐曲的乐谱依次输出这些音符所对应的频，就可以在扬声器上连续地发出各个音符的音调。而要准确地演奏出一首乐曲，仅仅让扬声器能够发生是不够的，还必须准确地控制乐曲的节奏，即乐曲中每个音符的发生频率及其持续时间是乐曲能够连续演奏的两个关键因素。

多个不同频率的信号可通过对某个基准频率进行分频器获得。由于各个音符的频率多为非整数，而分频系数又不能为小数，故必须将计算机得到的分频系数四舍五入取整。若基准频率过低，则分频系数过小，四舍五入取整后的误差较大，若基准频率过高，虽然可以减少频率的相对误差，但分频结构将变大。实际上应该综合考虑这两个方面的因素，在尽量减少误差的前提下，选取合适的基准频率。由于最大分频系数是 1274，故分频器采用 11 位二进制计数器能满足要求，乐曲中的休止符，只要将分频系数设为 0，即初始值=211-1=2047，此时扬声器不会发声。

下表为各个音符的频率，以及它在该模块中应该用多大的分频：

音符名	频 率 (Hz)	分频 系数	计数 初值	音符名	频 率 (Hz)	分频 系数	计数 初值
休止符	375000	0	2047	中音 4	796.178	468	1579
低音 1	294.349	1274	773	中音 5	882.353	425	1622
低音 2	330.396	1135	912	中音 6	989.446	379	1668
低音 3	370.92	1011	1036	中音 7	1136.363	330	1717

低音 4	386.598	970	1077	高音 1	1175.549	319	1728
低音 5	394.737	950	1197	高音 2	1353.790	277	1770
低音 6	495.376	757	1290	高音 3	1512.097	248	1799
低音 7	555.56	675	1372	高音 4	1609.442	233	1814
中音 1	588.697	637	1410	高音 5	1802.884	208	1839
中音 2	638.84	587	1480	高音 6	2027.027	185	1862
中音 3	742.574	505	1542	高音 7	2272.727	165	1882

以这次课设中我们采用的第一首歌曲（蓝精灵）为例，最小的节拍为 1/4 拍，若将 1 拍的时间定为 1 秒，则只需要提供一个 4Hz 的时钟频率即可产生 1/4 拍的时长（0.25 秒），对于其它占用时间较长的节拍（必为 1/4 拍的整数倍）则只需要将该音符连续输出相应的次数即可。计数时钟信号作为输出音符快慢的控制信号，时钟快时输出节拍速度就快，演奏的速度也就快，时钟慢时输出节拍的速度就慢，演奏的速度自然降低。

4.4.2 音乐播放模块的代码说明

首先是模块的接口设计：

module MusicPlayer(Clk50M,Clk500,GameSpeed,GameMusic);

其中，GameSpeed 是输入信号，根据难度等级，播放不同的乐曲，GameMusic 为输出电平，通过 GameMuic 信号在不同频率下的翻转，实现音乐的播放。

下面是不同的音阶对应的分频计数器的触发值的选择：

```
 98  always @(posedge clkChoose)
 99  begin
100     case({FreHigh,FreMed,FreLow})
101     12'b000000000001:origin<=139;
102     12'b000000000010:origin<=348;
103     12'b000000000011:origin<=532;
104     12'b000000000100:origin<=614;
105     12'b000000000101:origin<=771;
106     12'b000000000110:origin<=911;
107     12'b000000000111:origin<=1035;
108     12'b000000010000:origin<=1091;
109     12'b000000100000:origin<=1182;
110     12'b000000110000:origin<=1288;
111     12'b000001000000:origin<=1324;
112     12'b000001010000:origin<=1409;
113     12'b000001100000:origin<=1479;
114     12'b000001110000:origin<=1541;
115     12'b000100000000:origin<=1569;
116     12'b001000000000:origin<=1622;
117     12'b001100000000:origin<=1668;
118     12'b010000000000:origin<=1689;
119     12'b010100000000:origin<=1728;
120     12'b011000000000:origin<=1763;
121     12'b011100000000:origin<=1794;
122     12'b000000000000:origin<=2047;
123     endcase
```

下面是根据不同的音阶，通过 GameMusic 信号电平的翻转，实现扬声器发声：

```
78  reg GameMusic;
79  wire carry;
80  assign carry=(divider==11'd2047);
81  always @(posedge Clk1M)
82  begin
83     if(carry)
84     begin
85        divider<=origin;
86     end
87     else
88     begin
89        divider<=divider+1;
90     end
91  end
92
93  always @(posedge carry)
94  begin
95     GameMusic<=~GameMusic;
96  end
```

下面是我们预设的乐谱，本质上就是不同的音阶，形式如下：

```
147     case(MusicState)
148     0: {FreHigh,FreMed,FreLow}<='b000000110000;
149     1: {FreHigh,FreMed,FreLow}<='b000000110000;//3
150     2: {FreHigh,FreMed,FreLow}<='b000001000000;
151     3: {FreHigh,FreMed,FreLow}<='b000001000000;//4
152     4: {FreHigh,FreMed,FreLow}<='b000001010000;
153     5: {FreHigh,FreMed,FreLow}<='b000001010000;
154     6: {FreHigh,FreMed,FreLow}<='b000001010000;
155     7: {FreHigh,FreMed,FreLow}<='b000001010000;//5
156     8: {FreHigh,FreMed,FreLow}<='b000001000000;
157     9: {FreHigh,FreMed,FreLow}<='b000001000000;
158     10:{FreHigh,FreMed,FreLow}<='b000001000000;
159     11:{FreHigh,FreMed,FreLow}<='b000001000000;// 4
160     12:{FreHigh,FreMed,FreLow}<='b000000110000;
161     13:{FreHigh,FreMed,FreLow}<='b000000110000;
162     14:{FreHigh,FreMed,FreLow}<='b000000110000;
163     15:{FreHigh,FreMed,FreLow}<='b000000110000;//3
164     16:{FreHigh,FreMed,FreLow}<='b000000110000;
```

整个 case 仍然是一个大的循环体，循环控制方法与 LCD 模块中介绍的方法类似，在此不再赘述。在我们设计的打地鼠游戏机中，一共有四个难度等级，对应四首不同的歌曲：蓝精灵、最炫民族风、忐忑、甩葱歌。为了实现在不同的难度等级下，播放不同的音乐，我们定义了两个循控制变量：BotState、TopState：

```
126  always@(Clk1M)
127  begin
128     case(GameSpeed)
129        2'b00: begin TopState <= 9'd116;BotState <= 9'd0; end
130        2'b01: begin TopState <= 9'd309;BotState <= 9'd117; end
131        2'b10: begin TopState <= 9'd380;BotState <= 9'd310; end
132        2'b11: begin TopState <= 9'd501;BotState <= 9'd381; end
133     endcase
134  end
```

当难度等级不同时，我们就把对应的歌曲在预设乐谱中的起始位置和结束位置放入这两个变量中，同时通过下面的控制方式实现在这两个变量之间循环，播放固定的歌曲：

```
136  always @(posedge clkChoose)
137  begin
138     if(MusicState == TopState)
139        begin
140           MusicState <= BotState;
141        end
142     else
143        begin
144           MusicState <= MusicState+1;
145        end
146
147     case(MusicState)
148     0: {FreHigh,FreMed,FreLow}<='b000000110000;
149     1: {FreHigh,FreMed,FreLow}<='b000000110000;//3
```

以上就是音乐播放模块的全部内容，值得一提的是为了这个音乐播放模块的预设乐谱，负责这个模块的马可同学连续工作了一整天，将四首歌曲翻译成了程序需要的预设乐谱的形式，整整五百行，是一个非常大的工作量。

4.5 VGA 显示模块（VGADisplay）

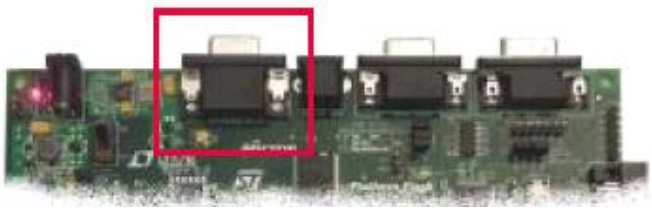
VGA 显示模块是本次课设中的一个扩展功能，然而因为我们小组没有安排好时间，在验收的前一天才开始动手做 VGA 显示，而且是在对 VGA 完全没有了解的情况下，在熬了一个通宵之后，还是实现了 VGA 的扩展功能。

4.5.1 VGA 显示原理

VGA 标准是一种计算机显示标准，最初是由 IBM 公司在1987 年提出的，分辨率是640*480。VGA 接口也叫做 D_Sub 接口，是显卡上输出模拟信号的接口。目前大多数计算机与外部显示设备之间都是通过模拟 VGA 接口连接，计算机内部以数字方式生成的显示图像信息，被显卡中的 D/A 转换器转变为 R、G、B 三原色信号和行、场同步信号，信号通过电缆传输到显示设备中。 常见的彩色显示器一般由阴极射线管(CRT) 构成，彩色由 GRB(Green Red Blue) 基色组成。显示采用逐行扫描的方式解决,阴极射线枪发出电子束打在涂有荧光粉的荧光屏上,产生 GRB 基色，合成一个彩色像素。扫描从屏幕的左上方开始,从左到右，从上到下，逐行扫描，每扫完一行，电子束回到屏幕的左边下一行的起始位置，在这期间，CRT 对电子束进行消隐，每行结束时，用行同步信号进行行同步；扫描完所有行，用场同步信号进行场同步，并使扫描回到屏幕的左上方，同时进行场消隐，并预备进行下一次的扫描。

要实现 VGA 显示就要解决数据来源、数据存储、时序实现等问题，其中关键还是如何实现 VGA 时序。VGA 的标准参考显示时序如图1所示。行时序和帧时序都需要产生同步脉冲(Sync a)、显示后沿(Back porch b)、显示时序段(Display interval c)和显示前沿(Front porch d)四个部分。

S3E 实验平台配备了一个 DB15 标准的 VGA 输出口，用于 VGA 的输出显示。VGA 的 DB15 接口位于实验平台左上角：



视频输出的原理为通过控制 DB1、2、3 脚的电平，同时使用 13、14 脚来控制列扫描和行扫描，实现 VGA 的显示。实验平台最多可显示 64 种颜色。

DB15 接口 1、2、3 的定义为 VGA_RED、VGA_GREEN、VGA_BLUE。列扫描信号为 VGA_HSYNC、行扫描信号为 VGA_VSYNC。

下表为 Sparten 3E 实验板中 VGA 控制和输出信号的管脚分配：

标识名	FPGA 引脚
VGA_RED	H14
VGA_GREEN	H15
VGA_BLUE	G15
VGA_HSYNC	F15
VGA_VSYNC	F14

在本次实验中，我们决定用闪烁的色块来表示地鼠的出现，在显示器上做一个九宫格出来，中心除外的八个块对应八只地鼠，“地鼠出现”用对应位置上的黄色色块来表示。很遗憾的是因为时间紧张，我们忘记拍下效果图，不能在报告中展示，在我们摄制的视频中，会进行展示。

4.5.1 VGA 显示模块代码

首先是模块的接口设计：

```
module VGADisplay(
    Clk,Reset,
    hsync,vsync,
    vga_r,vga_g,vga_b,
     LedOut
   );
```

可以看到，除了 VGA 显示自身必须的 5 个信号以外，输入的只有一组 3bit 的随机数，在 VGA 显示模块中，我们首先将 3bit 的随机数输出信号转换为 8bit 的信号，相当于一个软件实现的 3/8 译码器：

```
123     case(LedOut)
124        3'b000:Region<=8'b00000001;
125        3'b001:Region<=8'b00000010;
126        3'b010:Region<=8'b00000100;
127        3'b011:Region<=8'b00001000;
128        3'b100:Region<=8'b00010000;
129        3'b101:Region<=8'b00100000;
130        3'b110:Region<=8'b01000000;
131        3'b111:Region<=8'b10000000;
132     endcase
```

然后我们确定显示器上 X：200~580、Y：140~460 的区域为显示区域：

```
77   //显示一个小矩形
78  wire e_rdy; //矩形的显示有效矩形区域
79
80  assign e_rdy = ( (xpos>=200) && (xpos<=580) ) && ( (ypos>=140) && (ypos<=460) );
81
```

同时设定 9 个区域控制信号，当扫描信号扫描到某区域内时，区域控制信号就会为真：

```
65  wire a_dis,b_dis,c_dis,d_dis,e_dis,f_dis,g_dis,h_dis,i_dis; //矩形框显示区域定位
66
67  assign a_dis = ( (xpos>=200) && (xpos<=320) ) && ( (ypos>=140) && (ypos<=240) ) &&Region[0];
68  assign b_dis = ( (xpos>=200) && (xpos<=320) ) && ( (ypos>=250) && (ypos<=350) ) &&Region[3];
69  assign c_dis = ( (xpos>=200) && (xpos<=320) ) && ( (ypos>=360) && (ypos<=460) ) &&Region[5];
70  assign d_dis = ( (xpos>=330) && (xpos<=450) ) && ( (ypos>=140) && (ypos<=240) ) &&Region[1];
71  assign e_dis = ( (xpos>=330) && (xpos<=450) ) && ( (ypos>=250) && (ypos<=350) );
72  assign f_dis = ( (xpos>=330) && (xpos<=450) ) && ( (ypos>=360) && (ypos<=460) ) &&Region[6];
73  assign g_dis = ( (xpos>=460) && (xpos<=580) ) && ( (ypos>=140)  && (ypos<=240)) &&Region[2];
74  assign h_dis = ( (xpos>=460) && (xpos<=580) ) && ( (ypos>=250)  && (ypos<=350)) &&Region[4];
75  assign i_dis = ( (xpos>=460) && (xpos<=580) ) && ( (ypos>=360)  && (ypos<=460)) &&Region[7];
```

最后在扫描过程中加入判断语句，实现色彩的输出：

```
83   //r,g,b控制液晶屏颜色显示，背景显示蓝色，矩形框显示红蓝色
84  assign vga_r = valid ? e_rdy : 1'b0;
85  assign vga_g = valid ?  (a_dis | b_dis | c_dis | d_dis | e_dis | f_dis | g_dis | h_dis | i_dis) : 1'b0;
86  assign vga_b = valid ? ~(a_dis | b_dis | c_dis | d_dis | e_dis | f_dis | g_dis | h_dis | i_dis) : 1'b0;
87
```

这样，通过 Region 信号和区域控制信号，我们就选中了地鼠出现的位置，并在对应位置上输出黄色色块，实现了 VGA 显示的功能。

以上就是 VGA 显示部分的全部内容。

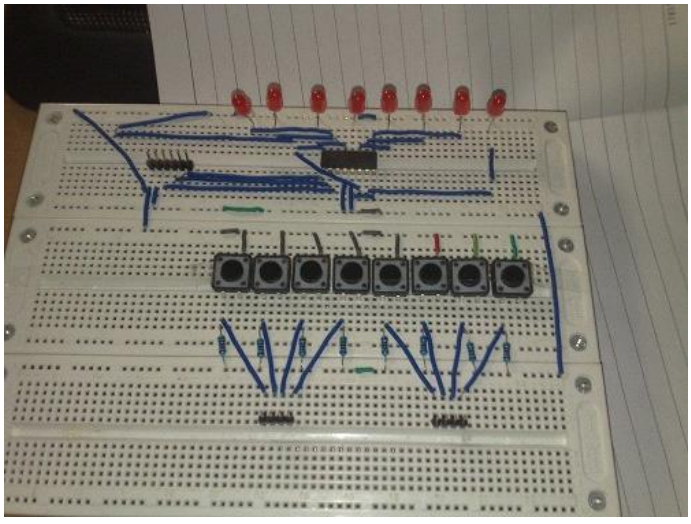
4.6 管脚分配（WAMTop.ucf）

下面是我们在工程中的管脚分配:

```
NET "Button8Bit[0]" LOC = A6;
NET "Button8Bit[1]" LOC = B6;
NET "Button8Bit[2]" LOC = E7;
NET "Button8Bit[3]" LOC = F7;
NET "Button8Bit[4]" LOC = D7;
NET "Button8Bit[5]" LOC = C7;
NET "Button8Bit[6]" LOC = F8;
NET "Button8Bit[7]" LOC = E8;
NET "Clk50M" LOC = C9;
NET "GamePause" LOC = L14;
NET "GameStart" LOC = L13;
```

```
NET "LcdEnable" LOC = M18;
NET "LcdOut[0]" LOC = R15;
NET "LcdOut[1]" LOC = R16;
NET "LcdOut[2]" LOC = P17;
NET "LcdOut[3]" LOC = M15;
NET "LcdRs" LOC = L18;
NET "LcdRw" LOC = L17;
NET "Reset" LOC = N17;



NET "LedOut[0]" LOC = A4;
NET "LedOut[1]" LOC = D5;
NET "LedOut[2]" LOC = C5;
NET "GameMusic" LOC = B4;


# PlanAhead Generated physical constraints

NET "vga_b" LOC = G15;
NET "hsync" LOC = F15;
NET "vga_g" LOC = H15;
NET "vga_r" LOC = H14;
NET "vsync" LOC = F14;
```

五、硬件电路展示



上图就是我们的外围电路，有八个按键和八个 LED 灯，以及一个 3/8 译码器和若干连线。

六、课设小结

在这次“基于 FPGA 的打地鼠游戏机的设计”硬件课设对我们小组的三位成员有着非常大的意义。

◆ 这是我们三位同学第一次以一种团队协作的方式完成一项任务。

◆ 这次课设带给我们三位同学第一次连续两个夜晚通宵做实验的经历。

◆ 通过这次课设，我们对利用 FPGA 控制 VGA 接口、控制 LCD 显示屏、播放音乐的原理有了大致的了解。相比以往任何一次硬件实验，我们参阅了更过的文献，查阅了更多了资料，参考了更多的代码，经历了更多的失败。

这都是这次硬件课设带给我们的体会。但是，更多的，却是遗憾。

我们小组的三位同学在一起时，经常会埋怨对方没有早点着手，到了最后没有时间去实现我们很多很好的想法。比如我们在开题报告中就提出的，可以设置多种老鼠：炸弹鼠（不能打的老鼠）、硕鼠（得分加倍的老鼠）。又比如我们可以在音乐播放的外围电路中加入模拟电路，使得音乐效果听起来更好，甚至可以加上耳机的接口，实现公放/耳机模式的切换。又比如我们可以把八个按键做成手柄的形式，这样玩起来更顺手。再比如我们可以花一番功夫通过 VGA 在显示器上显示出老鼠的动画，使得游戏更有可玩性。有太多的比如，太多的假设，可是在我动手开始写这份报告时，已经没有时间给我们去让我们假设了，课设已经结束了。

这样的遗憾充分暴露了我们的缺点：缺乏整体的计划和大局观。对于一个项目来讲，最重要的就是项目进度的控制。尽管我们的小组里有冰岩的副主管，有学生会主席，但是我们还是没能控制好我们这次课设的进度，导致课设结束前才匆匆忙忙地赶进度。虽然我们有能力，在没有人学过 FPGA，没有人会用 verilog 的情况下，在短短 5 天里亲手敲出近 1700 行代码，实现课设的基本功能和部分扩展功能。但是我们最后还是失败了，我们所完成的和我们所预期的，完全是在两个档次上。

在这次课设中，我们学会了如何利用 FPGA 控制 LCD 显示，如果利用 FPGA 控制 VGA 接口、如何用硬件电路去播放音乐。但是我们小组成员都认为，这都不是最重要

的，我们在本次课设中最大的收获就是：做事严格要求自己，做事要有计划。

非常感谢杨小献老师在课设中对我们的指导和帮助。在最后我们没能拿出很好的结果，感觉有愧于杨老师的关心和悉心指导，非常抱歉。

也感谢所有在本次课设中帮助过我们的老师和同学，没有这些在黑暗中给我们指导的人，我们也不会在短时间内掌握 verilog，了解赛灵思 ISE 开发环境，更不用说完成课设了。

非常感谢！