硬件课设总结报告

课程名称：硬件课程设计

题 目：基于Xilinx的FPGA的打地鼠游戏机

院 系：电子信息工程

班 级：电信1005班

姓 名：周堡俊

学 号：U201012918

指导老师：杨小献

1. 摘要

本文章是我个人对这学期硬件课设的大致总结。内容主要包括本次项目的描述，任务完成情况，任务分工情况的汇报，以及整个系统的框架介绍和各模块的实现原理与实现效果。最后阐明了本次课设中我遇到的问题及收获。

1. 关键词

Basys2，Verilog，打地鼠，游戏控制，音频输出

1. 项目描述

本次课程设计旨在利用Xilinx的basys2设计实现一个功能完善的打地鼠游戏机。主要游戏功能如下：

1) 在屏幕上显示2\*4的八个地鼠，与之对应用8个按键进行控制。每一次随机出现一个地鼠，在地鼠出现的时间内要求按到对应的按键，若按到则加1分，且地鼠消失；若按错按键则命数扣一条。

2) 设定初始命数为5，每连续得分达到10分便加1条命，失误一次扣一条命，命数为0时游戏结束。

3) 游戏分三个难度级别，每个级别地鼠出现的速度不同，级别越高，速度越快，地鼠消失的速度越快。得分达20分，进入第二个难度，达50分进入第三个难度。

4) 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。通过VGA显示屏进行相应的显示。

5) 每个难度级别对应产生不同速率的音乐，与地鼠出现的频率节奏相当。

**扩展功能：**

1. 每一关总时间40秒，必须在规定时间内达到下一关，否则时间耗尽后游戏结束。进入下一关后时间重置。
2. 通过VGA显示地鼠位置，得分，倒计时，命数等信息。界面友好，并有简单动画效果。
3. 按键音效，打到和打错的音效不同。游戏结束会播放一段音乐。
4. 在第三关能同时出现两只地鼠。
5. 设计与实现背景

硬件环境： Digilent Spartan 3E-Starter开发板

器件清单： VGA显示器；

双声道音响；

基于面包板的输入电路

软件环境： ISE Design Suite 14.4

系统环境： Window7

编程语言： VHDL和Verilog HDL混合

1. 硬件系统设计方案与系统框图

使用XILINX XC3S500E Spartan-3E FPGA芯片作为中央控制器控制整个系统，将整个系统划分为几个模块：按键输入模块，二进制转十进制模块，分频模块，速度控制模块，随机数产生模块，得分模块，VGA显示模块，背景音乐模块，音效模块等模块。各模块的作用分别为：

按键输入模块：通过按键实现人机交互，完成游戏机最基本功能。

二进制转十进制模块：系统内部各变量运算用的是二进制，而输出得转换为十进制一位一位输出，因此需要二进制转十进制模块。

分频模块：在FPGA开发板上提供的时钟频率为50Mhz，而系统各模块工作所需要的频率一般不同（如音频输出模块需要1K左右频率，而游戏控制模块需要100HZ左右频率），所以要使用分频模块将50Mhz的频率分成所需要的频率。

速度控制模块：用以控制系统的工作频率以适应不同难度的级别，得分模块在对得分满足一定条件的时候反馈给速度控制模块一个参量，使得速度控制模块能够根据这个参量改变接入的脉冲频率，从而改变游戏的难度。

随机数产生模块：产生一个0~7的随机数，通过这个随机数对地鼠数组进行控制，使这个随机数对应的位置出现地鼠。

地鼠模块：地鼠模块的核心是一个数组，每一元素分别对应于该地鼠的剩余寿命，每隔一个时钟，对该值减1，直至减为零，则地鼠消失。产生地鼠时则相当于给该数组赋一个初值。通过对该数组操作，可实现地鼠产生与消失效果。

得分模块：将地鼠数组与按键送来的按键数值进行比较，该按键对应的数组元素数值大于0，则说明该位置有地鼠，得分加1。若为0，则说明该位置没有地鼠，命数减1，每连续得10分，命数加1。当得分达到20分和50分时，反馈给速度控制模块，改变地鼠出现和消失速度，增加难度。

VGA显示模块：根据前面模块送来的数据进行显示，通过图形化界面，显示当前地鼠位置，得分，倒计时，命数，历史最高分，难度等信息。

背景音乐模块：在系统运行的时候播放音乐。

音效模块：打中和打错地鼠时分别赋予不同频率的脉冲驱动喇叭发出不同频率的声音。游戏结束时也会播放音效。

我们最终的系统功能框架如下：

VGA输出

背景音输出

按键音效

游戏控制模块

100Hz

Clk(50MHz)

随机数模块

Mouse[7:0]

输入模块

Reset、pause、start按键响应

分频模块

地鼠控制

按键响应

地鼠消失

地鼠产生

八个打地鼠按钮的响应

1. 团队分工

在本次项目中，我主要负责游戏控制模块和音频输出模块的编写。徐臻（同组）负责VGA的显示模块，和顶层模块的编写。明正莹（同组）负责随机数算法和按钮输入电路的搭建。

1. 关键算法与模块描述
2. 游戏控制模块

本模块用来实现游戏的基本控制及逻辑判断，包括地鼠的产生模块，地鼠的消失模块，计分模块，按键响应模块，分频模块等。各模块之间的关系上面的系统框图已表明，不赘述。下面分别介绍一下这几个模块的功能及实现方法。

1.分频模块。

本开发板用的时钟频率为50MHz，而显然主游戏控制模块是用不到这么高频率的，如地鼠产生（第一难度的）只需要0.3Hz的频率，而按键采样模块也不需要这么高频率（如果采样频率过快，如就用50MHz采样，由于芯片的计算时间跟不上按键的采样频率，则必会产生时序上的错误）。所以在本模块的开头先对50MHz频率进行分频，分出100Hz频率。

分频采用以下的方法：

always @(posedge clk)

begin

if(clk100\_count==249999)//每0.005秒执行一次，即频率200Hz

begin

clk100\_count<=0;

clk100=~clk100;//clk100翻转一次。Clk100周期应为2倍的0.005s，即0.01s，频率为100Hz

end

else clk100\_count<=clk100\_count+1;

end

2.地鼠的产生和消失。

在此模块中先声明一个数组mouse[7:0]，分别对应各位置地鼠的寿命（存在时限）。这样地鼠产生和消失实现起来就变得简单了：即只需要隔一段时间（通过分频实现）将mouse数组某一位赋一个初值（比如说5），再每隔一段周期给mouse每一位减1，直至减为零。而显示模块只需对mouse每一位进行判断，为0则地鼠不显示，不为零则显示。这样以上述为例，则该位置过5个周期地鼠会自动消失。这样做有两个好处，即可对每个地鼠的寿命单独赋不同的值，以实现不同难度下，地鼠存在时长不同，而不用专门去更改分频数。也可实现在同一时刻有两只地鼠存在的状况（只需使地鼠出现的周期小于地鼠存在的时长），而不会遇到要考虑什么时候消除哪只地鼠的窘境。

这样游戏速度控制模块的功能也可以简单实现了，只需在不同的难度下，更改产生模块的分频数及赋的寿命初始值，而消失模块完全不用变。

其实这样也会产生一个问题，即产生模块和消失模块的时间同步问题。假如两者不同步，即假设一只地鼠产生0.2秒后才执行第一次消失模块，另一只地鼠产生0.5秒后才执行第一次消失模块，这样就算两只地鼠寿命都为5，他们的存在时长差也相差0.5-0.2=0.3s。如果不加控制，两只地鼠相隔越长，他们寿命误差越大。解决方法是，将产生模块的执行周期设定为消除模块周期的整数倍。

简化后的实现过程如下：

//产生模块

if((appearCount>=299&&mission==0)||(appearCount>=249&&mission==1)||(appearCount>=149&&mission==2))//根据难度选择不同的分频，实现不同的地鼠出现速度

begin

appearCount<=0;

//根据难度不同，地鼠初始寿命也不同，即各难度地鼠存在时间不同

if(mission==0)

mouse[randNum]<=5;

else if(mission==1)

mouse[randNum]<=4;

else if(mission==2)

mouse[randNum]<=3;

end

else appearCount<=appearCount+1'b1;

//----------------------------------------------------------

//消失模块

if(disappearCount>=49)//恒定时间执行一次消失模块

begin

disappearCount<=0;

for(i=0;i<8;i=i+1)

begin

if(mouse[i]>0) mouse[i]<=mouse[i]-1;//给每只地鼠寿命减一，直 至0为止

end

end

else disappearCount<=disappearCount+1'b1;

3.按键响应模块。

游戏过程中要对系统进行实时控制，则需要对按键进行实时响应。本模块中对按键响应频率为100Hz，即0.01s对所有按键进行一次分析。

各按键分析逻辑如下：

Reset：如果按下将状态位置为停止，所有功能停止运行。再对所有变量重新赋初值（随机数和历史最高分除外）。

Pause：如果游戏正在运行则将状态位置为暂停，则所有功能停止运行。如果游戏已暂停，则将状态位设为运行状态。

Start：如果游戏暂停，则将状态位设为运行状态。如果游戏已结束，则对所有变量重新赋初值（随机数和历史最高分除外），再将状态位设为运行状态。

Btn[7:0]（打地鼠按钮）：如果按下则查找对应mouse[7:0]位置的值，如果为0，则生命值减一；如果不为0，则分数加一。

按键模块有一个问题一定要注意，即按键的按下时长问题。如果不管这一问题，只是对当前按键的值进行判断，执行相应的任务，这很有可能因为按钮按下的时长过长（超过0.02s），本模块对同一次按下动作进行了两次或更多次处理，导致很严重的问题。我的解决方案是用一个变量存上一次按键响应是按键的状态（按下、松开），当本次按键值与上一次的值进行比较，如果不同才进行处理。否则不处理（避免对同一次按键处理两次）。

4.计分模块。

此模块除了以上提到的按对按钮加一分外，还有连续得十分加一命和到达指定分数进入下一难度的功能。算法比较简单，只列出，不再详细解释。

以连续得十分加一命为例：

if(lifebuff==9) life<=life+1;

if(打中地鼠)

begin

score<=score+1;

lifebuff<=lifebuff+1;

end

else if(打错)

begin

if(life) life<=life-1;

lifebuff<=0;

end

5.游戏结束判断模块。

时间为0或生命值为0时，游戏结束。其原理只是改变一下游戏的状态位，不赘述。

1. 音乐输出模块

此模块负责背景音乐和各音效的输出。

通过开发板IO口输出一个频率可变的正方波，来驱动外接喇叭发声。由于我的设计是用两个完全不同的分模块分别输出背景声音和游戏音效，而两个模块（不在同一个always@里）是无法同时对同一端口输出，所以我用了两个端口来分别输出背景音和音效（一个双声道音响的两个声道）。

各音符对应的频率见下表：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 音符 | 对应频率(Hz) | 半周期(us) |
| 低音 | 1 | 261.6255653 | 1911.128216 |
| 1.5 | 277.182631 | 1803.864832 |
| 2 | 293.6647679 | 1702.621678 |
| 2.5 | 311.1269837 | 1607.060866 |
| 3 | 329.6275569 | 1516.863471 |
| 4 | 349.2282314 | 1431.728466 |
| 4.5 | 369.9944227 | 1351.371722 |
| 5 | 391.995436 | 1275.525055 |
| 5.5 | 415.3046976 | 1203.935334 |
| 6 | 440 | 1136.363636 |
| 6.5 | 466.1637615 | 1072.584446 |
| 7 | 493.8833013 | 1012.384907 |
| 中音 | 1 | 523.2511306 | 955.5641082 |
| 1.5 | 554.365262 | 901.9324159 |
| 2 | 587.3295358 | 851.3108391 |
| 2.5 | 622.2539674 | 803.5304332 |
| 3 | 659.2551138 | 758.4317353 |
| 4 | 698.4564629 | 715.8642329 |
| 4.5 | 739.9888454 | 675.6858608 |
| 5 | 783.990872 | 637.7625274 |
| 5.5 | 830.6093952 | 601.9676672 |
| 6 | 880 | 568.1818182 |
| 6.5 | 932.327523 | 536.2922231 |
| 7 | 987.7666025 | 506.1924535 |
| 高音 | 1 | 1046.502261 | 477.7820541 |
| 1.5 | 1108.730524 | 450.9662079 |
| 2 | 1174.659072 | 425.6554196 |
| 2.5 | 1244.507935 | 401.7652166 |
| 3 | 1318.510228 | 379.2158677 |
| 4 | 1396.912926 | 357.9321164 |
| 4.5 | 1479.977691 | 337.8429304 |
| 5 | 1567.981744 | 318.8812637 |
| 5.5 | 1661.21879 | 300.9838336 |
| 6 | 1760 | 284.0909091 |
| 6.5 | 1864.655046 | 268.1461116 |
| 7 | 1975.533205 | 253.0962267 |

根据每升高一个八度，频率扩大一倍，可推出其他音符的频率。

由之前提到的分频算法，分频数应为（用50MHz分）50M/(2\*f)。

由此公式得各音符分频数（部分）为：

1: 47800

1.5: 45125

2: 42588

2.5: 40191

3: 37935

4: 35815

4.5: 33828

5: 31886

5.5: 30119

6: 28408

6.5: 26823

7: 25328

1+: 23899

1.5+: 22562

2+: 21294

……

根据以上分频数，我写了一个模块doRiMi(note,clk,wave)专门用来输出各音符，输入音符号，则会得到一个波形，将此波形接至外接喇叭，则会放出相应音调。

其他两个背景音模块和音效模块都是调用此dorimi模块来完成的。

1.背景音模块。

本模块基本思想是将一首曲子的音符和音符长度分别记录下来，然后按一定速率循环播放，每个音符之间还应有一小段间隔。以此最小间隔为一时钟周期，则将一拍长度设为19，四分之三拍长度设为14，二分之一拍长度设为9，四分之一拍长度设为4。

而我记录音符和音符长度是直接用两个函数来实现的。即输入一个音符号则一个函数（getNote()）返回音符值，另一个函数（getLength()）返回音符长度。

函数实现示例：

function [3:0] getNote;

input [6:0]i;

case(i)

0:getNote=7;

1:getNote=2;

2:getNote=7;

3:getNote=7;

……

81:getNote=7;

82:getNote=7;

83:getNote=7;

84:getNote=0;

default:getNote=0;

endcase

endfunction

音符循环播放算法如下：

always @(posedge clk100)

begin

if((count>=3&&mission==0)||(count>=2&&mission==1)||(count>=1&&mission==2))//根据难度选择不同的播放速度

begin

count<=0;

if(situation==0)//正在游戏中

begin

if(i<85)//共85个音符

begin

j=j+1;

if(j>getLength(i))//一个音符播放完了

begin

j=0;

i=i+1;

note<=0;//停顿一个时钟周期（note=0则不输出波形）

end

else note<=getNote(i);//否则继续播放这个音符

end

else i=0;//整首曲子播放完了，再从头开始

end

else if(situation==3) note<=0;//如果游戏结束，播放停止

else

begin

i=0;//回到第一个音符

j=0;

note<=0;

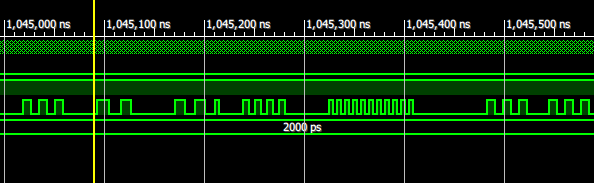
end

end

else count<=count+1;

end

仿真波形如下：



可清晰看到波形随着时间，频率发生变化。

2.音效模块

音效模块在按键按下时和玩家死亡时会播放音效。播放过程和背景音效模块除了声音不会循环播放外，区别不大，不再赘述。在此想详细说明一下音效的触发过程。

在主控模块里面设三个触发信号hitSound（打中音效）、faultSound（没打中音效）、loseSound（死亡音效）。当判断为打中时，将hitSound置1，并在下一时钟开始后置为0（相当于持续长度为一个时钟周期的冲激信号），当音效模块每检测到一次这一冲激，就播放一次打中音效，而不需要反馈给主控模块。其他音效触发类似。这种触发方式（无反馈触发）相比有反馈触发有不少优点，即避免了主控模块和音效模块之间交互的麻烦，从双向通信变为单向通信，提高了效率。

主控模块中触发算法如下：

always @(posedge clk100)

begin

hitSound<=0;

faultSound<=0;

loseSound<=0;

……

if(打中) hitSound<=1;

if(没打中) faultSound<=1;

if(死亡) loseSound<=1;

end

1. 项目中遇到的问题及解决方案

本次项目中遇到的大部分问题以上已经提及到。当然还遇到过一些其他的问题，在这里总结一下：

1. 模块响应频率并不是越快越好的，某些模块响应频率不能太高，如按键输入分析模块，和LCD显示模块。之前我直接使用50MHz频率对按键事件进行处理，结果逻辑完全不正确，后来改为100Hz处理就对了，我想可能是因为频率过快，芯片的计算时间跟不上按键的处理频率，导致产生时序上的错误。至于之前调试时，用LCD显示结果，当刷新频率过快时，会产生乱码，但一旦把频率降低，又显示正常。这个问题的原因我还没找到。

2.按键模块得考虑按键的按下时长问题。之前我写打地鼠按钮模块时，对一个按键进行判断，只是判断他是否被按下，按下了则执行什么什么语句。结果每次不管按哪个键，都会被扣命，而且一下会扣多条命。后来我用一个变量存上一次按键响应时按键的状态（按下、松开），当本次按键值与上一次的值进行比较，如果不同才进行处理。否则不处理（避免对同一次按键事件处理多次）。但后来我写暂停按钮响应模块时又犯了同样的错误，但这次错误的现象没那么显然（表现为暂停按键不灵，即有时暂停有效，有时又没效），我还以为是按钮坏了。但多次检查后，发现仍旧是按键的按下时长没考虑。当然我之前进行软件编程时完全都不用考虑这个问题，因为软件上按钮的按下都通过一个监听器就可以获取，而按钮按下时长根本不用编程者考虑。这个问题我第一次遇到，而且也只会在硬件编程上才会遇到。

3.同时存在两只地鼠的问题。我之前只用一个八位二进制数表示地鼠（0表示不存在，1表示存在）。可以完美实现每时刻只存在一只地鼠或两只地鼠同时产生同时消失，只需在一段时间后对此二进制数清零，然后再随机给其中一位或两位赋值1即可。但显示的效果总感觉不太好。我的设想是每一只地鼠都是独立的，一只产生了还没消失，另一只又出来了：这种参差不齐的效果。这个技术难点在于怎么用一个简单的算法实现每个地鼠相互独立，一只地鼠的出现与消失不会影响到另一只地鼠的产生和消失，而不是只能同时产生同时消失。后来我借鉴高级语言里类的思想，每一只地鼠为类的一个实例存在一个数组里，然后对这个数组统一管理。后来便改成了现在这个样子，虽然相比以前更耗资源，但效果总算令人满意。

4.我后来将地鼠产生与消失分割成两个模块后（将地鼠由一个二进制数改成一个数组后，不得不将产生与消失模块分开），产生模块和消失模块的时间同步问题也成了一个需要注意的问题。之前我没发现这个原因，然后总感觉即使在同一个难度下，每只地鼠存在时间都不同，后经同学的提醒，才发现是时间同步的问题。假如两者不同步，即假设一只地鼠产生0.2秒后才执行第一次消失模块，另一只地鼠产生0.5秒后才执行第一次消失模块，这样就算两只地鼠寿命都为5，他们的存在时长差也相差0.5-0.2=0.3s。如果不加控制，两只地鼠相隔越长，他们寿命误差越大。所以解决方法是，将产生模块的执行周期设定为消除模块周期的整数倍。在按键响应时也得注意这个问题，假如按键响应判断玩家打中地鼠了，则该地鼠消失，同时产生一新的地鼠。但这个产生新地鼠的时间也必须与地鼠消失模块时间同步，否则这个新产生的地鼠寿命会明显大于其他地鼠的寿命（最多可达0.5秒）。

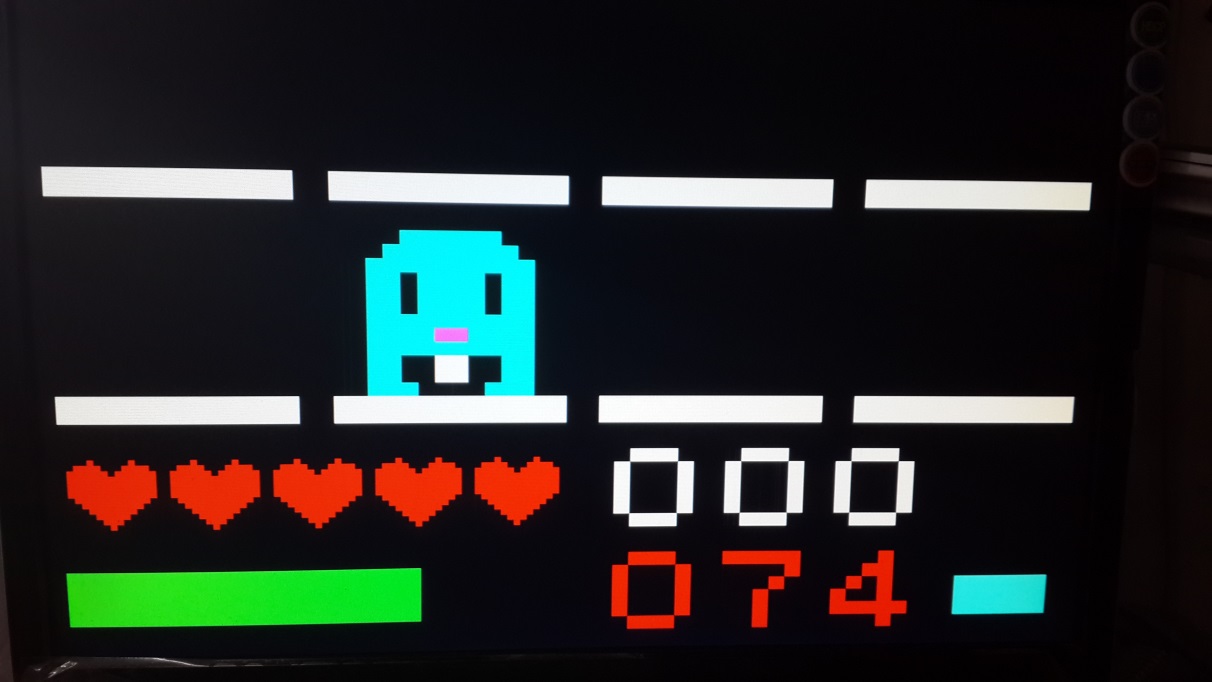
5.之前音效的激发也遇到过问题。以前写软件程序时，实现同样的功能（音效的激发）我肯定会这样写：先在主程序产生一个信号（如将某标志位置为1），子程序收到后再反馈一个信号（将此标志位再置为0），主程序响应后再执行后续任务。但在硬件电路里这样写会很麻烦。因为在硬件电路里一个信号要么是输入信号，要么是输出信号，（虽然也可以是inout型，但在系统内部逻辑模块里这么定义很容易出莫名其妙的错误，一般只在系统和外设间才用）所以要实现此标志位又能读入又能被输出（更改值）是很困难的。后来我借鉴IP协议，即无反馈激发，主控模块产生一个冲激信号（只存在于一个时钟周期，下一周期会自动重置），从模块里收到此信号后自动执行任务，不需要反馈。这样便可有效解决主控模块和音效模块之间通信的问题。

1. 系统的组装连接及运行结果



完整项目连线图如上。

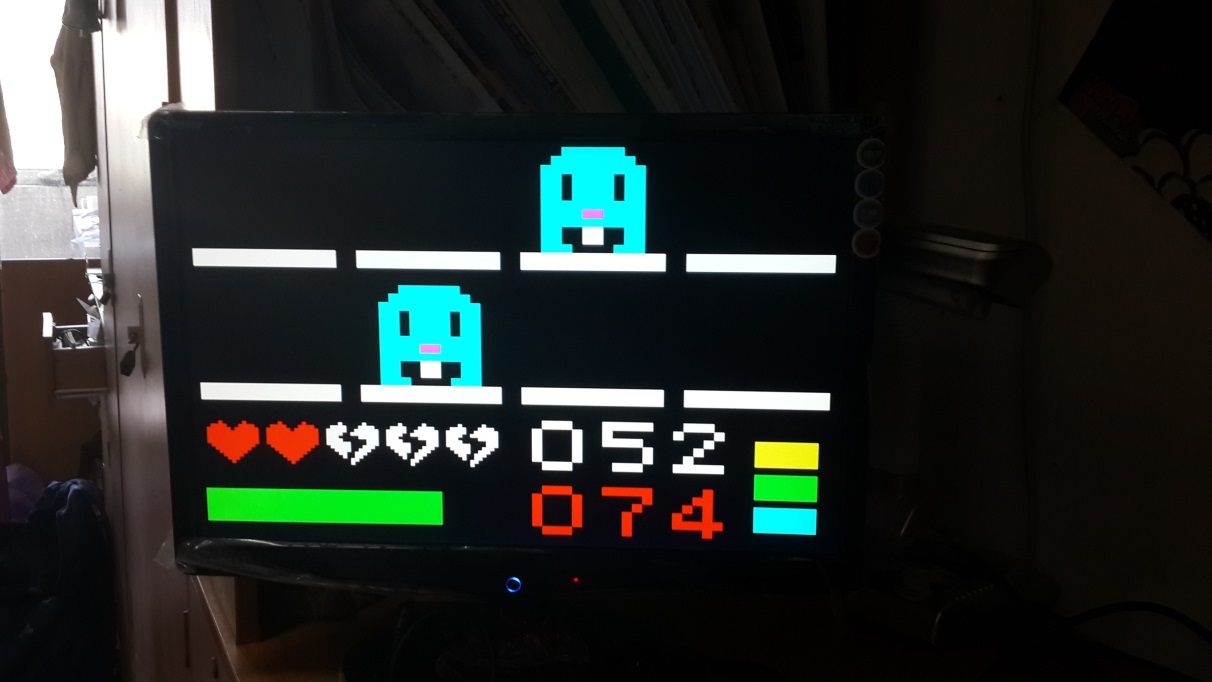
运行结果与原设想相同，运行很稳定。效果如下：



第一难度



第二难度



第三难度

1. 项目总结与收获

经过两个多月的准备和一个多星期的调试和完善，我们组的基于Xilinx的FPGA的打地鼠游戏机终于完成了。现在它已基本具备了任务指标中的所有功能：设有开始、暂停、结束功能，地鼠可以根据难度按时的产生与消失，打中地鼠是可加一分，打错会扣一条命。达到一定分数进入下一难度，连续得10分可回复一条命，生命值为零游戏结束，游戏设有背景音乐等。另外出于游戏性和易玩性的考虑，我们还附加了如下功能：通过VGA显示游戏信息，使游戏更直观更具吸引力。加入倒计时的设定，增加游戏难度。实时播放游戏音效，使游戏交互性更强，玩家能更代入其中。第三关可同时存在两只地鼠，游戏内容更丰富。

这次课设给我最大的感受就是团队协作真的很重要。由于我写的是主模块，而徐臻写的是显示模块，他写VGA模块时就总是来找我提意见，说这个数据类型得改改，他好写输出。而我主模块内部为了便于运算，都用的是二进制数，而要便于显示的话应转化为十进制，有时候还得进行一定编码。所以我得对于不同的变量，根据徐臻的意见，进行不同的转化。虽然有时候很麻烦，但我很高兴，因为每当显示屏上成功显示某一数值时，我总感觉有我的功劳。另外在音乐模块编写时，徐臻也给与了我很大的帮助。包括那个乐谱，也是他帮我修改的。

我第二大感受就是硬件编程和软件编程真的很不一样，最明显的一点就在资源利用上。开发板资源实在是有限，之前我们没有注意这个问题，开始是各模块分开调试，终于调成后，将各模块组建在一起，结果软件报错了，提示“too many slicers”我们这才发现资源的重要性。因为我们之前只接触过软件编程，而电脑资源相对比较富裕，我们平时编程也都没怎么在意资源的问题。于是最后这几天我们的重点都在程序优化上，如能节约点寄存器就节约点寄存器，将各分频模块合并（每用一次分频得耗费大量寄存器），以及随机算法的优化，重复的语句用函数代替等等。现在我们的资源使用率在50%左右，代码段也减少了将近1/3。

另外，经过这两个星期没日没夜的强化训练，我已养成了硬件模块化编程的习惯。也认识到了模块化的优点，它不仅可以去除很多重复的工作（如那个doRiMi音符模块的设置，使背景音和音效模块代码变得大为简洁），还能使代码条理更清晰。这个编程习惯不管是对软件大工程的编程还是硬件编程都非常重要。

1. 参考文献

《lab1\_4bitsLEDs\_201205015\_ISE13.1使用指导.pdf》

《Spartan3E开发板用户说明0108.pdf》

《Spartan3E板卡资料.doc》

1. 附录

1、芯片管脚连线

NET "mclk" CLOCK\_DEDICATED\_ROUTE = FALSE;

NET "btn[7]" LOC = F7;

NET "btn[6]" LOC = E7;

NET "btn[5]" LOC = C5;

NET "btn[4]" LOC = D5;

NET "btn[3]" LOC = B6;

NET "btn[2]" LOC = A6;

NET "btn[1]" LOC = A4;

NET "btn[0]" LOC = B4;

NET "led[0]" LOC = F12;

NET "led[1]" LOC = E12;

NET "led[2]" LOC = E11;

NET "led[3]" LOC = F11;

NET "led[4]" LOC = C11;

NET "led[5]" LOC = D11;

NET "led[6]" LOC = E9;

NET "led[7]" LOC = F9;

NET "OutBlue" LOC = G15;

NET "OutGreen" LOC = H15;

NET "OutRed" LOC = H14;

NET "fpgabtn[0]" LOC =H13;

NET "fpgabtn[1]" LOC =K17;

NET "fpgabtn[2]" LOC =D18;

NET "HS" LOC = F15;

NET "mclk" LOC = C9;

NET "VS" LOC = F14;

NET "soundWave" LOC = D7;

NET "BGM" LOC = C7;

NET "VS" SLEW = FAST;

NET "HS" SLEW = FAST;

NET "OutBlue" SLEW = FAST;

NET "OutGreen" SLEW = FAST;

NET "OutRed" SLEW = FAST;

2、源码清单

bgm.v 背景音乐控制模块

bin2dec.v 二进制转十进制，然后取十进制百位、十位、个位

doRaMi.v 输出一个音符

game.v 游戏控制

sound.v 游戏音效控制模块

HitTheMouse.vhd 顶层模块

led\_Out.vhd led灯输出模块

VGA\_Display.vhd VGA显示模块