综合实验报告

一、设计内容

本人设计了一个打地鼠小游戏，在 FPGAOL 上实现，在该游戏中，七个 LED 灯管的闪与灭代表地鼠的探头与缩头，玩家快速选定开关并按下按钮代表锤击，只有在一定时间内选定正确位置并按下按钮，才能得分。LED 灯管随机闪烁，每按一次按钮就会刷新一次，代表着地鼠的随机探头，一轮游戏时长为 30 秒，总得分显示在数码管上。本游戏支持查看最近四次游戏得分和历史最高分。

二、设计思路

1.确定状态机状态数

本人设计了一个包含 4 个状态的状态机，00：菜单（初始界面）；01：游戏；10：近四次分数；11：历史最高分。显然，这四个状态足以满足该游戏设计的需求。于是，我们将 sw 开关的前两位：sw[0]、sw[1]作为转换状态的入口，按下 button 时，根据 sw[0~1]的值改变状态。综上，这是一个 Mealy 型状态机。

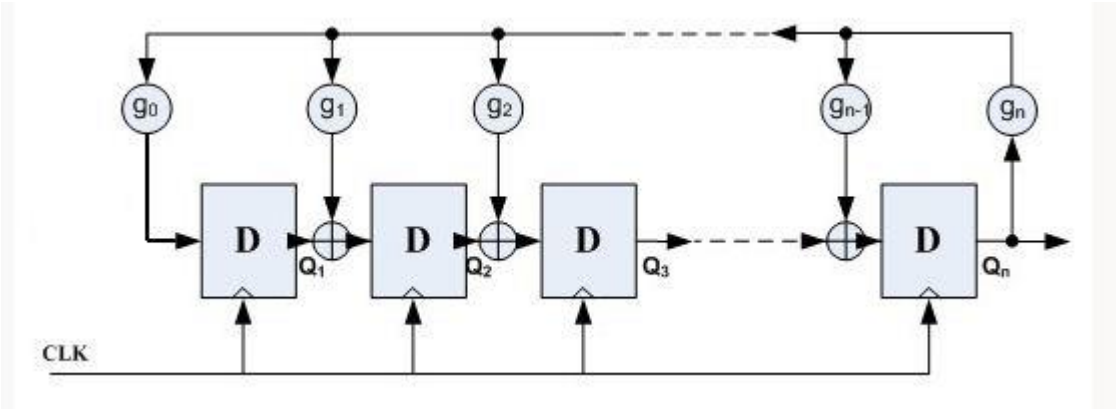
2.对 button 信号的处理

由于需要用到 button 来进行各种操作，如选取模式和传达敲击指令等，所以，我们需要对 button 信号做处理，在本设计中做了两种处理：去毛刺和取边缘信号，这两种处理都是在前面的实验中学习应用过的，故不在此赘述。

3.取随机数

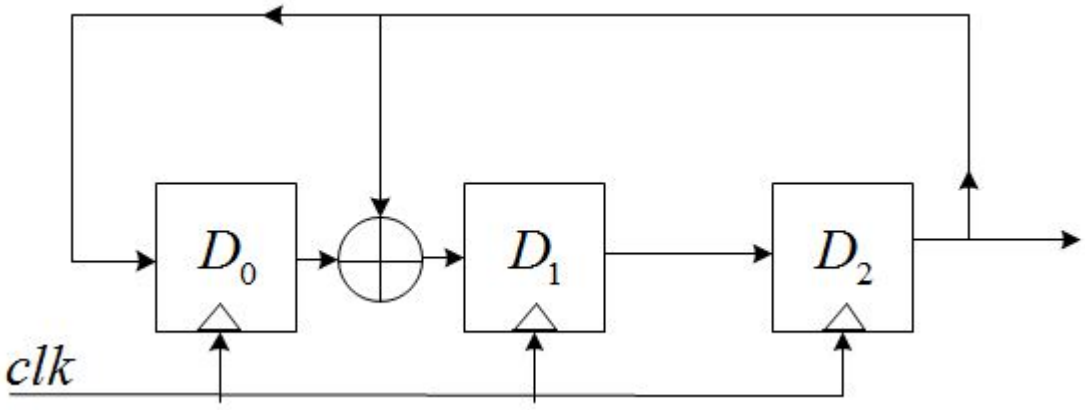
打地鼠游戏中，地鼠从哪个洞口探头是随机的，因此，本游戏中的 LED 灯闪烁也应当是随机的，那么，怎么实现这种随机功能呢?

本设计中，使用的是由 LFSR(线性反馈移位寄存器)引出伪随机数的方法。LFSR 是由 n 个 D 触发器和若干个异或门交替线性串接组成的，如下图:



其中，gn 为反馈系数，取值只能为 0 或 1，取为 0 时表明不存在该反馈之路，取为 1 时表明存在该反馈之路；n 个 D 触发器最多可以提供 2^n-1 个状态(不包括全 0 的状态)，为了保证这些状态没有重复，gn 的选择必须满足一定的条件。

下面以 n=3，g0=1，g1=1,g2=0,g3=1 为例，说明 LFSR 的特性，具有该参数的 LFSR 结构如下图：



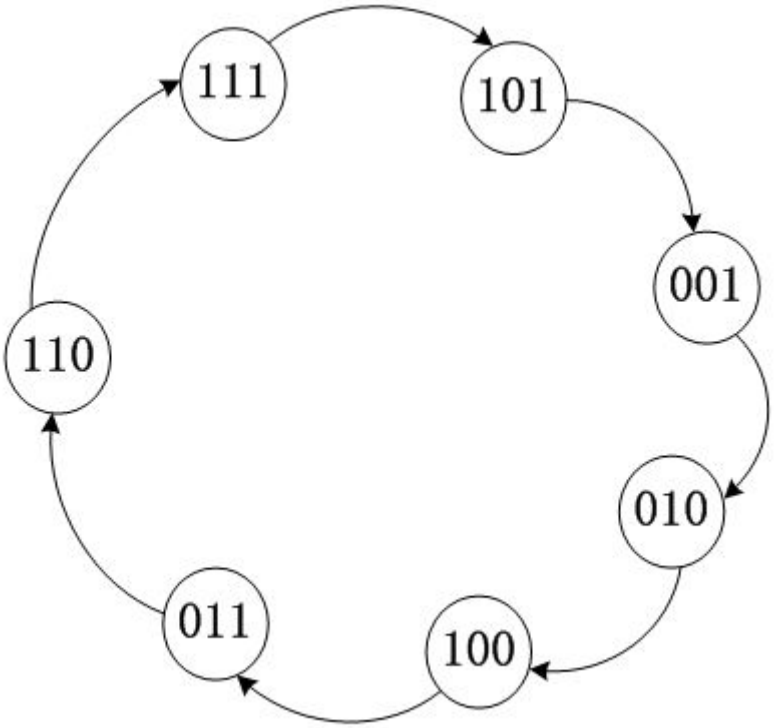
假设在开始时，D2D1D0=111(seed)，那么，当时钟到来时，有：D2=D1_OUT=1；

D1=D0_OUT^D2_OUT=0；

D0=D2_OUT=1；

即 D2D1D0=101；同理，又一个时钟到来时，可得 D2D1D0=001 ……

画出状态转移图如下：



从图可以看出，正好有 2^3-1=7 个状态，不包括全 0；

理解了上图，至少可以得到三条结论：

1)初始状态是由 SEED 提供的；

2)当反馈系数不同时，得到的状态转移图也不同；必须保证 gn===1,否则没有反馈

3)D 触发器的个数越多，产生的状态就越多，也就越“随机”

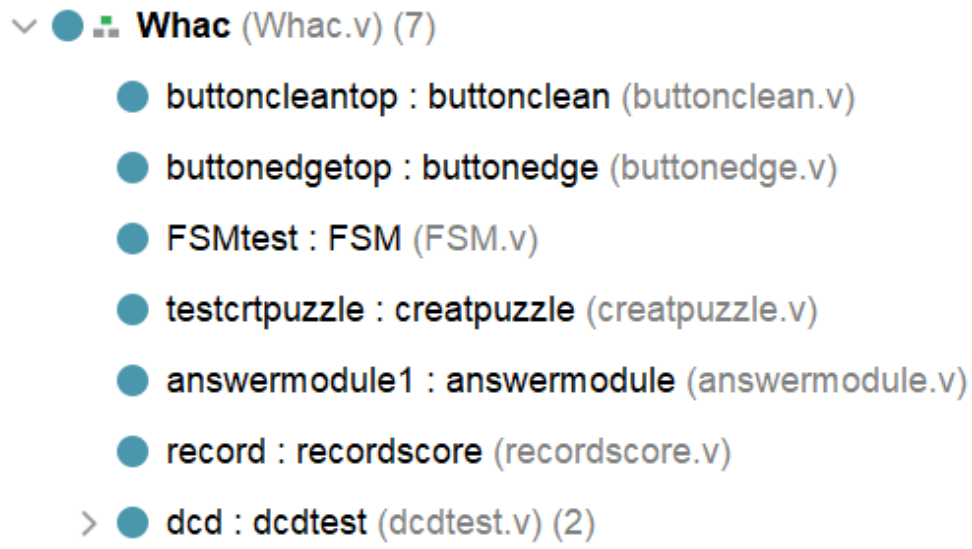
基于以上原理，我们就可以在 FPGA 上实现伪随机数的生成了，具体的代码在稍后展示。

4.在数码管上同时显示多个数字

使用前面实验中已经学习应用过的时分复用的原理。不做赘述。

三、代码实现

分七个模块编写该游戏的 Verilog 代码，再在顶层代码中引用串联，最终实现了该游戏。模块分布如下：



各模块功能为：

· Whac：顶层模块

· buttonclean：去毛刺模块

· buttonedg：取边沿信号模块

· FSM：状态机模块

· creatpuzzle：生成“地鼠”模块

· answermodule：“锤子”信号接受模块

· recordscore：得分记录模块

· dcdtest：数码管显示模块

下面对各模块的关键点做一些阐述：

· Whac

该模块是顶层设计模块，无需阐述太多，下面给出定义的各种信号，注释已在图中：

```
input clk,
input button,
input [7:0] sw,
output  [7:0] led,
output  [2:0] sel,
output  [3:0] out
```

```
wire buttonclean;
wire signal;   //button信号
wire [1:0] mod;  //模式
wire [7:0] game;  //puzzle
wire [3:0] data;  //分数
wire [15:0] near;  //最近四次成绩
wire [3:0] top;  //最高分
wire overflag;  //游戏结束标志
wire answerflag;  //回答标志
```

· buttonclean

该模块是去毛刺模块，在前几次实验中多次用到，故不多作阐述。

· buttonedg

该模块是取边沿信号模块，同上，故不多作阐述。

· FSM

对状态机的分析在上文中已经给出，下面给出具体代码。必要注释已在图中：

```
module FSM(
input clk,
input signal,
input [1:0] sw,
output reg overflag,
output reg [1:0] mod  //模式
    );
reg [35:0] count;
reg [35:0] cycle;
parameter state00 = 2'b00;//菜单界面
parameter state01 = 2'b01;//游戏模式
parameter state10 = 2'b10;//最近四次游戏的分数
parameter state11 = 2'b11;//最高分记录
initial
begin
    count = 0;
    mod = 4'b00;
    cycle = 36'b0000_1011_0010_1101_0000_0101_1110_0000_0000;//游戏时长30s
    overflag = 0;
end

always@(posedge clk)
begin
  case(mod)
    state00://菜单
    begin
    if(signal==1)
      mod <= sw;
    end

    state01://游戏
    begin
      if(count == cycle)
      begin
        mod <= state00;
        count <= 0;
      end
      else
      begin
        count <= count + 1;
        overflag <= 0;
```

```
      end
      if(count == cycle - 36'b0000_0000_0000_0000_0000_0000_0000_1000_0000)//略小于cycle的时候
      begin
        overflag <= 1;
      end
    end

    state10://分数
    begin
    if(signal==1)
      mod <= state00;
    end

    state11://最高
    begin
    if(signal==1)
      mod <= state00;
    end

    default mod <= state00;
  endcase
end
endmodule
```

· creatpuzzle

该模块是生成“地鼠”模块。在该模块中，上文阐述的产生(伪)随机数的方法就要发挥作用了。原理已在上文详细阐述了，下面给出代码细节：

```
module creatpuzzle(
input clk,
input [1:0] mod,
input answerflag,//回答题目的标识
output reg [7:0] led,//显示LED上的
output reg [7:0] game//真实的puzzle
    );
reg [35:0]cycle;//游戏周期
reg [35:0] count;//计数器
reg [31:0] randomnum;//用来产生随机数
reg [35:0] flashflag;//用来判断是否闪烁
reg cntnum;//闪烁模式用来判断
wire seed = randomnum[31]^randomnum[30]^randomnum[29]^randomnum[28]^randomnum[27]^randomnum[26]^randomnum[25]^randomnum[24]^
            randomnum[23]^randomnum[22]^randomnum[21]^randomnum[20]^randomnum[19]^randomnum[18]^randomnum[17]^randomnum[16]^
            randomnum[15]^randomnum[14]^randomnum[13]^randomnum[12]^randomnum[11]^randomnum[10]^randomnum[9]^randomnum[8]^
            randomnum[7]^randomnum[6]^randomnum[5]^randomnum[4]^randomnum[3]^randomnum[2]^randomnum[1]^randomnum[0]; //伪随机
parameter state00 = 2'b00;//菜单界面
parameter state01 = 2'b01;//游戏模式
parameter state10 = 2'b10;//最近四次游戏的分数
parameter state11 = 2'b11;//最高分记录
parameter fmag = 36'b0000_0000_0001_0111_1101_0111_1000_0100_0000;//0.25s
parameter fend = 36'b0000_0000_0101_1111_0101_1110_0001_0000_0000;//1s
initial
begin
    cycle = 36'b000000_01110_11100_11010_11001_01000_00000;//puzzle周期为5秒
    randomnum = 32'b0011_1110_1100_0100_0111_1010_0000_0010;//任意一个数
    count = 36'b0000_0001_1010_1101_0010_0111_0100_1000_0000;//4.5s,即0.5s后产生第一个puzzle
    game = 0;
end

always@(posedge clk)
begin
  case(mod)
    state00://菜单
    begin
        led <= 8'b11111111;
    end

    state10://最近四次
    begin
        led <= 8'b00000000;
```

```
42      end
43
44      state11://最高分数
45      begin
46          led <= 8'b00000000;
47      end
48
49      state01://游戏
50      begin
51       randomnum = {randomnum[30:0],seed};
52       if(flashflag == fend)  //  闪烁周期
53       begin
54        cntnum = ~cntnum;
55        flashflag = 0;
56       end
57       if(flashflag != fend)
58       begin
59        flashflag = flashflag +1;
60       end
61       if(cntnum == 1)//闪烁
62         led = game;
63       if(cntnum == 0)//闪烁
64         led = 8'b00000000;
65       if(count == cycle  ||  answerflag == 1)//回答一次或者计数完毕，更换题目
66       begin
67        count = 0;
68        cntnum = 1;
69        game = {  randomnum[6],randomnum[29],randomnum[17],randomnum[20],randomnum[19],randomnum[21],randomnum[11],randomnum[18]  };
70       end
71       else
72       begin
73          count = count + 1;//count
74       end
75      end
76    endcase
77  end
78  endmodule
```

可以看到，上述代码第 13 行开始，给出了 seed 的取法，即对每一位进行按位异或，然后在第 51 行可以看到，seed 与原数的[30:0]进行拼接，这就是上文中提到的 LFSR 引出伪随机数的代码实现。

· answermodule

该模块是“锤子”信号接受模块。上文中已经提到了其实现方式，即用 sw 开关对应需要敲击的位置，按下 button 代表着敲击，根据这个思路，就可以编写出代码了。此外，在这模块中，我们还需要设置一个信号，该信号传递的信息是 button 是否被按下。当 button 被按下，令该信号有效，传输到 creatpuzzle 模块中，立刻产生下一个新的随机数。用这样的方法，就可以模拟地鼠不断随机出现的情形了。具体代码实现如下:

```
module answermodule(
input clk,
input signal,//做出回答
input [1:0] mod,
input [7:0] question,
input [7:0] answer,
input [1:0] sw,
output reg [3:0] score,
output reg answerflag
    );
reg [3:0] even;
initial
begin
    score = 0;
    answerflag = 0;
end

always@(posedge clk)
begin
  if(mod == 2'b01)//游戏模式
  begin
    if(signal == 1)
    begin
      answerflag = 1;//只要回答了就令flag为1，产生新的puzzle
      even = (answer[0] == question[0])+ (answer[1] == question[1])+ (answer[2] == question[2])+ (answer[3] == question[3])
           +(answer[4] == question[4])+ (answer[5] == question[5])+ (answer[6] == question[6])+ (answer[7] == question[7]);
      if(even == 4'b1000)//全部回答正确
        score = score + 1;
    end
    else
    begin
      even <= 0;
      answerflag <= 0;
    end
  end

  else//其他模式
  begin
    if( signal == 1)
      if(sw == 4'b01)//进行下一次游戏时分数清零
        score = 0;
  end
end
endmodule

```

・recordscore

该模块是得分记录模块，得分的规定为：成功在一个有效时间内击打一次所有出现的地鼠，就得一分，否则不得分。根据这个规则，就能很容易地写出该模块的代码了：

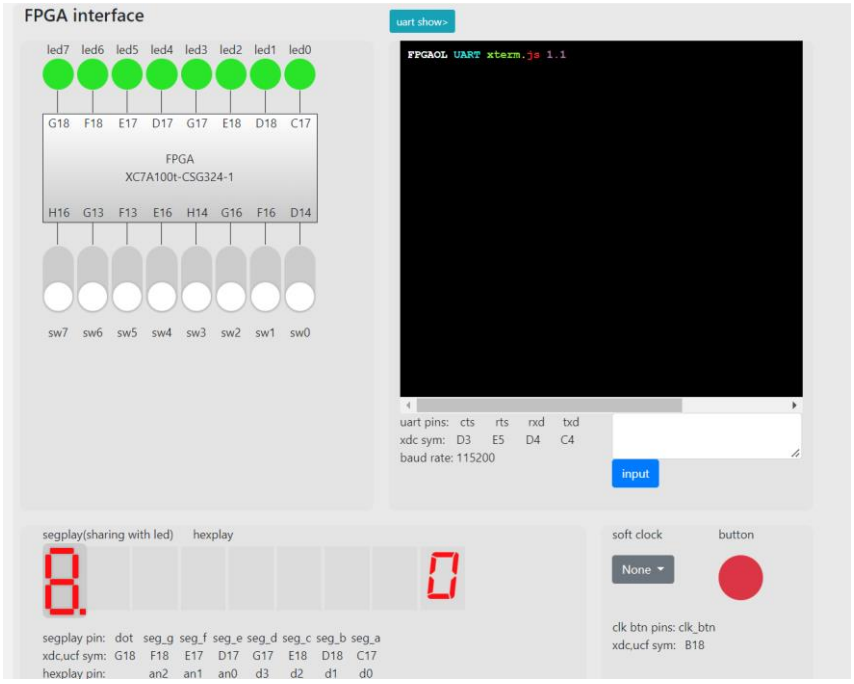
```
module recordscore(
input [3:0]score,
input clk,
input [1:0] mod,
input overflag,
output reg [15:0] nearrecord,//最近四次成绩
output reg [3:0] toprecord//最高分
    );

initial
begin
    nearrecord = 0;
    toprecord = 0;
end

always@(posedge clk)
begin
    if(overflag)
        if(mod == 4'b01)//游戏模式
        begin
            nearrecord <= { nearrecord[11:0],score };
            toprecord <= (score > toprecord)? score : toprecord;
        end
end
endmodule
```

· dcdtest

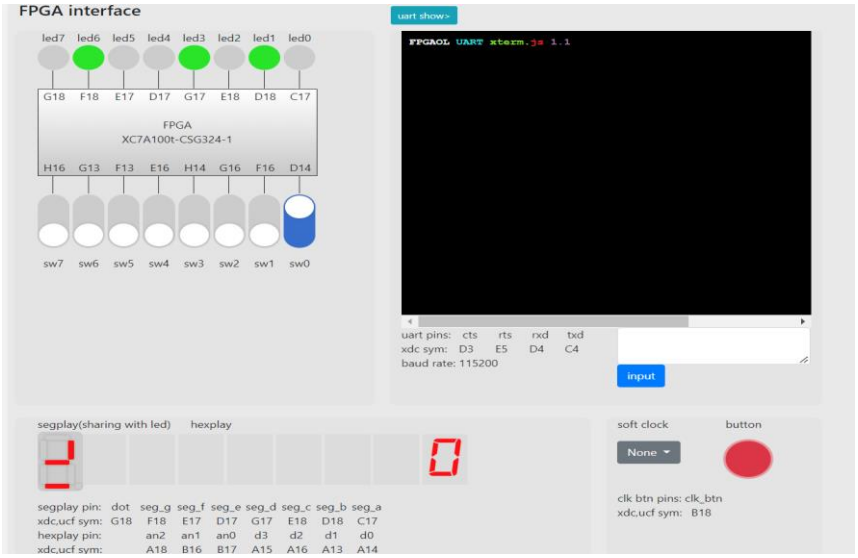
该模块是数码管显示模块，前面几次实验多次使用用时分复用的方法，就很容易得到该模块代码。由于代码很长，而原理其实很简单，故不做展示了。

五、效果展示

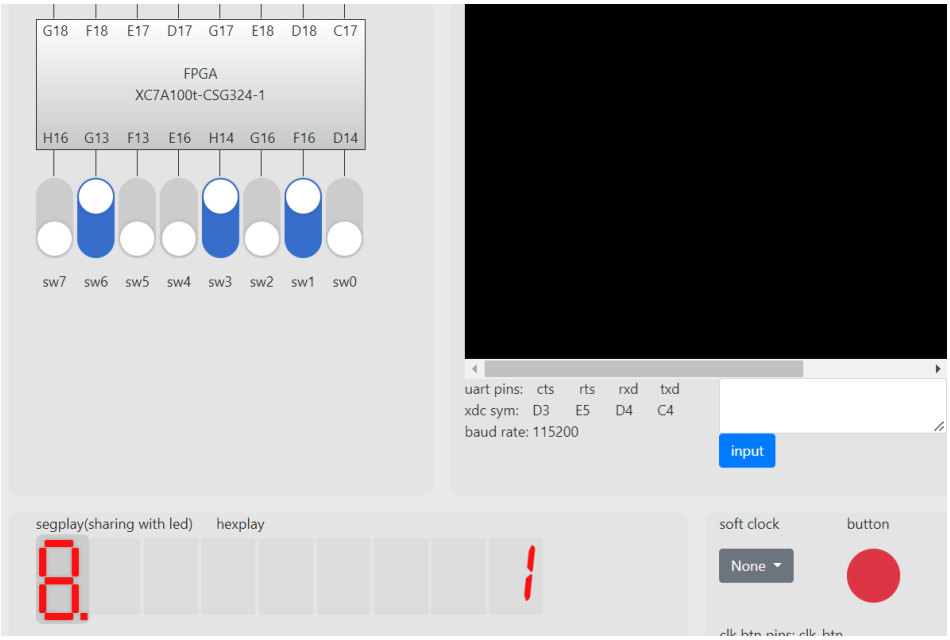
将 bit 流文件烧写到 FPGAOL 平台上，就可以开始游戏了。开始时，界面显示如下，所有 LED 灯亮起，这就是初始菜单界面：



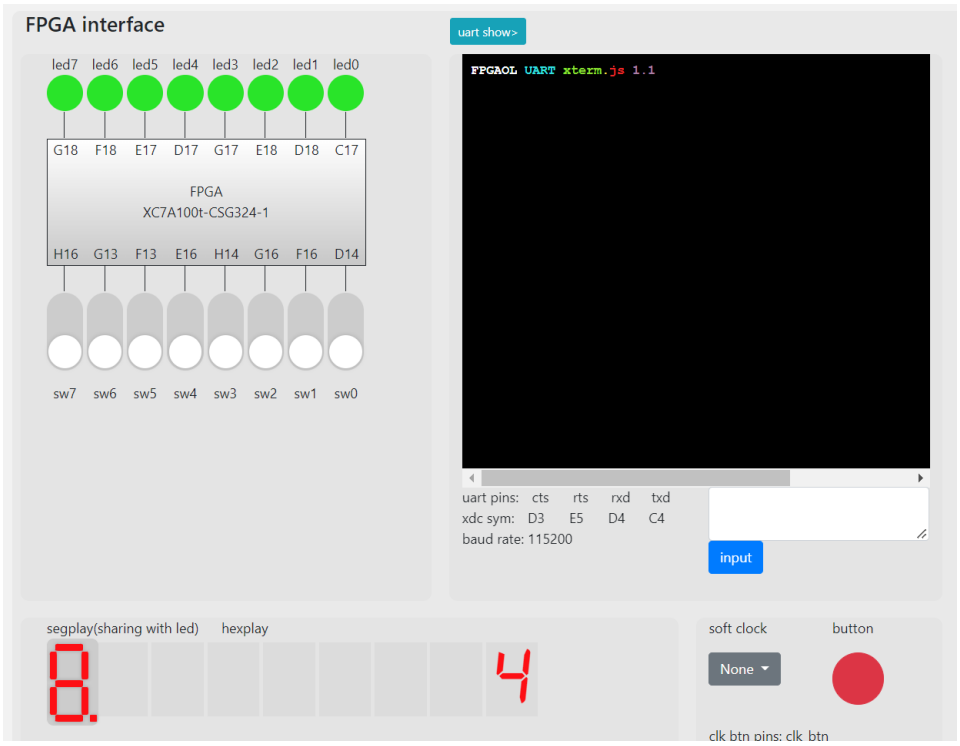
下面，选择 01 状态，按下 button，就可以进入游戏了：



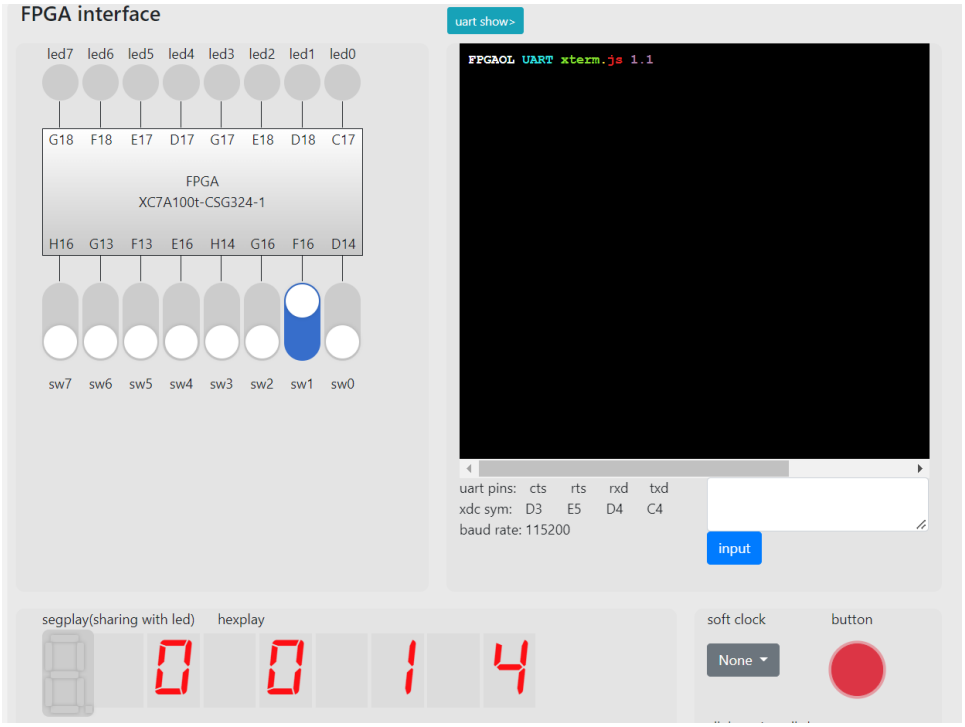
可以看到，led6/3/1 这三个灯在闪烁，这代表这三处有地鼠出现，于是，我们将对应的开关拨上去，按下 button，就获得了一分，分数在下方右边第一个数码管上显示。



30 秒后，游戏结束，本轮我们总共获得了 4 分：



下面选择状态 10，可以查看最近四次游戏的分数，测试时进行了两轮游戏，分别得分为 1、4：



接下来选择状态 11，按下 button，就可以查看历史最高分了：