哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：2020 秋季

课程名称：数字逻辑设计（实验）

实验名称：记忆游戏

实验性质：综合设计型

实验学时：6 地点：T2615

学生班级：7 班

学生学号：190110716

学生姓名：朱海峰

评阅教师：

报告成绩：

实验与创新实践教育中心制

2020 年 12 月

设计的功能描述

概述基本功能

生成五个五位八进制数，按时间顺序显示在数码管上。玩家记住第某个数后，在下一步输入地址（即序数）后，依次输入该五位八进制数，如果对应地址的数匹配，则提示匹配成功，否则继续匹配，知道最新输入的五个数匹配。

系统功能详细设计

用硬件框图描述系统主要功能及各模块之间的相互关系

包括状态描述、状态转换图及状态编码

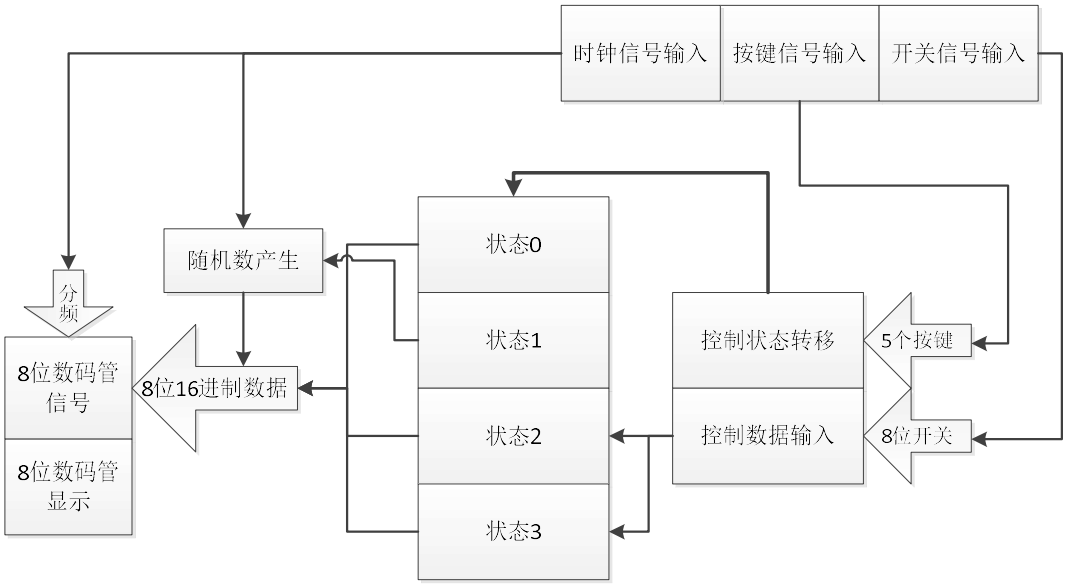

图 1

状态描述：

- 状态 1：初始状态，数码管显示 8 个 0；
- 状态 2：数码管按照 1Hz 频率显示五个五位八进制数；
- 状态 3：玩家输入选择的地址（即上一步某个五位数中的一个的序数）；
- 状态 4：玩家输入一个 3 位二进制数，按下 S3 确认，在输入下一个数。

状态间的转移通过按键信号控制。

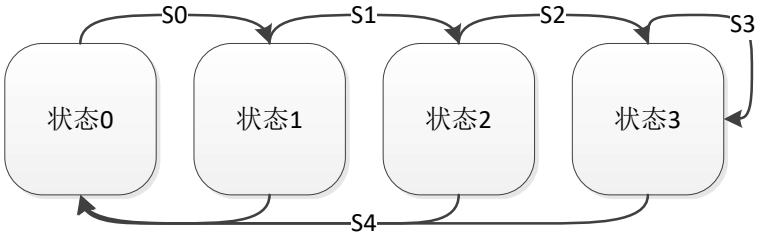

图 2 状态转换图

状态编码：

```
    localparam[2:0]
        state0 = 3'b000,
        state1 = 3'b001,
        state2 = 3'b010,
        state3 = 3'b011;
```

状态编码使用 3 位，为远期功能预留空间，当前实验并未用到最后一位。

模块描述

包括模块功能，输入、输出端口、变量含义及主要设计

topDesign 模块：

输入：5 个按键信号、时钟信号、8 位开关信号
输出：左右数码管信号、数码管使能信号。

1. 模块中通过按键信号控制状态转移：

现态	按键	次态
×	S4	
0	S0	1
1	S1	2
2	S2	3

```
if(S4)
    next_state <= state0;//初始
else if(cs==state0 && S0)
    next_state <= state1;//显示
else if(cs==state1 && S1)
    next_state <= state2;//输入地址
else if(cs==state2 && S2) begin
    next_state <= state3;//读数状态
    addrSel <= sw_i[2:0];
end
```

2. 不同状态下，对数码管输入不同状态对应寄存器的数据，其中 data1、data2、data3……data8 直接输入数码管模块：

```
if(cs==state0)begin //状态 0
    data1 = data1_0;
    data2 = data2_0;
    ...
    data8 = data8_0;
end
else if(cs==state1)begin //状态 1
    data1 = data1_1;
    ...
end
else if(cs==state2)begin //状态 2
    ...
```

3. 扫描更新伪随机数序列：

```
always@(posedge clk_i)begin
    i <=(i+1) % 'd25;
    if(i>='d0 && i<'d5)
        temp1[i%'d5] <= Y;
    else if(i>='d5 && i<'d10)
        temp2[i%'d5] <= Y;
    else if(i>='d10 && i<'d15)
        temp3[i%'d5] <= Y;
    else if(i>='d15 && i<'d20)
        temp4[i%'d5] <= Y;
    else if(i>='d20 && i<'d25)
        temp5[i%'d5] <= Y;
end
```

其中 Y 是当前时刻随机数模块生成的 4 位数（实际上只有 0~7），Y 每个时钟沿都在更新，temp1、temp2、temp3、temp4 和 temp5 为暂存的当前时刻随机数序列。

4. 状态为 0 时，按下 S0，当前随机数序列 temp1、temp2 等存入寄存器 randString1、randString2 等：

```
always@(posedge S0)begin
    if(cs ==state0)begin
            randString1[0] = temp1[0];
            ...
            randString2[0] = temp2[0];
            ...
            randString3[0] = temp3[0];
            ...
    end
end
```

5. 状态 1，显示 5 个 5 位八进制伪随机数：

```
    if(cs == state1)begin
        if(cnt == 'd0)begin
            data1_1 <= randString1[0];
            ...
        end
        else if(cnt == 'd1)begin
            data1_1 <= randString2[0];
            ...
        end
        else if(cnt == 'd2)begin
            ...
```

```
            end
            data6_1 <= 'he;//不显示
            data7_1 <= 'he;
            data8_1 <= cnt;
            if(cnt<'b100)
            cnt <= cnt + 1;//每秒钟计数增加
        end
        else
            cnt <= 0;
```

6. 状态 2，输入地址信号，如果不符合要求输出 8 个 f：

```
    if(cs==state2)begin //状态 2
            if(sw_i>'d4)begin
                data1 = 'hf;
                data2 = 'hf;
                ...
            else begin
                data1 = sw_i[3:0];
                data2 = ...
                end;
        end
```

7. 状态三，输入数据，数码管显示左移一位：

```
        if(cs==state3 && S3) begin
            data1_3 = data2_3;
            data2_3 = data3_3;
            data3_3 = data4_3;
            data4_3 = data5_3;
            data5_3 = sw_i[2:0];
        end
```

hexseg8 模块：

输入：8 个 4 位数、时钟信号
输出：左、右侧数码管信号、使能信号
功能：将八个四位数转化为 7 位的数码管信号，原时钟通过分频得到 1000Hz 扫描使能型号。
具体实现：实例化 8 个单个数码管信号转换模块，实例化一个分频模块。通过计数，利用 1000Hz 信号，输出扫描使能。

singleDisplay 模块：

输入：4 位数据
输出：7 位数码管信号

```
            case(data)
```

```
            4'b0000: led = 7'b1111110;  //0
            ...
            4'b1101: led = 7'b0000001;  //d 用来表示-
            4'b1110: led = 7'b0000000;  //e  用来表示不亮
            default: led = 7'b1000111;  //F
        endcase
```

由于本实验中不需要显示 9~E，故将 E 用来表示不亮，而 d 用来表示“-”（dash）。

shinningO 模块：

输入：系统时钟信号
输出：闪烁的 0。
功能：通过分频得到 4Hz 时钟信号，输出 4 位交替的 0 与 E。

divider1Hz 模块：

输入：系统时钟
输出：1Hz 时钟信号
功能：实现分频，以 1Hz 的模块为例。

```
if (rst == 1'b1)
    cnt <= 51'b0;
else
    cnt <= (cnt >= 'd100000000) ? 'h0 : cnt + 'h1;
clk_o <= rst_n_i & (cnt < 'd50000000);
```

randNum 模块：

输入：系统时钟
输出：4 位伪随机数（其中最高位为 0）
功能：基于 LFSR 产生一个随机数，由于本实验需要用到的 0~7，故输出模 8 即可。

```
begin
    rand_num[0] <= rand_num[7];
    rand_num[1] <= rand_num[0];
    rand_num[2] <= rand_num[1];
    rand_num[3] <= rand_num[2];
    rand_num[4] <= rand_num[3]^rand_num[7];
    rand_num[5] <= rand_num[4]^rand_num[7];
    rand_num[6] <= rand_num[5]^rand_num[7];
    rand_num[7] <= rand_num[6];
end
```

管脚分配表

```
#时钟信号
                    P17    clk_i
#输入信号
                    R1     sw_i[0]
                    N4     sw_i[1]
                    M4     sw_i[2]
                    R2     sw_i[3]
                    P2     sw_i[4]
                    P3     sw_i[5]
                    P4     sw_i[6]
                    P5     sw_i[7]
#数码管使能
                   G2     en_seg[0]
                   C2     en_seg[1]
                   C1     en_seg[2]
                   H1     en_seg[3]
                   G1     en_seg[4]
                   F1     en_seg[5]
                   E1     en_seg[6]
                   G6     en_seg[7]
#前四位数码管信号
                  B4     led_o_l[6]
                  A4     led_o_l[5]
                  A3     led_o_l[4]
                  B1     led_o_l[3]
                  A1     led_o_l[2]
                  B3     led_o_l[1]
                  B2     led_o_l[0]
#后四位数码管信号
                  D4     led_o_r[6]
                  E3     led_o_r[5]
                  D3     led_o_r[4]
                  F4     led_o_r[3]
                  F3     led_o_r[2]
                  E2     led_o_r[1]
                  D2     led_o_r[0]
#按键
                    U4     S4
                    V1     S3
                    R15    S2
                    R17    S1
                    R11    S0
```

调试报告

包括仿真代码、仿真波形截图及仿真分析（需包含状态转换过程）

由于仿真过程较慢，要仿出现实中的以秒为单位的变化很费时间。于是在仿真的时候，将每个时钟分频器输出的频率增大。

对系统的伪随机数生成功能进行仿真，要求在波形中观察生成随机数时的关键信号和所生成的伪随机数

1. 单个数码管显示仿真，其中 E 用于表示不亮，D 用于表示 dash 连接线。通过对数据递增，完成 0、1、…、D、E 每个数的数码管显示仿真。
   仿真代码：

```
reg [3:0]data_i = 0;
wire [6:0]led_o;
singleDisplay uut_singleDisplay_1(
.data (data_i),
.led (led_o)
);
always #1 data_i <= data_i +1;
```

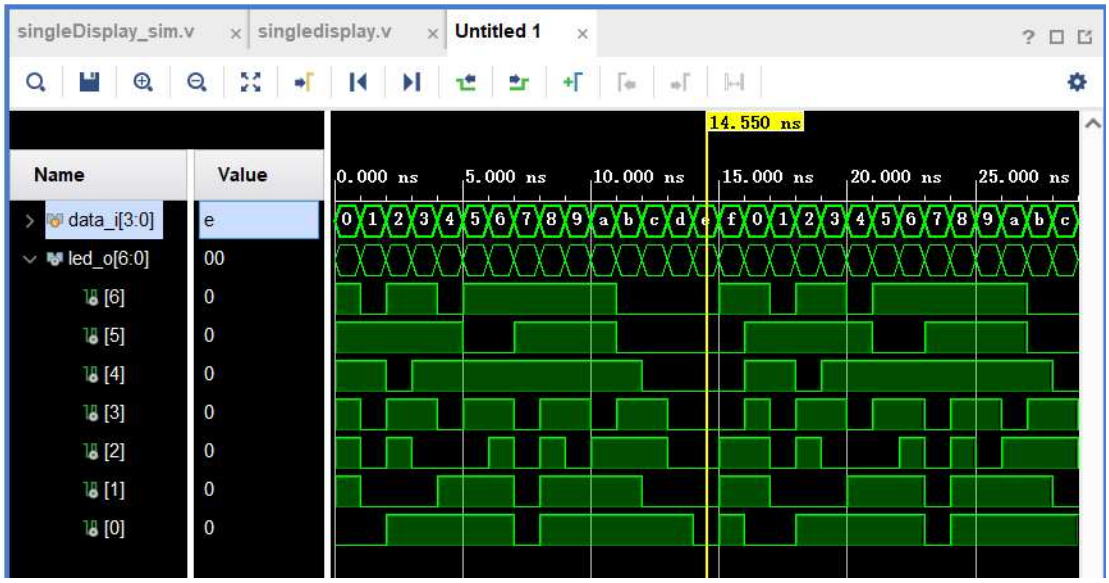

图 3

2. 八位数码管显示模块仿真

将需要显示的数据接入数码管模块，查看输出型号，主要是观察能信号。使能信号呈现出四个一组循环接替有效，说明仿真成功。

仿真代码：

```
reg clk=0;
reg [3:0]
   data1, data2, data3, data4, data5, data6, data7, data8;
reg [7:0] sw_i = 'b00000000;
wire [6:0] led_o_l;
wire [6:0] led_o_r;
wire [7:0] en;
wire dot;
hexseg8 uut_hexseg3_1(
.clk_i  (clk),
.data1  (data1),
.data2  (data2),
.data3  (data3),
.data4  (data4),
.data5  (data5),
.data6  (data6),
.data7  (data7),
.data8  (data8),
.led_o_l  (led_o_l),
.led_o_r  (led_o_r),
.en    (en)
);
always #1 clk=~clk;
always #10000000 sw_i = sw_i + 1;
initial begin
data1<='b1;
data2<='d2;
data3<='d3;
data4<= 'd4;
data5<= 'd5;
data6 <= 'd6;
data7 <= 'd7;
data8 <= 'd8;
end
```

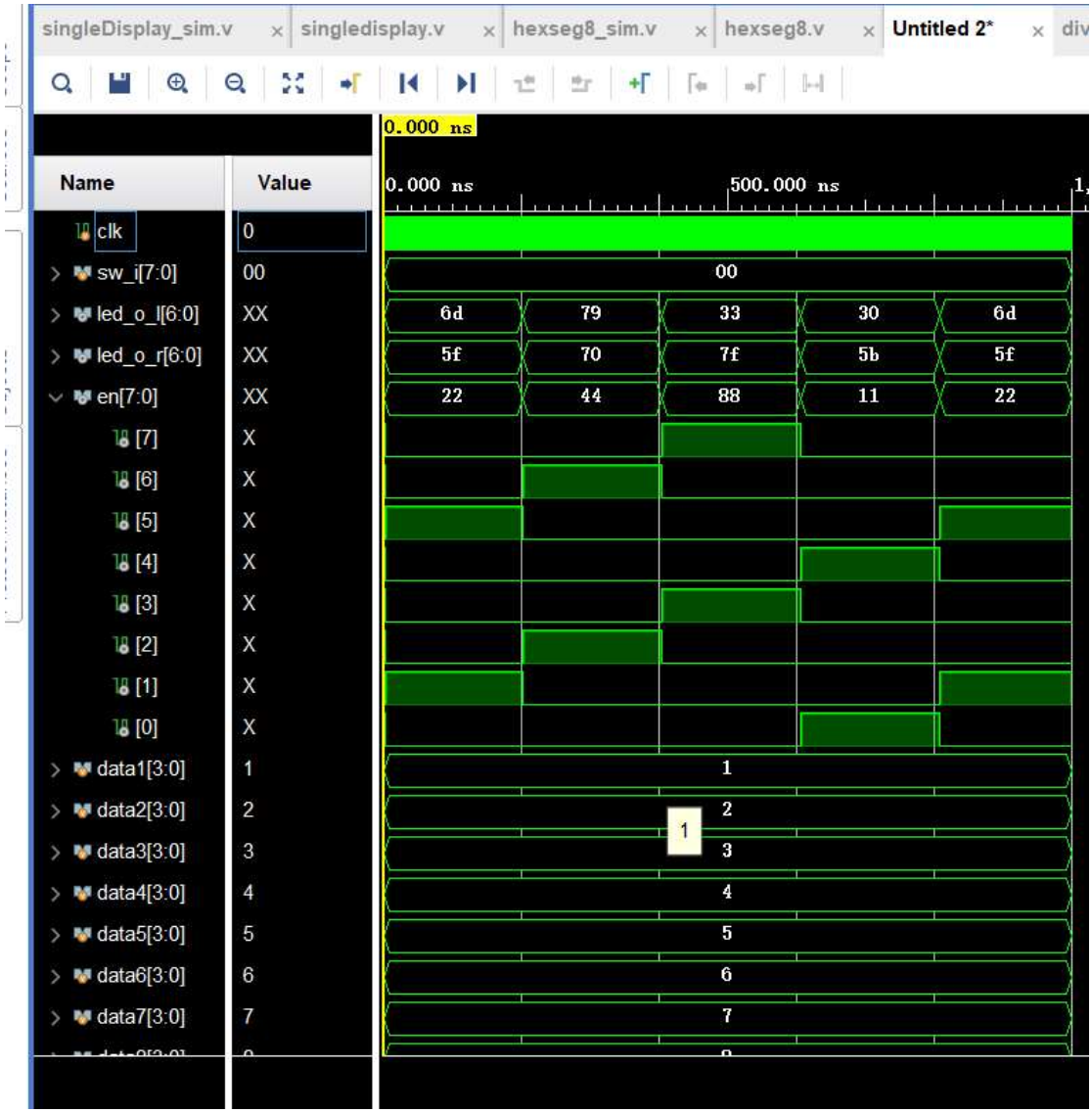

图 4

3. 伪随机数生成

伪随机数模块在每个时钟沿实时更新一个新 0~7 的随机数：out。

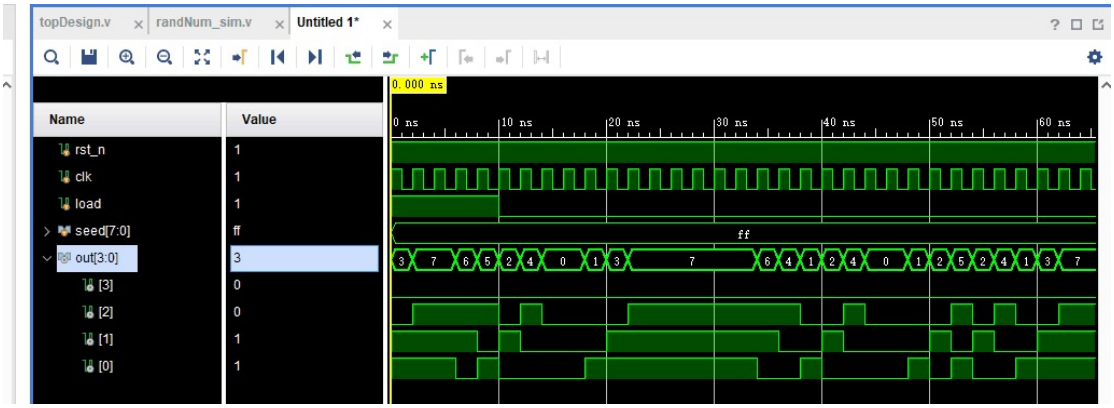

图 5

4. 状态转移控制

每次延时后给一个按钮按下有效信号，仿真状态表的逻辑。以下仿真代码

得到信息较多，之后的波形分析都是以下仿真代码得出：

```
reg clk_i = 0;
reg S1 = 0,
   S2 = 0,
   S3 = 0,
   S4 = 0,
   S0 = 0;
reg [7:0]sw_i;
wire [6:0]led_o_l,led_o_r;
wire [7:0] en_seg;
topDesign U_topDesign_sim1(
.clk_i  (clk_i),
.S4    (S4),
.S3    (S3),
.S2    (S2),
.S1    (S1),
.S0    (S0),
.sw_i   (sw_i),
.led_o_l(led_o_l),
.led_o_r(led_o_r),
.en_seg (en_seg)
);
always #1 clk_i = ~clk_i;
initial begin
#100 S0 = 1;
#10 S0 = 0;
#1000 S1 = 1;
#10 S1 = 0;
sw_i[7:0] = 8'b00000011;
#100 S2 = 1;
#10 S2 = 0;
#10 sw_i[7:0] = 8'b00000101;
#1 S3 = 1;#1 S3=0;
#10 sw_i[7:0] = 8'b00000010;
#1 S3 = 1;#1 S3=0;
#10 sw_i[7:0] = 8'b00000101;
#1 S3 = 1;#1 S3=0;
#1 sw_i[7:0] = 8'b00000011;
#1 S3 = 1;#1 S3=0;
#1 sw_i[7:0] = 8'b00000111;
#1 S3 = 1;#1 S3=0;
#1 sw_i[7:0] = 8'b00000111;
#1 S3 = 1;#1 S3=0;
end
```

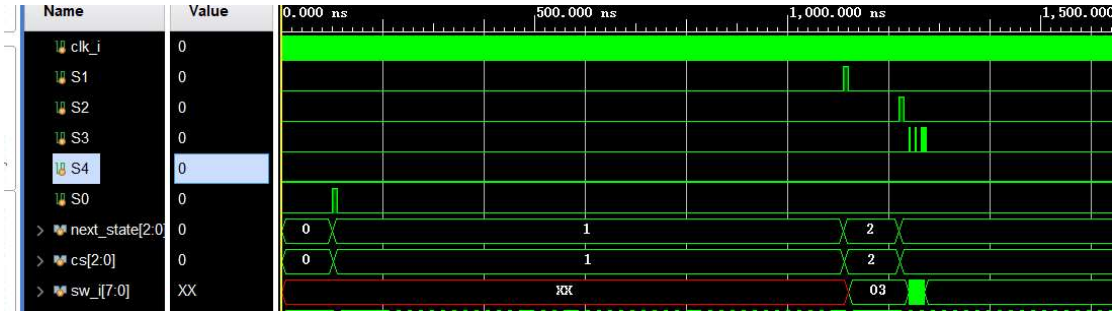

图 6

5. 通过计数 i 控制的实时更新伪随机数 temp

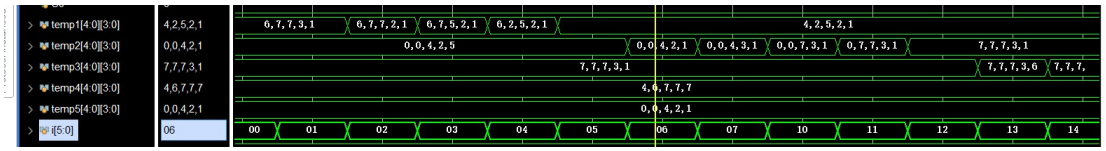

图 7

6. randString 为 S0 上升沿存入寄存器的随机数
S0 按下时，五组五个 0~7 的随机数存入寄存器里面。

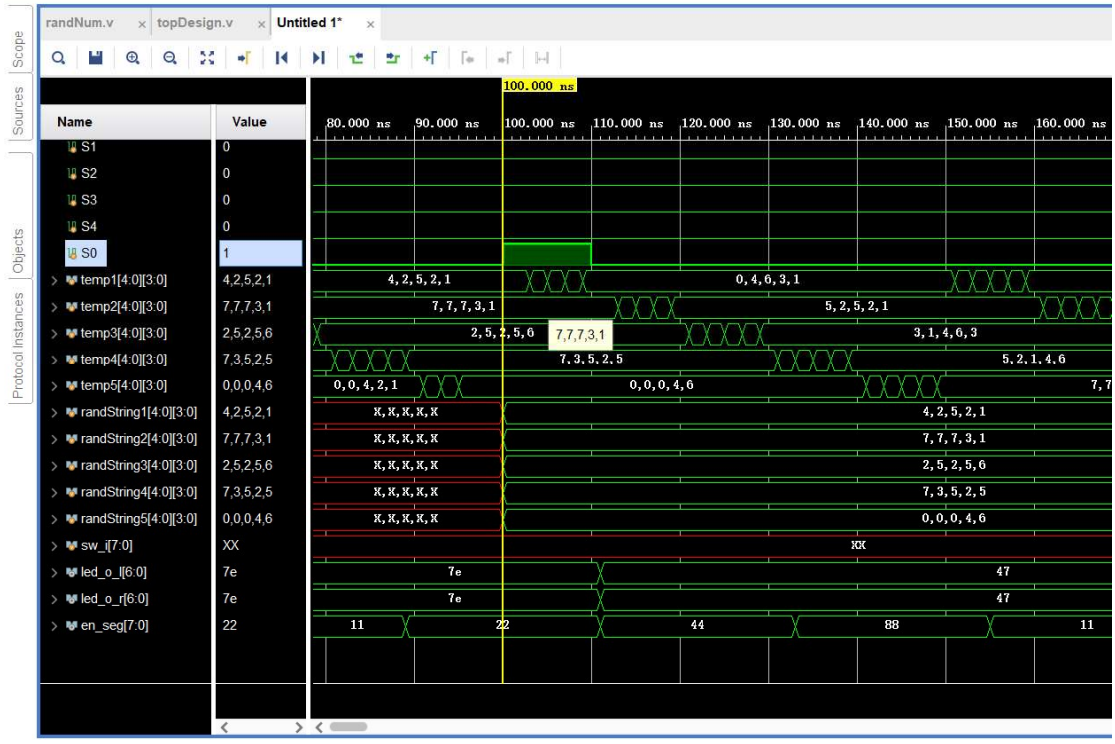

图 8

7. 成功匹配，data8 输出从闪烁的 0 变为一个 a
在输入五个正确的数后，数码管显示指示正确输入。

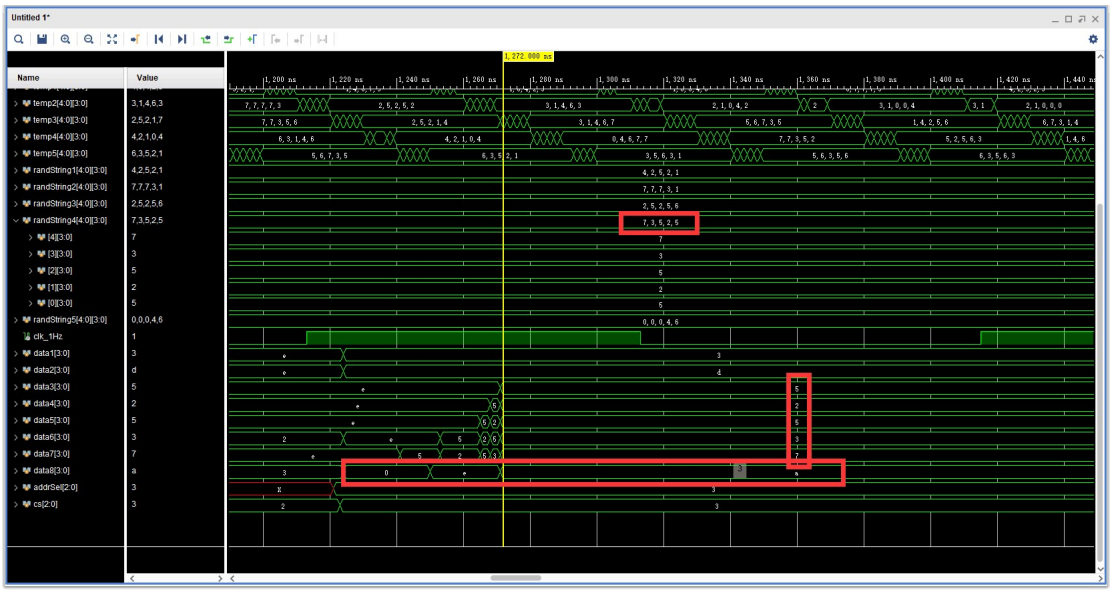

图 9

8. 成功匹配后又输入一个数，变成失败匹配状态，data8 回到 0 与 E 交替，即闪烁的 0 的状态。

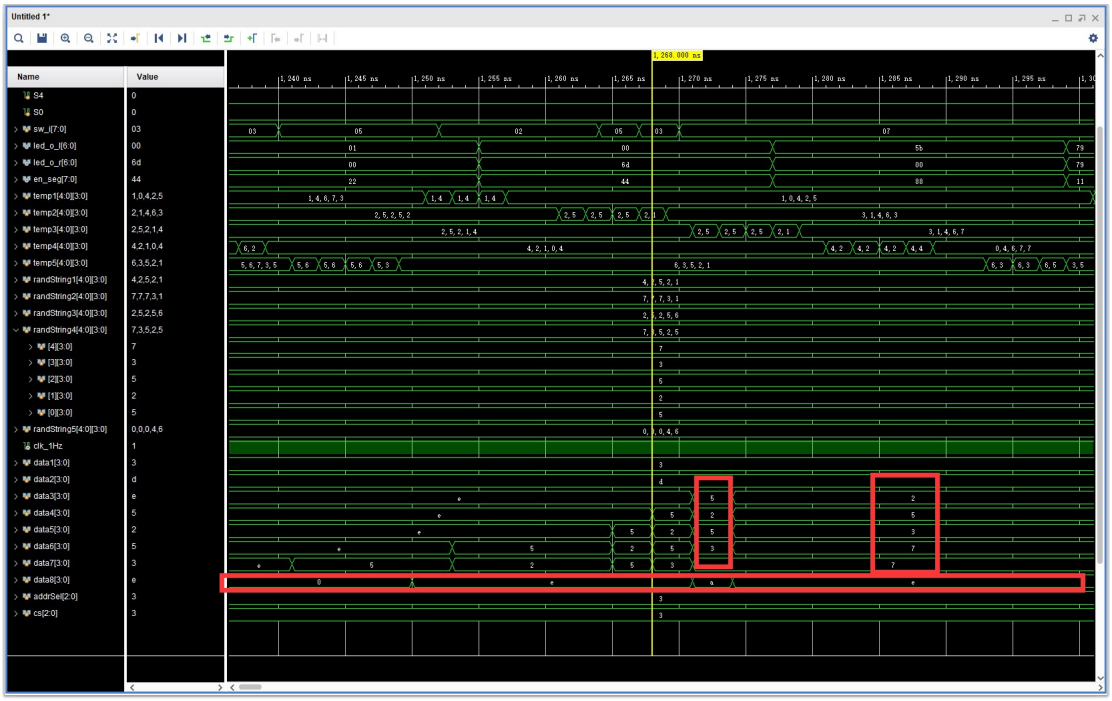

图 10

设计过程中遇到的问题及解决方法

仿真中遇到的问题（需包括仿真截图）

在闪烁的 0 模块制作中，发现 0 并没有闪烁，在仿真中 E 并没有出现，发现分频器计数位数不足，输出时钟并没有出现高低电平交替。

将计数位数增加后，得到成功分频后的 4Hz 时钟输出：

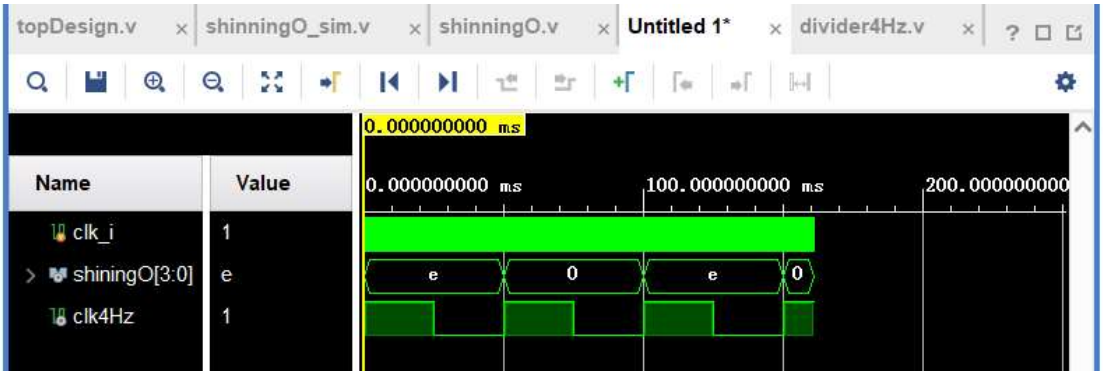

图 11

输入正确序列后，指示灯仍然闪烁，不提示正确。

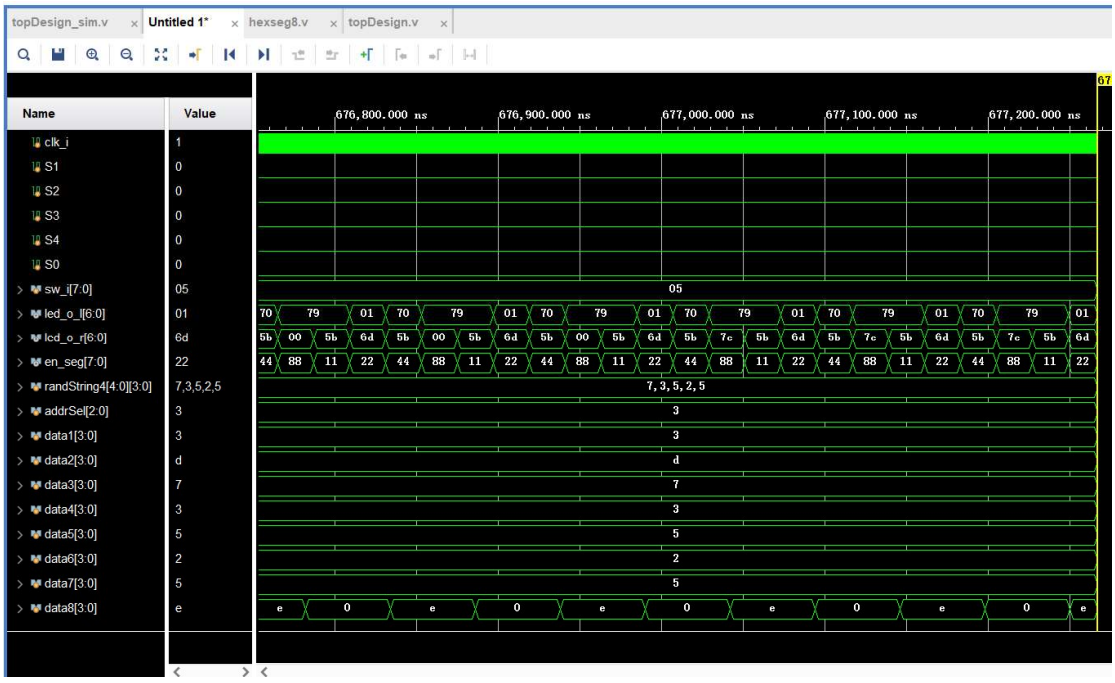

图 12

通过单步调试，缩小问题范围，发现是序列顺序搞反了，Verilog 默认的顺序是从高位到地位，与输入的顺序相反。

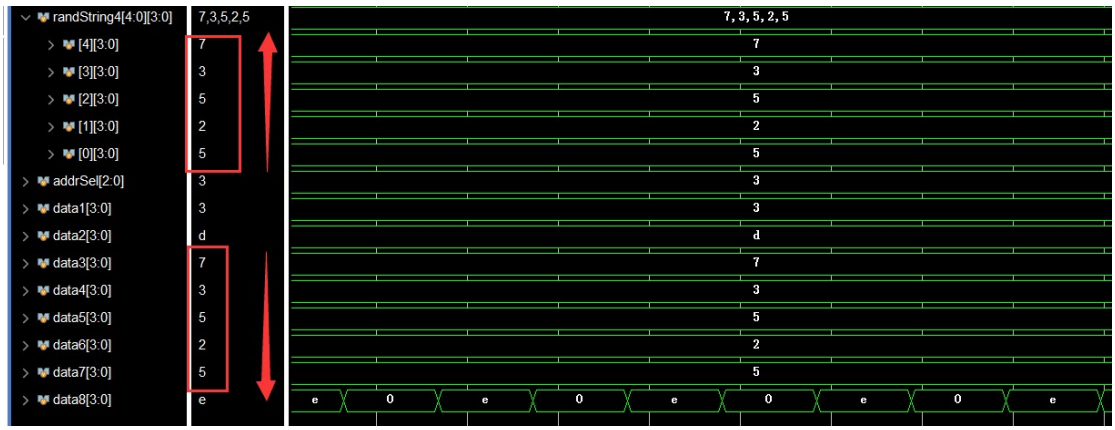
randString4[4:0][3:0]
[4][3:0]
[3][3:0]
[2][3:0]
[1][3:0]
[0][3:0]
addrSel[2:0]
data1[3:0]
data2[3:0]
data3[3:0]
data4[3:0]
data5[3:0]
data6[3:0]
data7[3:0]
data8[3:0]
7,3,5,2,5
7
3
5
2
5
3
3
d
7
3
5
2
5
e
7, 3, 5, 2, 5
e
0

图 13

课程设计总结

包括设计的总结和还需改进的内容以及收获

整个数字逻辑实验，我遇到了很多困难，和许多具体的问题。问题没有一一解决，但是每个实验都成功按时做出来了，很多东西是书本上得不到的。比如，某个变量在多个 always 中赋值，仿真没问题，上板全是问题，这个是我以前不知道的。以前不知道，就只能一组敏感信号写一个 always 块，但是对某个变量的赋值却非常混乱；再知道这个特性之后，编程思路一下子就清晰了：每个（组）变量的变化用一个 always 块，就不会出错。

每个实验的难度逐步上升，梯度十分合适，到了实验六，一个综合性的实验，怎么说呢，做出来了之后感觉也不算难。每次的困难都在于不清楚自己哪里的逻辑出问题了，或者是自己的逻辑与 verilog 的逻辑对不上，实现的方式不对。Verilog 语言没有专门去学，老实说，基础很差，如果需要进一步用的话，最好还是从头学。

写 Verilog，相信大多数同学都是从 C 语言这种面向过程的思维来写的，但是硬件远远不止一条线拉下来地这样运行。

实验逻辑简单，实现也不算难，调试是最繁琐的。从调试仿真到调试上板，方法区别也非常大。调试仿真，除了看波形，还能利用单步运行、断点调试，这个是我在运用过程中自己摸索出来的，仿真的调试跟 C 语言的调试很像，通过观测各个变量的变化，缩小问题所在的可能范围，然后改。上板的话，就复杂很多，可以看 message 里面的 warnings，特别是 critical warning，能解决一些问题。其实更多情况是这些 warnings 根本解决不了问题，只有凭着自己的直觉，去改一些 always 块的敏感信号，可能会有一些变化。

讲真，耐心最重要，老是调不出来，还得敢于放弃，看看别人怎么写的，问问老师，改一改不规范的代码。毕竟重写，是走入绝境后还想完成实验最大的可能。