浙江工业大学

计算机科学与技术学院、软件学院

数字电路与数字逻辑

课堂设计报告

学生姓名及学号	
专业班级	
提交日期	2025.6.29

目录

引言

随着现代电子系统向高性能、高集成度和低功耗方向不断发展，EDA（电子设计自动化）技术作为数字电路设计的重要支撑工具，已经成为电子工程领域不可或缺的一部分。而 Verilog 硬件描述语言作为 EDA 设计流程中的核心语言之一，因其语法简洁、功能强大、可读性强等特点，在 FPGA 与 ASIC 开发中得到了广泛应用。通过 Verilog，我们可以像写程序一样对电路行为进行建模与仿真，实现从“思维逻辑”到“电路逻辑”的桥梁搭建。

本次实验紧密结合 Verilog 语言的核心功能，围绕 GPIO 接口、矩阵按键识别、LED 控制与蜂鸣器反馈等内容，设计并完成了多个功能模块，包括 LED 流水灯、简易加法器及“打地鼠”游戏系统。这些模块的实现过程不仅锻炼了我们对时序逻辑、状态机设计、模块化编程的理解，也帮助我们掌握了 EDA 开发环境（如仿真、综合、引脚配置、板上验证等）的基本操作流程。

此外，Verilog 及 EDA 工具在未来数字技术发展中的前景十分广阔。随着 AI 芯片、物联网设备、可穿戴系统等新兴技术的兴起，越来越多的应用场景对定制化逻辑与高效硬件设计提出了更高要求。而 FPGA 作为灵活可重构的硬件平台，其基于 Verilog 的设计方法，将在边缘计算、工业控制、通信系统等领域扮演更加关键的角色。

通过本次实验，我们不仅巩固了课堂上所学的数字逻辑基础，也初步掌握了 Verilog 在 EDA 开发中的实际应用方法，为今后深入学习数字系统设计、嵌入式开发乃至芯片设计奠定了扎实的技术基础。

一 实验内容

1. GPIO 接口实验-LED 流水灯设计

流水灯设计要求：主板上的 LED 灯 D1-D8 依次点亮然后熄灭，循环反复。

下载到实验电路板上，学会如何操作实验电路板进行验证。

2. 矩阵按键实验-简易加法器设计

加法器设计要求：在按键 J1-J9 中，先后按下两个按键，实现两个 10 进制数相加，加法结果用 D4-D8 显示。

3. 砸地鼠游戏设计（或多功能序列发生器设计）

要求设计两种序列的砸地鼠：

① 顺序地鼠

要求：D1~D8 依次点亮，每个点亮时间持续 1 秒，代表地鼠顺序出现并停留 1 秒；在这 1 秒内按下相应的按键（J1~J8），若成功则蜂鸣器响 0.1 秒，表示砸地鼠成功。

② 随机地鼠

要求：D1~D8 随机点亮，每个点亮时间持续 1 秒，代表地鼠随机出现并停留 1 秒；在这一秒内按下相应的按键（J1~J8），若成功则蜂鸣器响 0.1 秒，表示砸地鼠成功。

二　组员分工

组长：

矩阵按键加法器代码实现、大部分实验调试、报告汇总撰写、优化、排版。

组员1：

流水灯代码实现、代码资料收集整理、实验报告 GPIO 接口实验-LED 流水灯设计撰写

组员2：

流水灯代码微调、按键矩阵加法器代码撰写，实验报告矩阵按键实验-简易加法器设计撰写。

组员3：

随机地鼠代码实现调试，实验报告砸地鼠游戏设计撰写，实验过程的照片视频拍摄。

三　GPIO 接口实验-LED 流水灯设计

1 设计要求

主板上的 LED 灯 D1-D8 依次点亮，然后熄灭，循环反复。

2 设计方案

2.1计数及时间控制

我们使用一个 32 位的计数器 time_cnt 来实现时间管理功能。该计数器在每个时钟上升沿时自增 1；当其数值达到 32'd90000_0000（即 9 秒）时，会被清零，重新开始下一轮计时。

代码展示：

```
reg [31:0] time_cnt =0;
always@(posedge clk)
if(time_cnt == 32'd90000_0000) time_cnt <= 32'd0;
else time_cnt <= time_cnt + 1'b1;
```

2.2LED 灯的状态判断与更新

我们采用一个 8 位寄存器 led_reg 来表示当前所有 LED 灯的状态。通过 if…else if… 条件语句，根据计数器的当前值对 led_reg 进行更新，使 LED 从 D1 到 D8 依次亮起，每种状态持续 1 秒。最后，所有 LED 熄灭 1 秒，表示本次计时周期已结束，准备进入下一轮循环。

代码展示：

```
reg [7:0] led_reg;
```

```
always@(posedge clk)
if(time_cnt <= 32'd10000_0000) led_reg<=8'b0000_0001;
else if(time_cnt <= 32'd20000_0000) led_reg<=8'b0000_0011;
else if(time_cnt <= 32'd30000_0000) led_reg<=8'b0000_0111;
else if(time_cnt <= 32'd40000_0000) led_reg<=8'b0000_1111;
else if(time_cnt <= 32'd50000_0000) led_reg<=8'b0001_1111;
else if(time_cnt <= 32'd60000_0000) led_reg<=8'b0011_1111;
else if(time_cnt <= 32'd70000_0000) led_reg<=8'b0111_1111;
else if(time_cnt <= 32'd80000_0000) led_reg<=8'b1111_1111;
else led_reg<=8'b0000_0000;
```

2.3LED 灯的控制

通过 assign 语句将 led_reg 的值直接赋给输出信号 led，这样一旦 led_reg 发生变化，led 就会立即更新，从而实现对实际 LED 灯状态的控制。流程图如图 1。

代码展示：

```
assign led = led_reg;
```

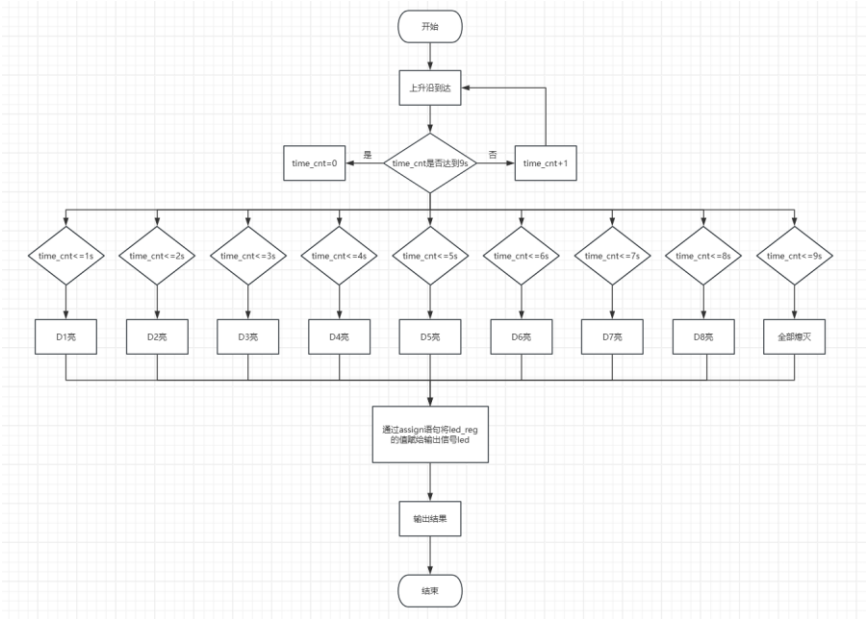

图 1LED 流水灯流程图

3 仿真结果

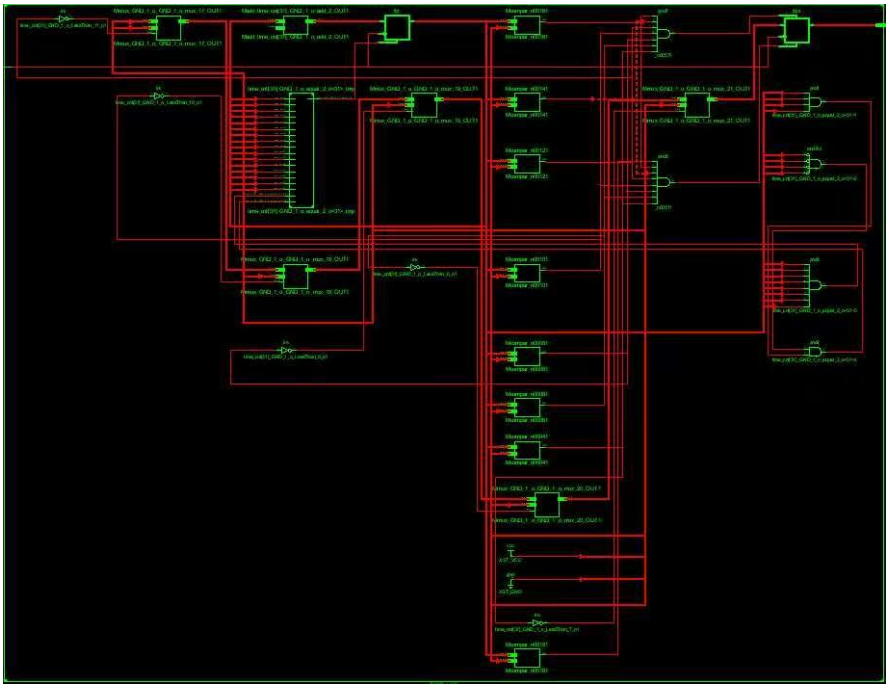

图 2 流水灯电路图

在程序编译完成后，进入仿真界面，将时钟信号 clk 和输出信号 led 添加到仿真文件中。随后可以看到对应的电路图（见图 2）、输入输出信号（见图 3）以及仿真波形结果（见图 4）。

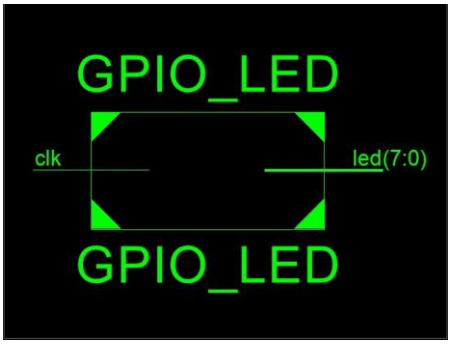

图 3 流水灯输入输出信号

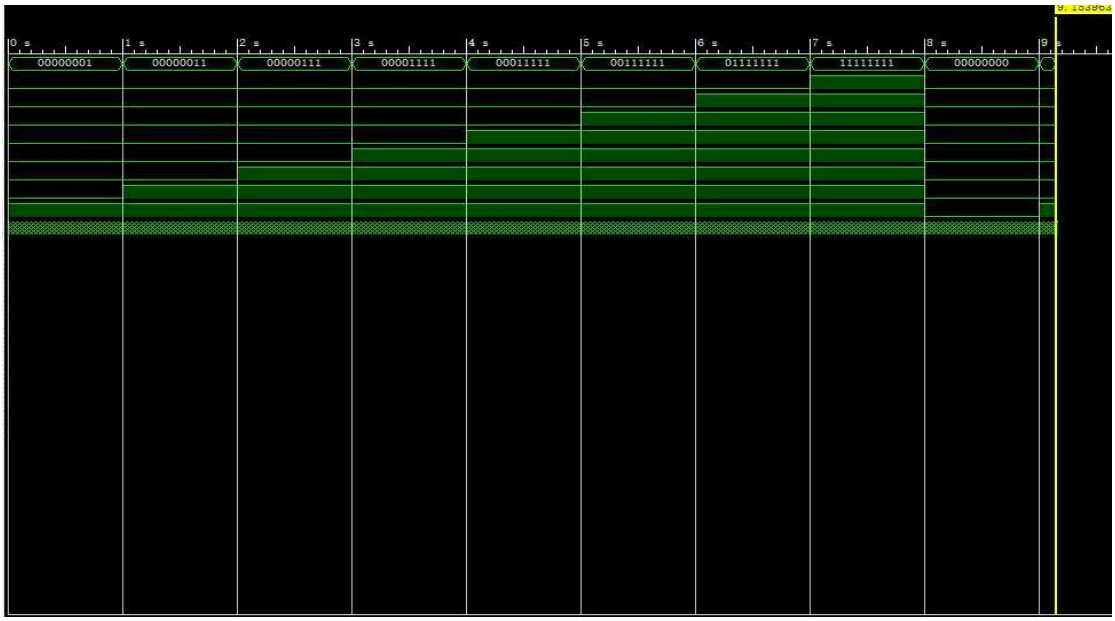

图 4 流水灯仿真结果

clk 是输入信号，led 是输出信号，而 led 的输出结果完全由 clk 的输入决定。

4 电路板验证

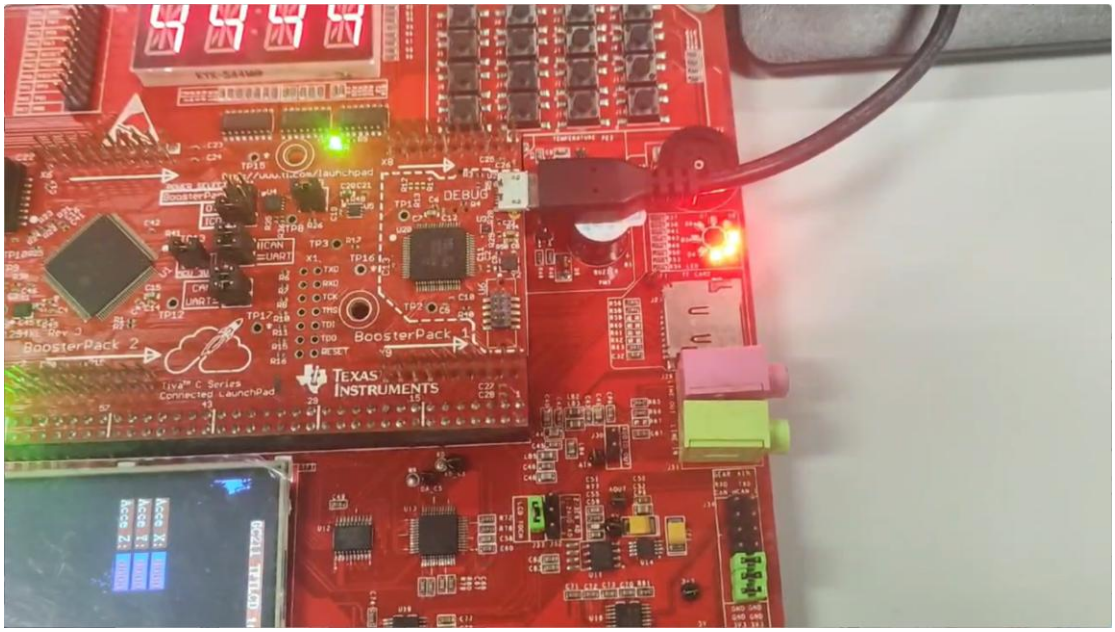

图 5 流水灯的电路板验证(1)

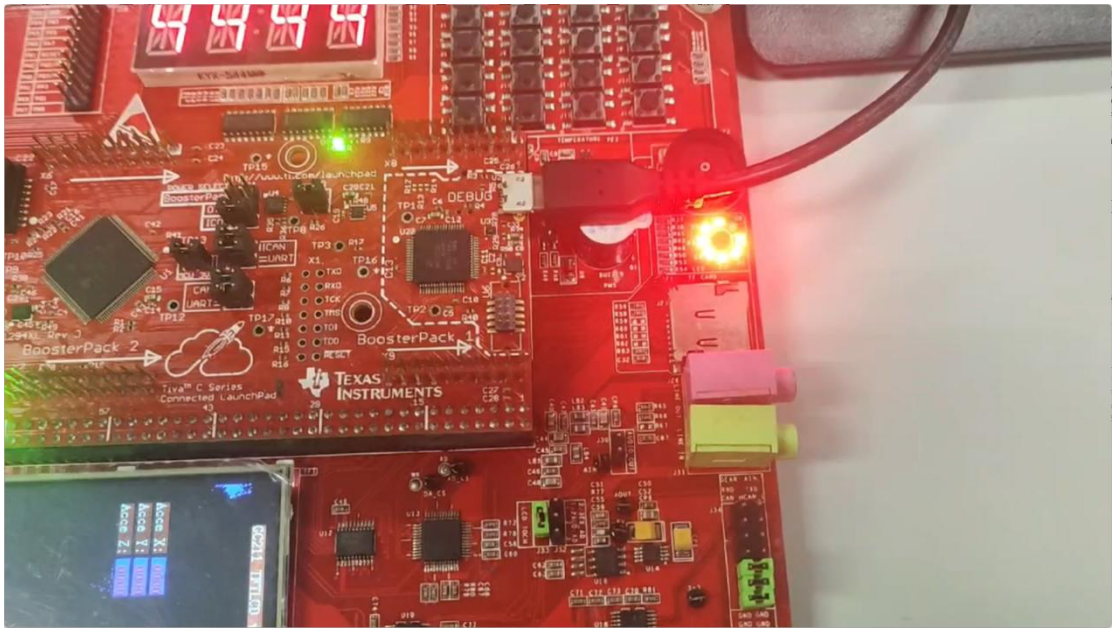

图 6 流水灯的电路板验证(2)

四　矩阵按键实验-简易加法器设计

1 设计要求

在按键 J1-J9 中，先后按下两个按键，实现两个 10 进制数相加，加法结果用 D4-D8 显示。

2 设计方案

在简易加法器的设计中，矩阵按键的输入被引入，以实现对用户输入的有效检测和处理。

1. 矩阵按键

矩阵键盘也叫行列式键盘，采用 4 根 I/O 口线作为行线，另外 4 根作为列线，每个按键都安装在行线和列线的交叉位置。如图 7。

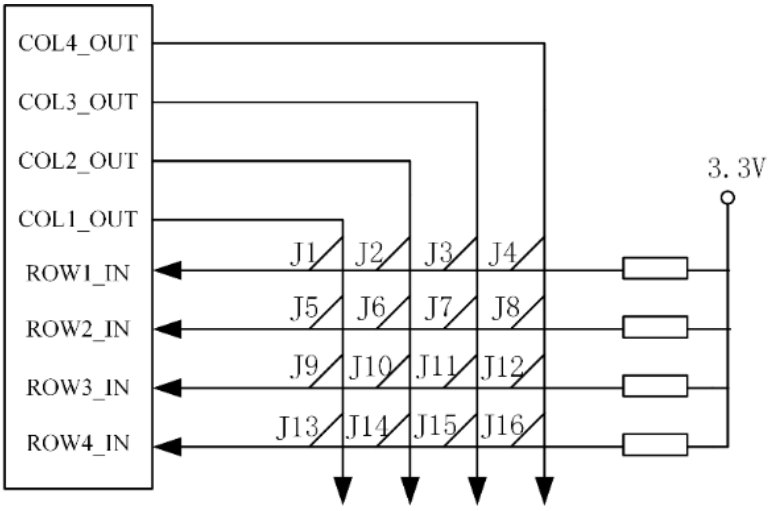

图 7 按键原理图

扫描时，每次只让其中一列输出低电平，其他列保持高电平。如果该列有按键按下，对应的行线电平会从高变低，从而检测到按键动作。具体过程是：先将第 0 列设为低电平，然后检查各行线是否有电平变化。如果有行线变低，说明该行与第 0 列交叉点的按键被按下；如果没有检测到变化，就切换到下一列继续扫描，依次循环检测所有按键。

系统使用一个四状态有限状态机进行扫描，四个状态分别是 CHECK_R1、CHECK_R2、CHECK_R3 和 CHECK_R4，每个状态对应一列的扫描，每列检测时间为 1ms。在扫描过程中，同时检测行线状态以获取按键值，并将有效的按键值存储到按键变量中。如图 8。

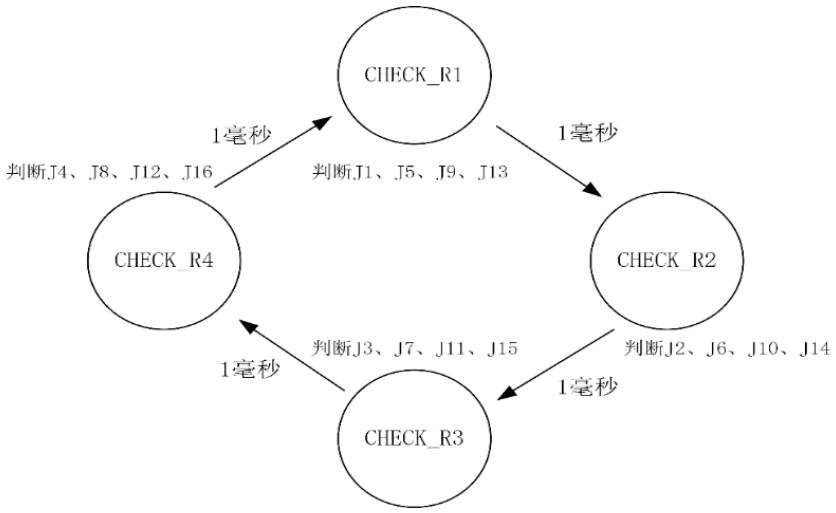

图 8 按键判断流程图

对应代码片段:

```
// 矩阵按键列状态定义
localparam CHECK_R1 = 3'b000;
localparam CHECK_R2 = 3'b001;
localparam CHECK_R3 = 3'b011;
localparam CHECK_R4 = 3'b010;
// 状态组合判断，不断地转换到下一组按键
reg [2:0] state = 3'b000;
always @(posedge clk) begin
    case(state)
        CHECK_R1:
            if(cnt_full) state <= CHECK_R2;
            else state <= CHECK_R1;
        CHECK_R2:
            if(cnt_full) state <= CHECK_R3;
            else state <= CHECK_R2;
        CHECK_R3:
            if(cnt_full) state <= CHECK_R4;
            else state <= CHECK_R3;
        CHECK_R4:
            if(cnt_full) state <= CHECK_R1;
            else state <= CHECK_R4;
        default: state <= state;
    endcase
end
// 状态机输出逻辑
reg [4:0] key_out = 5'd0;
always @(posedge clk) begin
```

```
    case(state)
        CHECK_R1: begin
            col <= 4'b1110;
            case(row)
                4'b1110: key_out <= 5'd1; // J1
                4'b1101: key_out <= 5'd5; // J5
                4'b1011: key_out <= 5'd9; // J9
                4'b0111: key_out <= 5'd13; // J13
                4'b1111: key_out <= 5'd17; // 无按键按下
            endcase
        end
        // 其他列检测逻辑同理
    endcase
end
```

2. 按键消抖处理

按键在按下时可能会有微小的抖动，导致检测到的波形会有一定的信号毛刺，为了降低该毛刺对结果的影响，引入滤波器进行按键消抖，判断按键值是否在一段时间（20us）内保持不变，并且有按键按下，才输出消抖后的最终的按键值。

对应代码片段:

```
// 按键消抖处理
reg [4:0] key_out_buf = 5'd0;
always @(posedge clk) key_out_buf <= key_out;
reg [19:0] cnt_900us;
always @(posedge clk) begin
    if (cnt_900us == 20'd600_00) cnt_900us <= 20'd0;
    else if (key_out_buf != key_out) cnt_900us <= 20'd0;
    else cnt_900us <= cnt_900us + 1'b1;
end
reg [4:0] key_out_fliter;
always @(posedge clk) begin
    if (cnt_900us >= 20'd200_00 && key_out_buf != 5'd17)
        key_out_fliter <= key_out_buf;
    else
        key_out_fliter <= key_out_fliter;
end
```

3. 加法器

系统设计使用 key_flag（0 或 1）来标记两次读数是否一致，并定义了三个功能键：J13（相同读数确认）、J14（清零）和 J15（蜂鸣器控制）。此外，还设置了三个寄存器：q1 和 q2 存储两个加数，qans 存储加法运算结果，以实现 1-12 范围内的十进制加法。

具体操作流程如下：首先按下第一个数字，判断其是否在 1-12 范围内，如果是则存入 q1，并将 key_flag 置 1 表示已输入第一个加数。如果第二个加数与第一个相同，需要先按 J13 将 xiangtong 标志置 1；如果不同则直接输入，符合范围要求后存入 q2。接着，系统计算 q1+q2 并将结果存入 qans，同时将 qans 的值转换为二进制信号输出到 D4-D8 LED 灯上显示。最后，按下 J14 可清零所有数据，以便进行下一次加法运算。

对应代码片段:

```
reg [4:0] key_flag = 5'd0; //用0 和1 标志两次读数
reg [4:0] q1=5'd0;//存储第一个加数
reg [4:0] q2=5'd0;//存储第二个加数
reg[4:0] qans=5'd0;//存和
reg [4:0] xiangtong=5'd0;
always@(posedge clk) begin
if(key_out_fliter>=5'd1&&key_out_fliter<=5'd12)//1~12 范围内的加法
begin
    if(q1==5'd0&&key_flag==5'd0)
    begin
        q1=key_out_fliter;
        key_flag=5'd1;//已经读取第一个加数
        end
    else q2=key_out_fliter;
if((q1!=5'd0&&q2!=5'd0&&q1!=q2)||(q1!=5'd0&&q2!=5'd0&&xiangtong==5'd1))
  qans=q1+q2;
        end
else if(key_out_fliter==5'd14)//清零重置按键
    begin
```

```
    qans=5'd0;
    q1=5'd0;
    q2=5'd0;
    key_flag=5'd0;//可以读取下一组数据
    xiangtong=5'd0;
    end
else if(key_out_fliter==5'd13)//相同读数按键
      xiangtong=5'd1;
end
assign led = {qans[4:0],3'b0};
```

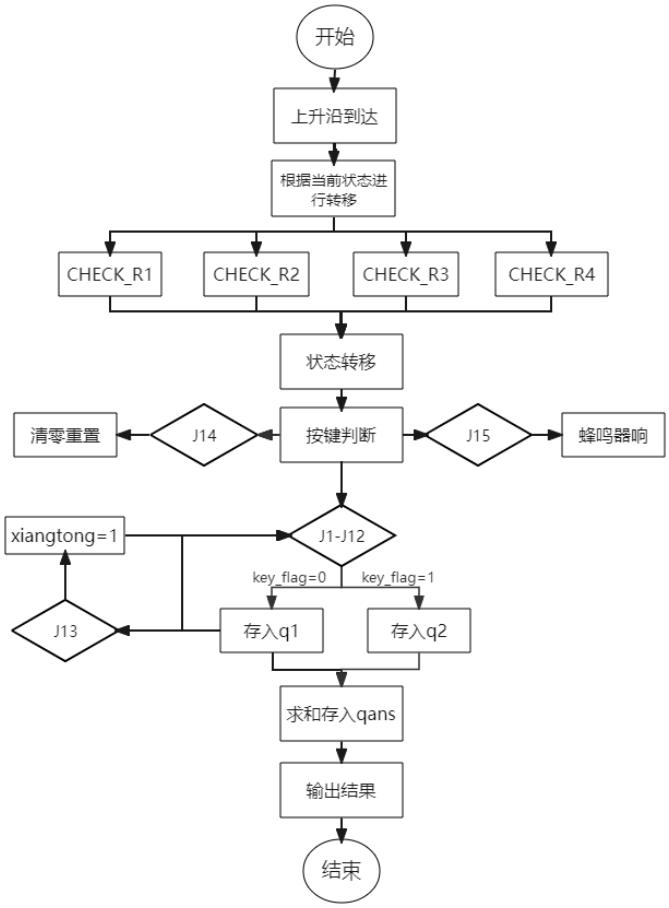

图 9 简易加法器流程图

综合以上代码模块，对加法器的整体设计思路进行汇总，绘制出如图 9 简易加法器流程图。

3 仿真结果

将程序编译后，进入仿真界面，设置时钟信号 clk、按键信号和输出信号 led，其内部电路图，输入输出信号以及仿真结果如图 10 图 11 图 12 所示。

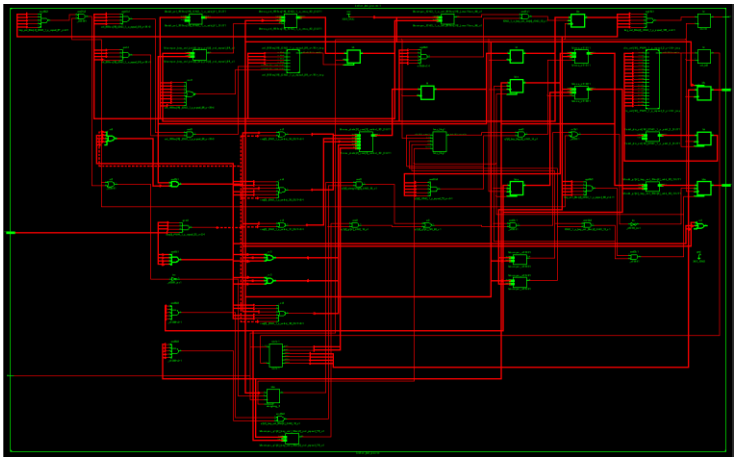

图 10 简易加法器内部电路图

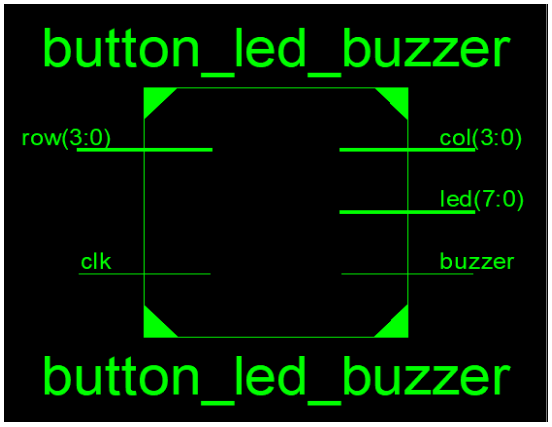

图 11 简易加法器输入输出信号

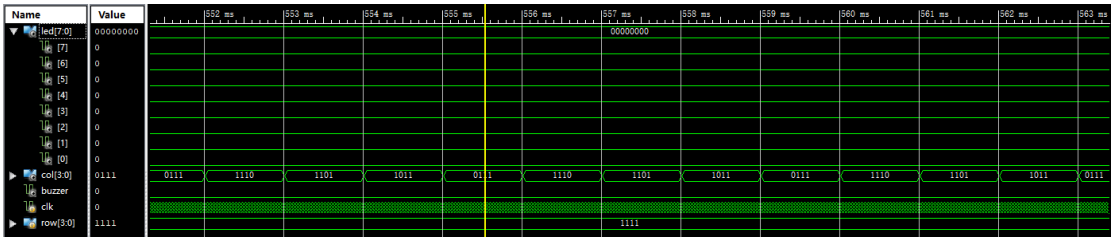

图 12 简易加法器仿真结果

4 电路板验证

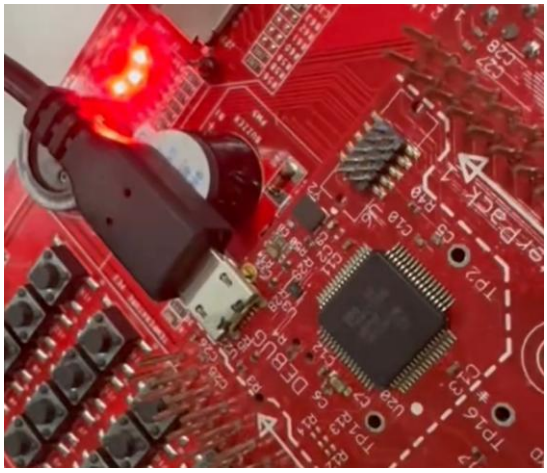

图 13 3+4 结果

图 14 6+6 结果

五 砸地鼠游戏设计

1 设计要求

分别实现顺序点亮和随机点亮八个 LED 灯中的一个，当 LED 亮起时若按下对应的按键，蜂鸣器就发出响声。

2 设计方案

我们的方案主要包含三个独立的模块：LED 模块、按键模块与蜂鸣器模块。接下来将对这三个模块进行说明。

2.1LED 模块

该模块的设计采用与流水灯相同的设计思路，设置一个计时器，当计时器到达阈值后就切换为下一个状态。记录 LED 状态的寄存器 led_reg 在顺序点亮 LED 时，下一个状态即为上一个状态循环左移 1

位的结果；在随机点亮 LED 时，下一个状态是上一个状态加 3 再对 8 取模的结果，也可以按照需求采用不同的随机方式。led_reg 同时作为与其他模块交互的接口。由于与流水灯代码几乎相同，此处不再展示。

2.2按键模块

按键模块主要功能为将按键信号转换为具体的按键编号。这部分的代码与加法器代码相同，因此不再过多赘述。最终产生名为 key_out_buf 的寄存器信号。

2.3蜂鸣器控制

蜂鸣器的逻辑较为简单，只需要当按键与 LED 寄存器的信号匹配，输出信号高，否则输出信号低。同时为了避免蜂鸣器无效长鸣的情况，我们还加了一个计时寄存器 time_cnt_1，每过 0.1 秒就置蜂鸣器为 0，如此蜂鸣器持续鸣叫的时长最多为 0.1 秒。

顺序砸地鼠代码如下：

```
reg [31:0] time_cnt_1 = 0;
always@(posedge clk)
if(time_cnt_1 == 32'd1000_0000)//蜂鸣器响0.1 秒后停止
time_cnt_1 <= 32'd0;
else
time_cnt_1 <= time_cnt_1 + 1'b1;
always@(posedge clk) begin
//led 的输出与按键匹配判断
if(led_reg==8'b0000_0001&&key_out_buf==5'd0)
buzzer <= 1'b1;
else if(led_reg==8'b0000_0010&&key_out_buf==5'd1)
buzzer <= 1'b1;
else if(led_reg==8'b0000_0100&&key_out_buf==5'd2)
buzzer <= 1'b1;
```

```
else if(led_reg==8'b0000_1000&&key_out_buf==5'd3)
buzzer <= 1'b1;
else if(led_reg==8'b0001_0000&&key_out_buf==5'd4)
buzzer <= 1'b1;
else if(led_reg==8'b0010_0000&&key_out_buf==5'd5)
buzzer <= 1'b1;
else if(led_reg==8'b0100_0000&&key_out_buf==5'd6)
buzzer <= 1'b1;
else if(led_reg==8'b1000_0000&&key_out_buf==5'd7)
buzzer <= 1'b1;
else if(time_cnt_1 == 32'd1000_0000)
buzzer <= 1'b0;//蜂鸣器鸣叫0.1 秒后停止
else
buzzer <= 1'b0;
end
```

随机砸地鼠部分代码如下：

```
always@(posedge clk)
if(time_rand == 32'd10000_0000)
    time_rand <= 32'd0;
else
    time_rand <= time_rand + 2'b11;

always@(posedge clk)
if(time_cnt >= update_interval)
    begin
        time_cnt <= 32'd0;
        rand <= time_rand % 8;
    end
else
    time_cnt <= time_cnt + 1'b1;
```

3 仿真结果

在程序编译完成后，进入仿真界面，将时钟信号 clk 和输出信号 led 添加到仿真文件中。随后可以看到对应的电路图、输入输出信号以及仿真波形结果，如图中所示：

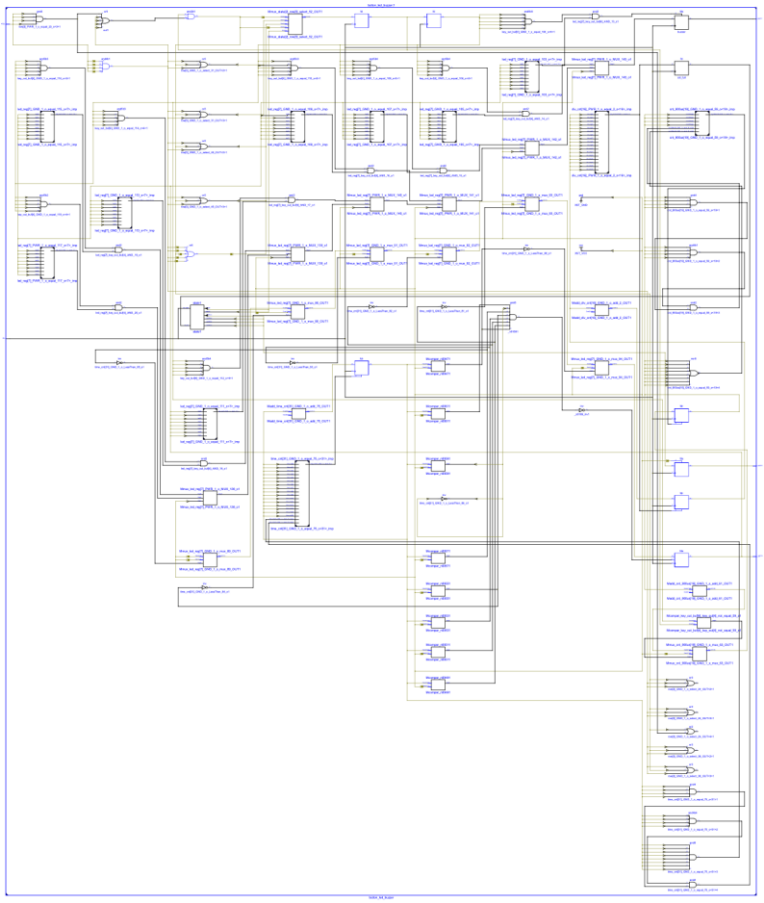

图 15 顺序打地鼠电路图

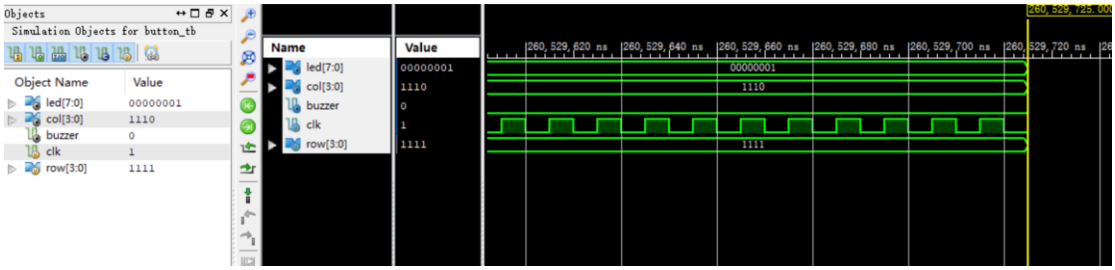

图 16 仿真结果图

图 17 随机打地鼠电路图

3.1 电路板验证

引脚配置后，编译工程，进行布局布线设计。将完全编译后产生的 bit 文件烧录至电路板。

顺序打地鼠：D1-D8 依次点亮，每个点亮时间持续 1 秒，代表地鼠顺序出现并停留 1 秒，在这 1 秒内按下相应的按键（J1-J8），若成功则蜂鸣器响起，表示砸地鼠成功。

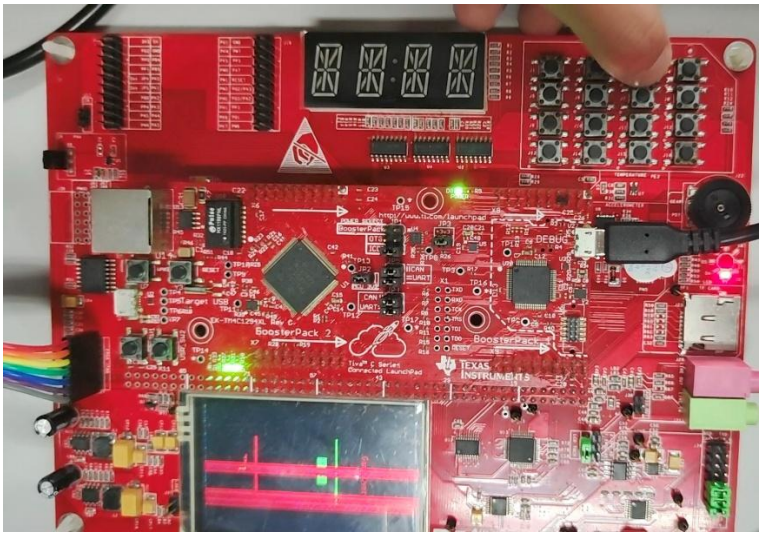

图 18 打地鼠电路板认证（1）

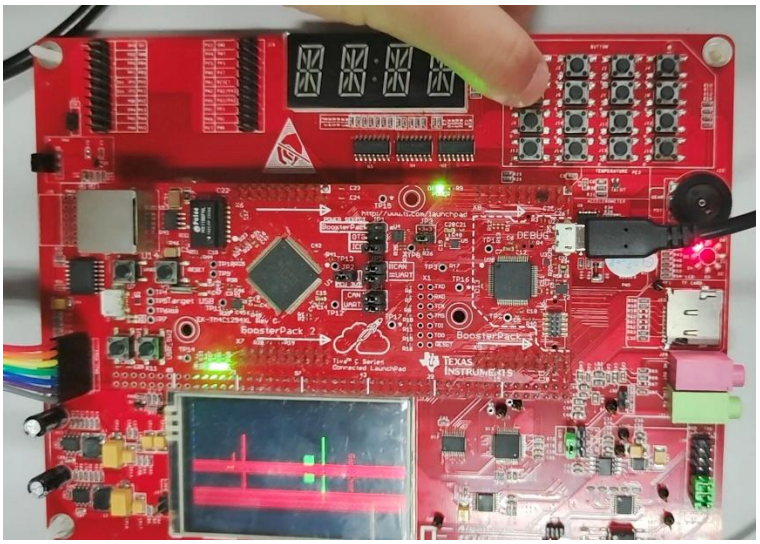

图 19 打地鼠电路板认证（2）

六　总结与体会

本次使用 Verilog 描述和仿真数字电路的过程，与以往教学中使用逻辑门图或真值表的方法有着明显不同。Verilog 提倡结构化、模块化的设计思路，虽然提高了电路设计的复杂性和可读性，但也对我们提出了更高的调试与错误定位要求。我们在学习如何用 Verilog

语言准确描述目标功能的过程中，也逐渐建立起对数字电路本质的理解和思维模式。

作为初次接触这门语言的学生，我们在实验中积累了许多宝贵的经验。例如，在设计过程中，将整体功能拆分成多个小模块再逐一实现，能显著提高开发效率和后期调试的便利性。同时，良好的代码注释习惯不仅方便自己后续阅读，也在团队合作中起到了沟通桥梁的作用。模块化设计也使我们发现，前期实验中实现的基础功能，在后续复杂设计中可以复用，体现了 Verilog 设计方法的可拓展性和系统性。

实验中的提高部分尤其激发了我们的创新思维。以“打地鼠”游戏为例，在实现基本功能的基础上，我们尝试加入关卡变速、手动调速等新特性，使设计更贴近真实产品。这一过程不仅锻炼了我们利用 Verilog 控制时序和状态机的能力，也帮助我们从“能运行”迈向“设计优化”。

当然，挑战也是不少的。由于 Verilog 并不像高级语言那样可以直接调试每一个变量，我们在仿真阶段难以实时交互输入，导致像打地鼠这类交互性强的功能难以在仿真环境下完全验证。同时，硬件测试时也无法像软件那样查看中间变量的状态，整个电路运行过程更像是一个“黑盒”，这在定位逻辑错误时无疑加大了难度。

尽管实验周期紧张，我们小组成员在一个周末几乎全天泡在实验室，从设计、验证到调试，一步步攻克难题。这段经历不仅锻炼了我们的专业技能，也提高了团队协作与问题应对的能力。更重要

的是，我们对 EDA 设计的实际流程、Verilog 语言的特点有了更深层的认知，也更加敬佩那些能从零开始设计芯片系统的工程师。希望未来我们也能不断提升自身能力，在数字设计这条路上越走越远。

七 参考文献

[1] 龙胜春，孙惠英，等. 电路与电子技术基础实验指导[M]. 北京：清华大学出版社，2015.

[2] Michael D. Ciletti 著，李广军，林水生，阎波，等译. Verilog HDL 高级数字设计（第二版）[M]. 北京：电子工业 出版社，2014.2

[3] 阎石，王红. 数字电子技术基础(第 6 版)[M]. 北京：高等教育出版社，2016.4（ISBN 9787040444933）

[4] FPGA 实践实验指导书［EB/OL］.2021. 06