北京理工大學
Beijing Institute of Technology

# 本科生实验报告

课程名称：数字电路实验

实验名称：数字电路实验软件仿真部分

| | | | | | |
|---|---|---|---|---|---|
| 任课教师： | [redacted] | | 实验教师： | [redacted] | |
| 实验日期： | [redacted] | | 实验地点： | [redacted] | |
| 实验类型： | ☐ 原理验证 ☑ 综合设计 ☐ 自主创新 | | | | |
| 学生姓名： | [redacted] | 班级： | [redacted] | 学号： | [redacted] |
| 学　　院： | 信息与电子学院 | | 专　　业： | 电子信息工程（徐特立英才班） | |
| 组　　号： | | 同组同学： | [redacted] | | |
| 成　　绩： | | | | | |

集成电路与电子学院
SCHOOL OF INTEGRATED CIRCUITS AND ELECTRONICS

# 目录

# 实验一 多个与门

## 一、实验目的

1. 掌握 Verilog 语法及仿真流程
2. 了解模块例化、Testbench 概念
3. 学习 Vivado 的安装与使用
4. 完成多路与门逻辑电路功能验证

## 二、实验原理（简要）

本实验通过 Verilog 编程语言实现 4 组二输入与门（and2 模块），将开发板 SW0-SW7 开关信号作为输入，LD2_0-LD2_3 LED 作为输出。每组与门逻辑可表示为：

- ✧ LD2_0= SW0 & SW1
- ✧ LD2_1= SW2 & SW3
- ✧ LD2_2= SW4 & SW5
- ✧ LD2_3= SW6 & SW7

## 三、实验开发环境及使用器件

1. Vivado
2. Ego-1 开发板
3. Xcom 串口调试工具

## 四、实验内容与主要步骤

### 4. 1 基础部分

1. 电路的实现代码（user_code.v）及功能描述

实现该电路的代码为：

```
// user_code.v
module user_code (
    input clk,
    input reset,
    input [7:0] sw7_sw0,
    output [7:0] led2
);
    // 实现 4 组二输入与门
    assign led2[0] = sw7_sw0[0] & sw7_sw0[1];
    assign led2[1] = sw7_sw0[2] & sw7_sw0[3];
    assign led2[2] = sw7_sw0[4] & sw7_sw0[5];
    assign led2[3] = sw7_sw0[6] & sw7_sw0[7];
endmodule
```

这段代码通过 4 组二输入与门（and2）处理 8 位开关信号（SW0-SW7），输出 4 位显示信号到 LED（LD2_0-LD2_3）模块。

- ✧ 输入端口：8 位开关信号 sw7_sw0（SW0 为最低位，SW7 为最高位）。
- ✧ 输出端口：4 位 LED 信号 led2（LD2_0 对应最低位，LD2_3 对应最高位）。

每个与门组合两个相邻开关信号（SW0&SW1、SW2&SW3 类推），最终生成 LED2[3:0]四个逻辑电平输出。

2. 管脚约束文件（ego1_pin.xdc）代码及功能描述；

管脚约束文件代码为：

```
set_property IOSTANDARD LVCMOS33 [get_ports PADi_SWCLK]
set_property IOSTANDARD LVCMOS33 [get_ports PADb_SWDIO]
set_property IOSTANDARD LVCMOS33 [get_ports PADi_clk_in]
set_property IOSTANDARD LVCMOS33 [get_ports PADi_resetn]
set_property IOSTANDARD LVCMOS33 [get_ports PADi_uart0_rxd]
set_property IOSTANDARD LVCMOS33 [get_ports PADo_uart0_txd]
```

```
7. set_property PACKAGE_PIN P17 [get_ports PADi_clk_in]
8. set_property PACKAGE_PIN G16 [get_ports PADb_SWDIO]
9. set_property PACKAGE_PIN P15 [get_ports PADi_resetn]
10.set_property PACKAGE_PIN T4 [get_ports PADo_uart0_txd]
11.set_property PACKAGE_PIN N5 [get_ports PADi_uart0_rxd]
12.set_property PACKAGE_PIN H16 [get_ports PADi_SWCLK]
13.
14.
15.create_clock -period 100.000 -name swclk -waveform {0.000 50.000}
   [get_ports PADi_SWCLK]
16.set_property BITSTREAM。常规。压缩真 [current_design]
17.set_property CONFIG_VOLTAGE 3.3 [current_design]
18.set_property CFGBVS VCCO [current_design]
19.set_property BITSTREAM.CONFIG。未使用的 PULLNONE [current_design]
20.
21.
22.
23.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW7_SW0[0]}]
24.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW7_SW0[1]}]
25.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW7_SW0[2]}]
26.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW7_SW0[3]}]
27.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW7_SW0[4]}]
28.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW7_SW0[5]}]
29.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW7_SW0[6]}]
30.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW7_SW0[7]}]
31.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD1[0]}]
32.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD1[1]}]
33.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD1[2]}]
34.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD1[3]}]
35.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD1[4]}]
36.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD1[5]}]
37.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD1[6]}]
38.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD1[7]}]
39.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD2[0]}]
40.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD2[1]}]
41.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD2[2]}]
42.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD2[3]}]
43.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD2[4]}]
44.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD2[5]}]
45.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD2[6]}]
46.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_LD2[7]}]
47.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_0[0]}]
```

```
48.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_0[1]}]
49.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_0[2]}]
50.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_0[3]}]
51.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_0[4]}]
52.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_0[5]}]
53.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_0[6]}]
54.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_0[7]}]
55.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_0_sel[0]]
56.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_0_sel[1]]
57.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_0_sel[2]]
58.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_0_sel[3]]
59.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_1[0]}]
60.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_1[1]}]
61.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_1[2]}]
62.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_1[3]}]
63.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_1[4]}]
64.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_1[5]}]
65.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_1[6]}]
66.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_1[7]}]
67.set_property IOSTANDARD LVCMOS33 [get_ports PADi_push_button_middle]
68.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_1_sel[0]}]
69.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_1_sel[1]]
70.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_1_sel[2]]
71.set_property IOSTANDARD LVCMOS33 [get_ports {PADo_segments_1_sel[3]]
72.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW[0]}]
73.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW[1]}]
74.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW[2]}]
75.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW[3]}]
76.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW[4]}]
77.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW[5]}]
78.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW[6]}]
79.set_property IOSTANDARD LVCMOS33 [get_ports {PADi_SW[7]}]
80.set_property PACKAGE_PIN U3 [get_ports {PADi_SW[7]}]
81.set_property PACKAGE_PIN U2 [get_ports {PADi_SW[6]}]
82.set_property PACKAGE_PIN V2 [get_ports {PADi_SW[5]}]
83.set_property PACKAGE_PIN V5 [get_ports {PADi_SW[4]}]
84.set_property PACKAGE_PIN V4 [get_ports {PADi_SW[3]}]
85.set_property PACKAGE_PIN R3 [get_ports {PADi_SW[2]}]
86.set_property PACKAGE_PIN T3 [get_ports {PADi_SW[1]}]
87.set_property PACKAGE_PIN T5 [get_ports {PADi_SW[0]}]
88.set_property PACKAGE_PIN K1 [get_ports {PADo_LD1[7]}]
89.set_property PACKAGE_PIN H6 [get_ports {PADo_LD1[6]}]
```

```
set_property PACKAGE_PIN H5 [get_ports {PADo_LD1[5]}]
set_property PACKAGE_PIN K6 [get_ports {PADo_LD1[3]}]
set_property PACKAGE_PIN L1 [get_ports {PADo_LD1[2]}]
set_property PACKAGE_PIN M1 [get_ports {PADo_LD1[1]}]
set_property PACKAGE_PIN K3 [get_ports {PADo_LD1[0]}]
set_property PACKAGE_PIN F6 [get_ports {PADo_LD2[7]}]
set_property PACKAGE_PIN P5 [get_ports {PADi_SW7_SW0[7]}]
set_property PACKAGE_PIN P4 [get_ports {PADi_SW7_SW0[6]}]
set_property PACKAGE_PIN P3 [get_ports {PADi_SW7_SW0[5]}]
set_property PACKAGE_PIN P2 [get_ports {PADi_SW7_SW0[4]}]
set_property PACKAGE_PIN R2 [get_ports {PADi_SW7_SW0[3]}]
set_property PACKAGE_PIN M4 [get_ports {PADi_SW7_SW0[2]}]
set_property PACKAGE_PIN N4 [get_ports {PADi_SW7_SW0[1]}]
set_property PACKAGE_PIN R1 [get_ports {PADi_SW7_SW0[0]}]
set_property PACKAGE_PIN G4 [get_ports {PADo_LD2[6]}]
set_property PACKAGE_PIN G3 [get_ports {PADo_LD2[5]}]
set_property PACKAGE_PIN J4 [get_ports {PADo_LD2[4]}]
set_property PACKAGE_PIN H4 [get_ports {PADo_LD2[3]}]
set_property PACKAGE_PIN J3 [get_ports {PADo_LD2[2]}]
set_property PACKAGE_PIN J5 [get_ports {PADo_LD1[4]}]
set_property PACKAGE_PIN J2 [get_ports {PADo_LD2[1]}]
set_property PACKAGE_PIN K2 [get_ports {PADo_LD2[0]}]
set_property PACKAGE_PIN R15 [get_ports PADi_push_button_middle]
set_property PACKAGE_PIN D5 [get_ports {PADo_segments_0[7]}]
set_property PACKAGE_PIN B2 [get_ports {PADo_segments_0[6]}]
set_property PACKAGE_PIN B3 [get_ports {PADo_segments_0[5]}]
set_property PACKAGE_PIN A1 [get_ports {PADo_segments_0[4]}]
set_property PACKAGE_PIN B1 [get_ports {PADo_segments_0[3]}]
set_property PACKAGE_PIN A3 [get_ports {PADo_segments_0[2]}]
set_property PACKAGE_PIN A4 [get_ports {PADo_segments_0[1]}]
set_property PACKAGE_PIN B4 [get_ports {PADo_segments_0[0]}]
set_property PACKAGE_PIN H1 [get_ports {PADo_segments_0_sel[3]}]
set_property PACKAGE_PIN C1 [get_ports {PADo_segments_0_sel[2]}]
set_property PACKAGE_PIN C2 [get_ports {PADo_segments_0_sel[1]}]
set_property PACKAGE_PIN G2 [get_ports {PADo_segments_0_sel[0]}]
set_property PACKAGE_PIN H2 [get_ports {PADo_segments_1[7]}]
set_property PACKAGE_PIN D2 [get_ports {PADo_segments_1[6]}]
set_property PACKAGE_PIN E2 [get_ports {PADo_segments_1[5]}]
set_property PACKAGE_PIN F3 [get_ports {PADo_segments_1[4]}]
set_property PACKAGE_PIN F4 [get_ports {PADo_segments_1[3]}]
set_property PACKAGE_PIN D3 [get_ports {PADo_segments_1[2]}]
set_property PACKAGE_PIN E3 [get_ports {PADo_segments_1[1]}]
```

```
set_property PACKAGE_PIN D4 [get_ports {PADo_segments_1[0]}]
set_property PACKAGE_PIN G6 [get_ports {PADo_segments_1_sel[3]}]
set_property PACKAGE_PIN E1 [get_ports {PADo_segments_1_sel[2]}]
set_property PACKAGE_PIN F1 [get_ports {PADo_segments_1_sel[1]}]
set_property PACKAGE_PIN G1 [get_ports {PADo_segments_1_sel[0]}]


create_clock -period 10.000 -name clk_soc -波形 {0.000 5.000} [get_pins u_autocheck_tool/clock_reset/clk_wiz_0/inst/clk_out1]
create_clock -period 10.000 -name clk_user -波形 {0.000 5.000} [get_pins u_autocheck_tool/clock_reset/clk_wiz_0/user_clk]
create_generated_clock -name clk_check -source [get_pins BUFGCE_inst/I0] -multiply_by 1 -add -master_clock clk_user [get_pins BUFGCE_inst/O]
set_clock_groups -name 异步 -group [get_clocks {clk_check clk_user}] -group [get_clocks swclk] -group [get_clocks clk_soc]
```

管脚约束文件的设计采用 3.3V LVCMOS33 电平标准，规范硬件接口的引脚分配和时序关系。系统架构包含以下关键模块：

1. 外设控制接口：

✧ JTAG 调试接口：通过 PADi_SWCLK（P17）和 PADb_SWDIO（G16）引脚建立调试通道。

✧ UAR 串行通信：通过 PADi_uart0_rxd（N5）接收数据，经 PADo_uart0_txd（T4）发送数据，完成串口收发。

✧ LED 开关：8 位拨码开关 PADi_SW7_SW0[7:0]（物理端口 P2-R2）控制两组组 LED 阵列输出（PADo_LD1[7:0]和 PADo_LD2[7:0]）。

✧ 显示模块：段码信号由 PADo_segments_0/1[6:0]驱动，位选信号通过 PADo_segments_X_sel[3:0]控制。

✧ 扩展输入：中间按键（PADi_push_button_middle，R15）及扩展开关组（PADi_SW[7:0]）接入控制总线。

2. 时钟架构管理：

✧ 主时钟 clk_soc（10MHz）与用户时钟 clk_user 保持同步，JTAG 时钟 swclk 采用异步独立运行机制，防止时序冲突。

3. 比特流配置策略：

✧ 启用 BITSTREAM.COMPRESS 压缩选项缩减配置数据量，设定 3.3V 为配置电压（CONFIG_VOLTAGE），未连接引脚配置为高阻模式（PULLNONE）。

通过上述约束定义，有效保障信号传输质量、电气兼容性及系统时序稳定性，满足功能实现要求。

## 4.2 实验结果

1. 软件仿真结果

仿真结果如下图所示：

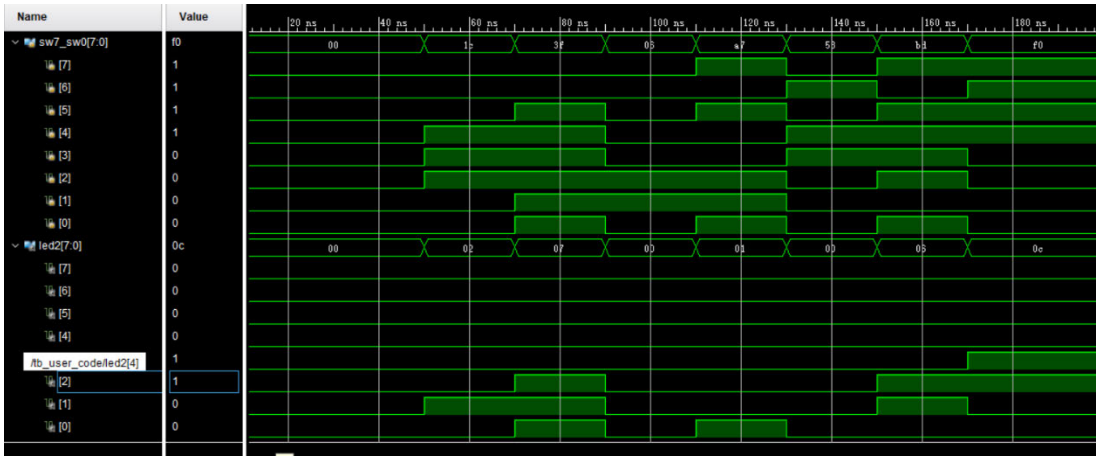


图 1-1：软件仿真结果

由上图的仿真波形分析可知，电路在不同工作状态下均能产生符合预期的逻辑输出。仿真数据与设计规范完全吻合，因此可生成 bit 流文件并烧录至 EGO-1 开发板进行硬件验证。

2. 下板运行结果

生成 bit 流文件后，将配置文件烧录至 EGO-1 开发板进行物理验证。在各种情况下硬件验证结果如图 1-2 至 1-6 所示，并判断与设计预想是否一致。

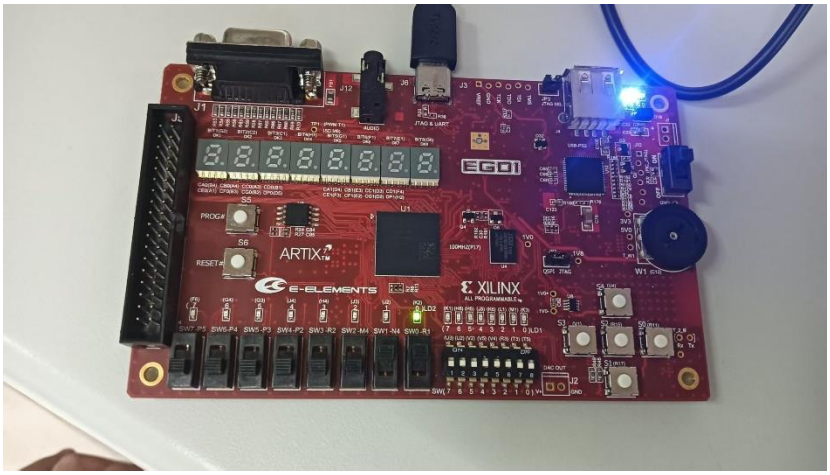

图 1-2：ex1_1 实验结果

✧ ex1_1：SW0、SW1 均输入 1 时，LED_0 发光，符合设计要求。

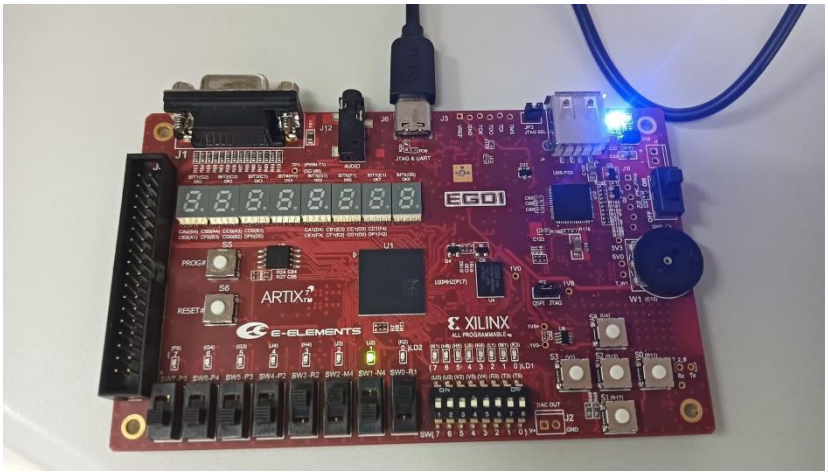

图 1-3：ex1_2 实验结果

✧ ex1_2：SW0 输入 1、SW1 输入 0 状态下 LED_0 熄灭；SW2、SW3 均输入 1 时触发 LED_1 点亮，电路的响应与预期相同。

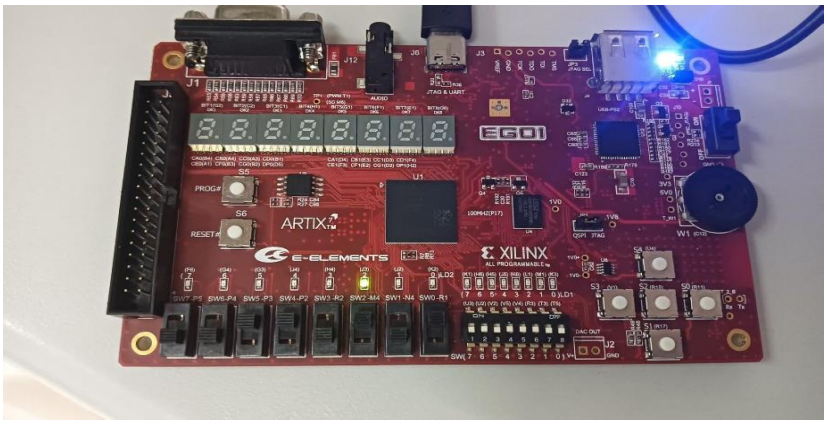

图 1-4：ex1_3 实验结果

✧ ex1_3：SW0 输入 1、SW1 输入 0 状态下 LED_0 熄灭；SW2 输入 1、SW3 输入 0 状态下 LED_1 熄灭；SW4、SW5 均输入 1 时 LED_2 发光，电路的响应与预期相同。

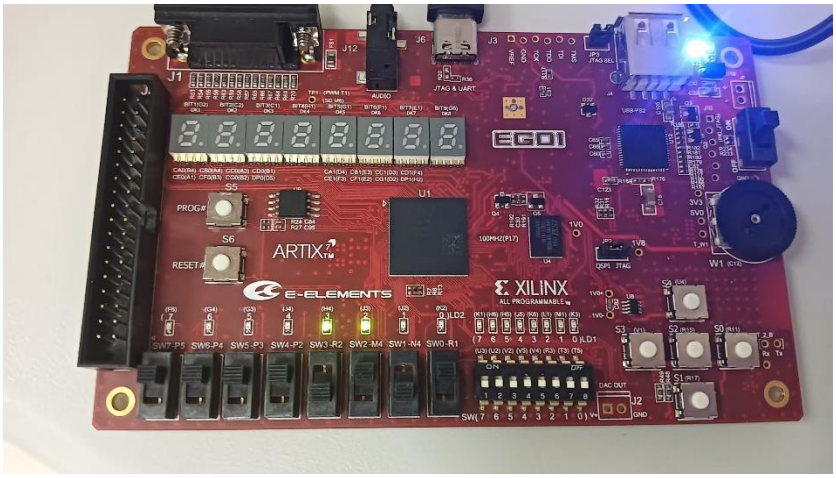

图 1-5：ex1_4 实验结果

✧ ex1_4：SW0 输入 1、SW1 输入 0 状态下 LED_0 熄灭；SW2 输入 1、SW3 输入 0 状态下 LED_1 熄灭；SW4、SW5 均输入 1 时 LED_2 发光；SW6、SW7 输入 1 时 LED_3 发光，电路的响应与预期相同。

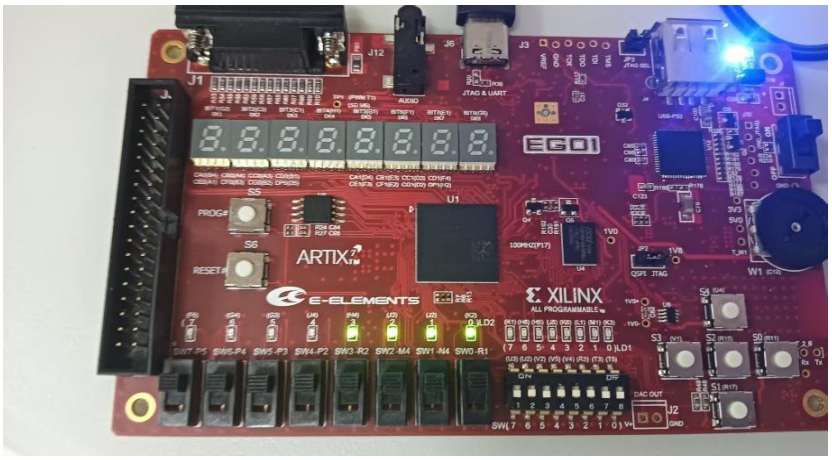

图 1-6：ex1_5 实验结果

✧ ex1_5：SW0、SW1 均输入 1 时 LED_0 发光；SW2、SW3 均输入 1 时 LED_1 发光；SW4、SW5 均输入 1 时 LED_2 发光；SW6、SW7 均输入 1 时 LED_3 发光，电路的响应与预期相同。

# 五、作业

## 5.1 作业 1

仿真显示波形如下所示：

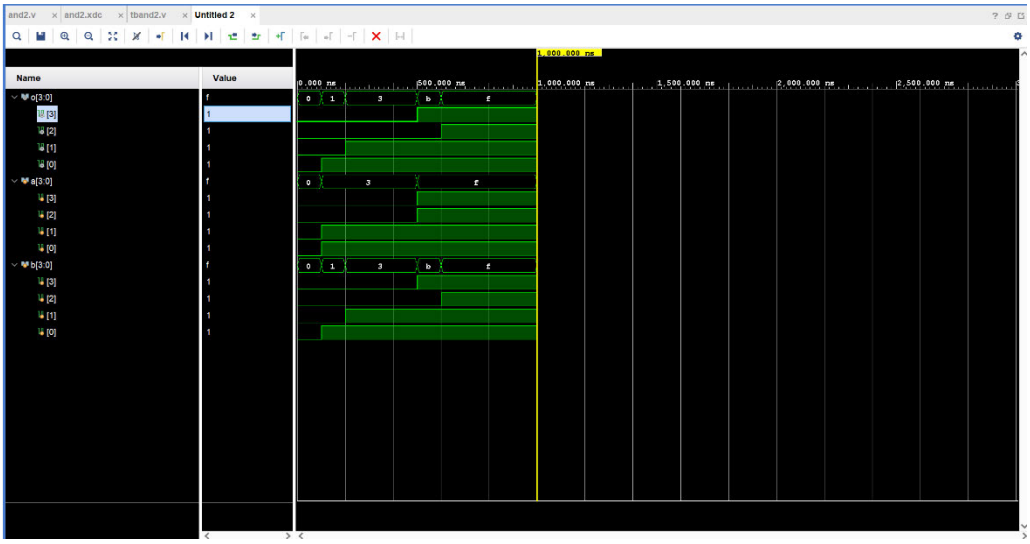


图 1-7：显示波形

仿真波形与 tb 中激励的时序对应关系分析如下：

✧ 初始状态（0~100ns）：

输入 a = 0000，b = 0000 时，输出 o = 0。这与与门的特性全 1 出 1，否则全 0 相符合。

波形特点：a[3:0] 和 b[3:0] 全为低电平，o[3:0] 全为低电平。

✧ 100ns~200ns：

a[0]跳变为高电平，即 a=0001，b=0000 时，由于 b[0]仍为低电平，o[0]维持低电位未触发，因此输出为 o[0] = 0。

波形特点：o[0] 在 100ns 时由 0 变为 0，即不发生变化，其他输出保持 0 不变。

✧ 200ns~300ns：

a[1]置位，即 a=0011，同时 b[0]升为高电平，即 b=0001。因此 a[0]&b[0]=1，则 o[0]=1。

波形特点：o[0]在 200ns 时产生上升沿。

✧ 300ns~400ns：

b[1]电平提升，即 b=0011。使得 a[1]&b[1]=1 成立，则 o[1] = 1。

波形特点：o[1]在 300ns 时由低转高，呈现级联触发特性

✧ 400ns~500ns：

a[3:2] = 11,即 a = 1100，b[3:2] = 10，即 b = 1001。因此 a[3]=1, b[3]=1，则 o[3] = 1。

波形特点：o[3]在 400ns 时跃升为高电平

## 5.2 作业 2

管脚截图如下图所示：

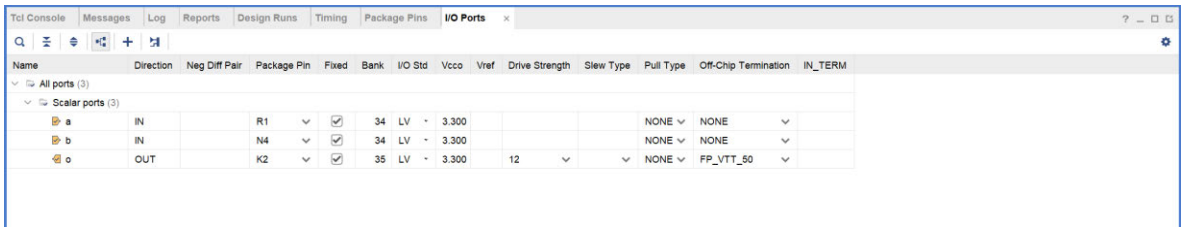


图 1-8：管脚情况

## 5.3 作业 3

文件内容如下图所示：

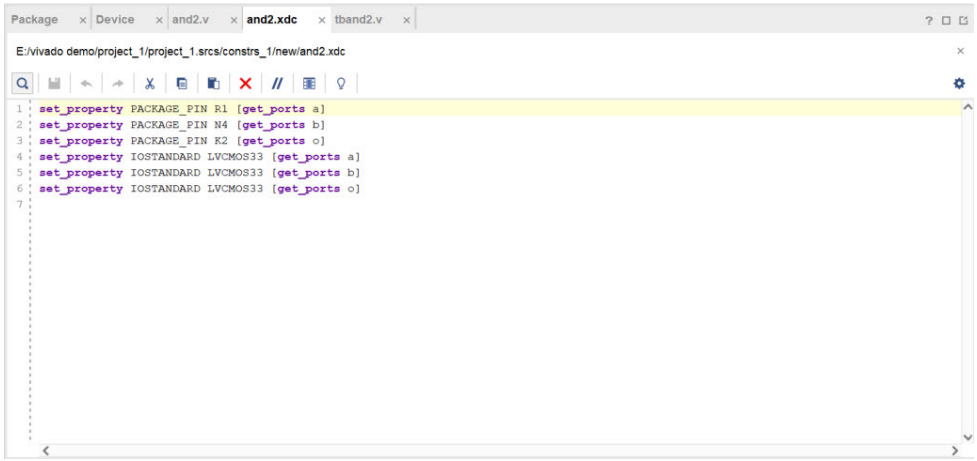


图 1-9：文件内容

## 5.4 作业 4

观察 Device 窗口，找到高亮的 LUT。找到高亮的 IO，位置如下图所示：

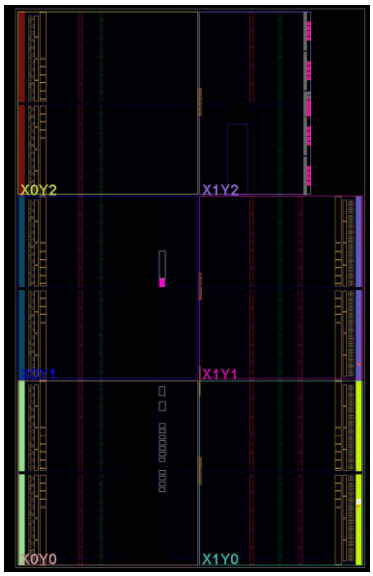


图 1-10：高亮的 LUT、高亮的 IO 位置

LUT 的高亮区域位于芯片中心区域或分散在多个 CLB 单元内，而 IO 的高亮区域则沿芯片边缘分布。通过布线追踪可见，LUT 输出信号经由金属互连层布线路径，最终接入边缘 I/O 缓冲电路完成信号传输。

## 5.5 作业 5

文件内容如下图所示：

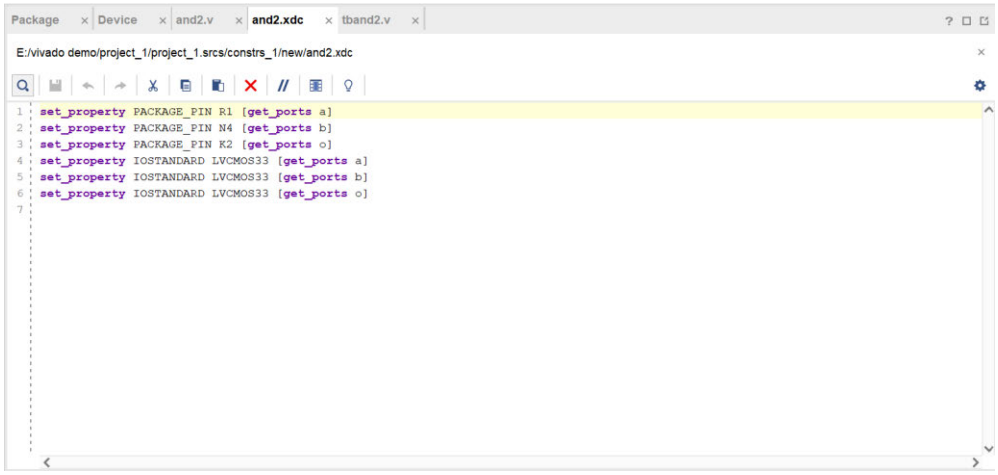


图 1-11：文件内容

由上图可知，文件内容与作业三相比完全一样。这是因为在前面的操作过程中我们没有改动管脚分配和程序代码。

## 六、总结

通过第二章的组合逻辑电路设计与验证实验，我系统性地掌握了基于 Verilog 语言的 FPGA 开发全流程，并深化了对数字电路设计理论的理解。实验以二输入与门为切入点，覆盖代码编写、功能仿真、逻辑综合、硬件部署等完整环节，使我对基于 Vivado 进行数字电路开发形成了具体的认识。

在 Verilog 开发实践中，我始终遵循模块化设计思想。通过参数化设计和层次化建模等方法，实现了逻辑功能的封装与复用。在仿真环节我们通过 Testbench 施加激励，观察波形验证电路的逻辑功能，这也进一步说明了我们之前理论设计的正确性。值得注意的是，代码风格直接影响综合结果。初始设计因未正确处理信号初始值，导致仿真中出现高阻态和未定义状态，后通过`initial`语句显式赋初值解决了该问题，这使我认识到硬件描述语言与软件编程的本质差异。

在电路的硬件实现过程中，我们要对照原理图进行正确的引脚分配和约束文件的编写，这样才能保证实现我们所设计电路的逻辑功能。与此同时在实验开始时，我们的工程路径中出现了中文字符或空格而使我们的仿真功能出现了异常，最终通过仔细排查才得以解决。这也是我更加意识到规范配置开发环境的重要性。

本阶段实验聚焦基础逻辑门，进一步加深了我对数字电路理论的认识，初步掌握了通过 Verilog 语言进行 FPGA 开发的方法，同时也培养了我的动手实践能力。这为我在后续实验中进行进一步的探究打下了良好的基础。

# 实验二 七段二进制数码管

## 一、实验目的

1. 了解 FPGA 实现逻辑功能的底层原理。
2. 进一步学习 Verilog 语法。
3. 了解七段二进制数码管的显示逻辑。

## 二、实验原理（简要）

本实验通过开发板 SW0-SW3 四位拨码开关输入四位二进制编码信号，经过译码逻辑电路处理后，驱动左侧首位七段数码管（DK1）动态显示对应的十六进制数值（0 至 F），完成 4 位二进制数据到十六进制数码显示的实时转换功能。

## 三、实验开发环境及使用器件

1. Vivado
2. Ego-1 开发板
3. Xcom 串口调试工具

## 四、实验内容与主要步骤

### 4.1 基础部分

1. 电路的实现代码（user_code.v）及功能描述

实现该电路的代码为：

```
1. module user_code(
2.     input [7:0] sw7_sw0,         // 输入：拨码开关 SW0~SW7（使用低 4 位 SW0~SW3）
```

```
    output [7:0] segments,       // 输出：七段数码管段选信号（共阴极，a~g段，最高位为小数点）
    output dn0_k1                // 输出：位选信号（固定使能最左侧数码管DK1）
);

// 调用译码函数，输入为 SW0~SW3（sw7_sw0[3:0]）
assign segments = gen_7ledoutput(sw7_sw0[3:0]);
assign dn0_k1 = 1'b1;           // 使能最左侧数码管

// 七段数码管译码函数（4 位二进制转七段码）
function [7:0] gen_7ledoutput(input [3:0] binary_input);
    case (binary_input)
        4'h0: gen_7ledoutput = 8'h3F;  // 显示 0
        4'h1: gen_7ledoutput = 8'h06;  // 显示 1
        4'h2: gen_7ledoutput = 8'h5B;  // 显示 2
        4'h3: gen_7ledoutput = 8'h4F;  // 显示 3
        4'h4: gen_7ledoutput = 8'h66;  // 显示 4
        4'h5: gen_7ledoutput = 8'h6D;  // 显示 5
        4'h6: gen_7ledoutput = 8'h7D;  // 显示 6
        4'h7: gen_7ledoutput = 8'h07;  // 显示 7
        4'h8: gen_7ledoutput = 8'h7F;  // 显示 8
        4'h9: gen_7ledoutput = 8'h6F;  // 显示 9
        4'ha: gen_7ledoutput = 8'h77;  // 显示 A
        4'hb: gen_7ledoutput = 8'h7C;  // 显示 B
        4'hc: gen_7ledoutput = 8'h39;  // 显示 C
        4'hd: gen_7ledoutput = 8'h5E;  // 显示 D
        4'he: gen_7ledoutput = 8'h79;  // 显示 E
        4'hf: gen_7ledoutput = 8'h71;  // 显示 F
        default: gen_7ledoutput = 8'hFF;  // 无效输入时全灭（可根据硬件调整）
    endcase
endfunction

endmodule
```

这段代码将 4 位拨码开关输入（SW0~SW3）转换为七段数码管显示的十六进制数字（0~F），显示方法如下所示：

- ✧ 拨码开关输入 0000，数码管显示 0
- ✧ 输入 0001，数码管显示 1

✧ 输入 0010，数码管显示 2

✧ ……

✧ 输入 1111，数码管显示 F

代码通过上述的显示方法将二进制输入译码为相应的数码管段选信号，从而使最左侧的数码管显示相应的数字。

2. 管脚约束文件（ego1_pin.xdc）代码及功能描述；

与实验一相同。

## 4.2 实验结果

1. 软件仿真结果

仿真结果如下图所示：

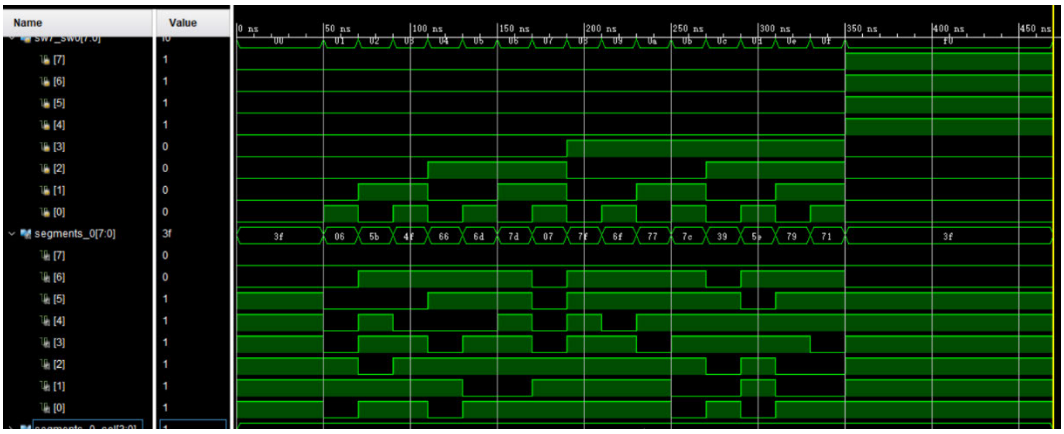


图 2-1：软件仿真结果

通过图 2-1 的仿真波形分析可知，在全部输入信号测试中，七段数码管显示的信号与预期十六进制字符的发光段完全匹配，仿真结果与预期设计一致。因此可将程序生成的 bit 文件并烧录至 EGO-1 开发板进行硬件验证。

2. 下板运行结果

生成 bit 流文件后，将配置文件烧录至 EGO-1 开发板进行物理验证。在部分情况的测试中，电路的测试结果示意图如下所示：

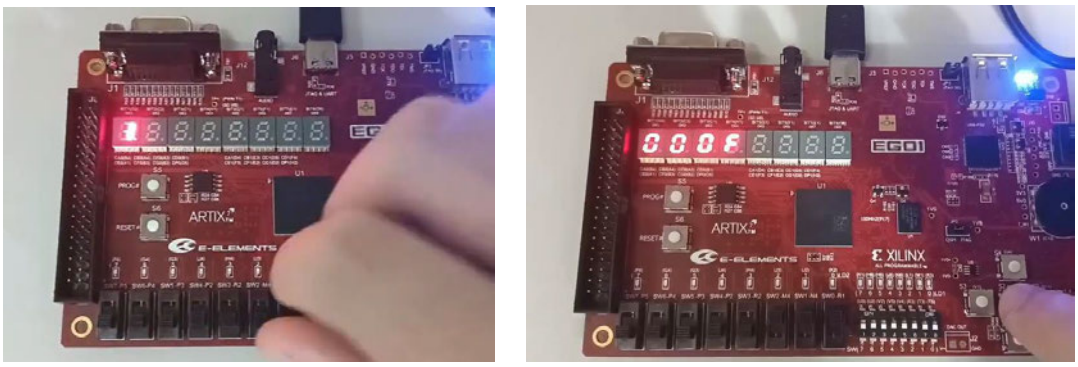

图 2-2：电路测试结果

完整测试流程见以下的视频链接：

**https://www.bilibili.com/video/BV1djjZzyEJQ/?share_source=copy_web&vd_source=d19749656fe7d6613a2372ba024a967c**

拨动操作开发板上的 SW0-SW3 拨码开关组（4 位二进制输入），实时转换并驱动最左侧七段数码管（DK1）动态显示对应的十六进制字符（0 至 F）。实测表明：当输入二进制码 0000（0）时显示字符"0"，输入 0001（1）对应"1"，直至输入 1111（F）准确显示"F"，实现了 4 位二进制输入转换为七段数码管显示的组合逻辑功能，也进一步验证了上述译码表与硬件驱动的一致性。

# 五、作业

## 5.1 作业 1

Open Implemented Design，定位使用的 LUT，截图如下：

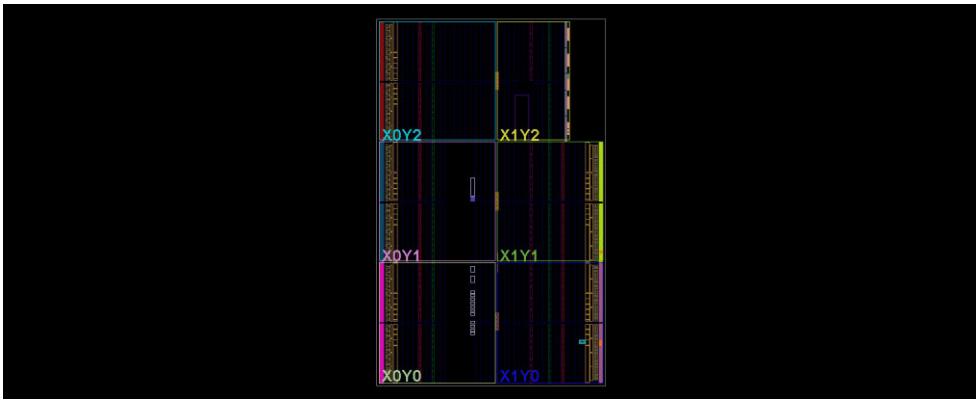


图 2-3：LUT 截图

FPGA 架构中 LUT 与 IO 相对位置关系呈现以下特征：

✧ IO 物理引脚通过输入缓冲器连接到 FPGA 内部布线网络。

✧ LUT 的输入端口通过布线资源接收来自 IO 的输入信号驱动，其输出端口（O）连接至数码管段选引脚的输出缓冲器。

## 5.2 作业 2

再次 Open Implemented Design，定位使用的 LUT，截图如下：

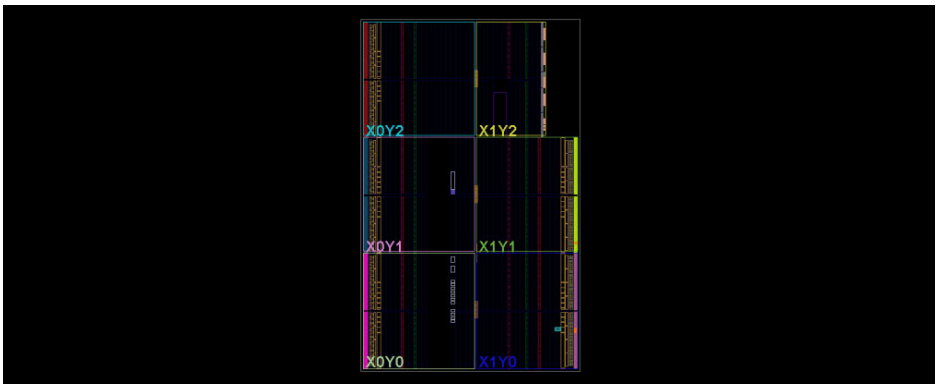


图 2-4：LUT 截图

LUT 与 IO 相对位置关系与作业 1 相同。

## 5.3 作业 3

再次 Open Implemented Design，再次定位使用的 LUT，截图如下：

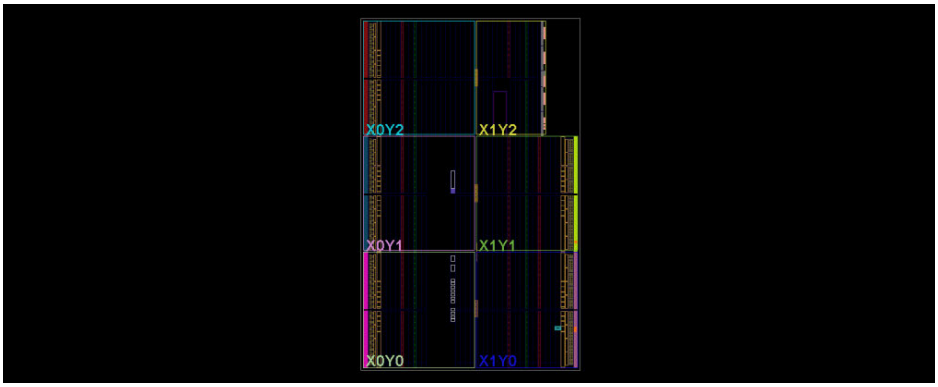


图 2-5：LUT 截图

LUT 与 IO 相对位置关系与作业 1 相同。

## 5.4 作业 4

查看 RTL ANALYSIS 中的 Schematic，截图如下：

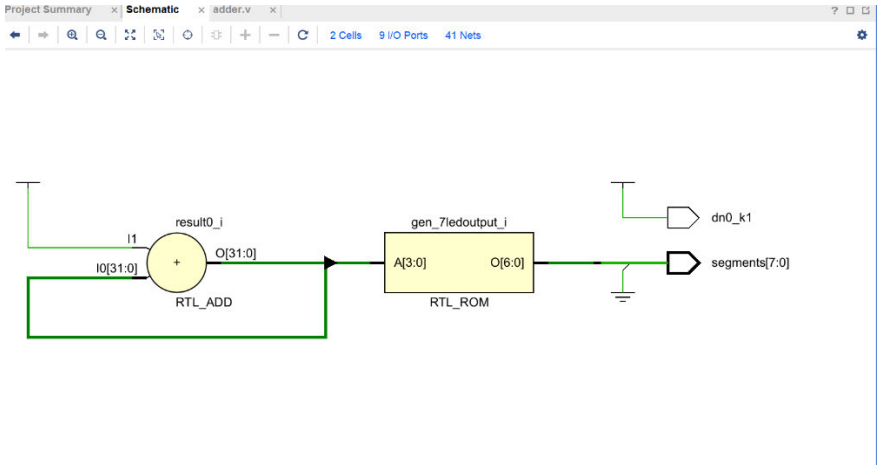


图 2-6：Schematic 截图

## 5.5 作业 5

写出多路选择器使用的 LUT3 每个小球的值，以及其与对应输入的关系，类比真值表的格式，用于分析的 Schematic 截图和真值表如下所示：

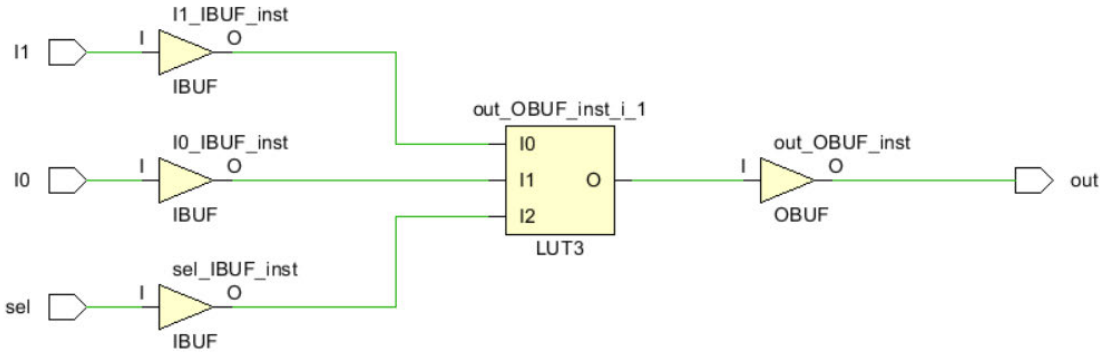


图 2-7：用于分析 LUT3 的截图

| sel | I1 | I0 | out |
|---|---|---|---|
| 0 | x | 0 | 0 |
| 0 | x | 1 | 1 |
| 1 | 0 | x | 0 |
| 1 | 1 | x | 1 |

图 2-8：真值表格式

# 六、总结

通过实验 2 的七段二进制数码管开发实践，我对 FPGA 硬件实现机制建立了多维度的认知。4 位二进制输入信号至十六进制的显示转换，本质是通过 LUT 物理实现真值表的硬件映射过程，这体现了 FPGA 通过配置存储单元(SRAM)重构逻辑电路的核心原理。

在使用 Verilog 编写译码函数时，我们采用了`case`语句实现。这种直接映射简单易行，但需反复核对段码与数码管极性是否匹配。我们在初始设计中因忽略数码管共阳极特性导致段码表逻辑值与实际显示相反，后通过逐位调节才解决这个问题。这使我认识到硬件描述语言需严格遵循物理电气特性，更直观地感知硬件设计与软件编程之间的差异。在 FPGA 设计中，所设计的与、或、非等逻辑运算，在硬件内部的是通过查找表的方式来实现。下板调试时，我们成功实现了七段数码管的实时转换功能，也对其工作原理有了更深刻的了解。

在本次实验中，我对组合逻辑电路的工作原理有了更深刻的理解，也通过动手实践实现了七段数码管的预期功能，提升了我的动手实践能力。在后续的实验中我也将认真对待，争取取得新的突破。

# 实验三 Let the Lights On

## 一、实验目的

1. 了解时序逻辑的基础概念及 D 触发器的 Verilog 描述方式。
2. 了解七段二进制数码管的扫描逻辑
3. 会根据给出程序仿写出驱动七段二进制数码管的电路。

## 二、实验原理（简要）

D 触发器在时钟信号的上升沿或下降沿，会把当时的输入信号 D 复制到次态输出 Q。

基于 D 触发器设计的时序逻辑电路实现左侧四个数码管（DK1~DK4）的十六进制递增显示，具体原理和功能如下所示：

1. 初始状态设定：当复位信号（高电平有效）触发时，仅最左侧数码管 DK1 被点亮，显示数值 “0”，其余三个数码管处于熄灭状态，整体显示数值为 “0000”。
2. 数值递增机制：在复位信号解除后，数码管显示数值将以 250ms 为周期，按照十六进制规则进行递增，从 “0000” 逐步变化至 “FFFF”，达到最大值后重新从 “0000” 开始循环。
3. 动态扫描显示：采用动态扫描技术驱动数码管，每隔 5ms 切换一次显示位。按照 DK1、DK2、DK3、DK4 的顺序依次点亮，每个数码管点亮时长为 5ms，利用人眼视觉暂留特性，实现四个数码管的同步显示效果。
4. 时钟与复位控制：系统使用板载 100MHz 时钟信号作为时序基准，复位信号为高电平有效。一旦复位解除，数值递增与动态扫描逻辑即刻启动运行。

## 三、实验开发环境及使用器件

1. Vivado
2. Ego-1 开发板

3. Xcom 串口调试工具

# 四、实验内容与主要步骤

## 4.1 基础部分

1. 电路的实现代码（user_code.v）及功能描述

实现该电路的代码为：

```
module user_code(
    input wire clk,             // 100MHz 系统时钟
    input wire reset,           // 复位信号（高有效）
    output reg [7:0] seg,       // 数码管段选信号
    output reg [3:0] en         // 数码管位选信号（低有效）
);

    // 内部计数器和寄存器
    reg [31:0] clk_div;         // 时钟分频计数器
    reg [1:0] scan_count;       // 数码管扫描计数器
    reg [15:0] display_data;    // 显示数据寄存器

    // 250ms 计数器（100MHz 时钟，250ms = 25,000,000 个周期）
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            clk_div <= 32'd0;
            display_data <= 16'd0;
        end
        else begin
            if (clk_div >= 32'd24_999_999) begin
                clk_div <= 32'd0;
                display_data <= display_data + 16'd1;  // 每 250ms 递增
            end
            else begin
                clk_div <= clk_div + 32'd1;
            end
        end
    end

    // 5ms 扫描计数器（100MHz 时钟，5ms = 500,000 个周期）
```

```
    reg [18:0] scan_div;
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            scan_div <= 19'd0;
            scan_count <= 2'd0;
        end
        else begin
            if (scan_div >= 19'd499_999) begin
                scan_div <= 19'd0;
                scan_count <= scan_count + 2'd1;  // 每 5ms 切换一位
                if (scan_count >= 2'd3)
                    scan_count <= 2'd0;
            end
            else begin
                scan_div <= scan_div + 19'd1;
            end
        end
    end

    // 数码管扫描控制
    always @(*) begin
        case (scan_count)
            2'd0: begin en = 4'b1110; end  // 选中 DK1（最左侧）
            2'd1: begin en = 4'b1101; end  // 选中 DK2
            2'd2: begin en = 4'b1011; end  // 选中 DK3
            2'd3: begin en = 4'b0111; end  // 选中 DK4（最右侧）
        endcase
    end

    // 七段译码器
    function [7:0] hex_to_seg(input [3:0] hex);
    begin
        case (hex)
            4'd0: hex_to_seg = 8'b00111111;  // 显示 0
            4'd1: hex_to_seg = 8'b00000110;  // 显示 1
            4'd2: hex_to_seg = 8'b01011011;  // 显示 2
            4'd3: hex_to_seg = 8'b01001111;  // 显示 3
            4'd4: hex_to_seg = 8'b01100110;  // 显示 4
            4'd5: hex_to_seg = 8'b01101101;  // 显示 5
            4'd6: hex_to_seg = 8'b01111101;  // 显示 6
            4'd7: hex_to_seg = 8'b00000111;  // 显示 7
            4'd8: hex_to_seg = 8'b01111111;  // 显示 8
```

```
            4'd9: hex_to_seg = 8'b01101111;  // 显示 9
            4'd10: hex_to_seg = 8'b01110111; // 显示 A
            4'd11: hex_to_seg = 8'b01111100; // 显示 B
            4'd12: hex_to_seg = 8'b00111001; // 显示 C
            4'd13: hex_to_seg = 8'b01011110; // 显示 D
            4'd14: hex_to_seg = 8'b01111001; // 显示 E
            4'd15: hex_to_seg = 8'b01110001; // 显示 F
            default: hex_to_seg = 8'b00000000;
        endcase
    end
    endfunction

    // 根据当前扫描位选择显示数据
    always @(*) begin
        case (scan_count)
            2'd0: seg = hex_to_seg(display_data[3:0]);    // DK1 显示最低 4 位
            2'd1: seg = hex_to_seg(display_data[7:4]);    // DK2 显示次低 4 位
            2'd2: seg = hex_to_seg(display_data[11:8]);   // DK3 显示次高 4 位
            2'd3: seg = hex_to_seg(display_data[15:12]);  // DK4 显示最高 4 位
        endcase
    end

endmodule
```

该 FPGA 数码管动态显示系统核心功能如下：以 100MHz 系统时钟为基准，通过两个计数器分别构建 250ms 数值更新周期与 5ms 数码管扫描周期。每 250ms 到达，16 位 display_data 寄存器自动执行十六进制递增，数值在 0000h 至 FFFFh 循环。

系统采用分时复用机制，由 scan_count 计数器控制 4 位数码管轮流选通，每 5ms 切换显示位，结合人眼视觉暂留实现同时显示效果。内置 hex_to_seg 函数，将 4 位二进制数据转换为段选信号，依当前选中数码管位置动态译码。

系统高电平复位有效，复位时初始化 display_data 为 0000h 并点亮最左侧数码管；复位解除后，自动启动计数与扫描逻辑，完成动态显示功能。

2. 管脚约束文件（ego1_pin.xdc）代码及功能描述：与实验一相同。

## 4.2 实验结果

1. 软件仿真结果

仿真结果如下图所示：

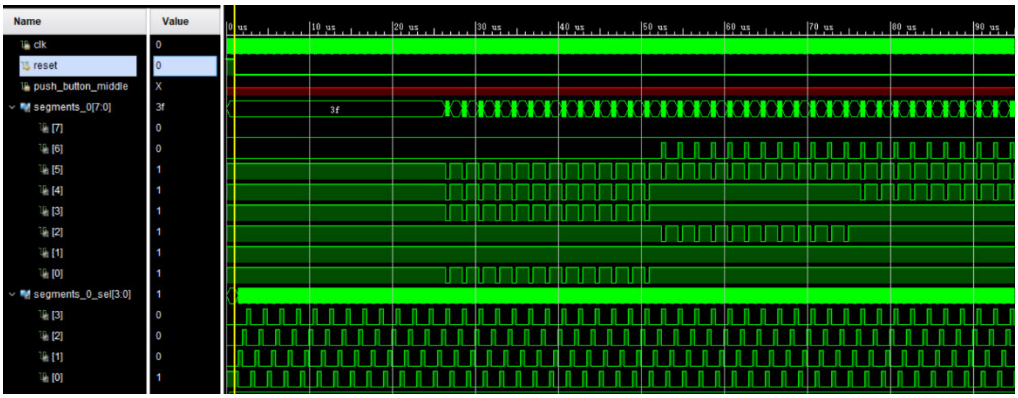


图 3-1：软件仿真结果

通过上图的仿真结果可以看出，我们设计的电路的输出均符合我们的设计要求，因此可将程序生成的 bit 文件并烧录至 EGO-1 开发板进行硬件验证。

2. 下板运行结果

生成 bit 流文件后，将配置文件烧录至 EGO-1 开发板进行物理验证。在部分情况的测试中，电路的测试结果示意图如下所示：

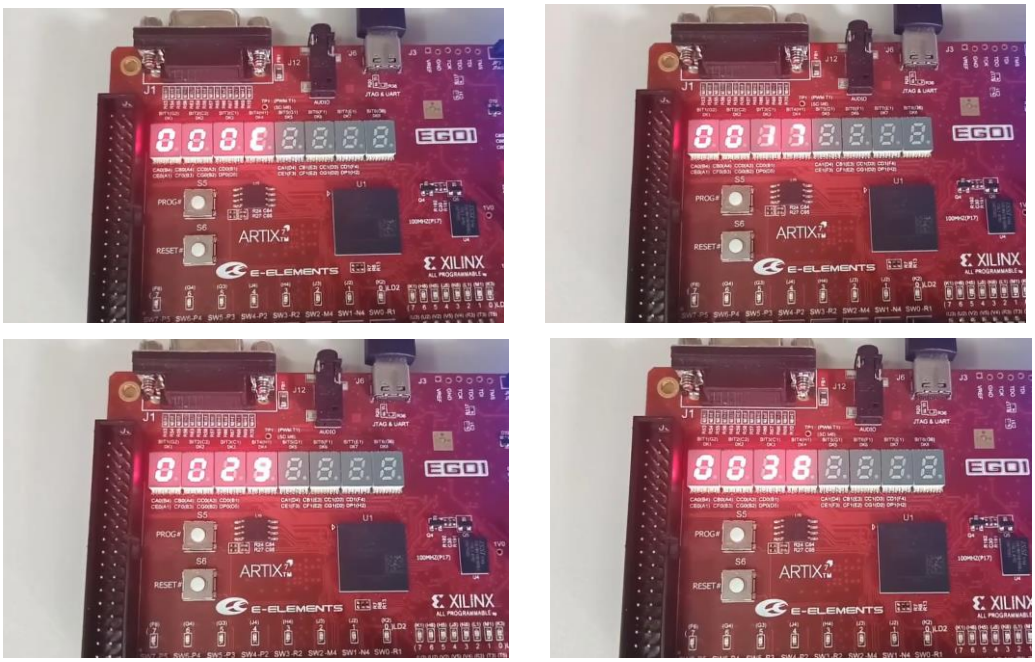

图 3-2：电路测试结果

完整测试流程见以下的视频链接：

https://www.bilibili.com/video/BV1nSjqzaESp/?share_source=copy_web&vd_source=d19749656fe7d6613a2372ba024a967c

在开发板实际运行中，左侧四个数码管（DK1~DK4）展现精准的动态显示效果。复位后，数码管即刻呈现 “0000”，随后以 250ms 为周期，按十六进制规则依次递增，从 “0000” 逐步递变至 “FFFF” 后循环往复。

显示过程采用动态扫描技术，每 5ms 切换一次显示位，依序点亮 DK1 至 DK4，借助视觉暂留原理，达成四位数同时显示的视觉效果。复位瞬间，仅 DK1 点亮显示 “0”，其余保持熄灭；运行全程，数码管显示流畅无闪烁，数值递增逻辑准确无误。

# 五、作业

## 5.1 作业 1

仿真讲义中的代码，并截图，保留 adder 中的 result 信号，如下图所示：

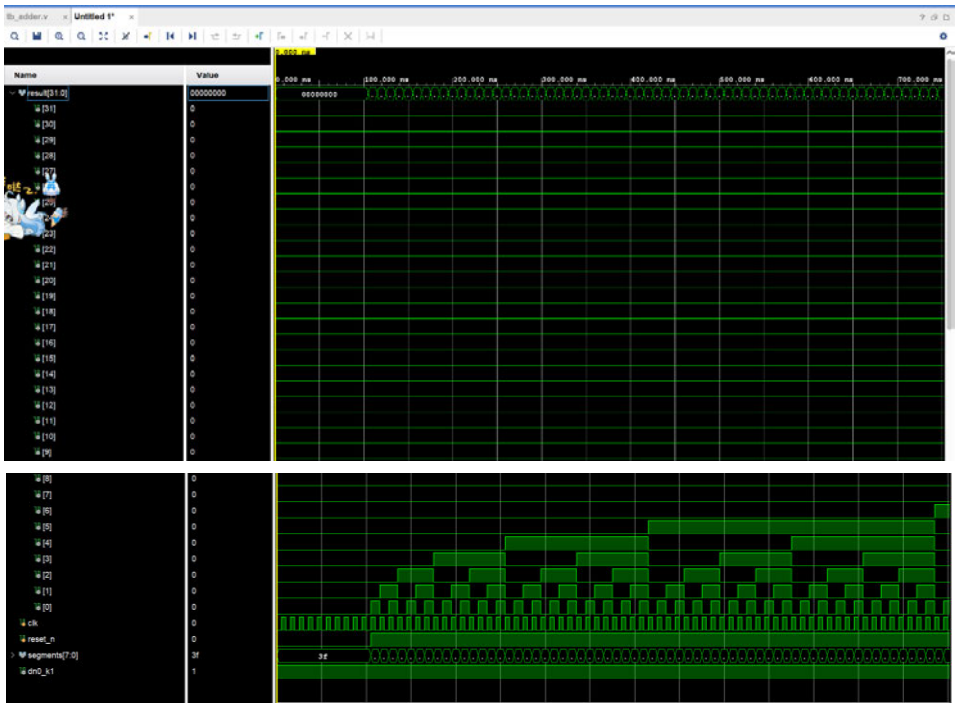

图 3-3：仿真截图

# 六、总结

实验 3 让我对时序逻辑与动态扫描技术有了更加深入的理解。在这次的实验中，我遇到的最大的问题是时钟分频和状态控制。通过设计计数器将 100MHz 时钟精准分频为 250ms 和 5ms 周期，分别驱动数值递增与数码管扫描。在实验中要精确计算时钟周期数，防止数值跳变过快或数码管闪烁现象的出现。

动态扫描技术在实验的成功进行中起到了非常重要的作用：通过每 5ms 快速切换位选信号，利用人眼视觉暂留形成错觉，使得四个数码管看似同时发光。这让我领悟到硬件设计中“时间换空间”的精妙，通过分时复用大幅节省硬件资源。调试时，扫描与递增周期不匹配导致显示错乱，调整计数器参数后才恢复正常，这让我深刻体会到时序参数必须精准把控。

这次实验让我深入理解了时序逻辑电路和 D 触发器的工作机制，也使我完成从组合逻辑到时序逻辑思维上的转变。我亲手实现七段数码管功能也锻炼了我的实践能力。未来实验中，我会继续保持严谨态度，努力探索新的技术领域。

# 实验四 按键消抖

## 一、实验目的

1. 掌握按键消抖的概念。
2. 开始学习独立设计 Verilog 满足用户功能。

## 二、实验原理（简要）

按键消抖就是通过硬件电路或软件算法，消除按键抖动带来的干扰，获取准确的按键动作信号。通过实时监测开发板按键输入信号，通过软件滤波算法消除抖动干扰，由此实现按键消抖。在本实验中只有当按键按下或释放状态持续≥100ms 时，才判定为有效按键动作，避免误触发。实现数码管的按键触发式十六进制递增显示原理与实验三相同。

## 三、实验开发环境及使用器件

1. Vivado
2. Ego-1 开发板
3. Xcom 串口调试工具

## 四、实验内容与主要步骤

### 4.1 基础部分

1. 电路的实现代码（user_code.v）及功能描述

实现该电路的代码为：

```
1. module user_code(
2.     input wire clk,            // 100MHz 系统时钟
3.     input wire reset,          // 复位信号（高有效）
4.     input wire button,         // 按键输入（低有效）
```

```
    output reg [7:0] seg,      // 数码管段选信号
    output reg [3:0] en        // 数码管位选信号（低有效）
);

    // 内部寄存器
    reg [31:0] clk_div;        // 时钟分频计数器
    reg [1:0] scan_count;      // 数码管扫描计数器
    reg [15:0] display_data;   // 显示数据寄存器
    reg button_state;          // 按键状态
    reg [19:0] debounce_cnt;   // 消抖计数器

    // 按键消抖逻辑
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            button_state <= 1'b1;         // 默认按键未按下
            debounce_cnt <= 20'd0;
        end
        else begin
            if (button != button_state) begin
                // 检测到按键状态变化，开始消抖计数
                debounce_cnt <= debounce_cnt + 20'd1;
                if (debounce_cnt >= 20'd999_999) begin  // 10ms @ 100MHz
                    button_state <= button;
                    debounce_cnt <= 20'd0;
                end
            end
            else begin
                debounce_cnt <= 20'd0;  // 状态稳定，清零计数器
            end
        end
    end

    // 按键上升沿检测（按键释放时触发计数）
    reg button_state_prev;
    wire button_pressed;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            button_state_prev <= 1'b1;
        end
        else begin
```

```
            button_state_prev <= button_state;
        end
    end

    assign button_pressed = (~button_state_prev) & button_state;  // 上升沿

    // 数据更新逻辑（仅在按键有效触发时递增）
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            display_data <= 16'd0;
        end
        else begin
            if (button_pressed && display_data < 16'hFFFF) begin
                display_data <= display_data + 16'd1;
            end
        end
    end

    // 数码管扫描控制（同实验 3）
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            clk_div <= 32'd0;
            scan_count <= 2'd0;
        end
        else begin
            if (clk_div >= 32'd499_999) begin  // 5ms @ 100MHz
                clk_div <= 32'd0;
                scan_count <= scan_count + 2'd1;
                if (scan_count >= 2'd3)
                    scan_count <= 2'd0;
            end
            else begin
                clk_div <= clk_div + 32'd1;
            end
        end
    end

    // 位选控制
    always @(*) begin
        case (scan_count)
            2'd0: en = 4'b1110;  // 选中 DK1
```

```
            2'd1: en = 4'b1101;  // 选中 DK2
            2'd2: en = 4'b1011;  // 选中 DK3
            2'd3: en = 4'b0111;  // 选中 DK4
        endcase
    end

    // 七段译码器（同实验 3）
    function [7:0] hex_to_seg(input [3:0] hex);
    begin
        case (hex)
            4'd0: hex_to_seg = 8'b00111111;  // 显示 0
            // ... 其余段码定义同实验 3 ...
            4'd15: hex_to_seg = 8'b01110001; // 显示 F
             default: hex_to_seg = 8'b00000000;
        endcase
    end
    endfunction

    // 段选控制
    always @(*) begin
        case (scan_count)
            2'd0: seg = hex_to_seg(display_data[3:0]);    // DK1 显示最低 4 位
            // ... 其余段选逻辑同实验 3 ...
            2'd3: seg = hex_to_seg(display_data[15:12]);  // DK4 显示最高 4 位
        endcase
    end
endmodule
```

该系统基于按键消抖技术实现数码管的精确控制，主要功能包括：

1. 按键消抖模块：

✧ 采用 20 位计数器对 100MHz 系统时钟进行分频，构建约 10ms 的消抖检测窗口。

✧ 状态机实时监测按键电平变化，仅当检测到稳定 10ms 的上升沿（按键释放）时产生有效触发。

✧ 通过边沿检测机制避免长按导致的连续计数，确保单次按键仅触发一次递增。

2. 数值更新逻辑：

✧ 仅当按键有效触发且当前数值未达到 FFFF 时，更新 16 位显示数据寄存器。

✧ 达到最大值后自动锁存，保持显示 FFFF 状态。

3. 动态扫描显示：

✧ 复用实验 3 的 5ms 周期扫描时序，依次选通 DK1~DK4 数码管

✧ 基于当前扫描位动态选择对应 4 位数据进行译码

✧ 使用相同的七段译码函数将二进制数转换为段选信号

✧ 利用视觉暂留效应实现四位数码管的同步显示

4. 复位控制机制：

✧ 高电平复位时同步清零显示数据和扫描控制逻辑

✧ 复位解除后进入待机状态，等待有效按键触发

2. 管脚约束文件（ego1_pin.xdc）代码及功能描述

与实验一相同。

## 4.2 实验结果

1. 软件仿真结果

仿真结果如下图所示：

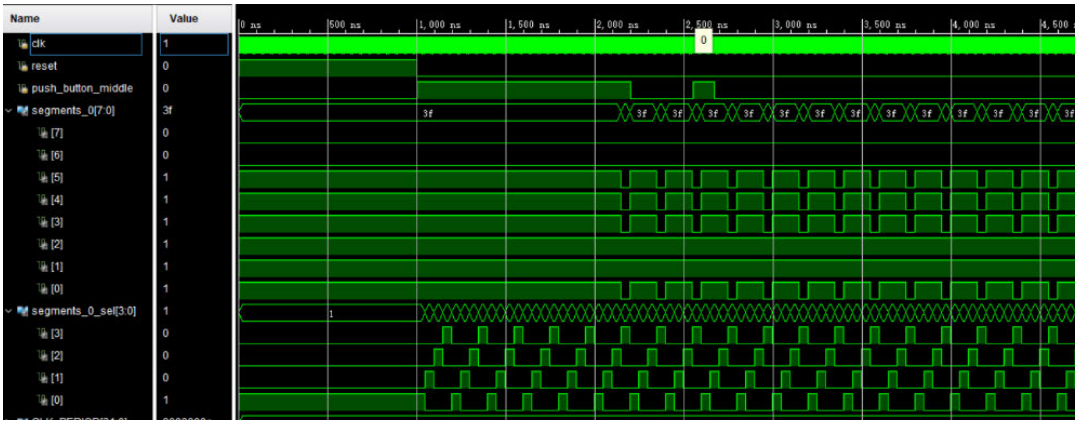


图 4-1：软件仿真结果

通过上图的仿真结果可以看出，我们设计的电路的输出均符合我们的设计

要求，因此可将程序生成的 bit 文件并烧录至 EGO-1 开发板进行硬件验证。

2. 下板运行结果

生成 bit 流文件后，将配置文件烧录至 EGO-1 开发板进行物理验证。在部分情况的测试中，电路的测试结果示意图如下所示：

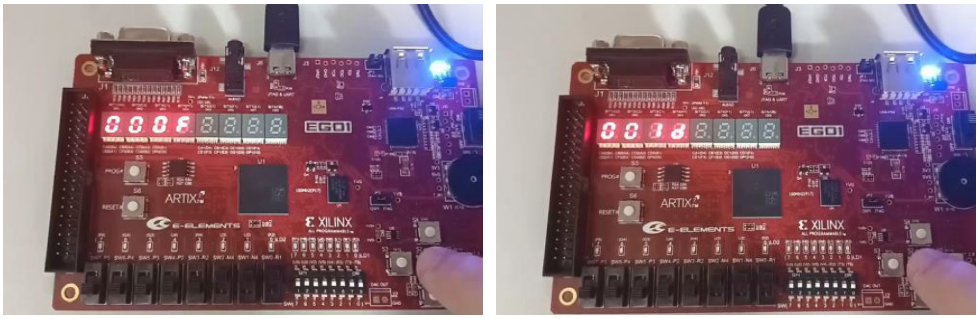

图 4-2：电路测试结果

完整测试流程见以下的视频链接：

**https://www.bilibili.com/video/BV1XijCzvEf2/?share_source=copy_web&vd_source=d19749656fe7d6613a2372ba024a967c**

下板运行时，开发板数码管初始状态为仅最左侧四个数码管点亮并显示“0000”，系统采用 5ms 动态扫描机制切换显示位，确保四位数字稳定无闪烁。当按下按键并保持≥100ms 释放时，数值按十六进制规则递增 1，从 “0000” 逐步递变，直至达到 “FFFF” 后停止计数。面对如快速按下松开等按键抖动情况，系统凭借消抖逻辑精准过滤干扰，仅响应有效长按键动作，最终实现了稳定可靠的按键触发式精确计数功能。

## 五、作业（无）

## 六、总结

在实验 4 我们实现了按键消抖的功能。按键的抖动看似只是一个小问题，但会导致触发器出现错误的边沿触发行为，严重干扰计数逻辑，从而时序逻辑电路无法正常工作。因此设计电路时必须借助软件消抖算法过滤无效的抖动信号。本实验通过计数器实现 10ms 消抖延时，使得电路的状态仅在按键状态稳定后更新。这一过程不仅加深了我对 “时序敏感型电路需严格控制信号边沿”

的理解，也使我认识到设计逻辑电路时要更多地考虑实际情况下可能遇到的各种问题。

在本次实验中我不仅进一步加深了对时序逻辑电路工作原理的理解，也掌握了按键消抖的相关算法。更重要的是我懂得了设计逻辑电路不仅要考虑理论上的需要，也要结合实际中遇到的问题，这对于以后进一步研究很有启发意义。

# 实验五 呼吸灯

## 一、实验目的

1. 了解 IP 生成及调用的过程，会使用 coe 文件给 bram 初始化。
2. 进一步训练独立设计 Verilog 的能力，形成根据时序图完成代码设计的初步思路。

## 二、实验原理（简要）

BRAM 是 FPGA 中集成的专用存储模块，用于高效存储和快速访问数据。BRAM 通常由多个存储单元阵列组成，每个单元可存储 1 位数据，并且按照一定方式地址排列，访问时并不依赖与前一次访问的位置。BRAM 的读写操作通过时钟信号同步，地址和数据在时钟边沿锁存，确保时序稳定。

本实验通过时序逻辑电路和 BRAM 实现呼吸灯效果，具体实现的技术方案如下所示：

1. 亮度曲线生成：采用数学模型$y = e^{\sin(2\pi t)} - e^{-1}$生成呼吸灯亮度曲线，通过调节 LED 占空比实现平滑渐变效果。
2. BRAM 数据存储：将预计算的 512 个亮度采样点存入单端口 BRAM，通过循环地址计数器读取数据，实现亮度值的时序输出。
3. 时钟分频系统：基于 100MHz 系统时钟分频产生 300 周期 / 单位时间的基准信号，每个亮度值对应 300 个时钟周期的占空比调节，形成 4.8ms / 周期的呼吸频率。
4. PWM 控制模块：根据 BRAM 读取的亮度值动态生成 PWM 信号，驱动 LED（LD1 [0]）实现亮度平滑变化，利用人眼视觉暂留效应形成呼吸灯效果。

## 三、实验开发环境及使用器件

1. Vivado
2. Ego-1 开发板
3. Xcom 串口调试工具

## 四、实验内容与主要步骤

### 4.1 基础部分

1. 电路的实现代码（user_code.v）及功能描述

   实现该电路的代码为：

```
module user_code(
    input wire clk,             // 100MHz 系统时钟
    input wire reset,           // 复位信号（高有效）
    output reg led_out          // LED 输出信号
);

    // 内部寄存器和计数器
    reg [8:0] brightness;       // 亮度值 (0-511)
    reg [8:0] addr_counter;     // BRAM 地址计数器
    reg [8:0] pwm_counter;      // PWM 计数器

    // BRAM 实例化 (使用参数化方式定义)
    reg [8:0] bram [0:511];     // 512x9 位 BRAM

    // 初始化 BRAM 数据 (预计算的亮度值)
    initial begin
        integer i;
        real t, y;

        for (i = 0; i < 512; i = i + 1) begin
            t = 2.0 * 3.1415926 * i / 512.0;  // 归一化到 0-2π
            y = $exp($sin(t)) - 0.36787944;    // e^sin(t) - e^(-1)

            bram[i] = $rtoi(y * 511.0);        // 缩放至 0-511 范围
        end
```

```
    end

    // 时钟分频和地址计数器
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            addr_counter <= 9'd0;
            pwm_counter <= 9'd0;
        end
        else begin
            // 300 周期/单位时间的基准信号
            if (pwm_counter >= 9'd299) begin
                pwm_counter <= 9'd0;

                // 每 300 周期更新一次地址，循环读取 BRAM
                if (addr_counter >= 9'd511)
                    addr_counter <= 9'd0;
                else
                    addr_counter <= addr_counter + 9'd1;
            end
            else begin
                pwm_counter <= pwm_counter + 9'd1;
            end
        end
    end

    // 从 BRAM 读取当前亮度值
    always @(posedge clk) begin
        brightness <= bram[addr_counter];
    end

    // PWM 控制逻辑
    always @(*) begin
        if (pwm_counter < brightness)
            led_out = 1'b1;  // LED 点亮
        else
            led_out = 1'b0;  // LED 熄灭
    end

endmodule
```

该代码基于 BRAM 和 PWM 技术实现呼吸灯效果，主要功能包括：

1. 数据存储与曲线生成。

✧ 使用 512×9 位 BRAM 存储预计算的亮度值。

✧ 通过数学公式$y = e^{\sin(2\pi t)} - e^{-1}$生成平滑亮度曲线。

✧ 将连续曲线离散化为 512 个采样点，量化为 0-511 范围的整数值。

2. 时序控制模块

✧ 采用 9 位地址计数器循环访问 BRAM（地址范围 0-511）。

✧ 每 300 个时钟周期更新一次地址，形成约 4.8ms /周期的更新速率。

✧ 实现亮度值在 0-511 范围内的周期性平滑变化。

3. PWM 调光逻辑

✧ 使用 9 位 PWM 计数器与当前亮度值进行比较

✧ 当 PWM 计数器值小于亮度值时，LED 输出高电平（点亮）

✧ 实现占空比从 0%-100%-0% 的周期性变化

4. 呼吸效果实现

✧ 通过指数正弦曲线控制亮度变化，呈现自然 "呼吸" 效果

✧ 亮度变化速率由地址更新频率（300 周期 / 单位时间）决定

✧ 完整呼吸周期为 512×4.8ms≈2.46 秒

5. 复位与初始化

✧ 复位时同步初始化地址计数器和 PWM 计数器

✧ 确保系统从亮度曲线起始点开始工作

该方案通过 BRAM 高效存储预计算数据，结合精确时序控制和 PWM 调光技术，实现了平滑自然的呼吸灯效果。

2. 管脚约束文件（ego1_pin.xdc）代码及功能描述；

与实验一相同。

## 4.2 实验结果

1. 软件仿真结果

仿真结果如下图所示：

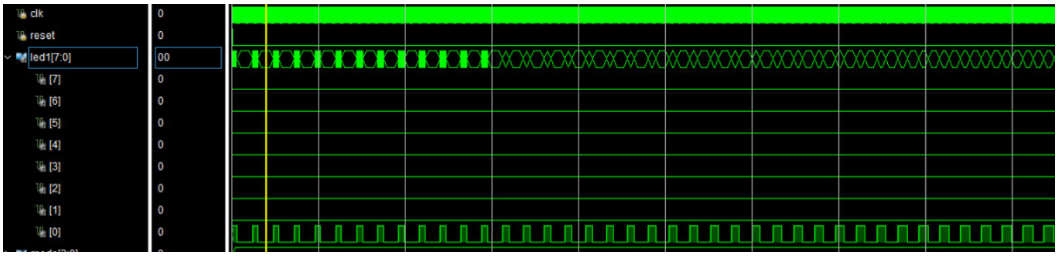


图 5-1：软件仿真结果

通过上图的仿真结果可以看出，我们设计的电路的可以实现呼吸灯的效果，因此可将程序生成的 bit 文件并烧录至 EGO-1 开发板进行硬件验证。

2. 下板运行结果

生成 bit 流文件后，将配置文件烧录至 EGO-1 开发板进行物理验证。在部分情况的测试中，电路的测试结果示意图如下所示：

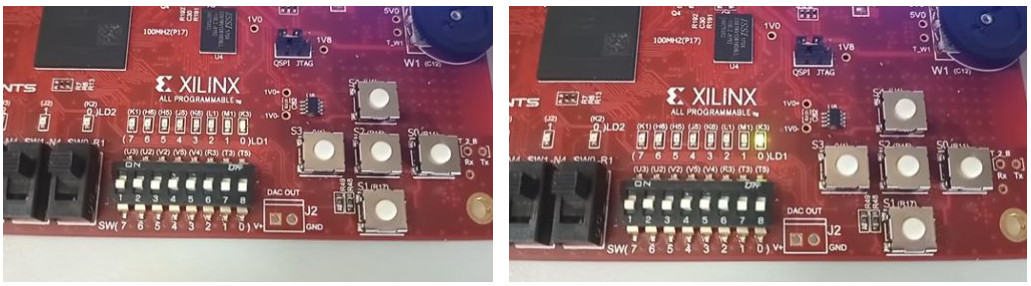

图 5-2：电路测试结果

完整测试流程见以下的视频链接：

**https://www.bilibili.com/video/BV1YijCzvE4j/?share_source=copy_web&vd_source=d19749656fe7d6613a2372ba024a967c**

开发板下板运行后，LED（LD1 [0]）呈现出流畅的呼吸灯效果。系统通电复位后，LED 从熄灭状态开始，依循数学公式$y = e^{\sin(2\pi t)} - e^{-1}$的曲线规律，亮度逐渐变大达到峰值，随后又逐渐变暗，如此循环往复，每个呼吸周期稳定维持在约 2.46 秒。通过 BRAM 预先存储的 512 个亮度采样点，配合 PWM 信号对占空比的精准控制，亮度过渡平滑自然、无任何闪烁，实现了预期的呼吸灯效果。

# 五、作业

## 5.1 作业 1

生成一个深度 512，宽度 32bit 的 Single Port BRAM，截图如下：

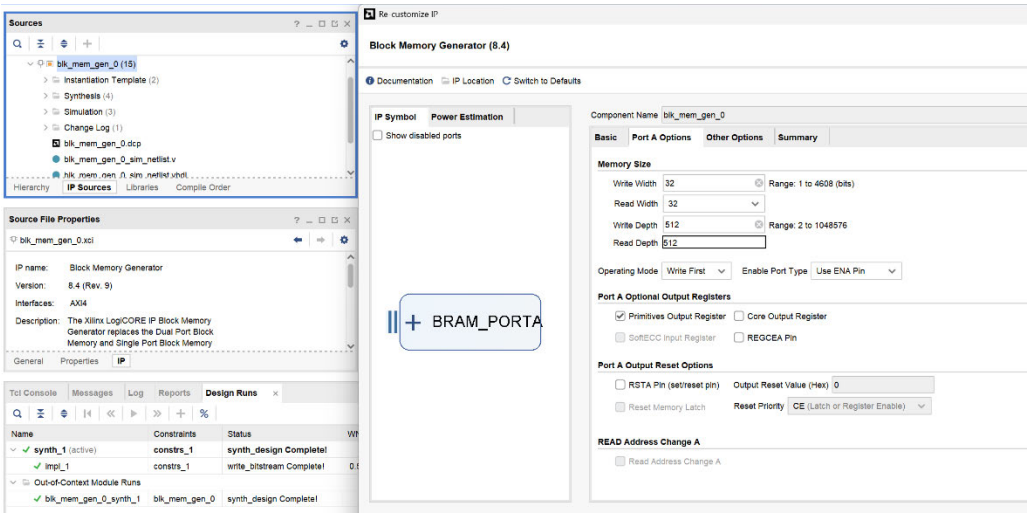


图 5-3：Single Port BRAM 截图

# 六、总结

在本次呼吸灯实验中，我深入探索了 FPGA 技术在波形生成领域的应用，通过将数学公式与硬件逻辑电路相结合，成功实现了 LED 呼吸灯效果。这一过程不仅使我加深了对课堂所学的理论知识的理解，更让我在实践培养了分析、设计并最终实现一个全新的数字电路的能力。

实验核心在于利用 BRAM 和 PWM 技术实现平滑的亮度渐变。我们首先通过数学公式$y = e^{\sin(2\pi t)} - e^{-1}$生成亮度变化曲线，并将其离散化为 512 个采样点存入 BRAM。这种“空间换时间”的设计策略极具创新性，通过预先存储数据避免了实时复杂运算，既保证了曲线精度，又显著降低了逻辑资源消耗，使我深刻认识到硬件设计中资源优化的重要性。

在调试过程中，我遇到了诸多困难与挑战。其中，PWM 控制与时钟分频的协

同配合是本实验的关键所在。刚开始我们因未严格校准 300 周期 / 单位时间的基准信号，导致呼吸节奏紊乱，LED 闪烁异常。通过反复调整计数器阈值才恢复正常。在 BRAM 初始化过程中进行的公式推导与离散化处理工作，让我对数字信号量化与模拟波形复现之间的关系有了更为深入的理解。这一实践过程也使我切实认识到，将理论模型转化为实际硬件实现时，必须进行一系列必要的调整与优化，才能较好地实现预期效果。

# 总结与反思

在这次的数电实验中，我学习了 Verilog 语法与 Vivado 数字电路设计等相关知识。我不仅深化了对数字电路理论知识的理解、学会基于 Verilog 语言的 FPGA 开发全流程，更提升了动手实践能力以及解决复杂问题的意志品格，得到了思维方式与学习能力的全面提升。

在这次的五个实验中，我充分体会到理论与实际相结合的重要意义。以往课本上的理论知识，如数字电路的分析与设计、硬件描述语言规则，在 Vivado 平台上具体展现了出来。从搭建简单的组合逻辑电路，到实现复杂的时序控制功能，每一次将代码转化为实际可运行的硬件效果时，都让我对抽象概念有了具象化的理解。这种理论与实践的深度结合，进一步加深了我对数字电路的认识，也让我明白只有通过动手实践，才能熟练掌握任何一项技术。

与此同时，这次实验培养了我解决问题的系统思维。Vivado 开发流程涵盖代码编写、综合、实现、仿真等多个环节，任何一个步骤出现问题都会影响最终结果。调试过程中，面对出现的各种异常状况，我逐渐学会从全局视角分析问题，利用已有的知识认真分析故障原因。这次的实验经历让我意识到，解决各种复杂问题要结合自己的理论知识，建立清晰的逻辑框架，分步骤排查与验证，这种思维方式对我未来的科研道路极具价值。

最后，实验拓宽了我的技术视野，让我对硬件描述语言和 EDA 工具的强大功能有了深刻认知。通过接触 Vivado 平台强大的功能，如模块化设计、资源优化配置等，我认识到现代数字电路开发早已超越了单纯的代码编写，而是需要结合工具特性与工程思维。这让我对 FPGA 技术的应用前景产生了浓厚兴趣，也激发了我探索更多前沿硬件设计领域的热情。我相信这次实验学习的 Verilog 编程与 Vivado 设计数字电路的方法，将在我未来的科研中发挥重要作用。