

《数字电路实验》实验报告

实验名称: 静态和动态数码管显示实验 指导教师: 高兴宇, 李扬波
姓名: 尹超 学号: 2023K8009926003 专业: 人工智能 班级: 2313 组号: 12
实验日期: 2025.5.7 实验地点: 阶一 2 是否调课/补课: 否 成绩: _____
小组成员: 尹超 张硕 李雨恒

目录

1 实验仪器与用具	2
2 实验内容与步骤	3
2.1 概述	3
2.2 实验步骤	4
3 实验结果	15
3.1 实验结果 1: 静态数码管显示实验基础问题	15
3.2 实验结果 2: 静态数码管显示实验变式 1	15
3.3 实验结果 3: 静态数码管显示实验变式 2	15
3.4 实验结果 4: 动态数码管显示实验基础问题	15
3.5 实验结果 5: 动态数码管显示实验变式	15
4 实验总结	15
4.1 实验中的问题与感想	15
5 附录: Verilog 代码	16

§1 实验仪器与用具

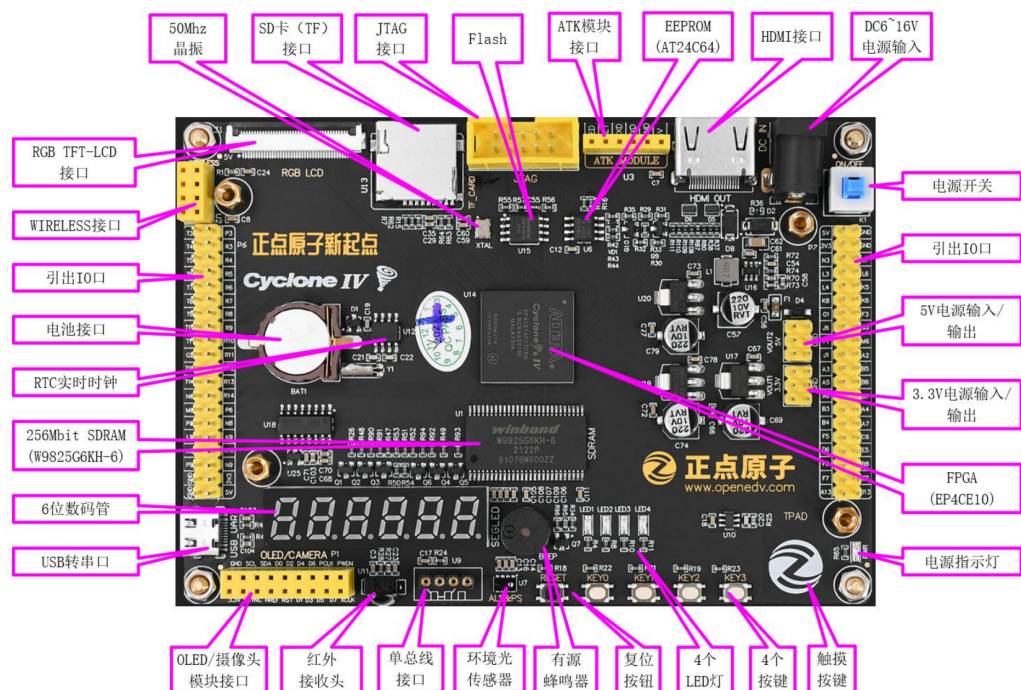


图 1: 实验仪器与用具

新起点 FPGA 开发板板载资源如下:

- 主控芯片: EP4CE10F17C8N, 封装: BGA256
- 晶振: 50 MHz
- FLASH: W25Q16, 容量: 16 Mbit (2 M 字节)
- SDRAM: W9825G6KH-6, 容量: 256 Mbit (32 M 字节)
- EEPROM: AT24C64, 容量: 64 Kbit (8 K 字节)
- 1 个电源指示灯 (蓝色)
- 4 个状态指示灯 (DS0 DS3: 红色)
- 1 个红外接收头, 并配备一款小巧的红外遥控器
- 1 个无线模块接口, 支持 NRF24L01 无线模块
- 1 路单总线接口, 支持 DS18B20/DHT11 等单总线传感器
- 1 个 ATK 模块接口, 支持正点原子蓝牙/GPS/MPU6050/RGB 灯模块
- 1 个环境光传感器, 采用 AP3216C 芯片
- 1 个标准的 RGB TFT-LCD 接口
- 1 个 OLED/摄像头模块接口
- 1 个 USB 串口
- 1 个有源蜂鸣器
- 1 个 SD 卡接口 (在板子背面)
- 1 个 HDMI 接口
- 1 个标准的 JTAG 调试下载口
- 1 组 5V 电源供应/接入口
- 1 组 3.3V 电源供应/接入口
- 1 个直流电源输入接口 (输入电压范围: DC6 16V)

- 1 个 RTC 后备电池座，并带电池（在板子背面）
- 1 个 RTC 实时时钟，采用 PCF8563 芯片
- 1 个复位按钮，可作为 FPGA 程序执行的复位信号
- 4 个功能按钮
- 1 个电容触摸按键
- 1 个电源开关，控制整个开发板的电源
- 两个 20×2 扩展口，共 72 个扩展 IO 口（除去电源和地）

§ 2 实验内容与步骤

2.1 概述

静态数码管显示实验

A. 按照指南开发指南第十三章（P239）复现静态数码管显示实验，仿真部分暂时不用做

B. 修改原实验的按键逻辑，使其分别实现下述功能：

- (1) 将数字切换的时间间隔从 0.5s 改为 0.25s，并固化程序到开发板中，实现断开下载器、电源后重启仍可实现 LED 流水灯（可通过修改 MAX_NUM 实现）
- (2) 实现按键 1 控制时间间隔为 0.5s，按键 2 控制时间间隔为 0.25s（修改代码后还需要加入输入并分配引脚）

动态数码管显示实验

A. 按照指南开发指南第十四章（P249）复现动态数码管显示实验，仿真部分暂时不用做

B. 修改原实验的按键逻辑，使其实现以下功能：

- (1) Key0 可以暂停计数，key1 可以改变计数加减模式（到 0 时不再减小），key2 可以清零计数（主要修改 count 部分加减逻辑即可，需传入新输入并分配引脚）

2.2 实验步骤

(1) 新建 Project

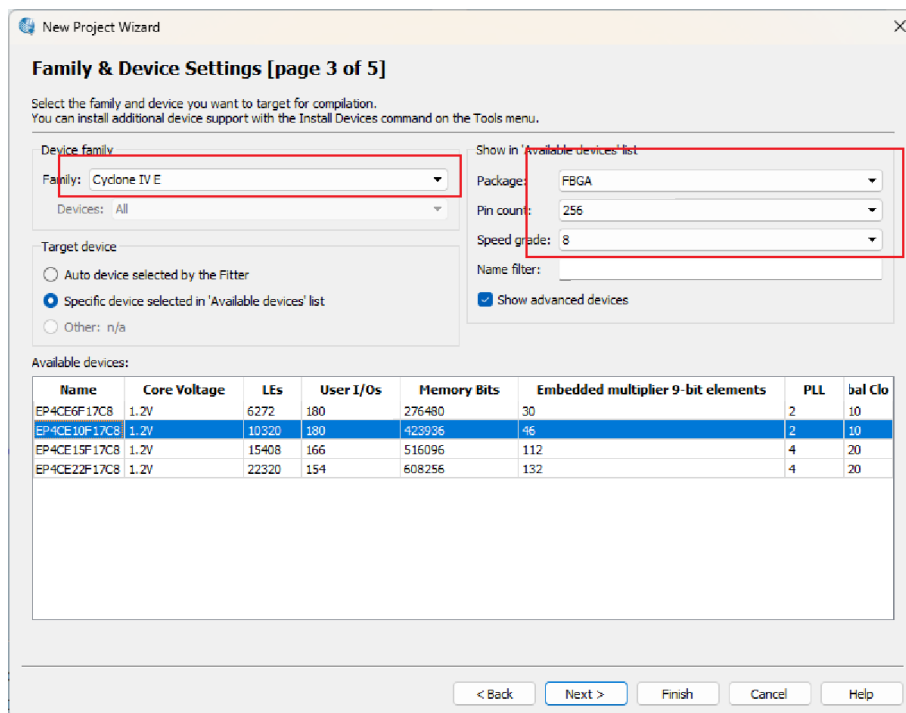


图 2: 新建 Project 的操作界面 1

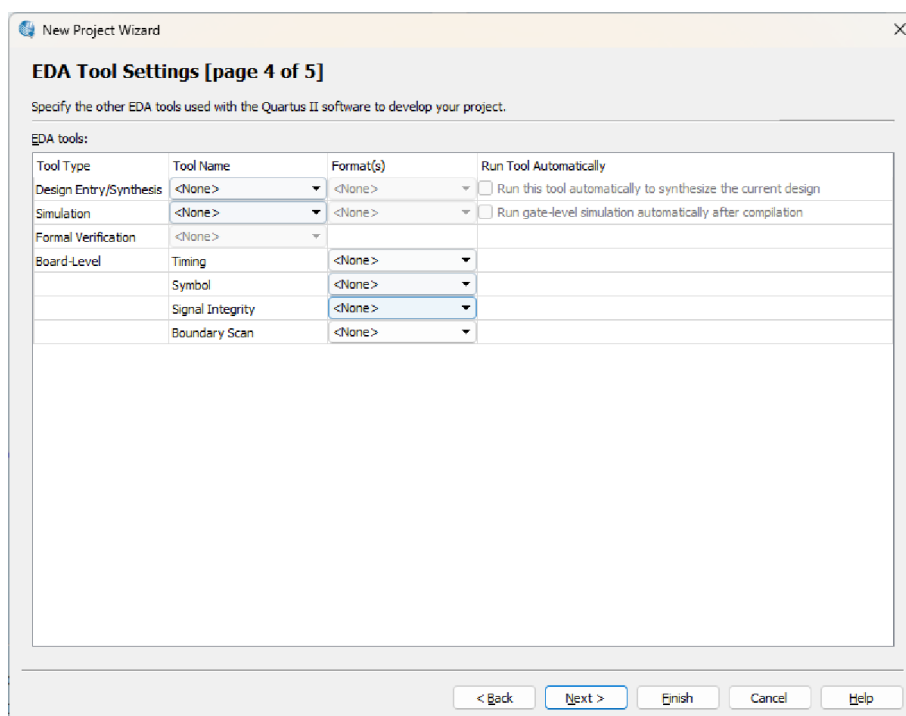
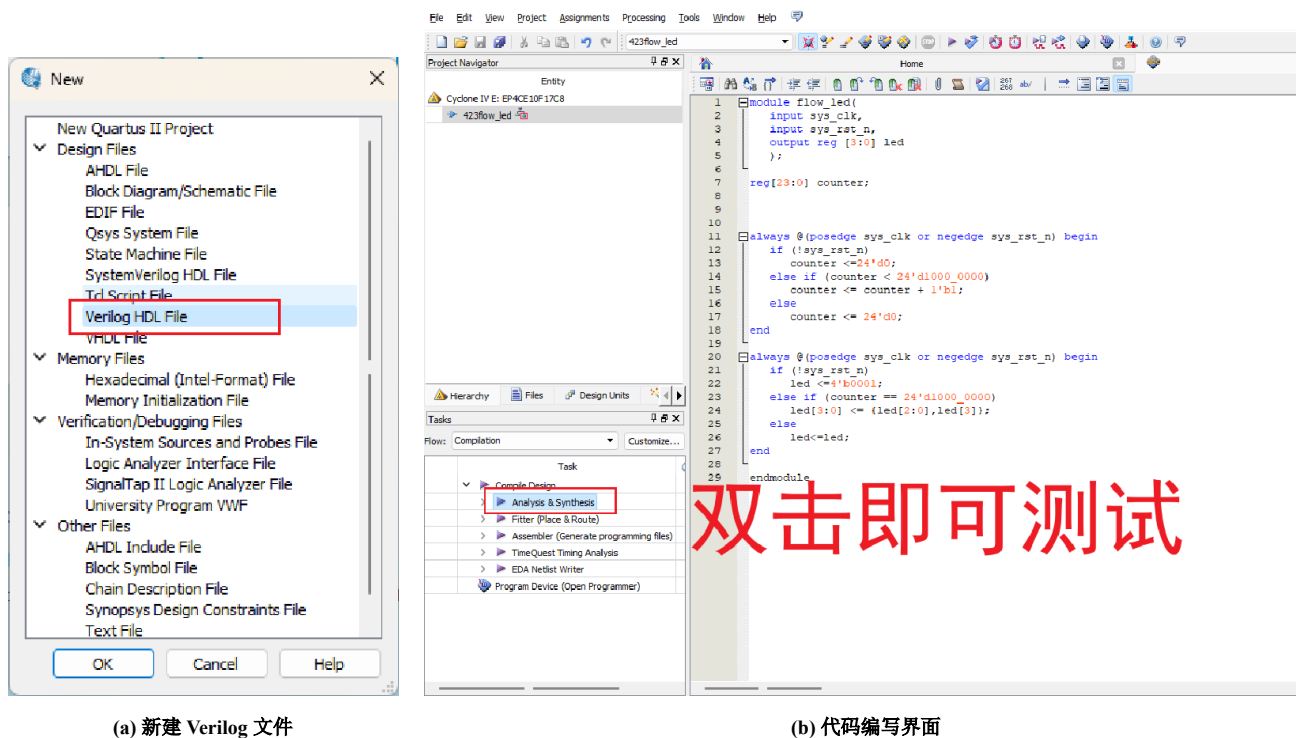


图 3: 新建 Project 的操作界面 2

(2) 新建.v 文件，编写代码并测试编译



说明:

- 在 Quartus 工程中新建.v 文件后，双击打开文件，按照实验要求输入 Verilog 代码。
- 编写完成后，按 **Ctrl+S** 保存代码文件，确保文件已保存到 **rtl** 文件夹下，便于后续管理和调用。
- 保存后，右键点击该文件，选择“Add File to Project”，将代码文件添加到工程中。
- 在主界面点击“Analysis and Synthesis”或“Start Compilation”按钮，进行代码的语法检查和综合编译。
- 编译过程中如有语法错误，软件会在 Messages 窗口提示具体报错信息。根据提示修改代码，直至编译通过。
- 建议每次修改代码后都及时保存并重新编译，确保每一步更改都能被正确识别和验证。

(3) 预编译, 确认无误

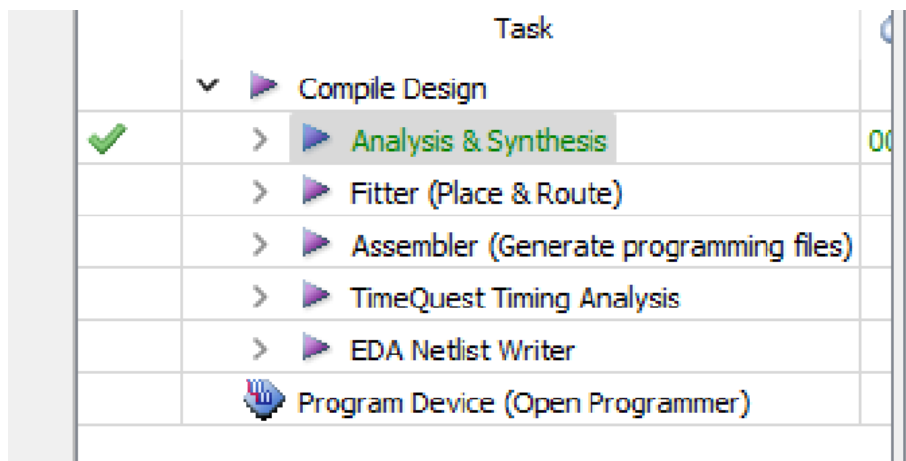


图 5: 预编译结果界面 1

```

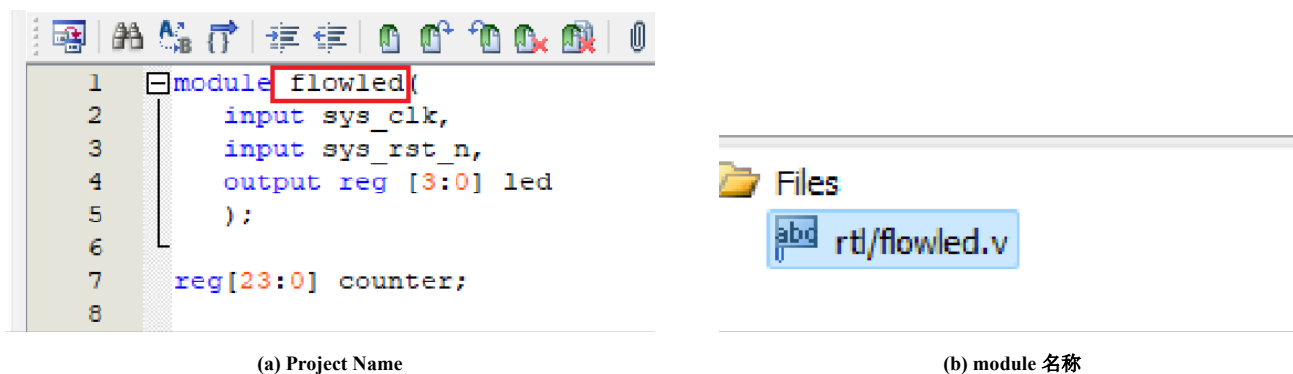
> 286030 Timing-Driven Synthesis is running
> 16010 Generating hard_block partition "hard_block:auto_generated_inst"
> 21057 Implemented 48 device resources after synthesis - the final resource count might be different
> Quartus II 64-Bit Analysis & Synthesis was successful. 0 errors, 0 warnings

```

图 6: 预编译结果界面 2

说明: 预编译通过后, Quartus 的 Messages 窗口不会显示红色的 Error 报错信息, 且 Analysis and Synthesis、Fitter 等阶段均能顺利完成。常见问题包括: 代码语法错误、端口未连接、模块名或端口名拼写不一致、文件未正确添加到工程、顶层模块设置错误等。遇到报错时, 可双击报错信息定位到具体代码行, 根据提示逐一修改, 直至所有错误消除。建议每次修改后都重新预编译, 确保问题及时发现和解决。

(4) 注意 Project Name 和代码中的 module 名称一致



说明: Project Name (工程名) 和代码中的 module 名称保持一致, 可以避免综合和仿真时出现顶层模块无法识别的问题。检查方法为: 在新建工程时设置的 Project Name 应与顶层 Verilog 文件中的 module 名称完全相同 (包括大小写), 可通过工程属性和代码首行 module 语句进行核对。如不一致, 建议修改工程名或 module 名称保持统一。

(5) 保存代码文件至 rtl 文件夹内

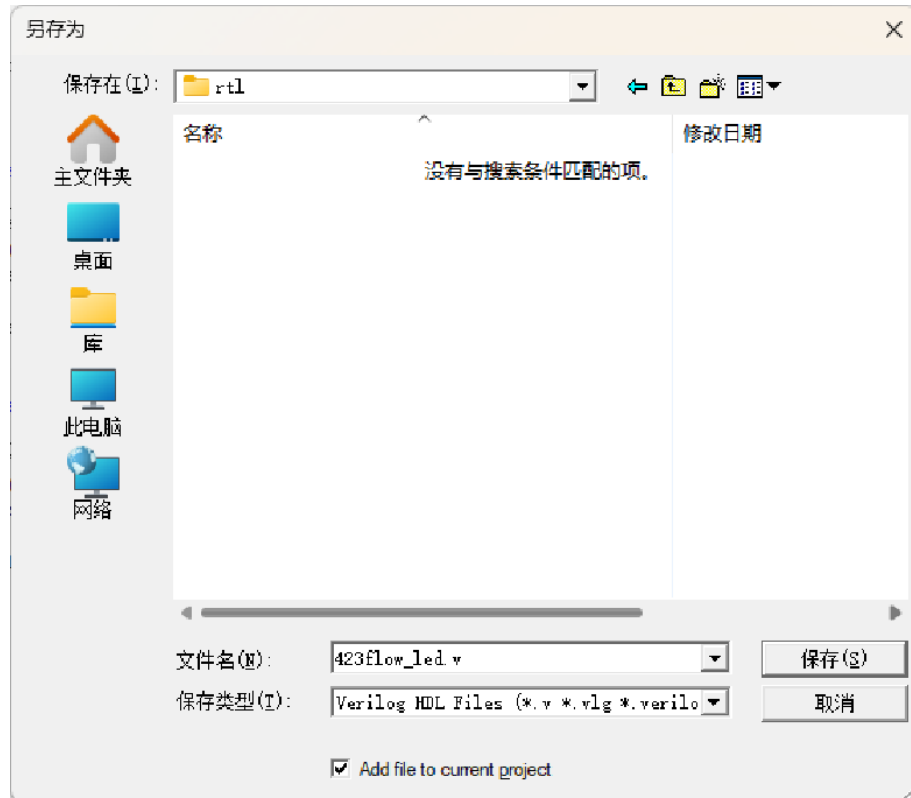


图 8: 保存代码文件到指定文件夹

说明: 建议所有 Verilog 源代码文件统一保存在工程目录下的 **rtl** 文件夹内, 便于管理和后续调用。图片文件建议存放于 **img** 文件夹, TCL 脚本等辅助文件可放在 **script** 文件夹。文件命名应简洁明了, 避免使用中文、空格或特殊字符, 推荐使用小写字母、数字和下划线组合。例如: **seg_led_static.v**、**step1.png**。这样有助于工程的规范化管理和后期维护。

(6) 打开 Pin Planner 分配引脚

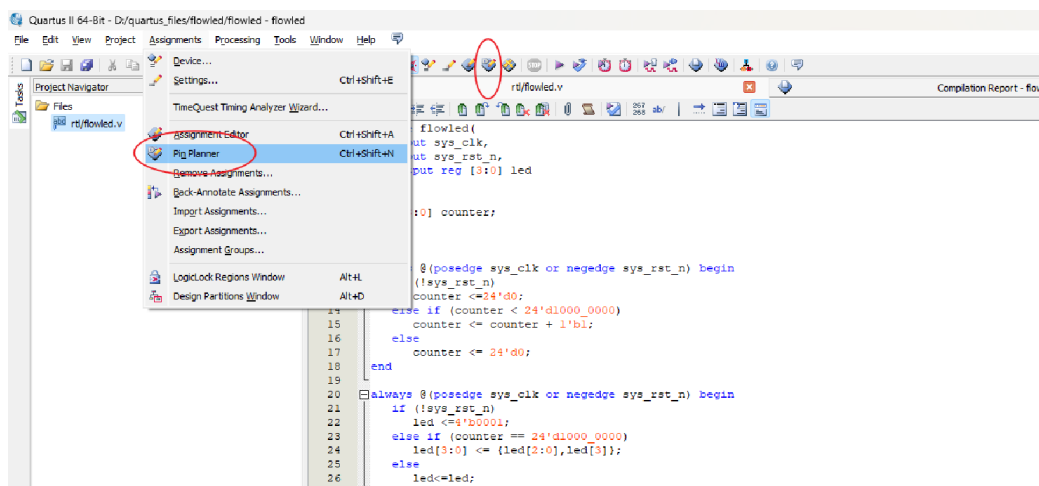


图 9: Pin Planner 分配引脚界面

说明: 在 Quartus 中分配引脚时, 首先点击工具栏的“Assignments”菜单, 选择“Pin Planner”进入引脚分配界面。根据原理图或实验指导书, 将各个信号(如时钟、复位、数码管段选、位选、按键等)对应到实际开发板的物理引脚编号。在 Pin Planner 界面左侧找到信号名称, 右侧输入或选择对应的引脚编号。分配完成后, 点击保存并关闭 Pin Planner。建议分配完引脚后进行一次完整编译, 确保无引脚冲突或未分配等错误。如有 tcl 脚本文件, 也可通过“Assignments”→“Import Assignments”导入脚本自动分配引脚。

(7) 手动输入/使用 tcl 脚本文件

信号名	方向	管脚	端口说明
sys_clk	input	M2	系统时钟, 50M
sys_rst_n	input	M1	系统复位, 低有效
led[0]	output	D11	LED0
led[1]	output	C11	LED1
led[2]	output	E10	LED2
led[3]	output	F9	LED3

图 10: tcl 脚本分配引脚示意

tcl 脚本示例:

```
1      set_location_assignment PIN_M2 -to sys_clk
2      set_location_assignment PIN_M1 -to sys_rst_n
3      set_location_assignment PIN_D11 -to led[0]
4      set_location_assignment PIN_C11 -to led[1]
5      set_location_assignment PIN_E10 -to led[2]
6      set_location_assignment PIN_F9 -to led[3]
7
```

说明: 手动分配引脚适合引脚数量较少或仅需少量调整的情况, 操作直观, 可直接在 Pin Planner 图形界面中完成, 便于初学者理解和检查。使用 tcl 脚本分配引脚则适合引脚较多或需多次复用、批量管理的场景, 只需准备好脚本文件即可一键导入, 效率更高, 也便于团队协作和版本管理。实际操作时, 建议在项目初期用 Pin Planner 熟悉引脚分配流程, 后续可通过 tcl 脚本实现自动化和规范化管理。

(8) 完整编译工程

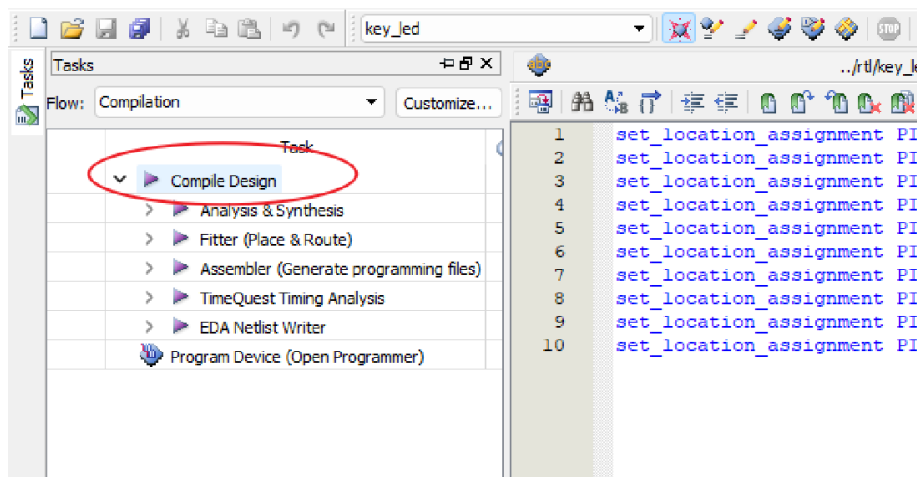


图 11: 完整编译工程界面

说明: 完整编译流程包括: 首先点击主界面的“Start Compilation”按钮, Quartus 会自动依次执行 Analysis & Synthesis、Fitter、Assembler、TimeQuest Timing Analysis 等步骤。编译过程中如遇到错误 (Error) 或警告 (Warning), 可在 Messages 窗口查看详细信息, 双击可定位到相关代码或设置。常见注意事项包括: 确保所有源文件已正确添加到工程、顶层模块设置无误、引脚分配无冲突、时钟约束合理等。建议每次修改代码或引脚分配后都重新完整编译, 确保设计的正确性和可用性。

(9) 下载程序至开发板

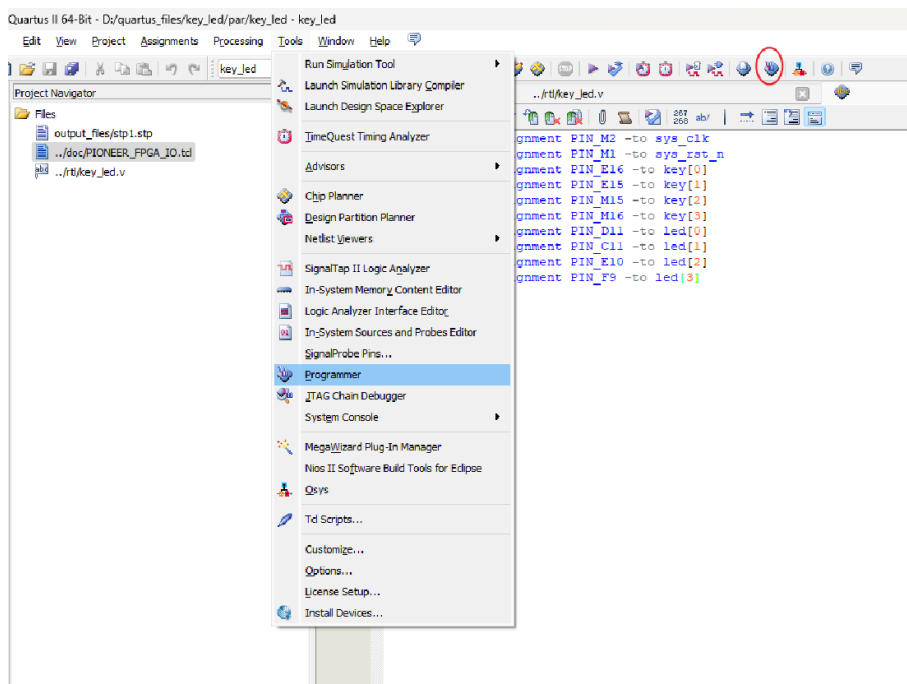


图 12: 下载程序到开发板界面 1

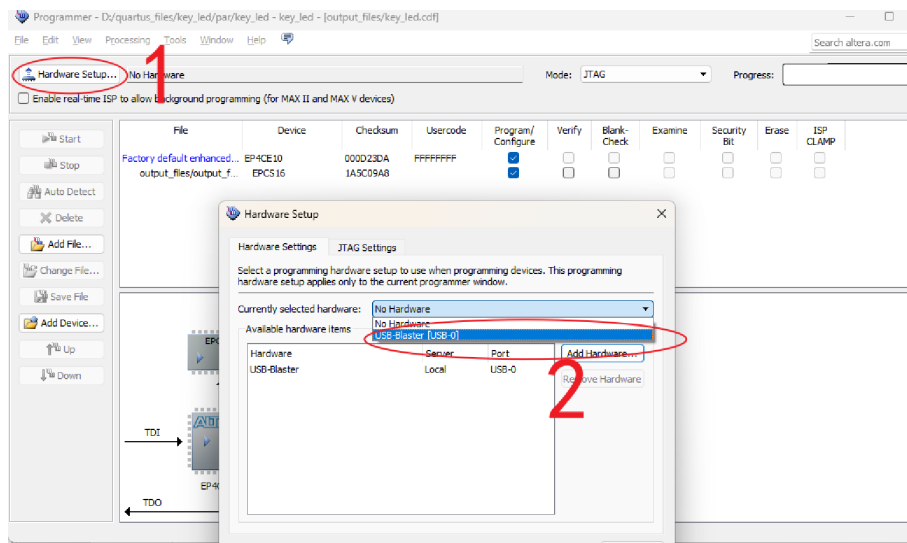


图 13: 下载程序到开发板界面 2

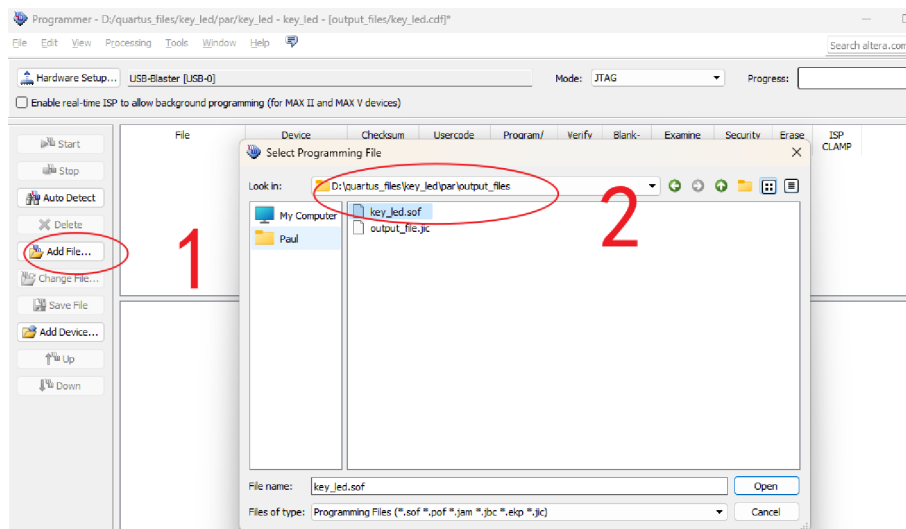


图 14: 下载程序到开发板界面 3

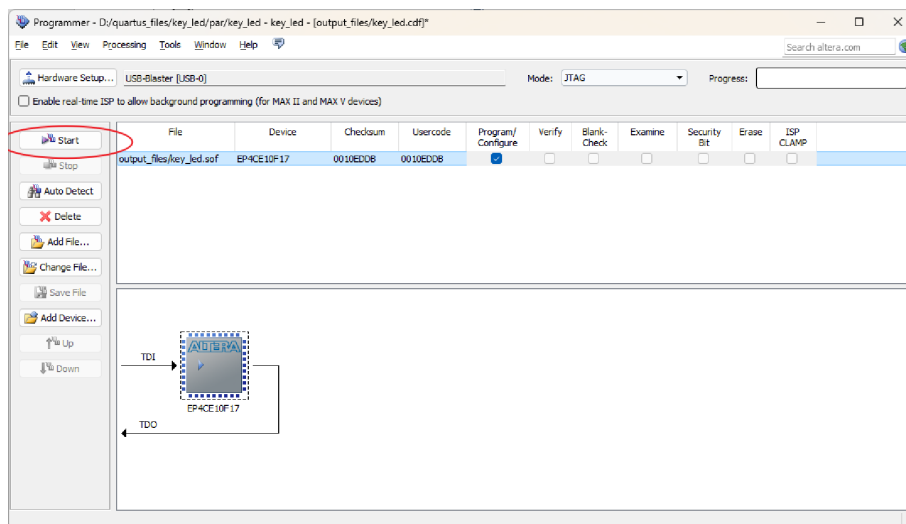


图 15: 下载程序到开发板界面 4

说明：下载程序到开发板的步骤如下：首先确保开发板已正确连接至电脑，并安装好驱动程序。打开 Quartus，点击工具栏的“Programmer”按钮，进入下载界面。点击“Hardware Setup”选择正确的下载器型号（如 USB-Blaster），然后点击“Auto Detect”自动识别芯片。添加编译生成的.sof 文件，勾选“Program/Configure”，最后点击“Start”按钮开始下载。下载完成后，观察开发板上的数码管显示是否正常。常见问题包括：未识别到下载器、芯片型号选择错误、下载过程中断等，遇到问题可尝试重新连接设备、检查驱动或重启软件。

(10) 导出 jic 文件, 固化程序

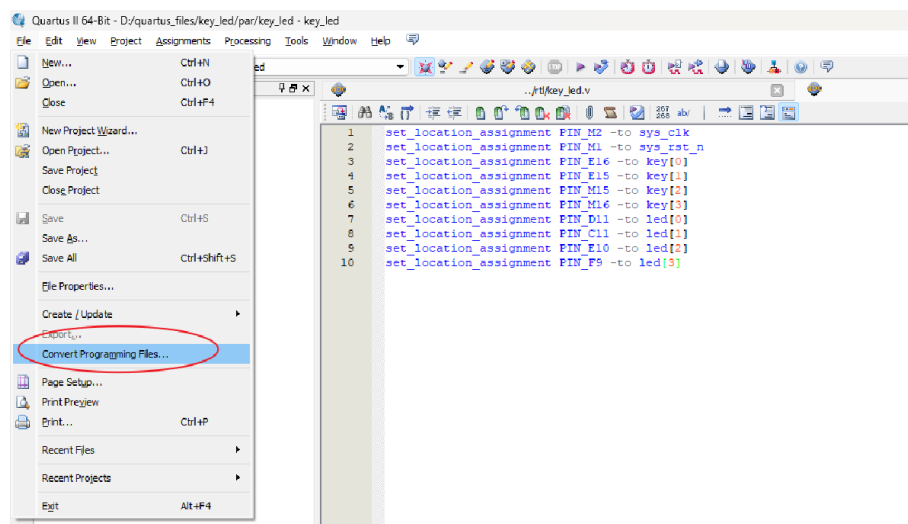


图 16: 导出 jic 文件界面

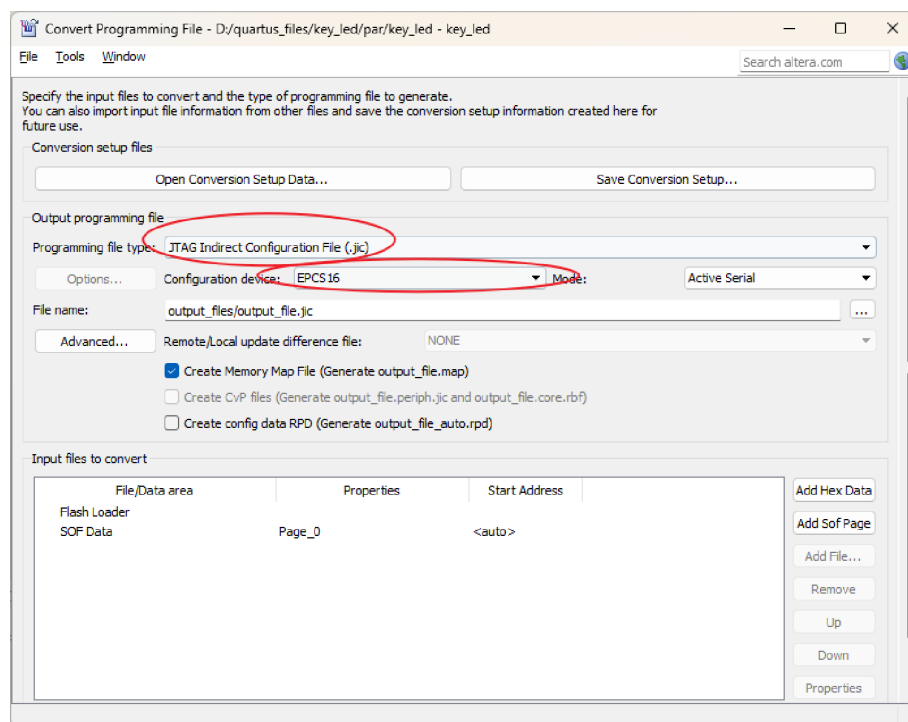


图 17: 导出 jic 文件界面 2

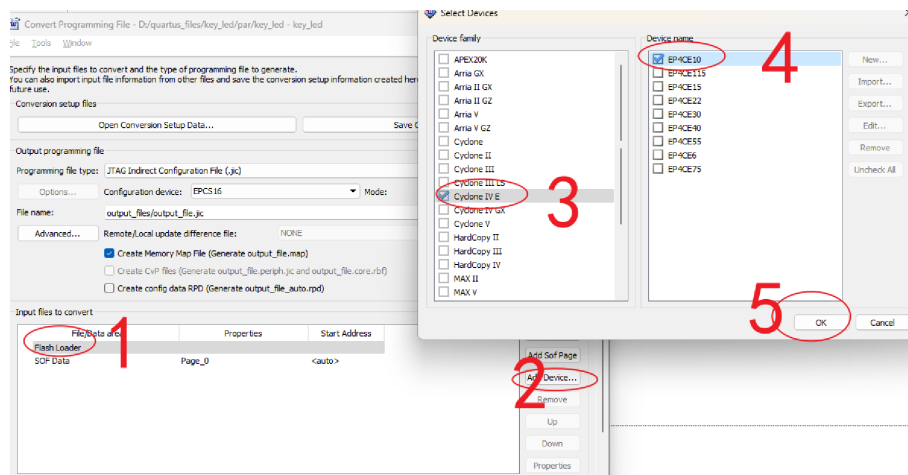


图 18: 导出 jic 文件界面 3

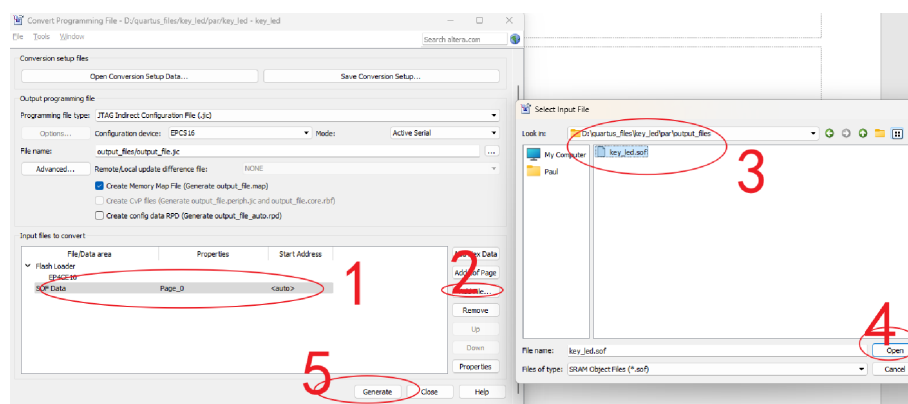


图 19: 导出 jic 文件界面 4

说明：固化流程如下：首先在 Quartus 中点击“File”菜单，选择“Convert Programming Files”，在弹出的窗口中将 File Type 选择为 JTAG Indirect Configuration File (.jic)，并选择对应的目标器件和 Flash 类型。添加已编译生成的.sof 文件，设置输出文件名和路径，点击“Generate”生成.jic 文件。生成后，打开“Programmer”界面，将 Mode 设置为“Active Serial Programming”，添加.jic 文件，选择目标 Flash 芯片，勾选“Program/Configure”，点击“Start”开始固化。固化完成后，断电重启开发板，程序即可自动运行。注意事项：固化前确保芯片型号和 Flash 类型选择正确，避免误操作导致固化失败；固化过程中请勿断开电源或数据线；如遇固化失败，可尝试重新生成.jic 文件或检查硬件连接。

§3 实验结果

3.1 实验结果 1: 静态数码管显示实验基础问题

演示视频: [点击查看演示视频](#)

3.2 实验结果 2: 静态数码管显示实验变式 1

演示视频: [点击查看演示视频](#)

3.3 实验结果 3: 静态数码管显示实验变式 2

演示视频: [点击查看演示视频](#)

3.4 实验结果 4: 动态数码管显示实验基础问题

演示视频: [点击查看演示视频](#)

3.5 实验结果 5: 动态数码管显示实验变式

演示视频: [点击查看演示视频](#)

§4 实验总结

4.1 实验中的问题与感想

- (1) 在这次实验过程中, 由于助教老师已经给出了 TCL 脚本文件, 所以我们只需要在代码中添加引脚分配即可, 节省了不少时间。
- (2) 但不得不提的是, 之前我们实验中使用的手动分配引脚, 现在采取导入 TCL 文件, 对于我们来说是一个新的挑战。因为我们需要在代码中添加引脚分配, 而不是在图形化界面中进行操作。
- (3) 这次实验中, 我们还需要对按键的逻辑进行修改, 这也是一个挑战。我们需要仔细阅读代码, 理解每一行的含义, 并进行相应的修改。
- (4) 这次还有一个问题, 在给出的代码和引脚 TCL 文件中出现了冲突, 原因是 sel 和 seg_sel 不同的命名方式导致了运行冲突。
- (5) 另外, 实验中还需要注意的是, 按键的逻辑需要根据实际情况进行修改, 比如按键的时间间隔、计数加减模式等, 这些都需要我们仔细考虑。

§5 附录: Verilog 代码

Verilog 代码: 静态数码管显示实验基础问题

```
1  module seg_led_static_top (
2      input          sys_clk  ,      // 系统时钟
3      input          sys_rst_n,      // 系统复位信号 (低有效)
4
5      output [5:0]    seg_sel  ,      // 数码管位选
6      output [7:0]    seg_led   // 数码管段选
7
8  );
9
10 //parameter define
11 parameter TIME_SHOW = 25'd25000_000; // 数码管变化的时间间隔0.5s
12
13 //wire define
14 wire      add_flag;                // 数码管变化的通知信号
15
16 //*****
17 /**                                main code
18 //*****
19
20 //每隔0.5s产生一个时钟周期的脉冲信号
21 time_count #(
22     .MAX_NUM    (TIME_SHOW)
23 ) u_time_count(
24     .clk        (sys_clk  ),
25     .rst_n      (sys_rst_n),
26
27     .flag       (add_flag )
28 );
29
30 //每当脉冲信号到达时, 使数码管显示的数值加1
31 seg_led_static u_seg_led_static (
32     .clk        (sys_clk  ),
33     .rst_n      (sys_rst_n),
34
35     .add_flag   (add_flag ),
36     .seg_sel    (seg_sel  ),
37     .seg_led    (seg_led  )
38 );
39
40 endmodule
41
42
43 module time_count(
44     input      clk      ,      // 时钟信号
45     input      rst_n    ,      // 复位信号
46
47     output reg  flag     // 一个时钟周期的脉冲信号
48 );
```



```
49
50 //parameter define
51 parameter MAX_NUM = 25000_000; // 计数器最大计数值
52
53 //reg define
54 reg [24:0] cnt; // 时钟分频计数器
55
56 //*****
57 /** main code
58 //*****
59
60 //计数器对时钟计数, 每计时到0.5s, 输出一个时钟周期的脉冲信号
61 always @ (posedge clk or negedge rst_n) begin
62     if (!rst_n) begin
63         flag <= 1'b0;
64         cnt <= 24'b0;
65     end
66     else if(cnt < MAX_NUM - 1'b1) begin
67         cnt <= cnt +1'b1;
68         flag <= 1'b0;
69     end
70     else begin
71         cnt <= 24'b0;
72         flag <= 1'b1;
73     end
74 end
75
76 endmodule
77
78
79 module seg_led_static (
80     input clk, // 时钟信号
81     input rst_n, // 复位信号 (低有效)
82
83     input add_flag, // 数码管变化的通知信号
84     output reg [5:0] seg_sel, // 数码管位选
85     output reg [7:0] seg_led // 数码管段选
86 );
87
88 //reg define
89 reg [3:0] num; // 数码管显示的十六进制数
90
91 //*****
92 /** main code
93 //*****
94
95 //控制数码管位选信号 (低电平有效), 选中所有的数码管
96 always @ (posedge clk or negedge rst_n) begin
97     if (!rst_n)
98         seg_sel <= 6'b111111;
99     else
100         seg_sel <= 6'b000000;
```

```
101 end
102
103 //每次通知信号到达时, 数码管显示的十六进制数值加1
104 always @ (posedge clk or negedge rst_n) begin
105     if (!rst_n)
106         num <= 4'h0;
107     else if(add_flag) begin
108         if (num < 4'hf)
109             num <= num + 1'b1;
110         else
111             num <= 4'h0;
112     end
113     else
114         num <= num;
115 end
116
117 //根据数码管显示的数值, 控制段选信号
118 always @ (posedge clk or negedge rst_n) begin
119     if (!rst_n)
120         seg_led <= 8'b0;
121     else begin
122         case (num)
123             4'h0 :    seg_led <= 8'b1100_0000;
124             4'h1 :    seg_led <= 8'b1111_1001;
125             4'h2 :    seg_led <= 8'b1010_0100;
126             4'h3 :    seg_led <= 8'b1011_0000;
127             4'h4 :    seg_led <= 8'b1001_1001;
128             4'h5 :    seg_led <= 8'b1001_0010;
129             4'h6 :    seg_led <= 8'b1000_0010;
130             4'h7 :    seg_led <= 8'b1111_1000;
131             4'h8 :    seg_led <= 8'b1000_0000;
132             4'h9 :    seg_led <= 8'b1001_0000;
133             4'ha :    seg_led <= 8'b1000_1000;
134             4'hb :    seg_led <= 8'b1000_0011;
135             4'hc :    seg_led <= 8'b1100_0110;
136             4'hd :    seg_led <= 8'b1010_0001;
137             4'he :    seg_led <= 8'b1000_0110;
138             4'hf :    seg_led <= 8'b1000_1110;
139             default : seg_led <= 8'b1100_0000;
140         endcase
141     end
142 end
143
144 endmodule
```

Verilog 代码: 静态数码管显示实验变式 1

```
1  module seg_led_static_top_acc (
2      input          sys_clk  ,      // 系统时钟
3      input          sys_rst_n,      // 系统复位信号 (低有效)
4
5      output [5:0]    seg_sel  ,      // 数码管位选
6      output [7:0]    seg_led   // 数码管段选
7
8  );
9
10 //parameter define
11 parameter TIME_SHOW = 25'd12500_000; // 数码管变化的时间间隔0.5s
12
13 //wire define
14 wire      add_flag;                // 数码管变化的通知信号
15
16 //*****
17 /**                                main code
18 //*****
19
20 //每隔0.5s产生一个时钟周期的脉冲信号
21 time_count #(
22     .MAX_NUM    (TIME_SHOW)
23 ) u_time_count(
24     .clk        (sys_clk  ),
25     .rst_n      (sys_rst_n),
26
27     .flag       (add_flag )
28 );
29
30 //每当脉冲信号到达时, 使数码管显示的数值加1
31 seg_led_static u_seg_led_static (
32     .clk        (sys_clk  ),
33     .rst_n      (sys_rst_n),
34
35     .add_flag   (add_flag ),
36     .seg_sel    (seg_sel  ),
37     .seg_led    (seg_led  )
38 );
39
40 endmodule
41
42
43 module time_count(
44     input      clk      ,      // 时钟信号
45     input      rst_n    ,      // 复位信号
46
47     output reg  flag     // 一个时钟周期的脉冲信号
48 );
49
50 //parameter define
51 parameter MAX_NUM = 12500_000; // 计数器最大计数值
```

```
52
53 //reg define
54 reg [24:0] cnt; // 时钟分频计数器
55
56 //*****
57 /** main code
58 //*****
59
60 //计数器对时钟计数, 每计时到0.5s, 输出一个时钟周期的脉冲信号
61 always @ (posedge clk or negedge rst_n) begin
62     if (!rst_n) begin
63         flag <= 1'b0;
64         cnt <= 24'b0;
65     end
66     else if(cnt < MAX_NUM - 1'b1) begin
67         cnt <= cnt +1'b1;
68         flag <= 1'b0;
69     end
70     else begin
71         cnt <= 24'b0;
72         flag <= 1'b1;
73     end
74 end
75
76 endmodule
77
78
79 module seg_led_static (
80     input          clk      , // 时钟信号
81     input          rst_n    , // 复位信号 (低有效)
82
83     input          add_flag, // 数码管变化的通知信号
84     output reg [5:0] seg_sel , // 数码管位选
85     output reg [7:0] seg_led  // 数码管段选
86 );
87
88 //reg define
89 reg [3:0] num; // 数码管显示的十六进制数
90
91 //*****
92 /** main code
93 //*****
94
95 //控制数码管位选信号 (低电平有效), 选中所有的数码管
96 always @ (posedge clk or negedge rst_n) begin
97     if (!rst_n)
98         seg_sel <= 6'b111111;
99     else
100         seg_sel <= 6'b000000;
101 end
102
103 //每次通知信号到达时, 数码管显示的十六进制数值加1
```

```
104 always @ (posedge clk or negedge rst_n) begin
105     if (!rst_n)
106         num <= 4'h0;
107     else if(add_flag) begin
108         if (num < 4'hf)
109             num <= num + 1'b1;
110         else
111             num <= 4'h0;
112     end
113     else
114         num <= num;
115 end
116
117 //根据数码管显示的数值，控制段选信号
118 always @ (posedge clk or negedge rst_n) begin
119     if (!rst_n)
120         seg_led <= 8'b0;
121     else begin
122         case (num)
123             4'h0 : seg_led <= 8'b1100_0000;
124             4'h1 : seg_led <= 8'b1111_1001;
125             4'h2 : seg_led <= 8'b1010_0100;
126             4'h3 : seg_led <= 8'b1011_0000;
127             4'h4 : seg_led <= 8'b1001_1001;
128             4'h5 : seg_led <= 8'b1001_0010;
129             4'h6 : seg_led <= 8'b1000_0010;
130             4'h7 : seg_led <= 8'b1111_1000;
131             4'h8 : seg_led <= 8'b1000_0000;
132             4'h9 : seg_led <= 8'b1001_0000;
133             4'ha : seg_led <= 8'b1000_1000;
134             4'hb : seg_led <= 8'b1000_0011;
135             4'hc : seg_led <= 8'b1100_0110;
136             4'hd : seg_led <= 8'b1010_0001;
137             4'he : seg_led <= 8'b1000_0110;
138             4'hf : seg_led <= 8'b1000_1110;
139             default : seg_led <= 8'b1100_0000;
140         endcase
141     end
142 end
143
144 endmodule
```

Verilog 代码: 静态数码管显示实验变式 2

```
1  module seg_led_static_top_mode (
2      input      sys_clk,
3      input      sys_rst_n,
4      input  [1:0] key,
5      output [5:0] seg_sel,
6      output [7:0] seg_led
7  );
8
9      parameter MAX_NUM      = 25_000_000;
10     parameter HALF_MAX_NUM = 12_500_000;
11
12     wire add_flag;
13
14     time_count #(
15         .MAX_NUM(MAX_NUM),
16         .HALF_MAX_NUM(HALF_MAX_NUM)
17     ) u_time_count (
18         .clk(sys_clk),
19         .rst_n(sys_rst_n),
20         .key(key),
21         .add_flag(add_flag)
22     );
23
24     seg_led_static u_seg_led_static (
25         .clk(sys_clk),
26         .rst_n(sys_rst_n),
27         .add_flag(add_flag),
28         .seg_sel(seg_sel),
29         .seg_led(seg_led)
30     );
31
32 endmodule
33
34
35 module time_count (
36     input      clk,
37     input      rst_n,
38     input  [1:0] key,
39     output reg  add_flag
40 );
41
42     parameter MAX_NUM      = 25_000_000;
43     parameter HALF_MAX_NUM = 12_500_000;
44
45     reg [24:0] cnt;
46     reg [1:0] key_last;
47     reg      mode;
48
49     wire [1:0] key_pressed;
50     assign key_pressed = (~key) & key_last;
51
```

```
52 // Key last state
53 always @(posedge clk or negedge rst_n) begin
54     if (!rst_n)
55         key_last <= 2'b11;
56     else
57         key_last <= key;
58 end
59
60 // Mode control
61 always @(posedge clk or negedge rst_n) begin
62     if (!rst_n)
63         mode <= 1'b0;
64     else begin
65         if (key_pressed[0])
66             mode <= 1'b0;
67         else if (key_pressed[1])
68             mode <= 1'b1;
69     end
70 end
71
72 // Counter and flag control
73 always @(posedge clk or negedge rst_n) begin
74     if (!rst_n) begin
75         cnt <= 0;
76         add_flag <= 1'b0;
77     end
78     else begin
79         add_flag <= 1'b0;
80         if (mode == 1'b0) begin
81             if (cnt < MAX_NUM - 1)
82                 cnt <= cnt + 1;
83             else begin
84                 cnt <= 0;
85                 add_flag <= 1'b1;
86             end
87         end
88         else begin
89             if (cnt < HALF_MAX_NUM - 1)
90                 cnt <= cnt + 1;
91             else begin
92                 cnt <= 0;
93                 add_flag <= 1'b1;
94             end
95         end
96     end
97 end
98
99 endmodule
100
101
102 module seg_led_static (
103     input clk,
```

```
104     input      rst_n,
105     input      add_flag,
106     output reg [5:0] seg_sel,
107     output reg [7:0] seg_led
108 );
109
110     reg [3:0] num;
111
112     // Segment select control
113     always @(posedge clk or negedge rst_n) begin
114         if (!rst_n)
115             seg_sel <= 6'b111111;
116         else
117             seg_sel <= 6'b000000;
118     end
119
120     // Number counter
121     always @(posedge clk or negedge rst_n) begin
122         if (!rst_n)
123             num <= 4'h0;
124         else if (add_flag) begin
125             if (num < 4'hf)
126                 num <= num + 1'b1;
127             else
128                 num <= 4'h0;
129         end
130         else
131             num <= num;
132     end
133
134     // Segment LED pattern
135     always @(posedge clk or negedge rst_n) begin
136         if (!rst_n)
137             seg_led <= 8'b0;
138         else begin
139             case (num)
140                 4'h0: seg_led <= 8'b1100_0000;
141                 4'h1: seg_led <= 8'b1111_1001;
142                 4'h2: seg_led <= 8'b1010_0100;
143                 4'h3: seg_led <= 8'b1011_0000;
144                 4'h4: seg_led <= 8'b1001_1001;
145                 4'h5: seg_led <= 8'b1001_0010;
146                 4'h6: seg_led <= 8'b1000_0010;
147                 4'h7: seg_led <= 8'b1111_1000;
148                 4'h8: seg_led <= 8'b1000_0000;
149                 4'h9: seg_led <= 8'b1001_0000;
150                 4'ha: seg_led <= 8'b1000_1000;
151                 4'hb: seg_led <= 8'b1000_0011;
152                 4'hc: seg_led <= 8'b1100_0110;
153                 4'hd: seg_led <= 8'b1010_0001;
154                 4'he: seg_led <= 8'b1000_0110;
155                 4'hf: seg_led <= 8'b1000_1110;
```



```
156         default: seg_led <= 8'b1100_0000;
157     endcase
158 end
159 end
160
161 endmodule
```

tcl 脚本: 引脚分配

```
1
2 # 时钟与复位引脚
3 set_location_assignment PIN_M2 -to sys_clk
4 set_location_assignment PIN_M1 -to sys_rst_n
5
6 # 数码管位选信号 seg_sel[5:0]
7 set_location_assignment PIN_N16 -to seg_sel[0]
8 set_location_assignment PIN_N15 -to seg_sel[1]
9 set_location_assignment PIN_P16 -to seg_sel[2]
10 set_location_assignment PIN_P15 -to seg_sel[3]
11 set_location_assignment PIN_R16 -to seg_sel[4]
12 set_location_assignment PIN_T15 -to seg_sel[5]
13
14 # 数码管段选信号 seg_led[7:0]
15 set_location_assignment PIN_M11 -to seg_led[0]
16 set_location_assignment PIN_N12 -to seg_led[1]
17 set_location_assignment PIN_C9 -to seg_led[2]
18 set_location_assignment PIN_N13 -to seg_led[3]
19 set_location_assignment PIN_M10 -to seg_led[4]
20 set_location_assignment PIN_N11 -to seg_led[5]
21 set_location_assignment PIN_P11 -to seg_led[6]
22 set_location_assignment PIN_D9 -to seg_led[7]
23
24 # 按键信号
25 set_location_assignment PIN_E16 -to key[0]
26 set_location_assignment PIN_E15 -to key[1]
27 #set_location_assignment PIN_M15 -to key[2]
28 #set_location_assignment PIN_M16 -to key[3]
```

Verilog 代码: 动态数码管显示实验基础问题

```

1  module top_seg_led(
2      //global clock
3      input      sys_clk ,          // 全局时钟信号
4      input      sys_rst_n,         // 复位信号 (低有效)
5
6      //seg_led interface
7      output [5:0] seg_sel ,         // 数码管位选信号
8      output [7:0] seg_led          // 数码管段选信号
9  );
10
11 //wire define
12 wire [19:0] data;                  // 数码管显示的数值
13 wire [ 5:0] point;                 // 数码管小数点的位置
14 wire      en;                      // 数码管显示使能信号
15 wire      sign;                   // 数码管显示数据的符号位
16
17 //*****
18 /**                                main code
19 //*****
20
21 //计数器模块, 产生数码管需要显示的数据
22 count u_count(
23     .clk      (sys_clk ),          // 时钟信号
24     .rst_n     (sys_rst_n),        // 复位信号
25
26     .data      (data ),            // 6位数码管要显示的数值
27     .point     (point ),           // 小数点具体显示的位置,高电平有效
28     .en        (en ),              // 数码管使能信号
29     .sign      (sign )             // 符号位
30 );
31
32 //数码管动态显示模块
33 seg_led u_seg_led(
34     .clk      (sys_clk ),          // 时钟信号
35     .rst_n     (sys_rst_n),        // 复位信号
36
37     .data      (data ),            // 显示的数值
38     .point     (point ),           // 小数点具体显示的位置,高电平有效
39     .en        (en ),              // 数码管使能信号
40     .sign      (sign ),            // 符号位, 高电平显示负号(-)
41
42     .seg_sel    (seg_sel ),         // 位选
43     .seg_led    (seg_led )          // 段选
44 );
45
46 endmodule
47
48
49
50 module count(
51     //module clock

```

```

52     input                clk ,           // 时钟信号
53     input                rst_n,         // 复位信号
54
55     //user interface
56     output reg [19:0]    data ,         // 6个数码管要显示的数值
57     output reg [ 5:0]    point,         // 小数点的位置,高电平点亮对应数码管位上的小数
    点
58     output reg          en ,           // 数码管使能信号
59     output reg          sign           // 符号位, 高电平时显示负号, 低电平不显示负号
60 );
61
62 //parameter define
63 parameter MAX_NUM = 23'd5000_000;      // 计数器计数的最大值
64
65 //reg define
66 reg [22:0] cnt ;                       // 计数器, 用于计时100ms
67 reg flag;                             // 标志信号
68
69 //*****
70 /**                                main code
71 //*****
72
73 //计数器对系统时钟计数达10ms时, 输出一个时钟周期的脉冲信号
74 always @ (posedge clk or negedge rst_n) begin
75     if (!rst_n) begin
76         cnt <= 23'b0;
77         flag<= 1'b0;
78     end
79     else if (cnt < MAX_NUM - 1'b1) begin
80         cnt <= cnt + 1'b1;
81         flag<= 1'b0;
82     end
83     else begin
84         cnt <= 23'b0;
85         flag <= 1'b1;
86     end
87 end
88
89 //数码管需要显示的数据, 从0累加到999999
90 always @ (posedge clk or negedge rst_n) begin
91     if (!rst_n)begin
92         data <= 20'b0;
93         point <=6'b000000;
94         en <= 1'b0;
95         sign <= 1'b0;
96     end
97     else begin
98         point <= 6'b000000;           //不显示小数点
99         en <= 1'b1;                   //打开数码管使能信号
100        sign <= 1'b0;                 //不显示负号
101        if (flag) begin               //显示数值每隔0.01s累加一次
102            if(data < 20'd999999)

```

```

103         data <= data +1'b1;
104     else
105         data <= 20'b0;
106     end
107 end
108 end
109
110 endmodule
111
112
113 module seg_led(
114     input          clk      ,          // 时钟信号
115     input          rst_n    ,          // 复位信号
116
117     input [19:0]    data     ,          // 6位数码管要显示的数值
118     input [5:0]     point    ,          // 小数点具体显示的位置,从高到低,高电平有效
119     input          en       ,          // 数码管使能信号
120     input          sign     ,          // 符号位 (高电平显示“-”号)
121
122     output reg [5:0]    seg_sel,          // 数码管位选, 最左侧数码管为最高位
123     output reg [7:0]    seg_led          // 数码管段选
124 );
125
126 //parameter define
127 localparam CLK_DIVIDE = 4'd10      ;          // 时钟分频系数
128 localparam MAX_NUM    = 13'd5000  ;          // 对数码管驱动时钟(5MHz)计数1ms所需的计数
129     值
130
131 //reg define
132 reg [3:0]    clk_cnt ;          // 时钟分频计数器
133 reg          dri_clk ;          // 数码管的驱动时钟,5MHz
134 reg [23:0]   num     ;          // 24位bcd码寄存器
135 reg [12:0]   cnt0     ;          // 数码管驱动时钟计数器
136 reg         flag      ;          // 标志信号 (标志着cnt0计数达1ms)
137 reg [2:0]    cnt_sel  ;          // 数码管位选计数器
138 reg [3:0]    num_disp ;          // 当前数码管显示的数据
139 reg          dot_disp ;          // 当前数码管显示的小数点
140
141 //wire define
142 wire [3:0]    data0   ;          // 个位数
143 wire [3:0]    data1   ;          // 十位数
144 wire [3:0]    data2   ;          // 百位数
145 wire [3:0]    data3   ;          // 千位数
146 wire [3:0]    data4   ;          // 万位数
147 wire [3:0]    data5   ;          // 十万位数
148
149 //*****
150 //**                               main code
151 //*****
152
153 //提取显示数值所对应的十进制数的各个位
154 assign data0 = data % 4'd10;          // 个位数

```

```

154 assign data1 = data / 4'd10 % 4'd10 ; // 十位数
155 assign data2 = data / 7'd100 % 4'd10 ; // 百位数
156 assign data3 = data / 10'd1000 % 4'd10 ; // 千位数
157 assign data4 = data / 14'd10000 % 4'd10; // 万位数
158 assign data5 = data / 17'd100000; // 十万位数
159
160 //对系统时钟10分频, 得到的频率为5MHz的数码管驱动时钟dri_clk
161 always @(posedge clk or negedge rst_n) begin
162     if(!rst_n) begin
163         clk_cnt <= 4'd0;
164         dri_clk <= 1'b1;
165     end
166     else if(clk_cnt == CLK_DIVIDE/2 - 1'd1) begin
167         clk_cnt <= 4'd0;
168         dri_clk <= ~dri_clk;
169     end
170     else begin
171         clk_cnt <= clk_cnt + 1'b1;
172         dri_clk <= dri_clk;
173     end
174 end
175
176 //将20位2进制数转换为8421bcd码(即使用4位二进制数表示1位十进制数)
177 always @ (posedge dri_clk or negedge rst_n) begin
178     if (!rst_n)
179         num <= 24'b0;
180     else begin
181         if (data5 || point[5]) begin //如果显示数据为6位十进制数,
182             num[23:20] <= data5; //则依次给6位数码管赋值
183             num[19:16] <= data4;
184             num[15:12] <= data3;
185             num[11:8] <= data2;
186             num[ 7:4] <= data1;
187             num[ 3:0] <= data0;
188         end
189         else begin
190             if (data4 || point[4]) begin //如果显示数据为5位十进制数, 则给低5位数码管赋
191                 值 num[19:0] <= {data4,data3,data2,data1,data0};
192                 if(sign)
193                     num[23:20] <= 4'd11; //如果需要显示负号, 则最高位(第6位)为符号位
194                 else
195                     num[23:20] <= 4'd10; //不需要显示负号时, 则第6位不显示任何字符
196             end
197             else begin //如果显示数据为4位十进制数, 则给低4位数码管赋
198                 值 if (data3 || point[3]) begin
199                     num[15: 0] <= {data3,data2,data1,data0};
200                     num[23:20] <= 4'd10; //第6位不显示任何字符
201                     if(sign) //如果需要显示负号, 则最高位(第5位)为符号位
202                         num[19:16] <= 4'd11;
203                     else //不需要显示负号时, 则第5位不显示任何字符

```

```

204         num[19:16] <= 4'd10;
205     end
206     else begin //如果显示数据为3位十进制数, 则给低3位数码管赋
值
207         if (data2 || point[2]) begin
208             num[11: 0] <= {data2,data1,data0};
209             //第6、5位不显示任何字符
210             num[23:16] <= {2{4'd10}};
211             if(sign) //如果需要显示负号, 则最高位(第4位)为符号位
212                 num[15:12] <= 4'd11;
213             else //不需要显示负号时, 则第4位不显示任何字符
214                 num[15:12] <= 4'd10;
215         end
216     else begin //如果显示数据为2位十进制数, 则给低2位数码管赋
值
217         if (data1 || point[1]) begin
218             num[ 7: 0] <= {data1,data0};
219             //第6、5、4位不显示任何字符
220             num[23:12] <= {3{4'd10}};
221             if(sign) //如果需要显示负号, 则最高位(第3位)为符号位
222                 num[11:8] <= 4'd11;
223             else //不需要显示负号时, 则第3位不显示任何字符
224                 num[11:8] <= 4'd10;
225         end
226     else begin //如果显示数据为1位十进制数, 则给最低位数码管
赋值
227         num[3:0] <= data0;
228         //第6、5位不显示任何字符
229         num[23:8] <= {4{4'd10}};
230         if(sign) //如果需要显示负号, 则最高位(第2位)为符号位
231             num[7:4] <= 4'd11;
232         else //不需要显示负号时, 则第2位不显示任何字符
233             num[7:4] <= 4'd10;
234         end
235     end
236 end
237 end
238 end
239 end
240 end
241
242 //每当计数器对数码管驱动时钟计数时间达1ms, 输出一个时钟周期的脉冲信号
243 always @ (posedge dri_clk or negedge rst_n) begin
244     if (rst_n == 1'b0) begin
245         cnt0 <= 13'b0;
246         flag <= 1'b0;
247     end
248     else if (cnt0 < MAX_NUM - 1'b1) begin
249         cnt0 <= cnt0 + 1'b1;
250         flag <= 1'b0;
251     end
252     else begin

```

```
253         cnt0 <= 13'b0;
254         flag <= 1'b1;
255     end
256 end
257
258 //cnt_sel从0计数到5, 用于选择当前处于显示状态的数码管
259 always @ (posedge dri_clk or negedge rst_n) begin
260     if (rst_n == 1'b0)
261         cnt_sel <= 3'b0;
262     else if(flag) begin
263         if(cnt_sel < 3'd5)
264             cnt_sel <= cnt_sel + 1'b1;
265         else
266             cnt_sel <= 3'b0;
267     end
268     else
269         cnt_sel <= cnt_sel;
270 end
271
272 //控制数码管位选信号, 使6位数码管轮流显示
273 always @ (posedge dri_clk or negedge rst_n) begin
274     if(!rst_n) begin
275         seg_sel <= 6'b111111;           //位选信号低电平有效
276         num_disp <= 4'b0;
277         dot_disp <= 1'b1;               //共阳极数码管, 低电平导通
278     end
279     else begin
280         if(en) begin
281             case (cnt_sel)
282                 3'd0 :begin
283                     seg_sel <= 6'b111110; //显示数码管最低位
284                     num_disp <= num[3:0] ; //显示的数据
285                     dot_disp <= ~point[0]; //显示的小数点
286                 end
287                 3'd1 :begin
288                     seg_sel <= 6'b111101; //显示数码管第1位
289                     num_disp <= num[7:4] ;
290                     dot_disp <= ~point[1];
291                 end
292                 3'd2 :begin
293                     seg_sel <= 6'b111011; //显示数码管第2位
294                     num_disp <= num[11:8];
295                     dot_disp <= ~point[2];
296                 end
297                 3'd3 :begin
298                     seg_sel <= 6'b110111; //显示数码管第3位
299                     num_disp <= num[15:12];
300                     dot_disp <= ~point[3];
301                 end
302                 3'd4 :begin
303                     seg_sel <= 6'b101111; //显示数码管第4位
304                     num_disp <= num[19:16];
```

```

305         dot_disp <= ~point[4];
306     end
307     3'd5 :begin
308         seg_sel  <= 6'b011111; //显示数码管最高位
309         num_disp <= num[23:20];
310         dot_disp <= ~point[5];
311     end
312     default :begin
313         seg_sel  <= 6'b111111;
314         num_disp <= 4'b0;
315         dot_disp <= 1'b1;
316     end
317 endcase
318 end
319 else begin
320     seg_sel  <= 6'b111111;           //使能信号为0时，所有数码管均不显示
321     num_disp <= 4'b0;
322     dot_disp <= 1'b1;
323 end
324 end
325 end
326
327 //控制数码管段选信号，显示字符
328 always @ (posedge dri_clk or negedge rst_n) begin
329     if (!rst_n)
330         seg_led <= 8'hc0;
331     else begin
332         case (num_disp)
333             4'd0 : seg_led <= {dot_disp,7'b1000000}; //显示数字 0
334             4'd1 : seg_led <= {dot_disp,7'b1111001}; //显示数字 1
335             4'd2 : seg_led <= {dot_disp,7'b0100100}; //显示数字 2
336             4'd3 : seg_led <= {dot_disp,7'b0110000}; //显示数字 3
337             4'd4 : seg_led <= {dot_disp,7'b0011001}; //显示数字 4
338             4'd5 : seg_led <= {dot_disp,7'b0010010}; //显示数字 5
339             4'd6 : seg_led <= {dot_disp,7'b0000010}; //显示数字 6
340             4'd7 : seg_led <= {dot_disp,7'b1111000}; //显示数字 7
341             4'd8 : seg_led <= {dot_disp,7'b0000000}; //显示数字 8
342             4'd9 : seg_led <= {dot_disp,7'b0010000}; //显示数字 9
343             4'd10: seg_led <= 8'b1111111;           //不显示任何字符
344             4'd11: seg_led <= 8'b1011111;           //显示负号(-)
345             default:
346                 seg_led <= {dot_disp,7'b1000000};
347         endcase
348     end
349 end
350
351 endmodule

```


Verilog 代码: 动态数码管显示实验变式 1

```
1  module top_seg_led_key (
2      input  sys_clk,
3      input  sys_rst_n,
4      input  [2:0] key,
5      output [5:0] seg_sel,
6      output [7:0] seg_led
7  );
8
9      wire [19:0] data;
10     wire [5:0] point;
11     wire      en;
12     wire      sign;
13
14     count u_count (
15         .clk      (sys_clk),
16         .rst_n    (sys_rst_n),
17         .key      (key),
18         .data     (data),
19         .point    (point),
20         .en       (en),
21         .sign     (sign)
22     );
23
24     seg_led u_seg_led (
25         .clk      (sys_clk),
26         .rst_n    (sys_rst_n),
27         .data     (data),
28         .point    (point),
29         .en       (en),
30         .sign     (sign),
31         .seg_sel  (seg_sel),
32         .seg_led  (seg_led)
33     );
34
35 endmodule
36
37
38 module count (
39     input      clk,
40     input      rst_n,
41     input  [2:0] key,
42     output reg [19:0] data,
43     output reg [5:0] point,
44     output reg      en,
45     output reg      sign
46 );
47
48     parameter MAX_NUM = 23'd5_000_000;
49
50     reg [22:0] cnt;
51     reg      flag;
```

```
52     reg [2:0] key_last;
53     reg      stop;
54     reg      add;
55
56     wire [2:0] key_pressed;
57     assign key_pressed = (~key) & key_last;
58
59     // Key last state
60     always @(posedge clk or negedge rst_n) begin
61         if (!rst_n)
62             key_last <= 3'b111;
63         else
64             key_last <= key;
65     end
66
67     // Stop and add control
68     always @(posedge clk or negedge rst_n) begin
69         if (!rst_n) begin
70             stop <= 1'b1;
71             add  <= 1'b1;
72         end
73         else begin
74             if (key_pressed[0]) begin
75                 if (stop == 1'b1)
76                     stop <= 1'b0;
77                 else if (stop == 1'b0)
78                     stop <= 1'b1;
79             end
80             else if (key_pressed[1]) begin
81                 if (add == 1'b1)
82                     add <= 1'b0;
83                 else if (add == 1'b0)
84                     add <= 1'b1;
85             end
86         end
87     end
88
89     // Counter logic
90     always @(posedge clk or negedge rst_n) begin
91         if (!rst_n) begin
92             cnt  <= 23'b0;
93             flag <= 1'b0;
94         end
95         else if (key_pressed[2]) begin
96             cnt  <= 23'b0;
97             flag <= 1'b0;
98         end
99         else begin
100             if (stop == 1'b0) begin
101                 cnt <= cnt;
102                 flag <= 1'b0;
103             end
```

```
104         else begin
105             if (cnt < MAX_NUM - 1'b1) begin
106                 cnt  <= cnt + 1'b1;
107                 flag <= 1'b0;
108             end
109         else begin
110             cnt  <= 23'b0;
111             flag <= 1'b1;
112         end
113     end
114 end
115 end
116
117 // Data output logic
118 always @(posedge clk or negedge rst_n) begin
119     if (!rst_n) begin
120         data  <= 20'b0;
121         point <= 6'b000000;
122         en    <= 1'b0;
123         sign  <= 1'b0;
124     end
125     else if (key_pressed[2]) begin
126         data <= 20'b0;
127     end
128     else begin
129         point <= 6'b000000;
130         en    <= 1'b1;
131         sign  <= 1'b0;
132         if (flag) begin
133             if (add == 1'b1) begin
134                 if (data < 20'd999999)
135                     data <= data + 1'b1;
136                 else
137                     data <= 20'b0;
138             end
139             else if (add == 1'b0) begin
140                 if (data > 20'b0)
141                     data <= data - 1'b1;
142                 else
143                     data <= 20'b0;
144             end
145         end
146     end
147 end
148
149 endmodule
150
151
152 module seg_led (
153     input      clk,
154     input      rst_n,
155     input [19:0] data,
```

```
156     input [5:0] point,
157     input      en,
158     input      sign,
159     output reg [5:0] seg_sel,
160     output reg [7:0] seg_led
161 );
162
163     localparam CLK_DIVIDE = 4'd10;
164     localparam MAX_NUM    = 13'd5000;
165
166     reg [3:0] clk_cnt;
167     reg      dri_clk;
168     reg [23:0] num;
169     reg [12:0] cnt0;
170     reg      flag;
171     reg [2:0] cnt_sel;
172     reg [3:0] num_disp;
173     reg      dot_disp;
174
175     wire [3:0] data0;
176     wire [3:0] data1;
177     wire [3:0] data2;
178     wire [3:0] data3;
179     wire [3:0] data4;
180     wire [3:0] data5;
181
182     assign data0 = data % 4'd10;
183     assign data1 = data / 4'd10 % 4'd10;
184     assign data2 = data / 7'd100 % 4'd10;
185     assign data3 = data / 10'd1000 % 4'd10;
186     assign data4 = data / 14'd10000 % 4'd10;
187     assign data5 = data / 17'd100000;
188
189     // Clock divider
190     always @(posedge clk or negedge rst_n) begin
191         if (!rst_n) begin
192             clk_cnt <= 4'd0;
193             dri_clk <= 1'b1;
194         end
195         else if (clk_cnt == CLK_DIVIDE/2 - 1'd1) begin
196             clk_cnt <= 4'd0;
197             dri_clk <= ~dri_clk;
198         end
199         else begin
200             clk_cnt <= clk_cnt + 1'b1;
201             dri_clk <= dri_clk;
202         end
203     end
204
205     // Number formatting
206     always @(posedge dri_clk or negedge rst_n) begin
207         if (!rst_n)
```

```

208         num <= 24'b0;
209     else begin
210         if (data5 || point[5]) begin
211             num[23:20] <= data5;
212             num[19:16] <= data4;
213             num[15:12] <= data3;
214             num[11:8] <= data2;
215             num[7:4] <= data1;
216             num[3:0] <= data0;
217         end
218     else begin
219         if (data4 || point[4]) begin
220             num[19:0] <= {data4, data3, data2, data1, data0};
221             num[23:20] <= sign ? 4'd11 : 4'd10;
222         end
223     else begin
224         if (data3 || point[3]) begin
225             num[15:0] <= {data3, data2, data1, data0};
226             num[23:20] <= 4'd10;
227             num[19:16] <= sign ? 4'd11 : 4'd10;
228         end
229     else begin
230         if (data2 || point[2]) begin
231             num[11:0] <= {data2, data1, data0};
232             num[23:16] <= {2{4'd10}};
233             num[15:12] <= sign ? 4'd11 : 4'd10;
234         end
235     else begin
236         if (data1 || point[1]) begin
237             num[7:0] <= {data1, data0};
238             num[23:12] <= {3{4'd10}};
239             num[11:8] <= sign ? 4'd11 : 4'd10;
240         end
241     else begin
242         num[3:0] <= data0;
243         num[23:8] <= {4{4'd10}};
244         num[7:4] <= sign ? 4'd11 : 4'd10;
245     end
246 end
247 end
248 end
249 end
250 end
251 end
252
253 // Counter for scanning
254 always @(posedge dri_clk or negedge rst_n) begin
255     if (!rst_n) begin
256         cnt0 <= 13'b0;
257         flag <= 1'b0;
258     end
259     else if (cnt0 <= MAX_NUM - 1'b1) begin

```

```
260         cnt0 <= cnt0 + 1'b1;
261         flag <= 1'b0;
262     end
263     else begin
264         cnt0 <= 13'b0;
265         flag <= 1'b1;
266     end
267 end
268
269 // Segment select counter
270 always @(posedge dri_clk or negedge rst_n) begin
271     if (!rst_n)
272         cnt_sel <= 3'b0;
273     else if (flag) begin
274         if (cnt_sel < 3'd5)
275             cnt_sel <= cnt_sel + 1'b1;
276         else
277             cnt_sel <= 3'b0;
278     end
279     else
280         cnt_sel <= cnt_sel;
281 end
282
283 // Segment selection and display
284 always @(posedge dri_clk or negedge rst_n) begin
285     if (!rst_n) begin
286         seg_sel <= 6'b111111;
287         num_disp <= 4'b0;
288         dot_disp <= 1'b1;
289     end
290     else begin
291         if (en) begin
292             case (cnt_sel)
293                 3'd0: begin
294                     seg_sel <= 6'b111110;
295                     num_disp <= num[3:0];
296                     dot_disp <= ~point[0];
297                 end
298                 3'd1: begin
299                     seg_sel <= 6'b111101;
300                     num_disp <= num[7:4];
301                     dot_disp <= ~point[1];
302                 end
303                 3'd2: begin
304                     seg_sel <= 6'b111011;
305                     num_disp <= num[11:8];
306                     dot_disp <= ~point[2];
307                 end
308                 3'd3: begin
309                     seg_sel <= 6'b110111;
310                     num_disp <= num[15:12];
311                     dot_disp <= ~point[3];
```

```

312         end
313         3'd4: begin
314             seg_sel  <= 6'b101111;
315             num_disp <= num[19:16];
316             dot_disp <= ~point[4];
317         end
318         3'd5: begin
319             seg_sel  <= 6'b011111;
320             num_disp <= num[23:20];
321             dot_disp <= ~point[5];
322         end
323         default: begin
324             seg_sel  <= 6'b111111;
325             num_disp <= 4'b0;
326             dot_disp <= 1'b1;
327         end
328     endcase
329 end
330 else begin
331     seg_sel  <= 6'b111111;
332     num_disp <= 4'b0;
333     dot_disp <= 1'b1;
334 end
335 end
336 end
337
338 // Segment LED pattern
339 always @(posedge dri_clk or negedge rst_n) begin
340     if (!rst_n)
341         seg_led <= 8'hc0;
342     else begin
343         case (num_disp)
344             4'd0: seg_led <= {dot_disp, 7'b1000000};
345             4'd1: seg_led <= {dot_disp, 7'b1111001};
346             4'd2: seg_led <= {dot_disp, 7'b0100100};
347             4'd3: seg_led <= {dot_disp, 7'b0110000};
348             4'd4: seg_led <= {dot_disp, 7'b0011001};
349             4'd5: seg_led <= {dot_disp, 7'b0010010};
350             4'd6: seg_led <= {dot_disp, 7'b0000010};
351             4'd7: seg_led <= {dot_disp, 7'b1111000};
352             4'd8: seg_led <= {dot_disp, 7'b0000000};
353             4'd9: seg_led <= {dot_disp, 7'b0010000};
354             4'd10: seg_led <= 8'b1111111;
355             4'd11: seg_led <= 8'b1011111;
356             default: seg_led <= {dot_disp, 7'b1000000};
357         endcase
358     end
359 end
360
361 endmodule

```

tcl 脚本：引脚分配

```
1  # 时钟与复位引脚
2  set_location_assignment PIN_M2 -to sys_clk
3  set_location_assignment PIN_M1 -to sys_rst_n
4
5  # 数码管位选信号 seg_sel[5:0]
6  set_location_assignment PIN_N16 -to seg_sel[0]
7  set_location_assignment PIN_N15 -to seg_sel[1]
8  set_location_assignment PIN_P16 -to seg_sel[2]
9  set_location_assignment PIN_P15 -to seg_sel[3]
10 set_location_assignment PIN_R16 -to seg_sel[4]
11 set_location_assignment PIN_T15 -to seg_sel[5]
12
13 # 数码管段选信号 seg_led[7:0]
14 set_location_assignment PIN_M11 -to seg_led[0]
15 set_location_assignment PIN_N12 -to seg_led[1]
16 set_location_assignment PIN_C9 -to seg_led[2]
17 set_location_assignment PIN_N13 -to seg_led[3]
18 set_location_assignment PIN_M10 -to seg_led[4]
19 set_location_assignment PIN_N11 -to seg_led[5]
20 set_location_assignment PIN_P11 -to seg_led[6]
21 set_location_assignment PIN_D9 -to seg_led[7]
22
23 # 按键信号
24 set_location_assignment PIN_E16 -to key[0]
25 set_location_assignment PIN_E15 -to key[1]
26 set_location_assignment PIN_M15 -to key[2]
27 #set_location_assignment PIN_M16 -to key[3]
```