电子森林-计算器 FPGA 软件

密级：公开

计算器 FPGA 软件设计与验证报告

Design Description and Verification Record of Calculator

MYFPGA_CN_FPGA_BASIC_YHJSQ_SJSM_012

拟制	ChanRa1n	时间	2024/01/15
评审	/	时间	/
签发	ChanRa1n	时间	2024/01/15

myFPGA.cn

灵活应变 配置可能

内容简介

1. 项目需求

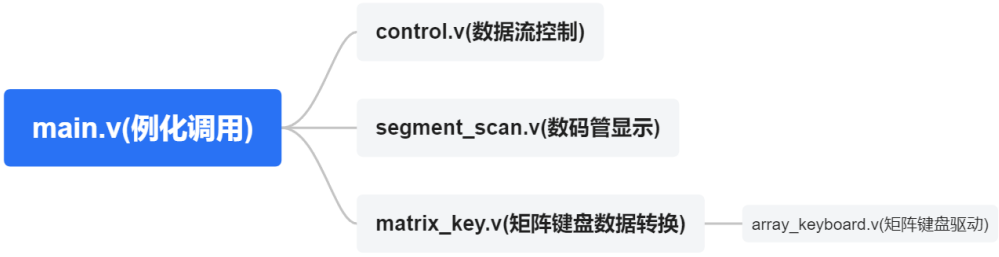
根据提供的信息，项目要求独立完成一个两位十进制数的加、减、乘、除运算的计算器。运算数和运算符由按键控制，结果通过 8 个八段数码管显示。基本要求还包括输入两位十进制数时，最高位先在右侧显示，然后其跳变到左侧的数码管上，低位在刚才高位占据的数码管上显示。扩展要求则是使用 TFTLCD 代替数码管，并设计自定义的显示界面。

2. 需求分析

基于提供的硬件资源，我将使用 Diamond 3.12 来实现所有功能。同时，由于只需要对两位十进制数进行加减乘除运算，不涉及复杂表达式和不定长数值，故不需要使用栈来进行优先级匹配。因此，我首先需要设计一个状态机，根据按键输入来控制运算数和运算符的输入，然后进行运算，最后将结果显示在数码管上。

3. 流程简述

输入控制->运算处理->结果显示（数码管/TFTLCD）。



修订记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本 01（2024-01-06）

第一次正式发布。

文档版本 02（2024-01-15）

根据实物进行实物测试，修复了一些易用性方面的问题。

文档版本 03（2024-03-15）

对文档进行了细节修复，统计字数 10101 字，满足 2000 字要求。

目录

1 范围

1.1 文档标识

- FPGA 软件标识：MYFPGA_CN_FPGA_BASIC_YHJSQ；
- 本文档标识为：MYFPGA_CN_FPGA_BASIC_YHJSQ_SJSM_012；
- 软件版本由 MyFPGA.SWS 组织过程资产管理工具动态管理。

1.2 文档综述

本文档主要内容包括计算器 FPGA 软件的范围(由需求定义)、概要设计、详细设计、验证设计、验证记录和验收与交付等方面的内容。

1.3 引用文档

1. (内部)项目管理部《民用 FPGA 软件配置管理 2021》;
2. (公开)苏州硬禾信息科技有限公司《原理图 STEP-Baseboard-V4.0.pdf》;
3. (公开)苏州硬禾信息科技有限公司《FPGA 引脚定义.pptx》;
4. (公开)苏州硬禾信息科技有限公司《小脚丫 FPGA 套件 STEP BaseBoard V4.0》。

1.4 环境

本节详细描述了 FPGA 软件工作的硬件环境和软件环境。

1.4.1 硬件环境

FPGA 软件工作的硬件环境主要包括以下几个关键组成部分：

1. STEP BaseBoard V4.0： 这是一个 FPGA 扩展底板，具有丰富的板载资源，包括存储器、温湿度传感器、接近式传感器、矩阵键盘、旋转编码器、HDMI 接口、RGBLCD 液晶屏、数码管、蜂鸣器模块、UART 通信模块、ADC 模块、DAC 模块和 WIFI 通信模块等。该底板的黄金比例尺寸为 100mm*161.8mm，为数字逻辑、微机原理、可编程逻辑语言以及 EDA 设计工具等课程提供了完美的实验平台。
2. STEP-MXO2-LPC 核心模块： 作为 FPGA 学习平台的核心，采用了 Type C 接口，基于 Lattice MXO2 芯片。该核心模块具有较小的尺寸（52mm x 18mm），支持 U 盘模式下载，无需外部重新配置 FPGA，适合初学者学习数字电路。
3. 硬件资源： FPGA 核心模块采用了 Lattice LCMXO2-4000HC-4MG132 芯片，具有 4320 个 LUT 资源、96Kbit 用户闪存、92Kbit RAM，2+2 路 PLL+DLL，支持 DDR/DDR2/LPDDR 存储器，104 个可热插拔 I/O 等。板上资源还包括数码管、RGB LED、拨码开关、按键等，为用户提供丰富的输入输出接口。

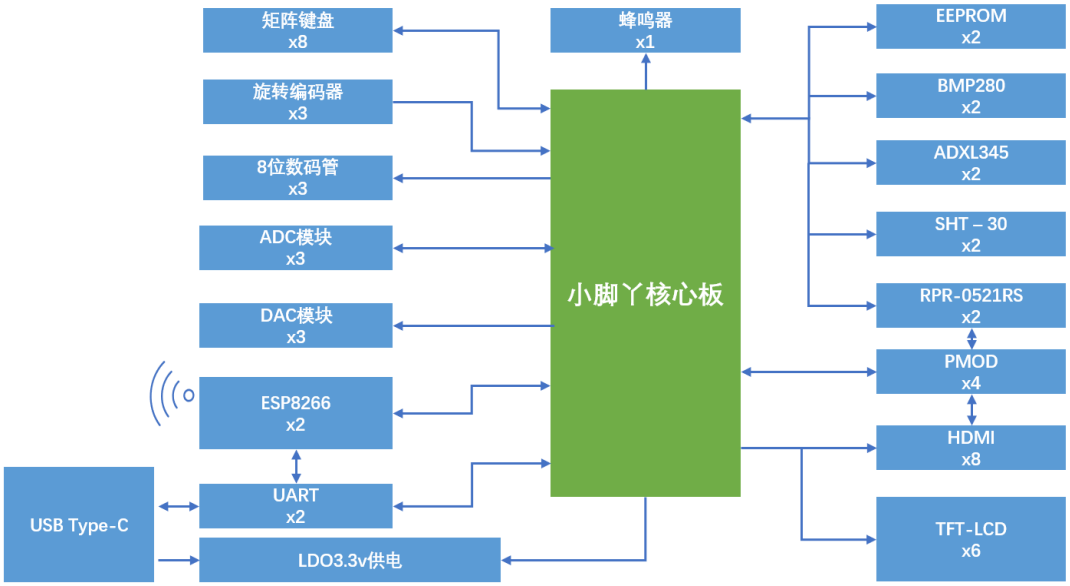


图 1.1 小脚丫硬件框图

1.4.2 软件环境

FPGA 软件工作的软件环境主要包括以下几个关键组成部分：

1. 操作系统：Windows 11 22H2 专业版(22621.2861)
2. FPGA 综合与布局布线工具： Diamond 3.12
3. 编程语言：Verilog 与 VHDL
4. 验证语言：SystemVerilog
5. 仿真工具：ModelSim 18.1

1.5 技术要求

本节详细描述了 FPGA 软件的技术要求，包括但不限于需求描述、接口定义、接口功能和性能等方面。

1.5.1 需求描述

原文需求描述

1.基本要求：

运算数和计算结果通过 8 个八段数码管显示。每个运算数使用两个数码管显示，左侧显示十位数，右侧显示个位数。输入两位十进制数时，最高位先在右侧显示，然后其跳变到左侧的数码管上，低位在刚才高位占据的数码管上显示。

2.扩展要求：

使用 TFTLCD 来做显示，自行设计显示界面，代替数码管显示输

入的参数、运算符以及计算后的结果。

定义范围(包含隐含需求)

1. 基本计算功能：
   - 实现两位十进制数的加、减、乘、除运算。
   - 通过 4x4 矩阵键盘输入运算数和运算符。
2. 数码管显示功能：
   - 使用 8 个八段数码管显示输入的运算数和运算结果。
   - 实现在输入两位十进制数时，最高位先在右侧显示，然后跳变到左侧的数码管上，低位在刚才高位占据的数码管上显示。
3. TFTLCD 显示功能（扩展要求）：
   - 设计自定义的 TFTLCD 显示界面，替代数码管显示。
   - 显示输入的参数、运算符以及计算后的结果。
4. 状态机控制：
   - 实现状态机来控制整个计算器的操作流程。
   - 状态机应包括空闲状态、输入状态、运算状态、显示状态等。
5. 按键控制：
   - 通过 4x4 矩阵键盘进行用户输入。
   - 包括数字键、运算符键、等号键等。
6. Web IDE 支持：
   - 使用思得普提供的 Web IDE 完成所有功能的开发。

7. 仿真验证：
   - 对代码中的每个模块进行仿真验证。
   - 提供详细的仿真波形图，验证各个模块的正确性。
8. FPGA 资源利用说明：
   - 提供详细的说明，说明 FPGA 芯片资源的使用情况。
   - 包括 LUT 资源、用户闪存、RAM 等。
9. 演示视频：
   - 创建 3-5 分钟的演示视频，展示计算器的各个功能。
   - 视频应包括使用 4x4 矩阵键盘输入、数码管显示、TFTLCD 显示（如果有）、运算过程等。
10. 完整的报告：
    - 提供完整的报告，包括项目需求、需求分析、实现方式、功能框图、代码及说明、仿真波形图、FPGA 资源利用说明等。
    - 确保报告中有充分的文字描述，让读者了解整个项目的设计和实现。
11. U 盘模式下载：
    - 通过 USB Type C 接口实现供电和 FPGA 的配置。
    - 支持 U 盘模式下载，使得在任何操作系统的电脑上都可以直接将生成的 JED 文件发送到 FPGA 板中完成编程。
12. 易用性提升：
    - 确保整个系统的易用性，尽量减少用户的配置和操作复杂性。

- 提供清晰的指导，使得初学者能够轻松上手。

13. 硬件环境兼容性：

- 确保软件在提供的硬件环境中能够正常运行，兼容 STEP BaseBoard V4.0 和 STEP-MXO2-LPC 核心模块。

14. 用户交互体验：

- 考虑用户交互体验，确保在输入和观察结果时，用户能够清晰地理解当前状态和操作。

15. 异常处理：

- 实现合理的异常处理机制，确保在用户输入错误或其他异常情况下系统能够给出友好的提示或恢复正常状态。

下面将在详细设计和验证验收部分对需求进行追踪和确认。

1.5.2 接口定义

计算器 FPGA 软件的输入输出信号及引脚分配如表 1.1 所示。

表 1.1 接口定义表

序号	名称	引脚映射	方向	位宽	作用
I2C 接口					
1	I2C_SCL	C8	/	1	I2C SCL 时钟端口
2	I2C_SDA	B8	/	1	I2C SDA 数据端口
编码器接口					
3	A_OUT	E3	/	1	编码器 A 端
4	B_OUT	F3	/	1	编码器 B 端
5	D_OUT	G3	/	1	编码器 D 端
矩阵键盘接口					
6	COL1	H3	/	1	矩阵键盘 COL1 端
7	COL2	J2	/	1	矩阵键盘 COL2 端
8	COL3	J3	/	1	矩阵键盘 COL3 端
9	COL4	K2	/	1	矩阵键盘 COL4 端
10	ROW1	P6	/	1	矩阵键盘 ROW1 端
11	ROW2	N5	/	1	矩阵键盘 ROW2 端
12	ROW3	L3	/	1	矩阵键盘 ROW3 端
13	ROW4	K3	/	1	矩阵键盘 ROW4 端
LCD/HDMI 接口					
14	DISP1	G13	/	1	HDMI DISP1 端
15	DISP2	N8	/	1	HDMI DISP2 端
16	DISP3	P8	/	1	HDMI DISP3 端
17	DISP4	N7	/	1	HDMI DISP4 端
18	DISP5	P7	/	1	HDMI DISP5 端
19	DISP6	N6	/	1	HDMI DISP6 端
串口/ESP-12					
20	TXD_CH340	E12	/	1	串口发送 TXD 端口
21	RXD_CH340	F12	/	1	串口接收 RXD 端口
22	TXD0	G12	/	1	ESP-12F TX-RX
23	RXD0	F13	/	1	ESP-12F RX-TX
调试 IO					
24	INT	F14	/	1	/

ADC/DAC 接口					
25	ADC_CS	G14	/	1	ADC CS 片选端口
26	ADC_SDATA	H12	/	1	ADC SDATA 数据端口
27	ADC_SCLK	J13	/	1	ADC SCLK 时钟端口
28	DAC_SYNC	J14	/	1	DAC SYNC 同步端口
29	DAC_SCLK	K12	/	1	DAC SCLK 时钟端口
30	DAC_SDI	K14	/	1	DAC SDI 输入端口
PMOD 接口					
31	PMOD_IO2(DISP7)	K13	/	1	PMOD 接口
32	PMOD_IO1(DISP8)	J12	/	1	PMOD 接口
74HC595 接口					
33	595_RCK	P3	/	1	74HC595 的 RCK 端
34	595_SCK	M4	/	1	74HC595 的 SCK 端
35	595_DIN	P13	/	1	74HC595 的 DIN 端
蜂鸣器接口					
36	BEEP	N4	/	1	蜂鸣器
系统接口					
37	CLK	C1	I	1	12M 系统时钟

1.5.3 接口功能

1.5.3.1 数据输入功能

计算器 FPGA 软件主要需要使用矩阵键盘实现数据输入功能，板载有 4*4 的矩阵键盘，通过行列互联的方式，以 4 个行和 4 个列的公共端口接入 FPGA。通过对矩阵键盘的功能进行定义，可以实现操作数 0-9，以及+-*/这四个操作符的输入。相关原理图如图 1.2。

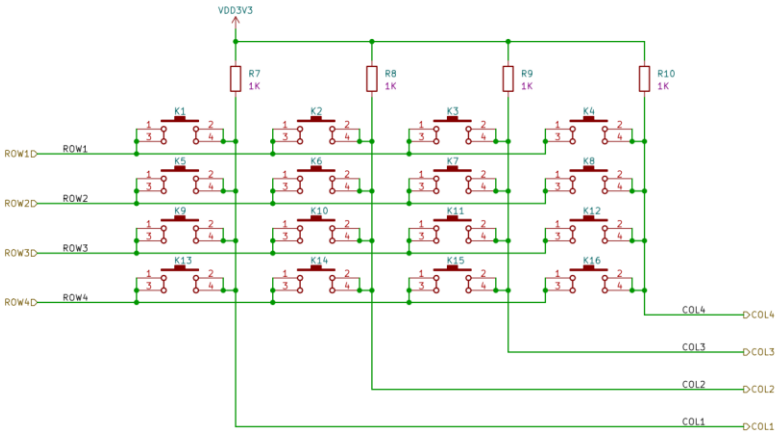


图 1.2 矩阵键盘原理图

1.5.3.2 数码管显示功能

计算器 FPGA 软件主要需要使用数码管实现操作数、操作符和运算结果的显示，板载 8 位数码管通过 74HC595 芯片实现扫描和显示。相关原理图如图 1.3。

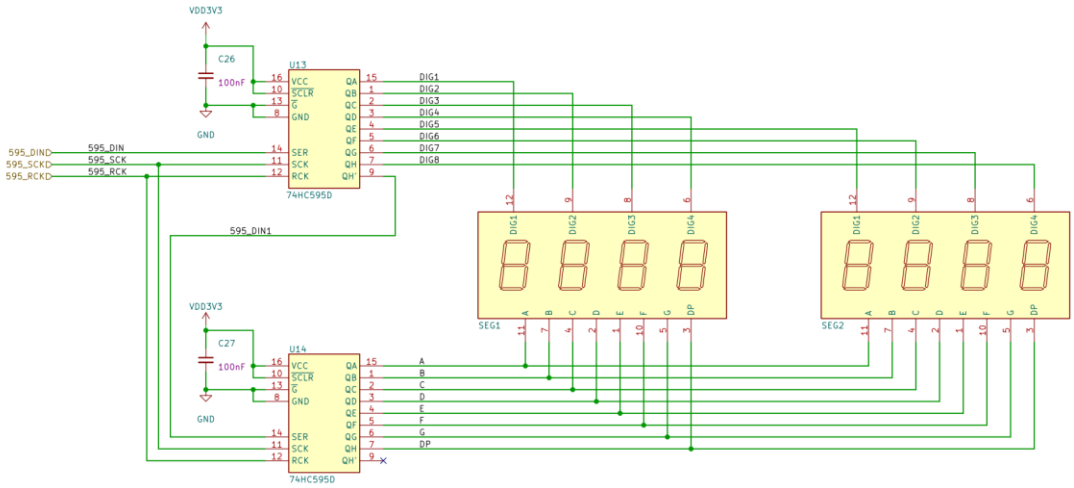


图 1.3 数码管原理图

1.5.3.3 TFLCD 显示功能

计算器 FPGA 软件主要需要使用 TFLCD 实现操作数、操作符和运

算结果的显示。相关原理图如图 1.4。

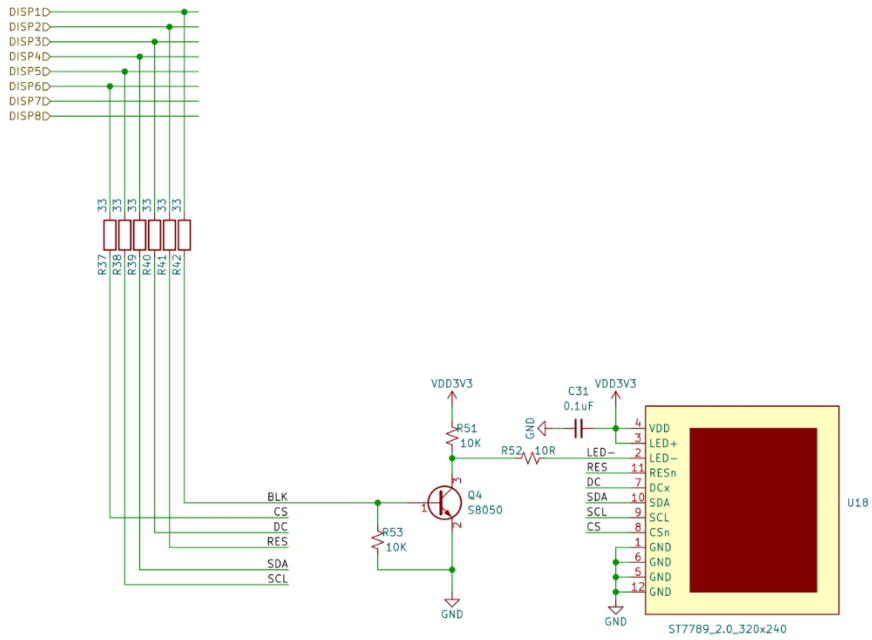


图 1.4 TFLCD 原理图

1.5.4 功能描述

1.5.4.1 基本计算功能

实现两位十进制数的加、减、乘、除运算。用户通过使用 4x4 矩阵键盘来输入运算数和运算符。

1.5.4.2 输入控制

通过状态机对用户输入进行控制，包括输入第一个数、输入第二个数、输入运算符等状态。这确保了用户在不同阶段能够正确输入所需的信息。

1.5.4.3 结果显示

将运算结果通过 8 个八段数码管进行显示。设计逻辑确保最高位先在右侧数码管显示，然后跳变到左侧的数码管，低位在刚才高位占据的数码管上显示。

1.5.4.4 TFTLCD 显示（扩展功能）

使用 TFTLCD 代替数码管进行结果的显示。此部分需要额外的设计，包括界面布局、显示参数、输入信息等。

1.5.5 性能

无强制性性能需求。

1.6 质量控制

本软件无质量控制需求，根据标准化要求，遵循《MyFPGA 质量保证计划(2021.2)》对本 FPGA 软件进行质量保证。

1.7 验收准则

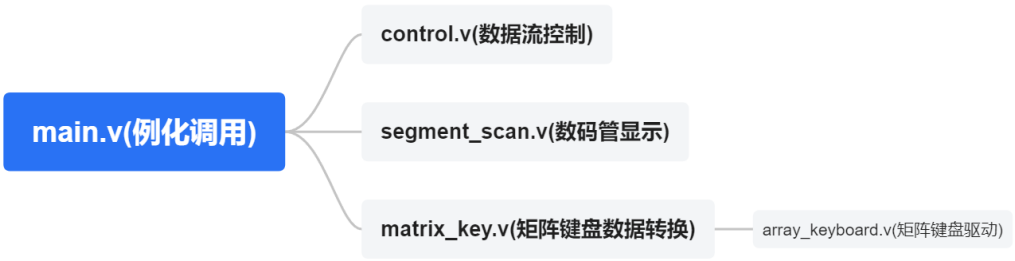
评审标准

1. 独立完成项目实现任务需求中所要求的功能，我们会验证每一个项目的代码，如发现任何抄袭者，原创者和抄袭者都会被取消资格。

2. 使用思得普的 WebIDE 来实现所有的功能
3. 针对代码中的每一个模块都要做仿真，并在报告中将仿真的波形图附上
4. 报告中要包含下面几个部分：
    1. 项目需求
    2. 需求分析
    3. 实现的方式
    4. 功能框图
    5. 代码（内嵌到报告中）及说明
    6. 仿真波形图
    7. FPGA 的资源利用说明
    8. 演示视频（3-5 分钟）
    9. 代码附件（上传到电子森林）

2 详细设计

2.1 原理框图



2.2 模块化设计

本软件使用自顶向下和模块化的开发方法，由于需求非常明确，采用瀑布模型进行开发，以下是详细的模块化设计记录和代码。

2.2.1 例化调用

Main.v 仅起到例化调用的作用，源代码如下：

```
//------------------------------------------------------------
// File Name        : main.v
// Author           : ChanRa1n
// Description      : 2024 电子森林一起练活动代码 Topmodule
// Called by        : /
// Revision History : 2024-01-06
// Revision         : 1.0
// Email            : chenyu@myfpga.cn
// Copyright(c) 2018-Now, MYFPGA.CN, All right reserved.
//------------------------------------------------------------
//`define SYS_DEBUG 0
`define SYS_RESET 0
//调试模式 SYS_DEBUG -> 调整系统周期
```

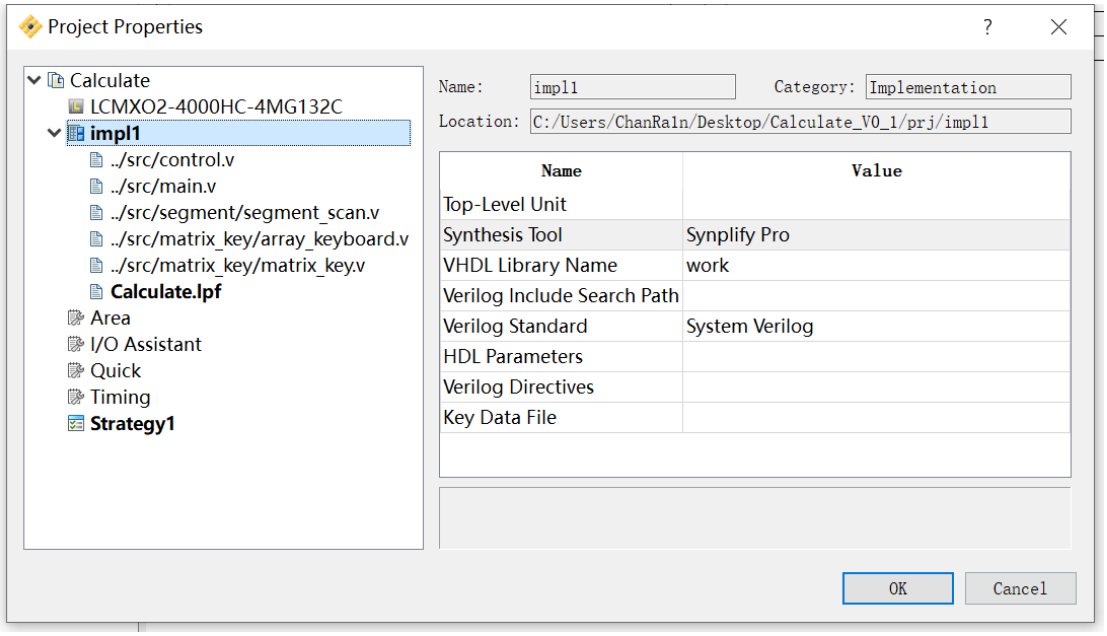
```
//复位选项 SYS_RESET -> 可选关闭全局复位，减少面积
module main #(
`ifdef SYS_DEBUG
    parameter SYS_CLK = 1000
`else
    parameter SYS_CLK = 10_000_000
`endif
) (
    input  wire i_sys_clk,
    input  wire i_sys_rst_n,
    //矩阵键盘
    input  wire [3:0] i_key_col,
    output  wire [3:0] o_key_row,
    //数码管
    output wire o_seg_rck,
    output wire o_seg_sck,
    output wire o_seg_din
);

  //矩阵键盘驱动模块 matrix_key
  //输入 WIRE 直接连接至矩阵键盘
  //输出数据为被按下的按键编号 key_data，经过功能的重新映射，其中数字 0-9 对应
数字 0-9
  //10->. 11->= 12->+ 13->- 14->× 15->÷
  wire [3:0] key_data;
  wire [0:0] key_data_en;
  matrix_key #(
      .SYS_CLK(SYS_CLK)
  ) u1_matrix_key (
      .clk(i_sys_clk),
`ifdef SYS_RESET
      .rst_n(i_sys_rst_n),
`endif
      .col(i_key_col),
      .row(o_key_row),
      .key_data(key_data),
      .key_data_en(key_data_en)
  );

  //数码管驱动模块
  //输入 4 个 REG 数据，分别是 seg_data_1 到 seg_data_4,对应显示从右往左的四个
数码管
```

```
  //其中数字 0-9 对应数字 0-9，10->. 11->= 12->+ 13->- 14->× 15->÷
  //输出 WIRE 直接连接至数码管
  wire [4:0] seg_data_1;
  wire [4:0] seg_data_2;
  wire [4:0] seg_data_3;
  wire [4:0] seg_data_4;
  wire [4:0] seg_data_5;
  wire [4:0] seg_data_6;
  wire [4:0] seg_data_7;
  wire [4:0] seg_data_8;
  wire [7:0] seg_data_en;
  segment_scan #(
      .SYS_CLK(SYS_CLK)
  ) u2_segment_scan (
      .clk   (i_sys_clk),
`ifdef SYS_RESET
      .rst_n (i_sys_rst_n),
`endif
      .dat_1 (seg_data_1),
      .dat_2 (seg_data_2),
      .dat_3 (seg_data_3),
      .dat_4 (seg_data_4),
      .dat_5 (seg_data_5),
      .dat_6 (seg_data_6),
      .dat_7 (seg_data_7),
      .dat_8 (seg_data_8),
      .dat_en(seg_data_en),
      .dot_en(8'b0000_0000),

      .seg_rck(o_seg_rck),
      .seg_sck(o_seg_sck),
      .seg_din(o_seg_din)
  );

  //计算器逻辑控制模块
  //输入矩阵键盘传递的按键编号 key_data
  //输出 4 个 REG 数据，送入数码管驱动模块
  //复位情况下数码管显示 MYFPGA.CN，当任意按键被按下，进入计算器模式。
  control #(
      .SYS_CLK(10_000_000)
  ) u_control (
      .clk        (i_sys_clk),
```

```
    .rst_n      (i_sys_rst_n),
    .key_data   (key_data),
    .key_data_en(key_data_en),

    .seg_data_1(seg_data_1),
    .seg_data_2(seg_data_2),
    .seg_data_3(seg_data_3),
    .seg_data_4(seg_data_4),
    .seg_data_5(seg_data_5),
    .seg_data_6(seg_data_6),
    .seg_data_7(seg_data_7),
    .seg_data_8(seg_data_8),
    .seg_dat_en(seg_data_en)
);

endmodule
```

2.2.2 计算模块

由于 Diamond 默认的编译器对乘法和除法的支持性不高，所以需要更换使用 Synplify Pro 进行编译。



计算模块主要起到数据流控制的作用，其软件需求说明书设计如下：

```
// 软件需求规格说明书
// 1. 简介
// 本文档旨在详细说明数码管显示控制模块的软件需求，以便于后续设计和实现这一功能。
// 2. 功能性需求
// 2.1 系统初始化
// 软件行为描述：当系统复位后，应在数码管 seg_data_1 至 seg_data_8 分别显示自 'd16 至 'd23 的值。
// 2.2 数字键输入
// 软件行为描述：当按下数字键时，将该数字显示在数码管的个位(seg_data_8)。如果连续按下数字键，则先前的数字后移变为十位(seg_data_7)，新按下的数字保持在个位。仅支持显示最大为两位数。
// 2.3 符号键输入
// 软件行为描述：当按下符号键('+', '-', '×', '÷')时，将相应符号显示在数码管的个位(seg_data_8)。之后按下的数字键行为与 2.2 节的描述一致。
// 2.4 等号键输入及运算
// 软件行为描述：当按下等于号('=')键后，系统应根据先前输入的数字和符号键进行整数运算，并将结果显示在数码管上。如果结果超出可显示范围，则需要定义如何处理（例如报错、截断或滚动显示）。
// 3. 输入需求
// 按键值定义：数字键 0-9 应对应数字 0-9，符号键应对应以下值：
// 10 - 小数点('.')
// 11 - 等于号('=')
// 12 - 加号('+')
// 13 - 减号('-')
// 14 - 乘号('×')
// 15 - 除号('÷')
// 4. 输出需求
// 数码管显示：数码管由 seg_data_1 至 seg_data_8 表示，其中 seg_data_8 是个位显示，seg_data_7 是十位显示。seg_dat_en 用于控制各个数码管位的显示使能。
// 5. 附加需求
// 按键处理：不需要定义按键抖动处理机制来确保输入的稳定性，输入的按键信号已经经过了消抖。
// 6. 流程描述
// 程序功能描述：程序复位后，在数码管的 seg_data_1~seg_data_8 显示'd16~'d23
// 当任意数字按键被按下，在数码管的个位显示被按下的按键值。接下来若任意数字按键被按下，则之前的数字变成十位，新按下的按键变成个位。最大支持 2 位数。
// 当符号按键被按下后，在数码管的个位显示被按下的符号按键
```

```
// 接下来可以按下任意数字按键，在数码管的个位显示被按下的按键值。接下来若任意数字按键被按下，则之前的数字变成十位，新按下的按键变成个位。最大支持 2 位数。
// 当等于号被按下后，程序会根据输入的数字和符号按键进行计算，并在将结果显示到数码管上。
// 计算仅保留整数即可，不需要计算小数。
```

基于此，设计了程序源码如下：

```
//------------------------------------------------------
// File Name        : control.v
// Author           : ChanRa1n
// Description      : 2024 电子森林一起练活动代码 简易计算器
// Called by        : Topmodule
// Revision History : 2024-01-06
// Revision         : 1.0
// Email            : chenyu@myfpga.cn
// Copyright(c) 2018-Now, MYFPGA.CN, All right reserved.
//------------------------------------------------------

module control #(
    parameter SYS_CLK = 10_000_000
) (
    input wire clk,
    input wire rst_n,

    input wire [3:0] key_data,  // 被按下的按键值
    input wire key_data_en,  // 按键值有效位

    output reg [4:0] seg_data_1,  // 数码管第 1 位要显示的值
    output reg [4:0] seg_data_2,  // 数码管第 2 位要显示的值
    output reg [4:0] seg_data_3,  // 数码管第 3 位要显示的值
    output reg [4:0] seg_data_4,  // 数码管第 4 位要显示的值
    output reg [4:0] seg_data_5,  // 数码管第 5 位要显示的值
    output reg [4:0] seg_data_6,  // 数码管第 6 位要显示的值
    output reg [4:0] seg_data_7,  // 数码管第 7 位要显示的值
    output reg [4:0] seg_data_8,  // 数码管第 8 位要显示的值
    output reg [7:0] seg_dat_en   // 数码管要显示值的有效位
);

  localparam [2:0] ST_IDLE = 'd0,  // Idle state
  ST_OPERAND1 = 'd1,  // Entering first operand
  ST_OPERATOR = 'd2,  // Entering operator
  ST_OPERAND2 = 'd3,  // Entering second operand
```

```
ST_CALCULATE = 'd4;  // Perform calculation

reg [2:0] state;  // Current state
reg [15:0] operand1, operand2;  // Operands
reg [ 3:0] operator;  // Operator
reg [15:0] result;  // Calculation result

// State machine
always @(posedge clk or negedge rst_n) begin
  if (~rst_n) begin
    // Reset logic
    state <= ST_IDLE;
    operand1 <= 'd0;
    operand2 <= 'd0;
    operator <= 'd0;
    result <= 'd0;
  end else begin
    case (state)
      ST_IDLE: begin
        if (!key_data_en || key_data >= 10) begin  //初始化
          operand1 <= 'd0;
          operand2 <= 'd0;
          operator <= 'd0;
          result   <= 'd0;
        end else begin
          operand1 <= key_data;
          state <= ST_OPERAND1;
        end
      end

      ST_OPERAND1: begin  //输入操作数 1
        if (key_data_en) begin
          if (key_data < 10) begin
            operand1 <= operand1 * 10 + key_data;
          end else begin
            operator <= key_data;
            state <= ST_OPERATOR;
          end
        end
      end

      ST_OPERATOR: begin  //输入操作符
```

```
          if (key_data_en && (key_data < 10)) begin
            operand2 <= key_data;
            state <= ST_OPERAND2;
          end
        end

        ST_OPERAND2: begin  //输入操作数 2
          if (key_data_en) begin
            if (key_data < 10) begin
              operand2 <= operand2 * 10 + key_data;
            end else state <= ST_CALCULATE;
          end
        end

        ST_CALCULATE: begin  //计算
          case (operator)
            11: result <= operand1 + operand2;
            13: result <= operand1 - operand2;
            14: result <= operand1 * operand2;
            15: result <= operand1[6:0] / operand2[6:0];  // prevent
divide by 0
            default: result <= 0;
          endcase

          if (key_data_en) state <= ST_IDLE;  // Any key press resets
the calculator
        end
      endcase
    end
  end

  // 重置数码管显示的过程
  task reset_display;
    begin
      seg_data_1 <= 'd16;
      seg_data_2 <= 'd17;
      seg_data_3 <= 'd18;
      seg_data_4 <= 'd19;
      seg_data_5 <= 'd20;
      seg_data_6 <= 'd21;
      seg_data_7 <= 'd22;
      seg_data_8 <= 'd23;
```

```
      seg_dat_en <= 8'b11111111;  // 启用所有数码管显示
    end
  endtask

  // State machine SEG
  always @(posedge clk or negedge rst_n) begin
    if (~rst_n) begin
      reset_display();
    end else begin
      case (state)
        ST_IDLE: begin
          reset_display();
        end
        ST_OPERAND1: begin  //显示操作数 1
          seg_data_8 <= operand1 % 10;
          seg_data_7 <= operand1 / 10;
          seg_dat_en <= 8'b0000_0011;
        end
        ST_OPERATOR: begin  //显示操作符
          seg_data_8 <= key_data;
          seg_dat_en <= 8'b0000_0001;
        end

        ST_OPERAND2: begin  //显示操作数 2
          seg_data_8 <= operand2 % 10;
          seg_data_7 <= operand2 / 10;
          seg_dat_en <= 8'b0000_0011;
        end

        ST_CALCULATE: begin  //计算
          seg_data_8 <= result % 10;
          seg_data_7 <= result / 10 % 10;
          seg_data_6 <= result / 100 % 10;
          seg_data_5 <= result / 1000 % 10;
          seg_dat_en <= 8'b0000_1111;
        end
      endcase
    end
  end
endmodule
```

2.2.3 数码管驱动模块

模块基于 Verilog HDL 编写，主要用于定义和控制数码管的显示内容和显示方式。尤其是通过 SEG1 到 SEG8 的数据输入和显示使能，可以灵活地显示各种数字或图案。

在模块中，通过定义了一个本地参数 SYS_CLK，用于设定系统的时钟频率。该模块中的计数器将对给定的时钟信号进行计数，根据计数器计数的周期产生分频的脉冲信号。

源码如下：

```
// ------------------------------------------------------------------------
// >>>>>>>>>>>>>>>>>>>>>>>> COPYRIGHT NOTICE <<<<<<<<<<<<<<<<<<<<<<<<
// ------------------------------------------------------------------------
// Module: segment_scan
//
// Author: Step
//
// Description: Display with Segment tube
//
// Web: www.stepfpga.com
//
// ------------------------------------------------------------------------
// Code Revision History :
// ------------------------------------------------------------------------
// Version: |Mod. Date:   |Changes Made:
// V1.0     |2015/11/11   |Initial ver
// V1.1     |2024/03/06   |修改显示内容
// ------------------------------------------------------------------------
module segment_scan #(
    parameter SYS_CLK = 10_000_000
) (
    input    clk,   //系统时钟 12MHz
    input    rst_n,  //系统复位 低有效
    input  [4:0] dat_1,  //SEG1 显示的数据输入
    input  [4:0] dat_2,  //SEG2 显示的数据输入
    input  [4:0] dat_3,  //SEG3 显示的数据输入
```

```
    input  [4:0] dat_4,  //SEG4 显示的数据输入
    input  [4:0] dat_5,  //SEG5 显示的数据输入
    input  [4:0] dat_6,  //SEG6 显示的数据输入
    input  [4:0] dat_7,  //SEG7 显示的数据输入
    input  [4:0] dat_8,  //SEG8 显示的数据输入
    input  [7:0] dat_en,  //数码管数据位显示使能，[MSB~LSB]=[SEG1~SEG8]
    input  [7:0] dot_en,  //数码管小数点位显示使能，[MSB~LSB]=[SEG1~SEG8]
    output reg seg_rck,  //74HC595 的 RCK 管脚
    output reg seg_sck,  //74HC595 的 SCK 管脚
    output reg seg_din  //74HC595 的 SER 管脚
);

  localparam CNT_40KHz = SYS_CLK / 40_000;  //分频系数

  localparam IDLE = 3'd0;
  localparam MAIN = 3'd1;
  localparam WRITE = 3'd2;
  localparam LOW = 1'b0;
  localparam HIGH = 1'b1;

  //创建数码管的字库，字库数据依段码顺序有关
  //这里字库数据[MSB~LSB]={G,F,E,D,C,B,A}
  reg [6:0] seg[23:0];
  always @(negedge rst_n) begin
    seg[0]  = 7'h3f;  // 0
    seg[1]  = 7'h06;  // 1
    seg[2]  = 7'h5b;  // 2
    seg[3]  = 7'h4f;  // 3
    seg[4]  = 7'h66;  // 4
    seg[5]  = 7'h6d;  // 5
    seg[6]  = 7'h7d;  // 6
    seg[7]  = 7'h07;  // 7
    seg[8]  = 7'h7f;  // 8
    seg[9]  = 7'h6f;  // 9
    seg[10] = 7'h7E;  // . (小数点)
    seg[11] = 7'h3E;  // = (等于)
    seg[12] = 7'h38;  // + (加号)
    seg[13] = 7'h0F;  // - (减号)
    seg[14] = 7'h39;  // × (乘号)
    seg[15] = 7'h0E;  // ÷ (除号)

    //LOGO HERE
```

```
    seg[16] = 7'h37;  // M
    seg[17] = 7'hEE;  // Y
    seg[18] = 7'h71;  // F
    seg[19] = 7'h73;  // P
    seg[20] = 7'h7d;  // G
    seg[21] = 7'h77;  // A.
    seg[22] = 7'h39;  // C
    seg[23] = 7'h37;  // N
  end

  //计数器对系统时钟信号进行计数
  reg [9:0] cnt = 1'b0;
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) cnt <= 1'b0;
    else if (cnt >= (CNT_40KHz - 1)) cnt <= 1'b0;
    else cnt <= cnt + 1'b1;
  end

  //根据计数器计数的周期产生分频的脉冲信号
  reg clk_40khz = 1'b0;
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) clk_40khz <= 1'b0;
    else if (cnt < (CNT_40KHz >> 1)) clk_40khz <= 1'b0;
    else clk_40khz <= 1'b1;
  end

  //使用状态机完成数码管的扫描和 74HC595 时序的实现
  reg [15:0] data;
  reg [ 2:0] cnt_main;
  reg [ 5:0] cnt_write;
  reg [ 2:0] state = IDLE;
  always @(posedge clk_40khz or negedge rst_n) begin
    if (!rst_n) begin  //复位状态下，各寄存器置初值
      state <= IDLE;
      cnt_main <= 3'd0;
      cnt_write <= 6'd0;
      seg_din <= 1'b0;
      seg_sck <= LOW;
      seg_rck <= LOW;
    end else begin
      case (state)
        IDLE: begin  //IDLE 作为第一个状态，相当于软复位
```

```
          state <= MAIN;
          cnt_main <= 3'd0;
          cnt_write <= 6'd0;
          seg_din <= 1'b0;
          seg_sck <= LOW;
          seg_rck <= LOW;
        end
        MAIN: begin
          cnt_main <= cnt_main + 1'b1;
          state <= WRITE;   //在配置完发给 74HC595 的数据同时跳转至 WRITE 状
态，完成串行时序
          case (cnt_main)
            //对 8 位数码管逐位扫描
            //data          [15:8]为段选，          [7:0]为位选
            3'd0: data <= {{dot_en[7], seg[dat_1]}, dat_en[7] ? 8'hfe :
8'hff};
            3'd1: data <= {{dot_en[6], seg[dat_2]}, dat_en[6] ? 8'hfd :
8'hff};
            3'd2: data <= {{dot_en[5], seg[dat_3]}, dat_en[5] ? 8'hfb :
8'hff};
            3'd3: data <= {{dot_en[4], seg[dat_4]}, dat_en[4] ? 8'hf7 :
8'hff};
            3'd4: data <= {{dot_en[3], seg[dat_5]}, dat_en[3] ? 8'hef :
8'hff};
            3'd5: data <= {{dot_en[2], seg[dat_6]}, dat_en[2] ? 8'hdf :
8'hff};
            3'd6: data <= {{dot_en[1], seg[dat_7]}, dat_en[1] ? 8'hbf :
8'hff};
            3'd7: data <= {{dot_en[0], seg[dat_8]}, dat_en[0] ? 8'h7f :
8'hff};
            default: data <= {8'h00, 8'hff};
          endcase
        end
        WRITE: begin
          if (cnt_write >= 6'd33) cnt_write <= 1'b0;
          else cnt_write <= cnt_write + 1'b1;
          case (cnt_write)
            //74HC595 是串行转并行的芯片，3 路输入可产生 8 路输出，而且可以级联
使用
            //74HC595 的时序实现，参考 74HC595 的芯片手册
            6'd0: begin
              seg_sck <= LOW;
```

```
            seg_din <= data[15];
        end  //SCK 下降沿时 SER 更新数据
        6'd1: begin
            seg_sck <= HIGH;
        end  //SCK 上升沿时 SER 数据稳定
        6'd2: begin
            seg_sck <= LOW;
            seg_din <= data[14];
        end
        6'd3: begin
            seg_sck <= HIGH;
        end
        6'd4: begin
            seg_sck <= LOW;
            seg_din <= data[13];
        end
        6'd5: begin
            seg_sck <= HIGH;
        end
        6'd6: begin
            seg_sck <= LOW;
            seg_din <= data[12];
        end
        6'd7: begin
            seg_sck <= HIGH;
        end
        6'd8: begin
            seg_sck <= LOW;
            seg_din <= data[11];
        end
        6'd9: begin
            seg_sck <= HIGH;
        end
        6'd10: begin
            seg_sck <= LOW;
            seg_din <= data[10];
        end
        6'd11: begin
            seg_sck <= HIGH;
        end
        6'd12: begin
            seg_sck <= LOW;
```

```
            seg_din <= data[9];
        end
        6'd13: begin
            seg_sck <= HIGH;
        end
        6'd14: begin
            seg_sck <= LOW;
            seg_din <= data[8];
        end
        6'd15: begin
            seg_sck <= HIGH;
        end
        6'd16: begin
            seg_sck <= LOW;
            seg_din <= data[7];
        end
        6'd17: begin
            seg_sck <= HIGH;
        end
        6'd18: begin
            seg_sck <= LOW;
            seg_din <= data[6];
        end
        6'd19: begin
            seg_sck <= HIGH;
        end
        6'd20: begin
            seg_sck <= LOW;
            seg_din <= data[5];
        end
        6'd21: begin
            seg_sck <= HIGH;
        end
        6'd22: begin
            seg_sck <= LOW;
            seg_din <= data[4];
        end
        6'd23: begin
            seg_sck <= HIGH;
        end
        6'd24: begin
            seg_sck <= LOW;
```

```
            seg_din <= data[3];
          end
          6'd25: begin
            seg_sck <= HIGH;
          end
          6'd26: begin
            seg_sck <= LOW;
            seg_din <= data[2];
          end
          6'd27: begin
            seg_sck <= HIGH;
          end
          6'd28: begin
            seg_sck <= LOW;
            seg_din <= data[1];
          end
          6'd29: begin
            seg_sck <= HIGH;
          end
          6'd30: begin
            seg_sck <= LOW;
            seg_din <= data[0];
          end
          6'd31: begin
            seg_sck <= HIGH;
          end
          6'd32: begin
            seg_rck <= HIGH;
          end  //当 16 位数据传送完成后 RCK 拉高，输出生效
          6'd33: begin
            seg_rck <= LOW;
            state   <= MAIN;
          end
          default: ;
        endcase
      end
      default: state <= IDLE;
    endcase
  end
end

endmodule
```

2.2.4 按键映射模块

为了将按键进行编码，我修改了 key_decode.v 模块，以实现操作数和操作符的分离，同时也方便后续进行升级改造，如加入 Fn 按键，以实现更多运算等。

图 3.2 矩阵键盘按键分配图

代码修改如下：

```
//------------------------------------------------------------
// File Name         : matrix_key.v
// Author            : ChanRa1n
// Description       : 2024 电子森林一起练活动代码
// Called by         : Topmodule
// Revision History  : 2024-01-06
// Revision          : 1.0
// Email             : chenyu@myfpga.cn
// Copyright(c) 2018-Now, MYFPGA.CN, All right reserved.
//------------------------------------------------------------
module matrix_key #(
    parameter SYS_CLK = 10_000_000
) (
    input clk,
```

```
    input rst_n,
    input [3:0] col,
    output [3:0] row,
    output reg [3:0] key_data,
    output reg key_data_en
);

  wire [15:0] key_out;
  wire [15:0] key_pulse;

  array_keyboard #(
      .CNT_200HZ(SYS_CLK / 200)
  ) u_array_keyboard (
      .clk  (clk),
      .rst_n(rst_n),
      .col  (col),

      .row      (row),
      .key_out  (key_out),
      .key_pulse(key_pulse)
  );

  always @(posedge clk or negedge rst_n) begin
    if (~rst_n) begin
      key_data_en <= 1'b0;
      key_data <= 4'd0;
    end else begin
      if (key_pulse) begin
        key_data_en <= 1'b1;
        case (key_pulse)
          16'b0000_0000_0000_0001:  key_data <= 4'd7;
          16'b0000_0000_0000_0010:  key_data <= 4'd8;
          16'b0000_0000_0000_0100:  key_data <= 4'd9;
          16'b0000_0000_0000_1000:  key_data <= 4'd15;
          16'b0000_0000_0001_0000:  key_data <= 4'd4;
          16'b0000_0000_0010_0000:  key_data <= 4'd5;
          16'b0000_0000_0100_0000:  key_data <= 4'd6;
          16'b0000_0000_1000_0000:  key_data <= 4'd14;
          16'b0000_0001_0000_0000:  key_data <= 4'd3;
          16'b0000_0010_0000_0000:  key_data <= 4'd2;
          16'b0000_0100_0000_0000:  key_data <= 4'd1;
          16'b0000_1000_0000_0000:  key_data <= 4'd13;
```

```
          16'b0001_0000_0000_0000:  key_data <= 4'd0;
          16'b0010_0000_0000_0000:  key_data <= 4'd9;
          16'b0100_0000_0000_0000:  key_data <= 4'd10;
          16'b1000_0000_0000_0000:  key_data <= 4'd11;
          default: key_data <= 4'd0;
        endcase
      end else begin
        key_data_en <= 1'b0;
        key_data <= 4'd0;
      end
    end
  end

endmodule
```

为了方便区分操作数和操作符，采用直接映射方法，增加了代码的易读性和可维护性。

2.2.5 矩阵键盘扫描模块

矩阵键盘的驱动文件参考自苏州硬禾信息科技有限公司的 array_keyboard.v(V1.0)，以下对该模块的驱动部分进行分析和解释说明。

该模块主要完成了三件事：

1. 频率分频器和状态机：通过计数器和状态机，生成一个 200Hz 的时钟信号 clk_200hz。状态机负责按周期扫描矩阵键盘的不同行，以检测按键的状态。
2. 矩阵键盘扫描： 在每个状态下，通过行和列的组合来扫描矩阵键盘。按键的状态通过 key_out 输出。
3. 下降沿检测：使用 key_pulse 信号来检测按键状态的下降沿。

在 key_out 从高电平切换到低电平时，key_pulse 产生一个脉冲信号。

以下是代码部分源码：

```
// ---------------------------------------------------------------------
// >>>>>>>>>>>>>>>>>>>>>>>> COPYRIGHT NOTICE <<<<<<<<<<<<<<<<<<<<<<<<
// ---------------------------------------------------------------------
// Module: array_keyboard
//
// Author: Step
//
// Description: array_keyboard
//
// Web: www.stepfapga.com
//
// ---------------------------------------------------------------------
// Code Revision History :
// ---------------------------------------------------------------------
// Version: |Mod. Date:   |Changes Made:
// V1.0     |2015/11/11   |Initial ver
// ---------------------------------------------------------------------
module array_keyboard #
(
    parameter           CNT_200HZ = 60000
)
(
    input                   clk,
    input                   rst_n,
    input           [3:0]   col,
    output reg  [3:0]   row,
    output reg  [15:0]key_out,
    output      [15:0]key_pulse
);

    localparam          STATE0 = 2'b00;
    localparam          STATE1 = 2'b01;
    localparam          STATE2 = 2'b10;
    localparam          STATE3 = 2'b11;

    /*
```

```
    因使用 4x4 矩阵按键，通过扫描方法实现，所以这里使用状态机实现，共分为 4 种状态
    在其中的某一状态时间里，对应的 4 个按键相当于独立按键，可按独立按键的周期采样法采样
    周期采样时每隔 20ms 采样一次，对应这里状态机每隔 20ms 循环一次，每个状态对应 5ms 时间
    对矩阵按键实现原理不明白的，请去了解矩阵按键实现原理
    */

    //计数器计数分频实现 5ms 周期信号 clk_200hz,系统时钟 12MHz
    reg [15:0]  cnt;
    reg             clk_200hz;
    always@(posedge clk or negedge rst_n) begin  //复位时计数器 cnt 清零，clk_200hz 信号起始电平为低电平
        if(!rst_n) begin
            cnt <= 16'd0;
            clk_200hz <= 1'b0;
        end else begin
            if(cnt >= ((CNT_200HZ>>1) - 1)) begin  //数字逻辑中右移 1 位相当于除 2
                cnt <= 16'd0;
                clk_200hz <= ~clk_200hz;  //clk_200hz 信号取反
            end else begin
                cnt <= cnt + 1'b1;
                clk_200hz <= clk_200hz;
            end
        end
    end

    reg     [1:0]       c_state;
    //状态机根据 clk_200hz 信号在 4 个状态间循环，每个状态对矩阵按键的行接口单行有效
    always@(posedge clk_200hz or negedge rst_n) begin
        if(!rst_n) begin
            c_state <= STATE0;
            row <= 4'b1110;
        end else begin
            case(c_state)
                //状态 c_state 跳转及对应状态下矩阵按键的 row 输出
                STATE0: begin c_state <= STATE1; row <= 4'b1101; end
                STATE1: begin c_state <= STATE2; row <= 4'b1011; end
                STATE2: begin c_state <= STATE3; row <= 4'b0111; end
```

```
                STATE3: begin c_state <= STATE0; row <= 4'b1110; end
                default:begin c_state <= STATE0; row <= 4'b1110; end
            endcase
        end
    end

    reg [15:0]  key,key_r;
    //因为每个状态中单行有效，通过对列接口的电平状态采样得到对应 4 个按键的状态，依次循环
    always@(negedge clk_200hz or negedge rst_n) begin
        if(!rst_n) begin
            key_out <= 16'hffff; key_r <= 16'hffff; key <= 16'hffff;
        end else begin
            case(c_state)
                //采集当前状态的列数据赋值给对应的寄存器位
                //对键盘采样数据进行判定，连续两次采样低电平判定为按键按下
                STATE0: begin key_out[ 3: 0] <= key_r[ 3: 0]|key[ 3: 0]; key_r[ 3: 0] <= key[ 3: 0]; key[ 3: 0] <= col; end
                STATE1: begin key_out[ 7: 4] <= key_r[ 7: 4]|key[ 7: 4]; key_r[ 7: 4] <= key[ 7: 4]; key[ 7: 4] <= col; end
                STATE2: begin key_out[11: 8] <= key_r[11: 8]|key[11: 8]; key_r[11: 8] <= key[11: 8]; key[11: 8] <= col; end
                STATE3: begin key_out[15:12] <= key_r[15:12]|key[15:12]; key_r[15:12] <= key[15:12]; key[15:12] <= col; end
                default:begin key_out <= 16'hffff; key_r <= 16'hffff; key <= 16'hffff; end
            endcase
        end
    end

    reg     [15:0]      key_out_r;
    //Register low_sw_r, lock low_sw to next clk
    always @ ( posedge clk  or  negedge rst_n )
        if (!rst_n) key_out_r <= 16'hffff;
        else  key_out_r <= key_out;   //将前一刻的值延迟锁存

    //wire  [15:0]       key_pulse;
    //Detect the negedge of low_sw, generate pulse
    assign key_pulse= key_out_r & ( ~key_out);   //通过前后两个时刻的值判断

endmodule
```

3 软件测试与验证

FPGA 软件类型： Ⅰ类新研；

关键性等级：民用级；

在对软件结构和成熟度分析后决定，跳过成熟的模块和 IP，对自行设计的逻辑控制、计算部分模块进行模块级仿真验证(前仿)，对集成后的整体进行功能性验证(黑盒)，最后对网表文件进行时序验证。以下是对应的详细记录。

3.1 模块化测试记录

3.2 例化调用模块测试记录

本模块仅起到例化链接的作用，无需进行仿真测试。

3.3 计算模块测试记录

本次计算模块的测试采用行为级测试方法，仅开展前仿真验证。测试激励文件如下：

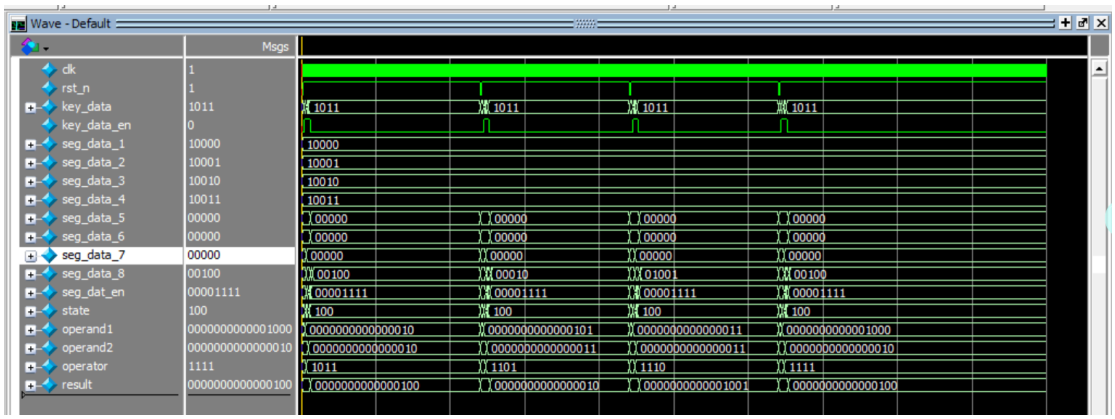
```
`timescale 1ns / 1ps
module sim_control;

reg clk;
reg rst_n;
reg [3:0] key_data;
reg key_data_en;
wire [4:0] seg_data_1, seg_data_2, seg_data_3, seg_data_4, seg_data_5,
seg_data_6,seg_data_7, seg_data_8;
wire [7:0] seg_dat_en;
```

```
always #5 clk = ~clk;

initial begin
  $dumpfile("sim_control.vcd");
  $dumpvars;
  clk = 0;
  rst_n = 0;
  key_data = 4'h0;
  key_data_en = 0;
  #10;
  rst_n = 1;
  #10;
end

control calc(
    .clk(clk),
    .rst_n(rst_n),
    .key_data(key_data),
    .key_data_en(key_data_en),
    .seg_data_1(seg_data_1),
    .seg_data_2(seg_data_2),
    .seg_data_3(seg_data_3),
    .seg_data_4(seg_data_4),
    .seg_data_5(seg_data_5),
    .seg_data_6(seg_data_6),
    .seg_data_7(seg_data_7),
    .seg_data_8(seg_data_8),
    .seg_dat_en(seg_dat_en)
);

// 测试加法:  2 + 2 = 4
initial begin
  #20;
  key_data_en = 1;
  key_data = 4'd2;
  #10;
  key_data = 4'd11;
  #10;
  key_data = 4'd2;
  #10;
  key_data = 4'd11;
  #10;
```

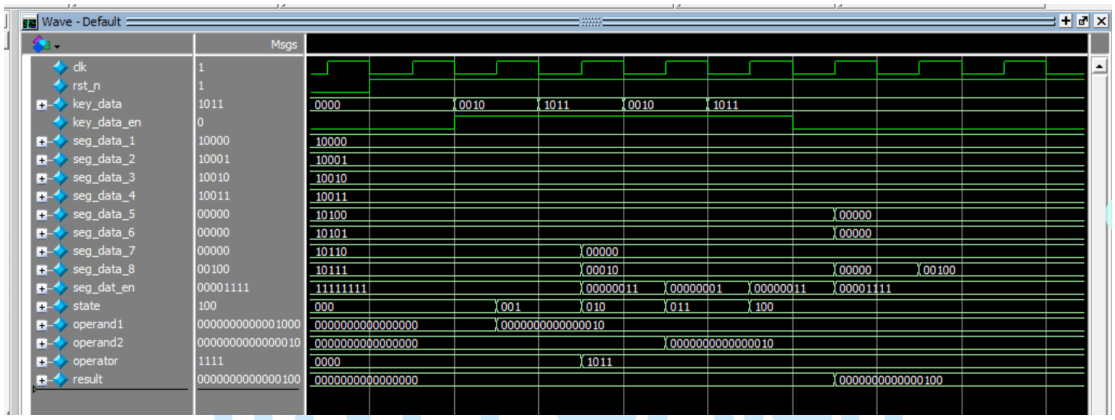
```
  key_data_en = 0;
  #10;
end

// 测试减法：  5 - 3 = 2
initial begin
  #1200;
  rst_n = 0;
  key_data = 4'h0;
  key_data_en = 0;
  #10;
  rst_n = 1;
  #10;
  key_data_en = 1;
  key_data = 4'd5;
  #10;
  key_data = 4'd13;
  #10;
  key_data = 4'd3;
  #10;
  key_data = 4'd11;
  #10;
  key_data_en = 0;
  #10;
end

// 测试乘法：  3 * 3 = 9
initial begin
  #2200;
  rst_n = 0;
  key_data = 4'h0;
  key_data_en = 0;
  #10;
  rst_n = 1;
  #10;
  key_data_en = 1;
  key_data = 4'd3;
  #10;
  key_data = 4'd14;
  #10;
  key_data = 4'd3;
  #10;
```

```
  key_data = 4'd11;
  #10;
  key_data_en = 0;
  #10;
end

// 测试除法: 8 / 2 = 4
initial begin
  #3200;
  rst_n = 0;
  key_data = 4'h0;
  key_data_en = 0;
  #10;
  rst_n = 1;
  #10;
  key_data_en = 1;
  key_data = 4'd8;
  #10;
  key_data = 4'd15;
  #10;
  key_data = 4'd2;
  #10;
  key_data = 4'd11;
  #10;
  key_data_en = 0;
end

initial begin
  #5000;
  $stop;
end

endmodule
```

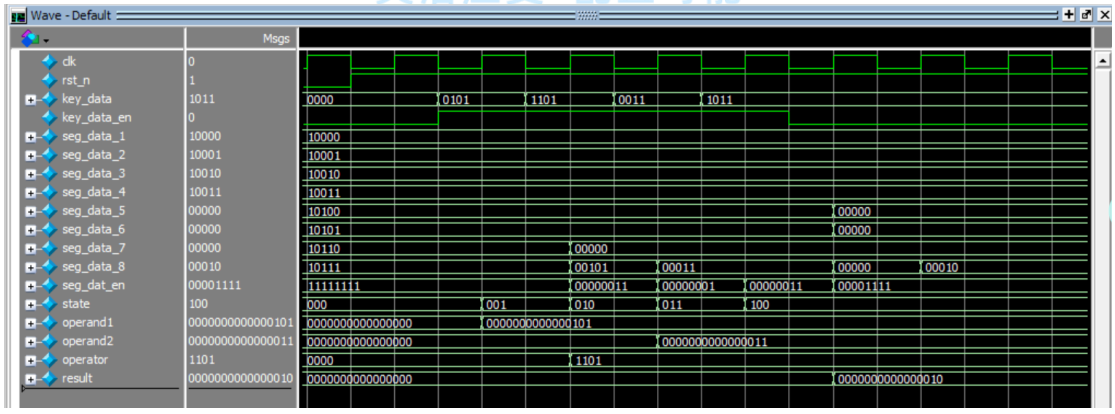
测试整体波形如下：



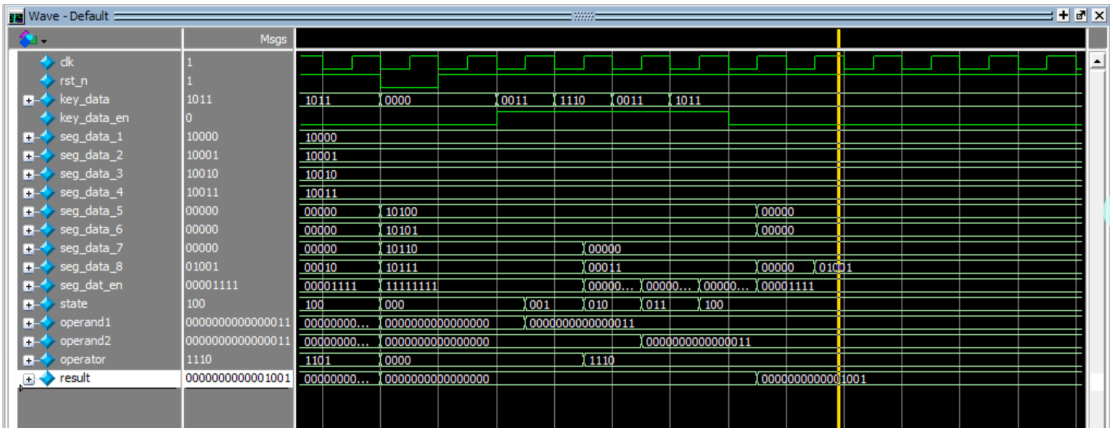
分功能进行详细测试：



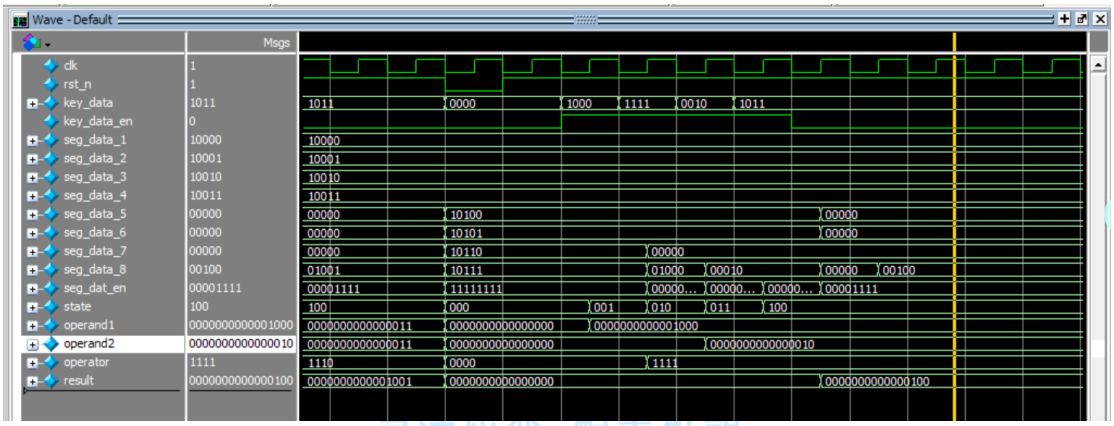
加法计算 2+2 结果 result 为 4，对应数码管位置显示也正常。



减法计算 5-3 结果 result 为 2，对应数码管位置显示也正常。



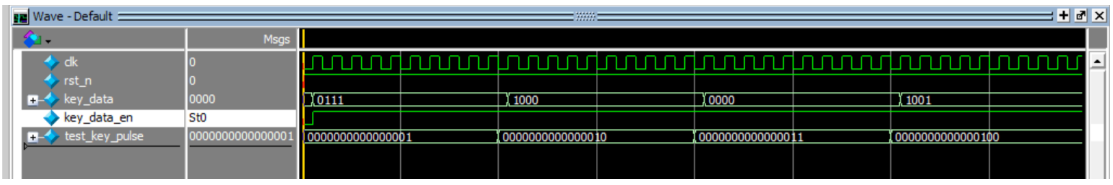
乘法计算 3*3 结果 result 为 9，对应数码管位置显示也正常。



除法计算 8/2 结果 result 为 4，对应数码管位置显示也正常。

3.4 按键映射模块测试记录

按键映射模块仅起到映射链接的作用，没有额外的逻辑，进行行为级简单仿真测试，记录如下：



经过确认，映射功能正确。

3.5 数码管驱动模块测试记录

数码管驱动模块为电子森林设计成熟模块，经过实物测试功能满足要求。

3.6 矩阵键盘扫描模块测试记录

矩阵键盘扫描模块为电子森林设计成熟模块，经过实物测试功能满足要求。

3.7 资源占用情况

3.7.1 编译后占用情况

```
Design Summary
   Number of registers:    213 out of  4635 (5%)
      PFU registers:          202 out of  4320 (5%)
      PIO registers:           11 out of   315 (3%)
   Number of SLICEs:       722 out of  2160 (33%)
      SLICEs as Logic/ROM:    722 out of  2160 (33%)
      SLICEs as RAM:            0 out of  1620 (0%)
      SLICEs as Carry:        460 out of  2160 (21%)
   Number of LUT4s:        1443 out of  4320 (33%)
      Number used as logic LUTs:        523
      Number used as distributed RAM:     0
      Number used as ripple logic:      920
      Number used as shift registers:     0
   Number of PIO sites used: 13 + 4(JTAG) out of 105 (16%)
   Number of block RAMs:  0 out of 10 (0%)
   Number of GSRs:        1 out of 1 (100%)
   EFB used :        No
   JTAG used :       No
   Readback used :   No
   Oscillator used : No
   Startup used :    No
```

```
   POR :            On
   Bandgap :        On
   Number of Power Controller:  0 out of 1 (0%)
   Number of Dynamic Bank Controller (BCINRD):  0 out of 6 (0%)
   Number of Dynamic Bank Controller (BCLVDSO):  0 out of 1 (0%)
   Number of DCCA:  0 out of 8 (0%)
   Number of DCMA:  0 out of 2 (0%)
   Number of PLLs:  0 out of 2 (0%)
   Number of DQSDLLs:  0 out of 2 (0%)
   Number of CLKDIVC:  0 out of 4 (0%)
   Number of ECLKSYNCA:  0 out of 4 (0%)
   Number of ECLKBRIDGECS:  0 out of 2 (0%)
```

3.7.2 布局布线后占用情况

```
Device utilization summary:

   PIO (prelim)   13+4(JTAG)/280     6% used
                  13+4(JTAG)/105     16% bonded
   IOLOGIC          11/280           3% used

   SLICE           722/2160         33% used

   GSR               1/1           100% used


Number of Signals: 1732
Number of Connections: 4134
```

3.7.3 资源利用说明

根据 4.2.2 布局布线后的设备使用度，我们可以对 FPGA 的资源利用情况做如下分析：

1. PIO (预设)：预先定义了 17 个预设 I/O，其中 13 个已被占用并且包括诸如 JTAG 之类的功能，另外 4 个是为 JTAG 保留的。总共占用了 280 个可用资源的 6%，其中有 16%的已经被束缚。

2. IOLOGIC 资源：总共使用了 280 个 IOLOGIC 资源中的 11 个，占用率为 3%，这代表着大部分的 IOLOGIC 资源还未使用，因此有足够的资源进行更多的操作处理。
3. 切片 SLICE: 切片是 FPGA 的基本计算单元，我们的设计中已经使用了 2160 个切片中的 722 个，占用率为 33%，这显示了仍然有很大比例的切片资源可以用于未来的扩展和优化。
4. 全局设置复位 GSR：充分利用了该资源，被使用的比例达到了 100%。这意味着我们无法添加需要 GSR 的新部分。程序中设置了宏定义，可以一键删去全局复位，这可能会降低逻辑的占用，但是 GSR 的性质，也可能不降低，这里不再进行测试。
5. 信号与连接：这个设计中有 1732 个信号和 4134 个连接。这是一个相对较大的设计，需要更多的资源和时间进行布局布线。

在该 FPGA 设计中使用了乘法和除法器，因此占用的资源相对较多。但从整体说来，这种设计仍有许多闲置的资源可以供未来的扩展和复杂运算使用。在未来的设计中，我们可以尝试使用更优化的设计，以减少资源的使用，提高设计的效率。

4 需求验证

经过需求对比和确认，本报告的全部需求均被满足。