科研训练说明书中文摘要

|  |
| --- |
| 针对计算器轻量化、便捷化的需要，设计并实现了一种基于FPGA的简便计算器。在Vivado开发环境下，采用Verilog HDL语言，使用自顶向下的方法对系统进行了描述。设计电路主要由输入模块、信号转换模块、数据记录模块、显示模块、计算模块、结果转换模块和顶层模块组成，完成源程序的编写、编译、模拟仿真，并在EGO1开发板实现程序的下载及调试。波形仿真和硬件下载验证了该设计的正确性和有效性。结果表明，基于FPGA设计的数字计算器具有稳定高效、简单可靠、安全灵活等优点。  关键词 FPGA Vivado 数字计算器 Verilog HDL |

科研训练说明书外文摘要

|  |
| --- |
| Title Design of Digital Calculator Based on FPGA  Abstract  In order to meet the needs of lightweight and convenient calculators, a simple calculator based on FPGA was designed and implemented. In the Vivado development environment, the system is described using the Verilog HDL language using a top-down approach. The design circuit is mainly composed of input module, signal conversion module, data logging module, display module, calculation module, result conversion module and top-level module, which completes the writing, compilation, simulation and simulation of the source program, and completes the download and debugging of the program on the EGO1 development board. Waveform simulation and download verification show the correctness and validity of the design. The results show that the digital calculator based on FPGA design has the advantages of stability, efficiency, simplicity and reliability, security and flexibility.  **Keywords**  FPGA Vivado Digital Calculator Verilog HDL |

目 录

[**1绪论 1**](#_Toc177224069)

[1.1课题研究的目的及研究现状 1](#_Toc177224070)

[1.2课题研究的主要工作内容 1](#_Toc177224071)

[**2相关技术软件介绍 3**](#_Toc177224072)

[2.1 FPGA简介 3](#_Toc177224073)

[2.2 Verilog HDL语言简介 3](#_Toc177224074)

[2.3 Vivado软件平台简介 4](#_Toc177224075)

[2.4 EDA技术 4](#_Toc177224076)

[**3数字计算器系统设计 6**](#_Toc177224077)

[3.1系统功能分析及预期目标 6](#_Toc177224078)

[3.2系统整体框架 6](#_Toc177224079)

[3.3输入模块 7](#_Toc177224080)

[3.3.1 输入模块的功能与设计 7](#_Toc177224081)

[3.3.2 输入模块Verilog HDL实现 7](#_Toc177224082)

[3.3.3 输入模块仿真 9](#_Toc177224083)

[3.4 信号转换模块 9](#_Toc177224084)

[3.4.1 信号转换模块的功能与设计 9](#_Toc177224085)

[3.4.2 信号转换模块仿真 11](#_Toc177224086)

[3.5 数据记录模块 11](#_Toc177224087)

[3.5.1 数据记录模块的功能与设计 11](#_Toc177224088)

[3.5.2 数据记录模块的仿真 13](#_Toc177224089)

[3.6 显示模块 13](#_Toc177224090)

[3.6.1 显示模块的功能与设计 13](#_Toc177224091)

[3.6.2 显示模块的仿真 15](#_Toc177224092)

[3.7 计算模块 16](#_Toc177224093)

[3.7.1 计算模块的功能与设计 16](#_Toc177224094)

[3.7.2 计算模块的仿真 19](#_Toc177224095)

[3.8 结果转换模块 20](#_Toc177224096)

[3.8.1 结果转换模块的功能与设计 20](#_Toc177224097)

[3.8.2 结果转换模块的仿真 22](#_Toc177224098)

[3.9 顶层模块 23](#_Toc177224099)

[3.9.1顶层模块的功能与设计 23](#_Toc177224100)

[**4 开发板调试 26**](#_Toc177224101)

[4.1 管脚分配 26](#_Toc177224102)

[4.2 下载步骤 27](#_Toc177224103)

[4.3 调试结果 27](#_Toc177224104)

[**结论 31**](#_Toc177224105)

[**致谢 32**](#_Toc177224106)

[**参考文献 33**](#_Toc177224107)

**1绪论**

**1.1课题研究的目的及研究现状**

计算器是一种可以进行数字运算的机器，它为人们的学习、工作及生活提供了很大的便利，是办公应用不可缺少的器件。现代的电子计算器是一种能进行数学运算的手持电子机器，拥有集成电路芯片，但结构比电脑简单得多，可以说是第一代的电子计算机，可广泛运用于商业交易中，是必备的办公用品之一。数字计算器在集成电路发展以后，其性能发生了质的飞跃，计算器已经脱离了曾经的“辅助计算工具”的定位，正朝着多功能化、可编程化的方向发展，可以说计算器就是一个“微微型”的计算机。在日常生活中，大部分普通的计算器只能完成简单的加减乘除等计算功能，这使得计算器的应用场景受到限制。在快速发展的社会下，人们对于计算器的功能也提出了更高的要求。因此，设计出一个功能多样，适用场景广泛的计算器显得尤为重要。

以前的数字计算器使用单片机组成，单片机具有较低的功耗和成本，可以应用在功能较为简单的数字计算器上，但是由单片机组成的数字计算器无法满足日益复杂与高效的计算需求。[1]而FPGA具备可编程灵活性高、开发周期短以及并行计算效率高等特点，逐渐在生活中广泛应用。因此，可以利用FPGA设计一款规模更大、性能更高、速度更快、能够满足更为复杂计算需求的可定制化数字计算器。[2]

**1.2课题研究的主要工作内容**

本课题主要利用Vivado软件以及FPGA芯片设计一个数字计算器。用户利用4×4矩阵键盘完成运算数和运算符号的输入，借此实现加减乘除三角函数值等算术功能，并将输入的运算数据、运算符号、运算结果等显示在数码管上。在完成电路的设计后，编写仿真文件，进行功能仿真，同时使用开发环境中的综合优化功能及编程下载功能验证系统设计的正确性。

本论文完成的主要工作体现在以下几个方面：

1. 数字计算器的总体方案设计、功能描述；
2. 采用Verilog HDL进行系统设计、逻辑描述；
3. 依靠开发环境的仿真功能，进行仿真验证；
4. 完成电路的综合、执行，生成比特流文件，实现上板验证。

**2相关技术软件介绍**

**2.1 FPGA简介**

FPGA（Field Programmable Gate Array，现场可编程门阵列）是在PAL （可编程阵列逻辑）、GAL（通用阵列逻辑）等可编程器件的基础上进一步发展的产物。它是作为专用集成电路（ASIC）领域中的一种半定制电路而出现的，既解决了定制电路的不足，又克服了原有可编程器件门电路数有限的缺点。FPGA可以实现极其复杂的时序与组合逻辑电路功能，可以适用于高速、高密度等具有复杂的算法逻辑和信号处理单元的逻辑设计领域。[2]可以预见未来FPGA技术必将呈现出巨大的创新和进步，而基于FPGA的数字计算器也必将被更多人看重，成为主流的计算器芯片。[3]

**2.2 Verilog HDL语言简介**

Verilog HDL是一种硬件描述语言，用于从算法级、门级到开关级的多种抽象设计层次的数字系统建模。被建模的数字系统对象的复杂性可以介于简单的门和完整的电子数字系统之间。数字系统能够按层次描述，并可在相同描述中显式地进行时序建模。

Verilog HDL 语言具有下述描述能力：设计的行为特性、设计的数据流特性、设计的结构组成以及包含响应监控和设计验证方面的时延和波形产生机制。所有这些都使用同一种建模语言。此外，Verilog HDL语言提供了编程语言接口，通过该接口可以在模拟、验证期间从设计外部访问设计，包括模拟的具体控制和运行。

Verilog HDL语言不仅定义了语法，而且对每个语法结构都定义了清晰的模拟、仿真语义。因此，用这种语言编写的模型能够使用Verilog仿真器进行验证。语言从C编程语言中继承了多种操作符和结构。Verilog HDL提供了扩展的建模能力，其中许多扩展最初很难理解。但是，Verilog HDL语言的核心子集非常易于学习和使用，这对大多数建模应用来说已经足够。当然，完整的硬件描述语言足以对从最复杂的芯片到完整的电子系统进行描述。

**2.3 Vivado软件平台简介**

Vivado软件是Xilinx公司发布的集成设计环境，是EDA工具的一种，支持Block Design、Verilog、VHDL等多种输入方式，内部镶嵌有综合器以及仿真器，可以满足从设计输入、综合适配、仿真到下载的全部FPGA设计流程。Verilog是一种IEEE标准的硬件描述语言，适合进行大规模的FPGA设计。在编写Verilog程序时，首先要理解Verilog的可综合性，可综合性就是将寄存器传输的描述转化为门级网表的描述；其次必须用硬件电路的设计思想来编写Verilog程序，要对所描述的每个模块，每个语句对应的硬件电路有一个大致的了解，也就是在硬件电路设计的基础上完成Verilog程序的编写。

为了解决集成的瓶颈问题，Vivado 设计套件采用了用于快速综合和验证C 语言算法IP 的ESL 设计，实现重用的标准算法和RTL IP 封装技术，标准IP 封装和各类系统构建模块的系统集成，模块和系统验证的仿真速度提高了3 倍，与此同时，硬件协仿真性能提升了100倍。

为了解决实现的瓶颈，Vivado工具采用层次化器件编辑器和布局规划器、速度提升了3 至15 倍，且为SystemVerilog 提供了业界最好支持的逻辑综合工具、速度提升4 倍且确定性更高的布局布线引擎，以及通过分析技术可最小化时序、线长、路由拥堵等多个变量的“成本”函数。此外，增量式流程能让工程变更通知单(ECO) 的任何修改只需对设计的一小部分进行重新实现就能快速处理，同时确保性能不受影响。最后，Vivado 工具通过利用最新共享的可扩展数据模型，能够估算设计流程各个阶段的功耗、时序和占用面积，从而达到预先分析，进而优化自动化时钟门等集成功能。

**2.4 EDA技术**

EDA涵盖了电子设计、仿真、验证、制造全过程的所有技术，[4]诸如：系统设计与仿真，电路设计与仿真，印制电路板（PCB）设计与校验，集成电路（IC）版图设计、验证和测试，数字逻辑电路设计，模拟电路设计，数模混合设计，嵌入式系统设计，软硬件协同设计，芯片上系统（SoC）设计，可编程逻辑器件（PLD）和可编程系统芯片（SOPC）设计，专用集成电路（ASIC）和专用标准产品（ASSP）设计技术等。[5][6]高级硬件描述语言和IP芯核被广泛采用，使得电子设计方式以及电子系统的概念发生了根本性的改变。

IP是集成电路知识产权模块的简称，可以定义为“经过预先设计、预先验证，具有相对独立功能，可以重复使用在SoC和复杂ASIC中的电路模块”。按照其在设计流程中的位置，IP可分为三种：软核IP、固核IP和硬核IP。

**3数字计算器系统设计**

**3.1系统功能分析及预期目标**

本次设计主要是运用Verilog HDL语言和Vivado软件，在EGO1开发板上模拟出用户使用计算机进行相关计算。包括用户通过按键输入数字，同时在EGO1板子的码管上根据输入数字进行显示；然后客户选择计算方式，按下确定后进行计算并显示在EGO1板子的码管上。以上即为整体系统设计的思路与预期目标。

**3.2系统整体框架**

在设计功能的基础上，进行自顶向下的模块化设计。[7]从系统总体要求出发，自上而下地逐步将设计内容细化。在Vivado开发环境下，对各模块运用硬件描述语言Verilog HDL进行逻辑描述和仿真。[8]

具体包括以下模块：按键输入模块（keyboard）、信号转换模块（myencoder）、数据记录模块（shift4）、显示模块（display）、计算模块（calculate）、结果转换模块（translation）、顶层模块（top）。总体设计结构如图 3.2.1 所示。

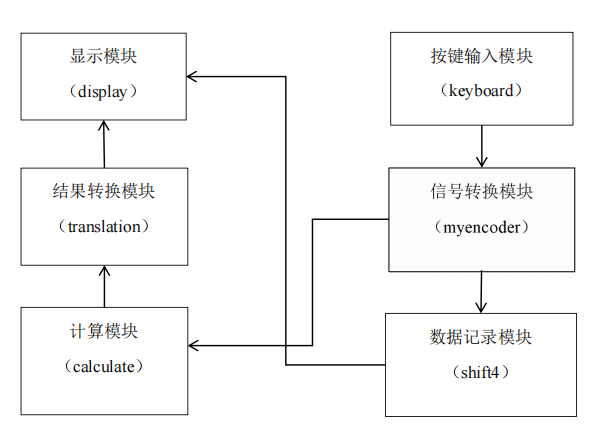


图 3.2.1 系统总体结构框架图

**3.3输入模块**

**3.3.1 输入模块的功能与设计**

本设计采用薄膜键盘进行计算数据以及计算符号的输入，输入模块的主要功能就是将键盘输入的信号转化为四位二进制码，不同的二进制码代表不同的按键信息，以便后续模块对这些二进制信息进行处理。

薄膜键盘实物图如图 3.3.1所示。

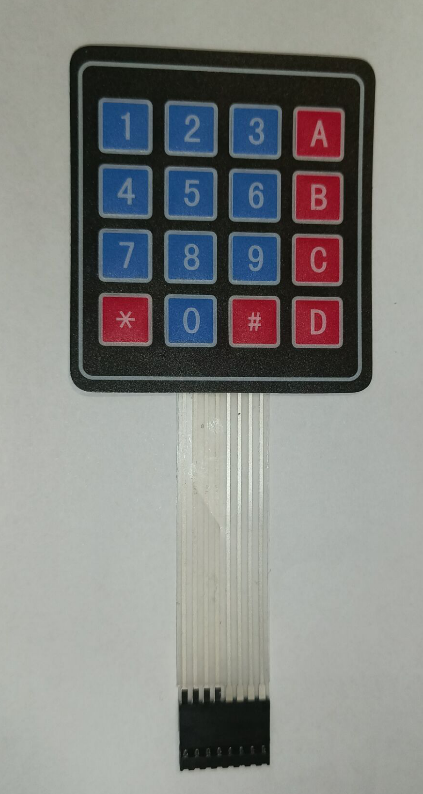


图 3.3.1 薄膜键盘

输入模块的封装图如图 3.3.2 所示。

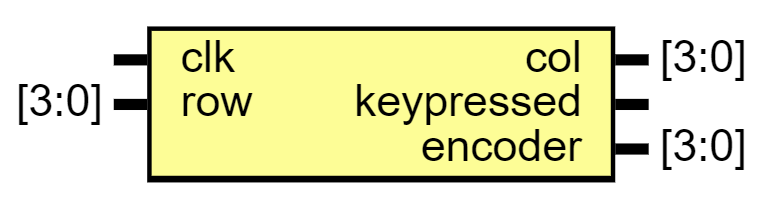


图 3.3.2 输入模块封装图

**3.3.2 输入模块Verilog HDL实现**

1. 输入信号端口

clk：系统时钟100MHz

row：键盘的行输入，一般处于上拉状态，键盘中按键被按下时输入低电平

1. 输出信号端口

col：键盘的列输出，一般处于下拉状态，正常工作时只有一列输出低电平，依次循环

keypressed：按键按下的信号，当有按键被按下时输出高电平

encoder：输出键盘的按键输入信息，不同的按键对应不同的输出编码，编码为按键所对应的行和列的序号

1. 功能：采集键盘输入信号
2. 说明：系统输入100MHz的时钟信号，经过两个分频块的处理，分出50Hz和5Hz的时钟信号。键盘扫描块以50Hz时钟作为触发信号，对一个四位二进制数count进行增计数处理，根据count[3:2]的值选择一个行，然后检查该行是否有按键被按下。如果没有按键被按下，count值加1；否则，将encoder\_r设置为count的值。接下来，根据count[1:0]的值选择一个列，并将其设置为相应的电平。有按键被按下时的count值即为要输出的按键输入信息，再将其赋值给encode输出。检测是否有按键被按下的模块以5Hz时钟作为触发信号，当有按键被按下时，keypressed输出高电平。
3. 输入模块主要代码
4. always @(posedge clk50hz)
5. begin
6. **case**(count[3:2])
7. 2'b00:
8. unkeypressed=row[0];  //矩阵键盘按键被按下时返回0，所以定义unkeypressed，即unkeypressed=1意味着没有按键按下
9. 2'b01:
10. unkeypressed=row[1];  //一般上拉
11. 2'b10:
12. unkeypressed=row[2];
13. **default**:
14. unkeypressed=row[3];
15. endcase
16. **if** (unkeypressed)
17. count=count+1;
18. **else**
19. encoder\_r=count;//
21. //键盘 列扫描，
22. **case**(count[1:0])
23. 2'b00:
24. col\_r=4'b1110;//一般下拉
25. 2'b01:
26. col\_r=4'b1101;
27. 2'b10:
28. col\_r=4'b1011;
29. **default**:
30. col\_r=4'b0111;
31. endcase
32. end
33. //检测是否有按键按下
34. always @ (posedge clk5hz)
35. begin
36. keypressed\_r<=!unkeypressed;
37. end

**3.3.3 输入模块仿真**

输入模块的仿真结果如图 3.3.3 所示。



图 3.3.3 输入模块的仿真结果

从仿真结果中可以看出，当row[0]=0时，encode高两位为“00”，col=4’b1101时，encode低两位为“01”，组合起来为“0001”，与仿真结果相同。

**3.4 信号转换模块**

**3.4.1 信号转换模块的功能与设计**

信号转换模块是将输入模块送来的按键编码值（行和列序号），按照其真正的含义进行转换，以便后续对其进行计算。

信号转换模块的封装如图 3.4.1 所示。

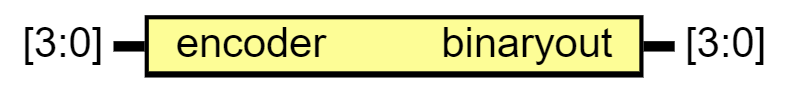


图 3.4.1 信号转换模块的封装图

该模块的编写采用了状态机，其状态转换图如图 3.4.2 所示。其中输入信号为第一个状态时，输出信号就为输入信号的下一个状态。例如信号“0001”的下一个状态为“0010”，则实际情况中，输入信号为“0001”时，输出信号就为“0010”。

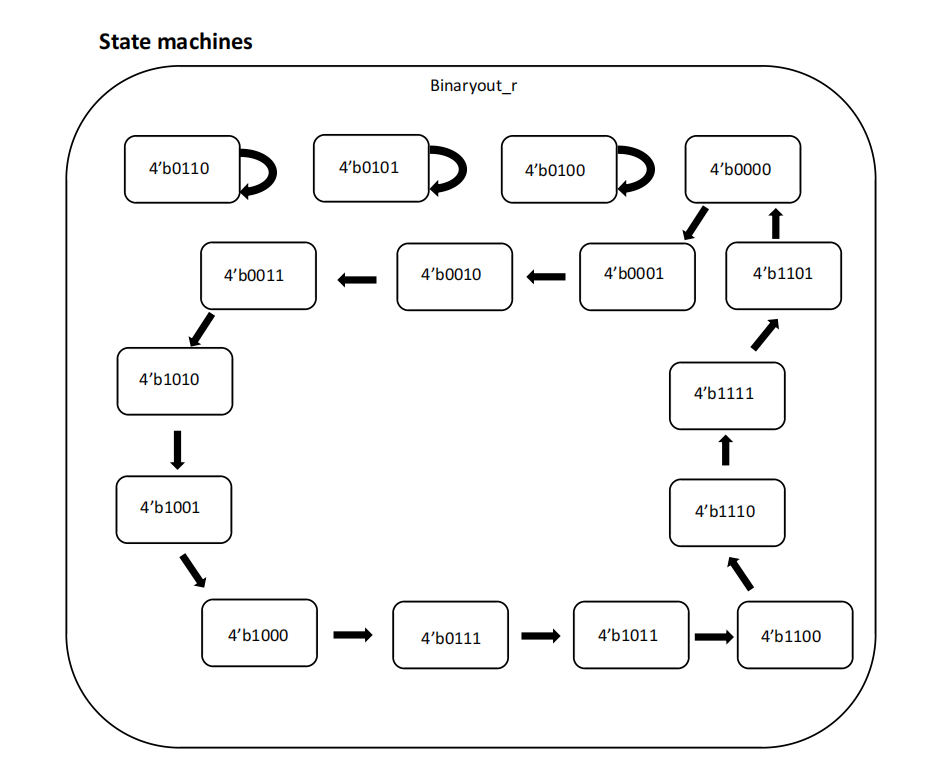


图 3.4.2 信号转换模块的状态转换图

1. 输入信号端口

encoder：输入模块送入的按键编码值

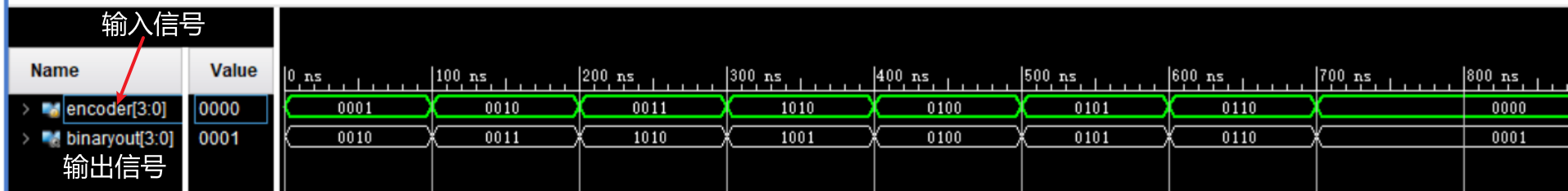
1. 输出信号端口

binaryout：对输入模块送入的按键编码值进行转换后输出结果

1. 功能：将输入模块的编码值转换为有利于计算的编码值
2. 说明：该模块主要利用状态机进行编写，将输入模块中代表按键“1”的编码转换为1的四位二进制数0001，其余数字依次类推。将键盘上的“A”键转换为4’b1010，“B”、“C”、“D”键的二进制码依次递增。键盘上的“\*”键转换为4’b1110，“#”键转换为4’b1111。
3. 信号转换模块主要代码
4. always @(\*)begin
5. **case** (encoder)
6. 4'b0000:  binaryout\_r=4'b0001;
7. 4'b0001:  binaryout\_r=4'b0010;
8. 4'b0010:  binaryout\_r=4'b0011;
9. 4'b0011:  binaryout\_r=4'b1010;//A,
10. 4'b0100:  binaryout\_r=4'b0100;
11. 4'b0101:  binaryout\_r=4'b0101;
12. 4'b0110:  binaryout\_r=4'b0110;
13. 4'b0111:  binaryout\_r=4'b1011;//B,
14. 4'b1000:  binaryout\_r=4'b0111;
15. 4'b1001:  binaryout\_r=4'b1000;
16. 4'b1010:  binaryout\_r=4'b1001;
17. 4'b1011:  binaryout\_r=4'b1100;//C,后面定义为×
18. 4'b1100:  binaryout\_r=4'b1110;//\*,定义为小数点
19. 4'b1101:  binaryout\_r=4'b0000;
20. 4'b1110:  binaryout\_r=4'b1111;//#,定义为    =
21. 4'b1111:  binaryout\_r=4'b1101;//D,定义为    ÷
22. endcase
23. end

**3.4.2 信号转换模块仿真**

信号转换模块的仿真结果如图 3.4.3 所示。



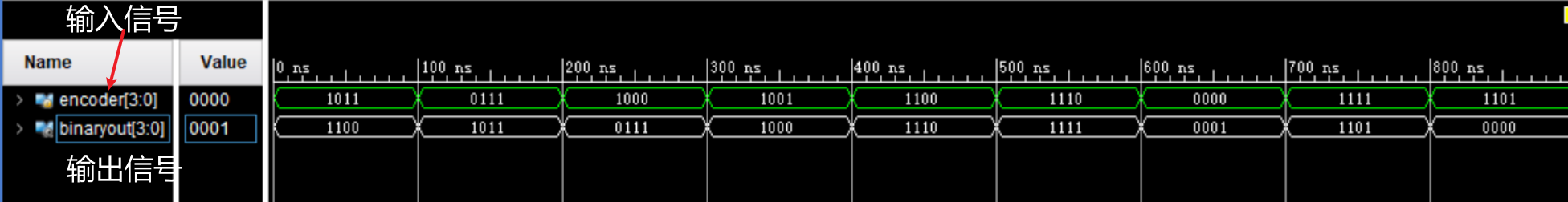


图 3.4.3 信号转换模块的仿真结果

由仿真结果中可以看见，encoder输入与binaryout输出之间的关系符合状态转换图所展现的关系，即第一行输入信号为前一个状态，对应第二行输出信号就为下一个状态。即输入信号为“0001”时，可转换为输出信号“0010”；输入信号为“0010”时，输出“0011”；输入“0011”时，输出“1010”；输入“1010”时，输出“1001”；输入“0100”，输出“0100”；输入“0101”，输出“0101”；输入“0110”，输出“0110”；输入“1011”，输出“1100”；输入“0111”，输出“1011”；输入“1000”，输出“0111”；输入“1001”，输出“1000”；输入“1100”，输出“1110”；输入“1110”，输出“1111”；输入“0000”，输出“0001”；输入“1111”，输出“1101”；输入“1101”，输出“0000”。仿真结果符合状态转换图所展示的结果。

**3.5 数据记录模块**

**3.5.1 数据记录模块的功能与设计**

数字记录模块是是将按键输入的信息记录下来，以便在数码管上实现移位显示信息的功能。

数据记录模块的封装如图 3.5.1 所示。

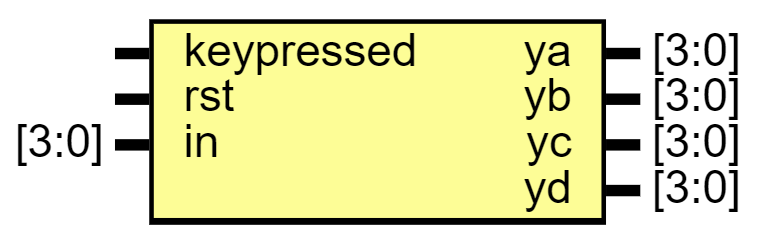


图 3.5.1 数据记录模块的封装图

1. 输入端口

keypressed：按键按下的信号，作为内部模块的触发源

rst：复位信号，高电平时输出全为0

in：从信号转换模块输入的数据

1. 输出端口

ya、yb、yc、yd：经过数据切换后的输出结果

1. 功能：将按键输入信息记录下来，以便在数码管上进行移位显示
2. 说明：ya输出的是最近一次接收的数据，然后将之前的数据传给yb，yb将之前的数据传给yc，yc将之前的数据传给yd，然后这四个输出端口再将数据送到显示模块中，以便在数码管上显示，每个数据端口对应一个数码管，通过数据一次传递的方式，来达到移位显示的功能。
3. 数字记录模块的主要代码
4. always @(posedge keypressed, posedge rst)
5. begin
6. **if** (rst==1)
7. begin
8. ya\_l<=4'b0000;
9. yb\_l<=4'b0000;
10. yc\_l<=4'b0000;
11. yd\_l<=4'b0000;
12. end
13. **else**
14. begin
15. ya\_l<=in;
16. yb\_l<=ya;
17. yc\_l<=yb;
18. yd\_l<=yc;
19. end
20. end

**3.5.2 数据记录模块的仿真**

数据记录模块的仿真结果如图3.5.2所示。

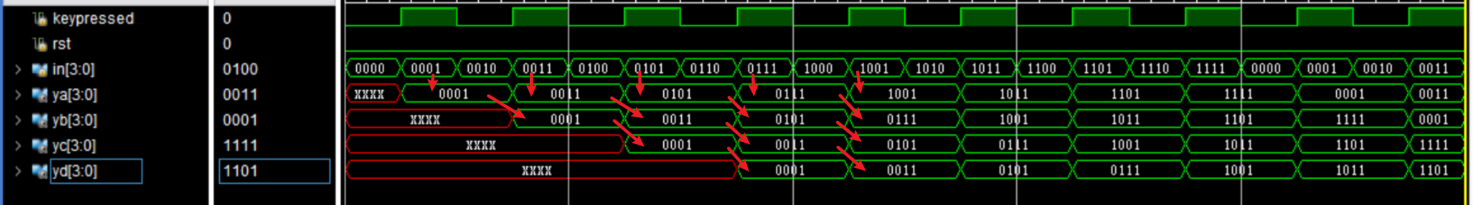


图 3.5.2 数字记录模块仿真结果

该模块的功能在keypressed的上升沿触发，第一次触发时将0001赋值给ya，第二次触发时将0011赋值给ya，ya将之前的0001赋值给yb，以此类推。由仿真结果可知该模块功能可以正常实现。

**3.6 显示模块**

**3.6.1 显示模块的功能与设计**

显示模块是将数字记录模块以及结果转换模块送来的四位二进制数正确的转换成段码，从而让数码管显示正确的内容。

显示模块的封装如图 3.6.1 所示。

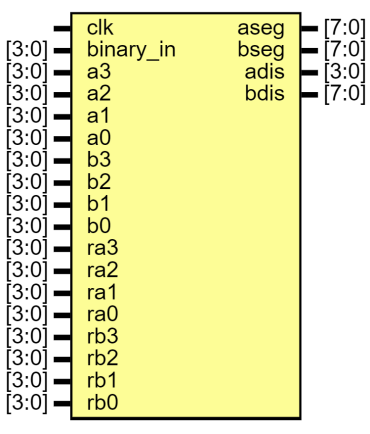


图 3.6.1 显示模块封装图

1. 输入端口

clk：系统时钟100MHz

binary\_in：由信号转换模块(myencoder)输入，用来判断是否输入了等号

a3~b0：由两个数据记录模块(shift4)输入，每一个端口对应一个数码管

ra3~rb0：由结果转换模块(translation)输入，每一个端口对应一个数码管，用于输出计算结果

1. 输出端口

aseg：右边四位数码管的段码

bseg：左边四位数码管的段码

adis：右边四位数码管的位码

bdis：左边四位数码管的位码

1. 功能：将输入的二进制码，转换为对应形式的段码，并输出到数码管上
2. 说明：先将输入的每一个二进制码转换成段码，再将转换后的段码赋值给数码管的段码，用case语句遍历，通过位码与段码的结合，实现四个数码管的动态显示，且每个数码管显示的内容都可单独控制。在未按下等号时，输出显示键盘输入的内容，在按下等号后，输出显示计算结果的内容。
3. 显示模块的主要代码
4. always @(posedge clk1k)
5. begin
6. **if**(binary\_in!=4'b1111)//当键盘输入的不是等号时，正常输出键盘信号
7. begin
8. cnt1<=cnt1+1;
9. **case**(cnt1)
10. 2'b00:
11. begin
12. dataa=a0;  datab=b0;
13. adis\_r=4'b0001;  bdis\_r=4'b0001;
14. end
15. 2'b01:
16. begin
17. dataa=a1;  datab=b1;
18. adis\_r=4'b0010;  bdis\_r=4'b0010;
19. end
20. 2'b10:
21. begin
22. dataa=a2;  datab=b2;
23. adis\_r=4'b0100;  bdis\_r=4'b0100;
24. end
25. **default**:
26. begin
27. dataa=a3;  datab=b3;
28. adis\_r=4'b1000;  bdis\_r=4'b1000;
29. end
30. endcase
31. end
32. **else**                  //输入的是等号，输出运算结果
33. begin
34. cnt2<=cnt2+1;
35. **case**(cnt2)
36. ……//与case(cnt1)下的内容形式一致，只是将datea、dateb等号右边的内容换为对应的ra3~rb0
37. endcase
38. end
39. end

**3.6.2 显示模块的仿真**

显示模块的仿真结果如图 3.6.2所示。

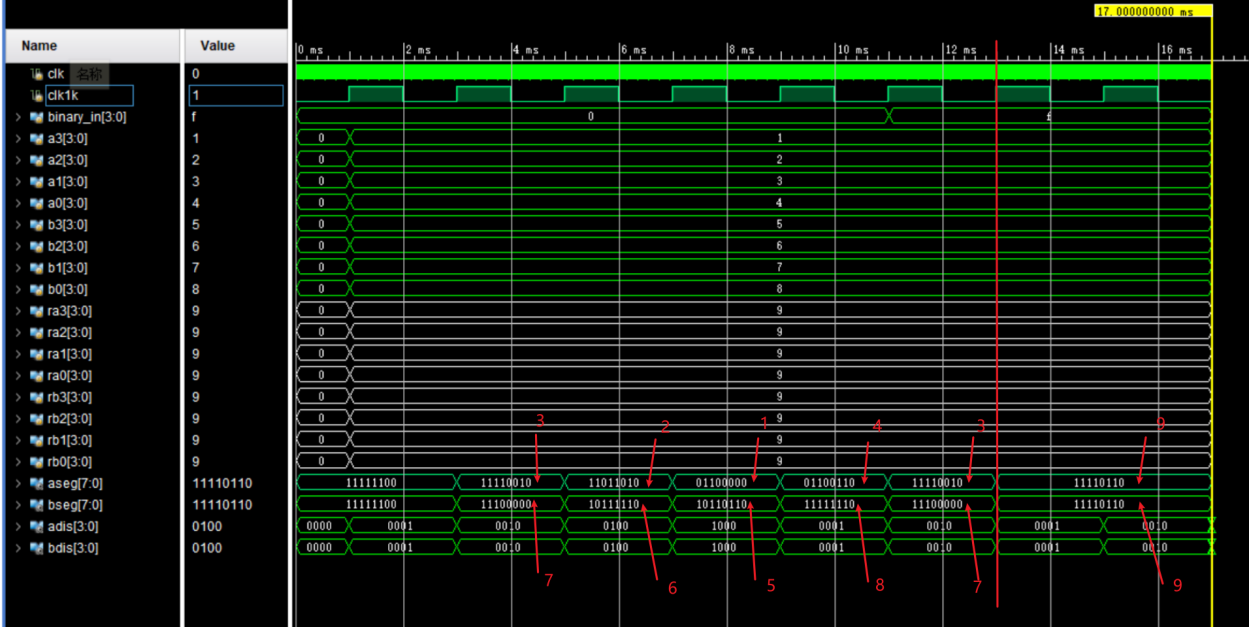


图 3.6.2 显示模块的仿真结果

由仿真结果可以看见，在键盘没有按下等号时，显示模块动态显示数据记录模块即端口a3~b0送来的数据，在实际情况中即为显示键盘输入信息；在按下等号，即binary\_in输入4’b1111时，显示模块动态显示端口ra3~rb0送来的数据，在实际情况中即为显示运算结果。

**3.7 计算模块**

**3.7.1 计算模块的功能与设计**

计算模块(calculate)是本设计的核心模块，是实现数字计算器功能的主要部分。计算模块是将信号转换模块(myencoder)送来的数据进行识别转换，将其转换为运算数以及运算符，然后对两个数进行加减乘除的操作。

计算模块的封装如图3.7.1所示。

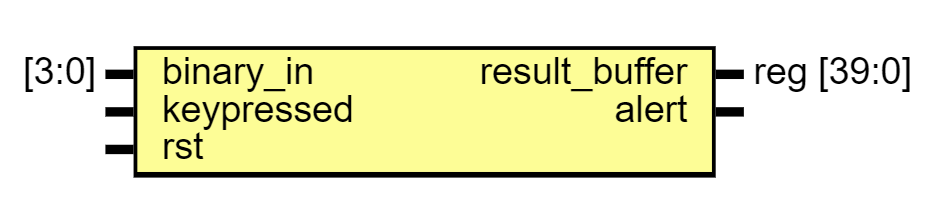


图 3.7.1 计算模块封装图

1. 输入端口

binary\_in：从信号转换模块输入的按键编码

keypressed：按键按下的信号，作为内部块的驱动信号

rst：复位信号，输入高电平时输出全为0

1. 输出端口

result\_buffer：计算结果的二进制数

alert：报警信号，当计算出现问题时，该信号输出高电平

1. 功能：对信号转换模块(myencoder)送来的按键编码进行识别，将编码组合成运算数后进行计算。
2. 说明：

该模块主要实现四位正整数的加减乘除，且运算结果保留两位小数。信号转换模块(myencoder)送来的编码只是运算数中其中一位数字的编码，由于无法事先知道用户输入数据的位数，所以在接收数据时采用数据记录模块(shift4)的方式移位接收，接收到第一个数据时，将其记录为个位，如果后面继续接收到数据，则将之前接收的数据记录为十位，新接收的数据记录为个位，以此类推，从而实现在无法确定用户输入数据位数的情况下，将不同位的8421BCD码组合起来转换为二进制数并扩大100倍便于后续的计算。完成第一个数的输入的标志是用户输入了运算符，在输入运算符时，首先将输入的运算符记录下来，然后接收第二个运算数，再根据之前记录的运算符，选择运算方式，对两个二进制数进行运算，得到的结果也是二进制数。

在除法运算中，由于最后的结果要保留两位小数，所以运算时先将被除数扩大100倍后再运算，这样得到的运算结果中包含了两位小数。再除法运算中还需判断除数是否为0，若为0，则发出警报，即alert端口输出高电平。

在乘法运算中，由于先前将每个运算数都扩大了100倍，如果不做处理就将两数相乘，则运算结果将会扩大10000倍，不利于后续对运算结果进行统一处理，所以在两数相乘后，再除以100，使运算结果也是扩大了100倍。

1. 计算模块的主要代码
2. always @(negedge keypressed,posedge rst)
3. begin
4. **if**(rst==1)//复位信号有效，所有状态清零
5. begin
6. result\_buffer<=0;
7. alert\_n<=0;
8. end
9. **else**//复位信号无效，正常进行运算
10. begin
11. **if**(binary\_in>=10 && binary\_in<=13)//加减乘除
12. begin
13. yunsuanfu<=binary\_in;//将运算符保存下来
14. cnt\_num<=cnt\_num+1;//使计算器等下接收第二个数
15. end
17. **else** **if**(binary\_in==15)//等号
18. begin
19. cnt\_num=cnt\_num+1;//使计算器等下接收下一个数
20. **case** (yunsuanfu)
21. 4'b1010: //加号
22. begin
23. result\_buffer<=a\_number+b\_number;
24. y\_a\_ones=0;
25. y\_a\_tens=0;
26. y\_a\_hundreds=0;
27. y\_a\_thousands=0;
28. y\_b\_ones=0;
29. y\_b\_tens=0;
30. y\_b\_hundreds=0;
31. y\_b\_thousands=0;
32. end
33. 4'b1011: //减号
34. begin
35. result\_buffer<=a\_number-b\_number;
36. ……//各信号赋0，与加法运算部分相同
37. end
38. 4'b1100: //乘号
39. begin
40. result\_buffer<=(a\_number\*b\_number)/100;
41. ……//各信号赋0，与加法运算部分相同
42. end
43. 4'b1101: //除号
44. begin
45. **if**(b\_number==0)//除数为0时发出警告
46. begin
47. alert\_n<=1;
48. ……//各信号赋0，与加法运算部分相同
49. end
50. **else**//除数不为0时正常计算
51. begin
52. //result\_buffer<=(a\_number\*b\_number)/100;
53. result\_buffer<=(a\_number\*100)/b\_number;
54. ……//各信号赋0，与加法运算部分相同
55. end
56. end
57. **default**:
58. begin
59. alert\_n<=1;
60. end
61. endcase
62. end
64. **else**//用来接收数据的
65. begin
66. alert\_n<=0;
67. **if**(!cnt\_num)//输入第一个数时
68. begin
69. y\_a\_ones<=binary\_in;
70. y\_a\_tens<=a\_ones;
71. y\_a\_hundreds<=a\_tens;
72. y\_a\_thousands<=a\_hundreds;
73. end
74. **else**//输入第二个数时
75. begin
76. ……//与接收第一个数的形式相同
77. end
78. end
79. end
80. end
81. //将不同位的BCD码组合起来
82. assign a\_number=a\_ones\*100+a\_tens\*1000+a\_hundreds\*10000+a\_thousands\*100000;
83. assign b\_number=b\_ones\*100+b\_tens\*1000+b\_hundreds\*10000+b\_thousands\*100000;

**3.7.2 计算模块的仿真**

计算模块的仿真结果如图 3.7.2 所示。

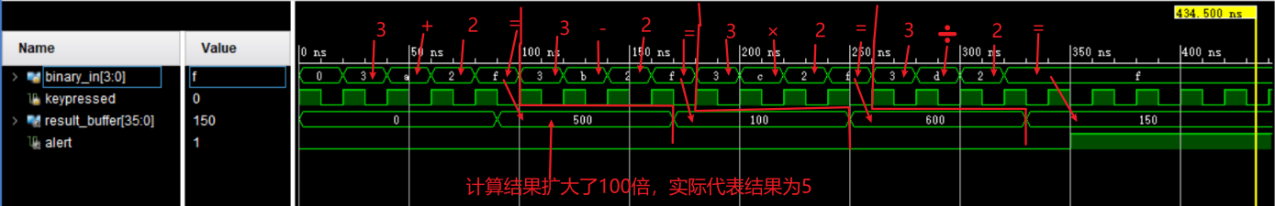


图 3.7.2 计算模块的仿真结果1

可以看见加减乘除的运算结果都正常，并且在3/2产生小数时，仿真结果为150，由于运算结果均为扩大100倍后产生的，所以实际运算结果为1.5，说明运算结果可以保留小数。

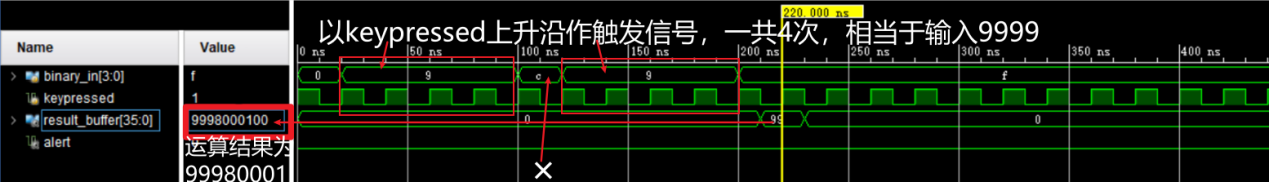


图 3.7.3 计算模块的仿真结果2

由图3.7.3的仿真结果可以看出，9999×9999的结果应为99980001，与仿真结果相同，所以本设计可以实现四位正整数的加减乘除，而不会造成数据丢失。

**3.8 结果转换模块**

**3.8.1 结果转换模块的功能与设计**

结果转换模块是对计算模块输出的数据进行处理，由于计算模块输出的计算结果是二进制数，所以需要结果转换模块将计算结果的每一位分离出来，形成每一位上对应数字的8421BCD码，从而便于输出到显示模块进行显示。

结果转换模块的封装如图 3.8.1 所示。

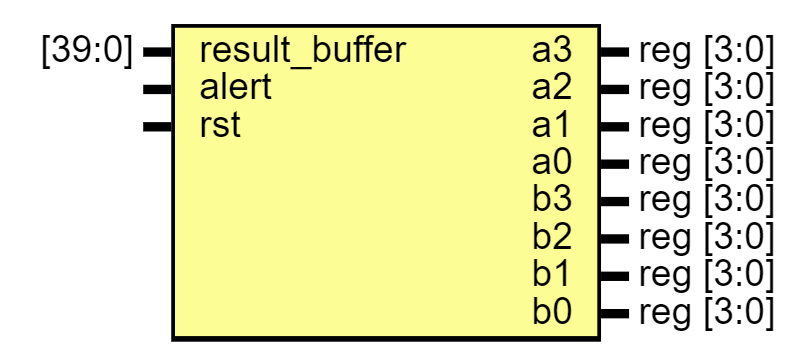


图 3.8.1 结果转换模块封装图

1. 输入端口

result\_buffer：从计算模块(calculate)输入的二进制计算结果

alert：警报信号，如果计算出现问题则输入高电平

rst：复位信号

1. 输出端口

a3~b0：计算结果每一位从高到低的8421BCD码

1. 功能：将计算模块(calculate)的二进制计算结果转化为每一位的8421BCD码进行输出
2. 说明：先将运算结果的二进制数按照最低的百分位到最高的千位转换为对应十进制数的8421BCD码，再按照每一位对应输出端口进行输出。由于只有除法运算会出现小数，运算结果最大不超过9999，且为了与输出模块匹配，小数点单独占用一个数码管，当有两位小数时，最多占用7个数码管，硬件资源足够。当运算结果中出现小数时，会有一个输出端口按照顺序输出小数点的键盘编码。当警报信号有效时，会间隔输出除号与等号的键盘编码，提示用户异常。
3. 转换模块的主要代码

将二进制数转换为对应十进制数的8421BCD码的相关代码：

1. **for**(i = 39; i >= 0; i = i - 1)//将运算结果的二进制数转换成每一位上的8421BCD码
2. begin
3. **if** (point\_twos >= 4'd5)
4. point\_twos = point\_twos + 4'd3;
5. **if** (point\_ones >= 4'd5)
6. point\_ones = point\_ones + 4'd3;
7. **if** (int\_ones >= 4'd5)
8. int\_ones = int\_ones + 4'd3;
9. **if** (int\_twos >= 4'd5)
10. int\_twos = int\_twos + 4'd3;
11. **if** (int\_threes >= 4'd5)
12. int\_threes = int\_threes + 4'd3;
13. **if** (int\_fours >= 4'd5)
14. int\_fours = int\_fours + 4'd3;
15. **if** (int\_fives >= 4'd5)
16. int\_fives = int\_fives + 4'd3;
17. **if** (int\_sixs >= 4'd5)
18. int\_sixs = int\_sixs + 4'd3;
19. **if** (int\_sevens >= 4'd5)
20. int\_sevens = int\_sevens + 4'd3;
21. **if** (int\_eights >= 4'd5)
22. int\_eights = int\_eights + 4'd3;
23. int\_eights   = {int\_eights[1:0],int\_sevens[3]};//千万位
24. int\_sevens   = {int\_sevens[2:0],int\_sixs[3]};//百万位
25. int\_sixs   = {int\_sixs[2:0],int\_fives[3]};//十万位
26. int\_fives  = {int\_fives[2:0],int\_fours[3]};//万位
27. int\_fours  = {int\_fours[2:0],int\_threes[3]};//千位
28. int\_threes     = {int\_threes[2:0],int\_twos[3]};//百位
29. int\_twos   = {int\_twos[2:0],int\_ones[3]};//十位
30. int\_ones   = {int\_ones[2:0],point\_ones[3]};//个位
31. point\_ones     = {point\_ones[2:0],point\_twos[3]};//十分位
32. point\_twos     = {point\_twos[2:0],result\_buffer[i]};//百分位
33. end

控制输出端口的相关代码：

1. **if**(alert)//有错误警报，除号与等号间隔显示
2. begin
3. b0<=4'b1101;  b1<=4'b1111;  b2<=4'b1101;  b3<=4'b1111;
4. a0<=4'b1101;  a1<=4'b1111;  a2<=4'b1101;  a3<=4'b1111;
5. end
6. **else** **if**(rst==1)//复位（清零）信号有效全部输出0
7. begin
8. b0<=4'b0000;  b1<=4'b0000;  b2<=4'b0000;  b3<=4'b0000;
9. a0<=4'b0000;  a1<=4'b0000;  a2<=4'b0000;  a3<=4'b0000;
10. end
11. **else**//正常输出计算结果
12. begin
13. **if**(point\_ones==0 && point\_twos==0)//没有小数
14. begin
15. b0<=int\_eights;  b1<=int\_sevens;  b2<=int\_sixs;  b3<=int\_fives;
16. a0<=int\_fours;  a1<=int\_threes;  a2<=int\_twos;  a3<=int\_ones;
17. end
18. **else** **if**(point\_ones!=0 && point\_twos==0)//有一位小数
19. begin
20. b0<=int\_sixs;  b1<=int\_fives; b2<=int\_fours;  b3<=int\_threes;
21. a0<=int\_twos;  a1<=int\_ones;
22. a2<=4'b1110;//小数点的按键编码
23. a3<=point\_ones;
24. end
25. **else**//有两位小数
26. begin
27. b0<=int\_fives;  b1<=int\_fours;  b2<=int\_threes;  b3<=int\_twos;
28. a0<=int\_ones;
29. a1<=4'b1110;//小数点的按键编码
30. a2<=point\_ones;  a3<=point\_twos;
31. end
32. end

**3.8.2 结果转换模块的仿真**

结果转换模块的仿真结果如图3.8.2所示。

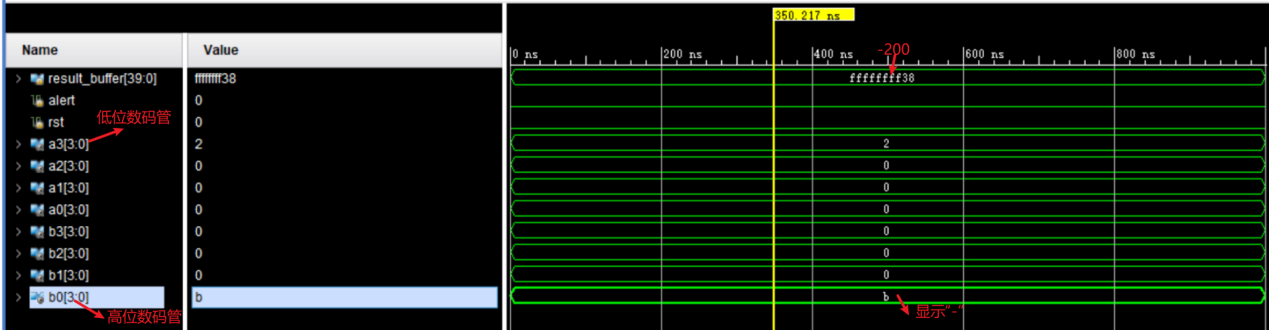


图 3.8.2 结果转换模块仿真结果

当输入-200的补码时，最高位数码管显示负号，由于计算结果是扩大100倍后的，且结果中没有小数，所以实际显示结果应为-2，最低位数码管显示数字2的段码，仿真结果符合预先设计。

**3.9 顶层模块**

**3.9.1顶层模块的功能与设计**

顶层模块是将各个子模块连接起来，使各模块整体实现数字计算器的功能。

顶层模块的封装如图 3.9.1 所示。

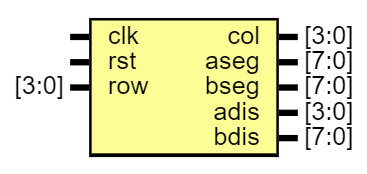


图 3.9.1 顶层模块的封装图

1. 输入端口

clk：系统时钟100MHz

rst：复位信号

row：键盘的行输入，一般处于上拉状态，键盘中按键被按下时输入低电平

1. 输出端口

col：键盘的列输出，一般处于下拉状态，正常工作时只有一列输出低电平，依次循环

aseg：输出右边四位数码管的段码

bseg：输出左边四位数码管的段码

adis：输出右边四位数码管的位码

bdis：输出左边四位数码管的位码

1. 功能：将各个子模块整合起来
2. 说明：将各个子模块按照原先的设计例化到顶层模块中
3. 顶层模块连接图

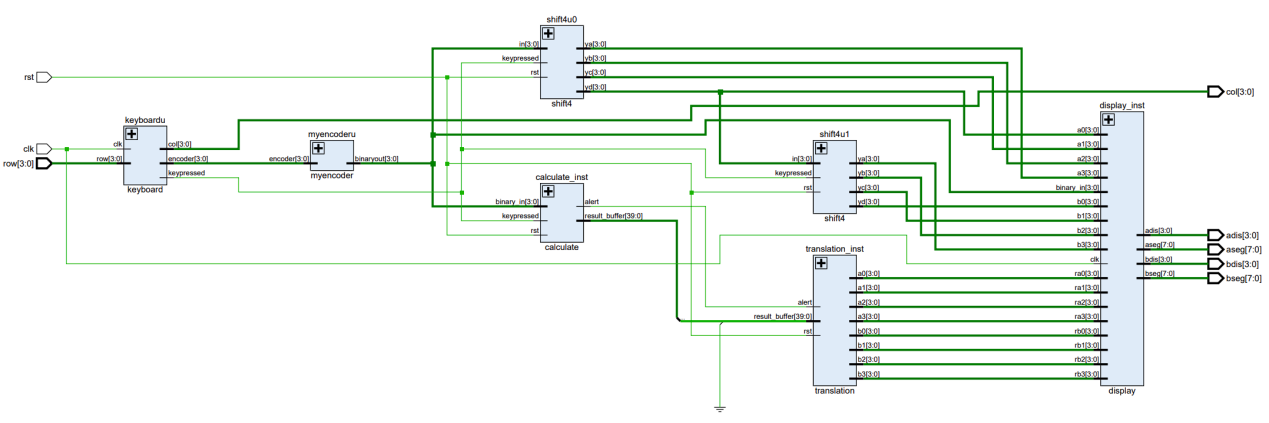


图 3.9.2 顶层模块的子模块连接图

1. 顶层模块的主要代码
2. keyboard keyboardu(
3. .clk(clk),  .row(row),  .col(col),
4. .keypressed(keypressed),  .encoder(encoder)
5. );
6. myencoder myencoderu(
7. .encoder(encoder),  .binaryout(binaryout)
8. );
9. shift4 shift4u0(
10. .keypressed(keypressed), .rst(rst),  .in(binaryout),
11. .ya(ya),  .yb(yb),  .yc(yc),  .yd(yd)
12. );
13. shift4 shift4u1(
14. .keypressed(keypressed),  .rst(rst),
15. .in(yd),  .ya(yya),  .yb(yyb),  .yc(yyc),  .yd(yyd)
16. );
17. display  display\_inst (
18. .clk(clk),
19. .binary\_in(binaryout),
20. .a3(ya),  .a2(yb),  .a1(yc), .a0(yd),
21. .b3(yya),  .b2(yyb),  .b1(yyc),  .b0(yyd),
22. .ra3(a3),  .ra2(a2),  .ra1(a1), .ra0(a0),
23. .rb3(b3),  .rb2(b2), .rb1(b1), .rb0(b0),
24. .aseg(aseg),  .bseg(bseg),  .adis(adis),  .bdis(bdis)
25. );
27. calculate  calculate\_inst (
28. .binary\_in(binaryout),
29. .keypressed(keypressed),
30. .rst(rst),
31. .result\_buffer(result\_buffer),
32. .alert(alert)
33. );
35. translation  translation\_inst (
36. .result\_buffer(result\_buffer),
37. .alert(alert),
38. .rst(rst),
39. .a3(a3),  .a2(a2),  .a1(a1),  .a0(a0),
40. .b3(b3),  .b2(b2),  .b1(b1),  .b0(b0)
41. );

**4 开发板调试**

**4.1 管脚分配**

使用Vivado中的I/O planning功能分配管脚。在主平台左侧的Flow Navigator选择SYNTHESIS下的Run Synthesis。在弹出的窗口中选择Open Synthesized Design。在Window菜单中找到I/O Ports选项，进行分配Package Pin和I/O Std。其中Package Pin根据Ego1\_UserManual进行选择，I/O Std选择LVCMOS33。同时软件自动生成.xdc约束文件。最终生成的管脚分配情况，如表1所示。

表 1 管脚情况分配表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Direction | Package Pin | I/O Std | Pull Type |
| adis[3] | OUT | G6 | LVCMOS33 | NONE |
| adis[2] | OUT | E1 | LVCMOS33 | NONE |
| adis[1] | OUT | F1 | LVCMOS33 | NONE |
| adis[0] | OUT | G1 | LVCMOS33 | NONE |
| aseg[7] | OUT | D4 | LVCMOS33 | NONE |
| aseg[6] | OUT | E3 | LVCMOS33 | NONE |
| aseg[5] | OUT | D3 | LVCMOS33 | NONE |
| aseg[4] | OUT | F4 | LVCMOS33 | NONE |
| aseg[3] | OUT | F3 | LVCMOS33 | NONE |
| aseg[2] | OUT | E2 | LVCMOS33 | NONE |
| aseg[1] | OUT | D2 | LVCMOS33 | NONE |
| aseg[0] | OUT | H2 | LVCMOS33 | NONE |
| bdis[3] | OUT | H1 | LVCMOS33 | NONE |
| bdis[2] | OUT | C1 | LVCMOS33 | NONE |
| bdis[1] | OUT | C2 | LVCMOS33 | NONE |
| bdis[0] | OUT | G2 | LVCMOS33 | NONE |
| bseg[7] | OUT | B4 | LVCMOS33 | NONE |
| bseg[6] | OUT | A4 | LVCMOS33 | NONE |
| bseg[5] | OUT | A3 | LVCMOS33 | NONE |
| bseg[4] | OUT | B1 | LVCMOS33 | NONE |
| bseg[3] | OUT | A1 | LVCMOS33 | NONE |
| bseg[2] | OUT | B3 | LVCMOS33 | NONE |
| bseg[1] | OUT | B2 | LVCMOS33 | NONE |
| bseg[0] | OUT | D5 | LVCMOS33 | NONE |
| col[3] | OUT | G17 | LVCMOS33 | PULLDOWN |
| col[2] | OUT | J13 | LVCMOS33 | PULLDOWN |
| col[1] | OUT | D17 | LVCMOS33 | PULLDOWN |
| col[0] | OUT | G14 | LVCMOS33 | PULLDOWN |
| row[3] | IN | F16 | LVCMOS33 | PULLUP |
| row[2] | IN | G16 | LVCMOS33 | PULLUP |
| row[1] | IN | C15 | LVCMOS33 | PULLUP |
| row[0] | IN | E16 | LVCMOS33 | PULLUP |
| clk | IN | P17 | LVCMOS33 | NONE |
| rst | IN | R1 | LVCMOS33 | NONE |

**4.2 下载步骤**

首先连接电脑与电路板，确保电源处于打开状态。在Vivado中点击Generate Bitstream生成一个比特流文件，随后点击Open Hardware Manager中的 Open target。再点击Auto Connect链接到目标板，最后点击Program Device中的Program进行芯片编程下载。

**4.3 调试结果**

验证加法运算时，首先从键盘输入第一个加数999，再按下加号，对应键盘上的“A”键，然后再输入第二个加数1000，最后按下等号，对应键盘上的“#”键。按下等号后在数码管上输出运算结果1999。

|  |  |
| --- | --- |
| 图 4.3.1 验证加法运算999+1000 | 图 4.3.2 计算结果为1999 |

验证减法运算时，按照之前的步骤先输入被减数1000，再按下减号，对应键盘上的“B”键，然后再输入减数99，按下等号后输出结果为901。

|  |  |
| --- | --- |
| 图 4.3.3验证减法运算1000-99 | 图 4.3.4 计算结果为901 |

验证乘法运算时，按照之前的步骤先输入第一个乘数9999，再按下乘号，对应键盘上的“C”键，然后再输入第二个乘数9999，按下等号后输出结果为99980001，说明该计算器可以实现最多四位整数的运算。

|  |  |
| --- | --- |
| 图 4.3.5 验证乘法运算9999×9999 | 图 4.3.6 计算结果为99980001 |

验证除法运算时，先验证正常情况下的运算，先输入被除数999，再按下除号，对应键盘上的“D”键，然后再输入除数3，按下等号后输出结果为333。

|  |  |
| --- | --- |
| 图 4.3.7 验证除法运算999÷3 | 图 4.3.8 计算结果为333 |

再验证除数为0时的运算，先输入被除数9999，再按下除号，再输入除数0后按下等号输出结果，结果数码管上并未显示数字，而是显示“F”，代表“false”，这种情况表示计算出现错误，输出警报。

|  |  |
| --- | --- |
| 图 4.3.9 验证除数为0时是否报错 | 图 4.3.10 数码管上报错 |

再验证除法运算结果保留两位小数的功能，先输入被除数10，再按下除号，再输入除数3，按下等号输出结果为3.33。

|  |  |
| --- | --- |
| 图 4.3.11 验证运算结果保留小数功能 | 图 4.3.12 运算结果保留了两位小数 |

**结论**

本论文介绍了一种基于FPGA的简便计算器设计原理及实现过程。运用自顶向下的方法，在Vivado开发环境下，采用Verilog HDL语言对输入模块、信号转换模块、数据记录模块、显示模块、计算模块、结果转换模块六个模块进行逻辑构建、编写、仿真。在顶层模块例化后，构成简易计算器，实现加减乘除三角函数值等算术功能，并将输入的运算数据、运算符号、运算结果等显示在数码管上。使用EGO1开发板完成程序的下载及调试，证明了该设计具有稳定高效、简单可靠、安全灵活等优点。

通过本设计的实现，将理论与实践相结合，对Xilinx 的FPGA结构及特点有了更深入的认识，真正理解了EDA技术对电路设计领域带来的变革作用，充分体验了FPGA的开发流程，明确了单片机系统与可编程逻辑器件的本质区别。同时，团队协作的形式也为本次课题注入了新的活力和创造力，为今后的科研学习工作提供了丰富的知识储备和实践经验。

**参考文献**

1. 周孟强,魏远江,李刚. 基于单片机控制的数字计算器[J]. 电子制作，2018(07): 77-79.
2. 杨海钢,孙嘉斌,王慰. FPGA器件设计技术发展综述[J].电子与信息学报，2010，32(03): 714-727.
3. 王红,彭亮,于宗光. FPGA现状与发展趋势[J].电子与封装, 2007(07): 32-37.
4. 张瑾. EDA技术与应用[M]. 北京：清华大学出版社，2022.
5. 花汉兵，吴少琴. EDA技术与设计[M]. 北京：电子工业出版社，2019.
6. 黄金凤. EDA技术及应用：基于FPGA的电子系统设计[M]. 西安：西安电子科技大学出版社，2021.
7. 罗杰. Verilog HDL与FPGA数字系统设计[M]. 北京：机械工业出版社，2022.
8. 张应华. FPGA系统设计与优化方法研究[M] .北京:中国原子能出版社，2021.