

中南大学

硕士学位论文

基于FPGA的MCS-51核的VHDL语言设计与实现

申请学位级别：硕士

专业：物理电子学

摘 要

本文以研究嵌入式微处理器为主,自主地设计了能够运行 MCS-51 系列单片机指令的 MCU 系统。系统采用了 VHDL 语言与原理框图的综合设计方法,并且在 Altera 公司的 FPGA 上通过验证。论文深入地研究了微处理器的指令系统和数据地址通路,采用 VHDL 语言完成了取指单元,指令译码器单元,存储器单元和逻辑运算单元的电路模块的设计与实现;研究了控制单元的实现方法和基于全局状态机的设计理论,采用硬件描述语言完成了对各个控制线的相关设计与实现。论文通过原理示意图和示例代码的演示,着重介绍了指令译码器的实现方式,基于此种方式形成的译码电路还能够实现更为复杂的 CISC 指令。

本系统采用分模块的设计方式,把具有相同功能的逻辑电路集中到一个框图里,使得系统的可移植性大大地提高。系统还采用层次框图的设计方式,把明显地具有主从关系的电路放在不同的层次里,这也使得系统模块功能的可扩展性大大地增强。内部逻辑共分为数据存储模块;程序存储器模块;时序控制模块;特殊功能寄存器模块和 Core 核心模块这五个部分,文中对各个模块的设计作了详细的介绍。本文在最后对已实现的部分典型指令进行了逻辑仿真测试,测试结果表明,本文所设计的 MCU 系统能够如预期地执行相应的指令。在指令执行的过程中,相应寄存器和总线上的值也均符合设计要求,实现了设计目标。

关键词: MCU, FPGA, VHDL, MCS-51 核



ABSTRACT

Based embedded microprocessor research, this thesis has independently designed a microprocessor system which can run MCS-51 instructions. The design is described by VHDL and block diagram. All of the instructions has been test successfully on the platform of FPGA . The thesis carefully researches on the set of instructions and data-address access ,and completes the design and realization of fetching instruction unit, instruction decoder unit, memory unit and ALU unit by hardware description language. The thesis researches on the realization method of control unit and the design theory based on overall state machine. Through Principle Map and sample code demonstration, the thesis introduces the instruction decoder of ways. Based on this approach to the formation of decoding circuit also be able to achieve more complex CISC instructions.

By means of sub-module design, the system migration is improved greatly by concentrating the same functional logic circuit to on one functional module. System also uses the design means of level block diagram and concentrates the primary and secondary circuit to different levels. This has made the system functional modules can be expanded greatly enhanced. Internal logic consists of data storage module, Program storage module, Timing Control module, Special function Registers module and Core Module. The thesis introduces each module for the design of the details. At the end of this paper, we introduced the result of logic simulation test to some typical instructions. Test results show that the MCU system can work normally. AS expected, the values of corresponding registers and bus meet design requirements, and this system achieves design goals.

KEY WORDS: MCU, FPGA, VHDL, MCS-51Core

第一章 绪 论

1.1 研究背景与意义

自从 20 世纪的 70 年代初 Intel 公司第一个推出 4004MCU(MicroController Unit)起到 80 年代初,是 MCU 技术高速发展的时期。MCU 技术的快速发展也刺激了 MCU 外围 LSI 器件的进步。在 80 年代中期 Intel 公司将 8051 内核使用权互换或出售给世界各大著名 IC 制造商,这样 8051 变成了众多制造商支持的,发展出上百个品种的大家族。8051 在小中型应用场合很常见,已经成为单片机领域的实际标准。在这个时期,MCU 由 8 位,16 位发展到 32 位,速度和集成度越来越高,再加上电子产品的少批量多品种化的趋势,高速低功耗以及小型化的要求,原来的电子系统中 MCU 的外围 LSI 和通用 IC 已适应不了这一技术上的变化^[1]。20 世纪中期以来,可编程 ASIC 以其现场可编程,高速,高集成度的优势充当了系统中新的积木块。由 MCU,存储器和可编程 ASIC 这三个可编程的积木块组成的现代电子系统已经成为了趋势和潮流。在可编程器件发展的潮流中,可编程器件从最初的由与或门组成的 PAL, GAL 发展到后来片内资源十分充足的 FPGA(Field Programmable Gate Array),CPLD(Complex Programmable Logic Device)。PAL 和 GAL 可编程逻辑器件只能实现简单的规模小的电子电路,而 FPGA,CPLD 很好的弥补了这一缺陷,它们的规模较大,可以代替通用的 IC 芯片^[2]。

由于 FPGA 的集成度较以前有很大地提高,因此鉴于传统的 MCS-51 资源不可更改的缺点,很多公司都把设计出的 51IPCore 集成到了 FPGA 内部^[3],这样就可以根据不同的需要对单片机的资源进行裁剪,优化。因此能够设计出高效,实用的 51IPCore 对于众多需要定制 51 单片机功能的厂商有着不同寻常的意义^[4]。

1.2 单片机的发展状况

自从上世纪 70 年代初世界第一块单片机的推出到现在已经有三十几年的历史了,在这短短的三十几年里,单片机的发展经历了好几次重要的飞跃,表现在如下几个方面。

单片机的处理位在逐渐翻倍,这意味着单片机处理数据的能力在不断提高,



速度也越来越快,最初的四位单片机只能用于一些简单的控制场合,如家用电器,玩具等。接着便出现了具有较快数据处理能力的八位单片机,八位单片机由于具有优异的性能,其运用领域也比四位单片机广阔的多^[12]。但是在某些需要适时较快数据处理的地方,八位单片机的速度还是达不到要求,这就又促使 16 位单片机的推出。如今,由于社会自动化生产的需要,控制软件也越来越复杂,能够支持操作系统的嵌入式系统在这方面就表现出明显的优势。所以单片机向 32 位嵌入式微处理器的发展也是一个趋势。除了位数翻倍外,单片机的封装也越来越小型化了,功耗也越做越低,这些都得益于集成电路工艺的创新^[13-15]。

1.3 研究内容及实现手段

1.3.1 本文研究内容

本文研究的主要内容包括:熟悉并运用 VHDL(Very High Speed Integrated CircuitHardware Description Language)硬件描述语言进行系统设计;嵌入式微处理器内部结构的研究;以 MCS-51 系列单片机的指令集为蓝本,利用原理框图和硬件描述语言的输入方式完成实现了处理器的各种功能;以 Altera 公司的 FPGA 芯片 EP1K50TC144-1 为目标芯片,用 Quartus II 的仿真工具对设计结果进行了验证。

1.3.2 VHDL 语言特点

VHDL 的英文全名为 VHSIC,由美国国防部于 1983 年创建,由 IEEE(The Institute of Electrical and Electronics Engineers)进一步发展,并在 1987 年作为“IEEE 标准 1076”发布。从此,VHDL 语言成为硬件描述语言的业界标准之一^[16-22]。

VHDL 作为一个规范语言和建模语言,随着它的标准化,出现了一些支持语言行为的仿真器。由于创建 VHDL 的最初目标是用于标准文档的建立和电路功能的模拟,其基本想法是在高层次上描述系统和元件的行为。但到了二十世纪九十年代初,人们发现,VHDL 不仅可以作为系统模拟的建模工具,而且可以作为电路系统的设计工具;可以利用软件工具将 VHDL 源码自动地转化为文本方式表达的基本逻辑元件连接图,即网络文件。这种方法显然对于电路自动设计是一个极大地推进。很快,电子设计领域出现了第一个软件设计工具,即 VHDL 逻辑综合器,它可以标准地将 VHDL 的部分语句描述转化为具体电路实现的网络文件。1993 年,IEEE 对 VHDL 语言进行了修订,从更高的抽象参差和系统描述能力上扩展了 VHDL 的内容,公布了新版本的 VHDL 语言,即 IEEE 标准的

1076-1993 版本。现在, VHDL 和 Verilog 作为 IEEE 的工业标准硬件描述语言, 得到众多的 EDA 公司的支持, 在电子工程领域, 已成为事实上的通用硬件描述语言。现在公布的最新的 VHDL 标准版本是 IEEE 1076-2002^[16-22]。

VHDL 语言具有很强的电路描述和建模能力, 能从多个层次对数字系统进行建模和描述, 从而大大简化了硬件设计任务, 提高了设计效率和可靠性^[23-25]。VHDL 具有与硬件电路无关和与设计平台无关的特性, 并具有良好的电路行为描述和系统描述能力, 并在语言易读性和层次化结构化设计方面, 表现了强大的生命力和应用潜力。因此, VHDL 语言在支持各种模式的设计方法, 自顶向下与自底向上或混合方法方面, 在面对当今许多电子产品生命周期的缩短, 需要多次重新设计以融入最新技术, 改变工艺等方面都表现了良好的适应性^[26-35]。

1.3.3 采用 FPGA 的设计验证手段

一般来说, 芯片设计包括三种方式, 全定制 ASIC 芯片, 半定制 ASIC 芯片和可编程的逻辑器件。对于前两种, 芯片的流片, 测试和验证花费都很大, 动辄就需几十万元。而且经常不能一次成功, 所以对于全定制和半定制的设计成本很高。而用可编程逻辑器件就不存在这样的问题, 因为其具有可重构的特性, 对硬件的修改可以直接通过对软件编程来实现, 还可以修改多次, 直到设计者满意为止。这样就可以明显地降低设计周期和开发费用, 是一种经济的设计验证手段^[36-43]。

1.4 MCS-51 指令系统简介

MCS-51 系列单片机的指令系统共有 111 条指令, 按照它们的操作性质可划分为数据传送指令、算术操作指令、逻辑操作指令、程序转移指令和位操作指令等 5 个大类。MCS-51 系列单片机的指令长度较短: 单字节指令有 49 条; 双字节指令有 46 条; 最长的是三字节指令, 只有 16 条。指令周期也较短: 单机器周期指令 64 条; 双机器周期指令 45 条; 只有乘、除两条指令需要四个机器周期。在 MCS-51 指令系统中, 有丰富的位操作指令, 形成了一个相当完整的位操作指令子系统, 成为该指令系统的重大特色。本文所设计指令均能在一个周期内完成。

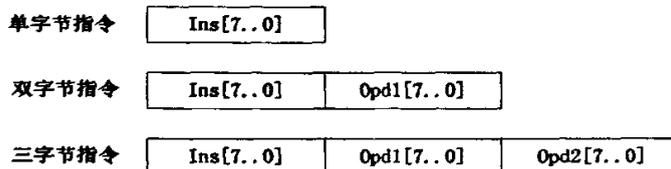
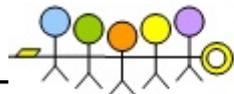


图 1-1 MCS-51 指令的结构



观察如图 1-1 所示，每一条指令通常由操作码和操作数两部分组成，前者表示该条指令将进行何种操作，后者表示操作数本身或操作数所在的地址。

处理器传输数据，执行算术操作、逻辑操作等都要涉及操作数。某条指令执行时，得先从操作数所在的地址寻找到与本指令有关的操作数，这便涉及到寻址。MCS-51 指令集的寻址方式有六种。它们分别为立即数寻址、寄存器寻址、寄存器间接寻址、直接寻址、基址寄存器加变址寄存器的间接寻址和相对寻址^[44]。

1.5 论文的结构层次及简要介绍

第一章概括了 MCS-51 核研究的背景和应用前景，51IPCore 的出现是单片机的广泛应用和 FPGA 集成度越来越高共同作用的结果，对于需要定制 51 单片机功能的应用场合显得尤为重要；然后就介绍了嵌入式微处理器研究发展的现状；最后介绍了本系统实现的方法以及本文结构层次的安排。

第二章介绍了 MCU 的整体模块布局以及各个模块的功能和它们的连接关系。介绍了本系统指令执行的时序步骤以及对信号线，寄存器等各个资源模块的命名。

第三章分别介绍了顶层模块 TC，RAM，ROM 和 SFRG 的功能和它们的实现方法。

第四章是本文重点，详细地介绍了 MCU 的核心模块 Core 的内部组织结构，在 Core 内部模块中，指令译码器（InstructionDecoder）又是本章重点介绍的部分，详细说明了指令译码器的工作原理以及控制线的信号生成原理。除了指令译码器之外，内存控制器（RamControl）和 ALU(Arithmetic Logic Unit)模块我们也作了重点介绍。

第五章也是本文的重点，详细地介绍了 Core 模块中核心寄存器的工作原理，核心寄存器包括 PC 寄存器，指令寄存器，数据指针寄存器以及堆栈寄存器等。

第六章为指令的测试章节，除了介绍系统设计时的一些相关问题外，我们重点介绍了部分典型指令的测试情况。从仿真结果可以看出，所有指令的测试结果和我们设计的目标一致。

第七章总结了本文的研究工作，提出了基于 FPGA 实现的 MCU 需要优化与改进之处，展望了整个课题的研究前景。



第二章 MCS-51Core 的整体设计

MCU 设计的第一步应该是设计指令的结构。由于本文借鉴了 MCS-51 系列单片机的指令集，因此本系统最先着手的便是先从整体结构上设计出合理的模块框图。本章先介绍了系统的整体层次框图；接着又介绍了指令执行的时序步骤以及数据总线的构建；在本章的最后我们还介绍了系统各类资源的命名。

2.1 MCS-51Core 的整体层次框图

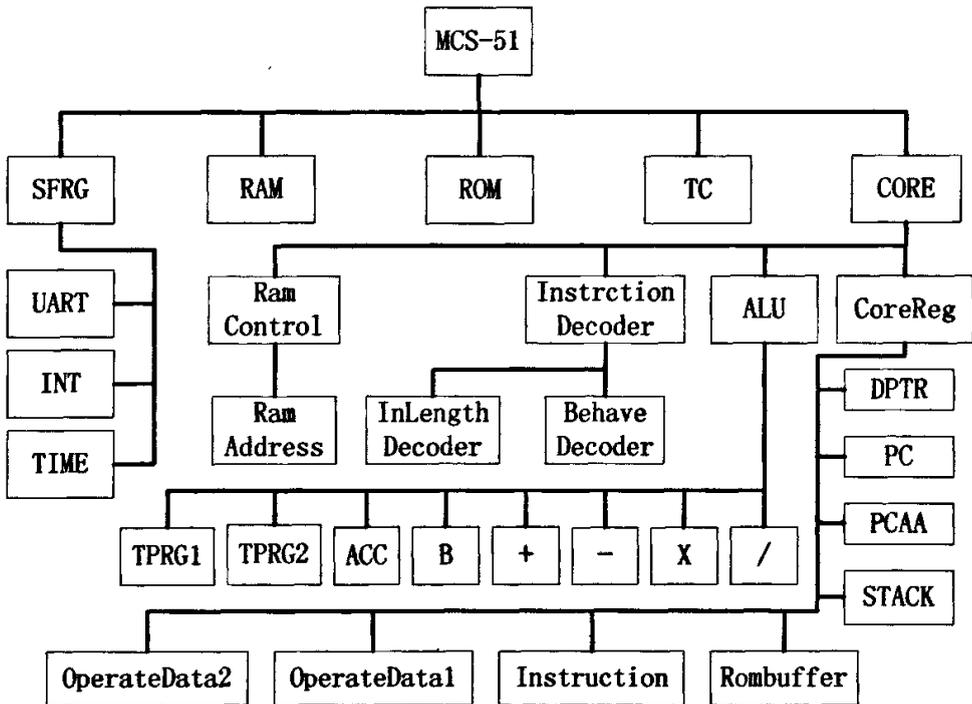
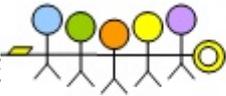


图 2-1 MCS-51Core 整体层次框图

系统划分模块功能的基本原则是尽量把具有相同或相似功能的逻辑电路放在同一个模块之内，尽量使模块之间的连线数量尽可能的少。本系统的整体框图如图 2-1 所示，最顶层的方框代表了整个 MCU 系统的整体，也可以看成是一块单片机。在第二层，从功能上分，包含五部分，第一部分是 SFRG(特殊功能寄存器组)，该系统所有的数字外设都包含在这部分，传统的 MCS-51 系列的单片机包含串口、外部中断和定时器等。第二部分是一块 ram，在这里本设计调用的是



目标芯片上集成的存储器模块，包含有 256 个存储字节^[45]。第三部分是一块 rom，同样调用的是目标芯片上的存储模块，所有要调试的指令就存储在这里。从左到右的第四个模块是一个时序控制模块 TC，本系统几乎所有和时序相关的信号如状态机，时钟，复位信号等都是由这个模块生成。

在这五大模块中，最复杂的就是最右边的 Core 核心模块了，它的下面又包含四个模块：第一个模块是 ram 控制器，基于统一地址寻址的特殊功能寄存器和数据 ram 的地址信号和控制信号就由这个模块生成；Core 模块的第二部分便是指令译码器，指令译码器包含两个部分，第一个部分是指令长度译码器，第二部分是指令行为译码器；Core 的第三部分是算术逻辑运算单元，其中所有的算术逻辑运算和位运算都包含在这个部分；第四个模块便是核心工作寄存器模块 (CoreReg)，这其中包含访问程序和数据存储器的地址寄存器和指令寄存器等。

2.2 指令的执行时序

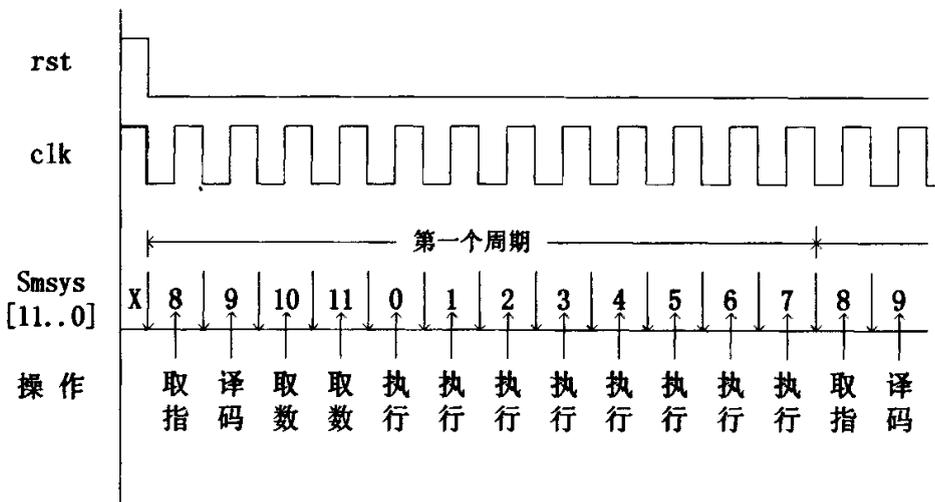


图 2-2 指令的执行时序

时钟对于系统的重要性就象“心脏”对于人体的作用一样，在这里我们简要地介绍一下指令的执行步骤。传统的 MCS-51Core 的机器周期（即执行完一条指令所需要的时钟）包含十二个时钟周期。为了和 MCS-51 系列的单片机时序完全兼容，本系统也把机器周期设计成了十二个时钟周期。如图 2-2 所示：图中第一排波形为系统已作延时处理过的复位信号，第二排波形为系统已作倍频或者降频处理（在这里就为外部时钟）过的时钟信号，第三排则为一个状态机的翻转信号。每当复位信号来临并经过一定时间的延时之后，状态机就被强制复位到状态 8，也即进入执行每条指令的第一个环节—取指阶段。当取得指令之后，系统就要对



该指令进行长度译码操作，以确定该指令是由几个字节构成，如果该指令是双字节指令或者三字节指令，那么在接下来的两个时钟或者三个时钟周期内将继续读取 rom 数据到相应的寄存器。状态机为 9 的阶段为指令的译码阶段，取数阶段为状态机中的状态 10 和 11。接下来的 0 到 8 的状态机阶段该指令就被分步骤地执行。

2.3 总线的构造

总线的作用就相当于一个城市的交通枢纽一样，如果总线设计的好，那么它就能实时高效地配合其它模块进行工作，就能在更短的时钟周期内完成尽可能多的数据交换。由于 MCS-51 系列单片机其内部结构并不复杂，所以本系统只在内核中设计了一条总线，即内部通用数据总线。它连接到了系统中几乎所有的寄存器和存储器，而寄存器和存储器又是通过一个三态门或者数据选择器和总线隔离的。相应寄存器对总线数据的读取或者写入的操作都是通过指令译码器单独控制完成的，因此系统就能在指令译码器的统一控制下对数据总线进行有序地操作，避免了同时写入而引起的总线错乱。但是如果所有的数据交换都是通过一条数据总线进行，那么指令执行的效率将会大大地降低。所以对于数据交换比较频繁的模块，系统还可以设计出专用数据通道，这样指令执行的效率就有所提高，但是耗费的硬件资源也就更多^[46-48]。

2.4 资源的定义与命名

2.4.1 时钟信号的定义

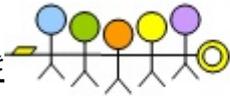
| | |
|--------------|----------------------|
| clock | : 外部时钟 |
| clk | : 内部时钟，通过外部时钟分频或倍频得到 |

2.4.2 复位信号的定义

| | |
|--------------|-------------------|
| reset | : 外部复位 |
| rst | : 内部复位，通过外部复位延迟得到 |

2.4.3 各个寄存器对总线的读写控制使能

| | |
|------------------|----------------------------|
| enin_ram | : 译码器使能 ram 输入 |
| enout_ram | : 译码器使能 ram 输出 |
| enin_ram1 | : ram 控制器使能低 128 字节 ram 输入 |



| | |
|--------------|-------------------------------|
| enin_ramh | : ram 控制器使能高 128 字节 SFR 寄存器输入 |
| enout_raml | : ram 控制器使能低 128 字节 ram 输出 |
| enout_ramh | : ram 控制器使能高 128 字节 SFR 寄存器输出 |
| enout_rombuf | : rom 数据输出至总线缓冲器使能 |
| enin_trg1 | : trg1 寄存器输入使能 |
| enout_trg1 | : trg1 寄存器输出使能 |
| enin_trg2 | : trg2 寄存器输入使能 |
| enout_trg2 | : trg2 寄存器输出使能 |
| enin_acc | : acc 寄存器输入使能 |
| enout_acc | : acc 寄存器输出使能 |
| enin_pcl | : 总线输出至 pc 低八位寄存器使能 |
| enin_pch | : 总线输出至 pc 高八位寄存器使能 |
| enout_pcl | : pc 低八位寄存器输出至总线使能 |
| enout_pch | : pc 高八位寄存器输出至总线使能 |

2.4.4 RAM 地址寄存器数据源使能

| | |
|--------------|---------------------------|
| enadd_opd1 | : 以立即数 1 形成 ram 地址使能 |
| enadd_opd2 | : 以立即数 2 形成 ram 地址使能 |
| enadd_Ri | : 以指令寄存器低 3 位形成 ram 地址使能 |
| enadd_Rj | : 以指令寄存器地 1 位形成 ram 地址使能 |
| enadd_stacku | : 以堆栈指针寄存器形成 ram 地址使能, 递增 |
| enadd_stackd | : 以堆栈指针寄存器形成 ram 地址使能, 递减 |
| enadd_dbda | : 以通用数据总线形成 ram 地址使能 |

2.4.5 指令译码器输出控制线名称定义

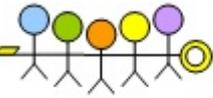
| | |
|-----------|-----------------------------------|
| enin_dptr | : 使能 opd1 和 opd2 赋值给 dptr |
| enpcaa | : 使能 pc+dbda |
| enswap | : acc 高四位与低四位交换使能 |
| enxchd | : acc 与 trg1 交换低四位, 用于特定指令 |
| enopl | : acc 按位求反, 用于特定指令 |
| enclr | : acc 每位清零, 用于特定指令 |
| enrl | : acc 循环左移, 用于特定指令 |
| enrr | : acc 循环右移, 用于特定指令 |
| enrlc | : acc 带进位循环左移, 用于特定指令 |
| enrrc | : acc 带进位循环右移, 用于特定指令 |
| enajmp | : instruction 和 opd1 合计 11 位送给 pc |
| enljmp | : opd1 与 opd2 合计 16 位送给 pc |
| ensjmp | : opd1 送给 pc |
| enjmp | : dptr+acc 送给 pc |
| encjne | : opd2 送给 pc |
| entrglu | : trg1 递增 |
| entrgld | : trg1 递减 |



| | |
|-----------------|------------------------|
| enani | : trg1 与 dbda 送给 trg2 |
| enxrl | : trg1 异或 dbda 送给 trg2 |
| enori | : trg1 或 dbda 送给 trg2 |
| ensetbc | : 设置进位位为 1 |
| enpclic | : 翻转进位位, 用于位操作 |
| enclrc | : 进位位清零, 用于位操作 |
| enclrb | : 任意位清零, 用于位操作 |
| enclpb | : 任意位翻转, 用于位操作 |
| ensetbb | : 任意位置一, 用于位操作 |
| enmcb | : 任意位送进位位, 用于位操作 |
| enmbc | : 进位位送任意位, 用于位操作 |
| enanlcb | : 任意位与进位位与并送进位位 |
| enorlcb | : 任意位与进位位或并送进位位 |
| enanlcnb | : 任意位非与进位位与并送进位位 |
| enorlcnb | : 任意位非与进位位或并送进位位 |

2.4.6 各个寄存器全局名称

| | |
|--------------------------|---------------------|
| stack[7..0] | : stack 指针寄存器 |
| dbda[7..0] | : 内部通用数据总线 |
| instruction[7..0] | : 指令寄存器 |
| smsys[11..0] | : 全局状态机 |
| dbda_rom[7..0] | : 程序存储器数据输出 |
| dbad_rom[15..0] | : 程序存储器地址输入 |
| dbad_ram[7..0] | : 数据存储器地址输入 |
| inlength[1..0] | : 指令长度指示 |
| opd1[7..0] | : 立即数 1 |
| opd2[7..0] | : 立即数 2 |
| pc[15..0] | : 程序地址计数器 |
| dptr[15..0] | : 数据指针 |
| trg1[7..0] | : 临时工作寄存器 1, 不可统一寻址 |
| trg2[7..0] | : 临时工作寄存器 2, 不可统一寻址 |
| acc[7..0] | : 累加寄存器 |
| B[7..0] | : 辅助乘法寄存器 |
| P0-P3 | : 通用 IO 口 |
| scon[7..0] | : 串口控制寄存器 |
| tcon[7..0] | : 定时器控制寄存器 |
| IE[7..0] | : 中断允许寄存器 |
| sbufir[7..0] | : 串口接收缓冲器 |
| sbufit[7..0] | : 串口发送缓冲器 |
| ras[1..0] | : 程序 rom 地址选择信号 |
| tempbit | : 临时位寄存器, 用于位操作。 |



2.4.7 各个寄存器内部名称

在全局名称后面加_r，如 acc 是全局名称，其内部名称为 acc_r。

2.5 本章小结

本章从整体构架上对整个层次模块图作了介绍，并且对每个模块的相应功能作了简要的说明。指令的执依靠时钟节拍驱动，因此在第二小结本文对指令执行时状态机的跳变作了介绍。总线是处理器中最常用的数据通路，在第三小结介绍了本系统总线构造的一些情况。在最后本文列出了本系统所用到的各类资源的命名。



第三章 MCS-51Core 的分模块设计

本章首先介绍了本系统的第二层次框图的连接关系,另外还将详细介绍系统中除 Core 模块之外的其余四大模块的功能以及实现原理。这四大模块分别为: Ram 存储器模块、Rom 存储器模块、时序控制模块 TC 和特殊功能寄存器组 SFRC 模块。Core 模块由于其内部复杂,我们将在第四章单独加以介绍。

3.1 顶层模块布局

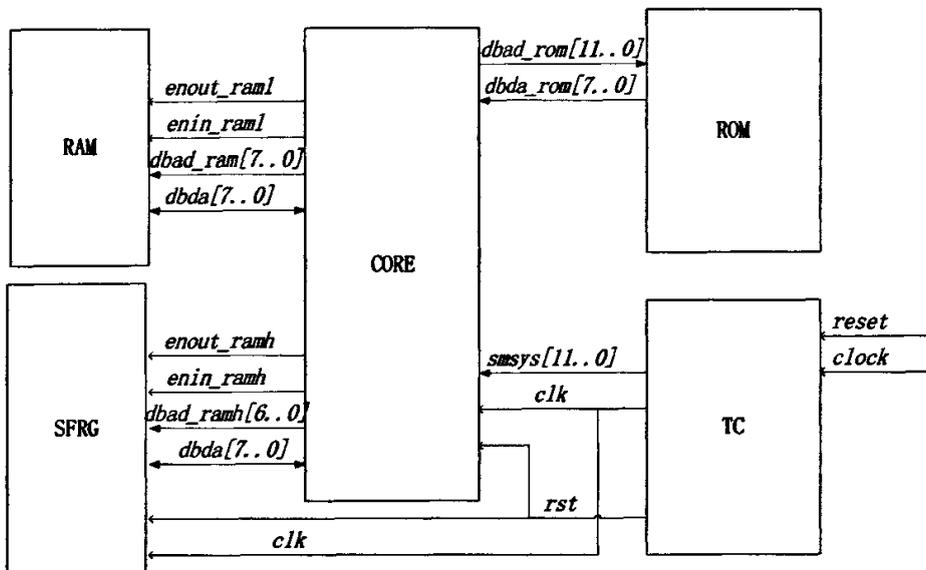
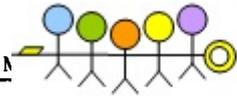


图 3-1 顶层模块布局

顶层模块的布局如图 3-1 所示。根据功能的不一样,系统将其内部逻辑共分成了五个部分。其中最核心的就是中间的 core 模块,对指令的译码和执行都是在 Core 模块内部完成的,而且 Core 模块还对除时序控制模块 TC 以外的其余三个模块进行控制。

TC 模块是一个时序控制模块,主要为系统提供三个信号,第一个就是内部复位信号 *rst*,第二个就是时钟信号 *clk*,最后一个就是系统状态信号 *smsys[11..0]*;

图中左上角是一块有着 256 个字节的 ram, core 有四个信号对它实现了读写操作。第一个信号就是写使能 *enin_ram1*;第二个信号就是读使能 *enout_ram1*;第三个信号是 ram 的地址信号 *dbad_ram[7..0]*;最后一个就是通用数据总线



dbda[7..0]。

由于 MCS-51 系列单片机属哈佛结构，指令与数据存放在不同的存储器里，所以本设计又构造了一个存放指令的 rom 存储器，见图右上角。因其容量为 4k，所以有 12 根地址线，命名为 dbad_rom[11..0]；数据输出为 dbda_rom[7..0]^[44]。

MCU 系统的数字外围设备都被集中放在了 SFRG 模块里，这样便可以很好地进行扩展。由于 MCS-51 中的特殊功能寄存器和它的数据 ram 是统一编址的，所以 SFRG 模块的地址输入和 ram 模块一样，都是 dbad_ram[7..0]。其数据口也和内部数据总线 dbda 相连。鉴于现今 S52 系列的单片机内部有 256 个字节的 ram，其高 128 字节的地址空间和 SFRG 的地址空间重合，所以在这里系统另外多设计了两根使能信号线 enin_ramh 和 enout_ramh，以实现对他们高 128 字节的分别访问。

3.2 SFRG 模块设计

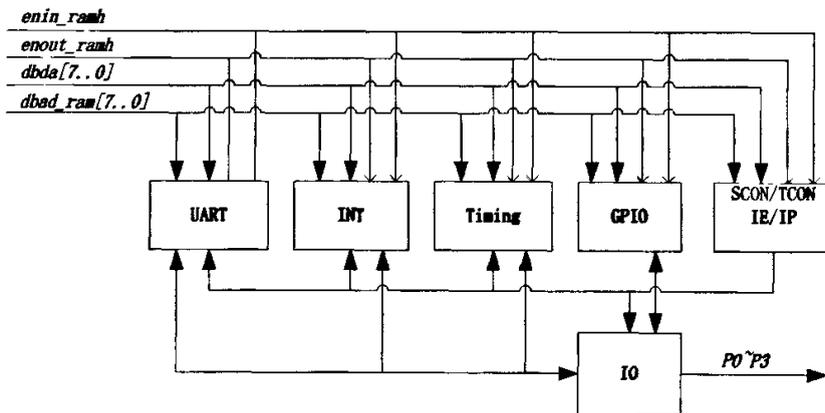


图 3-2 sfrg 模块结构

3.2.1 模块资源的说明

如图 3-2 所示，此模块包含了 MCS-51 系列单片机的数字外围设备，其中有串口—UART^[50]，外部中断 INT，定时器模块 Timing 和通用 IO 口 GPIO。

3.2.2 统一编址寄存器读写示例代码

假设系统要对 scon 寄存器进行读写，scon 的通用访问地址为 0x98，其寄存器的内部名称为 scon_r 对 scon 寄存器的读写的操作的 vhdl 代码如下：

例 3.1

--scon_r 寄存器的值输出到通用数据总线，这实际是一个三态门的设计

```
with enout_ramh&dbad_ram select
```

```
    dbda<= scon_r      when '1'&x"98",
```

--当 ram 总线地址为 0x98 并且访问特殊功能寄存器使能时输出至总线

```
    "ZZZZZZZ"        when others;
```

--如条件不满足要求则 scon_r 与总线断开

例 3.2

--通用数据总线 dbda 的值写入到 scon_r 寄存器

```
process(clk,rst)
```

```
begin
```

```
    if(rst='1')      then
```

```
        scon_r<=x"00";
```

--当系统复位信号 rst 为高电平时，scon_r 的值被设定为 x"00"

```
    elsif(clk'event and clk='1')      then
```

--当时钟上升沿到来后

```
        if(enin_ramh='1' and dbad_ram=x"98")      then
```

--如果存储器写有效并且存储器地址为 x"98"

```
            scon_r <=dbda;
```

--数据总线的值被写入到 scon_r 寄存器

```
        end if;
```

```
    end if;
```

```
end process;
```

3.3 RAM 模块设计

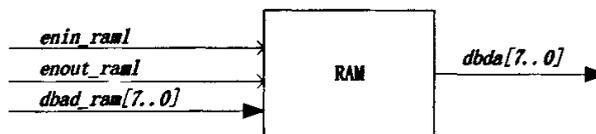
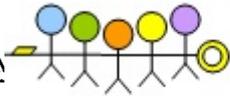


图 3-3 ram 模块

ram 结构如图 3-3 所示，由于本设计所用的目标芯片内部就集成的有 ram 存



储单元，所以在此只须调用即可。

3.4 ROM 模块设计

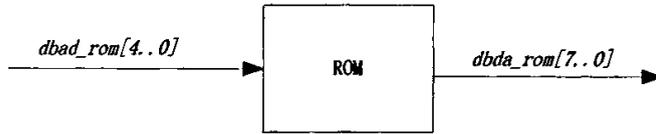


图 3-4 rom 模块

rom 结构如图 3-4 所示，同样是调用目标芯片上的存储单元。为了调试方便，在这里只调用了能够存储 32 个字节的存储单元作为 rom 模块。如果要调试指令，我们就先得要在 keil 编译器里把要调试的指令以汇编程序的格式输入，然后再编译生成 hex 文件，最后用生成的 hex 文件把 ROM 模块的数据更新。

3.5 TC 模块设计

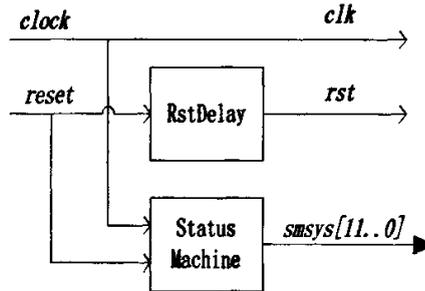


图 3-5 TC 模块

3.5.1 模块功能说明

TC 模块如图 3-5 所示，此模块主要为整个系统提供三类信号；第一类信号就是内部时钟信号 clk ，在这里系统直接把输入的外部 $clock$ 信号直接输出。如果要想提高本系统的时钟频率，锁相环就应加在 $clock$ 和 clk 中间。

第二个就是为整个系统提供复位信号 rst ，输入的复位信号 $reset$ 经过一定周期的延时之后便传递给 rst ，为什么要加入延时呢？因为系统的状态机在复位之后状态必须是 8，并且 clk 时钟必须为低电平，如果在复位之后 clk 是高电平，那么时钟低电平来临时状态机就直接跳转到 9，漏掉了取指这个阶



段。取指的条件是状态为 8 且时钟为上升沿。相应的状态机信号生成的示例代码如例 3.3。

3.5.2 状态机信号生成示例代码

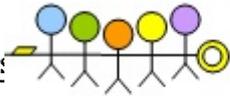
例 3.3

--状态机信号生成示例代码

```
process(clk_r,rst_r)
begin
    if(clk_r'event and clk_r='0')    then
        --如果时钟下降沿到来
        if(rst_r='1')    then
            smsys_r<="000100000000";
            --如果复位信号有效，则 smsys_r 就复位到状态 8
        else
            smsys_r(11 downto 1)<=smsys_r(10 downto 0);
            smsys_r(0)<=smsys_r(11);
            --否则的话状态机就循环左移移位，实际上是移位状态机的设计
        end if;
    end if;
end process;
```

3.6 本章小结

本章除介绍了本系统第二层次的五大模块的连接关系外，还详细地说明了各个模块（不包括 Core 模块）的功能以及内部结构。这些模块分别分别为 SFRG 模块、RAM 模块、ROM 模块和 TC 时序控制模块，同同层次的 Core 模块相比较，这几个模块结构简单，较易实现。



第四章 MCS-51Core 的 Core 模块设计

本章主要介绍了 Core 模块的结构。Core 模块是系统最重要的模块，也是该系统最复杂的部分和我们设计的重点。Core 模块中最复杂的部分是指令译码器单元，除此之外本章还介绍了内存控制模块 (RamControl)、算术逻辑运算单元 (ALU) 和一些核心寄存器。

4.1 Core 模块整体布局

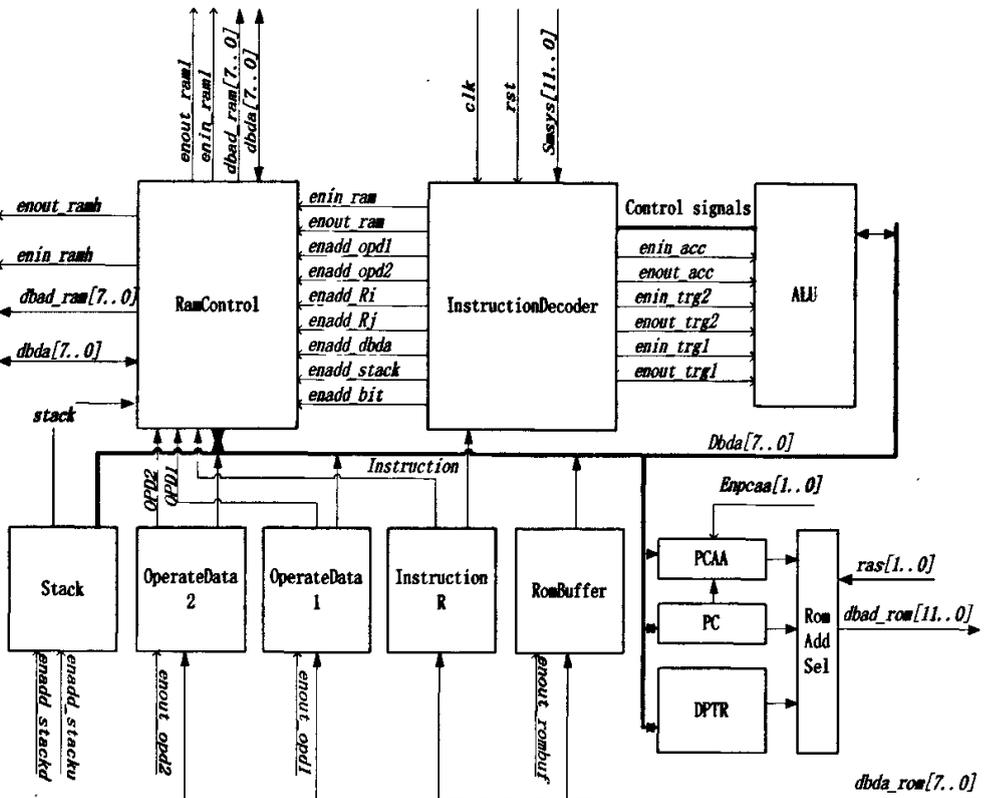


图 4-1 Core 模块整体布局

Core 模块的内部结构如图 4-1 所示^[49-55]，其中最核心的就是指令译码器 (InstructionDecoder) 模块，其余几个模块都是围绕着指令译码器展开的。如图中右下角有三个寄存器，分别为 *dptra*，*pc* 和 *pcaa*，它们都是 16 位地址寄存器，



专为程序 rom 提供访问地址。由指令译码器输出的 $ras[1..0]$ 的值决定具体选择哪个地址寄存器作为 rom 的访问地址。dptr 为数据指针，pc 为程序地址计数器，pcaa 为 pc 或者 dptr 与数据总线数值之和，主要应用于查表指令的操作。

Rombuffer 为程序存储器数据输出到数据总线的缓冲器，InstructionR 为指令寄存器，专为 InstructionDecoder 模块与 RamControl 模块寄存指令数据。OperateData1 与 OperateData2 与是两个储存立即数的寄存器。这四个寄存器的输入数据都来源于程序 Rom 存储器。stack 为堆栈指针，为 ram 提供地址。

RamControl 模块为内存控制模块，专为外部 ram 存储器和 SFR 特殊功能寄存器提供地址信号和读写控制信号。提供的地址数据为 $dbda_ram[7..0]$ ；提供的低 128 字节的 ram 读写控制信息为 $enin_raml$ 、 $enout_raml$ ；提供的 SFR 寄存器读写控制信息为 $enin_ramh$ 、 $enout_ramh$ 。这些都是受指令译码器输出的 $enin_ram$ 和 $enout_ram$ 控制的。地址的来源有 5 个，它们分别为数据总线 $dbda[7..0]$ ；立即数 $OPD1[7..0]$ ；立即数 $OPD2[7..0]$ ；指令 $Instruction[7..0]$ 的低三位或低一位。堆栈指针 $stack[7..0]$ ，由指令译码器发出的地址形成控制信号决定选择哪一个源地址作为 ram 访问地址，指令译码器传输到 RamControl 模块的地址形成控制线有 7 条，分别对应不同的访问指令，它们分别为 $enadd_opd1$ ，以立即数 1 作为地址； $enadd_opd2$ ，以立即数 2 作为地址； $enadd_dbda$ ，以数据总线的值作为地址； $enadd_Ri$ ，以指令的低三位和特殊状态位作为地址； $enadd_Rj$ ，以特殊状态位和指令的低一位作为地址； $enadd_stack$ ，以堆栈指针作为地址； $enadd_bit$ 以立即数 1 作为地址，用于位操作。

所有逻辑运算、算术运算和位操作都在 ALU 中实现，指令译码器输出至 ALU 单元的控制信号较多，但不外乎有以下三类；第一类为总线读写信号，这其中包括 acc ， $trg1$ 和 $trg2$ 对总线的读写操作；第二类为逻辑运算使能信号；第三类为算术操作使能信号。这些操作大部分都是在 acc ， $trg1$ 或 $trg2$ 中实现的。

4.2 RAM 控制器的设计

本小节主要介绍 ram 控制器的基本原理，用框图和源代码加以说明。

4.2.1 RAM 控制器模块功能说明

RAM 控制器的功能就是对所有的统一编址的寄存器和 ram 进行读写控制操

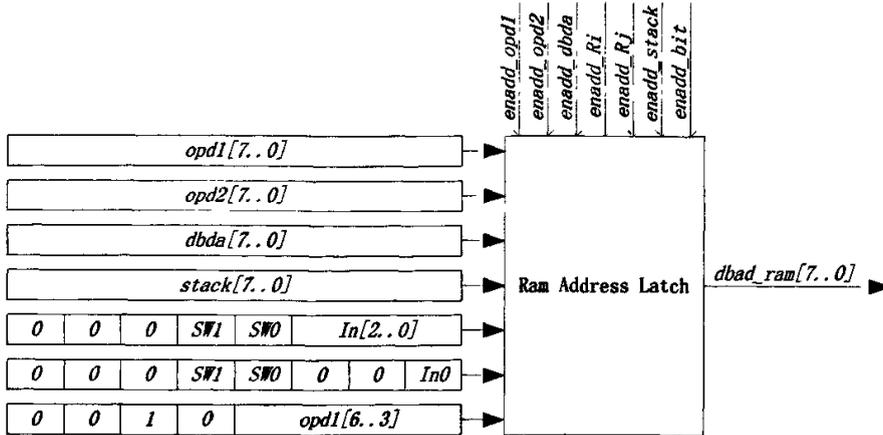
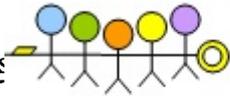


图 4-2 RAM 控制器的地址形成电路

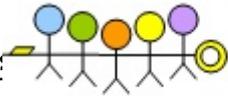
作，并在访问存储器或者寄存器前形成正确的地址。如图 4-2 所示，地址形成电路的地址源有 7 个，其中 opd1, opd2, dbda 和 stack 四个地址源可以对整个 ram 的 256 个字节进行访问。接下来的两个地址源用于形成工作寄存器区 Rx 和地址指针 Rj 的地址，最下面的一个地址源用于形成位地址。指令译码器输出的 ram 地址锁存信号为 enadd_opd1, enadd_opd2, enadd_dbda, enadd_Ri, enadd_Rj, enadd_stack, enadd_bit。由它们决定哪个地址源作为 ram 的访问的地址。

4.2.2 RAM 寄存器地址形成电路示例代码

例 4.1

```

process(clk)                                     --地址形成电路示例代码
begin
    if(clk'event and clk='1')                    --时钟上升沿到来
        then
            if(enadd_opd1='1')                   --立即数 1 形成地址
                then
                    dbad_ram_reg<=opd1;
            elsif(enadd_opd2='1')                then
                dbad_ram_r<=opd2;                --立即数 2 形成地址
            elsif(enadd_rx='1')                   then
                dbad_ram_r<="00000"&instruction(2 downto 0); --Ri 地址
            elsif(enadd_rj='1')                   then
                dbad_ram_r<="0000000"&instruction(0);      --Rj 地址
            elsif(enadd_dbda='1')                 then
                dbad_ram_r<=dbda;                --dbda 地址
    
```



```

elseif(enadd_stacku='1')      then      --堆栈地址
    dbad_ram_r<=stackadd;
elseif(enadd_stackd='1')      then      --堆栈地址
dbad_ram_r<=conv_std_logic_vector((conv_integer(stack(7 downto 0))-1),8);
elseif(enadd_bit='1')        then      --形成位地址
    dbad_ram_reg<="0010"&opd1(6 downto 3);
end if;
end if;
end process;
    
```

4.3 指令译码器(InstructionDecoder)的设计

4.3.1 指令译码器的原理示意图

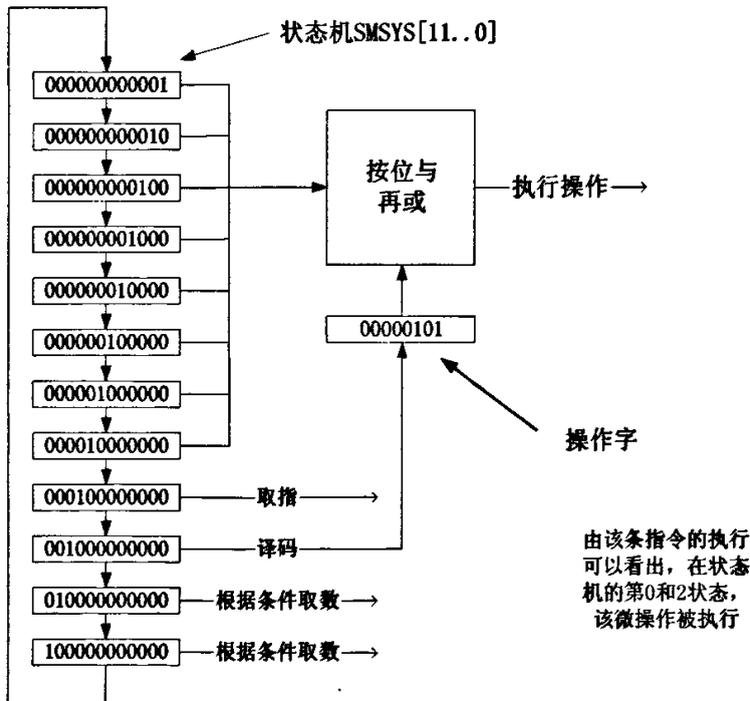
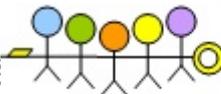


图 4-3 指令译码器的原理示意图

指令译码器的原理如图 4-3 所示，左边的状态机 smsys[11..0]实际上是一个 12 位的移位寄存器，每来一个脉冲，smsys[11..0]便左移移位，在复位状态下 smsys[11..0]的初值为“000100000000”；当状态为“000100000000”并且时钟上升沿



```

elseif(
    Instruction=X"E5"           --mov  A, direct
or Instruction(7 downto 3)="11101" --mov  A, Ri
or Instruction=X"93"           --movc A, @A+dptr
or Instruction=X"83"           --movc A, @A+pc
                                ) then
    enin_acc_r <="00000010";

```

--当为以上几条指令时，在状态为 1 时执行该微操作，下同

```

elseif(
    Instruction=X"54"           --and  A, #data
or Instruction=X"44"           --orl  A, #data
or Instruction=X"64"           --xrl  A, #data
                                ) then
    enin_acc_r <="00000100";

```

```

elseif(
    Instruction(7 downto 3)="01011" --and  A, Ri
or Instruction(7 downto 3)="01001" --orl  A, Ri
or Instruction(7 downto 3)="01101" --xrl  A, Ri
or Instruction=X"55"           --and  A, direct
or Instruction=X"45"           --orl  A, direct
or Instruction=X"65"           --xrl  A, direct
                                ) then
    enin_acc_r <="00001000";

```

```

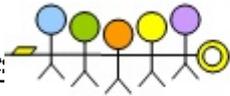
elseif(
    Instruction(7 downto 1)="1110011" --mov  A, Rj
or Instruction=X"C5"           --xch  A, direct
or Instruction(7 downto 3)="11001" --xch  A, Ri
                                ) then
    enin_acc_r <="00001000";

```

```

elseif(
    Instruction(7 downto 1)="1100011"
or Instruction(7 downto 1)="0101011" --and  A, @Rj
or Instruction(7 downto 1)="0100011" --orl  A, @Rj
or Instruction(7 downto 1)="0110011" --xrl  A, @Rj

```



```

        )      then
            enin_acc_r <="00100000";
        else
            enin_acc_r <=X"00";
            --如果都不是以上指令时，则操作字为 X"00";不执行任何操作。
        end if;
    end if;
end process;
--状态机与操作字先按位与，再作或运算，输出控制信号
enin_acc <= ( ( enin_acc_r (7)and smsys(7))           --第 7 位与
              or(enin_acc_r (6)and smsys(6))         --第 6 位与
              ...
              or(enin_acc_r (0)and smsys (0))      );   --第 0 位与
    
```

4.4 算术逻辑运算单元(ALU)的设计

4.4.1 算术逻辑单元的原理图设计

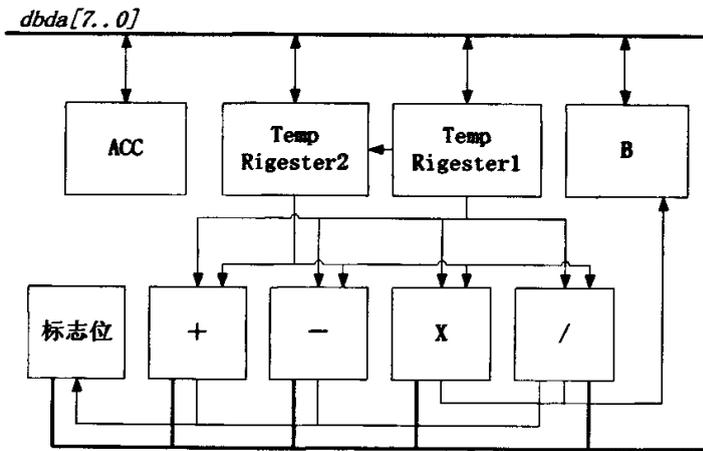
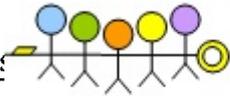


图 4.4 算术逻辑单元结构示意图

算术逻辑单元的原理框图如图 4-4 所示，在上图中我们可以看到，加减乘除的功能分别由四个不同的模块完成。它们的数据来源有两个，分别为 Temp Register1 和 Temp Register2。其中加减的运算输出结果直接传输至总线，经总线可直接传输至 ACC 或者立即数寻址的寄存器或者 ram 存储器。由于乘法和除法



有两个结果输出，因此它们的结果除传输给总线外，另一个结果还传送给寄存器 B。标志位也随着加减乘除的运算结果改变而改变^[49-55]。由于运算单元的源操作数都取自于 Temp Register1 和 Temp Register2，因此在作相应的运算之前必须先把要参与运算的数据传输至上面两个寄存器^[24, 19]。这就涉及到数据转移的微操作了。

由于逻辑运算的运算关系比较简单，需求的资源数量较少。所以本文把逻辑运算单元集成到了 Temp Register2 单元里，两个源操作数分别取自 Temp Register1 和 dbda[7..0]。

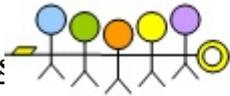
算术运算单元则是直接调用目标芯片内部的算术运算模块实现的。

4.4.2 算术逻辑运算单元中的示例代码

例 4.3

—————逻辑运算单元的实现代码

```
process(clk)
begin
    if(clk'event and clk='1')    then    --当时钟上升沿来临的时候
        if(enin_trg2='1')    then    --如果 trg2 应该输入，
            trg2_r<=dbda;    --则总线数据传输至 trg2_r
        elsif(enand='1')    then
            --如果是作与运算，那么下面每一位就按位相与
            trg2_r(7)<=dbda(7)and trg1(7);
            ...
            trg2_r(0)<=dbda(6)and trg1(0);
        elsif(enor='1')    then
            --如果是作或运算，那么下面每一位就按位相或
            trg2_r(7)<=dbda(7)or trg1(7);
            ...
            trg2_r(0)<=dbda(6)or trg1(0);
        elsif(enxrl='1')    then
            --如果是作异或运算，那么下面每一位就按位相异或
            trg2_r(7)<=dbda(7)xor trg1(7);
            ...
            trg2_r(0)<=dbda(6)xor trg1(0);
        end if;
```



```

end if;
end process;

```

4.5 位操作的实现

4.5.1 位操作电路的原理示意图

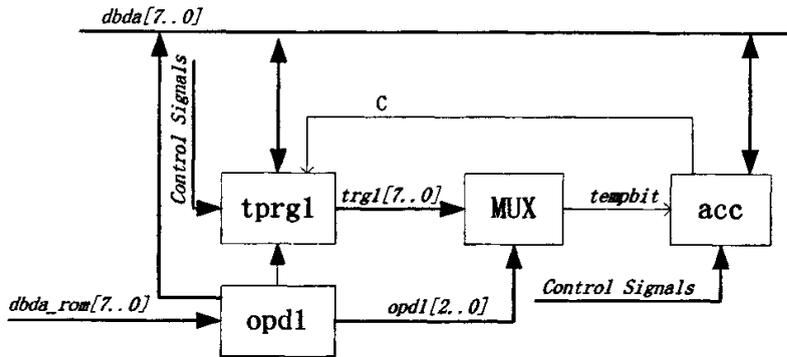


图 4-5 位操作电路的原理示意图

所有位操作指令一共有 17 条，可分为三类，第一类就是位自身状态的改变，第二类就是任意位与 C 之间的传送与运算，第三类就是以位为判断条件的跳转指令。如图 4-5 所示，所有的位运算都集中在 tprg1 和 acc 这两个模块中。对于直接寻址位的指令其第一个操作数的前 5 位为位字节所在的地址，在前面我们已经介绍了其 Ram 访问地址形成的电路原理，在此我们不再介绍。后面 3 位是表示该字节的哪一位。其立即数存放于 opd1 中，因此系统把立即数 opd1 的低 3 位引入到数据选择寄存器 MUX 中，被操作位的相应字节被传送到 tprg1 中，trg1 又送给 MUX，这样要操作的位就被 MUX 选择存储到临时位寄存器 tempbit_r 中并输出，tempbit 即可以作为跳转指令的判断条件，又可以作为位逻辑运算的一个操作位，另一个操作位为 C。

对于改变进位位 C 状态的位操作在 acc 模块中完成；对于改变任意位的状态的操作在 tprg1 模块中完成；对于任意位与 C 之间的逻辑运算的操作在 ACC 中完成；对于以位作为判断的跳转指令，tempbit 被直接传送到指令译码器作为判断条件。相应的代码在后文给出。



4.5.2 位操作的示例代码

例 4.4

-----模块 acc 中位操作的实现代码，其余代码略掉

```

process(clk)
begin
    if(clk'event and clk='1')    then
        if(encrc='1')            then
            c_r<='0';            --位 C 清零
        elsif(enstbc='1')        then
            c_r<='1';            --位 C 置一
        elsif(encplc='1')        then
            c_r<=not c_r;        --位 C 取反
        elsif(enmcb='1')         then
            c_r<=tempbit;       --任意位传送给 C
        elsif(enanlcb='1')       then
            c_r<=c_r and tempbit; --位 C 与任意位与并传给位 C
        elsif(enorlcb='1')       then
            c_r<=c_r or tempbit; --位 C 与任意位或并传给位 C
        elsif(enanlcnb='1')      then
            c_r<=c_r and (not tempbit); --位 C 与任意位非与并传给位 C
        elsif(enorlcnb='1')      then
            c_r<=c_r or (not tempbit); --位 C 与任意位非或并传给位 C
        end if;
    end if;
end process;

```

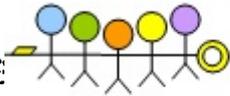
例 4.5

-----模块 tprg1 中位操作的实现代码，其余代码略掉

```

process(clk)
begin
    if(clk'event and clk='1')    then
        elsif(encplb='1')        then
            if(opd1(2 downto 0)="000") then
                --按位取反
            end if;
        end if;
    end if;
end process;

```



```

        trgl_reg(0)<=not trgl_reg(0);
    ...
    elsif(opd1(2 downto 0)="111")      then
        trgl_reg(7)<=not trgl_reg(7);
    end if;
elsif(enclrb='1')      then           --按位清零
    if(opd1(2 downto 0)="000")      then
        trgl_reg(0)<='0';
    ...
    elsif(opd1(2 downto 0)="111")      then
        trgl_reg(7)<='0';
    end if;
elsif(ensetbb='1')      then           --按位置一
    if(opd1(2 downto 0)="000")      then
        trgl_reg(0)<='1';
    ...
    elsif(opd1(2 downto 0)="111")      then
        trgl_reg(7)<='1';
    end if;
elsif(enmbc='1')      then           --位 C 移位给任意位
    if(opd1(2 downto 0)="000")      then
        trgl_reg(0)<=c;
    ...
    elsif(opd1(2 downto 0)="111")      then
        trgl_reg(7)<=c;
    end if;
end if;
end if;
end process;

```

例 4.6

-----数据位选择器示例代码

```

tempbit<=tempbit_r;           --寄存器位 tempbit_r 输出
with opd1(2 downto 0) select

```



```
tempbit_r<= trg1(0) when "000",    --当为 000 时, 选择 trg (0)
      trg1(1) when "001",    --当为 001 时, 选择 trg (1)
      ...
      trg1(7) when "111",    --当为 111 时, 选择 trg (7)
      '0'    when others;
```

4.6 本章小结

Core 核心模块是本文设计的重点, 本章详细地说明了 Core 下层各个模块的功能、实现方法以及它们的连接关系。Core 下层模块有 RAM 控制器模块、指令译码器模块和 ALU 算术逻辑运算单元。RAM 控制器模块对 RAM 和所有统一编址的寄存器发出读写控制信号以及形成访问它们的正确地址。指令译码器负责对所有指令的译码并且发出本系统所有的控制信息。ALU 是算术逻辑运算单元, 本设计中 ALU 模块有三个主要功能, 第一个功能就是进行算术运算操作, 第二个功能就是进行逻辑运算操作, 最后一个就是进行位操作。



第五章 Core 模块中重要寄存器的设计

本章主要介绍 Core 模块中的核心寄存器的设计，这其中包括最重要的 PC 程序地址计数器，很多跳转类的指令就是在 PC 模块中完成的。另外还有 $dptr16$ 位数据指针寄存器、PCAA 查表指令地址寄存器、InstructionR 指令寄存器、OperateData1 立即数 1 寄存器、OperateData2 立即数 2 寄存器和堆栈指针(Stack) 寄存器。

5.1 PC 寄存器的设计

5.1.1 PC 寄存器的结构框图

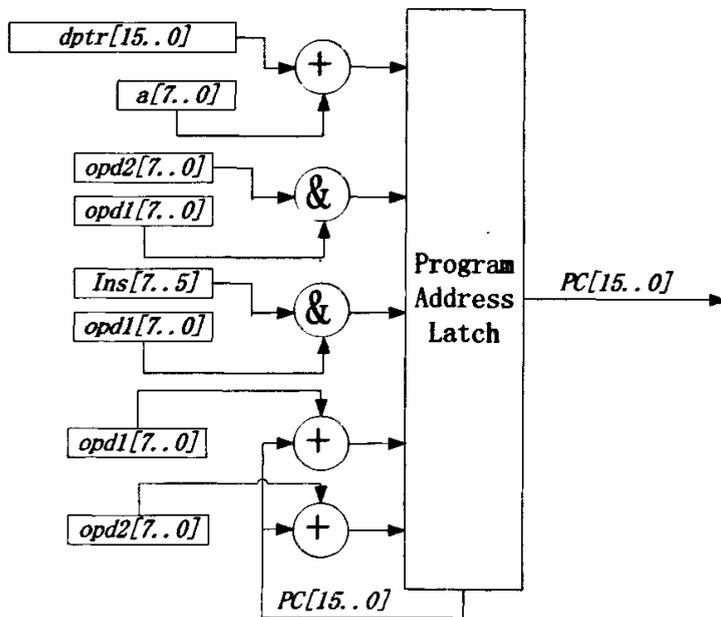
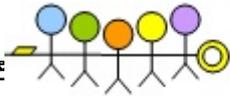


图 5-1 PC 寄存器结构框图

PC 寄存器的设计如图 5-1 所示，它的输入信号为 $enajmp$, $enljmp$, $ensjmp$, $enjmp$, $inlength$ (图中未画)。其中除最后一个输入外，前面的信号都是由跳转类指令触发的，跳转指令的工作原理就是把将要跳到的程序地址写入 PC 程序寄存器。而写入的程序计数器 PC 的数值来源有如下几个： $opd1[7..0]$ 、 $opd2[7..0]$ 、 $dbda[7..0]$ 、PC+1 和 $dptr[15..0]$ ，其中对于相对跳转指令来说还要和 PC 作有符号



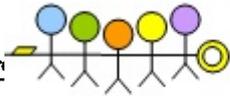
数的加减运算。另外所需的信号为 `rst`, `clk`, `smsys[11..0]`在图中未画出。每当状态机的状态信号为 8 时, PC 便自加 1; 当为 9 和 10 时, 加一与否就要由指令长度寄存器 `inlength` 来决定了, 当跳转指令来临时, 指令计数器 PC 中的值就有可能写入新的程序地址值, 或者在原值的基础上作一下加减操作。其实现代码如下所示。

5.1.2 PC 寄存器的示例代码

例 5.1

-----PC 寄存器单元的实现代码

```
process(clk,rst)
begin
    if(clk'event and clk='1')          then      --时钟上升沿触发
        if(rst='1')                    then
            pc_r<=0;                    --复位后, 从指令地址 0 处开始执行
        elsif(smsys(8)='1')            then
            pc_r <= pc_r+1;             --如果为状态 8 则 pc 寄存器加 1
        elsif(smsys(10)='1' and inlength(0)='1') then
            pc_r <= pc_r+1;             --如果为状态 10 且为多字节指令, 再加 1
        elsif(smsys(11)='1' and inlength(1)='1') then
            pc_r <= pc_r+1;             --如果为状态 11 且为三字节指令, 再加 1
        elsif(enajmp='1')              then
            pc_r <=conv_integer("00000"&
                Instruction(7 downto 5)&opd1(7 downto 0));
                --如果为短绝对跳转, 则给 pc 赋新地址值
        elsif(enljmp='1')              then
            pc_r <=conv_integer(opd1(7 downto 0)&opd2(7 downto 0));
            --如果为长绝对跳转, 则把 opd1 和 opd2 送给 pc
        elsif(ensjmp='1')              then
            pc_r<=
                pc_r+conv_integer(opd1(7)&opd1(7)&opd1(7)&opd1(7)&
                    opd1(7)&opd1(7)&opd1(7)&opd1(7)&opd1(7)&opd1(7)&opd1(7)&opd1(7));
            --如果为相对跳转, 则在原值的基础上加新值 opd1, 注意位扩展
        elsif(enjmpl='1')              then
```



```

pc_r<=
conv_integer(acc(7)&acc(7)&acc(7)&acc(7)&acc(7)&acc(7)&acc(7)
)&acc(7)&acc(7 downto 0))+conv_integer(dptr(15 downto 0));
--地址指针转移, 跳到新的地址
elsif(encjne='1') then
pc_r <= pc_r +conv_integer(opd2(7)&opd2(7)&opd2(7)&
opd2(7)&opd2(7)&opd2(7)&opd2(7)&opd2(7)&opd2(7 downto 0));
--比较转移跳转, 则在原值的基础上加新值 opd2, 注意位扩展
elsif(enin_pcl='1') then
pc_r <=
conv_integer((conv_std_logic_vector(pc_r,16)(15downto8))&dbda(7downto
0));
--程序返回的时候, 需把堆栈中的字节放回 pc
elsif(enin_pch='1') then
pc_r<=
conv_integer(dbda(7downto0)&(conv_std_logic_vector(pc_r,16)(7downto 0)));
--程序返回的时候, 需把堆栈中的字节放回 pc
end if;
end if;
end process;

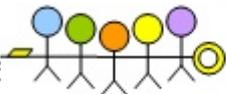
```

5.2 stack 堆栈寄存器的设计

5.2.1 stack 堆栈指针寄存器的功能介绍

堆栈是一种数据结构。数据写入堆栈称为入栈 (PUSH)。数据从堆栈中读出称之出栈 (POP)。堆栈数据操作规则: “后进先出” LIFO。即先入栈的数据由于存放在栈的底部, 因此后出栈; 而后入栈的数据存放在栈的顶部, 因此先出栈。堆栈的功能为: 堆栈主要是为子程序调用和中断操作而设立的。其具体功能有两个: 保护断点和保护现场。

因为堆栈是和常变量共享一个存储空间, 所以在访问存储器的时候也会以堆栈的值作为存储器的访问地址, MCS-51 系列的单片机堆栈指针为一个八位的寄存器。它总是指向一个空的存储器空间, 当压入一个数据后, 堆栈指针便自动增一; 当弹出一个数据后, 堆栈指针又自动减一; 所以根据堆栈的这个特点, 系统设计了堆栈指针寄存器; 它有两根触发信号线, 一根是 `enadd_stacku`; 代表堆栈



向上增一；另一根是 `enadd_stackd`，代表堆栈向下减一。每当这两个触发信号有效并且时钟上升沿到来时，堆栈指针便相应地做加一或者减一操作。

5.2.2 stack 堆栈指针寄存器的示例代码

例 5.2

—————堆栈的实现代码

```
process(clk,rst)
begin
  if(rst='1')      then
    stackadd_reg<=7;      --初始化时，堆栈的地址指针为 X"07"
  elsif(clk'event and clk='1')      then
    if(enin_ramh='1')      then
      if(dbad_ram=X"81")      then
        stack_r<=conv_integer(dbd(7 downto 0));
        --如果地址为 X"81"则向堆栈指针直接写入数值
      end if;
    elsif(enadd_stacku='1')      then
      stack_r<=stackadd_reg+1;
      --如果堆栈增使能信号有效，则堆栈值加一
    elsif(enadd_stackd='1')      then
      stack_r<=stackadd_reg-1;
      --如果堆栈减使能信号有效，则堆栈值减一
    end if;
  end if;
end process;
```

5.3 指令寄存器的设计

5.3.1 指令寄存器的功能介绍

因为 MCS-51 系列的指令是变长的，有的指令只有一个字节，有的指令有两个字节，有的指令有三个字节。因此系统所设计的指令寄存器有三个，第一个就是指令本身的寄存器，第二和第三个是立即数寄存器。它们分别为 `InstructionR`，`OperateData1`，`OperateData2`，此三个寄存器的写入数据均来源于程序 Rom 存储器。除指令寄存器的内容被直接送到指令译码器外，这三个寄存器的内容还被送



到 ram 控制器以备作存储器访问的源地址之用。指令寄存器写入的条件就为状态机为 8。而其余两个立即数寄存器的写入除了看状态机是否为 10 或 11 外，还要看该指令是否为双字节指令或者多字节指令，要根据指令译码器的译码结果而定。

5.3.2 指令寄存器实现的示例代码

例 5.3

—————指令寄存器的实现代码

```
process(clk)
begin
    if(clk'event and clk='1' and smsys(8)='1')    then
        instruction_r<=dbda_rom;    --状态为 8 则锁存指令
    end if;
end process;
```

例 5.4

—————立即数寄存器 opd1 的实现代码

```
process(clk)    --锁存指令
begin
    if(clk'event and clk='1')    then
        if((inlength(0)='1')and(smsys(10)='1'))    then
            opd1_r<=dbda_rom;
            --当状态为 10 并且指令长度为两个或三个字节的时候锁存数据
        end if;
    end if;
end process;
```

5.4 PCAA 寄存器的设计

5.4.1 pcaa 寄存器的功能介绍

pcaa 寄存器主要用于查表指令，在查表的操作中形成正确的偏移地址，主要对应的指令有两条：`movc a,@a+pc` 和 `movc a,@a+dptr`。其控制线为 `enpcaa`。主要的代码如下。



5.4.2 pcaa 寄存器实现的示例代码

例 5.5

-----pcaa 寄存器的实现代码

```
process(clk,enpcaa)
begin
    if(clk'event and clk='1')    then
        if(enpcaa="01")    then
            pcaa_r<=conv_integer(dbda(7)&dbda(7)&dbda(7)&dbda(7)&
            dbda(7)&dbda(7)&dbda(7)&dbda(7)&dbda(7 downto 0))+
            conv_integer(pc(15 downto 0));
            --如果为 movc a,@a+pc 则执行...
        elseif(enpcaa="10")    then
            pcaa_r <=conv_integer(dbda(7)&dbda(7)&dbda(7)&dbda(7)
            &dbda(7)&dbda(7)&dbda(7)&dbda(7)&
            dbda(7downto 0))+conv_integer(dptra(15 downto 0));
            --如果为 movc a,@a+dptra 则执行...
        else
            pcaa_r <=0;
        end if;
    end if;
end process;
```

5.5 ROM 地址形成电路设计

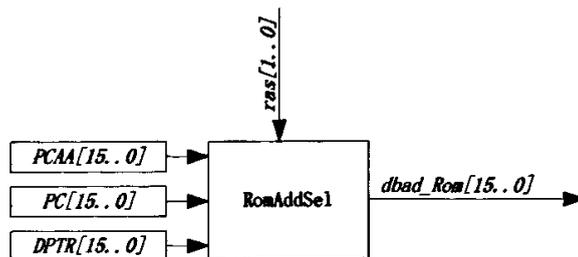
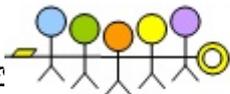


图 5-2 Rom 地址选择器原理框图



5.5.1 ROM 地址形成电路功能

如图 5-2 所示，程序 ROM 地址的来源有三个，它们分别为 PCAA,PC 和 DPTR。其中 PC 是程序地址计数器，DPTR 是数据指针寄存器，PCAA 有可能是 PC 与 dbda 之和，也有可能是 DPTR 与 dbda 之和，专用于查表指令。由 ras[1..0] 决定具体选哪个地址作为程序 ROM 的访问地址。

5.5.2 ROM 地址选择电路示例代码

例 5.6

-----rom 地址选择电路的实现代码

with ras select

```
dbad_rom(15 downto 0)<=PCAA(15 downto 0)      when "01",
--如果为 ras 为 "01" 则选择 PCAA 为 ROM 的地址
DPTR(15 downto 0)      when "10",
--如果为 ras 为 "10" 则选择 DPTR 为 ROM 的地址
PC(15 downto 0)      when others;
--其余条件则选择 PC 作为 ROM 的地址
```

5.6 本章总结

本章对 Core 核心模块中的寄存器作了详细介绍。除了堆栈指针寄存器外，其余的各个寄存器都是和程序存储器 Rom 直接相关的。其中，PC、PCAA 和 DPTR 为程序 Rom 的地址寄存器；Opd1、Opd2 和 Instruction 为程序 Rom 的数据寄存器。Instruction 为指令寄存器，Opd1 和 Opd2 为立即数寄存器。本章重点介绍 PC 寄存器模块，所有跳转指令的跳转行为都在该模块实现。

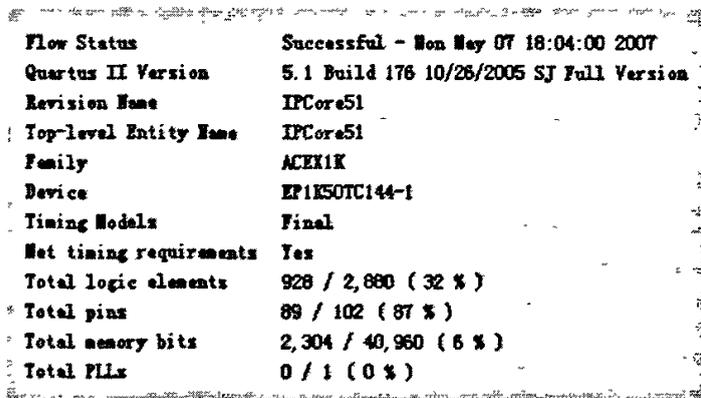
第六章 指令的测试与分析

本章主要介绍指令的测试结果,另外对开发的工具及资源使用情况也附带地加以说明

6.1 MCS-51Core 设计的相关问题

本次设计所使用的开发工具是 quartus5.1^[59-60]。quartus5.1 既支持 VHDL 等语言的输入,又支持原理图的输入,这为大规模的逻辑设计带来方便。本系统就是用图形加 VHDL 语言共同设计完成的,VHDL 代码在最底层,而上层则是用原理图实现。用 quartus5.1 做设计不方便的地方就是综合和仿真,这一点不能跟 ModelSim 比。在起初设计阶段,本人仿真一次所花费的时间不到一分钟,但是随着设计的不断加大,现在仿真一次居然需要三分多钟了!所以仿真很费时间。大型设计一般用 ModelSim 仿真比较合适,但 ModelSim 不支持 Quartus 的原理图,所以这也是本设计不用 ModelSim 仿真的原因^[61]。

6.2 FPGA 的资源分配情况



| | |
|-------------------------|------------------------------------------|
| Flow Status | Successful - Mon May 07 18:04:00 2007 |
| Quartus II Version | 5.1 Build 176 10/26/2005 SJ Full Version |
| Revision Name | IPCora51 |
| Top-level Entity Name | IPCora51 |
| Family | ACEX1K |
| Device | EP1K50TC144-1 |
| Timing Models | Final |
| Met timing requirements | Yes |
| Total logic elements | 928 / 2,880 (32 %) |
| * Total pins | 89 / 102 (87 %) |
| Total memory bits | 2,304 / 40,960 (6 %) |
| Total PLLs | 0 / 1 (0 %) |

图 6-1 FPGA 资源分配表

本设计所使用的目标芯片为 ACEX1K 系列的 EP1K50TC144-1,是 altera 公司的产品。一共耗去的逻辑门为 928,占总数的 32%。逻辑管脚本来只有 34 个,但由于需要调试的原因,一共占用了 89 个管脚,占总数的 87%。ram 有 256 个字节,程序 rom 有 32 个字节,所以它们占用的存储单元之和为 2304。占总的存

储单元的 6%。锁相环没有使用。

6.3 已实现的指令

本系统所实现的指令一共有 82 条，除了部分算术运算指令外，其余指令均得以实现。

已实现的指令见附录中的表二至表五：

6.4 典型指令测试

系统在完成整体结构以及各功能模块的设计之后，便要对各条指令进行测试。因为本设计是一个和 51 单片机兼容的微处理器系统，只有在每条指令都调通的基础上，才能正常运行由 KeilC51 编译器编译的各种程序。由于本文所实现的指令占 MCS-51 单片机总指令的 75%，指令没有完全实现，因此本文只针对特定的汇编程序指令进行测试，并给出 Quartus5.1 的仿真波形。

6.4.1 部分移位指令测试

```
org      0000h
mov     25h, #13h
mov     r0,  #25h
mov     a,   @r0
```

这是三条移位指令，不难看出，三条指令运行的结果为 acc 寄存器为 13h。

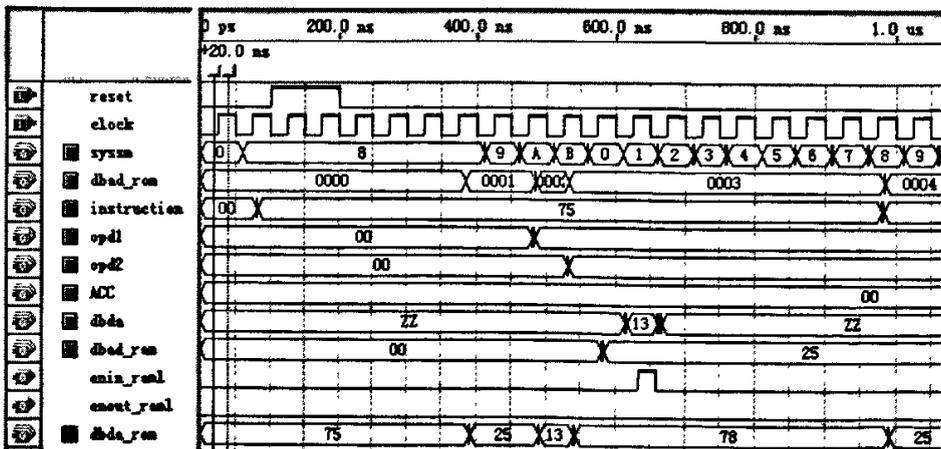


图 6-2 指令 (mov 25h, #13h) 运行时序图



如图 6-2 所示, 这是第一条指令 (`mov 25h, #13h`) 运行的时序图, 此条语句的指令机器码为 75H; 图中 `instruction` 为 75 的那一段时间就是本条语句的运行时间, 在 `ram` 写使能信号 `enin_raml` 出现脉冲那一刻, `ram` 的地址 `dbad_ram` 地址为 25H, 总线上的数据 `dbda` 为 13H, 这说明数据 13H 被写入到地址为 25H 的 `ram` 存储单元。

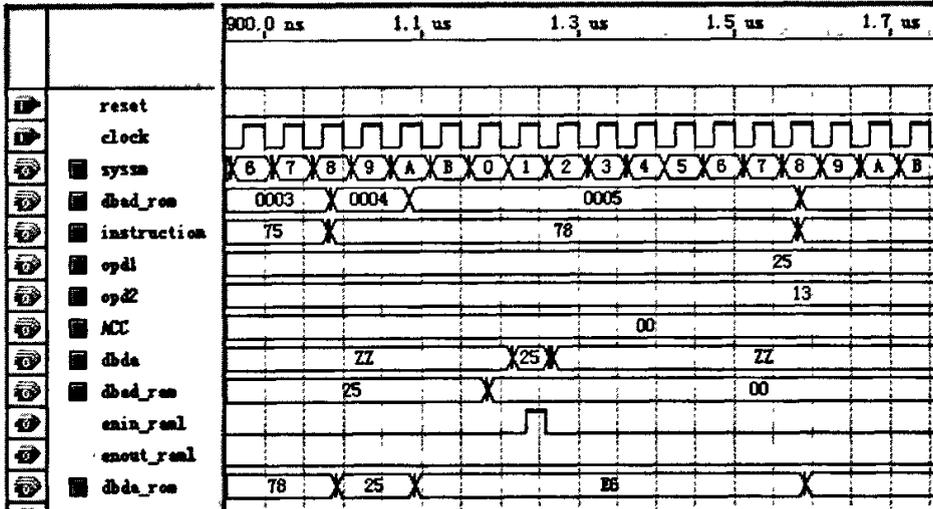


图 6-3 指令 (`mov r0, #25h`) 运行时序图

如图 6-3 所示, 这是第二条指令 (`mov r0, #25h`) 运行的时序图, 此条指令的指令机器码为 78H; 图中 `instruction` 为 78 的那一段时间就是本条指令的运行时间, 在 `ram` 写使能信号 `enin_raml` 出现脉冲那一刻, `ram` 的地址 `dbad_ram` 地址为 00H (`r0` 所在的地址就为 00H), 总线上的数据 `dbda` 为 25H, 这说明数据 25H 被写入到地址为 00H 的 `ram` 存储单元。

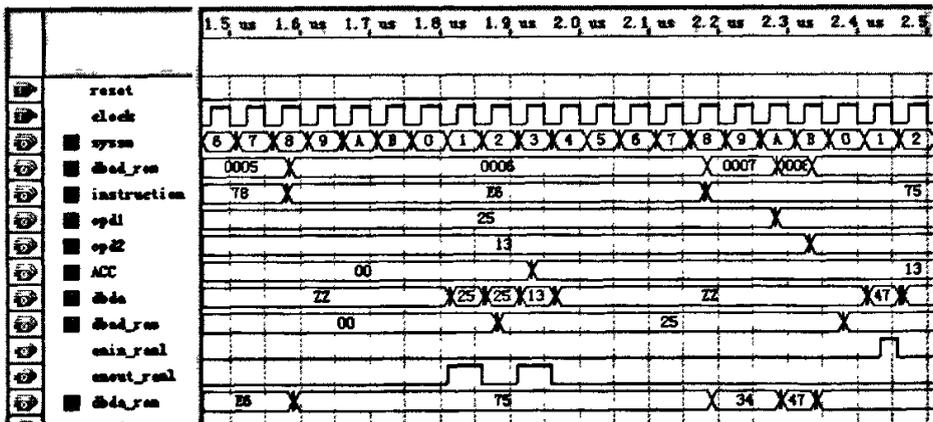


图 6-4 指令 (`mov s, @r0`) 运行时序图

如图 6-4 所示，这是第三条指令（mov a, @r0）运行的时序图，也是这三条语句运行的最终结果，此语句机器码为 E6H。从上面的程序我们可以看出，运行了这三条指令以后，ACC 中的值就应为 13H，从上图可以看出在状态机为 3 的时候 acc 的值就变为 13H，这说明指令运行无误。

6.4.2 部分跳转指令测试

```

org      0000h
mov      a, #11h
lcall   test1
mov      a, #14h
ljmp    test3
test1:  mov      a, #12h
        acall   test2
        ret
test2:  mov      a,#13h
        ret
test3:  mov      a,#15h
        mov     a,#78h
        end
    
```

这是一段由一系列跳转指令和移位指令组成的程序段，如果程序能够正常运行，则 acc 寄存器的值就应该分别为 11H；12H；13H；14H；15H；78H。

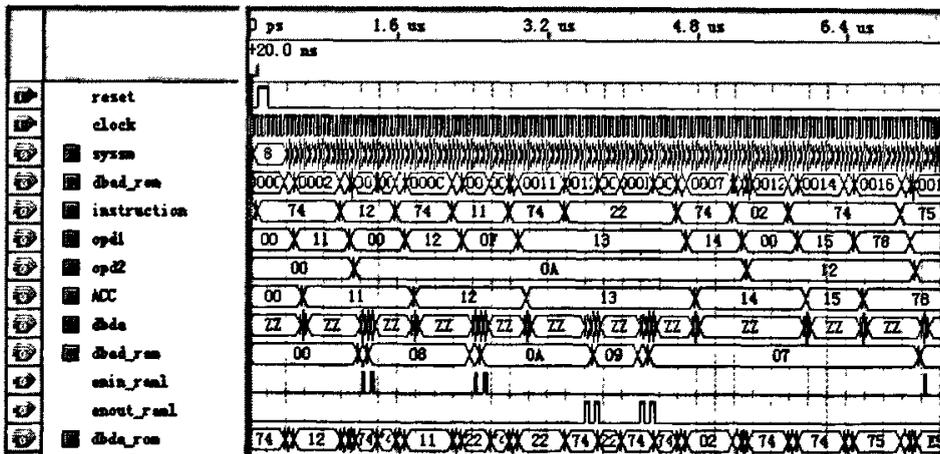


图 6-5 程序跳转指令执行时序图



如图 6-6 所示，在这一系列指令的运行过程中，acc 的值分别为 00，0F，02，0F，2F，0F，3B。仿真结果和我们编程预期得到的值一样，说明微处理器系统能够正确运行逻辑类指令。

6.4.4 部分位操作指令测试

```

org      0000h
mov      20h,    #55h
clr      20h.0
mov      a,      20h
setb     20h.1
mov      a,      20h
jb       20h.1,  test1
mov      a,      #01h
test1:  mov      a,      #02h
        mov      a,      #03h
        mov      a,      #04h
end
    
```

这是一段以位操作指令为主的程序，最开始把立即数 55H 送往 20H 的存储单元，然后对 20H 的第 0 位和 1 位分别进行了清零和置位的操作。最后以 20H.1 为判断条件作有条件的跳转。每次 20H 的存储单元变换值之后都被送往 A 寄存器，便于查看运行情况。

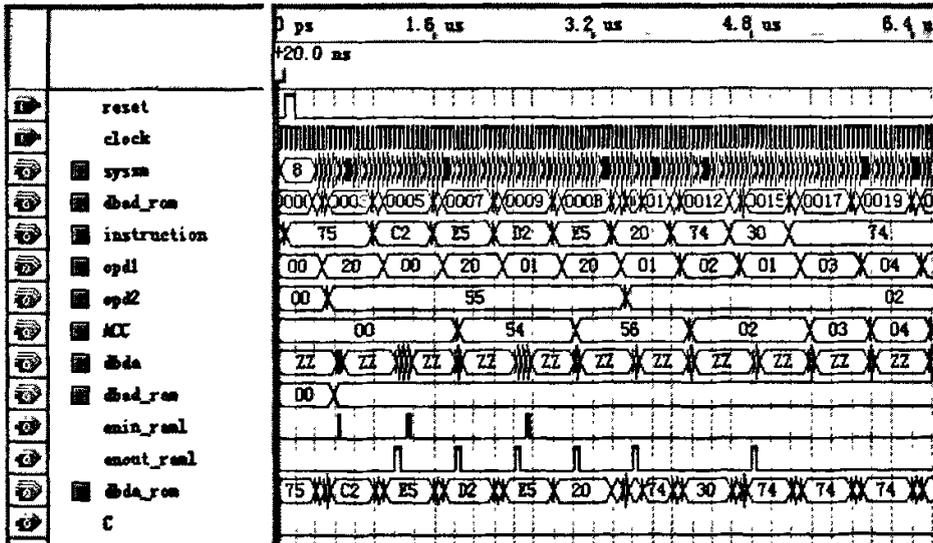


图 6-7 部分位操作指令执行情况

由图 6-7 可以看出, acc 寄存器中的值分别为 00H、54H、56H、02H、03H 和 04H。这和以上程序设计的期望值是一致的, 说明位操作指令运行正确。

6.5 本章小结

本章简要介绍了系统设计的一些问题, 重点介绍了指令的测试情况, 测试工具为 altera 公司的 quartus5.1 仿真软件。测试的步骤为: 先在 keilc51 软件中建立一个工程, 输入一段与测试指令相关的一段程序, 然后编译生成一个 HEX 文件, 再把该 HEX 文件作为程序 ROM 的初始化文件, 编译 quartus 中的工程, 最后打开仿真文件并运行, 运行完毕后观察各个寄存器中的值, 看是否与程序设计的目标值相一致。测试结果表明所有被测试指令的运行状况均符合我们的设计要求, 附录中表二至表五中的所有指令都通过了此种方式的测试。由于篇幅限制, 本章只介绍了具有代表性的比较复杂的指令的运行情况, 并给出了仿真结果图。

第七章 总结与展望

7.1 论文总结

本文设计出了能够运行 MCS-51 指令集的微处理器系统，该系统采用的是模块层次框图的设计方法。该设计的最底层用 VHDL 语言实现，上层用模块框图完成。所使用的开发工具是 Altera 公司的 QuartusII 5.1，目标芯片为 EP1K50TC144-1。

1. 设计采用自顶向下的模式进行，对微处理器系统进行了结构和功能的划分。最顶层是一个微处理器整体，第二层分为五大模块，它们分别为 Ram 模块、Rom 模块、TC 模块、SFRG 模块和 Core 核心模块。系统划分模块功能的原则是尽量把具有相同或相似功能的逻辑电路放在同一个模块之内，尽量使模块之间的连线数量尽可能的少。本文还介绍了系统中的各种资源的功能定义和控制使能定义，确定了指令执行的时序以及总线的构造方式，确定了各个寄存器对总线的读写使能控制方式。

2. 定义了 RAM 模块，ROM 模块，TC 模块和 SFRG 模块与 CORE 模块间的连接和使能控制方式，RAM 和 ROM 调用的是芯片上集成的内存模块，Ram 有 256 个字节，Rom 有 4K 的字节；TC 为时序控制寄存器模块，所有和时钟时序相关的信号都从该模块产生，此类信号包括全局时钟信号 clk、全局复位信号 rst、全局状态机信号 smsys；SFRG 模块为特殊功能寄存器模块，所有的数字外设都设计在此模块内，在本系统中数字外设包括串口、定时器、计数器和普通 I/O 口。

3. Core 模块是本文设计的重点，详细地说明了 Core 下层各个模块的功能、实现方法以及它们的连接关系。Core 下层模块有 RAM 控制器模块、指令译码器模块和 ALU 算术逻辑运算单元。RAM 控制器模块对 RAM 和所有统一编址的寄存器发出读写控制信号以及形成访问它们的地址。指令译码器负责对所有指令的译码并且发出本系统所有的控制信息，同时该模块也对指令的长度进行译码。ALU 是算术逻辑运算单元，ALU 模块有三个主要作用，第一个作用就是进行算术运算操作，第二个作用就是进行逻辑运算操作，最后一个作用就是进行位操作。

4. 本设计完成了对微处理器的指令设计，一共有 82 条，占 MCS-51 指令集的 75%左右，每条指令都通过了严格的软件仿真测试，测试工具为 altera 公司的 quartus5.1 仿真软件，测试的步骤为：先在 keilc51 软件中建立一个工程，输入一



段与测试指令相关的一段程序，然后编译生成一个 HEX 文件，再把该 HEX 文件作为程序 ROM 的初始化文件，编译 quartus 中的工程，最后打开仿真文件并运行，运行完毕后观察各个寄存器中的值，看是否与程序设计的目标值相一致。测试结果表明所有被测试程序的运行状况均符合我们设计的要求，附录中表二至表五中的所有指令都通过了此种方式的测试。仿真结果表明所有已实现的指令都符合本系统的设计要求，设计达到预期目标。

7.2 研究课题展望

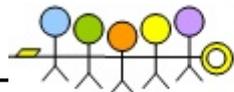
本课题成功地设计出了 8 位嵌入式 CISC (Complex Instruction Set Computer) 微处理器，它能够成功地运行 MCS-51 系列的单片机指令。该微处理器的一些功能还有进一步拓展的空间，以后的工作可以从以下几个方面着手：

1. 采用流水线的技术。本来流水线的设计是目前微处理器设计中常用的技术。但是在本设计中，由于时间问题而没有采用流水线的技术，指令只能顺序单条地执行。而流水线技术则通过重叠多条指令的执行来提高硬件的利用效率，节省了指令的执行时间。

2. 扩展微处理器的位数和设计成 RISC 型的处理器。本课题设计的是和传统 MCS-8051 兼容的微处理器，内部的数据总线和处理数据的位宽都是 8 位，指令集为 CISC 型的。以后可以在这种结构上把位宽拓展为 32 位，这样微处理器的数据处理能力能得到大大地提高。同时为了节省译码单元的资源，系统还可以设计成能够运行 RISC 型指令的处理器。这样就可以做成和 ARM9 处理器兼容的微处理器了。

参考文献

- [1] 窦振中. 单片机外围器件实用手册[M]. 北京航空航天大学, 1999
- [2] 吉训生. 三数据通道浮点加法器的 FPGA 实现[J]. 电子工程师, 2004, 30(8): 43-45
- [3] 曾繁泰等. EDA 工程的理论与实践-SOC 系统芯片设计[M]. 电子工业出版社, 2004.
- [4] 黄舒怀, 蔡敏. 超前进位加法器的一种优化设计[J]. 半导体技术
2004.8, 29(8): 65-68
- [5] 杨靛, 黄士坦. 基于可编程器件的加法器结构研究[J]. 计算机工程与应用, 2002, 24: 46-49
- [6] Abed K.H., Siferd R.E. VLSI implementations of low-power leading-one detector circuits[J]. Proceedings of the IEEE SoutheastCon 2006, 2006: 279-84
- [7] Bruguera J D. Lang Leading one Prediction with Concurrent Position Correction[J]. IEEE Trans. on Computers, 1999, 48(10): 1083-1097
- [8] 农英雄. 基于FPGA的8051的SOC设计[D]. 沈阳: 东北大学硕士论文, 2006
- [9] Michael Keating. Reuse Methodology Manual for System-on-a-Chip Design. Third Edition[M], 北京: 电子工业出版社, 2004.
- [10] 李哲英, 骆丽. SOC 与单片机应用技术的发展[J], 单片机与嵌入式系统应用, 2003.
- [11] Jean-Pierre Deschamps, Gery Jean Antoine Bioul, Gustavo D Sutter[M]. Synthesis of Arithmetic Circuits FPGA, ASIC, and Embedded Systems, 2005
- [12] Intel: Microcontroller Handbook, 1984
- [13] 向淑兰, 曹良帅. 数字信号处理器中阵列乘法器的研究与实现[J]. 微电子学与计算机, 2005, 22(10): 133-136
- [14] 常静波, 郭立. 一种 3 级流水线 wallace 树压缩器的硬件设计[J]. 微电子学与计算机, 2005, 22(1): 160-165
- [15] 王雪瑞. 32 位嵌入式 RISC 微处理器研究与设计[D]. 郑州: 中国人民解放军信息工程大学硕士论文, 2005.
- [16] 林敏, 方颖立. VHDL 数字系统设计与高层次综合[M]. 电子工业出版社, 2002
- [17] 侯伯亨, 顾新. VHDL 硬件描述语言与数字逻辑电路设计(修订版)[M]. 西



- 安电子科技大学出版社, 2001
- [18] Mark Zwolinski. Digital System Design with VHDL[M]. 北京: 电子工业出版社, 2002.
- [19] 刘哲,付宇卓.一种快速浮点加法器的设计与优化方法[J].微电子学与计算机,2004,21(12):210-213
- [20] 曾繁泰等. VHDL 程序设计[M]. 电子工业出版社, 2004.
- [21] The VHDL Golden Reference Guide[J],1995
- [22] Kevin Skahill. VHDL for Programmable Logic[M], Addison Wesley Longman Inc, 1996.
- [23] 刘清, 国海欣. IP 软核的 VHDL 设计与复用方法研究[J]. 武汉理工大学学报, 2003, 27(5):618-621.
- [24] 李丽, 高明伦, 张多利, 程作仁. 8 位 RISC 微控制器 IP 软核的设计[J]. 微电子学与计算机, 2001(3):10-18.
- [25] 刘峰, 庄奕琪, 史江一, 代国定. 高速 8 位 RISC 微控制器内核设计[J]. 微电子学, 2002, 32(6):465-468.
- [26] 赵曙光 郭万有 杨颂华. 可编程逻辑器件原理开发与应用[M]. 西安:西安电子科技大学出版社, 2000.
- [27] 朱明程. FPGA 原理及应用设计[M]. 北京: 电子工业出版社, 1994.
- [28] 蒋旋, 藏春华. 数字系统设计与 PLD 应用技术[M]. 北京: 电子工业出版社, 2001.
- [29] 孙富明. 基于多种 EDA 工具的 FPGA 设计[M]. 国防科技大学, 2003.
- [30] 金革主编. 可编程逻辑阵列 FPGA 和 EPLD[M]. 合肥: 电子科学与技术大学出版社, 1997.
- [31] 任晓东, 文博. CPLD/FPGA 高级应用开发指南[M]. 电子工业出版社, 2003.
- [32] 褚振用, 翁木云. FPGA 设计与应用[M]. 西安电子科技大学出版社, 2002.
- [33] 曾繁泰, 候亚宁. 可编程器件应用导论[M]. 清华大学出版社, 2001.
- [34] 徐欣, 孙广富, 卢启中. 基于 FPGA 的嵌入式系统设计[M]. 国防科技大学, 2003.
- [35] Kevin Banovic, Mohammed A.S.Khalid. FPGA-Based Rapid Prototyping of Digital Signal Processing Systems.
- [36] 孟宪元.可编程 ASIC 集成数字系统[M]. 北京: 电子工业出版社, 1998.
- [37] James R. Armstrong,F. Gail Gray.(李宗伯,王蓉晖,王蕾).VHDL 设计[M]: 表示与综合. 北京:机械工业出版社,2002, 469-471.
- [38] "ASIC to FPGA Design Methodology & Guidelines". Application note 311, July 2003, ver.1.0, Altera Corporation.
- [39] 周恒, 罗斯青. SOPC-基于 FPGA 的 SOC 设计策略 [J]. 山西电子技

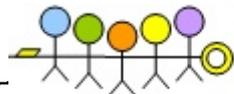


- 术, 2003(1):6-8.
- [40] 赵慧波, 刘政林等. RS-PC 译码器芯片设计的 FPGA 验证[J]. 计算机与数字工程, 2005(3):14-15.
- [41] 韩俊刚. 系统芯片的混合验证方法[J]. 西安邮电学院学报, 2002, 7(1):13-16
- [42] 罗春, 田晓明等. 系统芯片 FPGA 验证系统的软件调试环境的设计[J]. 电子器件, 2003, 26(2):208-210.
- [43] 吕涛, 李华伟等. 通用 CPU 设计中的模拟验证技术及应用[J]. 系统仿真学报, 2002, 14(12):1699-1701.
- [44] 丁元杰. 单片微机原理及应用[M]. 北京机械工业出版社, 1999.
- [45] On-Chip Memory Implementations Using Cyclone Memory Blocks, Altera, Inc., 2002.9
- [46] 王艳芳. RISC 技术发展研究[J]. 科技情报开发与经济, 2005, 15(13): 237-238.
- [47] 季特. CISC 和 RISC 微控制器[J]. 单片机与嵌入式系统应用, 2005(7):84-86
- [48] 陈瑞森, 郭东辉. 基于 CISC/RISC 混合架构的嵌入式 MCU 设计[J]. 计算机应用研究, 2006(8):194-196.
- [49] 戴梅萼. 微型计算机技术及应用[M]. 清华大学出版社, 1991
- [50] 黄海林, 沈绪榜. 基于有限状态机的 UART 设计[J]. 微电子学与计算机, 2002, 12: 52-55
- [51] 邹逢兴. 计算机硬件技术基础[M]. 北京: 高等教育出版社, 1998.
- [52] 周明德. 微型计算机系统原理及应用[M]. 北京: 清华大学出版社, 1998.
- [53] 胡越明. 计算机组成与系统结构[M]. 北京: 电子工业出版社, 2002.6.
- [54] 李压民, 计算机组成与系统结构[M]. 清华大学出版社, 2000.
- [55] John Catsoulis. Designing Embedded Hardware[M]. 北京: 中国电力出版社, 2004.
- [56] 居晓波, 李志斌, 宁兆熙, 程君侠, 王永流. 一种新型 CISC 微处理器指令译码设计方法[J]. 微电子学, 2003, 33(2):154-156.
- [57] 方俊锋, 杨银堂. CISC 微处理器嵌入式内核的设计[J]. 微电子学, 2003, 33(4):355-361.
- [58] 居水荣, 王效, 万海涛. RISC 和 CISC 两种架构 MCU 的比较[J]. 微电子技术, 2003, 31(3):1-7.
- [59] Quartus II Help Version 3.0
- [60] Configuring Altera FPGAs, Altera, Inc., 2002.9
- [61] Model Technology. Modelsim SE User's Manual[J], 2003.9

附录

表一 本系统涉及的微操作:

| 编号 | 信号线名称 | 相应执行动作 | 相关寄存器/模块 | 注解 |
|----|--------------|----------------------|----------------------|---------------|
| 1 | enin_acc | dbda>acc | acc/dbda | 向 acc 写数 |
| 2 | enout_acc | acc>dbda | acc/dbda | 从 acc 读数 |
| 3 | enin_trg1 | dbda>trg1 | trg1/dbda | 向 trg1 写数 |
| 4 | enout_trg1 | trg1>dbda | trg1/dbda | 从 trg1 读数 |
| 5 | enin_raml | dbda>raml | raml/dbda | 向 raml 写数 |
| 6 | enout_raml | raml>dbda | raml/dbda | 从 raml 读数 |
| 7 | enin_ramh | dbda>ramh | ramh/dbda | 向 ramh 写数 |
| 8 | enout_ramh | ramh>dbda | ramh/dbda | 从 ramh 读数 |
| 9 | enin_dptr | opd>dptr | opd/dptr | 向 dptr 写 16 位 |
| 10 | enin_trg2 | dbda>trg2 | trg2/dbda | 向 trg2 写数 |
| 11 | enout_trg2 | trg2>dbda | trg2/dbda | 从 trg2 读数 |
| 12 | enout_opd1 | opd1>dbda | opd1/dbda | 从 opd1 读数 |
| 13 | enout_opd2 | opd2>dbda | opd2/dbda | 从 opd2 读数 |
| 14 | enout_rombuf | rombuf>dbda | rombuf/dbda | 从 rom 读数 |
| 15 | enout_pch | pch>dbda | pch/dbda | 从 pch 读数 |
| 16 | enout_pcl | pcl>dbda | pcl/dbda | 从 pcl 读数 |
| 17 | enin_pch | dbda>pch | pch/dbda | 向 pch 写数 |
| 18 | enin_pcl | dbda>pcl | pcl/dbda | 向 pcl 写数 |
| 19 | enadd_opd1 | opd1>dbad_ram | opd1/dbda_ram | opd1 形成地址 |
| 20 | enadd_opd2 | opd2>dbad_ram | opd2/dbad_ram | opd2 形成地址 |
| 21 | enadd_Ri | instruction>dbad_ram | instruction/dbad_ram | Ri 形成地址 |
| 22 | enadd_Rj | instruction>dbad_ram | instruction/dbad_ram | Rj 形成地址 |
| 23 | enadd_dbda | dbda>dbad_ram | dbda/dbad_ram | dbda 形成地址 |
| 24 | enadd_bit | opd1>dbad_ram | opd1/dbad_ram | opd1 形成地址 |
| 25 | enadd_stacku | stack>dbad_ram | stack/dbad_ram | 堆栈形成地址 |
| 26 | enadd_stackd | stack>dbad_ram | stack/dbad_ram | 堆栈形成地址 |
| 27 | enxchd | trg1[3..0]↔acc[3..0] | acc/trg1 | 交换 |
| 28 | enswap | acc[7..4]↔acc[3..0] | acc | acc 交换 |
| 29 | enopl | acc↔acc | acc | acc 取反 |
| 30 | enclr | acc↔acc | acc | acc 清零 |
| 31 | enrl | acc↔acc | acc | acc 左移 |
| 32 | enrr | acc↔acc | acc | acc 右移 |
| 33 | enrlc | acc↔acc | acc | acc 进位左移 |
| 34 | enrrc | acc↔acc | acc | acc 进位右移 |



本系统涉及的微操作 (续表一):

| 编号 | 信号线名称 | 相应执行动作 | 相关寄存器/模块 | 注 解 |
|----|----------|---------------------------------|---------------------|------------|
| 35 | enanl | $trg1 < trg1 \text{ and } dbda$ | trg1/dbda | 与 |
| 36 | enorl | $trg1 < trg1 \text{ or } dbda$ | trg1/dbda | 或 |
| 37 | enxrl | $trg1 < trg1 \text{ xrl } dbda$ | trg1/dbda | 异或 |
| 38 | enajmp | $pc < instruction \& opd1$ | pc/instruction/opd1 | 短跳转 |
| 39 | enljmp | $pc < opd1 \& opd2$ | pc/opd1/opd2 | 长跳转 |
| 40 | enjmp | $pc < dptr + acc$ | pc/acc/dptr | 查表 |
| 41 | ensjmp | $pc < pc + opd1$ | pc/opd1 | 相对跳转 |
| 42 | encjne | $if(a \neq 0) pc < opd2$ | pc/opd2 | 条件跳转 |
| 43 | enclrc | $c < '0'$ | acc | c 清零 |
| 44 | ensetbc | $c < '1'$ | acc | c 置一 |
| 45 | enclpc | $c < not \ c$ | acc | c 取反 |
| 46 | enclrb | $b < '0'$ | trg1 | 任意位 b 清零 |
| 47 | ensetbb | $b < '1'$ | trg1 | 任意位 b 置一 |
| 48 | enclpb | $b < not \ b$ | trg1 | 任意位 b 取反 |
| 49 | enmcb | $c < b$ | acc/trg1 | b 数据送 c |
| 50 | enmcb | $b < c$ | acc/trg1 | c 数据送 b |
| 51 | enanlcb | $c < c \text{ and } b$ | acc/trg1 | c 与 b 送 c |
| 52 | enorlcb | $c < c \text{ or } b$ | acc/trg1 | c 或 b 送 c |
| 53 | enanlcnb | $c < c \text{ and } not \ b$ | acc/trg1 | c 与 b 非送 c |
| 54 | enorlcnb | $c < c \text{ or } not \ b$ | acc/trg1 | c 或 b 非送 c |
| 55 | enjc | $if(c) pc < pc + opd1$ | c/pc/opd1 | 位条件跳转 |
| 56 | enjnc | $if(not \ c) pc < pc + opd1$ | c/pc/opd1 | 位条件跳转 |
| 57 | enjb | $if(b) pc < pc + opd1$ | b/pc/opd1 | 位条件跳转 |
| 58 | enjnb | $if(not \ b) pc < pc + opd1$ | b/pc/opd1 | 位条件跳转 |
| 59 | enjbc | $if(b) pc < pc + opd1; b < '0'$ | b/pc/opd1 | 位条件跳转 |



表二 已实现的数据转移指令 (24 条):

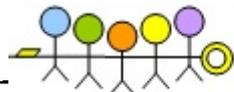
| 机器码 | 汇编格式 | 机器码 | 汇编格式 |
|-------|--------------------|-------|-------------------|
| 74 | mov a, #data | E5 | mov a, direct |
| E8-EF | mov a, Ri | E6-E7 | mov a, Rj |
| 83 | movc a, @a+pc | 93 | movc a, @a+dptr |
| F8-FF | mov Ri, a | 78~7F | mov Ri, #data |
| 76~77 | mov @Rj, a | F5 | mov direct, a |
| 88~8F | mov direct, Ri | A8-AF | mov Ri, direct |
| 86~87 | mov direct, @Rj | A6-A7 | mov @Rj, direct |
| 85 | mov direct, direct | C8-CF | xch a, Ri |
| C6-C7 | xch a, Rj | C5 | xch a, direct |
| C0 | push direct | D0 | pop direct |
| D6-D7 | xchd a,@Rj | 90 | mov dptr, #data |
| F6-F7 | mov @Rj, a | 75 | mov direct, #data |

表三 已实现的程序跳转指令 (17 条):

| 机器码 | 汇编格式 | 机器码 | 汇编格式 |
|-------|---------------------|-------|----------------------|
| 01~E1 | ajmp adr11 | 02 | ljmp adr16 |
| 80 | sjmp rel | 73 | jmp @a+dptr |
| 60 | jz rel | 70 | jnz rel |
| B5 | cjne a, direct, rel | B4 | cjne a, #data |
| B8-BF | cjne Ri, #data, rel | B6-B7 | cjne @Ri, #data, rel |
| D8-DF | djnz Ri, rel | D5 | djnz direct, rel |
| 11~F1 | acall adr11 | 12 | lcall adr16 |
| 22 | ret | 32 | reti |
| 00 | nop | | |

表四 已实现的逻辑运算指令 (24 条):

| 机器码 | 汇编格式 | 机器码 | 汇编格式 |
|-------|-------------------|-------|-------------------|
| 58~5F | anl a, Ri | 48~4F | ori a, Ri |
| 68~6F | xrl a, Ri | 56~57 | anl a, @Rj |
| 46~47 | ori a, Rj | 66~67 | xrl a, @Rj |
| 55 | anl a, direct | 45 | ori a, direct |
| 65 | xrl a, direct | 52 | anl direct, a |
| 42 | ori direct, a | 62 | xrl direct, a |
| 54 | anl a, #data | 44 | ori a, #data |
| 64 | xrl a, #data | 53 | anl direct, #data |
| 43 | ori direct, #data | 63 | xrl direct, #data |
| F4 | cpl a | E4 | clr a |
| 23 | rl a | 03 | rr a |
| 33 | rlc a | 13 | rrc a |



表五 已实现的位操作指令 (17 条):

| 机器码 | 汇编格式 | 机器码 | 汇编格式 |
|-----|--------------|-----|--------------|
| B3 | cpl c | C3 | clr c |
| D3 | setb c | A2 | mov c, bit |
| 92 | mov bit, c | B2 | cpl bit |
| C2 | clr bit | D2 | setb bit |
| 82 | anl c, bit | 72 | ori c, bit |
| B0 | anl c, /bit | A0 | ori c, /bit |
| 40 | jc rel | 50 | jnc rel |
| 20 | jb bit, rel | 30 | jnb bit, rel |
| 10 | jbc bit, rel | | |



致 谢

本文得以顺利完成，首先要特别感谢导师曹建教授。导师渊博的知识、严谨的作风、兢兢业业的工作态度，都深深地感染和教育了我。师从曹建教授，是我一生的荣幸。在此，谨向导师致以衷心的感谢和诚挚的敬意。

在攻读学位期间，本人还得到中南大学物理科学与技术学院领导和老师的亲切关怀和热心指导，特别是刘雄飞、盛利元、孙克辉、邓宏贵、丁家峰、王会海、许雪梅、吴建好以及光电所的其他老师给了我许多帮助，在此我也一并表示衷心的感谢。此外，感谢中南大学物理科学与技术学院提供本人良好的学习与科研条件。

感谢我的同学陈志敏、王文广、张萌、彭广、曹莉凌以及各位师弟师妹对我学习和生活上的帮助。

感谢所有曾给予本人帮助和关心的亲人和朋友。最后，感谢百忙之中抽出时间对本论文进行审阅的专家、学者和老师。

再次感谢所有支持过我，帮助过我的人。祝你们事业成功，生活愉快，家庭幸福。

雒 雄

二零零七年 五月 于 中南大学