

专业学位硕士学位论文

基于 MIPS 处理器的 TPM 安全芯片设计

Design of TPM Security Chip Based on MIPS Processor

作者姓名: 李佳琪

工程领域: 集成电路工程

学号: 31409032

指导教师: 张建伟

完成日期: 2016. 4. 28

大连理工大学

Dalian University of Technology



Y3056774

大连理工大学学位论文独创性声明

作者郑重声明：所呈交的学位论文，是本人在导师的指导下进行研究工作所取得的成果。尽我所知，除文中已经注明引用内容和致谢的地方外，本论文不包含其他个人或集体已经发表的研究成果，也不包含其他已申请学位或其他用途使用过的成果。与我一同工作的同志对本研究所做的贡献均已在论文中做了明确的说明并表示了谢意。

若有不实之处，本人愿意承担相关法律责任。

学位论文题目：基于MIPS处理器的TPM安全芯片设计
作者签名：李信琪 日期：2016 年 6 月 12 日

摘要

计算机为我们的生活带来了极大的便利，但同时也给我们带来了很多安全问题。在个人隐私方面，我们存储于电脑、手机中的信息可以轻易地被盗取；在财产方面，越来越多的不法分子通过网络获得用户的支付密码。计算机安全问题甚至可以给国防带来巨大的隐患。通常，解决计算机安全问题的手段是靠软件来进行防卫，但仅靠软件的防卫手段有其弊端，难以应对花样繁杂的攻击形式，也给计算机的性能带来很大的负担。而以可信计算为基础的 TPM 安全芯片是依靠硬件来对计算机进行保护的有效措施，不仅具有更好的安全性，而且其执行比软件更快，是当前有关计算机安全的研究领域中的热点。本文正是基于这样的思想设计一款 TPM 安全芯片。TPM 芯片一般是以 SOC 的形式具体实现的，本文的设计也是以 MIPS 处理器为核心搭建一个 SOC，通过 AMBA 总线挂载一些 TPM 协议中规定的模块，以此来作为 TPM 芯片的具体实现。

本文所完成的模块有四个。分别是 MIPS 处理器、DES 加密模块、AES 加密模块、SHA-1 加密模块。

关于 MIPS 处理器的设计，本文通过查阅资料、调查市场已有产品等方式，了解并借鉴了一些已有产品的成功经验。在此基础上，对 MIPS 处理器从基本框架着手开始设计，对处理器的流水线进行了合理的分配，并设计了冒险与异常的处理、中断处理等机制。最后完成了对 MIPS 处理器的设计，并配合编写的汇编器对其进行功能验证。

DES 与 AES 都是目前在国际上十分流行的加密算法，所以在本文所设计的 TPM 安全芯片中同时引入 DES 和 AES 两种加密算法作为对称加密引擎。本文根据 DES 与 AES 两个加密算法分别设计了 DES、AES 两个加密算法的工作内核，并根据 TCG 有关于 TPM 的协议要求，本文将 DES 模块与 AES 模块都设计成了密文反馈模式。

SHA-1 是安全哈希算法之一，在 TPM 协议中可以作为哈希引擎来使用。本文设计了 SHA-1 算法的运算内核，并对其进行验证，保证其基本功能正确。继而 SHA-1 内核与 AHB 总线的接口，使其能够在最终搭建的 SOC 中工作，并配合 AHB 总线对 SHA-1 内核与接口一同进行了总体的时序验证。

本文使用 verilog 语言编写了以上模块的 RTL 代码，并配合 AMBA 总线的 VIP 进行验证，保证所设计的模块功能基本正确。

关键词：可信计算；TPM；SOC；MIPS；DES；AES；SHA

Design of TPM Security Chip Based On MIPS Processor

Abstract

Computer not only makes many convenience to our live, but also bring about many problems about security. In terms of personal privacy, information stored in computers and mobile-phones can be got very easily. In terms of property, criminals can get our password through internet. Problems of computer security can even make hidden troubles to national defense. We generally use software to solve the problem of computer security, which has some malpractices. TPM security chip is an efficient solution for the problem of computer security. It not only improves security, but also has high speed of execution. TPM is generally in the form of SOC. The design of this thesis is a SOC with a MIPS processor as the core of the system, and a AMBA bus with some other modules specified by the specification of TPM.

The modules completed in this thesis include MIPS processor, DES encryption algorithm, AES encryption algorithm, and SHA-1 algorithm.

In this thesis, the architecture of MIPS processor is based on successful experience of some existing products and the pipeline is divided legitimately. The mechanisms to deal with exception, adventure and interrupt are added in architecture. Finally, the processor's function is verified with assembler.

DES and AES are currently popular encryption algorithms, which we choose as the symmetric encryption engine of TPM. In this thesis, the core of DES and AES are designed based on the algorithms of DES and AES. The work mode of the two modules is CFB.

SHA-1 algorithm is one of the security hash algorithms. In the thesis, the SHA-1 algorithm core is designed as the hash engine, whose interface accords with the AHB bus specification. Finally, the module's function and timing are verified.

All modules included in this paper are designed with Verilog, and verified with AMBA VIP. All modules functions are correct.

Key Words: Trusted Computing; TPM; SOC; MIPS; AES; DES; SHA

目 录

摘要	I
Abstract	II
1 绪论	1
1.1 选题背景及研究意义	1
1.2 研究现状	2
1.3 主要工作介绍与章节安排	3
2 TMP 安全芯片	4
2.1 可信计算	4
2.1.1 可信计算的概念	4
2.1.2 可信计算的发展	5
2.2 TCG 的信任链介绍	6
2.3 TPM 安全芯片	8
2.3.1 TPM 的结构	8
2.3.2 Hash 引擎	9
2.3.3 非对称加密引擎	10
2.3.4 对称加密引擎	11
3 MIPS 处理器	13
3.1 MIPS32 指令结构	13
3.2 MIPS 中的寄存器	14
3.3 MIPS 微处理器设计	15
3.4 MIPS 微处理器核结构	16
3.4.1 算数运算指令数据通路的搭建	16
3.4.2 存取指令数据通路的搭建	20
3.4.3 分支指令、跳转指令的搭建	22
3.4.4 流水线冒险处理与异常处理	25
3.5 模块的实现	27
4 DES 加密	35
4.1 初始置换与逆初始置换	35
4.2 乘积变换	36
4.3 轮密钥生成	41

4.4 3DES	43
4.5 模块的实现.....	43
5 AES 加密	49
5.1 加密运算.....	49
5.1.1 轮密钥加计算.....	50
5.1.2 S 盒替换操作.....	51
5.1.3 行位移变换.....	52
5.1.4 列混合变换.....	52
5.2 解密运算.....	52
5.2.1 逆 S 盒替换.....	53
5.2.2 逆行位移变换.....	54
5.2.3 逆列混合变换.....	55
5.3 密钥编排.....	55
5.4 模块的实现.....	57
6 SHA-1 算法.....	62
6.1 SHA-1 的加密过程.....	62
6.2 模块的实现.....	64
结 论	68
参 考 文 献	70
致 谢	72
大连理工大学学位论文版权使用授权书	73

1 绪论

1.1 选题背景及研究意义

随着电子技术、信息技术、计算机技术等相关技术的崛起，人类社会同计算机网络的结合愈加紧密。计算机网络给我们日常生活带来极大的便利，也为科学研究、国家防卫等方面带来了前所未有的进步和突破。但同时，计算机网络的普及和应用也给我们带来了诸多的问题。

首先，我们的个人隐私不能得到很好的保障。我们的存放于 PC、手机中的隐私信息可以很轻易地被他人窃取、窥探，甚至恶意破坏、传播。即使我们所使用的网络环境比较安全，有足够安全的防火墙、防入侵软件保护，但若我们不慎遗失或者被他人窃取了电子设备，那么存储于其中的信息也可以轻易地被他人得到。同样严重的是，我们的财产安全也正遭受着来自计算机和网络的威胁。近年来，高科技犯罪的新闻层出不穷，而多数受害者都是因为使用设备的环境不安全和自身安全意识欠缺才使得犯罪分子有机可乘。抛开使用者安全意识欠缺的方面，在计算机使用环境的安全得不到保障之前，把财产放在网络账户中存储真的不是一个明智的选择。当然，比以上更加严重的事情还有。计算机和网络的不安全甚至能够扰乱社会应有的秩序。在商业上，各种盗版软件猖獗不止，使得凭借自身才智和努力取得成就的开发者得不到应有的回报；不怀好意的人可以肆意篡改网页上的信息，给社会带来不良影响；甚至于，如果不法分子攻占、利用了主要的媒体设备时，可能会给国家安全造成威胁。最重要的是，计算机的安全隐患甚至能给国防带来威胁。美国政府臭名招呼的棱镜计划也给全世界敲响了警钟——如果国家不能保障自己的通讯网络自主可控，可以说毫无国防可言。所以，提升我国在计算机网络、信息安全等领域的技术已经成为了关乎国家安全的因素。计算机网络的应用给我们带来的如此多的隐患，但同时也给我们带来了前所未有的机遇。我们难以从社会中割舍计算机网络，所以只能积极地面的隐患，发展我们自主可控的计算机网络技术，利用其积极的一面为社会造福。

芯片是构成电子设备的基本原件，同时也是我国曾经被用力扼住的喉咙。国内的芯片设计、制造技术同国际顶尖水平还相距甚远，一些高端芯片目前还只能依赖进口。到了 2014 年，芯片已经超越了石油成为了我国第一大进口商品，进口花费超过 2000 亿美元，约合 1.8 万亿人民币。如此大宗的进口势必带来安全的隐患，而且，一些科研、国防等重要领域的高端芯片，面临着即使有钱也买不到、即使买到了也不敢用的尴尬境地。如果底层的芯片安全都不能做到自主可控，则上层的操作系统、网络、应用等

都面临着严重的安全隐患。因此，提高我国集成电路尤其是安全芯片相关领域的发展水平成为了当前亟待解决的问题。

针对上述问题，我国工业和信息化部经国务院同意于 2014 年 6 月 24 日颁布了《国家集成电路产业发展推进纲要》，将会规模投入人力、财力来发展我国的集成电路产业，目标是实现 2020 年开始使国内集成电路产业水平逐渐缩小与国际先进水平的差距，2030 年使国内集成电路产业水平达到国际先进水平。

在安全芯片领域，目前基本分为两大阵营，即国际上的 TPM 阵营和国内的 TCM 阵营。这两大阵营并不对立而是相互借鉴、吸取对方的优点趋于合同的。如果计算机设备中能够普及安全芯片，则对于我们的日常隐私安全、商业盗版等方面带来极大的改善，而针对安全芯片的发展也能为国防、科研带来保障。

1.2 研究现状

1985 年开始，美国国防部提出了一系列安全指导文件，命名为“彩虹系列安全指导文件”^[1]。包括了《可信计算机系统评价准则》(Trusted Computer System Evaluation Criteria, TCSEC)、《可信网络解释》(Trusted Network Interpretation, TNI)和《可信数据库解释》(Trusted Database Interpretation, TDI)等文件。彩虹系列安全指导文件中对“可信计算机(Trusted Computer)”和“可信计算基(Trusted Computing Base)”做出了明确的规范，从此拉开了关于可信计算的帷幕。

2003 年，14 家企业联合成立了可信计算工作组 (Trusted Computing Group, TCG)^[2]。这 14 家企业包括 AMD、Microsoft、Intel、HP、Sony 等跨国大公司。TCG 是一个面向工业界、非盈利的组织，它的目标是以研究、制定工业界的可信计算标准来维护计算机平台的计算环境安全。它继承了一些已有的规范，并且进一步提出了自己的规范。

TCG 提出的一系列规范中，关于可信平台模块 (Trusted Platform Module, TPM) 的协议是对计算机安全最底层的、芯片层次的保障。TPM1.2 协议被誉为计算机安全的分水岭^[3]。

国内在可信计算方面的研究紧跟国际先进水准的脚步。

自 2003 年，武汉瑞达公司开发出了国内一块可信计算平台安全芯片起，中国国内的多家相关领域公司都展开了关于安全芯片的研究，并且推出了可信计算相关的一些列产品，包括安全芯片、可信计算机、移动设备等等。而关于安全芯片的研究，国内的研究机构、学者们吸取了国际上的先进经验并加以改进，提出了我国自主可控的安全芯片技术，在 2008 年，国家密码管理局发布的《可信计算密码支撑平台功能与接口

规范》，成为了我国独立提出的可信计算规范。

如图 1.1 所示，均为国内相关科技公司推出的产品。其中图（1）为武汉瑞达推出的遵循《可信计算密码支撑平台功能与接口规范》的 J3210 安全芯片，图（2）为带有 TPM 安全芯片的联 YOGA2 13-IFI 型号 PC 机，图（3）为华为 Mate8 型号手机，应用了 Trustzone 安全芯片技术。

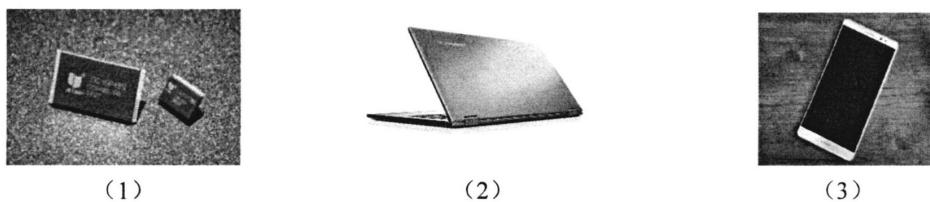


图 1.1 一些安全芯片相关的产品

Fig.1.1 Some security chip related products

1.3 主要工作介绍与章节安排

本文的主要工作为，搭建一个符合 TPM2.0 版本规范的 SOC 系统，独立地设计其中所需要的模块，包括 MIPS 处理器模块、AES 加密算法模块、DES 加密算法模块、SHA-1 加密算法模块。此外，还将整合其 SOC 系统，为以后的研究工作打下基础。

本文的章节安排如下：

第一章为绪论，主要介绍选题的背景和意义以及课题的发展现状，最后对全文的结构进行概述。

第二章对可信计算的基本概念、发展历史、应用领域等进行简单的介绍，并着重对 TPM 安全芯片的功能、结构、其内部模块的功能等进行介绍。

第三章介绍 MIPS 处理器，主要说明 MIPS 处理器的指令结构、MIPS 处理器的硬件结构和其运行原理。

第四章介绍 DES 和 3DES 加密算法，主要说明 DES 加密算法的应用和算法介绍。

第五章介绍 AES 加密算法，主要说明 AES 加密算法的历史、应用以及算法介绍。

第六章介绍 SHA-1 加密算法，主要说明了 SHA-1 的加密方法以及应用场合。

最后对全文做出总结和概括。

2 TMP 安全芯片

2.1 可信计算

2.1.1 可信计算的概念

目前，计算机已经渗透到了这个世界的各个领域。我们的日常生活离不开计算机、科学研究离不开计算机、工业生产离不开计算机，乃至国防系统也离不开计算机。随着人类活动与计算机结合的愈加紧密，计算机安全的问题也随之而来。如果把计算机网络比作一个虚拟的世界，那么这个世界现在大概是一片游荡着各种凶猛野兽的荒蛮大陆，我们所使用的计算机随时都面临着被侵犯的危险。而对于可信计算的研究，就好比要在这野蛮的大陆上为我们的计算机建立坚实的城墙和堡垒，从而在目前的计算机世界里建立一个安全、有序的国度。

业界对于什么是“可信计算”目前还没有一个统一的、明确的规定。虽然不同流派之间对于“可信计算”的概念略有不同，但显然的是，大家都有一个共同的目的，就是通过建立一种信任的机制来保护我们的计算机系统。

“可信”的理念源于社会学，可以将其解释为“可以信赖的”，也就是可以信任和依赖的。把这个“可信”的含义赋予我们的计算机行为，我认为可以有如下几层含义^[4-7]：

计算机行为的安全性。一个可信的计算机网络，它的行为应该是具有安全性的，它应该是既能保证自己的行为安全，又能对来自外界的安全隐患进行防范。

计算机行为的可靠性。假如我命令计算机给我播放一段音乐，但是计算机执行的却是关机，这样的计算机可能不会造成严重的安全问题，但它显然是不可靠的。计算机是可信的，那计算机的行为应该具有可靠性。

计算机行为的实用性。如果一个计算机不具有实用性，就好比我们有一个帮手，他会很多技能，但是我们却不需要，同样也是难以胜任任何工作的，即使他能可靠又安全地完成他能力之内的工作。

以上是我个人对计算机“可信”概念的理解。而不同的组织和学者对于“可信”都有其自己的定义。可信计算组织（Trusted Computing Group, TCG）精练地把“可信”理解为“行为的可预测性”，并且还在 TPM2.0 协议中举了生动的例子：我们会预测一个银行它的行为看起来像银行，一个贼的行为看起来像一个贼。武汉大学的张焕国老师则给出了一种通俗的解释：可信≈可靠+安全。仔细揣摩不难发现，大家对于“可信”的定义和理解虽然略有偏差，但其实业界整体的趋势是趋于合同的。

“可信计算”可以理解为以“可信”为根基建立的一种保护机制，归根结底还是从我们人类社会中抽象出来的。

信任是人类社会中人与人之间维持联系的一个很重要的因素。例如，甲可以选择信任乙或者不信任乙，若甲信任乙，甲可以完全信任乙或者只信任乙的某些行为。再或者，甲信任乙，乙信任丙，那么此时甲选择信任丙的理由就更多一些，也就是甲更有可能信任丙。又或者，甲原来信任乙，但是乙某些行为导致甲对乙的看法发生了变化，不再信任乙。以上这些行为，都是我们人类社会中信任存在形式和变化方式的体现。那么在计算机世界中，信任也具有类似的模样，具体表现为如下几个方面^[8]：

信任的度。一个实体对另外一个实体的信任程度是可以划分等级的，度为上限即完全信任，度为下限即不信任。

信任的传递。假设甲信任乙，乙信任丙，则甲就可以信任丙。但信任的度在传递过程中可能会有减损，比如甲完全信任乙，乙完全信任丙，但甲可能只信任丙的某些行为，甚至不信任丙。

信任的监测。假设甲信任乙，并且甲会记录乙的行为，并对其进行评估，若乙做出某些敏感行为，则甲会对乙的信任度减损，可能最终导致甲不再信任乙。

信任的单向性。即实体甲信任乙，但乙不一定信任实体甲。

关于信任理论，已经有基于概率统计的信任理论和基于模糊数学的信任理论等^[9-12]，形成了一个研究方向，这里不再赘述。

可信计算所建立的机制，正是通过信任的传递和评估来保护我们的计算机系统。简单说来即是，首先有一个信任根，这个信任根会信任可信硬件，可信硬件又会信任可信操作系统，可信操作系统信任可信网络和可信应用。这样，通过一层一层的传递，把我们的计算机系统保护在“可信”的墙壁之下，不受外界“不可信”的因素侵犯。

2.1.2 可信计算的发展

1985 年开始，美国国防部提出了一系列安全指导文件，命名为“彩虹系列安全指导文件”。

2003 年，14 家企业联合成立了可信计算工作组（Trusted Computing Group, TCG）。

2006 年，欧洲的一些机构和组织联合启动了“开放式可信计算(Open Trusted Computing)”的研究计划^[13]。

2007 年，Microsoft 启动了“Palladium”计划^[14]。Palladium 计划提出了 Microsoft 自己的可信计算定义。同时，Intel 配合微软在奔腾处理器中加入了支持 Palladium 的硬件技术。

2007 年，联想、中兴、同方、方正科技、中科院软件所、长城电脑、武汉瑞达、

北京兆日、成都卫士通、江南所、吉大正元、国防科大这十二家机构和公司联合宣布推出中国自主可信计算系列产品，包括 TCM 可信密码芯片、可信计算机、可信加密机等^[15]。同年 12 月，国家密码管理局以国内密码算法为基础，结合国内安全需求与产业市场，并借鉴国际先进的可信计算思想，发布了《可信计算密码支撑平台功能与接口规范》，此规范是我国自主创新的可信计算技术规范。

2008 年，由沈昌祥院士提议，上海中标、西安西电捷通、长城计算机、国家信息中心信息安全研究与服务中心、北京工业大学、武汉大学等 16 家单位联合发起，成立了中国可信计算联盟（Chinese Trusted Computing Union, CTCU）^[16]。同年 12 月，中国可信计算密码专项组更名为中国可信计算工作组。

到目前为止，中国在可信计算领域已经取得了丰富的成果，自 2003 年以来，武汉瑞达、北京兆日等公司已经推出了多款可信计算产品，联想、同方、长城、方正等公司也推出了自己的可信计算机产品。可以说，中国的可信计算研究水平已经走在了世界前列。

2.2 TCG 的信任链介绍

TCG 的信任模型中只考虑信任和不信任两种情况，忽略了信任的度，即以二值化的形式表达信任。由此也可以更容易理解，在 TCG 的信任链中，信任的传递是没有信任损失。另外，在 TCG 的信任链中以数据的完整性充当可信性。

TCG 的信任链从信任根开始逐渐由内而外传播开的。图 2.1 所示为 TCG 规定的信任链模型。

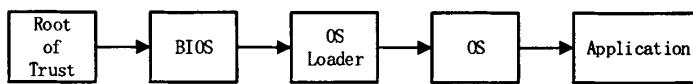


图 2.1 TCG 的信任链

Fig. 2.1 Trusted chain of TCG

在 TCG 的信任链中，由可信平台模块（Trusted Platform Module，TPM）充当整个信任链的信任根^[17]。信任根有以下三个：

可信存储根（Root of Trust for Storage，RTS）^[18]：TPM 需要对一些敏感数据提供保护，例如非对称加密算法中的私钥。所谓保护指的是包括软件和硬件在内的保护，也就是说 TPM 除了能够保护其内部的数据不受软件手段入侵之外，还应该具有一定的抵御物理攻击的能力。为了保证数据的安全，TCG 规定 TPM 不能与其他模块共用硬件资源。组成 TPM 的硬件资源，要么是永久专属于 TPM 使用的，要么是暂时只分配给

TPM 所使用的。通常，计算机是不能直接访问 TPM 内部存储器的。

可信度量根（Root of Trust for Measurement, RTM）^[18]: RTM 是由 TPM 内部的处理器和可信度量根核（CoreRoot of Trust for Measurement, CRTM）组成的。CRTM 是一个软件模块，执行于 TPM 内部处理器上的代码。TPM 完成对需要对其他模块的度量，确认被度量的模块是否可信，并将度量值存储于其内部的存储器中。而所谓度量，通常就是计算被度量模块的可能值并以此作为度量值，再把度量值与存储于 TPM 内部的期望值相比较，如果与期望值相符，则证明被度量的模块是可信的。

可信报告根（Root of Trust for Report, RTR）^[18]: 所谓报告是当使用者需要知道 TPM 的状态或者其内部记录的某些度量值时，TPM 需要为使用者提供相应的摘要。通常，摘要指的只是一些度量值，使用者需要具备相关的知识和能力来自己来判断报告所反映的信息。

系统启动时，必须要保证正确的启动顺序，否则不能建立起完整的信任链。系统启动时，必须先运行 RTM 中的 CRTM，由 CRTM 来度量 BIOS。所谓度量，其实就是根据规范来计算 BIOS 代码的可信值，如果计算出来的值符合预期，则认为 BIOS 是可信的，若不符合预期，则不能启动。若 BIOS 可信，则信任边界扩大到 BIOS，并赋予 BIOS 以信任权。之后由 BIOS 度量系统的加载程序，如果系统的加载程序也是可信的，

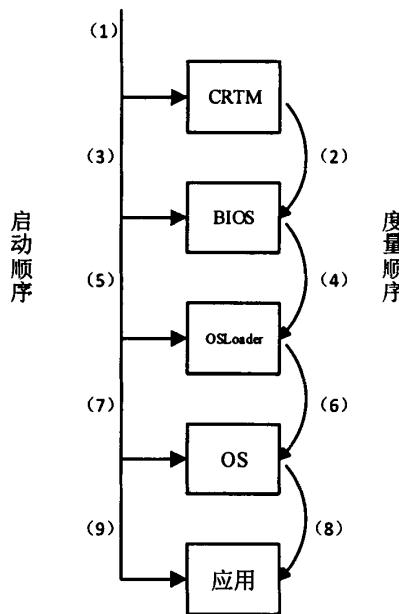


图 2.2 信任链的建立顺序

Fig. 2.2 The Order in Establishing The Trusted Chain

则信任边界继续扩大。如此类推，一级度量一级，最终将信任扩散到系统上的应用。系统模块的启动顺序不能有变化，必须严格地按照规定启动。图 2.2 描述了系统中一条信任链的建立顺序^[19]。

2.3 TPM 安全芯片

2.3.1 TPM 的结构

TCG 的协议中规定只能通过 LPC 接口与外界通信，外界模块不能越过 TPM 芯片的 I/O 而对其内部的状态进行操作。在 PC 中，TPM 通过 LPC 总线同南桥芯片连接。TPM 在可信 PC 中的连接如图 2.3 所示。

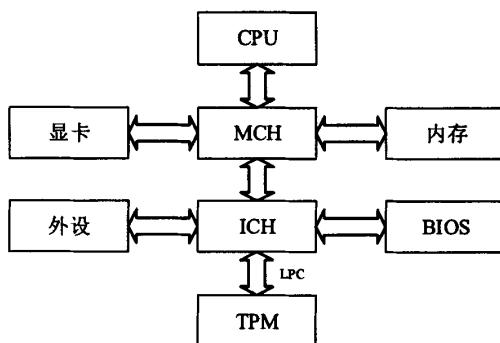


图 2.3 TPM 在可信 PC 中的连接
Fig. 2.3 The Connection Between TPM and PC

TCG 的协议中还对 TPM 内部的构成做出了规范。TCG 规定的 TPM 需要有如下配置：I/O 接口、非对称加密引擎、对称加密引擎、Hash 引擎、管理单元、授权单元、秘钥发生器、随机数发生器、电源监测、执行引擎、非挥发性存储器和挥发性存储器。TPM 的结构如图 2.4 所示。

TPM 芯片需要有一个用来与主系统交互的接口。TCG 规定 TPM 芯片通过 LPC 接口同主系统进行数据交换。主系统能够通过 I/O 口将要执行的命令送入 TPM 中，也要能从 TPM 的 I/O 口上读取反馈回来的数据。LPC 接口是 TPM 与外界通信的唯一接口，任何模块都不能被允许越过 I/O 口直接对 TPM 内部的状态进行更改。同时，TPM 应该保证 I/O 的数据正确。I/O 应该位于 TPM 的保护区域中，以防止推送到 I/O 上的数据在被主系统读取之前被篡改。

管理单元主要负责 TPM 芯片的管理与配置功能；授权单元主要负责证书的管理和

检验，例如某些程序可能会需要使用保护区域内的数据，则执行命令前 TPM 会检验其证书是否合法，只有通过授权单元检测的程序才能顺利执行；秘钥铲产生器用来制造所需要的秘钥；随机数发生器（RNG）是用来产生真随机数的部件，在软件中所使用的随机数通常是伪随机数，是通过算法模拟出来的假随机数，而在 TPM 安全芯片中要

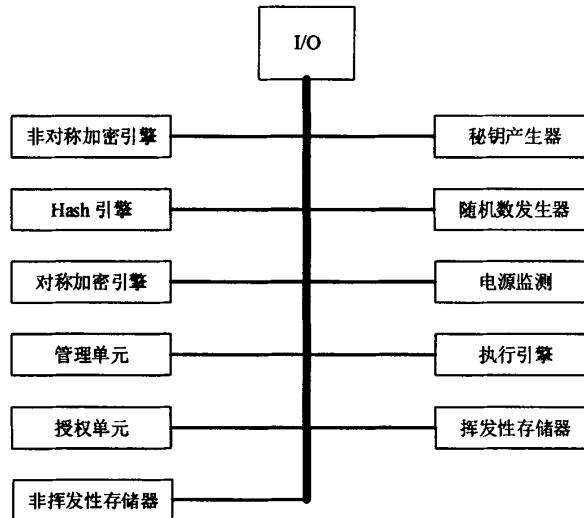


图 2.4 TPM 的结构

Fig. 2.4 Architecture of TMP

求配有真随机数发生器，由真随机数发生器产生的随机数不可预测；执行引擎负责执行 TPM 的指令，通常由处理器和一些软件组成；电源监测单元用来管理 TPM 的电源；挥发性存储器主要是指一些寄存器，用来存储临时使用的数据、变量等；非挥发性存储器用来存储秘钥、证书等重要的数据。

2.3.2 Hash 引擎

Hash 引擎是 TPM 中用来计算 Hash 值的硬件单元，在计算完整性和验证身份等功能时需要 Hash 引擎的支持。Hash 函数有如下特点：

- (1) Hash 函数的输入长度是长度几乎不受限制；
- (2) Hash 能够将庞大的输入数据压缩为数据长度相对较小的数据，适合小容量的 TPM；
- (3) 输入数据微小的不同也会引起 Hash 值很大的变化；
- (4) 不可逆。

使用 Hash 函数度量某程序的完整性时，把要度量的代码的度量值计算出来后，用

Hash 函数生成其摘要，再与已经存储于 PCR 中的期望值相比较，如果相同则是符合期望的，说明程序可信，如果不同，则程序不可信，不能被执行。

为了确定信任链的启动顺序，TCG 还规定了一种迭代计算 Hash 函数的方式，通过对 Hash 函数的迭代计算能够确认信任链是否按照规定的顺序启动。当系统的某一级正确启动后，其 Hash 值会存储于 PCR 中，当得到新的 Hash 值后，再将新的 Hash 值与原有的 PCR 值连接起来再进行 Hash 函数的计算。如果启动的顺序不同，那么计算出来的 Hash 值也会不同，由此就可以确定系统是否按照规定的顺序启动了。

在使用进行数字签名时，也会调用 Hash 函数来对所签署的内容生成摘要。此外，在 Hash 引擎中还应该提供哈希消息身份认证码（Hash Message Authentication Code，HMAC）算法。HMAC 是对称加密算法的一种，是基于 Hash 算法的一种迭代算法，可以用来进行消息身份验证等功能。HMAC 算法是用一个秘钥和一段消息作为输入，用一个消息摘要作为输出。式 2-1 为 HMAC 的算法公式。

$$\text{HMAC}[K, t] = H[(\bar{K} \oplus \text{OPAD}) \parallel H((\bar{K} \oplus \text{IPAD}) \parallel t)] \quad (2.1)$$

式中，K 是输入的秘钥，t 为输入的密文，OPAD 和 IPAD 分别为两个根据不同哈希算法而变动的值。在基于 SHA-1 算法的 HMAC 算法中，OPAD 为 0x5c 重复 64 次的值，IPAD 为 0x36 重复 64 次的值。

2.3.3 非对称加密引擎

TPM 使用非对称加密算法来实现签名、身份验证、保密通信等功能。TCG 规定 TPM 中至少提供 RSA 或者 ECC 其中一种非对称加密算法。

非对称加密算法的秘钥是成组出现的，由一个私钥可以对应多个公钥。非对称加密算法可以有如下两种用法：

密送，公钥加密私钥解密；

签名，私钥加密公钥解密；

用非对称加密算法实现密送功能时，首先由持有公钥的发送方对要发送的信息内容进行加密。把内容由公钥加密后，只有私钥才能进行解密，当接收方收到了信息的密后，使用对应的私钥进行解密即可获得信息的明文。在发送的过程中，即使信息的密文被第三方劫获，只要第三方没有对应的私钥就无法对密文进行解密，从而可以保证信息的安全。

而使用非对称加密算法进行签名时，则是完全不同的情况。签名信息的发布者可能会发布一些公告，希望拥有自己公钥的人都能够看到公告。在公告传输过程中，公告可能被人在中途篡改。使用非对称加密算法对公告进行签名则可以有效防范上述状

况的发生。

在发布公告时，首先由发布者将已经写好的公告内容进行运算（通常使用 Hash 函数），生成公告的摘要。生成公告的摘要后，发送方再使用私钥对摘要进行加密，生成数字签名。最后将数字签名和要发布的公告一同发布给接受者。当接收者收到了发布者发布的公告后，先使用对应的公钥来对数字签名进行解密，如果能够成功解密，则说明公告确实由发布者发布。此外，接受者还应该使用同发布者相同的 Hash 函数来计算公告的摘要，如果得到的摘要同解密出来的摘要相同，则说明公告在传输过程中没有被人篡改。

一旦 TPM 对某些内容签名，则说明该内容是可信的，所以 TPM 对签名的操作是由严格要求的。通常，TPM 所签署的内容都是产生于 TPM 内部的。TPM 内部产生需要简明的内容时，会以一个参数 (TPM_GENERATED_VALUE) 为起始。签名时，TPM 内部会有相关的模块来确认即将签名的内容是否是来自 TPM 内部。

有些时候，TPM 也需要对来自 TPM 之外的内容进行签名。需要被签名的内容如果是安全的，则在 TPM 内部会对这个内容生成一个特殊的证书，这个证书标志着该内容可以被签名。

2.3.4 对称加密引擎

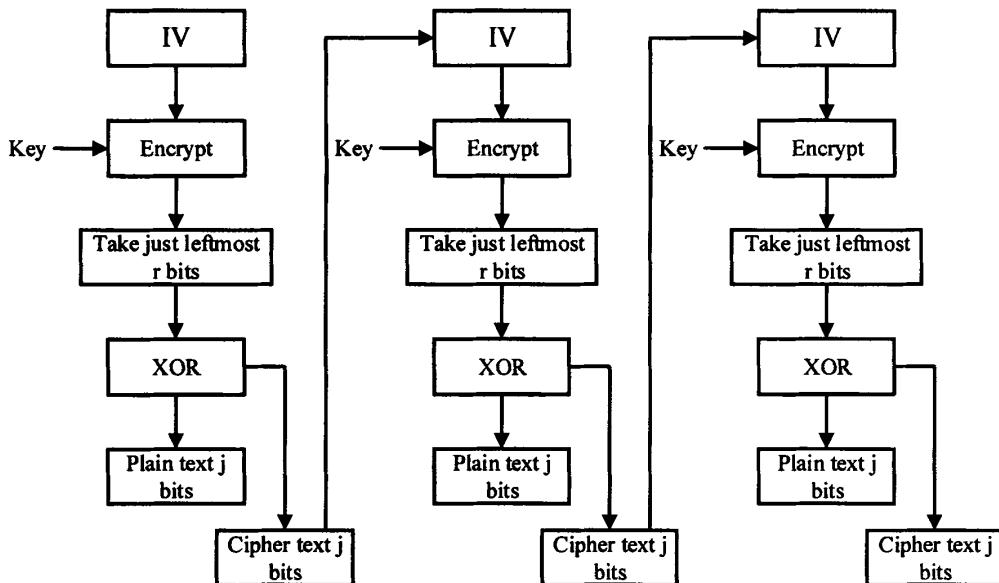


图 2.5 CFB 模式的加密过程

Fig. 2.5 Encryption Process in CFB Model

TPM 允许使用对称加密算法来对一些参数进行加密，或者对一些需要存储在 TPM 外部的内容加密。对称加密算法分为块加密和流加密。TPM 中，块加密的对称加密算法需要使用密文反馈模式（CFB）。

其中，CFB 模式的特点是，即使前后输入的明文相同，也能得到不同密文。其加密过程如下。

首先将要加密的明文分为大小相同、满足加密要求的 j 位长的若干明文块，记为 P_i ($i=1,2,3,4,\dots$)，然后随机生成（或人为拟定）一个初始向量 IV，IV 的大小应该与明文块的大小相同。然后用秘钥对 IV 加密，得到 IV 的密文 $E(IV)$ ，然后取 $E(IV)$ 最左侧的 r 位同明文块 P_1 循环异或得到密文块 C_1 。然后由刚刚计算得到了密文块 C_1 充当第二次加密的 IV，对明文块 P_2 进行加密，如此循环，直到所有的明文都被加密。加密过程如图 2.5 所示。

CFB 模式的解密过程与加密过程相似，先从 IV 开始，对 IV 进行加密后，再取 $E(IV)$ 的前 r 位与密文进行异或操作即可得到明文。不难看出，CFB 模式下，加密操作必须是串行的，但解密操作可以是并行的。

3 MIPS 处理器

MIPS 处理器以设计简洁和执行效率高为特点，其指令集体系（Instruction Set Architecture, IAS）属于精简指令集计算（Reduce Instruction Set Computing, RISC）体系。RISC 体系是从二十世纪八十年代发展起来的，目前的 ARM、MIPS、Power 等处理器都属于 RISC 阵营中的成员。

目前，嵌入式处理器市场中几乎是 ARM 一家独大，尤其是在移动终端领域，ARM 可谓一统天下。但 MIPS 以其卓越的性能在残酷的竞争中存活了下来，在不少领域中占有一定的地位。MIPS 处理器现在主要是应用于嵌入式系统中和 SOC 中，包括路由器、打印机、服务器中都有 MIPS 处理器的身影，而在新兴起的可穿戴设备中，MIPS 处理器也能大显身手^[20]。虽然在市场上占有的份额不大，但是 MIPS 处理器的出色性能也使其在一些难以被垄断的领域中占有一定的地位，开发的潜力非常大。

MIPS 处理器在中国已经具有了一定的发展基础。中科院计算所在 2001 年成立了一个研究处理器的项目组，这个项目组就是后来的龙芯中科公司。龙芯开发的龙芯系列处理器就是采用的 MIPS 架构，目前已经开发到了第三代产品。

3.1 MIPS32 指令结构

本文所设计的 MIPS 微处理器采用 MIPS32 指令集，所有的指令的长度都是 32 位。指令的种类上大体可以分为三类，即 R 型指令、I 型指令、J 型指令^[21]。

I 型指令，也叫立即数型指令，顾名思义，就是指令中包含一个立即数。其格式如图 3.1 所示。指令的 31~26 位是操作码，25~21 位是 GPR[Rs] 码，20~16 位是 GPR[Rd] 码，15~0 位是立即数。

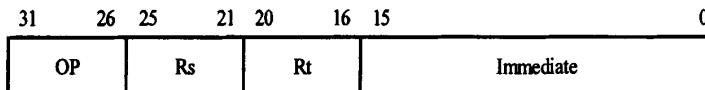


图 3.1 I 型指令结构

Fig. 3.1 Architecture of I Type Instruction

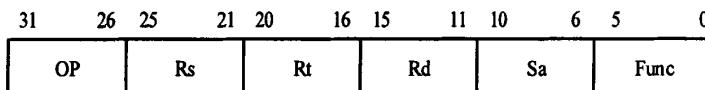


图 3.2 R 型指令结构

Fig. 3.2 Architecture of R Type Instruction

R 型指令，也叫寄存器型指令。R 型指令中包含三个寄存器的信息。其格式如图 3.2 所示。指令的 31~26 位是操作码，25~21 位是 GPR[Rs]码，20~16 位是 GPR[Rt]码，15~11 位是 GPR[Rd]码，10~6 位是 sa 码，5~0 位是功能码。其中的 sa 码只在位移指令中使用。

J 型指令，也叫跳转型指令。代码只有两部分，第一部分是 31~26 位的操作码，第二部分是后面的 26 位跳转地址。其结构如图 3.3 所示。

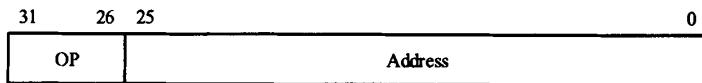


图 3.3 J 型指令结构

Fig. 3.3 Architecture of J Type Instruction

3.2 MIPS 中的寄存器

MIPS 处理器中有 32 个通用寄存器，其编号为 \$0~\$31。其中寄存器 \$0 比较特殊。寄存器 \$0 是 MIPS 为一个经常使用的数据 0 提供的一个专门的寄存器，不论向寄存器

表 3.1 寄存器名称和用途

Tab. 3.1 register's name and purpose

寄存器编号	寄存器名称	寄存器用途
0	zero	返回常数 0
1	at	用于汇编暂存
2、3	v0、v1	子程序调用返回值
4~7	a0~a3	子程序调用参数
8~15	t0~t7	临时使用的寄存器（就是可以随意使用）
16~23	s0~s7	子程序临时变量。改变这些寄存器的值时需要先保存旧值，使用完后需要恢复原来的值
24、25	t8、t9	临时使用的寄存器（就是可以随意使用）
26、27	k0、k1	至少为异常（包括中断）处理程序保留一个
28	gp	全局指针
29	sp	堆栈指针
30	fp	帧指针
31	ra	保存调用子程序时的返回地址

$\$0$ 中存入什么数据，从寄存器 $\$0$ 中取出的数据永远都是 0。除了寄存器 $\$0$ 比较特殊外，其他寄存器都是普通的寄存器，但是在使用的方法上有一些约定的习惯。表 3.1 中列出了全部 32 个通用寄存器的约定用法。

此外，MIPS 中还有三个特殊寄存器：Hi、Lo、Pc^[22]。Hi、Lo 两个寄存器供给乘除法指令使用，而寄存器 Pc 作为地址指针来使用，其存储的内容是下一条要执行指令的存储地址。

3.3 MIPS 微处理器设计

本文所涉及的微处理器核的整体结构如图 3.4 所示。

微处理器的执行单元用来完成指令的执行动作，包括运算、加载、跳转等，其核心部件是 ALU，几乎所有指令的执行都要经过 ALU 的计算才能完成。

乘除法单元（MDU）是独立于 ALU 之外的运算器件。因为乘除法的操作过于复杂，难以将其整合到流水线中，所以用专门的流水线来进行乘除法运算。

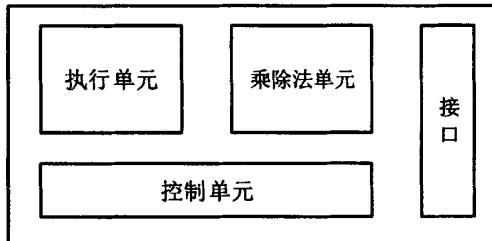


图 3.4 MIPS 微处理器核结构

Fig. 3.4 Architecture of MIPS Processor Micro Core

控制单元用来配置处理器核的工作状态，例如将数据格式从大尾端改为小尾端等。

为了提高处理器的运行速度，MIPS 采用了指令并行技术。最简单的指令并行就是使用流水线，使多条指令能够同时在处理器上执行，每条正在执行的指令都处于不同的流水级上。从指令执行的结果上看是每个周期完成一条指令，但实际几乎每个周期都有不止一条指令在执行。流水线将指令拆分成几个步骤，所有的指令执行时都要按照流水线的步骤执行。执行速度取决于执行速度最慢的一个步骤，即需要时间最长的一级流水级，所以要将处理器的执行速度提高，就要尽可能地合理地、平均地分配每一级流水线的工作。常见流水线的设计有四级流水、五级流水、七级流水，本文采用五级流水设计。参照已经被市场验证过的设计，本文的流水线设计如图 3.5 所示。

第一级为 I 级，即取指级。在这级中，处理器将要执行的指令从指令存储器中取出，

以供下级使用。

第二级为 E 级，即执行级。在这级中，处理器会根据指令的内容进行相应的执行动作，例如从寄存器堆中取出数据、计算 M 级所需要的地址、按照指令的内容进行算数运算或者位移操作。

第三级为 M 级，即访存级。这级主要是根据指令的要求对相应的存储地址进行存取操作。

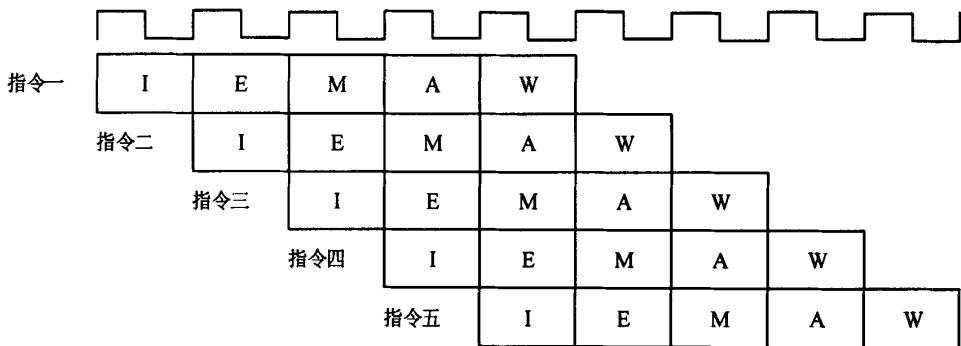


图 3.5 微处理器流水线

Fig. 3.5 Pipeline of Micro Processor

第四级是 A 级，即对齐级。对从存储器中加载的数据进行对齐操作。

第五级为 W 级，即回写级。是根据指令的要求将数据送回到相应的寄存器中。数据的来源可能是经过 ALU 计算的算数运算结果，也可能是在存储器中加载来的数据。

3.4 MIPS 微处理器核结构

本节通过介绍几条基本指令的实现方法来简要介绍 MIPS 微处理器核的构造以及流水线的具体工作方式。

3.4.1 算数运算指令数据通路的搭建

以 ADDU 指令为例。ADDU 指令是我们使用微处理器时经常要用到的指令，其汇编格式如下：

ADDU rd, rs, rt

ADDU 指令的功能是将寄存器 rs、rt 中的数据进行无符号加操作，并且将得到的结果存入寄存器 rd 中。ADDU 不会改变寄存器 rs、rt 中的值，但会改变 rd 中的值。其指令格式如图 3.6 所示。

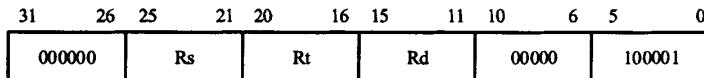


图 3.6 ADDU 指令格式

Fig. 3.6 Instruction Formate of ADDU

在 I 级, ADDU 指令有如下操作:

- (1) 根据寄存器 PC 中的值从指令存储器中取出指令;
- (2) 将 PC 的值加 4 来使其指向下一条指令。

图 3.7 为 I 级所需的硬件结构。

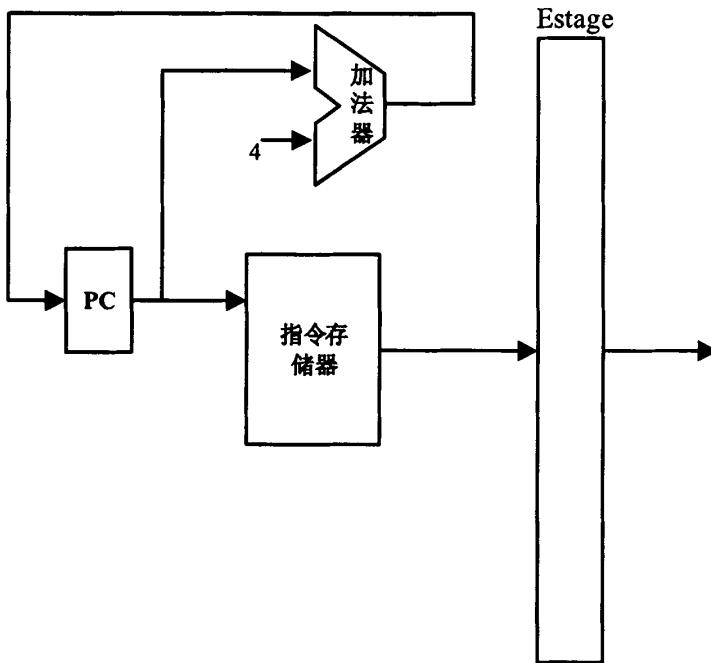


图 3.7 ADDU 指令所需的 I 级流水线结构
Fig. 3.7 Architecture of I Stage Needed by ADDU

指令 ADDU 在 I 级后将进入 E 级。图 3.8 所示为 E 级流水线的部分结构。E 级流水线将进行如下操作:

- (1) 将指令进行译码;
- (2) 根据指令内容从相应的寄存器中取出操作数;
- (3) 将操作数送入 ALU 进行无符号加法运算;

(4) 根据译码的结果产生必要的控制信号送入下一级流水。

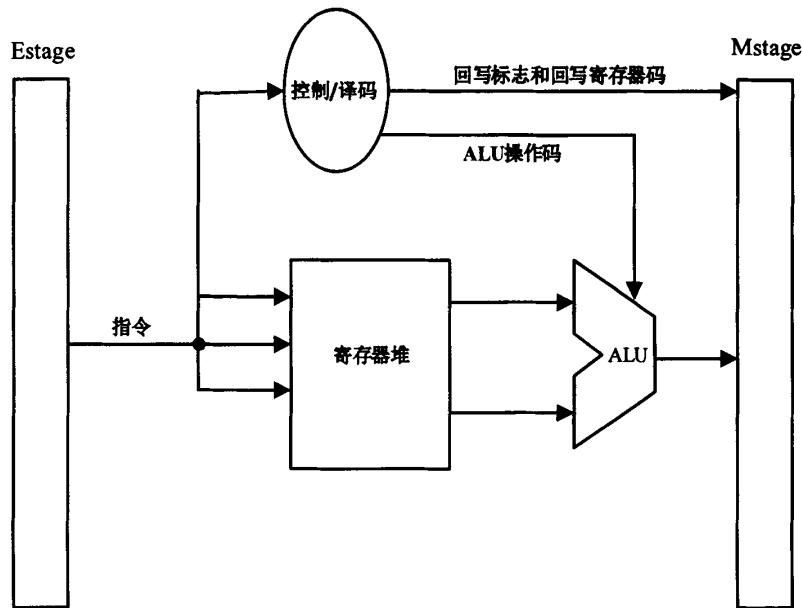


图 3.8 ADDU 指令所需的 E 级流水线结构
Fig. 3.7 Architecture of E Stage Needed By ADDU

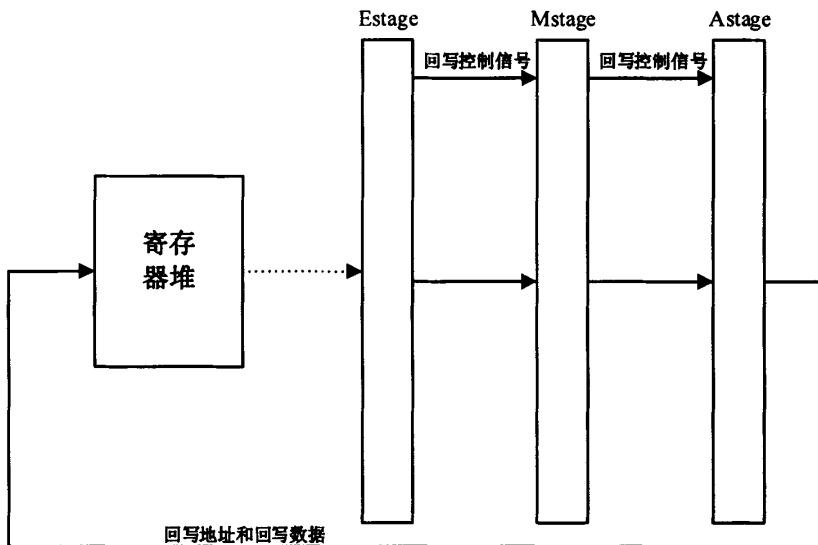


图 3.9 ADDU 指令所需的 E、M、W 级流水线结构
Fig. 3.9 Architecture of E, M, W Stage Needed by ADDU

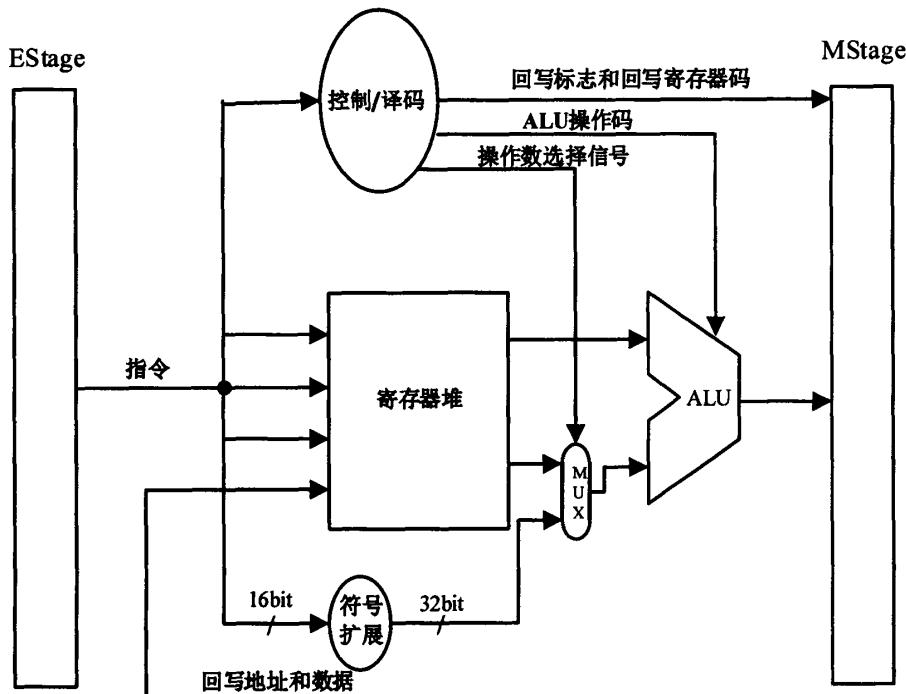


图 3.10 添加了符号扩展单元和多路选择器的 E 级流水线

Fig. 3.10 Architecture of E stage with SignExtend

由于 ADDU 指令不需要对外部的存储器进行存取操作，所以 ADDU 指令在 E 级所产生的数据和控制信号在之后的 M、A 级并未进行任何操作，而是在 M 级、A 级流水线内等待了两个周期后被送入 W 级。在 W 级流水线中，会根据由上一级送来的控制信号判断是否需要对寄存器进行回写操作，如果需要回写，则将送进 W 级流水线的数据根据一并送来的地址写回寄存器，显然 ADDU 指令是需要进行写回操作的。图 3.9 所示为 ADDU 指令执行时所需的 M 级和 W 级的硬件结构。

这样，经过五个周期后，ADDU 指令就完成了其相应的操作。完成 ADDU 指令后，流水线的数据通路大体上已经搭建起来，只要继续向流水线中添加相应的结构就能完成更多的指令。

ADDU 是 R 型的无符号数加法指令，在 MIPS32 指令集中还有一条 I 型的无符号数加法指令，即指令 ADDIU。要实现 ADDIU 只需要在 E 级流水线中增加符号扩展单元和多路选择器即可。符号扩展单元用来根据需要将指令中 16 位的立即数扩展为 32 位的操作数，而 ALU 输入口前添加的多路选择器用来根据指令的要求选择送去 ALU 中的操作数的来源。如图 3.10 所示为添加了符号扩展单元和多路选择器的 E 级流水线。

3.4.2 存取指令数据通路的搭建

MIPS32 指令集中用有一系列用于对外部存储器进行存取操作的指令，例如 LW 和 SW。

LW 指令全称为 load word，其汇编格式如下：

LW rt, offset (base)

LW 指令进行的操作是将存储器中目标地址的数据存入 rt 寄存器中。

SW 指令全称为 store word，其汇编格式如下：

SW rt, offset (base)

SW 指令进行的操作是将 rt 寄存器中的数据存入存储器的目标地址中。

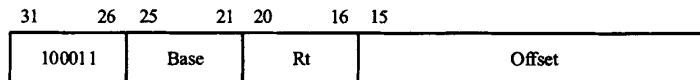


图 3.11 LW 指令格式

Fig. 3.11 Instruction Formate of LW

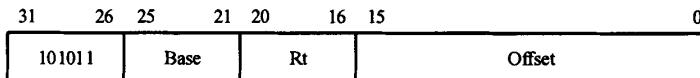


图 3.12 SW 指令格式

Fig. 3.9 Instruction Formate of SW

LW 指令和 SW 指令形式上属于 I 型指令，其指令格式分别如图 3.11 和 3.12 所示。

LW 和 SW 对外部存储器的存取操作的地址是由 ALU 对基址和偏移地址进行加和得来的。指令中的 base 部分是存储基地址的寄存器地址，offset 是一个代表偏移地址的立即数。LW、SW 指令都需要控制单元产生对外部存储器的读写控制信号，随流水线一并流入 M 级。此外 SW 需要对寄存器进行写回，也需要控制单元产生写回的控制信号。图 3.13 为添加了相应模块的 E 级和 M 级流水线。

MIPS32 中除了 LW、SW 外，还有一些特殊的存取指令，例如 LL、LH 和 SL、SH 等，这些存取指令不是对整个字进行存取，而是对半字或者字节进行存取操作。MIPS 指令中的存取操作必须严格对齐，所以在流水线中专门设有 A 级流水线来进行对齐操作。图 3.14 所示为 A 级流水线。

从存储器中取出的数据经过 A 级流水线对齐操作后，需要写回到相应的寄存器堆

中。但写回到寄存器堆中的数据有两种不同的来源，一个是 ALU 的计算结果直接写回到寄存器中，另一个是从存储器中加载来的数据写入寄存器中。所以，在 W 级需要进行写回数据的选择。图 3.15 为添加了回写选择机制的 W 级。

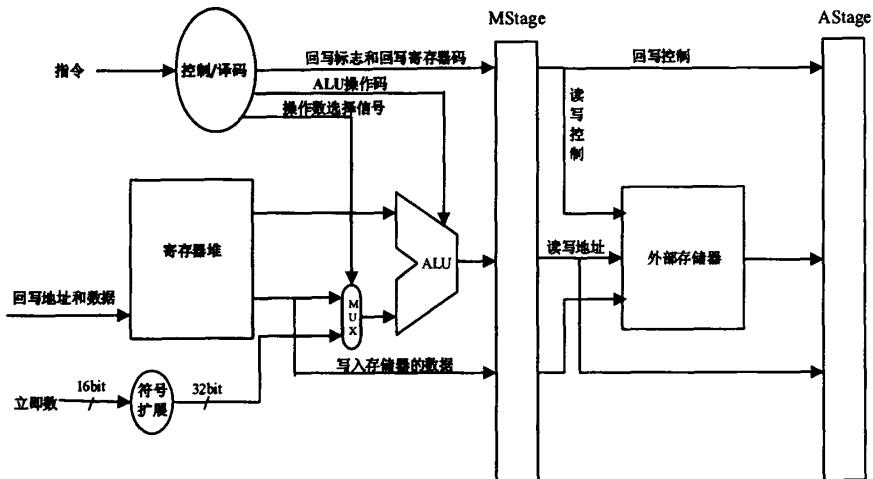


图 3.13 添加 LW、SW 指令的 E 级、M 级流水线
Fig. 3.13 Architecture of E and M stage with LW and SW instruction

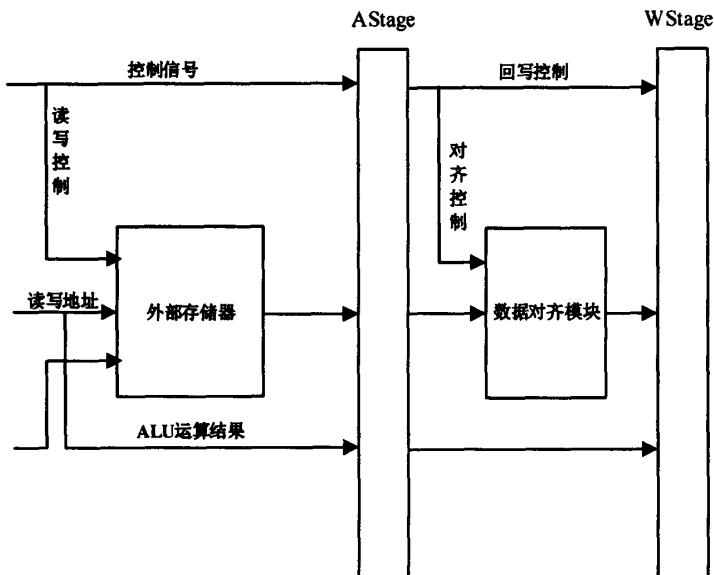


图 3.14 A 级流水线
Fig. 3.14 Pipeline of A Stage

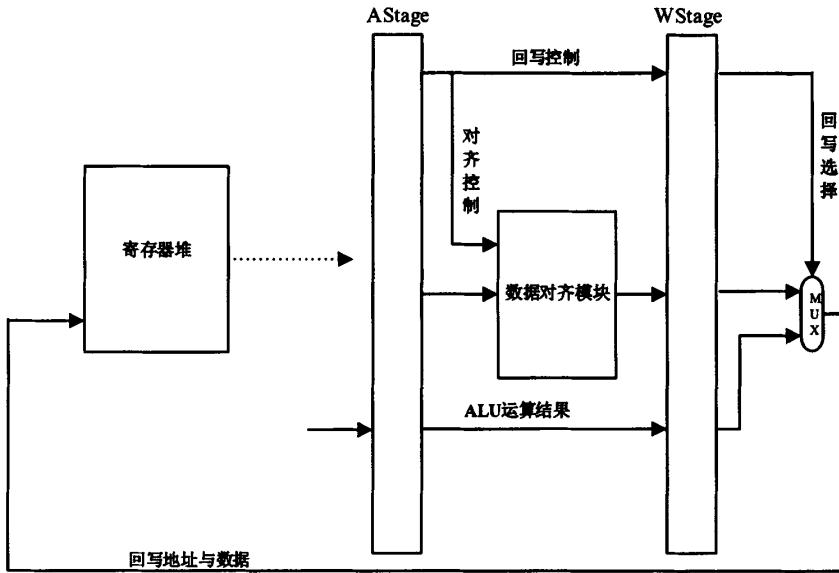


图 3.15 添加了回写选择机制的 W 级

Fig. 3.15 Architecture of W stage with the function of write back

3.4.3 分支指令、跳转指令的搭建

这一节以相等分支指令 BEQ 和跳转指令 J 来介绍 MIPS 微处理器核中的分支指令与跳转指令的搭建。BEQ 指令的汇编格式如下：

BEQ rs, rt, offset

BEQ 指令实现的功能是，先比较寄存器 rs、rt 中的值，如果两个寄存器中的值相等，则会改变 pc 的值使其指向 PC+ (offset||00) 地址的指令。如果寄存器 rs 和 rt 中的值不等，则不会发生分支。BEQ 指令的 offset 值是 16 位的立即数，但执行分支指令时，处理器加到 PC 值上的偏移值是左移两位后的 offset 值。由于执行的是有符号加法，所以分支地址的范围是±128KB。

跳转指令 J 的汇编格式如下：

J target

J 指令实现的是在 256MB 范围内的无条件跳转，执行时会把 target 值左移两位后加与 PC 值进行有符号加法，并将结果送回 PC 中，使其指向目标地址。图 3.16 和图 3.17 所示分别为 BEQ 指令和 J 指令的结构。

为了实现 BEQ 指令和 J 指令，需要在流水线的 I 级添加 PC 寄存器输入的多路选择器，用来选择送入 PC 中的值是顺序的 PC+4 还是经过分支、跳转指令重新计算过的值。另外，还需要将 PC 的值送入到下一级中用来计算分支、跳转的目标地址。如图 3.18 所示。

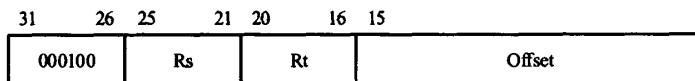


图 3.16 BEQ 指令结构
Fig. 3.16 Instruction Formate of BEQ

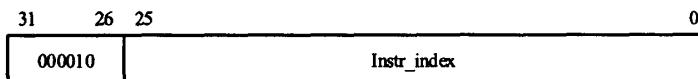


图 3.17 J 指令结构
Fig. 3.17 Instruction Formate of J

分支、跳转指令在进入到 E 级后，需要完成如下执行动作：

- (1) 完成对条件分支指令的条件判断；
- (2) 对偏移量进行合适地扩展；
- (3) 完成目标地址的计算；
- (4) 产生相应的控制信号。

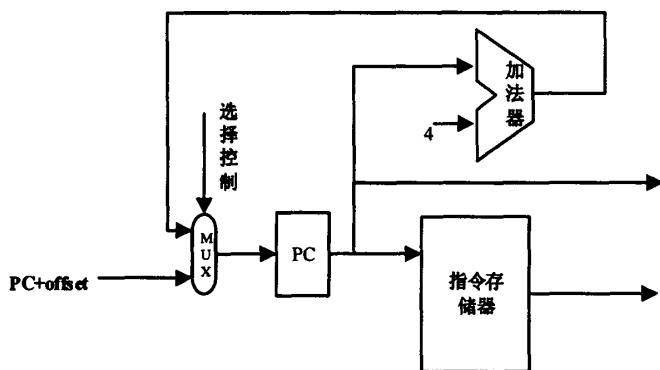


图 3.18 添加分支、跳转指令的 I 级流水线
Fig. 3.18 Architecture of I stage with the function branch and jump

根据指令需求，需要为 BEQ 添加条件判断单元，用来判断是否分支，如果分支成

功，则控制单元需要产生相应的控制信号把计算好的目标地址值送入 PC 中。因为分支、跳转指令对偏移量的操作都是要先左移两位的，所以还需要为偏移量计算添加位移模块。产生可用的偏移量后，将偏移量和当前 PC 值进行有符号数加法。添加相应结构的 E 级流水线如图 3.19 所示。

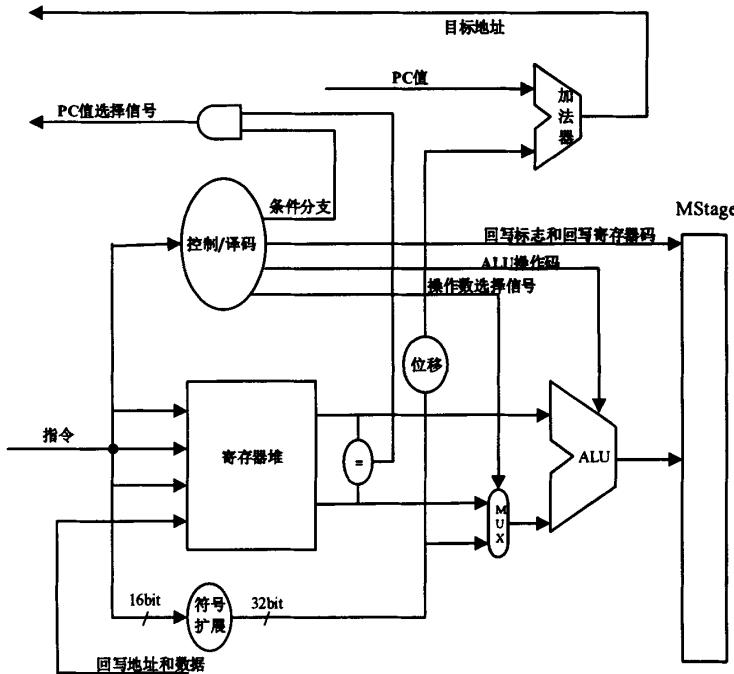


图 3.19 添加了分支、跳转机制的 E 级流水线

Fig. 3.19 Architecture of Estage with the function branch and jump

某些子程序调用指令在跳转后还需要将返回地址保存在寄存器中，MIPS 体系中规定 \$31 寄存器专门用来保存这些程序返回复制。另外，从流水线的结构上可以看出，在跳转的目标地址写入 PC 前， $PC+4$ 的地址就已经写入了 PC，也就是说跳转指令或者分支指令的下一条指令会被正常取出并执行。在跳转、分支指令和目标地址的指令之间额外的一个时钟周期称之为延迟槽，延迟槽内的指令就是跳转、分支指令后面的一条指令。MIPS 的做法是允许延迟槽内的指令正常进入流水线执行，而为此带来的影响将尽可能地交由编译器通过调整指令顺序来解决。在不影响执行结果的情况下，编译器通常会将分支、跳转一类的指令提前一个周期执行，如果实在不能提前，则会在分支、跳转类指令之后插入空指令。例如使用者本意是想要这样执行指令：

```

ADD $0, $1, $2
ADD $2, $3, $4
BEQ $7, $8, 0x1111

```

通过编译器调整顺序后，可能被调整为如下顺序：

```

ADD $0, $1, $2
BEQ $7, $8, 0x1111
ADD $2, $3, $4

```

3.4.4 流水线冒险处理与异常处理

流水线中有时会出现冒险情况。考虑如下执行情况：

```

ADD $1, $2, $3
ADD $4, $5, $1
SUB $6, $7, $1
SUB $8, $9, $1

```

当流水线执行到第二条指令时，显然使用者的本意是\$4 的值应该等于寄存器\$5、\$2 中的数相加再加 3 才对，但当第二条指令进入到 E 级流水线需要使用\$1 的结果时，本应该用来计算的\$1 结果还停留在 M 级，没有写回到\$1 寄存器中。此时使用\$1 中的值计算出来的并不是本来想要的结果。同理，执行第三条指令时，所需的\$1 的结果刚刚流到了流水线的 A 级上，也没有写回到寄存器。直到第四条指令执行时，正确\$1 的值才流到 W 级进行写回，所以第四条指令的结果是正确的（我们默认寄存器堆中如果同时进行读写操作，则写入的值会直接透过寄存器送到寄存器的输出端口）。这种现象称作流水线数据冒险。本文的流水线存在 M 级流水线冒险和 A 级流水线冲突两种冒险情况。

解决流水线数据冒险的方法就是建立旁路。实际上，当第二条、第三条进入 E 级时，第一条指令的结果已经被计算出来了，只是还没有写回到寄存器中而已。建立旁路可以将未写回寄存器的数据直接从流水线中转发到 E 级中以便正在 E 级中执行的指令使用。

一般，存取指令和算数运算指令在 M 级和 A 级都能转发给 E 级。如图 3.20 所示。

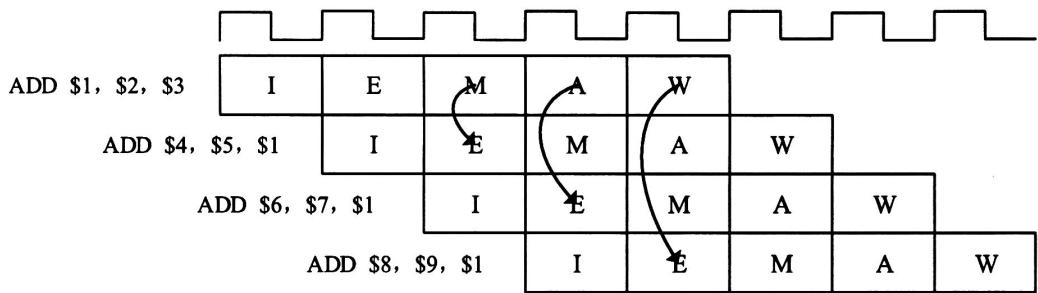


图 3.20 数据转发的一种情况

Fig. 3.20 A Case of Data Forwarding

然而，还有另外一种情况。例如下面这段程序：

LH \$1, \$2, \$3

ADD \$4, \$5, \$1

显然，由于 LH 是半字加载指令，需要的数据要在 A 级才能计算完整，当 ADD 指令进入 E 级时，LH 指令正处于 M 级，不能提供正确的转发数据。为了应对这种情况，让流水线暂停一下等待 LH 指令进入 A 级将正确的数据计算好后，才能转发给 ADD 指令使用。这种现象称为流水线气泡。在流水线中插入气泡的方法很简单，只要将 ADD 指令所计算的结果丢弃掉（保证不要回写到寄存器中）然后再执行一次 ADD 指令就可以了。如图 3.21 所示，图中画叉的位置表示执行结果被丢弃。

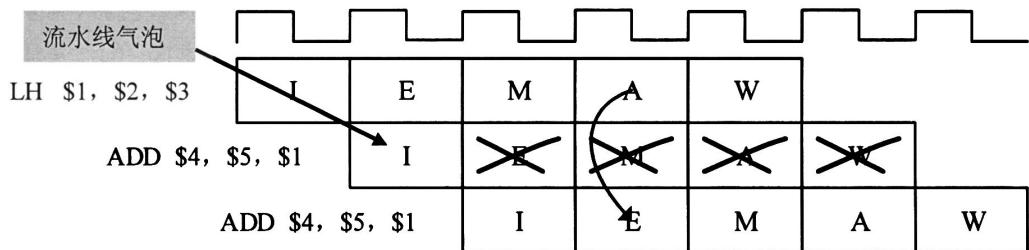


图 3.21 流水线气泡

Fig. 3.21 Pipeline Bubble

此外，本文的 MIPS 微处理器核中还提供了异常处理机制。当流水线中发生异常时，微处理器会将指令地址指向一个特定的异常处理地址，并在寄存器\$31 中记录返回地址的值。微处理器会根据发生异常的原因在协处理器 CP0 中记录相应的异常码，方便异常处理程序查看异常原因。

3.5 模块的实现

本文对设计的 MIPS 处理器核进行了功能验证，保证所设计的 MIPS 处理器功能正确，以确保后续的设计工作可以顺利进行。验证方法为：编写汇编程序，用汇编器翻译成机器码后在 VCS 上进行仿真，通过查看关键信号波形的方式来确认 MIPS 的工作是否正确。由于 MIPS 处理器设计时没有按照模块分别设计，而是通过在整体框架上逐渐添加结构的方式完成的，故验证时没有对每个模块进行单独的验证，而是直接进行整体的验证。

图 3.22 所示为以 ADDIU 指令为例的 I 型指令的执行情况。第一个周期，指令存储器的地址线 inst_addr 指向地址 0，指令存储器地址 0 中的指令被驱动到 inst_i 接口上，指令处于 I 级，等待上升沿到来被读取。待读取的指令是：

ADDIU \$1, \$0, 0x1010

经过编译后的机器码为：0x24011010。第一个上升沿后，inst_i 上的指令被读取的 E 级流水线的寄存器 inst 中，同时，inst_addr 由处理器内部的 PC 进行加 4 操作，指向下一条指令的地址。读取指令后，指令被译码，由控制电路产生写寄存器的标志 RegWrite 和要写入的寄存器标号 Rt，可见图中 RegWrite 被拉高，Rt 变为 1。RegWrite 和 Rt 会随着流水线流到 W 级的寄存器中作为寄存器堆的写控制信号。可以看出第四个上升沿后，寄存器堆的写入信号已经被驱动到寄存器堆的端口上，分别是地址信号 WriteReg、写信号 RegWrite、写入数据 Writedata。当第五个上升沿来到时，Writedata 将会被写入到对应的寄存器中。写入数据是 0x0000_1010，正是我们指令中要求的 0+0x1010 的结果。

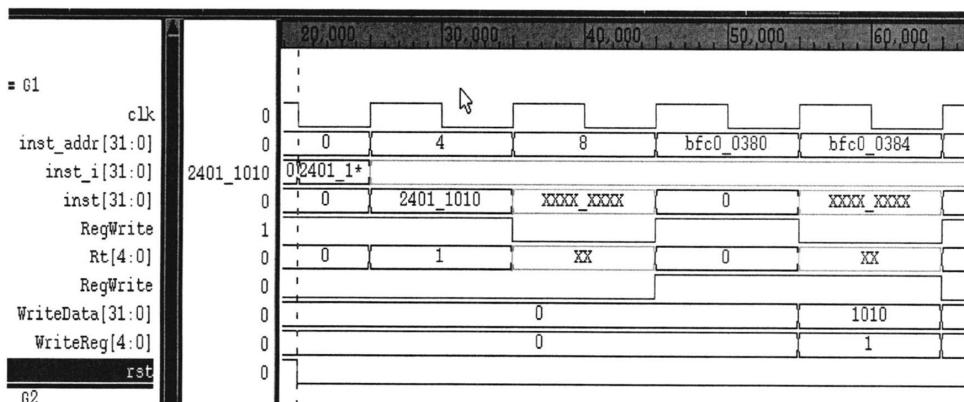


图 3.22 ADDIU 指令的执行

Fig.3.22 Execution of ADDIU

图 3.23 和图 3.24 所示为 MIPS 处理器执行以下指令的情况：

```
LUI      $5, 0x1234
XORI    $6, $0, 0x5678
ADDU   $7, $5, $6
```

其中 LUI (Load up immediate) 指令的动作是将一个 16 位的立即数存入指定的寄存器 Rt 的高 16 位中。XORI (Exclusive OR Immediate) 指令是将一个 16 位的立即数与 Rt 寄存器中的数进行异或操作。LUI 和 XORI 指令都属于 I 型指令。

ADDU 指令的动作是将寄存器 Rt、Rd 中的数据进行无符号加操作，结果存入寄存器 Rs 中。

以上三条指令所完成的功能是：

- (1) 将立即数 0x1234 存入寄存器\$5 的高位中；
- (2) 将立即数 0x5678 与寄存器\$0 中的数进行异或操作，结果存入寄存器\$6 中；
- (3) 将寄存器\$5、\$6 中的数进行无符号加操作，结果存入\$7 中。

最后得到的结果应该是：

- (1) 0x12340000 被存入寄存器\$5；
- (2) 0x00005678 被存入寄存器\$6 中；
- (3) 0x12345678 被存入寄存器\$7 中。

经过编译，三条指令的机器码分别为：0x3c051234、0x38065678 和 0x00a63821。

由图 3.23 所示，第一个周期中，指令 LUI 处于 I 级，被驱动到 inst_i 接口上等待读取。第二个周期，inst_i 上的 LUI 指令被读取到 E 级的 inst 寄存器中，进入到 E 级，并且 PC 的值自动加 4，同时 XORI 指令进入 I 级，被驱动到 inst_i 接口上等待读取。第三个周期时，指令 LUI 进入 M 级，指令 XORI 进入 E 级，PC 自动加 4 变为 8，指令 ADDU 进入 I 级被驱动到 inst_i 接口等待被读取。ALU 的 result 产生 XORI 操作的结果。第四个时钟周期时，指令 LUI 进入 A 级，XORI 进入 M 级，指令 ADDU 进入 E 级。由于 ADDU 所用到的两个操作数分别来自前两条指令的计算结果，所以在对 ADDU 指令进行译码后，触发了流水线冲突的处理机制。ADDU 指令所需的寄存器\$5 和\$6 的数据还分别停留在流水线的 A 级和 M 级中，需要通过旁路转发到 E 级以供 ALU 计算。可以看出图中的 RsSel 信号和 R1Sel 信号发生了变化，这两个信号是 ALU 的 A、B 两个接口前的多路选择器控制信号，分别变成了从对应流水线中转发数据的状态使进入到 ALU 中的数据分别是我们需要的 0x12340000 和 0x00005678。图中可见 ALU 的输出信号 result

产生结果 0x12345678。第五、第六、第七个时钟周期时 LUI、XORI 和 ADDU

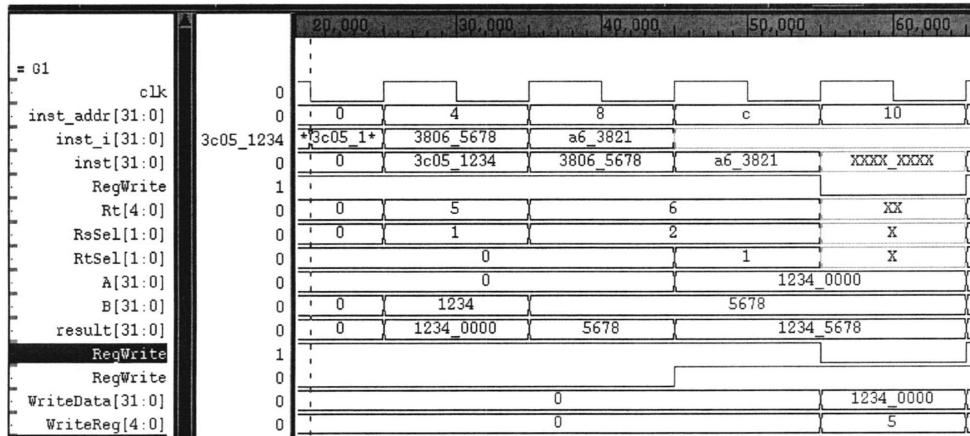


图 3.23 LUI、XORI 和 ADDU 指令的执行 (1)

Fig.3.23 Execution of LUI, XORI and ADDU(1)

指令先后进入 W 级，将要写回寄存器的数据和控制信息驱动到寄存器堆的端口上，分别是写地址信号 WriteReg、写使能信号 RegWrite 和写数据信号 WriteData。可见三条指令的功能都可以被正确地执行、产生流水线冲突时可以实现正确的数据转发功能。

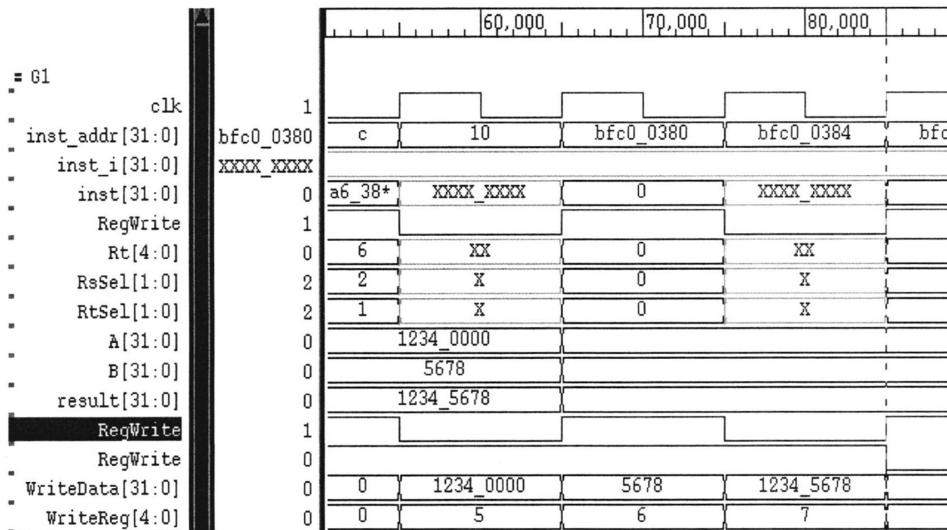


图 3.24 LUI、XORI 和 ADDU 指令的执行 (2)

Fig.3.24 Execution of LUI, XORI and ADDU(2)

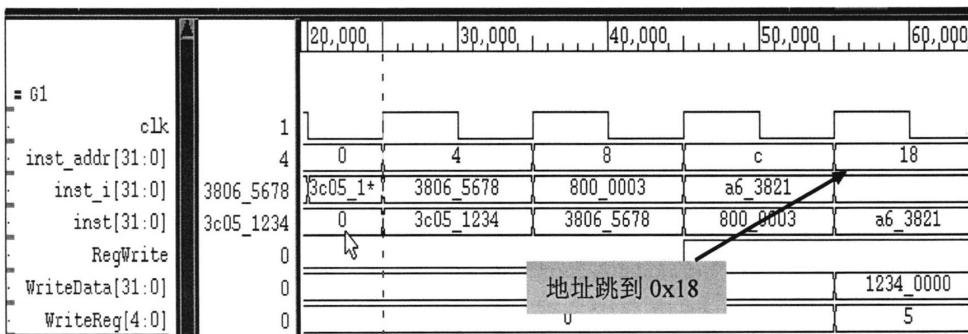


图 3.25 J 型指令的执行 (1)
Fig.3.25 Execution of J Type Instruction(1)

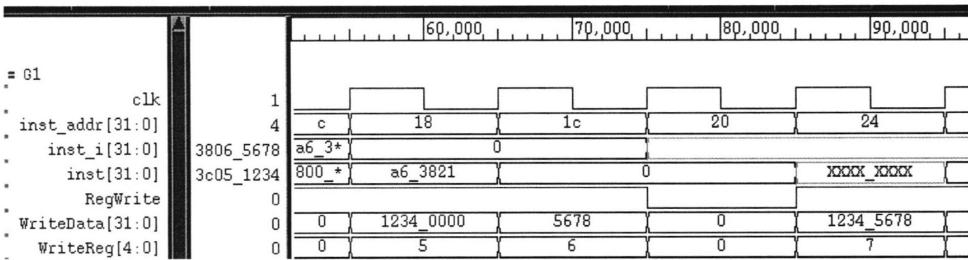


图 3.26 J 型指令的执行 (2)
Fig.3.26 Execution of J Type Instruction(2)

图 3.25 和图 3.26 是执行的以下命令的情况：

```

LUI      $5, 0x1234
XORI    $6, $0, 0x5678
J       0x10000003
ADDU    $7, $5, $6

```

新添加的 J 指令是跳转指令，属于 J 型指令，该指令经过编译后得到机器码位 0x08000003。

如图所示，第一个周期时，指令 LUI 处于 I 级，被驱动到 inst_i 接口上等待被读取。第二个周期，指令 LUI 被读取到 E 级流水线中执行，PC 加 4 后 XORI 指令进入 I 级，被驱动到 inst_i 接口上。第三个周期时，LUI 指令到达 M 级，XORI 指令到达 E 级被执行，J 指令进入 I 级被驱动到 inst_i 接口中，PC 加 4 后变为 0xC；第五个周期，J 指令进入 E 级被执行，其执行效果是使下一个周期的 PC 值变为当前 PC 值加偏移量，使指令

跳转。计算后得到的 PC 值（二进制）为 $1100+1100=11000$ ，即 `inst_addr` 变为 `0x18`。可以看出第五个周期时，`inst_addr` 值变为 `0x18`，指令的跳转功能正确实现。而处于分支间隙中的 ADDU 指令此时已经处于 I 级，将会继续被读入流水线中执行。图中可见在第五、第六和第八周期时，指令 LUI、XORI、ADDU 分别进入了 W 级对寄存器堆进行写操作。可以看出 J 指令的跳转功能可以正确执行，且处于间隙的指令按照协议要求也被正确执行。

图 3.27 所示是执行以下指令的情况：

`LW $1, 0x1000($0)`

`SLL $2, $1, 5`

`SW $2, 0x1000($0)`

这三句指令完成这样的动作：首先将数据存储器中地址为 `0x1000` 的字读取到寄存器 `$1` 中，然后将 `$1` 中的数进行逻辑左移，左移位数为 5，结果存入寄存器 `$2` 中，最后将 `$2` 中的数据写入到外部存储器的地址 `0x1000` 的字中。经过编译后得到机器码分别为：`0x8c010400`、`0x00011140` 和 `0xac020400`。

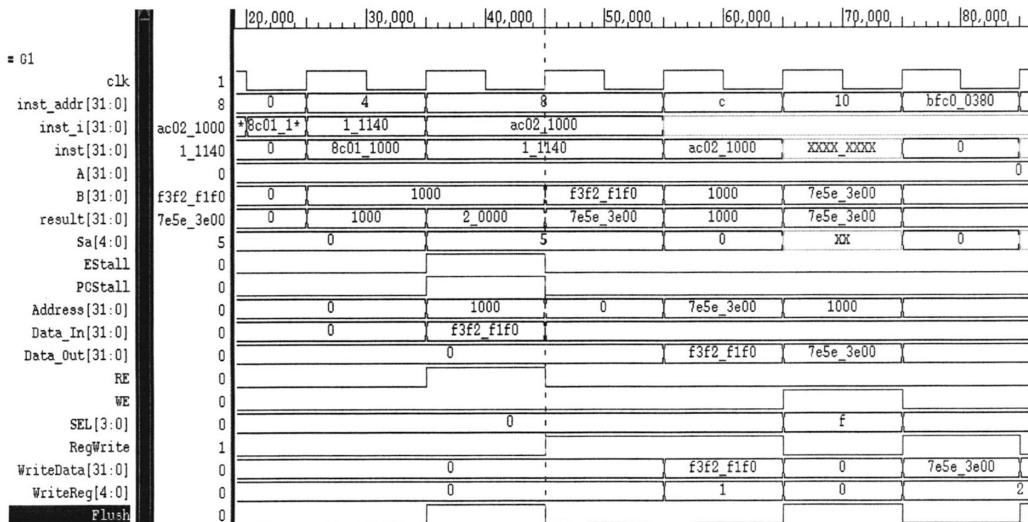


图 3.27 LW、SLL 和 SW 型指令的执行

Fig.3.27 Execution of LW, SLL and SW

第一个周期，指令 LW 处于流水线的 I 级，其指令被驱动到 `inst_i` 接口上等待被读取；第二个周期，LW 指令进入 E 级流水线，经过译码、计算后得到要读取的外部存储

器的地址为 0x00001000。PC 加 4 后，SLL 指令进入 I 级，并被驱动到 inst_i 接口上；第三个周期，LW 指令进入 M 级将读写外部存储器的信号驱动到接口上，可见图中的读使能信号 RE 变为高电平、读地址信号 Address 变为 0x00001000，从存储器中读取的数据被驱动到 Data_in 接口上等待下一个上升沿时被读取到寄存器中。此时 SLL 指令进入到 E 级，但由于需要使用的 \$1 数据此时刚刚被驱动到数据接口上还没有进入流水线，所以触发了流水线的暂停机制，产生一个流水线气泡。具体的动作如下：产生一个 PCstall 信号使 PC 值保持不变，产生一个 EStall 信号使 E 级流水线寄存器中的值保持不变，产生一个 MFlush 信号（即图中的 Flush 信号）使下一个周期进入 M 级的信息被清空。相当于丢弃在下一个周期进入 M 级流水线的信息，重新执行一次 SLL 指令。如图所示，如图 3.27 可见，流水线中分别产生了 PCStall、EStall、Flush 信号用于控制流水线气泡的产生。第四个周期，从存储器中读取的数据进入了流水线中，其数值为 0xf3f2f1f0，进入流水线后，该数据被转发到 E 级以供 ALU 计算 SLL 指令，PC 值加 4 后使 SW 指令进入 I 级，而刚刚读取的 LW 指令的数据进入 A 级。第五个周期，LW 指令进入 W 级，将写入寄存器堆的信息和控制信号驱动到寄存器堆的接口上等待写入，SLL 指令进入 M 级，SW 指令进入 E 级计算要进行存储操作的地址。第六个周期，LW 指令将要写入寄存器堆的数据写入完成，SLL 指令进入 A 级，SW 指令进入 M 级，将写入外部存储器的信息驱动到接口上，可以看出在这个周期中产生了 WE 信号、Address 信号、Data_out 信号；第七个周期，SLL 指令进入 W 级，将要写入寄存器堆的信息驱动到寄存器堆的接口上，准备将数据 0x7e5e3e00 写入 \$2 中。

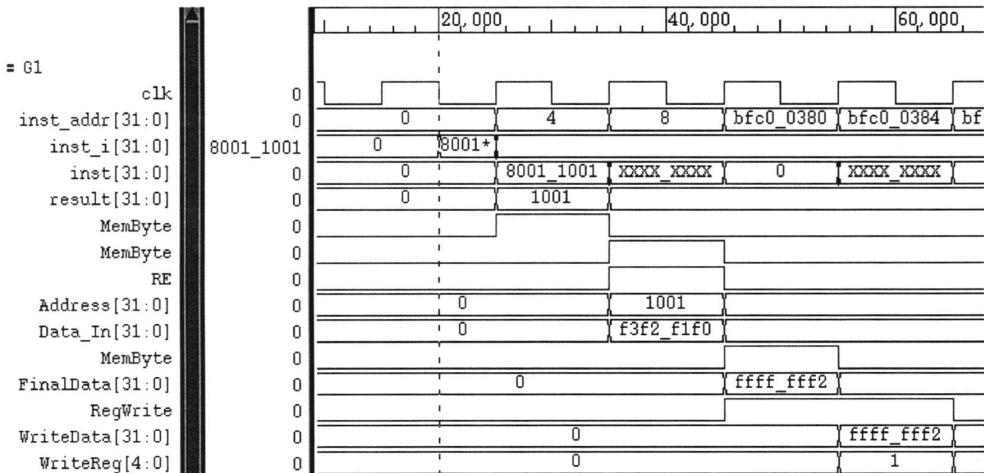


图 3.28 LB 指令的执行
Fig.3.28 Execution of LB Instruction

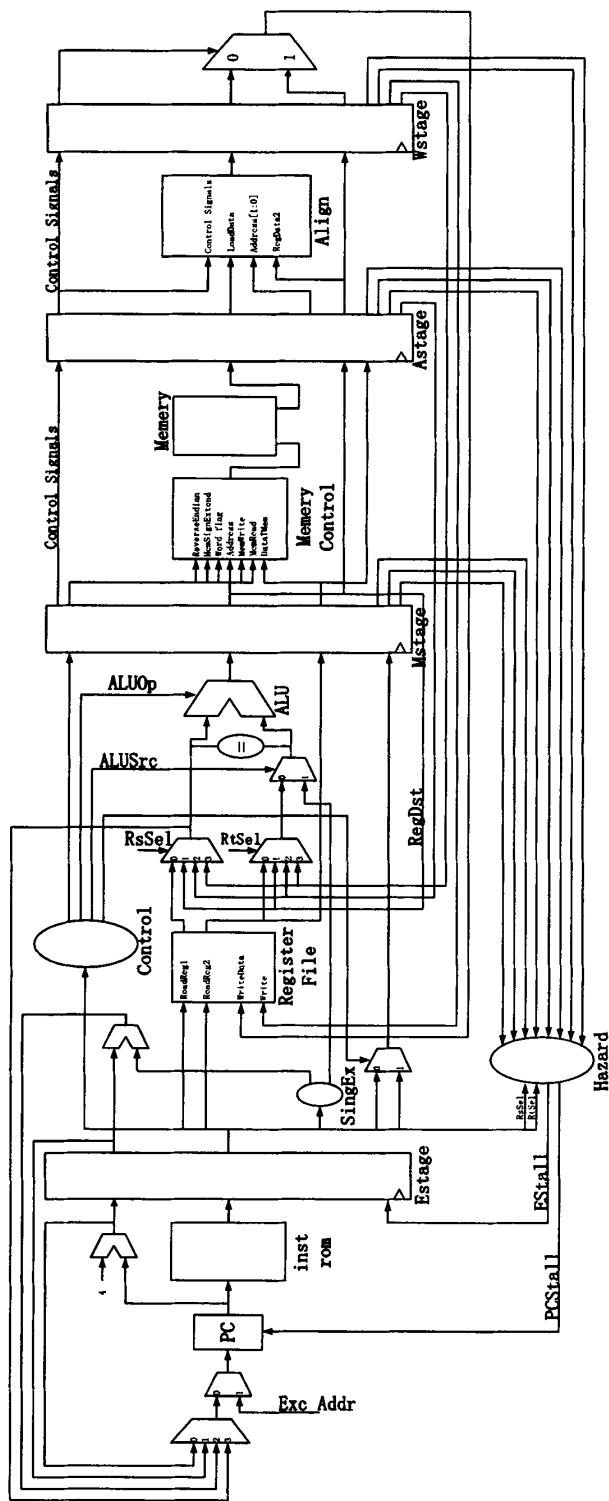


图 3.29 MIPS 处理器的整体结构
Fig.3.29 Architecture of MIPS Processor

图 3.28 所示为执行如下指令的情况：

LB \$1, 0x1001 (\$0)

这个指令的功能是读取指定的字节，并进行有符号扩展，将得到的结果存到指定的寄存器中。翻译成机器码为 0x80011001。第一个周期，指令处于 I 级，指令被驱动到 inst_i 上等待被读取。第二个周期，指令进入 E 级，由 ALU 计算出要进行读取操作的地址，并且有控制电路产生一个 MemByte 信号表示需要进行字节对齐操作；第三个周期，指令进入到 M 级，相应的控制信号 MemByte 也一起进入 M 级。在 M 级，处理器对存储器进行读取操作，产生读使能信号 RE、读地址信号 Address，并准备在下一个时钟周期将 Data_in 接口上的存储器数据读进流水线中。第四个时钟周期，指令进入 A 级。在 A 级，处理器将根据所读取数据的要求对数据进行处理。读取的数据为 0xf3f2f1f0，在大尾端的条件下，第一比特数据应该是 0xf2，进行有符号扩展后得到 0xffffffff2。第五个周期，指令进入 W 级，将写寄存器堆的写控制信号 RegWrite 拉高，写地址信号 WriteReg 置为 1，将准备写入的数据驱动到 WriteData 上等待下时钟上升沿时被读取到寄存器堆中。

以上的例子包含了 MIPS 处理器执行 I 型、R 型、J 型指令的情况，也简单地展示了流水线上数据冲突、流水线气泡等特殊情况的处理。经过全面地验证，MIPS 处理器的功能基本正确。图 3.29 是 MIPS 处理器的整体结构图。

4 DES 加密

数据加密标准 (Data Encryption Standard, DES) 曾是世界上最流行的对称加密算法。时至今日，虽然其国际标准已经被 AES 加密算法所取代，但 DES 算法和以 DES 算法为基础的 3DES 算法依然在很多领域被使用。

DES 算法是典型的 Feistel 型算法^[23]。所谓 Feistel 型算法，其特点就是在加密过程中同时会使用可逆和不可逆的模块。而非 Feistel 型算法则只使用可逆模块，例如 AES。在 Feistel 型加密算法中，分组长度、密钥长度、迭代轮数、子密钥生成算法和轮函数复杂度是影响算法安全性能的几个重要参数。DES 算法的加密过程如图 4.1 所示。

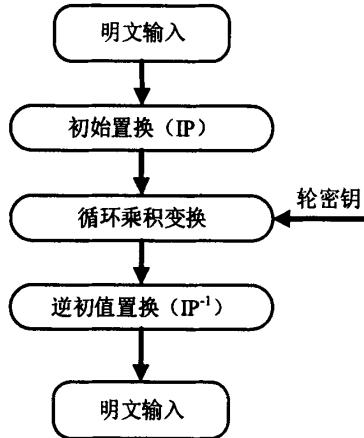


图 4.1 DES 算法流程

Fig. 4.1 The process of DES Algorithm

DES 算法数据块加密算法，每次加密的块长度为 64 位，也就是说输入的明文长度应为 64 位。明文输入后，首先经过一次初始置换 (Initial Permutation, IP)。IP 操作除了进行对输入明文的重新排序外没有其他的操作，经过 IP 操作后得到的 64 位值会进行 16 轮 Feistel 乘积变换，最后得到 64 位输出的中间值。最后将得到的 64 位输出值进行初始置换的逆置换，即逆初始置换 (IP^{-1}) 即可得到 64 位密文。

4.1 初始置换与逆初始置换

表 4.1 和表 4.2 分别是 DES 算法的初始置换表和逆初始置换表。表格按照从左到右、从上到下的顺序排列了置换后的 64 位码文，而表中的数字则代表置换前的码文位数。初始置换和逆初始置换是互逆的，也就是说，经过初始置换和逆初始置换后，其实码

文的顺序和原来是一样的。

表 4.1 初始置换表^[24]Tab. 4.1 Table of IP^[24]

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	55	37	29	21	13	5
63	55	57	39	31	23	15	7

表 4.2 逆初始置换表^[24]Tab. 4.2 Table of IP⁻¹^[24]

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

4.2 乘积变换

数据经过初始置换后，还需要再进行 16 轮乘机变换。乘机变换的流程如图 4.2 所示。码文经过初始置换后，得到 64 位的中间值。将这 64 位中间值分为左右两个部分，高 32 位为 L_0 ，低 32 位为 R_0 。然后通过 L_0 和 R_0 的乘积变换得到 L_1 和 R_1 。乘积变换时，右侧的 R_0 直接拿到左侧变为 L_1 。而右侧的 R_0 将要经过与轮秘钥 K_1 混合的运算得到 $F(R_0)$ 后， $F(R_0)$ 再与左侧 L_0 的进行异或运算后得到 R_1 。得到 L_1 、 R_1 后，再用 L_1 、 R_1 进行和轮秘钥 K_2 用相同的方式进行下一轮乘积变换得到 L_2 和 R_2 。如此循环，直到

最后得到了 L_{16} 和 R_{16} 。但最后一轮乘积变换和之前略有不同，之前的变换得到的结果其左侧部分都是直接使用上一轮的右侧部分来充当，右侧部分由上一轮的左侧经过计算得到，也就是说，每一轮的结算加过要经过左右交换的过程才能得到下一轮结果。但最后一轮不需要进行交换， R_{16} 由 R_{15} 直接充当，而 L_{16} 由 L_{15} 经过乘积变换得到。DES 算法的解密过程同加密过程是相同的，只是需要将加密时用的 16 个轮密钥倒过来使用。

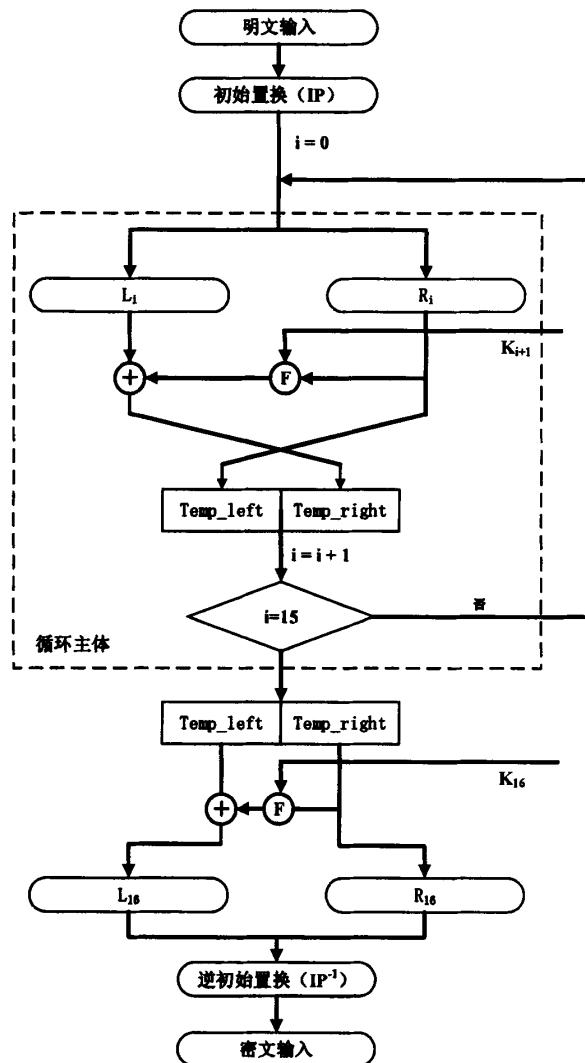


图 4.2 DES 算法乘积变换过程

Fig. 4.2 Process of DES Algorithm's multiplicative

乘积变换中的轮变换函数 F 的结构如图 4.3 所示。轮变换函数的输入为 64 位码文的低 32 位 R_i ，首先经过扩展变换 E 得到 48 位的 $E(R_i)$ 。扩展变换 E 也是一种置换运算，能够把输入的 32 位数据变为输出的 48 位数据，表 4.3 为扩展变换 E 的置换表，表格按照从左到右、从上到下的顺序排列了置换后的 48 位码文，而表中的数字则代表置换前的码文位数。

得到扩展变换的结果 $E(R_i)$ 后，用 $E(R_i)$ 与 48 位的轮秘钥 K_{i+1} 进行异或操作得到中间值 $K(R_i)$ ，再使用 S 盒对 $K(R_i)$ 进行替换操作得到 $S(R_i)$ 。DES 算法中共有 8 个 S 盒，每个 S 盒有 6 位输入，4 位输出。使用 S 盒时，将 48 位要进行替换的数据分组，从高到低六为一组，共八组。将这八组数据分别输入

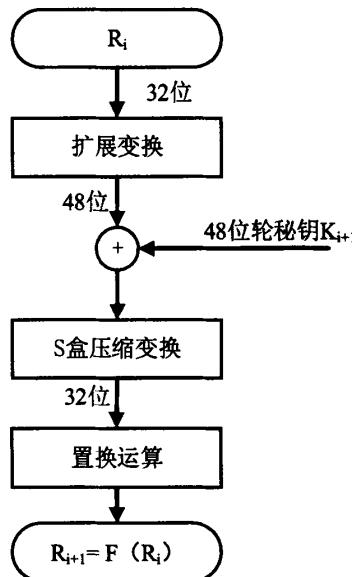


图 4.3 AES 轮变换函数结构

Fig. 4.3 Process of DES Algorithm's Round Function

到对应的八个 S 盒中，得到对应的输出数据，最后将得到的输出数据按照从高到低的顺序连接起来得到 32 位的 $S(R_i)$ 。每个 S 盒都可以看做输入数据的非线性变换，使用 S 盒的方法如下：设 6 位的输入数据 $K(R_i) = k_5k_4k_3k_2k_1k_0$ ，令 $t_0 = k_5k_0$, $t_1 = k_4k_3k_2k_1$ ，S 盒的输出为 $S_i(t_0, t_1)$ ，($i=1,1,2,\dots,8$) 其中 t_0 的取值范围是 0 到 3， t_1 的取值范围是 0 到 15。表 4.4 到表 4.11 分别是 S_1 到 S_8 的替换表。

例如，要进行 S 盒变换的码文最低 6 位 $k_0=101010$ ，则 $t_0=10$, $t_1=0101$ ，对应到 S_8 替换表中的数值为 $S_8(2,5)=12$ ，即输出的码文为 1100。

经过 S 盒变换得到 $S(R_i)$ 后，再对 $S(R_i)$ 进行 P 盒置换运算得到 $P(R_i)$ 。 $P(R_i)$ 就是最后需要的结果，即 $R_{i+1}=F(R_i)=P(R_i)$ 。表 4.12 为 P 盒置换表。

表 4.3 扩展变换 E 的替换表^[24]

Tab. 4.3 Table of E (Ri) [24]

32	1	2	3	4	5
4	5	6	7	7	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

表 4.4 S_1 盒替换表^[24]Tab. 4.4 Table of Sbox₁ [24]

		t1															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t0	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

表 4.5 S_2 盒替换表^[24]Tab. 4.5 Table of Sbox₂ [24]

		t1															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t0	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

表 4.6 S₃盒替换表^[24]Tab. 4.6 Table of Sbox₃^[24]

		t1															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t0	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	2	13	6	4	9	8	15	3	0	11	1	2	2	5	10	14	7
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	11

表 4.7 S₄盒替换表^[24]Tab. 4.7 Table of Sbox₄^[24]

		t1															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t0	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

表 4.8 S₅盒替换表^[24]Tab. 4.8 Table of Sbox₅^[24]

		t1															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t0	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

表 4.9 S₆盒替换表^[24]Tab. 4.9 Table of Sbox₆^[24]

		t1															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t0	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

表 4.10 S₇盒替换表^[24]
Tab. 4.10 Table of Sbox₇^[24]

		t1															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t0	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	3	6	11	13	8	1	4	10	7		5	0	15	14	2	3	12

表 4.11 S₈盒替换表^[24]
Tab. 4.11 Table of Sbox₈^[24]

		t1															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t1	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

表 4.12 P 盒置换表^[24]
Tab. 4.12 Table of Pbox^[24]

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

4.3 轮密钥生成

乘积变换中，每轮所用的轮密钥都是不同的。图 4.4 所示为轮密钥生成的过程。DES 要求输入的密钥为 64 位，但其中只有 56 位被用作轮密钥的生成，另外有 8 位密钥在生成过程中被丢弃，可以用来进行奇偶校验，或者完全随意输入。被丢弃的八位密钥分别是密钥的第 0、8、16、32、40、48、56、64 位。初始密钥首先经过一次初始选

择置换得到 $P_1(K)$ ，然后将 $P_1(K)$ 分为高、低密钥两部分，分别记做 C_0 、 D_0 。将 C_0 、 D_0 分别进行循环左移运算，即可得到 C_1 、 D_1 。将 C_0 、 D_0 连接起来进行输出选择置换 P_2 ，得到 $P_2(C_1, D_1)$ 。 $P_2(C_1, D_1)$ 就是第一轮的轮密钥，即 $k_1=P_2(C_1, D_1)$ 。得到 C_1 、 D_1 后，继续将 C_1 、 D_1 进行第二轮循环左移得到 C_2 、 D_2 ，继而得到 $k_{12}=P_2(C_2, D_2)$ 。如此循环共 16 次，得到全部 16 个轮密钥。在循环左移时，第 1、2、9、16 轮左移的位数为 1 位，其余 12 轮左移位数均为 2 位。表 4.13 为初始选择置换表；表 4.14 为输出选择置换表。

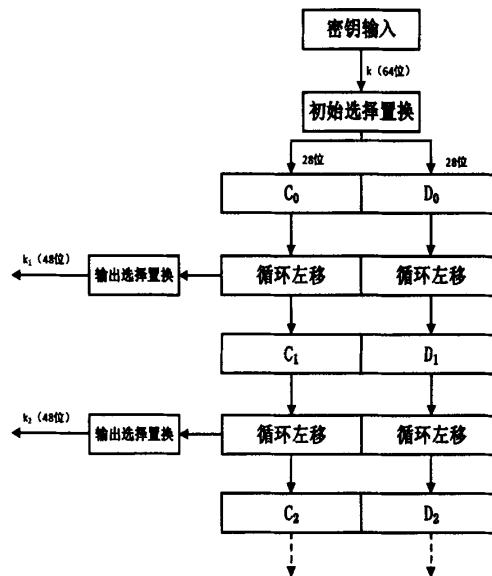


图 4.4 轮密钥生成过程

Fig. 4.4 Process of Round Key Generation

表 4.13 初始选择置换表^[25]Tab. 4.13 Initial Permutation of Key Generation^[25]

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

表 4.14 输出选择置换表^[25]Tab. 4.14 Output Permutation of Key Generation^[25]

14	17	11	24	1	5
3	28	15	6	21	10
21	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	33

4.4 3DES

对称码加密的安全保护主要在于密钥的安全，只要密钥不被泄露信息就是安全的。但随着计算机技术的进步，计算机的性能越来越高、计算速度越来越快，普通的 DES 只有 56 位的密钥长度已经难以抵挡计算机的暴力破解。因此，基于 DES 算法的 3DES 被提出。所谓 3DES 就是将 DES 加密重复进行三次，加密过程中使用两个或者三个不同的密钥加密，这样就能使加密算法的密钥扩展到 112 位或者 168 位，大大提高了加密算法的安全性能。

3DES 有四种不同的加密模式，分别是 DES-EEE3 模式、DES-EDE3 模式、DES-EEE2 模式、DES-EDE2 模式。

在 DES-EEE3 模式中，算法使用三个不同的密钥对明文进行连续三次加密得到密文。

在 DES-EDE3 模式中，算法使用三个不同的密钥按照加密、解密再加密的顺序得到密文。

在 DES-EEE2 模式中，算法使用两个不同的密钥对明文进行连续三次加密得到密文。其中第一次加密和第三次加密使用的密钥是相同的，第二次加密使用的密钥与第一次和第三次使用的密钥是不同的。

在 DES-EDE2 模式用，算法使用两个不同的密钥对明文按照加密、解密再加密的顺序运算得到密文。其中两次加密使用的是相同的密钥，解密使用的密钥与加密的密钥不同。

4.5 模块的实现

图 4.5 所示为 CFB 模式下的 DES 加密模块。

其工作方式如下：

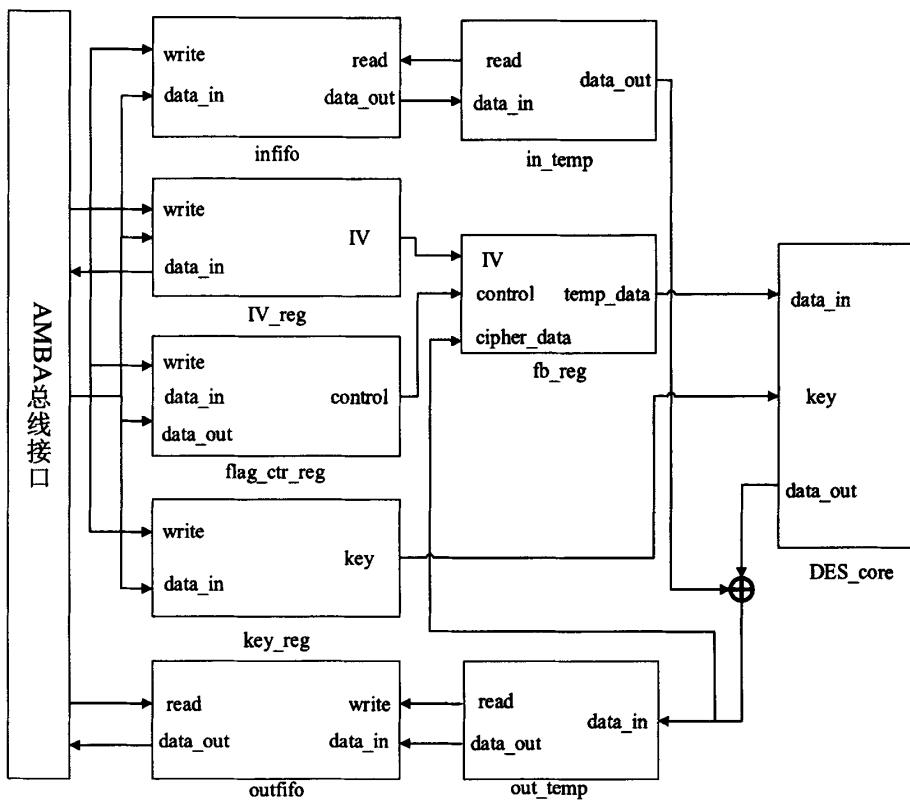


图 4.5 DES 算法模块结构图

Fig. 4.5 Architecture of DES Algorithm Module

- (1) 由总线向 infifo 中写入待加密的明文；
- (2) 由总线向 key 寄存器中写入密钥；
- (3) 由总线向 IV 寄存器中写入初始向量；
- (4) 由总线向 flag_ctr_reg 中写入控制位，控制位包括了 on、first_cyc、encipher、resetn_soft 等，其中 on 是模块工作的开关，on 为 1 模块正常工作，on 为 0 模块暂停工作，first_cyc 是首轮加密标志，使用一次后会自动置零，encipher 是工作方式选择信号，encipher 为高电平则模块进行加密操作，反之则模块进行解密操作。resetn_soft 是软复位。此外 flag_ctr_reg 还设置了 infifo 与 outfifo 的 full、empty 信号，以供主机读取、使用；
- (5) in_temp 读取 infifo 中的 2 个字，组成一组 64bit 的明文块。in_temp 是一个位移寄存器及其控制电路，主要的功能顺序从 infifo 中读取数据，使用时一次性读出。
- (6) DES_core 对 IV 或者 feed_back 值进行加密操作；

- (7) 得到的加密数据与明文块进行异或操作，得到密文；
- (8) 密文写入 out_temp 和 fb_reg 中，out_temp 也是一个位移寄存器，其功能是一次性写入 64bit 密文后顺序地按字将密文写入到 outfifo 中；
- (9) 将密文写入 outfifo 等待总线读取；

对所设计的 IP 核提取如下验证点：

- (1) AMBA 总线接口时序是否正确；
- (2) infifo、outfifo 读写功能、时序正确，空或者满时是否能够提供正确的标志位；
- (3) 总线能够正确完成对每个寄存器以及 fifo 的读写操作；
- (4) in_temp、out_temp 的功能、时序是否符合预期；
- (5) DES_core 是否能够正确地对数据进行加密；
- (6) 加密、解密操作时，fb_reg 是否能够提供正确的密文反馈；
- (7) 软复位、加密/解密选择、on 的开关功能、first_cyc 功能是否正确；

经过验证，所设计的 DES 加密 IP 核能够正确地进行加密、解密操作。总线接口时序正确，内部模块的功能与时序都符合预期。

图 4.6 至图 4.11 都是 DES 模块工作的时序仿真图。图中的 HCLK、HADDR、HRDATA、HREADY、HRESETN、HRESP、HWDATA 和 HWRITE 是 AHB 总线接口，DES_core_ld 是 DES_core 将要加密的数据加载到 DES_core 中的控制信号，标志着加密

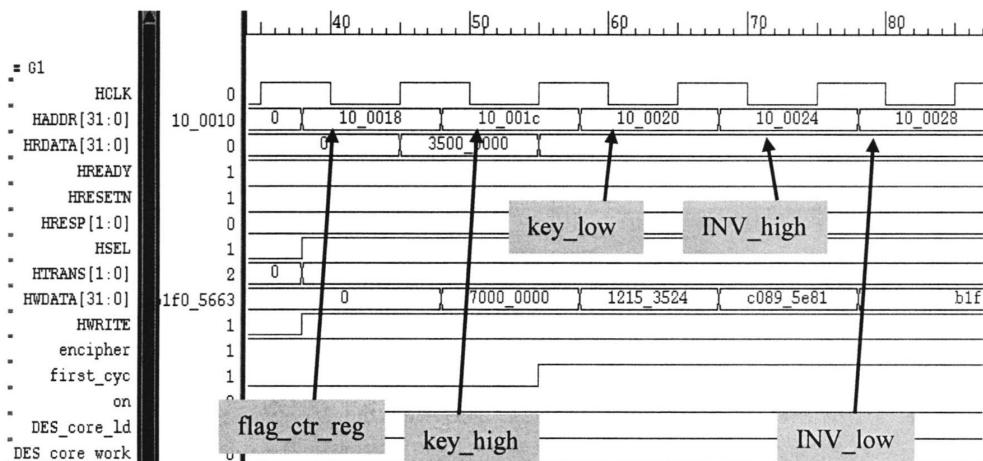


图 4.6 DES 模块工作过程 (1)

Fig. 4.6 Working Process of DES Algorithm Module(1)

操作的开始，DES_core_work 是 DES_core 开始工作后产生的标志。

图 4.6 中，由总线向 DES 模块中写入数据，一共写了五次数据，分别是写入 flag_ctrl_reg、key_high、key_low、INV_high、INV_low。可以看出，flag_ctrl_reg 中的 on 没有打开，而 encipher 和 first_cyc 被置为高电平，表示即将进行的操作是首轮数据加密。

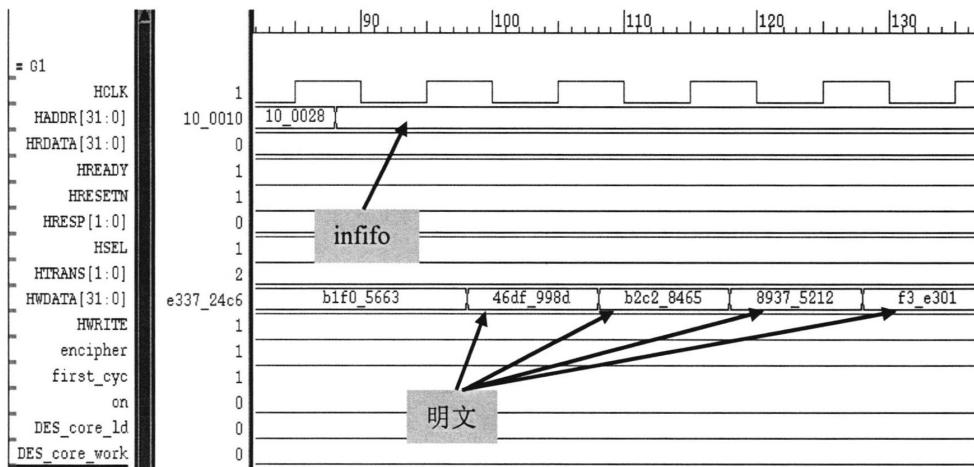


图 4.7 DES 模块工作过程 (2)

Fig. 4.7 Working Process of DES Algorithm Module(2)

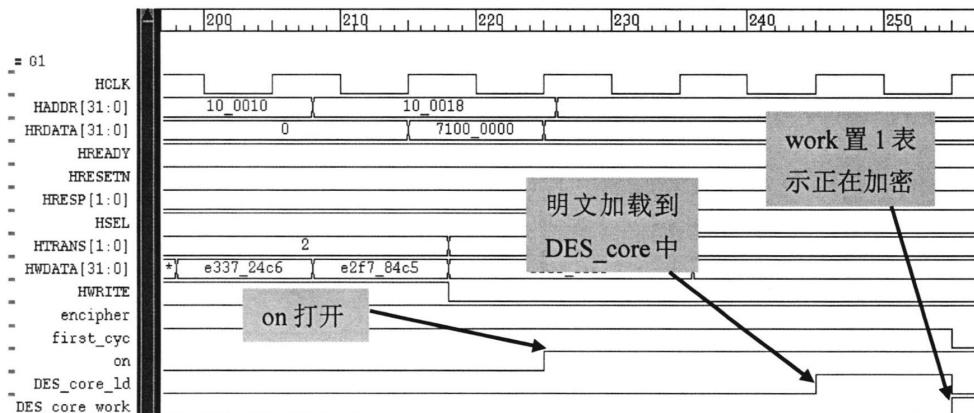


图 4.8 DES 模块工作过程 (3)

Fig. 4.8 Working Process of DES Algorithm Module(3)

图 4.7 和图 4.8 所示为总线向 infifo 中连续写入明文数据和控制信号。图 4.7 中，总线向 infifo 中连续写入明文。图 4.8 所示为明文写入完毕后，总线向标志与控制寄存器

`flag_ctr_reg` 中写入控制信号，将 `flag_ctr_reg` 中的 `on` 打开。可以看出 `on` 打开后，`DES_core_ld` 产生了一个高电平表示加密操作开始，首轮加密开始后，`first_cyc` 自动变为低电平。

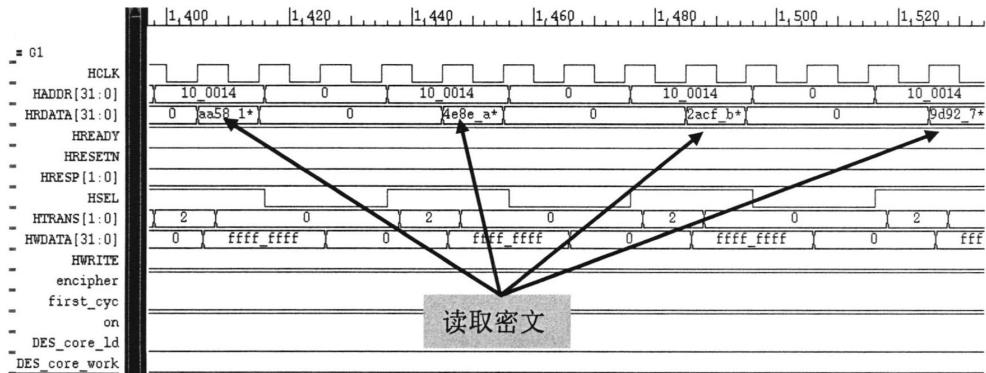


图 4.9 DES 模块工作过程 (4)

Fig. 4.9 Working Process of DES Algorithm Module(4)

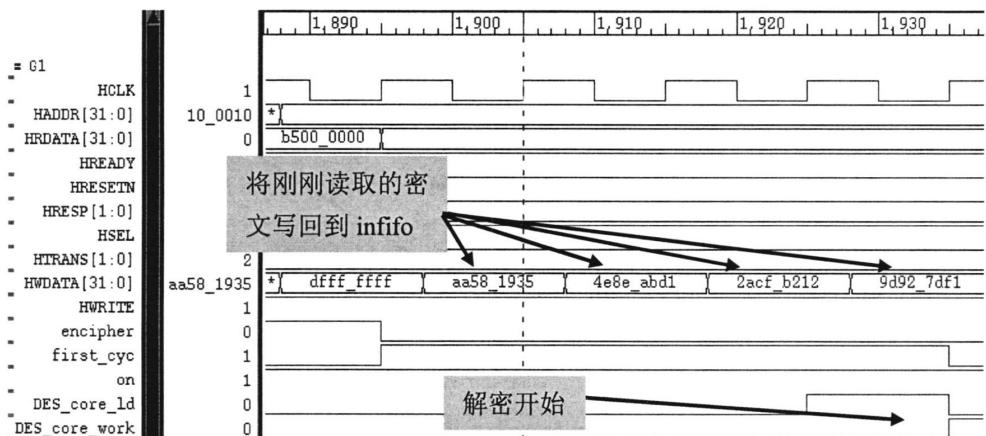


图 4.10 DES 模块工作过程 (5)

Fig. 4.10 Working Process of DES Algorithm Module(5)

图 4.9 表示加密结束后，总线将 `outfifo` 中的密文结果读出。图 4.10 表示总线将读出的密文再写回到 `infifo` 中，并将 `encipher` 设置为低电平，`first_cyc` 设置为高电平，表明将要进行数据解密的首轮操作。

图 4.11 表示总线将 `outfifo` 中解密后的明文读出。结果表明最后解密得到的明文和最初进行加密的明文是一致的。

图示中的数据如下：

```

key      = 64'h1215_3524_c089_5e81;
INV     = 64'h8484_d609_b1f0_5663;
P0      = 64'h46df_998d_b2c2_8456;
P1      = 64'h8937_5212_00f3_e301;
C0      = 64'haa58_1935_4e8e_abd1;
C1      = 64'h2acf_b212_9d92_7df1;

```

key、INV 分别密钥、初始向量，P 为明文，C 为密文。

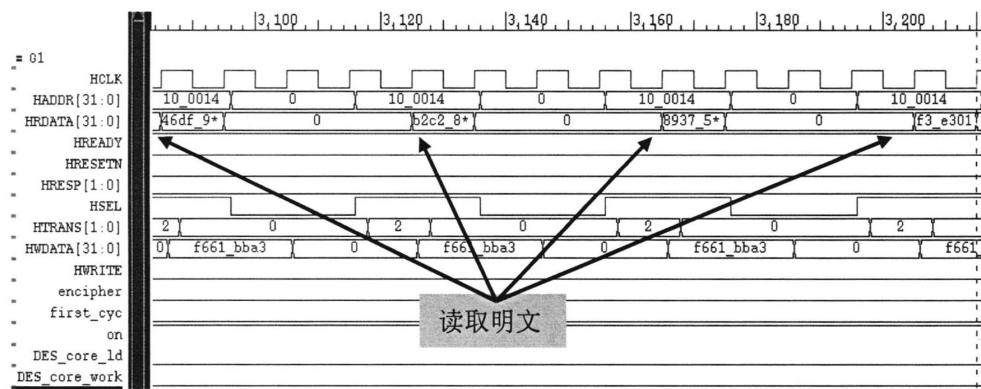


图 4.11 DES 模块工作过程 (6)

Fig. 4.11 Working Process of DES Algorithm Module(6)

5 AES 加密

高级加密标准（Advanced Encryption Standard, AES）是目前已知的最安全的对称加密算法之一^[26]。由于 DES 算法的性能愈显不足，美国国家标准技术研究所（NIST）在 2001 年 12 月发布了 AES 算法来代替之前的 DES 算法。AES 算法是 NIST 在 1997 年面向全世界公开征集，通过筛选确定的，其发明者是两名比利时学者 Vincent Rijmen 和 Joan Daemen。Vincent Rijmen 和 Joan Daemen 发明的算法原本叫做 Rijndael，被 NIST 采纳为高级加密标准，即 AES。Rijndael 算法是一种非 Feistel 加密算法，算法本身是可以对 128、192、256 的分组数据加密的，但 AES 中规定加密的分组长度为 128 位，密钥长度可以在 128、192、256 中选择。

5.1 加密运算

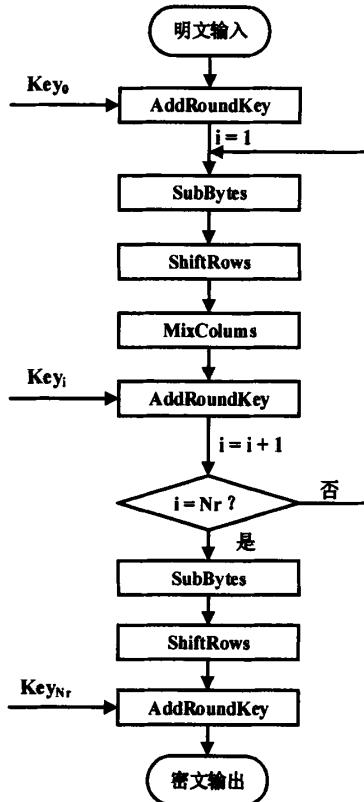


图 5.1 AES 算法加密过程
Fig. 5.1 Encipher Process of AES Algorithm

如图 5.1 所示为 AES 算法的加密流程。首先按照规定的长度输入明文，规定的长度为 128 位，即 4 个字的长度，记为 state0。将 state0 与初始密钥 key0 进行 AddRoundKey 操作，得到 state1。之后开始进行轮操作循环，轮操作循环的轮数和所使用的密钥长度有关，记为 Nr，表 5.1 为 Nr 与密钥长度的对应关系，其中分组长度为输入明文块的长度，密钥长度 Nk 和分组长度 Nb 的单位都是字。当循环的轮数小于 Nr 时，每轮操作依次进行 S 盒变换（SubBytes）、行位移（ShiftRows）、列混合（MixColumns）、轮密钥加（AddRoundKey）。最后一轮，循环轮数等于 Nr 的时候没有 MixColumns 操作，只有 SubBytes、ShiftRows 和 AddRoundKey。每循环一轮就记为一个新的 state，最后得到的 stateNr 就是输出的密文。

表 5.1 Nr 与 Nk 关系表
Tab. 5.1 Relationship Between Nr and Nk

分组长度	密钥长度 Nk	分组长度 Nb	加密轮数 Nr
128	4	4	10
192	6	4	12
256	8	4	14

5.1.1 轮密钥加计算

AES 算法的运算都是在有限域 $G(2^8)$ 中进行的字节运算。 $G(2^8)$ 是一个 256 阶的有限域，记为 $F=\{b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0 \mid b_i=0,1; i=0,1,2,3,4,5,6,7\}$ 。 $G(2^8)$ 规定了两种运算：“+”和“.”运算。

在 $G(2^8)$ 中，字节“+”运算即相当于两个字节的按位异或运算。AddRoundKey 操作实际上是当前 state 同轮密钥的异或。第一次进行 AddRoundKey 操作成为初始轮密钥加，所用的轮密钥是初始密钥本身，记为 $key0=\{w0, w1, w2, w3\}$ ，w 代表一个字，w0、w1 分别代表轮密钥的第一个、第二个字，依次类推。当第二次、第三次进行 AddRoundKey 操作时，初始密钥的长度已经不够，继而使用初始密钥的扩展密钥来代替，每扩展一轮，密钥的长度就会增加一倍。以 128 位的密钥为例，初始密钥记做 $key0=\{w0, w1, w2, w3\}$ ，经过一轮密钥扩展后，得 128 位的扩展密钥 $key1$ ，记做 $key1=\{w4, w5, w6, w7\}$ 。如此类推，直到密钥的长度满足加密轮数的需求。加密过程中使用密钥的顺序为 $w0, w1, w2, w3 \dots w_i$ ，解密时需要把加密时使用密钥的顺序按字颠倒过来使用。

5.1.2 S 盒替换操作

S 盒替换共分为两步操作，一是按字节求逆。在有限域 $G(2^8)$ 中，除了 0 之外的元素都有其乘逆。假设 a 在 $G(2^8)$ 中有乘逆，则 a 的乘逆记为 a^{-1} ，并且有在 $G(2^8)$ 中 $a \cdot a^{-1} = 1 \bmod f(x)$ 。S 盒替换操作中的求乘逆就是把当前的 state 按照字节来划分好，并分别按照字节来求其在 $G(2^8)$ 中的乘逆，用每个自己的乘逆来代替原来的字节。规定 0 的乘逆是 0。

将原来的字节都用 $G(2^8)$ 中的乘逆代替后，还需要对每个字节进行仿射变换。“ \oplus ”运算中的既约多项式为 设当前进行仿射变换的字节为“ $b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$ ”，变换后得到的新字节记为“ $b'_7 b'_6 b'_5 b'_4 b'_3 b'_2 b'_1 b'_0$ ”，则有如下等式：

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} \quad (5.1)$$

表 5.2 AES 算法 S 盒替换表^[27]Tab. 5.2 Sbox of AES Algorithm^[27]

63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

实际实现中，SubBytes 计算的过程过于繁琐，故多采用直接查表的方式进行替换。表 5.2 为 S 盒替换表，按照从左到右、从上到下的顺序依次列字节在十六进制中从 0x00 到 0xff 的替换。

5.1.3 行位移变换

一个 128 位的块可以分为 4 个字或 16 个字节进行操作。行位移操作就是基于字节进行变换的。假设当前的，其中表示一个字节，并且有 $i=0,1,2,3$ 和 $j=0,1,2,3$ 。将 state 按照如下方式排列：

$$\begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{bmatrix} \quad (5.2)$$

行位移变换对这个 state 的每一行都进行字节的循环左移。其中，第一行左移 0 个字节，第二行左移 1 个字节，第三行左移 2 个字节，第四行左移 3 个字节，得到新的 state。新的 state 如下式所示：

$$\begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{11} & s_{12} & s_{13} & s_{10} \\ s_{22} & s_{23} & s_{20} & s_{21} \\ s_{33} & s_{30} & s_{31} & s_{32} \end{bmatrix} \quad (5.3)$$

这个循环左移后的式子从左到右的每一列连接起来就是新的 state。

5.1.4 列混合变换

列混合变换规则如下：

$$\begin{bmatrix} s'_{0j} \\ s'_{1j} \\ s'_{2j} \\ s'_{3j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0j} \\ s_{1j} \\ s_{2j} \\ s_{3j} \end{bmatrix}, \quad (0 \leq j \leq 3) \quad (5.4)$$

其中 s_{0j} 、 s_{1j} 、 s_{2j} 、 s_{3j} 表示原来的 state， s'_{0j} 、 s'_{1j} 、 s'_{2j} 、 s'_{3j} 表示经过列混合变换后得到的新的 state 值。可以看出列混合变换是将 state 的每一列都作为多项式在 $G(2^8)$ 上进行“•”操作。其中的 01、02、03 分别指多项式 1、 x 、 $x+1$ ，既约多项式 $f(x) = x^8 + x^4 + x^3 + x + 1$ 。

5.2 解密运算

图 5.2 为 AES 算法的解密过程。从图中可以看出，AES 算法的解密过程实际上就

是加密过程的倒序操作，但加密和解密的计算是不同的，相对应于加密操作的 ShiftRows、SubBytes、MixColumns 变换操作，解密过程中分别设置了其逆操作，分别为 InvShiftRows、InvSubBytes、InvMixColumns 变换。解密的逆操作和加密的操作是相同的变换方法，只是运算的数值有所改变。另外，解密时使用的密钥是将加密时使用的密钥以字为单位倒过来使用的。

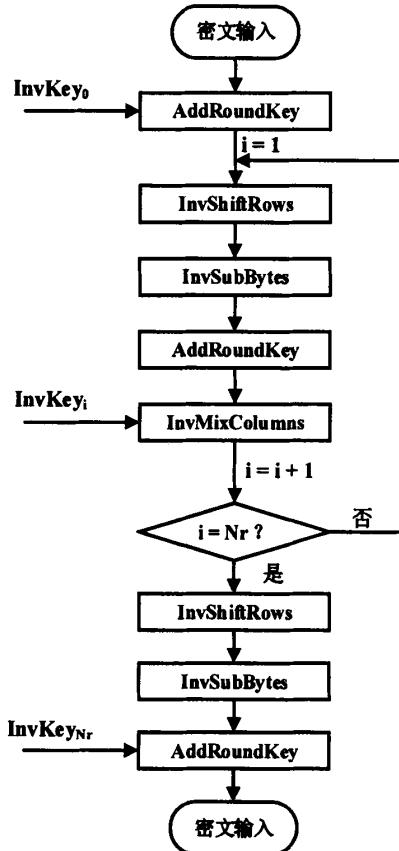


图 5.2 AES 算法解密过程
Fig. 5.2 Decipher Process of AES Algorithm

5.2.1 逆 S 盒替换

$InvSubBytes$ 变换首先是将 state 中的字节进行如式 5-5 的仿射变换。求出字节的仿射变换后，再求出字节在 $G(2^8)$ 中对应的乘逆即可得到新的 state。同样， $InvSubBytes$ 也可以通过查表的方式对 state 中的字节直接进行替换。表 5.3 为逆 S 盒替

换表。

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} \quad (5.5)$$

表 5.3 AES 算法逆 S 盒替换表^[27]Tab. 5.3 invSbox of AES Algorithm^[27]

52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
53	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
3a	91	11	41	4d	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

5.2.2 逆行位移变换

同加密时的 ShiftRows 相反, InvShiftRows 变换是将 state 中的每一列按字节向右循环位移。第一行右移 0 位, 第二行右移 1 位, 第三行右移 2 位, 第四行右移 3 位。若进

行 InvShiftRows 变换前的 state 如下：

$$\begin{bmatrix} S_{00}S_{01}S_{02}S_{03} \\ S_{10}S_{11}S_{12}S_{13} \\ S_{20}S_{21}S_{22}S_{23} \\ S_{30}S_{31}S_{32}S_{33} \end{bmatrix} \quad (5.6)$$

则进行 InvShiftRows 变换后的 state 如下：

$$\begin{bmatrix} S_{00}S_{01}S_{02}S_{03} \\ S_{13}S_{10}S_{11}S_{12} \\ S_{22}S_{23}S_{20}S_{21} \\ S_{31}S_{32}S_{33}S_{30} \end{bmatrix} \quad (5.7)$$

5.2.3 逆列混合变换

列混合变换规则如下：

$$\begin{bmatrix} s'_{0j} \\ s'_{1j} \\ s'_{2j} \\ s'_{3j} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0j} \\ s_{1j} \\ s_{2j} \\ s_{3j} \end{bmatrix}, \quad (0 \leq j \leq 3) \quad (5.8)$$

其中 s_{0j} 、 s_{1j} 、 s_{2j} 、 s_{3j} 表示原来的 state， s'_{0j} 、 s'_{1j} 、 s'_{2j} 、 s'_{3j} 表示经过列混合变换后得到的新的 state 值。可以看出列混合变换是将 state 的每一列都作为多项式在 $G(2^8)$ 上进行“•”操作。其中的 09、0b、0d、0e 分别指多项式 x^3+1 、 x^3+x+1 、 x^3+x^2+1 、 x^3+x^2+x 。既约多项式 $f(x) = x^8 + x^4 + x^3 + x + 1$ 。

5.3 密钥编排

AES 的密钥扩展算法是一个递归算法，必须已知前面的密钥才能计算后面的密钥。图 5.3 所示为 128 位密钥的扩展算法。从图中可见，每轮子密钥都是根据其上一轮秘密钥递归计算得来。每轮扩展密钥的第一个字是由上一轮密钥的第一个字异或 g 函数的输出得来的，而 g 函数的输入是上一轮密钥的最后一个字。除了每一轮的第一个字需要进行更多的函数运算外，其余的字都是由其在上一轮密钥中对应字异或其前一个字而得。196 位密钥和 256 位密钥的扩展方法同 128 位密钥的扩展方法是相同的，只不过每一轮扩展的字数不同。

图 5.4 所示为 AES 密钥扩展算法中的 g 函数。 g 函数的输入为每轮密钥的最后一个字，共四个字节，分别记为 b_0 、 b_1 、 b_2 、 b_3 。 g 函数首先将输入的字节进行顺序的调整，即按照字节循环右移 1 位，然后将得到的新字按字节进行 S 盒替换，再将得到的替换结果同轮常数异或，即得到输出的字。轮常数记做 $Rcon$ ，是一个 32 位的常数，每一轮

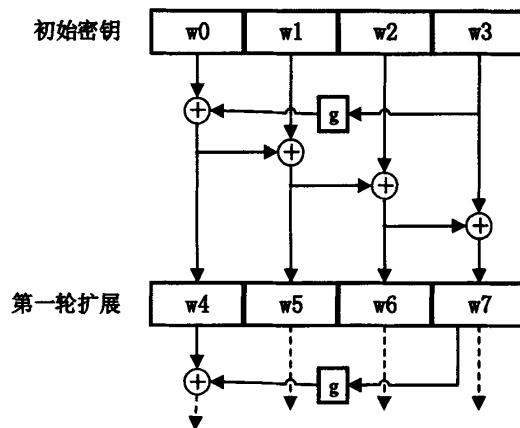


图 5.3 128 位密钥扩展算法
Fig.5.3 key extend algorithm of 128 bit

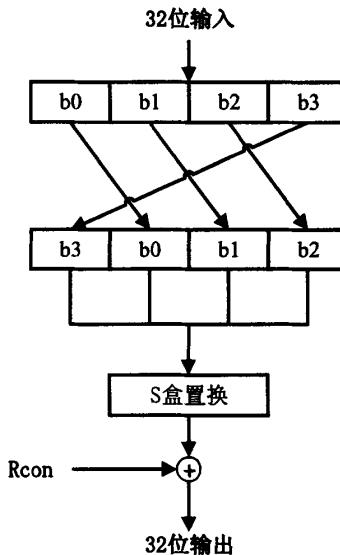


图 5.4 AES 密钥扩展算法中的 g 函数
Fig.5.4 Function of Key Extend Algorithm

的 Rcon 都不同，从第一轮扩展到第九轮扩展的轮常数如下：

$$\text{Rcon}[1] = 01000000;$$

$$\text{Rcon}[2] = 02000000;$$

$$\text{Rcon}[3] = 04000000;$$

```

Rcon[4] = 08000000;
Rcon[5] = 10000000;
Rcon[6] = 20000000;
Rcon[7] = 40000000;
Rcon[8] = 80000000;
Rcon[9] = 1b000000;
Rcon[10] = 36000000;

```

5.4 模块的实现

图 5.5 所示为本文设计的 CFB 模式下 AES 加密模块结构图。其工作方式与验证点同 DES 模块基本相同，这里不再赘述。经验证，模块的功能与时序均符合预期。

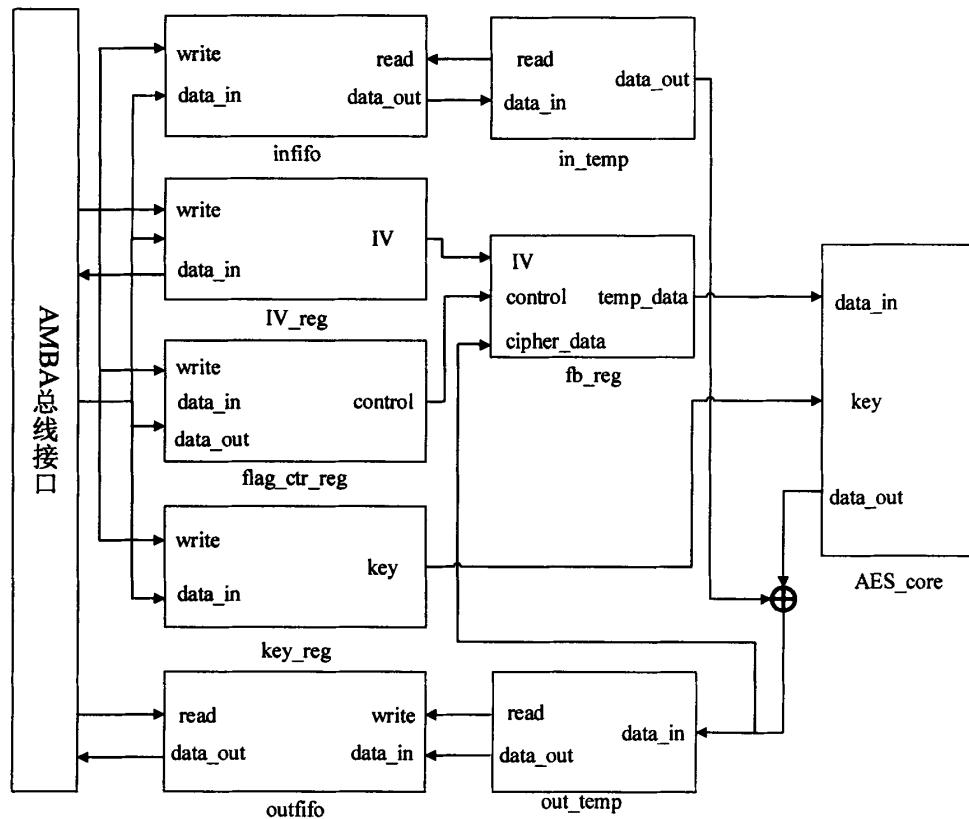


图 5.5 AES 加密模块结构图

Fig. 5.5 Architecture of AES Algorithm Module

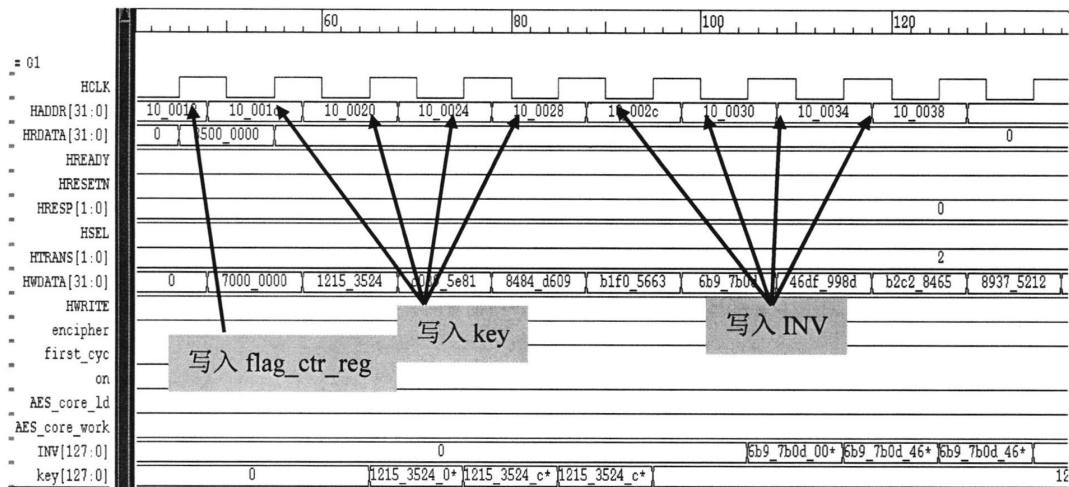


图 5.5 AES 模块工作过程 (1)

Fig. 5.5 Working Process of AES Algorithm Module(1)

图 5.5 至图 5.12 所示为 AES 模块工作结果的时序图。图 5.5、图 5.6 和图 5.7 所示为总线向 AES 模块中写入数据。图 5.5 表示先向地址为 0x00100018 的 flag_ctr_reg 中写入控制信号，将 on 关闭，并将 encipher 和 first_cyc 设置为高电平，表示将要开始的是数据加密的首轮操作。然后向地址为 0x0010001c 到 0x00100028 的 key 寄存器写入密钥。最后向地址为 0x0010002c 到 0x00100038 的 INV 寄存器中写入初始向量。

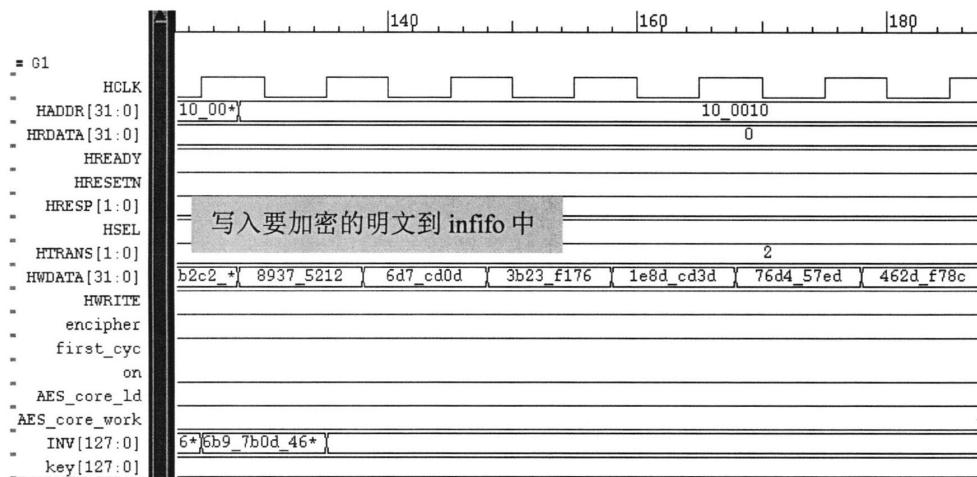


图 5.6 AES 模块工作过程 (2)

Fig. 5.6 Working Process of AES Algorithm Module(2)

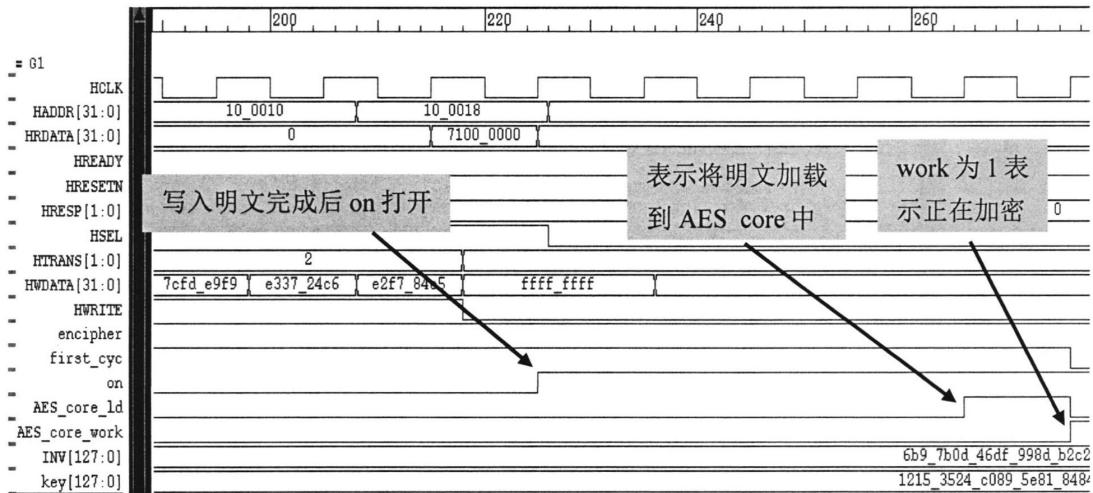


图 5.7 AES 模块工作过程 (3)

Fig. 5.7 Working Process of AES Algorithm Module(3)

图 5.6 表示向地址为 0x00100010 的 infifo 中写入要加密的明文。图 5.7 所示为明文写入结束后，总线再次向 flag_ctr_reg 中写入控制信号，将 on 打开。打开 on 后，可以看到表示 AES_core 将要加密的明文加载到 AES_core 中的 AES_core_ld 信号出现一个周期的高电平，表示着加密操作开始。并且首轮操作开始后，first_cyc 自动变为低电平。

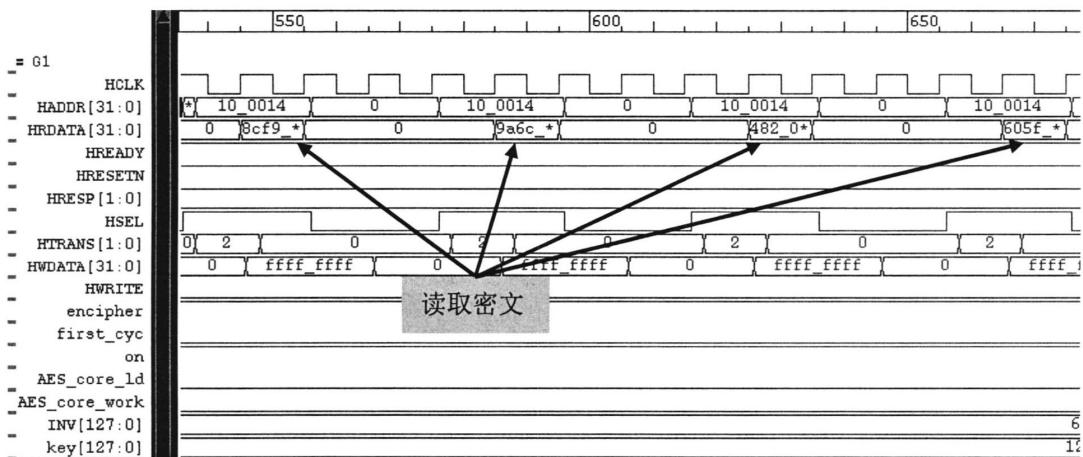


图 5.8 AES 模块工作过程 (4)

Fig. 5.8 Working Process of AES Algorithm Module(4)

图 5.8 和图 5.9 所示为加密完成后，总线从 outfifo 中将密文读出。图 5.10 所示为总

线将读出的密文再写入到 infifo 中，并将 first_cyc 设置为高电平，encipher 设置为低电平，表示将要开始的操作时解密操作。

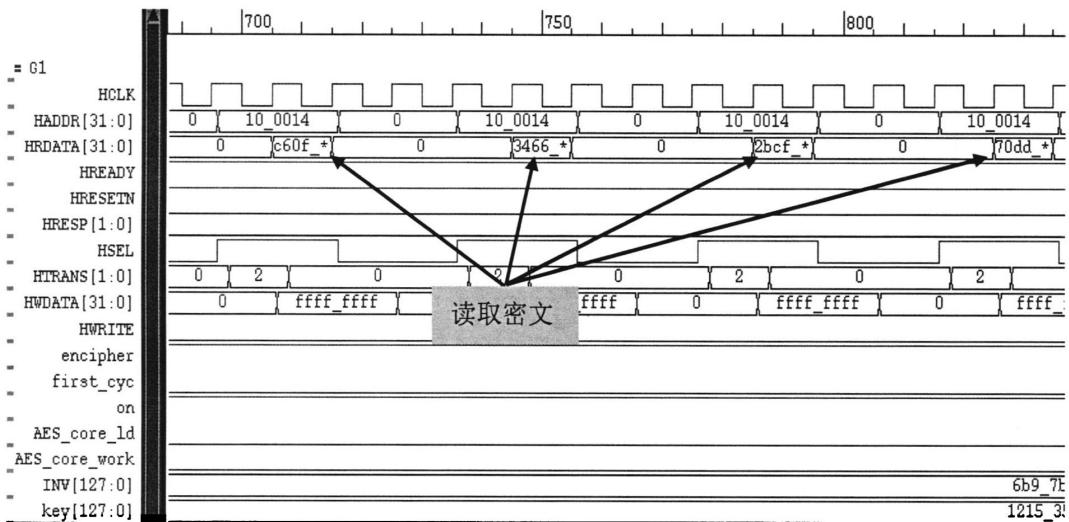


图 5.9 AES 模块工作过程 (5)

Fig. 5.9 Working Process of AES Algorithm Module(5)

图 5.11 和图 5.12 所示为总线从 outfifo 中将解密后的明文读出。结果表明最后解密得到的明文和最初进行加密的明文是一致的。

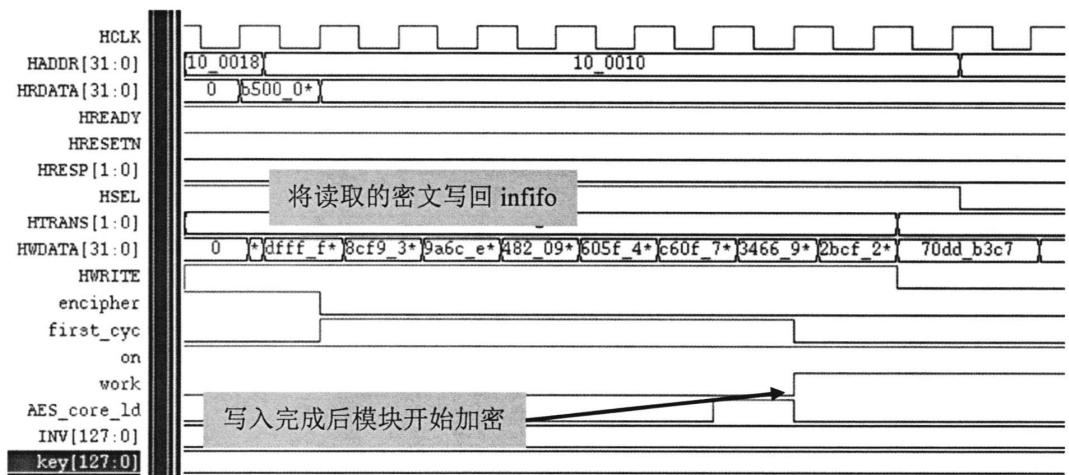


图 5.10 AES 模块工作过程 (6)

Fig. 5.10 Working Process of AES Algorithm Module(6)

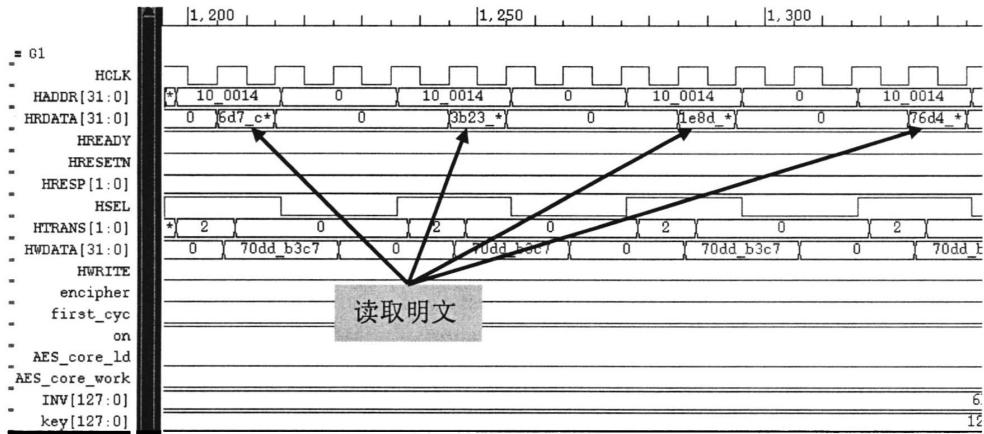


图 5.11 AES 模块工作过程 (7)

Fig. 5.11 Working Process of AES Algorithm Module(7)

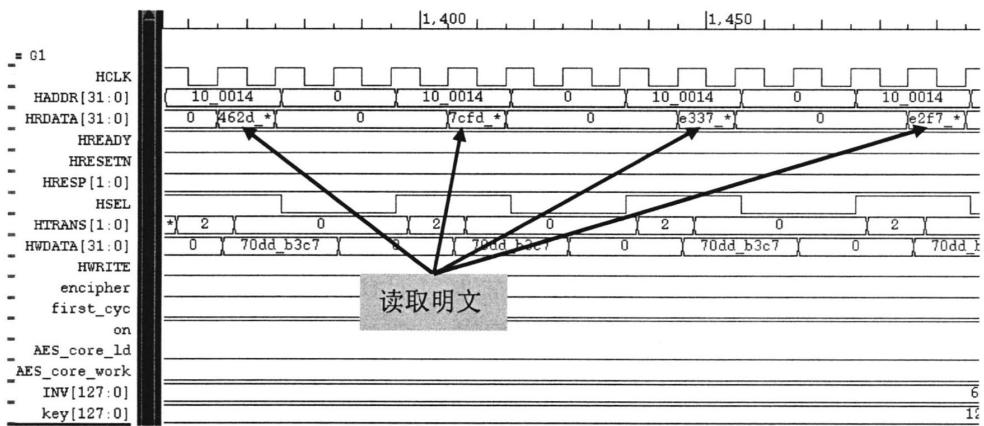


图 5.12 AES 模块工作过程 (8)

Fig. 5.12 Working Process of AES Algorithm Module(8)

图示中的数据如下：

```

key      = 128'h1215_3524_c089_5e81_8484_d609_b1f0_5663;
INV     = 128'h06b9_7b0d_46df_998d_b2c2_8465_8937_5212;
P0      = 128'h06d7_cd0d_3b23_f176_1e8d_cd3d_76d4_57ed;
P1      = 128'h462d_f78c_7cf9_e9f9_e337_24c6_e2f7_84c5;
C0      = 128'h8cf9_3676_9a6c_ed1c_0482_09f8_605f_43c8;
C1      = 128'hc60f_7b72_3466_9f3b_2bcf_2cbe_70dd_b3c7;

```

其中，key 和 INV 分别是密钥与初始向量，P 和 C 分别代表明文与密文。

6 SHA-1 算法

安全哈希算法（Secure Hash Algorithm,SHA）是一种杂凑加密算法，算法不可逆，即只能用来给数据加密，而无法进行解密，主要用于消息认证和签名^[28]。安全哈希算法目前有四个标准，分别是 SHA-1、SHA-256、SHA-384、SHA-512。从性能和安全等方面进行折衷的考虑，本文的设计采用了 SHA-1 算法。

SHA-1 算法能够将几乎任意长度的数据压缩为一个 160bit 的摘要，攻击者想要从 160bit 的摘要中复原出原文在不耗费相当大的资源时是很难做到了，非一般的攻击者能够实现。所以在一般用途的芯片中，使用 SHA-1 算法即已经能够保证数据的安全，不需要再耗费更多的资源来实现更高级别的算法。

SHA-1 算法有如下几个特点：

- (1) 输入为小于 264bit 长度的任意明文；
- (2) 输出为 160bit 的消息摘要；
- (3) 难以从摘要中还原出明文；
- (4) 发生碰撞(不同明文的密文相同)的概率很低，为 10^{-48} ；
- (5) 容易计算，消耗资源少。

6.1 SHA-1 的加密过程

第一步：填充长度、划分模块。SHA-1 能够对小于 2^{64} bit 长度的任意明文进行加密，但处理的数据须要划分成 512bit 长度的数据模块。其过程如下：

- (1) 在消息末尾添加一个 1 和若干个 0；
- (2) 在 0 之后连结一个 64bit 的数，代表真实消息的长度；
- (3) 将填充过的消息按照 512bit 的长度划分模块；
- (4) 假设明文的长度为 l ，添加 0 的个数为 x ，则 $x = (447-l) \bmod 512$ ，而最终得到的没有划分模块的消息为 $y = d \parallel 1 \parallel 0^x \parallel l$ 。其中 “ \parallel ” 表示连接， d 为真实消息， l 为表示真实消息长度的 64bit 数。

第二步：设置 IV 值。SHA-1 中有 5 个 32bit 宽的数据缓冲区，分别记为 A、B、C、D、E。这 5 个数据缓冲区用作中间值的保存。并且有一个固定的初始值。他们的初始值如下（十六进制）：

$$\begin{aligned} A &= 67452301; \\ B &= efcdab89; \\ C &= 98badcfe; \end{aligned}$$

$$D = 10325476;$$

$$E = c3d2e1f0;$$

第三步：分块处理消息。在这一步骤中，将消息块按照固定的运算方式分别处理，每一组消息的处理结果将作为反馈用于下一组消息的处理，直到最后一组消息处理完成，所得到的结果就是最后需要的消息摘要(Message Digest, MD)。消息处理的过程如下：

- (1) 将 512 bit 的消息划分为 16 个字 $W_t (t=0, 1, 2 \dots 15)$ ，即 $M_t = W_0 || W_1 || \dots || W_{15}$;
- (2) 产生 $W_t = S^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$ ($t = 16, 17 \dots 79$)，其中函数 S 代表循环左移， S^1 即是循环左移 1bit;
- (3) $(ABCDE)_i = f(H_{i-1}, (ABCDE)_{i-1})$;
- (4) $H_i = (H_{i-1} + (ABCDE)_i) \bmod 2^{32}$, $i = 0 \dots 79$;
- (5) $MD = H_{79}$;

步骤 (3) 中的函数 $f_i(H_{i-1}, (ABCDE)_{i-1})$ 是一个随着循环次数不同而变化的函数。但其运算过程不变，只是其内部的一个参数在不同的循环时期有不同的值。 $f_i(H_{i-1}, (ABCDE)_{i-1})$ 的运算如下：

$$\begin{aligned} A_t &= (E_{t-1} + f_t(B_{t-1}, C_{t-1}, D_{t-1}) + S^5(A_{t-1}) + W_t + B_t) \bmod 32; \\ B_t &= A_{t-1}; \\ C_t &= S^{30}(B_{t-1}); \\ D_t &= C_{t-1}; \\ E_t &= D_{t-1}; \end{aligned}$$

其中的 K_t 是一个随着循环次数增加而变化的量，在 80 次循环中共有四个值，每循环 20 次改变一次。其值如下：

$$K_t = 5a827999, (0 \leq t \leq 19);$$

$$K_t = 6ed9eba1, (20 \leq t \leq 39);$$

$$K_t = 8f1bbcd, (40 \leq t \leq 59);$$

$$K_t = ca62c1d6, (60 \leq t \leq 79);$$

函数数 $f_t(B_{t-1}, C_{t-1}, D_{t-1})$ 定义如下：

$$f_t(B_{t-1}, C_{t-1}, D_{t-1}) = (B \& C) | (\bar{B} \& D) \quad (0 \leq t \leq 19)$$

$$f_t(B_{t-1}, C_{t-1}, D_{t-1}) = B \oplus C \oplus D \quad (20 \leq t \leq 39)$$

$$f_t(B_{i-1}, C_{i-1}, D_{i-1}) = (B \& C) | (B \& D) | (C \& D) \quad (40 \leq t \leq 59)$$

$$f_t(B_{i-1}, C_{i-1}, D_{i-1}) = B \oplus C \oplus D \quad (60 \leq t \leq 79)$$

6.2 模块的实现

图 6.1 所示为 SHA-1 模块的结构图。模块由如下几个部分组成：AHB 总线接口、写入 FIFO、写出 FIFO、标志与控制寄存器、写入控制电路、写出控制电路与 SHA-1 核。其工作方式如下：

(1) 由总线向 infifo 中写入待加密的数据，infifo 是一个深度为 16，宽度为 32bit 的 FIFO，写满后刚好是 512bit 的模块；

(2) 写满 infifo 后向 flag_ctrl_reg 中写入控制信号，需要配置的控制位有 on、first_cyc、write、read、reset_soft。on 是电路工作的开关，为高电平时电路工作，低电平时电路停止工作；first_cyc 是首次加密标志，使用一次后自动置零；write 是用来控制 SHA-1_core 和写入控制电路读取 infifo 中数据的控制位；read 是来控制 SHA-1_core 和写出控制电路向 outfifo 中写入数据的控制位；reset_soft 是软复位。此外 flag_ctrl_reg 中还提供了 infifo 和 outfifo 的 full、empty 标志、SHA-1_core 的 busy 标志。

(3) 写入控制电路会将 infifo 中的数据顺序读出并传递给 SHA-1_core 进行加密操作；

(4) SHA-1_core 对数据进行加密，共 80 个周期；

(5) 如果还有其他的数据块需要继续加密，则重复上述操作，若全部数据块加密完成，则将 read 标志位置 1，SHA-1_core 和写出控制电路会将加密的结果分为 5 个字写入到 outfifo 中，等待总线读取；

针对所设计的电路提取如下验证点：

(1) 总线接口的功能、时序是否正确；
(2) infifo、outfifo 的读写功能、时序是否正确，是否能够提供准确的空/满标志位；

- (3) 总线是否能正确地完成读写操作；
- (4) SHA-1_core 是否可以正确地进行数据加密；
- (5) 写入、写出控制电路功能、时序是否正确；
- (6) 模块整体工作是否能够保证功能、时序正确；

经验证，内部各模块的功能、时序都正确，SHA-1 模块的功能、时序均符合要求；图 6.2 至图 6.5 是 SHA-1 模块的仿真时序图。图 6.2 表示由 AHB 总线向 SHA-1 模块中的 infifo 写入要加密的明文，一共 512bit，分 16 次写入。要加密的明文为 32'h1212_1212，

则写入 infifo 中被添加完整的数据为 $\{32'h1212_1212,1'b1,447'h0,64'h20\}$ 。首先，HSEL 和 HWRITE 信号变为高电平表示总线选中了 SHA-1 模块进行写操作，同时 HADDR 变为要 infifo 的地址 0x00100010、HTRANS 变为 2 表示传输状态为非连续传输。

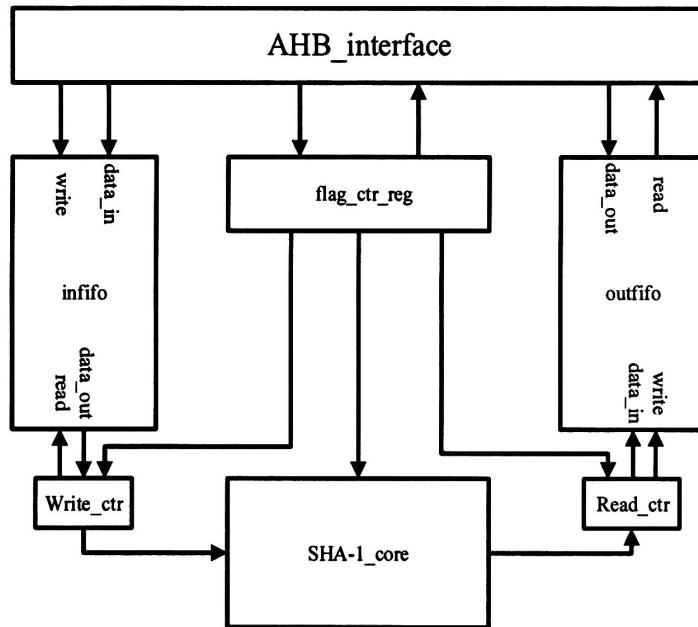


图 6.1 SHA-1 模块结构图

Fig.6.1 Architecture of SHA-1 Module

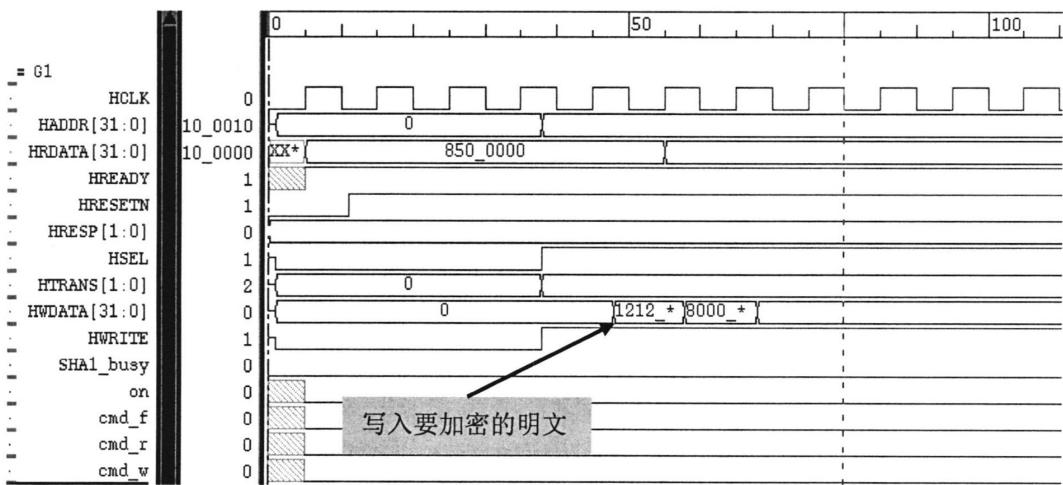


图 6.2 SHA-1 模块的工作过程 (1)

Fig. 6.2 Working Process of SHA-1 Algorithm Module(1)

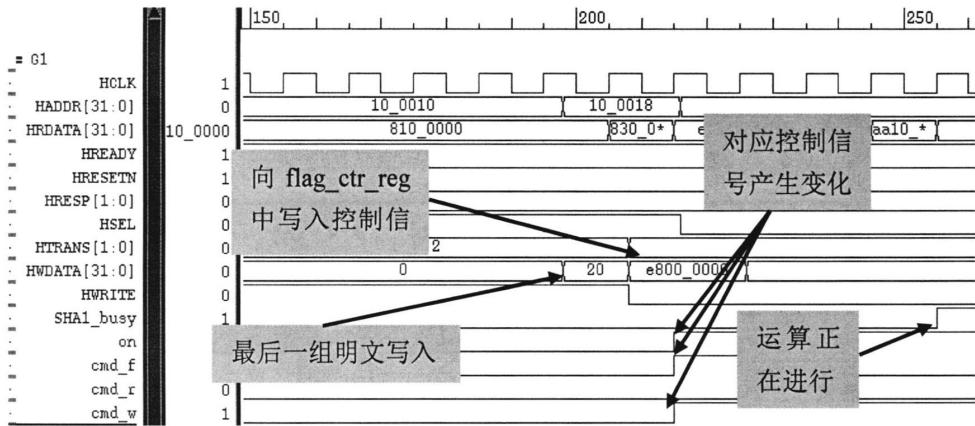


图 6.3 SHA-1 模块的工作过程 (2)

Fig. 6.3 Working Process of SHA-1 Algorithm Module(2)

图 6.3 表示写完要加密的数据后，向地址为 0x00100018 的 flag_ctr_reg 中写入控制信号，包括 on、cmd_f、cmd_r 和 cmd_w。正确写入控制信号之后，标志着 SHA-1_core 进入工作状态的 SHA1_busy 变为高电平，表示 SHA-1_core 开始进行加密操作，经过 80 个周期后加密操作完成。

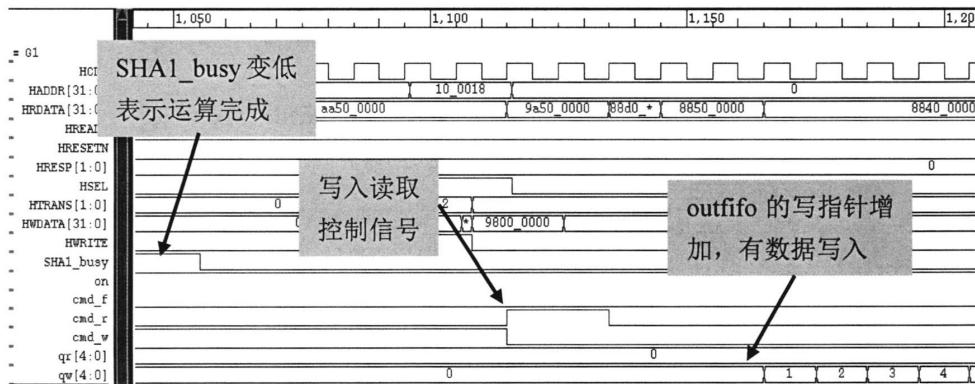


图 6.4 SHA-1 模块的工作过程 (3)

Fig. 6.4 Working Process of SHA-1 Algorithm Module(3)

图 6.4 所示为 SHA-1_core 对数据加密结束后，SHA1_busy 变为低电平。总线对 flag_reg_ctrl 重新写入数据，使 cmd_r 变为高电平，控制电路将加密后得到的消息摘要从 SHA-1_core 中读取到 outfifo 中。图 6.4 中的 qr 和 qw 信号分别是 outfifo 的读写指针，可以看出 cmd_r 脉冲出现的两个周期后 qw 值连续增高，表明 outfifo 中有五个数据写入。

入，正是 SHA-1_core 产生的 160bit 消息摘要。

图 6.5 表示总线从 outfifo 中读出密文。可以看出 outfifo 的读指针 qr 一次增大，表示有数据从 outfifo 中读出。被读出的五个数据如下

```
SHA1_C0 = 32'h5bea_a2dc;
SHA1_C1 = 32'h 9ffe_b9d4;
SHA1_C2 = 32'h 13af_bcc1;
SHA1_C3 = 32'h e816_05e3;
SHA1_C4= 32'h 3a00_282e;
```

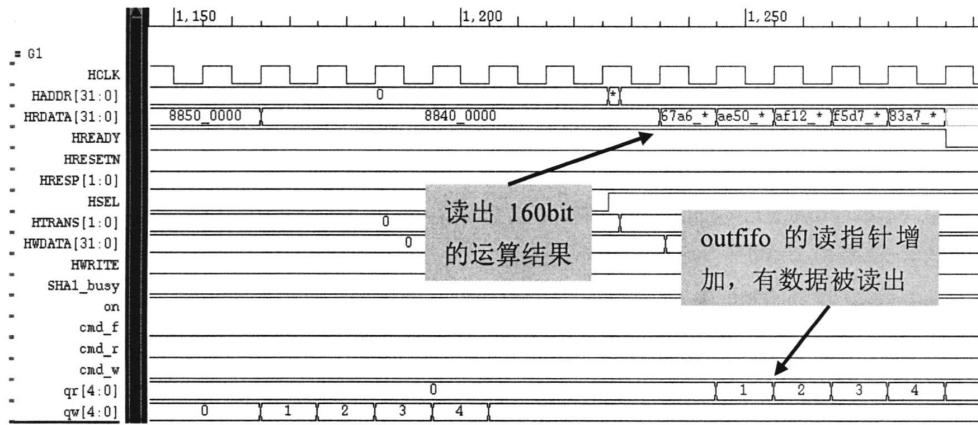


图 6.5 SHA-1 模块的工作过程 (4)

Fig. 6.5 Working Process of SHA-1 Algorithm Module(4)

结 论

本文的主要工作是参与完成一个基于 MIPS 处理器的 TPM 安全芯片设计。该 TPM 安全芯片是以一个以 MIPS 处理器为核心、通过 AMBA 总线挂载一些必要模块的 SOC 为具体实现方式的。MIPS 处理器具有执行速度快、硬件结构简单、开源等优点，非常适合在 TPM 安全芯片中作为主要的工作引擎。在要搭建的 SOC 中选用了 MIPS 处理器和 AMBA 总线。在加密模块方面，虽然 TCG 有意弱化对称加密而更推荐使用非对称加密，但考虑到一些实际的应用中使用对称加密会比非对称加密更加方便，所以设计中还是加入了 AES 和 DES 加密算法模块。至于 3DES 加密，作者认为可以通过软件配合硬件的 DES 核来实现，从而不必要单独添加一个 3DES 加密的模块。同样，在安全哈希算法方面，考虑到执行速度不宜过慢和硬件资源尽量节省，选用了 SHA-1 算法作为杂凑算法引擎。

本文承担并完成的模块有：

(1) MIPS 处理器核的设计。设计了一个 32 位的 MIPS 处理器核，在结构上借鉴了一些已经取得成功的产品设计并加以改进。总体结构采用了五级流水线，并完成了 MIPS32 releaseII 指令集的设计。此外，在 MIPS 处理器中还加入了冲突、异常等情况的处理机制以及中断机制。最后通过一些简单的汇编程序对处理器进行了功能验证，保证其功能基本正确。

(2) DES 加密模块设计。DES 加密算法曾是美国国家标准局用作数据加密标准，是当今应用最广泛的加密算法之一。本文设计了 DES 加密算法的内核，包括有初始置换模块、逆初始置换模块、乘积变换模块等，并设计了 DES 算法的轮密钥生成模块。最后对 DES 加密算法的内核进行整体的功能验证，保证其加密数据的准确性。根据 TPM 安全芯片协议的要求，设计了 CFB 工作模式的框架，将 DES 加密算法的内核嵌入其中，并为其设计了 AHB 总线的接口。最后对其时序、功能进行了验证，保证功能基本正确。

(3) AES 加密模块设计。AES 加密是目前最通用的对称加密算法之一，本文设计了一个 128 位 AES 加密的内核，包括有轮密钥加运算模块、S 盒替换模块、行位移模块、列混合模块等，并设计了 AES 加密算法的密钥生成模块。根据 TPM 安全芯片协议的要求，AES 加密模块的整体工作模式采用 CFB 模式，接口支持 AHB 总线。最后，对其模块整体的时序、功能进行了验证，保证功能基本正确。

(4) SHA-1 加密模块设计。SHA-1 是安全哈希算法之一，在 TPM 协议中可以作

为哈希引擎来使用。本文参考一些已有的设计，在安全哈希系列算法中进行了关于电路规模、执行速度、安全需求等方面的折衷考虑，最终选择了电路规模较小、执行速度快能够满足一般安全需求的 SHA-1 算法作为 TPM 安全芯片的哈希引擎。本文设计了 SHA-1 模块的加密内核，并编写了它与 AHB 总线的接口，使其能够在本文设计的 SOC 中工作。最后对其进行了功能与时序上的验证，保证其功能正确。

所有模块均是使用 verilog 语言进行 RTL 级设计，并用 VCS 进行编译和仿真。设计和验证过程中配合使用了 AMBA 总线的 VIP，将所设计的模块同已有的存储器控制器等模块一同挂载到 AHB 总线的验证模型上，保证了所有模块的接口时序能够很好地支持 AHB 总线，为以后的 SOC 集成工作打好基础。

本文所设计的并不是一个完整的 TPM 安全芯片，而是对 TPM 安全芯片中的一部分模块进行了设计与验证。预期的 SOC 中还缺少 HMAC 引擎、RNG、非对称加密算法模块等重要部分。而这些部分将会交由课题组的其他成员继续设计，最终完成一个 TPM 安全芯片的开发。

参 考 文 献

- [1] Department of Defense Computer Security Center. DoD 5200.28-STD. Department of Defense Trusted Computer System Evaluation Criteria[S]. USA:DOD, December 1985.
- [2] TCG Web Site. <http://www.trustedcomputinggroup.org>.
- [3] C114 中国通信网. PC 安全标准成众矢之的 TPM V1.2 成分水岭 [EB/OL]. <http://www.c114.net>.
- [4] 沈昌祥, 张焕国, 冯登国等. 信息安全综述, 中国科学 E 辑[J]. 信息科学, 2007, 37(2), 129-150
- [5] Shen Changxiang, Zhang Huanguo, Fengdengguo. Survey of Information Security[J]. Science in China Series F, 2007, 50(3) : 273-298
- [6] Shen Changxiang, Zhang Huanguo, Wang Huaimin, et al. Researches on trusted computing and its developments[J], SCIENCE CHINA: Information Sciences, 2010, 53(3) : 405-433.
- [7] 张焕国, 罗捷, 金刚, 等. 可信计算研究进展[J]. 武汉大学学报(理学版), 2006, 52(5).
- [8] 沈昌祥, 张焕国, 王怀民, 等. 可信计算的研究与发展[J]. 中国科学: 信息科学, 2010, 40(2) : 139-166
- [9] Jigar Patel, W. T. Luker Teacy, Nicholas R. Jennings, et al. A Probabilistic Trust Model for Handling Inaccurate Reputation Sources[C]. Trust Management, Third International Conference, iTrust 2005, Paris, France, 193-209, May 23-26 2005, Proceedings.
- [10] Beth T. Borchering M. , Klein B. Valuation of trust in open network[C]. In Proceeding of the European Symposium on Research in Security(ESORICS), Brighton: Springer-Verlag, 1994, 3-18.
- [11] 唐文, 陈钟. 基于模糊集合理论的主观信任管理模型研究[J]. 软件学报, 2003, 14(8) : 1401 - 1408.
- [12] 余发江. 可信计算 PC 平台关键技术与模糊信任理论[D]. 武汉: 武汉大学, 2007.
- [13] OpenTCWeb Site. <http://www.opentc.org/>.
- [14] Microsoft. Trusted Platform Module Service in Windows Longhorn[EB/OL]. <http://www.microsoft.com/>.
- [15] 网易科技报道. 联想等 12 家 IT 企业推出中国自主可信计算产品 [EB/OL]. <http://tech.163.com/07/1221/00/406PFIRJ000915BD.html>, 2007-12-21
- [16] 闵杰. 中国电子等 60 家单位发起中关村可信计算产业联盟成立 [EB/OL]. http://cyyw.cena.com.cn/2014-04/18/content_221943.htm, 2014-04-18
- [17] 刘振钧, 董贵山, 姜斌. 一种基于 TPM 技术增强嵌入式平台安全性的方法[J]. 信息安全通信与保密, 2009(2) : 104-107.

- [18] TCG. TPM Rev 2.0 Part 1 - Architecture 01.16[EB/OL].
<http://www.trustedcomputinggroup.org>.
- [19] 张焕国, 赵波等. 可信计算[M]. 武昌:武汉大学出版社, 2011. 23.
- [20] 迪建. MIPS 处理器芯片跻身 Andriod Wear 穿戴市场[J]. 集成电路应用, 2014. 8(26): 26-28.
- [21] MIPS, Technologies. MIPS® Architecture For Programmers Volume II-A: The MIPS32® Instruction Set[EB/OL]. <http://www.mips.com>.
- [22] Imagination, Technologies. MIPS® Architecture For Programmers Volume I-A: Introduction to the MIPS32® Architecture[EB/OL]. <http://www.imgtec.com>.
- [23] 胡向东, 魏琴芳, 胡蓉. 应用密码学(第3版)[M]. 北京:清华大学出版社, 2014.
- [24] 张洁, 朱丽娟. DES 加密算法分析与实现[J]. 信息安全, 2007. 2: 95-97.
- [25] 周建钦, 何凌云. DES 加密算法的密钥扩展[J]. 科技通报, 2011. 2(27): 263-267.
- [27] 吴文玲, 冯登国, 张文涛. 分组密码的设计与分析[M]. 北京:清华大学出版社, 2009.
- [28] Christof Pear, Jan Pelzl. 深入浅出密码学[M]. 北京:清华大学出版社, 2012.

致 谢

经过老师的指点以及和同学们的积极讨论，终于完成了我的毕业论文。在此，我要感谢一些陪伴我走过这两年美好时光的人。

首先要感谢的是我的导师张建伟老师。在完成毕业论文的过程中以及平时的生活和学习上，遇到不懂的问题，张老师都能耐心地给我提出许多宝贵的意见，每次都能使我获益匪浅。除此之外，张老师平时工作时严谨的态度、生活中谦逊的品格以及温和的性格都时时刻刻地影响着我，并将成为我一生受用的宝贵财富。

另外，师兄郑善兴、师姐丁秋红在我的学习、生活中都给了我许多至指导与帮助，在此也要感谢他们给我的莫大帮助。

同在一个实验室工作的腾飞、马万里、王政操和郝文凯也经常热心地与我讨论问题，同他们一起学习、工作的日子将会是今后美好的回忆。

还要感谢我可爱的女朋友冯丹，有她在身边陪伴使得学习和生活变得更加快乐。

最后要感谢所有陪伴着我的人，同时祝你们身体健康、工作顺利。

大连理工大学学位论文版权使用授权书

本人完全了解学校有关学位论文知识产权的规定，在校攻读学位期间论文工作的知识产权属于大连理工大学，允许论文被查阅和借阅。学校有权保留论文并向国家有关部门或机构送交论文的复印件和电子版，可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印、或扫描等复制手段保存和汇编本学位论文。

学位论文题目： 基于MIPS处理器的安全芯片设计

作者签名： 李伟理 日期：2016年6月12日

导师签名： 张建伟 日期：2016年6月12日