学校代号 10532　　学　　号 S180700592

分 类 号 TN492　　密　　级 不保密

湖南大學
HUNAN UNIVERSITY

硕士学位论文

MD5 和 SHA-256 算法研究与 FPGA 实现

学位申请人姓名 李　妮

培 养 单 位 物理与微电子学院

导师姓名及职称 王镇道　副教授

学 科 专 业 电子科学与技术

研 究 方 向 数字集成电路设计

论文提交日期 2021 年 5 月 13 日

学校代号：10532

学　　号：S180700592

密　　级：不保密

湖南大学硕士学位论文

MD5 和 SHA-256 算法研究与 FPGA 实现

学位申请人姓名：李　妮

导师姓名及职称：王镇道　副教授

培　养　单　位：物理与微电子科学学院

专　业　名　称：电子科学与技术

论文提交日期：2021 年 5 月 13 日

论文答辩日期：2021 年 5 月 22 日

答辩委员会主席：庄秀娟　教授

Research on MD5 and SHA-256 Algorithm Realize with FPGA

by

LI Ni

B.E.(Hunan University)2018

A thesis submitted in partial satisfaction of

the Requirements for the degree of

Master of Academic

in

Electronic Science and Technology

in the

Graduate School

of

Hunan University

Associate Professor WANG ZhenDao

May, 2021

湖 南 大 学

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

作者签名：李妮　　日期：2021年 5月 28日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权湖南大学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。

本学位论文属于

1、保密□，在_____年解密后适用本授权书。

2、不保密☑。

(请在以上相应方框内打“√”)

作者签名：李妮　　日期：2021年 5月 28日

导师签名：　　日期：2021年 5月 28日

摘 要

互联网技术的高速发展，给人们带来便利的同时也带来了许多安全隐患，不仅威胁人们的安全甚至威胁到国家的安全。信息安全已成为当下不容忽视的问题，密码技术是信息安全的重要保障，通过对明文信息进行加密传输，可以有效防止信息泄露和被窃取，是信息防护最重要也是最有效的手段。哈希算法是密码算法中重要的一员，可用于数字签名和验证签名，从而保证通信双方身份的真实性，另外广泛应用于各种密码系统和交换协议，保证信息传递的完整性。目前应用最广泛的哈希算法是 MD 系列和 SHA 系列算法，其中 MD5 算法和 SHA-256 算法的安全性高，应用广泛，最具有代表性。

本文研究了 MD5 和 SHA-256 算法，提出了算法架构的优化方案，并进行了硬件设计与验证。在 MD5 算法实现中，首先提出了利用插入中间寄存器，对输入数据进行预处理等方式优化算法单元数据通路，通过优化加法器设计，缩短循环迭代运算时间，从而减少电路的延时，缩短单步关键路径。其次，本文提出了 32 级流水线设计方案实现算法，将 2 步运算作为流水线的一级，在同一时间内运算两个步骤，提升算法运算速度，缩短消息运算时间，提高数据吞吐量。以 MD5 算法实现为基础完成了 SHA-256 算法硬件实现。

利用 Quartus II 以及 QuestaSim 软件进行设计和仿真验证，并使用 Altera Cyclone-V FPGA 开发板进行硬件实现与板级验证。实验结果表明，算法的最高时钟频率均达到了 173MHz，数据吞吐量超过 81Gbps。最后采用 0.18μm工艺进行 MPW 流片，芯片面积为 6mm^2，在工作电压 3.3V，时钟频率为 150MHz 时，功耗约为 10.7mW，测试结果表明，设计达到预期功能指标。

关键词：MD5 算法；SHA-256 算法；哈希算法；流水线设计；FPGA 设计

Abstract

The rapid development of Internet technology give people a great number of convenience, Meanwhile it hide a lot of danger. Those danger probably threatens not only people's safety, but also national security. At present, Information security has become a problem that can not be ignored. Cryptography technology is an important guarantee of information security, This technology can effectively prevent information from leakaging and stealing through the encryption transmission of plaintext information, It is the most important and effective means of information protection. Hash algorithm is one of the most important parts of the cryptographic algorithm, Hash algorithm can be apply to digital signature and verifying the signature, so as to ensure the authenticity of the communication status, This algorithm is widely used in various kinds of password system and exchange protocol to ensure the integrity of the information transmissions. At present, Hash algorithm is the most widely used, whose MD5 algorithm and SHA-256 algorithm are the most representative because of their high security and wide application.

In this thesis, Learning MD5 algorithm and SHA-256 algorithm and proposing architecture optimization scheme are helpful for achieving the hardware implementation and verification. At first, Data path of the algorithm unit is optimized by inserting the intermediate register and preprocessing the input data, The loop iteration operation speed became faster by optimizing the adder design, so as to reduce delay of circuits and shorten the single step critical path.Then this thesis proposed 32 level pipeline design architecture, using 32 level pipeline operating two steps at the same time, This way can enhance algorithm speed, shorten the operation time, improve the data throughput, Data throughput reached 81Gbps.The hardware implementation of SHA-256 algorithm is based on the implementation of MD5 algorithm.

Using Quartus II and Questasim software to design and verify simulation, Altera Cyclone-V FPGA is used for hardware implementation verification. The result indicated that the highest clock frequency of the algorithm reaches 173MHz and the data throughput exceeds 81Gbps. Finally, the 0.18μm process is used for MPW tape-out. The chip area is 6mm^2.When the working voltage is 3.3V and the clock frequency is 150MHz, the power consumption is about 10.7mW. The results showed that the design achieved the expected functional indicators.

Key words：MD5 algorithm；SHA-256 algorithm；Hash algorithm；pipeline design；FPGA design

目 录

插图索引

附表索引

第1章 绪论

1.1 研究背景及意义

互联网技术的高速发展给人们带来了许多便利的同时，也带来了诸多的问题。人们的生产和社会活动与网络密切相关，随时随地都在利用网络进行数据交互。如果重要的信息遭到不法分子的攻击而被窃取，可能会造成财损失甚至生命都将受到威胁，比如当下泛滥的电信诈骗行为就是不法分子通过不法途径获取个人私密信息，利用这些隐私信息骗取受害者信任并实施诈骗。在军事上，信息传递也需要依靠网络，如果他国间谍盗取了我国重要戎机，将直接威胁到我国安全，所以信息安全不仅是对于人们，对于国家来说也是重中之重，提供一个保证隐私性、完整性和真实性的信息生存环境是时代的需求，是迫切需要解决的问题。

信息安全是对数据处理系统建立安全保护，保护计算机硬件、软件、数据等要素不因偶然和恶意的原因而遭到破坏、更改和泄露。信息安全具有典型的三大特征，一是信息保密性，可确保只有获得授权的人才可对信息进行访问[1]；二是信息完整性，表现为信息完整以及未被损坏的状态；三是信息可用性，即允许用户或者其他系统对信息进行访问使用[2]。信息化时代面向大数据，面向互联网用户以及无线通信用户量急速增长，面向各类数据爆炸式增长，对安全措施和设备要求更为严格，保护利用开放通道传输的用户数据的安全也更加困难。现代科技和手持设备虽然满足了我们日常生活需求，如转账，投资管理，网上购物，网上预订等等，但是每一个优势的出现，一些劣势也应运而生，科技设备带来便捷的同时对信息安全的威胁也随之而来[3]。

在信息安全的研究领域中，密码学以及相关技术发挥着越来越关键的作用。密码学需要建立安全可靠的通信链路的各种技术，以此来保障数据传输过程中数据的安全性。加密哈希函数在密码学中扮演着至关重要的角色。虽然加密哈希函数与一般哈希函数具有一定的相关性，但是它却有一般哈希函数没有的附加属性。哈希函数的目的是确保系统或数据的完整性，而且可以与数字签名相结合进行使用，从而提供身份验证和身份不可否认的功能[4]。哈希函数主要特征是:

1. 单向不可逆性[5]。信息 M 通过哈希算法得到固定长度摘要 D，但是无法从 D 推出 M。

2. 无重复性。两个不同的信息 M1 和 M2 得到不同的摘要 D1 和 D2，并且 D1 和 D2 一定不同。

3. 计算简单，加密效果好。

常见的哈希算法包括有 MD5(Message Digest，信息摘要算法 5)以及 SHA(Secure Hash Algorithm，安全散列算法)[6]。互联网信息安全的重要性不言而喻，MD5 算法对于保证信息的完整性和安全性十分有效，因此国内外对 MD5 算法保持着极高的研究热度。NIST(美国国家标准与技术研究所，National Institute of Standards and Technology)针对安全性和最佳性能开发出 SHA 系列算法[7, 8]，最著名的哈希标准是 SHA-1、SHA-2 和 SHA-3，SHA-2 算法包括有 SHA-256、SHA-384、SHA-512。MD5 和 SHA-256 算法是当前国内外通用的密码标准，是国内外科研人员研究的重点。

MD5 算法是 MD 结构的典型代表，是密码学中应用广泛的一种哈希函数，在二十世纪末期由 Ronald Rivest 提出。MD5 算法将任意长度的输入数据可以压缩成只有 128-bit 的数据输出，算法符合散列算法的特性，具有不可逆性、数据完整性、不重复性等特点，可防止数据被篡改和数据丢失。SHA 散列算法是使用广泛的杂凑算法之一。由于 SHA0 和 SHA1 在发展过程中被接踵证实算法存在漏洞，故很快被 SHA-2 算法取代。虽然 SHA 系列算法结构类似，但由于 SHA-2 算法压缩生成的摘要值位数增加，有效避免 SHA0 和 SHA1 存在的问题。

哈希算法可以利用软件进行实现。哈希函数需要进行密集计算，由于这些函数的迭代性质，利用简单的并行技术提高哈希算法的计算性能，效果不佳，而且在通用处理器上执行的哈希算法软件常用于许多身份验证协议以及独立的应用程序中，极其依赖计算机硬件平台，而且算法运算时间长，运行效率低，不能满足物联网高速发展的需求，故使用硬件实现的哈希函数极大的提高了运算的效率，实时性和安全性都比软件实现高。就成本来说，虽然比软件实现方式要高，但是相比于性能上的提升完全可以接受，所以越来越多的哈希函数都通过硬件的方式来实现。

执行时间和数据吞吐量是两个比较软硬件实现优劣性的性能指标，当算法执行时间越短，数据吞吐量越高时，证明哈希函数性能就越好。哈希算法通过硬件方式实现时可以根据需求实现不同的性能，执行时间、数据吞吐量、硬件资源占用率，都是重点考虑的三个性能指标。例如在金融证券行业中要求实时处理千千万万的数据，须利用硬件实现大幅度提升数据吞吐量；在微芯片卡应用时，寻求面积和性能的平衡，实现占用更少面积，满足更高性能的理想状态。对于这样的应用场景，通常情况下，为了提高数据吞吐量常以牺牲硬件资源为代价，通过增大面积实现更高的性能。

采用 FPGA(Field-programmable-gate-array，现场可编程门阵列)设备进行算法硬件实现是一个非常吸引人的选择。因为 FPGA 即现场可编程门阵列不仅能够实现高速度运行，具备算法敏捷性和灵活性从而可以谋求动态系统的优化，还可提供哈希算法所需要的高度优化环境，由于外部攻击者无法访问哈希算法在计算过

程中的中间计算结果，因此哈希算法的硬件实现具备更好的安全性。FPGA 的特性为半定制，可重复编程，较其它硬件方案有着极高的实现灵活性[9]，可满足哈希算法对灵活性设计的需求。综合来看，FPGA 硬件实现优点是可进行重复配置，功耗低，效率高且具备突出的并行计算能力，FPGA 自问世以来就聚焦了高校和科研院的科研人员、企业研发人员的关注，并对此做出了深入研究。

综上所述，研究 MD5 和 SHA-256 算法的硬件实现是必然趋势，硬件实现方案对比软件实现方案至少能获得几个量级的速度提升，实现的算法速度快，实时性强，安全性高。优化硬件结构来实现加密算法是未来的主流，不仅提高了算法运算速度，加大数据运算吞吐量，而且避免病毒攻击等问题，具有重要的研究价值。

1.2 加密解密算法简介

数据加密技术是对信息进行保密处理的一种技术，通过对信息进行一系列处理，从而使得原信息被掩蔽，防止信息泄露，就算攻击者获得了信息，如果没有密钥的情况下，也无法破解出原信息。目前数据加密技术涉及到的常用加密算法有以下几种：

1. 对称密钥密码算法。使用相同密钥对明文进行加密和对密文进行解密的一种算法[10, 11]。由于这种密码算法密钥相同，所以通常情况下，在进行加密和解密时使用的算法是相似的甚至是完全相同的[12-14]，例如 DES(Data Encryption Standard)密码算法在加密和解密时使用了完全相同的算法函数[15]。唯一不同的是在加密和解密时使用密钥的顺序不同，因此密钥安全是决定整个数据是否安全的关键[16]。

对称密码算法虽然加解密过程速度快，可加密数据量大且加密算法对公众公开，方便使用，但缺点是密钥安全分发给 A、B 双方不易，秘钥管理复杂，而且容易泄露，风险高，容易被伪装身份的人攻击，从而造成密钥泄露。造成上述缺点最主要的原因是对称密钥加密/解密算法的特性使得参与安全通信的各方都需要拥有相同密钥，密钥传输需要通过一个绝对安全的通信链路进行，以防止密钥在传输过程中遭到攻击而被窃取，只有成功交换了密钥，才可以使用通信渠道进一步进行安全的通信[17, 18]，否则通信就存在风险。这是算法上的缺陷，从理论上没有办法可以解决，所以高要求的密钥生成过程以及更高要求的安全传输是一个重大的挑战。对称密钥密码加密解密模型如图 1.1 所示，A 和 B 希望在一个传输通道上传输机密消息 M，那么 A 需要将明文使用密钥 Key1 进行加密，通过物理传输信道将密文传送给 B，B 收到密文后使用相同的密钥 Key1 进行解密得到明文。假设密文在传输信道上被监听窃取，理论上在没有密钥 Key1 的情况下，窃取者无法破解其中的信息[19, 20]，但是该模型下 A 和 B 的通信前提是得到了 Key1，

而 Key1 也是通过物理传输信道分发的，如果 Key1 在分发过程中遭到攻击泄露，那么 A 和 B 之间的通信就存在着风险，攻击者可以冒充 B 的身份跟 A 进行信息交互。

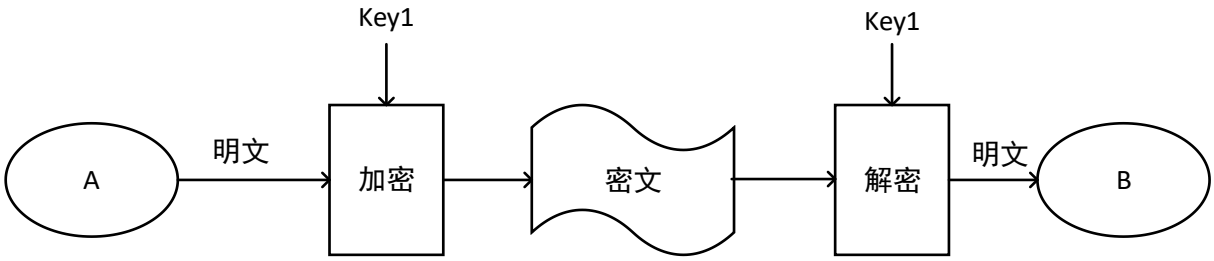

图 1.1 对称密钥算法加密解密模型

2. 非对称密钥密码算法。通信双方 AB 通过一个共用的传输数据通道使用两个不同的密钥对彼此进行互相通信，算法需要使用到两个具有相关性的密钥，分别称为公共密钥即公钥 pk 和私密密钥即私钥 sk。虽然这两个密钥存在关联性，但不允许通过私钥推导出公钥，反之亦然[21, 22]。在进行通信时，通信一方 A 发送数据是通过对方的公钥对信息进行加密，然后另一方 B 接收到密文后使用自己的私钥进行解密，所以公钥可以对外公开，但是私钥一定要妥善保存，防止被窃取。使用这种方法，即使是陌生人也可以加密消息，在没有相应的私钥的情况下仍然无法解密密文[23]。这一算法毋庸置疑是安全性更高的一种，但由于公钥是用户周知的，所以任何人都可以通过公钥给接收方发送消息，从而身份的真伪辨别成为了一个重要问题。

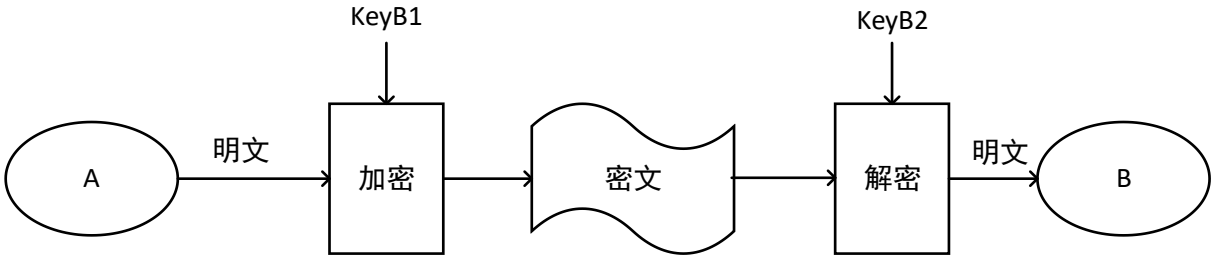

图 1.2 非对称密钥算法加密解密模型

公钥密码加密解密模型如图 1.2 所示，在图中 KeyB1 是 B 的公钥，是对外公开的，KeyB2 是 B 的私钥，理论上 KeyB2 只有 B 才知道。加密和解密算法是 AB 双方均知道的，A 与 B 进行数据通信时，A 需要发送的数据是通过 KeyB1 对消息进行加密，然后通过传输信道发送给 B，B 方接收 A 发送过来的密文信息后，通过使用自己的私钥 KeyB2 进行解密，辨别 A 身份的真实性，从而获得消息。非对称或公钥密码学的优点是其密钥分发系统优于对称系统。与对称系统相比，有更好的可伸缩性。此外，它还可以提供机密性和身份验证，保密性比较好，它省去了最终用户双方 A 和 B 互换 KeyB1、KeyB2 的步骤，但是此类算法加解密过程耗时长、处理速度慢、适用领域小。

3. Hash 加密算法，如 MD 系列算法和 SHA 系列算法。其特点是只进行加密，不可进行解密。算法的思想是接收一段位宽不定的明文，以一种原始数据不能被理解的方式将明文转换为无法识别的数据。Hash 加密算法模型如图 1.3 所示，Hash 算法接收输入数据(称为明文消息 M)，将它们转化为数据位数固定、长度较短的输出数据 (称为摘要值 D)，但值得注意的是，接收消息 M 可以运算得到摘要值 D，但是输出数据 D 推出原明文消息 M 这一过程是无法实现的，故 Hash 算法也称为单向不可逆算法。

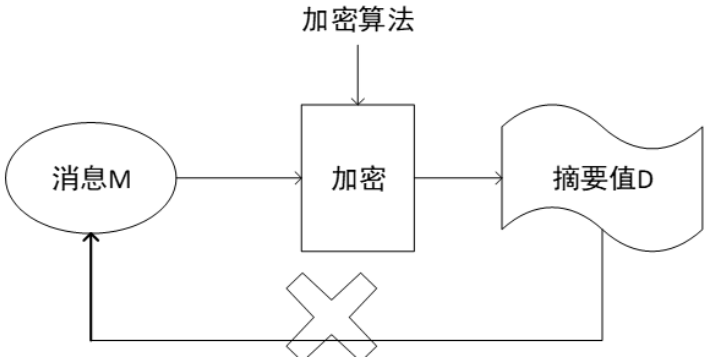

图 1.3 不可逆加密算法加密模型

无论是非对称密钥密码还是公钥密码，都存在身份冒充的问题，所以保证通信双方的身份真实性是一个重大的问题，Hash 算法的出现可以弥补这一缺点，利用 Hash 算法的单向不可逆特点，通信双方可以对信息进行签名，从而保证发送方身份的真实性，另外 Hash 算法还可以保证信息的完整性，防止信息在传输过程中丢失或者遭到攻击而被篡改，这些特点使得 Hash 算法在密码系统中被广泛使用。

1.3 国内外研究现状

1.3.1 MD5 算法国内外研究现状

MD5 算法是 MD 结构的典型代表，MD5 是罗恩·里维斯特(Ron Rivest)在上世纪 90 年代初设计的一系列加密哈希函数中的最后一个算法。它是一种著名的并且广泛使用的 128 位迭代哈希函数，用于各种应用程序，包括 SSL/TLS、IPSec 和许多其他加密协议。它还经常用于实现时间戳机制、承诺方案和用于在线软件、分布式文件系统和随机数生成的完整性检查应用程序。

近年来，随着人们对实时性、通信效率要求的不断提高，许多学者对 Hash 算法做了深入研究以求得到更好的性能，常见 MD5 和 SHA 算法优化方式主要包括了并行计算、循环展开、流水线等。在文献[24]中 Janaka Deepakumara Howard 等人第一次将 FPGA 硬件电路与 MD5 加密算法结合起来，进行硬件实现。自此以来大量研发人员研究 FPGA 并且发现 FPGA 硬件以极快速率利用数字集成电路去

实现 MD5 加密算法，同时可以验证其对应功能的正确性，从此 FPGA 硬件平台成为科研高校等研发机构用来实现 MD5 算法的首要选择，开创算法硬件实现的新时代。

文献[25]中 K. Jarvinen, M. Tommiska and J. Skytta 引进并行计算的概念，利用并行计算去实现 MD5 算法，在同一开发板上实现多个相同 MD5 模块，模块会在同一时间内工作，最后实现的 MD5 模块数量越多，得到的数据吞吐量最大。文中同时并行计算 10 组明文消息，得到其摘要值，以此方式增大数据吞吐量至 5.8Gbps；

文献[26]中 A. T. Hoang、K. Yamazaki、S. Oyanagi 等人继续以并行计算为基础，优化 MD5 算法实现的硬件架构，采用三级、四级流水线进行模块设计，实现数据吞吐量和硬件资源占用率的良好折中，可以较好地完成实验要求，为后续研究提供流水线设计思路。流水线设计目的是增加同时处理的数据总量，使算法能够同时处理多个消息块，算法可同时处理的数据量增大，提升数据吞吐量。根据 MD5 算法的结构，合理的流水线阶段数为 p= 1(迭代设计)、p=2、p=4、p=32 和 p=64(完全流水线设计)。如果使用了其他一些流水线级，例如 p = 8 或 p = 16，则主要提高了面积需求或延迟,而只有在相当大的面积需求下才能实现较快的性能。

文献[1]中 D.He 和 Z.Xue 利用多块 FPGA 去并行处理 MD5 加密算法，从而数据吞吐量随着 FPGA 数量或资源规模线性增加；文献[27]首先分析了 1、4、32 级多种流水线设计的吞吐性能，完成 32 级流水线设计，设计结果数据吞吐量达到了 32Gbps，但整个设计只对流水线结构进行扩展，电路结构非常复杂。除第一个 512bit 消息块在 64 个周期后出结果，其他均 2 个时钟周期内执行结果。

MD5 加密算法电路结构可再优化，提升数据吞吐量，提高数据处理速度；文献[28]由于 MD5 算法核心是 64 步循环迭代运算,故可以通过利用 full_pipeline 流水线设计去实现，从而整个设计实现 MD5 算法占用寄存器资源为 7253，吞吐量为 56.863Gbps；此设计占用大量硬件资源，并且数据吞吐量未得到大幅度提升；

文献[29]对数据流进行优化，提出多种方案实现 MD5 算法，通过对多种方案从硬件实现复杂度、最高时钟频率等方面进行分析后，将最佳方案落实到实践，完成硬件实现。最终实验结果表明数据吞吐量达到了 66.56Gbps，使用的硬件资源占 FPGA 所有硬件资源的 81.4%，实际应用中实现难度大；

文献[30]和文献[28]采用全流水架构，但此设计应用到拟态计算机平台实现 MD5 算法，并加入双端口 RAM 保障数据读写稳定进行，最终实验结果为占用硬件资源 12365，最高时钟频率为 241.6MHz，数据吞吐量为 123699Mbps，但它对硬件平台的要求过于苛刻，难以实现。故在信息化时代，如何快速且高效地实现 MD5 算法，保障信息安全传输显得格外重要。同时，采用流水线技术实现高的数据吞吐量对于算法实现具有重大意义。

1.3.2 SHA-2 算法国内外研究现状

SHA-2 系列算法是电子邮件、网上银行等领域必不可少的一种算法[31, 32]。在哈希函数领域通常使用碰撞性来衡量一个哈希算法的质量，碰撞性是指不同的输入产生相同输出的可能性。相比与 MD 系列算法，SHA-2 系列算法得到的摘要值位宽更大，所以碰撞性更小。对比两个相差不大的消息 M0 和 M1，若 M0 和 M1 仅一位数据不同，经过算法运算得到的结果相差甚远，这称为雪崩效应。由于这些特性，SHA-2 系列算法有取代 MD 系列算法的趋势。

中国科学院院士王小云教授得出了关于 SHA-1 算法以及 MD5 算法之间的碰撞实例，进一步地推动密码学的发展。之后王小云带领团队进行设计实现 SM3 杂凑算法[33]，随着越来越多的移动和无线设备被使用，安全问题也越来越受到关注。为了抵御软件攻击和获得高吞吐量，在安全芯片中通常将哈希算法集成为硬件 IP 核。然而移动和嵌入式产品是非常密集的，面积与成本和功耗有关。如何以低成本、低功耗的方式实现 SM3 的硬件模块具有重要的研究价值。2014 年郑佳敏等学生基于 SHA-256 算法原理以及嵌入式软件保护的理论[34]，研究讨论出方案，方案是在 Atmel Company 最新研发出的加密芯片基础上，进行上层的应用开发，此加密芯片型号为 ATSHA204A，完成 SHA-256 散列算法的软件实现。

文献[35]是 Lee H U 和 Lee S 等人对 SHA-1 散列算法进行优化实现，利用 Parallel Computing 即同时计算几组类似数据的方式进行优化，从而提高算法的数据吞吐量，提升数据处理速度。文献[36]也是利用并行计算技术进行 SHA-512 算法实现，提高设计的吞吐量。

文献[37]中 Michail H E 等人应用循环展开技术，提高算法实现性能。循环展开技术指的是对 SHA 安全散列函数的多轮哈希运算进行循环展开，即在一个时钟周期中展开多个 SHA 系列算法执行步骤，然后进行处理。由于循环展开技术可以减少部分的时钟延迟和寄存器延迟，从而减少了实现的总延迟，可以显著提高算法的吞吐量。

文献[38]也是利用这种技术。以 SHA-256 散列算法为例，SHA-256 散列算法需完成 4 次函数运算，每次函数运算为 16 轮迭代运算，因此正常情况下完成一次明文消息的 SHA-256 算法运算需要耗费 64 个时钟周期，这样才能得到运算结果即散列值，而循环展开技术就是指将这 64 轮循环迭代运算展开，将原本 64 轮哈希运算的轮数缩短变为 32 轮、16 轮、8 轮，如此一来 SHA-256 散列算法运算只需要花费 32、16、8 个 Clock 就能够得到最终结果，并将最终的散列值进行数据输出。因此循环展开技术就是通过缩短模块设计实现时的执行步骤，从而减少运算耗费的 Clock 来提高算法实现速度，优化算法实现性能。但在具体硬件实现过程中，循环展开技术是以牺牲算法实现的最高时钟频率为代价的，同时此技术会

延长设计的关键路径，故这种方式并不适合应用于对最高时钟频率和数据吞吐率要求更为严格的工艺制造中。

FPGA 由于可配置，功耗消耗少，以及并行计算能力强等特点，加之 FPGA 对 CPU 而言，FPGA 的并行计算极大提高运行效率，其效率是 CPU 的数十倍；而对比于 GPU，FPGA 性价比高且功耗更低。因此，哈希算法在 FPGA 上的硬件实现将成为未来长期研究的重点和热点。

文献[39]中 Athanasiou G S 等人利用预处理技术对输入的明文消息进行预处理，最后在 FPGA 上进行实现，用这样的方式来提高算法设计的效率。文献[35]也是利用这样的技术。其中预处理技术是指下一轮哈希运算时会用到的中间值，可以在上一轮哈希运算时计算得到，进行寄存器寄存以便进行下一轮哈希运算，而无需再对数据进行计算处理。更大的好处是由于这些中间值的计算是关键路径上的一部分内容，所以通过预处理技术可以缩短算法实现的关键路径，从而可以提高最终运行的最高时钟频率，但是预处理技术增加了附加的寄存器存放额外的中间值，所以对应的模块设计所占用的硬件资源也会有所提升，而这并不是设计人员所期待的。

文献[39, 40]都是利用流水线设计的技术去提高数据吞吐率。对于上述的其他技术而言，流水线技术提高数据吞吐率。文献[40]通过分析了不同级数的流水线设计(如 4 级、16 级、32 级)对 SHA-2 算法性能的影响，得出以下结论：当流水线设计应用的级数越高，最后得到模块实现的数据吞吐率会实现一种爆炸性增长趋势；但与此同时应用越多级的流水线设计，所需要的 slice 也越来越多。文献[39]Athanasiou G S 等人也实现了利用全流水线技术去实现 SHA-2 算法,但是硬件资源占用率几乎饱和，几乎占用了 FPGA 设备所有硬件资源，因此研究人员认为利用全流水线设计实现 SHA-2 算法硬件要求太高，硬件实现不太现实。

实际上在信息数据爆炸增长的时代，需要处理的数据量呈指数增长趋势，如何保证流水线技术在占用低的硬件资源的情况下实现较高数据吞吐量的 SHA-256 算法具有很重要的意义。如何能充分利用上述提及的优化方式如流水线设计的优势，同时满足尽量减少消耗 FPGA 上 Slice 资源，从而提高模块设计实现算法的效率，这是一个非常难攻克但是又是十分迫在眉睫的问题。

1.4 论文研究工作及章节安排

本文主要深入研究 MD5 信息摘要算法和 SHA-256 散列算法原理，为解决哈希算法实现存在执行时间长、数据吞吐量低、硬件资源占用率高这一问题，设计了一套硬件算法实现方案，最大限度的缩短处理的时钟周期消耗。同时，利用缩短单步实现的关键路径等优化方式，对算法设计的关键路径进行优化处理。本文提出的缩短关键路径的方式，一是优化加法器设计，利用三级加法器代替四级加

法器；二是插入中间寄存器，对输入数据进行预处理等方式优化算法单元数据通路，从而优化循环移位操作的方式。同时增添数据缓存设计，把串行数据流暂时缓存起来再进行流水线处理，加快数据获取速度。再者 MD5 和 SHA-256 算法采用 32 级流水线设计的模块架构设计完成硬件实现。

算法实现方式采用的是一个顶层模块链接多个子模块的架构设计，通过利用 FPGA 硬件实现 MD5 加密算法，完成模块实现的架构设计。对所设计模块进行测试与验证，利用 Quartus II 以及 QuestaSim 软件进行设计和仿真验证。采用的 FPGA 设备选择的是 Altera Cyclone-V 系列 FPGA，综合得到硬件资源利用率、速度、数据吞吐量方面的性能指标并进行性能分析，再以 MD5 加密算法硬件实现为基础实现 SHA-256 散列算法，利用相似的优化方式提升算法实现的数据吞吐量、提高最高时钟频率，减少占用的硬件资源。

本文章节安排如下：

第 1 章：介绍了本课题的研究背景及意义，接着简要介绍密码学中常用的密码算法，最后详细介绍了哈希算法包括 MD5 算法以及 SHA-256 散列算法的国内外研究现状。

第 2 章：简单介绍哈希函数，详细介绍了 MD5 算法和 SHA-256 算法的原理及算法结构，然后介绍算法的软硬件实现平台。

第 3 章：首先介绍 MD5 加密算法的硬件实现的架构设计及优化方案，主要由两部分组成，一部分是输入消息明文的预处理模块，另一部分是 MD5 哈希值计算模块。

第 4 章：介绍 MD5 算法的测试方案，以及 MD5 算法的仿真测试，对设计进行 FPGA 板级验证并对结果进行分析。

第 5 章：首先完成 SHA-256 算法模块架构设计。然后介绍架构中各个子模块的功能、原理框图、输入输出端口信号意义、算法实现时的表征标志信号等内容。最后介绍整个算法实现流程和 SHA-256 算法的硬件实现过程。

第 6 章：主要介绍 SHA-256 算法的功能仿真及结果分析，利用仿真结果与在线转换器得到的数据加密结果进行对比，验证模块设计的正确性。

总结与展望。简单回顾所完成的工作，对工作方案做出总结。并归纳出工作存在的提升点，明晰未来研究方向。

第2章 算法简介及原理

2.1 哈希函数简介

哈希函数在密码学中扮演着至关重要的角色，是一种复杂的加密函数，实质上是一种数学函数，它就是将大(或非常大)域的值映射到更小的范围，并将潜在的长消息减缩为“消息摘要”或“散列值”。利用数学关系式表示为Hn：$Z_2^{\geq 0} \rightarrow Z_2^n$，其中$Z_2^{\geq 0}$表示任意长度大于或者等于 0 的二进制字符串的输入数据，Z_2^n表示哈希函数运算后得到的 n 位长度的摘要值[36]。哈希函数是一种确定性过程，它接收一个有限长度的数据输入映射到一个较小的固定长度大小的数据输出[41, 42]，即散列值。要编码的数据常被称为消息，散列值也称消息摘要或摘要值。

哈希函数指出输入和散列值之间的独特关系，利用一个小得多的散列值取代大量信息(消息)的真实性，提出在大型数据库存储和快速检索信息方法。哈希函数也可用于实现关联记忆和纠错[43, 44]。随着公钥密码学的发展，哈希函数因能提供消息的真实性而受到越来越多的关注。一个“好的”哈希函数会产生一组均匀(和随机)分布在一定范围内的值。为了避免遭遇到攻击，用于加密的哈希函数应该具有以下几个属性：

1. 强抗碰撞性。对于不同消息可能映射到相同的摘要值的情况称为哈希碰撞现象。哈希函数不可能存在多个消息映射到同一个摘要值[45]。

2. 弱抗碰撞性。对于给定的明文消息 M0，找到另一个明文消息 M1，使得哈希函数运算得到的摘要值是相同的，即 D(M0)=D(M1)，这一场景几乎不可能出现。也就是说不同的明文消息得到的摘要值必定是不同的。

3. 单向不可逆性[46]。对于有限长度的消息 M，可以简易计算得到对应的摘要值 D(M)。但是如果已知摘要值 D(M)，根据算法逆运算得到相应的明文消息 M 十分困难。

4. 简便易算性[47]。对于任意给定的 M，哈希函数运算得到 D(M)相对容易。输入的明文消息的长度可以是任意的，在对应算法的长度范围内即可。输出的摘要值则可以是固定长度。

哈希函数称为密码学的“the duct tape”，如今哈希函数已发展到满足各个领域的各式各样的要求，包括加密、身份验证、承诺协议、密码管理、通信错误检测、入侵检测和同步等。更普遍地说，哈希函数几乎已经渗透在密码学的每个领域，甚至在其权限之外的许多领域。

数字签名作为哈希函数的一个主要应用。现实生活中有时需要验证文档来源、发送者身份、发送或签署文档的时间和日期、计算机或用户的身份等，这都可用

数字签名方式进行信息保护。利用 Digital Signature Algorithm(即数字签名算法或 DSA)和一组参数来计算，从而验证签名者的身份和数据的完整性。DSA 提供生成和验证签名的功能，一个数字签名方案 DS = (K, S, V)由三种算法组成，其中 K 代表的是随机生成密钥的一种算法，功能是返回一对具有相关性的密钥，分别为公钥 pk 和匹配的密钥 sk。数字签名即 DS 的生成是利用私钥完成的，验证数字签名则使用公钥[48]。如图 2.1 所示，展示了 DS 生成、验证模型。其中公钥与私钥是相关的，但又不同于私钥。每个用户都拥有一个私有密钥对和一个公钥对。一般来说，公钥设定为向公众公开的，但从不公开私钥，所以用户利用 pk 公开密钥对接收到的数字签名和消息进行考证，保障数据通信的真实性。但是数字签名的生成则只能通过拥有私钥的用户去生成。在签名生成过程中使用加密哈希函数来获得数据的压缩版本，称为消息摘要。

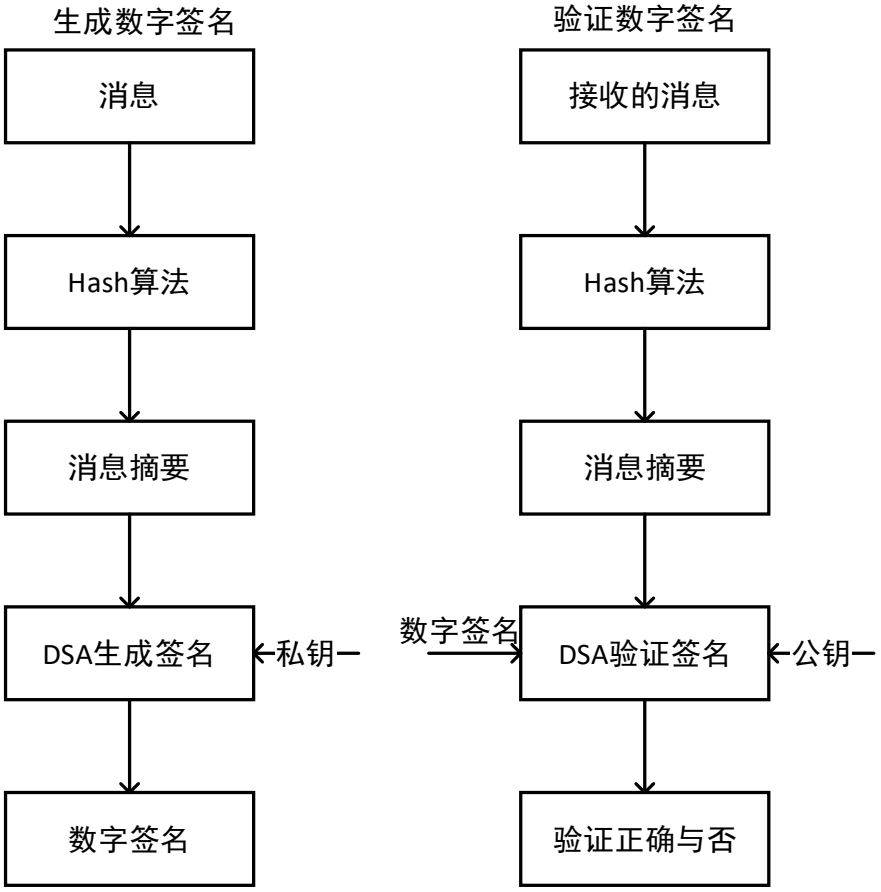

图 2.1 数字签名生成和验证模型

将消息摘要输入到 DSA 以生成数字签名。数字签名和需要传输的已署名数据(通常称为消息)一起发送给预期的验证者，预期的验证者利用发送方公开的公钥对签名和数据进行验证，验证身份保证数据真实性。验证过程中也必须使用相同的散列函数。

身份验证是用于保证安全通信的最重要的加密服务之一。身份验证一词在广义上用于涵盖许多安全目标。认证主要分为两种：

1. 身份验证：针对通信主体的认证，其目的是识别验证通信方身份，防止假冒等现象发生，常用数字签名的方法。

2. 消息验证：利用设置私钥的方式，解决通信双方在通过不安全的通道进行通信时存在的问题。比如有一个窃听者可以查看传输通道上的所有信息，并注入自己的信息。通信双方有一个窃听者不知道的共享密钥 sk，通信双方使用这个密钥进行数据通信，并提供一种方式可以验证彼此之间发送的消息。也就是说消息验证目标是阻止窃听者在消息传输或存储期间伪装成通信双方中的任一方。

2.2 MD5 算法原理

信息摘要算法 5 简称 MD5，是由麻省理工的罗恩·里维斯特提出的一种算法[49]，继承了 MD4 算法的优点，生成一个相对安全的版本。MD5 是当代密码学应用领域最广泛的一款哈希函数，大型文件必须以安全的方式压缩，然后在公钥公开的基础上利用私有(秘密)密钥将重要信息转换生成摘要值进行加密。MD5 算法接收的输入数据是有限长度的，输入的明文消息长度要满足小于 2^64-bit 的条件，映射到一个较小的固定长度大小的数据输出，即 128-bit 长度的哈希值。

MD5 算法以任意位的消息作为输入数据，输入数据通过一系列操作(也称预处理)后，得到以 512-bit 为一小单元的数据块[50]。明文消息可以由一个或者多个 512-bit 的数据块组成，然后将 512-bit 数据块进行分组，分成 16 个小分组。同时初始化链接变量 A、B、C、D，经过了 64 轮计算的循环体，循环计算完成后得到新的 A、B、C、D。接着判断是否再次进入循环，如果明文信息中预处理得到的所有 512-bit 数据块完成运算，则将这 A、B、C、D 级联成位宽为 128 的输出数据就是所求的 MD5 算法加密的摘要值。否则接收同一消息的其他 512-bit 数据块执行循环计算。现假设输入任意长度的明文消息，为得到对应的消息摘要值，处理过程包括以下步骤：

2.2.1 预处理步骤

1. 消息填充

消息填充完成的标志是填充完的数据加上 64-bit 的长度信息后可被 512 整除。需要注意的是：当消息长度对 512 取余后的值等于 448，执行消息填充操作；当消息长度除以 512 等于 447，执行消息填充操作，填充单个‘1’后完成填充操作。实质上填充数据由单个‘1’和必要数量的‘0’组成。

具体而言，如图 2.2 所示，对字符‘123’进行数据填充，其数据长度为 3*8=24bits，由于 24%512=24(%为取余符号)，并不等于 448。故填充单个‘1’和 423 个‘0’使得填充好的数据长度为 448，完成消息填充操作。

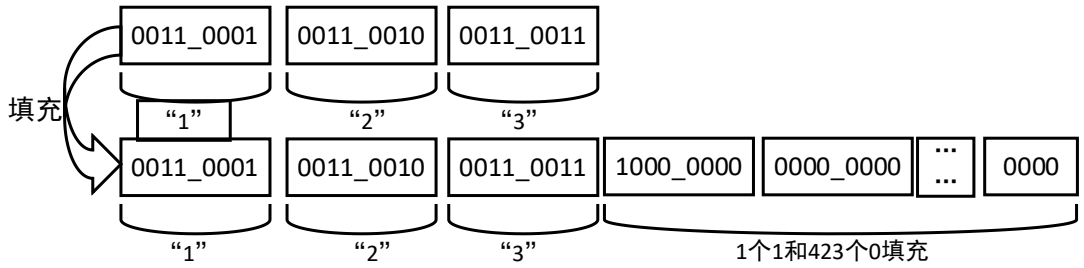

图 2.2 “123”消息填充示意图

2. 消息扩展

消息的原始长度用 64 位二进制表示连接到第 1 步骤的结果。如图 2.3 “ABC”消息扩展示意图所示，为第 1 步骤中“ABC”消息扩展情况。

具体来说，就是在完成消息填充的基础上，将 64-bit 位宽的长度信息以大端字节序放置到高地址中。其中消息填充完成后的数据以小端字节序存放在寄存器中，如图 2.4 所示。

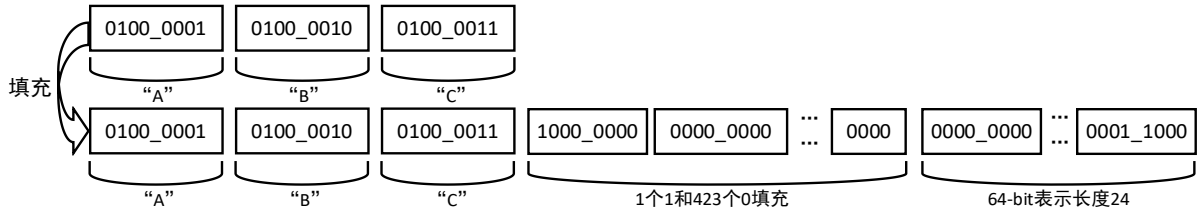

图 2.3 “ABC”消息扩展示意图

3. 消息分组

当第 1、2 步骤完成后，得到的数据均是以 512-bit 为一组的数据块。可按顺序 M[0]…M[15]表示一个 512 位的数，即将每 512-bit 数据块的消息划分成 16 组，每组 32-bit。

执行步骤：将输入的 512-bit 数据块划分成 16 个子分组，每个子分组是 32-bit 位宽的数据。在 32-bit 位宽的子分组中，以每个字节位宽进行数据的小端排序，小端字节序就是大端的相反过程。假设需要映射的明文消息为“ABC”字符串，其中一个字符 A 或 B 或 C 为一个 8-bit 位宽的字符数据，算法运算时存放于寄存器的数据是格式为“CBA”。因此 MD5 算法运算将字符串每 4 个字符分成一块，即以 32-bit 为一个子分组。字符串“ABC”用十六进制表示为“414243”，则用小端字节序表示为“43_4241”,将字符串进行填充扩展后如图 2.4 所示。

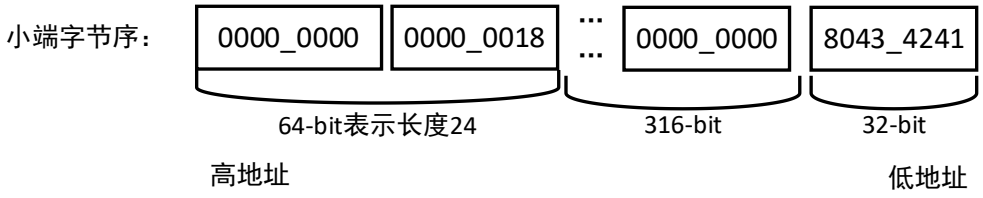

图 2.4 “ABC”预处理后的数据

2.2.2 循环迭代计算

1. 首先定义每轮运算的逻辑函数。假设 Mj 表示输入数据 512-bit 数据块第 j 个子分组[51](j 的定义域为[0,15])。ti 是 4294967296*abs(sin(i))的整数部分,其中 4294967296 等于 2 的 32 次方[52]，i 的定义域为[1,64]。

FF (a, b, c, d, Mj, s, ti)　表示 a = b + ((a + F(b, c, d) + Mj + ti) <<< s)

GG (a, b, c, d, Mj, s,ti)　表示 a = b + ((a + G(b, c, d) + Mj + ti) <<< s)

HH (a, b, c, d, Mj, s,ti)　表示 a = b + ((a + H(b, c, d) + Mj + ti) <<< s)

II (a, b, c, d, Mj, s, ti)　表示 a = b + ((a + I(b, c, d) + Mj + ti) <<< s)

2. 初始化 32-bit 初始值。由于生成的摘要值为 128-bit 位宽的数据，首先初始化四个初始值 a、b、c、d 作为循环迭代运算的初始值，分别为：a=0x01234567、b=0x89abcdef、c=0xfedcba98、d=0x76543210[53]。再者将 a、b、c、d 的值先寄存到 A、B、C、D 寄存器中，即 A=a；B=b；C=c；D=d。

3. 主要循环。循环块的循环次数为明文消息预处理完成后的 512-bit 数据块的个数。其中循环体为：

$$T = a + f_j(b,c,d) + M_j + t_i$$
$$a = d$$
$$d = c$$
$$c = b$$
$$b = b + T <<< s \tag{2.1}$$

其中 i、j 从 0 到 63 依次循环。MD5 有 4 轮，每轮包含 16 个压缩步骤，每轮在所有步骤中使用相同的逻辑函数f_j，如下:

$$F(x,y,z) = (xy)|(\sim x\&z) \tag{2.2}$$
$$G(x,y,z) = (xz)|(\sim z\&y) \tag{2.3}$$
$$H(x,y,z) = x\hat{}y\hat{}z \tag{2.4}$$
$$I(x,y,z) = y\hat{}(x|\sim z) \tag{2.5}$$

当 MD5 算法进行运算，完成四轮循环且每轮操作非常相似。其中每轮操作就是计算对应轮的逻辑函数的值且每轮的循环次数为 16 次后进入下一步。

2.2.3 摘要值输出

如果同一明文消息还有待处理的 512-bit 数据块，则更新链接变量，即 a=A+a；b=B+b；c=C+c；d=D+d 的值将此作为下一分组数据运算的链接变量，返回到步骤 2 继续进行循环计算；如果此时是最后一组 512-bit 数据块，则最终输出的 A=A+a；B=B+b；C=C+c；D=D+d，A 是高位，D 为低位，ABCD 级联后形成 128 位输出

结果即 MD5 算法计算得出的消息的摘要值，这与 MD5 算法在线转换器得到的结果顺序相反。

2.3 SHA-256 算法原理

NIST 研究哈希函数标准，在 1993 年，NIST 制定了安全哈希标准 SHA[54]。然而，在文章发表后不久，该算法就因一个未披露的“重大缺陷”而被撤回，被一个名为 SHA-1 的修订版本所取代。SHA-1 算法应用领域广泛，主要涉及各种信息安全方案，如安全外壳(Secure Shell, SSH)和完美隐私(Pretty Good Privacy, PGP)。

SHA-2 是对以前的 SHA-1 标准的更新[55]，SHA-2 算法包括 SHA-256、SHA-384、SHA-512 三种，SHA-2 中的哈希函数是根据生成摘要值的位数来命名的，例如 SHA-256 的摘要值为 256 位。三者的区别如表 2.1 所示。由表可以知，区别 SHA-256 和其他两种算法的主要依据有：接收输入的明文消息长度、生成的摘要值的位宽。SHA-256 算法接收长度不足 2^64 位的任意消息，计算过程中消息分割成 32-bit 位宽的子分组，加之 SHA-256 算法生成的摘要值为 256 比特位位宽数据，初始化四组链接变量作为初始值，除此之外其他操作非常相似。综合以上原因，本论文主要针对 SHA-256 算法进行硬件实现。

表 2.1 SHA-2 主要类型算法异同点

SHA-2 类型	消息长度	分组长度	计算字长	摘要长度
SHA-256	<2^64 位	512	32	256
SHA-384	<2^128 位	1024	64	384
SHA-512	<2^128 位	1024	64	512

SHA256 算法由于映射压缩生成的摘要值位宽长度为 256-bit，即 32 个字节，称为 SHA256。SHA-256 接收长度不足 2^64 位的任意消息，产生一个最终 256 位的摘要消息，它依赖于输入消息，由多个 512 位的数据块组成。这个数据块是以大端字节序形式进行输入的，将输入划分为 16 个以 32 位为一个子分组的数据进行 SHA-256 函数的 64 个循环。每次计算完一个 512-bit 块则将中间哈希值重新压缩到循环中继续进行下一次 64 个循环的计算，直到输入消息的最后一个 512-bit 块计算完成后输出 256bit 的摘要值。

SHA-256 算法和 MD5 算法执行步骤相同，由图 2.5 所示的四个阶段组成。为了处理单个数据块，要执行 64 轮消息压缩。当输入的明文消息是由多个 512-bit 数据块组成时，则循环体要等到所有的数据块处理完毕后才输出最终摘要值。如图 2.5 为 SHA-512 散列算法所涉及阶段的高级框图。

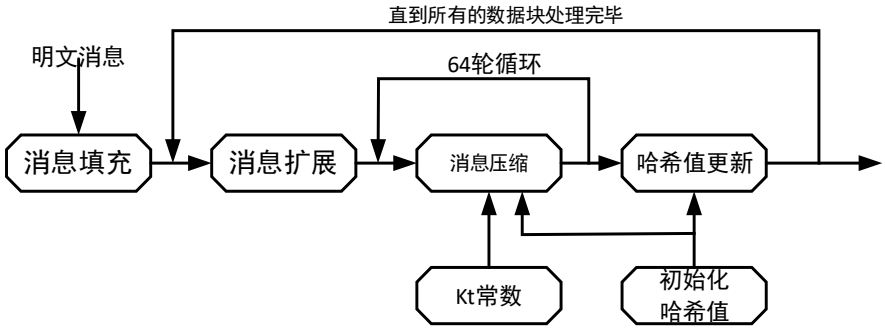

图 2.5 SHA-256 散列算法所涉及阶段的高级框图

2.3.1 消息填充

对 SHA-256 算法运算时首先要将待运算的明文消息进行填充，以便数据计算时数据长度是 512 位的倍数[56]。填充 n 比特位的明文消息时，在消息的末尾添加一个 1 位“1”，然后加“0”，直到消息长度对 512 取余等于 448，并将代表消息长度的 n 比特位以 64-bit 的形式被附加到填充的结果中。因此，明文消息处理后得到的是 512 位宽的倍数，且在最后一个 512-bit 数据块中的 64-bit 中包含消息明文的实际长度信息。消息用 M(i)表示，M(i)个消息块分别传递给消息扩展器。

具体操作：首先明文消息末尾添加单个二进制数“1”，再添加(448-Length-1)个“0”，最后附加扩展 64-bit 数据位宽的代表长度信息的数据[57]。若要输入消息数据“123”，已知字符用 ASCII 码表示，一个 ASCII 码用 8 个比特位表示，故字符串“123”长度为 3*8=24。将 24 对 512 取余的 24，则填充 1 个二进制“1”和 423 个二进制“0”至整个填充长度为 448，最后填充 64-bit 位的长度信息，表示输入的消息长度为 24。如图 2.6 所示，完成消息附加填充位的操作。然后继续添加 64 比特位代表字符串长度信息的十进制“24”，完成消息附加长度的操作。

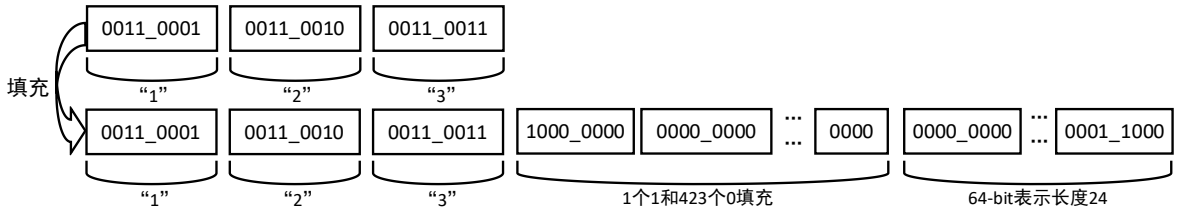

图 2.6 “123”数据填充示意图

将填充好的数据按大端字节序排序，换成十六进制，如图 2.7 所示

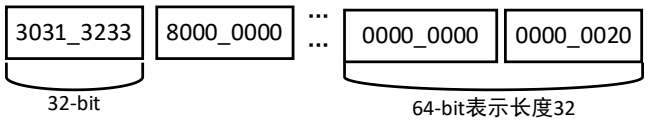

图 2.7“123”数据填充排序图

2.3.2 消息扩展

SHA256 算法对数据的处理和 MD5 算法类似，均是以 512-bit 为单个数据块作为数据处理单元，每次读入第 i 个 512-bit 的数据块后，算法内部会对 512-bit 数据块进行处理，即将数据划分成由 16 个 32-bit 位宽数据的子分组，并在四轮函数计算中均以 32-bit 子分组进行函数运算。输入的 16*32-bit 的数据通过式(2.6)转变扩展成 64*32-bit 的数据，用于后续循环迭代计算。

其中假设第一个 32-bit 的数据子分组为M_0^j，则依次往后第二个为M_1^j，直到最后一个M_{15}^j。随后利用式(2.6)将 16*32bits 的数据块进行扩展，最终实现成 64*32bits 的数据块,并将扩展后的数据块存放在W_t表达式中。即当0 ≤t≤ 15时，$W_t = M_t^j$；当 16 ≤t≤ 63时，

$$W_t = \sigma_1^{(256)}(W_{t-2}) + W_{t-7} + \sigma_0^{(256)}(W_{t-15}) + W_{t-16} \tag{2.6}$$

其中需要用到置换函数：

$$\sigma_0^{(256)}(X) = ROTR\text{^}7(X) \oplus ROTR\text{^}18(X) \oplus SHR\text{^}3(X) \tag{2.7}$$

$$\sigma_1^{(256)}(X) = ROTR\text{^}17(X) \oplus ROTR\text{^}19(X) \oplus SHR\text{^}10(X) \tag{2.8}$$

其中 X 表示 32-bit 位宽的子分组数据，ROTR^L(X)表示 X 数据进行循环右移，移动位数为 L，SHR^L(X)表示 X<<L 即 X 向右移动 L 位，低位利用二进制“0”填充。已知K_t表示 64 个常数，利用 64 位随机串数据破坏数据规则，保证随机性、任意性，这些值以从左到右、从上到下的方式为导向节省空间。该算法在消息压缩阶段使用。

2.3.3 消息压缩

SHA-256算法的消息压缩，就是循环迭代计算，是SHA-256算法的核心部分。SHA-256 包括 64 轮运算。主要的步骤为：

1. 初始化链接变量

对 a、b、c、d、e、f、g、h[58]进行初始化，进行数据赋值，即

a=32’h6a09e667；*b*=32’hbb67ae85；

c=32’h3c6ef372；*d*=32’ha54ff53a；

e=32’h510e527f；*f*=32’h9b05688c；

g=32'hlf83d9ab；h=32'h5be0cd19。

将 a、b、c、d、e、f、g、h 暂存在寄存器 A、B、C、D、E、F、G、H，即 A=a；B=b；C=c；D=d；E=e；F=f；G=g；H=h。

2. 循环迭代计算

执行 64 次循环迭代计算，循环体为：

$$T_1 = h + \Sigma_1^{(256)}(e) + Ch(e,f,g) + K_t^{(256)} + W_t$$
$$T_2 = Maj(a,b,c) + \Sigma_0^{(256)}(a)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2 \tag{2.9}$$

式(2.9)中

$$Ch(x,y,z) = (x\&y) \oplus (\bar{x}\&z) \tag{2.10}$$
$$Maj(a,b,c) = (x\&y) \oplus (x\&z) \oplus (y\&z) \tag{2.11}$$
$$\Sigma_0^{(256)}(x) = \text{ROTR\^{}2(X)} \oplus \text{ROTR\^{}13(X)} \oplus \text{POTR\^{}22(X)} \tag{2.12}$$
$$\Sigma_1^{(256)}(x) = \text{ROTR\^{}6(X)} \oplus \text{ROTR\^{}11(X)} \oplus \text{POTR\^{}25(X)} \tag{2.13}$$

其单步循环操作示意图如下：

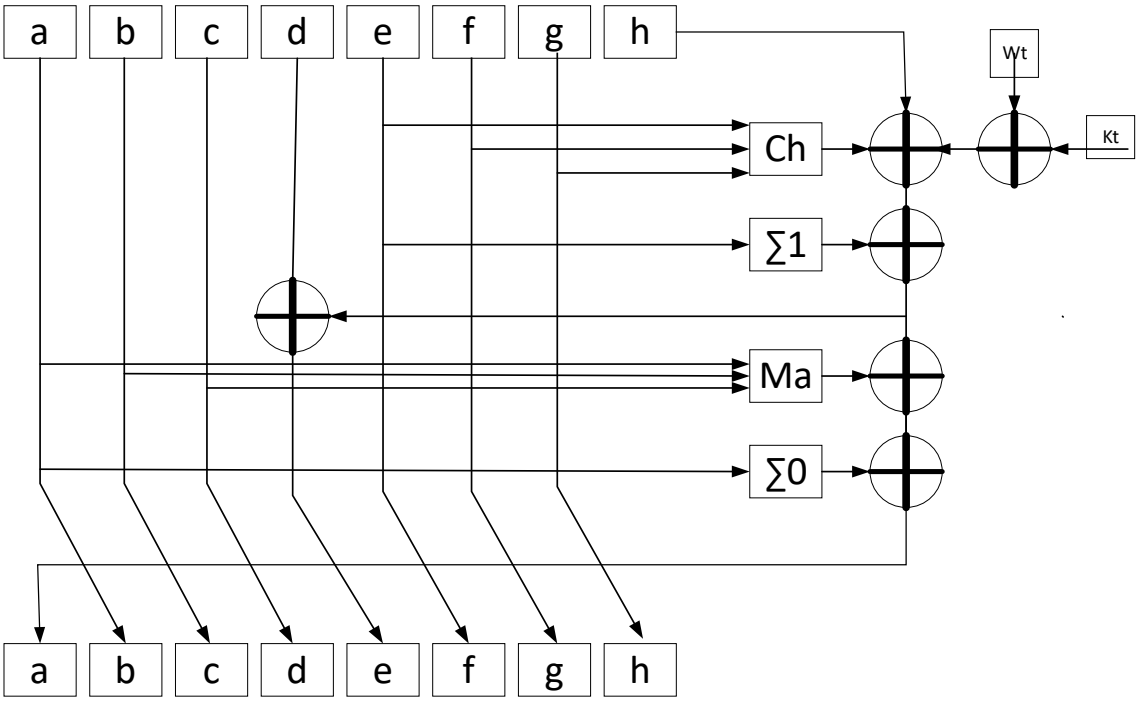

图 2.8 SHA-256 单步循环迭代运算步骤图

2.3.4 摘要值输出

如果同一明文消息还有 512-bit 数据块待处理，则更新链接变量，将哈希运算结果叠加。将完成 64 次循环迭代计算的结果和 A……H 寄存器中的值进行对应叠加，即 a=A+a；b=B+b；c=C+c；d=D+d；e=E+e；f=F+f；g=G+g；h=H+h 值将此作为下一分组数据运算的链接变量，返回到步骤 2 继续进行循环计算；

如果此时是最后一组 512-bit 数据块，则输出的 A=A+a；B=B+b；C=C+c；D=D+d；E=E+e；F=F+f；G=G+g；H=H+h；得到的 A_B_C_D_E_F_G_H 组成 256 位输出结果即 SHA-256 算法计算得出的消息的摘要值。

2.4 软硬件平台简介

2.4.1 软件平台简介

Quartus II 是目前世界上使用最多的 FPGA 开发的 EDA 软件之一。用户可利用此平台实现整个 FPGA 设计，其界面友好，操作便捷，可以完成设计输入(包括 verilog/vhdl/原理图)、电路综合、时序约束和自动化的布局布线，还可以进行仿真，其内嵌的 signaltap 可以进行硬件测试，极大减小了 FPGA 设计开发周期。图 2.9 是 Quartus II 软件用户界面图，开发者可以按照 verilog 编写、功能仿真、设计综合、时序约束、布局布线、生成比特流文件[59]、代码下载、板级调试的步骤进行设计开发。

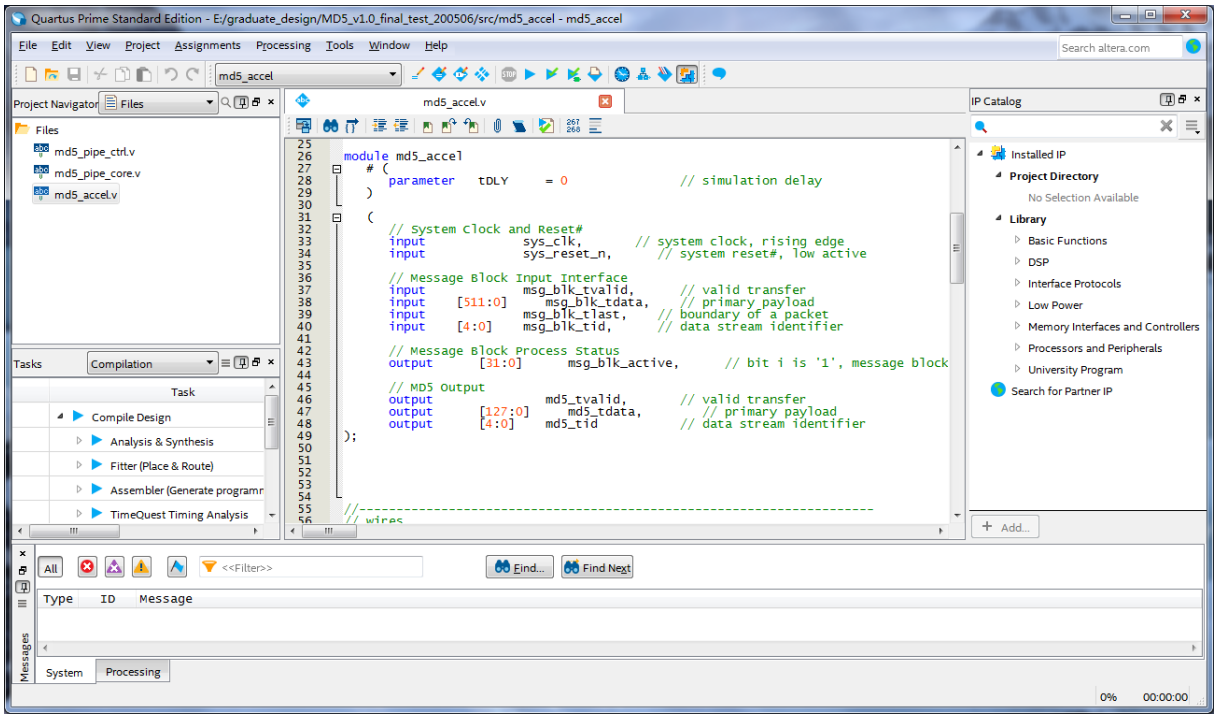

图 2.9 Quartus II 界面图

由于 Quartus II 软件自带的仿真工具功能比较单一，所以本论文选用第三方仿真工具 QuestSim 软件，如图 2.10 所示为 QuestaSim 通用的用户界面。软件既可以作为独立工具使用，也可以与 Altera Quartus 或 Xilinx Vivado 结合使用。实现仿真验证既可通过测试平台编写脚本自动完成，也可以进行点击操作手动完成。QuestaSim 基于对 System Verilog、System C、System C 语言进行验证，也可以搭建 UVM 测试验证平台完成验证模型 model 设计进行验证。这可与 Verilog HDL、VHDL、System Verilog 几种语言任意搭配去进行仿真验证。本文验证语言是基于 System Verilog。

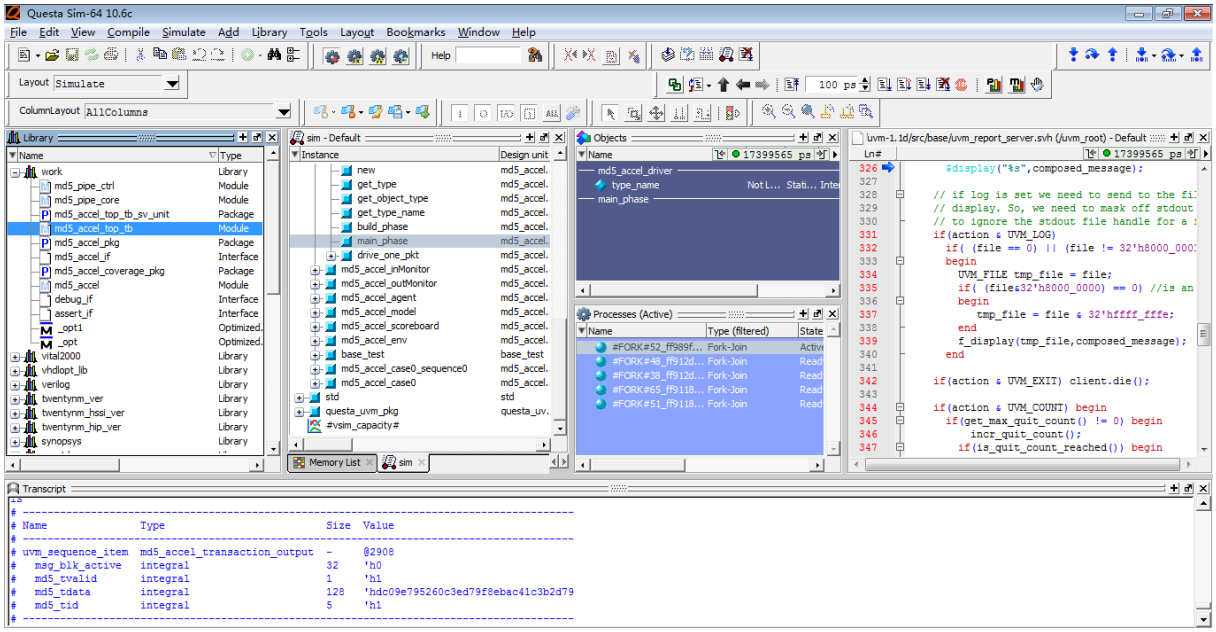

图 2.10 QuestaSim 用户界面

Questa Sim 是一款可显示覆盖率信息的工具软件，可将电路网络的抽象层次提高至 RTL 级，从而提供最好的性能与最有效的分析和调试。由于 QuestaSim 可通过单独文件描述硬件电路结构，所以对现有 RTL 流程无任何影响，RTL 代码也无需任何修改。软件主要特点：高性能的验证环境、高级验证方法学、测试自动化与 CDV、验证管理、基于断言的验证、具有功耗意识的验证、集成多语言调试、可扩展验证。

2.4.2 硬件平台简介

Altera FPGA 产品有一种是适用低成本开发的，容量中等的中低端系列产品，例如 Cyclone IV、Cyclone V 等；另外一种是高端系列产品，主要针对应用于高性能进行开发的，具有容量大的优点，如 Stratix、Stratix II 等系列产品，表 2.2 和表 2.3 分别是 Cyclone IV 系列和 Stratix II 系列 FPGA 资源表。在本论文中的算法实现，考虑到成本和所需要的资源占用情况，而且采用的流水线设计，需要较多

的寄存器单元，综合考虑后选用了 Cyclone IV 系列的 EP2S30 型号 FPGA，该型号 FPGA 拥有 33880 个 LE 单元，13552 个 ALM 单元，可以满足本设计的要求，从最终的实现结果来看选择该型号 FPGA 也是合适的。

表 2.2 Cyclone IV 资源表

产品名称	逻辑元素（LE）	数据信号处理（DSP）区块	最大嵌入式内存	封装选项
Cyclone IV EP4CE6 FPGA	6000	15	270Kb	E144，U256 F256
Cyclone IV EP4CE10 FPGA	10000	23	414Kb	E144，U256 F256
Cyclone IV EP4CE22 FPGA	22000	66	594Kb	E144，U256 F256
Cyclone IV EP4CE55 FPGA	56000	154	2.34Mb	U484，F484 F780
Cyclone IV EP4CE115 FPGA	114000	266	3.888Mb	F484，F780
Cyclone IV EP4CGX15 FPGA	14000	0	540Kb	F169

表 2.3 Stratix II 资源表

器件	ALM	等价 LE	M512 模块	存储器总容量	DSP 模块	18*18 乘法器
EP2S15	6240	15600	104	419328	12	48
EP2S30	13552	33880	202	1369728	16	64
EP2S60	24176	60440	319	2544192	36	144
EP2S90	36384	90960	488	4520448	48	192
EP2S130	53016	132540	699	6747840	63	252
EP2S180	71760	179400	930	9383040	96	384

2.5 本章小结

本章首先介绍哈希函数和主要应用——数字签名，再对 MD5 算法进行详细介绍，重点阐述 MD5 算法定义、具体实现原理，简述 SHA-2 算法发展历程，对 SHA-256 散列算法实现原理进行阐述，为后续算法硬件实现提供理论基础，另外介绍了本设计使用到的软硬件平台，对硬件平台的选取做了分析。

第3章 MD5 算法模块设计

3.1 输入数据预处理模块设计

MD5 算法的 FPGA 设计框图如图 3.1 所示。接收输入的明文消息，并对消息进行预处理操作，这一过程在数据预处理模块完成，预处理完成后将预处理完的数据输送到 MD5 哈希值计算模块完成 MD5 算法运算，等到所有的数据块运算完毕后输出哈希值，完成整个 MD5 算法模块功能设计。

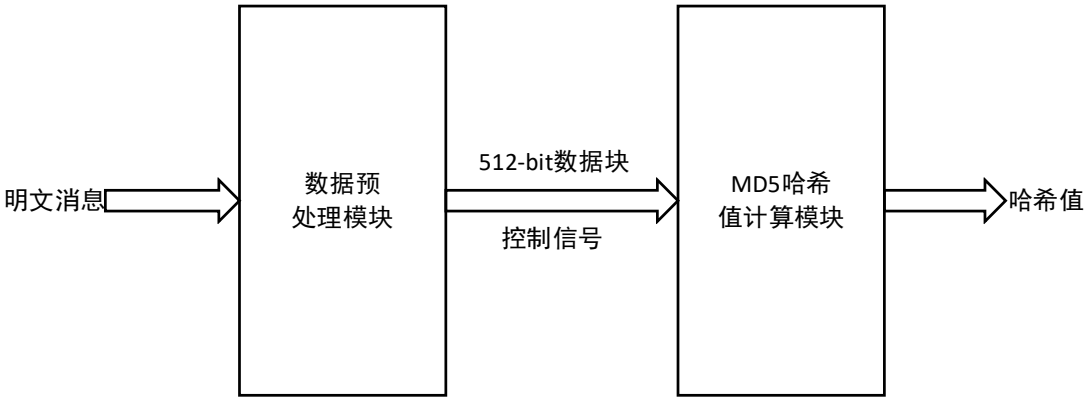

图 3.1 MD5 算法模块设计框图

数据预处理模块主要功能是接收明文消息，完成预处理后输出 512-bit 数据块以及相应的控制信号给 MD5 哈希值计算模块。从 MD5 算法原理可知，MD5 每次处理的分组长度为 512-bit，但是在实际应用中不可能数据长度都为 512-bit 或 512-bit 的倍数，所以需要根据相应的填充规则进行数据填充，完成填充后按 512-bit 为一组，将整个消息数据进行分组输入，在数据有效信号拉高的同一时钟周期输入相应的控制信号。

根据明文消息的长度，可将其对 512 求余，判断是否等于 448，如果不等于 448 则填充单个“1”和必要数量个“0”，直至满足对 512 求余等于 448 才停止对信息进行填充。接着将代表消息长度的信息以 64-bit 的形式被附加到填充的结果中。上述处理完成后，预处理好的数据长度符合预期格式，即预处理后数据由 N*512-bit+448-bit+64-bit 组成，即((N+1)*512)-bit。数据预处理模块的输入输出框图如图 3.2 所示。

该模块信号具体如表 3.1 所示，其中 msg_rdata 是上层发送过来的明文数据，msg_rdata_valid 表示数据的有效信号，数据以 32 比特为单元，选择此位宽的原因是因为一般数据都以 word 字为数据单元，以 32bit 为数据也可以提高数据的处理速度。

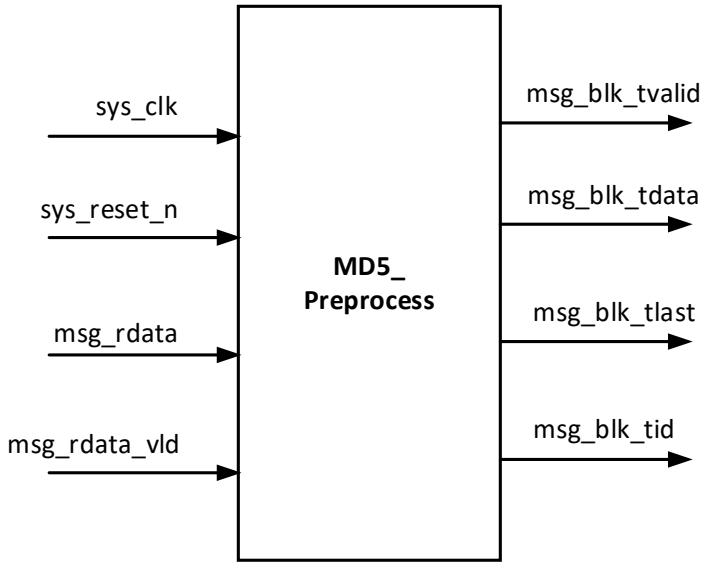

图 3.2 预处理模块输入输出框图

表 3.1 预处理模块端口信号

名称	位宽	方向	描述
sys_clk	1	I	时钟信号
sys_reset_n	1	I	复位信号
msg_rdata	32	I	接收数据信号
msg_rdata_vld	1	I	接收数据有效信号
msg_blk_tdata	512	O	输入的预处理完成的消息分组数据即消息数据块
msg_blk_tlast	1	O	“1”表示消息的最后一个分组数据块；“0”表示此时的 msg_blk_tdata 数据块并非是消息的最后一个分组数据块
msg_blk_tid	5	O	消息分组的通道编号
msg_blk_tvalid	1	O	“1”表示消息分组数据 msg_blk_tdata 有效；“0”表示消息分组数据 msg_blk_tdata 无效

举例计算“123”的 MD5 哈希值，预处理过程为：输入的明文消息是 313233 (此为十六进制表示)，长度为 24。长度对 512 求余为 24，进行数据填充，往后填充 1 比特位的二进制“1”和 423 个比特位的二进制“0”，如图 3.3 所示。

由于存储方式为小端字节序，若以十六进制表示，需将 512-bit 数据块进行数据划分成 16 个 32-bit 数据子分组，如图 3.4 所示。

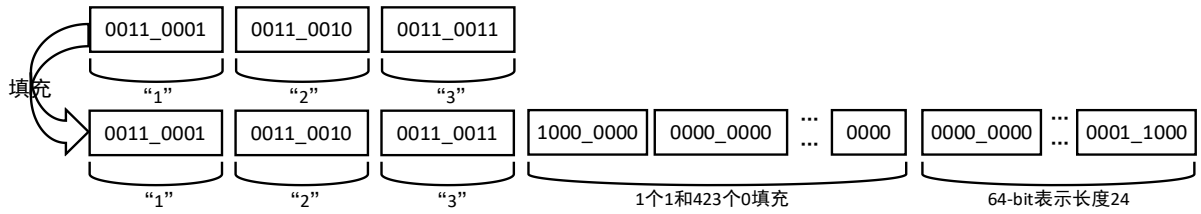

图 3.3 明文数据填充示意图

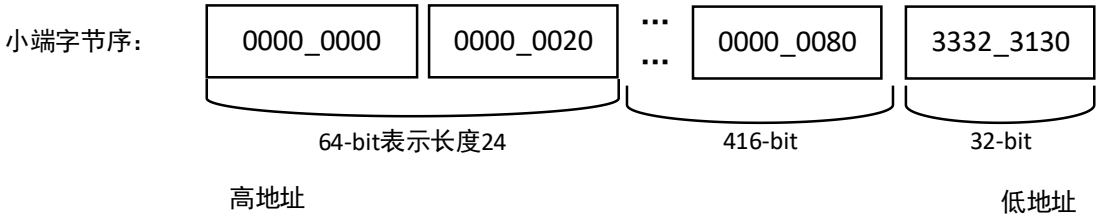

图 3.4 明文预处理完成数据

预处理操作完成后，将 512-bit 数据从 msg_blk_tdata 输出端口输出，同时拉高数据有效信号 msg_blk_tvalid。在同一时钟周期下将数据分配到任一空闲通道中，如果通道 10 空闲，则输出端口占用通道数 msg_blk_tid 为 10；因为待运算的明文消息预处理后只有一组 512-bit 数据，此时 msg_blk_tdata 输出端口输出的数据为明文消息的最后一组子分组数据，则输出端口 msg_blk_tlast 拉高。

3.2 MD5 算法优化方案

MD5 算法硬件实现时，可运行的最高时钟频率、数据吞吐量、硬件资源占用率是三个主要考虑的重要指标。为解决硬件实现 MD5 算法存在的可运行的最高时钟频率低、执行时间长、数据吞吐量小、硬件资源占用率高的问题，MD5 哈希值计算模块作为 MD5 算法硬件实现核心部分，优化算法实现主要针对 MD5 哈希值计算模块进行优化。MD5 算法硬件实现有两种模式，一是循环迭代模式，二是流水线设计模式。具体优化设计从以下两方面进行：

3.2.1 缩短关键路径

循环迭代模式优化，利用优化加法器设计、优化循环移位操作方式缩短 MD5 算法单步运算的关键路径，节省单步运算时间，提高运算执行速度。首先将逻辑函数 FF、GG、HH、II 的数据计算流程优化设计如图 3.5 所示。

1. 优化加法器设计

利用一个加法器替代实现两个加法器的操作，四级加法器操作优化成三级加法器。以单步函数 FF 计算为例：算法的第 1、2 步在创建激励时将任意位宽的消息拓展成所需的数组，每个数组包括数个 512-bit 数据分组。每组 512-bit 输入数

据划分为 16 个子分组数据Mj视为常数，ti表示$4294967296 * abs(\sin(i))$的整数部分视为已知常数[16]。

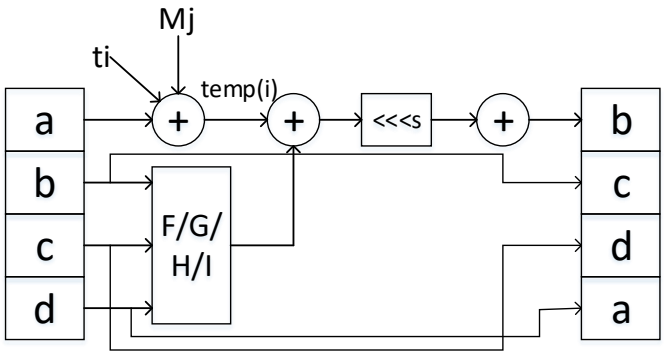

图 3.5 MD5 算法的逻辑函数计算流程图

从原理中分析得$B_{j+1} = B_j + (f + A_j + T_{j+1} + W_{j+1}) \lll S_{j+1}$,$A_i$ 表示第 j 轮寄存器 D 的值，而 D_j 的值是第 j-1 轮中 C_{j-1}，即 A_j 表示为第 j-1 轮中 C_{j-1} 的值，可对于$0 \le j \le 12$时先预处理两组中间值，即

$$temp(j) = c(j) + M(j+2) + t(i+2) \tag{3.1}$$

$$temp(j+1) = b(j) + M(j+3) + t(i+3) \tag{3.2}$$

当对于 j=13 时，则预处理的两组中间值为：

$$temp(j) = c(j) + M((5*16+1)\%16) + t(i+2) \tag{3.3}$$

$$temp(j+1) = b(j) + M((5*17+1)\%16) + t(i+3) \tag{3.4}$$

以此类推，利用流水线设计实现一个时钟周期内输出逻辑函数运算的结果并对应地输出给紧接的两组 FF 数据。其余轮的运算与此类似，得到的a,b,c,d的值向右轮换输出给到下一轮计算使用。

2. 优化循环移位操作。

循环展开技术表示在一个时钟周期中展开几个 MD5 运算步骤，再完成数据处理。循环展开技术减少部分的时钟延迟和寄存器延迟，从而减少了实现的总延迟，提高算法的吞吐量。

64 步循环步骤中分成 F、G、H、I 四轮操作，且每轮中循环移位的位数 s 按固定的 4 个数字进行循环移位，一共循环 4 次，各占 16 步操作。其中 F 轮操作中循环移位的位数 s 为 7、12、17、22；G 轮操作中循环移位的位数 s 为 5、9、14、10；H 轮操作中循环移位的位数 s 为 4、11、16、23；I 轮操作中循环移位的位数 s 为 6、10、15、21；由于位数固定，可取消 32-bit 数据进行循环移位的操作，替代的是直接将移位的结果作为加法器的加数相加。

通过以上两种方式缩短单步运算的关键路径,有效提高 MD5 算法运算速度，缩短数据处理时间。

3.2.2 流水线设计

采用流水线设计实现 64 步循环运算并行化，一个时钟周期内实现 2 步运算，采用 32 个数据处理通道可同时进行 MD5 运算，提高数据处理速度，提高硬件可运行的最高时钟频率，增大数据吞吐量。

流水线设计实质是“拆分”，如图 3.6 所示。实现既定功能需执行接收源触发器 D1 的输入端数据到目的触发器 D2 输出端输出。已知源触发器 D1 的输出端到目的触发器 D2 输入端经过一系列组合逻辑从而可以实现设定功能。故将这一系列组合逻辑拆分成若干步进行实现，从而完成流水线设计，使得每个周期均实现功能，输出想要结果。

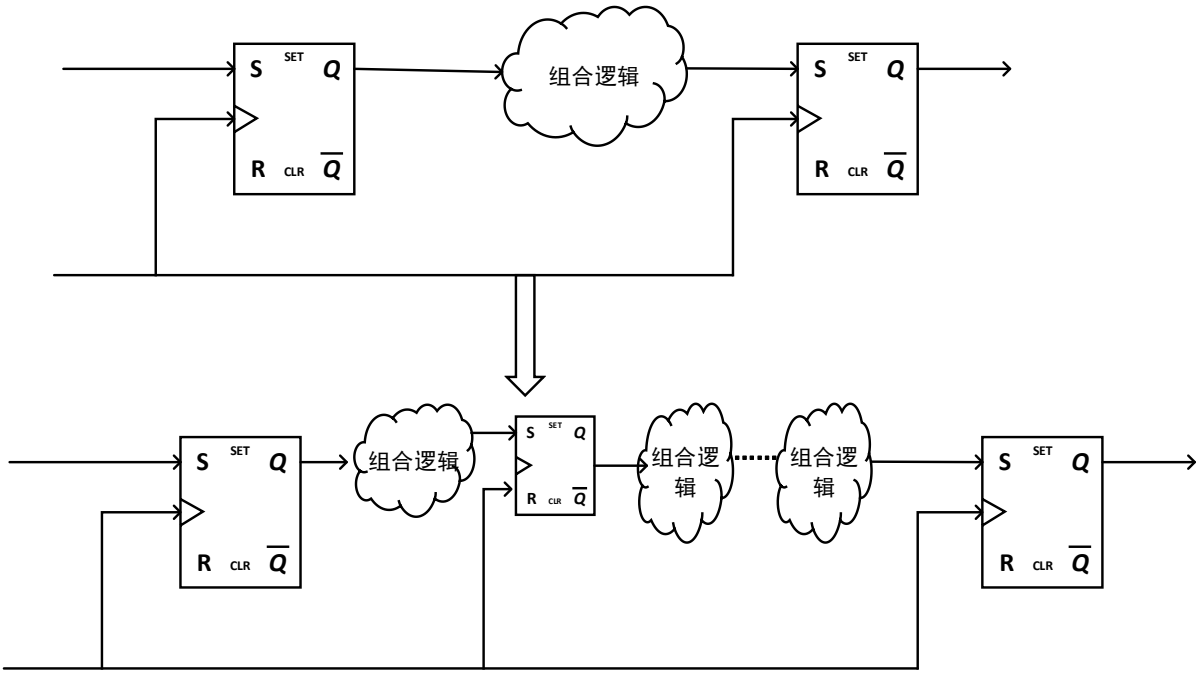

图 3.6 流水线“拆分”示意图

流水线结构主要为实现周期性地执行相同的操作任务。多级流水线表示的是在同一时钟周期 T 内，实现多任务执行。如果一个电路功能是通过四步组合逻辑实现，它需要 4 个时钟周期 T。此段功能可采用流水线设计，输入数据随流水线完成步骤实现情况如图 3.7 所示。0T 时接收数据“输入 In1”，下一 T 即 1T 时，接收新的输入数据“输入 In2”同时将正在处理的“输入 In1”输送到下一级流水；直到最后一级流水线完成整个数据处理。由此可知“输入 In1”从 0T 经过 4 个时钟周期到 3T 时才完成数据“输入 In1”的输出，输出结果命名为“输出 Out1”，紧接着每经过一个时钟周期就会有“输出 Out2”、“输出 Out3”、……、“输出 Outn”等数据输出。

综上，若流水线设计划分成 N 级流水，则第一个输入数据“输入 In1”要经过 N 个时钟周期 T 才会输出数据“输出 Out1”，每过 1T 输出一数据“输出 Outn”。故采用流水线设计对算法实现进行优化，缩短等量数据处理时间，提高数据吞吐量，

若采用 N 级流水线能够提高近 N 倍的处理速度，提高算法运算速度，以便信息快速算法加密。假如图 3.7 不采用四级流水线进行数据传递，完成 400 个数据输入输出要花费 1600T，总体数据处理速度慢、耗时长。如果采用四级流水线设计完成 400 个数据输入输出仅需 404T，提高近四倍的处理速度。

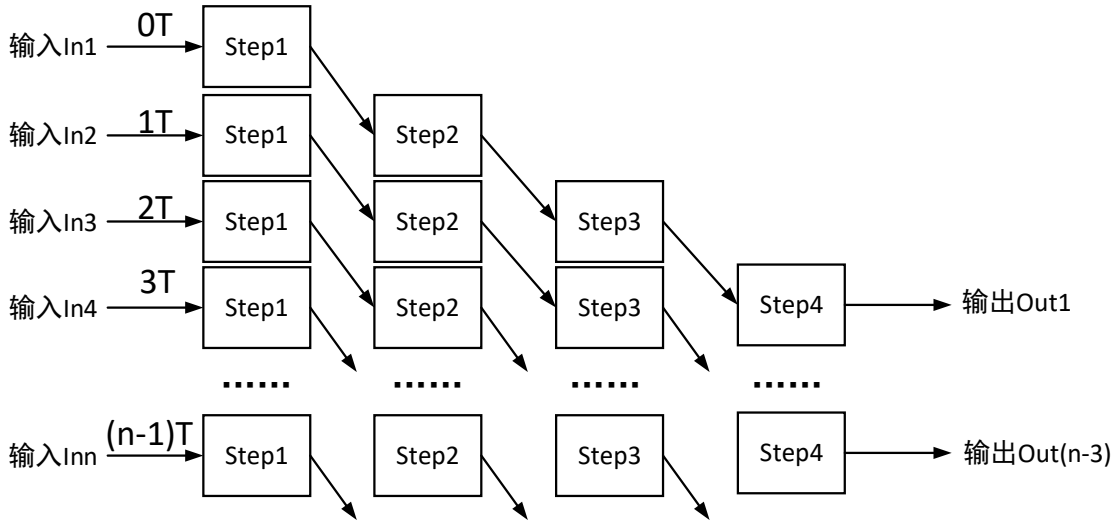

图 3.7 流水线步骤实现

MD5 算法的运算过程是串行执行，每一个逻辑函数计算的结果都要作为下一步逻辑函数计算的初始值，寄存器大部分时间都处于等待状态，故运算速度慢，数据吞吐量小。文献[26]以并行计算为基础，优化 MD5 算法实现的硬件架构，采用三级、四级流水线进行模块设计，文献[27]首先分析了 1、4、32 级多种流水线设计的吞吐性能，最终利用 32 级流水线设计，设计结果数据吞吐量达到了 32Gbps，但整个设计只对流水线结构进行扩展，电路结构非常复杂。文献[29]对数据流进行优化，提出多种方案实现 MD5 算法，最佳方案最终实验结果表明数据吞吐量达到了 66.56Gbps。但资源占用率较高，实现 MD5 算法对硬件要求高，不易实现。

MD5 算法实现时优化算法设计架构，借助流水线思想完成 64 步循环计算。将两步运算作为流水线的一级，构建一个 32 级的流水线设计，来完成 MD5 的 64 步运算。实现占用相对较少资源的基础上提高算法的运算速度，增大数据吞吐量，仅需 35 个时钟周期可以完成一次消息摘要值的计算。

3.3 MD5 哈希值计算模块设计

整个模块的外部接口可以分为三部分：消息经过预处理后的输入数据信号、哈希变换后的哈希值输出信号以及相应的控制信号，如图 3.8 所示。

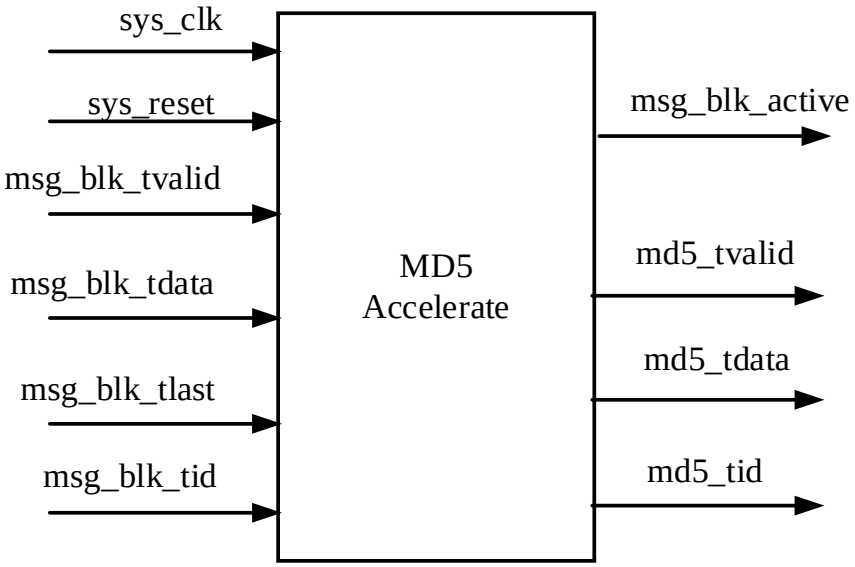

图 3.8 MD5 算法顶层模块示意图

顶层模块名为 MD5 Accelerate，功能是可同时计算 32 个通道消息分组(512-bit)的 MD5 摘要值。由于输入消息数据长度不一，算法第 1、2 步填充数据在输入数据预处理模块中实现，完成了对明文消息的预处理操作，并输出预处理的 512-bit 数据块数据和对应控制信号，输入到 MD5 Accelerate 模块中。

输入数据是 512-bit 位宽大小的数据块，同一个消息可能有多个消息分组(512-bit)也称数据块组成，对于属于同一个消息的不同消息分组，利用输入该消息分组(512-bit)的通道编号(tid)，以及开始(隐含)和结束(tlast)的标志来进行判别。如果为同一个消息的不同消息分组，输入该消息分组(512-bit)的通道编号(tid)始终都一致，非最后一组的消息分组结束信号(tlast)为低电平，直到最后一组的消息分组结束信号(tlast)才为高电平。对于不同消息的消息分组，利用开始(隐含)和结束(tlast)的标志来进行判别，表示最后一组的消息分组结束信号(tlast)为高电平。整个模块包括 2 个子模块 MD5_Pipe_Control 和 MD5_Pipe_Core，关系如图 3.9 所示。

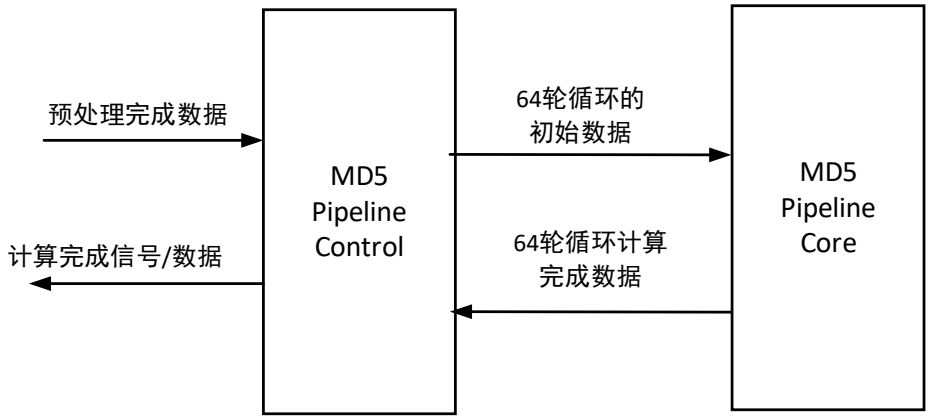

图 3.9 MD5 哈希值算法模块结构图

其中 MD5 顶层模块设计的输入输出端口如表 3.2 所示。

表 3.2 MD5 顶层模块端口

名称	位宽	方向	描述
sys_clk	1	I	时钟端口，为内部逻辑或其他端口提供同步参考。
sys_reset_n	1	I	复位端口，低电平有效，同步复位内部逻辑或其他端口信号到特定状态。
msg_blk_tvalid	1	I	“1”表示消息分组数据 msg_blk_tdata 有效；“0”表示消息分组数据 msg_blk_tdata 无效
msg_blk_tdata	512	I	输入的预处理完成的消息分组数据即消息数据块
msg_blk_tlast	1	I	“1”表示消息的最后一个分组数据块；“0”表示此时的 msg_blk_tdata 数据块并非是消息的最后一个分组数据块
msg_blk_tid	5	I	消息分组的通道编号
msg_blk_active	32	O	msg_blk_active[i]表示通道 i 消息分组数据块(512-bit)的工作状态，”1”表示被处理中，“0”表示空闲状态
md5_tvalid	1	O	“1”表示输出的 MD5 摘要值 md5_tdata 有效；“0”表示输出的 MD5 摘要值 md5_tdata 无效
md5_tdata	128	O	MD5 摘要值或哈希值
md5_tid	5	O	MD5 摘要值对应的通道编号

3.3.1 循环计算模块

循环计算模块作为 MD5 哈希值计算模块设计中的一个子模块，命名为 MD5 Pipeline Core，其输入输出端口如图 3.10 所示。实现功能是实现消息分组(512-bit)的 HASH 运算，共包括 4 轮，每轮包括 16 步运算，共 64 步。为了兼顾吞吐量与延时，将 2 步运算作为流水线的一级，构建一个 32 级流水线，完成 MD5 的 64 步运算。32 级流水线可以同时处理 32 个消息分组，对于属于同一个消息的不同消息分组，比如 M0 和 M1，由于 M1 的 HASH 运算依赖于 M0 的 HASH 结果，所以 32 个流水级中不允许出现同一个消息的不同消息分组。

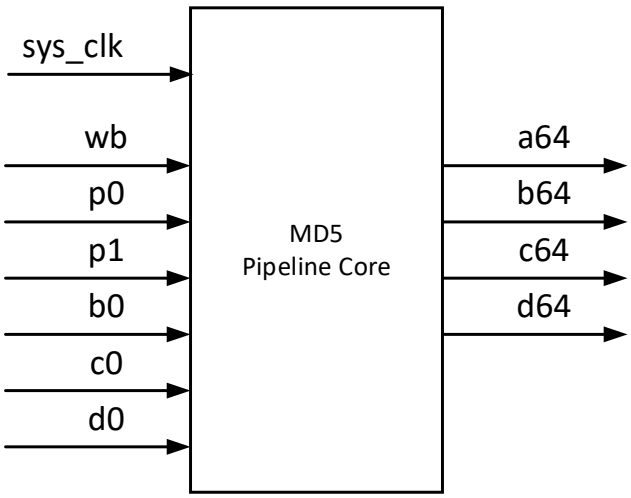

图 3.10 循环计算模块端口图

如表 3.3 所示，为 MD5 Pipeline Core 模块的输入输出端口信号。

表 3.3 循环迭代模块端口

名称	位宽	方向	描述
sys_clk	1	I	时钟端口
wb	512	I	输入的预处理完成的消息分组数据即消息数据块
p0	32	I	预处理计算 a+t0+w0 的值
p1	32	I	预处理计算 d+t1+w1 的值
b0	32	I	初始化链接变量 b 值
c0	32	I	初始化链接变量 c 值
d0	32	I	初始化链接变量 d 值
a64	32	O	循环迭代计算完成后输出的 a 值
b64	32	O	循环迭代计算完成后输出的 b 值
c64	32	O	循环迭代计算完成后输出的 c 值
d64	32	O	循环迭代计算完成后输出的 d 值

其中关于 p0、p1 数据预处理操作代码实现如图 3.11 所示。模块实现采用以 2 步运算为一级，实现 32 级流水线设计。若要对 2 组数据并行化处理，64 轮循环迭代计算中首先处理函数：

$$FF0(a,b,c,d,M[0],7,0xd76aa478) \tag{3.5}$$

$$FF1(d,a,b,c,M[1],12,0xe8c7b756) \tag{3.6}$$

故预处理 p0 为 a0+t0+M0 的值，p1 为 d0+t1+M1 的值。

```
always@(posedge sys_clk)
begin
    if (msg_blk_tvalid)
    begin
        case (msg_blk_tid)
            0 : if (!msg_blk_not1st[0])  //last group
                begin
                    p0 <= #tDLY 32'h67452301 + 'hd76aa478 + msg_blk_tdata[31:0];  //FF0表示FF函数的M0子分组
                    p1 <= #tDLY 32'h10325476 + 'he8c7b756 + msg_blk_tdata[63:32]; //FF1_M1
                    b0 <= #tDLY 32'hefcdab89;
                    c0 <= #tDLY 32'h98badcfe;
                    d0 <= #tDLY 32'h10325476;
                end
               else //实际上在仿真过程中输出给core的p0\p1\b0\c0\d0都是一样的，就是tdata的分组数据不同
                begin
                    p0 <= #tDLY a[0]  + 'hd76aa478 + msg_blk_tdata[31:0];   //a[0]=='h67452301 b[0]=='hefcdab89
                    p1 <= #tDLY d[0]  + 'he8c7b756 + msg_blk_tdata[63:32];  //C[0]=='h98badcfe d[0]=='h10325476
                    b0 <= #tDLY b[0] ;
                    c0 <= #tDLY c[0] ;
                    d0 <= #tDLY d[0] ;
                end
            1 : if (!msg_blk_not1st[1])  begin p0 <= #tDLY 32'h67452301 + 'hd76aa478 + msg_blk_tdata[31:0];
                                               p1 <= #tDLY 32'h10325476 + 'he8c7b756 + msg_blk_tdata[63:32];
                                               b0 <= #tDLY 32'hefcdab89; c0 <= #tDLY 32'h98badcfe;
                                               d0 <= #tDLY 32'h10325476;
                                          end
                                        else begin
                                               p0 <= #tDLY a[1]  + 'hd76aa478 + msg_blk_tdata[31:0];
                                               p1 <= #tDLY d[1]  + 'he8c7b756 + msg_blk_tdata[63:32];
                                               b0 <= #tDLY b[1] ; c0 <= #tDLY c[1] ; d0 <= #tDLY d[1] ;
                                          end
            2 : if (!msg_blk_not1st[2])  begin p0 <= #tDLY 32'h67452301 + 'hd76aa478 + msg_blk_tdata[31:0]; p1 <= #
```

图 3.11 p0/p1 预处理操作代码实现

关于处理 b0、c0、d0 值，分两种情况考虑。第一种为正在处理的数据为第一个分组消息，b0、c0、d0 值为初始化链接变量的值；第二种是正在处理的数据并非第一个分组消息，进行关于非第一个分组消息的 MD5 算法运算时需要继续使用上一组分组消息计算完成时寄存器 b0、c0、d0 的值。

预处理完成后，完成 FF 函数计算，如图 3.12 所示。其余 3 轮循环迭代运算如 GG、HH、II 函数与 FF 函数计算类似。

```
// F-round operation
assign  pre_add[0]  = c0  + 'h242070db + w0[2];     //p2,直接得到，下一T赋值给p2
assign  pre_add[1]  = b0  + 'hc1bdceee + w0[3];     //每1T出现两个pre_add值
assign  pre_add[2]  = c2  + 'hf57c0faf + w2[4];     //其实w0，w2内容一样，只是便于书写
assign  pre_add[3]  = b2  + 'h4787c62a + w2[5];
assign  pre_add[4]  = c4  + 'ha8304613 + w4[6];
assign  pre_add[5]  = b4  + 'hfd469501 + w4[7];
assign  pre_add[6]  = c6  + 'h698098d8 + w6[8];
assign  pre_add[7]  = b6  + 'h8b44f7af + w6[9];
assign  pre_add[8]  = c8  + 'hffff5bb1 + w8[10];
assign  pre_add[9]  = b8  + 'h895cd7be + w8[11];
assign  pre_add[10] = c10 + 'h6b901122 + w10[12];
assign  pre_add[11] = b10 + 'hfd987193 + w10[13];
assign  pre_add[12] = c12 + 'ha679438e + w12[14];
assign  pre_add[13] = b12 + 'h49b40821 + w12[15];
assign  pre_add[14] = c14 + 'hf61e2562 + w14[(5 * 16 + 1) % 16];
assign  pre_add[15] = b14 + 'hc040b340 + w14[(5 * 17 + 1) % 16];

assign  fadd_bcd0  = p0  + ((b0  & c0)  | ((~b0)  & d0)) ;
assign  fadd_bcd1  = p1  + ((b1  & c1)  | ((~b1)  & d1)) ;//p0p1外部输入已经准备好，直接得到
assign  fadd_bcd2  = p2  + ((b2  & c2)  | ((~b2)  & d2)) ;//与起始时钟相隔1T得到bcd2.bcd3
assign  fadd_bcd3  = p3  + ((b3  & c3)  | ((~b3)  & d3)) ;
assign  fadd_bcd4  = p4  + ((b4  & c4)  | ((~b4)  & d4)) ;
assign  fadd_bcd5  = p5  + ((b5  & c5)  | ((~b5)  & d5)) ;
assign  fadd_bcd6  = p6  + ((b6  & c6)  | ((~b6)  & d6)) ;
assign  fadd_bcd7  = p7  + ((b7  & c7)  | ((~b7)  & d7)) ;
assign  fadd_bcd8  = p8  + ((b8  & c8)  | ((~b8)  & d8)) ;
assign  fadd_bcd9  = p9  + ((b9  & c9)  | ((~b9)  & d9)) ;
assign  fadd_bcd10 = p10 + ((b10 & c10) | ((~b10) & d10));
assign  fadd_bcd11 = p11 + ((b11 & c11) | ((~b11) & d11));
assign  fadd_bcd12 = p12 + ((b12 & c12) | ((~b12) & d12));
assign  fadd_bcd13 = p13 + ((b13 & c13) | ((~b13) & d13));
assign  fadd_bcd14 = p14 + ((b14 & c14) | ((~b14) & d14));
assign  fadd_bcd15 = p15 + ((b15 & c15) | ((~b15) & d15));

assign  b1  = b0  + {fadd_bcd0[24:0] , fadd_bcd0[31:25]} ; assign a1  = d0 ; assign c1  = b0 ; assign d1  = c0;//直接得到，与
assign  b3  = b2  + {fadd_bcd2[14:0] , fadd_bcd2[31:15]} ; assign a3  = d2 ; assign c3  = b2 ; assign d3  = c2;//相隔1T得到，
assign  b5  = b4  + {fadd_bcd4[24:0] , fadd_bcd4[31:25]} ; assign a5  = d4 ; assign c5  = b4 ; assign d5  = c4;
assign  b7  = b6  + {fadd_bcd6[14:0] , fadd_bcd6[31:15]} ; assign a7  = d6 ; assign c7  = b6 ; assign d7  = c6;
assign  b9  = b8  + {fadd_bcd8[24:0] , fadd_bcd8[31:25]} ; assign a9  = d8 ; assign c9  = b8 ; assign d9  = c8;
assign  b11 = b10 + {fadd_bcd10[14:0], fadd_bcd10[31:15]}; assign a11 = d10; assign c11 = b10; assign d11 = c10;
assign  b13 = b12 + {fadd_bcd12[24:0], fadd_bcd12[31:25]}; assign a13 = d12; assign c13 = b12; assign d13 = c12;
assign  b15 = b14 + {fadd_bcd14[14:0], fadd_bcd14[31:15]}; assign a15 = d14; assign c15 = b14; assign d15 = c14;
```

图 3.12 FF 函数计算代码实现

3.3.2 流水线控制模块

流水线控制模块作为 MD5 哈希值计算模块设计中的另一个子模块，命名为 MD5 Pipeline Control，其输入输出端口如图 3.13 所示。关于输入输出端口具体意义上文已阐述，故不再重复。

子模块 MD5_Pipeline_Control 接收至多 32 个通道的消息分组(512-bit)，发送到 MD5 Pipeline Core 子模块进行 HASH 运算，同时接收 MD5 Pipeline Core 子模块的 HASH 结果进行输出。该模块提供 MD5 Pipeline Core 处理的消息分组(512-bit)的工作状态 msg_blk_active[31:0]，msg_blk_active[i]表示通道 i 消息分组(512-bit)的工作状态，“1”表示被处理中，“0”表示空闲状态。

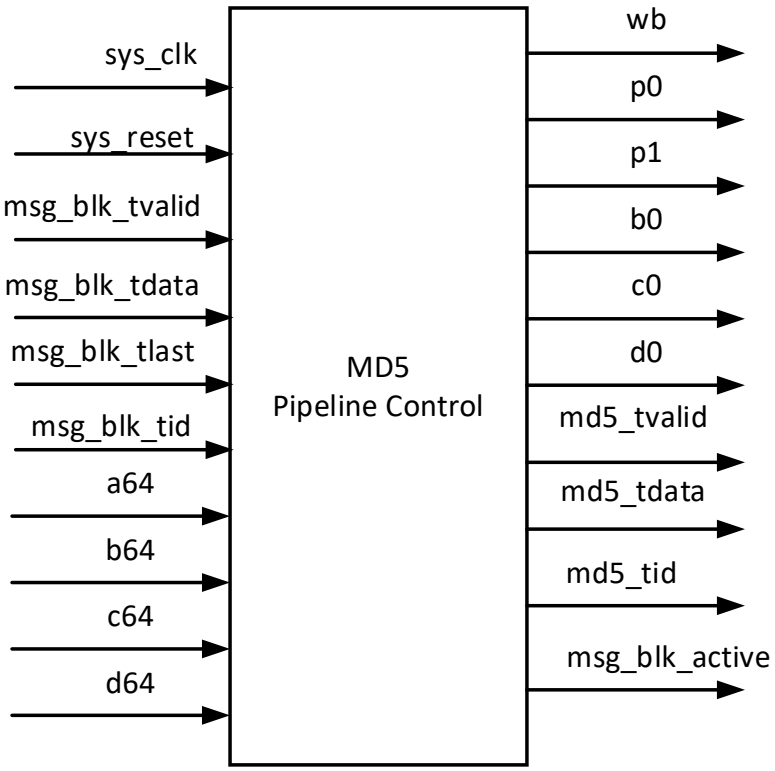

图 3.13 流水线控制模块端口图

由于 MD5 算法的运算具有依赖关系，消息分组(512-bit)的输入需要依据 msg_blk_active[31:0]，即如果 msg_blk_active[i]等于“0”，通道 i 可以输入消息分组(512-bit)。对于同一个消息有多个消息分组(512-bit)组成的情况，如果输入数据的 tid 相同且上一个 tid 对应的 tlast 不为 0 即上一个输入的数据不是对应 tid 的最后一个消息分组，此时输入数据依然是同一消息的一个分组。如果该消息分组(512-bit)的结束标志(tlast)为“1”，经过 35 个流水线延时后，模块可输出该消息的 MD5 摘要值。具体时序如图 3.14 所示。

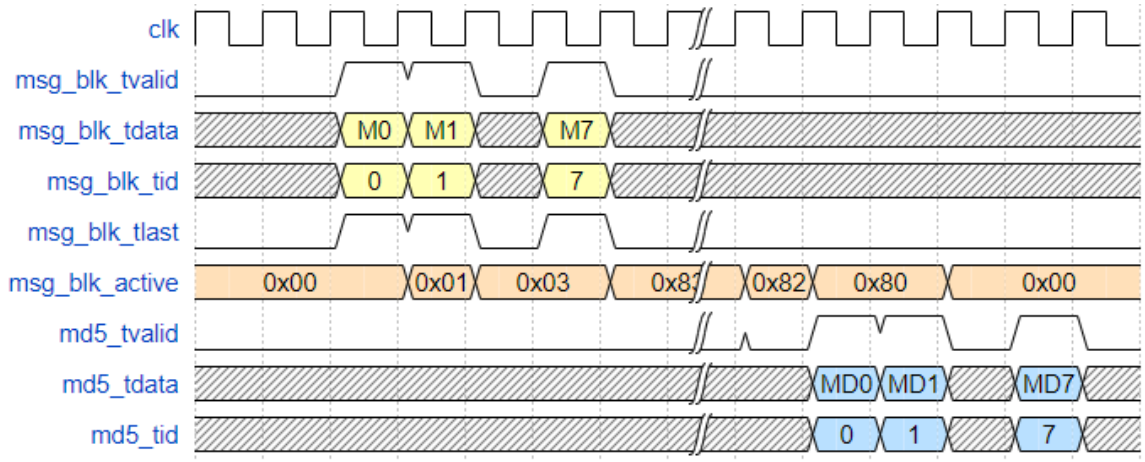

图 3.14 单个消息分组时序图

图 3.14 表示 3 个消息分别输入通道 0，1 和 7，这 3 个消息均只包含一个消息分组，比如通道 0 输入的消息分组 M0：msg_blk_tvalid 第一次有效时 msg_blk_tlast 等于 1，表示该消息分组是第一个也是最后一个分组。收到消息分组 M0 后 msg_blk_active[0]会置 1，表示该消息分组正在被处理。从输入消息分组 M0 到输出摘要 MD0 的时间间隔是 35 个时钟周期，这个延时是固定的。

图 3.15 表示 3 个消息分别输入通道 0，1 和 7，第一个消息包含 2 个消息分组：M00 和 M01，直到 msg_blk_active[0]等于 0，通道 0 才能输入下一个消息分组 M01：msg_blk_tvalid 有效且 msg_blk_tlast 等于 1，表示该消息分组是最后一个分组。从输入消息分组 M01 到输出摘要 MD0 的时间间隔是 35 个时钟周期。

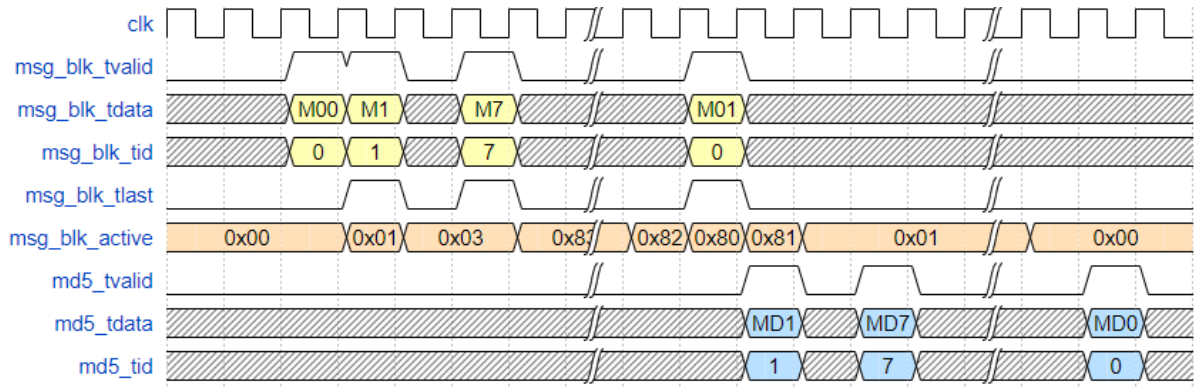

图 3.15 多个消息分组时序图

流水线控制模块实现处理计算得到的 a64、b64、c64、d64 值。初始化链接变量后，等到 64 轮循环迭代计算完成，得到新的 a64、b64、c64、d64 值。将 a64、b64、c64、d64 值与之前寄存器 a、b、c、d 值进行叠加，得到新值 a、b、c、d，具体的代码实现如图 3.16 和 3.17 所示。

```
// generate a, b, c, d
generate
begin
    genvar i;
    for (i = 0; i < 32; i = i + 1)
    begin : gen_a_b_c_d
        always@(posedge sys_clk)
        begin
            if (msg_blk_tvalid && (msg_blk_tid == i) && (!msg_blk_not1st[i]))
            begin
                a[i] <= #tDLY 32'h67452301;
                b[i] <= #tDLY 32'hefcdab89;
                c[i] <= #tDLY 32'h98badcfe;
                d[i] <= #tDLY 32'h10325476;
            end
            else if (pipe_valid[32] && (pipe_id[32] == i))
            begin
                a[i] <= #tDLY a[i] + a64;
                b[i] <= #tDLY b[i] + b64;
                c[i] <= #tDLY c[i] + c64;
                d[i] <= #tDLY d[i] + d64;
            end
        end
    end
end
endgenerate
```

图 3.16 部分代码实现

如果此时处理的数据为最后一个消息分组，则将数据以小端字节序格式输出，代码实现如所示，在同一时钟周期拉高数据有效信号 msg_blk_tvalid。否则下一组消息分组计算中利用新值 a、b、c、d 作为链接变量。

```
// output md5
always@(posedge sys_clk, negedge sys_reset_n)
begin
    if (!sys_reset_n)
    begin
        md5_tvalid <= #tDLY 1'b0;
    end
    else
    begin
        if (pipe_valid[33] && pipe_last[33])
        begin
            md5_tvalid <= #tDLY 1'b1;
        end
        else
        begin
            md5_tvalid <= #tDLY 1'b0;
        end
    end
end

always@(posedge sys_clk)
begin
    case (pipe_id[33])
        0  : md5_tdata <= #tDLY {d[0] , c[0] , b[0] , a[0] };   //小端字节序输出
        1  : md5_tdata <= #tDLY {d[1] , c[1] , b[1] , a[1] };
        2  : md5_tdata <= #tDLY {d[2] , c[2] , b[2] , a[2] };
        3  : md5_tdata <= #tDLY {d[3] , c[3] , b[3] , a[3] };
        4  : md5_tdata <= #tDLY {d[4] , c[4] , b[4] , a[4] };
        5  : md5_tdata <= #tDLY {d[5] , c[5] , b[5] , a[5] };
```

图 3.17 部分代码实现

3.4 本章小结

本章首先介绍了实现 MD5 算法架构设计，由输入数据预处理模块和 MD5 哈希值计算模块两部分组成，随后详细介绍各模块的功能、原理框图，接着以 MD5 哈希值计算模块为核心，提出优化 MD5 算法实现方案，介绍实现模块设计框架，解释顶层模块、子模块各端口信号意义，介绍部分关键部分的代码实现，最后画出 MD5 哈希值计算模块功能实现的时序图。

第4章 MD5 算法的仿真与性能分析

4.1 测试方案设计

随着硬件技术的不断发展，SoC 规模以及复杂度逐步提升，这对芯片验证提出更高要求。目前 SoC 常用的验证方法有：

1. EDA 功能仿真技术；
2. 基于 VIP 的验证技术；
3. 软硬件协同验证技术；
4. FPGA 板级验证技术。

FPGA 验证能够有效提高 SoC 的质量和设计效率，因为 FPGA 的可编程性可以模拟芯片应用的绝大多数甚至全部场景和状态，并对芯片进行 FPGA 反复验证，从而弥补在验证期间未发现的缺陷，再次完善芯片设计，以达到充分验证的目的，提高芯片的质量，极大地节省芯片开发成本。

图 4.1 是 FPGA 设计验证流程，首先根据用户或市场需求规定目标实现的功能，再根据功能制定实现方案，规定功能定义规划好输入输出端口信号。根据设计理念做好模块设计，分模块利用 Quartus II 软件进行 RTL 实现，利用 QuestaSim 完成仿真。最后进行 FPGA 板级验证，这种方式可以高效、快速的完成开发设计。

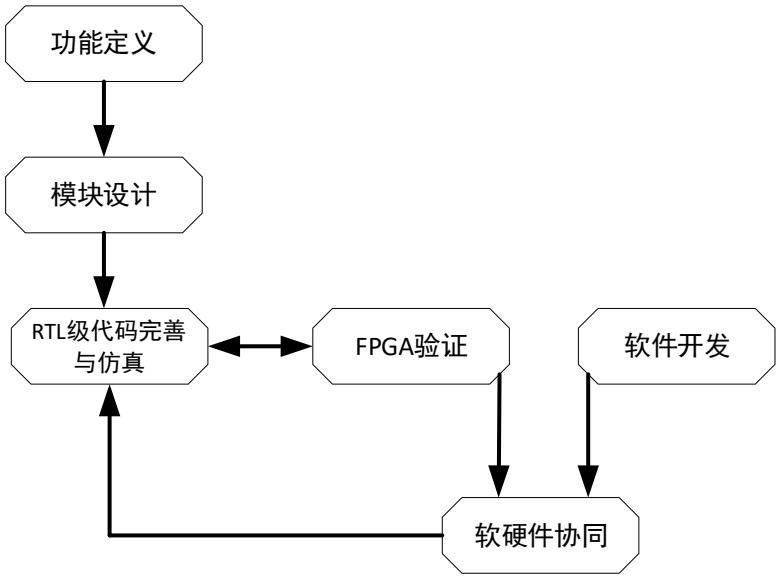

图 4.1 FPGA 设计验证流程

在之前的 SoC 验证过程中，软件调试只出现在流片完成后用于检测芯片的正确性，但现在可发生在搭建完 FPGA 软硬件设计平台中，现在的高端 FPGA 芯片已经集成了 cpu 核，可以完成系统级的验证，软件开发者可以直接编写软件 case 进行调试，及时反馈调试结果给芯片设计部门，从而提高芯片研发效率，对芯片

验证也有极大的帮助。

在FPGA验证之前，首先要对RTL级代码完善后的MD5算法进行功能验证，主要侧重于进行RTL仿真，这种功能仿真是理想条件下的，在实际电路中信号的传播过程是存在延时的，所以功能仿真并不能完成保证电路的正确性，其目的是检测出功能逻辑的缺陷。

其中RTL仿真的验证流程：

1. 制定验证策略和计划

2. 确定验证平台。验证平台采用的是QuestaSim。

3. 提取testcase。每个需求对应一到多个testcase，有些需求，如复位、使能在其他用例也能测试到，可以不用单独测试。采用的验证方法学本质就是个组合问题，分为面对输入数据还是面对内部功能或者是两者的结合。对应采用的验证原则是从简单到复杂，从定向到随机；

本文根据MD5算法涉及到的功能点进行仿真验证，其具体策略如表4.1所示。验证层次有模块级、子系统级、系统级，本文验证层次均是模块级的。

表4.1 仿真测试方案

序号	功能点	场景	验证的方法	透明度
1	MD5摘要计算	输入的有效消息数据在填充前，长度对512求余的结果覆盖小于448、等于448、大于448。	动态仿真	黑盒
2	MD5摘要计算	同时处理分组消息的数量覆盖边界值1、32。	动态仿真	黑盒
3	消息分组工作状态指示	输入有效的消息分组数据后，经过1个时钟周期，对应的工作状态指示变为1，再经过34个时钟周期，对应的工作状态指示变为0。	动态仿真	黑盒
4	摘要输出延时	输入消息的最后一个有效分组数据至输出摘要的延时为35个时钟周期。	动态仿真	黑盒
5	随机控制	随机控制出现上述场景，连续3次场景覆盖长度对512求余3种场景的排列组合。	动态仿真	黑盒

注：可以同时处理32个分组消息(512bit)，同时处理的分组消息中不允许出现同一个消息的不同消息分组。

(1)验证的方法：动态仿真、形式验证、硬件加速。

(2)透明度：白盒、黑盒、灰盒。其中黑盒验证：内部不可见，只看输入到输出，适用于算法/IP/简单模块；白盒验证：内部可见，不需要搭建 GOLDEN 模型，也就是常说的黄金参考模型，可以添加断言 assertion、监控器来保证操作正确性；灰盒验证：模块间信号线可见，在模块间加测试点，需要参考模型；验证工作量比较适中。

4. 覆盖率分析

QuestaSim 工具可以帮助开发人员进行覆盖率测试，涵括了代码、功能和断言三种，并统一存储在同一数据库中。利用软件进行 coverage 覆盖率测试过程中，Compile 编译时设置 Compile Option，打开测试的覆盖率功能，其中有类型包括有 toggle、branch、fsm、condition、line 五种，在 Simulate 时将工程调用后 enable coverage，并跑完工程后即可看到覆盖率情况。

4.2 MD5 算法功能仿真

使用 Questa Sim 进行仿真可通过利用 Quartus II 软件直接调用，或者单独进行功能仿真，其仿真流程简单分为五个步骤，如图 4.2 所示。分别创建工程文件、编写 RTL 源代码或添加代码文件、编译、仿真软件仿真、仿真结果分析。通过这样的方式，对上节提出的功能点编写对应的 testcase 文件，从而实现对 MD5 算法设计进行仿真验证，达到验证算法功能的目的。

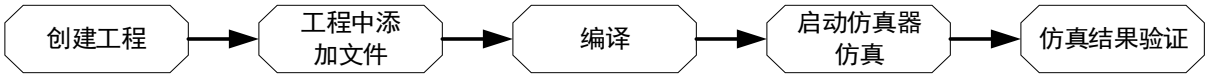

图 4.2 仿真设计流程图

4.2.1 单个消息块仿真验证

对消息长度不足 448 的明文信息进行 MD5 函数运算，验证映射得到的 128 比特位的摘要值。举例验证输入的消息明文信息 Mn 为“2013_4501_33”，验证经过 MD5 算法运算后的结果。

首先对明文消息执行预处理操作。先对明文信息进行数据填充。第一步将明文信息由 ASCII 码转换成二进制或者十六进制表示，为便于浏览转换成十六进制的值为 3230_3133_3435_3031_3333。第二步对数据进行填充，由于明文信息数据长度为 10*8 即 80 个比特位，而且 80 除以 512 余 80，此值并不等于 448，要对明文信息进行填充，填充方式是在明文数据后面填充 1 比特位的二进制数“1”和 367 比特位的二进制数“0”。第三步对数据进行扩展，即补齐代表明文长度信息的数据，长度信息如图 4.5 所示。当预处理完成后要将数据输入到 MD5 哈希值计算模块中去，同时伴随着数据控制信号一并输入。数据输入格式是以小端字节序输

入的即高字节数据放置在高地址，低字节数据放置在低地址，最后明文信息“2013_4501_33”预处理完成后的输入数据以及整个预处理操作如图 4.3 所示。

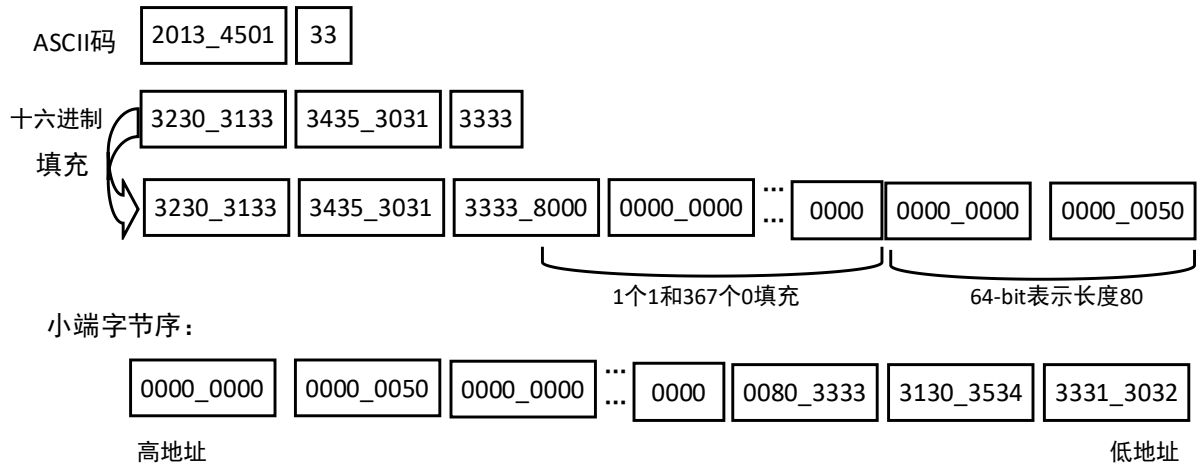

图 4.3 数据填充排序图

当预处理操作完成后输出输入数据 msg_blk_tdata，并在同一时钟周期生成对应的控制信号。拉高对应的数据有效信号 msg_blk_tvalid，数据有效信号只维持一个时钟周期；处理表征输入数据是否为最后一个有效 512-bit 数据块的信号 msg_blk_tlast，若这一输入数据为此明文消息的最后一组 512-bit 数据块则拉高表征信号 msg_blk_tlast，否则表征信号 msg_blk_tlast 保持为 0 状态。由于此次仿真验证是对单个消息块仿真验证，对应的表征信号 msg_blk_tlast 在同一时钟周期上一定拉高为高电平；由第三章可知在设计 MD5 哈希值计算模块时利用了 32 级流水线设计，设计允许 32 个通道同时进行 MD5 算法运算，输入数据占用的通道为 29 即消息分组的通道编号 msg_blk_tid 为 29，输入到 MD5 哈希值计算模块的信号如图 4.4 所示。

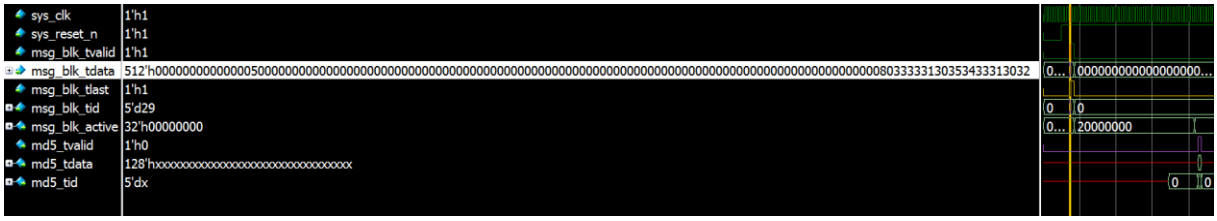

图 4.4 单个消息块的输入信号

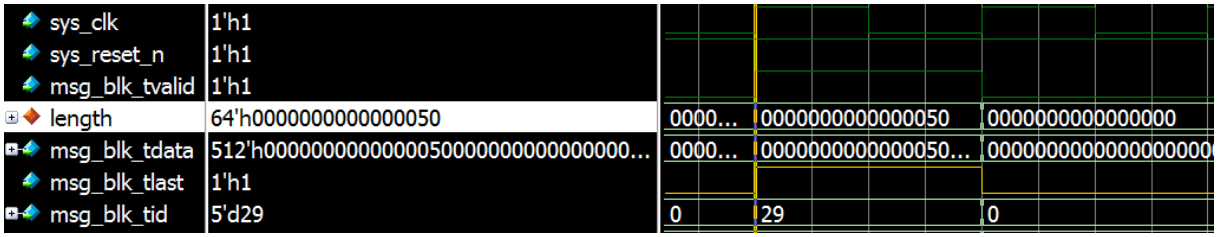

图 4.5 单个消息块的数据长度信息

提炼出输入数据 msg_blk_tdata 中表示明文消息长度的信息，如图 4.5 所示，

其中 length 信号代表的数据长度为 0x50 即 80 个比特位。证明明文消息预处理操作的正确性。

输入数据后经过一系列循环迭代运算，历经 700ns 即 35 个时钟周期后得到 MD5 哈希值即摘要值 MDn0= “f912_90bd_eba4_9712_a0a6_c599_1864_6a0e”，单个消息块的仿真结果如图 4.6 所示。

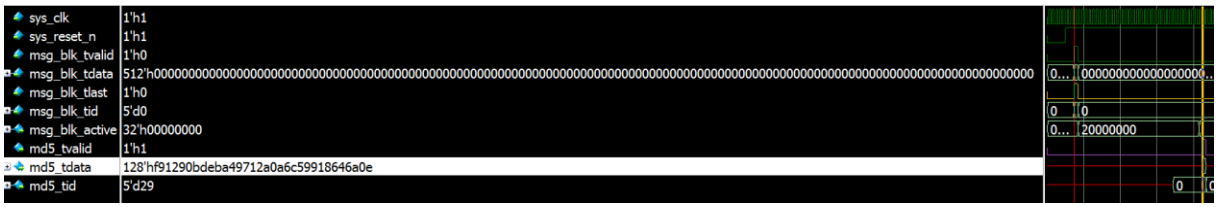

图 4.6 单个消息块的仿真结果

利用 MD5 加密算法在线转换器，得到的结果如图 4.7 所示。转换器的结果是以大端字节序显示，大端字节序与小端的恰恰相反，是将高 byte 位存放在低 address，低 byte 位存放在高 address。将输入相同的明文消息 Mn 进行 MD5 加密算法转换为 128 比特的摘要值 MDn1 与将明文消息 Mn 经过模块设计的 MD5 算法运算实现得到 MD5 摘要值 MDn0 的结果进行比较，可得出两者比对成功，即 MDn1=MDn0，故对于单个消息块的仿真验证，利用 MD5 加密算法验证仿真结果的准确性，从而可保证 MD5 算法运算的正确性。

图 4.7 MD5 加密算法转换器结果

4.2.2 多个消息块仿真验证

对于明文长度大于或者等于 448 比特位的明文信息，经过预处理操作后输入到 MD5 哈希值计算模块中进行 MD5 算法运算。通过举例验证输入明文信息 Mx，验证经过 MD5 算法运算后的结果是否与 MD5 加密算法转换器得到的结果一致。

执行步骤如下：

1. 将 Mx 执行预处理操作

(1)对明文信息 Mx 进行数据进制转换。将明文信息 Mx 由 ASCII 码转换成十六进制表示，Mx 转换成十六进制的值；

(2)对数据进行填充。由于明文信息数据长度为 115*8 即 920 个比特位，用十六进制表示为 0x398，如图 4.8 所示。而且 920 除以 512 余 408，此值并不等于 448，要对明文信息进行填充，填充方式是在明文数据后面填充 1 比特位的二进制数“1”和 39 比特位的二进制数“0”；

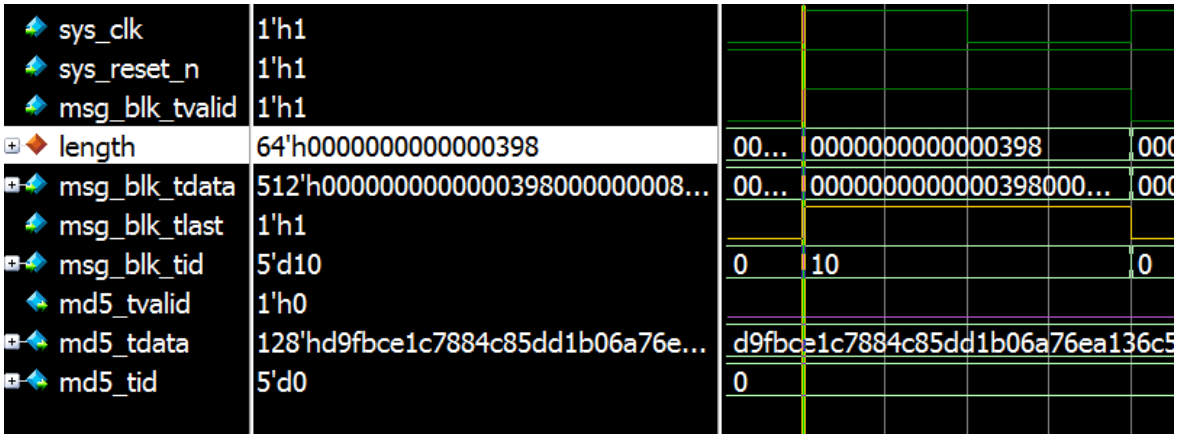

图 4.8 多个消息块数据长度

(3)对数据进行扩展。即补齐代表明文长度信息的数据。当预处理完成后要将数据输入到 MD5 哈希值计算模块中去，同时伴随着数据控制信号一并输入。数据输入格式是小端字节序，即将高字节数据放置在高地址，低字节数据放置在地址，最后明文信息预处理完成后的输入数据由三个 512-bit 的数据块组成，具体数据见图 4.9、图 4.10。

当预处理操作完成后输出三组 512-bit 的输入数据 msg_blk_tdata，该输入数据只能从一个数据通道输入，设计 MD5 哈希值计算模块时利用了 32 级流水线设计，设计允许 32 个通道同时进行 MD5 算法运算，输入数据占用的通道为 10 即消息分组的通道编号 msg_blk_tid 为 10，并且同一消息只能在一个数据通道中进行处理，也就是说 MD5 算法运算完一组数据块才可紧接着输入同一消息的下一组数据，直至这一消息的数据块全部循环迭代计算完输出对应的 128-bit 哈希值为止，才会释放 msg_blk_tid 为 10 的数据通道，以供其他消息数据输入。

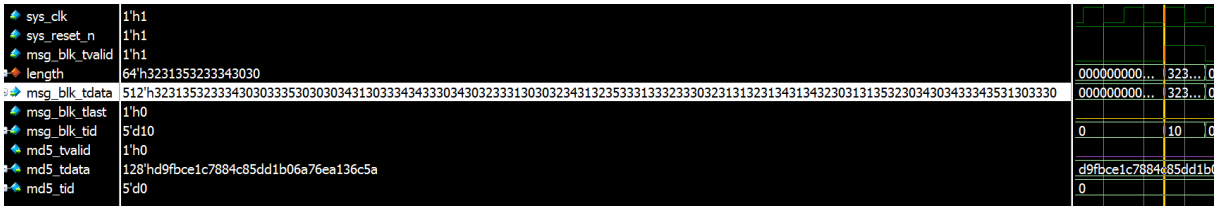

图 4.9 多个消息块数据第一分组

在同一时钟周期生成对应的控制信号。拉高表征输入数据有效的信号即此时 msg_blk_tvalid 为 1；处理表征输入数据是否为最后一个有效 512-bit 数据块的信号，若这一输入数据为此明文消息的最后一组 512-bit 数据块则拉高表征信号即 msg_blk_tlast 为 1，否则表征信号 msg_blk_tlast 保持为 0 状态。由于此次仿真验证是对多个消息块仿真验证，对应的表征信号 msg_blk_tlast 在前面两组数据块输

入时为低电平状态，最后一组数据块输入时为高电平状态。具体控制信号高低电平情况如图 4.9、图 4.10 所示。

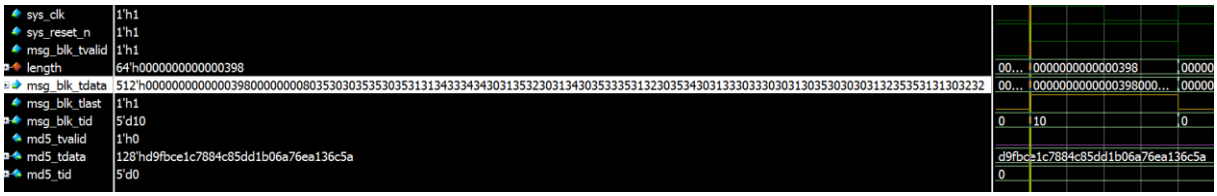

图 4.10 多个消息块数据最后一分组

2. MD5 算法运算得到仿真结果

输入数据后经过一系列循环迭代运算，接收最后一组数据块信息进行 MD5 运算，历经 700ns 即 35 个时钟周期后得到 MD5 哈希值即摘要值 MDx0，如图 4.11 所示，仿真结果利用的小端字节序规则显示。

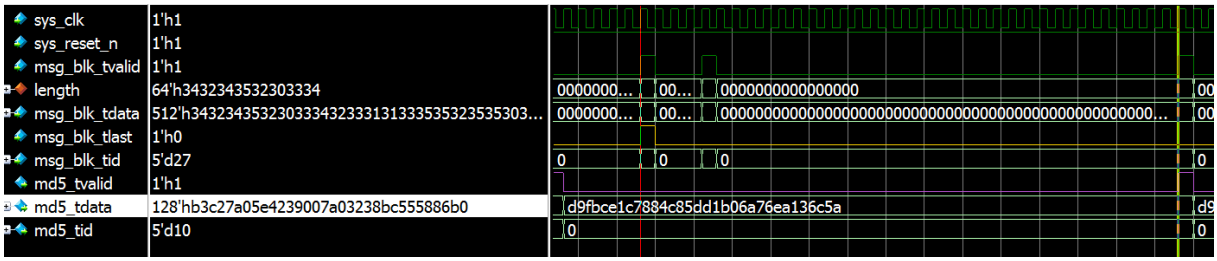

图 4.11 多个消息数据块仿真结果

3. 对比数据结果

利用 MD5 加密算法在线转换器，得到的结果如图 4.12 所示。转换器的结果是以大端字节序显示。将输入相同的明文消息 Mx 进行 MD5 加密算法转换为 128 比特的摘要值 MDx1 与将明文消息 Mx 经过模块设计的 MD5 算法运算实现得到 MD5 摘要值 MDx0 的结果进行比较，可得数据相等，即 MDx1=MDx0，故对于多个消息块的仿真验证，利用 MD5 加密算法在线转换器验证仿真结果的准确性，从而证明设计的 MD5 算法运算的正确性。

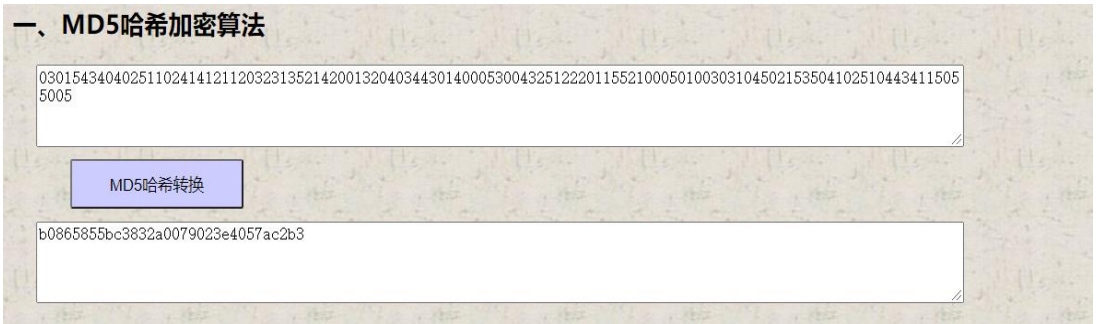

图 4.12 MD5 加密算法转换器结果

其中 MD5 算法实现其覆盖率情况如图 4.13 所示。其中代码覆盖率为 100%，表示的是模块设计中涉及的代码均验证到；功能覆盖率为 100%，表示模块设计中设计的功能点，在验证创建场景时覆盖到。

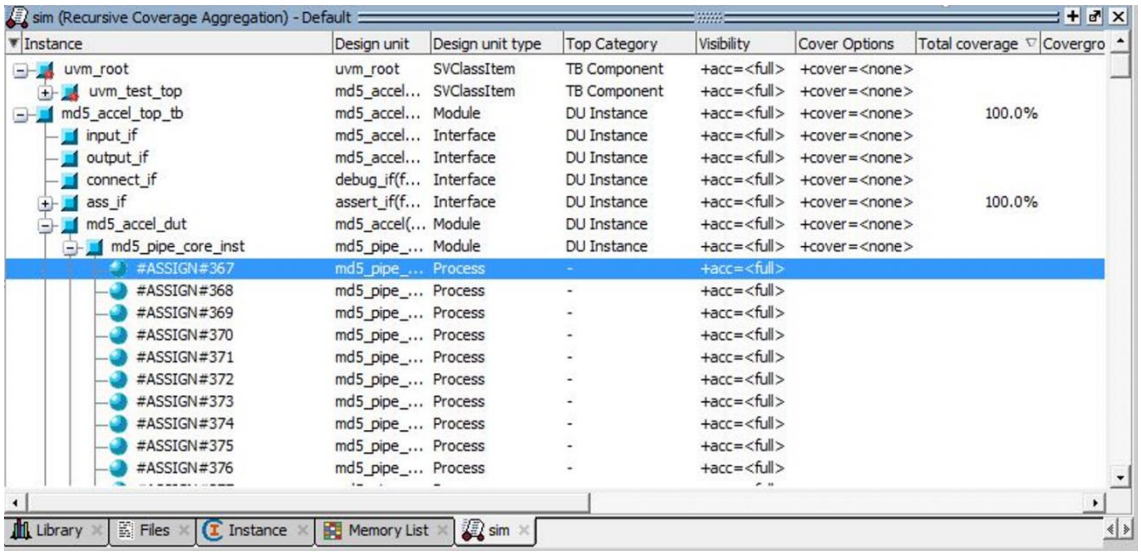

图 4.13 MD5 算法覆盖率情况

4.3 MD5 算法性能分析

MD5 算法设计环境利用 Altera 公司的 DEA 工具软件 Quartus II，利用 Quartus II、QuestaSim 软件完成 MD5 算法的设计、编译、综合分析、仿真验证等一系列操作，然后完成 FPGA 板级验证。该设计架构采用 32 级流水线设计，MD5 算法运算耗费 35 个时钟周期，包括循环迭代运算、3 个时钟周期的数据寄存处理的过程。已知数据吞吐率计算方法为：$P_{max} = f_{max} * B/D$*N，其中P_{max}为最大的数据吞吐量，f_{max}为最高时钟频率，B 为每个子分组数据块大小，D 为 MD5 算法运算的延迟周期，N 为设计中应用的流水线级数。

论文实现 MD5 算法是通过循环迭代、流水线设计两种实现模式下进行协同优化提高数据吞吐量，提高可运行的最高时钟频率以及减少硬件资源消耗。其优化方式是利用三级加法器替代四级加法器进行优化加法器设计和优化循环移位操作从而缩短 MD5 运算的关键路径，同时利用 32 级流水线设计完成对 MD5 加密算法的设计。最后通过 FPGA 板级验证，最终设计实现使用了 11903 个 ALUTs，最大的时钟频率 173MHz，占用寄存器为 12883 个，数据吞吐量最大为 81.31Gbps，其综合结果如图 4.14 所示。

表 4.2 FPGA 结果对比

设计	最大频率/MHz	寄存器资源/个	吞吐量/Mbps
文献[1]	48	30134	13010
文献[60]	64.45	5912	493.2
文献[29]	130	27050	66560
文献[28]	111	7253	56863
本设计	173.7	11903	81311

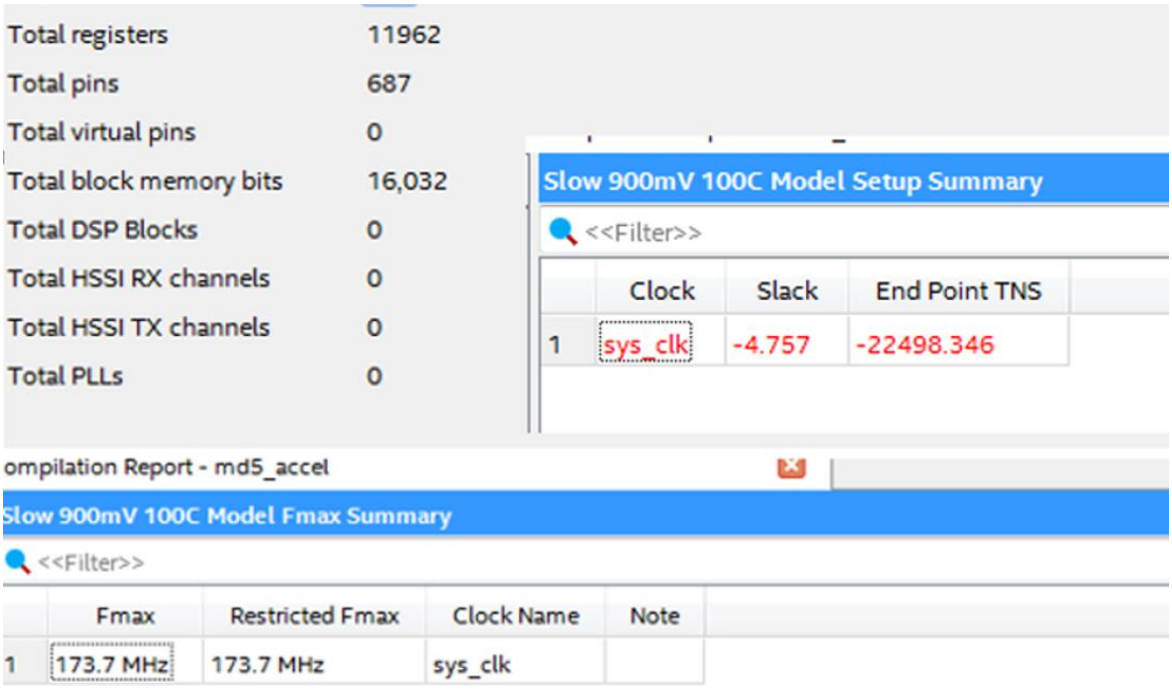

图 4.14 MD5 算法模块综合结果

表 4.2 列出了同类设计的资源消耗和性能比较，通过资源占用、执行速度、FPGA 上可运行最高时钟频率、数据吞吐量等方面进行对比，文献[1]中利用多块 FPGA 去并行处理 MD5 加密算法，文献[60]利用状态机完成基本的 MD5 算法运算，文献[28]和文献[29]和本文一样使用了流水线技术，但本设计吞吐率却比文献[28]和文献[29]的吞吐率高的多，以此来证明本论文利用 FPGA 进行硬件实现 MD5 算法结果优越性。

4.4 本章小结

本章对设计的 MD5 算法模块进行功能仿真验证。首先介绍模块级 IP 验证流程以及验证平台 QuestaSim 等，提出测试方案。接着对单个或多个消息块数据进行功能仿真和数据对比，验证仿真正确性。最后对 MD5 模块实现进行综合与性能分析，将此设计与现有的文献在资源占用、可运行最高时钟频率、数据吞吐量等方面进行对比，表明模块设计的优越性。

第5章 SHA-256 算法模块设计

5.1 SHA-256 算法模块简介

SHA-256 算法的顶层模块框图如图 5.1 所示。根据算法原理中对明文信息进行数据处理的过程可知，整个 SHA-256 散列算法和 MD5 加密算法一样。顶层模块由输入数据预处理模块以及 SHA-256 散列值计算模块两部分组成，输入数据预处理模块对应的功能是完成对输入明文信息的预处理，输出预处理完成数据即以 512-bit 为一单位的消息块以及对应的控制信息。SHA-256 散列值计算模块对应功能是完成明文信息进行 SHA-256 算法循环计算，输出散列值数据，即哈希值或摘要值。顶层模块对外连接的是接收明文消息输入、计算输出哈希值以及接收明文消息的信息即接收控制信号三部分。

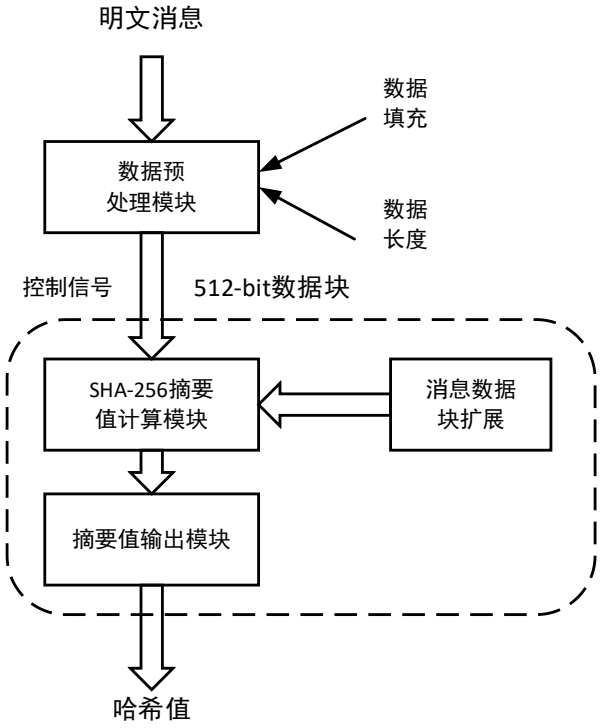

图 5.1 SHA-256 算法顶层模块框图

5.2 输入数据预处理模块

输入数据预处理模块完成对输入明文信息的预处理，输出预处理完成数据即以 512-bit 为一单位的消息块作为输出，同时输出对应的控制信息，就是将已完成预处理的数据以及控制信息输入到 SHA-256 散列值计算模块中。换句话说就是接

收明文信息，并对明文信息完成填充数据(1000…00)和扩展数据长度操作，实现输入到 SHA-256 计算模块的数据是每个分组为 512-bit 数据块，当字符串填充扩展至(N+1)*512 位(如果数据长度超出 448，则是(N+2)*512 位)时，按 512-bit 为一组，将整个消息数据进行分组输入，再进行 SHA-256 算法运算得到散列值。这一做法和 MD5 输入数据预处理操作类似，将不进行具体介绍。

5.3 SHA-256 散列值计算模块

SHA-256 算法中的计算模块主要是接收预处理完成的 512-bit 的分组数据，接收数据携带的控制信息，再进行 SHA-256 迭代计算，在计算完成后得到计算结果，再经处理后完成输出散列值。SHA-256 算法实现架构和 MD5 算法硬件实现相似，同时都采用 Verilog HDL 语言完成。各端口的定义以及功能说明如表 5.1，如图 5.2 所示为 SHA-256 散列值计算模块的输入输出端口图。

表 5.1 SHA-256 顶层模块功能端口

名称	位宽	方向	描述
sys_clk	1	I	时钟端口，为内部逻辑或其他端口提供同步参考。
sys_reset_n	1	I	复位端口，低电平有效，同步复位内部逻辑或其他端口信号到特定状态。
msg_blk_tlast	1	I	“1”表示消息的最后一个分组数据块；“0”表示此时的 msg_blk_tdata 数据块并非是消息的最后一个分组数据块
msg_blk_tid	5	I	消息分组的通道编号
msg_blk_active	32	O	msg_blk_active[i]表示通道 i 消息分组数据块(512-bit)的工作状态，“1”表示被处理中，“0”表示空闲状态
sha2_tvalid	1	O	“1”表示输出的 SHA-256 散列值 sha2_tdata 有效;“0”表示输出的 SHA-256 散列值 sha2_tdata 无效
sha2_tdata	256	O	SHA-256 摘要值或哈希值
sha2_tid	5	O	SHA-256 摘要值对应的通道编号

SHA-256 算法在硬件实现时，为了提高数据吞吐量，提升吞吐率，采用了二合一的化简方式,即以两次压缩为一步执行运算,数据扩展和压缩计算同步进行。SHA-256 算法采用了 32 级流水线设计，实现 64 轮循环迭代计算需要 32 个时钟周期完成的功能。本设计的数据通路如图 5.3 所示。

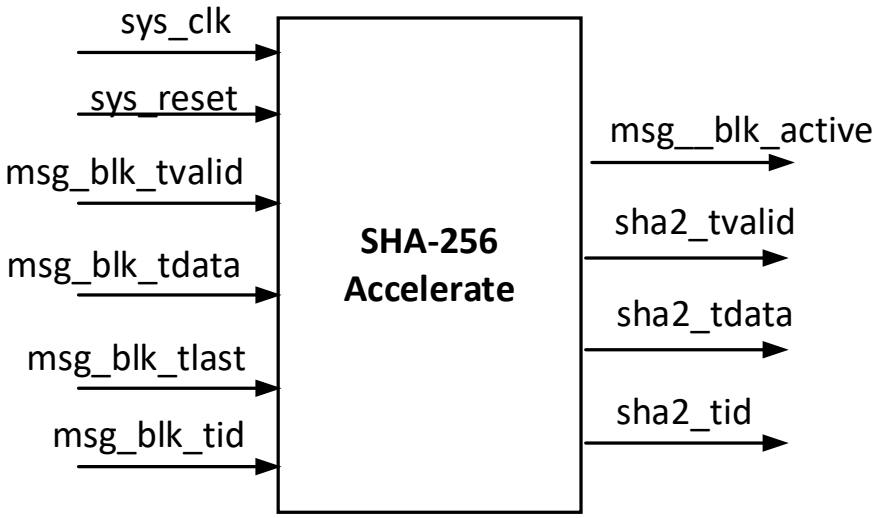

图 5.2 SHA-256 摘要值计算模块的输入输出端口图

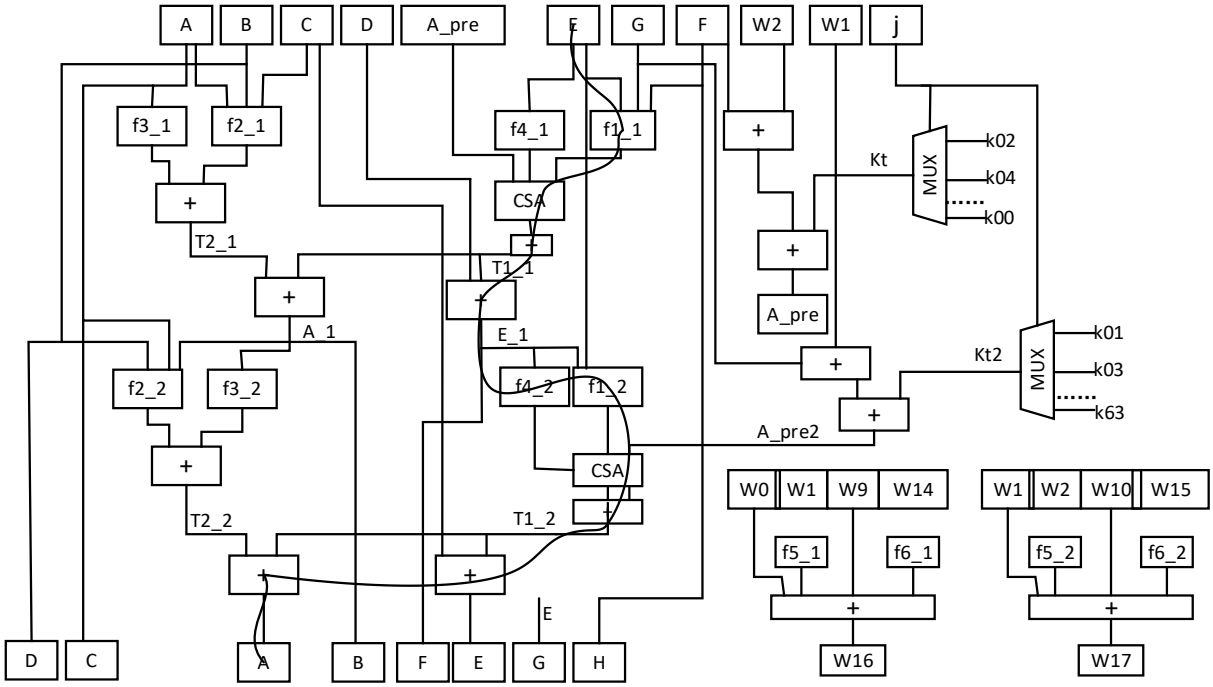

图 5.3 SHA-256 运算单元数据通路

其中黑色部分为关键路径，A、B、C、D、E、F、G、H、W1…W15、A_pre 均为 32 位寄存器，j 为 5 位寄存器，W1…W15 在每个时钟周期左移 64 位。

SHA-256 散列值计算模块的功能是进行 MD5 算法的循环计算，即接收预处理模块输出的数据 msg_blk_tdata，对应的数据位宽为 512-bit，输入数据通过压缩计算过程最终输出 256-bit 的散列值。压缩计算过程：先将 512-bit 的数据块 msg_blk_tdata 扩展成为 64*32 比特位的数据，同步初始化 32-bit 的寄存器 a、b、c、d、e、f、g、h 值，再进入循环体进行 64 轮计算后，更新寄存器 a、b、c、d、e、f、g、h 的值，直到计算完最后一组 512-bit 数据，级联输出散列值计算结果。

设计计算模块经历四个阶段，为接收输入数据 A，开始处理数据 B，数据计算 C 和计算完成输出 D。如图 5.4 所示为 SHA-256 散列值计算模块的流程图。

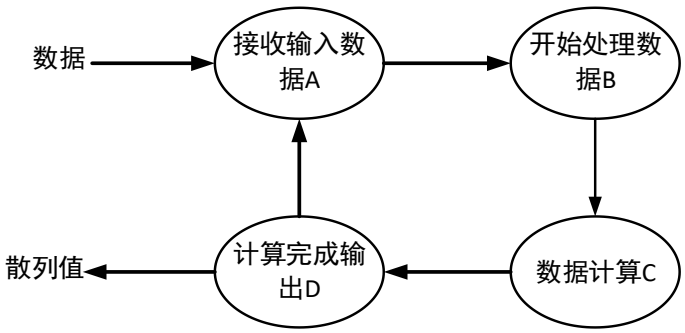

图 5.4 SHA-256 散列值计算模块的流程图

其中接收输入数据阶段 A 为一个空闲的理想阶段，也是整个流程中的初始状态，状态跳转条件是有效数据输入；

开始处理数据阶段 B 在接收到有效的输入数据时占用数据运算处理通道即 msg_blk_active 为 1，同时拉高数据开始处理标志信号。状态跳转条件为下一个时钟周期跳转；

数据计算阶段 C 是完成数据块的扩展成为 64*32bits 的数据，进行 SHA-256 算法中的 64 轮计算，计算完成后拉高计算完成标志信号。状态跳转条件是 64 轮计算完成；

计算完成输出阶段 D 是在算法完成 64 轮计算后对得到散列值进行一个处理。如果同一明文消息还有有效数据输入则跳转到接收输入数据阶段 A 进行接收输入数据，如果同一明文消息没有有效数据输入则输出计算得到的散列值，同时自动跳转到接收输入数据阶段 A 进行接收输入数据。

整个流程解释：当未开启消息加密功能即无数据输入时，模块时钟处于接收输入数据阶段 A，一旦有数据输入时则进行状态转变，状态转变到下一个阶段即开始处理数据阶段 B，在该阶段中，完成链接变量 a、b、c、d、e、f、g、h 初始化赋值。在下一个时钟周期到来时，状态自动转变到数据计算阶段 C，在该阶段中，完成数据扩展以及循环体计算，若已完成循环体的运算，自动进入下一计算完成输出阶段 D，否则流程将处于数据计算阶段 C。在计算完成输出阶段 D 时，如果同一明文消息还有有效数据输入则在下一个时钟周期到来时，状态跳转又回到最初的接收数据输入阶段 A 中。如果同一明文消息没有有效数据输入，则将经过 SHA-256 算法运算的数据进行输出，同时释放通道占用信号 msg_blk_active，这些操作表示 SHA-256 算法散列值计算完成，输出有效散列值，在下一个时钟周期到来时，状态跳转又回到最初的接收输入数据阶段 A。

5.4 本章小结

本章首先对设计的 SHA-256 算法模块架构进行简单介绍，其中着重介绍散列值计算模块的功能、原理框图、输入输出端口信号意义、算法实现时的表征标志信号等内容，详细介绍 SHA-256 计算模块流程和 SHA-256 算法的硬件实现过程。

第6章 SHA-256 算法的仿真与性能分析

6.1 SHA-256 功能仿真

SHA-256 算法电路的测试方案与 MD5 算法电路的测试方案基本一致，此处不做详细阐述。SHA-256 算法模块实现是在 Windows 操作系统下，利用 Quartus II 附带使用 QuestaSim 仿真器对生成散列值的计算模块进行功能验证。

6.1.1 单个消息块仿真验证

对消息长度不足 448 的明文信息进行 SHA-256 算法运算，验证映射得到的 256-bit 的散列值。举例验证输入的消息明文信息 Mx 字符为“abc”，利用十六进制表示为“616263”，验证字符串“abc”经过 SHA-256 算法运算后的结果。

1. 输入数据预处理模块完成预处理操作。

第一步将明文信息由 ASCII 码转换成二进制或者十六进制表示，为便于浏览转换成十六进制的值为 616263。

第二步对数据进行填充，由于明文信息数据长度为 3*8 即 24 个比特位，而且 24 除以 512 余 24，此值并不等于 448，要对明文信息进行填充，填充方式是在明文数据后面填充 1 比特位的二进制数“1”和 432 比特位的二进制数“0”。

第三步对数据进行扩展，即补齐代表明文长度信息的数据，预处理完成后的输入数据以及整个预处理操作如图 6.1 所示。

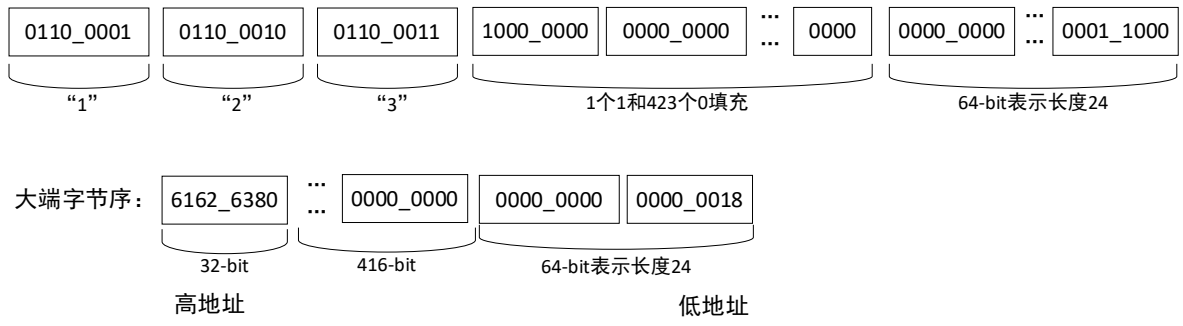

图 6.1 “abc”数据填充排序图

2. 当预处理操作完成后将填充后的数据 msg_blk_tdata 输出，并在同一时钟周期生成对应的控制信号，即拉高对应的数据有效信号 msg_blk_tvalid，由于仿真验证是对单个消息块仿真验证，对应的表征信号 msg_blk_tlast 在同一时钟周期上一定拉高为高电平；输入数据占用的通道为 15 即消息分组的通道编号 msg_blk_tid 为 15，输入到 MD5 哈希值计算模块的信号如图 6.2 所示。

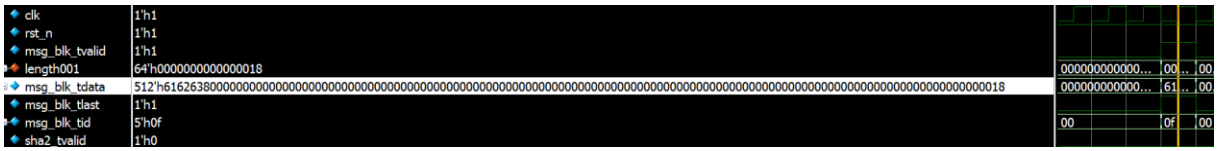

图 6.2 “abc”预处理的输入数据

3. 已知利用在线转换器得到的 SHA-256 算法运算散列值为：ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad，得到的结果如图 6.3 所示。图 6.4 所示为”abc”生成散列值的仿真图。由图 6.3 以及图 6.4 可得，模块设计实现的仿真结果与 SHA-256 算法在线转换器得到的最终计算结果一致，SHA-256 散列值计算模块的逻辑功能验证正确。

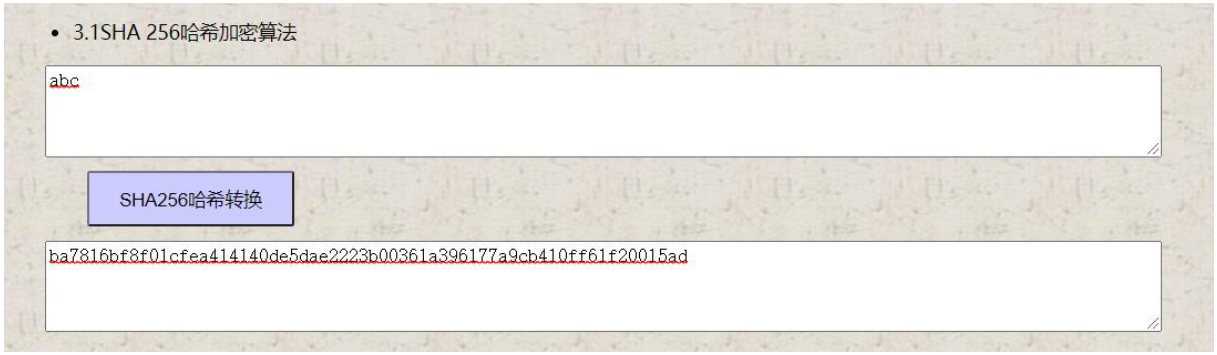

图 6.3 “abc”的 SHA-256 在线转换器值

图 6.4 “abc”生成的散列值

首先完成需加密数据的预处理操作，需加密数据为明文信息，此操作在输入数据预处理模块完成。如果需加密的明文消息数据长度为 512-bit，则经前端数据预处理后得到 2*512-bit 数据，即输入数据按 2 次对同样通道进行输入，计算模块也要对此消息进行 2 次散列值计算处理，再级联数据输出 256-bit 散列值，由于前端数据预处理模块完成了明文消息的预处理，输入给计算模块的数据是已经预处理好的 2*512 比特位数据，则在第一组输入数据输入到计算模块时，初始化链接变量，且寄存下来并在第一组计算完成后进行数据处理，得到一个散列值的暂存值。紧接着接收第二组输入数据，并此时链接变量更新为上一次哈希计算生成的输出数据，在第二组数据完成 64 轮计算后，输出散列值，此值就是输入的明文数据加密得到的散列值。由图 6.4 可知，仿真图中数据与 SHA-256 算法预期结果完全相同，证明 SHA-256 算法硬件实现的正确性。

6.1.2 多个消息块仿真验证

对于明文长度大于或者等于 448 比特位的明文信息，经过预处理操作后输入到 SHA-256 散列值计算模块进行 SHA-256 算法运算。通过举例验证输入明文信息 My，验证经过 SHA-256 算法运算后的结果是否与 SHA-256 加密算法转换器得到的结果一致。

若验证输入的消息明文信息 My 字符串为“abcdabcdabcdabcd_ abcdabcdabcdabcd_abcdabcdabcdabcd_ abcdabcdabcdabcd_abc”，利用十六进制表示为:“61626364_61626364_61626364_61626364_61626364_61626364_61626364_61626364_61626364_61626364_61626364_61626364_61626364_61626364_61626364_61626364_616263”，验证字符串经过 SHA-256 算法运算后的结果。

1. 输入数据预处理模块完成预处理操作

先对明文信息进行数据填充,再对数据进行扩展，完成预处理操作输出数据和对应的控制信号如图 6.5、图 6.6 所示。

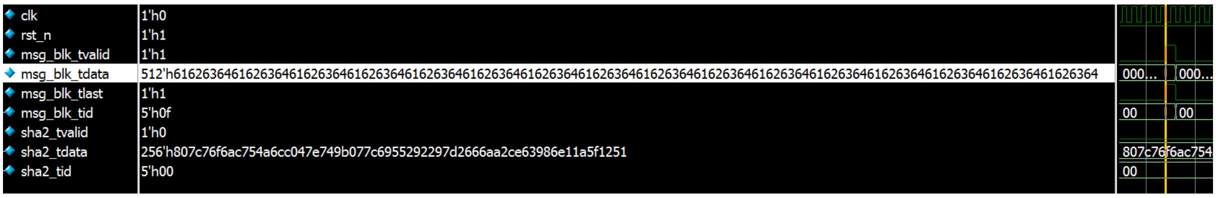

图 6.5 字符串 My 预处理后的输入数据第一组

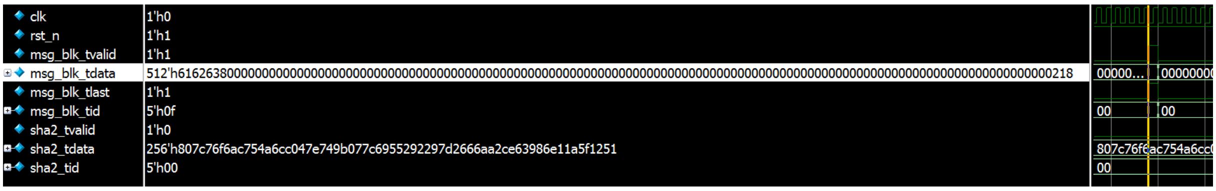

图 6.6 字符串 My 预处理后的输入数据最后一组

当预处理操作完成后输出两组 512-bit 的输入数据 msg_blk_tdata，拉高表征输入数据有效的信号即此时 msg_blk_tvalid 为 1，输入数据占用的通道为 15 即消息分组的通道编号 msg_blk_tid 为 15，对应的表征信号 msg_blk_tlast 在第一组数据块输入时为低电平状态，最后一组数据块输入时为高电平状态。同一消息只能在一个数据通道中进行处理，直至这一消息的数据块全部循环迭代计算完，输出对应的 256-bit 散列值，释放 msg_blk_tid 为 10 的数据通道，以供其他消息输入。

2. SHA-256 算法运算得到仿真结果

输入数据后经过一系列循环迭代运算，接收最后一组数据块信息进行 SHA-256 算法运算，历经 700ns 即 35 个时钟周期后得到 SHA-256 哈希值即散列值 MDy0，如图 6.7 所示，仿真结果利用的大端字节序规则显示。

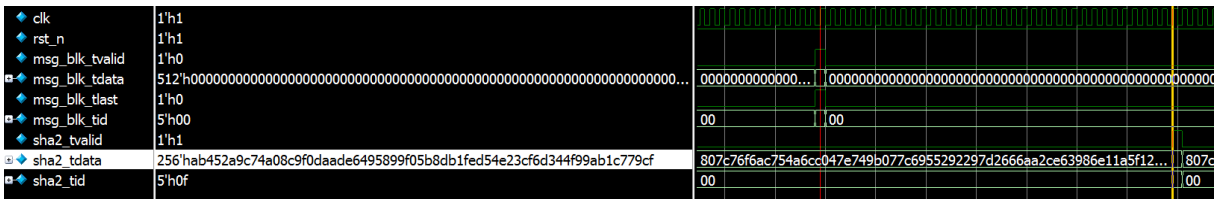

图 6.7 字符串 My 生成散列值 MDy0 仿真图

3. 对比数据结果

利用 SHA-256 算法在线转换器，得到的结果如图 6.8 所示。转换器的结果是以大端字节序显示。将输入相同的明文消息 My 进行 SHA-256 算法转换为 256 比特的摘要值 MDy1 与将明文消息 My 经过模块设计的 SHA-256 算法运算实现得到散列值 MDy0 的结果进行比较，可得数据相等，即 MDy0=MDy1，故证明设计的 SHA-256 算法运算的正确性。

图 6.8 字符串 My 的 SHA-256 在线转换器值

6.2 SHA-256 算法性能分析

SHA-256 算法同 MD5 算法一样采用 Verilog HDL 设计，并在 Altera Cyclone-V FPGA 开发板上进行实现，SHA-256 采用数据预处理的方式和流水线设计大大提高了数据运算效率。

表 6.1 SHA-256 性能对比

设计	最高频率/MHz	寄存器资源/个	吞吐量/Mbps
文献[61]	64.45	1251	118
文献[62]	135.3	26340	69270
文献[63]	172	10240	11010
本设计	173.8	12923	81511

表 6.1 列出了本设计与其它同类设计的性能对比，文献[61]采用四输入的 ALUs 进行 SHA-256 算法运算，文献[63]、文献[62]和本文一样采用流水线设计的

方式，可以看出采用这种方式可以在很大程度上提高数据吞吐率。本设计不仅使用了流水线技术，还使用预处理和缩短关键路径的技术，虽然本设计也采用的是流水线设计，但是吞吐率却比文献[62]和文献[63]的吞吐率高的多。

最后优化实现的 MD5 和 SHA-256 算法应用于一款密码安全芯片，图 6.9 为该芯片的版图，采用 0.18μm 工艺进行 MPW 流片，该芯片面积为 6mm^2,工作电压 3.3V，时钟频率为 150MHz 时，功耗约为 10.7mW，测试结果表明，该设计功能正确。

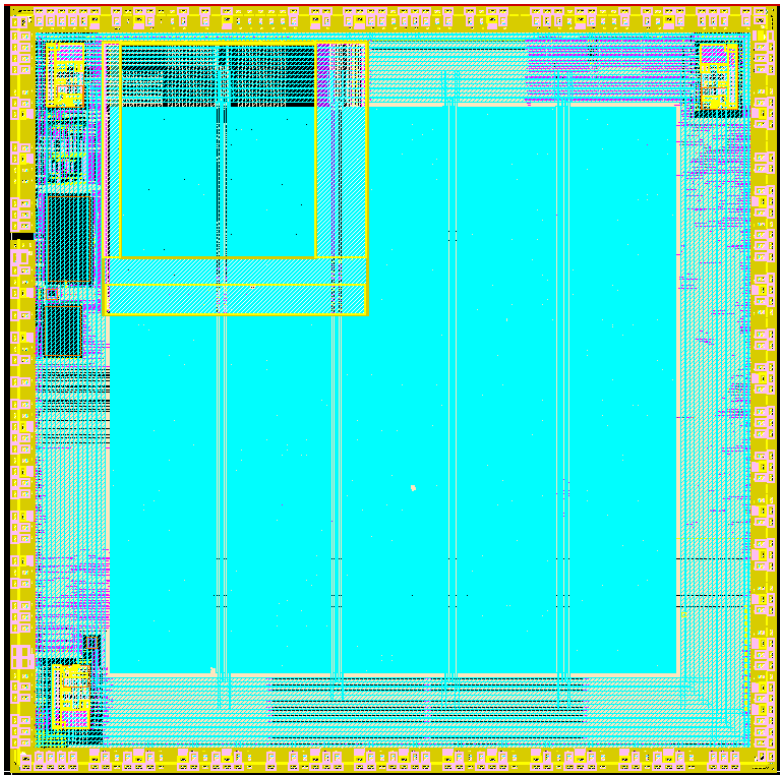

图 6.9 芯片版图

6.3 本章小结

本章主要介绍了 SHA-256 算法的功能仿真及结果分析，利用仿真结果与在线转换器得到的数据加密结果进行对比，验证 SHA-256 算法的正确性。最后将本设计与现有的文献进行性能对比，并给出芯片版图。

总结与展望

生活在互联网高速发展的信息化时代，信息作为信息化时代的主要载体，保护信息安全十分重要。哈希算法作为信息安全和密码学领域的重要算法之一，MD5 加密算法和 SHA-256 散列算法在数据加密、信息安全传输领域占有重要地位。

本文详细介绍了 MD5 加密算法和 SHA-256 算法原理，为解决哈希算法实现存在执行时间长、数据吞吐量低、硬件资源占用率高的问题，提出了对于节省硬件资源提高数据吞吐量的优化方案，完成 MD5 算法和 SHA-256 算法模块整体架构以及对应子模块设计。再利用 Quartus II 和 QuestaSim 进行仿真与功能分析，设计通过 Altera Cyclone-V FPGA 开发板进行硬件实现与验证，实现 MD5 和 SHA-256 算法达到的可运行最高时钟频率可达到 173MHz，数据吞吐量超过 81Gbps。最后该设计采用 0.18μm 工艺进行 MPW 流片，测试结果表明该设计满足性能指标。本文的主要工作为：

1. 根据 MD5 加密算法和 SHA-256 算法原理，完成 MD5 加密算法的架构设计，进行硬件实现。且在 MD5 加密算法硬件实现的基础上完成 SHA-256 算法的硬件实现，提高了设计的灵活度。

2. 对硬件实现的 MD5 加密算法和 SHA-256 散列算法进行仿真验证，对比仿真结果与在线转换器结果，硬件实现设计 MD5 加密算法和 SHA-256 散列算法满足性能要求。

其中主要创新点为：

1. 提出一种优化循环移位方案。由于 F、G、H、I 轮运算中循环移位的位数是固定的 4 个数字，故直接将移位的结果作为加法器的加数相加替代了 32-bit 数据循环移位操作，从而加快了数据运算速度。

2. 提出了优化加法器设计。利用三级加法器替代四级加法器方式提前计算 FF、GG、HH、II 函数的运算结果，极大提高了系统时钟频率，另外三级加法器面积更小，可以降低整体硬件资源的消耗。

3. 采用 32 级流水线设计，将算法 64 次循环迭代 2 次循环合为 1 次，只需要进行 32 次循环运算，插入中间寄存器，将中间数据进行暂存，提高了运算速度。

从总体上看，本文所设计的 MD5 加密算法模块以及 SHA-256 算法模块在执行速度以及数据吞吐量上取得了较大提升。本设计还存在进一步提升的空间，但由于项目紧张无法完成更多的工作，以下是对下一步研究工作的建议：

1. 目前 MD5 已经有很多的变种算法，这些变种算法相比于传统的 MD5 算法有更高的安全性，所以可以考虑选择一种合适的变种算法，同安采用流水线的设

计方案，以获得更好的性能，并提升算法安全性；

2. 可以对加法器进行优化，可使用更加高性能、高效能的加法器，例如设计时采用超前进位加法器，缩短运算过程的延时，提高运算效率，但是此加法器一定会造成硬件资源上的消耗，这一点需要注意和权衡；

3. 对于算法架构设计，追求做到简单实现、复用性高，但是一个算法优化完善程度，并不是简单地根据时钟频率、数据吞吐量、硬件资源等性能指标决定，还要看成本、应用产业的需求等，根据这些研究出一个最佳方案。

参考文献

[1] HE D,ZHI X.Multi-parallel Architecture for MD5 Implementations on FPGA with Gigabit-level Throughput.In:proceedings of the International Symposium on Intelligence Information Processing & Trusted Computing. Huanggang, 2010,535-538

[2] Ammar Mohammed Ali,Alaa Kadhim Farhan.A Novel Improvement with an Effective Expansion to Enhance the MD5 Hash Function for Verification of a Secure E-Document. IEEE Access,2020,PP(99):1-1

[3] Setiadi D,Najib A F,Rachmawanto E H,et al.A Comparative Study MD5 and SHA1 Algorithms to Encrypt REST API Authentication on Mobile-based Application.In:International Conference on Information and Communications Technology (ICOIACT).Yogyakarta,2019,206-211

[4] Feng F,Li X,Wang L.Design and implementation of identity authentication system based on fingerprint recognition and cryptography.In:proceedings of the 2016 2nd IEEE International Conference on Computer and Communications (ICCC).Chengdu,2016,254-257

[5] Akhavan A,A Samsudin,A Akhshani.A novel parallel Hash function based on 3D chaotic map.EURASIP Journal on Advances in Signal Processing. 2013,2013(1):126

[6] Al-Odat Z A-H.Analyses,Mitigation and Applications of Secure Hash Algorithms:[North Dakota State University PHD Dissertation].Ann Arbor:North Dakota State University,2020,1-251

[7] 林鑫.云计算环境下国家学术信息资源安全保障机制与体制研究:[武汉大学博士学位论文].武汉:武汉大学,2016,1-212

[8] Bhonge H N,Ambat M K,Candavarkar B R.An Experimental Evaluation of SHA-512 for Different Modes of Operation.In:proceedings of the 2020 11th International Conference on Computing,Communication and Networking Technologies(ICCCNT).Kharagpur,2020,1-6

[9] Hassen M,Kahri F,Bouallegue B,et al.Efficient FPGA Hardware Implementation of Secure Hash Function SHA-2.International Journal of Computer Network and Information Security,2014,7(1):9-15

[10] Jacobson S R.A method for performance verification of freeware HASH utitlies

using MATLAB:[niversity of Colorado at Denver Master Dissertation].Ann Arbor:University of Colorado at Denver,2013,1-128

[11] Yang B.Design and test for high speed cryptographic architectures:[Polytechnic Institute of New York University Master Dissertation].Ann Arbor:Polytechnic Institute of New York University,2009,1-145

[12] Baldanzi L,Crocetti L,Falaschi F,et al.Cryptographically Secure Pseudo-Random Number Generator IP-Core Based on SHA2 Algorithm.Sensors,2020,20(7):1869

[13] Tautvydas B.Analysis And Implementation Of Cryptographic Hash Functions In Programmable Logic Devices.Mokslas - Lietuvos ateitis,2016,8(3):321-326

[14] Padma B,Kumar G R.A Novel Approach to Thwart Security Attacks on Mobile Pattern Authentication Systems.International Journal of Computer Network and Information Security,2018,10(5):18-27

[15] Pich R,Chivapreecha S,Prabnasak J.A single triple chaotic cryptography using chaos in digital filter and its own comparison to DES and triple DES.In:proceedings of the 2018 International Workshop on Advanced Image Technology(IWAIT).Chiang Mai,2018,1-4

[16] 葛灿.基于 GPU 的 SHA-2 哈希算法的快速实现及应用:[上海交通大学硕士学位论文].上海:上海交通大学,2018,1-88

[17] Bergstra J A,Middelburg C A.Instruction sequence expressions for the secure hash algorithm SHA-256.Ithaca,Cornell University Library,arXiv.org.2013,1-14

[18] Pushpendra M,Tomar S,Shreevastava D M.The Study of Detecting Replicate Documents Using MD5 Hash Function.International Journal of Advanced Computer Research,2011,6(2):14-17.

[19] Kyoungsoo B,Yunjeong L,Junho P,et al.An Energy-Efficient Secure Scheme in Wireless Sensor Networks.Journal of Sensors,2016,1-11

[20] Tomar M,Shreevastava M.Result Analysis and Benefits of Detecting Replicate Documents Using MD5 Hash Function.International Journal of Advanced Computer Research,2011,1(2):21-26

[21] Erritali M,Oussama Mohamed R,Bouabid El O.A Contribution to Secure the Routing Protocol "Greedy Perimeter Stateless Routing" Using a Symmetric Signature-Based AES and MD5 Hash.International Journal of Distributed & Parallel Systems,2011,95-103

[22] Zhu S,Zhu C,Wang W.A New Image Encryption Algorithm Based on Chaos and Secure Hash SHA-256.Entropy,2018,20(9):1-18

[23] Khoa T A,Le M,Son H H,et al.Designing Efficient Smart Home Management with

IoT Smart Lighting:A Case Study.Wireless Communications and Mobile Computing,2020,2020:1-18

[24] Deepakumara J,Heys H M,Venkatesan R.FPGA implementation of MD5 Hash algorithm.In:proceedings of the Conference on Electrical & Computer Engineering.Toronto:IEEE,2001,919-924

[25] Jarvinen K,Tommiska M,Skytta J.Comparative survey of high-performance cryptographic algorithm implementations on FPGAs.Information Security Iee Proceedings,2005,152(1):3-12

[26] Hoang A T,Yamazaki K,Oyanagi S.Multi-stage Pipelining MD5 Implementations on FPGA with Data Forwarding.In:proceedings of the 16th IEEE International Symposium on Field-Programmable Custom Computing Machines. IEEE:Stanford,2008,271-272

[27] Wang Y,Zhao Q,Jiang L,et al.Ultra High Throughput Implementations for MD5 Hash Algorithm on FPGA.Second International Conference,2010,1-8

[28] 王臣,袁炎.超高吞吐量 MD5 算法的 FPGA 实现.信息技术,2011,35(9):55-61

[29] HAN J S,LIN J J,YE J W,et al.Design of MD5 High-Speed Models on FPGA.Transactions of Beijing Institute of Technology,2012,32(12):1258-1268

[30] 谭健,周清雷,斯雪明等.全流水架构 MD5 算法在拟态计算机上的实现及改进.小型微型计算机系统,2017,38(06):1216-1220.

[31] Ramirez N N,Londoo R.Implementación hardware de la función Hash SHA3-256 usando una arquitectura Pipeline.Ingeniare Revista Chilena de Ingenieria, 2019,27(1):43-51

[32] Deepakumara,Janaka T.Hardware implementation of message authentication algorithms for Internet security:[Memorial University of Newfoundland Master Dissertation].Ann Arbor:Memorial University of Newfoundland,2002,1-176

[33] KHAN E A H.Design and performance analysis of a reconfigurable,unified HMAC-Hash unit for IPSec authentication:[University of Victoria PHD Dissertation].Ann Arbor:University of Victoria,2006,1-157

[34] 吕雪,沈斌等.基于 ATSHA204A 的程序保护系统研究.江苏科技信息,2017,000(024):49-54

[35] Lee E H,Lee J H,Park I H,et al.Implementation of high-speed SHA-1 architecture.Ieice Electron Express,2009,6(16):1174-1179

[36] Kayalvizhi R,Subramanian R H,Santhosh R G,et al.VLSI Design and Implementation of Combined Secure Hash Algorithm SHA-512.Springer Berlin Heidelberg,2010,1-64

[37] Michail H E,Athanasiou G S,Theodoridis G,et al.On the development of high-throughput and area-efficient multi-mode cryptographic Hash designs in FPGAs.Integration the Vlsi Journal,2014,47(4):387-407

[38] Michail H E,Kakarountas A P,Milidonis A S,et al.A Top-Down Design Methodology for Ultrahigh-Performance Hashing Cores.IEEE Transactions on Dependable & Secure Computing,2009,6(4):255-268

[39] Athanasiou G S,Michail H E,Thodoridis G,et al.Optimising the SHA-512 cryptographic Hash function on FPGAs.IET Computers & Digital Techniques,2013,8(2):70-82

[40] Michail H E,Athanasiou G S,Kelefouras V I,et al.Area-Throughput Trade-Offs for SHA-1 and SHA-256 Hash Functions Pipelined Designs.Journal of Circuits,Systems and Computers,2016,25(4):1650032.1-1650032.26

[41] Zhang Y,Kim J,Choi K,et al.High Performance and Low Power Hardware Implementation for Cryptographic Hash Functions.International Journal of Distributed Sensor Networks,2014,1-12

[42] 刘坚.哈希算法复用IP核的SOC实现及抗攻击设计:[辽宁大学硕士学位论文].沈阳:辽宁大学,2014,9-18

[43] Kessler G.The Impact of MD5 File Hash Collisions On Digital Forensic Imaging.The Journal of Digital Forensics,Security and Law : JDFSL, 2016,11(4):129-318.

[44] 刘曼春.基于MD5改进算法的安全教师博客系统设计及开发[湖南大学硕士学位论文].长沙:湖南大学,2013,1-85

[45] Oza S,Matuszewski U,Jessa M.A Random Number Generator Using Ring Oscillators and SHA-256 as Post-Processing.International Journal of Electronics and Telecommunications,2015,61(2):1-4.

[46] Safarinia M,Garshasebi M H,Pourmahdi P,et al.The Parallel One-way Hash Function Based on Chebyshev-Halley Methods with Variable Parameter.International Journal of Computers,Communications and Control,2014,9(1):24-36.

[47] Selvakumar A,Ratastogi R S.Study the function of building blocks in SHA Family.Ithaca Cornell University Library,2014,1-6

[48] Preneel B.Cryptographic Hash Functions.Transactions on Emerging Telecommunications Technologies,2012,5(4):431-448

[49] Ali A M,Farhan A K.A Novel Improvement with an Effective Expansion to Enhance the MD5 Hash Function for Verification of a Secure E-Document.IEEE

Access,2020,PP(99):1-1

[50] Khan S A.FPGA Implementation of MD5 Algorithm for Password Storage.International Journal of Science and Research (IJSR),2013,136-139

[51] Aziz M V G,Wijaya R,Prihatmanto A S,et al.HASH MD5 function implementation at 8-bit microcontroller.In:2013 Joint International Conference on Rural Information & Communication Technology and Electric-Vehicle Technology(rICT & ICeV-T).Bandung,2013,1-5.

[52] ZheLong W,Yan D,Qing W,et al.Research on software comparison of electric energy data acquire terminal based on MD5 algorithm.In:2017 Chinese Automation Congress(CAC).Jinan,2017,1-5.

[53] Setiadi D,Najib A F,Rachmawanto E H,et al.A Comparative Study MD5 and SHA1 Algorithms to Encrypt REST API Authentication on Mobile-based Application.In:2019 International Conference on Information and Communications Technology(ICOIACT).Yogyakarta,2019,1-6

[54] Ratna A,Purnamasari P D,Shaugi A,et al.Analysis and comparison of MD5 and SHA-1 algorithm implementation in Simple-O authentication based security system.In:2013 International Conference on QiR.Yogyakarta,2013,1-3

[55] Lee S,Shin K.An efficient implementation of SHA processor including three Hash algorithms(SHA-512,SHA-512/224,SHA-512/256).In:2018 International Conference on Electronics,Information,and Communication(ICEIC). Honolulu, 2018,1-4

[56] Zhang X,R W U,WANG M,et al.A High-Performance Parallel Computation Hardware Architecture in ASIC of SHA-256 Hash.In:2019 21st International Conference on Advanced Communication Technology(ICACT). PyeongChang, 2019,1-5

[57] Omran S S,Jumma L F.Design of SHA-1 & SHA-2 MIPS processor using FPGA.In:2017 Annual Conference on New Trends in Information & Communications Technology Applications(NTICT).Baghdad,2017,1-6

[58] Omran S S,Jumma L F.Design of multithreading SHA-1 & SHA-2 MIPS processor using FPGA.In:2017 8th International Conference on Information Technology(ICIT).Amman,2017,1-6

[59] JianHua H,Hu C,HuaQing H.A compatible SHA series design based on FPGA.In:proceedings of the ECTI-CON2010 The 2010 ECTI International Confernce on Electrical Engineering,Telecommunications and Information Technology,Chiang Mai,2010,380-384

[60] Wang B T,Han G D,Zhang X J.Design and Implementation of MD5 Algorithm

Based on FPGA.Communications Technology,2010,1-6

[61] R García,Algredo-Badillo I,Morales-Sandoval M,et al.A compact FPGA-based processor for the Secure Hash Algorithm SHA-256.Computers & Electrical Engineering,2014,40(1):194-202.

[62] Michail H E,Athanasiou G S,Kelefouras V I,et al.Area-Throughput Trade-Offs for SHA-1 and SHA-256 Hash Functions Pipelined Design.Journal of Circuits,Systems and Computers,2016,25(4):1-35

[63] Michail H E,Athanasiou G S,Kelefouras V,et al.On the exploitation of a high-throughput SHA-256 FPGA design for HMAC.Acm Transactions on Reconfigurable Technology & Systems,2012,5(1):1-28.

附录 A　攻读学位期间取得的研究成果

[1] 李妮.基于安全 Hash 算法的动态加密管控系统.著作权登记号：2021SR0516318

致谢

时光荏苒，光阴如梭。作为一名湖南大学的硕士研究生，三年求学生涯即将结束，但是期间读过的书、走过的路、遇到的人都将是一生中最宝贵的财富。三年光景不待人，须臾发已疏，感谢那些在求学路上帮助过我的人是您们的帮助让我非常心满意足地充实地度过这三年时光,是您们的帮助让我能够顺利完成学业。

首先万分感谢我的导师王镇道老师。王老师，兢兢业业，对困难和挑战甘之如饴；道法自然，顺应时代之大潮；薪火相传，尽师职，传道授业解惑；志存高远，踏凌云，诠释青年担当。研究生求学期间，王老师不仅教会了我很多专业方面的知识，而且给我点播，引导一步步探索一条适合自己的研究方向，更重要的是教会了很多做人的道理，以目标为导向，踏踏实实做好当下之事，行好当下的路。本次论文的开题、撰写以及定稿都是在王老师的反复指导帮助下完成的，真的万分感谢王老师一路以来的帮助和指导。

再者特别感谢中科院计算所卢文岩博士以及中科驭数硬件组全体同仁。卢博气宇不凡，才气超然，给人稳若泰山之感。各位同事工作上认真刻苦，焚膏继晷，一直奋斗在第一线。在我联合培养期间，大家无私地教会我许多专业技能，让我在较短时间内提升专业基本能力，迅速地融入集体，并且愉快地度过联培一年时光。这段学习之旅给我未来规划提供了更多的可能。

同时特别感谢陈迪平老师、胡锦老师、曾健平老师、晏敏老师、廖蕾老师、秦志辉老师、赵楚军老师、杨红官老师、朱小莉老师等所有物电院的老师们耐心教导和悉心帮助，打开专业深入大学之门。再者感谢李庭予、韩晓明等辅导员以及教研工作的老师们，是你们让我们入学、求学、毕业之路有条不紊地进行。

其次感谢我的同学李勇彬、刘剑在论文修改上给予了我莫大的帮助。感谢赵阳凡、金邦兴、江中玉、姚小姣、宁颖、谭泽军、何琼、戴雨薇、唐吕攀、吴飘、沈明保等人的陪伴，想念研究生三年一起学习、一起游乐、一起聊八卦的时光。感谢谭卓智、李勇彬同学一同前往北京,怀念大家一起登八达岭长城的快乐时光。不知来岁牡丹时，再相逢何处，愿大家前程似锦、一切顺遂！感谢师兄方正、张一鸣、陈梦亮，感谢曾凌峰、李政、王振宇等师兄，一路以来的关心和指导。

最后感谢我的父母、爷爷奶奶等家人，感谢你们多年的关心关注，你们是我最强有力的后盾，是我奋发向上的源动力。

无可奈何花落去，珍惜当下才是真。愿大家健康平安、万事如意！贯彻湖南大学“实事求是 敢为人先”的校训理念，及时当勉励，岁月不待人。让我们一起努力前进，创造自身价值，功成名就时归来仍是那位少年！