电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

专业学位硕士学位论文

MASTER THESIS FOR PROFESSIONAL DEGREE

论文题目 **数据认证加密 SoC 的设计及其实现**

专业学位类别	电子信息
学　　号	**202052021527**
作者姓名	胡浩
指导教师	彭析竹　　研究员
学　　院	集成电路科学与工程学院

分类号 TN4 密级 公开

UDC[注1] 621.39

学 位 论 文

数据认证加密 SoC 的设计及其实现

（题名和副题名）

胡浩

（作者姓名）

指导教师 彭析竹 研究员

电子科技大学 成 都

向多春 高级工程师

成都振芯科技股份有限公司 成都

（姓名、职称、单位名称）

申请学位级别 硕士 专业学位类别 电子信息

专业学位领域 集成电路工程

提交论文日期 2023 年 3 月 31 日 论文答辩日期 2023 年 5 月 11 日

学位授予单位和日期 电子科技大学 2023 年 6 月

答辩委员会主席

评阅人

注 1：注明《国际十进分类法 UDC》的类号。

Design and Implementation of SoC for Data Authentication and Encryption

A Master Thesis Submitted to

University of Electronic Science and Technology of China

Discipline **Electronic Information**

Student ID **202052021527**

Author **HuHao**

Supervisor **Prof. PengXizhu**

School **School of Intergrated Circult Science and Engineering**

摘　要

随着当今社会信息的不断发展，信息安全传输问题也越来越重要，安全密码算法为数据的安全传输带来了有效的解决方案。本文的研究成果主要应用于汽车电子等设备中，属于消费电子类别。硬件实现的安全密码算法由于其安全性能更高，运行速度更快，灵活性更好而被广泛应用于不同应用场景。SoC 芯片技术由于其具有芯片尺寸小，功耗低，可靠性强，运行速度快，可编程性高，运行速度快等优势已经成为集成电路领域的核心技术。本文将设计的安全密码算法结合常用外设接口 IP 最后集成数据认证加密 SoC 芯片，可用于数据认证加密处理领域对数据的安全传输起到重要作用。

本文的具体研究内容如下：

(1)介绍了 ARM Cortex-A7 的基本知识及其架构和需要用到的总线的基础知识，然后通过对芯片进行功能分析，确定了选取 ECDH 算法来进行秘钥协商，选取 HMAC-SHA256 算法来对数据进行完整性认证，选取 AES-CTR 算法来对数据进行加解密，并且设计了芯片的总线架构和总体结构，最后完成 SoC 的系统集成。

(2)对选取的安全密码算法进行原理介绍，并进行设计和结果仿真。在用硬件设计算法时，针对功能需求，采取一定的措施对各安全密码算法进行逻辑优化。

(3)结合软件调试环境对 SoC 进行 FPGA 原型验证。结果表明 SoC 正常启动且 ECDH,HMAC-SHA256,AES-CTR 算法模块功能正确。

关键词：数据认证加密，SoC，密码算法，ARM A7

ABSTRACT

With the continuous development of information in today's society, the problem of information security transmission is becoming more important. Security cryptography algorithm has brought effective solutions for the safe transmission of data. The research results of this thesis are mainly applied to automotive electronics and other equipment, belonging to the category of consumer electronics. The security cryptographic algorithms implemented by hardware are widely used in different scenarios because of their higher security performance, faster running speed and better flexibility. SoC chip technology has become the core technology in thein the field of integrated circuits because its advantages of small chip size. low power consumption, strong reliability, fast running speed and so on. In this thesis, the design of the security cryptographic algorithm combined with the common peripheral interface IP finally integrated data authentication and encryption SoC chip, which can be used in the field of data authentication and encryption processing to play an important role in the security of data transmission.

The specific research content of this thesis is as follows:

（1）Introduced the basic knowledge of ARM Cortex-A7 and its architecture and the basic knowledge of the bus needed,and then through the functional analysic of the chip, selecting the ECDH algorithm for key negotiation, selecting the HMAC-SHA256 algorithm for the data integrity authentication, sececting the AES-CTR algorithm for data encryption and decrypyion, and then design the bus architecture and the overall structure of the chip, and finally complete the SoC system integration.

（2）the principle of the selected security password algorithm is introduced, and the design and result simulation. In the hardware design algorithm, according to the functional requirements, some measures are token to optimize the logic of eath security password algorithm.

（3）Combined with the software debugging environment to carry out FPGA prototype verification of SoC, The results show that SoC starts normally and ECDH, HMAC-SHA256 and AES-CTR algorithm modules function correctly.

Keywords: Data authentication and encription, SoC, cryptographic algorithms, ARM A7

目　录

图目录

表目录

第一章 绪论

1.1 研究背景及意义

随着社会的进步和科学技术的飞速发展，我们已经步入了大数据时代，随着日益增加的信息量，人们对数据安全的要求也越来越高，社会存在的信息安全问题也显得尤为重要。安防监控，智能医疗和自动驾驶等行业对安全通信数据安全要求也越来越高[1]。密码是一种保护自身隐私，财产，信息以及重要数据的工具，但不法分子通过技术手段破解密码，偷窥和盗取他人重要信息，资料及数据，导致资料和财产损失的事件时有发生，密码学技术的发展水平对保障信息安全至关重要，关系到社会稳定，经济发展以及国家安危[2]。

密码学技术是信息安全的基础，使用密码学技术可以使得信息的保密性，可用性，完整性和抵赖性这些特点得到保障[3]。密码算法主要有对称密码算法，非对称密码算法，单向散列函数[4]。对称密码算法是指在加密和解密时秘钥是一样，常见的对称密码算法有 AES，SM4，SM1，DES 算法[5]。非对称密码算法是指在加密和解密时双方可以享有公同的共享秘钥，但双方各自的内部的私钥是不一样的，且私钥基本上不能被破解，如此仅根据公钥和密文无法破解出数据，因此广泛应用于安全加密领域[6]。常见的非对称密码算法有 ECDSA，ECDH，SM2 等算法[7]。单向散列函数主要是 SHA1，SHA2，SM3 等，广泛应用于数据安全认证领域[8]。

随着集成电路的发展，SoC 芯片由于其具有芯片尺寸小，功耗低，可靠性强，运行速度快，可编程性高，运行速度快等优势已经成为集成电路领域的核心技术[9]。将密码算法以硬件的方式设计出来，相比于软件实现的密码算法，其安全性能更好，且运行速度更快[10]。将用硬件实现的密码算法单元，结合处理器，存储单元，各种接口等集成为一个完整的 SoC 芯片，如此则将 SoC 的优点和硬件实现密码算法的优点相结合了起来，极大的提高了信息传输的一个安全性，可以满足现在社会对信息安全传输的需求。数据认证加密 SoC 的实现可以有效解决各行各业对移动互联网安全便捷的需求，对各行各业产生深远影响[11]。所以研究数据认证加密 SoC 具有重要意义。

1.2 国内外研究现状

为了我国的商用密码安全性的发展，国家秘密管理局在 2002 年发布了一系列

的密码算法标准，发布的对称密码算法有 SM1，SM4，SM7 等，非对称密码算法有 SM2，SM9 等，单向散列函数则发布了 SM3[12]。目前国内已有多家公司研发出了基于国密算法的安全加密芯片，且芯片已经量产，并进行了广泛的应用，推动了国内数据安全认证加解密 SoC 的发展 [13]。基于国家安全算法的认证加密 SoC 主要有大唐微电子的 CBS-CE3D,复旦微电子 FM151M，华大电子 CIU98320A 等，且已经实现量产，广泛应用于金融，交通等行业 [14]。

将密码算法融入到芯片中，不仅能够提升密码的运算速度，而且更加具有安全性，更加符合目前大数据时代对于数据安全的一个要求。对安全认证加密 SoC 的研究一直是社会研究的重点。

储奕锋在 2007 年提出了一种 AES 算法的 FPGA 实现方式，该方案设计支持 AES 的三种密码长度：128，192，256，支持 ECB，CBC，CTR 算子工作模式，且在功能和速度上均取得了较优的性能 [15]。

李峰在 2009 年将 3DES 算法和 AES 加密核应用到 32 位 SoC 中 [16]。

ZhichengXie 等人在 2015 年实现了一种应用于生物信号的低功耗 AES 加密 SoC，实现的系统面积为为 48813um^2，最高可运行在 1GHz 下 [17]。

Philipp Koppermann 等人在 2017 年实现了低延时的基于曲线 Curve25519 的快速标量乘算法，实现延迟为 92us，为使用此椭圆曲线的 ECC 算法提供了低延时设方案 [18]。

Jun-Baek Choi 等人在 2020 年提出了一种 ECIES(椭圆曲线集成加密方案)的基于 SoC 的协议硬件架构。SoC 内部集成了 ECC 算法，SHA3 算法，和 TRNG(真随机数发生器)等 [19]。

Sihabul Islam 等人在 2021 年提出了一种双 AES 加密核的 SoC，在功耗和安全性能优于 AES 单核 SoC[20]。

明洋等人在 2022 年在开源蜂鸟 E203 MCU 的基础上扩展了适用于 AES，RSA 复合加密场景的协处理器，组成拥有安全场景扩展的 RISC-V SoC[21]。

1.3 研究目标和内容

1.3.1 研究目标

论文的研究目标是：分析 ARM Cortex-A7 的多核处理器的构造，结合国内外对 SoC 系统设计和数据认证加密的研究，设计完成基于 ARM Cortex-A7 处理器的的数据认证加密 SoC 系统电路总体结构的总线架构，分析并完成秘钥生成算法，消息完整性和加解密算法的设计和仿真，并结合 UART，GPIO，I2C 等通用接口 IP 将其集成到一个完成的 SoC 上，使其能够实现数据认证加密的数据的正确传输，

能够应用在数据传输认证加密领域。

1.3.2 研究内容

本文参考所参与的实际项目，分析了国内外数据认证加密的研究和发展状况，研究了秘钥生成算法，数据认证算法和数据加密算法，研究了 ARM Cortex-A7 多核处理器的架构，AMBA 协议及相关资料，提出了一套完整的基于 ARM Cortex-A7 多核处理器的数据认证加密 SoC 系统设计，主要工作内容如下：

（1）研究 ARM Cortex-A7 多核处理器的架构和 AMBA 总线协议，为满足数据的认证加密传输的功能需求，选取对应的密码安全算法，需要用的通用外设接口 IP，构建基于 ARM Cortex-A7 处理器的 SoC 系统总体架构并且完成该 SoC 系统的集成。

（2）对秘钥生成算法，消息完整性认证算法，消息加解密算法的相关基础知识和原理进行介绍，然后完成对应算法的设计及其仿真。

（3）对集成的数据认证加密 SoC 系统进行 FPGA 板级验证分析，在真实环境下测试 SoC 电路的基本功能是否正确。

1.4 论文章节安排

论文总共包括 6 章内容：

（1）第一章：绪论。本章主要介绍论文的研究背景及意义，以及数据认证加密 SoC 的国内外研究现状，论文的研究目标和主要工作内容。

（2）第二章：数据认证加密 SoC 的总体实现。本章主要对 ARM cortex-A7 处理器及其架构进行介绍，然后介绍了用到的 AMBA 总线协议。接着对该 SoC 系统进行资源分析并且选取了相关的安全密码算法和所需的通用外设接口 IP 资源，并且设计此系统的总体架构和总线架构，最后进行整个数据认证加解密 SoC 的系统集成。

（3）第三章：密钥协商算法。本章主要介绍 ECDH 密钥协商算法及其设计和仿真。

（4）第四章：消息完整性认证及加密算法。本章主要介绍消息完整性认证算法和加解密算法，包括这两个算法的实现和仿真。

（5）第五章：FPGA 原型验证。本章主要对该 SoC 系统进行板极验证，主要验证了 3 个安全密码算法的正确性。

（6）第六章：总结与展望。对论文的工作做出总结和分析，并对后续工作进行展望。

第二章 数据认证加密 SoC 的总体实现

2.1 ARM 处理器与 A7 架构

处理器作为 SoC 系统的核心组成部分，与芯片的整体性能有着密切的关系，我们需要根据具体的功能需求和实际应用场景来选择合适的处理器。

目前市场上广泛采用了 ARM 处理器来进行 SoC 的设计。ARM 处理器已经广泛被应用于通信领域，消费内移动电子，无线领域等。根据应用领域的不同 ARM 系列处理器可以分为 A 系列，R 系列和 M 系列 [22]。A 系列处理器比 R 系列处理器和 M 系列处理器多拥有了一个存储管理单元(Memory Management Unit，MMU)，故 A 系列处理器因其能满足大量运算的需求而被广泛应用于视频，图像和音频处理等领域 [23]。根据项目的需求，我们需要对大量的数据进行认证加密计算，所以选择 ARM Cortex-A7 作为本设计的处理器内核。

ARM Cortex-A7 使用 ARMv7-A 架构，主要特性如下：

(1)：具有可扩展的多核处理器和单核处理器，支持在一个处理器上配置 1~4 个内核。

(2)：具有高效的超标量流水线，可以维持较低的功效，降低了封装和操作成本，减少了中断延时。

(3)：支持浮点运算单元(Float Point Unit，FPU)。显著提高了单精度及双精度标量浮点的运算速度，为图像处理，图形处理等提供了较强的科学运算能力。

(4)：支持 NEON，可以加速多媒体和信号处理功能，提升了具体应用性能，使的软件开发可以更加的便利。

(5)：利用最低存取延时技术，大幅度提升了处理器的性能和降低了功耗。

(6)：支持 4 个数据缓存通道的填充请求，而且还能够通过自动或用户预取操作，保证了关键数据的可用性，并且减少了由内存延时导致的暂停现象。

2.2 AMBA 总线协议

ARM 公司在 1995 年开发了 AMBA 片上通信标准，通过该总线标准，可以将各个 IP 集成到一起，该标准已经成为主流芯片上的总线架构之一。由于对不同速率的需求，AMBA 也提供了多个通信通信总线标准，如 AXI 总线，APB 总线，AHB 总线等。本文中主要用到的总线是 AXI，APB。

2.2.1 AXI 总线协议

AXI 总线协议有读地址通道，读数据通道，写地址通道，写数据通道，写响应通道。由于读写通道的独立，可以使得读写数据可以单独进行，提高了数据传输的效率。AXI 总线的通道传输图如图 2-1 所示：

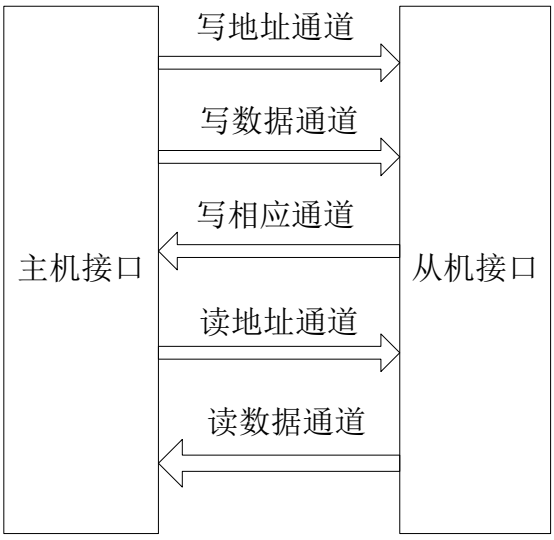


图 2-1 AXI 总线数据传输图

AXI 总线是基于猝发传输的总线协议，所以写地址通道和读地址通道需要传输数据的地址信息和数据的各种控制信息，如传输的数据大小等。从从机返回主机写操作传输是否成功的状态信息由写响应通道进行传输；主机和从机之间的数据传输是依据写地址通道中传输的地址信息和控制信息来完成。读地址通道中传输的地址信息和控制信息从从机返回主机数据并且返回主机读数据的完成状态则是由读数据通道完成的 [24]。

AXI 总线协议的几个关键特性如下：

(1)读写地址，数据是分开的。

(2)通过字节选通信号(WSTRB)实现支持非对齐数据传输功能。

(3)使用突发模式时，只需要传输首地址和设置突发长度即可。

(4)可以发送多个地址进行读写，且支持乱序传输。

(5)可以比较容易的添加寄存器级，以此来是实现时序收敛。

(6)每一个通道都有一对握手信号，读数据和写数据通道都有 LAST 信号来指明消息传输的最后一个数据。

AXI 总线的读写流程：

AXI 总线的写流程：首先，在写在地址通道内主设备向从设备说明要控制的寄存器地址，以及一些突发模式和控制信号；然后在写数据通道内主设备向从设

备写入数据；最后在写响应通道内从机进行应答；完成向从设备的写数据操作。

AXI 总线的读流程：首先，在读地址通道主设备向从设备说明要控制的寄存器地址和控制信号；从设备接收到这些信号后会才会对数据进行读取操作；完成这些处理后，在读数据通道内从设备将数据发送给主设备。

2.2.2 APB 总线协议

APB 总线协议是 AMBA 总线协议中的一种。APB 总线接口由于低成本低，低功耗，低复杂度而被广泛应用 [25]。

APB 主要用在低速 IP 中，如 UART 接口，I2S 接口等，且 APB 只有一个主设备，但可以有很多从设备 [26]。

APB 总线类型先后发布了 APB2，APB3，APB4。APB3 和 APB2 相比，APB3 多了从机准备信号 PREADY 和错误反馈信号 PSLVERR 这两个信号，功能上多了等待状态功能和错误报告功能。APB4 和 APB3 相比，APB4 多了保护信号 PPROT 和写选通信号 PSTRB 这两个信号，功能上多了安全功能，监督功能和稀疏数据传输功能。由于在本设计中，用不到 PPROT 和 PSTRB 这两个信号，所以选择性能较好的 APB3 作为本文的总线协议。

APB 总线协议的特点如下：

(1)主要应用在低带宽的外设上，如 I2C 等。

(2)APB 协议不是流水操作，两个周期完成一次读或者写的操作。

(3)读通道和写通道相互独立。APB 的读通道和写通道没有自己的握手信号，所以不支持读写并行操作。

(4)唯一的主设备是 APB 桥。

2.3 SoC 系统实现

2.3.1 设计资源需求分析

由于本文设计的为数据认证加密的 SoC 芯片，该 SoC 处理芯片主要用于对视频的传输数据进行认证加密处理，所以需要具备丰富的高速接口，和各种通用外设接口来满足对不同的设备间的认证加密通信。数据的安全传输主要从三个方面来体现数据传输的一个安全性：秘钥安全性，加密安全性，认证安全性。目前的秘钥安全性能的保护主要是利用椭圆曲线的离散对数问题来实现的，目前国际上流行的共享秘钥算法为 ECDH 算法，国内也发行了秘钥协商算法 SM2 算法 [27]。为了使得算法的通用性更强，本文选择 ECDH 算法来进行秘钥协商。而秘钥协商算法的安全性能的高低，主要取决于选取的椭圆曲线 [28]。本文选取安全性较高的

且目前还没有被攻破过的曲线 Curve25519 作为秘钥协商算法 ECDH 的基础椭圆曲线 [29]。目前国际上流行的加密算法有 AES，DES，3DES 等，国内也发行了对称密码算法标准 SM4[30]，目前 AES 的加密性能高，消耗资源少，运行速度快而被广泛使用，本分选择 AES 算法来实现加密功能，而为了满足对大量数据的快速加密运算，选取了 CTR 模式 AES-CTR 算法来对数据进行加密 [31]。目前的安全性认证算法有哈希认证算法和加密认证算法，哈希认证算法有 HMAC-SHA256，HMAC-SM3 等 [32]，加密认证算法有 AES-CMAC，SM4-CMAC。而目前基于 HASH 函数的认证算法由于其实现成本低，安全机制完善而被广泛使用，本设计选取哈希认证函数 HMAC-SHA256 来对数据进行一个认证处理 [33]。

基于以上分析，本设计的所选取的基本 IP 设计资源如表 2-1 所示：

表 2-1 IP 设计资源表

模块分类	软 IP 核
CPU	双核 ARM A7，支持 8KB L1Cache，128KB L2Cache
片上存储	SRAM：32KB；Boot Rom：32KB
低速外设	4 路 UART，2 路 SPI，I2C，I2S，GPIO，Timer，Whtch Dog
DDR	支持 32 位的 DDR3
功能模块(自主设计）	ECDH 算法，HMAC-SHA256 算法，AES-CTR 算法

处理器是采用的双核 ARM A7 处理器，内部的 NEON 协处理器支持视频，图像等指令操作，可以更加方便快捷的对视频数据进行处理。片上存储系统主要为 32KB 的 SRAM 和 32KB 的 ROM。只读存储器 ROM 的大小为 32KB，用来存放 SoC 的系统启动程序(Bootloader)。可读写的静态随机存储器 SRAM 的大小为 32KB，在 SoC 系统中用来对处理的数据进行缓冲，以此来提升处理器的性能。低速外设主要实现与片外低速外设的数据通信以满足不同场景通信需求。DDR SDRAM 主要存储芯片的数据以及处理所有模块的交互缓存，芯片通过 DDR3 接口读写外部 DDR Memory，外部扩展 memory 大小为 1G。

本文主要对功能模块中的三个算法进行设计，然后利用现有的 DDR，SRAM 等 IP 进行 SoC 的系统设计，以实现对数据进行认证加解密的功能。

2.3.2 SoC 总体结构设计

根据数据认证加密 SoC 的需求，本文采用嵌入式 SoC 来设计实现此方案，前由 4.1 节的分析可知，本文采用 ARM Cortex-A7 处理器作为处理器核。ARM A7 处理器与 ARM7 或 ARM9 为核心的芯片相比，具有体积小，性能高，适合大量数据的处理，十分适用于数据的认证加密计算 [34]。本文根据设计需要，对 ARM Cortex-A7 进行配置，其具体配置内容如表 2-2 所示：

表 2-2CPU 配置表

参数	描述
NUM_CPUS=2	配置的 CPU 内核数为 2
GIC_PRESENT=1	设计配置 Xx 中的通用中断控制器 GIC
NUM_SPIS=224	配置中断数为 224
L2_CACHE_PRESENT=1	配置有 L2 缓存
L2_CACHE_SIZE=3'b000	配置 L2 缓存的大小为 128KB
L1_DCACHE_SIZE=3'b001	配置 L1 数据缓存的大小为 8KB
L1_ICACHE_SIZE=3'b001	配置 L1 指令缓存的大小为 8KB
FPU_0=1	配置有 FPU0 单元
NEON_0=1	配置有 NEON0 单元
FPU_1=1	配置有 FPU1 单元
NEON_1=1	配置有 NEON1 单元
FPU_2=1	配置有 FPU2 单元
NEON_2=1	配置有 NEON2 单元
FPU_3=1	配置有 FPU3 单元
NEON_3=1	配置有 NEON3 单元
CLOCK_CONFIG=32'h00010001	配置 CPU 时钟为 50MHz

本设计跟据需求设计出的数据认证加解密 SoC 的系统框图如图 2-2 所示：

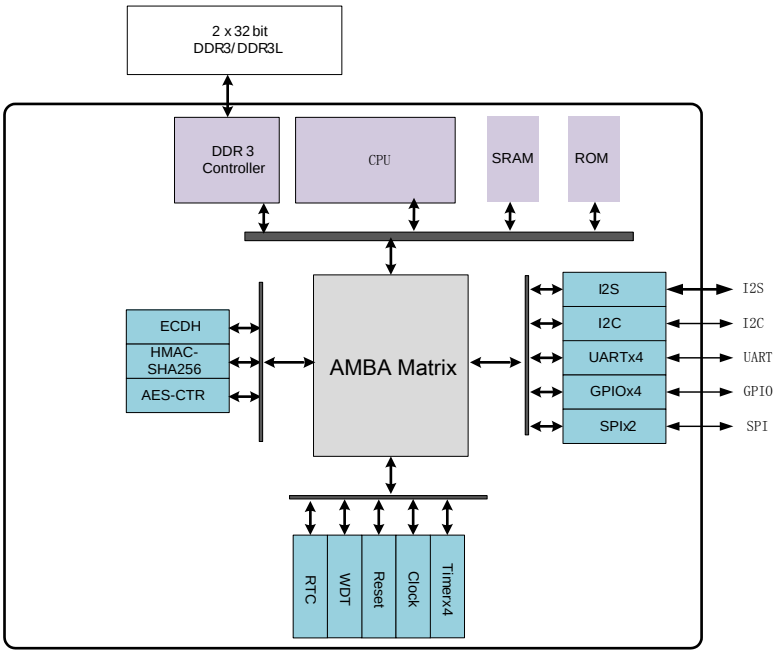


图 2-2 认证加密 SoC 结构图

在本设计中，利用 ARM 公司的 AMBA 总线互连来进行互连的设计，高性能

总线 AXI4 总线的外设包括 DDR 和 CPU，AXI3 总线的外设有 32KB 的 SRAM 和 32KB 的程序存储器 ROM。RAM 可以用于存储处理的数据，ROM 可以存储软件程序，提高了芯片的整体性能。APB 总线上挂的外设有 ECDH，HAMC-SHA256，AES-CTR，I2S，I2C，UART，GPIO，SPI，RTC，WDT，Reset，Clock，Timer。

2.3.3 总线架构设计

为了使 SoC 中的各个 IP 能够协同合理的工作，便需要设计合理的总线架构来连接各个 IP，使得它们形成一个有机的整体。本文利用 AMBA 总线矩阵来对总线进行设计，利用两个总线来进行互连，分别为 BUS-A 和 BUS-B。然后两个总线间利用互连矩阵 B0，B1，B2 进行连接。BUS-A 中主要利用 BO 来连接 WDT，RST，CLK，CTL，ECDH，HMAC-SHA256，AES-CTR 模块；利用 B1 来连接 4 路 Timer，3 路 UART，2 路 SPI；利用 B2 来连接 I2S，I2C 和 4 个 GPIO。BUS_B 中主要是连接 UART，ROM，RAM，CPU 和 2 路 DDR。BUS-A 和 BUS-B 的实现图分别如图 2-3 和图 2-4 所示：

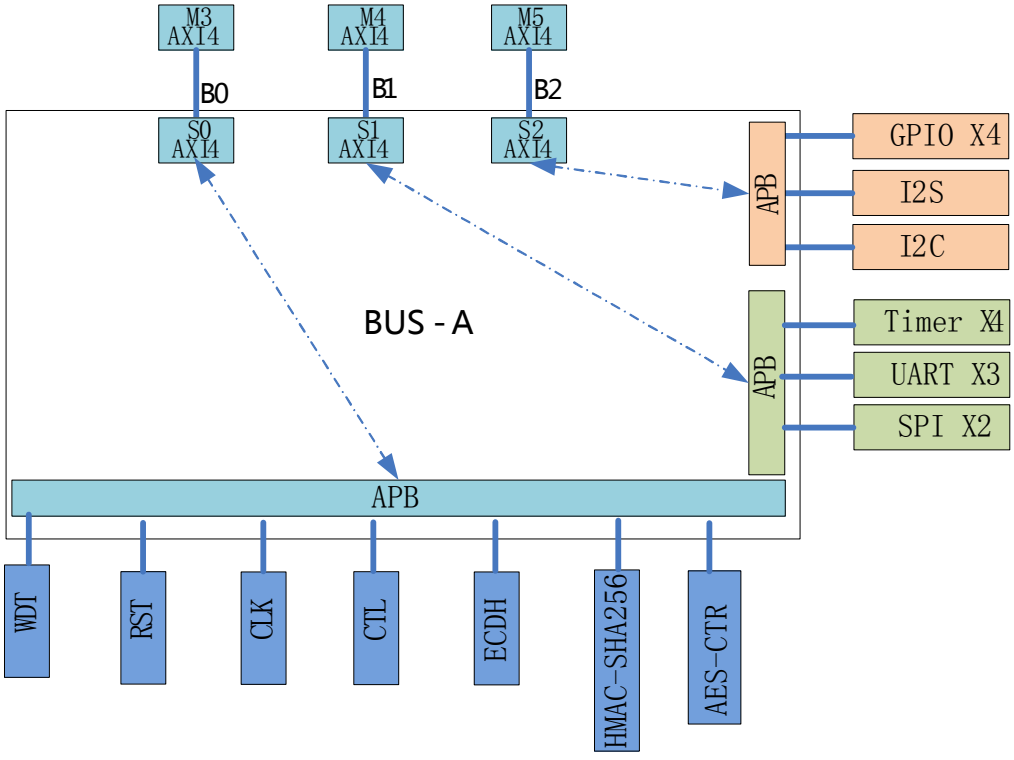


图 2-3 BUS-A 实现结构图

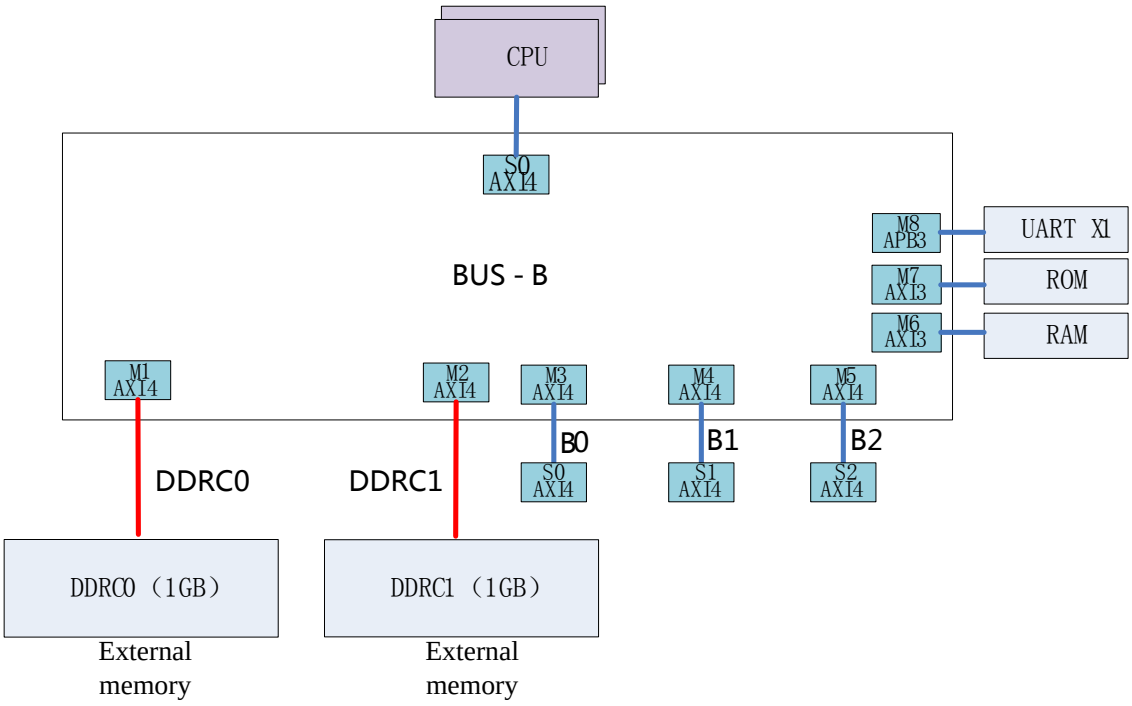


图 2-4 BUS-B 实现结构图

2.3.4 SoC 系统集成

认证加密 SoC 系统集成主要分为系统地址映射，中断的分布和模块的集成，以下分别其进行介绍。

(1)地址映射

Xx 核访问地址范围为 0x0-0xFFFFFFFF，主要是用来访问地址空间和访问系统中的寄存器配置。系统集成时采用的总线是由 ARM 公司的 AMBA 进行设计，利用 2 个总线 BUS-A 和 BUS-B 来对各个 IP 进行分配。本项目中设计的系统地址部分映射如表 2-3 所示：

表 2-3 系统地址映射表

模块	地址空间	大小	总线
DDR0	0x00000000-0x3fffffff	1GB	AXI4
DDR1	0x00000000-0x7fffffff	1GB	AXI4
B0	0x80000000-0x8fffffff	256MB	AXI4
B1	0xa0000000-0xafffffff	256MB	AXI4
B2	0x90000000-0x9fffffff	256MB	AXI4
UART1	0xb0000000-0xb0ffffff	16MB	APB3
BootRom	0xffff0000-0xffff7fff	32KB	AXI3
SRAM	0xffff8000-0xffffffff	32KB	AXI3

表 2-3 系统地址映射表（续）

模块	地址空间	大小	总线
GPIOA	0x90000000-0x900fffff	1MB	APB3
...	...	...	...
I2C	0x90900000-0x909fffff	1MB	APB3
TIMER	0xa0000000-0xa00fffff	1MB	APB3
HMAC-SHA256	0x80000000-0x800fffff	1MB	APB3
ECDH	0x80100000-0x801fffff	1MB	APB3
AES-CTR	0x80200000-0x802fffff	1MB	APB3

(2)中断分布

中断是指当 CPU 在执行当前程序时，由于系统出现了某种需要处理的紧急情况，CPU 暂停正在执行的程序，转而去执行另一段特殊程序来处理出现的紧急事务，处理结束后 CPU 自动返回到原先暂停的程序中继续执行的情况。中断的主要目的是：通过中断可以提供 CPU 的效率和维护系统的正常工作。所以给 SoC 系统分配合理的中断是十分重要的。SoC 系统电路中是通过将所有模块的中断信号连接到 CPU 中的中断控制器来控制各个模块的工作状态的。由 4.3.2 节可知，设计中的双核 Cortex-A7 系统内部集成的中断信号有 224 位，我们需要利用这 224 位来合理分配中断号给不同的中断源，以方便后期对系统的整体验证，调试和开发。

(3)模块集成

本文的整个系统的集成需要整个团队的参与。本文采取基于 IP 复用的方式对 SoC 系统电路进行集成设计，所用到的软 IP 的列表以在 4.3.1 节中介绍过，除了 ECDH 算法，HAMC-SHA256 算法，AES-CTR 算法外，其他如 UART，I2C 等模块均采用非商用开源软 IP 核。在集成第三方 IP 时，需要学习和研究其工作原理，研究所选用的 IP 与系统总线的接口是否匹配，信号位宽，时序等是否一样。在系统模块集成时，首先选取匹配的软 IP 核，然后利用设计的总线矩阵将 CPU 和各个功能模块进行连接，以此来构成一个完整的 SoC 系统。

2.4 本章小结

本章首先对 ARM 处理器及其架构进行了分析介绍，然后分析了用到的 AXI 和 APB 总线的基础知识，接着对数据认证加密 SoC 进行功能需求分析，整理出了需要的 IP 列表；然后对 SoC 的总体架构进行了设计，接着对 SoC 的总线结构进行分析和设计；最后完成整个 SoC 的系统集成。

第三章 密钥协商算法

密码学是数据保护的基础，数据传输的安全性高低又依赖于秘钥的质量。秘钥的质量越高，攻击方破解秘钥所付出的成本也会越多，该传输过程的保密性也就越好。公钥密码学的研究一直是密码学研究领域的重点，而基于椭圆曲线对数问题而构建的公钥密码算法也是近些年的一个研究热点。本文将利用CURVE25519 曲线来进行秘钥协商，即使用 CURVE25519 曲线来设计 ECDH 算法，然后使用 ECDH 算法生成的秘钥来提供给数据认证加密传输 SoC 使用。

3.1 ECDH 密钥协商算法

ECC 全称谓“Ellipse Curve Ctyptography”，是一种基于椭圆曲线数学的公开秘钥加密算法。在 1985 年由 Neal Koblitz 和 Victor Miller 分别独立的提出了将椭圆曲线应用于密码学中。ECC 相比与 RSA 等算法进行相比较，在同样安全强度下所需要的计算量和秘钥长度都要小的多 [35]。由于 ECC 秘钥具有很短的长度，所以运算速度比较快，到目前为止，对于 ECC 进行逆操作还是很难的，数学上证明不可破解，ECC 算法的优势就是性能和安全性更高 [36]。实际应用可以结合 DH 秘钥形成 ECDH 秘钥协商算法。Diffie-Hellman 秘钥协商算法是能够在未被保护的信道中建立安全的共享秘钥，已经成为了安全协议的基石。基于椭圆曲线的秘钥协商算法即 ECDH 算法由于结合了椭圆曲线的安全性，所以安全性能更高，从而被广泛应用于秘钥协商领域 [37]。目前由美国国家标准与技术研究院（National Institute of standards and Technology，NIST）发布的 NIST-P 系列椭圆曲线，如 NIST-P256 等在 ECDH 中的使用是最为广泛的。2006 年，Daniel J.Bernstein 设计并发表了 Curve25519 曲线 [38]。且 Curve25519 曲线相比与 NIST P-256 曲线有着性能方面上的明显优势，且此曲线的参数设计公开，因此被认为是安全的 [39]。因此本文选择使用 Curve25519 曲线来在硬件上实现 ECDH 算法。

ECDH 的每个用户都有一个 32 字节的私钥和一个 32 字节的公钥。每组 Curve25519 用户都有一个 32 字节的共享秘钥，用于验证和加密两个用户之间的消息。ECDH 算法的具体流程算法流程如下所示：

假设秘钥交换双方为 Alice 和 Bob，享有共享参数（p，a，b，G，n）。

(1):Alice 生成随机数 x，并计算 A=x*G。Bob 生成随机数 y，计算 B=y*G；

(2):Alice 将 A 传递给 Bob；

(3):Bob 收到 Alice 传递的 A,计算 Q=y*A；

(4):Alice 收到 Bob 传递的 B,计算 Q'=x*B；

(以上 4 个步骤中的乘法*是素数域的标量乘法）

Alice 和 Bob 双方得到 Q=y*A=y*x*G=x*y*G=x*B=Q'；即双方得到了已知的共享秘钥 Q(Q')；由于椭圆曲线的离散对数问题，所以攻击者不可以根据 A，G 算出 x，同理，攻击者不可以根据 B，G 计算出 y，所以 A 和 B 的传递可以公开。

3.1.1 素域及相关运算法则

由于 ECDH 算法选用的实现曲线为 Curve25519,所以该算法实现是基于有限域中的素数域的，且素数 P 为2^{255}-19，下面介绍素域及其相关运算法则。

有限域 F 是指只含有有限个元素的域，也称为 Galois 域；而有限域又分为素域 GF(p)和特征值为 2 的二元域 GF（2^m)。素域是由模 P 的全体余数的集合{0,1,2，...，p-1}构成的。素数域用符号表示为 F（p)，p 为 F（p）的模。amodp 的意思是 a 除以 p 的余数 r,其中 0≤r≤p-1；

素域上元素的四则运算规则如下：

（1）加法运算：为整数的模 P 加法运算，即 a,b∈F(p),a + b = (a+b)modp;

（2）乘法运算：为整数的模 P 乘法运算，即 a,b∈F(p),a + b = (a+b)modp;

（3）素域中的加法群的单位元为整数 0；

（4）素域中的乘法群单位元为整数 1；

（5）素域中的加法群的元素 a 的逆元素为 p-a；

（6）素域中的乘法群的元素 a 的逆元素为 b，b 满足 ab=1modp,记为 b=a^{-1};

以具体运算为例，若素数域的阶为 11，该素数域包含{0，1,2,3,4,5,6,7,8,9,10}这 11 个元素，其模运算结果如下所示：

模加：3+2=（3+2）mod11=5;

模减：2-3=（2-3）mod11=10;

模乘：3 • 5=（3 • 5）mod11=4;

模逆：3^{-1} = 3^{-1}mod11=4;

3.1.2 素域上的椭圆曲线及相关运算

基于 curve25519 曲线的 ECDH 算法其标量乘运算为素数域上的椭圆曲线计算。素数域上的椭圆曲线计算又分为点加运算，倍点运算和标量乘运算。

Curve25519 曲线就是素数域上的椭圆曲线，它是 Montgomery(蒙哥马利)曲线的一个实例，是建立在 255 比特长度的素域（p=2^{255}-19）上，能够提供 128 比特的安全强度，该曲线的表达式如式(3-1)所示：

$$y^2 = x^3 + 486662x^2 + x \tag{3-1}$$

该曲线的需要用到的参数为：

(1)该椭圆曲线的阶为$p=2^{255}-19$；

(2)基点 P=(x_p，y_p)其中x_p=9，

y_p =14781619447589544791020593568409986887264606134616475288964881837755586237401。

素数域上椭圆曲线的加法运算规则和实数域的加法运算规则是一样的，设$p=(x_1,y_1)$，其中$-p=(x_1,-y_1),(x_1,y_1),(x_2,y_2)\in E(GF(p))$,且满足$x_1 \neq x_2$，则素数域上的椭圆曲线点加运算的表示如式(3-2)所示。

$$(x_1,y_1)+(x_2,y_2)=(x_3,y_3) \tag{3-2}$$

则其运算结果如式(3-3)和(3-4)所示

$$x_3=\lambda^2-x_1-x_2，\ y_3=\lambda(x_1-x_3)-y_1 \tag{3-3}$$

$$\lambda=(y_2-y_1)/(x_2-x_1) \tag{3-4}$$

设素数域上的椭圆曲线上的一点为$(x_1,y_1)\in E(GF(P)),y_1 \neq 0$,则倍点运算的的表示如式(3-5)所示，

$$(x_1,y_1)+(x_1,y_1)=(x_3,y_3) \tag{3-5}$$

则其运算结果如式(3-6)和(3-7)所示：

$$x_3=\lambda^2-2x_1，\ y_3=\lambda(x_1-x_3)-y_1 \tag{3-6}$$

$$\lambda=(3x_1^2-y_1)/(2y_1) \tag{3-7}$$

标量乘是用一个随机整数 k 乘以椭圆曲线上的一个基点 P，即计算 k 个点 P 向加后得到的椭圆曲线上的另一个点 Q，Q = k*P。

3.2 ECDH 密钥协商算法设计及仿真

3.2.1 素数域加法模块

素数域的模加运算是实现素数域的两个域元素之间的相加运算，其与正常的代数学中的加法基本上是一样的，唯一的区别是需要对加法运算的结果取模 p，将计算结果约减到[0，p-1]内，即 a，b∈F(p)，a + b = (a+b)modp；由于本文采用的数据位宽是 256 位位宽的数据，故需要 256 位的模加法器。但是由于位宽的增加，会导致系统延时的增加，为了加快运算的速度，采用流水线的设计方式来加快模

加运算模块的运行速度，并且利用符号位来避免大数比较器，进一步增加了大数模加法器的运行速度。表 3-1 描述了素数域的模加的算法：

表 3-1 素数域加法算法表

算法名称：素数域加法运算
输入：a,b∈F(p),p 为素数域的阶
输出：c = (a+b)modp.
(1)c= a+b.
(2)如果 c>=p,c=c-p.如果 0<=c<p,则执行(3)
(3)返回 c 的值

由数学理论证明，两个符号相同的数相加，可能会产生溢出，导致加法的结果是错误的，所以需要对数据进行位扩展。素数域的两个数 a,b∈[0,p-1]的加法运算，通过计算可知，(a+b)∈[0,2p-2]，所以将加法结果约减到[0,p-1]内，最多只需要将加法计算结果减去一次 p。假设中间计算变量为 256 比特变量 mod_add_mid_r1,对于阶 p，计算溢出量结果为 param = 2^{255}-p=19；中间计算变量 mod_add_mid_r1 的计算方式为：mod_add_mid_r1=a+b+param；则当 sum1[256] = 1'b1 时，表示 a+b+param≥ p+param，进而得到 a+b≥p,进而如果需要将其结果约减到[0,p-1]内，则需将 a+b 的运算结果减去 p。当 mod_add_mid_r1[255]=1'b0 时，表示 a+b+param≤ p+param，进而得到 a+b≤p，此时，a+b 的值在[0,p-1]内，直接将结果输出即可。

设计该模块时，由于频率要求，所以需要提高大数模加模块的运算速度，故利用流水线的设计方式，在计算中间计算变量时，插入一级寄存器，以此来提高模加运算模块的运行速度。模加运算模块的接口信号定义如表 3-2 所示：

表 3-2 素数域加法模块信号表

信号名称	方向	位宽	功能描述
clk	输入	1	时钟信号
rst_n	输入	1	复位信号，低电平有效
en	输入	1	使能信号，高电平有效
add1	输入	256	模加数据输入 add1
add2	输入	256	模加数据输入 add2
done	输出	1	模加运算完成信号
mod_add_out	输出	256	输出的模加运算结果

素数加法运算也是 ECDH 算法系统框架设计实现的基本运算，采用流水线线的形式来设计素域加法模块，随机选择测试用例：p=

2^{255}-19=256'd57896044618658097711785492504343953926634992332820282019728792003956564819949；add1 = 256'd57896044618658097711785492504343953926634992332820282019728792003956564819948；add2 = 256'd11；通过计算可以得到，(add1+add2)modp = 10;使用 VCS 仿真工具可以得到仿真波形图如图 3-1 所示：

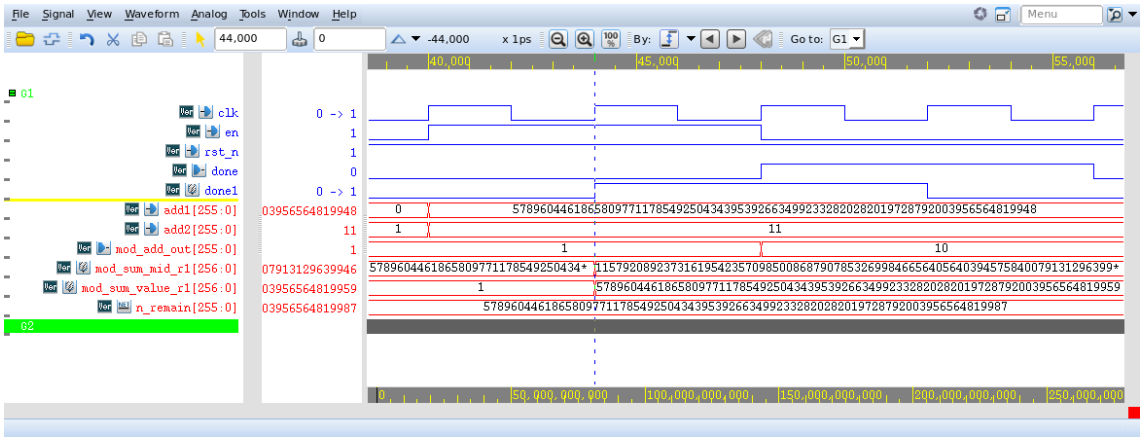


图 3-1 模加模块仿真波形图

由仿真图可以看出，当使能信号 en 拉高后，经过两个时钟，模加计算完成信号 done 拉高，并输出模加结果为 10，故由仿真可知，结果正确。

3.2.2 素数域减法模块

素数域的模减法运算是实现素数域的两个域元素之间的相减运算，其与正常的代数学中的减法基本上是一样的，唯一的区别是需要对减法运算的结果取模 p，将计算结果约减到[0，p-1]内，即 a，b∈F(p)，a - b = (a-b)modp；a -b = (a-b)modp；由于本文采用的数据位宽是 256 位位宽的数据，故需要 256 位的模减法器。但是由于位宽的增加，会导致系统延时的增加，为了加快运算的速度，采用流水线的设计方式来加快模减运算模块的运行速度，并且利用符号位来避免大数比较器，进一步增加了大数模减法器的运行速度。表 3-3 描述了素数域的模减的算法：

表 3-3 素数域减法算法

算法名称：素数域减法运算
输入：a,b∈F(p),p 为素数域的阶
输出：c = (a-b)modp.
(1)c= a-b.
(2)如果 c<0,c=c+p.如果 0≤c<p,则执行(3)
(3)返回 c 的值

由数学理论证明，两个符号相同的数做减法，当被减数小于减数的时候，其结果为负数。素数域的两个数 a，b∈[0，p-1]的加法运算，通过计算可知(a-b)∈[-p+1，p-1]，所以将减法结果约减到[0，p-1]内，最多只需要将减法计算结果加上一次 p。在设计时，我们利用带符号位的中间变量，来判断减法的结果是否小于 0。假设中间计算变量为 256 比特变量 mod_sub_mid_r1,中间计算变量 mod_sub_mid_r1 的计算方式为：mod_sub_mid_r1=a-b；则当 mod_sub_mid_r1[256] = 1'b1 时，表示 a-b<0，此时 a-b 的值在[-p+1,0)内，进而如果需要将其结果约减到[0,p-1]内，则需将 a-b 的运算结果加上 p。当 mod_sub_mid_r1[255]=1'b0 时，表示 a-b≥ 0，此时，a-b 的值在[0,p-1]内，直接将结果输出即可。

模减运算模块的接口信号定义如表 3-4 所示：

表 3-4 模减模块接口信号表

信号名称	方向	位宽	功能描述
clk	输入	1	时钟信号
rst_n	输入	1	复位信号，低电平有效
en	输入	1	使能信号，高电平有效
sub1	输入	256	模减的输入被减数
sub2	输入	256	模减的输入减数
done	输出	1	模减运算完成信号
mod_sub_out	输出	256	输出的模减运算结果

素域减法模块选取随机测试用例：

p=2^255-19=256'd57896044618658097711785492504343953926634992332820282019728792003956564819949；

sub1=256'd57896044618658097711785492504343953926634992332820282019728792003956564819948；

sub2=256'd57896044618658097711785492504343953926634992332820282019728792003956564819938；

通过计算可得，正确结果为 10；利用 VCS 软件对该模块进行仿真，得到的仿真图如图 3-2 所示：

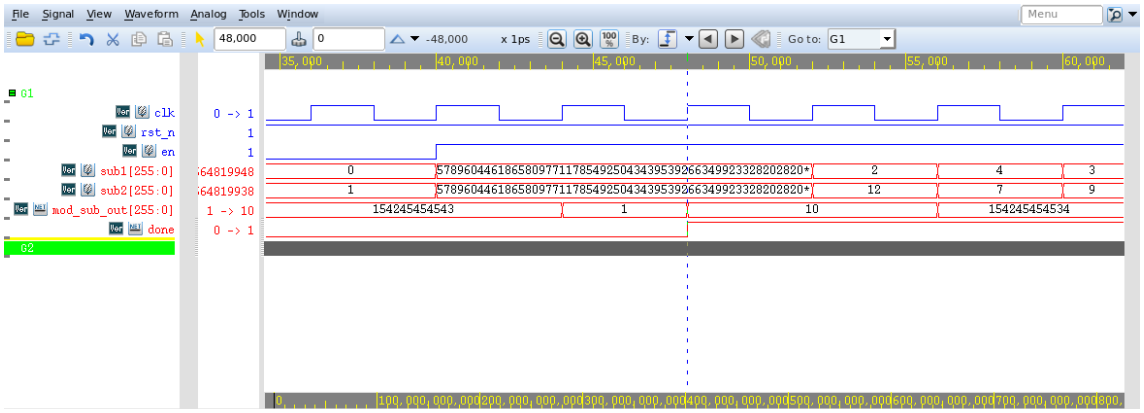


图 3-2 模减模块仿真图

由仿真图可以看出，当使能信号 en 拉高后，经过两个时钟，模减计算完成信号 done 拉高，并输出模减结果为 10，故由仿真可知，结果正确。

3.2.3 素数域乘法模块

有限域内的模乘模块是 ECDH 算法的核心运算模块，模乘模块的性能很大程度上决定了 ECDH 算法的性能。对于硬件实现模乘算法，通常使用的是 Montgomery 算法。但直接实现 Montgomery 算法会极大的消息资源，故设计选择简单，易实现，且资源消息适中的移位模乘算法，移位模乘算法的具体描述如表 3-5 所示：

表 3-5 移位模乘算法

算法名称：移位模乘算法
输入：a,b∈F(p),p 为素数域的阶
输出：c = (a·b)modp。
(1)c= 0；
(2)对于每个整数 i 从 0 到 t-1,重复执行：若 b_i=1,c =c+a<<1；
(3)返回 c 的值

模乘模块的的接口信号表如表 3-6 所示：

表 3-6 模乘模块接口信号表

接口信号名称	数据位宽	接口方向	信号描述
clk	1	输入	时钟信号
rst_n	1	输入	复位信号，低电平有效
a	256	输入	乘数输入 a
b	256	输入	乘数输入 b
en	1	输入	模乘使能信号
product	256	输出	输出模乘结果

done	1	输出	模乘完成信号

在设计该模乘法模块时，主要利用寄存器，状态机，及模乘运算单元三个部分组成，其设计的原理图如所图 3-3 所示：

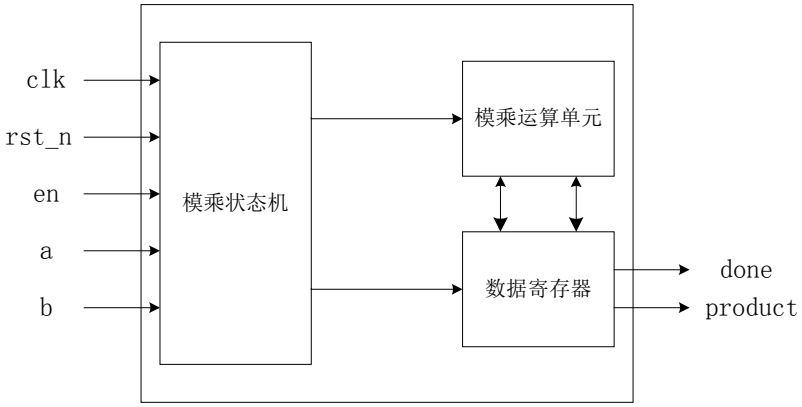


图 3-3 模加模块实现原理图

利用 VCS 对该模块进行仿真，随机选取测试用例：p=29；a=12；b=25；通过计算得知，(a·b)modp = 12·25mod29=300mod29=10；其仿真波形图如图 3-4 所示

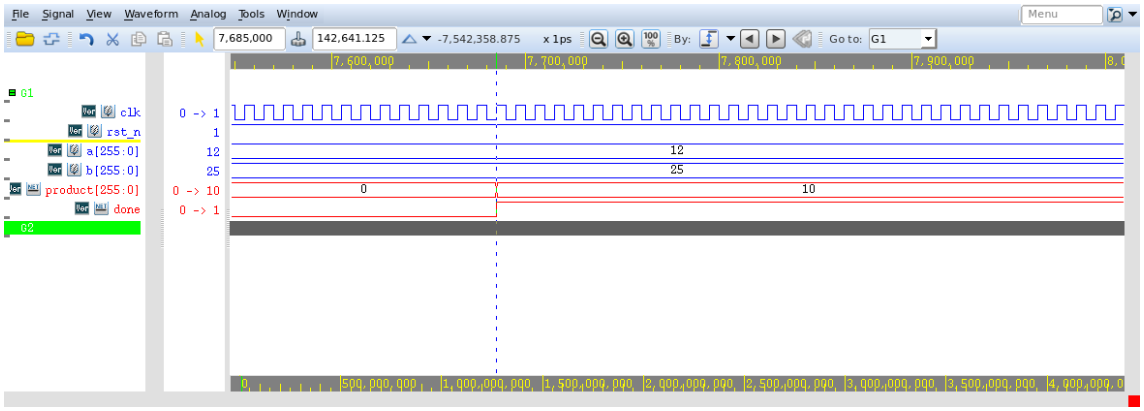


图 3-4 模乘模块仿真波形图

从仿真图中可以看到模乘的输出的正确结果 10。故由仿真结果可知，仿真正确。

3.2.4 素数域模逆运算模块

在素数域中，由于有限域的阶是素数，所以素数域中的元素的逆元都是存在的。在素数域中求一个元素的逆元的操作叫做模逆运算。模逆运算是有限域模运算中最复杂的运算。常用的模逆算法主要分为蒙哥马利算法和扩展欧几里得算法。本设计采用优化后的扩展欧几里得算法来进行模逆运算。扩展欧几里得具体算法

如表 3-7 所示：

表 3-7 扩展欧几里得算法

算法名称：扩展欧几里算法
输入：a，p；其中 a∈GF(p)，p 为素数域的阶
(1)初始化：u=a,v=n,A=1,C=0；
(2)若 u 的值不为 0，重复执行：
(2.1)若 u 为偶数，重复执行(2.1)：u=u/2；若 A 为偶数，A=A/2;否则 A=(A+p)/2；
(2.2)若 v 为偶数，重复执行(2.2)：v=v/2；若 C 为偶数，C=C/2;否则 A=(C+p)/2；
(2.3)若 u≥v，则 u=u-v，若 A>C，则 A=A-C；否则 A=A+p-C；
(2.4)若 u<v，则 v=v-u，若 C>A，则 C=C-A；否则 C=C+p-A；
(3)返回(Cmodp)

模逆运算接口信号表如表 3-8 所示：

表 3-8 模逆运算接口信号表

接口信号名称	数据位宽	接口方向	信号描述
clk	1	输入	时钟信号
rst_n	1	输入	复位信号，低电平有效
mod_inv_en	1	输入	模逆运算使能信号
in	256	输入	模逆元素输入
out	256	输出	模逆运算结果输出
mod_inv_done	1	输出	模逆运算完成信号

在设计该模模逆模块时，主要利用寄存器，状态机，及模逆运算单元三个部分组成，其设计的原理图如所图 3-5 所示：

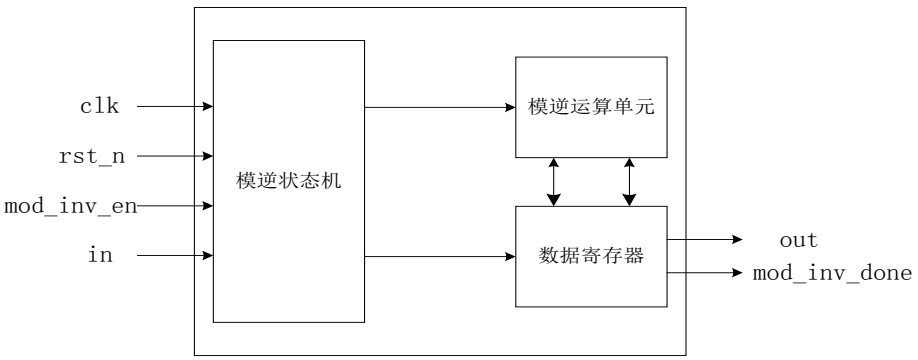


图 3-5 模逆模块原题图

利用 vcs 仿真工具对齐进行仿真，选取随机测试用例：p=197，a=17。通过计算得到，17·58mod197=986mod197=1，故17^{-1}mod197=58。模逆模块仿真结果如图 3-6 所示：

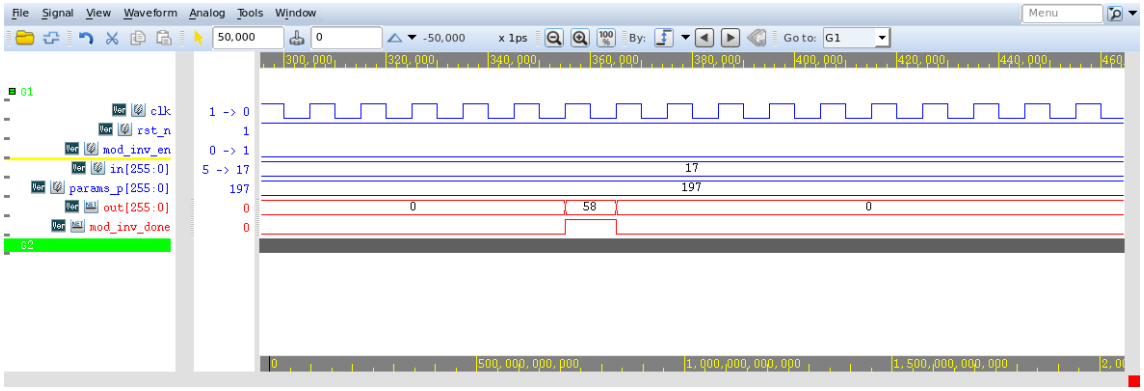


图 3-6 模逆模块仿真图

从仿真图中可以看出，经过一定时钟周期过后，输出模逆运算结果 58，与计算结果一致，仿真功能正确。

3.2.5 标量乘模块

为实现 ECDH 算法，则需要计算椭圆曲线上的标量乘运算。为了实现高效且轻量的素数域标量乘模块，本设计选取蒙哥马利阶梯算法来计算有限域中的椭圆曲线标量乘运算。该算法根据私钥的值，执行一系列的点加，倍点和点交换运算。使得两个被维护点的差值在每次迭代中都是恒定的。为了加速运算速度，该算法仅使用椭圆曲线上的点的 x 坐标和齐次射影坐标。在齐次射影坐标中，将点(x，y)投影到高维空间(x，y，z)中，使得(x，y)=(x/z，y/z)。蒙哥马利阶梯标量乘算法将点运算和倍点运算融合进阶梯运算中，不需要另外再设计点加和倍点模块。该标量乘具算法如表 3-9 所示：

表 3-9 蒙哥马利阶梯标量乘算法

算法名称：蒙哥马利阶梯标量乘算法
输入：256 比特长度的标量 k 与随机点 P 的 x 坐标x_p
输出：kP 的横坐标x_q
1：$X_1 = x_p; X_2 = 1; X_3 = x_p; Z_2 = 0;\ Z_3 = 1; K_{256} = 0$
2:对于 i 从 255 到 0，重复执行：
3:$c = k_{i+1} \oplus k_i$

4:(X_2, X_3)=cswap(X_2, X_3, c), (Z_2, Z_3)=cswap(Z_2, Z_3, c)
5:$t_1=X_2+Z_2, t_2=X_2-Z_2$
6:$t_3=X_3+Z_3, t_4=X_3-Z_3$
7:$t_6={t_1}^2, t_7={t_2}^2$
8:$t_5=t_6-t_7, t_8=t_1 \cdot t_4$
9:$t_9=t_3 \cdot t_2, t_{10}=t_8+t_9$
10:$t_{11}=t_8-t_9, X_3={t_{10}}^2$
11:$t_{12}={t_{11}}^2, t_{13}=121665t_5$
12:$X_2=t_6 \cdot t_7, t_{14}=t_7+t_{13}$
13:$Z_3=X_1 \cdot t_{12}, Z_2=t_5 \cdot t_{14}$
14: (X_2, X_3)=cswap$(X_2, X_3 k_0)$, (Z_2, Z_3)=cswap(Z_2, Z_3, k_0)
15:$Z_2={Z_2}^{-1}, x_q=X_2 \cdot Z_2$
16:返回x_q

表 3-9 中的 cswap 函数的算法如表 3-10 所示：

表 3-10swap 函数算法

算法名称：cswap 函数算法
输入：b∈{0,1},n 比特长度的x_0和x_1
输出：(x_b, x_{1-b})
(1) b=(b,b,...,b)n
(2)v=b&$(x_0 \oplus x_1)$
(3)返回$(x_0 \oplus v, x_1 \oplus v)$

在实现该设计模块时，利用状态机的设计方式，来对底层模运算模块(模加，模减，模乘，模逆)来进行调度，以此来实现对逻辑资源的一个优化和时钟周期的一个优化。

该标量乘运算的接口信号信号表如 3-11 所示：

表 3-11 蒙哥马利阶梯算法实现底层模块调度表

接口信号名称	数据位宽	接口方向	信号描述
clk	1	输入	时钟信号
rst_n	1	输入	复位信号，低电平有效
en	1	输入	标量乘运算使能信号
k	256	输入	标量输入
u	256	输入	椭圆曲线的横坐标
mod_scalar_out	256	输出	标量乘结果输出
mul_scalar_done	1	输出	标量乘运算完成信号

该标量乘运算模块的设计原理图如图 3-7 所示

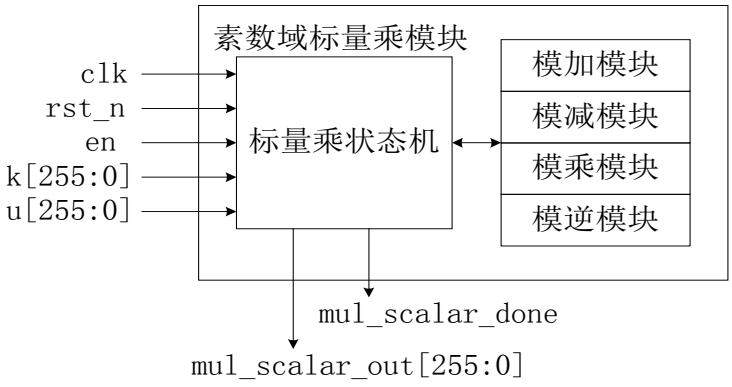


图 3-7 标量乘运算模块设计原理图

为了验证标量乘法的正确性，利用 c 语言实现了对应的标量乘算法，并选取相同的测试向量产生的结果来和 VCS 产生的结果进行比对。测试向量为：

k=256'h75；

u=256'h13；

k*u=256'h433cfeffd6c14b1b93b4e03e6d9ae66a0c758dc4bf36a48d5409ca229aad205f；

VCS 仿真结果图 3-8 所示：

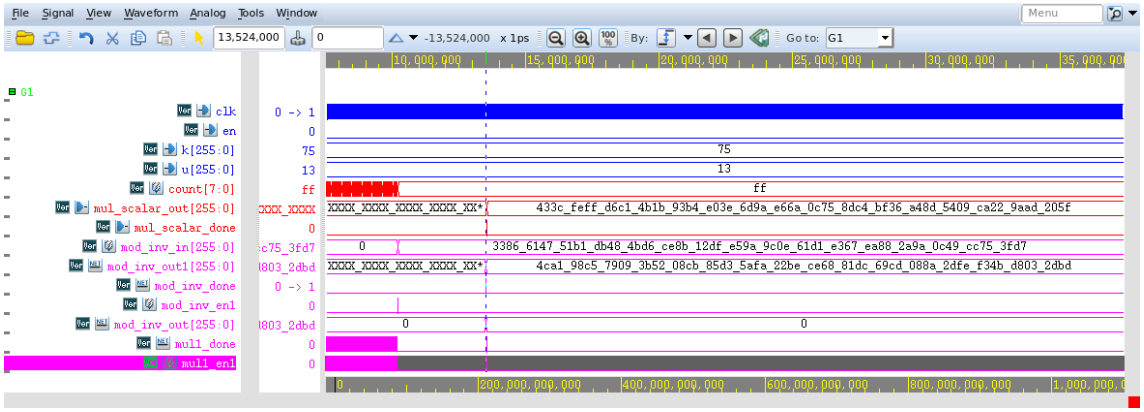


图 3-8 标量乘仿真结果图

首先对该模块进行上电复位操作，然后在输入标量 K 的值和椭圆曲线上一点的横坐标时，将标量乘使能信号 en 赋值为 1，并在一个 clk 过后复位为 0。在经过一定时间过后，可以标量乘运算完成信号 mul_scalar_done 信号拉高并且标量乘运算结果 mul_scalar_out 有输出。

将本文设计的点乘算法与近几年相关文献从逻辑资源，最高频率和完成一次

点乘运算所需时间来进行对比，其对比结果如表 3-12 示：

表 3-12 点乘设计对比

文献	FPGA 型号	逻辑资源/LUTs	最高频率/MHz	点乘时间
文献[40]	Xilinx Vetex-IV	91000	82.3	0.152892
文献[41]	Xilinx Artix-VII	15302	100	2.23
本文	Xilinx Zynq:XC7Z010	23152	95	0.24

从对比结果可以看出，本文设计兼顾了两者的优点，在消耗更少的逻辑资源的情况下，完成一次点乘所需要的时间也更加的少。利用该点乘算法产生的共享秘钥将用于数据认证加密 SoC 信息安全传输所使用。消息安全传输主要体现在安全认证和数据保护两方面，而安全认证和数据保护本设计分别选择消息散列消息认证码（Hash_based Message Authentication Code，HMAC）算法和 AES 算法。

3.3 本章小结

在本章中，本章首先介绍了 ECDH 算法的基础知识，明确选用的椭圆曲线为 curve25519。然后介绍素数域上的相关运算法及素数域上的椭圆曲线的相关知识及其相关运算。然后设计和仿真了素数域基本运算模块（模加，模减，模乘，模逆运算）。通过对素数域基本运算模块的复用，结合状态机设计了蒙哥马利阶梯标量乘模块，并实现了利用该标量乘算法生成共享秘钥。

第四章 消息完整性认证及加解密算法

4.1 消息完整性认证算法

本设计选取的消息完整性认证算法为 HMAC-SHA256 算法。HAMC 是和 Hash 函数有着密切关系的消息认证码，根据 hash 底层 hash 函数的不同，可以将 HMAC 算法分为很多不同的种类，记为 HMAC-X，其中 X 代表 HMAC 算法使用的 Hash 函数，如 SHA1，SHA256，SHA512 等 [42]。随着信息安全传输的要求越来越高，需要选取的安全散列算法的安全性也越来越高。此处选取安全性能较高的 SHA256 算法 [43]。

4.1.1 SHA256 算法

目前发布的 SHA 算法有 SHA-0，SHA-1，SHA-2，SHA-2 又可以分为 SHA-224，SHA-256，SHA-384，SHA-512，SHA-512/224，SHA-512/256[44]。SHA-2 系列的算法由于其散列值的长度更长，所以比其他 SHA-1 和 MD5 算法具有更高的安全性。而 SHA-512 虽然比 SHA-256 算法更安全，但由于其迭代次数（80）比 SHA-256 的迭代次数（64）更多，所以运算速度更慢，所以综合分析下，选择 SHA-256 作为 HMAC 算法的 Hash 函数。

SHA-256 算法的算法主要步骤如下：

（1）附加填充比特：假设输入消息序列的长度为 L，则需要在输入消息序列的末尾进行比特填充，其具体的填充方式为：填充比特串的最高位为 1，后面是 K 个 0，其中 K 为满足方程 L+K+1=448mod512 的最小非负解，并在最后附加用二进制表示的其数值为消息长度 L 的值。以此方式可以确保填充后的最终消息输入的长度是 512 位的整数倍。附加填充比特规则如图 4-9 所示，以输入比特串为字符“a，b，c”为例 1

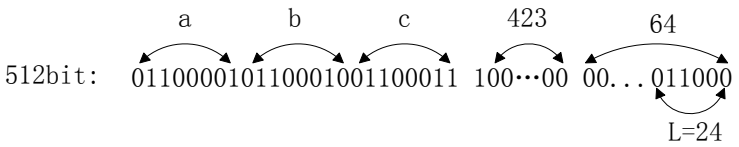


图 4-1 附加填充比特处理示列

（2）初始化 SHA-256 输出：在每一次的 SHA-256 计算开始前，都需要先将 SHA-256 输出进行初始化，利用 8 个 32bit 的寄存器来存放 SHA-256 函数的各个时刻的值。根据算法协议标准，其初始化值是由前 8 个质数的平方根的小数部分

的前 32 位的值，这 8 个初始值利用 16 进制表示为：

$$H_0^0=0x6a09e667$$

$$H_1^0=0xbb67ae85$$

$$H_2^0=0x3c6ef372$$

$$H_3^0=0xa54ff53a$$

$$H_4^0=0x510e527f$$

$$H_5^0=0x9b05688c$$

$$H_6^0=0x1f83d9ab$$

$$H_7^0=0x5be0cd19$$

（3）常量数组k_i的初始化，k_0-k_{63}的值是取自前 64 个质数(2-311)的立方根小数部分的前 32 位，用 16 进制可以将其从左到右表示为如表 4-1 所示：

表 4-1 常量数组 k_i 值表

428a2f98	71374491	b5c0fbcf	e9b5dba5	3956c25b	59f111f1	923f82a4	ab1c5ed5
d807aa98	12835b01	243185be	550c7dc3	72be5d74	80deb1fe	9bdc06a7	c19bf174
e49b69c1	efbe4786	0fc19dc6	240ca1cc	2de92c6f	4a7484aa	5cb0a9dc	76f988da
983e5152	a831c66d	b00327c8	bf597fc7	c6e00bf3	d5a79147	06ca6351	14292967
27b70a85	2e1b2138	4d2c6dfc	53380d13	650a7354	766a0abb	81c2c92e	92722c85
a2bfe8a1	a81a664b	c24b8b70	c76c51a3	d192e819	d698aa4a	f40e3585	106aa070
19a4c116	1e376c08	2748776c	34b0bcb5	391c0cb3	4ed8aa4a	5b9cca4f	682e6ff3
748f82ee	78a5636f	84c87814	8cc70208	90befffa	a4506cbe	bef9a3f7	c67178f2

（4）数组W_i的计算，W_i~W_{15}分别由 512 比特输入块以 32 比特从高位到低位均分得到，W_{16}~W_{63}的计算方式如式(4-1)所示：

$$W_i=W_{i-16}+W_{i-7}+S_0+S_1 \tag{4-1}$$

其中S_0，S_1的计算方式分别如式(4-2)和(4-3)所示：

$$S_0=(W_{i-15}ROR7)\oplus(W_{i-15}ROR18)\oplus(W_{i-15}SHR3) \tag{4-2}$$

$$S_1=(W_{i-15}ROR7)\oplus(W_{i-15}ROR18)\oplus(W_{i-15}SHR3) \tag{4-3}$$

（5）压缩函数的迭代计算：SHA-256 算法的迭代计算次数为 64 次，假设 8 个迭代常数变量分别为 a，b，c，d，e，f，g，h，e，f 他们都是 32 比特的数据变量。首先需要先对这 8 个迭代常数变量进行初始化，其初始化如下式(4-4)所示：

$$a=H_0^{(i-1)}$$

$$b=H_1^{(i-1)}$$
$$c=H_2^{(i-1)}$$
$$d=H_3^{(i-1)} \quad (4\text{-}4)$$
$$e=H_4^{(i-1)}$$
$$f=H_5^{(i-1)}$$
$$g=H_6^{(i-1)}$$
$$h=H_7^{(i-1)}$$

然后需要对迭代常数 a，b，c，d，e，f，g，h 进行更新，其更新公式如式(4-5)所示：

$$T_1^t=h_{t-1}+\sum_1^{\{256\}}(e_{t-1})+CH\left(e_{t-1},f_{t-1},g_{t-1}\right)+K_t^{\{256\}}+W_t$$
$$T_2^t=\sum_0^{\{256\}}(a_{t-1})+Maj\left(a_{t-1},b_{t-1}+c_{t-1}\right)$$
$$a_t=T_1^{(t-1)}+T_2^{(t-1)}$$
$$b_t=a_{t-1}$$
$$c_t=b_{t-1}$$
$$d_t=c_{t-1} \quad (4\text{-}5)$$
$$e_t=d_{t-1}+T_1^{t-1}$$
$$f_t=e_{t-1}$$
$$g_t=f_{t-1}$$
$$h_t=g_{t-1}$$

其中

$$CH(x,y,z)=(x\&y)\oplus(\neg x\&z) \quad (4\text{-}6)$$
$$Maj(x,y,z)=(x\&y)\oplus(x\&z)\oplus(y\&z) \quad (4\text{-}7)$$
$$\sum_0^{\{256\}}(x)=ROR2(x)\oplus ROR13(x)\oplus ROR22(x) \quad (4\text{-}8)$$
$$\sum_1^{\{256\}}(x)=ROR6(x)\oplus ROR11(x)\oplus ROR25(x) \quad (4\text{-}9)$$

最后计算每一步的 Hash 值，需要用到第（i-1）次的 Hash 值以及更新的迭代常数变量。其具体计算方式如式(4-10)所示：

$$H_0^{(i)}=a+H_0^{(i-1)}$$
$$H_1^{(i)}=b+H_1^{(i-1)}$$
$$H_2^{(i)}=c+H_2^{(i-1)}$$
$$H_3^{(i)}=d+H_3^{(i-1)} \quad (4\text{-}10)$$
$$H_4^{(i)}=e+H_4^{(i-1)}$$
$$H_5^{(i)}=f+H_5^{(i-1)}$$
$$H_6^{(i)}=g+H_6^{(i-1)}$$
$$H_7^{(i)}=h+H_7^{(i-1)}$$

当处理完最后的一个 512 比特的消息块之后，则将最后一迭代过后的运算结果作为最后的 SHA-256 输出，其输出结果如公式(4-11)所示：

$$H=H0|H1|H2|H3|H4|H5|H6|H7 \quad (4\text{-}11)$$

在以上式子中，ROR 表示循环右移，SHR 表示右移，⊕表示异或，|表示位拼接，&表示按位相与运算，┐表示取反运算。

4.1.2 SHA256 算法设计与仿真

SHA256 算法的电路在设计时，主要采用状态机的形式对其进行设计，其主要包括 SHA256 计算的控制状态机，W 和 K 的计算单元，以及压缩函数计算模块。其具体的设计原理框图如图 4-2 所示：

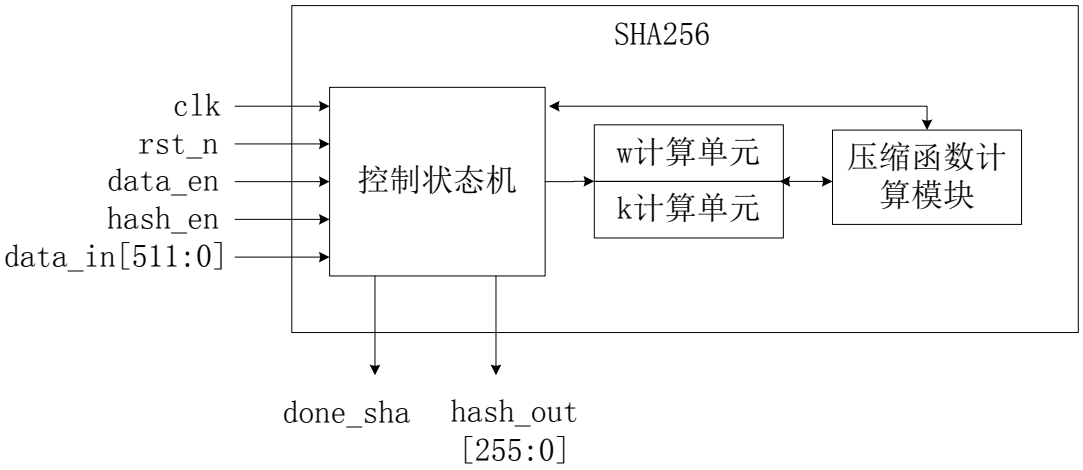


图 4-2 SHA256 算法设计框图

SHA256 算法模块的接口信号表如表 4-2 所示：

表 4-2SHA256 模块接口信号表

接口信号名称	数据位宽	接口方向	信号描述
clk	1	输入	时钟信号
rst_n	1	输入	复位信号，低电平有效
data_in	512	输入	512 比特消息块输入
data_en	1	输入	输入消息块有效信号
hash_en	1	输入	SHA256 计算使能信号
hash_out	256	输出	SHA256 计算结果输出
done_sha	1	输出	SHA256 计算完成信号

在控制状态机中，主要分为 3 个状态，idle(初始状态)，run(SHA256 计算状态)，done(结束状态)。最开始的先进行复位，便进入 idle 状态，在 idle 状态中，主要进行数值的初始化操作，将 SHA256 输出值初始化为初始 Hash 值，并且将计数器的值置为 0。接收到输入消息块有效信号 data_en 为 1 时，则开始进入 run 状态，这这个状态中，开始进行压缩迭代函数的计算，在此状态中，计数器的值每一个 clk，计数器的值加 1，当计数器的值计数到 63 时，则表示已经完成 64 轮迭代压缩计算，将进入最后一个 done 状态。

W 计算单元则主要利用组合逻辑来实现。在设计时，W_0~W_{15}的值由 512bit 的输入由高到低均分得到，利用 16 个寄存器分别来寄存W_0~W_{15}的值，由W_i的计算方式分析可知，每上一次 16 个W_i的值不会参与下一次 16 个W_i的计算，故设计时利用寄存器复用的方式，来完成W_0~W_{63}的计算，以此来减少设计使用的面积。

k 计算单元则是直接使用组合逻辑来对k_0~k_{63}来进行赋值，其值对应表(3-1)中的值。

压缩函数计算模块则是完成压缩迭代函数的计算，主要是利用 8 个寄存器来完成压缩迭代函数的更新即 Hash 值的更新。假设完成压缩迭代函数更新的 8 个寄存器分别为 a_new,b_new,c_new,d_new,e_new,f_new,g_new,h_new。当状态机处于 idle 状态时，首先将这 8 个寄存器的值进行初始化，将其值分别初始化为初始 Hash 值，即H_0^0，H_1^0，H_2^0，H_3^0，H_4^0，H_5^0，H_6^0，H_7^0。当状态机处于 run 状态时，则将寄存器的值按照计算公式(3-5)所示计算方式进行压缩函数计数器值的更新。在该状态内，计数器 cnt 的值每隔 1 个 clk 则加 1。当计数器的值为 63 时，则压缩函数迭代计算也就完成了，此时状态机机在下一个 clk 进入 done 状态。假设完成 Hash 值更新的寄存器位 H0,H1,H2,H3,H4,H5,H6,H7。当状态机处于 idle 状态时，这 8 个寄存器的值分别被初始化为初始 Hash 值，即H_0^0，H_1^0，H_2^0，H_3^0，H_4^0，H_5^0，H_6^0，H_7^0。当状态及处于 run 状态时，则将寄存器的值按照公式(3-10)所示方式进行更新，与压缩函数迭代寄存器更新一样，当计数器 cnt 的值为 63 时，进入最后一个转态 done

状态。进行 Hash 值的输出，在此状态中，寄存器的值组成的输出 H0|H1|H2|H3|H4|H5|H6|H7 便是完成一次 SHA256 计算的输出结果。

使用 VCS 仿真软件对该 SHA256 算法模块进行仿真，选取对应测试向量，测试向量来源于文献，选择测试向量输入为 24 比特长度的的字符串“abc”，经过填充处理过后的数据输入 M，SHA256 输出为 N，则

M=512'h61626380000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000018；

N=256'hBA7816BF8F01CFEA414140DE5DAE2223B00361A396177A9CB410F
F61F20015AD；

SHA256 的 VCS 仿真结果图如图 4-3 所示：

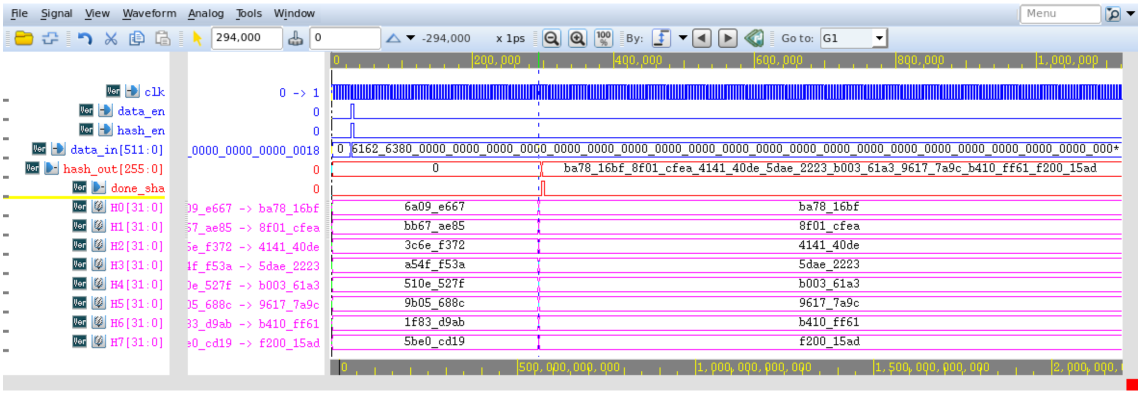


图 4-3 SHA256 仿真图

从仿真图中可以看出，放着结果与测试向量的计算结果一致，故 SHA256 算法的硬件设计是正确的。

4.1.3 HMAC-SHA256 算法

HMAC 是一种使用密码学 Hash 函数进行消息认证的机制，可以与任何迭代批准的密码散列函数一起使用，并与共享秘钥结合使用，来完成消息来源及完整性认证。HMAC-SHA256 则是指使用 SHA256 密码散列函数来进行消息认证的算法。消息发送者通过 HMAC-SHA256 算法产生一个消息验证码 MAC(Message Authentication Code)。MAC 值是通过压缩秘钥和消息输入而形成的。MAC 通常与消息发送者一起发给消息接受者。接收端使用于发送端相同的秘钥和 HMAC 函数计算接收到的消息 MAC，并与接收到的 MAC 值进行比较，如果两个值一样，则表示正确接收到消息，表明发送端的消息来源是可靠的，是完整的。

HMAC 算法的定义如式(3-19)所示：

$$HMAC=(K,text)=H((K_0 \oplus IPAD)||H((K_0 \oplus OPAD)||text)) \quad (3\text{-}19)$$

其中：K 是共享秘钥；text 是消息输入；K_0是扩展后的秘钥；OPAD 为输出填补常数，其 OPAD= 64{0x5C};IPAD 是输入填补常数，IPAD=64{0x36}；⊕表示异或运算，||表示拼接运算。HMAC-SHA256 算法的具体步骤如图 4-4 所示:

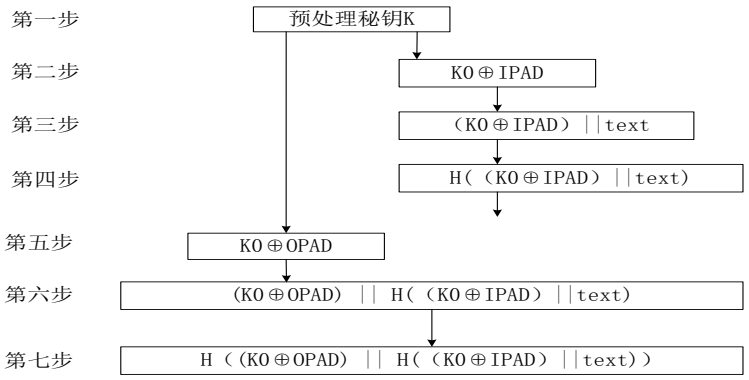


图 4-4 HMAC 算法

(1)第一步：预处理秘钥 K，将秘钥 K 处理成 512bit 的数据得到K_0，当秘钥 K 的长度等于 512 比特时，则不做处理；当 K 的长度大于 512 比特时，则需要对 K 进行一次 Hash 运算；如果 K 的长度小于 512bit，则需要在 K 的末尾添加‘0’将 K 扩展成 512bit 数据。

(2)第二步：将预处理过后的秘钥K_0与输入填补常数 IPAD 进行异或。

(3)第三步：将第二步处理后的结果与消息输入做异或运算。

(4)第四步：将第三步处理的结果作为第一次 SHA256 算法的输入，经过计算得到 256bit 的输出。

(5)第五步：将第一步预处理后得到的秘钥K_0与输入填补常数进行异或。

(6)第六步：将第五步处理的结果和第四步得到的 256bit 数据进行拼接。

(7)第七步：将第六步得到的结果作为第二次 SHA256 算法的输入，通过计算，便可以最终得到 256bit 的消息摘要输出。

4.1.4 HMAC-256 算法设计与仿真

通过对 HMAC-SHA256 算法的分析可知，HMAC-SHA256 算法总共使用了两次 SHA256 运算，故在设计时采用状态机的设计方式复用 SHA256 算法模块来减少电路消耗的面积。其具体的 HMAC 算法的设计框图如图 4-5 所示。

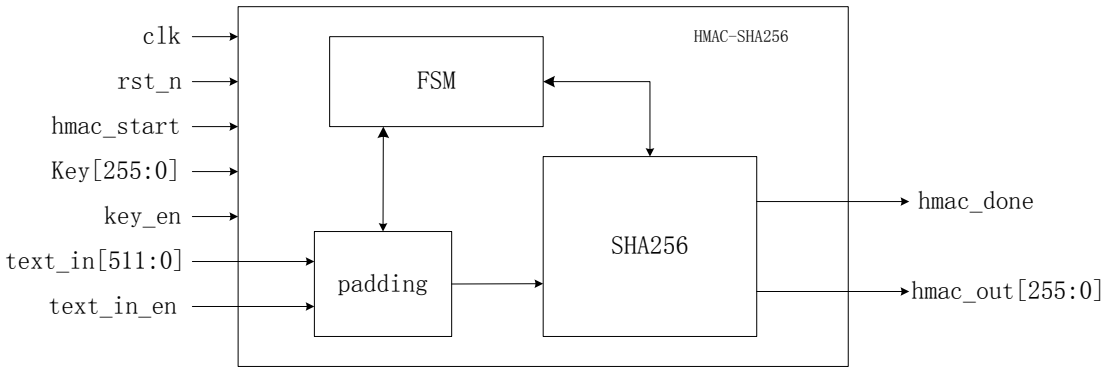


图 4-5 HMAC 算法

HMAC-SHA256 算法的接口信号表如表 4-3 所示：

表 4-3 HMAC-SHA256 模块接口信号表

接口信号名称	数据位宽	接口方向	信号描述
clk	1	输入	时钟信号
rst_n	1	输入	复位信号，低电平有效
hmac_start	1	输入	HAMC-SHA256 计算开始信号
key_en	1	输入	秘钥输入有效信号
text_in	512	输入	512 比特消息输入
text_in_en	1	输入	消息输入有效信号
hmac_out	256	输出	HMAC-SHA256 输出结果
hamc_done	1	输出	HMAC-SHA256 计算完成信号

设计 HAMC-SHA256 算法时，主要是分为 padding(填充)模块，SHA256 计算模块和状态机控制模块。

填充模块主要是对输入秘钥 K 和输入消息进行填充，设计时选用的共享秘钥长度为 256 比特，该秘钥长度小于 512 比特，故需要在输入秘钥后填充 256 个‘0’，得到扩展后的秘钥K_0。由于设计时的输入消息为 512 比特，故也需要对其进行填充，在第一次 SHA256 计算时，第一次 SHA256 的消息输入长度为 1024bit，故对其进行填充，其总长度为 1536bit，可以分为 3 个 512bt 的块，第一个 512bit 的块为扩展后的秘钥K_0与 IPAD 的异或后的值，第二个 512bit 的块为消息输入，第三个块为填充块{1'b1,447'd0,64'd1024}。第二次进行 SHA256 的计算时，其总的输入长度为 768bit，经过填充后的长度为 1024bit，可分为 2 个 512bit 块，其中第一个块为扩展后的秘钥K_0与 OPAD 的异或后的值,第二个块为{out1,1'b1,191'd0,64'd768}，其中 out1 为第一次 SHA256 完整计算的结果。

SHA256 模块主要是完成 SHA256 计算。

使用 VCS 软件对 HMAC-SHA256 算法进行仿真，选取的测试向量为：

key=256'h000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f；

text_in=512'h53616d706c65206d65737361676520666f72206b65796c656e3c626c6f636b6c656e8000000000000000000000000000000000000000000000000000000000000310；

hash_out=256'h392a111e5cce9c5793821eaf26ebac0a7315f2fd389ce71a60b00f5145cde7f8；

得到的结果如图 4-6 所示：

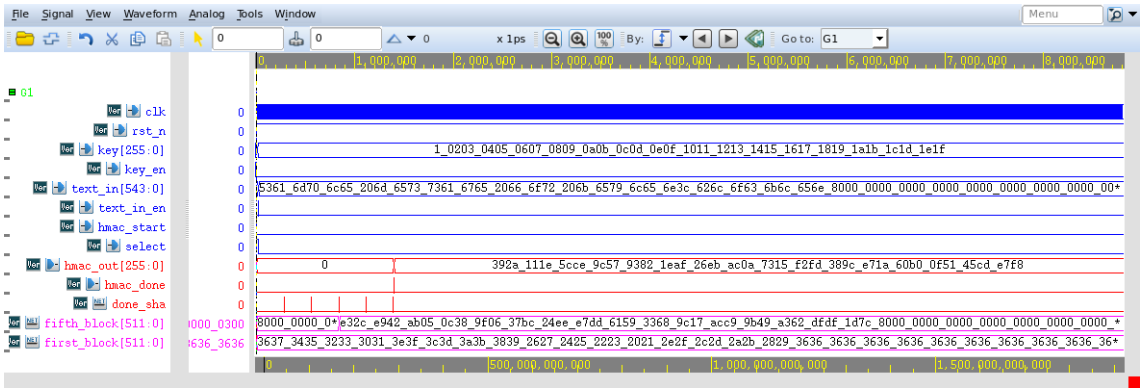


图 4-6 HMAC-SHA256 状态机

由仿真结果可知，结果正确

将本文的设计结果与文献[45]进行对比得到的结果如下表 4-4 所示：

表 4-4 HMAC-SHA256 实现对比

文献	最高时钟/MHz	吞吐率/Mbps	逻辑资源/LUTs
文献[45]	129.75	443.28	3657
本文	154	600.32	3081

从对比结果中可以看到，本文的设计实现的 HMAC-SHA256 算法的最高时钟更好，并且使用逻辑资源更少。

4.2 消息加解密算法

AES(Advanced Encryption Standard)是高级加密标准，采用分组加密体制，相比 DES(Data Encryption Standard)，3DES(Triple Data Encryption Standard)，AES 算法的安全性能更高，运算速度更快，资源消耗更小。故本设计选用 AES 算法来对

数据进行加密和解密。

4.2.1 AES 算法数学基础

由于 AES 算法过程中的轮单元迭代过程中的每个字节数据操作都涉及到有限域内的加法和乘法运算，故接下来介绍有限域及其相关运算法则。有限域 F 是指只含有有限个元素的域，也称为 Galois 域；而有限域又分为素域 GF(p)和特征值为 2 的二元域 GF（2^n)。AES 算法中涉及到的有限域运算主要是二元域 GF(2^8)。下面将介绍二元域 GF(2^8)上的运算规则

二元域 GF(2^8)上的加法运算：主要是将多项式对应的 x 幂系数进行异或。假设任意 2 个二元域 GF(2^8)上的 n 次多项式分别为如式(4-12)和式(4-13)所示，

$$f(x)=a_7x^7+a_6x^6+a_5x^5+a_4x^4+a_3x^3+a_2x^2+a_1x+a_0 \tag{4-12}$$

$$g(x)=b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b_0 \tag{4-13}$$

其中$a_i, b_i\in\{0,1\}$, $f(x),g(x)\in GF(2^8)$，则对其进行二元域 GF(2^8)上的加法运算后的结果 m(x)的急计算如式(4-14)所示：

$$m(x)=f(x)+g(x)=\sum_0^7(a_i+b_i)x^i \tag{4-14}$$

若 $f(x)=x^7+x^4+x^3+x^2+x+1\in GF(2^8)$，$g(x)=x^6+x^4+x^3+x^2+x+1\in GF(2^8)$，两个有限域内GF($2^8$)多项式相加的结果如式(4-15)所示：

$$f(x)+g(x)=(x^7+x^4+x^3+x^2+x+1)+(x^6+x^4+x^3+x^2+x+1)=x^7+x^6 \tag{4-15}$$

上诉计算结果以二进制表示则为{10011111}⊕{01001111}={11000000}。

二元域上的 GF(2^8)上的乘法运算：假设任意 2 个二元域 GF(2^8)上的 n 次多项式分别为如式(3-12)和式(3-13)所示，则对其进行二元域 GF(2^8)上的加法运算后的结果 m(x)的计算如式(4-16)所示

$$m1(x)=(f(x)\cdot g(x))\bmod(x^8+x^4+x^3+x+1) \tag{4-16}$$

由于f(x)·g(x)的结果的最高次数为 14，但是有限域 GF(2^8)中的最高次数必须小于 8。所以需要将相乘后的结果对多项式$m2(x)=x^8+x^4+x^3+x+1$进行求模，则可以乘法运算的结果的最高次数小于 8。假设多项式$a(x)=x^6+x^4+x^2+x+1\in GF(2^8)$,$b(x)=x^7+x+1$则$c(x)=(a(x)\cdot b(x))\bmod(x^8+x^4+x^3+x+1)=x^{13}+x^{11}+x^9+x^8+x^6+x^4+x^3+1)\bmod(x^8+x^4+x^3+x+1)$。在二元域中还存在着一种乘法叫做 x 乘法，记为 xtime()，当 f(x)的表达式如式(3-12)时，即为：$xtime(f(x))=x\cdot f(x) modm1(x)$，x·f(x)的操作如式(4-17)所示：

$$x \cdot f(x)=a_7x^8+a_6x^7+a_5x^6+a_4x^5+a_3x^4+a_2x^3+a_1x^2+a_0x \quad (4\text{-}17)$$

当a_7的值为 0 时，则$xtime(f(x)) = x \cdot f(x) mod m1(x) = x \cdot f(x) = a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x$；

当 a7 的值为 1 时，则$xtime(f(x)) = x \cdot f(x) mod m1(x) = (a_7x^8 + a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x) mod (x^8 + x^4 + x^3 + x + 1)$，这相当于与 16 进制数{1b}进行异或运算。所以 xtime(f(x))的计算用二进制可以用式(4-18)进行表示：

$$x \cdot f(x) = \begin{cases} a_6a_5a_4a_3a_2a_1a_00 & a_7 = 0 \\ (a_6a_5a_4a_3a_2a_1a_00) \oplus (00011011) & a_7 = 1 \end{cases} \quad (4\text{-}18)$$

4.2.2 AES-CTR 算法

NIST 在 2000 年 3 月通过召开两次会议，最后确定了 AES 算法的基本工作模式，分别为密码反馈 CFB(Cipher Feedback)模式，输出反馈 OFB(Output Feedback)模式，电子密码本 ECB(Electronic Codebook)模式，分组密码链接 CBC(Cipher Block Chaining)模式，计数器 CTR(Counter)模式 [46]。这 5 种模式的 AES 算法可以适用在不同场合，ECB 和 CTR 可以对数据进行独立处理，并且可以并行处理，所以运算速度很快，属于非反馈模式 [47]。而 CBC，CFB，OFB 这三种模式均为反馈模式，在计算过程中需要按照顺序进行计算，不能并行处理，只有当前消息块计算完过后，才可以进行下一个消息块的计算，所以数据处理的速度偏低 [49]。本文为了实现对数据的高效处理，所以选择 CTR 模式来对数据进行加解密。

AES 算法的 CTR 模式记为 AES-CTR，此模式是一种通过加密计算器的值来生成秘钥流 [50]。每一个消息块对应一个逐次累加的计数器，且每个计数器的值必须不一样，以此来来提高 AES-CTR 加密算法安全性，然后加密这些唯一的计数器的值来得到秘钥流。将秘钥流和明文分组进行异或，便可以得到密文，这就加密过程；将秘钥流和输入密文进行异或，便得到明文，这就是解密过程。AES-CTR 算法的加解密流程如图 4-7 所示：

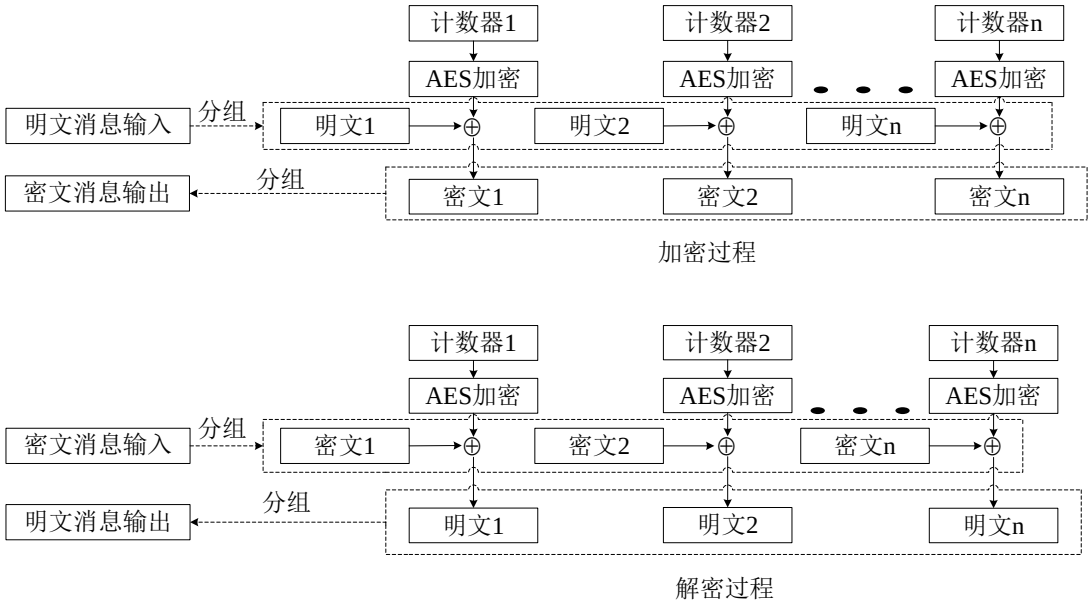


图 4-7 AES-128 加密流程图

其中，计数器的值为 n 个 128bit 的值，明文消息输入为 n 个 128bit 的消息块，密文为 n 个 128bit 的消息块，AES 加密算法使用的是 AES-128 算法。

NIST 在 2001 年正式发布 AES 算法标准，并且在 2001 年 11 月 26 日正式有效。AES。AES 算法标准规定该算法的明文和密文均为 18 比特，但为了适应在不同场合的对加密算法的需求，AES 算法又根据不同秘钥长度，分为了 AES-128，AES-192，AES-256。AES-128 算法的秘钥长度为 128bit，加密轮数为 10 轮；AES-192 算法的秘钥长度为 192 比特，加密轮数为 12 轮；AES-256 算法的秘钥长度为 256bit，加密轮数为 14 轮。本文选取 AES-128 算法来进行数据加解密。

AES-128 加密算法的的加密过程有 10 轮迭代，其具体加密流程图如图 4-8 所示：

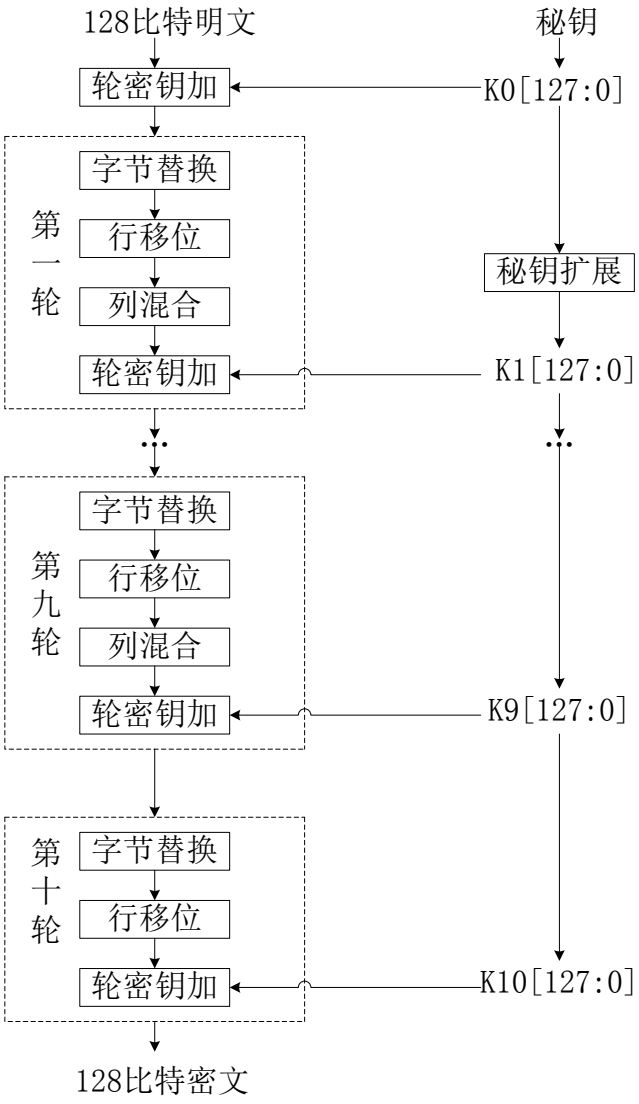


图 4-8 AES-128 加密流程图

其中秘钥 Ki(0≤i≤10)均为 128 比特，K0 为初始秘钥输入，而 Ki(1≤i≤10)是第 i 轮的 128bit 扩展秘钥。由 K0 得到 Ki(1≤i≤10)的过程称为秘钥扩展。AES 加密过程的前 9 轮的每一轮计算包括四个步骤，分别为字节替换(Sub_Byte)，行移位(Shift_Row)，列混合(Column_Mix)，轮密钥加(Round_Key_Add)。在第 10 轮的计算中，则只包括三个步骤，分别为字节替换(Sub_Byte)，行移位(Shift_Row)，轮密钥加(Round_Key_Add)，省略去了列混合操作。

AES 加密过程中的操作主要是以字节为单位进行运算的，当 AES 进行加密时，明文输入为 128bit 数据，也就是 16 个字节，在进行 10 轮计算的过程中，是以状态矩阵的方式进行进行操作的，假设明文输入为 A[127:0]，则其状态矩阵的表示方式如图 4-9 所示：

A[127:120]	A[95:88]	A[63:56]	A[31:24]
A[119:112]	A[87:80]	A[55:48]	A[23:16]
A[111:104]	A[79:72]	A[47:40]	A[15:8]
A[103:96]	A[71:64]	A[39:32]	A[7:0]

图 4-9 消息输入状态矩阵表示图

AES-128 加密算法主要有 10 轮轮函数迭代计算和秘钥扩展计算组成，其中轮函数迭代计算包括字节替换，行移位，列混合，轮密钥加。下面将具体介绍这些计算的原理。

(1)字节替换

字节替换主要是对输入数据进行非线性运算，非线性运算主要包含数学上的两个复杂变化：先在有限域 GF(2^8)内进行乘法逆元映射，然后再针对每一个字节做仿射映射运算。这里本文对逆元映射和仿射映射不做具体描述，因为一般为了实现快速字节替换运算和减少硬件资源消耗，一般采取 S 盒(Sbox)的形式来进行字节替换操作。假设字节替换前的状态矩阵输入为 A，字节替换后的状态矩阵输出为 B,则字节替换操作的流程图如图 4-10 所示：

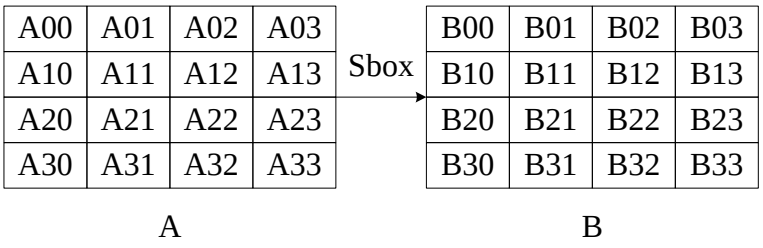


图 4-10 字节替换状态转移图

(2)行移位

行移位主要是针对字节替换后的状态矩阵的每一行进行操作，具体操作为将 4x4 的状态矩阵的第一行向左移动 0 位，第二行向左移动 1 位，第三行向左移动 2 位，第四行向左移动 3 位。行移位的具体的状态转移图如图 4-11 所示：

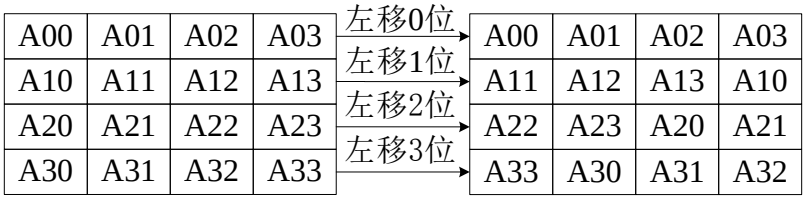


图 4-11 行移位状态转移图

(3)列混合

列混合是对行移位后的状态矩阵中的每一列进行操作，其具体操作为将状态矩阵的每一列在有限域 $GF(2^8)$内与多项式 $h(x)=(03)x^3+(01)x^2+(01)x+(02)$相乘，然后进行模$(x^4+1)$运算，假设列混合计算的输入的状态矩阵的每一列为 A(x)，进过列混合计算过后的输出的状态矩阵的每一列为 C(x),则 $C(x)=B(x)\cdot h(x)$，化成矩阵的形式如式(4-19)所示：

$$\begin{bmatrix} C0c \\ C1c \\ C2c \\ C3c \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} A0c \\ A1c \\ A2c \\ A3c \end{bmatrix}, 0 \le c \le 3 \tag{4-19}$$

列混合变化的状态转移图如图 4-12 所示：

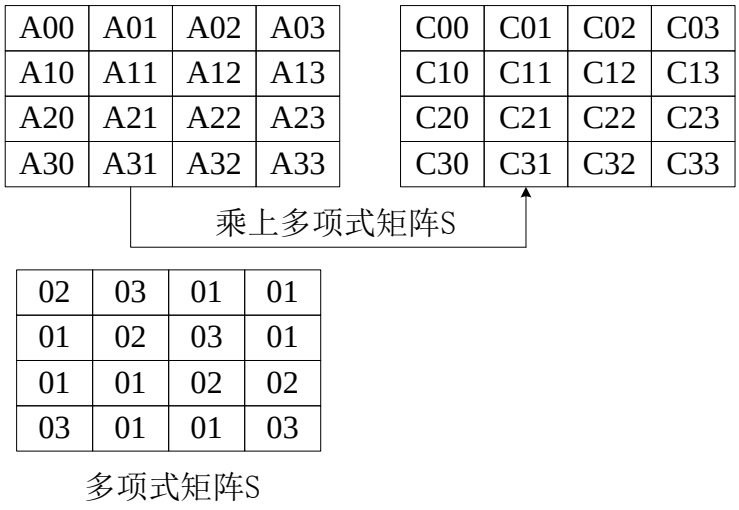


图 4-12 字节替换状态转移图

因为 AES-128 算法的第 10 轮计算中列混合计算并不会增加 AES-128 算法的加密的安全性，反而降低了算法的运行效率，所以第 10 轮轮变化计算省略了列混合计算。

(4)轮密钥加

轮密要加主要是将前 9 轮轮迭代计算中列混合变化后的状态矩阵和第 10 轮轮迭代计算中行移位变换后的状态矩阵分别与秘钥扩展矩阵后的秘钥按位进行异或，运算结果作为下一轮轮函数迭代的输入。若用 D 表示轮密钥加后的状态矩阵，K 表示轮秘钥矩阵，A 表示轮密钥加的输入状态矩阵，则轮密钥加的公式如式(3-28)所示：

$$D=A \oplus Ki, 0 \le i \le 10 \tag{3-28}$$

轮密钥加的状态转移图如图 4-13 所示：

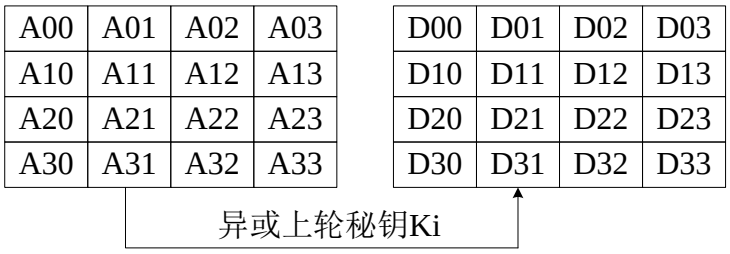


图 4-13 轮密钥加状态转移图

(5)秘钥扩展

AES-128 算法共有 10 轮轮函数迭代计算，而这 10 轮轮函数迭代计算中都需要用到不同的秘钥，其每一轮的子秘钥由 4 个字组成，秘钥扩展的第一个秘钥即 K0，就是初始秘钥输入 K[127:0]，将其表示为 Ka0，Ka1，Ka2，Ka3，然后基于 Ka0，Ka1，Ka2，Ka3 来计算以得到其余 10 个子秘钥，可以用 Kai，Ka(i+1)，Ka(i+2)，Ka(i+3)来表示每一轮的轮秘钥，其中$0 \le i \le 44$。当$i \ge 4$时，Ka(i)的计算与 Ka(i-4)和 Ka(i-1)有关。

当 i 的值是 4 的倍数时，秘钥扩展只需要通过异或运算便可以得到结果，其公式如式(3-29)所示：

$$Kai=Ka(i-1)\oplus Ka(i-4) \tag{3-29}$$

当 i 的值不为 4 的倍数时，则经过以下三个步骤来计算 Kai 的值：

步骤一：字节循环 Rotword()。将 Ka(i-1)中的字进行移位处理，将其循环左移一个字节。假设 Ka(i-1)中的字节表示为[a0，a1，a2，a3]，则经过字节循环过后的输出为[a1，a2，a3，a0]。

步骤二：字节替代 Subword()。利用 S 盒进行字节替换。

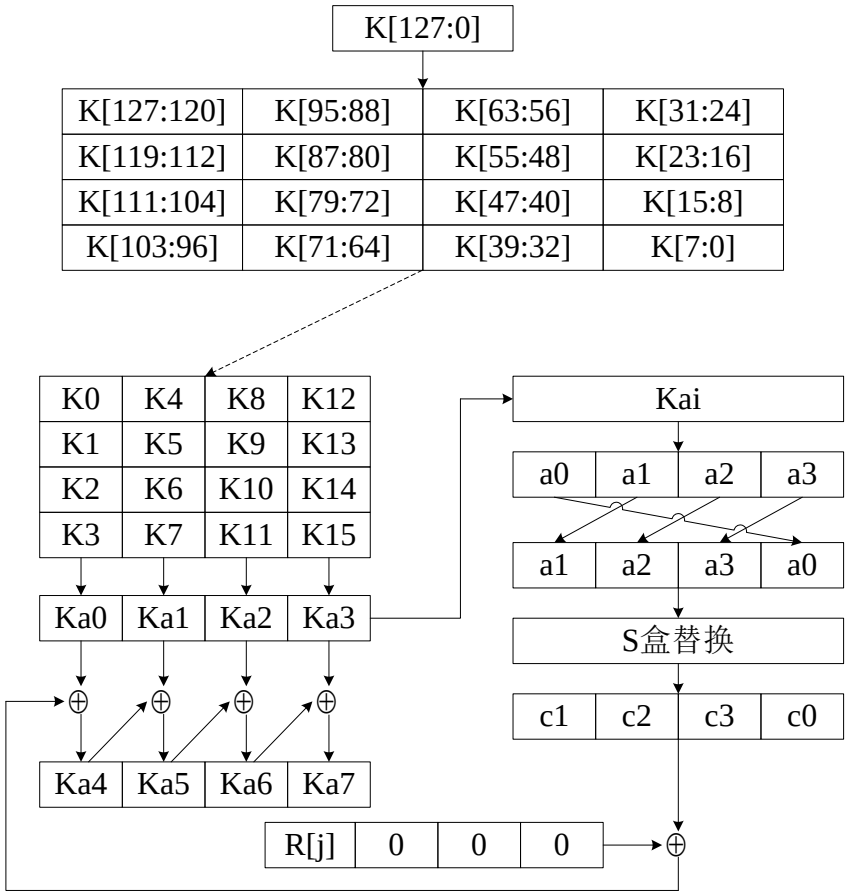
步骤三：轮常量异或。将轮常量 R[j]的值和字节替换后的结果进行异或运算，轮常量 R[j]的值如表 4-5 所示，最后再将 Ka(i-4)的值与异或后的结果进行异或运算。将上述三个步骤进行整理得到当 i 不等于 4 的倍数时的 Kai 的计算公式如式(3-30)所示：

$$Kai=Subword(Rotword(ka(i-1))\oplus R[j]\ \oplus Ka(i-4) \tag{3-30}$$

表 4-5R[j]对应值

j	1	2	3	4	5	6	7	8	9	10
R[j]	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80	0x1b	0x36

秘钥扩展的完整流程如图 4-14 所示：



秘钥扩展

图 4-14 轮密钥加状态转移图

4.2.3 AES-CTR 算法设计及仿真

由于是采用的 AES-CTR 算法，其加密和解密的硬件设计是一样的，进行 AES-CTR 加密时，输入为明文，输出为密文；进行 AES-CTR 解密时，只需要将 AES-CTR 的输入变为密文，则输出即为明文。下面介绍 AES-CTR 加密算法的具体设计。

本文采用流水线的设计结构来进行 AES-CTR 的算法设计，AES-CTR 算法的总体设计框图如图 4-15 所示：

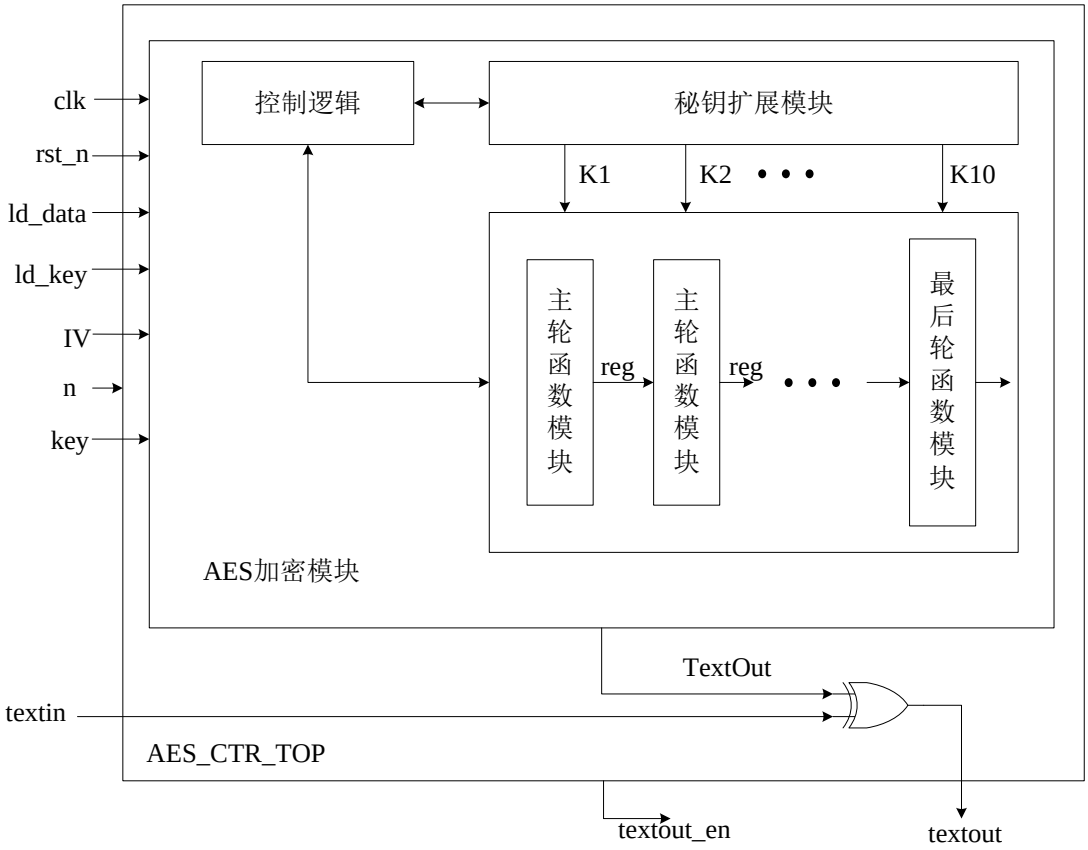


图 4-15 AES-CTR 算法总体设计框图

AES-CTR 算法的接口信号描述如表 4-6 所示：

表 4-6AES-CTR 模块接口信号表

接口信号名称	数据位宽	接口方向	信号描述
clk	1	输入	时钟信号
rst_n	1	输入	复位信号，低电平有效
ld_data	1	输入	明文数据有效信号
ld_key	1	输入	秘钥有效信号
textin	128	输入	明文输出
key	128	输入	输入秘钥
IV	64	输入	新鲜性参数
n	64	输入	输入的消息的长度
textout	128	输出	密文输出
textout_en	1	输出	密文输出有效信号

下面以此介绍各个模块的设计方案。

(1)S 盒模块的设计

S 盒模块的设计主要是通过查找表的形式来实现，输入和输出均为 8bit 数据，将输入 8bit 的数据的高四位看作字节替换过程中的行数，将输入 8bit 数据的第四位数据看作字节替换过程中的列数 [50]。由行数和列数可以唯一确定经过 S 盒变换后的数据输出，使用查找表来实现 S 盒，具体的查找表的值如表 4-7 所示：

表 4-7 字节替换查找表对应值

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

S盒的设计框图如图 4-16 所示：

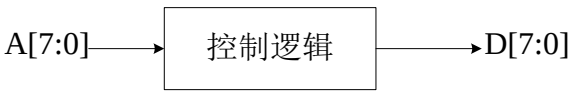


图 4-16 S 盒设计框图图

其中 A 为 S 盒 8bit 数据输入，D 为经过 S 盒变换后的 8bit 输出。

(2)主轮函数模块

主轮函数模块主要是完成 AES 加密算法的前 9 轮的每一轮的轮函数计算，每一轮包含字节替换，行移位，列变换，轮密钥加计算。在设计此模块时，首先将字节替换计算和行移位计算利用 S 盒放在一起计算，以此来缩减关键路径，然后在行移位和列混合计算间插入一级寄存器，这样便以流水线的方式来加快了轮函数的计算，以此来提高算法的运行效率。主轮函数模块的设计框图如图 4-17 所示：

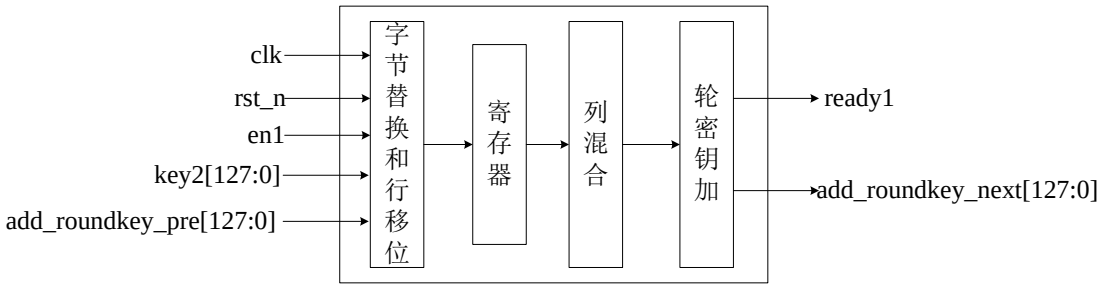


图 4-17 主轮函数模块框图

主轮函数模块的接口信号描述如表 4-8 所示：

表 4-8 主轮函数计算模块接口信号表

接口信号名称	数据位宽	接口方向	信号描述
clk	1	输入	时钟信号
rst_n	1	输入	复位信号，低电平有效
en1	1	输入	主轮函数计算使能信号
key2	1	输入	输入的扩展秘钥
add_roundkey_pre	128	输入	当前轮函数计算输入
add_roundkey_next	128	输出	当前轮函数计算输出
ready1	1	输出	当前轮函数计算完成信号

其中字节替换和行移位计算，列混合计算，轮密钥加计算均是以纯组合逻辑实现的。

(3)最后轮函数模块

最后轮函数模块主要是完成 AES-128 算法的最后一轮的计算，其具体为字节替换，行移位和轮密钥加计算。在设计时，将字节替换和行移位计算通过例化 S 盒的方式来放在一起计算，以此来缩短关键路径，然后在行移位和轮密钥加计算间插入一级寄存器，以流水线的设计方式来提高最后轮函数模块的计算速度。最后轮函数计算模块框图如图 4-18 所示：

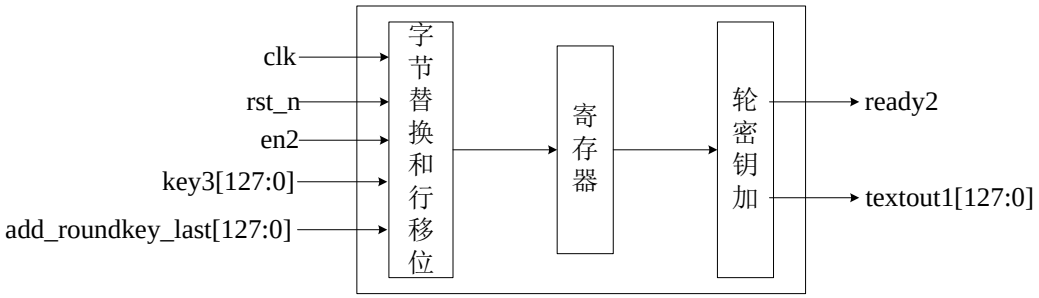


图 4-18 最后轮函数计算模块框图

最后轮函数模块的接口信号描述如表 4-9 所示：

表 4-9 最后轮函数计算模块接口信号表

接口信号名称	数据位宽	接口方向	信号描述
clk	1	输入	时钟信号
rst_n	1	输入	复位信号，低电平有效
en2	1	输入	最后轮函数计算使能信号
key3	1	输入	输入的扩展秘钥
add_roundkey_last	128	输入	最后轮函数计算输入
textout1	128	输出	最后轮函数计算输出
ready2	1	输出	最后轮函数计算完成信号

其中字节替换和行移位计算，轮密钥加计算均是以纯组合逻辑实现的。

(4)秘钥扩展模块

在加密和解密时，每一轮轮函数的计算都要用到扩展秘钥，可通过秘钥扩展模块来完成 10 轮变换所需要的子秘钥。在设计秘钥扩展模块时，采用纯组合逻辑进行设计，秘钥扩展模块的设计框图如图 4-19 所示：

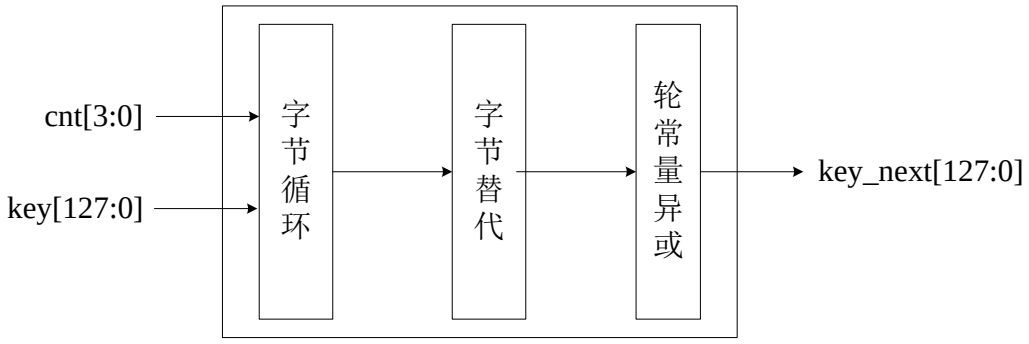


图 4-19 秘钥扩展模块设计框图

其中 cnt[3:0]信号为计数器信号，利用从 0~9 的计数器的值，可以分别得到扩展秘钥 K1~K10。

(5)AES 加密模块设计

AES-128 加密共有 10 轮计算方式，为了提高数据处理的速度，本文采用混合流水线结构来进行 AES-128 加密算法设计。本文选择在每一轮轮函数的计算间插入一级寄存器，然后在每一轮的轮函数的行移位和列变换计算中间插入一级寄存器。AES 加密算法的总体设计框图如图 4-20 所示

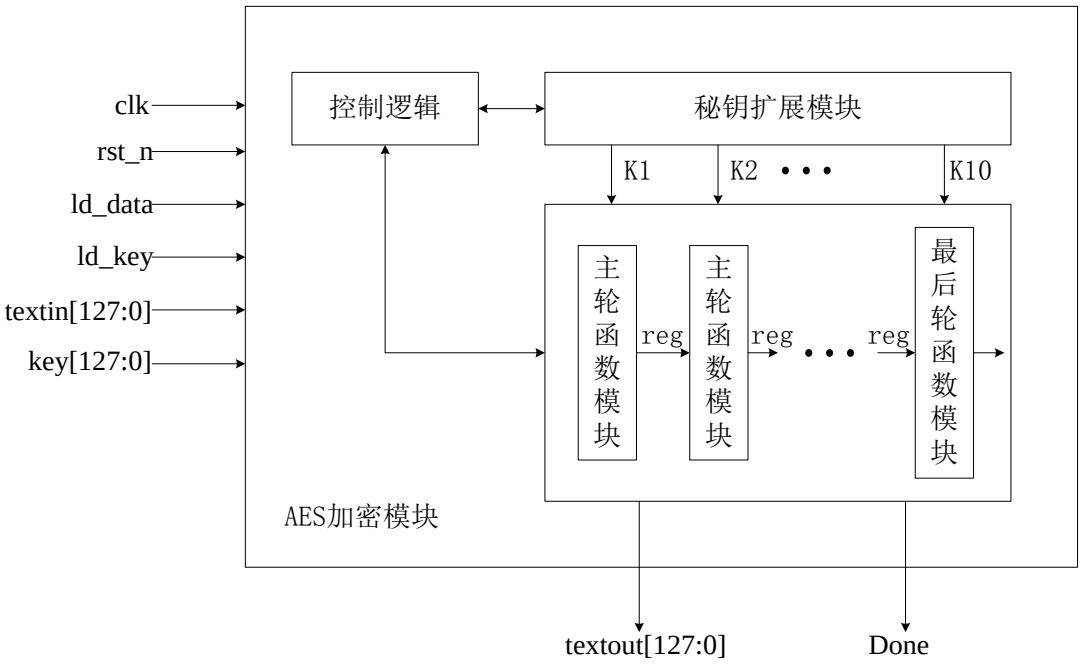


图 4-20 AES 加密模块设计框图

AES 加密模块的接口信号描述如表 4-10 所示：

表 4-10AES 加密模块接口信号表

接口信号名称	数据位宽	接口方向	信号描述
clk	1	输入	时钟信号
rst_n	1	输入	复位信号，低电平有效
ld_data	1	输入	输入明文有效信号
ld_key	1	输入	输入秘钥有效信号
textin	128	输入	输入明文
key	128	输入	输入秘钥
textout	128	输出	输出密文
Done	1	输出	密文输出有效信号

对 AES 加密模块进行 VCS 仿真，选取测试向量为：

明文输入：128'h6bc1bee22e409f96e93d7e117393172a；

秘钥输入：128'h2b7e151628aed2a6abf7158809cf4f3c；

密文输出：128'h3ad77bb40d7a3660a89ecaf32466ef97；

AES-128 加密模块的仿真图如图 4-21 所示:

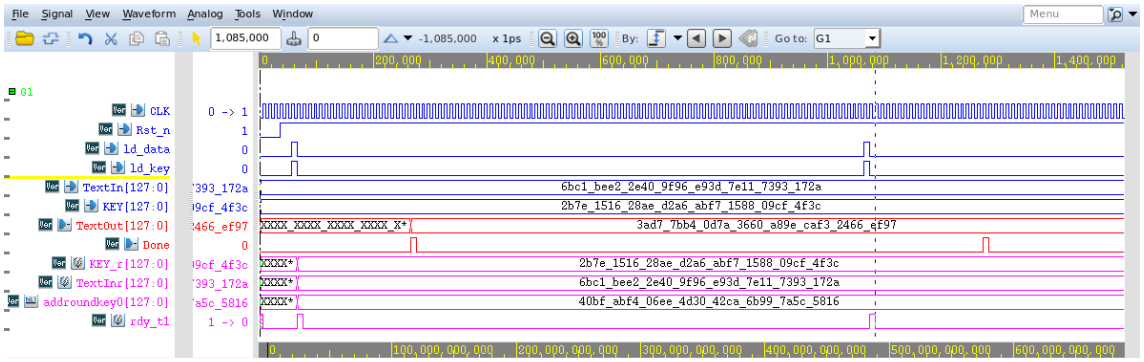


图 4-21 AES 加密模块仿真图

由仿真图中可以看到，当秘钥有效信号 ld_key 和密文输入有效信号 ld_data 拉高后，经过 21 个时钟周期，可以得到密文输出，图中的密文输出结果与测试向量中的结果是一样的，所以 AES-128 加密算法设计正确。

利用 VCS 对 AES-CTR 模块进行仿真，选取测试向量：

AES-CTR 加密：

明文输入：128'h6bc1bee22e409f96e93d7e117393172a;

计数器输入：128'hf0f1f2f3f4f5f6f7f8f9f10fafbfcfdfeff;

秘钥输入：128'h2b7e151628aed2a6abf7158809cf4f3c;

密文输出：128'h874d6191b620e3261bef6864990db6ce;

AES-CTR 解密：

密文输入：128'h874d6191b620e3261bef6864990db6ce;

计数器输入：128'hf0f1f2f3f4f5f6f7f8f9f10fafbfcfdfeff;

秘钥输入：128'h2b7e151628aed2a6abf7158809cf4f3c;

明文输出：128'h6bc1bee22e409f96e93d7e117393172a;

则 AES-CTR 加密的仿真图如图 4-22 所示，AES-CTR 解密的仿真图如图 3-31 所示：

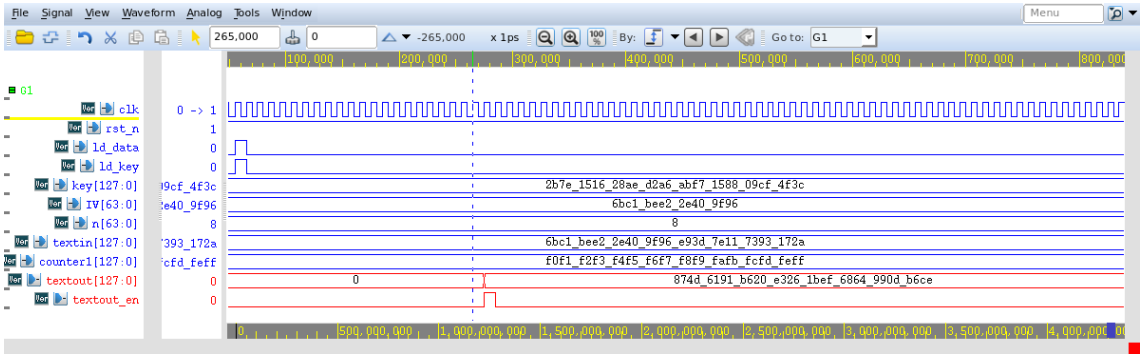


图 4-22 AES-CTR 加密模块仿真图

4.3 本章小结

然后介绍了安全散列算法 SHA256 的算法原理，对 SHA256 算法进行硬件设计，并选取 NIST 的测试向量对 SHA256 算法进行仿真，验证了设计的正确性。接着对基于 SHA256 的秘钥消息认证算法 HMAC-SHA256 的原理进行了介绍，接着为了减少资源的消耗，采取复用 SHA256 算法的方式来是实现 HMAC-SHA256 的硬件设计，然后选取 NIST 提供的测试向量对 HMAC-SHA256 算法的仿真，验证了其正确性。最后介绍了 AES-CTR 算法的数学基础和其加解密原理，为了加快运算速度，采取流水线的设计方式实现了 AES-CTR 算法的设计，并选取 NIST 提供的测试向量对 AES-CTR 进行仿真，验证了其加解密功能的正确性。

第五章 FPGA 原型验证

本文所设计的 SoC 中的核心模块：ECDH 算法，HAMC-SHA256 算法，AES-CTR 算法的模块级的仿真验证已经在第二章和第三章介绍，而其他的模块均是采用的成熟的 IP。经统计，验证在 SoC 设计流程中所需要花的时间约占整个设计周期的 70%。为了加快仿真验证的进度和更加接近真实情况的测试整个 SoC 系统，我们搭建 FPGA 原型验证环境来进行 FPGA 上板验证。

5.1 开发板介绍

FPGA 原型验证平台的与 FPGA 开发板的选取有着密切的关系，需要关注 FPGA 的选型是否满足要求来验证待验证的 ASIC 或者 SoC。FPGA 的选型主要从以下几个方面来考量：

(1)容量：主要考虑的就是 FPGA 的一个逻辑容量，存储容量等。需要 FPGA 的逻辑资源要大于待验证的 SoC 的逻辑资源。

(2)时钟：一般 SoC 中会包含多个时钟域，这就需要多个时钟模块来进行支持。同时要考虑支持的时钟频率大小。

(3)接口：主要考虑接口的数量，和 SoC 中所需要的特定接口。

本设计中用到的 FPGA 原型验证平台所选用的主芯片为 Xilinx VirtexUltraScale 系列 FPGA XCVU440-FLGA2892，它具有丰富的逻辑资源(5540850 个逻辑单元)，大量的 IO，且板级可以提供 6 对可编程差分时钟，时钟频率为 0.2MHz~350MHz，满足所待验证的 SoC 的要求。VU 系列的资源对比情况如下图 5-1 所示：

	VU065	VU080	VU095	VU125	VU160	VU190	VU440
System Logic Cells	783,300	975,000	1,176,000	1,566,600	2,026,500	2,349,900	5,540,850
CLB Flip-Flops	716,160	891,424	1,075,200	1,432,320	1,852,800	2,148,480	5,065,920
CLB LUTs	358,080	445,712	537,600	716,160	926,400	1,074,240	2,532,960
Maximum Distributed RAM (Mb)	4.8	3.9	4.8	9.7	12.7	14.5	28.7
Block RAM Blocks	1,260	1,421	1,728	2,520	3,276	3,780	2,520
Block RAM (Mb)	44.3	50.0	60.8	88.6	115.2	132.9	88.6
CMT (1 MMCM, 2 PLLs)	10	16	16	20	28	30	30
I/O DLLs	40	64	64	80	120	120	120
Maximum HP I/Os[1]	468	780	780	780	650	650	1,404
Maximum HR I/Os[2]	52	52	52	104	52	52	52
DSP Slices	600	672	768	1,200	1,560	1,800	2,880
System Monitor	1	1	1	2	3	3	3
PCIe Gen3 x8	2	4	4	4	4	6	6
150G Interlaken	3	6	6	6	8	9	0
100G Ethernet	3	4	4	6	9	9	3
GTH 16.3Gb/s Transceivers	20	32	32	40	52	60	48
GTY 30.5Gb/s Transceivers	20	32	32	40	52	60	0
Transceiver Fractional PLLs	10	16	16	20	26	30	0

图 5-1 VU 系列 FPGA 资源对比图

该开发板的实物如图 5-2 所示：

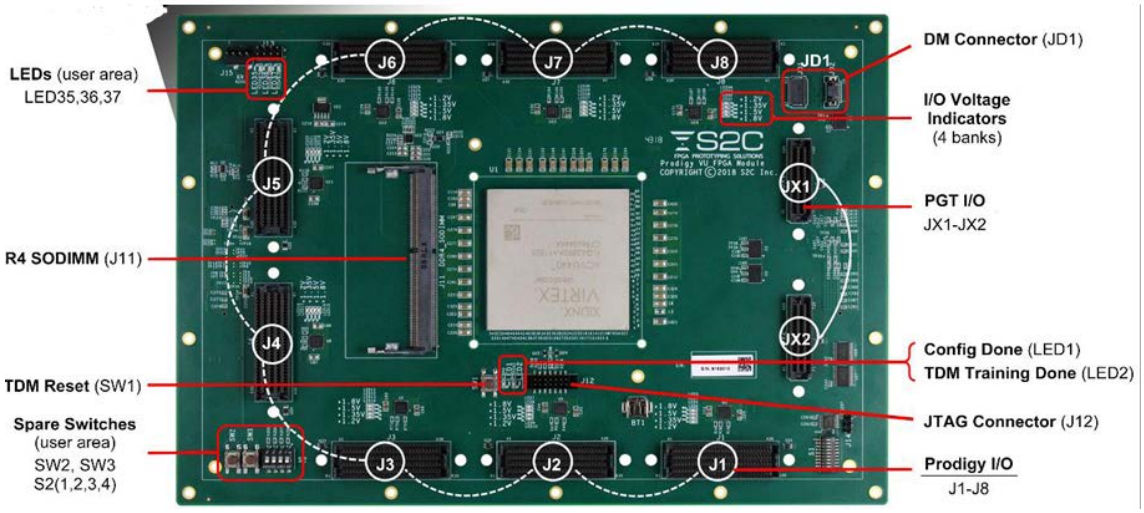


图 5-2VU440 FPGA 实物图

5.2 FPGA 验证流程

FPGA 的完整验证流程主要有以下 5 步：

(1)设计转换

(2)设计约束

(3)综合和布局布线

(4)时序分析

(5)生成比特流上板调试

5.2.1 设计转换

在进行 FPGA 原型验证时，为了验证的需要，需要将 ASIC 风格的代码转换为 FPGA 风格的代码。

在将 ASIC 代码转换为 FPGA 代码时，主要是从以下几个方面去进行代码转换：

(1)memory 的替换

主要是将 ASIC 风格的 memory 转换为 FPGA 风格的 memory。ASIC 的 memory 通常是代工厂所提供的 memory compiler 来进行定制的；而 FPGA 的 memory 则是利用 vivado 工具的 Block Memory Generator 来生成的，其具体生成示意图如图 5-3 所示：

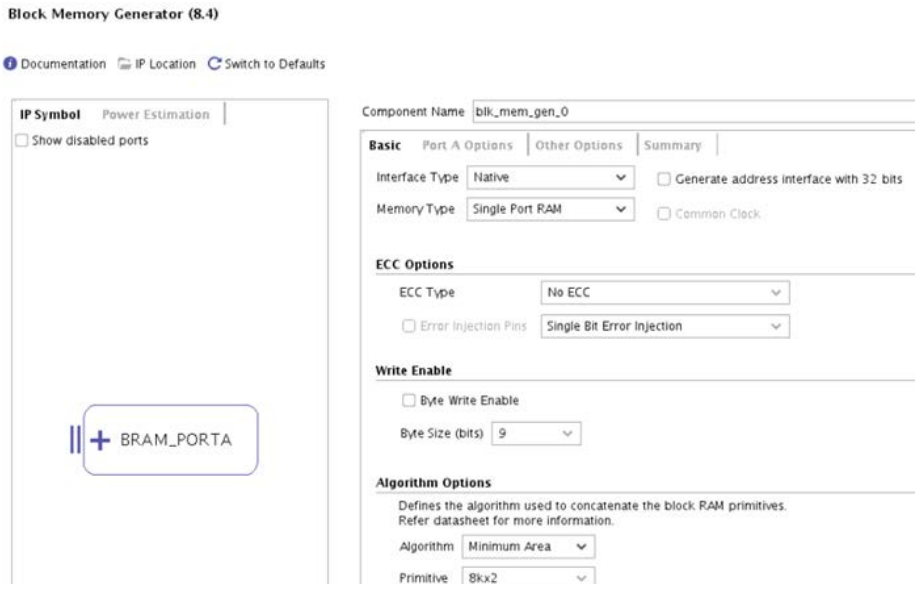


图 5-3 vivado memory 生成示列图

在进行 memory 的替换时，要确保生成的 FPGA 的 memory 和 ASIC 的 memory 行为逻辑要一致，体现在端口一致，时序一致上面。

(2)特定 IP 的处理

本文设计中用到了 DDR 模块，它是带有 PHY 的高速接口。由于 DDR IP 中的 PHY 为模逆 IP，所以在 FPGA 验证时不能验证，需要将其替换为 FPGA 中的 DDR IP，开发板上装载的 DDR 为 DDR4，调用 DDR4 的 controller + PHY IP，部分参数配置如图 5-4 所示：

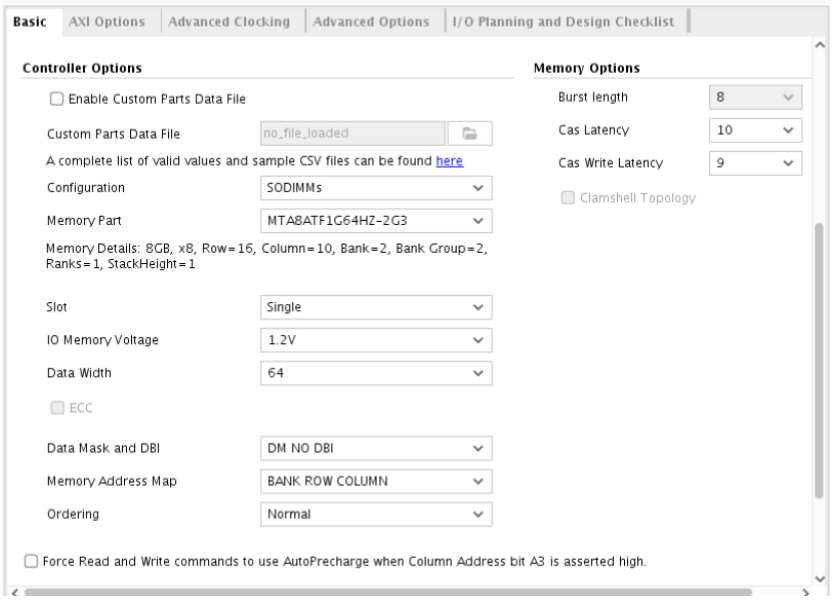


图 5-4 DDR IP 生成参数图

5.2.2 设计约束

我们为了在综合，映射，布局布线过程中进行逻辑优化，提高系统工作频率，减小布线延时便需要编写约束文件。并利用 vivado 的静态时序分析工具生成的时序报告来对设计的时序进行分析并判断时序是否满足要求。本设计中的部分约束文件设计如图 5-5 所示：

```
create_clock -period 500.000 -name uart_clk  -waveform {0.000 250.000} [get_pins SOC_TOP/CLK_TOP/nco_uart_reg[15]/Q]
create_generated_clock -name ui_clk -source [get_pins SOC_TOP/ddr_axi4/inst/u_ddr4_infrastructure/gen_mmcme3.u_mmcme_
adv_inst/CLKOUT0] -divide_by 1 [get_pins SOC_TOP/ddr_axi4/c0_ddr4_ui_clk]

set_false_path   -from [get_clocks JTAG_CLK] -to   [get_clocks ui_clk  ]
set_false_path   -to   [get_clocks JTAG_CLK] -from [get_clocks ui_clk  ]
set_false_path   -from [get_clocks JTAG_CLK] -to   [get_clocks uart_clk]
set_false_path   -to   [get_clocks JTAG_CLK] -from [get_clocks uart_clk]
set_false_path   -from [get_clocks JTAG_CLK] -to   [get_clocks XIN_25m ]
set_false_path   -from [get_clocks JTAG_CLK] -to   [get_clocks XIN_25m ]
```

图 5-5 约束文件设计图

5.2.3 综合和布局布线

在 FPGA 原型验证中综合就是把 HDL 语言从寄存器描述级(RTL 级)转换为门级网表的过程。本设计直接采用 vivado 工具对项目代码进行综合，本项目的部分综合结果如图 5-6 所示：

Site Type	Used	Fixed	Available	Util%
CLB LUTs*	205559	0	2532960	8.12
LUT as Logic	204525	0	2532960	8.07
LUT as Memory	1034	0	459360	0.23
LUT as Distributed RAM	960	0		
LUT as Shift Register	74	0		
CLB Registers	113057	0	5065920	2.23
Register as Flip Flop	113057	0	5065920	2.23
Register as Latch	0	0	5065920	0.00
CARRY8	1948	0	316620	0.62
F7 Muxes	4379	0	1266480	0.35
F8 Muxes	937	0	633240	0.15
F9 Muxes	0	0	316620	0.00

图 5-6 部分综合结果图

5.2.4 时序分析

在 FPGA 验证中，时序分析是十分重要的一步，它依据电路网表的拓扑结构，计算并检查电路中每个触发器的建立时间和延迟时间以及其他基于路径的延时是否满足要求，比如在工作时钟信号的锁存沿是否可以正确寄存我们需要的数据，以此来提高系统的工作主频和提高系统的工作的稳定性。我们可以通过添加约束

的方式来使得设计达到特定应用的设计要求。本项目在设计时利用 Vivado 工具的 Report Timing Summary 进行时序分析。通过查看时序分析报告，来确定不满足时序要求的路径，然后通过修改 XDC 约束或者设计代码来使得设计满足时序要求。最终需要保证电路的时序都满足时序要求。本项目中的时序分析结果如图 5-7 所示：

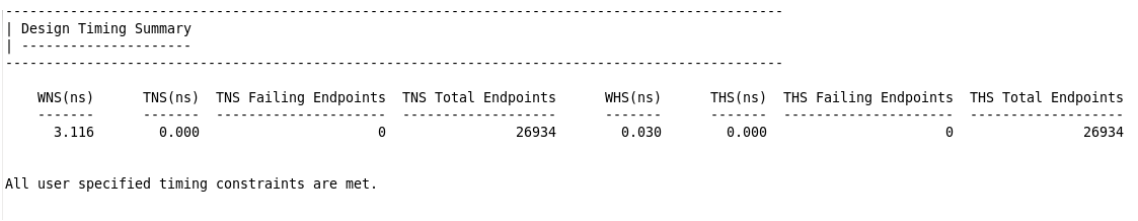


图 5-7 时序分析结果图

5.2.5 上板验证

在确保电路的时序满足要求后，需将生成的比特流下载到 FPGA 芯片中，并且在 FPGA 开发板上结合软件调试代码来验证设计的正确性。本项目利用 Vivado 工具自动生成比特流后，利用 Hardware Manager 选项将比特流烧录到 FPGA 开发板中，然后结合同事写的 ECDH，HMAC-SHA256，AES-CTR 算法模块的驱动程序进入调试，然后输入算法标准规定的测试激励来验证各算法模块的设计的正确性。将各算法模块的运算结果通过 UART 串口打印出来，然后和算法标准中的测试结果进行比对验证了各个算法模块的正确性。串口打印出的各算法模块运算结果如图 5-8 所示：

图 5-8FPGA 验证串口打印结果图

5.3 本章小结

本章首先介绍了 FPGA 的开发板的选取型号为 Xilinx VirtexUltraScale 系列的 XCVU440-FLGA2892。然后对 FPGA 验证的各个流程分析和介绍，最终按照整个流程完成了 FPGA 上板验证并确认了 ECDH，HMAC-SHA256，AES-CTR 这 3 个

算法模块的正确性。

第六章 总结与展望

6.1 论文总结

目前的信息安全问题已经变得越来越发的重要，针对目前视频图像数据的安全认证需求，本文根据国际主流的密码算法设计了一款集秘钥协商，数据认证，数据加密功能的数据认证加密 SoC 芯片，该芯片能够实现多种设备的安全通信与认证。本文所做的主要工作如下：

(1)首先研究了数据认证加密 SoC 的研究意义与背景，接下来对数据认证加密 SoC 的国内外研究现状做出介绍，然后提出了本文的主要研究目标和工作内容。

(2)接下来开始秘钥协商算法进行学习研究，介绍了秘钥协商算法 ECDH 算法的相关知识及其涉及到的相关运算法则，并出于对安全性能的考虑，确定选取 Curve25519 曲线为 ECDH 算法的椭圆曲线，然后完成了 ECDH 算法的模块设计并对其完成模块级的功能仿真。通过此算法可以完成共享秘钥的创建。

(3)对消息完整性认证算法 HMAC-SHA256 的原理进行了介绍，然后完成了此算法的代码设计和仿真，可以利用此算法生成消息认证码，然后完成身份的认证。完成了 AES-CTR 算法的加密电路的设计和仿真实现。

(4)最后根据功能需求分析，选取了所需要的 IP 资源，并结合设计的算法模块，完成数据认证加密 SoC 的总体结构设计，并为了使得各个模块能够协调合理的工作，完成了总线矩阵的设计，最后集成一个完成的 SoC 系统。

(5)将 ASIC 设计代码转换为 FPGA 代码，然后编写约束并进行综合，布局布线和时序分析，最终生成比特流，并将比特流下载到搭建的 FPGA 验证平台，然后结合软件调试代码对 3 个密码算法模块进行板级功能验证。

6.2 后续展望

本文设计的认证加密 SoC 虽然基本功能测试正确，但还达不到商用技术水平，本文的研究工作还不够完善，可以从下方面继续深入研究和改进：

(1)本文中只实现了 ECDH 算法，HAMC-SHA256 算法，AES-CTR 算法的基础功能，而将这些算法模块加上 APB 总线接口则是由其它同事完成，后续需继续研究如何将算法模块挂载到总线上。

(2)本文在 ECDH 算法的实现过程中的模乘采用的移位模乘算法，可以采取性能更好的蒙哥马利模乘算法来提高模乘模块的性能，以此来提高 ECDH 算法的性

能。且目前设计的 ECDH 仅用于简单双方秘钥协商，可以加入 ECDSA 算法来进行数字签名，提升整个认证加密 SoC 的安全性能。实现的 HMAC-SHA256 算法只满足输入为 512 比特的场景，可以利用参数化设计来增加算法设计的灵活性，以满足不同场景的需求。实现的 AES-CTR 算法的秘钥长度为 128 比特，后续可以将其秘钥长度扩展为兼容 128 比特，192 比特和 256 比特，提升算法的灵活性。

(3)在未来的设计中，可以考虑增加多个加密算法，和增加多种接口，以满足不同特定场景的应用。

参考文献

[1] 陈媛媛,陈菁. 密码认证技术研究现状分析[J]. 信息与电脑,2022,34(15):238-240.

[2] 于文奇. 大数据环境下密码安全技术研究[J]. 信息记录材料,2018,19(10):52-53.

[3] 沈昌祥,张焕国,冯登国,等. 信息安全综述[J]. 中国科学 E 辑,2007,37(2):129-150.

[4] Lin H Y ,Hsieh M Y . A dynamic key management and secure data transfer based on m-tree structure with multi-level security framework for internet of vehicles[J]. Connection Science, 2022, 34(1):1089-1118.

[5] 庄敏. 对称密码算法的实现与验证[D]. 广东:广东工业大学,2021.

[6] 贾宁. 密码算法的研究综述[J]. 现代电子技术, 2007, 30(11):3.

[7] 胡燏. 浅析对称密码算法与非对称密码算法的异同[J]. 计算机光盘软件与应用,2014(3):178-179.

[8] 曲思源, 戴紫彬, 李伟,等. SHA-2 算法在多核密码处理器上的实现研究[J]. 计算机应用与软件, 2016, 33(4):5.

[9] ZOU, JIAN, LIJI,et al. New quantum circuit implementations of SM4 and SM3[J]. Quantum information processing,2022,21(5).

[10] NANNIPIERI, PIETRO, MATTEO,et al. VLSI Design of Advanced-Features AES Cryptoprocessor in the Framework of the European Processor Initiative[J]. 2022,30(2):177-186.

[11] JAYSHREE, SEETHARAMAN, GOPALAKRISHNAN, et al. Enhanced TACIT Encryption and Decryption Algorithm for Secured Data Routing in 3-D Network-on-Chip based Interconnection of SoC for IoT Application[J]. 2021,80(6):520-527.

[12] 黎霖. 基于 ARMv6 架构的安全加密芯片研究与实现[D]. 广西:广西师范大学,2020.

[13] 杨家昌. 基于国密算法安全芯片的设计、实现与验证[D]. 广东:广东工业大学,2019.

[14] 胡景秀,杨阳,熊璐,等. 国密算法分析与软件性能研究[J]. 信息网络安全,2021(10):8-16.

[15] 储奕锋. AES 算法的 FPGA 实现[J]. 电脑知识与技术（学术交流）,2007,4(19):191-193.

[16] 李峰,于治楼,姜凯. 加密算法在 SOC 中的设计实现[J]. 信息技术与信息化,2009(4):27-29.

[17] Xiez ， Han j ， Yang j , et al. A low-cost SoC implementation of AES algorithm for bio-signals[C]. 2015 IEEE 11th International Conference on ASIC (ASICON). IEEE, 2015.

[18] KoppermannP ,Santis F D ， Heyszl J , et al. Low-latency X25519 hardware implementation: breaking the 100 microseconds barrier[J]. Microprocessors and Microsystems, 2017, 52(10):491-497.

[19] Choi J B， Kim D S， Choe J Y , et al. Hardware Implementation of ECIES Protocol on Security SoC[C]. 2020 International Conference on Electronics, Information, and Communication (ICEIC). 2020.

[20] S. Islam, M. L. Ali and R. Ruhana. Design of a Low Power Double AES Crypto-Processor SoC[C]. *2021 IEEE International Conference on Telecommunications and Photonics (ICTP)*, Dhaka, Bangladesh, 2021.

[21] 明洋,曲英杰. 基于 RISC-V 和密码协处理器的 SOC 设计[J]. 电子设计工程,2022,30(24):70-74.

[22] 高磊. ARM 系列芯片开发的前景[J]. 电子制作,2013(10):237-237.

[23] 乔霖,李永红,岳凤英. 基于 EIM 总线传输的数据通信接口设计实现[J]. 现代电子技术,2019,42(24):92-95,99.

[24] 杨金鑫. 基于 ARM 多核处理器的 SoC 系统设计与验证[D]. 陕西:西安电子科技大学,2020.

[25] 许云龙. 基于 APB 总线的 SPI 接口的设计与实现[J]. 电子质量,2020(7):128-132.

[26] 徐建皓, 万培元, 刘胜,等. 一种基于 APB 总线的单线接口设计[J]. 太赫兹科学与电子信息学报, 2022, 20(10):7.

[27] 李胜金, 张昌宏, 周大伟. 一种基于 ECDH 的可认证密钥协商协议[J]. 信息安全与通信保密, 2011, 9(7):3.

[28] NARESH, VANKAMAMIDI S., MURTHY, NISTALA V. E. S.. Provably secure group key agreement protocol based on ECDH with integrated signature[J]. Security and Communication Networks,2016,9(10):1085-1102.

[29] SALARIFARD, RAZIYEH, BAYAT-SARMADI, SIAVASH. An Efficient Low-Latency Point-Multiplication Over Curve25519[J]. IEEE transactions on circuits and systems, I. Regular papers: a publication of the IEEE Circuits and Systems Society,2019,66(10):3854-3862.

[30] 刘晓陆. 一种改进的 AES 算法及其性能分析[J]. 长江信息通信,2021,34(11):27-29.

[31] NHAT-PHUONG TRAN, MYUNGHO LEE, SUGWON HONG, et al. High Throughput Parallelization of AES-CTR Algorithm[J]. IEICE transactions on information and systems,2013,E96:1685-1695.

[32] 岳文文,魏胜非,周凯,等. 基于混沌系统和 HMAC 算法的图像加密研究[J]. 计算机仿真,2022,39(12):294-299,372.

[33] 李丹峰. 基于片上网络的 SHA256 算法优化[D]. 中国科学院大学,2019.

[34] 飞思卡尔在未来 i．MX 产品中许可使用 ARMCortex—A7 和 Cortex—A15 处理器[J]. 单片机与嵌入式系统应用,2011,11(12):83-83.

[35] MENGNI BIE, WEI LI, TAO CHEN, et al. An energy-efficient reconfigurable asymmetric modular cryptographic operation unit for RSA and ECC[J]. 信息与电子工程前沿（英文版）,2022,23(1):134-144.

[36] 刘帅. 椭圆曲线密码算法的硬件加速研究[D]. 山东:山东大学,2015.

[37] KUMARIK, ANITHA, SADASIVAM, et al. An Efficient 3D Elliptic Curve Diffie-Hellman (ECDH) Based Two-Server Password-Only Authenticated Key Exchange Protocol with Provable Security[J]. IETE journal of research,2016,62(6):762-773.

[38] [1]DJ Bernstein. Curve25519: New Diffie-Hellman Speed Records[C]// International Workshop on Public Key Cryptography. Springer, Berlin, Heidelberg, 2015.

[39] 成娟娟,郑昉昱,林璟锵,等. Curve25519 椭圆曲线算法 GPU 高速实现[J]. 信息网络安全,2017(9):122-127.

[40] 范云海. 椭圆曲线密码 ECC 二进制域的算法改进与硬件实现[D]. 上海:上海交通大学,2012.

[41] 张强,曲英杰. GF(2m)域椭圆曲线有限域的 VLSI 实现方法研究[J]. 信息技术, 2017, 41(12):6.

[42] HARRIS E. MICHAIL, GEORGE S. ATHANASIOU, VASILIS KELEFOURAS, et al. On the Exploitation of a High-Throughput SHA-256 FPGA Design for HMAC[J]. ACM transactions on reconfigurable technology and systems,2012,5(1):2.1-2.28.

[43] 王德鹏. HMAC-SHA256 算法的 VLSI 结构设计[D]. 黑龙江:哈尔滨工业大学,2015.

[44] 刘思辰. 安全散列算法的专用集成电路设计与性能分析[D]. 陕西:西安电子科技大学,2019.

[45] 须磊. HMAC-SHA256 算法的优化设计[J]. 价值工程, 2012, 31(29):3.

[46] 李炽阳,雷倩倩,杨延飞. 全通用 AES 加密算法的 FPGA 实现[J]. 计算机工程与应用,2020,56(10):83-87.

[47] 甘志超. AES 加密算法 FPGA 实现[J]. 现代信息科技,2022,6(10):29-31.

[48] 杨东轩, 张刚刚, 刘新亮. 基于 Cortex-M4 内核的 AES-128-CTR 算法汇编优化[J]. 华东师范大学学报:自然科学版, 2022(4):12.

[49] 费雄伟, 李肯立, 阳王东. 基于 CTR 模式的 GPU 并行 AES 算法的研究与实现[J]. 小型微型计算机系统, 2015(003):036.

[50] 罗春梅. 基于 FPGA 的 AES 加/解密系统的优化设计与实现[D].西华师范大学,2021.