论文题目：LTE 终端加解密硬件加速器的研究与设计

专　　业：信号与信息处理

硕 士 生：王　振　　(签名)_____

指导教师：吴延海　　(签名)_____

摘　要

随着无线移动通信技术的快速发展，无线移动通信已经逐步成为人们交流和联系的主要手段。然而由于无线传输介质的开放性使其存在很大的安全隐患。作为一种极具竞争力的无线接入技术，LTE 标准制定开始就将安全问题提到一定高度。然而无线移动终端存在运算能力、存储空间和电池容量有限等问题，一些复杂的数据运算需要通过专用的硬件来加速实现。

本文基于 AHB 总线，研究和设计了一款用于 LTE 终端 PDCP 层中的加解密硬件加速器。论文仔细研究了 LTE 标准中的安全性架构协议，对加解密硬件加速器进行了结构设计和模块划分。实现了 CMAC 模式、CTR 模式、UIA2 算法、UEA2 算法以及 AES 核和 SNOW 3G 核的硬件设计。在 AES 核的设计过程中，为了减小硬件面积，利用复合域降阶的方法，用组合逻辑电路实现了字节代换操作；采用矩阵因子分解的方法实现了列混合操作。在 SNOW 3G 核的设计中，S-BOX S_1 采用 AES S-BOX 的实现方式，节省了硬件资源开销。在设计加解密 IP 核时，加密与完整性保护模块并行处理，提高了数据处理速度的同时节约了资源。在密钥管理方案中，采用硬件提前对初始密钥进行密钥扩展运算并且存储的方式，在密钥不变的情况下，不用重新对初始密钥进行密钥扩展运算，降低了功耗。

本文使用 Verilog 硬件描述语言，对加解密硬件加速器进行了 RTL 描述。利用 EDA 工具对系统进行了逻辑功能仿真和静态时序分析。在基于 Virtex-6 的 FPGA 硬件平台上对系统进行了验证，验证结果表明：本设计在满足各项功能及设计目标的基础上实现了速度和资源的平衡，具有较好的速度面积比。

关 键 词： AHB 总线；高级加密标准；SNOW 3G；CTR；CMAC；UIA2；UEA2

研究类型： 应用研究

Subject ： Research and Design of Encryption and Decryption Hardware Accelerator in LTE Terminal

Specialty ： Signal and Information Processing

Name ： Wang Zhen （Signature）_____

Instructor： Wu YanHai （Signature）_____

ABSTRACT

With the rapid development of wireless communication technology, wireless communication has gradually been the main expression and communication in human life. however, a big security risk exists because of the openness of wireless networks. As a very competitive wireless access technology, standard of LTE has taken into account security issues at the very beginning. Whereas, there are limitations for the wireless mobile terminal, such as calculating ability, storage space, battery capacity, and so on. As a result, some complex data operations need to be realized by special hardware accelerator.

Based on the AHB bus, a encryption and decryption hardware accelerator used in PDCP layer of LTE communication terminal has been researched and designed in this thesis. Firstly, the security layer protocol in LTE standard was researched in detail to complete the structural design and module division of the hardware accelerator. Secondly, complete the CCM mode, CTR mode, UIA algorithm, UEA algorithm as well as the hardware design of AES and SNOW 3G. During designing the AES core, in order to reduce the hardware area, a special circuit structure is proposed to realize the ShiftRow operations. According to the composite field reduction method, combinational logic circuits are utilized to implement the SubByte operations. In the design of SNOW 3G, AES S-BOX was used by S-BOX S1 to save the sources while encryption and integrity protection module were parallel processed to improve the data processing speed and save resources in the design of encryption and decryption IP. In the key management scheme, hardware was used preliminarily in the extended algorithm and saving of the initial key, what's more, when the key is unchanged, no extended algorithm should be performed in the initial key, for which would reduce the power.

The hardware accelerator has been described in RTL level using Verilog hardware description language, and EDA was used to analyze the logic function simulation and static

timing. Then the whole system was tested by FPGA hardware platform based on Virtex-6. The results showed that the speed and resource in the design were balanced on the basis of achieving the whole functions and the design objective while the design was of high speed area ratio.

Key words: AHB Bus Advanced Encryption Standard SNOW 3G CTR CMAC UIA2 UEA2

Thesis : Applied Research

目 录

1 绪论

1.1 课题研究背景

随着无线移动通信的不断发展和无线网络速度的加快，更多的用户越来越愿意接触到互联网，网上银行、网络会议等将更加便捷。无线移动通信系统作为一种便捷、有效的通信技术，已逐步成为人们交流和联系的主要的手段，但其传递信息的安全性和保密性也更加广泛的被关注。越来越多的用户将自己的个人信息放到手机上，而手机又可以随时随地的接入到互联网中。这样许多黑客就可以通过对用户的手机植入木马程序来窃取用户的信息和个人隐私。而更为可怕的是，国外已经出现了通过木马自动打开手机通话功能的程序，通过自动通话功能来窃取重要会议的内容或监听用户的通话。这对用户而言，可以说是无密可保、无隐私可言。由于无线网络传输介质的开放性，人们在享受无线网络所带来的方便、灵活、经济的同时，更容易受到主动干扰攻击。因此在移动通信过程中，确保传播数据的安全性、可靠性及完整性至关重要。

LTE（Long Term Evolution）是由3GPP（3rd Generation Partnership Project,第三代合作伙伴计划）组织提出的下一代通讯标准，是GSM超越3G和HSPA阶段迈向4G的进阶版本，是移动通信技术发展的一个重要方向。其技术研究、系统标准化以及产业发展规划，受到了人们的高度关注。LTE技术研究从2004年底正式启动，主要设计目标是顺应无线通信宽带化的发展趋势，实现远远高于目前3G系统的频谱效率和通信带宽。同时使网络结构更简单，运营成本更低，以及对各种不同的业务需求提供更加灵活的支持。然而3GPP工作组在LTE标准制定开始就将LTE系统中的安全问题提到一定高度，在LTE标准制定同时LTE的安全性架构研究也同时开展。LTE的安全性是在PDCP（分组数据汇聚协议）层负责的，通过对信令（如RRC消息）采用加密和完整性保护，以及对用户数据进行加密来实现[1]。加密保护可以使发送方的消息内容不能被除接收方之外的第三方所获知。加密会改变发生消息的内容，使截获该条消息的一方在不知道密钥等参数的情况下无法获知该消息表达的真正内容。而完整性保护可以确保接收端收到的消息内容没有被第三方篡改，即与发送端保持一致性。

LTE中数据的安全性都是基于算法来实现的，目前LTE系统中数据的机密性和完整性保护的核心算法分别是高级加密标准（Advanced Encryption Standard）AES算法[2]和SNOW3G算法[3]。美国国家标准与技术协会（NIST）向全世界征集一种开放式加密算法，最终在15种候选的加密算法中确定Rijndael为AES的算法，于2011年在联邦信息处理标准刊物上公开发表新的数据安全加密标准。Rijndeal算法是对称的数据块加密算法，由比利时计算机学家Vincent Rijmen和Joan Deamen共同开发，该算法支持128、192和256位三种

长度密钥。由于AES是DES的继承，因此工业界、银行等多个行业将此作为自己的密码标准。一些标准协会甚至早于美国国家标准与技术协会已经将AES作为各自标准的一部分。如因特网工程组就将它纳入TLS、IPSec等多个协议体系中。SNOW 3G是一种面向字的流加密算法，迄今为止已经先后发布了SNOW2.0和SNOW 3G等一系列版本，其中SNOW 3G是上一版本的升级。它能够有效保证在网络中传输的用户数据和信令数据不被被窃听和修改。

要完成PDCP层数据加密和完整性保护工作，需要将AES算法和SNOW 3G算法分别与各自的模式和算法相结合。LTE安全性架构协议[4]中用到了四种模式，分别是CTR模式[5]、CMAC模式[6]、UEA2算法[7]和UIA2算法[7]。其中CTR模式和CMAC模式是以AES算法为基本运算单元对数据进行处理；UEA2算法和UIA2算法是以SNOW 3G算法为基本运算单元对数据进行处理。其中CTR模式和UEA2算法完成对LTE系统中PDCP层的信令数据和用户数据的加密与解密工作；CMAC模式和UIA2算法完成对LTE系统中PDCP层的信令数据的完整性保护和完整性验证工作。

由于无线移动终端的处理器能力和电池容量是非常有限的，所以利用软件实施PDCP层的安全性功能已经成为实时安全通信的重要瓶颈。商品化CPU无法跟上数据加密算法的速度要求，尤其在CPU完成了通信系统的相关任务之后留给安全算法的运算资源非常有限。然而，基于FPGA高度优化的可编程硬件可以达到所要求的安全算法处理性能基准。而且硬件实现可以很大程度提高运算速度以及更高的安全性。因此，LTE终端中的安全算法需要通过专用的硬件电路来实现。

1.2 国内外研究现状

在终端芯片方面，2008年，LTE开始加快了标准制定和产业化工作，并得到运营商和设备商的广泛支持，2009年LTE经历了高速发展的一年。LTE成为移动通信主流技术的地位已经没有悬念，不仅原有的GSM/CDMA厂家在全力的发展LTE，WiMAX厂家也被逐渐吸入LTE阵营，目前的主流设备商和芯片厂家都已全力进入到LTE产业中，LTE终端产业链日益壮大。

目前国内外多家企业正在抓紧研发LTE终端芯片，参与研发的企业主要有大唐、中兴、华为、高通、三星、博通等知名企业。预计2013年将有多款新产品商用。根据全球移动设备供应商协会（GSA）2011年的统计，全球已经研发出98款LTE终端，但是终端以USB Modem和路由器为主，手机和其他智能终端仍然匮乏。

2010年，LTE终端芯片工艺在65nm水平。2011年40nm-45nm工艺将逐渐普及，其主要是采用1G-1.2G的双核MCU，可以支持智能手机等终端，为LTE语音终端做好了上市准备。2012年，28nm的芯片工艺逐渐普及，主要采用1.5G的双核MCU，可以支持高端多模的智能手机和低价智能手机。2013年后，28nm及更高的芯片制造工艺将被规模使用，

将为LTE终端低价位打下基础[8]。

在LTE系统安全算法研究方面，国内研究起步比较晚，和国外差距比较大。但是经过这些年的发展，国内和国外同期在AES技术和SNOW 3G技术上的差距迅速减小。算法的实现一般为软件实现方式和硬件实现方式，软件实现起来相对比较容易，所以对于AES算法和SNOW 3G算法软件实现的研究现在比较多，国内的发展也比较成熟。市场上也出现了许多优秀的AES算法和SNOW 3G算法的软件加密产品。但是在国内外SNOW 3G算法的硬件实现基本还停留在理论上，在硬件实现方面提供的资料很少，虽然有全定制产品及IP核的出现，但是在通用性、功耗及速度方面还有很大差距。然而AES算法的硬件实现相对比较成熟，早期AES的硬件实现主要是基于FPGA，因为FPGA丰富的RAM资源实现了AES的高吞吐率。随着大规模集成电路的发展，AES算法的硬件实现主要向着低功耗、高性能、可配置性和在性能和功耗之间找一个平衡如文献[9]这四个方向发展。在低功耗方面，如文献[10]采用窄的数据总线等各种设计方法使设计面积尽可能减小；在高性能方面，如文献[11]优化流水线结构，采用多级流水线设计方法使吞吐率到几十个Gbps；在可配置性方面，如文献[12]实现分组大小和密钥长度可配置性。但是对于LTE终端PDCP层安全性和完整性功能的硬件研究国内外目前正在处于研发期，相应的设计方案及资料都非常少。文献[13]提出了一种实现LTE终端PDCP层加密功能的硬件设计方案，并对此方案的可行性进行了仿真验证。此方案在系统时钟为200MHz的情况下，吞吐率可以达到100Mbps。但是描述比较粗糙，且设计中没有包含完整性保护功能的设计。

1.3 本文主要研究内容

在终端产品设计中，是在满足用户需求的前提下再追求成本最低化为基本要求的。因此对于一些只考虑高性能或者低功耗的设计可能满足不了用户要求。所以本文将在性能和功耗之间找一个合适的平衡点，设计一款满足用户要求且又不过度耗费硬件资源的面向ASIC的加解密硬件加速器是本文研究的主要内容。本文将在基于AMBA总线的SOC系统架构的基础上，基于ARM公司的AHB总线对LTE终端PDCP层的加解密硬件加速器（下文中简称硬件加速器）进行研究与设计。本文的主要工作如下：

1、研究与分析 LTE 安全性架构协议和 ARM 公司的 AMBA 总线架构，提出并且确立一种有效的基于 AHB 总线的加解密硬件加速器和 ARM 处理器的数据交互流程。

2、基于 AMBA AHB 总线，对硬件加速器进行结构设计和模块划分，实现硬件加速器及各模块的硬件电路。

3、采用 Verilog 硬件描述语言，对硬件加速器及各模块进行 RTL 级描述。

4、建立仿真环境，使用 modesim 工具对各个模块及系统进行逻辑功能验证，使用 ISE 工具对设计电路综合布局布线后进行静态时序分析。

5、在 FPGA 搭建的验证平台上，模拟 LTE 终端通信环境，对设计进行全面验证，

最后对结果进行分析。

1.4 本文结构

本文结构如下安排：

第1章：绪论，介绍本课题的研究背景、国内外的研究现状以及本文的研究内容及意义。

第2章：加解密算法，阐述AES算法和SNOW 3G算法原理，同时介绍了与本文所用算法相关的一些数学知识。

第3章：硬件加速器设计，详细介绍了硬件加速器的设计，首先介绍了基于AMBA总线的LTE终端的SOC系统结构，其次基于AHB总线对硬件加速器进行了结构设计及模块划分，最后对硬件加速器中的各个模块进行详细设计。

第4章：加解密IP核设计，详细介绍了加解密IP核的设计，并对其进行模块划分以及对各模块进行详细设计。重点对AES核和SNOW 3G核进行了设计。

第5章：仿真与FPGA验证，对硬件加速器各模块及系统进行详细的功能仿真和静态时序分析，在FPGA硬件验证平台上对系统进行FPGA验证以及对实现结果进行分析。

第6章：总结与展望。

2 加解密算法

本章主要介绍加/解密和完整性保护/验证过程及系统中所用的加密模式、加密算法以及相关的数学知识，为后面对系统的研究设计做下铺垫。

2.1 加密和完整性保护

在PDCP层，LTE采用了加密和完整性保护。加密保护可以使发送方的消息内容不能被除接收方之外的第三方所获知。加密会改变发送消息的内容，使截获该条消息的一方在不知道密钥等参数的情况下无法获知该消息表达的真正内容。完整性保护可以确保接收端收到的消息内容没有被第三方篡改，即与发送端保持一致性[14]。

目前3GPP标准化的加密算法有空算法、SNOW 3G算法和AES算法；同样完整性算法也有空算法、SNOW 3G算法和AES算法。空算法可以用来在紧急呼叫时使用，即不进行安全保护。协议中规定，SNOW 3G算法必须结合UEA2和UIA2算法完成加密和完整性保护工作，AES算法必须结合CTR模式和CMAC模式完成加密和完整性保护工作。在加密和完整性保护过程中到底使用哪种算法，加密算法和完整性算法分别由LTE高层协议配置一个4bit标识符来进行选择加密和完整性保护算法。加密算法和完整性保护算法选择方式如表2.1所示。

表2.1 算法选择方式

标识符	算法
0000	空算法
0001	SNOW 3G算法
0010	AES算法

在LTE的PDCP层中，对于信令（如RRC消息）采用加密和完整性保护双重保护，发送端要对其进行完整性保护计算和加密计算，接收端要先对其进行解密然后再进行完整性验证。对于用户数据，出于等效等原因，只进行加密。

2.1.1 加密

加密过程如图2.1所示，发送端利用128bit加密密钥KEY、32bit的COUNT（计数器值）、5bit的BEARER（承载标识）、1bit的DIRECTION（上下行方向指示，0表示上行，1表示下行）和LENGTH（明文长度）作为加密算法输入参数，计算出密钥流与明文进行异或产生密文后发送。在本设计中，这些参数由软件通过描述符（参见3.3.7节）配置下来。

接收端收到密文后，利用与发送端相同的参数KEY、COUNT、BEARER、DIRECTION

和LENGTH以及和发送端相同的加密算法，将产生的密钥流与收到的密文进行异或操作恢复出明文。正常情况下，接收端计算出的密钥流和发送端用于加密的密钥流是一样的。

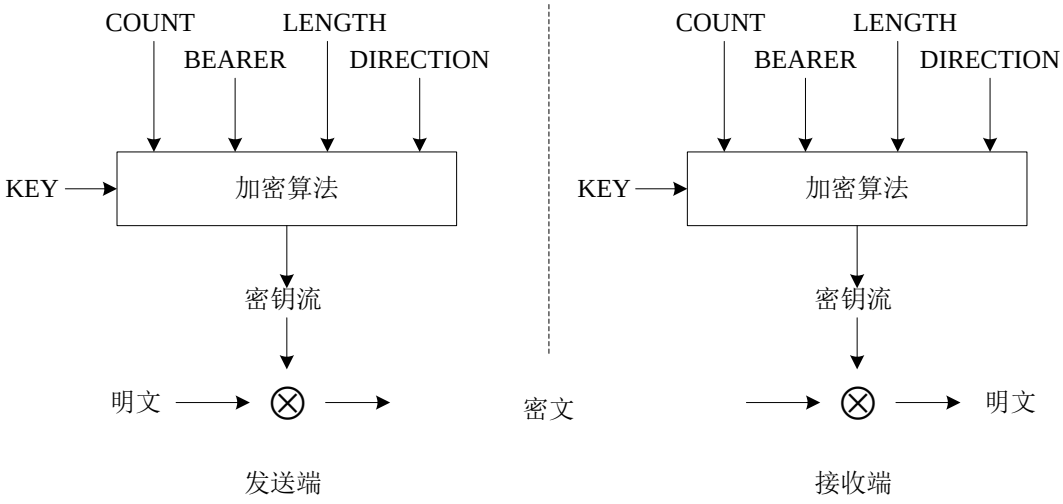

图 2.1 加密过程

2.1.2 完整性保护

完整性保护过程如图2.2所示。发送端利用完整性保护密钥KEY，以及COUNT、BEARER、DIRECTION和消息本身作为完整性保护算法的输入，生成一个32bit的完整性认证码，附在消息后面。发送端将消息本身和认证码一起发送给接收端；接收端利用和发送端一样的完整性保护密钥KEY，以及收到的消息本身和COUNT、BEARER、DIRECTION作为完整性保护算法的输入计算出一个32bit的完整性认证码，与收到的认证码进行比较。如果一致，则认为收到的消息和发送的消息是一致的，即没有被篡改。

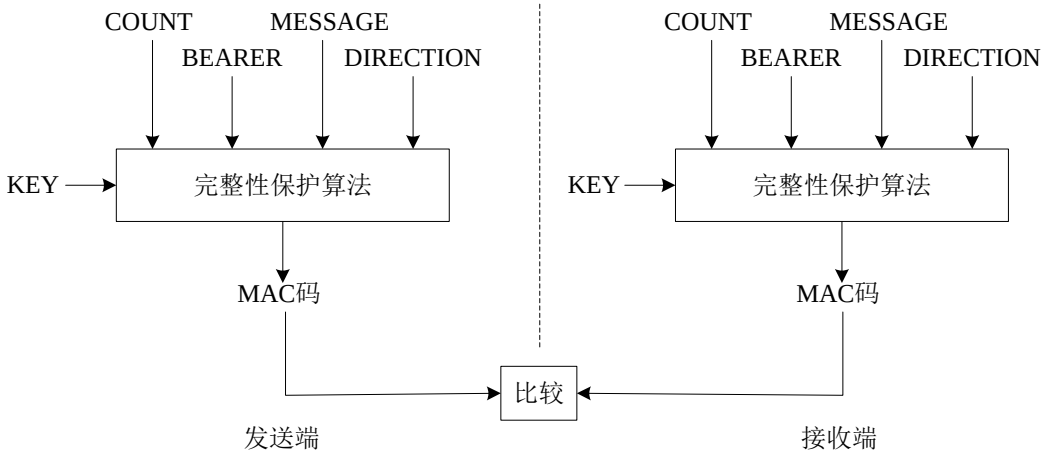

图 2.2 完整性保护过程

2.2 高级加密标准

2.2.1 有限域介绍

有限域是含有有限个元素的域，有限域中元素个数称为域的阶[15]。有限域的阶数等于素数的幂即p^n（p为素数），称p为有限域的特征，有限域记为$GF(p^n)$（n为正整数）。具有相同阶数的有限域是同构的，它们具有相似的代数结构。

在有限域$GF(p^n)$上定义一个多项式：

$$b(x)=b_{n-1}x^{n-1}+b_{n-2}x^{n-2}+\cdots+b_2x^2+b_1x^1+b_0 \tag{2.1}$$

称$b(x)$为有限域$GF(p^n)$上的多项式。其中x为有限域中多项式的形式变量，恒为“1”。

（1）字节运算

由于AES算法中任意一个字节都可看做为有限域$GF(2^8)$中的元素，所以对于一个字节$\{b_7,b_6,b_5,b_4,b_3,b_2,b_1,b_0\}_2$（$\{\}_2$表示括号内是二进制位序列，同理$\{\}_{16}$表示括号内是十六进制数序列）可以看作是一个系数属于$GF(2^1)$上的多项式，即：

$$b(x)=b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0 \tag{2.2}$$

因此字节间的加法和乘法运算就可以转化成有限域$GF(2^8)$中多项式间的加法和乘法运算。

有限域中两个多项式加法运算表示为两个多项式中相应系数的模 2 加，即异或运算（记为$\oplus$）。对于两个字节$\{a_7,a_6,a_5,a_4,a_3,a_2,a_1,a_0\}_2$和$\{b_7,b_6,b_5,b_4,b_3,b_2,b_1,b_0\}_2$，其和为$\{c_7,c_6,c_5,c_4,c_3,c_2,c_1,c_0\}_2$，其中 $c_i= a_i\oplus b_i$。

在多项式表示中，有限域$GF(2^8)$上的乘法(记为$\bullet$)定义为多项式的乘积模一个次数为8的不可约多项式：

$$m(x)=x^8+x^4+x^3+x+1 \tag{2.3}$$

使用模$m(x)$确保了多项式相乘后所得结果是次数小于8的二元多项式，结果多项式可以用一个字节表示。

假设 $a(x)$和 $b(x)$为有限域$GF(2^8)$上的两个多项式，$c(x)$为它们的乘积，则多项式 $a(x)$和 $b(x)$乘法运算定义为：

$$c(x)=a(x)\bullet b(x)\Leftrightarrow c(x)=(a(x)\times b(x))\bmod(m(x)) \tag{2.4}$$

用多项式 x 乘以等式(2.2)中定义的多项式将得到：

$$b(x)\bullet x=(b_7x^8+b_6x^7+b_5x^6+b_4x^5+b_3x^4+b_2x^3+b_1x^2+b_0x^1)\bmod(m(x)) \tag{2.5}$$

等式(2.5)的计算结果为：如果b_7=0，则结果为$\{b_6,b_5,b_4,b_3,b_2,b_1,b_0,0\}_2$，即把原字节左移一位，低位补零；如果$b_7$=1，则模运算需要异或多项式$m(x)$完成。乘$x$结果可通过把

原字节左移一位后和与$\{1b\}_{16}$进行按位异或来实现。将该操作记为xtime()。x的高次幂的乘法可以通过重复应用xtime() 然后将中间结果相加来实现。

（2）字节系数多项式运算

同(1)，定义一个四字节向量$[a_0,a_1,a_2,a_3]$，对于该字节向量可以看作是一个系数属于$GF(2^8)$上的多项式，即：

$$a(x)=a_3x^3+a_2x^2+a_1x^1+a_0 \tag{2.6}$$

注意该多项式与式（2.1）中定义的多项式操作起来是不同的，即使这两类多项式均使用相同的变量x。该式的系数本身就是有限域$GF(2^8)$中元素，即字节而不是比特。而式（2.1）的系数是有限域$GF(2^1)$中的元素。

为了说明（2.6）式的加法和乘法运算，令：

$$b(x)=b_3x^3+b_2x^2+b_1x^1+b_0 \tag{2.7}$$

为另一个4项多项式。加法运算是对x相应次数项的有限域系数字节进行按位异或运算，即：

$$a(x)+b(x)=(a_3\oplus b_3)x^3+(a_2\oplus b_2)\ x^2+(a_1\oplus b_1)x^1+(a_0\oplus b_0) \tag{2.8}$$

在字节系数多项式乘法运算中，要使得结果化简为一个次数小于4的多项式。在AES算法中，这一模多项式取x^4+1。设$c(x)$为$a(x)$和$b(x)$的代数积，$a(x)$和$b(x)$取模的乘积记为$a(x)\otimes b(x)$，则：

$$d(x)=d_3x^3+d_2x^2+d_1x^1+d_0=a(x)\otimes b(x)=c(x)\bmod(x^4+1) \tag{2.9}$$

当$a(x)$是一个固定多项式时，计算后等式（2.9）中定义的运算可以写成矩阵形式：

$$\begin{bmatrix}d_0\\d_1\\d_2\\d_3\end{bmatrix}=\begin{bmatrix}a_0&a_3&a_2&a_1\\a_1&a_0&a_3&a_2\\a_2&a_1&a_0&a_3\\a_3&a_2&a_1&a_0\end{bmatrix}\begin{bmatrix}b_0\\b_1\\b_2\\b_3\end{bmatrix} \tag{2.10}$$

本节只介绍了与AES算法有关的有限域知识，对于有限域的详细介绍见文献[15]。

2.2.2 AES 算法描述

AES算法是一个分组对称密码算法，其输入分组、输出分组和状态长度均为128bit，支持密钥长度为128、192和256位的比特序列。在LTE协议中规定使用128bit的密钥长度，所以本节只介绍密钥长度为128bit的情况。

AES算法处理的基本单元是由8bit二进制组成的字节，它的内部处理都是作用在一个名为State的二维字节数组上。如图2.3所示为输入、State数组和输出，其中in_0表示需要处理的128bit数据的高有效位字节，in_{15}表示需要处理的128bit数据的低有效位字节。State数组里的元素$S_{r,c}$为单字节，r代表行号，取值范围是$0\le r<4$；c代表列号，取值范围是

$0 \le c < Nb$，Nb表示State列数。

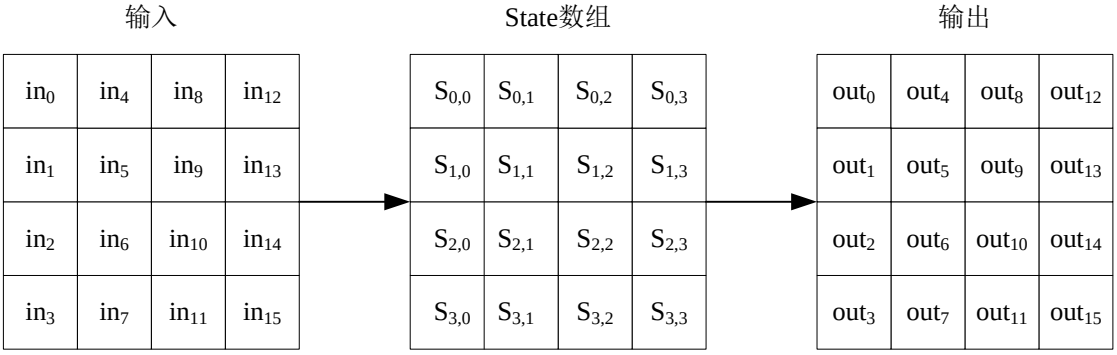

图 2.3 输入、State 数组和输出

（1）AES 算法流程

在本设计中，AES 算法通过执行10次轮函数来变换状态矩阵，其中轮函数在每次迭代过程中由4个不同的以字节为基本单位的变换复合而成。分别是字节替代、行移位、列混合和轮密钥加这几个操作。本节主要介绍LTE协议中所需的AES加密过程，如图2.4所示为AES加密流程。

加密开始时将输入复制到State矩阵中。经过初始轮密钥加后，通过执行10次轮函数来变换状态矩阵，最后一轮与前九轮不同，不用进行列混淆操作。其中每轮操作中所需的密钥由密钥扩展模块通过初始密钥获得，密钥扩展具体过程见本节⑤。接下来对轮函数中的四个操作进行描述。

① 行移位变换

在行移位变换中，只是对状态的后3行按照不同的偏移值r进行循环左移r个字节。第一行保持不变，第二行循环左移1字节，第三行循环左移2个字节，第四行循环左移3个字节。行移位变换具体过程如图2.5所示。

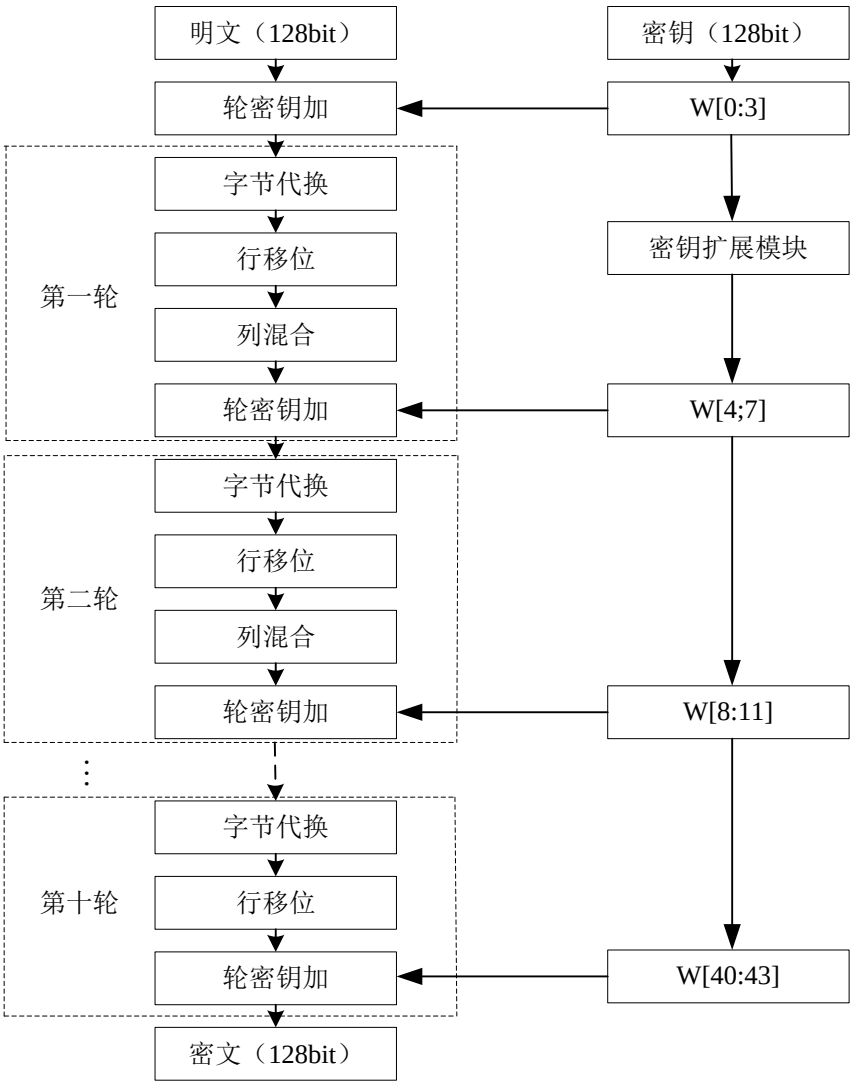

图 2.4 AES 加密流程

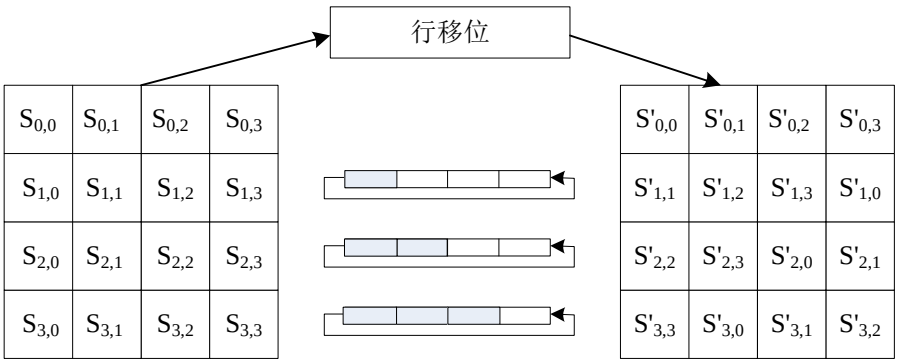

图 2.5 行移位变换

② 字节代换变换

字节代换变换是一个非线性的字节替代，它独立地将State中的每个字节利用替代表(S盒)进行运算，将State中的每个字节映射为一个新字节，字节代换变换过程如图2.6所示。S盒是可逆的，由两个变换复合而成：

1、对字节在有限域$GF(2^8)$上进行求乘法逆，其中元素$\{00\}_{16}$映射为它自身。

2、进行如下定义的仿射变换运算：

$$b_i' = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i \tag{2.11}$$

其中$0 \le i < 8$， c_i是值为$\{63\}_{16}$的第i比特。在文献[2]中以矩阵的形式表示仿射变换：

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{2.12}$$

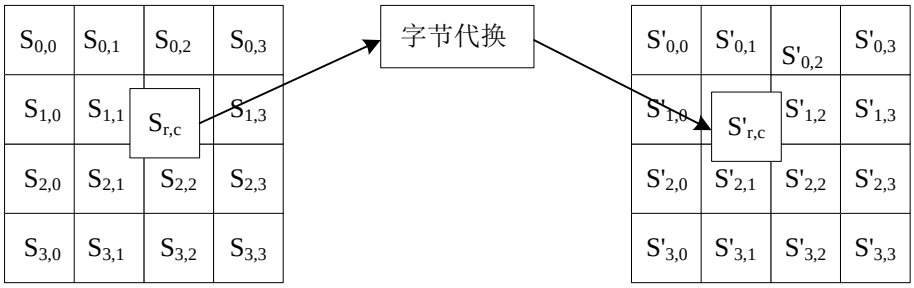

图 2.6 字节代换变换过程

③ 列混合变换

列混合变换在State上按照每一列进行运算，并将每一列看作是一个系数在有限域$GF(2^8)$中的多项式，并且与一个固定的多项式$a(x)$相乘后再对x^4+1取模。则：

$$a(x) = \{03\}_{16}x^3 + \{01\}_{16}x^2 + \{01\}_{16}x + \{02\}_{16} \tag{2.13}$$

由式（2.10）计算$s'(x) = a(x) \otimes s(x)$后，列混合变换后每列中的4个字节变换如式（2.14）所示，列混合变换过程如图2.7所示。

$$
\begin{aligned}
s'_{0,c} &= (\{02\}_{16} \bullet s_{0,c}) \oplus (\{03\}_{16} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\
s'_{1,c} &= s_{0,c} \oplus (\{02\}_{16} \bullet s_{1,c}) \oplus (\{03\}_{16} \bullet s_{2,c}) \oplus s_{3,c} \\
s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (\{02\}_{16} \bullet s_{2,c}) \oplus (\{03\}_{16} \bullet s_{3,c}) \\
s'_{3,c} &= (\{03\}_{16} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\}_{16} \bullet s_{3,c})
\end{aligned}
\tag{2.14}
$$

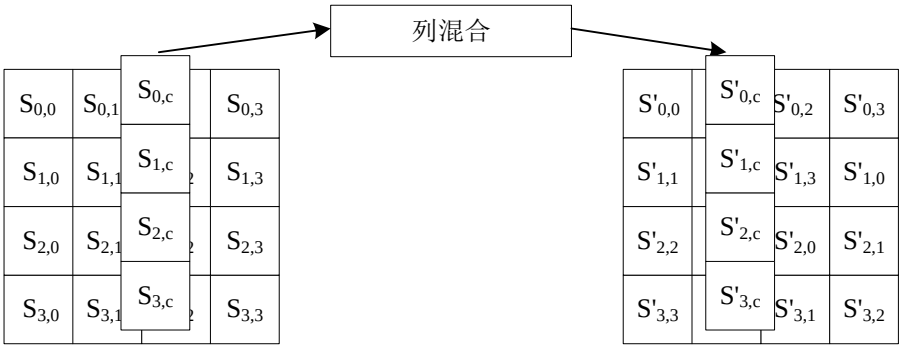

图 2.7 列混合变换过程

④ 轮密钥加变换

轮密钥加变换比较简单，但是却能影响State中的每一个字节。轮密钥加变换是将轮密钥按位与State进行异或运算，其轮密钥加变换过程如图2.8所示。

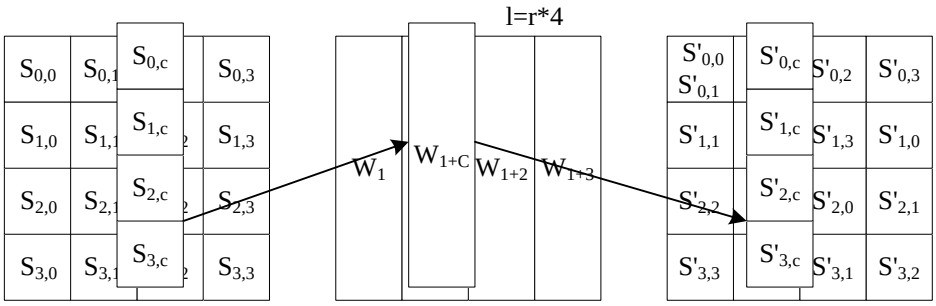

图 2.8 轮密钥加变换

⑤ 密钥扩展

密钥扩展就是根据初始密钥产生轮密钥加过程中所需的密钥，该算法需要一个*Nb*个字组成的初始密钥集合（本设计中*Nb*=4），*Nr*（AES迭代轮数，本设计中*Nr*=10）轮中的每一轮需要*Nb*个字的密钥数据。密钥编排结果由4个字节组成，记为w[i]，其中$0 \le i < Nb(Nr+1)$。图2.9为密钥扩展过程伪代码。

```
Key Expansion (byte key [4*Nk], word w [Nb*(Nr+1)], Nk)
{
    for   (i = 0; i < Nk; i++) {
           w[i] = word (key [4*i], key [4*i+1], key [4*i+2], key [4*i+3]);
       }
     for   (i = Nk; i < Nb*(Nr+1); i++) {
           if  (i mod Nk == 0)
                w[i] = w[i-Nk] ⊕ Subword (RotWord(w[i-1])) ⊕ Rcon[i/Nk];
           else  if  (Nk > 6 and i mod Nk == 4)
                w[i] = w[i-Nk] ⊕ Subword(w[i-1]);
            else
                 w[i] = w[i-Nk] ⊕ w[i-1];
        }
   }
```

图 2.9 密钥扩展过程伪代码

其中 SubWord（字节代换）函数功能为对输入的四个字节采用字节代换操作产生一个新字，函数 RotWord（字节左移）对输入的字$[a_0, a_1, a_2, a_3]$进行循环左移一个字节即$[a_1, a_2, a_3, a_0]$。Rcon[i]（轮常数）定义为$[RC[i], \{00\}_{16}, \{00\}_{16}, \{00\}_{16}]$，最右边 3 个字节一直为零，所以与 Rcon[$i$]按位异或结果只与最左边的字节相关。轮常数表如表 2.2（其中$1 \le i \le 10$，Rcon[i]是 16 进制数）所示。

表2.2 轮常数表

i	1	2	3	4	5	6	7	8	9	10
Rcon[i]	01	02	04	08	10	20	40	80	1B	36

密钥扩展过程开始将初始密钥复制到扩展密钥的前四个字中，接下来的每个字w[i]等于其前一个字w[i-1]与Nk（本设计中Nk=4）个位置之前的字w[i-Nk]进行异或操作。对于Nk的整数倍位置的字，在异或之前要对w[i-1]先进行一次字的字节循环移位，然后再做一次字节替代变换，最后再异或一个轮常数。

2.3 AES-CTR 模式

CTR（Counter，计数器）模式是以AES算法为基本单元的即AES-CTR，它的功能是对数据进行加密。通过一系列计数器$T_1, T_2, \ldots, T_n$ 作为输入，调用AES算法产生密钥流与明文数据进行异或操作产生密文。该计数器以某个常数为初始值，其长度等于明文块长度，T_i值为$T_i = T_{i-1} + 1$，$1 \le i \le n$。解密时使用和加密相同的计数器，将计数器产生的密钥流与密文做异或运算可得明文。AES-CTR模式加解密过程如图2.10所示。

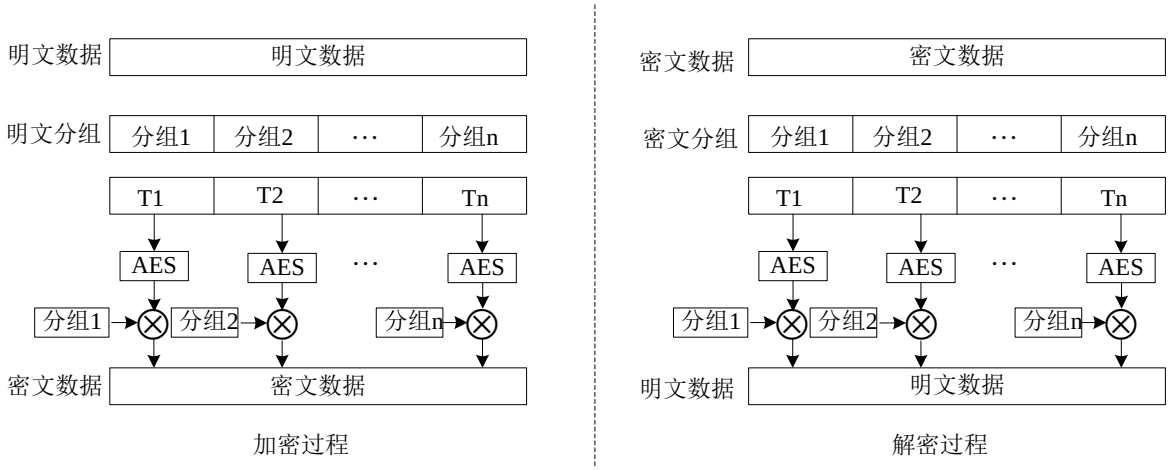

图 2.10 AES-CTR 模式加解密过程

由图2.10可知，加密时需先将明文数据进行分组，每组大小128bit，将分组后的明文分组分别与CTR模式产生的密钥流进行异或操作生成密文。解密过程和加密过程基本相同，解密时需要分组的数据是密文数据，将密文分组分别与CTR模式产生的密钥流进行异或操作产生明文。本设计中根据LTE协议规定计数器T_i的大小为128bit，初始值分别为：

T_1= COUNT|| BEARER|| DIRECTION||0^{26} ||0（||表示连接符）；

T_2= COUNT|| BEARER|| DIRECTION||0^{26} ||1；

⋮

T_n= COUNT|| BEARER|| DIRECTION||0^{26} ||n-1。

2.4 AES-CMAC 模式

CMAC（Cipher Block Chaining Message Authentication Code，加密分组链接消息认证）模式是一个基于对称密钥分组密码，由美国国家标准技术研究院制定。该模式的功能是根据输入的数据产生一个认证码（MAC）序列。由于在LTE协议中规定，在完整性保护计算中所用的CMAC模式是基于AES算法的，即AES-CMAC（下文中简称CMAC）。CMAC有着比校验及错误检测码更强健的完整性保护能力，而CMAC模式的安全性建立在AES算法强健的基础之上。CMAC模式产生的消息认证码能检测出数据在传输过程中被故意、非法及无意的修改，但是校验或者错误检测码只能检测出无意的修改。

在CMAC模式中需要用到两个子密钥K1和K2以及初始向量COUNT、BEARER和DIRECTION。在本设计中这两个子密钥由软件计算出来后通过描述符（参见3.3.7节）中配置下来，此处不再描述，子密钥的具体计算过程见文献[6]。CMAC模式的MAC码产生过程如图2.11所示及CMAC模式的解密验证过程如图2.12所示。

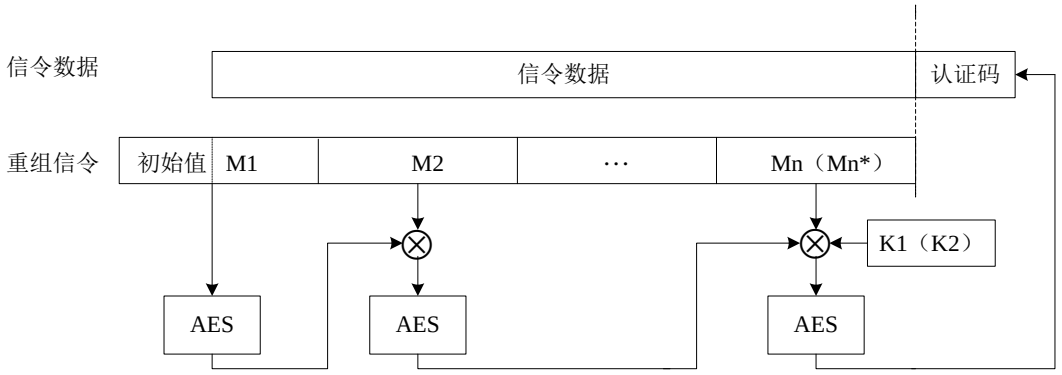

图 2.11 CMAC 模式的 MAC 码产生过程

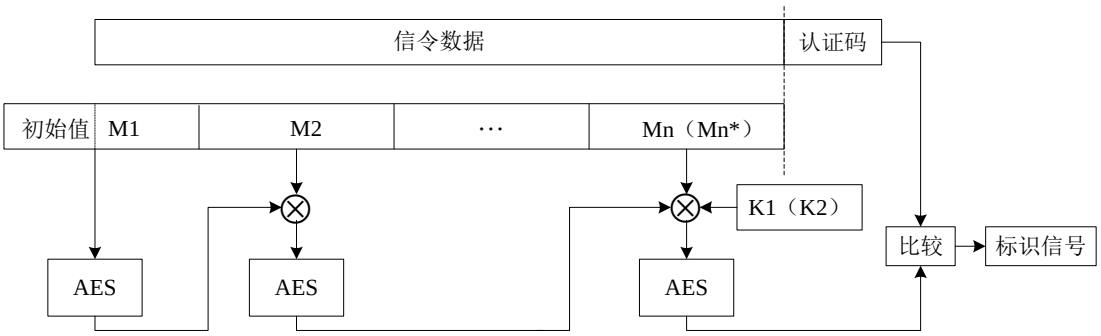

图 2.12 CMAC 模式的解密验证过程

加密认证开始，首先把原始信令数据进行分组，分为N组即$M_1…M_n$，每个分组块M_i为128bit。把当前分组块与前一次AES的结果按位异或后作为当前AES算法的输入。最后一个分组块M_n情况比较特殊，如果信令数据分块后M_n长度刚好为128bit，则M_n直接与前一次AES运算的结果及子密钥K1按位异或后作为最后一次AES算法的输入。如果M_n长度小于128bit，需要在分组块M_n后面补上一个“1”和若干个“0”使M_n的长度达到128bit，然后M_n与前一次AES运算的结果及K2按位异或后作为最后一次AES算法的输入。LTE协议规定采用CMAC模式运算结果的高有效位32bit作为认证码串连在原始信令数据后面。在接收端，需要对接收到的信令数据进行解密验证，以确定所接收的数据是否经过篡改及是否由合法的发送端发送。解密验证过程如图2.12所示，它对接收到的信令数据先进行解密认证处理，其处理过程和加密认证过程相同。产生32bit的认证码后和接收到的认证码进行比较，结果一致说明接收的消息合法，否则，说明接收的消息已被篡改，不响应该消息。

根据LTE协议规定，分组块$M_1…M_n$的分组模式为：

M_1 = COUNT || BEARER || DIRECTION || 0^{26} || $MESSAGE^{64}$;

M_2 = $MESSAGE^{128}$;

⋮

$M_n = MESSAGE^{L-128n+192}$;

$M_n^* = MESSAGE^{L-128n+192} \| 1 \| 0^{128n-L-65}$。

其中，MESSAGE表示信令数据，L表示MESSAGE的长度，M_n表示信令数据分块后最后一块长度正好为128bit的分组块，M_n*表示信令数据分块后最后一块长度小于128bit，进行数据填充后的分组块，$MESSAGE^{64}$表示MESSAGE的高64位。

2.5 SNOW 3G 算法

SNOW 3G算法是一种面向字的流加密算法，它在输入的128bit密钥K和128bit初始变量IV的控制下产生密钥流。密钥流由一系列32bit的字组成，把产生的密钥流与明文进行异或，从而实现对明文进行加密[16]。密钥流的产生由两步完成，首先初始化密钥，然后在时钟控制下产生密钥流。SNOW 3G算法加密输入需要密钥K和初始向量IV。该算法主要由基本功能函数、LFSR、FSM 和计时操作组成。

2.5.1 基本功能函数

SNOW 3G算法中有6个基本操作函数，分别用于转换输入的数据、数据加密和减小数据的线性相关性。下来分别介绍这几个函数：

（1）$MULx(V,c)$

该函数功能是将16bit数据转化为8bit数据。其中V为16bit数据的高8位，c为16bit数据的低8位。如果V的最高有效位为1，则$MULx(V,c)=(V<<_8 1)\oplus c$，否则，$MULx(V,c)=V<<_8 1$（$V<<_n t$表示在n位寄存器中左移t位）。

（2）$MULxPOW(V,i,c)$

该函数功能是将16bit数据和一个正整数i转化为8bit。其中V为16bit数据的高8位，c为16bit数据的低8位，i为正整数。如果i为0，$MULxPOW(V,i,c)=V$，否则，$MULxPOW(V,i,c)=MULx(MULxPOW(V,i-1,c),c)$。

（3）S-Box S_1

该函数功能是将32bit输入数据经过一系列操作转化为新的32bit数据作为输出，以减少数据相关性。$w = w_0 \| w_1 \| w_2 \| w_3$ 为32bit的输入数据，其中w_0为最高位，w_3最低位。每个字节经过字节代换操作后，$S_1(w) = r_0 \| r_1 \| r_2 \| r_3$ 为32bit的输出数据，其中r_0为最高位，r_3最低位。该处所使用的S_R盒和2.2.2②中的S盒相同。具体操作过程见文献[3]。

（4）S-Box S_2

该函数功能与S-Box S_1功能相同，操作过程也基本相同，具体操作过程见文献[3]，此处不再叙述。

（5）$MUL\alpha(c)$

该函数功能是将8bit数据转化为32bit数据，c为8bit的输入。则：

$$MUL\alpha(c) = (MULxPOW(c,23,0xA9) \| MULxPOW(c,245,0xA9) \| MULxPOW(c,48,0xA9) \| MULxPOW(c,239,0xA9))$$

（6） $DIV\alpha(c)$

该函数功能是将8bit数据转化为32bit数据，c为8bit的输入。则：

$$DIV\alpha(c) = (MULxPOW(c,16,0xA9) \| MULxPOW(c,39,0xA9) \| MULxPOW(c,6,0xA9) \| MULxPOW(c,64,0xA9))$$

2.5.2 结构说明

在SNOW 3G算法中主要包括LFSR和FSM这两个结构。

（1）LFSR

LFSR（Linear Feedback Shift Register）线性反馈位移寄存器，其中包括16个状态$s_0, s_1, s_2, \ldots, s_{15}$，每个状态32bit。LFSR不接受任何输入，对寄存器原有数据进行转化，生成LFSR的输出数据。假设$s_0 = s_{0,0} \| s_{0,1} \| s_{0,2} \| s_{0,3}$，其中$s_{0,0}$为高字节，$s_{0,3}$低字节。$s_{11} = s_{11,0} \| s_{11,1} \| s_{11,2} \| s_{11,3}$，其中$s_{11,0}$为高字节，$s_{11,3}$低字节。该结构的实现算法如下：

$s_0 = s_1$， $s_1 = s_2$， $s_2 = s_3$， $s_3 = s_4$， $s_4 = s_5$， $s_5 = s_6$， $s_6 = s_7$， $s_7 = s_8$， $s_8 = s_9$， $s_9 = s_{10}$， $s_{10} = s_{11}$， $s_{11} = s_{12}$， $s_{12} = s_{13}$， $s_{13} = s_{14}$， $s_{14} = s_{15}$， $s_{15} = v$。

$$v = (s_{0,1} \| s_{0,2} \| s_{0,3} \| 0x00) \oplus MUL\alpha(s_{0,0}) \oplus s_2 \oplus (0x00 \| s_{11,0} \| s_{11,1} \| s_{11,2}) \oplus DIV\alpha(s_{11,3}) \oplus F$$

（2）FSM

FSM（Finite State Machine）有限状态机，包括R_1、R_2和R_3这3个32bit寄存器。其中S-boxes S_1和S-boxes S_2用来更新寄存器R_2和R_3。假设FSM的输出为F，该结构的实现算法为：$F = (s_{15} \boxplus R_1) \oplus R_2$ 。其中，$R_3 = S_2(R_2)$， $R_2 = S_1(R_1)$， $R_1 = R_2 \boxplus (R_3 \oplus s_5)$（$\boxplus$为全加符号）。

2.5.3 SNOW 3G 算法操作

（1）初始化操作

Snow 3G算法的初始化操作是将128bit的密钥和初始向量进行初始化。其中密钥和初始向量分别包括4个32位字段k_0，k_1，k_2，k_3和IV_0，IV_1，IV_2，IV_3。其中k_0、IV_0为高有效位，k_3、IV_3为低有效位。

在初始化过程之前，LFSR中的$s_0, s_1, s_2, \ldots, s_{15}$和FSM中的$R_1$、$R_2$、$R_3$值均为0。在初始化过程开始后，FSM中的状态均设置为0，而LFSR中各部分的状态值由输入的密钥K和初始向量IV来计算，其计算过程为：$s_{15} = k_3 \oplus IV_0$，$s_{14} = k_2$，$s_{13} = k_1$，$s_{12} = k_0 \oplus IV_1$，$s_{11} = k_3 \oplus 1$，$s_{10} = k_2 \oplus 1 \oplus IV_2$，$s_9 = k_1 \oplus 1 \oplus IV_3$，$s_8 = k_0 \oplus 1$，$s_7 = k_3$，$s_6 = k_2$，$s_5 = k_1$，$s_4 = k_0$，$s_3 = k_3 \oplus 1$，$s_2 = k_2 \oplus 1$，$s_1 = k_1 \oplus 1$，$s_0 = k_0 \oplus 1$。

LFSR设置完毕后对FSM进行时钟操作产生32bit输出，然后对LFSR进行时钟操作，

该过程循环32次，这时就完成了初始化操作。SNOW 3G算法初始化操作过程如图2.13所示。

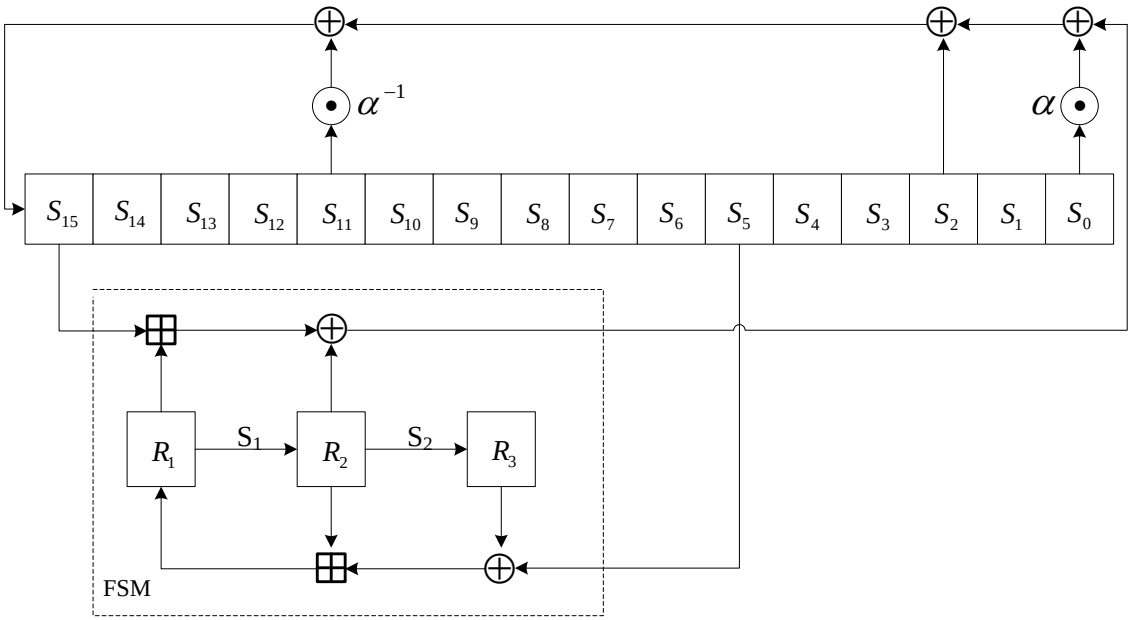

图 2.13 SNOW 3G 算法初始化过程

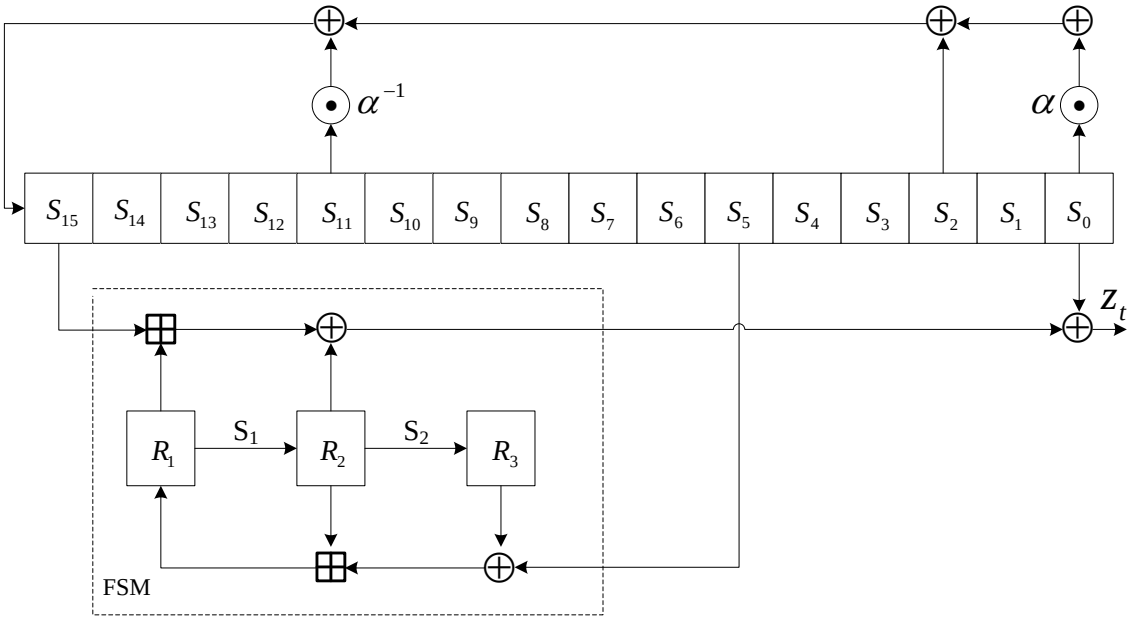

图 2.14 SNOW 3G 算法密钥流产生过程

（2）产生密钥流

完成初始化过程后进入密钥流生成过程，在此过程中FSM的输出不再进入LFSR，而

是作为整体的输出。FSM第一次输出将会被丢弃，其后的输出与s_0做异或运算后的结果作为密钥流完成对明文的加密过程（即密钥流与明文进行异或运算）。其中密钥流以32bit字的形式输出，输出字的个数由参数LENGTH来决定[17]。 SNOW 3G算法密钥流产生过程如图2.14所示。

2.6 UEA2 加密算法

UEA2算法是以SNOW 3G算法为基本单元的，它的功能是对无线链路中用户数据进行加密。通过初始密钥CK和一系列高层配置的参数COUNT、BEARER和DIRECTION经过算法操作产生新的密钥K和初始向量IV做为SNOW 3G算法的输入，通过SNOW 3G算法产生密钥流与明文数据进行异或操作产生密文。UEA2算法实现过程如图2.15所示，其中z_1，z_2 …z_L为密钥流。

生成密钥K和初始向量IV的算法为（IV_0、K_0 、CK[0]代表各自的高有效位）：

K_3 = CK[0] || CK[1] || CK[2] || … || CK[31]；

K_2 = CK[32] || CK[33] || CK[34] || … || CK[63]；

K_1 = CK[64] || CK[65] || CK[66] || … || CK[95]；

K_0 = CK[96] || CK[97] || CK[98] || … || CK[127]；

IV_3 = COUNT[0] || COUNT[1] || COUNT[2] || … || COUNT[31]；

IV_2 = BEARER[0] || BEARER[1] || … || BEARER[4] || DIRECTION[0] || 0 || … || 0；

IV_1 = COUNT[0] || COUNT[1] || COUNT[2] || … || COUNT[31]；

IV_0 = BEARER[0] || BEARER[1] || … || BEARER[4] || DIRECTION[0] || 0 || … || 0。

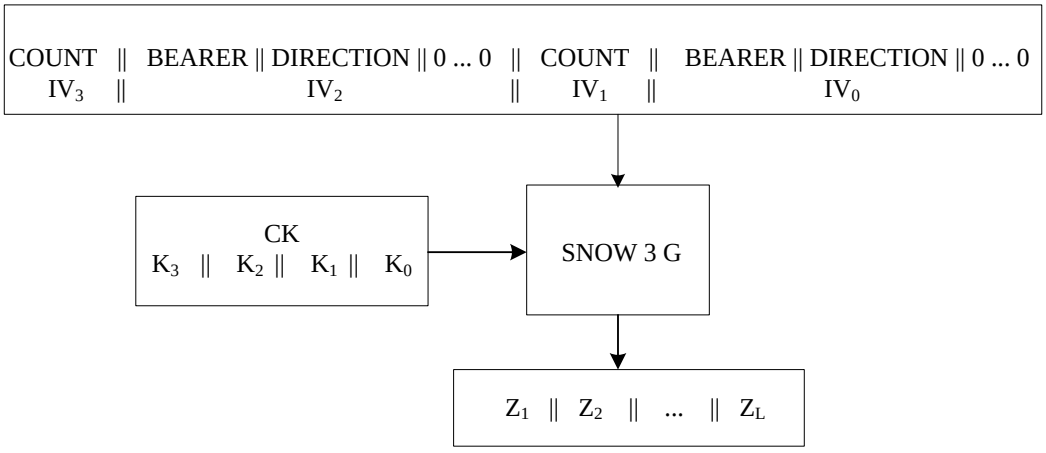

图2.15 UEA2算法实现过程

2.7 UIA2 完整性保护算法

UIA2算法是以SNOW 3G算法为基本单元进行操作的，它的功能是对无线链路中信

令进行完整性保护。它和AES-CMAC模式一样将产生的32位认证码附在明文后面。下来介绍UIA2算法中的基本函数和算法的操作过程。

2.7.1 基本函数

（1） $MULx(V,c)$

该函数功能是将128bit数据转化为64bit数据。其中V为128bit数据的高64位，c为128bit数据的低64位。如果V的最高有效位为1，则 $MULx(V,c)=(V<<_{64}1)\oplus c$，否则，$MULx(V,c)=V<<_{64}1$。

（2） $MULxPOW(V,i,c)$

该函数功能是将128bit数据和一个正整数i转化为64bit。其中V为128bit数据的高64位，c为128bit数据的低64位，i为正整数。如果i为0，$MULxPOW(V,i,c)=V$，否则，$MULxPOW(V,i,c)=MULx(MULxPOW(V,i-1,c),c)$。

（3） $MUL(V,P,c)$

该函数功能是将192bit数据转化为64bit的数据，V和c为64bit的输入，*result*为$MUL(V,P,c)$的结果。对于i为0到63的范围内，如果P的第i位为1，则$result=result\oplus MULxPOW(V,i,c)$，否则*result*的值不变，*result*初始值为0。

2.7.2 算法操作过程

首先初始化，通过初始密钥IK和一系列高层配置的参数COUNT、BEARER和DIRECTION经过算法操作产生新的密钥K和初始向量IV做为SNOW3G算法的输入，产生z_1，z_2，z_3，z_4，z_5五个32bit的密钥流，如图2.16 UIA2算法操作过程(a)所示。然后对明文进行EVAL函数操作后取高32位与z_5进行异或运算就得到所需的32位认证码。如图2.16 UIA2算法操作过程(b)所示，初始化算法为（IV_0、K_0、IK[0]代表各自的高有效位）：

K_3=IK[0] || IK[1] || IK[2] || … || IK[31]；

K_2=IK[32] || IK[33]|| IK[34] || … || IK[63]；

K_1=IK[64] || IK[65] || IK[66] || … ||IK[95]；

K_0=IK[96] || IK[97] || IK[98] || … || IK[127]；

IV_3=COUNT[0] || COUNT[1] || COUNT[2] || … || COUNT[31]；

IV_2=FRESH[0] || FRESH[1] || FRESH[2] || …|| FRESH[31]；

IV_1=DIRECTION[0] ⊕COUNT[0] || COUNT[1] || COUNT[2] || … || COUNT[31]；

IV_0=FRESH[0] || FRESH[1] || … || FRESH[15] || FRESH[16]⊕DIRECTION[0] || FRESH[17] || … || FRESH[31]。

其中，FRESH =BEARER[0] || … || BEARER[4] || 0^{27}。

对明文进行*EVAL*函数操作时需要将明文分组，每组64bit（用M_i表示第i组数据）。

*EVAL*函数计算过程如下，LENGTH为需要进行完整性保护的数据长度。

定义D = ⌈LENGTH / 64⌉ + 1，P = $z_1 \| z_2$，Q = $z_3 \| z_4$，z_5 = OTP[0] ‖ OTP[1] ‖… ‖ OTP[31] 。函数 *EVAL* 的初始值为 0，对于 i 从 0 到 D-2，$EVAL = MUL(EVAL \oplus M_i, P, 0x0000000000000001b)$，接着计算 $EVAL = EVAL \oplus M_D - 1$（$M_D - 1$为明文长度值），最后 $EVAL = MUL(EVAL, Q, 0x0000000000000001b)$。将*EVAL*的高32位与$z_5$进行异或运算后得到所需的32位认证码。

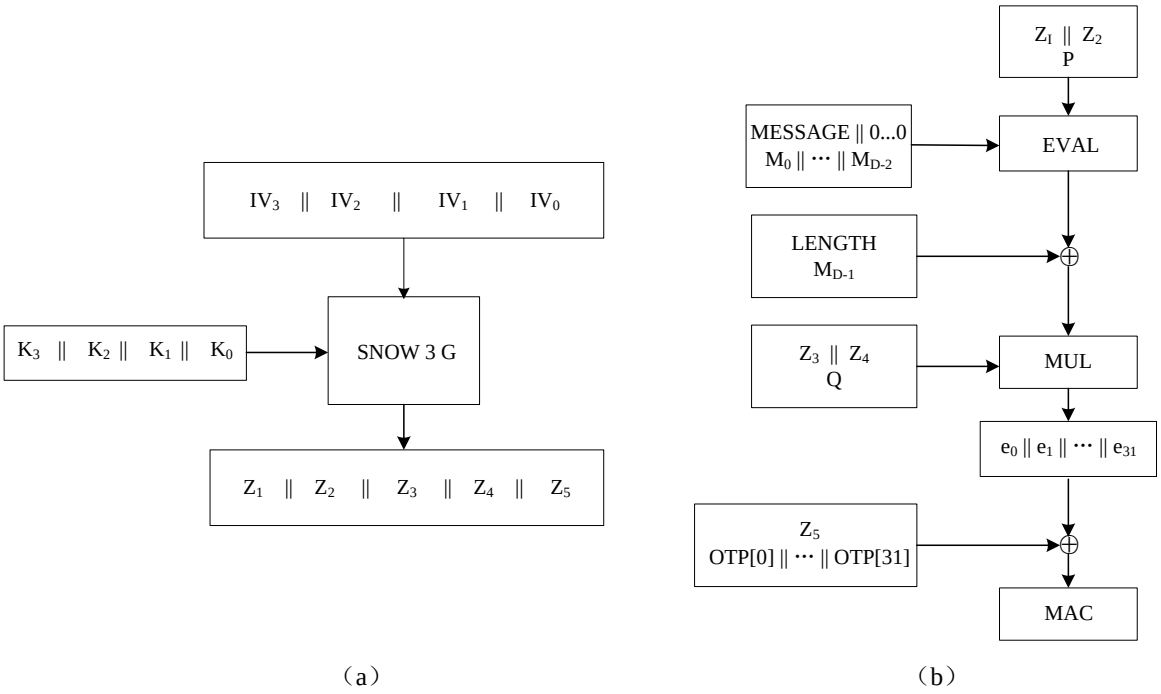

图 2.16 UIA2 算法操作过程

2.8 本章小结

本章首先介绍了LTE终端的加解密和完整性保护过程，接着对加密解密和完整性保护过程中所使用的AES算法及CTR加密模式和CMAC完整性计算模式进行介绍，最后对SNOW 3G算法及UEA2算法和UIA2算法进行了介绍。

3 硬件加速器设计

LTE 协议规定，在 PDCP 层中，需要对信令进行加密和完整性保护双重保护，对于用户数据需要进行加密。本文所设计的硬件加速器就是完成了 PDCP 层中的加密和完整性保护以及解密和完整性验证工作。

本章首先对 ARM 公司的 AMBA（Advanced Microcontroller Bus Architecture）总线和基于 AMBA 总线的 LTE 终端 SOC 系统进行介绍，接着以该系统为背景，对基于 AHB（Advanced High Performance Bus）总线的加解密硬件加速器的数据交互流程进行描述，以及对该硬件加速器进行模块划分并且对各模块进行设计。

3.1 LTE 终端 SOC 系统架构

AMBA 总线规范是由 ARM 公司设计的用于高性能嵌入式系统，为实现片上 IP 核互联而提出的总线规范。AMBA 总线规范是一个开放的标准，可以免费从 ARM 公司获得。AMBA 总线标准定义了 AHB、APB（Advanced Peripheral Bus）和 ASB（Advanced System Bus）这三种不同但是可以组合使用的总线。AHB 总线适用于连接高速设备，如 CPU、DMA(Direct Memory Access)和 SDRAM(Synchronous Dynamic Random Access Memory)等；APB 总线是速度相对较慢的总线，适用于连接系统外部的低功耗设备，如 UART、键盘、SPI 等设备。ASB 总线适用于高性能系统模块，在不适用 AHB 的高速场合使用，它同样支持 CPU、DMA 和 SDRAM 等[18]。这三种总线的详细介绍参考文献[19]。

本文研究的 LTE 终端是以 ARM 公司提供的基于 AMBA 总线的 SOC 系统架构为基础而开发的。基于 AMBA 总线的 LTE 终端 SOC 系统架构如图 3.1 所示。

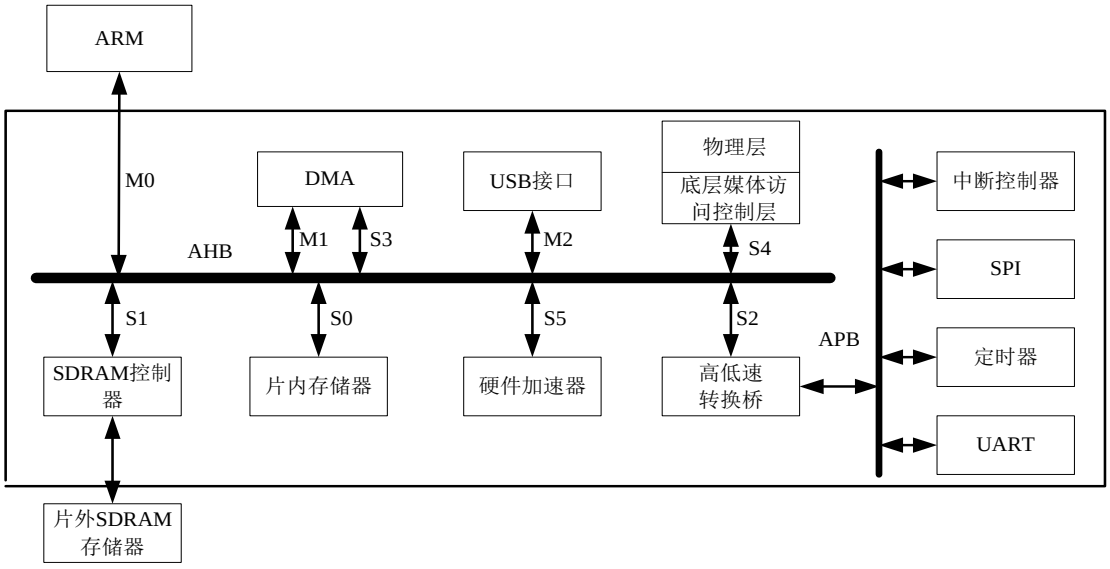

图 3.1 LTE 终端 SOC 系统架构

其中 AHB 总线主要是为了满足处理器与高速存储器间的带宽要求,它作为系统中枢总线用于连接 ARM、片内和片外存储器以及硬件加速器等这些高速模块。在 AHB 总线上挂载一个 DMA 模块，目的是为了提高 AHB 总线上大容量数据的传输速度。它接收到控制信息后，也可以不通过总线传输数据，以获得更高的传输速度。APB 总线用于连接系统中的中断控制器和计数器以及系统外部的 SPI（串行外设接口）和 URAT（通用异步收发器）这些低功耗设备。该系统中 AHB 有 3 个 master 设备 M0、M1 和 M2，5 个 slave 设备 S0、S1、S2、S3、S4、S5 和 S6。该系统以 ARM 处理器作为核心，根据需要启动各模块进行有序的工作。系统中总线带宽采用 32bit。

系统中的硬件加速器（即 S5）实现了 PDCP 层对数据的加密和完整性保护以及解密和完整性验证功能，是本文的主要工作所在。接下来主要介绍硬件加速器的研究与设计。

3.2 硬件加速器数据交互流程

本文中所设计的加硬件加速器是在ARM处理器的调度下实现数据的加密和完整性保护功能的。该硬件加速器的数据交互流程如图3.2所示。

系统在上电复位后，ARM 需要调用硬件加速器工作时将指令写入输入型 FIFO，同时在系统上电复位后，硬件加速器开始判断输入型 FIFO 深度是否大于或等于 2，当输入型 FIFO 深度大于或等于 2 时表示有描述符需要处理。则硬件加速器开始对输入型 FIFO 进行解析，获得描述符所在 ARM 内存（SDRAM）中的地址和描述符长度信息。否则，硬件加速器继续判断输入型 FIFO 的深度。如果解析出的描述符地址和描述符长度都为零，则硬件加速器向 ARM 处理器发出中断请求，而硬件加速器返回到初始状态继续判断输入型 FIFO 的深度。否则，调用 DMA，将描述符的地址和长度信息配置给 DMA，DMA 将描述符从 SDRAM 中搬运到硬件加速器中的 dscp_mem（RAM）中。描述符搬运结束后，硬件加速器开始对描述符进行解析（描述符结构见 3.3.7 节），获得需要处理的数据起始地址，数据长度等信息。然后根据解析出的数据起始地址和数据长度信息再次配置 DMA 将数据从 SDRAM 中搬运到硬件加速器中的 data_mem（RAM）中。如果从描述符中解析出的本次加密或完整性计算算法为 AES 算法，则在 DMA 搬运数据同时进行密钥扩展处理（先进行加密密钥扩展，然后进行完整性保护密钥扩展）。否则，不需要密钥扩展过程。等待数据搬运结束，根据软件配给描述符中加密算法及完整性保护算法的类型调用相应加密算法及完整性保护算法对数据进行安全处理（完整性计算过程和加密过程同时进行），最后将处理结果存入 data_mem 中。当数据处理结束之后，再次配置 DMA 将处理后的数据搬回 SDRAM 中相应的地址，同时将本次处理的描述符地址及完整性验证结果存入输出型 FIFO 中。这时，一次加密或解密过程结束，硬件加速器回到初始状态再次判断输入型 FIFO 是否有描述符需要处理，整个过程如此循环进行。

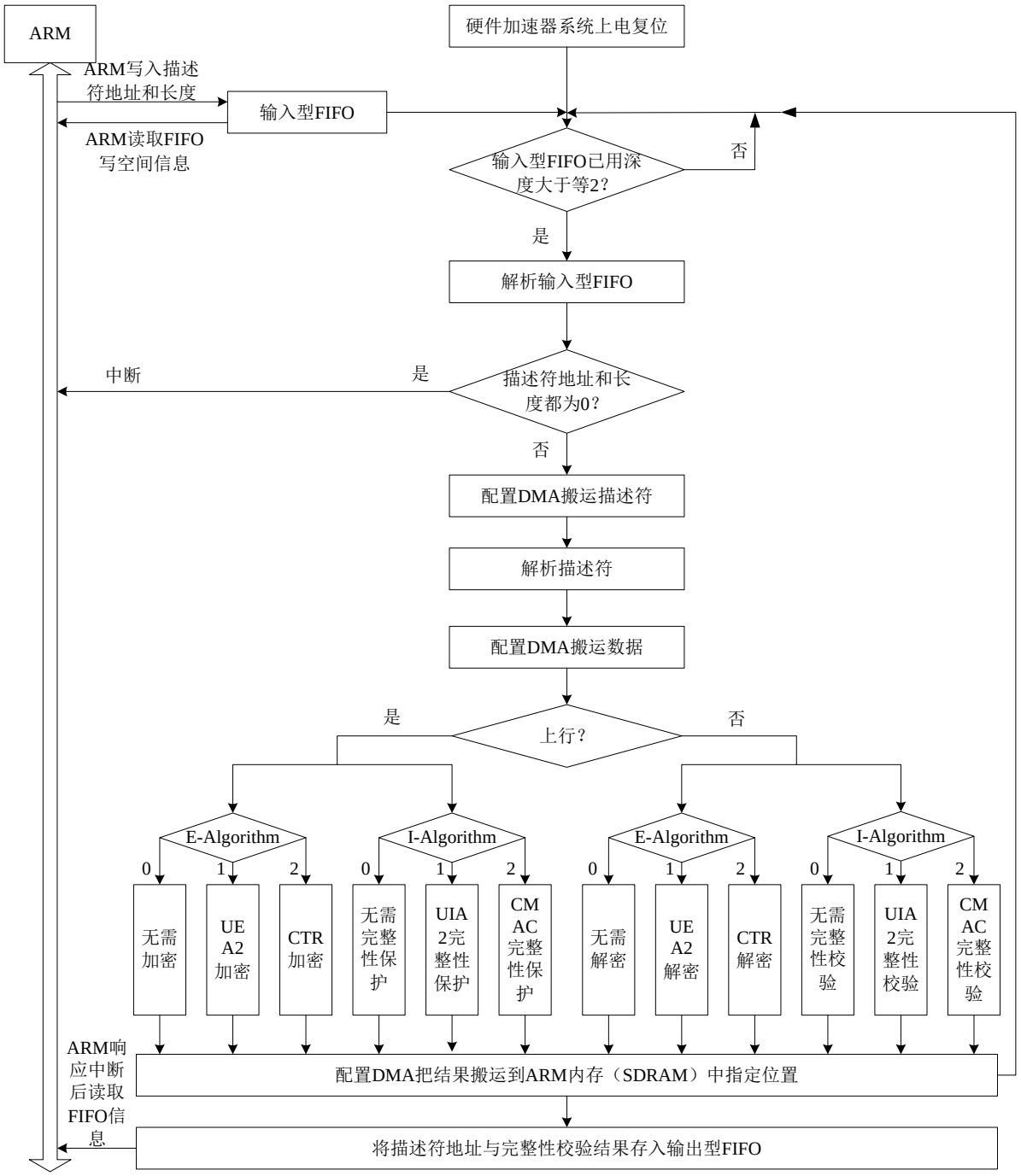

图 3.2 硬件加速器的数据交互流程

每次往输入型 FIFO 中写入描述符地址和长度时软件要进行判断硬件加速器中输入型 FIFO 是否有足够的空间接收。如果空间不够，则将这些信息加入待处理链表中。否则，直接写入输入型 FIFO。而后循环一次将待处理链表中的首节点信息写入输入型 FIFO，直到 FIFO 空间不够或待处理链表为空。这样做的目的是为了保证数据的正确写入而不发生溢出，也就是确保 FIFO 在写满的情况下，不再进行写操作。

3.3 硬件加速器模块结构设计

本文所设计的硬件加速器主要由逻辑功能控制模块、密钥存储模块、AHB 总线 slave 设备接口模块、密钥扩展模块、DMA 模块、输入型 FIFO、输出型 FIFO、描述符 RAM 模块、密钥存储模块、数据存储模块和加解密 IP 核模块这八部分组成。各模块在功能控制逻辑模块的协调下有序的进行工作，由图 3.3 所示为硬件加速器的模块结构图。其中，加解密 IP 核是实现加密和完整性保护的核心算法模块。它同时支持基于 AES 算法的 CTR 模式和 CMAC 模式以及支持基于 SNOW 3G 算法的 UEA2 算法和 UIA2 算法。该模块在下一章单独进行详细设计。

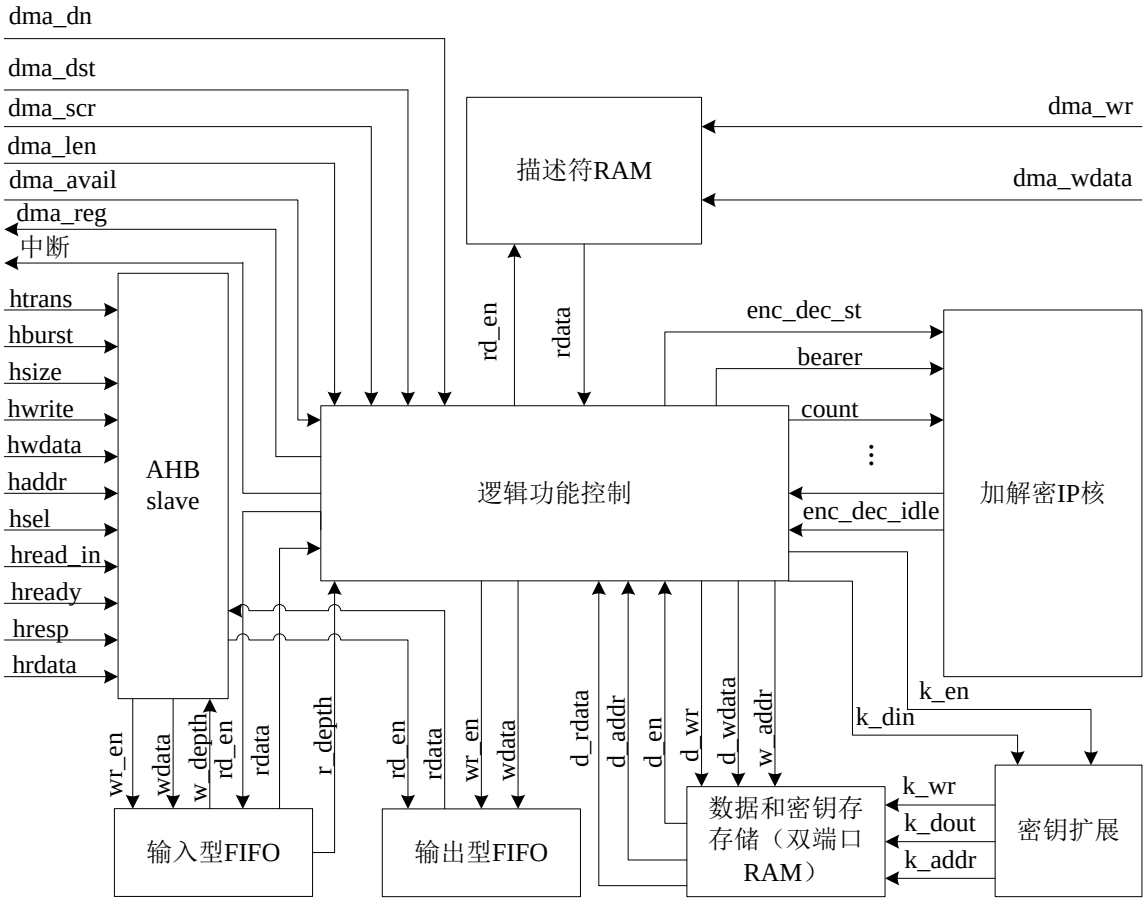

图 3.3 硬件加速器的模块结构图

3.3.1 逻辑功能控制模块

逻辑功能控制模块主要功能是协调各个模块进行有序的工作，如图 3.4 所示硬件加速器逻辑功能控制模块实现流程。

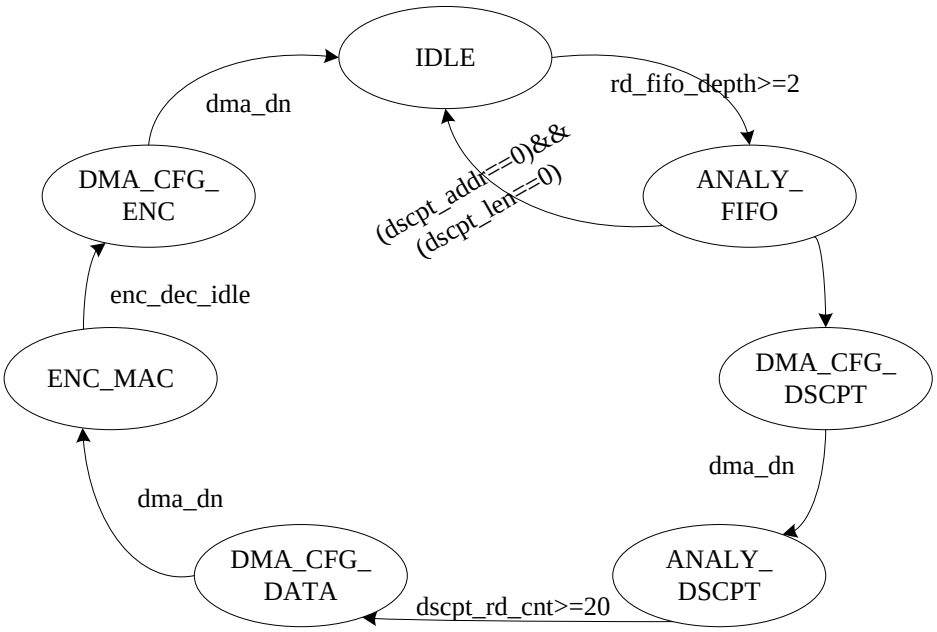

图 3.4 硬件加速器逻辑功能控制模块实现流程图

系统上电复位后，状态机处于 IDLE 状态，且系统每个时钟周期检测输入型 FIFO 已用深度是否大于或等于 2。如果当输入型 FIFO 已用深度大于或等于 2 时，状态机跳转到 ANALY_FIFO 状态，否则状态机返回 IDLE 状态。状态机进入 ANALY_FIFO 状态后开始对输入型 FIFO 进行解析，解析出描述符所在 ARM 内存中的地址和描述符大小信息。如果解析出的描述符地址和长度为“0”，则硬件加速器向 ARM 处理器发出中断请求，否则状态机跳转到 DMA_CFG_DSCPT 状态，在此状态向 DMA 发送使用请求信号，等待 DMA 响应请求之后，开始配置 DMA 寄存器进行描述符的搬移，即把描述符从 ARM 内存中搬移到硬件加速器中的描述符 FIFO 中。之后等到 DMA 工作结束信号 dma_dn 有效后状态机跳转到 ANALY_DSCPT 状态，开始解析描述符，等待描述符解析完毕后，在 DMA_CFG_DATA 状态下再次配置 DMA 从 ARM 内存中搬运需要进行加密和完整性保护的数据到硬件加速器中的数据存储 RAM 中。等待 DMA 搬运结束后状态机进入 ENC_MAC 状态，在该状态下调用加解密 IP 核进行加密和完整性保护处理。等待加解密 IP 核工作完毕后即 enc_dec_idle 信号有效后状态机跳转到 DMA_CFG_ENC 状态，在该状态下再次调用 DMA 把处理后的数据搬运到 ARM 内存中，同时将本次处理的描述符地址及完整性验证结果存入输出型 FIFO 中。DMA 搬运完数据后状态机进入 IDLE 状态。

3.3.2 密钥扩展模块

（1）密钥扩展模块设计

密钥扩展模块通过输入一个原始密钥，产生 AES 核每一轮变换所需的轮密钥，使加

密和解密的密钥具有更大的离散型，增加破解的难度。密钥扩展一般采用并行扩展和非并行扩展两种方案来实现。第一种方案是在 AES 加密过程中同时进行密钥扩展，每次产生一轮密钥[21]。第二种方案就是先将所有的轮密钥一次生成并进行存储，在加解密及完整性计算时通过 AES 核读取存储器获取轮密钥[22]。第一种方案比较省资源，但是轮密钥没有存储功能，每次加、解密时都需要进行密钥扩展。如果多个数据包采用相同密钥加、解密时需要重复计算密钥，功耗增大。第二种方案首先只在每一个数据包处理完且下一个数据包的初始密钥有改变时才进行密钥扩展，不需要每次加、解密都进行密钥扩展，大大降低了功耗，但是需要额外的存储器来存储密钥。其次，在密钥变换情况下，只在初始时刻进行密钥扩展，其后密钥调度的时序控制相对容易许多，方便使用。考虑到功耗、速度和面积这些因素，因此在本设计中采用第二种方案，从描述符中解析出初始密钥后开始计算轮密钥并存储到密钥存储器中，然后在加、解密时通过 AES 核读取密钥存储器获得轮密钥。

密钥扩展模块硬件设计结构如图 3.5 所示。

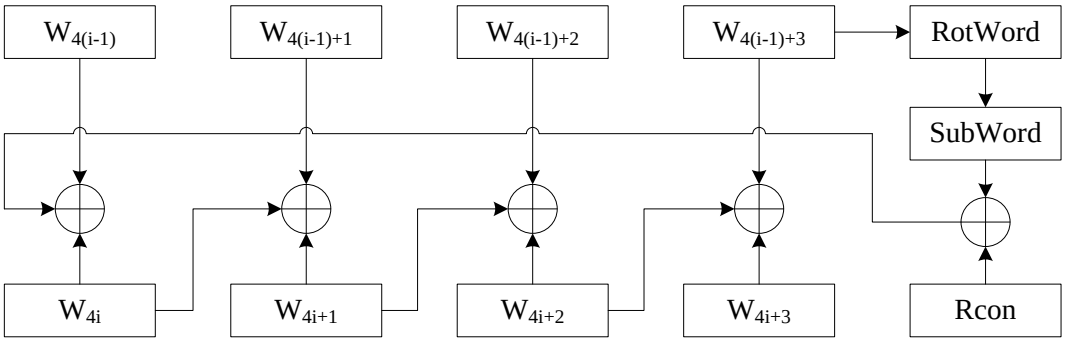

图 3.5 密钥扩展模块硬件设计结构图

RotWord 变换和 SubWord 变换分别与 AES 核中的 ShiftRows 和 SubBytes 在操作上是一样的。Rcon[i]操作是把经过 Subword 变换后的变量与该轮常数 Rcon 做异或运算，每一个轮密钥的产生都需要用到一个轮常数，所以 10 个轮变换共需要用到 10 个不同的轮常数。由轮常数定义可知每个轮常数中只有高位一个字节的数据有变化，其它的三个字节都为零。由于任何数据与 0 进行异或运算其值不变，所以低位三个字节不做任何处理。本设计采用查找表的方式，只需一个存 10 个字节的表格，从而减少了硬件开销。

（2）密钥扩展模块实现过程状态图

如图 3.6 所示为密钥扩展模块实现过程状态图。系统上电复位后状态机一直处于 IDLE 状态，当 k_en 使能时，开始读取初始密钥进入 ROTWORD 状态。在 ROTWORD 状态时完成对 key3 循环左移一个字节且计数器（cs==ROTWORD && ns==SUB_BYTEF 时）开始计数，在完成这个过程后进入字节代换状态，这个状态需要两个时钟周期完成

（所以本模块中采用两个状态 SUB_BYTEF 和 SUB_BYTES）。接着分别在 K0、K1、K2、K3 状态产生 key0、key1、key2、key3。在 K3 状态时产生新密钥及存储密钥所需的使能和地址。在此状态下判断计数 key_cnt，如果 key_cnt<10 则跳转到 ROTWORD 状态继续工作，否则计数器清零且跳转到 IDLE 状态结束操作。该模块输出地址从 0 开始，也就是从初始密钥开始总共输出 11 个密钥到密钥存储器。

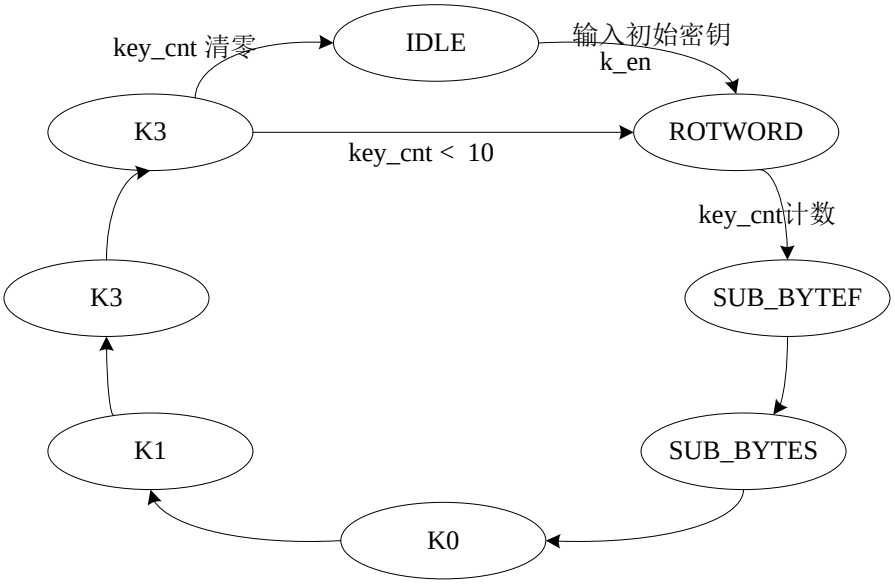

图 3.6 密钥扩展模块实现过程状态图

3.3.3 AHB 总线 slave 设备接口模块

AHB slave 模块主要完成 ARM 处理器和加解密硬件加速器之间的数据传输。AHB slave 主要功能是把 AHB 总线接口信号转化为几个简单的控制信号，目的是为了简化输入型 FIFO 的接口。ARM 处理器将需要处理的数据通过该接口写入输入型 FIFO，并对该 FIFO 的空间剩余信息进行读取。

如图 3.7 所示为 AHB 总线的一次基本传输时序，由图 3.7 可以看出一次完整的 AHB 传输分为两个阶段：地址阶段（地址相位）和数据阶段（数据相位）。地址阶段只维持一个时钟周期，而数据阶段可以在多个时钟周期完成，通过使用 HREADY 信号控制实现。若一个 AHB 传输需要等待，Slave 可以通过拉低 HREADY 信号（即 HREADY 为“0”）来延长传输时间。AHB 总线采用两级流水线型，本次的数据传输阶段与下次传输的地址在同一个时钟周期。有关 AHB 总线的具体描述参考文献[20]。本设计中使用了现有的 AHB slave 模块，没有对其进行设计，本文中不做详细介绍。

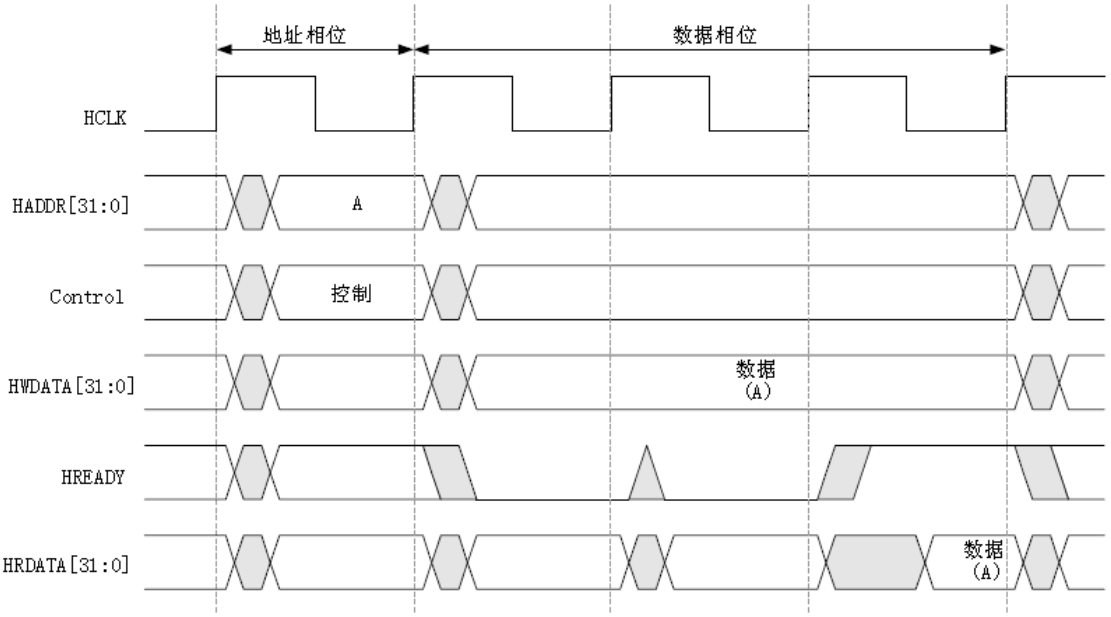

图 3.7 AHB 总线基本传输时序

3.3.4 DMA 模块

本系统使用 DMA 搬运数据的目的是为了提高 AHB 总线上大容量数据的传输速度，经过 DMA 搬运数据不经过处理器不但减轻了处理器的负担，而且使数据的传输速度大大的提高了。在硬件加速器工作过程中需要进行数据搬运时，发出申请使用 DMA 信号 dma_reg，然后等待 DMA 返回 dma_avail 信号后开始发出 dma_st 信号，同时将传输需要的长度信息 dma_len、源起始地址 dma_src 和目标搬运地址 dma_dst 配给 DMA。DMA 接收到配置信息后开始搬运数据。直到 DMA 工作结束信号 dma_dn 有效时，一次搬运结束。本设计中只是调用系统中的 DMA 进行数据搬运，没有进行设计，本文中不做详细介绍。

3.3.5 输入/输出型 FIFO 模块

输入型 FIFO 用于保存 ARM 给硬件加速器配置的描述符地址及描述符长度，其存储结构如图 3.8 所示。其中 addr（32bit）表示描述符所在 SDRAM 中的地址，len（32bit）表示描述符长度。

输入型 FIFO 所用 FIFO 模块是由 xilinx 公司提供的 IP 核，它经过充分测试和优化，系统运行稳定且占用 FPGA 内部资源也比较少。由于 ARM 和硬件加速器工作在不同的时钟域，所以该模块采用异步 FIFO。本模块所用 FIFO 是通过 ISE 工具产生了一个位宽为 32，深度为 1024 的异步 FIFO 模型。该 FIFO 模型不但提供了一些基本的控制和数据

接口，而且还提供了 FIFO 空满标志接口和已用空间接口。

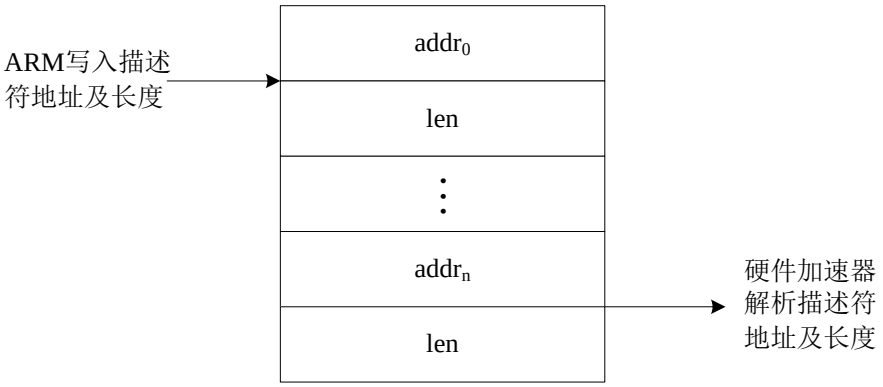

图 3.8 输入型 FIFO 存储结构图

输出型 FIFO 用于保存硬件加速器处理过的描述符地址及完整性校验结果，其存储结构如图 3.9 所示。其中 addr（32bit）表示描述符所在 SDRAM 中的地址，rst（32bit）表示完整性验证结果，低有效位为“1”表示验证通过，为“0”表示验证失败。（完整性校验时，如果校验失败则在 rst 字段低有效位置 0 值，其他情况 rst 字段低有效位置 1 值）。

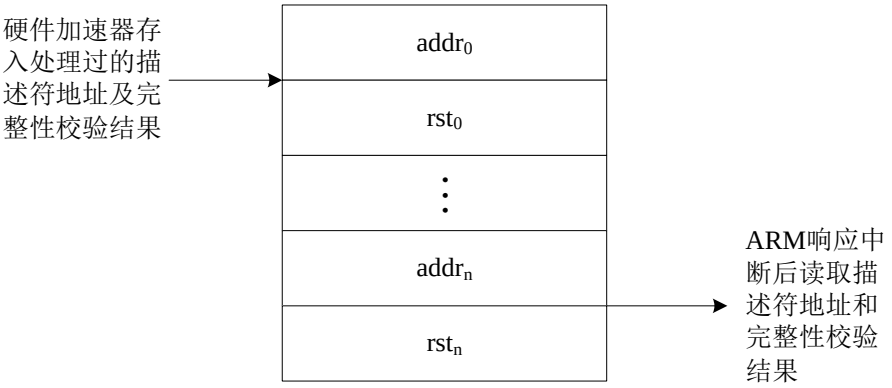

图 3.9 输出型 FIFO 存储结构图

本模块所用 FIFO 模型与输入型 FIFO 模型相同。每一次加密或解密过程结束，硬件加速器通过输出型 FIFO 的 wdata[31:0]接口将本次处理过的描述符地址及完整性校验结果写入。ARM 接受中断请求后，通过输出型 FIFO 的 rdata[31:0]接口将该信息读出。

3.3.6 描述符 RAM 模块

描述符 RAM 用于保存 ARM 所配给硬件加速器在加解密过程中所需的参数，这些参数经由 DMA 从 ARM 内存中搬运到描述符 RAM 中，描述符 RAM 编码结构如图 3.10 所示。本设计中所用 RAM 模块是由 xilinx 公司提供的 IP 核，通过 ISE 工具产生了一个位宽为 32，深度为 32 的双端口 RAM 模型。这两个端口中，一个端口用于 DMA 将搬运

的参数写入，另一个端口用于硬件加速器对描述符的解析。

<table>
<tr><td colspan="4">E-out saving addr（32bit）</td></tr>
<tr><td>E-Algorithm
(4bit)</td><td colspan="2">I-Algorithm
(4bit)</td><td>Padding
（23bit）</td></tr>
<tr><td colspan="4">E-key (128bit)</td></tr>
<tr><td colspan="4">I-key (128bit)</td></tr>
<tr><td colspan="4">sub-key1 (128bit)</td></tr>
<tr><td colspan="4">sub-key2 (128bit)</td></tr>
<tr><td colspan="4">COUNT (32bit)</td></tr>
<tr><td>BEARER
(5bit)</td><td>DIRECTION
(1bit)</td><td>data-len
(13bit)</td><td>Padding
(13bit)</td></tr>
<tr><td colspan="4">data-addr (32bit)</td></tr>
</table>

图 3.10 描述符 RAM 编码结构

其中：

（1）E-out saving addr：加解密及完整性计算后的结果存储地址（字对齐）。

（2）E-Algorithm：加解密算法选择，取值及意义：“0000”表示不需要加密。“0001”表示用基于 SNOW 3G 算法的 UEA2 模式进行加解密。“0010”表示用基于 AES 算法的 CTR 模式进行加解密。

（3）I-Algorithm：完整性保护算法选择，取值范围及意义为：“0000”表示无需完整性保护算法计算/验证。“0001”表示用基于 SNOW 3G 算法的 UIA2 模式进行完整性保护计算/验证。“0010”表示用基于 AES 算法的 CMAC 模式进行完整性保护计算/验证。

（4）Padding： 填充值项，无意义。

（5）E-key：加密的初始密钥。

（6）I-key：完整性保护的初始密钥。

（7）sub-key1：子密钥 1，其他情况下该项没意义。

（8）sub-key2：子密钥 2，其他情况下该项没意义。

（9）COUNT：计数器，加解密和完整性保护计算的参数，由高层配置。

（10）BEARER：链接标识，加解密和完整性保护计算的参数，由高层配置。

（11）DIRECTION：上下行方向指示标志，加解密和完整性保护计算的参数，由高层配置。

（12）data-len：需要处理的一包数据长度。

（13）data-addr：ARM 内存中需要处理的数据首地址，用来配置 DMA 搬运数据。

3.3.7 密钥和数据存储模块

密钥存储模块用于存储经过密钥扩展处理后的密钥及初始密钥，存储 AES 算法加、解密过程中所需的十一个 128bit 的密钥。它和描述符 RAM 一样是由 ISE 工具产生两个位宽为 128，深度为 16 的双端口 RAM 模型，通过调用 FPGA 芯片内部提供的 RAM 模块来实现的。这两个 RAM，一个用来存储加密密钥，一个用来存储完整性保护密钥。当密钥扩展过程完成后通过相应的密钥存储模块的输入端口将密钥扩展模块产生的密钥及初始密钥写入，当加解密 IP 核发出密钥读取使能信号后相应的密钥存储模块输出端口将密钥输出给加解密 IP 核。

数据存储模块用于存储 DMA 从 ARM 内存中搬运过来的需要处理的数据。该模块和密钥扩展模块一样采用双端口 RAM，通过 ISE 工具产生二个位宽为 128，深度为 512 的双端口 RAM 模型。其中一个用于存储加密数据，另外一个用于存储完整性计算数据。由于需要处理的一包数据最大长度为 8188byte，所以此处 RAM 深度设置为 512。在逻辑功能控制模块控制下 DMA 将从 ARM 内存中搬运过来的数据通过该模块的输入端口写入，当加解密核发出读取数据使能后，该模块通过输出端口将数据输出给加解密 IP 核。

3.4 本章小结

本章首先对 ARM 公司的 AMBA 总线和基于 AMBA 总线的 LTE 终端 SOC 系统进行介绍，接着以该系统为背景，对基于 AHB 总线的加解密硬件加速器的数据交互流程进行描述，以及对该硬件加速器进行结构设计和模块划分，并且对部分模块进行详细的设计。

4 加解密 IP 核设计

本章将具体介绍加解密 IP 核的设计，首先对加解密 IP 核的总体结构框架进行设计，接着对 CMAC 模式和 CTR 模式以及 UIA2 和 UEA2 算法模块设计，最后两节单独详细介绍 AES 算法和 SNOW 3G 算法的硬件实现。

4.1 加解密 IP 核设计

本章所设计的加解密 IP 核是实现对信令数据和用户数据进行加密和完整性保护的核心算法部分。它主要由控制模块、AES-CMAC 完整性计算模块、AES-CTR 加密模块、UIA2 完整性计算模块和 UEA2 加密模块这五部分组成。各模块在功控制模块的协调下有序的进行加密和完整性计算工作，由图 4.1 所示为加解密 IP 核模块结构图。

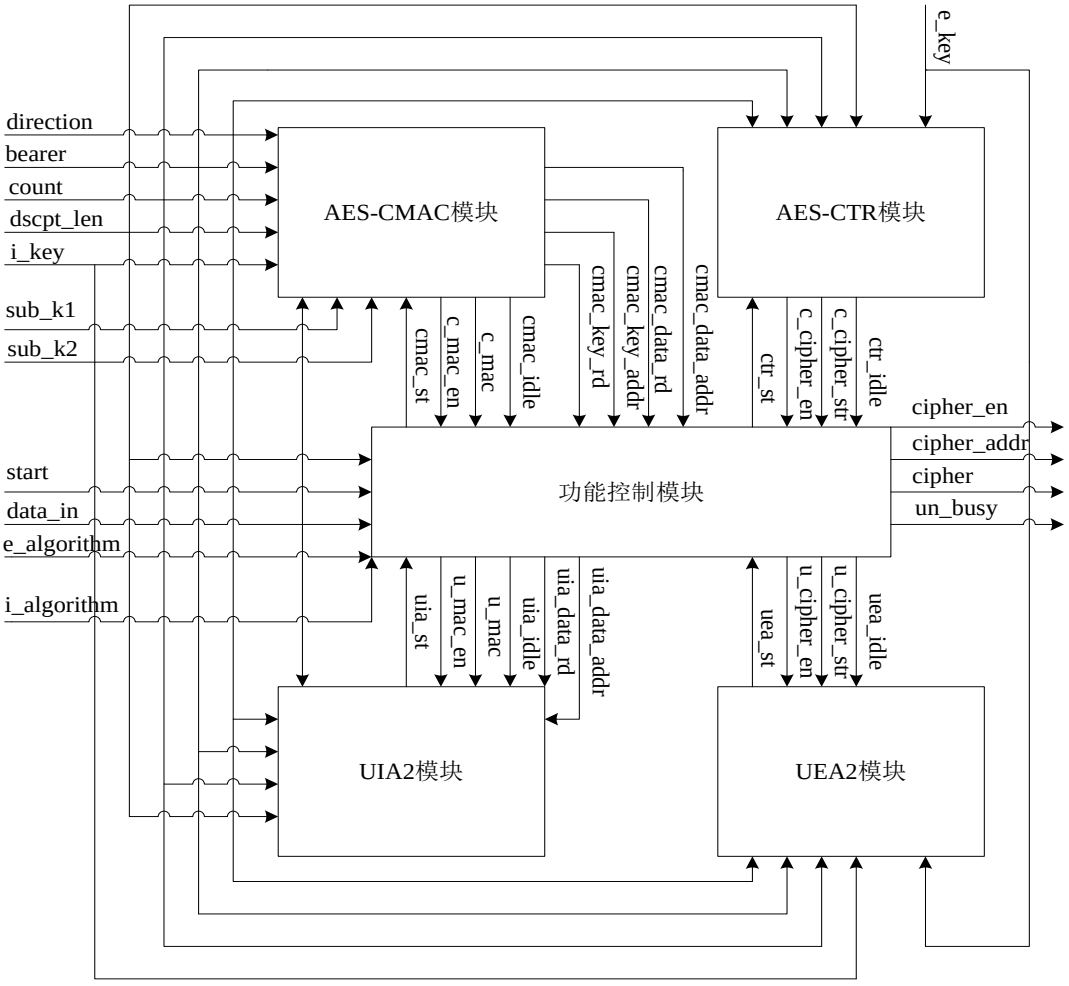

图 4.1 加解密 IP 核模块结构图

其中，CMAC 模式和 CTR 模式是基于 AES 算法实现的，而 UEA2 和 UIA2 是基于 SNOW 3G 算法实现的。AES 算法和 SNOW 3G 算法的设计分别在 4.2 节和 4.3 节中介绍。本章所设计的 CMAC 完整性计算模块、CTR 加密模块、UIA2 完整性计算模块和 UEA2 加密模块在控制模块的控制下可以并行工作。接下来介绍各模块的设计过程。

4.1.1 控制模块

控制模块主要功能是根据软件配置的标识符来选择需要工作的算法模块，以及协调加密模块和完整性计算模块有序的进行加密和完整性计算工作，最后负责将产生的密文数据和 MAC 码存入数据存储模块中。如图 4.2 所示为控制模块的实现过程状态图。

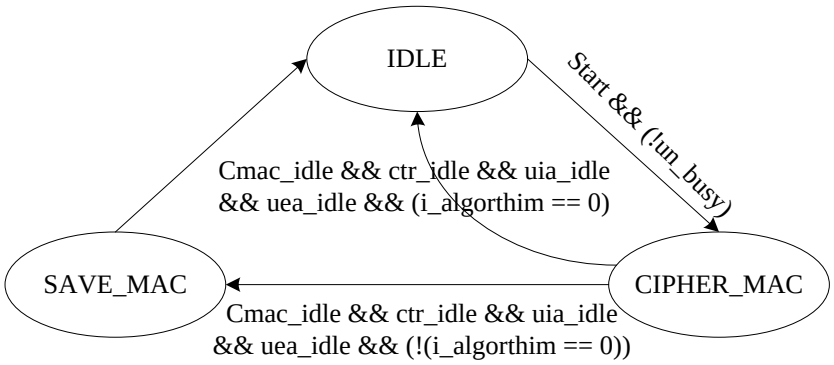

图 4.2 控制模块的实现过程状态图

系统上电复位后状态机一直处于IDLE状态，当加解密IP核启动信号start使能且CMAC完整性计算模块、CTR加密模块、UIA2完整性计算模块和UEA2加密模块都在空闲状态时，状态机进入加密和完整性计算状态即CIPHER_MAC状态。同时根据上下行方向指示标志DIRECTION判断此操作是进行加密还是解密，如果DIRECTION值为0则表示上行过程需要加密处理，否则DIRECTION值为0则表示此过程没下行过程需要解密处理。并且同时根据加密算法选择标识符e_algorhtim和完整性算法选择标识符i_algorhtim来选择相应的算法。在CIPHER_MAC状态，对数据进行加密和完整性计算。在该状态中，控制模块根据加密模块产生的密钥流和所需加密的明文进行异或运算产生密文并且同时存储到数据RAM中。如果本次操作不需要完整性计算过程，等待加密或解密过程结束信号有效后（高电平），状态机跳转到IDLE状态。否则状态机进入存储认证码状态即SAVE_MAC状态把认证码附在密文后面进行存储，下一个时钟周期，状态机进入IDLE状态，一次处理操作完成。

4.1.2 AES- CMAC 完整性计算模块

AES-CMAC完整性计算模块主要是对信令数据进行完整性保护，它是基于AES算法

来实现的，在该模块中例化了一个AES核（见4.2节）来完成对分组块的加密运算最后产生MAC码。AES-CMAC模式进行完整性计算需要对所处理的数据先进行分块，分块结构见2.4节。在本模块设计中对每次输入的128bit数据先寄存一个时钟周期，同时对其进行分块，每块128bit。第一块分块完成后开始调用AES核对分块后的数据进行加密，第一块完整性计算完成后开始对第二块进行完整性计算，如此循环直到全部数据完整性计算完成产生MAC码。该模块中定义一个计数器cmac_dscpt_len，初始值为数据长度（dscpt_len，单位为字节），每分一块数据计数器减16。该计数器用来确定数据的处理完成情况以及子密钥的选择。如图4.3所示为完整性计算实现过程状态图。

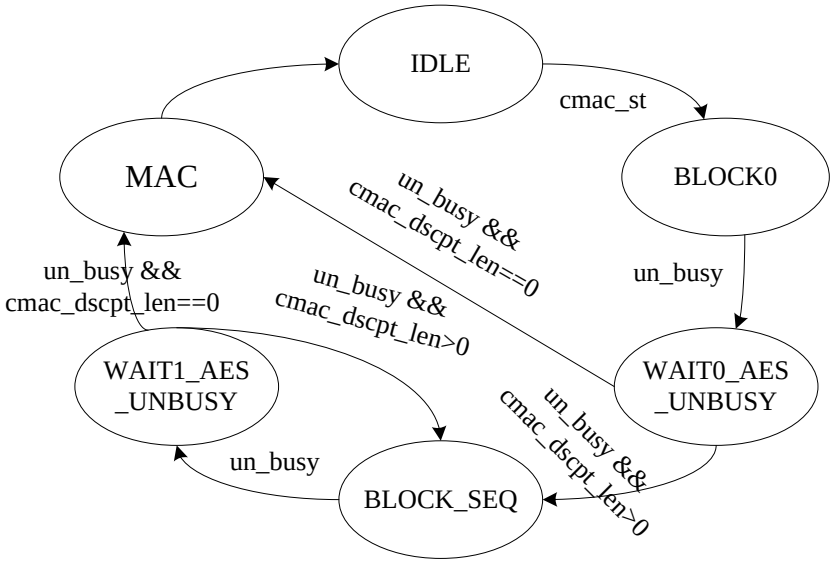

图 4.3 AES- CMAC 实现过程状态图

系统上电复位后状态机处于 IDLE 状态，当 cmac_st 信号使能时该模块开始工作，同时将描述符中的初始向量、子密钥、需要处理的数据长度（dscpt_len）存入该模块相应的寄存器中。下一个时钟到来状态机进入 BLOCK0 状态，在分块时由于第一块数据包含 64 位的初始向量值不同于其它块，所以第一块数据的分块方式也不同于其它块，该状态就是完成了第一块数据分块。在该状态中如果 AES 处于空闲状态，当下一个时钟到来时状态机进入 WAIT0_AES_UNBUSY 状态。在该状调用 AES 模块完成第一块数据加密，当 AES 模块状态标志位 un_busy 为空闲状态时表示加密完成。此时如果 cmac_dscpt_len 为零则下个时钟到来状态机跳转到 MAC 状态完成 32 位认证码的输出；如果 cmac_dscpt_len 大于零则进入 BLOCK_SEQ 状态，该状态完成剩余的数据分块。当下一个时钟到来时进入 WAIT1_AES_UNBUSY 状态。该状态调用 AES 模块完成剩余的数据加密，当 AES 模块状态标志位 un_busy 为空闲状态时表示加密完成。此时如果 cmac_dscpt_len 大于零则状态机跳转到 BLOCK_SEQ 状态继续处理剩余的数据；否则下

一个时钟系统进入 MAC 状态完成 32 位认证码的输出。等待下一个时钟到来系统进入 IDLE 状态，同时标志位 cmac_idle 信号拉高表示空闲状态。

4.1.3 AES- CTR 加密模块

CTR 加密模块主要是对用户数据进行加密，实现相对比较简单，它是基于 AES 算法来实现的。该模块主要完成初始化计数器以及对计数器 T_i 分组序列的加密运算，产生密钥流。计数器初始化模式见 2.3 节。该模块中例化了一个 AES 核来完成对计数器 T_i 分组序列的加密运算。在该模块中定义了两个计数器 count1 和 count2，count1 用来初始化计数器 T_i，count2 用来确定数据的处理完成情况。计数器 count2 初始值为数据长度，每处理一个计数器块计数器 count2 减 16。当第一个计数器块 T_1 初始化完成后开始调用 AES 对其进行加密，第一个计数器块 T_1 加密完成后开始对第二个计数器块 T_2 加密处理，如此循环直到全部计数器块 T_i 加密完成。

4.1.4 UIA2 完整性计算模块

UIA2 完整性计算模块主要是对信令数据进行完整性保护，它是基于 SNOW 3G 算法来实现的，在该模块中例化了一个 SNOW 3G 核（见 4.3 节）来产生 5 个 32bit 的密钥流，经过 EVAL 函数操作和 MUL 函数操作以及和一些异或运算最后产生 32bit 的 MAC 码。UIA2 完整性计算模块内部硬件结构如图 4.4 所示。其中初始化过程把初始密钥 key 和一系列初始参数 COUNT、BEARER 和 DIRECTION 经过 2.7.2 节的运算过程产生新的密钥 K 和初始向量 IV 做为 SNOW3G 模块的输入。EVAL 函数和 MUL 函数提前设计好加载到函数头文件中，在模块中需要使用时对其进行加载调用。如图 4.5 为 UIA2 模块实现过程状态图。

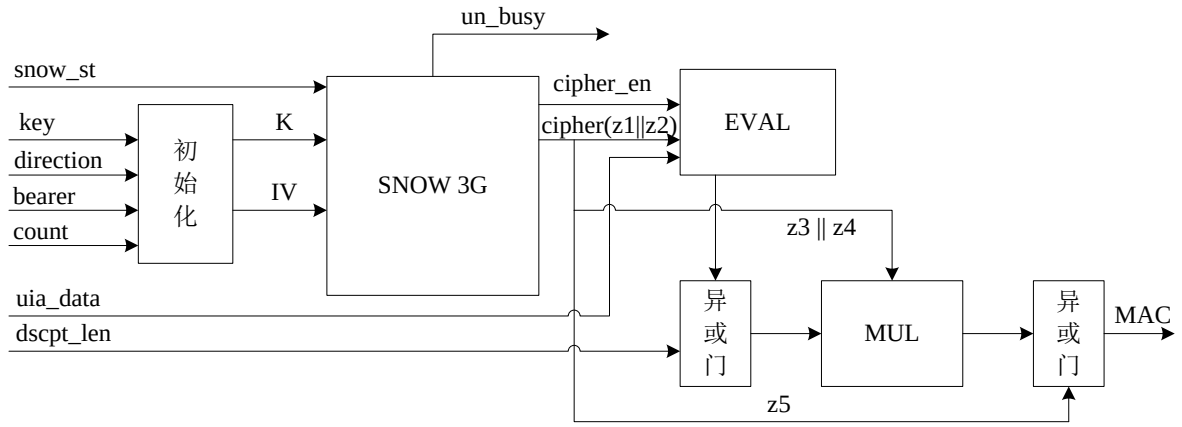

图 4.4 UIA2 完整性计算模块内部硬件结构

图 4.5 所示为 UIA2 模块实现过程状态图。系统复位后状态机处于 IDLE 状态，当启

动信号 uia_st 使能时该模块开始工作，状态机进入 SNOW_3G 状态且同时完成密钥初始化以及产生初始向量 IV。在 SNOW_3G 状态下调用 SNOW 3G 模块产生 5 个 32bit 的密钥流，之后状态机进入 EVAL 状态。此时对数据和密钥流进行 EVAL 函数操作，当数据处理状况计数器 m_cnt>=(*l*-1)（*l* 为进行完整性计算的数据按 64bit 分块后的数量）时表示 EVAL 函数操作结束。下一个时钟状态机进入 MUL 状态，同时对 EVAL 函数的计算结果和数据长度进行异或运算。在 MUL 状态将 EVAL 与数据长度的计算结果和密钥流 z3 || z4 用 MUL 函数运算。等待下一个时钟到来，状态机进入 MAC 状态。该状态下将密钥流 z5 与 MUL 函数运算结果的高 32bit 进行异或运算得到 MAC 码。下一个时钟周期状态机进入 IDLE 状态并且发出空状态信号 uia_idle 有效。

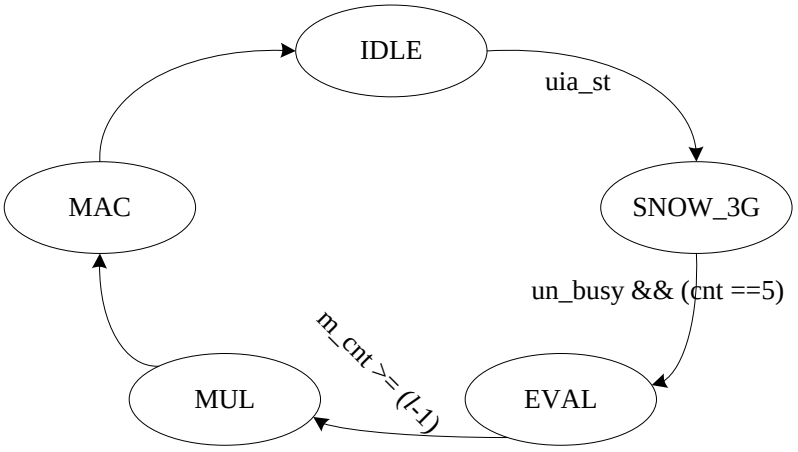

图 4.5 UIA2 模块实现过程状态图

4.1.5 UEA2 加密模块

UEA2 加密模块以 SNOW 3G 算法为基本单元，主要完成对用户数据和信令数据的加密。该模块实现起来比较简单，和 AES-CTR 模块实现过程基本相同。唯一不同处是本模块中需要定义一些异或门来完成密钥的初始化以及初始向量 IV 的产生，其中例化了一个 SNOW 3G 核，根据密钥和初始向量 IV 来产生加密所需的密钥流。

4.2 AES 核硬件设计

本设计中，AES 核在 CTR 和 CMAC 模式下实现对数据的加密和完整性保护功能。在现代的一些加密产品中，AES 算法的应用非常广泛。目前关于 AES 核的结构设计有非反馈方式和反馈方式这两种方式实现。非反馈方式是采用流水线架构，对迭代单元进行复制，将每一个迭代单元作为一级，级与级之间插入寄存器，这样在一个时钟周期中

就可以处理一组数据，极大的提高了带宽。而在反馈方式下，迭代操作将结果作为下一次迭代操作的输入，每一次迭代操作都采用同一套硬件结构[23]。非反馈方式是以较大的面积和较高的功耗换取高带宽。

本设计将在功耗与带宽之间找一个合适点，以可重用性为目标对 AES 核进行设计。因此，本设计将采用反馈方式设计一个支持数据分组和密钥长度为 128 位的可重用 AES 核。下来分别介绍 AES 核的接口、接口时序以及 AES 核的具体设计过程。

4.2.1 AES 核接口定义

表 4.1 所示为 AES 核接口信号及功能定义，为方便其他模块调用 AES 核，其接口以简单为基本原则进行定义。如表 4.1 所示 AES 核有 5 个输入，5 个输出接口。当明文输入使能接口 plain_text_en 有效时，AES 核开始工作且同时将需要加密的数据输入。当密文输出使能信号 cipher_text_en 有效时，表示此时输出的密文有效。当 AES 工作结束时，un_busy 信号拉高，表示 AES 模块处于空闲状态。

表 4.1 AES 核接口信号及功能定义

接口名称	位宽	传输方向	描述
clk	1	Input	系统时钟
rst_n	1	Input	系统复位
plain_text_en	1	Input	明文输入使能
plain_text	128	Input	明文输入接口
key	128	Input	密钥输入接口
cipher_text_en	1	Onput	密文输出使能
cipher_text	128	Onput	密文输出接口
un_busy	1	Onput	AES 核空闲标志（1 空闲，0 正在工作）
key_rd	1	Onput	读密钥使能
key_addr	4	Onput	读密钥地址

4.2.2 AES 核接口时序

当外部模块需要使用 AES 核时，在 plain_text_en 接口上输入一个时钟周期的脉冲信号，这时 AES 核启动。在 plain_text_en 接口上输入使能信号的同时通过 plain_text 接口输入需要加密的明文数据。AES 核在接收明文数据的同时就开始处理，在内部逻辑的控制下，在合适的时间通过 key_rd 接口和 key_addr 接口分别发出密钥读使能信号和密钥地址，外部模块则在下一个时钟周期将相应的密钥通过 key 接口返回。AES 核运算结束之后，在 cipher_text 接口将产生的密文输出，同时将 un_busy 信号拉高。如图 4.6 所示为

AES 核接口时序图。

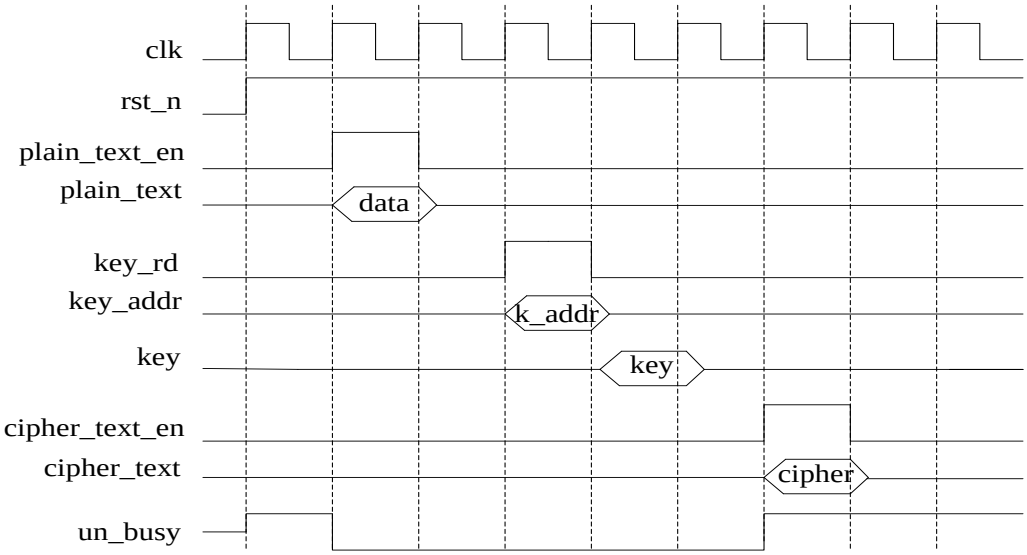

图 4.6 AES 核接口时序图

4.2.3 AES 核内部结构设计

AES 核主要由字节代换、行移位、列混合和轮密钥加这四部分组成。由于字节代换操作中与字节的位置无关,所以在算法实现中行移位操作和字节代换操作可以调换顺序。在迭代操作中行移位操作是将状态矩阵中从倒数第三行开始将每一行的字节进行循环左移。字节代操作是将输入的字节代换成一个新字节。列混合操作是将状态矩阵中每一列经过定义的算法运算。轮密钥加操作是将状态矩阵中的每一列和轮密钥进行按位异或运算。

在本设计中，由于字节代换操作数据通路的关键路径较长，采取对电路结构进行适当的分级处理并加入寄存器来满足时序要求，将使其在两个时钟周期中完成，剩下的行移位、列混淆、轮密钥加在一个时钟周期中完成。其中行移位操作和轮密钥加操作的实现非常简单，本设计在数据输入 AES 核的过程中就完成了状态矩阵的行移位功能，这种方式不仅节约了资源开销，而且还提高了 AES 核的速度。对行移位操作和轮密钥加操作的具体实现过程本处不做介绍，重点介绍字节代换操作和列混合操作的硬件实现。

（1）字节代换硬件设计

字节代换是 AES 算法中唯一的非线性变换,加密时每个字节根据 S 盒做替换运算产生一个新的字节。它是由一个有限域$GF(2^8)$上的乘法求逆和一个线性仿射变换构造的。字节代换的硬件实现一般由基于 ROM 的查找表方式和基于有限域降阶的组合逻辑方式这两种方法来实现。在早期的 AES 硬件实现基本上都使用基于 ROM 的查找表来实现字节代换，如文献[24]采用查找表的方法实现 S 盒。但是不仅在 AES 算法内部迭代过程中

需要字节代换操作，而且在密钥扩展中也需要字节代换操作。假如使用查找表的方式实现，整个资源开销相当大，无法满足高速低复杂度的 AES 核的实现，因此逐渐的被舍弃。文献[25-27]通过有限域的降阶来实现乘法求逆运算的思想，提出了一些用组合逻辑来实现字节代换操作的方法。这种实现方法不仅节约了资源而且还可以将流水线的设计思想应用其中，因此被普遍采用。本设计将根据这种思想把$GF(2^8)$域上的乘法求逆转化为复合域$GF((2^4)^2)$上的求逆运算和一些其他运算，采用组合逻辑实现字节代换运算。

由 2.2.1 节所知，有限域$GF(2^8)$和复合域$GF((2^4)^2)$是同构的，因此$GF(2^8)$求逆等效于$GF((2^4)^2)$上求逆，然而在这两个域内运算的复杂度却有很大的差异。复合域$GF((2^4)^2)$的模多项式为：

$$\mathrm{G}(x)=x^2+x+\lambda \tag{4.1}$$

有限域$GF(2^8)$和复合域$GF((2^4)^2)$中元素之间的同构映射为：

$$[a_l,a_h]^T=Ta^T \quad a\in GF(2^8),a_h,a_l\in GF(2^4),a_hx+a_l\in GF((2^4)^2) \tag{4.2}$$

式中 T 表示一个 8×8 的二进制矩阵，a 表示字节$\{b_7,b_6,b_5,b_4,b_3,b_2,b_1,b_0\}_2$。

同构映射变换是一个矩阵乘法运算。电路的复杂度取决于同构映射变换的复杂度，T 中非 0 项的个数越少，复杂度越高。因此选择一个系数 λ 使得矩阵 T 和 T^{-1} 中的非 0 项尽量少。文献[25]中提供了一种搜索算法，当式（4.1）中$\lambda=\{1110\}_2$时，同构映射矩阵及同构映射逆矩阵为：

$$\mathrm{T}=\begin{bmatrix}1&0&0&0&1&1&1&0\\0&1&1&0&0&0&0&0\\0&1&0&0&0&0&0&1\\0&0&1&0&1&0&0&0\\0&0&0&0&1&1&1&0\\0&1&0&0&1&0&1&1\\0&0&1&1&0&1&0&1\\0&0&0&0&0&1&0&1\end{bmatrix}\quad \mathrm{T}^{-1}=\begin{bmatrix}1&0&0&0&1&0&0&0\\0&0&0&0&1&1&0&1\\0&1&0&0&1&1&0&1\\0&1&0&0&1&1&1&0\\0&1&0&1&1&1&0&1\\0&0&1&0&1&1&0&0\\0&1&1&1&1&0&0&1\\0&0&1&0&1&1&0&1\end{bmatrix} \tag{4.3}$$

定义$GF((2^4)^2)$中的元素$\{a_h,a_l\}$的乘法逆元为$\{a_h',a_l'\}$，则有：

$$(a_hx+a_l)\otimes(a_h^{'}x+a_l^{'})=\{0\}x+1 \tag{4.4}$$

$$(a_hx+a_l)\otimes(a_h^{'}x+a_l^{'})=(a_hx+a_l)\bullet(a_h^{'}x+a_l^{'})\bmod(x^2+x+\lambda) \tag{4.5}$$

解以上两个方程，得：

$$\begin{aligned}a_h^{'}x+a_l^{'}&=(a_h\bullet\theta)x+(a_h\oplus a_l)\bullet\theta\\ \theta&=((a_h^2\bullet\lambda)\oplus(a_h\bullet a_l)\oplus a_l^2)^{-1}\end{aligned} \tag{4.6}$$

因此复合域$GF((2^4)^2)$中的求逆运算只是涉及到有限域$GF(2^4)$上的求逆运算和有

限域$GF(2^4)$上的平方、乘法、乘常数以及异或运算组成的。有限域$GF(2^4)$中的求逆过程可用一个较小的硬件电路来实现。图 4.7 所示为字节代换电路结构。

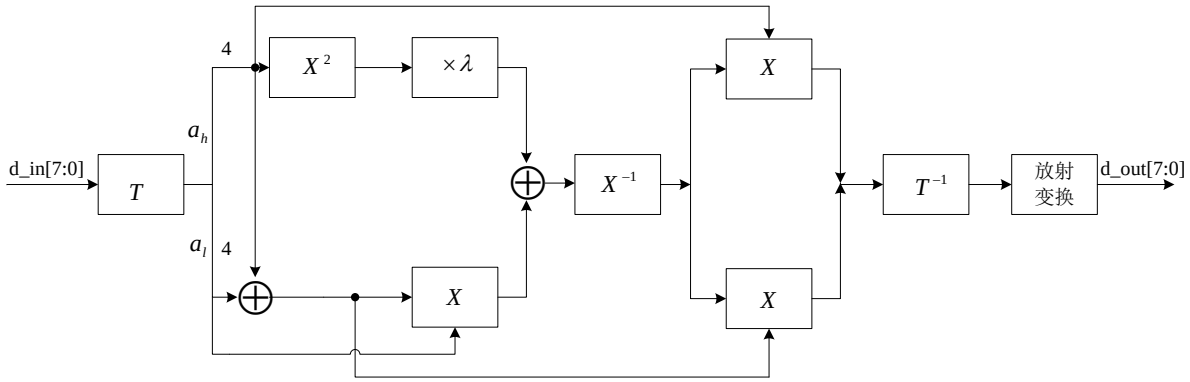

图 4.7 字节代换电路结构

其中X^2为有限域$GF(2^4)$上的平方运算，$\times\lambda$为有限域$GF(2^4)$上的常数乘法，T 为同构映射，T^{-1}为同构逆映射，X 为有限域$GF(2^4)$上的乘法，X^{-1}为有限域$GF(2^4)$上的乘法求逆运算。下来介绍本处所涉及的运算及优化实现。以下运算为$GF(2^4)$上的运算。

① $GF(2^4)$上的乘法运算

有限域$GF(2^4)$上的模多项式为：

$$Q(x)=x^4+x+1 \tag{4.7}$$

根据等式（2.4）同样将有限域$GF(2^4)$上的乘法定义为：

$$c(x)=a(x)\bullet b(x)=(a(x)\times b(x))\bmod(Q(x)) \tag{4.8}$$

文献[28]中提出了一种高效的并行乘法结构，对等式（4.8）用矩阵表示为：

$$\begin{bmatrix}c_0\\c_1\\c_2\\c_3\end{bmatrix}=ZB=\begin{bmatrix}a_0 & a_3 & a_2 & a_1\\ a_1 & a_0+a_3 & a_2+a_3 & a_1+a_2\\ a_2 & a_1 & a_0+a_3 & a_2+a_3\\ a_3 & a_2 & a_1 & a_0+a_3\end{bmatrix}\begin{bmatrix}b_0\\b_1\\b_2\\b_3\end{bmatrix} \tag{4.9}$$

对式（4.9）化简后得：

$$\begin{aligned}&a_A=a_0\oplus a_3,\quad a_B=a_2\oplus a_3\\ &c_0=a_0b_0\oplus a_3b_1\oplus a_2b_2\oplus a_1b_3,\quad c_1=a_1b_0\oplus a_Ab_1\oplus a_Bb_2\oplus(a_1\oplus a_2)b_3\\ &c_2=a_2b_0\oplus a_1b_1\oplus a_Ab_2\oplus a_Bb_3,\quad c_3=a_3b_0\oplus a_2b_1\oplus a_1b_2\oplus a_Ab_3\end{aligned} \tag{4.10}$$

② $GF(2^4)$上的平方运算

$GF(2^4)$上的平方运算可以看做是乘法运算的特殊情况，将$c(x)=a(x)^2\bmod(Q(x))$代入式（4.10），化简得：

$$c_0 = a_0 \oplus a_2,\quad c_1 = a_2,\quad c_2 = a_1 \oplus a_3,\quad c_3 = a_3 \tag{4.11}$$

③ $GF(2^4)$上的乘常数运算

当$\lambda = \{1110\}_2$时，同样把$c(x) = (a(x) \times \{1110\}) \bmod (Q(x))$带入式（4.10），化简得：

$$\begin{aligned} &c_0 = a_1 \oplus a_2 \oplus a_3,\quad c_1 = a_0 \oplus a_1, \\ &c_2 = a_0 \oplus a_1 \oplus a_2,\quad c_3 = a_0 \oplus a_1 \oplus a_2 \oplus a_3 \end{aligned} \tag{4.12}$$

④ $GF(2^4)$上的乘法逆运算

把有限域$GF(2^4)$中的元素 a 的乘法逆元记为a^{-1}，其中$a^{-1} = [a_0', a_1', a_2', a_3']$，则有

$$a(x)^{-1} \bullet a(x) = (a(x)^{-1} \times a(x)) \bmod (Q(x)) = 1 \tag{4.13}$$

由式（4.9）知，存在矩阵Z^{-1}，使

$$Z^{-1}[a_0, a_1, a_2, a_3]^T = [1,0,0,0]^T \tag{4.14}$$

其中Z^{-1}是由$a^{-1} = [a_0', a_1', a_2', a_3']$的系数推导出，并且$Z^{-1}$的第一列正是$a^{-1}$的系数，

因此求出 Z 的逆矩阵就可以得到a^{-1}的系数，即：$a_i^{'} = adj_{i,0}(Z) / \det(Z)$ （4.15）

由$\det(Z) \equiv 1$，所以，$a_i^{'} = adj_{i,0}(Z)$ （4.16）

由式（4-9）推导伴随矩阵并且利用等式$(a \oplus b \oplus a \bullet b) = (a + b)$化简可得$a^{-1}$的系数表达式为：

$$\begin{aligned} &a_A = a_0 \oplus a_1,\quad a_B = a_2 \oplus a_3 \\ &a_0' = a_A \oplus a_B \oplus a_1 \bullet a_2 \bullet a_3 \oplus a_2 \bullet (a_0 + a_1),\quad a_1' = a_A \bullet a_2 \oplus a_3 \oplus a_1 \bullet (a_0 + a_3) \\ &a_2' = a_0 \bullet a_1 \oplus a_B \oplus a_0 \bullet (a_2 + a_3),\qquad a_3' = a_1 \oplus a_B \oplus a_0 \bullet a_3 \oplus a_3 \bullet (a_1 + a_2) \end{aligned} \tag{4.17}$$

其中，+表示逻辑或运算。

⑤ 同构映射与逆映射

将式（4.3）中的 T 带入式（4.2）中化简后，得到的同构映射变换为：

$$\begin{aligned} &[a_l, a_h]^T = Ta^T \quad a \in GF(2^8),\quad a_h, a_l \in GF(2^4),\quad a_h x + a_l \in GF((2^4)^2) \\ &a_A = a_1 \oplus a_7,\quad a_B = a_5 \oplus a_7,\quad a_C = a_4 \oplus a_6 \\ &a_{l0} = a_C \oplus a_0 \oplus a_5,\quad a_{l1} = a_1 \oplus a_2,\quad a_{l2} = a_A,\quad a_{l3} = a_2 \oplus a_4 \\ &a_{h0} = a_C \oplus a_5,\quad a_{h1} = a_A \oplus a_C,\quad a_{h2} = a_B \oplus a_2 \oplus a_3,\quad a_{h3} = a_B \end{aligned} \tag{4.18}$$

同理可得同构逆映射为：

$$\begin{aligned} &a^T = T^{-1}[a_l, a_h]^T \quad a \in GF(2^8),\quad a_h, a_l \in GF(2^4),\quad a_h x + a_l \in GF((2^4)^2) \\ &a_A = a_{l1} \oplus a_{h3},\quad a_B = a_{h0} \oplus a_{h1} \\ &a_0 = a_{h0} \oplus a_{l0},\quad a_1 = a_B \oplus a_{h3},\quad a_2 = a_A \oplus a_B,\quad a_3 = a_B \oplus a_{h2} \oplus a_{l1} \\ &a_4 = a_A \oplus a_B \oplus a_{l3},\quad a_5 = a_B \oplus a_{l2},\quad a_6 = a_A \oplus a_{h0} \oplus a_{l2} \oplus a_{l3},\quad a_7 = a_B \oplus a_{h3} \oplus a_{l2} \end{aligned} \tag{4.19}$$

⑥ 仿射变换

化简式（2.12）所描述的仿射变换，利用与逻辑“0”异或不改变原来的值，与逻辑“1”异或意味着逻辑非的原理，得到放射变换为：

$$\begin{aligned}
&b = affine(a)\\
&a_A = a_0 \oplus a_1,\quad a_B = a_2 \oplus a_3,\quad a_C = a_4 \oplus a_5,\quad a_D = a_6 \oplus a_7\\
&b_0 = \bar{a}_0 \oplus a_C \oplus a_D,\quad b_1 = \bar{a}_5 \oplus a_A \oplus a_D,\quad b_2 = a_2 \oplus a_A \oplus a_D,\\
&b_3 = a_7 \oplus a_A \oplus a_B,\quad b_4 = a_4 \oplus a_A \oplus a_B,\quad b_5 = \bar{a}_1 \oplus a_B \oplus a_C\\
&b_6 = \bar{a}_6 \oplus a_B \oplus a_C,\quad b_7 = a_3 \oplus a_C \oplus a_D
\end{aligned} \tag{4.20}$$

（2）列混合硬件实现

在列混合硬件实现过程中，文献[29]中提出了一种积生成器，可以完成输入的字节与式（2.14）中所有系数的乘积，用这样四个并行的积生成器就可以组合出需要的列混合结果。但是该积生成器没有对资源进行充分利用。本文为了使硬件资源充分共享，使用了一种更经济的方法，采用因子分解来实现列混合操作。把等式（2.14）分解，使用矩阵形式表示可得：

$$\begin{bmatrix} s'_{0,c}\\ s'_{1,c}\\ s'_{2,c}\\ s'_{3,c}\end{bmatrix} = \begin{bmatrix} 02 & 02 & 00 & 00\\ 00 & 02 & 02 & 00\\ 00 & 00 & 02 & 02\\ 02 & 00 & 00 & 02\end{bmatrix}\begin{bmatrix} s_{0,c}\\ s_{1,c}\\ s_{2,c}\\ s_{3,c}\end{bmatrix} + \begin{bmatrix} 00 & 01 & 01 & 01\\ 01 & 00 & 01 & 01\\ 01 & 01 & 00 & 01\\ 01 & 01 & 01 & 00\end{bmatrix}\begin{bmatrix} s_{0,c}\\ s_{1,c}\\ s_{2,c}\\ s_{3,c}\end{bmatrix} \tag{4.21}$$

这样每列中的元素经过变换后得到：

$$\begin{aligned}
s'_{0,c} &= (\{02\} \bullet (s_{0,c} \oplus s_{1,c})) \oplus (s_{2,c} \oplus s_{3,c}) \oplus s_{1,c}\\
s'_{1,c} &= (\{02\} \bullet (s_{1,c} \oplus s_{2,c})) \oplus (s_{3,c} \oplus s_{0,c}) \oplus s_{2,c}\\
s'_{2,c} &= (\{02\} \bullet (s_{2,c} \oplus s_{3,c})) \oplus (s_{0,c} \oplus s_{1,c}) \oplus s_{3,c}\\
s'_{3,c} &= (\{02\} \bullet (s_{3,c} \oplus s_{0,c})) \oplus (s_{1,c} \oplus s_{2,c}) \oplus s_{0,c}
\end{aligned} \tag{4.22}$$

可以看出经过变换后的列混合操作为异或运算和乘$\{02\}_{16}$操作了。图 4.8 为列混合操作的硬件结构。乘$\{02\}_{16}$可以用式（2.5）描述的 xtime()操作来完成，xtime()操作的电路结构如图 4.9 所示。xtime()操作用了 3 个异或门和 8 个移位寄存器。

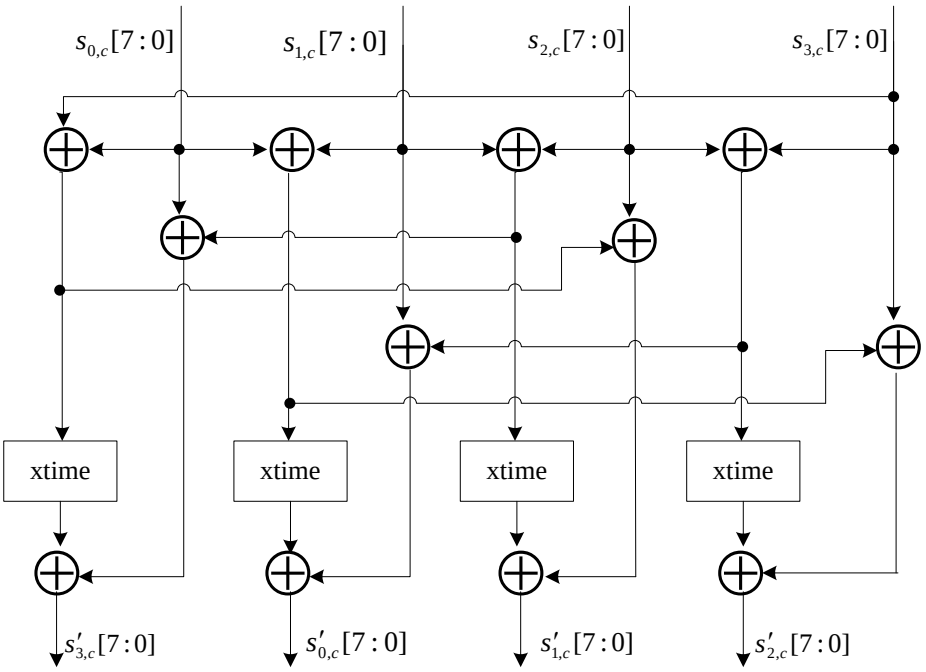

图 4.8 列混合操作硬件结构

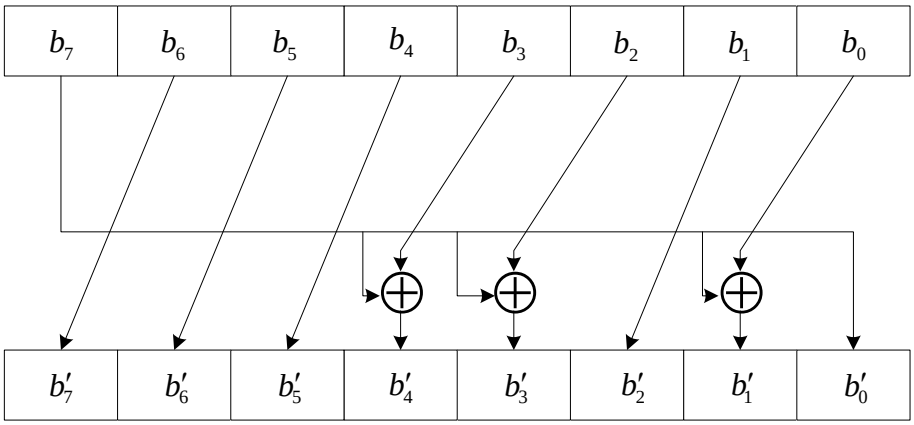

图 4.9 xtime()操作硬件结构

4.2.4 AES 核实现过程

AES 核实现过程状态图如图 4.10 所示。系统复位后状态机一直处于 IDLE 状态，当 plain_text_en 使能时 AES 核开始工作完成对明文的寄存并在下一个时钟周期跳转到 ADDKEY 状态。该状态下完成开始阶段的轮密钥加操作接着进行字节代换，本设计中字节代换需要两个时钟周期来完成，分别为 SUB_BYTEF 和 SUB_BYTES 状态，在 SUB_BYTEF 状态还完成计数器 round_cnt 的清零工作。完成字节代换操作后状态机进入

SH_MIX_ADD 状态，完成行移位、列混淆、轮密钥加操作。在此状态下判断轮计数器 round_cnt，如果 round_cnt<9 则跳转到 SUB_BYTEF 状态继续工作，否则输出密文同时跳转到 IDLE 状态结束操作。

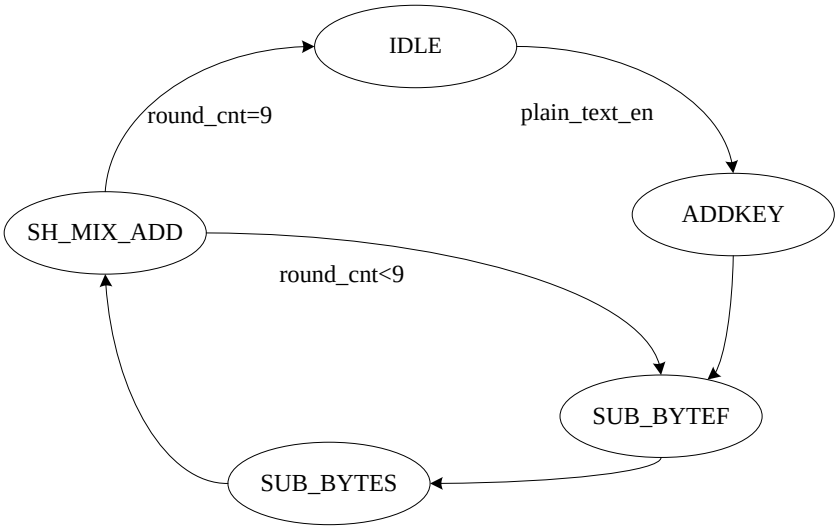

图 4.10 AES 核实现过程状态图

4.3 SNOW 3G 核硬件设计

SNOW 3G 核在 UIA2 完整性保护模块和 UEA2 加密模块的调用下产生一系列密钥流来实现对数据的加密和完整性保护功能。相对于 AES 算法的硬件实现，国内外 SNOW 3G 算法的硬件实现非常少。文献[30]提出了一种高性能 ASIC 的实现方法实现 SNOW 3G 算法 SNOW 3G 算法，在这里 S-Box S_1 和 S-Box S_2 这两个函数分别所使用的两个 S 盒 S_R 和 S_Q 及函数 $MUL\alpha$ 和 $DIV\alpha$ 的实现都是采用查找表的方式来实现。本文主要采用流水线设计结构对 SNOW 3G 核进行设计。下来分别介绍 SNOW 3G 核的接口、接口时序以及 SNOW 3G 核的硬件设计过程。

4.3.1 SNOW 3G 核接口定义

表 4.2 所示为 SNOW 3G 核接口信号及功能定义，为方便其他模块调用 SNOW 3G 核，其接口以简单为基本原则进行定义。如表 4.2 所示 SNOW 3G 核有 6 个输入，3 个输出接口。当启动信号 snow_st 有效时，SNOW 3G 核开始工作且在 snow_st 信号有效同时将初始密钥、初始向量以及所需密钥流长度信息分别通过 key 接口、iv 接口和 dscpt_len 接口输入。当密钥流输出使能信号 cipher_en 有效时，表示此时输出的密文有效。当 SNOW 3G 工作结束时，un_busy 信号拉高，表示该模块处于空闲状态。

表 4.2 SNOW 3G 核接口信号及功能定义

接口名称	位宽	传输方向	描述
clk	1	Input	系统时钟
rst_n	1	Input	系统复位
snow_st	1	Input	SNOW 3G 核工作启动接口
key	128	Input	密钥输入接口
iv	128	Input	初始向量输入接口
dscpt_len	13	Input	所需密钥流长度
cipher_en	1	Onput	密钥流输出使能
cipher_str	32	Onput	密钥流输出接口
un_busy	1	Onput	SNOW 3G 核空闲标志（1 表示空闲，0 表示正在工作）

4.3.2 SNOW 3G 核接口时序

当外部模块需要使用 SNOW 3G 核时，在 snow_st 接口上输入一个时钟周期的脉冲信号，这时 SNOW 3G 核启动。在启动信号 snow_st 有效的同时通过 key、iv 和 dscpt_len 接口输入需要产生密钥流所需的初始密钥、初始向量和所需密钥流长度信息。等待 SNOW 3G 核运算结束后，在内部逻辑的控制下，通过 cipher_en 接口和 cipher_str 接口分别输出密钥流使能信号及密钥流。同时在 un_busy 接口产生一个脉冲表示运算结束。如图 4.11 所示为 SNOW 3G 核接口时序图。

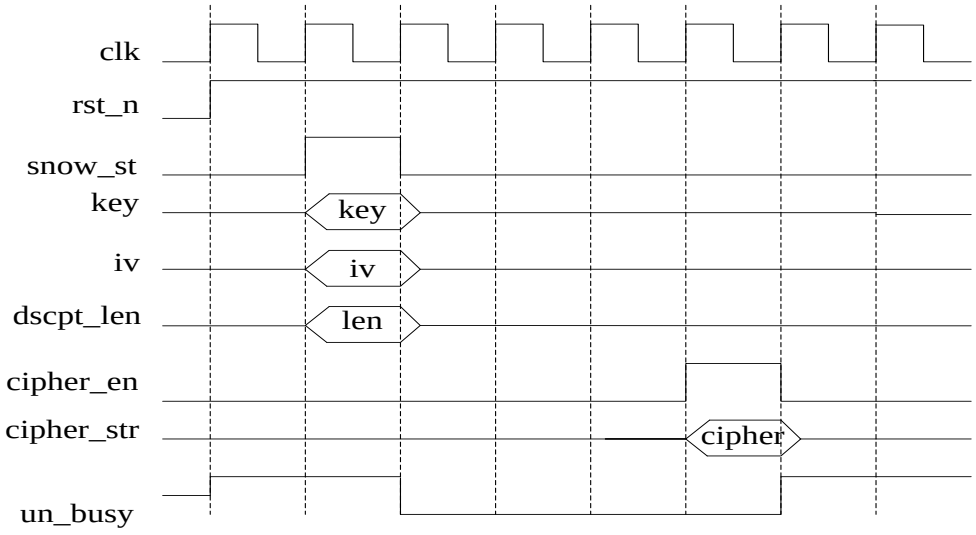

图 4.11 SNOW 3G 核接口时序图

4.3.3 SNOW 3G 算法硬件设计

SNOW 3G 算法硬件实现结构如图 4.12 所示。它主要由三部分组成，分别是线性反馈移位寄存器（LFSR）、有限状态机（FSM）和逻辑反馈。

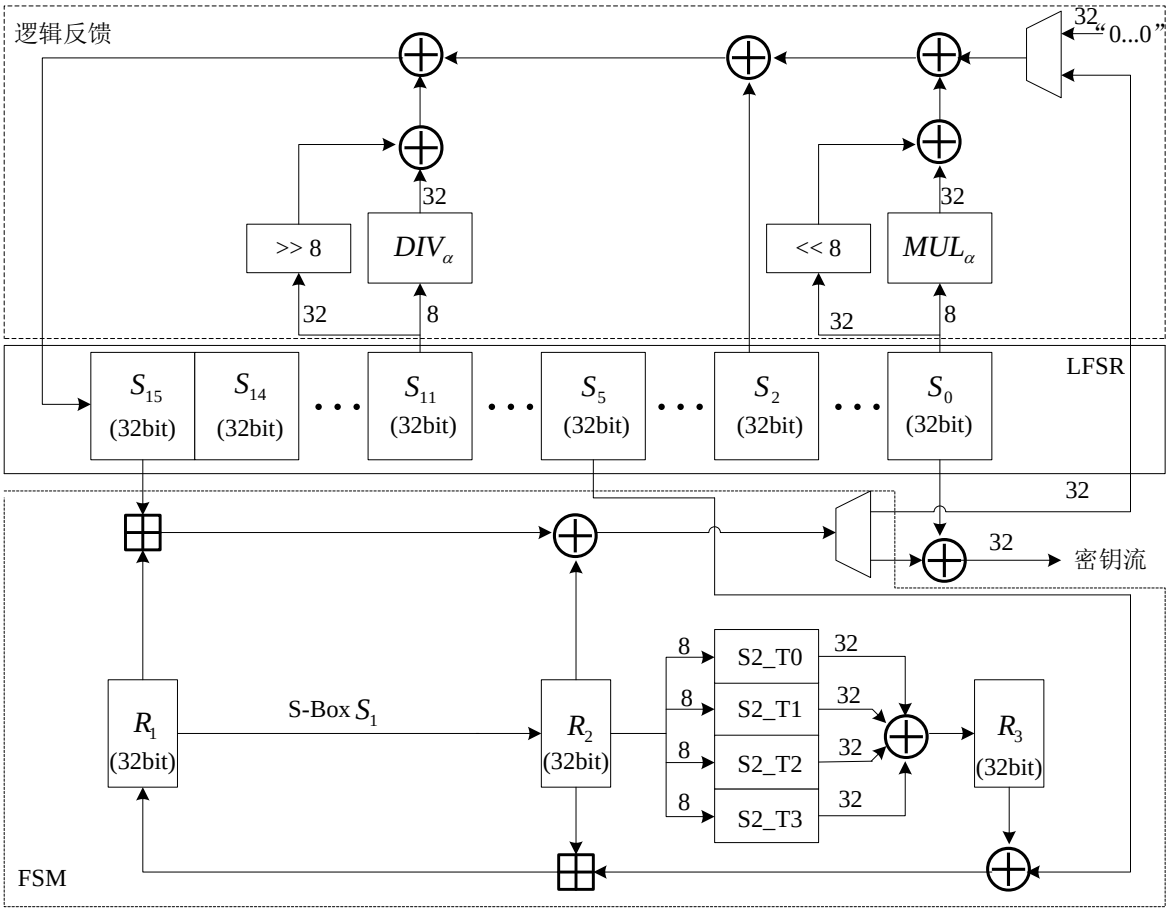

图 4.12 SNOW 3G 硬件实现结构图

LFSR 包含 16 个 32 位的寄存器 S_0 到 S_{15}。其中寄存器 S_0、S_2 和 S_{11} 用来进行逻辑反馈操作，在加密开始时要对 FLSR 中的状态寄存器进行初始化运算，此处采用 12 个异或门来对状态寄存器进行初始化。之后将初始化后的状态寄存器 S_5 和 S_{15} 的值作为 FSM 的输入。

在逻辑反馈部分，除了五个异或操作和两个移位操作外还有两个操作函数 *MULα* 和 *DIVα*，它们分别将 S_0 的高 8 位和 S_{11} 的低 8 位映射为 32 位的输出。每个时钟周期逻辑反馈部分会输出一个 32bit 的新值存储到寄存器 S_{15} 中。函数 *MULα* 和 *DIVα* 是基于有限域上的算法，本设计中通过查找表的方式来实现这两个函数，其表分别为 MULalpha 和 DIValpha，见文献[3]。基于查找表的优化实现方案和所使用的综合工具以及使用的库有关，因此功耗的大小也依赖所使用的综合库。本设计采用 xilinx 公司的 ISE13.1 综合工

具，可以运行一个高复杂的优化过程使产生的电路体积小，功耗低。

在 FSM 中，三个 32 位的寄存器 R_1，R_2 和 R_3 通过 S 盒替换连接着。其中，S-BOX S_1 是基于高级加密标准的 S-BOX S_R，然而 S_2 是用 S-BOX S_Q。此外该部分还用到两个异或操作和两个全加操作。其中，S_1 和 S_2 这两个操作函数是 FSM 设计的核心，然而 S_R 和 S_Q 的设计又是 S_1 和 S_2 设计的核心。这两个函数中所使用的 S_R 和 S_Q 盒保证了 SNOW 3G 加密算法的非线性。由于 S_R 盒和高级加密标准的 S-BOX 相同，所以此处 S_R 盒的实现方式和高级加密标准的 S-BOX 实现方式相同，采用组合逻辑实现 S_R 盒。对于 S_2 的实现，本设计中定义了四个查找表 $S_2\_T_0$, $S_2\_T_1$,$S_2\_T_2$ 和 $S_2\_T_3$ 来实现函数 S_2，每个表把 8bit 数映射为 32bit 数，$S_2\_T_0$, $S_2\_T_1$,$S_2\_T_2$ 和 $S_2\_T_3$ 的具体参数见文献[3]。其中定义 S_2 的 32bit 输入为 $w = w_0 \| w_1 \| w_2 \| w_3$（$w_0$ 为高有效字节，w_3 为低有效字节），根据 $S_2(w) = S_2_T_0(w_3) \oplus S_2_T_1(w_2) \oplus S_2_T_2\ (w1) \oplus S_2_T_3(w_0)$计算出 S_2 的结果。该处查找表的实现方式和逻辑反馈部分查找表的实现方式相同。

4.3.4 SNOW 3G 核实现过程

SNOW 3G 算法包含两个操作模式，初始化模式和密钥流产生模式。初始化模式中 FSM 和 LFSR 进行 32 次初始化运算，初始化过程中逻辑反馈模块把 FSM 的每次输出作为输入进行计算。之后进入密钥流产生模式，在密钥流产生模式中每个时钟通过 FSM 的输出和 LFSR 中的寄存器 S_0 进行异或运算产生一个 32bit 的密钥流，然而该模式中 FSM 的第一次输出要被丢弃。图 4.13 所示为 SNOW 3G 核实现过程状态图。

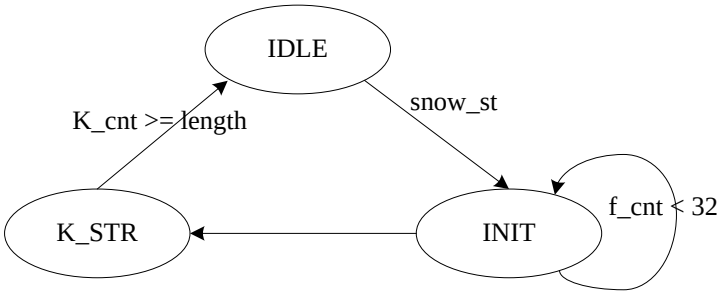

图 4.13 SNOW 3G 核实现过程状态图

系统上电复位后状态机一直处于 IDLE 状态，当 SNOW 3G 核启动信号 snow_st 使能时 SNOW 3G 核开始工作，并且同时从 key 接口和 iv 接口将密钥和初始向量输入。snow_st 使能后状态机进入 INIT 状态，SNOW 3G 核在该状态下完成 32 次初始化运算。当初始化计数器 f_cnt>=32 时，下一个时钟周期状态机进入 K_STR 状态。在该状态实现密钥流产生过程，在 K_STR 状态下，每个时钟周期产生一个 32 位的密钥流输出。直到密钥流计数器 k_cnt 大于或者等于所需密钥流长度 dscpt_len 的值时，表示密钥流产生结束。这

时状态机进入 IDLE 状态，表示操作结束。同时空闲标志信号 un_busy 拉高表示 SNOW 3G 核出于空闲状态。

4.4 本章小结

本章采用自顶向下的设计方式，首先对加解密 IP 核的总体结构框架进行设计，接着对控制模块、CMAC 完整性计算模块、CTR 加密模块、UIA2 完整性计算模块和 UEA2 加密模块进行设计，最后两节单独介绍可重用 AES 核和 SNOW 3G 核的硬件实现方式。

5 仿真分析与验证

在 RTL 代码设计完成之后，需要对设计结果进行验证，以保证设计对象达到产品功能规范中所定义的功能要求。在深亚微米工艺下，随着设计复杂度的迅猛增长，据统计对芯片进行系统的验证占到整个设计研发工作总量的 50%~70%。因此，实现高效的 SOC 功能验证可提高设计效率。

目前，工业界现有的验证技术很多，比较常用的验证技术有逻辑功能仿真、静态时序验证、FPGA 验证和软硬件协同验证等。基本上都采用自顶向下和自底向上的验证策略。本设计中采用了自底向上的验证方法，在设计阶段对各个模块进行了的功能验证，然后采用静态时序分析方法对设计进行时序验证，最后用 FPGA 验证的方法，对设计做了进一步的时序和功能验证。

5.1 逻辑功能验证

逻辑功能验证主要用于在综合之前，对设计的 RTL 代码验证其电路在逻辑功能上是否正确，控制上有没有混乱等。在设计中使设计与验证同时进行，这样在仿真过程中能及时发现设计中的错误，加快了设计进度，提高了设计的可靠性。

本设计中采用 Model 公司的 ModelSim6.2b 仿真工具来对设计进行逻辑功能仿真，通过比对待测模块的输出值和期望输出值来确定模块功能的正确性。下来对设计的各个子模块进行逻辑功能仿真。

5.1.1 子模块逻辑功能仿真

（1）AES 核功能仿真结果

对于 AES 核的逻辑功能仿真，仿真数据采用高级加密标准即文献[2]中所提供的测试数据。仿真使用的其中一组测试数据如下：

明文：$\{00112233445566778899aabbccddeeff\}_{16}$；

密钥：$\{000102030405060708090a0b0c0d0e0f\}_{16}$；

密文：$\{69c4e0d86a7b0430d8cdb78070b4c55a\}_{16}$；

图 5.1 为 AES 核功能仿真波形，其中，aes_test.v 为该模块的测试文件。由图 5.1 可以看出当密钥输出使能信号 cipher_text_en 有效时，密文输出接口 cipher_text 的输出值为 $\{69c4e0d86a7b0430d8cdb78070b4c55a\}_{16}$。其结果和高级加密标准中提供的测试结果相同，表示仿真结果正确。同时，使用其它几组测试数据进行仿真，测试结果及接口时序和预期相同，由此可以证明该硬核是正常工作的，在功能上达到设计要求。

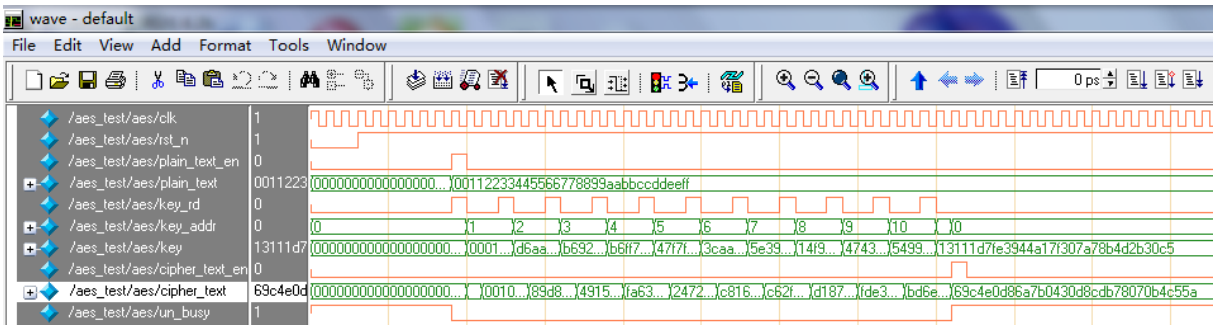

图 5.1 AES 核功能仿真波形

（2）SNOW 3G 核功能仿真结果

对于 SNOW 3G 核的功能仿真，仿真数据采用文献[32]中所提供的测试数据。仿真使用的其中一组测试数据如下：

密钥 K：$\{2bd6459f82c5b300952c49104881ff48\}_{16}$；

初始向量 IV：$\{ea024714ad5c4d84df1f9b251c0bf45f\}_{16}$；

输出密钥流长度：64bit； 密钥流：Z_1 ：$\{abee9704\}_{16}$；Z_2 ：$\{7ac31373\}_{16}$。

图 5.2 为 SNOW 3G 核功能仿真波形，其中，test_snow3g.v 为该模块的测试文件。由图 5.2 可以看出当密钥流输出使能信号 cipher_en 有效时，密钥流输出接口 cipher_str 的输出值为$\{abee9704\}_{16}$ 和$\{7ac31373\}_{16}$。其结果和文献[32]中提供的测试结果相同，表示仿真结果正确。同时，使用其它几组测试数据采用相同的方式进行仿真，测试结果及接口时序和预期相同，由此可以证明该硬核是正常工作的，在功能上达到设计要求。

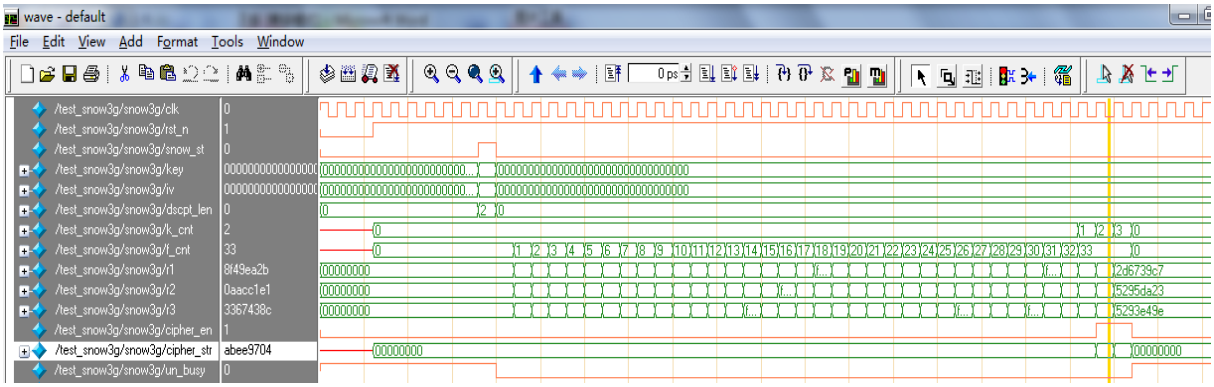

图 5.2 SNOW 3G 核功能仿真波形

（3）AES-CMAC 模块功能仿真结果

对于 AES-CMAC 模块的功能仿真，仿真数据采用 LTE/SAE 协议即文献[4]中所提供的测试数据。仿真使用的其中一组测试数据如下：

COUNT：$\{398a59b4\}_{16}$； BEARER：$\{1a\}_{16}$； DIRECTION：1；

子密钥 K1：{36e3e532 26522ba6 c0a4236b cbbf0ce3}$_{16}$；

子密钥 K2：{6dc7ca64 4ca4574d 814846d7 977e19c6}$_{16}$；

密钥 K：{d3c5d592327fb11c4035c6680af8c6d1}$_{16}$； 数据长度：64bit；

数据：{484583d5afe082ae }$_{16}$； MAC：{b93787e6}$_{16}$；

图 5.3 为 AES-CMAC 模块功能仿真波形，其中，test_cmac.v 为该模块的测试文件。图 5.3 中 m_block 为数据分块后的数据块，当 plain_text_en 使能时，AES 核开始工作。由图 5.3 中可以看出当认证码输出使能信号 mac_en 有效时，认证码输出接口 mac 的输出值为{b93787e6}$_{16}$。其结果和文献[4]中提供的测试结果相同，表示仿真结果正确。同时，使用其它几组测试数据采用相同的方式进行仿真，测试结果及接口时序和预期相同，由此可以证明该模块是正常工作的，在功能上达到设计要求。

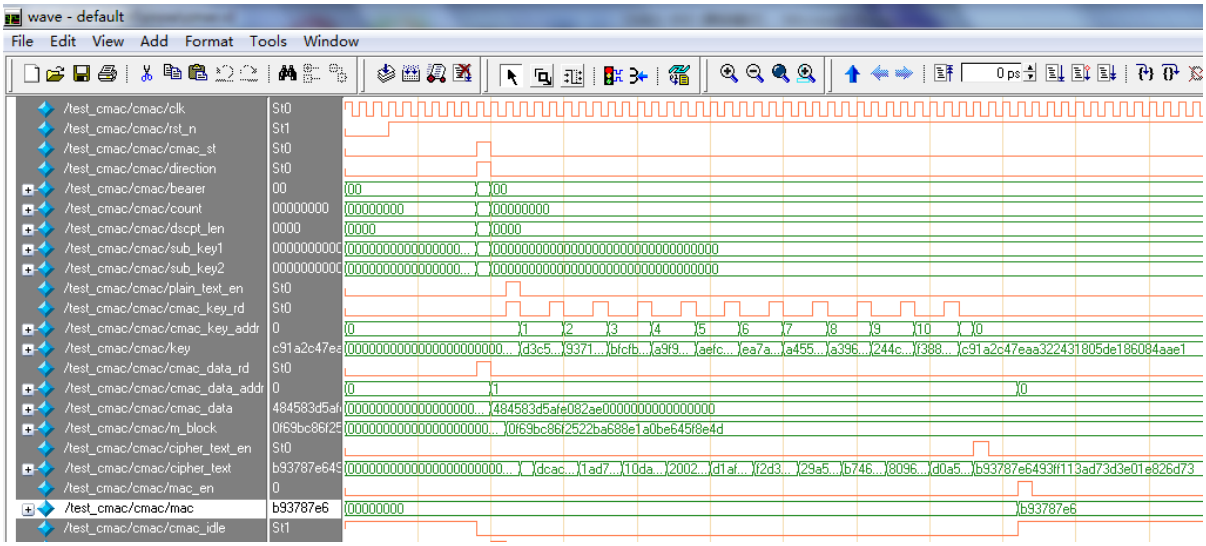

图 5.3 AES-CMAC 模块功能仿真波形

AES-CTR 模块、UEA2 模块、UIA2 模块和密钥扩展的仿真过程和上面的方式相同，其中 UEA2 模块、UIA2 模块使用文献[32]中提供的测试数据；AES-CTR 模块和密钥扩展模块分别使用 LTE/SAE 协议和文献[2]提供的测试数据。经过仿真测试，各模块测试结果及接口时序和预期相同，而且工作正常。因此，这四个模块同样在功能上达到设计要求。

5.1.2 系统逻辑功能仿真

在集成电路的开发过程中，应用总线功能模型（BFM），能提高 SOC 系统的验证效率。因此总线功能模型是验证系统中最主要的一部分。在系统逻辑功能仿真阶段，本文采用了由 System Verilog 语言搭建的可复用的仿真环境。如图 5.4 所示为系统仿真平台结构。

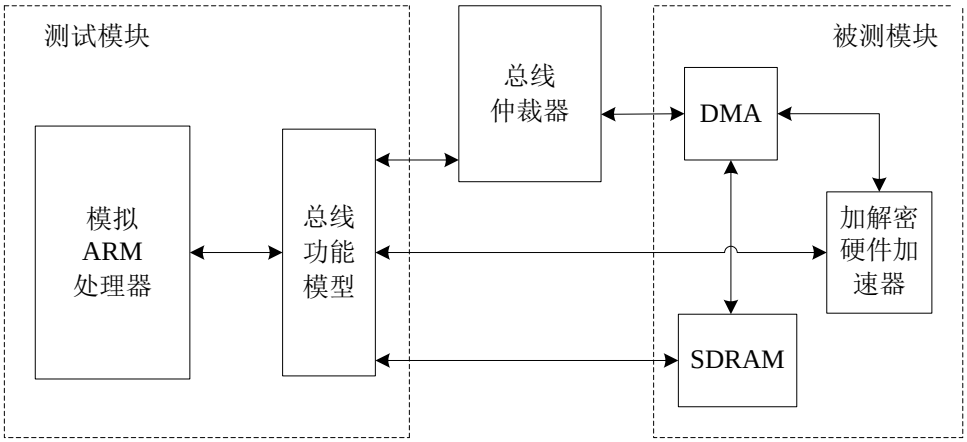

图 5.4 系统仿真平台结构

在测试模块中，模拟 ARM 处理器模块模拟 ARM 过程产生测试所需的激励，该处所用的总线功能模型为 AHB 总线功能模型。总线仲裁器负责将总线上的各个设备连接起来，并且根据优先级仲裁各个设备对总线的使用权，总线仲裁器是 AHB 总线结构中比较重要的一部分。在被测模块中需要用到 SDRAM 仿真模型和 DMA 这两个功能模块。在该仿真平台中，被测部分可以是任何符合 AHB 总线协议的系统。

对于硬件加速器的系统功能仿真，如图 5.5 和图 5.6 所示。其中图 5.5 为基于 AES 的加解密硬件加速器加密和完整性保护功能仿真波形，分别使用 CTR 模式和 CMAC 模式完成对数据的加密和完整性计算。本次仿真数据采用文献[4]中所提供的测试数据。其中使用的一组测试数据为：

COUNT：$\{36af6144\}_{16}$； BEARER：$\{0f\}_{16}$； DIRECTION：1；

子密钥 K1：$\{3bec38ae790d5958e09b73ab61bd480f\}_{16}$；

子密钥 K2：$\{77d8715cf21ab2b1c136e756c37a901e\}_{16}$；

加密密钥 K：$\{83fd23a244a74cf358da3019f1722635\}_{16}$；

完整性计算密钥 K：$\{83fd23a244a74cf358da3019f1722635\}_{16}$；

明文数据：{36af61447c00000035c68716633c66fb750c266865d53c11ea05b1e9fa49c839

8d48e1efa5909d3947902837f5ae96d5a05bc8d61ca8dbef1b13a4b4abfe4fb1006045b674bb547

29304c382be53a5af05556176f6eaa2ef1d05e4b083181ee674cda5a4 85f74d7a}$_{16}$；

密文数据：{ff19112df067ae690ffde4816dfea75b3753f7c0c962d01e1433f668eccc9d9fd5

025f097b0abec966f36d2f017cb68f1a6cd2532d30ad366669d8242500df3932708375ae4056ce

63d3e7f0371facc2f92abeb7fcda93bb251295fe9ebeda6a}$_{16}$；

数据长度：768bit； MAC：$\{e657e182\}_{16}$。

仿真开始，首先 ARM 读取输入型 FIFO 的空间信息，这里 arm_fifo_depth 的值为 $\{3fe\}_{16}$，说明有空间接受 ARM 写如描述符地址和长度信息。这时 ARM 通过 AHB 总线向硬件加速器中发送描述符地址和长度信息。当硬件加速器判断输入型 FIFO 的读深度大于或等于“2”时开始读取输入型 FIFO 中的描述符地址和长度信息，接着发出申请使用 DMA 信号 dma_reg，然后等待 DMA 返回 dma_avail 信号后开始发出 dma_st 信号配置 DMA，将搬运需要的长度信息 dma_len、源起始地址 dma_src 和目标搬运地址 dma_dst 配给 DMA，DMA 接收到配置信息后开始从 SDRAM 中搬运描述符。当描述符搬运结束后 DMA 发出 dma_dn 信号表示本次搬运结束。这时开始解析描述符，描述符解析完成后根据获得的数据长度和存储地址信息配置 DMA 从 SDRAM 中搬运数据。等待 DMA 发出 dma_dn 信号后开始使能加解密 IP 核启动信号 enc_dec_st，启动加解密 IP 核对数据进行加密和完整性计算。同时将产生的密文和认证码通过 cipher 接口存入数据存储模块中。之后再次配置 DMA 将处理后的数据搬运到 SDRAM 中指定的地址，一次处理结束。之后进行下一次处理。从图 5.5 中 cipher 接口可以看到进行加密和完整性计算结果和预期的测试结果相同以及时序控制和预期相同。

图 5.6 为基于 SNOW 3G 的加解密硬件加速器加密和完整性保护功能仿真波形，分别使用 UEA2 模式和 UIA2 模式完成对数据的加密和完整性计算。仿真数据采用文献[32]中所提供的测试数据。其中使用的一组测试数据为：

COUNT：$\{72a4f20f\}_{16}$； BEARER：$\{0c\}_{16}$； DIRECTION：1；

加密密钥 K：$\{2bd6459f82c5b300952c49104881ff48\}_{16}$；

完整性计算密钥 K：$\{c736c6aab22bff91e2698d2e22ad57e\}_{16}$；

明文数据：$\{7ec61272743bf1614726446Aa6c38ced166f6ca76eb5430044286346cef130f9$

$2922b03450d3a9975\ e5bd2ea0eb55ad8e1b199e3ec4316020e9a1b285e7627953\}_{16}$；

密文数据：$\{8ceba62943dced3a0990b06ea1b0a2c4fb3cedc71b369f42ba64c1eb6665e72a$

$a1c9bb0deaa20fe86058b8baee2C2e7f0becce48b52932a53c9d5f931a3a7c53\}_{16}$；

数据长度：512bit； MAC：$\{42f69325\}_{16}$。

图 5.6 的测试过程和图 5.5 的测试过程相同，从图 5.6 的 cipher 接口可以看到进行加密和完整性计算结果和预期的测试结果相同以及时序控制和预期相同。

对于硬件加速器的解密和完整性验证过程的仿真与加密和完整性保护过程相同，经过仿真测试，测试结果及接口时序和预期相同，而且工作正常。由以上测试结果说明该硬件加速器是正常工作的，在功能上达到了设计要求。

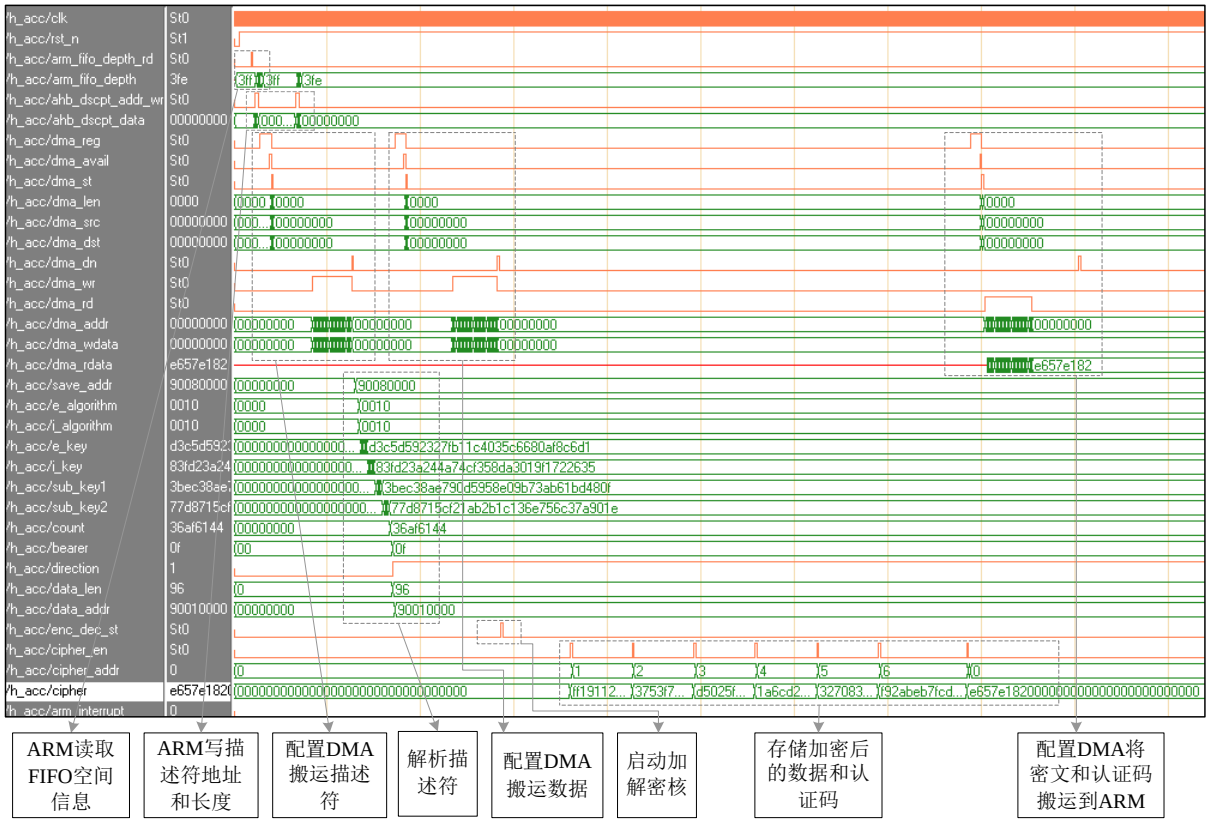

图 5.5 基于 AES 的加解密硬件加速器加密和完整性保护功能仿真波形

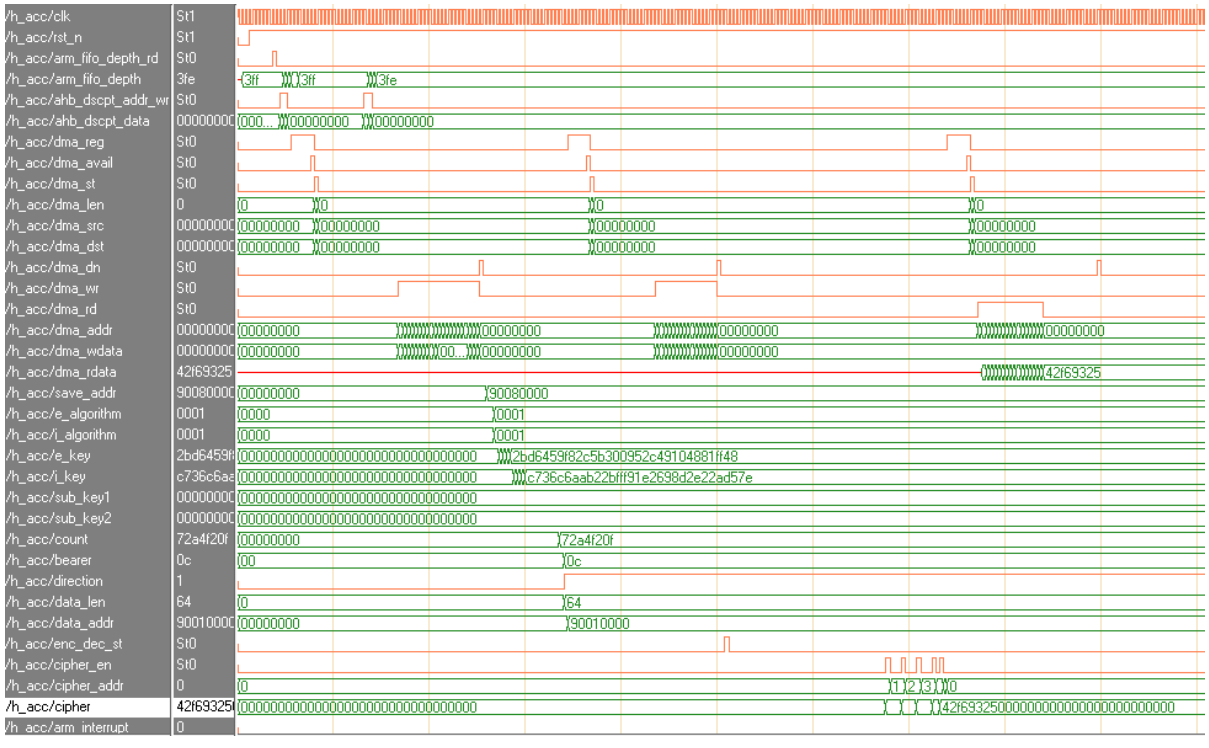

图 5.6 基于 SNOW 3G 的加解密硬件加速器加密和完整性保护功能仿真波形

5.2 静态时序分析

静态时序分析（STA）的分析速度比较快，占用内存少，不依赖于激励，而且它会对所有可能的路径都进行检查，不存在遗漏关键路径的问题。它完全克服了动态时序验证的缺陷，适合对超大规模的片上系统电路进行验证，能节省近 20%的设计时间。静态时序分析是通过计算信号沿在路径上的延迟传播找出违背时序约束的错误，主要是检查建立时间和保持时间是否满足要求。

本设计使用 ISE 软件，通过 ISE 综合产生的约束文件和中间文件做为输入，对设计进行静态时序分析。本设计的静态时序分析是在 Xilinx 公司的 xc6vlx550t 芯片上进行的。图 5.7 所示为本设计在 ISE 综合布局布线后 100MHz 时钟约束条件下的静态时序分析报告。

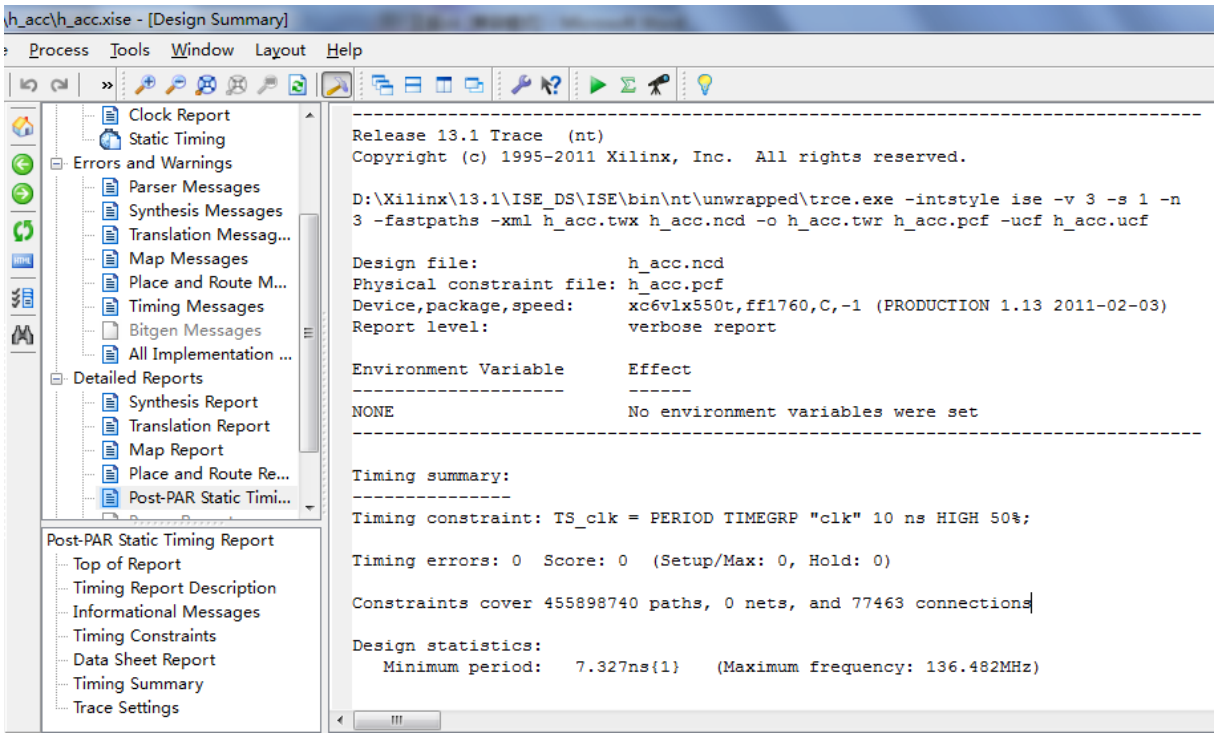

图 5.7 静态时序分析报告

从图 5.7 中可以看出本设计的最高时钟频率为 136.482MHz。在 100MHz 时钟约束条件下时序没有违背约束条件，因此，本文所设计的硬件加速器时序满足要求。

5.3 FPGA 验证

由于基于 FPGA 的设计比基于 ASIC 的设计具有很大的成本优势，且速度比逻辑仿真快许多，所以基于 FPGA 的验证技术在 SOC 系统级验证中得到了广泛的应用。而且

FPGA 是可编程可重用的器件，使得设计和验证的效率得到很大的提高[23]。

本设计使用 FPGA 硬件验证平台对硬件加速器进行验证，最后对关键模块所使用的 FPGA 资源及结果进行了统计。 本设计的 FPGA 验证平台使用 Xilinx 公司的 Virtex-6 开发平台，该平台的核心芯片为 xc6vlx550t。验证中 ARM 平台使用 ARM 公司的 ARM926EJ-S 开发平台。FPGA 硬件验证平台如图 5.8 所示。

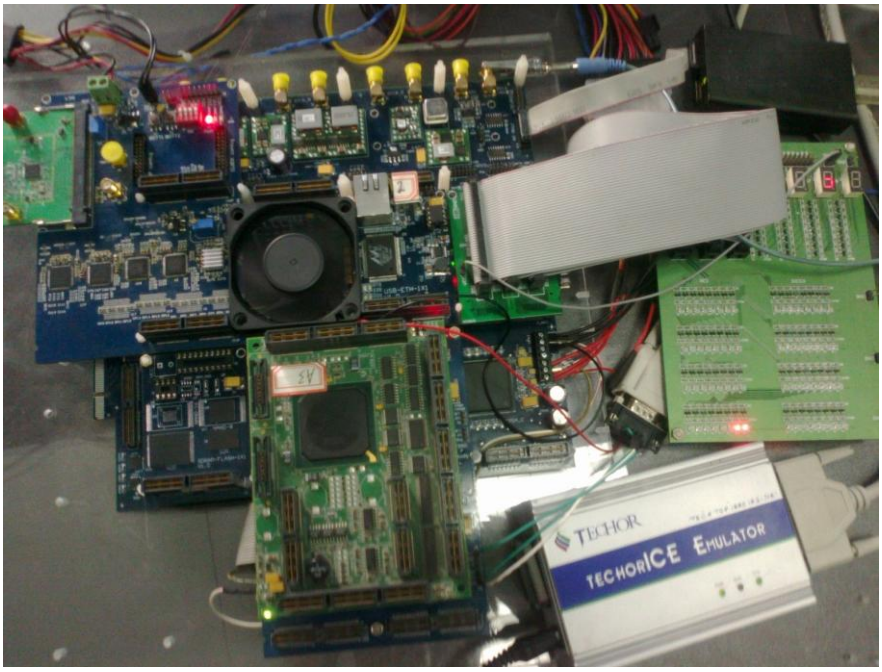

图 5.8 FPGA 硬件验证平台

由于 FPGA 内部信号的可观性差，所以在 FPGA 验证过程中传统做法是采用逻辑分析仪来观察信号波形，将需要观察的信号接到 FPGA 的引脚上进行观察。但是，由于 FPGA 管脚有限，每次能观察的信号数受限，而且逻辑分析仪价格十分昂贵。相对而言，Chipscope 是 Xilinx 推出的一款在线调试软件，价格便宜，其本身就是一个逻辑分析仪。通过它完全可以脱离传统逻辑分析仪来调时序，观察 FPGA 内部的任何信号，触发条件、数据宽度和深度等的设置也非常方便。所以本设计采用 Chipscope 对验证结果进行分析。编写代码时将数据采集核（ICON 和 ILA）例化到 RTL 代码中，将需要观察的信号接到该模块接口上。验证时，就可以使用 Chipscope 软件来观察 FPGA 内部的信号。

FPGA 验证过程中使用和系统仿真过程中相同的测试数据。测试时，把综合工具（ISE13.1）产生的 FPGA 配置文件通过 PC 机的 USB 接口经 JTAG 的转接器下载到 FPGA 中。同时将测试程序用调试软件通过 PC 机的并口经转接器下载到 SDRAM 中，ARM 处理器通过读取 SDRAM 只中的指令执行。 等待运行稳定通过 Chipscope 软件设定触发条件，通过 JTAG 口将采集到的 FPGA 内部信号的波形上载到 PC 机，观察到的结果与预期设定结果相符合。

5.4 FPGA 实现结果

本文所设计的硬件加速器在加密和完整性保护中加密和完整性计算过程同时进行，在解密和完整性验证中需先进行解密再进行完整性验证。本设计中 AES 核和 SNOW 3G 核是本文的关键设计模块，表 5.1 所示为 AES 核、SNOW 3G 核和加解密硬件加速器通过 ISE 工具在芯片 xc6vlx550t 上综合布局布线后所使用的资源情况。

表 5.1 各硬核使用 FPGA 资源统计表

名称	Slices	RAM 数量	最高频率（MHz）	吞吐率（Mbps）
AES 核	534	0	207.98	858.8
SNOW 3G 核	361	0	155.5	4976
加解密硬件加速器	3964	7	136.48	249.6

AES 核完成一轮操作需要 3 个时钟周期，由图 2.4 可知，完成一组数据处理需要进行十轮变换和一次轮密钥加过程，所以完成一组数据处理需要 31 个时钟周期。在 207.98 MHz 的时钟下吞吐率可以到达 858.8 Mbps。SNOW 3G 核在运算开始要对 FLSR 中的状态寄存器进行初始化运算需要 32 个时钟周期，之后每个时钟周期输出 32 位的密钥流，在 155.5MHz 的时钟下吞吐率最高基本能达到 4976 Mbps。

由上表可以看出本文所设计硬件加速器使用了七个 RAM 块，这七个 RAM 块分别用来存储 DMA 搬运的数据、扩展后的密钥和描述符信息等数据。该硬件加速器在 136.48MHz 的时钟频率下吞吐率为 249.6 Mbps，在 100MHz 的时钟频率下吞吐率为 182.9Mbps，达到了预期在 100MHz 的时钟频率下吞吐率为 100Mbps 的目标。验证结果表明本文所设计的硬件加速器在满足各项功能及设计目标的基础上实现了速度和资源的平衡，具有较好的速度面积比。

5.5 本章小结

本章主要介绍了硬件加速器的仿真及 FPGA 验证。主要包括功能验证、静态时序分析和 FPGA 验证。首先，对各子模块及系统进行了逻辑功能仿真；其次，在设计综合布局布线后对其进行了静态时序分析；最后，在基于 Virtex-6 的 FPGA 硬件验证平台上对系统进行了验证，FPGA 实现结果表明设计在满足各项功能及设计目标的基础上实现了速度和资源的平衡，具有较好的速度面积比。

6 总结与展望

6.1 总结

随着无线移动通信的不断发展，LTE 技术的推进，无线移动通信系统作为一种便捷、有效的通信技术，已逐步成为人们交流和联系的最广泛的手段，其传递信息的安全性和保密性也更加广泛的被行业所关注。因此在移动通信过程中，确保传播数据的安全性及完整性至关重要。

本文根据 LTE 安全性架构协议，通过对 AES 算法、SNOW 3G 算法、UEA2 算法和 UIA2 算法以及 CTR 模式和 CMAC 模式的深入研究。在基于 AMBA 总线的 SOC 系统架构的基础上，研究设计了一款基于 AHB 总线应用于 LTE 终端 PDCP 层的加解密硬件加速器。该硬件加速器实现了 PDCP 层的数据加密和完整性保护工作以及数据解密和完整性验证工作。最后通过实现结果可以看出该硬件加速器在 100MHz 的时钟频率下吞吐率为 182.9 Mbps，达到了预期在 100MHz 的时钟频率下吞吐率为 100Mbps 的目标，且在满足各项功能及设计目标的基础上实现了速度和资源的平衡。本论文完成的主要工作如下：

1、对 LTE 安全性架构协议和 ARM 公司的 AMBA 总线架构进行了深入的研究与分析，提出并确立了一种有效的基于 AHB 总线的加解密硬件加速器和 ARM 处理器的数据交互流程方案。并且采用自顶向下的设计方式，完成了硬件加速器总体结构框架设计及各模块划分。

2、实现了密钥扩展模块和 128 位数据通路的可重用 AES 核的硬件电路设计，并且实现了基于 AES 算法的 CTR 加密模式和 CMAC 完整性计算模式的硬件电路设计。

3、实现了 SNOW 3G 核的硬件电路设计以及基于 SNOW 3G 的 UEA2 加密算法和 UIA2 完整性计算算法的硬件电路设计。

4、采用 Verilog 硬件描述语言，对硬件加速器及各模块进行 RTL 级描述。

5、建立仿真环境，使用 modesim 工具对各个模块及系统进行了逻辑功能验证，使用 ISE 工具对本设计综合布局布线后进行了静态时序分析。

6、采用基于 Virtex-6 的 FPGA 硬件验证平台，模拟 LTE 终端通信环境，在 FPGA 上对本设计进行全面验证。实现结果表明本设计达到了预期的设计目标。

本文的特色在于：

1、加密和完整性保护及解密和完整性验证过程中使用相同的硬件电路结构，节约了近一半的硬件资源开销。并且在加密和完整性保护中加密和完整性计算模块并行工作，提高了设计性能。

2、在 SNOW 3G 核的设计中，S_R 盒采用与 AES 中的 S 盒相同的实现方式，节约了

硬件资源。

3、在 AES 核设计中，将行移位、列混淆和轮密钥加在一个时钟周期完成，提高了 AES 核的性能。

4、在密钥管理方案中，采用硬件提前对初始密钥进行密钥扩展运算并且存储的方式，在下次运算密钥不变换情况下，不用重新对初始密钥进行密钥扩展运算，节约了功耗。

6.2 展望

本设计的重点主要是前端的设计验证部分，由于时间和条件限制，本设计只进行了前端的 FPGA 验证，因此对于后端的版图设计是后续要做的工作。此外可以考虑将 SNOW 3G 算法中的 S_Q 盒采用基于有限域降阶的组合逻辑方式实现以及将 AES 算法中的字节代换操作在基于复合域 $GF(((2^2)^2)^2)$ 上实现，进一步减少硬件开销。

致谢

首先，我要衷心地感谢我的导师吴延海教授这三年对我在学习上的悉心指导和生活上的关怀。吴老师严谨的治学态度、勤奋务实的工作作风以及谦和的为人态度使我受益匪浅，从论文的选题，开题以及到最后完成，在每一步的进行当中都对我精心指导。所以在即将完成研究生阶段学业之际，我向吴延海老师表达我衷心的感谢和祝福。

感谢西安科技大学通信学院的所有老师，感谢老师们辛勤的授课，感谢学校给我们提供了良好的科研和学习环境。

感谢薛静、姜海旭、李佳新等同学在学习和生活上的支持和帮助。

感谢在信源通公司实习期间的各位同事，特别感谢实习期间的指导老师马崇良老师对我工作和生活中的帮助和鼓励以及验证方面的支持。

感谢深爱我的父母，我的成长，我在学业和生活上的进步都离不开你们默默无闻的付出。谢谢你们！

感谢所有曾经关心、支持和帮助过我的朋友们。

参考文献

[1] SESIA Stebnia, TOUFIK Issam, and BAKER Matthew. LTE-UMTS long term evolution: from theory to practice[M]. United States: John Wiley & Sons Ltd, 2009

[2] National Institute of Standards and Technology (NIST). Advanced Encryption Standard. FIPS PUB 197[S]. Springfield: FIPS. 2001

[3] Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Docuument 2: SNOW 3G Specification, ETSI/SAGE Specification, Version 1.1, Sep. 2006

[4] 3GPP System Architecture Evolution (SAE): Security Architecture, 3GPP Std. TS 33.401, Rev. 8.2.1, Dec. 2008

[5] National Institute of Standards and Technology (NIST), NIST Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation Methods and Techniques, Dec. 2001

[6] National Institute of Standards and Technology (NIST), NIST Special Publication 800-38B: Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, May. 2005

[7] Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 1: UEA2 and UIA2 Specification, ETSI/SAGE Specification, Version 2.1, 2009

[8] 司先秀．TD-LTE 终端芯片瓶颈期待突破，世界电信，2011

[9] Rouvroy G, Standaert FX, Quisquater JJ, et al. Compact and efficient encryption /decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications[J]. Itcc 2004: International Conference on Information Technology: Coding and Computing, Vol 2, Proceedings, 2004

[10] P. Hamalainen, T. Alho, M. Hannikainen, et al. Design and Implementation of Low-area and Low-power AES Encryption Hardware Core, Proc. of the 9th EUROMICRO Conference on Digital System Design (DSD 2006), IEEE, 2006

[11] A. Hodjat and I. Verbauwhede. Area-Throughput Trade-offs for Fully Pipelined 30 to 70Gbits/s AES Processors, IEEE Transactions on Computer, IEEE, 2006

[12] M. Alam, S. Ray, S. Ghosh, et al. An Area Optimized Reconfigurable Encryptor for AES-Rijndael. Design, Automation&Test in Europe Conference&Exhibition, 2007

[13] S. Hessel, D. Szczesny, N. Lohmann. Implementation and Benchmarking of Hardware

Accelerators for Ciphering in LTE Terminals[J], IEEE Global Telecommunications Conference, 2009

[14]王映民，孙韶辉，等．TD-LTE 技术原理与系统设计．北京：人民邮电出版社，2010

[15]林东岱．代数学基础与有限域．北京：高等教育出版社，2006．32-82

[16]张洪铭，何登平．基于 LTE 系统的 SNOW3G 密算法研究，宽带网络，2010

[17]皮埃尔，蒂埃里．演进分组系统（EPS）：3G UMTS 的长期演进和系统结构演进．北京：机械工业出版社，2009

[18]劳丰．AMBA 总线协议关键模块设计与验证研究：[学位论文]．合肥：中国科技大学，2008

[19]ARM L. AMB ATM Specification (rev2.0). 1995

[20]ARM Limited. AMBA Specification(ARM IHI OO11A)，Version2.0，1999

[21]J. Shim, D. Kim, Y. Kang, etc. A Rijndael Cryptoprocessor Using Shared On-the-fly Key Scheduler, Proceedings. IEEE Asia-Pacific Conference on ASIC, IEEE, 2002

[22]N. Pramstaller, S. Mangard, S. Dominikus, et al. Efficient AES Implementations on ASIC and FPGA, AES 2004,LNCS 3373, Springer-Verlag, 2005

[23]高利军．WiMAX 通信终端中安全层的硬件加速器的研究与设计：[学位论文] ．西安：西安交通大学，2011

[24]C. Lu and S.Tseng. Integrated Design of AES Encrypter and Decrypter, Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors(ASAP'02), IEEE, 2002

[25]A.Rudra, P.Dubey, C.Jutla, et al. Efficient Rijndael Encryption Implementation with Composite Field Arithmetic. In: Koç Ç, Naccache D, Paar C, editors. Cryptographic Hardware and Embedded Systems-CHES 2001. Springer Berlin/Heidelberg, 2001: 171-184

[26]Zhang XM, Parhi KK. On the optimum constructions of composite field for the AES algorithm[J]. IEEE Transactions on Circuits and Systems Ii-Express Briefs, 2006, 53 (10): 1153-1157

[27]A. Satoh, S. Morioka, K. Takano, et al. A Compact Rijndael Hardware Architecture with S-Box Optimization. In: Boyd C, editor. Advances in Cryptology-ASIACRYPT 2001. Springer Berlin/Heidelberg, 2001: 239-254

[28]E. Mastrovito. VLSI Designs for Multiplication over Finite Fields GF(2^m). In: Mora T, editor. Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. Springer Berlin Heidelberg, 1989: 297-309

[29]A. Satoh, S. Morioka, K. Takano, et al. A Compact Rijndael Hardware Architecture with

S-Box Optimization. In: Boyd C, editor. Advances in Cryptology-ASIACRYPT 2001. Springer Berlin/Heidelberg, 2001: 239-254

[30] P. Kitsos, G. Selimis and O. Koufopavlou. A High Performance ASIC Implementation of the SNOW 3G Stream Cipher, in 16th International Conference on Very Large Scale Integration (VLSI-SoC 2008), Rhodes Island, Greece, Oct. 2008

[31] 马光胜，冯刚．SOC 设计与 IP 核重用技术．北京：国防工业出版社，2006．122-123

[32] Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 3: Implementors' Test Data, ETSI/SAGE Specification, Version 1.0, Sep. 2006