基于RISC-V指令扩展方式的国密算法SM2、SM3和SM4的高效实现

王明登[1],严迎建[1],郭朋飞[1*],张 帆[2]
(1. 信息工程大学密码工程学院,河南郑州 450001;2. 浙江大学网络空间安全学院,浙江杭州 310058)

摘 要: 基于指令扩展的密码算法实现是兼顾性能和面积的轻量级实现方式,特别适用于日益普及的物联网设备. SM2、SM3和SM4等国密算法有利于提高自主可控设备的安全性,但针对这些算法进行指令扩展的相关研究还不够充分. RISC-V由于其开源、简洁及可扩展等优点已成为业界最流行的指令集架构之一,本文主要基于国产开源RISC-V处理器对国密算法SM2、SM3和SM4进行指令扩展和高效实现. 本文基于软硬件协同的理念提出总体指令的扩展方案. 对相关密码算法进行深入分析和方案对比,分别设计了硬件单元,提出高效的实现方式. 设计实现的协处理器具有2级流水线结构,顺序派遣、乱序执行和顺序写回的指令执行模式,以及独立内存访问单元和大位宽寄存器. 协处理器统一接管了密码算法的部分控制逻辑,降低硬件资源消耗. 实验结果表明,本文设计的密码协处理器硬件结构精简,资源利用率高. SM2、SM3和SM4算法占用资源少,但执行速率相比纯硬件有一定程度下降,资源面积和花费时间的乘积与其他相关文献相比有不同程度的优势.

关键词: RISC-V;协处理器;国密算法;指令扩展;蜂鸟E203;嵌入式系统
基金项目: 国家自然科学基金(No.62072398);河南省网络空间态势感知重点实验室开放课题(No.HNTS2022001)
中图分类号: TP332;TP309 **文献标识码:** A **文章编号:** 0372-2112(2024)08-2850-16
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20230391

Efficient Implementation of National Security Algorithms SM2, SM3 and SM4 Based on RISC-V Instruction Extension Method

WANG Ming-deng[1], YAN Ying-jian[1], GUO Peng-fei[1*], ZHANG Fan[2]
(1. *School of Cryptography Engineering*, *Information Engineering University*, *Zhengzhou*, *Henan* 450001, *China*;
2. *School of Cyber Science and Technology*, *Zhejiang University*, *Hangzhou*, *Zhejiang* 310058, *China*)

Abstract: The implementation of the cryptographic algorithm based on instruction extension is a lightweight scheme that balances both performance and area, which is especially suitable for the increasingly popular Internet of Things devices. The proposal of national cryptographic algorithms such as SM2, SM3, and SM4 is conducive to improving the security of self-controlled devices. However, the relevant research on instruction extensions for these algorithms is insufficient. RISC-V has become one of the most popular instruction set architectures due to its advantages of open source, simplicity, extensibility, etc. This paper mainly focuses on the instruction extensions and efficient implementation of the SM2, SM3, and SM4 algorithms based on a domestic open-source RISC-V processor. Specifically, this paper proposes an overall instruction expansion scheme based on the concept of hardware-software co-design; this paper conducts an in-depth analysis of the related cryptographic algorithms and comparison of the implementation schemes and then proposes efficient implementations of the hardware units, respectively. This paper designs and implements a coprocessor with a two-stage pipeline structure, sequential dispatching, out-of-order execution, and sequential write-back instruction execution modes, as well as an independent memory access unit and a large bit-wide register. The coprocessor takes over part of the control logic of the cryptographic algorithm, reducing hardware resource consumption. The experimental results show that the hardware structure of the cryptographic coprocessor designed in this paper is simplified, and the utilization of hardware resources is high. SM2, SM3, and SM4 algorithms occupy very few resources, but the execution rate decreases only to a certain extent compared

收稿日期:2023-04-28;修回日期:2023-11-06;责任编辑:梅志强
*通讯作者:郭朋飞

with pure hardware implementation. The product of resource area and time spent has varying degrees of advantages compared to other relevant literature.

Key words: RISC-V; co-processors; national cryptographic algorithm; instruction extensions; hummingbird E203; embedded system

Foundation Item(s): National Natural Science Foundation of China (No. 62072398); Open Foundation of Henan Key Laboratory of Cyberspace Situation Awareness (No.HNTS2022001)

1 引言

近年来,信息安全问题日益突出,特别是随着物联网技术的发展,端侧设备被大量部署,且经常搜集和处理诸如个人生物特征、健康监测数据、行为习惯、密码口令等敏感信息[1]. 密码技术是保障网络与信息安全的核心技术和基础支撑,而密码算法的具体实现方式多种多样,具有不同特点. OpenSSL[2]、Botan[3]和Libgcrypt[4]等密码算法库均为纯软件实现,这种实现方式具有成本低、灵活性高但效率低的特点. 密码算法的纯硬件实现包括专用集成电路(Application Specific Integrated Circuit, ASIC)[5]、可重构密码芯片[6]和现场可编程门阵列(Field Programmable Gate Array, FPGA)[7,8]等方式,这种实现方式的优势是算法执行速度快,但成本高且可扩展性差. 基于指令扩展的密码算法实现方式[9,10]兼顾了性能和成本,具有能效比高和可扩展性强等特点. 密码算法的每种实现方式都有相对合适的适用场景,在资源受限的轻量级嵌入式设备中使用指令扩展方式实现密码算法是合理的选择[11~13].

不同的指令集架构具有各自特点,x86和Arm等主流的指令集架构往往较复杂且可扩展性弱,也存在知识产权问题. RISC-V是新兴的指令集架构,具有架构精简、免费开源和可扩展等优势,开发人员可以非常容易地在RISC-V架构上实现专用加速器[14],因此,基于RISC-V指令集架构更适合进行密码算法自定义指令的扩展. 目前,该方向已有广泛研究,主要是面向AES、RSA、ECC和SHA等国外典型商用密码算法[14~17]. 在一些关键领域,使用国外算法仍然存在未知的安全隐患. 但目前针对中国提出自主商用密码算法进行RISC-V指令扩展的研究相对较少.

针对国密算法实现的研究主要集中在纯硬件实现和软硬协同实现上,其中,纯硬件实现已有大量研究[18~22],这些硬件结构大致可分为循环结构、流水线结构和混合结构,但无论哪种硬件结构均需要在硬件中完整实现整个算法流程,占用资源面积较大且复杂度较高. 部分文献对国密算法的软硬协同设计进行了研究,文献[23]针对SM3算法实现了处理器与FPGA协同工作的机制,使SM3能在FPGA上快速运行,处理器能够对FPGA进行控制和数据交互,这种实现方式具有较高的执行速度,但需要额外FPGA的支持,与纯硬件实现存在相同问题. 文献[24]提出具有SM2专用指令的硬件协处理器,通过类似通用处理器的工作方式执行专用指令序列,实现SM2签名、验签、加密、解密中除辅助函数之外的所有运算,然而该文献的椭圆曲线基于素数域,采用4级流水线结构,导致硬件资源消耗大且复杂度较高. 文献[25]采用软硬协同的设计方法,实现了基于SM2、SM3和SM4的高效加解密和数字签名方案,采用并行化的硬件结构以提高算法执行速度,这也导致硬件资源消耗较大. 文献[26]设计了低开销的SM4指令硬件电路结构,具有较低的资源面积,但该方案在主处理器的寄存器堆处添加了辅助电路,增加设计的复杂度. 可见,针对国密算法的大部分研究主要针对运算性能进行优化,而未重点考虑资源面积和复杂度,不适合轻量级的嵌入式系统实现. 此外,因为诸如安全启动[27]、身份认证[28]和混合加密[20]等安全机制需要多种不同类型的密码算法支持,许多研究仅针对对称、杂凑和公钥密码算法中的1种或2种,应用范围较窄. 本文基于RISC-V处理器——蜂鸟E203针对国密算法设计了适合轻量级嵌入式系统的密码加速协处理器,该协处理器能够在占用较少硬件资源的条件下完成SM2、SM3和SM4这3种国密算法的运算加速. 相比密码算法的纯硬件实现,本文设计的协处理器硬件资源消耗低,相比纯软件或传统的协处理器具有更快的执行速度. 相比以往的相关研究,本文设计中资源面积与花费时间的积(Area-Time product, AT)[20,29,30]较小,对于一般的硬件设计,花费时间减半则资源面积加倍,AT接近常数. AT客观反映了资源消耗和算法性能之间的关系,AT越小,表明在有限资源条件下,资源面积与性能之间取得的平衡越好.

本文的主要贡献有:(1)分析了国密算法SM2、SM3和SM4的低硬件开销算法实现方案,设计了3种密码算法的核心运算单元;(2)设计具有2级流水线结构的协处理器,该协处理器采用顺序派遣、乱序执行和顺序写回的指令执行模式,在没有资源或数据冲突的条件下实现自定义指令的并行执行;(3)采用软硬件协同的设计方式,将密码算法中部分控制逻辑交给软件层和协处理器公用控制单元,使协处理器的硬件资源消耗进一步降低;(4)扩展了数据读写和3种国密算法相关的自定义指令与函数接口,使软件层可在较小代码规模

下实现密码算法的运算加速.

2 相关基础

2.1 国密算法介绍

国家密码管理局从国家信息安全角度出发,制定了一系列自主可控的商用密码算法,并得到ISO/IEC(International Organization for Standardization/International Electrotechnical Commission)的认可,其中比较常用的3种密码算法是SM2公钥密码算法[31]、SM3杂凑算法[32]和SM4分组密码算法[33].

SM2算法是中国的公钥密码算法标准,达到了公钥密码算法的最高安全级别.SM2算法具有与ECC(Elliptic Curve Cryptography)算法相同的数学概念基础,但在签名、密钥交换方面不同于椭圆曲线数字签名算法(Elliptic Curve Digital Signature Algorithm,ECDSA)、椭圆曲线迪菲-赫尔曼密钥交换(Elliptic Curve Diffie Hellman,ECDH)等国际标准,SM2算法的待签名信息需要进行预处理,且在数字签名和验证、消息认证码的生成与验证以及随机数的生成等方面需要使用更为复杂的SM3杂凑算法,以及同一攻击都需要解决2个椭圆曲线离散对数问题.因此,与ECDSA、ECDH等国际算法相比,SM2算法具有更高的安全性.SM2算法具有明显的层次特点:最顶层为应用层,包括数字签名与验签、密钥交换和公钥加密;中间层包括椭圆曲线上的点加、倍点以及对同一点进行多次点加运算的点乘运算;最底层为模加减、模乘、模逆等有限域上的模运算.应用层算法基于椭圆曲线上的点运算和有限域上的模运算,椭圆曲线上的点运算同样基于有限域上的模运算,因此,对有限域上的模运算进行优化设计是提升性能和减少资源消耗的关键.

SM3算法是分组迭代的杂凑算法,将任意长度的消息经过填充和迭代压缩后生成256 bit的杂凑值,SM3算法可以分为3个部分:消息填充、消息扩展和迭代压缩.消息填充算法将长度为l bit的消息m填充为长度为512 bit整数倍的消息m',填充完成后将m'按512 bit 1组分组,每个消息组作为消息扩展算法的输入,消息扩展算法生成132个32 bit字$W_0, W_1, \cdots, W_{67}, W_0', W_1', \cdots, W_{63}'$,用于压缩函数CF,压缩函数根据消息扩展结果进行杂凑值的迭代更新,最后1轮迭代压缩结果即为最终的杂凑值.

SM4分组密码算法是国家密码管理局于2006年发布的第1个商用密码算法,其分组长度和密钥长度均为128 bit,加密算法与密钥扩展算法都采用32轮非线性迭代结构,数据解密和数据加密的算法结构相同,只是轮密钥的使用顺序相反,解密轮密钥是加密轮密钥的逆序.

2.2 蜂鸟E203 SoC总体架构

本文基于国内1款完全开源的微控制单元(MicroController Unit,MCU)级别的RISC-V SoC——蜂鸟E203进行设计,其结构如图1所示.该SoC使用了开源的蜂鸟处理器核,具有丰富的存储资源和外设接口,包括中断控制器、计时器、通用异步收发器(Universal Asynchronous Receiver/Transmitter,UART)、四通道串行外设接口(Queued Serial Peripheral Interface,QSPI)和脉冲宽度调制(Pulse-Width Modulation,PWM)等[34].

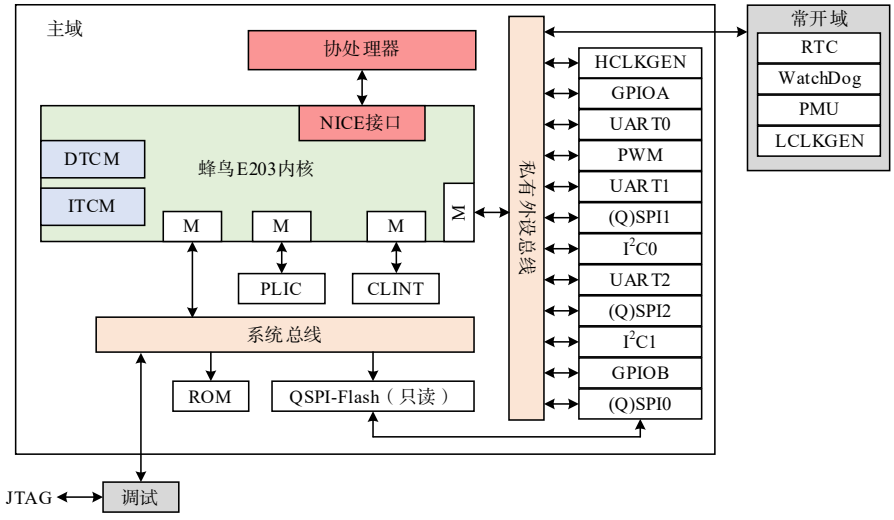


图1 蜂鸟E203 SoC结构图

蜂鸟E203处理器核主要面向低功耗与小面积场景,具有模块化、面积小和结构简单等特点,适合应用于物联网或其他低功耗场景.蜂鸟E203处理器核采用2级流水线结构,具有较高能效和较低成本,支持RV32IAMC等指令集的配置组合,提供标准的JTAG调试接口、成熟的软件调试工具和GCC编译工具链.

蜂鸟E203处理器核提供了自定义指令扩展接口——NICE（Nuclei Instruction Co-unit Extension）接口[35]，如图1所示，处理器核在译码阶段识别到自定义指令时，通过NICE接口的指令请求通道将自定义指令派遣到协处理器单元，协处理器执行完毕后通过反馈通道将执行结果写回. 此外，NICE接口中还提供了独立的内存访问通道，使协处理器具有直接访问系统内存的能力.

2.3　基于RISC-V的指令扩展方法

为了便于用户对指令集进行扩展，RISC-V指令集架构预定义了4组32位自定义指令类型，每种预定义指令均有自己的操作码[34]. 蜂鸟E203处理器核使用NICE接口进行协处理器扩展，用户将4组预定义指令扩展为NICE接口支持的指令，本文将这类指令简称为NICE指令，NICE指令的编码格式如图2所示，指令的第25~31位为额外编码空间，用于编码更多指令，因此，每组预定义指令可以使用额外编码空间编码出128条指令.

NICE接口包括4个通道[35]，如图3所示，分别是请求通道、反馈通道、存储器请求通道和存储器反馈通道. 当主处理器在译码阶段遇到任意1种预定义指令组时，会将指令和源操作数通过请求通道派发给协处理器，协处理器接收指令后做进一步译码并执行指令，执行完毕后通过反馈通道将结果反馈给主处理器. 协处理器执行NICE指令时，可通过存储器请求和反馈通道对主处理器中的数据进行连续读写，且读写过程与执行过程完全独立，具有较高读写效率.

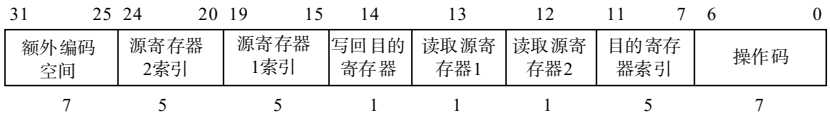


图2　NICE指令编码格式

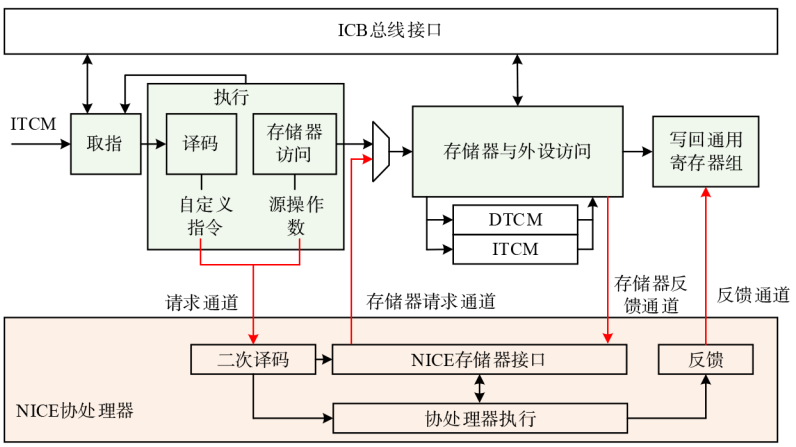


图3　NICE接口通道示意图

3　指令扩展方案设计

指令扩展方案设计要同时考虑软件层和硬件层的设计. 软件层设计包括NICE指令的约定和应用层对NICE指令的调用，硬件层设计包括密码运算单元和调用密码运算单元的协处理器设计.

RISC-V架构的汇编代码中调用用户自定义指令需要通过伪指令. insn来实现，本文对NICE指令的调用采用了C程序内嵌. insn伪指令并封装为接口函数方式，在后续的应用程序中只需按照C语言的规则进行调用即可. 对R类型指令. insn的使用格式为

. insn r opcode, func3, func7, rd, rs1, rs2

其中，r表示指令类型为R类型，opcode为操作码，func3表示指令源操作数和目的操作数寄存器索引是否有效，func7指定了具体的指令功能，rd、rs1和rs2分别是目的操作数和2个源操作数寄存器索引. 表1为本文在RISC-V架构基础上扩展的自定义指令，表中内嵌. insn伪指令的操作码均为custom-1类型自定义指令所对应的操作码0x2b，参数cfg用来对协处理器进行配置，包括协处理器的内部源操作数和目的操作数寄存器索引等，该参数可在协处理器设计时约定，参数addr表示主处理器内存读写的起始地址. 虽然利用NICE指令的源操作数和目的操作数寄存器可实现主处理器和协处理器间的数据交互，但这种方式速度较慢，蜂鸟E203处理器核的NICE指令提供了单独的存储器访问通道，利用该通道实现快速数据交互，表1中write_data和

表1 自定义指令

函数接口名	内嵌汇编指令	功能
write_data	asm volatile(".insn r 0x2b,7,0,%0,%1,%2":"=r"(zero): "r"(cfg),"r"(addr))	向协处理器寄存器中写入数据
read_data	asm volatile(".insn r 0x2b,7,1,%0,%1,%2":"=r"(zero): "r"(cfg),"r"(addr))	从协处理器寄存器中读取数据
sm4_ldkey	asm volatile(".insn r 0x2b,6,2,%0,%1,x0":"=r"(zero): "r"(cfg))	加载SM4算法的密钥
sm4_expkey	asm volatile(".insn r 0x2b,4,3,%0,x0,x0":"=r"(zero))	执行1轮SM4算法子密钥扩展
sm4_encrypt	asm volatile(".insn r 0x2b,6,4,%0,%1,x0":"=r"(zero): "r"(cfg))	执行1组明文数据的SM4加密
sm4_decrypt	asm volatile(".insn r 0x2b,6,5,%0,%1,x0":"=r"(zero): "r"(cfg))	执行1组密文数据的SM4解密
mod_add	asm volatile(".insn r 0x2b,6,6,%0,%1,x0":"=r"(zero): "r"(cfg))	完成256 bit数据的模加运算
mod_mul	asm volatile(".insn r 0x2b,6,7,%0,%1,x0":"=r"(zero): "r"(cfg))	执行1次模乘运算
mod_squ	asm volatile(".insn r 0x2b,6,8,%0,%1,x0":"=r"(zero): "r"(cfg))	执行1次模平方运算
sm3_ldblock	asm volatile(".insn r 0x2b,6,9,%0,%1,x0":"=r"(zero): "r"(cfg))	加载填充后的SM3消息分组
sm3_cmpress	asm volatile(".insn r 0x2b,6,10,%0,%1,x0":"=r"(zero): "r"(cfg))	完成1个消息分组的迭代压缩

read_data便是利用存储器访问通道进行数据交互的函数接口.

在表1所示自定义指令函数接口的基础上实现密码算法是较容易的,相比密码算法的纯软件实现,利用自定义指令不仅能大幅提高执行速度,还能减小代码规模,轻量级嵌入式系统的内存空间一般比较小,不能存储大量指令和数据,而自定义指令实现方式能很好地解决此问题. 同时,利用自定义指令实现密码算法能降低代码编写难度,减小出错可能性,提高系统稳定性. 此外,2个嵌入式设备完成密钥数据协商后,如果将密钥加载到协处理器而不是主处理器的内存中,将大大提高密钥的安全性.

在硬件层,需要设计协处理器来执行NICE指令,协处理器根据NICE接口规范完成主处理器和协处理器间指令和数据交互,根据约定的自定义指令生成控制信号,调用密码算法单元产生结果,最后将运算结果写入主处理器. 本文的协处理器设计方案为:首先,对密码算法进行分析,找到硬件资源消耗较少的实现方式;然后,对密码运算单元和调用密码算法单元的协处理器进行设计;最后实验验证.

4 密码算法分析

4.1 SM2算法分析

SM2算法的应用层依赖于点运算层和模运算层,椭圆曲线上的点运算包括点加、倍点和点乘运算,在不同坐标系下椭圆曲线上的点加运算和倍点运算的复杂度不同,其中,仿射坐标下的点加和倍点运算均需要进行1次模逆运算,占用大量运行时间,所以一般选用标准射影坐标或Jacobian加重射影坐标. 椭圆曲线点乘算法包括二进制展开法、加减法、滑动窗口法以及Montgomery算法等多种算法[31],其中,Montgomery算法是目前最安全、高效的算法之一,不需要进行预处理,整个点乘运算仅需要1次模逆运算,由于每轮主循环均需要进行1次点加运算和1次倍点运算,无法从功耗曲线判断k_i为0还是1,具有抗简单功耗分析的特性,标准投影坐标下的Montgomery算法如算法1所示.

算法1 标准投影坐标下的Montgomery点乘算法

输入: 大整数$k=(k_{l-1},\cdots,k_1,k_0)$, 且$k_{l-1}=1$, 椭圆点$P=(x_P,y_P)$

输出: $Q=kG=(x_Q,y_Q)$

1. IF $k==0$ OR $x_P==0$, RETURN $Q==(0,0)$
2. $X_1=x_P$, $Z_1=1$, $X_2=x_P^4+b$, $Z_2=x_P^2$
3. FOR $i=l-2$ DOWN TO 0 DO
4. IF $k_i==1$ THEN
5. $Z_1'=((X_1Z_2)+(X_2Z_1))^2$, $X_1'=(X_1Z_2)\cdot(X_2Z_1)+x_PZ_1'$
6. $Z_2'=X_2^2Z_2^2$, $X_2'=X_2^4+bZ_2^4$
7. ELSE THEN
8. $Z_2'=((X_1Z_2)+(X_2Z_1))^2$, $X_2'=(X_1Z_2)\cdot(X_2Z_1)+x_PZ_2'$
9. $Z_1'=X_1^2Z_1^2$, $X_1'=X_1^4+bZ_1^4$
10. $(X_1,X_2,Z_1,Z_2)=(X_1',X_2',Z_1',Z_2')$
11. END FOR
12. $\mathrm{inv}=(x_PZ_1Z_2)^{-1}$, $x_Q=x_PX_1X_2\cdot\mathrm{inv}$
13. $y_Q=((x_Q+x_P)(x_PZ_1X_2\cdot\mathrm{inv}+x_P)+x_P^2+y_P)\cdot(x_Q+x_P)$
$Z_1Z_2\cdot\mathrm{inv}+y_P$
14. RETURN $Q=(x_Q,y_Q)$

有限域上的模运算包括定义在素数域GF(p)上的模运算和定义在二元扩域GF(2^m)上的模运算. 由于GF(2^m)上的加法运算和减法运算均为异或运算,而乘法运算可以基于异或运算和移位操作,所以,GF(2^m)上模运算更适合轻量级嵌入式系统硬件实现. 有限域GF(2^m)中的元素有多种表示方法,其中,多项式基表示和正规基表示是最为常用的2种方法[31],用正规基表示的最大优点是模平方操作可以通过循环移位实现,但正规基下的模乘和模逆运算极为复杂,在硬件实现中占用大量时间和资源,所以本文选用多项式基下的表示方法,在多项式基下GF(2^m)上的元素可表示为

$$A(x)=a_{m-1}x^{m-1}+\cdots+a_1x+a_0 \tag{1}$$

GF(2^m)上的加法和减法运算均为2个多项式对应系数的异或运算,在硬件实现中也仅需要m个异或门即可实现.GF(2^m)上的乘法运算是整个SM2算法中最重要的部分,直接影响SM2架构的执行效率和资源开销.传统的模乘运算是将2个多项式相乘后对约减多项式$f(x)$取模,但这种方式需要用到多项式乘法和除法运算,时间和面积开销较大.目前GF(2^m)上常见的模乘算法有移位模二加算法[36,37]、Montgomery算法[38]和Karatsuba算法[39,40]等,其中,移位模二加算法因结构简单,不需要预计算和额外的存储空间被广泛应用在低功耗设备中.移位模二加算法主要有2种乘法器结构,分别是低位优先(Lest Significant Bit,LSB)乘法器和高位优先(Most Significant Bit,MSB)乘法器,2种结构相差不大,原始的LSB乘法器和MSB乘法器是串行迭代结构,完成1次模乘运算需要m次循环,迭代次数较多,算法效率太低,于是衍生出全并行结构和串并混合结构以减少迭代次数,全并行结构面积开销较大,基于优化LSB算法的串并混合结构更适合在轻量级嵌入式系统中实现,具体如算法2所示.

算法2　GF(2^m)上基于优化LSB算法的串并混合结构模乘算法

输入: $A(x)$, $B(x)\in$ GF(2^m), 单次迭代处理位数D

输出: $C(x)=A(x)\cdot B(x) \bmod f(x)$

1. $C=0$, $d=\lceil m/D \rceil$
2. FOR $i=0$ UP TO $d-1$ DO
3. 　IF $A==0$ RETURN C
4. 　$B_0=B$
5. 　FOR $j==1$ UP TO D
6. 　　$B_j=B_{j-1}+B_{j-1}^{m-1}f(x)$
7. 　$C=C+\sum_{k=0}^{D-1}a_kB_k$
8. 　$A=A>>D$
9. 　$B=B_D$
10. END FOR
11. RETURN C

注:$A=\sum_{k=0}^{m-1}a_kx^k, B_j=\sum_{k=0}^{m-1}B_j^kx^k$.

GF(2^m)上的模平方运算是特殊的模乘运算,多项式平方运算是线性操作,若$a(x)=a_{m-1}x^{m-1}+\cdots+a_2x^2+a_1x+a_0$,则$a^2(x)=a_{m-1}x^{2m-2}+\cdots+a_2x^4+a_1x^2+a_0$,平方运算后需要进行模约减运算,目前有2种常用的模平方约减方式,分别是预计算方式和约减矩阵方式[35],其中预计算方式需要根据约减多项式$f(x)$提前算出$a(x)$~$a^2(x) \bmod f(x)$对应位的关系,其优点是执行效率高,占用资源面积小,缺点是灵活性较低.约减矩阵的方式需要额外存储和计算约减矩阵,占用资源面积较大,但是灵活性较高.为了降低资源面积,本文采用了预计算的约减方式.

GF(2^m)上的模逆运算是有限域中最耗时的运算,基于扩展欧几里得算法和基于费马小定理的模逆算法是2种比较常用的模逆算法,前者需要额外增加模逆运算模块,后者可以将复杂的模逆运算转化为模乘和模平方运算,在硬件实现中可以复用模乘和模平方模块,本文将采用基于费马小定理的优化算法,具体如算法3所示.

算法3　GF(2^m)上基于费马小定理的改进模逆算法

输入: $A(x)=\sum_{i=0}^{m-1}a_ix^i$, $m-1=(m_{r-1},\cdots,m_1,m_0)_2$, 且$m_{r-1}=1$

输出: $C(x)=A^{-1}(x)=A^{2^m-2}(x)$

1. $B=A$; $u=1$
2. FOR $i=r-2$ DOWN TO 0 DO
3. 　$B=B^{2^u}\cdot B$, $u=u\cdot 2$
4. 　IF $m_i==1$ THEN
5. 　　$B=B^2\cdot A$
6. 　　$u=u+1$
7. END FOR
8. RETURN $C=B^2$

4.2　SM3算法分析

SM3算法可以分为消息填充、消息扩展和迭代压缩3个部分.消息填充算法将长度为l bit的消息m填充为长度为512 bit整数倍的消息m',对于长度为l bit的消息,将1 bit"1"添加到消息的末尾,再添加k个"0",k是满足$l+1+k\equiv 448 \bmod 512$的最小非负整数,最后添加1个64位比特串,该比特串是长度l的二进制表示.可见,消息填充算法主要针对末尾的消息分组,对长消息的杂凑值计算,使用软件或硬件进行消息填充的执行速度相差不大.

将填充后的消息m'按512 bit1组分组,并将消息分组划分为16个消息字$W_0,W_1,\cdots,W_{15}$,这些消息字作为消息扩展算法的输入,消息扩展算法如算法4所示,算法中需要进行116次循环才能生成132个32 bit字,在硬件电路中W_j和W_j'的生成可同时进行,在循环结构的硬件电路中进行64次循环迭代即可完成所有消息字的生成.

每个512 bit消息分组均会进行1次迭代压缩,具体压缩如算法5所示,每次迭代压缩需要进行64轮循环,每轮循环中所需要的外部输入为W_j和W_j'2个消息字,因此,在硬件单元设计中每个消息分组的消息扩展和迭代压缩过程可同步执行,在循环结构的硬件电路中完成1个消息分组的迭代压缩需要64次循环迭代.

算法4 SM3算法消息扩展

输入: 16个消息字 $W_0, W_1, \cdots, W_{15}$

输出: 132个32 bit字 $W_0, W_1, \cdots, W_{67}, W'_0, W'_1, \cdots, W'_{63}$

1. $(W_0, W_1, \cdots, W_{15}) = m'$
2. FOR $j = 16$ UP TO 67
3. $\quad W_j = P_1(W_{j-16} \oplus W_{j-9} \oplus (W_{j-3} <<< 15)) \oplus (W_{j-13} <<< 7) \oplus W_{j-6}$
4. END FOR
5. FOR $j = 0$ UP TO 63
6. $\quad W'_j = W_j \oplus W_{j+4}$
7. END FOR

算法5 SM3算法压缩函数

输入: 132个32 bit字 $W_0, W_1, \cdots, W_{67}, W'_0, W'_1, \cdots, W'_{63}$

输出: 256 bit杂凑值hash

1. $ABCDEFGH \leftarrow V^{(i)}$
2. FOR $j = 0$ UP TO 63
3. $\quad SS1 \leftarrow ((A <<< 12) + E + (T_j <<< j)) <<< 7$
4. $\quad SS2 \leftarrow SS1 \oplus (A <<< 12)$
5. $\quad TT1 \leftarrow FF_j(A, B, C) + D + SS2 + W'_j$
6. $\quad TT2 \leftarrow GG_j(E, F, G) + H + SS1 + W_j$
7. $\quad D \leftarrow C, C \leftarrow B <<< 9, B \leftarrow A, A \leftarrow TT1$
8. $\quad H \leftarrow G, G \leftarrow F <<< 19, F \leftarrow E, E \leftarrow P_0(TT2)$
9. END FOR
10. $V^{(i+1)} = ABCDEFGH \oplus V^{(i)}$

4.3 SM4算法分析

SM4算法包括加解密算法和密钥扩展算法2部分，SM4加解密算法采用32轮Feistel结构，加密和解密过程采用相同的轮函数结构和初始密钥，加解密过程中只是轮密钥的调用顺序相反，解密算法的轮密钥是加密算法轮密钥调用顺序的逆序，因此，在硬件实现中仅需要设计加密算法部分的硬件电路. SM4加密算法如算法6所示，该算法中主要有32 bit字异或、τ变换和L变换3种运算结构，其中，异或和L变换的硬件实现占用资源较少，而非线性变换τ占用资源较多，因此，对τ变换的实现方式关系到最终的资源占用和实现性能.

SM4密钥扩展算法如算法7所示，算法结构和SM4加密算法相似，主要将加密算法中的L变换改为了L'变换，同样非线性变换τ占用资源较多. 事实上，τ变换在硬件实现中为4个并行S盒，加解密算法和密钥扩展算法的S盒完全相同，因此，可以考虑在加解密算法和密钥扩展算法的硬件结构中共用同1组S盒，减少硬件资源消耗.

5 硬件电路设计

5.1 SM2算法硬件设计

SM2算法的常用硬件加速方案均是针对点运算和模运算进行硬件模块设计，并在应用层调用硬件加速模块完成SM2算法的加速，这种方式效率较高，但要实现所有的点运算和模运算模块，模逆和点乘算法的硬件实现会占用大量资源. 本文的设计方案为仅对模加、模平方和模乘3种比较简单的模运算进行硬件实现，而点乘和模逆算法通过调用这3种简单运算实现. 其结果使硬件资源的占用大幅降低，但SM2算法的运算速度也明显降低. 为此，本文对协处理器进行优化，在增加少量硬件资源条件下明显提高SM2算法的运算速度.

算法1和算法3所示的点乘运算和模逆运算均为循环结构，循环主体为占用时间最多的计算结构，在轻量级的嵌入式系统中，一般不会对循环主体采用流水线的硬件实现方式，而是通过控制单元反复调用循环主体中的计算单元，减少硬件资源开销. 然而，由于循环主体中产生的中间数据均为大位宽的数据，控制单元同样会占用较多硬件资源，本文将这部分控制逻辑在软件层实现，协处理器的通用控制单元接管了SM2算法的部分控

算法6 SM4加密算法

输入: 明文 $(X_0, X_1, X_2, X_3) \in (Z_2^{32})^4$，轮密钥 $\mathrm{rk}_i \in Z_2^{32}, i = 0, \cdots, 31$

输出: 密文 $(Y_0, Y_1, Y_2, Y_3) \in (Z_2^{32})^4$

1. FOR i = 0 UP TO 31 DO
2. $X_{i+4} = F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, \mathrm{rk}_i)$
3. $\quad = X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus \mathrm{rk}_i)$
4. $\quad = X_i \oplus L(\tau(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus \mathrm{rk}_i))$
5. 令 $B = X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus \mathrm{rk}_i = (b_0, b_1, b_2, b_3) \in (Z_2^8)^4$, 则
6. $X_{i+4} = X_i \oplus L(\tau(B)) = X_i \oplus L(\mathrm{Sbox}(b_0), \mathrm{Sbox}(b_1), \mathrm{Sbox}(b_2), \mathrm{Sbox}(b_3))$
7. 令 $C = (\mathrm{Sbox}(b_0), \mathrm{Sbox}(b_1), \mathrm{Sbox}(b_2), \mathrm{Sbox}(b_3)) \in Z_2^{32}$, 则
8. $X_{i+4} = X_i \oplus L(C) = X_i \oplus (C \oplus (C <<< 2) \oplus (C <<< 10) \oplus (C <<< 18) \oplus (C <<< 24))$
9. END FOR
10. RETURN
11. $(Y_0, Y_1, Y_2, Y_3) = R(X_{32}, X_{33}, X_{34}, X_{35}) = (X_{35}, X_{34}, X_{33}, X_{32})$

算法7 SM4密钥扩展算法

输入: 加密密钥 $\mathrm{MK} = (\mathrm{MK}_0, \mathrm{MK}_1, \mathrm{MK}_2, \mathrm{MK}_3) \in (Z_2^{32})^4$

输出: 轮密钥 $\mathrm{rk}_i \in (Z_2^{32})^4$

1. $(K_0, K_1, K_2, K_3) = (\mathrm{MK}_0 \oplus \mathrm{FK}_0, \mathrm{MK}_1 \oplus \mathrm{FK}_1, \mathrm{MK}_2 \oplus \mathrm{FK}_2, \mathrm{MK}_3 \oplus \mathrm{FK}_3)$
2. FOR i = 0 UP TO 31 DO
3. $\quad \mathrm{rk}_i = K_{i+4} = K_i \oplus T'(K_{i+1} \oplus K_{i+2} \oplus K_{i+3} \oplus \mathrm{CK}_i)$
4. $\quad = K_i \oplus L'(\tau(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus \mathrm{CK}_i))$
5. $\quad$ 令 $B = X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus \mathrm{CK}_i = (b_0, b_1, b_2, b_3) \in Z_2^{32}$, 则
6. $\quad \mathrm{rk}_i = K_i \oplus L'(\tau(\mathrm{B})) = K_i \oplus L'(\mathrm{Sbox}(b_0), \mathrm{Sbox}(b_1), \mathrm{Sbox}(b_2), \mathrm{Sbox}(b_3))$
7. $\quad$ 令 $C = (\mathrm{Sbox}(b_0), \mathrm{Sbox}(b_1), \mathrm{Sbox}(b_2), \mathrm{Sbox}(b_3)) \in Z_2^{32}$, 则
8. $\quad \mathrm{rk}_i = K_i \oplus L'(C) = K_i \oplus (C \oplus (C <<< 13) \oplus (C <<< 23))$
9. ENDFOR

注: $\mathrm{FK}_0 = (\mathrm{a3b1bac6})_{16}, \mathrm{FK}_1 = (56\mathrm{aa}3350)_{16}, \mathrm{FK}_2 = (677\mathrm{d}9197)_{16}, \mathrm{FK}_3 = (\mathrm{b}27022\mathrm{dc})_{16}, \mathrm{CK}_i = (\mathrm{ck}_{i,0}, \mathrm{ck}_{i,1}, \mathrm{ck}_{i,2}, \mathrm{ck}_{i,3}) \in (Z_2^8)^4$.

制逻辑,使整体所需的硬件资源大幅降低. 当然,这种实现方式的执行速度与纯硬件实现相比有所降低,点乘与模逆算法纯硬件实现的优势在于可以同时调用多个模运算单元实现并行计算,且中间变量的读写均在模块内部寄存器间进行,具有较高的数据交换速度. 为此,本文设计了可并行调用各运算模块的协处理器,该协处理器主要做了3点优化:一是协处理器的NICE指令执行过程采用顺序发射与派遣、乱序执行、顺序写回和反馈的实现方式,大幅提高指令执行的并行性;二是添加了8个256位的通用寄存器,大位宽的寄存器可以避免由于数据交互而花费大量的时钟周期;三是设计了独立的存储器访问单元,具有较高的存储器读写速度. 实验结果如表2和表3所示,优化后的协处理器具有更高的执行效率和资源利用率.

在协处理器的硬件结构中,本文针对SM2算法加入了2个模乘单元、1个模加单元和1个模平方单元,其中,模加和模平方单元均为简单的异或门连接而成的组合逻辑电路. 模乘单元的硬件结构如图4所示,该电路为算法2的物理映射,电路中共有2个m bit的寄存器,1个m bit的移位寄存器,$2m\times D$个与门和$(2m-1)\times D$个异或门. 当$D=1$时,该电路为完全串行展开结构,完成1次模乘运算至多需要m个时钟周期;当$D=m$时,该电路为完全并行结构,完成1次模乘运算需要1个时钟周期;当D取其他值时,该电路为串并混合结构,完成1次模乘运算最多需要$\lceil m/D\rceil$个时钟周期,其中符号"⌈ ⌉"表示向上取整. 在具体电路实现中,根据电路的时钟频率、资源面积和运算速度等条件综合考虑D的取值,一般优先考虑时钟频率,计算出D的取值范围,其次考虑资源面积,在满足资源面积约束的条件下取D的最大值,使得完成1次模乘运算花费的时钟周期更少. 经过FPGA测试,当$D=1$时,模乘单元的延时为1.02 ns,本文的时钟频率约束为100 MHz,所以D应满足$1\leqslant D<10$. 此外,为了将模乘单元消耗的Slices约束在500以内,需要满足$1\leqslant D\leqslant 6$,本文的D取最大值6,完成1次模乘运算最多需要39个时钟周期.

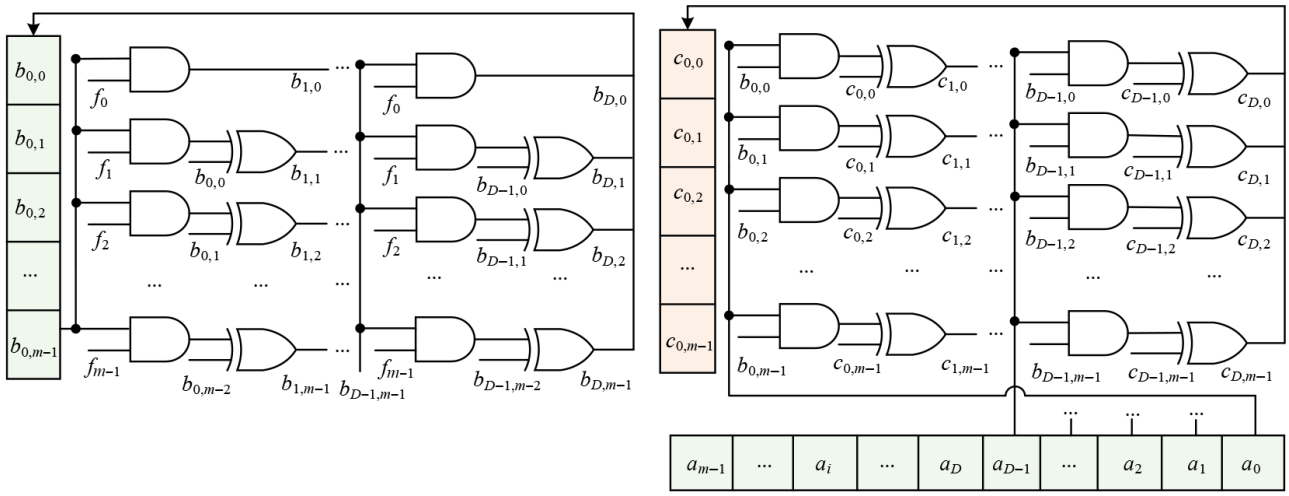


图4　模乘运算单元硬件结构图

5.2　SM3算法硬件设计

在传统的硬件电路中,SM3算法的3个部分:消息填充、消息扩展和迭代压缩均需要硬件实现,然而消息填充算法仅针对消息m的末尾进行填充,对消息m最后1个消息分组之前的所有分组均不做处理,消息填充算法的硬件实现和软件实现执行效率相差不大,因此,本文仅对消息扩展和迭代压缩模块进行硬件实现. 目前,SM3算法消息扩展和迭代压缩模块的硬件结构主要有循环结构、2合1结构、4合1结构以及流水线结构[41],本文采用占用资源最少的循环结构.

SM3算法硬件结构如图5所示,由于针对2个硬件模块的循环调用可以在软件层完成,且控制信号可以由协处理器统一生成,因此,图5中没有复杂的控制单元,减少了硬件资源消耗以及设计复杂度. 本文设计了加载消息分组和循环迭代压缩2条自定义指令,当软件完成消息分组计算并写入到协处理器中的寄存器时,首先调用加载消息分组指令将消息分组加载到消息扩展模块中的缓存中,消息扩展模块生成2个消息字W_j和W_j',之后调用循环迭代压缩指令,此时,消息扩展模块会依次生成后序的消息字,迭代压缩模块根据生成的消息字完成迭代压缩.

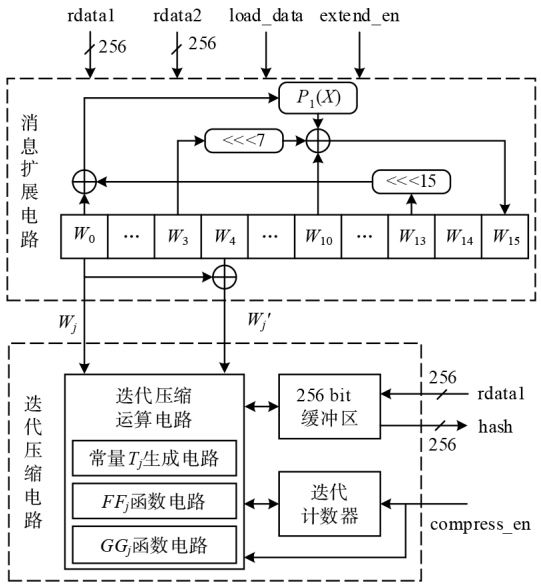


图5 SM3算法硬件结构图

5.3 SM4算法硬件设计

目前,SM4加密算法的硬件实现方式有流水线方式和循环迭代方式,本文采用占用资源更少的循环迭代方式[42],硬件电路中仅包含1轮加/解密轮函数.SM4算法的总体硬件结构如图6所示,传统的SM4算法硬件实现中有8个S盒,而本文采用分时复用的方式仅用4个S盒,进一步减小硬件资源开销,代价是每个消息分组的加解密时钟周期数加倍,由原来的32个时钟周期增加为64个时钟周期.

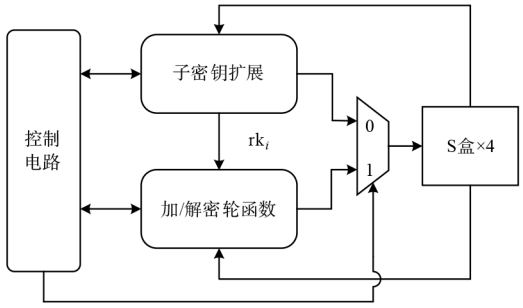


图6 SM4算法硬件结构图

SM4算法轮函数中包括异或、循环移位和S盒变换3种运算,其中,异或与循环移位占用资源较少,而传统的S盒变换实现方式——查表法会导致电路中生成较大的ROM块,因此,本文采用了复合域下的S盒实现方式.SM4的S盒变换在代数上由2次线性仿射变换和1次非线性的有限域求逆组成,如式(2)所示,x和c为GF(2^8)上的元素,$\boldsymbol{M}$为1个仅包含0和1的矩阵,如式(3)所示:

$$\boldsymbol{S}(x)=\boldsymbol{M}(\boldsymbol{M}x+c)^{-1}+c \tag{2}$$

$$\boldsymbol{M}=\begin{pmatrix}1&1&0&1&0&0&1&1\\1&1&1&0&1&0&0&1\\1&1&1&1&0&1&0&0\\0&1&1&1&1&0&1&0\\0&0&1&1&1&1&0&1\\1&0&1&1&1&1&1&0\\0&1&0&0&1&1&1&1\\1&0&1&0&0&1&1&1\end{pmatrix} \tag{3}$$

GF(2^8)上的线性仿射变换实现简单,而非线性的求逆变换实现复杂,相比之下,复合域下的求逆运算实现更简单.复合域下的S盒变换实现方式的基本思路是将GF(2^8)上的求逆运算映射为GF((2^4)^2)上的运算,而GF(2^4)上的运算又可以映射为GF((2^2)^2)上的运算,最终映射到GF(2^2)上的简单逻辑运算.

SM4密钥扩展算法的实现方式主要有2种,分别是预先计算所有轮密钥的方式和边加/解密边计算轮密钥的方式,前者占用资源较多,且目前许多应用采用1次1个会话密钥的方式,计算并保存所有轮密钥会导致效率低下.边加/解密边计算轮密钥的方式占用资源较少,资源利用率更高,适合轻量级嵌入式系统实现.SM4密钥扩展算法的轮密钥顺序和逆序生成可复用相同硬件结构,如图7所示,R_0和R_1为2个128 bit的寄存器,在空闲状态时,R_0保存$(K_{35},K_{34},K_{33},K_{32})$,$R_1$保存$(K_0,K_1,K_2,K_3)$,当进行轮密钥顺序生成时,$R_0$先置为$R_1$的值,之后每个时钟周期$R_1$置为$(K_{i-2},K_{i-1},K_i,K_{i+1})$,最终$R_1$中保存的值为$(K_{32},K_{33},K_{34},K_{35})$,此时需要将$R_1$置为$R_0$的值,再将$R_0$重新置为$(K_{35},K_{34},K_{33},K_{32})$.当进行轮密钥逆序生成时,先将$R_1$置为$R_0$的值,之后每个时钟周期$R_1$置为$(K_{i-2},K_{i-1},K_i,K_{i+1})$,最终$R_1$中保存的值为$(K_3,K_2,K_1,K_0)$,最后将$R_1$重新置为$(K_0,K_1,K_2,K_3)$.

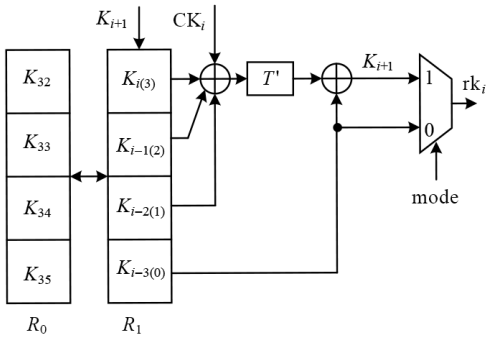


图7 SM4密钥扩展单元硬件结构图

5.4 协处理器硬件设计

本文针对3种国密算法设计了如图8所示的密码协处理器,相比于传统实现,本文主要面向面积和AT值进行优化,具有精简的硬件结构,并采用面积和功耗

较小的2级流水线的指令执行方式,运用顺序派遣、乱序执行和顺序写回的指令执行模式,提高了协处理器的执行效率.

指令流水线的第1级完成取指和译码,第2级完成执行、写回和反馈. 在取指阶段,不同于普通处理器从指令存储单元直接取指,取指单元需要通过请求通道的握手信号从主处理器中获得指令和源操作数,获得的指令和源操作数经过简单译码后被送入缓存. 在执行阶段,指令派遣单元首先从缓存中依次取出译码后的控制信号和数据,并为每条指令分配1个标签,标签值为1个单调递增的计数器值. 之后检查协处理器源操作数是否存在数据冲突,根据控制信号决定是否从寄存器堆读取256位协处理器源操作数,然后判断是否向写回队列中插入该指令的标签与写回长度信息. 最后,如果指令执行单元处于空闲状态,则启动指令执行单元. 如果指令执行结果需要写回到寄存器堆中,执行指令单元会根据写回队列判断是否轮到该指令进行写回,如果是,则将结果写入到寄存器堆中. 由于指令提交到对应执行单元后指令派遣模块不会停止工作,而是检查缓存中是否存在译码后的自定义指令,如果在不存在资源和数据冲突的条件下则可继续向执行单元提交自定义指令. 因此,不同执行单元在不存在数据冲突的情况下可并行执行,这使得本文提出的协处理器具有更强的并行处理能力. 正常来说,指令反馈单元会在指令执行结束后通过反馈通道向主处理器发起指令反馈请求,然而,这会导致协处理器失去并行处理自定义指令的能力,因为在协处理器发起反馈请求前,主处理器只会向协处理器提交1条自定义指令. 对此,本文设计了自定义指令的立即反馈执行模式,在该模式下,反馈请求会发生在取指阶段,每当取指单元获取1条自定义指令时,指令反馈单元会立即发起反馈请求,主处理器则会连续提交多条自定义指令,使协处理器同时执行多条指令.

协处理器与主处理器间的数据交换通过专门的存储器请求与反馈通道,而不是自定义指令的源操作数,这大幅提高数据交换速度,因为存储器请求与反馈通道具有连续读写主处理器内存的能力. 协处理器寄存器堆与主处理器内存的数据交换由存储器读写控制模块完成,本文设计了存储器读指令和写指令对该模块进行控制,由于这2条指令的执行流程与其他指令并无区别,因此,具有与其他执行单元并行执行的能力,进一步提高了对主处理器内存的访问速度.

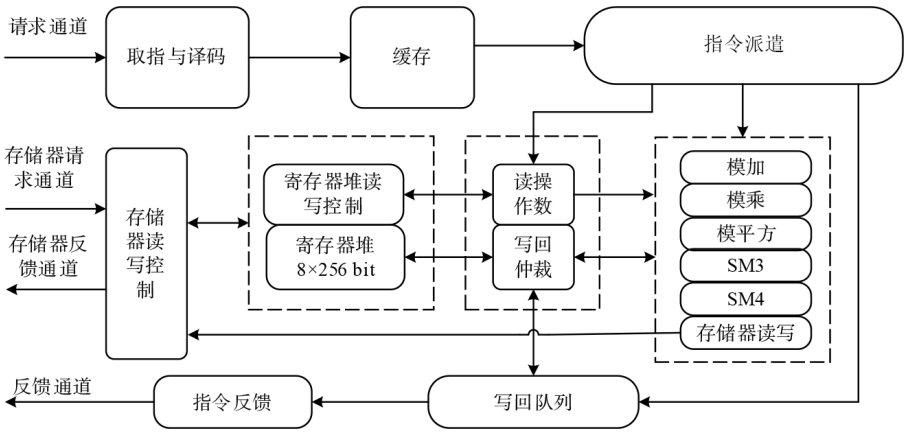


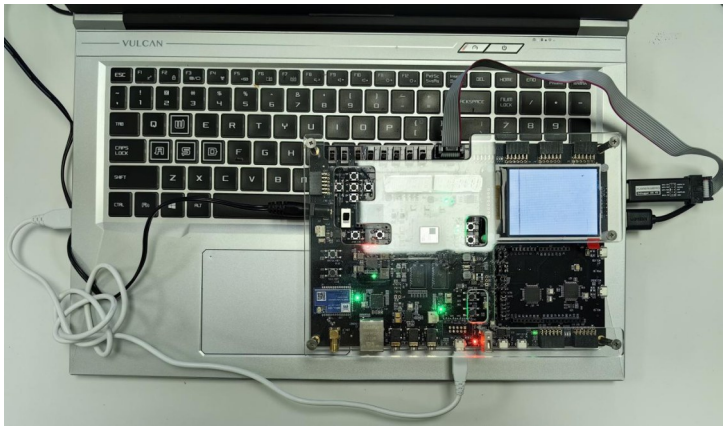
图8　密码协处理器硬件结构图

6　实验设置与结果

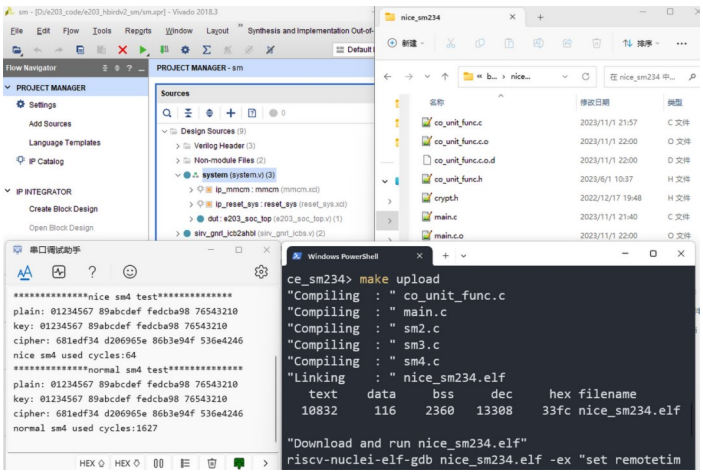
6.1　实验环境搭建

本文的实验平台包括Nuclei DDR200T开发板和PC机,Nuclei DDR200T开发板是芯来科技为其自研处理器内核IP定制的SoC原型验证硬件平台,PC机中需要安装Vivado软件、蜂鸟E203交叉编译环境和串口调试助手,并准备好蜂鸟E203的软件开发平台——HummingBird Software Development Kit(HBird SDK). Vivado软件用于将蜂鸟E203开源SoC的硬件描述语言代码综合执行后生成比特流文件,并将比特流文件写到Nuclei DDR200T开发板中. 蜂鸟E203交叉编译环境包含软件编译相关的GCC(GNU Compiler Collection)工具链,以及软件下载和调试相关的工具. HBird SDK作为衔接上层应用与底层硬件的中间平台,提供了底层硬件开发平台的应用接口以及RTOS的支持,使用户在进行应用开发时无需城市进行烦琐的寄存器配置,提高效率. 图9为本文的实验环境展示,图9(*a*)展示了Nuclei DDR200T开发板与PC端的连接,图中开发板集成的FPGA调试接口和MCU调试接口已与PC端相连,图9(*b*)展示了PC端实验环境,该图左上区域为Vivado软件界面,Vivado生成的比特流文件通过FPGA调试接口写到FPGA中,右上区域为HBird SDK文件界面,下方区域为

串口调试助手和Windows PowerShell界面，在HBird SDK文件位置打开Windows PowerShell，输入make upload命令即可进行C程序的编译、下载与调试，C程序中printf函数的打印结果可通过串口调试助手进行显示.



(*a*) DDR200T开发板连接图



(*b*) PC端实验环境图

图9 实验环境

6.2 结果分析与对比

将时钟约束为100 MHz并使用Vivado进行综合执行后，各模块的资源占用情况如表2所示，其中，模乘单元资源占用最多，本文模乘单元的参数D取6，适当减小参数的值以减小资源占用. 图10展示了各模块LUT资源占用百分比，模乘单元、SM3、SM4和存储器读写控制单元的资源占用明显高于其他模块，这4个关键运算单元和模平方单元占整个协处理器78.6%的硬件资源，寄存器堆也是必要的，其余控制单元仅占8.3%. 因此，本文所设计的协处理器具有精简的硬件结构，硬件资源利用率较高.

点乘运算是SM2中最耗时的运算，点乘运算速度常用来衡量SM2算法的实现性能. 表3为点乘算法的不同实现方案对比，图11为点乘算法的不同实现方式对比图. 表3中本文方案1为RISC-V指令集扩展实现方案，本文方案2为根据算法1进行的FPGA实现，本文的指令扩展方案的资源占用仅为纯FPGA方案的11.11%，执行速度仅下降到纯FPGA方案的41.10%，AT值下降到纯FPGA方案的27.03%，因此，本文方案1具有较高效能. 从表3中可知点乘算法的纯软件实现需

表2 协处理器各模块资源使用情况

硬件单元	LUT	FF	Slice
协处理器	4 305	2 730	1 444
模乘	1 127	709	453
SM3	1 119	917	381
存储器读写控制	595	334	344
寄存器堆	564	6	247
SM4	347	147	212
模平方	192	269	159
其他	356	348	254

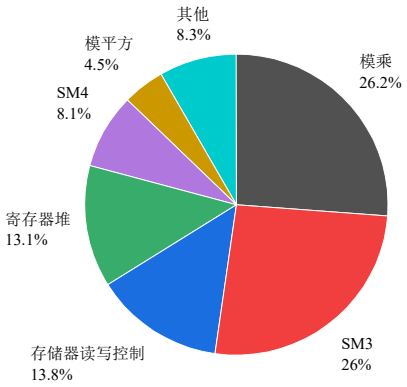


图10　协处理器各模块资源占用饼状图

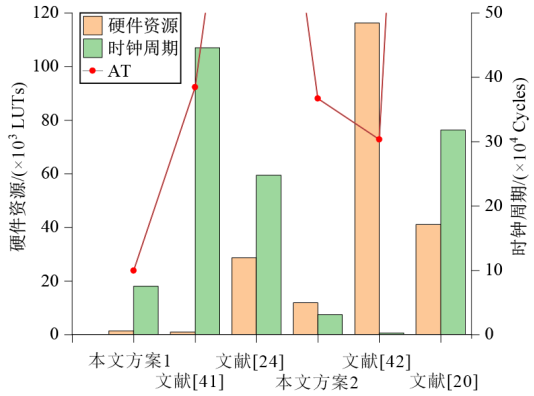


图11　点乘算法不同实现方式对比图

要消耗大量的时钟周期,不适合在时钟频率较低的轻量级嵌入式设备中实现. 文献[43]提出了灵活性高且适用性强的软硬件协同优化方案,并将该方案应用于ECC协处理器的设计,所设计的协处理器资源消耗较少,但是协处理器中没有独立的存储器读写模块,且操作数的位宽为32位,所以执行速度较慢. 文献[24]和文献[20]分别对素数域点乘算法进行协处理器和FPGA实现,然而素数域上的点乘算法比较复杂,硬件资源占用较大,不适合轻量级嵌入式系统实现. 文献[44]在FPGA上进行了点乘算法的并行化设计,具有较快的执行速度,但占用硬件资源面积较大. 相比以往研究,本文的指令扩展方案的AT值较低,在资源面积和性能上取得了较好平衡,适合轻量级嵌入式系统实现.

表3　点乘算法不同实现方式对比

方案	域	实现方式	硬件资源/LUTs	F_{max}/MHz	时钟周期/Cycles
纯软件实现	GF(2^{233})	纯软件	0	—	72 792 989
本文方案1	GF(2^{233})	软硬协同	1 319	125	75 250
文献[43]	GF(2^{163})	软硬协同	863	350	445 610
文献[24]	GF(p–256)	软硬协同	28 584	110	247 525
本文方案2	GF(2^{233})	FPGA	11 872	166	30 926
文献[44]	GF(2^{233})	FPGA	116 241	135	2 609
文献[20]	GF(p–256)	FPGA	41 067	215	318 000

表4和表5分别为SM3算法和SM4算法的自定义指令实现与其他同类文献的对比,图12和图13分别为SM3算法和SM4算法的不同实现方式对比图,表中时钟周期指每迭代压缩或加密1个消息分组所需要的时钟周期. 本文的SM3算法和SM4算法均采用消耗硬件资源较少的实现方式,且部分控制逻辑由协处理器完成,所以相较于FPGA具有更少的硬件资源消耗. 文献[25]采用较复杂的先进高性能总线(Advanced High-performance Bus,AHB)协议对SM3和SM4单元进行访问,消耗了大量硬件资源. 文献[26]基于RISC-V架构实现了SM4算法的自定义指令扩展,但并没有采用并行的协处理器架构,以及独立的内存访问通道,主处理器与协处理器间的数据传输较慢,因此,加密1个消息分组所需要的时钟周期数较多. 文献[21,23,45]和文献[42,46]分别对SM3和SM4算法进行了FPGA实现,硬件资源消耗均高于本文的设计方案,此外,本设计方案的吞吐率高于以往的协处理器设计,且AT值处于较低水平,说明本设计方案在资源面积和性能上取得较好平衡,具有很好的性价比.

7　总结与展望

为更好地在轻量级嵌入式系统中部署国产密码算法,本文基于RISC-V开源处理器完成了SM2、SM3和SM4 3种国密算法的自定义指令扩展及相关单元的硬件实现. 实验结果表明,相比传统的纯软件和纯硬件实现方式,本文提出的方案具有很好的性价比. 此外,本文在协处理器中引入了独立的内存访问模块和2级流水线结构,使性能进一步提高,同时协处理器接管了密码算法的部分控制逻辑,使硬件资源消耗进一步下降. 最后,相比以往的自定义指令集扩展实现,本文的设计具有更好的资源面积和性能优势. 本文的协处理器设计还缺少抗侧信道攻击的安全防护,后续将考虑加入对敏感数据进行掩码的硬件结构,兼顾设计的性能和安全.

表 4 SM3 算法不同实现方式对比

方案	实现方式	硬件资源/LUTs	F_{max}/MHz	时钟周期/Cycles	吞吐率/(Mbit·s^{-1})
纯软件实现	纯软件	0	—	12 618	—
本文	软硬协同	1 119	154.0	64	308.00
文献[25]	软硬协同	1 507	36.0	69	66.78
文献[23]	FPGA	2 808	286.33	64	572.66
文献[45]	FPGA	1 562	89.3	64	178.60
文献[21]	FPGA	364 400	227.0	1	29 056.00

表 5 SM4 算法不同实现方式对比

方案	实现方式	硬件资源/LUTs	F_{max}/MHz	时钟周期/Cycles	吞吐率/(Mbit·s^{-1})
纯软件实现	纯软件	0	—	1 627	—
本文	软硬协同	347	127	64	254.00
文献[26]	软硬协同	140	78	266	37.53
文献[25]	软硬协同	10 086	36	35	144.00
文献[46]	FPGA	2 312	100	34	376.47
文献[42]	FPGA	697	50	32	200.00

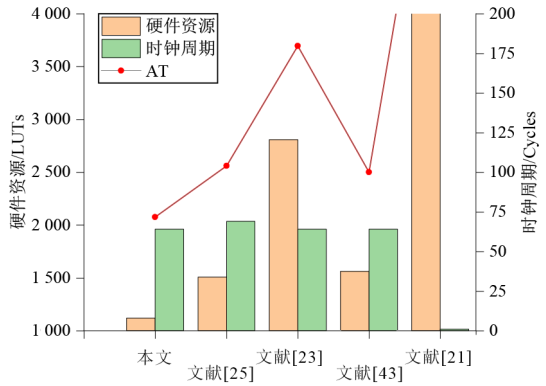


图 12 SM3 算法不同实现方式对比图

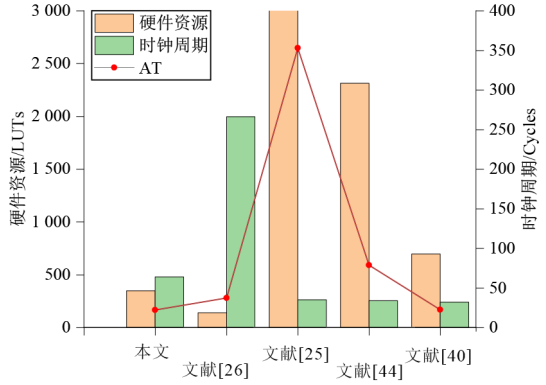


图 13 SM4 算法不同实现方式对比图

参考文献

[1] KOUICEM D E, BOUABDALLAH A, LAKHLEF H. Internet of things security: A top-down survey[J]. Computer Networks, 2018, 141(4): 199-221.

[2] OpenSSL. OpenSSL technical committee: OpenSSL project[EB/OL]. (2022-05-27)[2023-04-23]. https://www.openssl.org/.

[3] FABIAN A. Botan: Crypto and TLS for modern C++[EB/OL]. (2023-04-11) [2023-04-23]. http://botan.randombit.net/.

[4] NIIBE Y. Libgcrypt[EB/OL]. (2023-04-06) [2023-04-23]. https://www.gnupg.org/software/libgcrypt/.

[5] 刘云涛, 申泽生, 方硕, 等. 高吞吐率流水线结构的ZUC-256流密码硬件设计[J]. 电子学报, 2023, 51(2): 438-445.
LIU Y T, SHEN Z S, FANG S, et al. A hardware design of ZUC-256 stream cipher of pipelining structure with high throughput[J]. Acta Electronica Sinica, 2023, 51(2): 438-445. (in Chinese)

[6] 杜怡然, 杨萱, 戴紫彬, 等. 粗粒度可重构密码逻辑阵列智能映射算法研究[J]. 电子学报, 2020, 48(1): 101-109.
DU Y R, YANG X, DAI Z B, et al. Research on coarse-grained reconfigurable cryptographic logic array intelligent mapping algorithm[J]. Acta Electronica Sinica, 2020, 48(1): 101-109. (in Chinese)

[7] 李沐, 崔益军, 倪子颖, 等. ZUC-256流密码轻量级硬件设计与实现[J]. 数据采集与处理, 2022, 37(3): 695-702.
LI M, CUI Y J, NI Z Y, et al. Lightweight hardware design and implementations of ZUC-256 stream cipher on FPGA[J]. Journal of Data Acquisition and Processing, 2022, 37(3): 695-702. (in Chinese)

[8] 芮康康, 王成华, 范赛龙, 等. 一种高性能R-LWE格加密算法的电路结构及其FPGA实现[J]. 数据采集与处理, 2019, 34(4): 689-696.

RUI K K, WANG C H, FAN S L, et al. High performance hardware architecture of lattice-based cryptography and its FPGA implementation[J]. Journal of Data Acquisition and Processing, 2019, 34(4): 689-696. (in Chinese)

[9] NANNIPIERI P, DI MATTEO S, ZULBERTI L, et al. A RISC-V post quantum cryptography instruction set extension for number theoretic transform to speed-up CRYSTALS algorithms[J]. IEEE Access, 2021, 9: 150798-150808.

[10] HAO C, GROßSCHÄDL J, MARSHALL B, et al. RISC-V instruction set extensions for lightweight symmetric cryptography[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2022: 193-237.

[11] HADJ YOUSSEF W EL, ABDELLI A, DRIDI F, et al. An efficient lightweight cryptographic instructions set extension for IoT device security[J]. Security and Communication Networks, 2022: 9709601.

[12] PAN L H, TU G Q, LIU S B, et al. A lightweight AES coprocessor based on RISC-V custom instructions[J]. Security and Communication Networks, 2021: 9355123.

[13] LIU Z, CHOO K K R, GROSSSCHADL J. Securing edge devices in the post-quantum Internet of Things using lattice-based cryptography[J]. IEEE Communications Magazine, 2018, 56(2): 158-162.

[14] 刘畅, 武延军, 吴敬征, 等. RISC-V指令集架构研究综述[J]. 软件学报, 2021, 32(12): 3992-4024.
LIU C, WU Y J, WU J Z, et al. Survey on RISC-V system architecture research[J]. Journal of Software, 2021, 32(12): 3992-4024. (in Chinese)

[15] RAO J L, AO T Y, XU S, et al. Design exploration of SHA-3 ASIP for IoT on a 32-bit RISC-V processor[J]. IEICE Transactions on Information and Systems, 2018, E101.D(11): 2698-2705.

[16] KUNDI D E S, KHALID A, AZIZ A, et al. Resource-shared crypto-coprocessor of AES enc/dec with SHA-3[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2020, 67(12): 4869-4882.

[17] WANG W Z, HAN J, CHENG X, et al. An energy-efficient crypto-extension design for RISC-V[J]. Microelectronics Journal, 2021, 115: 105165.

[18] WANG D M, LIN Y H, HU J G, et al. FPGA implementation for elliptic curve cryptography algorithm and circuit with high efficiency and low delay for IoT applications[J]. Micromachines, 2023, 14(5): 1037.

[19] 李斌, 周清雷, 陈晓杰, 等. 可重构的素域SM2算法优化方法[J]. 通信学报, 2022, 43(3): 30-41.
LI B, ZHOU Q L, CHEN X J, et al. Optimization of reconfigurable SM2 algorithm over prime filed[J]. Journal on Communications, 2022, 43(3): 30-41. (in Chinese)

[20] 李建立, 莫燕南, 粟涛, 等. 基于国密算法SM2、SM3、SM4的高速混合加密系统硬件设计[J]. 计算机应用研究, 2022, 39(9): 2818-2825, 2831.
LI J L, MO Y N, SU T, et al. Hardware design of high-speed hybrid encryption system based on SM2, SM3 and SM4 algorithm[J]. Application Research of Computers, 2022, 39(9): 2818-2825, 2831. (in Chinese)

[21] 方轶, 丛林虎, 邓建球, 等. 基于FPGA的SM3算法快速实现方案[J]. 计算机应用与软件, 2020, 37(6): 259-262.
FANG Y, CONG L H, DENG J Q, et al. Fast implementation of SM3 algorithm based on FPGA[J]. Computer Applications and Software, 2020, 37(6): 259-262. (in Chinese)

[22] CHEN Y X, SONG J F, CHEN S, et al. Exploring the high-throughput and low-delay hardware design of SM4 on FPGA[C]//2022 19th International SoC Design Conference (ISOCC). Piscataway: IEEE, 2022: 211-212.

[23] HUANG X Y, GUO Z C, SONG M G, et al. Accelerating the SM3 hash algorithm with CPU-FPGA Co-Designed architecture[J]. IET Computers & Digital Techniques, 2021, 15(6): 427-436.

[24] 王腾飞, 张海峰, 许森. SM2专用指令协处理器设计与实现[J]. 计算机工程与应用, 2022, 58(2): 102-109.
WANG T F, ZHANG H F, XU S. Design and implementation of SM2 co-processor with specific instructions[J]. Computer Engineering and Applications, 2022, 58(2): 102-109. (in Chinese)

[25] ZHENG X, XU C Y, HU X H, et al. The software/hardware co-design and implementation of SM2/3/4 encryption/decryption and digital signature system[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(10): 2055-2066.

[26] 陈锐, 李冰, 刘向东. 基于RISC-V指令扩展的低开销SM4算法设计与实现[J]. 电子器件, 2021, 44(1): 108-113.
CHEN R, LI B, LIU X D. Design and implementation of low-cost SM4 algorithm based on RISC-V instruction set extension[J]. Chinese Journal of Electron Devices, 2021, 44(1): 108-113. (in Chinese)

[27] LING Z, YAN H Y, SHAO X H, et al. Secure boot, trusted boot and remote attestation for ARM TrustZone-based

IoT Nodes[J]. Journal of Systems Architecture, 2021, 119: 102240.

[28] MENG Y F, LI J Z. Data sharing mechanism of sensors and actuators of industrial IoT based on blockchain-assisted identity-based cryptography[J]. Sensors, 2021, 21(18): 6084.

[29] HU X H, ZHENG X, ZHANG S S, et al. A high-performance elliptic curve cryptographic processor of SM2 over GF(p)[J]. Electronics, 2019, 8(4): 431.

[30] LIAO K, CUI X X, LIAO N, et al. High-performance noninvasive side-channel attack resistant ECC coprocessor for GF(2^m)[J]. IEEE Transactions on Industrial Electronics, 2017, 64(1): 727-738.

[31] 国家密码算法局. SM2椭圆曲线公钥密码算法[EB/OL]. (2010-12-17)[2023-04-23]. https://www.oscca.gov.cn/sca/xxgk/2010-12/17/1002386/files/b791a9f908bb4803875ab6aeeb7b4e03.pdf.
State Cryptography Administration.Public Key Cryptographic Algorithm SM2 Based on Elliptic Curves[EB/OL]. (2010-12-17) [2023-04-23]. https://www. oscca. gov. cn/sca/xxgk/2010-12/17/1002386/files/b791a9f908bb4803875ab6aeeb7b4e03.pdf. (in Chinese)

[32] 国家密码算法局. SM3密码杂凑算法[EB/OL]. (2010-12-17)[2023-04-23]. https://www.oscca.gov.cn/sca/xxgk/2010-12/17/1002389/files/302a3ada057c4a73830536d03e683110.pdf.
State Cryptography Administration. SM3 Cryptographic Hash Algorithm[EB/OL]. (2010-12-17)[2023-04-23]. https://www.oscca.gov.cn/sca/xxgk/2010-12/17/1002389/files/302a3ada057c4a73830536d03e683110. pdf. (in Chinese)

[33] 国家密码算法局. 无线局域网产品使用的SMS4密码算法[EB/OL]. (2016-11-18)[2023-04-23]. https://www.oscca.gov.cn/sca/c100061/201611/1002423/files/330480f731f64e1ea75138211ea0dc27.pdf.
State Cryptography Administration. SMS4 cryptographic algorithm used in wireless LAN products[EB/OL]. (2016-11-18). [2023-04-23]. https://www.sca.gov.cn/sca/c100061/201611/1002423/files/330480f731f64e1ea75138211ea0dc27.pdf. (in Chinese)

[34] Nuclei. Hummingbirdv2 E203 Core and SoC[EB/OL]. (2022-10-27)[2023-04-23]. https://doc.nucleisys.com/hbirdv2.

[35] 胡振波. 手把手教你RISC-V CPU(上)处理器设计[M]. 北京: 人民邮电出版社, 2021: 282-289.
HU Z B. Hands-on RISC-V CPU Processor Design (book one)[M]. Beijing: The People's Posts and Telecommunications Press, 2021: 282-289. (in Chinese)

[36] 车光宁, 张钊锋. GF(2^m)域上的低功耗可配置ECC点乘算法ASIC设计实现[J]. 微电子学与计算机, 2018, 35(1): 15-20.
CHE G N, ZHANG Z F. Low power configurable ASIC based elliptic curve scalar multiplication over GF(2^m)[J]. Microelectronics & Computer, 2018, 35(1): 15-20. (in Chinese)

[37] 刘金龙, 张玉婷, 王尧. GF(2^m)域ECC点乘算法优化设计[J]. 通信技术, 2020, 53(6): 1488-1494.
LIU J L, ZHANG Y T, WANG Y. Optimal design of ECC point multiplication algorithm over GF(2^m)[J]. Communications Technology, 2020, 53(6): 1488-1494. (in Chinese)

[38] AWALUDIN A M, LARASATI H T, KIM H. High-speed and unified ECC processor for generic weierstrass curves over GF(p) on FPGA[J]. Sensors, 2021, 21(4): 1451.

[39] LI Y, MA X P, ZHANG Y, et al. Mastrovito form of non-recursive karatsuba multiplier for all trinomials[J]. IEEE Transactions on Computers, 2017, 66(9): 1573-1584.

[40] HEIDARPUR M, MIRHASSANI M. An efficient and high-speed overlap-free karatsuba-based finite-field multiplier for FGPA implementation[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2021, 29(4): 667-676.

[41] 苗佳. 杂凑算法SM3/SHA256/SHA3的硬件设计与实现[D]. 北京: 清华大学, 2018.
MIAO J. Hardware Design and Implementation of Secure Hash Algorithms SM3/SHA256/SHA3[D]. Beijing: Tsinghua University, 2018. (in Chinese)

[42] 何诗洋, 李晖, 李凤华. SM4算法的FPGA优化实现方法[J]. 西安电子科技大学学报, 2021, 48(3): 155-162.
HE S Y, LI H, LI F H. Optimization and implementation of the SM4 on FPGA[J]. Journal of Xidian University, 2021, 48(3): 155-162. (in Chinese)

[43] 夏辉, 于佳, 秦尧, 等. 嵌入式领域ECC专用指令处理器的研究[J]. 计算机学报, 2017, 40(5): 1092-1108.
XIA H, YU J, QIN Y, et al. The researches on the ASIP of ECC in embedded domain[J]. Chinese Journal of Computers, 2017, 40(5): 1092-1108. (in Chinese)

[44] ZHAO X, LI B, ZHANG L, et al. FPGA implementation of high-efficiency ECC point multiplication circuit[J]. Electronics, 2021, 10(11): 1252.

[45] 于永鹏, 严迎建, 李伟. SM3算法高速ASIC设计及实现[J]. 微电子学与计算机, 2016, 33(4): 21-26.
YU Y P, YAN Y J, LI W. High speed ASIC design and implementation of SM3 algorithm[J]. Microelectronics & Computer, 2016, 33(4): 21-26. (in Chinese)

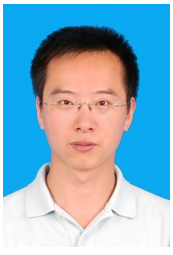
[46] 刘金峒, 梁科, 王锦, 等. SM4加密算法可裁剪式结构设计与硬件实现[J]. 南开大学学报(自然科学版), 2019, 52(4): 41-45.
LIU J T, LIANG K, WANG J, et al. Cuttable structure design and hardware implementation of SM4 encryption algorithm[J]. Acta Scientiarum Naturalium Universitatis Nankaiensis, 2019, 52(4): 41-45. (in Chinese)

作者简介

王明登　男,1998年10月出生,陕西汉中人. 信息工程大学密码工程学院硕士研究生. 主要研究方向为密码芯片设计与安全防护技术.
E-mail: wmd_work@163.com

严迎建　男,1973年1月出生,河南扶沟人. 信息工程大学密码工程学院教授. 主要研究方向为密码芯片设计与安全防护技术.
E-mail: yanyingjian@163.com

郭朋飞　男,1987年2月出生,河南漯河人. 信息工程大学密码工程学院讲师. 主要研究方向为计算机体系结构、密码SoC设计和微架构攻击.
E-mail: pfguo_work@126.com



张　帆　男,1978年10月出生,浙江杭州人. 浙江大学网络空间安全学院教授. 主要研究方向为网络空间安全、硬件安全和旁路攻击与防护.
E-mail: fanzhang@zju.edu.cn