计算机科学
COMPUTER SCIENCE

基于FPGA的ZUC高性能数据加密方案

张博林, 李斌, 燕云飞, 魏源鑫, 周清雷

引用本文

张博林, 李斌, 燕云飞, 魏源鑫, 周清雷. 基于FPGA的ZUC高性能数据加密方案[J]. 计算机科学, 2023, 50(11): 374-382.

ZHANG Bolin, LI Bin, YAN Yunfei, WEI Yuanxin, ZHOU Qinglei. ZUC High Performance Data Encryption Scheme Based on FPGA[J]. Computer Science, 2023, 50(11): 374-382.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.221100070

基于 FPGA 的 ZUC 高性能数据加密方案

张博林[1,2]　李　斌[1,2]　燕云飞[1]　魏源鑫[1]　周清雷[1]
1 郑州大学计算机与人工智能学院　郑州 450001
2 河南省网络密码技术重点实验室　郑州 450001
(421049800@qq.com)

摘　要　祖冲之(ZUC)算法是我国自主研制的流密码算法,现已被 3GPP LTE 采用为第四代移动通信加密标准。为适应大数据时代对于国产密码性能的高要求,设计了一套以祖冲之算法为核心的高性能数据加密方案。该方案中包含两种不同结构形式的加密算法核心,分别针对短报文和长报文两种不同的应用情形,基于 FPGA 平台,采用 CLA 和 CSA 加法器设计了半流水线和全流水线形式的 ZUC 流密码电路结构,以改进的 ZUC 加密模式,配合高速内存通信和多 iv 并行加密,实现了高性能加密方案,极大提高了加解密效率。该方案工作时,可使用控制模块来配置加密算法。实验结果表明,与其他方案相比,所提方案的算核工作频率分别提高了 40.8%～209.5%和 62.1%～445.4%,数据吞吐率达到了 25.728 Gb/s 和 46.08 Gb/s,适用于边缘设备、车联网数据加密等高性能加密场景。

关键词:祖冲之算法;现场可编程门阵列;高性能加密;硬件实现;流水线
中图法分类号　TP309

ZUC High Performance Data Encryption Scheme Based on FPGA

ZHANG Bolin[1,2], LI Bin[1,2], YAN Yunfei[1], WEI Yuanxin[1] and ZHOU Qinglei[1]
1 Country School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou 450001, China
2 Henan Key Laboratory of Network Cryptography Technology, Zhengzhou 450001, China

Abstract　ZUC algorithm is a stream cipher algorithm independently developed by China, which has been adopted as the fourth generation mobile communication encryption standard by 3GPP LTE. In order to meet the high requirements of the big data era for the performance of domestic passwords, a set of high-performance data encryption scheme with ZUC algorithm as the core is designed. The scheme includes two encryption algorithm cores of different structure forms. Aiming at two different application situations of short message and long message respectively, based on the FPGA platform, the semi-pipelined and full-pipelined ZUC stream cipher circuit structures are designed by using CLA and CSA adders. With the improved ZUC encryption mode, combined with high-speed memory communication and multi iv parallel encryption, the high-performance encryption scheme is realized, which greatly improves the encryption and decryption efficiency. When the scheme works, the encryption algorithm can be configured using the control module. Experimental results show that, compared with other schemes, the working frequency of the proposed algorithmis increased by 40.8%～209.5% and 62.1%～445.4% respectively, and the data throughput reaches 25.728 Gb/s and 46.08 Gb/s, meeting the high-performance encryption scenarios such as edge devices and Internet of Vehicles data encryption.

Keywords　ZUC algorithm, FPGA, High performance encryption, Hardware implementation, Pipelined

1　引言

随着云计算、雾计算、区块链、大数据等信息技术问世,人们日常的生产生活越来越依赖各种信息数据的传递。在信息技术的支持下,各种相关的信息产业也蓬勃发展。信息产业中要应用的各种数据含有大量涉及个人隐私以及其他有价值的信息,因此大数据的机密性、安全性以及隐私保护等安全问题受到了广泛的关注。但在物联网、车联网、远程医疗、加密视频实时传输等对于传输时延要求严格的场景下,需要保证安全性与传输效率的平衡。因此,如何在保证数据机密性的前提下进行高速加密是一个亟待解决的问题。

ZUC 算法(即 ZUC-128 流密码算法)是 3GPP 机密性算法 EEA3 和完整性算法 EIA3 的核心,是我国自主设计的加密算法。ZUC 算法是我国第一个成为国际密码标准的密码

到稿日期:2022-11-09　返修日期:2023-03-29
基金项目:河南省网络密码技术重点实验室研究课题(LNCT2022-A14);国家重点研发计划(2018XXXXXXX01);河南省科技攻关(232102211055)
This work was supported by the Henan Key Laboratory of Network Cryptography Technology(LNCT2022-A14), National Key R&D Program of China(2018XXXXXXX01) and Key Scientific and Technological Project of Henan Province(232102211055).
通信作者:李斌(iebinli@zzu.edu.cn)

算法。ZUC 算法是一种流密码算法，与分组密码相比，具有加解密速度快、适合硬件实现等特点。随着集成电路技术的高速发展，硬件电路实现密码算法与传统加密方法相比具有安全性高、计算速度快、成本低、性能可靠等优点。现场可编程门阵列(Field Programmable Gate Array，FPGA)器件因其可编程和成本低等特性在密码算法实现领域具有突出优势。当前我国自研密码已经在各种信息系统中被广泛应用[1]，而我国自研密码的性能是否可以适应当前蓬勃发展的信息产业的要求是一个至关重要的问题。

目前，国内外已经有许多学者针对 ZUC 算法分别基于硬件和软件进行了实现、优化与应用。在硬件方面，Liu 等[2]首次在硬件上实现了 ZUC 算法。Paris 等[3]于 FPGA 上利用优化关键路径的方式使得算法的工作性能获得提升。Zhang 等[4]同样基于 FPGA，通过复用 S 盒以及改造加法器链的方式使得吞吐量达到 5.504Gbps。Guo 等[5]在 FPGA 上通过减少加法器链上不必要的加法器的方式，对 ZUC 算法的面积进行了优化，同时吞吐量达到 5.647 Gbps。Liu 等[6]通过优化加法器链及采用改进的模加器使得 ZUC 算法在 FPGA 上的工作性能突破了 200 MHz。Lu 等[7]采用超前进位加法器(Carry Lookahead Adder，CLA)设计加法器链以及在复合域上构造 S 盒使得 ZUC 算法在 FPGA 上的工作频率达到了 235MHz，并且资源开销大大降低。Zhou 等[8]在 FPGA 平台 Zynq-7020 上实现了以 ZUC 算法为核心的视频加密算法的应用，工作频率为 142.7 MHz。在软件方面，Zhang 等[9]在 Inter i7 处理器上利用优化函数调用、编译器优化、延迟取模与合并 S 盒的方式优化实现了 ZUC 算法的软件形式，吞吐量达到 4.22 Gb/s。Bai 等[10]在 X86 处理器上通过 SIMD 技术并行实现了 ZUC-256 算法，吞吐量最高达到 21.1 Gbps。

虽然 ZUC 算法已经取得一定成果，但仍有优化前景：1)现有的 ZUC 加密算法设计在应用场景方面仍有扩展的可能；2)为满足国密算法在高性能场景中的应用，ZUC 算法在吞吐量、工作频率等方面仍有待提高；3)ZUC 算法的软件实现无法满足资源受限情境的高速加密。

针对以上问题，本文设计了一套以 ZUC 算法为核心的高性能加密方案，利用改进的 ZUC 加密模式，通过高速内存通信和多 iv 并行加密的方式实现了高性能加密方案。该方案中有一个包含两种不同设计的 ZUC 算法的加密算法模块。第一种 ZUC 算法模块为全流水线设计，通过采用进位保留加法器(Carry Save Adder，CSA)重新设计加法器链，并通过将各个轮次设计为独立的模块来实现全流水线设计。第二种为半流水线设计，通过复用单轮模块，采用进位超前加法器(Carry Look-Ahead Adder，CLA)设计模加器，并基于并行模加器设计加法器链，最终实现半流水线设计。

2 ZUC 算法的原理

ZUC 算法是一种面向字的流密码算法，需要输入一个 128 bit 的初始密钥 key 和一个 128 位的初始向量 iv，输出一串以 32 bit 为单位的密钥流[11]。生成的密钥流既可以用于加密，也可以用于解密。

2.1 ZUC 算法的整体结构

如图 1 所示，ZUC 算法分为 3 个功能模块，顶部是一个线性反馈移位寄存器(Linear KaFeedback Shift Register，LFSR)，中部是比特重组(Bit Reorganization，BR)部件，底部是一个非线性函数(Nonlinear Function，F)部件。

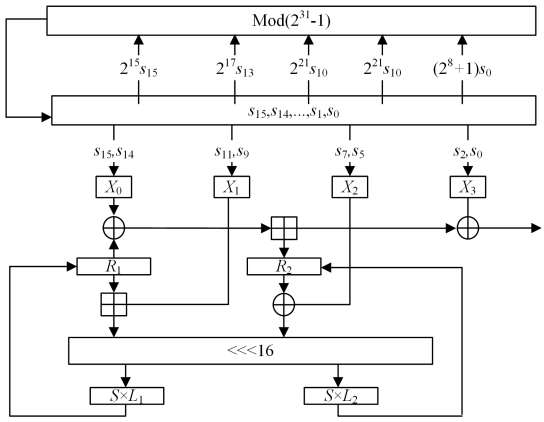

图 1 ZUC 算法结构

Fig. 1 ZUC algorithm structure

2.2 线性反馈移位寄存器

线性反馈移位寄存器(LFSR)部件中有 16 个 31 bit 的寄存器单元(s_0，s_1，…，s_{15})，寄存器中的值为[1，$2^{31}-1$]中的整数[12]。LFSR 有两个不同的工作阶段，分别为初始化阶段和工作阶段。

1)初始化阶段：在初始化阶段时，LFSR 接收 31 bit 的输入 u，u 是通过非线性函数输出的 32 bit 字 W 的最低位得到的。即 $u=W>>>1$。

(1)计算 $v=2^{15}s_{15}+2^{17}s_{13}+2^{21}s_{10}+(1+2^8) \bmod (2^{31}-1)$ 得到变量 v；

(2)计算 $s_{16}=(v+u) \bmod (2^{31}-1)$，若计算结果 s_{16} 为 0，则将其置为 $2^{31}-1$。

(3)将 s_1，s_2，…，s_{16} 依次存入 LFSR 中的 s_0，s_1，…，s_{15} 等寄存器单元。

2)工作阶段：LFSR 器件不需要接收输入。

(1)计算 $s_{16}=2^{15}s_{15}+2^{17}s_{13}+2^{21}s_{10}+(1+2^8) \bmod (2^{31}-1)$；若计算结果为 0，则将其置为 $2^{31}-1$。

(2)将 s_1，s_2，…，s_{16} 依次存入 LFSR 中的 s_0，s_1，…，s_{15} 等寄存器单元。

2.3 比特重组模块

本模块从 LFSR 中的寄存器单元中选取 128bit，组成 4 个 32 bit 的字。将其中 3 个字作为非线性函数模块的输入，另外 1 个字用于 ZUC 算法密钥流的生成。

比特重组模块使用 LFSR 中的 8 个寄存器单元，以式(1)－式(4)的形式生成 4 个 32 bit 的字。

$$X_0=s_{15H} \parallel s_{14L} \tag{1}$$

$$X_1=s_{11L} \parallel s_{9H} \tag{2}$$

$$X_2=s_{7L} \parallel s_{5H} \tag{3}$$

$$X_3=s_{2L} \parallel s_{0H} \tag{4}$$

2.4 非线性函数模块

非线性函数(F)模块需要输入 BR 中生成的 X_0，X_1 和 X_2。本模块中还设有 2 个 32 bit 的寄存器 R_1 和 R_2。在算法首次进入初始化状态时，R_1 和 R_2 需要全部置 0。F 模块经过计算后输出一个 32 bit 的 W，F 模块的工作步骤如式(5)－式(9)所示：

$$W=((X_0 \oplus R_1)+R_2) \bmod 2^{32} \tag{5}$$

$$W_1=(R_1+X_1) \bmod 2^{32} \tag{6}$$

$$W_2=R_2 \oplus X_2 \tag{7}$$

$$R_1 = S(L_1(W_{1L} \| W_{2H})) \tag{8}$$

$$R_2 = S(L_2(W_{2L} \| W_{1H})) \tag{9}$$

其中，S 为 32×32 的 S 盒；L_1 和 L_2 为非线性变换，变换形式如式(10)和式(11)所示：

$$L_1(X) = X \oplus (X <<< 2) \oplus (X <<< 10) \oplus (X <<< 18) \oplus (X <<< 24) \tag{10}$$

$$L_2(X) = X \oplus (X <<< 8) \oplus (X <<< 14) \oplus (X <<< 22) \oplus (X <<< 30) \tag{11}$$

2.5 ZUC 算法的工作流程

ZUC 算法每次运算需先经过密钥加载与 LFSR 寄存器初始化，再经过 32 次初始化阶段的计算与 1 次丢弃结果的工作阶段的计算，之后转入工作阶段，正式开始输出密钥。

2.5.1 密钥加载与 LFSR 寄存器初始化

在 ZUC 算法进行初次初始化阶段之前，需要进行 LFSR 寄存器初始化与密钥加载。密钥加载，也就是将输入的初始密钥 k 与初始向量 $\boldsymbol{iv}$ 扩展为 16 个 31bit 字段[13]。ZUC 算法的初始密钥 k 与初始向量 $\boldsymbol{iv}$ 的长度均为 128 bit，k 与 $\boldsymbol{iv}$ 的表达式如式(12)和式(13)所示：

$$k = k_0 \| k_1 \| k_2 \| \cdots \| k_{15} \tag{12}$$

$$\boldsymbol{iv} = \boldsymbol{iv}_0 \| \boldsymbol{iv}_1 \| \cdots \| \boldsymbol{iv}_{15} \tag{13}$$

在输入初始密钥与向量后，需要变换输入的初始向量 $\boldsymbol{iv}$，需要输入参数 $COUNT$(32 bit)，$BEARER$(5 bit)，$DIRECTION$(1 bit)[13]。以上参数在算法开始工作之前设置完毕。

记 $COUNT$ 如式(14)所示：

$$COUNT = COUNT[0] \| COUNT[1] \| COUNT[2] \| COUNT[3] \tag{14}$$

$\boldsymbol{iv}$ 的变换如式(15)－式(18)所示：

$$\boldsymbol{iv}_i = COUNT[i], 0 \leqslant i \leqslant 3 \tag{15}$$

$$\boldsymbol{iv}_4 = BEARER \| DIRECTION \| 00_2 \tag{16}$$

$$\boldsymbol{iv}_{5+i} = 00000000_2, 0 \leqslant i \leqslant 2 \tag{17}$$

$$\boldsymbol{iv}_{8+i} = \boldsymbol{iv}_i, 0 \leqslant i \leqslant 7 \tag{18}$$

随后在寄存器初始化的过程中，需引入一个由 16 个 15 bit 子串构成的 240 bit 的常量字符串，具体数值为文献[11]中所描述的数值，如式(19)所示：

$$D = d_0 \| d_1 \| d_2 \| \cdots \| d_{15} \tag{19}$$

加载到 LFSR 寄存器中的 s_i($0 \leqslant i \leqslant 15$)如式(20)所示：

$$s_i = k_i \| d_i \| \boldsymbol{iv}_i \tag{20}$$

2.5.2 ZUC 算法的工作阶段

在 ZUC 算法中有两种不同的工作阶段，分别为初始化阶段和工作阶段，而工作阶段也根据丢弃结果与否分为两种。以下是对 ZUC 算法两种工作阶段的描述：

1)初始化阶段

(1)比特重组，计算 X_0，X_1，X_2，X_3。

(2)非线性函数计算 R_1，R_2，W。

(3)LFSR 的初始化阶段计算，更新 LFSR 中的寄存器。

2)工作阶段

(1)比特重组，计算 X_0，X_1，X_2，X_3。

(2)非线性函数计算 R_1，R_2，W。同时，若算法首次进入工作轮次，则丢弃 W，否则按照式(21)输出密钥。

$$Z = W \oplus X_3 \tag{21}$$

(3)LFSR 的工作阶段计算，更新 LFSR 中的寄存器。

3 方案的总体设计

本方案采用松耦合结构，分模块对方案各组件进行设计，主要包括 XDMA、MIG DDR、核心控制模块、内存读写模块、核心算法模块，方案的总体结构如图 2 所示。

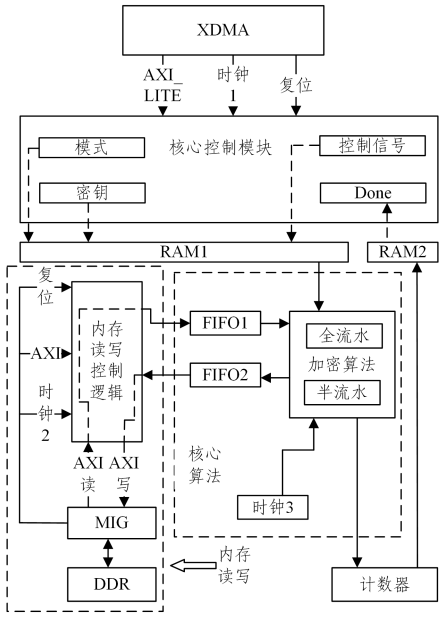

图 2 总体结构图

Fig. 2 Overall structure

各模块的功能如下：

XDMA：FPGA 主板与 CPU 之间进行传输的数据接口 IP 核，信息传输使用 AXI_Lite 接口。

MIG DDR：MIG 为 FPGA 与 DDR(Double Data Rate Synchronous Dynamic Random Access Memory，双倍速率同步动态随机存储器，也即内存)进行数据传输的接口 IP 核，通过 AXI_4 接口相互连接。

核心控制模块：根据外部设备从 XDMA 部件传入的配置信息控制核心算法模块和相关部件的工作，并利用 AXI_Lite 接口控制数据流的传输。图 2 中核心控制模块内有 2 组配置信息与 2 个控制信号。2 组配置信息为密钥(初始密钥与初始向量)和模式(加密/解密)。2 个控制信号为控制信号(用于控制核心算法部件运行状态的信号)和 Done(反馈给外设算法工作状态的信号)。

内存读写模块：用于控制待处理数据的读取和经过处理的数据的写入。数据读写的规则为：从 0 地址开始，工作到 1 GB 地址为止；之后再从 0 地址开始工作，循环往复。

加密算法模块：本模块以多 $\boldsymbol{iv}$ 并行加密的 ZUC 算法为核心设计，有半流水线与全流水线两套方案可供选择。

其他部件：计数器，用于控制 Done 信号，每处理 512MB 的数据将 Done 信号置 1；FIFO1 与 FIFO2，用于进行数据位宽转换并实现时钟域隔离；RAM1 与 RAM2，用于实现时钟域的隔离。

下面针对其中主要模块的工作流程与电路设计进行说明。

3.1 核心控制模块

图 2 详细地描述了核心控制模块的电路，因此本节仅对核心控制模块控制整个系统的工作流程进行描述。首先外部

设备传输数据到 DDR，并通过 XDMA 向控制模块配置参数与控制信号。随后控制模块通过控制信号向核心算法发出启动命令，并将密钥通过 RAM1 传输给核心算法。随后，核心算法从 DDR 中读入数据并由核心算法模块进行数据处理，随后将处理过的数据写入 DDR。每当核心算法模块处理 128 bit 数据后，计数器就加 1。当核心算法模块处理完 512 MB 数据后，通过计数器模块将 Done 置为 1 的方式通知控制模块，控制模块检测到后将 Done 置 0，并通过 XDMA 向外部设备发出读取新 512 MB 数据的命令。执行此循环直到数据终止。最后当工作需要结束时，将核心控制模块的控制信号置为全 0。

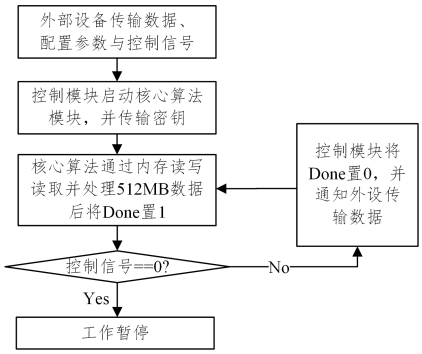

图 3 核心控制模块的控制流程图

Fig. 3 Flow of core control module

3.2 内存读写

由于数据加密的计算复杂度较高，且高性能加密对工作速率的要求也相当高，因此需将待处理数据暂存入内存中，防止数据传输与数据处理不匹配造成加密数据的拥塞和丢失。若要实现 FPGA 与 DDR 之间的高速数据交换，就需要采用 AXI_4 协议[14]。AXI_4 通信协议支持突发式传输，即在一次数据传输中，可连续获取数据[15]。在本方案中设置突发长度为 16，数据宽度为 256 位，即一次传输数据量为 512 字节，可以满足数据通信要求。FPGA 与 DDR 之间通过 MIG 进行通信连接，实现高速的数据传输，内存读写的整体结构如图 4 所示。

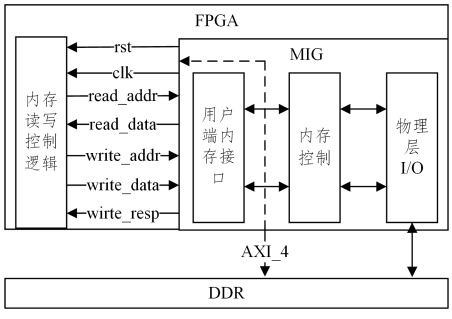

图 4 内存读写结构

Fig. 4 Memory read/write structure

图 4 中的内存读写控制逻辑通过 AXI_4 协议与 MIG 的用户端内存接口连接，内存读写控制逻辑控制的信号主要包括读地址(read_addr)、读数据(read_data)、写地址(write_addr)、写数据(write_data)、写响应(write_resp)。内存读写的模式有两种，分别是写数据和读数据。其中写数据的工作流程为：MIG 端发出 rst 信号，将内存控制逻辑重置并启动，随后用户控制逻辑端向 MIG 发出写命令，并传递写数据的起始地址(write_addr)以及相关控制信号，同时通过 write_data 与 MIG 向 DDR 写入待处理数据，写入完毕后向用户控制逻辑端传递写响应(write_resp)信号。读数据的工作流程为：MIG 端向内存控制逻辑发出 rst 信号，将控制逻辑重置(与写数据相关控制信号同时置 0)，随后控制逻辑端向 MIG 发出读命令，并传递读数据的起始地址(read_addr)以及相关控制信号，之后 DDR 按照 MIG 传递的控制信息通过 read_data 来读取数据，读取完毕后向 MIG 传递读取完毕信号。

3.3 核心算法

在实际应用的 ZUC 硬件算法实现方面，文献[8]采用 ZUC 算法单核加密，其运算速率较难满足高性能加密的要求；在软件实现方面，文献[10]使用多 ***iv*** 或多密钥的并行方式提升算法在软件上的工作效率。因此，本方案设置多 ZUC 模块，利用多 ***iv*** 并行加密来提升系统的工作效率，同时保证安全性以及更充分地利用板上资源。本方案在每个加密算法中设置了 4 个 ZUC 算法核心，在进入工作阶段后，每个时钟可以输出 128 位密钥。为保证生成密钥的安全性，需对算法模式中向量变换的过程进行改进，本方案规定 4 个模块使用的 COUNT 依次加 1。本节中 ZUC 算法模块的设置与多 ***iv*** 并行输入的方式如图 5 所示。

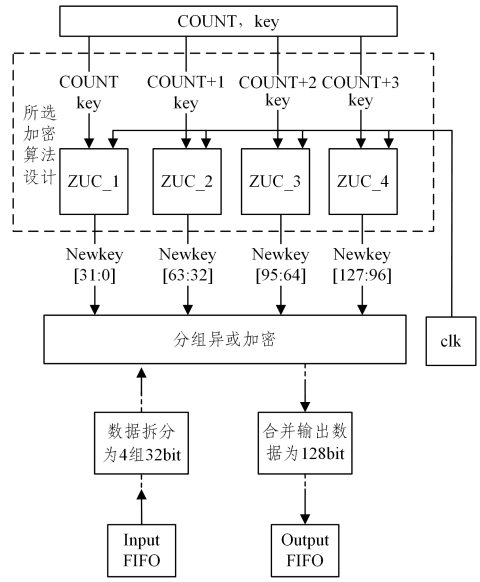

图 5 核心算法

Fig. 5 Main algorithm module

4 ZUC 模块设计

针对不同的情景应用，本文设计了两种不同的加密算法，一个为 ZUC 算法的全流水线设计，另一个为半流水线设计。包含全流水线设计的加密算法的工作方式为，每输入 128 bit 初始密钥，则将生成 128 bit 加密密钥。半流水线设计的工作方式则是输入 128 bit 初始密钥就可生成 128 bit 的加密密钥流。方案运行中，可以通过核心控制模块控制信号的配置来控制运行核心的选择。算法优化的主要目标就是优化关键路径的运行性能。在 ZUC 算法中关键路径的优化主要与 LFSR 中下一轮生成的 s_{16} 计算密切相关[16]。因此，我们针对 LFSR 中的模加运算链方向进行优化。下面对两种有不同设计的 ZUC 算法模块进行说明。

4.1 ZUC 算法全流水线设计

全流水线设计将 ZUC 算法的 34 个轮次设计为独立的模块，并在各独立模块之间插入寄存器，以实现流水线

化设计。在各独立模块内设置寄存器组，实现模块流水线化。整体结构如图 6 所示。

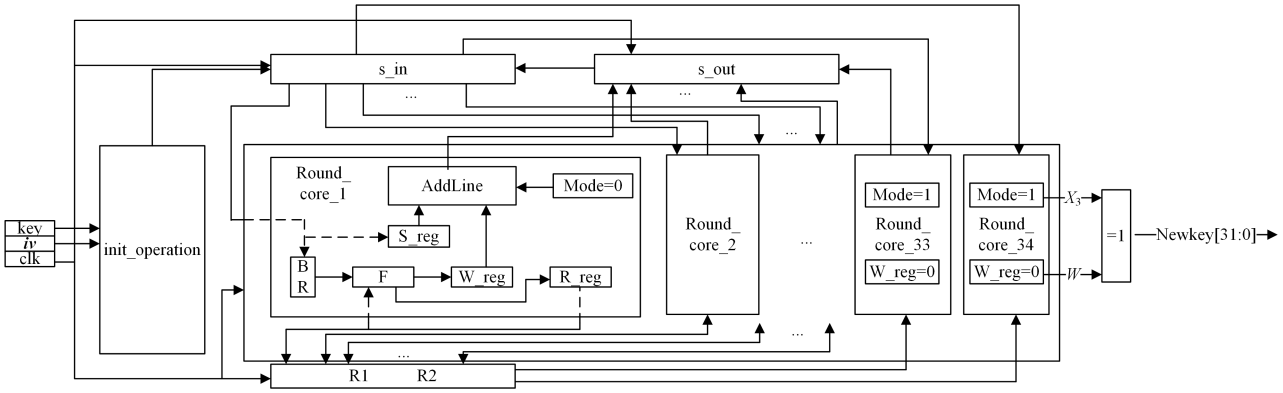

图 6 全流水线设计

Fig. 6 Full pipelined architecture

下面按照 ZUC 算法的工作流程对各轮次中的主要部件进行说明。

4.1.1 密钥加载与 LFSR 寄存器初始化实现

在设计中该过程被封装为 init_operation 模块，通过组合逻辑的方式进行向量变换，最终生成 LFSR 寄存器的初始内容。init_operation 模块的结构如图 7 所示，其中 BEARER 与 DIRECTION 声明为全 0。

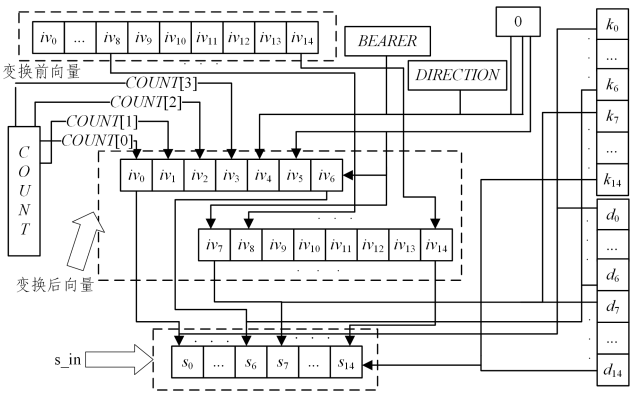

图 7 init_operation 模块结构

Fig. 7 init_operation module architecture

4.1.2 比特重组模块实现

比特重组也是一个纯组合逻辑实现的电路结构，它通过抽取 LFSR 寄存器中的内容形成 4 个 32bit 字，在设计中被封装为 BR 模块。BR 模块电路的实现如图 8 所示。

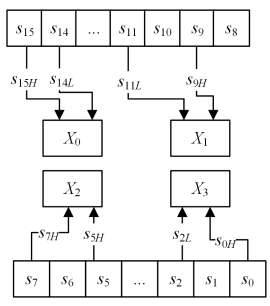

图 8 BR 模块电路实现

Fig. 8 BR module circuit implementation

4.1.3 非线性函数模块实现

在非线性函数模块的实现中，如文献[4]和文献[6]利用 RAM 来实现 S 盒的查表。但本文将各个轮次单独设置，若仍采用 RAM 方式实现 S 盒的查表，则使用的 RAM 资源会超出 FPGA 上块 RAM 数目的限制。故采用查找表的方式来代替实现 S 盒，在牺牲部分模块工作性能的代价下，换取全流水线与改进的加密模式对吞吐量的提升。非线性函数模块的实现如图 9 所示。

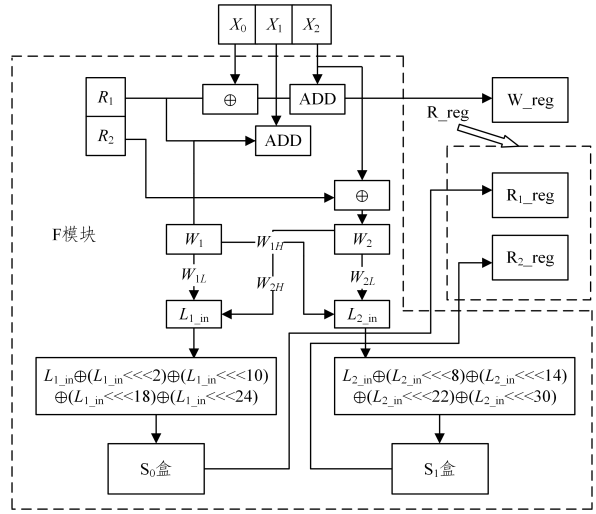

图 9 非线性函数模块

Fig. 9 Nonlinear function module

需说明的是，S 盒查找表的构成请见文献[11]。

4.1.4 全流水 LFSR 模块的实现

在各独立模块中，LFSR 的模加器链被封装为 AddLine 部件。考虑到时序平衡与性能效率的问题，我们改进文献[2-6]中提到的加法器链设计，选用 CSA 与 CLA(Carry Look-Ahead Adder，进位超前加法器)设计加法器链的结构。在该部件的设计中，我们尽可能将新的 s_{15} 涉及的模加运算分割开来。如图 10 所示，我们将涉及 W 的加法运算也融合进了模加加法器链的设计中。为提升轮次模块的通用性，本方案加入一个输入选择器来控制下一轮生成 s_{15} 的计算路径，从而区别初始化与工作轮次。由于 CSA 的输出并非加法的最终结果，需经过其他类型的加法器进行结果整合，因此在输入选择器后加入两个 CLA 构成一个模加器进行结果整合与模加运算。

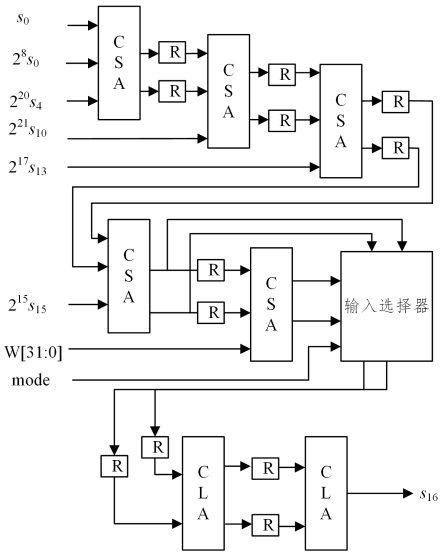

图 10 AddLine 内部结构的全流水线设计

Fig. 10 Full pipeline design of AddLine internal structure

在 CSA 中每 bit 均可计算 Carry 与 Sum，因此运算速度极快[17]。CSA 的计算方式如式(22)和式(23)所示：

$$Carry = A_i \& B_i \mid A_i \& C_i \mid B_i \& C_i \tag{22}$$

$$Sum = A_i \oplus B_i \oplus C_i \tag{23}$$

其中，A，B，C 为 CSA 的输入，$Carry$ 代表进位，Sum 为和。

若需得到最后结果，可使用其他类型的加法器进行如式(24)的计算：

$$Carry <<< 1 + Sum = A + B + C \tag{24}$$

若要进行模加运算，可引入式(25)：

$$2Carry (\mathrm{mod}(2^{31}-1)) = Carry <<< 1 \tag{25}$$

由式(25)可推出式(26)：

$$(A+B+C)(\mathrm{mod}(2^{31}-1)) = (Sum + Carry <<< 1)(\mathrm{mod}(2^{31}-1)) \tag{26}$$

本节 AddLine 的设计中采用的模加器结构如图 11 所示，其中 A 与 B 为 CSA 加法器链的输出。

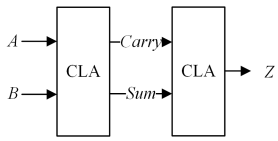

图 11 串行模加器

Fig. 11 Serial modular adder

在串行模加器的基础上，本方案采用了 CLA 加法器，CLA 采用了优化的进位逻辑，因此降低了电路逻辑时延[18]。设 A 和 B 分别为 4 位 CLA 加法器的输入，C 代表进位输出，G 代表生成信号，P 代表传播信号。

对于 CLA 的实现，有形如式(27)－式(30)的定义与公式：

$$G_i = A_i B_i \tag{27}$$

$$P_i = A_i + B_i \tag{28}$$

$$C_{i+1} = A_i B_i + A_i C_i + B_i C_i \tag{29}$$

由上式可得：

$$C_{i+1} = G_i + P_i C_i \tag{30}$$

图 12 给出了 4 位 CLA 加法器的结构。

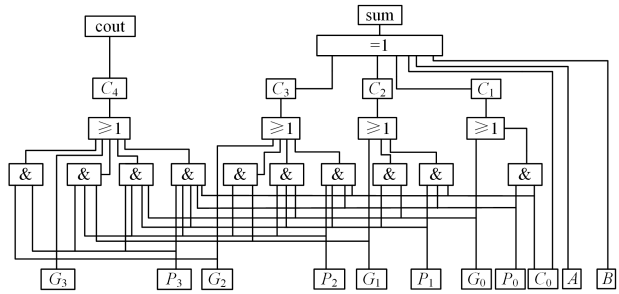

图 12 4 位 CLA 加法器电路结构

Fig. 12 4 bit CLA circuit diagram

但若 CLA 的扇出过大，也会增加逻辑延迟，因此我们采用 4 位 CLA 组合的形式来组成 $4n$ 位的 CLA。在本节的设计中 n 为 9。$4n$ 位 CLA 加法器的设计如图 13 所示。

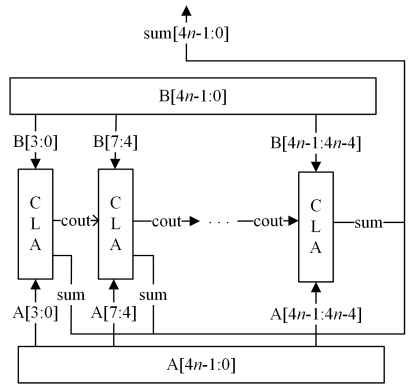

图 13 4n 位 CLA 加法器电路结构

Fig. 13 4n bit CLA circuit diagram

4.2 ZUC 算法半流水设计

在半流水线的设计中，BR，F，init_operation 模块的设计与全流水线相似，此处不再赘述。半流水线与全流水线的区别在于半流水线使用单轮模块实现多轮次功能，因此实现单模块流水设计需要复杂的逻辑控制，故图 14 中的半流水线设计中需引入一个逻辑控制模块(crt_block)来控制数据流的方向。

逻辑控制模块(crt_block)的工作时序如表 1 所列。其中，AddLine_res 为 AddLine 模块的输出，mode 为 crt_block 模块根据 clk_counter 变化而确定的 AddLine 的工作模式信号。当 mode 为 0 时，AddLine 为初始化状态；当 mode 为 1 时，其为工作状态。

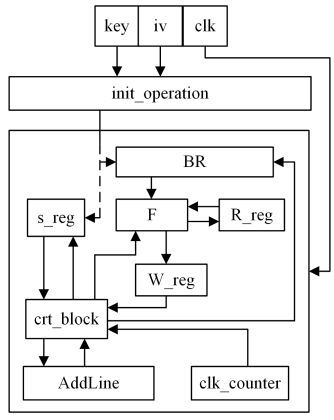

图 14 半流水设计

Fig. 14 Semi-pipelined architecture

表 1 模块输入与 clk_counter 的关系

Table 1 Relationship between module input and clk_counter

clk_counter	BR 输入	AddLine 输入
1	S[0],s[2],s[5],s[9],s[11],s[14],s[15]	s[0],*mode*=0
2	S[0],s[2],s[5],s[9],s[11],s[14],s[15]	s[1],s[4],*mode*=0
3	S[0],s[2],s[5],s[9],s[11],s[14],s[15]	s[2],s[5],s[10],*mode*=0
4	S[0],s[2],s[5],s[9],s[11],s[14],s[15]	s[3],s[6],s[11],s[13],mode=0
5	S[0],s[2],s[5],s[9],s[11],s[14],s[15]	s[4],s[7],s[12],s[14],s[15],*W*,mode=0
6	S[0],s[2],s[5],s[9],s[11],s[14],s[15]	s[5],s[8],s[13],s[15],AddLine_res,*W*,mode=0
7	S[0],s[2],s[5],s[9],s[11],s[14],s[15]	s[5],s[8],s[13],s[15],AddLine_res,*W*,mode=0
8	S[1],s[3],s[6],s[10],s[12],s[15],AddLine_res	s[5],s[8],s[13],s[15],AddLine_res,*W*,mode=0
…	…	…
38	S[1],s[3],s[6],s[10],s[12],s[15],AddLine_res	s[5],s[8],s[13],s[15],AddLine_res,mode=1
…	…	…

本章中 AddLine 采用流水线化设计，即在各个加法器之间添加寄存器，以达到细化操作，提升算法工作频率的目的。其与全流水线的区别在于链中的加法器使用并行模加器来替代。

为了配合单轮模块的流水线工作，并防止出现零延迟震荡现象，因此采用 $n=8$，即 32 位的 CLA 加法器组成的模加器设计加法器链。半流水线设计的 AddLine 内部结构如图 15 所示，其中 Mode 信号用来控制数据流的流向。

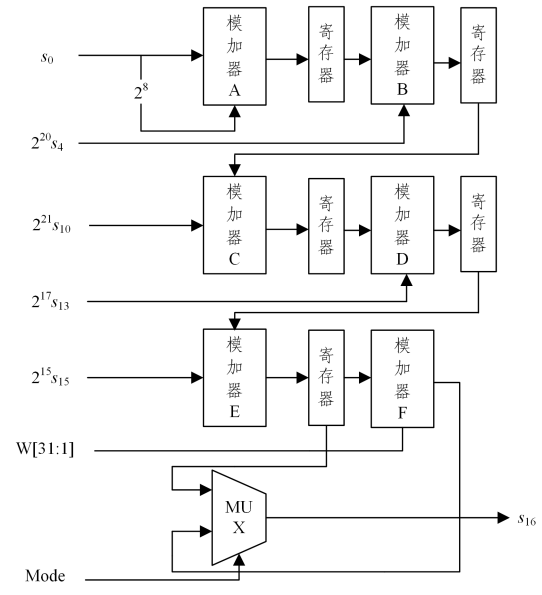

图 15 AddLine 内部结构的半流水设计

Fig. 15 Semi-pipelined design of AddLine internal structure

与 4.1 节中的模加器不同，此处的模加器的关键路径仅有一个加法器与一个多路选择器。ZUC 算法的一种模加算法如式(31)所示。其中，C 为 $A+B$ 结果的第 31 位，D 为第 32 位。

$$\begin{aligned} Z &= (A+B) \bmod (2^{31}-1) \\ &= C+2^{31}D \bmod (2^{31}-1) \\ &= C+D \end{aligned} \tag{31}$$

根据式(31)，可设计出文献[4]中提出的模加器架构，如图 16 所示。计算原理为：先利用两个加法器并行计算 $A+B$ 与 $A+B+1$，之后根据 $A+B$ 的进位结果来选择最终输出的结果；若进位为 1，则输出 $A+B+1$，否则输出 $A+B$。

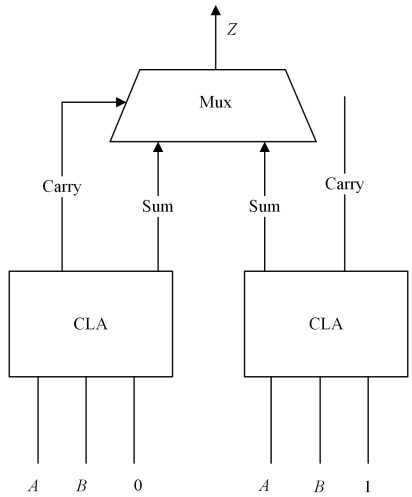

图 16 并行模加器

Fig. 16 Parallel modular adder

5 实验结果与分析

基于以上两个部分的改进，本文设计了以 ZUC 算法为核心的高性能数据加密方案。加密算法的核心有两种形式，一种是全流水线设计；另一种是半流水线设计，可以通过控制信号进行选择。本文利用硬件编程语言 Verilog 基于 vivado2019.02FPGA 设计平台，在以 xcku060-ffva1156-2-i 为核心的 Xilinx 公司的 FPGA 加速卡上对以上加密方案进行了综合仿真与上板测试。该 FPGA 加速卡集成了 PCIe2.0X8 硬核。

5.1 方案分析

各方案资源性能对比如表 2 所列。从表 2 来看，相比文献[5,8,16-18]，本文方案的吞吐量和工作频率都具有明显的优势。相比文献[2]，本文的全流水线设计的工作频率也具有相当的优势。与其他的设计相比，本文提出的加密方案的吞吐量和工作频率都具有较高的提升，并且具有更高的覆盖面。

表 2 各方案资源性能对比

Table 2 Resource performance comparison of various schemes

方案	器件	频率/MHz	吞吐量/Gbps	LUT	FF	Slice
文献[2]	Virtex 5	222.0	7.111	—	—	575
文献[5]	Artix-7 Xc7a100tfgg484-2	166.3	5.322	—	—	305
文献[8]	Zynq-7020	142.0	4.500	19553	—	6011
文献[16]	Virtex-5	172.0	5.504	—	—	395
文献[17]	Virtex-5	65.0	2.080	—	641	386
文献[18]	DE-115	115.0	3.680	—	—	—
加密算法半流水线设计模块	xcku060-ffva1156-2-i	201.0	25.728	5818	3475	—
加密算法全流水线设计模块	xcku060-ffva1156-2-i	360.0	46.080	91891	79810	—

各模块资源和性能如表 3 所列。结合表 3 与模块设计功能综合分析可知，由于全流水线模块的工作方式为每输入 128bit 初始密钥则生成 128bit 加密密钥，且其工作性能高，吞吐量大，因此其适用于在线网络、边缘设备的短报文超高速加密过程[19]。由于半流水线模块的工作方式为输入 128bit 初始密钥就可生成 128bit 的加密密钥流，且功耗较低，因此其不仅适用于传统视频流与图片集的加密，还适用于车辆网、物联网等的长数据包高速加密过程[20]。

表 3 各模块资源和性能

Table 3 Resources and performance of each module

模块	LUT	FF	功耗/W	频率/MHz	吞吐量/Gbps
核心控制模块	322	1104	0.086	—	—
内存读写控制逻辑	208	709	0.091	—	—
加密算法全流水线模块	91891	79810	22.780	360	46.080
加密算法半流水线模块	5818	3475	3.587	201	25.728

5.2 加解密效率分析

为了进一步验证 4.1 节和 4.2 节中设计的 ZUC 加解密核心算法的实际性能，本小节基于集成了 PCIe 硬核的板卡进行测试，测试时 XDMA 的配置如下：AXI_4 接口；AXI 总线宽度为 128 bit；参考时钟为 100 MHz；PCIe 接口为 PCIe2；采用的板卡最大带宽为 5GT/s×8＝40 GT/s，即 40 Gbps。为测试程序的稳定性，分别选取 100 MHz，180 MHz，260 MHz 对全流水线设计模块和半流水线设计模块进行测试。经测试，半流水线模块可在 100 MHz 与 180 MHz 工作频率下稳定工作，全流水线设计模块均可在上述工作频率下稳定工作。各模块启用时在相应频率下上板测试的吞吐量如图 17 所示。

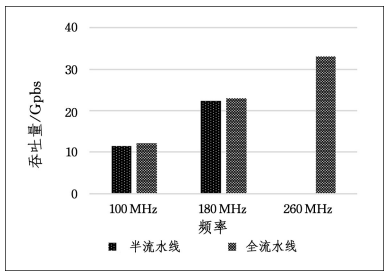

图 17 各频率下模块吞吐量

Fig. 17 Module throughput at each working frequency

结束语 本文提出了一套基于 FPGA 的 ZUC 高性能数据加密方案，该方案含有两个针对不同情景的算法核心。其中半流水线设计核心可以针对长数据包以及相关传统的 ZUC 算法工作场景进行加解密，而全流水线设计可以覆盖短报文、超高速在线加密等需要及时性高的场景。与同类的加密方案相比，本文提出的加密方案速度更快，适用场景更广。

未来研究工作将进一步探索 ZUC 算法的安全设计，防止侧信道攻击，在保证高性能的同时防止信息泄漏。

参 考 文 献

[1] WANG C，HOU Z K，LIU P C，et al. Security frequency hopping communication system based on improved ZUC algorithm[J]. Journal of Tsinghua University(Natural Science Edition)，2019，59(2)：154-161.

[2] LIU Z，ZHANG L，JING J，et al. Efficient pipelined stream cipher ZUC algorithm in FPGA[C]// First Int'l Workshop on ZUC Algorithm，China. 2010.

[3] KITSOS P，SKLAVOS N，SKODRAS A N. An FPGA implementation of the ZUC stream cipher[C]// 2011 14th Euromicro Conference on Digital System Design. IEEE，2011：814-817.

[4] ZHANG L，XIA L，LIU Z，et al. Evaluating the Optimized Implementations of SNOW3G and ZUC on FPGA[C]// 2012 IEEE 11th International Conference on Trust，Security and Privacy in Computing and Communications. IEEE，2012：436-442.

[5] GUO H J，DONG X Z，GAO X W. Hardware Implementation of ZUC Algorithm Based on FPGA[J]. Computer Engineering，2014，40(8)：268-272.

[6] LIU Z，ZHANG Q，MA C，et al. HPAZ：A high-throughput pipeline architecture of ZUC in hardware[C]// 2016 Design，Automation & Test in Europe Conference & Exhibition(DATE). IEEE，2016：269-272.

[7] LU B，YAN L M. Hardware architecture of an area optimized ZUC algorithm[J]. Journal of Fudan University(Natural Science)，2021，60(4)：492-498，509.

[8] ZHOU W，WANG B，PAN W T. ZUC hardware implementation research[J]. Foreign Electronic Measurement Technology，2015(7)：66-71.

[9] ZHANG Y P，GAO Y，YAN Y，et al. Fast Software Implementation of ZUC Algorithm[J]. Journal of Cryptologic Research，2021，8(3)：388-401.

[10] BAI L，JIA W Y，ZHU G Z. Lightweight Hardware Design and Implementations of ZUC-256 Stream Cipher on FPGA[J]. Journal of Cryptologic Research，2021，8(3)：521-536.

[11] GB/T 33133.1-2016. 2016-10-13，信息安全技术 祖冲之序列密码算法 第 1 部分：算法描述[S]. 国家密码管理局. 2016.

[12] 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 & Processing，2022，37(3)：695-702.

[13] GB/T 33133.2-2021. 2021-10-11，信息安全技术 祖冲之序列密码算法 第 2 部分：保密性算法[S]. 全国信息安全标准化技术委员会(SAC/TC 260). 2021.

[14] MATH S S，MANJULA R B，MANVI S S，et al. Data transactions on system-on-chip bus using AXI4 protocol[C]// 2011 International Conference on Recent Advancements in Electrical，Electronics and Control Engineering. IEEE，2011：423-427.

[15] NOAMI A，PRADEEP KUMAR B，CHANDRASEKHAR P. High performance AXI4 interface protocol for multi-core memory controller on SoC[M]//Data Engineering and Communication Technology. Springer，Singapore，2021：131-140.

[16] ZHANG L，XIA L，LIU Z，et al. Evaluating the Optimized Implementations of SNOW3G and ZUC on FPGA[C]// Proceedings of the 2012 IEEE 11th International Conference on Trust，Security and Privacy in Computing and Communications. 2012：436-442.

[17] JINPENG W, TENG Z, BO Z, et al. An Innovative FPGA Implementations of the Secure frequency hopping communication system based on the improved ZUC algorithm[J]. IEEE Access, 2022, 10: 54634-54648.
[18] WANG Y, WU L, ZHANG X, et al. A hardware implementation of ZUC-256 stream cipher[C]// 2020 IEEE 14th International Conference on Anti-counterfeiting, Security, and Identification (ASID). IEEE, 2020: 94-97.
[19] MUNDHE P, VERMA S, VENKATESAN S. A comprehensive survey on authentication and privacy-preserving schemes in VANETs[J]. Computer Science Review, 2021, 41: 100411.
[20] WANG Z Y, GUO Y, LI S Q, et al. Design of efficient anonymous identity authentication protocol for lightweight IoT devices[J]. Journal on Communications, 2022, 43(7): 49-61.

ZHANG Bolin, born in 1999, master. His main research interests include high-performance computing and information security.

LI Bin, born in 1986, Ph.D, lecturer. His main research interests include high-performance computing and information security.

（责任编辑：何杨）