文章编号: 1001 -9081(2013)02 -0450 -05 doi: 10.3724/SP.J.1087.2013.00450

可重构的串行高级加密标准加解密电路设计

谢惠敏[1],郭东辉[1,2*]
(1. 厦门大学 电子工程系,福建 厦门 361005; 2. 福建省集成电路设计工程技术研究中心,福建 厦门 361005)
(* 通信作者电子邮箱 dhguo@xmu.edu.cn)

摘 要: 为了进一步提高高级加密标准(AES)算法在现场可编程门阵列(FPGA)上的硬件资源使用效率,提出一种可支持密钥长度128/192/256位串行AES加解密电路的实现方案。该设计采用复合域变换实现字节乘法求逆,同时实现列混合与逆列混合的资源共享以及三种AES算法密钥扩展共享。该电路在Xilinx Virtex-V系列的FPGA上实现,硬件资源消耗为1871 slice、4 RAM。结果表明,在最高工作频率173.904 MHz时,密钥长度128/192/256位AES加解密吞吐率分别可达2119/1780/1534 Mb·s^{-1}。该设计吞吐率/硬件资源比值较高,且适用支持千兆以太网。

关键词: 高级加密标准; 现场可编程门阵列; 密钥扩展; 加密; 解密
中图分类号: TN402; TP309 **文献标志码:** A

Reconfigurable serial AES encryption and decryption circuit design

XIE Huimin, GUO Donghui*
(1. *Department of Electronic Engineering, Xiamen University, Xiamen Fujian* 361005, *China*;
2. *Fujian IC R&D Engineering Center, Xiamen Fujian* 361005, *China*)

Abstract: To improve the efficiency of hardware resources of the Advanced Encryption Standard (AES) algorithm on the Field Programmable Gate Array (FPGA), an implementation method of serial AES circuit that could perform both encryption and decryption with 128/192/256 bit key options was proposed. The design computed byte multiplication inverse in composite field transform, integrated MixColumn and InvMixColumn circuits, and fused three kinds of key expansion algorithms at the same time. The design was implemented in Xilinx FPGA Virtex-V and the consumption of hardware resources was 1871 slices, 4 block RAM. The results show that the throughput can be up to 2119/1780/1534 Mb·s^{-1} for 128/192/256 bit key length while the maximum frequency is 173.904 MHz. The design achieves high throughput/hardware resource ratio and can be applied to the Gigabit Ethernet.

Key words: Advanced Encryption Standard (AES); Field Programmable Gate Array (FPGA); key-expansion; encryption; decryption

0 引言

随着开放环境中传输敏感数据量的迅速增长,数据加密越来越重要。高级加密标准(Advanced Encryption Standard, AES)是美国国家标准技术研究所(National Institute of Standards and Technology, NIST)于2001年11月26日发布的对称数据加密算法,采用了Rijndael算法。它作为新一代数据加密标准,广泛应用于网络协议安全、无线局域网、自动取款机等领域[1-2]。

AES根据不同的密钥长度分为AES-128、AES-192、AES-256。到目前为止,已经有许多关于AES实现方法的研究。文献[3]采用流水设计实现了AES-128加密电路,使用RAM来存储S-box值,具有高达34 Gb/s的吞吐率,但用的硬件资源多达2389 slice、200 RAM。Hammad等[4]同样采用了流水结构,具有高达39 Gb/s的吞吐率,硬件资源消耗多达10662 slice。可以看出,虽然流水结构实现了高吞吐率,但是面积和功耗相对较大,而且对于千兆以太网的应用,所达到的吞吐率又远远超过实际所需要。文献[5-7]实现了AES-128加解密电路,但不能兼容三种密钥长度的加解密。可重构AES在无需改变硬件条件下,可以根据不同的应用需求,灵活选择不同的密钥长度进行加密和解密。Sever等[8]提出了兼容密钥长度128/192/256位AES电路算法实现,但没有对加密和解密过程进行整合,吞吐率/面积不具备优势。可重构AES设计面临的主要问题是如何以较小硬件资源消耗实现可重构,因此对可重构AES的研究很有现实意义。

本文在分析以上资料的基础上,采用非流水结构实现了可重构的AES-128/192/256的加解密电路,提出一种新的密钥扩展实现方案,缩短密钥扩展的关键路径,提高了最大吞吐率;采用RAM来存储密钥扩展生成的轮密钥,更有利于加解密过程中轮密钥的直接读取。相比于Sever等[8]的电路实现,本文等效硬件资源减少了59%,同时吞吐率提高了21%~26%;相比于Chen等[9]的实现,本文所用硬件资源节省相当,但加密吞吐率提高了3倍左右,解密吞吐率提高了6倍左右。

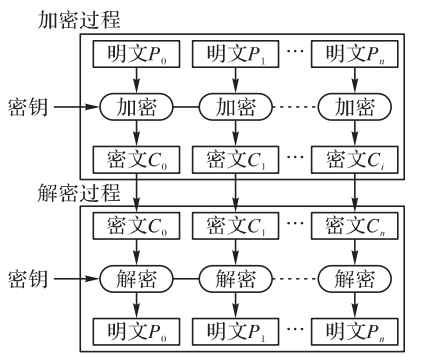
1 AES架构

AES采用了Rijndael算法,数据固定分组长度为128位,密钥长度和轮数可变,根据不同的密钥长度,AES-128/192/256分别需要10/12/14轮。如图1(a)所示,加密者对明文进行以128位为单位分组$P = P_0P_1\cdots P_n$。每个明文分组加密后生成128位密文流$C = C_0C_1\cdots C_n$。同样的,解密者用相同的初始密钥,将密文$C = C_0C_1\cdots C_n$解密得到明文$P = P_0P_1\cdots P_n$。

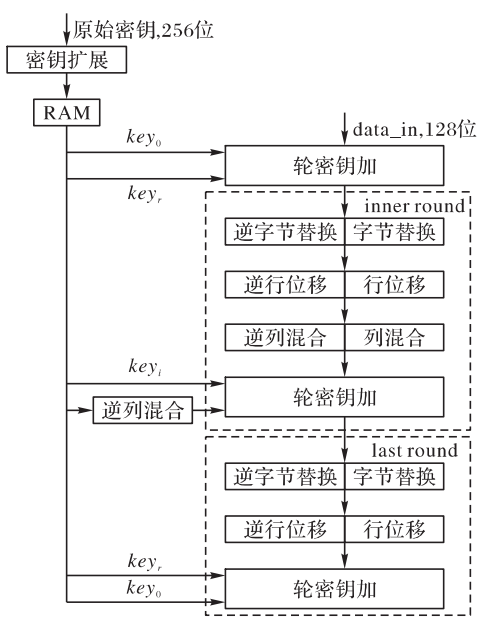
收稿日期: 2012-08-15; 修回日期: 2012-09-10。
作者简介: 谢惠敏(1988-),女,福建厦门人,硕士研究生,主要研究方向: 集成电路设计、密码学; 郭东辉(1967-),男,福建莆田人,教授,博士研究生,主要研究方向: 人工智能、网络通信、集成电路设计。

128 位的明文、密文和中间状态都用一个 4 × 4 字节状态数组矩阵来进行处理。

如果加密和解密过程由两个电路分开实现，这将会占用较大的硬件资源[9]。为了节省硬件资源消耗，在本文设计中，AES 加解密算法流程如图 1(b) 所示，它们有相同的数据操作流程。明文加密过程由轮密钥加、字节替换、行变换和列混合变换构成，最后一轮略有不同，不包含列混合变换变换。相应的密文解密过程由轮密钥加、逆字节替换、逆行变换和逆列混合变换构成，最后一轮不包含逆列混合变换变换。轮密钥 $key_0 \sim key_i$ 由 AES 算法的密钥扩展基于原始密钥计算产生。需要注意的是: 在解密过程中，除去首轮和最后一轮，从 RAM 中取出的轮密钥需要先经进行逆列混合运算。



(a) AES框图



(b) AES内部结构

图 1　AES 加解密流程

2　各个模块设计

在每一轮加密操作中，字节替换和行移位共需要一个时钟周期，列混合和轮密钥加共需要一个时钟周期。因此包括首轮的轮密钥加，AES-128/192/256 加密分别需要 21/25/29 个时钟周期。解密过程与加密过程有相同的数据流程，所需时钟周期跟加密过程一样。

2.1　字节替换和逆字节替换

字节替换和逆字节替换为非线性变换，常见方法有查找表替代法，但这种方法会占用大量的硬件资源。目前已经有很多关于 S 盒实现方法的研究[10-13]，已经证实基于复合域变换的实现方案能够降低硬件资源消耗。

1) 字节替换。

①状态矩阵上每个字节，取它在有限域 $GF(2^8)$ 上的乘法逆来代替;

②把上一步得出的字节值再进行如下仿射变换:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{1}$$

2) 逆字节替换。

①先进行如下逆仿射变换:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2}$$

②然后再对得到的每个字节取 $GF(2^8)$ 上的乘法逆。

3) 基于复合域变换的乘法求逆。

通过复合域 $GF((2^4)^2)$ 上的字节求逆代替 $GF(2^8)$ 上的字节求逆来达到减少了硬件资源消耗的目的。下面介绍具体实现过程:

①$GF(2^8)$ 上每个元素通过同构线性变换 T 函数可转化为复合域上 $GF((2^4)^2)$ 上的元素，$GF((2^4)^2)$ 上的元素又可以表示为系数在 $GF(2^4)$ 上的一次多项式 $bx + c$，如式(3) 所示将 $GF(2^8)$ 上的元素 a 转化到有限域 $GF(2^4)$ 上的元素 b，c。

$$(b,c) = Ta = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} b_3 \\ b_2 \\ b_1 \\ b_0 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} \tag{3}$$

② 构建有限域 $GF(2^4)$ 上二次不可约多项式 $p(x) = x^2 + x + 9$，利用域 $GF(2^4)$ 上的加法、乘法和求逆运算，得到域 $GF(2^4)$ 上元素 b，c 的逆元素 p，q。

$$\begin{cases} (bx + c)^{-1} = b(9 \times b^2 + bc + c^2)^{-1}x + \\ \qquad\qquad (c + b)(9 \times b^2 + bc + c^2)^{-1} \\ p = b(9 \times b^2 + bc + c^2)^{-1} \\ q = (c + b)(9 \times b^2 + bc + c^2)^{-1} \end{cases} \tag{4}$$

③ 将在 $GF(2^4)$ 上所求的逆元素通过 $T^{-1}(p,q)$ 线性逆变换，得到等效的 $GF(2^8)$ 上乘法逆元素。

$$T^{-1}(p,q) = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_3 \\ p_2 \\ p_1 \\ p_0 \\ q_3 \\ q_2 \\ q_1 \\ q_0 \end{bmatrix} = \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} \tag{5}$$

4) 字节替换与逆字节替换的统一架构。

字节替换模块的计算单元主要包括字节求逆和仿射变换，逆字节替换模块的计算单元主要包括字节求逆和逆仿射变换，在采用上述的基于复合域变换求字节逆算法后，还可以将字节求逆单元进行硬件资源复用。在采用上述的基于复合域变换求字节逆算法后，还可以通过加/解信号控制将字节求逆单元进行硬件资源复用，字节替换和逆字节替的统一架构包含字节求逆、仿射变换和逆仿射变换，大大地降低了硬件资源消耗，具体如图 2 所示。

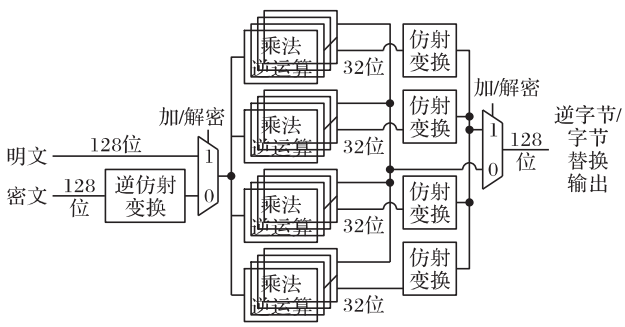


图 2 字节替换与逆字节替换电路

2.2 行移位和逆行移位

加密时，行移位对状态数组矩阵逐列行进行变换，第一行到第四行分别左移 0 ~ 3 个字节；解密时，逆行移位对状态数组矩阵逐列行进行变换，第一行到第四行分别右移 0 ~ 3 个字节。

2.3 列混合和逆列混合

如图 3 所示，列混合和逆列混合对 128 位状态数组矩阵分 4 列逐列进行变换。文献[14]详细分析了几种列混合和逆列混合实现方法和硬件资源消耗比较，其中 bothmixcol _sb 使用硬件资源最少，为 398 LUT。本文在逆列混合的实现过程中与文献[14]有所不同，使用了更少的硬件资源。下面是具体的分析以及实现。

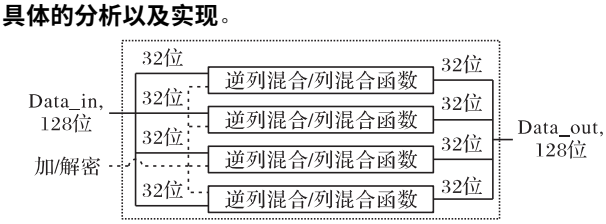


图 3 逆列混合和列混合实现框图

列混合变换与固定矩阵相乘，每一列变换过程如下：

$$\begin{bmatrix} s_{0,c}' \\ s_{1,c}' \\ s_{2,c}' \\ s_{3,c}' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}; \ 0 \leqslant c \leqslant 3 \tag{6}$$

$$\begin{cases} S_{0,C}' = (2 \times S_{0,C}) \oplus (3 \times S_{1,C}) \oplus S_{2,C} \oplus S_{3,C} = \\ \quad 2 \times (S_{0,C} \oplus S_{1,C}) \oplus (S_{0,C} \oplus S_{1,C} \oplus S_{2,C} \oplus \\ \quad S_{3,C}) \oplus S_{0,C} \\ S_{1,C}' = (2 \times S_{1,C}) \oplus (3 \times S_{2,C}) \oplus S_{0,C} \oplus S_{3,C} = \\ \quad 2 \times (S_{1,C} \oplus S_{2,C}) \oplus (S_{0,C} \oplus S_{1,C} \oplus S_{2,C} \oplus \\ \quad S_{3,C}) \oplus S_{1,C} \\ S_{2,C}' = (2 \times S_{2,C}) \oplus (3 \times S_{3,C}) \oplus S_{0,C} \oplus S_{1,C} = \\ \quad 2 \times (S_{2,C} \oplus S_{3,C}) \oplus (S_{0,C} \oplus S_{1,C} \oplus S_{2,C} \oplus \\ \quad S_{3,C}) \oplus S_{2,C} \\ S_{3,C}' = (2 \times S_{3,C}) \oplus (3 \times S_{0,C}) \oplus S_{1,C} \oplus S_{2,C} = \\ \quad 2 \times (S_{0,C} \oplus S_{3,C}) \oplus (S_{0,C} \oplus S_{1,C} \oplus S_{2,C} \oplus \\ \quad S_{3,C}) \oplus S_{3,C} \end{cases} \tag{7}$$

逆列混合变换是列混合变换的逆变换，逆列混合变换如下：

$$\begin{bmatrix} s_{0,c}'' \\ s_{1,c}'' \\ s_{2,c}'' \\ s_{3,c}'' \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0E \end{bmatrix} \cdot \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}; \ 0 \leqslant c \leqslant 3 \tag{8}$$

$$\begin{cases} s_{0,c}'' = (0e \times s_{0,c}) \oplus (0b \times s_{1,c}) \oplus (0d \times s_{2,c}) \oplus \\ \quad (09 \times s_{3,c}) = 2 \times (2 \times (2 \times (s_{0,c} \oplus s_{1,c}) \oplus \\ \quad 2 \times (s_{2,c} \oplus s_{3,c}) \oplus s_{0,c} \oplus s_{2,c})) \oplus s_{0,C}' \\ s_{1,c}'' = (09 \times s_{0,c}) \oplus (0e \times s_{1,c}) \oplus (0b \times s_{2,c}) \oplus \\ \quad (0d \times s_{3,c}) = 2 \times (2 \times (2 \times (s_{1,c} \oplus s_{2,c}) \oplus \\ \quad 2 \times (s_{3,c} \oplus s_{0,c}) \oplus s_{1,c} \oplus s_{3,c})) \oplus s_{1,C}' \\ s_{2,c}'' = (0d \times s_{0,c}) \oplus (09 \times s_{1,c}) \oplus (0e \times s_{2,c}) \oplus \\ \quad (0b \times s_{3,c}) = 2 \times (2 \times (2 \times (s_{0,c} \oplus s_{1,c}) \oplus \\ \quad 2 \times (s_{2,c} \oplus s_{3,c}) \oplus s_{0,c} \oplus s_{2,c})) \oplus s_{2,C}' \\ s_{3,c}'' = (0b \times s_{0,c}) \oplus (0d \times s_{1,c}) \oplus (09 \times s_{2,c}) \oplus \\ \quad (0e \times s_{3,c}) = 2 \times (2 \times (2 \times (s_{1,c} \oplus s_{2,c}) \oplus \\ \quad 2 \times (s_{3,c} \oplus s_{0,c}) \oplus s_{1,c} \oplus s_{3,c})) \oplus s_{3,C}' \end{cases} \tag{9}$$

将式(6)和(8)展开得到式(7)和(9)，通过式(7)和(9)分析发现，列混合和逆列混合可以使用同一套电路结构实现，具体实现电路如图 4 所示。实现的电路硬件资源消耗为 380 LUT，比文献[14]实现方案减少了 4.5%。

2.4 轮密钥加与密钥扩展

轮密钥加操作根据加解密轮数选择轮密钥进行异或运算，加密过程中依次取轮密钥 $key_0 \sim key_r$，解密过程中依次取轮密钥 $key_r \sim key_0$。轮密钥基于初始密钥扩展生成，扩展结果存放在 RAM 中，并在完成之后输出 key_ready 信号。密钥扩展电路实现如图 5 所示，集成了三种密钥长度的扩展，门控单元 control 根据不同密钥长度的 AES 加解密模式来选择存入 RAM 的轮密钥。R_1 与 R_2 为两个 256 位的寄存器，对 AES-128/256，寄存器 R_1 用来存储初始密钥或者下一轮密钥计算结果；对 AES-192，寄存器 R_2 [192:0] 用来暂存下一轮的密钥计算结果。为了减小关键路径，寄存器 R_{2_7}、R_{2_8} 用来存储字节替换值；对 AES-128/192/256，低 32 位需要经过字循环和字节替换，值存在寄存器 R_{2_7} 中；对 AES-256，还需要寄存器 R_{2_8}。一次性算好 AES-128/192/256 加解密扩展轮密钥分别需要 30/27/24 个时钟周期。三种密钥的扩展具体分析如下：

1) AES-128、AES-196 的 Nk 分别为 4、6：

$$W[i]=\begin{cases}W[i-Nk]+W[i-1], & i>Nk \text{ 且 } i \text{ 不是 } Nk \text{ 的倍数}\\ W[i-Nk]+T(W[i-1]), & i\geqslant Nk \text{ 且 } i \text{ 为 } Nk \text{ 的整数倍}\end{cases}$$

AES-256 的 Nk 为 8:

$$W[i]=\begin{cases}W[i-Nk]+T(W[i-1]), & i\geqslant Nk \text{ 且 } i \text{ 为 } Nk \text{ 的整数倍}\\ W[i-Nk]+Subword(W[i-1]), & i\geqslant Nk \text{ 且 } i \bmod Nk=4\\ W[i-Nk]+W[i-1], & \text{其他}\end{cases}$$

2) T 函数包括 3 个部分: 字循环、字节替换和轮常量异或。

①字循环:

输入: $[d_0,d_1,d_2,d_3]$;

输出: $Rotword\_data=[d_0,d_1,d_2,d_3]$。

②字节替换:

先进行 $GF(2^8)$ 上的乘法逆替换,再进行仿射变换。

$$Inverse\_data=Inverse(Rotword\_data)$$

$$Affine\_data=Affine(Inverse\_data)$$

③轮常量异或,如表 1 所示。

表 1　轮常量

i	$Rcon[i]$ (Hex)	i	$Rcon[i]$ (Hex)
1	01 00 00 00	6	20 00 00 00
2	02 00 00 00	7	40 00 00 00
3	04 00 00 00	8	80 00 00 00
4	08 00 00 00	9	1B 00 00 00
5	10 00 00 00	10	36 00 00 00

$Rcon[0]=(RC[i],0,0,0)$, $RC[0]=2*RC[i-1]$,其中"*"也是定义在 $GF(2^8)$ 上的乘法。观察表 1,发现 $RC[1]\sim RC[8]$ 可以通过初始值为 8'h01 的寄存器循环左移来实现,从 $RC[9]\sim RC[10]$ 则可以通过初始值为 8'h36 移位寄存器来实现,在 8 次的左移之后变为了 8'h1B。$RC[0]$ 在 $GF(2^8)$ 的实现用简单的移位逻辑实现,具体实现电路如图 6 所示。

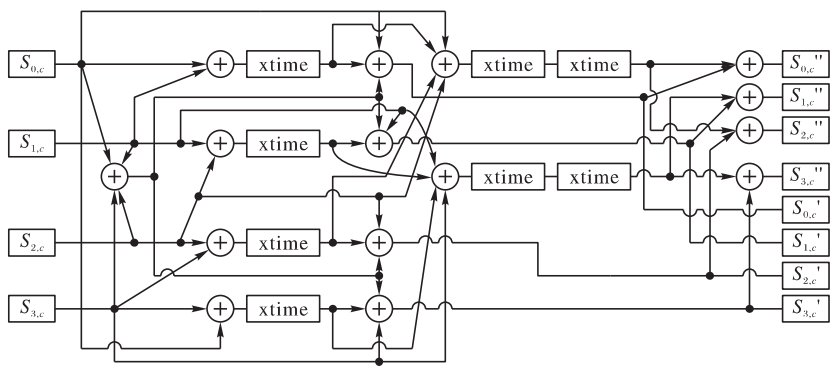


图 4　逆列混合和列混合具体实现电路

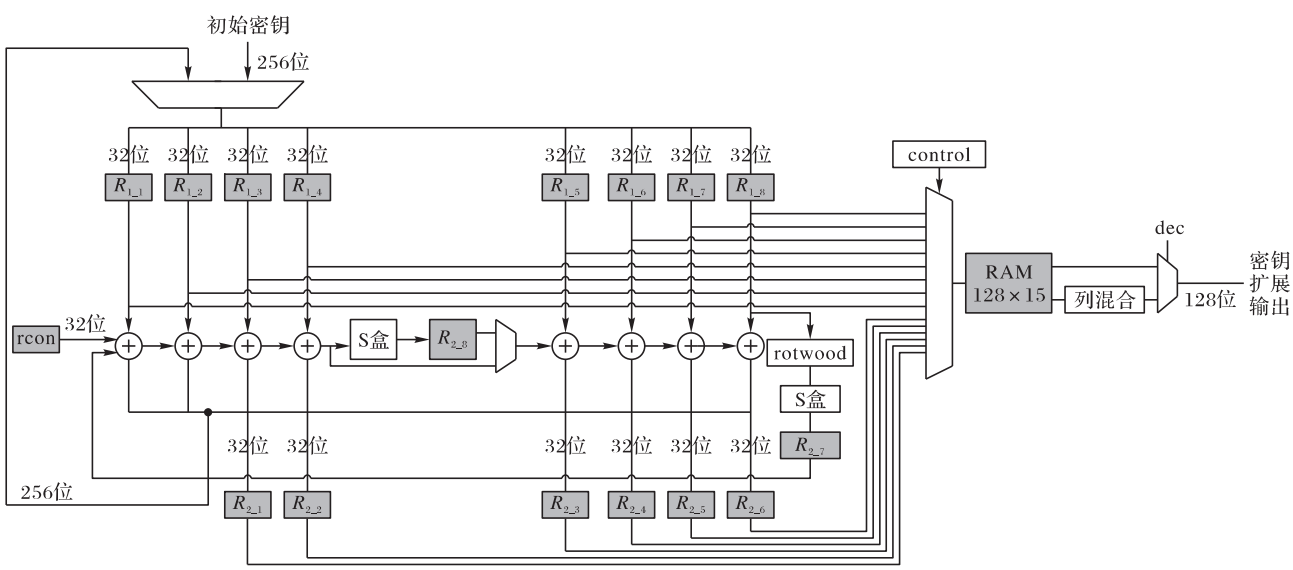


图 5　密钥扩展电路

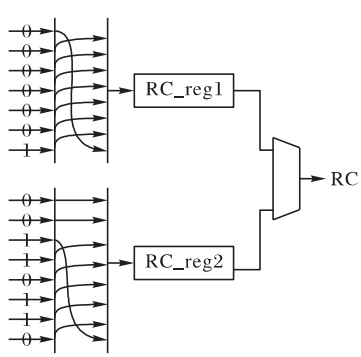


图 6　RC 实现电路

3　整体电路仿真以及性能分析

本文设计已经成功集成到 32 位处理器 Leon2 中,如图 7 所示,AES 通过接口模块作为从设备挂载到 AMBA 高速总线上,整体电路经 Modlesim 功能仿真正确。以 AES-256 加密为例,初始密钥 key 为 256'h00000001 00000002 00000003 00000004 00010203 04050607 08090a0b 0c0d0e0f,加密数据 data 为 256'h00000102 0000000E 0000000D 0000000C 0000000B 00000003 0000000A 00000009,仿真结果如图 8 所示与标准结果 256'h DC55916F FDF2ECA4 FF1B3E7E 275B394B 45F01896 C495699F 43435B86 6216F749 相一致。

用 Xilinx ISE 10.1 综合工具对电路进行综合仿真并下载到 Virtex-Ⅴ FPGA 上，得到结果如表 2 所示。AES 加解密电路硬件资源消耗为 1 871 slice + 4 RAM，其最高工作频率为 173.904 MHz。对于全流水型 AES，吞吐率等于频率与加解密分组长度的乘积；对于非流水 AES，密钥扩展一次性算好存储在 RAM 中，加密解密过程直接从 RAM 中读取，吞吐率的计算如式(10)所示[5]。在最高工作频率下，AES-128/192/256 吞吐率分别可达 2 119/1 780/1 534 Mb·s^{-1}。

$$throughput = blocksize * frequency / totalclockcycles \quad (10)$$

为了与其他同类 AES 设计具有可比性，目标器件分别选择了 XCV1000 和 XC2V3000，表 3 为本文设计与前期文献方法进行的总结比较。从表中可以看出，相比于文献[4-5]，虽然本文设计的硬件消耗较大，但是在可重构方面，本文设计兼容了三种密钥长度的加密和解密，并且吞吐率有所提高，硬件效率使用更高；相对于文献[6-7]，吞吐率有所提高，而且硬件资源大大减少；相比于文献[9]，虽然硬件资源相当，但是吞吐率得到明显的提高。其中一个 RAM 相当于 128 slice[15]。

表 2 基于 XC5V110T 仿真结果

密钥长度	密钥扩展周期	加解密周期	加密数据/位	硬件资源/slice	最高频率/MHz	加解密吞吐率/(Mb·s^{-1})
128	30	21				2 119
192	27	25	2×128	1 871	173.904	1 780
256	24	29				1 534

表 3 AES 核性能对比

算法	设备	E/D	硬件资源/slice	RAM	等效硬件资源/slice	最高频率/MHz	密钥长度	吞吐率/(Mb·s^{-1})
文献[5]算法	XCV300	E/D	2 358	—	2 358	22	128	259
文献[6]算法	XCV600	E/D	1 853	20	4 413	140.39	128	352
文献[7]算法	XC2V3000	E/D	7 617	—	7 617	75.3	128	876
文献[8]算法	XC2V8000	E/D	8 378	4	8 890	65	128 192 256	832 693 594
文献[9]算法	XC5VLX110T	E/D	2 420	0	2 420	175.75	128 192 256	681.7/340.85 576.82/288.41 499.91/249.96
本文算法	XC2V3000	E/D	3 148	4	3 660	85.39	128	461
	XCV1000	E/D	3 230	4	3 742	37.85	128 192 256	1 041 874.5 753.8

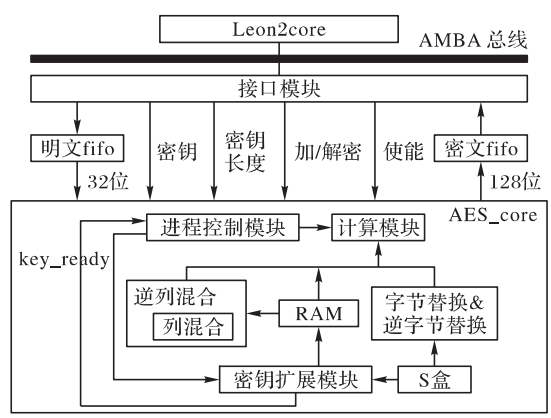


图 7 AES 集成到 Leon2 中

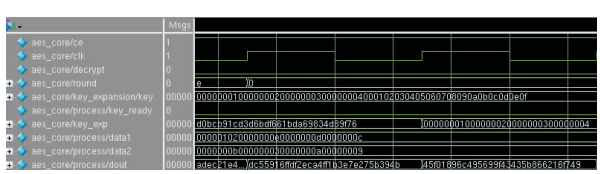


图 8 AES-256 加密仿真图

4 结语

综上所述，本文用 Verilog HDL 硬件描述语言设计实现了一种支持密钥长度 128/192/256 位的串行 AES 加解密电路，采用复合域变换实现字节乘法求逆，优化了列混合与逆列混合的电路共享，并提出了新的三种 AES 算法密钥的扩展共享实现方法。采用 Virtex-Ⅴ系列 FPGA 器件作为算法载体，在 Xilinx ISE10.1 下对系统进行综合、仿真并下载。FPGA 实验结果表明，本文设计能够根据需要灵活选择密钥长度，满足千兆以太网的数据交换速率的需求，并且硬件效率高于其他同类设计。

参考文献：

[1] Announcing the Advanced Encryption Standard (AES): Federal Information Processing Standards (FIPS) 197 [EB/OL]. (2001-11-26) [2010-09-01]. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[2] DAEMEN J, RIJMEN V. 高级加密标准(AES)算法 Rijndael 的设计[M]. 谷大武，徐胜波，译. 北京：清华大学出版社，2003.

[3] ZHANG Y L, WANG X G. Pipelined implementation of AES encryption based on FPGA [C]// 2010 IEEE International Conference on Information Theory and Information Security. Piscataway: IEEE, 2010: 170-173.

[4] HAMMAD I, EL-SANKARY K, EL-MASRY E. High-speed AES encryptor with efficient merging techniques [J]. IEEE Embedded Systems Letters, 2010, 2(3): 67-71.

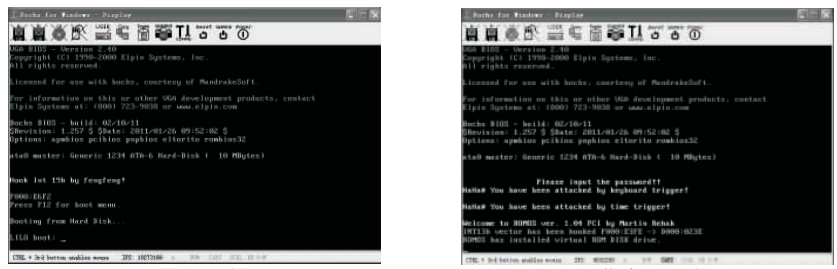
[5] SKLAVOS N, KOUFOPAVLOU O. Architectures and VLSI implementations of the AES-proposal Rijndael [J]. IEEE Transactions on Computers, 2002, 51(12): 1454-1459.

[6] BORKAR A M, KSHIRSAGAR R V, VYAWAHARE M V. FPGA implementation of AES algorithm [C]// The 3rd International Conference on Electronics Computer Technology. Piscataway: IEEE, 2011, 3: 401-405.

(下转第 459 页)

的检测。这是因为模块级陷门以整个模块为单位,按照 BIOS 模块的标准格式进行嵌入,改变了 BIOS 文件的模块构成,因此,利用基于模块构成分析的 BIOS 分析工具和本文检测方法都可以发现模块级陷门的存在。

2) 指令级陷门能够躲避 BIOS 分析工具的检测,却无法躲避本文所提方法的检测。其原因是指令级陷门在 BIOS 现有模块的内部进行修改,并没有改变 BIOS 文件的模块构成,具有更高的隐蔽性,因此能够躲避基于模块结构分析的 BIOS 分析工具的检测。然而,无论指令级陷门如何实现,都会对原始 BIOS 文件进行修改,同样会改变 BIOS 文件的完整性度量值,因此利用本文所提出的基于完整性度量的检测方法,可以发现指令级陷门的存在。指令级陷门完整性度量结果如表 5 所示,除 PCI ROM [A] 模块外,其余模块的完整性度量值都没有变化,因此可以判断,PCI ROM [A] 模块中嵌入了指令级陷门。



(a) 模块级陷门　　(b) 指令级级陷门

图 5　BIOS 陷门测试结果

表 5　指令级陷门完整性度量结果

模块名称	完整性度量值	
	原始 BIOS 文件	嵌入指令集陷门的 BIOS 文件
System BIOS	38D450A7	38D450A7
XGROUP CODE	541911B6	541911B6
ACPI table	696EE330	696EE330
YGROUP ROM	59F1B1D0	59F1B1D0
GROUP ROM [D]	31303AC3	31303AC3
PCI ROM [A]	55A08004	6EB5D648
Decompression block	94146927	94146927
Boot block	EF63726E	EF63726E

5　结语

随着计算机技术的发展,BIOS 安全的重要性日益凸显。目前,针对 BIOS 的攻击逐渐增多,而 BIOS 安全防护研究还相对滞后。本文在分析 BIOS 内部结构及代码混淆技术的基础上,研究了 BIOS 陷门的实现原理,提出了 BIOS 陷门检测方法,通过实验验证了检测方法的有效性。下一步的工作重点是在无法获得原始 BIOS 的情况下,研究如何有效地检测 BIOS 陷门。

参考文献:

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

[2] ZIMMER V, ROTHMAN M, HALE R. Beyond BIOS: implementing the unified extensible firmware interface with Intel's framework [M]. [S. l.]: Intel Press,2006: 2 – 9.

[3] 池亚平,许盛伟,方勇. BIOS 木马机理分析与防护[J]. 计算机工程,2011,37(13):122 – 124.

[4] HEASMAN J. Implementing and detecting an ACPI BIOS rootkit [EB/OL]. [2012 – 07 – 10]. http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Heasman.pdf.

[5] HEASMAN J. Implementing and detecting a PCI rootkit [EB/OL]. [2012 – 07 – 10]. http://www.blackhat.com/presentations/bh-dc-07/Heasman/Paper/bh-dc-07-Heasman-WP.pdf.

[6] ORTEGA A, SACCO A. Persistent BIOS infection [EB/OL]. [2012 – 07 – 10]. http://www.coresecurity.com/content/Persistent-Bios-Infection.

[7] KASPERSKY K. Shellcoder 编程揭秘[M]. 罗爱国,郑艳杰,译. 北京: 电子工业出版社,2006: 348 – 376.

[8] 王爽. 汇编语言[M]. 北京: 清华大学出版社,2008.

[9] 周振柳,刘宝旭,池亚平,等. 计算机 BIOS 安全风险分析与检测系统研究[J]. 计算机工程,2007,33(16):114 – 116.

[10] 王晓箴,刘宝旭,潘林. BIOS 恶意代码实现及其检测系统设计[J]. 计算机工程,2010,36(21):17 – 21.

[11] Removing a Bios-CMOS password [EB/OL]. [2012 – 07 – 10]. http://www.dewassoc.com/support/bios/bios_password.htm.

[12] Introduction to Bochs [EB/OL]. [2012 – 07 – 10]. http://bochs.sourceforge.net/doc/docbook/user/introduction.html.

(上接第 454 页)

[7] FAN C-P, HWANG J-K. Implementations of high throughput sequential and fully pipelined AES processors on FPGA [C]// ISPACS 2007: Proceeding of 2007 International Symposium on International Signal Processing and Symposium and Communication Systems. Piscataway: IEEE, 2007, 353 – 356.

[8] SEVER R, ISMAILGLU A N, TEKMEN Y C, *et al*. A high speed FPGA implementation of the Rijndael algorithm [C]// DSD 2004: Euromicro Symposium on Digital System Design. Piscataway: IEEE, 2004: 358 – 362.

[9] CHEN R J, PENG Y C, LAI J L, *et al*. Architecture design of high efficient and non-memory AES crypto core for WPAN [C]// NSS '09: Third International Conference on Network and System Security. Piscataway: IEEE, 2009: 36 – 43.

[10] RUDRA A, DUBEY P K, JUTLA C S, *et al*. Efficient Rijndael encryption implementation with composite field arithmetic [C]// CHES 2001: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer-Verlag, 2001: 171 – 184.

[11] 张志峰,林正浩. AES 加密算法中 S-BOX 的算法与 VLSI 实现[J]. 计算机工程与应用,2006,42(19):67 – 68

[12] 潘宏亮,高德远,张盛兵,等. 一种基于有限域求逆的 S-Box 实现算法[J]. 微电子学与计算机,2006,23(3):109 – 111,115.

[13] 韩少男,李晓江. 实现 AES 算法中 S-BOX 和 INV-S-BOX 的高效方法[J]. 微电子学,2010,40(1):103 – 107

[14] FISCHER V, DRUTAROVSKY M, CHODOWIEC P, *et al*. InvMixColumn decomposition and multilevel resource sharing in AES implementations [J]. IEEE Transactions on Very Large Scale Integration Systems,2005,13(8): 989 – 992.

[15] JARVINEN K, TOMMISKA M, SKYTTA J. Comparative survey of high performance cryptographic algorithm implementations on FPGAs [J]. IEE Proceedings Information Security, 2005, 152(1): 3 – 12.