第37卷第12期 中国科学技术大学学报 Vol. 37, No. 12
2007年12月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Dec. 2007

文章编号:0253-2778(2007)12-1461-05

一种基于 FPGA 的 AES 加解密算法设计与实现

张德学,郭 立,傅忠谦

(中国科学技术大学电子科学与技术系,安徽合肥 230026)

摘要:设计了一种用于低端设备、低功耗的 AES(advanced encryption standard)加解密硬件模块.混合设计加解密算法,减少了资源占用,使设备在较低的时钟频率下保持较高的性能,在20 MHz时,加解密速度仍可达 128 Mbit/s.

关键词:AES;加解密算法;FPGA

中图分类号:TP332.1 **文献标识码**:A

Design and implementation of AES algorithm based on FPGA

ZHANG De-xue, GUO Li, FU Zhong-qian

(*Department of electronic science and technology, USTC, Hefei* 230026, *China*)

Abstract: A low power consumption AES (advanced encryption standard) hardware module was designed. The mixed design of encryption-decryption saves hardware resources, keeping data rate, high at low clock frequency, with the maximal data rate up to 128 Mbit/s at 20 MHz.

Key words: AES; encryption-decryption algorithm; FPGA

0 引言

2000 年 10 月,美国 NIST(national standards and technology)选择 Rijndael[1]作为新的加密标准 AES(advanced encryption standard)[2],替代 1977 年制定的 DES 标准.文献[3~5]报道了不同性能的硬件 AES 算法实现及其应用,实现方式采用 ASIC 或者 FPGA,速度从 25 Gbit/s 到 2 Mbit/s 不等.本文关注的是用于低端设备、用 FPGA 方式的 AES 实现,低端应用要求的加、解密速度一般不超过 100 Mbit/s[4],文献[4,5]报道了以较少的资源占用达到 100 Mbit/s 的 AES 实现,但它们的最大速度是在其最大工作频率下获得的,当该设计应用到低时钟频率的设备上时,它们达不到 100 Mbit/s 的设计速度.本文设计了一款在 20 MHz 系统时钟下仍能达到最大 128 Mbit/s 的 AES 模块.

由于 AES 算法的加、解密不对称,大部分 AES 实现采用了分开实现加、解密的方法.AES 加、解密算法硬件实现过程中有很多资源是可以共享的,好的设计能共用更多的资源.本文根据 FPGA 资源的特点,设计了一种用于低端设备的混合加、解密算法,其性能较高,资源占用少,且功耗低.

1 AES 算法

1.1 AES 算法基本操作简介

AES(Rijndael)算法的数学操作是定义在有限域 GF(2^8)上的.AES 规定了三种分组长度 128,192 和 256,本文实现的是 128 bit.AES 算法将 128 bit 数据表达为 4×4 的字节矩阵,称为状态,算法操作用矩阵描述.

收稿日期:2006-06-29;**修回日期**:2006-10-23

作者简介:张德学,男,1977 年生,博士生/助教.研究方向:集成电路设计与应用研究.E-mail:dxzhang@ustc.edu

通讯作者:郭立,教授.E-mail:lguo@ustc.edu.cn

加密由四个操作组成：密钥加 addroundkey (ARK)，字节置换 Subbytes，行移 shiftrows，列混 mixcolumns(MC). 解密由四个类似操作组成：密钥加 addroundkey，逆字节置换 InvSubbytes，逆行移 Invshiftrows，逆列混 Invmixcolumns(IMC).

(Ⅰ) ARK 是将当前 state 与密钥 key 加，在有限域 GF(2^8)，加法就是按位异或，硬件很容易实现.

(Ⅱ) Subbytes 是将 state 中的值按字节替换为 Sbox 中的对应值，可通过查询 ROM 方式实现.

(Ⅲ) shiftrows 是按行移动 state 中的字节，在 FPGA 中可简单地通过硬件连线实现，不占用逻辑资源. 在加密算法中，Subbytes 与 shiftrows 总是同时出现，且两者可交换顺序执行，因而，把它们一起简记为 SS.

(Ⅳ) MC 操作比较复杂，对 state 中的每一列分别操作，其操作用矩阵表示为图 1(a)，其乘法、加法均定义在有限域 GF(2^8)上.

$$\begin{bmatrix} a_i' \\ b_i' \\ c_i' \\ d_i' \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} a_i \\ b_i \\ c_i \\ d_i \end{bmatrix}$$

(a) 加密列混

$$\begin{bmatrix} a_i' \\ b_i' \\ c_i' \\ d_i' \end{bmatrix} = \begin{bmatrix} e & b & d & 9 \\ 9 & e & b & d \\ d & 9 & e & b \\ b & d & 9 & e \end{bmatrix} \cdot \begin{bmatrix} a_i \\ b_i \\ c_i \\ d_i \end{bmatrix}$$

(b) 解密列混

图 1 (a)加密列混和(b)解密列混

Fig. 1 Mixcolumns (a) and Invmixcolumns (b)

(Ⅴ) 逆字节置换 InvSubbytes 与 Subbytes 操作类似，但采用的是逆置换盒 InvSbox.

(Ⅵ) 逆行移 Invshiftrows 与 shiftrows 操作类似，移动方向相反. 同样将 InvSubbytes 和 Invshiftrows 简记为 ISIS.

(Ⅶ) IMC 与 MC 操作类似，其每一列的操作如图 1(b).

1.2 AES 加解密流程

加、解密分别是用基本函数组成轮函数，经过 10 轮操作后给出结果. 加密算法表达为(ARK, 9{SS, MC, ARK}, SS, ARK)；解密算法表达为(ARK, 9{ISIS, ARK, IMC}, ISIS, ARK).

加、解密算法的区别有：使用密钥 key 序列顺序不同，置换盒不同，行移方向不同，列混系数不同，基本函数使用顺序不同[1,2].

2 AES 算法的 FPGA 实现

2.1 基本数学运算的硬件实现

有限域 GF(2^8)上加法与乘法易于用硬件实现. 加法就是异或操作；乘法可以实现一个乘 2 操作，其余操作可以转换为多级乘 2 操作与加法操作组合来实现. 乘 2 操作表达为

$$2 \cdot \{b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0\} = \begin{cases} \{b_6 b_5 b_4 b_3 b_2 b_1 b_0 0\}, & b_7 = 0 \\ \{b_6 b_5 b_4 b_3 b_2 b_1 b_0 0\} \mathrm{XOR} \{00011011\}, & b_7 = 1 \end{cases} \tag{1}$$

乘 2 操作可用移位、选择、异或来实现. 这是一个基本的操作，可由硬件简单实现，简记为 X，则 $X(B)=2 \cdot B$. 以数值 0xb 乘 B 为例，说明乘法操作转换为 X 与加法的组合如下

$$b \cdot B = \{1011\} \cdot B = \{1000 + 0010 + 0001\} \cdot B = X^3(B) + X(B) + B \tag{2}$$

2.2 总体框图

加、解密算法虽类似，但对应的每个函数几乎都有差别，因而文献中 AES 的加、解密算法大都分开实现，很少是混合实现[6]的. 本文认为，设计加、解密算法的混合实现可以共享硬件资源，得到更小面积的 AES 实现. 图 2 是本文设计的 AES 算法数据通路图，E/D 是 encrypt/decrypt 信号，round 指示轮数，通过简单的附加逻辑，将加、解密算法混在一个流程中实现，每一个功能函数只需要实现一次，减少

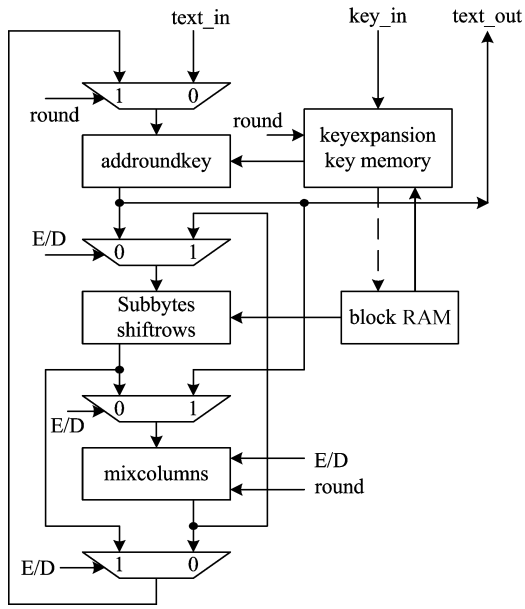

图 2 数据通路

Fig. 2 Datapath

了资源占用,加、解密算法共享同一个主控状态机,各功能函数实现时也考虑共享实现. 加密且 round 为 0 时,跳过列混部件;解密且 round 为 9 时,跳过逆列混部件,这样就能正确地实现加、解密操作顺序.(图 2 中虚线表示是否采用 block RAM 来实现密钥存储,见下文 2.6)

2.3　字节置换操作实现

从性能考虑,字节置换采用预先计算好置换数值后存入表 Sbox/InvSbox 中,需要时查表来实现. 并行计算时,加、解密各需要 16 个置换盒,每个置换盒是 8 bit 输入 8 bit 输出的 ROM,需要 256×8 bit 存储,恰是 FPGA 的 4 kbit block RAM 的一半. 若单独实现加、解密,则共需要占用 32 个 block RAM,采用混合方式,可以将 block RAM 配置为 8 bit 输入 16 bit 输出,16 bit 的输出高低 8 bit 分别表示 Sbox 和 InvSbox 的输出,充分利用了 block RAM,只需要占用 16 个 block RAM. 考虑到密钥扩展也需要四个置换盒,本设计将密钥扩展需用的置换盒与加、解密使用的置换盒分时复用,节省了 block RAM 资源,字节置换部件的框图如图 3,S 和 I 分别为 Sbox 和 InvSbox,两者在同一个 block RAM 中实现;E/D 标识加密或是解密,用于选择对应的置换输出字节;key_done 是密钥扩展是否完成的标志.

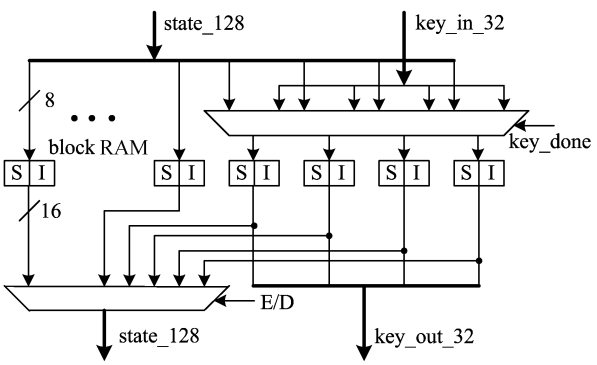

图 3　字节置换

Fig. 3　Byte substitute

2.4　列混操作实现

加密时 state 中的每一列的列混操作如图 1(a)所示,其硬件实现可表达为

$$\begin{aligned} a' &= 2\cdot a+3\cdot b+c+d = \\ &\quad 2\cdot a+(2\cdot b+b)+c+d = \\ &\quad X(a+b)+b+(c+d) \end{aligned} \tag{3}$$

$$\begin{aligned} b' &= a+2\cdot b+3\cdot c+d = \\ &\quad a+2\cdot b+(2\cdot c+c)+d = \\ &\quad X(b+c)+c+(a+d) \end{aligned} \tag{4}$$

$$\begin{aligned} c' &= a+b+2\cdot c+3\cdot d = \\ &\quad a+b+2\cdot c+(2\cdot d+d) = \\ &\quad X(c+d)+d+(a+b) \end{aligned} \tag{5}$$

$$\begin{aligned} d' &= 3\cdot a+b+c+2\cdot d = \\ &\quad (2\cdot a+a)+b+c+2\cdot d = \\ &\quad X(a+d)+a+(b+c) \end{aligned} \tag{6}$$

式中,"( )"内的操作在硬件实现时可以共用.

解密时的列混操作如图 1(b)所示,系数为 1 的个数很多,按上式方法展开,需要很多的 X 和异或操作. 但它的实现有更好的方法[7],如下式

$$\begin{bmatrix} a'_i \\ b'_i \\ c'_i \\ d'_i \end{bmatrix} = \begin{bmatrix} e & b & d & 9 \\ 9 & e & b & d \\ d & 9 & e & b \\ b & d & 9 & e \end{bmatrix} \cdot \begin{bmatrix} a_i \\ b_i \\ c_i \\ d_i \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 5 & 0 & 4 & 0 \\ 0 & 5 & 0 & 4 \\ 4 & 0 & 5 & 0 \\ 0 & 4 & 0 & 5 \end{bmatrix} \cdot \begin{bmatrix} a_i \\ b_i \\ c_i \\ d_i \end{bmatrix} \tag{7}$$

注意到系数中有两个为 0,且系数 5,4 的二进制表示中最多有两个是 1,可以高效地用硬件表达为

$$\left.\begin{aligned} a' &= 5\cdot a+4\cdot c = X^2(a+c)+a \\ b' &= 5\cdot b+4\cdot d = X^2(b+d)+b \\ c' &= 4\cdot a+5\cdot c = X^2(a+c)+c \\ d' &= 4\cdot b+5\cdot d = X^2(b+d)+d \end{aligned}\right\} \tag{8}$$

新方法能有效地降低硬件复杂度,并可与加密时的列混操作复用. 用一个辅助模块 aux_mix_32bit 实现式 5 的操作,则列混操作的内部实现如图 4 所示,32 bit 的列混操作所需的计算数量统计见表 1.

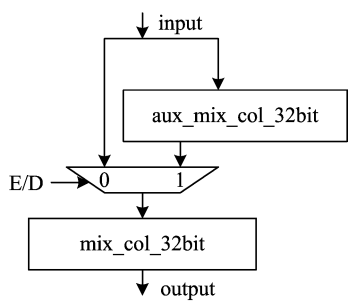

图 4　列混操作

Fig. 4　Mixcolumns operation

实现 128 bit 列混操作需要并行的四个 32 bit 部件,实现 aux_mix_col_32bit 需要 40LCs(240 门),实现 mix_col_32bit 需要 48LCs(288 门),实现 X 需要 3LCs(18 门). 用混合方式实现加、解密列混操作的资源占用从 550LCs 降到了 369LCs.

表 1 计算操作数量表

Tab. 1 Table of operation count

(32 bit)	byte 异或	byte 乘 2
单独加密模块	12	4
单独解密模块	23	15
加解密总和	35	19
解密辅助模块	6	4
混合加解密模块	18	8
节省	17	11

2.5 密钥扩展的实现

密钥扩展计算是一次性操作,需要四个 Sbox,单独为一次性的操作增加四个 Sbox 的存储,代价过高,且解密过程是逆序使用密钥,必须先计算出全部的密钥. 本文对密钥的设计是:先计算出密钥序列,保存到密钥存储中,进行加、解密计算时,统一从密钥存储中存取需要的密钥,这种设计使密钥计算部件能与加密部件分时共享四个 Sbox(见图 3),节省资源,加、解密能共用同一个主控状态机控制. 每一轮的密钥计算如图 5 所示.

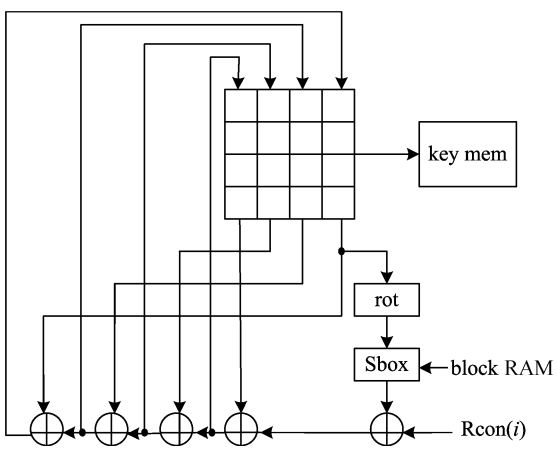

图 5 单轮密钥计算

Fig. 5 Key expansion

在密钥计算过程中,也需要使用置换盒 Sbox,而 Sbox 是用 block RAM 实现的,访问 block RAM 是同步操作,需要等待一个时钟周期,每一轮的密钥计算要两个时钟周期.

2.6 密钥存储的实现

密钥访问操作频繁,因而密钥存储的实现方式需要重点考虑. FPGA 中的存储资源有两类:block RAM 和分布式 RAM. block RAM 是专用 RAM 资源,是同步器件,设置读信号后,需要一个时钟后才能得到数据. 分布式 RAM 用 LUT 实现,可以异步读,适合实现小容量、快速访问的 ROM. 单个密钥是 128 bit 的存储扩展密钥需要 $128 * 11 = 1\ 408$ bit. 若用 block RAM 实现,由于端口太宽,实际实现时需要分布在 8 个 block RAM 中实现. 若用分布式 RAM 实现,需要 128 个双口 RAM16X1D 来实现,每个 RAM16X1D 需要两个 LUT 实现,共需要 256LCs. 不同的密钥存储实现方式由于访问周期不同,会影响主控状态机. 本文分别用两种方式实现了密钥存储,并给出实验结果,见表 2.

表 2 实现结果表

Tab. 2 Result of implementation

	block RAM 实现密钥存储	分布式 RAM 实现密钥存储
密钥计算周期	22	22
加解密周期	30	20
资源 LC	1 286	1 548
block RAM	24	16
频率 f/MHz	50.1	39.7
功耗 $f=0$ 时	34 mW	34 mW
功耗 $f=f_{max}$ 时	166 mW	141 mW
最大速率	123.3/213.7 Mbit/s	121/254 Mbit/s

【注】 表中速率数值,首个表示需要计算密钥时的速率,第二个表示不需要计算密钥时的速率.

2.7 主控状态机的实现

本文的设计对加、解密过程统一采用先计算密钥,再执行加、解密的方式,采用状态机控制流程. 对每一个密钥的计算只需要做一次,再次使用同一密钥加、解密时,直接跳过密钥计算状态,当加、解密数据量很大时,密钥计算所占用的周期可以忽略.

针对密钥存储,分别用 block RAM 和分布式 RAM 实现. 分布式 RAM 可以异步读,因而比 block RAM 实现方式少一个时钟周期,同频率下工作,性能要好一些. 采用 block RAM 实现时,每一轮需三个时钟周期,分别完成:读取密钥、等待置换完成、结果反馈;采用分布式 RAM 实现时,每一轮需要两个时钟周期,分别完成:等待置换完成、结果反馈.

2.8 实现结果

实现的目标器件是 Xilinx 的 XC2S400E. 用 Synplify Pro 综合,Xilinx ISE 实现,Xpower 分析功耗.

从实现结果来看,用分布式 RAM 实现密钥存储,要多占用一些 LC,但能少占用 8 个 block RAM,加、解密需要更少的周期,与 block RAM 实现方式相比,同频率下加、解密速度更快.

文献[4,5]中的设计采用32 bit为单位实现(本

表 3　不同实现的比较

Tab. 3　Comparison of different implementations

	文献[4]	文献[5]	本文分布式 RAM 方式
资源	222slices	163slices	774slices
block RAM	3 个 4 kbit,共 9 600 bit	3 个 18 kbit,共 34 176 bit	16 个 4 kbit,共 65 536 bit
加密周期(首次/随后)	84/40	88/44	42/20
解密周期(首次/随后)	84/40	92/44	42/20
最高频率 MHz	50	71.5	39.7
最大加密速率	76.2/160	104/208	121/254
最大解密速率	76.2/160	99.5/208	121/254
block RAM 等效 slice	300slice	1 068slice	2 048slice
总 slice	522	1231	2 822
最大速率/面积(slice)	0.306	0.169	0.090
20 MHz 时加密速率	30.5/64	29/58.2	60.9/128
20 MHz 时解密速率	30.5/64	27.8/58.2	60.9/128
20 MHz 时最大速率/面积(slice)	0.123	0.047	0.045
20 MHz 时功耗	N/A	N/A	134 mW

【注】 上表采用文献中的在 Spartan 系列芯片实现 128 bit 加解密时的结果,数据采用原始数据,1slice=2LC,1slice=32 bit 存储.

文的设计是采用 128 bit 为单位实现),完成 128 bit 数据加、解密需要四次 32 bit 加解密运算,文献考虑了用 32 bit 实现时的数据相关性,给出了资源占用和最高加、解密速率. 文献[4]的设计速率目标为 100 Mbit/s,其最高速率计算公式为最高频率/加(解)密周期×分组长度. 本文认为,从学术角度考虑,上式很合理,但从应用角度来看,有些不妥. 其设计出的 AES 硬件模块需要接入到应用系统中,一般是低端应用,其系统时钟通常不高,在采用低端设备的系统时钟时的加、解密速率更有应用意义. 本文以系统时钟 20 MHz 为例,对比本设计与其他类似设计,结果见表 3.

从表 3 中的数据可以看出,本文的设计在低端设备上能提供更好的性能,更有实用价值.

3　结论

本文设计了一种用于低端设备、功耗低的 AES 硬件模块. 采用加、解密算法混合设计,使共享资源最大化,各功能部件仅实现一次,有效地减少了资源占用;将加密置换盒与解密置换盒集成到同一个 block RAM 中实现,充分利用了 block RAM;使用改进的解密列混操作,与加密列混操作共用部分硬件;对加、解密过程统一采用先计算密钥再进行加、解密的设计,用同一个状态机控制加、解密过程;分别用 block RAM 和分布式 RAM 实现了密钥存储,并对结果进行了对比. 同其他设计相比较,本文设计在较低的时钟频率下,仍保持较高的性能,20 MHz 时,加解密速度均可达 128 Mbit/s,更有实用价值.

参考文献(References)

[1] Joan D, Vincent R. AES proposal: Rijndael (2nd version) [EB/OL]. http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf.

[2] National Institute of Standards and Technology. Advanced encryption standard [EB/OL]. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[3] Tim G, Mohammed B. AES on FPGA from the fastest to the smallest [C]// Proceedings of CHES 2005. Springer, 2005: 427-441.

[4] Chodowiec P, Gaj K. Very Compact FPGA Implementation of the AES Algorithm [M]. Heideeberg: Springer-Verlag. 2003.

[5] Gael R, Francois-Xavier S, Jean-Jacques Q, et al. Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications [C]// Proceedings of the International Conference on Information Technology: Coding and Computing. 2004,2: 583-587.

[6] Lu C C, Tseng S Y. Integrated design of AES (advanced encryption standard) encrypter and decrypter [J]. IEEE Transactions on Information Theory, 1991,37(5): 1 241-1 260.

[7] Rodriguez-Henriquez F, Saqib, N A, Diaz-Perez A. 4.2 Gbit/s single-chip FPGA implementation of AES algorithm [J]. Electronics Letters, 2003, 39(15): 1 115-1 116.