doi:10.11835/j.issn.1000-582X.2014.06.009

Verilog HDL 语言的 AES 密码算法 FPGA 优化实现

李 浪[1,2],邹 祎[1],李仁发[2],李肯立[2]
(1.衡阳师范学院 计算机科学系,湖南 衡阳 421002;2.湖南大学 信息科学与工程学院,长沙 410082)

摘 要:AES 密码算法是目前广泛使用的一种加密算法。为了对 AES 算法进行优化,通过对密钥扩展模块重复调用,实现代码的高效利用。具体方法为在 AES 算法进行加解密运算时,其中所需的密钥可在其他模块执行时重复调用,即一次生成十轮密钥,通过控制模块实现轮密钥加运算。详细叙述了改进后 AES 算法的 Verilog HDL 硬件语言实现,特别是对具体实现过程中关键核心代码进行了清晰描述,经 modelsim6.1f 仿真验证正确后进行了 FPGA 硬件实现,对 FPGA 硬件实现进行了实验结果正确性验证。实验结果表明,优化后的 AES 算法在 Xilinx Virtex-V FPGA 上仅占用了 3 531 个 Slice,5 522 个 LUT,与同类加密算法实现所需的资源数对比,在性能同等条件下占用面积更少,可满足芯片的较小面积应用需求,从而可以使得 AES 算法应用于目前流行的各种小面积智能卡上。

关键词:AES 算法;Verilog HDL;FPGA 实现
中图分类号:TP309.7 **文献标志码**:A **文章编号**:1000-582X(2014)06-056-09

FPGA optimal implementation of AES based on Verilog HDL

LI Lang[1,2], *ZOU Yi*[1], *LI Renfa*[2], *LI Kenli*[2]
(1. Department of Computer Science, Hengyang Normal University, Hengyang, Hunan 421002, China; 2. College of Information Science and Engineering, Hunan University, Changsha 410082, China;)

Abstract: AES algorithm is a widely used cryptographic algorithm. To improve AES algorithm, it's proposed to repeatedly call key expansion module to realize efficient use of the code. Ten-round keys are generated at the same time, and operations of add round key are achieved by the control module. The key is called repeatedly when the AES algorithm is running for encryption and decryption. The realization of AES is verified by modelsim6.1f. AES algorithm is designed with Verilog HDL, and a clear description about the critical core code realization of the process is proposed. The hardware implementation is verified by FPGA. Experimental results show that the optimized AES algorithm has only 3 531 slices, 5 522 LUTs on a Xilinx Virtex-V FPGA. Our implementation occupies less area and it can get the same performance with comparing with other implementations of the AES, so it can meet application requirements of smaller chip, which can

收稿日期:2013-10-11
基金项目:国家自然科学基金资助项目(61133005);湖南省教育厅青年资助项目(11B018);湖南省博士后基金资助项目(897203005);衡阳师范学院产学研基金项目(12CXYZ01)。
作者简介:李浪(1971-),男,教授,博士后,主要从事信息安全方向研究,(Tel)15873438955;(E-mail)lilang911@126.com。
李仁发(1957-),教授,博士生导师,主要研究方向为嵌入式计算。
李肯立(1971-),教授,博士生导师,主要研究方向为高性能计算与信息安全。

make the AES algorithm be applied to the popular small area on the smart card. It can make the AES algorithm use in the smart card.

Key words: AES(advanced encryption standard) algorithm; Verilog HDL; FPGA implementation

高级加密标准 AES(advanced encryption standard)于 2001 年提出，用于取代早期 DES 加密算法，并逐渐成为对称密码算法中最流行的一种[1]。AES 算法目前有多种实现方案，尤以软件实现方案为众。在大多数软件实现方案中，由于考虑算法是在个人电脑上运行，加密速度和加密数据的大吞吐量方面成为优先考虑的指标，并没有考虑 AES 密码算法在智能卡硬件实现时所需的低功耗、低资源、小面积特征[2-4]。同时，网上也有基于硬件描述语言 Verilog HDL 的 AES 代码资源，可能是因为知识产权的原因，一些关键核心代码缺少或错误，大都不能正确运行。

笔者详细叙述了 AES 算法的 Verilog HDL 硬件语言实现，特别是对具体实现过程关键核心代码进行了清晰描述，并对 AES 算法的实现通过模块间调用，进而实现代码的高效利用，降低芯片面积；经 modelsim6.1f 仿真验证正确后进行了 FPGA 硬件实现，

对 FPGA 硬件实现进行了实验结果正确性验证。模块间的重复调用可以降低 AES 的硬件实现面积，从而可以使得 AES 算法应用于目前流行的各种小面积智能卡上，给信息时代安全用卡研究提供一定参考价值[5]。

1　AES 算法简介

1.1　相关定义

以下定义引用于文献[1]。

定义 1　一个由 $b_7b_6b_5b_4b_3b_2b_1b_0$ 组成的字节 b 可以表示成系数为[0,1]的二进制多项式 $b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b_0$。

定义 2　$GF(2^8)$上的加法定义为二进制多项式加法，系数满足模 2 加(异或)。

定义 3　$GF(2^8)$上的乘法定义为二进制多项式乘积以 8 次不可约多项式为模的积，该 8 次不可约多项式：$m(x)=x^8+x^4+x^3+x+1$。

定义 4　函数 $x$time 定义为 $GF(2^8)$上的 $x\cdot b(x)$，若 $b_7=0$，字节 b 左移一位；$b_7=1$，则字节 b 左移一位再“异或”“0x1B”。

定义 5　一个固定多项式 $a(x)=a_3x^3+a_2x^2+a_1x+a_0$ 与多项式 $b(x)=b_3x^3+b_2x^2+b_1x+b_0$ 相乘构成的运算可以用矩阵乘法表述

$$\begin{pmatrix}c_0\\c_1\\c_2\\c_3\end{pmatrix}=\begin{pmatrix}a_0&a_3&a_2&a_1\\a_1&a_0&a_3&a_2\\a_2&a_1&a_0&a_3\\a_3&a_2&a_1&a_0\end{pmatrix}\begin{pmatrix}b_0\\b_1\\b_2\\b_3\end{pmatrix},\tag{1}$$

其中，方阵是一个循环矩阵，在 AES-Rijndael 密码算法中，乘法算法值限于乘一个固定有逆元的多项式，从而保证乘法的可逆性。

1.2　AES 算法

AES 是一个迭代型密码算法，有 128 位的分组长度，三种可选密钥长度：128 位、192 位、256 位。组成一个 4 行、N_b列(N_b＝数据块长/32)的二维数阵列，密钥设计为 4 行，N_k列(N_k＝密钥块长/32)，算法变换的圈数 N_r由 N_b、N_k共同决定，其加解密运算过程如图 1。

论文以密钥长度为 128 位为例，需要进行十轮变换，前九轮的变换过程相同，依次：字节替换、行移位、列混合和密钥加，第十轮变换跳过列混合变换。

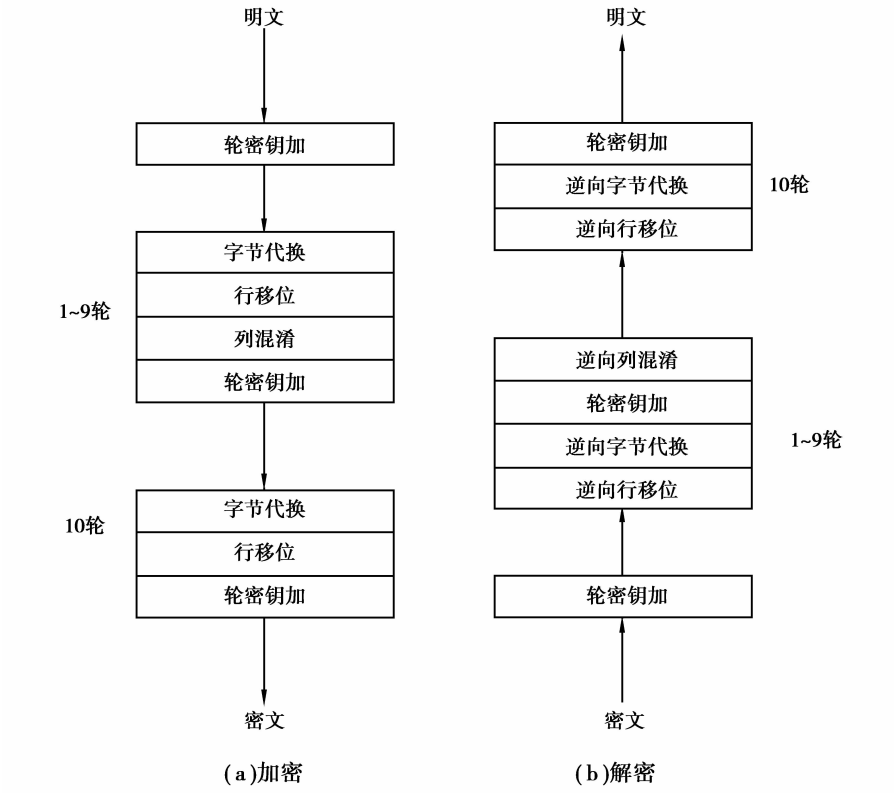

图 1 AES 加解密过程示意图

2 AES 算法的 Verilog HDL 语言 FPGA 优化实现

2.1 优化方法

图 2 是 AES 算法 Verilog HDL 硬件描述语言实现的总体架构流程[6-7]。

通过总体设计流程图可以发现，加密及解密的过程基本上是一个过程相同但方向相反的过程。虽然初始化和第 10 轮与中间的 9 轮过程有所区别，但是整个设计过程可分为字节替换，行移位，列混合与密钥扩展 4 个模块实现，其中密钥扩展模块将输出每轮加/解密中所需要的轮密钥，而轮密钥的调用是顺序完成的，因此十轮加/解密中所需的密钥可在其他模块执行的同时重复调用，并一次生成十轮密钥，最后通过控制模块实现轮密钥加运算。

通过密钥扩展模块的重复调用，可以实现代码的高效利用，关键是节省了密码芯片的实现面积，适合于智能卡类芯片加密应用。优化后的结构流程图如图 3。

图 4 为 AES 算法实现的具体工作流程。

在本设计中存储的 128 位数用 16 进制表示，存储位如表 1 所示。

表 1 存储方式示意表

S00(1—8)	S01(33—40)	S02(65—72)	S03(97—104)
S10(9—16)	S11(41—48)	S12(73—80)	S13(105—112)
S20(17—24)	S21(49—56)	S22(81—88)	S23(113—120)
S30(25—32)	S31(57—64)	S32(89—96)	S33(121—128)

2.2 字节替换

下面的 Verilog HDL 语言是实现了一个查找表，s 盒是一个 16×16 的矩阵，大小为 256 个字节。

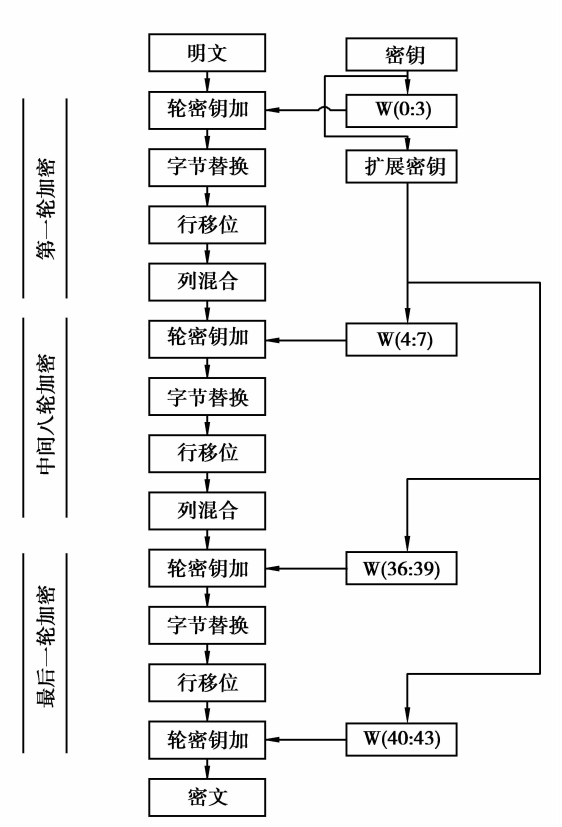

图 2　AES 原流程图

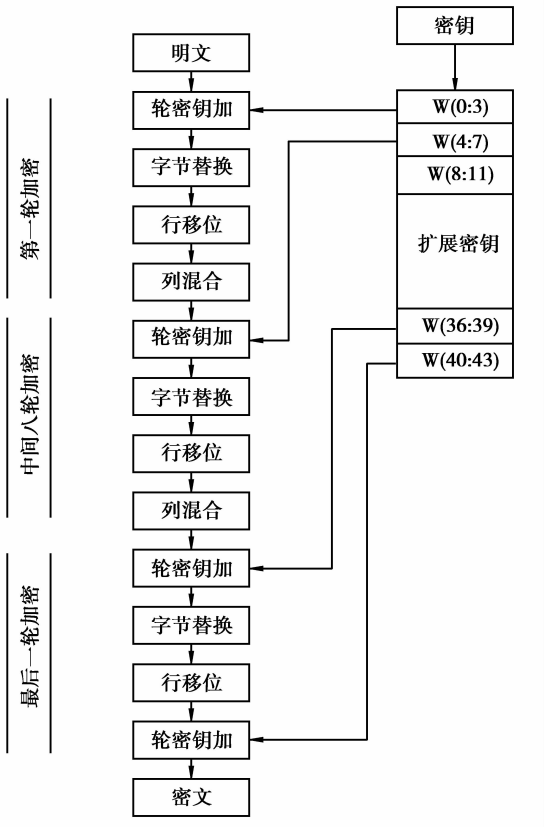

图 3　优化后的 AES 实现流程图

```
always @(a)
    case(a)
      8'h00:d=8'h63;
      8'h01:d=8'h7c;
      ……
      8'hff:d=8'h16;
    endcase
```

测试数据如下：

```
#5 a=8'h3b;
#10 a=8'h00;
#20 a=8'h26;
```

测试后的仿真波形如图 5。

从图 4 的波形分析可以看出每一个时钟信号的上升沿到来时，进行一轮变换，同时 counter 减 1。字节变换、行移位后产生结果用 aa 表示送入列混合模块，产生的结果 s_mix 与该轮密钥进行异或操作。

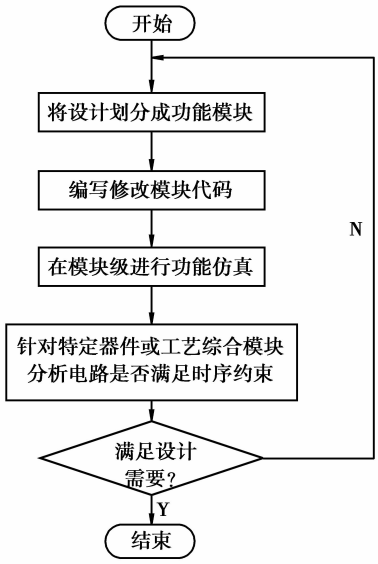

图 4　工作流程

2.3　列混合

要实现

$$\boldsymbol{S}'_{00} = (02 \quad 03 \quad 01 \quad 01)\begin{pmatrix} S_{00} \\ S_{10} \\ S_{20} \\ S_{30} \end{pmatrix},$$

即

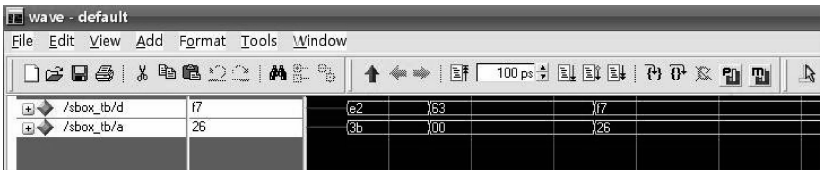

图 5 S盒仿真波形

$$\boldsymbol{S}'_{00} = \{02\}\ \boldsymbol{S}_{00} \oplus \{03\}\ \boldsymbol{S}_{10}\ \boldsymbol{S}_{20} \oplus \{01\}\ \boldsymbol{S}_{30},$$

而 02^$\boldsymbol{S}_{00}$ 可以表示为 $\boldsymbol{S}_{00}$ 的移位操作(可参考定义 4),用 bb[1:8]代替 $\boldsymbol{S}_{00}$,bb[1]表示 $\boldsymbol{S}_{00}$ 的最高位。判断 $\boldsymbol{S}_{00}$ 最高位是否为 1,若为 1,将 $\boldsymbol{S}_{00}$ 左移一位后异或 1b;若不为 1,仅将 $\boldsymbol{S}_{00}$ 左移一位。

详细的 Verilog HDL 描述代码如下

```
function [31:0] mix_col;
input[7:0]s0,s1,s2,s3;
reg[7:0]s0_o,s1_o,s2_o,s3_o;
begin
mix_col[31:24]=xtime(s0)^xtime(s1)^s1^s2^s3;
mix_col[23:16]=s0^xtime(s1)^xtime(s2)^s2^s3;
mix_col[15:08]=s0^s1^xtime(s2)^xtime(s3)^s3;
mix_col[07:00]=xtime(s0)^s0^s1^s2^xtime(s3);
end
endfunction

function [7:0] xtime;
input [7:0] b;xtime={b[6:0],1'b0}^(8'h1b&{8{b[7]}});
endfunction
```

等同于

$$\boldsymbol{S}'_{00} = ((\mathrm{S}[1]=1)?(\boldsymbol{S}_{00} << 1)\ 8'\mathrm{h1b}:(\boldsymbol{S}_{00} << 1))\hat{}((\mathrm{S}[9]=1)?$$
$$(\boldsymbol{S}_{10} << 1)\ \boldsymbol{S}_{10}\ 8'h1b:(\boldsymbol{S}_{10} << 1)\ \boldsymbol{S}_{10}\ \boldsymbol{S}_{20}\ \boldsymbol{S}_{3}。 \quad (4)$$

Xtime 函数作用是防止溢出的发生,通过移位异或操作,可以有效的防止高位溢出。

2.4 密钥扩展

每一轮的密钥都是由前一轮的密钥和本轮密钥共同产生的。

$$w_i = w_{i-4} \oplus \mathrm{Sub}(\mathrm{RotByte}(w_{i-1})) \oplus R\mathrm{con}, \quad (5)$$
$$w_{i+1} = w_{i-3} \oplus w_i, \quad (6)$$
$$w_{i+2} = w_{i-2} \oplus w_{i+1}, \quad (7)$$
$$w_{i+3} = w_{i-1} \oplus w_{i+2}, \quad (8)$$

密钥扩展的状态转移图如图 6 所示。

为实现十轮密钥同时在模块中生成,需重复调用密钥扩展模块,并将生成的十轮密钥通过计数控制依次进行轮密钥加运算。

详细的 Verilog HDL 描述代码如下:

```
always @(posedge clk)w[0]<=reset? key[1:32]:(ready? w[0]:w[0]^sub^rcon);
always@(posedge clk) w[1]<=reset? key[33:64]:(ready? w[1]:w[0]^w[1]^sub^rcon);
always@(posedge clk) w[2]<=reset? key[65:96]:
        (ready? w[2]:w[0]^w[2]^w[1]^sub^rcon);
always @(posedge clk)w[3]<=reset? key[97:128]:(ready? w[3]:
        w[0]^w[3]^w[2]^w[1]^sub^rcon);
```

```
always@(posedge clk)
begin
case(count2)
  4'b1001:begin
  outcopy[0]= {w[0],w[1],w[2],w[3]};
  out<=outcopy[0];
  count2<=4'b1000;
  end
  4'b1000:begin
  outcopy[1]= {w[0],w[1],w[2],w[3]};
  out<=outcopy[1];
  count2<=4'b0111;
  end
  4'b0111:begin
  outcopy[2]= {w[0],w[1],w[2],w[3]};
  out<=outcopy[2];
  count2<=4'b0110;
  end
  4'b0110:begin
  outcopy[3]= {w[0],w[1],w[2],w[3]};
  out<=outcopy[3];
  count2<=4'b0101;
  end
  4'b0101:begin
  outcopy[4]= {w[0],w[1],w[2],w[3]};
  out<=outcopy[4];
  count2<=4'b0100;
  end
  4'b0100:begin
  outcopy[5]= {w[0],w[1],w[2],w[3]};
  out<=outcopy[5];
  count2<=4'b0011;
  end
  4'b0011:begin
  outcopy[6]= {w[0],w[1],w[2],w[3]};
  out<=outcopy[6];
  count2<=4'b0010;
  end
  4'b0010:begin
  outcopy[7]= {w[0],w[1],w[2],w[3]};
  out<=outcopy[7];
  count2<=4'b0001;
  end
  4'b0001:begin
  outcopy[8]= {w[0],w[1],w[2],w[3]};
```

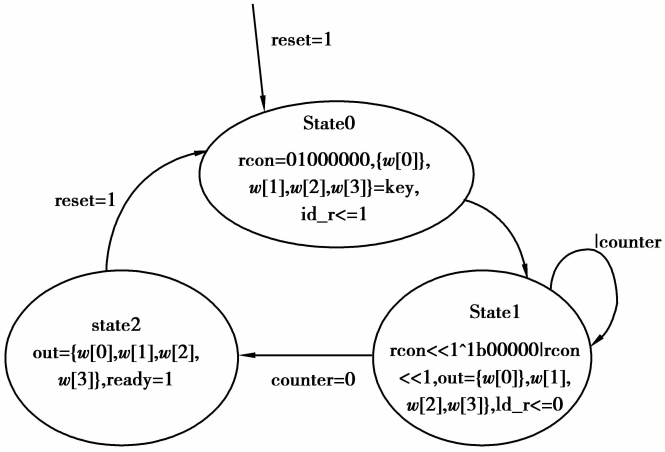

图 6　状态转移图

```
    out<=outcopy[8];
    count2<=4'b0000;
    end
    4'b0000:begin
    outcopy[9]= {w[0],w[1],w[2],w[3]};
    out<=outcopy[9];
    count2<=4'b1001;
    end
    endcase
  end
```

同样采取大家一般使用的测试初始密钥值(2b7e151628aed2a6abf7158809cf4f3c)进行测试代码实现正确性。得出的测试前 3 轮预期结果如下图 7 所示,用 moelsim6.1f 仿真上述代码结果如图 8,其中仿真波形如图 9。

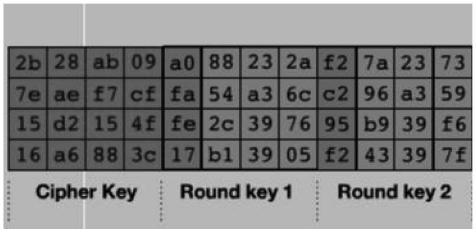

图 7 测试数据

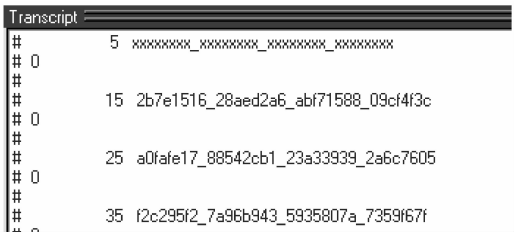

图 8 命令窗口下的结果输出

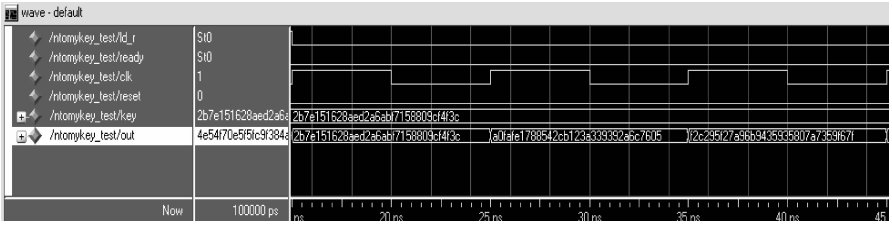

图 9 波形观察器中的结果输出

对图 9 的波形进行分析可以发现,本模块并不设定计算轮数,由最终的顶层控制最后的运算次数,因此此处不需要初始化 counter 值。

1)置初值,将初始密钥传给 $w[0]\sim w[3]$,ld_r 信号开始设置初值 0→1→0,作为一个信号量,表示 key 模块可以开始运行。

2)通过异或操作产生新的密钥,同一轮的密钥同时产生。如果计数器不为 0,ready 信号为 1,说明没有到第 10 轮密钥产生,保存当次产生的密钥,计数器减 1,同时进入下一轮密钥扩展。

3)计数器为 0,ready 信号为 0,所有 10 轮密钥产生完毕。

其中用到的 S 盒查找部分由以下代码完成

```
sbox u0(.a(tmp_w[23:16]),d(subword[31:24]));
sbox u1(.a(tmp_w[15:08]),d(subword[23:16]));
sbox u2(.a(tmp_w[07:00]),d(subword[15:08]));
sbox u3(.a(tmp_w[31:24]),d(subword[07:00]));
```

2.5 FPGA 实现

对 Verilog HDL 优化实现的 AES 算法使用 ISE9.1 进行综合仿真,并最终利用 EDK9.1 生成位流文件后,下载到 Xilinx Virtex-V FPGA 板上进行硬件实现正确性的验证[10]。

实验中用于加密验证的数据如下

原文(state):3243f6a8 885a308d 313198a2 e0370734

密钥(key):2b7e1516 28aed2a6 abf71588 09cf4f3c

密文(result):3925841d 02dc09fd dc118597 196a0b32。

利用开发平台支持的第三方工具 Modelsim 6.1f 仿真结果如图 10 所示

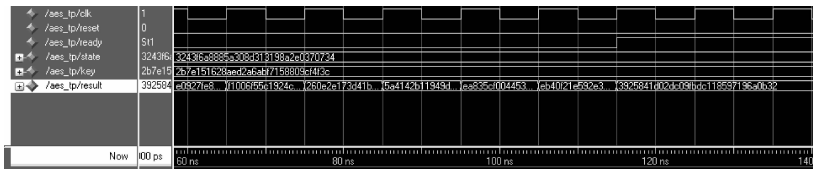

图 10　波形观察器中的加密输出结果图

然后,在 EDK 环境下添加自定义 IP 核,为 FPGA 设置外部管脚,生成可以下载到目标板的硬件比特流文件;最后,通过 JTAG 线连接到 FPGA,下载运行,通过超级终端显示运行结果。其结果如图 11 所示。从实验结果图中可以得出代码描述与实现正确。

该设计在 Xilinx 公司的 Virtex-V FPGA 上综合下载实现,其硬件配置如图 12 所示,主要逻辑资源的消耗情况如图 13 所示,仅占用了 3 531 个 Slice,5 522 个 LUT 和 2 057 个 FLIP FLOPS。同时根据图 12 的时钟频率可以由公式计算加密速率为:throughput = b_length × number_of_b_sametime/latency = 128 bits/10 ns = 12.8 Gb/s。,根据资源报告图 13 并与同类加密算法实现所需的资源数对比[11],对比结果见表 2,进一步验证了提出的 AES 密码优化算法的优越性,可满足较小的芯片面积应用需求,同时加密性能仍然较高。

表 2　FPGA 逻辑资源使用情况对比

实验设备	Slice	LUT	Flip Flops
Xilinx XC3s500e	4 230	7 222	1 758
Xilinx XC5vlx50t	3 531	5 522	2 057

3　结　语

研究对 AES 的 Verilog HDL 硬件描述语言实现进行了实验验证,对模块设计进行了优化,从而可以重复调用,用以减少 AES 硬件资源从而小面积实现,因此可以适应智能卡上的 AES 加密应用,所有代码均进行了 ModelSim 6.1f 验证其正确性,并用综合软件 ISE9.1 下载位流到 FPGA 上进行原型验证硬件功能正确性。

后续工作是在此基础上进行抗旁路攻击的 AES 密码算法 Verilog HDL 代码实现与 FPGA 硬件原型验证,从而使智能卡上的 AES 密码算法能够达到国标要求的抗旁路攻击安全性。

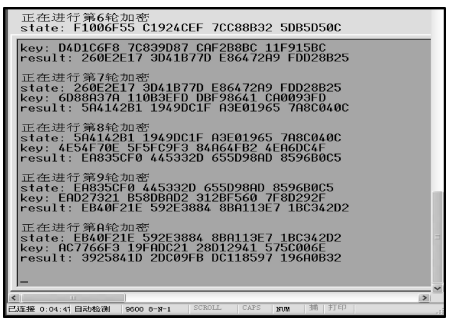

图 11　AES 加密结果

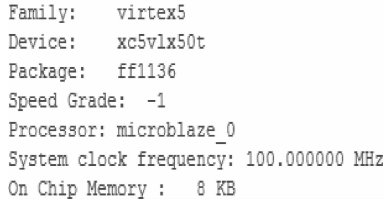

图 12　硬件配置图

```
Logic Utilization:
  Total Number Slice Registers:     2,065 out of  10,944   18%
    Number used as Flip Flops:                2,057
    Number used as Latches:                       8
    Number of Slice FFs used for
    DCM autocalibration logic:          7 out of   2,057    1%
  Number of 4 input LUTs:           5,185 out of  10,944   47%
    Number of LUTs used for
    DCM autocalibration logic:          4 out of   5,185    1%
      *See INFO below for an explanation of the DCM autocalibration logic
       added by Map
Logic Distribution:
  Number of occupied Slices:                        3,531 out of   5,472   64%
    Number of Slices containing only related logic:   3,531 out of   3,531  100%
    Number of Slices containing unrelated logic:          0 out of   3,531    0%
      *See NOTES below for an explanation of the effects of unrelated logic
Total Number of 4 input LUTs:          5,522 out of  10,944   50%
  Number used as logic:              5,185
  Number used as a route-thru:          66
  Number used for Dual Port RAMs:      208
    (Two LUTs used per Dual Port RAM)
  Number used as Shift registers:       63
  Number of bonded IOBs:                 4 out of     320    1%
  Number of BUFG/BUFGCTRLs:              1 out of      32    3%
    Number used as BUFGs:                1
    Number used as BUFGCTRLs:            0
  Number of FIFO16/RAMB16s:             32 out of      36   88%
    Number used as FIFO16s:              0
    Number used as RAMB16s:             32
  Number of DCM_ADVs:                    1 out of       4   25%
  Number of PPC405_ADVs:                 1 out of       1  100%
  Number of JTAGPPCs:                    1 out of       1  100%
```

图 13 优化后的 AES 算法 FPGA 下载资源报告

参考文献：

[1] Daemen J, Rijmen V. The design of Rijndael: AES - the advanced encryption standard[M]. Berlin: Springer, 2002.

[2] 何德彪，胡进，陈建华. 基于 FPGA 的高速 AES 实现[J]. 华中科技大学学报：自然科学版，2010，38(2)：101-104.
HE Debiao, HU Jin, CHEN Jianhua. Implementation of AES-128 using FPGA[J]. Journal of Huazhong University of Science and Technology: Natural Science Edition, 2010, 38(2): 101-104.

[3] 王赜坤，陈松涛. 一种 AES 密码算法的硬件实现[J]. 现代电子技术，2010，33(16)：10-13.
WANG Zekun, CHEN Songtao. Hardware implementation of AES cipher aalgorithm[J]. Modern Electronics Technique, 2010, 33(16): 10-13.

[4] 叶剑，李立新. 基于 GPU 的 AES 快速实现[J]. 计算机工程与设计，2010，31(2)：256-259.
YE Jian, LI Lixin. Fast implementation of AES based on GPU[J]. Computer Engineering and Design, 2010, 31(2): 256-259.

[5] 李浪，李仁发，吴克寿. 简化固定值掩码二阶差分功耗攻击方法及其防御措施[J]. 小型微型计算机系统，2010，31(9)：1894-1898.
LI Lang, LI Renfa, WU Keshou. Second-order differential attacking method of simple fixed-value masking and its defense measures[J]. Journal of Chinese Computer Systems, 2010, 31(9): 1894-1898.

[6] Zhang X M, Parhi K K. High-speed VLSI architectures for the AES algorithm[J]. IEEE Transactions on Very Large Scale IntegrationSystems, 2007, 12(9): 957-967.

[7] Chung Y L, Chien C F, Hong J H, et al. An efficient area-delay product design for mixcolumns/Invmixcolumns in AES[C]// Proceedings of 2008 IEEE Computer Society Annual Symposium on VLSI. Aprial 7-9, 2007, Montpellier. Piscataway: IEEE Press, 2008: 503-506.

[8] Kundi D S, Aziz A, Ikram N. Resource efficient implementation of T-Boxes in AES on Virtex-5 FPGA[J]. Information Processing Letters, 2010, 110(10): 373-377.

[9] Li Z R, Zhuang Y Q, Zhang C, et al. Low-power and area-optimized VLSI implementation of AES coprocessor for Zigbee system[J]. The Journal of China Universities of Posts and Telecommunications, 2009, 16(3): 89-94.

[10] 程海，丁群，杜辉，等. 基于 FPGA 实现的 SMS4 算法研究[J]. 仪器仪表学报，2011，32(12)：2 845-2 850.
CHENG Hai, DING Qun, DU Hui, et al. Study of SMS4 based on of FPGA realization[J]. Chinese Journal of Scientific Instrument, 2011, 32(12): 2845-2850.

[11] 黄前山，季晓勇. 基于低成本 FPGA 的 AES 密码算法设计[J]. 通信技术，2010，43(9)：156-158.
HUANG Qianshan, JI Xiaoyong. Design of AES encryption algorithm based on low-cost FPGA[J]. Communications Technology, 2010, 43(9): 156-158.

（编辑 侯 湘）