DRM 系统的 SHA256 算法设计及 FPGA 实现

陈穗光，葛建华
（西安电子科技大学 综合业务网国家重点实验室，陕西 西安 710071）

摘 要：介绍了一种适于 DRM 系统的 SHA-256 算法和 HMAC 算法，给出了在 FPGA 上实现 SHA256 算法和 HMAC 算法的一种电路设计方案，并对算法的硬件实现部分进行了优化设计，给出了基于 Altera 公司的 Stratix II 系列的 FPGA 的实现结果。

关键词：DRM SHA-256 HMAC VerilogHDL

数字版权管理 DRM（Digital Rights Management）是一项涉及到技术、法律和商业各个层面的系统工程，它为数字媒体的商业运作提供了一套完整的实现手段。DRM 技术的出现，使得版权所有者不用再耗费大量时间和精力与客户进行谈判，就能确保数字媒体内容能够被合法地使用。DRM 将使各个平台的内容提供商们，无论是通过 Internet、流媒体还是通过交互数字电视，都能提供更多的内容，采取更灵活的节目销售方式，同时有效地保护知识产权。由于杂凑函数即 Hash 函数是实现有效、安全可靠数字签字和认证的重要工具，所以它是 DRM 系统中安全认证协议的重要模块。本文在分析 SHA-256 算法的基础上，给出了在 FPGA 上实现 SHA256 算法和 HAMC 算法的一种电路设计方案。

1 杂凑函数

杂凑（Hash）函数是将任意长的数字串 M 映射成一个较短的定长输出的数字串 H 的函数，以 $h(M)$ 表示，称 $H=h(M)$ 为 M 的杂凑值、杂凑码或消息摘要。杂凑码是消息中所有比特的函数，因此它提供了一种错误检测能力，即改变消息中任何一个比特或几个比特都会使杂凑码发生改变。单向杂凑函数还可按其是否有密钥控制划分为两大类：一类有密钥控制，以 $h(k,M)$ 表示，为密码杂凑函数；另一类无密钥控制，为一般杂凑函数。无密钥控制的单向杂凑函数，其杂凑码只是输入字串的函数，任何人都可以计算，因而不具有身份认证功能，只用于检测接收数据的完整性，如串改检测码 MDC。而有密钥控制的单向杂凑函数，能满足各种安全性要求，其杂凑码不仅与输入有关，而且与密钥有关，只有持此密钥的人才能计算出相应的杂凑码，因而具有身份验证功能，如消息认证码 MAC。

杂凑函数的目的是为需要认证的数据产生一个“指纹”。为了能够对数据实现认证，杂凑函数应满足以下条件：

(1) 函数的输入可以为任意长度，而函数的输出则是固定长的。

(2) 已知 M，求 $h(M)$ 较为容易，可用硬件或软件实现。

(3) 已知 H，欲求使得 $h(M)=H$ 的 M 在计算上是不可行的，这一性质称为函数的单向性，并称 $h(M)$ 为单向杂凑函数。

(4) 已知 M，找出 $N(N\neq M)$，使得 $h(N)=h(M)$ 在计算上是不可行的。如果单向杂凑函数满足这一性质，则称其为弱单向杂凑函数。

(5) 找出任意两个不同的输入 M 和 N，使得 $h(N)=h(M)$ 在计算上是不可行的。如果单项杂凑函数满足这一

性质，则称其为强单向杂凑函数。

2 SHA-256 算法

美国国家标准技术研究所(NIST)与美国国家安全局(NSA)共同设计出一种与美国数字签字算法(DSS)一起使用的安全哈希算法 SHA (secure hash algorithm)，并于 1992 年 1 月 31 日公布，于 1993 年 5 月 11 日采纳为标准。后经不断改进，现在已经发布了五种安全的哈希标准，分别为 SHA-1、SHA-224、SHA-256、SHA-384、SHA-512。本文将对 SHA-256 算法进行分析。

该算法包括以下几步处理：

(1) 对消息填充。对消息填充，使得其比特长在模 512 下为 448(即长度=488 mod 512)，即使消息长度已满足要求，仍需填充。一次填充位数在 1~512 之间。填充方式是固定的，即第一位为 1，其余各位皆为 0。

(2) 附加消息长度。在消息后附加 64 位，将其看作 64 位无符号整数(最高有效位在前)，它表示了消息填充前的长度。

(3) 解析填充后的信息。填充后的消息被解析成 N 个 512 比特的分组。

(4) 初始化缓存。在开始计算 SHA-256 以前，必须先设定初始值。它包含 8 个长度为 32 位的字，该缓存的十六进制表示为 A=0x6a09e667，B=0xbb67ae85，C=0x3c6ef372，D=0xa54ff53a，E=0x510e527f，F=0x9b05688c，G=0x1f83d9ab，H=0x5be0cd19。

(5) 迭代压缩。SHA-256 算法使用了六种逻辑函数，由 64 步迭代运算组成。每步都以 256 位缓存值 ABCDEFGH 为输入，然后更新缓存内容用于下次迭代计算。每步使用一个 32 位的 W_t 和一个 32 位的常数值 K_t。其中每一个都是针对长度为 32 位的字(x,y,z)的操作，其结果是得到一个新的长度为 32 位的字。六个逻辑函数描述如下：

$$\mathrm{Ch}(z,y,x)=(x \wedge y)\oplus(\bar{z} \wedge x)$$

$$\mathrm{Maj}(z,y,x)=(x \wedge y)\oplus(x \wedge z)\oplus(y \wedge z)$$

$$\Sigma_0(x)=\mathrm{ROTR}^2(x)\oplus\mathrm{ROTR}^{13}(x)\oplus\mathrm{ROTR}^{22}(x)$$

$$\Sigma_1(x)=\mathrm{ROTR}^6(x)\oplus\mathrm{ROTR}^{11}(x)\oplus\mathrm{ROTR}^{25}(x)$$

$$\sigma_0(x)=\mathrm{ROTR}^7(x)\oplus\mathrm{ROTR}^{18}(x)\oplus\mathrm{SHR}^3(x)$$

$$\sigma_1(x)=\mathrm{ROTR}^{17}(x)\oplus\mathrm{ROTR}^{19}(x)\oplus\mathrm{SHR}^{10}(x)$$

在所有的 512 比特长的分组处理完毕之后，SHA-256 算法的最后一个分组产生的输出为 256bit 的报文摘要。图 1 详细说明了 SHA-256 算法单步迭代处理的流程。

3 HMAC 算法

杂凑函数并不是为 MAC 而设计的，由于一般杂凑函数不使用密钥，因此不能直接用于 MAC。目前已经提出了很多用杂凑函数构造 MAC 的方法，其中 HMAC 算法就是其中之一。由于杂凑函数的实现模块可作为实现 HMAC 算法的一个模块，因此 HMAC 算法的代码中很大一块可事先准备好，无需修改即可使用。如果 HMAC 算

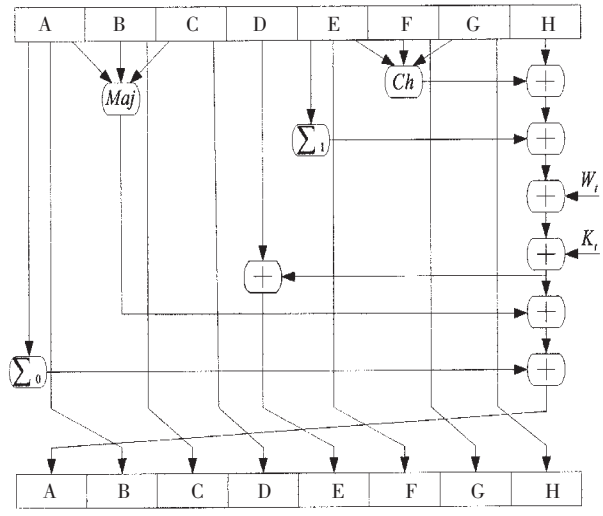

图 1 SHA-256 算法单步迭代处理示意图

法要求使用更快或更安全的杂凑函数模块，则仅需要用新的模块替代旧的模块即可。

图 2 为 HAMC 算法的具体运行框图，其中 $Hash$ 为嵌入的杂凑函数 (SHA256)，M 为 HMAC 算法的输入消息 (包括杂凑函数所要求的填充位)，Y_i $(0 \le i \le L-1)$是 M 的第 i 个分组，L 是 M 的分组数，b 是一个分组中的比特数，n 为嵌入的杂凑函数所产生的杂凑码的长度，K 为密钥(如果密钥长度大于 b，则将密钥输入到杂凑函数中，产生一个 n 比特长的密钥)，K^+是左边经填充 0 后的 K，K^+的长度为 b 比特，$ipad$ 为 8'b00110110，$opad$ 为 8'b01011010。IV 为杂凑函数的初值，可以预先计算。

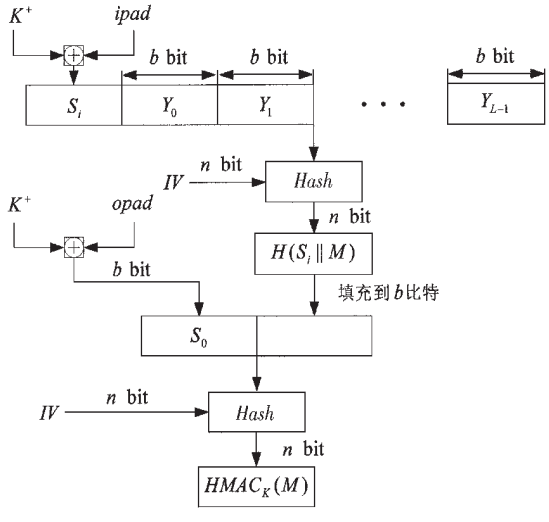

图 2 HMAC 的算法框图

4 FPGA 设计

由上述可知，SHA-256 算法实现的模块可作为 HMAC 算法的一个基本模块。本文的设计目标是在时序余量满足设计要求的情况下，通过对 SHA-256 算法的基本模块单元(process$^{(SHA-256)}$、update$^{(SHA-256)}$、finish$^{(SHA-256)}$)的复用，实现 SHA-256 算法模块及 HMAC 算法模块，提高

资源利用率。process$^{(SHA-256)}$模块是最低层的模块，它主要实现图1所示的迭代功能，update$^{(SHA-256)}$、finish$^{(SHA-256)}$模块通过调用process$^{(SHA-256)}$模块实现数据流的填充与更新。基本模块(process$^{(SHA-256)}$、update$^{(SHA-256)}$、finish$^{(SHA-256)}$)的具体电路设计由于篇幅所限不再赘述，本节将具体阐述模块复用的思想。图3是SHA-256算法及HMAC算法的FPGA设计方框图。

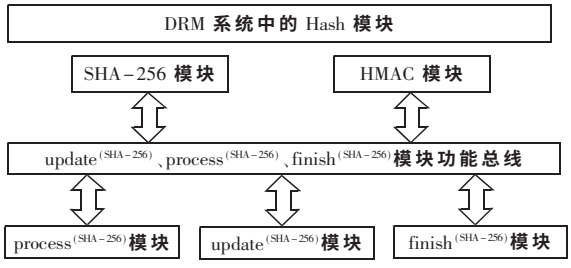

图3 SHA-256算法及HMAC算法的模块设计方框图

在实际的FPGA硬件电路设计中，设计者通常要考虑到“面积”和“速度”的平衡与互换原则问题。这里的“面积”是指一个设计所消耗的FPGA的逻辑资源数量，“速度”是指设计模块在芯片上稳定运行时所能达到的最高频率。科学的设计目标应该是在满足设计时序要求(包含对设计最高频率的要求)的前提下，占用最小的芯片面积，或者在所规定的面积下，使设计的时序余量更大，频率更高。由于Verilog HDL所描述的是具体的硬件电路，一个模块代表着拥有特定功能的一个电路块，每当一个模块在其他模块内被调用一次，被调用的模块所表示的电路结构就会在调用模块代表的电路内部复制一次。因此“模块调用”与“函数调用”在本质上有着很大的区别，如果不对基本模块进行合理的复用，将会造成芯片资源的极大浪费。模块复用即是顶层模块在时序逻辑的控制下通过总线对基本模块的数据流进行操作。如3图所示，DRM系统中所设计的Hash模块要实现两个功能，即SHA-256算法功能和基于SHA-256算法的HAMC算法功能。在时序控制下，这两个模块通过对process$^{(SHA-256)}$、update$^{(SHA-256)}$、finish$^{(SHA-256)}$模块功能总线的控制完成各自的功能运算。本文使用了Verilog HDL语言进行程序设计，且在QuartusII5.0环境下进行了编译综合，选用了Altera公司的EP2S90F1020C5芯片进行了整体的仿真。由表1可知，实现SHA-256算法和HMAC算法功能的Hash模块所占用的资源比单独实现SHA-256算法模块和HMAC算法模块所占用的资源总和有大幅度的降低(节省了将近130%的Total ALUT和将近120%的Total register，主频也提高了将近4MHz)，说明复用技术在硬件电路的设计中具有举足轻重的影响。

表1 复用前后硬件资源对比

名 称	SHA-256 模块	HMAC 模块	复用后的 Hash 模块
Total ALUT	16 977(23%)	18 637(26%)	15 455(21%)
Total register	11 287	12 418	10 789
Clock	61.60MHz	61.68MHz	65.34MHz

本文在分析SHA-256算法和HMAC算法基础上，使用Verilog HDL语言完成了Hash模块的程序设计，实现了SHA-256算法和HAMC算法功能，利用Altera公司生产的Stratix II系列芯片完成了电路的FPGA硬件实现，经电路版调试效果良好，且与其他算法模块(签名、验证等模块)配合实现了具体项目的数字内容保护模块。

参考文献

[1] 王育民，刘建伟. 通信网的安全——理论与技术[M]. 西安：西安电子科技大学出版社，2002.
[2] 杨波.现代密码学[M]. 北京：清华大学出版社.2003.
[3] 夏宇闻. Verilog 数字系统设计教程[M].北京：北京航空航天大学出版社，2005.
[4] National Institute of Standards and Technology. SHA-2 Standard: Secure Hash Standard, FIPS PUB 180-2[S]. www.itl.nist.gov/fipspubs/fip180-2.htm.
[5] National Institute of Standards and Technology: HMAC Standard: The Keyed-Hash Message Authentication Code: HMAC[S]. http://csrc.nist.gov/publications/fips/fips.htm.

(收稿日期：2006-08-24)