

一种能够实现多种散列函数的 VLSI-IP 模块设计

陈英杰,王海欣,白国强,陈弘毅
(清华大学 微电子研究所,北京 100084)

摘 要: 给出了一种能够实现多种散列函数的 VLSI-IP 模块设计,应用到一种网络安全处理器的认证模块设计中. 在实现 SHA-1 和 CHI 安全散列函数运算的基础上,进而利用迭代技术实现散列消息鉴别码 HMAC-SHA-1 和 HMAC-CHI-160,并生成 SSL(Security Socket Layer)协议中所需的主密钥和密钥块. 采用 SMIC 0.13 μm CMOS 工艺,综合后关键路径为 4.56ns,面积为 0.61mm²,运算 SHA-1 的吞吐率达到 1.82Gb/s.

关键词: 网络安全处理器;散列函数;散列消息鉴别码;主密钥;密钥块;迭代

中图分类号: TN 918

文献标识码: A

文章编号: 1000-7180(2010)04-0089-06

A VLSI-IP Module Design for Implementing Multi-hash Function

CHEN Ying-jie, WANG Hai-xin, BAI Guo-qiang, CHEN Hong-yi

(Institute of Microelectronics, Tsinghua University, Beijing 100084, China)

Abstract: This work presents a VLSI-IP module design for implementing multi-hash function, applied to an authentication module design for network security processor. On the basis of implementing hash function such as SHA-1 and CHI, by using iteration technology, this module supports not only keyed-hashing for message authentication such as HMAC-SHA-1 and HMAC-CHI-160, but also generating MASTER-KEY and KEY-BLOCK in SSL(Security Socket Layer) protocol. This module was designed with SMIC 0.13 μm CMOS technology occupying 0.61mm² with critical path of 4.56ns, the implementation result gives a throughput of 1.82Gb/s for SHA-1.

Key words: network security processor; hash function; keyed-hashing for message authentication; MASTER-KEY; KEY-BLOCK; iteration

1 引言

作为网络安全运行的基础,密码技术近年来发展更是十分迅速.随着密码算法复杂性增加和参数长度增加,软件方式已经不能满足需要,硬件实现成为一种必要选择.构造专用的密码芯片处理与加密相关的一系列问题,既可保证网络安全运行,又不降低处理速度,这里称这种协助网络处理安全问题的专用硬件为网络安全处理器.网络安全处理器适用于各种服务器、路由器的认证、数据加密等场所,其结构如图 1 所示^[1].图 1 中曲线所圈的即为文中要实现的模块,为方便陈述称之为 HMAC 模块.

2 HMAC 模块实现的功能

文中主要是在实现 SHA-1^[2]和 CHI 的基础上,实现 SHA-1 和 CHI 同时处理同一组输入数据,利用迭代实现散列消息鉴别码^[3-5]、SSL 协议^[6-8]中所需的主密钥和密钥块,HMAC 模块实现的具体功能如下文所述.

2.1 实现 SHA-1 和 CHI 同时处理同一组输入数据

SHA-1 模块和 CHI 模块同时处理同一组输入数据时,两模块速度不匹配.处理 512-bit 数据时,SHA-1 需要 80 轮迭代运算,CHI 需要 64 轮迭代运算,CHI 比 SHA-1 所需时钟周期数少,为了使两个模块接收的数据都正确,文中设计使用了一个内部

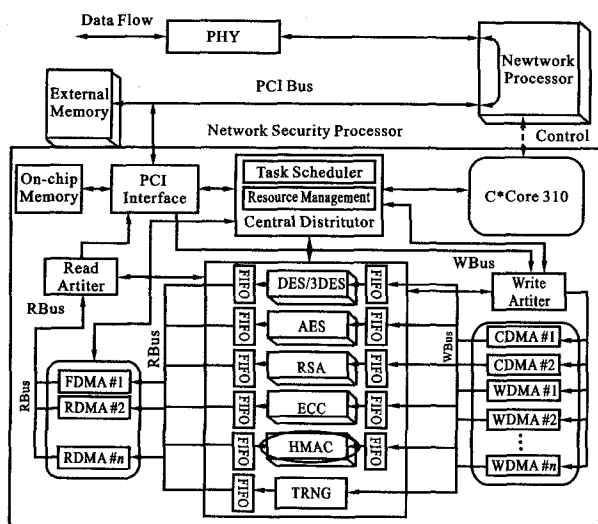


图1 网络安全协处理器结构图

存储器.接收第一组 512-bit 的数据时,两模块均从输入端接收;输入数据的长度超过 512 比特时, HMAC 模块的输入端在将第 512-bit 后的输入数据传输给 CHI 模块的同时,要将输入数据存入内部存储器中,当 SHA-1 模块处理完第一组 512-bit 数据时就从存储器中读取来自输入模块的数据,而 CHI 模块一直从输入模块接收数据. SHA-1 模块和 CHI 模块速度的不匹配导致存储器可能同时进行写操作和读操作,因此文中采用 128×64 位双口 SRAM 的存储器.

2.2 实现散列消息鉴别码 HMAC-SHA-1 和 HMAC-CHI-160

2.2.1 算法原理

图2给出了散列消息鉴别码算法结构,算法可描述为: $HMAC(K, M) = H[(K_1 \oplus opad) \parallel H[(K_1 \oplus ipad) \parallel M]]$

符号定义如下:

H 为嵌入的散列函数(如 SHA-1, CHI);

IV 为散列函数输入的初始值;

M 为消息输入(包括由嵌入散列函数定义的填充位);

Y_i 为 M 的第 i 个分组, $0 \leq i \leq (L-1)$;

L 为 M 中的分组数;

b 为每一分组所含的位数,文中要求 512 位;

n 为嵌入的散列函数所产生的消息摘要长度,文中为 160 位;

K 为密钥;

K_1 为密钥 K 经过处理后的结果, 512-bit. 若 K 长度小于 512 位, 在 K 左边填充 0; 若 K 长度等于

512 位, K_1 等于 K ; 若 K 长度大于 512 位, 将 K 作为嵌入散列函数的输入数据, 经散列函数运算生成 160 位的消息摘要, 在生成的消息摘要左边填充 352 位 0, 即得 K_1 .

$ipad$ -00110110 重复 $b/8$ 次的结果, 文中重复 64 次;

$opad$ 为 01011100 重复 $b/8$ 次的结果, 本文中重复 64 次

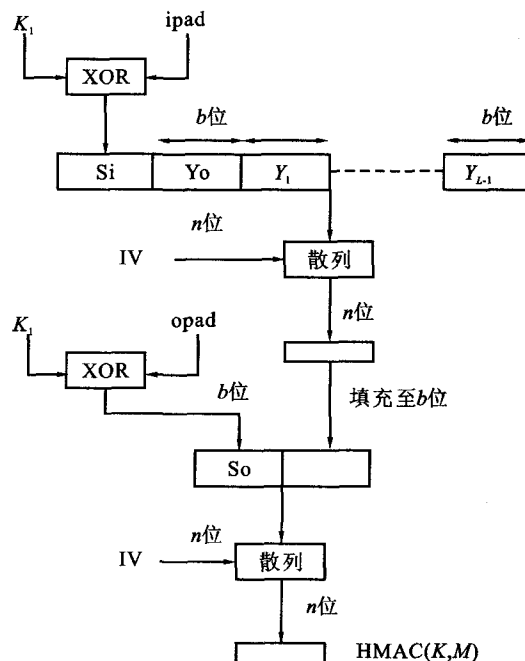


图2 散列消息鉴别码算法结构

2.2.2 实现步骤

步骤 1: 由输入模块给的密钥 K 生成 K_1 ;

步骤 2: K_1 与 $ipad$ 执行异或运算产生 b 位的分组 Si , 文中为 512 位的分组;

步骤 3: 将 M 附于 Si 后;

步骤 4: 将散列函数(SHA-1 或 CHI)作用于步骤 3 所得出的结果;

步骤 5: K_1 与 $opad$ 执行异或运算产生 b 位的分组 So , 文中为 512 位的分组;

步骤 6: 将步骤 4 中的散列码附于 So 后;

步骤 7: 将散列函数(SHA-1 或 CHI)作用于步骤 6 所得出的结果, 即为所求的散列消息鉴别码

2.2.3 实现 HMAC-SHA-1 和 HMAC-CHI-160

由前文所述的散列消息鉴别码算法说明可知, 要生成鉴别码需要执行两次或三次散列函数运算, 若密钥长度小于等于 512 位, 只要执行两次散列函数; 若密钥长度大于 512 位, 执行三次散列函数. 因此, 生成消息鉴别码的关键是如何进行散列函数的

迭代.由密钥 K 生成 K_1 时,无论是否需要执行散列函数,生成 K_1 的同时产生控制信号 CON1,并将 K_1 存入 512 位的寄存器中,供生成 S_i 和 S_o 时调用 K_1 ;将控制信号 CON1 作为散列函数的使能信号,执行第 4 步,散列函数执行完毕后产生控制信号 CON2,并将结果存入 160 位的寄存器 R1 中;利用 R1 执行第 6 步;将控制信号 CON2 作为散列函数的使能信号,执行第 7 步,散列函数执行完毕后产生控制信号 end_HMAC_SHA-1 或 end_HMAC-CHI,表示散列消息鉴别码算法结束,并将最终结果存入 160 位的寄存器中,供整个模块输出时调用.

2.3 生成主密钥 MASTER-KEY

2.3.1 主密钥说明

MASTER-KEY =

CHI(pre_master_key || Sha1('A' || pre_master_key || client.random || server.random)) ||

CHI(pre_master_key || Sha1('BB' || pre_master_key || client.random || server.random)) ||

CHI(pre_master_key || Sha1('CCC' || pre_master_key || client.random || server.random))

pre_master_key 为 384-bit,文中'A'、'BB'和'CCC'均为 64-bit,client.random 和 server.random 共 448-bit,SHA-1 运算结果为 160-bit,因此,SHA-1 模块的输入数据长度为 896-bit,CHI 模块的输入数据长度为(384 + 160)比特,即 544-bit.本文中 SHA-1 模块处理 512-bit 数据需要 88 个周期,其中用 8 个周期读入 512-bit 数据,用 80 个周期处理 512-bit 的数据,最终生成 160-bit 的消息摘要;CHI 模块处理 512-bit 数据需要 72 个周期,其中用 8 个周期读入 512-bit 数据,用 64 个周期处理 512-bit 的数据,最终生成 160-bit 的消息摘要.因此,SHA-1 模块输入了 896-bit 的数据时,需要执行 2×88 个周期,即 176 个周期才能结束 SHA-1 运算;CHI 模块输入了 544-bit 的数据时,需要执行 2×72 个周期,即 144 个周期才能结束 CHI 运算.利用迭代分别完成三次 SHA-1 和三次 CHI 运算,将三次执行 CHI 运算的结果拼接,即为主密钥,共 480-bit.

2.3.2 主密钥实现

执行 MASTER-KEY 时,由控制器分别产生'A'、'BB'和'CCC',SHA-1 模块接收来自输入模块的数据,同时将输入的 832-bit 数据存入内部存储器中,供第二次、第三次 SHA-1 运算和三次 CHI 运算调用.执行完第一次 SHA-1 运算时,生成控制信号 M1,并将运算结果存入 160-bit 寄存器中,供第一次

CHI 运算调用.

将 M_1 作为第一次 CHI 运算和第二次 SHA-1 运算的使能信号,SHA-1 模块先从控制器中读取 64-bit 的'BB',然后和 CHI 模块一起从内部存储器中读取输入数据,当读完 pre_master_key 后,CHI 再从 160-bit 的寄存器中读取第一次 SHA-1 运算结果作为 CHI 模块的输入数据,第一次 CHI 运算完毕时,将结果存入 160-bit 的寄存器中;第二次 SHA-1 运算完毕时,生成控制信号 M_2 ,并将运算结果存入 160-bit 寄存器中,供第二次 CHI 运算调用.

实现第二次 CHI 运算和第三次 SHA-1 运算时,控制方法与上述的第一次 CHI 运算和第二次 SHA-1 运算相同.第三次 SHA-1 运算完毕时生成控制信号 M_3 .

将 M_3 作为第三次 CHI 运算的使能信号,CHI 模块先从内部存储器中读取输入数据,读完 pre_master_key 后,再从 160-bit 的寄存器中读取第三次 SHA-1 运算的结果作为 CHI 模块的输入数据,第三次 CHI 运算完毕时,将结果存入 160-bit 的寄存器中,并生成控制信号 end_M,表示生成主密钥任务结束.

文中生成主密钥所需时间为执行三次 SHA-1 和一次 CHI 的时间总和,共 $3 \times 176 + 144 = 672$ 个周期.

2.4 生成密钥块 KEY-BLOCK

2.4.1 密钥块说明

KEY-BLOCK =

CHI(master_key || Sha1('A' || master_key || server.random || client.random)) ||

CHI(master_key || Sha1('BB' || master_key || server.random || client.random)) ||

CHI(master_key || Sha1('CCC' || master_key || server.random || client.random)) ||

CHI(master_key || Sha1('DDDD' || master_key || server.random || client.random))

master_key 为 480-bit,文中'A'、'BB'、'CCC'和'DDDD'均为 64-bit,client.random 和 server.random 共 448-bit,SHA-1 运算结果为 160-bit,因此,SHA-1 模块的输入数据长度为 992-bit,CHI 模块的输入数据长度为(480 + 160)比特,即 640-bit.因此,SHA-1 模块输入了 992-bit 的数据时,需要执行 3×88 个周期,即 264 个周期才能结束 SHA-1 运算;CHI 模块输入了 640-bit 的数据时,需要执行 2×72 个周期,即 144 个周期才能结束 CHI 运算.利用迭

代分别完成四次 SHA-1 和四次 CHI 运算,将四次执行 CHI 运算的结果拼接,即为密钥块,共 640-bit.

2.4.2 密钥块实现

执行 KEY-BLOCK 时,由控制器分别产生‘A’、‘BB’、‘CCC’和‘DDDD’,SHA-1 模块接收来自输入模块的数据,同时将输入的 928-bit 数据存入内部存储器中,供第二次、第三次、第四次 SHA-1 运算和四次 CHI 运算调用.执行完第一次 SHA-1 运算时,生成控制信号 N1,并将运算结果存入 160-bit 寄存器中,供第一次 CHI 运算调用.

将 N1 作为第一次 CHI 运算和第二次 SHA-1 运算的使能信号,SHA-1 模块先从控制器中读取 64-bit 的‘BB’,然后和 CHI 模块一起从内部存储器中读取输入数据,当读完 master_key 后,CHI 再从 160-bit 的寄存器中读取第一次 SHA-1 运算结果作为 CHI 模块的输入数据,第一次 CHI 运算完毕时,将结果存入 160-bit 的寄存器中;第二次 SHA-1 运算完毕时,生成控制信号 N2,并将运算结果存入 160-bit 寄存器中,供第二次 CHI 运算调用.

实现第二次 CHI 运算和第三次 SHA-1 运算、第三次 CHI 运算和第四次 SHA-1 运算时,控制方法与上述的第一次 CHI 运算和第二次 SHA-1 运算相同.第四次 SHA-1 运算完毕时生成控制信号 N4.

将 N4 作为第四次 CHI 运算的使能信号,CHI 模块先从内部存储器中读取输入数据,读完 master_key 后,再从 160-bit 的寄存器中读取第四次 SHA-1 运算的结果作为 CHI 模块的输入数据,第四次 CHI 运算完毕时,将结果存入 160-bit 的寄存器中,并生成控制信号 end_K,表示生成密钥块任务结束.

文中生成密钥块所需时间为执行四次 SHA-1 和一次 CHI 的时间总和,共 $4 \times 264 + 144 = 1200$ 个周期.

3 模块结构

3.1 HMAC 内部模块

整个 HMAC 模块可以划分为输入输出、任务分配器、SHA-1、CHI、控制器、K₁ 生成器、填充字生成器和存储器八个模块,如图 3 所示.

输入输出模块接收来自外部的原始信息,并接收来自内部控制器的反馈信息;任务分配器接收来自外部输入的每一个 64-bit 数据,根据低 8 位配置信息解析出所要执行的任务,产生相应的控制信号并送给控制器,配置信息如表 1 所示;控制器根据任

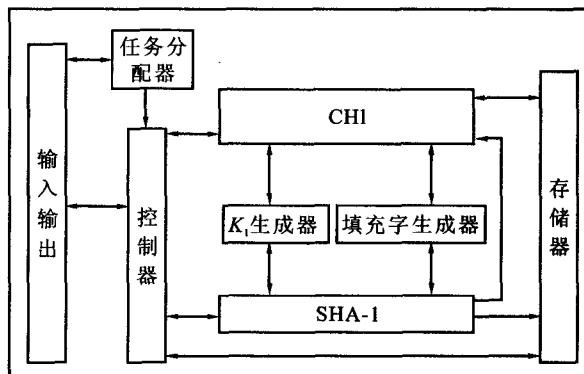


图 3 HMAC 结构

务分配器产生的使能信号来控制 CHI 模块、SHA-1 模块和存储器;K₁ 生成器为执行 HMAC-SHA-1 或 HMAC-CHI-160 任务提供所需的密钥 K₁,K₁ 生成的具体过程详见 2.2.1;填充字生成器为执行 SHA-1 或 CHI 运算提供所需的填充字.

表 1 配置信息

Bit	7	6	5	4	3	2	1	0	
类							C	S	消
		None		KEY-	MASTER-		H	H	息
				BLOCK	KEY		I	A-	鉴
别								1	别
									码
Bit	Name		Description						
4	KEY-BLOCK		高电平表示生成 KEY-BLOCK						
3	MASTER-KEY		高电平表示生成 MASTER-KEY						
2	010	SHA-1	单独运算 SHA-1						
	100	CHI	单独运算 CHI						
1	110	SHA-1_and_CHI	SHA-1 和 CHI 同时运算						
0	011	HMAC-SHA-1	HMAC-SHA-1 运算						
	101	HMAC-CHI-160	HMAC-CHI-160 运算						

HMAC 模块实现的关键在于如何实现 SHA-1 或 CHI 模块的迭代、利用内部存储器正确地 SHA-1 和 CHI 模块提供输入数据(详见文中的第 2 部分).

3.2 HMAC 模块接口

文中实现的 HMAC 模块接口如图 4 所示,接口信号说明如表 2 所示.

HMAC 模块输入输出端分别接 64×64 位的 FIFO.网络安全处理器首先将 64 位配置信息、散列函数 SHA-1 和 CHI、散列消息鉴别码 HMAC-SHA-1 和 HMAC-CHI-160、主密钥 MASTER-KEY 和密钥块 KEY-BLOCK 运算所需数据输入到输入 FIFO,即图 3 中的 WFIFO,供 HMAC 模块调用.根据 D_type 信号的状态,判断 WFIFO 中的数据是配置信息还是

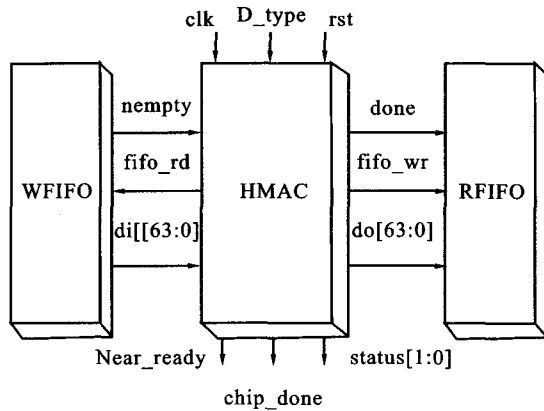


图4 HMAC 模块接口

表2 HMAC 模块接口信号

名称	宽度	方向	说明
rst	1	输入	异步复位,低电平有效
clk	1	输入	参考时钟,160MHz
D_type	1	输入	高电平时 di 为配置信息, 低电平时 di 为数据信息.
nempty	1	输入	高电平时 WFIFO 非空, 低电平时 WFIFO 空.
di	64	输入	输入任务和数据
fifo_rd	1	输出	高电平时从 WFIFO 读出数据
fifo_wr	1	输出	高电平时向 RFIFO 写入数据
near_ready	1	输出	任务将要完毕,高电平有效
chip_done	1	输出	完成任务
status	2	输出	模块状态
do	64	输出	运算结果

数据信息,由配置信息和 HMAC 模块反馈的 fifo_rd 信号确定模块执行的任务,并读取数据信息进行相应运算;根据 nempty 信号的状态判断 HMAC 模块状态 status 处于读数据、执行任务或任务结束状态;运算结束后输出高电平的 fifo_wr 信号给输出 FIFO,即 RFIFO,同时输出运算结果给 RFIFO;运算结果输出完毕前一个周期,输出一个周期的高电平脉冲信号 chip_done;chip_done 为高电平时,fifo_wr 信号置为低电平,HMAC 模块执行任务完毕.

4 性能评估

为了进行性能评估,使用 Verilog HDL 语言实现 HMAC 模块,采用 SMIC 0.13 μ m CMOS 工艺库,在 Sun workstation 工作平台上调用 Synopsys Design Compiler 进行综合.综合后 HMAC 整个模块关键路径为 4.56ns,面积为 0.61mm²;SHA-1 模块的关键路径为 3.2ns,CHI 模块的关键路径为 4.52ns.网络安全处理器系统提供的时钟频率为 160MHz,文中

实现的 HMAC 模块关键路径小于 6.25ns,满足速度要求.

在 160MHz 的时钟下,各任务性能如表 3 所示.

表3 各任务的性能

类 别	处理数据 长度/位		周 期 数	时间 /ns (160 MHz 时钟)	次 数	吞 吐 率 /(Gb/s)
	密 钥 K /b	数 据 信 息 /b				
SHA-1		512	88	550	1818	1.82
CHI		512	72	450	2222	1.57
SHA-1_and_CHI		512	88	550	1818	
HMAC_SHA-1	512	512	360	2250	444	0.89
HMAC_CHI_160	512	512	296	1850	541	0.77
MASTER-KEY		896	672	4200	238	0.38
KEY-BLOCK		992	1200	7500	133	0.25

为了对性能进一步评估,将 SHA-1 与其他已知的 SHA-1 引擎进行比较,见表 4.文献[8]中在实现 SHA-1 时利用调度并通过使用进位保留加法器来缩短关键路径,提高工作频率,但面积最大;由文献[8]知 SuperMAP SHA-1 引擎的性能,通过使用外部的 SRAM 对消息进行预处理,使得其面积最小,但工作频率和吞吐率最小.由表 4 可以看出,文中设计的 SHA-1 模块工作频率和吞吐率均为最高,面积居中,整体性能较优.

表4 SHA-1 性能比较

SHA-1	面积 (等效门数)	工作频率/ MHz	吞吐率/ /(Gb/s)	工艺/ μ m
文中	8854	312.5	1.82	0.13
文献[8]	20536	143	0.893	0.25
SuperMAP SHA-1	7000	100	0.530	0.18

5 结束语

文中设计了应用于一种网络安全处理器的认证模块,在实现散列函数 SHA-1 和 CHI 模块的基础上,通过迭代技术,充分利用存储器,为 SHA-1 模块和 CHI 模块正确提供输入数据,分别实现了散列消息鉴别码 HMAC-SHA-1 和 HMAC-CHI-160、主密钥 MASTER-KEY、密钥块 KEY-BLOCK.在满足速度的前提下,用最小的面积实现了集散列函数、散列消息鉴别码、主密钥、密钥块为一体的芯片设计,在网络认证方面解决了硬件共享的优化设计问题,特别适用于网络安全处理器,用于信息的完整性校验、

数字签名等,实现网络认证,确保网络安全运行。

参考文献:

- [1] Haixin Wang, Yao Yue, Chunming Zhang, et al. A novel unified control architecture for a high-performance network security accelerator[C]// Proceedings of the International Conference on Security and Management(SAM'07). Hongkong, 2007(6):538-544.
- [2] FIPS PUB 180-1. Secure hash standard(SHA-1)[S]. National Institute of Standards and Technology(NIST), 1995.
- [3] American Bankers Association, Keyed Hash Message Authentication Code[S]. ANSI X9.71, Washington, D C, 2000.
- [4] William Stallings. 密码编码学与网络安全:原理与实践[M]. 4版. 北京:电子工业出版社,2006:266-270.
- [5] Frier A, Karlton P, Kocher P. The SSL protocol version

3.0[EB/OL]. (1996-11-10)[2009-04-10]. <http://wp.netscape.com/eng/ssl3/draft302.txt>. Netscape.

- [6] Rescorla, SSL E. TLS:designing and building secure systems[M]. Reading, MA:Addison-Wesley,2001.
- [7] William Stallings. 密码编码学与网络安全:原理与实践[M]. 3版. 北京:电子工业出版社,2004:391-400.
- [8] Yu Mingyan, Zhou Tong. An efficient ASIC implementation of SHA-1 engine for TPM[C]// The 2004 IEEE Asia-Pacific Conference on Circuits and Systems. Taiwan, 2004.

作者简介:

陈英杰 女,(1978-),硕士研究生.研究方向为散列函数在网络安全处理器中的应用及实现.

王海欣 男,博士研究生.研究方向为网络安全处理器设计.

白国强 男,副教授.研究方向为信息安全芯片.

陈弘毅 男,教授.研究方向为集成电路(IC)设计.

(上接第 88 页)

5 结束语

随着固态硬盘原材料价格的不断下降和系统结构以及相关算法的不断更新,逐步取代传统硬盘将是一种趋势.对于闪存器件本身的缺点,也可以通过工艺技术的不断发展以及引入其他技术手段加以改善,文中提出的流水线技术和并行结构设计就能很好解决这个问题.

参考文献:

- [1] 杨鑫,刑小地,徐伟俊,等. 用 NiosII 搭建的固态硬盘设备系统[J]. 单片机与嵌入式系统应用,2008(4):55-58.
- [2] 吴东坡. Flash 存储器技术与应用[J]. 微电子学与计算机,1998,15(6):55-56.

[3] Ray Weiss. Flash memory takes over[J]. Electronic Design, 2001,49(17):56-64.

[4] 周钦,钱松,程君侠. 适用于 Flash Memory 的负高压泵的实现[J]. 微电子学与计算机,2007,24(1):21-24.

[5] Samsung Electronics. NAND flash memory data sheets[EB/OL]. (2008-05-18)[2009-01-10]. <http://www.samsung.com/>.

[6] William Stallings. Computer organization and architecture: designing for performance[M]. 张昆藏,译. 7版. 北京:清华大学出版社,2006:352-356.

[7] 高怡祯. 基于闪存的星载大容量存储器的研究和实现[J]. 电子技术应用,2003,29(8):75-78.

作者简介:

李佳琦 女,(1986-),硕士研究生.研究方向为精密仪器设计.