

高频率、低成本的 SHA-256 算法 VLSI 实现

应建华, 罗柳平

(华中科技大学 电子科学与技术系, 湖北 武汉 430074)

摘 要: SHA-256 安全散列算法广泛应用于数据完整性校验及数字签名等领域. 为满足安全 SoC 系统对 SHA-256 高工作频率和低硬件成本的设计需求, 提出了一种新颖的 SHA-256 VLSI 实现方法, 通过分解算法实现步骤, 进而缩短关键路径, 节省硬件资源. 采用 SMIC 0.13 μm CMOS 工艺综合实现, 结果表明其最高工作频率达 334.5 MHz, 资源消耗减少了 70%.

关键词: SHA-256 算法; VLSI 实现; 关键路径; 面积优化

中图分类号: TP31

文献标识码: A

文章编号: 1000-7180(2011)05-0028-04

Implementation of the SHA-256 Hash Function of a High-Frequency and Low-Cost VLSI

YING Jian-hua, LUO Liu-ping

(Dept. of Electronic Science and Technology, Huazhong University of Science & Technology, Wuhan 430074, China)

Abstract: The secure hash algorithm of SHA-256 is widely met in the fields of data integrity and digital signature authentication. A novel VLSI architecture of SHA-256 is presented to achieve the goal of high frequency and low cost of secure SoC. The optimized design shortens the critical path and reduces hardware resource by decomposing steps of the algorithm. Synthesized with SMIC 0.13CMOS technology, the result indicates that the proposed implementation delivers 334.5MHz of maximal requency and 70% of area cost reduction.

Key words: SHA-256 algorithm; VLSI implementation; critical path; area efficient

1 引言

Hash 函数在现代密码学中有着重要的作用, 它广泛应用于数据完整性校验和数字签名等领域. 在 2008 年美国国家标准技术研究所(NIST)最新公开发布的标准文档 FIPS180-3^[1]中, 定义了 5 种安全散列算法, 分别是: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512. 每种算法都采用迭代、单向函数将一则消息浓缩成一段消息摘要. 这些算法保证了原始消息的完整性: 对一则消息的任意改动, 将以极高的概率产生不同的消息摘要.

近年来, 随着 MD5, SHA-1 等通用密码算法被破解, 人们日趋认为以 SHA-256 为代表的 SHA-2 是最安全的 Hash 函数, 对 SHA-256 的硬件实现成为研究的热点. 文献[2]通过分析 SHA-256 函数的

迭代关系, 提出了一种高效的 SHA-256 硬件实现架构, 数据吞吐量达到 1 920 Mb/s, 但其工作频率仅 75 MHz, 无法满足高频率 SoC 系统设计需求. 文献[3]采用了被称为 7-3-2 压缩机(7-3-2 compressor)的加法器结构, 缩短了关键数据路径, 时钟频率和数据吞吐量都较高, 但硬件资源消耗较多. 由于 SHA-256 算法本身的复杂性, 且每次迭代都需要利用上次的计算结果, 因此在系统工作频率越来越高的安全 SoC 系统中, 其工作频率的提高以及面积的优化都是实现的难点.

文中通过分析 SHA-256 算法本身的特点, 利用算法内部的迭代关系, 对算法实现步骤进行分解, 用 Register File 代替传统的寄存器阵列存储主要数据, 缩短了关键数据路径, 提高了工作频率, 实现了硬件资源的节省.

2 SHA-256 算法流程

SHA-256 能将任意小于 2^{64} 长度的输入数据压缩成 256bit 的消息摘要. 其具体算法包括以下步骤.

(1) 消息填充. 在 SHA-256 计算之前, 需要根据文献[1]所述的规则对原始消息进行填充处理.

(2) 填充后的消息分解. 由于 SHA-256 算法中的每次迭代只能处理 512 bit 的数据块, 因此需要将填充后的消息以 512 bit 为单位进行分块.

(3) 初始化 $H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7$, 它们为 8 个 32 bit 寄存器, 用于存储中间计算结果及最终消息摘要.

(4) 对每块消息执行以下处理.

① 将 512 bit 消息分成 16 个 32 bit 存入寄存器 $W_0, W_1, \dots, W_{15}$ 中;

② 计算 W_t , For $16 \leq t \leq 63, W_t = \sigma_1^{(256)}(W_{t-2}) + W_{t-7} + \sigma_0^{(256)}(W_{t-15}) + W_{t-16}$;

③ 初始化 8 个工作变量, $a = H_0^{(i-1)}, b = H_1^{(i-1)}, c = H_2^{(i-1)}, d = H_3^{(i-1)}, e = H_4^{(i-1)}, f = H_5^{(i-1)}, g = H_6^{(i-1)}, h = H_7^{(i-1)}$;

④ 8 个工作变量更新 64 次, For $0 \leq t \leq 63$, $TEMP_1 = h + \sum_1^{(256)}(e) + Ch(e, f, g) + K_t^{(256)} + W_t$, $TEMP_2 = \sum_0^{(256)}(a) + Maj(a, b, c)$, $h = g, g = f, f = e, e = d + TEMP_1, d = c, c = b, b = a, a = TEMP_1 + TEMP_2$;

⑤ 计算 SHA-256 的中间值 $H^i: H_0^{(i)} = a + H_0^{(i-1)}, H_1^{(i)} = b + H_1^{(i-1)}, H_2^{(i)} = c + H_2^{(i-1)}, H_3^{(i)} = d + H_3^{(i-1)}, H_4^{(i)} = e + H_4^{(i-1)}, H_5^{(i)} = f + H_5^{(i-1)}, H_6^{(i)} = g + H_6^{(i-1)}, H_7^{(i)} = h + H_7^{(i-1)}$.

上述中变量 i 代表正在进行第个 512 bit 数据块的计算, 当所有消息块计算完成后, 由 $H_0^N \parallel H_1^N \parallel H_2^N \parallel H_3^N \parallel H_4^N \parallel H_5^N \parallel H_6^N \parallel H_7^N$ 所组成的 256 bit 数据即为消息摘要.

3 SHA-256 VLSI 实现

图 1 为典型的 SHA-256 VLSI 实现电路, 数据填充模块将待计算的输入数据进行预处理, 使参与计算的数据符合协议要求; $W_0 \sim W_{15}$ 计算模块完成 $W_0 \sim W_{15}$ 的计算并将结果保存到 $W_0 \sim W_{15}$ 寄存器阵列中, 用于临时变量 $TEMP_1$ 的计算; 64 轮 $a, b \dots g, h$ 计算模块用于完成 4 循环 64 轮的 $a, b \dots, g, h$ 计算, 并将每次计算的中间结果保存到 $a, b \dots g, h$

寄存器阵列中; 在完成 64 轮操作后, 32 bit 加法器将当前的 $a, b \dots g, h$ 寄存器阵列值与 $H_0 \sim H_7$ 寄存器阵列(保存了消息摘要初始值或上个循环结束后的值)进行加法操作, 并将结果保存到 $H_0 \sim H_7$ 寄存器阵列中; 在全部的 512 bit 数据块处理完毕后, $H_0 \sim H_7$ 寄存器阵列中的值即为最终的消息摘要.

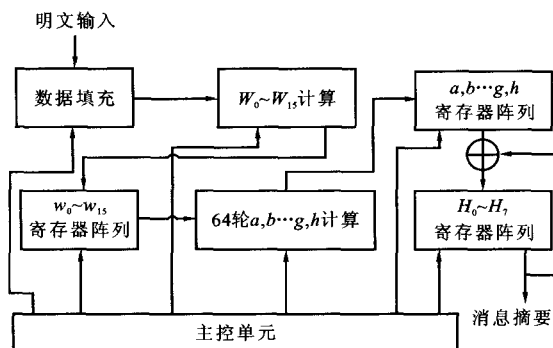


图 1 典型 SHA-256 VLSI 电路

由 SHA-256 的算法流程可知, 对变量的计算需要进行 6 次 32bit 的加法操作和若干组合逻辑, 是所有工作变量中最复杂的计算, 因此其计算需要的时间决定了 SHA-256 硬件实现的最高频率. 提高 SHA-256 工作频率的关键是对的计算过程进行适当处理, 缩短每次迭代过程的关键路径.

传统的 SHA-256 硬件实现电路中, 需要用到多组寄存器阵列, 以保存中间计算结果及消息摘要, 消耗了大量的硬件资源, 越来越不适于大规模 SoC 系统的低硬件成本发展需求. 为满足高工作频率以及面积敏感的安全 SoC 系统设计要求, 本设计对基本的 SHA-256 计算过程进行分解, 以实现寄存器、加法器的共享, 缩短关键路径; 为节省面积, 传统的寄存器阵列用 Register File 代替, 算法实现的硬件架构相应地做出变更. 图 2 为优化后的 SHA-256 VLSI 实现电路, 主要包括 SHA-256_St 状态机模块、数据填充模块、Register File 及相关的控制逻辑. 其中 SHA-256_St 状态机模块的设计是本优化设计的关键. 图 3 为实现对 N 个 512 bit 明文计算的 SHA-256_St 状态机流程图, 下文将对其进行重点分析.

3.1 SHA-256_St 状态机结构描述

(1) SHA-256_Idle 状态: 当启动 SHA-256 计算时进入 SHA-256_Wt 状态, 否则保持 SHA-256_Idle 状态不变.

(2) SHA-256_Wt 状态: 此状态用于计算 W_t , 保存结果用于 $TEMP_1$ 的计算, 之后进入 SHA-256_Ch 状态.

(3) SHA-256_Ch 状态: 此状态用于计算 Ch

c, 第 6 个周期取出 d 存入 r_2 ; 第 7 个周期将 r_1 存入 d ; 第 8 个周期取出 e 存入 r_1 ; 第 9 个周期将 r_2 与 r_3 进行加法操作存入 e ; 第 10 个周期取出 f 存入 r_2 ; 第 11 个周期将 r_1 存入 f ; 第 11 个周期取出 g 存入 r_1 ; 第 12 个周期将 r_2 存入 g ; 第 13 个周期将 r_1 存入 h .

(6) SHA-256_End 的实现: 由 $H_0 \sim H_7$ 的计算表达式, 第 1 个周期取出 a 存入 r_1 ; 第 2 个周期取出 $H_0^{(i+1)}$ 存入 r_2 ; 第 3 个周期将 r_1 与 r_2 进行加法操作, 结果保存在 r_2 中, 并存入 H_0 ; 第四个周期将 r_2 存入 a . 其余变量计算过程类似.

4 实现结果

文中实现的 SHA-256 算法模块主要消耗的资源包括: 4 个 32bit 寄存器、1 个 32bit 加法器、3232 的 Register File 以及相关控制逻辑. 使用 Verilog 硬件描述语言实现了优化后的 SHA-256 算法电路, 利用 Synopsys Design Compiler 在 SMIC 0.13 μm 标准 CMOS 单元库下综合, 具体结果如表 1 所示. 根据表 1 比较结果, 可得出: 相对于传统的设计方法, 优化后的 SHA-256 VLSI 实现通过分解算法计算过程, 提高了工作频率, 实现了大部分寄存器共享, 尽管数据吞吐率有一定程度下降, 但仍能满足一般 SoC 系统对 SHA-256 数据处理能力的要求. 在 ASIC 综合下, 其工作频率达 334.5 MHz, 硬件资源最大程度节省, 等效为 6 068 门.

表 1 ASIC 实现结果比较

设计方法	工作频率/ MHz	面积/ gates	数据吞吐率/ (Gb/s)
文献[3]	208	24 141	0.819
文献[4]	120	20 377	0.472
文献[5]	59	10 900	0.056
文中	334.5	6 068	0.141

表 2 为 FPGA 实现的结果, 比较分析可知通过分解 SHA-256 算法实现过程, 进行寄存器共享的设计方法, 在基于 FPGA 的硬件实现中同样具有工作频率和面积上的综合优势.

5 结束语

文中提出了一种提高工作频率, 有效降低资源消耗的 SHA-256 VLSI 实现方法. 传统的 SHA-256 实现往往强调数据吞吐率的提升, 而忽视了 SoC 系统工作频率的日益提高以及低硬件成本的设计要求

越来越强烈的发展趋势, 文中设计的 SHA-256 特别适用于高频率、面积敏感的 SoC 系统, 已应用于基于 ARM 核的安全芯片中.

表 2 FPGA 实现结果比较

设计方法	工作频率/ MHz	面积/ CLBs	数据吞吐率/ Gb/s
文献[6]	83	1 060	0.326
文献[7]	42.9	1 004	0.077
文献[8]	74	2 207	0.256
文献[4]	50	2 227	0.196
文中	191.2	411	0.081

参考文献:

[1] NIST, Secure hash standard, FIPS PUB 180-3[S]. USA: NIST, 2008.

[2] Kitsos P, Koufopavlou O. Efficient architecture and hardware implementation of the whirlpool hash function[J]. IEEE Transactions on Consumer Electronics, 2004; 208-213.

[3] Ling Bai, Shuguo Li. VLSI implementation of high-speed SHA-256[C]// IEEE 8th International Conference on ASIC. Changsha, Hunan: IEEE, 2009; 131-134.

[4] Marsgod. Secure Hash Algorithm[EB/OL]. (2004)[2004-04-20]. www.opencores.org, 2004.

[5] Dominikus S. A hardware implementation of MD4-family hash algorithms[C]// Proceedings of IEEE International Conference on Electronics Circuits and Systems (ICECS'02). Croatia: IEEE, 2002; 1143-1146.

[6] Sklavos N, Koufopavlou O. On the hardware implementations of the SHA-2 (256, 384, 512) hash functions [C]// Proceedings of the 2003 International Symposium on Circuits and Systems[s.l.]: 2003(5); 153-156.

[7] Sklavos N, Kitsos P. Random number generator architecture and VLSI implementation[J]. IEEE International Symposium on Circuits & Systems, (ISCAS), 2002 (4); 854-857.

[8] Sun Wanzhong, Guo Hongpeng, He Huilei, et al. Design and optimized implementation of the SHA-2 (256, 384, 512) hash algorithms[C]// IEEE 7th International Conference on ASIC. Guilin: IEEE, 2007; 858-861.

作者简介:

应建华 男, (1954-), 副教授. 研究方向为超大规模集成电路设计.