第33卷 第14期 计 算 机 工 程 2007年7月
Vol.33 No.14 Computer Engineering July 2007

·开发研究与设计技术· 文章编号：1000—3428(2007)14—0270—02 文献标识码：A 中图分类号：TP391

基于FPGA的SHA-1算法的设计与实现

孙 黎，慕德俊，刘 航

(西北工业大学自动化学院控制与网络研究所，西安 710072)

摘 要：SHA-1算法是目前常用的安全散列算法，被广泛地应用于电子商务等信息安全领域。为了满足安全散列算法的计算速度，该文将SHA-1分成5个硬件结构模块来实现，每个模块可以独立工作。对其进行了优化，达到了缩短关键路径的目的，提高了计算速度。独立的模块使得对每个模块的修改都不会影响其他模块的工作，为模块的进一步优化提供了方便。

关键词：散列算法；SHA-1算法；现场可编程门阵列；硬件描述语言

Design and Implementation of SHA-1 Algorithm Based on FPGA

SUN Li, MU Dejun, LIU Hang

(Institute of Control and Network, College of Automation, Northwestern Polytechnical University, Xi'an 710072)

【**Abstract**】Secure hash algorithms are important tools for cryptographic application such as digital signatures. To satisfy the requirements of computation speed, the paper divides SHA-1 into five modules of hardware architecture. Every module works independently and every module is optimized. It do not affect other modules when one module is modified.

【**Key words**】hash algorithm; SHA-1 algorithm; FPGA; VHDL

Hash函数主要用于提供信息交换时的完整性，验证数据在传输过程中是否被篡改。它赋予一个消息唯一的“指纹”，通过验证该消息的“指纹”，就可以辨别出该消息的完整性。其特点是加密数据时不需要密钥，并且经过加密的数据无法解密还原，只有使用相同的单向散列函数对同样的数据进行加密，才能得到相同的结果。

在实际应用中，常用的Hash函数有：MD5，SHA-1和MAC等。

1 SHA-1算法

(1)消息填充和附加原始消息长度。填充的目的是使得填充后的消息长度为512的整数倍，满足SHA-1以512位消息分组进行处理的目的。

(2)预先定义的函数和常数。预先定义一个函数$f(t,B,C,D)$，$0 \le t \le 79$，分别代表从0～79步的处理，B,C,D为32位输入，产生一个32位的输出：

$$f(t,B,C,D)=\begin{cases}(B\ and\ C)\ or\ (\overline{B}\ and\ D) & (0 \le t \le 19)\\ B\ xor\ C\ xor\ D & (20 \le t \le 39)\\ (B\ and\ C)\ or\ (Band\ D)\ or\ (C\ and\ D) & (40 \le t \le 59)\\ B\ xor\ C\ xor\ D & (60 \le t \le 79)\end{cases}$$

同时还要使用到一个预先定义的32位常数$K(t)$，用16进制的值如下：

$$K(t)=\begin{cases}5A827999 & 0 \le t \le 19\\ 6ED9EBA1 & 20 \le t \le 39\\ 8F1BBCDC & 40 \le t \le 59\\ CA62C1D6 & 60 \le t \le 79\end{cases}$$

(3)初始化摘要缓冲。SHA-1需要5个32位的缓冲区（A,B,C,D,E）来保存每一分组的处理结果，用于下一个分组的输入。初始化为：A＝67452301；B＝EFCDAB89；C=98BADCFE；D=10325476；E=C3D2E1F0。

(4)循环处理每一个分组。将填充后的消息以512位分组，分成L份的消息分组，分组后的消息用$y_0,y_1,...,y_{L-1}$表示。对每一个分组进行循环处理，其中，对于第1个分组y_0使用初始化的A,B,C,D,E作为输入，其余分组用上一分组的摘要输出作为输入，最后一个分组的输出即为消息的摘要输出。

(5)单个消息分组的处理。将消息分组分为16份，每份32位，用$M_0,M_1,...,M_{15}$表示，将它扩充成80份，分别表示为$W_0,W_1,...,W_{79}$，W_t的定义如下：

$$W_t=\begin{cases}M_t & 0 \le t \le 15\\ W_{t-3}\ xor\ W_{t-8}\ xor\ W_{t-14}\ xor\ W_{t-16} & 16 \le t \le 79\end{cases}$$

(6)摘要输出。所有的消息分组处理结束后，A,B,C,D,E保存着该消息SHA-1算法的消息摘要，低位始于A，终于E。

2 SHA-1算法的硬件设计

SHA-1的整体结构可分为5个主要模块：输入预处理模块，存储模块，核心操作模块，输出模块，状态机模块。其中，输入预处理模块完成对明文的填充附加位和填充原始明文长度；存储模块用于将16份32位字的明文M_t扩展成80份32位的W_t；核心操作模块根据状态机模块给出的控制信号，循环完成80步的操作；输出模块完成80步操作后的A,B,C,D,E与初始值AA,BB,CC,DD,EE相加，以及输出最后的结果；状态机模块通过输入信号以及其他模块给它的反馈信号，产生控制输出，控制整个SHA-1模块的运行。如图1。

(1)输入预处理模块：包含两个计数器Counter和Length

明文长度计数器(Length，64位)用于计算并保存原始明文的长度，在接到复位信号(reset＝1)或起始信号(start＝1)时清零，在finish＝1后停止计数。

另一个明文分组计数器(Counter，4位)，初始化为“0000”，每输入一个32位的明文，它的计数值就加“1”，当它计数满16个时，即表示正好输入一个明文分组(512位)，停止计数，

基金项目：西北工业大学青年教师创新基金资助项目

作者简介：孙 黎(1981－)，女，硕士研究生，主研方向：网络信息安全；慕德俊，教授、博士生导师；刘 航，博士、讲师

收稿日期：2006-07-30 **E-mail**：sunlihl@mail.china.com

同时存储模块不再从输入端 din 获得数据。

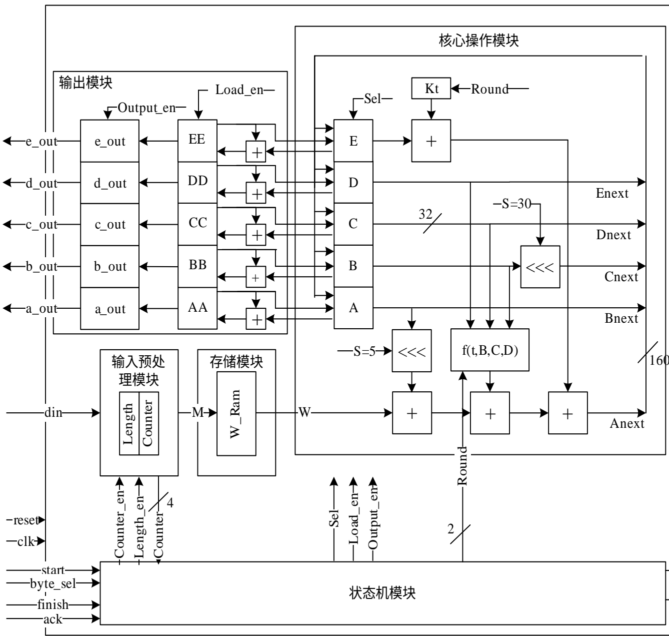


图 1　SHA-1 算法整体结构

(2)存储模块

存储模块用于将 16 份的M_t扩展到 80 份的W_t，对于$0 \le t \le 15$，须$W_t = M_t$即可，对于$16 \le t \le 79$，要用到前面的W_{t-3},W_{t-8},W_{t-14},W_{t-16}。计算出W_t后，不再需要W_{t-16}。根据这个特点，设计一个深度为 16，宽度为 32，具有 5 个输出端口和一个输入端口的队列缓冲器（先入先出）。仅需要一个周期即可完成M_t到W_t的转换。

(3)核心操作模块

SHA-1 的核心操作模块只涉及到一个函数$F_t(B,C,D)$，4 个加法和 1 个左移固定的移位操作。其表达式为

$(a<<<5)+F_t(b,c,d)\ +e\ +W_t+K_t$

可以对核心操作表达式进行并行优化，则优化后的表达式为

$(F_t(b,c,d))+(e+K_t))+((a<<<5)+W_t)$

表达式中加法之间的括号不能省略，因为所表示的优先级不同，Quartus II 综合出来的效果也不同。对于加法和循环移位操作，QuartusII 提供了现成的可调用模块，只要通过 MegaWizard Plug-In Manager 工具实例化就可实现。

(4)输出模块

1)初始摘要寄存器 A,B,C,D,E，用于保存每一个分组的初始值，在接到复位信号（reset＝1）或起始信号（start＝1）时，将 A,B,C,D,E 初始化。

2)4 个加法器，完成 64 步操作后的 a,b,c,d,e 与初始值 A,B,C,D,E 相加，并将结果保存在 A,B,C,D,E 中，作为下一个分组的初始值。

3)4 个输出寄存器，在没有输出时为高阻“Z”，在输出有效时产生一个输出有效信号(valid＝1)。

(5)状态机模块

通过 4 个子状态机的交互运行来控制整个 SHA-1 模块运行。其中，顶层状态机(main_state)包含 4 个状态，用于控制 SHA-1 模块分组完成全部明文的处理；中间一层状态机(pro_state)包含 4 个状态，用于控制单个明文分组(512 位)的处理；底层状态机(Round 和 Step)负责具体每一步的运行控制。

顶层状态机(main_state)的状态变换关系如图 2 所示，空闲或复位后(reset＝1)为 Idle 状态，当开始处理明文时(start＝1)，进入中间等待状态(Mid_wait)；在收到一个输入确认信号(ack＝1)后进入处理状态(Proc)，完成一个明文分组的处理；如果该分组不是明文的最后一个分组(finish＝0)则进入中间等待状态，如果是最后一个分组(finish＝1)，则进入输出状态(Output)；在输出状态输出结果，并跳到空闲状态(Idle)。

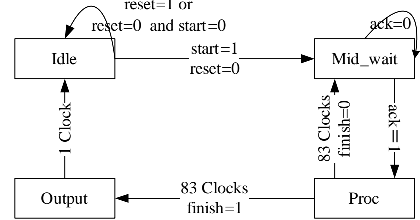


图 2　main_state 状态机

中间一层的状态机（Proc）控制单个明文分组的处理，由于要循环复用核心操作模块来完成 80 步的处理，因此该状态机至少要执行 80 个时钟，还需要一个时钟完成 80 步处理后的 a,b,c,d,e 与初始值 A,B,C,D,E 相加。因此理论上整个的状态机包含 81 个时钟周期，实际使用中由于存储模块输入到输出之间需要两个周期，总共是 83 个时钟，图 3 为该状态机的状态转换图。图中初始状态(Default)和输入等待(Temp)完成明文输入，Norproc 完成 80 步的循环操作，Finproc 完成 80 步处理后的 a,b,c,d,e 与初始值 A,B,C,D,E 相加，最后回到初始状态。这个状态机在顶层状态机处于处理状态(Proc)时开始运行。

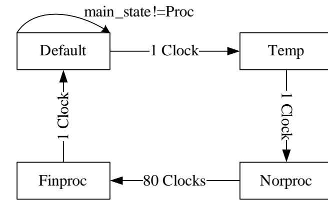
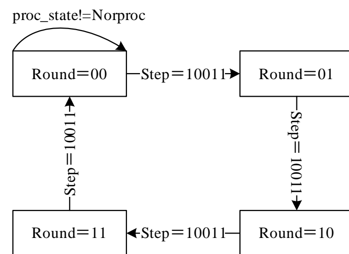
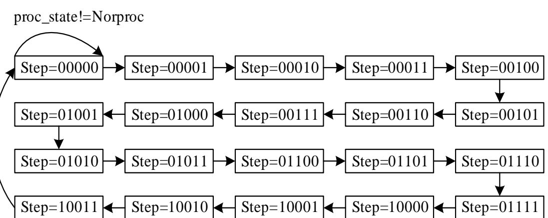


图 3　Proc_state 状态机

底层状态机包含两个状态机，分别为 Round 和 Step。Round 包含 4 个状态，分别代表四轮的处理。Step 包含 20 个状态，表示每一轮 20 步的操作。这个状态机在 pro_state 处于 Norproc 时开始运行，见图 4。



(a)Round的状态转换



(b)Step的状态转换

图 4　Round 和 Step 状态机　（下转第 274 页）