Design and FPGA Implementation of Hash Processor

Murat Aşkar, Tuğba Şıltu Çelebi

***Abstract*— Related to the tremendous developments in the wired and wireless communications area, the demand for secure data transmission increases. In order to find solutions for this increasing demand new algorithms and security standards are developed. Among these algorithms and standards, hash functions are widely used. In this paper design and implementation of an FPGA based hash processor is described. The proposed hash processor consists of an arithmetic logic unit, a message computation block, a constants ROM, a register file, a programmable control unit, program memory and standard UART serial interface. Hash processor is configured by the instructions in the program memory and supports SHA1 and SHA-256 hashing. The hardware is described in VHDL and verified on Xilinx FPGAs.**

***Index Terms*— VHDL, hash function, cryptography, processor**

I. INTRODUCTION

Hash functions are very important cryptographic primitives and used in the several fields of communication integrity and signature authentication. These functions are used to obtain a fixed-size fingerprint or hash value of an arbitrary long message. “One-wayness” and “collision resistant” are important characteristics of hash functions. One-wayness means that the method to calculate a hash value from a given message is easy but it is computationally infeasible to generate any message that corresponds to a given hash value. Collision resistant means that it is very difficult to find two messages that have the same hash value.

The known hash functions are MD4 (128 bit), MD5 (128 bit), RIPEMD-128 (RIPE RACE Integrity Primitives Evaluation), SHA1, SHA-256, SHA-384 and SHA-512.

NIST (National Institute of Standards and Technology) standardized four of these hash algorithms SHA1, SHA-256, SHA-384 and SHA-512 whose hash value sizes are 160, 256, 384 and 512 as SHS (Secure Hash Standard) and a 224-bit hash function, SHA-224 based on SHA-256 was added to SHS on 2004.

Murat Aşkar, Orta Doğu Teknik Üniversitesi, Bilgisayar Mühendisliği Bölümü, İnönü Bulvarı, 06531, Ankara

Tuğba Şıltu Çelebi Aselsan Mehmet Akif Ersoy Mah. 16.cad. No:16 Macunköy 06370Ankara

In this paper the design and FPGA implementation of a hash processor is given. The proposed hash processor consists of an arithmetic logic unit, a register file, a programmable control unit, program memory and standard UART serial interface.

Hash processor is configured by the instructions in the program memory and supports SHA1 and SHA-256 hashing. The hardware is described in VHDL and verified on Xilinx FPGAs.

In Section 2 a brief description of hash functions especially on SHA1 and SHA-256 is given, in Section 3 design and implementation details are presented and finally conclusions are given.

II. HASH FUNCTIONS

Secure hash standard defines four secure hash algorithms, SHA1, SHA-256, SHA-384 and SHA-512. All four of the algorithms are iterative one-way hash functions that process a message to produce a condensed representation called a message digest.

Each algorithm can be described in two stages. The first stage is the preprocessing stage. In this stage the message is padded, parsed into n blocks and the values that are going to be used in hash function are initialized. In the second stage hash calculation is done. Hash calculation generates a message schedule from the padded message and uses that schedule, along with functions, constants and word operations to iteratively generate a series of hash values. The final hash value generated by the hash computation is used to generate the message digest [2].

The security of the hash algorithm is directly related to the message digest length. In this context “security” refers to the fact that a birthday attack on a message digest of size n produces a collision with a work factor of approximately $2^{n/2}$ [5].

Birthday attack is a particular class of attack against one-way functions. It does find two inputs that correspond to the same output. Birthday attacks mean that under some conditions digest algorithms that are n bits wide and which have an a-priori difficulty of $O(2^n)$ can be cracked with just $O(\text{sqrt}(2^n))$ effort. It gets its name from the surprising result that the probability of two or more people in a group of 23 sharing the same birthday is greater than ½ [6].

In this study we have chosen to implement SHA1 and SHA-256 hash algorithms. The properties of SHA1 and SHA-256 hash functions are given below in TABLE I: The functions and constants used in SHA1 and SHA-256 are described in sections A, B.

TABLE I
PROPERTIES OF SHA1 AND SHA-256 HASH FUNCTIONS

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)	Security (bits)
SHA1	$< 2^{64}$	512	32	160	80
SHA-256	$< 2^{64}$	512	32	256	128

A. SHA1 Computation Flow

SHA1 calculation is completed in 80 rounds and 5 hash variables each of 32 bits are used. . The word size of all the calculations is 32 bits. The padded message is processed by 512 bit blocks This 512 bit block is composed of 16 message words. These 16 message words are expanded by means of functions and in each of the total 80 rounds a new message word is used. The computation is done as follows:
The hash variables shown in TABLE II are initialized.

TABLE II
INITIAL HASH VALUES of SHA1

H_0	H_1	H_2	H_3	H_4
67452301	EFCDAB89	98BADCFE	10325476	C3D2E1F0

The message schedule is prepared, ie. the message word that is going to be used in that round is prepared. This computation is done as described in (1).

$$\begin{aligned} W_t &= M_t^i && 0 \le t \le 15 \\ W_t &= ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) && 16 \le t \le 79 \end{aligned} \quad (1)$$

The 5 working variables A, B, C, D and E that are going to be used in the computation are prepared as in (2).

$$\begin{aligned} A &= H_0^{(i-1)} \\ B &= H_1^{i-1} \\ C &= H_2^{i-1} \\ D &= H_3^{i-1} \\ E &= H_4^{i-1} \end{aligned} \quad (2)$$

After these initializations, the final values of the working variables for that round are calculated as described in (3).

$$\begin{aligned} TEMP &= S\hat{}5(A) + f(t;B,C,D) + E + W_t + K_t \\ E &= D \\ D &= C \\ C &= S\hat{}30(B) \\ B &= A \\ A &= TEMP \end{aligned} \quad (3)$$

The value of the function f(t, B, C, D) in (3) changes with the round number t. These values are described below in (4).

$$\begin{aligned} f(t;B,C,D) &= (B \wedge C) \vee ((\neg B) \wedge D) && (0 \le t \le 19) \\ f(t;B,C,D) &= (B \oplus C \oplus D) && (20 \le t \le 39) \\ f(t;B,C,D) &= (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) && (40 \le t \le 59) \\ f(t;B,C,D) &= (B \oplus C \oplus D) && (60 \le t \le 79) \end{aligned} \quad (4)$$

As the final step, intermediate hash values are calculated as described in (5).After 80 rounds the hash value of the incoming 512 bit message block is obtained.

$$\begin{aligned} H_0^i &= A + H_0^{i-1} \\ H_1^i &= B + H_1^{i-1} \\ H_2^i &= C + H_2^{i-1} \\ H_3^i &= D + H_3^{i-1} \\ H_4^i &= E + H_4^{i-1} \end{aligned} \quad (5)$$

Basic SHA1 computation flow described above is shown below in Fig.1.

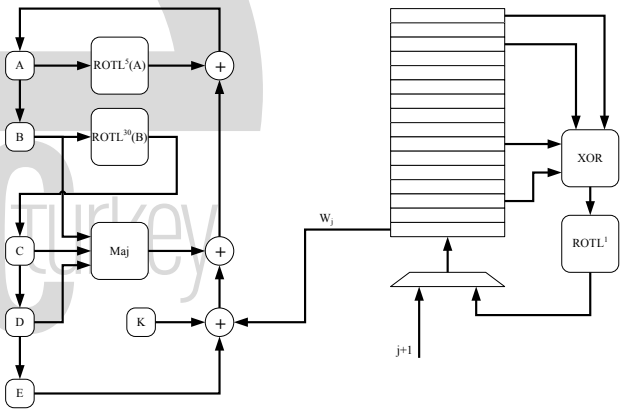

Fig. 1 SHA1 computation flow

B. SHA-256 Computation Flow

SHA-256 calculation is completed in 64 rounds and 8 hash variables each of 32 bits are used. . The word size of all the calculations is 32 bits. The padded message is processed by 512 bit blocks This 512 bit block is composed of 16 message words. These 16 message words are expanded by means of functions and in each of the total 64 rounds a new message word is used. The computation is done as follows:
The hash variables shown in TABLE III are initialized.

TABLE III
INITIAL HASH VALUES of SHA-256

H_0	H_1	H_2	H_3
67452301	BB67AE85	3C6EF372	A54FF53A
H_4	H_5	H_6	H_7
510E527F	9B05688C	1F83D9AB	5BE0CD19

The message schedule is prepared, ie the message word that is going to be used in that round is prepared. This computation is done as described in (6).

$$W_t = \begin{cases} M_t^i & 0 \le t \le 15 \\ \sigma_1^{256}(W_{t-2}) + W_{t-7} + \sigma_0^{256}(W_{t-5}) + W_{t-16} & 16 \le t \le 63 \end{cases} \quad (6)$$

The 8 working variables A, B, C, D, E, F, G, H that are going to be used in the computation are prepared as in (7).

$$\begin{aligned} A &= H_0^{(i-1)} \\ B &= H_1^{i-1} \\ C &= H_2^{i-1} \\ D &= H_3^{i-1} \\ E &= H_4^{i-1} \\ F &= H_5^{i-1} \\ G &= H_6^{i-1} \\ H &= H_7^{i-1} \end{aligned} \quad (7)$$

After these initializations, the final values of the working variables for that round are calculated as described in (8).

$$\begin{aligned} T_1 &= H + \sum_1^{256} E + Ch(E,F,G) + K_t + W_t \\ T_2 &= \sum_0^{256} A + Maj(A,B,C) \\ H &= G \\ G &= F \\ F &= E \\ E &= D + T_1 \\ D &= C \\ C &= B \\ B &= A \\ A &= T_1 + T_2 \end{aligned} \quad (8)$$

As the final step, intermediate hash values are calculated as described in (9).After 64 rounds the hash value of the incoming 512 bit message block is obtained.

$$\begin{aligned} H_0^i &= A + H_0^{i-1} \\ H_1^i &= B + H_1^{i-1} \\ H_2^i &= C + H_2^{i-1} \\ H_3^i &= D + H_3^{i-1} \\ H_4^i &= E + H_4^{i-1} \\ H_5^i &= F + H_5^{i-1} \\ H_6^i &= G + H_6^{i-1} \\ H_7^i &= H + H_7^{i-1} \end{aligned} \quad (9)$$

Basic SHA-256 computation flow described above is shown below in Fig.2.

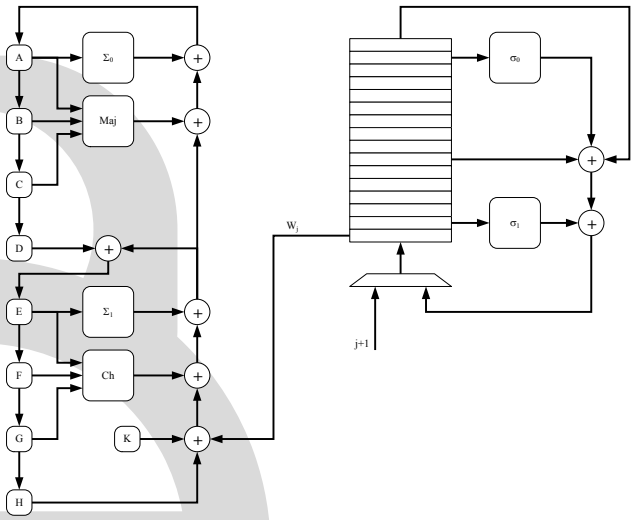

Fig. 2 SHA-256 computation flow

III. SYSTEM ARCHITECTURE AND IMPLEMENTATION

In order to implement the hash processor hash functions are examined in detail. When doing this examination, our main concern was how we can create similar or common instructions for these algorithms. After examining the functions in detail we have chosen to design a processor that have the ability of implementing SHA1 and SHA-256 hash functions as a first step and developed the instructions accordingly. The main reason of choosing to implement SHA1 and SHA-256 as a starting point was that these algorithms are widely used and have the same word and block sizes. The instructions developed for the hash processor are given below in TABLE IV. Among the 15 instructions developed, 6 instructions are special hashing instructions the remaining 9 instructions are common instructions such as loading the accumulator with the value in the addressed memory location.

TABLE IV HASH PROCESSOR INSTRUCTIONS

Instruction Mnemonic	Opcode	Brief Description
LDA loc	0000	Load accumulator with the contents of the memory location
AND loc	1001	AND accumulator with the contents of the memory location
ADD loc	0010	ADD the contents of the memory location to accumulator
SUB loc	0011	SUB the contents of the memory location from accumulator
JMPP addr	0110	Jump to address if the content of the acc is positive
STA loc	1000	Store accumulator to memory location
JMPZ addr	1111	Jump to address if the content of the acc is zero
SHA1	0001	generates the necessary control signals fort he datapath to make one step SHA1 calculation
SHA2	1010	generates the necessary control signals fort he datapath to make one step SHA-256 calculation
SRGF1	1011	stores the intermediate hash values of the SHA1 algorithm to the register file
SRGF2	1101	stores the intermediate hash values of the SHA-256 algorithm to the register file
RRGF1	1100	reads the intermediate hash values of the SHA1 algorithm from the register file
RRGF2	1110	reads the intermediate hash values of the SHA-256 algorithm from the register file.
HALT	0111	terminates the execution of the processor

The proposed hash processor consists of an arithmetic logic unit, a register file, a message computation block, a constants rom, a programmable control unit, program memory and standard UART serial interface. The block diagram of the hash processor is given below in Fig.3.

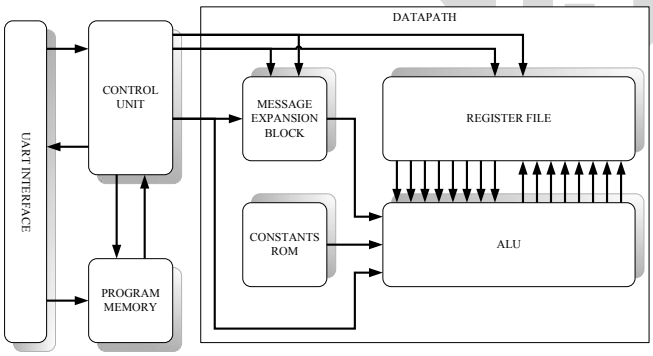

Fig. 3: Hash function processor block diagram

The message expansion block consists of an 80x32 block ram and some extra logic for message computation.

Register file consists of 16 registers each of 32 bits wide and holds the intermediate hash values.

Constants Rom is a read only memory of size 80x64 which holds the constants required by hash functions SHA1 and SHA-256.

Arithmetic logic makes the computations that are determined by the control unit. Program memory is a block ram of size 32x32.

All the algorithms are first described in VHDL and their description is verified through the functional simulation using Xilinx ISE software platform. Test vectors which are determined by NIST (National Institute of Standards and Technology) are used for debugging and verification of VHDL codes. Each instruction hash been tested and verified. An example assembly code that implements SHA-256 calculation is given below in TABLE V.

TABLE V
ASSEMBLY CODE FOR SHA-256 CALULATION

Instruction	Description
LDA 1E:*Loop*	Load accumulator A with the content of the memory location 1E (the content of the memory location is initially zero)
RRGF2	read the intermediate hash values of the SHA-256 algorithm from the register file.
SHA2	generate the necessary control signals fort he datapath to make one step SHA-256 calculation
SRGF2	store the intermediate hash values of the SHA-256 algorithm to the register file
LDA 1E	Load accumulator A with the content of the memory location 1E
ADD 1D	ADD the content of the memory location 1D to the value in accumulator A (the content of the memory location 1D is one)
STA 1E	Store the content of the accumulator A to the memory location 1E
LDA 1F	Load accumulator A with the content of the memory location 1F(the content of the memory location 1F is 64)
SUB 1E	Subtract the content of the memory location 1E from the value in the accumulator A
JMPZ 09	Jump to Halt if the result of the previous subtraction is zero (ie 64 round is completed)
JMPP 00	Jump to Loop is the result of the previous subtraction is positive (ie 64 round is not completed)
HALT:*Halt*	Terminate the execution

The design is implemented and tested on Xilinx 4vsx35ff668-12 FPGA. on Xilinx's ML402 development board. The board used is shown below in Fig.4:

Fig. 4. ML402 Development Board

The synthesis report of the design is shown below on TABLE VI.

TABLE VI
SYNTHESIS RESULTS OF HASH PROCESSOR

Number of Slices:	2494 out of 15360 16%
Number of Slice Flip Flops:	2793 out of 30720 9%
Number of 4 input LUTs:	3872 out of 30720 12%
Number of bonded IOBs:	350 out of 450 77%
Number of FIFO16/RAMB16s:	4 out of 192 2%
Number used as RAMB16s:	4
Number of GCLKs:	16 out of 32 50%

IV. CONCLUSION

In this study hash functions SHA1 and SHA-256 are implemented in a processor structure using the same hardware blocks. These hash functions are examined in detail and a new instruction set is proposed. Hash processor is fully designed and captured using VHDL in Xilinx ISE software environment. The design is also implemented on Xilinx FPGA and implementation results are given.

This study can be starting point for future studies and can be extended by developing special instructions for the hash functions other than SHA1 and SHA-256.

REFERENCES

[1] FIBS PUB 180-1 Secure Hash Standard
[2] FIBS PUB 180-2 with change notice Secure Hash Standard
[3] Tim Grembowski, Roar Lien, Kris Graj, Nghi Nguyen, Peter Bellows, Jaroslav Flidr, Tom Lehman, Brian Schott. Comparative Analyses of the Hardware Implementations of Hash Functions SHA1 and SHA-512. *In: Proc. of the 5th Int'l Information Security Conf., Sao Paulo, Brazil (2002)*
[4] Yong kyu Kang, Dae Won Kim, Taek Won Kwon, Jun Rim Choi. An Efficient Implementation of Hash Function Processor for IPSEC. *In Proceedings of the 2002 IEEE Asia-Pacific Conference on ASIC, Taipei, Taiwan, Aug. 2002*
[5] N. Sklavos , O. Koufopavlou. On the Hardware Implementations of the SHA-2 (256, 384, 512) hash functions. *Proceedings of IEEE International Symposium on Circuits & Systems (ISCAS'03),Vol. V, pp. 153-156, Thailand, May 25-28, 2003*
[6] A. Menezes, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography. CRC Press Inc., October 1997
[7] Akashi Satoh, Tadanobu Inoue. ASIC-Hardware-Focused Comparison for Hash Functions MD5, RIPEMD-160, and SHS. *Proceedings of International Conference Information Technology:Coding and Computing (ITCC'05 0-7695-2315-3/05)*