SHA-1 CRYPTOGRAPHIC ALGORITHM IMPLEMENTED ON FPGA

Vasile-Gabriel IANA[1], Petre ANGHELESCU[1],
Gheorghe SERBAN[1], Cristian-Iulian RINCU[2]
[1]University of Pitesti, Department of Electronics, Computers, and Electrical Engineering, Romania
[2]Military Technical Academy, Bucharest, Romania
[1]petre.anghelescu@upit.ro, [2]r_iulian@mta.ro

Keywords: Cryptography, Hash function, SHA-1, FPGA.

Abstract: *The SHA algorithm is one well-known and represents an important function in cryptographic systems used especially to assure data integrity and authenticity. Implementation on reprogrammable hardware structures is a solution that enables synthesis of algorithms in digital hardware modules that may operate at high speeds. In this work is being studied algorithm implementation of SHA-1, 24 bits on a Field Programmable Gate Array (FPGA) in order to obtain solutions to reduce the area of deployment. There are described constituent modules of the implementation and at the end of the paper are presented comparative results with other similar implementations.*

1. INTRODUCTION

In communication systems, such as mobile telephone networks, internet network are known as a huge expansion of the transmission of information and communication. Also, in medical field where the electronic devices collect and store data from patients, in automotive field, electronics which has reached a very high level; there are critical communications environments among the various entities of the engine and vehicle. There are developed hardware modules for storing and transporting the information on electronic support, like Subscriber Identifier Module (SIM) cards. Along with the development of technology we discover a growing demand for the protection of the exchanged information, leading to the study and development of techniques for more efficient information encryption, data integrity verification, and authenticity. Methods to protect information exchanged scrambled messages presents a very large area of interest such as the communication between banks, protection of databases, or protection of communication channels. Needed of communications and fast growing of the information submitted has led to the development of new techniques for security and protection of the information in the hardware structure answering at a speed much greater than general-purpose processors. They are created specialized circuits as well as Application Specific Integrated Circuit (ASIC), reprogrammable structures FPGA that have been implemented hardware algorithms data security and are used as hardware peripherals for the micro-systems with microprocessors or microcontrollers or used like independents hardware entity [1].

In security applications important roles is played by well-known cryptographic hash functions that are used to verify integrity message and digital signatures. In cryptography, a cryptographic hash function is defined as a transformation that receives a message on input and returns a fixed length string called the hash value. Hash functions with this property are used for a variety of computations, including cryptography. The hash is a concrete representation of the message or document from which it was calculated. Summary message can be seen as a fingerprint of the document. In many

standards and applications, the hash functions used are MD5 and SHA-1. After identifying certain security problems have developed superior version as SHA-2 [2] and studies are currently on the implementation of SHA-3 function [3].

The SHA-1 algorithm can be described in two stages: preprocessing and calculation of hash. The preprocessing involves a padded message, then division of the message into blocks of *m* bits and setting initialization values to be used for the transformation for hash valuation. Hash calculation generates a sketch of the message from padded message and together with other functions; constants and words work to iteratively generate a series of hash values. Last hash value generated by the phase calculating hash map is used to determine the summary of the message [4].

2. BASICS OF RECONFIGURABLE STRUCTURES

Encryption algorithms can be implemented in software or hardware. Software implementation is not recommended because it is slow. Modern hardware implementations are realized on development board with FPGA.

The devices are user programmable integrated circuits that allow quick access to configurable VLSI circuits. In this paper, the implementation of SHA-1 function was performed on a Spartan 3, Xilinx [6] hardware structures.Architecture of Xilinx FPGA families consists of the following building blocks Configurable Logic Blocks (CLB), Input/Output Blocks (IOB), block random access memory (BRAM), multipliers and digital clock manager (DCM).

Basic elements in FPGA structure are CLB's. They are used to implement combinational and synchronous logic. Each CLB has two pairs of two slices as shown in Fig. 1. Slices are grouped in pairs, each pair is organized on columns, with each independently carry channel. All slices have the following common elements: two LUT, two D flip-flops, multiplexers, logic circuits used in combinational logic. Some slices have distributed RAM.

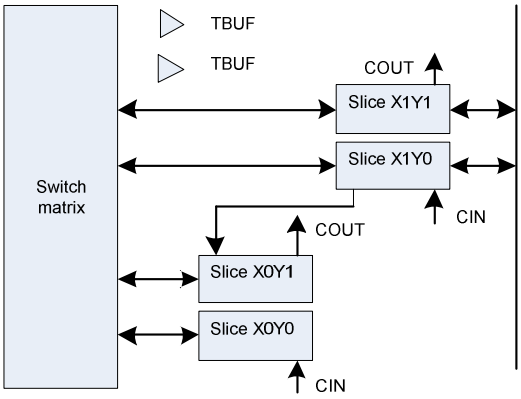

Fig. 1 Schematic of configurable logic blocks

3. HARDWARE DESIGN

Implementation presented in this paper allows to the input port a 24-bit fixed-length message and produces to the output port a block of 160 bit message summary. The maximum length of the message can be maximum 264 bits [4]. Entity SHA-1 module is shown in Fig. 2.

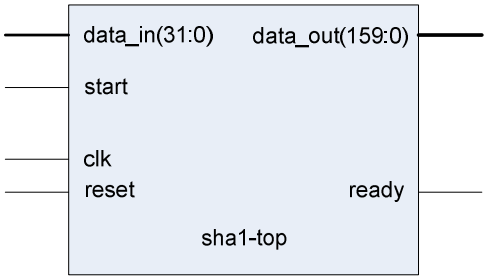

Fig. 2 SHA-1 module entity

The algorithm implementation is done in hardware modules structural interconnected as in Fig. 3. The *Padding* block divide the expanded message into 16 words W of 32-bit. These words come in W_CALC block where the other 64 are calculated using the 16 original words. Totally, 80 words are obtained to be used for the 80 steps of the algorithm.

The 80 words W get together with other constants and variables in *DIGEST_FUNC* block where functions are applied to generate abstract message. At each step the block calculates the intermediate values of variables a, b, c, d, e. These variables then pass to DIGEST_MSG block where summation occurs with H constants, and then the variables are concatenated, the result representing the summary message.

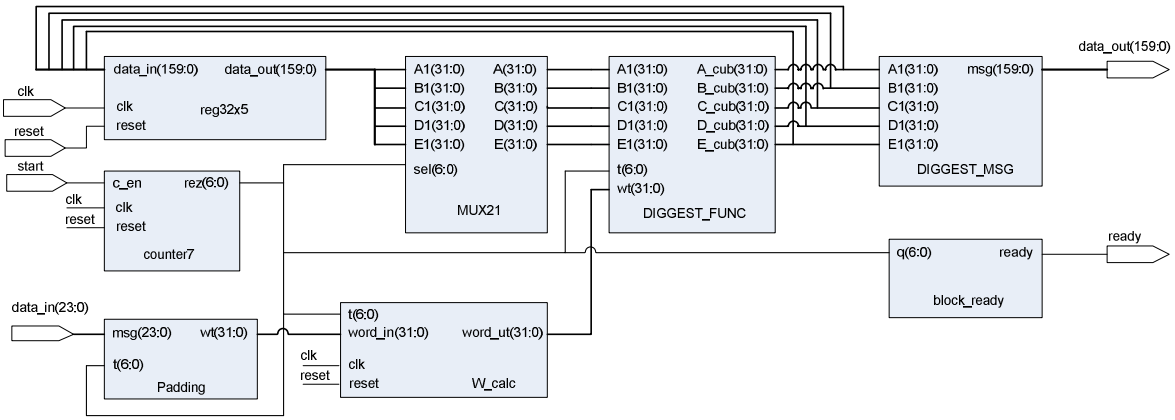

Fig. 3 SHA-1 schematic bloc

The system comprises eight main blocks for SHA-1 algorithm implementation. As mentioned above, the input message is fetched from the block on port *msg Padding (23:0)* in segments of one 24 bit. A second *input t (6:0)* provides to the block witch step algorithm is. Depending on the step the out port *wt (31:0)*, the Padding block will provide to the other blocks one of the 16 words, where the extended message was split properly to the first 16 steps of the process of calculating the hash value.

The first block that processes the extended message is named *W_calc*. Thus it receives at every step of the first 16 steps a word provided by the *Padding* block and then is provided to the output. After the 16-th step, the *W_calc* compute the word for current step from the first 16 steps taken from the *Padding* block. Internal block diagram of *W_calc* is shown in Fig. 4.

As is shown in block diagram of *W_calc*, the 16 registers containing the 16 original words that make extended message are connected in pipeline (one exit is connected to the another entry) in order to move words from one register to another, in a serial way, so that after 16 steps the first register will contain the first word and in the 16-th register will find the word with number 16. Also observe a multiplexer which is designed to provide at the output one of the 16 original words, if the step is less than or equal to 16 or word calculated at each step if the step is greater than 16. Words are calculated using 4-input adder present in the internal block diagram *W_calc*. The adders module add the registers 14, 8, 3 and 1 which then contains the 16 original words at the moment.

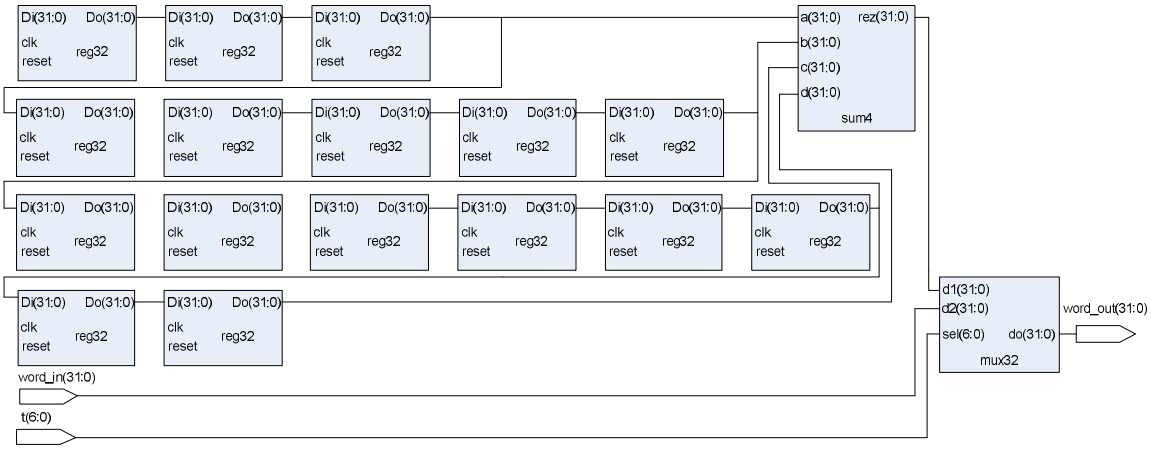

Fig. 4 Schematic bloc of W_calc module

The block that calculates the hash value is *Digest_func*. The input ports A (31:0), B (31:0), C(31:0), D(31:0) and E(31:0) designate five intermediate variables which are found in

the hash value calculation as is shown in Fig. 5. At each step of the algorithm on these variables is applied a lot of functions resulting new values of those variables that will be used in the next step.

Functions used are reported in the current step, so entry is present and t(6:0) indicating the current step of the algorithm. Also the calculation of intermediate values of the variables involved and block words from *W_calc* that are taken at each step, the entry wt (31:0).

As I said, the new values of intermediate variables are found at output ports of the block. It can be seen from Fig. 5 that they were marked with A_out(31:0), B_out(31:0), C_out(31:0), D_out(31:0) and E_out(31:0). All these intermediate variables are each defined on 32-bit.

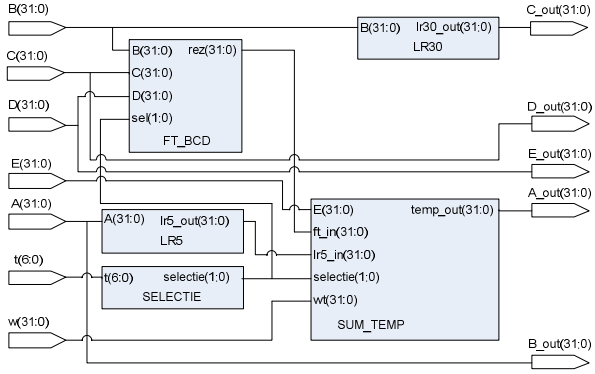

Fig. 5 Schematic block of digest_func module

The functions of this block are implemented through in blocks which than we will describe below:

- The first is called *FT_BCD* module and implement logic functions described in the SHA-1 algorithm [5]. Logic functions operate on the three inputs of module, words A (31:0), B (31:0) and C (31:0) and vary depending on the computing step. Selection is made through input port sel(1:0). Output port of the module is a temporary word witch become entrance for SUM_TEMP module, which will be presented later;

- Another block called *LR30* done rotation to the left with 30-bit of input variable B, the result of this operation is C_out(31:0) output variable from *Digest_func* block;

- LR30 is a similar block with LR5 to rotate to the left the bits of intermediate variable A, the difference between blocks is being made only 5-bit block turn LR30, unlike 30-bit block rotate as LR30. LR5 block output is the intermediate word that reaches *SUM_TEMP* summing block;

- The *SUM_TEMP* block gathers output ports of *FT_BCD* and *LR5* modules, input word wt (31:0) from *W_calc* block, temporary variable E and constant K are stored in this module. Such logical functions from *FT_BCD* module and constants K from *SUM_TEMP* module vary depending on the current step of the algorithm. K constant selection for each step is the input port selectie(1:0). Output port of *SUM_TEMP* A_out module reaches output variable A_out(31:0) of *Digest_func* block;

After the words A, B, C, D, E have been applied a number of functions described above we obtained intermediate values, for each step, called A_out, B_out, C_out, D_out and E_out. These intermediate values of the variables come again in *Digest_func* block for new operations at the next step, but also come in *Digest_msg* block where are collected and addend next to the H constants. Although it have done at every step the summing is useful only to the last step where values intermediate from step 80 are summed next to the constants H constant, after which they get concatenated the results and output them to the desired message. Input ports of the block are represented by the five intermediate variables A, B, C, D and E and as output we have calculated the message of the original message from SHA-1 input circuit.

Besides *Digest_func* and *W_calc* blocks, *Digest_msg* block has an important role in calculating of the hash. Basically in these blocks is the processing itself, other building blocks being with control functions

Counter7 block is a 7-bit counter that counts upward, the maximum amount that can reach the 128. It counts the algorithm steps for generate a hash value and has three input ports: clk, reset and c_en. The first two entries are functionally identical to those described in other synchronous blocks. C_en is input pin that controls start of generating summary message. At a higher level is can be identified by the SHA-1 chip trough start input port.

Reg32x5 block contains a 160-bit register that stores every step of the intermediate variables A, B, C, D, E, hence the name Reg32x5 (5 words, each on one 32-bit). The circuit is

synchronous, the input data into register memorizing every time a clock event occurs.

Mux21 block is a synchronous circuit that contains a 160-bit register that stores every step of the intermediate variables A, B, C, D, E, hence the name Reg32x5 (5 words, each on one 32-bit).

Ready block is used to signal the end of the generation of the output message and the *Selectie* block of the Digest_func block is the module that selects the K constants and the logic functions used in computing process to obtain the summary message.

4. RESULTS OF IMPLEMENTATIONS

The hash algorithm was coded in VHDL and synthesized and implemented using the Xilinx ISE tools. After the implementation of SHA-1 algorithm on XC3S400-4fg456 the following performances were obtained: minimum period: 17.910ns (Maximum Frequency: 55.834MHz; minimum input arrival time before clock: 8.652ns; maximum output required time after clock: 27.259ns.

Table 1 Structure utilization for SHA-1 module

Logic used	used	available	Area (%)
Number of Slices	478	3584	13
Number of Slice Flip Flops	465	7168	6
Number of 4 input LUTs	887	7168	12

This module was implemented and optimized for used area and not for the speed processing.

For a practical application the circuits can process a sequence of 24 bits for a time 15.8us with a 50 MHz clock.

It can process approximately 1.5 Gbps. Another works was realized to implement this algorithm and are presented comparative in Table 2.

Table 2 Comparison between different SHA-1 implementations

Reference	Device	Algorithm	Area	Clk	Speed
Seyed [7]	Virtex4 xc4vsx35ff668-12	SHA-512	12%	135Mhz	2890Mbps
Present work	XC3S400-4fg456	SHA-1 24bits	13%	50Mhz	1500Mbps
Jar[5]	Virtex-II 2V2000-6	SIG-SHA-1	-	117.5Mhz	734Mbps
Imt[8]	Stratix EP1S10F484C5	SHA-1 512bits	26%	45.8Mhz	293Mbps
Imt[8]	Stratix EP1S10F484C5	SHA-1 512bits	34%	24.86Mhz	160Mbps
Imt[8]	Stratix EP1S10F484C5	SHA-1 512bits	78%	48.7Mhz	312Mbps

5. CONCLUSIONS

The hash algorithm SHA-1 is used a wide scale in cryptography for different applications. In this paper was implemented the SHA-1 algorithm on a low cost FPGA of type XC3S400.

The SHA-1 design demonstrates that the algorithm can be implemented on small area requirements (practically on 478 numbers of slices) and can process 1.5 Gbps.

It can be extended up to 264 bits input port but will produce a module that will suffer an increased area used and latency trough structure.

In the immediate future, we will combine the SHA-1 algorithm with other one based on bio-inspired systems (as cellular automata) in order to achieve high speed and very good security.

ACKNOWLEDGMENT

This work was supported by CNCSIS UEFISCSU, project number PN II-RU PD 369/2010, Contract No. 10/02.08.2010.

6. REFERENCES

[1]. Wollinger, T., Guajardo, J. and C. Paar, *„Security on FPGAs: State of the Art Implementations and Attacks”*, ACM Transactions on Embedded Computing Systems, Special Issue on Security and Embedded Systems, 2004.

[2]. Ryan Glabba, Laurent Imbert, Graham Jullien, Arnaud Tisserand, Nicolas Veyrat-Charvillon, *„Multi-mode operator for SHA-2 hash functions”*, Journal of Systems Architecture 53, 127–138, Elsevier, 2007.

[3]. X. Guo, M. Srivastav, S. Huang, D. Ganta, M. B. Henry, L. Nazhandali, and P. Schaumont, *“Silicon Implementation of SHA-3 Finalists: BLAKE, Grøstl, JH, Keccak and Skein”*, in ECRYPT II Hash Workshop, 2011, May 2011.

[4]. Federal Information Processing Standards, Secure Hash Standard, FIPS PUB 180-2, August 1, 2002.

[5]. K. Järvinen, *„Design and Implementation of a SHA-1 Hash Module on FPGAs”*, November 2004.

[6]. Xilinx, Inc.: Spartan 6, Field Programmable Gate Arrays, available at www.xilinx.com.

[7]. Seyyed Ali Emam, Sareh Sadat Emami, *„Design and Implementation of a Fast, Combined SHA-512 on FPGA”*, IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.5, May 2007.

[8]. Imtiaz Ahmad, A. Shoba Das, *„Hardware implementation analysis of SHA-256 and SHA-512 algorithms on FPGAs”*, Computers and Electrical Engineering 31, 345–360, Elsevier, 2005.