An Efficient FPGA-Based Architecture of Skein for Simple Hashing and MAC Function

Filippos Pirpilidis
Department of Telecommunication Systems and Networks
Technological Education Institute of Messolonghi
Branch of Nafpaktos, Greece
e-mail: filippakoc@hotmail.com

Paris Kitsos
Computer Science
Hellenic Open University / KNOSSOSnet Research Group
Patras, Greece
e-mail: pkitsos@ieee.org

Nicolas Sklavos
KNOSSOSnet Research Group
Technological Educational Institute of Patras
Patras, Greece
e-mail: nsklavos@ieee.org

***Abstract* - Skein is a hash function that reached the semifinals of the NIST competition for the selection of standard SHA-3. This paper describes the implementation of Skein-512 operating as simple hash function and as MAC function. The design was coded using VHDL language and for the hardware implementation, two XILINX FPGAs, Virtex-6 and Virtex-7 were used. The proposed implementation reaches a data throughput of 894 Mbps at 110 MHz clock frequency for Virtex-6 and a throughput of 975 Mbps at 120 MHz clock frequency for Virtex-7.**

Keywords- SHA-3 competition, Skein hash function, MAC function, VIRTEX FPGA

I. INTRODUCTION

The SHA-3 competition [1] began in 2007, when NIST had reason to suspect that the current encryption method SHA-2 [2], could be compromised. The SHA-2 is currently still considered safe and suitable for general use by the NIST, but a second algorithm gives security system designers more flexibility. Skein [3] is a new hash function, introduced in the end of 2010. It combines speed, security, simplicity, and a great deal of flexibility in a modular package that is easy to analyze. The Skein is a function that reached the semifinals of the NIST competition.

This paper presents the design of Skein hash function on FPGA in dual mode of operation; simple hashing and MAC function. The tests, synthesis and simulation were carried out on the families of XILINX Virtex – 6 [4] and Virtex-7 [5] devices.

The rest of the paper is organized as follows: Section II briefly presents the specifications of Skein. The proposed architecture is explained in Section III while the implementation results and comparisons are given in Section IV. The paper conclusions are discussed in the last section.

II. SKEIN SPECIFICATIONS

The Skein is based on architecture with input block size equals to 256, 512 and 1024 bits [3]. The skein consists of two basic subcomponents; the Threefish block cipher and the Unique Block Identification (UBI). Each of these components is briefly described below:

Threefish: The Threefish block cipher takes as inputs 3 parameters. A message (block of plaintext) with N-bit length where N must be greater than or equals to 256-bit, a Key with the same length as the plaintext block and a variable tweak of 128-bit. The Threefish consists of MIX functions and word permutations. Each MIX function has two inputs and two outputs by the following mathematical and logical operations: MIX (A, B) = (A + B, B <<< R) ^ (A + B), where the symbol "<<<" is left rotation, the symbol "^" is the XOR logic operation and the symbol "+" is the arithmetic addition. The variable R denotes the number of bits to rotate.

The Threefish-512 require 72 rounds to complete the operation with each round contains four MIX functions and one word permutation. According to the specifications the 72 rounds are grouped in nine subsets. Each subset comprises eight rounds and uses different variable R (rotation number). In addition, a subkey is added with the outputs of the corresponding word permutation. The subkeys are generated by the Key Schedule subcomponent according to the specification.

Unique Block Identification (UBI): UBI is a chaining mode, variant of the Matyas-Meyer-Oseas hash mode that uses Threefish to build a compression function that maps an arbitrary input size to a fixed output size. The inputs of UBI consist of three parameters:

- An *Initialization Vector* IV.
- A part of the *plaintext* M
- A variable *type*

For Skein-512 on simple hashing mode initially splits the message *M* in 512-bit blocks, *M1*, *M2*, …, *Mm*. If the length of the last block is not equal to 512 bits then the block goes to the padding process. Then for each block the appropriate tweak value is calculated and finally, the output of UBI is XORed with the input block.

`/13 $26.00 © 2013 IEEE
DOI 10.1109/DSD.2013.119

III. SKEIN CORE ARCHITECTURE

The diagram of the Skein-512 for simple Hashing and Skein_MAC function is shown if Fig. 1. With more details, shows the entire process for hash creation of a message with length 170-bytes. The UBI1 takes as input the *C* which is the *config* string from the specifications, 0 as *key* and *tweak* with type config.

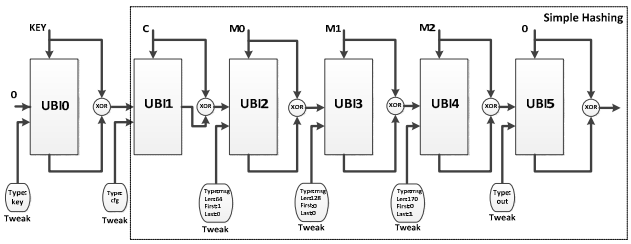

Figure 1. Simple hashing and MAC.

The result of the first block corresponds to IVs that are used by the next UBIs. The next three UBIs processes the entire message; so, the first 64-bytes are processed by the UBI2 with the appropriate tweak, then the next 64-bytes are processed by the UBI3 with the appropriate tweak and the last 42-bytes are processed by the UBI3. Finally the result is fetch as key in the last module, UBI5 that with the appropriate parameters implements the message digest of Skein. For the Skein-MAC the same diagram is used with the different of an extra UBI is used, UBI0, with message input the KEY, UBI key equal to zeroes and type of the tweak as key.

The top level of the proposed architecture is shown in Fig. 2. The basics components are the *Message_Padding* and the *Skein_Core*. Also, there are many multiplexers and some peripheral components. The inputs are: 1) The message block 512-bit that is part of the plaintext, 2) the 16-bit NB parameter which is the decimal number of bytes processed. The control signal CFG becomes "1" when the Skein_Core used to create the IVs and the multiplexer selects the *config* string as input to the skein core component. The mode signal becomes "0" when Skein_core used as output function, and the signals *mac* and *mac_key* are used when the algorithm operates as *MAC* function. The signals *first_t*, *final_t* and *bit_pad* are needed for the construction of the variable tweak. The Message_Padding component implements the padding procedure according to the specifications. The Change_bytes_words component makes the changing position on 64-bit strings. The first byte goes last, the penultimate goes second and so on. The same operation happens with the *strings*.

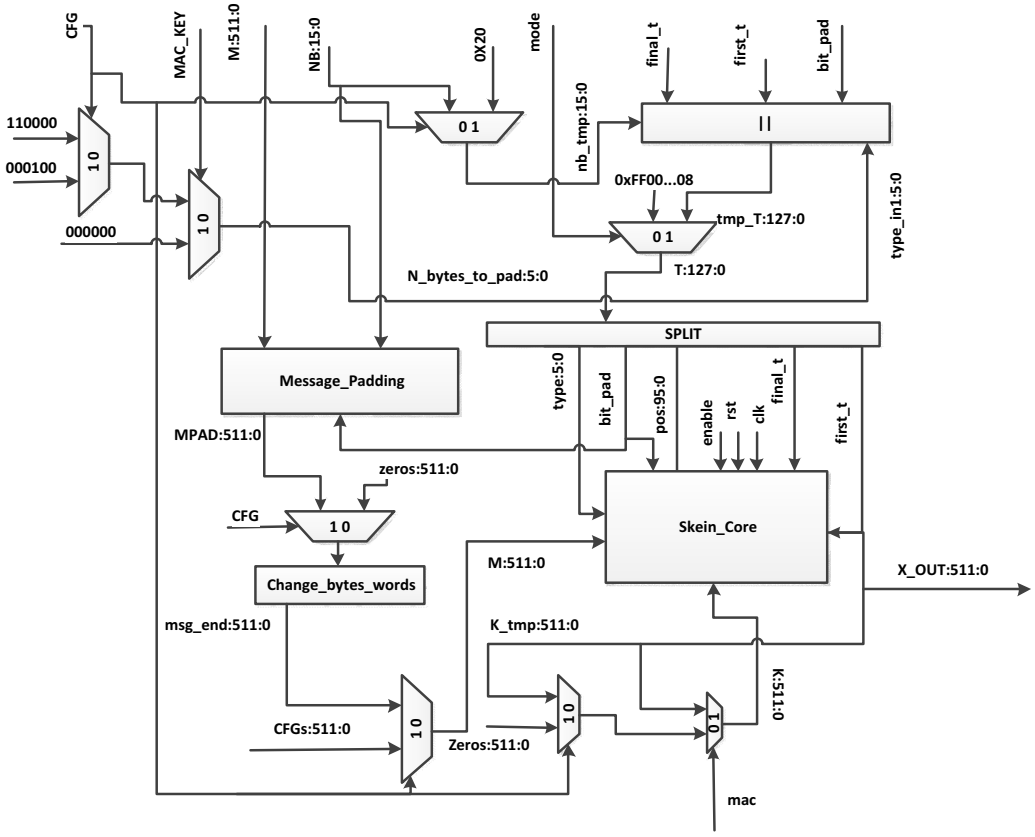

Figure 2. The proposed architecture.

Figure 3 shows the Skein_Core component. The signals *first*, *final*, *bit_pad*, *type* and *pos* inserts into tweak creator component to generate the 128 bit tweak value in the incoming block of UBI. The output of the UBI_Core is XORed with the input and after some bytes swaps is loaded into the register Reg_Load.

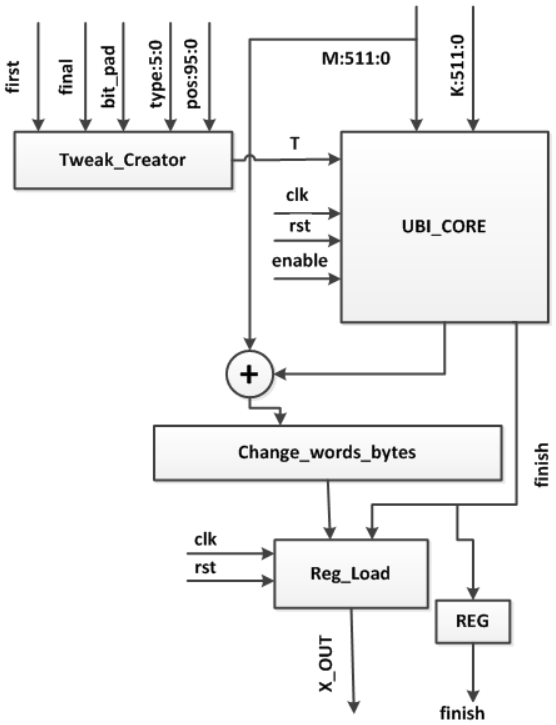

Figure 3. Architecture of the Skein core.

The UBI_Core component is depicted in Fig. 4. Mainly it consists of Threefish_core and Key_schedule. The Threefish_core has as input the 512-bit message, *M*, and the Key_schedule has as inputs the 512-bit key, *K*, and the 128-bit tweak value, *T*. The Key_schedule is responsible for the 19 subkeys needed for Threefish. Also, there are two auxiliary components, the round_counter and the proc_loop that are responsible the correct operation of the UBI.

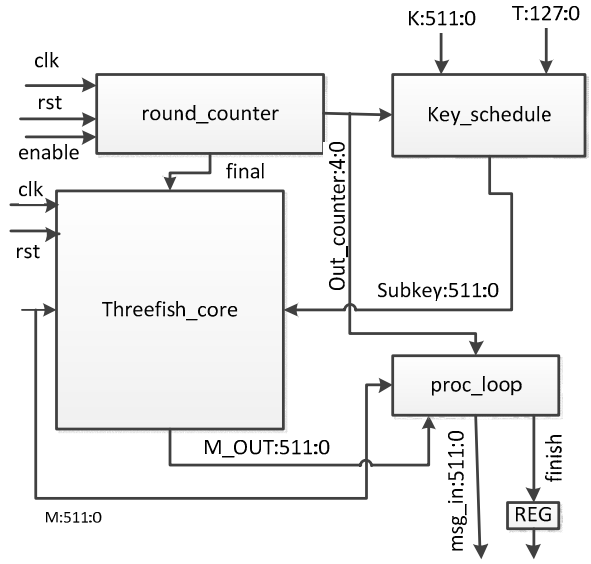

Figure 4. The UBI architecture.

Finally, the architecture of the Threefish_core is illustrated in Fig. 5. The adder_512 separates the message and the key round in eight strings of 64-bit and then adds the strings modulo64. The register_512 is used to synchronized the data with the appropriate key. The Threefish_rounds contain eight successive rounds of Threefish cipher. Finally, the multiplexer selects the appropriate result.

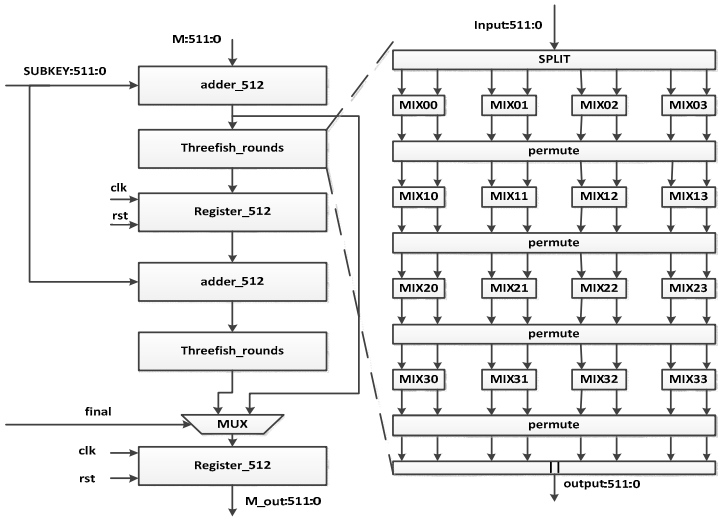

Figure 5. The Threefish_Core architecture.

IV. FPGA-BASED SYNTHESIS RESULTS

The proposed architecture was implemented by using VHDL with structural description logic. The VHDL code was simulated and verified by using the test vectors, provided by the specifications. The implementation has been synthesized with XILINX ISE tool and Virtex FPGAs were used. The synthesis results and performance analysis for the proposed architecture are depicted in Table I. The proposed implementation achieves variant throughput depend from the message length (*message_length*). The numbers of the operational clock cycles are given by the equation $clock\_cycles = 42 + 21x\frac{message\_length}{512}$. So, for a frequency of 120 MHz (Virtex – 7 device) the system's throughput in relation to the message length is given by the Fig. 6.

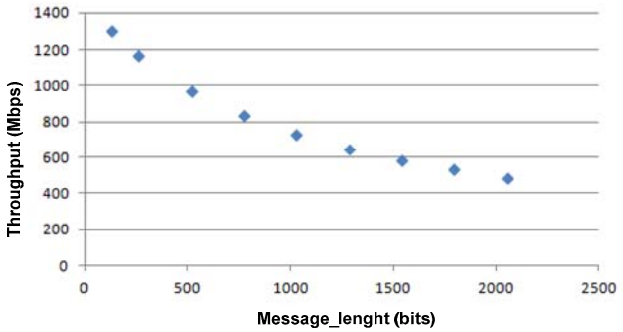

Figure 6. Throughput vs Message length.

The synthesis results are given in Table I. For the throughput a message equal to 512-bit was used.

TABLE I. SYNTHESIS RESULTS

FPGA Device	XC7VX1140TFLG930-2	XC6VLX760FF1760-2
FPGA Resources	#	#
Slice Registers	1543	1543
Slice LUTs	11625	11353
Freq (MHz)	120	110
Throughput (Bbps)	975	894

The implementation in VIRTEX-7 achieves better time performance (975 Mbps @ 120 MHz) compared to identical in VIRTEX-6 (894 Mbps @ 110 MHz). This means that the VIRTEX-7 reach a throughput/slice value equal to 0.08 versus the 0.07 for the VIRTEX-6 device. So, the VIRTEX-7 device fits better for this implementation.

Comparisons with other existing Skein implementations are given in Table II. The work in [6] uses an iterative architecture in a Virtex – 5 device that achieves a throughput up to 818 Mbps at 115 MHz.

TABLE II. COMPARISONS

Architecture	Frequency (MHz)	Throughput (Mbps)
[6]	115	818
[7]	200	223
[8]	276	80
[9]	-	1325
Proposed	120	975

In addition, in [7] a compact implementation was proposed that reach a bit rate up to 223 Mbps @ 200 MHz. Furthermore, in [8] a folded method was used for the Skein-512 architecture. It achieves a throughput equal to 80 Mbps with operational frequency of 276 MHz. The proposed implementations outperform all the previous implementations in term of time performance. Finally, in [9] a pipeline architecture was implemented with result a high speed implementation with throughput up to 1325 Mbps.

V. CONCLUSION

An efficient iterative architecture of Skein-512 hash function was design in this paper. For the implementation two FPGA devices were used (Virtex-6 and Virtex-7). The implementations achieve throughput equal to 975 Mbps (@ 120MHz) in Virtex-7 and 894 Mbps (@110 MHz) in Virtex -6. The proposed implementations outperform many of the corresponding existing ones in terms of time performance. The synthesis results prove that the architecture is a good choice for applications of high time performance requirements.

REFERENCES

[1] National Institute of Standard and Technology (NIST), "Cryptographic hash algorithm competition", 2007, available on line at http://www.nist.gov/itl/csd/ct/hash_competition.cfm

[2] Secure Hash Standard (SHS), National Institute of Standards and Technology (NIST), FIPS PUB 180-3, 2008, available on line at http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[3] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, J. Walker, "The Skein Hash Function Family", (2008). http://www.skein-hash.info/sites/default/files/skein1.1.pdf.

[4] Xilinx, Virtex-6 FPGAs, available on line at http://www.xilinx.com/products/silicon-devices/fpga/virtex-6/

[5] Xilinx, Virtex- FPGAs, available on line at http://www.xilinx.com/products/silicon-devices/fpga/virtex-7/

[6] M. Long, "Implementing Skein Hash Function on Xilinx Virtex-5 FPGA Platform", Intel Corporation Report Reports, 2009. http://www.skeinhash.info/sites/default/files/skein_fpga.pdf

[7] S. Kerckhof, F. Durvaux, N. Veyrat-Charvillon, F. Regazzoni, G. Meurice de Dormale, F. X. Standaert, "Compact FPGA Implementations of the Five SHA-3 Finalists", CARDIS 2011, Leuven, Belgium, 14-16 Sptember, pp. 217-233, 2011.

[8] N. At, J.-L. Beauchat, I. San, I, "Compact Implementation of Threefish and Skein on FPGA", IACR Cryptology ePrint Archive, 2012.

[9] E. Homsirikamol, M. Rogawski, K. Gaj, "Throughput vs. Area Trade-offs in High-Speed Architectures of Five Round 3 SHA-3 Candidates Implemented Using Xilinx and Altera FPGAs", 13th Intern. Conf. on Crypt. Hardware & Embed. Sys. (CHES 2011), Japan, 28 Sept.-1 Oct.,2011, pp. 491-506.