2011 14th Euromicro Conference on Digital System Design

Low Power FPGA Implementations of JH and Fugue Hash Functions

George Provelengios
Dept. of Telecommunication Systems & Networks
Technological Educational Institute of Mesolonghi, Greece
e-mail: georprob@ieee.org

Nikolaos S. Voros
Dept. of Telecommunication Systems & Networks
Technological Educational Institute of Mesolonghi, Greece
e-mail: voros@teimes.gr

Paris Kitsos
Computer Science
Hellenic Open University /
Dept. of Telecommunication Systems & Networks
Technological Educational Institute of Mesolonghi, Greece
e-mail: pkitsos@ieee.org

***Abstract* — Low power techniques in a FPGA implementation of the hash functions called JH and Fugue are presented in this paper. The JH hash function is under consideration for adoption as standard. Pipeline technique (with some variants) and the use of embedded RAM blocks are the techniques in order to reduce the power consumption. Power consumption reduction between 6.3% and 33 % was achieved for JH function and similar a power reduction between 1.7% and 13.3 % was achieved for Fugue by means of the proposed techniques compared with the implementation without any low power issue.**

Keywords-hash functions; low power techniques; FPGA; VIRTEX-5

I. INTRODUCTION

A cryptographic hash function H is a transformation that takes a variable-size input m and returns a fixed-size string, which is called the hash value h (that is, h = H(m)). Hash functions with just this property have a variety of general computational uses, but when employed in cryptography the hash functions are usually chosen to have some additional properties [1].

Among the available hash functions Secure Hash Algorithm-1 (SHA-1) [2] and Secure Hash Algorithm-2 (SHA-2) [3] are the most known. However, serious attacks have been reported against SHA-1 [4]. Moreover, the SHA-2 hash function belongs in the same general hash function family of SHA-1. Consequently,, both can constitute potential candidates for attacking using similar techniques.

After that the National Institute of Standard and Technology (NIST) organize a public competition to develop a new cryptographic hash function standard [5]. The new hash function will be called SHA-3 and will replace the functions in the SHA-2 family. The received hash algorithms are available for public comments in terms of security and hardware implementations efficiencies.

JH [6] and Fugue [7] hash functions have been submitted to SHA-3 competition. In this paper FPGA implementations of JH and Fugue hash functions are proposed. Also, low power techniques are adopted for both algorithms. These techniques are implemented at gate level and reduce the amount of signal glitching within the circuit. JH function has been qualified on final-five SHA-3 competition while Fugue has not been qualified. For this reason the implementation of JH is presented in details and the characteristics of Fugue implementation will be used for comparison purposes. Our major goal is to explore area-speed trade-offs in the implementations of two hash functions, to compare the efficiency of the designs by examining the throughput per FPGA slice and finally estimate the effects of the low power techniques.

The rest of the paper is structured as follows. In Section 2 techniques for low power FPGA designs are briefly described. In Section 3, the core architectures of the two hash functions are described while in Section 4 the implementations of the low power techniques on the proposed JH and Fugue architectures are presented. Synthesis results, comparisons with previously published designs and estimations in power consumption are given in Section 5. Finally, Section 6 concludes the paper.

II. LOW POWER FPGA TECHNIQUES

The power consumption P_{Total} of an FPGA is constituted by two components, namely the dynamic power, $P_{Dynamic}$ and the static power, P_{Static}. Dynamic power is dissipated when signals charge capacitive nodes. Static power, on the other hand, has nothing to do with the activity of the circuit. Total static power is the combined total of each transistor's leakage power and all bias currents in the FPGA. Dynamic power makes up the larger portion of the total amount of power consumed by an FPGA design. The gate level low-power techniques that have been applied to the hash functions designs are briefly described below.

Pipelining [8] is used in order to reduce the amount of signal glitching within the circuit. A pipelined design has less logic between registers and therefore is less prone to glitching.

Clock gating provides a means of reducing switching activity, by disabling the registers reading data [9]; so they just keep their contents when they are inactive.

In the double edge trigger (DET) pipeline [10], both rising and falling edges of the clock signal are used. A negative edge triggered Flip-Flop will hold the output wire state for the first half of a clock cycle, and when the clock

978-0-7695-4494-6/11 $26.00 © 2011 IEEE
DOI 10.1109/DSD.2011.56

signal toggles, the Flip-Flop will assume a new logic value. Power is saved because glitches are not propagated to logic circuit, simultaneously with a reduction of the circuit latency compared with the usual pipelining technique described above.

Finally, RAM blocks [11] save general logic resources because they do not carry any programmable design routing with them.

III. CORE ARCHITECTURES

The hardware implementation of the JH hash function is depicted in Fig. 1.

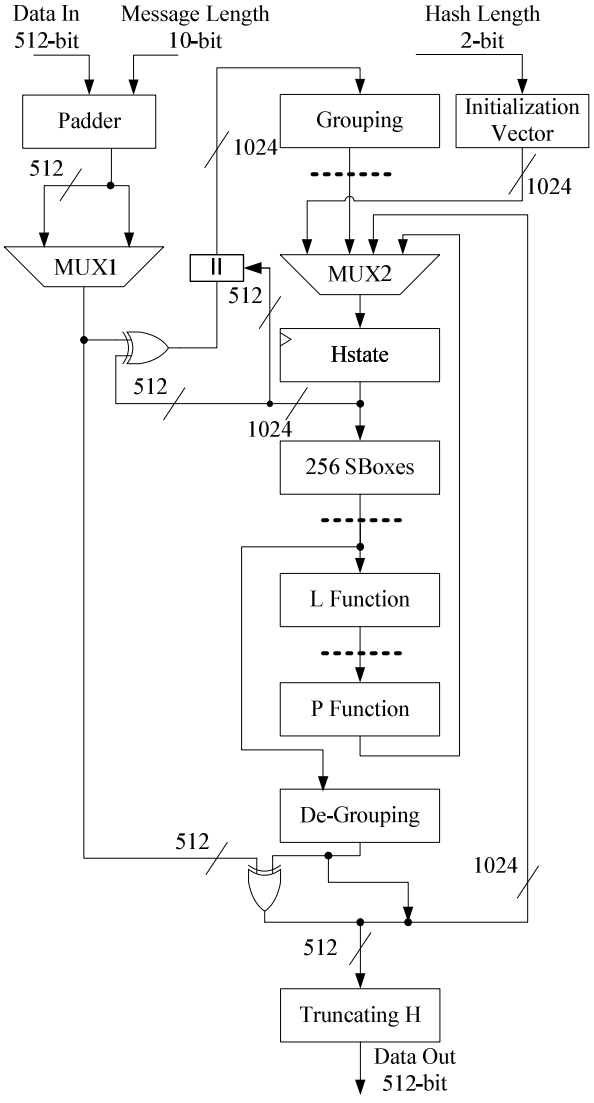

Fig. 1. Hardware implementation of the JH hash function

The Padder pads the input data and converts them to an n-bit padded message. In the proposed architecture an interface with 512-bit input for Data In is considered. The Message Length, specifies the total length of the message. The padded message is partitioned into a sequence of t 512-bit blocks. Other basic components are the Initialization Vector, the Grouping and De-Grouping, the Hstate, the SBoxes layer, the L- and P- layers and Trancating.

The initialization vector sets the initial hash value H(0) that is depending on the message digest size. The Grouping component groups the output XORed data and the output data from HState register into 4-bit elements according to the specifications. The opposite procedure is implemented from De-Grouping component. The De-Grouping component takes the data from SBoxes layer and re-assembles them in the original structure. Component Hstate is responsible for data synchronization at the begging of each new round. The SBoxes layer consists of two mini 4x4-bit s-boxes S0 and S1. Instead of being simply xored to the input, every round constant bit selects which s-boxes are used so as to increase the overall algebraic complexity. The linear transformation L implements a (4; 2; 3) Maximum Distance Separable (MDS) code over $GF(2^4)$. Here the multiplication in $GF(2^4)$ is defined as the multiplication of binary polynomials modulo the irreducible polynomial $x^4 + x + 1$. P layer is a simple permutation on some elements according to the specifications. The final component Truncating gives the output of JH circuit which is depending on Hash Length signal. This output varied in 224-bit, 256-bit, 384-bit and 512-bit blocks.

For the algorithm execution 74 clock cycles are needed and a controller is responsible for the correct operation of the algorithm.

Also, the hardware implementation of the Fugue hash function is depicted in Fig. 2a. The Padder pads the input data and converts them to an n-bit padded message. In this architecture an interface with 128-bit input for Data In is considered. The Message Length specifies the total length of the message. The padded message is partitioned into a sequence of, t, 32 blocks. Other basic components are the Initialization Vector, the TIX and SMIX, the ROR3 CMIX, the XOR ROR15 and XOR ROR14. The initialization vector sets the initial hash value of state S that depends on the message digest size.

The TIX component executes some XOR operations, truncate blocks and inserts some bits in their input. The CMIX executes column mix, while ROR14 and ROR15 executes rotation of the state S to the right by 14 and 15 columns respectively. Finally, the SMIX transformation takes a 4 × 4 matrix of bytes, passes each byte through the S-box layer (16 SBoxes in total), and then applies a linear transformation to the result. This transformation is called the Super Mix transformation. Fig. 2b also shows the implementation of Super Mix transformation. Actually, a matrix multiplication is executed between the outputs of the SBoxes and the matrix N (16x16) that is specified at the function specifications.

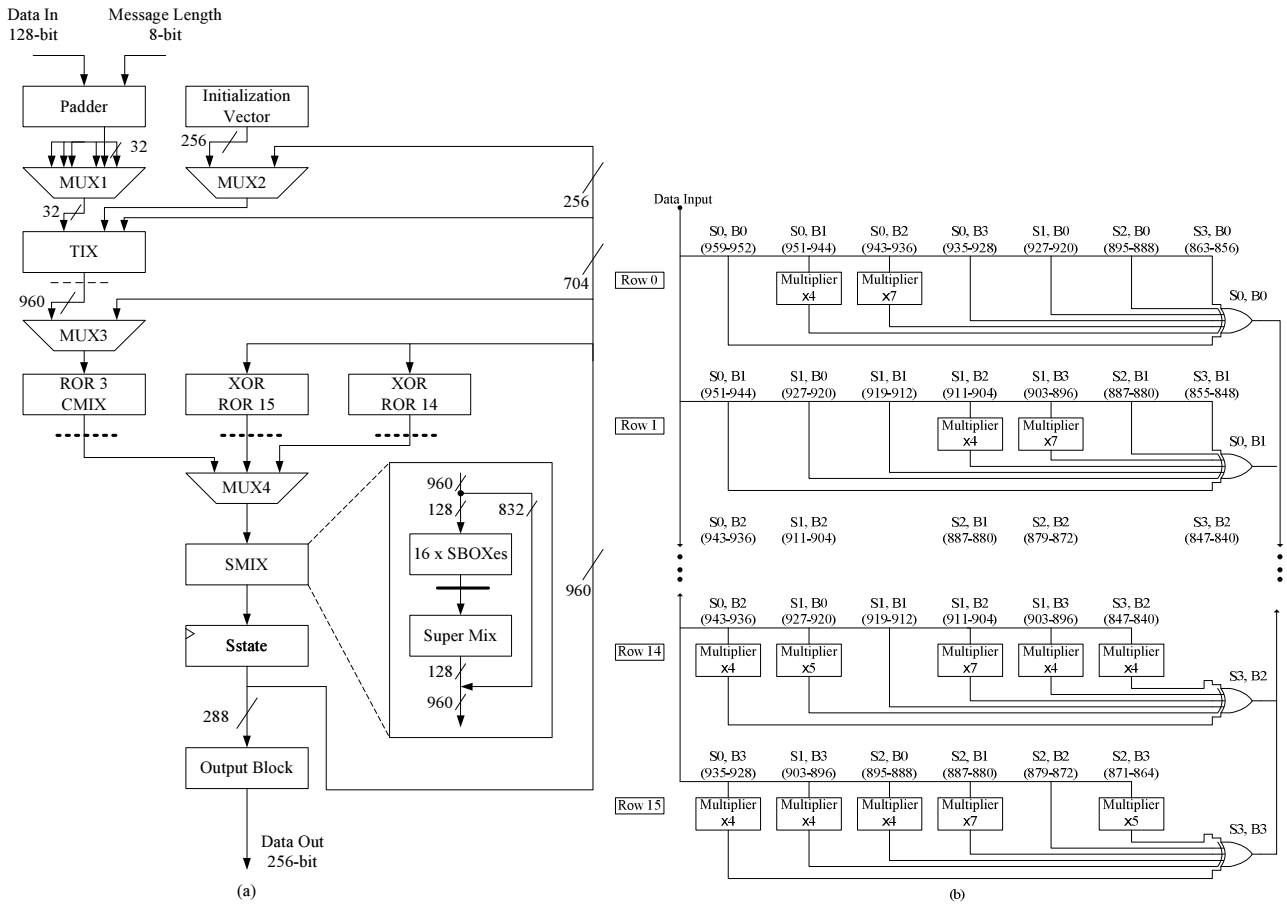

Fig. 2. a) Hardware implementation of the Fugue hash function, b) Hardware implementation of the Super Mix

According to the Fig. 2b some multipliers (x4, x5, x6 and x7) on field GF(2^8) are needed. These multipliers are simply implemented by shift registers and XOR gates.

For the algorithm execution 49 clock cycles are needed and a controller is responsible for the correct operation of the algorithm.

IV. Implementations Using Power Reduction Techniques

In this section, the gate level low-power design techniques that have been applied to the JH and Fugue architectures are described.

The first pipeline technique with only positive edge Flip-Flops is symbolized as JH_positive, the pipeline technique with the use of gating clock is symbolized as JH_gating and finally the pipeline technique with positive and negative edge Flip-Flops is symbolized as JH_negative. The technique with RAM blocks is symbolized as JH_RAM.

In the JH_positive technique the three pipeline registers are placed between the Grouping and MUX2, SBoxes and L Function and finally between the L Function and P Function as the Fig. 1 shows (discontinuous lines). The registers are single edge triggered and the data are transferred between two successive registers in one clock period. So the latency of the circuit is increased up to 218 clock cycles.

In the second pipeline scheme (JH_gating), the same places for the pipeline registers are used and each register is implemented by means of the Clock Enable.

Finally, in the third pipeline scheme (JH_negative) an inner pipeline with negative edge triggered register (Flip-Flops) is used. The negative edge triggered register replaces the usual pipeline registers (see Fig. 1). The usage of this technique leads to two very important results. Firstly, the glitches are not propagated to logic circuit and secondly the execution time of algorithm is reduced compared with the JH_positive scheme. So, 148 clock cycles are needed for the algorithm execution.

With regard to embedded RAM blocks (JH_RAM technique) there are three places in JH which could easily be used; firstly for the implementation of SBoxes, secondly for the implementation of the constant values are used in process of substation and thirdly for the implementation of the initialization vector. There are 256 SBoxes (S0, S1) in total, so 256 embedded RAM blocks with 32 positions of 4-bit each are used. Also, there are 37 constants values, which results 37 embedded RAM blocks with 64 positions of 4-bit

each position. Finally, in the initialization vector there are 4 vectors of 1024-bit, so 4 embedded RAM blocks with 256 positions of 4-bit are used.

The same techniques are used in order to reduce the power dissipation of the Fugue hash function. Similar to JH, in the Fugue_positive four pipeline registers were used as the Fig. 3a shows (discontinuous lines). The registers are single edge triggered and the data are transferred between two successive registers in one clock period. So the latency of the circuit is increased up to 103 clock cycles.

In the second pipeline scheme (Fugue_gating), the same places for the pipeline registers are used and each register is implemented by means of the Clock Enable.

Finally, in the third pipeline scheme (Fugue_negative) an inner pipeline with negative edge triggered register (Flip-Flops) is used. The negative edge triggered register inserted inside the SMIX component as it is illustrated in Fig. 2a.

V. EXPERIMENTAL RESULTS

The proposed implementations were captured by using VHDL. The VHDL code has been synthesized using XILINX ISE 10.1 tool and the target FPGA device was XC5VLX330-2FF1760. The total power dissipation is measured using XILINX XPOWER analyzer tool [12]. The synthesis results, performance analysis and power dissipation for the JH and Fugue architectures described in section 3 (denoted as JH_conventional and Fugue_conventional) are depicted in Table I. Comparisons to previously published JH and Fugue implementations are also given. The proposed implementations (JH_conventional and Fugue_conventional) achieve a throughput equal to 1.3 Gbps for a frequency of 201.2 MHz and 514 Mbps for a frequency of 103.2 MHz respectively.

TABLE I. RESULTS AND COMPARISONS

Archi-tecture	Techno-logy	# FFs	# Slice	Freq (MHz)	Bit rate (Mbps)
JH_ conv	XC5VLX330 -2FF1760		2251	201.2	1328
[13]	0.18 µm	58832 GE*		380.22	4.992
[14]	0.18 µm	51212		259.54	3.407
[15]	Virtex 5	-	1108	278.09	3955
[16]	XC5VLX330 -2FF1738	-	1763	144.11	1941
Fugue_ conv	XC5VLX330 -2FF1760	969	1394	103,2	514
[13]	0.18 µm	46257 GE*		255.75	4.092
[14]	0.18 µm	48401 GE*		161.19	2.579
[15]	Virtex 5	-	956	98.47	3151
[16]	XC5VLX330 -2FF1738	-	2046	200	914

* The *GEs* (*Gate Equivalent*) is the area metric for ASIC designs and is equal to the area of two-input NAND gate.

The JH implementations in [13-14] use 0.18 µm library for their implementations and achieve throughput up to 4.9 and 3.4 Gbps respectively. In [15] a Virtex 5 FPGA implementation is presented and achieves throughput up to 3.9 Gbps for a frequency of 278.09 MHz. Finally, in [16] a Virtex XC5VLX330-2FF1738 FPGA is used and achieves similar time and area performance to the proposed one. For all previous implementations [13-16] there are no estimations for power dissipation.

Similarly, the Fugue implementations in [13-14] use 0.18 µm library for their implementations and achieve throughput up to 4 and 2.6 Gbps respectively. In [15] a Virtex 5 FPGA implementation is presented and achieves throughput up to 3.1 Gbps for a frequency of 98.47 MHz. Finally, in [16] a Virtex XC5VLX330-2FF1738 FPGA is used and achieves similar time and area performance to the proposed one. Also, for all previous Fugue implementations [13-16] there are no estimations for power dissipation.

In Table II the estimations in term of power consumption are given. Each estimation corresponds to one of the power reduction techniques described in section 4. The estimations are produced with XPOWER tool. The power dissipation for the conventional architectures (described in section 3) is also given in Table II.

TABLE II. POWER ESTIMATIONS

Power Reduction Technique	Architecture	Power (mW)
Positive Pipeline	JH_positive	10726
Pipeline and Gating Clock	JH_gating	7685
Negative Pipeline	JH_negative	10672
RAM Blocks	JH_RAM	12262
Conventional	JH_conventional	11451
Positive Pipeline	Fugue_positive	4372
Pipeline and Gating Clock	Fugue_gating	4037
Negative Pipeline	Fugue_negative	4240
RAM Blocks	Fugue _RAM	3854
Conventional	Fugue _conventional	4446

It can be seen that the pipeline technique with positive edge registers achieves an improvement in terms of power consumption up to 6.3 % for JH function and 1.7 % for Fugue function. Also, the implementation with the positive edge pipeline technique with Clock Enable consumes less power by 33 % for JH and 9.2 % for Fugue. In addition, by using the pipeline technique with negative edge registers bit better performance is achieved in terms of power, i.e. power consumption is reduced down to 6,8 % and 4.6 % respectively. Finally, the results for the implementations with the use of RAM blocks are bit strange. While, in the Fugue implementation achieves a power reduction up to 13.3 % in the JH implementation had contradictious results. The power dissipation increased by a factor equal to 1.07. This is a physical result because this implementation uses many RAM blocks for both SBoxes and initialization vector which results a higher level of power.

VI. CONCLUSIONS

Efficient techniques for reducing the power consumption of the JH and Fugue hash functions FPGA implementations have been presented in this paper. Different versions of the pipeline technique and the use of RAM blocks were introduced. With these techniques a power reduction between 6.3% and 33 % was achieved for JH function and similar a power reduction between 1.7% and 13.3 % was achieved for Fugue compared to existing implementations.

REFERENCES

[1] S. Bakhtiari, R. Safavi-Naini, J. Pieprzyk, "Cryptographic Hash Functions: A Survey", Technical Report 95-09, Department of Computer Science, University of Wollongong, July 1995.

[2] SHA-1 Standard, National Institute of Standards and Technology (NIST), Secure Hash Standard, FIPS PUB 180-1, 1995, available on line at www.itl.nist.gov/fipspubs/ fip180-1.htm

[3] 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

[4] X. Wang, Y. Lisa Yin, and H. Yu, "Finding Collisions in the Full SHA-1," Proc. 25th Annual Int. Cryptology Conference- CRYPTO 2005, Santa Barbara, California, USA, August 14-18, 2005, LNCS 3621, 2005.

[5] 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

[6] Hongjun Wu, "The Hash Function JH", The First SHA-3 Candidate Conference, 2009, available on line at http://ehash.iaik.tugraz.at/wiki/JH

[7[S. Halevi, W. E. Hall and C. S. Jutla, "The Hash Function Fugue", The First SHA-3 Candidate Conference, 2009, available on line at http://ehash.iaik.tugraz.at/wiki/Fugue

[8] G. Sutter, E. Boemo, "Experiments in Low Power Design", Special Issue on Configurable Logic of Latin American Applied Research (LAAR), pp 99-104, Vol. 37, No. 1, Jan. 2007.

[9] Y. Zhang, J. Roivainen, and A. Mammela. "Clock-Gating in FPGAs: A Novel and Comparative Evaluation", EUROMICRO Conf. on Digital System Design: Architectures, Methods and Tools, pages 584–590, 2006.

[10] T. Czajkowski and S. D. Brown, "Using Negative Edge Triggered FFs to Reduce Glitching Power in FPGA Circuits", 44th Design Automation Conf., San Diego, California, June 4-8, 2007, pp. 324-329.

[11] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs", ACM Symposium on FPGAs, Feb. 2006, pp. 21-30.

[12] XILINX XPOWER analyzer tool, available on line at http://www.xilinx.com/products/design_tools/logic_design/verification/xpower_an.htm

[13] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J. – Marc Schmidt, and A. Szekely, "High-Speed Hardware Implementations of BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grostl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, and Skein", Cryptology ePrint Archive, Report 2009/510, 2009. http://eprint.iacr.org/

[14] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J.-Marc Schmidt, A. Szekely, "Uniform Evaluation of Hardware Implementations of the Round-Two SHA-3 Candidates", The Second SHA-3 Candidate Conference, University of California, Santa Barbara, August 23-24, 2010.

[15] K. Gaj, E. Homsirikamol, and M. Rogawski, "Comprehensive Comparison of Hardware Performance of Fourteen Round 2 SHA-3 Candidates with 512-bit Outputs Using Field Programmable Gate Arrays", The Second SHA-3 Candidate Conference, University of California, Santa Barbara, August 23-24, 2010.

[16] B. Baldwin, N. Hanley, M. Hamilton, L. Lu, A. Byrne, M. O'Neill and W. P. Marnane, "FPGA Implementations of the Round Two SHA-3 Candidates", 20th International Conference on Field Programmable Logic and Applications (FPL), Milano, ITALY, Aug. 31st - Sep. 2nd, 2010.