Low Power FPGA Implementations of 256-bit Luffa Hash Function

Paris Kitsos
Computer Science
Hellenic Open University, Greece
E-mail: pkitsos@ieee.org

Nicolas Sklavos
Informatics & MM Department,
Technological Educational, Institute of Patras, Greece
E-mail: nsklavos@ieee.org

Athanassios N. Skodras
Computer Science
Hellenic Open University, Greece
E-mail: skodras@ieee.org

Abstract

Low power techniques in a FPGA implementation of the hash function called Luffa are presented in this paper. This hash function is under consideration for adoption as standard. Two major gate level techniques are introduced in order to reduce the power consumption, namely the pipeline technique (with some variants) and the use of embedded RAM blocks instead of general purpose logic elements. Power consumption reduction from 1.2 to 8.7 times is achieved by means of the proposed techniques compared with the implementation without any low power issue.

1. Introduction

The most known hash function is the Secure Hash Algorithm-1 (SHA-1) [1]. In recent years serious attacks have been published against SHA-1 [2]. After that, the transition to the stronger SHA-2 [3] family of hash functions was decided. The SHA-2 hash functions are included in the same general family of hash functions as SHA-1. So, they could possibly be attacked with similar techniques. This led the National Institute of Standard and Technology (NIST) to organize an effort to develop more secure hash algorithms through a hash function competition (SHA-3) for usage in the future.

Luffa hash function [4] has been submitted and has been under consideration to SHA-3 competition.

Field Programmable Gate Arrays (FPGAs) are being used increasingly in embedded general purpose computing environments as performance accelerators.

In this paper low power techniques for an FPGA Luffa implementation are proposed. These techniques are implemented in gate level and reduce the amount of signal glitching within the circuit. Recently, some Luffa hardware designs have been proposed [5-7].

This paper is structured as follows. In Section 2 techniques for low power FPGA designs are introduced. In Section 3, the core architecture of the Luffa hash function is described while in section 4 the implementations of the low power techniques on the proposed Luffa architecture 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.

2. Designing for low power FPGA

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. The dynamic (switching) power is dependent on the supply voltage V and operating frequency f and equals to:

$$P_{Dynamic} = \frac{1}{2} C V^2 f \qquad (1)$$

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. The static power is equal to:

$$P_{Static} = V I_{Leakage} \qquad (2)$$

Dynamic power makes up the larger portion of the total amount of power consumed by an FPGA design. The gate level low-power design techniques that have been applied to current FPGA technology, in order to reduce the dynamic power, are 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. Also, with less logic between registers, the amount of interconnect between registers is also reduced.

Clock gating is disabling the clock for the inactive regions in a design to minimize signal transitions and hence dynamic power [9]. Clock gating provides a means of reducing switching activity, by disabling the registers reading external data; so they just keep their contents when they are inactive.

Both *positive and negative edge Flip-Flops* are also used. The double edge trigger (DET) pipeline is actually employed [10-11]. In a DET pipeline, 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 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* instead of general purpose logic elements can reduce the dynamic power consumption [12]. The RAM blocks save general logic resources because they do not carry any programmable design routing with them.

3. Luffa core architecture

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

The *Padder* pads the input data and converts them to an *n*-bit padded message. In the proposed architecture an interface with 256-bit input for *Message* is considered. The input *n*, specifies the total length of the message. The padded message is partitioned into a sequence of t 256-bit blocks m^1, m^2, … , m^t.

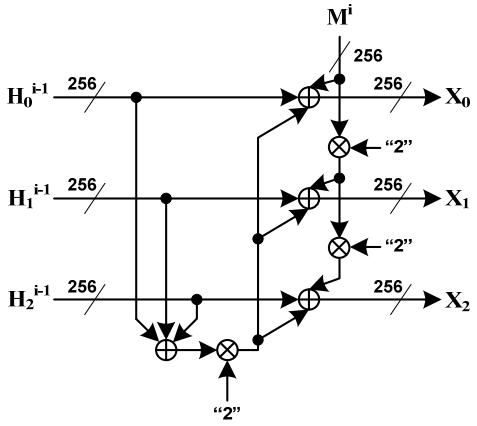

Fig. 2. Implementation of the message injection function

The multiplexer, *MUX*, selects either the fix message equal to 256-bit zeros or the message m^i. The *MUX* output, M^i is then used in order to generate a new sequence of 256-bit string, H_1, H_2, … , H_t in the following way. M^i and H^{i-1} are processed as inputs to the *MI* and the resulting 256-bit substrings are XORed in order to produce the hash value (Z). The outputs of the Luffa Round are concatenated and used as the H^{i-1} input to the Luffa Round. The Luffa Round mainly consists of the *MI* function and the permutation *P*. The implementation of the Luffa Round is also illustrated in Fig. 1.

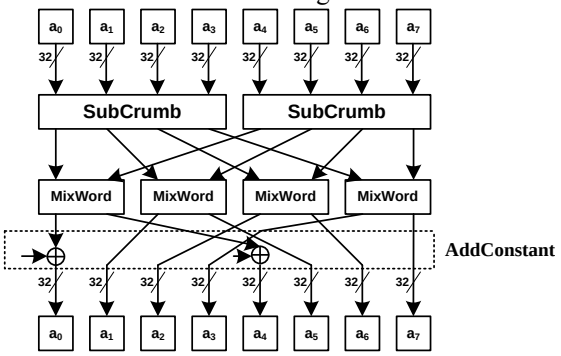

Fig. 3. Implementation of the *step* function

The implementation of the *MI* function is shown in Fig. 2. It consists of four 3-input logical XOR gates of 256-bit each one and three multipliers. Inside the Q_1 and Q_2 permutations there is a *tweak* permutation that is implemented by using combinational shifters. The *step* function consists of the three functions, *SubCrumb*, *MixWord* and *AddConstant*. At the beginning the 256-bit data are stored in eight 32-bit registers a_k as shown in Fig. 3. The *SubCrumb* function is composed of 32 substitution tables (*S-boxes*). The *MixWord* function is a linear permutation of two 32-bit words. The implementation of this function is depicted in Fig. 4. This module consists of four 2-input XOR gates and four rotators.

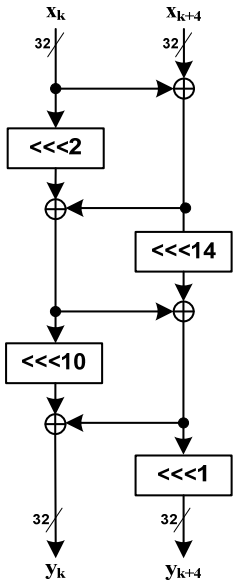

Fig. 4. Implementation of MixWord

4. Implementations using power reduction techniques

In this section, the gate level low-power design techniques that have been applied to the Luffa architecture are described.

The first pipeline technique with only positive edge Flip-Flops is symbolized as Luffa_positive, the pipeline technique with the use of gating clock is symbolized as Luffa_gating and finally the pipeline technique with positive and negative edge Flip-Flops is symbolized as Luffa_negative. The technique with RAM blocks is symbolized as Luffa_RAM.

In the Luffa_positive technique the pipeline registers are placed between the components of the Luffa round. Registers are placed after the *MI* functions and between the *tweak* and *steps* inside the Q_i (i=1, 2, 3) permutations as shown in Fig. 5. 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 10 clock cycles.

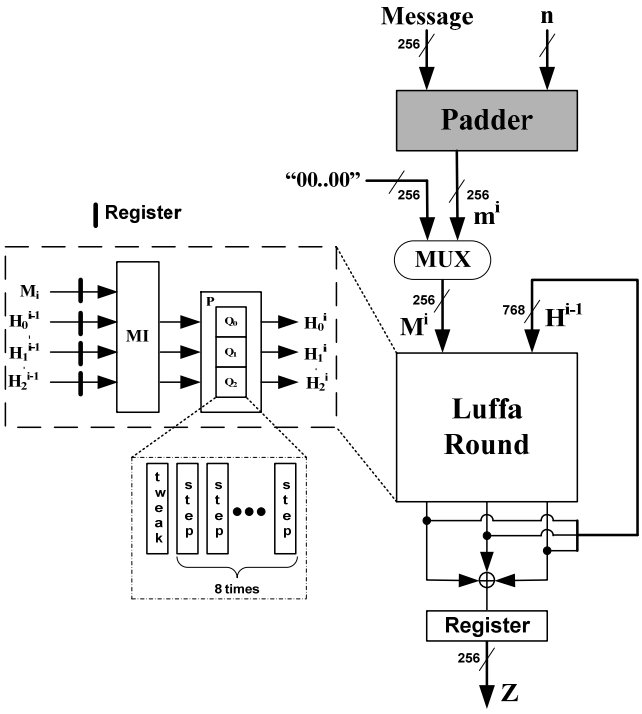

Fig. 1. Hardware implementation of the Luffa hash function

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

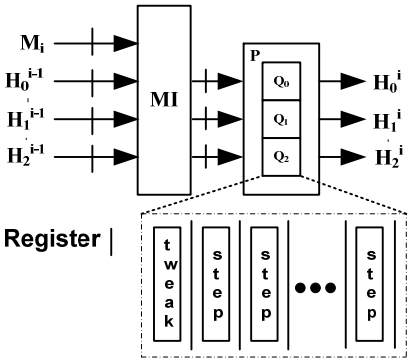

Fig. 5. Pipeline registers in Luffa_positive technique

Finally, in the third pipeline scheme (Luffa_negative) an inner pipeline with negative edge triggered register (Flip-Flops) is used as shown is Fig. 6. The negative edge triggered register is inserted in the *Qi* (*i=0*, *1*, *2*) permutations between in the third and forth *step* functions, which is roughly in the middle of the round data path. 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 each round is one system clock cycle, so the circuit latency is not increased compared with the architecture described in section 4. With regard to embedded RAM blocks (Luffa_RAM technique) there is a place in Luffa which could easily be used for the implementation of S-boxes inside SubCrumb function.

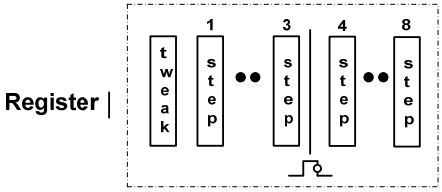

Fig. 6. Pipeline register in Luffa_negative technique

Each SubCrumb function consists of 32 S-boxes, so 32 embedded RAM blocks with 16 positions of 4-bit each, are used.

5. 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 XC5VLX50-3FF1153. The total power dissipation is measured using XILINX XPOWER analyzer tool [13]. The synthesis results, performance analysis and power dissipation for the Luffa architecture described in section 4 (denoted as Luffa_conventional) are shown in Table 1.

Comparisons to previously published Luffa implementations are also given. The proposed implementation (Luffa_conventional) achieves a throughput equal to 12.2 Gbps for a frequency of 63.5 MHz. For the throughput estimation a padded message equal to 768-bits is used.

Table 1. Experimental results and comparisons

Archi-tecture	Techno-logy	# FFs	# Slice LUTs	Freq (MHz)	Bit rate (Gbps)
Propo-sed	XC5VLX 50	2304	15749	63.5	12.2
[5]	0.18 µm	44972 GEs *		483	13.7
[6]	0.13 µm	11484 GEs *		250	0.32
[7]	Stratix III	3247	16552	47	12

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

The Luffa implementations in [5-6] use 0.18 µm and 0.13 µm libraries for their implementations and achieve throughput up to 13.7 and 0.32 Gbps respectively. In [7] a Stratix III FPGA was used and achieves similar time and area performance to the proposed one. For all previous implementations [5-7] there are no estimations for power dissipation.

Table 2. Power estimations

Power Reduction Technique	Luffa Architecture	Power (mW)
Positive Pipeline	Luffa_positive	1.1
Pipeline and Gating Clock	Luffa_gating	0.9
Negative Pipeline	Luffa_negative	6.3
RAM Blocks	Luffa_RAM	1.2
Conventional	Luffa_conventional	7.8

In Table 2 the estimations in term of power consumption are given. Each estimation corresponds to one of the power reduction techniques described in section 5. The estimations are produced by XPOWER tool. The power dissipation for the conventional architecture (described in section 4) is given in Table 2.

It can be seen that the pipeline technique with positive edge registers achieves a major improvement in terms of power consumption up to 7.1 times compared to the conventional implementation. Also, the positive edge pipeline technique with gating clock achieves a higher improvement of up to 8.7 times. In addition, Luffa implementation with the use of RAM blocks for the S-boxes implementation, consumes less power by 6.5 times. Finally, by using the pipeline technique with negative edge registers better performance is achieved in terms of power, i.e. power consumption is reduced down to 24%.

6. Conclusions

Efficient techniques for reducing the power consumption of the Luffa hash function FPGA implementation have been presented in this paper. The Luffa hash function is under consideration from the NIST for the SHA-3 competition project. Different versions of the pipeline technique and the use of RAM blocks were introduced. With these techniques a power reduction between 24% and 766 % was achieved compared to existing implementations.

References

[1] 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

[2] 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.

[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] C. D. Canniµere, H.Sato, and D. Watanabe, "Hash Function Luffa". The First SHA-3 Candidate Conference, 2009, available on line at http://ehash.iaik.tugraz.at/wiki/Luffa

[5] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J.-M. Schmidt, 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*", 2009, Cryptology ePrint Archive: Report 2009/510, available on line at http://eprint.iacr.org/2009/510

[6] M. Kneževi´c and I. Verbauwhede, "*Hardware Evaluation of the Luffa Hash Family*", Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems, Proc. 4th Workshop on Embedded Systems Security, Grenoble, France, 2009.

[7] A. H. Namin and M. A. Hasan, "*Implementation of the Compression Function for Selected SHA-3 Candidates on FPGA*", Report, available on line at http://comsec.uwaterloo.ca/seminarfiles/ReviewSeminar2010/Implementation_SHA3_Candidates_on_FPGA.pdf

[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] P. Kitsos, M. D. Galanis, and O. Koufopavlou, "*Architectures and FPGA Implementations of the 64-bit MISTY1 Block Cipher*", World Scientific Journal of Circuits, Systems, and Computers (JCSC), Vol. 15, No. 6, Dec. 2006.

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

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