Hardware Implementation of A SHA-3 Application-Specific Instruction Set Processor

Mahmoud A. Elmohr, Mostafa A. Saleh, Ahmed S. Eissa,
Khaled E. Ahmed, Mohammed M. Farag
Electrical Engineering Department, Faculty of Engineering, Alexandria University,
Alexandria, Egypt
mahmoud.a.elmohr@ieee.org, moustafa.i.saleh@gmail.com, eissa.s.ahmed@gmail.com,
k.e.elsayed@ieee.org, mmorsy@alexu.edu.eg

***Abstract*—Secure Hash Algorithm 3 (SHA-3) based on the KECCAK algorithm is the new standard cryptographic hash function announced by the National Institute of Standards and Technology (NIST). Hash functions are a ubiquitous computing tool that is commonly used in security, authentication, and many other applications. The calculation of SHA-3 is very computational-intensive limiting its applicability on RISC processors used in modern embedded systems and Systems on Chips (SoCs). In this work, we study the SHA-3 computation bottlenecks on a 32-bit RISC processor and introduce two Application Specific Instruction Set Processor (ASIP) architectures to speedup SHA-3 computation on the 32-bit MIPS processor. Two ASIP architectures namely native datapath and coprocessor-based ASIPs are developed with the aid of Codasip Studio, implemented and evaluated on a Xilinx Virtex-6 FPGA. Compared to the reference SHA-3 execution on MIPS, the evaluation results show a 25% and 61.4% speedup for the native and coprocessor-based ASIPs at the expense of a 8.6% and 25.8% resource overheads, respectively.**

***Index Terms*—SHA-3, Instruction Set Extension, Application-Specific Instruction-set Processor, SoC, MIPS, FPGA**

I. INTRODUCTION

The National Institute of Standards and Technology (NIST) released the new standard for the Secure Hash Algorithm (SHA-3) in August 2015 [1] based on the KECCAK function developed by G. Bertoni *et al.* [2]. The KECCAK function consists of five steps permutations $(\theta, \rho, \pi, \chi, \iota)$ repeated for a certain number of rounds (typically 24 for SHA-3), these permutations operate on a fixed length array (typically 1600 bits for SHA-3) called state matrix and produce an output of the same length. Each permutation consists of the iteration of a simple round function limited to bit-wise XOR, AND and NOT and rotations as shown in Figure 1. The state could be represented as a 3D array of $5 \times 5 \times 64$ bit elements, a 1600-bit block fills the array in 25 lanes of 64-bit size each [1].

The execution of the SHA-3 function is computationally-intensive on lightweight embedded RISC processors because of the large size of the SHA-3 state [3]. As reported by [4], the performance of SHA-3 can be improved by utilizing vector rotate instructions. Unfortunately, low-cost RISC instruction set architectures (ISAs) do not include such instructions.

An application-specific instruction set architecture processor (ASIP) can be developed to improve the SHA-3 computation performance on a specific RISC ISA. ASIPs are custom developed processors that are optimized for special applications,

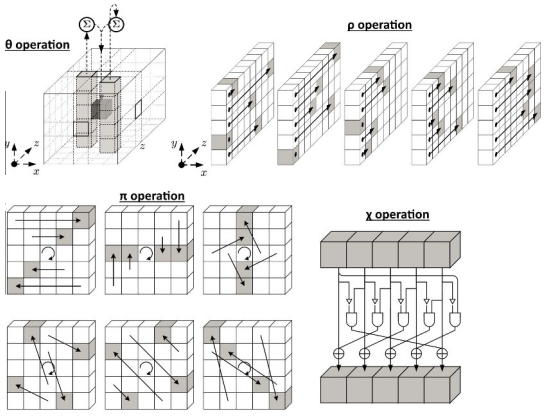

Fig. 1. KECCAK step permutations [1]

unlike general-purpose processors. The ASIP design methodology is commonly adopted in embedded system and Systems-on-Chip (SoC) design communities and its usage is rapidly growing. ASIPs provide a trade-off between the flexibility of general-purpose processors and the performance of ASICs. One of the common approaches in ASIP design is to extend a specific ISA with custom instructions, namely Instruction Set Extension (ISE), optimized for a specific set of functions. The ASIP ISE is tailored to meet the requirements of a specific application by augmenting the processor's microarchitecture with optimized hardware to perform the custom-developed instructions faster and more efficient.

The primary focus of this work is designing a SHA-3 ISE to improve the SHA-3 computation performance, specifically, on lightweight RISC processor architectures lacking for vector rotate instructions; the MIPS 32-bit ISA is selected to resemble such architectures. MIPS is a RISC processor architecture that is widely used in industry and taught in many educational institutes all over the world [5]. Recent MIPS versions such as those embedded in PIC microcontollers are widely deployed in contemporary electronic products and embedded systems.

In [6], various ISEs have been implemented for a 16-bit PIC24 microprocessor architecture to improve the performance of the five SHA-3 candidates. Another SHA-3 FPGA-based ISE on a 32-bit LEON3 processor is presented in [7]. In this work, we advance two ISEs to speedup the SHA-3 computation on a lightweight 32-bit MIPS processor [5] with the aid of Codasip Studio, an automated ASIP development

environment [8]. The rest of this paper is organized as follows: The adopted ASIP design flow is presented in Section II. SHA-3 computation bottlenecks on 32-bit MIPS ISA are introduced in Section III. The architectural and hardware design of the proposed SHA-3 ASIPs are advanced in Section IV. The performance and implementation results of the SHA-3 computation on the reference MIPS and ASIP architectures are provided in Section V. Conclusions are portrayed in Section VI.

II. Design Flow of The SHA-3 ASIP

The ASIP design flow adopted in this work is illustrated in Figure 2. Some design procedures are conducted using Codasip Studio and others are done manually to optimize the ASIP design process. The design flow procedures are:

1) Developing the Instruction Accurate (IA) and Cycle Accurate (CA) models of the lightweight MIPS processor using Codasip Studio. The IA model provides a detailed instruction set description. The CA model is a detailed description of the microarchitecture of the processor design. Both models are used to generate the CA and IA assemblers and simulators which are used to test and verify the design. The CA model also generates a synthesizable HDL code of the processor design.
2) Software implementation of SHA-3 on the 32-bit MIPS. This software design is tested using MARS simulator [9], and then tested and verified on both the Codasip IA and CA simulators. The results are validated against a SHA-3 reference code written in C language [10].
3) Multiple iterations of the profiling and software optimization are carried out to find computation bottlenecks and improve the performance of the software implementation until no further improvements could be reached in software.
4) Custom instructions are designed at the ISA level to improve the SHA-3 computation speed and incorporated into the reference MIPS microarchitecture by two approaches: the native datapath ISE and the coprocessor-based ASIP. In both approaches, the IA and CA models are implemented, tested, and verified using the first three procedures.
5) Register transfer level (RTL) Verilog HDL is generated for both the reference MIPS and the two SHA-3 extended ASIP processors using Codasip Studio. The hardware designs are simulated and verified using the Codasip-generated HDL testbenches on the Xilinx ISim simulator.
6) The RTL Verilog code of all designs is then synthesized for a Xilinx Virtex-6-XC6VLX75t-2-FF484 FPGA using Xilinx ISE. The Xilinx ISE tool is also used to report back the FPGA utilization as a hardware footprint indication.
7) The SHA-3 ASIPs are compared to the MIPS reference design in terms of the SHA-3 computation speed, total processor area, and SHA-3 program code size metrics.

III. Computation Bottlenecks of SHA-3 on MIPS

KECCAK is mainly composed of a number of step permutations which introduce the computation bottlenecks presented in this section. As the KECCAK permutations are repeated in nested loops, a minor improvement in one step can result in a major performance enhancement of the whole algorithm. Most of the bottlenecks discussed in this section are repeated 50 times (number of words in the state matrix) and each step is repeated 24 rounds. The main bottlenecks are listed below:

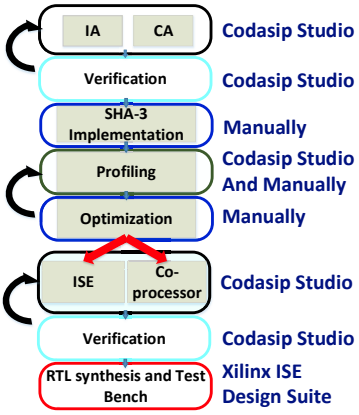

Fig. 2. The adopted ASIP design flow

1) θ **computation bottlenecks:** The θ permutation is a linear transformation applied to the state, in which each bit is XORed with the XOR result of the two surrounding columns. To implement θ each adjacent five lanes in a sheet should be XORed which requires loading them from the memory (loading two words for each lane), XORing them in a sequential manner, and storing the result back to the memory. Rotating each lane by one bit is needed to implement θ, this rotation is not straightforward because MIPS uses 32-bit register size while SHA-3 lanes are 64 bits. Therefore, multiple instructions are needed to implement this function leading to increase the computation time.
2) ρ **computation bottlenecks:** The ρ permutation rotates each lane by a variable offset, which is more complex than the previous 64-bit rotation by one bit because the rotation value is not constant and may exceed 32 bits which increases the number of cycles required to implement a generic function for 64-bit rotation.
3) π **computation bottlenecks:** The π step loads the ρ step output state from the memory, scrambles this state by rearranging lane positions, and stores it back to the memory again. The computation of the new positions requires multiple instructions containing multiplication and modulus. This bottelneck was solved in software by precalculating the positions at the compile-time.
4) χ **computation bottlenecks:** The χ permutation is a nonlinear transformation working on the lanes of the state. It requires loading three subsequent lanes; applying several NOT, AND and XOR operations on them in a sequential manner, and finally storing the result in memory.
5) ι **computation bottlenecks:** The ι permutation XORs the lane $A[0,0]$ with a variable round constant which is changed every round. Calculating the variable round constant is a significant overhead that requires many cycles. This bottleneck is solved in software by precalculating the positions at the compile-time.

IV. ISE and Hardware Design of The SHA-3 ASIP

A custom implementation of the 32-bit MIPS ISA is developed where a subset of the MIPS instructions are implemented with the aid of Codasip Studio, both the C-language compiler and the HDL design are generated and tested. Afterwards, two ISEs are advanced to resolve the SHA-3 bottlenecks presented in the last section: native datapath and coprocessor-based ISEs. Design of the ISE at the architecture level and their support at the hardware mircoarchitecture level are presented herein.

A. SHA-3 Native Datapath ASIP

In this approach, the SHA-3 bottlenecks are resolved by minor microarchitecture modifications to the MIPS processor native datapath. The syntax of the added instructions and their high-level descriptions are depicted in Table I.

TABLE I
NATIVE DATAPATH ASIP NEW INSTRUCTIONS

Instruction	Description
AndNot	AndNot $destination, $source1, $source2 $destination ← (∼ $source1)&$source2
Rot1	Rot1$source1,$source2 **if** $source2 < 32 **then** Most ← $source1 ≪ $source2 Least ← $source1 ≫ (32 − $source2) **else** Least ← $source1 ≪ ($source2 − 32) Most ← $source1 ≫ (64 − $source2)) **end if**
Rot2	Rot2 $destination, $source1, $source2 **if** $source2 < 32 **then** Least ← Least\|($source1 ≪ $source2) Most ← Most\|($source1 ≫ (32 − $source2)) **else** Most ← Most\|($source1 ≪ ($source2 − 32)) Least ← Least\|($source1 ≫ (64 − $source2)) **end if** $destination ← Most
MFLeast	MFLeast $destination $destination ← Least

To support the ISE at the hardware level, the MIPS ALU is augmented with additional logic as shown in Figure 3. For the rotation instructions, two special internal registers: *MSW* and *LSW* are introduced to hold the intermediate value for the 64-bit rotation because the full rotation operation takes two cycles to execute. Because only one 32-bit output can be written back to the register file, the most significant 32 bits of the output are written back to the register file and the least significant 32 bits are written back in another cycle. To switch between the most and least significant words, a multiplexer is introduced. In order to implement the AndNot instruction without adding extra logic, a resource sharing approach is used by adding a NOT gate before the ALU input (SrcA) incorporated within an already existing multiplexer.

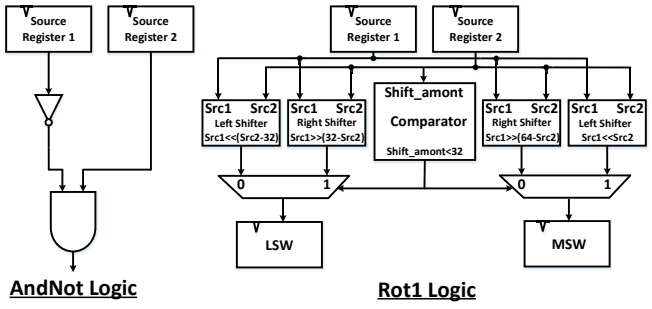

Fig. 3. SHA-3 MIPS-based native datapath ASIP

B. SHA-3 Coprocessor-Based ASIP

In this approach, the SHA-3 computation bottlenecks are tackled by adding a SHA-3 coprocessor that operates on multiple inputs at once, and in order to keep the MIPS register file architecture unchanged, five auxiliary registers are used. The syntax of the added instructions and their high-level description are depicted in Table II.

TABLE II
SHA-3 COPROCESSOR NEW INSTRUCTIONS

Instruction	Description
LWAu	LWAu $destinationAu, #offset($base) Executed by the regular ALU: $destinationAu ← memory[$base + #offset]
XOR5	XOR5 #offset($base) Executed by the additional ALU: Result ← $Au0 ⊕ $Au1 ⊕ $Au2 ⊕ $Au3 ⊕ $Au4 Executed by the regular ALU: memory[$base + #offset] ← Result
Chi	Chi #index, #offset($base) Executed by the additional ALU: Result ← $Au[#index]] ⊕ ((∼ $Au[#index + 1])&Au[#index + 2]) Executed by the regular ALU: memory[$base + #offset] ← Result
Rot	Rot #offset($base) Executed by the additional ALU: {Most, Least} = {$Au0, $Au1} ≪ $Au2 Executed by the regular ALU: memory[$base + #offset] ← Most
SWLeast	SWLeast #offset($base) Executed by the regular ALU: memory[$base + #offset] ← Least

The microarchitecture of the extended MIPS processor is shown in Figure 4. Five auxiliary registers are added to the decode stage to supply parallel inputs to the Co-ALU which can execute three operations only; XOR5, Rot and Chi. For the Rot instruction, only the most significant 32 bits are stored during the first cycle while the least significant 32-bits are held in an internal register to be stored in another cycle. This requires adding a multiplexer to select between the most and least words. To incorporate the coprocessor datapath within the MIPS processor without interfering with the original datapath, the memory data port input is multiplexed to select between the original and coprocessor datapaths. As the coprocessor supports loading from memory to its auxiliary registers, the auxiliary register data inputs are connected to the memory data output associated with a write enable and three bits for the auxiliary register address taken from the instruction bits[18:16] to choose between auxiliary registers. The same three bits are used as an immediate index for the Chi instruction.

V. Implementation Results and Evaluation

Both the reference MIPS and ASIP processor architectures are implemented and tested on a Xilinx Virtex-6 FPGA. The SHA-3 algorithm was executed on all architectures and the execution time was calculated for each one. The SHA-3

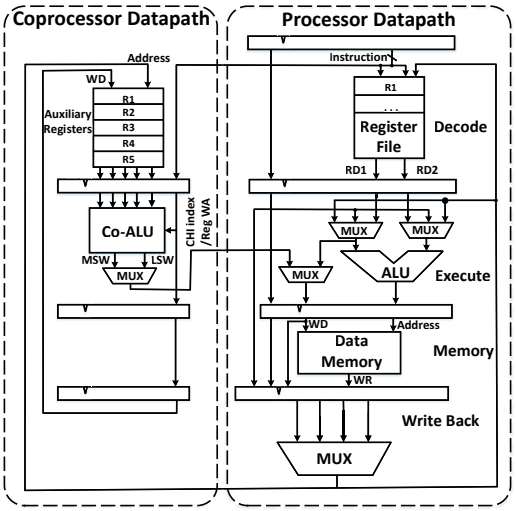

Fig. 4. SHA-3 coprocessor-based ASIP

extended architectures are compared to the reference MIPS processor in terms of the SHA-3 computation speed, total processor area, and SHA-3 code size or memory utilization. Table III provides a comparison between our basic MIPS, the native datapath and coprocessor-based extended MIPS, the KECCAK-extended PIC24 processor presented in [6] and the accelerated LEON3 processor presented in [7]. Results of the two developed ASIP in the table are referred to the reference MIPS results while the results of the compared architectures are referred to their base designs.

TABLE III
SHA-3 ASIP IMPLEMENTATION AND TESTING RESULTS

	Execution time Cycles/bytes	Area # of Slices	Code Memory Bytes
Reference MIPS	222.6	6074	3592
Native ISE	178.1(80%)	6595(109%)	3320(92%)
Coprocessor ISE	137.9(62%)	7643(126%)	3292(92%)
PIC24 [6]	188	-	3480
PIC24+ISE [6]	132(70%)	-	2415(69%)
LEON3 [7]	693	7902	20000
LEON3+ISE [7]	369(49%)	8648(109%)	18100 (89.5%)

The hash computation speed is defined as the number of bytes hashed per clock cycle which is inversely proportional to the computation time measured in clock cycles for hashing one byte of the message. The results show a significant reduction of SHA-3 execution time referred to the execution time on the MIPS reference architecture by 20% and 38%, which is equivalent to a 25% and 61% speedup in the computation of SHA-3 on native and coprocessor-based approaches, respectively.

The developed processors are synthesized and implemented on a Xilinx Virtex-6-XC6VLX75t-2-FF484 FPGA. Although extensions and modifications to the MIPS-I ISA are limited, the results show a relatively large overhead in the ASIP hardware resources specially for the coprocessor based ASIP. This is because the MIPS processor is lightweight and only a subset of the MIPS instructions are implemented. Applying both ISEs on a complete MIPS processor is expected to have less hardware area overhead. The SHA-3 executable code size is reduced by around 8% for both ASIPs.

From the previous results, it is clear that both ASIP approaches improved the execution speed and memory usage at the expense of hardware area. The coprocessor-based ASIP approach improves the throughput more significantly compared to the native datapath approach. Comparing our results to the ISE presented [6] shows a slight improvement of the execution time in our SHA-3 coprocessor ASIP, which is compensated in the code size of the PIC24 ISE. Comparing our work to the LEON3 FPGA-based ISE presented in [7] depicts that their execution time reduction ratio is better than ours. However, comparing the absolute execution time of SHA-3 on both 32-bit architectures illustrates a significant superiority of our SHA-3 ASIPs which is at least twice faster than the SHA-3 LEON3 ISE; the same applies for the memory usage.

VI. CONCLUSIONS

In this paper, we presented an ASIP design to improve the throughput of the SHA-3 algorithm computation on a lightweight MIPS processor. The ASIP design flow is adopted with the aid of Codasip Studio to develop a SHA-3 ISE to a 32-bit MIPS processor. Two ISEs were presented: native datapath ISE and coprocessor-based ISE. The extended MIPS processor is synthesized on a Virtex 6 FPGA, the native and coprocessor-based ASIPs improve the throughput of the reference implementation of the SHA-3 algorithm by 25% and 61.4%, respectively. The slice area of the MIPS processor, however, increased by 8.6% and 25.8% for the two approaches,respectively. The relative large overhead imposed by the two approaches is attributed to the small area of the MIPS reference processor because only a small subset of the MIPS ISA is implemented. As the ISA of the MIPS processor is similar to many 32-bit RISC processors, therefore, the presented SHA-3 ASIPs can be extended to other 32-bit RISC processor architectures.

REFERENCES

[1] NF Pub. DRAFT FIPS PUB 202: SHA-3 standard: Permutation-based hash and extendable-output functions. *FIPS Publication*, 2015.

[2] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge-based pseudo-random number generators. In *Cryptographic Hardware and Embedded Systems, CHES*, pages 33–47. Springer, 2010.

[3] External benchmarking extension (XBX). http://xbx.das-labor.org/trac/.

[4] Shu-jen Chang, Ray Perlner, William E Burr, Meltem Sönmez Turan, John M Kelsey, Souradyuti Paul, and Lawrence E Bassham. *Third-round report of the SHA-3 cryptographic hash algorithm competition*. US Department of Commerce, NIST, 2012.

[5] David Money and Sarah L Harris. *Digital design and computer architecture*. Morgan Kaufmann Publishers, 2007.

[6] Jeremy Hugues-Felix Constantin, Andreas Peter Burg, and Frank K Gurkaynak. Investigating the potential of custom instruction set extensions for SHA-3 candidates on a 16-bit microcontroller architecture. Technical report, Cryptology ePrint Archive, 2012.

[7] Yi Wang, Youhua Shi, Chao Wang, and Yajun Ha. FPGA-based SHA-3 acceleration on a 32-bit processor via instruction set extension. In *Electron Devices and Solid-State Circuits (EDSSC), 2015 IEEE International Conference on*, pages 305–308. IEEE, 2015.

[8] Codasip tutorials. www.codasip.com.

[9] Dr Kenneth Vollmar and Dr Pete Sanderson. A MIPS assembly language simulator designed for education. *Journal of Computing Sciences in Colleges*, 21(1):95–101, 2005.

[10] SHA-3 reference C code. keccak.noekeon.org.