Lightweight Cryptographic Instruction Set Extension on Xtensa Processor

Gabriel H. Eisenkraemer*, Fernando G. Moraes†, Leonardo L. de Oliveira* and Everton Carara*
*Federal University of Santa Maria, Santa Maria, Brazil,
gabriel.eisenkraemer@ecomp.ufsm.br, leonardo@mail.ufsm.br, carara@ufsm.br
†School of Tecnhology - PUCRS – Av. Ipiranga 6681, 90619-900, Porto Alegre, Brazil, fernando.moraes@pucrs.br

***Abstract*—The emerging popularity of the Internet of Everything makes the security an urgent issue, as well as the need for speed to cipher and decipher any information, which is essential for the embedded devices. Unlike many works in this field, where propositions considering application specific integrated circuits (ASICs), coprocessors, field-programmable gate arrays (FPGAs) or software were presented as alternatives to raise the efficiency of execution, we addressed the enhancement of the instruction set architecture (ISA) taking advantage of a hybrid design methodology to boost the performance as well as the design itself. We validate our ISA by measuring the area overhead, memory parameters and the speedup for different optimized implementations of AES, DES, 3DES and SHA using the Cadence® LX7 Processor and Xtensa® platform. The proposed architectures provided an excellent tradeoff with the area, memory and cycle count performance figures. Experimental results show that the proposed ISA can reduce cycle count between 1.76 and 10.99 with a cost of 6% in average of area overhead in a lightweight processor architecture.**

***Index Terms*—AES, DES, SHA, Cryptography, IoT, Xtensa®**

I. Introduction

For several decades security has been a concern in computer networks, ranging from LANs up to the World Wide Web. Nowadays, it is also a concern in the embedded world, ranging from Sytems-on-Chip (SoCs) up to Internet-of-Things (IoT). As the number of elements that can be connected to the Internet keeps increasing, a new term has been proposed: The Internet of Everything (IoE). It expands the concept of IoT in that it connects not just physical devices but quite literally everything [1].

Modern embedded computing systems are based on programmable platforms composed by several IPs (e.g. processors and memories) integrated on a single chip and are able to dynamically load and execute dozens of applications. As consequence, such systems are likely to be threatened by malicious applications which can compromise the security in many ways, like sensible data access or data tempering. Once the system is connected to IoT its vulnerability widens, becoming susceptible to threats like identity theft and device jailbreaking. Unsurprisingly, security vulnerabilities are abound in modern SoC designs, as evidenced by the frequency and ease in which attack activities are performed [2].

From now on, security policies concerning data confidentiality/integrity, access control, and authentication are also mandatory in embedded computing systems. Cryptographic algorithms have been widely employed to provide some level of security, and they are present into everything, from Web browsers and e-mail programs to cell phones, bank cards, cars, and even into medical implants. Due to the intensive computing nature of several algorithms, hardware implementation is common solution to overcome the typical software approach [3][4][5][6], mainly when considering embedded systems with limited computation power. Such specialized hardware is typically integrated following two approaches: (*i*) loose coupling or (*ii*) tight coupling. The former approach connects the specialized hardware to the system bus as an ordinary peripheral and processor communicates to it through memory-mapped registers (load/store instructions) or I/O ports (in/out instructions). With the latter approach, the specialized hardware is integrated into the processor architecture (ISA) and accessed by specialized instructions. The loose coupling is the most common option due to the impossibility of architectural changes in commercial processors. Besides, the specialized hardware usually implements whole algorithms with high performance being commonly called accelerators [3][4]. In the tight coupling, the specialized hardware can be implemented as a coprocessor (e.g., Intel 8087, MIPS FPU), or it can be added to the processor data path. The second option has been leveraged by customizable processors like Xtensa® [7] and ARC 600 [8] and lately by open architectures like RISC-V [9] and MIPS [10]. The result is a new processor with a custom ISA, and the new instructions are available to the programmer via the same compiler and assembler that target the processor's base instructions.

To design a new ISA or even extend it, researchers must take into account several aspects inherent to the processor, like the hardware to implement such instructions or the compiler to support them. These tasks take time and should be tested to provide a coherent processing architecture, which is possibly the main reason why not so many works were proposed in this direction. Many efforts have already been made aiming ultra-low power consumption regarding ASIC-oriented approaches [6] or high throughput (Gb/s) using reconfigurable hardware (FPGA) [5]. On the other hand, enhancements on the ISA may be a feasible choice to keep the desired degree of programmability and performance delivered by dedicated hardware. The work in [11] extended the ISA of a 32-bit processor through an ARM processor simulator. The authors analyzed the more timing consuming parts of a software implementation of the AES algorithm and moved them into hardware, achieving between 1.43 and 3.45 improvements regarding the time spent to cipher data, but

978-1-7281-3320-1/20/$31.00 ©2020 IEEE

with no area overhead information.

The *goal* of this work is to present the ISA extension of the Xtensa® processor to accelerate the following cryptographic algorithms: (*i*) DES/3DES (symmetric cryptography); (*ii*) AES (symmetric cryptography); (*iii*) SHA-256 (cryptographic hash).

Besides the acceleration gains, we aim to reduce memory footprint and memory accesses with a low area overhead, in accordance with the scarcity of resources available to embedded devices. The selected symmetric cryptographic algorithms are widely and frequently used on confidential communications, whereas the cryptographic hash is applied in authentications (e.g., MAC) and integrity assurance.

II. Algorithms and Hotspot Analysis

This Section presents a brief description and hotspot software analysis of the selected algorithms. Furthermore, the basis upon which the extensions were constructed is also established, and will be further detailed on Section III.

A. DES

The Data Encryption Standard is a symmetric-key block cipher algorithm that follows the Feistel Cipher structure. Therefore, both encryption and decryption share the same transformations, differing only on the key expansion. Based on Feistel's work, it makes use of two simple methods, substitution (the replacement of a plaintext element by a corresponding ciphertext element) and permutation, as tools to elaborate its transformations. The general structure of the DES algorithm is composed of 16 rounds of encryption/decryption realized upon each plaintext block of 64 bits. Before the round 1, the plaintext passes through an Initial Permutation (IP) and after round 16 through its inverse (IP^{-1}). The key expansion process is based on left circular shifts and permutations (Permuted Choice 1 and 2). It takes the 64 bits key and generates 16 round keys to be used at each round. A round consists of (*i*) apply the F function to the right half of the block, (*ii*) XOR the result with the left half and (*iii*) make the initial right half the new left half of the block. The F function is responsible for performing the mentioned substitutions, through the use of substitution tables (S-box).

The most computing intensive parts of the algorithm are F function and key expansion, making them the hotspots of DES. They perform the most transformations (permutations and substitutions), therefore, they will be the focus of the acceleration process later detailed.

B. AES

The Advanced Encryption Standard is a symmetric block cipher based on the Rijndael algorithm, which makes use of four transformations to perform encryption/decryption. Depending on the chosen key size (128, 192 or 256 bits), the number of rounds is distinct, however, the algorithm's structure remains the same. AES works with plaintext blocks organized into 4x4 byte matrices called States, which are transformed in every round. Encryption consists of first applying an initial transformation to the block, called AddRoundKey, and then performing multiple rounds made of four transformations (SubBytes, ShiftRows, MixColumns, AddRoundKey), followed by one last round of only three transformations.

As with DES, AES applies iterative substitutions and permutations over the plaintext block throughout many rounds, using a round key on each of them. The key expansion process generates the round keys based on the original one and they are also organized into matrix form. Such process makes use of another two transformations very similar to SubBytes and ShiftRows. Considering a word oriented version of AES, the code for SubBytes and ShiftRows can be re-utilised on the key expansion, which is interesting to note for a hardware implementation.

The algorithm uses finite field arithmetic over the Galois Field GF(2^8) to perform all of its mathematical operations. Thus, every element B of the State matrix is a polynomial $\in$ GF(2^8) and is stored in an eight bits variable. Equation 1 details its representation.

$$B(x) = \sum_{n=0}^{7} b_i x^i \tag{1}$$

This brings important implications to the execution of the four transformations and their further hardware implementations, since regular arithmetic operations have different definitions in GF(2^8). Most software implementations of AES actually avoid calculating according to those definitions, since generally there is no efficient way to process them, having then to rely on lookup tables (LUTs) with pre-calculated results (S-box). A hardware approach, though, has better means to perform the required computation and is explored in this work.

Evaluating the hotspots of AES, the highest cycle counts belong to MixColumns and its inverse, since matrix multiplication requires many accesses to LUTs (S-box). SubBytes and ShiftRows, together with their inverses, also count for a significant portion of the cycles, so the focus of optimization are on these transformations.

C. SHA-256

The Standard Hash Algorithm 256 maps a variable-length input plaintext into a fixed-length ciphertext, 256 bits wide, known as the message digest. In a hash function, typically the input plaintext is padded so that its total bit length is multiple of a certain integer, 512 in the case of SHA-256. After padding, the message is divided in 512 bits blocks, which are then processed to generate the message digest. The most important component of the algorithm is the one-way compression function F, which updates the contents of the block in a set of 64 rounds. The transformations made over the blocks are chained, with every iteration of the F function feeding the next ones. Each round is basically composed of permutations, logic operations and additions performed over a set of 8 variables, those being a, b, c, d, e, f, g and h. To execute these, the algorithm makes use of auxiliary vector

variables 64 elements wide, namely a set of constants K and a message schedule, which is updated every round.

The greatest hotspot of SHA-256 is the round transformation, since it alone performs all changes made over the plaintext. This evidences the upmost relevance of the F function, as the highest portion of processing is consumed by it. Thus, the efforts of hardware optimization should focus on the round transformations.

III. Cryptographic Instructions Extension

The proposal to accelerate the algorithms is to act over their hotspots, to obtain the best tradeoff between acceleration and processor area overhead. Xtensa® Xplorer was employed together with Cadence's specialized HDL language called TIE (Tensilica Instruction Extension) to develop specialized instructions. The Xtensa® Xplorer creates an RTL description of the processor and generates tailored versions of all necessary software development tools including the compiler, assembler, debugger and instruction set simulator. The developed instructions can be fired from C code through ordinary functions.

A. Extensions for DES

Many of the transformations used by DES are merely permutations of bits. This is the case with Initial Permutation and its inverse applied on the block, Permutated Choices and left circular shift used in key expansion. While these may not perform well in software due to several shift and logic instructions, dedicated hardware excels in their execution, since a specific permutation only requires rewiring of its input bits. Out of the 12 instructions proposed for DES, 11 were developed just for permutations, providing most of the performance gain with very low area overhead. These instructions also support the implementation of 3DES which corresponds to encryption/decryption of data three times in a row in order to increase the security with a larger key.

The substitutions performed inside the F function make use of the S-boxes, which require 512 Bytes of constant data. The last extension proposed for DES takes advantage of a TIE construct called TIE tables, which supports the implementation of LUTs in hardware and allows several accesses within a single cycle. By using such tables it is possible to reduce memory access and memory footprint. Thus, the entire F function could be implemented in hardware.

B. Extensions for AES

AES yields different obstacles in relation to DES, since encryption and decryption do not share the same transformations, requiring more hardware. Considering this, word oriented generic instructions that can be shared throughout the algorithm are proposed, with the main objective of reducing area overhead. Such instructions act on a full row or column of the 4x4 State matrix.

The approach to optimize MixColumns relied on implementing hardware that fully calculates the matrix multiplications performed over the field GF(2^8), instead of typical LUT based implementations storing the multiplications results. As can be deduced by the rules of finite field arithmetic, multiplication by two of an element B $\in$ GF(2^8) expressed by the bitstring $[b_7b_6b_5b_4b_3b_2b_1b_0]$ can be achieved by performing a left shift followed by a conditional XOR with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. This is equivalent to performing the following transformation to an element:

$$B^{'} = 2 * B = [b_6b_5b_4(b_3 \oplus b_7)(b_2 \oplus b_7)b_1(b_0 \oplus b_7)b_7] \quad (2)$$

As can be seen in Equation 2 the logical implementation to double an element belonging to GF(2^8) requires only 3 XOR gates and rewiring (left shift). Up from this basic unit a full MixColumns multiplier was created. Multiplications by larger numbers can be achieved with simple tricks, for instance, in a multiplication by three the element is first doubled using Equation 2 and then added with its original value. Additions in GF(2^8) are implemented as bitwise XOR. Figure 1 depicts the developed architecture for the MixColumns transformation of one element of the State Matrix. Since the final extension operates on a whole column of State, four equal instances of the presented circuit were used. The resulting MixColumns multiplication instruction can be used for encryption and decryption, and logically requires roughly 280 XOR gates and 16 AND gates.

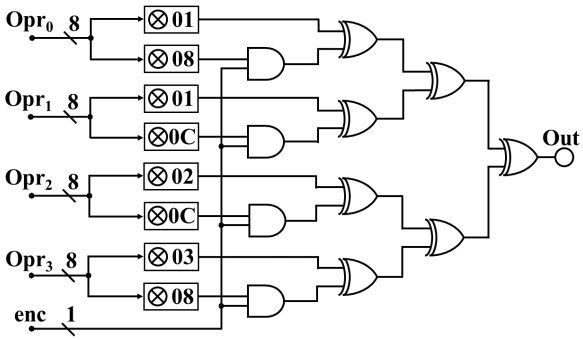

Figure 1. Architecture of the Mixcolumns Multiplier.

The second instruction created takes care of SubBytes and ShiftRows and can be used for encryption, decryption and key scheduling. ShiftRows is performed by merely selecting one out of the possible byte shifts (rewiring) that can be made over a State row or word (in case of key scheduling). SubBytes involved creating a GF(2^8) inverter and a module that performs the affine transformation and its inverse. Several works have already been proposed on this subject, the new instruction proposed is based on [12], though the hardware they presented was expanded here to work on a full row of the State matrix. The logical implementation of this second instruction component required 584 XOR gates and 280 AND gates.

C. Extensions for SHA-256

Conceptually SHA-256 acceleration is mostly straightforward, with the core of the matter being the compression function F. To make the chained transformations, the algorithm uses a temporary buffer throughout all iterations of the hash

process composed of the variables a, b, c, d, e, f, g and h, resulting in many memory accesses during the 64 rounds of the F function. To solve this issue special registers were created, cutting the need to push data back and forth between CPU and memory, making it simple to implement an instruction which calculates one full round of the F function.

Another costly computing task the algorithm has to process is preparing the message schedule, which needs to be done in every iteration of the hash process. This is the reason behind the second instruction proposed. The aforementioned special registers were also employed to accelerate message scheduling, which is based on several permutations.

IV. Evaluation and Discussion

The proposed extensions were implemented on a baseline Xtensa configuration upon which all original algorithms were executed. The baseline processor is a 32 bits architecture configured with the aim of simplicity, occupying an area of 110 Kgates, running at a clock rate of 910 MHz. All additional functionalities Xtensa Xplorer offers to configure processors were disabled, as to keep the use of resources to minimal levels. Figures 2 and 3 show the obtained results in terms of cycle count, memory footprint and memory access along with the area overhead. The baseline serves as the reference to compare the obtained results with the baseline processor executing each algorithm without any extension. Source C code of each algorithm is available at [13]. The bars below the baseline represent gains whereas above represent overhead.

The area overhead, ranging from 4.84% for AES to a maximum of 6.78% for DES, provides significant gains at low cost, achieving an attractive trade-off. Memory access and memory footprint have been drastically reduced. In the case of AES and DES, this is mainly due to the elimination of LUTs (S-boxes and multiplications), which were accessed multiple times on the base implementation during encryption/decryption. The reduction on memory access observed on SHA-256 is explained by the adoption of the special registers, which made it possible to perform the whole 64 rounds of the F function without accessing the memory buffer. Memory accesses have become much more costly than arithmetic computations, as noted by Horowitz [14]. For example, accessing a block in a 32-kilobyte cache involves an energy cost approximately 200x higher than a 32-bit integer add. This important differential makes optimizing memory accesses critical to achieving high-energy efficiency [15].

Cycle count has also decreased significantly. The extensions were able to nearly cut in half the cycle count of AES and SHA-256 achieving, respectively, an average speedup of 1.85 and 1.91. DES and 3DES performance increased even more achieving, respectively, an average speedup of 7.4 and 10.5. The permutations required by DES/3DES are bit oriented, which are poorly executed in software because shift/logic instructions are word oriented. A noteworthy observation is the consistency of performance gains throughout different variations of the algorithms. The three versions of AES performed roughly the same, while 3DES obtained small extra gains in relation to regular DES. This fact happened without

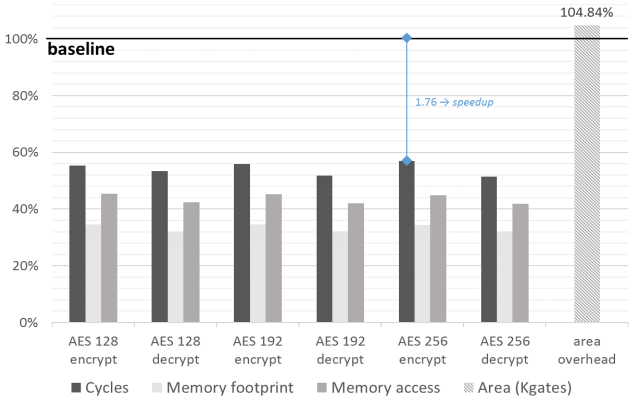

Figure 2. 128-192-256 encryption/decryption AES enhanced architecture.

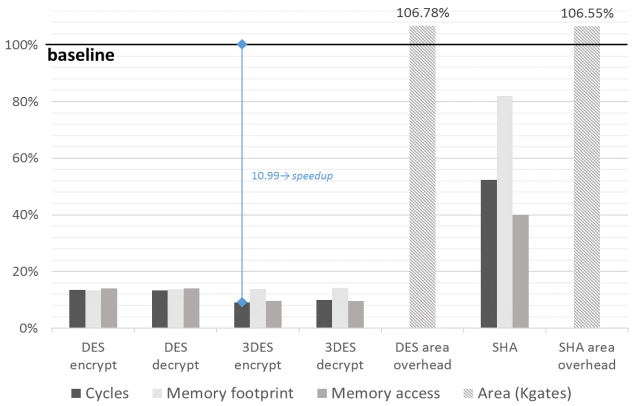

Figure 3. Encryption/decryption DES/3DES/SHA enhanced architecture.

any modifications on the proposed extensions, endorsing the flexibility of the solution.

V. Conclusion

This paper proposes an instruction set extension for 3 different cryptography systems, taking advantage of specifically designed instructions and software flexibility supported by the ISA of the processor. The Cadence® LX7 ® processor was adopted as the carrier of such algorithms, and the extended architecture was designed with Tensilica® TIE language. Experimental results show the viability of the proposal of optimizing the hotspots through a hardware-software co-design by the use of extensions. The developed extensions reduced at the minimum almost half of the cycle count at a cost of 6% in average of area overhead. The implementation has the resilience of being an instruction set of a configurable processor architecture, while at the same time providing excellent performance gains with low area trade-offs.

Acknowledgement

The Author Fernando Gehm Moraes is supported by FAPERGS (17/2551-0001196-1) and CNPq (302531/2016-5), Brazilian funding agencies. The authors would like to thank the Cadence® Academic Network.

REFERENCES

[1] R. Chandhok, "The internet of everything," in *2014 IEEE Hot Chips 26 Symposium (HCS), Cupertino, CA, USA, August 10-12, 2014*. IEEE, 2014, pp. 1–29. [Online]. Available: https://doi.org/10.1109/HOTCHIPS.2014.7478826

[2] S. Ray and J. Bhadra, "Security challenges in mobile and iot systems," in *2016 29th IEEE International System-on-Chip Conference (SOCC)*, Sep. 2016, pp. 356–361.

[3] Z. Jiang, H. Jin, G. E. Suh, and Z. Zhang, "Designing secure cryptographic accelerators with information flow enforcement: A case study on aes," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: ACM, 2019, pp. 59:1–59:6. [Online]. Available: http://doi.acm.org/10.1145/3316781.3317798

[4] M. Korona, T. Wojciechowski, M. Rawski, and P. Tomaszewicz, "Cryptographic coprocessor with modular architecture for research and development of countermeasures against power-based side-channel attacks," in *2019 MIXDES - 26th International Conference "Mixed Design of Integrated Circuits and Systems"*, June 2019, pp. 190–195.

[5] A. A. Nacci, V. Rana, D. Sciuto, and M. D. Santambrogio, "An open-source, efficient, and parameterizable hardware implementation of the aes algorithm," in *2014 IEEE International Symposium on Parallel and Distributed Processing with Applications*, Aug 2014, pp. 85–92.

[6] M. A. Bahnasawi, K. Ibrahim, A. Mohamed, M. K. Mohamed, A. Moustafa, K. Abdelmonem, Y. Ismail, and H. Mostafa, "Asic-oriented comparative review of hardware security algorithms for internet of things applications," in *2016 28th International Conference on Microelectronics (ICM)*, Dec 2016, pp. 285–288.

[7] CADENCE, "Tensilica customizable processors," https://www.synopsys.com/designware-ip/processor-solutions/arc-600-family.html, accessed: 2019-10-01.

[8] SYNOPSYS, "Designware ARC 600 processor core family," https://www.synopsys.com/designware-ip/processor-solutions/arc-600-family.html, accessed: 2019-10-01.

[9] D. Patterson and A. Waterman, *The RISC-V Reader: An Open Architecture Atlas*. Strawberry Canyon, 2017.

[10] W. Computing, "MIPS Open," https://wavecomp.ai/mips-open, accessed: 2019-10-01.

[11] G. M. Bertoni, L. Breveglieri, F. Roberto, and F. Regazzoni, "Speeding up aes by extending a 32 bit processor instruction set," in *IEEE 17th International Conference on Application-specific Systems, Architectures and Processors (ASAP'06)*, Sep. 2006, pp. 275–282.

[12] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An asic implementation of the aes sboxes," in *CT-RSA 2002. Lecture Notes in Computer Science*, vol. 2271. Springer, Berlin, Heidelberg, Oct. 2002.

[13] B. Conte, "Crypto-algorithms," https://github.com/B-Con/crypto-algorithms, accessed: 2019-10-01.

[14] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014, pp. 10–14.

[15] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, no. 2, pp. 48–60, Jan. 2019. [Online]. Available: http://doi.acm.org/10.1145/3282307