Extending a RISC-V core with an AES hardware accelerator to meet IOT constraints

Anthony Zgheib, Olivier Potin, Jean-Baptiste Rigaud, Jean-Max Dutertre

Extending a RISC-V core with an AES hardware accelerator to meet IOT constraints

Anthony ZGHEIB, Olivier POTIN, Jean-Baptiste RIGAUD, Jean-Max DUTERTRE
Mines Saint-Etienne, CEA-Tech Centre CMP, F - 13541 Gardanne, France
zgheib@emse.fr, olivier.potin@emse.fr, rigaud@emse.fr, dutertre@emse.fr

***Abstract*—Internet of Things devices and applications are subject to strong constraints in terms of cost, code size and power consumption. This leads to difficulties in using resource-hungry encryption algorithms to ensure the confidentiality of the exchanged data. In this paper, we extend with a custom instruction the RISC-V open source Instruction Set Architecture (ISA) and integrate an Advanced Encryption Standard (AES) hardware accelerator to an IBEX RISC-V core. This is achieved for the sake of reducing its energy consumption, encryption time and code size with respect to purely AES software solutions. We consider a Field Programmable Gate Array implementation and ascertain its relevance for an Electrocardiography use case.**

***Index Terms*—IOT, AES, RISC-V, ISA, ECG, power consumption.**

I. INTRODUCTION

The Internet of Things (IOT) consists in embedded systems driving physical objects (e.g. sensors, actuators) connected to the internet. Through the internet, these physical objects communicate and exchange information between each others or with servers. Designers aim at improving the performance of the IOT applications in terms of code size, speed, power consumption and security. For a secure data exchange in IOT applications, and from privacy concerns, the exchanged data need to be encrypted. The Advanced Encryption Standard (AES) is a well known algorithm used for data blocks' encryption and decryption [1]. Its implementation can be either hardware or software. However, the use of AES software encryption algorithms requires high energy consumption and encryption time. These characteristics do not meet IOT constraints. In this work, we describe a hardware-based 128-bit AES implementation to an IBEX [2] RISC-V processor [3] to solve this problem. We extend the RISC-V open source Instruction Set Architecture (ISA) with a custom instruction to encrypt the data to be exchanged. Furthermore, we compare our approach to two software-based implementations: the TinyAES [4] and the OpenSSL AES [5] to reflect the performance of our solution. We also consider an Electrocardiography (ECG) application use case. Our paper is organized as follows: Section II details the design of the hardware AES Intellectual Property (IP) and its connection with the RISC-V processor. Section III describes the simulations and the circuit characteristics. Section IV leads to the comparison between the software and hardware AES models. Furthermore, Section V mentions a ciphering example of ECG signals. Finally, Section VI presents our conclusion.

II. HARDWARE AES RISC-V CORE EXTENSION

A. Hardware AES IP design

Our 128-bit AES IP is based on the Federal Information Processing Standard (FIPS) 197 standards [1]. It computes the transformations of one AES round, from the initial round 0 through rounds 1 to 9 up to the final round 10 of the complete AES Rijndael design. The round ciphering depends on the round and the block cipher mode. We integrate four block cipher modes of operation: Electronic codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB) and Output Feedback (OFB) [6]. Fig. 1 shows the architecture of the AES IP. It has four 128-bit registers (drawn in green) for storing the data to be encrypted (plaintext), the encryption key, the initial vector (IV) or the encrypted result and the last register for storing the round key. The AES IP receives from the circuit: the clock and reset signals. Similarly, it receives from the RISC-V core the following input signals: the operating cipher mode (mode_i), the calculation round number (round_i), the enable round computing signal (en_rnd_cpt_i), the plaintext, key or IV (data_i), the writing address (wr_addr_i) and its enable signal (wr_en_i). When the encryption is done, the plaintext, key or encrypted result could be read with the read address (rd_addr_i) and read enable (rd_en_i) signals. The input and output signals (data_i and data_o) are 32-bit buses for easier compatibility while loading or storing values from or to the 32-bit RISC-V core registers. Hence, four store operations are needed to store a 128-bit plaintext, key or IV.

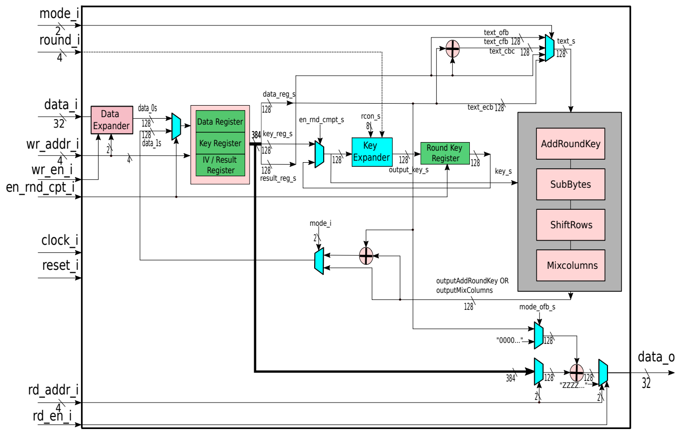

Fig. 1. Architecture and interface signals of the AES IP.

B. Extending the RISC-V ISA

RISC-V cores have a free open source ISA designed for extensibility in comparison to other cores like x86 and ARM [7]. In our work, we exploit the RISC-V ISA extension's capability to integrate the AES hardware accelerator efficiently. We extended the RISC-V ISA with a new custom instruction in order to communicate with and to control the operations of the AES IP through the IBEX core. The use of a custom dedicated instruction reduces the number of assembly instructions used to perform an encryption. In addition, it brings an improvement in regard of the AES encryption speed. This is also demonstrated by [8] with their AES implementation experiments based on a SPARC V8-compatible LEON-2 processor for the S-BOX calculation. Our AES custom instruction has to be able to perform the three following operations:

- A store operation (moving data from the IBEX registers to the AES registers).
- A round operation (executing the AES round calculations).
- A load operation (moving data from the AES registers to the IBEX registers).

A single RISC-V 32-bit R-type instruction (as a register to register operation is called) permits us to operate our IP in all its functionalities. Its format is given in Fig. 2. Funct7 is used to code the round number, the specified chain mode and the required operation. Table I enumerates the values of funct7 we used for this purpose. R_3, R_2, R_1 and R_0 represent the 4 bits of the round number. For our AES implementation, the funct3 and opcode labels are fixed to "111" and "0001011" having a compatibility with the RISC-V funct3 and opcodes required format [9]. Rs1 and rd represent respectively the addresses of the source and destination registers (the second source register rs2 is not used). Their values depend on the instruction operation, they could point to IBEX or AES registers.

C. AES IP - IBEX connection

The IBEX core is a 32-bit open source RISC-V central processing unit (CPU) core. It is a low power and small processor suitable for IOT applications. It has a 2-stage pipeline: Instruction Fetch (IF) and Intruction Decode and Execute (ID/EX) [2]. IBEX is used in the opentitan project provided by lowrisc [10]. This project also implements a hardware AES IP, for which the communication between the processor and the IP is made using a set of control and status registers (CSRS). This allows the ciphering/deciphering processes to be achieved in parallel to the processor activity. In our case, we communicate with the AES IP and execute its calculation using the new custom instruction which prevent a parallel execution.

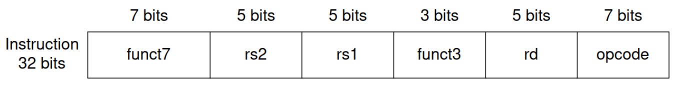

Fig. 2. The RISC-V 32-bit R-type instruction format.

TABLE I
FUNCT7 VALUES AND DESCRIPTIONS

Funct7 value	Description
XXXX001	Store instruction
$R_3R_2R_1R_0$100	Round Instruction - Mode ECB
$R_3R_2R_1R_0$101	Round Instruction - Mode CBC
$R_3R_2R_1R_0$110	Round Instruction - Mode CFB
$R_3R_2R_1R_0$111	Round Instruction - Mode OFB
XXXX000	Load instruction-Modes ECB,CBC and CFB
XXXX011	Load instruction-Mode OFB

We made modifications to the IBEX core, especially to its Decode (ID) stage to adapt it to the new custom AES instruction. A simple version of the connection between the IBEX's ID stage and the AES IP is illustrated in Fig. 3. As an instruction arrives from the Fetch Stage to the Decode Stage, it is decoded to various signals. In case of an AES instruction, dedicated signals will be used to activate and control the AES IP. Referring to Fig. 3, the bus data_i is used to transfer either the plaintext to be ciphered, the key, or the IV used in AES chain modes. These values are sent from the IBEX registers. Similarly, data_o can be used to transfer the plaintext, the key, or the ciphertext to be stored in the IBEX registers.

D. Application code modification and compilation

In order to avoid any modification of the compiler toolchain, we used the new AES instruction as a macro instruction to control the AES IP. An AES encryption process involves the use of the AES macro instruction several times, using different funct7 values as mentioned in Table I, in order to sequentially store the plaintext and key in the AES registers, then to launch the 11 AES round calculations and, at the end to store the ciphertext in the IBEX registers. The macro instruction is defined using the inline assembly (ASM) code. An instance of its define format is represented as follows: *#define AES_macro_insn(_funct7, _rs1, _rd) /*
r_type_insn(_funct7, 0b00000, _rs1, 0b111, _rd, 0b0001011)
The AES macro instruction is a R-type instruction with fixed values for the rs2, funct3 and opcode fields corresponding to the AES R-type instruction fields. Consequently, we can replace the funct7, rs1 and rd fields with the required values in the application code. By compiling the application code, we obtain the virtual memory (VMEM) file as one of the output files. This file contains the binary instructions interfaced with the RISC-V core to be executed. The VMEM code size of our AES instructions dedicated for an ECB encryption with our AES IP is equal to 0.6 KBytes.

III. SIMULATION RESULTS AND CIRCUIT CHARACTERISTICS

A. Simulation results

Using Vivado 2018.1 tools, we have successfully simulated the place & root design of the IBEX core with the AES IP for all the ciphering modes and their chain processes. As an illustration, we report the execution time of a test code that executes a simple AES encryption in the ECB mode. It takes 140 clock cycles for the whole test code to be executed, from

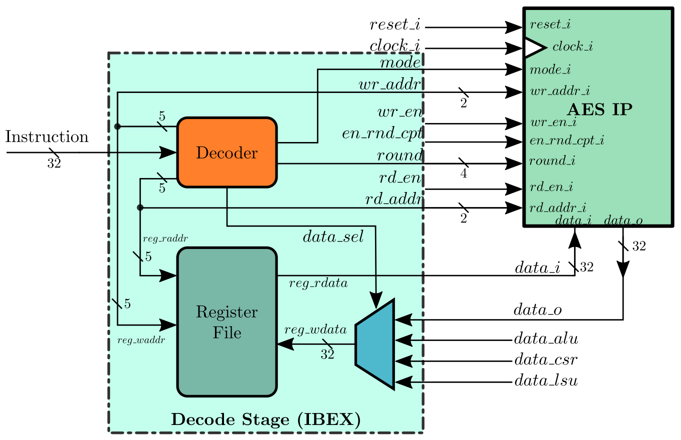

Fig. 3. The connection between the IBEX's decode stage and the AES IP.

the initialization of the plaintext and key values in the memory to the end of the program. From the 140 cycles, the AES IP consumes 60 from the first store instruction to its registers till the last load to the IBEX registers (ciphertext). And from these 60 cycles, 23 correspond to AES instructions execution: there are 4 for the storage of the plaintext and the same for the key, 11 for the AES rounds and 4 to store the ciphertext into the IBEX registers. The instructions that consume the most in terms of clock cycles are the Load Word (LW) operations from the memory to the IBEX registers. These operations are needed before transferring data from the core to the AES registers.

B. *Circuit characteristics*

1) Area: All our implementations targeted a Nexys Artix-7 Field Programmable Gate Array (FPGA) board, a 18nm CMOS technology. This board contains 33,650 logic slices. Each slice is composed of four 6-input LUTs (LookUp Tables), 8 flip-flops, multiplexers and carry units. As a first approach, we placed and routed the AES IP. Its design requires 547 slices referring to the Vivado utilization report. Also, we did the same for the IBEX core which needed 765 slices. The top level design containing the IBEX core with the clock generator and the random-access memory (RAM) component needs 886 slices. However, when implementing the AES IP to the IBEX core, their utilization requirements in terms of slices were adjusted to adapt the connection between their input and output signals. The AES IP and IBEX core require respectively 609 and 757 slices when placed in a top level design with a clock generator and RAM. In total, the design needs 1,479 slices. For the current AES model and comparing the two top levels, adding the AES IP makes the circuit bigger by 40% in terms of slices. As mentioned before, the IP contains 4 block cipher modes. By modifying it and just conserving the ECB mode, we can gain approximately 5% in the AES area. Depending on the application, a compromise could be made between the area gain or the block cipher modes requirements.

2) Power: We estimated the power consumption of our place & root design using the Vivado report power utility. For an accurate measure, we took into consideration the Switching Activity Interchange Format (SAIF) file which plots all the signals activity information for one ECB ciphering process simulation. The total power consumed for one ECB ciphering by the IBEX core and AES IP is equal to 397 mW.

IV. Comparison with 2 software AES algorithms

A. *Description of the software algorithms*

The 8-bit TinyAES is a small and portable implementation of the AES. It has the ECB, CounTeR (CTR) and CBC encryption chain modes [4]. The 32-bit OpenSSL AES uses LUTs containing intermediate results of an AES ciphering process and could operate under ECB, CBC, CFB and OFB chain modes [5]. Both codes contain functions for encryption and decryption. The unused functions like the decryption ones have been removed from our test code to have a fair comparison between the two algorithms and our hardware implementation. Also, only the ECB mode have been activated for the simulation. We give in table II, the test code size (VMEM size) dedicated to perform a single ECB encryption either by using the software algorithms (TinyAES or OpenSSL AES), or by using our hardware AES IP with their necessary time, power and encryption energy. The total power consumption for one encryption is equal to 380 mW for the TinyAES and 378 mW for the OpenSSL AES.

B. *Encryption energy interpretation*

Referring to Table II, the AES IP implementation consumes less energy in µJ than the TinyAES and OpenSSL AES. The largest encryption time for having one ECB encryption corresponds to the TinyAES (70,036*20ns) where 20 ns is the system's clock period. By taking the AES chain modes into consideration, the encryption time for the following encryption processes requires less clock cycles since the key has already been defined for the TinyAES and OpenSSL AES or has been loaded at the first encryption process in the hardware AES key register. Table III shows the chain encryption time with the required energy for the three AES solutions. We can check that, for a single ECB encryption with the TinyAES, 17 data encryption with the OpenSSL AES and 847 data encryption with our AES IP could be made with the same energy budget.

V. ECG use case Scenario

Several IOT applications require the confidentiality to be considered as a major aspect while exchanging data. For instance, in the medical field, patients' private medical data need to be protected. In this section, we refer to an AES ciphering process to cipher ECG signals. The ECG is known as a method to record the electric heart muscles' activity.

TABLE II
Comparison between AES solutions for a single ECB ciphering

AES Type	Vmem(kB)	Clock cycles	Power(mW)	Energy(µJ)
AES IP	0.6	140	397	1.11
TinyAES	5.7	70,036	380	532.27
OpenSSL	31.6	6,445	378	48.59

TABLE III
CHAIN ENCRYPTION DATA WITH THE SAME KEY

AES Type	Clock cycles	Energy(µJ)
AES IP	76	0.627
TinyAES	57,856	437.39
OpenSSL	3,874	29.13

We consider a Holter device which is a continuous ECG monitoring medical device. It monitors ECG signals from 24 to 48 hours as a short term or from 1 to 2 weeks as a long term [11]. Referring to [12] and [13], a database named "SHAREE" contains ECG Holter signals of 139 hypertensive patients recorded for 24h in order to be exploited in the context of medical research. The recordings were based on 3 ECG signals that were sampled at 128 samples per second with an 8-bit precision. This leads to 3,072 bits/second which is equivalent to 384 Bytes/second. We consider the case of sending the ECG signals gradually as the Holter device receives the patient heart's signals to the medical unit via an IOT platform. Using the AES encryption model, a message of 128 bits (16 Bytes) could be encrypted. Hence, on the base of this ECG use case, 24 encryption processes are required. Fig. 4, represented on a log-log scale, shows the encryption energy required for the three AES considered solutions. For the ECG application, the ciphering process using the AES hardware implementation requires 0.016 mJ compared to 10.592 mJ and 0.719 mJ for the TinyAES and OpenSSL AES. Our AES IP respectively consumes 662 and 44.9 times less energy than the TinyAES and OpenSSL AES. This reflects the low energy aspect of our IP in favor to be used in IOT applications. If the Holter system's total energy consumption also depends on the energy used for the communication process, the ECG signals filtering, recording and more, this energy will be approximately the same for the three encryption solutions.

VI. Conclusion

In this work, we report the implementation of an AES hardware accelerator for the purpose of reducing the energy consumption, encryption time and code size of a circuit designed for IOT applications. We also illustrate the potential of RISC-V cores through the use of their open-source ISA to improve their performances in IOT applications. This is achieved by extending the RISC-V ISA to implement our AES accelerator to an IBEX RISC-V core. We show the benefit of adding a new instruction to the RISC-V ISA by comparing our AES hardware implementation to two software algorithms: the TinyAES and OpenSSL AES. As a result, the application code size (VMEM size) dedicated to encrypt a message with the AES IP is small and have a size gain of 89.47% and 98.1% with respect to the TinyAES and OpenSSL AES. Furthermore, the AES IP requires 140 clock cycles to obtain an ECB ciphertext compared to 70,036 and 6,445 for the TinyAES and OpenSSL AES codes respectively. Alternatively, the AES IP adds 40% in terms of slices to the final circuit. However, more optimizations could be made to our IP to gain in area. Finally, we demonstrate in an ECG use case that our AES accelerator

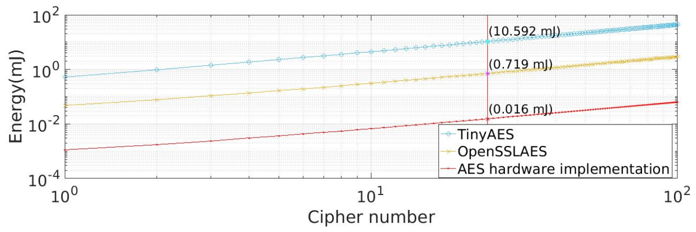

Fig. 4. The energy consumption for an ECB ciphering with the AES hardware implementation, TinyAES and OpenSSL AES via an ECG use case.

respectively consumes 662 and 44.9 times less energy than the TinyAES and OpenSSL AES. This shows the relevance of our solution and promotes its use in IOT applications. Another perspective is to investigate the LightWeight Cryptography (LWC) candidates of the National Institute of Standards and Technology (NIST) contest like ASCON [14] or already existing LWC algorithms (e.g. PRESENT [15]) to reduce further the size and energy overheads of the IBEX core. A similar work to our AES implementation could be found in [16], where they implemented ASCON and ISAP accelerators to a RI5CY core while extending the RISC-V ISA. Comparing to ASCON, ISAP offers in addition protection against various types of fault attacks.

References

[1] NIST-FIPS Standard, "Announcing the advanced encryption standard (aes)," FIPS Publication, 197, pp.1-51, Nov 2001.
[2] Lowrisc, IBEX documentation, Oct 2020, https://ibex-core.readthedocs.io/en/latest.
[3] D. Patterson and A. Waterman, "The RISC-V Reader: An Open Architecture Atlas," Strawberry Canyon, 1st ed., 2017.
[4] kokke, "Small portable AES128/192/256 in C," https://github.com/kokke/tiny-AES-c, 2016.
[5] The OpenSSL Project, "OpenSSL: The open source toolkit for SSL/TLS," http://www.openssl.org, 2019.
[6] M. Dworkin, "Recommendation for block cipher modes of operation: Methods and techniques," Technical report, USA, 2001.
[7] K. Asanović and D. Patterson, "Instruction sets should be free: The case for risc-v," Technical report, UCB/EECS-2014-146, EECS Department, University of California, Berkeley, Aug 2014.
[8] S. Tillich, J. Großschädl, and A. Szekely, "An instruction set extension for fast and memory-efficient aes implementation," in Communications and Multimedia Security, vol. 3677, J. Dittmann, S. Katzenbeisser, and A. Uhl, Eds. Berlin, Springer Berlin Heidelberg, pp. 11–21, 2005.
[9] The RISC-V International com., RISC-V custom opcodes, Nov 2013.
[10] Opentitan, AES HWIP Technical Specification, https://docs.opentitan.org/hw/ip/aes/doc, 2021
[11] P. Zimetbaum and A. Goldman, "Ambulatory arrhythmia monitoring," Circulation, vol. 122, no. 16, pp. 1629-1636, Oct 2010.
[12] P. Melillo et al., "Automatic prediction of cardiovascular and cerebrovascular events using heart rate variability analysis," PLOS ONE, vol. 10, no. 3, pp.1-14, Mar 2015.
[13] A. Goldberger et al., "Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals," Circulation, vol.101, no.23, pp. e215–e220, 2000.
[14] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2", NIST Round 2 Candidate, https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-2-Candidates, 2019.
[15] A. Bogdanov et al., "PRESENT: An ultra-lightweight block cipher," International workshop on cryptographic hardware and embedded systems, Springer, pp. 450–466, 2007.
[16] S. Steinegger and R. Primas, "A Fast and Compact RISC-V Accelerator for Ascon and Friends," in CARDIS 2020: 19th Smart Card Research and Advanced Application Conference, 2020.