Accelerated Software Implementation of Authenticated Encryption Stream Ciphers for High Speed Applications

Sara Taha[1], Hassan Mostafa[1, 2]
[1] Department of Electronics and Communications Engineering, Cairo University, Egypt [2] University of Science and technology, Nanotechnology and Nanoelectronics Program, Zewail City of Science and Technology, October Gardens, 6th of October, Giza 12578, Egypt.
sarahtahamostafa@gmail.com , hmostafa@uwaterloo.ca

***Abstract*– In this paper software implementation performance of CAESAR's competition round 3 Authenticated Encryption (AE) stream cipher algorithms is improved using Xilinx SDSoC tool. Xilinx SDSoC (Software Defined System-on-Chip) tool accelerates designs running on Zynq 7000 devices by implement heterogeneous co-design run on FPGA-CPU platform. AE schemes are providing both confidentiality and integrity of data which have a major role in wide number of applications such as IoT, Automotive, Medical devices, Sensors and real world protocols like TLS, SSH or IPsec. AE can be implemented either in software or in hardware. This paper will show that using SW-HW co-design improves the speed of the software implementation for ACORN, AEGIS, MORUS and Tiaoxin AE stream cipher algorithms.**

Index Terms – Authenticated Encryption, ACORN, MORUS, AEGIS, Tiaoxin, Stream Ciphers, CAESAR, SDSoC.

I. INTRODUCTION

Authenticated Encryption is symmetric key cryptographic algorithms that provide both security and integrity using same algorithm [2]. Competition for Authenticated Encryption: Security, Applicability and Robustness (CAESAR) cryptographic competition was announced to select a portfolio for single path AE scheme which is faster than AES-GCM. The first round of CAESAR was started with 57 candidates, 30 ciphers were selected for the 2nd round, 15 ciphers were selected for the 3rd round and the final portfolio was announced in February 2019.

There are three approaches to design an integrated authenticated encryption algorithm, block cipher, stream cipher, and dedicated cipher [1]. Stream cipher takes input as a secret Key (K) and an Initialization Vector (IV) and loads them to keystream generator. Then, either the output of keystream generator will be XORed with plaintext to produce ciphertext for encryption process or the output of keystream generator will be XORed with ciphertext to generate plaintext for decryption process [4]. Stream cipher usually divides the message into successive characters. Based on the size of the character, a stream cipher can be either bit-based or word-based. In the bit-based stream cipher, the cipher operates on each bit separately. In the word-based stream cipher, each character consists of a group of bits called a word and the cipher operates on these words to encrypt/ decrypt a message as described in [1].

Stream Cipher AE schemes have five phases of operation called: initialization, associated data loading, encryption, tag generation, and finally decryption and tag verification. For each phase, the internal state registers' values will be updated after applying some logic operations on them. Then these values will be used to define the keystream generator output. The output tag is generated by applying plaintext as an input of state registers as described in [1].

In this paper, the software implementation performance of CAESAR Round 3 AE stream cipher algorithms namely ACORN, MORUS, AEGIS, and Tiaoxin is enhanced using Zynq ZC702 evaluation board by moving some functions to be implemented on FPGA instead of CPU to reduce the number of clock cycles which are needed for encryption or decryption operations. C++ source codes which are available on ATHENa website [5] were used as a reference and modified to be applicable to be implemented on heterogeneous FPGA-CPU platform. Then some functions were selected to be implemented on FPGA. Finally, the speed enhancement was measured after using this platform for each algorithm.

The paper is organized as follows: section II describes Xilinx ZC702 heterogeneous CPU-FPGA platform. Section III describes an overview and the proposed implementation of ACORN, AEGIS, MORUS and Tiaoxin algorithms. Section IV shows the results of co-design implementation. Section V shows designs recommendation. Finally, Section VI concludes the paper work.

II. HETEROGENEOUS PLATFORM

Xilinx SDSoC (Software Defined System-on-Chip) tool is a C/C++ development environment used to create hardware-software co-designs on a heterogeneous FPGA-CPU for Zynq®-7000 All Programmable SoC platforms. It is used to improve performance of C/C++ code by reducing number of clock cycles of the function that is implemented on hardware[6].

As in [7], software implementations have fixed resources and offer limited opportunities for parallelization. A processor executes a program as a sequence of instructions generated by processor compiler tools, which transform an algorithm expressed in C/C++ into assembly language. Even a simple operation, like the addition of two values, results in multiple

978-1-7281-4058-2/19/$31.00 ©2019 IEEE

assembly instructions that must be executed across multiple clock cycles. It takes lower clock cycles when implemented on FPGA which has a high degree of parallelism in algorithm execution.

SDSoC allows implementing functions on hardware where data bus widths can be either 8-bits, 16-bits, 24-bits or 32-bits. Interface buses between FPGA and CPU can be AXI_FIFO, AXI_LITE, AXIDMA_SIMPLE…etc. AXIDMA_SIMPLE is the most efficient bulk transfer engine. However, it consumes a large area. It supports up to 8MB transfers [6].

In this paper, a standalone project was created for each algorithm on Xilinx Zynq ZC702 device, that uses dual-Core ARM Cortex A9 processor and XC7Z020-CLG484 based FPGA as the programming logic [6]. FPGA clock frequency is set to 100MHz. Choosing the functions to be implemented on FPGA is based on the number of function calls to encrypt/decrypt single message, number of required input/output ports for each function, and input/output data types and sizes. HLS pragmas were used in the hardware functions to improve performance as described in [8]. The board setup was set as in [9], [10] to measure software performance.

III. Stream Ciphers Overview

A. ACORN

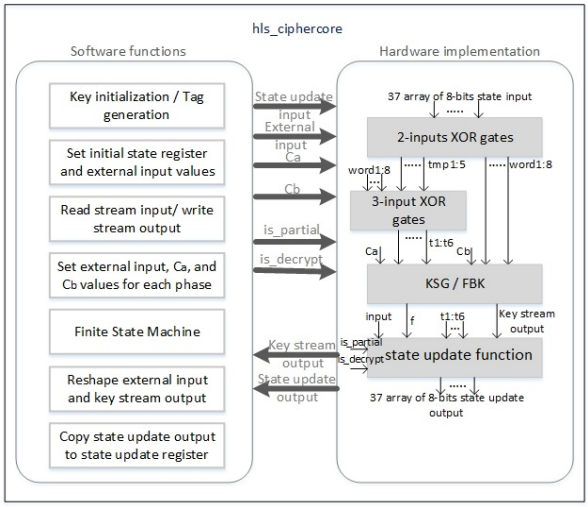

Fig. 1. ACORN IMPLEMENTATION

ACORN is a stream authenticated encryption cipher based on a binary feedback shift register [1] [12]. This paper shows the performance of High speed 32-bit ACORNv3 [11]. The state register of ACORN consists of 293 bits. Its value is updated by applying the following four logic operation functions:

- Non-Linear feedback function: which applies XOR operation on state register output and saves the results on fifteen 32-bit registers.
- KSG128 function: which takes the result of non-linear feedback function to generate keystream new bits (Ks_{31} ... Ks_0) which is used to generate ciphertext.
- FBK128 function: which takes the output of KSG128 function and non-linear feedback function to produce the 32-bits data output.
- State update function: which uses the previous value of state register and output data of the other functions to generate a new state register value.

In this paper, the C++ code was modified to change the variables' data types from arrays of 1-bit elements to be arrays of 8-bits or 32-bits elements to reduce the number of clock cycles. The core functions (non-linear feedback, KSG, FBK, and state update) were combined into one function and implemented on FPGA as shown in "Fig. 1".

B. AEGIS

AEGIS is a dedicated stream cipher with large state size which is updated continuously [1]. This paper shows the performance of AEGIS128-L. AEGIS128-L is based on AES round function, it processes a 16-byte message block with 8 AES round functions [1]. The state registers of AEGIS consist of eight 128-bit sub-registers (S_0, S_1, ..., S_7). Those values are updated by applying 8-parallel Rounds of AES function on state register output and XOR S_0, S_4 with external input to calculate those new values as described in [1] and [13]. The external input can be either IV or Key or Associated Data (AD) or current plaintext or previous ciphertext or input message length. The output of state registers is used to generate keystream output then it is XORed with a plaintext to generate a ciphertext.

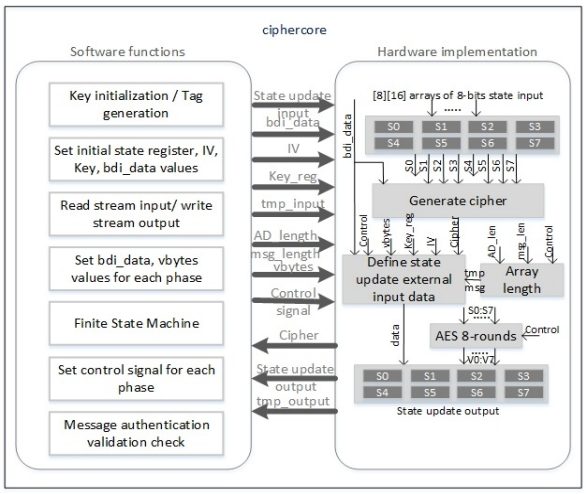

Fig. 2. AEGIS IMPLEMENTATION

The proposed implementation of AEGIS is shown in "Fig. 2". The state registers, cipher generator, the logic operations that define external inputs for each phase, and AES functions were implemented on FPGA, while the Finite State Machine (FSM), key initialization, state registers initializations, and tag generation were implemented on CPU.

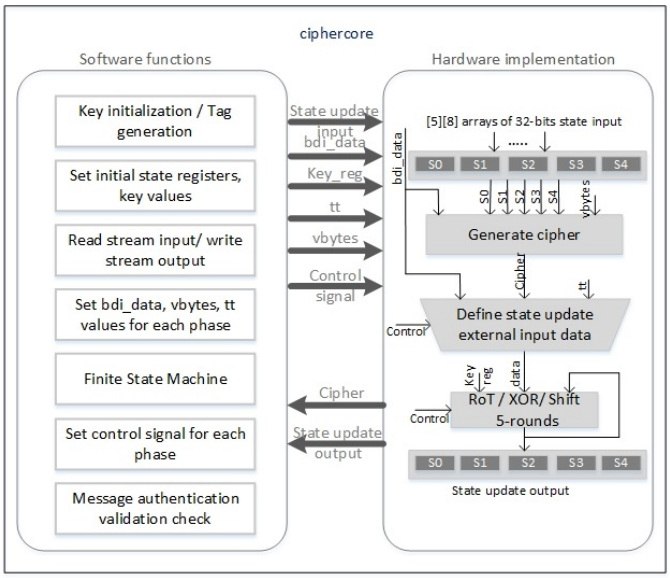

Fig. 3. MORUS IMPLEMENTATION

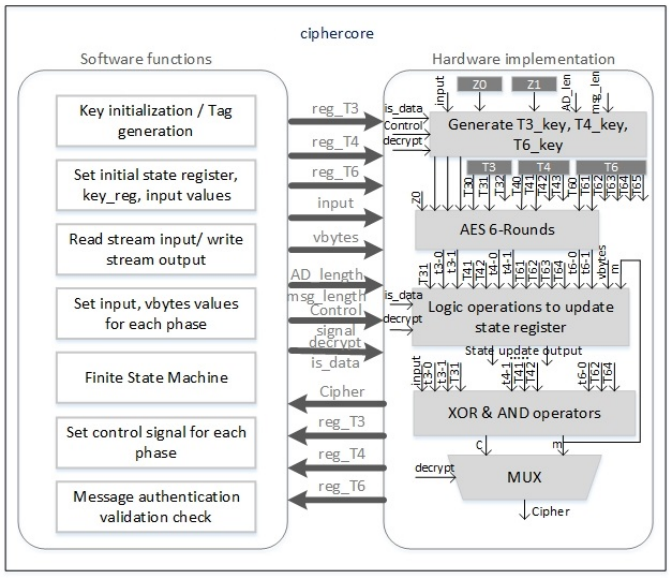

Fig. 4. Tiaoxin IMPLEMENTATION

C. *MORUS*

MORUS is a stream cipher with large state size. It uses a scheme similar to a type 3 Feistel scheme to update its state registers [14]. In this paper, the performance of MORUS-1280-128 is shown. MORUS consists of five 256-bit state registers ($S_{0,0},\ldots,S_{0,4}$), those values are updated by applying 5-rounds of MORUS core function on both external input and state registers output. MORUS core function consists of AND, XOR and rotate operations [12]. The 256-bit register is split into smaller parts of 64-bit registers to apply rotation function on them as described in [14]. Input of MORUS core function can be either input data length or previous ciphertext or plaintext or AD [15]. Keystream new values are generated by applying XOR and shift operations on the state registers output.

The proposed implementation is shown in "Fig. 3", state registers, cipher generator functions, the logic operations that define external inputs for each phase and core function were implemented on FPGA, while the other functions were implemented on CPU.

D. *Tiaoxin*

Tiaoxin is a dedicated word-based non-linear feedback shift register stream cipher [1]. This paper shows the performance of Tiaoxin-346v2. The state registers of Tiaoxin consist of thirteen 128-bit state registers that are divided into 3 groups namely: T3, T4, and T6 that consist of 3, 4, and 6 registers, respectively [16]. The first two registers of each group are updated by applying 6-parallel rounds of AES function on the first and last register of each group [17]. Then they are XORed with external input. The other registers values are updated by shifting the previous register output. External input can be either constant values (Z0, Z1) or Key or IV or AD or plaintext or input data length. The new keystream value is produced by applying XOR and AND operations on the state register output.

The proposed implementation of Tiaoxin is shown in "Fig. 4". Key generator, AES function, state update function, state registers, and cipher generator were implemented on FPGA, while the other functions were implemented on CPU.

IV. RESULTS

To evaluate the performance of the proposed implementations on SDSoC, 25 input messages were applied with different data length to each algorithm, 13 of them for encryption operation, and 12 of them for decryption operation. These messages were defined in the C++ test-benches that are available on ATHENa website [5]. Performance is measured in terms of the number of CPU clock cycles to encrypt or decrypt those messages and in terms of area utilization of the FPGA resources.

The function that is proposed to be implemented on hardware is called *cipher_hw* function and the function of the whole algorithm is called *ciphercore* function. Table I shows the performance of *cipher_hw* in terms of the equivalent number of CPU clock cycles when this function is implemented on FPGA. It also shows the speed up factor which is defined as the ratio between software implementation CPU clock cycles to hardware implementation equivalent CPU clock cycles and number of function calls that were used to encrypt or decrypt the defined messages.

Table I: Co-Design Performance for *cipher_hw* Function

Algorithm	CPU cycles of proposed implementation	Speed up factor	# of HW function calls
ACORN	8074	30.3	2634
AEGIS	10331	109.3	530
MORUS	8037	45.4	790
Tiaoxin	11166	84.2	998

Table II shows the performance of the whole algorithm (*ciphercore* function) which include *cipher_hw* function and the other functions that will be implemented on software. Performance of *ciphercore* function is measured in terms of number of the CPU clock cycles when *cipher_hw* function was implemented on hardware and the remaining functions implemented on software, number of CPU clock cycles for co-design implementation that needed to encrypt or decrypt one byte, and the speed up factor which is defined as the ratio between CPU clock cycles of software only implementation without moving any function from FPGA to CPU clock cycles of the co-design implementation. Table II shows also the number of bytes of the defined input messages for each algorithm.

Table II: Co-Design Performance for *ciphercore* Function

Algorithm	CPU cycles of proposed implementation	Speed up factor	# of bytes	Cycles per bytes
ACORN	103977664	7	1143	90969
AEGIS	28571250	21.7	2626	10880
MORUS	36421980	8.7	2943	12375
Tiaoxin	75880446	13.2	3205	23675

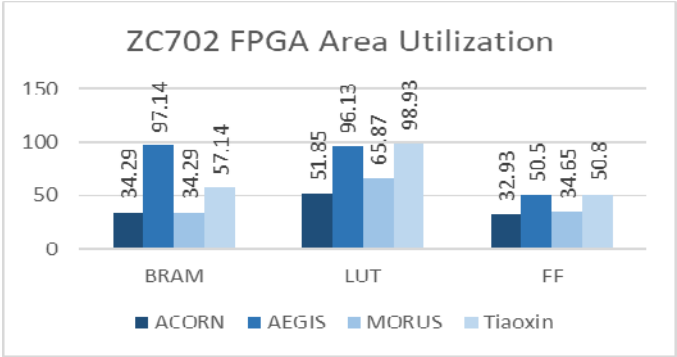

Fig. 5. FPGA UTILIZATION

"Fig. 5" shows the FPGA area utilization that was used to implement *cipher_hw* function on hardware for each algorithm in terms of LUT, FF and BRAM utilization. It includes the area of resources that was needed for *cipher_hw* function implementation and the area of AXIDMA interface buffers.

Zynq ZC702 Evaluation kit's FPGA is implemented based on Artix®-7 programmable logic [18], while each slice consists of 4 LUT and 8 FFs [19]. The number of slices that were used to implement interface buffers can be estimated by dividing number of LUT by 4 and dividing number of FF by 8 and getting the maximum value of those two values. However, the number of Slices that were required to implement *cipher_hw* function could not be estimated directly by the previously mentioned method as the array reshaping and loop unrolling pragmas were used on *cipher_hw* function to increase parallelism level and improve performance. This leads to increase the number of the required slices as the slices were partially utilized by the synthesis tool for parallel design in order to decrease the wires routing and satisfy the timing constraints as described in [20]. The actual number of slices that were used by *cipher_hw* function are measured using Vivado HLS tool after exporting this function to RTL.

Table III: Figure of Merit Results

Algorithm	Slice Utilization [%]	FoM
ACORN	49.6	15472
AEGIS	79.6	9005
MORUS	55.4	2661
Tiaoxin	81.2	11688

A new metric called Figure of Merit (FoM) was defined to represent the performance in terms of number of clock cycles, BRAM utilization and the FPGA slices utilization as shown in (1). Table III shows the FOM results and the area utilization in terms of number of slices for each algorithm. From these results, the MORUS algorithm has the best performance while the ACORN algorithm has the worst performance.

*FoM = Clock Cycles per Byte * BRAM Util. * FPGA Slice Util.* (1)

V. Design Insights

- Using SW-HW co-design implementation improves the number of CPU clock cycles for all algorithms. MORUS has the best performance in this proposal. However, the number of clock cycles for MORUS is slightly higher than AEGIS, but it has a lower area.
- ACORN is a lightweight AE algorithm, it has a very small area in hardware. ACORN has many control signals, those will consume many CPU clock cycles to update these values on software implementation. Also, for a co-design implementation, moving functions that are used to define control signals for each phase needs to use more input and output interfaces that will consume extra area utilization and extra clock cycles to move variables between CPU and FPGA. So, it is recommended to implement ACORN on hardware only rather than using software or co-design implementations.
- AEGIS is composed of eight different AES round functions. AES algorithm consists of *Sub Bytes*, *Shift Rows* and *Mix Columns* functions. *Sub Bytes* function can be implemented using BRAM or CLB. In this paper, it was implemented using BRAM. That is why it has the highest BRAM utilization. BRAM utilization will be reduced to 51.7% if AES *Sub Bytes* function is implemented using CLB.
- Tiaoxin is similar to AEGIS. Both of them consist of AES, XOR and shift operations for encryption. Tiaoxin has six similar AES functions, while *Sub Bytes*

function is implemented using BRAM. If *Sub Bytes* function is implemented using CLB, BRAM utilization will be reduced to 51.7% like AEGIS.

- AEGIS is faster than Tiaoxin by a factor of 2. Both of them have similar FPGA slice utilization and the same BRAM utilization if *Sub Bytes* function was implemented using CLB. So, it is recommended to use AEGIS algorithm instead of Tiaoxin.

VI. Conclusion

This paper showed that using SW-HW co-design improves the speed of the software implementation for ACORN, AEGIS, MORUS and Tiaoxin AE stream cipher algorithms. In this paper, the AEGIS algorithm had minimum cycles per byte. However, it consumed large area, while MORUS's number of CPU clock cycles per byte was slightly higher than AEGIS but it consumed 65% of AEGIS area. ACORN had the worst performance for software implementation, but the best area utilization.

Acknowledgment

This work was partially funded by ONE Lab at Zewail City of Science and Technology and at Cairo University, NTRA, ITIDA, and ASRT.

References

[1] Md Iftekhar Salam, "Analysis of Authenticated Encryption Based on Stream Cipher," PhD. Thesis, University of Technology, Australia 2018.

[2] N. Samir, Y. Gamal, A. N. El-Zeiny, O. Mahmoud, A. Shawky, A. Saeed, and H. Mostafa, "Energy-Adaptive Lightweight Hardware Security Module Using Partial Dynamic Reconfiguration for Energy Limited Internet of Things Applications", IEEE International Symposium on Circuits and Systems (ISCAS 2019), Sapporo, Japan, pp. 1-4, 2019.

[3] H.Wu, B.Preneel, "AEGIS: A Fast-Authenticated Encryption Algorithm (v1.1)," competitions.cr.yp.to, Sept. 15, 2016 [Online]. Available: https://competitions.cr.yp.to/round3/aegisv11.pdf. [Accessed May 20, 2019].

[4] P.Sarkar, "Modes of Operations for Encryption and Authentication Using Stream Ciphers Supporting an Initialization Vector," eprint.iacr.org, June 6,2011 [Online]. Available: https://eprint.iacr.org/2011/299.pdf . [Accessed May 10, 2019].

[5] ATHENa, *CERG* "Source Code for CAESAR Round 3 Candidates (HLS-ready C and automatically generated RTL VHDL, High-Speed Implementations)," cryptography.gmu.edu, Dec. 2017 [Online]. Available: https://cryptography.gmu.edu/athena/index.php?id=CAESAR_source_codes . [Accessed April 10, 2019] .

[6] Xilinx Inc., "SDSoC Environment User Guide," UG1027, June 20, 2017.

[7] Xilinx Inc., "SDSoC Programmers Guide," UG1278, Jan. 24, 2019.

[8] Xilinx Inc., "Vivado Design Suite User Guide, High-Level Synthesis," UG902, Dec. 20, 2018.

[9] Xilinx Inc., "SDSoC Environment Debugging Guide," UG1282, Jan. 24, 2019.

[10] Xilinx Inc., "ZC702 Evaluation Board for the Zynq-7000 XC7Z020 SoC User Guide," UG850, March 27, 2019.

[11] H.Wu, "ACORN: A Lightweight Authenticated Cipher (v3)," competitions.cr.yp.to, Sept. 15, 2016 [Online]. Available: https://competitions.cr.yp.to/round3/acornv3.pdf,. [Accessed May 10, 2019].

[12] S. Sharaf, and H. Mostafa, "A Study of Authentication Encryption Algorithms (POET, Deoxys, AEZ, MORUS, ACORN, AEGIS, AES-GCM) For Automotive Security", IEEE International Conference on Microelectronics (ICM 2018), Sousse, Tunisia, pp. 315-318, 2018.

[13] S. Soliman, M. A. Jaela, A. M. Abotaleb, Y. Hassan, M. A. Abdelghany, A. T. Abdel-Hamid, K. N. Salama, and H. Mostafa, "FPGA Implementation of Dynamically Reconfigurable IoT Security Module Using Algorithm Hopping", Elsevier Integration VLSI Journal, vol. 68, pp. 108-121, 2019.

[14] Rajesh Kumar Pal, "Implementation and Evaluation of Authenticated Encryption Algorithms on JAVA Card Platform," M. S. Thesis, Masaryk University, Brno 2017.

[15] H.Wu and T.Huang "The Authenticated Cipher MORUS (v2)," competitions.cr.yp.to, Sept. 19,2016[Online]. Available: https://competitions.cr.yp.to/round3/morusv2.pdf [Accessed May 20, 2019].

[16] A.Abbas, H.Mostafa, and A.N.Mohieldin, "Low Area and Low Power Implementation for CAESAR Authenticated Cipher," New Generation of CAS, Nov. 2018.

[17] Ivica Nikolic, "Tiaoxin-346 VERSION 2.1," competitions.cr.yp.to, Sept. 19,2016[Online]. Available: https://competitions.cr.yp.to/round3/tiaoxinv21.pdf. [Accessed May 20, 2019].

[18] Xilinx Inc., "Zynq-7000 SoC Product Advantages," *Xilinx Inc.* [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html. [Accessed: April 15, 2019].

[19] Xilinx Inc., "7 Series FPGAs Configurable Logic Block User Guide," UG474, Sept. 27, 2016.

[20] K.M.A.Ali "Parallel reconfigurable hardware architectures for video processing applications" pdfs.semanticscholar.org, May 14, 2018 [Online]. Available: https://pdfs.semanticscholar.org/7d58/5fb7e0d54bb230144b85983f249a63c6cb45.pdf . [Accessed April 2, 2019].