Efficient Parallel-Pipelined GHASH for Message Authentication

Karim M. Abdellatif, R. Chotin-Avot, and H. Mehrez
LIP6-SoC Laboratory, University of Paris VI, France
{karim.abdellatif, Roselyne.Chotin-Avot, Habib.Mehrez}@lip6.fr

***Abstract*—AES/GCM is a common mode for authenticated encryption. One of its components is Galois HASH (GHASH) which achieves the authentication task. In this work, we present an efficient key independent hardware implementation for parallel-pipelined GHASH. Karatsuba Ofman Algorithm (KOA) is used for Galois Field (GF) multiplication. Unlike previous parallel hardware architectures based on KOA, we use only one reduction array for all parallel KOA multipliers. Therefore, an area optimized design is achieved. In addition, pipelined KOA is adopted to get higher clock frequency. 4-Parallel pipelined GHASH is evaluated using Xilinx Virtex5. It occupies 7.128k Slices and achieves 113.8 Gbps as an authentication throughput. Higher hardware efficiency (throughput/slice) in comparison with prior art (Key independent GHASH) is achieved.**

I. Introduction

Authenticated Encryption (AE) is a block cipher mode of operation which simultaneously provides confidentiality, integrity and authenticity assurances on the data. The Galois/Counter Mode (GCM) [1] was considered as a new mode of operation of Advanced Encryption Standard (AES) since 2007. It can provide not only high speed authenticated encryption but also protection against bit-flipping attacks.

GCM [1] is an authenticated encryption mode that generates cipher text and authentication tag simultaneously. It can be implemented in hardware to achieve high speeds with low cost and low latency. Software implementations can achieve excellent performance by using table-driven field operations. Two main components in AES-GCM are AES engine and GHASH module. Because of the inherent computation feedback in multiplication over GF(2^{128}), the performance of the whole AES-GCM algorithm is limited by the structure of Finite Field Multiplier (FFM) in GHASH core.

Mathematically, a cryptographic GHASH function is a construct that performs universal hashing over a binary Galois field to generate a Message Authentication Code (MAC). GHASH has two distinct parameters, namely an input message and a secret key known only to the message sender and the corresponding receivers.

In order to support high speed applications up to 100 Gbps like IEEE 802.3ba Ethernet standard, 4-parallel architecture is used [2]. Unlike previous designs, this work introduces an area optimized method for parallel GHASHs based on independent key. Parallel multipliers are used with one reduction array to obtain an efficient area architecture compared to previous work. Pipelined KOA is also used in order to keep up with previous high speed designs.

This paper is organized as follows: Background on GHASH is provided in Section II. Previous Architectures for GHASH are presented in Section III. GF(2^{128}) Multiplier using KOA is discussed in Section IV. Efficient architecture for parallel GHASH is discussed in Section V. Implementation details and performance comparison are discussed in Section VI. Section VII concludes this work.

II. GHASH Function

The authentication mechanism within GCM is based on a hash function, called GHASH, that features multiplication by the hash subkey, within a binary Galois field. The hash subkey, denoted H, is generated by applying the block cipher to the zero block. The GHASH is a keyed hash function but not, on its own, a cryptographic hash function. It is based on GF(2^{128}) multiplier with irreducible polynomial $F(x) = x^{128} + x^7 + x^2 + x + 1$ as described in [1]. As shown in Fig. 1, the GHASH architecture accepts 5 inputs:

- A 128-bit hash key H derived from a symmetric cryptographic key K.
- An M-bit message which can be divided into *n* 128-bit blocks $M_1 - M_n$ and the last message block M_n is padded with zeros to create a 128-bit word.
- An optional 128-bit Additional Authenticated Data (AAD) is authenticated but not encrypted.
- A 128-bit LEN value which expresses the word lengths of AAD and the message M.
- A 128-bit cryptographic pad value (PAD) which ciphers the function output TAG to generate the message authentication code (MAC).

The resulting 128-bit is expressed as:

$$\begin{cases} H = E(K, 0^{128}) \\ X_0 = GHASH(H, AAD) \\ X_i = GHASH(H, M_i \oplus X_{i-1}) \\ LEN = length(AAD)_{64} \parallel length(M)_{64} \\ TAG = GHASH(H, X_n \oplus LEN) \\ MAC = PAD \oplus TAG \end{cases} \quad (1)$$

978-1-4673-2921-7/12/$31.00 © 2012 IEEE

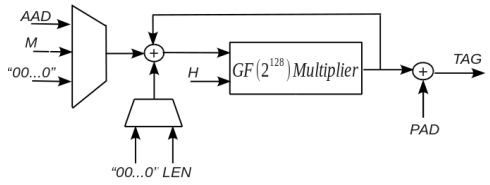

Figure 1. GHASH architecture

III. Previous Architectures for GHASH

The hash function of GCM repeats multiply-add operations according to (2). Satoh *et al.* [3] proposed 4-parallel hardware architecture based on four 128-bit GF multipliers using (2) as shown in Fig. 2. This method [3] is example operation for 10 data blocks $M_1 \sim M_{10}$. The blocks from M_1 to M_{10} are processed in 5 clock cycles. An optimized method for single GHASH using KOA was proposed in [4]. In order to increase the speed of single GHASH, pipelined architecture based on KOA was also achieved by [5]. For fixed key applications, Crenne *et al.* [6] limited the GHASH logic utilization by specializing the hardware implementation on a per-key basis.

$$\begin{aligned} X_i &= (X_{i-1} \oplus M_i)H \\ &= (...(((M_1H \oplus M_2)H \oplus M_3)H \oplus M_4)H...)H \\ &= (((M_1H^4 \oplus M_5)H^4 \oplus M_9)H^4 \oplus ...)H^4 \\ &\oplus(((M_2H^4 \oplus M_6)H^4 \oplus M_{10})H^4 \oplus ...)H^3 \\ &\oplus(((M_3H^4 \oplus M_7)H^4 \oplus M_{11})H^4 \oplus ...)H^2 \\ &\oplus(((M_4H^4 \oplus M_8)H^4 \oplus M_{12})H^4 \oplus ...)H \end{aligned} \tag{2}$$

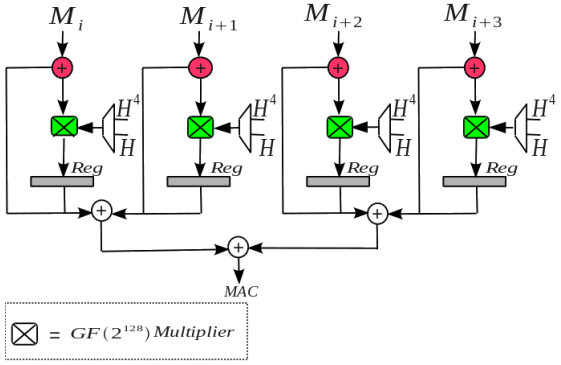

Figure 2. 4-Parallel GHASH [3]

IV. GF(2^{128}) Multiplier using KOA

KOA is used to reduce the complexity of the multiplication process. The single step KOA algorithm splits two m bit inputs *A* and *B* into four terms A_h, A_l, B_h, B_l which are m/2 bit terms. The 1-step iteration of KOA shown in Fig. 3

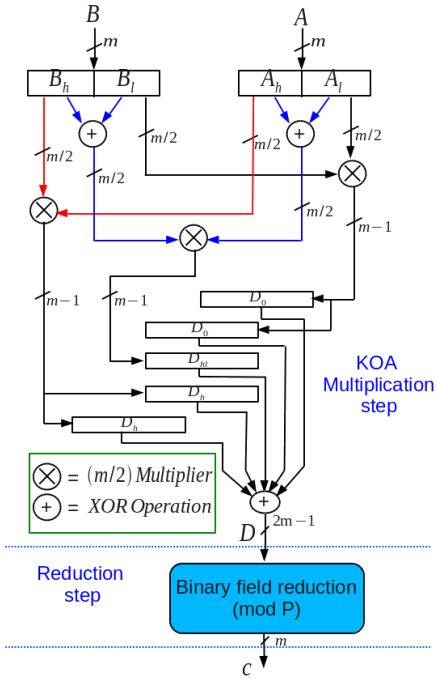

Figure 3. Polynomial Multiplication using KOA

can be described as:

$$\begin{cases} D_O &= A_lB_l \\ D_{hl} &= (A_h \oplus A_l)(B_h \oplus B_l) \\ D_h &= A_hB_h \\ D &= D_hX^m \oplus X^{m/2}(D_h \oplus D_{hl} \oplus D_l) \oplus D_l \end{cases} \tag{3}$$

KOA is commonly used in polynomial multiplication GF(2^{128}) [5] as the multiplication of large vectors can be modified into the multiplication of small vectors. After the multiplication stage is processed using KOA, the binary field reduction step is used to convert the length of the vector from $2m-1$ to m as shown in (4).

$$C(x) = \quad D \quad mod \quad P(x) \tag{4}$$

where P(x) is the field polynomial used for the multiplication operation.

$$P(x) = x^{128} + x^7 + x^2 + x + 1 \tag{5}$$

In order to support high speed applications up to 100 Gbps, parallel architectures of GF(2^{128}) are used. The next section describes an efficient hardware for parallel GF(2^{128}) multipliers used in GHASH.

V. New Hardware Implementation for Parallel GHASH

Our hardware implementation is based on parallel GHASH architecture. The goal is to get an area optimized architecture for the parallel scheme of GHASH without any effect on the operating frequency. We must keep up

Table I
FLOW CONTROL OF 4-PARALLEL GHASH USING PIPELINING APPROACH

clck	M_{i+1}/H^k	M_{i+2}/H^k	M_{i+3}/H^k	M_{i+4}/H^k	Q/Tag
1	M_1/H^{16}	M_2/H^{15}	M_3/H^{14}	M_4/H^{13}	-
2	M_5/H^{12}	M_6/H^{11}	M_7/H^{10}	M_8/H^9	-
3	M_9/H^8	M_{10}/H^7	M_{11}/H^6	M_{12}/H^5	-
4	M_{13}/H^4	M_{14}/H^3	M_{15}/H^2	M_{16}/H	Q_1
5	-	-	-	-	Q_2
6	-	-	-	-	Q_3
7	-	-	-	-	Q_4
8	-	-	-	-	Tag

with the previous architectures which support high speed applications up to 100 Gbps. The idea presented in this work is to use only one reduction array for all KOA multipliers. According to (6), N parallel multipliers can be implemented using only one reduction array unlike previous work [2] and [7] which presented 4-parallel multipliers with four reduction arrays. Therefore, the consumed logic of the overall design will be decreased because of using one reduction array for all multipliers rather than several reduction arrays equal to the number of the parallel multipliers. N parallel GHASH is shown in Fig. 4.

$$
\begin{aligned}
X_i &= (M_i \oplus X_{i-1}) \times H \\
&= (M_i \times H) \oplus (X_{i-1} \times H) \\
&= (M_i \times H) \oplus [(M_{i-1} \oplus X_{i-2}) \times H^2] \\
&= (M_i \times H) \oplus (M_{i-1} \times H^2) \oplus [(M_{i-2} \oplus X_{i-3}) \times H^3] \\
&= (M_i \times H) \oplus (M_{i-1} \times H^2) \oplus (M_{i-2} \times H^3) \\
&\oplus [(M_{i-3} \oplus X_{i-4}) \times H^4] \\
&= ((M_i \times H) \oplus (M_{i-1} \times H^2) \oplus (M_{i-2} \times H^3) \\
&\oplus (M_{i-3} \times H^4) .... \oplus [(M_{N-1} \oplus X_{N-2}) \times H^{N+1}]) mod P
\end{aligned}
\tag{6}
$$

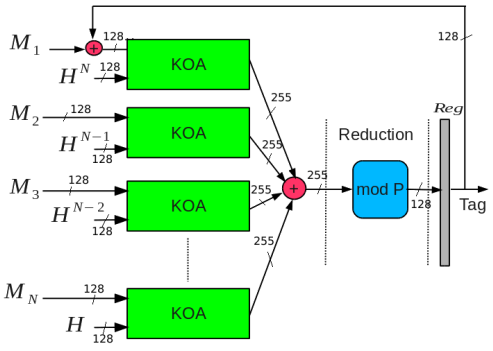

Figure 4. Parallel GHASH Implementation

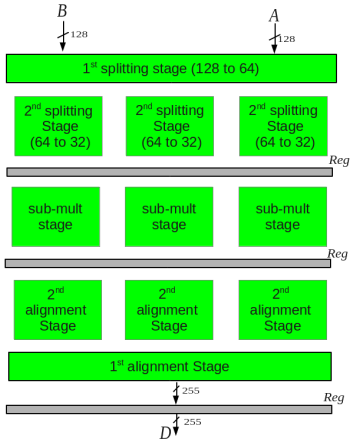

Figure 5. Pipelined KOA

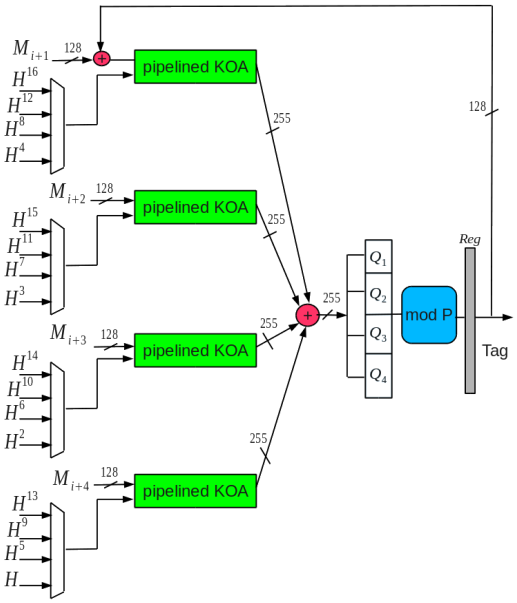

Figure 6. 4-parallel pipelined GHASH

KOA is used to convert the complexity of 128-bit multiplier into a 32-bit multiplier. In order to speed up our architecture we used Pipelined KOA. The used KOA consists of three pipelined stages in order to reduce the critical delay as shown in Fig. 5. Fig. 6 shows the hardware architecture of the proposed 4-parallel GHASH using Pipelined KOA. We concentrate on 4-parallel GHASH because it is commonly used in [2]. The overall architecture shown in Fig. 6 contains four pipelined stages because of the last register. When the operands M_{i+1}, M_{i+2}, M_{i+3}, and M_{i+4} are loaded into the 4-parallel architecture, the used multiplexers select the suitable value of H^k to be multiplied with the current operand.

Table II
HARDWARE COMPARISON

Design	Device	Architecture	PAR	key	Slices	Frequency Mhz	Throughput Gbit/s	Thr./Slice Mbps/slice
This work	xc5vlx220	4-parallel GHASH	•	•	7127	222.2	113.8	15.96
This work	xc4vlx25	4-parallel GHASH	•	•	16257	200	102.4	6.2
wang*et al*. [7]	xc5vlx85t	4-parallel GHASH	○	•	10943	240.3	123.1	11.2
Huo *et al*. [8]	xc5vlx30	Single GHASH	○	•	2992	240.2	30.8	10.3
Lu *et al*. [9]	xc5vlx50t	Single GHASH	○	•	3175	120.2	15.4	4.8
Chen *et al*.[10]	xc4vlx60	Single GHASH	•	•	10756	312.5	40.0	3.7
Crenne *et al*.[6]	xc4vlx25	4-parallel GHASH	•	○	6182	222.2	113.8	18.4

An example of data flow control for the proposed architecture is shown in Table I, where $M_1 ... M_{16}$ is the input sequence and "-" denotes "don't care". At the beginning, all values from H to H^{16} are calculated and stored. The final result of the tag is generated at 8^{th} clock.

VI. PERFORMANCE EVALUATION ON FPGA

The module of 4-parallel GHASH using Pipelined KOA (Fig. 6) was coded using VHDL and targeted to Virtex5(xc5vlx220). ModelSim 6.5c was used for simulation. Xilinx Synthesize Technology (XST) is used to perform the synthesize and ISE9.2 was adopted to run the Place And Route (PAR). Table II shows the comparison of the proposed hardware architecture with prior art. Some of the Previous work like [9], [8], and [7] reported their results before PAR as shown in PAR column in Table II, an estimation for these results (number of occupied slices) was taken from [6]. Key column indicates the flexibility situation of the key as [6] designed a fixed key GHASH in which there is no flexibility to change the key. Note the filled dots in Key and PAR columns.

The proposed 4-parallel pipelined GHASH operates on 222.2 MHz, reaches the throughput to 113.8 Gbit/s and consumes 7127 Slices. Also, there is a flexibility to change the key. Wang *et al*. [7] designed 4-parallel GF(2^{128}) multiplier using Mastrovito's method, their implementation occupies 10943 Slices (more than ours by 34.87 %) and gives a throughput of 123.1 Gbit/s (higher than ours by 7.5%). Chen *et al*. [10] designed a single pipelined GF(2^{128}) multiplier based on modified Mastrovito's method by [11], their implementation occupies 10756 Slices (smaller than ours by 51%) and gives a throughput of 40 Gbit/s (smaller than ours by 184 %) but this work is a single architecture not parallel. Zhou et *al*. [5] implemented single AES/GCM using pipelined KOA with 4628 slices and a throughput of 41.5 Gbit/s. Key dependent GHASH was proposed by Crenne *et al*. [6], they used constant key specialization (key dependent) and implemented their GHASH with 6182 Slices and a throughput of 113.8 Gbit/s but there is no flexibility to change the key as the FPGA must be reconfigured to support key change.

Our proposed 4-parallel GHASH module shows improved throughput per area compared with prior designs (key independent). Also, it facilitates the use of parallel multipliers for AES/GCM as there is only one reduction block for all parallel multipliers which decreases the consumed area. Furthermore, it is possible to change the key used unlike [6].

VII. CONCLUSION

In this work, a new optimized implementation for parallel GHASH is presented. For all parallel multipliers, there is only one reduction array to generate the desired Tag. Using one reduction block for all parallel multipliers decreases the consumed area. Also, Pipelined KOA is used to increase the throughput. The presented 4-parallel GHASH core reaches the throughput of 113.8 Gbit/s with 7128 Slices on Virtex5.

REFERENCES

[1] D. McGrew and J. Viega, "The Security and Performance of The Galois/Counter Mode (GCM) of Operation," *Progress in Cryptology-INDOCRYPT*, pp. 377–413, 2005.

[2] L. Henzen and W. Fichtner, "FPGA Parallel-Pipelined AES-GCM Core for 100G Ethernet Applications," in *Proceedings of the ESSCIRC*, 2010, pp. 202–205.

[3] A. Satoh, T. Sugawara, and T. Aoki, "High-Speed Pipelined Hardware Architecture for Galois Counter Mode," *Information Security*, pp. 118–129, 2007.

[4] G. Zhou, H. Michalik, and L. Hinsenkamp, "Efficient and High-throughput Implementations of AES-GCM on FPGAs," in *International Conference on Field-Programmable Technology, ICFPT, 2007*, pp. 185–192.

[5] G. Zhou and H. Michalik, "Improving Throughput of AES-GCM with Pipelined Karatsuba Multipliers on FPGAs," in *Reconfigurable Computing: Architectures, Tools and Applications*, 2009, pp. 193–203.

[6] J. Crenne, P. Cotret, G. Gogniat, R. Tessier, and J. Diguet, "Efficient Key-Dependent Message Authentication in Reconfigurable Hardware," in *International Conference on Field-Programmable Technology (ICFPT)*, 2011, pp. 1–6.

[7] J. Wang, G. Shou, Y. Hu, and Z. Guo, "High-Speed Architectures for GHASH Based on Efficient Bit-Parallel Multipliers," in *IEEE International Conference on Wireless Communications, Networking and Information Security (WCNIS)*, 2010, pp. 582–586.

[8] J. Huo, G. Shou, Y. Hu, and Z. Guo, "The Design and FPGA Implementation of GF (2^ 128) Multiplier for GHASH," in *International Conference on Networks Security, Wireless Communications and Trusted Computing, NSWCTC'09*, vol. 1, 2009, pp. 554–557.

[9] Y. Lu, G. Shou, Y. Hu, and Z. Guo, "The Research and Efficient FPGA Implementation of GHASH Core for GMAC," in *International Conference on E-Business and Information System Security, EBISS'09*, 2009, pp. 1–5.

[10] W. H. T. Chen and Z. Liu, "Design and Efficient FPGA Implementation of GHASH Core for AES-GCM," in *International Conference on Computational Intelligence and Software Engineering*, 2010, pp. 1–4.

[11] A. Reyhani-Masoleh and M. Hasan, "Low Complexity Bit Parallel architectures for Polynomial Basis Multiplication Over GF (2m)," *IEEE Transactions on Computers*, vol. 53, no. 8, pp. 945–959, 2004.