High performance AES-GCM implementation based on efficient AES and FR-KOA multiplier

Yong Zhang\textsuperscript{1a)}, Ning Wu\textsuperscript{1b)}, Fang Zhou\textsuperscript{1}, Xiaoqiang Zhang\textsuperscript{2}, and Jinbao Zhang\textsuperscript{1)

\textsuperscript{1} College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
\textsuperscript{2} College of Electrical Engineering, Anhui Polytechnic University, Wuhu 241000, China

\texttt{a)} nuaa.zyfy@163.com
\texttt{b)} wunee@nuaa.edu.cn

Abstract: This paper proposes a FPGA based efficient implementation of AES-GCM for wireless applications. For AES engine, we apply the DACSE algorithm to achieve a compact S-box. A new pipeline strategy is present to improve the throughput of AES engine without bring in extra resource consumption. For GHASH core, FR-KOA algorithm is present to implement a finite field multiplier (FFM). In addition, a 6-stage pipeline strategy is used to improve the FFM throughput. The proposed FR-KOA FFM can match the high-efficiency AES we designed to achieve the highly efficient AES-GCM. FPGA implementation on Xilinx FPGA, Virtex5 xc5vlx85 yielded a throughput value of 48.8 Gbps covering area of 6482 slices. The efficiency of our implementation is 7.54 Mbps/Slice which is higher than the previous works.

Keywords: AES-GCM, highly efficient, FR-KOA, FPGA

Classification: Integrated circuits

References

[1] D. A. McGrew and J. Viega: The Galois/Counter Mode of Operation (GCM). Updated submission to NIST, Modes of Operation Process (May 2005).
[2] NIST: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D (November 2007).
[3] IEEE: 802.1AE - Media Access Control (MAC) Security (2006).
[4] INCITS: Fiber Channel Security Protocols, REC 1.74 (2006).
[5] L. Henzen and W. Fichtner: “FPGA parallel-pipelined AES-GCM core for 100G Ethernet applications,” Proc. IEEE ESSCIRC \textbf{1930} (2010) 202 (DOI: 10.1109/ESSCIR.2010.5619894).
[6] K. M. Abdellatif, \textit{et al.}: “High speed authenticated encryption for slow changing key applications using reconfigurable devices,” IEEE Wireless Days (2013) 1 (DOI: 10.1109/WD.2013.6686460).
1 Introduction

In 2007, National Institute of Standards and Technology (NIST) officially announced AES-GCM [1] as a working mode for AES [2]. AES-GCM consists of two main parts: an AES engine which occupies most of the AES-GCM circuit area and a GHASH core whose performance is determined by the finite field multiplier (FFM) over $GF(2^{128})$. AES-GCM can provide both encryption and authentication compared to other AES modes. It has been widely used in wireless, magnetic recording systems and network protocols such as the new LAN security standard WLAN 802.1AE [3] and Fiber Channel Security Protocols [4]. At present, most works aim to optimize the throughput of the AES-GCM, however, many of the AES-GCM applications are high-speed and resource-constrained. So, it is essential to design a high efficiency circuit to broaden the scope of AES-GCM application.

The design and optimization of algorithm have been studied to realize AES-GCM for high speed or low area. Luca Henzen et al. designed an AES-GCM circuit with a pipeline AES and a 2-step KOA FFM in 4-parallel GHASH. It reached the throughput of 48.8 Gbps with 18505 slices resource overhead [5]. The throughput is high but it did not increase four times as much as the area, causing low efficiency. Karim M et al. designed the AES-GCM with the key-synthesized method, it cost less circuit area but the AES and FFM were not pipelined, resulting in low throughput and low efficiency [6]. Later, Karim M et al. designed a pipelined AES-GCM architecture that can eliminate feedback, achieving the throughput of...
31.36 Gbps occupying 7475 slices [7]. However, this architecture can only work with data packets smaller than 64. In addition, the implementations of FFM in GHASH have been discussed to make it match the throughput of AES engine [8, 9, 10, 11]. In this regard, our study is mainly focused on the efficient implementation of AES-GCM with low area and high throughput to make it applicable to high-speed and resource-constrained occasions.

In AES engine, delay-aware common sub-expression elimination (DACSE) algorithm [12] is applied to reduce hardware consumption and the pipeline is inserted at the appropriate place to improve the throughput without bringing in extra hardware overhead by analyzing the delay of each arithmetic unit to achieve a high-efficiency AES. In GHASH core, the FFM is designed based on the proposed fast reduction-Karatsuba Ofman algorithm (FR-KOA) to implement a high-efficiency carry-less FFM. Based on the high-efficiency FFM, a pipeline strategy is applied to form a 6-stage pipeline FFM improving the throughput.

2 Proposed high efficiency AES

AES includes the key expansion and round transformation. Key expansion provides round key for round transformation containing SubBytes (SB), ShiftRows (SR), MixColumns (MC) and AddRoundKey (ARK). SubBytes which also can be called S-box is the only nonlinear unit in AES and it takes the most delay and hardware resource in AES. It is significant to design a compact and high throughput S-box.

In order to reduce the area, the S-box is implemented by the composite field technology over $GF(2^4)^2$. S-box over $GF(2^4)^2$ includes five arithmetic units: Mapping, Multiplication (Mul), Square-Multiplying-Constant (SquMulC), Inverse (Inv) and Inverse-Mapping (InvMapping). DACSE algorithm is used to optimize the units reducing the redundant hardware consumption. The hardware consumption of each unit is shown in Table I.

| Arithmetic | Unoptimized Overhead | Optimized Overhead |
|------------|----------------------|--------------------|
|            | XOR | AND | XOR | AND |
| Mapping    | 22  | 0   | 11  | 0   |
| Mul        | 21  | 25  | 15  | 16  |
| SquMulC    | 6   | 0   | 0   | 0   |
| Inv        | 21  | 27  | 16  | 10  |
| InvMapping | 20  | 0   | 13  | 0   |
| Total      | 90  | 52  | 55  | 26  |

The DACSE algorithm can reduce the redundant hardware overhead by 38.89% of XOR gates and 50% of AND gates. Based on the compact S-box, we analyze the delay in each unit, then, pipelines are inserted into the proper positions to improve the throughput without bringing in unnecessary hardware consumption. The delay of each optimized arithmetic is shown in Table II.
Based on the delay shown in Table II, we insert pipeline in S-box. The pipelined S-box shown in Fig. 1 guarantees that the delay of each pipeline stage is nearly equivalent, so, it will not bring in the unnecessary hardware overhead.

Based on the delay shown in Table II, we insert pipeline in S-box. The pipelined S-box shown in Fig. 1 guarantees that the delay of each pipeline stage is nearly equivalent, so, it will not bring in the unnecessary hardware overhead.

| Arithmetic | Mapping | XOR | Mul | SquMulC | Inv | InvMapping |
|------------|---------|-----|-----|---------|-----|------------|
| Delay      | $3T_{Add}$ | $1T_{Add}$ | $3T_{Add} + 1T_{AND}$ | $0$ | $3T_{Add} + 2T_{AND}$ | $4T_{Add}$ |

Table II. Delay of each optimized unit

![Fig. 1. Proposed pipelined S-box](image)

**Fig. 1.** Proposed pipelined S-box

![Fig. 2. Proposed pipelined round transformation](image)

**Fig. 2.** Proposed pipelined round transformation

Based on the pipelined S-box, we design the pipelined high-efficiency AES circuit, whose round transformation is demonstrated in Fig. 2. We insert pipeline among each round transformation forming 10 extern pipeline stages. Inside the round transformation, we insert a pipeline after the S-box forming 5 inner pipeline stages, that’s because the delay of MC is $3T_{Add}$, the delay of ARK is $1T_{Add}$, and the total delay is $4T_{Add}$. The total pipeline stages in our implementation are 60. The detailed implementation of high-efficiency AES can be found in our previous work in Ref. [13].

3 Proposed high efficiency FR-KOA FFM

The FR-KOA FFM proposed in this paper is shown in Fig. 3. It contains four stages: Splitting stage, Sub-multiplier stage, Splicing stage and Reduction stage. The design details are explained as follows.

(1) Splitting stage

The first step in our FR-KOA FFM is to decompose the multiplicator with a larger data bit width into smaller bits. It provides operands for the subsequent sub-multiplier operation. The process of splitting stage is displayed in Fig. 4. Operands $A$ and $B$ can be decomposed into three parts $C$, $D$, $E$, and the expression is as shown in Eq. (1).
\[ C = A_H B_H \]
\[ D = A_L B_L \]
\[ E = (A_H + A_L)(B_H + B_L) \]

Eq. (1) shows an example when decomposition times \( r = 1 \). When \( r \geq 1 \) is needed, \( C, D, \) and \( E \) are split iteratively according to Eq. (1), and so on, all of the results of \( 1 \leq r \leq 6 \) can be obtained. For ease of calculation, the decomposed data nodes are numbered. The original input node is denoted as “0”, and the three new nodes \( C, D, \) and \( E \) obtained by each node are denoted as “1”, “2”, and “3”, respectively. Node numbers connected from beginning to end to form a branch. For example, when \( r = 2 \), there are 13 nodes and 9 branches. The branch names are “011”, “012”, “013”, “021”, “022”, “023”, “031”, “032” and “033”, it is necessary
to number the nodes and branches, because when \( r \) is large, the number of nodes and branches is thousands, and it is easy to cause confusion in calculations.

Decomposition times \( r \) (1 \( \leqslant r \leqslant 6 \)) have a direct impact on the circuit’s resource overhead and latency. In this paper, we look for the best point of circuit performance through traversal \( r \) in the splitting stage.

(2) Sub-multiplier stage

The sub-multiplier multiplies the split data. When the original operand is decomposed \( r \) times, a total of \( 3^r \) sub-multipliers are required, and the sub-multiplier performs a multiplication of two bit-width \( 2^{7-r} \) operands.

(3) Splicing stage

Splicing operation can be regarded as the inverse process of data splitting. Assuming that the sub-multiplier calculation result \( C \) can be expressed as
\[
C = \frac{CH}{CL} = C_{H} \oplus C_{L},
\]
and \( D \) can be expressed as
\[
D = \frac{DH}{DL} = D_{H} \oplus D_{L},
\]
then the splicing operation can be represented by Eq. (2).

\[
AH : AL \ast BH : BL = C_{H} \oplus C_{L} \oplus D_{H} \oplus E_{H} : D_{L} = D_{H} \oplus C_{L} \oplus D_{L} \oplus E_{L} : D_{L} \quad (2)
\]

(4) Reduction stage

Different from the traditional matrices method to get the reduction, we adopt a fast reduction algorithm in this paper [14]. The process is shown as follows:

**Input:** Length of 256 bit string \([X_3 : X_2 : X_1 : X_0] \), \(X_3, X_2, X_1, X_0\) are 64 bits

1) \(X_3\) shift right by 63, 62, 57 bits and assigned to \(A, B, C\) respectively:
\[
A = X_3 \gg 63, \quad B = X_3 \gg 62, \quad C = X_3 \gg 57
\]
2) Calculated \(D\) by XOR \(A, B, C\) and \(X_2: D = X_2 \oplus A \oplus B \oplus C\)

3) \([X_3 : D]\) shift left by 1, 2, 7 bits and assigned to \(E, F, G\) respectively:
\[
\begin{align*}
[E_H : E_L] &= [X_3 : D] \ll 1 \\
[F_H : F_L] &= [X_3 : D] \ll 2 \\
[G_H : G_L] &= [X_3 : D] \ll 7
\end{align*}
\]
4) Calculated \(H\) by XOR \(E, F, G\) and \([X_3 : D]\):
\[
[H_H : H_L] = [X_3 \oplus E_H \oplus F_H \oplus G_H : D \oplus E_L \oplus F_L \oplus G_L]
\]

**Output:** \([X_1 \oplus H_H : X_0 \oplus H_L]\) (the reduction)

The fast reduction method can effectively carry out the pipeline strategy and improve the throughput, shorten the reduction delay and reduce the hardware consumption.

As we can see from Fig. 3, the proposed carry-less FR-KOA multiplier has a 6-stage pipeline. The delay of the reduction is expressed in Table III.

| Table III. The delay of reduction |
|-----------------------------------|
| traditional matrix method | our method |
| Delay | \(7T_{\text{XOR}}\) | \(3T_{\text{XOR}}\) |

\(T_{\text{XOR}}\) donates the delay of XOR gate. In reduction stage, we insert the pipeline after step 2 and step 4, the delay of each pipeline stage is about \(3T_{\text{XOR}}\).

The fast reduction can help to reduce the delay of the carry-less multiplier. Besides, it only requires exclusive-OR gate without consuming the AND gate, which saves hardware overhead.
The implementation result of our FR-FFM is shown in Table IV.

Table IV. The implementation result of FR-FFM

| Works   | FPGA   | step | pipeline stage | Fre. (MHz) | Thr. (Gbps) | LUTs  | Eff. (Mbps/LUT) |
|---------|--------|------|----------------|------------|-------------|-------|-----------------|
| Ref. [8] | Virtex5 | 1    | 0              | 234.544    | 30.021      | 7596  | 3.95            |
| Ref. [9] | Virtex5 | 1    | 4              | 338.403    | 43.316      | 6935  | 6.25            |
| Ref. [10] | Virtex5 | 1    | 0              | 240.241    | 22.249      | 6162  | 3.61            |
| Ref. [11] | Virtex5 | 1    | 0              | 167.532    | 21.444      | 4269  | 5.02            |
| This Work | Virtex5 | 1    | 6              | 343.070    | 43.913      | 6039  | 7.27            |
|          |        | 2    |                | 389.756    | 49.889      | 5098  | 9.79            |
|          |        | 3    |                | 423.872    | 54.256      | 4786  | 11.34           |
|          |        | 4    |                | 357.389    | 45.746      | 4770  | 9.59            |
|          |        | 5    |                | 302.244    | 38.687      | 5465  | 7.08            |
|          |        | 6    |                | 271.437    | 34.744      | 6181  | 5.62            |

Our implementation of FR-KOA FFM can reach the throughput of 54.256 Gbps at the cost of 4786 LUTs, and the best efficiency is 11.34 Mbps/LUT. The KOA multiplier designed by Ref. [8] and Ref. [10] only took 1-step KOA, and there were no pipelines in their multiplier, resulting in large area and low throughput. In Ref. [9] there were 4 stages pipeline in KOA multiplier, the throughput was high, but the hardware overhead was not optimal. Compared to Ref. [11], although our hardware consumption is slightly higher, but our throughput is much higher, so our efficiency is much higher than Ref. [11] because of the pipeline strategy.

4 High efficiency AES-GCM overall architecture

Fig. 5 shows our high-throughput AES-GCM architecture.

![Fig. 5. High-throughput AES-GCM architecture](image)

After the input data is loaded into the multiplier, the multiplication result is obtained after 6 clocks, and then it feedback and is XORed with the next input. The last five inputs are multiplied by $H \sim H^5$ and the rest are multiplied by $H^6$. The calculation of is illustrated as Eq. (3).
\[ X_i = Q_6 \oplus Q_5 \oplus \cdots \oplus Q_2 \oplus Q_1 \\
Q_6 = ((I_1H^6 \oplus I_2)H^6 \oplus I_13)H^6 \oplus \ldots )H^6 \\
Q_5 = ((I_1H^6 \oplus I_6)H^6 \oplus I_14)H^6 \oplus \ldots )H^5 \\
Q_4 = ((I_3H^6 \oplus I_9)H^6 \oplus I_13)H^6 \oplus \ldots )H^4 \\
Q_3 = ((I_4H^6 \oplus I_{10})H^6 \oplus I_{16})H^6 \oplus \ldots )H^3 \\
Q_2 = ((I_5H^6 \oplus I_{11})H^6 \oplus I_{17})H^6 \oplus \ldots )H^2 \\
Q_1 = ((I_6H^6 \oplus I_{12})H^6 \oplus I_{18})H^6 \oplus \ldots )H \\
(I_1, I_2, \ldots, I_{m+n+1}) = (A_1, \ldots A_m^* \parallel 0^{128-v}, C_1, \ldots, C_n^* \parallel 0^{128-u}, \text{len}(A) \parallel \text{len}(C)) \tag{3} \]

\( I_i \) donates input data, the meaning of \( X_i, A_i, C_i, \) and \( \text{len}(A) \parallel \text{len}(C) \) can be found in Ref. [1]. Before encryption, some initialization operations are required, including AES key expansion and the computation of \( H \) and \( H^6 \). In this paper, the round keys are pre-calculated and stored in the register. So, there is no need to calculate the round key in real time, which is beneficial to the operation of the pipeline and reduces the complexity of AES. Hash operator \( H \) is calculated by \( E(k, 0^{128}) \), and \( H^6 \) is iteratively calculated using the multiplier, which costs 12 cycles. Pre-calculated \( H \) and \( H^6 \) are stored in the register.

From Fig. 5, we can know that it will consume the most time to obtain \( H^5 \), which the multiplier needs to run 5 times to get. Take a total of 6 input data for GHASH as an example, thus \( Q_5 = I_2H^5 \) needs \( 2 + 5 \times 6 = 32 \) cycles, so, from the inputs of GHASH to obtain \( T = E(Y_0) \oplus (Q_1, Q_2, Q_3, Q_4, Q_5) \), it needs a total of 33 cycles.

5 Comparison and results

The AES-GCM designed in this paper has been implemented in Xilinx Virtex5 platforms. The implementation results of FFM and the overall AES-GCM are shown in Table V.

| AES     | GHASH           | Fre. (MHz) | Thr. (Gbps) | Slices | Eff. (Mbps/\text{Slice}) | FPGA   |
|---------|-----------------|------------|-------------|--------|--------------------------|--------|
| Ref. [5]| pipeline parallel | 233        | 48.8        | 18505  | 2.64                     | Virtex5|
| Ref. [7]| pipeline no pipeline | 232        | 29.7        | 5512   | 5.38                     | Virtex5|
| Ref. [6]| pipeline pipeline | 264.2      | 31.4        | 7475   | 4.19                     | Virtex5|
| This Work | pipeline pipeline | 381.6      | 48.8        | 6482   | 7.54                     | Virtex5|

In terms of overall AES-GCM, in Ref. [5] the throughput did not increase four times as much as the area, causing low efficiency. Compared to Ref. [6], the throughput of this work is much higher because of our pipeline strategy in both AES and KOA FFM. In Ref. [7], the AES was implemented by key-synthesized method, so the slices consumption was lower than our work, however, their pipeline strategy was not reasonable, so the throughput was much lower than our work, as a result, our efficiency is higher than Ref. [7].
6 Conclusion

The letter presents a high efficiency AES-GCM with high efficiency AES and high efficiency FR-FFM. For the FR-FFM, the KOA and fast reduction algorithm are combined to design a 6-stage pipelined carry-less multiplier. The FR-FFM obtains the best efficiency when KOA is iterated by 3 times. Together with the high efficiency AES, it can form a high efficiency AES-GCM. The AES-GCM implemented in this paper can reach the throughput of 48.8 Gbps at the cost of 6482 Slices in Virtex5 platform, and the efficiency of the implementation is 7.54 Mbps/Slice.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61774086, No. 61376025), the Natural Science Foundation of Jiangsu Province (No. BK20160806), and the Fundamental Research Funds for Central Universities (No. NS2017023).