A High Speed Architecture for Galois/Counter Mode of Operation (GCM)

Bo Yang, Sambit Mishra, Ramesh Karri
ECE Department
Polytechnic University, Brooklyn, NY

Abstract

In this paper we present a fully pipelined high speed hardware architecture for Galois/Counter Mode of Operation (GCM) by analyzing the data dependencies in the GCM algorithm at the architecture level. We show that GCM encryption circuit and GCM authentication circuit have similar critical path delays resulting in an efficient pipeline structure. The proposed GCM architecture yields a throughput of 34 Gbps running at 271 MHz using a 0.18 μm CMOS standard cell library.

1 Introduction

Advanced Encryption Standard (AES) [1] and HMAC-MD5 [2] or HMAC-SHA1 [3] are the primary encryption and authentication infrastructures for current network security applications. They have been implemented as Application Specific Integrated Circuits (ASICs) [4][5][6][10][11] or on Field Programmable Gate Array (FPGAs) [8][9] to meet high throughput requirements.

Private key encryption algorithms can operate in various modes of operation, such as non-feedback electronic book code mode (ECB), output feedback mode (OFB), cipher feedback mode (CFB), and cipher block chain mode (CBC) [12]. In the feedback modes, the current computation step depends on the result of the previous step resulting in iterative hardware implementations [5][6][8] whose throughput are generally less than 4 Gbps. For example, an iterative AES implementation targeting a 0.18 μm CMOS ASIC library can achieve a throughput of 3.84 Gbps [7]. The only design that has a 10 Gbps throughput even in feedback mode is from IBM [6], but most of the contribution to the high throughput is from the advanced fabrication technology. Fully pipelined AES architecture can be applied to non-feedback ECB mode. Since there are 10 round operations in AES, a fully pipelined AES implementation can achieve $30 \sim 70$ Gbps throughput [4] consuming almost 10 times the area.

MD5 and SHA1 are inherently iterative in which every 512-bit message block is processed by 16 steps and the result is fed back for the computation of the next 512-bit message block. They are not parallelizable and cannot be pipelined. Their hardware implementations yields a throughput of around 1 Gbps [10][11]. The throughput of MD5 and SHA1 implementations are much smaller than that of AES implementations and are bottlenecks in any integrated authenticated encryption system that uses them. There is a compelling need for a mode of operation that can efficiently provide authenticated encryption at 10 Gbps and beyond in high speed network and computer system applications.

Several proposals have been submitted to National Institution of Standards and Technology (NIST) for the authenticated encryption modes [13]. These include Counter with CBC-MAC (CCM) [14], EAX [15], Carter Wegman with Counter (CWC) [17], and Galois Counter Mode (GCM) [16]. All of these proposals use AES in Integer Counter Mode (ICM) for encryption. In ICM, AES block cipher encrypts the value of a counter in the ECB mode to generate a keystream that is then bitwise exclusive-ored into the plaintext to produce the ciphertext. In ICM, pipelined AES implementations can be used resulting in encryption rates of > 10 Gbps. CCM [14] and EAX [15] modes also use AES in CBC mode to provide authentication. Since CBC is a feedback mode, authentication in CCM and EAX cannot be speeded up.

In contrast, CWC [17] and GCM [16] use universal hash based authentication, in which additions and multiplications are the main operations. Message authentication in CWC uses 127-bit integer multiplication and 127-bit integer addition, while message authentication in GCM uses 128-bit Galois Field (GF) multiplication and 128-bit GF addition (this is a simple bit-wise exclusive or operation). NIST has recommended CWC and GCM as candidates for authenticated encryption [13].

One straightforward approach to designing high speed GCM hardware architecture is to use fast implementations of AES and GF multiplier cores. Efficient hardware implementations of AES [6][9] and GF multiplier [21] [23][22] have been extensively studied. For example, different implementations such as look-up table, composite field and Binary Decision Diagram have been proposed to optimize S-box circuit that dominates the critical path of AES circuit [6][9]. The Galois field multiplier can be optimized for some specific types of modulus polynomials [22] or by using different bases for representation [24]. In this paper we will analyze the AES and GF multiplier cores and data dependencies in the GCM algorithm at the architecture level to develop hardware architectures for GCM.

The rest of the paper is organized as follows. In section 2, we will briefly introduce the GCM algorithm. We will then discuss the features of AES and GHASH cores in section 3. We will present the high speed GCM architecture in section 4. We will report the experimental results of the proposed GCM

architecture using a 0.18 μm CMOS standard cell library in section 5. We will discuss how this architecture can be adapted to CWC in section 6. Finally, we will summarize our contributions in section 7.

2 GCM Algorithm

GCM is a block cipher mode of operation that uses universal hashing over a binary Galois field to provide authenticated encryption. GCM supports authenticated encryption and authenticated decryption [16].

2.1 GCM Encryption

GCM authenticated encryption operation has four inputs:

- A secret key K. We assume that it is 128 bits long consistent with the underlying AES block cipher.
- An initialization vector (IV) can have up to 2^{64} bits. A 96-bit IV is recommended for efficiency.
- A plaintext P that can have up to $\sim 2^{39}$ bits.
- Additional authenticated data A that have up to 2^{64} bits. This additional authenticated data is authenticated but not encrypted.

and two outputs:

- A ciphertext C whose length is identical to that of the plaintext P.
- An authentication tag T that have up to 128 bits. The length of the tag is denoted as t.

The plaintext data and the additional authenticated data are segmented into 128-bit blocks. Suppose there are n plaintext blocks $P_1, P_2, ...P_{n-1}, P_n$ and m additional authenticated data blocks $A_1, A_2, ...A_{m-1}, A_m$[1]. The GCM authenticated encryption operation is defined as follows [16]:

$$\begin{aligned} H &= E(K, 0^{128}) \\ Y_0 &= \begin{cases} IV||\ 0^{31}1 & \text{if len(IV)=96} \\ GHASH(H, \{\}, IV) & \text{otherwise.} \end{cases} \\ Y_i &= incr(Y_{i-1})\ for\ i = 1, ..., n \\ C_i &= P_i \oplus E(K, Y_i)\ for\ i = 1, ..., n \\ C_n^* &= P_n \oplus MSB_u(E(K, Y_n)) \\ T &= MSB_t(GHASH(H, A, C) \oplus E(K, Y_0)) \end{aligned} \quad (1)$$

[1] P_n and A_m may not be 128-bit blocks. 0's are appended to make them into 128-bit blocks.

GHASH compresses a $128 \times (m+n+1)$-bit message stream into a 128-bit hash value X_{m+n+1} as follows [16]:

$$X_i = \begin{cases} 0 & \text{for i=0} \\ (X_{i-1} \oplus A_i) \cdot H & \text{for i=1,...m} \\ (X_{i-1} \oplus C_{i-m}) \cdot H & \text{for i=m+1,...m+n} \\ (X_{m+n} \oplus (length(A)||length(C))) \cdot H & \text{for i=m+n+1} \end{cases} \quad (2)$$

length() returns a 64-bit string representing the number of bits in its argument, with the least significant bit on the right.

2.2 GCM Decryption

The authenticated decryption operation has five inputs: secret key (K), initialization vector (IV) , ciphertext (C), additional authenticated data (A), and authentication tag (T); and it generates a single output: either the plaintext value P or a FAIL signal that indicates that the inputs are not authentic. A ciphertext C, and tag T are authentic for key K when they are generated by the encrypt operation with inputs K, IV , A and P, for some plaintext P.
GCM authenticated decryption computes the authentication tag T' and compares it with the input authentication tag T. If the two tags match, then the ciphertext is returned. Otherwise, the FAIL signal is returned. Authenticated decryption operation is similar to the encryption operation, but with the order of the hash and encryption steps reversed.

3 Component Design

AES and GHASH are the basic components in GCM encryption and in GCM decryption. We will describe the AES and GHASH component design and discuss architectural features of AES and GHASH that will be considered in designing a high speed GCM architecture.

3.1 AES Core

AES encrypts 128-bit data blocks under the control of a 128-bit user key. AES encryption or decryption supports 10 rounds, with each round using one round key. An additional key is used during pre-processing. Intuitively, AES operates on a two-dimensional table of plaintext bytes called State. Operations used in a round of AES are a nonlinear byte substitution operation (byte sub), a cyclic left shift of the rows in State (shift row), GF(2^8) multiplication of State with a constant polynomial (mix column), and exclusive-or of round key with State (key-xor) [1].

The hardware implementations of AES can be either iterative [6][8][7] or pipelined [4][8][9] as shown in Figure 1. Since there are 10 round operations in AES, the

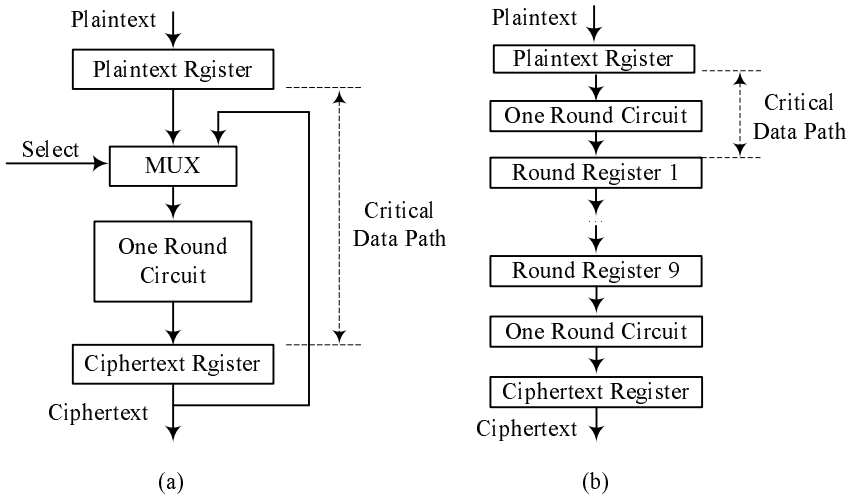

Figure 1: (a)The AES iterative data path (b)The AES pipelined data path

pipelined implementations can be understood as using ten times as much hardware overhead to achieve ten times the throughput. The iterative and pipelined architectures have similar critical paths and can run at similar clock rates, which are determined by the delay of one round circuit as shown in Figure 1 (a) and (b). Compared to the data path, the control logic segments of both the iterative and pipelined AES architectures are very simple and are omitted from Figure 1.

Pipelined AES implementations can only be used in the ECB mode. In this mode, the output ciphertext is only determined by the input plaintext. To make use of the high throughput pipelined AES implementations, GCM encryption and GCM decryption run AES in the integer counter mode (ICM), which is a special case of ECB mode. In ICM, the AES core generates a continuous key stream by encrypting a counter whose initial value is the IV. After the first 10 clock cycles, the AES core can output a 128-bit key every clock cycle [2].

3.2 GHASH Core

The GHASH architecture is shown in Figure 2. At the core of the GHASH architecture is a 128-bit parallel GF(2^{128}) multiplier. One operand of the GF multiplier is H. H is obtained by encrypting the secret key K with an all 0's key as described in Equation 1. The Register X that holds the hash value is initially set to zero. In the first m clock cycles, the 128-bit additional authen-

[2]If the IV is updated frequently, the throughput of the AES pipeline will degrade. If the IV is updated every 10 clock cycles or less, the pipelined AES architecture will have no advantage over an iterative AES architecture in terms of performance.

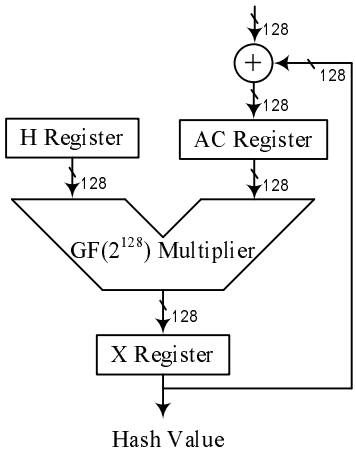

Figure 2: GHASH hardware architecture

ticated data words $A_1, A_2, ...A_m$ are applied to the right input one by one as described by Equation 2[3]. In the next n clock cycles, the 128-bit ciphertext $C_1, C_2, ...C_{n-1}, C_n$ are applied to the right input as described in the third row in Equation 2[4]. In the last clock cycle, 128-bit word $length(A)||length(C)$ is applied as described in the last row of Equation 2. Overall, it takes $m + n + 1$ cycles to compute the hash value.

A GF(2^w) multiplier multiplies two w-bit operands modulo a polynomial generating a w-bit output [18]. The polynomial used in GHASH is $1 + \alpha + \alpha^2 + \alpha^7 + \alpha^{128}$. A GF multiplier can be implemented in either parallel [19], digit-serial [20] or bit-serial architecture [21]. The hardware complexity of a parallel GF(2^w) multiplier using a modulus of fixed sparsity is $O(w^2)$ and the delay of the critical path is $O(logw)$. A bit-serial GF(2^w) multiplier takes w clock cycles to perform one multiplication. The hardware complexity of a bit-serial GF(2^w)is $O(w)$ and the delay of the critical path is $O(1)$, so a bit-serial GF multiplier can run at a very high clock rate. Digit-serial GF multiplier trade off hardware simplicity and for computational speed.

In the proposed GHASH architecture, we use a parallel GF multiplier. This is a pure combinational circuit that operates in a single clock cycle. In the GHASH architecture, the temporary result X_i is fed back and exclusive-ored with the next input to register AC to generate the next operand for the GF multiplier. Although the parallel GF multiplier can be pipelined to achieve a higher clock rate [25] this does not improve the throughput in the context of GHASH because of this feedback condition[5].

[3]If the last word of additional authenticated data is only v bits , $128-v$ zeros are appended.
[4]If C_n^* is not 128 bits long it is appended with approproate number of zeroes.
[5]feedback prohibits efficient CWC hardware architecture design

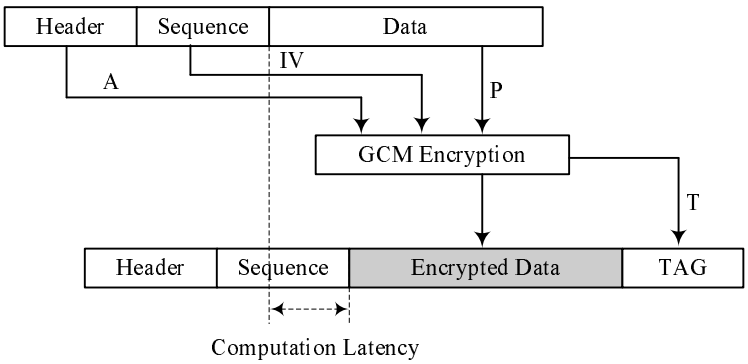

Figure 3: Using GCM to encrypt and authenticate a packet

3.3 Architectural Level Data Dependencies in GCM

GCM encrypts and authenticates a packet as shown in Figure 3. The data field is encrypted and authenticated, and is carried along with a header and a sequence number. The header is authenticated by including it in the additional authenticated data. The sequence number is included in the IV. The authentication tag is carried along with the encrypted data in an authentication tag field. The computation latency between getting the first payload word and outputting the first encrypted payload word is as shown in Figure 3. A design with high computation latency needs a lot of memory to buffer incoming packets and is not suitable for high data rates.

The data dependencies in GCM encryption are shown in Figure 4(a). It takes 10 clock cycles to compute H from the user key K for both iterative and pipelined AES implementations because of the cold start of the pipelined structure. Generally, a single secret key is used for all packets processed in a given secure session. This secret key is determined upon session initiation. Hence, the secret key and H are ready before packets are transmitted. GCM starts computing temporary hash value X_i when it receives packet header as additional authenticated data (A). It takes m clock cycles to generate X_m. Then the hash computation has to be halted for $11 + r$ clock cycles until the first cipher text is ready as shown in Figure 4(a). If the sequence number is 96 bits long, the IV for the ICM counter Y_0 can be generated without any latency. Otherwise, it takes r clock cycles to generate Y_0 assuming that the IV has r 128-bit words. The key stream for GCM encryption is available 10 clock cycles after Y_0 is available. Once the AES ICM pipeline is full, a 128-bit ciphertext word C_i is generated every clock cycle. The hash computation resumes when C_1 is ready. There is a one clock cycle bubble between the last cipher word C_n and the final hash value X_{m+n+1} because of the computation of $(length(A)||length(C)) \cdot H$.

The data dependencies in GCM decryption are shown in Figure 4(b). Since the hash computation is performed on the ciphertext that is from the input

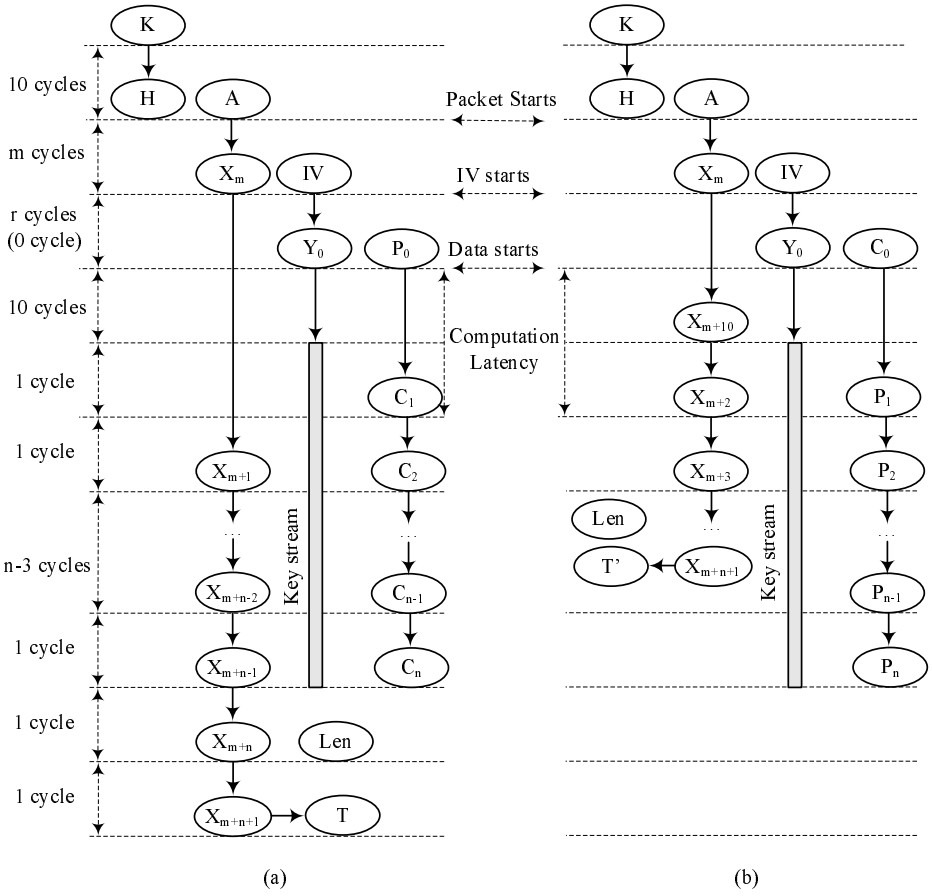

Figure 4: (a) The data dependency of GCM encryption (b) The data dependency of GCM decryption

directly in the GCM decryption, the hash computation can continues when the key stream is in computing. This saves 10 clock cycles. The hash value T' is generated before the last plaintext P_n is generated.

4 High Speed Architectures for GCM

Based on the above analysis of data dependencies in GCM encryption, a fully pipelined GCM hardware architecture with 11 clock cycle computation latency can be developed as shown in Figure 5. The shaded components are registers or are register bounded. All the buses are 128-bit wide. The control signals for multiplexors enable signals for registers are not shown. They are generated by the control unit and their timing can be determined according to the data dependency.

An iterative AES core is used to compute H from user secret key K. Computing H for future packets can be overlapped with the current GCM encryption operation and hence is not in the critical path of design. As we discussed in section 3.2, in the first m clock cycles, the input to register *AC Reg* is the additional authenticated data word (A_i). In the next n clock cycles, input to register *AC Reg* is the ciphertext word (C_i). Finally in clock cycle $m+n+1$ $length(A)||length(P)$ is the input. A 3-to-1 multiplexor MUX2 is used before $ACReg$ to select among these three inputs. A pipelined AES core is used to generate key stream for the integer counter mode encryption. The initial value of the counter is either from input directly (when IV is 96 bits) or the output of GHASH. The multiplexor MUX1 is used to select between $XReg$ (output of GHASH) and *Input Reg*. The first 128-bit word in the key stream is stored in $E(K, Y_0)$ *Reg*. This is then used to compute the authentication tag as described by the last step of Equation 2. The $CReg$ register is used to delay outputting the ciphertext by one clock cycle so as to remove the one clock cycle bubble between when the last ciphertext word C_n^* is computed and when the authentication tag is computed as shown in Figure 4.

Since the computation latency is 11 clock cycles (10 clock cycles to fill the AES pipeline+ 1 clock cycle to perform stream encryption) and there is a one clock cycle bubble between the receipt of the ciphertext and the authentication tag, a 12×128-bit FIFO has to be used to store the incoming packet. In the first 12 clock cycles, only the write enabled of the FIFO is valid. Subsequently, both write enable and read enable of the FIFO are valid. The output of FIFO either goes to output directly (for packet header (A) and sequence number (IV) as shown in Figure 3) or is exclusive-ored with the key stream generated by the pipelined AES core (for payload data (P) as shown in Figure 3). A 3-to-1 multiplexor MUX3 is used before *Output Reg* to select among (i) the packet header and sequence number (from *Input Reg*), (ii) encrypted payload data (from *C Reg*) and (iii) authentication tag (from the exclusive-or of *X Reg* and $E(K, Y_0)$ *Reg*).

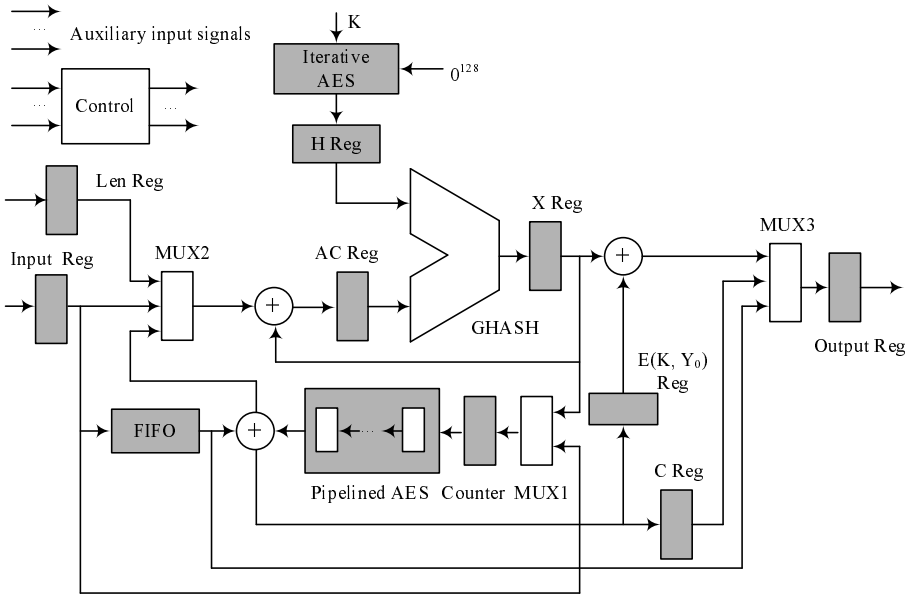

Figure 5: GCM encryption architecture

If bubbles are allowed between (packet header, sequence) and encrypted payload, the FIFO can be removed. The packet header and sequence can be forward to output directly and the output is invalid for 11 clock cycles until encrypted payload data is generated. However, work has to be done in the following chip to remove the bubble.

The critical path of this design is determined by the GF(2^{128}) multiplier, the delay through which is approximately a delay of 1 AND gate + 7 XOR gates. The delay of all other paths in this design is smaller than this as shown in Figure 5.

The GCM decryption architecture is similar to the GCM encryption architecture. The third input to MUX2 will not be used and hence will not be selected. This is because the authentication tag T' is computed directly from the (ciphertext) input. Similarly, the first input to MUX3 is never used and hence will not be selected. A comparator is used to generate the FAIL signal. The delay of a 128-bit comparator is approximately 1 XOR gate+7 AND gates which is still smaller than the delay of GF(2^{128}) multiplier. The $CReg$ register can be removed as there is no bubble between the last ciphertext word and authentication tag in the GCM decryption. This is because we do not need to output the authentication tag in GCM decryption.

Table 1: area, maximum clock rate, throughput,latency of the iterative AES core, pipelined AES core, GHASH, GCM encryption architecture, GCM decryption, GCM encryption/decryption architecture

Designs	Area (gates)	clock rate (MHz)	Throughput (Gbps)	Latency (cycles)
Iterative AES	29,436	276	3.53	10
Pipelined AES	287,184	282	36.09	1(steady status)
GHASH	78,974	271	34.69	1
GCM encryption	463,328	271	34.69	12
GCM decryption	446,108	271	34.69	11
GCM en/decryption	498,658	271	34.69	12

An architecture that combines GCM encryption with GCM decryption can also be designed taking into account the above discussion.

5 Experimental Results

The proposed GCM authenticated encryption architecture was modeled in Verilog HDL and simulated using Modelsim. The Verilog models were synthesized using Synopsys Design Compiler targeting a TSMC 0.18μm CMOS standard cell library. The area and clock rate were reported after the netlist generated by Synopsys Design Compiler was placed and routed by Cadence Silicon Ensemble.

The Look-up Table structure was used for S-box design in AES cores [6]. Mastrovito parallel GF multiplier architecture was used for GHASH component design [23]. A Mastrovito parallel GF(2^n) multiplier use n^2 two-input AND gates and $O(n^2)$ two-input XOR gates, with the constant factor of n^2 dependent upon the sparsity of the modulus polynomial. It is pure combinational logic and each output bit is a function of several input bits that is determined by the polynomial. An automatic Mastrovito parallel GF multiplier core generator was developed using C++. The core generator takes the polynomial as the input and output Verilog description. For GHASH, the input polynomial is $1+\alpha+\alpha^2+\alpha^7+\alpha^{128}$. Table 1 summarizes area, maximum clock rate, throughput,latency of the iterative AES core, pipelined AES core, GHASH, GCM encryption architecture, GCM decryption, GCM encryption/decryption architecture.

The critical path of the GCM architectures is from GHASH. After the first 11 or 12 cycles, the GCM architectures is fully pipelined and reach the maximum throughput of 34.69 Gbps (=271MHz$\times$ 128bit). If the interval between two consecutive packets is larger than 11 or 12 cycles, such an 11 or 12 clock cycle

cold start occurs for every packet. The throughput increases with the size of packets. For example, for a $2K$-byte packet, the throughput degrades to $91\%(\frac{(2048\times8)\div128}{(2048\times8)\div128+12})$ that is 31 Gbps.

6 Discussion

The initial value of counter is determined by the sequence number of a packet which is just before the payload data. The payload data has to be buffered when computing the keystream. If the packet structure can be modified by putting sequence number before some part of header, the 10 cycle cold start of AES can be overlapped with receiving packet header and the FIFO can be removed. When payload data arrives, the keystream is already ready. When both the sender and receiver's equipments are provided by the same vendor, such a modification may be appropriate.

The bubble between the last ciphertext word and authentication tag in GCM encryption can be removed by modifying GHASH algorithm a little. Since the length of additional authentication data (A) and payload data (P) are normally available after the packet header, we apply the 128-bit lenth(A)||lenth(P) after the additional authentication data instead of at the very end of hash computation. The one cycle computation time is overlapped with cold start of AES cores. The modified GHASH algorithm is defined as:

$$X_i = \begin{cases} 0 & \text{for i=0} \\ (X_{i-1} \oplus A_i) \cdot H & \text{for i=1,...m} \\ (X_{m+1} \oplus (len(A)||len(C))) \cdot H & \text{for i=m+1} \\ (X_{i-2} \oplus C_{i-m-1}) \cdot H & \text{for i=m+2,...m+n+1} \end{cases} \quad (3)$$

7 Conclusions

In this paper, we designed a 34 Gbps GCM encryption and GCM decryption architectures by analyzing the data dependencies of the GCM algorithm at the architecture level. We show that GCM is suitable for hardware implementations because the encryption circuit and authentication circuit in GCM have similar critical path delays resulting in well balanced pipeline stages. Some suggested modifications to GCM to further reduce computation latency are also presented.

CWC also uses AES in ICM for encryption, but uses a 127-bit integer multiplication based universal hash function for authentication [17]. Based on our understanding, a similar architecture can be designed for CWC. When targeted on the same 0.18μm CMOS standard cell library, a 127-bit parallel Wallace tree multiplier can only achieve approximately 78 MHz clock rate. This becomes the bottleneck in the design, resulting in unbalanced pipeline stages and preventing efficient hardware architectures for CWC. In section 3.2 we showed that pipelining the multiplier in universal hash functions used in CWC and GCM do not

improve the throughput of authentication because of the inherent feedback.

References

[1] J. Daemen and V. Rijmen, "AES proposal: Rijndael," http://www.esat.kuleuven.ac.be/ rijmen/rijndael/rijndaeldocV2.zip

[2] R. Rivest, "The MD5 Message-Digest Algorithm," IETF RFC1321,1992. http://www.ietf.org/rfc/rfc1321.txt

[3] D. Eastlake and P. Jones, "US Secure Hash Algorithm 1," IETF RFC3174, 1992. http://www.ietf.org/rfc/rfc3174.txt

[4] A. Hodjat, I. Verbauwhede, "Minimum Area Cost for a 30 to 70 Gbits/s AES Processor," IEEE computer Society Annual Symposium on VLSI, pp. 83-88, Feb. 2004.

[5] S. Mangard, M. Aigner and S. Dominikus, "A Highly Regular and Scalable AES Hardware Architecture," IEEE Transactions on Computer, Vol. 52(4), pp. 483-491, April 2004.

[6] S. Morioka and A. Satoh, "A 10 Gbps Full-AES Crypto Design with a Twisted-BDD S-Box Architecture," pp. 98-103, International Conference of Computer Design, 2002.

[7] A. Hodjat, D. Hwang, B.C. Lai, K. Tiri, I. Verbauwhed, "A 3.84 Gbits/s AES Crypto Coprocessor with Modes of Operation in a 0.18um CMOS Technology," ACM Great Lake Symposium on VLSI, April 2005

[8] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 9(4), pp. 545-557, Aug. 2001.

[9] X. Zhang and K. K. Parhi, "High-speed VLSI Architectures for the AES Algorithm," IEEE Transanctions on Very Large Scale Integration (VLSI) Systems, vol. 12(9), pp. 957-967, Sep. 2004.

[10] "Datasheet-High Performance SHA1 Hash Core for ASIC," 2003. http://www.heliontech.com/downloads/sha1_asic_helioncore.pdf

[11] "Datasheet-High Performance MD5 Hash Core for ASIC," 2003. http://www.heliontech.com/downloads/md5_asic_helioncore.pdf

[12] B. Schneier, "Applied Cryptography," Second Edition, John Wiley & Sons, Inc. New York, 1996

[13] "Modes of Operation for Symmetric Key Block Ciphers," http://csrc.nist.gov/CryptoToolkit/modes/

[14] D. Whiting, R. Housley, and N. Ferguson, "Counter with CBC-MAC: AES Mode of Operation," Proposal submitted to NIST for Authenticated Encryption Modes, Work in Progress, June, 2003. http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/ccm/ccm.pdf

[15] M. Bellare, P. Rogaway, and D. Wagner, "A Conventional Authenticated-Encryption Mode," Proposal submitted to NIST for

Authenticated Encryption Modes, Work in Progress, April, 2003. http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/eax/eax-spec.pdf

[16] D. A. McGrew and J. Viega, "The Use of Galois/Counter Mode (GCM) in IPsec ESP," Proposal submitted to NIST for Authenticated Encryption Modes, Work in Progress, October, 2004. http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-spec.pdf

[17] T. Kohno, J. Viega, and D. Whiting, "The CWC Authenticated Encryption (Associated Data) Mode",Proposal submitted to NIST for Authenticated Encryption Modes, Work in Progress, May, 2003. http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/cwc/cwc-spec.pdf

[18] R. Lidl and H. Niederreiter, "Introduction to Finite Fields and Their Applications," Cambridge University Press, New York, 1994.

[19] C. Paar, "Efficient VLSI Architectures for Bit-Parallel Computation in Galois Field," PhD Thesis, Institutes for Experimental Mathematics, University of Essen, Essen, Germany, June, 1994.

[20] L. Song and K.K. Parhi, "Efficient Finite Field Serial/Parallel Multiplication," International Conference on Application-Specific Systems, Architectures, and Processors, pp. 72-82, August, 1996.

[21] M.A. Hasan and V.K. Bhargava, "Bit-Serial Systolic Divider and Multiplier for Finite Fields GF(2^m)," IEEE Transactions on Computer, Vol. 41, No. 8, pp. 972-980, August, 1992.

[22] C. Paar, P. Fleischmann and P. Roelse, "Efficient Multiplier Architectures for Galois Fields GF(2^{4n})," IEEE Transactions on Computers, vol. 47, no. 2, pp. 162-170, February 1998.

[23] E. D. Mastrovito," VLSI architectures for multiplication over finite field GF(2^m). In Lecture Notes in Computer Science, No. 357, pp. 297ÍC309, Springer-Verlag, Berlin, March 1989.

[24] I.S. Hsu, T.K. Truong, L.J. Deutsch, and I.S. Reed, "A comparison of VLSI architecture of finite field multipliers using dual- normal- or standard bases," IEEE Transactions on Computers, Vol. 37, No. 6, pp. 735-739, June, 1988.

[25] G. Ahlquist, B. Nelson, and M. Rice, "Optimal Finite Field Multipliers for FPGAs," International Workshop on Field Programmable Logic and Applications, pp. 51-60, August, 1999.