GCM モードの並列高速ハードウェア実装

佐藤 証[†]

認証子付き暗号モード GCM（Galois Counter Mode）を，100 Gbps 以上のスループット処理することが可能な並列高速ハードウェアを提案する．GCM モードでは，暗号化処理については複数のデータブロックの並列処理が可能であるが，認証子を生成するハッシュ関数は暗号文ブロックをシーケンシャルに積和演算処理しなければならない．そこでハッシュ関数部の高速化のため，並列処理を可能とする積和演算回路を開発し，それを合成体と BDD（Binary Decision Tree）ロジックによる S-box を用いた 14 段パイプラインアーキテクチャの AES 暗号回路とともに GCM 回路に実装した．4 並列処理のデザインを 0.13 μm CMOS スタンダードセルライブラリで評価した結果，合成体と BDD の S-box を用いた場合にそれぞれ 102 Gbps（600 Kgates）と 162 Gbps（979 Kgates）というというきわめて高い処理能力が得られた．またスループットをゲートあたりに換算した回路効率においても，従来実装に対する優位性が示された．提案アーキテクチャのクリティカルパスは，多重並列処理による回路の規模増加には影響されないローカルブロック内にある．したがって，処理ブロック数を増やすことで，スループットをスケーラブルに増加させることが可能である．

High-speed Parallelized Hardware Implementation of GCM

Akashi Satoh[†]

Parallel hardware architecture for an authenticated encryption mode GCM (Galois Counter Mode) capable of a throughput higher than 100 Gbps is proposed. In GCM, multiple data blocks can be processed in parallel for encryption, but a hash function performs multiply-add operation with the cipher-text blocks sequentially. In order to accelerate the hash function block, a parallel multiply-adder was designed and integrated into the GCM hardware with a 14-stage pipelined AES circuit and two kinds of S-Box, composite field and BDD (Binary Decision Tree) versions. Performance of a 4-parallel version was evaluated by using a 0.13 μm CMOS standard cell library, and very high throughputs of 102 Gbps with 600 Kgates and 162 Gbps with 979 Kgates were obtained by the composite and BDD S-Boxes, respectively. Higher hardware efficiency (throughput/gate) in comparison with prior art was also achieved. The critical path of the design is the multiply-adder in each local processing block, and is not affected by the number of the parallel blocks. Therefore the proposed architecture has almost linear scalability in terms of throughput versus hardware size.

1. はじめに

認証付き暗号モード GCM（Galois Counter Mode）[1] は，CTR モード[2] による暗号化と同時に，ガロア体 $GF(2^{128})$ 上の積和演算を用いたハッシュ関数により改ざん防止用の認証子を高速に生成するものである．米国連邦標準技術研究所 NIST（National Institute of Standards and Technology）は，GCM を SP800-38D として採用しており[3]，また IETF（The Internet Engineering Task Force）の RFC4106[5] や IEEE 802.1AE[6] では，AES[4] を用いた AES-GCM が採用されている．

筆者は回路規模とスループットがスケーラブルな GCM の回路アーキテクチャを提案し，様々なアーキテクチャの AES 回路と組み合わせた高性能な実装について報告している[8]．それらの実装における速度のボトルネックは AES の暗号化処理にあったが，暗号化に使用している CTR モードでは，AES の回路ブロックを増やして並列処理することで容易に高速化を図ることができる．また認証子生成では，ハッシュ関数においてガロア体 $GF(2^{128})$ 上の積和演算が行われるが，それは AES による暗号化に比べてきわめて高速である．しかしながら，AES 回路の並列度を上げていくと，シーケンシャル処理であるこの積和演算回路が新たなボトルネックとなってくる．文献 1), 7) では，この処理を 2 つの独立した積和演算に分けることによる並列回路実装の可能性を示している．しかし，

† 独立行政法人産業技術総合研究所
National Institute of Advanced Industrial Science and Technology

そこでは演算式が示されているだけであり，並列回路アーキテクチャの提示や実装評価はいっさいなされていない．LSI 実装においては，演算式や計算アルゴリズムには現れないゲートサイズや信号遅延といった物理的なパラメータも性能に大きな影響を与える．また，同じ演算式やアルゴリズムであっても，回路構成が一意に定まるとは限らず，さらに同じ結果を与える異なる式やアルゴリズムも考えることができる．したがって，回路アーキテクチャを定め，ゲートサイズや動作速度の検証を行うことで，その演算式やアルゴリズムの LSI 実装における有効性や動作時の問題点などが明らかになることが多い．

そこで本論文では，GCM 回路においてハッシュ関数の積和演算を並列化して高速化を図るアーキテクチャの提案と実装性能の検討を行う．まず，上述の 2 並列の式を n 並列に拡張するとともに，異なる n 並列の積和演算の式も示し，そのどちらがより高性能回路実装に適しているかについて考察する．それに続き，GCM 全体の並列回路アーキテクチャを提案し，4 並列の場合の動作を詳解する．そして最後に，0.13 μm CMOS スタンダードセルライブラリによる論理合成結果を従来実装と比較し，提案アーキテクチャの有効性を検証する．

2. GCM のアルゴリズム概要

GCM における認証子 T の生成手順を式 (1) に示す．まず後述のハッシュ関数 $GHASH$ を用いて，秘密鍵 K によって決まる 128 ビット定数 H，暗号化は行わず認証だけを必要とする m ブロックのデータ A，そして CTR モードで処理された n ブロックの暗号文 C を引数として，128 ビットのハッシュ値を計算する．それとイニシャルベクタ IV（推奨 96 ビット）を鍵 K で暗号化した値を XOR し，必要な先頭 t ビットをとったものが認証子 T となる．なおデータ A，と平文 P は，128 ビットの m ブロック $A_1, A_2, \cdots, A_{m-1}, A_m^*$ と n ブロック $P_1, P_2, \cdots, P_{n-1}, P_n^*$ にそれぞれ分割の後に処理されるが，最終ブロック A_m^* と P_n^* が 128 ビットに満たない場合は 0 でパディングされる．なお，式 (1) で u は平文の最終ブロックのビット数を表している．

$$
\begin{aligned}
&H = Enc(K, 0^{128}) \\
&Y_0 = \begin{cases} IV||0^{31}1 & \text{if len}(IV) = 96 \\ GHASH(H, \{\}, IV) & \text{otherwise} \end{cases} \\
&Y_i = Y_{i-1} + 1 \qquad i = 1, \cdots, n \\
&C_i = P_i \oplus Enc(K, Y_i) \qquad i = 1, \cdots, n
\end{aligned}
$$

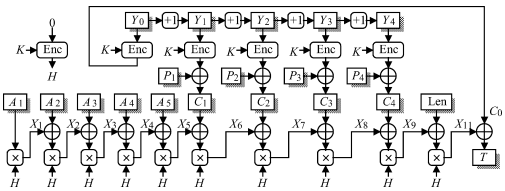

図 1 GCM の暗号化・認証子生成処理（$m = 5$，$n = 4$）
Fig. 1 Example GCM operation ($m = 5, n = 4$).

$$
\begin{aligned}
&C_n^* = MSB_u(C_n) \\
&T = MSB_t(GHASH(H, A, C) \oplus Enc(K, Y_0))
\end{aligned} \tag{1}
$$

復号時は，CTR モードで暗号文 C を平文 P に戻しながら，式 (1) と同様に認証子を再計算し，それが本来の認証子と一致するかどうかをチェックする．ハッシュ関数 $GHASH$ は既約多項式

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

で定義される $GF(2^{128})$ 上の積和演算を式 (3) に従って繰り返しながら，128 ビットのハッシュ値 X_{m+n+1} を生成する．

$$
X_i = \begin{cases}
0 & i = 0 \\
(X_{i-1} \oplus A_i)H & i = 1, \cdots, m-1 \\
(X_{m-1} \oplus (A_m^*||0^{128-v}))H & i = m \\
(X_{i-1} \oplus C_{i-m})H & i = m+1, \cdots, m+n-1 \\
(X_{m+n-1} \oplus (C_n^*||0^{128-u}))H & i = m+n \\
(X_{m+n} \oplus (len(A)||len(C)))H & i = m+n+1
\end{cases} \tag{3}
$$

図 1 に，$m = 5$，$n = 4$ の場合の GCM の暗号化および認証子生成の過程を示す．

3. 積和演算の並列化

文献 1), 7) には，$GHASH$ の入力を $\{A_1, A_3, A_5, \cdots\}$ と $\{A_2, A_4, A_6, \cdots\}$ の 2 系列にインタリーブし，それぞれに対して式 (4) のように積和演算を並列処理する手法が示されている．なお，本章と次章では簡単のため，$GHASH$ へ暗号文の入力がなく，$\{A_1, \cdots, A_m\}$ のブロック数が $m = pq$ の場合の動作を説明する．

$$
\begin{aligned}
X_i &= (X_{i-1} \oplus A_i)H \\
&= (((A_1H \oplus A_2)H \oplus A_3)H \oplus A_4)H \cdots \\
&= ((A_1H^2 \oplus A_3)H^2 \oplus A_5 \cdots)H^2 \\
&\quad \oplus ((A_2H^2 \oplus A_4)H^2 \oplus A_6 \cdots)H
\end{aligned} \tag{4}
$$

これをさらに拡張して式 (5) の 3 並列

$$X_i = ((A_1H^3 \oplus A_4)H^3 \oplus A_7\cdots)H^3 \oplus((A_2H^3 \oplus A_5)H^3 \oplus A_8\cdots)H^2 \oplus((A_3H^3 \oplus A_6)H^3 \oplus A_9\cdots)H \quad (5)$$

そして一般に q 並列化することができる．

$$\begin{aligned} X_i = {} & (\cdots((A_1H^q \oplus A_{q+1})H^q \oplus A_{2q+1})H^q \oplus \cdots \\ & \cdots \oplus A_{(p-1)q+1})H^q \\ & \oplus (\cdots((A_2H^q \oplus A_{q+2})H^q \oplus A_{2q+2})H^q \oplus \cdots \\ & \cdots \oplus A_{(p-1)q+2})H^{q-1} \\ & \vdots \\ & \oplus (\cdots((A_qH^q \oplus A_{2q})H^q \oplus A_{3q})H^q \oplus \cdots \\ & \cdots \oplus A_{pq})H \end{aligned} \quad (6)$$

また，これらとは別に，次式のような q 並列化も可能である．

$$\begin{aligned} X_i = {} & (\cdots(A_1H^q \oplus A_2H^{q-1} \oplus \cdots \oplus A_qH)H^q \\ & \oplus (A_{q+1}H^q \oplus A_{q+2}H^{q-1} \oplus \cdots \oplus A_{2q}H))H^q \\ & \oplus (A_{2q+1}H^q \oplus A_{2q+2}H^{q-1} \oplus \cdots \oplus A_{3q}H))H^q \\ & \vdots \\ & \oplus (A_{(p-1)q+1}H^q \oplus A_{(p-1)q+2}H^{q-1} \oplus \cdots \\ & \cdots \oplus A_{pq}H))H \end{aligned} \quad (7)$$

式 (6) と (7) は，第 1 サイクルで $\{A_1, A_2, \cdots, A_q\}$ を，第 2 サイクルでは $\{A_{1+q}, A_{2+q}, \cdots, A_{2q}\}$ を，といった具合にいずれも q 個の $GF(2^{128})$ 積和演算器で q ブロックずつ並列に処理していく．次章では，この両者の回路アーキテクチャについて考察する．

4. ハッシュ関数 $GHASH$ の並列回路アーキテクチャ

図 2 および **図 3** にそれぞれ式 (6) と式 (7) で $q = 4$ とした場合の回路アーキテクチャを，また**図 4** と**図 5** に A_1〜A_{10} の 10 ブロックに対する動作例を示す．なお，図 3 の H^k は，入力ブロック数に応じて H〜H^3 $(= H^{q-1})$ の値をとる．まず図 4 の演算過程を式 (8.a)〜(8.e) とともに説明する．ここで，4 並列の場合に積和演算に用いる定数 H^4 は事前計算されており，入力データ A_i の全体のブロック数もすでに与えられているものとする．まず図 2 (a) で 4 つの 128 ビットレジスタ Reg1〜Reg4 を 0 クリアし，各乗算器で A_1H^4〜A_4H^4 を計算し，次のクロックで各レジスタに T_1〜T_4 としてストアする．(b) ではこれら T_1〜T_4 が下段の XOR ツリーで加算されるが，この値を保存する必要はない．これと同時に T_1〜T_4 と新たな入力 A_5〜A_8 の間で積和演算を行うが，A_5〜A_7 には H^4 を，A_8 に H を乗じる点に注意する．これは式 (6) に示したように，最終 3 $(= q - 1)$ ブロック A_8，A_9，A_{10} にはそれぞれ H^3，H^2，H を乗じるためで，本アーキテクチャでは，A_8，A_9，A_{10} に H をそれぞれ 3 回，2 回，1 回乗じることでこれを実行する．式 (6) は簡単のため，ブロック数が q の倍数の場合を示したが，実際の回路ではこの例のように半端なブロック数に対応するために，細かな操作が必要となる．(c) では入力 A_9，A_{10} と T_5，T_4 をそれぞれ XOR 加算した後，H を乗じる．また前サイクルで A_8 を入力したラインは 0 とすることでレジスタの値 T_8' に H を乗じて T_8'' 更新する．(d) では T_8'' と T_9' に H を乗じて T_8 と T_9 に更新する．最後に (8.e) で T_7〜T_{10} を加算して X_{10} を得る．

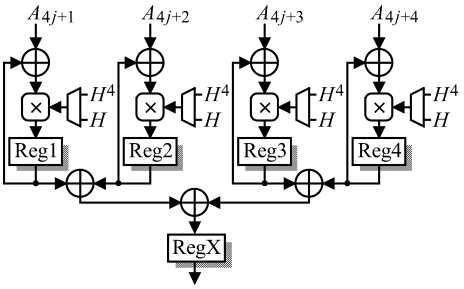

図 2 式 (6) を用いた 4 並列積和演算器
Fig. 2 4-parallel multiply-add circuit for Eq. (6).

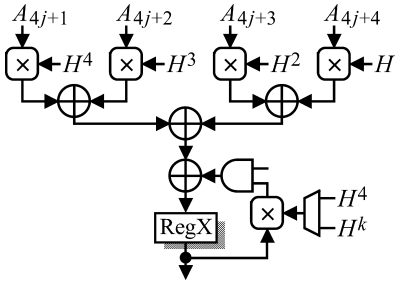

図 3 式 (7) を用いた 4 並列積和演算器
Fig. 3 4-parallel multiply-add circuit for Eq. (7).

A_8 と A_9 に対する積和演算で H を複数回乗じるのではなく，H^3 と H^2 を事前計算しておくこともできる．しかしその場合は，H^3 と H^2 の生成に余分なサイクル数を要するうえ，それらの値を保持するための専用レジスタを付加しなければならないため利点はない．なお，H^4 $(= H^q)$ はいずれにしても事前計算が必要である．

$$\begin{cases} T_1 = A_1H^4 \\ T_2 = A_2H^4 \\ T_3 = A_3H^4 \\ T_4 = A_4H^4 \end{cases} \quad (8.a)$$

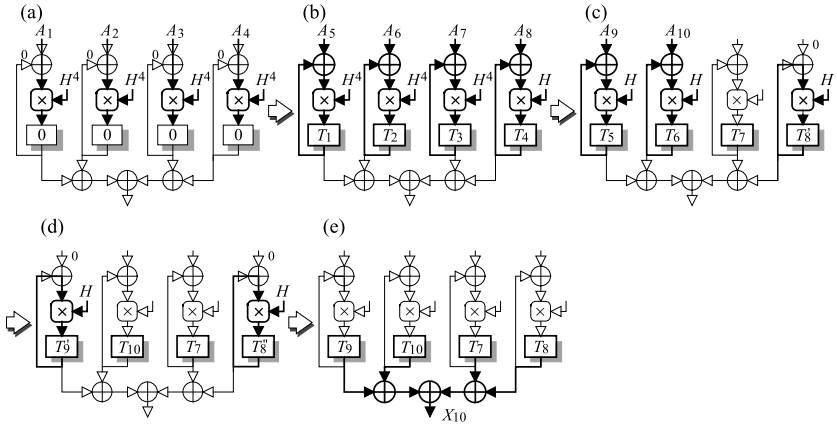

図 4 図 2 の積和演算器の動作例
Fig. 4 Example multiply-add operations of Fig. 2.

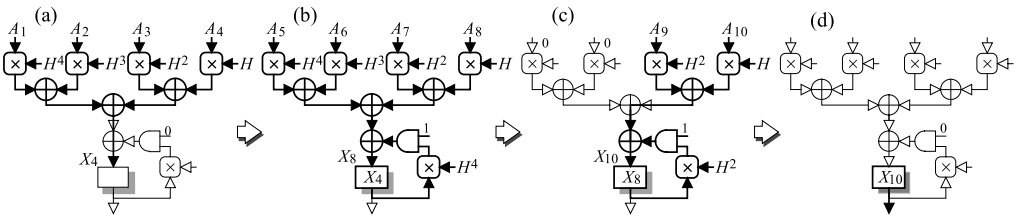

図 5 図 3 の積和演算器の動作例
Fig. 5 Example multiply-add operations of Fig. 3.

$$\begin{cases} T_5 = (T_1 \oplus A_5)H^4 = (A_1H^4 \oplus A_5)H^4 \\ T_6 = (T_2 \oplus A_6)H^4 = (A_2H^4 \oplus A_6)H^4 \\ T_7 = (T_3 \oplus A_7)H^4 = (A_3H^4 \oplus A_7)H^4 \\ T_8' = (T_4 \oplus A_8)H^4 = (A_4H^4 \oplus A_8)H \end{cases} \quad (8.b)$$

$$\begin{cases} T_8'' = T_8'H = (A_4H^4 \oplus A_8)H^2 \\ T_9' = (T_5 \oplus A_9)H = ((A_1H^4 \oplus A_5)H^4 \oplus A_9)H \\ T_{10} = (T_6 \oplus A_{10})H = ((A_2H^4 \oplus A_6)H^4 \oplus A_{10})H \end{cases} \quad (8.c)$$

$$\begin{cases} T_8 = T_8''H = (A_4H^4 \oplus A_8)H^3 \\ T_9 = T_9'H = ((A_1H^4 \oplus A_5)H^4 \oplus A_9)H^2 \end{cases} \quad (8.d)$$

$$\begin{aligned} X_{10} &= T_7 \oplus T_8 \oplus T_9 \oplus T_{10} \\ &= ((A_1H^4 \oplus A_5)H^4 \oplus A_9)H^2 \\ &\quad \oplus((A_2H^4 \oplus A_6)H^4 \oplus A_{10})H \\ &\quad \oplus(A_3H^4 \oplus A_7)H^4 \oplus (A_4H^4 \oplus A_8)H^3 \end{aligned} \quad (8.e)$$

次に図 5 の計算過程を順に説明する．こちらは H から H^4，H^3，H^2 を事前計算しなければならない．まず (9.a) で入力 A_1〜A_4 にそれぞれ H^4〜H を乗じ，それらを XOR 加算することで X_4 を生成する．次に (b) でも入力 A_5〜A_8 にそれぞれ H^4〜H を乗じ，またレジスタに保持している X_4 に H^4 を乗じてそれらをすべて加え合わせて X_8 を得る．(c) で入力されるのは A_9 と A_{10} の 2 ブロックなので，それらにはそれぞれ H^2 と H を，また X_8 には H^2 を乗じ，それらを合計した結果 X_{10} をレジスタに保存する．なお入力ブロック数が q の倍数でない場合は，この A_9 と A_{10} のように，最後の端数ブロックの入力位置を右詰めにする必要がある．そして，最後に (d) でレジスタから X_{10} を出力する．

$$X_4 = A_1H^4 \oplus A_2H^3 \oplus A_3H^2 \oplus A_4H \quad (9.a)$$

$$\begin{aligned} X_8 &= X_4H^4 \oplus (A_5H^4 \oplus A_6H^3 \oplus A_7H^2 \oplus A_8H) \\ &= A_1H^8 \oplus A_2H^7 \oplus A_3H^6 \oplus A_4H^5 \\ &\quad \oplus A_5H^4 \oplus A_6H^3 \oplus A_7H^2 \oplus A_8H \end{aligned} \quad (9.b)$$

$$\begin{aligned} X_{10} &= X_8H^2 \oplus (A_9H^2 \oplus A_{10}H) \\ &= A_1H^{10} \oplus A_2H^9 \oplus A_3H^8 \oplus A_4H^7 \oplus A_5H^6 \\ &\quad \oplus A_6H^5 \oplus A_7H^4 \oplus A_8H^3 \oplus A_7H^2 \oplus A_8H \end{aligned} \quad (9.c)$$

ここで，図 2 と図 3 のどちらのアーキテクチャを採用すべきかについて考察する．まず回路規模であるが，演算器部分に関しては，$GF(2^{128})$ 乗算器は 128 ビット XOR 加算器のおよそ 128 倍の規模となるため，乗

算器が 1 つ少ない図 2 の方が有利である．また，乗算器への定数入力を保持するレジスタは省略されているが，図 2 では H と H^4 を各乗算器でシェアすると 128 ビットレジスタが 2 セット必要であり，図 3 では，H〜H^4 と H^k で 5 セットを要する．途中結果保持用と合わせると図 2 の方がレジスタは 1 セット多いものの，乗算器のエリアを考慮すると小型化に向いているのは図 2 のアーキテクチャとなる．

次に動作速度について考える．図 2 のクリティカルパスは乗算器と XOR 1 段なのに対し，図 3 では XOR ツリーのゲートが 3 段と 2 段分多い．128 ビット乗算器のクリティカルパスのロジックは AND が 1 つと XOR が 10 段程度なので，XOR 2 段分のゲート遅延はそれほど大きなペナルティにはならない．しかし，この XOR ツリーは 128 ビットと幅が広くかつ長い配線を各ブロックから集まるため，大きな配線遅延を生じる可能性がある．LSI の微細化技術の進歩によって配線の寄生容量と抵抗が増大し，現在ではトランジスタのスイッチング遅延よりも配線遅延の方が支配的となってきている．特にこの XOR ツリーのように太く長いバスにおいては，配線遅延がゲート遅延の数倍になることも珍しくない．もちろん，この XOR ツリーの各段に T_i 加算の途中結果を保持するためのレジスタを挿入し，パイプライン化による高速化を図ることも可能である．しかしその場合，4 並列実装では 384 ビット，n 並列では $(n-1)\times 128$ ビットという大きなレジスタを付加する必要が生じてしまう．これに対して，図 2 では XOR ツリーの前にレジスタが入り，さらに XOR ツリーを通ってレジスタ RegX にデータがラッチされるのは，最終結果を出力するときの 1 回だけなので数クロックかけてもよく，この最終段を高速にする必要はない．積和演算のループ処理時はこの XOR ツリー部の入力をゲートして消費電力を抑えることも可能である．一方，図 3 は，途中結果をレジスタ RegX に毎回ラッチするため，この部分を高速動作させる必要があり，消費電力が大きくなる可能性が高い．

以上の考察から，図 2 のアーキテクチャは図 3 に対して，回路規模と動作速度の双方で優れているといえる．そこで，本論文の並列 GCM 回路の積和演算器には，図 2 のアーキテクチャを採用する．

5. GCM の並列回路アーキテクチャ

図 6 に暗号化部と $GF(2^{128})$ 積和演算器を実装した，4 並列の GCM 回路アーキテクチャを，また**図 7** にその動作例を示す．6 章の AES を用いた実装のバス幅は，鍵生成部 KeyGen のラウンド鍵出力の 128 ビット × 15 を除いて，すべて 128 ビットである．暗号化部 Enc はパイプラインアーキテクチャとし，4 入力 $Din1$〜4（平文または暗号文）を並列に受け取り，4 出力 $Dout1$〜4（暗号文または平文）をクロックごとに出力する．CTR モードで用いるレジスタ Y1〜4 は，ブロック数に合わせて，+4 ずつカウントアップする．暗号のパイプライン処理では，すべてのラウンド鍵を事前計算してレジスタに保存しておく必要があるが，それらは 4 つのブロックで共通に使用できる．“Mask” はデータブロックが 128 ビットに満たなかった場合にマスクパターンを生成するためのものであるが，データの流れのコントロールにも用いる．

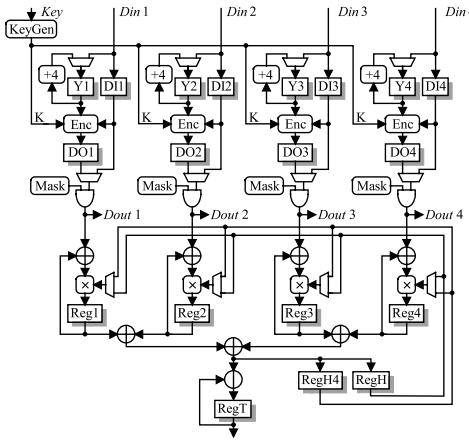

図 6 GCM 並列回路アーキテクチャ（$q=4$）
Fig. 6 Proposed parallel GCM hardware architecture ($q=4$).

図 7 は図 1 で示した $m=5$，$n=4$ の場合の演算を行うときの，初期パラメータ設定と，最終処理のデータの流れを示している．まず図 7 (a) では Y_0 を暗号化した C_0（$=Enc(K,Y_0)$）をレジスタ RegT にラッチする．このときデータが通るパスにある乗算器の一方の入力は 1 としておく．次に (b) でレジスタ Y1 をクリアしてデータ 0 を暗号化し，生成した H を 2 つのレジスタ RegH と RegH4 にラッチする．これらのレジスタの出力を (c)〜(d) のように乗算器にフィードバックして，積和演算に用いる定数 H^4 を生成する．次に (e) では H を乗算器に入力し，(f)〜(h) の最終処理に備えて次の値をレジスタに保存しておく．

$$C_X=(Len(A)||Len(C))H\oplus Enc(K,Y_0) \quad (10)$$

ブロック数 $n=5$，$m=4$ なので，(f) では左端に A_5，その右側に平文ブロック P_1〜P_3 が入力されている．そして，最終ブロック P_4 は (h) で左端への入力となる．式 (6) および図 4 から，積和演算におけ

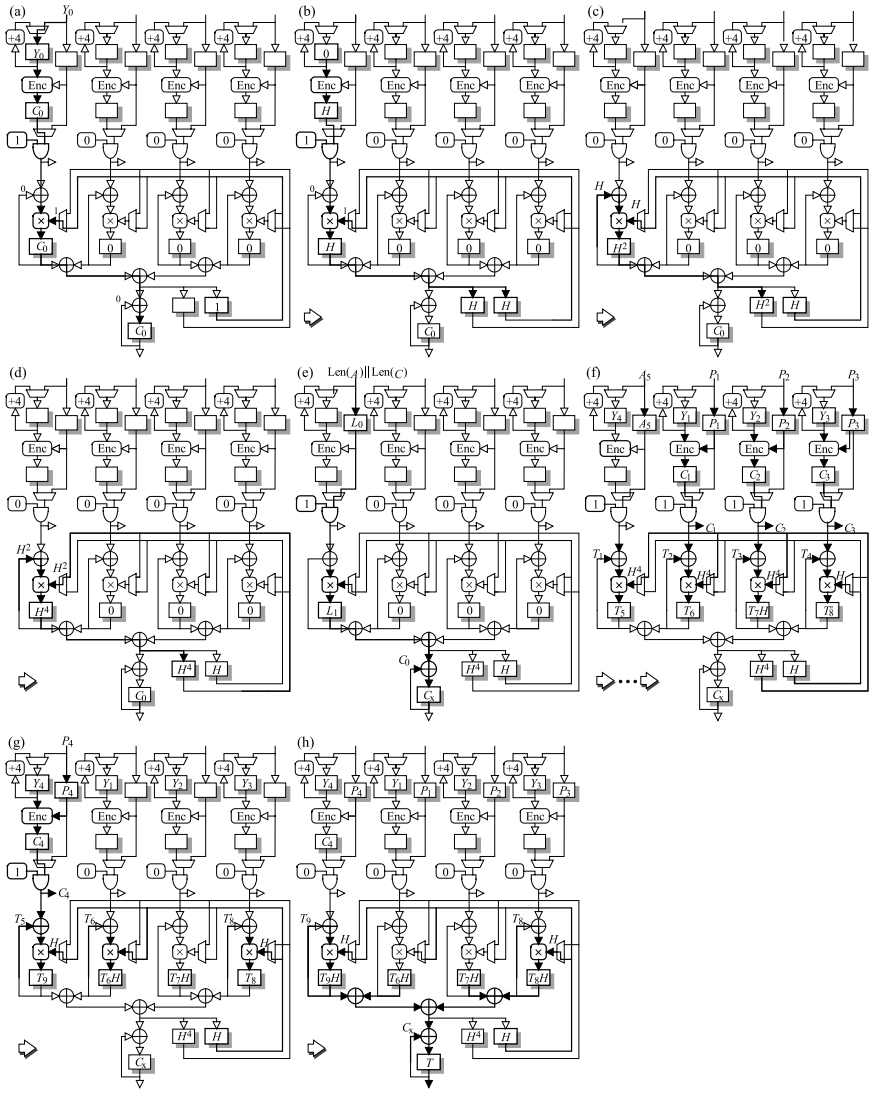

図 7 提案 GCM 並列回路アーキテクチャの動作例（$q = 4$, $m = 5$, $n = 4$）
Fig. 7 Example operations of the proposed parallel GCM hardware architecture ($q = 4, m = 5, n = 4$).

る最終 4 ブロック P_1〜P_4 には，それぞれ H^4，H^3，H^2，H を乗じることになっていたが，ここでは H^5，H^4，H^3，H^2 としている．これは，図 1 の最後の Len $(= Len(A)||Len(C))$ との積和演算における H と乗算を，式 (10) ですでに行っているからである．これらのレジスタを最後に (g) で XOR 加算することで認証子 T が得られる．

6. ASIC 性能評価

提案アーキテクチャによる 4 並列構成の GCM 回路を，0.13μm CMOS スタンダードセルライブラリにより実装評価した結果を，従来方式とともに**表 1** に示す．暗号化部は筆者の文献[8] に示した AES の 14 段のパイプライン方式で，鍵長は 128/192/256 ビットをサポートしている．また S-box は小型実装に向く合成体 $GF(((2^2)^2)^2)$ 上の演算[9] および高速動作に適した BDD（Binary Decision Tree）ロジックを用いた．合成体版は回路規模優先と動作速度優先の 2 種類を，BDD 版は速度優先だけで論理合成を行った．配置配線は行っていないが，仮想配線長は回路規模に応じて自動的に調節され，遅延時間が評価されている．なおスループットは初期設定や最終処理を含めずに，データが連続的に処理される定常状態における値を示している．また表中の “回路効率” は，2 入力 NAND ゲー

表 1 GCM 回路の ASIC 性能比較
Table 1 Performance comparison of GMC hardware in ASIC.

文献	GCM アーキテクチャ	AES アーキテクチャ	ラウンド関数ブロック数	鍵長 (bits)	S-Box	クリティカルパス (ns)	最適化	動作周波数 (MHz)	スループット (Gbps)	回路規模 (gates)	回路効率 (Kbps /gate)	ライブラリ (μm)
本実装	4 Parallel	Pipelined	14×4	128 ~ 256	合成体	5.00	規模	200.0	102.40	600,440	170.54	0.13
						4.00	速度	250.0	128.00	697,567	183.49	
					BDD	3.15		317.5	162.56	979,348	165.99	
[8]	Sequential	Pipelined	14	128 ~ 256	合成体	5.00	規模	200.0	25.60	174,016	147.11	0.13
						4.00	速度	250.0	32.00	181,198	176.60	
					BDD	3.00		333.3	42.67	297,542	143.40	
		4-stage pipelined loop	4	128 ~ 256	合成体	5.00	規模	200.0	6.40	73,104	87.55	
						4.00	速度	250.0	8.00	79,566	100.55	
					BDD	3.00		333.3	10.67	118,645	89.90	
	4 Parallel	Loop	4	128 ~ 256	合成体	5.00	規模	200.0	6.40	96,241	66.50	
						4.00	速度	250.0	8.00	106,893	74.84	
					BDD	3.00		333.3	10.67	162,373	65.69	
[10]	Sequential	Pipelined	10	128		3.69		271.0	34.69	498,658	69.57	0.18
[11]				128~256		3.33		300.0	7.00	97,000	72.16	
				128					10.00	190,000	52.63	
				128					20.00	180,000	111.11	
									40.00	330,000	121.21	

ト 1 個あたりのスループットで，この値が大きいほど性能が高いことになる．

S-box に BDD を用いた 4 並列の本実装では 162 Gbps ときわめて高いスループットが得られた．また，小型の合成体 S-box による実装でも 100 Gbps 以上が出ており，回路効率の点ではこちらの方が優れている．従来実装の ASIC ライブラリは文献 10) では 0.18 μm，また文献 11) では示されていないため，回路効率をそのまま比較することはできない．しかしながら，本論文のパイプライン処理の AES 回路による高速実装は文献 8) も含めて，128〜256 ビット鍵やマスク処理などもサポートしていることから，その回路規模と速度のパフォーマンスの優位性は明らかである．また，今回の実装の基本演算ブロックは文献 8) の 14 ブロックバージョンと同じであるが，回路効率は高い．これは鍵スケジューラなどが 4 つのメインブロックでシェアできるためである．**表 2** は図 6 の本実装を，AES 回路+マスク生成+積和演算器のメインブロック，鍵生成部 KeyGen，そして Reg1〜4 の出力を加算する XOR アレイや制御回路を含む残りの部分に分けて，そのゲート数を示したものである．たとえば 200 MHz 動作でメインブロックを 1 つ追加するとスループットを 25.6 Gbps 増やすことができ，そのときの回路の増分は約 144 Kgates（XOR アレイに 128 個の XOR が新たに必要となるが，これはわずか 200 gates である）となる．本アーキテクチャはブロック数を増やして並列度を上げても，クリティカルパス（積和演算部）は基本的に長くならないため，回路規模に応じてスループットをスケーラブルに高められるという利点も有している．

表 2 GCM 回路の内訳と 1 AES 回路ブロックあたりのスループット
Table 2 Sizes of each part of GCM hardware and throughput per one AES circuit block.

動作周波数 (MHz)	回路規模 (gates)			スループット /block (Gbps)
	AES, Mask Multiplier	KeyGen	XOR-array Others	
200.0	144,136×4	17,742	6,119	25.600
250.0	155,127×4	17,742	6,146	32.768
317.5	238,863×4	17,742	6,154	406.400

7. むすび

本論文では，GCM の並列回路アーキテクチャを提案し，パイプライン処理の AES 回路を用いた ASIC 実装の性能評価を行った．本アーキテクチャは回路規模に応じて動作速度がスケーラブルに選択可能であり，4 並列実装では 162 Gbps ときわめて高いスループットを得ることができた．また，回路効率も，従来実装に比べ高い値が実現された．

しかし本アーキテクチャでは，並列度を 1 つ上げるごとに，データ I/O 数が 256 本ずつリニアに増加するという問題がある．並列化の代わりにパイプライン段数を増やして動作周波数を上げることを考えるとき，CTR モードの AES は問題とならないが，$GF(2^{128})$ 積和演算器は出力結果をすぐ次のサイクルの入力として使用するため，そのままではパイプライン化ができない．そこで今後は，積和演算器をパイプライン化す

る手法の提案と，それを用いた GCM 回路の実装評価について報告していきたい．

参 考 文 献

1) McGrew, D.A. and Viega, J.: The Galois/Counter Mode of Operation (GCM) (May 2005). http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-revised-spec.pdf
2) NIST: Recommendation for Block Cipher Modes of Operation: Methods and Techniques, Special Publication 800-38A (Dec. 2001). http://csrc.nist.gov/CryptoToolkit/modes/800-38_Series_Publications/SP800-38A.pdf
3) NIST: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) for Confidentiality and Authentication, Draft Special Publication 800-38D (Apr. 2006). http://csrc.nist.gov/publications/drafts/Draft-NIST_SP800-38D_Public_Comment.pdf
4) NIST: Advanced Encryption Standard (AES) FIPS Publication 197 (Nov. 2001). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
5) Viega, J. and McGrew, D.A.: The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (EPS), RFC 4106 (June 2005). http://www.faqs.org/rfcs/rfc4106.htm
6) IEEE: 802.1AE—Media Access Control (MAC) Security, Draft 3.5 (June 2005). http://www.ieee802.org/1/pages/802.1ae.html
7) Kohno, T., Viega, J. and Whiting, D.: Carter Wegman (authentication) with Counter (encryption) (May 2003). http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/cwc/cwc-spec.pdf
8) Satoh, A.: High-Speed Hardware Architectures for Authenticated Encryption Mode GCM, *Proc. IEEE ISCAS2006* (May 2006).
9) Satoh, A., Morioka, S., Takano, K. and Munetoh, S.: A Compact Rijndael Hardware Architecture with S-box Optimization, *ASIACRYPT 2001*, LNCS 2248, pp.239–254 (Dec. 2001).
10) Yang, B., Mishra, S. and Karri, R.: High Speed Architecture for Galois/Counter Mode of Operation (GCM), Cryptology ePrint Archive: Report 2005/146 (June 2005). http://eprint.iacr.org/2005/146.pdf
11) Elliptic Semiconductor Inc.: CLP-15/-16/-24 AES-GCM Core Preliminary Data Sheet (2004). http://www.ellipticsemi.com/

(平成 18 年 10 月 2 日受付)
(平成 19 年 4 月 6 日採録)

佐藤 証（正会員）

昭和 39 年生．平成元年早稲田大学大学院理工学研究科電気工学専攻修士課程修了．同年日本アイ・ビー・エム（株）東京基礎研究所入所．平成 11 年早稲田大学より博士（工学）授与．平成 19 年より（独）産業技術総合研究所情報セキュリティ研究センターに勤務．情報セキュリティに関するアルゴリズムおよび，その高性能 VLSI 実装方式に関する研究に従事．電子情報通信学会会員