

暗号化アルゴリズム Rijndael のハードウェア実装と評価

下村 高範[†] 阿部 公輝[†]

[†] 電気通信大学情報工学科

〒182-8585 調布市調布ヶ丘 1-5-1

E-mail: †{shimom-t,abe}@cacao.cs.uec.ac.jp

あらまし 暗号化アルゴリズム Rijndael を VLSI に実装するときの回路規模とスループットを評価した。Verilog-HDL を用いてアルゴリズムを記述し、論理合成ツールにかけて面積と遅延時間を求めた。S-Box は case 文を用いた関数テーブルとして記述した。0.35 μ m CMOS スタンダードセルライブラリを用いて合成した結果、128 ビット鍵の Rijndael アルゴリズムが、26.3Gbps の処理速度で実装できることが分かった。ゲート当たりスループットは 2 入力 NAND 換算で 97.4Mbps/Kgate となった。

キーワード AES, Rijndael, ブロック暗号, ハードウェア実装

Hardware Implementation of Rijndael and Its Evaluation

Takanori SHIMOMURA[†] and Kôki ABE[†]

[†] Department of Computer Science, The University of Electro-Communications

1-5-1 Chofugaoka, Chofu-shi, Tokyo, 182-8585, Japan

E-mail: †{shimom-t,abe}@cacao.cs.uec.ac.jp

Abstract The area and time of VLSI implementations of the Rijndael block cipher algorithm have been evaluated by describing the algorithm in Verilog-HDL and synthesizing the descriptions. The S-Box was described as a function table using case statements. Using a 0.35 μ m CMOS standard-cell library for the synthesis, we obtained a VLSI realization which performs the encryption at a rate of 26.3 Gbps with the key length of 128 bits. The throughput per 2-input NAND gate of the implementation was found to be 97.4Mbps/Kgate.

Key words AES, Rijndael, Block cipher, Hardware implementation

1. はじめに

インターネットは社会活動を支える基盤環境にまで成長してきており、セキュリティ技術はこの基盤環境に信頼を与える意味で重要である[1]。暗号技術はセキュリティ技術の根幹を成す。

米国標準の共通鍵ブロック暗号 DES [2] は、現在では全数探索で鍵を見つけることができるようになり、すでに暗号としての役割を終えた。DES に代る次期の共通鍵暗号 AES [3] として、Rijndael [4] が採用されている。

Rijndael の実装については、標準的なソフトウェアによるもの [5] の他に、処理を高速化するため、ハードウェア実装も試みられてきた [6]~[8]。Rijndael の SPN (Substitution-Permutation Networks) 構造では、S-Box の規模が大きく、回路リソースが大きくなる傾向がある。モバイル機器への応用の点から小型化が望まれ、回路規模を小さくするための工夫がなされている [6]。また、AES の他の候補との性能比較を目的として、Rijndael の FPGA 実装も試みられた [8]。

一方ハイエンドサーバなどでは、高スループットが要求され、ハードウェア化による恩恵は大きい。また、資源の有効利用という点からは、スループットと回路規模との比も重要である。本研究では、必ずしも回路規模そのものの縮小を求めず、スループット、および、単位ゲートあたりのスループットに着目する。

スループットを上げるためには、ループ処理でなくパイプライン処理が有効である。パイプライン処理は、フィードバックをしない ECB モードなどで利用できる。また、FPGA による実装では性能に限界がある。

本稿では、まず、Rijndael 暗号化アルゴリズムをパイプライン処理するハードウェアを Verilog-HDL で記述し、標準的な論理合成ツールを用いて、0.35 μ m スタンダードセルライブラリにより回路合成した実験の結果を述べる。S-Box は、case 文により関数テーブルとして記述する。この方法によって実現される、1 ラウンド 1 ステージとするパイプライン実装の単位ゲートあたりスループットを求め、これまでのループ処理に基づく実装の結果 [7] と比較する。また、パイプライン 1 ステージ分を構成

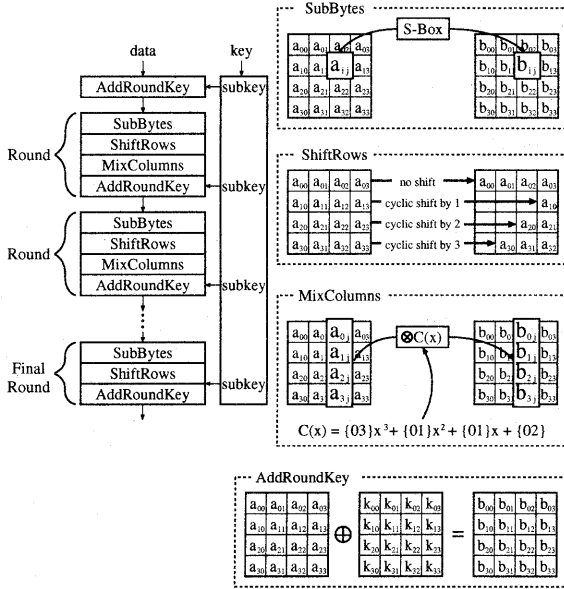


図1 Rijndael の暗号化アルゴリズムの構成

する個々のモジュールについて、合成時の遅延時間の条件を変え、合成回路の面積と遅延時間を求める。とくに S-Box の性能を詳しく調べ、逆元計算によるもの [6] との比較を行う。FPGA への実装例 [8] との比較も行う。

2. Rijndael 暗号化アルゴリズム

Rindael のアルゴリズム構造は図 1 のように表される。データブロック長 N_c と鍵長 N_k は、1 ワードを 32 ビットとして、4, 6, 8 ワードの中から選ぶことができる。実装を含め、以下では $N_c = N_k = 4$ (128 ビット) とする。

128 ビット (16 バイト) のデータは、各 1 バイトの要素 4×4 個からなる行列とみなされる。この行列に対し、アルゴリズムは SubBytes, ShiftRows, MixColumns, AddRoundKey からなる変換を繰り返す。(最後の繰返しでは SubBytes は行わない。また、繰返しに入る前に AddRoundKey を 1 回行う。) 変換を繰り返すラウンド数はビット長によって異なり、 $N_c = N_k = 4$ の場合、10 である。

図 1 の SubBytes は、行列の各要素に対し、1 バイトから 1 バイトへの置換を行う S-Box 16 個の並列操作を表す。S-Box は、入力バイトのガロア拡大体 $GF(2^8)$ 上の既約多項式

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad (1)$$

に関する乗法逆元 b を求め、その結果に次式で定義される affine 変換を行う。

$$b_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \quad (2)$$

ここに、 $0 \leq i < 8$, b_i は b のビット i の値、 c_i は定数バイト 01100011 のビット i の値である。S-Box が特定のアルゴリズム

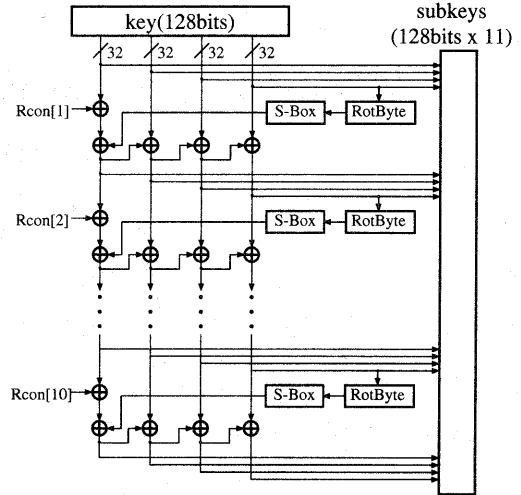


図2 副鍵スケジューリング

で構成される点で、S-Box が完全ランダムであった DES とは異なっている。

ShiftRows は、行列の後ろ 3 行の各要素に対し、巡回シフト $s_{r,c} = s_{r,(r+c) \bmod 4}$ を行う。ここに、 $0 < r < 4, 0 \leq c < 4$ 。

MixColumns は、行列の各列を 3 次の多項式とみなし、 $x^4 + 1$ を法として次式との積をとる。

$$c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

この積の効果は、列ベクトルに対し次の変換を行うことと等価である。

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad (3)$$

AddRoundKey は、各ラウンド用に生成された副鍵 (subkey) との排他的論理和をとる操作である。

鍵スケジュールは、128 ビットの場合、図 2 のようになっており、これにより、各ラウンドで使用される副鍵が算出される。例えば、128bit の鍵は 128×11 bit の副鍵に拡張され、各ラウンドへ分配される。各副鍵は、S-Box を 4 つ使用し、オンザフライで生成される。図において RotByte は入力ワード $[b_3, b_2, b_1, b_0]$ を $[b_0, b_3, b_2, b_1]$ へ変換する。また、 $Rcon[i]$ は定数値 $[x^{i-1}, 0, 0, 0]$, $0 < i < 11$, を表す。ここに、 x^{i-1} は $GF(2^8)$ の元である。

3. 設 計

前節で述べた Rijndael の暗号化アルゴリズムを、図 3 に示すように、各ラウンドを 1 ステージとする 11 段のパイプライン構造 AllRoundPipe として記述する。記述には Verilog-HDL を用いる。

AllRoundPipe は

入力: 128 ビットテキストデータと 128 ビット鍵

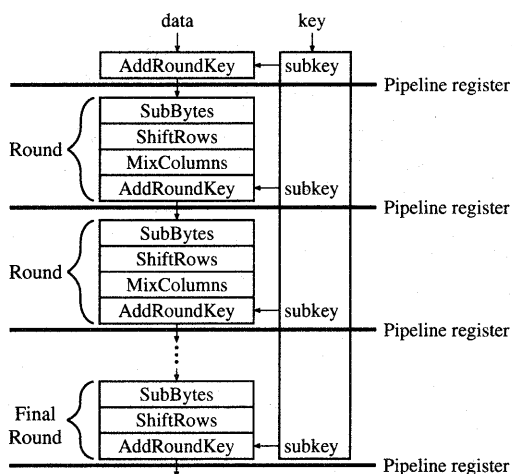


図3 AllRoundPipeのパイプライン構造

出力:128 ビット暗号データ
を持ち、各ステージは次のモジュールから成る。

- SubBytes
- ShiftRows
- MixColumns
- AddRoundKey
- SubKey

SubBytes は 16 個の S-Box を含み、SubKey は 4 個の S-Box を含む。モジュール S-Box は case 文を用いて、大きさ 16×16 の 1 バイト値関数テーブルとして記述する。

ShiftRows は入力データ行列の行に関するビットシフトである。本実装では入力データ長を 128bit に固定してあるので、それに対応したシフト (線のつなぎ変え) で記述する。

MixColumns は入力データ行列の各列に対し式 (3) の計算をする。ここで和の計算は、排他的論理和を用いる。8 ビット単位で乗算を行う際、オーバーフローすることがある。GF(2^8) の演算を考慮し、このときは定数 00011101 との排他的論理和をとる。

AddRoundKey は各ラウンドで使用する副鍵とデータとの排他的論理和で記述する。

SubKey は 1 ラウンド分の副鍵を生成する。前処理を行う RotByte モジュールと 4 個の S-Box からなる。RotByte は、バイト単位のシフトで記述する。

4. 論理合成の結果

前節で述べた Rijndael 暗号化モジュール AllRoundPipe に対し、CAD ツール DesignCompiler を用いて論理合成を行った。ツールのバージョンは 2000.05 である。合成に用いたライブラリは、ローム社の製造条件に基づき VDEC(東京大学大規模集積システム設計教育研究センター) で製作されたものである。主な製造条件は、CMOS 0.35 μ m, PolySi 2 層, メタル配線 3 層, 電源電圧 5V である。

図4にパイプライン1段分の組合せ回路について、遅延時間

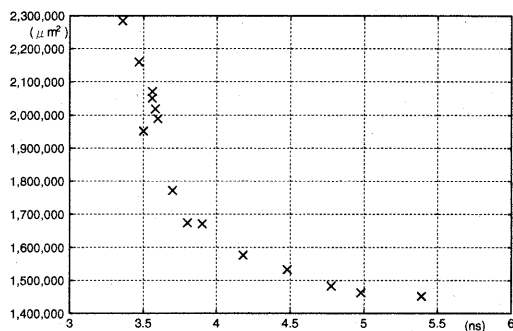


図4 パイプライン1段分組合せ回路の合成結果

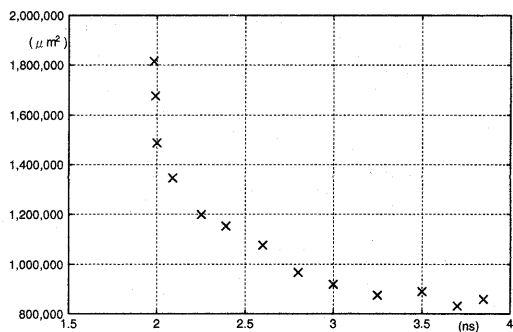


図5 SubByteの合成結果

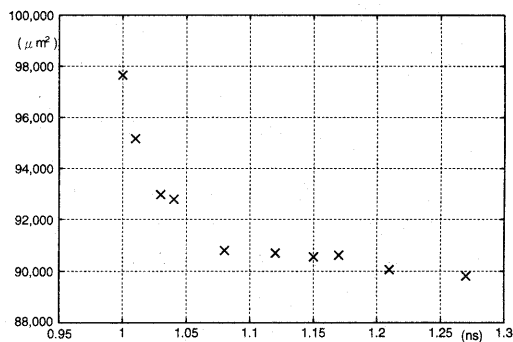


図6 MixColumnの合成結果

の制約条件を変えて合成したときの結果を示す。

また、パイプライン1段を構成するモジュール SubBytes, MixColumns, SubKey について、遅延時間の制約を変えて合成したときの結果を、それぞれ図5, 図6, 図7に示す。AddRoundKey は、排他的論理和の演算のみからなる単純な回路であり、遅延時間の制約を変えても合成結果に変化はみられず、面積 22,144 (μm^2), 遅延時間 0.29 (ns) となる。

表1に、AllRoundPipe と各構成モジュールについて、もっとも性能バランスの良い (原点に近い) 合成結果の速度と面積をまとめて示す。ShiftRows は単なる線のつなぎかえなので、表には含めていない。

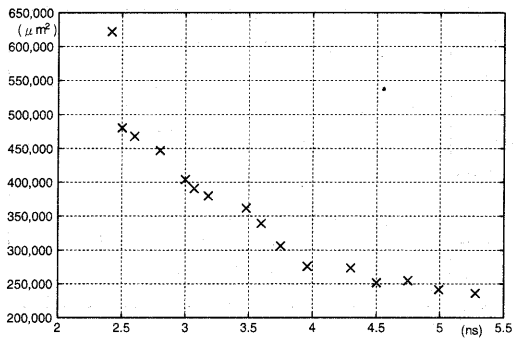


図7 SubKeyの合成結果

表1 AllRoundPipe, および、パイプライン1段を構成するモジュールの回路面積と遅延時間

modules	area (μm^2)	time (ns)
AllRoundPipe	20,028,873	4.87
組合せ回路	1,674,743	3.80
SubByte	1,198,802	2.25
MixColumn	92,802	1.04
AddRoundkey	22,144	0.29
SubKey	390,417	3.07
パイプラインレジスタ	94,976	0.40

5. 評価と考察

前節の結果から、スループットと単位ゲート当たりのスループットを求めると、表2のようになる。表には他の実装[6]～[8]との比較も示す。文献[6]の実装では、ガロア体演算の最適化を利用した逆元計算によりS-Boxを求め、1ループ1ラウンドのループ処理を行う。文献[7]の実装では、S-Boxは、case文により関数テーブルとして記述され、1ループ1ラウンドのループ処理を行う。文献[8]ではパイプライン処理を用いたFPGA実装をしている。

表から、本実装のスループットは他の実装と比べ非常に高いことが分かる。

本実験では暗号化処理のみを実装しているのに対し、他の実装では復号化処理も実装している。復号化もほぼ同程度の面積で実装できることを考慮すると、本実装は、S-Boxを関数テーブルとして記述実装しループ処理を行うものより6倍程度良い。逆元計算によりS-Boxを実装しループ処理を行うものは、暗号化モジュールと復号化モジュールのほとんどを共用している。したがって、暗号化と復号化を同時にはできないことを考慮に入れると、単位ゲート当たりのスループットは本実装と同程度である。FPGAへの実装でもかなりのスループットが得られるが、本実装の27万ゲート(2入力NAND換算)に対し、112万ゲート相当のFPGAチップを3個使用する必要がある[8]。

論理合成ツールやライブラリも異なるので、単純には比較できないが、S-Boxを関数テーブルとして記述し自動合成する場合でも、良い結果が得られることが分かる。1ラウンドのみを実

表2 スループットと単位ゲートあたりのスループット

	throughput (Mbps)	throughput/gate (Mbps/Kgate)
本実装	26,280	97.4
文献[6]の実装	1,780	83.3
文献[7]の実装	510	15.1
文献[8]の実装	12,160	-

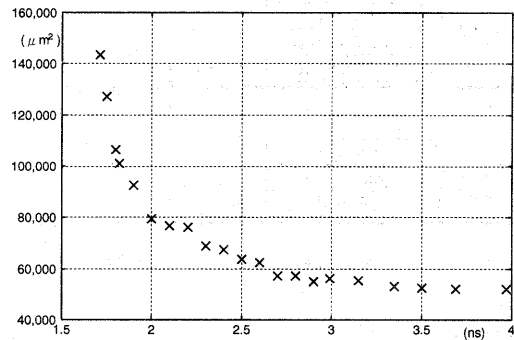


図8 S-Boxの合成結果

表3 S-Boxの他の実装との比較

	area (gates)	time (ns)	area*time (gates*ns)
本実装	1,069	2.00	2,138
文献[6]の実装	445	4.35	1,936
文献[7]の実装	620	5.13	3,181

装しループ処理をする場合、リソースの共有をするための制御に余分な回路を必要とする。したがって、ループ処理ではパイプライン処理に比べて、単位ゲート当たりのスループットは下がる傾向にある。

S-Boxの特性を詳しく調べるため、遅延時間の制約条件を変えてS-Boxを合成したときの結果を図8に示す。S-Boxの合成結果を他の実装と比較して、表3に示す。本実装におけるS-Boxの面積と遅延時間の積は、文献[6]の値にほぼ匹敵するものとなっている。

SubBytes, MixColumns, AddRoundKeyを通るパイプライン1ステージのクリティカルパスの遅延時間のうち、SubBytesは63%を占める。S-Boxの遅延時間はSubBytesの遅延時間の88%であるから、S-Boxの性能が全体の性能を大きく左右していることが分かる。

S-Boxをcase文を用いて関数テーブルとして記述し論理合成する方法は素朴であるが、高いスループットをもたらし、性能コスト比もかなり良い。面積をさほど重視しない応用の場合は、この方法が有効である。

6. おわりに

暗号化アルゴリズムRijndaelをVLSIに実装するときの回路規模とスループットを評価した。Verilog-HDLを用いてアルゴリズムを記述し、論理合成ツールにかけて面積と遅延時間

を求めた。0.35 μ m CMOS スタンダードセルライブラリを用いて合成した結果、128 ビット鍵の Rijndael アルゴリズムが、26.3Gbps の処理速度で実装できることが分かった。ゲート当たりスループットは 2 入力 NAND 換算で 97.4Mbps/Kgate となった。S-Box を case 文を用いて関数テーブルとして記述し論理合成する方法は素朴であるが、高いスループットをもたらし、性能コスト比もかなり良いことが分かった。

謝 辞

有益な議論をし、貴重なコメントをいただいた楯岡孝道博士に感謝します。また、合成ツールの使用に関し、貴重な助言をいただいた葛穀氏に感謝します。本研究は東京大学大規模集積システム設計教育研究センターを通しシノプシス株式会社の協力で行われたものである。

文 献

- [1] 山口英編, “ネットワークセキュリティ,” 情報処理学会誌, Vol.42, No.12, pp.1151–1190, Dec. 2001.
- [2] National Institute of Standards and Technology, “Data Encryption Standard (DES),” FIPS PUB 46-2, Dec. 1993.
- [3] National Institute of Standards and Technology, “Announcing the Advanced Encryption Standard (AES),” FIPS PUB 197, Nov. 2001.
- [4] J. Daemen and V. Rijmen, “AES Proposal: Rijndael,” AES Algorithm Submission, <http://csrc.nist.gov/encryption/aes/Rijndael.pdf>, Sep. 1999.
- [5] B. Gladman, “Implementations of AES (Rijndael) in C/C++ and Assembler, http://fp.gladman.plus.com/cryptography_technology/rijndael, Sep. 2002.
- [6] 佐藤証, 森岡澄夫, 宗藤誠治, “Rijndael の小型ハードウェア実装,” マルチメディア, 分散, 協調とモバイル (DICOMO) シンポジウム, pp.663–668, Jun. 2001.
- [7] 市川哲也, 時田俊雄, 松井充, “128 ビットブロック暗号のハードウェア実装について (III),” 暗号と情報セキュリティシンポジウム 2001 予稿集, pp.669–674, Jan. 2001.
- [8] P. Chodowiec, P. Khuon, and K. Gaj, “Fast Implementations of Secret-Key Block Ciphers Using Mixed Inner- and Outer-Round Pipelining,” Proc. ACM/SIGDA 9th International Symposium on Field Programmable Gate Array, pp.94–102, Feb. 2001.