SCIS 2006 The 2006 Symposium on Cryptography and Information Security
Hiroshima, Japan, Jan. 17-20, 2006
The Institute of Electronics, Information and Communication Engineers

積和演算器に基づくスケーラブル高基数モンゴメリ乗算器の設計と評価

Design and Evaluation of Scalable High-Radix Montgomery Multipliers based on Multiply Accumulators

宮本 篤志* 本間 尚文* 青木 孝文* 佐藤 証†
Atsushi Miyamoto Naofumi Homma Takafumi Aoki Akashi Satoh

あらまし 本稿では，スケーラブルな高基数モンゴメリ乗算器を設計し，ASIC ライブラリを用いてその評価を行う．高基数モンゴメリ乗算アルゴリズムは，基数 2 のアルゴリズムに比べ演算のステップ数を抑えた構成が可能である．これをモンゴメリ乗算器に適用する場合，積和演算器を基本としたスケーラブルアーキテクチャとすることで，演算速度や回路面積に優れたハードウェアが実現できる．さらに，積和演算器のハードウェアアルゴリズム（算術アルゴリズム）を網羅的に設計し，その違いによるハードウェアの性能の比較評価を行う．この結果を用いれば，要求される性能に応じて積和演算器の算術アルゴリズムとそのワード長を変更し，最適なモンゴメリ乗算器を設計することが可能となる．

キーワード 公開鍵暗号, 高基数モンゴメリ乗算，積和演算器，ハードウェアアルゴリズム

1 はじめに

身の回りのあらゆる情報機器がネットワークを介して結合されるユビキタス情報社会においては，個人情報の保護や高信頼な電子商取引が必須であり，情報セキュリティをいかに構築するかが重要な課題となる．特に，鍵配送および鍵管理，認証などに適した公開鍵暗号方式は，その基盤技術として不可欠である．RSA 暗号に代表される公開鍵暗号は，乗剰余演算を中心とした膨大な多倍長計算を必要とするため，高速動作が必要とされるサーバーや組み込み用途などでは，コスト面から専用ハードウェアによる実装が求められている．

乗剰余演算アルゴリズムの中でも，ハードウェア実装が有効なものとして，モンゴメリ乗算アルゴリズム [1] が知られている．モンゴメリ乗算アルゴリズムは，除算を行わずに加算とシフト演算で効率的に乗剰余演算を行うことができる．モンゴメリ乗算のハードウェアとしては，演算の基数を 2 としたアルゴリズムに基づく構成が従来より多く提案されている [2],[3]．基数 2 モンゴメリ乗算アルゴリズムは，加算の繰り返しにより実現されるため，そのハードウェアは加算器を用いた構成となる．一方で，近年，基数を 8，16，32 といった高基数に拡張したモンゴメリ乗算アルゴリズムが提案されている [4]．高基数モンゴメリ乗算アルゴリズムでは，積和演算を基本としたハードウェア構成が可能であることが示されている [5],[6]．

本稿では，スケーラブル高基数モンゴメリ乗算器の設計と評価について述べる．設計する高基数モンゴメリ乗算器のアーキテクチャは，演算のワード長を 8〜128 ビットの範囲で自由に設定でき，要求される性能に応じた構成が可能である．また，積和演算器に基づくアーキテクチャとすることで，従来の加算器による実装に対して演算速度や回路面積に優れた性能を実現する．

モンゴメリ乗算器の回路規模や動作速度は，積和演算器の性能に大きく左右される．また積和演算器には様々な構成法があり，その性能は用いたハードウェアアルゴリズム（算術アルゴリズム）に大きく依存する．そこで本稿では，積和演算器の算術アルゴリズムを網羅的に設計し，その違いによるモンゴメリ乗算器の性能を比較評価する．この結果より，演算のワード長を変更することに加え，積和演算器の算術アルゴリズムを適切に選択することで，要求される性能に応じた最適なモンゴメリ乗算器を設計することが可能となる．

本稿は，以下のように構成される．2 章では，モンゴメリ乗算アルゴリズムについて述べる．3 章では，スケーラブル高基数モンゴメリ乗算器と積和演算器の構成について述べる．4 章では，設計したモンゴメリ乗算器の性能を ASIC ライブラリにより評価する．最後に 5 章で本稿をまとめる．

* 東北大学大学院情報科学研究科 〒 980-8579 宮城県仙台市青葉区荒巻字青葉 6–6–05 Graduate School of Information Sciences, Tohoku University Aoba 6-6-05, Aramaki, Aoba-ku, Sendai-shi Miyagi 980-8579, Japan.
† 日本アイ・ビー・エム株式会社 東京基礎研究所 〒 242-8502 神奈川県大和市下鶴間 1623–14 IBM Research, Tokyo Research Laboratory, IBM Japan Ltd., 1623-14, Shimotsuruma, Yamato-shi, Kanagawa 242-8502, Japan.

ALGORITHM 1
MONTGOMERY MULTIPLICATION

Input:	$X, Y, N, R(=2^k)$, $W = -N^{-1} \bmod R$
Output:	$Z = XYR^{-1} \bmod N$

```
1 : t := XY · W mod R;
2 : Z := (XY + tN)/R;
3 : if (Z > N) then Z := Z − N;
```

ALGORITHM 2
HIGH-RADIX MONTGOMERY MULTIPLICATION

Input:	$X = (x_{m-1}, ..., x_1, x_0)_{2^r}$, $Y = (y_{m-1}, ..., y_1, y_0)_{2^r}$, $N = (n_{m-1}, ..., n_1, n_0)_{2^r}$, $W = -N^{-1} \bmod 2^r$
Output:	$Z = XY2^{-r \cdot m} \bmod N$

```
1 : Z := 0;
2 : for i = 0 to m − 1                          – Loop1
3 :   C := 0;
4 :   t_i := (z_0 + x_i y_0) W mod 2^r;
5 :   for j = 0 to m − 1                        – Loop2
6 :     Q := z_j + x_i y_j + t_i n_j + C;
7 :     if (j ≠ 0) then z_{j−1} := Q mod 2^r;
8 :     C := Q/2^r;
9 :   end for
10:   z_{m−1} := C;
11: end for
12: if (Z > N) then Z := Z − N;
```

2 モンゴメリ乗算アルゴリズム

本章では，モンゴメリ乗算アルゴリズムについて概説する．

モンゴメリ乗算では，2つの整数 X，Y に対し以下の演算を行う．

$$Z = XYR^{-1} \bmod N \tag{1}$$

ここで，X，Y，R，N は以下の関係式を満たす．

$$0 \le X, Y < N < 2^k = R \tag{2}$$

RSA 暗号などの公開鍵暗号系において，法 N は k=1,024 ～4,096 といった大きな数が用いられる．

ALGORITHM 1 に，Montgomery により提案された乗剰余演算アルゴリズム（モンゴメリ乗算アルゴリズム）[1] を示す．式 (1) において，XY の結果が R で割りきれるならば，N による mod 演算は必要なく，右 k ビットシフト演算で Z を求めることができる．**ALGORITHM 1** では，前処理で計算した W を用いて係数 t を求め，XY を R で割りきれる $XY + tN$ に補正する．ここで，tN は N の倍数であるため，N の剰余を求める式 (1) には影響しない．最後に，シフト演算により導出された Z は N より小さいとは限らないため，$Z > N$ である場合には，N による減算を一回行う．このように，モンゴメリ乗算アルゴリズムは，演算コストの高い除算を行わずに加算とシフト演算で乗剰余演算を効率的に行うことができる．

しかし，**ALGORITHM 1** では，常に k ビットの演算が必要となるため，RSA 暗号のように k が大きな場合の実装には向かない．その点を改良したアルゴリズムとして，演算のワード長を小さなワード（r ビット）に分割する手法が提案されている [3],[5]．本稿では，その中でも高基数 (2^r) のアルゴリズムについて考える．

ALGORITHM 2'
HIGH-RADIX MONTGOMERY MULTIPLICATION

Input:	$X = (x_{m-1}, ..., x_1, x_0)_{2^r}$, $Y = (y_{m-1}, ..., y_1, y_0)_{2^r}$, $N = (n_{m-1}, ..., n_1, n_0)_{2^r}$, $W = -N^{-1} \bmod 2^r$
Output:	$Z = XY2^{-r \cdot m} \bmod N$

```
1 : Z := 0;   V := 0;
2 : for i = 0 to m − 1
3 :   C := 0;
4 :   t_i := (z_0 + x_i y_0) mod 2^r;
5 :   t_i := t_i W mod 2^r;
6 :   for j = 0 to m − 1
7 :     Q := z_j + x_i y_j + C;
8 :     z_j := Q mod 2^r;   C := Q/2^r;
9 :   end for
10:   z_m := C;
11:   C := 0;
12:   for j = 0 to m − 1
13:     Q := z_j + n_j t_i + C;
14:     if (j ≠ 0) then z_{j−1} := Q mod 2^r;
15:     C := Q/2^r;
16:   end for
17:   Q := z_m + V + C;
18:   z_{m−1} := Q mod 2^r;   V := Q/2^r;
19: end for
20: C := 1;
21: for j = 0 to m − 1
22:   Q := z_j+!n_j + C;
23:   z_j := Q mod 2^r;   C := Q/2^r;
24: end for
25: if (C == 1 || V == 1) then return
26: C := 0;
27: for j = 0 to m − 1
28:   Q := z_j + n_j + C;
29:   z_j := Q mod 2^r;   C := Q/2^r;
30: end for
```

高基数モンゴメリ乗算アルゴリズム [5] では，標準の $r \times r$ ビット積和演算を用いるために，入力のビット幅（k ビット）をワードごとに m 分割する ($r \cdot m = k$)．ここで，r は 8 ～ 128 の範囲で 2 のべき乗となる値である．例えば，入力 X はワード x_i $(0 \le i \le m-1)$ によって以下のように表現される．

$$X = x_{m-1}2^{r(m-1)} + ... + x_1 2^r + x_0 \tag{3}$$

また本稿では，式 (3) を以下のように表す．

$$X = (x_{m-1}, ..., x_1, x_0)_{2^r} \tag{4}$$

ALGORITHM 2 に，高基数モンゴメリ乗算アルゴリズムを示す．ここで，一時変数 Q は $2r$ ビットであり，上位 r ビット，下位 r ビットがそれぞれ中間キャリー C，r ビットのワード z_j $(0 \le j \le m-1)$ となる．k ビットの入力 X，Y，N は，それぞれ r ビットごとのワード x_i，y_j，$n_j (0 \le i, j \le m-1)$ に分割され，Loop1 (x_i に対するループ) と Loop2 (y_j，n_j に対するループ) により，繰り返し演算される．演算の終了時に $Z = (z_{m-1}, ..., z_1, z_0)_{2^r}$ に格納されている値が出力となる．

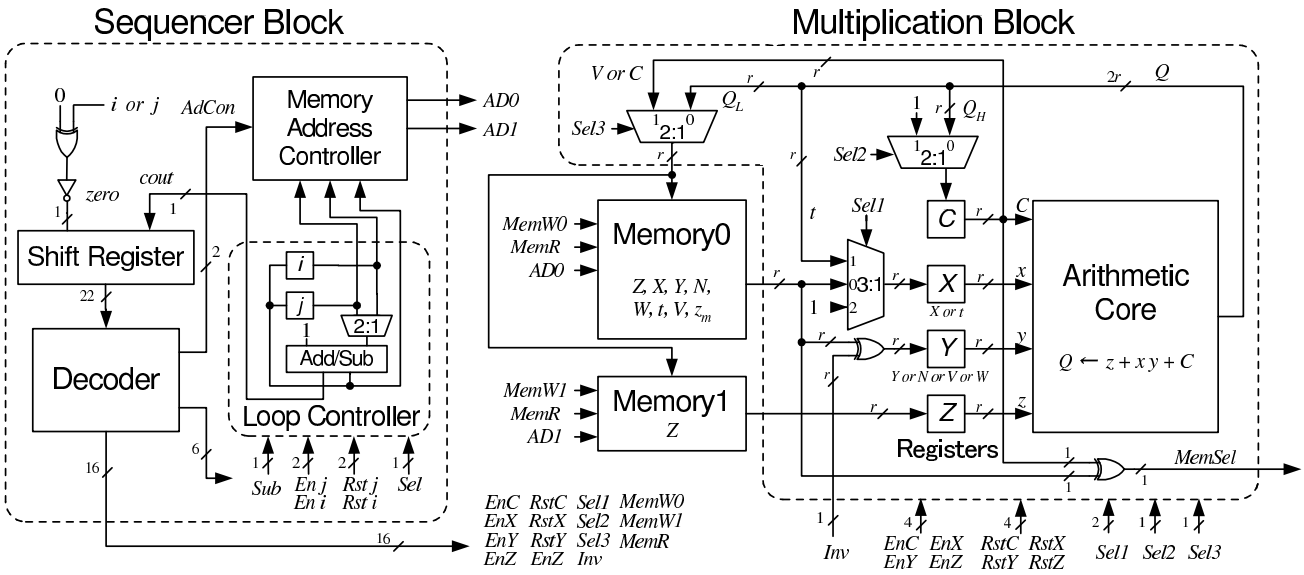


図 1: スケーラブル高基数モンゴメリ乗算器のアーキテクチャ

ALGORITHM 2では，1 ステップの演算に対し，2 つの積項を含む積和演算が必要となる（4，6 行目）．また，最後の減算において，k ビットの演算が必要となる．ここで各々の演算を異なる 2 つの積和演算器で実現することは，演算速度や回路面積の点であまり効率的ではない．そこで，**ALGORITHM 2** を書き下した **ALGORITHM 2'** を考える．**ALGORITHM 2'** において，4–5 行目，6–18 行目，20–30 行目は，**ALGORITHM 2** の 4 行目，5–8 行目，12 行目にそれぞれ対応する．**ALGORITHM 2'** では，$Q = z + xy + C$ のような $r \times r$ ビット 3 項積和演算を 1 ステップの基本演算とする．したがって，モンゴメリ乗算アルゴリズムの演算を共通の $r \times r$ ビット 3 項積和演算器で実現できる．

3 スケーラブル高基数モンゴメリ乗算器の設計

本章では，**ALGORITHM 2'** に基づくスケーラブル高基数モンゴメリ乗算器のアーキテクチャを示した後，そこで用いた積和演算器の構成と積和演算器の設計に用いる算術アルゴリズムについて述べる．

3.1 アーキテクチャ

設計したスケーラブル高基数モンゴメリ乗算器のアーキテクチャを図 1 に示す．本アーキテクチャは，演算を行う Multiplication Block，制御を行う Sequencer Block，入出力や中間変数の値を格納する 2 つのメモリ Memory0，Memory1 から構成される．**ALGORITHM 2'** では，k ビットの入力を r ビットのワードに分割するため，データのバス幅は r ビットとなる．また，メモリは，一度に読み出し/書き込みのどちらかだけを行う RAM を想定する（同時読み出し/書き込み可能な RAM を用いる場合でもモンゴメリ乗算器全体の構成に大きな変更は必要ない）．Multiplication Block は，$r \times r$ ビット積和演算器 Arithmetic Core とレジスタ X，Y，C，Z から構成される．

Arithmetic Core は，**ALGORITHM 2'** における以下の積和演算を行う．

$$Q := z_j + x_i y_j + C \tag{5}$$

$$Q := z_j + n_j t_i + C \tag{6}$$

出力 Q は $2r$ ビットであり，上位 r ビットはキャリーとして Arithmetic Core にフィードバックされ，下位 r ビットはワード z_j としてメモリ Memory0 と Memory1 に格納される．各サイクルにおいて Arithmetic Core は，レジスタに格納されているワードに対して演算を行う．そのため，入力となる x_i，y_j，n_j，z_j などのワードは，事前にメモリから読み出しておく必要がある．本アーキテクチャでは，これらのワードを同時に読み出せるように，ワード z_j は Memory0，Memory1 の両方にその値が格納される．

モンゴメリ乗算処理は，Sequencer Block が Arithmetic Core を繰り返し制御することで行われ，その全体のサイクル数は分割数を m として以下の式で与えられる．

$$6m^2 + 11m + 3 \tag{7}$$

ここで，このサイクル数はメモリの読み出し/書き込みおよび最後の減算処理を含むが，W の計算といった前処理は含まない．例えば，$k = 1,024$，$r = 32$ とした場合，本アーキテクチャは 6,499 サイクルでモンゴメリ乗算処理を行う．

3.2 積和演算器の構成

本アーキテクチャでは，Arithmetic Core（$r \times r$ ビット積和演算器）がクリティカルパス上にあり，その性能がモンゴメリ乗算器全体の処理速度に大きく影響する．また，積和演算器のような算術演算回路の性能向上には，適切な算術アルゴリズムを選択することが重要となる．そこで以下では，積和演算器の構成とその算術アルゴリズムについて述べる．

設計した積和演算器は，図 2(a) に示すように，部分積生成器 (PPG: Partial Product Generator)，部分積加算器 (PPA: Partial Product Accumulator)，最終段加算器 (FSA: Final Stage Adder) から構成される．これらの構成要素は，それぞれ以下のような役割を担う．

- PPG: 2 つの入力 x，y を桁ごとに乗算し，複数の部分積を求める．
- PPA: PPG から得られた複数の部分積，加算する入力 C，および z を加算し，Carry-Save 形式の 2 出力を求める．
- FSA: 2 入力 1 出力の加算を行う．

最終的に FSA の出力が積和演算器の出力 Q となる．図 2(b) に，16 ビット積和演算器の構成例を示す．ここでは，PPG に Radix-4 Modified Booth，PPA に Wallace Tree，FSA に Ripple Carry Adder という算術アルゴリズムを用いている．

図 3 は設計に用いた算術アルゴリズムの一覧であり，以下ではそれらを PPG，PPA，FSA に分けて簡単に説明する．なお，アルゴリズムの詳細については [7]–[9] を参照されたい．

PPG では 2 種類の算術アルゴリズムを用いた．それらは，最も一般的な 2 入力の論理積（AND）をとる Non-Booth と基数 4 の Booth エンコーダを用いて部分積の数を約半分に削減する Radix-4 Modified Booth である．

PPA では 7 種類の算術アルゴリズムを用いた．それらは，構成要素となる桁上げ保存加算器の種類と最適化の抽象度で分類される．ワードレベルの (3,2) カウンタ (CSA: Carry Save Adder) による Array，Wallace Tree，Balanced-Delay Tree，および Overturned-Stairs Tree，上記とは異なる桁上げ保存加算器，最適化の抽象度による Dadda Tree，(4;2) Compressor Tree，(7,3) Counter Tree である．

FSA では 10 種類の算術アルゴリズムを用いた．全加算器を順次接続した Ripple Carry Adder，桁上げを先見することにより高速化を計った Carry Lookahead Adder，Rippleblock CLA，Block CLA，桁上げ先見の考えを一般化した Kogge-Stone Adder，Brent-Kung Adder，Han-Carlson Adder，桁上げにより加算結果を選択する Conditional Sum Adder，Carry Select Adder，および桁上げの伝搬を飛び越すパスを用意した Carry Skip Adder である．

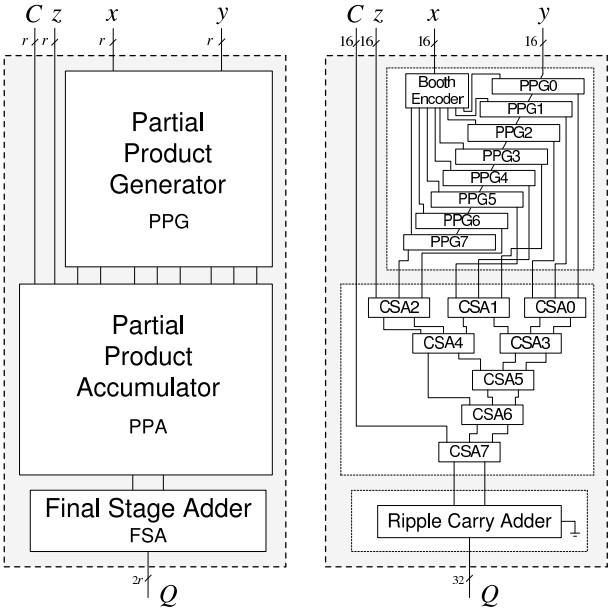


図 2: 積和演算器の構成: (a) 構成，(b) 構成例 (PPG: Radix-4 Modifed Booth，PPA: Wallace Tree，FSA: Ripple Carry Adder)

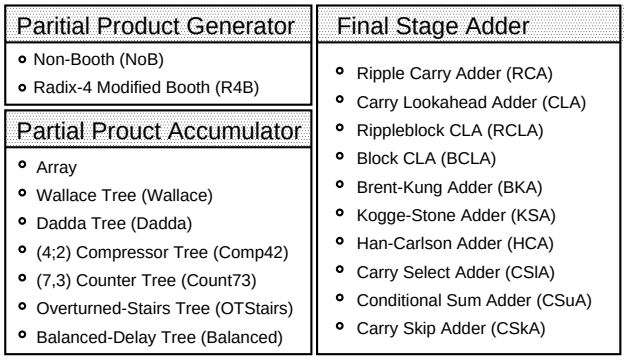


図 3: 設計に用いた算術アルゴリズム

PPG，PPA，FSA それぞれの算術アルゴリズムの組み合わせにより，r ビットの入力に対して 154 種類の積和演算器を設計することが可能である．これらの積和演算器を網羅的に設計し，Arithmetic Core としてモンゴメリ乗算器を設計した．

4 性能評価

本章では，設計した高基数モンゴメリ乗算器の ASIC ライブラリによる性能評価について述べる．回路の評価条件を表 1 に示す．ここでは法 N を $k = 1,024$ ビットとしているが，メモリサイズやカウンタの簡単な変更により任意のビット数に対応可能である．論理合成には，Synopsys 社の Design Compiler を用い，Hitachi 0.18μm CMOS スタンダードセルライブラリによって性能評価を行った．またこのとき，表 1 に示すようなオプションを指定した．以下では，(1) ワード長 r を変更した場合

表 1: 設計したモンゴメリ乗算器の評価条件:
(1) ワード長 r を変更した場合の評価条件
(2) 積和演算器の算術アルゴリズムを変更した場合の評価条件
(3) 特定のモンゴメリ乗算器に対して最適化した場合の評価条件

Cell Library: Hitachi 0.18μm CMOS (by Kyoto University)
Synthesis Tool: Synopsys Design Compiler

Input Length k: 1,024 bits
Word Length r: (1) 8, 16, 32, 64, 128 bits; (2), (3) 32 bits

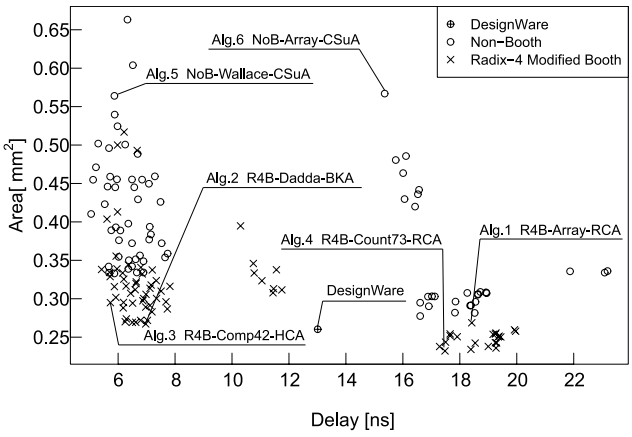
Design of Arithmetic Core:
(1) Synopsys DesignWare
(2) 154 Algorithms + DesignWare
(3) 6 Algorithms + DesignWare

Options for Synopsys Design Compiler:
(1), (2), (3) set_max_area 0
(1), (2) derive_timing_constraints -period_scale 0.5
(3) create_clock -period 3 -waveform {0 1.5} CLK
(1), (2), (3) compile -incremental -map_effort high

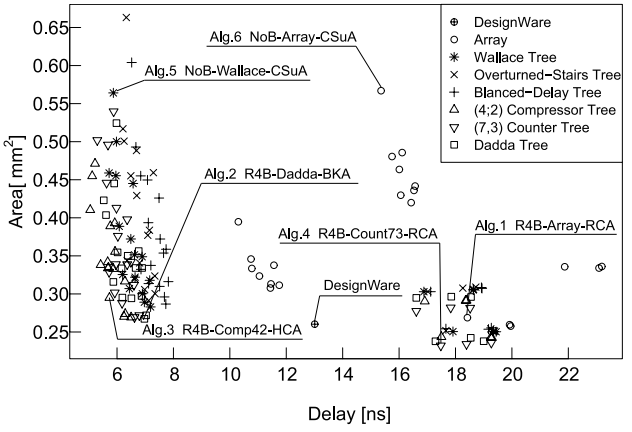
の評価結果，(2) 積和演算器の算術アルゴリズムを変更した場合の評価結果，さらに (3) 特定の積和演算器を用いたモンゴメリ乗算器に対して，最適化した場合の評価結果をそれぞれ示す．

まず，ワード長 r を 8〜128 ビットの範囲で変更した場合の結果を表 2 に示す．このとき，積和演算器には Synopsys 社の IP ライブラリ（DesignWare）から提供されたものを用いた．ここで，1 サイクルあたりの消費電力 Power とモンゴメリ乗算の処理時間 Time は，最大動作周波数 (MOF: Maximum Operating Frequency) で見積もった．なお回路面積 Area はメモリを含まない．表 2 より，1 サイクルあたりの最大遅延時間は，ワード長 r に比例して増大することがわかる．しかし，サイクル数の大幅な減少により，ワード長 r が大きくなるにつれて全体の処理時間は小さくなる．また，回路面積は，積和演算器が支配的であり，r を大きくすると自乗に近いペースで増大する．r に対して処理時間と回路面積はトレードオフの関係にあるので，要求される性能に応じて適切な r を設定することが重要である．また本アーキテクチャは高いスケーラビリティを有するため，全体の構成を変えることなく様々なワード長をサポートできる．

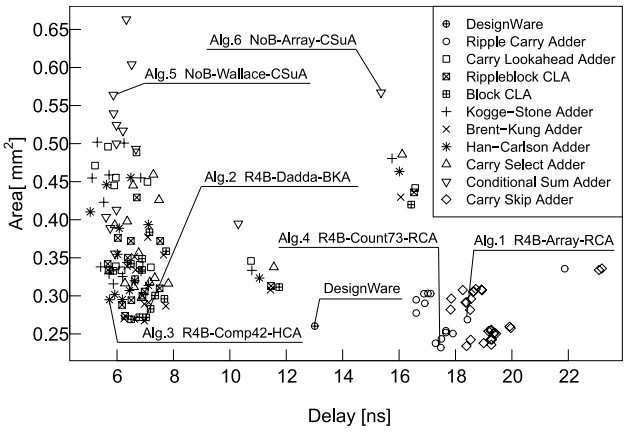
次に，ワード長 r を 32 ビットとし，積和演算器の算術アルゴリズムを変更した結果を図 4 に示す．(a)〜(c) はいずれも全 154 種類の実装をプロットしているが，算術アルゴリズムによる性能の違いがわかるように，プロットのマークを PPG，PPA，FSA の種類ごとに変えて表示した．横軸は 1 サイクルあたりの最大遅延時間，縦軸は回路面積を表す．また，比較のため DesignWare から提供された積和演算器を用いた場合も評価した．図 4 から，算術アルゴリズムの違いにより，多様な性能のモンゴメリ乗算器が設計されることが確認できる．DesignWare を用いた場合よりも高い性能を有するモンゴメリ乗算器が多数得られた．



(a)



(b)



(c)

図 4: モンゴメリ乗算器の性能分布 (a) PPG による分類，(b) PPA による分類，(c) FSA による分類

最後に，図 4 に示す 6 つの特徴的な積和演算器に基づくモンゴメリ乗算器を最適に合成した結果を表 3 に示す．ここで，消費電力は動作周波数を 50MHz に正規化して見積もった．評価に使用したライブラリでは，積和演算器の算術アルゴリズムを Radix-4 Modified Booth – Dadda Tree – Brent-Kung Adder とした場合に最も高速なモンゴメリ乗算器が得られた．また，面積遅延積

表 2: 評価結果：$k = 1,024$，ワード長 r を変更

$k = 1,024$ Montgomery Multiplier						
	Number of Cycles	Critical Path [ns]	Area [mm^2]	Power [μW] (at MOF)	Time [μs]	Area × Time
$r = 8$ bit	99,715	3.61	0.0436	237.87	359.97	15.69
$r = 16$ bit	25,283	6.95	0.0872	229.01	175.72	15.32
$r = 32$ bit	6,499	13.29	0.2440	306.25	86.37	21.07
$r = 64$ bit	1,715	27.42	0.7973	307.68	47.03	37.49
$r = 128$ bit	475	53.17	3.0452	308.03	25.25	76.89

表 3: 評価結果：積和演算器の算術アルゴリズムを変更

$k = 1,024$，$r = 32$ Montgomery Multiplier					
Arithmetic Core Algorithm	Critical Path [ns]	Area [mm^2]	Power [μW] (at 50MHz)	Time [μs]	Area × Time
DesignWare	12.59	0.324	600.39	81.82	26.40
Algorithm 1 (R4B-Array-RCA)	9.41	0.531	424.70	61.15	32.47
Algorithm 2 (R4B-Dadda-BKA)	4.79	0.523	203.22	31.13	16.28
Algorithm 3 (R4B-Comp42-HCA)	5.15	0.410	144.31	33.46	13.71
Algorithm 4 (R4B-Count73-RCA)	7.83	0.371	129.18	50.88	18.87
Algorithm 5 (NoB-Wallace-CSuA)	5.07	0.763	214.28	32.94	25.13
Algorithm 6 (NoB-Array-CSuA)	13.62	0.701	384.20	88.51	62.04

では，算術アルゴリズムを Radix-4 Modified Boorh – (4;2) Compressor Tree – Han-Carlson Adder とした場合に最も性能の高いモンゴメリ乗算器が得られ，典型的な Radix-4 Modified Booth – Array – Ripple Carry Adder の構成に対して 2.4 倍もの性能向上が得られた．

本稿のセルベース設計においては，傾向として規則的なレイアウトが可能な算術アルゴリズム用いた場合に優れた性能が示されている．しかし，ここで得られた評価は，使用するテクノロジによって大きく影響する可能性のあることに注意が必要である．そのため，ASIC や FPGA といったプラットホーム，あるいはライブラリの種類に応じて，最適なアルゴリズムを検討することも重要な課題である．

5 まとめ

ワード長 8〜128 ビットの高いスケーラビリティを有する高基数モンゴメリ乗算器を，算術アルゴリズムが異なる 154 種類の積和演算器に対して網羅的に設計し，0.18 μm CMOS ASIC スタンダートセルライブラリにより性能を評価した．積和演算器のワード長が同じであっても，算術アルゴリズムによって速度と回路規模，そして消費電力が大きく異なることが確認された．今回設計したモンゴメリ乗算器は，これらのパラメータを自由に変更可能であるため，その評価結果を基にして，要求される性能を満たす最適なデザインを選択することができる．今後は，種々の CMOS テクノロジライブラリおよび FPGA に高基数モンゴメリ乗算器を実装し，より詳細な性能評価を行う予定である．

謝辞 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じてシノプシス株式会社および株式会社日立製作所の協力で行われたものである．

参考文献

[1] P. L. Montgomery: “Modular multiplication without trial division,” Math. Comp., **44**, 170, pp. 519–521 (1985).

[2] A. Daly and W. Marnane: “Efficient architectures for implementing montgomery modular multiplication and rsa modular exponentiation on reconfigurable logic,” FPGA ’02: Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays, NY, USA, ACM Press, pp. 40–49 (2002).

[3] A. F. Tenca and C. K. Koc: “A scalable architecture for modular multiplication based on montgomery’s algorithm,” IEEE Trans. Comput., **52**, 9, pp. 1215–1221 (2003).

[4] C. K. Koc, T. Acar and J. Burton S. Kaliski: “Analyzing and comparing montgomery multiplication algorithms,” IEEE Micro, **16**, 3, pp. 26–33 (1996).

[5] A. Satoh and K. Takano: “A scalable dual-field elliptic curve cryptographic processor,” IEEE Trans. Comput., **52**, 4, pp. 449–460 (2003).

[6] A. F. Tenca, G. Todorov and C. K. Koc: “High-radix design of a scalable modular multiplier,” Cryptographic Hardware and Embedded Systems -CHES2001, LNCS 2162, Springer-Verlag, pp. 185–201 (2001).

[7] I. Koren: “ Computer arithmetic algorithms 2nd Edition,” A K Peters (2001).

[8] B. Parhami: “ Computer Arithmetic: Algorithms and Hardware Designs,” Oxford University Press (2000).

[9] Arithmetic Module Generator based on ARITH. http://www.aoki.ecei.tohoku.ac.jp/arith/mg/