RC-003

RSA 暗号プロセッサジェネレータの設計と評価
Design and Evaluation of RSA Processor Generator

馬場 祐一† 宮本 篤志† 本間 尚文† 青木 孝文† 佐藤 証‡
Yuichi Baba Atsushi Miyamoto Naofumi Homma Takafumi Aoki Akashi Satoh

1. まえがき

近年，ネットワークの発展や情報のディジタル化，電子マネーの普及などに伴い，情報セキュリティ技術への要求が高まっている．中核を担う技術として，プライバシー保護や情報の秘匿性を目的とした暗号技術がある．その中でも，公開鍵暗号方式は，鍵管理が容易であり，電子署名や認証などの応用に広く利用されている．公開鍵暗号は，一般に乗剰余算を中心とした膨大な多倍長演算を必要とするため，計算コストが非常に高い．デファクトスタンダードである RSA 暗号では，多倍長の乗剰余算を千回以上も実行する必要があるため，IC カードなどの計算リソースが限られた機器では，しばしばハードウェア実装が求められる．

モンゴメリ乗算アルゴリズム [1] は，演算コストの高い乗剰余算をシフト演算を用いて効率的に実行するアルゴリズムとして知られており，これまで暗号ハードウェアの多くに適用されている [2]–[8]．それらはオペランドの分割の幅に応じて，基数 2 と高基数のアーキテクチャに分類される．基数 2 のアーキテクチャは，乗算の片方のオペランドをビット毎に展開して扱うため，加算器を用いた比較的単純な構成となる．しかし，語長の大きな加算器を用いた場合には，スケーラビリティや動作周波数が低下してしまう．一方で，加算器の語長を小さくした場合は，全体の処理サイクルが増大してしまう．文献 [3][6] では，1 つの演算器を 1 ステージとし，それを直列に並べてサイクル数の削減を図っているが，回路面積に制限がある場合，スループットを向上させることは難しい．一方，高基数のアーキテクチャは，乗算の乗数や被乗数の片方もしくは両方を適当な語長のワードで扱うため，乗算器を用いた構成が基本となる．文献 [2] では，64 ビット × 64 ビット乗算を用いた積和演算器ベースのアーキテクチャが提案されている．高基数のアーキテクチャでは，演算をワード毎に処理するため，基数 2 のアーキテクチャと比べて演算効率に優れた実装が可能となる．

高基数のアーキテクチャでは，積和演算器を含むデータパスの構造が回路全体の性能に対して支配的であるため，用途に応じて適切なデータパスアーキテクチャを選択する必要がある．また，演算器にも様々な構成法があり，その性能は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．しかし，従来の研究では特定のアーキテクチャや演算器のみに着目した設計がなされており，アーキテクチャから算術アルゴリズムまでを考慮した系統的な設計・評価はなされていない．今後，暗号技術の用途が多様化するにつれ，要求される回路面積や動作速度に応じた系統的な設計の重要性は益々高まると予想される．

本稿では，高基数モンゴメリ乗算に基づくスケーラブルな RSA 暗号プロセッサアーキテクチャを提案する．提案するアーキテクチャは，回路面積を重視した積和演算器に基づく構成と演算速度を Carry-Save 形式により向上させた構成の 2 種類である．これらのアーキテクチャは，基数 2^8～2^{128} に対応可能であり，その中心となる積和演算器の算術アルゴリズムを任意に設計可能である．また，本稿では，その RSA プロセッサを設計仕様に応じて自動生成するジェネレータへの応用を示す．本ジェネレータは，2 種類のアーキテクチャ，5 種類の基数，および 77 種類の積和演算アルゴリズムに対応しており，これらの組み合わせにより 770 種類の RSA 暗号プロセッサを生成することができる．その有効性を検証するため，生成可能な全ての RSA 暗号プロセッサを 90nm CMOS スタンダードセルライブラリにより合成した結果を評価した．特に，より実装に近い評価を行うため，配置配線後の演算時間および回路面積を求めた．その結果，1,024 ビットの RSA 暗号に対して，1.25[Kgate]@174.89[ms] の小型実装（基数 2^8）から，227.16[Kgate]@1.45[ms] の高速実装（基数 2^{128}）まで多様な回路性能が実現された．

2. RSA 暗号

2.1 べき乗剰余演算

RSA 暗号は，べき乗剰余演算により暗号化・復号処理を行う公開鍵暗号である．P を元のデータ（平文），C を暗号文，E と N を公開鍵，D を秘密鍵とすると，暗号化および復号処理は次のような式で表される．

$$C = P^E \bmod N \quad (1)$$

$$P = C^D \bmod N \quad (2)$$

公開鍵である法 N や秘密鍵 D には，安全性の観点から 1,024 ビット以上の多倍長整数が利用される．また，平文 P や暗号文 C も同一の語長が用いられる．

RSA 暗号のべき乗剰余演算は，指数 E あるいは D のビットパターンに応じて，自乗剰余算と乗剰余算（以降，簡単のため自乗算と乗算と呼ぶ）を繰り返すことによって実現される．最も基本的なべき乗剰余演算アルゴリズムである左バイナリ法を **ALGORITHM 1** に示す．左バイナリ法では，指数ビットの最上位からスキャンし，ビットが 0 ならば自乗算（3 行目）を，1 ならば自乗算と乗算（3–6 行目）をペアで行う．

乗剰余演算で最も計算コストが高い演算は法による除算である．そこで，除算を行わずにシフト演算を用いて効率的に剰余を求めることができるモンゴメリ乗算アルゴリズム [1] が広く用いられている．

†東北大学 大学院情報科学研究科, GSIS, Tohoku University
‡産業技術総合研究所, AIST

ALGORITHM 1
MODULAR EXPONENTIATION (MSB FIRST)

Input: X, N, $E = (e_{k-1}, ...e_1, e_0)_2$
Output: $Z = X^E \bmod N$

```
1 : Z := 1;
2 : for i = k − 1 downto 0
3 :   Z := Z × Z mod N;        − squaring
4 :   if (e_i = 1) then
5 :     Z := Z × X mod N;      − multiplication
6 :   end if
7 : end for
```

ALGORITHM 2
MONTGOMERY MULTIPLICATION

Input: X, Y, N, $W = -N^{-1} \bmod R$
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;
```

2.2 モンゴメリ乗算

モンゴメリ乗算では，2 つの整数 X，Y に対して，次の演算を行う．

$$Z = XYR^{-1} \bmod N \tag{3}$$

ここで，X，Y，R，N は以下の関係を持つ．

$$0 \le X, Y < N < 2^k = R \tag{4}$$

ALGORITHM 2 に，モンゴメリ乗算アルゴリズムを示す．本アルゴリズムでは，XY の乗算結果に法 N の倍数を加算し，2^k で割り切れる値に補正することで，剰余算を不要にしている．2^k の除算はシフト演算により実現される．

RSA 暗号では，1,024 ビット以上の多倍長データを扱うため，しばしばデータをワード毎に分割したアルゴリズムが用いられる．本稿では，その中でもデータをビット長 r（基数 2^r，$r < k$）のワード毎に m 分割 ($m = k/r$) した高基数のアルゴリズム [2] を適用する．

ALGORTIHM 3 に高基数モンゴメリ乗算アルゴリズムを示す．本アルゴリズムにおいて，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．

$$X = x_{m-1}2^{r(m-1)} + ... + x_1 2^r + x_0 \tag{5}$$

また，ここでは，式 (5) を以下のように簡略化する．

$$X = (x_{m-1}, ..., x_1, x_0)_{2^r} \tag{6}$$

以上のように，k ビットの入力 X，Y，N は，それぞれ r ビット毎のワード x_i，y_j，n_j $(0 \le i, j \le m-1)$ に分割され，Loop 1 (x_i に対するループ) と Loop 2 (y_j，n_j に対するループ) により，繰り返し演算される．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，中間和 z $(0 \le j \le m-1)$ となる．このとき，7 行目において，

ALGORITHM 3
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                          − Loop 1
3 :   c := 0;
4 :   t_i := (z_0 + x_i y_0) w mod 2^r;
5 :   for j = 0 to m − 1                        − Loop 2
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;
```

ALGORITHM 4
MODULAR EXPONENTIATION WITH *MontMult*

Input: X, N, $E = (e_{k-1}, ..., e_1, e_0)_2$
Output: $Z = X^E \bmod N$

```
1 : W := −N^{−1} mod R;
2 : Y := XR mod N;
3 : Z := R mod N;
4 : for i = k − 1 downto 0
5 :   Z := MontMult (Z, Z, N, W);      − squaring
6 :   if (e_i = 1) then
7 :     Z := MontMult (Z, Y, N, W);    − mult.
8 :   end if
9 : end for
10: Z := MontMult (Z, 1, N, W);
```

中間和を 1 つ前のワード（すなわち z_{j-1}）に格納することにより，**ALGORITHM 2** におけるシフト演算も同時に実行している．ここで，mod 2^r という演算自体にも，除算が必要ないことに注意されたい．最終的に，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．

ALGORITHM 4 に，モンゴメリ乗算を組み合わせた左バイナリ法を示す．関数 *MontMult* は，式 (3) で表されるモンゴメリ乗算であり，1，2 行目はモンゴメリ乗算を適用する際に必要となる前処理の演算である．

3. RSA 暗号プロセッサ

3.1 モンゴメリ乗算回路アーキテクチャ

高基数モンゴメリ乗算アルゴリズムのハードウェア実装では，**ALGORITHM 3** の 6 行目の積和演算を実行する演算器がクリティカルパスとなり，その演算器を含むデータパスが回路全体の性能へ大きな影響を与える．本稿では，特に中間データの伝搬方式が演算性能に与える影響を考慮し，中間データの形式の異なる 2 種類のデータパスアーキテクチャを提案する．

図 1 に，提案するアーキテクチャ(**Type-I** および **Type-II**) を示す．k ビットのオペランドをワード毎に分割し処理するため，データのバス幅は r ビットとなる．各データパスは，積和演算器である Arithmetic Core およびその入力を保持するレジスタ，マルチプレクサなど

ALGORITHM 5 (Type-I)

```
1 :  Z := 0;   v := 0;
2 :  for i = 0 to m − 1
3 :     (c_a, z_0) := z_0 + x_i y_0;
4 :     t_i := z_0 w mod 2^r;
5 :     (c_b, z_0) := z_0 + t_i n_j;
6 :     for j = 1 to m − 1
7 :        (c_a, z_j) := z_j + x_i y_j + c_a;
8 :        (c_b, z_{j−1}) := z_j + t_i n_j + c_b;
9 :     end for
10:     (v, z_{m−1}) := c_a + c_b + v;
11:  end for
12:  if (Z > N) then Z := Z − N;
```

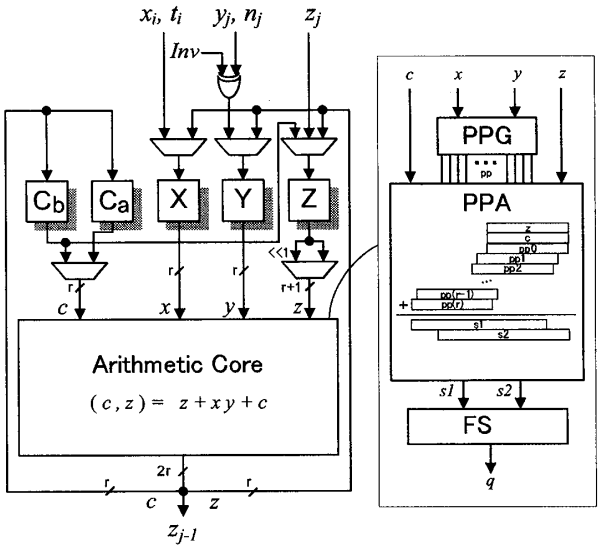
ALGORITHM 6 (Type-II)

```
1 :  Z := 0;   v := 0;
2 :  for i = 0 to m − 1
3 :     (cs1_a + cs2_a + ec_a, z_0) := z_0 + x_i y_0;
4 :     t_i := z_0 w mod 2^r;
5 :     (cs1_b + cs2_b + ec_b, z_0) := z_0 + t_i n_j;
6 :     for j = 1 to m − 1
7 :        (cs1_a + cs2_a + ec_a, z_j)
                := z_j + x_i y_j + cs1_a + cs2_a + ec_a;
8 :        (cs1_b + cs2_b + ec_b, z_{j−1})
                := z_j + t_i n_j + cs1_b + cs2_b + ec_b;
9 :     end for
10:     (v, z_{m−1})
          := cs1_a + cs2_a + ec_a + cs1_b + cs2_b + ec_b + v;
11:  end for
12:  if (Z > N) then Z := Z − N;
```

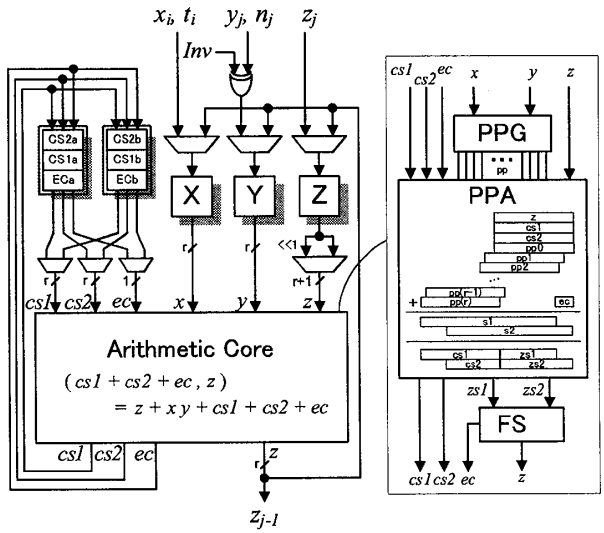
から構成される．また，Arithmetic Core は，図 1 のように部分積生成器 (PPG: Partial Product Generator)，部分積加算器 (PPA: Partial Product Accumulator)，最終段加算器 (FSA: Final Stage Adder) から構成される．まず，PPG では，x_i にビットごとに分割した $y_{j(l)}(0 \le l < r)$ を乗じて，複数の部分積 $x_i y_{j(l)}$ を生成する．次に PPA では，PPG から得られた複数の部分積，加算項のオペランドを多入力加算器を用いて加算する．結果は Carry-Save 方式で出力される．最後に，FSA では，PPA で得られた Carry-Save 方式の入力に対して桁上げ伝搬加算を行い，最終的な出力を得る．**Type-I** と **Type-II** では，FSA のサイズが異なることに注意されたい．**Type-II** では，FSA の桁上げ伝搬加算の遅延時間が低減されている．

ALGORITHM 5〜6 は，図 1 の **Type-I** および **Type-II** で用いる高基数モンゴメリ乗算アルゴリズムである．分割したワードの演算方式は FIOS (Finely Integrated Operand Scanning method) [9] に従う．ここで，(c, z) は $c \cdot 2^r + z$ を表す．また，v は i ループ間のキャリーである．

Type-I のアーキテクチャは，Arithmetic Core に基本的な 3 項積和演算器を用いる．この構成のため，**ALGORITHM 5** では，**ALGORITHM 3** の 6 行目の積和演算を 2 つに分割している（7，8 行目）．また，レジスタ数の増加を防ぐため，中間データの伝搬に Carry-Save 形式は用いない．そのため，最大 $2r$ ビットの入力を持つ FSA が必要となる．Arithmetic Core の出力 $2r$ ビットは，上位と下位で r ビットに分割され，キャリー側の出力 c と中間和側の出力 z に接続される．



(a) **Type-I**



(b) **Type-II**

図 1: 高基数モンゴメリ乗算回路アーキテクチャ

キャリー側の出力 c は Arithmetic Core へフィードバックされ，中間和側の出力 z は，**ALGORITHM 5** の 7 行目の演算時には Z レジスタへ，8 行目の演算時にはメモリへと格納される．

一方で，**Type-II** のアーキテクチャは，より演算速度を重視した実装となる，基本的なアイデアは，積和演算のキャリー側の出力に Carry-Save 形式を用いて FSA の遅延時間を削減することである．キャリー側の出力は，r ビットの $cs1$, $cs2$ および 1 ビットの ec の 3 つの信号を用いて $cs1 + cs2 + ec$ と表せる．ここで，$cs1$，$cs2$ は桁上げ保存加算からのキャリー，ec は桁上げ伝搬加算からのキャリーを表す．図 1(b) において，Arithmetic Core から得られるキャリー側の出力は，Carry-Save 形式でサイクル間を伝搬させる．キャリー用のレジスタ数は **Type-I** に比べて約 2 倍となるが，FSA のサイズを抑えることができるため，面積が大幅に増加することはない．また，Arithmetic Core の遅延時間も，FSA

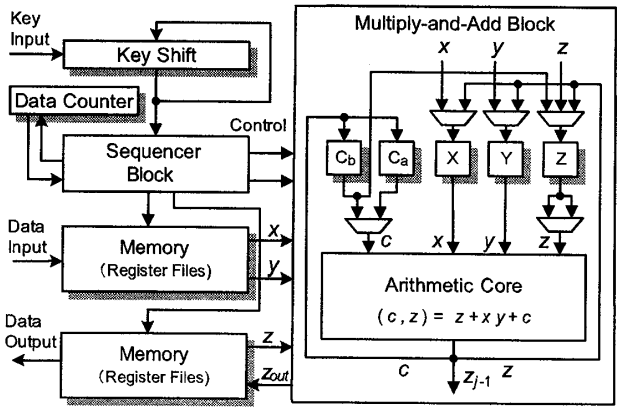


図 2: RSA 暗号プロセッサ

サイズの削減により，**Type-I** に比べて低減される．

3.2 プロセッサアーキテクチャ

図 2 に提案する RSA 暗号プロセッサのブロック図を示す．本プロセッサは，演算部 Multiply-and-Add Block，制御部 Sequencer Block，メモリ Memory，鍵レジスタ Key Shift およびカウンタ Data Counter から構成される．Multiply-and-Add Block はモンゴメリ乗算器のデータパスであり，本稿では上述の 2 種類を想定している．また，Sequencer Block は，**ALGORITHM 4** 用の制御回路である．各ステップにおいて，モンゴメリ乗算器を複数回呼び出すことによってべき乗剰余演算が実行される．

本プロセッサのクロックサイクル数 t_c は，以下の式で表される．

$$t_c = t_{pre} + t_{mont} \cdot (\frac{3}{2}k + 1) \quad (7)$$

ここで，t_{pre} は前処理（**ALGORITHM 4** の 1，2 行目），t_{mont} はモンゴメリ乗算の処理サイクル数を表す．RSA 暗号の処理サイクル数 t_c は，鍵のビット値によって変化するが，ここでは'0' と'1' が半分ずつ発生するとして見積もった．そのため，モンゴメリ乗算の処理サイクル数 t_{mont} の係数は $(3k/2)+1$ となる．t_{pre} および t_{mont} のサイクル数は，**Type-I** では以下の式で表される．

$$t_{pre1} = (2r+1) + (km + k + m + 1) \quad (8)$$
$$t_{mont1} = 2m^2 + 4m + 1 \quad (9)$$

一方，**Type-II** では以下の式で表される．

$$t_{pre2} = (2r+1) + (km + k + m + 1) \quad (10)$$
$$t_{mont2} = 2m^2 + 5m + 1 \quad (11)$$

ここで，t_{pre} の括弧で括られたそれぞれの部分は，**ALGORITHM 4** の 1，2 行目に対応する．**Type-I** および **Type-II** で t_{pre} のサイクル数が等しいのは，前処理の演算が加減算処理のみとなるためである．両アーキテクチャで同一の加減算器（図 1 中の FSA）が搭載されている．一方，モンゴメリ乗算の処理サイクル数は，**Type-I** と **Type-II** で異なる．アルゴリズム（**ALGORITHM 5** および **6**）の各行が 1 サイクルに対応しており，各ループの繰り返し回数は分割数 m に依存している．そのため，**Type-I**（**ALGORITHM 5**）のサイクル数は，7，8 行目で $2m^2$，3–5，10 行目で $4m$ および 1 行目で 1 となる．一方，**Type-II**（**ALGORITHM 6**）では，10 行目の処理が 2 サイクルに分けて行われるため，2–11 行目のサイクル数が $5m$ となる．

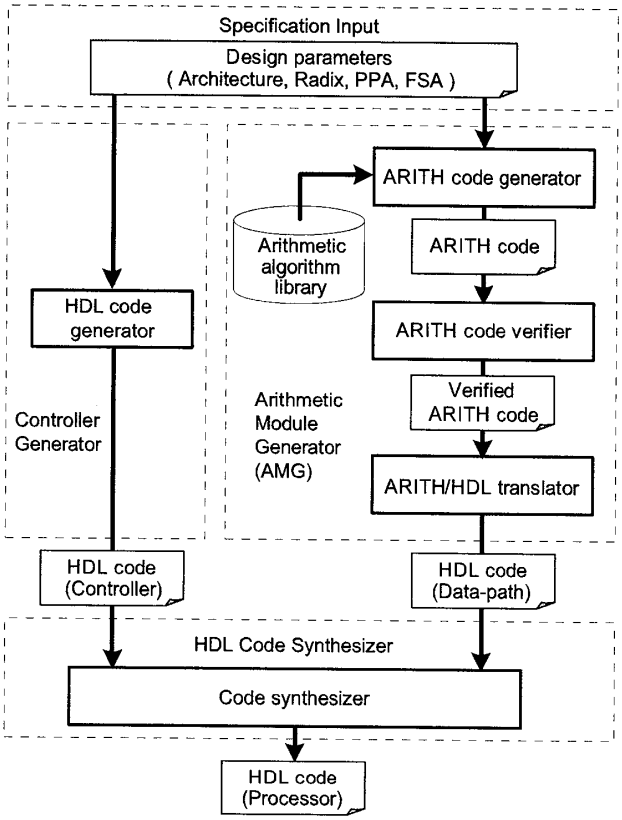


図 3: RSA 暗号プロセッサジェネレータ

例として，鍵長 $k = 1,024$，分割数 $m = 32$ の場合を考えると，**Type-I** では，モンゴメリ乗算のサイクル数 t_{mont} が 2,177，RSA 暗号のサイクル数 t_c が約 3,380K となる．一方，**Type-II** では，t_{mont} が 2,209，t_c が約 3,429K となる．**Type-II** のサイクル数は，**ALGORITHM 6** の 10 行目により，**Type-I** に比べて 2% 程増加する．

4. RSA 暗号プロセッサジェネレータ

4.1 ジェネレータの構成

RSA 暗号プロセッサジェネレータは，仕様入力部，演算器生成部，コントローラ生成部およびコード合成部からなる．図 3 に概略を示す．

まず，仕様入力部は，ユーザから指定されたプロセッサの設計アーキテクチャや基数，演算器の算術アルゴリズム等の各仕様を解釈し，演算器生成部およびコントローラ生成部に必要な情報を入力する．次に，演算器生成部およびコントローラ生成部では，それらの情報をもとにデータパスおよびコントローラを生成する．演算器生成部は，演算器モジュールジェネレータ (AMG: Arithmetic Module Generator)[10, 11] をもとに実装されている．AMG は，演算器の算術アルゴリズムを系統

表 1: ジェネレータの設計パラメータ

Architecture	**Type-I,Type-II**
Radix	2^8,2^{16},2^{32},2^{64},2^{128}
Arithmetic Components	**Partial Product Accumulator**
	Array Wallace Tree Balanced-Delay Tree Overturned-Stairs Tree Dadda Tree (4,2) Compressor Tree (7,3) Counter Tree
	Final Stage Adder
	Ripple Carry Adder Carry Lookahead Adder Ripple-Block Carry Lookahead Adder Kogge-Stone Adder Brent-Kung Adder Han-Carlson Adder Ladner Fischer Adder Conditional Sum Adder Carry Select Adder Fixed-Block Carry Skip Adder Variable-Block Carry Skip Adder

的にライブラリ化しており，算術アルゴリズム記述言語 ARITH [12] およびその処理系を用いて，完全に機能保証されたデータパスのコードを生成する．ARITH の詳細は文献 [12] を参照されたい．一方，コントローラ生成部は，基数やアーキテクチャの情報をもとに，演算順序を制御するシーケンサや中間データを保持するメモリ等のコードを生成する．最後に，コード合成部は，それぞれ個別に生成されたデータパスおよびシーケンサのコードからプロセッサのコードを合成する．

4.2 生成可能な RSA 暗号プロセッサ

本ジェネレータの設計パラメータを表 1 に示す．設計仕様は，アーキテクチャ，基数，および積和演算アルゴリズム（PPA および FSA のアルゴリズム）から決定される．

アーキテクチャは，3.1 節にて示した 2 種類（**Type-I** および **Type-II**）のモンゴメリ乗算回路アーキテクチャから選択される．

基数は，演算のワード長（アーキテクチャのバス幅）に対応している．基数の選択においても，回路規模とサイクル数（実行時間）の間にトレードオフが存在する．一般に，基数が小さい場合には回路規模は小さいがサイクル数は大きくなり，基数が大きい場合にはその逆となる．本ジェネレータでは，以上のトレードオフを考慮して，2^8，2^{16}，2^{32}，2^{64}，2^{128} の 5 種類から選択される．

積和演算器の算術アルゴリズムは，7 種類の PPA アルゴリズムおよび 11 種類の FSA アルゴリズムから選択される．PPA でのアルゴリズムは，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によって分類される．具体的には，ワードレベルの (3,2) Counter (CSA: Carry-Save Adder) から構成される Array，Wallace Tree，Balanced-Delay Tree および Overturned-Stairs Tree，ワードレベルの (4,2) Compressor から構成される (4,2) Compressor Tree，(7,3) Counter から構成される (7,3) Counter Tree，ビットレベルの CSA から構成される Dadda Tree がある．一方，FSA のアルゴリズムには，全加算器を順次接続する Ripple Carry Adder，桁上げ先見により高速な Carry Lookahead Adder，Ripple-Block Carry Lookahead Adder，桁上げ先見の考えを一般化した Kogge-Stone Adder，Brent-Kung Adder，Han-Carlson Adder，Ladner Fischer 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，桁上げ伝搬を飛び越すパスを持つ Fixed-Block Carry Skip Adder，Variable-Block Carry Skip Adder がある．これらの算術アルゴリズムの詳細は，[11, 13] を参照されたい．

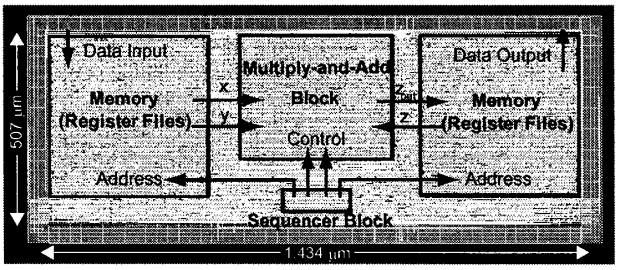


図 4: 配置配線後のレイアウト

本ジェネレータは，以上のアーキテクチャ，基数および演算アルゴリズムの組み合わせることで計 770 種類の RSA 暗号プロセッサを生成することができる．

5. 性能評価

RSA 暗号プロセッサジェネレータで生成可能な 770 種類のプロセッサについて性能評価を行った．評価に用いたライブラリは，ST Microelectronics 社 90nm CMOS スタンダードセルライブラリ [14] である．このとき，定格電圧 1.2V，最悪条件下（電源電圧 1.08V，125 ℃）とした．また，評価には，Synopsys 社の Design Compier および Astro を用いた．ジェネレータにより生成した RSA 暗号プロセッサの Verilog HDL のコードを Design Compiler と Astro を用いて論理合成・配置配線し，配線を含めた遅延時間，回路面積，面積遅延積（遅延時間と回路面積の積），消費電力により評価した．また，図 4 は配置配線を行った RSA 暗号プロセッサのセルである．

図 5 に基数を 2^{32} とした際の 154 種類の RSA 暗号プロセッサの性能プロットを示す．横軸は 1,024 ビット RSA 暗号の計算時間，縦軸は回路面積である．図より積和演算器のアルゴリズムに応じて様々な性能が得られていることがわかる．また，動作速度 (Speed)・回路面積 (Area)・面積遅延積 (Balance) に最も優れた実装を図中に示した．アーキテクチャの違いを見ると，**Type-I** は計算時間および回路面積について広く分布しているのに対して，**Type-II** は，その性能プロットが中心に集まっている．これは，**Type-II** において，FSA のサイズを半分にしたことにより，その遅延時間および回路面積が低減されたことに理由があると考えられる．しかしながら，**Type-II** ではレジスタ数は増加するため，回路面積が最小となる実装は **Type-I** から得られた．

図 5 に示す動作速度 (Speed)・回路面積 (Area)・面積遅延積 (Balance) に優れた 3 種類の実装をそれぞれのアーキテクチャで選択し，基数を 2^8〜2^{128} と変更し

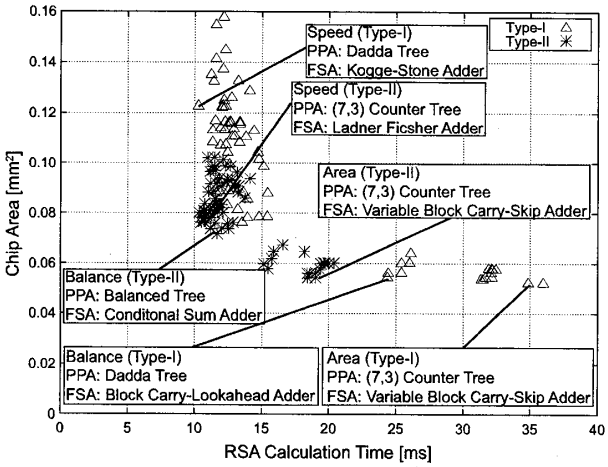


図 5: RSA 暗号プロセッサの性能分布 (基数 2^{32})

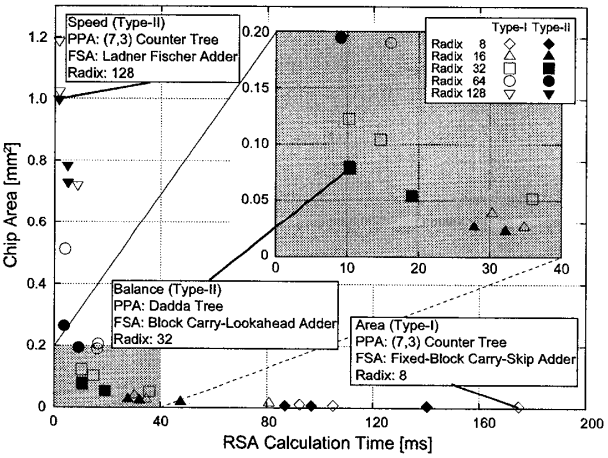


図 6: 各基数における性能分布

て最高性能を評価した．図 6 に回路面積 Chip Area と RSA 暗号の処理時間 RSA Calculation Time に対する全 30 プロットの結果を示す．図より，基数が大きいほど高速動作，基数が小さいほど小面積となることが確認できる．また，表 2 に結果をまとめたものを示す．ここで，MM Time はモンゴメリ乗算の処理時間，AD は面積遅延積を表す．Delay (DC) と Delay (Astro) はそれぞれ Design Compiler による論理合成時のゲート遅延のみを考慮したクリティカルパス，Astro による配置配線後のクリティカルパスである．消費電力 Power は，Design Compiler においてセル自体の消費電力と配線での消費電力を合計したものとして見積もった．

さらに，動作速度・回路面積・面積遅延積に優れた実装を図 6 および表 2 に太字で示した．さらに，比較のため，既存研究 [2][3][5] における評価結果も併せて表に示す．その結果，最も面積が小さい実装は，基数が 2^8，アーキテクチャが **Type-I**，演算アルゴリズム ((7,3) Counter Tree, Fixed-Block Carry Skip Adder) の場合であり，1.25[Kgate]@174.89[ms] であった．一方，最も高速な実装は，基数が 2^{128}，アーキテクチャが **Type-II**，演算アルゴリズム ((7,3) Counter Tree, Han-Carlson Adder) の場合であり，227.16[Kgate]@1.45[ms] であった．さらに，RSA 暗号の高速演算手法である CRT (Chinese Remainder Theorem) [15] をサポートすることで，1ms 以下の実行も可能となる．これらの値は，従来手法とは回路テクノロジが異なるため単純な比較は難しいが，従来では最も速い実装（130nm CMOS テクノロジ）[2] の 4.57[μs] に匹敵する性能となっている．ただし，提案手法の演算時間は Delay (Astro) から概算されていることに注意されたい．

6. むすび

本稿では，高基数モンゴメリ乗算に基づくスケーラブルな RSA 暗号プロセッサアーキテクチャ，およびその RSA プロセッサを設計仕様に応じて自動生成するジェネレータを提案した．本ジェネレータは，2 種類のアーキテクチャ(**Type-I** および **Type-II**)，5 種類の基数 (2^8～2^{128}) および 77 種類の演算器 (PPA 7 種類，FSA 11 種類) に対応しており，その組み合わせにより計 770 種類の RSA 暗号プロセッサを生成することができる．生成可能なすべてのプロセッサを 90nm CMOS スタンダードセルライブラリを用いて評価した結果，1,024 ビットの RSA 暗号に対して，1.25[Kgate]@174.89[ms] の小型実装（基数 2^8）から，227.16[Kgate]@1.45[ms] の高速実装（基数 2^{128}）までの多様な回路性能が得られた．今後は，ジェネレータの Web 上での公開するとともに，異なる CMOS テクノロジライブラリおよび FPGA による詳細な性能比較を行う予定である．

参考文献

[1] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, Vol. 44, No. 170, pp. 519–521, April 1985.

[2] A. Satoh and K. Takano. A scalable dual-field elliptic curve cryptographic processor. *IEEE Trans. Comput.*, Vol. 52, No. 4, pp. 449–460, April 2003.

[3] A. F. Tenca and C. K. Koc. A scalable architecture for modular multiplication based on Montgomery's algorithm. *IEEE Trans. Comput.*, Vol. 52, No. 9, pp. 1215–1221, September 2003.

[4] A. F. Tenca, G. Todorov, and C. K. Koc. High-radix design of a scalable modular multiplier. In *CHES '01: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 185–201. Springer-Verlag, 2001.

[5] E. Savas, A. F. Tenca, and C. K. Koc. A scalable and unified multiplier architecture for finite fields GF(p) and GF(2^m). In *CHES '00: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 277–292. Springer-Verlag, August 2000.

[6] D. Harris, R Krishnamurthy, S. Mathew, and S. Hsu. An improved unified scalable radix-2 Montgomery multiplier. In *ARITH '05: Proceedings of the 17th IEEE Symposium on Computer*

表 2: RSA 暗号プロセッサの性能

Ref.	Radix	Design	RSA Cycle	Delay (DC) [ns]	Delay (Astro) [ns]	MM Time [μs]	RSA Calc. Time [ms]	Chip Area [Kgate]	AD [ms·Kgate]	Power [mW]
This Work	**Type-I**									
	2^8	Speed	51,286K	1.70	1.81	60.11	92.39	2.54	234.51	0.96
		Balance		1.99	2.05	68.32	105.01	1.74	183.13	0.70
		Area		**3.09**	**3.42**	**113.78**	**174.89**	**1.25**	**218.51**	**0.34**
	2^{16}	Speed	13,053K	2.10	2.34	19.76	30.37	8.90	270.31	2.69
		Balance		2.47	2.68	22.66	34.83	5.95	207.08	1.77
		Area		5.08	6.23	52.63	80.89	3.48	281.52	0.59
	2^{32}	Speed	3,381K	2.55	3.07	6.68	10.27	27.90	286.46	7.53
		Balance		3.04	3.47	7.56	11.62	20.01	232.58	4.77
		Area		9.73	10.75	23.39	35.97	11.85	426.09	1.38
	2^{64}	Speed	905K	2.95	4.34	2.50	3.85	116.58	449.04	30.37
		Balance		4.74	5.77	3.33	5.13	52.40	268.60	9.04
		Area		18.56	18.10	10.44	16.07	43.38	697.11	4.20
	2^{128}	Speed	258K	4.19	5.85	0.94	1.46	270.45	393.72	59.16
		Balance		4.68	6.62	1.07	1.65	233.12	384.14	45.78
		Area		36.42	34.57	5.57	8.60	164.39	1413.70	15.02
	Type-II									
	2^8	Speed	51,483K	1.75	1.70	56.47	86.80	1.74	151.37	0.81
		Balance		1.75	1.70	56.47	86.80	1.74	151.37	0.81
		Area		2.38	2.74	91.29	140.32	1.32	184.64	0.51
	2^{16}	Speed	13,152K	2.01	2.14	18.08	27.80	6.02	167.28	2.03
		Balance		2.22	2.48	20.96	32.23	5.07	163.49	1.66
		Area		3.47	3.67	30.99	47.63	3.90	185.91	0.89
	2^{32}	Speed	3,430K	2.68	3.09	6.73	10.34	18.25	188.79	4.69
		Balance		**2.65**	**3.13**	**6.82**	**10.48**	**17.82**	**186.84**	**4.55**
		Area		5.51	5.68	12.38	19.03	12.35	234.94	1.87
	2^{64}	Speed	930K	3.46	3.99	2.30	3.54	60.40	213.79	12.29
		Balance		3.46	3.99	2.30	3.54	60.40	213.79	12.29
		Area		1.73	10.21	5.89	9.07	44.33	401.88	4.95
	2^{128}	**Speed**	270K	**4.27**	**5.82**	**0.94**	**1.45**	**227.16**	**328.87**	**41.88**
		Balance		4.46	6.30	1.01	1.57	198.39	310.94	31.60
		Area		19.21	19.75	3.18	4.91	166.08	815.93	16.07
[2]	2^{64}	64bit-Mult.	-	7.26	-	4.57	-	96.224	-	-
[3]	2	40PEs×8bit	-	12.5	-	43	88.2	28	2469.6	-
[5]	2	7PEs×32bit	-	12.5	-	61	-	-	-	-

Arithmetic, pp. 172–178. IEEE Computer Society, September 2005.

[7] T. Blum and C. Paar. Montgomery modular exponentiation on reconfigurable hardware. In *ARITH '99: Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, pp. 70–78. IEEE Computer Society, February 1999.

[8] T. Blum and C. Paar. High-radix Montgomery modular exponentiation on reconfigurable hardware. *IEEE Trans. Comput.*, Vol. 50, No. 7, pp. 759–764, 2001.

[9] C. K. Koc, T. Acar, and B. S. Kaliski. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, Vol. 16, No. 3, pp. 26–33, June 1996.

[10] Y. Watanabe, N. Homma, T. Aoki, and T. Higuchi. Arithmetic module generator with algorithm optimization capability. In *the 2008 IEEE International Symposium on Circuits and Systems*, pp. 1796–1799, May 2008.

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

[12] N. Homma, Y. Watanabe, T. Aoki, and T. Higuchi. Formal design of arithmetic circuits based on arithmetic description language. *IEICE Trans. Fundamentals.*, Vol. E89-A, No. 12, pp. 3500–3509, December 2006.

[13] I. Koren. *Computer arithmetic algorithms 2nd Edition.* A K Peters, 2001.

[14] Circuits Multi-Projets (CMP) CMOS 90nm from STMicroelectronics. http://cmp.imag.fr/products/ic/?p=STCMOS090.

[15] J. J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics Letters*, Vol. 18, No. 21, pp. 905–907, October 1982.