社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2009-SLDM-138 (13)
2009 / 1 / 29

# MXコアにおけるPE粒度変更による実行効率の改善

溝上 雄太† 中野 光臣† 飯田 全広†† 末吉 敏則††

† †† 熊本大学大学院 自然科学研究科 〒 860-8555 熊本市黒髪 2-39-1
E-mail: †{mizokami,nakano}@arch.cs.kumamoto-u.ac.jp, ††{iida,sueyoshi}@cs.kumamoto-u.ac.jp

**あらまし** MX コアは細粒度の演算器（Prpcessing Element：PE）を複数搭載した超並列 SIMD（Single Instruction Multiple Data）型プロセッサである．MX コアは細粒度 PE を超並列に動作させることでピーク性能を高めている．よって，その性能は PE の稼働状況（並列度）に依存する．一方，アプリケーションは一般に演算データのサイズや処理内容によって演算並列度は一定ではないため，常にすべての PE で処理を行えるわけではない．本稿では，並列度の低いアプリケーションにおいて PE の稼働率を向上させる手法として，演算粒度を変更可能な PE アーキテクチャを提案する．提案アーキテクチャを RSA 暗号に適用した結果，従来アーキテクチャと比較して 34%の性能改善が得られた．

**キーワード** MX コア，SIMD，粒度

# Improvement of Execution Efficiency by Applying Unitable PE Architecture for MX Core

Yuta MIZOKAMI†, Mitsutaka NAKANO†, Masahiro IIDA††, and Toshinori SUEYOSHI††

† †† Department of Mathematics and Computer Science, Graduate School of Science and Technology, Kumamoto University, 2-39-1 Kurokami, Kumamoto-shi, 860-8555 Japan
E-mail: †{mizokami,nakano}@arch.cs.kumamoto-u.ac.jp, ††{iida,sueyoshi}@cs.kumamoto-u.ac.jp

**Abstract** MX-Core is a massively parallel SIMD(Single Instruction Multiple Data) type processor which have fine-grained computing units (PE). The performance of MX-Core depends on the utilization rate of PEs. Therefore, it is necessary to achieve a high operational perfomance that it operate with high parallelism. However, the instruction level parallelism depends on applications or processing ,which is cause of performance deterioration in MX-Core. In this study,we proposed the unitable PE architecture. This architecture solves the parallelism problem of application. As a result, as compared with traditional architecture, the proposed architecture to RSA cryptography improves performance by 34% .

**Key words** MX core, SIMD, granularity

## 1. はじめに

近年、デジタルカメラや携帯電話をはじめとしたマルチメディア機器が普及し，高機能化，高性能化が進んでいる．特に画像処理を行う機器は，高画素化による処理データ量の飛躍的な増大だけでなく，機器の付加価値を高めるために，多彩で高度な画像処理機能を追加する傾向にあり，高い演算性能が要求される．また，機器の短期間での開発サイクルに対応するため，ソフトウェアによる機能変更の柔軟性も要求されている．加えて，携帯機器では，低消費電力であることが必須となる．

このような背景から，株式会社ルネサステクノロジは，マトリックス構造の超並列プロセッサ（以下 MX コア）を開発した[1]．MX コアは，細粒度のビットシリアル演算器（PE：Processing Element）を 1,024 個搭載しており，これらの演算器群を並列実行させることにより高速な処理を実現する．また，MX コアはメモリ技術ベースのプロセス技術により，小面積化かつ低消費電力化を実現している．

MX コアの性能は，演算時にいくつの PE が稼動しているか，即ち演算の並列度に依存する．従って，高い演算性能を実現するには高い演算並列度を実現する必要がある．しかし，演算並列度はアプリケーションや処理内容によって異なり，並列度の低いアプリケーションでは性能低下の要因の一つになっている．そこで本稿では，アプリケーションに依存せずに高い演算並列度を実現するための手法を提案する．提案手法では，処

理内容に応じて演算粒度を選択することによって実行効率を改善する．本稿では，提案手法を用いてアプリケーションを実装し，シミュレーションを行うことにより提案手法の有効性を評価する．

以下，第2章ではMXコアの特徴について説明する．第3章では従来手法における問題点を述べ，第4章ではその問題点を解決するために本稿で提案する手法について述べる．第5章では提案手法の評価を行い，その考察を行う．第6章でまとめと今後の課題を述べる．

## 2. MX コア

### 2.1 MX コアの構成

MXコアの構造を図1に示す．MXコアは，全体の制御を行うためのホストCPUやデータの高速転送を行うためのDMAC (Direct Memory Access Controller) などと共に1チップ上に実装される．MXコアは，このようなSoCシステム上において，主にメディアアプリケーションを処理するアクセラレータとして機能する．

MXコアは，ホストCPUや外部メモリと通信を行う「I/Oインタフェース」，MXコアで行う処理を制御する「コントローラ」および並列演算処理を行う「マトリクス演算部」から構成される．MXコアは，コントローラ内に設けられている命令メモリの内容を書き換えることによって演算処理の内容を柔軟に変更できる．

マトリクス演算部は，2ビット演算器 (PE) とその両翼に512ビットずつ設けられたデータレジスタの対を基本構成としており，これをエントリと呼ぶ．MXコアは，エントリを1次元アレイ状に1,024個並べ，SIMD (Single Instruction Multiple Data) 型の並列演算を行うことにより高速処理を実現する．PEと両翼のデータレジスタ間はHチャネル (H-ch) で配線されており，データ干渉することなく，1サイクルで水平方向のデータ通信が可能である．また，PE間（エントリ間）で垂直方向のデータ通信を行う際は，Vチャネル (V-ch) を使用することで，一定の距離にあるPE間で一律にデータ通信を行うことができる．

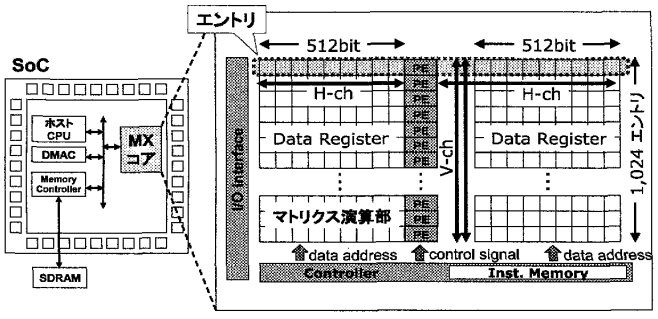


図1　MXコアの構成

### 2.2 PE の構成

MXコアのエントリ構成を図2に示す．エントリは，PEとその両翼にあるデータレジスタから構成される．PEは，両翼のデータレジスタから読み出したオペランドを格納するための2つの1ビットレジスタ（Xレジスタおよび XHレジスタ），各種フラグ操作を行うための4つの1ビットレジスタ（S，D，F，C），2ビット処理を行うための演算回路などから構成される．MXコアの演算はこれらのレジスタを使用して次の手順で行われる．

（1） 左翼のデータレジスタからX,XHレジスタへデータを読み出す
（2） Dレジスタの値により入力データの選択，またFレジスタの値によりデータを値を反転させる
（3） X,XHレジスタの値（IN1），右翼のデータレジスタの値（IN2），キャリーレジスタCの値（CIN）を演算回路にて処理する
（4） 演算結果（OUT）をデータレジスタへ，桁上げ（COUT）をキャリーレジスタへ格納する

上記の演算は1サイクルで実行可能であり，演算データのデータ長分だけ繰り返される．また，X，XHレジスタは，垂直方向のデータ通信用配線（Vチャネル）とも接続されており，異なるエントリのデータレジスタに配置されるオペランドとの演算も可能である．

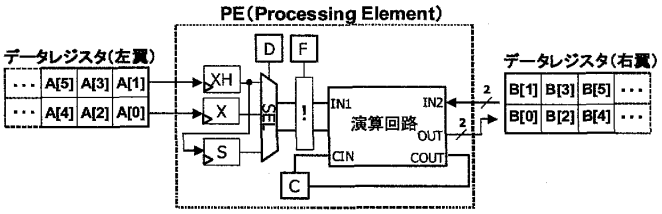


図2　エントリの構成

### 2.3 MX コアの命令セット

MXコアは1ビットおよび2ビットの各種演算機能を備えている．それらの命令セットを以下に示す．これらの命令の組み合わせにより任意の処理を実現する．

- ロード/ストア命令
- フラグ操作命令
- 算術論理演算命令
- PE間データ通信命令
- レジスタ-レジスタ間の演算命令/移動命令

## 3. 従来 PE アーキテクチャの問題点

### 並列度低下の問題

MXコアにおいて高い演算性能を実現するためには，高い並列度で演算を行う必要がある．しかし，処理内容によっては並列度が低下する場合があり，MXコアの性能低下につながっている．これを並列度低下の問題と定義し，その例を図3に示す．なお，データレジスタの右翼は省略している．

図3 (a) では，依存関係のあるデータ群 (Data1 から Data32) が複数エントリにわたって配置されている．データ群の各データには依存関係があり，シーケンシャルな処理が行われる．つまり，Data[K] の演算は，Data[K-1] の演算結果が決定するまで待たされる．従って，データ群が2エントリにわたる場合，

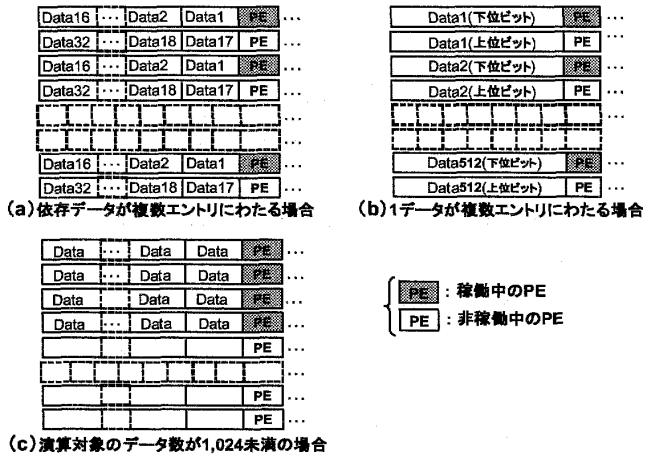


図 3　並列度低下の例

どちらか一方のエントリは演算を行うことができない．そのため，MX コアの並列度はおよそ半分になる．

図 3（b）では，取り扱うデータのサイズが大きく，1 データが 2 エントリにわたって配置されている．MX コアの演算は，下位ビットからビットシリアルに行われるため，上位ビットの演算は下位ビットの演算終了まで待たされる．そのため，演算の並列度はおよそ半分になる．

図 3（c）では，MX コアの全エントリのうち，約半分のエントリのみにデータが配置されている．演算対象のデータ数が 1,024 に満たない場合，演算データが配置されていないエントリの PE は稼動しない．従って，MX コアの並列度は低下する．

このように，演算データが複数エントリにまたがり，なおかつ互いのエントリ間の演算データに依存関係がある場合や演算対象のデータ数が 1,024 未満の場合において MX コアの実行効率は低下する．

## 4.　提案アーキテクチャ

### 4.1　概　　要

提案アーキテクチャの概要を図 4 に示す．図 4（a）は，従来のエントリ構成であり，PE の演算粒度は 2 ビットである．一方，図 4（b）は，隣接する 2 エントリを 1 エントリとみなした構成であり，PE の演算粒度は 4 ビットに粗粒度化されている．本提案アーキテクチャは，このような演算粒度の変更を可能にする．なお，演算粒度は，コントローラからの制御信号（*Select*）によって一律に選択される．

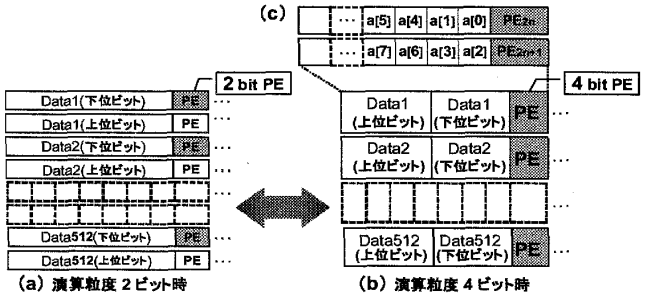


図 4　演算粒度の変更

### 4.2　粒度可変 PE アーキテクチャ

4.1 節で述べた粒度変更を実現するためには，データ配置の工夫および PE アーキテクチャの一部変更が必要である．以下，この 2 点について述べる．

#### 4.2.1　データ配置

提案手法では，2 つの PE に対して 4 ビット幅でのデータ入出力が必要になる．また，4 ビットの入出力データは下位から連続していなければならない．そこで，演算粒度 4 ビット時は，図 4（c）に示すように偶数エントリと奇数エントリ間においてデータを 2 ビット単位で交互に配置する．このデータ配置を取ることにより，データレジスター PE 間の配線リソース（H-ch）を増加させることなく，4 ビット幅のデータ入出力が可能になる．

#### 4.2.2　PE アーキテクチャ

提案する粒度可変 PE アーキテクチャを図 5 に示す．太線部は，従来の PE アーキテクチャ（図 2）に対して新たに追加したリソースを表す．以下，追加リソースの用途別に説明を行う．

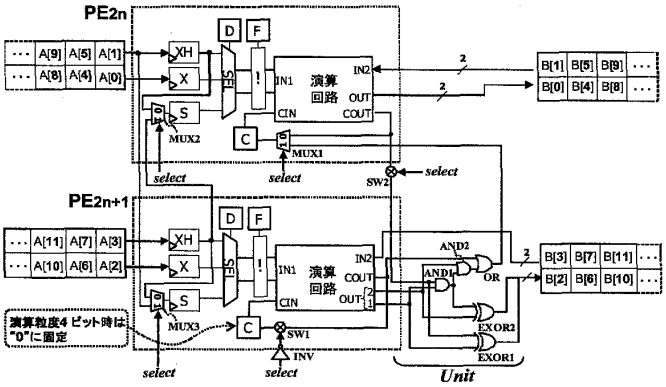


図 5　粒度可変 PE アーキテクチャ

#### (1) 桁上げ伝播遅延の削減用ユニット（*Unit*）

演算粒度 4 ビットの演算を行うためには，偶数エントリの PE（以下 $PE_{2n}$）と奇数エントリの PE（以下 $PE_{2n+1}$）の間で桁上げ伝播を行う必要がある．そこで，提案アーキテクチャでは，$PE_{2n}$ と $PE_{2n+1}$ をカスケード接続することで桁上げの伝播を行う．しかし，単純に 2 つの PE をカスケード接続した場合，粗粒度化後の PE において従来の 2 倍の組合せ遅延が発生することになり，動作周波数を低下させてしまう．そこで本研究では，加算結果に表れる規則性を利用することにより，桁上げ伝播遅延を削減する．

**加算結果の規則性**

本提案アーキテクチャで利用する加算結果の規則性を図 6 の 4 ビット加算を例に説明する．

$a[0]-a[3]$ および $b[0]-b[3]$ という 4 ビット入力のうち下位 2 ビットを下位桁ブロック，上位 2 ビットを上位桁ブロックと呼ぶ．また，下位桁ブロックで発生する桁上げを $C_2$，上位桁ブロックで発生する桁上げを $C_4$ とする．このとき，下位桁ブロックからの桁上げなし（$C_2$=0）として計算した上位桁ブロッ

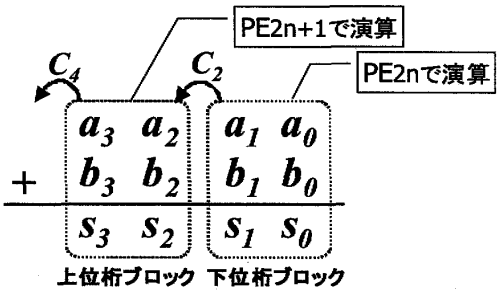


図 6　4 ビット加算

クの解（S2，S3，C4）と桁上げあり（$C_2$=1）として計算した上位桁ブロックの解（S2'，S3'，C4'）の間には次のような関係がある．

$$\mathbf{if}(\mathbf{C_2=1}) \quad S'_2=\bar{S}_2 \quad \mathbf{else} \quad S'_2=S_2 \quad (1)$$
$$\mathbf{if}(\mathbf{C_2=S_2=1}) \quad S'_3=\bar{S}_3 \quad \mathbf{else} \quad S'_3=S_3 \quad (2)$$
$$\mathbf{if}(\mathbf{C_2=S_2=S_3=1}) \quad C'_4=\bar{C}_4 \quad \mathbf{else} \quad C'_4=C_4 \quad (3)$$

桁上げなし（$C_2$=0）として求めた解（$S_2$，$S_3$，$C_4$）と，桁上げあり（$C_2$=1）として求めた解（$S'_2$，$S'_3$，$C'_4$）は，if 文の条件式を満たす場合に限り，値が反転する．if 文の条件式を満たさない場合は，（$S_2$，$S_3$，$C_4$）と（$S'_2$，$S'_3$，$C'_4$）は，同じ値になる．これは桁上げなし（$C_2$=0）として求めた解（$S_2$，$S_3$，$C_4$）から桁上げあり（$C_2$=1）として求めた解（$S'_2$，$S'_3$，$C'_4$）を生成できることを意味している．

本提案アーキテクチャでは $PE_{2n+1}$ に式（1）から式（3）の加算結果の規則性を反映したユニット（*Unit*）を追加することにより，回路の組合せ遅延の削減を行う．*Unit* は，演算粒度 4 ビット時は式（1）から式（3）に相当する処理を行うが，演算粒度 2 ビット時は演算回路から出力される演算結果をそのまま出力する．*Unit* を用いることにより，$PE_{2n}$ の演算回路と $PE_{2n+1}$ の演算回路を単純にカスケード接続するよりも，少ない遅延時間（ゲート段数）で演算可能となる．

**(2) ビットシフト用セレクタ（MUX2，MUX3）**

演算粒度 2 ビット時は，データレジスタに格納されている演算データに対する 1 ビット左シフトは次の手順で行われる．また，1 ビット左シフト演算の処理タイミングチャートを図 7 に示す．

（1） S レジスタを 0 で初期化する
（2） データレジスタから X，XH レジスタへ演算データを読み出す（X：下位ビット，XH：上位ビット）
（3） セレクタ（SEL）により，S，X レジスタの値を選択しデータレジスタへ書き込む
（4） XH レジスタの値を S レジスタに書き込む
（5） （2）から（4）を演算データのデータ長分だけ繰り返す

演算粒度 4 ビット時は，演算データが 2 エントリに跨って配置されるため，シフトを行うためには $PE_{2n}$ と $PE_{2n+1}$ 間で演算データの転送を行う必要がある．

そこで，提案アーキテクチャでは，$PE_{2n}$ と $PE_{2n+1}$ 間にデータ転送用の配線を設ける．そして，追加リソース MUX2 および MUX3 に対して制御信号（*Select*）を出力することにより，演算粒度に応じたデータ転送経路の選択を行う．演算粒度 4 ビット時の回路構成を図 8 に示す．$PE_{2n}$ と $PE_{2n+1}$ 間のデータ転送経路により，正しくシフト演算を行うことができる．MX コアの乗算は，2 次のブースアルゴリズに基づいて行われる．4 ビット演算時における乗算は，このシフト演算機構を用いて行われる．

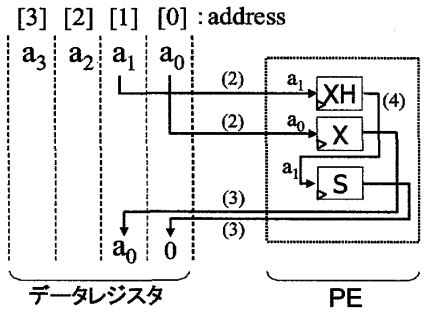


図 7　演算粒度 2 ビット時の 1 ビット左シフト演算

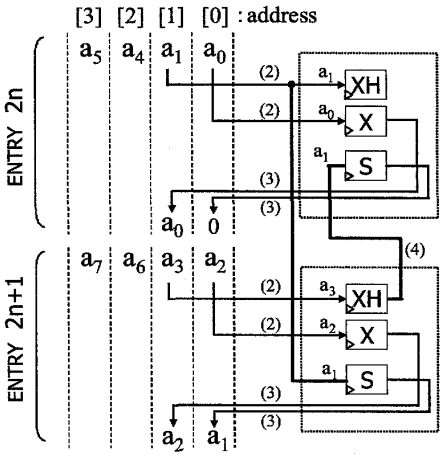


図 8　演算粒度 4 ビット時の 1 ビット左シフト演算

**(3) 桁上げ伝播パスの制御用回路（SW1/2，MUX1，INV）**

提案アーキテクチャは，演算粒度に応じて桁上げ伝播パスを切り替える必要がある．この桁上げ伝播パスの制御は，追加リソース SW1，SW2，MUX1，INV に対して制御信号（*select*）を出力することにより行う．各演算粒度における追加リソースの挙動は次のようになる．

**（ア）演算粒度 2 ビット時の制御**

演算粒度が 2 ビットの場合は，*select* 信号に“0”が出力される．この時，SW2 および MUX1 により PE 間を接続する桁上げ伝播パスは切断された状態になる．また，SW1 は導通状態となる．*Unit* は（1）で述べた通り，演算回路の演算結果をそのままデータレジスタへ出力する．従って，$PE_{2n}$ および $PE_{2n+1}$ はそれぞれ 2 ビット PE として動作する．

**（イ）演算粒度 4 ビット時の制御**

演算粒度が 4 ビットの場合は，*select* 信号に“1”が出力さ

れる．この時，SW2 および MUX1 により PE 間を接続する桁上げ伝播パスは接続された状態になる．また，SW1 は非導通状態となり，$PE_{2n+1}$ のキャリーレジスタの出力は“0”に固定される．*Unit* は（1）で述べた通り，演算回路の演算結果に対して加算結果の規則性に基づいた修正を加えた後データレジスタへ出力する．従って，$PE_{2n}$ および $PE_{2n+1}$ は，4 ビット PE として動作する．

#### 4.2.3 処理タイミング

演算粒度 4 ビット時における提案アーキテクチャの処理タイミングチャートを図 9 に示す．

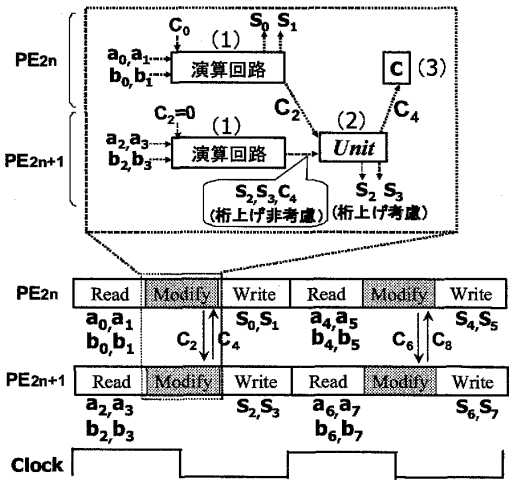


図 9　提案アーキテクチャの処理タイミングチャート

図 9 の加算は次の手順で行われる．

（1） $PE_{2n}$ が下位桁ブロックを加算する．これと並行して $PE_{2n+1}$ が上位桁ブロックを加算する（$C_2$=0 と仮定）

（2） $PE_{2n}$ の桁上げ $C_2$ が *Unit* へ伝播され，*Unit* にて $PE_{2n+1}$ で求めた解（$S_2$，$S_3$，$C_4$）を修正する（加算結果の規則性を利用）

（3） 桁上げ $C_4$ を $PE_{2n}$ のキャリーレジスタへ格納する

（4） (1) から (3) を演算データのデータ長分だけ繰り返す

提案手法では，*Unit* の導入により，$PE_{2n+1}$ は下位桁ブロックからの桁上げ（$C_2$）の伝播を待つことなく上位桁ブロックの演算を開始できる（手順 (1)）．

## 5. 評　　価

### 5.1 評価方法

提案アーキテクチャは，処理内容に応じて演算粒度を変更することが可能である．この粒度変更による並列度の改善効果は，粒度変更前の並列度や演算データのサイズなどに依存するため一般的な評価は難しい．本稿では提案アーキテクチャを有する MX コアを用いたアプリケーション実装と，従来アーキテクチャによる実装の並列度を比較する．そして，両実装の並列度を比較することにより，提案アーキテクチャの有効性を評価する．

評価にはルネサステクノロジ社の方式シミュレータを用いる．本評価では提案アーキテクチャを用いたアプリケーション実装を行うために，提案アーキテクチャ用のライブラリを追加している．このライブラリには，2.3 節で述べた四則演算やコピー命令などの命令セットを含まれる．なお，この方式シミュレータでは図 1 におけるコントローラ部および入出力部のオーバヘッドは考慮されない．

### 5.2 評価対象アプリケーション

#### 5.2.1 RSA 暗号の概要

評価対象のアプリケーションとして RSA 暗号 [2] の暗号化処理を実装する．RSA 暗号は公開鍵暗号方式の一種であり，情報の秘匿のみならず電子署名をも実現できる暗号方式である．暗号の強度を任意に変更できるという特徴があり，暗号の強度を示すセキュリティ・パラメータは，法 $n$ のビット長で表わされる．

RSA 暗号の鍵生成から暗号化までの処理は以下のステップにより行われる．ステップ（1）から（4）までが鍵生成フェーズであり，ステップ（6）が評価対象である暗号化処理フェーズである．暗号化処理は，べき乗および剰余演算が実行されるため，演算コストが非常に高い．

（1） 大きな桁数の素数 2 個 $(p, q)$ を生成

（2） 鍵ペア（公開鍵，秘密鍵）の作成に必要な $n, \phi(n)$ を計算

$$n = p \cdot q$$

$$\phi(n) = (p-1) \cdot (q-1)$$

（3） 適当な正整数 e の選択

（4） 鍵ペアの作成

公開鍵は {e,n}．秘密鍵は {d,n}

但し，$1 = d \cdot e \ mod \ \phi(n)$

（5） 暗号化に使用する公開鍵を公開

（6） 公開鍵を用いて，平文 $M(in)$ から暗号文 C を作成

$$C = M^e (mod\ n)$$

#### 5.2.2 RSA 暗号化処理の実装方法

本節では，RSA 暗号化処理を MX コアへ実装する方法について述べる．暗号化処理では，平文 M に対してべき乗剰余演算を行うことで暗号文 C を得る．通常 e の値は 65,537 に設定されるため，暗号化には平文 M を 65,537 乗して剰余演算を行う．このままでは演算コストが非常に大きいため，本稿では左向きバイナリ法 [3] を用いることで乗算回数を削減する．

左向きバイナリ法とは，指数を $a^{2x+1} = (a^x)^2 \times a$ と変換できる性質を利用したものである．左向きバイナリ法を用いることで，$M^{65,537}$ は $(M^{256})^2 \times M$ に変換でき，さらに $M^{256}$ は $(M^{16})^2$ に変換できる．この操作を繰り返すことにより，$M^{65,537} = (\cdots((((M^2)^2)^2)^2)^2\cdots) \times M$ と展開でき，その結果，$M$ の 65,537 回のべき乗算は 16 回のべき乗算と 1 回の乗算で実現できる．この時，メモリ使用量を削減するために，乗算を行った直後に剰余演算を行う．これにより演算の途中結果を 2n ビット以下に抑える事ができる．

以上を踏まえたアルゴリズムを表 1 に示す．MX コアは，演算負荷の高い Step4 および Step5 の乗算および剰余演算を行い，残りの処理はホスト CPU にて行う．

表 1　左向きバイナリ法を用いたべき乗剰余演算

| Input: | $n, 0 < M < n, e$ |
|---|---|
| Output: | $M^e mod\ n$ |
| Step1: | $A = M$ |
| Step2: | $i = \lceil log_2 e \rceil, j = 0$ |
| Step3: | $if(e_0 = 1)\ C = M$ |
| | $else\ C = 1$ |
| **Step4:** | $\mathbf{A = A \times A\ mod\ n},\ j{+}{+}$ |
| **Step5:** | $if(e_j = 1)\ \mathbf{C = C \times A\ mod\ n}$ |
| Step6: | $if(i \neq j)\ return\ Step4$ |
| | $else\ output C$ |

RSA 暗号を実装した際のデータ配置を図 10 に示す．なお，今回 RSA 暗号で使用する鍵長は 1,024 ビットとする．図中の積 **B** および剰余データ **D** は，表 1 の Step4 および Step5 によって算出される乗算および剰余演算の解を示す．また，テンポラリ領域は演算結果を一時的に格納するために使用する．

鍵長 1,024 ビット時の RSA 暗号で使用する演算データのサイズは 1,024 ビットである．また，乗算は，積を保持するために乗数の 2 倍のメモリ量が必要でである．MX コアのデータレジスタの片翼のサイズは 512 ビットであり，さらに複数の演算データを配置する必要があるため，今回の実装では 1 エントリあたり 160 ビットごとに折り返して 13 エントリにわたってデータ配置を行う．従って，これらのデータに対して演算を行う場合は，最下位ビットから逐次的に演算していく必要があるため，図 3（b）で述べた並列度低下の問題が発生する．

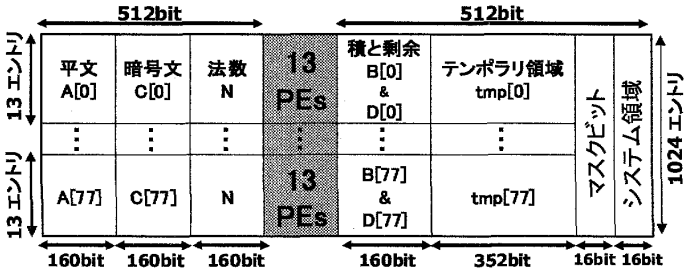


図 10　RSA 暗号実装時のデータ配置

### 5.3　評価結果と考察

従来手法および提案手法の 2 つの手法を用いて RSA 暗号を実装した時の評価結果を表 2 に示す．提案手法は常に演算粒度を 4 ビットとし，処理の途中に演算粒度の変更は行わない．なお，評価項目は，演算時間（200MHz 動作時）と並列度（PE 稼働率）である．並列度は単位時間（1 サイクル）あたりに稼動中の PE の割合であり，次の式によって求められる．

$$PE\ 稼働率\ [\%] = \frac{\sum(命令の並列度 \times 演算時間)}{1,024 \times 演算時間の合計} \times 100$$

式中の“各命令の並列度”とはその命令の実行中に稼動している PE の個数であり，“演算時間”とはその命令の実行サイクル数である．分子は RSA 暗号で使用する全命令に対してこれら 2 値の積を取ったものの総和である．分母は，全命令が並列度 1,024 で動作したと過程した場合の上記の積の総和である．

表 2　演算時間と並列度の比較（鍵長：1,024 ビット）

| 評価項目 | 従来手法 | 提案手法 | 効果 |
|---|---|---|---|
| 演算時間 [sec] | 0.448 | 0.295 | 34%削減 |
| 並列度 [%] | 34 | 42 | 8%向上 |

表 2 より，提案手法は従来手法と比べて性能は約 1.5 倍向上，並列度は約 8% 改善されている．

評価結果より，提案手法と従来手法において大きな性能向上は見られない．これは RSA 暗号の実装において，乗算および剰余演算の処理でシフト演算を多用したためである．シフト演算は高い並列度で実行されるが，多くの処理時間を要すという特徴がある．また，シフト演算において両手法に性能の差は出ないため，表 2 のような結果になったと言える．

今回の RSA 暗号の実装では，1 つの演算データが 13 エントリにわたって配置される．これは 4 ビット以上の粗粒度化により，さらなる性能向上が期待できることを意味する．4 章で述べた提案アーキテクチャは，*Unit* を割当てる PE の数を増やすことによってさらなる粗粒度化が可能である．提案アーキテクチャが，さらに演算粒度 8 ビットへの粗粒度化が可能であると仮定して RSA 暗号を実装した場合，演算粒度 8 ビットにおいて並列度は 66%に改善される．

## 6.　まとめと今後の課題

本稿では，アプリケーションにおいて PE 稼働率を向上させる手法として，演算粒度を変更可能な PE アーキテクチャを提案した．提案アーキテクチャを RSA 暗号に適用した結果，従来アーキテクチャと比較して 34% の性能改善が得られた．

本稿では対象アプリケーションに対して単一の演算粒度で実装を行ったが，アプリケーションによっては演算粒度を切り替えて実装した方が高い性能を実現できる可能性がある．今後は，単一アプリケーションに対して複数の演算粒度を適用した場合の評価を行う必要がある．但し，単一アプリケーション内で複数の演算粒度を使用するとデータ配置の相違により実装が困難になることが予想される．従って，今後は適切な演算粒度を解析し，命令を実装するツールの開発が必要である．

### 文　献

[1] K.Mizumoto, et al, *"A Multi matrix-processor core architecture for real-time image processing Soc"*, A-SSCC no. 6-2, pp. 80-183, Jeju, Korea, Nov. 2007.

[2] 岡本龍明，太田和夫，情報処理学会　監修 **"暗号・ゼロ知識証明数論"**．共立出版株式会社，1995 年．

[3] Johan Torkel Hastad *"Pseudo-random generators under uniform assumptions"*. Annual ACM Symposium on Theory of Computing Proceedings of the twenty-second annual ACM symposium on Theory of computing, pp395-404, 1990.