# 冗長２進数表現を利用したべき乗剰余演算回路の分割処理

# Division Method of Modular Exponentiation Using Redundant Binary Representation

長谷川 俊夫‡
Toshio HASEGAWA

山元 浩幸‡‡
Hiroyuki YAMAMOTO

‡三菱電機（株）情報技術総合研究所

〒247-8501 鎌倉市大船 5-1-1

Information Technology R&D Center
Mitsubishi Electric Corporation
5-1-1 Ofuna, Kamakura, 247-8501, Japan

‡‡三菱電機（株）設計システム技術センター

〒220-8120 横浜市西区みなとみらい 2-1-1

Design Systems Engineering Center
Mitsubishi Electric Corporation
2-1-1 Minatomirai, Nishi, Yokohama, 222-8120, Japan

E-mail : toshio@iss.isl.melco.co.jp, yamamoto@ds.lmt.melco.co.jp

**あらまし** 公開鍵暗号等で使用される演算は、千ビット長 (1024bit) 程度の多倍長整数べき乗剰余演算が基本となる。 この論文では、我々は冗長 2 進数表現を利用したべき乗剰余演算の分割処理方式を提案する。また我々は、分割処理方式の演算回路決定のパラメータをいくつか変化させたときの回路のハードウェア性能とゲート規模の概算を行ない、実際に論理合成ツールでいくつかのケースについて評価を行なった。その結果、冗長 2 進数表現を採用した分割処理を行なう高速演算回路を得ることができた。

**Abstract.** In this paper, we investigate how to realize long-bit (more than 1024 bit) modular exponentiation with small divided circuit using redundant binary representation. We roughly estimate performance and gate size and also evaluate several cases.

## 1．はじめに

電子商取引等が実用化されようとしている現在、暗号技術、情報セキュリティ技術が安全性を保つための重要な要素技術となる。暗号には、秘密鍵暗号と公開鍵暗号の2種類があるが、公開鍵暗号は秘密鍵暗号に比べて認証機能を実現するという点でメリットはあるものの処理時間がソフトウェアで約1000倍遅いことが問題であった。そこで将来Smart Cardのような制約の多いハードウェア環境で使用されることも考慮し、公開鍵暗号系のハードウェアによる高速化の検討の必要がある。

公開鍵暗号で使用される演算は、1024bit 長程度の多倍長整数べき乗剰余演算であり、更にこの演算の基本要素は乗算剰余演算である。乗算剰余演算の実現方法には大きく分けて2つある。1つは乗算を行なってから除算を行なう乗算除算法[2]、もう1つは乗算の部分積を求めながら部分積の除算を行なう逐次部分積法[3]である。前者は後者に比べてレジスタや乗算器を多く必要とするが制御は単純である。また後者は、乗算器の演算ビット幅も小さくすることができるので最小限の回路で構成しようとする場合に適しているが、高速演算には不向きであると考えられている。前者の方法で高速化の点で有利な方法としては、剰余テーブル法、モンゴメリ法[2]、冗長2進数法[4,5]であり、ハードウェア向きなものとしてモンゴメリ法、冗長2進数法である。

モンゴメリ法はハードウェア実装で一番一般的なものであり、多くの実装例がある。また、モンゴメリ法以外では、冗長2進数で乗算除算法を採用したPC card に搭載可能な公開鍵暗号用プロセッサを開発した例がある[5]。これは 1024bitRSA の処理に33MHz で約80msec（5MHz では約650msec と推定）で処理可能である。また、最近では通常2進数表現でモンゴメリ法を使用せずに高速なプロセッサを開発し、5MHz で約210msec という時間で1024bit RSA処理を可能とした開発例もある[6]。一般に冗長2進数法は演算ビット長が長くなるほど高速化に対する効率が良いと考えられているために乗算除算法を採用して実装される。

そこで我々は、ゲート規模が大きくなってしまう冗長2進数法で、乗算除算法ではなくゲートサイズ減少のため逐次部分積法でのハードウェア実現を試みた。しかし、一般に演算回路を演算ビット幅分全部所有すると非常に高速処理可能な演算回路ができるが、ゲート規模は逐次部分積法を使用しても、非常に大きくなってしまうという問題点がある。そのため我々は、演算器を商決めなどの特殊な処理が必要な上位ビットの演算器部と基本的にはビットスライスな残りの部分をある小さな処理ビット幅分の演算器のみを持つことによって小型化を図るような分割処理を検討し、実際に実装評価を行なった。

実装した方式は、べき乗剰余演算をハードウェアで実現す際に高速演算処理と VLSI 実装向きという2点で優れていると考えられている方式[1]である。高速演算処理のために、繰り返し加算などで効力を発揮するキャリー伝搬のない冗長2進数表現を使用しており、またVLSI実装に適するように、ビットスライス型の規則正しいセルラアレイの回路構造をとる。

本報告では、まず2章で準備として公開鍵暗号で使用される演算処理について説明する。3章では今回採用・実装したハードウェアアルゴリズム（乗算剰余演算）について記述する。そして4章では実際に実装する際の基本ハードウェア構成を示し、今回の分割処理実現方式を提案する。5章で分割処理方式の予想ハードウェア性能とその合成結果について議論する。6章で考察とまとめを述べる。

## 2. 公開鍵暗号での演算処理

公開鍵暗号やディジタル署名アルゴリズムの基本演算は剰余演算（具体的には乗算剰余 、べき乗剰余、逆元剰余演算）である。その中で特に、べき乗剰余演算の高速化が必要である。

(1)乗算剰余　　$A \bullet B \bmod N$

(2)べき乗剰余　　$X^e \bmod N$

(3)逆元剰余演算　　$A^{-1} \bmod N$

**べき乗剰余演算の簡単な実現例(binary method)**

$C := M^e \bmod Q$ (M, C, e, Q: 1024bit 長の整数)

e : k-bit binary number$[1e_{k-2}e_{k-3}\cdots e_0]$

$C := M$

for $i := k-2$ down to 0 do

begin

$C := C^2 \bmod Q$

if $e_i = 1$ then $C := C \bullet M \bmod Q$

end

## 3. ハードウェアアルゴリズム

### 3-1 アルゴリズム概要

今回採用した方式は高木が[1]で提案したものである。アルゴリズムの特徴は次の通り。

・乗数 Y を桁集合が{-2,-1,0,1,2}で 4 進拡張SD(signed-digit)表現$\hat{Y}$に変え、2bit ごとに処理

・冗長 2 進表現によるキャリー伝播なしの高速加算

・逐次部分積法

**演算式** $P = X \cdot Y \bmod Q$

**入力**

X, Y：n-digit 冗長 2 進数

$(-d_1 \cdot Q < X, Y < d_1 \cdot Q$ 但し $\frac{9}{16} \le d_1 \le \frac{5}{8})$

Q：n-digit 通常 2 進数 $(2^{n-1} \le Q < 2^n)$

**出力**

P: n-digit 冗長 2 進数 $(-d_1 \cdot Q < P < d_1 \cdot Q)$

**準備**

冗長 2 進数 Y を Signed-Digit radix4 表現の Y' に変換する

**アルゴリズム**

$P_{\lfloor \frac{n}{2} \rfloor + 1} := 0$ (n+2 digit)

for $j := \lfloor \frac{n}{2} \rfloor$ down to -1 do

begin

( ①Calulate $y_j'$ )

②$R_j := 4 \cdot P_{j+1} + y_j' \cdot X$ (n+4)-digit

( ③Determine $c_j$ )

④$P_j := R_j - 4 \cdot c_j \cdot Q$ (n+2)-digit

end

$P := \frac{P_{-1}}{4}$ n-digit

[$y_j'$の計算]

$$Y' = [y'_{\lfloor \frac{n}{2} \rfloor} y'_{\lfloor \frac{n}{2} \rfloor - 1} y'_{\lfloor \frac{n}{2} \rfloor - 2} \cdots y'_0] = \sum_{i=0}^{\lfloor n/2 \rfloor} y'_i \cdot 4^i$$

但し、$y'_i \in \{\bar{2}, \bar{1}, 0, 1, 2\}$, $y'_{-1} = 0$

[$c_j$の決定]

$c_j := \bar{2}$ if $top8(R_j) < -top7(6Q)$

$\bar{1}$ if $-top7(6Q) \le top8(R_j) < -top5(2Q)$

0 if $-top5(2Q) \le top8(R_j) < top5(2Q)$

1 if $top5(2Q) \le top8(R_j) < top7(6Q)$

2 if $top7(6Q) \le top8(R_j)$

ここで、top8( ), top7( ), top5( )は( )内の上位 8 ビット、上位 7 ビット、上位 5 ビットを意味する。

[冗長 2 進数同士 X,Y の加算テーブル]

$X = [x_{n-1} x_{n-2} \cdots x_0], Y = [y_{n-1} y_{n-2} \cdots y_0]$

ステップ 1. 中間キャリー $c_{i+1}$ と中間和 $d_i$ を求める

$$x_i + y_i = 2 \cdot c_{i+1} + d_i$$

$c_{i+1}$, $d_i$

| $x_i$ \ $y_i$ | $\bar{1}$ | 0 | 1 |
|---|---|---|---|
| $\bar{1}$ | $\bar{1}$,0 | 0, $\bar{1}$ / $\bar{1}$,1 | 0,0 |
| 0 | 0, $\bar{1}$ / $\bar{1}$,1 | 0,0 | 1, $\bar{1}$ / 0,1 |
| 1 | 0,0 | 1, $\bar{1}$ / 0,1 | 1,0 |

注: $x_{i-1}$, $y_{i-1}$がともに非負 / その他

ステップ 2. 和 $s_i$ を求める $d_i + c_i = s_i$

$s_i$

| $d_i$ \ $c_i$ | $\bar{1}$ | 0 | 1 |
|---|---|---|---|
| $\bar{1}$ | × | $\bar{1}$ | 0 |
| 0 | $\bar{1}$ | 0 | 1 |
| 1 | 0 | 1 | × |

注: × Never occurs

ここで、桁数が異なる冗長 2 進数同士の加算

$$R_j := 4 \cdot P_{j+1} + y_j' \cdot X$$

の上位ビットの処理や、桁数が異なる冗長 2 進数と通常 2 進数(0,1)の加算

$$P_j := R_j - 4 \cdot c_j \cdot Q$$

の上位ビットの処理は別の手続きを経て求める[1]。

## 4. 基本アーキテクチャと分割処理実現方式

一般に前章の乗算剰余演算の方式をそのままハードウェア化すると、商決め上位演算ブロックを除けば、残りの部分はビットスライスの構造をしているため、1 段で処理ビット幅全部演算回路として所有する構成で容易に実現できる。いまこのアーキテクチャを基本アーキテクチャ（図3参照）と呼ぶことにする。この構成だと高速処理が可能であるが、ゲート規模的には非常に大きくなってしまう。

このため、我々は小さな演算回路で処理ができるような分割処理方式を検討した。まずは1段の分割処理実現方式のアーキテクチャを、さらには1clockでNステップ分の処理を行なう、一般的なN段の拡張分割処理実現方式のアーキテクチャを提案する。

### 4-1 基本アーキテクチャ

まず基本アーキテクチャ[1]であるが、アーキテクチャを説明する前に、演算回路の基本構成要素についてふれる。演算回路の基本構成要素は、次の商決め部分を含む上位部分の演算回路と、上位部分を除く中間演算回路部分である。

商決め部分を含む上位部分の演算回路（1段分）

上位部分は乗算剰余演算の除算をする際の商決めの操作が必要で、ここでは上位 8 ビットの値から商を決めている。そのため 上位ブロックは、残りのビットスライスな構造とは回路が異なるため、分割処理する際もここは別に構成する。

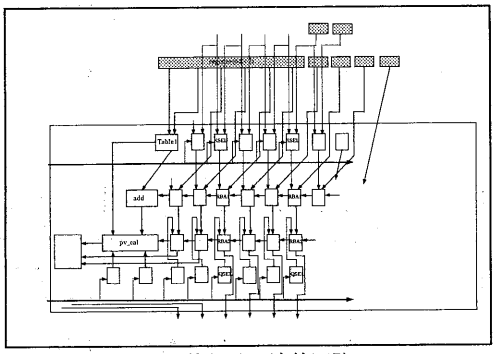


図1 上位部分の演算回路

上位部分を除く中間演算部分の演算回路（1段分）

上位部分を除く中間演算部分の演算回路部分に関しては、最下位ビットの処理を除けば、完全なビットスライス構造をとる。よって分割処理する際には、処理ビット長に対応した幅分の演算回路を持って上位ビット側へ順次処理をずらしていけばよい。

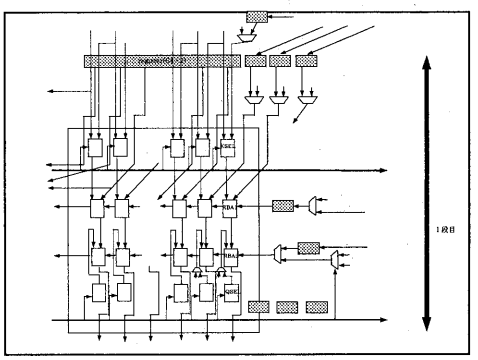


図2 中間演算部分の演算回路

また図1、2でRBA1は冗長2進数同士の加算器、RBA2は通常2進数と冗長2進数の加算器を意味する。

○基本アーキテクチャ

基本アーキテクチャは次の2つの部分から成る。

・上位部分の演算回路

・処理ビット長（1024bit 程度）に対応した中間演算部分の演算回路

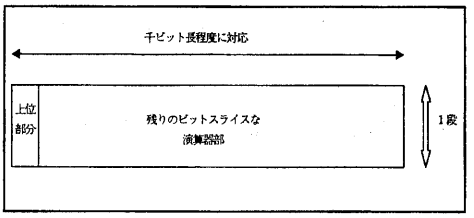


図3 基本アーキテクチャ

### 4-2 分割処理実現方式

次に分割処理実現方式のアーキテクチャを説明する。

○分割処理実現方式のアーキテクチャ

次の2つの部分から成る。

・上位部分の演算回路

・処理ビット長(32bit～256bit)に対応した中間演算部分の演算回路

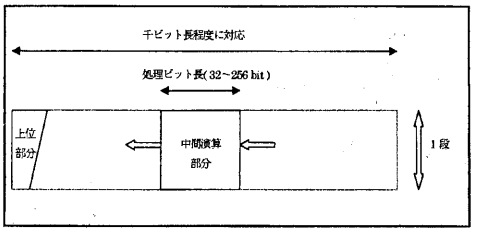


図4 分割処理実現方式のアーキテクチャ

○拡張分割処理実現方式のアーキテクチャ
・上位部分の演算回路
・処理ビット長(32bit～256bit)に対応した中間演算部分の演算回路
上記を動作周波数に対応する時間内に収まる最大遅延時間となる段数分、所有する。

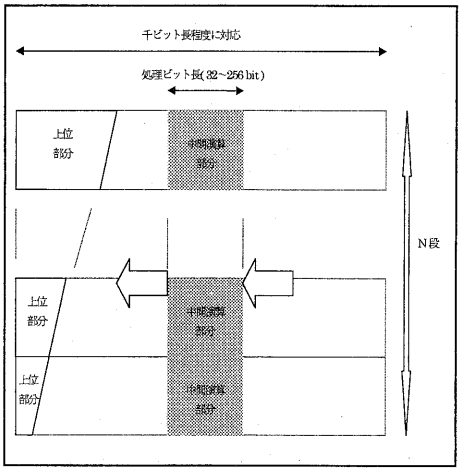

図5 拡張分割処理実現方式のアーキテクチャ

上位部分で、処理ビット幅が、段数を積むにつれて大きくなるが、これは商決めの為に必要な一番下の段の演算中間結果上位 8bit の正しい値を得る必要があるという理由からである。

乗算剰余演算で、逐次部分積法で除算の商決めの操作が必要である。分割処理をすると、一般に演算結果は（下位）ブロックから順に求まっていく。通常 2 進数だと、商決めに必要な被除数の上位数ビットは分割処理の最後に求まる。そのため分割処理をするのに障害となる。しかし冗長 2 進数の場合、ある注目しているビットの演算結果はその 2 つ下位のビット、すなわち合計 6bit（3bit×2）から決る。そのため、商決めのための小さな上位演算ブロックは分離でき、分割処理が可能になる。

## 5. 分割処理方式の予想値と評価結果

今回の分割処理で処理ビット幅、回路段数というパラメータを変えた時のハードウェア性能の予想値（総処理時間など）は次ページの表 1 の通り。アーキテクチャとしては、4 章で述べた拡張分割処理実現方式のアーキテクチャを採用し、パラメータを変化させて論理合成ツールでの開発評価結果をもとに 1024bit 長多倍長整数べき乗剰余演算回路の予想ゲートサイズ、5MHz 動作時での演算処理時間を概算した。ゲートサイズに関しては、上位ブロック、中間ブロック、その他の部分を含んだ演算器の合計について記述した。

今回の拡張分割処理方式アーキテクチャでは一般に、分割数 d（処理ビット幅/演算器処理ビット幅）と段数 N と総演算処理時間 T の間には $T \propto d/N$ の関係にある。またゲートサイズ G は $G \propto N/d$ の関係にある。すなわち、N/d でハードウェア性能がほぼ決ると予想される。我々はまず、同じゲートサイズでより高速に処理できる回路、あるいは同じ総演算処理時間でより小さなゲートサイズで実現できる回路を追求する。

例えば、分割数 8 段数 4 の回路と、分割数 2 段数 1 の回路とは一般的に、ゲートサイズはほぼ同じで総処理時間（正確には総演算サイクル数）もほぼ同じと考えられる。ただし、段数が多い回路の方が論理合成を行なうと、論理圧縮が更に進みゲート規模の削減、1 段分に換算した最大遅延時間の短縮を図ることができると期待できる。また Smart card のように低クロックでのみ動作させる状況を想定している回路の場合、動作周波数に対応する周期内に、作成回路のクリティカルパスが収まるように段数を最大限に増やすことにより、総演算サイクル数を段数分の 1 に減少させることができる。

実際の開発評価は、16 分割(内部処理幅 64 ビット)8 段、8 分割（内部処理幅 128bit）6 段、8 分割（内部処理幅 128bit）4 段の 3 つを行なった。

## 6. 考察とまとめ

まず今回提案した分割処理方式の実現可能性を、実際に実装評価することにより確かめた。そして評価結果から、この回路基本構成で今回のターゲットの設計ルール（CMOS Gate Array）では、動作周波数 5MHz に対応する 200ns 内で最大の段数は 6 段程度という結果が得られた。また 16 分割(内部処理幅 64 ビット)8 段、8 分割（内部処理幅 128bit）6 段、8 分割（内部処理幅 128bit）4 段の開発評価を行なった結果、一般には同じゲート規模と考えられるケース（8 分割 4 段、16 分割 8 段）でも、段数が小さい方がゲート規模的に有利であるという結果が得られた。すなわち方式から考えるとハードウェア性能は、（分割数 d）/（段数 N）で決るが、段数を少なく、あるいは分割数を大きくした方が回路規模が少なくなる。この理由は、一般には段数を増やした方がまとめての論理圧縮が進み、回路規模的に小さくなると予想され、実際表 1 の中間ブロックのゲート規模を見るとその傾向が

表1 べき乗剰余演算回路エンジン部の性能とゲート規模予想と合成結果

| 回路構成 | | 性能 | ゲート規模（K gate） | | |
|---|---|---|---|---|---|
| 内部演算処理幅(bit) | 演算器段数 | べき乗剰余演算処理時間(ms) 5MHz動作時 | 演算器本体 | | |
| | | | 上位ブロック | 中間ブロック | 合計（その他含む） |
| 64 | 4 | 約800ms | 7 | 20 | 30 |
| 64 | 6 | 約600ms | 10 | 30 | 46 |
| 64 | 8 | - | 15 | 40 | 60 |
| | | | 15.7 | 37.3 | 67.5 |
| 128 | 4 | 約400ms | 7 | 40 | 50 |
| | | | 4.9 | 40.8 | 50 |
| 128 | 6 | 約300ms | 10 | 60 | 76 |
| | | | 11.4 | 61.8 | 86.5 |
| 128 | 8 | - | 15 | 80 | 100 |
| 256 | 4 | 約200ms | 7 | 80 | 90 |
| 256 | 6 | 約150ms | 10 | 100 | 116 |
| 256 | 8 | - | 15 | 120 | 140 |

但し、3.0V時の値　　評価結果

あるが、実際は商決め操作を含む上位部分が段数を増やすと回路規模が増大するからである。

また8分割4段の演算回路が、このアーキテクチャを前提とした場合、ゲートサイズ的にも小さく、5MHz 程度の低クロックで動作するような環境での実装を考えると、冗長2進数表現を採用している[5]と比較して、総処理時間が2／3以下と短いので、実用的なレベルに達していると考えられる。また今回は分割処理方式自体の実現可能性の確認も目的の一つであったため、更なる小型化への改善箇所をまだ残している。今後の課題は、更なるゲートサイズの縮小であると考えている。

## 謝辞

本研究を進めるにあたり、有意義な議論をさせて頂いた三菱電機の宮田裕行氏、牧野博之氏、山岸篤弘氏、三菱電機エンジニアリングの市川哲也氏に感謝いたします。

## 参考文献

[1]Naofumi Takagi, "Radix-4 Modular Multiplication Hardware Algorithm for Modular Exponentiation,"IEEE Trans. Comput.,vol.41, no.8, August 1992

[2]P.L.Montgomery,"Modular Multiplication Without Trial Division", Mathematics of Computation, 44(170) April 1985.

[3]E.F.Brickell," A Fast modular multiplication algorithm with application to two key cryptography," Advances in Cryptography CRYPTO'82,pp.51-60, Plenum Press, 1982

[4] A.Vandemeulebroecke, et al., "A new carry-free division algorithm and its application to a single-chip 1024-b RSA processor," IEEE Journal of Solid-State Circuits, vol.25,no3,pp.748-756, June 1990.

[5]石井、大山、田中、"高速公開鍵暗号プロセッサ"、電気情報通信学会論文誌 D-1 Vol.J80 D-1 No.8 pp.725-735(1997.8)

[6]A.Satoh, et al., "A High-Speed Small RSA Encryption LSI with Low Power Dissipation", ISW'97(Information Security Workshop) 1997.

[7]K.Hwang, Computer Arithmetic/Principles, Architecture and Design. New York:Wiley,1979.