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

2004－SLDM－117 (43)
2004／12／2

並列加算を用いた高速モンゴメリ乗算器の設計

平良 健太郎　　島尻 寛之　　吉田 たけお
琉球大学 工学部 情報工学科
E-mail: {kent,shimajiri,tyoshida}@fts.ie.u-ryukyu.ac.jp

あらまし： モンゴメリ乗算 (Montgomery Multiplication : MM) アルゴリズムは，剰余乗算を高速に計算することを目的として提案されたアルゴリズムである．本稿では，MM 乗算器の高速化を目的とし，MM アルゴリズムにおける 2 回の加算を並列に行う並列加算モンゴメリ乗算アルゴリズムを提案する．さらに，近年，注目を集めているワードベース構成の MM (WMM) 乗算器と提案アルゴリズムを組み合わせることを検討し，高速な WMM 乗算器の構成を示す．評価を行った結果，従来の WMM 乗算器の遅延時間を約 63%削減できることがわかった．
キーワード： モンゴメリ乗算アルゴリズム，剰余乗算，キャリーセーブ加算器

Design of Fast Montgomery Modular Multipliers using Parallel Addition

Kentaro TAIRA, Hiroyuki SHIMAJIRI and Takeo YOSHIDA
Department of Information Engineering, Faculty of Engineering,
University of the Ryukyus
E-mail: {kent,shimajiri,tyoshida}@fts.ie.u-ryukyu.ac.jp

Abstract : The Montgomery Multiplication (MM) algorithm is the most efficient algorithm for implementing modular multiplication. The MM algorithm for n-bit operands needs to perform n iterations and it performs a series of two additions and one shift operation at each iteration. In this paper, we propose a new algorithm that performs these two additions in parallel at each iteration. A MM multiplier based on our algorithm can reduce the computation time of each iteration as compared with the MM multiplier. Furthermore, we apply our algorithm to the Word-based MM (WMM) multiplier that is the most popular MM multiplier. According to our analysis, a WMM multiplier based on our algorithm can achieve approximately 63% speedup as compared with the WMM multiplier.
KEYWORDS : Montgomery Multiplication Algorithm, Modular Multiplication, Carry Save Adder

1 はじめに

モンゴメリ乗算 (Montgomery Multiplication : MM) アルゴリズム [1] は，剰余乗算を高速に計算することを目的として提案されたアルゴリズムであり，公開鍵暗号システムなどに広く用いられている．MM アルゴリズムでは，加算とシフト操作のみによって剰余乗算を計算できるため，MM アルゴリズムに基づく剰余乗算器 (以降，MM 乗算器と呼ぶ) は，単純な構成で実現することができる [2–4]．本稿では，MM 乗算器の高速化を目的とし，高速 MM 乗算器の構成について検討する．

MM アルゴリズムでは，2 回の加算と 1 回のシフト操作を逐次的に行い，これを入力オペランドの桁数回繰り返すことによって，剰余乗算を計算している．本稿では，MM 乗算器を高速化するために，MM アルゴリズムの各イタレーションで実行する 2 回の加算に着目し，これを並列に実行することができる並列加算モンゴメリ乗算 (Parallel Addition MM : PAMM) アルゴリズムを提案する．

また，これまでに様々な MM 乗算器 [2–7] が提案されており，近年では，入力オペランドのビット長の変化に対して柔軟に対応できるワードベース構成の MM 乗算器 (WMM 乗算器) に注目が集まっている [4, 5]．そこで本稿では，PAMM アルゴリズムを WMM 乗算器に適用することを検討し，高速な WMM 乗算器の構成を示す．

以降，**2** で MM アルゴリズムと WMM 乗算器で実現しているワードベース構成の MM (WMM)

アルゴリズムについて述べる．**3** では，PAMM アルゴリズムと並列加算を適用した WMM アルゴリズムについて述べる．**4** では，PAMM 乗算器と高速 WMM 乗算器の構成を示し，**5** でその性能評価を行う．

2 モンゴメリ乗算アルゴリズム

MM アルゴリズムは，乗数 $X = (x_{n-1}, \cdots, x_1, x_0)$，被乗数 $Y = (y_{n-1}, \cdots, y_1, y_0)$，法 $M = (m_{n-1}, \cdots, m_1, m_0)$ の 3 つの n ビットの整数に対して，剰余乗算 $\mathrm{MM}(X,Y,M)=XYR^{-1} \bmod M$ を計算する．ここで，$0 \leq X < M, 0 \leq Y < M$, $R = 2^n, 2^{n-1} < M < 2^n$, $\gcd(M,R) = 1$ とする．MM アルゴリズムを図 1 に，MM 乗算器の処理の流れを図 2(a) に示す．図 2(a) では，各イタレーションで加算する部分積と法を四角で囲っている．また Q_i は，$i(= 0, 1, \cdots, n-1)$ 番目のイタレーションにおいて法を加算するか否かを選択する信号を表している．以降の例では $n = 8$ とする．

図 1 のアルゴリズムからもわかるように，MM アルゴリズムでは，1 回のイタレーションにおいて部分積と法の加算を行い，それを n 回繰り返す．法の加算は部分積の加算結果に依存しているため，図 2(a) に示すように，MM 乗算器では部分積と法の加算を逐次的に行う必要がある．

また，図 2(b) に WMM 乗算器の処理の流れを示す．WMM アルゴリズムでは，Y と M を複数の w ビット長のワードに分割し，MM アルゴリズムにおける 2 つの加算を下位桁からワード毎に計算する．したがって，各イタレーションにおいて，ワード毎の処理を n/w 回繰り返すことになる．以下では，1 回のワード処理をワードイタレーションと呼ぶことにする．WMM アルゴリズムでは，x_iY の最初の (図 2(b) の点線で囲まれた) ワードイタレーションを処理する際に Q_i の値を決定し，全体の処理に n^2/w 回のワードイタレーションを実行する．なお，以降で示す例では $w = 2$ とする．

3 並列加算モンゴメリ乗算アルゴリズム

まず，PAMM アルゴリズムを図 3 に，PAMM 乗算器の処理の流れを図 4(a) に示す．なお，図 3 中の $[a]$ は，ガウス記号であり，a を越えない最大の整数を表す．

図 3 に示す PAMM アルゴリズムでは，図 1 に示す MM アルゴリズムの 3 行目の式 $S = (S + x_iY + M)/2$ を部分積加算 (PP) パートと法累積 (MA) パートの 2 つに分けて計算する．まず PP

```
Algorithm MM(X, Y, M)
    S = 0;

1:  for i=0 to n-1
2:    if ((S + x_iY) is odd ) then
3:      S = (S + x_iY + M) / 2;
4:    else
5:      S = (S + x_iY) / 2;
6:    end if
7:  end for

8:  if (S >= M) then
9:    S = S - M;
10: end if

11: return S;
end.
```

図 1　MM アルゴリズム

パートでは，

$$PP_t = PP + x_iY + C \tag{1}$$

を計算する．部分積と同時に加算する値 C については，後ほど詳しく述べる．一方 MA パートでは，PP パートの加算結果が奇数であった場合に，

$$MA_t = MA + M \tag{2}$$

を計算する．同じイタレーションにおいて，式 (1) と式 (2) は互いの中間結果に依存しないため，それぞれ並列に計算することができる．

いま，i 番目のイタレーションにおける v の値を $v^{(i)}$ と表すことにする．このとき，MA パートでは，i 番目のイタレーションにおいて，

$$PP_t^{\langle i-1 \rangle} + MA_t^{\langle i-1 \rangle} \tag{3}$$

の結果が奇数のときのみ式 (2) を実行する．

MM アルゴリズムでは，$S = (S + x_iY + M)/2$ または $S = (S + x_iY)/2$ の計算を行うとき，それぞれ $S + x_iY + M$ と $S + x_iY$ の値は偶数であることが保証されている．しかし PAMM アルゴリズムでは，図 3 の 11 行目と 21 行目での PP_t と MA_t は必ず偶数になるとは限らない．そのため，MM アルゴリズムでは発生し得ない誤差を生じる恐れがある．したがって，PAMM アルゴリズムでは発生した誤差を補正する必要がある．PAMM アルゴリズムにおける誤差の発生条件を表 1 に示す．なお，表中の C は PP_t の最下位桁の値を表す．

表 1 より，PP_t が奇数の場合，すなわち PP_t の最下位桁 C の値が 1 の場合に，切り捨て誤差が発生することがわかる．そのため PAMM アルゴリズムでは，式 (1) に示すように，i 番目のイタレー

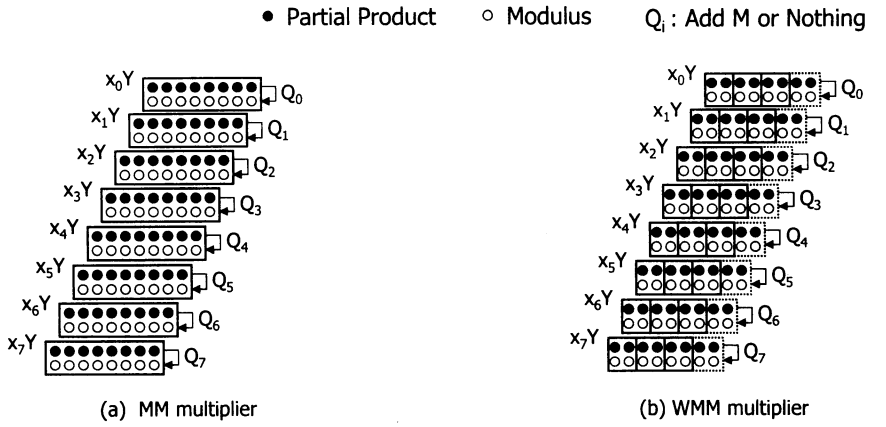

図 2　MM 乗算器 と WMM 乗算器の処理の流れ

```
Algorithm PAMM(X, Y, M)
    PP = 0;    MA = 0;    C  = 0;

1:  for i=0 to n
2:    C_t = C;

3:    //Partial Product Addition (PP) Part
4:    if (i < n) then
5:      PP_t = PP + x_iY + C;
6:      if (PP_t is odd) then
7:        C = 1;
8:      else
9:        C = 0;
10:     end if
11:     PP = [PP_t / 2];
12:   end if

13:   //Modulus Accumulation (MA) part
14:   if (i > 0) then
15:     if ((C_t = 1 and MA is even) or
16:         (C_t = 0 and MA is odd)) then
17:       MA_t = MA + M;
18:     else
19:       MA_t = MA;
20:     end if
21:     MA = [MA_t / 2];
22:   end if
23: end for

24: S = PP + MA + C;

25: if (S >= M) then
26:   S = S - M;
27: end if

28: return S;
end.
```

図 3　PAMM アルゴリズム

ションの PP パートの計算時に，$PP_t^{\langle i-1\rangle}$ の最下位桁 $C^{\langle i-1\rangle}$ を補正値として加算する．

また図 4(a) に示すように，MA パートは，PP パートからイタレーション 1 回分遅れて実行される．したがって，MM アルゴリズムにおける部分

表 1　PAMM アルゴリズムの誤差

PP_t	MA	MA_t	C	Truncation error		
				PP	MA	Total
even	even	even	0	0	0	0
even	odd	even	0	0	0	0
odd	even	odd	1	-0.5	-0.5	-1
odd	odd	odd	1	-0.5	-0.5	-1

剰余 S と PP, MA, C の関係は，

$$S^{\langle i\rangle} = PP^{\langle i\rangle} + MA^{\langle i+1\rangle} + C^{\langle i\rangle} \qquad (4)$$

となる．全ての i について式 (4) を実行した後，さらに

$$S = PP^{\langle n-1\rangle} + MA^{\langle n\rangle} + C^{\langle n-1\rangle} \qquad (5)$$

を実行する必要がある．

図 4(a) にも示しているように，PAMM 乗算器の加算回数は MM 乗算器と同じである．また，実行するイタレーション回数は $n+1$ 回となる．しかし，各イタレーションにおいて PP パートと MA パートの処理を並列に実行することができる．

図 4(b) に，PAMM アルゴリズムを組み合わせた WMM (WPAMM) 乗算器での処理の流れを示す．処理の流れは，WMM 乗算器とほぼ同様である．また x_iY の最初のワードイタレーションを処理する際に，Q_i を計算する以外に，補正値 $C^{\langle i\rangle}$ の計算も行う必要がある．なお WPAMM 乗算器では，PP パートの開始からワードイタレーション 1 回分遅れて MA パートの処理を開始する．その後は 2 つのパートを並列に実行する．したがって，WPAMM 乗算器の実行するワードイタレーショ

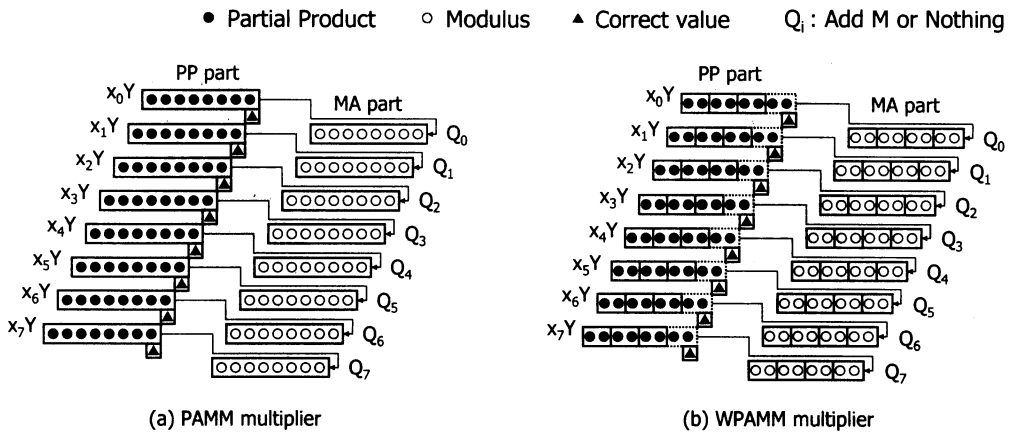

図 4　PAMM 乗算器と WPAMM 乗算器の処理の流れ

ン回数は，$(n^2/w)+1$ 回となる．

4　ハードウェア構成

PAMM アルゴリズムのハードウェア構成について検討する．一般に，高速な演算器を構成する際には，キャリー伝搬による遅延時間の増大を防ぐために，キャリーセーブ加算器 (Carry Save Adder : CSA) が用いられる．そこで本稿では，PAMM アルゴリズムを CSA を用いて構成する．

CSA では，加算結果は中間和と中間桁上げの和として出力される．ここで，$I_s(v)$，$I_c(v)$ をそれぞれ v の中間和，中間桁上げとし，$v = I_s(v) + I_c(v)$ とする．このとき，PP パートの式 (1) は，

$$PP_t = I_s(PP) + I_c(PP) + x_iY + C \qquad (6)$$

と表すことができる．式 (6) では 4 つのオペランドを加算するため，n ビット CSA を用いて計算する場合には，2 回の加算が必要となる．これは，MM アルゴリズムにおける 1 回のイタレーションで実行する加算回数と同じである．したがって，PAMM アルゴリズムをそのまま n ビット CSA を用いて構成しても，高速な MM 乗算器を構成することはできない．

そこで，補正値 C の値は高々 1 であり，$C = \frac{1}{2}C + \frac{1}{2}C$ であることを利用する．PS と PC を $(n+1)$ ビットの 2 進数とし，$PS = I_s(PP) + \frac{1}{2}C$，$PC = I_c(PP) + \frac{1}{2}C$ とする．PS と PC は，n ビットの整数部と 1 ビットの小数部からなり，それぞれ $I_s(PP)$ と $I_c(PP)$ の最下位桁に C を連結することで生成することができる．PS と PC を用いて式 (6) を

$$PP_t = PS + PC + x_iY \qquad (7)$$

に変更する．式 (7) から，PP_t は 1 段の $(n+1)$ ビットの CSA を用いて計算できることがわかる．

ここで，$PS + PC$ の小数部の値は常に 0 であるため，$I_s(PP_t)$ は n ビットの整数となる．一方，$I_c(PP_t)$ は，$PS + PC$ の小数部からの桁上げが発生するため $n+1$ ビットの整数となる．したがって，$I_s(PP_t)$ と $I_c(PP_t)$ はともに 2^0 の重みの桁を持つため，シフト操作により 2 つのビットが切り捨てられてしまう．その結果，PP_t の値が偶数であるにも関わらず，$I_s(PP_t)$ と $I_c(PP_t)$ の最下位桁がともに 1 の場合にも誤差が生じてしまう．したがって，CSA を用いる場合には，誤差が発生する条件を $I_s(PP_t)$ と $I_c(PP_t)$ の最下位桁のいずれか一方が 1 である場合に変更する必要がある．

一方，MA パートの式 (2) は，

$$MA_t = I_s(MA) + I_c(MA) + M \qquad (8)$$

と表すことができる．式 (8) は，$I_s(PP)$, $I_c(PP)$, $I_s(MA)$, $I_c(MA)$ の最下位桁の和が奇数である場合に実行される．また MA パートは，加算するオペランドが 3 つであるため，そのまま 1 段の n ビット CSA を用いて加算することができる．

CSA を用いた PAMM アルゴリズムの実行例を図 5 に示す．図 5 の例では，中央のカラムに補正値 $C^{(i)}$ の計算と法の加算信号 Q_i の計算を表す．また，CSA を用いた MM 乗算器，PAMM 乗算器の PE の構成をそれぞれ図 6(a),(b) に示す．なお図 6(a) のセレクタは，部分積の加算結果から法

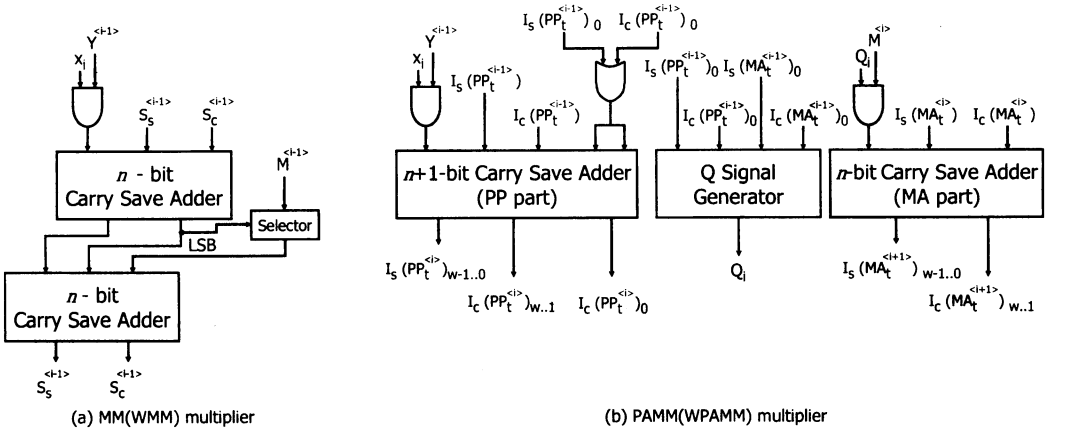

図 6　Processing Element の構成

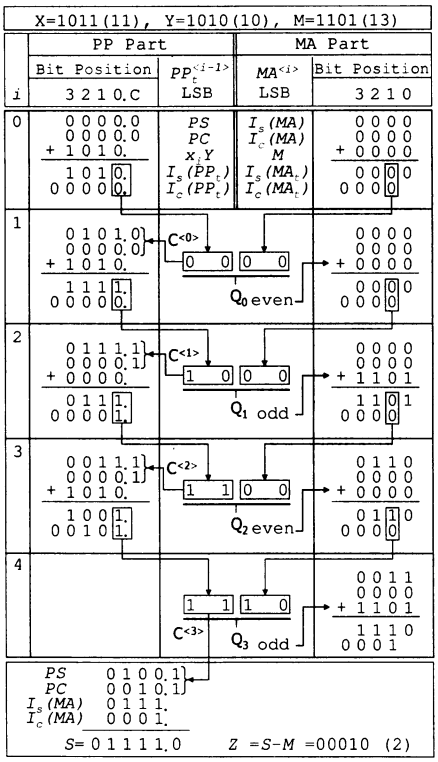

図 5　PAMM アルゴリズムの実行例

の加算を選択する．図 6(b) の Q 信号生成回路は，$I_s(PP)$, $I_c(PP)$, $I_s(MA)$, $I_c(MA)$ の最下位桁から Q_i を生成する回路であり，図 5 の中央のカラムの動作を実現する．また，図 6(b) の OR ゲートは，PAMM アルゴリズムにおける補正値 C の計算を行う．図 6 から，MM 乗算器の PE は 2 段の CSA を用いて構成しているのに対し，PAMM 乗算器では PP パートと MA パートはそれぞれ 1 段の CSA を用いて構成できることがわかる．

ここで，WMM 乗算器，WPAMM 乗算器の構成についても検討する．図 7 に一般的なワードベース構成の例を示す．なお図 7 では，オペランド V の $j(=0,1,\cdots,(n/w)-1)$ 番目のワードを V^j，PE の数を p と表している．また，網掛けの四角はレジスタを表している．WMM 乗算器と WPAMM 乗算器は，それぞれ PE の構成と制御回路が異なる．またワードベース構成にするにあたり，MM 乗算器では PE 内のセレクタ，PAMM 乗算器では PE 内の Q 信号生成回路がそれぞれ若干複雑になる．しかし PAMM 乗算器の場合，Q_i の生成は PP パートにおける x_iY の最初のワードイタレーションの処理と同時に開始できるため，遅延時間は増加しない．

5　性能評価

ここでは，PAMM 乗算器と WPAMM 乗算器の性能評価を行う．評価は，MM 乗算器，WMM 乗算器，PAMM 乗算器，WPAMM 乗算器のそれぞれを VHDL を用いて記述し，各 MM 乗算器の PE の遅延時間と回路規模を比較する．なお，WMM 乗算器と WPAMM 乗算器の PE の個数は，それぞれ 1 個とする．論理合成には，Synopsys 社の Design Compiler Ver.2003.06 を使用した．各 MM 乗算器の PE のビット幅を 16 ビットから 1024 ビットまで変化させたときの PE の遅延時間を表 2 に，回路規模を表 3 に示す．なお表 2 と表 3 の値は，それぞれ 16 ビットの MM 乗算器の値を 1 とした場合の相対値を表している．また，

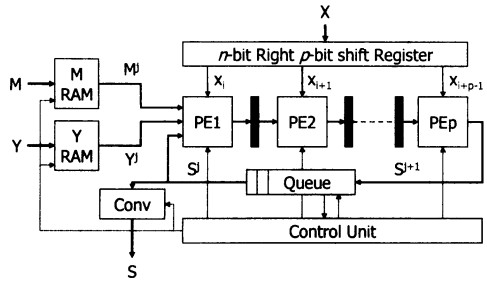

図 7　一般的なワードベース構成の例

MM 乗算器と PAMM 乗算器の PE のビット幅は n の値, WMM 乗算器と WPAMM 乗算器の PE のビット幅は w の値を表している. なお, WMM 乗算器と WPAMM 乗算器では, PE のビット幅が 16 ビットでも PE の処理回数を増やすことにより 1024 ビットのオペランドを処理することができる.

表 2 からわかるように, PAMM 乗算器は, MM 乗算器よりも遅延時間を削減することができ, 平均約 30%の性能向上を達成している. 同様に WPAMM 乗算器は, WMM 乗算器に比べ平均約 63%の遅延時間を削減できている. また表 3 から, PAMM 乗算器では MM 乗算器に比べて約 20%の回路規模の増加が見られるものの, WPAMM 乗算器では WMM 乗算器と ほぼ同じであることがわかる. なお, オペランドのビット長に伴って遅延時間が増えているのは, 内部信号のファンアウトが増大したためだと考えられる.

3 において, PAMM 乗算器と WPAMM 乗算器では, 実行するイタレーションの回数が増えることを述べた. しかし評価結果からわかるように, それぞれの乗算器は各イタレーションの処理時間を短縮することができるため, 結果として全体の処理時間を短縮することができる. 以上のことから, わずかな回路面積のオーバーヘッドで MM 乗算器と WMM 乗算器の高速化を達成できていると考えられる.

6　まとめ

今回我々は, MM アルゴリズムにおける 2 つの加算を並列に実行する PAMM アルゴリズムを提案した. また, PAMM アルゴリズムを適用した高速な WMM 乗算器の構成を示した.

今後の課題としては, 制御回路における遅延時間の短縮などが挙げられる.

表 2　各 MM 乗算器の遅延時間

	PE のビット幅 (bit)				
	16	32	128	512	1024
MM	1.00	1.34	4.37	15.34	29.25
WMM	1.52	2.15	7.54	27.20	52.23
PAMM	0.70	0.94	3.08	10.80	20.69
WPAMM	0.56	0.79	2.80	10.07	19.33

表 3　各 MM 乗算器の回路規模

	PE のビット幅 (bit)				
	16	32	128	512	1024
MM	1.00	2.00	8.00	32.00	64.00
WMM	1.01	2.01	8.01	32.01	64.01
PAMM	1.22	2.35	9.11	36.16	72.24
WPAMM	1.05	2.05	8.05	32.05	64.05

参考文献

[1] P.L.Montgomery, "Modular Multiplication without Trial Division,"Math. of Computation, vol. 44, no. 170, pp. 519-521, Apr. 1985.

[2] Marcelo E. Kaihara and Naofumi Takagi, "A VLSI Algorithm for Modular Multiplication/Division," Proceedings of the 16th IEEE Symposium on Computer Arithmetic (ARITH'03), pp. 220-227, 2003.

[3] Nadia Nedjah and Luiza de Macedo Mourelle, "Reconfigurable Hardware Implementation of Montgomery Modular Multiplication and Parallel Binary Exponentiation," Proceedings of the Euromicro Symposium on Digital System Design (DSD'02), pp. 226-235, 2002.

[4] Chih-Yuang Su, Shih-Arn Hwang, Po-Song Chen, and Cheng-Wen Wu, "An Improved Montgomery's Algorithm for High-Speed RSA Public-Key Cryptosystem," IEEE Trans, Very Large Scale Integration (VLSI) Systems, vol. 7, no. 2, pp. 280-284, 1999.

[5] Alexandre F.Tenca and Çetin K. Koç, "A Scalable Architecture for Modular Multiplication Based on Montgomry's Algorithm," IEEE Trans, Computers, vol. 52, no. 9, pp. 1215-1221, Sep. 2003.

[6] Nadia Nedjah adn Luiza de Macedo Mourelle, "Two Hardwaret Implementations for the Montgomery Modular Multiplication : Sequential versus Parallel," Proceedings of the 15th Symposium on Integrated Circuits and Systems Design (SBCCI'02), 2002.

[7] Seok-Yong Lee, Yong-Jin Jeong and Óh-Jun Kwon, "A Faster Modular Multiplication Based on Key Size Partitioning for RSA Public-Key Cryptosystem," IEICE Trans. Vol.E85-D, no. 4, pp. 789–791, Apr, 2002.