2004－ＣＳＥＣ－26 （3）
2004／7／20

RSA暗号処理における高基数剰余乗算回路

葛 毅† 櫻井 隆雄† 阿部 公輝†† 坂井 修一†

† 東京大学大学院情報理工学系研究科電子情報学専攻 〒113-8656 東京都文京区本郷 7-3-1
†† 電気通信大学情報工学科 〒182-8585 東京都調布市調布ヶ丘 1-5-1
E-mail: †{katsu-t,sakurai,sakai}@mtl.t.u-tokyo.ac.jp, ††abe@cs.uec.ac.jp

あらまし RSA 暗号の暗号化/復号化処理で用いられる剰余乗算回路の有効な手法の一つである，除算に基づく回路構成法を高基数化して一般化した．そして高基数化による面積/速度のトレードオフを評価した．数表現には冗長 2 進表現を用い，法の倍数の選択には高基数 SRT 除算で一般的に用いられるテーブルではなく，同じ構造を維持して高基数化するのに適した算術演算による構成法を用いている．評価の結果，基数 16 では基数 4 に比べて約 1.6 倍の面積コストで，約 1.5 倍の速度向上が得られた．

キーワード RSA 暗号，剰余乗算，SRT 除算，高基数，冗長 2 進表現

A Hardware Organization of High-Radix Modular Multiplication for RSA Cryptosystem

Yi GE†, Takao SAKURAI†, Koki ABE††, and Shuichi SAKAI†

† Department of Information and Communication Engineering, Graduate School of Information Science and Technology, The University of Tokyo, 7-3-1 Hongo Bunkyo-ku, Tokyo 113-8656, JAPAN
†† Department of Computer Science, The University of Electro-Communications, 1-5-1 Chofugaoka Chofu-shi,Tokyo 182-8585 Japan
E-mail: †{katsu-t,sakurai,sakai}@mtl.t.u-tokyo.ac.jp, ††abe@cs.uec.ac.jp

Abstract Hardware organized modular multiplication based on division algorithm is one of the effective methods used for RSA encryption/decryption. This paper generalizes the hardware organization of the modular multiplication based on the higher-radix SRT division algorithm, and describes the area/time trade-off of the organization. For the number representation we used the signed-digit number system and for selecting the multiple of modular we employed the arithmetic operation instead of the conventional look-up table. The method based on the arithmetic operation is suitable for keeping the same structure over wide range of high radices. The result of the evaluation revealed that a radix-16 multiplier produced 1.5 times speedup at about 1.6 times area cost compared with radix-4 multiplier.

Key words RSA cryptosystem, modular multiplication, SRT division, high-radix, signed-digit number

1. はじめに

RSA 暗号 [1] を始めとする公開鍵暗号の暗号化/復号化処理はべき剰余演算で行われる．このべき剰余演算は剰余乗算に分解できる．演算のビット数が非常に大きく，処理時間が長いことから，高速化のために多くの回路構成法が研究されている．剰余乗算アルゴリズム [2] の中で，回路化に有効なものとして，除算に基づく方法 [3] と，モンゴメリ乗算に基づく方法 [4] がある．後者は幾つかの前処理と後処理が必要になるが，基本的な基数 2 の構成同士の比較では前者に比べて 2 倍程度高速であるとされる [5].

本論文では，除算に基づく構成法を検討する．この構成法は，文献 [3] から始まり，文献 [5]～[7] で検証され，文献 [8]～[13] で改良されてきた．除算に基づく剰余乗算回路の高速化には次の手法がある．

（1） 引き込む法の倍数選択をフルビットの比較ではなく，上位数ビットの比較で行う．また同時に，数表現に冗長表現を用いて加算を桁上げなしで行う．これは SRT 除算 [14] に対応する．

（2） 法を大きくし，部分積の影響を小さくする．また，法

を基数倍する場合，演算結果が法の基数倍近くの値をとり得るが，繰り返しを一回多く行い演算結果を右シフトすることで，法より小さな値に収める．

（3） 高基数化により繰り返し回数を減らす．

（4） オペランドにも冗長表現を用い，べき剰余演算全体を冗長表現で行う．このため剰余乗算毎に結果を通常2進数に変換する必要がなくなる．この変換には桁上げ伝播加算が必要になり，演算するビット長が大きい場合，回路化すると比較的大きくなる．

（5） 部分積と法の倍数を正負対称にして，生成を容易にする．

各文献の構成法を検討すると，これらの高速化手法を皆含んでいる文献[13]の構成法が有効であると考えられ，そこでは基数4の場合について述べられている．そこで，本論文では，この構成法を高基数化して一般化した．そして，除算に基づく剰余乗算回路の高基数化による面積/速度のトレードオフを評価した．定性的には文献[7]で議論されている．法の倍数選択には，高基数SRT除算で一般的に用いられるテーブルによる構成法[15]ではなく，同じ構造を維持して高基数化するのに適した，算術演算による構成法を用いている[16]．

以下，2章で剰余乗算のアルゴリズムについて述べ，3章で高基数剰余乗算回路について述べる．4章で評価結果を示し，5章でまとめる．

2. 剰余乗算のアルゴリズム

文献[13]の剰余乗算のアルゴリズムを一般化する．

基数をr，rは2のべき乗，演算ビット数をn，法を$N, 2^{n-1} \leqq N < 2^n$，被乗数，乗数，積を各々x，y，RR，$-\rho_1 N \leqq x, y, RR \leqq \rho_1 N$，中間積を$R_j$，$-\rho_2 N \leqq R_j \leqq \rho_2 N$，$r\rho_1 = \rho_2$，$1/2 < \rho_1 \leqq 1$，$\hat{y}_j \in \{-r/2, ..., -1, 0, 1, ..., r/2\}$，$q_j \in \{-q_{max}, ..., -1, 0, 1, ..., q_{max}\}$とする．剰余乗算$RR = (xy) \bmod N$は次式を$j = \lfloor n/\log_2 r \rfloor$から$j = -1$まで繰り返すことで行う．

$$R_j = rR_{j+1} + \hat{y}_j x - rq_j N \qquad (1)$$

q_jは$-\rho_2 N \leqq R_j \leqq \rho_2 N$を満たすように決定する．積は$RR = R_{-1}/r$である．$\hat{y}_{-1} = 0$よりこの$r$による除算はシフトのみで行える．また，式(1)は法Nをr倍することで部分積$\hat{y}_j x$の影響を小さくして，q_{max}を小さくする手法を適用している．

次にρ_2，q_{max}の条件を決定する．まず，$rR_{j+1} + \hat{y}_j x$が最小値，最大値のとき収束する条件から次式が得られる．

$r\rho_2 N + r/2 \cdot \rho_1 N - rq_{max} N \geqq \rho_2 N \Leftrightarrow$

$$\rho_2 \leqq \frac{2rq_{max}}{2r-1} \qquad (2)$$

次に，あるq_jを選択できる領域を考えると，$-\rho_2 N \leqq rR_{j+1} + \hat{y}_j x - rq_j N \leqq \rho_2 N \Leftrightarrow (rq_j - \rho_2)N \leqq rR_{j+1} + \hat{y}_j x \leqq (rq_j + \rho_2)N$となる．よって，隣り合う$q, q-1$を選択できる領域が連続する条件から次式が得られる．$(rq - \rho_2)N < (rq - r + \rho_2)N \Leftrightarrow$

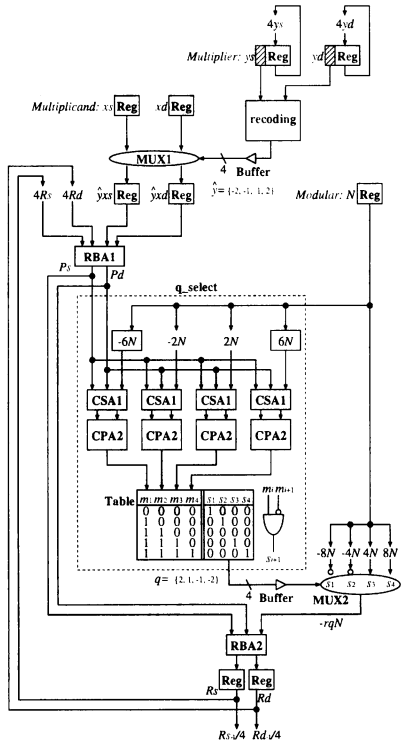

図1 基数4の回路構成

$$\rho_2 > \frac{r}{2} \qquad (3)$$

式(3)より，q_jの選択に冗長性が生じ，$rR_{j+1} + \hat{y}_j x$とNの上位数桁からq_jを決定できる．

q_{max}の最小値を決定する．式(2)，(3)より，$q_{max} > (2r-1)/4$となり，q_{max}は整数であることから次式が得られる．

$$q_{max} = \lfloor \frac{2r-1}{4} \rfloor + 1$$

3. 高基数剰余乗算回路

除算に基づく高基数剰余乗算回路について述べる．

3.1 回路構成

図1に基数4の回路構成を示す．

(Rs, Rd)，(xs, xd)，(ys, yd)を各々R，x，yの冗長2進表現とする．また，$P = rR + \hat{y}x$とし，(Ps, Pd)をPの冗長2進表現とする．

最初のサイクルで，部分積生成回路 **recoding** とマルチプレクサ **MUX1** で$\hat{y}_j \in \{-2, -1, 0, 1, 2\}$として，冗長2進表現の部分積$\hat{y}_j x$を生成する．**MUX1** はデコードされた信号で制御する．部分積$\hat{y}_j x$を$(\hat{y}xs, \hat{y}xd)$として，レジスタ **Reg** に保持する．

次サイクルで，冗長2進加算器(RBA) **RBA1** で冗長2進表現された中間積の基数倍$(4Rs, 4Rd)$と$(\hat{y}xs, \hat{y}xd)$の加算$(4Rs, 4Rd) + (\hat{y}xs, \hat{y}xd) = (Ps, Pd)$を行う．法の倍数選択回路 **q_select** で上の加算結果PとNの上位数桁から法Nの

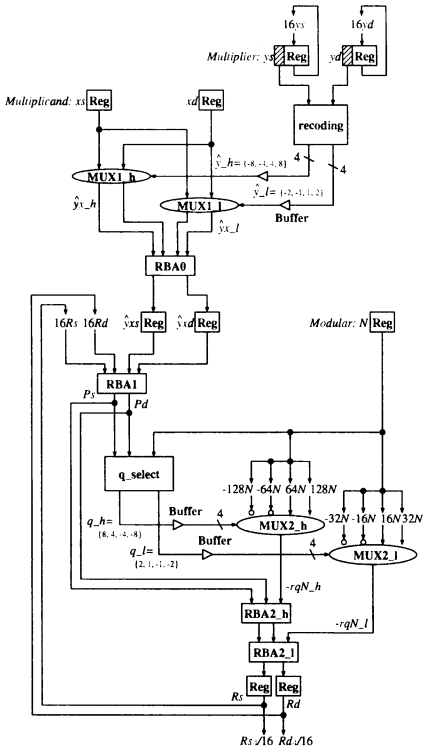

図 2　基数 16 の回路構成

倍数 q_j を決定し，**MUX2** で $-rq_jN$ を選択する．**RBA2** で $(Ps, Pd) + (-rq_jN) = (Rs, Rd)$ を行い，**Reg** に保持する．$j = -1$ における (Rs, Rd) を 2 桁右シフトしたものが，積 RR である．部分積 $\hat{y}_j x$ を 1 サイクル前で生成することで，部分積 $\hat{y}_j x$ の生成と，q の選択および法の倍数 $-rqN$ の加算とを並列実行している．

図 2 は基数 16 の回路構成である．$\hat{y} \in \{-8, ..., -1, 0, 1, ..., 8\}$ として，これを，$\hat{y}_h \in \{-8, -4, 0, 4, 8\}$，$\hat{y}_l \in \{-2, -1, 0, 1, 2\}$，$\hat{y} = \hat{y}_h + \hat{y}_l$ に分ける．各々に対応する部分積 $\hat{y}x_h, \hat{y}x_l$ を **RBA0** で加算する．同様に，$q \in \{-8, ..., -1, 0, 1, ..., 8\}$ として，これを，$q_h \in \{-8, -4, 0, 4, 8\}$，$q_l \in \{-2, -1, 0, 1, 2\}$，$q = q_h + q_l$ に分ける．各々に対応する法の倍数 $-rqN_h, -rqN_l$ を **RBA2_h**，**RBA2_l** で P と加算する．

更に基数を上げる場合は，各 MUX，RBA が多重化される．

以下，各モジュールの詳細を述べる．

3.2　部分積生成回路 (recoding)

部分積生成回路 **recoding** では，次の手順で $\hat{y}_j$ を $\{-r/2, ..., -1, 0, 1, ..., r/2\}$ に符号化する．最下位桁を 0 桁目とする．(1) (ys, yd) の $(\log_2 r)j + (\log_2 r - 1)$ ビット目から $(\log_2 r)j$ ビット目を $(\hat{y}s'_j, \hat{y}d'_j)$ として，表 1 を引く．(2) $\hat{y}_j = c_j + s_j$ とする．

3.3　冗長 2 進加算器 (RBA)

冗長 2 進数の加算について述べる [17]．ある冗長 2 進数 $a = (as, ad)$ のビット割り当てを，$(1, 1) = -1$，$(0, 0) = 0$，$(0, 1) = 1$ とする．

表 1　**recoding** の変換表

$(\hat{y}s'_j, \hat{y}d'_j)$	c_{j+1}	s_j
$-r+1$	-1	1
$-r$	-1	2
⋮	⋮	⋮
$-\frac{r}{2}-1$	-1	$\frac{r}{2}-1$
$-\frac{r}{2}$	$^*0/-1$	$^*-\frac{r}{2}/\frac{r}{2}$
$-\frac{r}{2}+1$	0	$-\frac{r}{2}+1$
⋮	⋮	⋮
-1	0	-1
0	0	0
1	0	1
⋮	⋮	⋮
$\frac{r}{2}-1$	0	$\frac{r}{2}-1$
$\frac{r}{2}$	$^*1/0$	$^*-\frac{r}{2}/\frac{r}{2}$
$\frac{r}{2}+1$	1	$-\frac{r}{2}+1$
⋮	⋮	⋮
r	1	-2
$r-1$	1	-1

$^*y_{(\log_2 r)j-1}$ が非負 / その他

表 2　**RBA1** の変換表

(a)

$(rR)_{n'-1}$	$(rR)_{n'-2}$	$P_{n'-1}$	$s1$
-1	-1	$^*-1/\times$	$^*-1/\times$
-1	0	-1	0
-1	1	$^*0/-1$	$^*-1/1$
0	-1	$^*0/-1$	$^*-1/0$
0	0	0	0
0	1	$^*1/0$	$^*-1/1$
1	-1	$^*1/0$	$^*-1/1$
1	0	1	0
1	1	$^*\times/1$	$^*\times/1$

$^*(rR)_{n'-3}$ が非負 / その他

(b)

$p_{n'-3}$	$v_{n'-3}$	$s1$	$P_{n'-2}$
0	1	0	1
0	1	-1	0
0	0	1	1
0	0	0	0
0	0	-1	-1
1	1	1	1
1	1	0	0
1	1	-1	-1
1	0	1	0
1	0	0	-1

3.3.1　冗長 2 進数同士の加算 (RBA0, RBA1)

二つの冗長 2 進数 a, b の加算 $(as, ad) + (bs, bd) = (zs, zd)$ は，次の論理式により行う．ここでの添え字 i は桁の位置を表す．まず，次の変数を用意する．$p_i = as_i \vee bs_i$，$sd_i = ad_i \oplus bd_i$，$u_i = sd_i \oplus p_{i-1}$，$v_i = \neg(sd_i \wedge p_{i-1}) \wedge \neg(as_i \wedge bs_i) \wedge (ad_i \vee bd_i)$．

表 3 **RBA2** の変換表

pv	$R'_{n''-1}$	$R'_{n''-2}$	$R_{n''-1}$	$R_{n''-2}$
-1	1	−	-1	$R'_{n''-2}$
-1	0	1	-1	-1
0	−	−	$R'_{n''-1}$	$R'_{n''-2}$
1	0	-1	1	1
1	-1	−	1	$R'_{n''-2}$

それから，次式より (zs, zd) を得る．$zs_i = u_i \wedge \neg v_{i-1}$, $zd_i = u_i \oplus v_{i-1}$.

RBA0 では，$\hat{y}x_h$ と $\hat{y}x_l$ を上の論理式の a, b として加算 $\hat{y}x_h + \hat{y}x_l = \hat{y}x$ を行う．

RBA1 では，rR と $\hat{y}x$ を上の論理式の a, b として加算 $rR+\hat{y}x = P$ を行う．また，(Ps, Pd) の桁数を $n' = n+2(\log_2 r)$ に抑えるために，上位 2 桁を表 2(a)，(b) のようにする．R が取りうる数値の範囲から，表 2 の × の場合は起こらない．

3.3.2 冗長 2 進数と通常 2 進数の加算 (RBA2)

冗長 2 進数 a と通常 2 進数 b の加算 $(as, ad) + b = (zs, zd)$ は，次の論理式により行う．まず，次の変数を用意する．$u_i = sd_i = ad_i \oplus b_i$, $v_i = \neg as_i \wedge (ad_i \vee b_i)$．それから，次式より (zs, zd) を得る．$zs_i = u_i \wedge \neg v_{i-1}$, $zd_i = u_i \oplus v_{i-1}$. b が 2 の補数の場合は，b の最上位ビットに負の重みを与えるため，最上位桁の加算は，次の論理式で行う．$u_i = ad_i \oplus b_i$, $zs_i = u_i \wedge \neg v_{i-1}$, $zd_i = u_i \oplus v_{i-1}$, $zs_{i+1} = as_i \wedge b_i$, $zd_{i+1} = (as_i \wedge b_i) \vee (\neg as_i \wedge ad_i \wedge \neg b_i)$.

RBA2_h では，P と $-rqN_h$ を上の論理式で b が 2 の補数の場合の a, b として加算 $P + (-rqN_h)$ を行う．

RBA2(_l) では，P（または **RBA2_h** の出力）と $-rqN(\_l)$ を上の論理式の a, b として加算 $P + (-rqN) = R'$ を行う．また，R の桁数を $n'' = n + (\log_2 r)$ に抑えるために，P と $-rqN(\_l)$ の n'' 桁目以上を各々 P', $(-rqN)'$ として，次のようにする．(1) $pv = P' + (-rqN)' + u_{n''-1}$ を計算する．必ず，$pv \in \{-1, 0, 1\}$ となる．(2) 表 3 により n'' 桁にする．$n''-3$ 桁目以下は，$R = R'$ とする．

3.4 法の倍数選択回路 (q_select)

法の倍数選択回路 **q_select** では，SRT 除算と同様にして P と N の上位数桁から q を決定する．ここでは，SRT 除算で一般的なテーブルを用いる方法ではなく，同じ構造を維持して高基数化するのに適した算術演算による構成法を用いている [16].

図 3 に **q_select** の回路構成の一部を示す．P と N の上位数桁から $P-kN$ の誤差を含んだ値 $\widehat{P-kN}$ の符号ビットを計算し，この符号ビットから図 1 の **Table** を引き，q_j を選択する．k は，後で述べる重複領域の境界線である．q_j をデコードした値にすれば，**Table** は AND ゲート 1 段である．(a) は **CSA1** に 3-2CSA を用いた場合，(b) は 4-2CSA を用いた場合である．モジュール **kN** で $-kN$ の上位数ビット $\widehat{-kN}$ を計算し，**CSA1, CPA2** で (Ps, Pd) の上位数桁 $(\widehat{Ps}, \widehat{Pd})$ との加算を行う．$-kN$ は最初のサイクルで予め計算しておく．**CSA1** では入力を $(\widehat{Ps}, \widehat{Pd})$ の正の桁を集めたものと，負の桁を集めてビット反転させたものに変換している．**CSA1** はクリティカルパスであるため，(a) の方が少し速くなるが，$\widehat{-kN}$ の生成の回路量が増える．

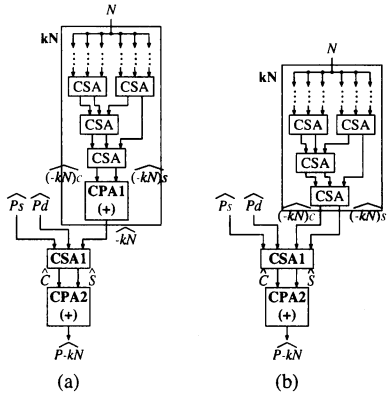

図 3 **q_select** の回路構成の一部

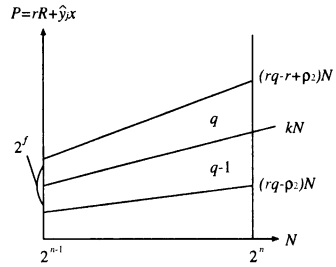

図 4 重複領域における P-D プロット

3.4.1 CPA2 のビット数

q を選択するための **CPA2** のビット数を計算する．図 4 に対応する P-D プロットを示す．

まず，$P - kN$ の許容誤差 Δ を計算する．$\widehat{P-kN} = (P - kN) - \Delta$ とする．また，隣り合う q, $q-1$ を選択できる重複領域の境界線を kN とする．許容誤差 Δ の上端は，P が $N = 2^{n-1}$ における重複領域の上端の値のときに正となるように決まるから，$(rq - r + \rho_2) \cdot 2^{n-1} - k \cdot 2^{n-1} - \Delta \geqq 0 \Leftrightarrow \Delta \leqq (rq - r + \rho_2 - k) \cdot 2^{n-1}$ である．同様に下端は，$(rq - \rho_2) \cdot 2^{n-1} - k \cdot 2^{n-1} - \Delta \leqq 0 \Leftrightarrow (rq - \rho_2 - k) \cdot 2^{n-1} \leqq \Delta$ である．以上より許容誤差は，$(rq - \rho_2 - k) \cdot 2^{n-1} \leqq \Delta \leqq (rq - r + \rho_2 - k) \cdot 2^{n-1}$ である．

次に Δ に収まる **CPA2** のビット数を計算する．S, C を $P-kN$ の桁上げ保存表現であるとする．$S + C = P - kN$ である．ここで，$N = 2^{n-1}$ における重複領域の幅 $(rq - r + \rho_2) \cdot 2^{n-1} - (rq - \rho_2) \cdot 2^{n-1} = (2\rho_2 - r) \cdot 2^{n-1}$ から，$2^f \leqq (2\rho_2 - r) \cdot 2^{n-1}$ を満たす最小の整数

$$f = \lfloor \log_2((2\rho_2 - r) \cdot 2^{n-1}) \rfloor$$

を導入する．

S, C の誤差を各々 $\Delta S, \Delta C$, $0 \leqq \Delta S, \Delta C < 2^c$ とする．$S = \widehat{S} + \Delta S$, $C = \widehat{C} + \Delta C$ である．加算器 **CPA2** で $\widehat{S} + \widehat{C} + 2^c$ を行うと，$(S + C) - (\Delta S + \Delta C - 2^c)$ より誤差の範囲は，$-2^c \leqq \Delta S + \Delta C - 2^c < 2^c$ である．これが，Δ の範囲に

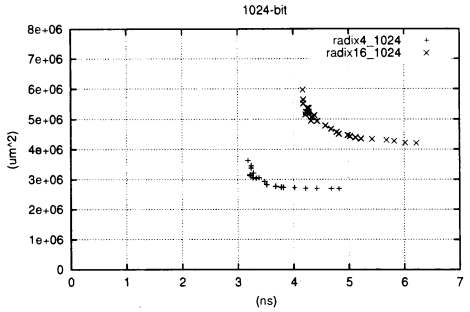

図 5　1024 ビットの合成結果

含まれていればよいことから，$(rq-\rho_2-k)\cdot 2^{n-1} \leqq -2^c$，$(rq-r+\rho_2-k)\cdot 2^{n-1} \geqq 2^c$ が得られる．ここで k を重複領域の中心線

$$k = r(q'-1/2),\ q' \in \{-q_{max}+1, ..., q_{max}\}$$

とすると，$c \leqq f-1$ が得られる．よって，最下位ビットを 0 ビット目として，$f-1$ ビット目以上で計算すればよい．

この条件においては，**CPA1** は **CPA2** と同様 $f-1$ ビット目以上で計算すればよい．その他の CSA は，それより後の CPA，CSA より 1 ビット大きければ誤差は生じない．

次に **CPA2** の上端のビット位置を決定する．これは，$-(2rq_{max}+\rho_2-r/2)\cdot 2^n < (rR+\hat{y}_i x)-kN < (2rq_{max}+\rho_2-r/2)\cdot 2^n$ より，$\lfloor \log_2\{(2rq_{max}+\rho_2-r/2)\cdot 2^{n+1}\} \rfloor$ である．

4. 評　　価

基数 4 と基数 16 の剰余乗算回路を Verilog-HDL で設計し，論理合成ツール DesignCompiler で合成して得られる速度と面積の数値を用いて比較した．ゲート間の駆動遅延は含む．配線遅延は含まない．合成には日立社の製造条件に基づき京都大学で作成された CMOS 0.18μm セルライブラリを用いた．このライブラリはおよそ 300 個のセルがある．

合成ツールの遅延時間制約を変化させて繰り返し自動合成し性質を調べた．図 5 に 1024 ビットの合成結果を示す．他のビット数でもほぼ同様の傾向である．図 5 より基数 16 では基数 4 に対し面積が約 1.6 倍，速度が約 1.5 倍となっている．表 4 に代表的な合成結果を示す．演算ビット数が大きいため，回路量がビット数に依存しない **q_select**，**recoding** の影響は小さい．回路規模は演算ビット数に比例している．また表より，1024 ビットのべき剰余演算を，最長の場合で，基数 4 では約 3.4ms，基数 16 では約 2.3ms で行える．

表 5 に 1024 ビットのクリティカルパスの遅延時間を示す．自動合成のため数値は多少前後している．表より，**q_select** と **RBA2** の遅延が比較的大きいことが分かる．**RBA2** では，上位数桁に複雑な処理をしているため遅延が大きくなっている．

5. おわりに

本論文では，除算に基づく剰余乗算回路の高基数化による面積/速度のトレードオフを検討するため，現在知られている高速化手法を全て含んだ構成法を高基数化して一般化した．そして，この構成法に基づく基数 4 と基数 16 の回路を設計し，比較した．比較の結果，高基数化により，数割程度の回路量の増加で，比較的大きな速度向上が得られた．

表 4　代表的な合成結果

n	r	number of cycles	cycle time (ns)	total time (μs)	total area (μm^2)
256	4	131	3.22	0.422	849784
	16	67	4.22	0.283	1388766
512	4	259	3.23	0.837	1636070
	16	131	4.26	0.558	2634339
1024	4	515	3.28	1.689	3219271
	16	259	4.28	1.109	5182771

表 5　クリティカルパスの遅延時間

radix-4		radix-16	
module	time (ns)	module	time (ns)
Reg	0.41	**Reg**	0.40
RBA1	0.65	**RBA1**	0.75
q_select	0.98	**q_select**	1.28
MUX2	0.34	**MUX2_h**	0.33
RBA2	0.90	**RBA2_h**	0.45
total	3.28	**RBA2_l**	1.07
		total	4.28

謝辞　本研究は，21 世紀 COE 情報科学技術戦略コア，および，科学技術振興事業団戦略的創造研究推進事業ディペンダブル情報処理基盤の支援により行われた．また，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しシノプシス株式会社の協力で行われたものである．

文　　献

[1] R. L. Rivest, A. Shamir and L. Adleman: "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communication of the ACM, **21**, 2, pp. 120–126 (1978).

[2] 高木直史：“初等関数計算回路のアルゴリズム”，情報処理，**37**, 4, pp. 362–368 (1996).

[3] E. F. Brickell: "A fast modular multiplication algorithm with application to two key cryptography", Advances in Cryptology - CRYPTO '82, Chaum et al., Eds., New York, Plenum, pp. 51–60 (1983).

[4] P. L. Montgomery: "Modular Multiplication Without Trial Division", Mathematics of computation, **44**, 170, pp. 519–521 (1985).

[5] S. E. Eldridge and C. D. Walter: "Hardware Implementation of Montgomery's Modular Multiplication Algorithm", IEEE Trans. Computers, **42**, 6, pp. 693–699 (1993).

[6] C. D. Walter and S. E. Eldridge: "A verification of Brickell's fast modular multiplication algorithm", Intern. J. Comput. Math., **33**, pp. 153–169 (1990).

[7] C. D. Walter: "Space/Time Trade-Offs for Higher Radix Modular Multiplication Using Repeated Addition", IEEE Trans. Computers, **46**, 2, pp. 139–141 (1997).

[8] 亀山充隆，魏書剛，樋口龍雄：“Signed-Digit 数多値演算回路に基づく RSA 暗号処理プロセッサの構成”，信学論，**J71-D**, 12, pp. 2659–2668 (1988).

[9] H. Morita: "A Fast Modular-multiplication Algorithm based on a Higher Radix", Lecture Notes on Computer Science (Advances in Cryptology – CRYPTO'89), Vol. 435, New York: Springer-Verlag, pp. 387–399 (1990).

[10] C. D. Walter: "Faster Modular Multiplication by Operand Scaling", Lecture Notes on Computer Science (Advances in Cryptology – CRYPTO'91). Vol. 576, New York: Springer-Verlag, pp. 313–323 (1992).

[11] H. Orup and P. Kornerup: "A high-radix hardware algorithm for calculating the exponential M^E modulo N", Proc. IEEE 10th Symp. Comput. Arithmetic, pp. 51–56 (1991).

[12] N. Takagi and S. Yajima: " Modular Multiplication Hardware Algorithms with a Redundant Representation and Their Application to RSA Cryptosystem", IEEE Trans. Computers, **41**, 8, pp. 949–956 (1992).

[13] N. Takagi: "A Radix-4 Modular Multiplication Hardware Algorithm for Modular Exponentiation", IEEE Trans. Computers, **41**, 8, pp. 949–956 (1992).

[14] J. E. Robertson: "A New Class of Digital Division Methods", IRE Trans. Electronic Computers, **EC-7**, 9, pp. 218–222 (1958).

[15] N. Burgess and T. Williams: "Choices of Operand Truncation in the SRT Division Algorithm", IEEE Trans. Computers, **44**, 7, pp. 933–938 (1995).

[16] 葛 毅, 阿部公輝, 浜田穂積 : "高基数 SRT 除算の論理回路実現に基づく回路構成と評価", 情報処理学会論文誌, **43**, 8, pp. 2665–2673 (2002).

[17] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi and N. Takagi: "Design Of High Speed MOS Multiplier And Divider Using Redundant Binary Representation", Proc. IEEE 8th Symp. Comput. Arithmetic, pp. 80–86 (1987).