算術演算回路と暗号処理ハードウエアへの応用

葛 毅, 櫻井隆雄, 坂井修一, 田中英彦

1 はじめに

高速な算術演算回路に基づく暗号処理ハードウエアに関する研究を行う. 2 章で, 算術演算回路の中の除算回路に関するこれまで研究と結果, 3 章で, 2 章の成果を土台とした, 現在の研究テーマである暗号処理の中の公開鍵暗号で用いられる乗算剰余演算のアルゴリズムと回路化に関する研究を紹介する.

2 算術演算回路

プロセッサの中心に位置する算術演算回路は高速である必要があり, 様々なアルゴリズムと回路構成法が研究されている. 基本的には, 加算, 乗算, 除算の四則演算である. 他に, 初等関数のハードウエアアルゴリズムも研究されている.

加算や乗算では, 基本的な回路構成法である順次桁上げ伝搬加算や配列型乗算法に対して, 高速化の手法として桁上げ先見加算や Wallace-Tree 乗算法が良く知られている. 演算のビット長を N とすると, 前者の初歩的なアルゴリズムの遅延時間が $O(N)$ であるのに比べて, これらの高速化手法は $O(\log N)$ であり, また, 回路面積においても, 前者に比べてそれほど大きくならない.

除算はこれらに比べて複雑であり, 選択肢の余地が多い. 筆算の場合を考えると, 各商の桁を選択するのに, 部分剰余と除数を比較する必要があり, この部分剰余が一つ前の商の桁の選択に依存する. このため, 並列度が低く, 回路面積を抑えて高速化するのが難しい. 次節以降で除算回路の研究の背景と概要について簡単に紹介する.

2.1 除算回路の研究の背景

除算は加算, 乗算についで重要な基本演算の一つであり, これまでプロセッサにおいては, ハードウエア実装されずにソフトウエアにより行うか, または, 低速であるが容易に実装できる初歩的なアルゴリズムを用いた回路が実装されてきた. 近年の LSI 技術では面積コストの制約は低くなっており, より高速な除算器が要求されている. 応用においては, 近年広く利用されている画像処理の座標計算や, 公開鍵暗号における剰余計算等の処理で, 除算は不可欠な演算であり, かつ, 高速であることが要求される. 公開鍵暗号では非常にビット長の大きい剰余演算が必要である.

除算の基本的なアルゴリズム [1] は, 大きく減算シフト型除算法と乗算型除算法に分けられる. 前者は筆算と同様に加減算とシフトで行う方法である. 一方, 後者は比較的高速であるが回路規模が大きい. 例えば, Newton-Raphson 法を用いた除算法では, 64 ビットでは基数 4 の SRT 除算に比べて 2.69 倍の速度向上を得るのに, 約 20 倍の面積を要する [9]. また, 剰余を同時に得ることができない. そこで, 減算シフト型除算法に着目する.

減算シフト型除算法は, 回復法, 非回復法, SRT 除算に分けられる. この中で, 回復法は筆算に用いている方法である. SRT 除算 [2] は回復法を工夫したものであり, 回復法に比べて高速化できるとされる. 減算シフト型除算法では商の桁の選択が主に複雑な部分である. 特に, 基数が 2 より大きい高基数 SRT 除算を構成するには詳細な誤差計算を必要とする. 1994 年の Pentium における除算回路のバグは, 基数 4 の SRT 除算の商の桁を選択するテーブルに誤りがあったことによる [10]. 高基数 SRT 除算では, 商の桁の選択にテーブルを用いる回路構成 [3][5] が一般的である. 本稿ではこれをテーブルモデルと呼ぶ.

基数 2 の回復法, 非回復法は古くから知られている方法である. SRT 除算は 1958 年頃に提案された [2]. このアルゴリズムは, 考案者 D. Sweeny, J. E. Robertson, K. D. Tocher の先頭の 1 文字をとり SRT と呼ばれる. 1968 年頃に高基数 SRT 除算

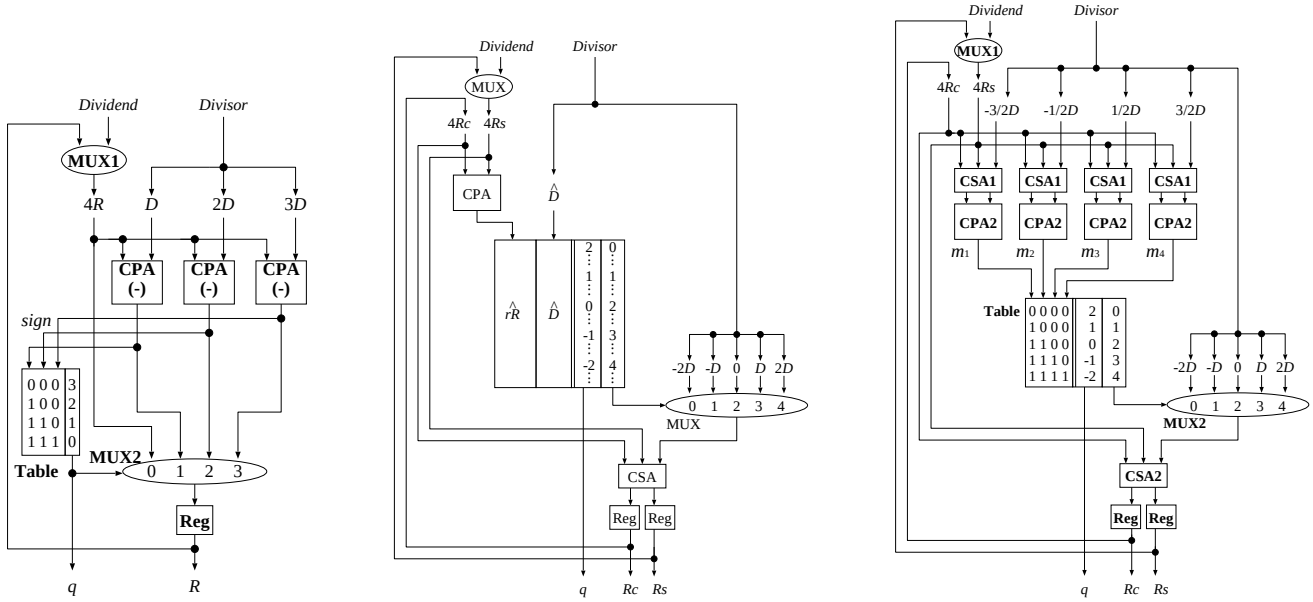

図 1: 基数 4 の回復法 (左), SRT 除算テーブルモデル (中), SRT 除算算術モデル (右)

で, 部分剰余を桁上げ保存形式で表し, 商の桁の選択にテーブルを用いて高速化するテーブルモデルが提案され, 詳細な誤差計算が行われた [3]. その後, on-the-fly 変換やスケーリング等の SRT 除算を改良する各手法が提案されてきた [4][6]. 1994 年に先のテーブルモデルの誤差計算を行った論文 [3] に比べて, より正確な誤差計算が行われた [5]. 最近は, 非常に基数の大きなものに対してスケーリングを用いる方法 [7] や除数の逆数の近似値と乗算器による除算法 [8] に関する研究がある. これらは, 54 ビットでは基数 4 の SRT 除算に比べて, 速度が約 2 から 3 倍, 面積が約 6 から 13 倍である [7][8].

2.2 これまでの研究と成果

SRT 除算の一般的な構成法は, 商の桁の選択を表を引く事で行うテーブルモデル [3][5] である. これに対して, 本研究では, 高基数 SRT 除算で商の桁を選択するのにテーブルではなく算術演算を用いる回路構成を提案し, 誤差計算を行い, 評価している [11]. 図 1 に各手法の回路構成を示す.

以下に示す各々数種類の回路を設計し, 論理合成ツールで合成して得られる速度と面積の値を用いて比較している. 設計には VDEC で製作された CMOS 0.6μm のセルライブラリを用いている. 商を選択する回路以外は, 手作業で設計している.

- r2_restore : 基数 2 の回復法.
- r4_restore : 基数 4 の回復法.
- r8_restore : 基数 8 の回復法.
- r16_restore : 基数 16 の回復法.
- r2_srt : 基数 2 の SRT 除算.
- r4q2_arith : 基数 4, $q_{max} = 2$ の算術モデル.
- r8q4_arith : 基数 8, $q_{max} = 4$ の算術モデル.
- r16q8_arith : 基数 16, $q_{max} = 8$ の算術モデル.
- r32q16_arith : 基数 32, $q_{max} = 16$ の算術モデル.
- r4q2_table : 基数 4, $q_{max} = 2$ のテーブルモデル.
- r8q7_table : 基数 8, $q_{max} = 7$ のテーブルモデル.
- r4x4q2_overlap : 基数 4, $q_{max} = 2$ のテーブルモデルを二つオーバーラップさせた基数 16[4].

図 2 は各手法の 16, 32, 54, 114 ビットでの比較結果である. 縦横の軸は面積と時間であり, 原点に近いほど性能が良い. 図より 32 ビット以下では高基数回復法, 54 ビット以上では算術モデルが有効であることが分かる. 本手法は, 回路面積を抑えた高速化に成功している.

3 暗号処理ハードウエア

近年, インターネットの広い普及により, 暗号処理が必要になってきており, 装置が高速, 小型であることが要求されている.

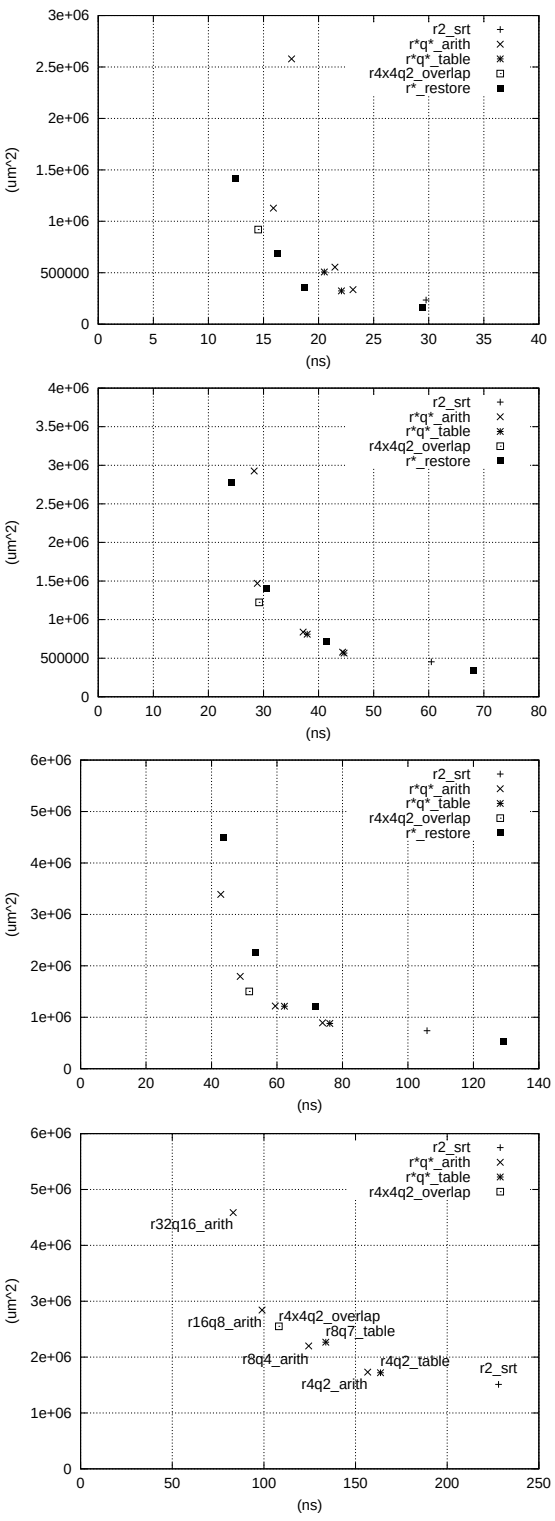

図 2: 除算回路の各手法の比較: 上から 16, 32, 54, 114 ビットの場合.

暗号処理は暗号化, 復号化に多くの演算を必要とし, アルゴリズムによっては, ソフトウエアでは処理が遅い. 特に, 公開鍵暗号では非常に長ビットの剰余演算を必要とする. そのため, 近年, 各種暗号アルゴリズムのハードウエア実装に関する研究が盛んになされている.

暗号アルゴリズムは, 秘密鍵暗号と公開鍵暗号に大別される. 本研究では, 2 章の研究を基礎として, 算術演算を必要とする公開鍵暗号に着目する. 次節では, 公開鍵暗号で一般的な RSA 暗号について述べる.

3.1 公開鍵暗号と冪剰余演算

RSA 暗号 [12] では, m を平文, c を暗号文とすると, 公開鍵 (e, N) により $c = m^e \bmod N$ を行うことで暗号化する. 復号も同様である. なお, (e, N), (d, N) はある規則の数であるが, ここでは予め与えられているとする. ここで, $e = e_{n-1} \cdot 2^{n-1} + ... + e_2 \cdot 2^2 + e_1 \cdot 2 + e_0, (e_i \in 0, 1)$ とすると, 先の式 $c = m^e \bmod N$ は, $m^e \bmod N = (m)^{e_0} \cdot (m^2)^{e_1} \cdot (m^4)^{e_2} \cdot ... \cdot (m^{2^{n-1}})^{e_{n-1}} \bmod N$ と変形でき, 次のように高速処理できる. $Result$ が結果である.

$C := m;$
$Result := 1;$
$\text{for}(i = 0; i <= n - 1; i++)\{$
  $\text{if}(e[i] == 1) Result := Result \cdot C \bmod N;$
  $C := C \cdot C \bmod N;$
$\}$

これより, 乗算剰余演算 $x \cdot x \bmod N$ が基本単位となり, この高速化が問題となる. 次節で乗算剰余演算のハードウエアについて述べる.

3.2 乗算剰余演算回路

RSA 暗号で必要な乗算剰余演算 [13] では, 一般的に 1024 ビット程度の大きな数 N による剰余をとる必要がある. これには次の回路構成手法がある.

1. 乗算と除算で行う方法 [14].
2. モンゴメリのアルゴリズムを用いる方法 [15].

1. の方法については, 除算を用いた剰余演算の高速化が問題になる. これには 1024 ビットという長ビットの除算が必要になり, 時間がかかる. 除算では, 一般的に, 商の桁を決定するために, 桁上げが生じる加算が必要であるが, ここで, SRT 除算を用い

ることにより, 加算での桁上げをなくすことで遅延時間を小さくできる. また, 除算の基数を上げることで, 繰り返しの回数を減らし高速化できる.

2. の方法は, モンゴメリにより提案された乗算剰余演算の高速化アルゴリズムである [15]. 剰余演算で必要な除算をなくし, 乗算のみで行うことにより高速化する. x, N を整数, N を n ビットの奇数, $R = 2^n$ とし, $0 \leq x < RN$ とする. このとき, $x \bmod N$ を計算したいとすると, x に N の整数倍 aN を足した数 $x + aN$ の N による剰余 $(x + aN) \bmod N$ は, $x \bmod N$ と等価である. ここで a に適切な数を選ぶと, $x + aN$ は R で割りきれるようにできる. これは, $x + aN$ の下位 n ビットが全て 0 であることと同じであり, $x + aN$ を n ビット右シフトして簡単な補正をすれば, それは $xR^{-1} \bmod N$ と同じになる. 以上を行う手続きが次の $\mathrm{REDC}(x)$ である. x を与えると $xR^{-1} \bmod N$ が得られる. なお, N' は $0 < R^{-1} < N, 0 < N' < R, RR^{-1} - NN' = 1$ を満たす整数である.

function $\mathrm{REDC}(x)$
  $a := (x \bmod R)N' \bmod R$;
  $X := (x + aN)/R$;
  if $X \geq N$ then return $X - N$ else return X;

ここで, $z = x \cdot y \bmod N$ を計算したい場合は, 予め, $X = REDC(x(R^2 \bmod N)) = xR \bmod N$ と $Y = REDC(y(R^2 \bmod N))$ を計算しておき, $Z = REDC(XY) = XYR^{-1} \bmod N = xyR \bmod N$ を計算してから, $REDC(Z) = ZR^{-1} \bmod N = z$ を計算する. このアルゴリズムは, 何度も $\bmod N$ を行う場合に有効になる. また, 予め幾つかの初期値を用意する必要がある. 近年は, このモンゴメリ乗算に基づく回路構成法に関する論文が多い [16][17][18].

3.3 進行状況と今後の課題

現在は, 上記 1.2. の各々に基づく乗算剰余演算回路の典型的な回路を設計している. 今後は, まず設計した回路から面積と速度を概算して, 各々の手法を比較することで, 明らかに有効である手法を決定する. 次に, その手法に基づく回路構成を改良していく予定である.

参考文献

[1] 高木直史: “除算回路のアルゴリズム,” 情報処理, vol. 37, no. 3, pp. 280-286, Mar. 1996.

[2] J. E. Robertson: “A New Class of Digital Division Methods,” *IRE Trans. Electronic Computers,* vol. EC-7, no. 9, pp. 218-222, Sept. 1958.

[3] D. E. Atkins: “Higher-Radix Division Using Esitimates of the Divisor and Partial Remainders,” *IEEE Trans. Computers,* vol. C-17, no. 10, pp. 925-934, Oct. 1968.

[4] G. S. Taylor: “Radix 16 SRT Dividers With Overlapped Quotient Selection Stages,” *Proc. 7th IEEE Symp. Computer Arithmetic,* pp. 64-71, 1985.

[5] N. Burgess and T. Williams: “Choices of Operand Truncation in the SRT Division Algorithm,” *IEEE Trans. Computers*, vol.44, no.7, pp.933–938, July 1995.

[6] M. D. Ercegovac and T. Lang: “*Division and Square Root – Digit-Recurrence Algorithms and Implementations*,” Kluwer Academic Publishers, 1994.

[7] P. Montuschi and T. Lang: “Boosting Very-High Radix Division with Prescaling and Selection by Rounding,” *IEEE Trans. Computers*, vol.C-50, no.1, pp.13–27, Jan. 2001.

[8] D. Wong and M. Flynn: “Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations,” *IEEE Trans. Computers*, vol.41, no.8, pp.981–995, Aug. 1992.

[9] A. R. Omondi: “*Computer Arithmetic Systems – Algorithms, Architecture and Implementations.*,” Prentice hall Inc., 1994.

[10] M. Blum and H. Wasserman: “Reflections on the Pentium Division Bug,” *IEEE Trans. Computers*, vol.45, no.4, pp.385–393, Apr. 1996.

[11] 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, vol.43, no.8, pp.2665–2673, Aug. 2002.

[12] R. L. Rivest, A. Shamir, and L. Adleman: “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,” *Commnication of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[13] 高木直史: “初等関数計算回路のアルゴリズム,” 情報処理, vol. 37, no. 4, pp. 362-368, Apr. 1996.

[14] N. Takagi: “A Radix-4 Modular Multiplication Hardware Algorithm for Modular Exponentiation,” *IEEE Trans. Computers*, vol. 41, no. 8, pp. 949–956, Aug. 1992.

[15] P. L. Montgomery: “Modular Multiplication Widthout Trial Division,” *Mathematics of computation,* vol. 44, no. 170, pp. 519–521, Apr. 1985.

[16] S. E. Eldridge, and C. D. Walter: “Hardware Implementation of Montgomery’s Modular Multiplication Algorithm,” *IEEE Trans. Computers*, vol. 42, no. 6, pp. 693–699, June 1993.

[17] J. H. Hong, and C. W. Wu: “Cellular-Array Modular Multiplier for Fast RSA Public-Key Cryptosystem Based on Modified Booth’s Algorithm,” *IEEE Trans. VLSI Systems*, vol. 11, no. 3, pp. 474–484, June 2003.

[18] 新保淳, 野崎華恵, 川村信一: “高速 RSA 暗号 LSI,” 東芝レビュー, vol. 55, no. 7, 2001.