論　文

Karatsubaアルゴリズムに基づく小面積乗算器

川島　裕崇[†]　　柴岡　雅之[††*]　　高木　直史[†]　　高木　一義[†]

Reduced Area Multipliers Based on Karatsuba Algorithm

Hirotaka KAWASHIMA[†], Masayuki SHIBAOKA[††*], Naofumi TAKAGI[†], and Kazuyoshi TAKAGI[†]

あらまし　Karatsubaアルゴリズムに基づく小面積乗算器を提案する．Karatsubaアルゴリズムはソフトウェアで多倍長乗算を効率良く行うアルゴリズムである．Karatsubaアルゴリズムを並列乗算器に適用した場合，必要な論理素子数は少ないが，配線が複雑になる．そのため，従来はKaratsubaアルゴリズムは並列乗算器には向かないと考えられてきた．VLSIにおいて使用できる配線層数が増加しており，配線が回路面積に及ぼす影響が小さくなっている．そのため，必要な論理素子数が少なくなる構成法を選択することで，小面積の回路を構成できると考えられる．Karatsubaアルゴリズムに基づく並列乗算器を設計したところ，配列型乗算器よりも小面積であった．更に，内部の計算順序を変更することによって必要な論理素子数を削減し，けた上げ伝搬加算器を削減することによって高速化した．これらの手法を適用することにより，回路面積，遅延時間の両方で改善が見られた．

キーワード　小面積乗算器，Karatsubaアルゴリズム，配線層

1. まえがき

多くのディジタル機器では，性能向上のためにASIC（特定用途向け集積回路，Application Specific Integrated Circuit）と呼ばれる集積回路を搭載している．搭載される演算回路はASICの性能を左右するため，より高速，小面積の演算回路が求められている．コストや設計期間が重視されるASICでは，スタンダードセル方式を用いた設計が広く採用されている．スタンダードセル方式では，セルと呼ばれる汎用部品を単位として回路の設計を行う．ASICでは演算回路もスタンダードセル方式で設計されることが多くなってきた．

スタンダードセル方式においては，回路の面積は回路を構成するセルの総面積とそれらを接続する配線の面積によって決まる．配線量の多い回路ではセルの上だけでは配線が収まりきらず，回路の面積を広げて配線を行う必要がある．そのため，従来から小面積の演算回路を設計するためには，用いるセルの総面積が小さく，配線がしやすい構造をもつ演算回路の構成法が用いられてきた．

近年の半導体集積回路の製造技術の進歩により，配線層数が増加している．配線層数の増加により，セルの上を通ることのできる配線量が増加する．そのため，配線量が多くなっても回路面積の増加は小さく抑えられるようになってきている．すなわち，配線量の影響が小さくなり，相対的にセルの総面積が回路面積に直接反映される．小面積の演算回路を構成するためには，配線量が多くても用いるセルの総面積が小さくなることに重点を置いて構成法を選択することが有効であると考えられる．

本論文では，少ないセル数で乗算器を構成できるアルゴリズムとして，Karatsubaアルゴリズム[1], [2]に着目し，小面積の乗算器を構成する．Karatsubaアルゴリズムは，少ない乗算命令回数で多倍長乗算を効率良く実行するアルゴリズムである．Karatsubaアルゴリズムを並列乗算器に適用した場合，必要なセル数は少ないが，配線が複雑になる．配線の複雑さから，従来はKaratsubaアルゴリズムは並列乗算器には向かないと考えられてきた．しかし，Karatsubaアルゴリ

[†] 名古屋大学大学院情報科学研究科情報システム学専攻，名古屋市
Department of Information Engineering, Graduate School of Information Science, Nagoya University, Nagoya-shi, 464–8603 Japan
[††] 名古屋大学工学部電気電子情報工学科，名古屋市
Department of Information Engineering, School of Engineering, Nagoya University, Nagoya-shi, 464–8603 Japan
[*] 現在，(株）東芝

ズムを用いた並列乗算器は必要なセル数が少ないため，配線層数が多い場合には小面積の乗算器を構成できると考えられる．

Karatsuba アルゴリズムを用いて多倍長乗算を行う専用回路の研究が行われている．[3] ではガロア体上の乗算を行う専用回路，[4], [5] では整数の多倍長乗算を行う専用回路で Karatsuba アルゴリズムを用いている．これらの専用回路では順序回路による繰返し演算で多倍長乗算を行っている．

本論文では，数十ビット程度の並列乗算器を対象とし，Karatsuba アルゴリズムに基づく小面積の乗算器を提案する．提案乗算器では，必要なセル数が少なくなるように Karatsuba アルゴリズムの計算順序を変更した．また，内部に複数現れるけた上げ伝搬加算器を削減することによって高速化した．

Karatsuba アルゴリズムをそのまま回路として実現した基本構成と，更に小面積化，高速化を施した構成の 2 種類を複数のセルライブラリを用いて設計し，評価を行った．Karatsuba アルゴリズムに基づく乗算器は，基本構成で配列型乗算器と比較して回路面積が 32 ビットで約 10%，64 ビットで約 25%小さくなった．また，小面積化，高速化手法を適用したところ，基本構成と比較して遅延面積積で 32 ビットで 13〜19%，64 ビットで 23〜33%の改善が見られた．

以降，**2.** で Karatsuba アルゴリズムの説明を行う．**3.** で提案手法とそれを用いた乗算器の構成について述べ，**4.** で提案手法に基づく乗算器を評価する．**5.** で本論文のまとめを行う．

2. Karatsuba アルゴリズム

Karatsuba アルゴリズムは少ない乗算命令回数で多倍長乗算を行うアルゴリズムである．n ビット数 X, Y の乗算を考える．X, Y の上位 $\frac{1}{2}n$ ビット，下位 $\frac{1}{2}n$ ビットをそれぞれ X_H, X_L, Y_H, Y_L とすると，X, Y は

$$X = 2^{\frac{1}{2}n} X_H + X_L$$
$$Y = 2^{\frac{1}{2}n} Y_H + Y_L$$

と表現される．この表現を用いると X と Y の積は

$$\begin{aligned} XY &= \left(2^{\frac{1}{2}n} X_H + X_L\right)\left(2^{\frac{1}{2}n} Y_H + Y_L\right) \\ &= 2^n X_H Y_H + X_L Y_L \\ &\quad + 2^{\frac{1}{2}n}\left(X_H Y_L + Y_H X_L\right) \end{aligned}$$

と表される．ここで，

$$\begin{aligned} X_H Y_L + X_L Y_H = &\left(X_H + X_L\right)\left(Y_H + Y_L\right) \\ &- X_H Y_H - X_L Y_L \end{aligned}$$

と変形を行い，$X_H Y_H, X_L Y_L$ を再利用する．変形前は 2 回必要であった乗算が，$X_H Y_H$，$X_L Y_L$ の再利用によって変形後は $(X_H + X_L)(Y_H + Y_L)$ の 1 回のみとなっている．この変形により，

$$\begin{aligned} XY = &2^n X_H Y_H + X_L Y_L \\ &+ 2^{\frac{1}{2}n}((X_H + X_L)(Y_H + Y_L) - X_H Y_H \\ &- X_L Y_L) \end{aligned}$$

となる．

$$P_1 = X_H Y_H$$
$$P_2 = X_L Y_L$$
$$P_3 = \left(X_H + X_L\right)\left(Y_H + Y_L\right)$$

として整理すると，

$$XY = 2^n P_1 + P_2 + 2^{\frac{1}{2}n}\left(P_3 - P_1 - P_2\right) \qquad (1)$$

となる．

Karatsuba アルゴリズムを用いて乗算を行う場合，P_1，P_2 はそれぞれ 1 回の乗算で，P_3 は 2 回の加算と 1 回の乗算で求められる．$2^n P_1$ と P_2 は n ビットずれており重なる部分がないため，$2^n P_1 + P_2$ は加算を行う必要がない．そのため，P_1，P_2，P_3 から XY を求めるためには，3 回の加減算が必要となる．これらを合計すると，Karatsuba アルゴリズムを用いることで n ビット乗算は約 $\frac{1}{2}n$ ビットの乗算 3 回と加算 5 回で実行できる．

Karatsuba アルゴリズムは再帰的に適用することができる．アルゴリズム中に現れる 3 回の乗算を更に Karatsuba アルゴリズムを用いて計算することができる．

3. Karatsuba アルゴリズムに基づく乗算器の構成法

Karatsuba アルゴリズムを用いると，少ない演算量で乗算を実行できる．したがって，Karatsuba アルゴリズムを用いて並列乗算器を構成した場合，少ないセル数で構成できると考えられる．本論文では Karatsuba アルゴリズムに基づく乗算器を Karatsuba 乗算器と呼ぶ．**3. 1**，**3. 2** では Karatsuba 乗算器の基本構成について述べる．**3. 3** では Karatsuba 乗算器

の小面積化手法，**3.4** では用いる全加算器数を増やさずに高速化する手法を提案する．

3.1 Karatsuba 乗算器の基本構成

本節では，式 (1) を 1 回だけ用いた場合の Karatsuba 乗算器の構成について述べる．式 (1) に基づく符号なし整数乗算器の構成を図 1 に示す．図中の CPA はけた上げ伝搬加算器（Carry Propagate Adder）を示す．図 1 では二つのけた上げ伝搬加算器と三つの乗算器を用いて P_1，P_2，P_3 を算出する．$-P_1$，$-P_2$ を得るためには，P_1，P_2 をビット反転し，更にそれぞれの最下位けたに 1 を加算する．これらを加算して積を求める．本論文では，P_1，$-P_1$，P_2，$-P_2$，P_3 から最終的な乗算結果を求めるための回路を加算モジュール（Addition Module）と呼ぶ．

Karatsuba アルゴリズムを適用することで小面積となる n の範囲を見積もる．並列乗算器は，主に全加算器，半加算器，AND 回路によって構成される．このうち，並列乗算器の面積の多くは全加算器に占められる．ここでは全加算器数を概算し，比較することによって大まかな有効範囲の見積りとする．簡単のために半加算器，AND 回路の個数については考えないこととする．

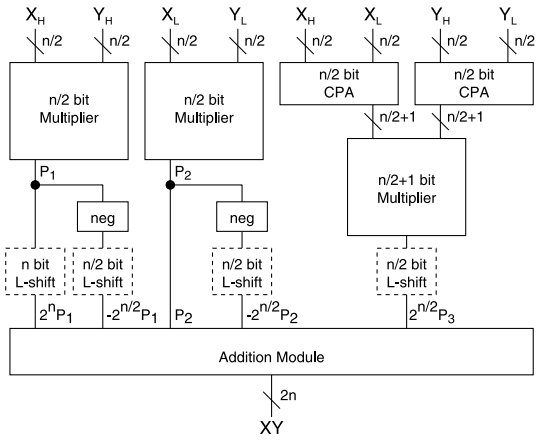

図 1　Karatsuba 乗算器の構成
Fig. 1　Block diagram of Karatsuba multiplier.

まず，P_1，P_2，P_3 を算出するために必要な全加算器数を求める．P_1，P_2 はそれぞれ 1 個の $\frac{1}{2}n$ ビット乗算器，P_3 は 2 個の $\frac{1}{2}n$ ビット加算器と 1 個の $\frac{1}{2}n+1$ ビット乗算器が必要である．$\frac{1}{2}n$ ビット加算器は $\frac{1}{2}n-1$ 個の全加算器で構成できる．$\frac{1}{2}n$ ビット乗算器，$\frac{1}{2}n+1$ ビット乗算器に必要な全加算器数をそれぞれ $C_{mult}\left(\frac{1}{2}n\right)$，$C_{mult}\left(\frac{1}{2}n+1\right)$ とする．これらを合計すると $2C_{mult}\left(\frac{1}{2}n\right)+C_{mult}\left(\frac{1}{2}n+1\right)+n-2$ となる．

次に，P_1，P_2，$-P_1$，$-P_2$，P_3 を加算するために必要な全加算器数を見積もる．P_1，P_2，$-P_1$，$-P_2$ はそれぞれ n ビット，P_3 は $n+2$ ビットである．P_1，P_2，$-P_1$，$-P_2$，P_3 を合わせると $5n+2$ ビットとなる．これを全加算器を用いて $2n$ ビットまで加算する．全加算器は 1 個につき，3 ビットを入力し，2 ビットが出力されるため，全加算器を 1 個用いることで 1 ビット減らすことができる．そのため，加算によって減るビット数と必要な全加算器数が一致する．$5n+2$ ビットを $2n$ ビットまで加算するために必要な全加算器数は $3n+2$ 個となる．8 ビット Karatsuba 乗算器における加算モジュールの入力を図 2 に示す．P_1 の下位から i ビット目を $P_1[i]$ と表す．P_2，P_3 についても同様である．最下段の 1 のうち，下位の 1 ビットは $-P_1$，$-P_2$ の 2 の補正表現のための 1 を加算したものである．上位の 3 ビットは $-P_1$，$-P_2$ の符号拡張のための 1 を加算したものである．

以上より，Karatsuba アルゴリズムに基づく乗算器に必要な全加算器数を $C_K(n)$ とすると，

$$C_K(n) = 2C_{mult}\left(\frac{1}{2}n\right) + C_{mult}\left(\frac{1}{2}n+1\right) + 4n \qquad (2)$$

となる．Karatsuba 乗算器の内部で配列型乗算器や Wallace 乗算器を用いた場合を考える．n ビットの配列型乗算器や Wallace 乗算器では，部分積が n^2 ビット生成され，これを加算して $2n$ ビットの出力を得る．

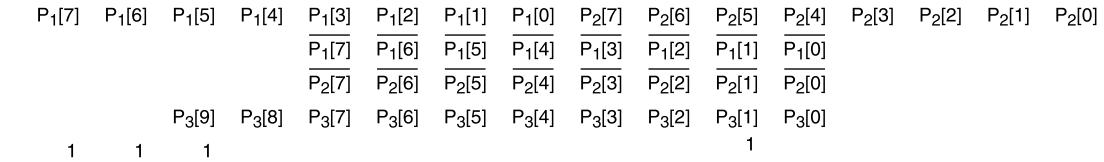

図 2　Karatsuba 乗算器の加算モジュールの入力
Fig. 2　Inputs for addition module.

n^2 ビットを $2n$ ビットまで減らすため，n ビットの配列型乗算器，Wallace 乗算器には n^2-2n 個の全加算器が必要である．そのため，

$$C_{mult}\left(\frac{1}{2}n\right)=\frac{1}{4}n^2-n$$
$$C_{mult}\left(\frac{1}{2}n+1\right)=\frac{1}{4}n^2-1$$

となる．このとき Karatsuba 乗算器の全加算器数は

$$\begin{aligned}C_K(n)&=2\left(\frac{1}{4}n^2-n\right)+\left(\frac{1}{4}n^2-1\right)+4n\\&=\frac{3}{4}n^2+2n-1\end{aligned}\tag{3}$$

となる．配列型乗算器と Karatsuba 乗算器の全加算器数を比較すると，18 ビット以上で配列型乗算器よりも Karatsuba 乗算器の全加算器数が少なくなる．そのため，入力幅が 18 ビット周辺で配列型乗算器よりも Karatsuba 乗算器の方が小面積となると予想される．

3.2 Karatsuba アルゴリズムの再帰的適用による小面積化

本節では乗算器に Karatsuba アルゴリズムを再帰的に適用した場合を考える．

Karatsuba アルゴリズムを用いた乗算器は $\frac{1}{2}n$ ビット乗算器 3 個と式 (1) の 5 回の加算を行う加算器が必要となる．一方，配列型乗算器は $\frac{1}{2}n$ ビット乗算器約 4 個のハードウェア量となる．配列型乗算器などと比較すると，$\frac{1}{2}n$ ビット乗算器約 1 個分のハードウェア量が削減でき，一方で加算器の追加が必要となる．n がある値よりも大きい場合には，追加する加算器よりも削減できる $\frac{1}{2}n$ ビット乗算器のセル数の方が多くなる．このとき，Karatsuba 乗算器は配列型乗算器よりも少ないセル数で構成できる．一方，n がある値よりも小さい場合には，Karatsuba 乗算器が配列型乗算器よりも多くのセルを必要とするようになる．

Karatsuba アルゴリズムを再帰的に適用し，乗算器を分解していくと，内部で用いる乗算器のサイズは小さくなっていく．そのため，再帰的に適用することによってセル数を少なくすることのできる最小のサイズがあると考えられる．

Karatsuba アルゴリズムを 2 回適用した場合の全加算器数を求める．n は 4 の倍数とする．式 (3) と同様に考えると，

$$C_K\left(\frac{1}{2}n\right)=\frac{3}{16}n^2+n-1$$
$$C_K\left(\frac{1}{2}n+1\right)=\frac{3}{16}n^2+2n+5$$

表 1　配列型乗算器，Karatsuba 乗算器の全加算器数
Table 1　Number of FAs for array and Karatsuba multipliers.

	array multiplier	Karatsuba multiplier	
		1 level	2 levels
8 bit	48	63	103
16 bit	224	223	275
32 bit	960	831	835
64 bit	3968	3199	2819
128 bit	16128	12543	10243

となる．これらを用いると，Karatsuba アルゴリズムを 2 段適用した乗算器に必要な全加算器数 C_{K2} は

$$\begin{aligned}C_{K2}(n)&=2\left(\frac{3}{16}n^2+n-1\right)\\&\quad+\left(\frac{3}{16}n^2+2n+5\right)+4n\\&=\frac{9}{16}n^2+8n+3\end{aligned}\tag{4}$$

となる．

式 (3) と式 (4) を比較すると，36 ビット以上で 1 段よりも 2 段の Karatsuba 乗算器の全加算器数が少なくなる．そのため，36 ビット前後で 2 段の Karatsuba 乗算器の方が小面積となると予想される．

式 (3)，式 (4) から，いくつかの n について得られた必要な全加算器数を表 1 に示す．16 ビット以下では配列型乗算器，32 ビットでは 1 段の Karatsuba 乗算器，64 ビット以上では 2 段の Karatsuba 乗算器が最も全加算器数が少なくなっている．

3.3 計算順序の変更による素子数の削減

本節では，計算の順序を変更することによって素子数を削減する手法を提案する．**3.1** の構成では，P_1，P_2 を分岐して，適切にシフト，反転した上で加算モジュールでそれぞれを加算している．P_1，P_2 それぞれを分岐するのではなく，加算してから分岐することによって加算モジュールの入力ビット数を減らすことができる．これは，式 (1) に以下の変形を行うことに相当する．

$$\begin{aligned}XY&=2^nP_1+P_2+2^{\frac{1}{2}n}(P_3-P_1-P_2)\\&=-2^{\frac{1}{2}n}\left(-2^{\frac{1}{2}n}+1\right)P_1\\&\quad+\left(-2^{\frac{1}{2}n}+1\right)P_2+2^{\frac{1}{2}n}P_3\\&=\left(-2^{\frac{1}{2}n}P_1+P_2\right)\left(-2^{\frac{1}{2}n}+1\right)+2^{\frac{1}{2}n}P_3\end{aligned}\tag{5}$$

ここで，

$$P_4 = -2^{\frac{1}{2}n}P_1 + P_2$$

とすると

$$XY = \left(-2^{\frac{1}{2}n} + 1\right)P_4 + 2^{\frac{1}{2}n}P_3 \tag{6}$$

となる．

ソフトウェアで式 (6) を用いた場合，乗算 3 回と加算 5 回が必要となり，演算量は式 (1) と変わらない．一方，組合せ回路では，式 (6) に基づいて回路を構成することにより，必要な全加算器数を削減できる．式 (6) を用いた Karatsuba 乗算器の構成を図 3 に示す．この構成における必要な全加算器数を求める．P_1，P_2，P_3 の算出に必要な全加算器数は **3. 2** と同様である．P_1，P_2 はそれぞれ n ビットであり $\frac{1}{2}n$ ビットが重なっているため，P_4 を求めるためには全加算器 $\frac{1}{2}n$ 個が必要となる．P_3 は $n+2$ ビットであり，P_4 は $\frac{3}{2}n$ ビットとなるため，P_4，$-P_4$，P_3 を合わせて $4n+2$ ビットとなる．本手法を適用した場合の加算モジュールへの入力を図 4 に示す．P_4 の下位から i ビット目を $P_4[i]$ と表す．また，最下段の 1 は $-P_4$ の 2 の補数表現を補正するための 1 である．P_4 は 2 の補数表現で表され，最上位ビットをそろえるために $\frac{1}{2}n$ ビットの拡張が必要となる．これらを合計すると $\frac{9}{2}n+2$ ビットとなり，$2n$ ビットまで加算するため $\frac{5}{2}n+2$ 個の全加算器が必要となる．以上より，Karatsuba アルゴリズムに基づく乗算器の全加算器数は

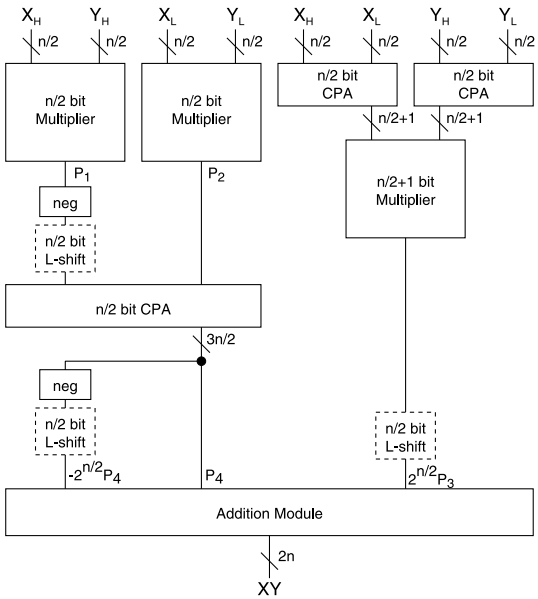

図 3　P_4 を用いた Karatsuba 乗算器の構成
Fig. 3　Block diagram of Karatsuba multiplier using P_4.

$$C_{K'}(n) = \frac{3}{4}n^2 + \frac{3}{2}n - 1 \tag{7}$$

となる．**3. 2** の構成と比較すると，加算モジュールで必要な全加算器数が n 個削減できる一方，P_4 の算出に $\frac{1}{2}n$ 個の全加算器が必要となるため，これを差引いて $\frac{1}{2}n$ 個の全加算器が削減できる．Karatsuba アルゴリズムを 2 段適用した場合の全加算器数は，**3. 2** と同様に考えると，

$$C_{K2'}(n) = \frac{9}{16}n^2 + \frac{29}{4}n + 7 \tag{8}$$

となる．

式 (7)，式 (8) より得られた必要な全加算器数を表 2 に示す．必要な全加算器数は表 1 の Karatsuba 乗算器の全加算器数よりも少なくなっている．また，32 ビットで同数，64 ビット以上で 2 段の Karatsuba 乗算器の全加算器数が少なくなっており，32 ビット以上で再帰的に Karatsuba アルゴリズムを適用すると全加算器数を減らすことができる．

3. 4　けた上げ伝搬加算の削減による高速化

本節では，用いる全加算器数を増やさずに Karatsuba 乗算器を高速化する手法を提案する．図 3 の構

表 2　計算順序を変更した Karatsuba 乗算器の全加算器数
Table 2　Number of FAs for Karatsuba multipliers using the arranged algorithm.

	1 level	2 levels
8 bit	59	101
16 bit	215	267
32 bit	815	815
64 bit	3167	2775
128 bit	12479	10151

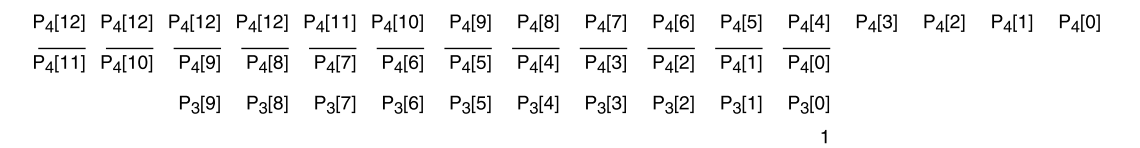

図 4　P_4 を用いた加算モジュールの入力
Fig. 4　Inputs for the addition module using P_4.

$P_4[12]$	$P_4[12]$	$P_4[12]$	$P_4[12]$	$P_4[11]$	$P_4[10]$	$P_4[9]$	$P_4[8]$	$P_4[7]$	$P_4[6]$	$P_4[5]$	$P_4[4]$	$P_4[3]$	$P_4[2]$	$P_4[1]$	$P_4[0]$
$\overline{P_4[11]}$	$\overline{P_4[10]}$	$\overline{P_4[9]}$	$\overline{P_4[8]}$	$\overline{P_4[7]}$	$\overline{P_4[6]}$	$\overline{P_4[5]}$	$\overline{P_4[4]}$	$\overline{P_4[3]}$	$\overline{P_4[2]}$	$\overline{P_4[1]}$	$\overline{P_4[0]}$				
							P_304	P_303	P_302	P_301	P_300				
						P_314	P_313	P_312	P_311	P_310	1				
					P_324	P_323	P_322	P_321	P_320						
				P_334	P_333	P_332	P_331	P_330							
			P_344	P_343	P_342	P_341	P_340								

図 5　P_3 の部分積を用いた加算モジュールの入力
Fig. 5　Inputs for addition module using PPs of P_3.

			$\overline{P_103}$	$\overline{P_102}$	$\overline{P_101}$	$\overline{P_100}$	P_203	P_202	P_201	P_200
		$\overline{P_113}$	$\overline{P_112}$	$\overline{P_111}$	$\overline{P_110}$	P_213	P_212	P_211	P_210	
	$\overline{P_123}$	$\overline{P_122}$	$\overline{P_121}$	$\overline{P_120}$	P_223	P_222	P_221	P_220		
$\overline{P_133}$	$\overline{P_132}$	$\overline{P_131}$	$\overline{P_130}$	P_233	P_232	P_231	P_230			
		1	1	1	1	1				

図 6　$-P_1$，P_2 の部分積
Fig. 6　Partial products of $-P_1$, P_2.

成では，7 箇所でけた上げ伝搬加算器を用いている．すなわち，X_H と X_L，Y_H と Y_L の加算を行う加算器，3 箇所の乗算の最終加算器，$-2^{\frac{n}{2}}P_1 + P_2$ を行う加算器，加算モジュールの最後に現れるけた上げ伝搬加算である．けた上げ伝搬加算は大きな遅延時間が必要となるため，これを減らすことによって高速化が期待できる．

まず，P_3 の計算に着目する．図 3 では，乗算器を用いて P_3 を求め，加算モジュールで P_4，$-2^{\frac{1}{2}n}P_4$ と加算している．P_3 の部分積を加算モジュールの入力とし，P_4，$-2^{\frac{1}{2}n}P_4$ と加算するように変更する．P_3 の部分積を用いた加算モジュールの入力を図 5 に示す．P_3ij は P_3 の i 個目の部分積の下位から j ビット目を表す．この変更によって P_3 を求める乗算器の最終加算が必要なくなる．加算するビット数は変化していないため，必要な全加算器数は変化しない．

次に，P_4 の計算に着目する．P_4 は $-2^{\frac{1}{2}n}P_1$ と P_2 を加算することによって得られる．P_4 を算出するためには，P_1 を求めるための乗算の最終加算，P_2 を求めるための乗算の最終加算，$-P_1$ と P_2 の加算の三つのけた上げ伝搬加算器を用いている．このうち，P_1，P_2 を求める乗算器の最終加算のけた上げ伝搬加算器を削除できる．そのためには，$-P_1$，P_2 を部分積を統合して加算し，P_4 を算出する．その場合，$-P_1$ と P_2 の部分積が必要となる．$-P_4$ の各部分積は負数となるため，符号拡張が必要である．この符号拡張は 1 ビットで済む．図 6 に 8 ビット Karatsuba 乗算器において，P_4 を算出するために加算する $-P_1$ と P_2 の部分積を示す．ここで P_1ij は P_1 の i 個目の部分積の下位から j ビット目を表す．P_2ij についても同様である．$-P_1$ を扱うために，P_1 の部分積の値を反転し，最下位けたに 1 を追加している．また，$-P_1$ の各部分積の符号拡張は，事前に足し合わせることにより 1 ビットで済む．図中の 5 ビットの 1 のうち，最上位の 1 ビットが符号拡張ビットであり，下位 4 ビットが部分積の 2 の補数表現の補正ビットである．

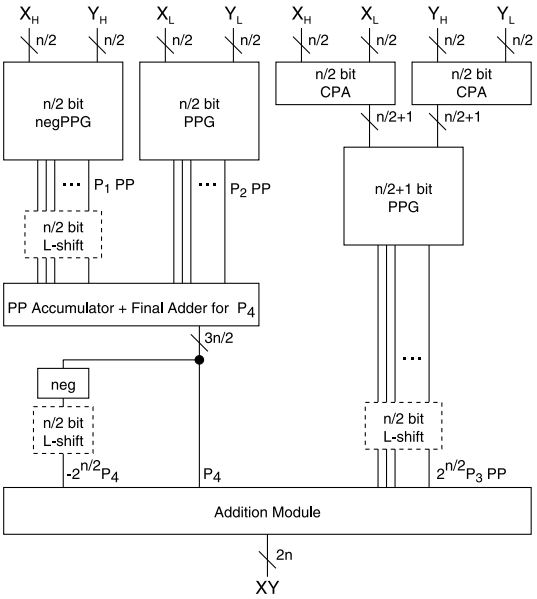

図 7　けた上げ伝搬加算を削減した Karatsuba 乗算器の構成
Fig. 7　Block diagram of Karatsuba multiplier with reduced number of CPAs.

けた上げ伝搬加算器を削減した Karatsuba 乗算器の構成を図 7 に示す．図中の PP は部分積（Partial Product），PPG は部分積生成回路（Partial Product Generator）を示す．これらの手法を用いることで図 3

表 3　乗算器のセルの総面積（mm^2）
Table 3　Total cell area of multipliers.

	ROHM0.35 μm			ROHM0.18 μm			STARC90 nm		
	array multiplier	Karatsuba (fund.) 1 level	Karatsuba (fund.) 2 levels	array multiplier	Karatsuba (fund.) 1 level	Karatsuba (fund.) 2 levels	array multiplier	Karatsuba (fund.) 1 level	Karatsuba (fund.) 2 levels
14 bit	0.090065	0.096306	—	0.017938	0.018850	—	0.004160	0.004375	—
16 bit	0.118775	0.121944	—	0.023698	0.023957	—	0.005499	0.005562	—
18 bit	0.151445	0.150552	—	0.030259	0.029663	—	0.007024	0.006889	—
20 bit	0.188075	0.182155	—	0.037620	0.035978	—	0.008736	0.008356	—
28 bit	0.374195	0.355612	0.372854	0.075063	0.069205	0.072099	0.017446	0.016044	0.016708
32 bit	0.491015	0.454330	0.468391	0.098584	0.088846	0.087904	0.022919	0.020623	0.020445
36 bit	0.625316	0.574043	0.572227	0.125644	0.112028	0.106974	0.029161	0.025999	0.024884
40 bit	0.774012	0.698681	0.684499	0.156603	0.136782	0.128395	0.036127	0.031747	0.029854
44 bit	0.938548	0.838579	0.803442	0.188762	0.163667	0.152722	0.043838	0.038110	0.035483

で 7 個用いていたけた上げ伝搬加算器は 4 個まで減らすことができる．Karatsuba アルゴリズムを 2 段適用した場合には，21 個のけた上げ伝搬加算器を 13 個まで減らすことができる．

4. 評　　価

本章では Karatsuba 乗算器の評価を行う．比較には以下の乗算器を用いる．

- 配列型乗算器
- Wallace 乗算器
- 基本 Karatsuba 乗算器
- 改良 Karatsuba 乗算器

基本 Karatsuba 乗算器では図 1 の構成を用いている．改良 Karatsuba 乗算器は基本 Karatsuba 乗算器に対して **3.3**，**3.4** の手法を適用した乗算器である．改良 Karatsuba 乗算器でも内部の部分積加算には Wallace 木を用いている．

乗算器の設計には以下のプロセスのセルライブラリを用いた．

- ROHM0.35 μm/メタル 3 層
- ROHM0.18 μm/メタル 5 層
- STARC90 nm/メタル 6 層

論理最適化には Synopsys 社 Design Compiler を用いた．レイアウトツールは ROHM0.35 μm，STARC90 nm では Synopsys 社 Astro，ROHM 0.18 μm では Cadence 社 SOC Encounter を用いた．

4.1 Karatsuba アルゴリズムの適用範囲

3.1，**3.2** では，全加算器数を用いて Karatsuba アルゴリズムを適用することの有効範囲を見積もった．実際には，全加算器以外のセルや用いるセルライブラリによって，Karatsuba アルゴリズムを適用することによって小面積となる範囲は変動する．本節では，設計した乗算器のセルの総面積を比較し，本論文で用いているセルライブラリでの実際の有効範囲について調べる．Karatsuba 乗算器の全加算器数とセルの総面積を表 3 に示す．

配列型乗算器と 1 段の Karatsuba 乗算器の比較には，14〜20 ビットの乗算器を用いた．どのセルライブラリを用いた場合でも，18 ビット以上で基本 Karatsuba 乗算器のセルの総面積が小さくなっている．

1 段と 2 段の Karatsuba 乗算器の比較には，28〜44 ビットの乗算器を用いた．ROHM0.35 μm では 36 ビット，その他のセルライブラリでは 32 ビット以上で 2 段の基本 Karatsuba 乗算器のセルの総面積が小さくなっている．

4.2 提案手法の評価

32 ビット，64 ビット乗算器を比較し，Karatsuba 乗算器を評価した．32 ビットでは 1 回，64 ビットでは 2 回 Karatsuba アルゴリズムを適用する．すなわち，64 ビット基本 Karatsuba 乗算器の内部では 32 ビット，33 ビットの基本 Karatsuba 乗算器を用いている．更に，それぞれの内部では 16 ビット，17 ビット，18 ビットの Wallace 乗算器を用いている．

それぞれの乗算器は以下の構成をとる．

- 回路面積が最小となる構成
- 遅延時間と回路面積の積（$D \times A$）が最小となる構成

回路面積が最小となる構成では，けた上げ伝搬加算に順次けた上げ加算器を用いた．$D \times A$ が最小となる構成では，けた上げ伝搬加算器に Synopsys 社の DesignWare を用い，乗算器の $D \times A$ が最小となるように最適化を行った．最適化を行うにあたり，制約条件として回路面積を最小とし，遅延制約を段階的に変化させた．

表 4　面積が最小となる構成の回路面積（mm^2）と遅延時間（ns）
Table 4　Circuit area (mm^2) and delay (ns) of the multipliers with the smallest area.

		ROHM0.35 μm		ROHM0.18 μm		STARC90 nm	
		Area	Delay	Area	Delay	Area	Delay
32 bit	array	0.507299	32.32	0.1013333	27.16	0.02326834	14.12
	Wallace	0.515861	25.52	0.1027876	20.88	0.02391352	9.00
	Karatsuba (fund.)	0.465462	27.75	0.0918419	22.45	0.02097859	10.82
	Karatsuba (prop.)	0.460385	27.76	0.0880906	22.79	0.02033453	10.50
64 bit	array	2.066402	69.85	0.4104705	55.41	0.09551770	29.16
	Wallace	2.114112	54.83	0.4152839	41.71	0.09682051	17.99
	Karatsuba (fund.)	1.592628	56.93	0.3045652	44.22	0.07064960	23.02
	Karatsuba (prop.)	1.590741	58.43	0.2985214	45.78	0.07042650	20.55

表 5　$D \times A$ が最小となる構成の回路面積（mm^2），遅延時間（ns）と消費エネルギー（nJ）
Table 5　Circuit area (mm^2) and delay (ns) of the multipliers with the smallest D×A.

		ROHM0.35 μm		ROHM0.18 μm		STARC90 nm	
		Area	Delay	Area	Delay	Area	Delay
32 bit	array	0.739596	17.76	0.1133860	15.20	0.0321913	6.44
	Wallace	0.769566	6.63	0.1203564	6.11	0.0341806	2.54
	Karatsuba (fund.)	0.721634	9.20	0.1270503	8.20	0.0342324	4.10
	Karatsuba (prop.)	0.664218	8.75	0.1162835	7.56	0.0321913	3.54
64 bit	array	2.324094	39.91	0.4432495	30.96	0.1268492	13.37
	Wallace	2.324094	8.93	0.4513399	7.99	0.1214239	3.85
	Karatsuba (fund.)	2.122848	18.63	0.4161224	13.25	0.1043675	7.39
	Karatsuba (prop.)	2.118480	12.69	0.3877662	10.94	0.1011365	5.91

回路面積が最小となる構成での回路面積と遅延時間を表 4，$D \times A$ が最小となる構成での回路面積と遅延時間を表 5 に示す．レイアウトでは，すべての乗算器で回路面積に対してセルの面積の合計が 95% 以上となった．遅延時間はバックアノテーションを行い，論理合成ツールによって求めた．回路面積が最小となる構成では，全加算器数は表 1，表 2 とほぼ一致した．

基本 Karatsuba 乗算器の評価を行う．回路面積が最小となる構成では，配列型乗算器，Wallace 乗算器と比較して 32 ビットでは約 8〜10%，64 ビットでは約 23〜26%小面積となっている．

次に，改良 Karatsuba 乗算器を評価する．基本 Karatsuba 乗算器と比較すると，回路面積が最小となる構成では，32 ビットでは最大約 4%，64 ビットでは最大約 2%小面積となっている．一方，遅延時間は改良 Karatsuba 乗算器の方が大きくなっている．これは P_4 を用いることで，基本 Karatsuba 乗算器よりもけた上げ伝搬加算器の遅延時間が大きくなったためであると考えられる．$D \times A$ が最小となる構成では，32 ビットでは回路面積で約 6〜8%，遅延時間で約 5〜14%の改善が見られる．$D \times A$ は約 13〜19%改善されている．64 ビットでは回路面積で最大 7%，遅延時間で約 18〜32%の改善が見られる．$D \times A$ は約 23〜33%改善されている．

一方，改良 Karatsuba 乗算器と Wallace 乗算器と比較すると，遅延時間と $D \times A$ で Wallace 乗算器が優れている．Karatsuba アルゴリズムを用いると，クリティカルパスに 2 回のけた上げ伝搬加算が含まれる．そのため，Wallace 乗算器よりも遅延時間が大きくなったものと考えられる．

Karatsuba 乗算器は小面積であるため，消費電力の面でも優位であると考えられる．近年，プロセスの微細化により，リーク電流による消費電力が増大している．リーク電流は回路の素子数に比例する．Karatsuba 乗算器は少ないセル数で構成できるため，リーク電流による消費電力を少なく抑えることが可能であると考えられる．

5. むすび

本論文では，近年の配線層数の増加により，配線量が多くても少ないセル数で演算回路を構成することによって，小面積の演算回路を構成できると考え，Karatsuba アルゴリズムに基づく乗算器を提案した．基本 Karatsuba 乗算器は配列型乗算器，Wallace 乗算器よりも小面積であった．用いるセルの総面積が小さくなることに重点を置いて構成法を選択することに

より，小面積の乗算器が構成できることが確認できた．

Karatsuba 乗算器を小面積化，高速化する手法を提案，評価した．Karatsuba アルゴリズムの計算順序を変更することで必要なセル数を削減し，Karatsuba 乗算器で使用されるけた上げ伝搬加算を削減して用いる全加算器数を増やさずに高速化した．提案手法を適用した改良 Karatsuba 乗算器は，基本 Karatsuba 乗算器と比べ回路面積，遅延時間の両方で改善が見られた．提案手法を用いることにより，Karatsuba 乗算器を更に小面積化，高速化することが可能であることが確認できた．

他の演算器でも，多くの配線層数が使用できる設計において小面積の演算器を構成するためには，配線量が多くても少ないセル数で回路を構成できるアルゴリズムを選択することが有効であると考えられる．

謝辞　本研究は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，日本シノプシス（株），日本ケイデンス・デザイン・システムズ社，ローム（株），（株）半導体理工学研究センターの協力で行われたものである．

文　献

[1] A. Karatsuba and Yu. Ofman, “Multiplication of multidigit numbers on automata,” Soviet Physics Doklady, vol.7, pp.595–596, Jan. 1963.

[2] D. Knuth, The Art of Computer Programming, vol.2: Seminumerical Algorithms, Third ed., Addison-Wesley, 1997.

[3] Z. Dyka and P. Langendoerfer, “Area efficient hardware implementation of elliptic curve cryptography by iteratively applying Karatsuba’s method,” Proc. Design, Automation and Test in Europe Conference and Exhibition, vol.3, pp.70–75, Munich, Germeny, March 2005.

[4] S. Yazaki and K. Abe, “VLSI implementation of Karatsuba algorithm and its evaluation,” Proc. International Workshop on Modern Science and Technology 2006, pp.378–383, Wuhan, China, May 2006.

[5] S. Yazaki and K. Abe, “VLSI design of iterative Karatsuba multiplier and its evaluation,” Proc. 4th IASTED International Conference on Circuits, Signals, and Systems, pp.313–318, San Francisco, USA, Nov. 2006.

[6] P.L. Montgomery, “Five, six, and seven-term Karatsuba-like formulae,” IEEE Trans. Comput., vol.54, no.3, pp.362–369, 2005.

[7] C.S. Wallace, “A suggestion for a fast multiplier,” IEEE Trans. Electronic Computers, vol.EC-13, no.1, pp.14–17, 1964.

（平成 19 年 10 月 18 日受付，20 年 2 月 1 日再受付）

川島　裕崇　（学生員）

平 14 名大・工・電気電子情報工卒，同年日本ユニシス（株）入社，平 17 同社を退社．平 19 名大大学院修士課程了．現在，同大学院博士課程在学中．算術演算回路の設計に関する研究に従事．

柴岡　雅之

平 17 名大・工・電気電子情報工卒，平 19 東工大大学院情報理工学研究科修士課程了．現在，（株）東芝に在籍．

高木　直史　（正員）

昭 56 京大・工・情報工卒，昭 58 同大大学院修士課程了，昭 63 京大工博．昭 59 京大・工・情報工助手，平 3 同助教授，平 6 名大・工・情報工助教授，平 10 同教授，平 15 名大・情報科学・教授．算術演算回路，ハードウェアアルゴリズム，論理設計等の研究に従事．平 7 日本 IBM 科学賞，情報処理学会坂井記念特別賞等受賞．

高木　一義　（正員）

平 3 京大・工・情報工卒．平 5 同大大学院修士課程了．平 7 奈良先端大助手．平 11 名大・工・情報工助手，平 12 同講師，平 18 同助教授，平 19 同准教授．工博．現在，計算理論，VLSI 設計，VLSI CAD アルゴリズムの研究に従事．