Vol.2009-SLDM-141 No.18
2009/10/16

セレクタ論理を用いた高速な差積演算器の設計とバタフライ演算への応用

塚本洋平[†1] 戸川望[†1] 柳澤政生[†1]
大附辰夫[†1] 外村元伸[†2]

システム LSI は通信，動画像，音声処理などの複雑で規模の大きな演算を高速に処理するために特定の計算に特化した専用演算器を搭載してきた．その一つが積和演算を行う MAC 演算器である．これは部分積加算を拡張することで桁上げ伝播遅延を削減でき，結果として乗算 1 回分と同等の遅延時間で計算できる．一方差積演算に注目すると，部分積が決定するのに減算の桁上げ遅延を待たねばならず全体の遅延は減算と乗算 2 つの遅延の合計となる．本稿ではこの問題に対し差積演算の部分積を適切にまとめたものがセレクタ回路の計算と等価となることに注目し，セレクタ論理を用いて部分積を高速に生成し差積演算の速度を向上する手法を提案する．次に設計した差積演算器を FFT におけるバタフライ演算に組み込むことを考える．FFT は無線通信，動画像処理などの分野で高サンプル数の演算が求められており，それらに対応するために高速なバタフライ演算器が必要である．これに対しバタフライ演算のクリティカルパスは複素減算，乗算演算でありこれに上述の差積演算回路を適用することで高速化できることを示す．

High-speed Sub-Multiplication Arithmetic Unit Design by Selector Logic and Novel Buttefly Unit As an Application

Youhei Tsukamoto,[†1] Nozomu Togawa,[†1]
Masao Yanagisawa,[†1] Tatsuo Ohtsuki,[†1]
Motonobu Tonomura[†2] and

Large-scale network and multimedia application LSIs include application specific arithmetic circuits. A multiply-accumulator (MAC) which is one of these optimized circuits extends partial-products addition and decreases carry propagations. However, there is no method similar to MAC to execute subtract-multiplication. In this paper, we propose a high-speed subtract-multiplier that decreases latency of subtract operation by bit-level transformation using selector-logics. Partial products are calculated directly by bit-level transformation and its total number is compressed to approximately half. The proposed subtract-multiplier can apply to even any kind of systems using subtract-multiplications and butterfly operation in FFT is a suitable application using them. Experimental results show that our proposed butterfly operation circuit improves the performance by 33.0%, compared to a conventional one.

1. はじめに

システム LSI の特徴の一つは MPU にはない多様な専用演算器が組み込まれている点である．これは通信，動画像，音声処理など複雑で規模の大きな演算を高速に処理するために開発されてきたものである．今日でも消費者のニーズや科学技術計算のために高速な専用演算器を搭載したシステム LSI に対する需要が大きい．

専用演算器の 1 つに，積和演算がある．積和演算とは 2 変数の積に別の 1 変数を加算する演算でディジタルフィルタなどで多用される演算である．これを効率的に計算するのが MAC 演算器で，乗算の部分積加算に 1 変数の加算を組み入れることで全体の桁上げ伝播時間を 2 変数の乗算と同等にする[1)]．

同様な専用演算に差積演算がある．差積演算とは 2 変数の差に別の 1 変数を乗算する演算で，FFT のバタフライ演算[2)] などで用いられる．上述の MAC 演算器とは異なり乗算の部分積を決定するためには 2 変数の差の結果が必要であり，結果として 2 変数減算と 2 変数乗算それぞれの桁上げ遅延の合計が全体の遅延となってしまう問題があった．

この問題に対し本稿ではセレクタ論理を用いることで部分積を高速に生成する手法を提案する．セレクタ論理とはセレクタ回路を用いた桁上げ伝播省略手法である．本稿では差積演算をビットレベル式変形[3)] することでセレクタ回路に帰結できる項を見つけ出す．ビットレベル式変形とは各変数を $-x^{n-1}2^{n-1}+\sum_{i=0}^{n-2}x_i2^i$ の形式で計算し，各項を 2 の指数が等しい項でまとめる変形である．差積演算の部分積をビットレベル式変形によって表現し，セレクタ論理に帰着することで桁上げ伝播遅延を削減する手法を提案する．しかし差積演算にセレクタ論理を適用しただけでは複数の負の項が残り，符号拡張のための変数項が複数発生する．これは部分積加算回路への入力を増加させ結果として構成面積，遅延時間が増加する．そこで符号拡張を必要とする負の項を，セレクタ論理が適用できる形にまとめることでこの問題を解決する．

差積演算は主に FFT のバタフライ演算に用いられている．FFT は無線，動画像処理の分野で広く応用される技術の一つである．近年 DVB-T 規格で OFDM が採用されるなど，高サンプル数の FFT 演算回路が必要とされている[4)]．FFT 演算は複素加算と複素乗算を並列実行するバタフライ演算を繰り返して計算する．しかし高サンプル数の FFT 演算回路は複素乗算を伴うバタフライ演算回数が増加し，演算時間が延びる問題がある．これに対し，高

†1 早稲田大学大学院基幹理工学研究科情報理工学専攻
Dept. of Computer Science and Engineering, Waseda University
†2 大日本印刷株式会社電子モジュール開発センター
Electronic Module Development Center, Dai Nippon Printing Corporation

Vol.2009-SLDM-141 No.18
2009/10/16

基数化によって複素乗算を削減したりバタフライ演算器を多数直列するパイプライン構成[5]による高速化が提案されている．しかしこれまでバタフライ演算器では既存の乗・加算器を用いるのが一般的であった．バタフライ演算専用回路に関する研究としては V. D. Lecce らの手法[6]などがあるが，これらは演算回路の小面積化を狙ったものである．そこで，本稿では上述したセレクタ論理に基づく差積演算回路をバタフライ演算の差積演算に応用することで高速な Radix-2，4 バタフライ演算器を設計する[⋆1]．

2. セレクタ論理帰着型演算回路

本節では演算式をビットレベル式変形してセレクタ論理を適用するビットレベル処理手法を取り上げる．

2.1 セレクタ論理

1 ビットのブール変数 a, b, c, d があるとき，セレクタ論理は a, b を被選択信号，c を選択信号，d を出力とすると次の式で表せる．

$$d = a\bar{c} + bc$$

これは AND 演算 2 つと OR 演算 1 つの組み合わせであるが遅延時間，回路面積ともに AND や OR 一つ分と同程度で実現できる[7]．また，この演算の値域を求めると

$$\max|a\bar{c} + bc| - \min|a\bar{c} + bc| \leq 1$$

となり，その値域は 2 を越えず 2 進数の桁上げ伝播遅延が発生しない．ビットレベル式変形で数値演算をビットレベルに分解しセレクタ論理を適用することで桁上げ伝播遅延を削減できる．

なお，セレクタ論理は 3 入力演算であるのに対しフルアダーも 3 入力の演算であり全ての四則演算で多用されるが出力の値域が 2 を超えるため桁上げ伝播が発生してしまう．

2.2 差積演算のビットレベル式変形

n ビット，2 の補数で表現された整数 a, b, c について差積演算 $(a-b)c$ をビットレベル式変形し，セレクタ論理に帰着させることを考える．各変数は

$$a = [a_{n-1}a_{n-2}\cdots a_1a_0]_b = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \tag{1}$$

$$b = [b_{n-1}b_{n-2}\cdots b_1b_0]_b = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \tag{2}$$

$$c = [c_{n-1}c_{n-2}\cdots c_1c_0]_b = -c_{n-1}2^{n-1} + \sum_{i=0}^{n-2} c_i 2^i \tag{3}$$

とする．a_k, b_k, c_k $(k = 0, \cdots, n-1)$ は各桁のブール変数である．2 の補数表現で $-c$ は次のように示せる．

$$-c = -\overline{c_{n-1}}2^{n-1} + \sum_{i=0}^{n-2} \overline{c_i} 2^i + 1.$$

⋆1 なお，セレクタ論理適用型 Radix-2 バタフライ演算器は文献[8),9)]で提案されている．

よって，$(a-b)c$ は次のように表せる．

$$\begin{aligned}
&(a-b)c = ac + b(-c)\\
&= \left(-a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i\right)\left(-c_{n-1}2^{n-1} + \sum_{i=0}^{n-2} c_i 2^i\right) + \left(-b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i\right)\\
&\left(-\overline{c_{n-1}}2^{n-1} + \sum_{i=0}^{n-2} \overline{c_i} 2^i + 1\right)\\
&= \sum_{i=0}^{n-2}\sum_{j=0}^{n-2}(a_ic_j + b_i\overline{c_j})2^{i+j} + a_{n-1}2^{n-1}\left(-\sum_{i=0}^{n-2} c_i 2^i\right) + c_{n-1}2^{n-1}\left(-\sum_{i=0}^{n-2} a_i 2^i\right)\\
&+b_{n-1}2^{n-1}\left(-\sum_{i=0}^{n-2} \overline{c_i} 2^i\right) + \overline{c_{n-1}}2^{n-1}\left(-\sum_{i=0}^{n-2} b_i 2^i\right) - b_{n-1}2^{n-1} + \sum_{i=1}^{n-2} b_i 2^i\\
&+(a_{n-1}c_{n-1} + b_{n-1}\overline{c_{n-1}})2^{2n-2}.
\end{aligned} \tag{4}$$

ここで 2 の補数の関係式から

$$-\sum_{i=0}^{n-2} c_i 2^i = -1\cdot 2^{n-1} + \sum_{i=0}^{n-2} \overline{c_i} 2^i + 1 \tag{5}$$

$$-\sum_{i=0}^{n-2} a_i 2^i = -1\cdot 2^{n-1} + \sum_{i=0}^{n-2} \overline{a_i} 2^i + 1 \tag{6}$$

$$-\sum_{i=0}^{n-2} b_i 2^i = -1\cdot 2^{n-1} + \sum_{i=0}^{n-2} \overline{b_i} 2^i + 1 \tag{7}$$

$$-\sum_{i=0}^{n-2} \overline{c_i} 2^i = -1\cdot 2^{n-1} + \sum_{i=0}^{n-2} c_i 2^i + 1 \tag{8}$$

を代入すると次式が得られる．

$$\begin{aligned}
\text{式 (4)} = &\sum_{i=0}^{n-2}\sum_{j=0}^{n-2}(a_ic_j + b_i\overline{c_j})2^{i+j} + \sum_{i=0}^{n-2} b_i 2^i + \sum_{i=0}^{n-2}\{(b_{n-1}c_i + a_{n-1}\overline{c_i})\\
&+ (\overline{a_i}c_{n-1} + \overline{b_i}\overline{c_{n-1}})\Big\}2^{i+n-1} + (a_{n-1}c_{n-1} + b_{n-1}\overline{c_{n-1}} - a_{n-1}\\
&-c_{n-1} - b_{n-1} - \overline{c_{n-1}})2^{2n-2} + a_{n-1}2^{n-1} + (c_{n-1} + \overline{c_{n-1}})2^{n-1}.
\end{aligned} \tag{9}$$

式 (9) の係数 2^{2n-2} の項について考える．正の項 2 個は Wallace Tree[10] の 2^{2n-2} 桁目への 2 ビットの入力となる．しかし負の項 4 個は符号処理のため Wallace Tree の 2^{2n-1} 桁

Vol.2009-SLDM-141 No.18
2009/10/16

目に 4 ビット，2^{2n-2} 桁目に 4 ビットの入力となる．Wallace Tree への入力ビット数が増加すると遅延時間や面積が増加する．そこで式 (9) の係数 2^{2n-2} の項を次のように変形し正の項 2 個と負の項 4 個を正の項 2 個と負の項 1 個にまとめる．

$$\begin{aligned}
&(a_{n-1}c_{n-1} + b_{n-1}\overline{c_{n-1}} - a_{n-1} - c_{n-1} - b_{n-1} - \overline{c_{n-1}}) \\
=\ &(a_{n-1}c_{n-1} - c_{n-1} - a_{n-1} + 1) - 1 + (b_{n-1}\overline{c_{n-1}} - b_{n-1} - \overline{c_{n-1}} + 1) - 1 \\
=\ &(\overline{a_{n-1}}\ \overline{c_{n-1}}) + (\overline{b_{n-1}}c_{n-1}) - 2. \qquad (10)
\end{aligned}$$

式 (10) を式 (9) に代入した結果を示す．

$$\begin{aligned}
\text{式 (4)} = &\sum_{i=0}^{n-2}\sum_{j=0}^{n-2}(\underline{a_i c_j + b_i \overline{c_j}})2^{i+j} && (11)\\
&+\{\underline{(\overline{a_{n-1}}\ \overline{c_{n-1}}) + (\overline{b_{n-1}}c_{n-1})}\}2^{2n-2} && (12)\\
&+\sum_{i=0}^{n-2}\left\{\underline{(b_{n-1}c_i + a_{n-1}\overline{c_i})} + \underline{(\overline{a_i}c_{n-1} + \overline{b_i}\overline{c_{n-1}})}\right\}2^{i+n-1} && (13)\\
&+\sum_{i=0}^{n-2} b_i 2^i && (14)\\
&+a_{n-1}2^{n-1} + 2^{n-1} && (15)\\
&-2^{2n-1}. && (16)
\end{aligned}$$

以上より Wallace Tree への入力は式 (11) で表される項が $(2n^2 - 4n + 2)$ ビット，同様に式 (12) で 2 ビット，式 (13) で $4n - 4$ ビット，式 (14) で $n - 1$ ビット，式 (15) で 2 ビット，式 (16) は符号拡張されて 2 ビットとなり合計 $(2n^2 + n + 3)$ ビットとなる．ここで式 (11)，(12)，(13) の下線部はセレクタ論理に帰着できるからこれらをセレクタ回路で計算すれば Wallace Tree への入力を $(n^2 + n + 3)$ ビットまで削減でき，Wallace Tree の遅延と規模を削減できる．$(a - b)c$ は通常桁上げ処理を減算部と乗算部の 2 回に分けて行うが式 (11) を用いれば Wallace Tree を用いて 1 回の桁上げ処理で計算できる．差積演算をこのように設計することで従来の自動合成ツールと算術演算子を用いる設計に比べ遅延時間の削減が期待できる．

3. セレクタ論理帰着バタフライ演算器

FFT 演算では複素数 2，4 入力の加算と差積演算で構成される Cooley-Turkey 型の Radix-2，4 バタフライ演算アルゴリズムがよく用いられる[4),5),11)]．多くの FFT 用プロセッサでは加算と差積を並列に行うが差積が遅延時間のボトルネックとなる．本節では 2 節で示したセレクタ論理帰着型差積演算器を用いた Radix-2，4 バタフライ演算器を提案する．従来の算術演算子を用いて設計されたバタフライ演算器に対し差積演算高速化分の演算時間短縮が期待できる．

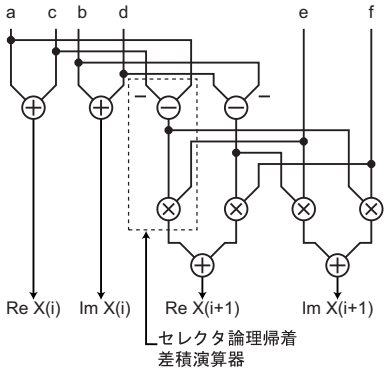

図 1 Radix-2 バタフライ演算
Fig. 1 Radix-2 Butterfly computation

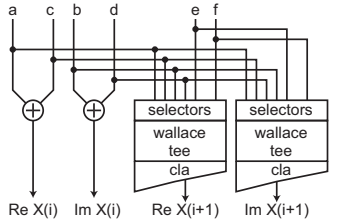

図 2 セレクタ論理型 Radix-2 バタフライ演算
Fig. 2 Radix-2 Butterfly computation by selector logic

3.1 Radix-2 バタフライ演算

入力系列が $x(s)$，出力系列が $X(k)$ で標本数 N の DFT 演算は次式で示される．

$$X(k) = \sum_{s=0}^{N-1} x(s)W_N^{sk} \quad \left(\text{ただし } W_N = e^{-j\frac{2\pi}{N}}\right). \qquad (17)$$

これを変形すると次式になる[12)]．

$$X(k) = \sum_{s=0}^{N/2-1}\left\{x(s) + (-1)^k x(s + \frac{N}{2})\right\}W_N^{sk}. \qquad (18)$$

k の偶数，奇数系列はそれぞれ $N/2$ 点の DFT でありこの変形を再帰的に繰り返すと 2 点 DFT となる．入力を $x(i), x(i+1)$，出力を $X(i), X(i+1)$，回転係数を W^k とすると

$$\begin{aligned}
X(i) &= x(i) + x(i+1) \\
X(i+1) &= \{x(i) - x(i+1)\}W^k \qquad (19)
\end{aligned}$$

と表せる．これを Radix-2 バタフライ演算と呼ぶ．j を虚数単位として，2 の補数で表現された n ビットの変数 a, b, c, d, e, f を用いて $x(i) = a + bj$，$x(i+1) = c + dj$，$W^k = e + fj$ として定義すると Radix-2 バタフライ演算は図 1 で表せる．

3.2 セレクタ論理を用いた Radix-2 バタフライ演算器の設計

図 1 の破線で囲まれた部分に注目するとこれは差積演算を構成しており，出力 $Re\ X(i+1)$ は 2 つの差積演算の出力の和であるので

2 節で示した手法を用いることで従来の算術演算子を用いた設計に対し高速な回路を設計できる．そこで差積演算部を 2 節と同様にセレクタ論理が適用できるようにビットレベル

Vol.2009-SLDM-141 No.18
2009/10/16

式変形する．$Re\ X(i+1)$ は次式で表現される．

$$
\begin{aligned}
&Re\ X(i+1) = (a-c)e + (d-b)f \\
&\sum_{i=0}^{n-2}\sum_{j=0}^{n-2}(a_i e_j + c_i\overline{e_j})2^{i+j} + \sum_{i=0}^{n-2} c_i 2^i + \sum_{i=0}^{n-2}\{(c_{n-1}e_i + a_{n-1}\overline{e_i}) \\
&+ (\overline{a_i}e_{n-1} + \overline{c_i e_{n-1}})\}\,2^{i+n-1} + \{(\overline{c_{n-1}}e_{n-1}) + (\overline{a_{n-1}e_{n-1}})\}2^{n-2} \\
&-2^{2n-1} + a_{n-1}2^{n-1} + 2^{n-1} \\
&+ \sum_{i=0}^{n-2}\sum_{j=0}^{n-2}(d_i f_j + b_i\overline{f_j})2^{i+j} + \sum_{i=0}^{n-2} b_i 2^i + \sum_{i=0}^{n-2}\Big\{(b_{n-1}f_i + d_{n-1}\overline{f_i}) \\
&+ (\overline{d_i}f_{n-1} + \overline{b_i f_{n-1}})\Big\}\,2^{i+n-1} + \{(\overline{b_{n-1}}f_{n-1}) + (\overline{d_{n-1}f_{n-1}})\}2^{n-2} \\
&-2^{2n-1} + d_{n-1}2^{n-1} + 2^{n-1} \\
=&\sum_{i=0}^{n-2}\sum_{j=0}^{n-2}\{(a_i e_j + c_i\overline{e_j}) + (d_i f_j + b_i\overline{f_j})\}2^{i+j} + \sum_{i=0}^{n-2}(c_i + b_i)2^i + \sum_{i=0}^{n-2}\{(c_{n-1}e_i + a_{n-1}\overline{e_i}) \\
&+(\overline{a_i}e_{n-1} + \overline{c_i}\ \overline{e_{n-1}}) + (b_{n-1}f_i + d_{n-1}\overline{f_i}) + (\overline{d_i}f_{n-1} + \overline{b_i}\ \overline{f_{n-1}})\}2^{i+n-1} \\
&+\{(\overline{c_{n-1}}e_{n-1}) + (\overline{a_{n-1}}\ \overline{e_{n-1}}) + (\overline{b_{n-1}}f_{n-1}) + (\overline{d_{n-1}}\ \overline{f_{n-1}})\}2^{2n-2} \\
&-2^{2n} + (a_{n-1} + d_{n-1})2^{n-1} + 2^n. \qquad (20)
\end{aligned}
$$

式 (20) は式 (11) と同様にセレクタ論理によって部分積を生成し，Wallace Tree へ入力することで計算できる．この演算器のブロック図を図 2 に示す．セレクタによって生成された部分積は Wallace Tree によって，その出力は CLA（Carry Look Ahead）[7] によって最終加算する．

3.3 Radix-4 バタフライ演算

式 (17) は次式のようにも表現できる．

$$X(k) = \sum_{s=0}^{N/4-1}\left\{x(s) + (-j)^k x(s+\frac{1}{4}N) + (-1)^k x(s+\frac{2}{4}N) + (j)^k x(s+\frac{3}{4}N)\right\}W_N^{sk}. \qquad (21)$$

$k \bmod 4$ が $0,1,2,3$ となる系列はそれぞれ $N/4$ 点の DFT であり Radix-2 バタフライ演算と同様にこの変形を再帰的に繰り返すと 4 点 DFT となる．そのときの入力を $x(i), x(i+1), x(i+2), x(i+3)$，出力を $X(i), X(i+1), X(i+2), X(i+3)$，回転係数を $W^i, W^{i+1}, W^{i+2}, W^{i+3}$ とするとこの 4 点 DFT は次式で表せる．

$$
\begin{aligned}
&X(i+m) = \{x(i) + (-j)^m x(i+1) + (-1)^m x(i+2) + (j)^m x(i+3)\}W^{km} \\
&(m = 0,1,2,3). \qquad (22)
\end{aligned}
$$

これを Radix-4 バタフライ演算と呼ぶ．2 の補数で表現された n ビットの変数

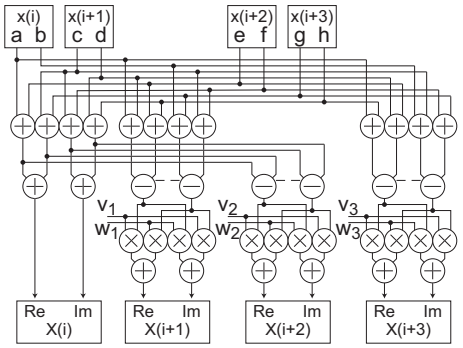

図 3 Radix-4 バタフライ演算
Fig. 3 Radix-4 Butterfly computation

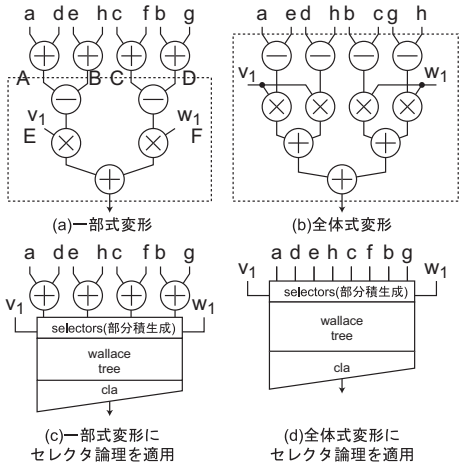

図 4 セレクタ論理型 Radix-4 バタフライ演算
Fig. 4 Radix-4 Butterfly computation by selector logic

$a, b, c, d, e, f, g, h, v_1, v_2, w_1, w_2$ を用いて各変数を

$$
\begin{aligned}
&x(i) = a + bj \quad x(i+1) = c + dj \quad x(i+2) = e + fj \quad x(i+3) = g + hj \\
&W^k = v_1 + w_1 j \quad W^{2k} = v_2 + w_2 \quad W^k = v_1 + w_1 j \qquad (23)
\end{aligned}
$$

として定義すると Radix-4 バタフライ演算は図 3 で表せる．また，$Re\ X(i+1)$ は図 4(a) で表せる．

3.4 セレクタ論理を用いた Radix-4 バタフライ演算器の設計

図 4(a) の破線で囲まれた部分に注目するとこれは Radix-2 バタフライ演算器と同じ構成をしており 3.2 項同様にビットレベル式変形によってセレクタ論理に帰着できる．

まず図 4(a) の破線部より上側の演算を行う必要がある．2 の補数で表現された $n+1$ ビットの変数 A を

$$A = [A_n A_{n-1} \cdots A_1 A_0]_b = -A_n 2^n + \sum_{i=0}^{n-1} A_i 2^i \qquad (24)$$

とおく．さらに B, C, D についても同様に定義する．また，2 の補数で表現された n ビットの変数 E, F を式 (1) と同様に定義し，$A = a+d$，$B = e+h$，$C = c+f$，$D = b+g$，$E = v_1$，$F = w_1$ とおき破線部より上側の演算を実行する．すると図 4(a) に従い $Re\ X(i+1)$ は次式のように変形できる．

$$Re\ X(i+1) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-2}\{(A_i E_j + C_i\overline{E_j}) + (D_i F_j + B_i\overline{F_j})\}2^{i+j} + \sum_{i=0}^{n-1}(C_i + B_i)2^i$$

Vol.2009-SLDM-141 No.18
2009/10/16

$$+\sum_{i=0}^{n-2}\{(C_nE_i+A_n\overline{E_i})+(B_nF_i+D_n\overline{F_i})\}2^{i+n}$$
$$+\sum_{i=0}^{n-1}\{(\overline{A_i}E_{n-1}+\overline{C_i}\ \overline{E_{n-1}})+(\overline{D_i}F_{n-1}+\overline{B_i}\ \overline{F_{n-1}})\}2^{i+n-1}$$
$$+\{(\overline{C_{n-1}}E_{n-1})+(\overline{A_{n-1}}\ \overline{E_{n-1}})+(\overline{B_{n-1}}F_{n-1})+(\overline{D_{n-1}}\ \overline{F_{n-1}})\}2^{2n-1}$$
$$-2^{2n+1}+(A_{n-1}+D_{n-1})2^n+2^n. \quad (25)$$

式 (25) は式 (11) と同様にセレクタ論理によって部分積を生成し，Wallace Tree へ入力することで計算できる．この演算器のブロック図を図 4(c) に示す．これを Radix-4 バタフライ演算の**一部式変形**と呼ぶことにする．

このままでは初段の加算の桁上げ処理待ち時間が残っているのでさらに以下のように式変形する．

$$Re\ X(i+1)=\sum_{i=0}^{n-2}\sum_{j=0}^{n-2}\{(a_iV_j+e_i\overline{V_j})+(d_iV_j+h_i\overline{V_j})+(b_iW_j+c_i\overline{W_j})$$
$$+(g_iW_j+f_i\overline{W_j})\}2^{i+j}+\sum_{i=0}^{n-2}\{(e_{n-1}V_i+a_{n-1}\overline{V_i})+(\overline{a_i}V_{n-1}+\overline{e_i}\overline{V_{n-1}})$$
$$+(h_{n-1}V_i+d_{n-1}\overline{V_i})+(\overline{d_i}V_{n-1}+\overline{h_iV_{n-1}})+(c_{n-1}W_i+b_{n-1}\overline{W_i})$$
$$+(\overline{b_i}W_{n-1}+\overline{c_i}\overline{W_{n-1}})+(f_{n-1}W_i+g_{n-1}\overline{W_i})+(\overline{g_i}W_{n-1}+\overline{f_i}\ \overline{W_{n-1}})\}2^{i+n-1}$$
$$+\sum_{i=0}^{n-2}(e_i+h_i+c_i+f_i)2^i-2^{2n+1}+\{(\overline{e_{n-1}}V_{n-1}+\overline{a_{n-1}}\overline{V_{n-1}})+(\overline{h_{n-1}}V_{n-1}$$
$$+\overline{d_{n-1}}\ \overline{V_{n-1}})+(\overline{c_{n-1}}W_{n-1}+\overline{b_{n-1}}\ \overline{W_{n-1}})+(\overline{f_{n-1}}W_{n-1}+\overline{g_{n-1}}\overline{W_{n-1}})\}2^{2n-2}$$
$$+2^{n+1}+(a_{n-1}+d_{n-1}+b_{n-1}+g_{n-1})2^{n-1}. \quad (26)$$

式 (26) は式 (11) と同様にセレクタ論理によって部分積を生成し，Wallace Tree へ入力することで計算できる．これより図 4(d) の構成を得る．これは 4 つの差積演算器出力の和である．これを Radix-4 バタフライ演算の**全体式変形**とする．この構成では図 4(d) に示すように初段の加算が不要となっている．この回路構成は部分積項が約 2 倍に増加して Wallace Tree の面積を増大させる欠点を持つが式 (25) の構成に対してさらに高速化できる．

4. ゲート段数，トランジスタ数見積もり

本節ではセレクタ論理型差積演算器と算術演算子を用いた設計とで必要となるゲート段数，トランジスタ数を比較する．具体例として入力 8 ビットの差積演算器について考える．算術演算子の加算，減算は CLA，乗算は Wallace Tree による部分積加算と CLA による最終加算で構成されると仮定する．CLA と Wallace Tree の構成は文献 7)，9) に拠る．各ゲートのトランジスタ数を表 1 に示す．

表 1 各ゲートのトランジスタ数
Table 1 The number of transistors in each logic gate

ゲート	Tr 数	ゲート	Tr 数
Inverter	2	AND	6
XOR	8	3 入力 AO	10
Half Adder	14	4 入力 AO	10
Full Adder	32	Selector	10

4.1 ゲート段数比較

本節でゲート段数とは基本セル，複合セルを一段とし，インバータはカウントしない．

4.1.1 算術演算子

算術演算子を用いた場合，8 ビット CLA，8×9 ビット Wallace Tree，Wallace Tree 最終加算用の 11 ビット加算器が必要である．CLA は初段の PG 計算に 1 段，PG の生成に 4 段，S の生成に XOR1 段を必要とする．よって 1＋4＋1＝6 段をカウントする．Wallace Tree は初段の部分積生成に 1 段，部分積加算に Full Adder4 段を必要とする．Full Adder は基本セル 1 段と複合セル 1 段の計 2 段を必要とするため Wallace Tree 全体では 1＋4×2＝9 段としてカウントする．Wallace Tree の最終加算は 11 ビット CLA で 1＋5＋1＝7 段必要である．算術演算子で構成した場合のゲート段数は 6＋9＋7＝22 段と見積もることができる．

4.1.2 セレクタ論理適用型差積演算

セレクタ論理を用いた場合部分積生成にセレクタ 1 段，Wallace Tree の部分積加算に 4 段必要である．最終加算 CLA は 12 ビットとなり 1＋5＋1＝7 段必要となる．よってセレクタ論理型差積演算器は 1＋4×2＋7＝16 段必要と見積もることができる．またゲート段数比は算術演算子構成に対し 16/22＝0.73 となる．

4.2 トランジスタ数比較

本節では各回路のトランジスタ数を表 1 として差積演算回路のトランジスタ数を見積もる．

4.2.1 算術演算子

8 ビット CLA では初段の PG 計算に Half Adder が 8 個，PG 生成に 4 入力 PG 計算器が 4 個，3 入力 PG 計算器が 8 個，そして最終的に S を計算するのに XOR を 8 個必要とする．よってトランジスタは 8×14＋4×16＋8×10＋8×8＝320 個必要である．8×9 ビット Wallace Tree は部分積生成に AND 回路 72 個，部分積加算に Full Adder47 個，Half Adder11 個を必要とする．よってトランジスタは 72×6＋47×32＋11×14＝2090 個必要である．Wallace Tree 最終加算は 11 ビット CLA となり，Half Adder11 個，4 入力 PG 計算器 5 個，3 入力 PG 計算器 10 個，XOR11 個を必要とする．よってトランジスタは 11×14＋5×16＋10×10＋11×8＝422 個必要である．以上から，算術演算子で構成した場合は 320＋2090＋422＝2832 個のトランジスタが必要と見積もることができる．

4.2.2 セレクタ論理適用型差積演算

セレクタ論理適用型差積演算器を用いた場合，部分積生成にセレクタ回路 64 個，部分積加算に Full Adder46 個，Half Adder12 個を必要とする．よってトランジスタは $64 \times 10 + 46 \times 32 + 12 \times 14 = 2280$ 個必要である．Wallace Tree の最終加算は 12 ビット CLA となり，Half Adder12 個，4 入力 PG 計算器 7 個，3 入力 PG 計算器 11 個，XOR12 個を必要とする．よってトランジスタは $12 \times 14 + 7 \times 16 + 11 \times 10 + 12 \times 8 = 486$ 個必要である．以上から．セレクタ論理適用型差積演算器で構成した場合は $2280 + 486 = 2766$ 個のトランジスタが必要と見積もることができる．またトランジスタ数比は算術演算子構成に対し $2766/2832 = 0.98$ となる．

5. 実 装 結 果

本節では差積演算，Radix-2 バタフライ演算回路，Radix-4 バタフライ演算回路について算術演算子構成とセレクタ論理を用いた構成を論理合成し性能を評価する．入力ビット長は 8，16 ビットとした．ハードウェア記述には Verilog-HDL を用い，論理合成には Synopsys 社の Design Compiler を遅延時間が最小となるように設定して用いた．セルライブラリには STARC[⋆1] (CMOS90[nm]) を用いた．

合成結果を表 2 に示す．遅延時間は入出力信号間の最大遅延時間である．セレクタ論理を用いた設計は従来の算術演算子による手法に比べ高速である．差積演算を必要とするアルゴリズムでは本手法を適用することで高速化できる．バタフライ演算を繰り返し用いる FFT アルゴリズムにセレクタ論理型演算器を用いることで FFT の高速化の要求に対応することができる．

表 2 従来手法とセレクタ論理応用回路の性能比較
Table 2 Performance comparison of conventional structure and proposed structure

演算器	設計	遅延時間 (ns)	ゲート数 (gates)
差積 8 ビット	算術演算子	2.75(1)	2573(1)
	提案積演算器	1.83(0.67)	2318(0.90)
差積 16 ビット	算術演算子	4.89(1)	9063(1)
	提案差積演算器	3.21(0.66)	9152(1.01)
Radix-2 バタフライ演算 8 ビット	算術演算子	3.12(1)	5657(1)
	提案バタフライ演算器	2.25(0.72)	4812(0.85)
Radix-2 16 ビット	算術演算子	5.42(1)	19069(1)
	提案バタフライ演算器	3.63(0.67)	18622(0.98)
Radix-4 バタフライ演算 8 ビット (一部式変形)	算術演算子	3.51(1)	7075(1)
	提案バタフライ演算器	2.99(0.85)	6335(0.90)
Radix-4 バタフライ演算 16 ビット (一部式変形)	算術演算子	5.85(1)	21930(1)
	提案バタフライ演算器	4.74(0.81)	21569(0.98)
Radix-4 バタフライ演算 8 ビット (全体式変形)	算術演算子	3.32(1)	11761(1)
	提案バタフライ演算器	2.62(0.79)	9708(0.83)
Radix-4 バタフライ演算 16 ビット (一部式変形)	算術演算子	5.63(1)	39017(1)
	提案バタフライ演算器	3.96(0.70)	37502(0.96)

6. む す び に

本稿では差積演算の部分積項をビットレベル式変形によって減算部の桁上げ伝播を省略し部分積を直接生成することで高速化する手法を示した．さらに部分積項をセレクタ論理を用いて半分にし Wallace Tree の規模を増大させることなく高速な差積演算回路を構成できることを示した．これは差積演算を用いるアルゴリズム一般に応用できる．3 節では具体例として FFT の Radix 演算に応用することで FFT の高速化要求に対応できることを示した．

参 考 文 献

1) K. F. Pang, H.-W. Song, R. Sexton, and P.-H. Ang, “Generation of high speed CMOS multiplier-accumulators,” in *Proc. ICCD*, pp.217–220 (1988).
2) J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complexfourier series,” *Mathematics of Computation*, vol.19, pp.297–301 (1965).
3) 外村元伸, “実設計に応用できる演算回路のスキルを身につけよう，” in *Design Wave Magazine*, May, pp.39–48 (2006).
4) C. L. Wey, W.-C. Tang, and S. Y. Lin, “Efficient VLSI implementation of memory-based FFT processors for DVB-T applications,” in *Proc. ISVLSI*, pp.98–106 (2007).
5) Y.-T. Lin, P.-Y. Tsai, and T.-D. Chiueh, “Low-power variable-length fast Fourier transform processor,” in *Proc. Comput. Digit. Tech.*, vol. 152, no. 4, pp. 299–506 (2005).
6) V. D. Lecce and D. E Sciascio, “A VLSI implementation of a novel bit-serial butterfly processor for FFT,” in *Proc. CompEuro ’91. Advanced Computer Technology, Reliable Systems and Applications*, pp.875–879 (1991).
7) N. Weste and D. Harris, *CMOS VLSI Design*: Addison Wesley (2004).
8) 名村　健，戸川　望，柳澤政生，大附辰夫，外村元伸, セレクタ論理を用いたバタフライ演算器の設計，信学技報，VLD2008-5, (2008).
9) 名村　健，戸川　望，柳澤政生，大附辰夫，外村元伸, ビットレベル式変形によるセレクタ帰着型バタフライ演算器の設計と評価，信学技報，VLD2008-52, (2008).
10) G. W. Bewick, “Fast multiplication algorithms and implementation,” Ph.D. dissertation, University of Stanford (1994).
11) C.-P. Hung, S.-G. Chen, and K.-L. Chen, “Design of an efficient variable-length FFT processor,” in *Proc. ISCAS*, vol.2, pp.833–836 (2004).
12) A. Peled and B. Liu, *Digital Signal Processing*: John Wiley & Sons (1976).

⋆1 START[90nm] ライブラリは東京大学大規模集積システム設計研究センターを通じ，株式会社半導体理工研究センター (STARC) との協同で開発されたものである．