計測自動制御学会東北支部 第167回研究集会 (1997.5.16)
資料番号 167-8

分散演算を用いた適応フィルタのVLSI実現

VLSI Implementation of LMS Adaptive Filters Using Distributed Arithmetic

○豊田真嗣*，恒川佳隆*，三浦　守*

○Shinji Toyoda*, Yoshitaka Tsunekawa*, Mamoru Miura*

*岩手大学

*Iwate University

キーワード： 適応フィルタ (Adaptive Filters)，LMSアルゴリズム (Least Mean Square algorithm)，
DA適応フィルタ (Adaptive Filter using Distributed Arithmetic)，
適応関数空間 (Adaptive Function Space)，VLSI評価 (VLSI evaluation)

連絡先： 〒020　盛岡市上田4-3-5 岩手大学 工学部 情報工学科 恒川研究室
豊田真嗣，Tel.: (019)621-6468，Fax.: (019)623-5491，E-mail: shinji@cis.iwate-u.ac.jp

1. はじめに

近年，適応フィルタの応用範囲が広くなり，様々な分野において，その必要性が高まっている．適応フィルタの代表的な応用例としては，エコーキャンセラ，ノイズキャンセラ，自動等化器等が挙げられる．適応フィルタの必要性が高まるとともに，より性能の高い適応フィルタが望まれている．適応フィルタとして，十分な性能を得るためには，高次の適応フィルタが必要となる．しかしこれまでに，高次の適応フィルタのハードウェア実現及び，収束特性に対する検討はあまり行われてこなかった．

現在，適応信号処理の分野において，最も広く用いられている適応アルゴリズムとして，LMSアルゴリズム (Least Mean Square algorithm) が挙げられる．このアルゴリズムは，ハードウェア実現が容易であることと，少ない演算量の割には良好な収束特性を持つという特長がある．しかし，ハードウェア実現を考慮した場合，クロックレートを高い値に保つためには，DLMSアルゴリズム (Delayed LMS algorithm) を用いて[1)2)3)]，タップ毎にパイプライン処理を行う必要があり，タップ数分の乗算器が必要となる．そのため，高次での実現を考慮した場合，膨大なハードウェア量が必要となる．更に，タップ数分ディレイが増加するので，収束特性の劣化，滞在時間の増大などの悪影響が生じる．

そこで，本研究では，適応フィルタを構成する際に，ハードウェア量を減少させることを目的として，我々がこれまでに提案してきた分散演算 (Distributed Arithmetic) を用いたFIRフィルタの構成法を適用した[4)]．この構成法の大きな特長として，(1) 処理時間が次数によらず，語長のみに依存すること，(2) 乗算器を用いずにハードウェア実現することができるので，大幅にハードウェア量

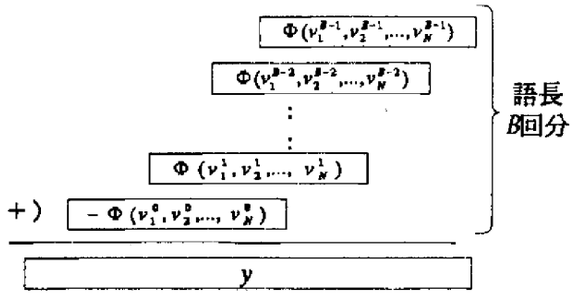


Fig. 1　分散演算の概念図

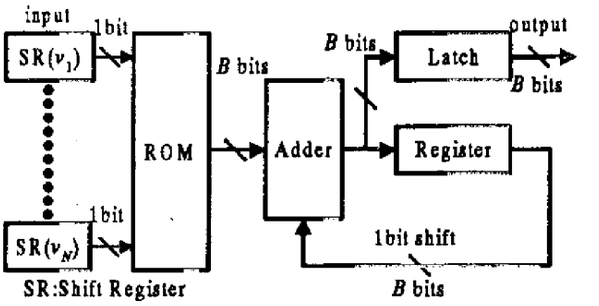


Fig. 2　分散演算の基本構成

を減少させること，(3)滞在時間に関しても非常に小さく抑えることができること，などが挙げられる．これらの点から，分散演算を用いた場合の適応フィルタの構成法は，ハードウェアの効率性を考慮した場合，非常に有効な構成法であることがいえる．

本研究では，この分散演算を用いたFIRフィルタの構成法の特長を生かした，適応フィルタの構成法を提案する．さらに，その構成をVLSI設計システムPARTHENONを用いて設計し，性能評価を行う．その結果，高次な場合においても，ハードウェア量を抑えた上で，高速性を一定に保つことが可能となることをVLSI評価によって検討する．

2.　分散演算を用いた適応フィルタ

本章では，まず，分散演算の原理について述べ，次に，その分散演算を用いた適応フィルタ(以下，DA適応フィルタと呼ぶ)の原理について述べる．

2.1　分散演算

分散演算は，定係数の内積演算をテーブルルックアップによって実現する計算手法である．

ここで，項数Nの係数ベクトル$\boldsymbol{a}=(a_1,\ldots,a_N)$と変数ベクトル$\boldsymbol{v}=(v_1,\ldots,v_N)$との内積

$$y=\boldsymbol{a}\boldsymbol{v}=\sum_{i=1}^{N}a_iv_i \qquad (1)$$

を考える．ただし，$-1\le v_i<1$で，v_iはBビットの固定小数点形の2の補数表示である．つまり，

$$v_i=-v_i^0+\sum_{k=1}^{B-1}2^{-k}v_i^k \qquad (2)$$

と表される．ここで，v_i^kはv_iのkビット目の値で0または1である．(2)式を(1)式に代入すれば，内積演算$\boldsymbol{a}\boldsymbol{v}$は次式で示される．

$$y=-\Phi(v_1^0,\ldots,v_N^0)+\sum_{k=1}^{B-1}2^{-k}\Phi(v_1^k,\ldots,v_N^k) \qquad (3)$$

ただし，関数Φは

$$\Phi(v_1^k,\ldots,v_N^k)=\sum_{i=1}^{N}a_iv_i^k \qquad (4)$$

である．

式(3)からわかるように，分散演算を用いた場合，フィルタ出力を求める手順として，まず，関数Φを実現したテーブルを生成し，入力のビットパターンによって，関数Φのテーブルから内積演算の結果を読み出す．次に得られた値を1ビット右シフトして加え合わせる．この操作を語長回行うことによってフィルタ出力を求めることができる．この概念図を図1に示す．

図2にハードウェア実現した際の構成を示す．関数Φのテーブルは$(v_1^k,\ldots,v_N^k)$をアドレスとするROMで実現でき，右シフト加算は加算器とレジスタによって実現できる．

これから分かるように，原理的には処理時間が次数Nに依存せず，語長Bのみに依存する構成が実現可能である．また，乗算器を用いずにフィル

タ出力を求めることができるので，大幅にハードウェア量を減少することが可能となる．このように分散演算は高次のハードウェア実現を行う際に非常に有効な手法である．

2.2 DA適応フィルタの更新式の導出

これまでにLMSアルゴリズムの係数更新式に分散演算を用いることによって，DA適応フィルタの更新式が導かれてきた[5]．このDA適応フィルタの更新式は，オフセットバイナリ形式に基づいて信号の符号化を行っていた．しかし，オフセットバイナリ形式は，符号ビットを用いない表現形式であるため，取り得る信号値の範囲が狭いという欠点がある．

本節では，このLMSアルゴリズムに基づく更新式を2の補数形式によって式変形することにより，DA適応フィルタの更新式を導出する．

準備として，前節で述べた分散演算の表現式を，ベクトル形式を用いて簡潔に記述することにする．

次数Nの入力信号ベクトルを
$\boldsymbol{S}(k)=[s(k),s(k-1),\ldots,s(k-N+1)]^T$，係数ベクトルを$\boldsymbol{W}(k)=[w_0(k),w_1(k),\ldots,w_{N-1}(k)]^T$
とするとフィルタ出力を求める式は次のように表される．

$$y(k)=\boldsymbol{F}^T\boldsymbol{A}(k)^T\boldsymbol{W}(k) \tag{5}$$

$$\boldsymbol{S}(k)=\boldsymbol{A}(k)\boldsymbol{F} \tag{6}$$

ここで，アドレスマトリクス$\boldsymbol{A}(k)$を

$$\boldsymbol{A}(k)=$$

$$\begin{bmatrix} -b_0(k) & -b_0(k-1) & \cdots & -b_0(k-N+1) \\ b_1(k) & b_1(k-1) & \cdots & b_1(k-N+1) \\ \vdots & \vdots & & \vdots \\ b_{B-1}(k) & b_{B-1}(k-1) & \cdots & b_{B-1}(k-N+1) \end{bmatrix}^T$$

スケーリングベクトル$\boldsymbol{F}$を

$$\boldsymbol{F}=[2^0,2^{-1},\ldots,2^{-(B-1)}]^T$$

とする．

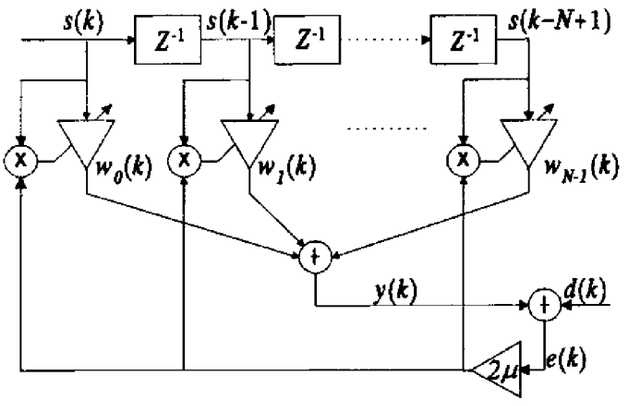


Fig. 3　LMSによる適応フィルタのブロック図

次に，2の補数形式によるLMSアルゴリズムの更新式からDA適応フィルタの更新式の導出の過程を示す．まず，LMSアルゴリズムによる係数更新式は，次のように表される．また，LMSによる適応フィルタのブロック図を図3に示す．

$$\boldsymbol{W}(k+1)=\boldsymbol{W}(k)+2\mu e(k)\boldsymbol{S}(k) \tag{7}$$

誤差信号

$$e(k)=d(k)-y(k) \tag{8}$$

式(7)の両辺に$\boldsymbol{A}^T(k)$を左からかけることにより，式(9)のようになる．さらに，$\boldsymbol{A}^T(k)\boldsymbol{W}(k)$を関数$\boldsymbol{P}(k)$と定義することによって，式(10)，(11)が得られる．

$$\boldsymbol{A}^T(k)\boldsymbol{W}(k+1)=\boldsymbol{A}^T(k)\{\boldsymbol{W}(k)+2\mu e(k)\boldsymbol{A}(k)\boldsymbol{F}\} \tag{9}$$

$$p_i(k+1)=p_i(k)+2\mu\boldsymbol{A}^T(k)\boldsymbol{A}(k)e(k)2^{-i} \tag{10}$$

(i=1,2,...,B-1)

$$p_0(k+1)=p_0(k)-2\mu\boldsymbol{A}^T(k)\boldsymbol{A}(k)e(k)2^0 \tag{11}$$

($p_0(k)$は入力信号が符号ビット時の関数)

ここで，DA適応関数ベクトルを

$$\boldsymbol{P}(k)=[p_0(k),p_1(k),\ldots,p_{B-1}(k)]^T$$

のように定義する．

0000	0
0001	$w_3(k)$
0010	$w_2(k)$
⋮	⋮
1110	$w_0(k)+w_1(k)+w_2(k)$
1111	$w_0(k)+w_1(k)+w_2(k)+w_3(k)$

Fig. 4　適応関数空間の概念図

関数$\boldsymbol{P}(k)$について述べる．適応フィルタにおけるLMSによる更新アルゴリズムの場合は，係数$\boldsymbol{W}(k)$を更新するのに対して，DA適応フィルタの更新アルゴリズムでは，関数$\boldsymbol{P}(k)$(以下，適応関数空間と呼ぶ)の値を更新することになる．概念図を図4に示す．

式(9)の入力信号行列$\boldsymbol{A}$とその転置行列$\boldsymbol{A}^T$の乗算の部分は，この入力信号が平均0の白色信号と仮定されるなら，対角要素は平均Nとなり，それ以外は0となる[5]．よって，$\boldsymbol{A}^T\boldsymbol{A}$の乗算を統計的に$N$に置き換えることができるので，更新値はシフト操作のみで求めることができる．従って，行列の乗算を行わずに済むので，大幅に計算量を減少することができる．最終的な更新式を以下に示す．

$$p_i(k+1) = p_i(k) + 2\mu N e(k) 2^{-i} \qquad (12)$$

(i=1,2,...,B-1)

$$p_0(k+1) = p_0(k) - 2\mu N e(k) 2^0 \qquad (13)$$

これがDA適応フィルタの関数更新式となる．このDA適応フィルタのブロック図を図5に示す．

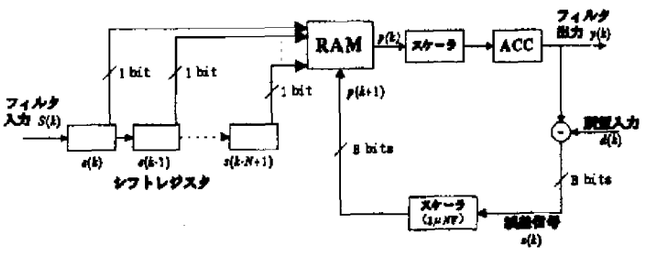


Fig. 5　DA適応フィルタのブロック図

3.　適応関数空間の分割化

DA適応フィルタでは，適応関数空間の入力アドレスのビット数が次数Nと等しいため，適応関数空間の大きさが2^Nとなる．従って，次数の増加とともに適応関数空間が非常に大きくなるため，適応関数空間の更新に時間がかかり，収束特性が大幅に劣化する．

この問題に対する解決策として，適応関数空間の分割化を行う手法が提案されてきた[6][7]．この手法では，適応関数空間を分割化することによって，更新する空間を小さくして，収束特性の劣化を減少している．具体的には，入力のビット数がNビットである適応関数空間を，M個の適応関数空間に分割し，それぞれの関数適応空間の入力のビット数を$R(=N/M)$(N:次数,M:適応関数空間の分割数)ビットとする．これによって，各適応関数空間の大きさを2^Rとし，更新に要する時間を減少させる．この適応関数空間の構成をマルチメモリブロック構造(Multi-Memory Block Structure)と呼び，この構造を用いたDA適応フィルタをMDA適応フィルタと呼ぶ．

3.1　式展開

MDA適応フィルタの2の補数形式のよる式展開を以下に示す．

フィルタ係数ベクトルを

$$\boldsymbol{W}_m(k) = [w_{m0}(k), w_{m1}(k), \ldots, w_{m(R-1)}(k)]^T$$

MDA適応関数ベクトルを

$$\boldsymbol{P}_m(k) = [p_{m0}(k), p_{m1}(k), ..., p_{m(B-1)}(k)]^T$$

(i=0,1,...,M-1)

のように定義すると，フィルタ出力を求める式は次のように表される．

$$y(k) = \sum_{m=0}^{M-1} \boldsymbol{F}^T \boldsymbol{P}_m(k) \qquad (14)$$

$$\boldsymbol{P}_m(k) = \boldsymbol{A}_m^T(k)\, \boldsymbol{W}_m(k) \qquad (15)$$

(i=0,1,...,M-1)

ここで，アドレスマトリクス$\boldsymbol{A}_m(k)$を

$$\boldsymbol{A}_m(k) =$$

$$\begin{bmatrix} -b_{m0}(k) & -b_{m0}(k-1) & \cdots & -b_{m0}(k-R+1) \\ b_{m1}(k) & b_{m1}(k-1) & \cdots & b_{m1}(k-R+1) \\ \vdots & \vdots & & \vdots \\ b_{m(B-1)}(k) & b_{m(B-1)}(k-1) & \cdots & b_{m(B-1)}(k-R+1) \end{bmatrix}^T$$

スケーリングベクトル$\boldsymbol{F}$を

$$\boldsymbol{F} = [2^0, 2^{-1}, \ldots, 2^{-(B-1)}]$$

とする．

式(14)とMDA適応フィルタのブロック図である図6からわかるように，MDA適応フィルタのフィルタ出力を求める部分もDA適応フィルタの場合と同様に，処理速度が次数に依存せず，語長のみに依存する．

MDA適応フィルタの特長は，適応関数空間を分割化し，1つの適応関数空間の入力アドレスをRビットとすることによって，適応関数空間の容量を2^Rと小さく抑えていることである．これによって，収束特性の劣化を抑えることができる．

さらに，ハードウェア実現を考慮した場合，適応関数空間を実現しているRAMは，DA適応フィルタのようにアドレス線数が次数Nと等しい場合は，2^Nという膨大な容量を必要とする．しかし，MDA適応フィルタの場合では，適応関数空間を分割することによって，RAMのアドレス線を$R(=N/M)$(N:次数,M:適応関数空間の分割数)とし，適応関数空間をm個の2^Rの容量のRAMで実現できる．この分割したRAMの出力を加え合わせることによって，DA適応フィルタのRAMと同等の機能を持つ回路を実現することができる．

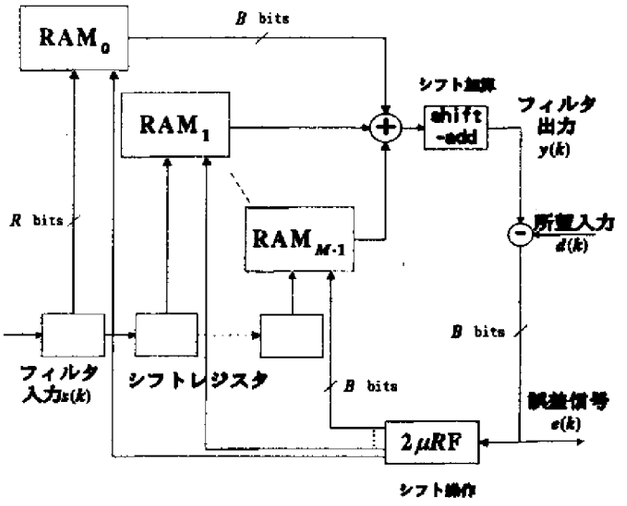


Fig. 6　MDA適応フィルタのブロック図

つまり，大容量のメモリを小容量のメモリと加算器によって実現することにより，大幅にハードウェア量を減少することができる．

ここで，この構成の特長として，適応関数空間が小さければ小さいほど，収束特性が向上する．

MDA適応フィルタの更新式を以下に示す．

$$p_{mi}(k+1) = p_{mi}(k) + 2\mu Re(k)2^{-i} \qquad (16)$$

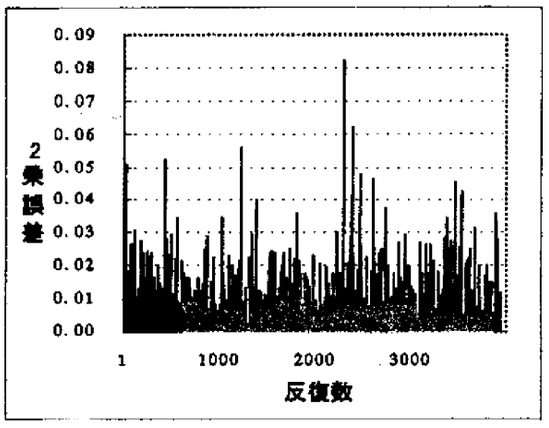
(i=1,2,...,B-1)

$$p_{m0}(k+1) = p_{m0}(k) - 2\mu Re(k)2^0 \qquad (17)$$

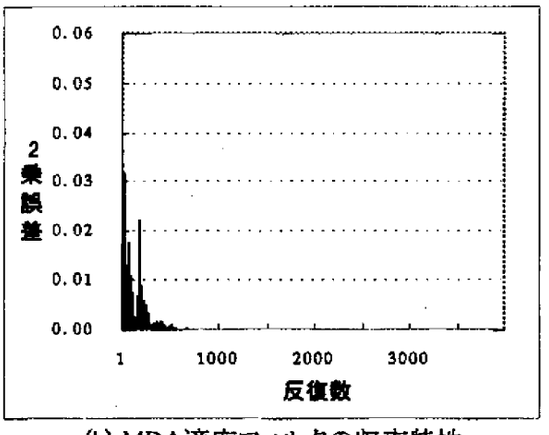
各メモリブロックの更新式は，前述のDA適応フィルタとほぼ同じで，NをRと置いただけである．それは，DA適応フィルタの修正分$2\mu Ne(k)\,\boldsymbol{F}$を適応関数空間の分割数$M$で割ったものに等しくなる．

3.2　シミュレーションによる比較

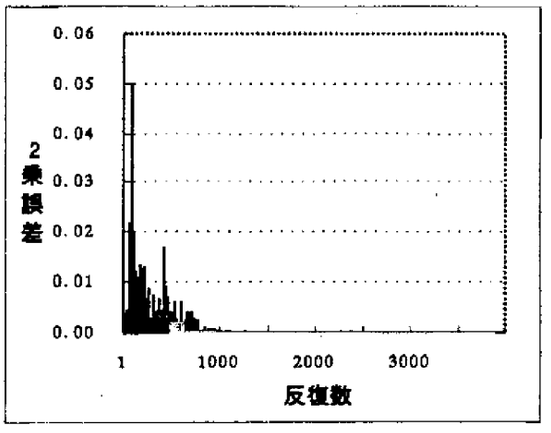
ここまで，DA適応フィルタとMDA適応フィルタの原理について述べてきた．本節では，それぞれの収束特性をシミュレーションによって比較する．



(a) DA適応フィルタの収束特性
(次数 64, ステップサイズ 2^{-4},
RAMのアドレス線数 64)



(b) MDA適応フィルタの収束特性
(次数 64, ステップサイズ 2^{-5},
RAMのアドレス線数 2, RAMの分割数 32)



(c) DLMSによる適応フィルタの収束特性
(次数 64, ステップサイズ 1.0)

Fig. 7　収束特性

それぞれのシミュレーション結果も図7に示す．図7からわかるようにMDA適応フィルタでは，DA適応フィルタに比べて，非常に良好な収束特性を得られることがわかる．比較対象として，DLMSの収束特性を示す．この例では，MDA適応フィルタの方が，DLMSよりも良好な収束特性が得られている．

4.　高速形アーキテクチャについて

本章では，MDA適応フィルタの高速実現について検討する．これまで，分散演算を用いた適応フィルタの高速実現についての検討は行われてこなかった．そこで，我々は分散演算を用いた適応フィルタの高速実現について検討を行った．今回我々は，収束特性を重視した基本形の構成と，高速性を重視した高速形の構成について提案する．

4.1　基本形の構成

まず，基本形のMDA適応アルゴリズムについて述べる．このアルゴリズムは大きく2つの動作に分けることができる．誤差を求める部分とRAMの更新を行う部分である．

まず，誤差を求める部分の動作について説明する．ここでは，現在の入力によって，RAMのアドレスを指定し，関数を読み出す．この値をシフト加算していく．この操作を語長回繰り返し，フィルタ出力を求める．得られたフィルタ出力と所望信号との差をとり，誤差を求める．

次に，RAMの更新を行う部分であるが，誤差が得られたら，現在の入力によって読み出された関数の値と誤差をシフトした値を加え合わせ，関数が読み出されたアドレスと同じアドレスに書き込む．この操作を語長回行うことによって更新を行う．RAMの更新に必要なデータは，現在の誤差と，その現在の誤差を求めるために用いた関数の値と，RAMの読み出しに用いた入力アドレスで

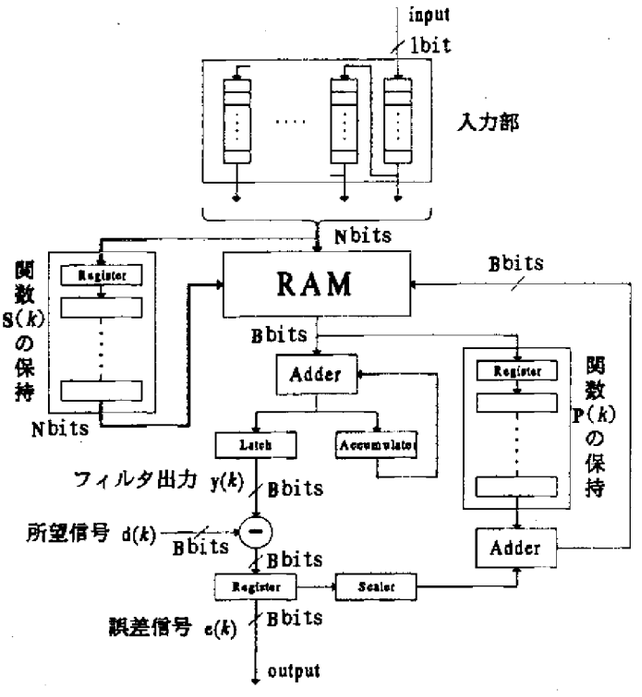


Fig. 8　基本形の構成図

ある．つまり，入力アドレスとそれによって読み出された関数の値を保持しておく必要がある．

これらの点を考慮して，図8のような構成を提案する．この構成では，誤差を求める動作とRAMを更新する動作を，それぞれカウンタによって制御している．誤差を求める動作では，図8のような語長個のレジスタを用いることによって，入力と関数の値を保持している．RAMの更新を行う動作では，レジスタによって保持していた入力$x_i(k)$を用いて，更新すべきRAMのアドレスを指定し，そのアドレスに関数$P_i(k)$と誤差$e(k)$にシフト操作を行ったものを加算器によって加えた値を書き込む．

この構成をパイプライン化した場合のタイムチャートを図9に示す．この図からわかるように，パイプラインの段数分だけ空きが生じる．その欠点として，高次の実現を考慮した場合は，加算段数が増えるため処理時間が多少増加する．しかし，収束特性に関しては，図7で示したように，この例では，DLMSの収束特性より，良好な収束特性

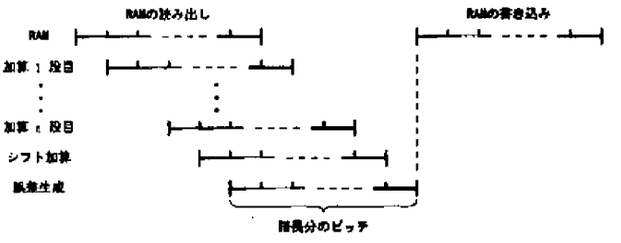


Fig. 9　基本形のタイムチャート

が得られた．

さらに我々は，処理時間が次数に依存しない構成を次節で提案する．

4.2　高速形の構成

前節で述べた基本形の構成を実現する場合には，更新に現在の誤差を用いるため，現在の誤差が求め終わるまで，RAMの更新を行うことができなかった．そこで，我々は，1サンプル前の誤差を用いることによって，パイプラインの空き時間を削除することができる構成を高速形として提案する．1サンプル前の誤差を用いることに対応させて，更新に用いるデータは，1サンプル前の入力と1サンプル前の関数の値を用いることにした．つまり，遅延を1つ入れることによって，RAMの読み出しのステージと書き込みのステージの空き時間を削除している．図10に，そのタイムチャートを示す．これによって，処理時間が次数に依存せず，語長のみに依存する構成が実現可能となる．この構成は図8の構成の入力，関数を保持する部分の構成に改良を加えることによって実現可能となる．この構成の改良点を図11に示す．この図では，入力を保持する部分の構成のみ示したが，関数を保持する構成も同様の構成である．これによって上述の動作が実現可能となる．

この構成の場合の収束特性について図12に示す．基本形の収束特性に比べて，収束特性は劣化している．

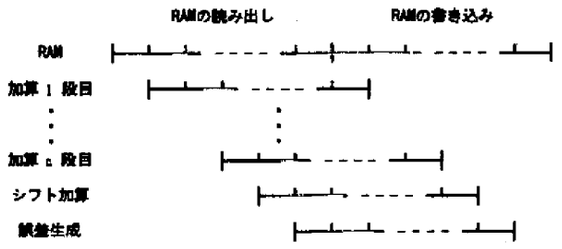


Fig. 10　高速形のタイムチャート

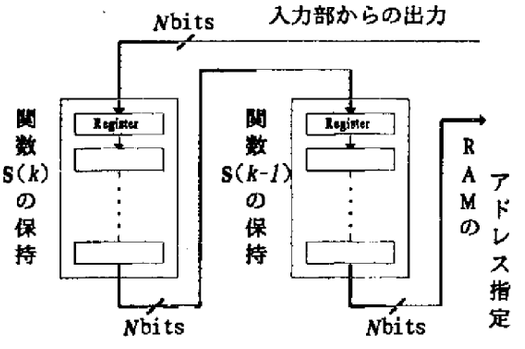
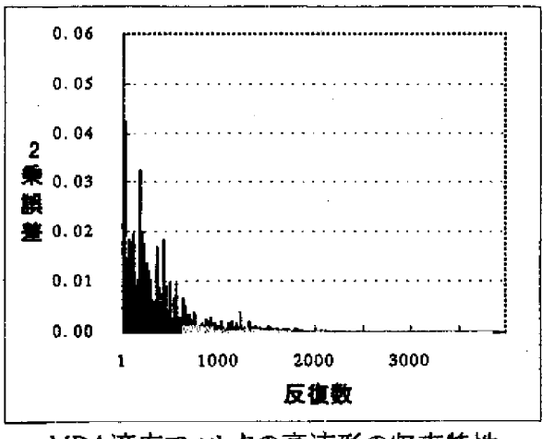


Fig. 11　高速形の改良点



MDA適応フィルタの高速形の収束特性
(次数 64, ステップサイズ 2^{-5},
RAMのアドレス線数 2, RAMの分割数 32)

Fig. 12　高速形の収束特性

Table 1　MDA適応フィルタの高速形アーキテクチャのVLSI評価

Machine cycle[ns]	31
Sampling rate[MHz]	1.15
Latency[ns]	992
Power dissipation[W]	2.72
Area[mm^2]	17.433
Number of gates	69642

Table 2　DLMSによる適応フィルタのVLSI評価

Machine cycle[ns]	61
Sampling rate[MHz]	16.39
Latency[ns]	1952
Power dissipation[W]	6.21
Area[mm^2]	47.247
Number of gates	230528

5.　VLSI評価

前章で述べた高速形の構成についてVLSI設計システムPARTHENONによってVLSI評価を行った．その結果を表1に示す．セルライブラリの設計ルールは0.8μmCMOSスタンダードセル(VLSIテクノロジー社)である．設計仕様は，次数32，RAMのアドレス線数4，RAMの分割数8である．比較対象として，DLMSによる乗算器を用いた場合の性能評価の結果も表2に示す．

この結果から，今回我々が提案した高速形の構成は，DLMSによる適応フィルタの構成に対して消費電力を56%，面積を63%，ゲート数を70%低下させることができた．

6.　おわりに

本研究では，我々が提案した分散演算を用いた適応フィルタの基本形，高速形に対して，収束特性，及びアーキテクチャを検討した．

高速性の構成では，処理時間が次数に依存せ

ず，語長のみに依存する構成が実現可能となることを示した．さらに，ハードウェア量に関しても，DLMSアルゴリズムを用いた場合の構成と比較して，大幅に削減できることを明らかにした．また，基本形の構成は，多少処理時間を犠牲にすることによって，良好な収束特性と，高速形よりも少ないハードウェア量での実現が可能となる．これらのことから，分散演算を用いた適応フィルタの構成法が，ハードウェア量を効率的に小さく抑えることができる実現法であることを示した．

今後の課題として，より高次の場合の構成について，収束特性の検討とVLSI評価を行う必要がある．

謝辞

本研究を進めるにあたり，PARTHENONをご提供いただいたNTTコミュニケーション科学研究所に感謝申し上げます．

参考文献

1) 松原勝重，西川清史，貴家仁志: Delayed LMSアルゴリズムに基づくパイプライン適応フィルタ，電子情報通信学会論文誌 A，Vol.J79-A No.5, 1050/1057(1996)

2) Meyer, and Agrawal: A High Sampling Rate Delayed LMS Filter Architecture, IEEE Trans. CAS-II 93 NOV.(1993)

3) Wang: Bit-Serial VLSI Implementation of Delayed LMS Transversal Adaptive Filters, IEEE Trans. SP 94 AUG.(1994)

4) 野崎剛，恒川佳隆，三浦守：高次FIRフィルタの高速・低消費形アーキテクチャ，情報処理学会東北支部，第3回研究会，96-3-7, (1996)

5) C.F.N.Cowan, and J.Mavor: New Digital Adaptive Filter Implementation using Distributed Arithmetic Techniques, IEE Proc., Vol.128, Pt.F, No4, AUG., 225/233(1981)

6) C.-H.Wei: Multimemory Block Structure for Implementing a Digital Adaptive Filter using Distributed Arithmetic, IEE Proc., Vol.133, Pt.G, No.1, FEB., 19/26(1986)

7) Yi-Yung Chiu and Che-Ho Wei: On the Realzation of Multimemory Block Structure Digital Adaptive Filter using Distributed Arithmetic, Journal of the Chinese Institute of Engineers, Vol.10,No.1, 115/122(1987)

8) 辻井重男，久保田一，古川利博，趙晋輝：適応信号処理，5/45，昭晃堂(1995)