■1 群（信号・システム）- 9 編（ディジタル信号処理）

8 章 信号処理システム実現技術

（執筆者：尾上孝雄・黒田一朗・中山謙二）［2011 年 3 月 受領］

■概要■

本章では，ディジタル信号処理システムの実現技術として，FIR 実現技術，Interpolator/Decimator 及び信号処理プロセッサについて概説する．

【本章の構成】

本章の構成は以下のとおりである．8-1 節では，信号処理システムの要素技術として必要不可欠な FIR フィルタのハードウェアに適した四つの構成法として Reduced Adder Graph，Distributed Arithmetic，Pipeline，Systolic について述べ，それらの特徴についてまとめる．8-2 節では，マルチレート信号処理の導入による FIR フィルタの高速化について解説する．またレート変換の効果的な実現法として，ポリフェイズ実現法と Cascaded Integrator-Comb 実現法について述べる．8-3 節では，プログラマブルなディジタル信号処理プロセッサ（DSP）について解説する．DSP のアーキテクチャ，信号表現及び量子化誤差についてまとめる．

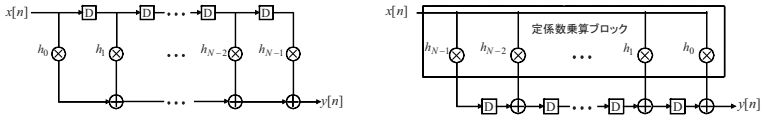
■1 群 - 9 編 - 8 章

8-1 FIR 実現技術

（執筆者：尾上孝雄）[2009 年 9 月 受領]

信号処理システム，特に FIR フィルタは，その安定性から多くの応用システムで用いられている．また積和演算の単純接続という演算構造をもつことから，ディジタルフィルタ用アーキテクチャの基礎として広く研究されてきている．一般に FIR フィルタは式 (8･1) で表され，図 8･1 に示す基本構成で処理される．本節では四つの代表的な FIR フィルタ構成法について説明する．

$$y[n] = \sum_{k=0}^{N-1} x[n-k]h_k \tag{8･1}$$



(a) 直接型　　(b) 転置型

図 8･1　FIR フィルタの一般構成

8-1-1 RAG（Reduced Adder Graph）

実用されている FIR フィルタの多くは，各タップの係数（h_k）すなわち被乗数が定数である定係数乗算となっている．演算ビット数にも依存するが，一般に乗算の実装コストは加算のそれに比べて高く数倍～数十倍である．このため，定係数乗算であることを積極的に活用して効率のよい処理を達成するアーキテクチャが多く提案されている．

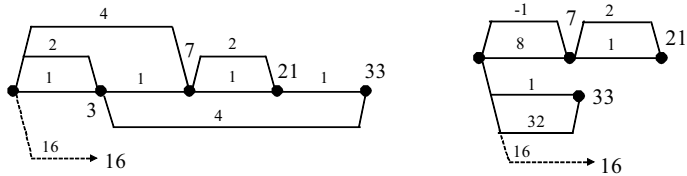
図 8･1(b) の FIR フィルタは，入力 $x[n]$ に対する定係数乗算ブロックと遅延素子を介した累算と考えることができる．ここで，2 の累乗倍がビットシフトを行うことで簡単に得られることを利用して，図 8･2(a) に示すようなグラフ構造で定係数乗算ブロックを実現する方法がある．始点終点を除くグラフ中の各点は加算による部分積を表し，各枝は枝ラベルに示される 2 の累乗倍を表している．本グラフを実現する最小演算リソースを求める最適化問題は NP 完全問題に帰着できるため，さまざまなヒューリスティック手法が提案されている．図 8･2(a) は 1，7，16，21，33 という係数を BH（Bull-Horrocks）アルゴリズム[1)]を用いて生成した例である．五つのシフトと四つの加算で係数を実現している．

RAG（Reduced Adder Graph）[2)]は，下記の三つの拡張を施すことにより，さらに少ない演算リソースでの実現を目指すものである．

1)　各部分積の正負組に対する 2 の累乗倍を用いることにより，係数を超える部分積も活用する．(例: 7 = 4 + 2 + 1 の代わりに 7 = 8 – 1 を利用する)

2)　各部分積に対しても 2 の累乗での分解を行うことにより，後段で係数を生成する部分積の組み合わせ数を増やす．

3)　係数を生成する順序を小さい順ではなく，生成コスト順に変更する．RAG を用いて係数を生成した例を図 8･2(b) に示す．上記の拡張により，RAG は BH に対して 20 %以上（12 ビット 5 係数の場合）のコスト削減を達成している．



(a) BH アルゴリズム　　(b) RAG アルゴリズム

図 8･2　定係数乗算ブロックのグラフ構造による表現

8-1-2 Distributed Arithmetic

Distributed Arithmetic（DA: 分散演算）は，二つのベクトル内積を求める際にビットシリアル演算を適用する構成法である．DA により定係数 FIR フィルタを実現する場合，被乗数が定数ベクトルとなるため，乗算器を用いずにメモリと累算器のみで構成することが可能であり，ハードウェアコストを低く抑えることができる[3, 4]．

式 (8･1) 中の $x[n-k]$ が B ビットの 2 の補数として表現されている場合，式 (8･2) のように表すことができる．

$$x[n-k] = -x^0[n-k] + \sum_{j=1}^{B-1} x^j[n-k]2^{-j} \tag{8･2}$$

ここで，$x^j[n-k]$ は $x[n-k]$ の j ビット目であり 0 もしくは 1 の値をとる．また，$x^0[n-k]$ は符号ビットである．式 (8･2) を用いることにより，式 (8･1) は

$$y[n] = \sum_{j=1}^{B-1} F_j 2^{-j} - F_0 \tag{8･3}$$

と書き換えられる．但し，F_j は式 (8･1) で計算される各フィルタタップの j ビット目の部分積の和をとったものである．

$$F_j(x^j[n], x^j[n-1], \cdots, x^j[n-(N-1)]) = \sum_{k=0}^{N-1} x^j[n-k]h_k \tag{8･4}$$

式 (8･4) の右辺で加算される各要素は，定係数である h_k（$x^j[n-k]$ が 1 の場合）あるいは 0

（$x^j[n-k]$ が 0 の場合）となるため，F_j は式 (8･4) 右辺の引数である $x[n-k]$ $(k=0,1,\cdots,N-1)$ の j ビット目を入力としたメモリ参照で実現できる．このように，DA では図 8･3 に示す 2^{N-1} ワードのメモリと累算器で FIR フィルタ処理を実行する．

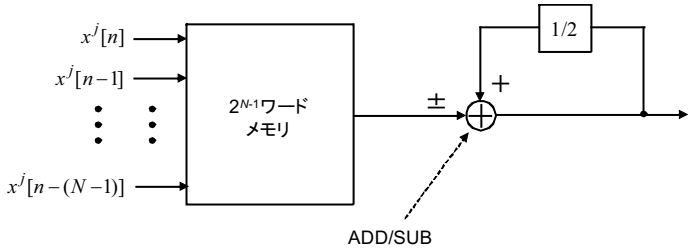


図 8･3　DA による FIR フィルタの表現

2^{N-1} ワードのメモリは，フィルタタップ数である N が大きくなると規模が膨大になり，実用性に欠けてしまう．そこで，式 (8･4) を i 段に分割することで，

$$\begin{aligned} & F_j(x^j[n], x^j[n-1], \ldots, x^j[n-(N-1)]) \\ &= \sum_{k=0}^{m_1-1} x^j[n-k]h_k + \sum_{k=m_1}^{m_2-1} x^j[n-k]h_k + \cdots + \sum_{k=m_{i-1}}^{N-1} x^j[n-k]h_k \\ &= F_{j1} + F_{j2} + \cdots + F_{ji} \end{aligned} \tag{8･5}$$

と表現し，i 個の小さいサイズのメモリを用いて図 8･4 のようにすれば効率よく実現できる．

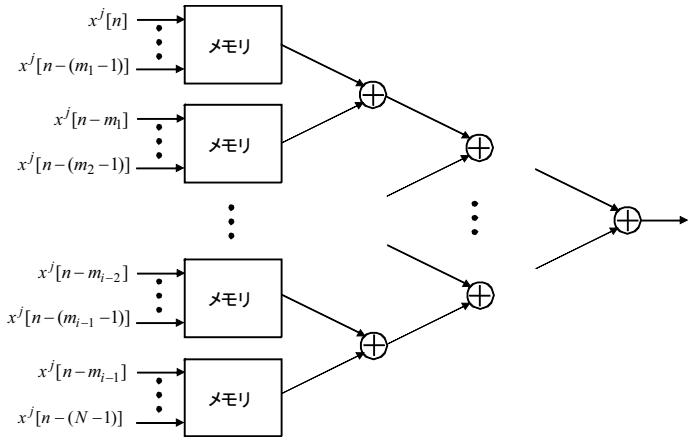


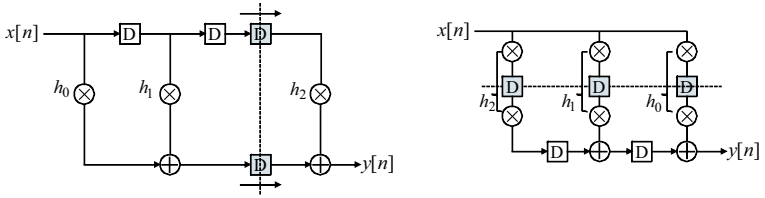
図 8･4　DA のメモリ分割

8-1-3 Pipeline

FIR フィルタの性能は各演算の処理時間に依存する．乗算，加算の処理時間をそれぞれ T_M，T_A とすると，直接型構成（図 8･1(a)）の場合 1 サンプルの処理に $T_M + N \cdot T_A$ の時間が必要となり，最大サンプル周波数は $1/(T_M + N \cdot T_A)$ となる．パイプライン構成は，この処理を分割することにより最大サンプル周波数，すなわちスループットを向上させるものである[5]．

パイプライン分割は，フィルタを表すシグナルフローグラフにおけるフィードフォワードカットセットに対して，遅延素子を挿入することにより実現できる．複数ステージに分割する場合，各ステージでの処理遅延を平均化するようなカットセットを選択する．図 8･5(a) は，3 タップの直接型 FIR フィルタを 2 ステージのパイプライン構成で実現する例である．パイプライン構成は，スループットを向上させる一方で，レイテンシならびに面積オーバヘッドが発生する．

転置型 FIR フィルタ（図 8･1(b)）の場合は，フィルタタップ数に関わらず $T_M + T_A$ の時間ごとに 1 サンプルの処理が可能である．また，一般に乗算の処理時間は加算のそれに加えて大きいため，転置型に対しても図 8･5(b) に示すように乗算を 2 ステージに分割し，遅延素子を挿入することでスループットを向上させることも可能である．



(a) ステージ分割　　(b) 乗算の分割

図 8･5　パイプライン構成

8-1-4 Systolic

1980 年代に H.T. Kung によって提案されたシストリックアレイは，単純処理を行う PE（Processing Element）を規則的に配置し，並列かつパイプライン的に動作させるものである．FIR フィルタ設計では PE は積和演算に相当し，図 8･6 のように依存グラフに対して入力（$x[n]$），出力（$y[n]$），係数（h_k）をマッピングすることによりさまざまな構成が実現できる．但し，横軸は時間を示している．

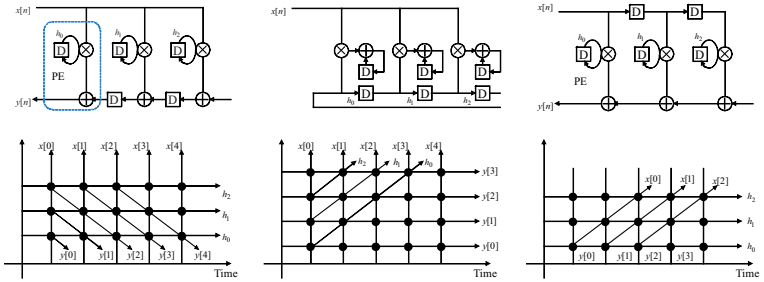


図 8･6　シストリックアレイ構成

■参考文献

1) D.R. Bull and D.H. Horrocks, “Primitive operator digital filters,” IEE Proc. G, vol.138, no.3, pp.401-412, Jun., 1991.
2) A.G. Dempster and M.D. Macleod, “Use of minimum-adder multiplier blocks in FIR digital filters,” IEEE Trans. Circuits and Systems-II, vol.42, no.9, pp.569-577, Sep., 1995.
3) A. Peled and B. Liu, “A new approach to the realization of nonrecursive digital filters,” IEEE Trans. Audio and Electroacoustics, vol.AU-21, no.6, pp.477-484, Dec., 1973.
4) A. Peled and B. Liu, “A new hardware realization of digital filters,” IEEE Trans. Acoustics, Speech, and Signal Processing, vol.ASSP-22, no.6, pp.456-462, Dec., 1974.
5) K.K. Parhi, “VLSI Digital Signal Processing Systems: Design and Implementation,” Wiley-Interscience, Jan., 1999.

■1 群 - 9 編 - 8 章

8-2 FIR フィルタの高速化

（執筆者：中山謙二）[2009 年 9 月 受領]

音声・音響帯域では，ディジタルシグナルプロセッサ（DSP）を用いて，標準的な構成法を用いることができるが，画像・映像処理や無線通信の分野では，より広帯域のディジタル信号処理が必要とされ，単位時間に要する計算量の低減が重要となる．本節では，応用範囲が広い FIR フィルタに関して，計算量を低減する手法を述べる．

8-2-1 間引きと並列構成による FIR フィルタの高速化

FIR フィルタの連続する M 個の出力信号 $y(n), y(n-1), \cdots, y(n-M+1)$ を並列計算することにより生じる冗長性を除去することにより高速化を行う[1)]．

ここでは，$M = 2$ の場合について説明する．入力信号を $x(n)$，FIR フィルタのインパルス応答を $h(n)$ とすると，$y(n)$，$y(n-1)$ は次のように表される．

$$y(n) = \sum_{m=0}^{N-1} h(m)x(n-m) \tag{8·6}$$

$$y(n-1) = \sum_{m=0}^{N-1} h(m)x(n-1-m) \tag{8·7}$$

次に，式 (8·6)，(8·7) を行列の形で表現する．まず，次のような間引きされたベクトルを定義する．ここで，N は偶数としている．

$$\boldsymbol{h}_0 = [h(0), h(2), \cdots, h(N-2)]^T \tag{8·8}$$

$$\boldsymbol{h}_1 = [h(1), h(3), \cdots, h(N-1)]^T \tag{8·9}$$

$$\boldsymbol{x}_e(n) = [x(n), x(n-2), \cdots, x(n-N+2)]^T \tag{8·10}$$

$$\boldsymbol{x}_o(n-1) = [x(n-1), x(n-3), \cdots, x(n-N+1)]^T \tag{8·11}$$

$$\boldsymbol{x}_e(n-2) = [x(n-2), x(n-4), \cdots, x(n-N)]^T \tag{8·12}$$

これらのベクトルを用いて，$y(n)$ と $y(n-1)$ は次のように表される．

$$\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_o^T(n-1) & \boldsymbol{x}_e^T(n-2) \\ \boldsymbol{x}_e^T(n) & \boldsymbol{x}_o^T(n-1) \end{bmatrix} \begin{bmatrix} \boldsymbol{h}_0 \\ \boldsymbol{h}_1 \end{bmatrix} \tag{8·13}$$

この式を更に次のように変形する．

$$\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_o^T(n-1)(\boldsymbol{h}_0 + \boldsymbol{h}_1) + (\boldsymbol{x}_e^T(n-2) - \boldsymbol{x}_o^T(n-1))\boldsymbol{h}_1 \\ \boldsymbol{x}_o^T(n-1)(\boldsymbol{h}_0 + \boldsymbol{h}_1) - (\boldsymbol{x}_o^T(n-1) - \boldsymbol{x}_e^T(n))\boldsymbol{h}_0 \end{bmatrix} \tag{8·14}$$

上式で，$\boldsymbol{x}_o^T(n-1)(\boldsymbol{h}_0 + \boldsymbol{h}_1)$ の計算は $y(n-1)$ と $y(n)$ で共通である．これをブロック図で表すと図 8·7 のようになる．

$\boldsymbol{x}_o^T(n-1) - \boldsymbol{x}_e^T(n)$ は $\boldsymbol{x}_o^T(n-3) - \boldsymbol{x}_e^T(n-2)$ の第 i 要素を第 $i+1$ 要素にシフトし，第 1 要素に $x(n-1) - x(n)$ を加えることにより計算できる．また，例えば，$x(n-1) - x(n)$ を計算す

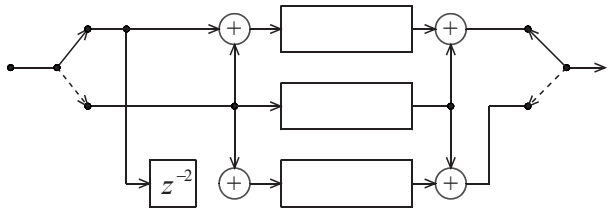


図 8･7　間引きと並列構成による FIR フィルタの高速化（2 チャネルの場合）.

るために，これらを同期させる必要がある.

元の FIR フィルタのタップ数を N とすると，$\boldsymbol{h}_0$，$\boldsymbol{h}_1$，$\boldsymbol{h}_0 + \boldsymbol{h}_1$ のサンプル数は約 $N/2$ となる．更に，入力信号も FIR フィルタのインパルス応答も間引きされているので，これらのブロックにおける標本化周波数は半分になる．計算量は出力 2 サンプルを計算するのに通常の FIR フィルタでは単位時間当たり約 $2N \times f_s$ 回の計算が必要であるが，図 8･7 では，約 $(3N/2) \times (f_s/2)$ 回の計算で済み，約 3/8 に低減されたことになる．この例では，並列数が 2 であったが，これを更に大きくすることにより計算量を低減できる.

8-2-2 間引きフィルタと内挿フィルタ

ディジタルシステムでは標本化周波数（レート）の変換がよく行われる．標本化周波数の変換は信号の間引き（Decimation）と内挿（Interporation）に相当する．ここでは，間引きフィルタと内挿フィルタの高速化について述べる.

低い標本化周波数を f_{sl}，高い標本化周波数を f_{sh} とし，$f_{sh} = R f_{sl}$ の関係があるものとする．標本化周波数を f_{sh} としたときの入力信号を $x(n)$，フィルタのインパルス応答を $h(n)$，出力信号を $y(n)$ とする.

$$y(n) = \sum_{m=0}^{N-1} h(m)x(n-m) \tag{8･15}$$

これを R サンプルごとに間引くと，次のようになる.

$$\begin{aligned} y(Rn) &= \sum_{m=0}^{N-1} h(m)x(Rn-m) \\ &= \sum_{i=0}^{R-1}\sum_{m=0}^{R_i-1} h(Rm+i)x(Rn-Rm-i) \end{aligned} \tag{8･16}$$

R_i は $h(Rm+i)$ のサンプル数である．$h(Rm+i)$，$x(Rn-Rm-i)$ の z 変換を $H_i(z^R)$，$X_i(z^R)$ とすると，$y(Rn)$ の z 変換は

$$Y(z^R) = \sum_{i=0}^{R-1} H_i(z^R)X_i(z^R) \tag{8･17}$$

となる．$H(z)$ のタップ数を N とすると，$H_i(z^R)$ のタップ数 R_i は約 N/R であり，かつ，$H_i(z^R)X_i(z^R)$ は f_{sl} で計算される．従って，間引きフィルタは f_{sl} で動作する約 N/R タップの FIR フィルタを R 個並列接続して構成される．回路構成を図 8･8 に示す．この回路形式はポリフェイズフィルタと呼ばれる[2]．$X(z)$ から $X_i(z^R)$ を同期させて取り出すために遅延器 z^{-i} が用いられる．単位時間当たりの計算量は $Nf_{sl} = Nf_{sh}/R$ である．

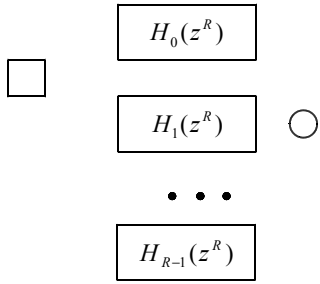


図 8･8　間引きフィルタの回路構成（$f_{sh} \rightarrow f_{sl} = f_{sh}/R$）．

内挿フィルタでは，入力信号は f_{sl} で標本化されているから，出力信号は次のように表される．

$$y(Rn+i) = \sum_{m=0}^{R_i-1} h(Rm+i)x(R(n-m)), \quad i = 0, 1, \cdots, R-1 \tag{8･18}$$

$h(Rm+i)$，$x(R(n-m))$ の z 変換を $H_i(z^R)$，$X(z^R)$ とすると，$y(Rn+i)$ の z 変換は

$$Y_i(z^R) = H_i(z^R)X(z^R), \quad i = 0, 1, \cdots, R-1 \tag{8･19}$$

となる．これを z^{-i} を通して内挿する．内挿された出力は

$$Y(z) = \sum_{i=0}^{R-1} z^{-i} H_i(z^R)X(z^R) \tag{8･20}$$

となる．内挿フィルタの構成を図 8･9 に示す． $H_i(z^R)$ のタップ数は約 N/R で，$H_i(z^R), i = 0, 1, \cdots, R-1$ が必要とされるので，全体のタップ数は N である．一方，$H_i(z^R)X(z^R)$ の計算は f_{sl} で行われる．計算量は間引きフィルタと同じである．

8-2-3 櫛形フィルタと積分器による間引き／内挿フィルタの高速化

櫛形フィルタと積分器の縦続構成により，乗算器を用いない（Multiplierless）間引きフィルタ，内挿フィルタの構成法[3]について述べる．

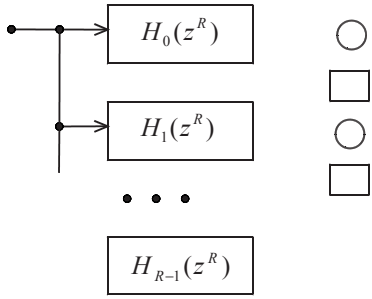


図 8･9　内挿フィルタの回路構成（$f_{sl} \rightarrow f_{sh} = Rf_{sl}$）

櫛形フィルタの伝達関数を次に示す.

$$H_C(z) = 1 - z^{-RM} \qquad \text{標本化周波数：} f_{sh} = Rf_{sl} \tag{8･21}$$

$$H_C(z) = 1 - z^{-M} \qquad \text{標本化周波数：} f_{sl} \tag{8･22}$$

この伝達関数は，標本化周波数を f_{sh} とすると，$z = exp(j2\pi i/RM), i = 0, 1, \cdots, RM - 1$ に零点をもつ．その内，$z = 1$ にある零点を積分器で相殺し，帯域制限フィルタを実現する．

積分器の伝達関数を次に示す.

$$H_I(z) = \frac{1}{1 - z^{-1}} \qquad \text{標本化周波数：} f_{sh} = Rf_{sl} \tag{8･23}$$

これは，$z = 1$ に極を有し，この極と $H_C(z)$ の $z = 1$ における零点が相殺する．

内挿フィルタは入力側（f_{sl}）に $H_C(z)$ を配置し, 出力側（f_{sh}）に $H_I(z)$ を配置する．また, 間引きフィルタでは, その逆となる．櫛形フィルタと積分器の縦続構成（Cascaded Integrator-Comb (CIC) filters）を図 8･10 に示す.

図 8･10　内挿フィルタ（左側）：櫛形フィルタ（左：f_{sl}）と積分器（右：f_{sh}）の縦続構成．間引きフィルタ（右側）はその逆順．

全体の伝達関数は次のようになる.

$$H(z) = H_I(z)H_C(z^R) = H_C(z^R)H_I(z) = \frac{1 - z^{-RM}}{1 - z^{-1}} \tag{8･24}$$

これは次のように展開できる．

$$H(z) = 1 + z^{-1} + z^{-2} + \cdots + z^{-RM+1} \tag{8･25}$$

上式では，入力信号の直流成分，あるいは低周波数成分が RM 倍されるので，これを 1 とするスケーリングを行い，更にこの CIC 区間を K 段用いると，全体の伝達関数は次のようになる．

$$H(z) = \left(\frac{1}{RM}\frac{1 - z^{-RM}}{1 - z^{-1}}\right)^K \tag{8･26}$$

この伝達関数の周波数特性は次式で与えられる．

$$H(e^{j\omega T_h}) = \left(\frac{sin(\omega T_h RM/2)}{RMsin(\omega T_h/2)}\right)^K \tag{8･27}$$

ここで，$T_h = 1/f_{sh}$，$T_h R = 1/f_{sl}$ である．$|H(e^{j\omega T_h})|$ では，低域が通過域であり，f_{sl}/M の整数倍（零を除く）の所に零点を有し，繰り返し成分を抑圧する．元の信号が $|f| < f_c$ に帯域制限されているとすると，通過域端の減衰量は $|H(e^{j2\pi f_c T_h})|$ で決まり，最初の阻止域端の減衰量は $|H(e^{j(2\pi(f_{sl}-f_c)T_h})|$ で決まる．これらの値が所望の値となるように M と K が決められる．

間引きフィルタでは，k 個の積分器が入力側（f_{sh}）に配置され，k 個の櫛形フィルタが出力側（f_{sl}）に配置される．この様子を図 8･11 に示す．入力側の積分器は f_{sh} で動作し，出力側の櫛形フィルタは f_{sl} で動作する．逆に，内挿フィルタでは，櫛形フィルタが入力側に，積分器が出力側に配置される．

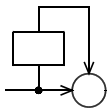


図 8･11　積分器（左：f_{sh}）と櫛形フィルタ（右：f_{sl}）を k 段用いる CIC 形間引きフィルタ

直流成分は積分器では無限大に増幅されるため，入力信号に含まれる直流成分を零にする必要がある．また，櫛形フィルタでは，直流成分は零になるため，これを利用することはできない．また，低周波数帯域も積分器で大きく増幅されるため，信号のレジスタを長くする必要がある．

次に，櫛形フィルタと積分器を一体化して考える．式 (8･25) で $M = 1$ とすると，

$$H(z) = 1 + z^{-1} + z^{-2} + \cdots + z^{-R+1} \tag{8･28}$$

となる．内挿フィルタでは，入力信号（f_{sl}）のサンプルを出力側（f_{sh}）で $R-1$ サンプル分保持（ホールド）する．間引きフィルタでは，入力信号（f_{sh}）の連続する R サンプルの和を f_{sl} のレートで計算し，出力する．直流成分は R 倍されるので，レジスタ長を長くする必要である．

更に，低ビットのフィルタ係数を許せば，フィルタ特性を改善できる．低ビット FIR フィルタの構成法としても節で述べたポリフェイズフィルタが有効である．

■参考文献

1) Z.J. Mou and P. Duhamel, “Short-length FIR filters and their use in fast nonrecursive filtering,” IEEE Trans. on Signal Processing, vol.39, no.6, pp.1322-1332, Jun., 1991.
2) M. Bellanger and J.L. Daguet, “TDM-FDM transmultiplexers: Digital polyphase and FFT,” IEEE Trans. Commun., vol.COM-22, pp.1199-1205, Sep., 1974.
3) E.B. Hogenauer, “An economical class of digital filters for decimation and interpolation,” IEEE Trans. Acoust., Speech, Signal Processing, vol.ASSP-29, pp.155-162, Apr., 1981.

■1 群 - 9 編 - 8 章

8-3 信号処理プロセッサ

（執筆者：黒田一朗）[2009 年 9 月 受領]

8-3-1 DSP（Digital Signal Processor）

組込み用途のディジタル信号処理システムの実現手段としてプログラマブルなディジタル信号処理プロセッサ（以下 DSP）がある．DSP はマイクロプロセッサをベースとしてディジタル信号処理を効率よく実現できるアーキテクチャを採用している（10 群 5 編 2 章 2-2 参照）．専用の信号処理 LSI を設計・開発する場合と比較して応用システムを実現（ソフトウェア開発）するコスト，期間が少ないことから，音声・オーディオ処理，画像処理，通信処理をはじめ様々な分野で利用されている．DSP は代表的なディジタル信号処理アルゴリズムである FIR ディジタルフィルタや FFT などを効率よく実現するように設計されており，主な特徴として，高速積和演算機能，同時アクセス可能な複数の内蔵メモリ，信号処理向きのアドレス演算やプログラム制御機能のなどの機能を内蔵している．

(1) 積和演算アーキテクチャ

ディジタルフィルタに代表される信号処理演算では，積和演算（乗算＋加算）を多用する．DSP はこのような積和演算の繰り返しを高速に実現するために，乗算器，あるいは積和演算器を内蔵し，積和演算を毎サイクル実行可能にしたプロセッサとして実現されている．固定小数点積和演算アーキテクチャの詳細については本章 3-2 で説明する．

(2) メモリアクセスとアドレス演算機能

(a) 並列メモリアーキテクチャ

DSP で強化された演算機能を処理性能に結びつけるためには，演算器に十分なデータを供給できるだけのメモリアクセス性能（メモリバンド幅）を用意する必要がある．例えば，FIR ディジタルフィルタを実現する場合は，プログラムメモリのアクセス以外に，1 回の積和演算に対して信号のベクトル（ディレイライン）と係数のベクトルのアクセスなど少なくとも 2 回のデータアクセスが必要になる．このために，DSP では，複数のメモリを同時にアクセスできるメモリアーキテクチャを採用している．図 8･12 に複数のメモリバンク（マルチバンクメモリ）と複数のデータバスからなる構成例を示す．

(b) アドレス演算

効率的な並列メモリアクセスを実現するためには，アクセスするメモリのアドレスの計算を効率よく（追加サイクルなしに）実現する必要がある．例えば，ディジタルフィルタの演算では，1 積和演算あたり 2 回のデータメモリアクセスが発生するが，これに対応して 2 回分のアドレス生成（供給）を行う必要がある．これらのアドレスをプログラムメモリで直接指定することもできるが，命令語長が長くなりプログラムメモリ容量の増加を招く．そこで，DSP では効率的なアドレス生成のための専用のアドレス演算器を用意しており，レジスタ間接アドレッシングを始めとするアドレス演算を実現している．

(c) レジスタ間接アドレッシング

レジスタ間接アドレッシング方式では，レジスタに格納されたアドレスデータを用いてアドレス生成演算を行いデータメモリのアクセスを行う．アドレス生成演算としては，+1（Increment），−1（Decrement），アドレスレジスタ間加算／減算などがある．例えば，ディジ

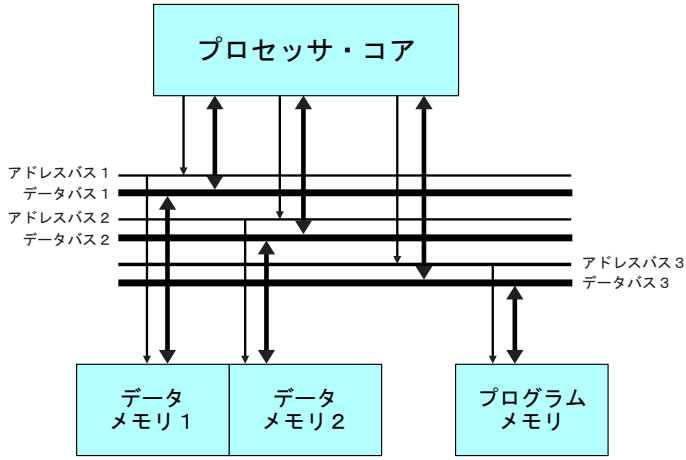


図 8･12　並列メモリアーキテクチャ

タルフィルタの連続積和演算では，信号ベクトルと係数ベクトルの各要素を順次アクセスするために +1 または −1 演算を行ってアドレス生成を行う．

(d) モジュロアドレッシング及びそのほかのアドレッシング方式

ディジタルフィルタの演算では，一回の出力サンプルの演算を行う度に，信号ベクトルのデータを 1 サンプル分更新（シフト) する必要がある．これを直接メモリのリード，ライトで実現すると必要なメモリアクセスバンド幅を増加させてしまうために，信号ベクトル（ディレイライン）を巡回バッファ（Circular Buffer）として実現し，アクセスの開始番地を 1 番地ずつずらすことにより等価的に信号データのシフトを実現する．巡回バッファは図 8･13 に示すように連続したメモリの領域において両端のアドレスが隣接するようにしたもので，最大のアドレスに 1 を加えると最小のアドレスに戻るようになっており，モジュロ演算を用いたモジュロアドレッシングにより実現する．

そのほか，ディジタル信号処理特有のアドレス演算機能として，高速フーリエ変換（FFT）におけるビットリバース（ビット反転）アドレス演算をサポートする機能などが搭載される（例 11001010 ⇒ 01010011）．

(3) 並列処理

DSP では，以上に示したような積和演算，2 回のメモリアクセス及びそれに伴うアドレス生成演算を N タップ分実行することにより，N タップ ディジタルフィルタを実現する．例えば，1 命令（1 サイクル）でこれらのデータ演算，メモリアクセス，アドレス演算を並列に実行することにより N タップ ディジタルフィルタを N サイクル（＋初期設定等）で実行する．これらを実現するプログラム制御方式としては，複数の操作を 1 命令で実行する命令セットを用意する場合や，複数命令の同時実行で実現する場合などがある．

(a) 並列処理による性能向上

二つ以上の積和演算器を同時に動作させることにより，N タップディジタルフィルタを N サイクル以下で実現することが可能になる．複数の演算器，メモリアクセスに対応した制御方式

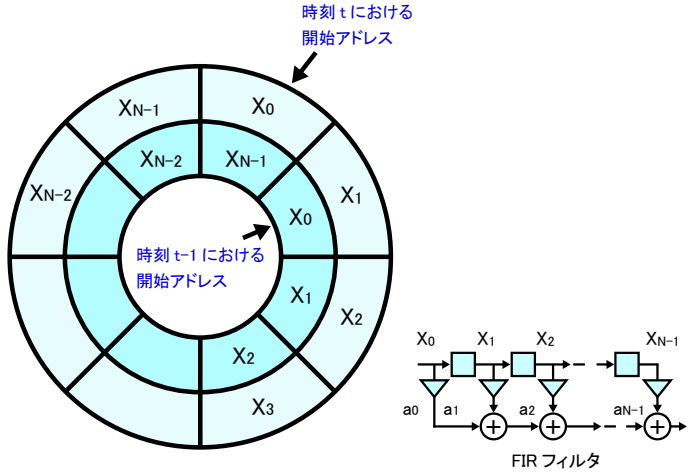


図 8･13　モジュロアドレッシングと FIR フィルタ実現

としては，複数の命令を同時に実行する制御方式の一つである VLIW（Very Long Instruction Word）制御方式が利用されることが多い．また，複数命令の同時実行による性能向上以外に，演算器を分割して利用（例えば 32 ビットの演算器を四つの 8 ビット演算器として利用）することによる並列処理や，一つの LSI に複数の DSP を内蔵して並列処理を行い，性能向上を図る場合もある．複数の DSP の制御方法としては，一つのプログラムで複数の DSP を制御する方式（SIMD: Single Instruction Multiple Data）と各々の DSP がそれぞれ（別の）プログラムを実行する方式（MIMD: Multiple Instruction Multiple Data）とがある．

8-3-2 固定小数点実現と誤差

(1) 固定小数点演算

音声等のディジタル信号処理演算では，シミュレーションやプロトタイピングの際に浮動小数点演算を用いるが，浮動小数点演算のための演算器のハードウェアの規模が大きいことから，組込みシステム用には一部の例外を除き固定小数点演算が用いられることが多い．固定小数点表現としては，16 ビットあるいは 24 ビット語長のデータが用いられることが多い．

(2) 2 の補数表現

固定小数点データフォーマットとして代表的なものに 2 の補数表現がある．2 の補数表現では図 8･14 に示すように，最上位ビットを符号ビット S_0 として，正数のとき $S_0 = 0$，負数のとき $S_0 = 1$ として，$-1 =< D < 1$ の範囲にある n ビット固定小数点数 $D = (S_0 S_1 S_2 \cdots S_{n-1})$（$Si = 0$ または 1）を次式で表す．

$$D = -S_0 + \sum_{i=1}^{n-1} S_i \cdot 2^{-i} \qquad (8\cdot29)$$

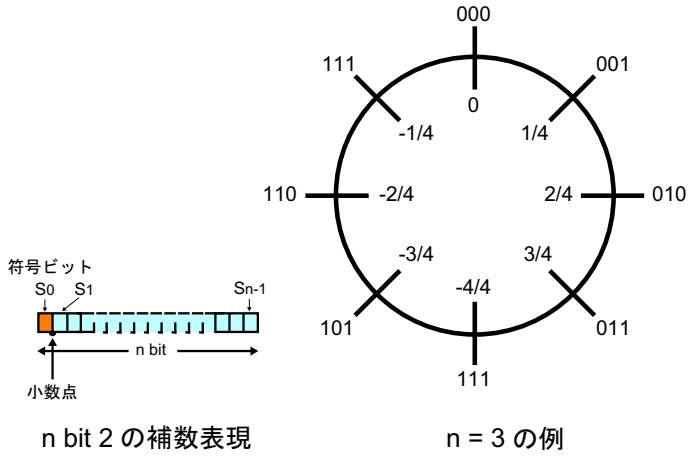


図 8･14　2 の補数表現

(3) 固定小数点積和演算アーキテクチャ

16 ビット固定少数点積和演算アーキテクチャの例を図 8･15 に，2 の補数表現の各データの形式を図 8･16 に示す．16 ビットの固定小数点データ同士（①，②）の乗算を行うと乗算結果は倍長の 32 ビットになる．このため，32 ビットの乗算器出力（③）をすべて利用した積和演算では少なくとも 32 ビットの加算器及びアキュムレータレジスタ（④）が必要になる．一方，アキュムレータレジスタ（④）上のデータを再度 16 ビットのメモリに格納する場合（あるいは乗算器の入力とする場合），シフタまたはリミッタ（⑤）による飽和演算，スケーリング，及び下位ビットの切捨て，あるいは丸め処理などを行って 16 ビットのデータとする．

(4) 演算オーバーフロー対策（ヘッドルームと飽和演算）

積和演算を繰り返して行う場合，32 ビットの乗算結果を繰り返して加算することにより，積和演算結果を 32 ビットでは表現できなくなり，演算オーバーフローが発生する．オーバーフローが発生すると正しい結果が得られなくなり，システムが不安定になるなどの問題が生じるため以下の対策を行う．

(a) ヘッドルーム（ガードビット）

加算器及びアキュムレータレジスタにおいて，乗算結果の最上位ビットのさらに上位側に数ビット（4～8）のヘッドルームと呼ばれる追加ビット用意して 36 ビット～40 ビットで演算を行う．これにより，例えば 40 ビット演算を行う場合は，2 の 8 乗（256）回の繰り返し演算までオーバーフローしないことが保証される．

(b) 飽和演算（サチュレーション）

加算・減算を行う際に，オーバーフローを起こした場合は演算結果を正または負の最大値で置き換える（貼り付ける）操作を行う．これにより演算結果は必ずしも正しい値にならないが，近似的に正しい値となるため，オーバーフローが発生した場合（例えば正の大きい数

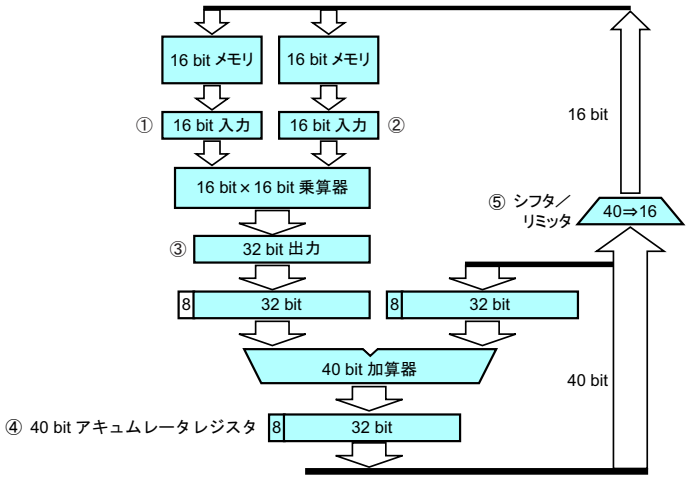


図 8･15　固定小数点積和演算アーキテクチャ

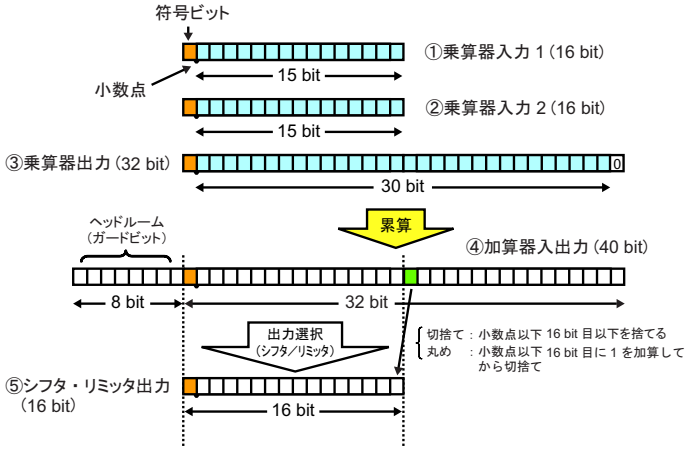


図 8･16　固定小数点積和演算における各データの表現

が負になることがある）の悪影響（発振等）が回避できる．特にヘッドルームがサポートされていない場合，飽和演算が多用される．

（c）スケーリングシフト

データ語長が n ビットの演算において，オーバーフローが発生することが予測される場合に，小数点の位置を m ビット（m＜n）分下位ビットにシフトして演算を行う．具体的には m ビット算術右シフト演算を行う，あるいはフィルタ係数の小数点の位置を m ビット下位に置いて（元のデータの 2 の-m 乗のデータを利用する）積和演算を行う（スケーリング）．また，乗算器の出力やレジスタの出力にシフタを置くことによりサイクル数を増やさずにシフトを実現する場合がある．ただし，スケーリングシフトした分だけデータの精度は失われるため，実際のプログラミングにおいては，実現しようとする信号処理アルゴリズムが必要とするダイナミックレンジと精度に応じて必要な対策の組み合わせが選択される．

（d）切捨て誤差と丸め誤差

乗算や加算の出力データ結果の語長は入力データの語長に比べて増加する．（一般に n ビット同士の乗算結果は 2n ビット，加算結果は n+1 ビット）．語長を一定に保つ必要がある場合，切捨て，あるいは丸めなどの量子化操作を行う．最上位ビットから n+1 ビット以下を捨てて n ビットとする操作を切り捨て（truncation）と呼び，最上位ビットから n+1 ビット目が 1 の時に n ビット目に 1 を加算し，0 の時はそのままで n+1 ビット以下を捨てて n ビットとする操作を丸め（rounding）と呼ぶ．図 8･17 は，横軸を入力，縦軸を量子化（切り捨て，丸め）結果として，2 の補数表現に対するそれぞれの量子化特性を示したものである．丸めによる量子化平均誤差は切り捨てに比べて小さく，また，切り捨てによる量子化平均誤差は負にバイアスがかかるのに対し，丸めによる量子化平均誤差は 0 になる．このため，丸めによる量子化が使われることが多い．

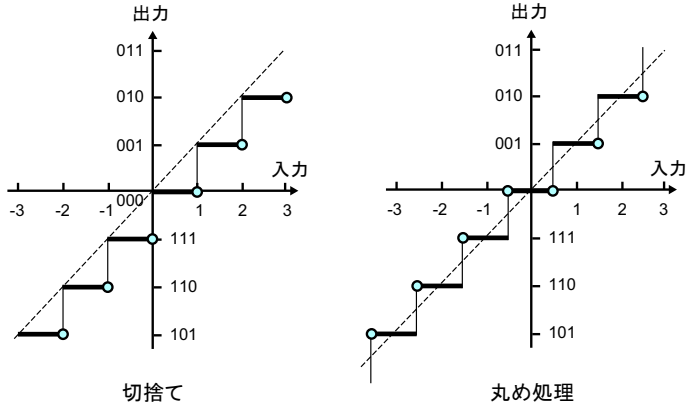


図 8･17　切捨て及び丸め処理による量子化特性

■参考文献

1) P. Lapsley, J. Bier, A. Shoham, and E.A. Lee, “DSP Processor Fundamentals,” IEEE Press, New York, 1997.

2) R.G. Lyons, “Understanding Digital Signal Processing,” Prentice Hall, 2004.