組み込み技術者のためのオープン・ソースによるDSPアルゴリズム開発手法

第3回　信号処理アルゴリズムのデータ構造

この連載では，ディジタル信号処理プロセッサ（DSP）の上で動作するソフトウェアのアルゴリズムを開発する方法を解説している．前回は，DSPソフトウェアの開発方法などを紹介した．今回は，DSPのファームウェア開発におけるメモリの使い方や容量，管理方法について説明する．

（編集部）

冴木 元

● DSPで使うメモリについての考え方

今回は，DSPのファームウェア開発において重要な，メモリについての考え方を解説します．

メモリは単に「合計でどれだけ必要なのか」が問題なのではなく，「どのような使い方をする領域が，どれだけ必要なのか」が重要です．メモリの管理方針は，DSPのアルゴリズムに対する考え方と関係があります．

ここでは，DSP特有の演算処理（積和演算）で必要となるデータ構造にさかのぼって考えます．それによって，DSPの開発全般ではどのようなメモリ管理が必要となるか，管理方法はどのようなものなのかが明らかになります[注1]．

DSPはもともと，ある値を持つ係数と，毎フレーム変化する変数を掛け合わせ，結果を足していく積和演算が得意なアーキテクチャを持っています．

近年では純粋な信号処理以外の応用にもDSPを使うようになりました．そのためDSPにはさまざまな命令が組み込まれ，使い勝手が向上しています．しかし最適化という観点からは，「（プログラムとして）積和演算をどう実装するか」ということが，現在でも重要です．

本連載の第1回（2008年1月号，pp.174-180）に書いたように，DSPや組み込みシステムの開発では，メモリ領域の大きさは限られています．パソコン上で動くアプリケーション開発のように，メモリ使用量の上限を気にしなくてもいいということは，DSPのアルゴリズム開発では考えられません．なお今回の話の大前提として，「OSにメモリ管理を任せられない」という，DSP特有の制約があります．

DSPのアルゴリズム開発において，簡易なOS環境が用意されることが多くなりました．これらのOSは，主にDSP上でタスク処理の便宜を図るものであり，メモリへの割り当てはユーザが管理しなければならないものが多いようです（今回採用した開発環境はそのような制約を持っている）．

● 信号処理の典型的なアルゴリズム

図1をご覧ください．これは，DSPの代表的な演算処理であるFIR（Finite Impulse Response ；有限インパルス応答）フィルタをブロック図にしたものです．「D」と表された部分は，保存されている過去の入力信号です．一番左の信号線が現在の信号を表しており，右に進むに従って，1単位時間前，2単位時間前…というように読みます．1単位時間というのは，実装では「フレーム」に当たるものです．

これに三角形で表した係数を掛け合わせ，それらを足し算していきます．現在の信号に一定の係数を掛け，1フ

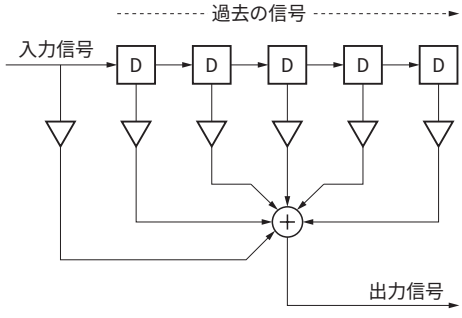
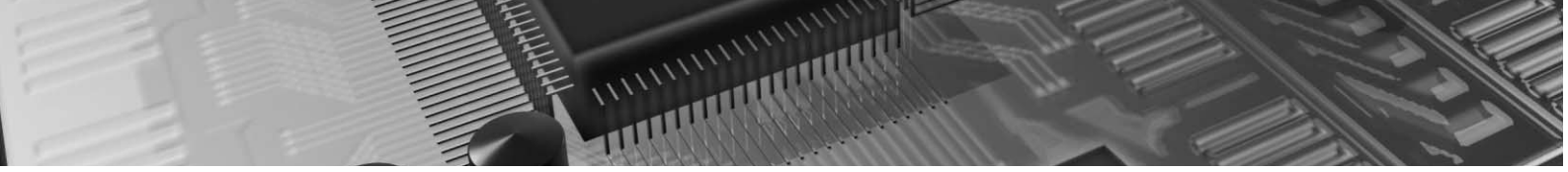


図1　FIRフィルタのブロック図

DSPのアルゴリズムとして，積和演算（掛け算を繰り返して足し込んでいく計算）が挙げられる．Dは保持しておいた過去の信号，三角形は係数．それぞれ掛けたものを足していく．

注1：Texas Instruments社では，ここに挙げたようなメモリ管理やAPIの呼び出し形式などを含めて，アルゴリズムのインターフェースを規定したDSP Algorithm Standard（XDAIS）が定められている．Code Composer Studioをインストールしたディクトリの下の¥docs¥PDF dsp_algorithm_standard.htmlを参照．

KEYWORD ── DSP，積和演算，static関数，グローバル関数，main関数，ローカル関数，スクラッチ・データ，パーシステント・データ，FIRフィルタ，CCS，CODEC，エンコーダ，デコーダ，オーバフロー



レーム前の信号に別の一定の係数を掛け，これらを足し合わせます．これは積和演算と呼ばれます．

以下では，これらの変数の扱いについて解説していきます．これらの変数の扱いは，DSPの組み込みソフトウェア開発において，最初に方針を確定するべき重要な事柄です．なぜなら，メモリ管理の方針の巧拙は，DSPの組み込みソフトウェアの品質だけでなく，性能も左右するからです．

● リセット時に初期化するパーシステント・データ

図1で「D」と書かれた部分は，前述のように過去の入力，すなわち数フレーム前の信号を保存している部分です．DSPのアルゴリズムは，例えば音声に関するものであれば160ワードごとに「フレーム」を設け，これを処理の1単位とすることが多いようです．

アルゴリズムを組み込んだ呼び出し元から呼ばれる関数は，このフレームごとに呼び出されてアルゴリズム処理を行います．従って，「D」のような種類のデータは，フレームをまたがってアルゴリズム内でデータを保持しておく必要があります．これらのデータは，ローカル変数のような値の保持が不要なデータと区別しなければなりません．これをパーシステント・データと呼びます．

パーシステント・データは，複数の関数呼び出しにまたがって内容を保持しなければならないため，C言語で書く場合には，static変数またはグローバル変数として宣言する必要があります．あるいはmalloc関数などを用いて，動的に領域を確保する設計方針をとる場合もあります．

パーシステント・データは値を保持し続けるので，必ずリセットのタイミングで初期化する必要があります(**図2**)．パーシステント・データは，仕様としてその初期値を定め，リセット用の関数において，初期値を設定する構成をとらなければなりません．

この初期値は「ゼロ」であることが多いのですが，いかなる値が初期値として望ましいかは，アルゴリズムの設計によって決まります．パーシステント・データの初期値は，設計段階で値を定めるべきなのです．

リファレンス・コードによっては，パーシステント・データは外部参照する形で与えられていることがあります．グローバル変数は外部参照可能なため，リンクされたどのファイルからも参照できるからです．

しかし，C言語を用いた開発において，外部変数の扱いには注意が必要です．リンクされたプログラムのどこからでも参照・更新可能であるという外部変数の性質は，プログラムの保守性を著しく下げてしまう恐れがあるからです．従って，外部変数の使用はできる限り限定するべきです．

● データの容量を調べ，領域を確保する

パーシステント・データ(**図3**)は，後述するスクラッチ・データ(**図4**)と同じように，一つの構造体を頂点とする階層構造の中にまとめ込むようにすると，ソース記述の際に便利です．まずアルゴリズムの開発元がパーシステント・データに必要なデータのサイズを明らかにします．処理系における構造体のサイズはそれが求める領域長となり

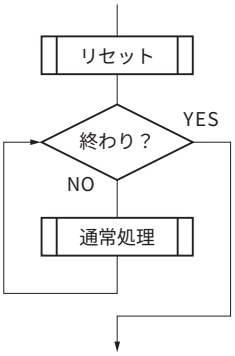


図2　リセット時の初期値について

値を保持するには「最初の値」，すなわち「初期値」が必要．アルゴリズムには必ず，最初の「リセット」とフレーム分繰り返す「通常処理」がある．パーシステント・データを持つ限り，アルゴリズムには最低限，この2種類のAPIが必要．

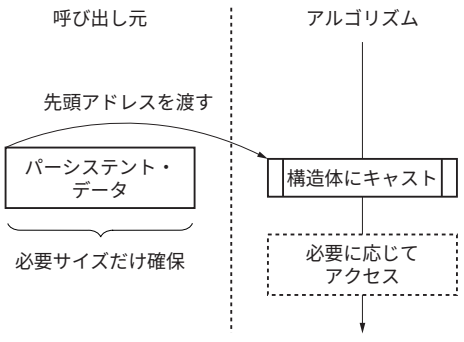


図3　構造体にまとめて管理(パーシステント・データ)

パーシステント・データは，外部変数として個別にリファレンス・コードによって与えられていることもあるが，C言語では混乱のもととなる．構造体にまとめて各データをたどれるようにしておけば管理しやすく，ソースの可読性も上がる．

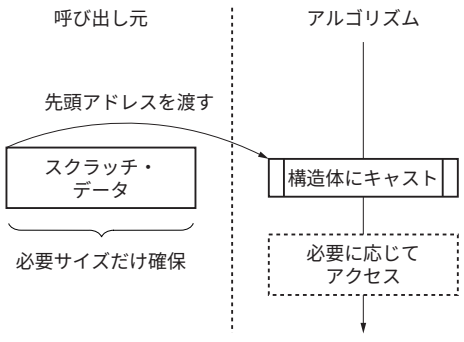


図4　構造体にまとめて管理(スクラッチ・データ)

スクラッチ・データも，パーシステント・データと同じように構造体にまとめる．アルゴリズムのデータ構造設計では，パーシステント用とスクラッチ用の二つの構造体が大きな役割を果たす．