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2回　なぜリファレンスが必要なのか

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アルゴリズムを開発するために必要な開発環境を紹介した．今回は，DSPアルゴリズムの開発の方法や，順序の全体をどうやって適切に組み立てていくかなどについて解説する．（編集部）

冴木 元

今回から，DSPに最適化したアルゴリズムの開発の方法や全体的な工程の組み立て方を解説します．

DSPのアルゴリズム開発についての書籍は，あまり多くありません．それらの書籍には，アルゴリズム全体についてというより，その中の部品である演算処理をDSPがどう扱うかということが主に書かれています．例えば，FFT（Fast Fourier Transform ；高速フーリエ変換）などの信号処理アルゴリズムにはどのようなものがあって，それらを高速処理する仕組みをDSPがどう整えているかなどです．

信号処理モジュールの高速実装は，実際のDSPのアルゴリズム開発において重要です．しかし，実際のDSPのアルゴリズムというのは，複数の信号処理モジュールが集合してできています．従ってもっと重要なのは，全体をいかにして適切に組み立てていくかということです．

全体を組み立てるとは，一言で言えば適切な設計が必要になるということですが，そのためのテクニックが一貫した観点のもとで語られることはほとんどないと思います．本連載は，そのためのたたき台となるようなものを用意していきたいと思っています．

● リファレンス・コードの存在

前回（2008年1月号，pp.174-180）の最後に少し触れましたが，DSP上で動くアルゴリズムを実装する際には，それに先立って，パソコンや汎用機上で実際に動いて正しい処理結果を出力してくれるようなコードをあらかじめ用意しておきます．これを「リファレンス・コード」と呼びます．

図1　リファレンス・コードとその移植

DSPのアルゴリズム開発をする際の大きな特徴は，アルゴリズムの実装であるリファレンス・コードがパソコン（ないし汎用機）で動く形で既に用意されていること．従って，開発を始めるに当たっては「いかにして移植するか」を初めに考える必要がある．

DSPアルゴリズム開発というのは，まずこのリファレンス・コードの移植作業からスタートします（**図1**）．最初の目標は，パソコンで動くコードの品質を保ったままDSP上で動かすことです．DSPのアルゴリズムの高速実装（最適化）は，この移植作業が一段落した次の段階で，本格的な問題になります．もちろん，移植作業を行う上で，後に最適化しやすいように工夫する必要があります．また，移植作業の中で省ける処理があれば省きます．

移植作業を進めるにあたり大切なことは，移植した先のDSPで動くコードが，元のリファレンス・コードと等価であるということです．移植した結果，元のリファレンスと別物ができあがったのでは元も子もありません．ここで問題となるのは，この「等価」をどうやって判断するかです．

● ビット一致が動作確認の基本

元のリファレンスとDSPの実装が「等価」であると確認する方法は至ってシンプルです．二つのコードがまったく同じ動きならば，同じ入力を同じ動作パラメータで与えると，両者は全く同じ結果を出力するはずです．つまり，リファレンスとDSP実装の等価性を示すためには，さまざまな入力データと動作パラメータを与え，出力が同じであることを確認するテストを行えばよいのです．リファレンスと実装の出力を共にファイルに出力して保存しておけば，何らかのツールを用いて比較できます．完全に同じ出力であれば，1ビットも違わずに一致するはずです（**図2**）．「ビット一致」という言葉は，DSPアルゴリズム開発の世界では「テスト結果OK」とほとんど同義です．

● パソコン上での改造が鍵となる

上記の話より，移植作業を開始するには，開発の元とな

KEYWORD ── DSP，リファレンス・コード，実装コード，アセンブリ言語，C言語，HDL，クロス環境，ネイティブ環境

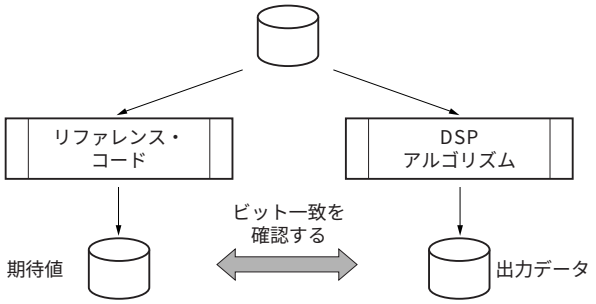

図2　リファレンス・コードとのビット一致が基本

DSPのアルゴリズムについて，リファレンス・コードと等価なコードを作ることが目標である．そのため，同じ入力に対して常にリファレンスと同じ出力をもたらす，という確認を繰り返す．「ビット一致」というのは，「テスト結果OK」とほぼ同じ意味で用いられる．

るリファレンス・コードをいかにしてDSPの実装に適したものにするかが重要になります(**図3**)．具体的には，

1）データ構造の確立

2）演算精度の実装検討

の2点がポイントとなるのですが，詳しい中身については，連載の中で順番に解説していきたいと思います．

● なぜリファレンスが必要なのか

さて，組み込みソフトウェアの開発に携わっている技術者の中にはDSPのアルゴリズムを実装した経験がない方も大勢いらっしゃると思います．そういう方は「同じプログラムなのにリファレンスと実装を別々に用意する必要性が分からない」と思うかもしれません．あるいは「DSPで動かなければ製品化できないなら，パソコン上で方式検討するときからDSPの実装を意識すれば余計な工数が省けるのではないか」，「リファレンスと実装コードをわざわざ別に用意するのは大げさなのではないか」と思うかもしれません．

リファレンスと実装を統合するのは，現在のDSPの開発環境では無理です．なぜなら，リファレンスと実装を両方こなすには，二つの役割を同時にこなせる人が必要だからです．すなわち，

1）方式検討の専門知識を持った研究者

2）移植の専門知識を持ったエンジニア

残念なことに，大抵の人は1)と2)の両方の能力を兼ね備えてはいないのです(**表1**)．

● 方式検討には専門知識が必要

DSPで実装するアルゴリズムは，MPEGなどのように規格が国際団体で決まっている場合もあれば，メーカ独自でアルゴリズムを一から開発する場合もあります．これらは，最初は紙の上で理論的に検討されていますが，最終的にはC言語などのプログラムの形に直され，シミュレーションを行って効果を確かめる必要があります．いずれの場合も，ディジタル信号処理という工学分野に精通した人でなければC言語の形にはできません．そして，作ったC言語のプログラムが「正しいかどうか」を判断するにも当然工学的な専門知識が必要です(**図4**)．「方式の検討」と呼ばれる作業工程は，その方式をシミュレートするC言語のプログラムができたところで終わるのですが，この作業には，通常のソフトウェア開発とは異なる知識が必要です．

表1　方式の検討と実装の比較

	方式の検討	実装
目的	アルゴリズムが目的を満たしているかを検討する	製品に適用するためにDSP上に実装する
環境	パソコン	DSP
言語	C言語	C言語，アセンブリ言語

● DSPへの移植には専門知識が必要

アルゴリズムをDSPに実装する場合，

1）パソコン(CPU)ではなくDSPで動くように移植する

2）製品として使えるように最適化する

という二つの目的を満たす必要があります．1)の移植作業は，DSPにコンパイラがなかった時代であれば，一つ一つの関数をアセンブリ言語に置き換えていく作業でした．コンパイラが使えるのであれば，C言語による実装もDSP上で可能となります．しかし，そのためには，DSP開発特有の条件を満たしたコードを整える必要があります．例えば，前回解説した「メモリ・マップ」に応じてプログラムやデータの配置を整えることなどもそれに含まれます．

製品としてアルゴリズムを実装する場合には，単に正しく動作させることに加えて，「最適化」という2)の作業が

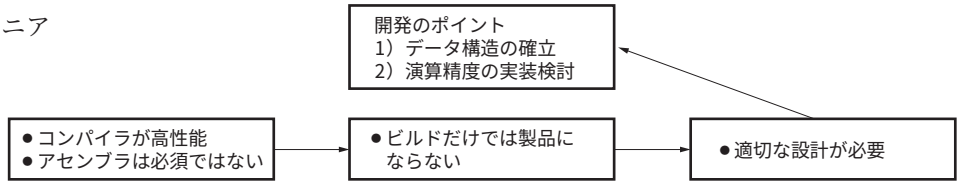

図3　リファレンスの改造が移植作業のポイントになる

現在のDSPのコンパイラはかなり高性能で安定しているため，昔のように関数一つ一つをアセンブリ言語化する必要はない．しかし，リファレンス・コードをそのままビルドしただけではいけない．リファレンスをDSPの実装としてふさわしい形にいかに改造するかがポイントとなる．