第1章

DSPの歴史，汎用CPUとの違い，C++言語のメリット

DSPとC++言語

DSP(digital signal processor)[注1]とは，ディジタル信号処理向けのマイクロプロセッサである．携帯電話に代表されるように，今日ではディジタル信号処理を応用した機器は広く使われているが，その原動力になっているのがDSPである．したがって，DSPは今日のようなIT時代を支えるキー・デバイスだと言っても過言ではないであろう．

本格的な商業ベースのDSP[注2]が最初に世の中に現れたのは1980年代の初期のことで，NECのμPD7720[(1)]（**写真1**，**図1**）が世界で最初である．それ以来20年以上が経過し，DSPの性能も飛躍的に向上している．そのため，初期のころには想像もできなかったような分野にまでDSPの応用範囲が広がっている[注3]．

従来，DSPのプログラミングは難しいと言われていた．その理由はいろいろあるが[注4]，アセンブリ言語を用いなければならないということもその一つであった．しかし，現在ではC言語やC++言語などの高級言語を使える環境も整い，プログラム開発もVisual C++のような統合開発環境が整ってきたので，ひところとは違ってDSPのプログラミングもずいぶん楽になっている．

従来のDSPのプログラミングは，いかにして実行効率を上げるかということに主眼が置かれていた．しかし，最近ではDSPの性能が大幅に向上したため，プログラムの実行効率は多少悪くても，プログラムの開発効率を向上させるほうがより重要な課題だという声もよく聞く．そのような観点からすると，C言語よりはC++言語を使ってプログラミングを行ったほうが開発効率を向上できると思われる．

本書では，Texas Instruments社(以下，TI社)から発売されているDSPであるTMS320C6713を搭載したDSPスタータキット(DSK)を使った，C++言語によるプログラミングについて解説を行っていく．

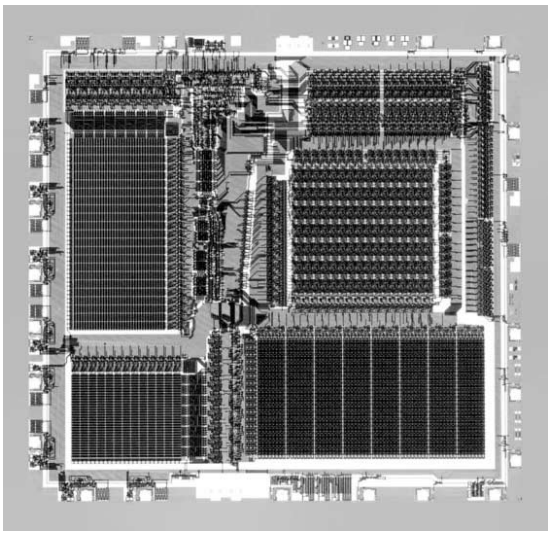


写真1　世界最初の商業ベースのDSPであるμPD7720の回路パターン(提供：NEC)

注1：DSPは，ディジタル信号処理(digital signal processing)という意味で使われることもあるが，本書では，「digital signal processor」という意味で使う．

注2：1970年代の終わりごろには，Intel社やAMI社からDSPが発表されているが，いろいろな面で，まだ本格的なDSPとは言い難い状態であった．

注3：DSPの応用分野は広く，とても書ききれないのでので，次に示すDSPの代表的なメーカのWebサイトなどを参照してほしい．

- 日本テキサス・インスツルメンツ
  `http://www.tij.co.jp/jsc/docs/dsps/index.htm`
- アナログ・デバイセズ
  `http://www.analog.com/processors/Japan/index.html`

注4：本質的な原因は，もちろんディジタル信号処理の理論を理解していないということが大きい．

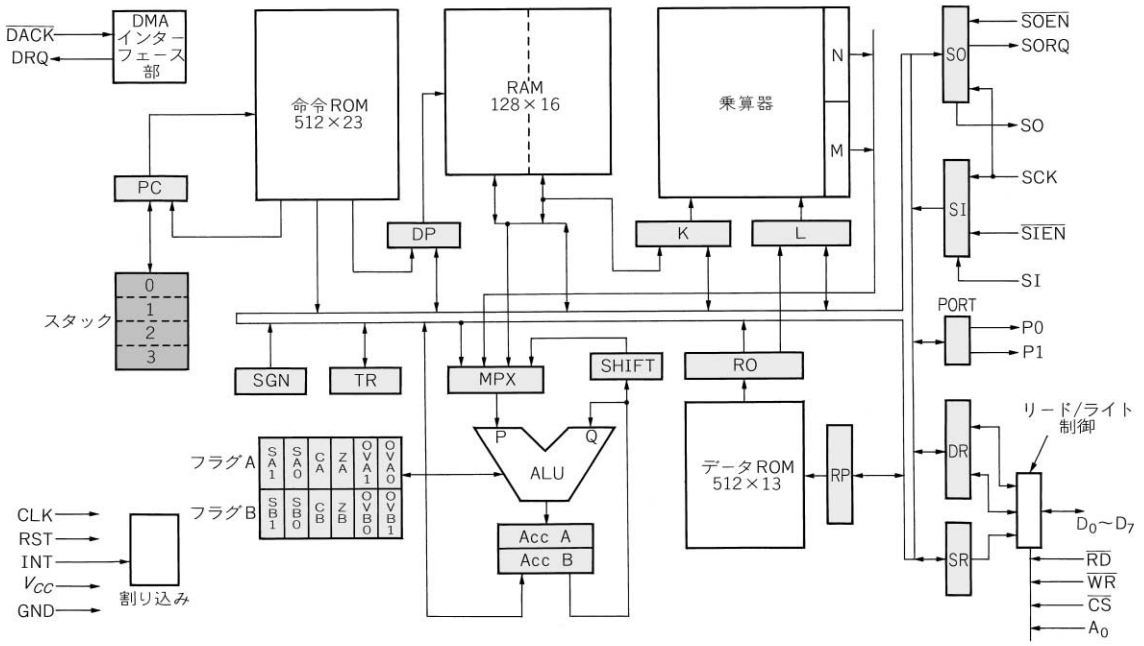


図1　μPD7720のアーキテクチャ

1 DSPと汎用CPU

DSPが開発された当初は，とくにディジタル・フィルタのリアルタイム処理を念頭に置いて考えられていたということが，そのころのユーザーズ・ガイドからもうかがえる(2)．ディジタル・フィルタの処理では，加算と乗算を数多く使うので，リアルタイム処理として実現するためには，高速の演算器，とくに乗算器が不可欠である．

ところで，1980年代の初めごろの汎用CPUでは，乗算にかかる時間はどのくらいであったかというと，当時の汎用CPUとして代表的なIntel社の8086で，約20μs程度であった．一方，たとえば電話帯域の信号処理を考えると，標本化周波数は8kHzなので，実時間処理を行うためには125μsの標本化周期の中ですべての処理を完了する必要がある．ところが，8086ではこの時間内に乗算を6回しか実行できないため，一つの汎用CPUだけでは実用的なディジタル・フィルタを実現することは難しかった．

TI社の最初のDSPであるTMS32010(2)は，高速の乗算器を内蔵しており，乗算を200nsで実行できた．したがって，かなりのディジタル・フィルタ処理を実時間処理で実行できるようになった．

このように，DSPが世の中に出てきた当時は，乗算を高速に実行できることが特徴であった．そのため，その乗算器を有効に働かせるため，乗算器にデータを転送するスピードを上げるようなくふうが各所に取り入れられ，汎用CPUとはかなり異なったアーキテクチャであった．たとえば，改良型ハーバード・アーキテクチャ[注5]を採用して複数のバスをもたせたり，並列動作を取り入れたり，クロックを上げるためパイプライン処理を取り入れたりといったことが行われていた．

しかし，LSIの集積度が急速に向上するにつれて，汎用CPUにも高速化のためにパイプライン処理を取り入れ，高速の乗算器などを内蔵するようになり，複数の演算器で並列処理することが当たり前の時代になってきた．

一方，本書で取り上げているTMS320C6713を含むTMS320C6000シリーズのDSPの命令体系を見ると，演算は基本的にレジスタ間で行われ，メモリにアクセスするのはロード(load)命令とストア(store)命令だけになっている．これはRISCマシンが採用するロー

注5：ハーバード・アーキテクチャとは，メモリ空間がデータ用とプログラム用に分離しているようなアーキテクチャで，そのためバスもデータ・バスとプログラム・バスに分離されている．

ド・ストア・アーキテクチャと同じである．また，従来のDSPがもっていた，加算と乗算を同時に行うような複合処理命令ももっておらず，RISCマシンのような単純な命令セットしかもっていない[注6]．そのため，筆者が最初にTMS320C6000シリーズのDSPの命令体系を見たときに，これはDSPというよりも，むしろRISCマシンではないかという感想を持ったくらいである．

このように，汎用CPUもDSPもそれぞれお互いの長所を取り入れて発展してきており，現在ではあえて両者を区別する必要はないと思われる．

2 DSPのプログラミング——アセンブリ言語からC言語へ

初期のDSPはスピードが命だったので，当然ながらアセンブリ言語を使ってプログラミングを行っていた．それには，DSPの内部まで詳しく知る必要があり，それがDSPのプログラミングを行う際の大きな壁として立ちはだかっていた．その対策として，従来からDSP用のCコンパイラも発表されていたが，その当時のC言語で開発したプログラムは，アセンブリ言語で開発したものに比べると実行効率が非常に悪かったため，どうしてもアセンブリ言語に頼らざるをえない状態であった．

しかし，DSPの構造が複雑になり，とくにTMS320C6000シリーズ(以下，C6000シリーズ)のDSPのようにVLIW(very long instruction word)アーキテクチャを採用するDSPが現れるようになり，事情は変わってきた．

C6000シリーズは付録Bに示したように，改良されたVLIWアーキテクチャを採用している．そのため，機能ユニットと呼ばれる演算などを行う種類の異なるユニットを8個備えており，最大で八つの命令を同時に実行することができる．さらにC6000シリーズはパイプライン処理を採用している．その関係で，分岐，ロード，ストアの各命令が遅延分岐，遅延ロード，遅延ストアになっている．以上のことから，アセンブリ言語でプログラミングを行う場合，そのようなことをすべて把握して実行速度が最大になるように最適化することは，使用するDSPについて相当熟練しないかぎりは無理であるといってもよい．

しかし，C6000シリーズはC言語によりプログラムを開発するという前提で設計がなされている[(3)]ため，C言語でプログラムを書いても，アセンブリ言語で書かれたものに匹敵する実行スピードが得られるようになった[(3)]．そのため，DSPのもっている性能をぎりぎりまで発揮させようというような場合を除いて，多くの場合において，DSPのプログラムもC言語で開発を行う時代になってきた．

3 C++言語を使う理由

本書のプログラムはC言語ではなくC++言語を使って書いている．なぜC++言語を使うかというと，C言語に比べてプログラムを作りやすく，開発の効率も上がるからである．その大きな理由は，C++言語がクラス(class)という言語要素をサポートしていることによる．

ここで，クラスを使えることのメリットを挙げてみよう．

● 初期化とコンストラクタ

ディジタル信号処理では，多くの場合に処理に先立って初期化が必要になる．たとえばFFT(高速フーリエ変換)を行う場合，計算で使う作業領域を確保することや，処理の中で使う数表をあらかじめ計算しておくという初期化が必要になる．その後，FFTの処理を行うことになる[注7]．

このような処理の場合，C言語ではFFTの初期化のための関数と，FFTを実行するための関数は互いに関連なく作成するため，FFTの初期化を忘れたまま実行するようなプログラムを書いてしまうということが起こりうる．また，すでに初期化されているにもかかわらず，まちがって再び初期化するというようなプログラムを書いてしまうこともある．

しかし，C++言語でクラスを使ってFFTを作成すると，そのクラスのコンストラクタ(constructor)が初期化を行い，FFTの実行はそのクラスのメンバ関数(member function)が行うようにすることができる[注8]．コンストラクタはクラスのオブジェクトを宣言したときに自動的に実行される．そのため，クラス

注6：正確にはTMS320C62xx，67xxは単純な命令体系といえるが，改良されたTMS320C64xxなどは通信や画像処理専用の命令も含むようになった．そのなかにはSIMD(single instruction stream, multiple data stream)型の命令も含んでいる．

注7：場合によっては，いらなくなった領域をシステムに返却するという後処理も必要になる．

注8：後処理が必要な場合は，そのクラスのデストラクタ(destructor)に自動的に行わせることができる．

のオブジェクトが宣言されていないかぎりはFFTを実行するようなプログラムは書けないことになり，初期化を忘れることを確実に防止できる．また，再度初期化を行うというプログラムも書けないことになる．

● 演算子の多重定義

クラスを使えば，演算子の多重定義(operator overloading)を行うことができる．たとえば，ディジタル信号処理では複素数の演算を行うこともある．その場合，演算子の多重定義を使えないとしたら，関数を使わざるをえない．そのため，プログラミングが非常にやっかいになり，可読性も悪くなる．

例として，次のような多項式をHornerの方法で計算することを考えてみよう．

$$a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 \quad \cdots\cdots (1)$$

二つの複素数の加算を行う関数を`Add()`，乗算を行う関数を`Mpy()`とすると，式(1)の計算は次のようになる．

```
Add(Mpy(Add(Mpy(Add(Mpy(Add(Mpy(a4,
    x), a3), x), a2), x), a1), x), a0);
```

一方，複素数の加算/乗算に対する演算子が多重定義されていたとすると，同じことを次のように書くことができる．

```
(((a4*x + a3)*x + a2)*x + a1)*x + a0;
```

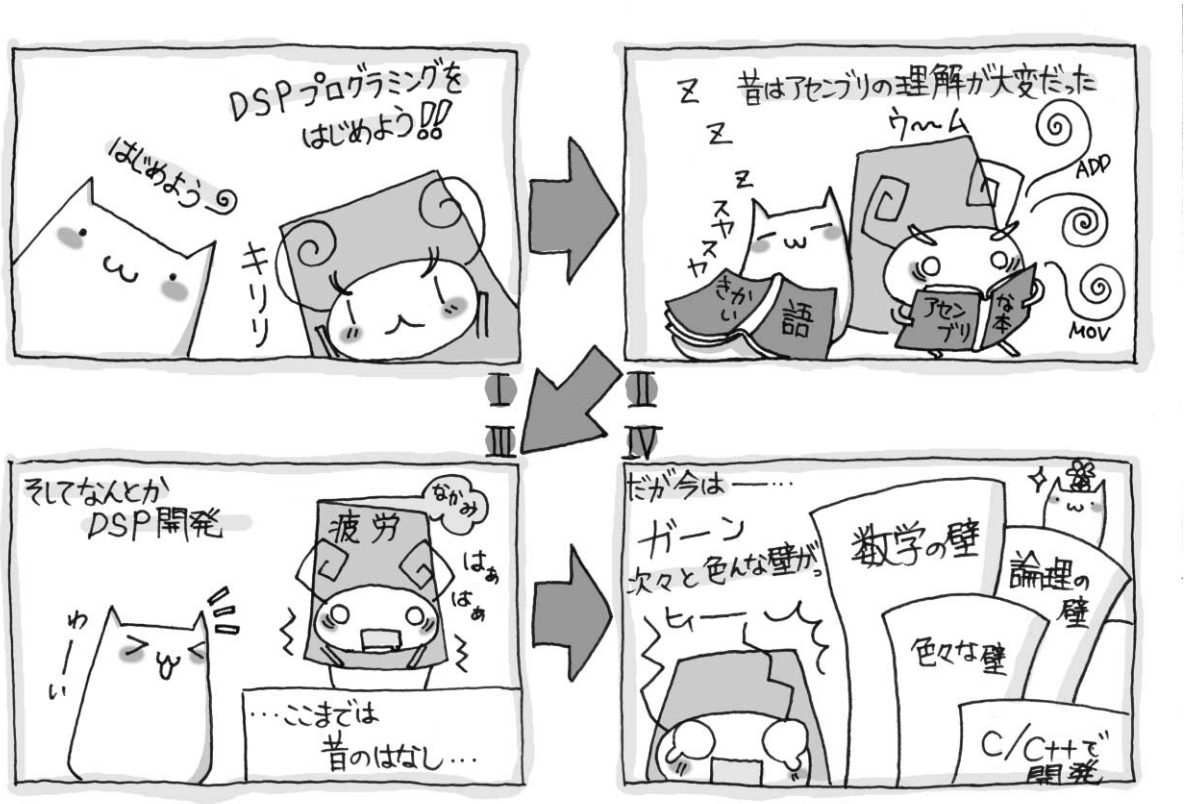
このように，演算子を使って書いたほうが簡単で，しかもわかりやすいことは改めて説明するまでもないであろう．

● 仮想関数

クラスを使うと，仮想関数が使えるということも大きなメリットである．たとえば，A-D変換器のデータを読み込む関数を作る場合，ハードウェアが異なっても，同じ名前の関数〔たとえば`Read()`という関数〕で呼び出すことができれば，プログラムの開発が楽になる．このようなことは，クラスの継承と仮想関数を使って実現することができる．つまり，基底クラスの中で，メンバ関数を仮想関数として定義しておき，基底クラスを継承する派生クラスの中で，この仮想関数を再定義(overriding)することで，同じ名前で異なった機能を実現することが可能になる．

● 情報の保護

クラスを使うと情報の隠蔽(いんぺい)ができるということも安全なプログラムを作成するうえで非常に役に立つ．これは，外部からクラスの内部へのアクセスを禁止でき，



クラスの内部だけで使うようなデータ[注9]や関数[注10]を定義できるからである．そのため，クラスの外部からの干渉や，外部からまちがった使い方をするというようなことから，クラスの内部を保護することができる．

＊　　＊　　＊　　＊

クラスとは関係のない機能の中にも，C言語にはなくてC++言語がもっている便利な機能がある．関数の多重定義の機能を使えば，関数名は同じであって，引き数が異なるという関数を作ることができる．また，関数テンプレートの機能を使えば，一つの関数の定義だけで，引き数が変わってもそのまま使うことができる関数を作ることもできる．

ところで，この節の最初でC++言語を使う理由はプログラムを作りやすいことにあり，それはクラスがサポートされているからだと書いた．ただし，すでに存在するクラスを使ってプログラムを作ることは非常に楽ではあるが，クラスそのものを作る，それも適切なクラスを作るということは，場合によっては非常にたいへんなことであるということに注意しなければならない．したがって，クラスを作ること自体が目的である場合を除いて，クラスを自分で作る場合は，まずトータルに見て，それが本当にプログラムを作る効率を上げることになるのかどうかということを見きわめなければならない．

注9：このようなデータをデータ・メンバという．
注10：このような関数をメンバ関数という．

4 C++言語とディジタル信号処理に関する文献

本書は，読者がある程度C++言語を知っているという前提で書いているので，基本的にC++言語についての解説は行わない．しかし，C++でプログラムを開発するうえで重要な点については，コラムという形で説明するようにした．そのため，C++言語をあまりよく知らない読者には，C++言語についての参考書も手元に置いておくことを勧める．

参考までに，筆者が本書を書くにあたって，よく参考にしたC++言語に関する文献を参考文献(4)～(8)に示しておく．

また，ディジタル信号処理についても同様で，読者にある程度の知識があるという前提で書いている．これについても説明のつごう上，どうしても知っておいて欲しい点については本文で最低限の説明を行っている．また，概略を知る場合には読み飛ばしてもかまわないが，知っていればよりよいと思われる事項については，コラムの形で示した．そのため，ディジタル信号処理についてあまりよく知らないという読者は，参考文献(9)や(10)などを参考にしてほしい．

参考文献

(1) 丸田 力男，西谷 隆夫；シグナルプロセッサとその応用，昭晃堂，1988年．
(2) TMS32010 User's Guide，文献番号：SCJ1104，日本テキサスインスツルメンツ，1984年．
(3) Ray Simar；米TIのDSP，MPUに先駆けて最大8命令のVLIWを採用，日経エレクトロニクス，1997年6月16日号，pp.135-146，1997年．
(4) ポール・アンダーソン，ゲイル・アンダーソン；最新ANSI C++オブジェクト指向プログラミング，ピアソン・エデュケーション，1999年．
(5) ハーバート・シルト；独習C++，第3版，翔泳社，2002年．
(6) M. クライン，G. ロモウ，M. ギウル：C++ FAQ 第2版，ピアソン・エデュケーション，2000年．
(7) 大城 正典；詳細C++ 第2版，ソフトバンク・パブリッシング，2005年．
(8) B. Stroustrup；プログラミング言語C++ 第3版，アジソン・ウェスレイ・パブリッシャーズ・ジャパン，1998年．
(9) 三上 直樹；はじめて学ぶディジタル・フィルタと高速フーリエ変換，CQ出版社，2005年．
(10) A. V. Oppenheim，R. W. Schafer，J. R. Buck；Discrete-time signal processing Second Edition，Prentice-Hall，1999年．