解　説

1. DSP の特徴と基礎理論

1.3 DSP のプログラム開発環境†

小　野　定　康††

1. まえがき

DSP は他のマイクロプロセッサと同様にプログラムによりその機能が定まるものである．したがって DSP を用いてなんらかの機能を実現するにはプログラムが必ず必要になる．DSP のプログラム開発は現在の時点では，あまり容易なものではない．このプログラム開発の難しさのために現在までのところ DSP は，DSP を用いなければならない特別の理由をもつ人たちだけの間で使われているにすぎない．価格の点では特に高価なマイクロプロセッサではなく，むしろ安価なマイクロプロセッサである．それにもかかわらず，あまりユーザの数が増加しない原因はプログラム開発の困難なことにある．

プログラム開発が困難な主な理由は次の3点である．

（1） DSP の命令セットがきわめて低レベルのものである．

（2） DSP のプログラムはリアルタイム処理を要求されることが多い．

（3） DSP のプログラム開発環境はあまり優れたものではない．

こうしたプログラム開発の困難さの問題があるにもかかわらず，DSP は着実にその適用分野を広げ，ユーザを増やしつつある．その理由はやはり DSP のもつ演算能力の大きさの魅力にある．現在市販されている代表的な DSP の演算能力はピーク値で 10〜40 MFLOPS（固定小数点 DSP の場合は MIPS or MOPS）であり，この値は最近の RISC プロセッサもしくはその数値演算プロセッサよりも1桁上の値である．しかも DSP は基本的にスタンドアロン動作をするように設計されているから，システムのハードウェア規模は他のマイクロプロセッサの数分の一ですむ．消費電力も同様であることから，さまざまな分野で利用され始めているのである．このことを反映して国内外の数社からさらに高い性能の DSP が開発，販売されつつある[1),6),7)]．

しかし DSP の性能が上がり，価格が下がり，適用分野が広がり，ユーザ数が増えるにしたがって，プログラム開発の困難さは前にも増して問題になってきている．このプログラム開発の問題が DSP の普及の鍵になってきている．

ここで誤解のないように明言しておくことがある．DSP のプログラム開発は確かに汎用のマイクロプロセッサのプログラム開発に比較すればそのプログラム開発効率はきわめて悪い．しかしこの比較自体にそれほどの意味はないのである．なぜならば DSP の代わりに汎用マイクロプロセッサを用いることは，演算能力の違い，ハードウェア規模などの点からできないからである．したがって代用できないものを比較してもあまり意味はない．むしろ比較の対象は代用しうるゲートアレイなどによる専用 VLSI との開発時間・工数の比較がもっとも適切である．VLSI の設計は CAD システムの発達により容易になりつつあるが，まだかなりの開発時間・工数が必要であるので，この開発時間・工数の比較において，DSP は専用 VLSI よりも約1桁少ない．この約1桁少ない開発時間・工数が現在までさまざまな DSP が開発され続けてきた主な要因である．システムの開発費用もだいたいこの程度の違いがあると考えてよい．

汎用マイクロプロセッサと DSP のプログラム開発の比較自体にあまり意味がなくても，DSP のプログラム開発を容易にして汎用マイクロプロセッサのプログラム開発に近づけることには大きな意味がある．それは DSP のもつ高速な演算処理能力をより広い応用分野に適用することが可能となるからである．

事実，現在の DSP のプログラム開発環境はこうし

† DSP Program Development Environment by Sadayasu ONO (NTT Transmission Systems Laboratories).

†† NTT 伝送システム研究所

た方向に向かって進展しつつある．特に最近はマルチメディア対応のパソコンの出現で，こうした方向のプログラム開発環境の整備が加速されつつある．

2. DSP プログラム開発の困難さ

2.1 背 景

DSP プログラム開発が困難な理由は先に述べた(1)～(3)に尽きるのであるが，こうした状況が作られた背景を説明する．

まず(1)であるが，これは DSP においては何よりも演算器の処理速度が優先され，それに加えて1チップのマイクロプロセッサとして実現しなければならないための使用トランジスタ数に強い制限があることから生じた結果である．ダイナミックアーキテクチャの実現などを目的として，積極的な意図をもって導入されているマイクロプログラム制御（＝低レベル命令）ではない．命令レベルを低くすることにより使用素子数の低減が実現できるために採用されているにすぎない．

具体的にいえば，使用素子数の強い制限のもとで，その多くの素子数をパイプライン演算器の高速化に割り当てて，その残りのあまり多くない素子数でプログラム制御を実現しようとした結果，命令レベルを低くしてプログラム制御を実現する以外の方法がなかったということである．

いうまでもなくこのことは，DSP の命令セットがハードウェアの細部の動作まで熟知していなければ理解できない難解な命令セットとなり，それがプログラム開発の効率低下に大きく影響している．

次の(2)のリアルタイム処理を要求されることは信号処理のもつ固有の性質に根ざすものである．つまり信号処理はリアルタイム処理に限らず，処理結果を得るのは速ければ速いほどよい，もしくはより有効に利用できる，という一般的な傾向がある．それが決められた時間内にあるデータに対する処理が確実に終了するというリアルタイム性を保証される場合には，この有効性はさらに大きなものとなる．またこの性質ゆえに応用される分野も少なくない．

ところが一般にリアルタイム処理を実現するプログラムはもっとも難しいプログラムに属する．それはプログラムの論理的な正しさだけではなく，それに加えて時間経過を考慮したプログラムの論理的な正しさが要求されるからである．簡単にいえば，処理結果が正しくても要求された時間をオーバしたり，途中での割り込み処理の影響を受けたり，データ（現時点のデータだけではなく過去のデータの影響も含めて）の値に依存しないことが保証されなければならないからである．

リアルタイム処理を実現するプログラムを比較的容易に開発するための手段として，リアルタイム処理用 OS（モニタ）を利用する方法がある．しかしこの方法も DSP ではメモリ空間の制約，DSP の命令セットが OS に適していない，オーバヘッドとして負荷が重すぎるなどの理由で DSP プログラムには採用できない．

いずれにしても DSP に対するリアルタイム処理の要求は回避できないし，決定的に有効な解決手段もないのが現状である．

(3)は現在までの DSP が置かれた状況が大きく関係している．それは DSP のユーザがあまり多くなく，かつこれらのユーザはプログラム開発環境の整備に関心が低かったことである．ユーザの関心はプログラム開発環境よりも，より高速の演算処理能力をもつ DSP に注目してきた過去の経緯がある．もちろんこれにはこれまでの DSP のユーザは少数ではあるが，その技術レベルがかなり高いエキスパートであるという事情が存在している．これらのユーザにはプログラム開発環境の整備によるシステム開発期間の短縮よりも，演算処理能力の向上による小形化，経済化のほうが重要であり，それゆえに整備されたプログラム開発環境はあまり魅力とはならなかった．

また技術的にも DSP のプログラム開発環境の整備は汎用のマイクロプロセッサに較べて多少難しい点もあり，優れたプログラム開発環境の整備は遅れている．

技術的な点は次の3点である．第1点目は，DSP が非常に高速な動作をするパイプライン演算器を内部にもつプロセッサであるために，プロセッサの外部にオンラインデバッガと呼ばれる回路を付加して，内部の状態を観測し，制御することが構造的にも電子回路的にも容易でないことである．第2点目は DSP の命令セットの低さとこれに直接対応するアセンブリ言語のみが主なプログラム言語となっている点である．後で述べるが，DSP用コンパイラにまだあまり実用性がないことである．第3点目は DSP プログラム開発のための効果的な方法が確立されていないことである．

2.2 現 状

現在の DSP プログラムに使われている言語は，(マイクロ) アセンブリ言語である．マイクロに ( ) を付けたのは DSP は通常の意味のマイクロプログラ

ムではなく，単に命令セットのレベルの低さ，具体的に言えばプログラムから直接データパスのセレクタ，ゲートを指定するという意味のマイクロ制御命令にすぎないからである．DSP のマイクロ制御命令には通常の ISP (Instruction Set Processor) レベルの命令の下位に位置し，ダイナミックアーキテクチャの実現のための手段という意味はない．事実ほとんどの DSP では単にアセンブリ言語と呼んでいる．

DSP のプログラム開発はこのアセンブリ言語を使い，汎用マイクロプロセッサのインサーキットエミュレータに相当するハードウェア（名称は DSP 各社でまちまちなのでオンラインデバッガと呼ぶ）を用いて行われている．このオンラインデバッガにより得られる情報は，ブレークポイントで DSP を停止させた状態でのレジスタファイル/メモリの内容が中心となる．

こうしたスタイルの DSP プログラム開発があまり効率のよいものにならないことは自明に近い．ただ DSP プログラム開発がこうした初等的なものでもそれなりの価値があったのは，布線論理の同一機能のシステムを開発するために必要な開発時間・工数と比較すると，このような DSP プログラム環境でも DSP を用いたほうがはるかに少ない開発時間・工数ですむからであった．

またこうしたスタイルのプログラム開発でもライブラリプログラムの充実，活用などによりそれなりの開発効率の向上が図れたことも事実である．したがって各 DSP メーカはプログラム開発環境の整備項目のうち，ライブラリプログラムの整備・拡充には継続的なサポートを続けている．

しかしこのアセンブリ言語，特にあまりレベルの機能の高くない命令に対応したアセンブリ言語と，汎用マイクロプロセッサのインサーキットエミュレータに比較して低い機能しかもたないオンラインデバッガによるプログラム開発の開発効率の向上には一定の限度があることも事実である．

DSP のプログラム開発ツールとして，シミュレータを提供するメーカもある．これはホストコンピュータシステム上に DSP の命令を実行する仮想マシンを実現するもので，4/8 ビットマイクロプロセッサの初期にプログラム開発に用いられた方法である．

この方法はインサーキットエミュレータの機能が充実してきた現在ではあまり用いられることが少なくなってきたが，DSP でも同様である．プログラムの実行結果が知りたければ，仮想マシンの上でなくてもオンラインデバッガを用いて実行結果を得ることができる．しかもこちらのほうが手間がかからないし，何よりもプログラムデバッグにこの実行結果は有用な情報をもたらすからである．シミュレータ上でハードウェアの状況を再現することは意外に手間がかかり，しかもそれが確実に再現されているという確認をすることもかなりの手間がかかるものである．

特にシミュレータはリアルタイム処理のプログラムデバッグにはあまり役に立たない．それは微妙なタイミングで入力される信号に対する DSP の動作を再現するのが困難である，もしくはこうした大量の入力信号に対するシミュレーション結果を得るのが困難であるためである．

もう少し詳しく述べると次のことである．一般にシミュレーションは設計者が自分の意図としたことを確認するには非常に有効である．しかし設計者が意図しなかったこと，もしくは設計者が抜け落としてしまったことに対する確認にはほとんど役に立たない．ところが非同期の割り込み処理などのリアルタイム処理に必ず含まれる処理はこうしたケースが含まれることが常である．これは致し方のないことで，非同期割り込みのタイミングを十分に考慮するといっても，この場合は一般的に膨大な場合の数があるのが普通であり，これらの場合をすべて抜け落ちなく考えることは事実上不可能である．このためにシミュレーションで確認できることには限度があるのは，VLSI の開発でも DSP プログラム同様である．このためにできるだけ実際の状況に近い状態でプログラムデバッグを行うのが，もっとも有効な方法となる．このためにシミュレータはあまり使われないのが現状である．もちろんシミュレータにはハードウェアがなくてもプログラム開発をある程度進められるなどのメリットがあるが，これにより決定的に DSP プログラム開発の効率化が達成されるわけでもないのであまり重要視されていない．

2.3 プログラム開発法

プログラム開発には明確な方法論が必要で，これによるプログラム開発の効率化も広く認められている．これがソフトウェア工学の中心課題である．もちろん現在のソフトウェア開発において，そのプログラム開発効率の多くの部分がプログラマの力量に負うことは事実であるが，プログラム開発の方法論もこの力量に準じた位置を占めつつある．すなわち適切なプログラム開発方法は大いにプログラム開発効率の向上に寄与する．

DSP プログラム開発にはプロトタイピングが有効である[2]．プロトタイピングは特に厳密な概念ではない．最終的に使用するプログラム（ここでは製品プログラムと呼ぶ）を最初から開発せずに，その前にプロトタイププログラムと呼ぶ予備的なプログラムを開発し，そのプロトタイププログラムの結果をみて，製品プログラムを開発するプログラム開発法である．

このプロトタイピング自体はごく自然なシステム開発法であり，特に馴染みのない概念でもない．DSP のプログラム開発ではプロトタイピングは無意識に発生していたと言える．それはディジタル信号処理の研究がメインフレーム，ミニコンピュータ上で FORTRAN，C などの言語を使い，実際のリアルタイム処理を実現するシステムは布線論理で作られていることに関係している．

こうした布線論理のシステムに対して，FORTRAN，C で書かれたプログラムをシミュレーションプログラムと呼んでいる．このシミュレーションプログラムという意味は，布線論理のリアルタイム処理をするシステムとメインフレーム，ミニコンピュータ，ワークステーション（WS）の間にあまり直接的な関係がなく，ただ単に機能をシミュレートするからである．ただシリコンコンパイラの発展により，FORTRAN，C で書かれたプログラムが布線論理のシステムの構成に無関係であるとはいえなくなってきているから，このことは歴史的な意味に限定すべきである．

リアルタイム処理をするシステムが DSP で構成され，そのために DSP プログラムが必要になってくると，FORTRAN，C で書かれたプログラムと DSP プログラムの関係は無関係どころではなくなる．したがってシミュレーションという言葉も適切ではなくなってきた．この関係がプロトタイププログラムと製品プログラムの関係になってきたのである．

プロトタイピングには大きく分けて，使い捨てプロトタイピングと骨格プロトタイピングがある[3]．前者はプロトタイププログラムと最終的な製品プログラムとは別個に開発する．つまりプロトタイププログラムで確かめることを確かめたら，また新たに製品プログラムを開発し，これらの両者には直接的なつながりがないケースである．一方後者はプロトタイププログラムに手を加えて最終的に製品プログラムにしてしまうケースである．

使用する言語からいえば，前者はプロトタイププログラムと製品プログラムで同一の言語を使用しても，異なる言語を使用してもかまわない．なぜならばプロトタイププログラムと製品プログラムはあくまでも別のプログラムであるからである．ところが後者はプロトタイププログラムと製品プログラムは同一の言語を用いる必要がある．プロトタイププログラムの改良版が製品プログラムなのであるから，この二つのプログラムは同一の言語を用いることがほとんど必須となる．もちろんこれらのプログラムで異なる言語を使うことが考えられないことはないが，自然でないことは自明であろう．

現在の時点で，DSP の主たるプログラム言語はアセンブリ言語である．アルゴリズムの開発，確認，プログラム中のパラメータの決定などのために，WS などの上で使われる言語は FORTRAN，C などの高級言語であるから，DSP のプロトタイピングは必然的に使い捨てプロトタイピングとなる．すなわち WS などの上で DSP で実現するプログラムのアルゴリズムの確認，プログラム構造の決定，パフォーマンスの推定，最大ダイナミックステップ数の推定，プログラム中に含まれるパラメータ値の決定などをプログラム開発環境の優れた WS などの上で高級言語で行い，これをもとに新たに DSP 用プログラム（ターゲットプログラム）を開発するのである．

プログラムを 2 回開発するという心理的な抵抗感があるのが，この方法の欠点であるが，実際に DSP プログラム開発を経験してみるとこの心理的な抵抗感は消えるのが普通である．

プロトタイピングが DSP プログラムの開発効率を大きく向上させる理由は明確である．それは，先に述べたような DSP プログラム開発に必要な事項の決定は必ずしも DSP を用いて行わなくてもすむものであり，これを DSP を用いたあまり優れたとはいえないプログラム開発環境を使わずにすますことによる．つまりリアルタイム処理の実現のための入出力信号のタイミングの確認などの実際の DSP を用いなければ確認できないこと以外は，プログラム開発環境の優れた WS などで走行するプロトタイププログラムで必要事項の確認，決定をするからである．つまりできるかぎりのプログラム開発作業を優れたプログラム開発環境のもとでプログラム開発を行うからである．

DSP のシミュレータも優れたプログラム開発環境のホストコンピュータ上で走行するから，これをプログラム開発効率の向上に役立てる方法に結びつけることが可能に思われる．ところが先に述べたプロトタイ

ププログラムで確認できる事項は高級言語のプロトタイププログラムで行ったほうが効率がよく，リアルタイム処理の鍵になるタイミングが関係するプログラムデバッグはシミュレータでは不可能ということで，あまり役に立たないのが実状である．

2.4 DSP プログラムの特殊性

DSP プログラムはリアルタイム処理を要求されることが多い．ディジタル信号処理システムの入力信号は，ほとんどの場合正確な周期性をもつ．したがってリアルタイム処理実現の条件はその周期内にその入力されたデータに対する所定の処理(プロセス)が終了するか否かで判定できることが多い．このためにプロセスの処理時間が過去，現在のデータの数値に依存しない場合は，ダイナミックステップが一定になるので，このダイナミックステップ数が与えられた周期内に納まるか否かで，リアルタイム処理の実現が判定できる．

しかしこうした場合に該当するのは，FFT，DCT，ディジタルフィルタなどの比較的単純な処理の場合であり，もっともディジタル信号処理の特徴が発揮される適応信号処理では，このダイナミックステップでは一定の条件は満足されない．

この場合プロセスの処理時間は過去のデータの値と現在のデータの値により変化するので，なんらかの方法でこの処理時間の最大値を知り，これが入力信号の周期を超えることがないようにしなければならない．

プログラムも，繰り返しループからの脱出について最大反復回数を決めておくこと，最長の処理時間となる制御フローについて把握し，場合により強制打ち切り（ならびにこの後始末）を行うなどの処置が必要となる．もちろんアルゴリズムによってはこれらの処置と馴染まないものもありうるので，ケースバイケースの工夫が必要となる．

DSP のリアルタイム処理の要求はある時間内に処理が正常に終了しなければ，処理自体が無意味になることが多く，大変に厳しいリアルタイム処理の要求である．その反面，入力信号の周期が確率的な変動をもつものではなく，正確な周期をもつために，リアルタイム処理が実現できるか否かの判定は比較的容易に行える．

ただしプログラムをどのように変更しても要求される周期内に処理が終了できない場合には，DSP を追加して周期内に処理を終了させる以外に方法はなくなる．この場合通常のマルチプロセッサのプログラミングとなり，共有資源の管理，同期の問題は避けて通れない．

一般に DSP は特に積極的なマルチプロセッサシステム対応の設計をしているわけではない．I/O ポートに外部からリセット・セットできるフラグをもつ程度なので，マルチプロセッサ構成にはそれなりの周辺回路が必要となり，これらを考慮したプログラムが必要となる．この共有資源の管理，同期に適した命令を DSP はもっているわけではないので，DSP をマルチプロセッサシステムに使うとこの部分のオーバヘッドが問題になることもある．共有資源の管理，同期に適した命令とは OS に適した命令であり，これが DSP の命令セットに含まれていないことはすでに述べたとおりである．

DSP プログラムの特殊性として，データの演算精度に対する確保の問題がある．DSP プログラムは通常の数値解析と異なり，データの精度だけを優先させたプログラムを書くわけにはいかない．その理由は精度を優先のプログラム，実際には倍精度演算を多用したプログラムは演算処理時間が非常に増加し，リアルタイム処理の要求と衝突するからである．この妥協点をどこにするかについては問題に依存し一律の判定基準はない．結局リアルタイム処理の実現と精度の不足によるパフォーマンスの劣化の許容値のバランスをみて，ケースバイケースに判断する問題である．

3. 高級言語による DSP プログラム開発

3.1 C言語による DSP プログラム開発

最近の RISC (Reduced Instruction Set Computer) の発展により，コンパイラの開発にセマンティックギャップが大きいことは致命的な欠点とはならないことが認められつつある．

DSP の命令セットは RISC よりもさらに低いレベルなので，DSP 用C（クロス）コンパイラのセマンティックギャップは RISC 用Cコンパイラよりもさらに大きい．(以後に現れる DSP 用コンパイラはすべてクロスコンパイラである，これは自明に近いのでクロスを省略する）コンパイラ開発の困難さはこのセマンティックギャップに比例するから，十分な性能をもつ DSP 用Cコンパイラの開発はあまり容易なものではない．

しかし高級言語のプログラム開発に占める重要さは揺るぎないものであるから，DSP で高級言語を使えるようにするための DSP 用コンパイラの開発が進められてきている．

この問題にはどのような言語を選択するかという問題もあるが，現在はC言語が優勢になっている．事実C言語を実際にリリースもしくはリリースの予告をしているDSPメーカは現在数社あり，今後も増加すると予想されている．

DSP用高級言語の開発は日本で最初に始められた．1982年に富士通の池坂などによるPascalのサブセットの言語のDSP用コンパイラが開発された[3]．1985年に著者などによりC言語のDSP用コンパイラが開発され[4]，以後TI，ATTと続き，今日のDSP用高級言語はC言語が主力となってきている．

これらのコンパイラの言語仕様はフルセットのC言語に近いものであり，特にサブセットであることを明示していない．言語仕様をフルセットに近づけるのは明確な理由がある．それは近い将来DSP用コンパイラが十分な性能をもつときに，ホストコンピュータ上で作成したC言語によるプロトタイププログラムをわずかな修正でDSP上でも走らせられるようにすることを考慮しているからである．

もしこのことが可能になれば，プロトタイピングは使い捨てプロトタイピングから骨格プロトタイピングに変わることになり，これによりDSPプログラム開発の効率が大きく向上することは明らかである．

ここで十分な性能といっているのは，DSP用コンパイラで生成するオブジェクトプログラムのダイナミックステップ数が可能なかぎり小さいという意味であり，コンパイラ自体のホストコンピュータ上での走行時間などは意味していない．また将来ともDSP用コンパイラに要求されるのはこの意味での性能であり，DSPコンパイラの走行時間，その大きさなどはほとんど問題にならないであろう．

現在のDSPコンパイラの性能はまだ十分ではない．ライブラリプログラムを利用して，あまりリアルタイム処理の要求の厳しくない分野で利用が可能になったという程度である．ハードウェアの能力自体は汎用マイクロプロセッサの1桁上の能力をもつが，DSP用コンパイラはこの演算能力を完全に引き出していない．したがってベクトル演算処理に関して現在の汎用マイクロプロセッサの数倍のパフォーマンスが実現できる程度である．

しかしコンパイラの性能はユーザ数の増加とともにその性能向上の努力が続けられ，実際に性能がかなり向上する．このために今後あまり遠くない将来にDSPの主言語はC言語となる可能性は非常に高い．このことはスーパコンピュータでも，マイクロプロセッサでも起きてきたことで，DSPがこの例外であるとする技術的な理由はない．今後DSPはNeXTにみられるように，マルチメディア対応のパーソナルコンピュータ，WSに組み込まれてベクトル処理の高速化に使われるであろう．この動きはDSPのユーザを飛躍的に増加させるので，コンパイラの性能向上を促進するであろう．

3.2 オブジェクト指向言語を用いたDSPプログラム開発

DSPのプログラム開発にプロトタイピングが有効であり，プロトタイププログラムの開発がDSPプログラムの開発にかなりの部分をしめる．したがってこのプロトタイププログラムの開発を効率よく行えるプログラム開発環境，言語はDSPプログラム開発の効率を向上させる．

プロトタイププログラムの開発に適したプログラム開発環境（言語を含む）としてオブジェクト指向の*Smalltalk-80*がある．この*Smalltalk-80*を用いてプロトタイププログラムを迅速に開発することは，明らかにDSPプログラムのプログラム開発の効率向上に有効である．

*Smalltalk-80*が非常に優れたプログラム開発環境であり，プロトタイププログラムの開発に非常に有効であることに加えて，著者は*Smalltalk-80*が信号処理に適しているのは次の3点によると考えている．

- 信号処理アルゴリズムは多くの場合シグナルフローグラフもしくはそれを拡張した記法で正確に記述される．この記法はあるモノに信号を送り，そのモノから信号を受け取ることを意味し，これがオブジェクト指向の基本概念とよく馴染む．
- 信号処理においては信号のグラフィック表示が非常に有効で多用される．DSPプログラム開発でも同様に有効で多用される．このグラフィック表示を操作するのにオブジェクト指向は非常に適している．
- プロトタイピングを前提にプログラム開発環境が整えられている．

プロトタイププログラムが「使い捨て」の場合は特にDSP用言語との関係は考慮する必要がない．完成した*Smalltalk-80*によるプロトタイププログラムをもとにアセンブリ言語，もしくはC言語でDSP用プログラムを新たに開発すればよい．

骨格形プロトタイピングを採用する場合，現在の段階ではDSP用言語とのつながりにかなりの問題があ

る. *Smalltalk-80* は言語とプログラム開発環境が融合した形になっており，クロスの形態は現在のところ考慮されていない.

現在のところ *Smalltalk-80* とC言語を結ぶものとして，Objective-C と C++がある. *Smalltalk-80* のプロトタイププログラムを同じオブジェクト指向言語の Objective-C か C++に書き換えることは，プログラムのあまり大きな変更にならない. 得られた Objective-C か C++ のプログラムをCに変更することは容易である. なぜならば多くの Objective-C と C++ コンパイラはCのプリプロセッサの形態のものが多く，またネイティブな C++ コンパイラでも C++ 自体Cの自然な形のスーパセットになっているからである. 前者の場合は自動的にCプログラムが得られ，後者の場合はトランスレータを開発するのにもあまり大きな工数はいらないからである. しかしこうした *Smalltalk-80* を用いる骨格形プロトタイピングが実用的な意味をもつためには DSP 用Cコンパイラの性能向上が不可欠である.

Smalltalk-80 の信号処理への応用は文献 5) が非常に参考になる. この文献では DSP プログラムのプロトタイピングという観点は打ち出されていないが，信号処理におけるオブジェクト指向の有効性を丁寧に例示している.

4. プログラム開発ツール

4.1 ハードウェア

DSP プログラム開発に必要なハードウェアはホストコンピュータとオンラインデバッガと DSP が実現するシステムのパフォーマンスを測定する機器，具体的にはシグナルジェネレータ，ディジタルシグナルジェネレータ，オシロスコープ，ロジックアナライザを基本とする機器である. ただしこのパフォーマンスを測定する機器は DSP で実現しようとするものにより大きく変わる. MODEM, CODEC, エコーキャンセラなどではそのパフォーマンスを測定する機器が異なるのは当然であろう. オンラインデバッガは先にも述べたように汎用マイクロプロセッサのインサーキットエミュレータに相当するもので，DSP 上のプログラムをホストコンピュータの制御下におくためのハードウェアである.

4.2 ソフトウェア

DSP のソフトウェア開発に必要なソフトウェアは，プロトタイププログラムの作成までは特に特別のものが必要なわけではない. プログラム／データ・ファイル作成，維持，管理に必要な OS, ウィンドウシステム，エディタ，プロトタイププログラムのためのネイティブコンパイラ，さまざまなユーティリティはプログラム開発に不可欠である. これらのホストコンピュータの標準的なソフトウェアに加えて，DSP プログラム生成のための（クロス）アセンブラ，（クロス）コンパイラ，DSP プログラムライブラリが必要となる. さらにプログラムデバッグのためにオンラインデバッガ制御プログラムが必要である. オンラインデバッガ制御プログラムはオンラインデバッガと組で DSP 上で走行している DSP プログラムを制御する.

以上のソフトウェアが基本的なものであるが，これに加えてディジタルフィルタ設計プログラム，種々のグラフィック表示ソフト (GKS など)，またパフォーマンス測定機器を制御するプログラムなどもプログラム開発の効率向上に有効である.

5. あとがき

DSP プログラムのプログラム開発が困難な理由は冒頭に述べた. 現在この状況はかなり改善の方向に向かっており，その象徴が DSP 用Cコンパイラの開発である. DSP は今までの少数のエキスパートにより使われるパフォーマンスの高いプロセッサから，不特定多数のユーザにより使われる普通のプロセッサに変わりつつある.

この変化を促しているのは，マルチメディア対応のパソコン，WS からのニーズである. このニーズは不特定多数のユーザの利用を前提にするために，必然的に DSP プログラム開発を容易に行えることが要求される. これを具体化しているのが，方法論としてのプロトタイピングであり，ツールとしての DSP 用Cコンパイラである.

DSP 用Cコンパイラの性能が向上すれば，現在のパソコン，WS のコプロセッサのように，あまり DSP が組み込まれていることを意識せずに DSP が使われるであろう. C言語のソースプログラムのベクトル処理の部分を DSP が行うという形は，近い将来に常識的なことになると著者は予想している.

なお本論文では画像処理を主な目的とするマルチプロセッサ構成用の DSP については一切省略した. これはこの DSP が現在まだ明確な形で市場が形成されていないこと，技術的にもまだ未知数の部分が多いためである.

参 考 文 献

1) 小野：ディジタルシグナルプロセッサの最近の動向，電気学会論文誌C，Vol. 108-C, No. 10, pp. 761-765 (Oct. 1988).
2) 小野，石井：DSPのソフトウェア，p. 166，コロナ社，東京 (Aug. 1989).
3) 池坂，蓮，相馬，峰島：ディジタル・シグナルプロセッサ用試作コンパイラの最適化手法，情報処理学会第24回（昭和57年前期）全国大会資料4L-4, p. 255 (1982).
4) Ono, S. and Kanayama, Y.: A Program Development System for High Performance DSP DSSP1, Proc. ISCAS 85, pp. 1141-1144 (1985).
5) 小林：オブジェクト指向と Smalltalk, p. 191, CQ 出版社，東京 (1989).
6) Lee, E. A.: Programmable DSP Architecture: Part 1, IEEE ASSP MAGAZINE, Vol. 5, No. 4, pp. 4-19 (Oct. 1988).
7) Lee, E. A.: Programmable DSP Architecture: Part 11, IEEE ASSP MAGAZINE, Vol. 6, No. 1, pp. 4-14 (Jan. 1989).
8) 有沢：ソフトウェアプロトタイピング，近代科学社，東京 (1986).

(平成元年7月3日受付)