第17章

高度化・複雑化するプログラムに対応する

新時代のDSP開発環境

本章では，これからの新しいDSP開発環境について紹介します．本書で取り上げてきたDSPプログラムはいずれも簡単なものばかりですが，実際のDSPを内蔵した製品のプログラムはどんどん高度化・複雑化しています．このような状況で，DSPプログラムの開発手法にも次のような新しい流れが出てきています．

① アルゴリズムのIP化・IP流通

② OSの導入

③ 高レベルの設計ツール導入

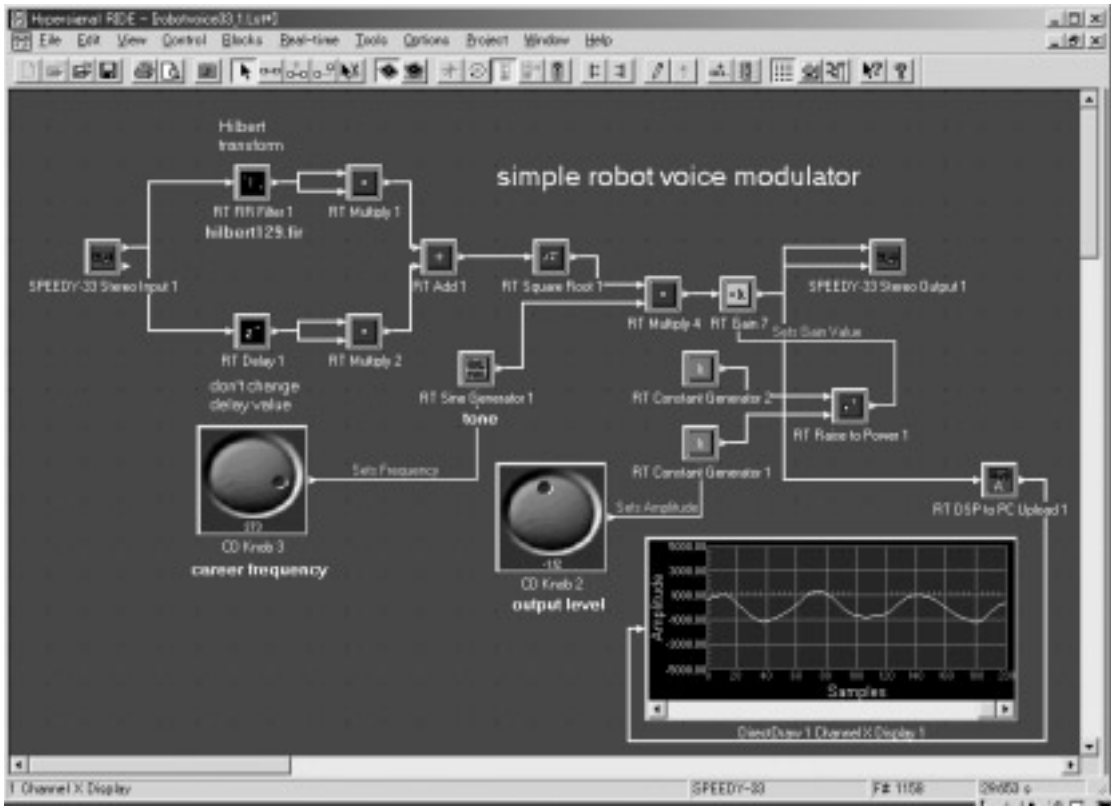
複雑なDSPシステムでは，すべてを自前で開発することは困難で，標準規格に準拠したデータ圧縮伸張アルゴリズムや変調復調アルゴリズムなどは，外部からIPとして購入して使うのが当たり前になってきています．場合によっては，開発工数の低減などの目的で，DSP向けのOSを使うことも珍しくはありません．

例えば，DSPのみを使って設計した携帯オーディオ機器などで，ハード・ディスクやフラッシュ・メモリ上のデータ・ファイルを扱う場合には，ファイル・システムを備えたOSを使えばそれだけ設計が楽になり，自前で複雑なファイル・システムを開発する必要がなくなります．

17-1 ツール RIDE

これから紹介する設計環境は，前述した③に相当する高レベルの設計ツールです．RIDE（Realtime Integrated Design Environmnet）という製品名で，米国Hyperception社製のツールです（Hyperception社はLabVIEWの開発・販売元であるNational Instruments社のグループ会社です．

RIDEがどのようなツールかというと，**図17-1**のようなブロック線図方式のDSPプログラムの開発環境です．パソコン用のシミュレーション・ツールにはLabVIEW，SystemView，VisSimなどのブロック線図プログラミングを用いた製品がありますが，RIDEはシミュレーション・ソフトではなく，DSPに特化したコード生成ツールであるところが異なります．



〈図17-1〉RIDE（ブロック線図方式のDSP開発環境）

ブロック線図方式のプログラミング・ツールをDSP開発に用いるメリットは何でしょうか？ 一つは，テキストでプログラムを入力する必要がないことです．多くのディジタル信号処理アプリケーションは比較的単純なパイプライン処理なので，ファンクション・ブロック（**図17-1**中の一つ一つの四角い箱）を並べて，シグナル・フローを表す矢印で結んでいくだけでプログラムを作成できます．信号処理アルゴリズムの説明に用いられるシグナル・フロー・グラフやブロック線図が，そのままプログラムになるのですから簡単です[注17-1]．

しかし，ブロック線図プログラミングの最大のメリットは別のところにあります．それは，プログラムのメンテナンスの容易さ，コードの効率的な再利用，そしてアルゴリズムのIP化対応にあります．従来のプログラミング言語でテキスト記述された信号処理プログラムを保守することはたいへんです．ソース・コードがあれば，すべてがわかるのではないかと思われるかもしれませんが，ディジタル信号処理のプログラムはソースを読んでも，どのようなアルゴリズムで処理をしているのか，理解するのは困難です．それと比較すると，四則演算，各種算術関数，ディジタル・フィル

注17-1：これはシグナル・フローが処理プログラムそのものを表すディジタル信号処理ならではの話で，一般的な計測制御などのアプリケーションでは，パソコン用のブロック線図方式のプログラミング・ツールが必ずしも有効でない場合もある．

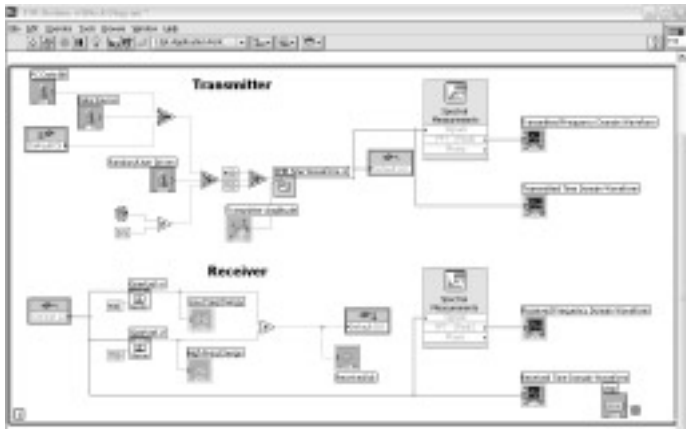
Column 17-1

RIDEとLabVIEW DSPに関する追記

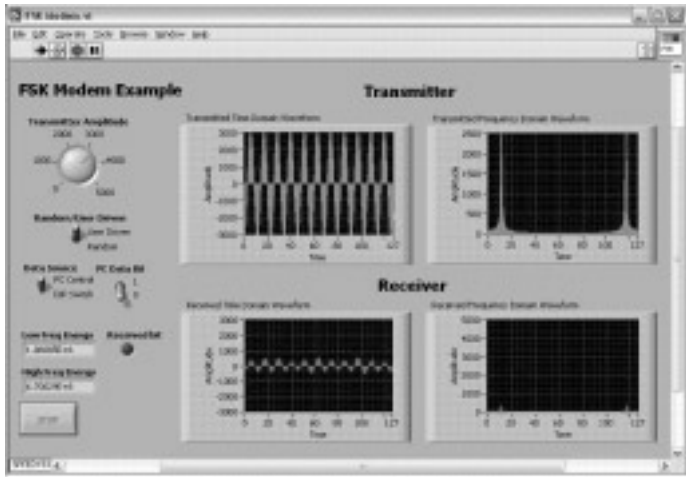
本章の原稿執筆後にHyperception社のRIDEをはじめとするDSP開発ツール群は販売停止となりました．今後，Hyperception社を買収したNational Instruments社（NI）のLabVIEWにRIDEの先進的なDSP開発機能が順次取り込まれていく予定だそうです．

すでにNIは，RIDEの機能の一部を移植したLabVIEW DSPモジュールをリリースしています（**図17-A**，**図17-B**）．LabVIEWに合わせてGUIが変わっていますが，RIDEと同様のブロック線図方式プログラミングでDSPのコード開発ができます．

なお，本稿の執筆時点で，LabVIEW DSPがサポートしているDSPボードは，SPEEDY-33（NI社，TMS320VC33搭載），C6711 DSK（TI），C6713 DSK（TI）です．SPEEDY-33の開発元はHyperception社ですが，販売元がNI社に変わっています．



〈図17-A〉
LabVIEW DSPモジュールのプログラミング画面（モデム変調・復調）



〈図17-B〉
LabVIEW DSPモジュールのグラフ表示画面（モデム変調・復調信号の波形とスペクトル）

タ，FFT，相関などの既存のライブラリ中の標準的なファンクション・ブロックを用いて作られたブロック線図方式プログラムのほうが，かえって理解しやすく再利用も容易なことが多いのです．

RIDEのコード生成のしくみは，次のようになっています．

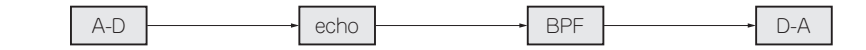
例えば，**図17-2**に示すようなA-D変換した音声データにエコーをかけた後，ディジタル・フィルタに通してD-A出力するという処理を考えてみましょう．RIDEでは，画面上に**図17-2**のブロック線図を描いたものが，そのままプログラムとなります．

RIDEでは，**図17-2**の一つ一つの箱がファンクション・ブロックを表します．ブロックの実体はC言語で記述された関数で，ライブラリの中にはブロックに一対一で対応したコンパイル済みのC言語の関数が用意されています．(**図17-2**の例では，関数ad，echo，bpf，daをRIDEのライブラリとしてもっています)．

RIDEの環境でユーザがブロック線図で描いたプログラムを実行するときの処理の流れは，次のとおりです．

まず，ブロック線図の結線にしたがって，各ブロックに対応する関数を逐次呼び出して処理を行うDSPプログラムを生成します．生成されたプログラムは，**リスト17-1**に示すC言語のmain関数に相当する処理を行います．RIDEが生成したプログラムから呼び出している関数(ad，echo，bpf，da)は，RIDEの標準ライブラリに含まれていますから，コンパイルは不要です．

次に，main関数に相当するプログラムとライブラリの関数をRIDE内蔵のリンカでリンクして実行プログラム(*.out)を生成します．最後に，実行プログラムをDSPボードにロードし起動します．このように，RIDEでのコード生成手順は，C言語を用いた一般的なプログラム作成の流れと変わりありません．連続時間系と離散時間系の混在シミュレーションに対応したパソコン用のブロック線図方式のシミュレーション・ツールと異なり，離散時間処理専用なので演算効率の良いコードを生成できることがRIDEの大きな特徴です．



〈図17-2〉簡単なパイプライン処理の一例

〈リスト17-1〉
RIDEが生成するDSPプログラムに相当するC言語のコードのスケルトン

RIDEはmain関数に相当するコードを生成し，ライブラリ中の関数ad，echo，bpf，daとリンクして実行プログラムを出力する

```
void ad(float x[]);
void echo(float x[]);
void bpf(float x[]);
void da(float x[]);

void main(void) {
  float x[FRAME_SIZE];

  while (TRUE) {
    ad(x);
    echo(x);
    bpf(x);
    da(x);
  }
}
```