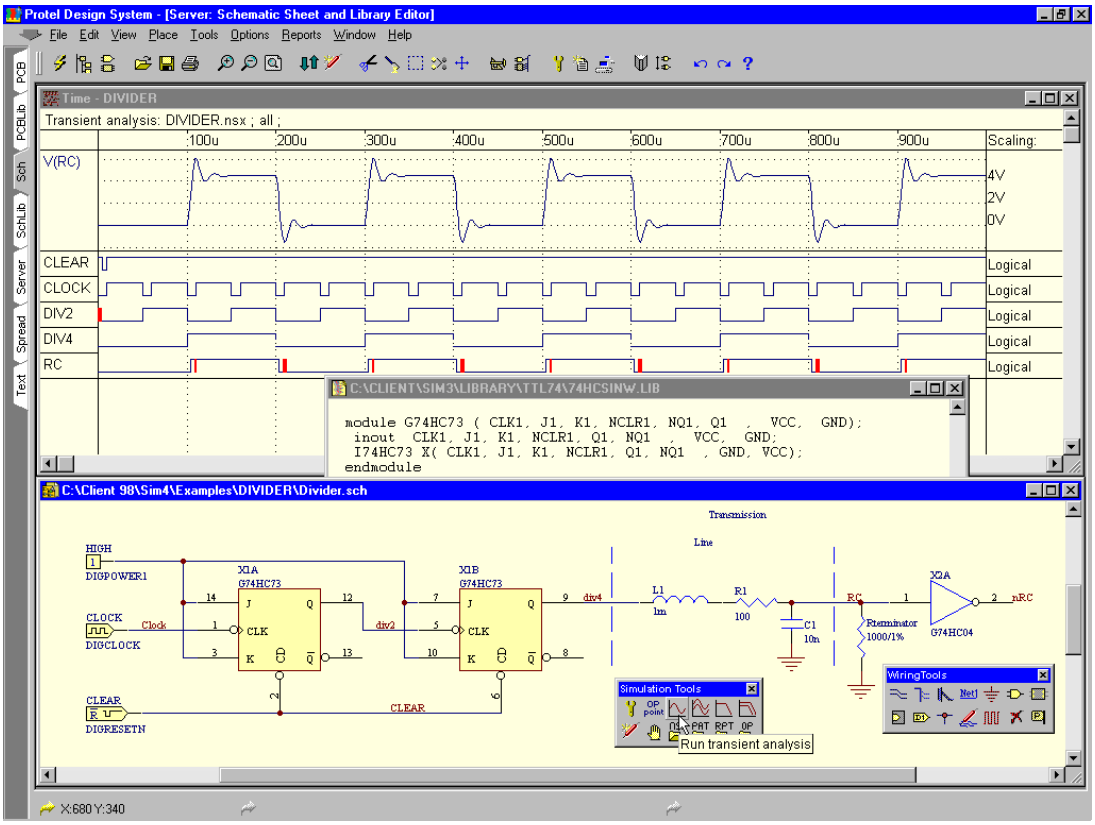
アドバンストSimイントロダクション



Protel デザイナーズハンドブックのこのセクションでは、アドバンストSimによるアナログ／デジタル混在シミュレーションの方法を紹介します。アドバンストSimは、真の混在シミュレーターでアドバンストスケマティックと共に使用します。アナログ波形とデジタル信号のシミュレーションが行え、併せて表示する事ができます。

まず、このセクションをお読み頂く事でシミュレーターを使用する上で必要な情報を見つけ、作画方法や操作方法を学び、自分で設計された回路のシミュレーションを行うための準備として下さい。

アドバンストスケマティックを使っての作画

アドバンストスケマティックは、設計製造に必要な多くの便利な特長を備えた電子回路デザイン環境です。アドバンストスケマティックだけで、回路作成が行なえます。アドバンストSimのようなシミュレーションツールや、アドバンストPCBのような基板レイアウトツールと組み合わせれば、アドバンストスケマティックは、完全に自動化され統合化されたエンドツーエンドの設計システムになります。

アドバンストSimには、シミュレーションモデルにリンクしている、「すぐシミュレーションが可能」なスケマティックシンボルのセットが含まれています。アドバンストSimをEDA/クライアントにインストールすれば、アドバンストスケマティックの内部からシミュレーションをセットアップし、動作させる事ができます。回路から波形に直接クロスプローブすることができ、また、いつでもシミュレーションを再構成、再動作することができます。

アドバンストSim - アドバンストシミュレーションテクノロジー

アドバンストSimは主に、ネットリスターとシミュレーションエンジンそして波形ビューワーの3つのコンポーネントから構成されています。アドバンストSimは、これまであったアナログ／デジタルシミュレーターの中でも、独自のオリジンをもっています。アドバンストSimは、シミュレーションテクノロジー開発のワールドリーダーである、Dolphin

Integration（ドルフィンインテグレーション）社所有のシミュレーションテクノロジーを元に開発されました。ドルフィンの経験と専門的技術は、そのシミュレーションテクノロジーへのアプローチにより、業界の標準となっているSPICEとVerilog-HDLとの互換性を備えた、高速でメモリー効率の良いパワフルなシミュレーションテクノロジーを獲得しました。

Spiceコンパチビリティ

アドバンストSimは、SPICEコンパチブルです。.SUBCKTを使用した階層定義を含む、すべてのクラシカルコンストラクトがサポートされています。（新しいBSIMやEPFLモデルももちろん）1、2、3レベルのようなクラシカルトランジスタモデルもサポートされています。ダイオード、JFETトランジスタ及びバイポーラトランジスタは、SPICEモデルを使用します。

Verilog-HDLによるデジタルシミュレーション

アドバンストSimは、IEEE 1364 Verilog-HDLシミュレーションを提供します。アドバンストSimは、IEEEスタンダードデジタル言語をサポートしている数少ない混在信号（混在）シミュレーターの中の1つで、外部のVerilog-HDLライブラリを使用することが可能です。アドバンストSimは、外部のVerilogシミュレーターを当てにしている訳ではなく、Verilog-HDLの記述を解析するプログラムとストラクチャー（ゲート）レベルのシミュレーションをサポートしているシミュレーションエンジンがすべて含まれています。

デバイスマニュファクチャーモデルのサポート

アドバンストSimは、アナログシミュレーターとしての高い互換性により、モトローラやテキサスインストゥルメンツやその他のモデルプロバイダーから供給されているSPICEモデルをサポートしています。アドバンストSimは、その高い互換性により、これらのモデルを直接読む事ができます。

広範囲にわたるモデルライブラリ

アドバンストSimには、多くのライブラリセットが含まれており、スケマティックシンボルは6400以上になります。これらのシンボルは、それぞれ適切なシミュレーションモデルファイルにリンクされ、回路図上に配置するだけで「すぐシミュレーションが可能」です。これらのライブラリはプロテルライブラリディベロップメントセンターで常にメンテナンスが行われています。ここで、すべてのプロテル製品のライブラリが開発されています。プロテルユーザーはプロテルのWebサイト（ http://www.protel.com ）から、これらのライブラリを自由にダウンロードして使用する事ができます。

シミュレーションの制限

アドバンストSimは、制限のないサーキットレベルアナログシミュレーションと、制限のないゲートレベルデジタルシミュレーションを提供します。回路サイズは理論的に、システム上のメモリの総量によって制限されるだけです。アドバンストSim は、メモリが高価で容量が少ない頃からの密度の高いデータ構造を継承しています。その結果として大きな回路をロードすることができます。

多くの「混在信号」シミュレーターと言われるものは、デジタルゲートを使わないデータ構造のため極端にメモリを消費します。この種のものとは違いアドバンストSimでは、デジタル回路に大量のメモリを必要としない専用のデータ構造をもっています。実際には、シミュレートすることができる回路図のサイズの制限は、シミュレーションに関するユーザーの理解力と操作力の技術によります。

サポートされている解析

アドバンストSimは、AC小信号、過渡解析、ノイズ、DCトランスファー及びパワーアップ解析をサポートしています。更にこれらの基本的な解析には、モンテカルロ解析、コンポーネントの値を変化させるスイープ解析があります。また、シグナルトゥノイズとトータルハーモニックディストーションの計算ができる高速フーリエ変換も含まれています。

強力なモデル

アドバンストSimは、基本シミュレーション能力を拡張したいくつかのモデルのレベルをサポートしています。

コントロールソース

これらは、簡単なリニア効果をシミュレートする方法を提供します。E、F、G、Hデバイスは、単純なリニアに属する「定義済み」方程式を提供します。

方程式 - 条件付きソースの定義

電圧・電流ソースはユーザー側の設定で、非直線電圧・電流が使用できます。アドバンストSimでは、この中に条件式を使うことができます。

ルックアップテーブル

解析を必要としないモデルは、ルックアップテーブルを使用します。ルックアップテーブルは、直線補正、キュービックスプライン補正、及びナチュラルスプラインをサポートしています。

ラプラス - ブロック定義

ラプラス変換は、いくつかのシミュレーションタイプで使用することができます。変換を定義するには、2つの方法があります。2つの多項式に係数値を与える方法と、電極と最下点のセットを与える方法とがあります。

アドバンストSimを始めるには

アドバンストSimへようこそ！アドバンストSimは簡単に使えるアナログデジタル混在型のシミュレーターです。アドバンストSimは、EDA/クライアントの環境下にパワフルなシミュレーション技術をもたらします。アドバンストSimはProtelの回路図エディタであるアドバンストスケマティックと統合されています。アドバンストスケマティックで回路を作画すれば、アドバンストスケマティック上からシミュレーションの設定を行い、起動することができます。

このクイックツアーは、アドバンストSimで設計をシミュレートする基礎を紹介します。この章を読み終えた後、アドバンストSimに含まれているサンプルファイルでタイプの異なるシミュレーションを実行し、設定を変えたり、波形をズームしたり、スケマティックと波形との間でクロスプローブを行なったりしてシミュレーションの世界を探ってみて下さい。その後、アドバンストSimであなたの行った設計を分析できるようになり、アドバンストSimのパワフルな特徴について、更に理解することができます。

アドバンストSimの概要

アドバンストSimを起動すると、ネットリストファイル（*circuit*.NSX）とパターンファイル（*circuit*.PAT）という少なくとも2つのファイルから、シミュレーションエンジンに必要とされる情報をロードします。ネットリストファイルは回路の部品とそれらの接続が記述されます。パターンファイルはシミュレーションに必要なコントロールディレクティブが記述されます。もし、回路図にコンポーネントとして入力ソース（アナログの電圧、電流ソース、デジタルパターン等）配置しない場合は、このパターンファイルに記述する事ができます。

また、アドバンストSimはコンポーネントのモデルの情報をライブラリファイルから読み込みます。ライブラリファイルはサブサーキットの定義、モデルの定義、マクロの定義、ユーザーが定義したプリミティブを入れることができます。そのような要素がネットリストファイルやパターンファイルに見つからないときは、アドバンストSimはライブラリファイルの中を捜そうとします。ライブラリの要素の定義はまた、他のライブラリの要素を参照する事もできます。

ネットリストファイルもパターンファイルも自動的に作られますが、それらのファイルについて今すぐ理解する必要はありません。まず、シミュレーションのセットアップから起動させるまでのプロセスについて記載しているこの章を読み、アドバンストSimの基礎をマスターすれば、後からもっと進んだ概念に取り組むことができるでしょう。

➔ アドバンストSimのシミュレーションの波形を見ている途中で、メニューアイテムの機能にわからない所が出てきたら、メニューアイテムをハイライトして F1ボタンを押して下さい。オンラインヘルプのファイルが立ち上がり、その特徴を説明します。

➔ アスキーファイル中にある .LTRACE や .EPSの様な用語を知りたい場合は、カーソルをその単語の内側移動し、キーボードでCTRL+F1を押して下さい。オンラインヘルプでステートメントやディレクティブについて説明が表示されます。

コンベンション

数値の表記

数値を固定小数点もしくは浮動小数点で表記する事ができます。 また、下記のスケール表記も可能です：

記号	大きさ
Tまたは t はテラ	1E+12
Gまたは g はギガ	1E+09
MEGはメガ	1E+06
Kまたは k はキロ	1E+03
Mまたは m はミリ	1E-03
Uまたは u はマイクロ	1E-06

N または n はナノ	1E-09
P または p はピコ	1E-12
F または f はフェムト	1E-15
dB はデシベル	20.log()

例：

```
1E-9 or 1n or 1N or 0.000000001
```

➔ 数値と記号の間には、スペースを入れないで下さい。50N は有効ですが、 50 N は無効です。

セパレータ

セパレータとしてスペースとタブが使用できます。但し、タブを使用して間隔を空けると、異なるテキストエディタでは文字の間隔が変わり、うまく受け渡しできない事があるのでお勧めできません。

行

ファイルは行単位で解析されます。入力ファイル（*circuit*.NSX、*circuit*.PAT、及びライブラリファイル）に使用される行はリターンで区切られている必要があります。空白行も認められます。通常、1行に1つの要素（*circuit*.NSX）、もしくはディレクティブ（*circuit*.PAT）が記入されています。

構文変換インジケーター

メインとなるネットリストファイル（*circuit*.NSX）あるいは、 .CKTライブラリファイル（サブサーキット定義を含むもの）でSPICE/Verilog混在記述を取り扱う場合は、構文変換インジケーターを使う必要があります。

これらのインジケーターは >>> SPICE 或いは >>> VERILOG と表示され、SPICE形式の記述、及びVerilog形式の記述である事を示します。一度、構文インジケータが現れると、その後の全てのテキストは、次の構文インジケーターが表示されるまで、その構文として解釈されます。以下にSPICE及びVerilogの両方を用いているサブサーキット記述が示されていますので参考にして下さい。構文変換メカニズムに関する詳細は、*階層記述*を参照して下さい。

例：

```
.SUBCKT MIXED A B C D CLK NRST
C1 A 0 10Pf
C2 B 0 15Pf
Q1 C B A Q2N2222
>>> VERILOG
// 以下の記述はVerilog-HDLです:
not n(nclk, CLK);
nand n2(y, A, B, C);
>>> SPICE
// 以下の記述は、またSPICE 構文です:
C3 CLK 0 20Pf
R1 A B 100K
.ENDS
```

行の継ぎ足し

アナログ要素の記述が長すぎて1行に納まらない場合、改行コード（リターンキー）を挿入後、次の行の先頭を “+” にする事で、その記述を複数の行に分割する事ができます。

例：

```
RLOAD OUT 0 10K
```

は、以下の表記と同じです：

```
RLOAD
+ OUT
+ 0 10K
```

Verilog-HDL記述の場合、この記号は必要なく改行だけで構いません。

例：

```
mymod m(a, b, c);
```

は以下の表記と同じです：

```
mymod m(a,
b, c);
```

コメント

いくつかのレベルのコメントが使用することができます。入力ファイルは初めにVerilog-HDLディレクティブ、マクロなどを取り扱う処理が行われます。Verilog-HDL形式のコメントは *circuit*.NSXや *circuit*.PAT等の入力ファイル内はもちろん、アナログ記述の中でも使用することができます。ここでは2つ基本的なコメント形式をサポートしています。

1行のコメントは、以下の様に // の記号を用います:

例：

```
// this is a valid comment
  // this is a valid comment as well
R1 OUT VSS 100K  // this is a valid comment as well
not n(out, in);  // this is a valid comment as well
```

複数行に渡る場合には、以下の様に /* */の記号を使用します：

例：

```
/* これは有効なコメントです */
not n(out, in);  /* これも同様に有効なコメントです */

/* これも同様に有効なコメントです
R1 OUT VSS 100K
...
not n(out, in);
*/

/ * これはエラーになります。（ブランクがある事に注意して下さい。） */
```

SPICE形式のコメントも使用できます。但し、Verilog-HDLが処理された後の次の段階で処理される事を覚えておいて下さい。* はSPICEコメント行であることを示す記号として用いられます。また * は、その行の最初に記述されている必要があります。

例：

```
* これは有効なコメントです
  * これも有効なコメントです
R1 OUT VSS 100K  * これはエラーになります
R1 OUT VSS 100K  // これは有効なコメントです
```

➔ 使用する言語に合ったコメント形式を使用して下さい。.Vファイルでは // と /* */ をまた、SPICE記述では * をコメントとして使用します。尚、 // 及び /* */ は .NSX、.PAT及び .Vファイル内のどこでも使用できます。また、*が行の最初の文字でない場合、コメントインジケータとして見なされないので、いわゆる « inline »コメントを挿入するには、// を使用して下さい。

ノード名

ノード名は文字列で作成され、最大（半角）64文字まで使用できます。

アナログノード名で使用できる文字は：

A-Z 0-9 $ _ . + - #

デジタルノード名で使用できる文字は：

A-Z 0-9 $ _ .

➔ 式（*アナログプリミティブ*の章の方程式定義ソース及び、オンラインヘルプの .TRACEディレクティブの項参照）の中に入るノード名には + 及び − の文字は使用できません。

デジタルノードではVerilog-HDL LRMに従い、名前の最初に $ 及び数字（0-9）を使用する事はできません。

アナログノードでは、最初の文字に数字を使用することや数字だけの名前を付けることが可能です。もしアドバンストSimをアナログシミュレーションにだけ使用する場合、ユーザーのニーズに応じた名前を付けて下さい。但し、混在モードで使用する場合、インターフェースノードで発生する問題を避ける為、ノード番号を使用せずにノード名を使用して下さい。

アナログノードの場合、SPICEのルールに従い、大文字・小文字は区別されず、内部的にすべて大文字とみなされます。デジタルノードはVerilog-HDLのルールに従い大文字・小文字も区別されるので注意して下さい。これはインターフェイスノード（アナログ部品、デジタル部品両方が接続している箇所）で問題が発生するので注意して下さい。

インターフェースノードには、大文字を使用する必要があります。一般的にVerilog-HDL記述では、小文字のアイデンティファイヤーが使用されています。

階層名は一番上のレベルからそのノードの最も高いレベルまでのブロックのインスタンス名を使用して作成されます。インスタンス名を作成するデフォルト文字は ‘.’ （ドット）です。また、パターンファイルで .HIERCHARディレクティブを使用して修正する事ができます。例えば： .HIERCHAR _ は ‘_’（下線）文字をインスタンス名を作成するときに使用するよう設定されます。例えば：

```
CPU_ALU_ADD32_FADD07_NAND4_XINTERN
```

階層名は全部で64文字という制限がある事に注意して下さい。回路の階層が深い場合、ノード名及びインスタンス名はできるだけ短く作成して下さい。

➔ *階層記述*の章も参照して下さい。

インスタンス名

インスタンス名は、最大（半角）64文字までの文字列で作成されます。ノード名に使用できる文字は次の通り：

```
A-Z  0-9  $  _  .  +  -  #
```

純粋なアナログ記述の場合は大文字と小文字は区別されません。MOS42はMos42と同じです。

純粋なデジタル記述の場合は大文字と小文字は区別され、CPU はCpU では同じではありません。

インスタンス名はユニークである必要が有ります。2つのデバイスに同じインスタンス名を使用する事はできません。

デジタル素子の場合、階層的なインスタンス名は一番上のレベルから、そのインスタンスの親レベルまでのブロックのインスタンス名を使用して作成されます。

例：

```
CPU.ALU.ADD32.FADD07.NAND4.M3
```

アナログ素子の場合、階層的なインスタンス名は一番上のレベルからそのインスタンスのレベルまでのブロックのインスタンス名を使用して作成されます。また、SPICE構文のプリフィックスにあった名称で始まります。

例：

```
MOS3.CPU.ALU.ADD32.FADD07.NAND4
* in this example MOS3 is the instance name of a
* MOS transistor located in the NAND4 instance of
* instance FADD07 of instance ADD32 of instance
* ALU in CPU block.
```

階層名を作成する文字は、‘.’（ドット）がデフォルトです。これは、パターンファイルに.HIERCHARディレクティブを使う事で修正できます。例えば、“.HIERCHAR_”とすると、‘_’（下線）を階層名を作成する文字に指定できます。

例：

```
CPU_ALU_ADD32_FADD07_NAND4
* this example assumes the presence of
* ".HIERCHAR_" in the pattern file.
```

➔ 階層名は全部で64文字という制限がある事に注意して下さい。回路の階層が深い場合、ノード名及びインスタンス名はできるだけ短く作成して下さい。

デバイス内部のノード

直列寄生抵抗を持つデバイスを使用すると、自動的に「内部の」ノードがいくつか作成されます。これらのノードは、そのデバイスのインスタンス名と端子に関連する名前を持つことになります。（下記の表参照）これらは、*circuit*.OPファイルをスクロールすれば見つける事ができます。また、ダイアログのノード名のリストに表示されます。これらは他のノードと同じ様にアクセスできます。内部ノードにもアクセスする事ができます。これらの内部ノードを見る必要はほとんどありませんが、例えばバイポーラトランジスタ内部ベースを見るのも面白いかもしれません。

バイポーラトランジスタQNAMEのベース抵抗:	B$QNAME
バイポーラトランジスタQNAMEのエミッタ抵抗:	E$QNAME
バイポーラトランジスタQNAMEのコレクタ抵抗:	C$QNAME
ダイオードDNAMEの直列抵抗:	A$DNAME
MOSトランジスタMNAMEのドレイン抵抗:	D$MNAME
MOSトランジスタMNAMEのソース抵抗:	S$MNAME
FETトランジスタJNAMEのドレイン抵抗:	D$JNAME
FETトランジスタJNAMEのソース抵抗:	S$JNAME

バス表記

信号線のセットについて次のバス表記が認められています： `BUSNAME[i:j]`

インデックス i と j は正の数で、コロンで区切られている必要が有ります。

バス内の個々の信号は、`BUSNAME[0]`、`BUSNAME[1]`等の名前を用いることによりアクセスすることができます。

この表記は：

```
A[i:j]
```

次の表記と同じです：

```
A[i], A[i+1], … , A[j-1] , A[j]
```

反対表記もでき `A[7:0]` は：

```
A[7], A[6], ... , A[0]
```

と同じです。

バス表記はネットリストをより読みやすくするといった用途にも使用されます。またデジタルデザインを行っている時に、バスの値をプロットするといった用途にも使用されます。オンラインヘルプの .LTRACEディレクティブに関する記述を参照して下さい。

例：

```
* in circuit.NSX:
>>> VERILOG
module ADD16( A, B, S, CIN, COUT)
  input [15:0] A, B;
  input CIN;
  output [15:0] S;
  output COUT;
  wire W0, W1;
  ADD1 A0(A[0], B[0], S[0], CIN, W0);
  ADD1 A1(A[1], B[1], S[1], W0, W1);
endmodule
```

例：

```
* in circuit.PAT
.LTRACE TRAN HEX Q[15:0]
.LTRACE TRAN BIN A[7:0]
```

グローバルノード

0（ゼロ）の名称の定義済みノードは、規則によってアナログの基準点（ゼロボルト）になります。このノードは全アナログ階層を通してグローバルとなり、これはどのSUBCKT定義にも一切「パス」を通す必要が無い事意味しています。

もしノード0以外のノードをグローバルに設定したい場合は、パターンファイルで .GLOBAL ディレクティブを使う必要が有ります。例えば、VDD及びVSSというノードを全階層でグローバルに設定したい場合、以下のディレクティブをパターンファイルに追加して下さい：

```
.GLOBAL VDD VSS
```

またこれらのノードを電源に設定したい場合は、以下の行をパターンファイル内に加えて下さい：

```
V1 VDD 0 5V
V2 VSS 0 0V
```

これによりVDD 及びVSS を全ての階層でグロ－バル電源として使用することができます。

➔ グローバルに設定しただけでは電源の意味を持ちません。もし上記の例から電圧表記を削除してしまった場合、ノードVDD及びVSSはハイインピーダンスとなります。

グローバルなデジタルノードありません。これはVerilog言語のパートではありません。もし、モジュール内に電源が必要な場合は、supply0及びsupply1ネットを使用するのが最も簡単です。

回路の作画

アドバンストSimでデザインのシミュレートを行う第一段階は、回路図入力です。アドバンストSimでは設計で使用された回路の要素について、シミュレーションモデルのような特別な情報を必要とします。\Client98\sch98\library\sim のディレクトリに含まれるコンポーネントのライブラリを使用して設計を行えば、シミュレーションがうまくいくはずです。

コンポーネントライブラリ

アドバンストSimに含まれるコンポーネントライブラリには2つのタイプがあります。1つはシンボリックライブラリで回路図に使われ、もう1つのモデルライブラリはシミュレーション用の特性が定義されています。アドバンスト Simには、参照するモデルが設定されているシンボルライブラリとモデルライブラリがセットで含まれています。これらのシンボルライブラリは\Client98\sch98\library\sim のディレクトリにインストールされます。

回路上で使用するエレメントはシミュレーションモデルを必要としないコンポーネントと必要とするものの2つのグループに分けられます。

電圧ソース、抵抗、コンデンサ、コイルのようなモデルを必要としないコンポーネントの場合、状況は単純です。ただ数的な値（例えば抵抗の値）だけが必要とされ、これらの値はコンポーネントのアトリビュートとして具体化されます。

シミュレーションモデルを必要とするコンポーネントの例は、トランジスタ、アンプ、コンパレータなどです。もし、設計がモデルを必要とするコンポーネントを含んでいる場合、参照するモデルライブラリが設定されているコンポーネントシンボルを使うかシミュレーターをセットアップし、適したモデルライブラリを指定します。

モデルライブラリを参照するコンポーネントシンボル

\Client98\sch98\library\sim のディレクトリのライブラリの中のすべてのシンボルは適切なモデルライブラリを参照するよう設定されています。これらのライブラリのうちの1つからシンボルを配置した後でアトリビュートを編集するために、そのシンボルをダブルクリックして下さい。参照するモデルの記述がリードオンリーフィールドのフィールド1にあります。そこには次のような文字列が記入されています。

```
pristel=\Opamps\Motorola\<parttype>.ckt
```

アドバンストSimはこの文字列で2つの事を行います。1つは、<parttype>の文字列をコンポーネントのパートタイプに置き換えます。次にこの文字列に*advsim*.INIの [Protel] セクションにある `pristel_root_directory' で指定されている文字列が付加されます。モデルライブラリへの完全なパスは*design*.BILファイルに .LIBディレクトリとして記憶されます。次に示すのは自動的に .BILファイルに書き込まれた .LIB ディレクティブの一例です。

```
.LIB c:\client98\sim98\library\Opamps\Motorola\LM324.ckt
```

➔ モデルライブラリがどのように参照されるかを詳しく知るには、*ライブラリとデバイスモデル* の章を参照して下さい。

回路図を入力する際の重要なポイント

- 回路図のリファレンスポイントをグランド、または0ボルトにしなければなりません。
- 回路図中の全てのノードは、グランドへのDCパスを持っていなければならない。
- 全てのコンポーネントとソースは正しくつながなければなりません。アドバンストSimは、接続されていないピン（とりわけソースピン）があると動作しません。
- デジタルの回路図でVCCやGNDが表われても、それらを個々の供給品として含める必要はありません。

電源と入力波形の供給

スケマティックでの入力が終わったら、回路に電源と入力波形を定義します。アドバンストSimには、このタスク用にアナログ・デジタルのスティミラス／ソースを配置するツールバーがあります。もし必要とする適当なソースがツールバーに存在しない（たぶん周波数が正しくないと考えられます）場合は、最も近いものを配置し、それをダブルクリックして必要なアトリビュートに修正します。

➔ ソースアトリビュートはAC解析と過渡解析の両方の値を定義します。例えば、VSINソースには AC Magnitude（ACマグニチュード）や Amplitude（振　幅）アトリビュートがあります。ACマグニチュードはAC小信号解析に使用され、振幅は過渡解析に使用されます。典型的な設定は回路図の例と次の表を参照して下さい。

➔ *回路プリミティブとスティミラス*の章も参照して下さい。

アナログソースとスティミラス

アドバンストSimは次のアナログソースとスティミラス（入力波形）をサポートしています；

- コンスタントソース
- 周期的なパルスソース
- ピース・ワイズ・リニアソース
- 正弦波ソース

➔ それぞれのソースはグランド（GND）か0V（ボルト）へ接続して下さい。

コンスタントソース

回路の電源としてコンスタントソースを使います。ソースを配置した後、次にあげるアトリビュートを指定する必要があります；

アトリビュート	設定
Designator	必要な部品のレールネームを設定する（VCC等）
DC Magnitude (v)	必要なDC電圧（5や-12）
AC Magnitude (v)	小信号AC解析を行う場合に設定する（1ボルト）
AC Phase (degrees)	小信号電圧の位相
Transient Spec	（このアトリビュートは無視）

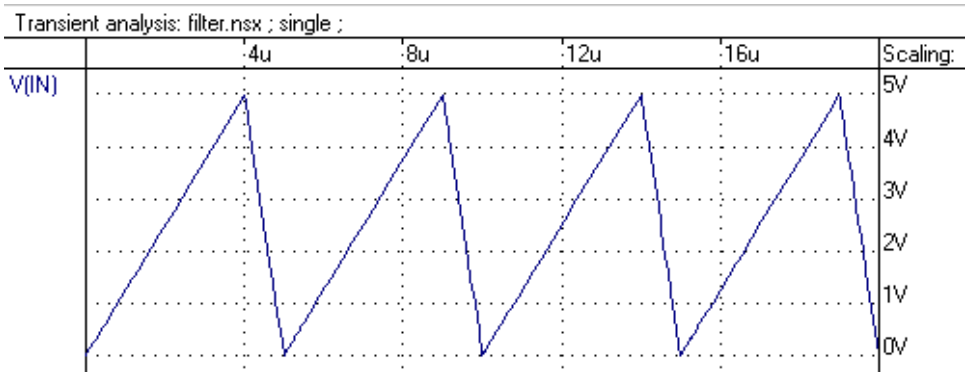
➔ 値だけを入力して下さい。単位は必要ありません。必要があればスケールサフィックスを付けます。（例えばMEG、uなど）スケールサフィックスについての詳細は、*アドバンストSimを始めるにはのコンベンション*のトピックを参照して下さい。

周期的なパルスソース

連続的に繰り返されるパルス波を定義するため、周期的なパルスソースを使います。ソースを配置した後で次のアトリビュートを指定する必要があります；

アトリビュート	設定
Designator	必要な波形の名称をセットする（例 VIN）
DC Magnitude (v)	（このアトリビュートは無視）
AC Magnitude (v)	AC小信号解析をする場合は設定する（通常は1ボルト）
AC Phase (degrees)	小信号AC電圧の位相
Initial Voltage (v)	時間が0のときの電圧 （例 0）
Pulsed Voltage (v)	遅延と上昇の時間の合計した時の電圧（例 5）
Delay (s)	ソースが初期電圧から波動電圧まで変化する前の遅延
Rise Time (s)	初期電圧から波動電圧まで上昇するのにかかる時間で、0より大きな値（例 4u）
Fall Time (s)	波動電圧から初期電圧まで下降するのにかかる時間で、0より大きな値（例 1u）
Pulse Width (s)	波動電圧にソースが残っている時間（例 0）
Period (s)	最初の波動の始まりと次の波動の始まりの間の時間（例 5u）

➔ 値だけを入力して下さい。単位は必要ありません。必要があればスケールサフィックスを付けます。（例えばMEG、uなど）スケールサフィックスについての詳細は、*アドバンストSimを始めるにはのコンベンション*のトピックを参照して下さい。



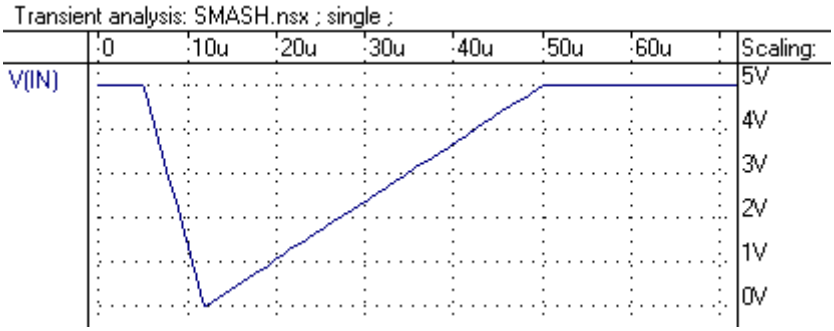
周期的なパルスソースを使用して発生させた電圧のサンプル。上記、表の値を使用。

ピースワイズリニアソース

様々な時点での電圧の設定のためのソースとして、このピースワイズリニアソースを使います。ソースを設定した後で次のアトリビュートを指定する必要があります；

アトリビュート	設定
Designator	必要な波形の名称をセットする（例 VIN）
DC Magnitude (v)	（このアトリビュートは無視）
AC Magnitude (v)	小信号AC解析をする予定の場合は設定する （通常1ボルト）
AC Phase (degrees)	小信号電圧の位相
Time/Voltage Pairs (s v)	それぞれの時間における電圧を具体的に記す（例 0U 5V 5U 5V 12U 0V 50U 5V 60U 5V）

➔ 値だけを入力して下さい。単位は必要ありません。必要があればスケールサフィックスを付けます。（例えばMEG、uなど）スケールサフィックスについての詳細は、*アドバンストSimを始めるにはのコンベンション*のトピックを参照して下さい。



PWLソースを使用して発生させた電圧のサンプル。Time/Voltageの値は、0U 5V 5U 5V 12U 0V 50U 5V 60U 5Vに設定。

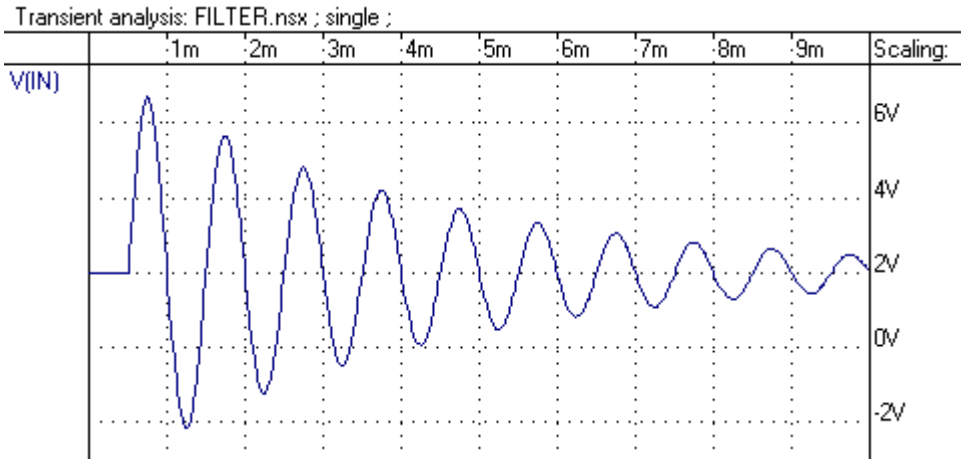
正弦波ソース

正弦波ソースを定義するために使います。ソースを設定した後で次のアトリビュートを指定する必要があります；

Attribute	設定
Designator	必要な波形の名称をセットする（例 VIN）
DC Magnitude (v)	（このアトリビュートは無視）
AC Magnitude (v)	小信号AC解析をする予定の場合は設定する （通常1ボルト）
AC Phase (degrees)	小信号電圧の位相
Offset Voltage (v)	DCオフセット電圧（例 2）
Amplitude (v)	電圧の振幅のピーク（例 5）
Frequency (Hz)	電圧の周波数（例 1000）
Delay (s)	電圧ソースが始動するまでの遅延時間（例 1m）
Damping Factor (1/s)	正弦曲線の幅が減少する率を設定する（例 250）
Phase (degrees)	時間が0の時の正弦曲線の位相の変化（例 0）

ソースの計算式は；

V(t) = offs+ampl•exp(-(t-delay)•damp)•sin(2•pi•(f.(t-delay)+phase/360))



正弦波ソースを使用して発生させた電圧のサンプル。アトリビュートの設定は、上記の表の値を使用。

アナログの電圧ソースと電流ソースのシンボル

シミュレーションソースツールバーを押したときに配置された電圧ソースは、次のライブラリから読み込まれます；

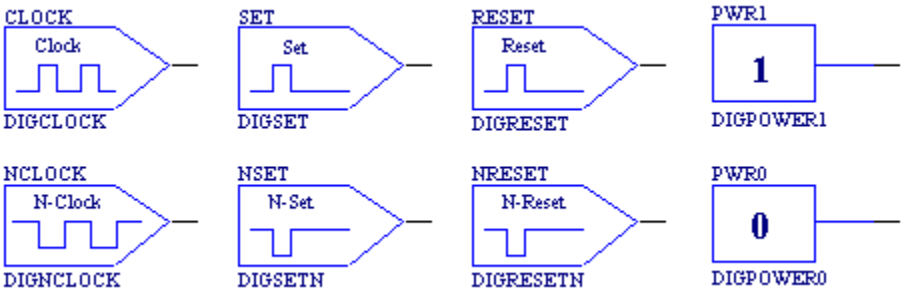
```
c:\client98\sch98\library\symbols.lib
```

このライブラリには電流ソースも含まれています。同じアトリビュートには電圧ソースと電流ソースを具体的に入力する必要があります。ただ、前述の例において電圧と電流を置き換えるだけです。

デジタルスティミラス

デジタル回路では VCC と GND は自動認識されます。この為、特別に電圧ソースを配置する必要はありません。

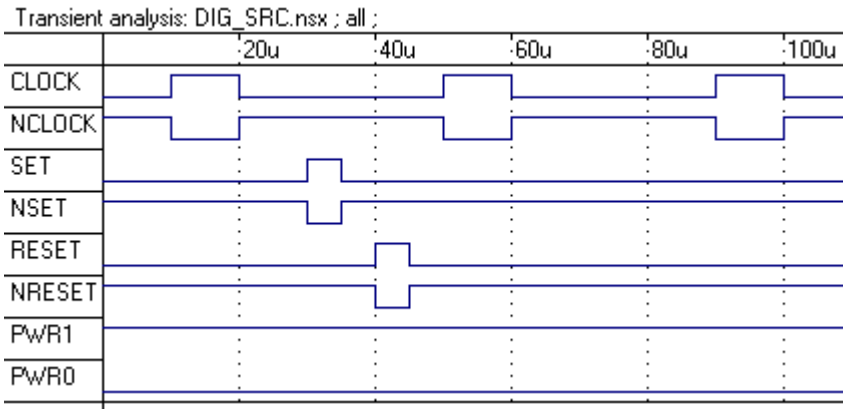
アドバンストSimには、次のようなデジタル入力シンボルが用意されています；



これらのシンボルは、次のライブラリファイルにあります；

```
c:\client98\sch98\library\symbols.lib
```

これらのシンボルを配置した時に書き込まれるアトリビュートについては、次のセクションで説明します。



各デジタルスティミラスのサンプル波形。アトリビュートはそれぞれに記載されている表の値を使用

Digclock と DigNclock（クロック）

これらのクロックシンボルは、連続する波形の立ち上がりと立ち下がりの定義に使用されます。これらは、単一波形と繰り返しの属性を備えています。

Attribute	設定
Designator	スティミラス名の定義（例えばCLOCKや NCLOCK）
Rising_Edge_Time	波形の立ち上がり点までの時間（例 10000）注：0より大きな値である事
Falling_Edge_Time	波形の立ち下がり点までの時間（例 20000）
Period	連続波形の周期（例 40000）

➔ アドバンストSim ではナノセカンド（ns）が時間を定義する単位として使用されている為、全ての時間の定義を、ナノセカンドを用いて入力しなくてはなりません。.LTIMESCALE ディレクティブをパターンファイルに追加する事により、これを変更する事ができます。この詳細はオンラインヘルプの*ディレクティブ*のトピックに示されています。

Digset と DigNset（セット）

これらのシンボルは単一ショットパルスの立ち上がりと立ち下がりの定義に使用されます。

Attribute	設定
Designator	スティミラス名の定義（例 SET や NSET）
Rising_Edge_Time	波形の立ち上がり点までの時間（例 10000） 注：0より大きな値である事
Falling_Edge_Time	波形の立ち下がり点までの時間（例 20000）

➔ アドバンストSim ではナノセカンド（ns）が時間を定義する単位として使用されている為、全ての時間の定義を、ナノセカンドを用いて入力しなくてはなりません。.LTIMESCALE ディレクティブをパターンファイルに追加する事により、これを変更する事ができます。この詳細はオンラインヘルプの*ディレクティブ*のトピックに示されています。

Digreset と DigNreset （リセット）

これらのシンボルはリセットパルスの立ち上がりと立ち下がりの定義に使用されます。

Attribute	設定
Designator	スティミラス名の定義（例 RESET や NRESET）
Rising_Edge_Time	波形の立ち上がり点までの時間（例 10000）注：0より大きな値である事
Falling_Edge_Time	波形の立ち下がり点までの時間（例 20000）

➔ アドバンストSim ではナノセカンド（ns）が時間を定義する単位として使用されている為、全ての時間の定義を、ナノセカンドを用いて入力しなくてはなりません。.LTIMESCALE ディレクティブをパターンファイルに追加する事により、これを変更する事ができます。この詳細はオンラインヘルプの*ディレクティブ*のトピックに示されています。

Digpower1 と Digpower0（ロジック）

これらのシンボルはデジタル回路に1または0を入力する場合に使用します。

Attribute	設定
Designator	スティミラス名の定義（例 PWR1）

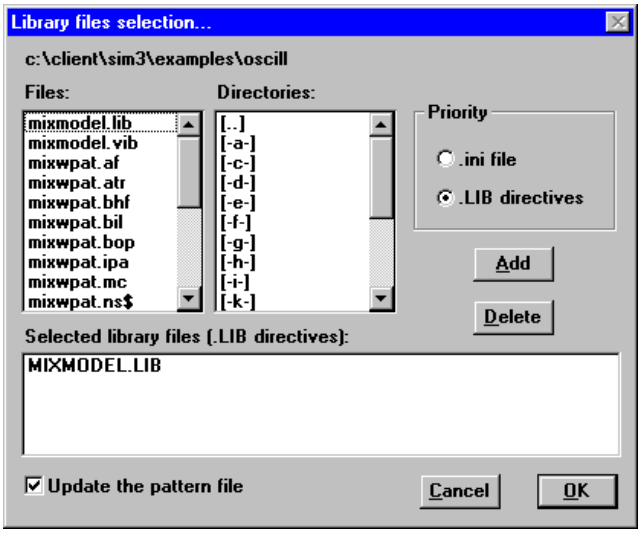
シミュレーターのセットアップと起動

回路図を書き終えた後セットアップを行い、シミュレーションを開始します。

モデルライブラリの選択

回路をシミュレーションする為にアドバンストSimでは、回路図上のそれぞれの部品の数学的モデルが必要になります。これらのモデルはモデルライブラリに格納されています。回路図シンボルとモデルとを結び付けるには2つの方法があります；

1. `c:\client98\sch98\library\sim`フォルダの中のシンボルライブラリを使用する。これらのシンボルにはそれぞれモデルを関連付ける属性が含まれています。



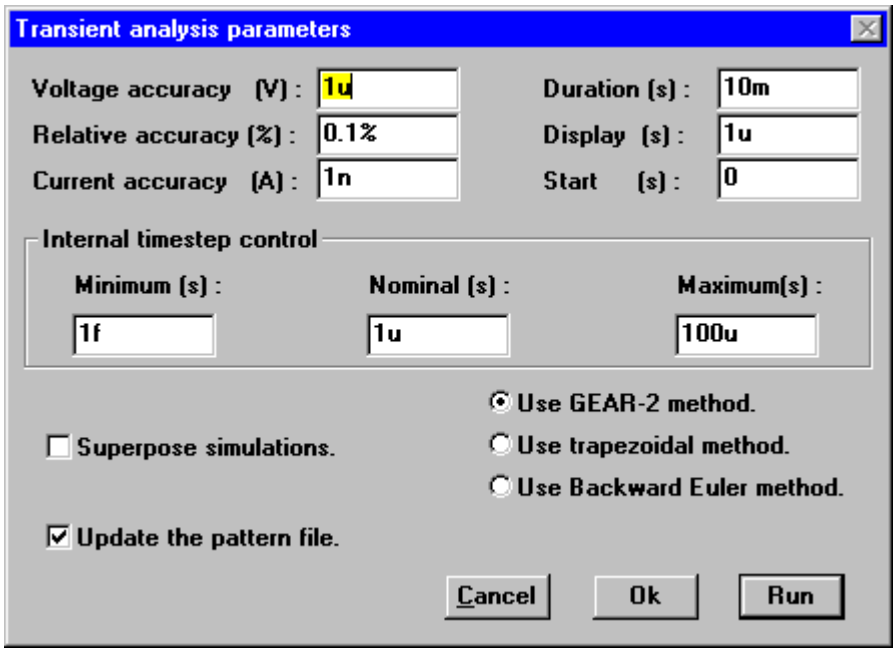
2. 自分自身でモデルを関連付ける。これを行なうには Simulateメニューから Setup Simulator - Select Model Libraries を選びます。この操作により Library Files Selectionダイアログボックスが表示され、ここで必要なライブラリをみつけ、 Selected library files listに追加します。この情報は .LIB ディレクティブとして、 パターンファイルに追加されます。全てのモデルライブラリは、`c:\client98\sim98\library`ディレクトリに格納されています。

➔ .LIB ディレクティブに関する詳細は、*ディレクティブ*のトピックを参照して下さい。また*ライブラリとデバイスモデル*の章にはモデルライブラリについての詳細が示されています。

過渡解析

アドバンストSimの過渡解析では 、ユーザーが指定した時間（間隔）における出力の変化を計算します。初期の回路状態がDC解析により計算された後、Transient analysis parametersダイアログボックスに設定された時間（間隔）に渡って動作を解析します。

アドバンストSimの過渡解析の設定を行なうには、Simulateメニューから Setup Simulator - Transientを実行します。これでTransient analysis parametersダイアログボックスが表示されます。これに対して必要なフィールドのパラメータを書き込みます。



このダイアログボックスの設定内容はパターンファイル（*circuit*.PAT） にディレクテイブとして書き込まれます。以下にこれらの機能とディレクティブについて説明します。

Internal timestep control（インターナルタイムステップコントロール）

これらの設定項目はシミュレーションエンジンに使用されるタイムステップをコントロールします。アドバンストSimでは信号の変化の速さに合わせて、タイムステップを最小値と最大値で調整する事ができます。適切なタイムステップパラメータを選ぶ事はシミュレーションを成功させる為の重要なポイントです。以下のガイドラインは適切なパラメータの設定に役立ちます：

1. 最初の設定として、Nominal timestep を信号周期の1000分の1に設定します。例えば、1kHzの正弦波の場合には周期が1ms（time=1/frequency）になるため、これを1000で割った1us－これが適切な値となります。もし、信号が間欠的でない場合（例えばパルスなど）パルス幅の2倍を周期と仮定します。
2. Maximum timestep は Nominal valueの100倍に設定します。
3. ほとんどの場合、デフォルトのMinimum timestepで、良好なシミュレーションが行われます。

シミュレーションがあまりにも遅い場合、まずMaximum timestepの設定値を10倍にして下さい。これでも解決しない場合は、Nominal timestep の設定値を10倍にして下さい。

➔ これらの設定はパターンファイルに .H ディレクティブのパラメータとして保存されます。詳細は、オンラインヘルプの*ディレクテイブ*のトピックを参照して下さい。

Duration, Display, Start（表示、表示間隔、開始）

これらの設定は、ウインドウに表れるシミュレーション結果の表示コントロールの為に用いられます。以下のガイドラインは適切なパラメータの設定に役立ちます。

1. Durationは結果の表示時間を定義します。初期値として10サイクルの周期が表示される設定です。例えば、1kHzの正弦波の場合は10msとなります。
2. Display は波形表示の為のプロットポイントの間隔を設定します。ソースとなる波形の1000分の1（Nominal timestepと同じ値）に設定するのが適当で、1kHzの場合にはこの設定（1us）により1サイクルあたり1000 ポイントの表示が行なわれます。この設定で波形の表示があまりにも遅い場合、設定値を10倍（この例では10us）に増加させます。

➔ これらの設定は、パターンファイルに .TRAN ディレクテイブのパラメターとして保存されます。詳細はオンラインヘルプの*ディレクテイブ*のトピックを参照して下さい。

Accuracy（精度）

timestepとaccuracy の双方の設定は、正確で高速なシミュレーションに対してクリチカルに作用します。また多くの場合、デフォルトの設定で満足な結果を得る事ができます。

➔ これらの設定は、パターンファイルに .EPS ディレクテイブのパラメータとして保存されます。詳細はオンラインヘルプの*ディレクテイブ*のトピックを参照して下さい。

Method（メソッド）

アドバンストSim は、過渡解析の為に3つの統合アルゴリズムを備えています。デフォルトのメソッドはGEARであり、ほとんどの場合この設定で満足な結果が得られます。

➔ それぞれのメソッドのもつ利点や弊害については、詳細はオンラインヘルプの*ディレクテイブ*のトピックの .METHODディレクティブを参照して下さい。

過渡解析の実行

Transient analysis parameters ダイアログボックスの設定が終った後、Run ボタンを押しシミュレーションプロセスを起動します。（Simulateメニューの Transient Analysis や ツールバーの Run Transient Analysis ボタンによっても、シミュレーターを起動する事ができます。）初めてシミュレーションを行う場合、次のようなワーニングメッセージが表示されます；

> “No signals will be displayed during simulation. Would you like to set up the simulation screen now?” （「シミュレーションの間、どの信号も表示されません。シミュレーション画面の設定をここで行いますか？」）

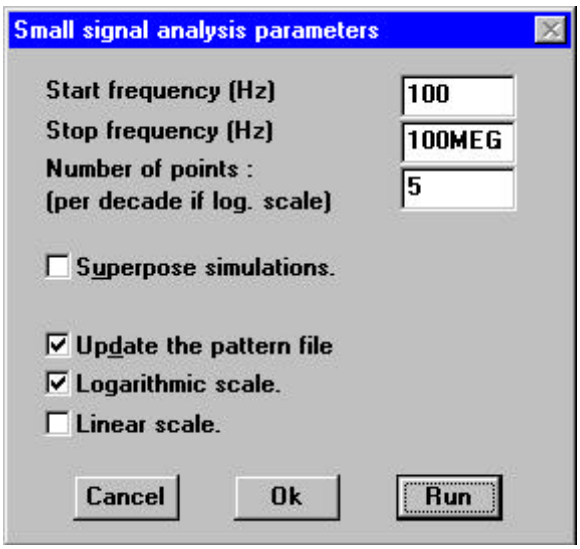
ここで Yes ボタンをクリックし、表示させたい信号名を設定します。この章の後の波形の設定*波形ウインドウの設定*のトピックに波形表示のコントロールについて説明されていますので参照して下さい。

適切な信号を選択した後 "Time" という名称の波形ウインドウが開きシミュレーションの結果が表示されます。

AC小信号解析

アドバンストSimのAC小信号解析では、周波数の変化による出力の変化を計算します。この際、まずDC動作点の計算と、回路中のノンリニアデバイスに用いる小信号モデルのイニシャライズが行なわれます。このサーキットモデルは Small signal analysis parameters ダイアログボックスによって設定された周波数の全域にわたり解析されます。

アドバンストSimのAC小信号解析の設定を行なう場合、Simulateメニューより Setup Simulator - AC Small Signal を選択します。これによりSmall signal analysis parametersダイアログボックスが表示され、この中のフィールドにパラメータを設定します。



シミュレーションを起動する前にこのダイアログボックス上での設定を行なわなかった場合、アドバンストSimが自動的にパラメータを割り当てます。この場合にはおそらく、回路に合った値には設定されず、無意味な結果が波形として出力されます。もし、シミュレーション結果に問題があった場合、必ずこの設定をチェックして下さい。

➔ このダイアログボックスの設定はパターンファイル（*circuit*.PAT）に .AC ディレクティブとして保存されます。パターンファイルの詳細については、*アドバンストSim ファイル* の章を、.AC ディレクティブについての詳細はオンラインヘルプの*ディレクティブ*のトピックを参照して下さい。

AC小信号解析の実行

Small signal analysis parametersダイアログボックスの設定が終った後、Runボタンを押してシミュレーションプロセスを起動します。（Simulateメニューの Small signal analysis や、

ツールバーの Run Small signal analysis ボタンによっても、シミュレーターを起動する事ができます。）もし、初めてシミュレーションを実行する場合には、次のようなワーニングメーッセージが表示されます；

> “No signals will be displayed during simulation. Would you like to set up the simulation screen now?”

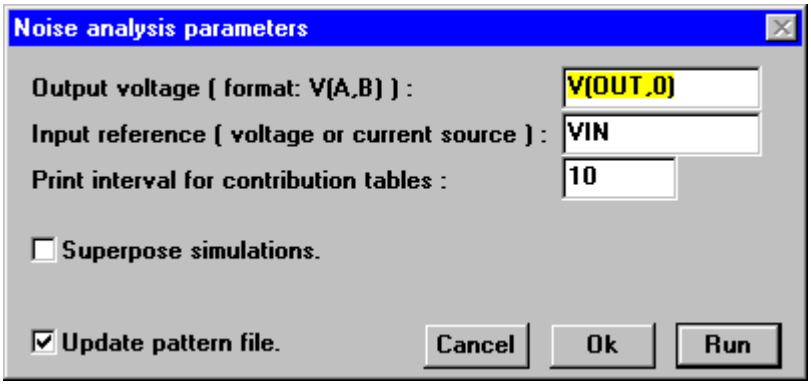
ここで Yes ボタンを選択して、表示させたい信号名を設定します。この章の後の*波形ウインドウの設定*に波形表示のコントロールについて説明されていますので参照して下さい。

適切な信号を選択した後 "AC" という名称の波形ウインドウが開きシミュレーションの結果が表示されます。

ノイズ解析

アドバンストSimのノイズ解析では、 デバイスが回路上に発生させるノイズを計算します。この解析では出力に発生する全てのノイズが実効値（RMS）で加算され、また特定の入力ソースに対する入力換算ノイズの値が計算されます。 この解析はSmall signal analysis parametersダイアログボックスで設定された周波数レンジの各々のポイントで行なわれます。（前項の*AC小信号解析*を参照）

アドバンストSimのノイズ解析の設定を行なうにはSimulateメニューよりSetup Simulator - Noiseを選択します。これによりNoise analysis parametersダイアログボックスが表示され、この中のフィールドにパラメータを設定します。



Output voltage の欄には、回路図上の計測するAとBの2点のノードをV（A, B）のように示しこの差分を結果として出力します。この例では回路図上のネットラベル "OUT"、とパワーポートによるグランドネットが "0"と示されています。input reference には電圧または電流ソースの名前を入れます。

計算されたノイズの値は、結果が波形表示されると同時に*circuit*.NZEの名称のファイルに保存されます。例に示したダイアログボックスでのプリントインターバル設定では*circuit*.NZEファイルに10ポイントごとにノイズ値が記録されます。またそれぞれの周波数ポイントでは、ノイズの発生源となっている10個のデバイスのリストが簡潔に示されます。

➔ このダイアログボックスの設定は、パターンファイル（circuit.PAT）に . NOISEディレクテイブとして保存されます。パターンファイルについての詳細は、*アドバンストSim ファイル*の章を、.NOISEディレクティブについての詳細は、オンラインヘルプの*ディレクテイブ*のトピックを参照して下さい。

ノイズ解析の実行

Noise analysis parametersダイアログボックスの設定が終った後、Runボタンを押してシミュレーションプロセスを起動します。（SimulateメニューのNoise analysisや、ツールバーのRun Noise analysisボタンによっても、シミュレーターを起動する事ができます。）もし、これが一番最初のシミュレーションである場合には、次のようなワーニングメッセージが表示されます；

> “No signals will be displayed during simulation. Would you like to set up the simulation screen now?”

ここで Yes ボタンをクリックし、表示させたい信号名を設定します。この章の後の*波形ウインドウの設定*に波形表示のコントロールについて説明されていますので参照して下さい。

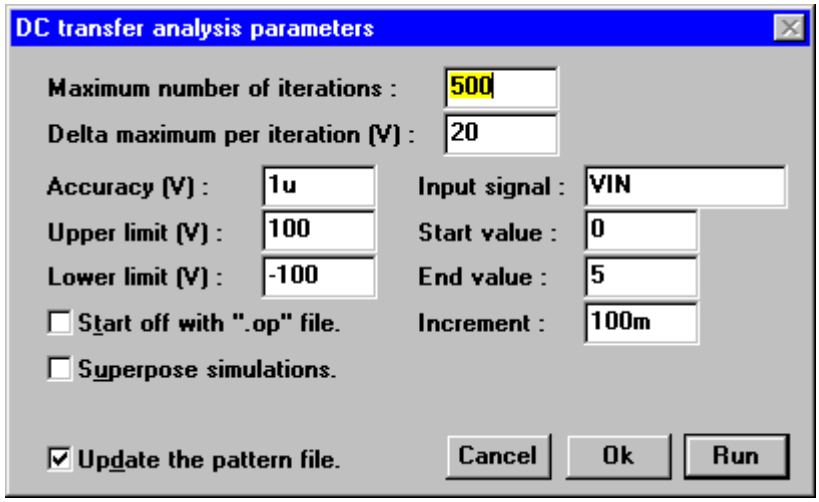
適切な信号を選択した後 "Noise" という名称の波形ウインドウが開き、シミュレーションの結果が表示されます。

DCトランスファー

アドバンストSimのDCトランスファー解析は、入力電圧ソースを開始値から終了値まで（Incrementで指定された値で）変化させ、連続したオペレーティングポイント解析を行ないます。

DCトランスファー解析を行う前に、アドバンストSimはインダクタがショートし、キャパシタがオープンになった回路のDCオペレーティングポイントを計算します。

アドバンストSimでDCトランスファーシミュレーションのセットアップを行うには、メニューから Simulate - Setup Simulate - DC Transfer のコマンドを選択して下さい。 DC transfer analysis parameters のダイアログボックスが表示されますので、必要なフィールドを設定して下さい。



➔ Input、Start、End 及び Increment の設定は、DCディレクティブとしてパターンファイル（*circuit*.PAT）に保存されます。 IterationとAccuracyの設定はパターンファイルの .OPディレクティブの一部として保存されます。次の項目は、オペレーティングポイント解析の参考として読んで下さい。パターンファイルの詳細については*アドバンストSimファイル*の章を、.DC及び .OPディレクティブの詳細についてはオンラインヘルプの*ディレクティブ*のトピックを参照して下さい。

DC トランスファー解析の実行

DC transfer analysis parameters のダイアログボックスの設定が終わったら、Run ボタンを押して（又はメニューから Simulate - DC Transfer Analysis を選択して）シミュレーションプロセスを開始します。その回路が初めてシミュレートされる場合は、次のようなワーニングメッセージが表示されます;

> “No signals will be displayed during simulation. Would you like to set up the simulation screen now?”

ここで Yes ボタンをクリックし、表示させたい信号名を設定します。この章の後の*波形ウインドウの設定*に波形表示のコントロールについて説明されていますので参照して下さい。

表示させたいシグナルを選択すると、“DC”波形ウインドウが表れシミュレーションの結果が表示されます。

モンテカルロ解析

アドバンストSimでは、先に説明したそれぞれのシミュレーションタイプについてモンテカルロ解析を行うことができます。モンテカルロ解析を行うとコンポーネントの値が表示値と少し異なっている場合の回路のビヘービアをチェックする事ができます。

モンテカルロ解析を選択すると、コンポーネントの値を指定された許容範囲（トレーランス）内でランダムにアジャストし、複数のシミュレーションを行います。複数のシミュレーション結果が同じウインドウに表示されますので、パラメータの許容範囲が出力波形に与える影響を確認することができます。

各々のシミュレーションは、単独でシミュレーションを行うのと同じコントロールパラメータを使用します。パラメータの値は、指定されたディストリビューションのタイプに基づいて名目上の値と付帯する許容範囲から計算されます。アドバンストSimでは、許容範囲の偏差が全体の範囲内で均一に生成される*uniform*とトレーランスの偏差がゴーシアンディストリビューションで広がる*Gaussian*の二つのタイプのスタティカルディストリビューションをサポートしています。

アクティブデバイスでは、コンポーネントの許容範囲はモデルの一部です。ディスクリートデバイスでは、許容範囲はコンポーネントの値とともに次のように表示されます；

VALUE/TOLERANCE　　（例： 100K/10%）

表記する文字中にスペースを空けない事と、%のシンボルを使用することに注意して下さい。

許容範囲は*circuit*.BIL ファイルの .MONTECARLOディレクティブとして保存され、回路図をシミュレーターに読み込むときにパターンファイルに含まれます。 各々のシミュレーションに使用したパラメータ値をレポートするために*circuit*.MC ファイルが自動的に生成されます。

モンテカルロ解析の実行

モンテカルロ解析はすべてSimulate - Monte Carloのサブメニューから実行することができます。トランジェントとAC小信号モンテカルロ解析は Simulation Toolsのツールバーからも実行することができます。

➔ .MONTECARLOと .RUNMONTECARLOの詳細については、オンラインヘルプの*ディレクティブ*のトピックを参照して下さい。

スイープ

スイープ解析を実行すると、アドバンストSimは特定のコンポーネントの値を変化させて一連のシミュレーションを行います。これは、一つのコンポーネントの値を変えて回路のパフォーマンスを検証する場合に役立ちます。スイープ解析は、過渡、ノイズ、AC小信号やDCトランスファーの各解析に使用する事ができます。

スイープ解析はSimulateメニューの Sweepサブメニューから実行します。複数のスイープ解析結果が同じウインドウに表示されますので、パラメータの変化が出力波形に与える影響を確認することができます。各々のシミュレーションは、単独のシミュレーションを行うのと同じコントロールパラメータを使用します。スイープ解析には、リニア、ロガリズミック、及びリストスイーピングという三つの種類があります。

スイープ解析を実行するためには、パターンファイルの中に .PARAMSWEEP（又は .STEP）というディレクティブが必要です。これらのディレクティブについての詳細はオンラインヘルプの*ディレクティブ*のトピック、パターンファイルについての詳細は、*アドバンストSimファイル*の章を参照して下さい。

スイープするパラメータは .PARAMディレクティブで規定されているどのパラメータ（コンポーネント値）でも構いません。（ .PARAMディレクティブで規定されている）通常のパラメータ値は、カレントスイープ解析の値に書き換えられます。

➔ .PARAMSWEEPディレクティブは一つだけ実行することができ、累積する事はできません。

スイープ解析の実行

スイープ解析は、Simulate - Sweep のサブメニューから実行します。 スイープ解析が終わったら、各々の波形で使用された実際のコンポーネント値を決定する事ができます。

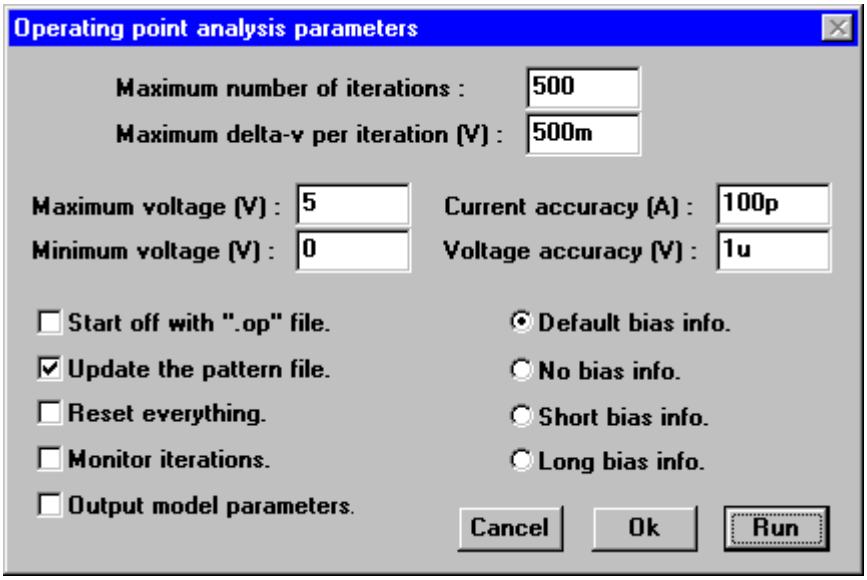
オペレーティングポイント解析

アドバンストSimでは、過渡解析や小信号解析を行う前に、先ずオペレーティングポイント解析を行う必要があります。これで各デバイスのDCオペレーティングポイントが決定し、デバイスモデルパラメータを設定することができます。

アドバンストSimでは、このオペレーティングポイントを設定するのに、前もって設定されたアウターリミットから始めてオペレーティングポイントにコンバージ（収束）する試みを繰り返して、オペレーティングポイントを設定します。このオペレーティングポイン

ト解析は自動的に行われ、通常ユーザーが係わることはありません。しかし、アドバンストSimでオペレーティングポイントにコンバージすることができなかった場合、回路をシミュレートすることができず、ユーザーがオペレーティングポイントの設定をマニュアルで行うことになります。

オペレーティングポイント解析を設定するには、Simulate - Operating Point Analysis のメニューを選択します。Operating point analysis parametersのダイアログボックスが現れます。ここでのセッティングは .OPディレクティブのパラメータとして保存されます。



オペレーティングポイントアナライシスパラメータダイアログ

Runボタンをクリックすると解析が実行されます。Monitor irerationsオプションを選択すると、波形ウインドウが現れ繰り返しの経過を見ることができます。コンバージする作業が完了すると*circuit*.OPというファイルが作成され画面にオープンします。このファイルにはバイアスポイントの記述が含まれています。 すでに *circuit*.OPというファイルが存在している場合は、それは*circuit*.BOPという名前で再度セーブされます。

OKボタンをクリックすると、ダイアログボックスで入力したデータは .OPディレクティブに保存されますが、シミュレーションは実行されません。

➔ オンラインヘルプのディレクティブのトピックの .OP ディレクティブの項目を参照して下さい。

繰り返しの最大数の設定

Maximum number of iterations のフィールドで、アドバンストSimがオペレーションポイントにコンバージできないと判断するまでに行なう試行回数を設定します。 コンバージできないと判断されると、解析は中止され、ヘッダーにUNABLE TO CONVERGEというメッセージの入ったcircuit.OPファイルが生成されます。この場合、*circuit*.OPファイルに含まれているバイアス情報は中止する直前に得られた点であり、最終的な結果ではありません。

➔ このフィールドは、回路がコンバージしない場合に使用する特殊なアルゴリズムを選択するためにも使用します。このセクション後半のトラブルシューティングの項目を参照して下さい。

試行ごとの変化を制限する

Maximum delta-v per iteration のフィールドで各試行間の電圧の差を制限します。このパラメータを使うことによって、繰り返しのプロセスでのオシレーションを防ぎコンバージがされやすくなり、また、オペレーティングポイント発見のためのアルゴリズムを引き出すことができます。このセクション後半のトラブルシューティングの項目を参照して下さい。

最小電圧と最大電圧

アドバンストSimは、この Minimum Voltage と Maximum Voltageで設定された電圧の範囲内でオペレーティングポイントを探します。必ず、回路内での最大のマイナス電圧と最大

プラス電圧を設定して下さい。（ユーザーがマニュアルで設定しない場合、自動的にそのような最小・最大値が設定されます。）

➔ 最小・最大電圧を自動的にチェックした結果が（電圧ソースが規定されていないネットリストのように）ゼロであった場合、各々 -100V と 100V に設定されます。これらの値が適当でない場合、マニュアル入力して修正して下さい。

精度のコントロール

オペレーティングポイント解析は、f（v）= 0 を解く数値アルゴリズムです。この種の数値アルゴリズムはコンバージのための基準を設定する必要があります。アドバンストSimでは、オペレーティングポイント解析のコンバージの基準は Current accuracy（次に Voltage accuracy）です。通常、デフォルトで設定されている値がそのまま使用することができます。これらの値を変更する必要がある場合は、この項目の後半の、*動作しない場合の対処*のセクションを参照して下さい。

電圧のモニタリング

解析を行っている間、電圧と電流をウインドウに表示させることができます。 Monitor iterationsのオプションを使用すると、コンバージされていないノードは黒かグレーで表示されコンバージされたノードは赤で表示されます。

.OPファイルからの開始

オペレーティングポイント解析を行うと、直前のシミュレーションから得たノード電圧とバイアスの記述が含まれた*circuit*.OPファイルが生成され、このファイルは後で同じ回路のシミュレーションを行うときに使用することができます。このファイルを使う場合は、Start off with .OP fileのオプションをイネーブルして下さい。

すべてをリセットーシミュレーションをどこから再開するか？

ダイアログボックスから次の解析を行う場合、カレントのバイアスポイント、又はスクラッチから再開することができます。どこから再開するのが良いかはコンバージプロセスの状態によって決まりますが、一般のシミュレーションプログラムではコンバージの状態が解りません。アドバンストSimでは、シミュレーションを行っている間の電圧と電流が計算され表示されますので、うまく行かないときどんな状態であるのかを知ることができます。

時として、シミュレーションの途中で電圧がランダムに振動するのが見えることがあります。この場合シミュレーションを中止して（Simulate - Abort Analysis）、ダイアログボックスの Reset everything オプションをイネーブルし、コントロールパラメータを変更（例えば、delta-Vのパラメータを小さくする）し、その後シミュレーションを再度実行して下さい。

また、電圧はソリューションに近くなっていても振動が続いている場合もあります。この場合は、カレントの値がメモリに記憶されていますのでリセットは行わない方が良く、アルゴリズムがソリューションに近い状態であればその時点でシミュレーションを続けた方が良いでしょう。

➔ ほとんどの場合、*circuit*.OPファイルの残余値の変化を見るだけで振動を発見することができます。Monitor iterationsのオプションは必ず使わなければならない事はありません。

動作しない場合の対処

ある種の回路ではコンバージで問題が発生することがあります。アドバンストSimには、このような場合のために強力な機能が用意されています。以下のヒントを参考にして下さい。

1. まず、設定した最大・最小電圧が正しいかどうかチェックして下さい。高すぎたり、低すぎる値を設定する事はお勧めできません。回路で発生する実際の最大・最小電圧（又はその10%増し）に設定するのがベストです。
2. 残余値の変化を見ることによって最大・最小電圧が適切であるかどうか解ることがあります。これらのパラメータがタイトであれば、通常シミュレーションが繰り返し行われても残余値はまったく変化しません。残余値がこういう傾向であれば、各電圧は適切であると判断できます。

3. 多くのコンポーネントメーカーは、オペアンプ用にSPICEマクロモデルを供給しています。通常、モデルは数値的に解析されなければならない点が無視されています。また、同様に供給される等価回路図にもコンバージの問題があります。レジスタには非理論値を使い、バイポーラトランジスタにはベータパラメータ、そして受動デバイス（E、F、G、H）のゲインを使用して理想的な効果を出しています。結果は、多量のゲインを含むインターナルループがモデルの中に存在することになるか、又はセルのパワーサプライをはるかに越えた値のインターナル電圧が生成されてしまいます。これではシミュレーションができません、というのはデフォルトの最大・最小電圧値がパワーサプライと同じになりアルゴリズムによってコンバージができなくなるからです。この問題が原因であるかどうかは判断が難しいですが、この疑いのある場合は、別のメーカーのオペアンプ（とモデル）に差し替えてシミュレーションを行って下さい。

トラブルシューティング－デフォルト値で動作しない場合

1. アドバンストSimから「unable to converge－コンバージ（収束）できない」というメッセージが出たら、まず回路を見直して下さい。ワイヤリングをチェックし、コンポーネントに適切な値がつけられているか、回路のどこかに未使用のコンポーネンやソースが残されていないかチェックして下さい。さらに、回路にグラウンドリファレンスポイントがあるか、回路の各ノードにはグラウンドへのDCパスがあるかチェックして下さい。チェックの結果問題があれば、（可能であれば）コンバージできる程度にデザインを簡略化してシミュレーションを行って下さい。
2. パターンファイルに .OPディレクティブが無い状態で動作しないことを確認して下さい。
3. Maximum delta-v per iterationのパラメータ値を下げてシミュレーションを再度実行して下さい。 回路が大きくてこのパラメータが極端に小さいと、電圧がソリューションに向かう動きが遅くなりコンバージに至る時間が長くなります。Maximum delta-v per iterationを小さくすると恐らく試行繰り返しの回数が増えるので、Maximum number of iterationのパラメータを大きくする必要があるかもしれません。通常、この方法でコンバージの問題は解決します。
4. アドバンストSimには、コンバージに到達するための多くの特殊なアルゴリズムが含まれています。Maximum delta-v per iterationをゼロにし、Reset everythingをイネーブルして実行してみて下さい。これで試行繰り返しの最初は電圧がゆっくり動き、後半では動きが速くなるような別のアルゴリズムが起動します。Maximum delta-v per iterationをゼロに設定した後は、Maximum number of iteration のパラメータでアルゴリズムの性格をコントロールします。例えば、Maximum delta-v per iterationをゼロに設定し、Maximum number of iterationをまず200に設定し、その後100、300、400、500、1000と値を変えてチェックしてみて下さい。この方法では、コンバージに至る時間は長くなりますが良い結果が得られます。
5. まだシミュレーションが動作しない場合、別のアルゴリズムが用意されています。Maximum delta-v per iterationを－1に設定すると、通常の4－5倍の遅さで動作する第3のアルゴリズムが起動します。このアルゴリズムはシグナルフローを検出し、通常のアルゴリズムで避ける事ができない数値トラップを避ける働きがあります。このアルゴリズムの詳細記述はここでは行いません。
6. バイアス（オペレーティング）ポイントを見つけるために、パワーアップ解析も実行してみて下さい。この方法は過渡解析で、すべてのインプットはゼロからスタートし、徐々にスムーズにそれぞれの最終DC値に達するものです。パワーアップ解析の終了時には、circuit.OPファイルが生成されます。この章のパワーアップ解析（及び .POWERUPディレクティブ）の項目を読んで下さい。

アドバンストSimには、問題のある回路でシミュレーションを行う機能が用意されています。.GMINJUNK、.GDSMOS、.GBDSMOS、.OPHELP、.PIVMIN及び .VSTARTディレクティブを参照して下さい。

bias infoオプションを使用して出力フォーマットをコントロール

bias info オプションを使って、トランジスタ（MOS 及びバイポーラ）のバイアス情報の出力フォーマットをコントロールすることができます。

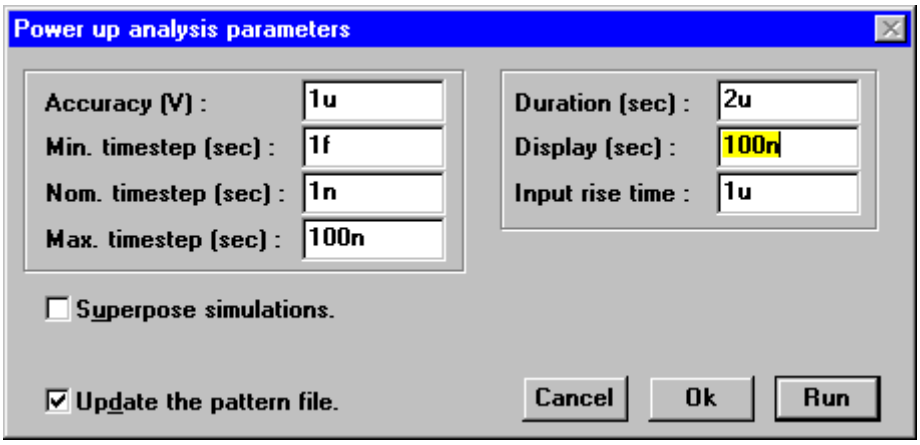
- 何も選択されていない状態では、出力フォーマットは Default bias infoで、これにはスモールシグナルパラメータに関する貴重な情報が多く含まれています。
- No bias infoを選択すると、トランジスタのバイアス情報はまったく出力されません。バイアス情報を見たくない場合、このオプションを使用します。

- Short bias infoを選択すると、バイアス情報は最も基本的な情報のみ、トランジスタ1個につき1行で出力されます。
- Long bias infoを選択すると、さらにパラメータが増えて（デフォルトフォーマットに匹敵する）拡張フォーマットで出力されます。

➔ .OP、.USEOP、.GBDSMOS、.GDSMOS、.GMINJUNC、.OPHELP、.POWERUP の各ディレクティブの項目を参照して下さい。

パワーアップ解析

パワーアップ解析は、パワーサプライのレールをゼロの状態から安定状態の値へ移動させて回路をシミュレートするものです。入力とサプライ（電圧・電流の両方）はDC値に向かってスムーズに増加します。パワーアップ解析は、オペレーティングポイントが見つけられないような回路で、簡単にパワーアップのシーケンスを実現することができます。



パワーアップアナライシスパラメータダイアログ

パワーアップ解析を実行するには、Simulate - power Up Analysisのコマンドを選択して下さい。Power up analysis parametersのダイアログボックスが表示されます。

Input rise time（入力立ち上がり時間）

このフィールドで、ソースの安定状態値に達するまでの時間を設定します。個々のデザインに合った適切な立ち上がり時間を設定して下さい。この設定は、.POWERUPディレクティブのパラメータの一つとしてセーブされます。

Duration（持続時間）

ここではシミュレートさせたい時間を設定します。通常、立ち上がり時間の2、3倍に設定します。この設定は、.POWERUPディレクティブのパラメータとして保存されます。

Display（表示）

ここでは波形のプロット間隔を設定します。立ち上がり時間の100分の1、又は1000分の1に設定します。この設定は、.TRANディレクティブのパラメータとして保存されます。

Accuracy（精度）

シミュレーターがコンバージに到達するポイントを設定します。通常、デフォルトの値のままで大丈夫です。この設定は、パターンファイルの .EPSディレクティブのパラメータとして保存されます。

タイムステップの設定

シミューレーターの使用するタイムステップを設定します。アドバンストSimでは、信号の変化するスピードに基づいて、最大値と最小値の間でタイムステップを調整します。タイムステップの設定は効率的なシミュレーションの鍵となりますので、次のガイドにしたがって設定を行って下さい：

1. 最初に、ノーマルタイムステップを立ち上がり時間の1000分の1に設定して下さい。立ち上がり時間が1usであれば、1nsと設定して下さい。
2. 最大タイムステップ（Maximum timestep）をノーマル値（Nominal）の100倍に設定して下さい。
3. ほとんどの場合、最小タイムステップ（minimum）はデフォルト値で大丈夫です。

シミュレーションのスピードが遅すぎる場合は、最大タイムステップ（Maximum）を10倍単位で増加させて下さい。改善されなければ、ノーマルタイムステップ（Nominal）を10倍単位で増加させて下さい。

➔ これらの設定は、パターンファイルに .Hディレクティブのパラメータとして保存されます。詳細については、オンラインヘルプの*ディレクティブ*のトピックを参照して下さい。

スーパーインポーズシミュレーション

スーパーインポーズのオプションをイネーブルにすると、波形ウインドウの既存の波形の上に、次のシミュレーション結果が重ねて表示されます。

パワーアップシミュレーションの実行

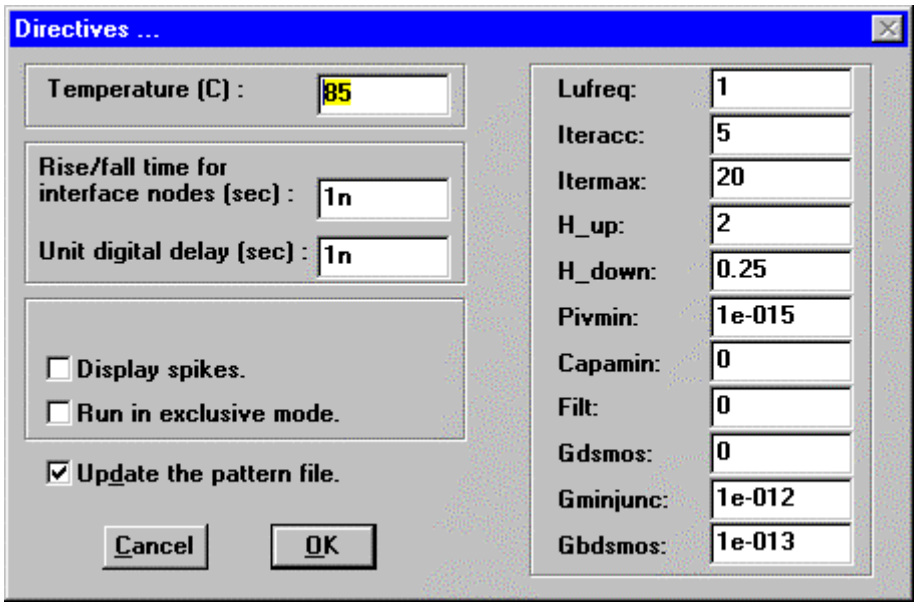
Runボタンをクリックするとシミュレーションが開始されます。シミュレーションの終了時に、*circuit*.OPというファイルが作成されテキストウインドウにオープンします。ここには、次のオペレーティングポイント解析での開始点して使用できる、バイアスポイントの記述が含まれています。

OKボタンをクリックすると、ダイアログボックスに入力したデータがセーブされますが、シミュレーションはスタートしません。

➔ オンラインヘルプの*ディレクティブ*のトピックの .POWERUPディレクティブの項目を参照して下さい

コントロールパラメータ

メニューからSimulate - Setup Simulator - Contorl Parametersを実行すると、ダイアログボックスが表示され、コントロールパラメータを修正する事ができます。それぞれのパラメータは対応するディレクティブがあり、パターンファイルに取り込まれます。 それぞれのデレクティブの詳細についてはオンラインヘルプのデレクティブのトピックを参照して下さい。



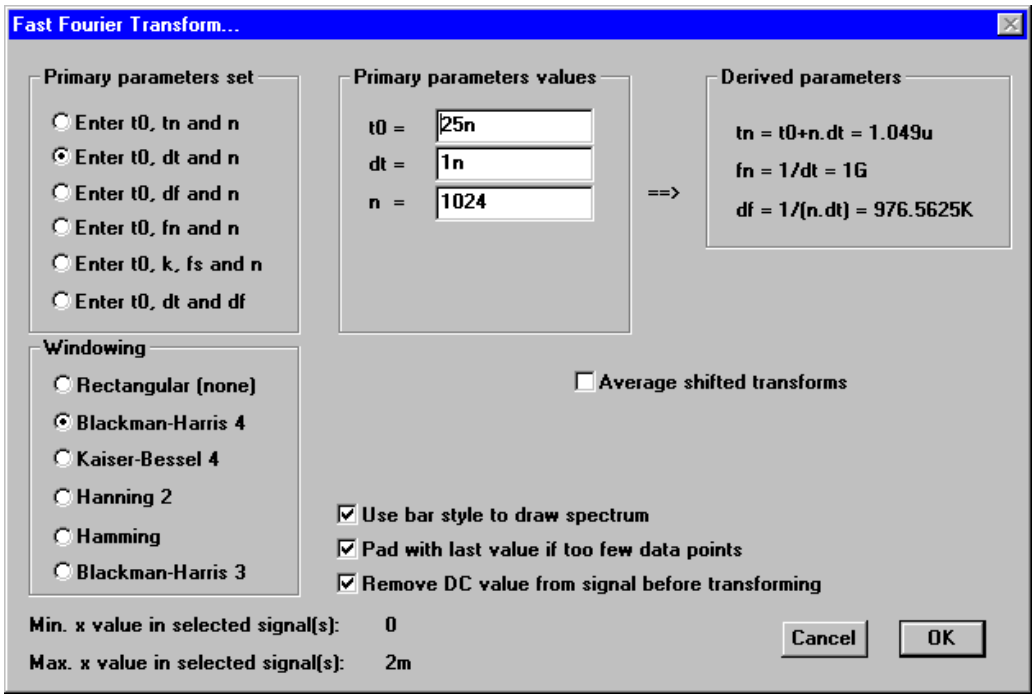
Temperature (°)	.TEMP
Rise/fall time for interface nodes (sec)	.LRISEDUAL
Unit digital delay (sec)	.LTIMESCALE
Display spikes	.VIEWSPIKES
Run in exclusive mode	.EXCLUSIVE
Lufreq	.LUFREQ
Iteracc	.ITERACC
Itermax	.ITERMAX
H_up	.H
H_down	.H
Pivmin	.PIVMIN
Capamin	.CAPAMIN
Filt	.FILT
Gdsmos	.GDSMOS
Gbdsmos	.GBDSMOS
Gminjunc	.GMINJUNC

➔ それぞれのパラメータは、初期値から変更されたものがパターンファイルに保存されます。

高速フーリエ変換（FFT）の計算

FFTを実行するには、信号を選ぶ必要が有ります。信号をクリックした後、Tools - Compute FFTのメニューを選ぶと、Fast Fourier Transform...のダイアログボックスが表示されます。

➔ FFTを行えるのはアナログ信号、スカラデジタル信号とバスタイプデジタル信号です。スカラ信号、ロジック1は、+1.0に、ロジック0は-1.0に割り当てられます。バス信号では、10進値が使用されます。FFTはシングルパスで計算されますので、複数のデジタルグラフ（デジタルグラフには、1つのデジタル信号、スカラまたはバスしか含む事ができません。）の選択は、このコマンドを実行する前に行って下さい。



Fast Fourier Transform ダイアログ

Primary Parameter Set（1次パラメータセット）

Primary Parameter Setのグループから設定したい入力パラメータセットをラジオボタンで選択して下さい。1次パラメータを選ぶと、Primary Parameter Value（1次パラメータ値）のところに入力できる項目が変わります。その右側のDerived parametersのグループには、1次パラメータセットの関係式と、入力した値を用いた計算値が表示されています。

n	FFTを行うポイント数
dt	時間解像度 （入力データ サンプリングステップ）
df	周波数解像度 （出力データ（スペクトル）サンプリングステップ）
t0	開始時間（入力データ）
tn	終了時間（入力データ）
fs	基準信号周波数
k	fs の位置

Primary Parameter Value（1次パラメータ値）

これらの値は独立していません。それぞれの値は、計算式で関連付けられています。サンプリングステップを指定し、1000ポイントでFFTを実行するとします。これであれば、周波数解像度（出力スペクトルの隣接した範囲の幅）は決定します。また、別のケースでは入力データに、時間t0とtn間、dfで指定できる出力スペクトルの間隔を使用するとします。この状況では、n（使用するポイント数）とdt（入力データサンプリングステップ）が決定されます。Fast FourierTransforme...のダイアログでは、パラメータ入力に関し、いくつかの方法が用意されており、計算で決まる値以外のものをユーザーが入力できます。

ポイント数には、2、3、5の倍数が使用できます。式としては2^l. 3^m.5^n となりl、m、nは正の整数である必要があります。これによりかなりの数値が使用できます。100、1000、5000、10000のような簡単な値はすべて使用できます。もし、使用できない値が入力された場合、アドバンストSimでは、プロンプトバーにもっとも近い値（小さい値と大きな値）が表示されます。

Data Padding（データの補足）

“Pad with last value if too few data point”のチェックボックスにチェックがあれば、アドバンストSimは入力データに前回のデータセットを繰り返し使用します。このオプションは入力データが、変換のためのポイント数より不足している場合に必要です。 “Pad with last value if too few data points”

Remove DC （DCの削除）

このチェックボックスは、FFTを実行する前に、DCコンポーネントの入力データを取り除く事ができます。これは、低周波スペクトルのコンポーネントの信号を解析する際に役立ちます。実際、BH4のようなウインドウ機能は、いくつか（5から7）の範囲（bins）に信号を広げます。もし、DCコンポーネントが存在すれば、信号の範囲とオーバーラップしているDCの範囲と一緒に、オーバーラップているスペクトルの一端を上にします。FFTを実行する前に、DCの値を入力データから削除する事で、このような望ましくない事態を避けられます。

Averaging（平均）

アドバンストSimでは、FFTを何回か実行し、それらの平均を求める事ができます。Average shifted transformのチェックをつけると、変換回数（number of transform…以後NAと表記します）とそれぞれの変換でシフトする総量（shift each transform…以後SHIFTと表記）を入力する箇所が表示されます。NA FFTの実行する事で結果にスペクトルが含まれます。それぞれのFFTは前の実行結果からSHIFTで設定した値で変化していき、NA FFTの平均が求められます。これは、ノイズの多い信号を解析するときに役立ちますが、結果を平均した数学的なノイズのような決まった信号では意味がありません。

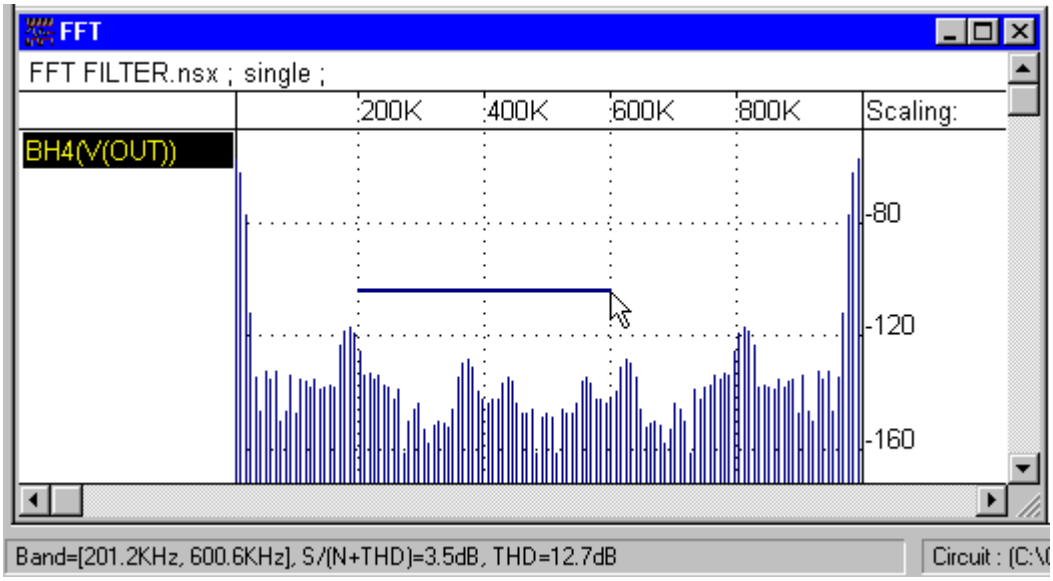
Windowing

これらのオプションは、違ったウインドウの機能を選ぶ事ができます。これは、波形に不連続な箇所があった場合に役立ちます。不連続な箇所があると、変換結果に間違った部分を導きます。ウインドウのオプションは、重みの機能で、データを終点で0になるようにします。ウインドウの各オプションについての説明は、このマニュアルの内容を超えていますので、ここでは省略します。

FFTの実行

ダイアログボックスの設定終了後、OKのボタンをクリックします。結果が新しいFFTの波形ウインドウに表示されます。もし、波形が正しく表示されない場合、FFTのウインドウをクリックし、画面を再描画するためにENDキーを押して下さい。もし、違った設定でFFTを何回か実行すると、新しい波形が前のものに追加されていきます。

SNRとTHDの計算



SNRとTHDを計算

これはFFTの中での機能します。FFTを実行し、SNRとTHDの計算を行う信号を選びます。

SNRとTHDを計算するには、

1. FFTウインドウで信号名をクリックします。
2. Tools - Compute SNR and THDメニューを選択します。
3. 波形ウインドウで必要とするバンドをマウスでクリックし、そのままドラッグします。
4. マウスを放した後、SNRとTHDが表示されます。

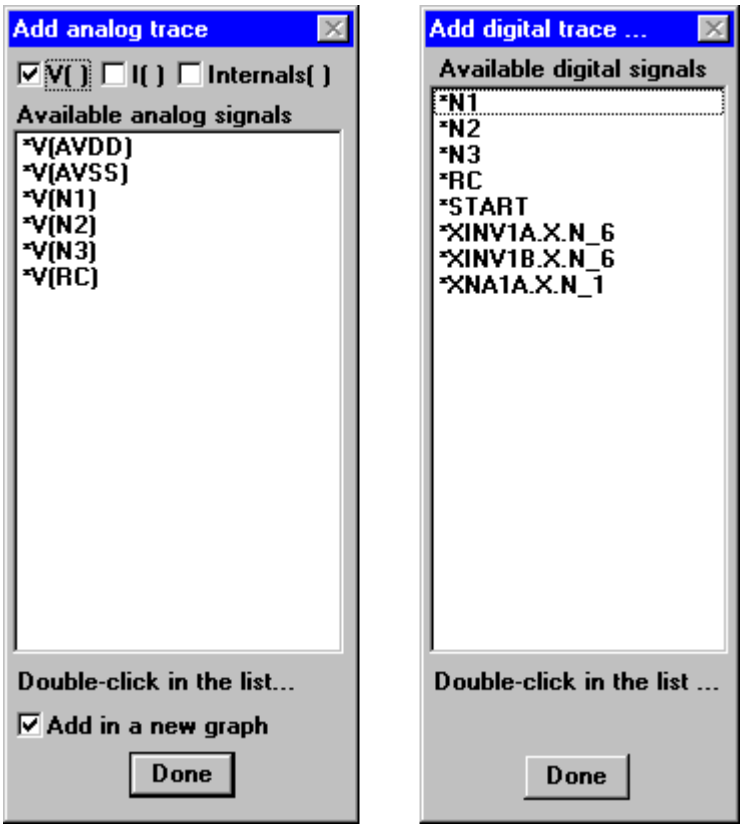
ウェーブフォームウインドウの設定

表示波形の選択

回路図を作成し、どの種類のシミュレーションを行うかの設定が済めば、シミュレーションを実行します。回路を作成し、最初にシミュレーションを実行すれば、次のようなメッセージが表示されます：

> “No signals will be displayed during simulation. Would you like to set up the simulation screen now?” （「シミュレーションの間、どの信号も表示されません。シミュレーション画面の設定をここで行いますか？」）

はい（Y）をクリックすると、次のようなダイアログボックスが一つ、あるいは両方表示されます：

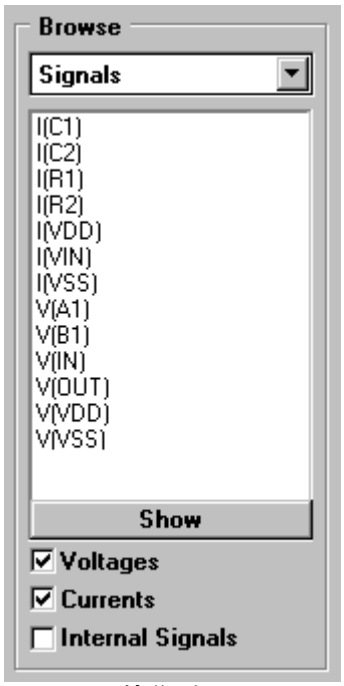


最初にシミュレーションを実行すると、Add analog trace とAdd digital traceのダイアログボックスが表示されます。

波形表示したい信号をダブルクリックして下さい。ここには電流とインターナルアナログ信号もあります。もし、ここで選択を行わなくても後から簡単に追加できます。

Doneのボタンを押すと、波形が表示されます。もし、実際の波形が表示されない場合、或いはスケールがあってない場合は、ドキュメントをアクティブにするために波形ウインドウをクリックし、View - Fit Windowsを実行して下さい。

波形がアクティブになっていれば、パネルから信号を追加する事ができます。（もしパネルが表示されていなければ、メニューから View - Panelを実行して下さい。）信号名をクリックするだけで、波形ウインドウにその信号が表示されます。クリックした信号だけが波形ウインドウに表示されてます。次の「ウェーブフォームウインドウでの作業」のところで表示方法の変更の仕方等が書かれています。パネルの下にあるチェックボックスで信号リストが切り替えられます。



波形パネル

シミュレーターでどの信号が保持されるか？

シミュレーションでどの信号が保持されるかは、（それらはもちろん表示する事が可能です。）アナログ信号では .PRINTと .PRINTALL、デジタル信号は .LPRINTと .LPRINTALLというディレクティブを使う事で指定する事ができます。アドバンストSimでは、自動的に作成されるパターンファイルに .PRINTALLと .LPRINTALLを書き込みます。もし、なんらかの理由で特定の信号だけを表示させたい場合は、パターンファイル（*circuit*.PAT）にディレクティブを書き込んでやる必要が有ります。パターンファイルについての情報は*アドバンストSimファイル*の章を、.PRINTディレクティブについてはオンラインヘルプの*ディレクティブ*のトピックを参照して下さい。

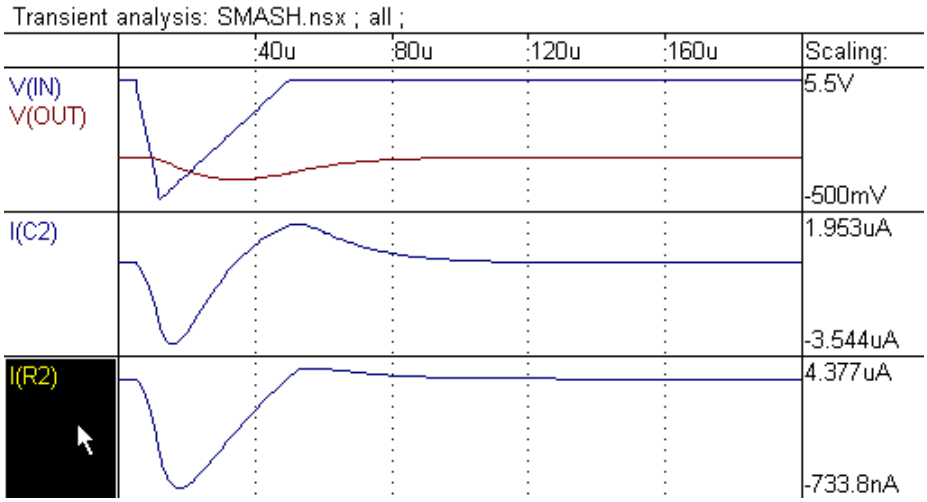
➔ .PRINTALLと .LPRINTALLのディレクティブを使用すると大きなファイルが作成される可能性があります。もし、これで問題が発生する場合は、信号を特定するために .PRINT、.LPRINTのディレクティブを使用し、パターンファイルを修正する必要が有ります。

どの信号が表示されるか？

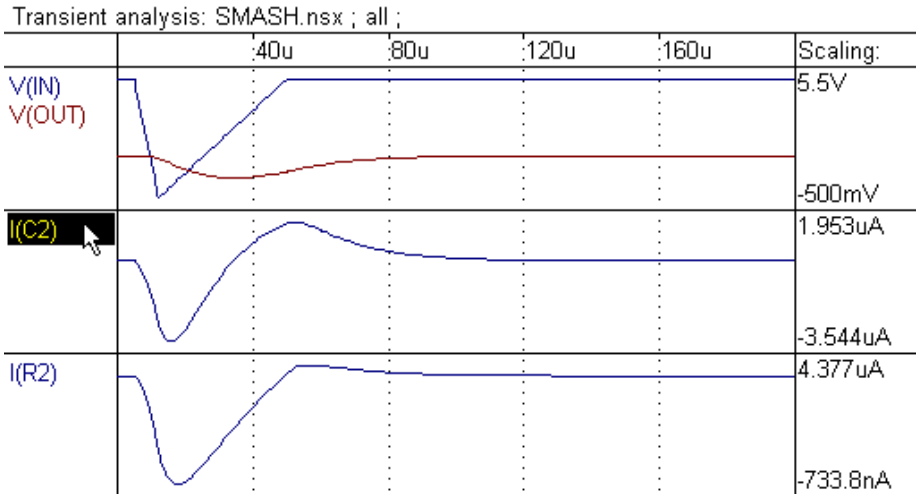
波形ウインドウで表示される信号は、パターンファイルに .TRACEと .LTRACEというディレクティブを記入する事で指定できます。これらのディレクティブについての詳細は、オンラインヘルプの*ディレクティブ*のトピックを参照して下さい。

ウェーブフォームウインドウでの操作

アドバンストSimには波形を解析するのに便利なツールがいくつかあります。次に、図を用いてどのようにして波形表示を扱うかを説明します。



（R2）の所をダブルクリックするとこの波形だけが画面表示されます。もう一度、ダブルクリックすると元の状態に戻ります。クリックし、ドラッグアンドドロップで表示の順序を入れ替える事ができます。シフトキーを押しながら、クリックすると複数の波形を選択する事ができます。デリートキ ーを押すと波形表示は消えます。



波形のタイトルをクリックし、I（C2）をドラッグアンドドロップで移動するとその波形が他の波形の領域に表示されます。この図では、既にV（OUT）がそのようになっています。

波形の拡大・縮小表示

次のツールはズームレベルの変更に使用します：

ズーム 水平方向 ― このボタンをクリックした後、カーソルがクロスになります。ズームさせたい波形の部分でクリックし、水平方向にドラッグを行って下さい。このコマンドを終了するには、マウスの右ボタンをクリックします。

ズーム 垂直方向 ― このボタンをクリックした後、カーソルがクロスになります。ズームさせたい波形の部分でクリックし、垂直方向にドラッグを行って下さい。このコマンドを終了するには、マウスの右ボタンをクリックします。

波形フィット ― ズームイン等を行った後、このボタンをクリックすると元の状態に戻ります。

ズーム モア ― このボタンを押すと、波形表示が大きくなります。

ズーム レス ― このボタンを押すと、波形表示が小さくなります。

波形の測定

波形をクリックすると、ウインドウの一番上のところにクリックした位置についての情報が表示されます。また、デルタについては前にクリックしたポイントからの値が表示されます。

またメニューのTools - Measureを実行すると、2点間の距離を測定できます。クリックし、マウスをドラッグさせ始点と終点を指示して下さい。位置、デルタ、スロープがウインドウの上部に表示されます。マウスをドラッグするごとに表示が更新されますので注意して下さい。マウスを放すと最終的なデルタとスロープが表示され、2つの十字のマークが画面に表示されます。画面の再描画を行うと、この表示は消えます。（ENDキーを押すか、メニューのView - Refreshを実行します。）

シミュレーションポイントの表示

波形ウインドウで実際のシミュレーションポイントを表示する事ができます。信号をクリックし、Tools - Show Pointsを実行して下さい。すべてのポイントが表示されます。このメニューは切り替えになっており、もう一度信号を選びこのコマンドを実行すると、ポイント表示は消えます。過渡解析では、すべてのポイントがセーブされているわけではありません。シミュレーションの表示ステップが非常に小さい場合でなければ、表示ステップノーマルの時間ステップより小さいか同じであれば、実際のシミュレーションポイントが見れます。波形がマークされている場合は、表示が遅くなります。

線形とログ表示の切り替え

View - Log X Scaleを実行すると、水平方向のスケールがリニアとログの切り替わります。これを行うには、X軸がすべて正の値である必要が有ります。

波形をX軸として使用

Edit - Use as X Axisメニューを選択すると、アドバンストSimは新しい水平軸（信号名を最初にクリックしてください）と選択された信号の波形を再描画します。前の表示に戻すには同じメニューを選択します。

新規グラフでの波形作成

新規のグラフでのEdit - Drawは、複数の波形を1つに重ねたり、再度複数に分ける場合に有用です。このオプションを選択する前に、信号名を選択する必要があります。

波形の削除

選択した波形を削除します。DELETEキーでも削除できます。

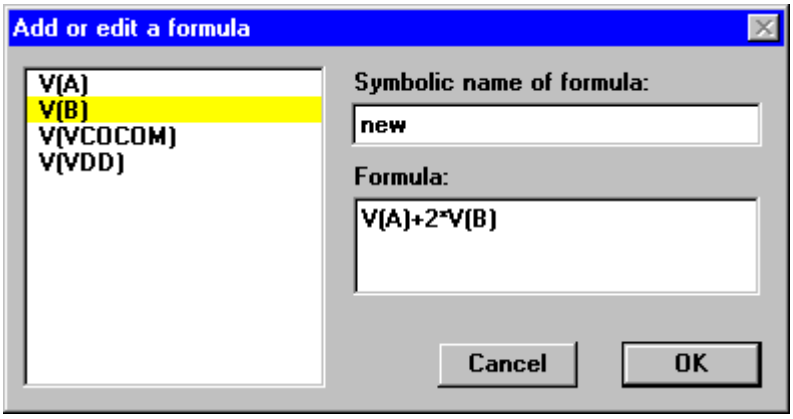
➔ これらはアンドゥを使用することができません。波形を復活させるには、もう一度シミュレーションを行う必要があります。

波形の追加

Editメニューには、現在の波形ウインドウに波形を表示するオプションの番号があります。表示されるダイアログボックスは追加する波形の形式に依存します。

アナログ式の追加

波形ウインドウで、式を追加するにはこのオプションを使用します。Add or Edit a formula ダイアログボックスに、式に使用できる信号の一覧が表示されます。 .PRINTディレクティブにリストされている信号を保存することが可能です。式とexpressionに名前を入力する必要があります。すでに存在する式の名前を入力した場合には、自動的に編集が取り消されます。式の中の信号を使用するには、名前をタイプするか、リスト内の名前をダブルクリックします。



式の追加・編集のダイアログ

式には、どの演算子（+ - * /）、関数（sin（），cos（） log（），etc.）、信号（電圧と電流）でも利用する事ができます。

含まれている内部の変数（IN（MOS_OUT.GDS）の様な多く）を式に入れる場合には、パターンファイルの .TRACEディレクティブに式を入力しなければなりません。このダイアログボックスを使用することはできません。

➔ サポートされている関数についての完全な記述は .TRACEディレクティブを参照して下さい。

➔ 式を追加した後、File - Saveを選択してパターンファイル内の式を保存します。

クロスプローブ

アドバンストSimは、EDA/クライアント内でオープンされている他のドキュメントとのクロスプローブをサポートしています。スケマティックから波形にクロスプローブを行うには、最初にスケマティックのドキュメントをアクティブにします。ツールバーからクロスプローブボタン  をクリックし、クロスプローブを行うコンポーネント、またはネットラベルをクリックします。

コンポーネントのボディーをクリックすると、波形の該当するコンポーネントにクロスプローブされます。ネットラベルをクリックすると電圧ノードにクロスプローブされます。ネットリスト（*circuit*.NSX）などのシミュレーション用のテキストファイルでもクロスプローブを行うことができます。

ドローイングとテキストツール

アドバンストSimはシンプルなドローイングとテキストツールを含んでおり、四角形、矢印やテキストはシミュレーションウインドウに追加することができます。それらによってシミュレーション結果へのコメントなどを簡単にできます。

それらのドローイングオブジェクトは、連続したシミュレーションの間で削除することはできません。現在のドローイングオブジェクトは、ファイルに保存することができず、Close AllまたはQuitした際にそれらは失われます。

ドローイングまたはテキストオブジェクトの追加

ドローイングまたはテキストオブジェクトをウインドウに追加するには、Toolsメニューからオブジェクトのタイプを選択します。オブジェクトのアウトラインを描くにはマウスを使用します。テキストオブジェクトダイアログボックスが表示されます。テキストを入力し、OKをクリックします。追加を行うと、オブジェクトはウインドウに入ります。それぞれのシミュレーションウインドウはオブジェクトのセットを持つことができます。

ドローイングまたはテキストオブジェクトの固定

デフォルトでは、ドローイングまたはテキストオブジェクトを追加する際に、ズームや全体を表示すした際と同じ<<pixel>>の位置のままになります。スクリーン上の位置を変更することはできません。ドローイングまたはテキストオブジェクトを物理的な位置に固定するには、CTRLキーを押して追加します。その際に、スクリーンを最大のズームレベルにすることができます。

ドローイングまたはテキストオブジェクトの修正

ドローイングまたはテキストオブジェクトは、シミュレーションウインドウの中でサイズや位置を修正することができます。ドローイングオブジェクト（ライン、四角形、または矢印）のサイズを変更するには、マウスを1回クリックしてオブジェクトをセレクトします。コーナーハンドルを使用してサイズを変更します。オブジェクトを移動するには、ドラックアンドドロップで行います。オブジェクトは親ウインドウの波形のエリア内のみ移動することができます。

テキストオブジェクトの修正

テキストで構成されるオブジェクトを変更するには、テキストをダブルクリックします。ダイアログボックスが表示され、テキストを変更することができます。OKボタンをクリックするとテキストは新しい内容に更新されます。

ドローイングまたはテキストオブジェクトの削除

ドローイングまたはテキストオブジェクトを削除するには、波形エリアの外側をドラッグアンドドロップします。

グラフィックフォントの変更

Tools - Change Fontはウインドウで使用しているフォントを変更します。これはスクリーンの解像度に合わせてサイズや最適な印刷結果を得るために有用です。標準のフォント選択ダイアログボックスでは、フォント名、スタイルやサイズを選択します。選択されたフォントは*advsim*.INIファイルの[Fonts]セクションに保存されます。詳細は*アドバンストSimファイル*の章を参照して下さい。

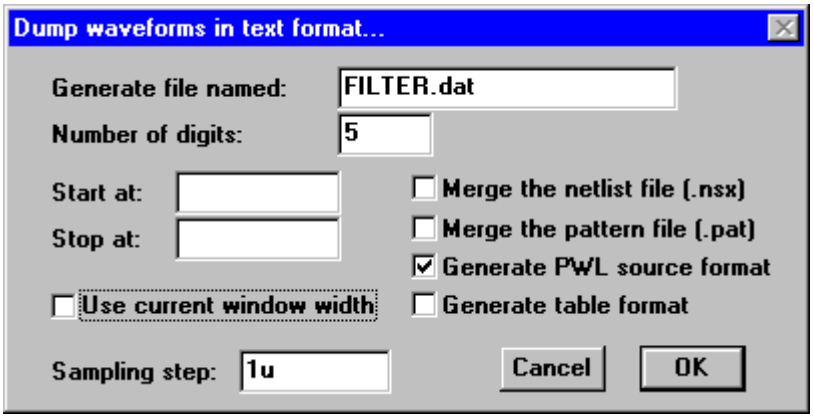
波形の印刷

印刷のダイアログボックスではアクティブなウインドウを印刷します。印刷オプションの番号は、*advsim*.INIファイルの設定で修正できます。詳細は*アドバンストSimファイル*の章を参照して下さい。

レポート

アスキーファイルの生成

Reports - Generate ASCII fileを選択して、アクティブな波形ウインドウ（transient、small、signalなど）のアナログ波形のテキストフォーマットのファイル（circuit.DAT）を作成します。



波形ダンプのダイアログ

Generated file named（作成するファイル名）

出力ファイルの名前を指定するために使用します。ファイルは常に*circuit*.NSXファイルと同じディレクトリに生成されます。

Number of digits（桁数）

10進数の小数点以下を何桁まで出力するかを指定します。

Merge netlist/pattern file（ネットリスト/パターンファイルのマージ）

出力とともに、ネットリストと（または）パターンファイルに10進数の番号を付加するかを選択することができます。

Generate PWL format（PWLフォーマットの作成）

このオプションはPWLフォーマットの出力ファイルを生成する場合に使用します。生成したファイルを他のシミュレーションで再利用することができます。このオプションをセレクトすると、それぞれのトレースはPWL文に変換され、パターンファイルに含めることができます。

Generate table format（テーブルフォーマットの作成）

このオプションがチェックされている場合には、tableスタイルの出力ファイルが生成されます。出力ファイルのフォーマットは、SPICE出力ファイルの .PRINTセクションと同じです。

Use current window width（現在のウインドウ幅を使用）

シミュレーションウインドウの表示幅を使用したい場合に、チェックして下さい。チェックをしない場合には、Start atとStop atフィールドが表示されます。これらのフィールドは、出力ファイルの生成の境界線を明確に示すことに使用されます。

Sampling step（サンプリングステップ）

このオプションは、次の値を記述するステップを指定します。これはX軸が時間軸の場合にのみ必要です。

➔ デジタル波形がウインドウに表示された場合、OKをクリックするとそれらは 無視されます。アナログ波形のみテキストフォーマットのファイルに落とすことができます。

変換

Reports - Convertでを実行すると、*circuit*.HISを*circuit*.H2P ファイルに変換できます。このファイルには、出力波形ファイル*circuit*.HISから*circuit*.PAT ファイルで指定されている波

形が含まれています。この機能により論理シミュレーションの結果を、他のシミュレーションの入力パターンとして使用できます。

回路プリミティブとスティミラス

アナログプリミティブ

構文

アドバンストSimでは、SPICEの構文を使用して回路内のアナログデバイスを記述します。アナログエレメントとして解釈されるデバイスは、SPICE構文の中にある必要があります。アドバンストSim は、SPICE構文のスイッチングインジケーター >>> SPICEで、これに続く構文がSPICE構文であることを判断します。デザイン内にアナログ回路しか存在しない場合、構文を指定する必要はありません。デフォルトの構文としてSPICEが指定されます。SPICEとVerilogを使用してアナログ／デジタル混在のシミュレーションを行なう場合、>>> インジケーターを使用して、SPICE構文を切替える必要があります。詳細については、*階層記述*の章の混在スタイル .SUBCKTSを参照して下さい。

デバイスタイプは、通常その名前の最初の文字で識別します。この最初の文字を‘ プリフィックス ’と呼びます。

それぞれのエレメントタイプに対して説明がされます。この説明では以下のような決まりがあります:

- 大文字のアイテムは、キーワードです。これらは正確に記述されなければなりません。
- 小文字のアイテムは、通常プレイスホルダーです。これらは、ユーザによって与えられる実際の文字列と置き換えられる必要があります。（例えば、ノード名や数値）
- ［スクエアブラケット］で囲まれた物は、オプションです。オプションを使用する場合、ブラケットは取除いて下さい。

抵抗

説明

```
Rname  n1  n2  val [TC=tc1[,tc2]]
```

Rname は抵抗の名前です。名前はR で始まらなければなりません。抵抗は、ノードn1とn2の間に接続されます。抵抗値（Ω）はvalで、ゼロ以外の値です。一般的には正の値になります。

温度係数

オプションのパラメータtc1とtc2により、抵抗の温度係数が定義されます。抵抗の実際の値は、tc1とtc2を用いて以下の式で表わされます：

$$R(T) = val \cdot (1+tc1 \cdot (T-TNOM)+tc2 \cdot (T-TNOM)^2)$$

ここで、TNOM は公称温度 （27°C）でTはシミュレーション温度です。（.TEMPディレクティブを参照して下さい。）

ノイズ

抵抗により、温度ノイズが発生します。抵抗のノイズは以下の式で表わされます：

$$i^2 = 4.k.T/R$$

抵抗を通過する電流

抵抗を流れる電流は、.TRACEディレクティブまたは .PRINTディレクティブで構文I(Rname)で取得することができます。電流は、ノードn1からデバイスを通過してノードn2へ流れる場合、正の値になります。

例：

```
* in circuit.NSX:
RL OUT 0 100K
RT VDD COLL 1.5K TC=0.0034,0.00056
```

```
* in circuit.PAT
.TRACE TRAN I(RT) min=-1mA max=1mA
```

➔ .TRACE ディレクティブも参照して下さい。

コンデンサ

説明

線形コンデンサ：

```
Cname n1 n2 val
```

非線形コンデンサ：

```
Cname n1 n2  POLY val vc1 vc2 vc3
```

Cname はコンデンサの名前です。名前はCで始まらなければなりません。ノードn1とn2の間に接続されます。Valはコンデンサの容量（F）で、ゼロでない正の値である必要があります。

最初の構文は、線形のコンデンサを表わすために使用され、2番目の構文は非線形コンデンサを表わすために使用されます。非線形コンデンサのコンデンサの容量は、以下の式で計算されます：

$$C(v) = val + vc1 \cdot v + vc2 \cdot v^2 + vc3 \cdot v^3$$
$$\text{where } v = V(n1)-V(n2)$$

コンデンサを通過する電流

コンデンサを通過する電流は、.TRACEディレクティブまたは .PRINTディレクティブで構文I(Cname)で取得することができます。電流は、ノードn1からデバイスを通過してノードn2へ流れる場合、正の値になります。

例：

```
* in circuit.NSX
Cload OUT 0 10P
Ccoup IN OUT POLY 50F 0.0021 1E-3 7.5U

*in circuit.PAT
.PRINT I(Cload) V(OUT)
```

➔ .TRACE ディレクティブも参照して下さい。

インダクタ

説明

```
Lname n1 n2 val
```

Lname はインダクタの名前で、L で始まらなければなりません。インダクタLnameは、ノードn1とn2の間に接続されます。

valは、インダクタのインダクタンス値（H）で、ゼロでない正の値である必要があります。

インダクタを通過する電流

インダクタを通過する電流は、.TRACEディレクティブまたは .PRINTディレクティブで構文I(Lname)で取得することができます。電流は、ノードn1からデバイスを通過してノードn2へ流れる場合、正の値になります。

例：

```
* in circuit.NSX
LOSC OUT 34 10N

* in circuit.PAT
.TRACE TRAN I(LOSC)
```

➔ .TRACE ディレクティブも参照して下さい。

結合インダクタ

説明

```
Kname L1name L2name k
```

Kname は、トランスを表わす名前で、K で始まらなければなりません。kは、インダクタL1nameとL2nameとの間のカップリング係数を定義します。これらのインダクタは、回路内に存在していなければなりません。

kはカップリング係数です。この値は、1 より小さい絶対値でなければなりません。インダクタL1nameとL2nameの電流を表わす式は、以下の式から導出されます：

```
v(L1name) = L1value*i'(L1name) + mval*i'(L2name)
v(L2name) = mval*i'(L1name) + L2value*i'(L2name)
```

ここで mval = k/sqrt(L1*L2)

例：

```
* in circuit.NSX
L1     IN1    IN2         10N

L2     OUT1   OUT2        20N

KL1L2  L1     L2    0.92
```

MOSトランジスタ

説明

```
Mname  nd  ng  ns  nb  model  W=w  L=l [AD=ad]  [AS=as]
+ [PD=pd] [PS=ps] [NRD=nrd] [NRS=nrs] [M=m]
```

➔ モデル毎の説明については、*ライブラリとデバイスモデル*の章を参照して下さい。

Mnameはデバイス名で、M で始まらなければなりません。

ノード nd、ng、ns、nb はそれぞれMOSトランジスタのドレイン、ゲート、ソース、バルク端子を表わします。

Modelはこのトランジスタに使用されるトランジスタのモデル名です。modelの最初の文字は、数値ではなく文字である必要があります。このモデルは、.MODELステートメントで参照されます。.MODELステートメントがパターンファイル（*circuit*.PAT）に無い場合、アドバンストSim によりファイル名 *model*.MDLか、拡張子.LIBのファイルを検索します。（*ライブラリとデバイスモデル*の章を参照。）

wとlはトランジスタの幅と長さです。デフォルト値はありませんので、必ず記述して下さい。

ad、as、pd、ps、nrd、nrsは、オプションの値です。それぞれ、ドレイン面積、ソース面積、ドレイン周囲長、ソース周囲長、ドレイン側の拡散領域の数、ソース側の拡散領域の数を表わします。これら6個のパラメータのデフォルト値はゼロです。ad、as、pd、psは、トランジスタモデルと関連付けられるCJとCJSWパラメータの掛け算の係数で、ドレイン－バルク間とソース－バルク間のキャパシタンスと電流が計算されます。

パラメータmは、オプションの掛け算の係数です。指定されない場合、デフォルト値は1です。このパラメータによって、m個のトランジスタが並列に接続されることを表わします。この場合、幅やオーバーラップキャパシタンス、ジャンクションキャパシタンスや電流はm倍され、一連の抵抗値はmで割られます。

単位

レベル5（EPFLモデル）を除いて、単位には標準の単位を用います。レベル5では、LUNIT モデルパラメータを標準の単位系または“マイクロ－エレクトロニクス”単位系の変わりに使用して下さい。

➔ *Libraries and Device Models* の章を参照して下さい。

レベル0, 1, 2, 3 及び4の場合:

w, l, ad, as, pd 及び ps の単位はそれぞれメートル、メートル、メートル^2、メートル^2、メートル、メートルです。Nrd, nrs 及び m には単位はありません。

例：

```
.MODEL NTYP NMOS LEVEL=3 ...
* and:
M2 XOUT XIN VDD VDD NTYP W=20U L=2U AD=50P PD=150U NRD=10 NRS=20
```

レベル5の場合:

LUNIT モデルパラメータが1.0の場合、w, l, ad, as, pd 及び psの単位はそれぞれメートル、メートル、メートル^2、メートル^2、メートル、メートルです。

LUNIT モデルパラメータが1e-6の場合w, l, ad, as, pd 及びps の単位はそれぞれμメートル、μメートル、μメートル^2、μメートル^2、μメートル、μメートルです。

LUNITに関係なく、nrd と nrs、mは単位がありません。

例：

```
.MODEL NTYP NMOS LEVEL=5 LUNIT=1 ...
* and:
M2 XOUT XIN VDD VDD NTYP W=20U L=2U AD=50P PD=150U NRD=10 NRS=20
```

例：

```
.MODEL NTYP NMOS LEVEL=5 LUNIT=1e-6 ...
* and:
M2 XOUT XIN VDD VDD NTYP W=20 L=2 AD=50 PD=150
+ NRD=10 NRS=20
```

端子を流れる電流

.TRACEディレクティブまたは .PRINTディレクティブでトランジスタの4つの端子を流れる電流値を取得できます。端子部'T'の電流を表わす構文はIT(Mname)です。

例：

```
* in circuit.PAT
.TRACE TRAN ID(M42) IG(M67)
* 上記の記述により、MOS-M42のドレインの電流と
* MOS-M67のゲートの電流がプロットされます。


* in circuit.NSX
M1 XOUT XIN 0 0 NTYP W=10U L=2U AD=200F PD=50P
M2 XOUT XIN VDD VDD PTYP W=20U L=2U AD=350F PD=150P NRD=10 NRS=20
```

拡散領域の面積と周囲長のデフォルト値

ad, as, pd , ps が指定されておらず、モデルに関連付けられたLDIFパラメータがゼロの場合、拡散領域のキャパシタンスはゼロです。拡散領域のキャパシタンスは以下の式で定義されます:

```
AS = AD = W • LDIF and PS = PD = 2 • (W + LDIF)
```

上式のW の値はネットリストで定義されます。

寄生抵抗

Nrdやnrsは、RSH モデルパラメータの掛け算係数として使用され、ドレインとソース抵抗が計算されます。RSHと一緒にNRD/NRSやRD/RS モデルパラメータを利用して直列抵抗を追加することは、回路のノードの数が増えるということにお気付きだと思います。

ドレインまたはソースの寄生抵抗がゼロでない場合、内部ノードが追加されます。ソース抵抗の場合、S$Mname という名前のノードが作成され、ドレイン抵抗の場合、D$Mname という名前のノードが作成されます。

➔ 寄生抵抗を持つ各MOSトランジスタインスタンスは、最大2個の内部ノードが作成されることに注意して下さい。これにより、行列の大きさが急速に大きくなります。

このセクションでは、例としてドレイン抵抗を使用して直列の寄生抵抗の計算について説明します。パラメータ名を変更すれば、ソース抵抗の場合も同様に計算できます。

ケース1

この場合、RD パラメータは .MODELステートメントに関連付けられて指定されます。

ドレイン抵抗は：

```
rd = RDC + RD
```

ケース2

この場合：

- **RDパラメータは、.MODELステートメントに関連付けられて指定されません。**
- **NRDは、MOSインスタンスパラメータで与えられます。**

ドレイン抵抗は：

```
rd = RDC + RSH * NRD
```

➔ **RSH が与えられなかったり .MODELステートメントでゼロ（RSHのデフォルト値はゼロ）の場合、デバイス内のNRDの記述は、無効になります。**

ケース3

この場合：

- **RDパラメータは、.MODELステートメントに関連付けられて指定されません。**
- **NRDは、MOSインスタンスパラメータで与えられません。**
- **LDIF パラメータが、.MODELステートメントに関連付けられて指定されます。**

ドレイン抵抗は：

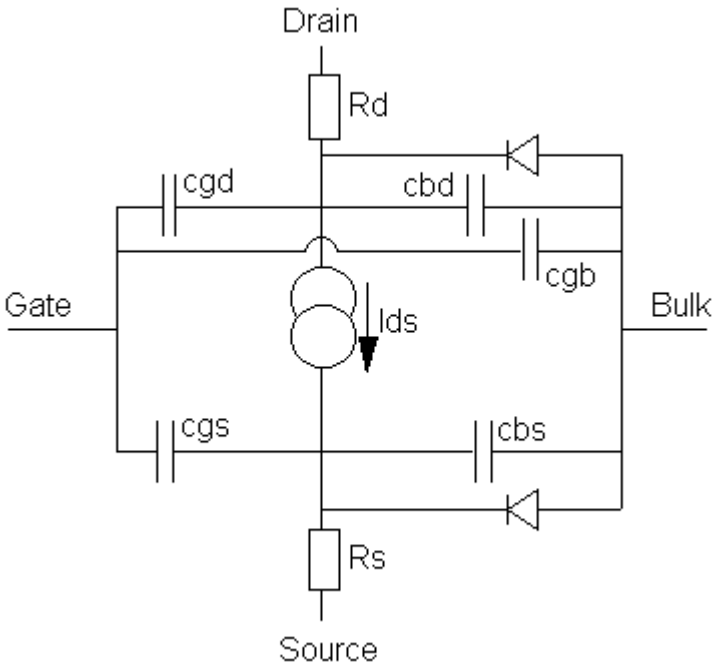
```
rd = RDC + RSH * LDIF / W
```

➔ **式の中で使用されるW は、記述された値（ネットリストで入力された値）で、実際の値ではありません。**

内部変数の取得

MOSトランジスタ（gds、cgsなど）の内部変数は、.TRACEディレクティブでアクセスできます。

➔ **.TRACEディレクティブ（オンラインヘルプのディレクティブトピック）と *ライブラリとデバイスモデル*の章を参照して下さい。**



MOS トランジスタ (levels 1, 2, 3)

バイポーラトランジスタ

説明

```
Qname nc nb ne [ns] model [area]
```

Qname はトランジスタに使用されるモデルの名前です。Qで始まらなければなりません。

Nc, nb, ne及びnsはそれぞれバイポーラ端子のコレクタ、ベース、エミッタ、サブストレート端子を表します。

サブストレートノードが省略された場合、自動的にグランドに接続されます。ns ノード名が文字で始まる場合、アドバンストSimがサブストレート名に続くモデル名とノード名とを区別できるように括弧をつけて下さい。

Model はこのトランジスタに使用されるモデルの名前です。modelの最初の文字は数値ではなく文字である必要があります。このモデルは、.MODELステートメントで参照することができます。.MODELステートメントがパターンファイル（*circuit*.PAT）内に存在しない場合、アドバンストSimは、model.MDLまたは、拡張子.LIBのファイルを検索します。

➔ *ライブラリとデバイスモデル* の章を参照して下さい。

関連付けられたモデルにCJS値が設定されている場合、ジャンクションキャパシタンスは、コレクタとサブストレートとの間でモデル化されます。CJS 値がゼロの場合、サブストレート端子は影響されません。

Areaは、オプションの面積係数で、デフォルト値は1です。areaパラメータは、電流やキャパシタンスに対して掛け算されたり、抵抗に対しては割り算で効いてきます。また、Area はモンテカルロ解析に使用されます。

端子を流れる電流

トランジスタの3つの端子（ベース、コレクタ、エミッタ）を流れる電流は、.TRACEディレクティブまたは .PRINTディレクティブで取得することができます。ターミナル'T' を流れる電流を表わす構文は IT(Mname)です。

例：

```
.TRACE TRAN IC(Qload) IB(Qin)
//上記のディレクティブにより、トランジスタQloadのコレクタを流れる
//電流とトランジスタQinののベースを流れる電流がプロットされます。
```

関連したモデルのベース抵抗またはエミッタ抵抗、コレクタ抵抗がゼロでない場合、内部ノードが追加されます。ベース抵抗により、B$Qnameという名前のノードが作成され、エミッタ抵抗により、E$Qnameというノードが作成されます。さらに、コレクタ抵抗により、C$Qnameという名前のノードが作成されます。

➔ それぞれのバイポーラトランジスタで最大3本のノードが作成されることに注意して下さい。ノードの増加により、行列の大きさが急激に大きくなります。

例：

```
Q1234 VDD INB EM 100 Q2N2222  4.5
* Q1234のサブストレートはノード100
QBIP1 VCC B E [SUBTR] QBC109
* QBIP1のサブストレートはノードSUBTR
QBIP2 VCC B E QBC109 3.4
* QBIP2のサブストレートはノード0（グランド）
```

ダイオード

説明

```
Dname na nc model [area]
```

Dnameはダイオードに使用されるモデル名で、D で始まる必要があります。

Naとncはそれぞれダイオードのアノードとカソード端子を表わします。

Modelは、このダイオードに使用されるダイオードモデルの名前です。model の最初の文字は数値以外の文字である必要があります。

このモデルは、.MODELステートメントにより参照されます。.MODELステートメントがパターンファイル（*circuit*.PAT）の中にない場合、アドバンストSimにより、ファイル名

model.MDLか、拡張子.LIBのファイルが検索されます。（*ライブラリとデバイスモデル*の章を参照。）

Area はオプションの面積係数で、デフォルト値は1です。Area はモンテカルロ解析に使用されます。

例：

```
Dclam OUT VPOS DrefX12
D2 VNEG OUT Drefx14 2.7
```

ダイオードを流れる電流

.TRACEディレクティブや .PRINTディレクティブでダイオードを流れる電流を取得できます。ダイオードを流れる電流を表わす構文は、I(Dname)です。

例：

```
.TRACE TRAN I(D1)
*上記のディレクティブにより、ダイオードD1を流れる電流がプロット
 されます。
```

ジャンクションFET

説明

```
Jname nd ng ns model [area]
```

JnameはJFETに使用されるモデル名で、Jで始まる必要があります。

Nd、ng、nsはそれぞれJFETのドレイン端子、ゲート端子、ソース端子を表わします。

modelは、このJFETに使用するJFETのモデル名です。modelの最初の文字は、数値以外の文字である必要があります。

このモデルは、.MODELステートメントにより参照されます。.MODELステートメントがパターンファイル（*circuit*.PAT）の中にない場合、アドバンストSimにより、ファイル名*model*.MDLか、拡張子.LIBのファイルが検索されます。（*ライブラリとデバイスモデル*の章を参照。）

Areaはオプションの面積係数で、デフォルト値は１です。Areaはモンテカルロ解析に使用されます。

例:

```
J1 DR G1 GND JMOD 3
```

JFETの端子を流れる電流

.TRACE ディレクティブや.PRINTディレクティブで各端子を流れる電流を取得できます。FETの各端子を流れる電流を表わす構文は、ID(Jname)、IG(Jname)、IS(Jname)です。

例:

```
.TRACE TRAN ID(J1)
* 上記ディレクティブにより、J1という名前のJFETのドレイン端子を流れる電流がプロットされます。
```

電圧制御電流ソース

説明

```
Gname out+ out- vctrl+ vctrl- val
```

または

```
Gname out+ out-
+ POLY(n) vctrl1+ vctrl1-  ... vctrln+ vctrln-
+ p0 p1 p2  ... pn
```

Gnameは電流ソースに使用されるインスタンス名で、Gで始まる必要があります。

out+とout-はそれぞれ電流ソースの正の出力ノードと負の出力ノードを表わします。

最初の形式は、単純な電圧で制御される電流ソースです。例えば、変換コンダクタンスの値valを以下のように指定することができます。

```
Iout = val•(V(vctrl+)-V(vctrl-))
```

ここで、Ioutは電流ソースを通過してout+からout-へ流れる電流です。

２番目の形式は、複数の入力で制御される電流ソースを表わします。それぞれのペア(vctrli+, vctrli-)は、pi係数と関連付けられます。p0は定数項です。

```
Iout = p0
+p1•(V(vctrl1+)-V(vctrl1-))
...
+pn•(V(vctrln+)-V(vctrln-))
```

➔ nは10までです。

➔ アドバンストSimのE, F, G, H デバイスは線形のデバイスです。すなわち、それらはSPICEの中で完全多項式で表現されません。非線形の関係が必要な場合、数式定義のソースを使用して下さい。

例:

```
Ga OUTP OUTM IN1 IN2 5M
Glim OUT 0 POLY(2) (IN1,VDD) (IN2,VSS)
+ 0.0 0.001 -0.001
```

➔ 上記の例では括弧が使用されています。わかりやすく記述するために括弧を使用することができます。

電圧制御電圧ソース

説明

```
Ename out+ out- vctrl+ vctrl- val
```

または

```
Ename out+ out- POLY(n) vctrl1+ vctrl1-  ... vctrln+ vctrln-
+ p0 p1 p2  ... pn
```

Enameは電圧ソースに使用されるインスタンス名で、Eで始まる必要があります。

out+とout-はそれぞれ電圧ソースの正の出力ノードと負の出力ノードを表わします。

最初の形式は、単純な電圧で制御される電圧ソースです。例えば、ゲインの値valを以下のように指定することができます。

```
Vout = val•(V(vctrl+)-V(vctrl-))
```

ここで、Voutはout+とout-との間の電圧です。

２番目の形式は、複数の入力で制御される電圧ソースを表わします。それぞれのペア(vctrli+, vctrli-)は、pi係数と関連付けられます。p0は定数項です。

```
Vout = p0
+ p1•(V(vctrl1+)-V(vctrl1-))
+ ...
+ pn•(V(vctrln+)-V(vctrln-))
```

➔ nは10までです。

➔ アドバンストSimのE, F, G, Hデバイスは線形のデバイスです。すなわち、それらはSPICEの中で完全多項式で表現されません。非線形の関係が必要な場合、数式定義のソースを使用して下さい。

例:

```
E_AOP POUT NOUT PIN NIN 10000
```

電流制御電流ソース

説明

```
Fname out+ out- vsource val
```

または

```
Fname out+ out- POLY(n) vsource1 ... vsourcen p0 p1 p2  ... pn
```

Fnameは電流ソースに使用されるインスタンス名で、Fで始まる必要があります。

out+とout-はそれぞれ電流ソースの正の出力ノードと負の出力ノードを表わします。

最初の形式は、単純な電流で制御される電流ソースです。例えば、ゲインの値valを以下のように指定することができます。

```
Iout = val•I(vsource)
```

ここで、Ioutは電流ソースを通過してout+からout-へ流れる電流で、I(vsource)は、vsourceという名前の電圧ソースを通る電流です。

２番目の形式は、複数の入力で制御される電流ソースを表わします。それぞれのvsourceiは、それぞれpi係数と関連付けられます。p0は定数項です。

```
Iout = p0
+ p1•I(vsource1)
+ ...
+ pn•I(vsourcen)
```

➔ n は10までです。

➔ アドバンストSimのE, F, G, H デバイスは線形のデバイスです。すなわち、それらはSPICEの中で完全多項式で表現されません。非線形の関係が必要な場合、数式定義のソースを使用して下さい。

例:

```
VIN1 IN1 0 DC 5.0 SIN 0.0 1.0 1K
VIN2 IN2 0 DC -5.0 SIN 0.0 1.0 10K
FOUT OUT VGND POLY(2) VIN1 VIN2 0.0 1.0 1.0
```

電流制御電圧ソース

説明

```
Hname out+ out- vsource val
```

または

```
Hname out+ out- POLY(n) vsource1 ... vsourcen p0 p1 p2  ... pn
```

Hnameは電圧ソースに使用されるインスタンス名で、Hで始まる必要があります。

out+とout-はそれぞれ電圧ソースの正の出力ノードと負の出力ノードを表わします。

最初の形式は、単純な電流で制御される電圧ソースです。例えば、ゲインの値valを以下のように指定することができます。

```
Vout = val*I(vsource)
```

ここで、Voutはout+とout-との間の電圧で、I(vsource)は、vsourceという名前の電圧ソースを通る電流です。

２番目の形式は、複数の入力で制御される電圧ソースを表わします。それぞれのvsourceiは、それぞれpi係数と関連付けられます。p0は定数項です。

```
Vout = p0
+ p1•I(vsource1)
+ ...
+ pn•I(vsourcen)
```

➔ nは10までです。

➔ アドバンストSimのE, F, G, H デバイスは線形のデバイスです。すなわち、それらはSPICEの中で完全多項式で表現されません。非線形の関係が必要な場合、数式定義のソースを使用して下さい。

例:

```
VIN1 IN1 0 DC 5.0 SIN 0.0 1.0 1K
VIN2 IN2 0 DC -5.0 SIN 0.0 1.0 10K
HOUT POUT NOUT POLY(2) VIN1 VIN2 0.0 1.0 1.0
```

方程式定義ソース

前のセクションで説明した電流制御付き、または電圧制御ソースは、単純な線形のシミュレーションを行なうには有効な方法です。例えば、電圧増幅器をシミュレートするには、“E”デバイスを使用して電圧ゲインを指定することができます。（前の章の電圧制御付き電圧ソースを参照して下さい。）E, F, G, Hデバイスにより、単純な線形モデルが予め定義されています。しかし、信号間のより複雑な関係をシミュレートしなければならない場合があります。

アドバンストSimの場合、電圧や電流を含む任意の非線形式を記述して電圧ソースや電流ソースの値を定義することができます。さらに、コンディショナルステートメントにより、条件に応じたソースの値を定義することができます。これについて以下で説明します。

式

非線形式を定義するために使用できる最初の簡単な形式は、

```
Ename n1 n2 VALUE { equation }
```

（これはコントロール付きの電圧ソースです。）

または、

```
Gname n1 n2 VALUE { equation }
```

（これはコントロール付きの電流ソースです。）

equationの項は数式を表わし、+-*/の演算子やsin()やsqrt()等の数学関数、回路の電圧や電流を記述できます。もっと正確には、equation内の変数としてノードとグランドとの間の電圧や２つのノード間の電圧、電圧ソースを通過する電流を使用することができます。ノードnとグランドとの間の電圧は、V(n)と表現し、ノードnとmとの間の電圧はV(n,m)、Vs電圧ソースを通過する電流はI(Vs)と表わします。式が長すぎて一行に収まらない場合、式が終わっていないことを示すためのキャラクタ+を用いて数行にわたって式を記述することができます。

+や-の文字を持つノード名は式の中で使用できません。これらの文字は演算子として予約された文字です。

例1

```
Esum sum 0 VALUE { sqrt(V(in)) + 0.1 * abs(V(out, in)) }
```

この式では、ノードoutとノードinとの間の電圧の絶対値を取得し、これに0.1を掛けた結果にノードinの電圧のルートを加えます。

式の中でスペースが使用できることがわかると思います。スペースを有効に使用することで式を分かりやすく記述することができます。また、式が64文字を越える場合、スペースを使用する必要があります。そうでなければ、“Word is too long”というメッセージが表示されます。スペースで算術演算子を囲むようにすれば上のようなことは起こりません。

記述した式が複雑な場合、パターンファイルの中で.FORMULASIZEディレクティブを使用して使用できる式の複雑さを増やすことができます。

➔ オンラインヘルプのディレクティブのトピックの.FORMULASIZEを参照して下さい。

例2

```
VALIM VDD 0 DC 5V
EPWR powervalue 0 VALUE { 1e-6 * V(VDD) * I(VALIM) }
```

EPWRソースにより、ノードVDDの電圧とソースVALIMの電流の積の出力が生成される。結果は読み取りやすいようにμWで出力される。

条件形式 (if...then...else)

上で説明された単純な形式は、非線形で定義されるソースの値のような単純な問題の解決には有効です。これでは不十分な場合があり、基本的な制御構造を使用してソースの動きを定義したい場合があります。アドバンストSimでは、強力なIF...THEN...ELSE構造によりこのような問題を解決することができます。

この構造の構文は

```
Ename n1 n2
+ IF { condition }
+ THEN  { expression1 }
+ ELSE  { expression2 }
```

または

```
Gname n1 n2
+ IF { condition }
+ THEN  { expression1 }
+ ELSE  { expression2 }
```

式のつながりを表わす+文字は必要ありませんが、分かりやすく記述するために使用子他方がいいと思います。3つの節（IF, THEN, ELSE）は必ず必要でこの順番で記述して下さい。

ソースの値を評価するには、アドバンストSimによりまずコンディションが評価されます。結果が“TRUE”の場合、ソースの値はexpression1になり、“FALSE”の場合、expression2になります。

conditionは以下の形式で記述する必要があります。

```
expa testop expb
```

ここで、expaとexpbはこれまでに説明した記述ルールに従う算術表現で、testopは関係算術演算子です。

testopで使用できる演算子は、>, >=, <, <=, ==, !=です。

➔ ==は等しい（equal）ことを表わし、!=は等しくない(not equal)ことを表わします。

例えば、正弦波波形にダイオードを適用した場合のシミュレーションを実行したい場合、以下のように記述します。

```
* in circuit.NSX:
VS S 0 SIN 0 1 1K
ED SD 0
+ IF { V(S) > 0.0 }
+ THEN { V(S) }
+ ELSE { 0.0 }
```

➔ IFステートメントの中で他のIFステートメントを記述することはできません。ただし、中間出力を使用して、複数のコンディション問題を扱うことができます。

演算子

使用できる算術演算子は：

```
+ - * /
```
です。

関数

使用できる数学関数は：

```
sin, cos, abs, sqr, sqrt, log, exp, ln, p10, tan, atan, sgn,
time, x
```
です。

sgn関数は引き数の正負をかえします。（引き数が正の場合、+1、負の場合、-1）time関数はシミュレーション時間をかえします。time関数は関数として使用し、引き数はありません。x()関数は、解析の現在の横軸座標をかえします。過渡解析の場合、time()関数と同じです。DC解析では、スイープするソースの現在の値をかえします。

電気的変数

使用できる電気的変数は：

```
V(n1), V(n1, n2), I(Vindep)
```
です。

パラメータ

.PARAMステートメント（ディレクティブ）で定義されるパラメータも同様に使用できます。識別子として直接使用することができます。

例:

```
* in circuit.PAT
.PARAM VTHRESH = 2.5
E1 OUT 0 IF {V(S) > VTHRES*2} THEN { 5.0 } ELSE { 0.0 }
```

テスト演算子

使用できる演算子は：

>, >=, <, <=, ==, != です。

➔ これらの式で定義されるソースにより、信号間の複雑な関連を扱うことができます。しかし、シミュレーションの内部の構造は複雑になるため、注意して使用する必要があります。例えば、“E”デバイスや“G”デバイスでも可能な処理の場合は、効果があまりないので数式を使用しない方が賢明です。また、式の記述には注意して下さい。（ゼロによる割り算、logやsqrt関数への負の値などです。）

ルックアップテーブル

ルックアップテーブルを使用して制御付きの電圧や電流ソースを記述することもできます。この方法は非解析的なモデルを作成したい場合に有効な方法です。もっとも単純なものは、ポイント間の線形補間を行うテーブルです。その他では、ポイント間をスムースな曲線で補間する3次スプライン補間があります。または、いわゆるナチュラルスプラインすなわちテーブルの終端ポイントで0次微分を用いるスプラインを使用することができます。制御入力には、電圧V(N)または、電圧差V(N1, N2)、電圧ソースを通過する電流I(VSRC)を使用することができます。テーブルには任意の数の点を記述することができますが、性能面からあまり大きなテーブルの使用は控えた方が賢明です。

テーブルタイプの電流ソースの構文は：

```
Gxxxx OUTP OUTM TABLE|S_TABLE|N_TABLE { V(N)|V(N1,N2)|I(VSRC) }
=
+ (x0, y0) (x1, y1)
+ ...
+ (xn, yn)
```

この表現は、n+1点のテーブルを表します。ソースGxxxxをとおりノードOUTPからノードOUTMへ、電流が流れます。制御入力には、単純な電圧V(N)または電圧差V(N1, N2)、電圧ソースを通過する電流I(VSRC)を使用できます。もっと複雑な制御入力を使いたい場合、中間的な数式定義のソースを組み込んで下さい。テーブルは、一連の(xi, yi)の組みで定義します。ここで、xiは出力yiに対応する入力値です。ここでは、xiは電圧か電流で、yiはソースを通過する電流になります。入力表現が中括弧 {} で囲まれていることに注意して下さい。また、テーブル定義の最初に=があることに注意して下さい。TABLEキーワードを使用すれば線形テーブルを組み込むことができます。N_TABLEキーワードを使用すれば、ナチュラルスプラインテーブルを組み込むことができます。（テーブルに応じて点x0とxnで微分が行われ、x0とxnでの2次微分がゼロになる。）x0とxnでの微分値を強制的にゼロにする。S_TABLEキーワードを使用すれば、スプラインテーブルを組み込むことができます。すべての場合（線形テーブルまたはスプラインテーブル）で、入力が[x0, xn]の範囲外の場合、終端値y0またはynが使用されます。点x0とxnでの振る舞いが重要な場合、これらの点でスプラインの形状を確認することを推奨します。（スプラインによって、滑らかな補間多項式が得られますが、過渡が急峻な場合振動する可能性があります。）

同様に、テーブルタイプの電圧ソースの構文は：

```
Exxxx OUTP OUTM TABLE|S_TABLE|N_TABLE { V(N)|V(N1,N2)|I(VSRC) } =
+ (x0, y0) (x1, y1)
+ ...
+ (xn, yn)
```

この場合、yiの値が電圧になります。

ラプラス変換ブロック

このデバイスは、ラプラス変換H(p)により定義される。あらゆるタイプのシミュレーションで使用できます。しかし、このデバイスを使用する周波数解析はスモールシグナル解析または過渡解析です。伝達関数の一般的な形はN(p)/D(p)となり、ここでNとDは“p”の多項式です。（ラプラス変数に“s”を使う場合もありますが、ここでは“p”を使用します。）

伝達関数を定義するには、二つの方法があります。2つの多項式N(p)とD(p)の係数の値を与えます。または、極と零点の値を与えることもできます。（2番目の方法がよく使われる定義の方法です。）ここでは、2つの方法の一般形を示します。

最初の方法の一般的な形式

```
Sname n1 n2 in1 in2
+ A0=a0 A1=a1 ... AN=an
+ B0=b0 B1=b1... BM=bm
+ [GAIN(freq)=gain]
+ [UNIT=RAD|HERTZ]
```

Snameはデバイス名で、Sで始まる必要があります。n1とn2はデバイスの出力、in1とin2はデバイスの入力です。デバイスの動作は、差分入力V(in1)-V(in2)を持ち、n1とn2の差分出力を持つ理想電圧ソースに似ています。ほとんどの場合は、入力と出力はグランドに接地されます。数値a0, a1, ..., an, b0, b1, ..., bmは、N(p)とD(p)の係数です。この場合の伝達関数はN(p)/D(p)となり、

$$N(p) = a0 + a1 \cdot p + a2 \cdot p^2 + \ldots + an \cdot p^n$$
$$D(p) = b0 + b1 \cdot p + b2 \cdot p^2 + \ldots + bm \cdot p^m$$

nとmの最大値は19です。更に高次のシステムのシミュレーションが必要な場合、幾つかのSデバイスを直列に接続して使用して下さい。

➔ 定義しなかった係数のデフォルト値はゼロです。

分子の次数は分母の次数と同じかそれよりも小さくなければなりません。すなわち、n <= mとなります。

2番目の方法の一般的な形式

```
Sname n1 n2 in1 in2
+ POLE=p0
+ POLE=p1
...
+ POLE=pn
+ ZERO=z0
+ ZERO=z1
...
+ ZERO=zm
+ [GAIN(freq)=gain]
+ [UNIT=RAD|HERTZ]
```

この2番目の形式は、シミュレーションを行いたい伝達関数の極と零点の位置がわかっている場合に使用します。この方法は、周波数フィルタのシミュレーションを行う場合に特に有効です。

各極と零点は、共役な値でも共役でないあたいでも使用できます。共役でない値の場合、分母の次数は１ずつ増えます。共役な値の場合、分母の次数は２ずつ増えます。分子と分母の次数がそれぞれ19まで極と零点を定義することができます。共役でない極や零点を定義するには、一つの正の値を記述するだけです。共役の極や零点を定義するには、2つの値をカンマで区切って記述します。（実部-正の値、虚部）

例えば：

```
S1 S 0 IN 0
+ POLE=10K
+ POLE=1K
+ ZERO=332,12.5
+ UNIT=HERTZ
```

➔ コマンドの続きを表す文字 + は必要ありませんが、判りやすく記述するために使用しています。

* このデバイスには、1Kと10Kの位置に二つの実数の極があり、332+j•12.5の位置に一つの複素数の零点があります。分子の次数は２で分母の次数も２です。伝達関数は：

$$H(p) = \frac{(p + 332 + j \cdot 12.5) \cdot (p + 332 - j \cdot 12.5)}{(p + 10000) \cdot (p + 1000)}$$

GAINパラメータとUNITパラメータ

GAINパラメータは任意です。伝達関数が特定の周波数で特定のゲインを持つようにゲインを定義するために使用します。例えば、通過帯域である増幅のある単純なローパスフィルターを記述するには、以下のようにします：

```
SLP OUT 0 IN 0
+ POLE=100K
+ GAIN(0)=10dB
```

➔ サフィックスdBを追加してゲインをデシベル単位に設定します。

UNITスペシフィケーションは任意です。このスペシフィケーションは行末に最後のキーワードとして記述する必要があります。これにより、極や零点の位置をラジアンまたはヘルツで指定できます。デフォルトの単位は、HERTZです。単位の指定は、オプション指定のゲインのfreqパラメータに適用されます。

例：

```
SL OUT 0 IN 0
+ AO=1 A1=1 A2=3
+ B0=1 B3=1
+ GAIN(0)=10

SLAP OUT 0 EP EM
+ POLE=10
+ POLE=1MEG
+ GAIN(0)=0dB
```

➔ ラプラスデバイスをシミュレートするために使用されるアルゴリズムはFFTではありません。従って過渡解析の結果に驚かないで下さい。しかし、ラプラスデバイスを使用する場合、最大の時間ステップがアドバンストSimの生成する結果に影響します。過渡解析の結果の精度を検証するには、デフォルトの時間ステップh1でシミュレーションをまず調整し、それから最大時間ステップをh1/10でシミュレーションをすることをお薦めします。二つのシミュレーション結果の差が無視できるくらいになるまでこの作業を繰り返して下さい。

サブサーキット

説明

```
Xname  n1 n2 n3 ... [\ p1 p2 ...]  typename
```

または

```
Xname  n1 n2 n3 ... typename [PARAMS: p1name=p1, p2nam2=p2 ...]
```

➔ サブサーキットについてこの章では簡単な説明に留めます。詳しい説明については、*階層化記述*の章を参照して下さい。

Xnameは、サブサーキットのインスタンス名でXで始まる必要があります。

typenameは、サブサーキットのタイプ名です。（SUBCKT宣言で使用される名前）

n1 n2 ...は、インスタンスに接続されるノードです。p1 p2 ...は、オプションのパラメータでサブサーキットに渡される引き数です。（この場合、宣言はパラメータ化される必要があります。）

➔ パラメータ化されたサブサーキットの呼び出しのための構文は宣言の構文と一致している必要があります。パラメータに¥構造を用いて.SUBCKTを宣言する場合、¥構造を使用してインスタンスを宣言して下さい。PARAMS:構文で.SUBCKTを宣言する場合、PARAMS:構文でインスタンス宣言して下さい。

電圧ソースと電流ソース

➔ 詳細な文法の記述は、この章のほかの節を参照して下さい。

ソースはネットリストファイルかパターンファイルのどちらかに記述できます。しかし、回路の外部ピンにスティミュリが接続されるモデルのソースが位置するパターンファイルに記述するのが好ましいと思われます。パターンファイルには、更新メカニズムが作動しますが、ネットリストファイルには更新メカニズムが作用しないためです。また、別のファイルで外部スティミュリを定義することにより、何人かで同じプロジェクトで作業する

場合に非常に便利になります。回路モデルが一度構築されると、参照として使用し変更を加える必要がなくなります。

"Update pattern file"オプションがチェックされたパターンファイルに定義されているソースのパラメータをOutputs/Sourcesダイアログを使用して変更する場合、ダイアログのOkボタンを押してダイアログを抜ける時パターンファイルに変更事項がレポートされます。ネットリストファイルに記述されたソースではこのようなことはできません。

さらに、ソースは通常トランジスタのようなデバイスとして回路に属しているわけではありません。したがって概念的にもソースの位置する場所はパターンファイルが適当です。

デジタルプリミティブ

イントロダクション

アドバンストSimは、Verilog-HDLの階層モデル機能をサポートしています。この章では、デジタルシミュレーションの原理とVerilog-HDLの階層モデル機能をアドバンストSimで実行する場合のデジタルプリミティブについて説明します。Verilog-HDLの階層モデルについては、Verilog LRM（ランゲージリファレンスマニュアル）の6章を参照して下さい。Verilog LRMを読まれることをお薦めします。この章では、実行のときの注意やゲートレベルのモデリングの簡単なリファレンスを説明します。

論理値とストレングス

アドバンストSimは、Verilog-HDLで定義されたモデルのデジタルシミュレーションを実行できます。簡単なロジックシミュレーションであれば、4つの基本的な論理値：0, 1, X, Zがシミュレーションされることを知っているだけで十分です。MOSネットワークやプルアップ、プルダウンデバイスやプリチャージなどの論理シミュレーションを行う場合Verilog-HDL LRMの第6章を勉強する必要があります。以下のセクションでは、LRMで定義される論理シミュレーションモデルに簡単に説明します。

デジタルネットまたはゲートの出力ピンは論理値（0, 1, X, Z）とストレングスを持っています。ストレングスの概念は、MOSネットワークなどの回路のシミュレーションを正確に行う場合に必要です。論理値は、ノード上に現れる電圧のイメージです。ストレングスはノードまたはピンのインピーダンスのイメージです。Verilog-HDLのlogic-1ドメインのストレングスモデル信号では、あるひとつの信号に対して、非常に弱い信号、hz1（hz1の意味は“ハイインピーダンスロジックの一つ”という意味です。）から、強い信号、supply1（もっとも強い）までの8つのストレングスを定義できます。同様に、logic-0ドメインの信号のストレングスはhz0からsupply0までの8つのストレングスを定義できます。このように、16のストレングスレベルがあります。すなわち、既知の値を持つ信号がある場合、ストレングスレベルは0ストレングスサイドか1ストレングスサイドのどちらかにあるということになります。未知の信号は両方のサイドにストレングスレベルがあります。信号のストレングスは、曖昧でない（信号のストレングスレベルが一つの場合）か曖昧である（信号のストレングスレベルが一つでない）かのどちらかです。以下の図に示すように、ストレングスレベルを対照的なスケールで考えることができます。信号のストレングスは、スケールの一点として表される（例えば、St1）か、ある範囲として表される（例えば[Pu0:St1]、これはPu0からSt1の範囲のすべてのストレングスレベルに信号が含まれることを意味します。）かのどちらかです。

0 strength side								1 strength side							
7	6	5	4	3	2	1	0	0	1	2	3	4	5	6	7
Su0	St0	Pu0	La0	We0	Me0	Sm0	Hz0	Hz1	Sm1	Me1	We1	La1	Pu1	St1	Su1

➔ 8段階のストレングスレベルの内、5つはドライビングストレングス（Su/pply, St/rong, Pu/ll, We/ak, Hz）として分類され、その他の3つ（La/rge, Me/dium, Sm/all）は、チャージストレージストレングスとして分類されます。ドライビングストレングスはゲート出力からきています。ゲート出力のデフォルトのドライブストレングスは、Strong0またはStrong1です。これらのデフォルト値を変更したい場合、各ゲートの設定にドライブストレングスのデフォルト値を定義することができます。チャージストレージストレングスは、triregタイプのネットからきています。（ネットタイプはこの章の“ネットタイプ”セクションで説明します。）

下図は、0, 1, X, Zの値を持つ“ノーマル”信号値のストレングスインターバルを示します。

Logic 0

7	6	5	4	3	2	1	0	0	1	2	3	4	5	6	7
Su0	St0	Pu0	La0	We0	Me0	Sm0	Hz0	Hz1	Sm1	Me1	We1	La1	Pu1	St1	Su1

Logic 1

7	6	5	4	3	2	1	0	0	1	2	3	4	5	6	7
Su0	St0	Pu0	La0	We0	Me0	Sm0	Hz0	Hz1	Sm1	Me1	We1	La1	Pu1	St1	Su1

Logic X

7	6	5	4	3	2	1	0	0	1	2	3	4	5	6	7
Su0	St0	Pu0	La0	We0	Me0	Sm0	Hz0	Hz1	Sm1	Me1	We1	La1	Pu1	St1	Su1

Logic Z

7	6	5	4	3	2	1	0	0	1	2	3	4	5	6	7
Su0	St0	Pu0	La0	We0	Me0	Sm0	Hz0	Hz1	Sm1	Me1	We1	La1	Pu1	St1	Su1

➔ いわゆるインタフェースノードと呼ばれるアナログエレメントとデジタルエレメントの両方に連結するノードは、さらに特別な処理がなされます。詳細については、アナログ／デジタルインタフェースの章を参照して下さい。

シミュレーションモデル

デジタルゲートのシミュレーションのためにアドバンストSimで使用されるモデルは“ワイヤードモデル”と呼ばれ以下のルールに従って記述されます。

- ゲートは入力ピンと出力ピンです。
- ノードのように、ゲートの出力ピンの状態は、値とストレングスレベルで定義します。（スケール内での領域については、前の図で説明しました。）
- ゲートは、入力ピンに接続されるノードの値とストレングスレベルに応じて変化する。
- 入力や論理関数、伝播遅れの状態に応じてゲートにより出力ピンのイベントのスケジュールが決定されます。
- ノードの状態はコンフリクトソルバーにより決定されます。コンフリクトソルバーにより、ノードを駆動するゲートの出力ピンの状態が解析され、ノードの状態が計算されます。
- ノードの状態が変わる場合、このノードで駆動されるゲートはアクティブにされます。

➔ シミュレーションの結果を理解しようとする場合、知っておかなくてはならない事は、ノードの状態は常にノードを駆動する出力ピンの状態との結合結果であるという事です。これは一般的なモデルで、基本的なアルゴリズムを修正する事無く、例えばスイッチのシミュレーションも可能です。

➔ 異なる強度レベルの信号の結合に使用されるメカニズムの詳細は、Verilog-HDLリファレンスマニュアル（第6章）に記述されています。

ディレイモデル

ゲートディレイモデルとは慣性的ディレイモデルのことで、そのゲートの定義ディレイより短い幅を持つ出力のパルスが濾過されます。（出力上には何ら変化はありません。）

ディレイは、min:typ:maxというフォーマットで指定します。この3つの値でディレイを定義します。（最小ディレイ、通常ディレイ、最大ディレイをコロンで区切って定義します。）

➔ 重要：ゲートのディレイやネットのディレイは、標準のVerilog-HDLタイムスケールディレクティブにより指定されるパラメータの値から計測されます。この値の単位は秒でdoubleの精度があります。

ネットタイプ

Verilog-HDLでは、ネットタイプがデザインのすべてのネットに割り付けられます。ネットタイプには以下のものがあります。：

wire, tri, wand, wor, triand, trior, trireg, supply0, supply1, tri0 and tri1

wire（または、wireとして同じtri）が、もっとも一般的です。明確なタイプの宣言がないネットは、ワイヤネットになります。

電源を持つsupply0モデルやsupply1モデルのあるネットは、ネット上に電源と一緒に定義されます。これらのネットのストレングスはsupplyです。

tri0モデルタイプを持つネットは、プルダウンと一緒にネット上に定義されます。triregノードの少なくとも一つのドライバが0, 1, Xにドライブされる場合、triregノードは、ドリブ

ンステートです。その値は通常の競合解決の結果になります。ドライバがすべて接続されない場合、（Zにドライブされる場合）trireg ノードは最後にドライブされた値を保持します。キャパシティブステートの場合、ネットのストレングスは、チャージストレージストレングス（スモールまたはミディアム、ラージ）のどれかになります。これにより、小さいコンデンサ、中位のコンデンサ、大きいコンデンサにモデル化されます。

例：

```
wire a, b, c;
wire [15:0] q;
wire [15:0] #(10) q;
tri1 out;
supply0 vss;
supply1 vdd;
trireg (small) out;
trireg (large) [7:0] #(5) qbus;
```

➔ チャージの放出については、インプリメントされていません。

ネットディレイ

ネットにディレイを割り付けることができます。ネットディレイにより、駆動がかけられた時点から値が変わるまでにかかる時間がモデル化されます。さらに、伝播遅れもモデル化できます。ネットディレイは、ネットタイプ宣言と一緒に宣言します。

例：

```
wire #(15) out;
buf #(10) b(out, in);
```

ネットの値が変わると、ネットの出力は10+15 時間単位後に変わります。

ネットディレイには、3つの値を定義することができます。一つだけ値が与えられた場合、その値がすべてのトランジションに適用されます。2つの値が与えられた場合、立ち上がりと立ち下がりのトランジションに適用されます。3つの値が与えられた場合、立ち上がりのトランジションと立ち下がりのトランジション、値がZになるときのトランジションに適用されます。Xになる時のトランジションは、先程の3つのディレイに比べて無視できるほど小さいです。

例：

```
wire #(10) outall;
wire #(10,12) outrisefall;
wire #(10,12,40) outrisefallz;
```

ネットディレイも同様に、min:typ:maxフォーマットで指定できます。シミュレーションで実際に使用されるディレイは、.SELECTDELAYディレクティブで決定されます。以下に例を示します。

```
wire #(8:10:12, 9:11:15) outmtm;
```

ロードディペンデントディレイ: コンデンサプリミティブとマージナルディレイ係数

アドバンストSimでは、論理ゲートのロードディペンデントディレイモデルをサポートすることにより、Verilog-HDLの基本機能を拡張しています。Verilog-HDLでは、プリミティブのディレイは、以下のように定義されます。

```
nand #(10, 12) a1(y, a, b);
```

この構文では、境界係数をサポートしていません。この問題を解決するために、アドバンストSimではディレクティブを使用します。これにより、境界係数やプリミティブとして « capacitor »を定義することができます。

ここでは、アドバンストSimでどのように境界遅れ係数を定義するかについて説明します。

```
`marginal_delay_coefficient [mr = <mr>] [mf = <mf>] [mz = <mz>]
`end_marginal_delay_coefficient
```

上のディレクティブは、アドバンストSimに取り込むことができます。<mr>, <mf>, <mz> は実数です。`marginal_delay_coefficientディレクティブはすべてのゲートとディレクティブ

が定義された位置から次の`marginal_delay_coefficientまたは`end_marginal_delayまでに定義されるすべてのパスディレイに適用されます。これらのゲートには、`marginal_delay_coefficientディレクティブ（<mr>, <mf>, <mz>）で定義される境界遅れ係数が適用されます。これらのディレクティブは、モジュール定義の中にも外にも記述できます。

以下のプリミティブが定義されます。

capacitor #(<capa_value>) [<instance_name>](<net_name>);

ネット<net_name>に値<capa_value>を持つコンデンサが追加されることを意味します。幾つかのコンデンサプリミティブが同じネットに接続される場合、それらは加算されます。

これらの値をもっと柔軟に定義できる２つのディレクティブを紹介します。

`scale_capacitor <scale_value>

`scale_marginal_delay_coefficient <scale_value>

`scale_capacitorディレクティブにより、コンデンサのインスタンスの値の縮尺が定義されます。（上記の例では<capa_value>の値）縮尺<scale_value>は、実数です。以下に例を示します。

`scale_capacitor 1f

上のように記述すると、コンデンサの値はすべてフェムトファラッドになります。従ってcapacitor #(25) (out)は、ノードoutに25フェムトファラッドのキャパシタンスがあることを意味します。

`scale_capacitor 10p

は、コンデンサの値は10ピコファラッドであることを示します。すなわち、capacitor #(0.018) (out)は、ノードoutに0.18ピコファラッドのキャパシタンスがあることを意味します。

`scale_marginal_delay_coefficientディレクティブにより境界遅れ係数の値の縮尺が定義されます。（上記の例の<mr>, <mf>, <mz>の値）縮尺は実数です。

`scale_marginal_delay_coefficient 0.1e3

上記のように定義すると、境界遅れ係数に0.1e3がかけられ、その後最終的な立ち上がりと立ち下がり時間の遅れが計算されます。

複雑な例を示します：

```
// gate delays will be given in nano-seconds:
`timescale 1ns / 1ps
// capacitor values will be given in femto-Farads:
`scale_capacitor 1f
// marginal delay coefficient will be in ns/pF (nano-seconds per
pico-
Farad)
`scale_marginal_delay_coefficient 1e3

module adder(a, b, c, s);
input a,b,c;
output s;

not #(1.4) n1(ny, y);
`marginal_delay_coefficient mr=2.3 mf=2.4
nand #(3.8, 4.5) n2(w, a, b);
`end_marginal_delay_coefficient
xor #(4.5, 3.2) n3(w2, a, b, c);
// other components...


capacitor #(12) c1(ny);
capacitor( #(58) c2(w);

endmodule
```

ゲートn1には、1.4nsの遅れが定義されています。これは、キャパシタによって影響されません。というのは、最初の`marginal_delay_coefficientディレクティブの前に定義されているからです。ゲートn2は、境界遅れとキャパシタの影響を受けます。n2の立ち上がり遅れは、3.8n + (2.3*1e3) * (58 * 1f) = (3.93ns)です。また、立ち下がりの遅れは、4.5n + (2.4 * 1e3) * (58*1f) = (4.64ns)です。ゲートn3は、境界遅れもキャパシタの影響も受けません。なぜなら、`end_marginal_delay_coefficientディレクティブのすぐ後でゲートn3が定義されているからです。

➔ デフォルトの単位系はMKSAです。従って、スケーリングディレクティブを使用しない場合、遅れは秒、コンデンサ容量はファラッド、境界係数は秒／ファラッドで定義する必要があります。

ゲートとスイッチ

アドバンストSimは、プリミティブとして多くの基本的なデジタルゲートやスイッチをサポートしています。プリミティブであることは、アドバンストSimはその動作をシミュレートする方法を知っていることを意味します。回路のデジタルパーツは、これらのプリミティブを仲介してモデル化されます。これは、ユーザにより定義されたプリミティブで同じです。（LRMの7章を参照して下さい。）多くのデジタル回路は、階層構造のネットリストで記述されます。

➔ *階層記述*の章を参照して下さい。

プリミティブは、階層記述の最下部に位置します。プリミティブゲートには、モジュールの定義が必要ありません。Verilog-HDLでサポートされているプリミティブの他に、tran, tranif, rtran, rtranifがサポートされます。これらは、アドバンストSimのインプリメンテーションではサポートされません。

プリミティブを使用する場合、以下のルールが適用されます。

- ゲートタイプは小文字のキーワードです。
- インスタンス名はオプションです。
- インスタンス名をベクトル化することが可能です。これによりいろいろなゲートのインスタンスを一行に書くことができます。
- ディレイは、単一の値でもmin:typ:maxフォーマットでも定義可能です。ゲートのディレイがmin:typ:maxフォーマットで定義される場合、.patファイルの.SELECTDELAYディレクティブによりディレイが選択されます。
- and, nand, or, nor, xor, xnorゲートは、他入力で一出力です。
- notとbufゲートは、入力が一つで出力に複数設定できます。
- notif0とnotif1, bufif0, bufif1ゲートは、入力が一つでコントロール入力が一つ、さらに出力が一つです。
- nmos, rnmos, pmos, rpmosゲートは、データ入力が一つでゲート入力が一つ、さらに出力が一つです。
- pullupとpulldownゲートには、入力も出力もありません。

➔ これらのゲートはすべて組み合わせゲートです。シーケンシャルゲート（フリップフロップやラッチ）は、プリミティブにはありません。このようなコンポーネントが必要な場合、シーケンシャルデバイスをモデル化したUDP（ユーザ定義プリミティブ）を作成して下さい。

ゲートのインスタンスの構文

以下に、ゲートのインスタンスのBNF構文（Verilog LRM 2.0から）を示します。

```
<gate_declaration>
  ::=<GATETYPE><drive_strength>?<delay>?<gate_instance>
    <,<gate_instance>>*;

<GATETYPE> is one of the following keywords:
  and nand or nor xor xnor not buf notif0 notif1 bufif0 bufif1
    pullup pulldown nmos rnmos pmos rpmos cmos rcmos

<drive_strength>
  ::=(<STRENGTH0>,<STRENGTH1>)
```

```
    ||=(<STRENGTH1>,<STRENGTH0>)

  <delay>
    ::=#number
    ||=#<identifier>
    ||=#(<mintypmax_expression><,<mintypmax_expression>>?
      <,<mintypmax_expression>>?)

  <gate_instance>
    ::=<name_of_gate_instance>?(<terminal><,<terminal>>*)

  <name_of_gate_instance>
    ::=<IDENTIFIER><range>?

  <range>
    ::=[<constant_expression>:<constant_expression>]

  <terminal>
    ::=<IDENTIFIER>
```

ゲートタイプの定義

GATETYPEキーワードは、定義の中に記述されている必要があります。このキーワードにより、ゲートのタイプが指定されます。

➔ tranとtranifゲートは、アドバンストSimのインプリメンテーションではサポートされていません。

ドライブストレングスの定義

ドライブストレングスの定義は、ゲートインスタンスの出力端子の論理値のデフォルトのストレングスを変更する場合に使用して下さい。以下にあげたゲートで、ドライブストレングスを定義できます。

and, nand, or, nor, xor, xnor, buf, not, bufif0, bufif1, notif0, notif1, pullup, pulldown

その他のゲート（nmosやpmosなど）では、ドライブストレングスを定義することはできません。

オプションのドライブストレングスの定義には、STRENGTH1とSTRENGTH0の2つのパートがあります。ゲートのインスタンスでは、pullupとpulldownを除いて両方を指定する必要があります。一方のパートは、ゲートの出力ピンが1にドライブされるときのストレングスを表し、もう一方のパートは、ゲートの出力ピンが0にドライブされるときのストレングスを表します。ゲートがXにドライブされる場合、2つのパートにより、Xストレングスの範囲が定義されます。

ドライブストレングスは、通常以下のフォーマットで定義されます。

(STRENGTH0, STRENGTH1)

(STRENGTH1, STRENGTH0)と記述してもかまいません。

STRENGTH1は以下のキーワードの内の一つです。

```
supply1 strong1 pull1 weak1 highz1
```

Verilog-HDLの敏感なので、Strong1は文法チェックの段階で削除されます。

STRENGTH0 は以下のキーワードの内の一つです。

```
supply0 strong0 pull0 weak0 highz0
```

ドライブストレングスの仕様が与えられる場合、ゲートタイプのキーワードも必要です。この記述は、遅れの仕様が記述される前に必要です。ストレングスの仕様(highz0, highz1)と(highz1, highz0)は、無効です。

例：

```
nor (strong0, highz1) n1(out, in1, in2);
```

このnorゲートは、logic 1に代わってlogic 0とlogic Zの強さを出力します。

ディレイ仕様

デフォルトでは、ゲートにディレイはありません。ディレイの指定は、ゲートタイプに応じて3つの値を定義することができます。ディレイの記述は、単純な値を記述するかかmin:typ:maxのように記述することができます。以下のゲート記述のセクションの例を参照して下さい。プルアップとプルダウンゲートにディレイを割り当てることは当然ですができません。

プリミティブインスタンス識別子

フォーマル構文定義のIDENTIFIERはオプションです。ただし、インスタンスをベクトル化する場合、IDENTIFIERは必ず必要です。レンジの指定を参照して下さい。

レンジの指定

インスタンス名に続いてレンジを指定することができます。レンジの指定により、インスタンスのベクトル（配列）が表わされます。この指定は、インスタンスのベクトル（例えばバスドライバ）を接続する時に有効です。レンジの指定は以下のように記述します。

[a:b]

ここで、aとbは整数定数で、aはbよりも大きくなくてはいけません。aとbが同じ場合、単一のインスタンスが生成さえます。aもbもゼロであってはいけません。[a:b]のようなレンジの指定により、abs(a-b)+1個のインスタンスが定義（接続）されます。レンジがインスタンス名に与えられる場合、端子リストとそれらが一致する必要があります。以下の例を参照して下さい。

例：

```
nand n[7:0](out[7:0], in1[7:0], in2[7:0]);
nor nx[3:0](outx[3:0], a, b, c, d, in2[3:0]);
```

and, nand, nor, or, xor and xnor ゲート

これらは単純な組み合わせゲートです。一つの出力を持ち最初に端子リストが与えられ一つ以上の入力を持ちます。オプションでディレイの指定ができ、一つか二つの値を設定できます。値が一つ与えられる場合、その値はすべての出力トランジションに適用されます。２つのディレイが与えられる場合、最初の値は立ち上がりのディレイで２番目の値は経ち下がりのディレイになります。XとZへのトランジションでは、２つのディレイよりも小さい値が使用されます。入力の数により、伝播遅れが変更されることはありません。

入力の論理Zは、論理Xとして解釈されます。出力は常に（ドライブストレングスの指定のない）0, 1, Xとなります。

例：

```
and  a1(out, a, b, c);
and  a1[7:0](out[7:0], a[7:0], b[7:0]);
and  (out, a, b, c);
and  #(2) a1(y, a, b);
and  #(2,3) a1(y, a, b);
and  (strong0, highz1) #(2,3,2) a1(y, a, b);
and  #(2:3:5) a1(y, a, b);
and  #(2:3:5, 3:4:6) a1(y, a, b);
nand (nout, x, y);
nand (weak0, weak1) n1(out, a, b, c);
nand #(2:5:7) n1(out, a[0:3]);
or   #(10) g1(out, a, b);
nor  #(10, 20) g2(nout, a, b);
xor  #(1, 3) g3(out, a, b, c);
xnor g4(nout, a, b, c);
```

buf と not ゲート

これらはの入力は一つで、端子リストの最後に記述されます。出力は端子リストの最初に記述します。出力は一つ以上を記述できます。オプションでディレイの指定ができ、一つか二つの値を設定できます。値が一つ与えられる場合、その値はすべての出力トランジションに適用されます。2つのディレイが与えられる場合、最初の値は立ち上がりのディレイで2番目の値は経ち下がりのディレイになります。XとZへのトランジションでは、2つのディレイよりも小さい値が使用されます。

入力の論理Zは、論理Xとして解釈されます。出力は常に（ドライブストレングスの指定のない）0, 1, Xとなります。

bufif0, bufif1, notif0 and notif1 ゲート

これらのゲートはトリステートバッファとインバータです。コントロール入力が0 (bufif1, notif1)または1 (bufif0, notif0)の場合、これらの出力に論理Zが設定されます。これらのゲートは一本の入力（端子リストの最初）と一本のデータ入力（リストの2番目）、一本のコントロール入力（リストの最後）を持ちます。オプションで1つ、または2つ、または3つのディレイ値を設定できます。値が一つ与えられる場合、その値はすべての出力トランジションに適用されます。2つのディレイ値が与えられる場合、最初の一つは立ち上がりのディレイ、2つめの値は立ち下がりディレイに適用されます。XとZへのトランジションでは、2つのディレイよりも小さい値が使用されます。3つのディレイ値が与えられる場合、最初の一つは立ち上がりディレイ、2つめは立ち下がりディレイ、3つめは、Zへのトランジションディレイに適用されます。さらにXへのトランジションディレイは3つの値の最小値が適用されます。下の表は、bufif0, bufif1, notif0, notif1ゲートの真理値表です。シンボルLとHは、単純なStrong0とStrong1の信号ではなくハイインピーダンスからストロングまでのストレングスレベルを持つ0と1の信号を表わします。

Bufif0 output	Control = 0	control = 1	control = X	control = Z
input = 0	0	Z	L	L
input = 1	1	Z	H	H
input = X	X	Z	X	X
input = Z	X	Z	X	X

bufif1 output	Control = 0	control = 1	control = X	control = Z
input = 0	Z	0	L	L
input = 1	Z	1	H	H
input = X	Z	X	X	X
input = Z	Z	X	X	X

notif0 output	Control = 0	control = 1	control = X	control = Z
input = 0	1	Z	H	H
input = 1	0	Z	L	L
input = X	X	Z	X	X
input = Z	X	Z	X	X

notif1 output	Control = 0	control = 1	control = X	control = Z
input = 0	Z	1	L	L
input = 1	Z	0	H	H
input = X	Z	X	X	X
input = Z	Z	X	X	X

nmos, rnmos, pmos and rpmos gates (switches)

➔ アドバンストSimではゲートとスイッチに違いはありません。アドバンストSimでは、nmosは論理“スイッチ”として機能します。しかし、それはプリミティブな“ゲート”としてシミュレーションが実行されます。

これらのゲートにより、MOSトランジスタがモデル化されます。また、これらは、方向性のあるスイッチ（ゲートが“オン”の場合、入力から出力へデータが流れる）です。多くのCMOSデジタルセルはこれらのスイッチを用いてモデル化することができます。なぜなら、情報の流れは通常すべてトランジスタで行われているからです。（双方向ゲートの場合、このようなことはあてはまりません。）2種類のスイッチが使用可能です。通常は、nmosまたはpmosでもう一つはrnmosやrpmosのような抵抗を持つものです。nmosとrnmosゲートの論理動作は、同じです。rnmosとrpmosはそれらが“オン”のとき、nmosやpmosよりも高いインピーダンスになります。CMOS回路のスタティックプリチャージデバイスはrnmosとrpmosデバイスの例です。

オプションで1つ、または2つ、または3つのディレイ値を設定できます。値が一つ与えられる場合、その値はすべての出力トランジションに適用されます。2つのディレイ値が与えられる場合、最初の一つは立ち上がりのディレイ、2つめの値は立ち下がりディレイに適用されます。XとZへのトランジションでは、3つのディレイよりも小さい値が使用されます。3つのディレイ値が与えられる場合、最初の一つは立ち上がりディレイ、2つめは立

ち下がりディレイ、3つめは、Zへのトランジションディレイに適用されます。さらにXへのトランジションディレイは3つの値の最小値が適用されます。下の表は、ゲートの真理値表です。シンボルLとHは、ハイインピーダンスからストロングまでの範囲のストレングスレベルを持つ信号を表わします。

nmos/rnmos output	control = 0	control = 1	control = X	control = Z
input = 0	Z	0	L	L
input = 1	Z	1	H	H
input = X	Z	X	X	X
input = Z	Z	Z	Z	Z

pmos/rpmos output	control = 0	control = 1	control = X	control = Z
input = 0	0	Z	L	L
input = 1	1	Z	H	H
input = X	X	Z	X	X
input = Z	Z	Z	Z	Z

プルアップとプルダウンゲート

これらの“ゲート”は論理ゲートより、むしろ信号ソースのように動作します。これらのゲートにより、それらが接続されるネット上で論理1（プルアップ）または論理0（プルダウン）がドライブされます。駆動ストレングスが指定されない場合、それらはpullストレングスの信号をドライブします。ディレイの指定はできません。その他のゲート出力がネットに接続されている場合、最終的な論理値とストレングスは、“pull”された値と異なる場合があります。

例：

```
pullup (out);               // pullを論理1に駆動
pullup (weak1) (out);       // weakを論理1に駆動
pulldown (strong0) (out); // strongを論理0に駆動
```

アナログスティミラス

概要

アナログスティミラスは、電圧と電流の独立ソースから構成されます。独立ソースは4種類あります。それらの宣言はスケマチックのコンポーネントに記述することも、ネットリストやパターンファイルに記述することもできます。この章では、パターンファイルにソースを指定する方法について説明します。

➔ ソースを使用する方法については、回路のパワーリングとスティミラスの適用の章を参照して下さい。

ソースパラメータはパターンファイルまたはネットリストファイルのどちらでも宣言することができますが、パターンファイルに記述するようにして下さい。唯一の例外は、内部バイアスを必要とするサブサーキットの場合で、ソース電圧はネットリストファイル内（*circuit*.NSXまたはbloc.CKT）で使用されなければなりません。

独立ソース電圧

一般的な記述：

```
Vname n1 n2 parameters
```

独立ソース電流

一般的な記述：

```
Iname n1 n2 parameters
```

n1ノードは、ソースの正ノード、n2ノードは負ノードです。電流ソースの場合、n1を通りソースを通過してn2へ流れる電流を正として扱います。パラメータにより、独立ソースのDCやトランジェント、小信号の動作を指定することができます。

4種類の電圧/電流独立ソースのタイプがあります。以下の図や説明は、独立電圧ソースの説明です。同じことが独立電流ソースでも言えます。違うのは、最初の文字（電圧ソースはV、電流ソースはI）だけです。

➔ 制御付きソース電圧およびソース電流（線形または非線形コントロール）も使用できます。ただし、スティミラスとして見なされず回路要素として見なされます。これらについては、この章のアナログプリミティブを参照して下さい。

➔ デフォルトでは、ソース電圧およびソース電流の数はそれぞれ64個に制限されています。これ以上の数が回路に必要な場合、パターンファイル内で .MAXSOURCESディレクティブを使用して下さい。

ソースを通過する電流

電圧・電流ソースを流れる電流は、I(Vname), I(Iname)により取得できます。これらの電流はトレースすることができます。オンラインヘルプのディレクティブトピックの .TRACEディレクティブを参照して下さい。

コンスタントソース

構文

```
Vname n1 n2 [DC] val [AC acmag acphase]
```

これは、n1とn2との間にあるソースで、電圧値valの定常DC電圧ソースです。

ここでキーワードACが指定される場合、振幅acmag（ボルト）と位相acphase (degrees)がソースに指定されます。acmagが1に設定される場合、入力と出力との間の伝達関数は直接得られます。acmagとacphaseのデフォルト値は両方ともゼロです。

➔ 小信号解析の場合、一つ以上のソース（電圧または電流）にゼロでないacmagを持つACの指定が必要です。

例：

```
VREF VREF 0 2.5
VIN INP INM DC 0V AC 1 0
```

➔ スパイスとの互換性のために代替の構文を使用できます。以下に例を示します。

```
VS NODE1 NODE2
* これはNODE1とNODE2との間に接続されている0Vのソース電圧です。

VS NODE3 NODE4 DC 3.65
* ここでは、D
```

周期的なパルスソース

構文

```
Vname n1 n2 PULSE v1 v2 td tr tf tw tp [AC acmag acphase]
```

td はソースがv1からv2へ変わる前のディレイです。

tr は増加時間です。（0より大きいこと。）

tf は減少時間です。（0より大きいこと。）

tw はソースが値v2を保持している時間です。

tp は周期です。

時間に関する表記の単位は全て秒です。

v1 と v2 の単位はボルトです。

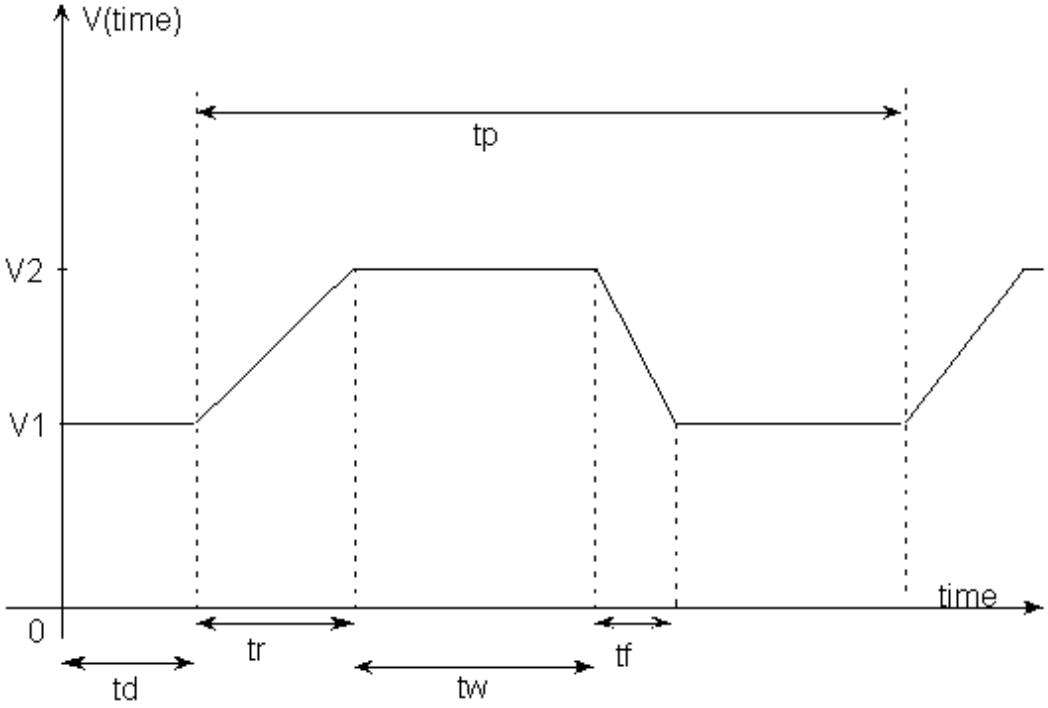
力と出力との間の伝達関数は直接得られます。acmagとacphaseのデフォルト値は両方ともゼロです。

➔ 小信号解析の場合、一つ以上のソース（電圧または電流）にゼロでないacmagを持つACの指定が必要です。

例：

```
VDD VDD 0 PULSE 0 5 0N 5N 5N 50N 100N AC 1 0
// 図のパルスを表わします。

VIN IN 0 PULSE 5 0 0N 10N 10N 25N 100N AC 1 0
VFL INP INM PULSE( 0V 5V 1N 2N 25N 100N )
```



パルスタイプソースの時間推移

ピースワイズリニアソース

構文

```
Vname n1 n2 PWL 0.0 v0 t1 v1...[AC acmag acphase]
```

ソースの値は、v_0からv_1までt_1秒でリニアに切り替わります。その後v_2へt_2秒で同様にリニアに切り替わります。

➔ t_{i+1} はt_iよりも大きい値でなければなりません。

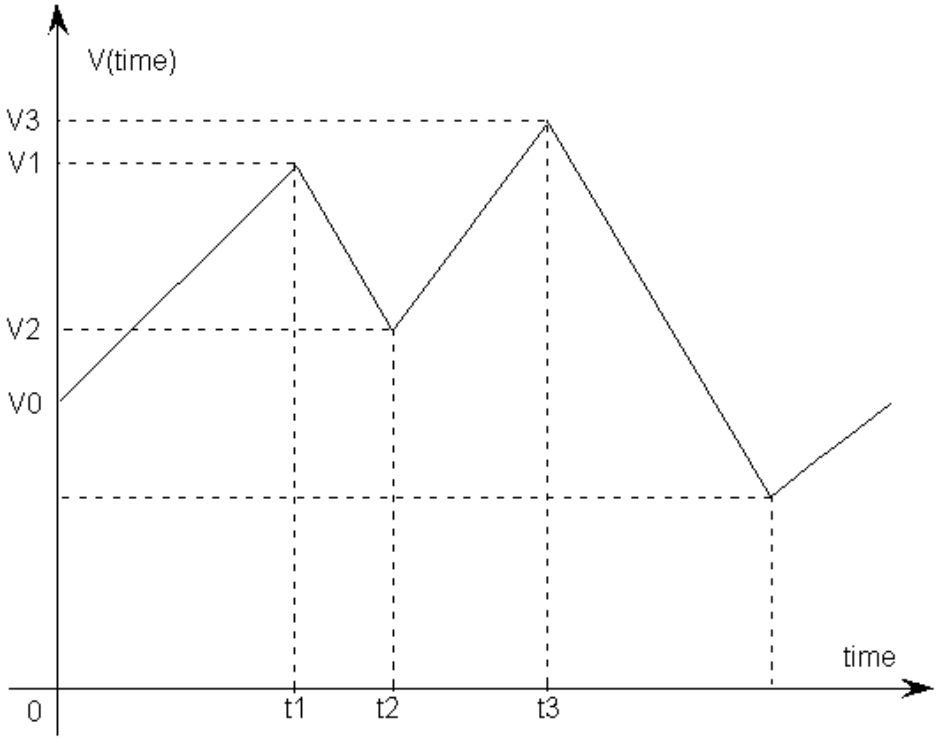
キーワードACがある場合、振幅acmag（ボルト）と位相acphase (degrees)がソースに指定されます。acmagが1に設定される場合、入力と出力との間の伝達関数は直接得られます。acmagとacphaseのデフォルト値は両方ともゼロです。

➔ 小信号解析の場合、一つ以上のソース（電圧または電流）にゼロでないacmagを持つACの指定が必要です。

例：

```
VDD VDD 0 PWL 0 0 10N 5 // 図の様になります。
VIN IN VSS PWL 0N 5V 10N 5V 50N 0V

*互換性のために（）を使用することができます。
VIN IN VSS PWL( 0 5 10N 5 50N 0 )
```



PWLタイプソースの時間推移

正弦波ソース

構文

```
Vname n1 n2 SIN offs ampl freq [delay damp phase]

+ [AC acmag acphase]
```

これはオフセットoffs（V）、振幅ampl（V）、周波数freq（Hz）、ディレイdelay（s）、減衰係数damp（1/s）、及びフェーズphase（度）を持つ正弦波ソースです。delayは、ソースが実際に動作を開始するまでの時間（秒）を表します。（t=0からdelayまで、この値はoffset ボルト）

パラメータdelay、damp及びphaseはオプションとなり、これらのデフォルト値はゼロです。

ソースを表す式：

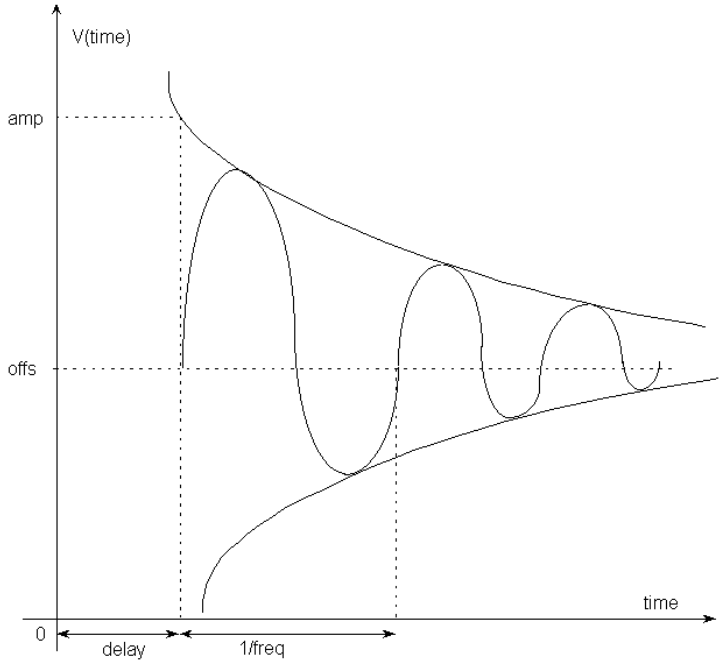
```
V(t) = offs+ampl•exp(-(t-delay)•damp)•sin(2•pi•(f.(t-
delay)+phase/360))
```

キーワードACがあるところでは、振幅acmag（ボルト）及びフェーズacphase（度）がそのソースへ割り当てられています。もしacmagが1に設定された場合、入力と出力間の転送機能が直接得られます。acmag及び acphaseのデフォルトはゼロです。

➔ 小信号解析を行なうには、acmag がゼロ以外の値でACが指定されているソース（電圧又は電流）が一つ以上必要です。

例：

```
VREF REF 0 SIN 0 1 100K 0 0 0
// VREF is the one in the dialog...
VIN IN 0 SIN( 0 0.158 1K 0 100 0 AC 1 0 )
```



正弦波ソースの時間推移

デジタルスティミラス

概要

回路上にデジタルスティミラスを適用するには.CLKを用いる方法と、WAVEFORMを用いる方法があります。.CLKはクロックのような周期パターンの定義に使用し、WAVEFORMは任意のパターンの定義に使用します。

デジタルスティミラスは、入力を持たず1本の出力ピンしかもたない特別なゲートとして内部的にシミュレーションされ、出力ピンに指定されたパターンを出力します。

ほとんどのデジタルスティミラスは、“strong”信号すなわち“Supply”レベルのストレングスをもつデジタル信号により定義されます。デジタル信号は、ハイインピーダンス、レジスティブ、ドライビング、サプライの4種類のストレングスを持つことができることを思い出して下さい。非常にインピーダンスが低いソースで駆動される回路や基板の入力パターンを定義する場合に特に注意が必要です。ただし、このようなことは必ず必要なわけではなく、例えば、“weak”信号を定義するのに.CLKステートメント使用することもできます。

.CLK 文の記述

```
.CLK node 0 state0 [t1 state1 ... ti statei ...] [.REP period]
```

このステートメントは、パターンファイル内に記述しなくてはなりません。ノードは指定されたスティミラスにより駆動される純粋なデジタルノード名である必要があります。.CLK“ゲート”により配られるパターンファイルは時刻tiのときの状態を表すtiとstateiの対と一緒に記述される必要があります。stateiは文字列です。

複数の.CLKステートメントにより、同じデジタルノードを«ドライブ»することができます。これにより、以下で示すような«ホール»を持つクロック信号を生成することができます。

➔ .CLKステートメントを使用してアナログノードまたはインタフェースノードを駆動することはできません。ノードは純粋なデジタルノードである必要があります。デジタルスティミラスでアナログノードやインタフェースノードを駆動する必要がある場合、入力が .CLKステートメントによって駆動され出力がインタフェースノードを駆動するbufゲートのような補助的なデジタルゲートを挿入して下さい。

➔ 重要：デジタルベクトルを記述するために使用される時間やデュレーションはすべて仮想時間の単位で記述する必要があります。（仮想時間と実際の時間との間の対応は、.LTIMESCALEディレクティブで与えられます。）

stateiの値を指定する場合の注意事項を示します。

- もっとも簡単な記述法は、0, 1, X, Zなどの一文字をstateiに記述することです。この場合、信号はSupplyストレングスで論理0, 1, X, Zに駆動されます。stateiがZの場合、ハイインピーダンスに« 駆動 »されます。
- ドライビングストレングスを3文字で指定することもできます。最初の２文字によりドライビングストレングスを指定し、最後の一文字により値を指定します。使用できる文字列は、SU0, ST0, PU0, WE0, SU1, ST1, PU1, WE1, SUX, STX, PUX, WEXです。最初の2つ文字によりノードに接続されるサプライ信号のストレングスが表されます。3番目の文字により、レベルがあらわされます。

周期パラメータは、パターンファイルのオプションです。周期パラメータが指定されない場合、最後のステートがシミュレーションが終わるまで有効になります。

例：

```
.LTIMESCALE 1P
.CLK CLOCK 0 ST0 50000 ST1 .REP 100000
```

上記のステートメントにより、CLOCKは10MHzの“矩形”クロックに定義されます。クロックは時刻ゼロで、“Strong 0”で、時刻50nsで“Strong 1”になります。周期は100nsです。.LTIMESCALEが、1psに定義されていることに注意して下さい。従って、50000は50000ps （50ns）の意味で、周期は100000psすなわち100nsです。

例：

```
.LTIMESCALE 1U
.CLK STRANGE_H 0 0 10 1 50 Z .REP 100
```

上記のステートメントにより、“ クロック ” は時刻ゼロで、“Supply 0”で、時刻10usで“Supply-1”になります。さらに、50usで “ ハイインピーダンス ” になり、100usで“Supply 0”、110usで“Supply-1”、150usで “ ハイインピーダンス ” ...となります。

例：

```
.CLK VDD 0 SU1
```

上記のステートメントにより、時刻ゼロで“Supply 1”になりそのままずっと“Supply 1”が保持されます。

« ホール » のあるクロックの構築

基本的には単純なクロックでありながら、信号が１または０のままの« 休止 »時間のあるデジタルパターンが必要な場合があります。このパターンを作成する一つの方法は、ANDゲートのようなデジタルゲートを使用する方法です。すなわち、ベースクロック（.CLKステートメントで得られる）を片方の入力に接続し、もう片方をバリデーション信号に接続します。バリデーション信号は.CLKステートメントまたはWAVEFORM節（次のセクションを参照して下さい。）により生成します。ただし、この回路に属していない回路内にゲートを含んでおかなければなりません。また、アドバンストスケマティックからのネットリストを含んでいる場合、これはスケマチックの一部ではなくなります。

このようなクロック信号を作成する他の方法は、2つの.CLKステートメントを使用して同じノード駆動することです。一つは通常のベースクロックでもう一つはバリデーションパターンを出力します。.CLKステートメントで同じノードを駆動する場合に未知の信号が作成されることを回避するには、ベース・クロックにはSTRONGストレングス（デフォルトのSUPPLYストレングスではありません。）の信号だけを使用し、バリデーションクロックは、SUPPLYストレングスかHIGH-IMPEDANCEストレングスのどちらかの信号を使用して下さい。このように二つの信号を組み合わせる場合、結果はベースクロックかバリデーションパターンになります。生成される信号はStrongストレングスだけです。このために、問題が発生する場合も有り得ることに注意して下さい。

例：

```
.CLK myclock 0 ST0 100 ST1 .REP 200
.CLK myclock 0 HIZ 1000 SU1 1500 HIZ 30000 SU0 30500 HIZ .REP
100000
```

これら2つのステートメントにより、ノードmyclockにホールのあるクロックが出力されます。myclockは、純粋なデジタルノードであることを忘れないで下さい。インタフェースノードやアナログノードではありません。時刻1000までは最初の.CLKが<<勝ちます>>。したがってmyclockはベースクロックになります。時刻1000で、2番目の.CLKが勝ちます。従ってSupplyストレングスはStrongストレングスよりも強くなります。これにより、myclockはSupplyストレングスを持つ論理1になります。時刻1500で２番目の.CLKが再び切り離され、myclockは再びベースクロックになります。時刻30000から時刻30500までは、myclockは論理0になります。時刻30500から時刻10000までは、myclockはベースクロックです。2番目の.CLKは周期パターンなので、時刻10000でバリデーションパターンが再スタートします。

WAVEFORM ステートメントの記述

WAVEFORMステートメントは任意のパターンを定義するために使用されます。スカラ信号と同様にバス信号のパターンを定義することができます。

スカラ信号の構文

スカラ信号の構文は以下のように記述します。：

```
WAVEFORM wavename
0    {signal{, signal} = value} ;
[+]time {signal{, signal} = value} ;
...
FINISH.
```

“WAVEFORM”キーワードにより、セクションを始め、“FINISH”キーワードによりWAVEFORM セクションを終了します。wavenameはWAVEFORMセクションの名前です。signalはデジタルノードの名前を表します。valueには、0, 1, X, Zの4種類の値のうちの一つを記述します。

WAVEFORMセクションの各行は、仮想時間単位で表される時刻値で始まります。時刻の値が相対値の場合、この値には'+'文字のプリフィックスをつけることができます。（以下のセクション“相対時刻の表記”を参照して下さい。）

時刻の値は、昇順で記述されなければなりません。

WAVEFORMセクションの各行には、指定された信号に発生するトランジションを記述します。waveform内の各行は、セミコロンで終わります。

WAVEFORMセクションの各行には、指定された信号に発生するトランジションを記述します。waveform内の各行は、セミコロンで終わります。

同じ時刻点で多くのトランジションがある場合、改行して記述することができます。（同じ時刻の値を各行の最初に記述して下さい。

複数の信号で同じトランジションが同じ時刻にある場合、それらをリストとしてグループ化することができます。

➔ アナログノードまたはインタフェースノードをWAVEFORMステートメントで駆動することはできません。ノードは純粋にデジタルノードでなくてはなりません。デジタルスティミラスでアナログノードやインタフェースノードを駆動する必要がある場合、入力がWAVEFORMステートメントによって駆動され出力がインタフェースノードを駆動するbufゲートのような補助的なデジタルゲートを挿入して下さい。

➔ 重要：デジタルベクトルを記述するために使用される時間やデュレーションはすべて仮想時間の単位で記述する必要があります。（仮想時間と実際の時間との間の対応は、.LTIMESCALEディレクティブで与えられます。）

例：

```
10 A,B,C=0;
is allowed instead of :
10 A=0 B=0 C=0;
```

例：

```
WAVEFORM wavename
0 a,b,c = 0 e,f,g,h,y = 1;
// comment lines must start with a //
10 a=1 b=1 e,f = 0 g=X ;
10 h=1 y=1 ;
11 a=1 b=0 e,f = 0 g=X ;
15 a=X b=1 e,f = 1 g=X ;
// repeating "g=X" on the two previous
//lines is useless
20 a=0 g=Z;
100 b,f,g = 0;
FINISH.
```

➔ スカラ信号はレベル0, Xまたは1のSupplyストレングスの信号として指定するか、ハイインピーダンス（Z）の信号として指定する方法しかありません。

バス信号の構文

WAVEFORMステートメントを使用すると、前の例のような単一（スカラ）信号やバスのトランジションを指定することができます。WAVEFORMの中でバスのトランジションは以下のように記述します。：

```
[+]timevalue bus[i:j] = BIN|HEX|DEC value;
```

指定される値は、バイナリ文字列（0, 1, X, Z）または16進数、符号なしの10進数である必要があります。XやZシンボルを使用してHEXまたはDECフォーマットのバスの全ビットをXまたは Zにセットすることもできます。

➔ バス幅は32ビットまでです。

Bus[j] は、バイナリ値の最下位ビットに対応します。

バイナリ文字列の場合、文字列の長さはバスの幅と一致する必要があります。

例：

```
100 CLK = 1, ADR[7:0] = HEX 4B;
+10 ADR[7:0] = DEC 123, NRST = 0;
```

```
+10 ADR[7:0] = BIN 01011101, NRST = 1, QOUT = Z;
+10 DATA[15:0] = HEX EF3B;
+10 DATA[15:0] = HEX X;          // all bits DATA[15:0] go X
+10 DATA[7:0] = DEC 255;
+10 DATA[7:0] = DEC Z;           // all bits of DATA[7:0] go Z
```

➔ BIN, DEC, HEXキーワードとバスの値との間にはスペースが必要です。“100 A[3:0] = BIN0101”と書くのは間違いです。“A[3:0] = BIN 0101”が正しい書き方です。

相対時刻の表記

WAVEFORMステートメントの内部では、前の例のように絶対時刻を指定するか、相対時刻を指定することができます。

行の最初のワードは以下の形式になります。：

```
+time
```

ここでtimeは正の整数（行の最初の文字は+でなくてはなりません。）です。すなわち、その行に記述されるトランジションは、前のトランジションの“time”時間後に起こります。

例えば、以下の２つのWAVEFORMは同じ意味になります：

```
WAVEFORM pulse_abs
* Absolute times style:
0 CK=0,A=1;
10 CK=1;
20 CK=0,A=0;
50 CK=1;
80 CK=0;
FINISH.

WAVEFORM pulse_rel
* Relative times style:
0 CK=0,A=1;
+10 CK=1;
+10 CK=0,A=0;
+30 CK=1;
+30 CK=0;
FINISH.
```

➔ 重要：相対時間表記を使用する場合、+と時間の間にはスペースを開けないで下さい。

この相対時間表記は、擬似ランダムパターンの長いwaveformを記述する場合やループ内のwaveformを記述する場合に特に有効です。

ループ

WAVEFORMステートメントの中で、サブセクションをN回記述する代わりに“ このサブセクションをN回繰り返したい ”と思うことがあると思います。ループによりこれを行うことができます。

与えられたパターンを何回か繰り返す場合、ループを使用してパターンファイルをより読みやすくすることができます。ループの一般的な形式を以下に示します。

```
LOOP N
+deltat1 ... ;
+deltat2 ... ;
...
ENDLOOP
```

このようなループセクションをWAVEFORMセクションの外に記述することはできません。“WAVEFORM”キーワードと“FINISH”キーワードの間に記述して下さい。同じWAVEFORM内に幾つかの別のループを記述することができます。

➔ ループ内でのループの宣言はサポートされていません。

ループの中で記述されたパターンはN回繰り返されます。ここでは相対時刻表記を使用する必要があります。

例：

```
WAVEFORM foolish
// some "normal" patterns:
0 CK,A,B=0;
500 A=0;
// and now a loop:
LOOP 10
+10 CK=0,A=1,B=0;
+10 CK=1;
+10 CK=0,A=0,B=1;
+10 CK=1;
ENDLOOP
// here current time is 900 ...
+1000 CK=0;
// this last transition occurs at time 1900.
FINISH.
```

このループにより、20個のCKパルスが10回生成されます。ループを使わずに記述する場合、パターンファイルが40行以上になります。CKの最初の立ち上がりエッジは、時刻500+10+10=520で起こります。

階層記述

アナログモード、デジタルモード、混在モード

回路の記述を階層構造にするには、サブブロックを定義したり、これらのサブブロックをインスタンス化することができる機能が必要です。

アナログデザインでは、Spiceに.SUBCKT機能があります。デジタルデザインでは、Verilog-HDLにmodule機能があります。アナログとデジタル回路混在のデザインでは、これら2つをミックスして使用できます。

この章では、以下のことを説明します。

- 純粋なアナログ階層記述での使用が仮定されている.SUBCKT機能を見直します。
- 純粋なデジタル階層記述での使用が仮定されているmodule機能について見直します。
- どのように.SUBCKTをデジタルプリミティブに記述し、デジタルとアナログ混在の階層記述を扱うのかを示します。
- SPICEとVerilog-HDLの形式では回路のトップレベルを定義するために使用されていたルールについて再度説明します。

SPICE 形式のサブブロック(.SUBCKT)

SPICEでは、.SUBCKT機能を使用してネットリストの階層構造を扱います。オリジナルのSPICEでは、サブサーキットをパラメータ化することはできませんでした。アドバンストSimでは、.SUBCKT構文の機能を拡張してパラメータを渡すことできるようにしています。

最初にパラメータを渡さない単純な.SUBCKT構文について説明します。それからサブサーキットにパラメータを渡す2つの方法について説明します。

単純な.SUBCKT の定義

```
.SUBCKT typename pin1 pin2 ...
  element
  element
  ...
.ENDS [typename]
```

サブサーキットの定義は.SUBCKTキーワードで始まり、これに続いて回路タイプ名（最大8文字）、さらにピン名リストが続きます。

サブサーキット定義には、その内部の要素の記述が必要です。これらの要素には、抵抗やコンデンサ、MOS、ソースなどのプリミティブか、その他のサブサーキットのインスタンス、ローカルモデル（.MODELステートメント）が可能です。

サブサーキットの定義は.ENDSキーワードで終わらなくてはなりません。これに続いてサブサーキット名を記述します。（これは記述の必要はありませんが、なるべく記述することをお薦めします。）

単純な.SUBCKT のコール

サブサーキットはXステートメントによりコールされます。Xは予約されているプリフィックスでサブサーキットのコールを表します。

Xステートメントの構文を以下に示します。：

```
Xinst node1 node2 ... typename
```

インスタンス名の最初の文字はXです。例では、“Xinst”になっています。

サブサーキットのタイプ名は、その行の最後に記述して下さい。

このサブサーキットはnode1とnode2との間に接続されます。node1はpin1に対応し、node2はpin2に対応します。

パラメータ化された .SUBCKT の定義

サブサーキットにパラメータを渡す方法は2種類あります。

サブサーキットの定義は« ¥ »構文を使用してパラメータ化されます。

```
.SUBCKT typename pin1 pin2 ... \ par1name par2name ...
  element
  element
  ...
.ENDS [typename]
```

パラメータ化されたサブサーキットの定義は通常のタイプ名、ピンリストに続き、バックスラッシュ\文字、さらにサブサーキットの本体で使用するパラメータ名のリストを記述します。

アドバンストSimでは、\文字の前後には、1個以上のスペースが必要です。

このようにパラメータ化されたサブサーキットのインスタンスには、パラメータpar1name, par2nameが渡されます。

一方、以下のように« PARAMS : »構文を使用してサブサーキットをパラメータ化することもできます。

```
.SUBCKT typename pin1 pin2 ... PARAMS: par1name=par1,
par2name=par2 ...
  element
  element
  ...
.ENDS [typename]
```

このようにパラメータ化されたサブサーキットの定義は、通常のタイプ名、ピンリストに続き、PARAMSキーワード、さらにサブサーキットの本体で使用するパラメータ名のリストとデフォルト値を記述します。

« ¥ » 構文を使用するパラメータ化された.SUBCKT のコール

コールするためには、Xで始まるインスタンス名とノード名、\文字に続いてインスタンスパラメータ値、最後にサブサーキットタイプ名が必要です。

```
Xinst node1 node2 ... ¥ p1 p2 ... typename
```

ここでp1はパラメータpar1nameに対応し、p2はpar2nameに対応します。

（.SUBCKTの行で）宣言されるパラメータはすべて値としてサブサーキットに渡され、順番が一致している必要があります。この構文では、« PARAMS: »構文の場合と違い、パラメータのデフォルト値を指定する方法はありません。

サブサーキットの定義では、パラメータは数値、ノード、サブサーキット名の場所で使用できます。

例：

```
.SUBCKT RCNET X Y \ RVAL CVAL WN
R1 X Y RVAL
R2 Y Z RVAL
C1 Y 0 CVAL
C2 Z 0 CVAL
M1 X Y Z 0 MOSN W= WN L=2U
.ENDS RCNET

* サブ・サーキットRCNETには、2つのピンXとYがあり、3つパラメータ
* RVAL, CVAL, and WN.  があります。

* in circuit.NSX:
X_RC1 A B \ 100K 10P 15U RCNET
X_RC2 B C \ 125K 9.7P 27U RCNET
```

上記例のように数値パラメータでサブサーキットをパラメータ化するためにこの構文を使用することができます。また、トポロジーそのものをパラメータ化することもできます。これにより、以下の例で示すようにより複雑な階層化が可能になります。

例：

```
.SUBCKT RCNET X Y \ RVAL CVAL WN INST NODE VALUE
R1 X Y RVAL
INST Y NODE VALUE
C1 Y 0 CVAL
C2 Z 0 CVAL
M1 X Y Z 0 MOSN W= WN L=2U
.ENDS RCNET

/* サブサーキットRCNETには、2つのピンX, Yと6つのパラメータRVAL, CVAL,
WN, NODE, VALUEがあります。RVAL, CVAL, WN, VALUEはコールされた時に数値
を受け取り、INSTはインスタンス名でノードはノード名を受け取ります。
*/

// in circuit.NSX:
XRC1 A B \ 100K 10P 15U R2 Z 100K RCNET
XRC2 B C \ 125K 9.7P 27U C3 X 10P RCNET

/* XRC1にはR2_XRC1という名前の100Kの抵抗があり、ノードBとXRC1_Zとの間に接
続されます。XRC2には、C3_XRC1という名前の10Pのコンデンサがあり、ノードBとA
との間に接続されます。
*/
```

« PARAMS: » 構文を使用するパラメータ化した.SUBCKTの呼び出し

« PARAMS: » 構文を使用するパラメータ化された .SUBCKTの呼び出すには、Xで始まるインスタンス名とノード名、サブサーキットタイプ名、さらにPARAMS:キーワードに続きカンマで区切られた« paramname=paramvalue »文字列のリストが必要です。

```
Xinst node1 node2 ... typename PARAMS: par1name=p1,
par3name=par3 ...
```

ここでp1はパラメータpar1nameに対応し、p2はpar2name に対応します。

（ .SUBCKT 行で）宣言された全てのパラメータを渡さす必要はありません。受け渡されないパラメータは .SUBCKT定義で宣言されたデフォルト値になります。

パラメータが渡されれば、その値はデフォルト値を無効にし、渡されない場合は、デフォルト値が用いられることになります。

このサブサーキットの定義では、パラメータは数値の替わりとして表われます。

例：

```
.SUBCKT RCNET X Y PARAMS: RVAL=100K CVAL=10P WN=3U
R1 X Y RVAL
R2 Y Z RVAL
C1 Y 0 CVAL
C2 Z 0 CVAL
M1 X Y Z 0 MOSN W=WN L=2U
.ENDS RCNET

/*
サブサーキットRCNETは2つのピン X と Yを持ち、3つのパラメータ  RVAL, CVAL,
WNがあります。
*/

// in circuit.NSX:
XRC1 A B RCNET PARAMS: RVAL=150K CVAL=12P WN=15U
```

```
XRC2 B C RCNET PARAMS: CVAL=9.7P
```

/*
インスタンスXRC1では、3つのパラメータがすべて渡されますが、インスタンスXRC2では、CVALの値がオーバーライドされます。RVALとWNはデフォルト値に設定されます。
*/

階層名

.SUBCKTのそれぞれのインスタンスでは、階層名を持つ内部ノードを作成することができます。.SUBCKTの内部のノードがピンリストの一部でない場合と、そのノードがグローバルノードでない場合、内部ノードが作成されます。ノード“0”（文字ゼロ）とグランドノードは常にグローバルです。任意の.SUBCKTで使用できます。パターンファイルに.GLOBALディレクティブを使用してグローバルノードを定義すれば、その他のアナロググローバルノードを使用できます。階層ノード名とインスタンス名は、階層文字を使用して作成します。この階層文字は、デフォルトでは‘_’アンダスコアです。この文字は、パターンファイルで.HIERCHARディレクティブを使用すれば変更することができます。

➔ 階層インスタンス名は、64文字までです。回路の階層が深くなるとインスタンス名が長くなりエラーになることがあります。

要求項目

- Xステートメント内のノードの数は、.SUBCKT定義行で宣言されるピンの数と正確に同じである必要があります。
- « \ »構文の場合、Xステートメントのパラメータの数は、.SUBCKT定義行で宣言されるパラメータの数と正確に同じである必要があります。
- サブサーキットの中で他のサブサーキットを定義することはできません。また、サブサーキットの中でサブサーキット自信を参照することもできません。

.SUBCKT 内のモデル

.MODELステートメントを利用してサブサーキット内でデバイスモデルを宣言する場合、このモデルはサブサーキット内でローカルになります。すなわち、デバイスとしてサブサーキットの外で使用することはできません。

➔ *circuit*.NSX（サブサーキット定義の外）または、*circuit*.PATの最上位レベル定義されるモデルは任意のサブサーキットで使用できます。

サブサーキットのライブラリ化

.SUBCKTをライブラリ化するには、定義行は、サブサーキット名に.CKTという拡張子がついた名前のファイルに保存する必要があります。（前の例では、RCNET.CKTとなります。）

.CKT拡張子は変更できません。

ライブラリファイルには、最初にコメント行、それから“.SUBCKT …”から“.ENDS …”までの定義文があります。

例：

```
------------------------------ File: RCNET.CKT ---
* Cell: RCNET
* Created: xx-xx-xx
* By: joe
* Modified: yy-yy-yy
* By: no one knows...
.SUBCKT RCNET X Y \ RVAL CVAL WN
R1 X Y RVAL
R2 Y Z RVAL
C1 Y 0 CVAL
C2 Z 0 CVAL
M1 X Y Z 0 MOSN W= WN L=2U
```

```
.ENDS RCNET
------------------------------------------------
```

“.LIB”ファイルを使用したい場合、これらの行を拡張子.LIBファイルに保存することもできます。ただし、定義の方法が違ってきます。

➔ *ライブラリとデバイスモデル*の章を参照して下さい。

Verilog-HDL 形式のサブブロック(モジュール)

moduleは、デジタルのSUBCKTと考えて構いません。すなわち、moduleを利用して回路内でデジタルの階層ブロックを作成します。Spiceの.SUBCKTと同様に、moduleにパラメータを渡す機能があります。まず、簡単なmoduleの構文について説明しそれからパラメータ化されたmoduleについて説明します。

Verilog-HDLのモジュールは、非常に複雑な記述も可能です。このセクションでは、階層構造を定義するために必要最小限の情報をVerilog-HDLのモジュールコンセプトを見ながら説明していきます。詳細な説明については、LRMを参照して下さい。

単純なモジュールの定義

基本的なモジュールの定義構文を以下に示します：

```
module typename (port1, port2, ...) ;
  input|output|inout [range] port1, port2, ... ;
  [wire [range] int1, int2, ... ;]
  element;
  element;
  ...;
endmodule
```

moduleはサブブロックの定義を識別するためのキーワードです。このキーワードに続いて、ライブラリファイルにモジュールが保存される場合8文字までのサブブロック名が記述されます。

タイプ名に続き"(",")"で囲まれるモジュールのポート名のリストが記述されます。module行はセミコロンで終わります。

モジュールの定義は、endmoduleキーワードで終わります。（セミコロンではありません。）

ポートの極性はすべてモジュール定義の内部で宣言されます。モジュールポートの極性は、input（入力）とoutput（出力）、inout（入出力）となります。これらの極性の宣言は、通常モジュール定義の一番最初に記述されます。ポートの極性の宣言は、ポートのリストの順と同じである必要はありません。

極性の宣言のすぐ後に、オプションでレンジの指定子を指定することができます。レンジ指定子により、ポートのサイズが定義されます。レンジ指定子の構文は、[n:m]です。ここでnとmは整数です。例えば[7:0]のようにレンジを指定することは、対応するポートが8-bit幅のバスであることを意味します。レンジの指定が無い場合、ポートはスカラ接続（1bit幅）になります。« バス表記の使用 »のセクションの例文を参照して下さい。

module行とendmodule行との間に、サブブロックを形成する要素を記述します。これらの要素は通常、ネット宣言またはデジタルプリミティブ、その他のモジュールのインスタンス、ユーザ定義のプリミティブです。

ポートリストは空でもかまいません。この場合、モジュールはポートを持たないことになります。モジュールがポートを持たない場合、モジュール行は、module typenameのようになります。

ポートがバスの場合、そのサイズはレンジ指定子により極性の定義部で宣言されます。

例：

```
module adder (a, b, s, cin, cout);
// input/output 宣言は必ず必要です。
  input [7:0] a, b;
  input cin
  output [7:0] s;
  output cout;
```

```
// wire 宣言はオプションです。
// wire はネットのデフォルト・タイプです。
  wire [7:0] c;
// ここからモジュール内のエレメントです。
// fulladderという名前の他のモジュールのインスタンスです。
  ...;
endmodule
```

単純なモジュールのインスタンスの接続

モジュール内で他のモジュールのインスタンスを接続することができます。moduleのインスタンスを接続するには、以下の構文を使用します。

```
typename instname ( node1, node2, ... );
```

moduleのタイプ名を最初に記述し、それからインスタンス名、"("、")"で囲まれた接続ノード（またはネット）のリストを記述します。この行はセミコロンで終えます。

アドバンストSimでインスタンスのあるネットリストをうまくロードするには、モジュール定義を.NSXファイルまたは.V拡張子のライブラリファイル、.LIB拡張子のライブラリファイルのどこかでする必要があります。

ノードのリストが長すぎて一行に書ききれない場合、何行かに分けることができます。（行のつづきを表す演算子は必要ありません。）

ポートは接続しなくてもかまいません。下の例のようにカンマは必要です。

例：

```
typename instname( node1,,node3 );
```

この例では、モジュールtypenameのインスタンスinstnameの最初のポートはnode1に接続され、2番目のノードは未接続、3番目のポートはnode3に接続されます。

パラメータ化されたモジュールの定義

パラメータ化されたmoduleの定義を以下に示します。：

```
module typename (port1, port2, ...) ;
  parameter param1 = defvalue1 ;
  parameter param2 = defvalue2 ;
  input|output|inout [range] port1, port2, ... ;
  [wire [range] int1, int2, ... ;]
  ...
  element;
  element;
  ...;
endmodule
```

ノーマルのmoduleの構文と比較した場合の違いは、モジュールの最初でパラメータリストが定義されることです。各パラメータはパラメータキーワードに続き、パラメータ名、イコールサイン、最後にパラメータのデフォルト値の順で定義されます。モジュールのインスタンスがオーバーライド値を指定していない場合、パラメータのデフォルト値はこのインスタンスの値が使用されます。パラメータは、モジュールの内部で数値として扱われます。

パラメータ化されたモジュールのインスタンスの接続

以下の構文を使用して、モジュールの中で、他のパラメータ化されたmoduleを接続することができます。

```
typename #(p1, p2,...) instname(node1, node2,...) ;
```

moduleのタイプ名を最初に記述します。その次に括弧で囲まれたパラメータ値のリスト、インスタンス名、最後に括弧で囲まれた接続ノードのリストの順で記述します。パラメータのリストは、最初に#文字をプリフィックスして下さい。

渡されるパラメータ（p1, p2等）の値は、デジタルプリミティブでは遅れ指示子と置き換えることができます。これらにより、モジュール内の信号のサイズを定義することができます。さらに、パラメータを下位レベルのmoduleに渡すこともできます。次の例を参照して下さい。

例：

```
module MYNANDS(A1, B1, Y1, A2, B2, Y2);
  parameter TP1 = 5;
  parameter TP2 = 7;
  parameter TP3 = 6;
  input A1, B1, A2, B2;
  output      Y1, Y2;
  nand #(TP1, 4.3) N1(Y1, A1, B1);
  nand #(TP2, TP3) N2(Y2, A2, B2);
  MYNORS #(TP2, TP3, 7.8, 9.6) M1(A1, B1, W1, A2, B2, W2);
endmodule
```

➔ この例では、MYNANDSはパラメータ化されたmoduleで３つのパラメータを持っています。これらのパラメータは、MYNANDSの中でnandゲートN1とN2の遅れとして使用されます。さらに、TP2とTP3はパラメータ化されたモジュール、ブロックMYNORSのインスタンスM1に渡されます。

例：

```
module adder (a, b, s, cin, cout);

// a, b, c はパラメータ化されたベクトルです。
  parameter N = 3;
  input [N:0] a, b;
  input cin;
  output [N:0] s;
  output cout;
  wire [N:0] c;
// ここからモジュールのエレメントが始まります。
// これはベクトル化されたインスタンスです。詳細はLRMを参照して下さい。
  fulladder f[0:N] (a, b, s, {cin,c[1:N]}, c);
endmodule

module top;
  wire [3:0] a, b, s;
  wire [7:0] s1, s2;
  wire cin, cout, cout2;
  adder #(3) ia1(a, b, s, cin, cout);
  adder #(7) ia2(a, a, b, b, s1, cin, cout2);
endmodule
```

要求項目

モジュールがインスタンス化され、ノードの名前だけが与えられる場合、moduleコールのノードの数はmodule定義行で宣言されるポートの数と同じでなくてはなりません。接続するノードのポート名を指定してノードとポートを正確に接続する方法もあります。この場合、ノードの順番とポートの順番とが一致する必要はありません。以下の例を参照して下さい：

例：

```
module MYNANDS(A1, B1, Y1, A2, B2, Y2);
  parameter TP1 = 3;
  parameter TP2 = 3;
  parameter TP3 = 6;
  input A1, B1, A2, B2;
  output      Y1, Y2;
  nand #(TP1, 4.3) N1(Y1, A1, B1);
  nand #(TP2, TP3) N2(Y2, A2, B2);
  MYNORS #(TP2, TP3, 7.8, 9.6) M1(A1, B1, W1, A2, B2, W2);
endmodule

module top;
  MYNANDS #(2, 3, 4) i0
(.A1(NET1), .A2(NET2), .B1(NET3), .B2(NET4), .Y1(NET5), .Y2(NET6)
```

```
);
endmodule
```

他のモジュール内で新しいmoduleを定義することはできません。

また、モジュール内で自分自身のmoduleを参照することもできません。

階層名

moduleのインスタンスでは、階層名を持つ内部のノードは作成することができます。module内部のノードが、ポートリストに無い場合、内部ノードが作成されます。階層化されたノード名とインスタンス名は、階層文字を使用して作成します。Verilogのデフォルトでは、階層文字は'.'（ピリオド）です。このデフォルトは、*advsim*.INIの[Defaults]セクションのDefault Hierarchy Characterという名前のエントリで変更することができます。与えられたシミュレーションで使用するための文字を変更するには、パターンファイルの.HIERCHARディレクティブを使用して下さい。

➔ 最終的な階層インスタンス名は64文字までです。回路の階層が深くなるとインスタンス名が長くなりエラーが発生するかもしれません。

バス表記の使用

modulesにバスを接続するには、以下の構文によりワイヤのセットを指定して下さい。: NAME[beg:end];ここでNAMEは識別子（バスの名前）、begとendは整数です。

例：

```
A[7:0]
SUM[3:15]
```

バスの各ビットは、NAME[i]と記述することにより参照できます。ここでi は、レンジ[beg, end]の間の整数です。

バスのビットの一部を参照するには、A[i:j]と記述します。ここで、iとjはレンジ[beg, end]の中の整数です。

例：

```
module REG4B(CLK, D, Q);
  input   [3:0] D;
  output  [0:3] Q;
  REG2B   R1(CLK,D[0:1],Q[0:1]);
  REG1B   R2(CLK,D[2],Q[2]);
  REG1B   R3(CLK,D[3],Q[3]);
endmodule
```

モジュールのライブラリ化

moduleをライブラリ化するには、module名に拡張子.V（前の例の場合、REG4B.V）をつけた名前のファイルmoduleを保存して下さい。

.V拡張子は、ライブラリエレメントとしてファイルを認識させるために必要です。

パラメータ化されたmoduleでも、そうでないmoduleでもライブラリ化できます。

ライブラリファイルには、最初にコメント行、それからmodule行からendmodule行までの定義文があります。

例：

```
------------------------------ File: REG4B.V -----
// Cell REG4B

module REG4B(CLK, D, Q);
  input   [3:0] D;
  output  [0:3] Q;
  REG2B   R1(CLK,D[0:1],Q[0:1]);
  REG1B   R2(CLK,D[2],Q[2]);
  REG1B   R3(CLK,D[3],Q[3]);
endmodule
-------------------------------------------------
```

➔ この定義を拡張子.LIBファイルに保存することもできます。ただし、定義の方法が違ってきます。*ライブラリとデバイスモデル*の章を参照して下さい。

デジタルアナログ混在形式の.SUBCKT

混在の記述を扱うには、混在のサブブロックを定義する必要があります。このサブブロックには、アナログエレメントとデジタルエレメントの両方が記述されます。例えば、DAC（デジタルアナログコンバーター）またはADC（アナログデジタルコンバーター）をサブブロックとして定義するには、レジスタやMOSトランジスタと同時にNORゲートやNANDゲートが必要です。更に、アナログとデジタルの両方のI/Oを持つサブブロックを定義する必要があります。このようなことは、アドバンストSimを使えば簡単にできます。

.SUBCKT定義には、アナログエレメント（アナログプリミティブやXステートメント）とデジタルエレメントを記述すつことができます。このような混在のサブサーキットを作成するために使用する構文をこのセクションで詳しく説明します。

デジタルアナログ混在形式の.SUBCKTの作成

Verilog-HDLのmoduleと同じように、.SUBCKTにデジタルゲートを記述することができます。

これを行なうには、構文切替指示子を設定してSPICE構文とVerilog-HDL構文とを切替える必要があります。これらの構文切替え指示子は>>> SPICE、>>> VERILOGです。

例：

```
>>> SPICE
.SUBCKT MYBLOCK A B C OUT
  R1 A B 10K
>>> VERILOG
  nand #(10, 10) N1 (OUT, A, B);
>>> SPICE
  C1 B C 1U
  X1 A B OUT COMPAR
>>> VERILOG
  nor #(12, 12) N2 (NX, A, B);
>>> SPICE
  M1 A B C VSS N W=10U L=2U
>>> VERILOG
  DIGBLOCK D1(A, B, NX);
>>> SPICE
  C2 B 0 1U
.ENDS MYBLOCK
```

この混在形式のサブサーキットには、アナログエレメント(R1, C1, M1)とデジタルエレメント(N1, N2)があります。また、COMPARという名前のブロックのコール(X1)とDIGBLOCKという名前のブロックへのコールがあります。COMPAR自身もアナログエレメントとデジタルエレメントがあります。

➔ .SUBCKT内のデジタルエレメントはmoduleで使用される形式で使用します。特に、セミコロン文字';'は、デジタルエレメント記述された行の最後には必ず必要です。

MYBLOCKのインスタンスは、SPICEの構文であるXステートメントで普通にコールされます。これらのコールは、トップレベルまたは .SUBCKT定義の内部に記述することができます。

MYBLOCKのコールの例：

```
XM X Y Z OUT1 MYBLOCK
```

前の例は多くの構文切替えスイッチが使用されていたため、少し不格好な記述です。これは単に、構文を切替える方法を示すためです。アナログエレメントとデジタルエレメントはより自然に分割され、スイッチはもっと少ない場合がほとんどです。下の書き直された例を参照して下さい。最初の>>> SPICEスイッチは必要ありません。.SUBCKTが処理される場合、デフォルトでSPICE構文になります。

書き直された例：

```
>>> SPICE

.SUBCKT MYBLOCK A B C OUT

// this first switch is optional...

>>> SPICE
  R1 A B 10K
  C1 B C 1U
  X1 A B OUT COMPAR
  M1 A B C VSS N W=10U L=2U
  C2 B 0 1U

>>> VERILOG
// 以下の行は、Verilog記述です。
  nor #(12, 12) N2 (NX, A, B);
  nand #(10, 10) N1 (OUT, A, B);
  DIGBLOCK D1(A, B, NX);

// >>> SPICEスイッチは.ENDSの前には必要ありません。
```

階層のトップレベル

どんな回路でも、トップレベルの階層があります。これは。階層構造の最上位レベルで、階層構造のルートになります。このトップレベルはSPICEとVerilog-HDLでは扱われ方が異なります。両者の方法には、短所もあり長所もあります。

SPICE互換のシミュレータ（アドバンストSimを含む）では、回路のトップレベルはコンポーネント（プリミティブやサブサーキットのインスタンス）のセットで定義されます。サブサーキットの定義は外側で記述されます。サブサーキットの定義（.SUBCKT ... .ENDS）だけを含むネットリストがある場合、それは実際には空の回路です。サブサーキットの定義がネットリストの中に記述されながらXステートメントでコールされない場合、ネットリストにサブサーキットが存在しなかった場合と同じことです。

例：

```
------------------------------ File: DUMMY.NSX -----
// Circuit netlist

// サブサーキットの定義
.SUBCKT RC IN OUT
R1 IN OUT 100K
C1 OUT 0 100P
.ENDS RC

// トップレベル
RIN INPUT N1 100K
X1 N1 N2 RC
COUT N2 0 100P
-----------------------------------------------------
```

Verilog-HDLの場合、この辺りが違います。階層ブロック（モジュール）を定義したりコールする方法は、SPICEの方法（.SUBCKT）とよく似ています。ただし、トップレベルは違います。Verilog-HDLではトップレベルは、ネットリスト内に現れるインスタンス化されないモジュールにより定義されます。これには、SPICEユーザが不思議に思う多くの議論があります。

1. モジュール定義の外には何も現れません。
2. モジュール定義をネットリストの内部に記述した場合、このモジュールをトップレベルにある他のモジュールから呼ぶことはできません。
3. トップレベルのネットは、トップレベルのモジュールのポートリストに記述されるポートにより定義されます。ネットがトップレベルのモジュール内で使用されるがポートリストにない場合、内部ネットして扱われます。その階層名はトップレベルモジュールの名前と階層文字‘.'、ネットの基本ネームから作られます。

以下にやってはいけない例を示します。その次に正しく書き直した例を示します。

例：

```
-------------------------------- File: BAD.NSX -----
>>> VERILOG

not n1(NCLK, CLK);
REG2B   R1(CLK,D[0:1],Q[0:1]);
REG1B   R2(CLK,D[2],Q[2]);
REG1B   R3(CLK,D[3],Q[3]);

// このファイルは間違いです。
// プリミティブやモジュールのインスタンスはモジュール定義の外で記述
// できません。このファイルはコンパイル時にエラーがでます。
-----------------------------------------------------
```

これは、やってはいけない例です。

```
-------------------------------- File: GOOD.NSX -----
>>> VERILOG

module REG4B(CLK, D, Q);
  input   [3:0] D;
  output  [0:3] Q;
  input   CLK
  wire    NCLK;
  not     n1(NCLK, CLK);
  REG2B   R1(CLK,D[0:1],Q[0:1]);
  REG1B   R2(CLK,D[2],Q[2]);
  REG1B   R3(CLK,D[3],Q[3]);
endmodule

/*
REG2BとREG1Bモジュール定義はライブラリファイルに保存されています。このファイル正しいです。
トップレベルはREG4Bモジュールにより定義されます。この定義はインスタンス化されません。トップのレベルのネットは、CLK, D[3], D[2], D[1], D[0] , Q[3], Q[2], Q[1], Q[0]です。
ネットREG4B.NCLKは内部ネットです。
SPICEユーザーからすると、ファイルGOOD.NSXに何かが足らないように思われるかもしれません。すなわち、トップレベルを定義するためのREG4Bモジュールのコール。
*/
-----------------------------------------------------
```

マクロ

概要

➔ この章では、WAVEFORMステートメントについてすでにわかっているものとして説明をします。まだWAVEFORMステートメントについて分からない場合、この章に入る前に回路プリミティブとスティミラスのデジタルスティミラスを参照して下さい。

マクロは複雑なデジタルパターンの記述を単純にするための機能です。機能のシミュレーションやパターンが長く、繰り返しがあり複雑なテストモードのシミュレーションに有効です。マクロを使用すれば、パターンのパラメータ化されたブロック（またはシーケンス）を定義したりインスタンス化することができます。

➔ C言語を知っていれば、マクロ機能はCの#defineディレクティブに非常に近い機能であることがわかると思います。

➔ アドバンストSimでは、マクロのこの方法とVerilog-HDLのマクロ機能をサポートしています。シリアルインタフェースのパターンをインプリメントしたい場合、strbin()関数の機能を利用できるため、（この章で説明する）アドバンストSimのマクロ機能を使用したくなると思います。

例

マクロの有効性を説明する最もよい方法は、簡単なテストケースの例題を扱う事です。8ビットのシフトレジスタに任意の8ビットの値を与えるようなパターンを記述する場合を思い浮かべて下さい（8ビットのシフトレジスタを介してRAMに値を与えるような例）。WAVEFORMセクションを使用して16進数の$A5(10100101)を記述する場合、以下のようになります。

```
WAVEFORM loadreg
...
+100 LOAD=1;
+100 QIN=1;
+100 QIN=0;
+100 QIN=1;
+100 QIN=0;
+100 QIN=0;
+100 QIN=1;
+100 QIN=0;
+100 QIN=1;
+100 LOAD = 0;
...
FINISH.
```

これでも動作しますが、使用されている記述方法にはいくつかの欠点があります。

まず、パターンファイルの10行を使用します。多くはありませんが、レジスタを128回ロードする場合、非常に長いパターンファイルができてしまいます。2番目にパラメータ化が何もされていません。クロック周期、シリアル入力ピンの名前（QIN）、ロードされる値すべてがパラメータ化されていません。3番目に、“$A5”という16進数が見た目にわかりません。なぜなら、$A5という値を、クロック周期ごとに各ビット単位で与える必要があるからです。

マクロを使えばこれらの欠点は無くなります。

パターンファイルでロードする値を一行で指定するようにします。マクロを定義しましょう。

```
DEFINE_MACRO mloadreg( Q7, Q6, Q5, Q4, Q3, Q2, Q1, Q0 )
+100 LOAD=1;
+100 QIN = Q7;
+100 QIN = Q6;
+100 QIN = Q5;
+100 QIN = Q4;
```

```
+100 QIN = Q3;
+100 QIN = Q2;
+100 QIN = Q1;
+100 QIN = Q0;
+100 LOAD = 0;
END_MACRO
```

“mloadreg”マクロが上記のように定義されれば、EXPAND_MACROステートメントを使用することができます。：

```
WAVEFORM load
...
EXPAND_MACRO mloadreg(1, 0, 1, 0, 0, 1, 0, 1);
...
FINISH
```

このEXPAND_MACROステートメントにより、前のWAVEFORM セクションで10行で表わされた処理と同じことができます。ワードをロードするには、パターンファイルでは１行を記述するだけです。$A5のあと$B3、その後$0Fをロードしたい場合、以下のようになります。：

```
WAVEFORM load
...
EXPAND_MACRO mloadreg(1, 0, 1, 0, 0, 1, 0, 1);
EXPAND_MACRO mloadreg(1, 0, 1, 1, 0, 0, 1, 1);
EXPAND_MACRO mloadreg(0, 0, 0, 0, 1, 1, 1, 1);
...
FINISH
```

今まで、ビットの値Q7からQ0までだけをパラメータ化しました。いろいろなシリアルインタフェースの場合、同様に入力ピンの名前をパラメータ化することができます。：

```
DEFINE_MACRO mloadreg( QIN, Q7, Q6, Q5, Q4, Q3, Q2, Q1, Q0 )
+100 LOAD=1;
+100 QIN = Q7;
+100 QIN = Q6;
+100 QIN = Q5;
+100 QIN = Q4;
+100 QIN = Q3;
+100 QIN = Q2;
+100 QIN = Q1;
+100 QIN = Q0;
+100 LOAD = 0;
END_MACRO
```

対応するマクロコールは以下のようになります。：

```
WAVEFORM load
...
EXPAND_MACRO mloadreg(QA, 1, 0, 1, 0, 0, 1, 0, 1);
EXPAND_MACRO mloadreg(QB, 1, 0, 1, 1, 0, 0, 1, 1);
EXPAND_MACRO mloadreg(QC, 0, 0, 0, 0, 1, 1, 1, 1);
...
FINISH
```

だいぶ良くなりましたが、16進数が表現されていません。従って、EXPAND_MACROステートメントに一連のビットの列が渡されています。

strbin()関数を使用して16進数を表現します。$A5という値をロードするには、以下のようにコールします。：

```
WAVEFORM load
...
EXPAND_MACRO mloadreg(QA, strbin($A5, 8));
...
FINISH
...
```

以下の記述の代わりに上記のように表現します：

```
EXPAND_MACRO mloadreg(QA, 1, 0, 1, 0, 0, 1, 0, 1);
...
FINISH
```

このステートメント“strbin($A5, 8)”は、マクロに展開される前に“1 0 1 0 0 1 0 1”という数列に置き換えられます。‘A5’のプリフィックス‘$’はA5が16進数の値であることを表わします。‘8’パラメータは$A5を8ビットでコード化することを表わします。

同じレジスタに幾つかの値をロードする前のケースに戻ります。strbin()を使用して以下のように書き換えることができます。：

```
WAVEFORM load
...
EXPAND_MACRO mloadreg(QA, strbin($A5, 8));
EXPAND_MACRO mloadreg(QA, strbin($B3, 8));
EXPAND_MACRO mloadreg(QA, strbin($0F, 8));
...
FINISH.
```

マクロ構文の正式な定義

マクロ機能では、DEFINE_MACRO, END_MACRO, EXPAND_MACROという３つのキーワードを使用します。

マクロの定義はDEFINE_MACROキーワードで始まりEND_MACROキーワードで終わります。

EXPAND_MACROキーワードにより、割り付けられた実際の値で置き換えられたマクロ定義のパラメータを持つマクロに展開されます。

マクロの定義

マクロ定義の構文を以下に示します。：

```
DEFINE_MACRO mname(formal_param_list)
...
...
END_MACRO
```

この定義は、パターンファイルのどこに書いてもかまいません。通常最初に記述します。また、mname.MACと呼ばれるファイルやライブラリにに保存することもできます。mname識別子はマクロの名前で、最大8文字までです。

置き換えられる物

formal_param_listは識別子（パラメータ）のリストで、ブランクまたはカンマで区切られます。これらのパラメータは定義内の行で使用されます。展開時に置き換えられるためには、パラメータは“語”である必要があります。語の区切り文字を以下に示します。

カンマ	,
括弧	(
括弧	)
イコール	=
セミコロン	;

例えば：

```
DEFINE_MACRO load(Q3, Q2, Q1, Q0)
+100 QIN=Q3;
+100 NQ3=Q2;
+100 QIN =Q1;
END_MACRO
```

パラメータリストは、“Q3, Q2, Q1, Q0”です。パラメータQ3, Q2, Q1は定義のステートメントの中で語として現れます。これらはシミュレーション時に置き換えられます。ただし、NQ3は、“Q3”が含まれていますが展開時にNQ3のままです。

➔ あまり薦めませんが、定義の中で使用されないパラメータをパラメータリストに記述することもできます。例えば、前の例ではQ0は定義の中で使用されていません。

➔ また、パラメータリストに何も記述しないこともできます。

バス表記の使用

バス表記をパラメータリストに記述して記述をよりコンパクトにすることができます。以下のように書くことができます。

```
DEFINE_MACRO load(Q[3:0])
+100 QIN=Q[3];
+100 NQ3=Q[2];
+100 QIN =Q[1];
END_MACRO
```

バス表記をしなければ以下のようになります。

```
DEFINE_MACRO load(Q[3], Q[2], Q[13], Q[0])
+100 QIN=Q[3];
+100 NQ3=Q[2];
+100 QIN =Q[1];
END_MACRO
```

マクロの展開

マクロを使用（展開）する構文は：

```
EXPAND_MACRO mname(actual_param_list);
```

mnameはあらかじめ定義されたマクロを参照するための識別子でパターンファイルまたはライブラリに保存されているmname.MACファイルにあります。

actual_param_listは、ブランクまたはカンマで区切られる実際のパラメータのリストです。これらの実際のパラメータはマクロが展開されると定義の中のフォーマルパラメータに置き換わります。実際のパラメータの数と順番は、フォーマルパラメータの数と順番とに一致している必要があります。

➔ 実際のパラメータに何も指定しないこともできますが、この場合、定義のフォーマルパラメータも何も指定されていない必要があります。

EXPAND_MACROステートメントは、パターンファイルで一行で記述される必要があります。マクロのパラメータが多すぎる場合、バス表記することを考えて下さい。

EXPAND_MACROステートメントは通常WAVEFORMセクションの内部に記述します（これは必須ではありません）。EXPAND_MACROステートメントは、この例のように純粋なステートメント自由に混在することことができます。

```
WAVEFORM testram
0 NRST=0;
+100 NRST=1;
+100 RWBAR=0;
+100 ADR[7:0]=DEC 0;
EXPAND_MACRO loadvalue(1, 0, 1, 0);
+100 ADR[7:0]=DEC 1;
EXPAND_MACRO loadvalue(0, 1, 0, 1);
+100 ADR[7:0]=DEC 2;
...
+100 RWBAR=1;
+100 ADR[7:0]=DEC 0;
EXPAND_MACRO readvalue();
+100 ADR[7:0]=DEC 1;
EXPAND_MACRO readvalue();
+100 ADR[7:0]=DEC 2;
FINISH
```

strbin() 関数

strbin()関数を使用して、10進数または16進数の値を等価なバイナリ値に変換します。この関数は、EXPAND_MACROステートメントの実際のパラメータリストで使用できます。

strbin()関数には2つのパラメータが必要です。すなわち、変換される値と変換に使用するビットの数です。値が16進数の場合、'$'サインをプリフィックスとして使用して下さい。

例：

```
strbin(12, 8)       0 1 0 0 1 0 0 0
strbin($AA, 8)      1 0 1 0 1 0 1 0
strbin($BA, 5)      1 1 0 1 0
```

strbin()があるとマクロを展開する前に、ブランクで区切られた0と1のリストで置き換えられます。実際のパラメータにstrbin(value, n)がある場合、実際のパラメータの数はn個になります。

例：

```
EXPAND_MACRO loadreg(X, Z, strbin(5, 3), X);
```

は、以下と同じです。：

```
EXPAND_MACRO loadreg(X, Z, 1 0 1, X);
```

階層マクロ

EXPAND_MACROステートメントは通常WAVEFORMセクションに記述します。EXPAND_MACRO行は定義の行と置き換えられます。

EXPAND_MACROステートメントを他のマクロ定義の中で使用することが有効な場合があります。

➔ ネストレベルは小さくするほうがいいです。パターンが読みにくくなります。

例：

```
DEFINE_MACRO loadfifo(Q3[3:0], Q2[3:0], Q1[3:0], Q0[3:0])
+100 INITFIFO=1;
EXPAND_MACRO loadreg(Q3[3:0]);
+100 INITFIFO=0;
+100 INITFIFO=1;
EXPAND_MACRO loadreg(Q2[3:0]);
+100 INITFIFO=0;
+100 INITFIFO=1;
EXPAND_MACRO loadreg(Q1[3:0]);
+100 INITFIFO=0;
+100 INITFIFO=1;
EXPAND_MACRO loadreg(Q0[3:0]);
END_MACRO
```

* マクロloadfifoの定義の中で、マクロloadregが4回使用されます。

前の例のように、パラメータは内部のEXPAND_MACROステートメントに渡されます。

strbin()関数への引き数も渡されます。

例：

```
DEFINE_MACRO dummy(Q3,Q2,Q1,Q0,adr)
+100 NRST=0;
+100 NRST=1;
EXPAND_MACRO loadmsb(Q3,Q2,strbin(adr, 2));
EXPAND_MACRO loadlsb(Q1,Q0,strbin(adr, 2));
END_MACRO
```

マクロ記述のヒント

マクロによりパターンが読みやすくなります。しかし、マクロを正しく使用しないと、読みやすくなりません。

マクロは簡単に

マクロは短く、単純なタスクに使用して下さい。マクロを再利用性を考えた場合このようなことが重要になります。

階層レベルの制限

マクロの“スタック”（DEFINEの中でのEXPAND）は、注意して使用して下さい。読みやすさを損なわない最大の階層レベルは1階層か2階層までです。このようなガイドラインはソフトウェアのプログラミングで使用される階層レベルと同じです。

コメントの使用

マクロが何をするのか、使用する場合のコンテキストは何か等のコメントはできるだけ詳しく記述して下さい。

ライブラリとデバイスモデル

ライブラリの概要

あるエレメント（.SUBCKT定義やモジュール定義、.MODELステートメントなど）がネットリストファイル（*circuit*.NSX）またはパターンファイル（*circuit*.PAT）に見つからない場合、アドバンストSimはライブラリファイルからそれらのエレメントを取り込もうとします。いくつかのエントリをライブラリエレメントとして保存することができます。以下にこれらのエントリを示します。

- .MODELステートメント（この章のデバイスモデルのトピックを参照して下さい。）
- .SUBCKTステートメント（*階層記述*の章を参照して下さい。）
- モジュール定義（*階層記述*の章を参照して下さい。）
- プリミティブ定義(UDPs)（*階層記述*の章を参照して下さい。）
- DEFINE_MACROステートメント（*マクロ*の章を参照して下さい。）

これらのエントリをライブラリファイルに保存する方法には2つあります。“一つのエレメント＝一つのファイル”に保存するやり方と、“.LIB”に保存するやり方、さらに2つの方法をミックスしたやりかたです。（各ファイルともに拡張子は“.LIB”です。）

“一つのエレメント＝一つのファイル”方式では、ライブラリファイルには、.MODELまたは.SUBCKT、モジュール、プリミティブ、DEFINE_MACROのいずれかの定義が一つだけ記述されます。ファイルの名前は、.MODELまたは .SUBCKT、モジュール、プリミティブ、マクロの名前になります。ファイルの拡張子は、ファイルに何が保存されているかを表します。

この“一つのエレメント＝一つのファイル”モデルでは、ファイルのベース名は、アイテムの名前を一致している必要があります。すなわち、“.MODEL YSTUFF NMOS LEVEL=3”ステートメントのあるXSTUFF.MDLファイルを作成することはできません。なぜなら、ライブラリのベースネームであるXSTUFFファイルはYSTUFFというアイテムの名前と一致していないからです。同様の決まりがサブサーキット名やモジュール名、マクロ名に適用されます。

ここでいくつかの例を示します：下線部の語が一致していることに注意して下さい。

a) 1N4148.MDLファイルには以下のアイテムが保存されます。：

```
---------------------------- file: 1N4148.MDL ----
.model 1N4148 D Is=0.1p Rs=16 CJO=2p Tt=12n
+ Bv=100 Ibv=0.1p
-------------------------------------------------
```

b) LT1013.CKTファイルには以下のアイテムが保存されます。：

```
---------------------------- file: LT1013.CKT ----
.SUBCKT LT1013 1 2 3 4 5
C1 11 12 8.66P
C2 6 7 30P
DC 8 53 DX
...
.ENDS
-------------------------------------------------
```

c) I7474.Vファイルには以下のアイテムが保存されます。：

```
------------------------------ file: I7474.V ------
module I7474( NPRE1, CLK1, D1, NCLR1, NPRE2, CLK2, D2, NCLR2, Q1,
NQ1, Q2, NQ2 );
  input NPRE1, CLK1, D1, NCLR1, NPRE2, CLK2, D2, NCLR2, Q1, NQ1,
Q2, NQ2;
  output Q1 ,NQ1 ,Q2 ,NQ2;
  DFF I1(Q1, NQ1, D1, CLK1, NCLR1, NPRE1);
  DFF I2(Q2, NQ2, D2, CLK2, NCLR2, NPRE2);
```

```
    // DFF definition has to defined somewhere else.
    // Library elements can use other library elements.
endmodule
-------------------------------------------------
```

“.LIB”モデルでは、ライブラリファイルの拡張子は、“.LIB”となり、.MODELステートメントや .SUBCKTステートメント、モジュールステートメント、DEFINE_MACROステートメントを任意の数記述することができます。

個々のファイルを利用する利点は、ファイル名を見ればそれらに含まれている内容がわかることで、さらに.LIBファイルよりもアクセスが速いです。.LIBファイルの場合、内容に関する情報はわかりませんがよりコンパクトでファイルの数が少なくできます。

➔ .LIBファイルに保存されているエレメントにアクセスは、通常若干遅くなります。従って、非常に大きな .LIBファイルを作成する事は避けて下さい。

回路がロードされた時、アドバンストSimにより実際に使用されるライブラリエレメントの情報は、*circuit*.RPTファイルに保存されます。どのライブラリファイルが使われ方を確認するには、.RPTファイルを参照して下さい。

➔ 回路と同じ名前のファイルの .LIBファイルを使用することはできません。例えば、スケマティックで *circuit*.SCHを作成した場合、*circuit*.LIBファイルを使用することはできません。これを行うと .LIBファイルのモデルが見つからないというエラーが発生します。

ライブラリファイルの場所

ライブラリファイルは、*advsim*.INIファイルの[Library]セクションで指定されるディレクトリまたはパターンファイル（*circuit*.PAT）の.LIBディレクティブによりに保存します。または、.LIBディレクティブによりパターンファイル（*circuit*.PAT）の中でライブラリファイルの場所を指定します。

➔ 詳細については*アドバンストSimファイル*の章を参照して下さい。また、オンラインヘルプのディレクティブトピックの.LIBディレクティブも参照して下さい。

advsim.INIファイルに示されるディレクトリには、通常基本的なライブラリファイルが保存されます。.LIBディレクティブにより参照されるファイルは、通常特定の“ユーザ”ライブラリファイルです。

➔ ライブラリファイルを指定するこれら2つの方法は固定された方法ではありません。好みに応じて2つをミックスして使用することができます。多くのユーザは完全にライブラリ化したエレメントを*advsim*.INIファイルのディレクトリに保存し、プロジェクトのチューニング段階のライブラリは .LIBディレクティブで指定します。

advsim.INIのライブラリセクション

ライブラリファイルのあるディレクトリは、*advsim*.INIファイル（*アドバンストSimファイル*の章を参照して下さい。）の[Library]セクションで指定します。[Library]セクションのエントリは、ディレクトリ名（フルパス名）に続いて“=yes”または“=no”が記述されます。これにより、特定のディレクトリが使用できるかどうかを指定します。

➔ アドバンストSimは、*advsim*.INI.の[Library]セクションで “=yes” のフラグがあるディレクトリを検索します。

advsim.INI ファイルの例：

```
[Library]
c:\soft\team\basic = yes
c:\user\tom\bipolar = yes
c:\user\jerry\diodes = no
```

アドバンストSimが最初に起動された時点で、*advsim*.INIで指定されるライブラリのリストがメモリに構築されます。これは回路をロードするたびに行われるわけではありません。*advsim*.INIファイルの[Library]セクションを変更した場合、変更内容を有効にするにはアドバンストSimを終了して再起動する必要があります。変数のライブラリエレメントは .LIBステートメント、またはLoad Libraryダイアログ（以下を参照）で扱うのがもっとも良い方法です。

.LIB ライブラリエレメント

advsim.INIファイルの[Library]セクションのディレクトリ以外に、ライブラリファイルとして厳密に指定する方法があります。.LIBディレクティブをパターンファイルに記述して、ライブラリファイルの名前を与えます。複数の.LIBステートメントをパターンファイルに記述することもできます。

これらの.LIBディレクティブはLoad Libraryダイアログで編集（追加や削除）することができます。このダイアログにより、名前を入力しないでファイルシステム内を移動したり必要なファイルを選択したりすることができます。このダイアログでの選択結果によりパターンファイルを更新することができます。

以下に、.LIBディレクティブに記述されるファイル名の拡張子と内容を示します。

.MDL -> .MODELステートメント一つ。

.CKT -> .SUBCKTステートメント一つ。

.V -> モジュール定義が一つかプリミティブ定義が一つ。

.MAC -> DEFINE_MACRO 定義が一つ。

.LIB -> 任意の数の上記のエントリが単に連結されたもの。

➔ .Vまたは.LIB拡張子を持つライブラリファイルは、使用される前にVerilogのプリプロセッサをパスされます。

通常、ファイルを指定するにはフルパスが指定されます。また、ドット文字を使用してカレントディレクトリ（*circuit*.NSXまたは*circuit*.PATのディレクトリ）を指定し、相対パスを指定することもできます。以下の例を参照して下さい。

.LIBディレクティブファイルの例：

```
* in circuit.PAT:
.LIB c:\user\bill\simul\gates.LIB
.LIB c:\user\bill\simul\control\control.V
.LIB c:\user\tom\bipolar\q2n2222.MDL
.LIB c:\team\behavrl\zd_ram.dmd
.LIB c:\team\behavrl\za_aop.amd
.LIB .\aoplocal.LIB
```

一致判定基準

エレメントの検索を行う場合、検索対象のエレメントが含まれるファイルを見つけるためにすべての".LIB"ファイルが走査されます。

検索が成功する場合を以下に示します：

1. 検索されるエレメントの名前と一致するベースネームを持ち、検索されたエレメントのタイプと一致する拡張子を持つファイルが存在する。これらの拡張子を以下に示します。

   .MDL .MODELが検索される場合

   .CKT .SUBCKTが検索される場合

   .V module が検索される場合

   .MAC DEFINE_MACROが検索される場合

2. もし1.が失敗したにもかかわらず検索されるエレメントを含む".LIB"ライブラリファイル（ベースネームは不明）が存在する場合、この".LIB"ライブラリファイルは.LIBディレクティブにより指定する必要があります。または、*advsim*.INIで宣言されるライブラリディレクトリの一つにそのファイルが存在している必要があります。

例：

circuit.NSX ファイルに以下のステートメントがある場合：

```
X1 EP EM VDD VSS OUT MAG357
```

ただし、この場合、MAG357サブサーキット（.SUBCKT ステートメント）の定義は*circuit*.NSXファイルに存在していません。アドバンストSimはライブラリディレクトリの中でサブサーキットの定義を検索します。以下の場合、この検索は成功します。

- MAG357.CKTという名前のファイルは、.LIBディレクティブで指定されているか、または、*advsim*.INIの[Library]セクションで指定されるディレクトリの一つにある。内容は以下のような内容です。

```
--------------------------- file: MAG357.CKT ----
.SUBCKT MAG357 1 2 3 4 5
...
.ENDS
-------------------------------------------------
```

- .LIBディレクティブまたは*advsim*.INIの[Library]セクションのディレクトリに（例えば）OPAMP.LIBという名前のファイルがあり、その内容にMAG357の定義がある。

```
--------------------------- file: OPAMP.LIB ----
...
.SUBCKT P38 1 2 3 4 5
...
.ENDS
.SUBCKT MAG357 1 2 3 4 5
...
.ENDS
.SUBCKT OP741 1 2 3 4 5
...
.ENDS
...
-------------------------------------------------
```

優先順位

デフォルトでは、.LIBディレクティブで記述されるファイルが最初に検索され、エレメントが見つからなかった場合、*advsim*.INIファイルの[Library]セクションのディレクトリが検索されます。複数の.LIBディレクティブがパターンファイルに記述されている場合、ファイルはパターンファイルに記述されている順に検索されます。エレメントを見つけることができなかった場合、エラーになります。

この優先順位を逆転させてアドバンストSimに*advsim*.INIで指定するディレクトリを最初に検索させ、その後.LIBディレクティブのファイルを検索するようにするには、Library File Selectionダイアログ(File-Select Model Libraries)のpriorityオプションを設定して下さい。

advsim.INIの[Library]セクションで指定したディレクトリツリーに重複が発生した場合、片方のファイルを使用することになります。*circuit*.RPTファイルでどのファイルが使用されたかを確認して下さい。

ダブルチェック

回路がロードされる時にアドバンストSimにより実際に使用されるライブラリエレメントは、使用されたエレメントのフルパス名を含むメッセージが*circuit*.RPTファイルにレポートされます。以下に例を示します：

```
“Using c:\team\lib\aop.CKT - aop.CKT”
“Using c:\team\lib\analog\bip.LIB - q2N222.MDL”
```

2番目のメッセージにより、Q2N2222コンポーネントの.MODELステートメントはbip.LIBファイルで見つかった（使用された）ことがわかります。*circuit*.RPTを参照してどのライブラリファイルが実際に使用されたかを確認して下さい。

デバイスモデル

概要

モデルについて

MOS トランジスタやバイポーラトランジスタ、ダイオードのようなアナログデバイスの中には、“model”と関連つけられているものがあります。たまたま、“model”という語はある実在するものを表すために使用されています。

最初は、デバイスの電流やキャパシタンス、チャージを計算するために使用する式を参照する場合です。例えば、ほとんどのアナログシミュレータではバイポーラトランジスタの

モデルには“Gummel-Poon”モデルを使用します。アドバンストSimでもこのモデルを使用しています。これらの数式は、通常i=f(u, p)という式で表されます。ここで i はデバイスを流れる電流で、uはデバイスの端子上の電圧のベクトル、p はパラメータの組みです。

モデルという言葉の2番目の使われかたは、SPICE用語としてです。ここでは、特定のパラメータの組みを参照するために使用されます。この使われかたは、SPICE 2G.6（とほとんどすべてのアナログシミュレータ）からはじまっており、.MODELから始まるステートメントを使用してパラメータの組みの記述をはじめます。

MOSトランジスタモデルには、“level”も用いられます。詳細は後述のMOSトランジスタの項を参照して下さい。

.MODEL ステートメント

ネットリストでデバイスを接続する場合、端子の名前と“model”に関連つけられた名前を指定します。このモデルは実際には、.MODELステートメントを使用して記述されるパラメータの組みです。.MODELステートメントはパターンファイルまたはネットリストファイル（推奨しません）、モデルライブラリファイル（拡張子.MDLのファイル）、サブサーキット定義（.SUBCKT）で記述することができます。

.MODELステートメントの一般的な構文を以下に示します：

```
.MODEL modname KEY [param=value] [param=value]...
```

modnameはモデルの名前です。複数のデバイスで同じ.MODELステートメントを参照することもできます。このモデルがライブラリに保存されている場合、*modname*.MDLという名前のファイルに.MODELステートメントを記述しライブラリツリーにファイルを保存する必要があります。

KEYはデバイスがモデルがどのタイプかを指示するためのキーワードです。キーワードを以下に示します。：

MOSトランジスタの場合:	NMOS または PMOS
バイポーラトランジスタの場合:	NPN または PNP
JFETの場合:	NJF または PJF
ダイオードの場合:	D
インタフェースデバイスの場合:	ITF

キーワードの後に続く“param=value”は、一連のパラメータです。不正なパラメータが指定された場合、*circuit*.RPTファイルのWARNINGセクションにエラーが出力されます。

➔ 通常、.MODELステートメントのパラメータはすべてオプションです。唯一の例外はBSIMモデルの場合です。以下のMOSトランジスタの項を参照して下さい。

必要ならば、.MODELステートメントは‘+’文字を使用して複数行に渡り、記述することができます。

例：

```
.MODEL NTYP NMOS LEVEL=2 UO=550 VTO=0.67
+ PHI=0.76 GAMMA=0.56
+ CJ=3e-4
```

.op ファイルへのモデルパラメータの出力

.OPディレクティブでMODELINFO=YESフラグを追加することにより、*circuit*.OPファイルにモデルパラメータを出力することができます。（.OPディレクティブの記述を参照して下さい。）このフラグが.OPディレクティブに存在すると、モデルパラメータの値が*circuit*.OPファイルの特別なセクションに出力されます。出力された値はアドバンストSimの内部的な値で、.MODELステートメント内の値ではありません。これは .MODELステートメントが正しく回路にロードされているかどうかを確認するのに有効です。また、指定されていないパラメータのデフォルト値を確認するのにも有効です。

MOSトランジスタモデル

MOSトランジスタのモデルとして幾つかのモデル（数式や関連つけられたパラメータ）を使用することができます。.MODEL行のキーワードはNMOSまたはPMOS（この章の概要を参照して下さい。）です。

.MODELステートメントでは、LEVELパラメータを使用して使用するモデルを切り替えます。.MODELステートメントに“LEVEL=n”（ここでnは使用するモデルのレベルです。）を追加することにより特定の“ レベル ”を選択することができます。

“level”は数値(1, 2, 3, 4...)ですが、レベルは単に複雑さや緻密さなどの尺度では、なく、歴史的な経緯でこのようになっています。それぞれのレベルには、それぞれの特性があります。

パラメータの中には、すべてのレベルで共通で同じように使われるものもあります。これらの共通なパラメータを以下の特別なセクションに示します。

すべてのレベルで共通に同じ様に使用されるパラメータ:

名前	デフォルト値	単位	説明
CREC	0.0	F/m	ｹﾞｰﾄ/ｿｰｽ-ﾄﾞﾚｲﾝのｵｰﾊﾞｰﾗｯﾌﾟｷｬﾊﾟｼﾀﾝｽ
CGSO	0.0	F/m	CRECと同じ
CGDO	0.0	F/m	CRECと同じ
PB	0.8	V	ジャンクション電位
CGBO	0.0	F/m	ｹﾞｰﾄ/ﾊﾞﾙｸｵｰﾊﾞｰﾗｯﾌﾟｷｬﾊﾟｼﾀﾝｽ
CJ	0.0	F/m^2	底部ｼﾞｬﾝｸｼｮﾝｷｬﾊﾟｼﾀﾝｽ
CJSW	0.0	F/m	側壁ｼﾞｬﾝｸｼｮﾝｷｬﾊﾟｼﾀﾝｽ
MJ	0.5	-	底部ｷｬﾊﾟｼﾀﾝｽを表す公式の指数部
MJSW	0.33	-	側壁ｷｬﾊﾟｼﾀﾝｽを表す公式の指数部
IS	0.0	A	ｼﾞｬﾝｸｼｮﾝの飽和電圧
JS	0.0	A/m/m	ｼﾞｬﾝｸｼｮﾝの飽和電圧密度
LDIF	0.0	m	側面拡散幅
FC	0.5	-	ｼﾞｬﾝｸｼｮﾝｷｬﾊﾟｼﾀﾝｽの逆公式の係数
RD	0.0	Ohm	ドレインのオーム抵抗
RS	0.0	Ohm	ソースのオーム抵抗
RDC	0.0	Ohm	ドレインの接触抵抗
RSC	0.0	Ohm	ソースの接触抵抗
RSH	0.0	Ohm/_	拡散抵抗 (面積あたり)

レベル0のパラメータ:

名前	デフォルト値	単位	説明
VTO	0.0	V	スレッショルド電圧
RON	1000	Ohm	スイッチの«on»抵抗
ROFF	1e12	Ohm	スイッチの«off»抵抗
TRW	0.5	V	«on»から«off»への遷移幅

コメント

レベル0は、本当のトランジスタモデルよりも、よりスイッチモデルに近いモデルでドレイン－ソース抵抗を変数として持ちます。スイッチとキャパシタ回路のシミュレーションを簡単にするためにこうような構造になっています。レベル0のトランジスタモデルは、off（vgs-VTO<0の条件により« off »状態になります。この時ドレイン－ソース間の抵抗はROFFオームです。）またはon（vgs-VTO>TRWの条件により« on »状態になります。この時ドレイン－ソース間の抵抗はRON*L/Wです。）です。vgs-VTOが0.0ボルトからTRWボルトまで変化すると、抵抗はROFFからRON*L/Wまで変化します。

レベル0（本当のMOSトランジスタのようには動作しませんが）を使用すると便利です。というのは、レベル0を使用していれば同じネットリストでスイッチモデル（レベル0）またはリアルモデル（レベル> 0）を使用したシミュレーションができます。

➔ 注意：レベル0モデルは、スイッチとして動作するトランジスタのシミュレーションに適しています。リニアな領域で動作するトランジスタとして使用しないで下さい。オペアンプのシミュレーションには使用しないで下さい。正しく動作しません。

レベル1のパラメータ:

名前	デフォルト値	単位	説明
GAMMA	0.0	$V^{1/2}$	バルク係数
TOX	1e-7	m	酸化膜厚さ
VTO	0.0	V	ｽﾚｯｼｮﾙﾄﾞ電圧(VBS=0)
UO	600	cm^2/Vs	モビリティー
PHI	0.6	V	表面電位
NSUB	1e16	cm^{-3}	不純物濃度
LD	0.0	m	側面拡散
DL	0.0	m	チャンネル長補正値
DW	0.0	m	チャンネル幅補正値
LAMBDA	0.01	1/V	チャンネル長補正値
AF	1.0	-	フリッカーノイズの指数
KF	0.0	-	フリッカーノイズの係数
NSS	0.0	$1/cm^2$	表面状態密度
KP	-	A/V^2	コンダクタンス係数

➔ 上記のパラメータに“すべてのレベルで共通に使用されるパラメータ”が加わります。

コメント

レベル1は、Shichman-Hodgeモデルです。モデルとしては単純なモデルですが、精度とスピードの妥協点という意味では多くの人々が考えているよりも価値があります。多くの単純なシミュレーションで使用できます。

このモデルには、サブスレッショルド伝達モデルがありません。チャンネルの長さによる減衰モデルは飽和領域で最小(`ids=ids•(1+LAMBDA•vds)`)です。モビリティーの減衰モデルもありません。短く幅の狭いチャンネルの影響は無視できます。

係数`L`と`W`は以下の式で計算されます。：

```
L_elec = L_drawn - 2•LD + 2•DL
W_elec = W_drawn + 2•DW
```

`KP`が与えられない場合、`UO•epsilon_ox/TOX`で計算します。

`GAMMA`, `PHI`, `VTO`について：

パラメータが与えられる場合、その値が使用されます。

パラメータが与えられず`NSUB`が与えられる場合、`NSUB`（と`VTO`の`NSS`）の関数として計算されます。

パラメータが与えられず`NSUB`も与えられない場合、デフォルト値が使用されます。

➔ `LAMBDA`のデフォルト値は0.01です。0ではありません。

レベル2のパラメータ:

名前	デフォルト値	単位	説明
GAMMA	0.0	$V^{1/2}$	バルク係数
TOX	1e-7	m	酸化膜厚さ
VTO	0.0	V	ｽﾚｯｼｮﾙﾄﾞ 電圧 (VBS=0)
UO	600	cm^2/Vs	モビリティー
PHI	0.6	V	表面電位
DELTA	0.0	-	Vth従属wrtチャンネル幅
UCRIT	1e4	V/cm	モビリティー性能低下因数
UEXP	0.0	-	モビリティー性能低下指数

UTRA	0.0	-	受付けますが使用しません
NFS	0.0	$1/cm^2$	高速表面状態密度
NSUB	1e16	cm^{-3}	不純物濃度
XJ	0.0	m	結合深度
LD	0.0	m	側面拡散
DL	0.0	m	チャンネル長補正値
DW	0.0	m	チャンネル幅補正値
LAMBDA	0.0	1/V	チャンネル長補正値
AF	1.0	-	フリッカーノイズの指数
KF	0.0	-	フリッカーノイズの係数
NSS	0.0	$1/cm^2$	表面状態密度
KP	-	A/V^2	コンダクタンス係数

➔ 上記のパラメータに“ すべてのレベルで共通に使用されるパラメータ ”が加わります。

コメント

レベル2は解析的なモデルです。複雑で処理速度も遅いモデルです。2umより細い技術には使用しないで下さい。未だに多くのシリコン製造工場では、彼らの技術にレベル2のパラメータだけを与えています。サブスレッショルド伝達モデリングは、NFSパラメータがあれば行われます。短くて幅の狭いチャンネルでは、チャンネル効果を考慮する必要があります。

係数LとWは以下の式で計算されます。：

```
L_elec = L_drawn - 2•LD + 2•DL
W_elec = W_drawn + 2•DW
```

KPが与えられない場合、UO•epsilon_ox/TOXで計算します。

GAMMA, PHI, VTOについて：

パラメータが与えられる場合、その値が使用されます。

パラメータが与えられずNSUBが与えられる場合、NSUB（とVTOのNSS）の関数として計算されます。

パラメータが与えられずNSUBも与えられない場合、デフォルト値が使用されます。

レベル3のパラメータ

名前	デフォルト値	単位	説明
GAMMA	0	$V^{1/2}$	バルク係数
THETA	0	1/V	VGSによるﾓﾋﾞﾘﾃｨｰの調整値
TOX	1e-7	m	酸化膜厚さ
VTO	0	V	ｽﾚｯｼｮﾙﾄﾞ電圧(VBS=0)
UO	600	cm^2/Vs	モビリティー
PHI	0.6	V	表面電位
KAPPA	0.2	-	チャンネル短縮効果
KAPPACONT	0	-	連続gdsのフラグ(本文参照)
VMAX	0	m/s	飽和速度
ETA	0	-	静的フィードバック効果
DELTA	0	-	Vth従属wrtチャンネル幅
NFS	0	$1/cm^2$	高速表面状態密度
NSUB	1e16	cm^{-3}	不純物濃度
XJ	0	m	結合深度
LD	0	m	側面拡散
DL	0	m	チャンネル長補正値
DW	0	m	チャンネル幅補正値
AF	1.0	-	フリッカーノイズの指数
KF	0.0	-	フリッカーノイズの係数
NSS	0.0	$1/cm^2$	表面状態密度
KP	-	A/V^2	コンダクタンス係数

➔ 上記のパラメータに“すべてのレベルで共通に使用されるパラメータ”が加わります。

コメント

レベル3は、semi-empiricalモデルです。レベル2よりも処理速度は速いです。レベル2と同じコメントですが、2umより細い技術には使用しないで下さい。未だに多くのシリコン製造工場では、彼らの技術にレベル2のパラメータだけを与えています。サブスレッショルド伝達モデリングは、NFSパラメータがあれば行われます。短くて幅の狭いチャンネルでは、チャンネル効果を考慮する必要があります。

レベル2と比較すると、レベル3の式とパラメータはより単純です。すなわち、各パラメータの影響をレベル2よりも予測しやすいということです。

パラメータKAPPACONTにより、チャンネル長の調整値が変更されます。パラメータKAPPAが1.0の場合を除いて、リニア領域から飽和領域の遷移のときのgdsの不連続が元の式により生成されます。KAPPACONTが1に設定されると、式の変更が行われます。これは、もとのバージョンとは違います。要約すると、KAPPACONT=1に設定しなければ、KAPPAが1.0ならばgdsが不連続になる元の式が使われます。KAPPACONT=1に設定すれば、任意のKAPPAでgdsは連続なり、このgdsはオリジナルのgdsとわずかに違います。

係数LとWは以下の式で計算されます。：

```
L_elec = L_drawn - 2•LD + 2•DL
W_elec = W_drawn + 2•DW
```

KPが与えられない場合、UO•epsilon_ox/TOXで計算します。

GAMMA, PHI, VTOについて：

パラメータが与えられる場合、その値が使用されます。

パラメータが与えられずNSUBが与えられる場合、NSUB（とVTOのNSS）の関数として計算されます。

パラメータが与えられずNSUBも与えられない場合、デフォルト値が使用されます。

レベル4のパラメータ(BSIM)

名前	デフォルト値	単位	説明
TOX	0	µm	酸化膜厚さ
TEMP	0	°C	温度
VDD	0	V	バイアス適用範囲
DL	0	µm	チャンネル長補正値
DW	0	µm	チャンネル幅補正値
VFB *	0	V	フラット－バンド電圧
PHI *	0	V	極性反転表層電位
K1 *	0	$V^{1/2}$	ボディー効果係数
K2 *	0		ﾄﾞﾚｲﾝ/ｿｰｽ消耗電荷分割係数
ETA *	0		ｾﾞﾛﾊﾞｲｱｽﾄﾞﾚｲﾝによるﾊﾞﾘﾔ低下係数
X2E *	0	1/V	ｻﾌﾞｽﾄﾚｰﾄﾊﾞｲｱｽへのETA感度
X3E *	0	1/V	vds=VDDでのﾄﾞﾚｰﾝﾊﾞｲｱｽへのETA感度
MUZ	0	cm^2/Vs	ゼロバイアスモビリティー
X2MZ *	0	cm^2/V^2s	vds=0でのｻﾌﾞｽﾄﾚｰﾄﾊﾞｲｱｽへのMUZ感度
MUS *	0	cm^2/Vs	vds=VDDでのｾﾞﾛﾊﾞｲｱｽﾓﾋﾞﾘﾃｨ
X2MS *	0	cm^2/V^2s	vds=VDDでのｻﾌﾞｽﾄﾚｰﾄﾊﾞｲｱｽへのMUS感度
X3MS *	0	cm^2/V^2s	vds=VDDでのﾄﾞﾚｰﾝﾊﾞｲｱｽへのMUS感度
U0 *	0	1/V	モビリティー低下係数
X2U0 *	0	$1/V^2$	ｻﾌﾞｽﾄﾚｰﾄﾊﾞｲｱｽへのU0感度
U1 *	0	µm/V	飽和速度
X2U1 *	0	$µm/V^2$	ｻﾌﾞｽﾄﾚｰﾄﾊﾞｲｱｽへのU1感度
X3U1 *	0	$µm/V^2$	ﾄﾞﾚｲﾝﾊﾞｲｱｽへのU1感度
N0 *	0		ｻﾌﾞｽﾚｯｼｮﾙﾄﾞ立ち下がり係数
NB *	0		ｻﾌﾞｽﾄﾚｰﾄﾊﾞｲｱｽへのN0感度
ND *	0		ﾄﾞﾚｲﾝﾊﾞｲｱｽへのN0感度
XPART	0		ﾄﾞﾚｲﾝ/ｿｰｽ電荷分割点 XPART=0の場合、40/60ﾓﾃﾞﾙが使われる。 XPART=1の場合、0/100ﾓﾃﾞﾙが使われる。
JSSW	0	A/m^2	結合飽和電流密度(ﾊﾟﾗﾒｰﾀ)
PBSW	0	V	結合電位（周辺の長さ）
AF	1.0	-	フリッカーノイズ指数
KF	0	-	フリッカーノイズ係数

➔ 上記のパラメータに“すべてのレベルで共通に使用されるパラメータ”が加わります。

コメント

BSIMモデルは、アドバンストSimのレベル4としてインプリメントされます。このモデルには、電荷を保持するという興味深い特性があります。この特性は特別なアプリケーションに必要になります。しかし、電荷の式は極めて複雑でモデルの処理速度を低下させます。

係数LとWは以下の式で計算されます。：

```
L_elec = L_drawn - 2DL + 2DL
W_elec = W_drawn - 2DW
```

➔ パラメータの単位は変わることがあることに注意して下さい。また、有効長さや有効幅の計算はレベル1、2、3とは違う方法で計算されることに注意して下さい。

ほとんどのパラメータ（名前の後ろに*のついたもの）は、トランジスタの長さや幅に影響されない係数です。これらの係数はLPARAMやWPARAMと呼ばれ、PARAMがパラメータの名前です。パラメータの実際の値は以下の式で計算されます。：

```
PARAM = PARAM + LPARAM/L_elec + WPARAM/W_elec
```

もしUNITがPARAMの単位なら、WPARAMやLPARAMの単位はUNIT•μmです。

例：

PHIはボルト、LPHIはVμm、WPHIはVμm、
PHI=PHI+LPHI/L_elec+WPHI/W_elec、ここで、
L_elecとW_elecは有効長さと有効幅です。

➔ 注意：物理的に意味のあるデフォルト値が与えられているわけではありません。従って、パラメータは必ず指定して下さい。

レベル5のパラメータ(EPFL)

名前	デフォルト値	単位	説明
LUNIT	1e-6	m	長さの単位変換係数 1.0または1e-6の値しか取りません。 （デフォルトは1e-6）
COX**	0.7e-15	F/µm²	酸化膜厚総静電容量
VTO	0.5	V	スレッショルド電圧(VBS=0)
GAMMA	1.0	V^1/2	ボディー効果
PHI	0.7	V	表面電位
KP	5e-5	µA/V²	コンダクタンスパラメータ
THETA	0.0	1/V	モビリティー減少係数
UCRIT**	2.0	V/µm	縦方向の臨界磁界
DW	0.0	µm	チャンネル幅縮小値
DL	0.0	µm	チャンネル長縮小値
LAMBDA	0.5	-	削除長係数
WETA	0.25	-	チャンネル幅縮小効果係数
LETA	0.1	-	チャンネル長縮小効果係数
NQS	0	-	近似静的モデル起動フラグ
SATLIM	exp(4)	-	飽和比率の定義
TCV	0.0	V/°C	スレッショルド電圧温度係数
BEX	-1.5	-	モビリティ温度指数
KF	0.0	-	フリッカーノイズ係数
AF	1.0	-	フリッカーノイズ指数

➔ 上記のパラメータに“すべてのレベルで共通に使用されるパラメータ”が加わります。

LUNIT パラメータ

LUNITパラメータは、長さの単位（mまたはµm）を切替えるために使用します。

LUNITの値が1.0までの長さの単位はm（メートル）になります。これは標準的なシステムでのルールで、デバイスラインのWとLの単位もmを使用します。ADはm²、ASはm²、PDはm、PSはm単位でそれぞれ表記します。

LUNITの値が1e-6（デフォルト値）に設定されている場合、長さはすべてµmで表記されます。この時、WやLもµmで表記する必要があります。ADはµm²、ASはµm²、PDはµm、PSはµm単位でそれぞれ表記します。

CREC, CGSO, CGDO, CGBO, CJ, CJSW, JS, DW, DL

これらのパラメータの中には長さが含まれているために、LUNITの値に応じてこれらのパラメータ(CREC, CGSO, CGDO, CGBO, CJ, CJSW, JS, DW, DL)は、異なる単位で指定する必要があります。以下の表を参照して下さい。

If LUNIT is set to 1e-6:

パラメータ	デフォルト値	標準値	単位
CREC	0.0	2e-4	(F/µm)
CGSO	0.0	2e-4	(F/µm)
CGDO	0.0	2e-4	(F/µm)
CJ	0.0	5e-10	(F/µm²)
CJSW	0.0	5e-16	(F/µm)
JS	0.0	5e-17	A/µm²
DW	0.0	0.1	(µm)
DL	0.0	0.1	(µm)

If LUNIT is set to 1.0:

パラメータ	デフォルト値	標準値	単位
CREC	0.0	2e-4	(F/m)
CGSO	0.0	2e-4	(F/m)
CGDO	0.0	2e-4	(F/m)
CJ	0.0	5e4	(F/m^2)
CJSW	0.0	5e-10	(F/m)
JS	0.0	5e-11	A/m^2
DW	0.0	0.1e-6	(µm)
DL	0.0	0.1e-6	(µm)

COX** とUCRIT**

LUNITが1e-6の場合、前の表のデフォルト値がデフォルト値になります。LUNITが1.0の場合COXとUCRITは異なるデフォルト値を持ちます。

LUNITが1e-6の場合

パラメータ	デフォルト値	単位
COX	0.7e-15	(F/µm2)
UCRIT	2.0	(V/µm)

LUNITが1.0の場合

パラメータ	デフォルト値	単位
COX	0.7e-3	(F/m2)
UCRIT	2.0e6	(V/m)

例題

LUNITの値に応じた異なる.MODELステートメントを使用することにより、一貫した結果を得ることができます。以下に、.MODELステートメントとトランジスタに関連した例を示します。

LUNIT=1uの場合

```
.MODEL N NMOS LEVEL=5 LUNIT=1u NQS=0
+ COX=0.7M VTO=0.75 GAMMA=0.57 PHI=0.7
+ KP=77U THETA=0.03 UCRIT=1.8E6 LAMBDA=0.4
+ WETA=0.6 LETA=0.1
+ CJ=2.4E-4 CJSW=4E-10 CGSO=3.0E-10
+ CGBO=3.0E-10
+ JS=4e-6 DW=0.25U DL=0.23U
* and:
MTEST DR GT SRC 0 N W=50U L=2.5U
+ AD=50e-12 PD=47U
```

LUNIT=1の場合

```
.MODEL N NMOS LEVEL=5 LUNIT=1 NQS=0
+ COX=0.7e-15 VTO=0.75 GAMMA=0.57 PHI=0.7
+ KP=77U THETA=0.03 UCRIT=1.8 LAMBDA=0.4
+ WETA=0.6 LETA=0.1
+ CJ=2.4E-10 CJSW=4E-16 CGSO=3.0E-16
+ CGBO=3.0E-16
+ JS=4e-18 DW=0.25 DL=0.23
* and
MTEST DR GT SRC 0 N W=50 L=2.5
+ AD=50 PD=47
```

コメント

EKVモデルは、アドバンストSimの場合レベル5としてインプリメントされます。このモデルは、スイスのE.PF.L. (Ecole Polytechnique Federale de Lausanne)により開発されました。アナログアプリケーション用のレベル2, 3, 4に関連する多くの問題をこの方法により解くことができます。特に低電力のアナログアプリケーションの問題に優れています。主な特徴を以下に示します。

- 正確なサブスレッショルド伝達モデリングができます。
- ドリフト領域のDC電流にすべて同じ公式を適用できます。
- 極性反転のドリフト領域が充分考慮されています。（レベル2, 3, 4の場合、この基本的なドリフト領域は無視）
- 電流と導関数は連続です。よく知られているweakからstrongへの逆遷移時の“gm-kink”は無視されます。gdsモデリングは、リニア領域から飽和領域まで連続です。
- 固有静電容量CSBとCDBが考慮されます。
- ノイズモデリングが正確です。
- 非準安定状態動作が可能です。

- パラメータが少ないです。

➔ EKVモデルの詳しい説明は、“The Enz-Krummenacher-Vittoz or EKV MOSFET model and equations for SMASH™”, C. Enz, EPFL-LEGを参照して下さい。

この文書が必要な場合、以下へ問い合わせて下さい。

EPFL Electronics Laboratories (LEG)
ELB Ecublens
CH-1075
Lausanne,
SWITZERLAND.

実際のLとWが以下の式で計算されます。

```
L_elec = L_drawn + DL
W_elec = W_drawn + DW
```

実際の長さや幅はレベル1, 2, 3, 4でそれぞれ異なるので注意して下さい。

チャンネルの幅と長さの実際の値

あるシミュレータから他のシミュレータへ .MODELを変換する場合、2つのシミュレータの間でチャンネル幅やチャンネル長さの実際の値の補正が同じでない時、これは難しい問題になります。

アドバンストSimでは、レベル1, 2, 3においてはLD, DL, DWをパラメータに使用します。LDはオリジナルのSPICEパラメータで縦方向の拡散幅を表わします。SPICEでは2•LDを引くことにより単に長さを補正します。一方は幅は補正しません。

SPICE 2G.6による式では、うまくシミュレーションを合わせる必要がある場合、LDが物理的に無意味な値を持つことがよくあるということが問題です。これは、LDが実効長さの計算に使用されるだけでなく、チャンネル短縮効果の計算にも使用されるためです。前述のDLパラメータは、これらの問題点を解消してくれます。

アドバンストSimのレベル1, 2, 3

```
L_elec = L_drawn - 2•LD + 2•DL
W_elec = W_drawn        + 2•DW
```

SPICE 2G.6のレベル1, 2, 3

```
L_elec = L_drawn - 2•LD
W_elec = W_drawn
```

レベル4 (BSIM)では、パラメータDLとDWは以下の様に使用されます。

レベル4のアドバンストSim/BSIM公式

```
L_elec = L_drawn - DL
W_elec = W_drawn - DW
```

レベル5 (EKV)では、パラメータDLとDWは以下の様に使用されます。

レベル5のアドバンストSim/EKV公式

```
L_elec = L_drawn + DL
W_elec = W_drawn + DW
```

➔ レベル4 (BSIM)とレベル5 (EKV)では、DLとDWパラメータは全体の補正に使用されます。一方、レベル1, 2, 3では、per-edge補正値を表わします。

デフォルトの拡散領域と周辺の長さ

LDIFパラメータを使用することにより、ドレイン／ソース面積と周囲長を計算することができます。（ネットリスト内の）デバイスラインにAD, AS, PD, PSの値が指定されていないトランジスタでは、面積と周囲長は以下のように計算されます。

```
AD = AS = (W_drawn - total_dw) • LDIF
```

と

```
PD = PS = W_drawn - total_dw + LDIF • 2
```

➔ 上の公式のWの値は、ネットリストに記述された値です。

パラメータLDIFにより、面積と周囲長が計算されます。しかし、バルク－ソース拡散静電容量とバルク－ドレイン拡散静電容量は以下の式で計算されるため、静電容量がゼロでないためには、CJとCJSW モデルパラメータの値はゼロでないことが必要です。

```
cj_bs = CJ•AS•f(vbs,MJ)+CJSW•PS•f(vbs,MJSW)
cj_bd = CJ•AD•f(vbd,MJ)+CJSW•PD•f(vbd,MJSW)
```

➔ 結局、AD, AS, PD, PSに使われるのはどの値が不安な場合、.OPディレクティブでBIASINFO=LONGスイッチを使用して下さい。これにより、*circuit*.OPファイルでこれらのパラメータの実際の値を列挙することができます。

➔ LDIF パラメータは直列抵抗の計算に影響を及ぼす可能性があります。以下の説明を参照して下さい。

直列寄生抵抗

どのデバイスパラメータ(NRD, NRS)やどのモデルパラメータ(RD, RS, RDC, RSC, RSH, LDIF)が与えられるかに応じて、直列寄生抵抗の計算方法が変わります。

NRDとNRS はデバイスラインに与えられるパラメータであることを思い出して下さい。（回路プリミティブとスティミュラスの章のアナログプリミティブの項、MOSトランジスタを参照して下さい。）これらのパラメータは、トランジスタのドレインとソースの拡散面積の数です。デフォルト値は0です。

例：

```
M1 D G S VSS MOSNTYP W=10U L=1.4U NRD=3 NRS=4
```

説明の中でドレイン抵抗が使用されていますが、ソース抵抗についても同様です。rdとrsを最終的な抵抗値として使用します。

3つの場合があります。

ケース1)

パラメータRDは、関連の .MODELステートメントで指定されます。

この場合、ドレイン抵抗は、

rd = RDC + RD

ケース2)

パラメータRDは、関連の .MODELステートメントで指定されません。

NRDはMOSインスタンスパラメータとして与えられます。この場合ドレイン抵抗は、

rd = RDC + RSH • NRD

RSHが与えられないか .MODELステートメント内でゼロ（RSHのデフォルト値はゼロ）が設定されている場合、NRDを指定するポイントはありません。

ケース3)

パラメータRDは、関連の.MODELステートメントで指定されません。

NRDはMOSインスタンスパラメータとして与えられません。

LDIFパラメータは関連の.MODELステートメントで与えられます。

この場合ドレイン抵抗は、

rd = RDC + RSH • LDIF / W_drawn

式の中で使用されるWは記述された値（ネットリストの中で記述される）で係数ではありません。

➔ 結局、AD, AS, PD, PSに使われるのはどの値が不安な場合、.OPディレクティブでBIASINFO=LONGスイッチを使用して下さい。これにより、*circuit*.OPファイルでこれらのパラメータの実際の値を列挙することができます。

正重合静電容量

パラメータCREC, CGS0, CGD0は同様の意味を持ち、どれかを使用してドレイン／ソースの正重合静電容量が表されます。アドバンストSimでは、ゲート／ソースとゲート／ドレインについて正重合の違いはありません。CRECの幅は実行幅と掛けられ電圧から独立した静電容量が計算されます。

正重合静電容量の値は、*circuit*.OPファイルのbiasセクションに列挙されます。

固有静電容量のように、正重合静電容量はトランジスタの内部端子の間に接続されます。寄生直列抵抗は内部端子と外部端子との間に接続されます。外部端子はネットリストに記述された端子です。寄生直列抵抗がある場合、内部端子と内部ノードは自動的に作成されます。

MOSトランジスタの内部変数へのアクセス

トランジスタの端子電流[ID(), IS(), IG(), IB()]に加えて、特別な.TRACEディレクティブや.PRINTディレクティブの構文を使用すればトランジスタの内部変数（gmやgds、vdsatのような変数）にアクセスすることができます。

➔ オンラインヘルプのディレクティブの項の .TRACEディレクティブの説明を参照して下さい。

内部変数にアクセスする一般的な構文は、

```
.TRACE TRAN|DC IN(Mname.VARNAME)
```

です。

ここで、MnameはMOSトランジスタのインスタンス名、VARNAMEは有効な内部変数の一つです。このセクションの最後にある表と例を参照して下さい。

このセクションでは、関連したモデルの“ レベル ”に応じて若干違いますが、使用できる内部変数を示します。特に、レベル4 (BSIM)は電荷指向のため、電荷に関連する多くのパラメータがあります。

レベル1, 2, 3, 5で使用できる内部変数：

IN(M.GDS)	ソース－ドレインコンダクタンス(dI/dVDS)
IN(M.GM)	ゲートトランスコンダクタンス(dI/dVGS)
IN(M.GMBS)	バルクトランスコンダクタンス(dI/dVBS)
IN(M.LOGGDS)	log(IN(GDS))
IN(M.LOGM)	log(IN(GM))
IN(M.LOGGMBS)	log(IN(GMBS))
IN(M.VTH)	スレッショルド電圧
IN(M.VDSAT)	飽和電圧
IN(M.BETA)	µ0•cox•weff/leff
IN(M.IBD)	バルク－ドレインダイオードの電流
IN(M.IBS)	バルク－ソースダイオードの電流
IN(M.GBD)	dIBD/dVBD
IN(M.GBS)	dIBB/dVBS
IN(M.CBD)	結合静電容量（バルク－ドレイン）
IN(M.CBS)	結合静電容量（バルク－ソース）
IN(M.CGB)	ゲート－バルク固有静電容量
IN(M.CGS)	ゲート－ソース固有静電容量
IN(M.CGD)	ゲート－ドレイン固有静電容量
IN(M.W)	配線幅
IN(M.L)	配線長
IN(M.WEFF)	実行幅
IN(M.LEFF)	実行長
IN(M.COX)	eps_ox/tox•weff•leff

この表はMOSトランジスタのインスタンス名がMであることを仮定しています。

例：

```
* in circuit.NSX
MOUT OUT GCAS 0 0 NTYP W=120U L=1.2U

* in circuit.PAT
TRACE DC ID(MOUT) IN(MOUT.GDS) IN(MOUT.GM)
```

```
TRACE DC {CG = IN(MOUT.CGB) + IN(MOUT.CGS)}
TRACE DC {CGBN = IN(MOUT.CGB)/IN(MOUT.COX)}
```

レベル4 (BSIM)で使用できる内部変数:

IN(M.GDS)	ソース－ドレインコンダクタンス(dI/dVDS)
IN(M.GM)	ゲートトランスコンダクタンス(dI/dVGS)
IN(M.GMBS)	バルクトランスコンダクタンス(dI/dVBS)
IN(M.LOGGDS)	log(IN(GDS))
IN(M.LOGM)	log(IN(GM))
IN(M.LOGGMBS)	log(IN(GMBS))
IN(M.VTH)	スレッショルド電圧
IN(M.VDSAT)	飽和電圧
IN(M.BETA)	µ0•cox•weff/leff
IN(M.IBD)	バルク－ドレインダイオードの電流
IN(M.IBS)	バルク－ソースダイオードの電流
IN(M.GBD)	dIBD/dVBD
IN(M.GBS)	dIBB/dVBS
IN(M.CBD)	結合静電容量（バルク－ドレイン）
IN(M.CBS)	結合静電容量（バルク－ソース）
IN(M.W)	配線幅
IN(M.L)	配線長
IN(M.WEFF)	実行幅
IN(M.LEFF)	実行長
IN(M.COX)	eps_ox/tox•weff•leff
IN(M.QG)	ゲート電荷
IN(M.IQG)	ゲート電荷からの電流
IN(M.QB)	バルク電荷
IN(M.IQB)	バルク電荷からの電流
IN(M.QD)	ドレイン電荷
IN(M.IQD)	ドレイン電荷からの電流
IN(M.QS)	ソース電荷
IN(M.CGGB)	dQG/dVG
IN(M.CGDB)	dQG/dVD
IN(M.CGSB)	dQG/dVS
IN(M.CGBB)	dQG/dVB
IN(M.CBGB)	dQB/dVG
IN(M.CBDB)	dQB/dVD
IN(M.CBSB)	dQB/dVS
IN(M.CBBB)	dQB/dVB
IN(M.CDGB)	dQD/dVG
IN(M.CDDB)	dQD/dVD
IN(M.CDSB)	dQD/dVS
IN(M.CDBB)	dQD/dVB

この表はMOSトランジスタのインスタンス名がMであることを仮定しています。

例：

```
* in circuit.NSX
MOUT OUT GCAS 0 0 NTYP W=120U L=1.2U

* in circuit.PAT
.TRACE DC ID(MOUT) IN(MOUT.GDS) IN(MOUT.GM)
.TRACE DC {CG = IN(MOUT.CGB) + IN(MOUT.CGS)}
.TRACE DC {CGBN = IN(MOUT.CGB)/IN(MOUT.COX)}
```

ダイオードモデルパラメータ

構文

```
.MODEL type D [param=…]
```

ダイオードモデルのキーワードは“D”です。

ダイオードモデルは、SPICE 2G.6にインプリメントされているモデルと同じです。

ダイオードモデルのパラメータ

名前	デフォルト値	単位	説明
IS	1e-14	A	飽和電流
RS	0.0	Ohm	オーム抵抗
N	1.0	-	放出係数
CJ	0.0	F/m^2	結合静電容量
VJ	1.0	V	結合電位
MJ	0.5	-	CJ(V)式の指数
FC	0.5	-	フォーワード／リバース式の係数
TF	0.0	s	遷移時間
EG	1.11	eV	エネルギギャップ
XTI	3.0	-	IS(T°C) 式の指数
BV	-	V	破壊電圧
IBV	1e-10	A	破壊電流

例：

```
.MODEL MYDIODE D Is=1e-16 RS=12 N=1.07
```

バイポーラトランジスタモデル

構文

```
.MODEL type NPN|PNP [param=…]
```

バイポーラトランジスタモデルのキーワードはNPNまたはPNPです。バイポーラジャンクショントランジスタモデルは、SPICE 2G.6にインプリメントされているGummel-Poonモデルです。

バイポーラトランジスタモデルのパラメータ

名前	デフォルト値	単位	説明
IS	1e-16	A	飽和電流
BF	100.0	-	順方向ベータ
IKF	0.00	A	BF高電流ロールオフ
NF	1.0	-	順方向放出係数
BR	1.0	-	逆方向ベータ
IKF	0.0	A	BR 高電流ロールオフ
NR	1.0	-	逆方向放出係数
ISE	0.0	A	B-E 漏れ飽和電流
NE	1.5	-	B-E 漏れ放出係数
ISC	0.0	A	B-C 漏れ飽和電流
NC	2.0	-	B-C 漏れ放出係数
RB	0.0	Ohm	ベース抵抗
RBM	0.0	Ohm	最小ベース抵抗
IRB	0.0	Ohm	RB=(RB+RBM)/2の時の電流
RC	0.0	Ohm	コレクタ抵抗
RE	0.0	Ohm	エミッタ抵抗
CJE	0.0	F	ベース－エミッタ結合静電容量
VJE	0.75	V	ベース－エミッタ結合電位
MJE	0.33	-	CJE(VBE) 式の指数

CJC	0.0	F	ベース－コレクタ結合静電容量
VJC		0.75	V　　ベース－コレクタ結合電位
MJC	0.33	-	CJC(VBC) 式の指数
XCJC	1.0	-	内部ベースに接続されるCBCの小数部
CJS	0.0	F	コレクター－バルク結合静電容量
VJS	0.75	V	コレクター－バルク結合電位
MJS	0.33	-	CJS(VCS) 式の指数
VAF	1e6	V	順方向初期電圧
VAR	1e6	V	逆方向初期電圧
FC	0.5	-	順・逆方向静電容量計算式の係数
EG	1.11	eV	エネルギ・ギャップ
TB,	0.0	-	ベータ（T°C）式の指数
XTI	3.0	-	IS(T°C) 式の指数
TF	0.0	sec	順方向遷移時間
TR	0.0	sec	逆方向遷移時間
XTF	0.0	-	順方向遷移時間のバイアス依存度
VTF	0.0	V	順方向遷移時間のVBC依存度
ITF	0.0	A	順方向遷移時間のIC依存度
TRE1	0.0	$°C^{-1}$	RE(T)の温度係数
TRE2	0.0	$°C^{-2}$	RE(T2)の温度係数
TRB1	0.0	$°C^{-1}$	RB(T)の温度係数
TRB2	0.0	$°C^{-2}$	RB(T2)の温度係数
TRC1	0.0	$°C^{-1}$	RC(T)の温度係数
TRC2	0.0	$°C^{-2}$	RC(T2)の温度係数
TRM1	0.0	$°C^{-1}$	RBM(T)の温度係数
TRM2	0.0	$°C^{-2}$	RBM(T2)の温度係数

バイポーラトランジスタの内部変数へのアクセス

トランジスタの端子電流[IC(), IB(), IE()]に加えて、特別な.TRACEディレクティブや.PRINTディレクティブの構文を使用すればトランジスタの内部変数（FTやCBE、CBCのような変数）にアクセスすることができます。

➔ オンラインヘルプのディレクティブの項の.TRACEディレクティブの説明を参照して下さい。

内部変数にアクセスする一般的な構文は、：

```
.TRACE TRAN|DC IN(Qname.VARNAME)
```

です。

ここで、Mnameはバイポーラトランジスタのインスタンス名、VARNAME は有効な内部変数の一つです。このセクションの最後にある表と例を参照して下さい。

このセクションでは、使用できる内部変数を示します。この変数は、.opファイルのバイアスセクションに表示される主な小信号パラメータです。

バイポーラトランジスタの使用できる内部変数:

IN(Q.RB)	ベース抵抗
IN(Q.CBE)	ベースエミッタ静電容量
IN(Q.CBC)	ベースコレクタ静電容量
IN(Q.CSC)	サブストレートコレクタ静電容量
IN(Q.CBCX)	外部ベースコレクタ静電容量
IN(Q.GPI)	1/rpi
IN(Q.LOGGPI)	log(gpi)
IN(Q.GM)	1/gm
IN(Q.LOGGM)	log(gm)
IN(Q.GMU)	1/gmu
IN(Q.LOGGMU)	log(gmu)

IN(Q.GZERO)	1/r0
IN(Q.LOGGZERO)	log(gzero)
IN(Q.BETADC)	ic/ib DC
IN(Q.BETADC)	ic/ib AC
IN(Q.FT)	遷移周波数
IN(Q.LOGFT)	log(FT)

この表は、バイポーラトランジスタのインスタンス名がQであることを仮定しています。

例：

```
* in circuit.NSX
Q12 COL BASE EM QTY

* in circuit.PAT
TRACE DC IC(Q12) INQ12.CBE) IN(Q12.CBC)
TRACE DC IN(Q12.LOGFT)
```

電界効果トランジスタモデル

構文

```
.MODEL type NJF|PJF [param=…]
```

JFETのキーワードはNJFまたはPJFです。

JFETモデルのパラメータ

名前	デフォルト値	単位	説明
VTO	-2.0	V	スレッショルド電圧
BETA	1e-4	A/V^2	トランスコンダクタンス
LAMBDA	0.0	V^{-1}	チャンネル長補正値
IS	1e-14	A	結合飽和電流
N	1.0	-	結合放出係数
RD	0.0	Ohm	ドレイン直列抵抗
RS	0.0	Ohm	ソース直列抵抗
CGD	0.0	F	ゲート－ドレイン静電容量（vgd=0の時）
CGS	0.0	F	ゲート－ソース静電容量（vgs=0の時）
M	0.5	-	cgd/s(vgd/s)式の指数
PB	1.0	V	結合電位
FC	0.5	-	順方向バイアス式の係数
VTOTC	0.0	V/°C	VTOの温度係数
BETATCE	0.0	%/°C	BETAの温度係数
XTI	3.0	-	IS(T°C)式の指数
ISR	0.0	A	リコンビネーション電流パラメータ
NR	0.0	-	放出係数
ALPHA	0.0	V^{-1}	イオン化係数
VK	0.0	V	イオン化ニー電圧

アナログ／デジタルインタフェース

概要

アドバンストSimは、混在（ミクスト）モードシミュレータです。従って、アナログ部分とデジタル部分の混在する回路のシミュレーションを行うことができます。アナログ回路とデジタル回路のインタフェース部の動作を定義するために、インタフェースノードを使用します。インタフェースノードは、アナログデバイスとデジタルデバイスの両方を接続します。このノードは、電圧と論理ストレングスインターバルを持つ論理値との両方を伝達することができます。

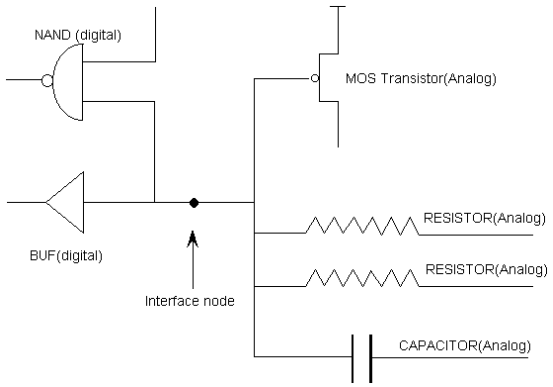
等価な方法としては、論理レベルを電圧で伝える方法です。また、逆に電圧を論理レベルで伝える方法もあります。電圧の論理レベルへの変換は、非常に簡単です。スレッショルド関数を使用して、電圧が論理“ハイ”または“ロー”、“不定”かどうかを決定します。このやり方は、インターフェースノードに接続された入力ピンを持つデジタルゲートにより使用されます。

インターフェースノードを駆動する出力ピンを持つゲートでは、もっと複雑になります。デジタルゲートの出力ステージと等価なアナログ回路が使用されます。デフォルトの等価回路を使用することもできますし、インタフェースデバイスやインタフェースモデルを使用してその回路をカスタマイズすることもできます。

設定可能な項目

定義によればインタフェースノードは、アナログデバイスとデジタルデバイスとを接続するノードです。しかし、設定可能な項目に制限があります。

デジタル出力ピンが１つの場合

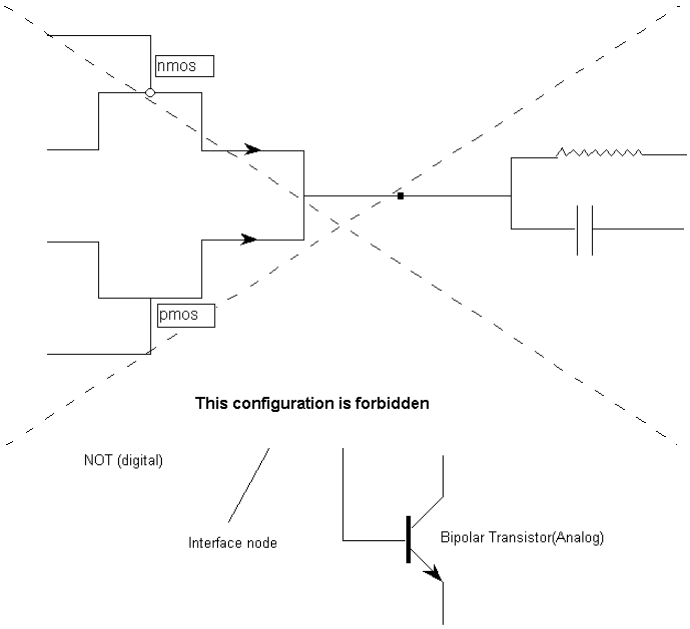


設定1.インタフェースノードは４つのアナログデバイス（より正確には、４つのアナログピン）と２つのデジタル入力ピンに接続される。デジタル出力ピンは、インタフェースノードを駆動しません。任意の数のデジタル入力ピンに接続することもできます。

しかし、インタフェースノードには、デジタル出力ピンしかありません。以下にその様子を図示します：

設定2.インタフェースノードは３つのアナログデバイスと２つのデジタル入力ピン、一つのデジタル出力ピンに接続されます。

インタフェースノードに接続されるデジタル出力ピンの数の制限のために、下図のような配置はできません。



このような配置は禁止されています。すなわち、２つのデジタル出力ピンが、ひとつのインタフェースノードに接続されています。.

ゼロディレイの無いループ

2番目の制限は、ループに関するものです。ゼロディレイループとは、ディレイの無いデジタルゲートでできた閉じたループのことです。ゼロディレイループは純粋なデジタル記述に存在します。（しかし、推奨しません。）そして、問題も発生します。しかし、ゼロディレイループ内にインタフェースノードを記述することはできません。すなわち、インタフェースノードからゼロディレイデジタルゲートを通り、再びインタフェースノードに戻るようなデジタル経路を記述することはできません。

デジタルクロックはインタフェースノードを駆動できません

デジタルスティミラスは通常、周期パターンに使用される.CLKディレクティブや任意パターンやバスパターンに使用されるWAVEFORM... FINISH節を持つ回路に適用することはできません。

➔ 回路プリミティブとスティミラスの章のデジタルスティミラスの項を参照して下さい。

実際、デジタルスティミラスはデジタル“ゲート”の特別な形式で、入力をもたない特別なパターンを与える一個の出力です。これらを“クロックタイプ”ゲートと呼ぶことにします。それらはアナログの世界では独立した電圧ソースと等価です。

インタフェースノードをこのような“クロックタイプ”ノードで駆動することはできません。これは、これらのゲートがインタフェースノードを駆動することができないような特別な方法で扱われるからです。

インタフェースを“クロックタイプ”信号で駆動したい場合、“クロックタイプ”ゲートとインタフェースノードとの間に（bufタイプのゲートがいい候補です。）ノーマルゲートを挿入して下さい。以下の例を参照して下さい。

例：

```
// in circuit.NSX
>>> VERILOG
module top(ITFNODE, CLKNODE);
  inout ITFNODE, CLKNODE;
  buf DUM(ITFNODE, CLKNODE);
```

```
  not N1(N34, ITFNODE);
endmodule
>>> SPICE
R1 ITFNODE BIASM 100K
// in circuit.PAT
.CLK CLKNODE 0 S0 10 S1 .REP 20
```

インタフェースノードの識別

ネットリストがアドバンストSimにより分析される場合、インタフェースノードは自動的に識別されます。アナログピンとデジタルピンの両方を持つノードはすべて（インタフェースノードには任意の数のデジタル入力ピンと一つまでのデジタル出力ピンを持つことができることを思い出して下さい。）インタフェースノードとして認識されます。例えば以下のネットリストを見てみましょう。

```
...
>>> SPICE
R1 OUT 0 10K
...
// R1 is a resistor (an analog primitive)

>>> VERILOG
module top(OUT, ... );
 output OUT;
 ...
 buf B1(OUT, N24);
endmodule
...
// B1 is a buffer (“buf” is a digital primitive)
```

このネットリストの分析中、ノードOUTはアナログデバイスピン（レジスタR1のピン）とデジタルデバイス（B1の出力ピン）とに接続されているので、インタフェースノードとして識別されます。

デフォルトインタフェースデバイスと明示的なインタフェースデバイス

一旦識別されると、各インタフェースノードより、“インタフェースモデル”を参照する“インタフェースデバイス”が取得されます。インタフェースデバイスは明示的に指定することもできますし、アドバンストSimにより自動的にインタフェースデバイスを作成することもできます。

明示的にインタフェースデバイスを指定するには、ネットリストの“N”デバイス（下記参照）の行を追加して下さい。このインタフェースデバイスにより、.MODELステートメントを使用して明示的に与えられるインタフェースモデルが参照されます。このやり方は、“実際”のシミュレーションを行ううえでもっともよく使う方法になると思います。

一方、インタフェースノードに何も指定しなかった場合、アドバンストSimにより、デフォルトのインタフェースモデルを参照するインタフェースデバイスが自動的に作成されます。このデフォルトのインタフェースモデルは、5V CMOSテクノロジー用のものです。（その他のテクノロジーに合うように、デフォルトのモデルのパラメータをオーバーライドするディレクティブもあります。）

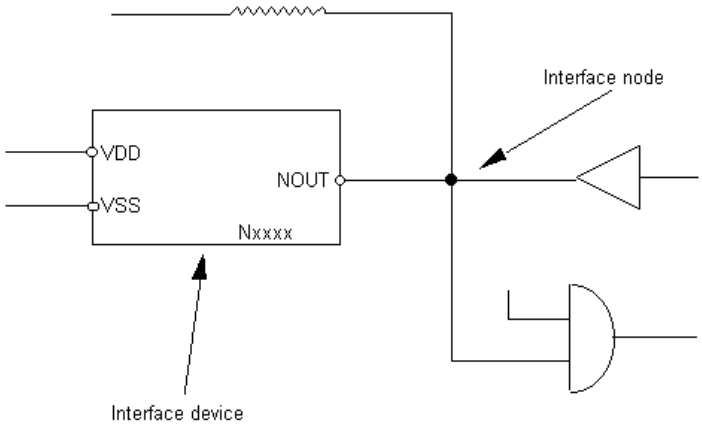
以下で詳細は示しますが、明示的なインタフェースデバイスはデフォルトのものと少し違います。明示的なインタフェースデバイスは、回路の3つのノード；インタフェースノードそのものと二つの電源ノードを接続する必要があります。これらの電源ノードは通常各電圧ソース（“V”タイプのアナログデバイス）に接続されます。デフォルトのインタフェースデバイスは電源ノードを使用しません。しかし、ハイとローの出力電圧の値が必要です。以下のスケマティックを参照して下さい。

➔ インタフェースノードはすべてそれに接続されるインタフェースデバイスが必要です。そのインタフェースデバイスは明示的に指定するかデフォルトのものを使用するかのどちらかです。

明示的インタフェースデバイスとモデル

複数のテクノロジー（例えばCMOSとTTL）を混在する事がある時や複数の電源（例えば、0/+5ボルトと-15/+15ボルト）が必要な場合、おそらく明示的なインタフェースデバイスと関連のあるモデルが必要になります。

明示的なインタフェースデバイスは特別なアナログデバイスでインタフェースノードの動作を変更するためにそのインタフェースノードに接続されます。



明示的なインタフェースデバイスに３つのピンがあり、インタフェースモデルを参照します。.

ネットリストの中で明示的なインタフェースデバイスを使用するための構文を以下に示します：

```
Nxxxx nout nvss nvdd modelname
```

Nxxxxはインタフェースデバイスのインスタンス名です。nvddとnvssは電源供給ピンで、通常独立の電圧ソース（“V”デバイス）に接続され、デジタルゲートの正と負の電源供給に影響します。noutはインタフェースノードに接続されます。modelnameはデバイスに関連付けられるインタフェースモデルの名前です。このモデルは、トランジスタやダイオードモデルと同様に.MODELステートメントで指定されます。その目的は、アナログからデジタルへの変換モデルとデジタルからアナログへの変換モデルに使用されるパラメータを指定することです。複数の明示的なインタフェースデバイスで同じインタフェースモデルを参照することができます。

➔ インタフェースデバイスは、インタフェースノードの動作を指定する以外には使用することができません。

例：

```
// in circuit.NSX:
...
>>> SPICE
R1 OUT 0 10K
NTTL OUT VSS VCC TTLITF
...
// R1 is a resistor (an analog primitive)
>>> VERILOG
module top(OUT, ... );
 output OUT;
 ...
 buf B1(OUT, N24);
endmodule
// B1 is a buffer (“buf” is a digital primitive)

// in circuit.PAT
V_VCC VCC 0 5.0V
V_VSS VSS 0 0.0
.MODEL TTLITF ITF TPLH=34.7n ....
...
```

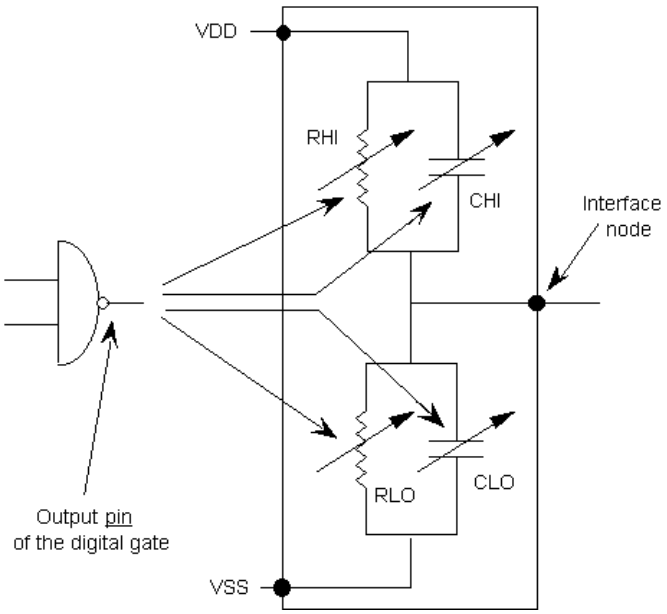
```
/*
```

NTTLインタフェースデバイスにより、OUTインタフェースノードの動作が変更されます。TTLITFインタフェースモデル（.MODELステートメント）のパラメータが使用されます。

```
*/
```

明示的なインターフェースデバイスの回路

明示的なインタフェースデバイスは、デジタルゲートの出力段をモデル化した小さなアナログ回路です。それぞれの電源供給ピンは可変抵抗と可変コンデンサを通りインタフェースノードピンに接続されます。



インタフェースデバイスは、デジタルゲートの出力段のモデルです。抵抗値とコンデンサ容量は、ゲートによりインタフェースノードが駆動される値と強さにより変更されます。出力ピンの論理値と論理ストレングスはこのピンを含むノードの値とストレングスとは若干違います。

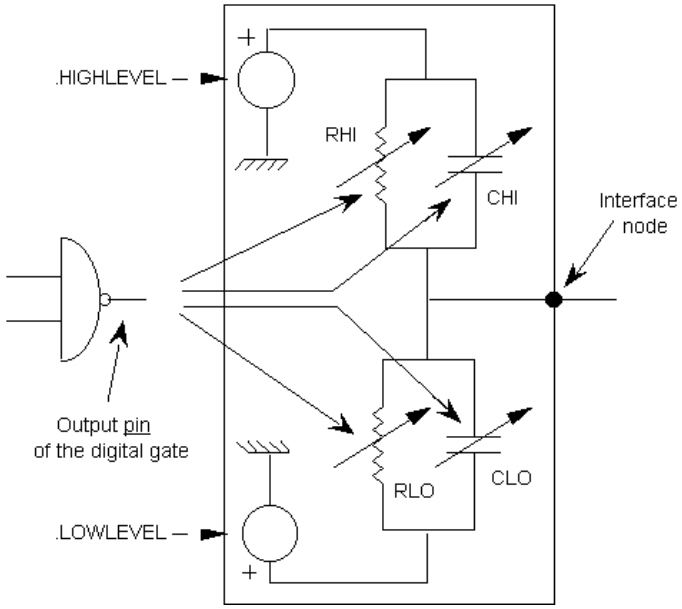
デジタルゲートによりインタフェースノードが駆動される場合、その出力ピンはあるストレングスの論理値に駆動されます。値とストレングスまたはどちらかが変化すると、インタフェースデバイスの抵抗値とコンデンサ容量もそれに応じて変化します。

出力ピンの各デジタルステートに対して、RHI, CHI, RLO, CLOの値が対応します。多くのステートが可能です。（論理ステートは16の値を持つストレングスにより定義されます。回路プリミティブとスティミラスの章のデジタルプリミティブの項を参照して下さい。）単純化のためにストレングスの幅をインタフェースノードに適した幅に変換しても、複雑なモデルの場合、88もの値を持つことになる。実際には、これら88個の値になかには多くの同じ値が存在します。

➔ 抵抗やコンデンサは、出力ピンのステートに応じて変わります。インタフェースノードのステートではないことを常に覚えていて下さい。

デフォルトインタフェースデバイススケマティック

デフォルトインタフェースデバイスは実際にはデジタルゲートの出力段をモデル化した小さなアナログ回路です。電源供給ピンは使用されません。代わりに内部電圧ソース（アクセス不可）が可変抵抗と可変コンデンサを通ってインタフェースノードに接続されます。これらの電圧ソースの値は、パターンファイルの.HIGHLEVELディレクティブと.LOWLEVELディレクティブで設定されます。



インタフェースデバイスは、デジタルゲートの出力段のモデルです。抵抗値とコンデンサ容量は、ゲートによりインタフェースノードが駆動される値と強さにより変更されます。出力ピンの論理値と論理ストレングスはこのピンを含むノードの値とストレングスとは若干違います。

➔ 明示的なインタフェースデバイスと比較して、出力デジタルピンのステートにより抵抗とコンデンサの組み合わせは単純です。以下のインタフェースモデルパラメータのセクションを参照して下さい。

➔ ストレングスの幅を変更して最大のストレングスまでストレングスを減らす（既知の値の信号の場合）か、最大のストレングスに同期する（未知の値の信号の場合）にします。これは、0ドメインと1ドメインの両方です。

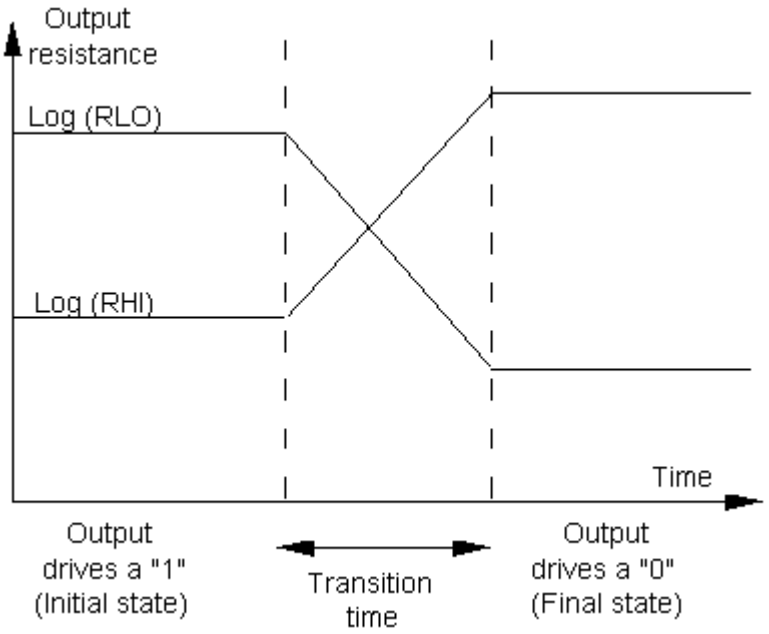
遷移時間

出力ピンがあるステートから他のステートへ変化する場合、ある時間が経過しますが、この時間を遷移時間と呼びます。抵抗RHIとRLOの値は、この遷移時間中に初期状態の値から最終状態の値へと指数関数的に変化します。コンデンサ容量は遷移時間中線形的に変化します。

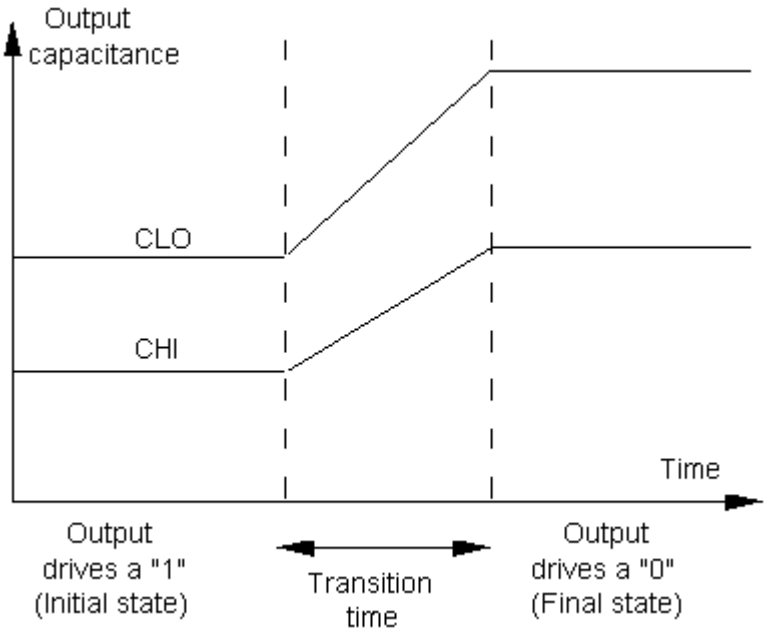
➔ この遷移時間はデジタルゲートの伝播遅れではありません。ゲートの伝播遅れは別に追加します。遷移時間は出力の立ち上がりと立ち下がり時間です。

４種類の遷移時間を指定することができます。“tplh”, tphl”, “tpz”, “tpnz”です。tplhとtphlは、ある“strong”ステート（“strong”とはsupplyまたはstrongを意味します）から他の“strong”ステートへの遷移の間を遷移時間とします。tpzは“strong”ステートから“weak”ステートへの遷移時間です。tpnzは"weak"ステートから“strong”ステートへの遷移時間です。下の図は遷移中に何が起こっているかを図示したものです。

出力ピンがハイからローへ遷移するときの抵抗の変化の典型例です。



➔ 上の図で示されるように、“1”に駆動されたときのRLOとRHIの値と“0”に駆動された時のRHIとRLOの値は違います。



出力ピンが遷移中のコンデンサの典型的な変化。

➔ ステートに関係なくCLOとCHIの値は、ほとんどのインタフェースモデルで同じです。

インターフェイスモデルパラメータ

インタフェースモデルは.MODELステートメントで指定されます。このステートメントはパターンファイル（*circuit*.PAT）またはライブラリファイルに記述することができます。

また、ネットリストファイル（*circuit*.NSX）にも記述できますが、あまりお勧めできません。

インターフェースモデルに使用される構文は、次の通りです：

```
.MODEL modelname ITF [paramname=paramvalue] ...
```

キーワード ITF は.MODEL ステートメントで指定されたインターフェースモデルの3番目インジケーターとして使用し、これはインターフェースモデルを定義します。modelname はインターフェースモデルの名称となり、インターフェースデバイスから参照されます。

パラメータとして次のものが指定できます：

名称	デフォルト値
TPLH	.LRISEDUAL
TPHL	.LRISEDUAL
TPZ	.LRISEDUAL
TPNZ	.LRISEDUAL
VINLOW	.UNKZONE
VINHIGH	.UNKZONE
VINMIN	.VINRANGE
VINMAX	.VINRANGE
RTOLOW_???	.RTOLOW_???
RTOHIGH_???	.RTOHIGH_???
CTOLOW_???	.LOWCAPA
CTOHIGH_???	.HIGHCAPA

➔ インターフェースモデルのパラメーターのデフォルト値は、パターンファイルにディレクティブを記述する事で指定します。（これらのディレクティブの記述についてはオンラインヘルプの*ディレクティブ*のトピックを参照して下さい。）

遷移時間

パラメータTPHL, TPLH, TPZ 及びTPNZのデフォルト値は.LRISEDUAL ディレクティブで指定します。このディレクティブは、インターフェースノードの立ち上がり／立ち下がり時間を指定します。パターンファイルに.LRISEDUAL ディレクティブない場合、1nsが使用されます。

例1：

```
...
MODEL MYCMOS ITF TPLH=12.5n
...
.LRISEDUAL 17n
...
```

TPLH の値は .MODEL ステートメントで指定されますが、TPHL, TPZ 及び TPNZ は指定されていません。TPLHは12.5nで、TPHL, TPZ, TPNZは（.LRISEDUALディレクティブで指定され）17nとなります。

例2：

```
...
.MODEL MYCMOS ITF TPLH=12.5n
...
```

TPLHの値は .MODELステートメントで指定されますが、TPHL, TPZ,及びTPNZは指定されていません。TPLHは12.5nで、TPHL, TPZ, TPNZは（.LRISEDUALディレクティブがないので）1nになります。

入力電圧

パラメータVINLOW とVINHIGHは、入力のスレッショルド電圧を指定するために使用されます。これらのパラメータは、インタフェースノードにより駆動されるデジタルゲートによってのみ使用されます。インタフェースノードにより駆動されるデジタルゲートは、VINLOW とVINHIGHパラメータの値と電圧とを比較してインタフェースノードが“ ロー ”

か“ 不定 ”か“ ハイ ”かを決定します。VINLOW よりも低い電圧はすべて“ ロー ”（論理“0”）とみなされます。VINHIGHよりも高い電圧はすべて“ ハイ ”（論理“1”）とみなされます。VINLOW とVINHIGHの間の電圧は、“ 不定 ”（論理“X”）とみなされます。

VINLOW とVINHIGHのデフォルト値は.UNKZONEディレクティブにより与えられます。

パラメータVINMINとVINMAXを使用して入力電圧の範囲を指定します。デフォルトでは、VINLOW (VINHIGH)よりも低い（高い）電圧はすべてインタフェースノードで駆動されるデジタルゲートにより論理“ ロー ”（“ ハイ ”）とみなされます。パラメータVINMIN(VINMAX)が指定される場合、VINMIN(VINMAX)よりも低い（高い）電圧はすべて同様のゲートにより論理“ 不定 ”とみなされます。これにより、入力に非常に高い電圧または非常に低い電圧のあるデジタルゲートでこのような電圧を有効な論理値（“ ロー ”または“ ハイ ”）の変わりに、“ 不定 ”とみなすことができます。VINMINとVINMAXのデフォルト値は.VINRANGEディレクティブで指定します。

➔ オンラインヘルプの *ディレクティブ*のトピックのこれらのディレクティブの説明を参照して下さい。

出力抵抗

TPLHのRTOLOW_???とRTOHIGH_???により出力ピンの22のステートの出力抵抗を指定します。例えば、出力ピンが“Strong 0”のステートの時、RTOLOW_ST0=10の場合、RLOの抵抗は10オームです。RTOLOW_???とRTOHIGH_???のデフォルト値はRTOLOW_???とRTOHIGH_???ディレクティブで指定します。これらのディレクティブがパターンファイルに記述されていない場合、グローバルデフォルト値が使用されます。

➔ これらのディレクティブについては、オンラインヘルプのディレクティブの項を参照して下さい。

出力キャパシタンス

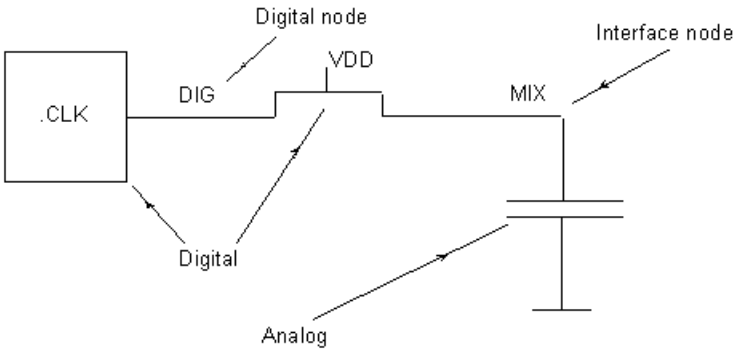
CTOLOW_???とCTOHIGH_???パラメータを使用して異なるステートの出力静電容量の値を指定します。これらのデフォルト値は、.LOWCAPAと.HIGHCAPAディレクティブにより設定されます。.LOWCAPAと.HIGHCAPAディレクティブが指定されない場合、すべての静電容量はデフォルトでゼロが指定されます。

➔ これらのディレクティブについては、オンラインヘルプのディレクティブの項を参照して下さい。

例

インタフェースノードの簡単な例を説明します。この例では、nmosデジタルゲートを .CLKタイプのデジタル信号で駆動します。nmosゲート出力ピンはコンデンサを駆動します。nmosゲートが入力のストレングスレベル（論理値だけではない）を伝達します。さらに .CLK信号と任意の値／ストレングスとを組み合わせることができるので、インターフェースノードは任意の値／ストレングスの組み合わせで駆動することができます。

nmosゲートのコントロール入力を« supply1 VDD »ネットに接続し、nmosゲートが常に« on »になるようにします。



例題のスケマティック

対応する*circuit*.NSXファイルと*circuit*.PATファイルを以下に示します。：

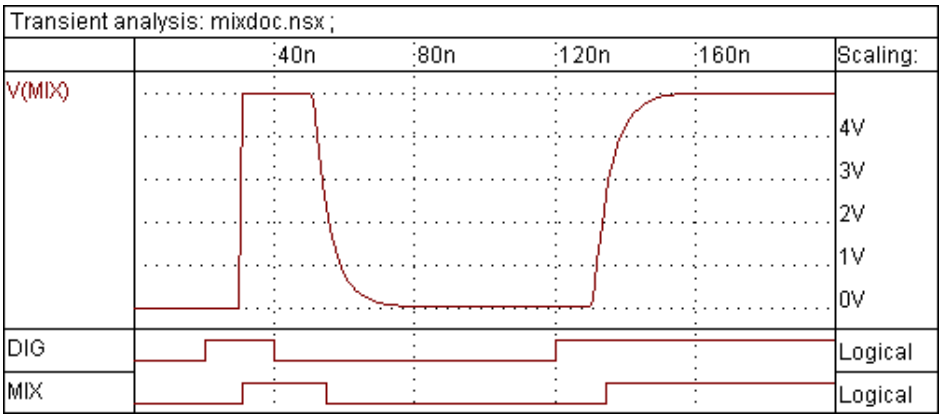
```
------------------------------ mix.nsx ---------
>>> VERILOG
`timescale 1ns / 10ps
module top(DIG, MIX);
 inout DIG, MIX;
 nmos #(10, 10) (MIX, DIG, VDD);
 supply1 VDD;
endmodule

>>> SPICE
CLOAD MIX 0 0.5P
------------------------------ mix.pat ---------

.CLK DIG 0 ST0 20 ST1 40 WE0 120 WE1
VDD VDD 0 5

.LPRINTALL
.PRINTALL
.TRAN 1N 200N
.TRACE  TRAN   V(MIX)
.LTRACE TRAN   DIG
.LTRACE TRAN   MIX
```

.LTRACEディレクティブを使用すれば、MIXをデジタル信号として表示することができます。また、.TRACEディレクティブを使用すればMIXをアナログ信号として表示することができます。.PRINTALLディレクティブと.LPRINTALLディレクティブにより、シミュレーション中のすべてを保存することができます。異なるパラメータを与えた場合の過渡現象のシミュレーションを実行して、デフォルト値がそれぞれの場合にどのように使用されるかを説明します。



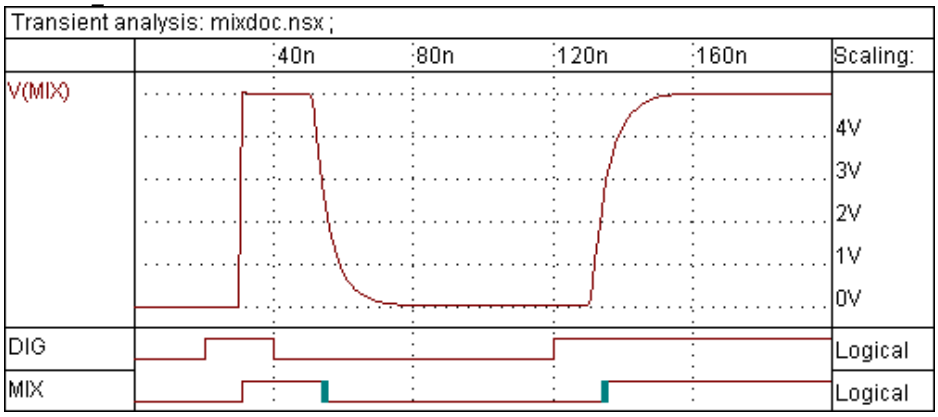
過渡シミュレーション結果－シミュレーション1.

このシミュレーションは、パラメータすべてがデフォルト値です。MIXインタフェースノードにインタフェースデバイスは使用されません。従って、デフォルトのインタフェースデバイスとデフォルトのインタフェースモデルがMIXインタフェースノードに与えられます。（.OPファイル内のモデルパラメータの値を参照して下さい。）MIXの最初の立ち上がりエッジは、DIGがローからハイにDrivingストレングスで変化したときです。デフォルトのDrivingストレングスは抵抗が1Kです。従って、コンデンサ容量が0.5Pファラッドのコンデンサを DELAYゲートの出力ピンにより簡単に駆動できます。その後、DIGはResistive 0さらにResistive 1になります。デフォルトのresistiveストレングスは100Kです。従って、RC ディレイの遷移が始まります。パターンファイルに .LTIMESCALEディレクティブが記述されていないので、デフォルトの時間スケール1nsがDELAYゲートの遅れに適用され10nsの遅れとなることに注意して下さい。この遅れはMIXノードの遷移の時に観察できます。また、MIXノードはノードの電圧が2.5ボルトを通過すると、0から1または1から0へ変化することに注意して下さい。これは .UNKZONEディレクティブが与えられていないためです。.HIGHLEVELディレクティブまたは .LOWLEVELディレクティブが記述されていないため、出力電圧の最大値と最小値には0と5が使用されます。

以下のステートメントをパターンファイルに追加しましょう：

```
.UNKZONE 2 3
```

もう一度、過渡シミュレーションを実行します。



過渡シミュレーション結果－シミュレーション2.

何が起こりましたか？.UNKZONEディレクティブが入力されたので、電圧がローかハイかを決定するためのスレッショルドの値が変更されました。実は、“.UNKZONE 2 3”とは、2Vと3Vの間の電圧は論理“ 不定 ” (X)とすることを表わします。これは、MIXの波形で確認することができます。

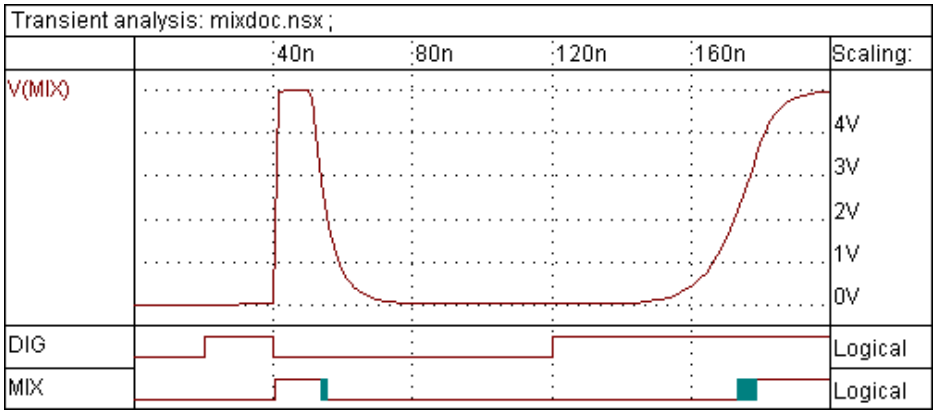
次に、明示的なインタフェースデバイスとそれに関連付けられたモデルをネットリスト内に追加しましょう。

```
-------------------------------- mix.nsx ---------
>>> VERILOG
`timescale 1ns / 10ps
module top(DIG, MIX);
 inout DIG, MIX;
 nmos #(10, 10) (MIX, DIG, VDD);
 supply1 VDD;
endmodule

>>> SPICE
CLOAD MIX 0 0.5P
NMIX MIX 0 VDD MYINTERF

-------------------------------- mix.pat ---------
.CLK DIG 0 ST0 20 ST1 40 WE0 120 WE1
VDD VDD 0 5
.LPRINTALL
.PRINTALL
.TRAN 1N 200N
.TRACE  TRAN   V(MIX) MIN=-5.0007175E-001 MAX=5.5008493E+000
.LTRACE TRAN   DIG
.LTRACE TRAN   MIX
.UNKZONE 2 3
.MODEL MYINTERF ITF TPLH=50N VINHIGH=3.5
```

そして、再び過渡シミュレーションを実行して下さい：



過渡シミュレーション結果－シミュレーション3.

三番めのシミュレーションでは、明示的なインタフェースデバイスNMIXがMIXノードに接続され、さらに電源供給ノード0（グランドノード）とVDD （パターンファイルで宣言される独立した電圧ソース）に接続されます。このインタフェースデバイスにより、パターンファイル内で宣言されるMYINTERFインタフェースモデルが使用されます。この明示的なインタフェースデバイスにより、MIXインタフェースノードの動作をカスタマイズすることができます。MYINTERFインタフェースモデルでは、多くのパラメータを指定しません。50nsに設定されるTPLH値と3.5Vに設定されるVINHIGH値だけです。

これは、出力ピンがハイからローへ遷移中に、インタフェースデバイスの抵抗と静電容量が50nsで切り替わることを意味します。デフォルトの遷移時間は .LRISEDUALディレクティブ（与えられているなら）により指定されます。.LRISEDUALディレクティブが与えられない場合、遷移時間は1nsに設定されます。

VINHIGHを3.5Vに設定することにより、“ 不定 ” 領域の上限が3Vから3.5Vに拡大されます。.MODELステートメントでVINLOWパラメータが指定されていないため、VINLOWは .UNKZONEディレクティブからの値（2V）を引き継ぎます。

この50nsの遅れはシミュレーション3の波形ではノードMIXの最初の立ち上がりエッジで顕著に現れています。

トラブルシューティング

この章には、いくつかのヒントとよくある質問とそれに関する回答が記載されています。

ウインドウのチップス

ファイルシステムのエラーが発生しないように、時々スキャンディスクを実行して下さい。ファイルシステムが壊れていると、アドバンストSimでは、結果がおかしくなる事が有ります。出力ファイルがおかしくなるだけでなく、システムがハングアップすることも有ります。

アドバンストSimの実行中にクラッシュした場合、ウインドウズを終了し、ディスクをスキャンするプログラムを実行して下さい。ウインドウズベースのプログラムでは、たまにファイルシステムを壊すことが有ります。多くの問題を避けるために定期的にディスクのスキャンを行って下さい。

オペレーティングポイントが得られない

.OP ディレクティブについてはオンラインヘルプの *Directives* トピックと *Getting Started with Advanced Sim*のオペレーティングポイント解析のトピックを参照して下さい。

もしBSIMモデルを使っているならパラメータを NO に、NBとNDを0.0にして下さい。

また、可能ならばトランジスタに出来る限り簡単なモデルを使います。簡単なモデルで *circuit*.OPファイルを作成してから再度、元のモデルを使用し、"Start off with .op file"オプションを使用するか、あるいは .USEOPディレクティブを使用します。

“Bad news! time step …” というメッセージが表示された

回路の接続やコンポーネントの値、モデルパラメータをチェックして下さい。

パターンファイルから全てのオプションディレクティブを除き、すべての設定が "初期状態"になっているか確認して下さい。

全てのアナログノードのグランドに、パターンファイル内の.CAPAMINディレクティブも含めて、小容量のキャパシタを加えてみます。これは過渡解析に対して有効です。

Minimum time step（Simulate-Setup Simulator-Transient）の値を減少させて見ます。

Nominal time step（同じダイアログ）の値を減少させて見ます。

電圧ソース（PULSE、PWL）の立上がり、立下がり時間を増加して見ます。

"Current accuracy"（Simulate-Setup Simulator-Transient）を増加して見ます。1e-9（初期値）から1e-8などに変更します。

もし、インターフェースノード（混在モードの解析）がある場合、それらのノードは過渡期のシャープさが少ないので .LRISEDUALパラメータ又はインターフェースモデル TPLH、TPHL、TPZ、TPNZを増加して見ます。

EKVモデルでは NQS=0を使います。

過渡解析に時間がかかる

パターンファイルから全てのオプションディレクティブを除き、設定が "初期状態" になっているか確認して下さい。

ポイント数が十分足りているか確認します。（これは.TRAN命令内の接続／表示の割合です）もし設定が “.TRAN 1p 1m” になっているなら1,000,000,000ポイントの設定を行っていることになります。（Simulate-Setup Simulator-Transientダイアログより、DurationとDisplay項目を変更して下さい）

Internal time stepでの設定値を確認して下さい。nominal time step値を最小にしてはいけません。maximum time stepの値を大きくしてみて下さい。（Simulate - Setup Simulator - Transientダイアログより）

"Current accuracy"（Simulate - Setup Simulator - Transient）を増加して見ます。

.EXCLUSIVEディレクティブの使用して下さい。インタラクティブな操作はできなくなりますが、シミュレーションのスピードは向上するでしょう。

もしデザインが大きい場合は .PRINTALLと .LPRINTALLディレクティブは使わない方がいいかもしれません。必要な信号を選んで .PRINTと .LPRINTディレクティブを使用して下さい。

必要な波形だけを表示して下さい。

.RELAXディレクティブとリラクゼーションモードを使います。

アナログ機能の代わりに方程式定義ソースを使う事を考慮します。

何回か続けて拡大表示後、使用不可になる

.DIGITSディレクティブを使い、表示される数値の桁数を増加します。

数値が丸められる

.DIGITSディレクティブを使い、表示される数値の桁数を増加します。

回路をロードすると"Too many analog sources"というメッセージが表示される

.MAXSOURCESディレクティブを使って電圧、電流電源の最大数を増加します。

ルートディレクトリにt_xxファイルが作成される

それらのファイルは削除してもかまいません。C:\TMPのディレクトリを作成し、Autoexec.batファイルで設定を行なえば、それらはC:\TMPに作成されます。SETコマンドによるテンポラリディレクトリの環境設定は、次のような記述となります。

```
SET TMP = C:\TMP
```

時々、C:\TMPディレクトリ内に残っているt_xxファイルを削除して下さい。

AC小信号解析を行ったが何も起こらない

ソースのAC Magnitudeの値（一般的には "1" ）が設定されているか確認して下さい。

トレースする数が大きくなると何も表示されない、クラッシュする

config.sysファイルにコマンドFILES=50を入力する。

RS、JKフリップフロップのシミュレーションを行うには

LIBESディレクトリのシミュレーションモデルを使います。

WAVEFORMを使用しないで、デジタルクロックで休止状態'を作るには

2つの .CLK「ゲート」を同じノードに接続する必要が有ります。

例：

```
.CLK CLOCK 0 D0 10 D1 .REP 20
.CLK CLOCK 0 Z0 1000 D0 1350 Z0
```

0から1000までがノーマルのクロックで、次に1000から1350まで0で、1350からまた、クロックとして再スタートします。

個々の .CKTファイルからの .LIBファイルを作成するには

特定の部品が記述された .CKTファイルから、テキストエディタでコンテンツをコピー&ペーストしてライブラリファイル（拡張子 .LIBにより識別）を作成する事が出来ます。

シミュレーションの間、信号が選択出来ない

大規模なシミュレーションであれば、対話的な操作は損なわれます。あるいはセレクトできないのは、ズームモードであるためかもしれません。これらのモードの間は、カーソルがノーマルの形状とは違っています。それ以外の場合はシミュレーションの実行中です。

ズームモードから抜けるには、マウスの右ボタンをクリックするかESCキーを押して下さい。それから信号を選択して見て下さい。

アドバンストSim ファイル

入力ファイル

シミュレーションの入力データには少なくとも「ネットリスト」ファイルと「パターン」ファイルという、2つのファイルが存在します。前者は回路とそれらの連結を示しており、後者はシミュレーションに必要なスティミラス（アナログ電圧と電流、デジタルパターン）とコントロールディレクティブを指定します。

更にアドバンストSimは、ライブラリファイルからデータ情報を読み込むことができます。ライブラリファイルはサブサーキット定義、モデル定義、マクロ定義とユーザー定義を含むことができます。*circuit*.NSXファイルまたは*circuit*.PATファイルに必要な要素がない場合、アドバンストSimはライブラリファイルから得ようとします。ライブラリ要素の定義は、他のライブラリ要素を参照する事もできます。

➔ ライブラリディレクトリとファイルの指定についての情報は、*ライブラリとデバイスモデル*の 章を参照して下さい。

ネットリストファイル(*circuit*.NSX)

シミュレーターの主な入力はネットリストファイル（*circuit*.NSX）です。以後、このマニュアルでは、*circuit*.NSXをシミュレーションネットリスト名として使用します。*circuit*.NSXファイルは結線情報（netlist）を含まなければなりません。アドバンストSimは、1つのメインとなるネットリストファイルとライブラリファイルを融合させ、回路として扱います。（回路の詳細は*階層記述*と*ライブラリとデバイスモデル*の章を参照して下さい。）ネットリストファイルはアドバンストスケマティックから自動的に生成されます。ネットリストは設計者がマニュアルで作成、編集することができますが、これはお勧めできません。

➔ ネットリストで使用される構文は、*回路プリミティブとスティミラス*、そして*階層記述*の章で詳しく説明されています。

circuit.NSXファイルの要約は以下のとおりです。（用語の定義に関しては*階層記述*の章を参照して下さい。）

- サブサーキット定義（.SUBCKT ステートメントについては、*階層記述*を参照して下さい。）
- デバイスモデル定義（.MODEL ステートメントについては*ライブラリ*と*デバイスモデル*を参照して下さい。）サブサーキット定義外でモデルが定義されている場合、その範囲は共通であり、階層のあらゆるレベルに使用できることを意味しています。
- Verilog-HDL モジュール（モジュール定義については、*階層記述*を参照して下さい。）
- Verilog-HDL ユーザー定義プリミティブ
- アナログプリミティブ、電圧と電流ソース、サブサーキット定義以外のものは、アナログ階層の« top-level »です。デジタル階層のtop-levelはインスタンスモジュールではないものによって定義されています。*階層記述*を参照して下さい。

パターンファイル(*circuit*.PAT)

それぞれのネットリストは、それと結び付けられているパターンファイルを持っています。パターンファイルは、スティミラスの詳細と、シミュレーターのディレクティブを含んでいます。個々の入力データをネットリストセクション（*circuit*..NSX）とパターンセクション（*circuit*.PAT）分けるのは、アドバンストSimでの決まりです。SPICEやそれから派生したシミュレーターでは、すべての入力データが1つのファイルにまとめられるので、（また、すべての出力データは、1つの出力ファイルにまとめられます。）それらを使い慣れた方々は、少し不便に思われるかもしれません。

このファイルが存在しない場合、自動的に生成されます。すでにパターンファイルがある場合には、訂正されません。

多くのパラメータは、アドバンストSimのダイアログボックスで修正することができます。パラメータを修正した場合は、パターンファイルが更新され、次のシミュレーションから有効になります。

BILファイル(*circuit*.BIL)

この自動的に作成されるファイルは、コンポーネントのパラメータ（.PARAMディレクティブ）を定義しています。これは .PARAMSWEEPディレクティブにより、スイープさせる事ができます。コンポーネントが許容値（例:100K/5%）を含んでいる場合には、アドバンストSimは、.MONTECARLOディレクティブも生成します。BILファイルの内容は、インクルード文によりシミュレーションを行う際、パターンファイルに取り込まれます。

IPAファイル(*circuit*.IPA)

これは自動的に生成されるファイルです。スケマティックのテキストフレームの最初の行に .IPAが含まれていた場合には、このテキストフレームの内容は、 *circuit*.IPAファイルの先頭部分にコピーされます。シミュレーションディレクティブをスケマティックで入力したい場合には、.IPAのテキストフレームに記入します。また、このファイルはスケマティックにコンポーネントとしてデジタルスティミラスを含んでいる場合であれば、 .CLKディレクティブが記入されます。それらはパターンファイルで記載されたタイミング詳細のリファレンスです。IPAファイルの内容は、インクルード文によりシミュレーションを行う際、パターンファイルに取り込まれます。

ライブラリファイル

サブサーキットライブラリファイル(.CKT)

サブサーキットライブラリファイルはサブサーキット定義（SPICE用語の中の .SUBCKTステートメント）を含んでいます。ファイルの基本名は、サブサーキットのファイル名と一致している必要が有ります。.CKTファイルは*advsim*.INIファイルの[Library]セクションでリストされているディレクトリに存在するか、またはパターンファイルの中の .LIBディレクティブで直接指定します。

➔ サブサーキット定義の作成方法については*階層記述*の章を、ライブラリへサブサーキットを入力する方法については*ライブラリとデバイスモデル*の章を、*advsim*.INIについては、*アドバンストSimファイル*を参照して下さい。

Verilog-HDL ライブラリファイル(.V)

Files拡張子 .Vのファイルは Verilog-HDLライブラリファイルです。それらには、シングルモジュール、またはユーザー定義プリミティブ（Verilog-HDL用語による）の定義が含まれています。ファイルの基本名は、モジュールの名前と一致していなければなりません。この一致は、大文字と小文字は区別しません。.Vファイルは*advsim*.INIファイルの[Library]セクションにリストされているディレクトリに存在するか、またはパターンファイルの中の .LIBディレクティブで直接指定します。

➔ サブサーキット定義の作成方法については*階層記述*の章を、ライブラリへサブサーキットを入力する方法については*ライブラリとデバイスモデル*の章と*アドバンストSimファイル*の章を参照して下さい。

モデルライブラリ(.MDL)

拡張子が .MDLのファイルは、モデルライブラリのファイルです。これらは、モデル（SPICE用語の.MODELステートメント）定義を含んでいます。ファイルの基本名は、モデルの名前と一致している必要が有ります。MDLファイルは、*advsim*.INIファイルの[Library]セクションにリストされているディレクトリに存在するか、またはパターンファイルの中の .LIBディレクティブで直接指定します。

➔ .MODELステートメントとライブラリへサブサーキットを入力する方法については*ライブラリとデバイスモデル*の章を、*advsim*.INIについては*アドバンストSimファイル*の章を参照して下さい。

マクロライブラリ(.MAC)

拡張子が .MACのファイルは、マクロライブラリファイルです。これらは、マクロの定義を含んでいます。ファイルの基本名は、マクロの名前と一致していなくてはなりません。MACファイルは*advsim*.INIファイルの[Library]セクションにリストされているディレクトリに存在するか、またはパターンファイルの中の .LIBディレクティブで直接指定します。

➔ マクロについては*マクロ*の章を、モデルライブラリについては*ライブラリとデバイスモデル*の章を、*advsim*.INIについては*アドバンストSimファイル*の章を参照して下さい。

混在ライブラリファイル(.LIB)

拡張子が .LIBのファイルは混在ライブラリファイルです。これらには、サブサーキット（.SUBCKT）、モデル（.MODEL）、モジュール（module）、ユーザー定義プリミティブ（primitive）、そしてマクロ（DEFINE_MACRO）の定義が含まれています。LIBファイルは、それらのエレメントの番号、命令、簡潔な連結を含むことができます。これは、ライブラリエレメントの蓄えをシングルファイルが含む個々のエレメントと同じ様にコンパクトにすることができます。しかし、1つのエレメントにアクセスすることは、他のライブラリファイル（例えばCKTファイルとVファイル）より遅くなり、LIBファイルの中にエレメントが含まれていることが明らかではありません。LIBファイルは*advsim*.INIの[Library]セクションにリストされているディレクトリに存在するか、またはパターンファイルの中の .LIBディレクティブで直接指定します。

➔ ライブラリとデバイスへのエレメントの入力については*ライブラリとデバイスモデル*の章を、*advsim*.INIについては*アドバンストSimファイル*の章を参照して下さい。

出力ファイル

アドバンストSimは多くの出力ファイルを生成します。それらの個々のファイルに、同種の情報が含まれています。それぞれの解析はそれ自身の出力ファイルを生成します。例えば、過渡解析は*circuit*.TMFファイルを生成し、DC解析は*circuit*.DMFファイルを生成し、それら個々のファイルにはそれぞれの結果が含まれています。

➔ ダイアログリストボックスでファイルを見つけやすくするために、行う解析別にディレクトリを分けることをお勧めします。

レポートファイル(*circuit*.RPT)

circuit.RPTファイルは回路をロードした後に、アドバンストSimにより生成されます。回路をロードする度に、前回の*circuit*.RPTファイルは確認を行わずに上書きされます。これには回路で発見されたワーニング、エラーとライブラリの識別の概要が含まれます。

ワーニングが生成された場合には、プロンプトウインドウで報告されます。

➔ *circuit*.RPTファイルにワーニングが含まれている場合には、修正を行って下さい。

オペレーティングポイントファイル(*circuit*.OP)

circuit.OPファイルは、オペレーティングポイント解析とパワーアップ解析から生成されます。これにはstatic解析を実行する際の回路の状態の詳細が含まれています。（インターフェイスノードとアナログ電圧、デジタルとインターフェイスノードのレベル／強度、トランジスタバイアスの詳細情報）

アドバンストSimは*circuit*.OPファイルがすでに存在する場合、*circuit*.OPファイルを上書きする前に、*circuit*.BOPというファイル名で自動的に保存されます。これは、*circuit*.OPファイルが「最優先」として処理され、アドバンストSimは*circuit*.OPファイルをスターティングポイントとして再利用することができるので時間を節約する事ができます。この様にバックアップを行う事は、操作エラーに対する保護の第一段階としての役割があります。

➔ *circuit*.OPファイルが作成されるのは3つのケースがあります。最初の1つは、オペレーティングポイント解析を明確に実行する場合。2つめは、オペレーティングポイント解析を行っていなくても、それまでに過渡解析、小信号、またはノイズ解析を行った場合にも作成されます。3つめはパワーアップ解析を行なった場合です。それぞれの解析の詳細は、オンラインヘルプの*ディレクティブ*のトピックを参照して下さい。

タイミング違反ファイル(*circuit*.TVL)

ネットリストの中に $setup()、$hold()、$width()などの文が含まれている場合、過渡解析中に*circuit*.TVLが生成されます。シミュレーション中に発生したタイミング違反の詳細が含まれています。違反メッセージには、混乱したノード、違反が発生した時間、違反の数量が含まれています。これらのステートメントについては、Verilog-HDLのLRMを参照して下さい。

過渡解析波形ファイル(*circuit*.TMF)

このバイナリファイルには、過渡解析で生成されたアナログ波形（電圧と現在のvs時間）が含まれています。ファイルに含まれる波形は、パターンファイルの中の .PRINTディレクティブにリストされたものです。パターンファイルの中に .PRINTALLディレクティブがある場合、*circuit*.TMFファイルにすべてのアナログ波形が保存されます。シミュレーション中、またはシミュレーション後に パワーアップウインドウ、または一般的なウインドウに追加される波形は、このファイルから抽出されます。

➔ ノード、コンポーネント数が多いシミュレーションでは、.PRINTALLディレクティブは、大きなファイルを生成することがありますので注意して下さい。

パワーアップ解析波形ファイル(*circuit*.OMF)

このバイナリファイルには、パワーアップ解析で生成されたアナログ波形（電圧、電流、時間）が含まれています。ファイルに含まれる波形は、パターンファイルの中の .PRINTディレクティブにリストされたものです。パターンファイルの中に .PRINTALLディレクティブがある場合、*circuit*.OMFファイルにすべてのアナログ波形が保存されます。シミュレーション中、またはシミュレーション後に パワーアップウインドウ、または一般的なウインドウに追加される波形は、このファイルから抽出されます。

➔ ノード、コンポーネント数が多いシミュレーションでは、.PRINTALLディレクティブは、大きなファイルを生成することがありますので注意して下さい。

小信号解析波形ファイル(*circuit*.AMF)

このバイナリファイルは、小信号解析によって作成されるアナログ波形（実際のパートと想像上の電圧と電流 vs周波数）を含んでいます。ファイルに含まれる波形は、パターンファイルの .PRINTディレクティブでリストされます。パターンファイルの中に .PRINTALLディレクティブがある場合、すべてのアナログ波形は *circuit*.AMFファイルに保存されます。シミュレーション中、またはシミュレーション後にパワーアップウインドウ、または一般的なウインドウに追加される波形は、このファイルから抽出されます。

➔ ノード、コンポーネント数が多いシミュレーションでは、.PRINTALLディレクティブは、大きなファイルを生成することがありますので注意して下さい。

DCトランスファー解析波形ファイル(*circuit*.DMF)

このバイナリファイルはDC過渡解析で作成されたアナログ波形（電圧と電流、スイープ電圧）を含んでいます。ファイルに含まれる波形は、パターンファイルの .PRINTディレクティブでリストされます。パターンファイルの中に .PRINTALLディレクティブがある場合、すべてのアナログ波形は*circuit*.DMFファイルに保存されます。シミュレーション中、またはシミュレーション後に、パワーアップウインドウ、または一般的なウインドウに追加される波形は、このファイルから抽出されます。

➔ ノード、コンポーネント数が多いシミュレーションでは、.PRINTALLディレクティブは、大きなファイルを生成することがありますので注意して下さい。

バイナリノイズ解析ファイル(*circuit*.NMF)

このバイナリファイルはノイズ解析で生成されます。これには、 ONOISE, INOISE, DB（INOISE） DB（ONOISE）の4つが含まれます。ファイルは .PRINTまたは .PRINTALLディレクティブに関わらず、常にノイズ解析で作成されます。

デジタル波形ファイル(*circuit*.BHF)

このバイナリファイルは、過渡またはパワーアップ解析で作成されるデジタル波形（論理レベルと強さ vs 時間）を含んでいます。ファイルに含まれる波形は、パターンファイルの .LPRINTALLディレクティブでリストされます。パターンファイルの中に .LPRINTALLディレクティブがある場合、すべてのデジタル波形は*circuit*.BHFファイルに保存されます。シミュレーション中、またはシミュレーション後に、過渡ウインドウ、または一般的なウインドウに追加される波形は、このファイルから抽出されます。

➔ ノード、コンポーネント数が多いシミュレーションでは、. LPRINTALLディレクティブは、大きなファイルを生成することがありますので注意して下さい。

➔ デジタル波形は、個別のファイルに保存されます。デジタル波形のための能率的なフォーマットは、イベントスタイル（*circuit*.BHF）であり、テーブルスタ イルを使用しているアナログ波形（*circuit*.TMF）のフォーマットと全く異なっています。

過渡解析ファイル(*circuit*.HIS)

circuit.HISファイルは、テキストフォーマットの中の過渡解析の結果を含んでいます。これはパターンファイルに .CREATEHISFILEディレクティブがある場合、過渡解析の最後に作成されます。*circuit*.HISファイルは、解析結果をHISフォーマットから .PATフォーマットに変換することができます。次の*circuit*.H2Pファイルを参照して下さい。

アナログノイズ解析ファイル(*circuit*.NZE)

circuit.NZEファイルは、ノイズ解析で作成されます。これは冗長なテーブルであり、セレクトされたそれぞれの周波数に対する、コンポーネントのノイズのリストの並べ変えです。これにも指定されたバンド幅を超えるノイズの統一した値が含まれています。

➔ ノイズ解析の詳細（オンラインヘルプの*ディレクティブ*のトピック .NOISE）を参照して下さい。

circuit.H2Pファイル

The Reports - Convertで*circuit*.HISを*circuit*.H2Pに変換することができます。出力波形の注釈を含んだこのファイルは、*circuit*.PATファイルに含むことが適している*circuit*.HISファイルの中から見つけることができます。この概要は、論理シミュレーションの結果を他のシミュレーションの入力パターンとして使用できます。

➔ .CREATEHISFILEディレクティブについてはオンラインヘルプの*ディレクティブ*のトピックを、Reports - Convertの詳細に関しては*Getting Started with Advanced Sim*を参照して下さい。

アーカイブファイル(*circuit*.ARC)

.ARCHIVEディレクティブがパターンファイルの中にあれば、回路をロード（Load Circuit コマンド）した際に*circuit*.ARCファイルが作成されます。このファイルはロードする実際の回路で使用でれているすべてのテキストファイルのコピーを含んでいます。これは、スタンドアローンのシミュレーションを作成する際に有用であり、ライブラリリファレンスを必要としません。

➔ .ARCHIVEディレクティブについてはオンラインヘルプの*ディレクティブ*のトピックを参照して下さい。

Advsim.INI 初期設定ファイル

概要

この章では、“*advsim*.INI”ファイル内のセクションと記述を要約します。 この初期設定ファイルは、様々なオプションの指定に使用されます。*advsim*.INIファイルはセクションで構成されており、それぞれのセクションは一つまたは多くのエントリを含んでいます。

各セクションは［］に囲まれたキーワードから始まっています。それは次のセクション又はファイルが終わる最後のセクションまで拡張します。全てのセクションは自由に設定でき、セクション内のエントリもまた自由に設定できます。セクションとエントリは大文字・小文字は関係ありません。

空白行も認められます。*advsim*.INIファイル内でコメントを記述するには、セミコロン(;)を使用します;

```
; これはコメントです
```

advsim.INIはどこにありますか?

Advsim.INI は \windows ディレクトリにあります。

波形表示色の設定

[colors]セクションでは、波形表示に使われる色を定義します。表示色はcolor1からcolor5までの5色の色を利用できます。それぞれの色に関しては、RGB形式による3つの数値で希望する色を定義します。定義は0～255の範囲で行います。

```
[Colors]
color1 = r1 g1 b1
color2 = r2 g2 b2
color3 = r3 g3 b3
color4 = r4 g4 b4
color5 = r5 g5 b5
```

例：

```
[Colors]
; these are the default settings:
color1 = 0 255 255
color2 = 0 255 0
color3 = 255 0 0
color4 = 200 200 0
color5 = 255 0 255
```

フォントの設定

[Fonts]セクションにはグラフィックウインドウでセレクトされたフォントが記述されています。このセクションは読書きします。すなわちアドバンストSimは起動時やメニューアイテムからEdit - Change Fontを選択し、フォントを修正したときに更新され、このセクションを読みます。これは比較的スタンダードなフォント選択ダイアログで、システム内で利用できるフォントと選択された一つを表示しています。

advsim.INI 内の [Fonts] セクションを編集するのは、タイプミス等を起こしやすいので、なるべく避けて下さい。フォントの変更を行う時は安全の為、ダイアログを使用して下さい。

[Fonts]セクションの記述：

```
[Fonts]
graphicsfontname =
graphicsfontstyle =
graphicsfontsize =
```

例：

```
[Fonts]
graphicsfontname = arial
graphicsfontstyle = bold
graphicsfontsize = 9
```

プリントオプション

[Print] セクションは、プリント時のオプション設定が記述されています。

scalingfactorはプリント出力時のスケールを使います。それは%内を明細に記します。例としてscalingfactor = 50にすると出力が半分に減ります。初期設定値が100であれば、プリントはスケールされません。

fillpaperpage は紙の上でできる限り広いエリアを占めて作画するスケールを希望するかどうかを示す時に使います。

markers エントリはプリント時の波形図を加えたマークが必要かどうかを示す時に使います。この考慮は全てのケースの波形図を区別します。初期設定値はyesです。

blackandwhite エントリは白黒のみでプリントする事を強いる場合に使い、不十分な（または価値のない）命令を避け多くのプリンタ上にカラーの波形図を与えます。初期設定値はyesです。

[Print]セクションの記述：

```
[Print]
scalingfactor = n
```

```
fillpaperpage = yes | no
markers = yes | no
blackandwhite = yes | no
```

設定の保存

初期設定では、File - CloseまたはFile - Quitコマンドを選択すると、パターンファイルは更新され、現在表示されているシミュレーション画面が .TRACEと .LTRACEディレクティブに反映します。

もし、この動作の変更したい場合、`saveoncloseall = no`または `saveonquit = no` エントリをこのセクション中に付け加えます。初期設定値はこれらのエントリはyesになっています。

アドバンストSimが終了する前に確認を行うには`confirmquit = yes`エントリをこのセクションに加えるとできます。初期設定では確認は必要としません。

[AutoSave]セクションの記述：

```
[AutoSave]
saveoncloseall = yes | no
saveonquit = yes | no
confirmquit = yes | no
```

初期設定

`[defaults]`セクションでは、次に行うシミュレーションの初期設定を記述します。

初期設定の階層キャラクタは、キャラクタが階層ノードと、インスタンス名を構築する時使います。もし、何も指定されなければ初期設定の階層キャラクタはドット ‘.’です。

階層キャラクタを修正するには、互換性のある目的が必要になります。例として、多くのシステムは構築した階層名の下線キャラクタを使います。もし、いろいろな初期設定の階層キャラクタを使うのであれば、初期設定の階層に必要とするキャラクタをセットします。

`[defaults]` セクションの記述例：

```
[Defaults]
DefaultHierchar = _
```

➔ 下線またはドットと同様の階層キャラクタのどちらかを使う事を薦めます。‘A’または‘B’のようなキャラクタは使えません。.HIERCHAR ディレクティブ　についても参照して下さい。

ライブラリディレクトリの設定

`[Library]` セクションでは、ライブラリ要素が存在するディレクトリを指定する事ができます。それぞれのエントリは `= yes`または `= no` の後でディレクトリ名で構築します。それらのディレクトリは `= yes` の詳細を調べることに対して印を付けます。構文は“`C:\down\here`”です。

[Library]セクションの記述は：

```
[Library]
directory = yes | no
directory = yes | no
...
```

例：

```
[Library]
c:\user\joe\simul\pro\libs\npn = yes
c:\trash\old\npn = no
```

➔ アドバンストSimがライブラリ要素を検索するときは上から順に調べます。

➔ *ライブラリとデバイスモデル*の章を参照して下さい。

プロテルセクション

このセクションには、回路図シンボルの部品から直接参照するモデルライブラリの場所が明記されています。ディレクトリc:\client98\sch98\library\simにあるライブラリ中のそれぞれのシンボルは、適切なモデルファイルへの相対的なパスが記述されています。pristel_root_directoryで指定されている文字列にこのパスが付加され、回路が初めてアドバンストSimにロードされる時、*circuit*.BIL ファイルに .LIBディレクティブとして記入されます。

[protel]セクションの記述は：

```
[protel]
pristel_root_directory=C:\CLIENT98\SIM98\LIBRARY
```

アクセスコードセクション

[Access]セクションには正確に入力されたアクセスコードが保管されています。