# Verilog-Aモデルから最高のパフォーマンスを得るために

## はじめに

Verilog-A で記述されたモデル、特にトランジスタ・モデルはそのままでもシミュレート可能ですが、次のポリシーに従うことにより SmartSpice からさらに高いパフォーマンスを得られるようになります。

新規のモデルも、このポリシーに従うことでパフォーマンスが高くなります。

## 概要

1. パラメータ値の計算
2. 内部ノードの数
3. ノイズ・ブロック
4. 関数の代わりにマクロを使用
5. GMIN

## ポリシー

### 1. パラメータ値の計算を 1 回のみにする

```
parameter length = 2;
parameter width = 3;
real       area;

analog begin
    area = length * width;
    ….
end
```

ここでは反復の都度、area が何度も計算されています。しかし、最初以降の計算は必要ありません。

これは、パラメータとして宣言されている length と width から area が求められるためであり、実行時にこれらの値の変更は不可とされているため (LRM 2.3.1 Sec. 3.4)、また変数の値は後続する反復の間保持されるため (LRM 2.3.1 Sec. 5.6.1.3)、area の計算をシミュレーション開始時 1 回のみにすることが可能となります。

次の変更により、望ましい結果を得ることができます。

```
parameter length = 2;
parameter width = 3;
real       area;

analog begin
    @(initial_step) begin
     area = length * width;
     ….
     end
     ….
end
```

initial_step イベントがトリガとなるイベント・ブロックは、シミュレーション開始時に 1 回だけ実行されます (LRM 3.2.1 Sec 5.10.2)。Verilog-A も initial_step を認識し、割り当てられた変数を異なるカテゴリに分けるため、大きなメリットとなります。

ADMS を直接サポートするいくつかのモデル (PSP、BSIMCMG) では、initial_step の挿入が簡単です。これらのモデルは、`define を使用することで ADMS 以外での使用に柔軟に対応できます。

```
`ifdef ADMS
  `define INITIAL_MODEL     @(initial_model)
  `define INITIAL_INSTANCE
                            @(initial_instance)
`else
  `define INITIAL_MODEL
  `define INITIAL_INSTANCE
`endif

analog begin
  `INITIAL_MODEL
  begin
    area = length * width;
    …
  end
end
```

ADMS を対象としない場合、`INITIAL_MODEL および `INITIAL_INSTANCE はスペースで置換されますが、これは望ましいことではありません。次のような簡単な変更により、対処することができます。

```
`ifdef ADMS
   `define INITIAL_MODEL @(initial_model)
   `define INITIAL_INSTANCE
                    @(initial_instance)
`else
   `define INITIAL_MODEL @(initial_step)
   `define INITIAL_INSTANCE
                      @(initial_step)
`endif
```

### 2. 内部ノードの数を最小限にする

モデル内の寄生素子によりシミュレーション速度が遅くなるため、モデルは一般的にさまざまなレベルにインプリメントされます。ベーシックレベルは最も高いレベルよりインプリメントされる寄生素子が少なく、シミュレーションに使用する内部ノードが少ないため、シミュレーション速度が速くなります。Verilog-A LRM は SPICE のレベル・パラメータに相当する機能を持たないため、同様の結果を得るために別の方法を案出しました。簡単な方法の 1 つとして、レベル毎に異なるモデル名(モジュール名）をつける方法があります。さらに複雑な方法では、与えられたパラメータ値に基づき Verilog-A パーサが自動で適切なレベルを求めることが必要となります。Verilog-A のコード・スタイルが異なると、1 つの Verilog-A ソース・モデルからでも結果は違うレベルになります。条件コンパイルおよびノード折りたたみという 2 つのスタイルについて、次に述べられています。

条件コンパイル（例 : EKV3 モデル）では、局所的に `ifdef を使用することで各レベルに固有のモデル名 ( モジュール名 ) が与えられ、内部ノード数が正確にコントロールされます。

```
`ifdef DC
          module xyz      (d,g,s,b);
`endif
`ifdef RF
          module xyz_rf   (d,g,s,b);
`endif
….
`ifdef RF
          electrical internal_d,
          internal_s, internal_b;
`endif
….
`ifdef RF
          I(internal_d,internal_s) <+ ….;
`endif
          endmodule
```

モジュールを RF という定義でコンパイルする場合、固有のモジュール名は xyz_rf となり、そのモデルには正確に 3 個の内部ノードが追加され、内部ノードに対する代入定義が存在します。これらの機能は RF が定義されている条件下で現れるため、条件コンパイルと呼ばれます。コンパイル・フェーズ中、条件コンパイル・スタイルが適用されることに注意してください。

ノード折りたたみ ( 例 :PSP、BSIMCMG) では、特別な Verilog-A 構文 if-then-else が使用され、Verilog-A パーサはシミュレーション・セットアップ・フェーズ中にこの構文を認識し作用することで、モデルを特定のレベルに区別します。

```
parameter rg;


if(rg != 0)
  I(g, internal_g) <+ V(g, internal_g)/rg;
else
   V(g, internal_g) <+ 0;
```

このコードでは、寄生 rg が 0 ではないとき、rg という値の抵抗をノード internal_g と g の間に配置します。寄生がない、すなわち rg が 0 のとき、0 ボルトの電圧源によりノード間の短絡が発生します。if-then-else 構文は定数パラメータ rg により設定されるスイッチを形成します。Verilog-A パーサはセットアップ中、rg が 0 のときスイッチは必ず ON で、ノード g と internal_g が短絡し 2 つのノードが 1 つに折りたたまれて、ノードの数が減少するという認識がなされることが求められています。したがって、ノード折りたたみと呼ばれます。

0 ボルトの電圧源は端子ノードを短絡させてノードを折りたたむ一方で、Verilog-A LRM では別の目的で 0 ボルトの電圧源を使用することがあります。具体的には、電流プローブが必要とされている場合に 0 ボルトの電圧源が使用されます (LRM 2.3.1 Sec 5.4.2.1)。Verilog-A はノード折りたたみを行いませんが、LRM に従い 0 ボルトの電圧源を解釈します。

Verilog-A が if-then-else 構文を正しくシミュレートした場合、rg の値が 0 でも 0 でなくても、内部ノードの数が改善することはありません。internal_g 用に常時 1 つのノードがあり、さらには電圧源を通る電流用に内部ブランチ・ノードが予備としてあります。rg が 0 でノードが 1 つだけ必要な場合でも、常に 3 つのノードが存在します。

内部ノードの数を減らすため、Verilog-A モデルの if-then-else 構文を検索し、該当の場合は条件コンパイル・スタイルに置き換えることを推奨します。

### 3. ノイズ解析されていない時はノイズ・ブロックを実行しない

ノイズ解析が要求されていないとき、ノイズ解析に代わる計算の値は 0 であるため、回避することが可能です。幸い Verilog-A モデルでは、多くの場合ノイズ解析ステートメントはグループにまとまって出現します。

```
I(…)  <+ white_noise(…);
I(…)  <+ white_noise(…);
….
```

そして、次の簡単な解決法により、必要な時まで計算の実行を回避することができます。

```
if (analysis("noise")) begin
   I(…)  <+ white_noise(…);
   I(…)  <+ white_noise(…);
   ….
end
```

### 4. 関数呼び出しの代わりに同じ意味のマクロを使用する

関数呼び出しの実行は高負荷です。関数が頻繁に呼び出されている場合、マクロと置き換えてください。

```
analog function real myexp;
   input x;
   real x;
   begin
  myexp = exp(x);
   end
endfunction

analog begin
   y = myexp(z);
end
```

これは、簡単にマクロ展開に変更することができます。

```
`define myexp(_x) (exp(_x_))

analog begin
   y = `myexp(z);
end
```

### 5. GMIN に注意！

gmin の値が必要なモデルでは、モデル中に gmin の値をハードコードできます。

```
`define GMIN 1.0e-12
```

または

```
parameter real GMIN = 0.0;
```

この場合、SmartSpice のオプションである回路の gmin の変更は、モデルに作用しません。モデルを SmartSpice の gmin オプションに対応させるには、GMIN への参照を次と置き換えます。

```
$simparam("gmin")
```

## まとめ

Verilog-A モデルにこれら 5 つのポリシーを適用することでコードが最適化され、SmartSpice のシミュレーション・パフォーマンスが最高になります。Verilog-A 言語に慣れているエンジニアであれば、30 分程度で既存のモデルにポリシーを適用可能です。これは最適化されたモデルをシミュレーションすることで節約される時間により、簡単に取り返すことができます。