Verilog-AMS 実用モデリング講座 第3回

イベント・ベース・モデル，PLL，サンプル・ホールドの記述方法を理解する

デバイスの記事

桜井 至，石塚総一郎

前回（本誌2003年5月号，pp122-131）は，アナログ演算子や算術関数，アナログ素子の記述方法について解説しました．今回は，イベント・ベース・モデルを記述するために重要なクロス関数やトランジション関数などについて説明します．また，PLL回路やサンプル・ホールド回路の記述例も紹介します．（編集部）

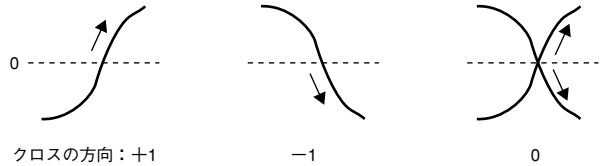
イベント・ベース（event based）のモデリングの対象となる回路は2種類あります．一つはディジタル回路です．ディジタル回路は，主にVerilog-Dを用いてモデリングします．二つ目はスイッチト・キャパシタ構造のアナログ回路です．Verilog-AMSを用いると，高速なディジタル回路のアナログ的な動作を，より詳細に記述することができます．

① イベント・ベース・モデルのキーとなる関数

ここでは，イベント・ベース・モデリングを行うために重要な構文について説明します．

●イベント・ベース・モデリングに必要な基本関数

イベント・ベース・モデルにおいて基本となるcross，last_crossing，timer，transition，$bound_stepの各基本関数について紹介します．



〔図1〕クロス関数の方向

クロス（cross）関数では，指定したゼロの値を横切るときにイベントを発生する．クロス関数を用いることで，信号と同期した動作を記述することができる．

1）クロス関数

```
@(cross(式,方向,時間誤差,式の誤差))
```

クロス関数は入力信号の変化を検出して，イベントを生成します．入力式は，電圧プローブ（ブランチのポテンシャルの参照），電流のプローブ（ブランチのフローの参照），線形の時間変数式のいずれかになります．イベントは，ゼロとクロス（交差）した後，指定した時間誤差（デフォルトは1ps）の範囲で生成されます．方向は，＋1ならば負から正のクロス（立ち上がり），－1ならば正から負のクロス（立ち下がり），0ならば両方のクロスです（**図1**）．クロス関数には式の許容誤差を付け加えることができます．また，通常のイベント・ベース・モデルは時間精度のみを指定します．なお，Verilog-AMS LRM（Language Reference Manual）において，－1，0，＋1は整数と定義されているので注意してください．

2）ラスト・クロッシング関数

```
last_crossing(式,方向)
```

ラスト・クロッシング関数は，入力式とゼロが最後に交差する時刻を返します．クロス関数とは異なり，時間ステップの制御は行いません．交差した前後の時刻を直線補間して交差時刻を求めます．

3）トランジション関数

```
transition(式,遅延,立ち上がり時間,
                        立ち下がり時間,誤差)
```

トランジション関数はフィルタとして分類され，不連続の発生を防ぐために使用します．トランジション関数を用いることにより，一つの離散的な状態値から別の状態値に遷移できます．また，時間誤差（デフォルトはゼロ）を指定

できるので，シミュレーションの時間ステップを制御します．**図2**はトランジション関数による入出力波形の関係を示します．なお，離散的な状態が発生することをシミュレータに伝える$discontinuity(不連続)関数もありますが，できる限りトランジション関数を利用して不連続を発生させないほうがよいでしょう．

4）タイマ関数

```
@(timer(開始時刻,周期))
```

タイマ関数は，イベントを生成する時刻と，イベント生成を繰り返す周期を指定することができます．タイマ関数によって，クロックのような繰り返し波形を定義できます．

5）バウンド・ステップ関数

```
$bound_step(式)
```

バウンド・ステップ関数は，現在の時間ポイントから次の時間ポイントまでの，最大の時間ステップを指定します．バウンド・ステップ関数によって，時間ステップの刻みを指定することができます．

●イベント・ベース・モデルを利用してインバータを記述

イベント・ベース・モデルの記述例として，**リスト1**に簡単なインバータの記述を示します．

このモデルでは入力の変化を検出して，出力の立ち上がりまたは立ち下がりエッジを生成します．ここで，二つのパラメータは，遅延時間と立ち上がり/立ち下がりのタイミング時間を設定しています(**リスト1**の①)．入力のしきい値電圧は，電源とグラウンドの電圧をプローブしているため，電圧変動に追随して変化しています(**リスト1**の②)．

リスト1の③は，クロス関数を用いて入力状態の変化を検出していますが，つねに入力値を観測しているわけではありません．このため，このモデルが正確に初期化されることを保証するため，initial_step文を用いて，時刻がゼロのときの入力状態を観測することが重要です．

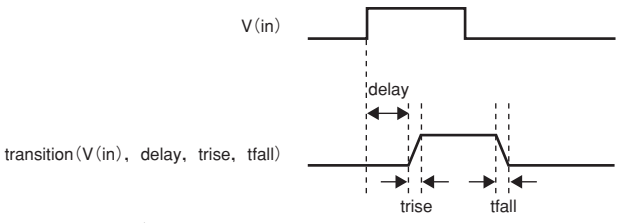
リスト1の③は@(initial_step or cross(...))を削除し，次のように変更することで，連続ベース・モデルになります．

```
outstate = V(in,vss)> threshold;
```

この場合，outstate変数は交差時刻だけでなく，つねに，すべての時間ポイントについて評価しています．ここでは，直接，交差が発生する時間ポイントを求めずに，交差が発生した後の時間ポイントの状態を変化させるので，受動的なクロス関数と言えます．この記述は，シミュレーションの実行時間を短くできるため，余分な時間ポイントを生成しないというメリットがある一方で，タイミングが不確実になるというデメリットがあります．この不確実なタイミングは，クロス関数により明示的に観測される式ではなく，ほかの回路のシミュレーション・アクティビティの時間ポイントで決まるために生じます．

クロス関数は，時間誤差(デフォルトで1ps)の範囲で交差が発生した直後の時間ポイントを生成します(**リスト1**の③)．時間誤差の値を厳しくせず，より正確に交差時刻を求めるため，ラスト・クロッシング関数を利用します．この変更を**リスト2**に示します．ラスト・クロッシング関数を用いることにより，シミュレータは交差する前後の時刻を補間して，より正確な交差時刻を求めます．

リスト2の記述では，ラスト・クロッシング関数を用いて時間精度を向上させたため，クロス関数の精度を緩めることができました．同等のタイミング精度でありながらシ



〔図2〕トランジション関数の入出力波形

トランジション(transition)関数は，整数値のような離散的な値に対して立ち上がり/立ち下がりの遷移波形を付加する．連続信号についてはスルー・フィルタ(slew)関数を使用する．

〔リスト1〕インバータの記述例

```
`include "disciplines.h"
module inv1(in, out, vdd, vss);
  input in;
  output out;
  inout vdd, vss;←①
  electrical in, out, vdd, vss;
  parameter real ttol = 1p, td = 100p, trf = 150p;
  integer outstate;
  real threshold;                                    ③
  analog begin                                       ↓
     threshold = V(vdd,vss)/2; ←②
     @(initial_step or cross(V(in,vss)-threshold,0,ttol))
                          outstate= V(in,vss)>threshold;
     V(out,vss) <+ V(vdd,vss) *
                    transition(outstate, td-trf/2, trf);
  end
 endmodule
```