第13回

論理回路のプロパティ検証技術（5）

――プロパティ記述言語（PSL）の構文と使いかた

藤田昌宏

今回は，プロパティ（仕様）を記述する言語の一つであるPSLについて解説する．PSLはAccelleraが標準案をまとめ，現在，IEEEが仕様策定を進めている検証言語である．多くのEDAベンダがPSLのサポートを表明している．PSLは，コメント文としてHDLの設計記述に埋め込むなどして使用する．時相論理を記述できるほか，信号値のシーケンスをシンプルに表現するための記法などが定義されている．（編集部）

前回説明した時相論理や正規表現など，プロパティを記述するしくみを言語としてまとめたものの一つに「PSL（Property Specification Language）」があります．PSLは設計言語などの標準化・普及推進組織であるAccelleraが標準化案をまとめています．

標準化案となっている仕様記述手法は，PSL以外にもいくつかあります．言語によるもの，ライブラリ関数的な方法によるものなど，表現手法はいろいろありますが，基本的なしくみや考えかたは同じです．

ここでは，それらの代表としてPSL言語について説明します．また，前回までに説明した時相論理ではすべての動作を設計の状態ごとに表現する形になっていましたが，状態の集まりである「期間（interval）」ごとに記述するインターバル時相論理についても紹介します．

● PSLは四つの階層を定義

仕様記述言語の標準案とされているPSLは，米国IBM社の研究所で研究・開発されたSugarと呼ばれるものをもとに，さまざまな関係者の意見を取り入れてでき上がった言語です．Sugarを研究・開発していた研究者は，もともと時相論理を使ったハードウェアの仕様記述と形式的検証手法について研究してきた人たちなので，PSLの基本も時相論理であり，前回説明した正規表現なども含めて言語の形にまとまっています．

PSLでは，以下の四つの仕様表現の階層があります．

● ブーリアン（boolean）層　● モデリング（modeling）層
● テンポラル（temporal）層　● 検証（verification）層

それぞれの階層について説明していきます．

1）ブーリアン層

もっとも基本となるのは，ブーリアン層と呼ばれる階層で，設計対象の状態を論理式の形で記述します．これは，各状態で成立すべきことを通常の論理式（時相演算子を含んでいないという意味）で表現します．一般に，通常の論理式には時間の概念がありません．例えばa＋b（信号aまたは信号bが‘1’であるという論理式）と書くと，a＋bがすべての状態（つまり，すべての時刻）で成立することになってしまいます．一方，これを時相論理式として解釈すると，「現在の時刻でa＋bが成立する」ということのみを意味します．すなわち，時相論理式として記述されたものに時相演算子が現れないと，その時刻やその状態でのみ成立する式であると解釈されるわけです．したがって，ブーリアン層では各状態で成立すべきことを状態に合わせて論理式で表現することになります．

2）テンポラル層

次がテンポラル層と呼ばれる階層です．この階層では設計対象の時間軸上における動作シーケンスが記述されます．記述には，基本的に時相論理や正規表現などが利用されます．この部分は，前回までに説明してきたプロパティの記述法がそのまま当てはめられます．時相論理や正規表現を使うことでかなり柔軟にプロパティを表現できますが，それでもいくつかのプロパティについては，表現が複雑になったり，場合によっては表現できないケースもありました．

例えば，偶数時刻でのみ信号aが‘1’となるが，奇数時刻では信号aに関する条件はない（つまり，値は‘0’でも‘1’でもよい）というプロパティは，時相論理では直接記述することができません．しかし，状態遷移図を利用すると，この性質は**図1**に示すように簡単に表現できます．

3）モデリング層

そこで，次の階層としてモデリング層と呼ばれるものが定義されています．モデリング層では必要に応じて中間変数なども導入して，状態遷移の形で動作を表現します．設計が想定しているまわりの動作環境などの表現としても利用できます．中間変数を利用して状態遷移を一般的に表現できることから，実質的に，任意の複雑度のプロパティを（言い換えると，一種の設計記述も）表現できます．また，かなり複雑なインターフェース条件も記述可能です．

4）検証層

最上位の階層は検証層と呼ばれています．この階層では，検証ツールに対して，記述されているプロパティが成立することを要求するのか，それとも記述されているプロパティが成立すると仮定して設計対象を検証することを要求するのかなどを指定します．表現されているプロパティは設計が満たすべき仕様を示しているのですが，設計はある種の動作環境における動作のみを保証すればよく，任意の入力シーケンスを考える必要がない場合が多々あります．このような場合，設計が「仮定」している動作環境をプロパティに含める必要があります．検証層では`assume`というキーワードを使って，これを表現できるようになっています．

● コメント文としてHDL記述に埋め込んで使う

PSLの記述例を見ていきましょう．実際には，PSLは既存のハードウェア記述言語であるVHDLやVerilog HDLを拡張する形で定義されています．以下は，VHDLを拡張した形のPSLについて説明していきます．なお，Verilog HDLの場合も，構文がVerilog HDL用に変わるだけなので，内容はすべて同じと考えて差し支えありません．

PSLの記述例として，まず，次の記述を考えます．

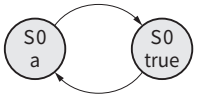

図1　時相論理で記述できないプロパティの例

偶数時刻でのみ信号aが‘1’となるが，奇数時刻では信号aに関する条件はない（つまり，値は‘0’でも‘1’でもよい）というプロパティは，時相論理では直接記述することができない．しかし，状態遷移図を使うと，この性質を簡単に表現できる．

```
-- psl property FIFO_P1 is always
((not push) or pop or (not full))
                        @rising_edge(CLK1);
```

この記述は，**図2**に示すように分解されて解釈されます．まず，`--`はVHDLではコメントなので，VHDLの通常の処理系から見るとPSL記述はすべてコメントということになります．`-- psl`と記述すると，PSLの処理系（PSL記述を理解できるシミュレータや検証ツール）は，それ以降の文をPSLの記述と認識します．

次のキーワード`property`は，以降がプロパティの記述であることを示し，次にプロパティ名（この場合は`FIFO_P1`）が続きます．そして，次のキーワード`is`以下がプロパティ`FIFO_P1`の定義となります．次の`always`で以下のプロパティは「いつも成り立つべきである」ということを表現しています．その次に，VHDLの構文に沿った形で論理式で条件が示されています．この場合，FIFOの操作と状態は，「push操作を行っていないか，pop操作を行っているか，fullではないかのいずれかの場合である」ということが表現されています．これは一見わかりにくい論理式ですが，じつはこの否定の論理式は以下のようになります．

```
（not（(not push) or pop or（not full)))
=（(not（not push)) and（not pop）and
                              （not（not full)))
=（push and（not pop）and full)
```

否定の論理式に対応するFIFOの操作は，fullの状態のときにpopせずにpushすることになり，オーバフローしてしまうことになります．したがって，このようなことは

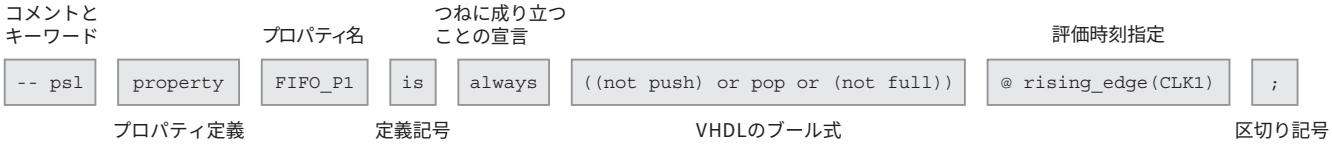

図2　PSLの記述例

ここではVHDLを拡張した形のPSLの例を示している．図のように分解されて解釈される．