SystemVerilog

デバイスの記事

記述能力，再利用性，検証機能を強化した次期Verilog言語の全ぼう

SystemVerilogの特徴と評価結果

赤星博輝

ここではSystemVerilogの特徴について解説する．SystemVerilogはVerilog 2001（IEEE 1364-2001）に続く次期Verilog HDLの言語仕様である．すでに，多くのEDAベンダがサポートを表明している．SystemVerilogでは，記述量を減らしたり，記述ミスを減らすための構文が追加されている．また，通信方式の再利用に有効なインターフェースの概念に対応している．さらに，アサーションやランダム・テスト生成など，検証のための機能も新たに用意された．（編集部）

日本ではここ数年，C言語ベースの設計が盛り上がり，SystemCがようやく認知されてきたという状況だと思います．これに対してVerilog HDLなどのハードウェア記述言語は，今後，どうなってしまうのだろうと考えている方もいらっしゃるかもしれません．2003年には，現在のVerilog HDLの後継として，SystemVerilog 3.1の言語仕様が発表されました．

このようなハードウェア記述言語を含む設計言語の進化は，設計する回路の大規模化および複雑化が一つの要因となっています．米国Intel社の創設者のひとりであるGordon E. Moore氏が「LSIに集積されるトランジスタ数は，2年で倍のペースで増える」と述べたムーアの法則で示されるように，LSIで実現できる回路規模は指数関数的に増加してきました．回路規模が大きくなることにより，その大規模な回路を効率良く設計する手段や設計検証項目が増大することへの対応が必要となります．また，事前の性能評価などの検討がより重要となってきました．

Cベース設計では，C言語やC++などのデータ型を使ってシミュレーションすることで，検証速度を高速化しています．また，ビヘイビア合成を使うことにより，RTL（register transfer level）設計より記述量を削減できるなどの利点もあります．さらに，最近のLSIでは機能をソフトウェアとして実現する比率が高く，ソフトウェア開発とリンクしやすいということもありますし，ハードウェアとソフトウェアを合わせたシステムとして包括的に設計を行える可能性もあります．

● 記述量を削減し，モデリングと検証のための機能を強化

それではなぜ，わざわざSystemVerilogが開発されたのでしょうか？ これは，ハードウェアの設計ではVerilog HDLが主要な言語の一つであるからです．これまでの設計資産や設計手法を捨てて，新しい設計言語や新しい設計手法に移行することはそれほど簡単なことではありません．ただし，既存のVerilog HDLにはいくつかの問題点がありますし，最近登場した言語と比べると機能的に遅れている点もあります．そのため，これまでのVerilog HDLの設計資産や設計手法をそのまま使用でき，さらに新しい機能が使えるようになったSystemVerilogが注目されています．

SystemVerilogの新しい機能としては，以下の点を挙げることができます．

- RTL記述をより効率的に行える
- 高位のモデリングを容易に行える
- 検証向けの機能を言語仕様として用意している

設計を効率良く行う（端的に言えば，記述量を減らす）ための方法としては，高位のビヘイビア・レベル記述を使う手法があります（ビヘイビア・レベル記述による設計はあまり普及していないが…）．これに対して，SystemVerilogではVerilog HDL言語を改良することで，RTLでも記述量を減らせるようになりました．これまでの設計手法を使用したまま，記述量を減らせることは設計者にとって利点となります．

また，これまでは高位のモデリングはC言語やC++が中心的な役割を担っていましたが，SystemVerilogではC言語とのリンクも含め，高位のモデリングが行いやすくなりました．

検証についてはVerilog HDLのサポートが弱く，最近，検証のための専用言語（いわゆる検証言語）などを使用する事例が増えています．SystemVerilogでは検証に対する機能を大幅に追加し，別の言語を用いる必要がなくなりました．

ここでは，Verilog HDLの問題点や機能的に遅れている点を中心に，SystemVerilogで大きく変わった点について解説していきます．なお，本稿で説明するうえで「Verilog HDL」と書いた場合は，現在，一般的に使用されているIEEE 1364-1995を指すこととします．一方，「Verilog 2001」と書いた場合には，現在のEDAツールでも一部サポートされているIEEE 1364-2001を指すこととし，「SystemVerilog」はSystemVerilog 3.1を指すこととします．

● always文を大幅に改良

まずはalways文の問題から見ていきます．always文には設計時にミスが発生しやすいという問題がありました．

1）always_comb文の追加

always文を使って組み合わせ回路を記述する場合，センシティビティ・リストに必要な信号をもれなく記述しなければなりません．例えば，**図1**の例では2入力ANDを作成しようとしたのですが，センシティビティ・リストが不十分であるため，シミュレーションは誤った結果になっています．しかし，よく考えてみるとセンシティビティ・リストを書くという作業は，設計者のためというよりも，昔，このように書くと決められたからというものです．

この問題を解決するため，SystemVerilogではalways_combという文が追加されています．これにより，設計者はセンシティビティ・リストを書く必要がなくなります．これは，設計者にとってはうれしいことです．複雑な回路を記述していると，どうしてもセンシティビティ・リストに書く信号が多くなるため，ミスしやすくなります．always_comb文を使えばセンシティビティ・リストに関するミスがなくなり，センシティビティ・リストが不要になるため，記述量も減ることになります．

2）always_latch文の追加

組み合わせ回路を記述するためにalways文の中でif文やcase文を使うときに，すべての条件が記述されていないとラッチを生成するという問題があります．もちろん，ラッチを記述したい場合はそれで良いのですが，条件のもれによって発生してしまったというケースが多いようです．

そこで，SystemVerilogではラッチを明示的に記述するためにalways_latch文が追加されました（**リスト1**）．

リスト1　always_latchを使った記述

これまでの記述と異なり，センシティビティ・リストは不要になる．回路構造を明示的に指定できることで，デバッグ・ツールなどの支援を期待できる．

```
module case1(a,sel,c);
  input a;
  input[1:0]sel;
  output c;
  reg c;

  always_latch

  case(sel)
   2'b00:  c = a;
   2'b01:  c = 1;
   2'b10:  c = 0;
  endcase

endmodule
```

これまでの記述では always@(a or sel)

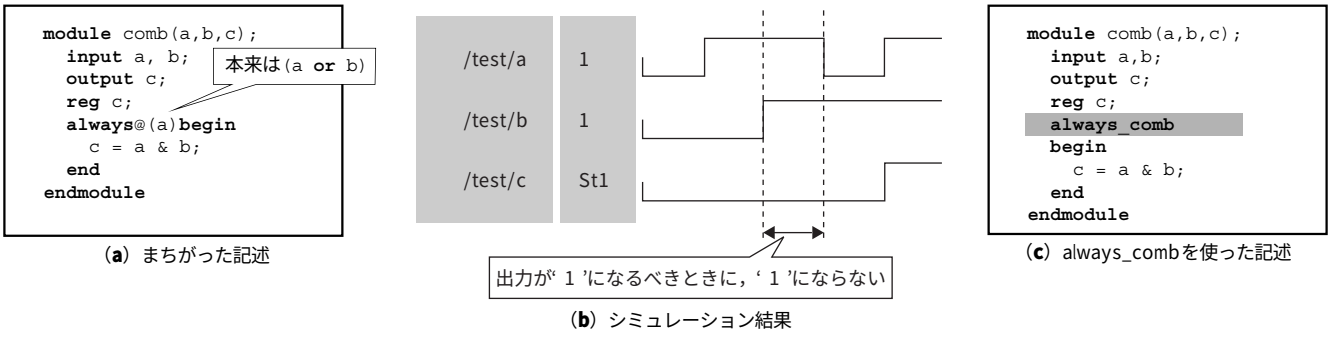

図1　まちがったセンシティビティ・リスト

(a)のようなまちがったセンシティビティ・リストでシミュレーションを行うと，(b)のように入力aとbが‘1’のときに出力cが‘1’にならず，‘0’になったりする．こういうミスは急いでいるときほど発生しやすい．SystemVerilogの場合，組み合わせ回路の記述にalways_combを使えば，(c)のようにセンシティビティ・リストは不要である．