システムLSI 検証技術入門

デバイスの記事

第1回

フォーマル・ベリフィケーションの意味

シミュレーションやエミュレーションとの違いを再確認しよう

藤田昌宏

本連載では，システムLSIの検証技術を，特にフォーマル・ベリフィケーションに焦点を絞って解説していく．フォーマル・ベリフィケーションとは，すべての場合について網羅的にシミュレーションすることと同等のことを，数学の定理証明のように「証明」してしまう方法である．連載第1回の今回は，フォーマル・ベリフィケーションの考えかた，およびシミュレーションやエミュレーションとフォーマル・ベリフィケーションの違いなどについて説明する．（編集部）

LSIの集積度が上がるにつれて，ますます大規模な回路を短期間に設計しなければならなくなっています．最近では，半導体製造技術の微細化に伴う電子回路の諸問題（クロストーク・ノイズや配線遅延の見積りの難しさなど）が指摘されており，LSI 設計期間が増大する大きな要因となっています．

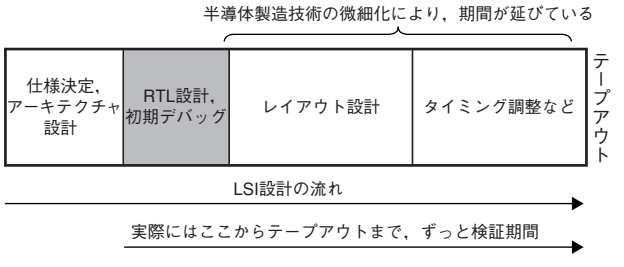
その一方で，大規模で複雑な「論理回路」をいかに効率良く設計するかということも大きな課題です．特に，論理回路をどうすれば「正しく」設計することができるのかは，最重要課題の一つです．LSI設計では，設計期間全体に対する検証期間の割合が50%，あるいは80%以上になっているとも報告されています．これはある意味，当然のことであると言えます．ソフトウェア・プログラム開発において，大規模プログラムの場合，プログラムのコーディング期間よりもデバッグ期間のほうがずっと長いのは常識です．

例えば，**図1**に示すように，仕様記述をもとにアーキテクチャ設計が行われた後，RTL（register transfer level；レジスタ転送レベル）設計が行われ，論理合成，レイアウト設計へと進んでいきます．設計がより大規模・複雑になってくると，RTL設計以降ずっと検証作業を行うことになるのが普通です．このため，検証期間が設計期間全体の80%以上を占めるというケースも発生します．

現在，設計作業はVerilog HDLやVHDL などのHDL（hardware description language；ハードウェア記述言語）を利用して，主にRTLから開始されています．そして，HDLによるコーディングの過程では，ソフトウェア・プログラミングと似たような作業が行われています．

ここで重要なことは，通常のソフトウェアではOSなどの例外を除き，プログラムはすべて「シーケンシャル動作」であり，個々の処理（タスク）は順番に一つずつ実行されるということです．一方，ハードウェアは本質的に「並列動作」が基本です．LSI内部の個々の論理ゲートはすべて，並列に動作しています．HDLで表現された場合も，複数の回路が同時に動いています．つまり，HDL設計とは「並列・並行処理プログラミング」であるとも言えるのです．

したがって，非常に小規模なLSIの設計であっても，回路の動作を考えるとき，同時に考慮しなければならない要素の数はかなり多くなります．そのため，LSI検証問題は，ソフトウェア検証問題と比較して，とても難しい問題であ



〔図1〕LSI設計工程と検証期間

LSI設計では，仕様記述をもとにアーキテクチャ設計が行われた後，RTL設計が行われ，論理合成，レイアウト設計へと進んでいく．従来，レイアウト設計におけるタイミング調整などの期間が長かった．設計がより大規模・複雑になってくると，RTL設計以降，ずっと検証を行うことになる．このため，検証期間がLSI設計の全期間の80％を占めるという報告も出てきている．

ると言えます．この事実が，LSI設計過程において，検証期間の割合が支配的になりつつある理由の一つです．

このような状況を踏まえ，本連載では論理設計における検証技術，特にフォーマル・ベリフィケーション（形式的検証）技術に重点をおいて，以下の問題について解説していきます．

- **論理検証の現状の概観**
- **シミュレーション，エミュレーション，フォーマル・ベリフィケーション技術を効率良く組み合せて利用する方法**
- **各種フォーマル・ベリフィケーション技術の基本と，その利用面からの比較**

連載第1回目の今回は，導入部として，現在の主流であるシミュレーションやエミュレーションによる論理検証技術と，今後の論理検証の大きな道具の一つとなる「フォーマル・ベリフィケーション技術」の関係について，説明していきたいと思います．

1 シミュレーションとエミュレーション

従来，LSIの論理検証では，通常，「できるだけたくさんシミュレーションを行って，設計の正しさを確認する」というアプローチがとられてきていました．これは，許される時間内で，思い付く限りの項目についてシミュレーションするパターンを考え，適用するということです．

しかし，このアプローチには以下のような問題があることが知られています．

- **人の考察には，いつも，考慮し忘れや思い違いなどがある．調べるべき項目について，すべての場合を正しく調べていないことが普通である．**
- **より広くチェックするため，乱数によって生成されたシミュレーション・パターンが使われることもある．しかし，実際にシミュレーション可能なパターンは，すべてのパターンから考えるとごく小数であり，いくつかの場合を試行してみたという程度にすぎない．**
- **「こうなるべきだ」という，正しく動作する場合についての考察はおおむね網羅的にできる．しかし，「こうなってはいけない」という，禁止されている場合については，すべてを列挙することは，通常不可能に近い．**

●検証に実アプリを利用できるエミュレーション

FPGA（field programmable gate array）など，再プログラム可能なLSIを使用して，設計を実際にハードウェアとして実現し，シミュレーションする技術も利用されています．これは通常，エミュレーション（論理エミュレーション）と呼ばれています．

エミュレーションの速度は，シミュレーションの数百倍または数千倍になることが多く，シミュレーションと比べてはるかに多くのケース（多くのシミュレーション・パターン）を調べることができます．また，人手で作成されたシミュレーション・パターンだけでなく，エミュレーションの高速性を利用して，設計対象のハードウェア上で実際のアプリケーション・プログラムを走らせて，動作をチェックすることも可能です．

実際，米国Intel社のPentiumプロセッサなどの設計では，リビング・ルームくらいの部屋に論理エミュレータを何台も配置して，WindowsやMicrosoft Officeなどのプログラムを走らせ，動作をチェックすることが常時行われているそうです．

このようにエミュレーションは，貴重，かつ有効な技術ですが，バグを完全に取り去るという立場からみると，利用されているシミュレーション・パターンに見落しがある場合が多いのも現実です．

●シミュレーションによる網羅率は事実上ゼロ

シミュレーションやエミュレーションでは，基本的に可能な限り多くの場合を個々に一つずつ調べていく技術であるため，個々の検証すべき事項について，十分に調べたかどうかをチェックする機能が重要となります．つまり，検証カバレッジ（網羅率）が重要となるのですが，実はその計算は簡単ではありません．

例えば，もっとも単純な検証カバレッジとして，「すべての可能な入力パターンのうち，どれだけの割合のパターンをチェックしたか」という数値を考えてみます．これは，簡単でかつわかりやすいカバレッジです．この数値が高ければ，十分に検証できたと言えます．

しかし，どんなにがんばってシミュレーション（またはエミュレーション）しても，カバレッジはいつも限りなく0に近くなります．これは次のような議論からわかります．今，検証対象にN個の入力があり，中の回路にM個のフリップフロップがあるとします．フリップフロップがM個ありますから，内部状態は最大2^M個ある可能性があります．実際には，すべての状態が実現可能（初期状態から到達可