システムLSI 検証技術入門

デバイスの記事

第14回

論理回路のプロパティ検証技術（6）

――アサーション・ベース検証の基礎

藤田昌宏

ここでは，アサーション・ベース検証について解説する．アサーション・ベース検証は，検証対象となる設計の内部に観測点を設け，この点に注目して設計バグを検出する手法である．バグの信号値を外部端子に導いて確認する従来手法では検出しにくいケースでも，比較的容易にバグを検出できるという．学会などでは，この手法を用いることにより，検証期間が約1/2になったと報告されている．（編集部）

今回は，アサーション記述言語を実際の設計に利用する際の考えかたについて説明したいと思います．これは，一般に「アサーション・ベース検証」と呼ばれています．シミュレーション，形式的検証のいずれにも利用でき，実際の設計にも取り入れられ始めています．

● チップ・レベル検証への依存度が高い

図1に，現在，広く利用されているLSI設計とその検証の流れを示します．仕様は，設計対象のチップ全体として与えられ，設計作業ではその仕様を詳細化していきます．その際に，チップ全体を徐々に分割していき，分割されたものに対する仕様を定義していくことになります．チップ・レベルでは，メモリ，CPU，IP（intellectual property）コア，そして外部とのインターフェースという単位で分割されます．この過程では，設計対象のシステムLSIに対するハードウェア部分とソフトウェア部分の分割なども実施されます．

CPUについては，既存のIPコアがすでに用意されており，新たに設計する必要はありません．設計対象となるLSIの中のカスタム・ハードウェアの部分が詳細化・分割の対

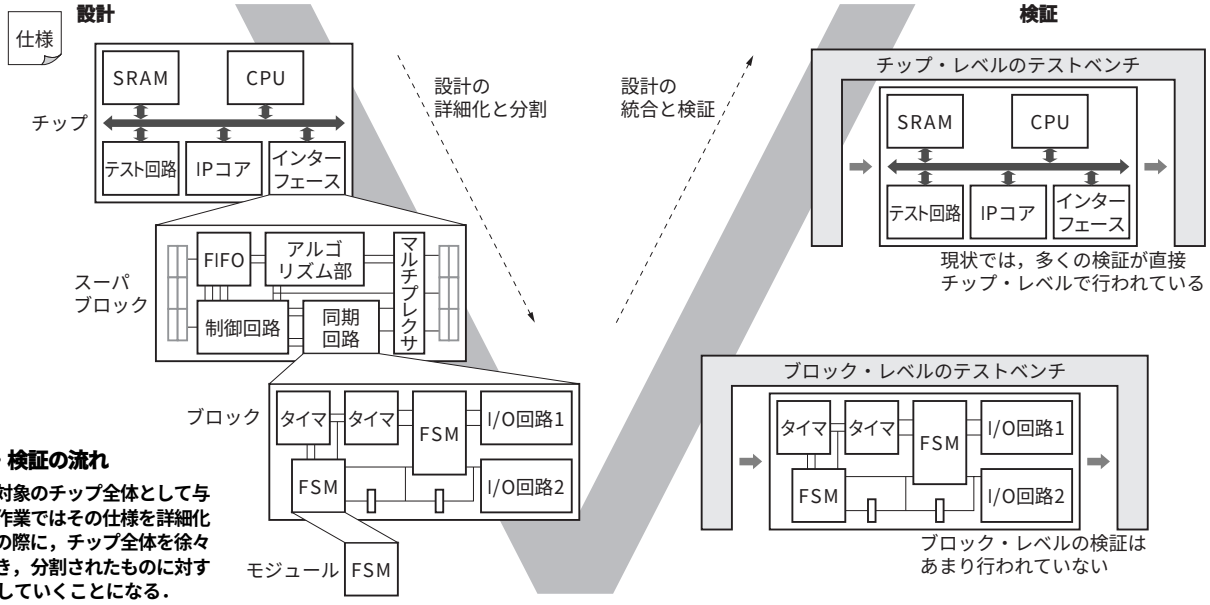

図1
現状の設計・検証の流れ
仕様は，設計対象のチップ全体として与えられ，設計作業ではその仕様を詳細化していく．その際に，チップ全体を徐々に分割していき，分割されたものに対する仕様を定義していくことになる．

象となります．**図1**の例ではインターフェースがカスタム・ハードウェアであり，詳細化していきます．この部分は，FIFO（first-in first-out）などのバッファ・メモリ，なんらかの処理を行うアルゴリズム部，また，外部との通信の同期をとる部分，そして，外部ポートのマルチプレクサや制御回路などに分割できます．この比較的大きなブロックのことを，ここではかりにスーパブロックと呼ぶことにします．

さて，このスーパブロックはさらに詳細化され，ブロックの集まりに分解されます．**図1**の例では，通信の同期をとる部分について詳細化しています．これは，制御を実現する状態遷移（FSM：finite state machine），I/O回路，そしてタイマなどから構成されます．さらに設計を詳細化し，いわゆるモジュール単位に分割してRTL（register transfer level）記述を作成することになります．

以上のように設計の各部分がモジュール・レベルまで詳細化されると，次にそれらを結合しながら，検証することになります．ほんとうは設計の詳細化の各段階ごとに検証を行っていくことが望ましいのですが，設計期間に制限があったり，各モジュール，ブロック，スーパブロックに対する仕様がかならずしも明確になっていない場合も多く，全部をつないで行うチップ・レベル検証への依存度が高くなっているようです．

● ブロック・レベルでバグを検出することが理想だが…

本連載でも説明してきたように，形式的検証が利用できる局面は，現在のところかなり限られます．チップ全体を扱える形式的検証ツールは，現状では組み合わせ回路の等価性検証用だけであり，LSIの設計データを製造部門に引き渡す際に，その設計データともとのRTL記述の間の等価性をチェックする程度です．

しかし見かたを変えると，設計規模さえある水準以下であれば，形式的検証ツールが利用可能であるとも言えます．**図1**の検証の流れでは，モジュールやブロックといった小規模回路については十分に利用可能なはずです．アサーション・ベース検証とは，このモジュールやブロックの検証に対して，とくに形式的検証（実際には，アサーション・ベース検証をシミュレーションによって実施することも多い）をうまく利用できるようにする技術であると言えます．

検証をチップ・レベルで行う場合，まず問題となるのが，チップ全体の設計データがそろわなければ作業が始められないという点です．つまり，設計作業がある程度進んでからでないと，検証を始められないことになってしまいます．

また，設計規模の点から形式的検証手法が適用できず，シミュレーションを行うことになるのですが，一般に，チップ全体に対するシミュレーションだけでは検証カバレッジ（網羅率）を高くすることが困難です．検証カバレッジでは，例えばRTL記述にある条件文に対して，条件が成立する場合と成立しない場合の両方をチェックしないとカバレッジが上がりませんが，チップの外部端子からのみ信号パターンを供給していると，なかなか内部の状態を思うように制御できません．結果として，きわめて低いカバレッジしか得られないか，あるいは膨大な入力パターンが必要となってしまいます．また，後で例を示しますが，設計バグのため，内部の動作におかしな点があっても，その影響が外部端子に伝わるような入力パターンになっていないと，バグを外部から観測できません．

さらに，テストベンチの再利用も難しくなってしまいます．チップ全体に対するテストベンチであるため，まったく同じ機能のチップでないとそれを再利用できません．現在，システムLSIの全開発期間に占める検証期間の割合が増大しており，テストベンチなど，検証データの再利用が課題となっていますが，それもたいへん難しいわけです．

これに対して，ブロック・レベルで検証すれば，設計作業のより早い段階で検証を始められます．また，検証対象規模がそれほど大きくない場合も多く，形式的検証ツールも利用しやすくなります．ここでブロック・レベル検証とは，チップ全体ではなく，その中の各ブロックを切り出してきて，それを検証対象とすることを指します．

しかし，ブロック・レベル検証がそれほど簡単に実施できないのも事実です．まず，各ブロック単位でテストベンチを用意する必要があるわけですが，それはかならずしも容易ではありません．各設計者は，自分が担当するブロックの動作はよく理解していますが，もし，検証を行う人が別の人だった場合，各ブロックの動作や仕様がかならずしも明確ではなく，テストベンチの作成にかなりの時間がかかってしまいます．この理由として，各ブロック間のインターフェースがかならずしも明確に定義されていない，ということが挙げられます．

これはよくないことなのですが，設計しながら各ブロック間のインターフェース仕様を決めていったり，各設計者が自分で適当に決めているといった場合も少なくありません．結果として，ブロック・レベルでもテストベンチの再