F5 RTL設計における論理検証手法

古川　寛

初心者にとって，ASICの開発フローは複雑でわかりにくい．なかでも「検証」についての意味と位置づけが理解しにくい．また，検証の工程が，大規模化していくASICの開発において大きなボトルネックとなっているという．そして，その解決に向けて新しい手法が導入されつつある．そこで，ここでは検証にスポットをあてて，その種類と原理，処理手順を中心に解説する．（編集部）

1 検証の意味

■次の工程に進むために必要な検証

図1に示す開発フローを見ると，作り込みの箇所と検証の箇所の二つからなることがわかります．そのうち「検証」とは，作り込んだと思ってるモノがきちんと予定どおりに作り込まれているかどうかの確認作業のことです．

作り込んだ後，検証してバグが0であることを確認できて初めて，次の工程へデータを受け渡すことができるのです．もし，バグが0だったら，検証はたんなる時間の無駄なのでしょうか．このような意見を，ゲート・レベル検証に対して言う人がいます．そうではありません．たまたまバグが0だったから，そういいたくなるだけなのです．

■検証が開発のボトルネックに

この検証という作業は，最近どんどん進化しています．チップの大規模化にともない，従来の手法ではＴＡＴ（turn arround time）やメモリ使用量といった点で限界があり，新しい手法にシフトせざるを得ないのです．

現在はその過渡期なのですが，本稿では，その現状をまとめます．理論先行で開発が進められているものもありますが，ここで紹介するのは，いずれも現実的に使用可能なものです（現在，デザイン・プランニング・ツールというモノが登場し，さらに進化しようとしている）．

なお，以下ではあくまでも代表的な例であったり，わかりやすいように細かい部分を省いているところがあることを，お断りしておきます．

2 開発フローと検証

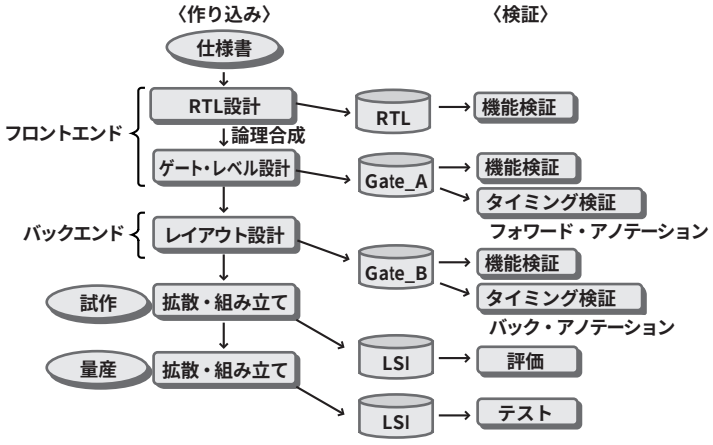
■ASICの開発フロー

最近のRTL設計における開発フローを**図1**に示します．RTL設計やゲート・レベル設計の工程を**フロントエンド**，レイアウト設計以降の工程を**バックエンド**と呼んで区別します．それぞれ開発担当者は別に存在します．

フロントエンドでは，開発するシステムの仕様を詳細に理解し，回路に落とし込む必要があります．ニーズを十二分に把握し，いかに魅力のある仕様を実現するかが重要です．

バックエンドでは，ターゲット・サイズ内でタイミング要求を満足させ，シリコンに落とし込む必要があります．回路設計とは異なるバックエンド特有のスキルが重要です．

バックエンドは，フロントエンドからRTL（register transfer level）ソースが出てこないと何も作業を始められません．最終的には，バグがまったく残っていないRTLソースで行うのですが，バグがなくなってからバックエンドの作業を始めても競争の激しい市場では生き残れません．バグがあってもできるだけ規模やタイミングの仕様が同一のRTLを早い時期に用意する必要があります．バックエンドは，バグの含まれるRTLでタイミングやサイズを満足させるようトライアルを繰り返し，バグのないRTLができたら流



〔図1〕ASICの開発フロー

すだけという状況を用意しておきます．大事なのはtime to market[注1]です．ちなみに，フロントエンドからバックエンドへの最終的なRTLの提供が遅れることは多々あります．ASICユーザへのサンプル・チップの提出日程は遅れるわけにはいかないので，このような場合，バックエンド側が被害をこうむり，徹夜の突貫作業でレイアウト・データを作成することになります．

かくして，フロントエンドとバックエンドの仲が悪くなる場合が発生するのです!?

■検証の種類

論理回路の検証は，**機能検証**と**タイミング検証**(性能や信号遅延の検証)の二つに分類されます．また，最近では，二つの設計データの機能やタイミングが等価であることを判定する**論理等価性検証**に注目が集まっています．

それでは，これらについて詳しく見ていくことにしましょう[注2]．

3 機能検証

■機能検証の意味

RTL設計では機能を作り込みますが，具体的にはRTLのHDLソースを記述することになります．また，最近はESDA(electronic system design auto-mation：電子システム設計自動化)ツールで真理値表やフローチャートを書くと，RTLソースを吐き出してくれる便利なものもあります．

さて，ここまでは頭の中で考えた回路がRTLで実現されただけです．もちろん，きちんと動くと思って作成はしているでしょうが，本当に予定したとおりに動くかどうかは，頭のでき(?)にかかっています．しかし，回路が複雑になってくると，いくら頭のよい人でも一発で完璧に動作させることは難しくなります．そこで出てくるのが，機能検証です．

■機能検証のためのツール ―― 論理シミュレータ

機能検証には，**論理シミュレータ**を用いるのが一般的です(Cadence社のVerilog-XL，Synopsys社のVCSなど)．シミュレータは，作成した設計データに対して，作り込んだ機能が動作するような信号パターンを入力し，出力ピンや回路内部のネットの信号値をダンプすることによって，回路の動作を確認します．

RTLでのシミュレーション自体はそれほど遅くないのですが[注3]，100MHz動作のチップの実動作の1秒を再現するためには1億サイクルのクロックを入力させる必要があるということを考えると，それなりの検証時間が必要であることも理解できるかと思います[注4]．

また，人間がパターンを考えたり，動作を確認するには多大な時間と労力を要します．フロントエンド工程が，製品開発フロー全体でのボトルネックになることもしばしばあります．チップの大規模化に伴い，回路の複雑さが増し，機能検証の工数は指数関数的に増大しているというのが実状です．

■機能検証の難しさ ―― 主体はツールではなく人間

ところで，機能検証には難しい問題があります．それは，回路を作り込むのも人間だし，回路を検証するのも人間だということです．検証はシミュレーションで行いますが，シミュレーションの入力パターンを作成し，出力結果を見て判定するのは人間なのです．設計者は，作り込んだ回路に対して，ありうるすべての状況(組み合わせ)を考え，すべての場合のシミュレーションを行う必要があります．ここで，想定した条件に漏れがあったり，みるべき出力部分を見落としたりするとバグの発生に直結します．

これについては永遠の課題で，どれだけEDA(electronic design automation：電子設計自動化)が進化しても，最後まで残る部分でしょう．やはり，機能を設計し検証するといった**考える**部分は人間のなせる業なのです(逆にこの部分のスキルをもっていると，どれだけ設計が自動化されても職に追われることはないかもしれない)．ちなみに筆者は，この部分が一番おもしろく，設計者として生きがいを感じるところです．単純に作った回路が動くのを見るのは，おもしろいものです．

■検証の度合いを測定する カバレッジ・ツール

判定するのは人間ですが，検証の度合いを測定するツールがあります．これを**カバレッジ・ツール**(coverage tool)といいます．

一昔前までは，ゲート・レベルでトグルを見るといった手法が一般的でした．これは内部ノードが，0→1，1→0に変化した具合いをカバレッジ(網羅率)として表す手法です．たんにトグル率を測定しているだけなので，正確に検証具合いを見ているとはいえませんが，トグルすらしてないような部分はまったく検証されていないということがわかります．この手法の欠点は，ゲート・レベルということで，シミュレーションの負荷が重いということでした．

注1：製品を世に出すまでの時間．ニーズにあわせて，早期に製品を市場投入することが重要であることを意味する．

注2：また**レイアウト検証**というものもある．こちらは，LVS(layout vs schematic)と呼ばれ，レイアウト・ツールへの入力データである論理回路図(schematic)と出力データである物理レイアウト(layout)が等価であるかを検証する．この場合，トランジスタ・レベルで比較が行われる．

注3：シミュレータも，従来のインタプリタ型からコンパイラ型，そしてサイクル・ベース型へとどんどん高速になってきている．

注4：このように実時間単位で検証を行いたい場合，**ハードウェア・エミュレータ**というものがよく用いられる．これは，ネットリストを入力すると，その回路データに相当するハードウェアを実現するという装置である(大規模で高速のFPGAが大量に組み込まれている)．画像系などの実時間検証には有用だが，検証するには入力も出力も実チップと同じように実際の信号を入力してやる必要があるため，設計の終盤で用いられる．また非常に高価なのがネックである(数千万円から１億円を超える)．