重点企画 デバッグ鉄人への道2

第1章 初心者のためのテストベンチ記述テクニック

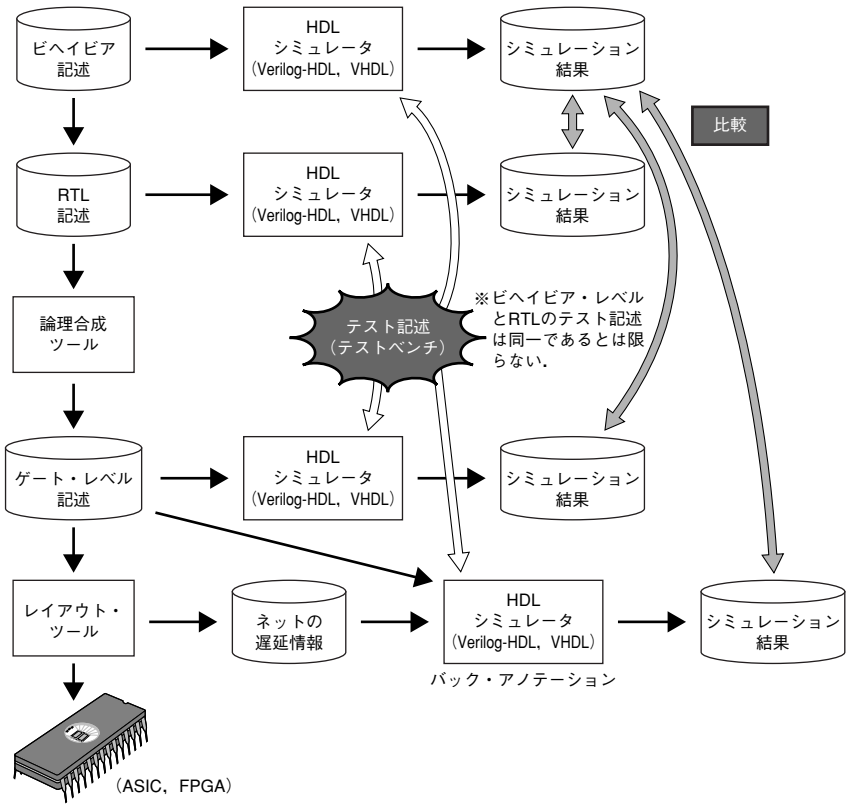
鳥海佳孝，田原迫仁治，横溝憲治

最近の大規模LSIの開発では，設計そのものに費やす時間よりも検証に費やす時間のほうが長くなっている．テストベンチのよしあしが設計品質や設計期間に影響を与え，ひいてはプロジェクトの成否を左右するようになってきた．ここでは機能検証とテストベンチの基本的な考え方を解説する．また，サンプル記述を示しながら，テストベンチ作成のテクニックを紹介する．なお，本記事で紹介したサンプル記述は，本誌のホームページ（http://www.cqpub.co.jp/dwm/）からダウンロードできる．また，次号に付属するCD-ROMにも収録する．（編集部）

前回は，HDLのサンプル記述集の記事（本誌2000年1月号の特集）でみなさんにお目にかかりましたが，今回はテストベンチについてお話ししたいと思います．1回ですべてをお話しするのはむずかしいので，3回程度の短期集中連載を予定しています．

1．テストベンチ作成の基礎知識

まず，「テストベンチ」とはいったいなんなのかというところからお話しします．ちょっと話が脱線しますが，エンジニアが好きなものを好きなように設計するだけで，「あとは，し〜らないっ！」ということであれば，エンジニアという職業はきっといま以上に楽なものになるに違いありません．しかし実際には，作った（設計した）ものが本当に正しいかどうか，ありとあらゆる手段を使ってたしかめなければなりません．このたしかめるための手段が論理シミュレーションにおける，いわゆる「テストベンチ」です．



〔図1〕HDLを用いたトップダウン設計のフロー

一般的なトップダウン設計フローを示した．必ずしもビヘイビア・レベルで使用するテストベンチと，RTLなどで使用するテストベンチが同一のものになるとは限らない．また，設計の抽象度が下がっていくにしたがってテスト項目が増えていくので，シミュレーション結果も同一のものになるとは限らない．

●不具合の原因は3種類

図1を見てください．この図は一般的なトップダウン設計のフローを表しています(FPGAでもASICでも同じ)．トップダウン設計では，各シミュレーション・フェーズ(ビヘイビア・レベル，RTL，ゲート・レベル)で，それぞれのシミュレーション結果を比較します．シミュレーション結果を得るために，それぞれのモデルに入力パターン(入力信号列)を与えます．入力パターンを与えて，モデルやテスト対象となる回路の出力を観察するのが，テストベンチの役割であり目的です．

さて，もしここで，いい加減な入力パターンを与えたり，実際に起こり得ない入力パターンを与えて出力を観察していたら，どうなるかを考えてみましょう．とりあえず，得られた出力(シミュレーション)結果は正しいものになるに違いありません．しかし，その結果だけで本当に良しとするべきかどうかは，はなはだ疑問です．なぜなら，実機で動かしたときに入力されるであろう事象が，シミュレーション上で検証されていないからです．

筆者がLSI設計の世界に飛び込んだころは，プロセス技術に起因する不良(たとえば，製造ばらつきによる不良など)が結構ありました．現在ではプロセス技術が進歩したこともあって，そのようなプロセス的な不良が起こる確率はかなり減ってきたと思われます．したがって，実際にLSIを作ってうまく動作しなかった場合の原因は，以下の三つになります．

⑴ **仕様上のミス**
⑵ **設計上のミス**
⑶ **配線遅延などによる遅延の増加に起因する不良**

前述したプロセス的な不良の場合，不具合が起こるかどうかを，実際にチップを作る前にシミュレーションなどによって確認できません．しかし，上記の⑴～⑶の不良は，テスト・パターンさえしっかり作れば，すべてシミュレーションによって確認できます．

多少言い過ぎかもしれませんが，実機でバグが出るというのは，たとえ⑴の仕様上のミスが原因であったとしても，ほとんどの場合，「テスト・パターンに漏れがあったからバグが見つからなかった」と言ってよいのではないかと筆者は考えています．ですから，これから設計するLSIが使われようとしているアプリケーションに対して，いかに完璧なテスト・パターンを作れるかが勝負となります(ただし，完璧なテスト・パターンを作るのはきわめてむずかしい)．

話が回りくどくなりました．**図1**に示したように，ビヘイビア・レベルで出力されるシミュレーション結果が，いわゆる期待値になります．いくら論理合成ツールやレイアウト・ツールを駆使して良い回路を作ったとしても，十分なテスト・パターンを通していないような回路は役に立ちません(ただし，その逆も言えて，いくらきちんとシミュレーションしてあっても，目標としたサイズに収まらなかったり，目標としたスピードが出ないのでは役に立たない)．

●完璧なテスト・パターンを作ったのは第三者のソフト技術者

では，どうしたら完璧なテスト・パターンを作れるのでしょうか？「こうすれば必ずできる」と言い切ることはできませんが，筆者の経験から言うと，自分でテスト・パターンを作らないことです．できれば第三者に作ってもらうのが，完璧なテスト・パターンを手に入れる早道だと思います．

以前，筆者は上記のことを裏付けるできごとに遭遇しました．筆者がプロセッサを設計していたとき，そのテスト・パターンを筆者が作るのではなく，まったくの第三者に作ってもらいました．しかもハードウェア・エンジニアではなく，ソフトウェア・エンジニアの人に担当してもらいました．その“彼”はプロセッサの仕様書を受け取り，テスト・パターンを作成しました．おそらく“彼”からは，そのプロセッサの中身(ハードウェア)がどうなっているのか，ほとんどわからなかったのではないでしょうか．プロセッサの入出力ピンと，なかで抱えているレジスタだけしか見えていなかったと思います．動作などを理解したうえで，いろいろな命令の組み合わせや，ある演算命令が実行された後，この演算だったらこのフラグはこうなっているはず，といったようなことをしらみつぶしにチェックしてもらいました．また，できるだけすべての命令を1度は実行するように，テスト・パターンを作ってもらいました．

ある程度RTL記述ができあがったところで，そのテスト・パターンを使ってシミュレーションすることになりました．演算ユニットなどのシミュレーションは，事前にある程度すませていたのですが，いざ彼が作ってくれたテスト・パターンを使ってシミュレーションしてみると，論理シミュレータ上とはいえバグだらけであることが判明しました(作り込む前にいくらバグを出しても，取り返しはいくらでもきく)．一つ一つの命令をていねいに見ているテスト・パターンですから，その量はそれなりに膨大でした．いろいろと試行錯誤のすえ，時間をかけてシミュレーションした結果，なんとかすべてのテスト・パターンをとおすことに成功しました．その後，論理合成ツールにかけてゲート・レベル・シミュレーションを行い，チップを作りました．

実際にチップができあがってきて評価してみると，なんと一発で動いてしまいました(いまとなっては当たり前のことかもしれないが…)．1箇所だけ非同期で入ってくるNMI(ノン