第7章

HDLのデバッグを体験する

■Novas Software社のHDLデバッガ「Debussy」■

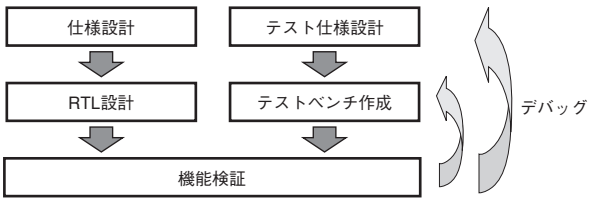
山本美由紀

ここでは，HDL設計の機能検証やタイミング検証の際に行うデバッグの作業を体験する．デバッグ作業そのものはテキスト・エディタなどがあれば進められる．しかし，回路が複雑になり，過去の設計資産の流用が増えてきた現在では，この方法は効率が良いとは言えない．そこでHDL設計専用のデバッガが登場している．本チュートリアルでは米国Novas Software社のHDLデバッガ「Debussy」と，ブラックジャック・ゲームのサンプル記述を用いる．ツールの評価版とサンプル記述は，本誌付属のCD-ROMに収録されている．（編集部）

LSI設計では，検証を抜きに作業を進めることはできません．検証はLSI設計のあらゆる工程で実施されます．例えば，RTL（register transfer level）記述ができ上がった時点では，HDLシミュレータなどを使って機能検証を行います．また，リント・ツールを使ってシンタックス（構文）やセマンティクス（語義）のチェックも行います．論理合成や配置配線が終了すれば，静的タイミング解析ツールを使ってタイミング検証を行います．

検証時にエラーが発生した場合，なぜそのエラーが生じたのかを突き止め，処置しなければなりません．これが，いわゆる「デバッグ」です．つまり検証とデバッグは対になっているわけです．検証がLSI設計のあらゆるフェーズで実施されているように，デバッグもまたあらゆるフェーズで必要となります．

ところで，「バグ」はなぜ発生するのでしょうか？ ひと言にバグと言いますが，発生原因はさまざまです．以下に，HDL設計でよく見つかるバグの例を挙げてみました．



〔図1〕機能検証フローとデバッグの位置付け

機能検証によって不ぐあいが発見された場合，仕様やRTL記述，テストベンチを見直して修正する．この作業が「デバッグ」と呼ばれるものである．デバッグは機能検証が終わるまで繰り返し行われる．

- **仕様書そのものが誤っていた**
- **仕様書に対する理解が誤っていた**
- **単なるコーディング時のミス**
- **仕様書の内容や表現にあいまいな部分があり，コーディング時にそのことに気が付かなかった**

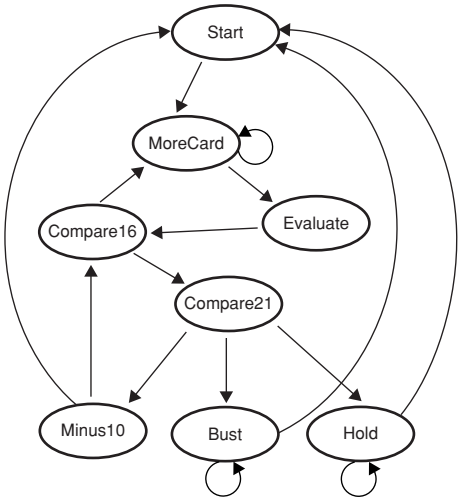
図1は，一般的な機能検証のフローを表しています．機能検証の結果が期待どおりでない場合，その原因を調べて修正しなければなりません．RTL記述が誤っていた場合にはRTL記述を修正します．仕様書が誤っていることが原因だった場合には仕様とRTL記述の両方を修正します．また仕様書の理解が誤っているようなケースでは，検証用のテストベンチや期待値が誤っている可能性があります．機能検証の工程では，RTL記述が期待どおりに動作するまで，このフローを繰り返すことになります．

1 サンプル記述と使用するツール

ここでは，ブラックジャック・ゲームのHDL記述を用いて，HDLデバッグの実際をみなさんに体験していただきます．

●サンプル記述はブラックジャック

本チュートリアルのブラックジャックでは，カードの合計が21を超えたら負け，超えない場合は勝ちとします．11，12，13は10としてカウントされ，1は1または11としてカウントされます．このゲームを繰り返し実行します．



〔図2〕チュートリアルで使用するデータの状態遷移図
遷移条件などの記述は省略した．

〔表1〕使用する変数

(a) 入力

信　号	説　明
BJ_clock	クロック
reset	リセット（アクティブ“L”）
NewCard	1：新しいカードである
NewGame	1：新しいゲームである
Card	カードの値

(b) 出力

信　号	説　明
Total	カードの合計値
NextCard	1：次のカードを要求
GameOver	1：ゲーム終了
Result	0：ノー・ゲーム 1：負け 2：勝ち
Ace	1：カードに「1」がある

サンプルの記述ファイルは本誌付属のCD-ROMに収録されているので，具体的な記述についてはそちらを参照してください．

BJkernelモジュール（BJkernel.v）はゲームの処理全般を行います．カードの合計値を計算したり，もう1枚カードを取るか取らないかを判断したり，勝敗を判定したりするモジュールです．合計値が16より大きいか小さいかを判断し，16以上の場合はそのゲームを終わり，16より小さい場合はもう1枚カードを取ります．

BJkernelモジュールの状態遷移図を**図2**に，主な変数の説明を**表1**に示します．このモジュールの状態は以下のとおりです．

- **Start** —— ゲームを開始し，変数を初期化する．
- **MoreCard** —— 新しいカードを要求する．
- **Evaluate** —— カードの合計を求める．カードが「1」の場合，変数Aceをアクティブ（“H”）にする．ここでは「1」のカードを11としてカウントする．
- **Compare16** —— カードの合計が16より大きいか小さいかを判断する．16より大きい場合は次の状態を「Compare21」に，16以下の場合はさらに新しいカードを要求するために次の状態を「MoreCard」にする．
- **Compare21** —— カードの合計値と21を比べ，勝敗を決める．合計が21よりも大きく，Aceがアクティブの場合は次の状態を「Minus10」にする．
- **Minus10** ——「1」のカードを11ではなく1としてカウントするため，カードの合計値から10引く．
- **Hold** —— 勝ち．変数GameOverをアクティブ（H）にし，ゲームが終了したことを知らせる．
- **Bust** —— 負け．変数GameOverをアクティブ（H）にし，ゲームが終了したことを知らせる．

●４種類のビューワを備えたツールを使ってデバッグ

デバッグは，設計者にとってストレスのたまる作業です．デバッグ作業そのものはEDAツールがなくても進められます．ソース・コードの入力・編集には一般のテキスト・エディタを使い，信号の検索にはテキスト・エディタの検索機能を使えます．しかし，回路が複雑になり，レガシ・コード（過去の設計資産）の流用が増えてきた現在では，この方法は効率的とは言えません．そこで登場したのが，HDL専用のデバッグ・ツール（HDLデバッガ）です．

ここではHDLのデバッグを体験するために，HDLデバッガの一つである米国Novas Software社の「Debussy」を利用することにします．このDebussyは，Verilog HDLとVHDL，RTLとゲート・レベルが混在した設計に対応しています．また，将来的にはSystemCやプロパティ言語（プロパティ検証ツールの入力言語），検証言語（テストベンチ生成ツールの入力言語）などをサポートすることが予定されています．

Debussyは複数のツール・モジュールから構成されています（**図3**）．代表的なツール・モジュールとして，ソース・コード・ビューワ（nTrace），波形ビューワ（nWave），回路図ビューワ（nSchema），状態遷移図ビューワ（nState）があります（本チュートリアル用の評価版では，これら四つのツール・モジュールを使用できる）．

これらのツール・モジュールは共通のデータベースを参照しているので，お互いがシンクロして動作します．そのため，モジュール間を自由に行き来しながらデバッグ作業を進めることができます．例えば，RTL記述はソース・コ