ハード設計者のためのPerl入門　最終回

Perl使いになろう《実践編2》

――Design Compiler用ログ・チェッカを作成する

古都 哲生

テキスト処理言語Perlの入門講座の第6回（最終回）です．今回は，前回に続いて，筆者が実際の設計業務のなかで利用していたPerlスクリプトを紹介します．これは，論理合成ツールの業界標準となっている米国Synopsys社のDesign Compilerのログ・ファイルなどをチェックするスクリプトです．なお，今回紹介するスクリプトのスケルトンはDesign Wave Magazineのホームページ（http://www.cqpub.co.jp/dwm/）から入手することができます．　（編集部）

前回は，実際に筆者が設計業務で使用していたスクリプトを紹介しましたが，みなさんの設計に応用できましたでしょうか？　今回は“Perl使いになろう”の最終回として，論理合成を支援するスクリプトを紹介したいと思います．このスクリプトも，前回同様，筆者が実際に業務に使用していたスクリプトをベースにしています．

Perlスクリプトに関わらず，設計支援ツールは設計手法と密接に関係しています．スクリプト運用の限界をわかったうえで使用すれば，大きな効果を期待することができますが，それを軽んじると大きなリスクを負う場合があります．したがって，ここで紹介するスクリプトをみなさんの設計に流用する場合には，プロジェクト・リーダと十分話し合って使用するようにしてください．

Design Compiler用ログ・チェッカの作成

論理合成を使ったLSI設計におけるPerlの実用例として，論理合成用ログ･チェック･スクリプトを紹介したいと思います．対応する論理合成ツールは米国Synopsys社のDesign Compilerです．

今回のスクリプトは，設計手法や半導体メーカ，設計対象にかなり依存しているので，たんなるスクリプトの紹介だけでは再利用しにくいと思われます．したがって，今回は筆者らの設計の考え方も含めて，Perlスクリプトの概要を説明したいと思います．

●ログ解析に時間を割いたほうが設計効率が上がる

論理合成における合成後のログ・ファイルやレポート･ファイルなどの解析は非常に重要な作業です．

論理合成ツールは，与えられたHDLを論理回路にする際に，じつにさまざまな情報をユーザに報告してくれます．ところが実際にはそれらを詳細に解析する余裕はなく，生成されたゲート・レベルの回路をシミュレーションして，結果オーライでLSIを開発している場合も多いかと思います．

じつはそこに大きな落とし穴があります．RTLの検証としては，「機能検証」がもっとも重要ですが，いったんゲート・レベルの回路になると，「機能検証」，「トグル検証」，「タイミング検証」など，さまざまな検証を実施しなければなりません．そのため，テスト・パターンの量は，RTLの場合と比較して爆発的に多くなってしまいます．

筆者らのこれまでの経験では，ゲート・レベルの検証工程に時間を割くよりも，論理合成のログやレポートの調査･分析に時間を費やしたほうが，全体の設計効率が上がる場合が多いようです．そして，設計期間を最短にする方法は，論理合成ツールにマッチしたHDLで回路を記述することのようです．

また，最近では，LSIのゲート規模の増大とともに，論理合成ツールの実行回数が多くなり，1回あたりの実行時間も長くなってきています．そのため，論理合成のログやレポートを調査･分析する機会が増えています．

ところが，開発・設計工程において，生産性に直接関与しないログ解析などの作業は優先順位が低く，さっさと次の工程に進んでしまいがちです．極端な話，エラー・メッセージが出ていなければ論理合成は完了したものとし，詳細な検証は後工程（ゲート・レベル検証工程）でカバーするという方法で突き進んでいくことが多いようです．

ここで紹介するスクリプトは，そのようなログ解析を含ん

〔リスト1〕論理合成ログ・チェック・スクリプトの出力例

```
================================================================================
  Welcome to Synopsys DesignCompiler Log Analizer v.3.2
    DATE: Mon Apr  3 22:34:17 JST 2000
    HOST: mcc21
    PATH: /xxx…/mcc/sample
================================================================================
---- ------------ ---------------- ----- -------- --⑥-- --⑦-- ---- ---- ---------- -------------- ---------------- -⑬- -------⑭------
①    ②            ③                ④     ⑤        log    check  ⑧    ⑨    ⑩          ⑪              ⑫               CLOCK  === slack ===
 No. script-name   date             CPU   op-time  Err War Err War wire scan model      Real-Area[gate] const area      Feq.    max      min
---- ------------ ---------------- ----- -------- --- --- --- --- ---- ---- ---------- -------------- ---------------- ----- ------- -------
   1 sample       2000/02/08 23:25 mcc22  0:16:06   0   5   0   0 G08C SCAN SAMPLE      38592[ 10720]  38600[ 10722]   34.00  4.39    0.93
   2 sample_seq   2000/02/08 23:08 mcc13  0:38:01   0  19   0   0 G08C SCAN SAMPLE_SEQ  10179[  2827]  10900[  3027]   34.00  3.27    1.04
   3 sample_sub   2000/02/08 22:30 mcc22  0:12:19   0   9   0   0 G08C SCAN SAMPLE_SUB   6967[  1935]   7000[  1944]   34.00 13.82    1.17
--------------------------------------------------------------------------------------------------------------------------------------------
Report                                  1:06:26                                                                    0             0       0
--------------------------------------------------------------------------------------------------------------------------------------------
 NOTE: 'V':VIOLATED, ' ':MET
--------------------------------------------------------------------------------------------------------------------------------------------
```

だ，論理合成後のチェック作業を簡単にし，だれでも同等の品質でチェックできること，そして，安全にサインオフを迎えられることを目的として作成しました．もちろん，このスクリプトを使ってログ解析の労力を軽減するためには，各設計工程でちょっとした工夫が必要になります．ここでは，それらのノウハウも含めてお伝えしたいと思います．

●論理合成ログは四つの基準で判定

まず，スクリプトの使用によって，どのような結果が表示されるのかを見ていただくのが一番わかりやすいと思います．**リスト1**がスクリプトの使用結果です．それぞれの行は，各ディレクトリで実行された論理合成ログごとの合成情報を示しています．論理合成ログは，以下の方法で取得しています．

```
% dc_shell 《スクリプト・ファイル名》 | tee 《スクリプト・ファイル名》.log
```

リスト1の行内の項目は，①通し番号，②スクリプト･プリフィックス名，③合成日時，④実行ホスト名，⑤処理実行時間，⑥ログ・ファイルのエラー個数とワーニング個数，⑦チェック・ファイルのエラー個数とワーニング個数，⑧ワイヤ・ロード・モデル名，⑨スキャン・ライブラリ使用表示，⑩モデル名，⑪実エリア(ゲート数)，⑫制約エリア(ゲート数)，⑬クロック周波数，⑭タイミング余裕(最大，最小)を示しています．また，この例では表示されていませんが，論理合成ツールのバージョンの違いや，RTLのHDLが参照するディレクトリの違いを検出することも可能です．

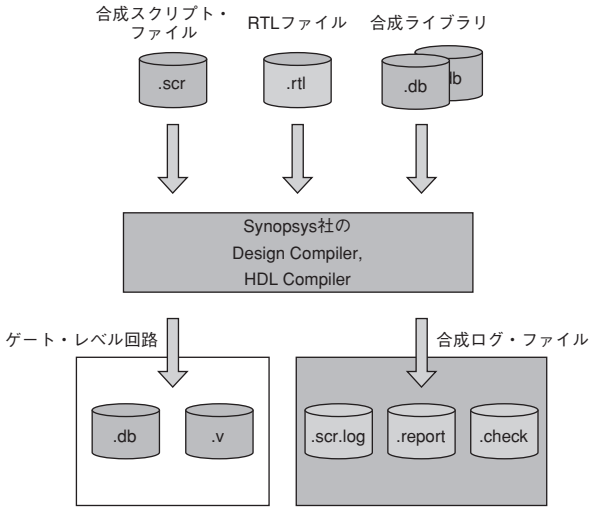
チェック基準は，その設計手法により違いがありますが，筆者らは以下の四つの基準で判定するようにしています．

⑴ **ログ・ファイルとチェック・ファイルにエラーがないこと**
⑵ **ワーニングは既知のもので，実害がないこと**
⑶ **エリア，タイミングともに制約条件を満足していること**
⑷ **設定にまちがいがないこと(スキャン・ライブラリ，ワイヤ・ロード・モデル，クロック周波数など)**

〔リスト2〕レポート・ファイルとチェック・ファイルを生成する合成スクリプト

```
(a)レポート・ファイル
        REPORT_FILE = MODULE_NAME +  ".report"
        report_area                                  >>REPORT_FILE
        report_design                                >>REPORT_FILE
        report -cell                                 >>REPORT_FILE
        report_clock  -attributes -nosplit           >>REPORT_FILE
        report_clock  -skew       -nosplit           >>REPORT_FILE
        report_constraint                            >>REPORT_FILE
        report_constraint    -verbose                >>REPORT_FILE
        report_constraint    -all_violators          >>REPORT_FILE

(b)チェック・ファイル
// 論理合成終了後
        CHECK_FILE = MODULE_NAME +  ".check"
        check_design > CHECK_FILE
```



〔図1〕論理合成のデータの流れ

米国Synopsys社のDesign Compiler，HDL Compilerは，①合成スクリプト・ファイル(Synopsys社独自のコントロール・スクリプト言語)，②RTLファイル(Verilog-HDLフォーマットで記述されたRTL)，③合成ライブラリ(半導体メーカに依存するライブラリ・ファイル)を入力とし，ゲート・レベル回路，合成ログ・ファイルを出力する．

リスト1の表は，論理合成の担当者自身も使いますし，そのプロジェクトのリーダも随時，論理合成ディレクトリで確認しています．筆者の所属する開発グループでは，サインオ