

# はじめに

論理回路設計の回路規模が増大するに従い、設計資産を有効的に再利用していくことが重要になってきています。設計資産はIP (Intellectual Property) と呼ばれ、システム・オン・チップ(System-on-a-Chip)を実現する上での重要な技術になっています。

IPは、現在、VSI(Virtual Socket Interface)という組織において共通化が計られています。IPを共通化することで、IPを複数のベンダーから購入してシステムを構築し、1つのチップ内に大規模なシステムを容易に統合することができます。そのため、従来より大規模な回路を、従来と同様の期間で設計することが可能になります。

ソフトIPは、主にHDL(Hardware Description Language)を用いて、RTL(Register Transfer Level)で記述されています。設計資産をRTL記述で残すことで、HDL記述を変更することが容易になります。また、パラメータ化された記述では、アプリケーションの目的に応じて回路を容易に変更することができます。

このようなソフトIPは、論理合成ツールを用いて、設計制約条件とテクノロジーを指定することで、最終的にゲートレベル回路に実現します。設計制約条件は、要求する回路性能としてタイミング条件や面積、動作環境などを指定します。合成時に与える制約条件が適切でない場合には、適切ではない回路として合成される恐れがあり、目的の性能を満足させることが難しくなります。また、記述が適切でない場合、目標とする性能を満足することができない可能性もあります。

本設計スタイルガイドは、設計資産の再利用手法に基づきながら、HDL設計を進めるための設計スタイルを定義したものです。設計スタイルを共通化することで、設計者によって異なる記述スタイルや合成、検証などの設計手法を共通化することができます。そのため、記述者以外の設計者がその記述を変更したり、理解するための読解性を改善することができます。

本設計スタイルガイドでは、Verilog-HDLの構文規則を理解していることを前提として記述されています。Verilog-HDLの構文規則については、IEEE1364 Verilog Hardware Description Language Reference Manualまたは、各種Verilog-HDL書籍を参照してください。Verilog-HDLによる論理回路設計では、Verilog-HDL対応のシミュレータによる検証、Verilog-HDL対応の論理合成ツールを使用することが前提となります。それらは各ツールのマニュアルを参照してください。

論理回路設計に使用する言語としては、Verilog-HDLの他にVHDLがあります。本設計スタイルガイドは、Verilog-HDLにそって記述されています。VHDLをお使いの方は、設計スタイルガイドVHDL版をお読みください。

本設計スタイルガイドは、日本の各半導体および電子機器製造会社が、自社内で適用しているVerilog-HDL(あるいはVHDL)による論理回路設計ルールを一般化し、IP流通を促進するために編纂されたものです。編集にあたりましては、株式会社半導体理工学研究センターが参画企業からの意見をまとめ、IP技術支援小委員会の合意のもと作成されています。編集にあたりご尽力頂いた半導体理工学研究センターの方々に感謝の意を表します。

2002年2月1日

長谷川 裕恭  
株式会社 エッチ・ディー・ラボ

## 設計スタイルガイド 2002 年度版発行に関して

設計スタイルガイドは、技術革新に対応するため、毎年 1 度見直しを行ないます。今回は、初版 2001 年度版で見送りとなった内容が大幅に加わりました。その他にも、理解をより深めて頂くために、解説文の強化も図りました。その結果、2001 年度版に比べ、ほぼ倍のページ数に増加しています。主な追加変更ポイントは下記の通りです。

「第 3 章 3.3. テスト容易化設計」に LSI 製造試験に関するテスト容易化に、ロジック回路設計としてどのような考慮が必要か、細かい規定と解説が追加されています。(9 節構成、新規 37 項目)

付録として、シノプシス社の論理合成ツール Design Compiler の使用方法、ノウハウ集、ケイデンス社の論理合成ツール Build Gates の使用方法とノウハウが追加されています。

「第 3 章 3.4. 低消費電力設計」で低消費電力化の考え方、3.5.7. ~ 3.5.9. にバージョン管理ツール CVS の使用方法が新たに追加されています。(新規 14 項目)

各項目のレベルを「必須」、「禁止」、「推奨」、「注意」、「参考」から、「必須」、「推奨 1」、「推奨 2」、「推奨 3」、「参考」に変更し、各項目の明確性を図りました。

「第 3 章 3.3, 3.4, 3.5.7」以外でも各項目の見直しを行ないました。その結果、14 項目が削除され、新たに 27 項目が追加されています。既存項目での 59 項目の内容が変更されています。追加、変更された項目に関しては、本文解説も変更されています。

シミュレーションによって結果が異なる原因について、検証手法については、解説が大幅に変更、追加されています。

クロックツリーシンセシスに関して、同期 RAM に関しての解説が大幅に変更されています。

本設計スタイルガイドは、以下のような章によって構成されています。

「第1章 基本設計制約」では、設計時に考慮する必要がある命名規則や考慮すべき設計スタイル、クロックの設計、同期設計や非同期設計時の考慮事項、および階層設計時の考え方など、設計を開始する際に考慮すべき基本的な設計制約について紹介します。

「第2章 RTL 記述テクニック」では、Verilog-HDL を用いて RTL 記述を作成するために必要な基本的な RTL 記述スタイルについて紹介します。記述スタイルでは、組み合わせ回路や順序回路の記述スタイル、および always 文や function 文、if 文や case 文などの使用方法についても紹介しています。

「第3章 RTL 設計手法」では、回路の分割設計を前提として、機能ライブラリの作成方法、設計資産のパラメータ化、テスト容易化設計、低消費電力設計、設計データの管理などを紹介します。この設計手法に従うことで、設計資産の再利用性が向上します。

「第4章 検証のテクニック」では、シミュレーションを実行するためのテクニックについて紹介します。紹介するテクニックには、テストベンチを記述する際のパラメータ化、タスクの使用、また、検証の進め方などがあります。

また、付録として

「A-5 Design Compiler による論理合成」と「A-6 Build Gates による論理合成」に、論理合成ツールの使用方法とノウハウを紹介しています。

本マニュアルで紹介している項目を以下のように分類しています。

項目	内容
必須	必ず守らなければならない項目を示します。
推奨 1	守らなければならない項目だが、違反した場合でも適切に対応策がとられていれば間違いのない項目を示します。
推奨 2	回路品質、可読性、様々なツール使用上の問題を考え、推奨される設計スタイルの項目を示します。
推奨 3	推奨されるスタイルだが、注意を促す内容で厳守する必要がない項目を示します。
参考	知っておいた方がよい設計ノウハウの参考事項です。

## 目次

1章 基本設計制約 .....	1-1
1.1. 命名規則 .....	1-2
1.1.1. 基本命名規則を守る .....	1-2
1.1.2. 回路名や端子名は階層構造を意識した命名規則に従う .....	1-5
1.1.3. 信号名には意味のある名前を付ける .....	1-9
1.1.4. include ファイル、パラメータ、define の命名規則(VHDL 版と異なる) .....	1-11
1.1.5. レジスタ出力はクロックまたはレジスタを意識した命名をする .....	1-13
1.2. 同期設計 .....	1-14
1.2.1. クロック同期設計を行なう .....	1-14
1.3. 初期リセット .....	1-15
1.3.1. 初期リセットは非同期リセットにする .....	1-15
1.3.2. リセットラインのハザード .....	1-19
1.3.3. 初期リセットの外部ノイズに注意 .....	1-20
1.4. クロック .....	1-22
1.4.1. クロック生成回路はモジュール化する .....	1-22
1.4.2. クロックバランスにはクロックツリー合成を使用 .....	1-23
1.4.3. ゲーテッドクロックの使用は注意 .....	1-25
1.4.4. 2系統以上のクロック .....	1-27
1.5. 非同期対策 .....	1-29
1.5.1. 非同期クロック間の信号にはメタ・ステーブルを考慮 .....	1-29
1.5.2. 非同期同周期クロック間転送にメモリを利用する .....	1-34
1.5.3. 同期 RAM のセットアップ/ホールド・マージンを確保する .....	1-35
1.6. 階層設計 .....	1-37
1.6.1. 階層の規模による制限を考慮する .....	1-37
1.6.2. 基本ブロックは組み合わせ回路入力の FF 出力にする .....	1-40
1.6.3. サブブロック (基本ブロック以下の階層) の制限に従う .....	1-41
1.6.4. 基本ブロックの上位階層にはゲートを挿入しない .....	1-43
1.6.5. データ・パス部と制御部は分離する .....	1-45
1.6.6. 20万ゲート以上の上位階層ではバッファ出力を指定する .....	1-47
2章 RTL 記述テクニック .....	2-1
2.1. 組み合わせ回路 .....	2-2
2.1.1. always 文と function 文を使い分ける (Verilog only) .....	2-2
2.1.2. function 文によって組み合わせ回路を定義 (Verilog only) .....	2-5
2.1.3. function では、引数やビット幅に対するチェックを入念にする .....	2-7
2.1.4. 論理式記述の注意点 (VHDL 版と異なる) .....	2-9
2.1.5. 条件代入文 (? :) の使用は 1 回まで (Verilog only) .....	2-12
2.1.6. 配列の範囲指定 .....	2-14
2.2. 組み合わせ回路の always 文記述 .....	2-16
2.2.1. ラッチを生成する危険を避ける .....	2-16

2.2.2. always 文内の入力信号はすべてセンシティブティ・リストに定義する .....	2-18
2.2.3. always 文の初期値記述 (Verilog only) .....	2-20
2.3. FF の推定 .....	2-24
2.3.1. FF 推定の記述スタイルを統一する .....	2-24
2.3.2. ノンブロッキング代入文とブロッキング代入文では回路が異なる (Verilog only) .....	2-28
2.3.3. エッジの異なる記述を混在させない .....	2-30
2.3.4. FF の初期値を記述内に指定しない (VHDL 版と異なる) .....	2-31
2.3.5. 入力値の固定した FF を生成させない .....	2-32
2.3.6. 非同期リセットありとなしの混在記述は禁止 .....	2-33
2.4. ラッチ記述 .....	2-35
2.4.1. ラッチ記述は組み合わせ回路と明確に区別する .....	2-35
2.5. トライステート・バッファ .....	2-38
2.5.1. トライステート・バッファは階層化する .....	2-38
2.5.2. トライステート・バスでのハイ・インピーダンスの伝播を考慮する .....	2-42
2.6. 回路構造を意識した always 文記述 .....	2-43
2.6.1. 回路構造を意識した記述を行なう .....	2-43
2.6.2. 1 つの always 文の中で複数の出力信号を定義することは避ける .....	2-45
2.7. if 文記述 .....	2-47
2.7.1. if 文はプライオリティ・ロジックを生成する .....	2-47
2.7.2. 同じ内容の条件式を削減する .....	2-49
2.7.3. if 文のネストの回数を減らす .....	2-51
2.7.4. 複数の文は必ずブロック文(begin-end)でくくる (Verilog only) .....	2-53
2.8. case 文記述 .....	2-54
2.8.1. case 文はデコーダ/エンコーダ記述を容易にする .....	2-54
2.8.2. 大きなテーブルを作成せずに、if 文等で分割する .....	2-56
2.8.3. default 項を使用する .....	2-58
2.8.4. 複雑な casex 文は使用しない (Verilog only) .....	2-61
2.8.5. parallel_case に頼った記述は禁止 (Verilog only) .....	2-63
2.8.6. if 文と case 文の混在したネスティングに注意 (VHDL 版で 2.8.4) .....	2-65
2.9. for 文記述 .....	2-67
2.9.1. 単純な繰り返し以外は for 文を使用しない .....	2-67
2.9.2. for 文におけるループ変数の演算の制限 .....	2-68
2.10. 演算子の記述 .....	2-70
2.10.1. 演算子の順序と X の代入 .....	2-70
2.10.2. 論理演算式を利用して記述を効率化する (Verilog only) .....	2-73
2.10.3. 右辺と左辺のビット幅を揃える (Verilog only) .....	2-74
2.10.4. 左右のデータ・タイプの違いに注意 (Verilog only) .....	2-76
2.10.5. 速度方向に厳しい回路ではリソース・シェアリングを行なわない .....	2-78
2.10.6. 算術演算の注意点 .....	2-80
2.10.7. 分岐条件式内の共有項は外に出す .....	2-83
2.11. ステート・マシン記述 .....	2-84
2.11.1. ミーリ型とムーア型を使い分ける .....	2-84
2.11.2. ステート・マシン回路は独立させる .....	2-87
2.11.3. FF と case 文を分離する .....	2-89
2.11.4. ステートの割り付けを考慮する .....	2-90

3章 RTL 設計手法 .....	3-1
3.1. 機能ライブラリの作成 .....	3-2
3.1.1. 機能ライブラリを作成し活用する .....	3-2
3.1.2. 再利用を考慮した記述スタイルを行なう .....	3-3
3.1.3. モジュールの入出力ポートの記述順序を統一する .....	3-5
3.1.4. RTL 記述の可読性を考慮する .....	3-6
3.1.5. モジュールの入出力の配列範囲をパラメータ化する .....	3-7
3.1.6. `ifdef を利用したパラメータ記述 (Verilog only) .....	3-9
3.2. 機能ライブラリの使用 .....	3-11
3.2.1. ライブラリは共通ディレクトリで管理する (VHDL 版とは異なる) .....	3-11
3.2.2. グローバル・パラメータは別ファイルに定義 (VHDL 版とは異なる) .....	3-12
3.2.3. コンポーネント接続にはポート名接続を使用する .....	3-15
3.2.4. 上位階層からのパラメータの書き換えは # を使用 (VHDL 版とは異なる) .....	3-16
3.2.5. 算術演算の機能ライブラリはデータ・パス設計の性能を向上する .....	3-18
3.3. テスト容易化設計(DFT) .....	3-20
3.3.1. DFT 向きのクロック、リセットを考える .....	3-23
3.3.2. ハードマクロ、非同期回路の対応 .....	3-26
3.3.3. FF 使用の制限 .....	3-28
3.3.4. ラッチ使用時の注意 .....	3-30
3.3.5. クロックラインの DFT 対策例 .....	3-32
3.3.6. リセットラインの DFT 対策 .....	3-38
3.3.7. 異なるクロック間の対策 .....	3-41
3.3.8. トライステート回路の DFT 対策 .....	3-43
3.3.9. ハードマクロ、非同期回路の対策と大規模 ASIC でのテスト戦略 .....	3-46
3.4. 低消費電力設計 .....	3-51
3.4.1. ゲーテッドクロックを利用した低消費電力設計 .....	3-51
3.4.2. 論理回路の工夫により低消費電力化を図る .....	3-54
3.4.3. クロックツリーの工夫により低消費電力化を図る .....	3-56
3.5. ソースコード、設計データの管理 .....	3-57
3.5.1. ディレクトリを目的ごとに作成する .....	3-57
3.5.2. ファイルのサフィックス名 .....	3-60
3.5.3. ファイルヘッダに必要な情報を定義する .....	3-61
3.5.4. ファイルのバージョンを管理する .....	3-62
3.5.5. ファイルのバックアップは定期的に .....	3-64
3.5.6. コメントを多用する .....	3-65
3.5.7. バージョン管理に CVS を使用する .....	3-69
3.5.8. CVS の基本コマンド checkout と commit を使用する .....	3-70
3.5.9. CVS を用いてバージョンを管理する (実施例) .....	3-71
3.5.10. CVS の履歴によって変更内容を確認する .....	3-73
4章 検証のテクニック .....	4-1
4.1. テストベンチ記述 .....	4-2
4.1.1. テストベンチの階層化を図る .....	4-2

4.1.2.	基本テストベクタ記述を用いる	4-3
4.1.3.	入力信号のタイミングに注意	4-5
4.1.4.	複数の initial 文からの代入は避ける (VHDL 版とは異なる)	4-7
4.1.5.	クロックエッジベースの記述を行なう	4-8
4.1.6.	周期はパラメータで設定	4-10
4.1.7.	複数クロックではクロックごとに信号を定義	4-11
4.1.8.	シミュレータによって結果が異なる記述 (VHDL 版とは異なる)	4-12
4.1.9.	非同期クロック間のシミュレーション	4-18
4.1.10.	処理サイクルが不明な時はハンドシェイクを活用	4-20
4.1.11.	シミュレーション結果はファイルに出力	4-22
4.2.	task 記述	4-23
4.2.1.	タスクによる構造化された記述を行なう	4-23
4.2.2.	タスクによる機能動作の記述	4-24
4.2.3.	タスクの入出力引数に注意 (VHDL 版とは異なる)	4-28
4.3.	検証の進め方	4-30
4.3.1.	検証の進め方をフロー化	4-30
4.3.2.	テスト仕様書を作成する	4-32
4.3.3.	シミュレーション・チェックリストを作成する	4-37
4.3.4.	シミュレーション結果の明確化	4-41
4.3.5.	期待値ファイルとシミュレーション結果を比較する	4-43
4.3.6.	期待値照合を実行しながらシミュレーション	4-44
4.3.7.	ビヘイビアモデルを使用して検証を効率化	4-47
4.3.8.	大規模回路での検証手法	4-49
4.3.9.	機能検証用パターンと故障検出用パターンを分離する	4-51
4.3.10.	ランダム関数を使用した検証方法について	4-52
4.3.11.	\ifdef による記述の埋め込みでデバッグを効率化 (Verilog only)	4-54
4.4.	ゲートレベル・シミュレーション	4-55
4.4.1.	ゲートレベル・シミュレーションの問題	4-55
4.4.2.	X の伝播で RTL とゲートレベルで不一致が生じる可能性がある	4-58
4.4.3.	タイミングによる誤動作に注意	4-60
4.4.4.	その他の RTL、ゲートレベルでの不一致原因	4-61
4.5.	スタティック・タイミング解析	4-63
4.5.1.	スタティック・タイミング解析のポイント	4-63
4.5.2.	スタティック・タイミング解析に合わせた回路設計	4-66

## A 付録

A-5	Design Compiler による論理合成	5-1
5.1.	dc_shell 基本コマンド	5-2
5.1.1.	コマンドスクリプト例	5-2
5.1.2.	read コマンド ( デザインの読み込み )	5-4
5.1.3.	current_design コマンド ( デザインの指定 )	5-6
5.1.4.	set_operating_conditions コマンド ( 動作環境の指定 )	5-6
5.1.5.	set_wire_load_model コマンド ( 配線遅延モデルの指定 )	5-8
5.1.6.	create_clock コマンド ( クロックの定義 )	5-10

5.1.7. set_input_delay, set_output_delay (入力、出力の遅延設定).....	5-12
5.1.8. set_driving_cell コマンド (入力ポート接続側の駆動能力設定).....	5-14
5.1.9. set_load コマンド (出力ポートの負荷を設定).....	5-16
5.1.10. set_max_area コマンド (面積制約の設定).....	5-17
5.1.11. compile コマンド (論理合成および回路の最適化).....	5-18
5.1.12. report コマンド類 (合成結果レポート).....	5-19
5.1.12.1. report_timing .....	5-19
5.1.12.2. report_constraint .....	5-21
5.1.12.3. report_reference .....	5-23
5.1.12.4. report_area .....	5-24
5.1.13. write コマンド (デザインの保存、ネットリストの生成).....	5-24
5.1.14. check_design コマンド .....	5-25
5.2. dc_shell 応用コマンド .....	5-26
5.2.1. ホールドタイム保証の設定 .....	5-26
5.2.2. 階層合成 (uniquify, set_dont_touch, ungroup, compile -inc).....	5-30
5.2.2.1. uniquify .....	5-30
5.2.2.2. set_dont_touch .....	5-31
5.2.2.3. ungroup .....	5-31
5.2.2.4. 階層合成 (compile -incremental_mapping).....	5-32
5.2.2.5. set_clock_transition .....	5-33
5.2.2.6. remove_unconnected_ports .....	5-34
5.2.3. 平坦化 (set_flatten).....	5-35
5.2.4. 構造化 (set_structure).....	5-36
5.2.5. グループパス・コマンド (group_path).....	5-37
5.2.6. set_false_path .....	5-39
5.2.7. set_multicycle_path.....	5-40
5.2.8. write_script, write_sdf.....	5-41
5.2.9. fanout, capacitance, transition .....	5-41
5.2.10. 指定した方がよい Design Compiler の変数 .....	5-43
5.3. 合成の基本方針 .....	5-45
5.4. スクリプト例 .....	5-48
5.4.1. 面積方向合成 .....	5-48
5.4.2. 速度方向合成 .....	5-49
5.4.3. 組み合わせ回路のみ合成.....	5-53
5.4.4. 複数クロック合成 .....	5-54
5.4.5. 階層合成とその考え方 .....	5-57
5.5. characterize 合成.....	5-61
5.6. 演算子を含む回路の合成 .....	5-64
5.7. 2000.11-SP2 の性能と過去バージョンとの比較 .....	5-67
5.7.1. 実行速度.....	5-67
5.7.2. 速度方向への性能向上 .....	5-68
5.7.3. if 文、case 文での性能低下 .....	5-69
5.7.4. 1998.03 から 2001.08 までの新機能 .....	5-71
5.7.4.1. 動作環境 MAX、MIN サポート (有用).....	5-71
5.7.4.2. タイムバジェット(有用だが利用することはまれ) .....	5-72
5.7.4.3. compile -inc 時のインプリメンテーション(cla、rpl)の選択注意 .....	5-73

5.7.4.4.	TNS(トータル・ネガティブ・スラック、2番目以降の遅延パスの考慮)注意 ..	5-73
5.7.4.5.	新しい boolean オプティマイズ機能(有用だが効果は少ない) .....	5-73
5.7.4.6.	マルチビット・セルのサポート(不要).....	5-73
5.7.4.7.	set_simple_compile_mode(1999.05より) .....	5-74
5.7.4.8.	compile -top(1999.05より)(有用).....	5-75
5.7.4.9.	propagate_constraints(1999.05) .....	5-75
5.7.4.10.	ACS コンパイル(2000.05より) .....	5-76
5.7.4.11.	case_analysis(2000.11より) .....	5-77
5.7.4.12.	clock gating(2000.11より) .....	5-77
5.7.4.13.	set_isolate_ports(2001.08) .....	5-78
A-6	BuildGates による論理合成 .....	6-1
6.1.	BuildGates の概要とマニュアルの参照方法 .....	6-2
6.1.1.	BuildGates の概要 .....	6-2
6.1.2.	マニュアルの参照方法 .....	6-2
6.2.	BuildGates 通常のシンセシス・フロー .....	6-3
6.2.1.	ライブラリの設定 .....	6-4
6.2.1.1.	テクノロジー・ライブラリの読み込み .....	6-4
6.2.1.2.	ターゲット・テクノロジーの設定 .....	6-4
6.2.1.3.	配線遅延モデルの追加 .....	6-4
6.2.1.4.	使用禁止セル等の指定 .....	6-5
6.2.2.	デザイン・データの読み込み .....	6-6
6.2.2.1.	デザイン・データの読み込み .....	6-6
6.2.2.2.	ジェネリック・ネットリストの構築 .....	6-6
6.2.2.3.	デザインのチェック .....	6-6
6.2.3.	制約の設定 .....	6-7
6.2.3.1.	最適化禁止指定 .....	6-7
6.2.3.2.	クロック制約の設定 .....	6-7
6.2.3.3.	入出力ポートへの制約の設定 .....	6-7
6.2.3.4.	パス例外の設定 .....	6-8
6.2.3.5.	配線負荷モデルの設定 .....	6-8
6.2.3.6.	制約のチェック .....	6-8
6.2.4.	デザインの最適化 .....	6-9
6.2.4.1.	セットアップ違反の修正 .....	6-9
6.2.4.2.	クロック制約の設定 .....	6-9
6.2.5.	レポートの生成 .....	6-9
6.2.6.	最終的なネットリストの保存 .....	6-10
6.3.	コマンド・リファレンス .....	6-11
6.3.1.	基本コマンド .....	6-11
6.3.1.1.	read コマンド類(ライブラリおよびデザインの読み込み).....	6-11
6.3.1.2.	do_duuld_generic(ジェネリック・デザインの構築).....	6-11
6.3.1.3.	check_netlist(デザインの問題点の抽出).....	6-12
6.3.1.4.	set_current_module(カレント・モジュールの設定).....	6-12
6.3.1.5.	set_top_timing_module(トップ・タイミング・モジュールの設定).....	6-12
6.3.1.6.	set_operating_condition(動作環境の設定).....	6-13

6.3.1.7. set_wire_load (配線負荷モデルの指定)	6-14
6.3.1.8. set_wire_load_mode (配線負荷モードの指定)	6-14
6.3.1.9. set_clock (理想クロックの定義)	6-15
6.3.1.10. set_clock_root (理想クロックの実際のクロック・ポートへの関連付け)	6-15
6.3.1.11. set_input_delay (入力遅延の設定)	6-16
6.3.1.12. set_external_delay (出力遅延の設定)	6-16
6.3.1.13. set_drive_cell (入力ポートのドライブ・セルの設定)	6-17
6.3.1.14. set_port_capacitance (出力ポートの負荷容量の設定)	6-17
6.3.1.15. do_optimize (デザインの最適化)	6-18
6.3.1.16. report コマンド類 (合成結果のレポート)	6-19
6.3.1.17. write コマンド類 (デザインの保存、ネットリストの生成)	6-23
6.3.2. 応用コマンド	6-24
6.3.2.1. クロック遅延の取り扱い	6-24
6.3.2.2. 分周クロックの取り扱い	6-25
6.3.2.3. デザインルール制約	6-26
6.3.2.4. パス例外指定コマンド	6-26
6.4. Constraint Translator(dc_shell ac_shell)	6-29
6.4.1. トランスレータの使用にあたって	6-29
6.4.1.1. トランスレータのセットアップ	6-29
6.4.1.2. SDC 制約	6-29
6.4.1.3. サポートしている制約	6-30
6.4.1.4. サポートしていない制約	6-30
6.4.2. 制約の交換	6-31
6.4.2.1. read_dc_script コマンド使用方法	6-31
6.4.3. dc_shell コマンドと ac_shell コマンドの対応関係	6-33
6.5. ノウハウ集	6-35
6.5.1. BuildGates の性能を最大限に発揮させるために	6-35
6.5.1.1. できる限りトップダウンで合成する	6-35
6.5.1.2. 厳しすぎる制約を与えない	6-35
6.5.1.3. 階層をまたぐクリティカル・パスには注意する	6-35
6.5.2. 知っているると便利な機能	6-36
6.5.2.1. 入力したコマンドを画面やログ・ファイルに表示させる	6-36
6.5.2.2. タイミング・レポートをカスタマイズする	6-36
6.5.2.3. ネットやインスタンスの名前を変更する	6-36
6.5.2.4. ネットやインスタンスの名前を変更する	6-37
6.5.2.5. ジョブの実行途中でデータベースを出力する	6-37
6.5.2.6. Tcl スクリプトを利用する	6-38
6.5.2.7. タイミング・レポートの表示桁数を増やす	6-38
6.5.2.8. グローバル変数の設定を出力する	6-38
6.5.2.9. 初期設定ファイル	6-39
6.5.3. スクリプトの例	6-40
6.5.3.1. 標準的なスクリプト	6-40

## 3.3.1. DFT 向きのクロック、リセットを考える

クロックは、LSI 外部入力端子から直接制御可能にする	必須
2系統のクロックを切り替えて使用している場合、テスト時はどちらか片方のクロックに固定する	必須
ランダムロジックの出力をクロックにしない	推奨 1
FFのリセットは、LSI 外部入力端子から直接制御可能とする	必須

## 解説

スキャンを挿入するにあたって、もっとも重要なことは、スキャンテスト時にスキャンシフトが安全に行なわれる回路構造をとることです。スキャンシフトが安全に行なわれる回路構造とは、スキャンシフト時にスキャンパスに接続される全てのFFのクロック端子が、外部入力端子で直接制御可能なことと、スキャンパスに接続される全てのFFの非同期セット、リセット信号が、スキャンシフト中にアクティブにならないような回路構造をいいます。もし、非同期セット、リセット信号がスキャンシフト中にアクティブになるような回路構造だった場合、スキャンシフト中にFFのデータを破壊（FFをセットまたはリセットしてしまう）してしまい、正常にデータのシフトが行なえなくなってしまいます。したがって、非同期セット、リセット信号は、スキャンシフト中に非アクティブとなるような回路構造が必要となります。

クロックは、LSI 外部入力端子から直接制御可能にする

スキャンを挿入するためには、スキャンテスト時にFFのクロック端子を外部入力端子から直接制御できるようにする必要があります。FFのクロック端子が外部入力端子から直接制御できない場合、スキャン挿入ツールは、このFFをスキャンの対象から除外してしまいます。また、スキャン挿入されていない部分に関しては、ATPGツールによって故障を検出することが困難になってしまいます。

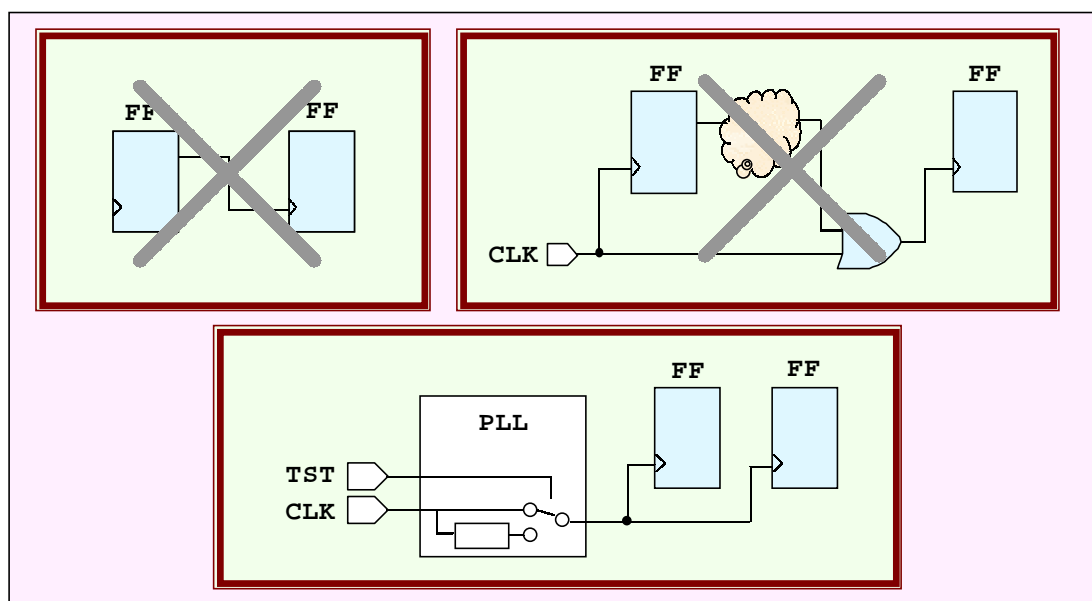


図 3-3 内部で発生したクロック、直接制御できないクロック、PLLの例

### 3.3. テスト容易化設計(DFT)

図3-3の上図のように、FFの出力がFFのクロック端子に接続されている場合や、FFのクロックラインがゲーティングされているような場合、そのFFはスキンの対象から除外されてしまいます。このような場合は、スキンのテスト時に、クロックラインを切り替えるなどして、外部入力端子から直接、制御可能にする必要があります。また、LSI内部のハードマクロの出力がクロックになっているものは、スキンの挿入ができません。LSI内部では、PLLやDLLといったクロック発生マクロを利用します。通常このようなマクロには、図3-3の下図のようにテスト端子がついていて、テストモード時には、PLLに入力した基準クロックが、そのままクロックラインに出力されます。もし、このような切り替え回路がついていないPLL、DLLを利用する場合には、設計者が、外部クロックとの切り替え回路を挿入する必要があります。

「クロックは外部入力端子から直接制御可能にする」に関して、詳しい対策方法は「3.3.5. クロックラインのDFT対策例」で解説します。

2系統のクロックを切り替えて使用している場合、テスト時はどちらか片方のクロックに固定する

クロックが、LSI外部入力端子から制御可能であっても、図3-4のように2つのクロックを切り替えて使用している場合、スキンの挿入ができません。図の様にセレクタの選択信号をテスト信号によって固定し、スキンのテスト時は、常時どちらか一方のクロックを選択している必要があります。

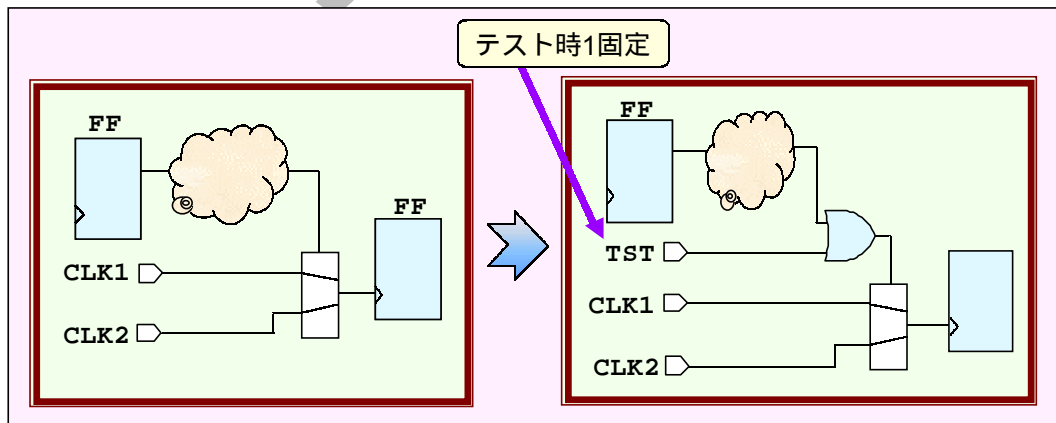


図3-4 2系統のクロックを切り替えている回路のDFT対策例

ランダムロジックの出力をクロックにしない

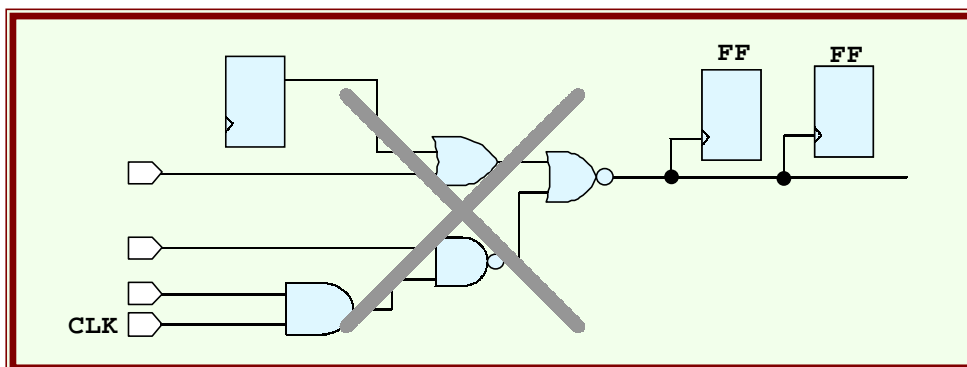


図3-5 ランダムロジックの出力をクロックにしない

図3-5のように、ランダムロジックの出力をクロックに使用している場合は、ランダムロジックの最終出力部分にセレクタを挿入し、外部クロックと切り替えられるようにしてください。

クロックラインにANDゲート、ORゲートあるいはセレクタを利用する方法は「3.3.5. クロックラインのDFT対策例」で解説します。

FFのリセットは、LSI外部入力端子から直接制御可能とする

スキャンを挿入するためには、クロック系だけでなく、リセットラインにも対策を施す必要があります。リセットラインの対策としては、スキャンシフト中、FFにリセットがかかり、シフトデータを破壊することがないようにする必要があります。テスト信号等により、LSI内部でリセットラインが非アクティブとなるように固定した場合でもスキャン挿入は可能ですが、この場合、ATPGツールによって、リセットラインの故障が検出できないこととなります。したがって、リセットラインは外部入力端子から直接制御可能にする必要があります。

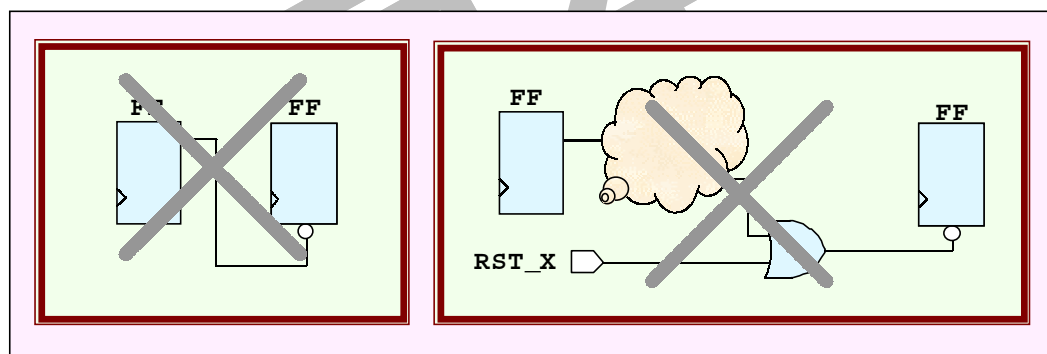


図3-6 内部で発生したりセット、外部入力端子から直接制御できない回路

図3-6の左図のように、FFの出力がFFのリセット端子に接続されている場合、このFFは、スキャンの対象から除外されてしまいます。また、図3-6の右図の回路では、RST\_Xを'1'に固定することで、FFのリセットをディセーブルにすることができますので、スキャン挿入は可能ですが、この場合、FFのリセットラインの故障が検出できないこととなります。したがって、これらの回路は、外部入力端子から直接制御できるようにしなければなりません。

リセットはノイズ対策、あるいはタイミングの問題でFFを何段も挿入して同期化することが一般的ですが、このような場合も、外部端子から直接制御することができませんので、対策が必要となります。リセットについての諸問題は、「3.3.6. リセットラインのDFT対策」で解説します。