■10 群（集積回路）－ 1 編（基本構成と設計技術）

3 章 集積回路設計

（執筆者：編幹事団）［2009 年 4 月 受領］

■概要■

【本章の構成】

この章では，3-1 節“システム仕様設計”，3-2 節“機能・動作設計”，3-3 節“論理設計”，3-4 節“回路設計”，3-5 節“レイアウト設計”，3-6 節“マスク設計”，及び 3-7 節“テスト設計”において，各設計工程別に種々の設計技術を概説する．また，消費電力の最小化を設計目標とした場合の設計技術を，3-8 節“低消費電力設計”において述べる．なお，これらの設計技術を，設計工程別並びに以下の三つの作業別に概観する 2 次元目次を，付録に示しておく．

（Ⅰ）モデル化（Modeling），データ表現（Representation）
（Ⅱ）合成（Synthesis），最適化（Optimization）
（Ⅲ）検証（Validation），解析（Analysis），シミュレーション（Simulation）

■10 群 － 1 編 － 3 章

3-1 システム仕様設計

3-1-1 仕様記述

(執筆者 : 山田晃久) [2008 年 9 月 受領]

LSI の仕様としては，大きさ，製造プロセス，価格，電源電圧，消費電力，ピン数，パッケージの種類など様々なものがあるが，動作と構造に注目して記述されることが多い．記述方法は自然言語のほかに，アルゴリズムを表現するために擬似プログラミング言語，フローチャートなどで記述され，アーキテクチャを表現するためのブロック図などで記述される．

仕様設計での誤りは大きな手戻りを生じさせてしまうため，仕様を可視化し，仕様漏れ，仕様誤りを防ぐために，もともとソフトウェアのモデリング用に開発された UML（Unified Modeling Language）[1] を用いて LSI の仕様を記述し，下流工程の設計データに自動的に変換する方法も提案されている [2].

■参考文献

1) ISO/IEC 19501:2005 Information technology -- Open Distributed Processing -- Unified Modeling Language (UML) Version 1.4.2
2) Robert Thomson, V. Chouliaras and David Mulvaney, “From UML to structural hardware designs,” Proceedings of the 44th Design Automation Conference (DAC 2007), San Diego, CA, USA, 2007.

3-1-2 システム検証

(執筆者 : 浜口清治) [2009 年 2 月 受領]

システム検証（System Verification）は，システム全体の動作を確認する作業をいう [1, 2]．ここでは，設計の最も上位に当たる仕様記述に対する検証について述べる．

検証手法は，与えられた記述がどのようなものであるかに依存する．C, C++, SystemC, SpecC, MATLAB の M-code など実行可能な記述として与えられた場合は，それらの動作を検証することになる．一般にソフトウェアの検証手法を適用することも可能である．通常，システムに対する要件をすべて網羅するようなテスト入力を作成・実行して，期待どおりの動作を実現しているかどうかを確認する．記述中に，満足するべき条件（要件）をアサーションの形式で書き込むことが可能な場合，実行結果の確認は容易になる．また，これらの記述が比較的小規模のものであれば，モデル検査のようなフォーマル手法によってアサーションをチェックすることも考えられる．

一方，UML のユースケース（Use Case Diagram）図などでは，上記のような機械的な実行はできないが，レビューを行ってシステムに対する要件がもれなく記述されているか検討する必要がある．

3-1-3 HW/SW 協調設計

(執筆者 : 浜口清治) [2009 年 2 月 受領]

ハードウェア／ソフトウェア協調設計（Hardware/Software Codesign, HW/SW Codesign）[3, 4, 5] は，広い意味では，ハードウェア開発とソフトウェア開発を並行して行うことをいう．ここでは，狭い意味でのハードウェア／ソフトウェア協調設計を取り上げる．すなわち，システム全体に対する仕様を複数の機能モジュールに分割した後，各機能モジュールをハードウェアまたはソフトウェアへ振り分けて，全体のアーキテクチャを決めていく設計手法である．この際，性能，消費電力，柔軟性や設計に要する期間についてトレードオフを勘案しながら

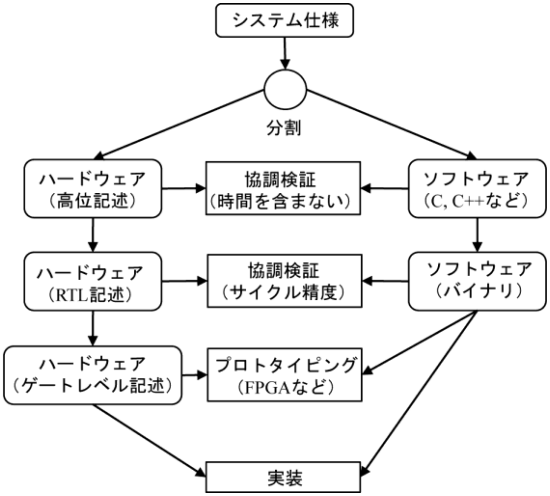

図 3・1　HW/SW 協調設計

振り分けを行うこととなる．典型的な HW/SW 協調設計のフローを**図 3･1** に示す．

ハードウェアとソフトウェアへ振り分ける前に，システム全体の機能を分割する．システム全体の動作が SystemC や SpecC などで，並行プロセスとして記述されている場合は，適宜詳細化しながら各機能モジュールを決定する．C, C++ など逐次型言語で記述されている場合には，手作業または自動的な手段で並行性を抽出して分割することになる［本章 3-1-4 節参照］．各機能モジュールは通信と計算を分けて記述することが望ましく，システムレベルの記述言語はこれを可能にするための構文を備えている．ここでは計算部分をビヘイビア（Behavior）と呼び，通信部分をチャネル（Channel）と呼ぶことにする．この場合，機能分割後のシステムの記述は，ビヘイビア間をチャネルがつないだかたちのネットワークとなる．この時点では，それぞれのビヘイビアやチャネルの動作については，クロックサイクル精度の動作などタイミングに関する詳細は決まっていない．

次にビヘイビアの集合を分割して，複数のハードウェア及びソフトウェアのモジュール(プロセシング・エレメントなどと呼ばれる）へ割り当てる．割当ての際，性能・消費電力などの見積りを行って，システム全体に対する要件が満足されるようにする．また，最適な分割を探索する．この後それぞれのモジュールを実装する．詳細化を行ってハードウェアに関しては RTL 記述を，ソフトウェアに関してはプログラムを得ることになる．チャネル部分についても詳細化を行う必要があり，これは通信設計などと呼ばれる〔本章 3-1-4 節参照〕．例えば，チャネルがバスとして実現されるのであれば，クロックサイクル精度の通信プロトコルの決定と実装が必要となる．プラットフォームベース設計の場合は，割当て先のアーキテクチャがほぼ固定されているため，ハードウェア設計，通信設計は不要または軽減化される．

ハードウェア／ソフトウェア協調検証（Hardware/Software Coverification, HW/SW Coverification）とは，ハードウェアの記述とソフトウェアの記述を同時に検証することをい

う．それぞれのモジュールが SystemC や SpecC など同一の言語で書かれている場合は，既存のシミュレーション環境があるため，検証環境の構築が比較的容易に行えるが，この場合でも，モジュールの詳細度が異なる（例えば，クロックサイクル精度でのインタフェース記述をもつモジュールとそうでない記述が混在している）状況で協調検証を行う場合は，変換器を挟んで通信の整合性を保つ必要がある．RTL 記述とプログラム記述（をコンパイルしたコード）に対する協調検証の場合，普通，RTL 記述についてはハードウェア記述言語に対するシミュレータが，プログラム記述については命令セットシミュレータ（Instruction Set Simulator, ISS）が用いられる．協調シミュレータが両者間の同期機構を提供する．一般にこのレベルでの検証は，タイミングに関する詳細を含まない検証よりも，精度の高い検証を行える反面，1 桁以上低速となる．

■参考文献

1) M. Fujita, I. Ghosh and M. Prasad, “Verification Techniques for System-Level Design,” Morgan Kaufmann, 2008.
2) 藤田昌宏（編著）, “システム LSI 設計工学,” オーム社, 2006.
3) G. De Micheli and M. Sami (Ed.), “Hardware/Software Co-Design,” Kluwer Academic Publishers, 1996.
4) J. Staunstrup and W. Wolf (Ed.), “Hardware/Software Co-Design: Principles and Practice,” Kluwer AcademicPublishers, 1997.
5) G. De Micheli, R. Ernst, and W. Wolf (Ed.), “Readings in Hardware/Software Codesign,” Morgan Kaufmann Publishers, 2001.

3-1-4 システム合成

（執筆者：戸川　望）[2008 年 4 月 受領]

システム合成とは，アルゴリズムやアーキテクチャ，インタフェースなどのシステムレベルの仕様記述から，レジスタ転送レベルの記述を自動的に合成することを指すことが多い．システム合成のための要素技術として，ハードウェア／ソフトウェアの自動分割や，インタフェースの自動生成などがある．

一般に，設計対象となるアプリケーションをすべてソフトウェアで実現すれば，ハードウェアとしてそのソフトウェアを実行するマイクロプロセッサのみが必要であり，そのほかのハードウェアは必要としない．アプリケーションに含まれるすべての処理をマイクロプロセッサでソフトウェア実行するために，アプリケーションの実行時間が多くかかる．一方，設計対象となるアプリケーションの一部（あるいは全部）を何らかの専用ハードウェアで実現すれば，マイクロプロセッサのほかに専用ハードウェアが必要となり面積が増加する．アプリケーションに含まれる処理が専用ハードウェアにより高速化されるため，アプリケーションの実行時間は少なくなる．また付加的な専用ハードウェアが増加するため，電力消費も増加する可能性がある．実行時間，チップ面積，電力消費の 3 点を，計算機によって最適化するのがハードウェア／ソフトウェアの自動分割である．ハードウェア／ソフトウェアの自動分割では，設計対象となるアプリケーションの動作記述を，タスクグラフ（Task Graph）やコントロールデータフローグラフ（Control/Data-Flow Graph）などの中間的な表現に変換し，その後，これらのグラフ中のノードをハードウェアあるいはソフトウェアに割り当てる．割当ての方法により実行時間，チップ面積，電力消費の 3 点が変化するが，例えば，実行時間並びにチップ面積を決められた制約値以下に抑えたうえで，電力消費を最小化するような割当てを決定する．このような割当てには，総探索，分枝限定法，あるいはシミュテーティッ

ドアニーリング法や遺伝的アルゴリズムといった確率的な算法が用いられる．

ハードウェア／ソフトウェアの自動分割により，二つの処理が，別々のハードウェアやソフトウェアに割り当てられると，その間のインタフェースを取る必要が出てくる．インタフェース回路の自動生成は，このような要求から登場する技術で，単に AMBA バス [1)]（AMBA とは，Advanced Microcontroller Bus Architecture の頭文字を取ったものである）のような標準のインタフェースを使用することを目指したものや，正規表現などによってインタフェースを形式的に記述し，独自のインタフェースを自動生成する技術 [2)]もある．

■参考文献

1) ARM: http://www.jp.arm.com/.
2) 鈴木敬，荒宏視，矢野和男，中田恒夫，岩下洋哲，古渡聡，“インタフェースに着目した IP 再利用設計手法‘インタフェース・セントリック・デザイン’の提案，”第 15 回回路とシステム軽井沢ワークショップ, 2003.

■10 群 － 1 編 － 3 章

3-2 機能・動作設計

3-2-1 モデル表現

（執筆者：山田晃久）[2008 年 9 月 受領]

(1) アルゴリズム・動作レベル

アルゴリズムや動作の仕様を形式的にモデル化するために，複数のプロセスが互いに通信するプロセスネットワークやペトリネット（Petri Net），有限状態機械ベースの計算モデルなどが提案されている [1].

動的な検証目的では，C や C++などのプログラミング言語を用いて表現されることが多い．このレベルの抽象度はあいまいであり，回路化する際に必要となる演算精度，並列性，時間などの情報を含まないレベルから，これらの情報のすべて，または一部を含むレベルまで設計方法，設計工程により使い分けられる．抽象度のあいまいさを解消し，モデルの再利用性を高めるための活動も行われている [2].

(2) レジスタ転送レベル（Register Transfer Level，RTL）

同期回路設計で，1 クロックサイクル（1 状態）の間にレジスタとレジスタの間でデータ転送をして処理する機能を定義することで，回路の動作を表現する．論理合成ツールを用いて設計する場合には，VerilogHDL や VHDL などのハードウェア記述言語を用いてレジスタ転送レベルで設計が行われる．このレベルでは，クロックサイクルの精度で検証ができる．

(3) コントロールデータフローグラフ（Control/Data Flow Graph，CDFG）

システムの動作を表現するグラフの一つで，動作合成を行うための計算機の内部表現として用いられる．コントロールグラフは，動作の中の分岐やループなどの制御の流れを表し，データフローグラフは，演算間のデータの流れを表す．統一された CDFG の表現方法はなく，コントロールグラフとデータフローグラフを個別に表現する方法や，制御の流れとデータの流れを一つに表現する方法が用途に応じて用いられる．

(4) 有限状態機械（Finite State Machine，FSM）

システムの動作を，有限個の内部状態と，その状態間の遷移によって表現する．システムの内部状態を S，外部入力を I，出力を O とするとき，ある状態から次の状態への遷移を決定する次状態関数 f は $f : S \times I \rightarrow S$ と表現できる．システムの出力 O を決定する出力関数 g の表現方法は二つある．一つはミーリ型と呼ばれるもので，出力 O は内部状態と外部入力の組合せにより決まり，$g : S \times I \rightarrow O$ と表現できる．他方は，ムーア型と呼ばれるもので，出力 O は内部状態のみによって決定され，$g : S \rightarrow O$ と表現できる．

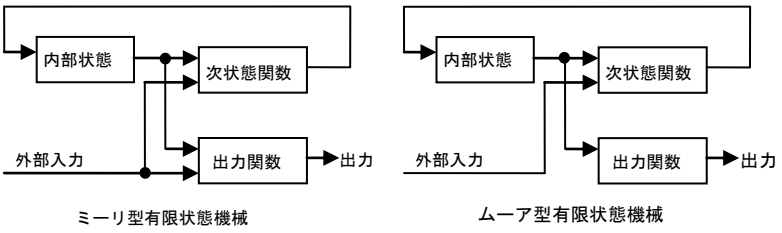

図 3・2 ミーリ型有限状態機械とムーア型有限状態機械

(5) データパスと制御回路

データ処理を行う回路設計において，データ処理を行う部分（データパス）とデータパスの動きを制御する部分（制御回路）に分けて表現する．データパスは，データの演算を行う，加算器，乗算器，ALU などの演算器，データを保持するためのレジスタ，それらの間を接続する通信パスからなる．制御回路は，データを処理するためのデータパスのすべてのサイクルの動作を規定する必要があり，一般的には有限状態機械として表現される．データパス中の回路素子は，制御回路からの制御信号に基づいて動作する．制御回路は，内部状態とデータパスからの出力信号に基づき状態遷移を行いながら，データパスに制御信号を出力する．

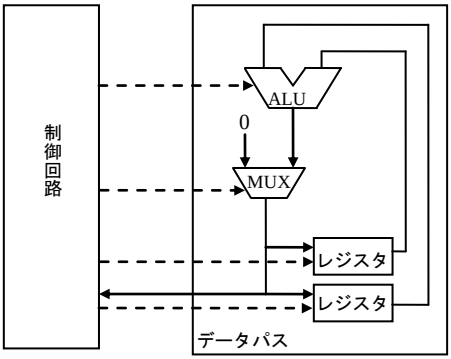

図 3・3　データパスと制御回路

■参考文献
1) 黒坂，温，荒木，吉永，齋藤，野々垣，大塚，竹村，塚本，“研究・標準化動向分析にもとづくシステムレベル設計手法の提案 －JEITA SLD 研究会の活動から－，”情報処理学会，DA シンポジウム 2003 論文集，pp.183-196，2003.
2) (株)半導体理工学研究センター，“TL モデリングガイド，”初版，2007.

3-2-2 動作合成

（執筆者：戸川　望）[2008 年 4 月 受領]

動作合成とは，コントロールデータフローグラフを入力として，演算要素，記憶要素，転送要素から構成されるデータパスと，データパスを制御する制御回路を合成する工程である [1]．ここに，演算要素とは主に演算器などを指し，記憶要素とはレジスタやメモリを指し，転送要素とはマルチプレクサやバスなどを指す．

動作合成の工程は，完全に決まった手順が確立しているわけではないが，おおよそアロケーション，スケジューリング，バインディング，制御回路合成と呼ばれる各処理によって構成される．また，動作合成の各処理に導入されている要素技術として，チェイニング（Chaining），モジュール合成，プロトコル変換，自動パイプライン化などがある．

(1) アロケーション（Allocation）

動作合成におけるアロケーションという用語は，多くの意味で用いられる．一般に広い意味でアロケーションとは，コントロールデータフローグラフ中の演算に対し演算要素（演算

器アロケーション）を，変数に記憶要素（レジスタアロケーション）を，演算器への変数の受け渡し，すなわちデータ転送に転送要素（転送路アロケーション）を，割り当てることをいう．アロケーションは，広義にはこれら全体を意味するが，狭義には，アロケーションの工程を更に分割して，演算要素，記憶要素，転送要素として何を用いるかを決定するセレクション（Selection，あるいはユニットセレクション（Unit Selection）ともいう），具体的に演算をどの演算要素に割り当てるか，変数をどの記憶要素に割り当てるか，データ転送をどの転送要素に割り当てるかを決定するバインディング（Binding，あるいはユニットバインディング（Unit Binding））に分けることがある．また，アロケーションという用語を，ユニットセレクションの意味で用いる場合もある．とりわけ，次小節で説明するスケジューリングの前工程としてのアロケーションはユニットセレクションの意味合いが強い．

ユニットセレクションの意味でのアロケーションの主たる処理の一つは「コントロールデータフローグラフの各演算をどの演算器で実行するか」を決定する工程である．例えば，データフローグラフ中に「加算」を表すノードがあったとしよう．加算を実行するには，加算器（加算だけを実行する演算装置）を用いる，あるいは，加算をはじめ各種の多機能な演算を実行することが可能な ALU（Arithmetic Logic Unit）を用いることもできる．あるいは，加算が可能な新規の演算器を設計することも考えられる．加算器を用いると，ALU に比較して，加算器単体としての面積や実行速度の面で有利である．一方，ALU を用いると，加算だけでなく，ほかの演算と ALU を共有することが可能なため，データフローグラフ全体の実行にとって，総演算器要素数を小さくできる可能性がある．どのような演算器を用いるかは，全体の設計方針や，実行速度，チップ面積，消費電力などの制約，IP 再利用などの要因による場合が多い．変数並びにデータ転送のアロケーションも同様である．

(2) スケジューリング（Scheduling）

スケジューリングとは，コントロールデータフローグラフの各ノードを，コントロールステップあるいは制御ステップと呼ばれる時間区切りに割り当てる処理である．コントロールステップは，状態遷移機械の一つの状態に対応し，状態遷移機械からの制御信号によって，データパスが制御されることになる．また，データパスの演算結果を制御回路にフィードバックすることによって状態遷移機械の次状態や出力が決定し，その結果として，データパスが制御される．コントロールステップで与えられる時間区切りは，そのハードウェアの動作周期を与えることになる．コントロールフローグラフの一つのノードに一つのデータフローグラフを対応付けたものと考えると，コントロールデータフローグラフはデータフローグラフの集合としてとらえることができるため，以下では，データフローグラフに対するスケジューリングについて概観する．

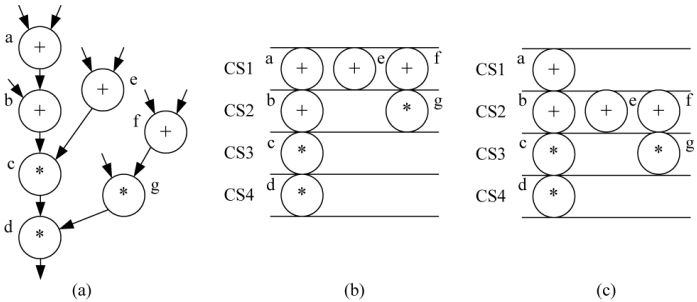

図 3・4 (a) データフローグラフ (b) ASAP スケジューリング (c) ALAP スケジューリング

データフローグラフのスケジューリングには，ASAP スケジューリング（As-Soon-As-Possible scheduling）と ALAP スケジューリング（As-Late-As-Possible scheduling）と呼ばれる基本的なアルゴリズムがある（**図 3･4**）．ASAP スケジューリングとは，データの依存制約を満足する限り，できるだけ上詰めでノードをスケジューリングするものであり，ALAP スケジューリングとは，逆に，できるだけ下詰めでノードをスケジューリングするものである．あるノード i が，ASAP スケジューリングでコントロールステップ s_i に，ALAP スケジューリングでコントロールステップ l_i に割り当てられたならば，その差($l_i - s_i$)は，ノードの i に対して割り当ての自由度を与え，モビリティなどと呼ばれる．例えば，図 3･4 では，ノード g は，ASAP スケジューリングではコントロールステップ 1（CS1）に，ALAP スケジューリングではコントロールステップ 2（CS2）に割り当てられているため自由度は 2－1＝1 となる．ノード g は，必ずコントロールステップ 1 かコントロールステップ 2 のどちらかに割り当てられることとなる．

データフローグラフのスケジューリングには，演算器やレジスタなどの「資源」をあらかじめ与えておき必要となる総コントロールステップ数を最小化する資源制約スケジューリングと，総コントロールステップ数をあらかじめ与えておき必要となる資源を最小化する時間制約スケジューリングがある．ASAP スケジューリングと ALAP スケジューリングによって，各ノードのモビリティが算出されたという前提のもとに，各種のアルゴリズムが提案されている．例えば，資源制約スケジューリングの代表的なものにリストスケジューリング [1]，時間制約スケジューリングの代表的なものにフォースダイレクティッドスケジューリングがある [2]．

(3) バインディング（Binding）

バインディングとは，スケジューリングが完了した後に，コントロールデータフローグラフの演算を個別の演算要素に，変数を個別の記憶要素に，データ転送を個別の転送要素に割り当てる処理である．同じ時刻に実行される演算は，同じ演算要素に割り当てることはできないが，異なる時刻で実行される演算は，同じ演算要素を共有することができる．変数やデータ転送についても同様である．このような制約を満足し，必要な演算要素数，記憶要素数，転送要素数を最小化することがバインディングの目的となる．

バインディング問題を解法する手法は数多く提案されてきたが，バインディングに関する

制約を整数計画問題として定式化し解法する手法や，制約条件をグラフによって表現し，グラフの最小クリーク分割問題として定式化することで，バインディング問題を解法する手法が提案されている．また変数のレジスタへのバインディングについては，特定の変数 x が使用され始める時刻から使用が完了する時刻で決まる区間（これを変数 x のライフタイムと呼ぶ）を列挙し，ライフタイムを LSI のチャネル配線問題のセグメントと同一視することで，レフトエッジアルゴリズム [3]が適用できる．レフトエッジアルゴリズムは，セグメント数を n とすると，$O(n \log n)$の計算複雑度で，必要なチャネル数（あるいはレジスタ数）を最小化することができる．

(4) 制御回路合成

アロケーション，スケジューリング，バインディングが完了すると，データパス部分の全体設計が完了する．データパスに対して，これを制御する信号を入出力する回路が制御回路となる．制御回路の合成では，一般的な論理合成手法をそのまま適用することができる．

(5) チェイニング

一つのコントロールステップに，縦属接続された二つ以上の演算を割り当てることをチェイニングあるいは演算チェイニングという．縦属接続された演算 i と演算 j が一つのコントロールステップに割り当てられたとしよう．この場合，演算 i の出力結果はレジスタに保存されることなく演算 j の入力となる．つまり，演算 i と演算 j は配線によって結線されることになる．また，演算 i の実行と，これに続く演算 j の実行が一つのコントロールステップの時間区切りで完了する必要があるため，演算 i の実行時間と演算 j の実行時間と和が，クロック周期よりも短くなる必要がある．

(6) モジュール合成

動作合成における演算要素や記憶要素，転送要素は，既設計のライブラリあるいは IP（Intellectual Property; 知的財産）を使用することもできるが，特定の機能をもった演算要素などを新規に設計することもできる．モジュール合成とは，このように演算要素などを設計することをいう．モジュール合成では，マニュアル設計によって新たな演算器を設計することもあれば，モジュール合成ツールを利用して，必要な機能をもった演算器を設計することもある．

(7) プロトコル変換

複数のハードウェアやプロセッサが何らかの通信をするとき，共通のプロトコルをもてばいいが，動作合成などで新規のハードウェアが設計されると，独自のプロトコルをもつ場合がある．このようなハードウェアが，既設計のハードウェアと通信する場合，何らかのプロトコルの変換が必要となる．

プロトコル変換には，大きく分けて二つの手法がある．一つ目はプロトコル変換を実現するライブラリを使う手法である．これは，各種の標準的なプロトコルに対して，互いに変換するライブラリをあらかじめ用意して用いる手法である．どのようなライブラリを用意しておくかが，設計の品質を左右する．もう一つは二つのプロトコルからプロトコル変換器を自動に合成する手法である．プロトコルを正規表現や状態遷移機械など形式的な方法で記述し，これらから互いの信号をやりとりする新たなプロトコル変換器を合成する．

(8) 自動パイプライン化

ディジタル信号処理などに多く見られるように，動作記述にループが含まれるとき，ループアンローリング（Loop Unrolling），ループフォールディング（Loop Folding）と呼ばれる最適化技術が適用できる場合がある．

ループアンローリングとは，複数のループを展開して，展開されたループ全体を一つのデータフローグラフとみなし，最適化する技術である（**図 3･5**(a)と(b)）．図 3･5(b)では，2 個のループを展開して，一つのデータフローグラフとみなし動作合成したため，1 個のループと比較して多くの並列性を引き出すことができ，結果的に，短い時間で全ループを完了することができている．

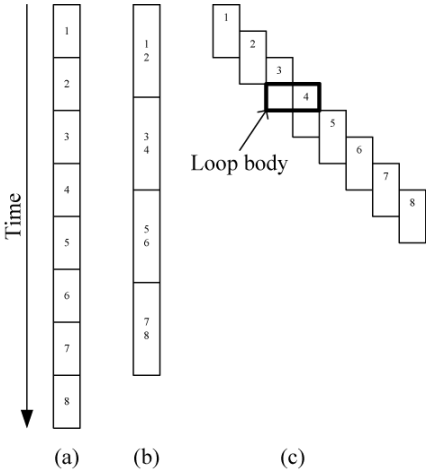

図 3・5 (a) 8 回のループ (b) ループアンローリング (c) ループフォールディング

ループフォールディングとは，ループパイプライン化あるいはパイプライン化とも呼ばれ，（図 3･5(a)と(c)）のように，一つのループが完了する前に，次々に，次のループの実行を開始するものである．この場合，例えば，3 回目のループの後半と，4 回目のループの前半は，並列して実行されていることになる．また，このようなループの本体のなる部分はループボディと呼ばれる．ループフォールディングでは，全体のループの最初と最後を除いて，ループボディが繰り返し実行されていることになる．ループフォールディングによって，ループ全体の実行が短縮されていることが分かる．

ループアンローリング並びにループフォールディングを自動化するためには，異なるループ間の依存関係を考慮して動作合成を実行する必要がある．

■参考文献

1) D. Gajski, N. Dutt, A. Wu, and S. Lin, “High-Level Synthesis: Introduction to Chip and System Design,” Kluwer Academic Publishers, Boston/Aordrecht/London, 1992.

2) P.G. Paulin and J.P. Knight, “Force-directed scheduling for the behavioral synthesis of ASIC’s,” IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol.8, no.6, pp.661-679, 1989.
3) A. Hashimoto and J. Stevens, “Wire routing by optimizing channel assignment within large apertures,” in Proc. Design Automation Workshop, pp.155-169, 1971.

3-2-3 機能検証

（執筆者：浜口清治）[2009 年 2 月 受領]

機能検証（Functional Verification）は，設計が意図した機能を実現しているかどうかを確かめることをいう[1, 2]．RTL 設計記述を対象とした検証を指すことも多いが，より一般的には，更に上位の記述に対する検証も含む．本節では機能検証の枠組みと技術について述べる．

検証の方式には，大きく分けて動的検証と静的検証がある．

(1) 動的検証

動的検証（Dynamic Verification）では，テストパターンを準備して，設計記述に与えた際の振る舞いをシミュレーションすることによって，設計の正しさを検証する．テストベンチ（Testbench）はシミュレーションを行うための仮想的な環境であり，通常**図 3･6** のように設計回路に加え，パターン生成器（Pattern Generator）やモニタ（Monitor）などのモジュールが付加されている．これらのモジュールは，VHDL，Verilog などのハードウェア記述言語や e, SystemVerilog など高度の記述が可能なハードウェア検証言語によって記述される．

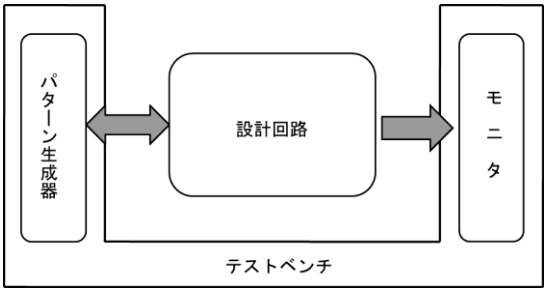

図 3･6　テストベンチの基本構成

パターン生成器はテスト用のパターンを設計記述に与えるためのモジュールである．設計回路がインタラクティブな通信を行うものである場合は，設計記述からの出力を観測して，インタラクティブにテストパターンを発生する必要がある．テストパターンは人手で準備する場合と，パターン生成部に記述されたコードに従って自動的に生成する場合がある．SystemVerilog などでは，パターン生成規則や制約条件に従って，ランダムにテストパターンを発生するための構文が用意されており，こうしたパターンの生成法は，制約付きランダムパターン生成（Constraint Random Pattern Generation）と呼ばれている．

モニタはシミュレーション実行時に，設計記述の動作が意図したものとなっているかどうかをチェックするモジュールである．複雑な動作をチェックするためのモニタを記述することは一般的に容易ではなく，また異なる設計に使い回すことも難しいため，近年では後述するアサーション記述を利用した手法も普及してきている．

テストパターンをシミュレーションして，設計記述の検証を行う際，検証の進み具合を評価する指標の一つがカバレッジ（Coverage）である．設計記述について可能な動作パターンの数は，設計規模に対して指数的に増大するため，すべての場合を網羅的に尽くすことは一般に難しい．カバレッジによる評価の目的は，網羅性の達成よりはむしろ，設計記述の各部分がどの程度活性化されたかを調べることにある．

カバレッジは大別すると，設計記述の構造に着目した構造カバレッジ（Structural Coverage）（またはコードカバレッジ（Code Coverage））と，機能に着目した機能カバレッジ（Functional Coverage）に分類される．構造カバレッジには，トグルカバレッジ（Toggle Coverage）（各信号線について信号の変化があったか），ステートメントカバレッジ（Statement Coverage）（設計記述の各行が実行されたか），分岐カバレッジ（Branch Coverage）（各分岐条件について成立・不成立の両方が実行されたか），FSM（有限状態機械）カバレッジ（FSM Coverage）（FSM の各状態，各遷移が実行されたか）などがある．機能カバレッジは，論理条件として与えるもので，例えば，「FIFO が一杯である」「キューが一杯の状態になった後，空になる」などである．これらのカバレッジ条件は，具体的には PSL などの言語を使って記述される．設計記述の中に埋め込んでおき，シミュレーション時に評価する．

アサーション検証（Assertion-Based Verification）[3]は，アサーションと呼ばれる論理条件を設計記述に埋め込んでおき，シミュレーション時に違反がないかどうかを調べる検証手法をいう．アサーションを記述する標準言語しては，OVL（Open Verification Library），PSL（Property Specification Language），SVA（SystemVerilog Assertion）などがあり，これらの言語は主として RTL 記述を対象として設計されている．アサーションをチェックするため，シミュレーション時間は増大するが，アサーション記述による観測性改善によって，デバッグ工程の短縮が可能となる．アサーション記述はまた静的検証にも用いることができる．

RTL 記述や論理設計に対するシミュレーションには，大別してイベント駆動（Event-Driven）方式とサイクルベース（Cycle-Based）方式がある．イベント駆動方式では，設計記述中で発生するイベント，あるいは信号値の変化を順に追跡していく．ゲートごとの遅延を考慮した検証や非同期回路のシミュレーションに用いられる．一方，サイクルベース方式では，タイミングに関する詳細は捨象され，1 クロックサイクル中で起こる動作は，論理のみに着目してシミュレーションが実行されるため，イベント駆動方式に比べて高速となる．高位の設計記述については，それぞれの動作意味に基づいてシミュレーションを行う．例えば，C や C++ の記述の場合は，通常のプログラムとして実行され，SystemC，SpecC 記述では，それぞれの動作モデルに基づいて C++のコードを生成して，これを実行する方式がとられている．

シミュレータは通常ソフトウェアとして実装されるが，加速するためにハードウェアを用いる方式もある．ハードウェア・エミュレーション（Hardware Emulation）は，設計記述の一部または全部を論理合成して，FPGA などを用いて実現することによって速度向上を計る方式である．このほか，シミュレーション高速化のために VLIW（Very Long Instruction Word）プロセッサなど特化したハードウェア（ハードウェア・アクセラレータ（Hardware Accelerator））及び専用のコンパイラを用いる方式もある．

(2) 静的検証

静的検証（Static Verification）は，形式検証あるいはフォーマル検証（Formal Verification）とも呼ばれ，設計記述を数学的に解析することにより，設計の正しさを確認する方法である．

網羅性をねらいとしている点でシミュレーションと異なる．一般に必要となる計算量が非常に大きいため，設計記述中の一部分を対象として適用される．テストパターンを必要としない，周囲のモジュールも不必要という特徴があることから，設計の初期段階において，誤りの早期発見のための手段として用いられることもある．

静的検証の手法は等価性判定（Equivalence Checking）[4]とプロパティ検査（Property Checking）（またはモデル検査（Model Checking）[5] に大別される．等価性判定は二つの記述の動作が等価であるかどうかを判定するものである．最も実用化が進んでいるのは，組合せ回路部分に対する等価性判定であり，タイミング修正やスキャン回路の付加の前後に等価性が保たれているかどうかを確かめるために用いられている．近年の SAT（Satisfiability）技術の発展により，実用規模の回路に適用することができるようになっている．より一般的には，異なるレベルで与えられた記述間の等価性判定も考えられる．この場合，二つの記述の違いが大きくなり，入出力やタイミング上の対応関係を見つけなければならないため，より難しくなるが，RTL 記述と C, C++などで書かれた記述との等価性判定などについては，商用ツールも開発されている．

プロパティ検査は，設計記述に対して，チェックしたい論理的な条件を与えて，設計の初期状態から到達可能な状態すべてについて，この条件が成立するかどうかを網羅的に調べることを目標とする．プロパティ検査の基本構成を**図 3･7** に示す．

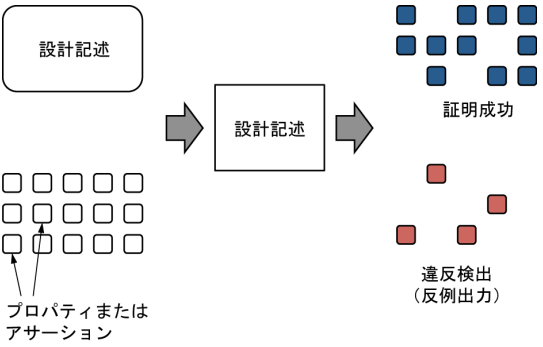

図 3・7　プロパティ検査の基本構成

論理的な条件の記述方法には，時相論理と呼ばれる論理体系やアサーション記述などを用いる方法がある．モデル検査はプロパティ検査とほぼ同義に用いられるが，狭い意味ではライブネス条件（Liveness Condition）（いつか必ず成立するべき条件）まで扱える場合のみを指すこともある．有界モデル検査（Bounded Model Checking）[6] は，初期状態から決められたステップ数まで，網羅的な探索を行うものであり，モデル検査に比べて，取り扱い可能な規模が大きい．

一般に静的検証において探索しなければならない状態空間は，構成要素（フリップフロップなど）の数に対して指数的に大きくなる．これが状態爆発（State Explosion）と呼ばれる現象である．状態爆発に対処するために，状態の集合を圧縮表現して，それを操作するアルゴ

リズムが用いられる．具体的には，二分決定グラフ（Binary Decision Diagram, BDD）の操作アルゴリズムや，ブール式に対する充足可能性判定（Satisifiability, SAT）アルゴリズムなどである．

抽象化（Abstraction）[6] は，静的検証において，多数の状態をまとめることにより状態爆発を抑える手法である．RTL 記述から得られる状態遷移関数を操作して抽象化する方法や，述語（例：$x > 0$）を与え，記述の各点における述語の真偽のみに注目して状態遷移関数を構成し直す方法などがある．通常抽象化は検証結果が保守的になるように行われ，正しいと判定した場合の結果は抽象化以前の記述に対しても正しい．一方，誤り（反例）が発見された場合は，抽象化以前の記述においても，誤りであるかどうかを調べる必要がある．必要な場合は，詳細化を行い，より精度の高いモデルを得た後，検証を繰り返す．抽象化した設計に対して，反例に基づいて詳細化を自動的に行うアプローチは，反例に基づく抽象化・詳細化法（Counter-Example Guided Abstraction and Refinement, CEGAR）と呼ばれている．

3-2-4 設計レベルと検証

（執筆者：浜口清治）[2009 年 2 月 受領]

設計階層を考えると，検証は二つの異なるレベルの記述を対象とする場合と一つのレベルの記述を対象とする場合に分けることができる．異なるレベル間の検証では，通常，上位での設計の動作を，下位の設計が実現できているかどうかを検証することになる．典型例は，C などの高位の記述言語で与えられた，仕様に相当する参照モデル（Reference Model）と RTL 記述で与えられた設計の動作の比較である．

これ以外にもテストベンチの作成を容易にするため，パターン生成器やモニタを，検証対象となる設計よりも抽象度の高い記述で与えることがある．異なるレベルの記述間では，一般に入出力のタイミングやデータの形式が異なるため，何らかの方法で対応をとる必要がある．対応を取るために変換器（ラッパー（Wrapper），トランザクタ（Transactor），バス・ファンクショナル・モデル（Bus Functional Model）とも呼ばれる）を挟みこむ場合もある．**図 3･8** に，複数のレベルにまたがる記述を含むテストベンチの構成例を示す．この例では，レベル間の信号のやりとりをトランザクタが仲介しており，モニタが出力比較を行っている．

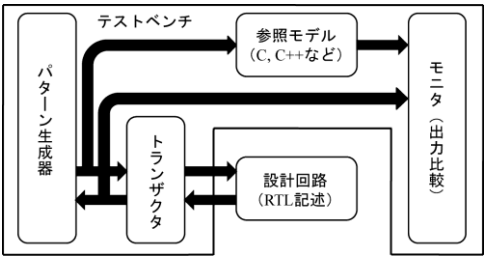

図 3・8　複数のレベルにまたがるテストベンチ

一つのレベルの設計記述に対しては，詳細化された部分に対する検証を行う場合が考えられる．例えば，導入したキューの動作などが正しいかどうかを，アサーションを用いて検証

する場合や，モジュール間の通信のためのインタフェース回路が定められた通信プロトコルに準拠する動作をするかどうかをチェックする場合などがあげられる．また，最適化を行っ前後の二つの設計記述に対する等価性判定も，一つのレベルの設計記述に対する検証である．

■参考文献

1) Wile, B., Goss, J.C., Roesner, W., “Comprehensive Functional Verification, The Complete IndustryCycle,” Morgan Kaufmann, 2005.
2) J. Bergeron, E. Cerny, A. Hunter and A. Nightingale, “Verification Methodology Manual for System Verilog,” Springer, 2005.
3) H. Foster, A. Krolnik and D Lacey, “Assertion-Based Design,” Second Edition, Kluwer, 2004.
4) E.M. Clarke, O. Grumberg and D. Peled, “Model Checking,” MIT Press, 1999.
5) P. Molitor and J. Mohnke, “Equivalence Checking of Digital Circuits, Fundamentals, Principles, Methods,” Kluwer Academic Publishers, 2004.
6) C. Wang, G.D. Hachtel and F. Somenzi, “Abstraction Refinement for Large Scale Model Checking,” Springer 2006.

■10 群 － 1 編 － 3 章

3-3 論理設計

3-3-1 論理表現

(執筆者：湊　真一)[2013 年 8 月 受領]

論理回路を設計する際には，実現しようとする機能（仕様）を何らかの方法で表現し，それに基づいて，効率の良い論理回路を誤りなくつくる必要がある．論理回路は大きく分けて，組合せ回路と順序回路に分けられるが，組合せ回路に対しては，一般には任意の論理式の形で与えられた論理関数を，面積や遅延がなるべく小さくなるように（または所定の制約の範囲内で）実現する論理回路を設計することになる．順序回路の場合は，これに加えて記憶素子をもつので，更に状態遷移を表す論理関数を考える必要がある．いずれの場合でも，論理関数をコンパクトに表現して，論理演算や等価性判定などを高速に行うことは，論理設計を行ううえで基盤となる重要な技術である．

論理関数の入力変数が 4〜5 個までであれば，紙と鉛筆を使って人手で設計することも可能だが，それより少し大きな規模の論理関数になると，手間がかかりすぎるうえに，人間はしばしば誤りを犯す．そこで，計算機上で論理関数を表現し，高速に誤りなく演算処理を行う方法が種々提案され，利用されている．

論理合成や検証を目的とした場合，計算機上での論理関数表現には，(1) 論理関数の表現がコンパクトであること，(2) 任意の論理式からその論理関数表現を高速に生成することができること，(3) 論理関数の表現に一意性があること，または等価性判定や包含性判定等の処理が高速に行えること，(4) 論理関数を充足する（出力が 1 になる）ような入力組合せを高速に求めることができること，などの性質が要求される．

計算機上での論理関数の表現方法としては，現在までに主なものとして

(a) **真理値表**（Truth Table）

(b) **積和形論理式**（Sum-of-Products Form）

(c) **二分決定グラフ**（BDD, Binary Decision Diagram）

が知られている．以下，それぞれの方法について簡単に述べる．

(a)の真理値表は，2^n 通りのすべての入力組合せに対する論理関数の出力を列挙したものである．一つの n 変数論理関数を表すのに，ちょうど 2^n ビットを要する．真理値表による表現は，どんな簡単な論理関数に対しても，指数関数的な記憶量と処理時間を必要とするという欠点がある．逆に，どんな複雑な論理関数でも全く同じ形式で扱えるという特長もある．真理値表は，n が 10 程度であれば十分実用的であり，20〜30 程度までならば，何とか現実的に使用可能である．

(b)の積和形論理式は，正または負のリテラルの組合せからなる積項の和集合により，論理関数を表現する方法である．これは，論理関数の出力を 1 にするための入力変数の条件を列挙する形式とみなすこともでき，人間の直感に合い，比較的コンパクトな表現である．積和形は論理式のかたちに一定の制約を加えて，計算機上で処理しやすくしたデータ構造といえる．積和形論理式は，可変長データであるという点で，真理値表とは大きく異なる．例えば，

$a, b, c, \ldots, z$ の 26 変数の場合を考えると，これらの単純な論理積や論理和を表す論理関数は，真理値表では 2^{26} ビット使用するのに対し，積和形論理式では 26 文字（と演算子）だけで表現できる．逆に，パリティ関数のように，積和形にすると常に指数的なデータ量となる関数もある．積和形論理式は，否定演算が苦手でデータ量が爆発的に増大しやすいこと，論理関数表現としての一意性が保証されない，という欠点ももつ．

(c)の BDD（二分決定グラフ）（Binary Decision Diagram）[1]を用いる方法は，記憶効率や処理速度の面で優れており，1990 年頃から，論理回路設計をはじめとする様々な分野で広く使われるようになっている．BDD は，**図 3･9**(a) に示すような論理関数のグラフによる表現である．これは，それぞれの入力変数に 0, 1 の値を入れて場合分けしたときの論理関数の出力（すなわち，シャノン展開をすべての変数に対して繰り返し適用して得られる入れ子構造）を，図 3･9(b) のように場合分け二分木で表現し，更にこれを縮約することにより得られる．このとき，場合分けする変数の順序を固定し，

・冗長な節点を削除する（**図 3･10**(a)）．

・等価な部分グラフを共有する（図 3･10(b)）．

という処理を可能な限り行うことにより「既約」なかたちが得られ，論理関数をコンパクトかつ一意に表現できることが知られている．多くの場合，単に BDD というときはこのような既約な BDD のことを指す．

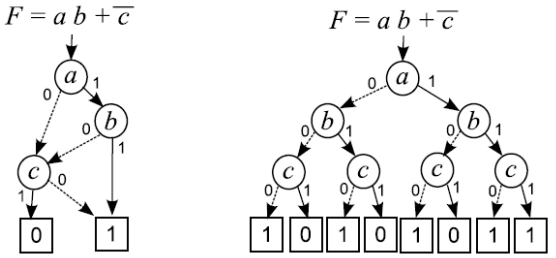

図 3・9　BDD と場合分け二分木

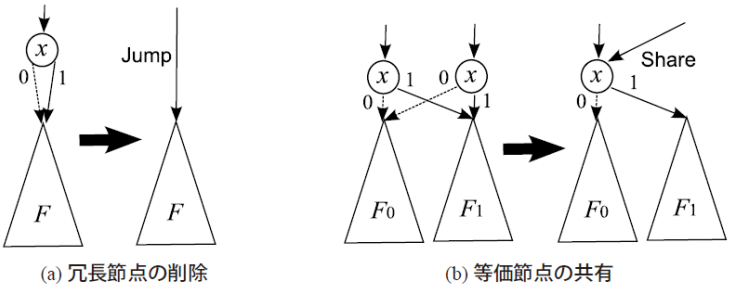

図 3・10　BDD の簡約化規則

BDD は，真理値表や積和形論理式と比較して，多くの優れた性質をもっている．BDD の記憶量は，論理関数の性質に依存し，最悪の場合は入力変数の個数に対して指数関数的なサイズとなるが，人間が設計するような論理式や論理回路で表現された論理関数に対しては，多くの場合，非常にコンパクトに圧縮されたグラフになる．それに加えて BDD は論理関数を一意に表せるという大きな特長がある．**図 3･11** に典型的な論理関数の BDD を示す．このように，*n* 入力 AND, OR, EXOR の論理関数（更に一部の入力や出力に否定がついたもの）は，*n* に比例する節点数で表現可能である．

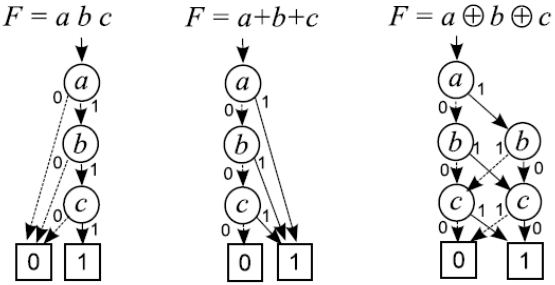

図 3・11　典型的な論理関数の BDD

BDD は，データが計算機の主記憶に納まる限りは，各種の論理演算をその記憶量にほぼ比例する時間で効率よく実行できるという特長をもつ．論理関数の性質にもよるが，数十から数百個もの入力変数を持つ論理関数を，汎用 PC の主記憶容量（1GB 程度）の範囲で現実的に表現し，種々の論理演算を実行することができる．

以上，論理関数の表現方法について簡単に述べたが，より詳細については，1 群 8 編 1 章 3 節「論理関数の計算機上での表現」において，参考文献も含めて詳しく解説している．

■参考文献

1) R.E. Bryant, “Graph-based algorithms for boolean function manipulation,” IEEE Transactions on Computers, vol.C-35, no.8, pp.677–691, 1986.

3-3-2 論理合成

（執筆者：松永裕介）［2013 年 8 月 受領］

広義の意味での論理合成とは，ハードウェア記述言語から回路のネットリストを自動的に生成する全般的な処理を指し，高位レベル合成を含むこともある．しかし，通常は有限状態機械（Finite State Machine: FSM）や論理関数を入力として，論理回路を生成する処理のことを指すことが多い．このような論理設計の自動化に関する研究は比較的古くから行われており，’50 年代や’60 年代に今日の研究の基礎となる論文は発表されていた．しかし，当時の計算機の能力や記憶容量などに適したものではなかったことや，論理設計以後の設計行程が自動化されていなかったことなどから実用化は行われていなかった．その後，PLA（Programmable Logic Array）を用いた設計が行われるようになると，積和形論理式の簡単化プログラムという形で論理合成が広く実用で使われはじめる．その後，ゲートアレイやスタ

ンダードセルを用いた設計手法が主流になるにつれて設計対象は多段論理回路に移り，'90 年代のはじめに今日の論理合成技術が確立した．

前述のとおり，論理合成の入力としては状態遷移表などの順序回路（有限状態機械）の記述や積和形論理式などの組合せ回路の記述がある．通常，有限状態機械は，組合せ回路とフリップ・フロップなどの記憶素子を用いて実現される．一般に出力値や次の時刻における状態は入力と現在の状態の関数として与えられるので，組合せ回路部分がその計算を行い，記憶素子が状態を保持する*．

このように順序回路に対する合成処理も最終的には組合せ回路の合成処理に変換される．続いて組合せ回路合成では，回路の実現方式によって異なった処理が行われる．PLA を用いた実現の場合，積和形論理式がそのまま PLA 回路の設計結果となるので積和形論理式の簡単化処理が行われる．この処理は二段論理回路の合成処理とも呼ばれる．二段論理回路の設計に関しては 1 群 8 編 2 章 2-2 節を参照のこと†．

ゲートアレイやスタンダードセルを用いた実現の場合，まず，使用するセルライブラリのテクノロジに独立な多段論理回路（Muliti-Level Logic Circuit）‡を合成する処理が行われ，その後でテクノロジライブラリに依存した回路をつくり出す処理－テクノロジマッピング（Technology Mapping）－が行われる．このような戦略は，多くの実用的なシステムが用いているもので，各々の処理の目的関数を単純なものとすることによって解法の見通しを立てやすくしている．

多段論理回路は二段論理回路に比べて小さな面積で少ない消費電力で構成することができ，また，多くの場合，高速でもあるが，反面，その構成の自由度の大きさから厳密最適解を求めることはほとんど不可能である．そこで，主にヒューリスティックアルゴリズムが用いられている．多段論理回路合成で行われる処理は様々であるが，いくつかの尺度で分類することができる．

・回路全体の構造を変更するもの（Global Operation）

・回路の部分的な構造を変更するもの（Local Operation）

・論理式に対する簡単化処理（Algebraic Method）

・論理関数に対する簡単化処理（Boolean Method）

このうち，回路全体の構造を変更するものや論理関数に対する簡単化処理は計算時間や使用メモリ量の点から，大規模回路に対する堅牢性（スケーラビリティ）がないと考えられるために実用的なシステムではあまり用いられていない．多段論理合成の詳細に関しては 1 群 8 編 2 章 3 節を参照のこと．

---

* 有限状態機械は，出力の値が状態と入力の両方に依存する Mealy 型と状態のみに依存する Moore 型に分類することができるが，論理合成に関する限りこれらはほとんど区別なく扱われる．

† PLA は一段目が AND, 二段目が OR という二段の回路で実現されるので二段論理回路とも呼ばれる．

‡二段論理回路に対する対語で多段論理回路と呼ばれる．実際にはゲート段数が 2 段以下の回路も含む．

3-3-3 テクノロジマッピング

(執筆者 : 松永裕介)[2013 年 8 月 受領]

テクノロジマッピング（Technology Mapping）は論理合成の最後の段階であり，与えられた論理機能を実現する回路のネットリストを自動的に生成する．VLSI 回路の実現方法は多岐にわたり，その各々について専用の回路設計手法を考えなければならないわけであるが，ここでは ASIC（特定用途向け専用 LSI）で広く用いられているセルベース設計用のテクノロジマッピングについて述べる．

セルベース設計とは，あらかじめ NAND ゲートや INVERTER などの基本セルのセット（セルライブラリと呼ぶ）を用意しておいて，それらのセルを組み合わせて回路をつくるものである．MOS トランジスタはソースとドレイン間の電流の向きに制約がないため双方向であり，MOS トランジスタで構成された回路の論理的な動作を解析することはそれほど容易ではない．これに対し，セルライブラリのほとんどのセルは入力，及び出力の区別がついた端子をもつ．また，一つの出力端子に対して，制限数以下であれば自由に入力端子を接続することができ，その挙動は簡単な論理式で表現できる．このようにセルベース設計は論理設計で考慮すべき探索空間を大幅に狭め，自動で回路生成を行うテクノロジマッピングを実用可能にしたと言ってよい．なお，セルベース設計方式のなかには，セル内部のトランジスタ構成を共通にして，配線のみを変えることで様々なセルをつくり出すゲートアレイ方式と，セルごとに異なる内部構造を持つスタンダードセル方式があるが，テクノロジマッピングにおいてはほとんど区別なく扱われる．

論理合成が実用になりはじめた当初（'80 中頃），ルールベースを用いたテクノロジマッピング手法が提案された[1, 2, 3]．これは様々なテクノロジルールを考慮した場合，システマティックなアルゴリズムでは解けないだろうという見込みがあり，ルールベースにしておけば，ルールの追加／変更いかんでどうにでもできるという判断があったわけだが，その後，DAGON[4]という純粋なアルゴリズムベースのテクノロジマッピング手法が提案され，世の中の大部分のテクノロジマッパーはこのようなアルゴリズムベースの手法に切り替わっていった．今となっては，ルールベース手法には，テクノロジが新しくなる度にルールの見直し，追加を行わなければならないとか，最適解を求めることができない，無駄な探索を行う，などの欠点ばかりが目立つのでほとんど用いられていない[§]．

(1) テクノロジマッピングの入力

実現したい回路の仕様もしくは初期回路としてはブーリアンネットワークが用いられる．ブーリアンネットワークはテクノロジ非依存な論理合成処理で用いられているデータ構造で，多段論理回路の構造を表す．ブーリアンネットワークの各々の節点は任意の論理関数をもてるので，その節点の一つひとつが，セルライブラリ中のセルに対応づけられるとは限らない．逆に言えば，その節点の一つひとつがセルに対応するようなブーリアンネットワークをつくり出すことがテクノロジマッピングの処理そのものとみなすことができる．このように，ブーリアンネットワークの構造をどのように決めるかはテクノロジマッピングの重要なポイントとなっている．そこで，文献 4, 5)などで用いられている手法では，ブーリアンネットワー

§ アルゴリズム的に解くことが難しい演算器や多出力セルのマッピングなどにはルールベースが用いられている．

クの各節点を 2 入力 NAND ゲートに分解したものを初期入力として用いている[**]．任意の論理関数は AND-OR のかたちで表現可能であり，任意の AND ゲート及び OR ゲートは 2 入力 NAND ゲートとインバータに分解可能なので，任意の論理関数は 2 入力ゲートに分解可能である．このように 2 入力ゲートに分解されたネットワークをサブジェクトグラフ（Subject Graph）と呼ぶ．一つのブーリアンネットワークに対して分解の異なるサブジェクトグラフが考えられ，また，その分解の仕方によって得られる最終的な解も異なるが，マッピング前にどのような分解が良いのかを知る術がなく，また，すべての分解を試すことも困難なので，適当なサブジェクトグラフが用いられる．

セルの論理式もサブジェクトグラフと同様に 2 入力 NAND ゲートに分解される．これはパタングラフと呼ばれ，サブジェクトグラフへのマッピングに用いられる．パタングラフの場合にはサブジェクトグラフと異なり，すべての分解が列挙される．

(2) DAG Covering と Tree Covering

回路を表すサブジェクトグラフとセルを表すパタングラフを用いてテクノロジマッピングの問題を定式化してみると，サブジェクトグラフのすべての節点を被覆するようなパタングラフの集合を求める問題（すなわち被覆問題）とみなすことができる．ただし，通常の被覆問題よりも難しいのは，ただ単に被覆すれば良いのではなく，パタングラフの入力（すなわちセルの入力）にはほかのパタングラフの出力（もしくは外部入力）が接続していなければならないという条件が付加されていることである．この問題は NP 困難に属することが知られており，数千～数万ゲート規模の回路のマッピング問題を厳密に解くことは現実的には不可能である．

そこで考え出されたのが，サブジェクトグラフをそのファンアウト数が 1 の部分グラフ（グラフ理論でいうところの Tree：木）に分解し，その各々の部分グラフに対して最適なマッチの組合せを求める，という手法である [4]．対象が DAG から Tree に変わると動的計画法によってグラフのサイズに対して線形時間で最適解を求めるアルゴリズムが存在する．これは，ある節点 v を根とする部分木に対する最適解は，その部分木に含まれる節点（を根とする部分木）に対する最適解から計算できることによる．つまり，入力の節点から始めて自分を根とする部分木に対する最適解を計算してゆくことで，最終的に与えられた木全体の最適解が求められるというものである．

この Tree Covering はその木に対するマッピングとしては常に最適解を生成し，実用的にも高速なアルゴリズムであるが，回路全体を tree 状に分割することによって，いくつかの最適性は損なわれてしまうが，現実的な時間で解を求めるためにこのようにサブジェクトグラフを木状に分割してから Tree Covering を行う手法が広く用いられている[††]．

図 3･12 に面積最小の解を求める Tree Covering アルゴリズムを示す．サブジェクトグラフの節点 v に対応した変数 C_vを用意し，その節点を根とする部分グラフを被覆する最小コストを表すものとする．外部入力に対応する節点 i のコスト C_i は 0 としておく．

[**] 4)では 3 入力 NAND，4 入力 NAND なども用いている．

[††] 実際には分割せずに DAG に対して以下の Tree Covering を強引に適用している場合もある．もちろん，解の最適性の保証はなくなる．

```
tree_covering(Graph G) {
    G の各節点を入力から出力方向へトポロジカル順で整列
    for G の各節点 v に対して {
        C_v ← +∞;
        for v を根 (出力) とするマッチ m に対して {
            c ← m のセル面積;
            for m の各入力に対応する節点 u に対して
                c ← c + C_u;
            C_v ← min(C_v, c);
        }
    }
}
```

図 3・12　Tree Covering のアルゴリズム

実際には，各節点は最小コストの他にそのコストを実現するマッチも保持しているので，上のアルゴリズムが終了した時点で最小被覆が得られることになる．

テクノロジマッピングにおいては面積のほかに遅延時間や消費電力を目的関数にする場合や制約条件にする場合もあり，それらもこの Tree Covering アルゴリズムの応用で解くことが可能である．

■参考文献

1) J. Darringer, D. Brand, J. Gerbi, W. Joyner, and L. Trevillyan, “Lss: a system for production logic synthesis,” IBM J. Res. Develop., vol.28, no.5, pp.537-545, Sep. 1984.

2) D. Gregory, K. Bartlett, A. de Geus, and G. Hactel., “Socrates: a system for automatically synthesizing and optimizing combinational logic,” In Proceedings of the 23rd Design Automation Conference, pp.79-85, Jun. 1986.

3) J. Ishikawa, H. Sato, M. Hiramine, K. Ishida, S. Oruri, Y. Kazuma, and S. Murai, “A rule based reorgani-zation system lores/ex,” In Proceedings of the International Conference on Computer Design (ICCD-88), pp.252-266, Oct. 1988.

4) K. Keutzer, “Dagon: technology binding and local optimization by dag matching,” In Proceedings of the 24th Design Automation Conference, pp.341-347, Jun. 1987.

5) R. Rudell, “Logic synthesis for VLSI design,” Ph.D. thesis, University of California, Berkeley, 1989.

3-3-4 論理・タイミング検証

（執筆者：浜口清治）［2013 年 8 月 受領］

論理回路の論理のみを考慮した検証については，本章 3-2-3 項の機能検証の項目で触れているためここでは述べない．本節では，タイミング検証について解説する．

タイミング検証では，ゲート素子に対する遅延モデルを定める必要がある．遅延モデルには，信号の伝搬がある決まった時間だけ遅延する純粋遅延モデルのほか，立上り・立下り遅延モデル（0→1 と 1→0 のそれぞれの信号変化で遅延値が異なる），最大・最小遅延モデル（遅延値の最大値と最小値が与えられる）などがある．遅延値は通常ゲート素子ごとに定められ，素子の種類や出力数によって異なる．レイアウトを行ったあと，その情報を用いて遅延値を

与える場合もある．タイミング・シミュレーションでは，これらの遅延モデルのもとで回路の振舞いがシミュレートされることになる．複雑なモデルほど扱いは難しく，例えば上記の最大・最小遅延モデルでは，入力変化に対して複数の出力変化があり得るため，必要な計算量も大きくなる．

論理回路レベルで最もよく用いられているタイミング検証は，静的タイミング解析（Static Timing Analysis）である．この解析では，シミュレーションを行うのではなく，回路の構造のみから，組合せ回路部分の出力が安定するまでの時間を計算する．遅延モデルにより細部は異なるが，基本的には，外部入力から順に，各ゲートの各入力への最大（最小）遅延時間の最大値（最小値）を求め，各ゲートに対応する遅延時間を加えることによって，そのゲートに対する最大（最小）遅延時間を計算する．この操作を外部出力まで繰り返して，最大（または最小）遅延時間を求める．

同期式順序回路を考えた場合，最大遅延時間が大きい場合は，クロック周期を大きくする必要があり，また，最小遅延時間が小さすぎる場合は，フリップ・フロップのホールド時間分の信号値の安定が確保できなくなる．この意味で最大・最小遅延時間評価の正確さが重要である．最大・最小遅延時間を与えるパスが論理的に活性化できないフォールス・パス（False Path）となっていることがある．あるパスがフォールス・パスであるかどうかは，テスト生成に用いられるアルゴリズムに類似した手法により判定することができる．この技術はタイミング最適化において用いられている．

プロセスの微細化に伴い，チップ内での遅延のばらつきが大きくなりつつあり，これに対処するため，統計的タイミング解析（Statisitical Static Timing Analysis）の研究が進んでいる [12, 13]．統計的タイミング解析では，各素子の遅延を確率変数によってモデル化する．パス一つひとつについてタイミング解析を行う手法は，一般にパスの数が回路規模に対して指数的となるため，多大な計算量を要する．静的タイミング解析と同様に，回路構造を利用して計算する場合，遅延に正規分布を仮定し，更に互いに相関がないとすると，遅延の和や最大値の確率分布の計算は容易になるが，相関を仮定した場合に比べ，回路の構造に依存して悲観的な結果も楽観的な結果も得られることが知られている．

遅延間に相関が発生する原因には，組合せ回路内の再収斂構造やチップ内の位置に依存した遅延への影響などがある．相関を考慮すると，和や最大値の確率分布を計算することは難しく，遅延値を離散化した上で近似的に上限と下限を求める手法なども開発されている． 更には，正規分布でない分布や，ある遅延値がほかの遅延値の非線形関数になっていたりする場合を取り扱うための手法，フォールス・パスを考慮した手法なども研究されている．

■参考文献

1) M. Fujita, I. Ghosh and M. Prasad, “Verification Techniques for System-Level Design,” Morgan Kaufmann, 2008.
2) 藤田昌宏（編著），“システム LSI 設計工学，” オーム社，2006.
3) G. De Micheli and M. Sami (Ed.), “Hardware/Software Co-Design,” Kluwer Academic Publishers, 1996.
4) J. Staunstrup and W. Wolf (Ed.), “Hardware/Software Co-Design: Principles and Practice,” Kluwer Academic Publishers, 1997.
5) G. De Micheli, R. Ernst, and W. Wolf (Ed.), “Readings in Hardware/Software Codesign,” Morgan Kaufmann Publishers, 2001.

6) Wile, B., Goss, J.C., Roesner, W., “Comprehensive Functional Verification, The Complete Industry Cycle,” Morgan Kaufmann, 2005.

7) J. Bergeron, E. Cerny, A. Hunter and A. Nightingale, “Verification Methodology Manual for System Verilog,” Springer, 2005.

8) H. Foster, A. Krolnik and D Lacey, “Assertion-Based Design,” Second Edition, Kluwer, 2004.

9) E.M. Clarke, O. Grumberg and D. Peled, “Model Checking,” MIT Press, 1999.

10) P. Molitor and J. Mohnke, “Equivalence Checking of Digital Circuits, Fundamentals, Principles, Methods,” Kluwer Academic Publishers, 2004.

11) M. Ganai and A. Gupta, “Sat-based Scalable Formal Verification Solutions,” Springer-Verlag, 2007.

12) 築山修治, “統計的タイミング解析:概論, ”第 18 回回路とシステム軽井沢ワークショップ, pp. 533-538, April 2005.

13) D. Blaauw, K. Chopra, A. Srivastava and L. Scheffer, “Statistical Timing Analysis: From Basic Principles to State of the Art,” IEEE Trans. on Computer-Aided Design of Integrated circuits and Systems, vol.27, no.4, April 2008.

■10 群 － 1 編 － 3 章

3-4 回路設計

（執筆者：佐藤高史）［2013 年 8 月 受領］

3-4-1 回路モデル

集積回路設計における典型的な性能の指標は，遅延時間（クロック周波数）と電力である．標準論理セルを用いる自動設計ツールにおいて，論理セルの遅延や電力は，様々な使用条件に対し事前に計算されており，ライブラリとして保持される．このライブラリは，様々な設計ツールが遅延時間の低減や低電力化等の回路性能向上のために参照する．

(1) 遅延モデル

同期式回路では，正しい論理のもとにタイミング制約を守ることが，設計の第一目標である．このため，回路合成，レイアウト設計，など様々な設計段階で回路遅延が見積られる．人手による計算時や，精度よりも計算速度が要求される場合には，Elmore 遅延 [1]が広く用いられる．いま，節点 i におけるステップ応答を求めるために，その微分であるインパルス応答 $h_i(t)$を考えて，次式を満たす（インパルス応答の中央値）を遅延時間とする．

$$\int_0^{\tau} h_i(t)dt = \frac{1}{2} \qquad (3・1)$$

ここで遅延時間τを，$h(t)$の平均値（1 次モーメント m_1）で近似する．$h_i(t)$を非負の単峰な分布関数と考えれば，その平均値である Elmore 遅延 T_dは次式で近似される．

$$T_d = m_1 = \int_0^{\infty} t h_i(t)dt \qquad (3・2)$$

抵抗と容量からなるツリー状の回路ネットワーク（RC 木）に対し，本手法は次の計算に帰着され，高速かつ容易に計算できる．ソース s から節点 i への遅延 T_{di}は次式で表される．

$$T_{d_i} = \sum_k C_k R_{ik} \qquad (3・3)$$

ここに R_{ik}は，s から i に至る経路と s から k に至る経路の共通部分にある抵抗である．

$$R_{ik} = \sum R_j, \quad j \in [\mathrm{path}(s \to i) \cap \mathrm{path}(s \to k)] \qquad (3・4)$$

Elmore 遅延は RC 木の遅延時間の上限を与えることが知られている．すなわち，常に悲観的な見積りとなる．より精度の高い計算が必要な場合，及びインダクタやグラウンドに接続されていない容量素子を含む場合等には AWE（Asymptotic Waveform Evaluation）法 [3]や PRIMA[4]などに代表されるモーメントマッチング法が使用されている．これらの方法では，状態変数形式での回路方程式をもとに周波数領域における近似的な応答波形を求める．いま，応答波形$v(t)$の周波数領域での応答波形を $V(s)$及びその近似解$\hat{V}(s)$とする．$V(s)$を $s = 0$ のまわりで級数展開（Mclaurin 展開）するときの s の係数をモーメントと呼ぶ．これをラプラス逆変換しやすい関数形$\hat{V}(s)$に n 次項まで一致させることを n 次のモーメントマッチング法と呼ぶ．$v(t)$の近似解$\hat{v}(t)$は，$\hat{V}(s)$のラプラス逆変換により求める．AWE 法ではモーメントマッチングを係数比較により陽（Explicit）に行うのに対して，PRIMA では Kryrov 部分空間に基づいて陰（Implicit）に行う．

自動設計ツールにおいては，遅延ライブラリを参照する遅延計算が一般的である．遅延ライブラリは通常，標準セルへの入力信号の遷移時間と標準セルの負荷容量をインデックスとする 2 次元の表として実現されている．長距離配線など，集中定数としての負荷容量で直接表現されない負荷を駆動する場合には，配線抵抗を考慮した等価的な負荷容量に換算することで，表参照に帰着させて遅延計算を行う．その際には，単純な負荷容量を充放電するための電流の積分値と，実際に駆動すべき寄生素子を充放電するための電流積分値が一致するような計算を行うことで，遅延計算をより正確にする [2]．配線遅延は，前述したモーメントマッチング法により計算する．

(2) 電力モデル

回路の消費電力は，動的な電力と静的な電力に大別できる．更に動的な電力は，負荷容量の充放電電力と貫通電力に分類できる．動的な電力 P_d は，回路中の容量 C の充放電にともない消費される．

$$P_d = \alpha C V_{dd}^{\ 2} f \qquad (3・5)$$

ここに V_{dd} は電源電圧，f はクロック周波数，α は回路の実効的な活性化率である．実効的な活性化率とは，論理値の反転が生じて負荷容量を充電または放電する節点の割合を負荷容量で重み付けしたものである．回路の論理反転は，通常，論理シミュレーションにより求める．

貫通電流は，論理回路の反転にともない生ずる電流のうち，容量の充放電に寄与せずに電源－グラウンド間を直接流れる成分である．CMOS 論理回路では，PMOS トランジスタと NMOS トランジスタが相補的に動作し，電源 - グラウンド間の直流電流経路をもたない特徴があるが，論理遷移時には，PMOS，NMOS 両トランジスタが同時に導通するタイミングが存在し，これが貫通電流の原因となる．

静的な電力は漏れ電流（リーク電流）により，論理動作の有無にかかわらず常に消費される電力である．漏れ電流は，主としてサブスレショルド電流，ゲート電流，接合電流に分類される [5]．サブスレショルド電流とは，MOS トランジスタが弱反転領域にあるときにソース・ドレイン間を流れる電流である．ゲート電流は，微細プロセスにおける MOS トランジスタにおいて，絶縁膜である薄いゲート酸化膜を通じて流れるトンネル電流である．接合電流は，基板とソース・ドレインの接合部における結晶欠陥により発生するリーク電流である．

自動設計環境においては，電力についても，ライブラリを参照することで計算できる．

3-4-2 高速伝送技術

高速な信号を伝送する配線については，チップ内においても伝送線路型の配線が用いられる場合がある．例えばクロックツリーにおける配線では，上層配線層を用いかつ太幅に設計するとともに，配線の両脇にグラウンド配線を配置するコプレナ（Coplanar）構造がしばしばとられる．

長距離の伝送を小さい遅延時間で行うことを目的とする場合，太幅の配線を用いるほかに，バッファ（リピータ）の駆動力を大きくすること，リピータを適切に挿入して配線長を制御する方法がある．これは，Elmore 遅延によれば，配線遅延時間は配線長の 2 乗に比例するため，リピータによる配線長の分割は遅延低減に対する効果が大きいこと，信号の立上り時間・

立下り時間を急峻に保ちやすいこと，などによる．回路ブロック間の伝送など，通常の信号における長距離伝送については，リピータ挿入による配線長の分割が広く行われている．

3-4-3 回路シミュレーション

回路シミュレーションとは，素子の接続関係から電気・電子回路の振舞いを計算機上で模擬し予測する計算のことである．このため，電気回路の基本方程式を自動生成し，数値計算によりこれを解く機能をもつ．回路シミュレーションにより，回路を実際に製造することなくその動作を知ることができる．狭義にはアナログ回路のシミュレーションのための回路シミュレーションを指すが，より広義には論理回路の動作を模擬する論理シミュレーションなどを含む．回路シミュレーションでは，節点電圧や素子電流がアナログ値で得られるのに対し，論理シミュレーションで得られる出力応答は，一般に論理値（H, L, 高インピーダンス(Hi-Z), 不定値(X)）である．また，回路シミュレーションでは，直流動作点解析，小信号周波数応答解析，過渡解析等が可能であるが，論理シミュレーションでは，過渡解析のみが可能である．

回路シミュレーションのための基本的なアルゴリズムは，1970 年代に確立されている．なかでも，カリフォルニア大バークレイ校により開発された SPICE（Simulation Program with Integrated Circuit Emphasis）[6] は，ソースコードが公開されたため世界中で広く使用されている．SPICE では，修正節点法（Modified Nodal Analysys: MNA）と呼ばれる定式化手法により，回路方程式を作成する．修正節点法は，節点電圧を変数として回路方程式を立て解く節点解析法（Nodal Analysis: NA）を，電流変数を扱えるよう拡張した手法である．

3-4-4 ミックスドシグナルシミュレーション

ミックスドシグナルシミュレーションとは，異なる抽象度のシミュレータを同時に連携させて動作させることにより，大規模な回路の解析を高速に行う解析方式である．ディジタル・アナログ混載回路を効率良くシミュレーションするために，高い精度が必要な（アナログ）回路部分を回路シミュレータで，ディジタル回路部分を論理シミュレータで解析するシミュレータがその典型的な例である．デジタルブロックを含む大規模なアナログ回路に関しては，更に，Verilog-A などのアナログ・ビヘイビア記述により回路ブロックの動作を高い抽象度で記述して同時解析を行うことも行われている．

■参考文献

1) W.C. Elmore, “The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers,” Journal of Applied Physics, vol.19, no.1, pp.55-63, 1948.

2) J. Qian, S. Pullela and L. Pillage, “Modeling the “Effective Capacitance” for the RC Interconnect of CMOS Gates,” IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol.13, no.12, pp.1526-1535, 1994.

3) L.T. Pillage and R.A. Rohrer, “Asymptotic waveform evaluation for timing analysis,” IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol.9, no.4, pp.352-366, 1990.

4) A. Odabasioglu, M. Celik, and L.T. Pileggi, “PRIMA: passive reduced-order interconnect macromodeling algorithm,” in Proc. international conference on Computer-aided design, pp.58-65, 1997.

5) K. Roy, S. Muknopadhyay, and H. Mahmoodi-Meimand, “Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits,” Proc. IEEE, vol.91, no.2, pp.305-327, Feb. 2003.

6) Nagel, “SPICE2, A computer program to simulate semiconductor circuit,” Memorandum UCB/ERL M520, Univ. California, Berkeley, May 1975.

■10 群 － 1 編 － 3 章

3-5 レイアウト設計

3-5-1 回路分割

(執筆者 : 澁谷利行)[2013 年 8 月 受領]

回路分割とは，回路，もしくはシステムを複数のより小さな部分回路に分割することである．記述されている回路の抽象度により，回路の機能や素子が分割される単位になる．また，システムを LSI や FPGA などのデバイスに分割する場合と，LSI 内の回路を分割して，階層化設計やフロアプラン設計する場合とでは，分割される部分回路の規模も異なる．

システムレベルの回路を複数デバイスに分割する場合には，各デバイスに割り当てる回路規模と外部端子の制限を満たすことが重要となる．その際に，回路遅延やノイズなどの制約条件を守ることも要求される．

LSI 内の回路を分割する場合には，互いにネット接続が疎である部分回路を生成することが目的になる．フロアプラン（3-5-2 節）により分割された部分回路を適切に指定することにより，レイアウトツールの処理時間の短縮や，配線性の向上が可能となる．

(1) 階層化設計

LSI 内の回路を階層的に分割し，その階層に従って設計を進めていく手法を階層化設計という．

プロセッサ設計のように先端プロセスのもつ特性の限界を目指す設計の場合には，人手の介在が不可欠である．人が扱いやすいように回路機能単位で分割が行われ，分割された回路はフロアプラン（3-5-2 節）された後，フロアプランの領域単位で独立に並行して回路最適化やレイアウト作業を進めることが可能となる．

一方，製品の出荷時期や設計コストを重視する場合には，自動化ツールを積極的に導入して，設計期間を短縮する必要がある．自動化ツールの最適化能力や解析能力に適した回路規模に分割し，適用することにより，自動化ツールの能力を最大限に引き出すことを狙う．

階層化設計の問題点としては，回路分割の品質が最終的に実現された回路の面積や遅延などの品質に大きく影響するにもかかわらず，設計の初期段階で回路分割をしなくてはならない点である．また，一度回路を分割してしまい，設計を進めてしまうと，分割された回路間での素子の移動や再分割を行うことは難しい場合が多く，設計仕様の変更などへの対応の柔軟性も失われる．

(2) 分割手法

回路分割の問題では，回路の機能や素子などの分割される対象をグラフのノードとして考え，その素子間の接続関係をエッジと考えて表現するのが一般的である．分割の対象となるノードのすべてを N 個のノードの集合に分けることを N 分割と呼び，カット数と呼ばれる分割されたノードの集合間を結ぶエッジの本数の最小化が分割の目的となる．更に，素子の面積に相当するノードの集合のサイズや，配線に相当するエッジに重みを付けることにより，デバイスやフロアプランなどの物理的な制約や，遅延やノイズなどの回路動作に関する制約条件を考慮することが可能になる．

2 分割の代表的な繰り返し最適化手法としては，KL 法や FM 法がある．KL 法では，あるノードに注目し，そのノードの属さない分割集合の全てのノードと一対一で交換を仮に行い，そのなかで最もカット数の減るノードのペアの交換を実際に行い，次に注目するノードを選び，同様の処理を繰り返して，カット数を最小化する．FM 法はノードペアの交換の代わりに注目したノード単体が自分が属さない分割集合へ移動する前提で，減少するカット数を効率的に計算することにより，KL 法よりも高速な処理を実現している．

一方，繰返しではなく，一回の計算で 2 分割されたノードの集合を求める手法としては，グラフの固有値と固有ベクトルを利用する方法や，グラフの最大フローを求めて，最小カットとなっている分割部分（カットライン）を求める方法もある．これらの手法は，数学的な理論を背景に最適な解を求めることが可能ではあるが，物理的，回路動作に関する制約条件を柔軟に考慮することが難しく，前記の繰り返し最適化手法との組合せで分割問題を解くのが現実的とされている．

■参考文献

1) T. Lengauer, “Combinatorial Algorithms for Integrated Circuit Layout,” John Wiley & Sons, Inc, 1990.

3-5-2 フロアプラン・配置

(1) 配置表現

（執筆者：藤吉邦洋）［2013 年 8 月 受領］

配置とは，与えられたブロックやモジュールについてその形状と置く座標を決定したものである．これに対してフロアプランとは，大まかな配置，もしくは，これを決めることであり，具体的には，部分回路の実現に必要な領域の見積りを行い，各部分回路を実現するために必要な領域をチップ内に割り当てる．

配置の表現は，各モジュールの絶対座標によるという古くから用いられている方法と，モジュールどうしの相対位置関係による方法に大別される．後者には様々な方法が提案されているが，スライス構造だけに限った表現としては，二分木を用いた表現 [1)]と，この二分木を逆ポーランド記法の文字列にした表現 [2)]がある．他方，スライス構造に限定しない一般構造の配置の表現方法も幾つか提案されており，主なものでは，Sequence-Pair [3)]，BSG [4)]，O-Tree[6)] がある．ほかに，チップ全体を表す方形を垂直もしくは水平分割線にて分割して方形の領域に切り分け，各々の領域にモジュールを割り当てるという方法もある．これの表現方法としては CBL[5)]，Q-Sequence[7)] が提案されている．

(2) 配置手法

（執筆者：藤吉邦洋）［2013 年 8 月 受領］

配置手法には，旧来はチップ面積の使用率や配線長の最小化だけが求められたが，動作速度の高速化や，近年は熱分布の平均化なども求められる．またアナログ集積回路では，指定されたセル対について対称に配置したり，近接して配置することなどが求められることがある．

よい配置を求めるために様々な手法が研究されてきているが，Min-Cut 手法は，作り込む回路を相互接続の少ない二つに切り分け，そのサイズの比に応じてチップの領域も二つの小領域に切り分け，それぞれの小領域に切り分けられた回路を割り当てるという操作を再帰的に行っていき配置を求める．

力学的手法（Force-Directed Method）では，間にある配線要求に応じた「引力」を各モジュール対について仮想し，チップ外周上の端子からも引力を及ぼすことにより，力学的につり合った位置を求める．そして詳細配置と呼ばれる後処理によって，モジュールどうしの重なりをなくす．これにより，配線要求が沢山あるモジュール対は近くに置かれた配置を得ることができる．

また，配置の表現方法に基づいて焼きなまし法（シミュレーテッドアニーリング，Simulated Annealing）などの探索アルゴリズムにて評価の良い配置を求めるという手法も，多く研究されている．この手法では，様々な配置を発生しては評価するという操作を探索手法の下で多数回繰り返す．

(3) タイミング駆動配置手法

（執筆者：若林真一）[2013 年 8 月 受領]

近年の LSI チップは非常に高い周波数で動作するものも多く，そうしたチップのレイアウト設計においては，与えられたタイミング制約を満たすことが重要である．タイミング制約を考慮した配置手法はタイミング駆動配置（Timing-Driven Placement）手法と呼ばれ，多くの手法が提案されている [7]．与えられるタイミング制約としては，各ネットに対する許容最大（最小）遅延，及び前段のフリップ・フロップ（FF）の出力端子から次段の FF の入力端子までの信号経路の許容最大（最小）遅延である．これらの遅延制約は配置手法においては直接，制約として，あるいは制約違反に対するペナルティ関数を目的関数に組み込むことでセル配置時に考慮することが一般的である．

■参考文献（※藤吉先生 箇所）

1) R.H.J.M. Otten, “Automatic floorplan design,” in Proc. 19th ACM/IEEE DAC, pp.261–267, 1982.

2) D.F. Wang and C.L. Liu, “A new algorithm for floorplan design,” in Proc. 15th ACM/IEEE DAC, pp.101–107, 1986.

3) H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, “VLSI Module Placement Based on Rectangle-Packing by the Sequence-Pair,” IEEE Trans. CAD, vol.15, no.12, pp.1518-1524, 1996.

4) S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, “Module Packing Based on the BSG-Structure and IC Layout Applications,” IEEE Trans. on CAD, vol.17, no.6, pp.519–530, 1998.

5) X. Hong, S. Dong, Y. Ma, Y. Cai, C.-K. Cheng, and J. Gu, “Corner Block List: An efficient topological representation of Non-Slicing floorplan,” in Proc. IEEE/ACM ICCAD, pp.8–12, 2000.

6) P.-N. Guo, T. Takahashi, C.-K. Cheng, T. Yoshimura: “Floorplanning using a tree representation,” IEEE Trans.CAD, vol.20, pp.281–289, 2001.

7) K. Sakanushi, Y. Kajitani and D.P. Mehta, “he quarter-state sequence floorplan representation,” IEEE Trans. CAS-I, vol.50, no.3, pp.376–386, 2003.

3-5-3 配　線

（執筆者：若林真一）[2013 年 8 月 受領]

(1) 概略配線・詳細配線

セル配置が決まると，次に与えられたネットリストに基づいて各ネットの配線経路を決定する．この工程を配線設計という．配線経路の決定においても，フロアプラン・配置設計と同じく，配線経路の概略を決定する概略配線と，概略配線の結果に基づいてデザインルール

を満たした詳細な配線経路を決定する詳細配線の 2 段階に分けることができる．

(2) 配線手法

LSI のネットリストにおいて，通常の信号線に対する配線は，2 次元領域を配線領域としネットの端子は領域中の任意の位置に与えられるエリア配線と，横長の長方形領域を配線領域としネットの端子は配線領域の周囲に位置し，配線領域の上辺と下辺の端子位置は固定されており，左辺と右辺の端子位置は任意とするチャネル配線に大まかに分類される．ほかの種類の配線としてクロック信号を分配するためのクロック配線と，電源供給のための電源配線があるが，これらについては後述する．

エリア配線の主な手法としては迷路法と線分探索法が知られている．迷路法は幅優先探索法の一種であり，配線領域を格子に分割し，ネットのいずれかの端子を始点として始点の周囲から順に 1, 2, 3,…とラベル付けしていき，終点に到達すればラベルを逆にたどることで配線経路を決定する手法である（**図 3･13**）[8]．迷路法は，配線経路があれば必ず最短経路を見つけることができるという特長をもつ．迷路法にはいくつもの改良が知られており，例えば終点に近い方向に優先的に探索を進めることで計算時間を短縮する手法がある．このような手法は A*アルゴリズムと呼ばれる．

一方，線分探索法は，始点と終点から縦横の線分を障害物にぶつかるまで伸ばし，障害物にぶつかったら障害物を回り込むように直角に別の線分を伸ばしていく（**図 3･14**）．この操作を繰り返していき，始点と終点から順次発生させた線分どうしが交わったら，線分を逆にたどることで配線経路を見つける手法である [6]．線分探索法は迷路法と比較して効率よく配線経路を見付けることができるが，最短経路を見つける保証はない．

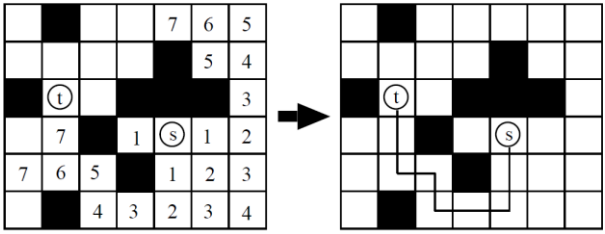

図 3･13 迷路法

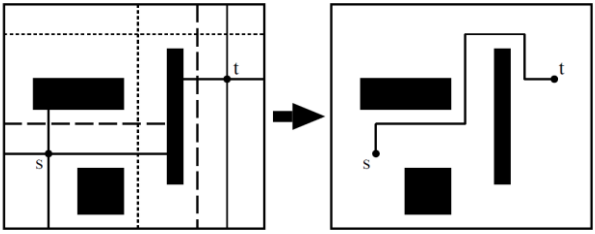

図 3･14 線分探索法

複数のネットを対象とするエリア配線においては，ネットの配線順序も重要である．一般にはネットの予測配線長の短いもの，あるいはタイミング制約の厳しいものから配線するが，途中で配線できなくなった場合は，障害物と判定された配線済みのネットを一旦引きはがして現在のネットの配線を行い，次に引きはがされた配線をやり直す引きはがし再配線手法もよく使われる [5].

次にチャネル配線について説明する．チャネル配線領域は一般に水平配線線分を割り当てることのできるトラックと，縦方向の配線線分を通すことのできるコラムで構成される（**図 3･15**）．与えられたネットリストに対し，ネットの端子位置に従って一番上のトラックから順に，左詰めにネットの水平配線線分を割り当てていくことで配線を行うレフトエッジアルゴリズムが知られている．コラムでの垂直配線線分の重なりが生じなければ，レフトエッジアルゴリズムは配線長が最小となる最適なチャネル配線を出力する．一方，コラムで配線経路が重なる場合は，水平配線線分を 2 本以上のトラックに割り当てたり，あるいは配線を迂回させる必要があり，そのような機能をもつチャネル配線アルゴリズムが知られている．

チャネル配線の拡張として，チャネル配線領域の上下に接するセル列上も配線領域として使用できると仮定したセル上チャネル配線アルゴリズムも知られている [4]．また，チャネル配線の特別な場合として，チャネル配線領域の上下左右の辺上のすべての端子位置が固定されたスイッチボックス配線がある．スイッチボックス配線についても専用のアルゴリズムが提案されている [3].

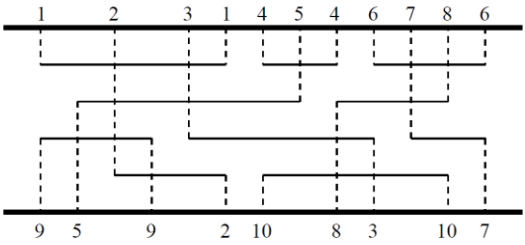

図 3･15　チャネル配線

(3)　タイミング駆動配線手法

一般にエリア配線においては長大な配線経路が生ずる場合がある．一方，高性能 LSI においては，ネットのタイミング制約が厳しいため，通常の配線では信号遅延の増加によりタイミング制約を満たさなくなる場合が生ずる．この問題に対処するための配線手法としてタイミング駆動配線手法が知られている [10].

タイミング駆動配線手法では，配線幅調整，バッファ挿入，分岐点の変更，の 3 通りの手法を利用してタイミング制約を満たす配線経路を求める（**図 3･16**）．一般に幅広配線のほうが信号遅延は小さくなるので，タイミング制約の厳しいネットの配線経路の一部に対しては配線幅を太くする．更に，配線経路の適当な位置にバッファを挿入することでネットの駆動ゲートから見た配線容量を減らし，信号遅延を短くする．また，複数のシンク端子がある場合，タイミング制約の特に厳しいシンク端子に対しては，ソース端子からすぐに分岐して，専用の配線経路を確保することで信号遅延を減らすことができる．

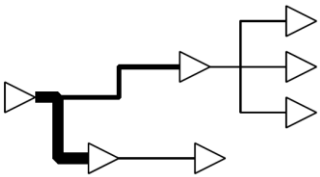

図 3・16　タイミング駆動配線

(4)　製造容易化設計（Design for Manufacturability）

微細加工技術の進んだ今日の LSI においては，製造時の欠陥の発生を防ぐレイアウト設計が求められる．このような設計を一般に製造容易化設計（Design For Manufacturability）という．配線設計における主な DFM としてはアンテナ効果対策配線と銅配線プロセス対応 CMP ダミーパターン生成がある．

アンテナ効果とはプラズマを用いた製造工程において，配線セグメントに蓄積された電荷が放電し，ゲート酸化膜にダメージを与える現象のことをいう．この現象を軽減するため，配線設計においては, MOS トランジスタのゲートに直接接続する配線セグメントはできるだけ短くし，シリコン基板から離れた上の配線層の配線につなぐことで，ダメージを少なくする [11].

次に，近年，一般的になってきた銅配線プロセスにおいては，製造時の CMP（Chemical Mechanical Polishing，化学的機械研磨）において，シリコン基板を均一に削るため，銅配線の密度の下限制約が与えられる．このため，配線結果が制約を下回る場合は，ダミーの配線経路を生成することで密度制約を満たすようにする．ダミーの配線経路の挿入は隣接する配線の遅延等に影響を与えるので，CMP を考慮し，均一な配線密度になるように配線を行うことでダミー配線の挿入を削減するエリア配線手法も提案されている [2].

(5)　クロック配線

LSI で実現されるディジタル回路は同期式順序回路であり，回路中のすべてのフリップフロップ（FF）にクロックを供給する必要がある．クロックに対する配線（クロック配線）は通常のネットに対する配線と異なり，クロックの供給元（ソース）から FF のクロック端子までの信号遅延は問題とはならず，FF 間でのクロックの到達時刻の差（クロックスキュー）が問題となる．理想的にはスキューが 0 となるクロック配線が望ましい．このようなクロック配線を実現する方法として，H 木によるクロック配線とクロックメッシュ（トランク方式）によるクロック配線がある．

H 木とはチップ領域を階層的に 4 分割し，それぞれの領域にクロック端子をもつように構成したクロック木である（**図 3･17**(1)）．このようなクロック木を求める手法として，距離が近い FF どうしを対にしながら信号遅延の評価を行い，信号遅延が均等になるようにボトムアップにクロック木を求める手法（Deferred Merge Embedding 法）が知られている [1]．一方，クロックメッシュとはチップ領域を覆う格子状の配線経路である（図 3･17(2)）[12]．縦横の配線（トランク）の幅と間隔は RC 回路モデルを用いた回路シミュレーションによるクロック

スキューの評価で決定される．各垂直トランク配線はトランクバッファ回路（図 3･17(3)）により駆動される．一般に，トランク方式のクロック配線はクロック木方式のクロック配線よりスキューが小さくなる一方，消費電力はより大きい．

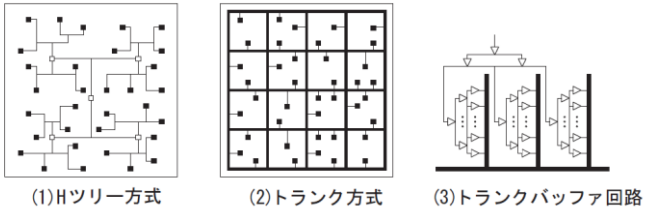

図 3・17　クロック配線

(6)　電源配線

電源配線とは，セル（モジュール）の電源端子とチップ領域の周囲に配置された電源セルを配線する工程である．通常，チップ領域を囲むように電源リングを設け，電源リングと電源セルを接続する（**図 3･18**）．電源リングの上辺と下辺は，チップ領域内においては電源ストラップで接続される．セルベース LSI の場合，セルの電源端子はセルの左辺と右辺の対称の位置にあるので，セルを横方向に並べると横方向に電源供給配線が形成されるので，その配線に対して電源リングと電源ストラップから電源を供給する．また，大きな形状をもつマクロセル（モジュール）に対しては，モジュールの周囲を電源のリング配線で囲み，そのリング配線に対してチップの電源リングと電源ストラップから電源を供給する．

電源配線幅の決定においては，配線抵抗による電圧降下（IR ドロップ）が許容値以下であること，および各配線セグメントに流れる電流を計算し，エレクトロマイグレーションが発生しない電流密度であること，を条件とし，更に余裕（マージン）を見込んで実際の配線幅を決定する [9]．

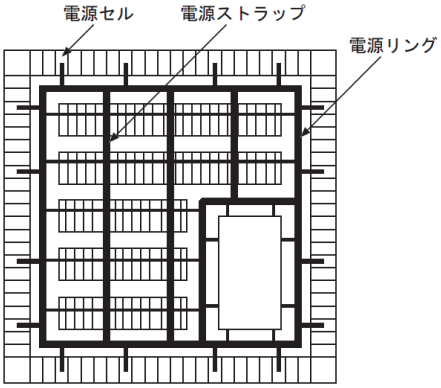

図 3・18　電源配線

■参考文献（若林先生 箇所）

1) T.-H. Chao, Y.-C. Hsu and J.-M. Ho, “Zero skew clock net routing,” Proc. Design Automation Conference, pp.518-523, 1992.

2) H.-Y. Chen, S.-J. Chou, S.-L. Wang and Y.-W. Chang, “A novel wire-density-driven full-chip routing system for CMP variation control,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.28, no.2, pp.193-206, 2009.

3) J.P. Cohoon and P.L. Heck, “Beaver: A computational geometry based tool for switchbox routing,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.7, no.6, pp.684–697, 1988.

4) J. Cong and C.L. Liu, “Over-the-cell channel routing,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.9, no.4, pp.408-418, 1990.

5) W.A. Dees,Jr. and P.G. Karger, “Automated rip-up and reroute techniques,” Proc. Design Automation Conference, pp.432-439, 1892.

6) D.W. Hightower, “A solution to line-routing problems on the continuous plane,” Proc. Design Automation Conference, pp.1-24, 1969.

7) M.A.B. Jackson and E.S. Kuh, “Performance-driven placement of cell based IC’s,” Proc. Design Automation Conference, pp.370-375, 1989.

8) C.Y. Lee, “An algorithm for path connections and its applications,” IRE Transactions on Electronic Computers, vol.EC-10, no.2, pp.364-365, 1961.

9) T. Mitsuhashi and E.S. Kuh, “Power and ground network topology optimization for cell based VLSIs,” Proc. Design Automation Conference, pp.524-529, 1992.

10) T. Okamoto and J. Cong, “Bu ered Steiner tree construction with wire sizing for interconnect layout optimization,” Proc. International Conference on Computer-Aided Design, pp.44-49, 1996.

11) 城田博史，定兼利行，寺井正幸，高橋一浩，柴谷　聡，小谷　健，岡崎　芳，“「アンテナ効果」を低減する ASIC 設計向けの配線手法，”情報処理学会論文誌，vol.40, no.4, pp.1626-1634, 1999.

12) 寺井正幸，金本俊幾，小谷　健，柴山泰範，岡崎　芳，堀場康孝，岩出秀平，“大規模高速 ASIC 用クロック分配回路レイアウト設計ツールの開発，”情報処理学会論文誌，vol.43, no.5, pp.1294-1303, 2002.

3-5-4 レイアウト検証

（執筆者：澁谷利行）[2013 年 8 月 受領]

(1)　Layout Versus Schematic (LVS) Check

LVS では，マスクパターンとネットリストを入力とし，その両者において同じ素子間の接続が実現できているかを検証する．マスクパターンの情報からは，トランジスタ領域と等電位の接続関係を抽出する．また，ネットリストもトランジスタレベルの接続情報に展開され，両者を比較することにより，検証を行う．

(2)　Design Rule Check (DRC)

DRC では，入力されたマスクパターンが設計ルールを違反していないかを検証する．マスクパターンの情報は，図形データに変換され，図形演算処理により，最小幅，最小間隔などのテクノロジごとに設定された設計ルールの条件に合うかのチェックを行う．テクノロジの微細化にともない，設計ルールはテクノロジが進むごとに複雑化してきており，処理時間の増加が問題視されており，分散処理などによる高速化が実現されている．

■参考文献

1) L. Lavagno, L. Scheffer, and G. Martin, “EDA for IC Implementation, Circuit Design, and ProcessTechnology,” CRC Press, 2006.

■10 群 － 1 編 － 3 章

3-6 マスク設計

（とりまとめ：広瀬文保）[2013 年 8 月 拝受]

3-6-1 図形分割

（執筆者：安倍慈久仁）[2013 年 8 月 拝受]

図形分割とは，Fracturing とも呼ばれ，OPC（Optical Proximity Correction）などの RET（Resolution Enhancement Technology）処理が行われた最終のマスクパターンの図形を，E-Beam（Electron Beam）装置にてマスク描画を行うために，それぞれの装置のフォーマットに合わせ分割することである． E-Beam 描画装置は一般的に，VSB（Variable Shaped Beam）と言う方式をとり，すべての図形を長方形と二等辺三角形（斜めは 45 度）の組合せで表現するので，それに合わせて分割，再構成する必要がある．どのように分割するかにより，描画効率，図形数，ファイルサイズなどが変化する．

3-6-2 マスク生成

(1) Optical Proximity Correction (OPC)

（執筆者：安倍慈久仁）[2013 年 8 月 拝受]

OPC とは，光の近接効果（Optical Proximity）により，光学像がマスクパターンを再現できなくなる現象を予測し，マスクパターンを補正する技術である．露光における解像度は以下の式で表される．λは露光に使われる波長．ArF 光源（波長 193 nm）が先端（90 nm, 65 nm, 45 nm ノード）の量産に用いられている．NA（Numerical Aperture）は，レンズの開口数であり，露光装置で決まる．ArF 光源の場合の実現可能な NA の最大値はドライで 0.9 程度，液浸（水）で 1.35 である．k1 はプロセス定数であり，プロセス条件，露光光学系で決まる．OPC などの RET 技術により，この k1 を小さくすることで，解像度を増し，露光波長より小さな図形を解像させる．

$$\text{解像度} = k1^{*}\lambda / NA \qquad (3\cdot 6)$$

最小図形が 1/2λ 程度までは，ルールベース OPC が主流であったが，1/3λ 以下においてはルール生成が困難で補正が不十分となり，露光シミュレーションを用いたモデルベース OPC が用いられている．

(2) Phase Shift Mask (PSM)

（執筆者：安倍慈久仁）[2013 年 8 月 拝受]

PSM（位相シフトマスク）とは，解像の際のコントラストを向上するための技術で，マスクを通過する光の位相と透過率を制御して逆位相の光振幅の足し算を行わせる．主なものとして Alternative PSM，Attenuated PSM がある．Alternative PSM は，発明者の名前にちなんでレベンソンマスクとも呼ばれる．これは，マスク開口部の石英を削る，またはマスク上に適当な屈折率の位相シフターを形成することで，マスク通過後の光の位相を反転させる．Attenuated PSM は，ハーフトーンマスクともいわれる．これは，マスク遮光部に位相シフターを形成し，屈折率，透過率を制御する．透過率は 6%と 8%が多い．

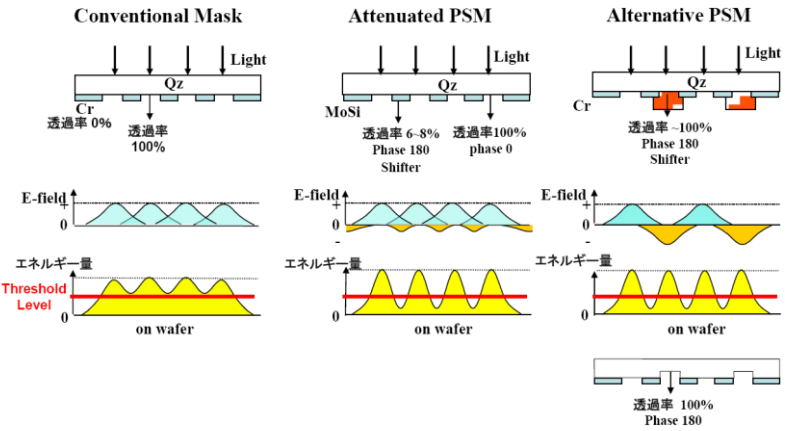

図 3・19 PSM

(3) Dummy Metal Fill

（執筆者：市川仁子）［2013 年 8 月 拝受］

Dummy Metal fill とは，配線パターンの密度の偏りを緩和する目的で，配線周辺に配置されるダミーパターンのことである．CMP（Chemical Mechanical Polishing）工程の平坦性と，エッチング工程でのエッチング特性が配線パターンの粗密に大きく影響されるため，Metal Fill による緩和が必要となる．ダミーパターンの発生は，ルールベースがメインであったが，65 nm ノードより最適なルール作成が困難となった．そこで，CMP シミュレーションによるルールの検証の重要性が増すとともに，CMP モデルを開発し，ルールを用いることなくモデルベースにより Metal Fill を生成する技術が実用されるようになった．なお，ダミーパターンにより配線間の寄生容量が増加するため，RC 抽出し，これを考慮することが必要になる．

3-6-3 パラメータ抽出

(1) デバイスシミュレーション

（執筆者：奥秋勝己）［2013 年 8 月 拝受］

デバイスシミュレーションとは，コンピュータ上でデバイスの動作を模擬させることを表す．各種製造工程を経て得られるドーピング密度プロファイルを正確に反映したうえでデバイスシミュレーションし，結果として，電気特性，光学特性，温度特性といったデバイスの各種特性を，DC・AC・過渡解析の観点から分析できるように 2 次元，3 次元に表示する．デバイスシミュレーションを導入することで，実際に試作製造した場合に必要となる膨大なコストと時間を削減することが可能となる．近年の微細化に伴うナノメータ・オーダーの物質の構造，機能に対し，量子論に基づいた解析ができるシミュレーション技術が求められている．

(2) 配線寄生素子

（執筆者：奥秋勝己）［2013 年 8 月 拝受］

導体層（配線）の引き回しによって寄生的に生ずる抵抗，容量，インダクタンスを総称して，配線寄生素子と呼んでいる．配線とそのまわりの電磁場の相互作用を含めた配線寄生素

子の正確な見積り方法の一つとして，その電磁場におけるマクスウェルの方程式から導出される偏微分方程式の解を，有限要素法（Finite Element Method；FEM）で求める方法がある．これを EDA ツールでは一般的に，Field Solver と呼んでいる．しかし，Filed Solver での解析に膨大な計算時間がかかることより，より高速に配線寄生素子の個々の値を求める手法として，パターンマッチング手法が普及している．これはプロセスの断面構造と，配線とその隣接パターンまで含む領域の配線寄生素子に対してあらかじめモデル化したテクノロジライブラリを作成しておき，実際のレイアウトパターンで配線寄生素子を抽出する段階では，実際のレイアウトパターンをテクノロジライブラリのモデルに対してパターンマッチングするもので，最も近いモデルの値で近似することにより，より高速に配線寄生素子の個々の値（抵抗，容量，インダクタンス）を求めることが可能となっている．

(3) 製造ばらつき

（執筆者：安倍慈久仁）［2013 年 8 月 拝受］

集積回路を構成する各部品の特性がばらついて製造されることを製造ばらつきという．製造ばらつきは，ランダムばらつきと，システマティックばらつきに分類される．ランダムばらつきは，統計的にしか扱えないばらつきのことで，その要因としては，製造工程における不純物濃度分布などの不均一性（不純物の個数がポアソン分布に従う）や，パーティクルによるパターン不良などがある．ランダムばらつきについては，TEG（Test Element Group）を作成し，統計データを取ることにより，統計モデルを作成し，シミュレーションでばらつきの影響を統計的に予測する．一方，システマティックばらつきは，何らかの規則性が見出されるもので，その要因としては，リソグラフィ工程における近接効果，CMP によるメタル厚さの変化などがある．システマティックばらつきについては，TEG を作製しデータを取ることでモデルを作成し，モデルを用いたシミュレーションにより値（回路パラメータや，Yield など）の予測が可能となる．いかにして誤差の少ない良いモデルをつくるかがポイントとなる．なお，本来はシステマティックばらつきだが，実用的な現実解として統計的モデルを用いる物も存在する．別の分類の仕方として，チップ内で起こるローカルばらつき（OCV：On Chip Variation）と，チップ間で起こるグローバルばらつきがある．グローバルばらつきは，更に Wafer 間，ロット間などに分類する場合もある．ばらつきに対応するために，製造技術の改善によるばらつきの減少，モデルによる予測を用いての DFM（Design For Manufacturability）設計，新たな材料やデバイス構造の採用などのアプローチが取られている．

■10 群 － 1 編 － 3 章

3-7 テスト

3-7-1 故障モデル

（執筆者：高橋　寛）[2008 年 4 月 受領]

製造不良によって誤動作が生じた集積回路における物理的な原因は，配線の短絡・断線，コンタクト不良，ビア不良などである．これらの物理的な原因を欠陥（Defect）と呼ぶ．欠陥の箇所では正常回路と異なる振る舞いが生ずる．これを故障（Fault）と呼ぶ．集積回路の外部出力で観測された故障の影響を誤り（Error）と呼ぶ．故障検査のために集積回路において起こり得る故障をモデル化したものを故障モデル（Fault Model）と呼ぶ[1~4]．

(1)　論理故障（Logical Fault）

故障の影響によって回路の論理動作が正常回路における論理動作とは異なるならば，その故障を論理故障と呼ぶ．

縮退故障（Stuck-at Fault）：縮退故障は，信号線と電源線やグランドが短絡する欠陥のモデル化であり，信号線の論理値が 0 または 1 に固定する故障である．

ブリッジ故障（Bridging Fault）：ブリッジ故障は，隣接する信号線が短絡する欠陥のモデル化であり，短絡した 2 本の信号線の論理値が常に同じになる故障である．短絡した 2 本の信号線の論理値が 2 本の信号線の AND（OR）演算の結果となる故障を AND（OR）ブリッジ故障と呼ぶ．また，短絡した信号線の一方の論理値に他方の信号線の論理値が影響を受ける故障をドミナントブリッジ故障と呼ぶ．

オープン故障（Open Fault）：オープン故障は，配線の断線や多層配線におけるビアのオープンをモデル化したものとゲート内のトランジスタが常に開放状態になる欠陥をモデル化したものに分けられる．

(2)　遅延故障（Delay Fault）

遅延故障は，抵抗性の短絡・断線などの欠陥，信号線間のクロストーク，電源ノイズなどによって信号線における信号変化の伝搬遅延時間が増加し，クロック信号に同期してフリップフロップに取り込まれる論理値が正常回路と異なる故障である．遅延故障は，遷移故障（Transition Fault），ゲート遅延故障（Gate Delay Fault），及びパス遅延故障（Path Delay Fault）に分類される．

(3)　メモリの故障（Memory Fault）

規則的に配置されたセルで構成されたメモリの故障モデルは，メモリセルの内容が隣接したメモリセルのパターンの影響を受けて変化するパターン依存故障（Pattern Sensitive Fault）やあるメモリセルにおける信号変化が隣接したメモリセルの内容に影響を与える結合故障（Coupling Fault）である．

(4)　プロセッサの故障（Processor Fault）

レジスタを選択する際に誤って別のレジスタを選択するレジスタ選択機能故障や命令を実行する際に誤って命令を解読し，その結果として別の命令を実行する命令解読機能故障がある．

3-7-2 テスト方式

（執筆者：高橋　寛）[2008 年 4 月 受領]

集積回路に印加した入力パターンに対する出力応答によって故障が存在しているか否かを調べることを故障検出（Fault Detection）という．故障回路に対して，故障の箇所や故障の種類を調べることを故障診断（Fault Diagnosis）という．故障検出と故障診断を総称してテスト（Testing）という．更に故障の物理的な原因を特定することを故障解析（Fault Analysis）という．製造された集積回路に対するテスト方式について整理する[1~4]．

(1)　静的テスト（Static test）と動的テスト（Dynamic test）

被検査回路にテストパターンを印加する速度に関してテスト方式を分類すると，静的テストと動的テストに分類できる．

テストパターンを印加する速度が，本来の動作速度に比べて十分に遅い環境においてテストを行うことを静的テストと呼ぶ．代表的なテスト方式は DC テストである．DC テストでは，集積回路の入出力部の直流的な電気特性を保証する．

一方，本来の動作速度でテストパターンを印加し，テストを行うことを動的テストと呼ぶ．代表的なテスト方式は，AC テスト及び実動作速度テスト（At-Speed Test）である．AC テストでは，集積回路の動的な特性を確認する．At-Speed Test では，本来の動作速度においても集積回路が正しく動作することを保証する．

(2)　機能テスト（Functional Test）と構造テスト（Structural Test）

機能テストでは，設計検証用パターンによって論理回路が正しく動作するか否かを確認する．

構造テストでは，論理回路の構造に着目して，その論理回路が回路図どおりに製造されているか否かを確認する．構造テストに用いるテストパターンは，論理ゲートの接続情報を元に自動的に生成される．

(3)　外部テスト（External Test）と組込み自己テスト（Built-In Self Test）

被検査回路に対してテストパターンを印加する装置及び被検査回路の応答を解析する装置を集積回路の外部に設置するかまたは，内部に組み込むかによってテスト方式を分類すると外部テストと組込み自己テストに分類できる．

多数の集積回路を同時にテストするために，高価な LSI テスタの台数を増やすことは容易ではない．そこで，テスト時間の短縮を図るために，組込み自己テストではテストに必要な回路を集積回路に組み込んでテストを行う．

(4)　電流テスト（Current Test）（IDDQ テスト）

電流テストでは，集積回路の電源電流を測定し，正常回路では流れない電流を調べることで良品／不良品の判定を行う．代表的な方式は，CMOS 回路において静的な電源電流（IDDQ）はほとんど流れない性質を利用した IDDQ テストがある．

■参考文献

1) M. Abramovici, M.A. Breuer, and A.D. Friedman, “Digital Systems Testing and Testable Design,” New Jersey, IEEE Press, 1990.
2) M.L. Bushnell and V.D. Agrawal, “Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits,” Kluwer Academic publishers, 2000.
3) 藤原秀雄，“ディジタルシステムの設計とテスト，”工学図書，2004.
4) 米田友洋，梶原誠司，土屋達弘，“ディペンダブルシステム，”共立出版，2005.

3-7-3 テスト生成

(執筆者：梶原誠司)[2008 年 4 月 受領]

テストパターンとは，回路の故障検査の際に用いる入力パターンのことである．テストパターンの性質やその生成方法については，古くから多くの研究が行われている．本節では，テストパターンの考え方の基礎となるブール微分から，故障シミュレーション，テスト生成のためのアルゴリズムなどについて説明する．

(1) ブール微分

入力変数 $x_1, x_2,\ldots, x_n$ をもつ論理回路の出力関数を $f(x_1, x_2,\ldots, x_n)$ とする．入力変数 x_i に関する f のブール微分（Boolean Difference）は，

$$\frac{df}{dx_i} = f(x_1, x_2,\ldots, x_{i-1},0, x_{i+1}, \ldots, x_n) \oplus f(x_1, x_2,\ldots, x_{i-1},1, x_{i+1}, \ldots, x_n) \qquad (3・7)$$

で定義される．ここで，$f(x_1, x_2, \ldots, x_{i-1},0, x_{i+1}, \ldots, x_n) = fx_i(0)$ とすると，$fx_i(0)$ は x_i が 0 縮退故障を起こしているときの回路の出力関数（故障関数）となっている．x_i の 0 縮退故障を検出するテストパターンは，正常な回路の出力関数と故障関数により $f(x_1, x_2,\ldots, x_i,\ldots, x_n) \oplus fx_i(0) = 1$ を満たす入力パターンとして表されるが，この式の左辺の関数を故障差関数という．

故障差関数は $f(x_1, x_2,\ldots, x_i,\ldots, x_n) \oplus fx_i(0) = x_i \dfrac{df}{dx_i}$ となるため，x_i の 0 縮退故障を検出するテストパターンは，ブール微分を使って $x_i \dfrac{df}{dx_i} = 1$ と表現できる．

(2) 等価故障解析

故障 a の故障関数を $f_a(x_1, x_2, \ldots, x_n)$，故障 b の故障関数を $f_b(x_1, x_2, \ldots, x_n)$ とする．$f_a(x_1, x_2, \ldots, x_n) = f_b(x_1, x_2, \ldots, x_n)$ のとき，故障 a と故障 b を等価故障（Equivalent Fault）という．等価故障は，故障差関数も同じになるので，これら二つの故障に対するテストパターンも同じものになる．例えば，NOT ゲートの入力の 0 縮退故障とそのゲート 1 縮退故障は，等価故障である．一般に，テスト生成では，回路内に生ずるすべての故障を対象にしなくても，等価故障を考えることで，テスト生成の対象となる故障数を減らすことができる．等価故障を求める処理を等価故障解析といい，等価故障のうちテスト生成の対象にする故障を代表故障という．

(3) テストの品質

テスト品質はテストにおける故障を検出する能力として考えられる．生成したテストパターンが仮定した故障モデルの故障をどれだけ検出できるかの評価は，故障検出率（Fault Coverage）や故障検出効率（Fault Efficiency）として定義され，テスト品質を表す一つの指標となっている．

$$\text{故障検出率〔\%〕} = \frac{\text{検出故障数}}{\text{総仮定故障数}} \times 100 \qquad (3・8)$$

故障検出率は，単純にそのテストパターンで検出できる故障の割合を示す．故障の中には，どのような入力パターンに対しても故障差関数が 0 となる場合があり，それらは検出不能故障となる．特に組合せ回路における縮退故障の検出不能故障は，故障のある信号線が回路に

とって論理的に冗長であることを意味するため，冗長故障（Redundant Fault）と呼ばれる．検出不能故障があれば故障検出率の上限が低下するが，それによりテストパターンの評価が悲観的になることを防ぐため，検出不能故障と判定された故障も考慮に入れた故障検出効率でテストパターンを評価することもある．故障検出効率には以下に示す 2 通りの計算式があり，検出不能と判定された故障があれば，互いに異なる値となる．

$$\text{① 故障検出効率〔\%〕} = \frac{\text{検出故障数}}{\text{総仮定故障数} - \text{検出不能故障数}} \times 100 \qquad (3 \cdot 9)$$

$$\text{② 故障検出効率〔\%〕} = \frac{\text{検出故障数} + \text{検出不能故障数}}{\text{総仮定故障数}} \times 100 \qquad (3 \cdot 10)$$

検出不能故障を仮定故障から除いた①の式は，テストすべき故障のうち，そのテストパターンで検出できる故障の割合を表すため，テスト品質評価に適している．一方で，検出不能故障の判定は NP 完全問題であり，必ずしもすべての検出不能故障が実用的な時間内に判定できるとは限らない．検出不能と判定できた故障は検出できた故障と同等と考えれば，②の式のように故障検出効率を計算できる．この式は，検出可能性を判定できた故障の割合を表しており，テスト生成アルゴリズム［本章 3-7-3(5)節，3-7-3(6)節］の性能評価によく用いられる．

(4) 故障シミュレーション

故障シミュレーションとは，故障回路の動作をシミュレートするもので，一般には，与えられたテストパターンにより検出される故障を計算する処理のことである．その目的は，与えられたテストパターンの故障検出率の計算や，テスト生成，故障診断における故障回路の動作解析など様々である．故障シミュレーションは，単純に実行すると計算時間が回路規模，故障数，テストパターン数にほぼ比例して増加するため，高速化が重要である．代表的な故障シミュレーションの手法として，並列故障シミュレーション，演繹故障シミュレーション，同時故障シミュレーションなどが，知られている．

(5) 組合せ回路のテスト生成アルゴリズム

テスト生成は，回路のネットリストと仮定する故障が与えられたときに，故障差関数が 1 となるような入力パターンを求める，またはその故障が検出不能故障であることを証明する処理である．論理回路のテスト生成問題は NP 完全問題であることが知られており，処理の効率化のため多くの研究がなされている．特に組合せ回路に対するテスト生成は，スキャン設計の普及により有用性が高まり，様々な手法の開発が進んでいる．テスト生成の処理には，回路構造に基づく手法と，故障差関数から直接論理関数処理により求める手法がある．回路構造に基づく手法としては，経路活性化法．D アルゴリズム，PODEM, FAN, SOCRATES などのアルゴリズムが開発され，実用化されている．また，故障差関数を SAT（充足可能性問題）として解く手法や二分決定グラフ（BDD）として表現する論理関数処理による手法もある．

(6) 順序回路のテスト生成アルゴリズム

スキャン設計を用いない順序回路に対するテスト生成は，回路の各時刻における動作を展開し，組合せ回路部を直列に接続した回路で組合せ回路的に行う手法がよく用いられる．テ

スト生成の処理としては,故障シミュレーションをベースにテストパターンを生成する手法,現状態から時間を前方向に進めながら故障を活性化・伝搬する入力を求める手法,故障伝搬経路を確定後に時間をさかのぼって信号値を決定する手法に大別される.しかし,回路を何時刻分展開すれば十分かを正確に求めることは難しく,また,セット・リセット機能がない回路では,正常回路と故障回路のどちらも初期化して内部状態を確定するテストパターンを求める必要がある.更に,同規模の組合せ回路と比較して検出不能故障の割合も多くなるため,テスト生成処理に占める検出不能故障の判定に費やされる時間も大きくなる.これらのことから,スキャン設計を用いない大規模順序回路に対しては,高い故障検出率のテストパターンを実用的な時間内で生成する方法はいまだ開発されておらず,大規模順序回路のテスト生成が実用化する見込みは極めて低いと考えられている.

(7) 遅延故障のテスト生成

遅延故障(Delay Fault)は,同期式回路において,回路の信号伝搬時間が増大するような故障で,連続する二つのテストパターンを印加して検出する.これを 2 パターンテストという.1 パターン目で回路に信号値変化前の状態を設定し,2 パターン目は信号値が変化したことをテストする.遅延故障のモデルとしては,遷移故障とパス遅延故障がよく用いられる.遷移故障は,ある信号線の遅延が増大するような故障であり,2 パターンテストの 2 パターン目は縮退故障のテストパターンと同じように考えることができる.パス遅延故障は,フリップフロップからフリップフロップまでの信号値伝搬経路上の遅延が累積して大きくなるような故障で,テスト対象となるパスの活性化条件をどのように決めるかによりロバスト,ノンロバストなどのクラスがあり,故障検出の確実性やテスト可能性が変化する.

2 パターンテストの場合にもスキャン設計によるテスト容易化は不可欠である.このとき,2 パターン目の設定にスキャン動作を用いると,1 パターン目に設定した初期状態が保たれない.この問題を避けるため,スキャン設計された回路の 2 パターンテストには,ラウンチオフキャプチャ(Launch-Off-Capture またはブロードサイド, Broadside)とラウンチオフシフト(Launch-Off-Shift またはスキュードロード, Skewed-Load)の二つのテスト方式が提案されている.遅延テストのテスト生成は,テスト方式に応じて,回路の扱い方を変更する必要がある.

(8) 高位テスト生成

論理回路のテスト生成は NP 完全問題であり,回路規模が大きくなれば回路の構成要素であるゲート数や信号線数が増加し,テスト生成の計算量は増大する.高位テスト生成は,RTL など論理回路より高位のレベルで記述された回路を用いるため,扱う素子数は少なくなり,回路規模の増加によるテスト生成の難しさを緩和できる.故障モデルも機能故障のような論理回路レベルのテストとは異なる故障モデルを用いることができ,特にプロセッサやコントローラ等のテストで有効である.一方で,テスト品質の観点から,回路レベル・物理レベルで表されるような故障が高位テスト生成によるテストパターンでどの程度検出できるかが問題となる.現状ではまだ技術的に発展途上であるが,LSI の大規模化・システム化に伴い高位テスト生成の重要性は増しており,今後研究が進むものと期待される.

■参考文献

1) 米田友洋,梶原誠司,土屋達弘,“ディペンダブルシステム,”共立出版,2005.

2) 藤田昌弘（編著），“IT TEXT システム LSI 設計工学，”オーム社，pp. 174-200, Oct. 20, 2006.
3) M. Abramovici, M.A. Breuer, and A.D. Friedman, “Digital Systems Testing and Testable Design, Piscataway,” New Jersey: IEEE Press, 1990.
4) M.L. Bushnell, V.D. Agrawal, “Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits,” Kluwer Academic Publishers, 2000.
5) L.-T. Wang, C.W.Wu, and X. Wen, “VLSI Test Principles and Architectures,” Morgan Kaufmann Publishers, 2006.

3-7-4 テスト容易化設計

（執筆者：細川利典・井上智生）［2008 年 12 月 受領］

(1) テスタビリティ解析

テスタビリティ（可検査性, Testability）とは，どれだけ容易にテスト生成を行うことができるかということであり，テスト容易性ともいう．与えられた回路のテスタビリティを評価し，回路のどの部分のテスタビリティが悪いかなどを解析することをテスタビリティ解析（Testability Analysis）という．テスタビリティを解析するために，それを数値化したものをテスタビリティ尺度といい，回路中のある信号線に所望の値を設定するために外部入力の値を制御する計算量を可制御性，ある信号線の値を外部出力で観測する計算量を可観測性という．テストポイント（Test Point）とは，回路中の信号線の可観測性や可制御性を向上させることを目的として挿入する論理回路であり，主に組込み自己テストを中心とした擬似ランダムパターンテストで用いられてきた．近年，テストデータ量やテスト実行時間の削減を目的としたテストポイント挿入法が提案されている [4].

(2) スキャン設計

スキャン設計とは，被テスト回路の機能や構造と独立して，フリップフロップに外部から直接値を制御でき，かつフリップフロップの値を外部で直接観測できるように，通常の動作モードのほかに制御信号によりフリップフロップを直列のシフトレジスタとして動作させるテスト容易化設計の一つである．この外部から直接制御・観測可能なシフトレジスタ状の経路をスキャンチェーンと呼ぶ．テスト容易化設計とは，テスト生成が容易な論理回路を最初から積極的に設計する，あるいは変更することである．スキャン設計は順序回路のテスト容易化設計の代表的な手法であり，レジスタ転送レベルあるいは，ゲートレベルで適用されることが多い．特にレジスタ転送レベルで適用されたスキャン設計をレジスタ転送レベルスキャン設計と呼ぶ．スキャン設計にはスキャンパス（Scan Path），LSSD（Level Sensitive Scan Design），RAS（Random Access Scan）などがある．スキャン設計では，シフトレジスタを擬似的な外部入出力とみなす．回路中のすべてのフリップフロップをシフトレジスタとして動作させるテスト容易化設計を完全スキャン設計（Full Scan Design）といい，そのテスト生成の問題は組合せ回路の問題として取り扱うことができる．スキャン設計のテスト順序は，(i) シフトモードでスキャンチェーンを通して，フリップフロップの値を設定する，(ii) 通常モードで外部入力に値を設定し，外部出力での値を期待値と比較するとともに，フリップフロップの入力値を取り込む，(iii) シフトモードでスキャンチェーンを通してフリップフロップの値を読み出し，期待値と比較する．完全スキャン設計は，ハードウェアオーバーヘッドの増加，回路の動作速度の低下などの弊害を伴う．この弊害を解消するために，必要なフリップフロップだけをスキャン設計する部分スキャン設計（Partial Scan Design）も提案されている [5]．また今日の大規模化した回路では，フリップフロップ数が膨大であるので，フリップ

フロップへの値の設定や読出しに必要な時間を削減するために，回路全体のスキャンチェーンを複数のスキャンチェーンに分割する多重スキャン設計(Multiple Scan Design, Parallel Scan Design)が一般的である. 更に, ボードのテスト方式として, 境界スキャン方式(Boundary Scan)が提案されている．境界スキャン方式とは，ボードに搭載されているチップの入出力をスキャン設計するテスト容易化設計であり，IEEE の標準規格になっている（IEEE1149.1)．

(3) 非スキャン設計

スキャン設計を用いないテスト容易化設計を非スキャン方式によるテスト容易化設計という．非スキャン方式によるテスト容易化設計は，論理回路に対してテスト容易化設計を行うものと，論理回路が設計される前のレジスタ転送レベルの回路に対してテスト容易化設計を行うものがある．レジスタ転送レベルの回路に対して非スキャン方式のテスト容易化設計の一つに階層テスト生成に基づく手法がある．階層テスト生成法では，2 段階でテスト生成を行う．まずレジスタ転送レベルを構成するモジュールに対してゲートレベルでテスト生成を行う．このモジュールはテスト容易な回路（組合せ回路など）でなければならない．次に，テスト生成されたテストパターンを外部入力からそのモジュールに伝搬し，モジュールの出力応答を外部出力まで伝搬させるための一連のテスト系列をレジスタ転送レベルで生成する．レジスタ転送レベル回路を構成する各モジュールに対して，外部入力から任意の値をそのモジュールの入力端子に伝搬可能でかつ，その出力端子の任意の値を外部出力まで伝搬可能ならば，その回路は強可検査性である．回路が強可検査であれば，スキャン設計した回路と同等のテスタビリティが得られる．回路を強可検査性にするためのテスト容易化設計が提案されている [6].

(4) テストデータの圧縮・展開

テストベクトルの表現を変換してテスタに格納するデータ量を削減する手法をテスト圧縮（Test Compression）という [*1]．近年のLSI の大規模複雑化に伴うテストデータ増大に対する対策として重要である．圧縮されたテスト入力系列はテスタから被テスト回路に送られLSI に埋め込まれた展開器によって圧縮前のテストデータに展開されたのち被テスト回路に入力される．一方，被テスト回路からの出力応答は，LSI 内の圧縮器によって圧縮されテスタに出力される．テスト入力系列の圧縮は符号化によって行われる手法が多い．ハフマン符号（Huffman Coding）に基づくものやテストデータ中の 0（または 1）のラン長（Run Length）に着目した符号化に基づく圧縮法であるGolomb, FDR, VIHC などが提案されている．多重スキャン設計では，一つの外部入力を複数のスキャンチェーン（シフトレジスタ状に接続されたフリップフロップ）の入力で共有することで，スキャン入力を削減するイリノイスキャン（Illinois Scan）や，組込み自己テスト回路（3-7-4(5)節）を応用してスキャン入力を削減するBAST（BIST Aided Scan Test）などがある．また，EDT（Embedded Deterministic Test）はテスト入力系列の展開器と応答圧縮器を合わせたアーキテクチャとして提案され，実用化されている手法の一つである [2, 4].

(5) 組込み自己テスト

テスト対象のLSI チップ内にテスト回路を埋め込み，チップ自身のテストを可能とする技術またはその回路構成を組込み自己テスト（Built-In Self-Test, BIST）という．BIST は高性能

[*1] テストパターン数を削減する方法もテスト圧縮（test compaction）という．3-7-3 節を参照のこと．

な外部テスト装置を必要としないため，テスト圧縮［3-7-4 (4)節］と並ぶテストコスト削減の重要な技術の一つである．一般に，BIST はテストパターン発生器（Pattern Generator）と応答解析器（Response Analyzer）からなる．BIST の構成はテストの対象となる回路が論理回路かメモリかによって大きく異なる．それぞれ，論理BIST，メモリBIST という．ここでは論理BIST を紹介する．メモリBIST については 3-7-6 節を参照のこと．パターン発生器，応答解析器には，それぞれ，疑似ランダムパターンを発生する線形フィードバックシフトレジスタ（LFSR, Linear Feedback Shift Register），MISR（Multiple Input Signature Register，マイザー）が用いられることが多い. 検出率の高いテスト系列を効率よく発生させるための手法として，フェーズシフタ（Phase Shifter）がある. 特に，初期値を入れ替えるリシーディング（Reseeding）やパターンの一部のビットを制御（反転Flip，または，固定Fix）する手法は，テスト生成アルゴリズムなどで得られたテスト系列をパターン発生器で発生させるための技術[*2]であり，上述のEDT, BAST（3-7-4(4)節）などのテストデータの圧縮・展開のための要素技術にもなっている. 一方, MISRによって圧縮された出力応答の系列はシグネチャ（Signature）と呼ばれる. シグネチャを元に被テスト回路の故障の有無を判定することをシグネチャ解析（Signature Analysis）という．被テスト回路からの出力応答のビット幅を削減するXORツリーや，応答圧縮に含まれる不定値が応答解析器に取り込まれることを防ぐXマスクも重要な技術である [2].

SoCのテスト（3-7-5 節）では，組込みプロセッサのプログラム機能を利用して内部のコアをテストするソフトウェアBIST（Software-Based Self-Test, Instruction-Based Self-Test）[*3]もある [3, 5].

(6) テスト容易化合成

面積，遅延などの一般的な最適化目標と合わせてテスト容易性を考慮して合成を行うことをテスト容易化合成（Synthesis For Testability）という．面積，遅延などの最適化を終えた回路に後からテスト容易化設計を適用する一般的な手法と比べてオーバーヘッドを小さくできる利点をもつ．テストの対象となる故障モデルやテスト生成アルゴリズムに応じた様々なテスト容易化論理合成法，テスト容易化高位合成法が提案されている．また，合成結果の回路に対して適用するテスト容易化設計法を指向し，そのオーバーヘッドの最小化を目的とする合成法もある [1].

(7) 高位テスト容易化設計

スキャン設計（3-7-4 (2)節）など，今日利用されている多くのテスト容易化設計法がゲートレベルを対象としているのに対し，動作レベルやレジスタ転送レベル記述の設計情報にテスト容易性を考慮した記述を追加・変更することを高位テスト容易化設計という．動作レベルテスト容易化設計（Behavioral Modification For Testability）としては，変数の代入操作を追加してレジスタの可制御性，可観測性の向上を指向したものがある．また，レジスタレベルテスト容易化設計としては，スキャン設計と同様に，レジスタをシフトレジスタ状に接続して制御観測を可能にするテストモードを追記するもの（レジスタ転送レベルスキャン設計［3-7-4(2)節］）などがある．このような手法は，合成系（高位合成アルゴリズム，論理合成アルゴリズム）がテスト容易性を考慮していないものであっても，テスト機構が通常動作の

[*2] このような組込み自己テストを決定論的 BIST（Deterministic BIST）という．疑似ランダムパターンによるものをランダム BIST（Random BIST）という．

[*3] これに対し，上述のような一般的な組込み自己テストはハードウェア BIST（Hardware-Based Self-Test）と呼ばれる．

ための回路と合わせて最適化されるので，面積，遅延などのオーバーヘッドを小さく，あるいは皆無にすることができるほか，実動作速度でのテスト実行（At-Speed Testing）ができるなどの利点をもつ[1,2].

■参考文献
1) N. K. Jha, S. K. Gupta, “Testing of Digital Systems,” Cambrige Univesity Press, 2003.
2) L.-T. Wang, C.-W. Wu, and X. Wen, “VLSI Test Principles and Architectures: Design for Testability,” Elsevier, 2006.
3) A. Krstic, L. Chen, W.-C. Lai, K.-T. Cheng, and S. Dey, “Embedded Software-Based Self-Test for ProgrammableCore-Based Designs,” IEEE Des. & Test, vol.19, no.4, pp.18-27, Jul/Aug. 2002.
4) 樋上，梶原，市原，高松，“論理回路に対するテストコスト削減法－テストデータ量および実行時間の削減－，”電子情報通信学会論文誌 D-I, vol.J87-D-I, no.3, pp.291-307, Mar. 2004.
5) 井上，神戸，V. Singh，藤原，“縮退故障とパス遅延故障のためのプロセッサの命令レベル自己テスト法，”電子情報通信学会和文論(DI), vol.J88-D-I, No.6, pp.1003-1011, Jun. 2005.
6) M. Abramovici, M.A. Breuer, and A.D. Friedman, “Digital Systems Testing and Testable Design,” Piscataway, New Jersey: IEEE Press, 1990.
7) Satoshi Ohtake, Hiroki Wada, Toshimitsu Masuzawa and Hideo Fujiwara, “A non-scan DFT method at register-transfer level to achieve complete fault efficiency,” Proc. of Asia and South Pacific Design Automation 2000 (ASP-DAC 2000), pp.599-604, Jan. 2000.

3-7-5 システムオンチップのテスト

（執筆者：米田友和）[2008 年 6 月 受領]

システムオンチップ設計では，IP コアベース設計が用いられることが多い．IP コアは，プロセッサ回路，メモリ回路，アナログ回路など様々である．それゆえに，用いられるテスト容易化設計方式も IP コアごとに異なり，テストパターンは各 IP コアごとに用意される．このような IP コアが組み込まれたコアベースシステムオンチップでは，新たなテスト問題に対処する必要が生ずる．それらは主に，テストアクセス機構の設計，コアラッパの設計，及び，テストスケジューリングである．

(1) テストアクセス機構設計

IP コアがシステムオンチップに組み込まれると，システムオンチップの外部入力から，IP コアに用意されたテストパターンを IP コアの入力まで伝搬し，更に，そのテストパターンに対する IP コアのテスト応答をシステムオンチップの外部出力に伝搬することは困難となる．そこで，各 IP コアに対して，用意されたテストパターン，及び，そのテスト応答の伝搬を実現するテストアクセス機構の設計が必要となる．テストアクセス機構の実現方式としては，テスト用に付加したハードウェアのみで実現するテストバス方式，テストレイル方式，境界スキャン方式，既存のハードウェアの一部をテスト時に再利用して実現する透明経路方式などが提案されている．

(2) コアラッパ設計

テストアクセス機構と IP コアのインタフェース回路となるのがコアラッパであり，IEEE Standard 1500 として標準化されている．コアラッパは IP コアを包みこむように設計され，通常動作とテスト動作で IP コアの入出力の接続先を切り替える機能をもつ．標準化では，通常動作（BYPASS），コアテスト（INTEST），コア外部テスト（EXTEST）の三つが必須動作として規定されている．

(3) テストスケジューリング

各 IP コアに対して，テストアクセス機構とコアラッパを設計することでシステムオンチップのテストは可能となる．しかしながら，これらの設計とシステムオンチップのテスト時間には密接な関係があり，テスト時間を考慮してテストアクセス機構とコアラッパを設計することが重要となる．例えば，システムオンチップの外部入力から同時に転送できるテストデータのビット幅はテストアクセス機構に依存する．また，同時にテストする IP コアのラッパに入力されるテストデータのビット幅の総和は，テストアクセス機構のビット幅を超えることはできない．これらの制約を考慮し，テスト時間の最小化を目的として，同時にテストする IP コアの組合せや順序を決定することをテストスケジューリングという．また，テスト時の消費電力は，通常動作時の 2〜3 倍であることが知られており，テストスケジューリングでは，テスト時の消費電力制約を考慮することも重要である．

3-7-6 メモリのテスト

（執筆者：安藏顕一）[2008 年 6 月 受領]

(1) メモリの製造テスト

メモリデバイスの製造テストは，デバイス製造時の欠陥をスクリーニングするためのものであり，実使用時に想定される温度や電圧，周波数などの仕様範囲で動作することを保障するために行われる．メモリ機能単体を 1 チップとする汎用メモリに対しては，メモリテスタを用いて多数個同時テストが可能であり，これによりテストコストを抑制しながら，必要な品質を達成するためのテストを行うことができる．一方システム機能の一部としてチップに搭載されるメモリでは，一つの LSI チップ上でも，多種のメモリデバイスや多数の異なる構成のデバイスがシステム内に埋め込まれており，テスタからの同時テストが困難である．このため，後述の組込み自己テスト（BIST: Built-In Self Test）の技術が必須となる．

(2) メモリのテストアルゴリズム

メモリの代表的な機能的故障モデルには，縮退故障，開放故障，遷移故障，状態相関故障，多重選択故障などがあり，これらのモデルを対象としたテストアルゴリズムの開発が行われている [1]．一方メモリデバイスのレイアウト上の特徴を考慮し，想定される実際の物理的欠陥を対象としてテストアルゴリズムを開発する，帰納的故障解析（IFA: Inductive Fault Analysis）手法も提案され，各アルゴリズムの欠陥検出能力の比較が行われている [2]．

メモリテスト内容は，マーチング（Marching）やウォーキング（Walking），ギャロッピング（Galloping）などのテストアルゴリズムと，電圧・温度やアドレス方向，データバックグランドなどのテストストレス条件の組合せで表現される [3]．マーチングテストのいくつかは，前述の機能的故障モデルや IFA での欠陥検出能力が高く，テスト時間もメモリのサイズ（ワード数）に対し線形で収まるので，基本的なアルゴリズムとして用いられている．DRAM やフラッシュメモリに対してはほかに，デバイス特性に応じたテストが追加される．

(3) メモリの組込み自己テスト技術

メモリの組込み自己テストは，メモリテストに必要なテスト用信号を生成し，結果を判定する論理回路を，メモリデバイス自身あるいはメモリが埋め込まれた LSI に搭載する技術であり，特に多数のメモリを搭載した SoC において重要度が高い．汎用メモリでも，少数ピンテストによる多数個同時テストの要求から，自己テスト技術が用いられる．この場合デバイス自体に BIST 回路が組み込まれるか，テスト治具上に自己テスト用機能をもつデバイスを

搭載する BOST（Built-Out Self Test）手法が用いられる．

BIST は例えば RAM に対しては，アドレスやデータ，書込みや読出し動作制御などのメモリ入力信号を自動的に生成して各メモリに与え，一方各メモリの出力に不良があるかを判定する回路をもつ必要がある．これらはメモリデバイスに対して一対一に付加されるか，回路サイズを抑えるため複数デバイスで共有される．BIST とメモリ間のデータ転送には，データ幅分を一度にやり取りするパラレル BIST 方式と，シリアルなシフトパスでやり取りするシリアル BIST 方式とがある．テスト発生がハードウェアで固定的に実現され，回路設計後に変更することができない固定パターン BIST が歴史的に適用されてきたが，プロセス技術の微細化により必要となるパターン追加や，市場不良品の解析などの面で限界があるとされている．このため，マイクロコードをテスト時にロードすることによりテストの設定をある程度自由に行える，プログラマブル BIST も実用化されている [4]．

■参考文献
1) C.-W. Wu, “Memory Testing and Built-In Self-Test,” in “VLSI Test Principles and Architectures,” ed. L.-T. Wang et al, Morgan Kaufmann Publishers (an imprint of Elsevier Inc.), pp.462-515, 2006.
2) J.P. Shen, W.Mary, and F.J. Ferguson, “Inductive Fault Analysis of CMOS Integrated Circuits,” IEEE Design & Test of Computers, vol.2, no.6, pp.13-26, Dec. 1985.
3) A.J. van de Goor, S. Hamdioui and R. Wadsworth, “Detecting Faults in the Peripheral Circuits and an Evaluation of SRAM Tests,” IEEE International Test Conference, pp.114-123, 2004.
4) J. Dreibelbis, J. Barth, H. Kalter, R. Kho, “Processor-based Built-In Self-Test for Embedded DRAM,” IEEE Joumal of Solid-state Circuits, vol.33, no.11, pp.1731-1740, Nov. 1998.

3-7-7 アナログ／ミックスドシグナル系のテスト

(1) テストの概要

（執筆者：三浦幸也）[2008 年 9 月 受領]

多くのアナログ／ミックスドシグナル（AMS）回路では，連続値である電圧や電流（連続信号）が意味のある信号として扱われる．また AMS 回路には多種多様な回路機能や回路構成が存在し，回路機能と回路構成が一意に対応しない [1]．このため，AMS 回路のテストにおいては，故障モデルの構築が困難であるので抽象的な故障モデルの概念が存在しない．また AMS 回路では，回路内のパラメータ変動や使用環境の変動などに対して回路特性を安定させるために帰還回路が用いられており，AMS 回路自体が耐故障性を有している．更に，故障箇所における故障情報の現れ方が複雑であることや，故障箇所から特定方向への故障の影響の伝搬が容易でないことなどから，外部出力での故障影響の観測が困難である．

AMS 回路が正常か否かを識別することは，その回路の仕様（スペック）や用途に深く依存している [1]．このように AMS 回路のテストは，回路仕様を満たすか否かを調べることに帰着するので，テスト項目が多数あり，また高い計測精度が要求される．このほかに，回路構造や回路機能に依存しないテスト方法として，自己発振回路を構成して動作速度や回路特性を調べるオシレーションテスト，回路内の素子パラメータの変動による回路動作への影響を調べる感度解析に基づいたテスト，信号のタイミングを評価するジッターテスト，などがある．近年は高周波 AMS 回路をテストするために低コスト・広帯域・高精度のテスト方法への要求も高まってきている．

(2) AD コンバータ，DA コンバータのテスト

（執筆者：三浦幸也）[2008 年 9 月 受領]

AD コンバータ（ADC）と DA コンバータ（DAC）の特性を表す代表的な指標として，量子化誤差と非直線性誤差がある．**図 3･20** に 2 ビット出力の ADC の理想伝達特性を示す．n ビット出力の場合，アナログ入力電圧範囲（フルスケールレンジ：FSR）に対して，FSR/2^n ごとの入力に対して出力コードが変化する．量子化誤差はアナログ値（連続値）とディジタル値（離散値）との変換過程で本質的に発生し，図 3･20 に示す理想コンバータにおいても最大で±(1/2)LSB の量子化誤差がある．非直線性誤差は微分非直線性誤差（DNL）と積分非直線性誤差（INL）に分けられる．それぞれ，1LSB に対応する理想電圧と実際のアナログ入力（DAC では出力）電圧との差，及び理想変換直線との最大偏差を表す．このほかに ADC や DAC の特性を表す指標としてゲイン誤差やオフセット誤差などがある．

ADC の一般的な非直線性誤差のテスト法としてヒストグラム法がある．**図 3･21** は，図 3･20 の変換特性に対して，ランプ電圧を印加しながら 4 サンプル／出力コードで出力コードの発生回数をカウントしてヒストグラムを作成した例である．この方法では，検出可能な DNL の精度は 1LSB ごとのサンプル数に依存する．ADC と DAC にも各種の回路構成方法があるため，実際には回路構成ごとにテスト項目（評価特性）が異なる．

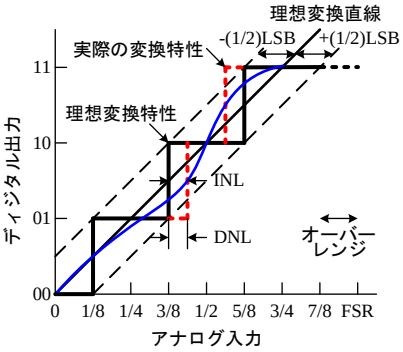

図 3・20　2 ビット ADC の理想伝達特性

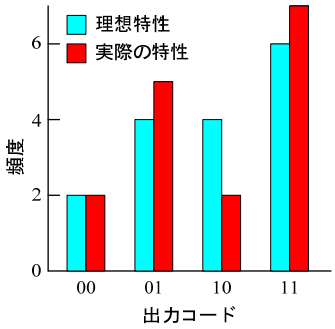

図 3・21　ヒストグラム法

(3) テスト容易化設計手法，テストインタフェース

（執筆者：亀山修一）[2008 年 9 月 受領]

電子回路の高集積化に伴い，従来の実装基板試験手法であるインサーキットテスト（ICT）や自動外観検査（AOI）ではテストカバレージ低下の問題があり，IEEE1149.1 規約に基づくテスト容易化設計手法（バウンダリスキャンテスト／JTAG テスト）が広く用いられるようになった．ディジタル回路を対象とした 1149.1 規約をアナログまで拡張したものが 1149.4 規約（通称アナログバウンダリスキャン）である [2, 3]．

1149.4 規約では物理的なテストインタフェースピンとして従来の 1149.1 規約での 5 ピン（TDI/TDO/TCK/TMS/TRST）に加え，アナログテストポート（AT1/AT2）の 2 ピンが追加されている．LSI デバイスは**図 3･22** のようにパッケージピンと内部コア間にテスト用の回路が組み込まれ，ディジタル信号ピンには従来のバウンダリスキャンレジスタと同等のディジタルバウンダリモジュール（DBM），アナログ信号ピンにはアナログバウンダリモジュール

（ABM）が組み込まれている．各 ABM はアナログテストバス（AB1/AB2）でアナログテストポートと接続される．AT1/AT2 は基板上のテストバスに接続され，外部のテスタに接続される．

テストインタフェースにおいて，1149.4 規約は 1149.1 規約の上位互換となっており，各々の規格に基づく LSI を基板上で共通のテストバスに接続可能である．

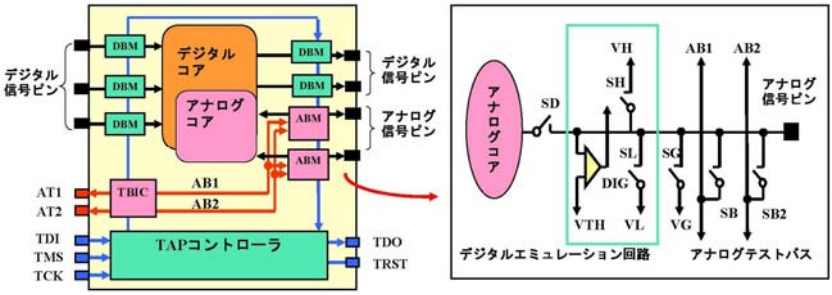

図 3・22　IEEE1149.4 対応デバイスの構造と ABM 回路

(a)　インタコネクションテスト(EXTEST)

ABM のディジタルエミュレーション回路により，アナログネットを 1149.1 でのディジタルネット等価にし，LSI 間のインタコネクション（オープン／ショート）試験を行う機能（**図 3･23**）．

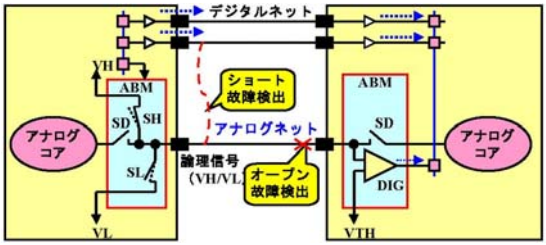

図 3・23　インタコネクションテスト

(b)　パラメトリックテスト（インピーダンス測定）

アナログ信号ピンに接続されている基板上の部品（抵抗，コンデンサなど）の値を ABM 経由で外部計測器に接続して測定し，正しい部品が実装されているかテストする機能（**図 3･24**）．

(c)　アナログプローブ

システム動作中の任意のアナログピンの信号波形を ABM 経由で外部計測器に接続して観測する機能（**図 3･25**）．

(d)　インテスト（INTEST）

LSI 内部のアナログ／ディジタル混在回路（DAC, ADC など）をテストする機能．

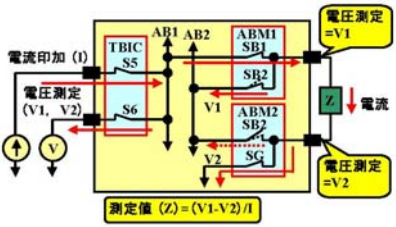

図 3・24　パラメトリックテスト

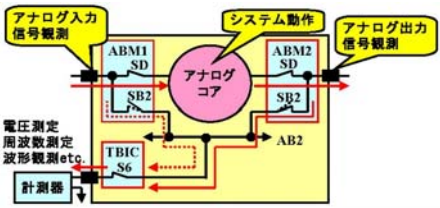

図 3・25　アナログプローブ機能

■参考文献

1) P. Kabisatpathy, A. Barua, and S. Sinha, “Fault diagnosis of analog integrated circuits,” Springer, The Netherlands, 2005.
2) “IEEE Standard for a Mixed-Signal Test Bus (IEEE Std 1149.4-1999),” IEEE, New York, 2000.
3) Kenneth P. Parker, “THE BOUNDARY-SCAN HANDBOOK Third Edition,” Kluwer Academic Publications, 2003.

■10 群 － 1 編 － 3 章

3-8 低消費電力設計

(執筆者：中村　宏)[2008 年 10 月 受領]

集積度の向上に伴い集積回路の消費電力は増大し，特に単位面積当たりの消費電力は冷却の観点から製造技術上大きな問題となるため，集積回路の低消費電力化は重要な設計技術となっている．性能を低下させずに消費電力を抑えるには，不要不急の動作をする回路素子を停止させるか低速に動作させるのが効果的である．この実現のために，主にアーキテクチャ技術は集積回路内で不要不急の動作をする部分を抽出する役割を，回路技術は効果的に動作を停止あるいは低速にする役割をそれぞれ担う．本節では，この低消費電力化を実現するアーキテクチャレベルから回路レベルまでの設計技術を説明する．

3-8-1 低電力化システムアーキテクチャ技術

消費電力には，スイッチング動作に伴うダイナミック電力（Dynamic Power）と，スイッチングの有無によらずリーク電流によって消費されるスタティック電力（Static Power）があり，後者はリーク電力(Leakage Power)とも呼ばれる．ダイナミック消費電力 P_{dyn} は $P_{dyn} = C V_{DD}^2 fa$ で表され，C は負荷容量，V_{DD} は電源電圧，f は周波数，a はスイッチング確率，をそれぞれ表す．

(1)　電源電圧・周波数制御

電源電圧を低下させるとスイッチング動作が遅くなるため周波数も低下する．その結果，電源電圧の 2 乗と周波数の積に比例するダイナミック消費電力は大きく低減される．また，ある処理を完了するのに必要な消費エネルギーも，必要となるスイッチング回数が等しいと仮定すれば，1 回のスイッチングに要するエネルギー，すなわち電源電圧の 2 乗に比例するため大きく削減される．したがって，設計対象のシステムが満たすべき性能制約が与えられた場合，その制約を満たす範囲で可能な限り電源電圧と周波数を低下させることが良い設計となる．いつも決められたジョブを実行するシステムでは，性能制約を満たす最低周波数があらかじめ決定できるため，その周波数に合わせた電源電圧でシステムを設計すれば良い．これは静的な電源電圧・周波数制御と呼ばれる．しかし，例えプログラムが同じでもデータセットが違うなどの理由で実行に要する時間が変化する場合には，実行時に OS（Operating System）などを介して実行状況を把握し，実行時に必要最低限の周波数を決定し，実行時に周波数と電源電圧を変更する動的電源電圧周波数制御（DVFS: Dynamic Voltage and Frequency Scaling）が有効となる．

(2)　スタンバイ時の電源制御

システム LSI のように多くの機能を一つの LSI で実現する場合，すべてのモジュールが常に動作する必要はない．例えば携帯電話の待受け時などは，着信処理に必要な部分以外は動作する必要はない．そのようなモジュールにおいてはスタンバイ（Stand-By）時には電源を遮断することで消費電力を抑えることができる．

(3)　性能と消費電力のモデリング

システム全体の消費電力は各モジュールの消費電力の総和となるが，性能は各部の処理能力の最小値で決定される．このとき，システム全体の性能を律する部分をボトルネック部と

呼ぶ．ボトルネック部以外の処理能力を向上させると，システム全体の性能は向上しないにもかかわらずその部分の消費電力は上昇するため，消費電力当たりの性能という観点からは効率の悪い設計となる．そのため，ボトルネック部以外のモジュールの処理能力を，ボトルネック部の処理能力と均衡が取れるように低下させ，消費電力も低下させるのが良い設計となる．この実現には，各モジュールの性能と消費電力の関係のモデリングと，実現すべき処理が各モジュールに要求する処理能力をあらかじめ導出しておけばよい．この情報に基づいて，各モジュールの設計では要求される処理能力を実現し，しかも最も消費電力の低い設計を選択することが可能となるからである．設計上難しいのは，各モジュールに要求される処理能力が実現すべき処理の特徴に依存する場合である．このような場合には各モジュールの電源電圧・周波数を処理内容に応じて制御する，などの動的な制御が効果的となる．その場合でも，各モジュールの性能と消費電力を電源電圧・周波数などの動的制御するパラメータを用いてモデリングしておく必要がある．

3-8-2 低電力化アーキテクチャ

(1) 並列性を利用した電源電圧・周波数制御

3-8-1(1)節で述べたように電源電圧と周波数を低下させることで消費電力と消費エネルギーを低減できるが性能も低下するため処理時間は長くなる．この性能低下は，並列性（Parallelism）を活用した高速化で補償できる．電源電圧をα (0< α <1)倍に低下させると周波数がβ (0< β <1)倍に低下すると仮定する．この場合 $P_{dyn} = C\, V_{DD}^2\, f\, a$ の式より消費電力は$\alpha^2\beta$ に低下するが，性能もβに低下（処理時間は $1/\beta$）し，消費電力と処理時間の積である消費エネルギーはα^2 に低下する．ここで，プロセッサ N 個による並列処理を考える．理想的に並列性を活用できれば性能も N 倍になるため，消費電力は $N\,\alpha^2\beta$ に増加するが性能も $N\beta$ になるため，消費エネルギーはα^2 に低減できたままである．したがって，（$N\beta$>1）を満たすように N とβを選択できれば，消費エネルギーの低減と高速化の両方を達成できる．更に，（$N\alpha^2\beta$ < 1）をも満たすようにαを選択できれば，消費電力の低減をも達成できる．最近の高性能マイクロプロセッサが，周波数向上による単体プロセッサの性能向上ではなく，同一チップ上に複数のプロセッサコアを搭載するチップマルチプロセッサ（Chip Multi-Processor）で性能向上を目指すのも，この原理による．上記の例では，並列性を理想的に活用できるとしたが，現実的には並列性は理想的ではなく，投入するハードウェア資源を N 倍にしても性能は N 倍にはならない．仮に，上記の例で，並列効果が全くない場合には消費エネルギーは $N\alpha^2$ になるため，プロセッサ台数を増やすと消費エネルギーはかえって増加する．このように，この手法では並列性をどれだけ活用できるかが重要となる．

(2) 専用アクセラレータ

消費電力当たりの処理性能は消費電力効率と呼ばれるが，消費電力効率と汎用性はトレードオフ関係にある．すなわち，汎用性を指向すると消費電力効率が低下する．専用アクセラレータ（Accelerator）はその逆で，汎用性を失う代わりに高い消費電力効率を目指すものである．このトレードオフ関係は定性的には以下のように説明できる．汎用性を指向した場合，限られたハードウェア資源で広い範囲の処理に適用するため，種々の処理が必要とする最大公約数的な基本機能（具体的には加減算や乗除算など）のみをハードウェア機能として提供し，各命令はこれらの基本機能のみを指示する設計となる．そのため，複雑な処理を実現す

るためには複数の命令の組合せで実現する必要があり，命令数が増え，更に演算器間でのデータ転送，レジスタなどの記憶素子に対する中間結果の書込みと再読出しが必要となり，これらの部分で消費される無駄な電力が増加する．文献 1)では，汎用高性能マイクロプロセッサにおいては，処理すべき命令の選択・供給や，データ処理部への有効なデータの供給のために必要な消費エネルギーの方が，本来必要なデータの処理・演算に要する消費エネルギーより大きく，スーパースカラ（Superscalar）処理のように命令レベル並列性（Instruction Level Parallelism）を利用した高性能化を目指すとこの傾向がより更に強くなることが示されている．これらの問題に対し，専用アクセラレータは汎用性を犠牲にしつつも，特定の処理に特化したハードウェアを提供することで，上記の無駄な消費電力を削減し消費電力当たりの性能の向上を目指すものである．

3-8-3 低電力化メモリアーキテクチャ技術

すべての命令とデータをメモリに保持するフォンノイマンアーキテクチャ（Von Neumann Architecture）において，メモリはシステム全体の性能を律する隘路（あいろ）でありフォンノイマンボトルネックと呼ばれているが，消費電力も大きい部分である．低電力化メモリアーキテクチャは，従来は高性能化のために最適化されているメモリ階層を，消費電力の観点から最適化するアーキテクチャ技術ということができ，レジスタ（Register）やキャッシュ（Cache Memory）などの高速化のためのメモリ階層が主たる対象となる．

(1) レジスタの低消費電力化

レジスタはメモリ階層の最上位に位置しアクセス頻度は最も高い．ダイナミック消費電力はアクセス頻度とアクセス当たりの消費エネルギーの積に比例するため，プロセッサ内では最も消費電力の高い部分の一つである．1 回のアクセスに要するエネルギーと，レジスタ容量やポート数とは正の相関がある．しかし，命令レベル並列性（Instruction Level Parallelism）を活用するプロセッサの性能向上には，レジスタ容量とポート数の増加が必須である．ここに，レジスタの低消費電力化とプロセッサの性能向上には相容れないトレードオフ関係が存在する．この問題を解決するために，以下のような手法が提案されている．

(a) レジスタの多バンク化

レジスタを複数のバンクに分割することで一つのバンク当たりの容量とポート数を削減し，アクセスあたりのエネルギーを削減する手法である．しかし，各バンク当たりのポート数が減少するため，特定のレジスタバンクにアクセスが集中するとポート不足により競合（バンクコンフリクト）が発生し性能が著しく低下する．バンクコンフリクトを回避するためには，バンク構成を考慮してレジスタアクセスを適切に調整する必要があるが，これをコンパイラが行うことは，バンク構成という詳細なハードウェア構成をソフトウェアに意識させることとなり一般には容易ではない．一方，ベクトルプロセッサにおけるベクトルレジスタ（Vector Register）ではこの手法が広く用いられている．ベクトル処理においては，一つのベクトルレジスタ内の各要素へのアクセスは連続的に行われる．この特徴により，ベクトルレジスタをバンク構成にしてもアクセスは全バンクへ分散されるためバンクコンフリクトによる性能低下は発生しない．

(b) ビット分割レジスタ

レジスタが保持すべきビット幅はアーキテクチャによって規定されるが，プログラム実行

中に実際にレジスタに保持されるデータの有効ビット幅は小さい場合が多い．そこで，上位ビットを保持するレジスタと下位ビットを保持するレジスタに分割する手法が文献 2)で提案されている．この構成では，実行中にデータの有効なビット幅をハードウェア的に検出しながら，ビット幅が小さいデータは下位ビット用のレジスタのみを用い，ビット幅の大きいデータだけは両方のレジスタを用いる．分割されたレジスタのポート数は分割前と同じであるが，不要なビットを保持する必要がなく総容量を減らせるため，消費電力を削減することができる．

(c)　フォワーディングによるポート削減レジスタ

命令のオペランドは常にレジスタから供給されるわけではなく，直前の命令により生成される結果をフォワーディング（バイパス転送）して使うことも多い．文献 3)で提案されたこの手法は，このことを積極的に利用しあらかじめレジスタのポート数を削減することで消費電力の削減を目指すものである．フォワーディングの発生頻度は実行命令順に依存するため，場合によってはポートが不足し性能が低下することもある．

(2)　ウェイ予測キャッシュ

キャッシュメモリのダイナミック消費電力の削減を目指す手法であり，文献 4)で提案されている．セットアソシアティブキャッシュでは，処理の高速化のためすべてのウェイをアクセスしたあとで，有効なデータを有するウェイを判定しデータを選択する．この場合，有効なデータを有するウェイ以外へのアクセスは無駄な電力を消費している．この点に着目し，どのウェイに所望のデータがあるかを予測し，予測に基づいて一つのウェイだけをアクセスする手法である．予測が正しかった場合には電力消費を削減できるが，予測が間違っていた場合には，再度ほかのウェイをアクセスする必要があるため，実行時間が延び性能が低下する．予測手法としては MRU（Most Recently Used），すなわち前回有効なデータを有していたウェイが今回も所望のデータを有すると予測するのが一般的には有効であることが知られている．

(3)　Cache Decay

キャッシュメモリのスタティック消費電力削減手法であり，文献 5)で提案されている．キャッシュの各ラインの状態を Alive と Dead に分類すると，プログラム実行中に Dead 状態の時間が大半であることが分かった．ここで，Alive 状態というのは該当ラインが将来参照される状態であり，Dead 状態は将来にわたって参照されない状態を指す．Dead 状態のラインは不要なデータを保持していることになる．Cache Decay は，そのラインの電源電圧を遮断することでリーク電力を削減する手法である．電源電圧を遮断するため該当ラインのデータは失われる．問題は，Dead 状態か Alive 状態かの判断は将来のデータ参照で決まるため，正確には判定できないことである．そこで Decay Interval という時間間隔を導入し，各ラインが Decay Interval で定義された時間アクセスされない場合には，該当ラインを Dead 状態と判定し電源電圧を遮断する手法が提案されている．この判定方法は必ずしも正確ではないが，仮に Alive 状態のラインを Dead 状態と判定してデータを失った後にアクセスがあっても，キャッシュミスと判定すれば下位層のメモリにある正しいデータを利用できるため，正しい実行は保証される．この Decay Interval を長くするとリーク電力削減効果は減少するが，短くすると Alive 状態のラインを誤って Dead 状態と判定する確率が増え，キャッシュミスの増加により電力消費が大きい下位層のメモリアクセス回数が増え，かえって消費電力が増加する．し

たがって，この Decay Interval を適切に選択することが重要となるが，適切な Decay Interval はキャッシュの構成と実行するプログラムの特徴に依存するので，適切な選択は容易ではない．

(4) Drowsy Cache

これもキャッシュメモリのスタティック消費電力削減手法であり文献 6)で提案されている．原理は前項(3)の Cache Decay と同じであるが，Dead 状態の電源を遮断せず電圧を低くすることで，低リーク状態でライン中のデータを保持する点が異なる．この低リーク状態を Drowsy 状態（直訳するとうたた寝状態）と呼ぶ．電源を遮断しないので，各ラインのリーク削減効果は前項(3)の Cache Decay ほどではない．しかし，Drowsy 状態のラインへアクセスがあっても，Drowsy 状態から通常モードへ戻してからデータを読むことが可能であり，アクセス時間は長くなるが，下位層へのメモリアクセスが不要である．このため，Cache Decay に比べると誤って Alive 状態のラインを低リーク状態へ遷移させても処理時間と消費電力の増加は抑えられる利点がある．

(5) ソフトウェア可制御メモリ

キャッシュメモリがハードウェアで制御されるに対し，ソフトウェアから制御できるアドレス指定可能な高速メモリをプロセッサチップ上に搭載する構成方法である．この構成手法はキャッシュメモリの登場前から存在するもので，スクラッチパッドメモリ（Scratch Pad Memory）とも呼ばれている．当初は，集積度の不足から極めて小容量の高速メモリしか実現できなかったため，スクラッチパッドメモリをソフトウェアから苦労して利用していた．その後，キャッシュメモリが広く用いられるようになったが，組込みシステム用プロセッサなどではソフトウェア可制御メモリは引き続き用いられていた．これらのシステムが重視するリアルタイム性やハードウェアコストに優れているためである．また，処理の特性，特にデータアクセスの特性に応じて適切に制御することでキャッシュメモリよりも高性能を達成できるため，ゲーム用などの高性能プロセッサでも使われるようになっている．更に，ソフトウェア可制御メモリは消費電力でもキャッシュメモリに対して以下の優位点があることが文献 7)で述べられている．

- **ダイナミック消費電力**：キャッシュメモリではどのウェイにデータが存在するか分からないため全ウェイをアクセスする必要がある．(2)のウェイ予測キャッシュを適用しても予測が外れれば消費電力は削減できず性能は低下する．これに対し，ソフトウェア可制御メモリはアドレス指定可能なメモリであるため，データの存在場所を一意にアドレスで指定されるため，必要な箇所のみをアクセスすることが可能である．
- **スタティック消費電力**：(3)や(4)で述べたキャッシュメモリのスタティック消費電力削減手法において問題なのは，各データが将来いつ参照されるか分からないために，各ラインの dead 状態を正確に判定できない点である．これに対しソフトウェア可制御メモリでは，参照はすべてソフトウェアで明示的に行われるため，将来参照されないデータをソフトウェアは知ることができる．このため，誤りなく Dead 状態の箇所を判定でき，理想的に該当部のリーク電力を遮断することが可能である．

3-8-4 低電力マイクロアーキテクチャ技術

(1) パイプラインゲーティング

条件分岐命令があると分岐の成否が判明しないと次に実行すべき命令は決定されないが，

高性能プロセッサでは分岐の成否を予測して後続命令を投機的に実行する高速化手法が広く採用されている．しかし，分岐予測（Branch Prediction）が間違っていた場合には，投機実行（Speculative Execution）された命令の処理は意味がなく，それらの命令が消費するエネルギーは無駄である．パイプラインゲーティング（Pipeline Gating）は文献 8)で提案された手法であり，分岐予測の確度を別の手法で得て，確度が低いときには後続命令のパイプライン処理全体を止める手法である．

(2) パイプラインバランシング

これは，命令レベル並列性を活用し 1 サイクルに複数命令を実行するスーパースカラ処理に適用される手法である．これらのプロセッサは，同時に何命令を実行するか，を表す命令発行幅（Instruction Issue Width）を設計時に決め，並列実行可能な命令をその数だけ選択可能な論理や，その数だけの命令を同時に実行するのに必要な複数の演算ユニットなどがハードウェアで実現される．しかし，実行するプログラムの命令レベル並列性がこの上限値より小さい場合には，用意したハードウェア機能が全部は使われないため，これらの部分で消費される電力は無駄となる．この点に着目し，文献 9)では，実行中の処理の命令レベル並列度を観測し，この並列度が小さい場合には，ハードウェア部の機能を縮退させることで消費電力を削減する手法を提案している．ハードウェアが提供する命令パイプラインの処理能力を，実行中の処理が要求する能力と均衡をとらせることから，パイプラインバランシング（Pipeline Balancing）と呼ばれている．

(3) パイプラインステージ統合

文献 10)で提案されているこの手法は，命令レベル並列性が十分ではないときに，命令パイプラインステージを融合する手法である．ステージ統合は，ステージ間のレジスタをバイパスし隣接するステージを直列に接続することで実現される．ステージ統合時には周波数は低下しハードウェアが提供できる処理能力も低下するが，命令レベル並列性が少ないときには性能への影響は限定的である．一方，ステージ統合時にはステージ間レジスタがバイパスされるため，当該レジスタの読み書きと当該レジスタへのクロック分配が不要となることから消費電力は削減される．

(4) 命令キューの動的サイジング

命令の実行順が動的に決定される高性能プロセッサでは，命令は命令キュー（Instruction Queue）と呼ばれる記憶部に一度保持され，その中から，命令間の依存関係がなく並列に実行可能な命令が選択されて演算ユニットへ発行（Issue）される．命令キューの容量は大きい方が多くの命令を保持できるため，依存関係のない命令が命令キュー内に存在する確率が増える．そのため，命令レベル並列性を抽出しやすく性能向上に有利である．しかし，命令キュー内の全命令に対し依存関係の有無を判定する必要があるため，容量を大きくするとその判定に要する消費電力も増大する．文献 11)で提案されているこの手法は，必要十分な命令キューの大きさが実行するプログラムの性質に依存することに着目し，命令キューのどの部分から発行すべき命令が選択されたかの情報を実行時にハードウェア的に収集し，命令キューを小さくしても性能が低下しないと判定できるときには命令キューの一部を利用しないことで，電力を削減する手法である．

(5) 命令グルーピングによる命令キューの低消費電力化

この手法も前項(4)と同じく命令キューの消費電力を抑える手法である．命令キューにおけ

るダイナミック消費電力は，3-8-3(1)節で述べたレジスタと同様に，キューの容量やポート数と正の相関がある．前項(4)は容量を小さくすることを目指したが，本手法はポート数の削減を目指す．命令キューでは，命令キュー内の任意の位置から命令発行幅分の命令を選択・発行する必要があるため，必要となるポート数はこの命令発行幅となる．しかし，命令キューを二つに分割し，各々のポート数を半分にすることができれば，命令キューの実装面積を大きく削減でき，消費電力を抑えることができる．命令キューにおいては，命令キュー内の全命令の依存関係を調べる必要があるため，このような分割は通常不可能である．しかし，命令キューに命令を入れるときに，片方だけのバンク内だけで命令の依存関係を調べればよいようなヒントを付加できれば，この分割は可能になる．文献 12)ではこの点に着目し，命令を命令キューに入れるときに，例えば「命令Ｂは命令Ａにしか依存しないので，命令Ａが発行できればその後では必ず命令Ｂが発行できる」などの，依存関係が当該命令間のみにしか存在しない命令群をグループ化することを提案している．このようにすれば，命令Ｂの発行可能性は調べる必要がなく，命令Ａの発行可能性だけを調べればよい．なぜなら，命令Ｂは，同じグループの命令Ａが発行されれば必ず発行できるからである．このことを利用し，命令キューを 2 分割し，片方のキューだけにキュー内の全命令の依存関係を調べ発行可能な命令を選択するハードウェアをもたせ，上記命令Ａはこの命令キューに，上記命令Ｂは他方の命令キューに保持する．後者のキューからの命令発行は，同じグループに属する前者のキューから命令が発行されたか否か，だけで判断できる．この構成では，各キューのポート数は命令発行幅の半分で十分なため，実装面積が小さくなり，ダイナミック消費電力を削減できる．

(6) GALS 構成における動的電源電圧・周波数制御

GALS（Globally Asynchronous Locally Synchronous）構成とは，LSI 内部を単一のクロックで動作させるのではなく，**図 3･26** に示すように LSI 内部を複数の領域に分割し，個々の領域内は単一クロックで同期動作をさせるが各領域は独立したクロックで動作させ，領域間の転送路は非同期動作させる構成である．GALS はもともと，半導体微細化が進み LSI 全体を単一クロックで動作させることが難しい，という発想から出てきた構成法である．しかし，LSI 内のハードウェアすべてが常に忙しく動作する必要はないことに着目すると，GALS 構成を採用すれば，各領域に必要とされる処理速度に応じて各領域の動的電源電圧・周波数制御を行うことができるため，単一クロックで動作する場合と比較するとより消費電力の削減を実現できる．この手法で重要となるのは，各局所同期領域に必要とされる処理能力を適切に把握し，その要求に見合った電源電圧と周波数を選択することである．文献 13)では，Attack and Decay と呼ばれる手法が提案されている．これは，局所同期で動作するモジュールの負荷の変化量があらかじめ設定した閾値を越える場合には電源電圧と周波数を積極的に上昇または下降させ（Attack），それ以外の場合には，電源電圧と周波数を緩やかに下降させる（Decay）手法である．

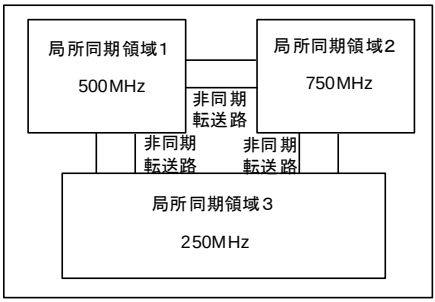

図 3・26　GALS 構成

(7)　タイミング余裕を利用した低消費電力化

半導体の微細化に伴い素子特性のばらつき（Variation）は増大し，温度や電源電圧などの環境変動に起因するばらつきも大きくなる．そのため，典型的な回路遅延と最悪ケースの回路遅延の差は大きくなり，まれにしか発生しない回路遅延の最悪ケースに合わせたタイミング設計を行うと，必要となるタイミング余裕を大きくせざるを得ず，動作速度向上に大きな足かせとなっている．この手法は，最悪ケースは極めてまれにしか起こらないことに着目し，典型的な回路遅延で動作するときには無駄な時間となっているこのタイミング余裕を，低電力化に積極的に利用する手法である．文献 14)では，まれに発生する最悪ケースではタイミング誤りとなるが，それ以外の場合には回路の遅延時間がタイミング制約を満たす程度に，電源電圧を低下させることで低電力化を図る手法が提案されている．**図 3･27** にその原理を示す．図 3･27 の左側は，周波数で規定される遅延時間よりも必ず遅延時間が短くなる通常設計を表す．このとき，電源電圧を低下させると右側のような遅延時間の分布になり，図中の○で囲まれた部分はタイミング誤りとなり誤動作となる．しかし，ほとんどの場合には正常動作するため，電源電圧低下による低電力化が達成できる．この手法では，タイミング誤りを検出する論理が必要となる．タイミング誤り検出時には再実行による回復が必要となるが，分岐予測を用いた投機実行を行う高性能プロセッサでは予測失敗時のための再実行機構をもつので，その機構を流用できる．タイミング誤りの発生確率が上がると，再実行による性能低下と電力増加が生ずるので，その発生確率を極めて低く抑えつつ電源電圧をできるだけ低下させることが，この手法で消費電力を削減するためには重要となる．

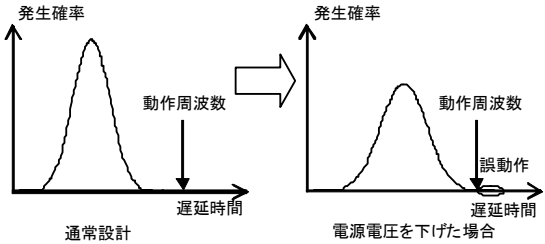

図 3・27　タイミング余裕を利用した低電力化

3-8-5 レジスタトランスファレベルでの低消費電力化技術

(1) ゲーテッドクロック

フリップフロップやレジスタなど，クロックに同期して入力データを保持する記憶素子では，クロック入力のたびに新しいデータが書き込まれる．しかし，記憶素子への入力が有効でない場合には，新しいデータを保持する必要はない．このことに着目し，新しいデータを保持する必要はないときに，クロックの信号を抑止する手法がゲーテッドクロックである．**図 3･28** にその構成例を示す．この図では，レジスタを D フリップフロップで構成している．図 3･28(a)はゲーテッドクロック非適用の回路で，イネーブル信号が‘1’のときだけ組合せ回路の出力を保持するが，‘0’のときにもレジスタへの書込みは行われるため，電力を消費する．これに対し，図 3･28(b)では，イネーブル信号が‘0’のときにはクロック信号の変化が伝播されないため，書込みは行われず電力消費が抑えられる．なお，図 3･28(b)の回路では，クロック信号が‘1’のときにイネーブル信号にグリッチ（Glitch）が発生するとそのまま記憶素子のクロック端子に伝播して誤動作を引き起こす．これを避けるために図 3･28(c)の回路を用いるのが一般的である．

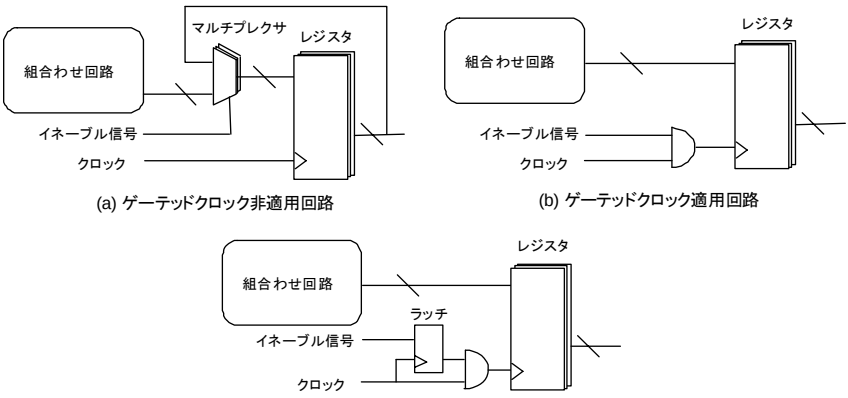

図 3・28　ゲーテッドクロック

(2) オペランドアイソレーション

演算回路などの組合せ論理は毎サイクル動作する必要がない場合も多いが，入力値が変化すると必ず組合せ論理の内部で信号遷移が発生するため電力を消費する．この消費電力を削減するには，動作する必要がない組合せ論理の入力（オペランド）を固定させ，演算回路内のスイッチング動作を抑えればよい．この手法をオペランドアイソレーションという．具体的には，演算器の入力側に入力固定回路を入れ，演算器を動作させる必要がないときには演算回路の入力値を変化させないようにする．入力固定回路はラッチ，AND ゲート，OR ゲートなどで実現できるが，面積的には AND ゲート，OR ゲートが有利である．

(3) バスエンコーディング

バスのように長い配線を駆動するためには大きなエネルギーが必要となる．そのため，信号遷移が少なくなるようなバスエンコーディング（Bus Encoding）をデータに施すことが電力消費削減に有効となる．例えば，ビット幅 N の二つのデータ d1 と d2 を連続して転送する場合，d1 と d2 のハミング距離が $N/2$ より大きい場合には，d2 の各ビットを反転させて転送した方が，d1 と d2 の間で信号遷移が少なくなるため電力消費は低下する．半導体の微細化が進むと隣接配線間の容量が増大するため，隣接配線間のクロストーク（Crosstalk）の影響は大きくなり，隣接配線で異なる信号を転送する際には遅延時間だけでなく電力消費も増大する．この場合は，隣接配線間ではできるだけ極性が同じ信号を送信できるようにエンコーディングすることが転送電力消費削減に有効であり，例えば各ビットのデータを入れ替える手法などがある．

■参考文献

1) V. Zyuban, P. Kogge, “Optimization of high-performance superscalar architectures for energy efficiency,” Proc. of ISLPED’00, pp.84-89, 2000.
2) M. Kondo and H. Nakamura, “Small, Fast and Low-Power Register File by Bit-Partitioning,” Proc. of HPCA-11, pp.40-49, 2005.
3) I. Park, M.D. Powell, T. N. Vijaykumar, “Reducing register ports for higher speed and lower energy,” Proc. of Micro-35, pp.171-182, 2003.
4) K. Inoue, T. Ishihara, K. Murakami, “Way-predicting set-associative cache for high performance and low energy consumption,” Proc. of ISLPED’99, pp.273-275, 1999.
5) S. Krxiras, Z. Hu, and M. Martonosi, “Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power,” Proc. of ISCA-28, pp.240-251, 2001.
6) K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge, “Drowsy Caches: Simple Techniques for Reducing Leakage Power,” Proc. of ISCA-29, pp.148-157, 2002.
7) M.Kondo, H.Nakamura, “Reducing Memory System Energy by Software-Controlled On-Chip Memory,” IEICE Trans. on Electronics, vol.E86-C, no.4, pp.580-588, 2003.
8) Srilatha Manne, Artur Klauser, and Dirk Grunwald, “Pipeline gating: speculation control for energy reduction,” Proc. of ISCA-25, pp.132-141, 1998.
9) R. Iris Bahar, Srilatha Manne, “Power and energy reduction via pipeline balancing,” Proc. of ISCA-28, pp.218-229, 2001.
10) H.Shimada, H.Ando, and T. Shimada, “Pipeline stage unification: a low-energy consumption technique for future mobile processors,” Proc. of ISLPED’03, pp.326-329, 2003.
11) D. Folegnani, A.Gonzalez, “Energy-effective issue logic,” Proc. of ISCA-28, pp.230-239, 2001.
12) H. Sasaki, M. Kondo, and H. Nakamura, “Energy-Efficient Dynamic Instruction Scheduling Logic through Instruction Grouping,” Proc. of ISLPED-06, pp.43-48, 2006.
13) A. Iyer, D. Marculescu, “Power and Performance Evaluation of Globally Asynchronous Locally Synchronous Processors,” Proc. of ISCA-29, pp.158-168, 2002.
14) D. Ernst, N.S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, “Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation,” Micro-35, pp.7-18, 2003.

3-8-6 論理レベルでの低消費電力化技術

（執筆者：宇佐美公良）［2008 年 5 月］

システム LSI の論理設計では，論理合成（本章 3-3-2 節）が用いられることが多い．論理レベルでダイナミック電力を小さくするような最適化技術が開発され，論理合成ツールに組み込まれている．ダイナミック電力は，全セルに対して取った総和 $P=\Sigma C_i V^2 f \alpha_i$（$C_i$：セル

i の負荷容量，V：電源電圧，f：動作周波数，α_i：セル i のスイッチング確率）で与えられるが，V と f が一定の下で，C と α の積の総和 $\Sigma C_i\alpha_i$ を小さくする手法，あるいは C の総和 ΣC_i を小さくする手法がとられる．

(1)　低電力テクノロジーマッピング

テクノロジーマッピングは元来，論理合成の一部のステップであり（本章 3-3-3 節），半導体ベンダが用意するライブラリに，論理関数をマッピングする操作のことをいう．低電力テクノロジーマッピングでは，各ノードのスイッチング確率をあらかじめ求めておき，スイッチング確率の高いノードがセル内部のノードになるようにマッピングする．**図 3･29** は，もともと 2 入力 NAND で構成された回路網に対し，スイッチング確率が大きいノードは 3 入力 NAND の内部ノードにしてしまうように，マッピングを変えている例である．通常，セル間ノードに比べセル内部のノードは，配線容量を含めた静電容量が小さい．スイッチング確率 α の大きいノードを静電容量 C の小さいノードに割り付けることにより，α の大きいノードを C の大きいノードに割り付ける場合に比べて $\Sigma C_i\alpha_i$ が小さくなるので，ダイナミック電力が低減する．

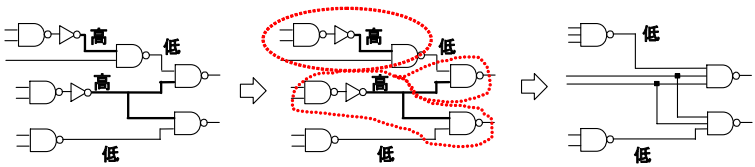

図 3・29　低電力テクノロジーマッピングの例

(2)　低電力ピンスワッピング

論理セルの中には，NAND や NOR など，入力信号を入れ換えても論理が変わらないものがある．こういった論理的に対称な入力ピンに対し，入力容量の大きなピンにはスイッチング確率の小さいネットが接続するよう，接続を入れ換える手法が，ピンスワッピングである．これにより，$\Sigma C_i\alpha_i$ が小さくなるので，ダイナミック電力が低減する．

(3)　ゲートサイジング

回路網には通常，タイミング的に厳しいパス（クリティカルパス）と，タイミング的に厳しくないパス（ノンクリティカルパス）が存在する．また，クリティカルパスは回路網のごく一部であることも，多くの設計例で判明している．クリティカルパス上にないセルに対して，タイミング違反を生じない範囲で，駆動力，すなわちゲートサイズを小さくする技術が，ゲートサイジングである．結果として，ゲート容量が減り，C の総和 ΣC_i を小さくすることができるため，ダイナミック電力が減る．

3-8-7 回路レベルでの低消費電力化技術

（執筆者：宇佐美公良）［2008 年 5 月］

ダイナミック電力は電源電圧 V の 2 乗に比例するため，V を下げる手法がダイナミック電力の低減化には最も効果的である．ところが，電源電圧を下げると MOS トランジスタの性能が低下するため，動作速度が遅くなる．この問題を解決しつつ，ダイナミック電力を低減する技術として，マルチ V_{DD} 技術，及び，電源電圧の動的制御技術がある．一方，リーク電

力の低減技術として，マルチ *Vth* 技術，基板バイアス制御技術，パワーゲーティング技術が開発され実用化されている．更に，超低消費エネルギー要求される分野では，トランジスタ閾値以下の電圧（サブスレショルド領域）で動作させる超低電圧回路の研究が進められている．

(1) マルチ V_{DD} 技術

マルチ V_{DD} 技術は，回路中で高性能が必要な部分にだけ従来の電源電圧を使い，さほど速い動作速度が必要とされない部分には低い電源電圧を使うという方法である．この手法は，低消費電力化に複数の電源電圧を使うので「マルチ V_{DD}」と呼ばれている．一般に，LSI 中で高性能が必要とされる部分は回路全体のごく一部であり，それ以外の部分は低い電源電圧で動作させるので，全体としてダイナミック電力が小さくなる．

マルチ V_{DD} を実装する方法として，①回路ブロック単位（粗粒度）で電源電圧を変える手法と，②ゲート単位（細粒度）で電源電圧を変える手法がある．粗粒度のマルチ V_{DD} の例として，ほとんどの論理ブロックは低電源電圧（0.95V）で動作させる一方，高速動作が要求される一部の回路ブロックとアナログ回路ブロックは高電源電圧（1.2V）で動作させた LSI が報告されている [1]．電源電圧の異なるブロックに信号を送出する場合には，電圧振幅を変換する回路を挿入する必要がある．この変換回路はレベルコンバータ（またはレベルシフタ）と呼ばれる．一方，細粒度のマルチ V_{DD} では，クリティカルパス上にあるゲートを高い電源電圧で動作させ，クリティカルパス上にないゲートは低い電源電圧で動作させる．論理合成を使って生成された回路では，クリティカルパス上のゲートは全体の 10～30%程度といわれており，かなりのゲートを低電圧にすることができる．低電圧で動作するゲートから高電圧で動作するゲートへの接続がある場合，レベルコンバータを挟む必要がある．レベルコンバータは面積や電力のオーバーヘッドとなるので，レベルコンバータの必要箇所を減らす工夫が施される．代表的な方法として，接続しているゲートどうしはできるだけ同じ電源電圧を割り当てるようにして，論理回路を 2 電源化する CAD 技術が開発され，LSI にも適用されている [2, 3]．マルチ V_{DD} 手法は Voltage Island 手法とも呼ばれ，市販 CAD ツールでもサポートされるようになった．

(2) 電源電圧の動的制御技術

CPU や DSP などのプロセッサ LSI では，処理（負荷）の量が動的に変化する．軽い負荷のときには，LSI を最高性能で動作させる必要がないので，電源電圧 V や動作周波数 f を下げることができ，ダイナミック電力が低減する．負荷の量に応じて，電源電圧 V を動的に変える技術を DVS（Dynamic Voltage Scaling）と呼び，電源電圧 V と動作周波数 f の両方を動的に変化させる技術を DVFS（Dynamic Voltage and Frequency Scaling）と呼ぶ. DVFS 技術は 2000 年代初頭，Crusoe をはじめとする CPU で採用されて注目を浴び，その後，携帯機器用 SoC にも適用されるようになった [4, 5]．なお，半導体の微細化が進むにつれ，プロセス，電圧，温度のばらつきがトランジスタの性能に及ぼす影響が無視できなくなっており，ばらつきに応じて電源電圧を制御する AVS（Adaptive Voltage Scaling）技術が考案されている [6]．

(3) マルチ *Vth* 技術

プロセスの微細化とともに, MOS トランジスタのサブスレショルドリーク電流の増大が著しく，リーク電力増大の原因となっている．サブスレショルドリーク電流は *Vth* を高めれば減少するが，トランジスタの性能が低下し遅延時間が増大するという問題がある．そこで，

チップ内で複数の *Vth* を使うマルチ *Vth* 技術が考案された．特に，チップ内のクリティカルパスには低 *Vth* を使い，それ以外の部分には高 *Vth* を使う手法はデュアル *Vth* 技術と呼ばれ，高性能 CPU チップから携帯機器用 LSI まで，非常に広い範囲で使われている．標準的な設計手法としては，高 *Vth* のセルと低 *Vth* のセルの両方をセルライブラリの中に用意しておき，論理合成の実行段階で，クリティカルパス上のゲート（またはフリップフロップ）には低 *Vth* のセル，クリティカルパス以外の部分には高 *Vth* のセルを割り当てる．半導体メーカーも高 *Vth* セルと低 *Vth* セルを用意したセルライブラリを提供しており，また，デュアル *Vth* の設計を可能にする市販の CAD ツールもサポートされている．

(4)　基板バイアス制御

基板バイアス制御技術は，基板バイアスを印加することにより，MOS トランジスタの実効閾値を動的に変える手法である．スタンバイ時に基板に逆バイアスをかけて実効閾値を上げる手法は，Reverse Body Biasing（RBB）と呼ばれる．当初は VTCMOS（Variable Threshold-voltage CMOS）と呼ばれていた [7]．この手法では，製造段階で MOS トランジスタを低い *Vth* でつくっておき，動作時にはこの状態で高速論理動作を実現する．スタンバイ時には，基板バイアスを印加してトランジスタの実効 *Vth*（の絶対値）を大きくし，リークを低減する．動作時とスタンバイ時で実効 *Vth* を変えることにより，高速動作と低スタンバイリーク電流を実現する手法である．

一方で，このような逆バイアスを印加する方式ではなく，順バイアスを印加する Forward Body Biasing（FBB）という手法も研究されている．この方式では，製造段階で高い *Vth* でトランジスタをつくっておき，スタンバイ時にはこの状態で低リークを達成する．動作時には，わずかな順バイアスを印加して実効閾値を低くし，高速論理動作を実現する．FBB 方式には，微細化に対しても効果を維持できるという性質がある．

更に，チップのばらつきに応じて，順バイアス（FBB）と逆バイアス（RBB）を切り替える ABB（Adaptive Body Bias）技術が考案されている．低速チップに対しては，順バイアスを印加し性能を向上させる一方，高速チップに対しては，逆バイアスを印加してリーク電力を低減する [8]．

(5)　パワーゲーティング

論理回路とグランド（または電源）との間に，パワースイッチと呼ぶトランジスタを挿入し，スリープ時にはパワースイッチをオフして電源遮断する手法が，パワーゲーティングである．**図 3･30**(a)のように nMOS をパワースイッチとして挿入するフッタ・スイッチ方式と，図 3･30(b)のように pMOS をパワースイッチとして挿入するヘッダ・スイッチ方式がある．低 Vth で構成された論理回路に対し，高 Vth のトランジスタをパワースイッチとして使う手法は最も典型的な手法であり，とくに MTCMOS（Multiple Threshold-Voltage CMOS）技術と呼ばれる [9]．この手法の特長は，動作時にはパワースイッチをオンさせ低 *Vth* のトランジスタで論理動作を行うので高速である点と，スリープ時には高 *Vth* のトランジスタで論理回路のサブスレショルドリークを遮断できる点である．パワースイッチとして，*Vth* のみならずゲート絶縁膜厚 *tox* も大きいトランジスタを使用することにより，ゲートリークも低減することができる．

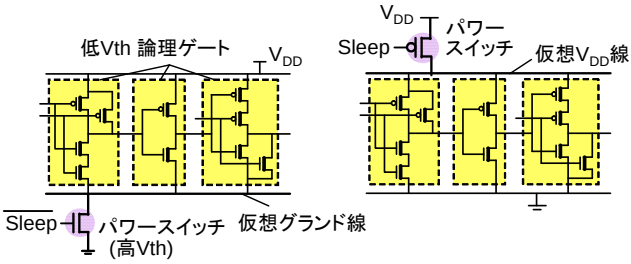

図 3・30　パワーゲーティングの回路構造

ダイナミック電力低減のために電源電圧 V_{DD} を下げる場合には，高 *Vth* のパワースイッチの性能が著しく低下する．このため，フッタ・スイッチ方式で，パワースイッチのゲート電圧を V_{DD} 以上に上げて駆動力を向上させる BGMOS（Boosted Gate MOS）[10]が提案されている．また，パワースイッチに低 *Vth* を使って低電圧での性能低下を抑える一方，スリープ時には負電圧を印加してリークをカットする方式（SCCMOS: Super Cut-off CMOS）[11]も提案されている．

パワーゲーティングの問題点の一つに，電源遮断時に記憶データを保持できないという点がある．そこで，スリープ期間にも保持が必要な記憶回路には，リテンション・フリップフロップと呼ばれる特別な回路を使う．典型的なリテンション・フリップフロップ回路として，D フリップフロップ内部のスレーブラッチの記憶ノードに，高 *Vth* で構成した常時オンのラッチ回路（バルーンラッチと呼ばれる）を接続し，マスターラッチとスレーブラッチが電源遮断されても記憶データを保持する回路が考案されている [12]．

(6)　超低電圧技術

センサネットワークのノードで使われる LSI や RFID タグ，生体へ埋め込むデバイスなどでは，バッテリーの充電や交換が著しく困難なことから，「消費エネルギー」が最大の制約条件となる．CMOS LSI の消費エネルギーが最小となるのは，トランジスタをサブスレショルド領域，すなわち *Vth* 以下の電源電圧で動作させたときであることが，最近の研究で明らかになってきている [13]．180 mV 動作の FFT プロセッサ（0.18 μm CMOS, *Vth*＝450 mV）[14]や，200 mV 動作の 8-bit マイクロプロセッサ（0.13 μm CMOS, *Vth*＝400 mV）[15]の設計事例が報告されている．一方で，こういった超低電圧動作では，V_{DD}, *Vth*, 及び温度のばらつきに対する感度が著しく増大するため [16]，ばらつきに対する対応策が必須となる．

■参考文献

1) J. Carballo, et al, “A Semi-Custom Voltage-Island Technique and Its Application to High-speed Serial Links,” Proc. IEEE International Symposium on Low Power Electronics and Design (ISLPED’03), pp.60-65, Aug. 2003.
2) K. Usami, et al, “Design methodology of ultra low-power MPEG4 codec core exploiting voltage scaling techniques,” Proc. ACM/IEEE Design Automation Conference (DAC’98), pp.483-488, Jun. 1998.
3) R. Puri, et al, “Pushing ASIC Performance in a Power Envelope”, Proc. ACM/IEEE Design Automation Conference (DAC’03), pp.788-793, Jun. 2003.

4) S. Akui, et al, “Dynamic voltage and frequency management for a low-power embedded microprocessor,” 2004 ISSCC Digest of Technical Papers, Feb. 2004.
5) T. Fujiyoshi, et al, “An H.264/MPEG-4 audio/visual codec LSI with module-wise dynamic voltage/frequency scaling,” 2005 ISSCC Digest of Technical Papers, Feb. 2005.
6) A. Drake, et al, “A distributed critical-path timing monitor for a 65nm high-performance microprocessor,” 2007 ISSCC Digest of Technical Papers, Feb. 2007.
7) T. Kuroda, et al, “A 0.9V, 150MHz, 10-mW, 4mm2, 2-D discrete cosine transform core processor with variable threshold-voltage (VT) scheme,” IEEE Journal of Solid-State Circuits, vol.31, no.11, pp.1770-1779, Nov. 1996.
8) J. Tschanz, et al, “Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage,” IEEE Journal of Solid-State Circuits, vol.37, no.11, pp.1396-1402, Nov. 2002.
9) S. Mutoh, et al, “1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS,” IEEE Journal of Solid-State Circuits, vol.30, no.8, pp.847-854, Aug. 1995.
10) T. Inukai, et al, “Boosted gate MOS (BGMOS): device/circuit cooperation scheme to achieve leakage-free giga-scale integration,” Proc. IEEE Custom Integrated Circuits Conference (CICC’00), p.409, May. 2000.
11) H. Kawaguchi, et al, “A CMOS scheme for 0.5V supply voltage with pico-ampere standby current,” 1998 ISSCC Digest of Tech. Papers, Feb. 1998.
12) M. Keating, “Low Power Methodology Manual,” Springer, pp.54-55, 2007.
13) B. Zhai, et al, “Theoretical and practical limits of dynamic voltage scaling,” Proc. ACM/IEEE Design Automation Conference (DAC’04), pp.868-873, 2004.
14) A. Wang and A. Chandrakasan, “A 180 mV FFT processor using subthreshold circuit techniques,” 2004 ISSCC Digest of Tech. Papers, Feb. 2004.
15) B. Zhai, et al, “A 2.60 pJ/inst subthreshold sensor processor for optimal energy efficiency,” 2006 Symp. VLSI Circuits Digest of Tech. Papers, pp. 154-155, 2006.
16) S. Hanson, et al, “Exploring variability and performance in a sub-200-mv processor,” IEEE Journal of Solid-State Circuits, vol.43, no.4, pp.881-891, Apr. 2008.