■10 群（集積回路）－ 1 編（基本構成と設計技術）

1 章 集積回路設計

（執筆者：編幹事団）［2009 年 4 月 受領］

■概要■

この章では，集積回路の設計技術を概観するいくつかの視点を紹介する．それらは，設計工程，設計目標，設計環境，記述言語，及び歴史である．

【本章の構成】

1-1 節“工程による分類”では，集積回路の設計工程を，(1) システム仕様設計，(2) 機能・動作設計，(3) 論理設計，(4) 回路設計，(5) レイアウト設計，(6) マスク設計，及び (7) テスト設計の七つに分類し，それらの概要を述べる．

1-2 節“目標による分類”では，設計目標を，(1) 性能，(2) コスト，及び (3) 柔軟性の三つ指標を用いて分類し，各目標における設計の概要を述べる．

1-3 節“設計環境・Data Base”では，上に示した各設計工程を連携させ，効果的・効率的に実行するために必要となる設計環境及び設計データベースについて述べる．

1-4 節“記述言語”では，設計目標や各種要求仕様などの情報を，設計・製造工程間で伝達・共有するために用いられる記述言語の概要について述べる．

1-5 節“歴史”では，集積回路の設計技術がどのように発展してきたかについて概説する．

■10 群 － 1 編 － 1 章

1-1 工程による分類

（執筆者：宇野　正）［2008 年 4 月 受領］

集積回路設計は，回路規模の増大と設計技術の EDA 化（コンピュータによる自動化）により，対象範囲が拡大している．設計工程は，抽象的な C レベル言語での表現から，RTL（Register Transfer Level），ゲートレベル，Tr レベル，レイアウトレベルへとトップダウン的に具体化される．ここでは設計工程を次の七つに分類して説明する．

(1)　システム仕様設計

対象となる機器システムの仕様を C レベル言語で記述し，機能シミュレータで検証する．システムを実現する最適なアーキテクチャを選択し，ハードウェア（HW）で実現する部分と CPU や DSP の組込みソフトウェア（SW）で実現する部分に，性能・コスト・消費電力を考慮して分割する．そして SW/HW 協調設計検証ツールで確認する．

(2)　機能・動作設計

HW で実現する部分を動作合成ツールで RTL へと変換し，RTL シミュレータで検証する．また，動作合成ツールを用いないで RTL レベルの IP（Intellectual Property）へ割り付ける場合もある．

(3)　論理設計

RTL の段階では，実現する半導体のデザインルール（DR）とは独立である．RTL を論理合成ツールでゲートレベルのネットリストへ変換する．このとき，性能，面積，消費電力のトレードオフを行う．結果は，スタティックタイミング解析ツール（STA）や論理シミュレータで検証する．

(4)　回路設計

回路はディジタル回路とアナログ回路があり，Tr などの素子の組合せで構成する．この回路のタイミング，パワーを回路シミュレータで検証し，ライブラリや IP のかたちにモデル化し，再利用を図る．

(5)　レイアウト設計

レイアウト設計は，製造プロセス工程に対応した素子や配線を構成する具体的な図形（層）を DR に合わせ作図する．手段としてネットリストを配置配線ツールでレイアウトへ自動変換する場合と，半自動レイアウトツールでレイアウトへ変換する場合がある．前者は，ディジタル回路に多用され，後者はアナログ回路に多用される．レイアウト結果から RC 抽出を行い，タイミング，パワーを検証し，また DR 基準を検証する．

(6)　マスク設計

レイアウト結果をウェーハに実装するために，各層ごとにレチクルマスクへ描写する．このとき製造精度を検査する TEG（Test Element Group）やレチクルマスクの自動合せ用のマーク類も同時に描写し，ウェーハ製造・検査の自動化を図る．

(7)　テスト設計

レイアウト設計工程で，製造欠陥を自動検出する回路上の仕組みを設計することをテスト設計と呼ぶ．ロジックの 0/1 縮退を検出するスキャン回路，メモリビット誤りを検出する BIST（Built In Self Test）回路，ボード実装後の LSI を外部制御するバウンダリスキャン回路を設

計し追加する．また ATPG（Auto Test Pattern Generator）で，製造欠陥を検出するテストパターンを生成する．

RTL 以下の設計工程のほとんどは，「設計自動化技術」（EDA: Electronic Design Automation）というコンピュータのソフトウェアプログラムで実行されている．

近い将来，システム仕様設計からマスク設計まで，一括して同時に実行する EDA 技術が実現されようとしているが，最近の超微細化技術に伴って考慮すべき製造上の課題を，設計段階でカバーする設計技術（DFM: Design For Manufacturability）を取り込む必要が出てきている．

現在のところは，上流のシステム仕様設計から RTL 記述までの設計自動化については，機器のシステム設計といかに連携させるかが大きな開発課題になっている．

微細化の進展に伴う DFM 設計技術，システム設計からの設計自動化技術など，設計技術そのものの開発とシステム LSI 製品の設計開発には多くの技術者と長い開発期間が必要になってきているが，応用市場からの価格と開発期間短縮への要求は厳しさを増し，これを解決するために半導体企業間の共同開発や産学連携がグローバルに拡大している．一方，製造マスクセットの価格は急激に高くなってきているので，EDA 技術の進歩によるリスピン回避は必須な状況にあり，設計付加価値はますます増大する傾向にある．

■10 群 － 1 編 － 1 章

1-2 目標による分類

（執筆者：村方正美）［2008 年 9 月 受領］

集積回路の適用分野は多岐に渡り，高速処理が期待されるもの，低消費電力化が期待されるもの，あるいは低コスト化が期待されるものなどがあり，用途・目的に応じた設計を行う必要がある．以下では，目標とする指標を性能，コスト，柔軟性の三つに分類し，それぞれについて概要を述べる．

1-2-1 性能優先設計

代表的な性能指標には，処理速度と消費電力があり，性能優先設計では，これら指標を最適とする設計を行う．処理速度を優先した設計は，タイミングドリブン設計（Timing Driven Design）とも呼ばれる．タイミングドリブン設計は，タイミング制約の設定処理，タイミング最適化処理及びタイミング制約を満足しているか否かの判定処理から構成される．タイミング制約の設定処理では，対象とする回路は一般に同期回路であるため，クロック周波数に応じた遅延時間の制約をフリップフロップ間の論理ゲートと配線の連鎖からなるパスに与える．タイミング最適化処理は，遅延を改善する処理であり，論理ゲート自身の駆動力を上げる，配線負荷を小さくするために配線長を短くするなどの処理を行う．

消費電力を優先した設計は，一般に低消費電力化設計と呼ばれ，設計初期段階での電力見積もりと効果的な電力低減処理の適用がポイントとなる．消費電力には，ダイナミック電力（$P=f\times C\times V^2\times\alpha$，$P$：消費電力，$f$：周波数，$C$：負荷容量，$V$：電圧振幅，$\alpha$：回路の動作頻度）とスタティック電力（主にリーク電流に起因する電力）がある．ダイナミック電力に関しては，動作モードに応じてクロック信号の供給を止めたり（クロックゲーティング），タイミングに余裕のある回路に対しては，タイミング制約を違反しない範囲で低電圧化する（AVS: Adaptive Voltage Scaling）などの手法を適用することで，電力の低減を図る．スタティック電力に関しては，回路動作に影響を与えない範囲で論理ゲートの閾値電圧を上げる，あるいは高い閾値電圧をもつスイッチトランジスタを論理ゲートのグランド側に接続することでリーク電流を遮断する（パワーゲーティング処理），などにより電力の低減を行う．

1-2-2 コスト優先設計

コスト低減のためには，チップサイズを小さくする，チップの設計・製造にかかるコストを低減するために設計期間の短縮を図る，などの方法があり，これらコスト優先設計では，これら指標の最適化を優先的に行う．代表的な設計手法としては，フルカスタム設計手法（Full Custom Design Method），セミカスタム設計手法（Semi-Custom Design Method），プラットフォームベース設計手法（Platform Based Design Method）などがある．

フルカスタム設計手法は，基本的な演算回路や加算器や乗算器などの機能回路を含めて，すべてを一から設計する手法である．この設計手法では，設計期間は長くなるが，設計の自由度が高く，チップサイズを小さくすることが可能である．

これに対して，基本的な演算回路はあらかじめライブラリとして用意しておき，これらライブラリを使って設計を行う方法として，セミカスタム設計手法がある．セミカスタム設計

手法には，ゲートアレイ方式（Gate Array）とスタンダードセル方式（Standard Cell）の 2 種類がある．ゲートアレイ方式は，基本論理回路（ゲート回路）をあらかじめチップ上に規則的に配置（製造）しておき，ライブラリにはこれらゲート回路間の接続情報を登録し，回路接続情報を元にチップ上のゲート回路にライブラリを割り当て（配置），これらの間の配線を行う．このように，ゲートアレイ方式では，配線層の製造工程だけで済むため，製造期間が短く，製造コスト的には有利だが，標準的な論理ゲートの組合せで回路を構成するため，一般に集積度は不利となる．これに対して，スタンダードセル方式は，トランジスタから構成される基本演算回路をあらかじめ作り込んだライブラリとして登録しておき，これらライブラリやほかの設計済みの機能ブロックをチップ上に配置し，これらの間を配線して設計を行う．そのため，一般に集積度・性能ともゲートアレイ方式よりも有利だが，下地から作るため製造期間が長く，製造コスト的に不利である．

プラットフォームベース設計は，ある程度のアーキテクチャや CPU，BUS の形式をあらかじめ決めておき，その上にハードウェア IP や各種ソフトを追加・変更して設計する方式である．このようにあらかじめ用意された IP や BUS などを活用するため，効率よく設計でき，設計期間の短縮が期待できる．

1-2-3 柔軟性優先設計

通常の集積回路は，仕様変更／機能変更による動作中の回路変更や，製造後の回路構成の変更は一般に難しい．これに対して，プログラマブルロジックデバイス（Programmable Logic Device）は，ユーザが手許で任意の回路を書き込むことができるだけでなく，何度でも書き直し，再利用ができるなど，設計に柔軟性をもつことができる．また，動作中に回路機能の変更，処理速度や消費電力など性能指標の見直しなど，回路を定義し直すことができるものとして，リコンフィギュラブルデバイス（Reconfigurable Device，再構成可能型集積回路）と呼ばれるものがある．リコンフィギュラブルデバイスでは，チップ面積・コスト・消費電力・処理速度の各指標に対して，どの性能指標を優先するのかの選択の自由度が大きく，柔軟に対応可能である．

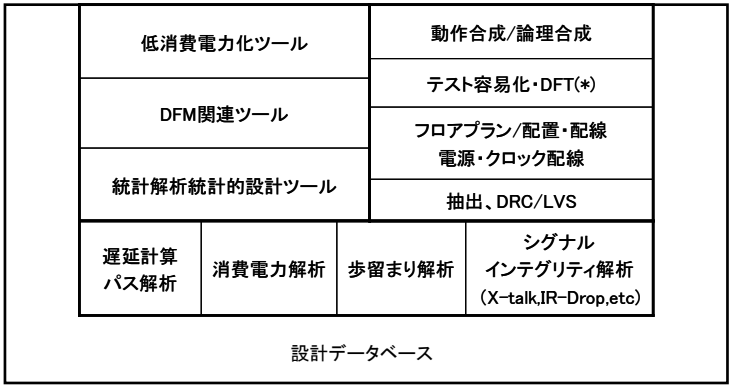
■10 群 － 1 編 － 1 章

1-3 設計環境・Data Base

（執筆者：村方正美）［2008 年 9 月 受領］

1-3-1 設計環境

集積回路の設計環境は，各設計工程に必要な各種設計・検証・解析・最適化ツール，ライブラリ情報，回路接続情報及び設計の各工程で生成される各種設計情報などを一元管理する設計データベースなどから構成される．設計環境として備えるべきツールの例としては，次のようなものがある．すなわち，高位・機能・論理設計では，動作合成／論理合成ツール，テスト容易化設計ツール，論理等価性検証／タイミング解析／消費電力解析ツールなどを備える必要がある．レイアウト設計では，フロアプラン，配置・配線，電源・クロック配線，RC 抽出，タイミング／クロストーク／IR-Drop 解析，低消費電力化設計ツール，製造性考慮設計ツール及びばらつき考慮設計ツールなどである．**図 1･1** に集積回路の設計環境の例を示す．



＊DFT；Design For Test

図 1・1　設計環境の例

1-3-2 Data Base

データベースとは，データを一定の規則に従って格納し，一元的に管理できるようにしたもののことである．このため，特定のデータの参照や利用を効率よく行うことができる．データベースの中でも，一つのデータをカラム（列）とレコード（行）によって整理し，テーブル（表）の中に配置した形式のデータベースはリレーショナルデータベース（RDB: Relational Data Base）と呼ばれ，近年で主流のデータベース形式となっている．リレーショナルデータベースのほかにも，データとデータへの手続きを一体化して（オブジェクトとして）扱うオブジェクトデータベース（Object Data Base）や，XML（Extensible Markup Language）文書を格納しデータベースとして活用する XML データベースなどがある．データベース内のデータの操作や保守管理は，データベース管理システム（DBMS: Data Base Management

System）によって行われる．リレーショナルデータベースの場合は特にリレーショナルデータベース管理システム（RDBMS: Relational Data Base Management System）と呼ばれている [1].

■参考文献

1) IT 用語辞典バイナリ：http://www.sophia-it.com/content/

■10 群 － 1 編 － 1 章

1-4 記述言語

（執筆者：今井正紀）［2008 年 4 月 受領］

EDA（Electronic Design Automation）ツールを活用し，プロジェクトとして複数の設計者により効率的な設計を行うためには異なるツール間，ツールサポートを得るための設計者－ツール間，及び設計者間で設計目標，意図，制約，機能仕様，実装仕様，その他各種要求仕様などの情報を伝達，共有する必要があり，そのための言語が各種の記述言語である．

システムを物理的に実現するフローは，設計，機能検証，性能検証，実装，テストという設計ステージからなるが，各ステージにおいてその処理に関連する情報の表現に向いた仕様，構文をもった記述言語，フォーマット（形式）が考えられている．設計の各ステージごとに関連する記述言語について主なものについて述べる．

1-4-1 設計，機能検証

(1) システムレベル設計記述言語

システムの仕様を表現する言語として UML（Unified Modeling Language），C 言語がある．UML は OMG（Object Management Group）によって管理されており，2007 年 8 月現在の OMG 標準は UML 2.0 である．これはもともとソフトウェア開発のためのモデリング言語であるが，2004 年頃からハードウェア開発への適用が検討されるようになった．

一般にシステムはマルチドメイン（電気系，機械系など，及びソフトウェア）のものである．また各ドメインにおいて信号の性質からマルチシグナル（アナログ，ディジタル，ミックスシグナル）である．

システムをハードウェア，ソフトウェアとして実現するためのシステムの要求仕様定義の前にアルゴリズムの検証が必要である． そこに用いられる言語に Matlab がある． これによりシステムを実装に依存しないかたちで数学的にモデリングし，シミュレーションによりアルゴリズムの検証を行うことができる．このマルチドメインのシステムのうちソフトウェアを含めた電気系の設計階層が ESL（Electronic System Level）と呼ばれるものである．まず，アルゴリズム検証後にシステム実現のために ESL での実行可能仕様を定義する．ここで用いられる言語に SystemC（IEEE std 1666-2005）などがある．C++の基本構文，基本クラスライブラリをベースに，C++のオブジェクト指向言語としての機能を用いてハードウェア設計へ適用できるように並列化動作と，それに向いたデータタイプなどの記述を容易化するクラスライブラリを加えたものである．ほかの言語として SpecC, Esterel, Handel-C, BachC がある．これに基づきコスト面，性能面（速度，消費電力），信頼性面などから仕様をハードウェアで実現するかソフトウェアで実現するかの分割であるハードウェア，ソフトウェア分割，その前後での協調検証が行われる．同時にハードウェアアーキテクチャとしてどの構成をとるべきかのアーキテクチャ探索が行われ，候補となるアーキテクチャの中から制約の面から適切なものが選択される．これにはアーキテクチャのリファインメントも伴う．ここでモデルとはシステム，あるいはサブシステムの注目した機能を記述したものあり，各モデルはそれぞれ固有の属性とメソッドをもつものである．オブジェクト指向におけるクラス変数とメソッドにより自然に表現できる．

(2) アナログ，ミックスシグナル回路記述言語

アナログとディジタルのミックスシグナルを扱う言語としては，それぞれ Verilog HDL（IEEE std 1364-2005），VHDL（Very High Speed Integrated Circuit Hardware Description Language, IEEE std 1076-1993）を拡張した Verilog-AMS，VHDL-AMS（IEEE 1076.1）がある．Verilog-AMS のアナログ仕様を Verilog-A，このときディジタル仕様部を Verilog-D と呼ぶ．また Verilog-AMS との整合化のため，Verilog HDL の後継言語である SystemVerilog においても，SystemVerilog-AMS の検討が Accellera の場で行われている．将来的にはこれらの統合も検討されている．

(3) RT レベル(RTL)設計記述言語

SoC（System on a Chip）などのシステムは，3rd パーティのデザイン IP（あるいは単に IP，Intellectual Property）などが多数搭載されたものであり，この段階での検証は各 3rd パーティ IP，ユーザ IP の記述言語により，言語が混在したマルチランゲージの検証となる．マルチランゲージには二つの側面がある．一つは設計階層の抽象度の違いからくるマルチランゲージであり，ほかの一つはアナログ，ディジタルという信号の違いからくるマルチランゲージである．

検証後，TL（Transaction Level）にあるブロックは論理合成可能な RTL（Register Transfer Level）記述（Verilog HDL（IEEE std 1364-1995，2001，2005），VHDL（IEEE std 1076-1987，1993，2000，2002，2008），SystemVerilog（IEEE std 1800-2005）が用いられる）と呼ばれる記述に書き換えられることによって詳細化される． これらの言語は総称して HDL（Hardware Description Language）と呼ばれる．この変換のプロセスは高位合成（High Level Synthesis）あるいは動作合成（Behavioral Synthesis）と呼ばれる．デザイン内の各処理のクロックサイクルへの割当てなどがこの過程で行われる．一般に合成とはこのように段階的に構造的な詳細情報を付加していくプロセスのことである．論理合成可能な RTL 記述はこのように動作合成の結果として，あるいは TL 記述の人手による書換えなどで得られる．

(4) アサーション， プロパティ， 検証用モデル記述言語

機能の検証にはダイナミックな手法とスタティックな手法があり，ダイナミック検証ではその適用のためにテストベンチを構築する必要がある．テストベンチを開発するための言語としては SystemC，SystemVerilog，e（IEEE std 1647-2006），OpenVera，Verilog HDL，VHDL などがある．これらの言語により DUV（Design Under Verification），あるいは DUT（Design Under Test）を検証するためのテストベンチ構成要素であるジェネレータ，トランザクタ，チェッカなどが記述される．テストベンチに依存したダイナミック検証の可観測性，可制御性を向上するためにアサーションの挿入が行われる．記述の生産性を向上させる時相論理をサポートする言語が考えられており， アサーション言語と呼ばれる． アサーションとは設計の意図であり検証の目的となっている表明である．このための言語には PSL（Property Specification Language:IEEE std 1850-2005），SVA（SystemVerilog Assertion），OVA（Open Vera Assertion），e（Temporal e）などがある．アサーションはそれがスタティックツールの検証の目的となる場合には伝統的にプロパティと呼ばれることがある．プロパティ記述に適した言語として PSL，SVA がある．SystemC，SystemVerilog，PSL はすべて 2005 年に IEEE 標準として承認された．PSL，Vera，e など検証専用の言語は総称して HVL（Hardware Verification Language）と呼ばれる．これに対してアサーションや検証 IP をテストベンチで利用するため

の API（Application Programming Interface）群からなる OVL（Open Verification Library）というものがある．API を用いて呼び出されるライブラリの実装は Verilog HDL，PSL，System Verilog などを用いて行われている．機能検証の網羅性はカバレッジと呼ばれる指標で表され，これを計測するための外部モデルをカバレッジモデルという．System Verilog や Verilog と PSL などの組合せで構成される．またスタティック検証ツールやダイナミック検証ツールなどの異なるツールが定義するカバレッジ指標に対して尺度の整合化が Accellera で検討されており，その記述フォームは UCIS（Unified Coverage Interoperability Standard）と呼ばれる．そのほか，ダイナミック検証に関係する標準として論理シミュレーションモデルがある．VHDL 用のそのようなモデル標準として VITAL（VHDL Initiative Towards ASIC Libraries, VITAL-2000 or VHDL 1076.4）がある．

表 1・1　記述言語（設計，検証）

抽象レベル			言語，ライブラリ	
System Level			UML, Matlab, C, OpenMAST	UML は主にソフトウェア仕様向き
ESL			SystemC, SpecC, Bach-C, Handel-C, Metropolis (meta model), Esterel, System Verilog	SytemC で RTL も記述可能，つまり言語が抽象レベルに固定的に対応しているわけではない．逆に SystemVerilog で ESL レベルの記述も可能である．
ミックスシグナル			Verilog-AMS, VHDL-AMS	
	ディジタル		Verilog-D	Verilog-A と区別する意味で，-D を付けて呼ぶ場合がある．
	アナログ		Verilog-A	
RTL	設計		Verilog HDL,VHDL,SystemVerilog	SystemVerilog は設計，検証を扱う．その意味で HDVL とも分類される．
		性能目標	SDC	目標に応じた合成制約記述
	検証	assertion	SVA, PSL, e, OVA, OVL,（VHDL）	OVL は，Verilog HDL，VHDL， System Verilog，PSL を実体言語とするチェッカのライブラリ
		coverage	UCIS	異種ツール間の整合的カバレッジ仕様．
		Testbench (transactor ,etc)	Verilog HDL, VHDL, e, SystemC, OpenVera	ホストとデバイスの接続検証で用いる対向モデル c どの記述用．
		simulation model	VITAL	VHDL 論理シミュレータ用モデル
Gate レベル（論理）			Verilog HDL,VHDL	ネットリスト
Tr.レベル			SPICE	BSIM3, 4，HISIM, PSP は，C，Verilog-A などを言語として記述するトランジスタモデル（SPICE ネットの Tr.の動作モデル）

1-4-2 実装，性能検証

(1) 性能仕様表現（遅延，電力）

ライブラリセルの遅延，消費電力などの特性の記述は.lib（あるいは liberty）フォーマットが業界標準的に用いられている．最近では微細化対応で電圧などのばらつきに依存する特性（雑音，電力，遅延）を表現するために.lib の拡張が考えられている．ばらつきに対応した拡張フォーマットには CCS（Composite Current Source）や ECSM（effective current source model）などがある．このようなネットリストに基づき配置配線を行うことによって配置位置，配線長などが確定する．遅延モデルに基づいてこれらの長さなどの物理次元をもつ実装依存情報は，遅延値に変換される．遅延情報は SDF（Standard Delay Format: IEEE std 1497-2001）を用いてネットリストを構成する各ネットにバックアノテートされ，具体的な遅延値に基づいた STA（Static Timing Analysis）などによるタイミング解析により利用される．

遅延と並んで性能検証のほかの目標である消費電力についても制約を記述するための標準フォーマットが UPF（Unified Power Format, IEEE std 1801-2009）がある．そのほかに Si2（Silicon Integration Initiative）で検討されている CPF（Common Power Format）がある．現在 CPF 1.0, 1.1 が策定されており，パーサと共に Si2 から公開されている．遅延計算のための記述言語も IEEE（1481-1999）で標準化されており DCL（Delay Calculation Language）と呼ばれ，DPCS（Delay&Power Calculation System:IEEE std 1481-1999）の中で用いられる．寄生容量などの物理量が遅延計算システムにより遅延時間に変換されるが，元となる RC を記述するフォーマットには SPEF（Standard Parasitic Exchange Format）がある．ほかに DSPF（Detailed Standard Parasitic Format），及び RSPF（Reduced Standard Parasitic Format）がある．

(2) レイアウト記述，接続記述

機能の正しさを確認した合成可能 RTL 記述は論理合成ツールによってゲートレベルの記述（ネットリスト）に変換される．最適化目的に応じた，合成時に与える制約を記述するフォーマットに SDC（Synopsys Design Constraints）がある．そのほか，設計制約を与えるための言語に DCDL（Design Constraints Description Language）がある．ゲートレベル記述に使われる言語としては Verilog HDL, VHDL, EDIF（Electronic Design Interchange Format）などが用いられる．合成された結果としてネットリストを構成するゲートはターゲットとなるプロセステクノロジー，デザインルールでのライブラリのセル（特定の駆動能力，容量，抵抗，処理スピードなどが定義）にテクノロジーマッピングされる．ライブラリセルの実装設計の観点からの必要な情報を表現する形式として LEF（Library Exchange Format）がある．これにはセルの大きさやオリエンテーション，レイヤ数やレイヤごとのデザインルールやキャパシタンスなどの物理情報が記述される．またデザインに対しては配置配線に対するフロアプランなどの指定情報や配置結果，接続情報などのレイアウトデータを表現するフォーマットとしては PDEF（Physical Design Exchange Format），DEF（Design Exchange Format）がある．

マスクデータの元となる最終的な実装情報である図形データを表現するものとして GDSII フォーマットが用いられている．ただ設計されるシステム（SoC）の大規模化によって，GDSII のファイルサイズが非常に大きくなりその読み書きなどにも時間を要するなど問題点が指摘されるようになっている．対応として OASIS（Open Artwork System Interchange Standard）というフォーマットがある．SEMI（Semiconductor Equipment and Materials International）によって 2003 年に標準として採用された． 圧縮を施さない場合でも圧縮した GDSII に比べて数分

の 1～10 分の 1 になるといわれている．移行を進めるにはファンドリ， 実装，実装検証の分野間での整合化が必要となる．そのほか，実装設計のみに関係するものではないが，XML（eXtensible Markup Language）を利用した，SoC への IP のツールによるインテグレーション容易化のために，IP のデリバラブルズ（シミュレーションモデルなどの各種ツールモデル）を記述するフォーマットに IP-XACT（IP-XACT 1.4）ある．

表 1・2　記述言語（実装，性能検証，テスト）

用途		フォーマット	備考
レイアウト用接続		EDIF	
バックアノテーション		SDF	DPCS
属性パラメータ	遅延	.lib, CCS, ECSM	CCS, ECSM はばらつき対応の拡張仕様
	電力	CPF, UPF	Si2, Accellera のそれぞれで検討されている.
	容量	SPEF, DSPF, RSPF	
レイアウト結果		LEF, DEF, PDEF	
図形データ		GDSII, OASIS	OASIS ではデータサイズの削減可能
テスト		STIL	テストの意図記述，テストデータの記述性
		CTL, OCI	コアテスト，テストデータ圧縮などの拡張仕様
データベース		Open Access	共通 API によるアクセス（XML データベース）

1-4-3 製造テスト

LSI の設計フローはマスクデータを出力とするものとテストデータを出力するものの二つに分けられる．テストデータの設計フローにおいても製造テストに必要な情報を記述するフォーマットには多くのものがある．半導体メーカーと EDA ツールのインタフェース及びこれらとテスタ間のインタフェースにそれぞれ記述フォーマットがある．このためにリソースの非効率化が指摘されてきた．回路規模増大によるテストデータの大規模化への対応と併せてこれら各インタフェース共通化による効率化のために取り組まれているのが STIL（Standard Test Interface Language: IEEE std 1450-1999）である．STIL は ATE（Automatic Test Equipment）に依存せずテストの意図を記述し，単純なものから複雑なテストデータまで表現できる記述性と ATPG（Automatic Test Pattern Generation），BIST（Built-in Self Test），及び ATE 間の高密度のテストデータをやりとりできるように設計されている．従来，テストの意図は各 ATE ごとに VCD（Value Change Dump）で記述されたテストデータをプログラムなどを用いて大きな手間をかけて変換していた．VCD には実際のデバイス上で解析したいイベント間の関係を記述する機能が欠けていたためである．テストデータ記述フォーマットには VCD を拡張した EVCD（Extended VCD）や WGL（Waveform Generation Language）があるが STIL に標準化されていくとみられる．またコアのテスト用に STIL の拡張仕様として CTL（Core Test Language）があり，IEEE 標準（IEEE1450.6-2005）として承認されている．拡張仕様にはほかにテストデータの圧縮フォーマットの仕様である OCI（Open Compression Interface）が

ある．プリント基板上の回路の動作をシステムレベルでシミュレーションするために IBIS（I/O Buffer Information Specification），デバイス内部の本来のコアロジックと外部ピン間にプローブと等価な働きをするセルと呼ばれるレジスタを配置し，これを結合してシフトレジスタを構成する JTAG（Joint Test Action Group:IEEE 1149.1）を拡張したバウンダリスキャンの実装の共通言語提供を目指した BSDL（Boundary Scan Description Language: IEEE std 1532）がある．

1-4-4 計算，通信精度表現

(1) 計算，通信タイミング精度表現

システムを構成する各モデルの抽象レベルとしては，まず因果関係のみを制約とし，機能の処理にかかる時間を 0 としたモデルである UTF（Un-Timed Functional）モデルがあり，アルゴリズムの検証に用いられる．一方，処理にかかる時間として近似的に見積もった時間を想定したモデルが ATF（Approximately-Timed Functional）モデルでありシステム仕様のハードウェアとソフトウェアへの分割における評価に用いられる．続いてハードウェアについてそのマイクロアーキテクチャの評価のために機能の処理にかかる時間を見積もったモデルが TF（Timed Functional）モデルである．更にバスを介しての送受信手順の単位がバスサイクルであり，その精度で定義されるモデルが BCA（Bus Cycle Accurate）モデルである．計算，時間精度のどちらかが CA（Cycle Accurate）になるまでシステム仕様から実装へと順に詳細化していく過程で扱われる．このレベルは総称して TL（Transaction Level）と呼ばれる．計算，時間精度のどちらもが CA であるようなモデルが CA モデルである．RTL（Register Transfer Level）に相当する．SytemC による RTL の記述も可能であるが，通常 SystemC で扱われるレベルは TL である．特にこの階層で行われるブロックのモデリングを TLM（Transaction Level Modeling）という．2009 年 7 月現在，TLM 2.0 LRM（Language Reference Manual）が OSCI（Open SystemC Initiative）の最新のものとなっている．また SystemVerilog も仕様として TLM に対応可能な言語である．TLM ではタイミング精度をそれほど落とすことなしに高速に機能検証が可能なように考えられたモデリング手法である．ここでトランザクションとはシミュレーションされているモデルの間での，読出し，書込みトランザクションなどの RTL より抽象度の高いイベントやデータの交換であり，目標となるアプリケーションに応じて，必要なだけ抽象化することでシミュレーションの高速化が図られる．

(2) 通信の粒度表現

通信される粒度には，トランザクション，バスパケット，ワード，ビットがある．また通信タイミング精度とそれに適した通信の粒度には，ほぼ相関がある．トランザクションには ATF，バスパケットには BCA，バス幅で定義されるワードの粒度には CA が対応する．各信号線に対応するビットの粒度でも同期設計では CA が対応する．

■参考文献

1) Accellera home: http://www.accellera.org/home/
2) OSCI home: http://www.systemc.org/home
3) PSL: http://www.eda.stds.org/ieee-1850/Issues/issues.html
4) http://www.eda.org/ieee-1850/ieee-1850-issues/hm/
5) UML（Unified Modeling Language）: http://www.uml.org/

6) Spec C Technology Open Consortium: http://www.specc.org/
7) Si2 Home: http://www.si2.org/openeda.si2.org/
8) Andreas Gerstlauer, Rainer Domer, Junyu Peng, and, Daniel D. Gajski（共著），木下常雄（訳），“システム設計 SpecC による実現,”SpecC Technology Open Consortium，2001.
9) （株）半導体理工学研究センター，“RTL 設計スタイルガイド,”Verilog HDL 編，第 2 版, 2006.
10) 同，“RTL 設計スタイルガイド,”VHDL 編，初版，2003.
11) 同，“TL モデリングガイド,”第 2 版，2008.
12) Janick Bergeon, Eduard Cerny, Alan Hunter, and Andrew Nightingale, “Verification Methodology Manual for SystemVerilog,” Spriger, 2005.
13) Mentor Graphics, “The Verification Cookbook － Advanced Verification Methodology SystemC and SystemVerilog(3rd Edition),” 2007.
14) Lucai Cai and Daniel Gajski, “Transaction Level Modeling: An Overview,” Proceedings of the International Conference on Hardware/Software Codesign & System Synthesis, Newport Beach, CA, Oct. 2003.
15) Metropolis: http://embedded.Eecs.berkeley.edu/metropolis/index.html

■10 群 － 1 編 － 1 章

1-5 歴　史

（執筆者：吉田憲司）［2008 年 8 月 受領］

1958 年に Jack Kilby によって集積回路が発明されて以来，半世紀経った今も，その機能の複雑多様化，高速化など，その進歩は続いている．これは主として寸法の微細化と高集積化によるものであるが，それを実現するための設計技術の進歩があったからこそ，これが可能であったといえる．

1-5-1 集積回路の進歩

Intel の Gordon Moor は 1965 年に，経済的に製造できる集積回路の規模（トランジスタ数）は，約 2 年に 2 倍の割合で指数関数的に増大すると指摘し（Moore's Law と呼ばれる），その傾向は現在に至るまで続いている（**図 1･2**）．これに伴い，処理性能や記憶容量，消費電力など関連する各種パラメータの値も同様に，指数関数的に増加している．これらのパラメータの値の増加は設計の困難さを指数関数的に増すが，そのために，自動設計技術の改良や新しい設計技術の導入が促進された．

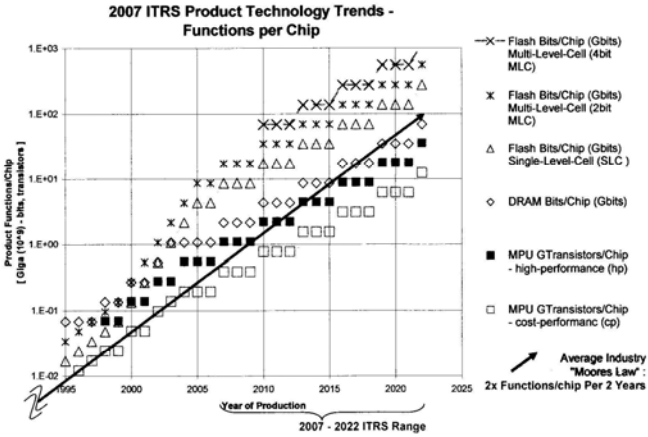


Figure 9 2007 ITRS Product Technology Trends:
Product Functions/Chip and Industry Average "Moore's Law" Trends

図 1･2　2007 ITRS Product Trends と Moore's Law（2007ITRS Executive Summary Figure 9）

この Moore's Law がいつまで続くのかについては，消費電力（発熱）が大きな制限要素となる，あるいはトランジスタのゲート長が 10nm 以下になると物理的限界がくる，などの議論もあり，ITRS（International Technology Roadmap of Semiconductor）でも議論されている．ITRS は半導体の製造技術及び設計技術のトレンドについて予測する国際組織で，毎年予測が見直しされ，業界で共有されている．

また，集積回路の品種数に関する法則として著名なものとして，Makimoto’s Wave と呼ばれるものがある（**図 1･3**）．これは集積回路の「カスタム化」と「標準化」が約十年のサイクルで入れ変わるということを指摘したもので，設計技術の進歩により可能になる「カスタム化」（少量多品種）と大量生産の経済性を重視した「標準化」（少品種大量生産）とが，振り子のように周期的に入れ替わるというものである．

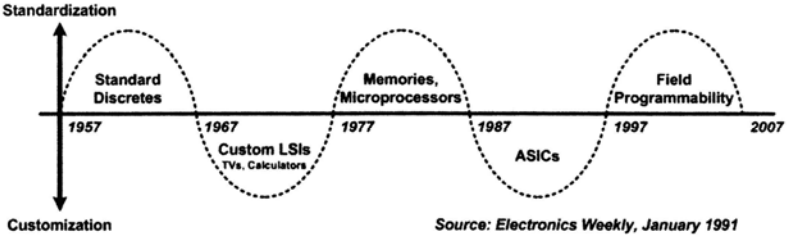


図 1・3　Makimoto’s Wave（Electrnics Weekly, January 1991）

1-5-2 設計自動化の進歩

上記 Moore’s Law に従って集積回路規模が増大するためには，誤りなく迅速に設計するための設計自動化技術の進歩が不可欠であった．その歴史は，絶えず増大する設計作業を合理化するために，

（1）人手設計入力の合理化から自動設計（合成，最適化）へ
（2）シミュレーションによる動作確認からルールなどによる設計検証の自動化，高速化へ
（3）設計の下流工程の合理化から上流工程へ，より抽象的レベルへ
（4）設計 IP（回路ブロック）の再利用からその標準化へ

の方向で技術が進歩したものであった．また，指数関数的に増大するデータ量を処理するために，計算機科学（アルゴリズム）の進歩と計算機ハードウェアの進歩（専用ハードウェア含む）が不可欠であったことはいうまでもない．

集積回路開発に実際に活用された設計自動化技術の代表的なものは以下のとおりである．

マスク設計：設計の自動化はマスクパターンの自動製図機用データ作成とその検証から始まった（1970 年代後半）．その後マスク及びその製作法の進化につれて変遷し，最近は困難さを増すリソグラフィ技術に対応するため，RET（Resolution Enhancement Technology）や OPC（Optical Proximity Correction）などの処理も併せて行われる．

レイアウト自動設計：標準化したセルを配置，配線するスタンダードセル方式レイアウトの自動設計は 1970 年代から使用が始まり，1980 年代にはゲートアレイ方式の自動レイアウト設計に適用され，ASIC ビジネスと EDA ベンダーの発展に寄与した．当初は 100%配線完了とチップ面積最適化が目的であったが，その後微細化の進展とともに，配線遅延，消費電力，製造性（Manufacturability）なども同時に考慮した設計最適化が必要になっている．

回路シミュレーション：アナログの回路解析（シミュレーション）は最も古く（1970 年代）から使われていた設計ツールであるが，1980 年代後半にはトランジスタモデルによる大規模（100 素子）な回路シミュレーションが可能になり，更にマクロなモデルによるより大規模高速な回路のシミュレーションが可能になった．微細化の進展とともに，寄生パラメータや SI（Signal Integrity），更にはパラメータのばらつきをも考慮して精度良くシミュレーションすることが要求されている．

論理シミュレーション：計算機の開発のために古く（1970 年代）から使われていたものが，1980 年代には素子遅延，あるいは配線遅延が取り扱えるように改良されて論理集積回路の設計に広く使われるようになった．1990 年前後からは機能記述や動作記述などより高位レベルの記述に対する機能シミュレーションも一般的になった．

論理合成：RTL（Register Transfer Level）のハードウェア記述から論理回路ネットリストを自動合成するツールが 1990 年代から，特に ASIC の設計で実用的に使用されるようになった．当初は最小の素子数で論理機能を実現することが目標であったが，配線遅延，消費電力も同時に最小化し，更には製造性をも考慮することが必要となっている．

形式的検証：シミュレーションによらないシステムの検証手法として，形式的検証が研究されているが，その例として，抽象レベルの異なる設計データの等価性を検証する手法（形式的等価判定）が近年（2000 年代）市販され，実用化されている．

高位合成：RTL より高位レベルのアルゴリズムや動作記述を RTL 記述に変換する高位合成，あるいは動作合成，が最近（2000 年代）使用されるようになった．

テスト設計：論理回路のテストパターン自動生成としては，IBM の D アルゴリズムが 1960 年代に発表され，その後，PODEM, FAN など高速のアルゴリズムが実用化された．故障モデルに関しては縮退故障がまず提案され，最近では遅延故障のテスト生成が重要となっている．また，大規模な回路に対して高い故障検出率でテスト集合を自動生成するためのテスト容易化設計手法としてスキャン設計方式が考案され，現在も広く使われている．今後は，SoC（System-on-Chip）を対象とし，フィールドテストも考慮したディペンダブル VLSI（Dependable VLSI）設計が重要になる．