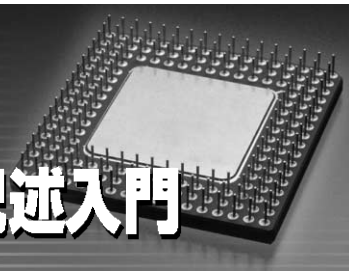
第1章 やさしいVerilog HDL記述入門



本章では，まずVerilog HDLで記述し，シミュレーションするところから始めます．細かい文法のことは後回しにして，記述例をお見せします．「加算回路」と「カウンタ」を例に，Verilog HDLの記述スタイルを紹介します．Verilog HDLによるシステム記述の基本構造がどのようになっているかを理解してください．

1.1 HDLって何だ？

1.1.1 HDL記述と論理合成

● HDLとは？

HDL(hardware description language；ハードウェア記述言語)は，文字どおりハードウェアを記述するための言語です．ここでいうハードウェアとは，おもに論理回路です．OPアンプやトランジスタなどを扱うアナログ回路ではありません．HDL自体は，とくに新しい概念ではありません．以前から研究や実用化がなされてきました．しかし，近年にわかに脚光を浴びているのは，

●論理合成ツールが実用レベルに達した
●ワークステーション，パーソナル・コンピュータの低価格化，高性能化

などの理由によります．そして，HDLが注目される最大の理由は，

●大規模論理回路を短期間で設計する必要が生じた

からです．

半導体技術の向上やコスト低下により，大規模論理回路を実現することが可能となりました．技術の向上に合わせて，市場がより高性能，低価格の商品を要求するようになりました．そして，主要部品である論理回路ICをより大規模化することが要求に対する答えとなったのです．しかし，従来の「回路図による論理回路設計」には限界が見えはじめてきました．より効率的な設計手法が必要とされたのです．そして，回路図に代わる設計手法がHDL設計なのです．

● HDLの種類

HDLには，**表1.1**に示すような多数の種類があります．これらの言語の中でもっともよく用いられ

表1.1 HDLの種類

種 類	特 徴
Verilog HDL	C言語に似た文法体系．ASIC開発においては，ライブラリの充実，採用実績多数などの理由で，実質的な業界標準．1995年にIEEE 1364として標準化された．
VHDL	米国国防省を中心に，いち早く標準化(IEEE 1076)されたHDL．言語仕様が豊富でかつ厳格．
UDL/I	日本電子工業振興協会(電子協)の標準化委員会で採択された純国産の標準HDL．実用的な処理系(シミュレータ，論理合成ツール)がないため，利用されていない．
SFL	論理合成ツールPARTHENON(NTT)用のHDL．同期回路に用途を限定したため，記述や処理系が簡潔．いくつかのASIC開発実績はある．

ているのが，“**Verilog HDL**”と“**VHDL**”です．この両言語は，ことあるごとに比較されます．そしてさまざまな立場(EDAベンダ，半導体ベンダ，教育関係者，社内設計ツール支援部門，設計マネージャ，設計者)によって，意見の相違が見られます．

そこで，実際に使う設計者の立場に立って(著者の主観も一部含めて)Verilog HDLのVHDLに対する優位性を示します．

(1) C言語をベースにした文法体系であり，記述が簡潔．
(2) 構文や演算子がC言語とほぼ同じなので，類推によって記述でき，習得が容易．
(3) シミュレーション用言語として誕生した経緯もあって，シミュレーション向けの記述力が充実している．例えば，従来，シミュレータ環境の中で行ってきたこと(プローブを立てて信号観測/信号印加，ROM/RAMデータのファイル・アクセスなど)が記述の中でできる．
(4) 言語体系が簡素なためシミュレータが高速．さらに，VHDLのようなパッケージ・ファイル(データ・タイプや演算をすべて定義したファイル)が不要なため，シミュレータの負担が軽い．
(5) ASICの開発実績が多数ある．このため，シミュレーション用のライブラリやツール類がVHDLに比べて充実している．

● 論理合成とは？

現状の論理合成ツールは，HDLで記述された論理機能を実際のゲート回路に変換します．入力するHDLは，論理式や真理値表で表現したものや，条件分岐や繰り返し構造などにより動作表現したものです．論理合成による出力は，目的のASICやFPGA用の**ネットリスト**です．「ネットリスト」とは，「回路部品(セル)の接続関係をテキストで表現したもの」です．

HDLによってASICやFPGAを設計する手順を**図1.1**に示します．

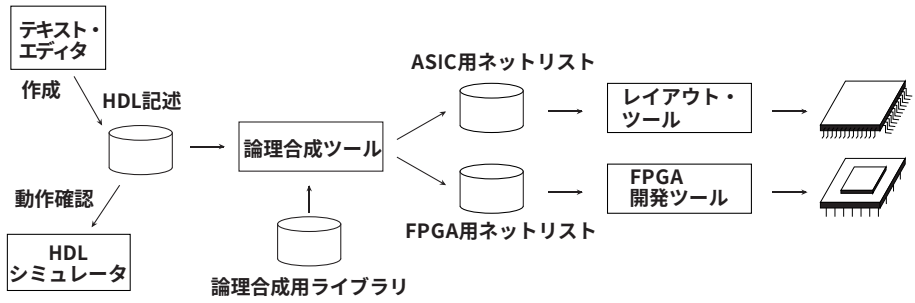
(1) 回路は，テキスト・エディタなどの上でHDLによって記述・作成する．
(2) 記述されたものは，HDLシミュレータを用いて動作の確認を行う．
(3) 動作確認がOKであれば，これを論理合成ツールに通す．

これらの手順で，所望のネットリストが生成されます．

HDL記述では，ASICやFPGAなどの設計対象を意識せずに設計できます．論理合成時に使用する合成ライブラリを変えることで，ターゲットをFPGAやゲートアレイなどに変えることができます．したがって，「FPGAで試作検証」を行い，「ASICで商品化」する，ということが容易に行えるのです．

ただし，論理合成は，現状ではASICやFPGA向きです．TTLなどのプログラマブルでない回路に

図1.1　HDLによるASICおよびFPGAの開発手順



は，適用できません．しかし，HDLは回路記述だけでなく，ボード・レベル・シミュレーションなどにも利用できます．つまり，すべての論理回路に用いることができるのです．

1.1.2　HDL設計のメリット/デメリット

● HDL設計のメリット――設計の効率化

(1) 半導体ベンダにとらわれない(ベンダ・フリー)設計が可能

従来ですと，ベンダとASICの品種が決定してから詳細設計に着手していました．回路図入力で用いるライブラリがベンダごとに，シリーズごとに異なっていたからです．HDLを用いれば，設計してから半導体ベンダを選ぶことすらできます．論理合成時に用いるライブラリを換えるだけで，各社のネットリストに変換できるからです．

(2) 論理合成による設計期間短縮

「カルノー図を書いてゲート数を減らす」などという作業は，過去のものとなりました．論理圧縮や遅延量の制御は，論理合成の得意とするところです．論理式から回路を生成する点においては，驚くほどの期間短縮を達成できます．

(3) 設計資産の活用

半導体ベンダにとらわれない設計ができる点だけでも，設計資産の活用といえます．さらに発展させて，HDLライブラリを構築することができます．カウンタ，シフト・レジスタなどの小規模なものから，オリジナルのCPUコア，画像圧縮伸張コアなどの大規模なものまでをライブラリ化すれば，強力な設計資産となりえます．いずれも論理合成できる記述とすることが前提ですし，できればテスト用の記述も含めてライブラリ化するべきでしょう．

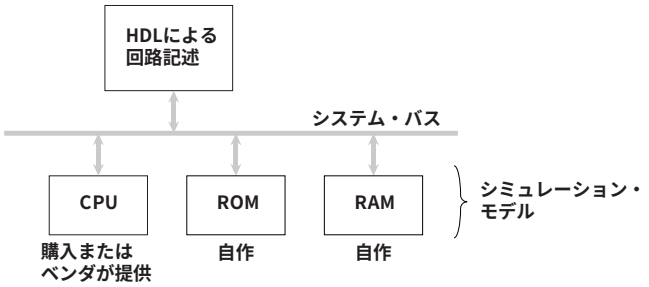
● HDL設計のメリット――検証精度の向上

ASICやFPGAに限らず，ボード設計においても**シミュレーションによる検証**の重要性が増してきました．

- 回路規模の増大
- 動作周波数の向上
- 個々の部品の高機能化，高集積化

などのため，安易にボードを作成しても満足に動作しないケースが目立ってきました．

図1.2
システム・レベル検証



このため，ボードを作成する前に，ASICを含めたボード全体のシミュレーションを行うことが，確実に動作させるための唯一の手段になりつつあります．

HDL設計を利用すると，回路図設計の場合より「検証精度」を向上させることができます．その理由をいくつか挙げてみます．

(1) 設計の途中で検証できる

HDLによる記述は，回路の「**概略設計の段階**」，「**詳細設計の段階**」，そして論理合成後の「**ゲート回路の段階**」で用いることができ，それぞれの段階でシミュレーションによる検証が可能です．

概略設計の段階から検証できるので，根本的な誤りを早期に発見できます．また，概略設計のブロックと，論理合成後のゲート回路ブロックが混在したシミュレーション(ミックスト・レベル・シミュレーション)が可能です．これにより，各ブロック間の設計作業の進みぐあいに差があっても，全体の検証が行えます．

(2) 入力の印加，出力の観測・比較が容易

回路検証のためには，回路の入力部に，動作に適した「**入力信号を印加させる必要**」があります．HDLでは，検証用の入力も同じ文法体系の中で記述できます．プログラム言語のようにきめ細かい記述が可能なので，例えばROM用のデータをファイルから読み込んだり，あらかじめ用意しておいた期待値との自動比較を行うこともできます．従来の手法では，回路図を書くこととシミュレーションのための入力を作成することはまったく別の作業でした．HDLでは可読性の高い検証入力を作成することができます．

(3) システム・レベルの検証が行える

検証のために設計対象に接続する周辺回路を，HDLで記述することができます．これを「**シミュレーション・モデル**」と呼びます．シミュレーション・モデルは，設計対象の検証だけに用途を絞って，大幅に簡略化して記述してもかまいません．**図1.2**に示すように，周辺チップを含めたシステム・レベルの検証が可能です．ボードで検証する前に，シミュレーション上で基本的な動作を確認することができます．

● HDL設計のデメリット

前記のようなメリットがある反面，HDLには次のようなデメリットもあります．

(1) 現状の論理合成は単相同期回路向き

非同期回路や多相の回路では，タイミング条件が複雑なため，論理合成時の最適化が単相ほど容易