Vol. 33 No. 11　　情　報　処　理　　Nov. 1992

解　説

ハードウェア記述言語

1. 論理合成時代のハードウェア記述言語†

安　浦　寛　人††

1. まえがき

集積回路技術の長足の進歩により，大規模な回路を比較的安価にかつ短期間で製造することが可能になった．さらにその設計においても 80 年代の論理合成技術の急速な進歩により，ソフトウェアにおける高級言語によるプログラミングと同等な程度にまで自動化が進められてきた．すなわち，高級言語に対応するハードウェア設計記述言語により機能設計（演算器やレジスタなどの基本的なハードウェアモジュールとその間のデータの移動，およびその制御条件）を記述しさえすれば，論理合成とレイアウト合成により自動的に回路のマスクパターンを生成することが可能となった．原理的には，レイアウトに関する知識や論理設計に関する細かい知識をもたなくても，システムの動作を言語で記述できれば（プログラムできれば），ハードウェアの設計ができる状況が産み出されている．これらの技術は，ASIC (Application Specific Integrated Circuits) 技術と統合され，システム設計の重要な技術として広く利用されはじめている[1]．

ASIC は，従来汎用の回路を一部の専門家が設計するものとされていた集積回路技術を一般の回路の利用者に解放する技術である．また，ASIC は，これまで分業されていたハードウェア設計とソフトウェア設計を融合する技術ともいえる．従来，ハードウェアの設計は，コスト面からの制約により，より汎用のものを作ってそれをソフトウェアで柔軟に機能を変えて利用することが多かった．ASIC は，これまでソフトウェアで実現していた機能の一部をハードウェアとして直接実現することを可能にし，各応用に適した専用の集積回路を作る手段を与えている．言い換えると，ASIC は，従来システムの中で，メモリ上にのみ実現されたソフトウェアを集積回路全体に分散することを可能とした．

システム設計者は，システム構想の段階で，どのような機能をハードウェアで実現し，どの部分をソフトウェアで実現するかを決める．ソフトウェアで実現される機能は，高級言語でプログラムされ，コンパイルされて，機械語のコードとなってメモリへロードされる．プログラムの実行は，CPU がメモリ上の命令を逐次呼び出して実行することで行われる．一方，ハードウェア部分は，ASIC チップとして実現され，システムに組み込まれる．このハードウェア部分の仕様を記述する道具が，ハードウェア記述言語 (HDL : Hardware Description (Design) Language) である．すなわち，ハードウェア記述言語は，ソフトウェアの高級言語に対応するものと考えられる．ソフトウェアのコンパイラに対応して，論理合成ツールおよび自動レイアウトツールがある．これらのツール群を統合して，シリコンコンパイラとよぶこともある．ソフトウェアの機械語コードに対応するのが，製造プロセスに渡されるマスクパターン情報であり，プロセスで製造されたチップがシステムに組み込まれて実際の計算を行う．半導体物理学はもちろん論理設計さえ知らない設計者が，ハードウェア記述言語を使ってプログラミング感覚で ASIC の設計を行う時代がきているのである．

本特集では，ハードウェア記述言語とそれを用いた新しいシステム設計手法の現状と問題点を解説し，その将来を占う．まず，本解説でハードウェア記述言語に関する一般的な解説を行う．第 2 部では，現在標準化活動など広く普及活動が行

† Hardware Description Languages in Logic Synthesis Era by Hiroto YASUURA (Department of Information Systems, Interdisciplinary Graduate School of Engineering Sciences, Kyushu University).

†† 九州大学大学院総合理工学研究科情報システム学専攻

われている4つの言語（UDL/I, VHDL, Verilog HDL, SFL）の特徴と現状を各言語の開発や標準化活動に参加されている方々に紹介していただく．さらに，第3部で，具体的な設計をそれぞれの言語で記述してその言語の相違点を明らかにする．

本特集の目的は，ハードウェア記述言語を利用してシステムを設計している人々の間の議論とするだけではなく，広く回路設計者，システム設計者，CAD 技術者にハードウェア記述言語とそれを用いた設計手法の現状を知らせることを目的としている．さらに，高級言語によるソフトウェアのプログラムとのアナロジや(超)並列言語としての位置づけ，ソフトウェアとハードウェアの同時/協調設計への発展の可能性など広く議論の出発点となることを願っている．

2. 論理合成とハードウェア記述言語による設計

2.1 論理合成の実用化による言語の役割の変化

ハードウェア記述言語は，1960 年代から開発され実際のシステムの設計過程でも利用されてきた[2),3)]．従来のハードウェア記述言語の多くは，

1) 論理シミュレーションや回路シミュレーションのための設計情報の計算機への入力手段としての言語

2) 機能設計やアーキテクチャ設計の段階での設計検証と性能評価のための設計記述言語

のいずれかの目的のために設計されたものがほとんどであった．1) のシミュレータへの入力手段としての言語は，グラフィックスエディタ上のスケマティック入力（論理図や回路図の図形としての直接入力）が普及するにつれ，ヒューマンインタフェースとしての存在価値は低下し，回路情報の計算機内でのデータ形式としての言語（たとえば EDIF[4)]）のみが生き残っている．2) の設計の初期段階での利用は，古くから提唱されてきたが，実際の設計現場にはなかなか普及しなかった．理由は，このレベルの言語で設計してシミュレーションなどで検証や性能評価を行っても，それをより詳細な論理回路レベルの設計へ変換するのは論理設計者の人手によるしかなかったからである．いくら上位で設計の検証を行っても，この論理設計過程で誤りが混入したり，設計の実質的な変更が加えられたりして，機能設計やアーキテクチャ設計段階での設計記述と実際の回路の間に大きな溝が存在したのである．

1980 年代後半になって論理合成の技術が実用的なものとなるにつれ，ハードウェア設計言語の立場は大きく変化してきた．すなわち，ハードウェア設計の全工程に深くかかわる設計ツールの中心的な存在へとその地位が変わってきたのである．たとえば，設計の対象となるハードウェアの機能記述は，以下のようにさまざまな設計ツールで利用される．

1) シミュレータなどの検証ツールでその正当性を検証される．

2) 論理合成ツールやレイアウト合成ツールなどの設計ツールによって，製造プロセスへ渡される最終データであるマスクパターンへ変換される．

3) テスト設計ツールの入力情報として利用され，回路のテストパターンが作成される．

4) ドキュメント作成のための原情報となる．

論理合成技術の進歩が機能設計やアーキテクチャ設計段階での設計と実際の回路の間の大きな溝を埋めたと言える．

このような言語の地位の変化は，

1) 言語の備えるべき機能/性質の変化

2) 言語に対する普遍性の要求

3) ハードウェア設計記述言語をベースとした新しい設計手法の確立

など新しい問題を産み出している．

1) については，論理合成を意識した UDL/I[5)] や SFL[6)] のような言語の出現や，テストや検証のための新しい記述形式の導入などが議論されている．2) は，標準化の問題である．米国を中心とした VHDL[7)] や Verilog HDL[8)] の標準化活動，また我が国の UDL/I[9)] の標準化など種々の活動が営まれており，その問題点も明らかになりつつある．3) は，実際にハードウェアの設計に従事している設計者にとってきわめて切実な問題である．米国ではすでに大学教育にハードウェア設計記述言語を導入し，言語をベースとした設計手法を身に付けた設計者を養成している．産業界でもかなりの数の回路が言語ベースで設計されている．

さらに，ハードウェア設計がソフトウェア設計に近づいたことで，システムの上流設計において，ハードウェアとソフトウェアの同時/協調設計の可能性もできた．本質的に並列に動作するシステムは，一部はハードウェアで，ほかの部分はソフトウェアで実現される．ソフトウェアとハードウェアは互いに交換可能な実現技術と考えることができ，両者のバランスを考えることが設計者の重要な仕事となる．また，ソフトウェアに比べて養成が遅れているハードウェア設計者の不足を補う上からも，ソフトウェアと同じ感覚でハードウェア設計が行える環境を整えることは重要である．

2.2 論理合成時代のハードウェア記述言語への要求[5)]

論理合成時代のハードウェア記述言語は，ソフトウェアにおけるFortranやCのような高級言語に対応しており，その重要性は，ソフトウェアにおける高級言語とまさるとも劣らないものがある．このようなハードウェア記述言語に求められる性質としては，以下のようなものがあげられる．

1) 表現力/書きやすさ/読みやすさ：単に，CADツールへの入力言語というだけではなく，ユーザがこの言語を用いて思考し設計を進められる．ハードウェア設計に本質的な並列動作を自然に記述できる．また，記述が設計仕様として，あるいは設計ドキュメントとしても利用できることが望ましい．

2) 一般性/標準性：幅広い設計文化や設計手法で利用できる．多くのCADツールや製造プロセスとインタフェースを確立している．

3) 適応性/柔軟性/拡張性：テストや検証など，設計に付随する種々の問題の解決にも利用できる．また，今後の技術の進展に柔軟に対応できる．

4) 保守性/検証容易性：記述のデバッグや検証，保守が容易である．

これらの性質は，ソフトウェアにおける高級言語のもつべき性質と多くの共通点をもっている．しかし，ハードウェア記述言語固有の問題点もいくつか指摘されている．

1) 記述のレベルの多様性：ソフトウェアでも，種々の高級言語からアセンブラ言語まで，いくつかの記述レベルがあり，いろいろな言語が開発/利用されている．これらの言語は，レベルの違いはあっても，逐次型言語の場合，その記述の意味の解釈機構はほとんど共通のもので，このことがユーザの理解のしやすさを助けている．一方，ハードウェア記述言語の場合は，回路の動作や機能を記述する種々のレベル，機能素子の結線構造を記述するレベル，マスクパターンのような図形の形状/位置情報を記述するレベルなどその意味解釈の規則が大きく異なるレベルを含んでいる．動作/機能の記述レベルに限っても，記述の対象となる動作が本質的に並列的であるため，意味解釈の規則は必ずしも統一されておらず，ユーザの理解の妨げとなっている．

2) 設計手法への依存性：ハードウェア記述言語（特に，論理回路記述や動作/機能記述）は，設計手法に依存しないように作られるほうが一般性が高くなるが，言語の設計は難しくなる．完全同期設計を前提とする言語と非同期設計を許す言語では，同期回路を記述するときの記述量や書きやすさが大きく異なる．

3) CAD技術の制約：ハードウェア記述言語（特に，論理回路記述や動作/機能記述）は，従来，シミュレータへの入力言語として開発されてきた．このため，シミュレーション技術の制約を強く受けた言語が多い．しかし，論理合成時代になると，合成ツールが主役であり，シミュレータは脇役（検証ツール）となると予想される．しかし，従来の経緯からシミュレーション技術の制約の影響は大きい．論理合成技術はまだ発展段階の技術であり，今後の発展の方向は定かではなく，現在の合成技術に基づいて将来の言語を議論するのも危険である．

4) 製造技術との整合性：ハードウェア記述言語で設計される回路は，最終的には集積回路としてチップ上に実現される．理想的には，集積回路のデバイステクノロジは，上位の設計には影響しないようにすることが望まれるが，現実には難しい．このため，ハードウェア記述言語によっては，ある程度の範囲でデバイステクノロジを意識したものとなる．これは，ユーザにある程度のデバイスに対する知識を要求することになり，また，記述を実現できるデバイスの範囲を限定することになる．

3. ハードウェア記述言語の特徴

ハードウェア記述言語を特徴付ける主な要因としては，1) 言語がカバーする設計レベルと設計手法（言語のスコープ），2) 言語の意味論，3) 言語の文法がある．これらは，言語設計の基本思想に従って決定されるべきものである．ここでは，言語のスコープと意味論について，既存の言語における問題点を整理し，今後の言語設計の方向を議論する．

3.1 言語のスコープ

回路の設計には，段階的にいくつかのレベルが想定される．設計の当初は，まだ具体的なハードウェアのイメージが希薄なアーキテクチャやアルゴリズムのレベルであり，それが，具体的な機能単位に分解されて，動作/機能レベルの記述となる．アーキテクチャやアルゴリズムレベルの記述から，動作/機能レベルの記述を生成する技術を高位合成とよぶこともある．次に，よく利用される論理素子などを用いた論理回路レベルの記述がある．動作/機能レベルの記述から，論理回路レベルの記述を合成することを（狭義の）論理合成という．さらに，トランジスタや抵抗を含みアナログ的な動作も考える電子回路レベルがある．論理回路レベルまでがデバイスや製造プロセスに依存しないテクノロジ独立性を保持しやすいのに対し，電子回路レベルはデバイステクノロジに依存せざるをえない．電子回路を実際に2次元平面上に配置して，チップのマスクパターンの幾何学的な情報を表したものがレイアウト記述である．レイアウト記述は，完全にデバイステクノロジに依存しており，各製造ラインのプロセスパラメータに依存することも少なくない．論理回路からレイアウト記述を作る技術は，自動レイアウト技術として，広く実用されている．

これら種々の設計レベルに対応するハードウェア記述言語について考える．高位レベルの記述では抽象度が高くなり，同じシステムを記述するのに使われる記述量は，より下位の記述に較べて少なくなる．これは，設計の各レベルで，詳細化が行われ，情報が付加されることに対応する．この詳細化にともない，情報および時間の粒度も細かくなる（**図-1** 参照）．

アーキテクチャレベルでは，情報の単位は整数

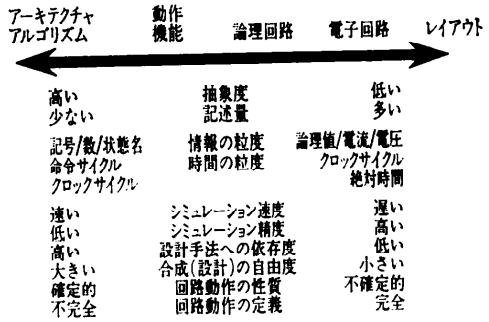

図-1 設計のレベルと記述の性質

や記号であり，時間の単位も命令サイクルやクロックサイクルであるが，論理回路レベルでは，論理値 (0/1) で情報が表され，時間の単位も絶対時間が用いられる．さらに，電子回路やレイアウトのレベルでは，情報を表す電流や電圧のような物理量が問題になる．これらの性質は，各レベルでの設計検証を行うシミュレータの性質にも反映される．上位では，抽象度が高く，情報や時間の粒度が粗い分だけシミュレーションの精度は低く，速度は速くなる．一方，論理合成の立場からみると，上位になるほどハードウェアとの距離が大きくなり，実現の自由度が大きくなる．

回路の動作は，上位では，数学的なモデル化を行って，確実に動作する回路を仮定することができるが，電子回路やレイアウトのような下位では，制御できない物理現象の壁にぶつかる．トランジスタのサイズや特性は，製造時にはある精度以上制御できない．言語の記述精度がこの製造精度に近づくと，言語で厳密に記述しても製造に反映できないということになる．たとえば，素子の遅延を正確に 3 ns とすると記述しても，製造時にその値をきっちりと実現できる保証はない．このため，下位の言語では，製造精度の不足を補完するための手段が必要となる．一方，記述による回路動作の定義は，上位では大まかな部分だけしか記述しないことが多く，不完全になりがちである．特に，例外処理や Don't Care 条件は設計の詳細化が進むまで決定されずにおいておかれる．一方，下位では，あらゆる入力に対する動作が必然的に定義される．

多くのハードウェア記述言語においては，構造化設計手法を支援するために，異なったレベルの設計が混在した設計記述ができるものが多い．しかし，このようなハードウェア記述言語には，本

質的に難しい問題が内在している．すなわち，各レベル間では抽象度が異なり，情報や時間の粒度が異なるため，記述の表す意味（回路の動作）を定義するためには，レベル間の信号の授受の規約を設定する必要がでてくる．通常，レベルの低いほうに合わせてつじつまをあわせるような規約が用いられるが，つきつめて考えると矛盾を含む場合や，上位の記述の意味が制限されてしまう場合が多い．一般にこの問題をどのように解決すれば良いかは，まだ明らかにはなっていない．

実際の設計現場では，設計を正確にかつ迅速に行うために多くの設計ルールを定め，設計手法を限定している．タイミングの問題が起こりやすい非同期設計は一切禁止して，同期式回路だけを用いるなどがその例である．ハードウェア記述言語の対象をある特定の設計手法に限定すると，その設計手法の設計ルールに反するような設計は記述できないようにすることができ，設計に誤りが混入することを防ぐことができる．さらに，特定の設計手法に対しては論理合成系も作りやすい．しかし，このような設計手法の限定は，設計者の自由度を奪うことにもなり，新しい設計手法の開発や細かな工夫による性能向上の芽を摘むことにもなる．一方，どのような設計手法にも対応するような言語を考えると，論理合成系の実現が難しくなったり，製造技術との整合性をとることが困難になるという問題が起こる．

3.2 言語の意味論 (Semantics)

ハードウェア記述言語は，互いに並列に動作するモジュールの集まりである回路の動作を記述するので，本質的に並列言語でなければならない．一般に，並列言語の意味論をどのようにして与えるか，またどのような意味論が目的とする記述にふさわしいかについては，ソフトウェアの世界においても解決された問題とは言えない．

ハードウェア記述言語の意味論の構築の難しさは，言語による記述（仕様）に対して異なった2種類の実現形態が存在する点にもある．すなわち，伝統的なシミュレータによるソフトウェアとしての実現と論理合成系によるハードウェアとしての実現である．現存のハードウェア記述言語はその成立過程から，シミュレーション向きのハードウェア記述言語と合成向きのハードウェア記述言語に分けることができる．これらの間には，以下のような相違点が考えられる．

シミュレーション向きのハードウェア記述言語は，対象とする回路の動作/機能をいかに効率良く通常の計算機で模擬するかを念頭において作られる．すなわち，記述される対象は，対象回路とほぼ同じ動作をするシミュレーションモデル（ここでは，ソフトウェアモデルと呼ぶ）である．実際には，ハードウェア記述言語による記述は，シミュレータ用のコンパイラでソフトウェアモデルに変換されるが，ユーザは細かな動作を考えるとき，ソフトウェアモデルを書いていることを意識せざるをえない．

一方，合成向きの記述は，論理合成システムにおいて回路（ハードウェアモデルとよぶ）に変換されることを前提に記述される．ユーザは，合成される回路のイメージを意識しつつ，回路の仕様や実現上の希望を記述することになる．

同じハードウェア記述言語で，シミュレーションにも合成にも向いたものがあれば，問題は簡単である（ほとんどの，新しい言語は両方に向いていると称している）．しかし，現実においては，ソフトウェアモデルとハードウェアモデルの間に大きな溝（ギャップ）が存在する．これは，ソフトウェアモデルが，現在の限られた計算機能力のもとで，意味のある時間内に実用的な規模の回路をシミュレーションするために，種々の近似を行っていることに起因する．従来のほとんどすべての言葉が，シミュレータを一義的な対象としているため，ソフトウェアモデルは書きやすいが，ハードウェアモデルについて合成の立場から書きたいことを必ずしも書きやすくなっていないということが問題である．

言語の意味論の立場から言えば，シミュレーション向きハードウェア記述言語の意味論は，ソフトウェアモデルにきわめて近く，シミュレータ用コンパイラは，意味的な変換をほとんどしなくてもよかった．しかし，このようなハードウェア記述言語を合成システムの入力とすると，合成システムやユーザは，ソフトウェアモデルとハードウェアモデルの間のギャップを埋めなくてはならず，大きな負担がかかると思われる．一方，合成だけを考えてハードウェア記述言語をつくれば，シミュレータ側に負担がかかる可能性がある．今後，CAD ツールの主役が合成システムへ移行す

るならば，ハードウェア記述言語の意味論のあり方も大きく変わる可能性がある．

論理合成向きのハードウェア記述言語では，どのようなことが書きたいであろうか．以下のような項目は，現在使用されているシミュレーション向きのハードウェア記述言語では，書くことが難しい事項である．

1) 回路動作の物理的な不確定性：回路動作は，物理現象として実現されるが，われわれが制御できる現象には限界がある．たとえば，素子の遅延のばらつきや準安定状態と呼ばれるような回路動作などは制御できない．ユーザが制御できないと分かっていることは，素直にハードウェア記述言語による記述に反映したい．最大/最小遅延などを定義できる言語もあるが，その意味論はシミュレーションの都合によって大きく制限されている．

2) 記述の不完全性からくる設計の自由度：想定しない入力に対する出力値（通常 Don't Care として扱われる）などのように設計初期で指定したくない動作については設計の自由度を残すうえからも意識的に記述したい．シミュレータの立場からは，このような自由度はシミュレーションを規定できないのでできるだけ排除する方向になる．

3) 回路性能の相対的な仕様：「回路の遅延を 100 ns 以下でできるだけ小さくしたい」というような定量的で相対的な仕様の記述も必要である．既存のハードウェア記述言語でこのような設計者の「気持ち」が書けるものは皆無である．

今後，論理合成技術の進歩にともない，ハードウェア記述言語の形が大きく変わると考えられる．また，テストや設計検証（シミュレーションのほかに形式的な検証も含む），回路の仕様やドキュメント記述への利用なども今後のハードウェア記述言語の意味論に大きく影響すると考えられる．

4. ハードウェア記述言語の標準化

これまでに，多くのハードウェア記述言語が提案され実用されてきた．レイアウトレベルのデータの表現のための CIF[10] や EDIF[4] のようなデータフォーマット，ネットリストを表現する数多くのシミュレータ入力言語，機能シミュレータ用の機能記述言語など数えればきりがない．従来の言語は，ある特定の CAD ツールの入力用言語であるものが多く，また，特定の記述レベルを対象としたものがほとんどであった．しかし，CAD のツールの多用化により，ツール間でのデータの互換性が問題となるようになって，まずレイアウトレベルの言語の標準化が始まった．CIF や EDIF はその例である．レイアウトレベルでは，言語は基本的にツール間のデータ互換性を保証するためのもので，構造も単純で，言語というよりデータフォーマットとよぶほうがふさわしいものである．それゆえ，その標準化も比較的簡単であった．

論理設計や機能記述用の言語となると，設計手法への依存性，CAD 技術の制約などが大きく影響して，話が難しくなる．特に，回路の動作や機能を記述する言語は，ネットリストやレイアウトのような回路の構造を記述するものとは違い，並列的な回路動作を記述する難しさが加わる．従来は，シミュレータをベースに言語を定義することが広く行われてきたが，論理合成のような新しい技術への応用を考えるとより一般的な言語の定義法が必要となる．上位レベルのハードウェア記述言語の標準化としては，70 年代の CONLAN[11] の活動を皮切りに，80 年代に入って米国の VHDL[7] や Verilog HDL[8]．日本の UDL/I[9] などの活動が行われている．

ハードウェア記述言語の標準化は，ユーザの便宜（インタフェースの共通化），CAD ツール開発の効率化，設計資産の共有/再利用などの立場からきわめて重要である．しかし，一般に技術の標準化は，標準化による技術の発展の阻害という側面をもつ．標準を技術が成熟してから作るのでは合意を得ることが難しいし，成熟する前に作れば新しく現れる技術の前に陳腐化するか新しい技術の芽をつむことになる．ハードウェア記述言語の標準化が，いろいろな機関で議論されてきた背景には，シミュレーション技術の成熟と合成技術の今後の伸びに対する期待とがある．

シミュレーションの立場からみると，従来のハードウェア記述言語が基本的にシミュレーション用言語であり，ハードウェア記述言語の意味論とシミュレータの構造（アルゴリズム）とはほぼ一致していた．このため，ハードウェア記述言語の意味論は，シミュレータによって規定されてい

たといえる．時には，シミュレータ設計者が高速化のためにアルゴリズムを変えたことで，ハードウェア記述言語の意味論が変わることさえあった．これは，各ハードウェア記述言語に一つのシミュレータが対応していた場合には許されることであった．しかし，標準化ハードウェア記述言語では，複数の組織で複数のシミュレータが開発されることになる．このため，ハードウェア記述言語の意味論を特定のシミュレータで与えることはできなくなる．標準化ハードウェア記述言語の意味論は，ある程度シミュレータから独立に定義されることとなる．特定のシミュレーション技術に依存し過ぎると，一般に受け入れられにくく，標準として機能しない．しかし，既存の技術の最大公約数的な標準を作ると，最先端の技術と程遠く，利用されないか，利用された場合は技術の進歩を止めることになる．

一方，合成技術はまだ成熟した技術ではなく，現在の技術を基礎に標準ハードウェア記述言語のあり方を規定するのは時期尚早と思われる．しかも，CAD ツールの主役はまだシミュレータであり，シミュレータを無視した合成向き言語は現時点では考えにくい．

5. 各言語の解説に対する質問事項

本特集の第2部では，現在，標準化や普及活動が進められている4つの言語（UDL/I, VHDL, Verilog HDL, SFL）についてそれぞれ解説をお願いした．各著者は，それぞれの言語の開発者または普及/標準活動で活躍されている方々である．執筆にあたって，読者に各言語の特徴を理解してもらいやすいように下記のような項目について書いていただくようにお願いした．

1) 言語設計の基本思想：各言語の基本的な設計思想と言語設計の背景を明らかにし，その言語が目指しているものを明確にする．

2) 言語の主な特徴（長所，短所，制約など）：各言語のスコープ，意味論，文法などの特徴を示し，論理合成時代のハードウェア記述言語としての長所，短所を明らかにする．

3) 処理系の開発状況：この言語をベースとする CAD システム（特にシミュレータと論理合成系）の開発状況を報告してもらう．

4) 標準化の状況：標準化を目指している言語について，その標準化活動の概要を紹介してもらう．

5) 実際の設計への適用例：実際の設計現場で使用した経験を報告してもらい，言語を利用した設計の利点や欠点を明確にしてもらう．

6. あ と が き

ハードウェア記述言語の議論は，種々の技術的，政治的問題（特に標準化において）が錯綜して，難しい問題を含んでいる．言語の文法自体は，必ずしも本質的なものではなく，ユーザごとに自分の設計文化や設計手法に合わせてマクロを定義して使うことも考えられる．重要なのは，回路動作の並列計算モデルと言語の意味論の提供である．この分野の研究はまだ始まったばかりで，確立した技術もない．今後の研究成果が期待される．アナログ回路への対応や，電子回路レベルからアーキテクチャレベルまでを統一的に扱う手法なども議論されているが，最終的には意味論の問題となる．プログラミング言語とのアナロジでハードウェア記述言語の定義を行うことはそんなに難しい作業ではない．しかし，今後の CAD 技術の進歩や設計手法の変化への対応も考え，現在の CAD 技術や設計手法のすべてをカバーできる言語の枠組みを構築するのは幅広い知識と洞察力を要求される困難な仕事である．単なる駆け引きや流行だけですむ話ではない．今後もいくつかの言語が提案され，淘汰されつつ，より使いやすい言語が産み出されるものと考える．

参 考 文 献

1) 田丸他：ASIC の現状と将来動向，平成3年電気・情報関連学会連合大会講演論文集分冊 3 S 15, pp. 21-52 (1991).
2) 山田編，上原，白石，鹿毛：VLSI コンピュータの CAD，コンピュータサイエンス・ライブラリー，産業図書 (1983).
3) 中村，小栗：ハードウェア記述言語とその応用，情報処理，Vol. 25, No. 10, pp. 1033-1040 (1984).
4) Electronic Industries Association : Electronic Design Interchange Format Version 200, ANSI/EIA-548-1988 (1988).
5) 唐津，星野，石浦，安浦：論理合成時代のハードウェア記述言語：UDL/I，電子情報通信学会論文誌，Vol. J 74-A, No. 2, pp. 170-178 (1991).
6) 中村，小栗，野村：RTL 動作記述言語 SFL，電子情報通信学会論文誌，Vol. J 72-A, No. 10, pp. 1579-1593 (1989).

7) IEEE: VHDL ハードウェア記述言語，日本規格協会 (1991).
8) Open Verilog International: Verilog Hardware Description Language Reference Manual (1991).
9) UDL/I 標準化委員会: UDL/I 言語仕様第 1.0m 版，日本電子工業振興協会 (1992).
10) Mead, C. and Conway, C.: Introduction to VLSI Systems, Addison-Wesley (1980).
11) Piloty, R., Barbacci, M., Borrione, D., Dietmeyer, D., Hill, F. and Skelly, P.: CONLAN Report, Lecture Notes in Computer Science 151, Springer-Verlag (1983).

(平成4年7月8日受付)

安浦 寛人 (正会員)

1953 年生. 1976 年京都大学工学部情報工学科卒業. 1978 年同大学院修士課程情報工学専攻修了. 1980 年より京都大学工学部助手. 1986 年より同電子工学科助教授. 1991 年より九州大学大学院総合理工学研究科教授. 並列アルゴリズム，並列計算機アーキテクチャ，論理設計，VLSI 用 CAD の研究に従事. 1987 年度電子情報通信学会，1990 年度情報処理学会論文賞受賞. IEEE, ACM, 電子情報通信学会，ソフトウェア科学会，LA シンポジウム，EATCS 各会員.