論理合成とマイクロプログラムの統一設計環境

野地 保[†] 中村 彰[††]

専用の設計記述言語とシミュレータを用いるマイクロプログラム設計環境では論理合成を主体とするハードウェア設計環境との設計データの共有化を図ることが困難なため，共通に使用できる設計データがあるにもかかわらず各々独立に開発しなければならず余分な開発工数がかかっていた．本論文では論理合成可能なハードウェア記述言語（HDL）を共通の設計記述言語として用い，論理合成環境とマイクロプログラム設計環境を融合させたトップダウン設計手法指向の統一設計環境の一構成法について提案する．本構成法では，同一の，論理合成可能な標準 HDL で，ハードウェア動作の機能記述とマイクロプログラム記述を行い，設計の前段階で必要な，機能/論理/マイクロプログラム検証（シミュレーション）モデルが一つに融合した形で構築できる．シミュレーション実行後，ハードウェア機能記述は論理合成によりゲート（論理回路）へと展開され，マイクロプログラム記述は自動抽出されてマイクロアセンブラ，リンカによりロードモジュールへと展開できる．実機器の開発に適用した結果では，従来の専用マイクロプログラム設計環境と比べて，5人月かかっていたマイクロプログラム専用シミュレータの開発工数を0にすることができた．これらの設計データは論理合成可能な標準 HDL で記述されているため，設計財産の共通化と流用性の向上にも効果が期待できる．

Top-Down Design System Integrating Logic Synthesis and Microprogram Design

Tamotsu Noji[†] and Akira Nakamura[††]

There are a lot of common design data that "could be" shared between hardware design environments that consist mainly of logic synthesis, and microprogram design environments that utilize a design description language and a simulator. The two environments, however, have been developed separately and therefore design data sharing between them has not achieved. It has resulted in taking more development manpower. This paper proposes a method for constructing an integrated design environment oriented top-down design methodology. The function, logic and microprogram are described by a common language (standard HDL) in this environment, and they are verified integratively at an early design stage. Then the verified hardware functional description is developed into the gate-level logic circuits by the logic synthesizer. On the other hand, the verified microprogram description is extracted automatically and converted into the load-module by the micro-assembler and linker. The results of applying the method to an actual VLSI machine development shows that manpower for developing the microprogram simulator can be cut down to zero, and that design data sharing can be enhanced. The design data written by standard HDL can also be used in other VLSI machine designs, thus the re-usability of design data (properties) will be enhanced.

1. はじめに

大規模 LSI（ASIC）で構成される情報処理機器開発分野において，設計方式は，論理図面を基本とした方式からハードウェア記述言語（HDL）を基本とし論理合成を活用する方式（トップダウン設計手法[1]）に移りつつある．設計者共通のコミュニケーション手段としての HDL の標準化[2]も進み，IEEE 標準の VHDL[3]や業界標準の Verilog HDL[4]が使用されている．論理合成を活用したトップダウン設計手法では，方式設計，機能設計後 RTL(Register Transfer Level) での機能検証（シミュレーション）を行い，誤りがとれた段階で，論理合成により自動的に論理設計（ゲートレベルへの展開）を行い，さらに再確認のためゲートレベルの論理シミュレーションを実施する．この手法は人手による論理設計期間を大幅に短縮でき大規模 LSI(ASIC) の設計効率を向上することができる．

一方，マイクロプログラム方式[5),6]は，計算機をは

† 三菱電機(株)システム LSI 開発研究所
Mitsubishi Electric Corp., SL-Lab.
†† 長崎大学工学部電気情報工学科
Department of Electrical Engineering and Computer Science, Nagasaki University

じめとする多くの機器分野で採用されている．機器内部に組み込まれるマイクロプログラムは機種依存性が高いため，開発支援ツールも機種専用のシステムとして作られた場合が多く，汎用性をどれだけ持たせることができるかが[7]大きな課題である．マイクロプログラム記述言語[8]で記述されるマイクロプログラムは，実際の情報処理機器（マシン）に近い形の検証モデル（シミュレータ[9]）上でその正当性チェック（デバッグ）を実施する必要がある．

マイクロプログラム設計専用の支援システムでは，論理設計環境で使用するハードウェア記述言語とは異なるC言語や検証モデルを記述する専用の設計記述言語[11),12)]を用いて論理シミュレーション環境とは独立にマイクロプログラムシミュレーション環境[23)]を構築する．この場合，論理設計環境とマイクロプログラム設計環境を別々に提供するため，マイクロプログラムの検証モデルは，機能的に論理の検証モデルに近い関係にあるにも係わらず，同じようなシミュレーション環境を別個に開発する必要がある．

シミュレーション環境の統一化を図るため，マイクロプログラムの検証モデルの記述に専用のハードウェア記述言語[13)]を用い，またそのシミュレーション機能を用いてマイクロプログラムの検証を行う例[14)]もある．

さらに，設計記述言語 HSL-FX[15)]を用いて論理設計，マイクロプログラム設計などの設計レベルを統一的に支援する方法や，マイクロプログラムのシミュレーション結果を論理シミュレーションの入力データとして使用する方法[16)]，これを混合シミュレーションで行う方法[17)]などが試みられている．

しかしながら論理合成を前提とした設計データをマイクロプログラム設計でも利用できる両者を融合した設計環境づくりには，いくつかの課題がある．マイクロプログラムのシミュレーション環境は，論理合成前に実施する機能シミュレーション環境との共通化を図る必要があるが，単なる統一化を考えるだけでなく，シミュレーション後，機能記述部は自動的に論理合成され，マイクロプログラム記述部はマイクロプログラムのアセンブラ処理ルーチンへ繋がる，いわゆる自動的にトップダウン設計ができる統一環境の実現が望まれる．設計データや設計環境の流用性を向上させるためには，ハードウェア設計者とマイクロプログラム設計者が同一言語を使う必要があるが，両者の目的を包含する適した言語の選択が必要となる．

本論文では，このような課題を解決する一手法として，同一の論理合成可能な，標準ハードウェア記述言語を使い，ハードウェア動作の機能記述とマイクロプログラム記述を同一のシミュレーション環境で実施した後，ハードウェア機能記述部とマイクロプログラム記述部を各々自動的に抽出して，ハードウェア機能記述部は論理合成へ，マイクロプログラム記述部はマイクロアセンブラに展開できるトップダウン設計手法指向の統一設計環境について，その一構成法を報告する．

以下，第2章で設計環境のモデル化方針を，第3章で実現方式を，第4章で本統一設計環境を実機器に応用した例とその評価結果を示す．

2. 設計環境のモデル化方針

2.1 モデル化の基本方針

マイクロプログラム設計環境と論理合成環境の両者を統一した設計環境のモデル化を行うため，次の基本方針を設定する．

(1)ハードウェアの機能記述結果がそのまま論理合成に利用できることが望まれるため，マイクロプログラム記述も同一の論理合成可能なハードウェア記述言語（HDL）を使用する．

(2)設計期間の短縮と効率化を図るため，ハードウェア設計に用いるシミュレーションモデルと，マイクロプログラム設計に使用するシミュレーションモデルを同一の設計環境で実現する．

(3)設計効率の向上を図るため，ハードウェア設計とマイクロプログラム設計を同時に並行して行うことができる設計環境とする．

(4)ハードウェアの機能記述結果は，人手変換なしに論理合成できる設計環境へ自動的に接続できる．

(5)マイクロプログラム記述部を HDL 記述部より自動的に抽出し，マイクロプログラム設計環境に接続しアセンブル，リンクできるようにする．

(6)論理合成環境における論理合成の対象範囲は，レジスタトランスファレベル（RTL）とする．

(7)マイクロプログラム設計環境は，多様な機種のマイクロプログラム開発に適用できる汎用のあるものとする[18)]．

2.2 設計フロー

トップダウン設計手法を考慮した論理合成環境とマイクロプログラムの設計環境との統一設計環境の設計フローを**図1**に示し，以下説明する．

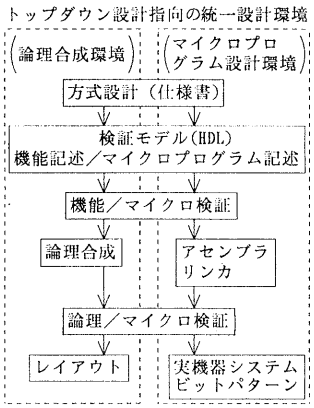

図 1 統一設計環境での設計フロー
Fig. 1 Integrated logic/microprogram design flow.

方式・機能設計の仕様書に従い，HDL を用いてハードウェア動作（機能記述）とマイクロプログラム動作（マイクロプログラム記述）を盛り込んだ形で，検証モデルを構築する．機能記述のレベルは RTL である．人手作成もしくは診断プログラム等から抽出したテストパターンを用いて検証モデル上でハードウェアとマイクロプログラムの動作確認（機能検証/マイクロプログラム検証）を行う．

機能/マイクロプログラム検証により機能記述とマイクロプログラム記述両方の設計の誤りがなくなれば，ハードウェア動作を RTL で表現した機能記述部は論理合成によりゲートレベルへと自動的に展開できる．マイクロプログラム記述部は，抽出プログラムにより機能記述部から自動的に分離/抽出され，アセンブル，リンクを経て，実行可能なビットパターン（ロードモジュール）に展開される．

この後，マイクロプログラムのロードモジュールも含めて，論理合成結果に対して，再度論理検証/マイクロプログラム検証を実施して，論理回路とマイクロプログラムの正当性を確認する．

論理/マイクロプログラム検証で誤りがなくなれば，論理合成環境は LSI チップ開発を行うためレイアウト工程の手順に進める．マイクロプログラム設計環境は，論理合成環境で開発される LSI チップで構成される実機器システムの ROM/RAM 用ビットパターンを出力して設計を完了する．以上の設計フローを前提とした実現方式を以下述べる．

3. 実 現 方 式

同一 HDL により機能記述とマイクロプログラム記述を共存させてシミュレーション，設計できるトップダウン設計手法指向の統一設計環境を実現するためには，どちらの環境を主体にするかが大きな課題である．一般的にハードウェア設計の結果としての LSI チップ開発は開発時間，開発費用とチップ性能が要求され，やり直しに大きなリスクを伴う．マイクロプログラム設計は，通常書き換え可能なメモリ（RAM）にロードされ使用される場合が多く，再設計時のターンアラウンド時間がハードウェア設計に比べて短い．このような観点から本統一設計環境では，論理合成環境を主体に考え，マイクロプログラム設計環境を論理合成環境に取り込むこととする．以下にマイクロプログラム設計環境を論理合成環境にいかに融合させるべきかの点に的を絞ってその実現方式について述べる．

3.1 ハードウェア記述言語（HDL）と統合記述方式

統一設計環境で要求されるハードウェア記述言語（HDL）の機能仕様は，(1)ハードウェア動作の機能記述結果が論理合成可能であること，(2)マイクロプログラム記述が可能であること，(3)業界標準で大学，メーカなどの研究機関で使用実績があるもの，(4)言語仕様が公開されており，誰でも自由に使用できること，などである．これらに適した HDL として VHDL と Verilog HDL が考えられるが，使用実績とシミュレータや論理合成ツールなどの処理系の豊富さから業界標準（OVI[4]：Open Verilog International）の Verilog HDL の活用を検討した．Verilog HDL は，アーキテクチャ，アリゴリズム，RTL，ゲート，スイッチレベルの記述が可能で合成環境でも多く使用され，(1)と(3)(4)の条件を満たしている．(2)の点に関しては，①マイクロプログラムの言語仕様定義が可能であること，②マイクロ命令の動作記述が可能であることが必要であり，Verilog HDL（以下，HDL と略す）ではマイクロ命令を構成する複数のフィールドをシンボルとして定義する単純な方式を採った．

3.1.1 統合記述方式

ハードウェア動作の機能記述とマイクロプログラム記述の混在方式には，以下の2つの方法が考えられる．

①マイクロ命令の動作を表すマイクロプログラム記述

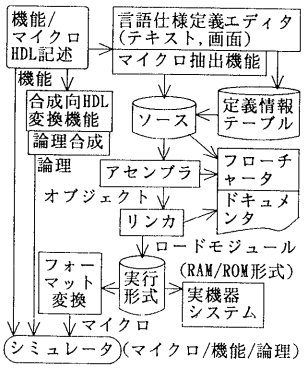

図 2 統一設計環境のシステム構成図
Fig. 2 Integrated design system configuration.

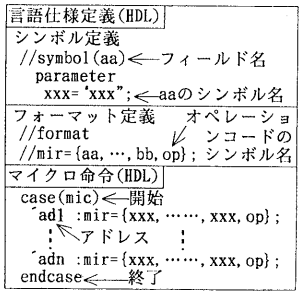

図 3 マイクロプログラムの記述形式
Fig. 3 Microprogram description format.

部とハードウェア動作の機能記述部を各々分離して HDL で記述する．設計データが2本立てとなる．②マイクロプログラム記述部をハードウェアの機能記述部の中に埋め込み（統合記述），マイクロプログラムのアセンブル処理に移る段階でマイクロプログラム記述部のみを自動的に抽出する．

本統一設計環境では，設計データの一元化を図るため②の統合記述方式を採り，マイクロプログラム記述を一定の HDL 形式で行い識別する方式を実現した．**図2**に統一設計環境のシステム構成図，**図3**にマイクロプログラムの記述形式を示す．マイクロプログラムはマイクロ命令の集まりであり，マイクロ命令は複数のフィールドで構成される．マイクロプログラムの言語仕様定義では各フィールドの動作（意味）を定義する．マイクロプログラム記述部は図3に示すように言語仕様定義部とマイクロ命令部で構成される．この例では，各フィールドに対応するシンボル定義を //symbol (aaa) 以下の記述で，マイクロ命令のフォーマット定義を//format 以下の記述で表現している．マイクロ命令はマイクロ命令単位に記述され，case(mic) がマイクロ命令の開始を，endcase がマイクロ命令の終了を，ad1〜adn がアドレスを表している．記述されるマイクロ命令のステップ数は可変でハードウェア動作に対応して，機能記述と混在して記述される．

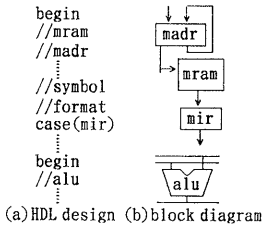

図 4 混在記述例
Fig. 4 Mixed description example.

図4に両者混在の HDL 記述例と機能ブロック図を示す．マイクロプログラムのアドレスカウンタ madr で指定されるアドレスに従い mram に入っているマイクロ命令がマイクロ命令レジスタ mir に読み出され，そのデコード信号により alu 演算動作を行う．madr のハードウェア動作は begin,//madr で，alu 動作は，begin,//alu で示される機能記述により実行される．マイクロプログラム記述は，//symbol などで記述される．

3.1.2 自動抽出

マイクロプログラムとハードウェア動作の確認が完了すれば，混在記述部からマイクロプログラムの記述形式に従い，マイクロプログラム記述部を自動的に抽出する．言語仕様定義部は図2に示す言語仕様定義エディタで処理，マイクロ命令部は図2に示すマイクロ抽出機能（抽出プログラム）で処理する．これらの処理の中で HDL 形式をアセンブラ，リンカが処理できる形式にフォーマット変換する．

3.2 シミュレータ

論理合成環境とマイクロプログラム設計環境に共通なシミュレータに必要な条件として次のことを検討した．

第一に汎用性である．マイクロプログラム専用のシミュレータ[18]は存在するが，新しい開発ごとにシミュレータを開発する手間隙を避けたい．また汎用のワークステーションで動くことも必要である．

第二に実用に耐え得る高速性である．論理合成環境では2つのハードウェアの動作確認作業が必要である．1つは HDL 機能記述結果を論理合成にかける前に実施する RTL 記述を対象とする機能検証（機能モデルによる機能シミュレーション）であり，もう1つ

は論理合成後のゲートレベルを対象として論理の確認を行う論理検証（論理モデルによる論理シミュレーション）である．機能検証は論理検証に比べて HDL 記述のレベルが高いため高速に実行できる．またマイクロプログラム検証用のシミュレータは従来，一般的にC言語などで構築される場合が多く，論理設計用シミュレータに比べて性能が1桁以上速い．

第三に HDL との親和性である．高速のシミュレータであっても例えばC言語から論理合成用 HDL への言語変換により使用上低速になったのでは意味がない．統一環境におけるシミュレータの選択基準は最も時間のかかる論理シミュレーションが高速に実行できることである．具体的目標性能を，開発設計者が 100 KG（キロゲート）の論理シミュレーションを一晩(12 時間）以内に実行できる時間に設定した．このような観点から統一設計環境では，論理合成環境の機能/論理検証を行うシミュレータをマイクロプログラム検証用のシミュレータとして兼ねるのが良く，論理設計分野で広く使われている高速のシミュレータである例えば CADENCE 社の Verilog-XL[19]シミュレータなどの活用を検討した．Verilog-XL のシミュレータインターフェースは HDL 記述とシミュレーションコマンドが区別なく使用できる豊富な機能を持ち，実行時のステートメントに関する情報も詳細に表示されるため，ディバッグなどがやり易く，機能面ではマイクロ命令のシミュレーションに適用できる．そこでこれと従来の専用シミュレータ（C言語ベースの専用記述言語使用）との性能面での評価を実施した．評価記述モデルとして 50 KG 程度の CPU を両方の言語で記述し，10 MIPS のワークステーションで簡単な機械語命令を 100 クロック分実行する方法をとった．記述のソースコード量は，Verilog HDL で約 4000 行，専用記述言語で約 3600 行である．この結果，Verilog-XL は 2.4〜3.1 倍遅いことが判明した．これは，当初予想値より良い．専用記述言語そのものがC言語をモディファイしたものであるので，当然の結果とも言える．しかしながら，マイクロ命令のシミュレーションは，数 10〜100 ステップ単位に行うことが多く，実行時間は数分程度で終わるのでこの程度の遅れであれば論理シミュレーションに比べてかなり速く，許容でき実用に耐え得ることが分かった．

3.3 論理合成環境

マイクロプログラム設計環境を論理合成環境に融合する形をとったため，ハードウェア設計を従来通りの設計手法で行うことができる．機能シミュレーションと論理シミュレーションが行えることと，更に HDL 機能記述結果が論理合成にそのまま接続できることが必要である．現在活用できる市販の論理合成ツールとして例えば，Synopsys 社の Design Compiler[20]等があるが，それらは予め論理合成を意識して[21]HDL 記述することが必要である．そこで本環境では，合成向 HDL 変換機能を論理合成ツールに付加することにより，HDL 記述に際して論理合成を特に意識する必要がないようにした（図2参照）．

3.4 マイクロプログラム設計環境

混在記述部からマイクロプログラム記述部を自動抽出することと，マイクロ命令単位にアセンブル，リンク処理を行い，シミュレーションできる環境を実現する．シミュレーション環境は，3.2 節で述べたように論理合成環境でのシミュレーション環境を共用する．

3.4.1 マイクロプログラムの言語仕様定義方式

ユーザが入力する HDL 形式のマイクロプログラムの言語仕様定義方式を2つの点に重点をおいて検討した．1つは，入力データの定義をテキスト形式でできることに加えて，ユーザの使い勝手を向上させるためにグラフィック画面から視覚的に入力することも可能とする．この実現のため言語仕様定義エディタとして図2に示すテキストエディタと画面エディタの2つを開発する．もう1つの点は，新しいマイクロプログラムの開発を行う場合，一般的には古い設計データのマイナチェンジで実現できるケースが多く，設計データの再利用が必須となる．この実現のため，言語仕様定義エディタに HDL 形式の仕様定義情報を抽出して，アセンブラ，リンカが使用できる形にフォーマット変換した後，定義情報テーブルに登録できる機能を持たせた．これにより，言語仕様の変更が HDL 記述や定義情報テーブルの内容を再定義するだけで簡単に行うことができ，異なる機種（仕様）の新しいマイクロプログラムを容易に開発できる．言語仕様定義での主な定義項目を以下に示す．

①マイクロ命令のフォーマットに関する定義情報で，マイクロ命令語の長さ，フィールド名，ビットアサインなどがある．

②マイクロ命令のシンボル定義に関する情報で，HDL 記述に使用するニーモニックコードとビットパターンおよびそのビットパターンを埋め込むフィールド間の関係などがある．

③各マイクロ命令へのアドレス割付けに関する情報

で，アドレスビット長，アドレスを指定するニーモニックの定義，実行アドレスの順序指定の定義，多方向分岐アドレス割り付けの定義などがある．

④マイクロプログラム記述の際の文法に関する情報．

3.4.2 アセンブラ

アセンブラは，マイクロプログラムの定義情報に従い，マイクロプログラムのソースをアセンブルしてオブジェクト形式にする．マイクロプログラム実行アドレスの順序制御方式には，直接アドレス指定方式（マイクロ命令のアドレス部による指定）とプログラムカウンタ方式（マイクロプログラムカウンタによる指定）の2種類がある．マイクロプログラム開発環境に汎用性を持たせるため，本システムではアセンブラ，リンカはそのどちらの方式にも対応できるようにし，両者の区別は，オプション指定等で行う方式とした．またオプション指定によりアセンブルリスト，クロスリファレンスリストを画面に出力することも可能とする．

3.4.3 リンカ

リンカは，アセンブラから出力されたマイクロプログラムのオブジェクトに対して，絶対アドレスを割り付け，実行形式ファイルを作成する．リンカ関連のリストをユーザが見たい場合には，オプション指定によりシンボルマップリスト，クロスリファレンスリストを画面に出力できる．

3.4.4 フォーマット変換

マイクロプログラムとハードウェアのシミュレータを共用するため，リンカから出力されたマイクロプログラム実行形式ファイルをシミュレータへ入力できる形式にフォーマット変換する．

3.4.5 フローチャータ

マイクロプログラムの保守性とデバッグの容易性を高めるためマイクロプログラムの動作の流れを視覚的に把握できるようにするため，マイクロプログラムのソースファイルからフローチャートを自動生成するフローチャータを用意した．フローチャータはマイクロプログラムリストの各マイクロ命令を1ボックスで囲み，ボックス間を結んで流れ図形式で出力する．このようなボックスと線の結合により，多方向へ分岐するマイクロプログラムのフローを視覚的に把握することができる．フローチャータが出力する情報としては，ソースリストで記述したバージョン，命令ラベル，行き先ラベル，日付，設計者名，ページ番号のほか，ビットパターン，アドレス，コメント等がある．

表1 主な動作制御命令とその機能
Table 1 The control command and function.

制御命令	機能
BEGIN/END	アセンブル：先頭～終了範囲指定
REV	バージョンの指定
SET	リンクアドレスの割付（範囲／強制割付の指定）
ASSIGN	絶対アドレスの強制割付
LINKDATA	オブジェクト間のリンケージデータの定義
REXT/RENT	行先／来先ラベルの指定
TITLE/	フローチャートのタイトル／
LAYOUT ENTRY/ EXIT/SUB	レイアウトの指定 フローチャート編集（先頭／終端／サブルーチン図形の指定）

3.4.6 ドキュメンタ

マイクロプログラムに割り付けたラベルと，それに対応するアドレスをそれぞれソートした形で出力する．さらに，マイクロプログラムのビットパターンをアドレス順にソートして出力するほか，ソースプログラムを追加した形で出力することも可能とした．

3.4.7 マイクロプログラム環境の動作制御

マイクロプログラム環境に汎用性を持たせるために，ユーザがマイクロプログラムソースにコマンド形式で直接記述して制御できる方式とした．アセンブラ，リンカ，フローチャータの動作を指示する「動作制御命令」とマイクロ命令の分岐動作を指示する「分岐制御命令」の2種類を設定した．動作制御命令の代表的なものを**表1**に示す．分岐制御命令として設定した主なものは，次のマイクロ命令にジャンプする場合に行き先マイクロ命令を指定する GOTO，サブルーチンジャンプの時に呼び先マイクロ命令を指定する CALL，サブルーチンからの戻りを指定する RETURN などである．

3.5 マイクロプログラム検証

統一設計環境でのマイクロプログラム検証フローの概念図を**図5**に示す．各検証のステップごとに以下に示す正当性検証を可能とした．

①マイクロプログラム記述の正当性検証

マイクロ命令の HDL 記述形式，フィールド，ニーモニックなどマイクロプログラム定義情報に従って正しく定義されているかどうかのチェックを行う．

②マイクロ命令単位の動作の検証

HDL 形式のマイクロプログラム記述を含んだハードウェア機能モデル上で，ステップ単位の実行，さら

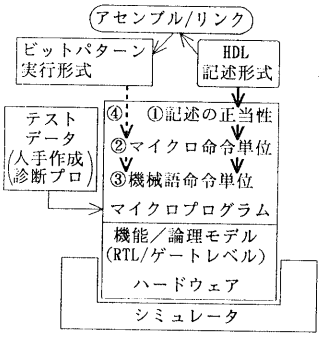

図 5 マイクロプログラム検証の概念図
Fig. 5 Microprogram evaluation flow.

にステップを継続してのブレークポイントまでの連続実行等により，1マイクロ命令単位の動作を確認する．

③機械語命令単位の検証

マイクロプログラムの集合体である機械語命令レベル，ならびに機械語命令の組み合わせ動作の検証を行う．シミュレータに入力するテストパターンは，人手または診断プログラムにより自動生成する．

④ビットパターン形式での検証

手順①②③の実行結果，HDL 記述のマイクロプログラムの正当性が確認されると，マイクロプログラムに関する HDL 記述部を抽出して，アセンブル，リンクにより実行形式（ビットパターン形式のバイナリデータ）に変換する．手順④では，HDL 形式のものをビットパターン形式のものに代えて，再度手順②③を繰り返し実行（シミュレーション）し，ビットパターン形式での動作の確認を行う．

なお，これらのマイクロプログラムはテストデータと共にシミュレータへの入力データとなり，シミュレータの制御の下にハードウェアの機能/論理モデルに従いシミュレーションが実行される（図5参照）．

4. 応用と評価

本統一設計環境を実システム開発に適用した結果について述べる．

4.1 適用機器の概要

特定用途向け専用シグナルプロセッサのマイクロプログラム方式 16 ビット CPU 開発に適用した．

マイクロプログラムの構成は，48ビット×2KW（キロワード）の垂直型，CPU-LSI の論理規模は約 80 KG，0.8 μm CMOS テクノロジを使用している．**図6**にマイクロ命令の構成例を示す．

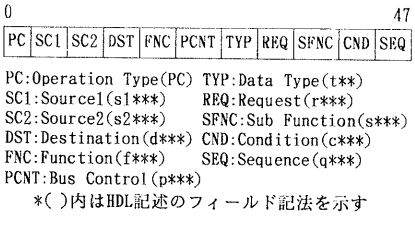

図 6 マイクロ命令の構成例
Fig. 6 Micro-instruction format.

```
begin //HW ex unit
@(posedge ck1)
  case(ex_st2)        機能記述
  A:begin             ALU動作
     ex_st1=B;
     ex_fnc_C;
    ----
  endcase
case(mic)     マイクロ命令 演算
^STAT   :mir={s1nop, s2nop, dnop,
         fnop, pnop, tsp, rnop,
         sp1f1, cnop, qcont, pc };
^STAT+1 :mir={s1nop, s2nop, drac,
           fnop, pnop, tsp, rnop,
           snop, cnop, qcont, pc};
         ----
endcase
begin //HW mr unit 機能記述
        ----        シーケンス制御
end
```

図 7 機能記述と混在したマイクロ命令の HDL 記述例
Fig. 7 HDL design mixing logic and micro-instruction.

4.2 適用結果

機能記述と混在したマイクロ命令の HDL 記述例を**図7**に示す．begin//HW ex unit で始まる機能記述は ALU 動作を，begin//HW mr unit で始まる機能記述はシーケンス制御を表す．演算を実行するマイクロ命令はマイクロ命令の開始を示す case(mic) とマイクロ命令の終了を示す endcase で囲まれた範囲で表現される．記述は STAT などアドレスを示すラベル名で表現されるマイクロ命令単位に行い，各フィールド対応のシンボル名や定数，オペレーションコードなどのニーモニック形式で表現される．HDL 記述に対するマイクロシミュレーション性能の結果は，約 1,100 秒（18 分）であった．測定条件は実行機械語命令数約 4,000，実行クロック数約 90,000，の診断（検証）プログラムを 22 MIPS マシン上で実行させ，汎用レジスタ 16 個，専用レジスタ 4 個を 1 命令ごとに表示させた場合である．

この HDL 記述形式のマイクロ検証が終了したのち統一設計環境の抽出プログラムは，case(mic) と endcase で囲まれたマイクロプログラム記述部を抽出して，アセンブル，リンクを開始する．リングが終了すると結果は，ロードモジュールとしてバイナリのビッ

```
[  1] 0 STAT  0000 400000000d02
[  2] 0 STAT1 0001 400400000002
[  3] 0 STAT2 0002 400000000002
[  4] 0 STAT3 0003 500002480002
[  5] 0 STAT4 0004 800104000416
  ⋮      ⋮     ⋮        ⋮
```

図 8 リンカ出力例
Fig. 8 Linker output data.

```
[  1] @0000 400000000d02
[  2]     1 400400000002
[  3]     2 400000000002
[  4]     3 500002480002
[  5]     4 800104000416
  ⋮       ⋮       ⋮
```

図 9 シミュレータ入力データ例
Fig. 9 Simulation data.

表 2 性能と工数
Table 2 The performance and manpower.

	マイクロ	論 理
実行性能	18 分	11時間
開発工数	3人月	42人月

トパターン形式で出力される．リンカの出力例を**図 8** に示す．ビットパターン形式マイクロプログラムの検証を行う場合は，フォーマット変換によりシミュレータ入力データに変換する．**図 9** にシミュレータ入力データ例を示す．一方，論理合成環境を用いたハードウェア設計は，マイクロプログラム記述部を除いた予め論理合成を意識して RTL 記述されている機能記述部に対して機能シミュレーション実施後さらに論理合成を行い，論理合成後の回路に論理シミュレーションを実施して論理設計が完了する．論理シミュレーション性能は，入力パターン数約 120,000 で約 40,000 秒 (11時間) であった (**表 2** 参照)．

4.3 評 価

論理合成環境とマイクロプログラム設計環境を融合したことの効果として，開発期間の短縮があげられる．CPU 全体の開発期間は 14 か月，開発工数は 45 人月で，内マイクロプログラム言語仕様の定義/作成工程から設計/検証までの工数としては，3人月であった．従来では，マイクロプログラム専用の検証環境の構築にさらに5人月の開発工数が必要となるが，この構成法では論理合成環境との共有化が図れるため，この部分の開発工数を0にすることができた．

シミュレータの実行性能面では，従来のC言語ベースのマイクロプログラム専用のシミュレータの方がこの構成法での Verilog-XL シミュレータに比べて高速に実行できることは分かっていたが，マイクロシミュレーションの実行時間は 22 MIPS の計算機で約 18 分と約 11 時間を要する論理シミュレーションに比べてかなり短いことも事実である．マイクロプログラムの容量が増えて 10 KW 程度になっても数時間以内に終了することが予測され，実用に十分耐え得るとの見通しが得られた．

さらに言語を標準の HDL(Verilog HDL) に統一したことにより，過去の設計データの流用性は勿論のこと，異なった機種間での設計データの流用性が向上する効果が予測される．統一設計環境による効果をまとめると，(1)設計期間の短縮，(2)設計財産の共有化と流用性，があげられる．

5. おわりに

論理合成とマイクロプログラム設計を融合したトップダウン設計手法指向の統一設計環境の一構成法について述べた．マイクロプログラムの作成からシミュレーションまでの開発環境とハードウェア機能/論理シミュレーションから論理合成/レイアウトまでの開発環境をドッキングさせることにより，マイクロプログラム専用のシミュレーション環境の開発工数を0にすることができた．また両者の設計記述言語に論理合成可能な標準のハードウェア記述言語 (HDL) を使用することにより設計の標準化，設計財産の蓄積，流用性が増加する効果も生まれた．本構成法を要約すると以下のとおりである．

(1)マイクロプログラム記述言語に論理合成可能な標準のハードウェア記述言語 (HDL) を使用する．ハードウェアの HDL 機能記述部は，レジスタトランスファレベル (RTL) を対象とし，人手変換なしに論理合成できる設計環境へ自動的に接続できる．マイクロプログラムの HDL 記述部は自動的に抽出され，マイクロプログラム設計環境に接続されアセンブル，リングできる．

(2)ハードウェア設計に用いる機能/論理シミュレーションモデルと，マイクロプログラム設計で使用するシミュレーションモデルを同一環境で提供する．

(3)マイクロプログラム設計環境は，多様な機種のマイクロプログラム開発に適用でき汎用性がある．

今後の課題としては，他の VHDL などの標準 HDL や，より上位言語への適用の拡大，マイクロプログラムの自動合成[10),22)]，ソフトウェア開発環境と

の融合化があげられる．ソフトウェア開発環境では，Cなどの設計言語が使用されているが，これらソフトウェア設計言語から直接論理合成できるツールは当分開発が期待できそうもない．現実的な解は，まず特定用途向きに範囲を限定してソフトウェア開発環境とのドッキングを図り，対応する設計記述言語自体もアーキテクチャ記述向け，RTL 記述向けなど目的別に階層化して，各設計レベルにあった設計記述言語に自動的に展開できる環境を提供する必要があると考えられる．

謝辞 本システムの応用開発を実施し，データを提供して頂いた三菱電機(株)鎌倉製作所の佐藤幸男氏，貴重なコメントを頂いた同コンピュータ製作所の須藤純吾氏に感謝いたします．

参考文献

1) 野地，清水，小山，国岡：トップダウン設計手法と論理合成，第 44 回情報処理学会全国大会論文集，4 E-1, pp. 6-149-150 (1992).
2) 安浦：特集「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，情報処理，Vol. 33, No. 11, pp. 1236-1243 (1992).
3) 今井：2. 2 VHDL，情報処理，Vol. 33, No. 11, pp. 1250-1255 (1992).
4) 野地：2. 4 Verilog HDL，情報処理，Vol. 33, No. 11, pp. 1263-1268 (1992).
5) 萩原：特集「マイクロプログラム技術」1. マイクロプログラム技術の進展，情報処理，Vol. 28, No. 12, pp. 1533-1539 (1987).
6) 元岡，菅野，渡辺，淵，石井：VLSI コンピュータI，p. 356, 岩波書店，東京 (1984).
7) 杉本：3. 2 マイクロプログラムの作成支援ツール，情報処理，Vol. 28, No. 12, pp. 1585-1594 (1987).
8) 馬場：3. 1 マイクロプログラム記述言語とその処理系，情報処理，Vol. 28, No. 12, pp. 1573-1584 (1987).
9) Skibbe, E. S.: Pace—A Microprogram Evaluation System, *Proc. MICRO 14*, pp. 181-191 (1982).
10) 木下，直井，今井：E同一化を用いたマイクロプログラム自動合成手続き，信学会論文誌 D-I, Vol. J 76-D-I, No. 11, pp. 566-574 (1993).
11) Dasgupta, S.: Hardware Description Language in Microprogramming Systems, *IEEE Conputer*, Vol. 18, No. 2, pp. 67-76 (1985).
12) Nixson, J. F. et al.: A Microarchitecture Description Language for Retargeting Firmware Tools, *Proc. MICRO 19*, pp. 34-43 (1986).
13) 中村，小栗：ハードウェア言語とその応用，情報処理，Vol. 25, No. 10, pp. 1033-1040 (1984).
14) Dam, A. V. and Barbacci, M.: Simulation of a Horizontal Bit-Siced Processor Using the ISPS Architecture Simulation Facility, *IEEE Trans. Comput.*, Vol. C-30, No. 7, pp. 513-518 (1981).
15) 星野，唐津：VLSI 統合設計記述言語 HSL-FX，信学会研資，SSD 85-80, pp. 103-110 (1985).
16) Samubrala, S. et al.: Design Verification of a VLSI VAX Microcomputer, *Proc. MICRO 17*, pp. 59-63 (1984).
17) Sasaki, T. et al.: MIXS: The Mixed Level Simulator for Large Digital System Logic Verification, *Proc. 17th DA Conf.*, pp. 626-633 (1980).
18) 須藤，野地，村田，小橋，杉本：ワークステーションをベースとしたマイクロプログラム開発支援システム，三菱電機技報，Vol. 63, No. 12, pp. 63-67 (1989).
19) CADENCE: Verilog-XL Reference Manual V 1. 6 (1991).
20) Synopsys: Design Compiler Reference Manual V 3. 0 (1992).
21) 野地，小山，清水：機能設計からの論理合成評価，信学会，VLD 91-136 ICD 91-222, pp. 59-66 (1991).
22) Mahmood, M., Mavaddat, F. and Elmasry, M. I.: Experiments with an Efficient Heuristic Algorithm for Local Microcode Generation, *ICCAD 90*, pp. 319-323 (1990).
23) 杉本，阿部，黒田，加藤：オブジェクト指向方式による対話型マイクロプログラムシミュレータ，信学論，Vol. J 70-D, No. 2, pp. 315-324 (1987).

(平成 5 年 10 月 25 日受付)
(平成 6 年 9 月 6 日採録)

野地 保（正会員）

昭和 22 年生．昭和 45 年長崎大学工学部電気工学科卒業．同年三菱電機(株)入社．同社計算機製作所，情報電子研究所を経て，現在，同社システム LSI 開発研究所勤務．平成 4 年より長崎大学大学院後期博士課程在学中．コンピュータ・アーキテクチャ，HDL ベースのトップダウン設計手法，論理合成の研究に従事．電子情報通信学会会員．

中村 彰（正会員）

昭和 6 年生．昭和 33 年九州大学工学部通信工学科卒業．日本電信電話公社において大型計算機の開発等に従事した後，現在，長崎大学工学部電気情報工学科教授．画像処理およびパターン認識等の研究に従事．電子情報通信学会会員．工学博士．