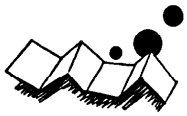
解　説

2. 方式・機能・論理設計における CAD



2.2 VLSI におけるマイクロプログラム設計支援†

迫田行介††　石原孝一郎††

1. はじめに

半導体プロセス技術の進歩は，VLSI の高集積度化を可能とし[10]，1チップ上に実現できる機能をますます拡大しつつある．マイクロプログラム制御方式[1)~5)]は，計算機の制御部の組織的設計法として 1951 年に M. V. Wilkes により提案され，すでに大型計算機やミニコンピュータでは広く使用されているものであるが，ランダムな論理と比較して制御部の複雑な論理を簡明にし，(1)論理設計ミスの発生率を低くおさえ，(2)ハードウェアの設計とマイクロプログラムの設計が並行してでき，(3)部分的な設計変更や修正も融通性のあるマイクロプログラムの変更，すなわち ROM などのビットパターンの修正だけですみ，ハードウェアの変更を最小限にすることができ，(4)論理構造が整然としているため小ブロックに分割して容易にシミュレーションができる，などにより設計工数を下げることができるばかりか，(5)素子が規則化されることによりレイアウト設計を簡単化し，(6)チップ内の制御部に要する面積の削減にも大きく寄与する，ということで VLSI の設計技術の1つとして重要な位置を占めるようになってきた．

VLSI のマイクロプログラムの設計手順は大型計算機あるいはミニコンピュータのそれと基本的に変るところはなく，必要なツールもほとんど同じものが使用できる．しかし大型機やミニコンピュータでは，マイクロプログラムは ROM あるいは RAM に書き込まれることが多いが，VLSI では ROM の他，面積削減のために PLA を使用することも多く，この場合 PLA パターンの圧縮のための最適化が大きな問題となる．

本文では，まず2章でVLSI設計におけるマイクロプログラム設計の位置付けとマイクロプログラムの設計手順全般について述べた後，3章でマイクロプログラムの設計支援のためのツールや技法について述べる．

† Design Tools and Techniques for Microprograms in VLSI by Kousuke SAKODA and Kouichiro ISHIHARA (Systems Development Laboratory, Hitachi Ltd.).

†† (株)日立製作所システム開発研究所

2. マイクロプログラム設計の流れ

VLSI の設計の流れは，おおむね図-1 に示すように，仕様設計，方式設計，論理設計，レイアウト設計となっている．論理設計はさらに機能設計（レジスタ・トランスファ・レベル）と詳細論理設計（ゲート・レベル）とに分けられるが，マイクロプログラム設計はほぼ機能設計と詳細論理設計とに並行して行われる．

実際には方式設計や機能設計のレベルでマイクロアーキテクチャが決定されていくが，マイクロプログラム設計中にマイクロプログラムの実行性能上の問題やその他の理由でマイクロアーキテクチャに変更が生じることが多い．したがってマイクロプログラム設計の初期の段階では，必ずしも完全に分離しない．マイクロアーキテクチャの設計に関しては本解説の範囲外とする．

マイクロプログラムの設計手順は，図-2 に示すように，ソフトウェアの設計手順と概念的には同じである．しかし，マイクロプログラムの場合には，マイクロプログラムがハードウェアを直接制御するものであるために，(i)ハードウェアからの制約が強い，(ii)性能が特に重視される，(iii)容量的制限が強い，(iv)ハードウェア設計者が設計することが多いなど，ソフトウェア設計の場合とは条件，環境が大きく異なる．これらはその支援ツールの機能にも大きく影響してくる．

以下，マイクロプログラムの設計手順を順を追って簡単に述べる．

(1) 機能設計

実現すべきマイクロプログラムの機能を各マイクロルーチンごとに明確にする．また，マイクロプログラム全体の構造や共通ルーチンなど下位のレベルのマイ

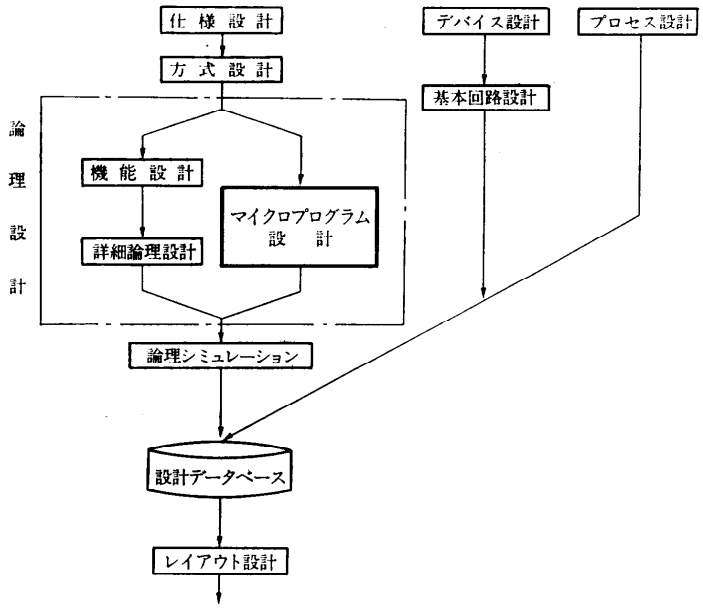


図-1 VLSI の設計フロー

クロルーチンのインタフェース設計も行う.

（2） 詳 細 設 計

実際の各マイクロルーチンの処理の手順をフローチャートやマイクロプログラム記述言語で記述する. アドレス割付のための情報やラベルなどを付記することもある.

（3） ソースファイル作成・保守

詳細設計されたマイクロプログラムはコーディングされ, ソースファイルとして入力される. ソースファイルの作成および修正や機能拡張などの保守を支援するツールとしては, 汎用のテキスト・エディタで十分であるが, バージョン／リビジョン管理機能のあるシステムが望ましい. ソースファイルから, フローチャート形式を再現して印字する清書ツールも使われる.

（4） トランスレート

マイクロプログラムのソースファイルを入力し, ビットパターン形式のマイクロコードに変換する. このときに文法チェック, ハードウェアの制約条件違反のチェック, アドレス指定違反のチェックなどの他, アドレスの自動割付, 最適化などの処理を行う.

（5） リンケージエディット

マイクロプログラムが大規模化してくると, マイクロルーチンに分割し, 何人かで分担して設計することになる. このような場合のためにマイクロルーチン単位に設計し, トランスレートしたマイクロ・オブジェクトを結合して1つのマイクロ・ロードモジュールにするマイクロリンケージエディタが提案されている[8),10)].

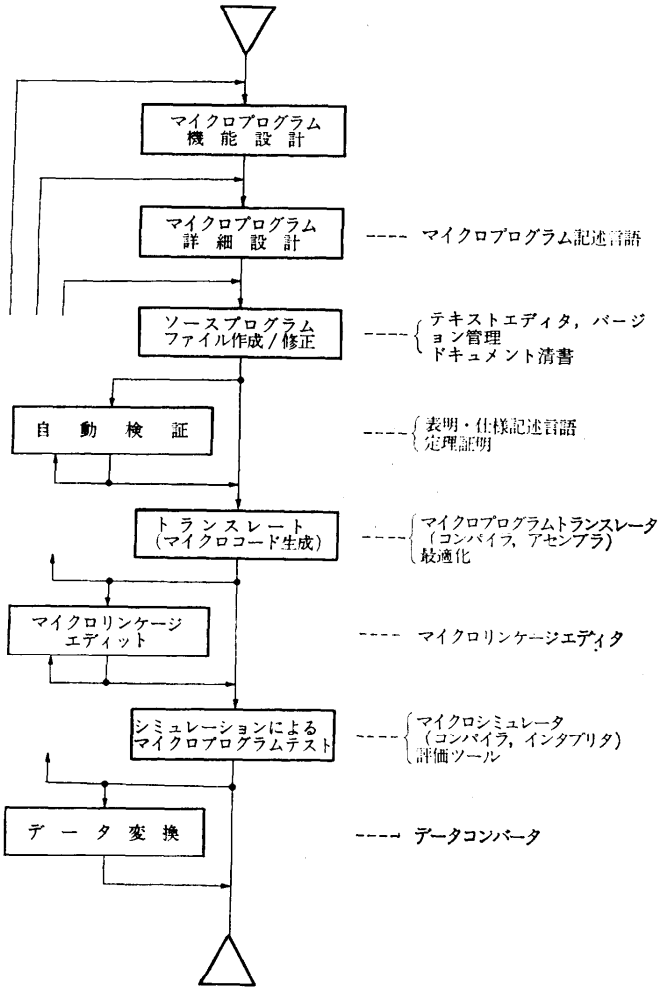
（6） 最 適 化

マイクロプログラムの最適化には, 主としてマイクロプログラムの実行性能向上を目的としたものと制御メモリ量削減を目的としたものとがある. 前者にはマイクロ操作の並列実行可能性を自動的に抽出してマイクロ命令を再構成するマイクロ操作圧縮（語方向の圧縮, 並列化, コンパクションなどともいう）があり, 後者には PLA を制御メモリに用いた場合の各種の PLA パターン圧縮法がある.

（7） テスト・評価

VLSI のマイクロプログラムのテストは, 他の論理回路と一体化して行う場合と, マイクロプログラムだけを単独で行う場合とが考えられる. 前者ではマイクロプログラムも1つの構成要素として扱われ, 論理シミュレータなどの支援ツールが使用される. 単独で行う場合でも後で一体化してテストすることが前提になるが, 詳細論理設計と並行してできることや比較的高速にできることなどの理由から行われることが多い.

また, マイクロプログラムをフロー解析して各種の



(a) マイクロプログラム設計フロー (b) マイクロプログラム設計支援

図-2 マイクロプログラム設計フローと支援

統計データや評価データを出力したり，動的なトレース結果と組み合わせてテスト・パス情報や網ら率を出力するツールも提案されている[26]．

(8) 変換ツール

マイクロプログラミングシステムの機能が充実してくると，いろいろなツールが作られ，それらを組み合わせて使用されることになり，必然的にデータ形式，コード系，媒体の変換ツールが必要になってくる．システムの規模が大きくなり，多目的になるほど変換ツールの数も多くなり，その開発・保守もかなりの手間になる．変換ツール開発用の標準言語なども支援しておくべきであろう．

3. マイクロプログラム設計支援

マイクロプログラム設計支援とは，前章の一連の設計作業を効率よく行っていくためのソフトウェア・ツール群あるいは技法のことである．設計支援ツールのうち，トランスレータやシミュレータなどは，機種に依存する部分があり，そこはハードウェア仕様の設計が終らないと決定できない．また，ハードウェア仕様の設計が終れば，すぐにマイクロプログラムの設計に必要となるため，短期間に開発することが要求される．機種に依存する部分がソフトウェアツールの中のいたるところにプログラムとして混在しているものはその機種専用ツールになり，きめ細かい機能を具備でき，性能もすぐれているが，開発に時間がかかるうえ，柔軟性がない．一方，機種に依存する部分をハードウェア仕様記述として一括して記述するような方式をとっている汎用ツールでは，性能は多少劣るものの，新機種への対応も早く，柔軟性も大きい．

本章では，VLSI のマイクロプログラムの設計手順中で重要な役割をはたしているトランスレータ，最適化ツール，検証ツールあるいはその技法について述べる．

3.1 トランスレータ

マイクロプログラムのトランスレータは，マイクロプログラムの記述言語を規定し，それで記述されたマイクロプログラムを入力し，ビットパターン形式のマイクロコードを生成する．その記述言語とトランス

手続向き機種独立言語 手続向き機種依存言語	高水準言語
マクロレジスタトランスファ言語 レジスタトランスファ言語 アセンブラ言語，フローチャート言語 マイクロ命令語	低水準言語

図-3 マイクロプログラム記述言語レベル
(文献 2) より引用)

レータの処理方式について述べる.

3.1.1 マイクロプログラム記述言語

マイクロプログラムの記述言語にはいくつかのレベルが考えられている[2] (図-3).

これらの言語については参考文献 3)～6) に詳しく述べられているので本稿では省略する.

3.1.2 トランスレータの処理方式

マイクロ命令語, アセンブラ言語を記述言語とするマイクロプログラムトランスレータの処理方式は明らかであろう. マイクロ命令のフォーマットや記号名称の値を別に定義できるようにして汎用化しているものが多い.

レジスタトランスファ言語の場合には, 上記の定義の他に, レジスタやフリップフロップなどの定義, バスやセレクタの接続関係, 条件判定や演算の種類, 定数の発生方法などの定義が必要であり, これらの情報を用いてレジスタ間のデータ転送パスの探索, リソースの競合チェック, アドレス割付などを行う.

高水準言語の場合には特定の機種専用のトランスレータと機種に依存しない汎用のトランスレータとがある. 専用のトランスレータでは, 言語で記述できる要素や操作に実際のハードウェアのリソースや演算機能を反映させることができるが, 通常の汎用プログラミング言語を流用したような場合にはどのようにそれらを対応付けるかが問題となる. しかし, 対象となるハードウェアが明確に分かっているので専用トランスレータの処理方式は比較的容易である.

汎用のトランスレータでは上記の対応付けが大きな問題となる. その処理方式は大きく次の3つに分類することができる[6].

(1)ある抽象マシンを仮定して定義した言語を実際のマシンのハードウェア仕様に従ってマイクロコードに変換するもの[27], このときハードウェア仕様で定義した記号名称を言語で使用できるようにしたもの[9], (2)核となる言語と言語の拡張機能により実際のマシンの機能を定義していくもの[28], (3)基本データ型と基本文など言語の枠組のみ与え, あとは実際のマシンの動作, 対応など細部を個別に定義していき言語を確定していくもの[6),29]などがある. しかし, 実用性から見て効率のよいマイクロコードが生成できるものは実現にいたっていない[7].

3.2 最 適 化

3.2.1 マイクロ操作圧縮

水平型のマイクロプログラムでは, 1マイクロ命令で複数のマイクロ操作を同時に実行することができることを利用し, 与えられたマイクロ操作系列の中から同時に実行できるマイクロ操作を検出し, データの依存関係を保存したままで, かつそれに必要なリソースやマイクロフィールドが競合しないようにしながら, できるだけ多くのマイクロ操作を1マイクロ命令の中に詰めていき, 結果としてマイクロ命令ステップ数を(同時にメモリ量も) 減少させる技法である[15),16].

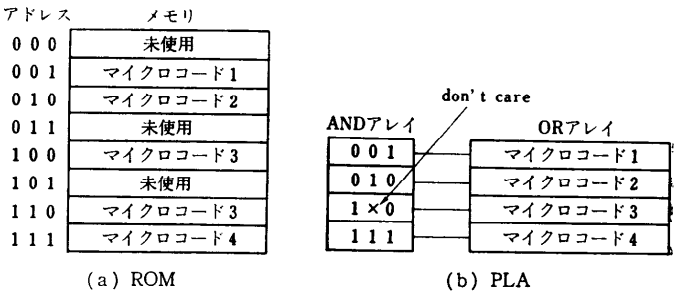
この問題は一般に NP 問題といわれ, 真の最小解を求めるのは現実的ではなく, 実用レベルでの極小解を求めるアルゴリズムが提案されている. 圧縮の単位をセグメント (入口が1つで出口以外に分岐がないマイクロ操作列, straight line microprogram ともいう) に限定した局所的圧縮法[12)~14)]と, セグメントにまたがって並列性を検出する効果の大きい大局的圧縮法が提案されている.

局所的圧縮法については参考文献 4), 5), 15), 16) などを参考にされたい.

局所的最適化では実際に対象となるマイクロ命令数が小さく, 並列化の効果が小さい[18]. Tokoro[17]は, セグメント間にまたがって移動できるマイクロ命令のパターンを示し, それに従った方法を, Isoda[19] はデータ依存関係グラフを, セグメント間にまたがって並列性を調べ, より並列性の高いグラフに変形していく方法を示した. どちらも熟練者が設計したものより平均5%前後の増加におさえられるという.

3.2.2 PLA パターン圧縮

VLSI 上のマイクロプログラムの占める面積を小さくし, 機能の柔軟性を高めるために ROM の代りに PLA を用いることが多い[20]. PLA は AND アレイと OR アレイの二段論理であり, ROM のアドレスデ



(a) ROM (b) PLA

図-4 制御メモリとして用いた ROM と PLA

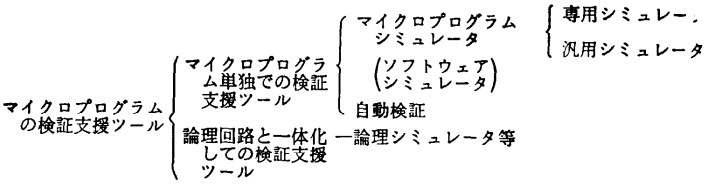


図-5 マイクロプログラムの検証支援ツールの分類

コーダが固定であるのに対してANDアレイを自由に設計することができるものである．この場合1マイクロ命令を1積項線に対応させ，マイクロコードをORアレイに，そのアドレスをANDアレイに格納する．したがってROMの場合と違って未割付アドレスを生じないばかりか，特定の入力線をdon't careにしておくことにより，マイクロコードが共用される（部分デコード方式）．この方式ではさらに積項線の数を減らすことができ，ひいてはマイクロプログラム占有面積を減らすことができる（図-4）．

ROM用のマイクロプログラム設計ツールを用いてPLAのアレイのビットパターンを求めるには，実際に割り付けられているマイクロコードだけを取り出してORアレイを構成し，それらの各アドレスのビットパターンからANDアレイのビットパターンを生成すればよい．部分デコード方式の場合にはアドレスにdon't careを指定する機能が必要となる．

この他，PLAに関しては従来から積項数を減らすように論理式を変換するアルゴリズムや，ANDアレイやORアレイの信号線を切断して2つに分割して畳み込む方法などが提案されており[21]，マイクロプログラムについても適用は可能である．

3.3 テスト

マイクロプログラムを単独でテストするための支援ツールにはマイクロプログラムシミュレータがある．この他自動検証の試みもある（図-5）．

3.3.1 マイクロプログラムシミュレータ

マイクロアーキテクチャを開発用の計算機上にマッピングし，ソフトウェアでシミュレートするものである[22),23)]．マイクロプログラムシミュレータについては参考文献3)～5)などを参照されたい．

3.3.2 自動検証

マイクロプログラムに関する自動検証の試みには，バローズ社のD-マシン用のSTRUM[24)]やナノデータ社のQM-1用のS*(QM-1)[25)]などがある．

STRUMについては参考文献4)にも詳しく述べられているので参照されたい．

4. おわりに

マイクロプログラム方式は，VLSIの制御回路の実現方式としてすでにその地位は確立されていると言える．しかしマイクロプログラム方式における新しい構成法や制御方式が次々と開発されており，それに対応した支援が必要になってくる．

今後VLSIに組み込まれるマイクロプログラムは，

（1） その制御メモリ構成が複雑な構造を持ったものになってくる，

（2） 1チップ上に乗せるマイクロプログラム総量が増大してくる，

という傾向にあることが十分予想される．

たとえば，チップ内の制御メモリの占める面積を小さくするために，分割型制御メモリなどをはじめ，階層構造を持たせたものにしたり，アドレスデコーダ（ANDアレイ）を自由に設定できる（命令デコーダと兼ねたりできる）PLAタイプの制御メモリが多く使われるようになる．この他，高速化を図るために次アドレス指定方法が複雑化してくることが考えられる．

また，1チップ上に乗せられる論理素子数が多くなるにともなって，マイクロ命令のビット幅が広がり語数も増大し，より多くの機能をチップ上に実現できるようになり，OSの一部をVLSIの中に取り込むようなことも考えられる．

このような傾向に対して，マイクロプログラム設計支援ツールに対する要求として

（1） マイクロプログラムの信頼性向上への支援機能

（2） マイクロプログラムの生産性向上への支援機能

（3） 支援ツール自身の柔軟性

がますます重要になってくる．

大容量のマイクロプログラムの開発はソフトウェアのそれに近い性質を呈するようになり，その検証は大きな問題となる．論理回路と一体化して論理シミュ

レーションするのではもはや手に負えず，並列設計の立場からもマイクロプログラムシミュレータによる早期検証が必須になろう．

マイクロプログラムの設計では，ハードウェアの特殊性を生かし，性能を引き出す必要があり，高度な専門知識と経験に頼るところが多いが，マイクロプログラム開発量の増大は非ハードウェア技術者や非熟練者による設計を必要とするようになるであろう．この場合でもマイクロプログラムの性能や生産性を確保するには，高水準言語の実用化が解決の鍵をにぎっているのかもしれないが，最適化や汎用化等解決すべき課題も多い．この分野では，近年その研究が盛んになってきた知識工学の応用[11]が有効であろう．

新技術に対応して新支援ツールを提供していくための開発環境を整えるとともに，支援ツール自身を柔軟性のあるものにし，かつ各ツール間の整合性をよくしておくことが望まれる．この意味でもツール開発治工具としては，開発効率が高く，柔軟性のあるPROLOGなどは有望である．

参 考 文 献

[マイクロプログラミング全般]

1) Husson, S. S.: Microprogramming: Principles and Practices, Prentice-Hall (1970).
2) Agrawala, A. K. and Rauscher, T.: Microprogramming: Perspective and Status, IEEE Trans. Comput., Vol. C-23, No. 8, pp. 817-837 (1974).
3) 萩原：マイクロプログラミング，産業図書 (1977).
4) 馬場：ファームウェア工学，情報処理，Vol. 20, No. 7, pp. 622-646 (1979).
5) 相磯，飯塚，坂村編：ダイナミック・アーキテクチャ，bit 臨時増刊8月号 (1980).

[マイクロプログラミング言語/トランスレータ]

6) Dasgupta, S.: Some Aspects of High Level Microprogramming, ACM Computing Surveys, Vol. 12, No. 3, pp. 295-323 (1980).
7) Sint, M.: A Survey of High Level Microprogramming Language, MICRO 13, pp. 141-153 (1980).
8) Meyers, W. J.: Design of a Microcode Link Editor, MICRO 13, pp. 165-170 (1980).
9) Baba, T. and Hagiwara, H.: The MPG System: A Machine-Independent Efficient Microprogram Generator, IEEE Trans. Comput., Vol. C-30, No. 6, pp. 373-395 (1981).
10) Nash, J. and Spak, M.: Hardware and Software Tools for the Development of a Microprogrammed Microprocessor, MICRO 12, pp. 73-83 (1979).
11) Shimizu, T. and Sakamura, K.: MIXER: An Expert System for Microprogramming, MICRO 16, pp. 168-175 (1983).

[最適化]

12) Ramamoorthy, C. V. and Tsuchiya, M.: A High-Level Language for Horizontal Microprogramming, IEEE Trans. Comput., Vol. C-23, No. 8, pp. 791-801 (1974).
13) Yau, S. S., Schowe, A. C. and Tsuchiya, M.: On Storage Optimization of Horizontal Microprograms, MICRO 7, pp. 107-118 (1974).
14) Dasgupta, S. and Tartar, J.: The Identification of Maximal Parallelism in Straight-Line Microprograms, IEEE Trans. Comput., Vol. C-25, No. 10, pp. 986-991 (1976).
15) 松崎：実用化の試みが始まったマイクロプログラムの最適化，日経エレクトロニクス，1978. 5. 1, pp. 76-91 (1978).
16) Landskov, D., Davidson, S., Shriver, B. and Mallett, P. W.: Local Micro Code Compaction Techniques, ACM Computing Surveys, Vol. 12, No. 3, pp. 261-294 (1980).
17) Tokoro, M., Tamura, E. and Takazuka, T.: Optimization of Microprogram, IEEE Trans. Comput., Vol. C-30, No. 7, pp. 491-504 (1981).
18) 迫田：拡張データ依存関係グラフを用いたマイクロプログラムの大局的並列化法，情報処理学会論文誌，Vol. 23, No. 3, pp. 304-311 (1982).
19) Isoda, S., Kobayashi, Y. and Ishida, T.: Global Compaction of Horizontal Microprograms Based on the Generalized Data Dependency Graph, IEEE Trans. Comput., Vol. C-32, No. 10 (1983).
20) 南谷：PLA の使い方，産報出版 (1978).
21) 田中：LSI 化に向いた論理設計法，日経エレクトロニクス，1981. 4. 13, pp. 122-144 (1981).

[マイクロプログラムの検証・評価]

22) 馬場，萩原，藤本，高橋，碇谷：MPG マイクロプログラム・シミュレータ，情報処理，Vol. 19, No. 5, pp. 412-420 (1978).
23) 倉地：マイクロプログラムの記述とシミュレーション，情報処理，Vol. 14, No. 6, pp. 397-403 (1973).
24) Patterson, D. A.: Strum: Structured Microprogram Development System for Correct Firmware, IEEE Trans. Comput., Vol. C-25, No. 10, pp. 974-985 (1976).
25) Wagner, A. and Dasgupta, S.: Axiomatic Proof Rules for a Machine-Specific Microprogramming Language, MICRO 16, pp. 151-158 (1983).
26) Skibbe, R. E.: PACE—A Microprogram Evaluation System, MICRO 15, pp. 181-191 (1982).

27) Malik, K. and Lewis, T. G.: Design Objectives for High Level Microprogramming Languages, MICRO 11, pp. 154-160 (1978).
28) Dewitt, D. J.: Extensibility—A New Approach for Designing Machine-Independent Microprogramming Languages, MICRO 9, pp. 33-41 (1976).
29) Dasgupta, S.: Towards a Microprogramming Language Schema, MICRO 11, pp. 144-153 (1978).

(昭和59年6月27日受付)