設計自動化28−5
(1985・9・17)

# オブジェクト指向言語ＶＥＧＡによるマイクロプログラム開発支援システム

# A Microprogram Development System Using an Object−oriented Language VEGA

杉本明＊　阿部茂＊　黒田正博＊＊　加藤幸男＊＊
Akira SUGIMOTO　Shigeru ABE　Masahiro KURODA　Yukio KATOU
(　＊三菱電機（株）中央研究所　＊＊同計算機製作所　)
Mitsubishi Electric Corp.

## 1．はじめに

マイクロプログラミングは計算機のハードウェア制御部の組織的設計法として、大型計算機からマイクロコンピュータに至るまで広く採用されている。ところが、マイクロプログラムは効率の良さが特に要求され、しかも多数のリソースに対するマイクロ操作の並列性や実行のタイミングを考慮する必要がある。このため、マイクロプログラム開発支援システムに関する研究が従来から盛んに行われてきた〔1-10〕。

しかしながら、従来の研究はマイクロプログラム記述言語の高水準化〔3〕や最適化アルゴリズム〔4〕、知識工学利用による自動作成〔5〕、汎用シミュレータの開発〔6〕等に見られる通り、自動化とその汎用化を主な目的としていた。ところが、近年のＬＳＩ技術の進展によりマイクロプログラムで制御するハードウェアが複雑・多様化することはもちろん、マイクロプログラム制御自体も複雑・多様化しつつある。従って、今日でもマイクロプログラム作成は対象ハードウェアに精通した熟練技術者にとってさえ非常に時間を要する煩雑な作業になっている。

これに対して本報告では、マイクロプログラム作成を人間・機械系の知的生産としてとらえ、対話性の向上によりその生産性を上げるアプローチをとる。我々は、デジタルシステムの図的モデル作成用として、オブジェクト指向言語ＶＥＧＡをＬＩＳＰをベースとして開発している〔11-12〕。そして、ハードウェアの設計階層における種々のレベルについてオブジェクト指向によるモデル化を試みてきた。その中で、マイクロアーキテクチャレベルのハードウェアについても、ＶＥＧＡによりモデル化を試みている〔13〕。そして、今回、これを用いシミュレータを中心としたマイクロプログラム開発支援システムを作成した。

本システムは、従来のマイクロプログラム開発支援システムやシミュレータ〔6-10〕に見られない以下のような特徴を持ち、対話性を向上させている。

（1）ビジュアルシミュレーション　−−　対象ハードウェアをカラーグラフィックディスプレー上にモデル化し、画面上のモデル要素に対する直接操作を基本とすることにより操作性を高めた。また、シミュレーションの進行に伴い、モデル要素を状態に従って画面上で変化させ、難解な水平型マイクロプログラムの並行動作の直感的把握を容易にした。

（2）エディタとシミュレータの統合化　−−　マイクロプログラムアセンブラのエディタとシミュレータを統合化した。これにより、マイクロプログラムの変更がすぐにシミュレーションに反映し、シミュレーションの結果によりすぐにマイクロプログラムに修正を行うことができる。

（3）デバッグ情報の多角的提示　−−　データの数値情報はもちろん、データ間の依存性や履歴、数式表現等、デバッグのための高度な情報を多角的に提示できる。多様な情報は問題点の把握を容易にする。

（4）シミュレーションのバック機能　−−　シミュレーション中に任意の時点に戻って詳しく状態を調べたり、状態を変えてそこから再実行する機能を持つ。これにより、マイクロプログラムのアルゴリズムの不備を容易に発見できるようにした。

そして、本システムを実際のマイクロプログラム開発現場で使用し、これらの特徴の有効性を確認した。

２章では。本システムの背景について述べる。その中で、本システムで使用したオブジェクト指向言語ＶＥＧＡの目的、特徴を説明する。また、マイクロプログラム開発の対象とした内蔵型ベクトルプロセッサＭＥＬＣＯＭ７０　ＭＸ／３０００・ＳＰ〔14-16〕についても簡単に紹介する。

３章では、上記の特徴を中心に本システムのユーザインタフェースについて具体的に述べ、その実現方法を示す。マイクロプログラムは対象機種に密接に依存するため、機能の実現方法が重要である。すなわち、対象機種に依存する部分が独立な部分と分離され、かつ小さいことがマイクロプログラム開発支援システムの汎用性の点で鍵となる。機能の実現方法から、本システムを他機種に適用することが容易であり汎用性を持つことを示す。

最後に、４章では、本システムに対し実使用に基づく考察を加え、上記特徴を評価し、問題点について論じる。

## ２．本システムの背景

ＶＥＧＡによるマイクロプログラムレベルのハードウェアのモデル化については別途報告している〔13〕。ＶＥＧＡは、設計対象のモデル化と検証を、グラフィック・ディスプレーを使用したビジュアルな環境で支援する図的モデル作成用言語である。個々のモデル要素の動作、形を自然かつ簡潔に記述できるように、オブジェクト指向方式〔17〕を採用した。

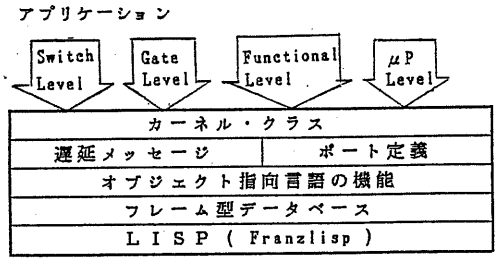


図１　ＶＥＧＡの構成レイヤ図

図１に、ＶＥＧＡの構成レイヤ図を示す。まず、ＬＩＳＰ言語（Franzlisp〔18〕）を全体のベースに、フレーム型データベースを構築している。次に、それをプログラム情報の管理に使用し、オブジェクト指向方式の諸機能（クラス定義、多重継承、メッセージ）を効率重視により実現している。次のレイヤがＶＥＧＡの特徴をなす機能である。ここでは、ハードウェアのモデル化に必要な『タイミングと並行性』及び『ローカルなデータの流れ』を記述するため、従来のオブジェクト指向方式に以下の拡張を行っている。

（１）遅延メッセージ（リクエスト）　－　メッセージ転送を特定の遅延時間とプライオリティにより行なえるようイベントととして登録でき、その時はメッセージの応答を待つことなく処理が進む。

（２）ポート定義　－－－　外部に公開された変数としてオブジェクトにポートが定義できる。また、ポートごとに、そのポートからローカルに入力されたメッセージに応答するメソッドが記述できる。

最後のレイヤとして、図的モデル作成用のカーネルクラスが用意されている。ここには、カーソルやウィンド等のクラスと共に、グラフィック機能を中心に図的モデル要素の雛型が定義されている。

今回、マイクロプログラム開発の対象とした計算機は、ＭＥＬＣＯＭ７０　ＭＸ／３０００に導入された内蔵型ベクトルプロセッサ（Scientific Processor，以下ＳＰと略す）である。ＳＰは、基本処理装置とキャッシュパスを共有した演算パイプラインユニットである。ＳＰは演算パイプラインのハードウェアブロックの役目を果たす。ＳＰの演算器にデータを供給する役目（アドレス生成やメモリアクセス制御）はＭＸ／３０００のＣＰＵの中にハードウェアとして装備されている。従って、マイクロプログラムはＭＸ／３０００のＣＰＵとＳＰの両方を同時に作成する必要がある。どちらも６４ビット幅のビットパタンによる水平型マイクロプログラムである。

## ３．ビジュアルインタフェースとその実現方法

図２、図３に本システムのグラフィック画面を示す。これは、マイクロプログラムのレベルから見たハードウェアの構成図を中心としている。マイクロプログラム作成やシミュレーション時に最も参照するのが、このハードウェアの構成図である。そこで、ユーザインタフェースモデルによる対話性の向上［19］を狙いとして、マイクロプログラムレベルのハードウェア図的モデルを本システムの中心に置いた。

画面上に現われた個々の要素は、ＶＥＧＡで定義したオブジェクトである。オブジェクトは画面上での形、色以外にその固有の機能を持つ。特に、ハードウェアブロックを表現したオブジェクトは、〔13〕で報告した

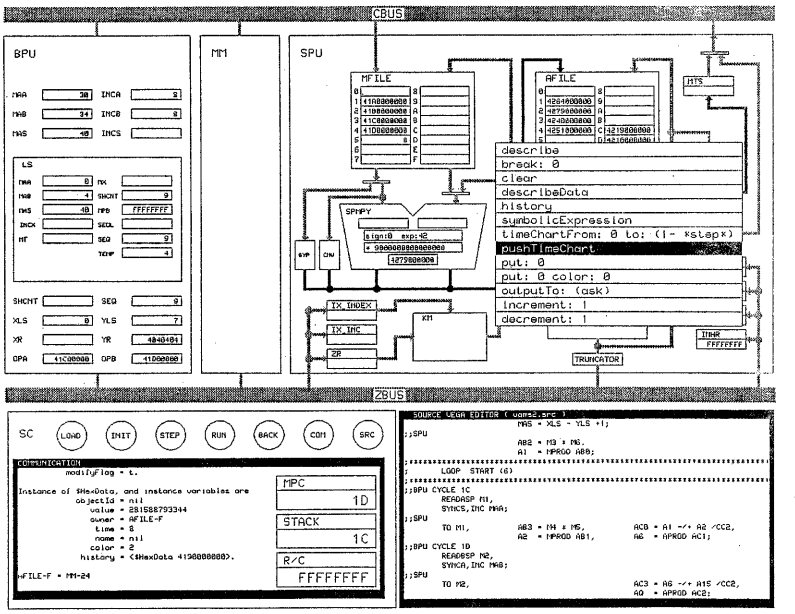


図２　グラフィック画面の例

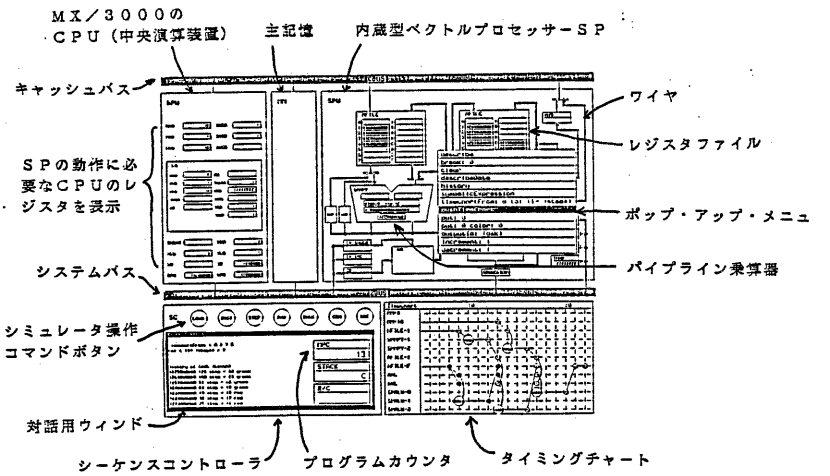


図３　画面の説明図

ように、オブジェクト間のメッセージやリクエスト（遅延メッセージ）で互いに通信し、マイクロプログラムレベルのハードウェア動作をシミュレートする。

図４に本システムのソフトウェア構成を示す。マイクロプログラムは後述する専用エディタを使用してアセンブラで記述し、ＬＩＳＰコマンドに変換される。ＬＩＳＰコマンドは、ハードウェアブロックを表現したオブジェクトに対するリクエストを生成する。ハードウェアブロックを表現するオブジェクトは、ＶＥＧＡのカーネルクラスから属性の継承（インヘリタンス）により定義する。この時、マイクロプログラムレベルのハードウェアに汎用的に使用するオブジェクトを共通クラスファイルにまとめ、ＭＸ／３０００のＣＰＵに特有なオブジェクトをＢＰＵクラスファイルに、ＳＰに特有なオブジェクトをＳＰＵクラスファイルにそれぞれまとめている。

このように、本システムではハードウェア要素をオブジェクト指向言語の差分記述方式により定義している。従って、対象ハードウェアの変更・拡張に容易に対処できる汎用性を持つ。また、グラフィック機能を持つＶＥＧＡのカーネルクラスを継承することで、図的モデルの作成が容易となる。

### ３．１．ビジュアルシミュレーション

シミュレーションの初期化・実行は図２左下のＳＣと呼ぶ制御部のボタン操作で行う。すなわち、ＳＣオブジェクトはハードウェアのマイクロ制御部として動作すると共に、シミュレーション自体も管理している。

シミュレーションの進行と共に画面が変化する。レジスタの内部の小さい窓には、常に保持しているデータの値がその色で表示されている。従って、レジスタの値が変ると表示も変化する。また、バスやワイヤの色は、データが流れるとデータの色に従い変化する。このように、シミュレーション中のデータの流れを視覚化することにより、難解な水平型マイクロプログラムの同時並行動作の把握を容易にしている。

シミュレーション中に画面に自動的に現れない情報は、ポップ・アップ・メニュにより対話的に得る。図５に、レジスタのポップ・アップ・メニュの例を示す。メニュの内容はレジスタへのメッセージである。例えば、ここで『describe』を選択するとそのレジスタの内部状態を表示する。同様にメモリやバスの内部状態も、そのオブジェクトに対応したメニュを通して見ることができる。

本システムではハードウェアの仕様変更にすぐ追随するために制御の記述とデータパス構造の記述を分離

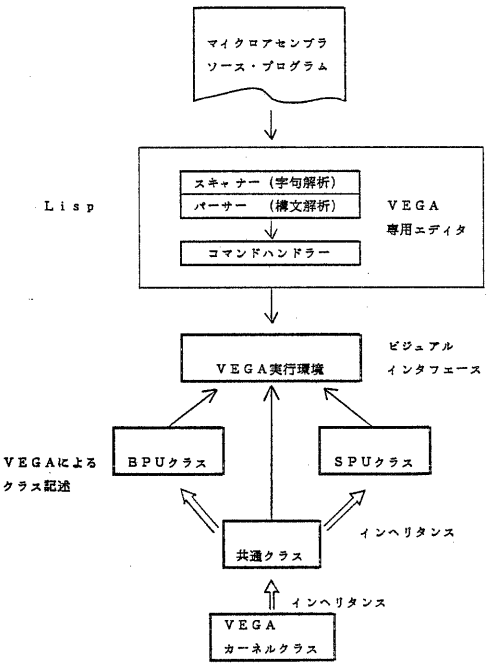


図４　ソフトウェア構成図

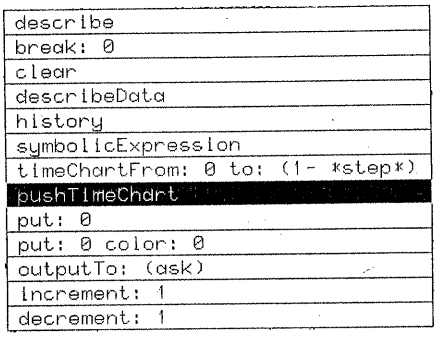


図５　レジスタのポップ・アップ・メニュ

している〔13〕。例えば、レジスタＡの入力ポート『in』からクロック相＜ｎ＞でデータを取込む場合は、

```
〔phase: 〈n〉 | A input〕
```

と記述し、Ａに対し“input”リクエストを発行する。

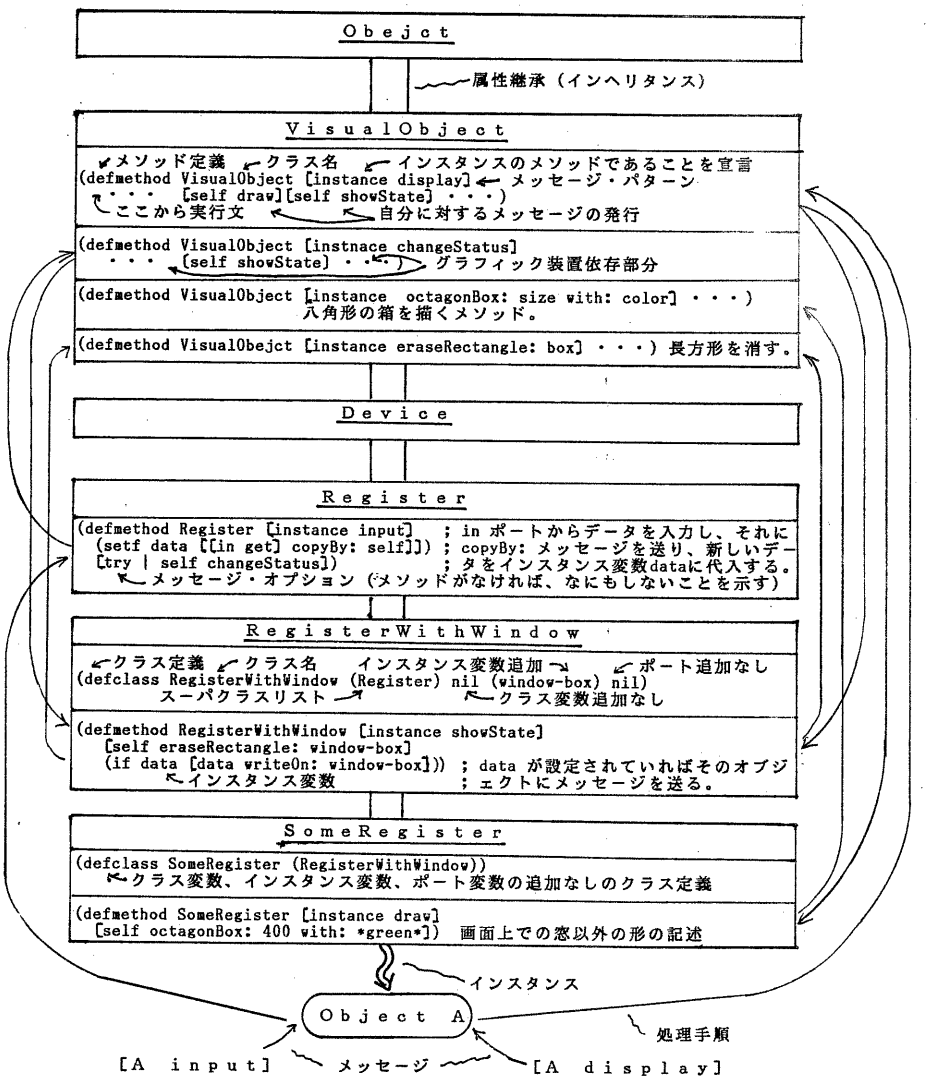


図６　クラス階層とＶＥＧＡプログラムの例

いま、Ａがクラス『SomeRegister』に属するものとする。図６にクラス階層とプログラム例により、“input”と画面上への表示メッセージ“display”の処理の流れを示す。カーネルクラス『VisualObject』に、グラフィックディスプレー装置に依存した部分を集約している。また、『VisualObject』のメソッドは１種のグラフィック言語をそのサブクラスに提供する。『Register』はハードウェア動作を担当し、その後、自分自身に対し“ChangeStatus”というメソッドを発行するだけでよい。『RegisterWithWindow』は、窓にデータを表示する。『SomeRegister』には、形の描き方だけを『VisualObject』のメソッドにより定義する。もちろん省略した場合、標準の形となる。

このように、ハードウェアブロック内部の動作記述とグラフィック部分が分離している。一方、ハードウェア外部での制御記述ではグラフィック操作を一切意識しなくても良い。また、グラフィック部分もモデュラリティ良く分割している。

### ３．２．エディタとシミュレータの統合化

マイクロプログラム作成には専用アセンブラ言語を使用する。このプログラムの作成及び編集のため、図７のような専用エディタが用意されている。専用エディタも画面に現われた一つのオブジェクトである。

```
; SOURCE VEGA EDITOR ( vama2.src )
;*****************************************************************
;;BPU CYCLE 1C
        READASP M1,
        SYNCS,INC MAA;
;;SPU
        TO M1,          AB3 = M4 * M5,          AC0 = A1 -/+ A2 /CC2,
                        A2  = MPROD AB1,        A6  = APROD AC1;
;;BPU CYCLE 1D
        READBSP M2,
        SYNCA,INC MAB;
;;SPU
        TO M2,                                  AC3 = A6 -/+ A15 /CC2,
                                                AQ  = APROD AC2;
;;BPU CYCLE 1E
        WRITESP,
        SYNCB,INC MAS;
;;SPU
        FROM AQ,        AB0 = M1 * M6,
                        A3  = MPROD AB2;
```

図７　専用アセンブラエディタ

専用エディタは、マイクロ命令単位でアセンブラ言語をＶＥＧＡの内部表現に変換する。その際、シンタックスチェック及びビットパターン生成時のコンフリクトチェックも行なう。エラーが起ると、アセンブラソースの対応する位置がエディタ上に示される。また、専用エディタはシミュレーションシステムと連動している。ＶＥＧＡの内部表現に変換されたマイクロプログラムは、直接、ＳＣ（マイクロ制御部）のプログラムメモリに格納される。また、シミュレーション中の実行位置を、エディタ中のアセンブラソースに表示することもできる。シミュレーション中にマイクロプログラムソース中の修正箇所が見つかると、その部分を専用エディタで直す。そして、プログラムメモリに部分的に再格納し、シミュレーションを再開する。このようなシミュレータとエディタの連動により、実行エラー時のマイクロソースの早い修正が可能である。

専用エディタは、ＶＥＧＡのカーネルクラス『スクリーンエディタ』のサブクラスとして定義し、上記のマイクロプログラミング支援機能を追加している。アセンブラソースの字句・構文解析はＬＩＳＰにより、ソース中の注目箇所の表示は『スクリーンエディタ』の機能により実現している。

### ３．３．デバッグ情報の多角的提示

エラー箇所の早期検出や、マイクロプログラムのアルゴリズムの改良のためには、シミュレーションによる結果（途中結果）を、豊富にかつ多様な形式でユーザに提供する必要がある。

図８はデータのタイムチャートの例である。縦軸がデバイスを、横軸がステップ数を示す。ＭＭ－１８はメインメモリの１８番地、ＭＦＩＬＥ－１はレジスタファイル（ＭＦＩＬＥ）の１番レジスタである。パイプライン回路（SPALU と SPMPY）については、表示上の理由でデータ入力の時点によりデバイスを３分割している。この図は、レジスタ（ＡＱ）のデータがどような時点でどのような操作を受けて作られたかを表現している。この図からデータ間の依存関係が明白となる。また、一群のデータのタイムチャートを表示するとデバイスの使用状況もつかむことができる。

データ間の関係については、その数式表現〔10〕を得ることも便利である。本システムでは、図５のメニュの“SymbolicExpression”を選択すると、例えば、図

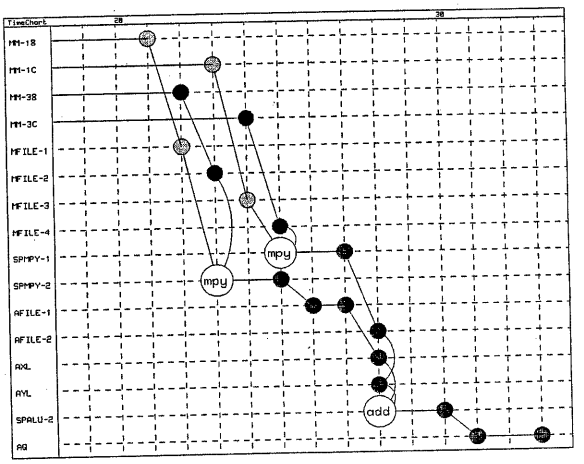


図８　タイムチャートの例

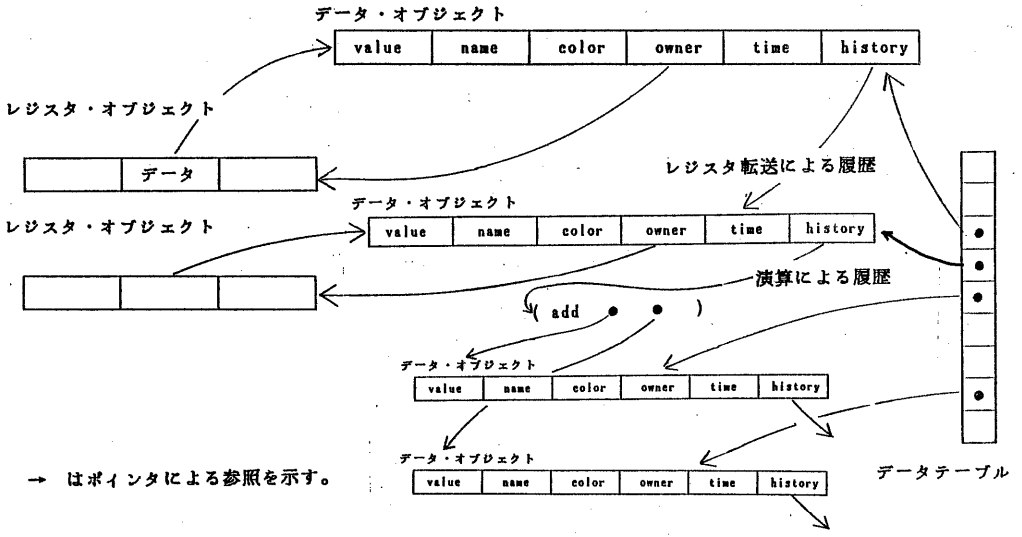


図９　データ・オブジェクトの構造

８のＡＱのデータからは、

```
AQ = (MM-18 * MM-38) + (MM-1C * MM-3C)
```

の数式表現が得られる。

データの履歴をデータ自体に保存し、シミュレータの他の部分と分離するため、本システムではデバイス間を流れるデータもオブジェクトとして記述している。

データ・オブジェクトは、図９のようにインスタンス変数として owner、time、history 等を持っている。owner 変数はどのレジスタに格納されているかを示す。time はデータが生成されたステップを、history はそのデータの履歴を示す。history 変数は、レジスタ転送によって生成された時は直接転送元のデータを参照し、演算により生成された時は演算種類と元になったデータを参照する。

### ３．４．　シミュレーションのバック機能

シミュレーション中に、状態を何ステップか元に戻したい時がある。データの値や動きがおかしく原因をつきとめるために戻りたい場合や、途中のソースの誤りを修正してそこから再実行したい場合等である。このような時は、ＳＣの『back』ボタンを押すと過去の任意の時点に戻ることができる。

データ間のhistory による連結では、最終結果に反映されたデータのみが保存されている。そこで、バック機能のため生成されたすべてのデータを図９の右のデータテーブルから参照できるようにした。データオブジェクトは生成された時間（time）と場所（owner）を持っているので、この情報と制御情報（リクエスト）の履歴から過去のシミュレーション状態に戻ることができる。

### ４．使用経験による考察

今回のシミュレータ開発はハードウェア開発と並行して進めたため、シミュレーション対象の拡大とともにレジスタの追加、データバリッドタイミング、データバスなどの拡張・変更があった。このような拡張・変更に対し、ＶＥＧＡではオブジェクトの生成、ポート定義の変更、遅延メッセージのタイミングの変更がインタラクティブにでき、ＶＥＧＡの有効性が確認できた。

また、マイクロプログラムのデバッグ時においてはハードウェアの状態推移がビジュアルシミュレーションにより一目瞭然であった。加えてタイミングチャートやバック機能によりマイクロプログラムのアルゴリズム上の不具合、コーディングミスを簡単に発見することができた。さらに、統合化された専用エディタとバック機能によりソースの修正と動作の再確認をする

ことができた。本システムで各々約５０ステップのマイクロプログラムを５０本以上デバッグした。それらは実機で簡単に動作し、動作確認として１人で１本／１時間以上の進捗であった。

今回対象とした内蔵型ベクトルプロセッサＳＰは、シーケンス制御、データリクエスト制御及びパイプライン演算制御を含めた多くのハードウェア・ブロックをネクスト・アドレス方式のマイクロ・プログラム（１２８ビット）で制御する必要があった。このように複雑なハードウェアのマイクロプログラム開発に本システムが有効であったということは、汎用のマイクロプログラム開発支援システムとしても有望を言える。本システムのソフトウェア量を表１に示す。

マイクロプログラムのアセンブラにＬＩＳＰのシンタックスを採用すると、処理がはるかに簡単となる。しかし、マイクロプログラム作成や保守部門の『慣れ』を考慮し、従来使用されてきたシンタックスをとった。

ハードウェアの図的モデルはできるだけ実際のハードウェアの構造に忠実な方が構成しやすく、ハードウェア仕様の変更にも対処しやすい。しかし、実際には、マイクロアセンブラのニーモニックとハードウェア・ブロックが一対一に対応しない場合がある。例えば、システムのイニシャライズ時にセットすればよいだけの制御レジスタなどの場合、複数の小さなビット幅のレジスタ、フリップフロップをまとめ１つのマイクロフィールドに割り当てることが多い。マイクロプログラマは、この複数のハードウェアレジスタ、フリップフロップをまとめあげた仮想的なレジスタを実際のレジスタと見てプログラミングする。このため、図１０に示すように仮想的なレジスタを画面上に表示し、実際のシミュレーションは個々のハードウェアレジスタで行った。

インターフェイスの仕様を設計するにあたり、実行効率と柔軟性のトレードオフが問題になった。本システムでは、マイクロプログラムのアルゴリズムの検証に主眼が置かれているため、各ステップの状態が詳細に観察できる必要があった。しかし、いかに多機能であっても、マイクロ命令１ステップ（マイクロ操作で４～１０個）の実行速度が５秒以上になると対話性を著しく阻害する。

表１　ソフトウェア量

| | |
|---|---|
| ・ ＶＥＧＡカーネルクラス | ２．０kstep |
| ・ その他のＶＥＧＡ環境 | ４．５kstep |
| ・ ＶＥＧＡ専用エディタへの追加機能（スキャナー，パーサー，コマンドハンドラー） | １．５kstep |
| ・ ＭＥＬＣＯＭ７０　ＭＸ／３０００－ＳＰ シミュレーションのためのクラス定義 | ２．５kstep |
| ・ オブジェクト総数（データを含まない） | １４４個 |

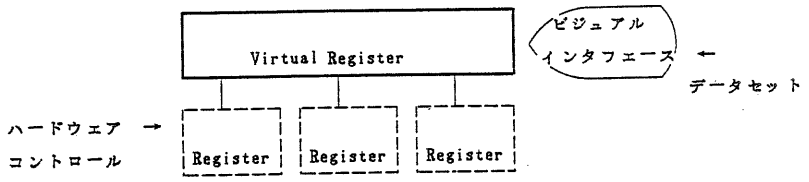


図１０　バーチャル・レジスタ

本システムのシミュレーションにおけるマイクロ命令１ステップ当りの平均実行時間を表２に示す。ここに見る通り、必要な時のみマイクロプログラムソースの参照を行なうならば、対話的に実行を進めるには十分な実行速度が保たれている。しかし、多様なデータに対し多数の反復計算をするようなケースがある場合には、実行効率をより重視したシミュレーション方式をあわせて持つ必要があるだろう。その場合、現在のインタープリタ方式とマイクロプログラムをあらかじめコンパイルする方式との併用が考えられる。

効率と柔軟性のトレードオフで一番問題となったのはバック機能である。過去の状態を再現するのためのメモリの消費量が大きい。しかし、使用上の効果はその欠点を上回るものがあり、エラーの原因の追求が容易であった。

## ５．おわりに

本報告では、対話性の向上によりマイクロプログラム作成の生産性を上げることを目指し開発した、オブジェクト指向言語ＶＥＧＡによるマイクロプログラム開発支援システムについて述べた。そして、従来のシステムには見られない、（１）ビジュアルシミュレーション、（２）エディタとシミュレータの統合化、（３）デバッグ情報の多角的提示、（４）シミュレーションのバック機能等の特徴について説明した。また、クラスの構成法やオブジェクト間の構造等、汎用性・発展性の高い実現方法を示した。さらに、本システムを実際のマイクロプログラム開発現場で使用し、上記特徴の有効性を確認した。

表２　１ステップ当りの平均実行時間

| | |
|---|---|
| グラフィック表示なし …… | ０．５秒 |
| グラフィック表示あり | |
| 連続実行 …… | １．８秒 |
| ステップ操作 …… | 約３秒 |
| ステップ操作及び専用エディタ上でのソースプログラムの参照 | 約８秒 |

## 参考文献


〔1〕 相磯、飯塚、坂村　編　『ダイナミック・アーキテクチャー』共立出版(1980).

〔2〕 迫田、石原、『ＶＬＳＩにおけるマイクロプログラム設計支援』情報処理、Vol.25, No.10, pp.1041-1047 (1984).

〔3〕 馬場、萩原他、『マイクロプログラム記述言語：ＭＰＧＬ』情報処理、Vol.18, No.6, pp558-565 (1977).

〔4〕 滝塚、田村、所、『マイクロプログラミングの広域最適化に関する考察』信学論（Ｄ）J62-D, No.3, pp.193-200 (1979).

〔5〕 清水、坂村、『知識工学に基づいたマイクロプログラム開発システム、ＭＩＸＥＲ』信学論（Ｄ）J66-D, No. 7, pp.864-871 (1983).

〔6〕 馬場、萩原他、『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).

〔7〕 Charlton C. et al. "An Interactive Software System for Microcode Development," North-Holland, Microporgramming and Microprocessing 13, pp.105-114 (1984).

〔8〕 Myers G. et al. "The Use of Software Simulators in the Testing and Debugging of Microprogram Logic," IEEE Trans. COMPUTERS, Vol. c-30, No.7, pp.519-523 (1981).

〔9〕 Dam A. et al. "Simulation of a Horizontal Bit-sliced Processor Using the ISPS Architecture Simulation Facility," IEEE Trans. COMPUTERS, Vol. c-30, No.7, pp.513-519 (1981).

〔10〕 Abbott C. "A Symbolic Simulation for Microprogram Development," IEEE Trans. COMPUTERS, Vol. c-32, No. 8, pp.770-774 (1983).

〔11〕 杉本、『デジタル・システム設計のための図的モデル作成言語ＶＥＧＡについて』　信学会研究報告　EC84-28 (1984).

〔12〕 A. Sugimoto, M. Fukushima, "VEGA: A Visual Modeling Language for Digital Circuit Design," Proc. IEEE ICCD'84, pp.807-812 (1984).

〔13〕 杉本、黒田、阿部　『マイクロプログラミングのためのオブジェクト指向によるハードウェアのモデル化』　信学会研究報告 EC85-4 (1985).

〔14〕 黒田、『スーパーミニコン内蔵ベクトル演算アクセラレータ』アーキテクチャワークショップインジャパン'８４，情処学会、pp73-82 (1984).

〔15〕 黒田、『MELCOM70 MX/3000　内蔵型ベクトルプロセッサ』情処学会第３０回全国大会　pp. 97 (1985).

〔16〕 高井他、『MELCOM70 MX/3000　マルチプロセッサ』情処学会第３０回全国大会　pp. 125, (1985).

〔17〕 米沢、『オブジェクト指向プログラミングについて』コンピュータソフトウェア Vol.1, No. 1, pp.29-41 (1984).

〔18〕 Foderaro J. et al. "The FRANZ LISP Manual," on-line documentation (1982).

〔19〕 Smith C. et al. "Designing the Star User Interface," in P. Degano and E. Sandewall(Eds.) "Integrated Interactive Computing Systems," North-Holland, pp.297-313 (1983).