計算機アーキテクチャ　103－15
設　計　自　動　化　69－15
(1993. 12. 17)

16ビットマイクロコントローラ(M16)の高速シミュレータの開発

那須 隆　岩田 俊一　清水 徹　斉藤 和則

三菱電機(株) システムLSI開発研究所
〒664 兵庫県伊丹市瑞原 4-1

あらまし

16ビットのマイクロコントローラM16とその高速シミュレータを開発した。M16は、32ビットのデータパスを備え4段のパイプライン処理によりレジスタ-レジスタ間の演算を1クロックで実行可能なCPUをコアにしている。このチップの開発では、ハードウェア記述言語を使用して機能設計を行なった。このハードウェア記述言語による機能記述を基に、より高速でかつハードウェアでの動作とクロック単位で一致するソフトウェアシミュレータを開発した。この高速なシミュレータにより、応用ソフトウェアの性能評価やチューニングなどを行なうことが出来た。

和文キーワード　マイクロコントローラ、ハードウェア記述言語、機能設計、ソフトウェアシミュレータ

Software Simulator of 16-bit Microcontroller M16

Takashi Nasu, Shunichi Iwata, Toru Shimizu, Kazunori Saitoh

System LSI Laboratory
Mitsubishi Electric Corporation
4-1 Mizuhara, Itami, Hyogo 664 JAPAN

Abstract

We developed 16-bit microcontroller M16 and its cycle-accurate software simulator. The core CPU has a 32-bit datapath and a 4-stage instruction processing pipeline. It executes register to register instructions at 1 cycle/instruction. Its functional design is described by a hardware description language. Based on this description, we developed the software simulator, which simulates the hardware functional behavior. The software simulator is much faster than simulation based on the hardware description language. This faster simulator made it more efficient to evaluate and tune performance of application programs on M16.

英文 key words　microcontroller, hardware description language, functional design, software simulator

1 はじめに

半導体製造技術の進歩により大規模なLSIが実現できるようになった。さまざまな機能を持つ大規模なLSIでは、LSI自体の開発に必要な工数だけでなく、そのLSIで動作させるソフトウェアの開発工数や性能に対する比重も大きくなってきている。このため、開発中のLSIに対するソフトウェアの先行開発が必須となり、ソフトウェア開発用のシミュレータを用意しデバッグを行なうことがなされてきた。しかし、単に正しく動作することを確認するだけではなく、性能評価をし、チューニングを行なうことが求められるようになってきている。

今回、我々は16ビットのマイクロコントローラM16の開発において、ハードウェア記述言語Verilog-HDLを使用したLSIの機能設計を行なった。このVerilog-HDLによる記述を基にハードウェアの動作とクロックサイクル単位で一致するシミュレーションを行なうソフトウェアシミュレータ(以下これを『パイプラインシミュレータ』と呼ぶ)を開発した。

本稿では、M16の概要を述べたあと、M16で採用したハードウェア記述言語による機能設計、ハードウェア記述言語のモデルを基にしたパイプラインシミュレータの開発について報告する。

2 M16の概要

M16は、OA機器やコンピュータ周辺機器などの組み込み制御をおもなターゲットとするマイクロコントローラである。表1にM16の諸元、図1にチップ写真を示す。

外部データバスは16ビット、アドレスバスが24ビットで16Mバイトのリニアアドレス空間となっている。32ビットのデータパスを備えたCPUをコアにして、RAM、タイマ、ウェイトコントローラ、DMAコントローラ、A/Dコンバータなどの周辺機能を集積している。

CPU部は、命令フェッチ、命令デコード、μROMアクセス、命令実行の4段のパイプラインで構成されており、レジスタ-レジスタ間の基本演算を1クロックで実行可能である。外部クロックの最高周波数は20MHzである。これをチップ内部では2分周して10MHzのクロックを生成している。このときの最小命令実行時間は100nsである。CPU部のトランジスタ数は13.8万で、これを0.8μm CMOSプロセスにより3.7×4.2mm^{2}に集積している。

表1 M16の諸元

命令セット	96命令
アドレス空間	24ビット (16Mバイトリニア)
動作周波数	外部20MHz (内部10MHz)
命令最小実行時間	100ns
プロセス	0.8μm CMOS
チップサイズ	8.83×8.78 mm^2
電源電圧	5V
消費電力	約300 mW

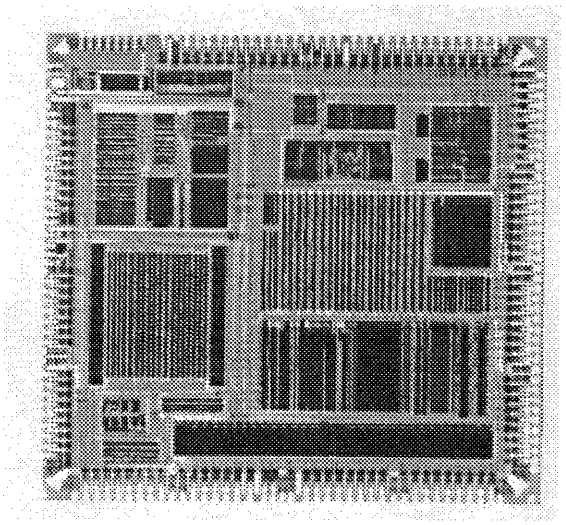

図1 M16チップ写真

3 機能設計

M16のCPU部の開発においては、ハードウェア記述言語を用いた動作モデルの作成を行なうことで、機能設計や機能検証を行なった。また、ハードウェア記述言語から論理回路を自動生成する論理合成ツールを使うことで、論理設計期間の短縮を図った。

論理合成ツールを使用する場合には、論理回

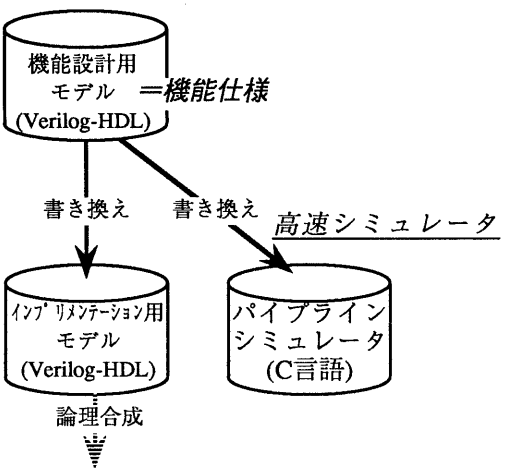

図 2　M16CPU の機能設計フロー

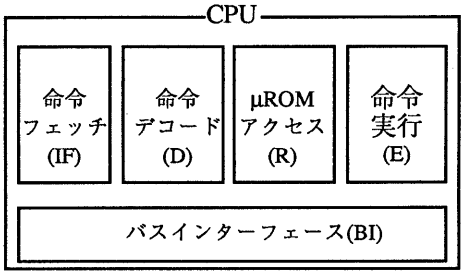

図 3　機能設計用モデルのモジュール構成

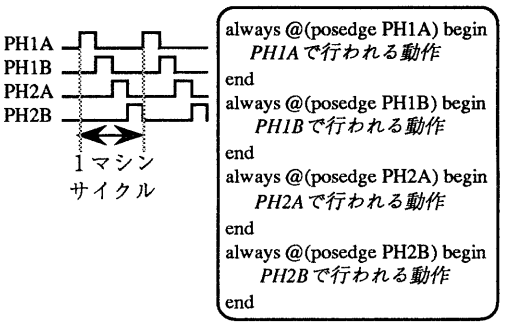

図 4　機能設計用モデルのクロックと記述フォーマット

路の機能だけでなく回路の動作速度やレイアウトを考慮した詳細な記述が必要となる。

そこで、図 2 に示すように、まず機能設計と機能検証を目的とした機能モデル (=機能設計用モデル) を作成し、これを論理合成可能なモデル (=インプリメンテーション用モデル) に書き換えるという 2 段階に分けた機能モデルの開発を行なった。

機能設計用モデルは、ハードウェア記述言語である Verilog-HDL を使用したレジスタトランスファレベルの記述であり、このモデルそのものを機能仕様書として使用する。記述上の主な特徴を以下に挙げる。

- パイプラインステージを基本にモジュール分割した (図 3)
- 実際のクロックは 2 相だが、記述を容易にするために、1 マシンサイクルを PH1A, PH1B, PH2A, PH2B の 4 相に分割して記述した (図 4)
- 上記で定めた各クロック毎に動作をまとめて記述した (図 4)

インプリメンテーション用モデルは、機能設計用モデルを論理合成用可能な記述に書き換えたものである。

機能設計用モデルとインプリメンテーション

モデルの記述量とシミュレーション速度を表 2 に示す。

4　高速シミュレータの開発

ハードウェア記述言語は、

- ハードウェアの動作・構造に即した動作モデルの作成が容易
- ゲートレベルの論理シミュレータに比べてシミュレーション速度が速い
- 論理回路を自動生成する論理合成ツールが使用できる

などからハードウェア設計の効率化に有効である。

しかし、ソフトウェアの開発やその速度性能評価に用いるには、シミュレーション速度が十分速いとは言い難い。たとえば、M16 の実際のチップで 1 秒間の実行時間を Verilog-HDL の機能設計用モデルを用いてシミュレーションする

表2　各機能モデルの記述量と速度

	記述量	シミュレーション速度†1
機能設計用モデル	5,200 行 (Verilog-HDL)	20 clock/秒
インプリメント用モデル	16,000 行 (Verilog-HDL)	7 clock/秒
パイプラインシミュレータ	10,000 行†2 (C 言語)	2000 clock/秒

†1 SPARC 370 での測定値
†2 CPU 部の記述は 5600 行

と、6 日間必要となる。これでは、実際に使用されるような応用プログラムを用いての速度性能評価を繰り返して行なうことは難しい。

そこで、クロックサイクル単位でハードウェアの動作をシミュレートするより高速なシミュレータを開発する必要があった。この高速シミュレータは、通常のソフトウェアシミュレータと区別するために、パイプラインを含めたハードウェア動作のシミュレータの意味で『パイプラインシミュレータ』と呼ぶことにした。

パイプラインシミュレータのシミュレーション速度は、表2のように機能設計用モデルの 100 倍となり、実際のチップの 1 秒間を 1 時間半程度でシミュレーション出来るようになった。

このパイプラインシミュレータの実現について述べる。

4.1　パイプラインシミュレータ

パイプラインシミュレータは、M16 上での応用プログラムの速度性能評価およびチューニングを目的としたシミュレータである。

パイプラインシミュレータは図5に示すように、次の 3 つの部分から構成されている。

- CPU 部
  CPU のハードウェア動作をシミュレートする
- モニタ部
  ユーザからのコマンドを解釈し、CPU 部のシミュレーションを制御する

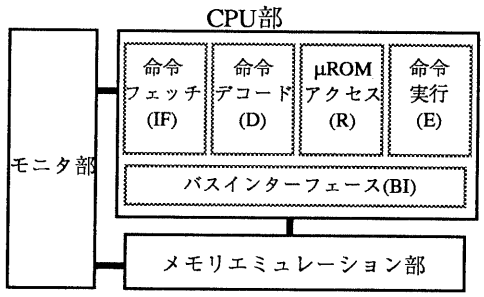

図5　パイプラインシミュレータの構成

- メモリエミュレーション部
  シミュレートしている CPU が使用するメモリを提供する
  ユーザのコマンドによりシミュレートプログラムのロードや内容の表示/変更が可能

パイプラインシミュレータのそれぞれの部分の実現について順に述べる。

4.1.1　CPU 部

CPU 部は、ハードウェア記述言語 Verilog-HDL による機能設計用モデルの記述をその動作に相当する C 言語のプログラムにする方法で実現した。

一般のプログラミング言語である C 言語とハードウェア記述言語の Verilog-HDL では、次のような違いがある。

(1) 並列動作, イベント駆動
  Verilog-HDL では並列動作やイベント駆動の動作を記述できるが、C では逐次実行のみ

(2) 不定値
  Verilog-HDL では不定値 X やハイインピーダンス Z を扱い、X の伝搬などのシミュレーションが可能

(3) ビット長
  C の整数型では 32 ビット (処理系依存) が最大のビット長となるが、Verilog-HDL ではさらに長いビット長の整数が使用できる

これらの違いを考慮したうえで、Verilog-HDL での記述を C 言語プログラムに人手によって書き換える方法をとった。人手による最適な書き

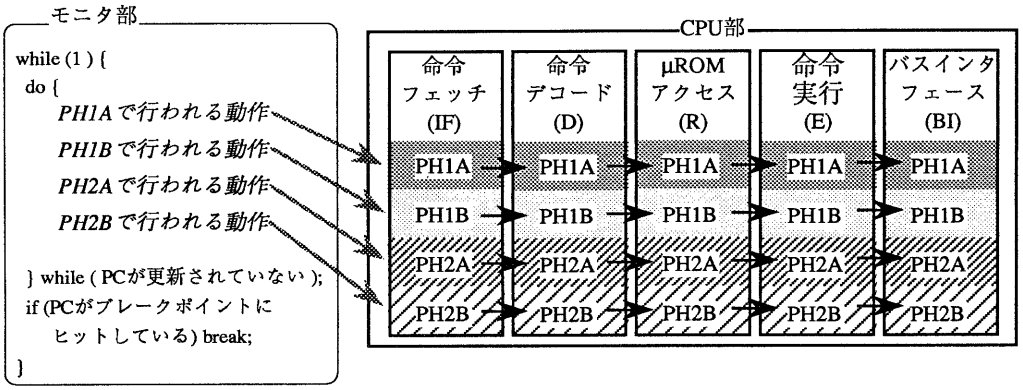

図 6　イベントの発生とスケジューリング

換えで高速なシミュレータを開発することが最大の理由であるが、機能設計用モデルが 5200 行と比較的規模が小さく、また 4 相のクロックごとの動作で記述されておりこのクロック以外のイベントがないという機能設計用モデルの記述上の性質から、比較的容易に人手による書き換えを行なえると判断したこともその理由である。

先に述べた Verilog-HDL と C 言語で違う部分をどのように書き換えたかを順に述べる。

(1) 並列動作, イベント駆動

機能設計用モデルでは、記述を容易にするために 1 マシンサイクルを PH1A, PH1B, PH2A, PH2B の 4 相に分割して記述されている。この各相毎にモジュール分割された命令フェッチ (IF)、命令デコード (D)、μROM アクセス (R)、命令実行 (E)、バスインタフェース (BI) の動作が並列に行なわれる。

この動作は図 6 に示すように、クロック PH1A, PH1B, PH2A, PH2B が順次発生した状態をシミュレートするモニタ部から、各相毎に IF → D → R → E → BI といった順序で逐次実行する構成とした。

(2) 不定値

ハードウェア記述言語では、0 か 1 のいずれかわからない不定値 X を扱うことができるが、C 言語では不定値 X を簡単には扱うことはできない (図 7)。しかし、不定値 X を扱いそれを伝搬させる意味は、ハードウェア論理の検証にあるといえる。論理検証は、機能設計用モデルなどハードウェア記述言語によるモデルで行なうので、パイプラインシミュレータは不定値を扱わないとした。ただし、don't care の X は、そのビットをマスクすることでシミュレートする。

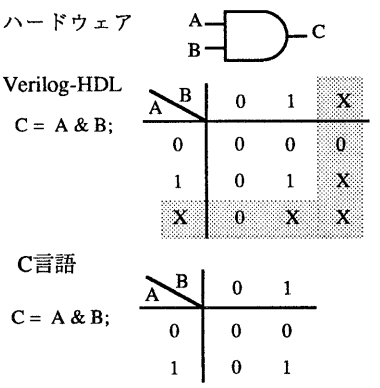

図 7　不定値の扱い

また、ハイインピーダンス Z を出力する時の動作は、出力する信号線 (変数) の値を変更しないという動作、すなわち何もしない文とする。

(3) ビット長

機能設計用モデルでは、μROM の出力信号、ALU の信号などが 32 ビットを越えるビット長を持っている。これらの信号線は、整数型変数の配列を持つ構造体とし、その構造体を操作するマクロや関数を用意した。

4.1.2 モニタ部

モニタ部は、ユーザからのコマンドを解釈し、CPU 部のシミュレーションを制御する。ここでは、CPU 部の制御の中で特徴のあるブレーク機能と任意アドレスからの実行開始について述べる。

(1) ブレーク機能

パイプライン処理を行なうマイクロプロセッサにおいて、実行しているアドレスを命令フェッチのアドレスなどチップ外部の信号から知ることは難しい。しかしパイプラインシミュレータでは、実行ステージにある PC(プログラムカウンタ) の値を知ることができる。ブレーク機能の実現にあたっては、この PC の更新をとらえてブレークするようにすれば良い。ただし、パイプライン処理の状況によっては、さらに検討が必要となる。

たとえば、図9のようなジャンプ命令の実行について考えてみる。PC は、実行ステージでジャンプ命令 (JMP) の処理が終了すると、次に実行すべき命令 (MOV) のアドレスに更新される。この更新時点は、次に命令 MOV に対する実行が開始される時点ではない。これは、ジャンプ命令の実行でパイプライン中の命令列が削除されたためで、命令 MOV に対する実行ステージの処理は待たされることになる。このような場合、ブレークをどのタイミングでかけるべきかが問題となる。我々は、待たされることによってジャンプ命令の実行時間が長くなるのではなく、ジャンプした先の命令の実行に時間がかかるとした。すなわち、次に実行する命令のアドレスに PC が更新されると即停止する方法をとった。

実行ステージの PC が更新されたとき、そのアドレスにブレークポイントが設定されているかどうかをチェックし、もし設定されている場合は、ブレークポイントにヒットした判断する。設定されていないときにはハードウェア動作のシミュレーションを継続する。

これにより、該当命令の実行前に停止するブレーク機能を実現した。

(2) 任意アドレスからの実行開始

実際のチップを用いたシステムにおいては、デバッグモニタと呼ばれるソフトウェアによって、

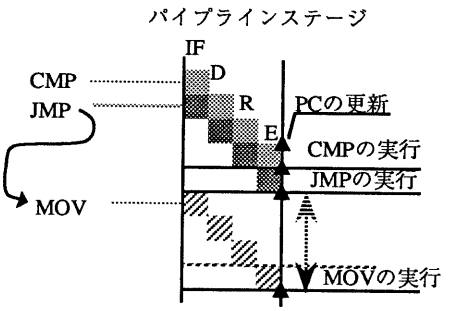

図9　ジャンプ命令の実行

レジスタの内容を変更したりユーザが指定した任意のアドレスからの実行が可能であり、この機能はユーザのプログラムのデバッグなどには有用である。

先に述べたように、実行ステージ PC の更新時点を命令の開始時点とし、該当命令の実行前のブレーク機能を実現した。しかし、ブレークポイントで停止した場合、実行ステージでの処理はまだ行われていない状態でも、命令フェッチやデコードといったパイプライン処理は次に実行される命令について行われている。このため、単に実行ステージの PC を書き換えるだけでは、実行するアドレスを変更できない。また、メモリやレジスタの内容を書き換えた場合、そのまま処理を継続できない。なぜなら、パイプラインのステージ中に存在する情報にメモリやレジスタの変更を正しく反映させることは難しいためである。

このような問題を解決するために、パイプラインシミュレータでは次のような方法をとった。

- まず、CPU 部に対してハードウェアリセットを行なった状態を仮想的にシミュレートする。
- 次に、レジスタの設定を行い、ユーザの指定した実行開始アドレスにジャンプするプログラムの実行を仮想的にシミュレートする。

この方法で、任意番地からの実行開始をパイプラインシミュレータで実現した。

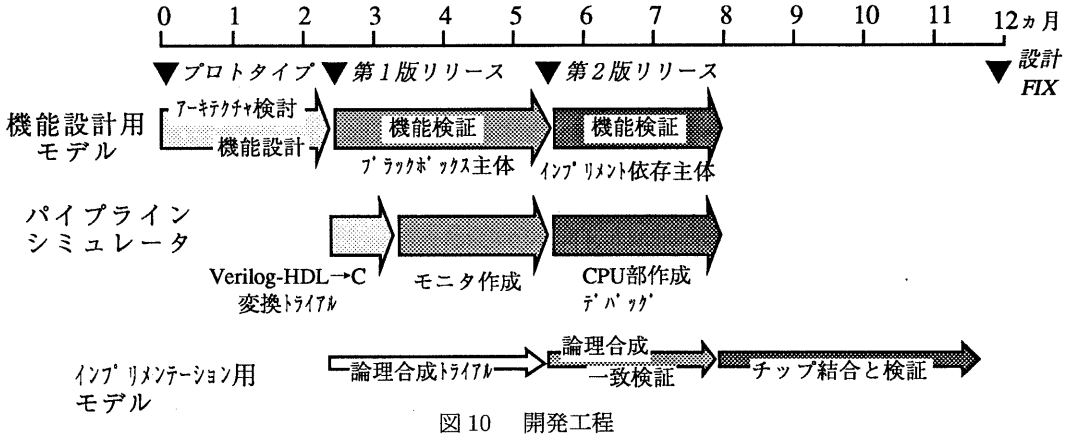

図10　開発工程

4.1.3　メモリエミュレーション部

シミュレートしているCPUが使用するメモリを、動的な割り当てを行なって用意する。アドレスによる指定領域毎にメモリのウェイト数を設定できる。このメモリは、ユーザからのコマンドによりシミュレートプログラムのロードや内容の表示/変更が可能になっている。

4.2　パイプラインシミュレータの検証

パイプラインシミュレータが、機能設計用モデルで定義された仕様どおりに作成されているかを確認するために一致検証を行なった。これには、機能設計用モデルで検証されたテストプログラムを用いてシミュレーションを実行し、その時の信号線の値を比較することで行なった。

実際には、命令実行開始毎に、

- 実行ステージでの命令処理が開始されたシミュレート時刻
- PC値
- マイクロアドレス

を比較した。

この方法では、パイプラインの全ての状態の一致を保証することできないが、命令の実行に要するクロック数は一致していることが保証される。

パイプラインシミュレータの目的は、各命令の実行時間を正しく測定することであり、内部の動作については、命令に対する動作という外部仕様レベルでの一致がとれれば良く、内部の状態を含め厳密に一致することを確認する必要がないので、このような簡略化した一致検証とした。

5　開発の実際

機能設計開始からの工程を図10に示す。

機能設計用モデル第1版のリリースを受けて、パイプラインシミュレータへの書き換えのトライアルを開始し、シミュレーション・イベント制御用のモニタ部を作成した。

第1版に対してブラックボックス・テストにより検証が終了した時点で、第2版のリリースが行なわれた。第1版の3カ月後である。これを基に、C言語への書き換えを行なった。この書き換えには、コーディングに約1カ月、デバッグに約1.5カ月を要した。C言語のプログラムの規模は、μROMを除くCPUのシミュレーション部は5600行、イベント制御やユーザインタフェースのモニタ部などを含めると10000行である。

6　評価

8クィーン問題の解を求めるプログラムの実行のシミュレーションに要した時間を表3に示す。なお、このプログラムの実行時間は10MHz(外部入力20MHz)のM16上で、256msである。

パイプラインシミュレータでは、ハードウェア記述言語による機能設計用モデルの100倍以

表3　シミュレーション速度

	時間 (時:分)	シミュレーション速度†1
機能設計用モデル	30:58	23 clock/秒
パイプラインシミュレータ	00:17	2500 clock/秒

†1 SPARC 370での測定値

上のシミュレーション速度が得られ、実際のチップで実行時間1秒のシミュレーションが約1時間半で可能になった。

7　書き換えの自動化の検討

今回パイプラインシミュレータの開発では、ハードウェア記述言語Verilog-HDLによる機能設計用モデルを人手でC言語に書き換える方法をとった。しかし、人手による方法では、どうしても誤りが入ってしまう。実際パイプラインシミュレータのCPU部の開発では、Verilog-HDLからCへの書き換えよりも書き換えた結果のデバッグに長時間要した。この書き換えを自動化し、誤りの混入をなくすことが検討課題である。

しかし、ハードウェア記述言語で表された動作を完全な形で自動変換すると、高速なシミュレータを実現することは難しくなる。これは、

- 不定値をどのように扱うのか
  (パイプラインシミュレータでは取り扱わないことにした)
- どのようなイベントをシミュレートするか
  (4相のクロックのイベントのみ発生。非同期の信号なし)

といった点で、今回のパイプラインシミュレータの開発で行なったような柔軟な単純化ができないためである。

高速なシミュレータを用意することを機能設計の段階から想定して、例えばハードウェア記述言語のサブセットで記述を行なうなどの必要性があるのではないだろうか。

8　まとめ

1チップマイクロコントローラ「M16」と、そのハードウェア動作のクロック単位で正しくシミュレートする高速なシミュレータ「パイプラインシミュレータ」を開発した。

CPU部については、ハードウェア記述言語による機能設計用モデルを基にC言語に人手で書き換える方法で、高速なシミュレータを開発できた。このシミュレータは、機能設計用モデルの100倍以上のシミュレーション速度が得られた。

参考文献

[1] 岩田、清水、土居、中尾、水垣、三輪、「1チップマイクロコンピュータM16の機能設計」、信学技報 ICD93-88(1993-09)、pp.37-44