マイクロコンピュータ　60－2
とワークステーション
（１９９０．５．８）

ＡＩチップ（ＩＰ１７０４）の開発環境

岡村　光善*　相川　健*　的場　司*　皆川　健二*
木南　英志**　今井　徹*　斎藤　光男*
*（株）東芝　総合研究所
**東芝ソフトウェアエンジニアリング（株）

新しいアーキテクチャを持つプロセッサを開発する場合には、アーキテクチャ設計段階での充分なデバッグ，検証と各開発行程におけるテストプログラムの共有化，マンマシンインタフェースの統一などが重要であり、このことがＶＬＳＩプロセッサの開発効率の向上にもつながる。

我々は、ＶＬＳＩプロセッサ（ＩＰ１７０４）の開発にあたり、命令仕様定義，アーキテクチャ設計から実チップの評価に渡る開発行程をサポートするツールの開発を行なった。設計の各段階で計算機でサポートされたツールを用いて検証を行なうことにより、確実な設計が可能となり、かつ設計のターンアラウンドタイムが短縮される。

ENVIRONMENT FOR DEVELOPING AN AI PROCESSOR

Mitsuyoshi Okamura*　Takeshi Aikawa*　Tsukasa Matoba*　Kenji Minagawa*
Hideshi Kiminami**　Toru Imai *　Mitsuo Saito*

* Toshiba Research and Developoment Center,Toshiba Corporation
**Toshiba Software Engineering

1 Komukai Toshiba-cho,Saiwai-ku,Kawasaki-shi,Kanagawa 210,Japan

We have newly developped VLSI procesor design environment on a engineering workstation. The environment includes instruction definition tools ,test program development environment, a architecture leve simulator/debugger , a logic level simulator/debugger , and a hard ware debugger.

The man-machine interface of each debugger are uniformed and all simurator can execute same test programs which was developped by the test program development environment.

We have developped a VLSI processor (AI chip) with this environment in a short term.

1　まえがき

ＶＬＳＩプロセッサを開発する際には、アーキテクチャ設計，論理設計，レイアウト設計，タイミング検証，実チップ評価の段階がある。ＡＳＩＣの設計に比較するとアーキテクチャ設計に重点がおかれ、この段階において充分に仕様，性能の評価を行なう必要がある。

近年ＡＳＩＣをターゲットとするＶＬＳＩ・ＣＡＤについては多くの優れたシステムが開発されているが、ＶＬＳＩプロセッサの開発を行なう場合にはアーキテクチャ設計に対して充分な機能を持っているとはいえない。

ＶＬＳＩプロセサ開発効率の向上の為には、

(1) 上記の各段階、特にアーキテクチャ設計の段階で充分なデバッグ、検証手段を有すること。
(2) 検証に使用されるテストプログラムが各段階で共有できること。
(3) できる限りマンマシンインタフェースが統一化されていること。
(4) テストベクタはテストプログラムを実行することで自動的に発生すること。

などが重要である。

我々は、ＡＩチップ（ＩＰ１７０４）の開発にあたり、上記の考え方に基ずきアーキテクチャ設計から実チップ評価に渡る各開発行程をサポートするツールの開発を行った。その結果、このツールを使用することにより、ＡＩチップを効率的に開発することが出来た。

2　ＡＩチップの概要

ここで、今回開発したＡＩチップについて簡単に説明する。我々はすでに、新しいアーキテクチャに基ずくＡＩプロセッサ（ＩＰ７０４）を開発した[1][2]。本チップはＩＰ７０４をチップ化する目的で開発されたＬＩＳＰ，ＰＲＯＬＯＧ等で書かれたＡＩ言語及び、汎用言語で書かれたプログラムを高速に実行するＶＬＳＩプロセッサである。

命令セットアーキテクチャは、ＲＩＳＣ的な汎用言語用の命令セットに、ＡＩ言語用の命令セットが追加されたものである。命令フォーマットはどちらの命令セットともＲＩＳＣ的な単純化されたものである。

追加されたＡＩ言語用命令セットは、

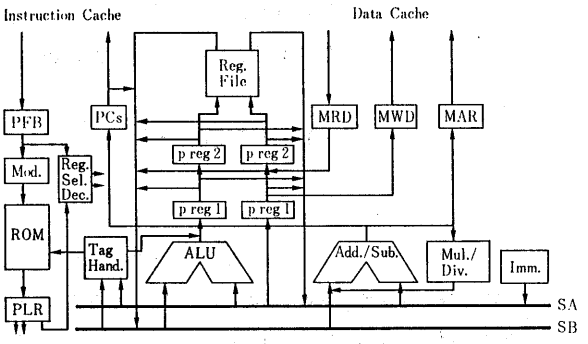
(1) レジスタなかのデータのタグ値によるコンディショナルブランチ命令
(2) タグ部を除いたデータ部のみ演算対象とする演算命令
(3) ジェネリックな演算命令
(4) ＷＡＭ (Warren Abstract Machine)ベースの命令

などである。

ハードウェアアーキテクチャはＲＩＳＣのアーキテクチャをベースに、柔軟な制御を行うために水平型マイクロプログラム技術を取入れ、さらに、ＡＩ言語で書かれたプログラムの高速実行をサポートするハードウェアを付加したものである。

付加されたハードウェアは、

(1) ＰＲＯＬＯＧ，ＬＩＳＰが扱うタグ付きデータを、高速処理するためのタグ取扱ハードウェア



PFB : PreFetch Buffer
Mod. : Instruction Modifier
ROM : Read Only Memory
PLR : Pipe Line Register
PC : Program Counter
MAR : Memory Address Rg
ALU : Arithmetic Logical Unit
MWD : Memory Write Register
MRD : Memory Read Register
SA : Source bus A
SB : Source bus B
Imm. : Immediate generator

図1　ＩＰ１７０４の構成

(2) チップ内部のデータ転送経路を二重化し、低レベルの並列処理を行うための２ポートデータパス構造

(3) 実行時に、機械語命令の処理をハードウェアで切り替える命令すりかえハードウェア

(4) ＰＲＯＬＯＧ，ＬＩＳＰにおいて、高頻度で発生するスタック領域アクセスを高速化するスタック領域チエックハードウェア

などである。

図１に、ＩＰ１７０４の構成を示す。

３　サポートツール

ＡＩチップの開発環境は、命令仕様定義の段階から、実チップ評価の段階までをサポートするツール群であるが、大きく分けて次の５つの部分から成る。

(1) 命令仕様定義、マイクロプログラム作成ツール

(2) テストプログラム作成ツール

(3) アーキテクチャ検証のためのツール

(4) 論理設計，レイアウト設計，タイミング解析のためのサポートツール

(5) 実チップ，周辺ハードウェアを実装した評価用ハードウェア

図２に、ＡＩチップ開発環境の各サポートツールの全体構成を示す。

４　アーキテクチャ検証

新しいアーキテクチャを持つプロセッサを新規開発する場合には、そのプロセッサを構成する回路ブロックの内部構造を詳細に定める前に、そのプロセッサのアーキテクチャの正統性を厳密に確認しつつ、かつ、その仕様に沿ったハードウェアがリーズナブルにインプレメントできるか否かの

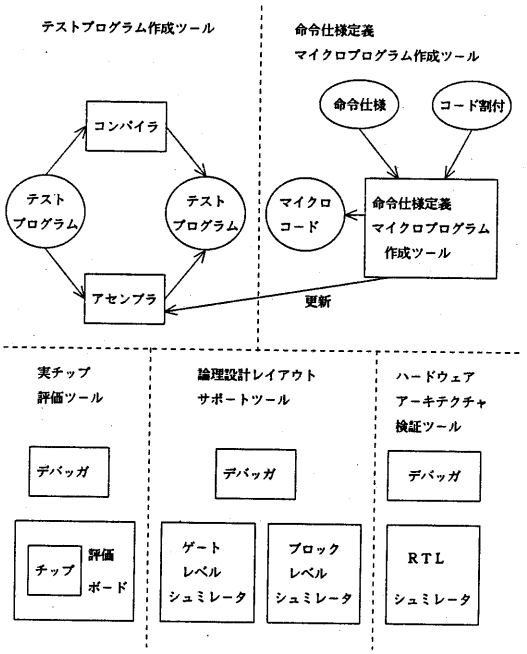


図２　ＡＩチップ開発環境の構成

確認を行なっていく事が重要である。

我々は、プロセッサ仕様レベルの検証手段として、Ｃ言語でプロセッサを構成するブロック（十数個のオーダ）及びキャッシュ，メインメモリ等の周辺回路の外部仕様を記述し結合する事により、シミュレーションモデルを作成した。また、Ｃ言語で書かれたシミュレーションモデルの実行制御、及びデバッグを行なうためのデバッガも同時に開発を行なった。このシミュレーションモデル，デバッガをＥＷＳ上で走らせ、Ｃ言語，ＬＩＳＰ，ＰＲＯＬＯＧで書かれた代表的なベンチマークプログラムをシミュレーションモデル上で走らせる所までアーキテクチャレベルのシミュレーションを行ないアーキテクチャの正統性、及び性能評価を行なった。

４．１　シミュレーションモデル

通常、このレベルのシミュレーションは、ハー

ドウェア記述言語、或いはハードウェアのネット記述そのものをシミュレーションモデルとして行なわれてきたが、我々は敢えて、汎用言語であるCを用いて記述を行った。C言語でプロセサのシミュレーションモデルを記述するメリットデメリットは、以下のとおりである。

メリット

(1) 多くの人間が言語仕様を理解している。
(2) アルゴリズムレベルからゲートレベルにいたるまでいろいろなレベルで記述が可能である。
(3) ｄｂｘを初めとしてデバッグ手段，開発ツールが整っている。
(4) シミュレーション速度が速い。
(5) デバッガ等のシミュレーションモデルを制御するプログラムとのインタフェースが容易である。
(6) 従って、仕様記述　－＞　仕様検証のターンアラウンドが速い。

デメリット

(1) 仕様記述から回路を生成する事ができない。
(2) 従って、仕様　－＞　ハードウェア記述は人手で行なわなくてはならない。
(3) 記述の順序をまちがえると正しく動作しない。

我々は、記述順序を意識しなくてはいけないというデメリットを差し引いても、C言語の方が速く、かつ他のメリットを考え併せて、アーキテクチャレベルの検証をC言語で書かれたモデルで行なう事が、現在の所、トータルの開発スピードを向上させると考えた。

また、仕様記述と実際のハードウェアとの同一性を検証するために、このシミュレーションモデルの検証で用いるテストプログラムと、7章で述べる論理設計フェーズの検証で用いるテストプログラムと共有化するようにした。さらに、4．2章で述べるデバッガのインタフェースを、7章で述べるデバッガとできるだけ合わせる事により同一性の確認を容易にした。

4．2　デバッガ（ｓｄｂｇ）

デバッガｓｄｂｇは、4．1章で述べたシミュレーションモデルの実行制御，デバッグを行なうためのツールである。このデバッガのマンマシンインタフェースは、8章で述べるデバッガｈｄｂｇとマンマシンインタフェースを完全に同一にした。また、7章で述べるデバッガともできる限りマンマシンインタフェースを同一になるようにした。

ｓｄｂｇは以下の機能を持つ。

(1) ハードウェアリソース・アクセス機能
　・メインメモリ，キャッシュ，レジスタ等のハードウェアをアクセスする機能
(2) プログラム実行制御機能
　・テストプログラム作成ツール上で作成されたプログラムのロード／セーブ
　・実行，マイクロ／機械語レベルのステップ実行
　・マイクロ／機械語レベルのブレーク
(3) Ｉ／Ｏ機能の肩代わり
　・シミュレータモデル上で走るプログラムのＩ／ＯをＥＷＳで肩代わりする機能

この方法で、約１１３０００トランジスタからなるＡＩチップを、レジスタトランスファレベルで記述されたシミュレーションモデルに対して、ＡＳ３２６０上で１０００～３０００クロック／秒で動作している。

5　命令仕様定義，マイクロプログラム作成ツール

命令仕様はプロセサアーキテクチャの大きな部分を占めており，3章で述べたツールでアーキテクチャ検証を行なう過程のあるフェーズに於いては、日々、命令仕様の追加変更の必要がある。また命令仕様をインプレメントしているマイクロプログラムは、アーキテクチャ検証の期間を通して

バグフィックスのため変更が行なわれる。命令仕様が追加変更になると、それに伴い、仕様書の変更，アセンブラの変更などが必要となり、変更のコンシステンシを採る事が重要であり、人手でこれを管理する事は容易ではない。

そこで我々は、命令のニーモニック，コード，フォーマット，マイクロコード等を、計算機上のファイルに定義し、マイクロコード，アセンブラ，逆アセンブラは、この命令定義ファイルから自動変更されるようなツールを開発した。このツールを使用する事により、命令定義ファイルを変更してから数分で全ての更新を終える事が可能で、命令定義の追加変更を開発のかなり後のフェーズまで危険無しに行なう事が可能であった。

6　テストプログラム作成ツール

ＡＳ３０００上に、クロスのＣコンパイラ，アセンブラ，リンカを開発した。ｄｈｒｙｓｔｏｎｅ，スタンフォードベンチマーク等は、Ｃコンパイラでコンパイルを行い性能評価に用いた。またアーキテクチャ検証のためのテストプログラムは、おもにアセンブラで記述を行なった。このツールで作成されたテストプログラムは、アーキテクチャ検証，論理検証，実チップ検証の各フェーズで共通に用いた。また、システムコールが発生したときにデバッガのＩ／Ｏ肩代わり機能を呼び出す簡易モニタを開発し、ベンチマークプログラム、或いはテストプログラム中のＩ／Ｏを行えるようにした。この簡易モニタも上記３つのフェーズで共通に使用でき、３つのモデルの論理的な同一性を確認する事を容易にした。

7　論理設計・レイアウトサポートツール

論理設計の段階では、論理シュミレータを使用して検証されたアーキテクチャを、いかに忠実にインプリメントするかが重要である。これは、今の所、論理シミュレータ上のモデルを自動的に論理設計のためのモデルとしてＣＡＤシステム上に構築できないため、唯一人手が介在することによるものである。したがって、この段階の検証ではアーキテクチャの検証で使用したテストプログラムを実行させ、その結果が同じ動作を行うことを十分確認する必要がある。

論理設計、論理シミュレーション，ゲートレベルシミュレーション，タイミング解析及び、レイアウトのサポートツールとしてシリコンコンパイラ社のＣＡＤシステムであるＧＥＮＥＳＩＬ（注1）を使用した。

デバッガは、ＧＥＮＥＳＩＬで用意されている、ｇｅｎｉｅと呼ばれるＬＩＳＰライクの言語を用いて、論理シミュレータのデバッガとほぼ同一のマンマシンインタフェースを持つものを開発した。

ＧＥＮＥＳＩＬ上には、図３に示すように、ＩＰ１７０４と命令キャッシュ，データキャッシュを含む周辺回路の論理モデルをチップセットとして構築しシミュレーションを行った。シミュレーションモデルはｇｅｎｅｓｉｌにより論理設計データから自動的に作成される。シミュレーションは、このモデルを用い、ブロックレベル，ゲートレベルの２段階で行なった。

テストプログラム作成ツールで作成されたテストプログラムは、この環境でも使用でき、テストプログラム実行後には自動的にテストベクターが発生される。これはチップ製造時のテストベクターにもなる。

レイアウトは、設計者が対話的に各ブロックの配置を決定するだけで自動生成される。

(注1)　GENESIL は、Silicon Compiler Systems Copr.の商標。

8　実チップテスト環境

実チップテストの段階では、論理シュミレーションで検証したことが、実チップで正常に動作するかのデバッグ，検証が必要となる。

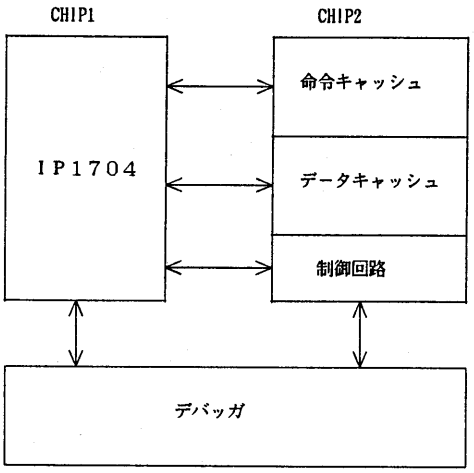


図3　GENESIL上のモデル

8．1　チップ内のデバッグ手段

実チップ評価では、全てのテストプログラムが全て正常にパスすれば問題ないが、必ずしも一度でパスするとは限らないのが普通である。そこで、実チップテストのために、チップ内に次ぎのようなデバッグ手段を組み込んでいる。

内部リソースへの直接アクセス手段

アクセス可能なリソースとしては、

(1) プログラムカウンタ
(2) マイクロプログラムカウンタ
(3) プログラムステータスワード
(4) レジスタファイル
(5) マイクロコードROM

などがある。

チップの実行制御手段

(1) ステップ制御
(2) ブレーク制御
(3) マイクロアドレスマッチブレーク制御
(4) ディスパッチブレーク制御
(5) 強制ストップ制御

などの制御が可能である。

これらは、スキャンパスによりシリアルにアクセスされる。

8．2　デバッガ

実チップテストのためのデバッガは、アーキテクチャ検証の段階で使用した論理シュミレーション用のデバッガを利用した。このことにより、マンマシン・インタフェースは全く同じである。但し、下位レベルのドライバールーチンは当然異なる。さらに、アーキテクチャ検証，論理設計の段階でのシュミレーションで用いたテストプログラムもそのまま使用できる。

8．3　評価ボード

評価ボードの基本構成を、図4に示す。評価ボードは、IP1704，命令キャッシュ，データキャッシュ，タグメモリ，スキャンパス制御，割込み制御，及びトラップ制御等から構成されており、VMEインタフェースを介してデバッガから制御される。また、IP1704の入出力信号もデバッガから観測する事ができるようになっており、異状な動作が発生した際、その原因はチップなのか、評価ボードなのかの確認ができる構成にしてある。

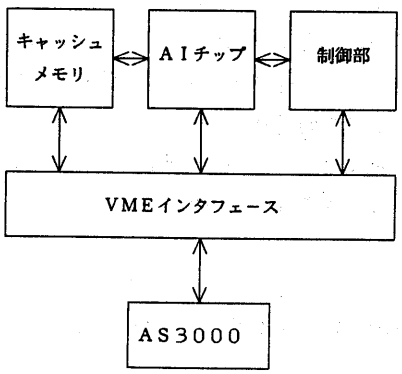


図4　評価ボードの構成

実チップ評価には機能動作確認の他に、動作速度の評価も重要である。そこで、評価ボードはチップの基本的な機能評価に関しては全てハードウェアで処理し、リアルタイムでの動作を可能にしている。速度評価に関係のない、例えば、キャッシュミス時のキャッシュ入替えなどのコントロールロジックはＡＳ３０００上のソフトウェアでシミュレートしている。

９　おわりに

命令仕様定義，アーキテクチャ設計から実チップ評価までをサポートするツールを開発した。このように設計の各段階で計算機サポートされたツールを用い検証を行なうことにより、確実な設計が可能となり、かつ設計のターンアラウンドタイムが短縮される。また実際にこれらのツールを使用してみて、各設計段階において使用するデバッガのマンマシンインタフェースを統一する事は、使い勝手を非常に向上させる事がわかった。

参考文献

[1] 皆川，斎藤，相川，的場，岡村，石井，「ＡＩチップ（ＩＰ１７０４）のアーキテクチャ」，『情報処理学会第38回全国大会論文集』，pp.1538-1539,1989 年3 月

[2] 相川，萬代，木下，「ＡＩワークステーション用ＶＬＳＩ」，『東芝レビュー』，44,10，pp.779-782,1989