# 1チップCPUプロセッサの設計検証(2)
# 機能モデルと論理モデルの比較による一致検証

5N−8

東 辰輔†, 中野 孝†, 平岡 精一†, 市川 周一†, 岩田 俊一†, 仙名 一朗‡
†三菱電機(株)　‡三菱電機エンジニアリング(株)

## 1 はじめに

大規模化するマイクロプロセッサの開発において、設計検証はますます困難になっている。トップダウン設計が叫ばれてはいるものの、論理合成技術が未熟である現在においては、人手による論理設計に頼らざるを得ないというのが実情である。その際、仕様通りに設計されているということをいかにして検証するかということが問題となる。

我々はこのたび、ビジネスコンピュータ用1チップCPUプロセッサをフルカスタム設計するにあたり、機能モデルと論理モデルという2つのモデルを開発した。その上で、両モデルの動作をクロック単位で比較する「一致検証」を導入し、論理検証では発見が困難なバグの検出および解析を行なった。

本稿では、設計検証の手法、および一致検証の効果について報告する。また、表1にプロセッサの諸元を示す。

## 2 機能モデルと論理モデル

今回のマイクロプロセッサ開発では、同一の機能仕様に基づいて、並行して独立に機能モデルと論理モデルを開発した。機能モデルはハードウェア記述言語Verilog-HDLを用いて機能レベルで記述され、論理モデルはゲート/トランジスタレベルで回路図として記述されたものである。

機能モデルは、その記述の柔軟性とシミュレーションの速さから論理設計に先立って、機能仕様の検証、マイクロコードの検証、テストプログラムの開発等に活用された。また論理モデルは、論理検証や回路設計、レイアウトに使用された。

両モデルには、テストプログラム実行用に主記憶や割込みコントローラを模擬する外部回路が付加される。

表1 1チップCPUプロセッサの諸元

| | |
|---|---|
| パイプライン | 6段 |
| キャッシュ | 命令/データ各8Kバイト |
| 内蔵 | FPU、MMU、TLB |
| 製造プロセス | 0.8μm CMOS |
| トランジスタ数 | 約170万 |

## 3 一致検証

### 3.1 検証手法

設計検証は、次の2点を検証することを目的とする。

1. 仕様に誤りがない
2. 仕様通りに設計されている

1は、上で述べたように機能モデルを用いて検証される。2は論理モデルの検証に相当するが、我々は、テストプログラムを用いて命令レベルでの確認を行なう論理検証に加えて、1クロックごとの動作確認を短時間で行なえる「一致検証」を新たに導入した。

図1に設計・検証のフローを示す。機能モデルと論理モデルのそれぞれにテストプログラムを読み込み、シミュレーションを行なう。その際に、チップ入出力信号等の主要な信号の値を1クロックごとに出力し、それぞれ機能ダンプファイル、論理ダンプファイルを生成する。比較ツールを用いて機能ダンプファ

---

Design Verification of One-Chip CPU Processor (2)
Comparative Verification between Functional Model and Logical Model
Shinsuke AZUMA, Takashi NAKANO,
Seiichi HIRAOKA, Shuichi ICHIKAWA,
Shunichi IWATA, Ichiro SENNA
Mitsubishi Electric Corp.
Mitsubishi Electric Engineering Co., Ltd.

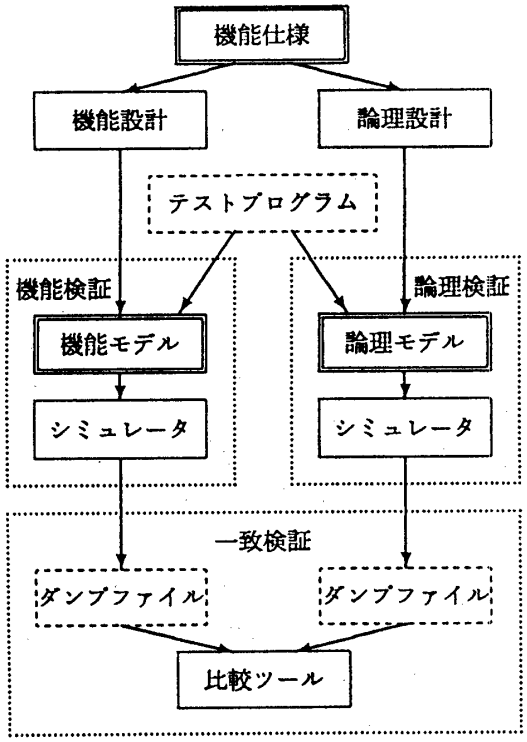


図 1 設計・検証フロー

イルと論理ダンプファイルを比較し、不一致箇所を抽出する。機能モデルでは、論理モデルに先行してテストプログラムの実行および動作確認がなされているので、機能ダンプファイルを期待値として、不一致があればその原因を追求するという手法である。

### 3.2 一致基準と比較ツール

一致検証を実際に導入するにあたっては、一致基準をどう定めるかが重要である。そのための要素としては、ダンプファイルに出力する比較対象信号の選択と、比較の際の条件付けが挙げられる。

比較対象信号としては、チップ入出力信号のほかに、汎用レジスタ等のソフトウェアから見えるレジスタ、プログラムカウンタ、パイプライン中の実行ステージの状態を示すマイクロアドレスを選択した。

また、比較の際の条件付けを実現するために、比較ツールにマスク機能をもたせた。仕様上は不定 (don't care) となっている信号に関しても、論理モデルでは 0/1 いずれかの値が出力される。一方、機能モデルでは不定のまま扱う。このような場合に無意味な不一致を発生させないよう、マスク機能を活用して対処した。

## 4 効果

一致検証を行なった結果得られた効果を挙げる。

- 検証効率の向上 — 設計検証で検出した全バグのうち、約 12% を一致検証によって検出することができた。しかも一致検証に要する時間は、論理シミュレーションに要する時間に比べ無視できるほど短い。開発期間中の限られた時間で、効率の良い検証が行なえる。
- きめ細かな検証 — 一致検証で検出したバグのほとんどは、キャッシュの更新制御やパイプラインのインタロック制御のように、誤動作にはならないが性能劣化の原因となるものであった。このように一致検証では、命令レベルのテストプログラムでは検出が不可能あるいは困難なバグを検出することができる。
- 不具合解析の容易化 — 論理検証では不具合の発生時点は、エラーの検知より以前であることが多い。一致検証では不具合の発生時点を容易に特定できる。
- 検証の自動化 — 仕様変更やバグ改修等の理由で設計変更があった場合の再検証の作業を自動化できる。

## 5 まとめ

ビジネスコンピュータ用 1 チップ CPU プロセッサ開発において我々が行なった一致検証について、その手法と効果を述べた。

フルカスタムチップ開発において、トップダウン設計の各プロセスが自動化されることが理想であるが、人手による論理設計が一般的である現状では、レベルの異なる 2 つのモデルの動作を比較するという我々が行なった検証手法は有効であることが分かった。