2G－2

１チップ・マイクロコンピュータM16の開発（２）パイプラインの状態を考慮した機能検証

土居俊雄　岩田俊一　桝井規雄　前田弘美　水垣重生　清水徹

三菱電機（株）

１．はじめに

LSIの大規模化に伴い論理合成や自動配置配線などの各種の設計支援ツールが整備され、インプリメントする機能に適する種々の設計手法を適切に選択することにより機能に対して相対的に設計期間を短縮することが可能になりつつある。また仕様や機能の設計においてもハードウエア記述言語で「シミュレーション可能な」仕様書を記述することにより、論理設計を待つことなく実際的な検証ができるようになった。このように複雑な機能を持つLSIの開発期間が短縮され、開発初期から検証が可能な設計環境においては、初期の機能モデル開発に同期して検証品質を向上させることが要求される。

しかし、パイプライン処理方式のプロセッサでは命令のシーケンスやメモリアクセスの競合状態に依存する複雑な内部状態を持つため、開発の初期段階では検証が不十分になりがちであった。

今回１チップのマイクロコンピュータ「M16」を開発するにあたり、開発の比較的早い段階で機能モデル上でパイプラインの状態を考慮した機能検証をおこない検証品質の向上を試みた。

２．検証手法

機能モデルが動作可能なレベルに達した時点でまず、ハードウエアのインプリメントをブラックボックスとして扱ったテストプログラムを用いて検証した。これは主に命令や割り込みが仕様のとおり処理されることを検証するものである。

これらの検証の完了後、パイプラインの各ステージの状態に着目し、インプリメントに依存するテストプログラムを作成し検証に用いた。

開発の初期段階に作成したテストプログラムの分類とサイクル数を表１に示す。

表１　M16テスト項目の分類と
テストプログラムの実行サイクル数

分類	実行サイクル数
ブラックボックス	8684500
ｲﾝﾌﾟﾘﾒﾝﾄ依存	
ﾏｲｸﾛﾌﾟﾛｸﾞﾗﾑに着目	58245
ﾊﾟｲﾌﾟﾗｲﾝ状態に着目	206175
合計	8948920

２．１　マイクロプログラムのカバレッジ

「M16」はマイクロプログラムによって演算資源を制御しており、命令などの詳細な仕様はマイクロプログラムで実現されている。したがってパイプラインの「実行ステージ」の演算資源の状態はマイクロプログラムに依存することになる。

そこで既存のテストプログラムがマイクロプログラムの全アドレスを網羅しているか否かを、マイクロプログラムの実行アドレスをトレースすることによって確認した。

具体的には

１）通過したアドレスのトレース

２）分岐したアドレスのトレース

を行った。

通過していないアドレスが検出されたとき、そのアドレスが不要な場合とテスト漏れの場合が考えられ、分岐していないアドレスは、そのアドレ

M16 Single Chip Microcomputer Design (2)
Design Verification Based on Pipeline State Coverage
Toshio DOI, Shunichi IWATA, Norio MASUI,
Hiromi MAEDA, Shigeo MIZUGAKI,
Toru SHIMIZU
Mitsubishi Electric Corporation

スへの分岐が不要な場合と、テストケース漏れの場合が考えられる。検出したすべてのアドレスについてマイクロプログラムを解析して、不要なアドレスや不要な分岐は削除し、テスト漏れの場合はテストプログラムの追加を行った。

テストケース追加によりマイクロプログラムのカバレッジは100%となった。

2．2　パイプラインの状態のカバレッジ

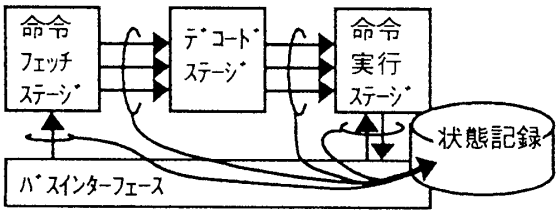
パイプライン方式のプロセッサの不具合には、命令や割り込みの処理を特定の条件、特定のシーケンス、で実行したときにのみ発生するタイプのものがある。

しかし高機能なプロセッサは多くの命令や割り込み機能を備えており、そのすべての組み合わせをブラックボックスとして網羅的に検証することはきわめて困難である。

今回の開発ではこのようなタイプの不具合を開発初期に発見するために、機能モデルの各ステージの出力信号を状態変数として分類し、取り得る状態を網羅的に再現するテストプログラム群を作成した。

これに対応して、各ステージの出力信号を状態変数としてサンプリングする機能を機能モデルに追加し、作成したテストプログラムが意図した状態を網羅しているか否かを容易に確認できる環境を構築した。図１にそのイメージ図を示す。

図１　パイプラインの状態のサンプリング

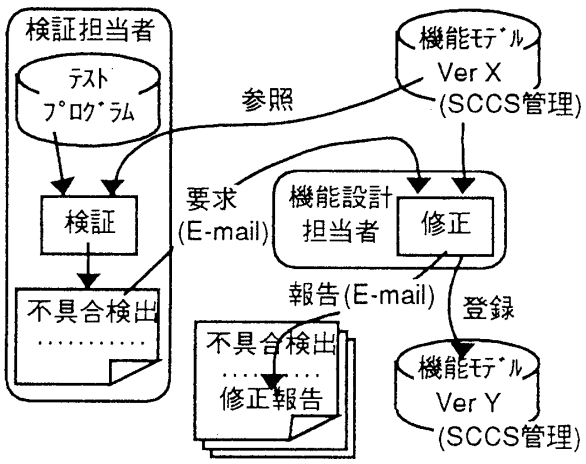


意図した状態が再現されていないとき、その状態が実際には発生しない場合と、テスト漏れである場合が考えられる。この判定は実際にはかなり難しい場合もあるが、設計者が個別に検討・判断して、テストプログラム開発にフィードバックした。

3．検証品質の維持

この開発では、機能モデルはUNIXのバージョン管理ツールであるSCCSで管理し、検証で発見された機能モデルの不具合については、検出→修正要求→修正完了、の一連のサイクルと、機能モデルのSCCS - IDとをUNIXのスクリプトやE-mail等のツールを用いて統合的に管理した。図２に不具合管理の概略を示す。

図２　M16の開発における不具合管理



機能モデルのバージョン管理と不具合のライフサイクルを統合的に管理した結果、修正漏れや、不具合解析の重複などによる時間のロスを防ぐことができた。

4．まとめ

１チップのマイクロコンピュータ「M16」を開発するにあたり、初期の機能モデル開発に同期して検証品質を向上させることに注力した。特に、開発の比較的早い段階で機能モデル上でパイプラインの状態を考慮した機能検証をおこなうことにより、特定の条件下で実行したときにのみ発生するタイプの不具合を早期に発見することができた。

また、今回のような検証において、必要な状態を発生させるテストパタン作成の負荷が大きかったことから、プロセッサのテストパタンの作成を支援するツールの整備が望まれる。