4A-4

画像処理用ＬＳＩのメモリコントローラにおける ランダム機能検証適用事例

濱川 洋平† 吉屋 史生† 佐藤 真† 加藤 義幸†

株式会社 東芝†

1．はじめに

現在の半導体市場で成功を収めるためには、高機能化・コスト・開発期間の 3 点が重要とされるが、ハードウェアの設計もより複雑化してきており、それらの機能検証による設計品質の確保は大きな課題となっている。複雑化した機能に対し、従来の人手によるテストベクタ作成という検証方法では対応しきれず検証漏れによる設計品質の低下や、拡張性の乏しさが問題となっている。そこで、検証用言語を用いて検証環境を構築し、大量のテストベクタをランダム生成することで網羅的で品質向上を求めるランダム検証手法が近年注目を浴びている。

本稿では、現在弊社で開発を進めている画像処理用 LSI のメモリコントローラへのランダム検証適用事例を紹介する。

2．画像処理用 LSI について

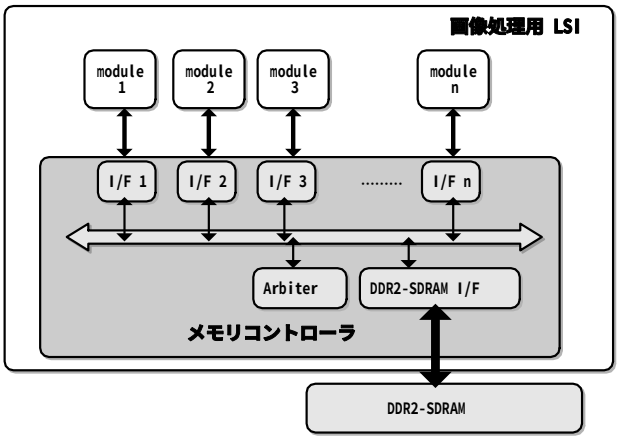

図 1：画像処理用 LSI の構成

現在弊社では次期 AV 機器等への搭載が予定される、画像処理用 LSI(図 1)を開発している。

本 LSI では、内部の各ブロックモジュール(バスや Video decoder、 Audio decoder 等がある)が、外部に接続された DDR2-SDRAM をアクセスできるよう UMA(Unified Memory Architecture)のメモリコントローラを備えている。メモリコントローラは、各ブロックモジュールからのアクセスを、DDR2-SDRAM の使用効率を高く維持しながら競合を調停し、メモリアクセスを行なう構造となっている。そのため、起こりうる内部状態も膨大な数となり、機能検証に十分なテストベクタを人手で作成するのが困難であるという課題があった。

3．ランダム検証環境の構築

今回構築した検証環境の全体構成を図 2に示す。検証環境には対向モデル群、チェッカ、モニタ・カバレッジ収集機能等が含まれ、検証対象であるメモリコントローラを取り囲んでいる。以下、これらの検証環境要素を項目別に示していく。

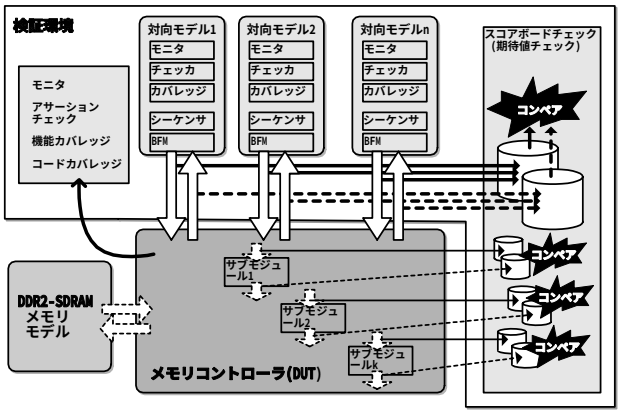

図 2：検証環境

3.1. 対向モデル

メモリコントローラにインタフェースを持つ内部モジュールの動作を対向モデル(1～n)としてそれぞれ検証用言語により記述する。モデルには、各インタフェースのプロトコルに準拠したバスモデル(以下 BFM[Bus Function Model]と呼ぶ)と、BFM を制御するシーケンサ、入出力信号のモニタ・カバレッジ・チェック機構を持たせている。また、開発工程やモデルの再利用性を考慮し、作成するモデル群を以下のように分類した。

- 汎用性(バス等)の高いもの
- 今回の検証独自のもの

汎用性の高いものに対しては、検証用 IP として使えるよう開発に多くの時間を割いたが、今

Example of random functional verification on LSI for image processing

† Embedded System Platform Development Department, CORE-TECHNOLOGY CENTER, TOSHIBA CORPORATION

回の検証独自のものに関しては最低限の機能だけ実装し、他にも検証用 IP を流用することでトータルの検証環境開発期間を削減した。

3.2. テストシーケンス

対向モデルはそれぞれシーケンサ(図 3のシーケンサ 1～n)を持ち、従来の手続き的なものからランダム性の高いものまで、様々なテストベクタの生成を制御できるようにしている。今回はこれらのシーケンサを制御する LSI トップレベルの仮想シーケンサを定義し、図 3のような階層構造することで、効率良くテストベクタ生成できるようにした。

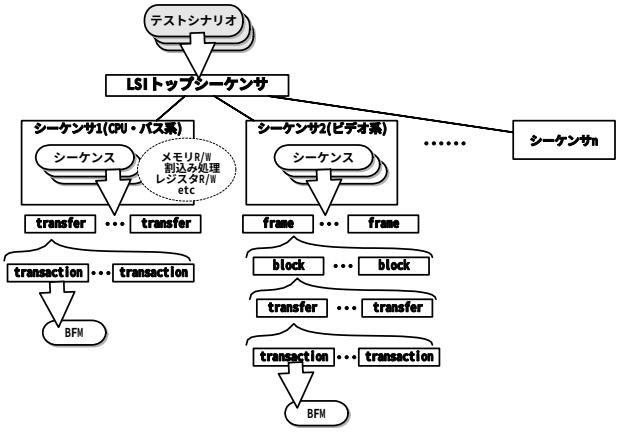

図 3：テストシーケンス階層図

3.3 チェッカ

検証対象が仕様通りに動作しているかを判定するため、テスト実行中に自動的にデザイン仕様を確認するチェッカを実装した(図 2)。ここではまずマクロな観点からメモリアクセスの基本機能の期待値チェック機構を実装し、次にサブモジュール毎に入出力データ間期待値チェッカを持たせた。また、期待値チェックが難解な箇所等についてはアサーションチェッカを実装することで、トータルで必要十分なチェック機構を備えた。

3.4 カバレッジ

テストパターンが十分かどうかの判定に、従来から、RTL コードの網羅性を計る“コードカバレッジ”が実施されていた。今回は、検証対象の網羅したい機能を検証用言語で記述・収集する"機能カバレッジ"を併用することで、検証の網羅性向上を狙った。機能カバレッジ収集は、シミュレーション中に適切なサンプリングイベントを定義し、イベント発生時の RTL 内部変数・検証モデル内部変数を記録することで行なわれる。ここでは各モジュールアクセスの調停パターン・データ転送数等定義した。

4．検証の成果

今回は、従来の手続き的検証手法とランダム検証手法を同時に適用してメモリコントローラの機能検証を行なった。特にランダム検証では、従来手法で検出できなかった複数の不具合を発見することができた。またコードカバレッジ解析も、両手法が互いの未達成箇所を補間し合うことで、100%のカバレッジ率が達成された。

5．考察

今回のランダム検証で発見された不具合は、いずれも”複雑な競合状態”に起因するコーナーケースにおけるものであった。これらは従来検証手法ではテストベクタの作成はもちろん、テスト項目をリストアップすることすら困難であったと考えられ、ランダム検証の有効性を確認することができた。以下に考察を述べる。

機能カバレッジ項目の抽出について

結局は検証対象仕様から検証項目を定義する作業であるため、漏れなく項目抽出することは、従来検証手法と同様に困難である。ただし、一覧性の良いカバレッジレポートやクロスカバレッジ等の検証ツールのサポートを利用することで、検証進行段階においても検証開始当初にリストアップできなかった項目を発見でき、随時項目追加していく方法もとれる。

ランダム検証が有効な検証対象について

複数の競合がランダムに起こるようなモジュール(アービター、バス等)には、ランダム検証が非常に有効であると考える。また、そのような検証対象の規模が大きくなり、人手でテストベクタ作成が困難になるほどランダム検証が効果的になってくる。その反面、チェックアビリティ、テストシーケンスの制御が複雑になり検証精度の低下につながる問題が起こる可能性もある。ただし、本稿で示したよう、適度な規模のサブモジュール毎の期待値チェッカや、アサーションチェッカの実装で必要十分なチェック機構を備えることも可能になると思われる。また、テストシナリオを実行するシーケンサを適切な階層構造にすることで、複雑なテストシーケンスも制御可能になると考える。

5．最後に

開発中の LSI 内部モジュールにランダム検証を適用することで、ランダム検証の有効性を確認することができた。今後の課題としては、ランダム検証対象の拡大、ノウハウと検証用 IP の蓄積、さらなる検証の高品質化が挙げられる。