複数のSoC設計プロジェクトにおけるエミュレーション性能の向上

ケイデンス・デザイン・システムズ社　Frank Schirrmeister

設計の規模が大きくなれば、検証作業の規模も大きくなります。実際、検証はSoC開発における最大の課題になってきており、ハードウェア自体とハードウェア/ソフトウェアのインターフェースの検証は開発コストの大部分を占めています。現在、企業内で複数のチームに複数のSoC設計を配分し、並行して進めることは珍しくありません。数十のプロジェクトのエミュレーション用リソースを共有していることで知られているチームもあります。この文書では、複数の複雑なSoC設計の完成と性能目標を効果的かつ効率的に管理する目的で、エミュレーション性能と設計チームの生産性を大幅に向上させるために必要な観点について説明します。

目次

はじめに

2014年の平均的な設計サイズはおよそ1億8千万ゲートでしたが、Semico社によると、2018年には平均3億4千万ゲートになると予想されています。もちろん、これは単なる平均値にすぎず、10億ゲートを超える設計も存在します。大規模なSoCは通常、数百のクロックドメインとIPブロック、およびヘテロジニアス/ホモジニアス統合プロセッサのコアから構成されています。これらのコンポーネントすべてを確実に意図どおりに動作させるには検証が必要です。コア上でソフトウェアを実行することにより、検証に関する課題と複雑さが発生します。このため、設計全体の規模の増大は重要ですが、各コンポーネントのサイズも考慮する必要があります。さらに、現在の市場のプレッシャーによって、設計スケジュールは以前にも増してタイトになっています。図1の高度なSoCの例を参照してください。

設計サイクルの短縮化に呼応して、ハードウェア設計の再利用や高度な電源管理技法の適用例も増えており、このような場合、機能は組み込みソフトウェアに実装されています。設計の再利用や電源管理により、さまざまなレベルのIPインタラクションからハードウェア/ソフトウェアコンポーネント間のインタラクションまであらゆることを考慮する必要が生じ、その結果、検証の課題が増えています。

これらを考慮すると、小規模なIPブロックからSoCレベルのサブシステムに至る、さまざまなサイズと実行時間を持つタスクを同時に処理できるハードウェア/ソフトウェア検証用のプラットフォームが必要になります。また、この検証作業を企業内で並行して開発中の他の設計にも展開する必要があります。

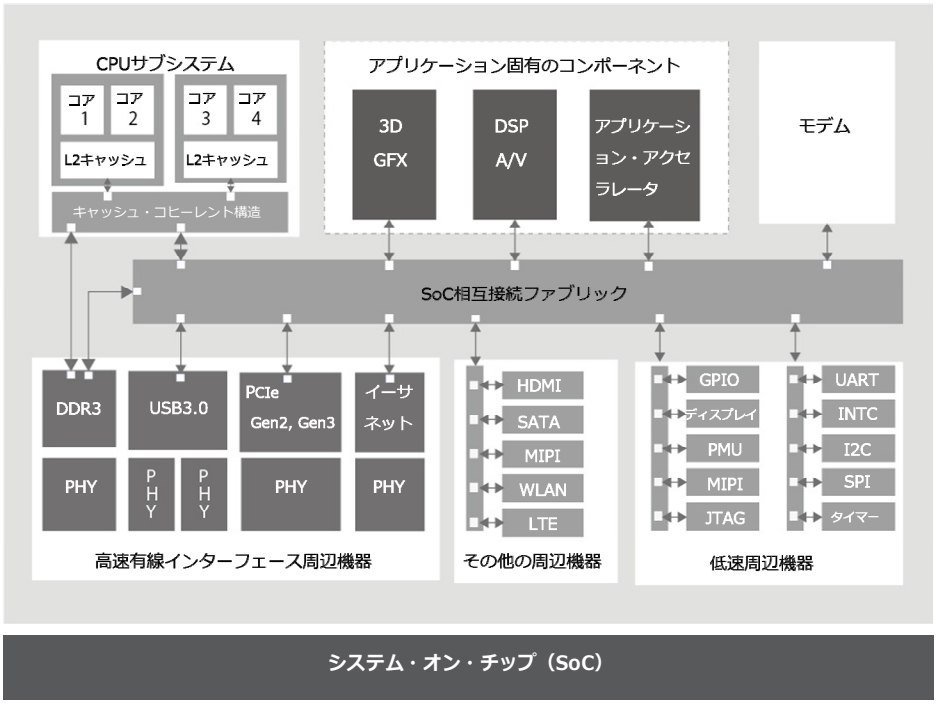

図1：高度なSoCアーキテクチャ

エミュレーション性能向上のループ

効果的なシステム検証フローには、多くの技法が統合されています。

- 仮想プロトタイプ作成は、RTLが利用できないソフトウェア開発の初期段階を支援します。また、RTLが利用可能になって、RTLとのハイブリッド構成でトランザクション・レベル・モデル（TLM）を実行できるようになると、システムのハードウェア/ソフトウェアのインターフェースを検証します。
- シミュレーションによる高度なRTL検証は、現在あらゆるプロジェクトで使われている重要なリファレンスです。速度に限界があるのでハードウェア検証が対象ですが、立ち上げ時間と高度なデバッグでは他の手法の追随を許しません。
- シミュレーションのアクセラレーションは、RTLハードウェアをソフトウェアベースのシミュレーションとハードウェア補助による実行として検証し、既存のシミュレーション検証環境を再利用可能にします。
- シミュレーションは、設計の迅速な立ち上げとシミュレーション的なデバッグにより、さまざまなサイズと時間のジョブを並行して実行可能なハードウェア/ソフトウェアの同時検証手段として効果的なのは確かです。
- RTLの完了後、FPGAベースのプロトタイプ作成を利用すれば、早期のソフト開発、システム検証、性能の不具合に対応するためのプリシリコン・プラットフォームを提供できます。

プラットフォームおよびROIのエミュレーション性能を考える際に、エミュレーション性能向上のループの主な4つのステップ（図4）と、各ステップで生じる疑問を考えておくことは重要です。

- ビルドの過程では、検証ジョブをビルドし、さまざまな仕事量やタスクに対応するデータベースをコンパイルします。次のような疑問を考慮しておく必要があります。コンパイラの速度は？RTLとゲートレベルのネットリストでは、効率と自動化レベルはどれくらいか？使い勝手はいいのか？サポートする構造の数は？データベースのビルドの速度は1つのパラメータにすぎません。特に、設計のさまざまなシナリオに対応する中でユースモデルが変化していくので、リソースの効果的な利用も重要な点です。
- 割り当てとは、特定のリソースに対して作業量をできるだけ効率的に割り当てることです。ここで次のような疑問について考えてみましょう。システムが並行して処理できるジョブ数は？ユーザーが大きく介入せずに、どれくらい迅速に別のリソースに割り当てられるか？要求される負荷に応じて、タスクに優先順位を付けてスワップする方法は？これらすべてが、コンピューティングリソースがチームの検証作業を行う際の作業効率に大きく影響します。

- 実行ステップとは実行そのもののことです。システムの実行速度の向上に注力しがちですが、それがすべてではありません。このステップは、ユースモデルと、そのユースモデルのインターフェース・ソリューションにも関連しています。次のような疑問を考慮しておく必要があります。システムは設計の優先順位に基づいてジョブを実行しているか？システムが並行して実行可能なジョブ数は？ジョブの一時中断や再開は可能か？必要なポイントで保存して、後で再開可能か？
- デバッグは、プリシリコン、ポストシリコン両方のバグの検出を目的とします。次のような疑問を考慮しておく必要があります。検証プラットフォームを使って、バグを特定するための再テストの必要性を最小限に抑えながら、どのくらいうまくデバッグできるか？再コンパイルせずに、どのくらい効率的にトリガーを動的に調節できるか？ハードウェア、ソフトウェアの両方の解析で、デバッグが必要な大きなウィンドウをキャプチャするのに十分なデバック・トレースがあるか？それとも、デバッグ対象のデータを得るために複数回実行する必要があるか？

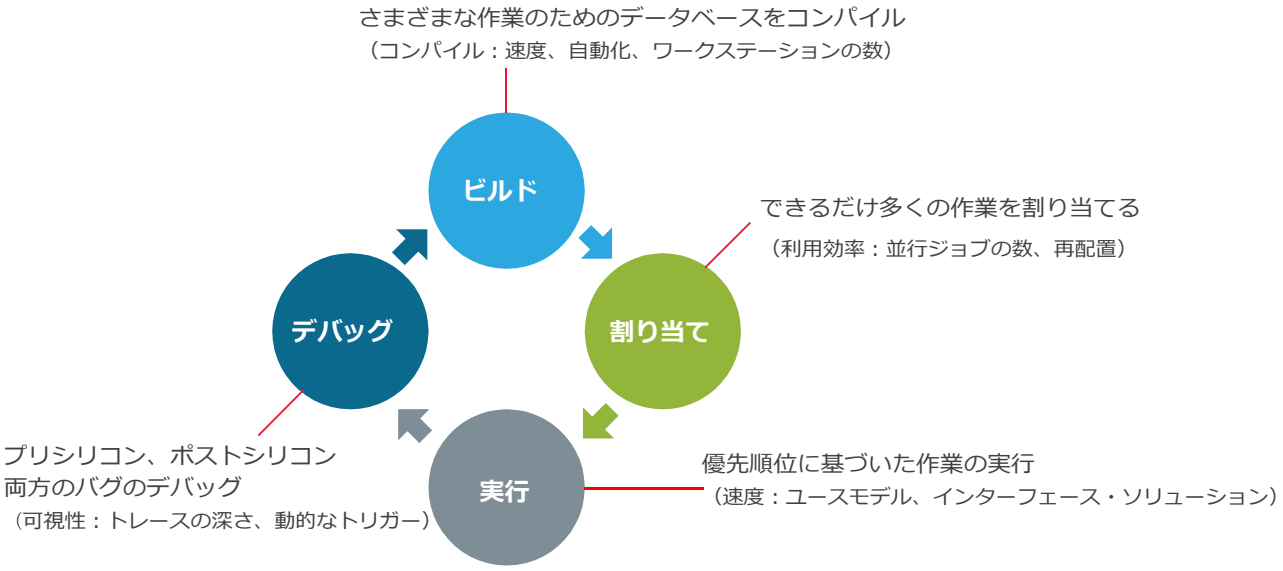

図2：エミュレーション性能向上のループ

これらの各ステップは、エミュレーション性能全般にそれぞれ影響を与えます。例えば、デバッグ解析は、データキャプチャや生成の要件を考慮すると、FPGAベースのプロトタイプ作成とFPGAベースのエミュレーションの速度を大幅に落とします。このため、設計ごと、フェーズごとにどの程度のデバッグが必要かを決定する必要があります。エミュレーションの速度に影響せずにデバッグを実行できるプロセッサベースのエミュレーションを使うなどの調整を行うこともあります。

図3から分かるように、このエミュレーション性能向上のループは、初回立ち上げからシリコン・テープアウトまでの間に、設計に組み込む機能の統合、検証と同時に複数回並行して繰り返されます。つまり、チーム内のエンジニアは、ソースの編集、チェック、コンパイル、チェックイン、作業のピアへの統合の各プロセスを踏みます。フローを経て、最終的にシステムは確定的かつコヒーレントになります。機能セットを決定し、リグレッションを実行した後も、テープアウトまでエミュレーションのループを継続しますが、頻度は低くなります。

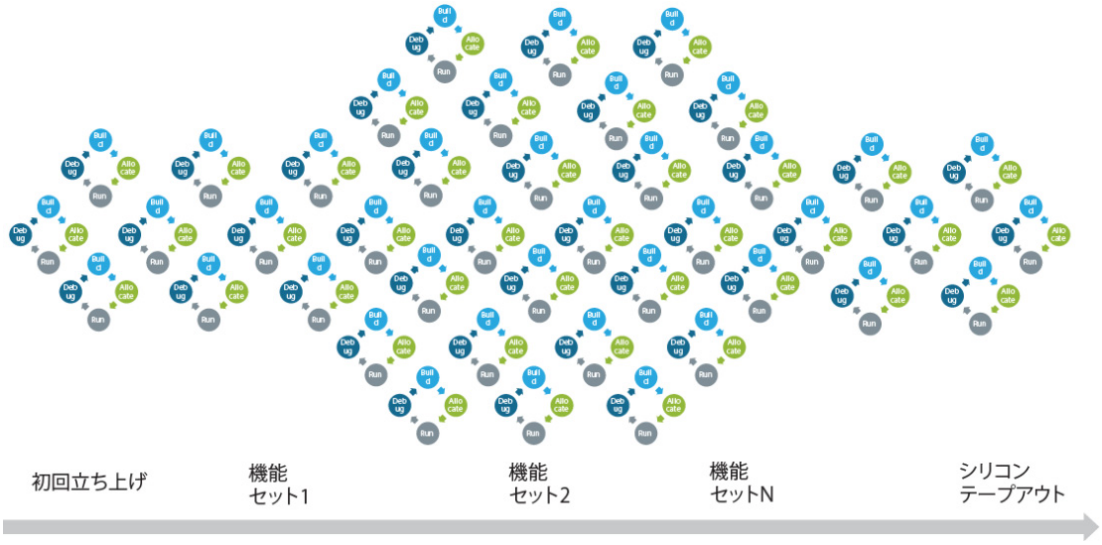

図3：検証生産性ループは設計サイクルで複数回繰り返されます。

エミュレーション性能の条件を包括的に考えることの重要性

エミュレーション性能を評価する際は、複数の条件を包括的に考えることも重要です。エネルギーコストについて考えてみてください。この値を、本来の消費電力の積算値として算出することもあるでしょう。しかし、本来の消費電力は消費電力の一部にしかすぎません。例えば、FPGAベースのエミュレータとプロセッサベースのエミュレータを比較してみると、プロセッサベースのエミュレータの合計消費電力の方がFPGAベースのエミュレータより少なくなることに気が付くでしょう。これは、コンパイル（FPGAベースのシステムでは大規模な並列サーバー・ファームが必要になる場合が多い）、割り当て（エミュレーションのリソースの精度が十分高ければ、プロセッサベースのエミュレーションの方が作業効率が高い場合がある）、実行、トレース用バッファのサイズが適切でない場合に元の量のデバッグ情報を生成するために必要な複数回のデバッグ実行など、検証タスクで消費する電力を合計して算出した場合です。

さらに考慮すべき重要な点は、ジョブの精度とゲートの使用率です。システムによっては、無駄なエミュレーション用スペースを最小化して実行時間を短縮できるものがあり、この点で他のシステムより設計が優れていると言えます。サポートするジョブ数だけの問題ではありません。例えば、きめ細かく許容量を設定したシステムは、必要最小限のリソースでより多くの並行ジョブをサポートできます。合計所要時間を計算するには、ジョブの数、システムで実行可能な並行ジョブの数、それらの実行およびデバッグ時間を考慮に入れる必要があります。

企業レベルの検証コンピューティングのニーズへの対応

世界中に分散している複数のチームによって複数の設計プロジェクトが進行中の場合、検証用のコンピューティングシステムだけでは十分ではありません。企業レベルの検証用コンピューティングには、次の要件を満たすシステムが必要になります。

- 性能や消費電力の目標範囲内で動作して、数十億ゲートの設計に拡張可能
- 多数の日常的な設計変更をサポート可能
- 必要なデバッグ情報を生成するのに十分なトレースの深さ、高速な解析およびコンパイルなど、高いデバッグ生産性を提供
- 特定のジョブをロジックボード上の他のドメインに割り当てて、コンピューティングリソースを最大限活用できる柔軟性を提供
- ジョブを不連続部分に分割して、システム内で利用可能なリソースにマッチさせることにより、効率的な動的リソース割り当てを提供
- ジョブを特定の物理インターフェースに依存させず、ターゲットとジョブを仮想的に切り替え可能

end-to-endフローの利点

これまでの条件を考慮すると、end-to-endフローの利用には確かに利点があります。エミュレーション用システムが接続された一連のプラットフォームの一部である場合、相互運用可能なプラットフォーム間、およびハードウェア/ソフトウェアドメイン間で簡単に移行して、システムレベルの設計、統合、検証タスクを完了できます。フローを統合することで、設計上のリスクを最小化しながら設計チームの生産性を大幅に向上できます。

Cadence System Development Suiteの中に、ハードウェア/ソフトウェアの設計および検証をサポートする統合開発プラットフォームの例が見つかります。このパッケージの統合プラットフォームにより、プリシリコンの早期のソフト開発、IP設計および検証、サブシステムおよびSoCの検証、ネットリストの検証、ハードウェア/ソフトウェアの組み込みおよび検証、システムおよびシリコンの検証を含む、システム統合に要する時間を最大50%短縮できます。

要約

設計チームがハードウェアリソースを最大限効率的かつ効果的には利用していないなら、貴重な生産性を損ねていることになります。現在の大規模かつ複雑なSoCにおいて、複数ユーザーによる高レベルな検証性能と生産性は、競争上の優位性全体にとって欠かせない要素です。従来の多くのコンピューティングリソースは、これらの条件を満たしていません。一方、現在の設計を迅速にテープアウトさせるためには、大容量で効率的なジョブ割り当て、高速な実行時間、大規模なデバッグなどの機能が必要なのです。

cadence

日本ケイデンス・デザイン・システムズ社

本社／〒222-0033　神奈川県横浜市港北区新横浜2-100-45
営業本部　HSV営業部
TEL: (045)474-9407 FAX: (045)476-3406
〒541-0054 大阪府大阪市中央区南本町2-6-12 サンマリオンNBFタワー16F
TEL.(06)6121-8095　FAX.(06)6121-7510
URL https://www.cadence.co.jp/

© 2015 Cadence Design Systems, Inc. All rights reserved worldwide.
Cadenceおよび Cadence ロゴ は、Cadence Design Systems, Inc.の登録商標です。
その他記載されている製品名および会社名は、各社の商標または登録商標です。
＊ 掲載の内容は、2015年11月現在のものです。