6 System Level Design

ハードウェア／ソフトウェア協調シミュレーション技術

(株)東芝 ソフトウェア技術センター
野々垣 直浩
nobuhiro.nonogaki@toshiba.co.jp

SoC開発におけるハードウェア／ソフトウェア協調シミュレーション技術への期待

今日，処理能力の高いマイクロプロセッサやディジタル信号プロセッサ（DSP, Digital Signal Processor）と，ハードウェア・ロジック（特定用途向けの論理演算回路）を混載することを特徴としたSoC（System-on-Chip）が，モバイル通信端末やディジタル家電を中心に利用されている．

マイクロプロセッサやDSPで実行されるソフトウェアの利点と，ハードウェア・ロジックの利点を組み合わせることで，画像処理や信号処理等，高い演算スループットを必要とするアルゴリズムの実現においてSoCの特徴は特に発揮される．

こうしたSoCを効率よく開発するためには，マイクロプロセッサ，メモリ，バスや，ハードウェア・ロジックの開発を担当するハードウェア開発チームと，マイクロプロセッサやDSP上で動作するソフトウェアの開発を担当するソフトウェア開発チームが，密に連携をとりながら並行的に開発することが不可欠になる．

SoC開発は概念的には図-1のようないくつかの専門チームに分かれて，開発作業が進められる．明確に分け

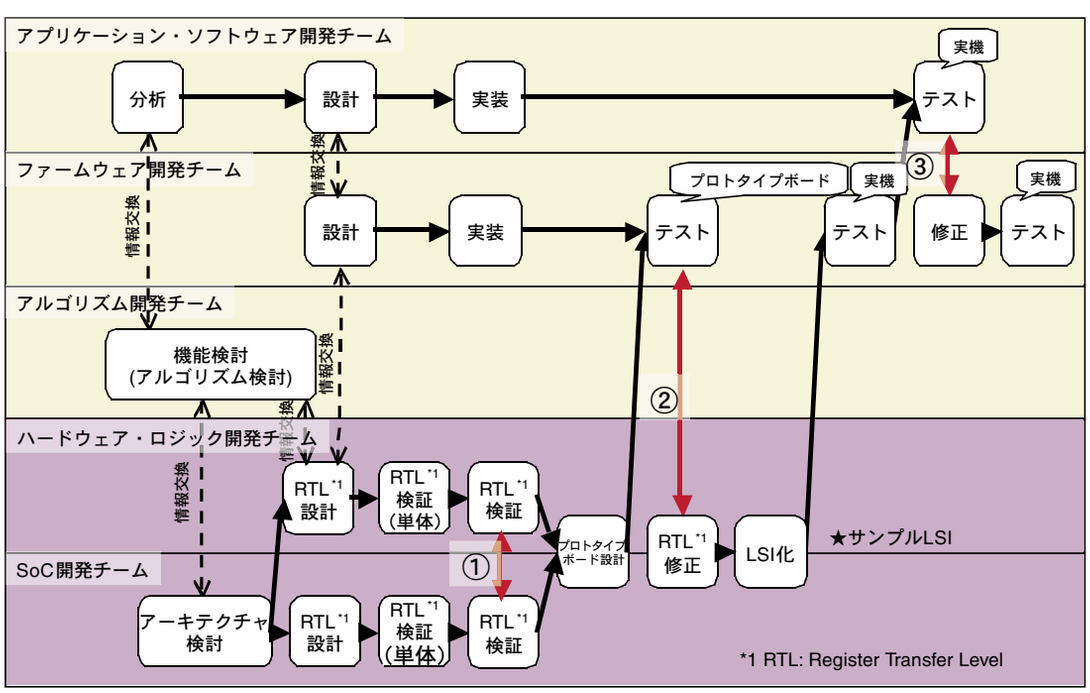

図-1　SoC開発における開発チームの協働

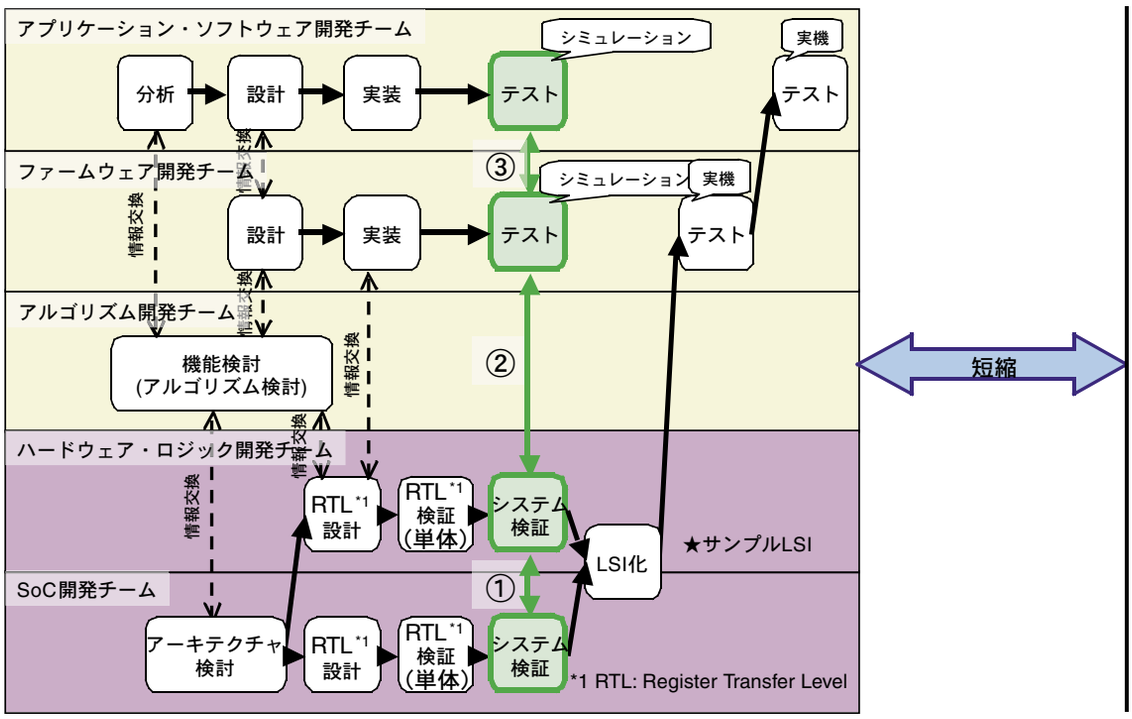

図-2　SoC開発におけるハードウェア／ソフトウェア協調シミュレーション技術のインパクト

られたチームとして編成されることもあるが，実際には図中のハードウェア・ロジック開発チームとSoC開発チームを兼務するようなハードウェア開発チームとして編成されたり，検証チームなどのかたちで特定の工程を担当するチームに分けて編成されたりすることも多い．

従来型のSoC開発では図-1中の①，②，③の各ポイントで，それぞれのチームの成果物を統合し，テスト／検証が行われる．従来型のSoC開発では，ファームウェアやアプリケーション・ソフトウェアの最初のテストが開発後期に位置し，かつSoC開発全体のクリティカルパスにのるケースが多かった．結果的にこれがソフトウェアの品質，納期が大きくバラつく原因になっている．また，実機テストの段階でハードウェアとのインタフェースに問題が発見されたとしても，ハードウェア側の大幅な変更は期間的に難しいため，ファームウェア等で吸収するなどの対症療法的な対策が必要となるケースがあり，こうした対策の痕跡がソフトウェア保守時の生産性低下につながっている．

こうした課題を解決するためには，ハードウェア開発チームとソフトウェア開発チームが円滑にコミュニケーションを進めて，双方にかかわる問題を早期に発見／切り分け／対策を施し，開発前半で品質を作り込むことが重要である．ハードウェア／ソフトウェアの双方にかかわる問題を早期に発見するため，できるだけ早い段階でチームの成果物を統合してテスト／検証を行う必要がある．テスト／検証工程を先行して実施する手段として，ハードウェア／ソフトウェア協調シミュレーション技術が期待されている．

ハードウェア開発チームとソフトウェア開発チームが段階的に開発作業を進めるには，機能モデル，トランザクションレベルモデル，組み込みソフト実装モデル等の複数のモデルが使い分けられる[1)～3)]．この各モデルを用いて段階的にハードウェア／ソフトウェア協調シミュレーションを行うことで，図-2のようにサンプルLSIができあがる前に一度，システムを構成する要素の大部分を検証することが可能になる．実機によるテストおよび修正期間が短縮され，SoC開発全体の短納期化およびSoCの品質向上が望める．またハードウェアが完全にフィックスされる前の段階で，ハードウェア開発チームは，ソフトウェア開発チームからのフィードバックを受けることができるため，ファームウェア等で吸収せざるを得なくなるなどのリスクを低減することができる．

しかし，近年，ソフトウェア，ハードウェアともにその規模は急速に膨張しているため，すべてを一括してハードウェア／ソフトウェア協調シミュレーションを行おうとすると，シミュレーション処理時間が非常に長くなってしまう．このシミュレーション処理時間の大幅な増加が，図-2のような開発工程を組む上で障害になっ

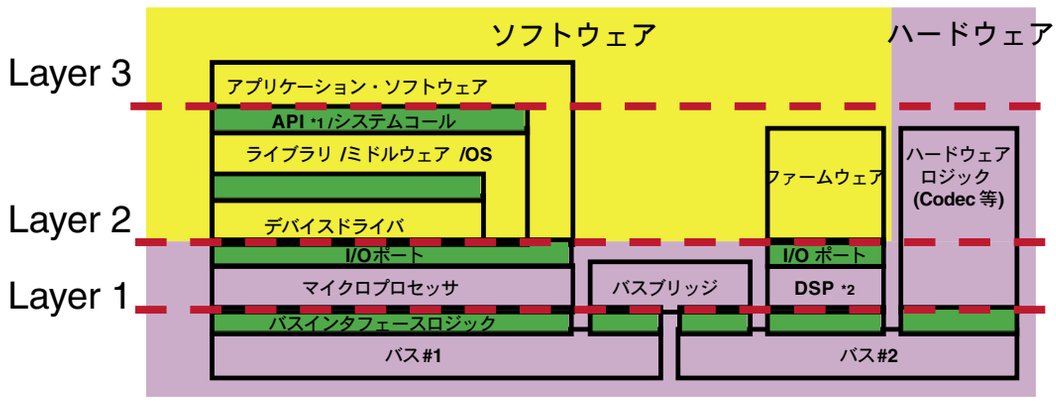

図-3　SoCが持つハードウェアとソフトウェアの階層構造

ている．

SoCは複数の階層から構成されており，各階層の開発ごとに必要な検証目的が異なる．そこで，検証目的の違いに基づいて複数のシミュレーション階層を設定することによって，個々の検証ごとに必要十分な精度を定義することができる．処理時間と精度はトレード・オフの関係にある[4]ため，こうしたシミュレーション階層の定義によって，処理時間の短縮を期待することができる．

ハードウェアとソフトウェアのインタフェース

SoCを構成するハードウェアとソフトウェアは，いくつかの階層によって構成されていると見ることができる．この階層構造はシステムの用途，マイクロプロセッサの種類，DSPの有無，OSの有無，などの条件によって詳細は異なるが，多くのシステムが，図-3のような階層構造を持っている．これらの階層構造を構成する各階層は原則的に，上位の階層に対してインタフェースとして規定した振舞いを提供して，より下位階層の実装の詳細を隠蔽するよう実現される．

本稿ではSoCの階層を，

- Layer 3アプリケーションの階層
- Layer 2メカニズムとアルゴリズムの階層
- Layer 1システム・アーキテクチャとオンチップ通信網の階層

の3階層に分けている．ハードウェア開発チームとソフトウェア開発チームの双方にかかわる検討課題が，それぞれの階層を開発するにあたってどのように現れてくるのか，階層ごとに整理する．こうした検討課題の特性に適したかたちでハードウェア／ソフトウェア協調シミュレーション環境を構築することが，開発の効率化を達成するために重要である．

Layer 3アプリケーションの階層

アプリケーションの階層は，利用者が直接触れるユーザインタフェースや，外部システムの制御ロジックなど，システムの用途固有のロジックを実現する．この階層は主にソフトウェアで実現されることが一般的である．たとえば，Webブラウザや，録画予約ユーティリティのような機能がアプリケーション・ソフトウェアとして実現される．

アプリケーション・ソフトウェアは，OSやデバイス・ドライバ，ライブラリによって，実装の詳細が隠蔽され仮想化されたハードウェアを，システムコールやAPI（Application Programming Interface）を通して利用する（図-4）．

アプリケーション・ソフトウェアの開発では，機能やサービスの過不足，他の機能やサービスの呼び出し手続きの合理性等が，インタフェースの課題となりやすい．実装の詳細は隠蔽されて仮想化されたハードウェアとして扱われるため，この課題にはOSやライブラリといったソフトウェアの課題と論理回路の課題が混在することもある．

Layer 2メカニズムとアルゴリズムの階層

限られたシステム資源についての，複数のアプリケーション・ソフトウェア間の調整を行うメカニズムや，アプリケーションによって共通に使用される高度なアルゴリズムを実現する．共通に使用される高度なアルゴリズムとは，たとえば動画像の圧縮伸張，暗号処理，通信プロトコルの処理等がある．

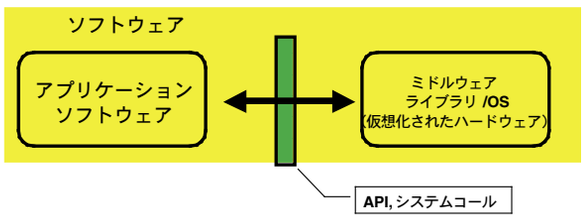

図-4　ハードウェアとソフトウェアのインタフェース（アプリケーションの階層）

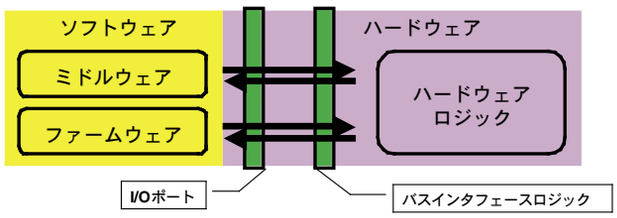

図-5　ハードウェアとソフトウェアのインタフェース（メカニズムとアルゴリズムの階層）

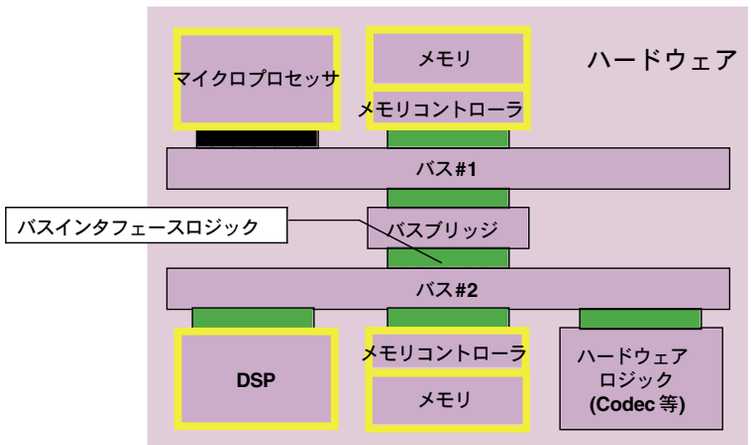

図-6　ハードウェアとソフトウェアのインタフェース（システム・アーキテクチャとオンチップ通信網の階層）

Layer 2メカニズムとアルゴリズムの階層は，マイクロプロセッサ上で動作するソフトウェアやハードウェア・ロジックとして実現される．ソフトウェアによる実現部分とハードウェア・ロジックによる実現部分が密接に結合して，1つの機能を実現する場合もある（図-5）．

この階層にあるソフトウェアは，特定のマイクロプロセッサ固有の機能，シリアル通信，DMA,タイマ等の周辺回路の機能，アルゴリズムの一部または全部が実現されたハードウェア・ロジックの機能等を，IOポートやOSのプリミティブなシステムコール等を通して利用する．たとえば，この階層にあるソフトウェアには，ブート・プログラムやデバイス・ドライバ，ハードウェア診断プログラム，ハードウェア・ロジックと協調動作するミドルウェアやファームウェアなどがある．

ハードウェア・ロジックは，計算量の多いアルゴリズムの一部または全部を実現し，バス・インタフェース等を通して，その機能制御手段を提供する．

ハードウェア・ロジックの状態は，ソフトウェア側からはメモリやIO空間上にマップされたコントロール・レジスタやステータス・レジスタとして見える．一方，ハードウェア・ロジック側からは，アクセス方法（readやwrite等）と，アクセスされたレジスタを識別するアドレスおよびデータの組として見える．

この階層を実現するソフトウェアとハードウェアは機能的に密接に結合することが多いため，それぞれで実現する機能の役割分担（ハードウェア／ソフトウェア分割）と，ハードウェア／ソフトウェア間の処理シーケンスについて十分に検討することが重要である．複数のチームが協調して開発を進めるためには，ソフトウェア側から見たハードウェアの動作，ハードウェア・ロジック側から見たソフトウェアの動作について，相互に十分理解する必要がある．

■Layer 1システム・アーキテクチャとオンチップ通信網の階層

マイクロプロセッサ，DSP，メモリ，ハードウェア・ロジック等のハードウェア同士の通信を実現する．

Layer 1においては，ソフトウェアはマイクロプロセッサから発行されるメモリや周辺機器に対しての要求，またはメモリ上に格納されたデータとして観察される（図-6）．

この階層の開発では，マイクロプロセッサの選択，メ

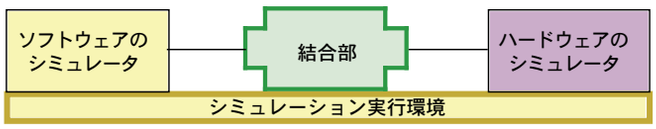

図-7　ハードウェア／ソフトウェア協調シミュレーション環境の構成

モリシステム構成や通信路（バスやインターコネクション）構成の選択などが重要な検討課題である．特にバスを中心にシステムを構成する場合，システム全体の性能要求（レスポンス，スループット，レイテンシ等）にバスが大きなインパクトを与えるため，特に通信路への負荷の重い場合（e.g. 画像処理アルゴリズム等）について，十分な検討が必要になる．

■ハードウェア／ソフトウェア協調シミュレーション環境の構成

ハードウェア／ソフトウェア協調シミュレーション環境は，
(a) ハードウェアのシミュレータ
(b) ソフトウェアのシミュレータ
(c) ハードウェアのシミュレータとソフトウェアのシミュレータをつなぐ橋渡し
で構成される（図-7）．

(a) のハードウェアのシミュレーション手法，(b) のソフトウェアのシミュレーション手法の組合せから，(c) の橋渡し方法が決まる．

そこで，ハードウェアのシミュレーション手法とソフトウェアのシミュレーション手法についてそれぞれ述べ，その後，Layer 1～3について，検証目的に合わせたハードウェアのシミュレータとソフトウェアのシミュレータを選択して，各Layerに適したハードウェア／ソフトウェア協調シミュレーション環境の構成について述べる．

■ハードウェアのシミュレーション手法

ハードウェアのシミュレーション手法は，(1) ハードウェア記述を解釈してハードウェア動作をシミュレーションするHDL（Hardware Description Language）シミュレータを用いる方法，(2) プログラミング言語（C言語やC++等）や，プログラミング言語から派生したシステムレベル設計言語（SystemC, SpecC等）を用いて作成した実行可能なハードウェアモデルを利用する手法[5), 6)]の2つに大別できる．

(1) HDLシミュレータを用いる場合，原則的にハードウェア記述を変更する必要がなく，ハードウェア・ロジックの設計者が使い慣れた環境でシミュレーションを行うことができる．ハードウェア記述を直接解釈して詳細なシミュレーションを行うため回路の接続ピンや配線単位での詳細な動作まで観察することができる．しかしながら，細かいシミュレーション精度を実現するために処理時間がかかる．

(2) システムレベル設計言語等で実行可能なハードウェアモデルを使用する場合，シミュレーション目的に合わせて適切にシミュレーション内容を省略したモデルを作成することができる[7)]．このため，(1) の方式に比べて，大幅にシミュレーション処理時間を短縮できる可能性がある．また，C言語等のプログラミング言語とのインターオペラビリティが配慮されているため，開発ホストのソフトウェア開発環境とも，一定の親和性があることも環境構築の上でメリットになる．

一方，従来のハードウェア記述とは別に，あらたにモデルを作成する手間がかかる，モデルの詳細度を決める基準がはっきりしていない，システムレベル設計言語で記述したモデルの再利用性や，複数の抽象度のモデルを作成した場合のモデル間での等価性の保証方法など課題も多い．

■ソフトウェアのシミュレーション手法

ソフトウェアのシミュレーション手法は，(1) ホストネイティブコードにコンパイルしてそのまま実行する方法と，(2) ターゲットコードにコンパイルしてそのオブジェクトコードを命令セットシミュレータ上でシミュレーションする方法の2つに大別できる．

(1) ソフトウェアの開発環境がある開発ホスト用にソースコードをコンパイルして，ホストネイティブコードとして実行する方法は，シミュレーション処理時間が短いという点で大きな特徴を持つ．また，開発ホスト用のデバッガ等のソフトウェア開発環境も利用することができる．機器利用者に対するユーザビリティが求められるユーザインタフェース部や，機器の設定ユーティリティなどのソフトウェア開発では有用である．その反面，ホスト環境とターゲット環境の差異を吸収できるようにす

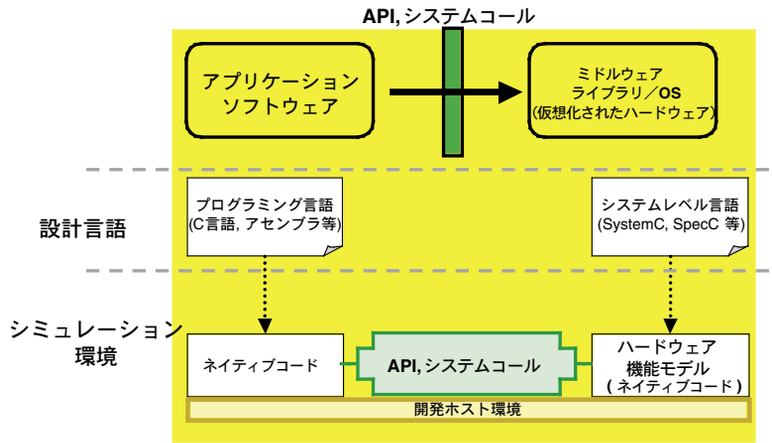

図-8 “Layer 3アプリケーションの階層” のシミュレーション環境

るなどソフトウェア実装上の工夫が必要になり，また周期処理や割り込みといった時間がかかわる処理のシミュレーションには向いていない．

(2) 命令セットシミュレータを用いる方法は，開発しているソースコードを原則として変更しなくてもよいことがメリットである．命令セットシミュレータは，オブジェクトコードの命令単位で解釈してシミュレーションするものが多いが，さらに詳細にマイクロプロセッサのパイプライン状態のシミュレーションや，キャッシュの状態のシミュレーション，動作クロック精度での詳細なシミュレーションが可能なプロセッサシミュレータも存在する．(1) の方法に比べ，シミュレーション処理時間は長くなる．

■ハードウェア／ソフトウェアの協調シミュレーション手法

ハードウェアのシミュレーション手法と，ソフトウェアのシミュレーション手法を組み合わせて，各階層の検証目的に合わせたハードウェア／ソフトウェア協調シミュレーション環境は構築される．

“Layer 3アプリケーションの階層” のシミュレーション環境

アプリケーションの階層では，利用者や他のシステムとのインタラクション・シーケンスが重要な検討課題である．使用性や理解容易性といった，定性的な品質指標が重要になる．その中で「ハードウェア」やミドルウェア，ライブラリ等が提供する機能やサービスの過不足や呼び出し手続きの合理性などについての検討が行われる．

この階層では，人間が認知できるオーダーの時間的課題を除いて，シミュレーションに時間的正確さは必要なく，利用するハードウェアも，その呼び出し手続きと応答の手順が正確であれば，実際に提供される予定の「機能」の内容については，原則的に高い精度のシミュレーションは求められない．

そこで，こうした検討を行うために効率的なハードウェア／ソフトウェア協調シミュレーション環境は図-8のような構成になる．

Layer 3のハードウェア／ソフトウェア協調シミュレーション環境は，定性的な品質指標も含めてアプリケーション・ソフトウェアを評価するためにプロトタイプとしてホスト環境上で動作するソフトウェアとして実現される．アプリケーション・ソフトウェアが利用するハードウェアまたはミドルウェアやライブラリを同時に開発する予定がある場合，それらの機能の呼び出し手続きと，アルゴリズム，およびそのアルゴリズムに由来するデータの入出力結果のみを保証するハードウェア機能モデルを作成する．このハードウェア機能モデルは，アプリケーション・ソフトウェアにスタブとして挿入し，ハードウェア／ソフトウェア間のデータ転送手続きなどは，関数呼び出し等に置き換えて省略する．

“Layer 2メカニズムとアルゴリズムの階層” のシミュレーション環境

メカニズムとアルゴリズムの階層では，メカニズムやアルゴリズムの正しさ，システム・アーキテクチャとの適合性，ソフトウェアによる実現部分と，ハードウェア・ロジックによる実現部分の役割分担の適切さ等についての検討が行われる．

この階層では，ソフトウェアとハードウェア・ロジックの具体的なシーケンスと，それぞれの実現部分の負荷は重要な検討課題になる．そのためクロックサイクルごとにどのような処理が行われるのか，それが具体的にど

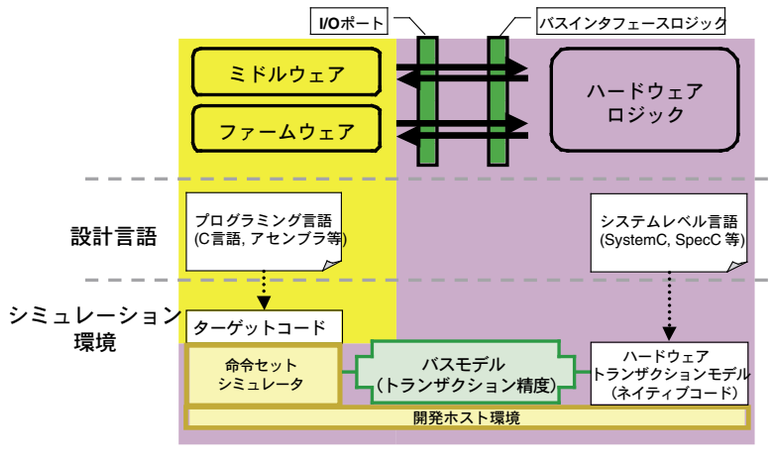

図-9 “Layer 2 メカニズムとアルゴリズムの階層” のシミュレーション環境

のピンの信号なのかを特定するほどの高い精度のシミュレーションは求められない．ただし，クロックサイクル数をもとにして，互いの同期をとるなどクロックに依存するような手続きの場合は，それに依存する部分に限ってはクロックサイクル精度での動作をシミュレーションする必要がある．

こうした要求を満たす，ハードウェア／ソフトウェア協調シミュレーション環境は，命令セットシミュレータとトランザクション精度のバスモデル，トランザクション精度での動作をシミュレーションするハードウェアモデルを用意し，これらをつなぎ合わせることで構成できる（図-9）．

プログラミング言語で記述されたソフトウェアによる実現部分は，ターゲットプロセッサのコンパイラ（とアセンブラ）によってターゲットコードに変換され，命令単位でマイクロプロセッサの挙動として観察する．一方，ハードウェア・ロジックは，バス・インタフェースを通して伝達される read/write 等のトランザクションの組合せに対して，応答するものとして観察できるようにモデルが作成される．

“Layer 1 システム・アーキテクチャと通信網の階層” のシミュレーション環境

システム・アーキテクチャと通信網の階層では通信路のトラフィックと，その同期タイミング等についての検証が行われる．オンチップバスを用いる場合，調停方式や帯域保証方式等の検討が重要である．そのため，トラフィックに強い影響を与える，CPU の負荷，キャッシュのヒット／ミスの解析，バスの使用率等について検討できる必要がある．また，ハードウェアのシミュレーションは少なくとも，クロックサイクル単位およびピンレベルで正確である必要がある．

Layer 1 では，ハードウェア・ロジックだけではなく，それらをつなぐバスとそのアービタ等の制御回路も詳細に RTL でモデル化する．RTL のモデルは従来から使用しているハードウェア記述言語（Verilog,VHDL）や，システムレベル設計言語（SpecC，SystemC）によって記述する．そして，この RTL モデルは HDL シミュレータ（システムレベル設計言語の場合はシミュレーションカーネル）を用いてシミュレーションを行う（図-10）．

クロックサイクル精度のプロセッサシミュレータ上でターゲットコードを動作させることで，実機にのる予定のソフトウェアを，これら RTL モデルのテストベンチとして利用することができる．

また，Layer 2 のシミュレーション環境と併用することで，通信路の性能が特に問題になった状況のみにスポットを絞り，詳細にシミュレーションによって再現して原因調査に利用することもできる．

おわりに

ハードウェア開発チームとソフトウェア開発チームが双方にかかわる問題点の切り分けを行いながら並行開発をすすめる今日の SoC 開発において，その開発期間を短縮する手段としてハードウェア／ソフトウェア協調シミュレーション技術は重要な位置にある．

ハードウェア／ソフトウェア協調シミュレーション技術の導入によって，SoC 開発期間の短縮の効果を得るには，ソフトウェア開発チームおよびハードウェア開発チームの開発プロセス，設計対象構成，開発体制，過去の設計資産の状況等，多角的に注意深く分析して定義していくことが重要である．

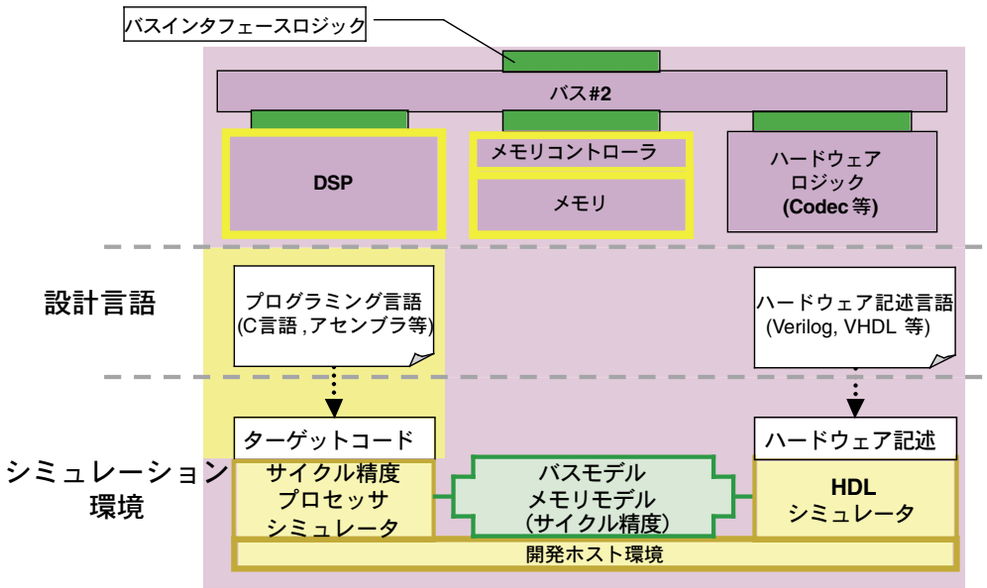

図-10 “Layer 1システム・アーキテクチャと通信網の階層” のシミュレーション環境

この時，SoCの設計階層ごとに必要な検証項目を検討することで，ハードウェア／ソフトウェア協調シミュレーション環境に要求すべき必要十分な精度を定義することができる．シミュレータ精度の保証要求を定義するとともに，保証要求しない精度を明示的に定義することが，精度を定義する上で重要である．

現在，ハードウェア／ソフトウェア協調シミュレーション環境の構成要素となる各種のシミュレータは，EDAツールベンダの製品やフリーソフトとして流通しはじめている．今後，これらを組み合わせた環境をいかに構築するかが，ハードウェア／ソフトウェア協調シミュレーション技術をSoC開発に導入していく上で重要になる．ハードウェア／ソフトウェア協調シミュレーション環境の構築には，シミュレーションモデル（機能モデルやトランザクションモデル等）について，そのモデルの詳細度，再利用性，検証容易性，各モデル間の等価性の保証，シミュレータとしての適切な実装方法など検討すべき技術的課題が残っている．今後，これらの課題が解決するとともに，開発プロセスの中における検証目的を明確にしたシミュレータの開発をしていかなければならない．

最後に，本稿の執筆にあたりご教示いただいたJEITA EDA技術専門委員会SLD研究会を始めとする関係機関の各位に感謝します．

参考文献

1）Hwang, D., Lai, B., Schaumont, P., Sakiyama, K., Fan, Y., Yang, S., Hodjat, A. and Verbauwhede, I.:Design Flow for HW/SW Acceleration Transparency in the Thumbpod Secure Embedded System, Proceedings of the 40th Conference on Design Automation, pp. 60-65（2003）.
2）Cai, L., Gajski, D., Kritzinger, P. and Olivares, M.: Top-Down System Level Design Methodology Using SpecC, VCC and SystemC, Proceedings of the Conference on Design, Automation and Test in Europe, p.1137（2002）.
3）Gerin, P., Yoo, S., Nicolescu, G. and Jerraya, A.: Scalable and Flexible Cosimulation of SoC Designs with Heterogeneous Multi-processor Target Architectures, Proceedings of the Conference on Asia South Pacific Design Automation Conference, pp.63-68（2001）.
4）Hoffman, A., Kogel, T. and Meyr, H.: A Framework for Fast Hardware-Software Co-simulation, Proceedings of the Conference on Design, Automation and Test in Europe, pp.760-765（2001）.
5）Grotker, T.， Liao, S.， Martin, G. and Swan, S.: SystemCによるシステム設計，丸善（2003）.
6）Gerstlauer, A., Doemer, R., Peng, J., Gaiski, D. and木下常雄: システム設計: SpecCによる実現，SpecC Technology Open Consortium（2002）.
7）Panda, P.: SystemC: A Modeling Platform Supporting Multiple Design Abstractions, Proceedings of the International Symposium on Systems Synthesis, pp.75-80（2001）.

（平成16年3月22日受付）