SYNOPSYS®

SystemCを用いた
複数の抽象レベルでの設計/検証を
可能にするCoCentric System Studio

2002年1月

はじめに

本書では、システムオンチップ(SoC)のハードウェア(HW)およびソフトウェア(SW)を設計する際の基本的な抽象レベルを定義し、解説した上で、各レベルで実行すべき設計と検証の側面について説明する。またSystemCがサポートする抽象レベルと共に、CoCentric System Studioを使用したSystemC設計および検証についても述べる。SystemCは、HWとSWの開発を、コンセプト･レベルからインプリメントまでの抽象段階でサポートして行くためのオープンな標準設計言語である。SystemCにより、HW設計者とSWエンジニアは、全ての抽象レベルにおいて、共通言語を使用して設計と検証を実行することができ、最終的なSWおよびHWインプリメントに近づくにつれて、より詳細な設計へとスムーズに移行することができる。特にHW設計者は、C++で設計したアーキテクチャ･モデルからRTLの詳細なHDLに書き換える必要がなくなる。このステップを排除したことにより、リスクと設計のやり直しを大幅に削減することができる。またSWエンジニアは、HWの詳細なRTLモデルの完成を待たずに、SWをSoCに組み込んで、検証を行うことができる。

現在、設計者は、これらの抽象レベルと各SoCプロジェクト独自の必要性に基づく設計フローに則って作業を進行している。したがって、本書では特定の設計フローについて詳細に述べることは避ける。特にSystemCでは、設計フローがトップ・ダウンである必要はない。SoCを構成する各部分は、それぞれの検証内容に従って、異なる抽象レベルでモデリングできる。

時間の情報を持たない機能レベル

アルゴリズムの定義、検証、および最適化は、時間の情報を持たない機能モデル(Untimed Functional Model)を使用することによって最も容易に行うことができる。このレベルでは、FIFOチャネルを介したポイント間方式でモデル同士の通信が行われる。FIFOチャネには、ブロッキング･リード/ライト･オペレーションを使用してアクセスする。このモデルを使用することにより、設計者には2つのメリットがある。モデルの開発が(通信が単純で同期が保証されるため)容易であるという点と、このレベルでは不要な詳細部分はモデリングされないため、シミュレーションが高速になるという点である。

System Studioは、業界最速のアルゴリズム・シミュレータである。このシミュレーション速度は、主要素となるスケジューリング・テクノロジとコンパイル・テクノロジを、独自のシミュレーション・エンジンで統合した結果である。このエンジンは、ダイナミック･データフローの表現力と、コンパイル式疑似スタティック･スケジューリング・シミュレータの比類なきシミュレーション速度を兼ね備えている。

制御系の機能は、この抽象レベルでは拡張有限ステート・マシン(FSM)を基本として記述される。System Studioでは、任意のネスト化されたデータフローとFSMモデルをサポートすることによって、膨大なデータ処理を必要とする機能を独自の方法で統合することができる。この結果、任意の深さの階層構造として回路を構成することができるため、理解が容易で、自然に記述できる直感的なモデリングに役立つ。

アルゴリズム設計者は、このレベルにおいてアルゴリズムの定義、検証、および最適化を行う。System Studioには、2,000を超える既存の信号処理ブロック、一般的なアルゴリズム、およびリファレンス･デザイン･キットのライブラリが用意されている。通信システムでは、実世界の伝送路を通じた状態で送受信を最適化するためのシミュレーションを行え、その特性はBERなどの値で測定できる。これにより、圧縮アルゴリズムの音声および画像の品質も最適化できる。また、有限ワード長効果と浮動小数点演算について、正確にシミュレーション可能である。

シミュレータは、パラメータ範囲から最適なソリューションを検索するためのtcl制御言語によって制御できる。データは、強力なグラフィック機能で視覚化でき、アイ・ダイアグラムやスキャッタなど、特定用途向けのフォーマットも用意されている。また、データやレジスタ変数を統計的にモニタすることで、HWまたはSW実装を実現する際に必要な最適なワード長に関する情報を収集することができる。ブレークポイントやシングルステップなどのデバッグ機能は、ブロックレベルとソースコード・レベルの両方で実行可能である。

時間情報を持つ機能レベル

遅延情報が付加された機能(アルゴリズム)モデルは、時間情報を持つ機能モデル(Timed Functional Model)と呼ばれる。このモデルの遅延は、処理(計算)遅延または通信遅延のどちらかである。この抽象レベルは、以下に対するレイテンシ効果を解析するために使用される。

- システム動作(制御系のアルゴリズムが存在する場合に多い)
- 設計プロセス早期におけるシステム・アーキテクチャ

例えば、処理遅延を付加することにより、時間情報を持つ機能モデルを使用して、システムがタイミング通りにデータを生成するかどうか、またはデッドラインを越えていないかどうかなどを判断することができる。通常は、時間情報を持たない機能レベルで使用されるポイント間通信方式が保持される。共有通信チャネル(バスなど)がこのレベルで登場することはめったにない。遅延を持たない理想的な通信インフラストラクチャを想定した動作か、または近似的に通信遅延を付加した共有データ・リンク効果をモデリングするのが一般的である。時間情報を持つ機能モデルは、初期のHW/SWトレードオフ解析で使用されることが多い。この解析は、処理タスクをHWに(ある遅延値で)マッピングした場合と、SWに(別の遅延値で)マッピングした場合の影響を評価することによって行われる。System Studioの高速シミュレーション性能により、設計者は多くの異なるHW/SW構成を試験して、設計条件に対してシステムを最適化することができる。

通常、タイミングはシステム・クロック(1つまたは複数)に対する相対的なクロック数として表現される。アルゴリズム動作から離れたシステムの機能は、この段階では完全に定義される場合とそうでない場合がある。この時間情報を持つ機能レベルにおけるモデルを使用する際には、時間情報を持たない機能レベルにおいて、まだシステムが完全に記述されているわけではないという前提条件があることを理解すべきである。設計タスクの内容により、これらのステップを実行する順序は異なり、場合によっては同時に実行される。

System Studioは、コードに遅延値をアノテートするための容易な手法を備えている。SystemCコードの処理遅延は、**wait(sc_time)**文または**next_trigger(sc_time)**文を使用してモデリングされる。通信遅延は、追加遅延ブロックを挿入することによってモデリングできる。System Studioは、4つのプリミティブなチャネル・タイプ(信号、FIFO、mutex、およびセマフォ)をサポートし、ユーザ定義のチャネルもサポートする。時間情報を持つ機能レベルでは、設計者が元のチャネルモデルを置き換えるのが普通である。例えばFIFOを、インターフェイスは同じだが、遅延が追加されているチャネルに置き換える。

System Studioは、SystemC言語標準を完全にサポートする。つまり、すべての(ユーザによって派生したチャネルを含む)SystemCチャネルがサポートされる。マクロ・デバッグ(ブロックレベル)とミクロ・デバッグ(C++ソースコード・レベル)は、両方とも完全にサポートされている。アルゴリズム・データの表示ツールは、SystemCのコンテキストでも使用できる。時間情報を持つ機能レベルにおいてデザインを探索するために特に便利な特長は、遅延付加を“パラメータ”として設定できるという点である。パラメータ化した遅延は、tclシミュレーション･コントロール･ファイルによって割り当てたり、シミュレーションの実行中にインタラクティブに変更したりできる。

バス・サイクル精度レベル(トランザクション・レベル・モデル)

SoCプラットフォーム・アーキテクチャの概念が、通信インフラストラクチャの概念を導入しているのに対し、時間情報を持つ機能レベルでは、通信スタイルがブロック間のポイント間通信としてモデリングされていた。トランザクション・レベルでは、有限数のアーキテクチャ・コンポーネント(メモリ、演算ユニット、アドレス・ジェネレータ、キャッシュなど)が、共有されるリソース(バス)を介して互いに通信を行う。複数のコンポーネントが共通バスへのアクセスを同時に要求した場合、バスは、アービトレーション方式を使用してバスの動きを制御する。最近まで、アーキテクチャのモデリングにはピンレベル・ハードウェア記述(通常はRTLコード)が必要であった。モデルの設計と検証には多大な労力が必要とされ、この詳細レベルでのシミュレーションには長い時間がかかった。正確なハードウェア構成の中でSWをデバッグするためには、HWをRTLでシミュレーションする必要があるため、早期のコード検証とアーキテクチャ・トレードオフ解析は、現実的には不可能だった。

この問題を解決するのがトランザクション・レベルのモデリングである。この抽象レベルでは、通信チャネルの詳細なインプリメント情報はモデリングされず、その動作のみがトランザクションの形でモデリングされる。トランザクション・レベル・モデル(TLM)はサイクル精度にもできるが、必ずしもそうである必要はない。TLMは、開発も利用も容易である。TLMは、RTLモデルとは比較にならないほど高速にシミュレーションできる実行可能プラットフォーム・モデルを効率よく作成でき、この結果、設計者はHW/SW統合の検証を早期に行うことができる。そのためSW開発者は、アブストラクト･プラットフォームを使用し、コードをSoCプラットフォームの構成内で試験することができる。また、HW/SWトレードオフ解析のための手法を利用し、設計期間の早期において、質の高いシステム機能検証が可能となる。HW信号とトランザクション間の変換を行うピン・レベル・アダプタを使用することで、TLMをRTLのHWモデルの試験に利用することもできる。

TLM回路図内では機能モデルも使用できる。膨大なライブラリからアルゴリズム・モデルをSystemCアーキテクチャ・モデルにドラッグ&ドロップして再利用できるため、さまざまなスティミュラスを短時間で生成することができる。(後処理または解析機能の追加、まだ実装されていない新しいブロックのプレイスホルダの追加、SoC外の環境のモデル化など)。タイミング情報を持つ機能モデルでは、ユーザが新しいブロックの詳細を定義するために貴重な時間を費やす前に、システムレベルの性能を確認するための有効なプレイスホルダを作成することができる。

すべての設計チーム(HW/SW/システム/アーキテクチャ)が互いに連結されるのは、TLMレベルである。TLM解析により、バス利用/負荷の最適化、キャッシュ方式効率、HWとSW間の相互動作の確認といった重要な設計要因を検討できるようになる。統合に関する問題をTLMレベルで発見して解決することは、RTLあるいは初期シリコン後に行う場合と比較して、コストがはるかに低く済む。

System Studioは、SystemCのすべてのTLM機能をサポートしている。さらに、モデル作成ウィザードと各種回路図対応エディタを備えているため、新しいSystemCアーキテクチャ・コンポーネントを容易に作成することができる。TLM設計は、階層的ブロック図でも見ることができる。このレベルでデバッグを行う場合は、設計者はさまざまなSystem Studioのデバッグおよび表示機能を利用できる。SystemCモジュールでは、3種類の“モニタ”が定義されていて、重要な情報をSystem Studioで表示することができる。“メッセージ・モニタ”は、バス動作を追跡して表示する。“テーブル・モニタ”は、メモリの状態を調べて表示したり、System Studioでグラフとして表示するほか、メモリの内容をスプレッドシートに出力することができる。“データ・モニタ”は、ユーザがSystemCコードで定義した状態のリストから、現在の状態を表示する。このツールは、現在の階層レベルにおけるすべての信号、パラメータ、モデル、およびモジュールの現在値のリストを表示する“レベル・ウォッチ”機能を備えている。“データ・ウォッチ”機能は、回路内の任意の場所の任意のデータを表示するように設定できるため、モニタすべき重要データのデバッグ・パネルが構築できる。アーキテクチャ設計のマクロ・デバッグを行うには、ブレークポイントをセットおよびクリアする。System StudioをSystemCと併用することにより、アーキテクチャ・モデルの作成とSoC環境のモデリングの両方をスピードアップでき、早期のHW/SW検証が可能になる。

ピン精度レベル(ビヘイビア・ハードウェア・レベル)

HWモジュールをインプリメントする際の重要なステップは、選択したピンレベル通信・プロトコルに準拠して読み書きされるピンレベル・インターフェイス(クロック、リセット、データ、および制御ラインから構成される)を設計することである。ビヘイビアHWモデルは、このピンレベル・インターフェイスを活用できるようにリファインされた機能モデルとして考えることができる。内部では、最終的なRTLインプリメント構造をまだ示していない。

検証(スティミュラスの生成、モニタ、データの後処理)の目的であれば、このようなモデルで十分である。これらのモデルは、論理合成に向けた踏み石に過ぎない場合も多い。このモデルを完全に合成可能なブロックに変換するには、回路を合成コーディング・ルール群と比較する。シノプシスはこれらのコーディング･ルールも提供している。合成可能コードは、RTLスタイルまたはビヘイビア・スタイルのSystemCで記述できる。正確なレジスタ・トランスファおよびクロッキングの詳細をコーディングする場合には、RTLスタイルが適している。アルゴリズム表現をコーディングし、求められる機能を実現するためのソリューションをコンパイラが提示できるようにする場合には、ビヘイビア・スタイルが適している。シノプシスのSystemCからのハードウェア合成ツールCoCentric SystemC Compilerは、コーディング・スタイルに関係なく、回路をFPGAまたはASICターゲット・ネットリストとして合成できる。

ゼロから、あるいは時間情報を持つ機能モデルを使用してビヘイビア・モデルを作成する場合にも、System Studioの機能を利用することにより、設計者の生産性は非常に高くなる。密接に統合されたインターフェイス・エディタにより、モデルのインターフェイスを短時間で変更できる。また、モデル・ウィザードを使用することにより、他のプロセスやレジストリを容易に作成できる。

ここまでで述べたデバッグ機能とデータ表示機能に加えて、System Studioは、グラフィカル・デバッガ VirSimによりHW設計者が理解しやすい表示ができる。VirSimは、シノプシスのVerilog-HDLシミュレータ VCSおよびVHDLシミュレータSciroccoにも搭載されている波形ビューワおよびシミュレーション･コントロール･ツールである。VirSimは、シミュレーション終了時にVCDデータ・ファイルの内容を表示するだけでなく、シミュレーションを制御して、SystemCの信号とパラメータをインタラクティブに表示することができる。また、オンデマンド・データ・トレース機能により、最高のシミュレーション速度が保証される。

レジスタ・トランスファ・レベル

RTLでは、ハードウェア・レジスタ、クロック、ラッチ、および組み合わせロジックによってデジタル・ハードウェアが正確に記述される。RTLでは、論理合成で必要な詳細レベルでシリコン・コンポーネントの内部構造を指定することができる。RTLで論理合成用のSystemCコードを記述するための方法を解説した文書は、http://www.synopsys.com/sldからダウンロードできる。

System Studioで最も低い抽象レベルはRTLである。RTLは、Verilog-HDLコード、VHDLコード、または(ビヘイビア・スタイルではなく)RTLコーディング・スタイルで書かれたSystemCコードになる。ただし、System Studioは、VHDLやVerilog-HDLのシミュレータではない。RTLでのHDLコードの検証には、VCSおよびSciroccoが使用される。これらのツールは、System Studioと組合わせて使用できる。そのため、回路の一部がSystemCで、他の部分がVHDLやVerilog-HDLで記述されている言語混在デザインの設計のデバッグ、および検証を短時間で行うことができる。設計者は、既存のRTLのHDLコンポーネントを使用たり、シノプシスのIP/IPモデル･ライブラリ DesignWareのコンポーネント群を活用できる。SystemCとVHDL/Verilog-HDLとの間のシミュレーション・インターフェイスは、System StudioをHDLシミュレータと併用したときに自動的に生成される。

非常に複雑なSoCでは、RTLにおけるシミュレーション時間が途方もなく長くなることがある。System Studioは、この点においても完璧に対応する。System Studioでは、一部のコンポーネントがRTLで、他のコンポーネントがそれより高い抽象レベルである回路の場合でもシミュレーション可能であるため、シミュレーション速度を大いに向上させることができる。設計者は、特定の要素のみをRTLで開発しつつ、そのRTLブロックを含んだシステム全体をシミュレーションすることができる。System Studioは、高いシミュレーション性能の提供と、テスト環境の作成と管理に要する労力を激減させることにより、生産性を向上させる。特にテスト環境の作成は、HW検証において最も高コストにつくことが多い。

ソフトウェアの検証

最近のSoCでは、明らかに、組込みCPUやDSPコアで動作するSWの開発に多くの設計労力が注がれている。System Studioを使用することにより、設計者はアブストラクト･レベルでSWをモデリングし、SoC上で動作する実際のSWをシミュレーションするためのSoCアーキテクチャ・モデルを提供することができる。

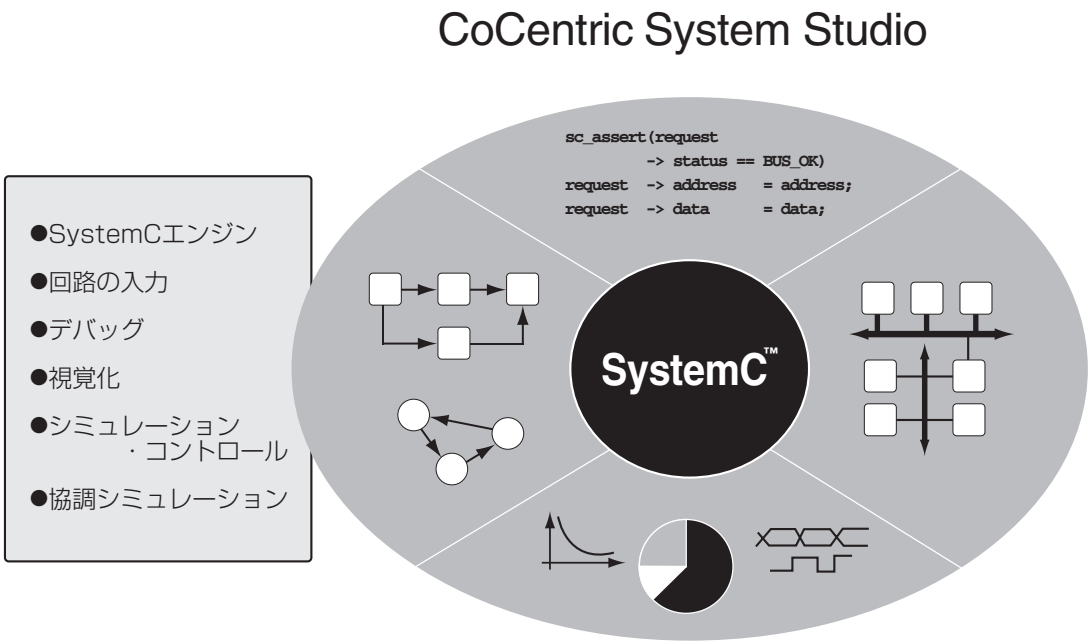
SWエンジニアは、通常、CまたはC++をソースコード言語として、プロセッサ・ベンダから提供された統合設計環境上でプロセッサ・コア用のコードを作成する。これらのCまたはC++コードはSystemCのサブセットであるため、SystemCブロックとして、容易にSystem Studioに取り込むことができる。時間情報を持つ機能レベルでは、ベンダ固有のツールを使用して、正確な処理遅延を測定し、SystemCモデルに手作業で付加することができる。

System Studioでは、プロセッサのトランザクション・レベルまたはピンレベル・モデルと、ベンダのツール・スイートで開発されたコードを使用して、対象となるプロセッサ上でのSWの実行をシミュレートできる。このため、SWコードを特定のSoC環境でデバッグすることが可能になる。対象となるアプリケーションで、リアルタイム動作条件下での複数のソフトウェア・タスクの管理が必要な場合には、リアルタイム・オペレーティング・システム(RTOS)を使用する。System Studioでは、ターゲット・プロセッサ・アプリケーション・コードと周囲のSoCハードウェアと共に、フル・システム構成の中でRTOSをシミュレーションできる。抽象化することで、RTOSのビヘイビアを(トランザクション・レベルなどで)さらに解析または最適化することができる。特定のハードウェア信号のSW制御が重要である場合は、System Studioでピンレベル・モデルを使用して、ロー･レベルHW/SW相互動作をシミュレーションできる。

System Studioを用いることの主要なメリットは、SoC固有のSWを設計プロセスの非常に早い段階でシミュレーションできるという点である。この段階では、SoCを変更する際のコストも低く、変更に要する時間も短くて済む。

ハードウェア合成へのパス

シノプシスは、シミュレーション用にCoCentric System Studioを提供しているほか、SystemCからのハードウェア合成用にCoCentric SystemC Compilerを提供している。System Studioは、SystemCを使用したコンセプトからインプリメントまでの設計フローを完全に実現する。欠けている部分は、合成可能なSystemCコードから実際にASICやFPGAネットリストを作成するというステップのみである。CoCentric SystemC Compilerは、合成可能なコードをASICまたはFPGAネットリストに合成する。また、シノプシスの合成ツールであるFPGA Compiler II、Design Compiler、およびPhysical Compilerと緊密に連携しているため、SystemCを用いたRTLスタイルでのHW記述は、従来のVHDL/Verilog-HDLから開始した場合と同等の結果品質(QOR)を得ることができる。



まとめ

本書では、SystemCがサポートする異なる抽象レベルと、これらのレベルをSoC検証において使用するメリットについて説明した。シミュレーションおよび合成用ツールの使用により、これらのメリットを最大限に活かした設計が可能になる。

CoCentric System Studioは、任意のSystemCモデルを任意の抽象レベルでシミュレーションでき、また、強力なグラフィカル設計環境により、これらのモデルの作成と管理を容易にする。さらに、強力なSystemCメソドロジを既存のHDL設計ツールと連結することにより、SystemCを実際の設計および検証に使用するために必要なすべてのシミュレーション機能を提供する。

そして、CoCentric SystemC Compilerを用いてSystemCからのHW合成を実現することにより、システムレベル設計とゲートレベル設計との間のギャップを埋めることが可能となる。

お問い合せ先：

日本シノプシス株式会社

〒163-0420　東京都新宿区西新宿2-1-1　新宿三井ビルディング20F　TEL.03-3346-7030(代) FAX.03-3346-7050

〒531-0072　大阪府大阪市北区豊崎3-19-3　ピアスタワー13F　TEL.06-6359-8139(代) FAX.06-6359-8149