社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－ＳＬＤＭ－115 （５）
2004／5／27

[チュートリアル講演]

ハードウェア／ソフトウェア・コデザインの技術動向

冨山 宏之

名古屋大学 大学院情報科学研究科 〒464-8603 名古屋市千種区不老町

E-mail: tomiyama@is.nagoya-u.ac.jp

あらまし 本稿ではハードウェア／ソフトウェア・コデザイン技術の動向について解説する。まず、コデザインの背景と歴史を説明する。次に、コデザインの全体的な流れを説明し、システム仕様記述、機能分割、スケジューリング、通信合成、および、コシミュレーションなどの個々の技術について解説する。最後に、プロセッサのコデザイン技術についても動向を説明する。

キーワード コデザイン、組込みシステム、システム仕様記述、機能分割、コシミュレーション、ASIP

Technology Trends of Hardware/Software Codesign

Hiroyuki TOMIYAMA

Graduate School of Information Science, Nagoya University, Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan

E-mail: tomiyama@is.nagoya-u.ac.jp

Abstract This paper presents technology trends of hardware/software. First, the backgrounds and history of codesign are briefly described. Then, a codesign methodology is presented which covers system specification, partitioning, scheduling, communication synthesis, and cosimulation. Finally, codesign techniques for application specific processors are presented..

Keyword Codesign, Embedded Systems, System Specification, functional partitioning, cosimulation, ASIP

1. はじめに

ハードウェアとソフトウェアのトレードオフを考慮しながら両者の設計を同時に協調させて行うシステム設計を、**ハードウェア／ソフトウェア協調設計**、または、**ハードウェア／ソフトウェア・コデザイン**(以下、単に**コデザイン**)と言う。コデザインは**組込みシステム**の設計を上で極めて重要である。組込みシステムとは、様々な電気電子機器に組み込まれ、その制御や情報処理を行う電子計算機システムのことであり、例えば、ディジタル TV や DVD 機器などの情報家電製品、炊飯器や電子レンジなどの家電製品、プリンタやコピー機などのオフィス機器、更には、自動車のエンジン制御システムやエアバッグ・システムなども組込みシステムの一種である。組込みシステムの最大の特徴は、アプリケーション(用途)が特定されていることである。組込みシステムを設計する場合、アプリケーションの特徴を十分に解析し、そのアプリケーションを効率的に実行するようにシステムの構成を最適化することが重要である。ここで言う効率とは、コスト、性能、消費電力・エネルギーなどのことである。当然ながら、実現すべき処理内容(機能)だけでなく、要求されるコスト、性能、消費電力・エネルギーなどもアプリケーションにより異なる。例えば、携帯機器を設計する場合には消費エネルギーを最小化することが重視され、大量生産される製品を設計する場合には製造コストを最小化することが重視されるであろう。また、アプリケーションによっては、信頼性、安全性、拡張性なども重視されるかも知れない。組込みシステムに求められる機能は年々大規模かつ複雑になっており、それと同時にコスト、性能、消費電力・エネルギーなどに対する要求も厳しくなっている。その一方で、設計期間(Time-to-Market)の短期化も強く求められている。つまり設計者は、大規模、複雑、かつ、多種多様な組込みシステムを、短期間で、かつ、最適に設計することを求められている。

組込みシステムを設計することは、ハードウェアとソフトウェアの両者を設計することである。しかし当然ながら、ハードウェア設計とソフトウェア設計を独立に行ったのでは最適な組込みシステムを設計することはできないし、また、設計したハードウェアとソフトウェアとの間で不整合(不具合)が生じやすい。大規模、複雑、かつ、多種多様な組込みシステムを、短期間で、かつ、最適に設計するためには、ハードウェアとソフトウェアを同時に協調させて設計すること、すなわちコデザインが不可欠である。

本稿では、コデザイン技術について幅広く解説を行う。まず1章でコデザインという言葉や概念が生まれた背景を説明する。2章で典型的なコデザインのメソドロジと要素技術について、4章ではプロセッサのコデザイン技術について説明する。最後に5章でまとめを述べる。

2. コデザインの背景と歴史

ハードウェアとソフトウェアを同時に協調させて設計するというコデザインの考え方自体は古くから―マイクロプロセッサが発明された 1970 年代から―存在していた。しかし、コデザインという言葉が使われるようになったのは比較的最近の 1990 年代初期のことである。最近になってコデザインという言葉や概念が広く普及した背景として、以下の事項が挙げられる。

第一の背景として、1980 年代までと比較して、安価かつ短期間にハードウェアを設計できるようになった

ことが挙げられる。これは、半導体デバイス技術、製造技術、および、設計自動化技術が大きく向上したことによる。特に、カスタム IC だけでなく、FPGA や PLD などのプログラマブルなデバイスが普及したことや、VHDL や Verilog-HDL などのハードウェア記述言語(HDL: Hardware Description Language)と論理合成ツールが広く普及し、ハードウェアの設計生産性が大幅に向上したことが大きく寄与している。これらのことから、個々のアプリケーションに応じてハードウェアを設計することが容易になってきた。近年では、アプリケーションに応じて専用回路を設計するだけでなく、プロセッサ自体も最適化することも可能となった。

第二に、システム設計におけるソフトウェアの比重が年々大きくなっていることが挙げられる。ハードウェアの設計が容易になってきているとはいえ、同じ機能を実現するのであればハードウェアよりもソフトウェアを設計する方が依然として容易である。組込みシステムに要求される機能は年々大規模化、複雑化しており、複雑なシステムを短期間で設計するためには、多くの機能をソフトウェアで実現した方が良い。しかしながら、プロセッサの性能は年々向上しているものの、すべての機能をソフトウェアで実現すると要求性能を満たせないことが多く、また近年では性能よりも消費電力・エネルギーの要求を満たすためにハードウェアを設計する場合が増えている。複雑化する機能の中から、どの部分をハードウェアとして実現するかという判断が非常に重要になってきている。

第三に、半導体デバイス技術や製造技術の進歩により、従来は別々の LSI チップとして実現され、プリント基板上に搭載されていたプロセッサ、メモリ、専用回路などを、現在では 1 つの LSI チップ上に集積することが可能となったことである。このように高度に集積された LSI は**システム LSI** と呼ばれる。SOC 技術より、ハードウェア設計とソフトウェア設計の境界が柔軟になり、設計の自由度が大きく拡大した。特に、ピンの制約が無くなったことにより、プロセッサ、メモリ、専用回路などのコンポーネントをチップ上の任意の位置に配置できるようになり、コンポーネント間の接続形態の自由度も向上した。

また、前章で述べたように、組込みシステムのアプリケーションは複雑化、大規模化、そして、多種多様化している一方、コスト、性能、消費電力・エネルギーなどに対する要求は厳しくなり、更に、設計期間の短縮も求められている。

現在、ハードウェアの設計記述は、レジスタ転送レベル(RTL: Register-Transfer Level)と呼ばれる抽象度で行われることが多い。RTL とは、回路のクロック・サイクル単位の動作を記述する設計の抽象度、あるいは、レジスタ(レジスタファイル)、メモリ、及び、加算器などの演算器を最小コンポーネントと考え、これらの接続関係を記述する設計の抽象度のことである。現在 RTL 記述を行う際には、VHDL や Verilog-HDL などの HDL が使用されている。RTL で記述すれば、それより下流の設計工程、例えば、論理合成や配置配線などは、ほぼ自動化されている。しかし、RTL よりも上流の設計工程、例えば、ハードウェアとソフトウェアへの機能分割、プロセッサやメモリの選択、プロセッサやメモリや専用回路間の接続形態の決定などは、設計者の勘と経験に任されているのが現状である。しかし、もはや勘と経験だけでは、複雑で大規模な組込みシステムを短期間で最適に設計することは不可能である。

以上の背景などにより、1990 年代に入った頃からコデザインの重要性が叫ばれ始め、学会において様々な研究が行われ、産業界においても EDA ツールが開発されてきた。コデザインの研究の目的は、従来設計者の勘と経験により行われてきたシステム設計を、理論的な裏付けに支えられた設計メソドロジ(方法論)として体系化することにある。コデザインはハードウェア設計とソフトウェア設計の両方を包含するため、コデザインの研究は多岐に渡っている。例えば、記述言語、モデリング、性能やコストや消費電力などの見積り、ハードウェア／ソフトウェア分割、スケジューリング、インタフェース合成、ハードウェア／ソフトウェア・コシミュレーション、アーキテクチャなどの研究が行われ、更には、コデザインの観点から RTOS やコンパイラの研究なども行われてきた。コデザインに関する現在までの研究のサーベイは文献[1]や[2]に詳しい。これらの他、コデザイン技術については様々な学術雑誌に特集記事が組まれており[2]-[9]、また、書籍も数多く出版されている[10]-[18]。

3. コデザインのメソドロジと要素技術

本章では、典型的なコデザインのメソドロジと要素技術を紹介する。しかし、ここで紹介するメソドロジは決して万能ではない。何故なら、アプリケーション分野により有効なメソドロジは異なるし、また、アプリケーション分野だけでなく、設計環境や設計者自身—例えば、設計グループの体制、設計者のスキル、過去の設計資産、有する EDA ツールなど—にも大きく依存するためである。設計者（設計の責任者）は、設計すべきアプリケーションや設計環境に応じて、ここで紹介するメソドロジをチューニングする必要がある。

3.1. コデザインの流れ

図 1 に典型的なコデザイン・フローを示す。まず組込みシステムの設計者は、設計対象であるアプリケーションの**仕様**を定義し、記述する。ここで言う仕様とは、主としてアプリケーションの機能のことである。また、性能、コスト、消費電力・エネルギーなどに関する設計制約を指定する。この段階で、シミュレーションにより機能の検証を十分に行う必要がある。次に、機能のうち、どの部分をハードウェア(専用回路)で実行し、どの部分をソフトウェア(プロセッサ)で実行するかを決定する。この作業は**ハードウェア／ソフトウェア分割**または**機能分割**と呼ばれる。機能分割の際には、各部分処理の**スケジューリング**も考慮しなければならない。次に、ハードウェアとソフトウェアとの間の通信の設計を行う。この作業は**通信合成**と呼ばれる。この段階で、ハードウェアとソフトウェアに分割された機能が通信しながら正しく動作することを検証する。ハードウェアとソフトウェアを協調させて同時にシミュレーションを行うことを、**ハードウェア／ソフトウェア協調シミュレーション**、または、**ハードウェア／ソフトウェア・コシミュレーション**(以下、単に**コシミュレーション**)と呼ぶ。次に、ハードウェア部分は**動作**

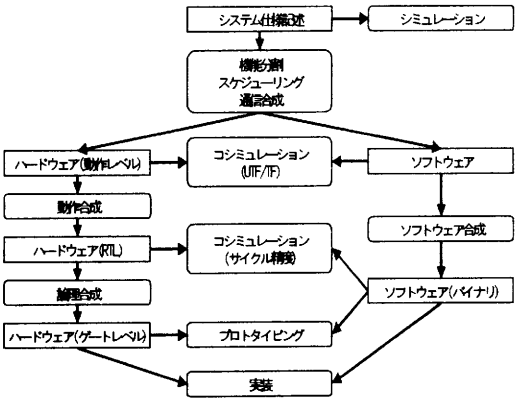

図 1 コデザイン・フロー

合成と呼ばれる作業より RTL 記述へと変換され、ソフトウェア部分は**ソフトウェア合成**と呼ばれる作業によりバイナリコードへと変換される。これらの作業により、ハードウェアもソフトウェアもクロック・サイクル単位で動作が正確な、いわゆる**サイクル精度(CA: Cycle Accurate)**のタイミングを持つ。この段階でコシミュレーションを行い、サイクル精度でハードウェアとソフトウェアが通信しながら正しく動作することを確認する。また、アプリケーションの実行時間をサイクル精度で正確に評価することができる。シミュレーションにより十分に検証した後、FPGA ボードやエミュレーション装置などを用いてシステムの**プロトタイピング**を行う。このプロトタイプ・システム上にハードウェアの設計データとソフトウェアのバイナリコードをダウンロードし、実行することにより、動作確認を行う。その後、ハードウェア(LSI)を製造し、最終的にシステムを**実装**する。

以下、各々の作業についてより詳細に説明する。

3.2. システム仕様記述

設計者はまずシステムの仕様(機能)を記述する。この段階では、ある機能をハードウェアとソフトウェアのどちらで実現するかが定まっていないので、ハードウェアとソフトウェアの両者を区別せず、一体的に記述する必要がある。システムの仕様を記述することを目的として、既存の HDL をベースにした言語、ソフトウェアのプログラミング言語をベースにした言語、全く新しい言語など、現在までに多くの言語が提案されてきた。その中で、現在最も有望視されているのが SystemC [19][20]という言語である。文法的には、SystemC は C++言語のクラスライブラリによる拡張である。SystemC は 1990 年代後半に開発が開始され、2000 年に 1.0 版が公開された比較的新しい言語である。Open SystemC Initiative (OSCI) [19]という団体を中心に標準化活動が行われ、SystemC 用のシミュレータは OSCI の WWW サイトより無償で入手できる。SystemC 1.0 ではハードウェアを RTL またはサイクル精度で記述することが主たる目的であったが、2001 年に公開された SystemC 2.0 で大幅な拡張が行われ、ハードウェアだけでなくソフトウェアも記述できるようになった。SystemC の他、システム仕様記述を目的として開発された言語に SpecC 言語がある。SpecC 言語は元々カリフォルニア大学アーバイン校で開発され、現在は SpecC Technology Open Consortium (STOC)という団体で標準化活動が行われている[21][22]。SpecC のシミュレータも STOC の WWW サイト[21]で無償公開されている。文法的には、SpecC 言語は ANSI-C 言語をベースにしており、並行性、パイプライン処理、割込み、ハードウェア構造、入出力ポート、信号、任意長のビットベクタなど、多くの概念が明示的に新しく導入されている。SystemC が 2.0 版に進化した際、SystemC は SpecC が有していた多くの概念を導入したため、表面的な文法は異なるものの、両者の本質的な違いは小さくなった。

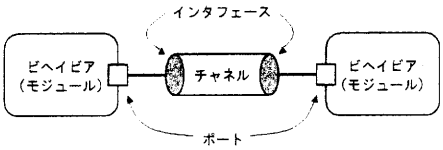

図 2 トランザクションレベル・モデリングの例

仕様記述を行う際、特に、以下の事項に留意しなければならない。

- 特定のハードウェア・アーキテクチャに特化した記述を行わないこと。これは、仕様記述の段階では、システムのハードウェア構成が決定されていないためである。
- 機能のうち、計算を行う部分と、通信を行う部分は分離して記述すること。もし通信と計算が混在して記述されていると、ハードウェア／ソフトウェア間のインタフェース設計が困難となる。
- 粗粒度(関数レベルやタスクレベル)の並列性を明示的に記述すること。これは、現在の合成ツールやコンパイラは細粒度(演算レベル)の並列性は自動的に抽出できるが、粗粒度の並列性を抽出する能力は低いためである。

SpecC 言語や SystemC は、計算と通信を分離して記述するため、ビヘイビア(SystemC ではモジュール)、ポート、インタフェース、および、チャネルというオブジェクトを有している。図 2 にこれらのオブジェクトの関係を示す。計算処理はビヘイビア内に記述し、通信に関わる処理はチャネル内に隠蔽する。モジュールとチャネルは階層的に記述することができ、ビヘイビアの中にビヘイビアやチャネルを、チャネルの中にもビヘイビアやチャネルを定義することができる。このように通信と計算を分離してモデル化することを、**トランザクションレベル・モデリング(TLM: Transaction-Level Modeling)**と言う。TLM については、文献[20][22][23]などにまとめられている。

この段階で、シミュレーションにより機能検証を十分に行うと同時に、プロファイリングや静的解析を行い、各ビヘイビアの計算量や実行回数、更に、通信量などの測定や見積りを行う。

3.3. 機能分割とスケジューリング

仕様記述の段階では、各ビヘイビアがハードウェア(専用回路)とソフトウェアのどちらで実現されるかは

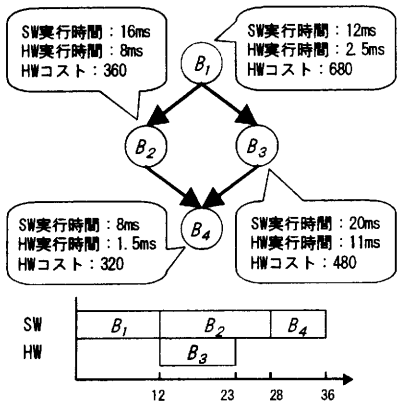

図 3　機能分割とスケジューリングの例

決定されていなかった。この決定を行う作業を機能分割と呼ぶ。また、ビヘイビアの実行順序を決定する作業をスケジューリングと呼ぶ。機能分割とスケジューリングは互いに大きく依存するため、同時に行われなければならない。

図 3 に単純な機能分割の例を示す。ノードがビヘイビアを現し、エッジがビヘイビア間の依存関係を表している。この例では、B_1 の実行が終了した後に、B_2 と B_3 の実行が開始されなければならないことを表している。また、各ビヘイビアをソフトウェアとハードウェアで実行した場合の実行時間と、ハードウェアで実現した場合のコストを記している。この例を用いて、実行時間が 36ms 以下という制約の下で、コストを最小化する機能分割を考えよう。ビヘイビアが 4 つ存在し、それぞれハードウェアとソフトウェアという選択肢が存在するので、可能な機能分割は 2^4 通り (16 通り) 存在する。機能分割を決定する際は、ビヘイビア間の並行性を考慮し、スケジューリングも同時に行う必要がある。例えば図 3 の場合、ビヘイビア B_2 と B_3 の間には依存関係が無いため、もしこれらを異なるコンポーネントに割り当てれば、並列に実行することができる。このことを考慮すると、最適な機能分割とスケジュールは図 3 の下図に示す通りになる。

図 3 の例題は非常に単純化されており、現実とはかけ離れている。例えば、プロセッサは 1 個だけであるし、ビヘイビア間の通信に要する時間やコストを全く無視している。処理のパイプライン化を行っていない、消費電力・エネルギーを考慮していない、メモリの構成やコストを考慮していないなどの問題もある。また、実際にはビヘイビアの数はもっと多く、周期的に実行されるものと、そうでないものが混在しているだろう。更には、各ビヘイビアをハードウェアとソフトウェアで実行した場合の実行時間と、ハードウェアで実現した場合のコストが既知であることを想定しているが、実際には、詳細設計を行う前にこれらの値を正確に見積もることは困難である。

現在までに学界において、機能分割、スケジューリング、及び、性能やコストなどの見積りの手法やアルゴリズムが数多く提案されてきた。しかし、これらのほとんどは対象とするシステム構成に何らかの仮定を設けており、実設計にそのまま適用できることは稀であろう。過去の研究を理解し、実設計に応用できる能力を身に付けることが重要である。

3.4. 通信合成

機能合成とスケジューリングが終わった段階では、ハードウェアとソフトウェアの間の通信は抽象的なチャネルにより行われている。より具体的に言うと、チャネルを表すオブジェクトのメソッドを呼び出すことにより通信が行われている。通信合成において、仮想的な通信チャネルはバスに割り当てられ、バスインタフェース回路や、必要に応じて調停回路が合成される。また、ソフトウェア側は、デバイスドライバが合成される。通信合成については文献[22]や[23]に詳しい。現在、通信合成を部分的に支援する EDA ツールは存在するが、まだ広く普及するには至っていない。

3.5. 動作合成と論理合成

機能分割により、ハードウェアで実現すべきビヘイビアは決定されるが、ハードウェアの内部構造はまだ決定されていない。これを決定するのが動作合成と論理合成である。まずハードウェアで処理すべきビヘイビアを動作合成により RTL 記述に変換し、更に論理合成を行うことによりゲートレベルのネットリストに変換される。その後、配置配線が行われ、製造データが生成される。現在論理合成ツールは広く普及している。一方、動作合成は 1980 年代から 1990 年代にかけて活発に研究が行われ、基本的な技術は成熟したと考えられているが、本格的なツールの普及はこれからである。動作合成の具体的な処理内容については文献[24]-[26]に詳しい。

3.6. ソフトウェア合成

ソフトウェア合成とは、ソフトウェアとして実現されるビヘイビアを、ターゲット・プロセッサのバイナリコードに変換する作業である。ソフトウェア合成以前のビヘイビアは SystemC や SpecC 言語などの言語で記述されているが、現在、これらの言語から直接ターゲット・プロセッサのバイナリコードに変換する処理系は存在しない。そこで、まずこれらの記述を C 言語によるプログラムに変換し、その後、ターゲット・プロセッサ用のクロスコンパイラを用いてバイナリコードに変換する。SystemC や SpecC による記述には、ビヘイビア間の通信や割込み処理を行うコードが含まれているだろう。C プログラムに変換する際には、これらを RTOS のサービスコールへ置き換える必要がある。また、先に決定した機能分割とスケジューリングに従い、ビヘイビアからタスクを構築し、タスクに対して優先度を与える。現在ソフトウェア合成は人手で行う必要があるが、学界では研究が進められている。例えば、文献[27]では SpecC 記述を ITRON 上で動作する C プログラムに変換する手法が提案され、ツールが試作されている。

3.7. コシミュレーション

ここまで説明してきたように、最初にシステム仕様記述を行った後、設計者は様々な作業を行い、それに伴って設計記述を変更する必要がある。記述を変更する度に、シミュレーションにより動作の正しさを検証

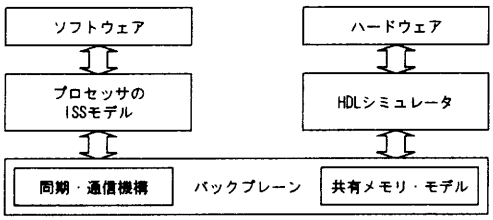

図 4 サイクル精度のコシミュレーション

する必要がある。ハードウェア記述とソフトウェア記述を同時に協調させて行うシミュレーションを、ハードウェア／ソフトウェア・コシミュレーションと呼ぶ。図 1 では、仕様記述を行った後、通信合成を行った後、そして、動作合成とソフトウェア合成を行った後の 3 回、シミュレーションが行われているが、当然ながら実設計では 3 回だけでなく、何度も繰り返し行われる。

動作合成が行われる前までは、ハードウェアもソフトウェアも同一の言語、例えば、SystemC や SpecC で記述されているため、その言語用のシミュレータを用いて実行することができる。しかし、動作合成以降のハードウェアは HDL で記述され、ソフトウェアも合成されてターゲット・プロセッサのバイナリコードに変換されている。この両者のシミュレーションを協調して行うため、図 4 に示すようなシミュレータ(コシミュレータ)が使用される。コシミュレータの内部では、ソフトウェアはターゲット・プロセッサの**命令セット・シミュレータ(ISS: Instruction Set Simulator)**上で実行され、ハードウェアは HDL シミュレータ上で実行される。コシミュレータのバックプレーンが両者の同期と通信を制御する。多くのコシミュレータはバックプレーン内部に共有メモリのモデルを有しており、ソフトウェアからもハードウェアからもアクセスできるようになっている。

現在、多くの EDA ベンダがコシミュレータを販売しており、実際の製品開発にも広く使用されている。

4. プロセッサのコデザイン

前章で紹介したメソドロジでは、アプリケーションに応じてプロセッサの命令セット・アーキテクチャ(ISA: Instruction-Set Architecture)の最適化設計を行うことは想定していなかった。ここで ISA とは命令セット、命令長、レジスタ数、命令レベル並列度、データパス幅などのことを言う。プロセッサの ISA は、プロセッサの性能、コスト、消費電力・エネルギーを決定するだけでなく、メモリ、バス、および、周辺回路の設計にも大きな影響を与える。よって、プロセッサの ISA の最適化を行うことは、組込みシステム設計を行う上で非常に効果的である。アプリケーションに応じて ISA が最適化されたプロセッサは **ASIP (Application Specific Instruction-set Processor)**と呼ばれる。

プロセッサの ISA の新規設計あるいは設計変更を行う際には、単なるハードウェアの回路設計だけに留まらず、コンパイラ、アセンブラ、オペレーティング・システムなどの基本ソフトウェアも新規設計／設計変更する必要がある。また、プロセッサのプロトタイプが完成する前にソフトウェアの動作検証を行うために、そのプロセッサの ISS も必要である。これらのソフトウェア開発環境を開発することは、組込みシステムの設計を行う上で非常に大きな負担である。現在までに世界中でプロセッサのコデザイン技術に関する研究が数多く行われてきたが、コンパイラや ISS の自動生成あるいはリターゲット技術や、プロセッサとコンパイラの協調設計技術は重要な研究テーマであった。

プロセッサのコデザインに関する研究は日本でも活発に行われてきた。例えば、日本で開発されたプロセッサのコデザイン・システムとして、PEAS-I [28]、PEAS-III (ASIP-Meister) [29]、Satsuki [30]、ソフトコア・プロセッサ／Valen-C コンパイラ・システム[31]などがある。プロセッサのコデザイン手法は、コンフィギュラブル・プロセッサ方式(テンプレート方式)と、プロセッサ記述方式の 2 つに大別することができ、それぞれ一長一短がある。

コンフィギュラブル・プロセッサ方式とは、アプリケーション分野毎にベースとなるプロセッサをあらかじめ用意しておき、個別のアプリケーションに応じて ISA のコンフィギュレーションを行う方法である。このような変更可能なプロセッサのことを**コンフィギュラブル・プロセッサ**あるいはソフトコア・プロセッサと呼ぶ。コンフィギュラブル・プロセッサは基本構成があらかじめ定められているため、設計が容易である、コンパイラや ISS などの開発環境のリターゲットが容易である、プロセッサの動作周波数、回路面積、消費電力などの見積りが容易であるなどの利点がある。一方、設計最適化の自由度が小さいという欠点がある。

コンフィギュラブル・プロセッサ方式は、更に、コンフィギュレーションを自動で行う方式と、手動で行う方式の 2 種類に分類される。PEAS-I システム[28]は自動コンフィギュレーション方式である。アプリケーション・プログラムと入力データを与えると、命令セットが自動的に決定され、合成可能な HDL 記述やコンパイラなどが自動生成される。ベースとなるプロセッサは 4 段のシングルパイプライン方式であり、追加可能な命令の種類もあらかじめ定められている。PEAS-I 以外にも自動コンフィギュレーション方式の研究が日本国内外で行われてきたが、筆者の知る限り、産業界で実用可能なツールはまだ存在しない。

一方、手動コンフィギュレーションが可能なプロセッサは、近年、産業界において実際の製品開発にしばしば使用されている。手動コンフィギュラブル・プロセッサとして、Tensilica 社の Xtensa プロセッサ、ARC 社の ARC プロセッサ、東芝の MeP プロセッサなどが挙げられる。例えば、Xtensa は既定の命令セットを持つ RISC プロセッサであるが、TIE と呼ばれる言語を用いて命令を追加することができる。この記述から、合成可能なプロセッサの HDL 記述とコンパイラが生成される。先述の Satsuki システム[30]も手動コンフィギュレーション方式に分類される。九州大学で開発されたソフトコア・プロセッサ[31]も手動コンフィギュレーション方式であり、命令セットやレジスタ数だけでなくプロセッサのデータパス幅も変更可能である。また、それに対応したリターゲッタブル・コンパイラも用意されている。

プロセッサ記述方式は、専用の言語を用いてプロセ

ッサの ISA (各命令の動作やデータパス構造など)を記述し、その記述からプロセッサを合成する方式である。また、理想的には、同一記述からコンパイラや ISS などのソフトウェア開発環境も自動生成される。これらの目的に用いられる言語は総称して**アーキテクチャ記述言語(ADL: Architecture Description Language)**と呼ばれる。プロセッサ記述方式は、コンフィギュラブル・プロセッサと比較して設計最適化の自由度が大きいが、プロセッサの設計や、コンパイラや ISS などの開発に要する工数も増える。大阪大学で開発された PEAS-III (ASIP Meister)システム[29]はプロセッサ記述方式に分類され、プロセッサの ADL 記述から合成可能な HDL 記述とコンパイラが生成される。PEAS-III 以外にも、アーヘン大学で開発され LISATek 社[1]により事業化された LISA、ドルトムント大学の MIMORA、カリフォルニア大学アーバイン校の EXPRESSION、マサチューセッツ工科大学の ISDL など、様々な ADL が開発されている。これら及びその他の ADL は文献[32]で幅広くサーベイされている。また、リターゲッタブル・コンパイラやコンパイラ最適化技術に関しては書籍[33]や[34]に詳しい。

5. おわりに

本稿ではハードウェア／ソフトウェア・コデザイン技術について幅広く解説を行った。現在まで新興 EDA ベンダを中心に、様々なコデザイン用のツールが開発されてきた。これらのツールは部分的にコデザインを支援するものの、コデザイン・フロー全体を提供する実用的な統合環境はまだ存在していない。また、本稿で紹介したコデザイン・フローのうち、論理合成やコシミュレーションについては成熟したツールが存在し、実際の製品開発に広く使用されているが、機能分割、スケジューリング、通信合成、および、動作合成については大部分を人手に頼っているのが現状である。現在、EDA ツールによる支援が限定的であるため、設計対象のアプリケーションや開発体制に応じてそれらの EDA ツールを上手く組み合わせ、良いコデザインのメソドロジを構築することが重要である。

参考文献

[1] W. Wolf, "A decade of hardware/software codesign," *IEEE Computer*, vol. 36, no. 4, 2003.

[2] G. De Micheli and R. K. Gupta, "Hardware/software codesign," *Proc. of IEEE*, vol. 85, no. 3, 1997.

[3] D. E. Thomas, J. K. Adams, and H. Schmit, "A model and methodology for hardware-software codesign," *IEEE Design & Test of Computers*, vol. 10, no. 3, 1993.

[4] G. De Micheli, "Computer-aided hardware-software codesign," *IEEE Micro*, vol. 14, no. 4, 1994.

[5] G. De Micheli (松永裕介訳), "ハードウェア／ソフトウェア協調設計のコンピュータ支援における問題および方法について," *情報処理*, vol. 36, no. 7, 1995.

[6] 今井正治, "ハードウェアの見積りと生成," *情報処理*, vol. 36, no. 7, 1995.

[7] 安浦寛人, "基本ソフトウェアとコデザイン," *情報処理*, vol. 36, no. 7, 1995.

[8] 宮崎敏明, "信号処理分野におけるコデザイン," *情報処理*, vol. 36, no. 7, 1995.

[9] 今井正治, 武内良典, "ハードウェア／ソフトウェア・コデザイン手法," *電子情報通信学会誌*, vol. 81, no. 11, 1998.

[10] G. De Micheli and M. Sami (Ed.), *Hardware/Software Co-Design*, Kluwer Academic Publishers, 1996.

[11] J. Staunstrup and W. Wolf (Ed.), *Hardware/Software Co-Design: Principles and Practice*, Kluwer Academic Publishers, 1997.

[12] A. A. Jerraya and J. P. Mermet (Ed.), *System-Level Synthesis*, Kluwer Academic Publishers, 1999.

[13] G. De Micheli, R. Ernst, and W. Wolf (Ed.), *Readings in Hardware/Software Codesign*, Morgan Kaufmann Publishers, 2001.

[14] R. Gupta, *Co-Synthesis of Hardware and Software for Digital Embedded Systems*, Kluwer Academic Publishers, 1995.

[15] T.-Y. Yen and W. Wolf, *Hardware-Software Co-Synthesis of Distributed Embedded Systems*, Kluwer Academic Publishers, 1995.

[16] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, *Hardware-Software Co-Design of Embedded Systems: The Polis Approach*, Kluwer Academic Publishers, 1997.

[17] Y.-T. S. Li and S. Malik, *Performance Analysis of Real-Time Embedded Software*, Kluwer Academic Publishers, 1998.

[18] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems*, Prentice Hall, 1994.

[19] Open SystmC Initiative, http://www.systemc.org/.

[20] T. Grotker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*, Kluwer Academic Publishers, 2002.

[21] SpecC Technology Open Consortium, http://www.specc.org/.

[22] D. D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, and S. Zhao, *SpecC: Specification Language and Methodology*, Kluwer Academic Publishers, 2000.

[23] L. Cai and D. D. Gajski, "Transaction level modeling: An overview," *Proc. of Int'l Conf. on Hardware/Software Codesign and System Synthesis*, 2003.

[24] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers, 1992.

[25] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[26] J. P. Elliott, Understanding *Behavioral Synthesis: A Practical Guide to High-Level Design*, Kluwer Academic Publishers, 1999.

[27] 本田晋也, 高田広章, 中島浩, "SpecC によるソフトウェア記述の実装記述への変換," *情報処理学会論文誌 ACS3*, vol. 44, no. SIG11, 2003.

[28] J. Sato, A. Y. Alomary, Y. Honma, T. Nakata, A. Shiomi, N. Hikichi, and M. Imai, "PEAS-I: A Hardware/software co-design system for ASIP development," *IEICE Trans. Fundamentals*, vol. E77-A, no. 3, 1994.

[29] M. Itoh, S. Higaki, J. Sato, A. Shiomi, Y. Takeuchi, A. Kitajima, and M. Imai, "PEAS-III: An ASIP Design Environment," *Proc. of Int'l Conf. on Computer Design*, 2000.

[30] B. Shackleford, M. Yasuda, E. Okushi, H. Koizumi, H. Tomiyama, and H. Yasuura, "Satsuki: An Integrated Processor Synthesis and Compiler Generation System," *IEICE Trans. Information & Systems*, vol. E79-D, no. 10, 1996.

[31] H. Yasuura, H. Tomiyama, A. Inoue, and F. N. Eko, "Embedded System Design Using Soft-Core Processor and Valen-C," *IIS Journal of Information Science and Engineering*, vol. 14, no. 3, 1998.

[32] A. Halambi, P. Grun, H. Tomiyama, N. Dutt, and A. Nicolau, "Automatic Software Toolkit Generation for Embedded Systems-on-Chip," *Proc. of Int'l Conf. on VLSI and CAD*, 1999.

[33] P. Marwedel and G. Goossens, *Code Generation for Embedded Processors*, Kluwer Academic Publishers, 1995.

[34] R. Leupers and P. Marwedel, Retargetable Compiler Technology for Embedded Systems: Tools and Applications, *Kluwer Academic Publishers*, 2001.

[1] 2003 年に LISATek 社は CoWare 社に買収された。