

1. シーズとしてのシステムレベル設計技術

システムレベル設計フローを検討するためのシーズ調査として、現状のシステムレベル設計技術調査を行った。調査としては、まず、

EDA ビジョン研究会の「2002 年 EDA 技術ロードマップ」^[1]

を参考に、技術の傾向を調べ、次に、実際に、

大学・研究機関の開発システム

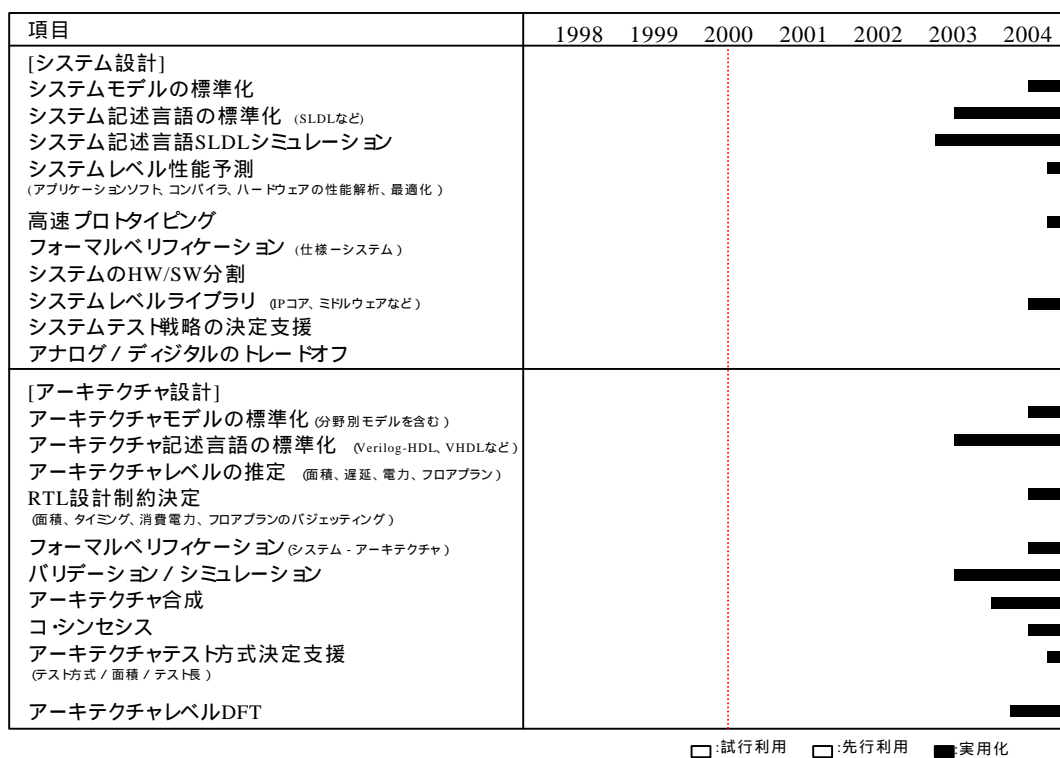
EDA ベンダーが提供する開発システム

からそれぞれ 3 つの開発システムを選んでその技術概要を調査した。対象開発システムは、研究会に参加している EDA ベンダーのツールを中心に、最近、話題になっているものを選んだ。調査は一般に公開されている情報源に限定し、ウェブサイト、書籍等からの情報に基づいた。調査のポイントとしては、システムレベル設計にどのような技術が使われているかをそれぞれの開発環境で調査し、それぞれの開発環境における設計フローと使用されている言語についても調査した。

1.1. EDA 技術の予測

まず、EDA ビジョン研究会の「2002 年 EDA 技術ロードマップ」によると、今後、数年の技術予測は、図 3.1-1 のようになっている。

EDA ビジョン研究会の予測では、2000 年にはシステム設計及びアーキテクチャ設計に関連するほとんどの技術が試行段階になっている。実際にはこれらの技術はどのようなになっているかを調べるために、6 つの開発システムを選び、調査を行った。



(EIAJ 『2002 年 EDA 技術ロードマップ』、1998 年、P.79 より)

図 3.1-1 EDA 技術ロードマップ

1.2. 大学・研究機関係システム

1.2.1. POLIS

POLIS^[2,3]はカリフォルニア大学/バークレー校で開発されたシステムで、マイコンやDSP をベースとした組込み系システムを設計するための協調設計環境である。POLIS の特徴は、下記の通りである。

- 機能とアーキテクチャの明確な分離により、最適なアーキテクチャ探索を実現
- ソフトウェアの性能評価のために、Fuzzy Instruction Model を導入

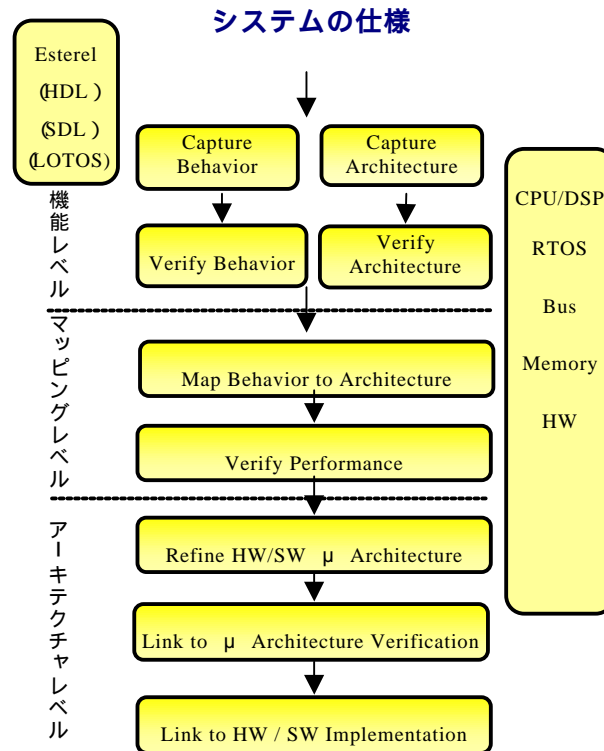


図 3.2-1 POLIS 設計フロー

設計フロー（図 3.2-1）は、ESTEREL や VHDL/Verilog-HDL のサブセットなどを用いてシステム動作を記述し、検証を行う。次にその動作を実現するためのアーキテクチャを CPU、DSP、バス、メモリ、ハードウェアなどのコンポーネントで定義し、動作をアーキテクチャにマッピングし性能検証をしながらハードウェア/ソフトウェア分割、コンポーネントの選択などを行い最適なアーキテクチャを決定する。この時、システムのソフトウェア・コードを実際のプロセッサに依存しない仮想的なプロセッサのインストラクションセット（Fuzzy Instruction Set）にコンパイルし、より抽象度の高い性能検証が可能になっている。次にアーキテクチャをマイクロアーキテクチャレベル(詳細な命令セット、RTL モデル、プログラム言語など)に置き換え、より詳細

な性能評価を行い、プロトタイプを作成する。

1.2.2. OCAPI-xl

OCAPI-xl^[4]はベルギーに本拠地を置く半官半民の非営利研究機関 IMEC(Independent Micro Electronic Research Center) によって開発され、デジタル信号処理用 ASIC の開発環境である OCAPI を拡張し、ソフトウェアを含めたシステム全体を扱えるようにしたもので、特徴は下記の通りである。

- ハードウェア、ソフトウェアを共通に表現できる C++による単一システムモデルのサポート
- 並列性と、ハードウェアとソフトウェアの間で共通な時間の概念の導入
- メッセージパッシング、セマフォ、シェアードバリアブルなどのハイレベルなコミュニケーション表現のサポート
- 他の言語とのインタフェースをサポート

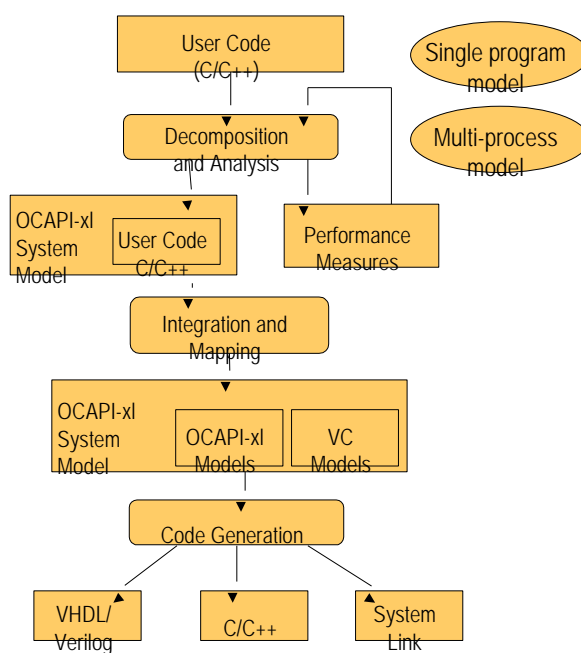


図 3.2-2 OCAPI-xl 設計フロー

OCAPI-xl の設計フロー（図 3.2-2）は、ハイレベルな実行可能なモデルからスタートする。典型的にこのモデルは、ユーザ定義のシングル・プログラム（C/C++コード）が使われる。次に、ユーザがシングル・プログラムをブロック単位に分解し、マルチプロセス・モデル（OCAPI-xl System Model）にする。ここで並列性の概念が導入さ

れる。さらに、OCAPI-xl のサポートする記述を用いて、プロセス間コミュニケーションが定義される。マルチプロセス・モデルをシミュレートすることにより、性能解析が可能になり、メッセージ間隔やプロセス使用率といった様々な性能測定基準を得ることができる。これらの測定値は、適切にプロセス分解が行われたか、正しいシステムコミュニケーションモデルが使われたかどうかを指し示す指標になる。

性能解析により、適切なマルチプロセス・モデルが作られた後で、設計フローは、Integration and Mapping フェーズに入り、個々のブロックのリファインが行われる。ここでは、既存のコアと OCAPI-xl C++モデルの両方を用いることができる。OCAPI-xl C++モデルを用いた場合は、ハードウェア、ソフトウェアのいずれにも変換可能になる。最後に、コード生成フェーズで、実装コードの生成が行われる。ハードウェアブロックは、合成可能な VHDL コードまたは Verilog-HDL コードに、ソフトウェアブロックは、OS (SoCOS)上の C / C++コードに自動変換される。また、それぞれのブロックのインタフェースも生成される。

1.2.3. IpChinook

IpChinook^[5]はワシントン大学で開発された設計環境である。応用範囲は IP ベースの制御系組込みシステムである。このような IP ベースシステム設計においては、個々のコンポーネントの設計よりも、コンポーネント間のインタフェースや最適化などシステムインテグレーションのほうに作業時間を要する。

アーキテクチャやマッピングはユーザが指定しなければならないが、CPU やメモリなどブロック間の制御やインタフェースを自動生成できるようになっている。

IpChinook の特徴は下記の通りである。

- IP ベース・Design Reuse 設計手法のサポート
- 機能ブロック間の制御やインタフェースを自動生成
- SCSI、USB、IrDA など豊富な通信プロトコルライブラリーを用意
- 専用言語をサポート (Java、Pia、C)

IpChinook の設計フローは図 3.2-3 で示す。IpChinook の入力には動作記述、ターゲット記述及びアロケーション定義で構成される。設計者はシステムの機能をプロセスモジュールで定義し、これらのモジュールの動作条件を定義する。そして、ターゲットのアーキテクチャを定義する。アロケーション関数を用い、プロセスを CPU に、チャンネルをバスにマッピングする。IpChinook のほうでモードマネジャー、ブロック間のインタフェース回路、ドライバなどを自動的に生成する。コミュニケーションプロトコルはターゲットアーキテクチャ独自のリアルタイム OS として生成される。

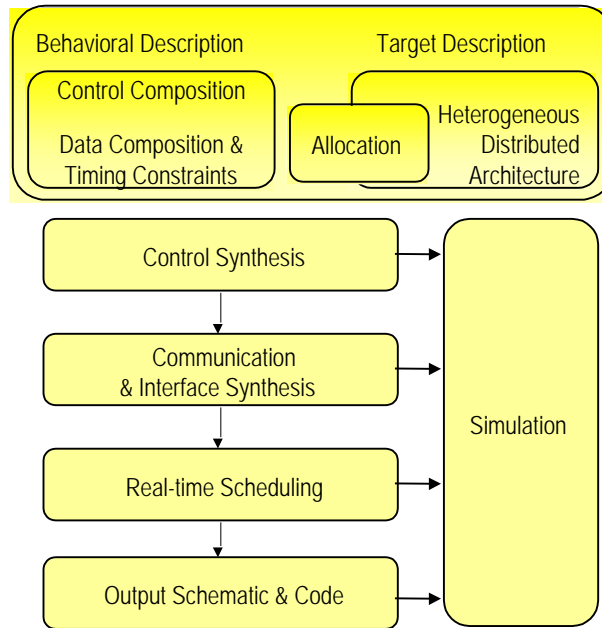


図 3.2-3 IpChinook 設計フロー

システム動作については Pia というシミュレーション環境のもとで協調検証を行うことができる。しかも、最初のターゲット非依存記述から自動合成されたデザインまで各段階のモデルを使用することができる。このように、システムインテグレーションの作業を軽減することによって、いろいろなアーキテクチャを試すことができる。

1.3. ベンダー系システム

1.3.1. CoCentric

CoCentric^[6]は Synopsys 社のシステムレベル設計用ツール群の総称である。CoCentric は複数の製品から構成される。データフロー系と制御系の混在システムをモデル化しシミュレーションするツール (CoCentric System Studio)、浮動小数点システムを固定小数点システムに変換するツール (CoCentric Fixed-Point Designer)、ピヘイビアレベル SystemC コードからゲート / R T レベルへ論理合成するツール (CoCentric SystemC Compiler) から構成されている。

CoCentric の設計フローは図 3.3-1 で示す。

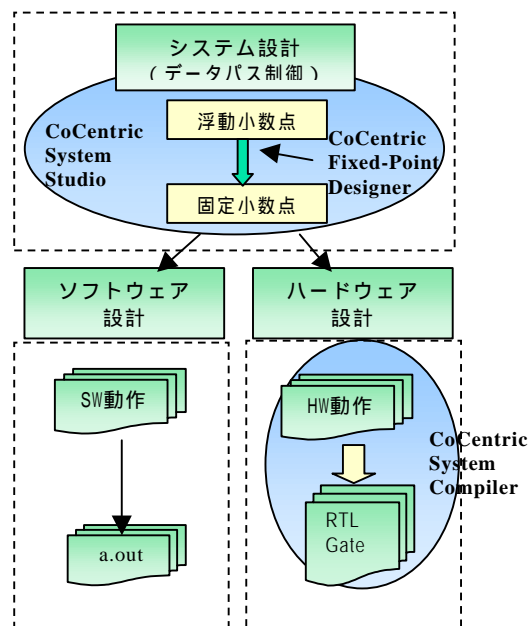


図 3.3-1 CoCentric 設計フロー

CoCentric の特徴は下記の通りである。

- システムレベル設計に関して SystemC を共通言語としてサポート。システム設計フローのハードウェア側について、最上位のシステム仕様レベルから下位のゲートレベル実装まで SystemC によりカバーしている。
- 他言語へのインタフェース有り。システムレベル仕様確定時には Matlab¹、SDL² へのインタフェース、ハードウェア実装時は Verilog-HDL、VHDL をサポート。
- データフローと FSM³の混在した階層化モデルによりシステムを記述する。使

¹ Matlab: プログラミング言語の一つ、制御系の解析などに使われる。

² SDL: Specification and Description Language

³ FSM: Finite State Machine

用されるモデルはタイプパラメータによりデータタイプを指定でき、異なる抽象度での設計へ同一モデルを再利用できる。(System Studio)

- C/C++ソフトウェアデバッグツールへのインタフェースを備えて協調検証可能
- 多くの論理シミュレータ、システム設計ツールとのインタフェースあり。

1.3.2. N2C

N2C^[7]は、IMEC からスピノフしたベンチャー企業 CoWare 社⁴によって開発されたツールであり、IMEC での研究成果がベースになっている。

N2C の特徴は下記の通りである。

- ハードウェア / ソフトウェア分割前の機能仕様定義からハードウェア / ソフトウェア実装までの設計フローをサポート
- 独自の言語(CoWareC⁵)で様々な抽象モデルをサポート
- ブロック間コミュニケーションのリファインをサポート
- ハードウェア / ソフトウェア・インタフェース合成

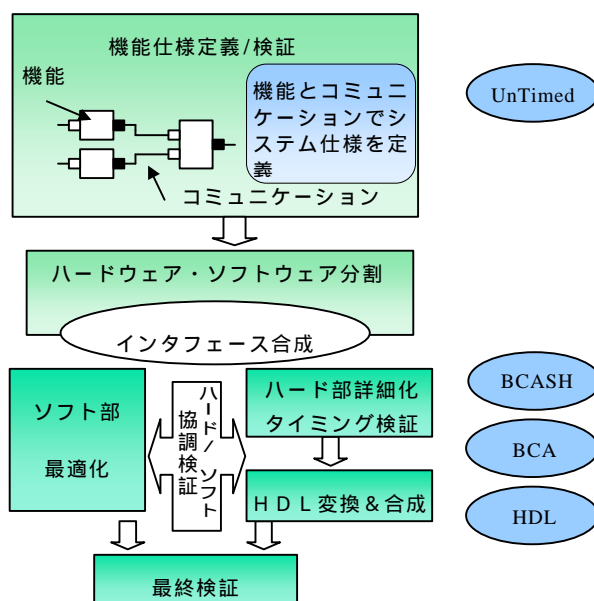


図 3.3-2 N2C 設計フロー

N2C の設計フローは図 3.3-2 で示す。設計者は最初に時間を考慮しない UnTimed の CoWareC を使ってシステムのビヘイビアを定義する。次に解析機能を使いながらハー

⁴ N2C に関する記述は CoWare 社より了承済み。

⁵ ANSI C にクロック、ビット、コンカレンシを拡張した CoWare 社独自の言語。SystemC に対応予定。

ドウェア/ソフトウェアの分割を変更して最適なアーキテクチャの探索を行う。このとき、デバイスドライバやアドレスデコーダといった、ソフトウェアブロックとハードウェアブロックのインタフェースの自動生成機能を使用することができる。ハードウェアでは UnTimed⁶、BCASH⁷、BCA⁸、RTC⁹の記述レベルをサポートしており、必要に応じて、抽象度をブレイクダウンしながら設計を行うことができる。RTC まで詳細化されたブロックは Verilog-HDL、VHDL の両方にトランスレートすることが可能である。ブロック間のコミュニケーションは、データタイプ、in/out、コントロールのプリミティブなプロトコルから、ハンドシェイクを考慮したプロトコルまでをサポートしており、状況に応じてリファインしていくことができる。

1.3.3. VCC

VCC^[8]は、3.2.1 で紹介した POLIS の成果をベースに Cadence 社により開発されている。

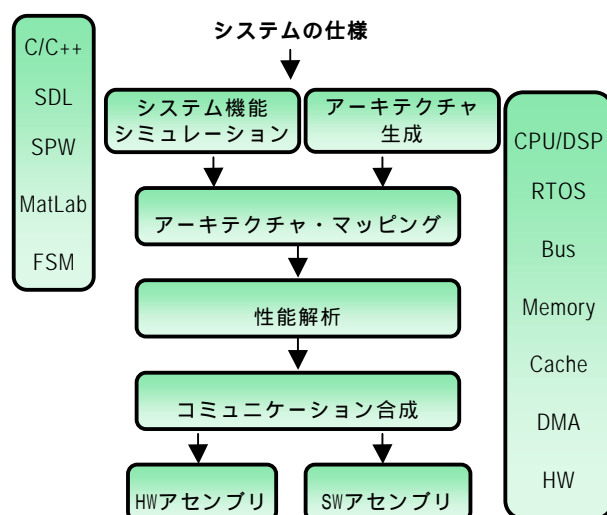


図 3.3-3 VCC 設計フロー

特徴は下記の通りである。

- システムの機能とアーキテクチャを分離することで最適なシステム構成探索
- C++記述によるカスタマイズ可能な、性能モデリング構造を提供

⁶ UnTimed: 時間概念を持たない抽象度レベル

⁷ BCASH: BCA Shell、UT レベルと BCA レベルの中間の抽象度レベル

⁸ BCA: Bus Cycle Accurate、クロック精度の抽象度レベル

⁹ RTC: Register Transfer C、RT レベルの抽象度

- Virtual Processor Model¹⁰や ISS (Instruction Set Simulator) 統合による、異なる抽象度のソフトウェア性能見積り
- 異なる言語や設計環境で作成された機能モデルのインポート
- コミュニケーション合成やハードウェア/ソフトウェアのエクスポートによる協調検証へのリンク

VCC の設計フローは図 3.3-3 で示す。システム機能は、C、C++、SDL、Matlab Simulink、SPW¹¹、状態遷移図などで作成した機能ブロックを接続し定義する。機能検証は、シミュレーションにより行う。次にアーキテクチャ構成を、抽象化された、CPU、DSP、バス、メモリ、RTOS、ハードウェア、キャッシュ、DMA などのモデルで定義し、機能ブロックとそれを処理するアーキテクチャブロックとのマッピングを行うことで性能解析を可能にする。これらの作業は全て GUI 上で行い、追加・変更も可能であるため、最適なシステム構成探索が可能となる。また、C++記述のアーキテクチャ・サービスという構造により、性能モデルや消費電力モデルも定義でき、要求レベルに応じたモデルのリファインが可能である。HDL トップレベル構造記述、コミュニケーション合成、ソフトウェアタスク生成、協調検証ツール実行スクリプト生成機能を有し、下流設計環境とのリンクが取られている。

¹⁰ POLIS の Fuzzy Instruction Model をベースに、アーキテクチャサービス構造を追加したもの。

¹¹ Cadence 社の DSP 開発ツール

1.4. シーズ調査のまとめ

以上の調査した6つの開発システムを基に、現状のシステム設計技術についてまとめると、以下の技術が現時点で実現されている、または実現されつつある。

- 機能検証
- 性能予測
- ハードウェア/ソフトウェア分割
- ハードウェア/ソフトウェア協調検証

しかし、これらの技術も現状では、そのほとんどが限定した条件下で実現されており、今後は、その条件の拡大や技術の一般化、標準化を行っていくが必要になる。例えば、システムの機能検証の場合、C/C++を用いて、機能のシミュレーションは行えるが、システムとして何が検証されるべきか、そのためには、システムをどう記述すれば良いのかといった方法論や記述方法については、明確になっていない。これらの明確化、標準化を行っていくが必要になる。また、性能予測は、いくつかの開発システムで既の実現されているが、より複雑なモデルに対して、高速かつ高精度な見積りを実現できることが必要になる。

未解決の課題として残されている技術は、以下の技術があげられる。

- 見積りの高速化
- 面積、電力、コスト見積り
- システム仕様、設計制約の記述標準化
- テスト戦略の決定支援
- アナログ/デジタルのトレードオフ

見積りについては、前述したように、高速・高精度な見積りを行うことのできる技術は実現されていない。また、現在、実現されている見積りは、性能及び、一部消費電力だけで、チップ面積、開発コストなどの見積りは実現されていない。さらに、機能仕様よりも上位の、システム仕様や設計制約の記述の標準化などは、進んでいないようである。このように、現状では、システム設計技術は、まだ多くの課題が残されている。実現された技術のブラッシュ・アップを図るとともに、大学や研究機関等で未解決の技術を確立していく必要がある。

[参考文献]

- [1] EIAJ EDA ビジョン研究会、『2002 年 EDA 技術ロードマップ』、1998 年
- [2] “Hardware-Software Co-Design of Embedded Systems, The POLIS Approach”, Kluwer Academic Publishers, 1997
- [3] <http://www-cad.eecs.berkeley.edu/Respep/Research/hsc/abstract.html>
- [4] <http://www.imec.be/ocapi>
- [5] <http://www.cs.washington.edu/research/chinook/index.html>
- [6] <http://www.synopsys.com>
- [7] <http://www.coware.com>
- [8] <http://www.cadence.com>