システムＬＳＩ設計技術　101－7
（2001．5．17）

〔招待論文〕

# システム設計記述言語の動向

橘　昌良

高知工科大学　電子光システム工学科
〒782-8502 高知県香美郡土佐山田町宮の口 185, (0887)57-2212
tacibana@ele.kochi-tech.ac.jp

**あらまし**　近年のデバイス技術の進歩により，ワンチップに1000万を超えるトランジスタの集積が可能となってきた．一方，設計規模の増大，設計の複雑さ，市場からの開発期間短縮の要求により，従来のLSI設計手法では要求を満たすLSIの設計が困難になってきた．この問題への解決方法として，システムレベル設計手法と呼ばれる新しい設計方法が提唱されているが，その実現に必要な設計言語やツール等にさまざまな課題が残されている．本報告ではJEITAシステムレベルデザイン研究会で行ったシステムレベル設計手法の提案とシステムレベル設計言語の適用化検討に基づき，設計者からの要望および大学/研究機関/EDAベンダー等が持つ設計技術，システムレベル設計言語の動向についてなど解説する

**キーワード**　システムレベル設計，　システムLSI，　システムレベル記述言語，　EDA, HDL, 高位合成

# A Survey of System Level Design Methods and Languages

Masayoshi TACHIBANA

Department of Electronic and Photonic Systems Engneering, Kochi University of Technology
185 Miyanokuchi, Tosayamda, Kochi, 782-8502, JAPAN, +81-887-57-2212
tacibana@ele.kochi-tech.ac.jp

**Abstract** In this paper, a survey of system level design methods and description languages is presented. This survey is based on a study of JEITA System Level Design Study Group. The main goal of this paper is to make a understanding of a larger variety of system level design methods and description languages. We first define a model system design flow, then we classify the existing system level design systems and description languages developed in universities, research institutes, and EDA vendors.

**Key words** System level design, System LSI, System level description language, EDA, HDL, High level Synthesis

## 1 はじめに

近年のデバイス技術の進歩により，ワンチップに 1000 万を超えるトランジスタの集積が可能となってきた．一方，設計規模の増大，設計の複雑さ，市場からの開発期間短縮の要求により，従来の LSI 設計手法では要求を満たす LSI の設計が困難になってきた．この問題への解決方法として，システムレベル設計手法と呼ばれる新しい設計方法が提唱されているが，その実現に必要な設計言語やツール等にさまざまな課題が残されている．システムレベルデザイン研究会 (以下 SLD 研究会) では，設計者からの要望および大学/研究機関/EDA ベンダー等が持つ設計技術を調査/検討し，期待されるシステムレベル設計フローの一例を提案した．さらに，この設計フローを実現するための技術課題の抽出と，3 つのシステムレベル設計言語である SystemC, SpecC, UML を選定し，提案フローへの適用評価を行った．

本報告は，以下の様に構成されている．2 節では，システムレベル設計に対する設計者のニーズ調査結果をまとめた．3 節では，システムレベル設計環境の現状と将来予測について調査した結果について述べる．4 節では，SLD 研究会が独自に検討したシステムレベル設計手法と設計フローについて述べる．5 節では，標準化を目指すシステムレベル設計言語について調査し，提案する設計手法を実現することができるかどうかを検討した．6 節では，本活動結果をもとに，あるべきシステム設計手法を実現するための課題についてまとめた．

## 2 設計者ニーズ調査

SLD 研究会では，システム設計においてどのような課題があるか把握するため，設計者ニーズの調査を行った．まず，1999 年 2 月 24 日に設計者技術懇談会 *1 を開催した．この懇談会は，設計の第一線で活躍している 5 人の設計者に，予め用意しておいた 5 つのテーマに関して，意見を述べてもらい自由に討議するという形で進めた．続いて，1999 年 9 月に，JEITA (旧 EIAJ) 加盟企業を中心に 7 分類 53 項目にわたる設計者アンケート *2 を実施した．そして，これらの結果と研究会メンバーが認識しているニーズを基に分析し，設計者ニーズとして課題を分類，その解決案を検討した．

### 2.1 課題となる点

設計者の抱える問題点は以下のように要約できる．

**■機能決定に関する課題** 仕様決定の際に十分な検討がなされないことによる仕様内容の曖昧，仕様表現の曖昧さからくる仕様の誤解釈が挙げられる．また，機能仕様を現実的な時間で検証することが困難な点が挙げられる．

**■アーキテクチャ決定に関する課題** 机上レベルで行われているために，妥当性を確認し決定する手段がないことがあげられる．また，決定したアーキテクチャに対する見積りの甘さによる後工程でのやり直しがあげられる．

**■実装設計へのインタフェースに関する課題** 実装系へのインタフェースが不明確である点，RTL から実装までを実証してくれる設計環境が少ないことがあげられる．また，機能仕様検証やアーキテクチャ検討結果と，実装設計後の検証結果を合わせることが難しい点が挙げられる．

**■設計全般での課題** IP の組込みやカスタマイズに関する問題が挙げられる．これは，IP 利用への要求が高い反面，IP の有効性を簡単に評価できない点と，性能を満足するようにカスタマイズする際多くの工数がかかることによる．さらに，システムのソフトウェアについての検証を行う場合，現在の協調検証環境では満足のいく性能が得られていない点が挙げられる．

### 2.2 課題解決方法の検討

**■機能決定に関する課題解決** 仕様内容の曖昧さに関しては，機能仕様定義の方法論を検討していく必要がある．また，曖昧な表現による仕様の誤解釈に関しては，仕様表現を実行可能なものにすることにより曖昧な表現を除去することが考えられる．

**■アーキテクチャ決定に関する課題解決** アーキテクチャの妥当性の確認，及び決定手段がない点に関しては，最適なアーキテクチャを高速に探索する仕組みを検討していく必要がある．また，アーキテクチャに対する見積りの甘さに関しては，必要な見積りモデルの導入を考えていく必要がある．

**■実装設計へのインタフェースに関する課題解決** 実装設計の前段階においてハードウェア/ソフトウェアのインタフェース合成を導入する必要があると考える．これにより，実装後のリスピンを減らすことできる．さらに，一貫したテストベンチの導入によって，機能仕様検証，アーキテクチャの検討結果，実装設計後の検証結果を統一した環境で検証することが可能となる．

**■設計全般での課題解決** IP の組込み，カスタマイズに関しては，IP のインタフェースの標準化や IP の合成技術の導入，システムレベル設計のフロー全体にわたる様々なフェーズでの IP の充実が必要となる．また，システムの協調検証の性能不足に関しては，ネックとなっているハードウェアモデルの抽象度を上げること，アクセラレータなどとの連係をとることによって，性能を引き上げることができると考える．

### 2.3 ニーズ調査のまとめ

これらのニーズから抽出した課題に対する解決案は，本研究会を含め，企業・大学の研究機関，EDA ベンダー等，全体の取組み無くして，実現は困難と考える．

---

*1 「EDA アニュアルリポート 1998」に掲載

*2 「EDA アニュアルリポート 1999」に結果を掲載

## 3 シーズとしてのシステムレベル設計技術

EDA ビジョン研究会の予測 [1] 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. これらの技術の現状を調査するために大学研究機関によるものと EDA ベンダーが提供するそれぞれ 3 つの開発システムを選択した.

### 3.1 大学研究機関系システム

■POLIS [2, 3] 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, 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.

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, ソフトウェアは仮想的なプロセッサの Fuzzy Instruction Set にコンパイルされ, 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.

■OCAPI-xl [4] ベルギーに本拠地を置く研究機関 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ある.

実行可能なモデルであるユーザ定義のシングル・プログラム (C/C++コード) によりシステムの記述を行う. 次に, これ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. 並列性の概念はこの時点で導入される. さらに,OCAPI-xl のサポートする記述を用いて, プロセス間通信が定義される. 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.

マルチプロセス・モデルが作られた後, 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. 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.

■IpChinook [5] 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. 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.

入力は動作記述, 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, チャネルをバスにマッピングする. 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される. 通信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.

システム動作は 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.

### 3.2 ベンダー系システム

■CoCentric [6] Synopsys 社のシステムレベル設計用ツール群の総称であり, 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), ビヘイビアレベル 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.

- SystemC を共通言語としてシステムレベル設計を行う.
- Matlab [*3], SDL, Verilog-HDL, VHDL とのインターフェースが可能.
- データフローと FSM の混在した階層化モデルによりシステムを記述する.
- 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がある.

■N2C [7] IMEC からスピンオフしたベンチャー企業 CoWare 社によって開発されたツールであり,IMEC での研究成果に基づいている.

まず, 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, ソフトウェアとハードウェアのブロック間インタフェースを自動生成することが出来る. ハードウェアでは UnTimed [*4], BCASH [*5], BCA [*6], RTC [*7] 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からは Verilog-HDL, VHDL のコードを生成することが可能であ

[*3] Matlab:制御系の解析などに使われるプログラミング言語の一つ
[*4] UnTimed: 時間概念を持たない抽象度
[*5] BCASH: BCA Shell,UT レベルと BCA レベルの中間の抽象度
[*6] BCA: Bus Cycle Accurate, クロック精度の抽象度
[*7] RTC: Register Transfer C,RT レベルの抽象度

る. ブロック間通信は, 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.

■VCC　[8] POLIS の成果をベースに Cadence 社により開発されている.

システム機能は, C, C++, SDL, Matlab , Simulink, SPW[*8] , 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, 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.

### 3.3　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, システム設計技術についてまとめると, **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**, が現時点で実現, または実現されつつある.

しかし, 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としては, **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**などがあげられる.

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.

## 4　提案するシステムレベル設計手法

### 4.1　設計フロー

既存のシステムレベル設計フローには**要求仕様定義フェーズ**と**実装設計フェーズ**の間に大きなギャップがある. このギャップがシステムレベル設計に関する様々な問題や課題を生み出す原因になっている. この二つのフェーズの間を**機能決定フェーズ**と**アーキテクチャ決定フェーズ**で埋める必要がある.

機能決定フェーズでは要求仕様定義で決定したシステムの機能仕様をシステムレベル設計言語により定義（機能定義と呼ぶ）しシミュレーションにより検証する. これにより, システムの機能仕様とそれを表現した機能定義との妥当性が評価される. アーキテクチャ決定フェーズでは機能決定フェーズで検証された機能定義を実現するときにどのようなシステム・アーキテクチャで実現するのかを決定する. このフェーズは**設計空間生成**と**設計空間探索**の 2 つに分けることができる.

設計空間生成フェーズでは後の設計空間探索の対象となる分割された**機能定義**と**アーキテクチャ候補群**を生成する. この作業は**プロファイリング**を用いて収集した情報をもとに行われる.

設計空間探索フェーズでは設計空間生成で作成された分割された機能定義をアーキテクチャ候補群から選ばれたアーキテクチャに写像しシミュレーション等でその性能を見積る. そしてこのアーキテクチャ候補の選択と写像を繰り返し, 設計制約を満たす最適なアーキテクチャを探索決定する. 最適アーキテクチャが決定されるとそれをもとにハードウェア・コードやソフトウェア・コードが自動的に生成される.

この設計フローでは要求仕様定義から導かれたテスト仕様や機能検証の結果を期待値として用いてテストベンチを作成する. それを設計フロー全体にわたって適用することによりフロー全体での整合性を保証しようとする.

また各設計フェーズでの IP や見積りデータベースなどを関連付けてまとめた**設計データベース**を用意することによりフロー全体で過去の設計資産の再利用性を高めようとしている.

### 4.2　分析

**■要求仕様定義**　現状では要求仕様定義に対応できるツールはなく, ユーザは各々のシステムに応じてシステム仕様および設計制約を作成する必要がある. この分野はソフトウェア設計手法の拡張およびその適用が期待される.

**■機能決定**　全てのツールが機能記述をシミュレーションすることで実現している. しかしその検証内容および機能定義をいかに記述するかは明確になっておらず機能定義手法の確立が必要である.

**■アーキテクチャ決定**

**設計空間生成**　プロファイリングに近い機能を持ち, 機能定義からプロセスモデルの変換までに対応できるツールが存在する. アーキテクチャ生成についてはアーキテクチャモデルを候補として提示する機能を有しているツールは存在するが生成そのものが出来るツールはない. 整合性検証は全システムでシミュレーションにより実現されている.

**設計空間探索**　徐々にツールが整備されてきているが処理速度以外の消費電力やコストの見積りの実現, ハードウェア／ソフトウェアのいずれで実現するかのトレードオフ評価技術, また, ある程度の精度を確保し高速に見積れる手法

[*8] Cadence 社の DSP 開発ツール

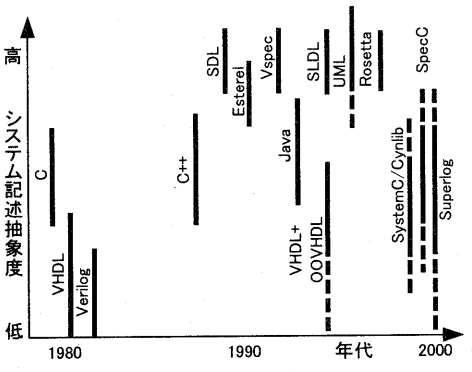

図 1: 言語マップ

の確立とツールの実現が課題として残されている.

**■実装設計へのインタフェース** ほとんどのツールが動作合成および論理合成技術によりハードウェア部分のインタフェース生成できる.

**■テストベンチ** 現時点では, システムレベル設計で対応しているツールはほとんど無い. テスト仕様や機能検証結果からそれに合致するテストベンチの生成を実行するツールの実現が望まれる.

以上の評価, 考察から我々の提案フローは設計者ニーズからの課題をほぼクリアでき, また現状の技術の延長線上で実現できる可能性は十分にあることがわかる.

## 5 システムレベル設計言語の動向

### 5.1 全体動向

システム LSI のシステムレベル設計が重要になるとともに, 設計フローを実現するためのシステムレベル設計言語の重要性が増してきた. 特に 1999 年 9 月ごろから C/C++ ベースのシステムレベル設計言語の標準化推進団体が次々と発足し, 標準化の動きが活発になってきた.

本節では, このような背景のもと, システムレベル設計言語の調査結果をまとめている.

図 1 はシステムレベル設計への適用が試みられた既存のシステムレベル設計言語の動向を系統立てて整理したものである. システムレベル設計言語は, 大まかに, ハードウェア系言語, ソフトウェア系言語, 新規言語など上記以外, の 3 つの範疇に分類できる. 全体的な傾向としては, 年代とともに徐々に記述抽象度を広げ, 適用範囲がシステム全体をカバーするように推移している.

### 5.2 ハードウェア設計言語系の言語

既存言語のシステムレベル拡張を行うアプローチとして, VHDL にもとづくシステムレベル拡張は IEEE/DASC 内で 1996 年頃から議論され, 以前から OOVHDL, VHDL+ といった言語の標準化活動が行なわれてきた. Verilog-HDL に関しては, 2000 年に規格改訂が Verilog 2000 として行われ, 上位の抽象レベルでのシステム記述のためにも仕様が一部拡張されたが, ハードウェア記述言語としての位置づけは変わっていない. 他に Superlog が提案されている.

以下に主な言語の概要を示す.

**■OOVHDL** [9] (Object-Oriented Extensions to VHDL) IEEE/DASC の OOVHDL Study Group で 1998 年から議論が進められ, 標準化活動を行っている. OOVHDL では VHDL にオブジェクト指向性を取り入れて設計再利用性やシステム抽象化を高め, テストベンチも含むシステムの抽象化やカプセル化を用いた容易なモデル化を実現することを目的としている. VHDL にオブジェクト指向性を取り入れる考え方は古くから SUAVE [11] と Objective VHDL [12] で議論されていたが, 2000 年度に Objective VHDL が言語仕様として採択された.

**■VHDL+** [10] IEEE/DASC の SID (System and Interface based Design) 中心に標準化が進められていた. VHDL+は英国 ICL 社が開発し,70 以上の企業・大学がユーザ会に登録している. オブジェクト指向の概念は導入せず, ポートを介さないメッセージ通信やデータタイプを指定しない通信など抽象度の高い通信モデルのサポートが特徴である. 2000 年 10 月に SID, ICL 社は活動を中断し停止状態となっている.

**■Superlog** [13] Verilog 系システムレベル設計用言語として米国 Co-Design Automation 社から提案された新しい言語である. Verilog-HDL と C をベースにしてさらに VHDL と Java のエッセンスを加えて提案された. ハードウェア, ソフトウェア, システムの各設計分野について単一言語により設計手法の統一, 設計効率改善, 従来手法からの発展性などを実現する.

### 5.3 ソフトウェア設計言語系の言語

最近, 最も注目されているシステムレベル設計言語は, もともとソフトウェア用であったプログラム言語をシステムレベルに拡張した言語である.

#### 5.3.1 C/C++ベース言語の動向

C を拡張し, システムレベル設計言語として使用する研究は, 1980 年代後半に始まった. スタンフォード大学の DeMicheli 教授のグループにより開発された Hardware-C [16] が有名である. 1990 年代後半になり, EDA ベンダーからも C/C++をベースにしたシステムレベル設計言語が提案されてきた. カリフォルニア大学アーバイン校の Gajski 教授らにより開発された SpecC [20, 21] は ANSI-C 言語の仕様を拡張したものであり, 米国 Synopsys 社が中心になって開発した SystemC [24] と米国 CynApps 社 (現 Forte Design Systems 社) が開発した Cynlib citecynl はともに C++をベースにしている. また, 米国 C Level Design 社 [19] の様に ANSI-C/C++の両方をサポートする動きも

ある．また，シャープが開発した Bach-C [14, 15] や NEC が開発した BDL [17, 18] のように社内使用を目的とした C/C++ベースのシステムレベル設計言語も研究され，動作合成ツールの入力言語として社内での実績を出しているものも多い．

C/C++ベース言語は，C や C++のシンタックスを利用し，ハードウェアを記述するために必要なセマンティクスを拡張しているため，以下のような特徴を持つ．

- ハードウェアを記述するために，機能を追加している．
- 多くのソフトウェア設計者，システム設計者／アルゴリズム開発者に使用されている言語をベースとしており，容易に習得することができる．
- ソフトウェアプログラム開発環境を利用できる．
- HDL など他のシステムレベル設計言語にくらべ，検証スピードが速い．
- C/C++ベース言語から，RTL-HDL を自動生成できる動作合成ツール（高位合成ツールとも呼ぶ）の開発が他のシステムレベル設計言語に比べ進んでいる．

以下に，代表的な言語の概要を示す．

■SystemC　[24] 米 Synopsys 社が中心になって開発した言語で，推進団体 OSCI により標準化活動が進められている．

■SpecC　[20, 21] カリフォルニア大学アーバイン校の Gajski 教授が提唱した言語であり，推進団体 STOC [22] により標準化活動が進められている．

■Cynlib　[23] 米 CynApps 社が開発したハードウェア記述可能な C++ライブラリである．検証可能なシミュレーションカーネル，ハードウェア記述用のクラスライブラリ，モデル記述方法を提供している．ハードウェア記述のためのデータ型，並列性，階層記述，同期／非同期などの機能を含むほか，Verilog-HDL とのインタフェースを用意している．CynApps 社は 2001 年 3 月に Chronology 社と合併し，Forte Design Systems 社に社名変更した．

■C Level Design 社の ANSI-C/C++　[19] C Level Design 社は，ANSI-C と C++の両方の言語をそのまま使用できる設計検証環境を提供している．言語拡張は行っていないが，ハードウェアを記述するための工夫を行っている．ファンクショナル C と RTL － C との 2 つの記述レベルを定義している．CSim と呼ばれる検証ツールと System Compiler とよばれる C/C++モデルを RT レベルの HDL に変換するツールを提供している．また，OVI の Architectural Committee による “Semantics Reference Manual Standard for C++ class library” に従った System C++と呼ばれる C++クラスライブラリも提供している

■Hardware-C　[16] 動作合成を備えた設計検証システム（Olympus Synthesis System）における動作レベルのハードウェア記述用言語としてスタンフォード大学で開発された．C に対して，並列動作や構造の記述，タイミングなどの拡張を行っている．設計制約を記述できる点が，他の C/C++ベース言語にはない特徴である．すでに研究は終了しているが，CD や DAT のバスとオーディオバスとのインタフェース回路や画像処理などの設計に適用された例が報告されている．

■Bach-C　[14, 15] シャープとシャープヨーロッパ研究所が共同開発した言語である．C に，並列動作，同期／非同期通信，ビット幅，合成用のプラグマなどの拡張がなされている．動作合成ツール，検証ツールを含む Bach システムが用意されている．シャープ社内では画像処理，通信関係の設計を中心に実設計で適用されている．

■BDL　[17, 18] NEC が開発した言語である．C に，ビット幅，並列実行，割込み，ハンドシェイク，クロック・サイクル指定，レジスタやメモリの指定の機能を追加している．Cyber とよばれる動作合成ツールや ClassMate とよばれる C++シミュレータ環境とリンクし，動作レベルから RT レベルまでの設計検証環境が整っている．NEC 社内では，デジタル家電，ネットワーク，コンピュータなどデータパス回路，制御回路をとわず実回路の設計に使用されている．次世代の携帯電話のシステム開発にも適用されている．

### 5.3.2 Java ベース言語

オブジェクト指向言語の特徴であるクラスライブラリを用いて，本来の Java が持たないハードウェア的な性質を拡張しようとしている．このアプローチを採るグループは，C++に比べて以下の点で優れた性質を持っていると主張している．

- オブジェクト指向性を利用した高い生産性
- 標準的なプラットフォーム
- オブジェクト指向による高い構造化
- 暗黙の並列性の識別と抽出
- マルチスレッドによる明確な並列性

欠点として実行速度が遅いことがあげられるが，これが解決されれば，システムレベル設計言語としての有力な候補になる可能性を持っている．

■JavaTime　[26] 言語というよりは方法論にもとづくシステムである．カリフォルニア大バークレイ校の Newton 教授により提案されたエンベデッドシステム設計のための実験システムである．システムの仕様，モデル化，実装に対して，コンポーネントベースのアプローチを適用している．構文として，時間の概念の組込みは行わず，写像形式で時間概念を導入しているのが特徴．また，アーキテクチャ自動生成は行わず，最初に制限のない汎用プログラム言語で記述し，言語解析を通じて段階的に目的の制限された計算モデルへ変換する，SFR （Successive Formal Refinement） という仕様開発の方法論を導入している．

### 5.4 その他の言語

■UML [29, 30] (Unified Modeling Language) 3つのオブジェクト指向設計手法を統合した標準化言語であり, ソフトウェアシステム設計のための言語であるが, 近年になってUMLをハードウェア含めたシステムレベルへの記述に適用しようとうする動きが出てきている.

■SLDL [31] (Systems Level Design Language) これまでに発表された数少ない多言語系システムレベル設計言語の1つである. 多言語系言語とは, 1つの言語をベースにするのではなく, 言語自体が他の言語で記述されたシステムを取り込み統合できるハーネスのような仕組みを持った言語である. 1999年6月にシステムの設計制約記述を中心としたRosetta (Phase I) が発表された. このRosettaは, 設計制約記述に関してVspec言語を参考にしている.

■SDL [27] 通信のプロトコル仕様を記述するために開発された言語で, ITU-Tの標準規格となっている. テキスト及びグラフィカルなエントリ言語が定義されており, 主な特徴として, 動作の階層記述, 構造の階層記述, 状態遷移記述, メッセージ送受信記述, 時間記述を扱える.

■Esterel [28] 1980年代初期にリアクティブシステムのプログラム言語としてSophia-Antipolisで開発され, 1990年代初期にハードウェア設計のための変更が行われた. カルファルニア大バークレイ校で開発されたPOLISシステムのエントリ言語として採用されている. 主な特徴として, 動作の階層記述, 同時発生動作記述, 時間記述を扱える.

■**その他** 上記以外にも, 並列システムをモデリング, 検証するために開発されたCSP (Communicating Sequential Processes), リアクティブシステムの仕様言語として開発されたペトリネット, StateChartや後にSpecCへと展開されたSpecChart, ISOのFDT (Formal Description Technique) グループにより開発されたLOTOS等の言語が知られている.

### 5.5 適用評価

SLD研究会で提案したシステムレベル設計フローを1つの評価基準としてとらえ,SystemC, SpecC, UMLを提案フローに適用してそれぞれの言語の能力を評価し比較する.

■**要求仕様定義** 要求仕様定義とは実行可能なプログラムではなくシステムの要求事項が書いてあるものである. 従って, 自然言語を形式的な書式で記述することができ, IBMが考案した制約記述言語OCL (Object Constraint Language) [32]を採用し, 設計制約を記述できるUMLが有利である. SystemCやSpecCは, このような定義方法を仮定していないため, UMLとのリンクを考えるか言語仕様の拡張が必要になる. SystemC, SpecCにおいても, 少なくとも設計制約記述の拡張は検討しても良いと思われる.

■**機能決定** SystemC, SpecCともに可能であるが, それぞれは異なる優位性を持つ. SpecCは方法論が明確に規定されていることが有利であり, SystemCは既存ツールが使えるという点が有利である. UMLに関しては, ツールのサポートによりステートチャートの検証が可能であるが, 現状ではUMLを用いたハードウェア設計の方法論は未だ研究レベルであり, この設計工程がUML単独での使用の限界である.

■**アーキテクチャ決定** アーキテクチャの決定は, 言語／ツール／方法論のすべてがそろうことが望ましい. この観点から各項目を評価すると以下のようになる.

**プロファイリング** SystemCは既存のC/C++用のプロファイリングツールがそのまま使える. その半面, SpecCは, そのままのコードでは既存ツールを直接使うことができないので専用ツールの整備が必要になる.

**機能ブロック分割** SpecCではハードウェアとソフトウェアの記述手法の差異がないので, 分割の変更に伴う記述の変更がSystemCより少ない.

**分割整合性検証** 機能ブロック記述のシミュレーション実行が可能な点で, SystemC, SpecCに差異はない.

**アーキテクチャ生成** SystemC, SpecCともに, アーキテクチャ中のコンポーネント間の接続関係を記述することはできるが, アーキテクチャ候補を選択・評価するための基準や制約条件などの記法は用意されていない.

**アーキテクチャ・マッピングと見積り** 方法論が不足しているとともに, 利用目的に応じたさまざまな精度での見積り情報をトランザクション・モデルに記述する能力の点で不足がある. 特に, 粗い精度の見積りモデルを記述する観点で弱い.

総合的に見てSystemCとSpecCには一長一短はあるものの, 設計空間生成の機能ブロック分割とその検証までは可能である. しかし, アーキテクチャ生成からマッピング／見積り評価までは, どちらも満足できるレベルにはない.

■**実装設計へのインタフェース** SystemCがクロックを明確に記述するBCAレベルをサポートしている分, このレベルの設計への適用に向いている.SpecCは言語仕様の拡張か, 下流へのリンクを行うインタフェースが必要である.

■**テストベンチ生成** SystemC, SpecCともに, テストベンチ作成については設計者が記述可能である.

■**IP設計** SystemC, SpecCともに, 既存のHDLと比べて, オブジェクト指向の導入や抽象度の高い記述が可能である点で, モデルをIPとして記述したり再利用がしやすい. しかし, アーキテクチャIP化や見積りのためのIP化が行いにくい.

## 6 まとめ

本報告は1998年11月にEIAJ EDA技術委員会 (現JEITA EDA技術専門委員会) の下部組織として発足したSLD研究会が約3年間の活動を通じて「あるべきシステ

ムレベル設計像」を提案するために調査検討した内容を要約したものである

システムレベル設計技術は，現在最も活発に研究され，産業界でもさまざまな動きが見られる分野であり，状況は刻々と進化している SLD 研究会が発足してからの 3 年間でも，さまざまな標準化団体の発足，統廃合が起こり，新たな提案が次々と行われている．

本報告がシステムレベル設計を研究する大学・研究機関や，企業，標準化団体にとっての情報ソースとなるとともに，今後の研究，活動に方向性を与え，システムレベル設計言語の検討への指針となることを期待する．

最後に，本書の発行にあたり，SLD 研究会の調査活動にご協力頂いた多くのシステム設計者，国内外のベンダや研究機関の方々，発行の機会を与えて頂いた JEITA　EDA 技術専門委員会に感謝の意を表する

## 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
[9] http://www.vhdl.org/oovhdl
[10] http://www.vhdl.org/sid/
[11] http://www.cs.adelaide.edu.au/~petera/suave.html
[12] http://eis.informatik.uni-oldenburg.de/research/objective_vhdl.shtml
[13] http://www.superlog.org/
[14] A.Yamada, K.Nishida, R.Sakurai, A.Kay, T.Toshio, T.Kambe, “Hardware Synthesis with the Bach System”, ISCAS ’98, 1998
[15] T.Kambe, A.Yamada, K.Nishida, K.Okada, M.Ohnishi, A.Kay, P.Boca, V.Zammit, and T.Nomura, “A C-based synthesis system, Bach, and its application”, Proc. ASP-DAC 2001, pp.151-155, 2001
[16] Camposano and Wayne Wolf, “High Level VLSI Synthesis”, pp.127-151, Kluwer Academic Publishers, 1991
[17] K.Wakabayashi, “C-based Synthesis Experiences with a Behavior Synthesizer “Cyber””, Proc. of DATE’99, pp.390-393, 1999
[18] 若林，池上，大山，“C 言語ベースのシステムレベル合成・検証手法と開発事例”，第 13 回軽井沢ワークショップ，2000 年
[19] http://www.cleveldesign.com
[20] http://www.ics.uci.edu/~specc
[21] D.Gajski et al., “SpecC: Specification Language and Methodology”, Kluwer Academic Publishers, Publishers, 2000 木下常雄他訳 “SpecC：仕様記述言語と方法論”，CQ 出版，2000 年 12 月
[22] http://www.SpecC.org
[23] http://www.cynapps.com
[24] http://www.systemc.org
[25] D.Davis, “Using the Java Language and Environment for High-Level Circuit Design”, http://www.lavalogic.com/forge/java_wp.htm
[26] JavaTime Project Overview CAD Lunch, http://www-cad.eecs.berkeley.edu/~jimy/research/cadlunch.4.98/index.htm
[27] http://www.telelogic.com
[28] http://www-sop.inria.fr/meije/esterel/
[29] Rational Software Corporation, http://www.rational.com, http://www.rational.co.jp/
[30] http://www.omg.org
[31] http://www.sldl.org/, http://www.accellera.org
[32] Object Constraint Language, IBM, http://www-4.ibm.com/software/ad/standard/ocl.html