Vol. 36 No. 7　　情　報　処　理　　July 1995

解　説

3. 基本ソフトウェアとコデザイン†

安　浦　寛　人††

1. はじめに

1970年代に現われたマイクロプロセッサは，ソフトウェアによる柔軟で複雑な機能の実現を広く普及させ，多くのシステムがマイクロプロセッサとその上のソフトウェアとの組合せで実現されるようになった．1990年代に入って，ハードウェア記述言語と論理合成およびレイアウト合成によるハードウェアの設計自動化が実用化し，ハードウェアの設計もようやくソフトウェア並の柔軟さを獲得した．さらに，ASIC（特定用途向け集積回路）技術の進歩により，ハードウェアの実現のコスト（特に製造にかかる時間と費用）が大幅に削減されてきた．このような状況の下で，ソフトウェアとハードウェアを一体化した新しいディジタルシステム設計手法（Hardware/Software Codesign）の確立が重要な課題となっている[1)~4)]．

ハードウェア/ソフトウェア・コデザインの基本的な考え方は，「システム設計は，設計の各部の決定をできるだけ応用に近いところで行う」ことである．すなわち，システム設計者がソフトウェアのみでなくハードウェア（プロセッサを含むかどうかで違いがでる）も必要に応じて設計する手法である．ソフトウェアは，ハードウェアとして実現されたマイクロプロセッサのようなプログラム可能デバイス（以下プロセッサと呼ぶ）の上で動作する．ハードウェア/ソフトウェア・コデザインの環境ではベースとなるプロセッサの構造が可変であるか否かによって，状況が大きく違ってくる．プロセッサのアーキテクチャ，すなわちソフトウェアに対する命令体系が一定であれば，設計者はソフトウェアとプロセッサ以外のハードウェア部分との間での仕事の分担を決定し，それぞれをプログラムおよび回路として実現すればよい．一方，プロセッサまでが可変である場合は，プロセッサ自身の変更にともなってオペレーティングシステムやコンパイラなどの基本ソフトウェアの変更を行う必要がある．また，ソフトウェアの設計/評価環境も構築しなければならない．

通信機器や家電製品などに利用される組込み型のシステムでは，それぞれの処理に必要な資源や性能要求はまちまちであり，コストや消費電力の制約も考慮するとプロセッサの速度や機能に対する要求は用途に応じて多様に変化する．現在は，汎用のマイクロプロセッサのほか4-bit, 8-bit, 16-bitのMCU（Micro Controller Unit）やDSP（Digital Signal Processor）などの中からシステム設計者は用途に合わせてプロセッサを選ぶことができる[5)]．しかし，システムの多様化とコストや消費電力の制約が厳しくなる中，このような既設計のプロセッサだけでは十分に対応できない状況も起きている．これに対処するためには，用途に合わせたプロセッサ（特定用途向けプロセッサ）を使ったシステム設計手法の確立が必要となる．ここでは，このような特定用途向けプロセッサに対する基本ソフトウェアの設計技術の研究を中心に紹介する．特に，コンパイラの自動生成を中心に，基本ソフトウェアの自動設計の研究の現状と問題点を紹介する．

2.で組込みシステムを対象としたコデザインの問題点を明らかにし，3.でコンパイラやオペレーティングシステムをコデザインの中でどのように取り扱うべきかを議論する．4.では具体的な例としてコンパイラの自動生成技術を利用したコデザイン手法を紹介する．5.では，ソフトウェアとハードウェアを合わせたシステム全体の性能評価を

† Basic Software Codesign by Hiroto YASUURA (Department of Information Systems, Interdisciplinary Graduate School of Engineering Science, Kyushu University).
†† 九州大学大学院総合理工学研究科情報システム学専攻

設計の初期の段階でどのように行うかを議論する．

2. 組込みシステムとコデザイン

現在までに提案されているハードウェア/ソフトウェア・コデザインの手法としては，各種のシステムの制御部分として利用される組込みシステムに関するものが多い．特に，米国のハードウェアの自動設計の研究を進めてきたグループにおいては，既存の標準的なプロセッサを利用し，ソフトウェアの一部をハードウェアで実現すると共にハードウェアとソフトウェアのインタフェースを作成しようとする研究が主流である[1)~4)]．また，多くの場合に，システムの仕様が設計前に完全に定義されていることを前提とした設計フローが仮定されている．一方，今井らのグループやDespainらのグループは，応用に適したアーキテクチャのプロセッサを設計する方法でのハードウェア/ソフトウェア・コデザインの確立を目指している[6)~8)]．

米国を中心に議論されている標準のプロセッサを利用するハードウェア/ソフトウェア・コデザイン手法は，プロセッサ設計とシステム設計が基本的に異なる企業で行われるという米国の産業形態を前提とする考え方である．一方，我が国には，集積回路から大規模な産業用システムや家電製品までの設計/生産を行っている総合電機メーカが少なくない．これらのメーカでは，社内でマイクロプロセッサを生産し，それを使ったシステム製品を設計することが可能である．このような産業形態を前提とした場合，米国流のハードウェア/ソフトウェア・コデザインとは異なった設計手法が必要と考えられる．プロセッサ自身の仕様を，応用する組込みシステムに合わせて変更可能とするような，より柔軟な設計手法とそのための環境の確立が必要となると考えられる．すなわち，プロセッサの基本仕様の一部をユーザ（システム設計者）が指定できるようにし，ユーザの希望にあった高性能あるいは低コストまたは低消費電力のプロセッサを容易に設計できる環境を構築する方向での議論が重要である．

以下，本稿では，プロセッサ自身も変更されることを前提としたハードウェア/ソフトウェア・コデザインについて議論する．

組込みシステムの設計におけるソフトウェアの役割について考えてみよう．

1) 変更ができる機能の実現：消費者に直接利用され，製品のライフサイクルが短い家電製品や携帯用情報機器などは，市場に出荷された後にその仕様が部分的に変更されることが多い．このような変更をすべてハードウェアで吸収することは難しく，ソフトウェアによって対応することが必要となる．また，基本的な機能は同一にして付加機能を追加したり，低消費電力化のためにハードウェアを削減したりするような場合にもソフトウェアによる各種機能の実現が必要であり，実際のシステム開発でも広く利用されている．システム内で利用されるプロセッサに対しては，コンパイラやオペレーティングシステム，関数ライブラリなどの基本ソフトウェアやプログラム開発環境が必要となる．このような基本ソフトウェアや開発環境の充実度がプロセッサの選択に大きく影響している．応用に応じて専用のプロセッサを利用する場合には，これらの基本ソフトウェアや開発環境の整備をどうするかが大きな問題である．

2) システムの仕様の記述：ハードウェアとソフトウェアからなるシステム全体の仕様を統一的に記述するツール（言語）は，現在のところまだ実用的なものは存在しない．システムの機能を記述するために，プログラミング言語やその拡張版がしばしば用いられる．この場合，システムの機能を汎用のハードウェア上のソフトウェアとして実現していることになる．複雑なシステムの機能をソフトウェアの形で形式的に記述し，進歩の早いハードウェア部分を更新してもシステム全体の基本機能は変えないでおくという手法も重要である．しかし，現在のプログラミング言語やハードウェア記述言語では，ハードウェアによる実現に対する時間的/資源的な制約や処理の並列性などの設計者の細かな意図まで記述できる状況にはない．Wooらはオブジェクト指向型の機能記述言語（Object-Oriented Functional Specifications：OOFS）を仕様記述言語として用いる手法を提案している[4)]．この手法では，ハードウェアで実現される部分はハードウェア記述言語に，ソフトウェアで実現される部分はプログラミング言語に変換されてそれぞれ実現される．しかも，仕様記述自身をプログラミング言語記述へ変換して，シミ

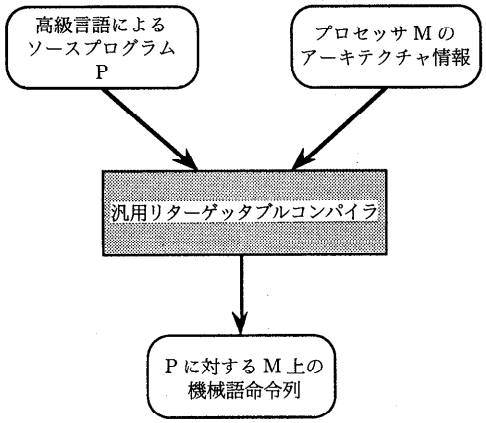
ュレーションを可能にしている．

このように組込みシステムの設計では，従来の汎用計算機を中心に発展してきたハードウェアとソフトウェアの関係とは違った観点から，言語や基本ソフトウェア，開発環境を考えていくことが要求される．

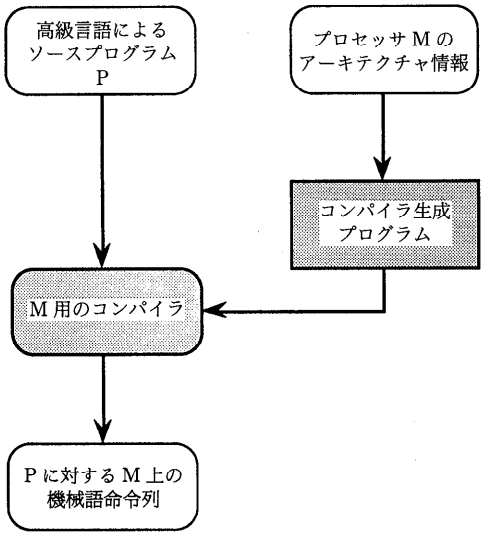
3. 基本ソフトウェアとコデザイン

ハードウェア/ソフトウェア・コデザインにおいて，プロセッサの命令セットやデータパスの構成を変化させる場合，これらのプロセッサアーキテクチャの変更に対応して，コンパイラ，オペレーティングシステム，各種ライブラリなどの基本ソフトウェアも変更することが必要となる．ここでは，ハードウェア/ソフトウェア・コデザインにおける基本ソフトウェアの位置づけと，今後解決が望まれる問題点を概観する．

コンパイラに関しては，高級言語で記述されたプログラムを異なるプロセッサに対応した機械語命令列に変換する技術が重要である．このような技術は，コンパイラ・コンパイラあるいはリターゲッタブルコンパイラとして1960年代から種々の研究開発が行われてきた[9]．使用する高級言語が決まっている場合，プロセッサのアーキテクチャの違いに応じて最適な機械語命令列を生成するコンパイラを作る問題となる．近年のRISCアーキテクチャの流行や各種DSPの開発などによって，アーキテクチャの種類が飛躍的に増え，このようなコンパイラ作成の技術も進歩し，実用的なコンパイラ作成支援ツールも普及している[10]．コンパイラが行う仕事のうち特に機械命令を割り当てるコード生成の部分は，論理合成の処理とも共通性があるため，近年は論理合成のアルゴリズムの研究者などもこの分野の研究を始めている．アーキテクチャを自動合成する高位合成とも密接な関係がある[11]．コンパイラの技術自身は，ハードウェアとソフトウェアの機能の分割やシステム全体のアーキテクチャを決定するときにも利用できる重要な要素技術である．今後，システム設計全体を視野に入れた新しいコンパイラ技術の研究の展開が望まれる．ハードウェア/ソフトウェア・コデザインの立場で考えると，プロセッサの変更に対応するコンパイラを実現する方法として，汎用のリターゲッタブルコンパイラを利用する方法とプロセッサごとに専用のコンパイラを生成する方法の二つの手法がある（**図-1**）．



1)　汎用のリターゲッタブルコンパイラを利用する方法



2)　プロセッサごとに専用のコンパイラを生成する方法

図-1　プロセッサの変更に対応するコンパイラを実現する方法

1)　汎用のリターゲッタブルコンパイラを利用する方法：高級言語プログラムと対象のプロセッサアーキテクチャ情報を入力として，そのプログラムの対象プロセッサ上での機械語命令列を生成する汎用リターゲッタブルコンパイラを開発して利用する．プロセッサごとにコンパイラを生成することはない．

2)　プロセッサごとに専用のコンパイラを生成する方法：プロセッサアーキテクチャ情報を入力として対象プロセッサに対応したコンパイラを自動生成するコンパイラ生成プログラムを開発し，

プロセッサごとのコンパイラを生成する．応用プログラムは，生成されたコンパイラを用いて機械語命令列へと変換される．

コンパイラ生成時間やコンパイル時間，対応できるアーキテクチャの範囲，設計者の負担などを考慮して，個々の組込みシステムに適した方法を選ぶ必要がある．

高機能な組込みシステムでは，マルチタスク処理やリアルタイム処理の実現が必要となる場合もあり，高い機能を持つオペレーティングシステムが使われることも多い．応用プログラムの立場からは，オペレーティングシステムの違いは，プロセッサアーキテクチャの違い以上に大きな問題となることも少なくない．組込みシステム用のオペレーティングシステムの標準化の努力も続けられている．現在のところ，ITRON[5]のように外部仕様がある程度統一されている場合でも，それぞれのプロセッサにオペレーティングシステムを個別に開発しなければならず，開発の工数はきわめて大きい．プロセッサのアーキテクチャを変更した場合に，オペレーティングシステムも追随して変更されるような技術は確立していない．今後の研究テーマである．さらに，従来の汎用計算機とともに発展してきたプロセッサ，オペレーティングシステム，応用プログラムという構図を組込みシステムにもこのまま踏襲するかどうかも検討すべき課題である．

オペレーティングシステムの機能の一部またはすべてをハードウェア化することも考えられる[17]．応用によってはレジスタやタイマなどの資源の必要量の上限がわかる場合もある．このような場合，必要な量の資源をハードウェアとして用意し，従来オペレーティングシステムで処理していた複雑な資源管理をやめることができる．このようなオペレーティングシステムとハードウェアの機能分担の再定義も解決すべき問題である．

このほか，各種ライブラリもプロセッサの変更に対応して変える必要がある．今後のマルチメディア対応機器などでは，データの符号化/復号化，各種通信プロトコルなど複雑な処理をライブラリ化して利用する技術も重要となると思われる．ソフトウェアのライブラリと同等の機能を実現するハードウェアマクロのライブラリの整備や利用法の確立も重要な課題である．

4. ハードウェアの言語記述からのコンパイラの生成

近年，プロセッサの設計に合わせてコンパイラを自動生成する手法の提案がいくつかなされている．

豊橋技術科学大学の今井らのグループは，フリーソフトウェアであるGNU CC[10]を利用して，プロセッサの命令セットの変更に対応したコンパイラの生成を行っている．プロセッサの変更の選択肢を制限し，ある種のパラメータ化を行うことで，コンパイラ生成に必要な情報を得やすくする工夫をしている[6),7)]．

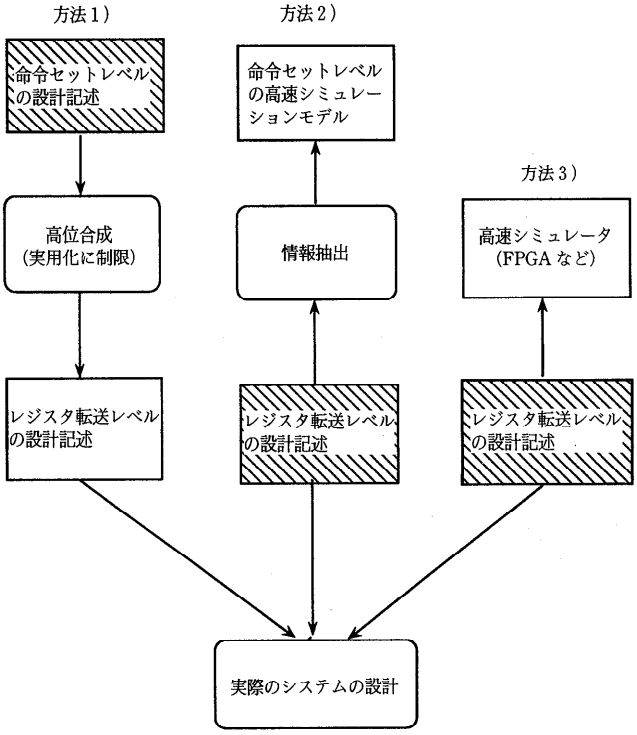
ドルトムント大学のMarwedelらは，汎用のリターゲッタブルコンパイラに与えるプロセッサのアーキテクチャ情報を，直接ハードウェア記述言語による記述から取り出す手法を開発し，プロトタイプシステムを構築している[12]．九州大学では，論理合成用に作られるプロセッサのハードウェア記述言語による記述からコンパイラ生成に必要な情報を自動的に抽出し，その情報を使って対応するコンパイラを自動生成する技術を開発している[13)~15)]．論理合成用の記述は，各クロックサイクルにおけるレジスタの間のデータの転送を記述するレジスタ転送レベルの記述であり，この記述から，より詳細な論理回路やレイアウトレベルの設計データを論理合成やレイアウト合成ツールにより自動的に生成できる．同じようにこの記述から，コンパイラ作成に必要なデータとして，各種レジスタのサイズや機能の情報，各命令ごとのデータのレジスタ間での移動と加工の状況などを抽出する．この作業は，レジスタ転送レベルでの記述をより抽象的な命令セットレベルの記述に変換することに対応する．簡単な命令セットを持つプロセッサに対して，そのハードウェア記述言語による記述から数分間でC言語用のクロスコンパイラを生成できる．この手法の最大の利点は，論理合成用の記述を変更するだけで，対応するコンパイラが自動的に生成できる点にある．

このような技術を利用すれば，プロセッサのアーキテクチャを種々変えてみて，応用プログラムをコンパイルしてシステムの性能を評価でき，プログラムの実行ステップ数まで考慮したシステム設計の最適化が可能となる．

5. ソフトウェアを含んだ性能評価

ハードウェア/ソフトウェア・コデザインでは，ハードウェアとソフトウェアの双方を含むシステム全体の性能評価を行う必要がある．現在，システムの性能評価は，シミュレーションによるのが一般的である．ハードウェア記述言語の記述を利用したレジスタ転送レベルでのシミュレーションでは，数十万ゲートのシステムに対して，実際の動作の 10^5～10^7 倍の計算時間を必要とする．実時間処理を要求されることの多い組込みシステムの場合には，このシミュレーション速度は絶望的である[16]．特に，プロセッサの動作のシミュレーションに大きな計算量を要する．命令セットレベルのシミュレーションモデルを利用すれば，10^2～10^3 倍の速度低下で済むため，実際の設計ではしばしばC言語などでかかれた命令セットレベルのシミュレーションモデルが利用されている．既存のプロセッサを利用する場合には，命令セットレベルのシミュレーションモデルは，あらかじめ用意したものを利用できるが，プロセッサの設計を変更する場合は，設計を変更するたびに新しいシミュレーションモデルが必要になる．現在のハードウェア合成技術では，レジスタ転送レベルより上位記述レベルで一般的にアーキテクチャの設計を行うのは現実的でない．このため，レジスタ転送レベルと命令セットレベルの2種類の記述が必要となり，両者の作成や整合性の維持などの問題が発生する．

このような問題を解決する手法としては，1) 命令セットレベルの記述でプロセッサの設計を行う方法，2) レジスタ転送レベルの記述から命令セットレベルのモデルを自動生成する方法，3) レジスタ転送レベルのシミュレーションを高速化



1) 命令セットレベルで設計を記述して下位レベルの合成を行う方法
2) レジスタ転送レベルで設計を記述して上位レベルのモデルを作る方法
3) レジスタ転送レベルで設計を記述してシミュレーションを高速化する方法
（斜線部がオリジナルの設計記述）

図-2 性能評価の方法

する方法などが考えられる（図-2参照）．

命令セットレベルでの設計を行う方法は，アーキテクチャの変化の幅を限定すれば現実的な解である．しかし，バスの構成や演算器の接続法など細かなアーキテクチャ上の工夫などにも対応できるようにすることは難しい．高位合成やアーキテクチャ合成の研究の成果が期待される[7),8)]．レジスタ転送レベルの記述から命令セットレベルのモデルを自動生成する方法は，記述の抽象化を行うことであり，コンパイラの生成のための情報を抽出する技術と基本的に同じである[18)]．シミュレーションを高速化する方法としては，FPGAなどのプログラム可能な回路を利用したエミュレーションが考えられる．この場合，回路規模やコストが問題となる．

将来的には，シミュレーションによらず，システムの精度良いモデル化を行って，設計の初期の段階から精度の高い性能見積ができる環境を構築することが望まれる[19)]．従来のハードウェアとソフトウェアの壁を取り払った新しい方法論の確立が望まれる．

6. おわりに

組込みシステムの設計においては，従来の汎用計算機を対象として発展してきたハードウェアとソフトウェアの関係，特に両者のインタフェースである基本ソフトウェアの在り方について本質的な再検討が必要である．これは，今後の情報関連機器の設計手法全体に，ひいては情報関係産業の産業形態にも大きく影響する問題である．ハードウェア/ソフトウェア・コデザインの種々の研究や提案は，このような計算機科学全体に対する問題提起であるとも考えることができる．

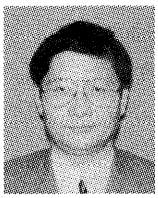
謝辞 コンパイラの生成やシステムシミュレーションについてご議論いただいた豊橋技術科学大学の今井正治教授，九州大学の赤星博輝氏，富山宏之氏に感謝します．オペレーティングシステムに関する部分は九州大学の谷口秀夫助教授との議論に拠るところが多いので，ここに謝意を表します．コデザインについてご議論いただいた，小泉寿男氏をはじめとする三菱電機(株)の諸氏ならびに広瀬文保氏をはじめとする富士通研究所の諸氏に感謝します．

参考文献

1) Gupta, R. K. and De Micheli, G. : Hardware-Software Cosynthesis for Digital Systems, IEEE Design and Test of Computers, Vol. 10, No. 3, pp. 29-41 (Sep. 1993).
2) Gupta, R. K., Coelho, C. N. Jr. and De Micheli, G. : Program Implementation Schemes for Hardware-Software System, IEEE Computer, Vol. 27, No. 1, pp. 48-55 (Jan. 1994).
3) Thomas, D. E., Adams, J. K. and Schmit, H. : A Model and Methodology for Hardware-Software Codesign, IEEE Design and Test of Computers, Vol. 10, No. 3, pp. 6-15 (Sep. 1993).
4) Woo, N.-S., Dunlop, A. E. and Wolf, W. : Codesign from Cospecification, IEEE Computer, Vol. 27, No. 1, pp. 42-47 (Jan. 1994).
5) 高田, 田丸, 工藤, 清水, 坪田 : ITRONサブプロジェクトの現状と展望—カーネル仕様とその実装技術, 情報処理, Vol. 35, No. 10 (Oct. 1994).
6) Imai, M., Alomary, A. Y., Sato, J. and Hikichi, N. : An Integrated Approach to Instruction Implementation Method Selection Problem, Proc. of EURO-DAC '92, pp. 106-111 (Sep. 1992).
7) Alomary, A. Y., Nakata, T., Honma, Y., Imai, M. and Hikichi, N. : An ASIP Instruction Set Optimization Algorithm with Functional Module Sharing Constraint, Proc. of IEEE/ACM International Conference on CAD ICCAD '93, pp. 526-532 (Nov. 1993).
8) Pyo, I., Su, C.-L., Huang, I.-J., Pan, K.-R., Koh, Y.-S., Chi-Ying, Chen, H.-T., Cheng, G., Liu, S., Wu, S. and Despain, A. M. : Application-driven Design Automation for Microprocessor Design, Proc. of 29th Design Automation Conference, pp. 512-517 (June 1992).
9) Ganapathi, M., Fischer, C. N. and Hennessy, J. L. : Retargetable Compiler Code Generation, Computing Surveys, Vol. 14, No. 4, pp. 573-592 (Dec. 1982).
10) Stallman, R. M. : Using and Porting GNU CC, Free Software Foundation, Inc. (1990).
11) Lin, Y.-L. and Wu, C.-H. : Tutorial on High-Level Synthesis, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E78-A, No. 3 (Mar. 1995).
12) Marwedel, P. : Tree-Based Mapping of Algorithms to Predefined Structure, Proc. of IEEE/ACM International Conference on CAD ICCAD '93, pp. 586-593 (Nov. 1993).
13) Akaboshi, H. and Yasuura, H. : COACH : A Computer Aided Design Tool for Computer Architects, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E76-A, No. 10, pp. 1760-

1769 (Oct. 1993).
14) Aakaboshi, H. and Yasuura, H. : Behavior Extraction of MPU from HDL Description, Proc. of 2nd Asia Pacific Conf. on Hardware Description Languages (APCHDL '94), pp. 67-74 (Oct. 1994).
15) Tomiyama, H., Akaboshi, H. and Yasuura, H. : Compiler Generator for Hardware/Software Codesign, Proc. of 2nd Asia Pacific Conf. on Hardware Description Languages (APCHDL '94), pp. 267-270 (Oct. 1994).
16) Shoji, M., Hirose, F. and Takayama, K. : VHDL Compiler of Behavioral Descriptions for Ultrahigh-speed Simulation, Proc. of 2nd Asia Pacific Conf. on Hardware Description Languages (APCHDL '94), pp. 85-88 (Oct. 1994).
17) Utama Andy, 板橋光義, 塩見彰睦, 今井正治, 仲野 巧: シリコン TRON の設計と評価, 電子情報通信学会 VLSI 設計技術研究会, 信学技報 VLD94-40, pp. 9-16 (1994).
18) 赤星博輝, 安浦寛人: 情報抽出技術を用いたアーキテクチャ評価用シミュレーションモデルの生成, 情報処理学会設計自動化研究会資料 73-1, pp. 1-8 (1995).
19) Li, Y-TS. and Malik, S.: Performance Analysis of Embedded Software Using Implicit Path Enumeration, Proc. of 32nd Design Automation Conference, pp. 456-461 (June 1995).

(平成7年5月9日受付)



安浦 寛人(正会員)

昭和51年京都大学工学部情報工学科卒業. 昭和53年京都大学工学研究科修士課程(情報工学専攻)修了. 昭和55年より同大工学部助手. 同大工学部電子工学科助教授を経て, 平成3年より九州大学大学院総合理工学研究科情報システム学専攻教授. VLSI システムの設計手法と CAD の研究およびハードウェアアルゴリズムの研究に従事. 昭和57年電子通信学会学術奨励賞, 昭和63年および平成6年電子情報通信学会論文賞, 平成4年情報処理学会論文賞, 平成5年情報処理学会坂井記念特別賞および Best Author 賞をそれぞれ受賞. 電子情報通信学会, IEEE, ACM, EATCS など各会員.