Vol. 35 No. 8　　情　報　処　理　　Aug. 1994

解　説

論理設計の形式的検証

4. プロセス代数を用いた形式的検証†

高　原　厚††

1. はじめに

設計検証では検証対象を形式的にモデル化する手法が必要である．レジスタ・トランスファ・レベル (RTL)，論理レベルでは，従来，状態遷移機械，ブール代数などを形式的モデルとして採用している．しかし，RTL よりも上位の設計レベルでは実現方法の詳細が明確ではないことから，より抽象度の高い形式的なモデルを用いることが望ましい[6]．本稿では，RTL より上位の設計レベルにおける形式的検証手法として，プロセス代数を用いる手法について述べる．

R. Milner の CCS (A Calculus of Communicating Systems)[10]，C. A. R. Hoare の CSP (Communicating Sequential Processes)[5] に代表されるプロセス代数 (process algebra)[17] は，ソフトウェアやハードウェアにより実現される並行システムの動作を形式的にモデル化する手法である．プロセス代数は，システムの動作をモジュール間の通信動作として表現することを特徴としている．モジュールのインタフェース間の動作を簡潔に表すことができる．さらに，並行動作についての厳密な意味を与える枠組を備えており，システムの並行動作を形式的に解析することが可能である．そのため，プロセス代数を用いた検証手法は，複数のモジュールを結合して構成される並行システムのインタフェースの検証に有効である．

以下では，プロセス代数の概念をハードウェアのモデル化を例にとり説明した後，設計検証とプロセス代数の関係，及び，提案されているプロセス代数を用いた形式的設計検証手法について述べる．

† Formal Design Verification Methods Based on Process Algebra by Atsushi TAKAHARA (NTT LSI Laboratories).
†† NTT LSI 研究所

2. プロセス代数

2.1 プロセス代数によるハードウェアのモデル化

プロセス代数における「プロセス」とは，外界と通信動作を行うシステムと考えることができる．プロセス代数によりハードウェアをモデル化する場合，ハードウェアの各モジュールはプロセスに対応づけられる．また，モジュールがもつ端子は外部と通信を行うプロセスのポートに対応づけられる．ハードウェアの動作は，プロセスのポートの動作順序を示すことにより表現される．プロセス代数で示される時間関係はポートの動作順序により表現される離散的なものである．

ハンドシェイクによりデータの転送を行う二つのモジュール (*Send*, *Rec*) をプロセス代数的に表現することを考えてみよう(**図-1**)．プロセス代数では，モジュール *Send*, *Rec* をそれぞれ独立したプロセスとしてモデル化する．これらのモジュールの端子 (*in*, $\overline{reqS}$ など) は各プロセスがもつポートとして表現され，*Send* の動作は式(1)に示されるポートの動作順序として表現される．

$$in \rightarrow \overline{reqS} \rightarrow ackS \rightarrow \overline{dataS} \qquad (1)$$

これは，入力 *in* において値を受けとり，出力 $\overline{reqS}$ から値を外部へ渡すという動作の系列を示している．同様に，*Rec* の動作は，

$$reqR \rightarrow \overline{ackR} \rightarrow dataR \rightarrow \overline{out} \qquad (2)$$

というポートの動作順序として表現される．

次に，*Send* と *Rec* の端子を図-1 に示すように接続して構成されるシステム T の動作を考える．プロセス代数では，接続された端子に相当するポートは接続された相手と同期をとり動作する．*Send* と *Rec* を接続したモジュール T の動作をポートの動作順序で示すと式(3)となる．

$$in \rightarrow (\overline{reqS}, reqR) \rightarrow (ackS, \overline{ackR})$$

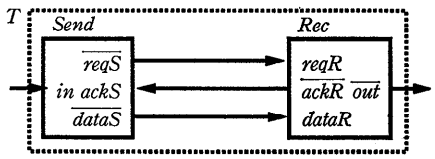

図-1 データ転送回路

$$\rightarrow(\overline{dataS}, dataR)\rightarrow\overline{out} \qquad (3)$$

T では，$Send$ のポート in は接続されていないので最初に動作可能である．これに対し，Rec のポート $reqR$ は，$Send$ のポート $\overline{reqS}$ が動作可能となって初めて動作を行う．

このように，通信を行うポートの動作によりシステムの動作をモデル化することがプロセス代数の基本である．

2.2 形式的表現

前節で示したハードウェアのプロセス代数によるモデル化を形式的に表現する手法として CCS に基づく形式的表現方法を紹介する．

CCS では，$Send$, Rec がそれぞれプロセスとして表現される．各プロセスのポートは，名前 (label) と入出力の属性 (A が入力，$\bar{A}$ が出力を示す．) により表現される．ポートの動作順序は，ポートの表現を '.' で接続して表す．$Send$ の CCS による表現は式(4)となる．

$$Send\Leftarrow in.\ \overline{reqS}.\ ackS.\ \overline{dataS}.\ Send \qquad (4)$$

ここで，最後に示した $Send$ は再帰を意味し，ポートの動作の系列が繰り返されることを示している．同様に，Rec の動作は式(5)で表現される．

$$Rec\Leftarrow reqR.\ \overline{ackR}.\ dataR.\ \overline{out}.\ Rec \qquad (5)$$

プロセスに複数のポートの動作の系列が存在する場合には，選択的な動作を表す '+' を用いてこれを表現する．例として，入力 in に与えられたデータを二つの出力先 ($\overline{outA}$, $\overline{outB}$) に記号 s により切り替えて出力するスイッチ Sw のモデル化を考える．この動作を表すためには入力 s が動作する場合と，入力 in が動作する場合の二つの動作が選択的に行われることを示す必要がある．この動作は，式(6)，(7)のように表現できる．

$$Sw \Leftarrow in.\ \overline{outA}.\ Sw + s.\ Sw1 \qquad (6)$$

$$Sw1\Leftarrow in.\ \overline{outB}.\ Sw1 + s.\ Sw \qquad (7)$$

式(6)は，入力データが $outA$ へ出力される場合と，s が動作することによりスイッチの出力先が変化する動作が選択的に行われることを示している．この選択(+)の表現を用いて，非決定的な動作を表現することも可能である．

CCS では，複数のプロセスを組み合わせてさらに複雑なプロセスを表現するための操作として，ポートの接続，プロセスの合成，ポートの隠蔽という操作が用意されている．

CCS では，異なるプロセスで同じ名前をもつポートがある場合，それらが接続されていることを示す．リラベリング (relabeling '/') とはポートの名前を変更する操作であり，異なるプロセス間のポートを接続するための操作である．$P[p1'/p1, p2'/p2, \ldots]$ は，プロセス P のポート $p1$ を $p1'$, $p2$ を $p2'\ldots$ に変更することを示している．リラベリングは $Send$, Rec として定義されたモジュールのインスタンスを生成する操作と考えることもできる．図-1 に示されたシステム T は，式(4)，(5)で定義されたプロセス Rec, $Send$ の対応するポートを接続して構成されたシステムである．これらは，リラベリングを用いて式(8)，(9)のように表現できる．

$$Send_{Impl}\Leftarrow Send[reqS/req, ackS/ack, dataS/data] = in.\ \overline{req}.\ ack.\ \overline{data}.\ Send_{Impl} \qquad (8)$$

$$Rec_{Impl}\Leftarrow Rec[reqR/req, ackR/ack, dataR/data] = req.\ \overline{ack}.\ data.\ \overline{out}.\ Rec_{Impl} \qquad (9)$$

次に，これらの二つのモジュールから構成される T を表現する．CCS では，合成 (composition '|') という操作で，並行に動作する複数のプロセスから構成される新たなプロセスを生成できる．

$$T\Leftarrow(Send_{Impl}|Rec_{Impl}) \qquad (10)$$

式(10)は，$Send_{Impl}$ と Rec_{Impl} で同じ名前のポートが接続されて互いに同期をとって動作するプロセスを表す．これは，図-1 で示された T を表現している．

さらに，ポートの動作を隠蔽することにより，プロセスの動作を抽象化する操作がある．CCS ではこの操作を制限 (restriction) と呼び，隠蔽するポート名を '\' の後に指定することで表現する．

$$T\Leftarrow(Send_{Impl}|Rec_{Impl})\backslash req\backslash ack\backslash data) \qquad (11)$$

この指定では，モジュールの内部のポートを隠蔽しており，外部のポートの動作のみに注目した T の動作を示している．

2.3 プロセス動作の追跡

プロセス代数によりモデル化されたシステムの動作は，プロセスのポートがどのような順序で動作したかを追跡することにより示せる．これは，ポートの動作順序をグラフ（遷移グラフ）として表すと理解しやすい*．式(6),(7)の動作をグラフとして表したものが**図-2** である．図-2 では，ポートの名前をもつ各枝はその対応するポートが動作可能であることを示す．また，ノードから複数の枝がある場合には，複数のポートが動作可能であることを示す．

合成により接続された複数のプロセスの動作は，展開規則 (expansion law) により決定される．これは，直観的には合成されたプロセスのポートの中で，接続相手のないポートは任意に動作可能であるが，接続されたポートは，その両者が動作可能となる場合に限り動作するという規則である．ただし，複数のポートが動作可能である場合には，そのうちの任意の一つのポートが動作を行う．式(11)で表現されるプロセス T では，リラベリングを行った $Send_{Impl}$ (式(8)), Rec_{Impl} (式(9))において，その表現の最初のポートから動作が行われる．これらには，途中で選択的な動作がないので式(3)で示した順でポートが動作する．この様子を遷移グラフとして示したものが**図-3** である．ここで，T において制限されているポートの動作は外部からは認識することができない．この制限されたポートの動作を記号 τ で表現し，システムの外部からは特定できない動作がシステム内で行われたことを示す．

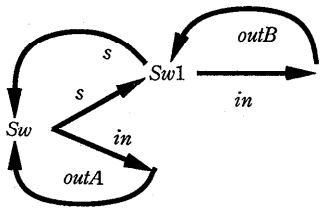

図-2　遷移グラフ

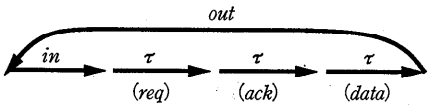

図-3　通信動作の解析

* 一般には，CCS により表されたプロセスのポートの動作順序を有限のグラフとして表現することはできない．ただし，再帰の動作を限定すれば有限に表すことができる．

3. プロセス代数による設計検証

3.1 検証手順

プロセス代数を用いた設計検証では，仕様を表すプロセスと，実現されたハードウェアを表すプロセスが同じ動作をするときに仕様が実現を満たしていることを示している．

図-1 のシステムの仕様 T_{Spec} は，in に与えられたデータを $\overline{out}$ に伝達することであるので，式(12)のように表すことができる．

$$T_{Spec} \Leftarrow in.\overline{out}.T_{Spec} \qquad (12)$$

設計検証は，この T_{Spec} と式(11)の T の動作が等しいことを示すことである．しかし，T_{Spec} と T の動作を比べると両者の動作は一致しない．なぜなら，T では ack などの内部的な動作が示されているが，T_{Spec} では，この動作に対応する動作が示されていないからである．

仕様が実現までを規定していないという立場で考えた場合，仕様に示されたポートの動作である「in の動作に引き続き out が動作すればよい」ということが示されれば正しく実現されていることになる．これは，システム T の内部的動作 (τ) を考えずに，外部から見た動作として T_{Spec} と T の動作が等しいということである．実際，T の動作は図-3 に示された τ の動作を除くと，

$$T \Leftarrow in.\overline{out}.T \qquad (13)$$

となり式(12)と同じ動作になる．

さらに別の実現方法として，式(6),(7)で示されるスイッチを用いた実現方法 T_{new} (**図-4**) を考える．このシステムの $Control$, MPX は次の動作を行うモジュールである．

$$MPX \Leftarrow inA.\overline{out}.MPX + inB.\overline{out}.MPX \qquad (14)$$

$$Control \Leftarrow \overline{Ctrl}.Control \qquad (15)$$

これらを接続したシステム T_{new} の動作は，

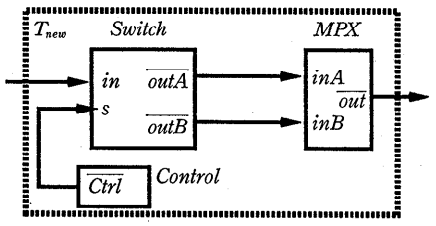

図-4　通信システム T_{new}

$$T_{new} \Leftarrow (Sw[s/ctrl] \mid Control \mid MPX[inA/outA, inB/outB]) \backslash outA \backslash outB \backslash ctrl \quad (16)$$

と表され，この動作の遷移グラフは**図-5** となる．図-5 において，τ を除いた動作は *in*, *out* というポートの動作が順に繰り返されるものであり，T_{new} と T_{spec} の動作は一致することが分かる．このように，τ による遷移を除いて，外界とのやり取りを行うポート *in*, *out* の動作に注目したとき，T_{spec}, T, T_{new} は等価であり，T, T_{new} は仕様を満たした実現となっていることが示される．CCS ではこのような等価性を弱等価 (weakly equivalent) と呼び，式(17)のように≈で表す．

$$T_{Spec} \approx T \approx T_{new} \quad (17)$$

このように，プロセス代数における設計検証は，

1. 実現されたハードウェアの動作を，仕様に示されるポートのみの動作へ制限，合成の操作により抽象化する．
2. 抽象化された実現を表現するプロセスと，仕様を表現するプロセスの動作との等価性(≈)を示す．

という手順で行うことができる．

3.2 プロセス代数における等価性

プロセス代数では，二つのプロセス A, B の等価性を，A において α という動作が可能ならば，B においても α という動作が可能であり，その動作の後，二つのプロセスが遷移した状態でも同じことが成り立つことと定義する．この関係は，互いが互いを模倣できるという意味で双模倣 (bisimulation)[10),18)] とも呼ばれる．**図-6** に示す遷移グラフ(a)，(b)では，どのポートが動作した後においても，常に両者が同じ動作が可能な状態に遷移するため双模倣の関係にある．これに対し，(c)，(d)では，両者でポート a が動作した後に，(c)ではポート b，及び，c が動作できるのに対し，(d)においては b，または，c のどちらかのみしか動作できないため，双模倣の関係にはない．

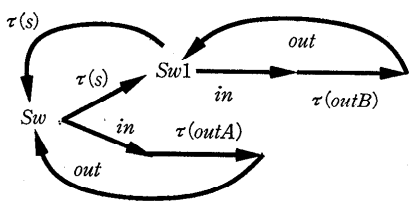

図-5 T_{new} に対する遷移グラフ

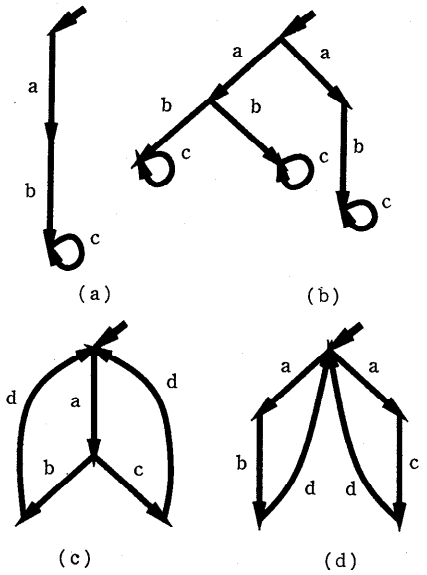

図-6 双模倣関係

双模倣の関係を満たすプロセスを等価なプロセスとする考え方では，τ の動作も他のポートの動作と同様に扱われるため，式(17)の等価な関係は成立しない．そこで，τ の動作を除いた等価性を次のように定める．これは，弱双模倣 (weak bisimulation) と呼ばれる関係であり，これに対し，前者は強双模倣 (strong bisimulation) と呼ばれる．弱双模倣は強双模倣においてポートの動作を α としていたのに対し，$\tau^* \alpha \tau^{*\star}$ という動作について双模倣が成立するという関係である．この弱双模倣の関係は，仕様で示される動作についてのみ注目しその動作が実現されたハードウェアについて満たされることを示していることになる．一方，強双模倣の関係にある仕様，実現は，それらの間に一対一の対応関係が明確につくことを示している．

双模倣に基づくプロセス代数の等価性は，オートマトンの等価性と異なることに注意されたい．オートマトンの等価性は，それが受理する言語が同じであることと定義される．図-6 (c), (d) を，ポートの名前を表すアルファベットによる言語を受理するオートマトンとして考えた場合，両

☆ これはポートの動作を正規表現として表している．ポート α の動作が，その前後に 0 回以上の内部的な動作をともなうことを示している．

者の受理する言語は同じであり等価なオートマトンと考えることができる．しかし，前述したように双模倣の考えではこれらの動作は等価ではない．オートマトンの等価性がアルファベットの系列としての等価性であるのに対し，双模倣では一つの動作ごとに常に等価な状態を経ながら二つのプロセスが動作することを要請する意味で，より粒度の細かい等価性であるといえる．

ここで CCS と CSP の違いについて述べる．CCS と CSP は，プロセス動作の表現方法は類似しているが，両者の等価性の意味が異なる．CCS では同じ動作が可能なプロセスを等価なプロセスと定めたが，CSP では同じ入力に対して二つのプロセスが動作できない状態 (Dead Lock) になるとき，これらのプロセスが等価であるとしている．CSP において等価なプロセスは強双模倣においても等価であるが，逆は成り立たない[1]．

3.3 等価性判定法

プロセス代数の解析を計算機処理により行う手法が提案されている．プロセス代数の処理系は，数式の変形，及び，定理証明系を用いて実現することができる[7]．しかし，複雑なシステムをモデル化する場合，状態遷移機械の解析と同様に，状態数の爆発が容易に起こりうる．この問題に対し，記号モデル検査法[8] を応用して，効率的にプロセス代数の解析を行う手法が提案されている[4),3]．これは，ポートの動作によりプロセスの経る状態を遷移関係と考えプロセスの動作を解析するものである．分散スケジューラの例を用いた実験結果では，状態数 4.9×10^5，遷移数（ポートの動作した回数）3.8×10^6 の例において，このプロセスの動作を表すために必要な BDD (Binary Decision Diagram)[2] のノード数が 1943，これに対する仕様との双模倣の計算時間が 1672 秒という結果が示されている[3]．

4. プロセス代数を応用した設計検証手法

本章では，プロセス代数を応用，拡張した設計検証手法の例を示す．

プロセス代数に基づいたハードウェアの検証に適したモデルが提案されている．CCS では動作の同時性を示すことができないため，同期式回路として実現されるシステムを簡潔にモデル化できない．G. J. Milne らは，CIRCAL[9] という同時に複数の動作が起こることを許容するモデルを提案している．CIRCAL でのポートは通信動作のみを示すのではなく，ハードウェアにイベントが起こったことを示すものである．CIRCAL での設計検証は，ハードウェアの仕様 (D_{Spec}) とその実現された回路 (D_{Impl}) の間で，イベントの系列の等価性を示すことになる．さらに，CIRCAL では，システムの満たすべき部分的な仕様 ($D_{PartialSpec}$) が実現された回路で満たされていることを，

$$(D_{PartialSpec} | D_{Impl}) \backslash InternalPorts \approx D_{PartialSpec} \quad (18)$$

が成立することで検証する．CIRCAL の合成は，複数のプロセスで動作可能なポートは全て同時に動作することを示す．式(18)では，D_{Impl} で示されるイベントの発生順序が $D_{PartialSpec}$ で定義されたイベントの発生順序に束縛されると考える．もし，この動作が D_{Impl} において許容される場合，D_{Impl} も $D_{PartialSpec}$ と同じ動作を行うことになり，結果的に $D_{PartialSpec}$ と同じイベントの系列が生成されるということである．

上位レベルのハードウェアの仕様を検証する手法として，非同期的に表現されたプロセスを同期的に解釈するモデル，疑似非同期モデル (PAS モデル) が提案されている[14]．上位レベルの仕様では，システムの具体的な構成が示されるのではなく，外部仕様としての動作が示される．このような仕様は，外部から観測できるポートの動作として非同期的にモデル化することが自然である．しかし，実際に同期式回路として実現されるシステムを，非同期的に解釈し検証する場合より，同期的な解析を可能とした非同期モデルを用いるほうが簡潔に解析が行える．PAS モデルを用いた検証方式[13]では，ハードウェアの仕様として，外部からシステムへの動作と，その動作による期待値の組の集合として仕様を与える．この仕様と実現をモデル化したプロセスを合成したシステムが矛盾なく動作することを示すことで検証が行われる．この検証方法は，詳細な構造的対応関係を必要とせず構造的抽象度が異なる仕様，実現間の検証に有利な方法である．

実際の設計検証問題では，実時間を扱いたいという要求が多い．そこで，プロセス代数で時間を扱えるように拡張したものがいくつか提案されている[12]．提案されている手法は，ポートの動作に

対し遅延時間を付与し，従来のプロセス代数の枠組に時間に関するポートの動作規則を加えた拡張を行っている．

5. むすび

プロセス代数の概念とそれを用いたハードウェアのモデル化，検証手法について述べた．プロセス代数を用いたモデル化では，複数のモジュールにより構成されるシステムを合成として簡潔に表現でき，設計検証での異なる表現レベルの比較を制限という概念でその動作を抽象化し比較することができるという特徴をもつ．プロセス代数は設計検証のみならず，通信システムのプロトコルを実現するハードウェアの自動合成のためのモデルとして利用することができる[15]．プロセス代数の通信という概念は，ソフトウェア，ハードウェアにより実現されるシステム間のインタフェースのモデル化に応用することも可能である．また，近年注目されている非同期回路[16]を CSP を用いて検証する手法が報告されており[11]，非同期回路の解析への応用も考えられる．

参考文献

1) Brookes, S. D.: On the Relationship of CCS and CSP, in Goos, G. and Hartmanis, J. eds., *Lecture Notes in Computer Science 154*, pp. 83-96, Springer-Verlag (1983).
2) Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, **C-35** No. 8, pp. 677-691 (1986).
3) Camurati, P., Corno, F. and Prinetto, P.: Exploiting Symbolic Traversal Techniques for Efficient Process Algebra Manipulation, in *proc. of IFIP Conference on Hardware Description Languages and their Applications (CHDL '93)*, pp. 21-34 (1993).
4) Enders, R., Filkorn, T., Taubner, D.: Generating BDDs for Symbolic Model Checking in CCS, in *proc. of CAV '91 Computer-Aided Verification*, pp. 203-213 (1991).
5) Hoare, C.: Communicating Sequential Processes, *Communications ACM*, **21** No. 8, pp. 666-677 (1978).
6) Koomen, C. J.: *The Design of Communicating Systems: A System Engineering Approach*, Kluwer Academic Publishers (1991).
7) Lin, H.: PAM: A Process Algebra Manipulator, in *proc. of CAV '91 Computer-Aided Verification*, pp. 136-146 (1991).
8) McMillan, K. L.: *Symbolic Model Checking*, Kluwer Academic Publishers (1993).
9) Milne, G. J.: Simulation and Verification; Related Techniques for Hardware Analysis, in *proc. of IFIF Conference on Hardware Description Languages and their Applications (CHDL '85)*, pp. 404-417 (1985).
10) Milner, R.: *Communication and Concurrency*, Prentice Hall (1989).
11) Mishra, Y., Sherlekar, S. and Venkatesh, G.: Path Breaker: A Tool for the Optimal Design of Speed Independent Asynchronous Controllers in *proc. of Euro-DAC '92*, pp. 2-8 (1992).
12) Nicollin, X. and Sifakis, J.: An Overview and Synthesis on Timed Process Algebras, in *proc. of CAV '91 Computer-Aided Verification*, pp. 376- 398 (1991).
13) Takahara, A. and Nanya, T.: A Higher Level Hardware Design Verification, in *proc. of International Conference on Computer Design (ICCD '88)*, pp. 596-599 (1988).
14) 高原，南谷：擬似非同期システムモデル，情報処理学会論文誌，Vol. 30, No. 1, pp. 109-117 (Jan. 1989).
15) 小林，宮崎，星野：プロトコル仕様記述言語からのハードウェア合成手法，信学技報，**VLD92** No. 80, pp. 81-88 (1993).
16) 南谷：非同期式プロセッサー超高速 VLSI システムを目指して—，情報処理，Vol. 34, No. 1, pp. 72-80 (Jan. 1993).
17) 二木，富樫：形式仕様とプロセス代数，*bit*, **23** No. 11, pp. 1449-1464 (1991).
18) 富樫：プロセス代数と等価性（前編），*bit*, **23** No. 12, pp. 1642-1653 (1991).

（平成6年1月5日受付）

高原　厚（正会員）

1983年東京工業大学工学部情報工学科卒業．1985年同大学院修士課程修了．1988年同大学院博士課程修了．同年日本電信電話(株)入社．以来，同社 LSI 研究所にて，LSI の論理設計システムの研究開発に従事．工学博士．電子情報通信学会，IEEE, ACM, USENIX 各会員．