研究課題別評価

1　研究課題名：ハードウェアアルゴリズムの進化的合成に関する研究

2　研究者氏名：本間　尚文

3　研究の狙い：

近年，マルチメディア信号処理やセキュリティ情報処理に要求される演算能力は増加の一途をたどっており，多種多様なデータパスの設計が必要とされている．データパスの大部分を占める算術演算回路の性能は，デバイスレベルや論理レベルでの最適化のみならず，算術演算のハードウェアアルゴリズム（算術アルゴリズム）に大きく依存する．今後，VLSI システムに対する要求の著しい多様化に伴い，最適な算術アルゴリズムを設計する必要性はますます高まると予想される．しかし，現在の EDA（Electronic Design Automation）技術は論理回路の記述や検証を基本として発展しており，算術アルゴリズムの設計に対して十分な設計環境が整っていない．従来のハードウェア記述言語（HDL: Hardware Description Language）では論理式によって回路を記述するため，算術演算を基本とする算術アルゴリズムを直接記述することは難しい．また，2進数系以外の算術アルゴリズムを記述・検証するためには，2値論理信号に基づいた低水準の構造が必要となる．論理式の簡単化に基づく汎用の論理合成では，算術アルゴリズムの自動合成までは原理的に困難である．

本研究では，このような設計問題の本質的ブレークスルーのためには，データパス設計を本質的なアルゴリズムレベルで行う新しい設計パラダイムが必須であるとの観点から，算術アルゴリズムの記述・検証・合成技術の開発をおこなった．その中心となる着想が，2進数と非2進数を統合した数系・数式による算術アルゴリズムの表現手法である．提案する表現手法は，①算術アルゴリズムを整数方程式により形式的に記述可能，②非2進数系を含む任意の重み数系を記述可能，③アルゴリズムの正当性を数式処理等により静的に検証可能，④正当性の証明された算術アルゴリズムを従来の論理式に変換可能などの特長を有する．さらに，高信頼な算術アルゴリズムライブラリの構築や，算術アルゴリズムの自動合成への応用が期待できる．

4　研究成果：

本研究では，2進数と非2進数を統合した算術アルゴリズムの高水準な設計技術を実現するため，以下の3項目について研究を実施した．

① 算術アルゴリズム記述言語およびその言語処理系
② 加算アルゴリズムの統一的なデータ構造
③ 進化的グラフ生成手法に基づく算術アルゴリズムの自動合成

以下ではそれぞれについて説明する．

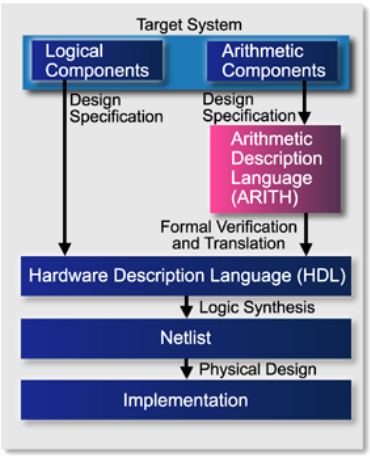
① 算術アルゴリズム記述言語およびその言語処理系

本研究で提案する表現手法に基づく算術アルゴリズム記述言語 ARITH とその言語処理系を開発した（図1）.

ARITH では，算術演算回路の構造を数系と整数方程式によって階層的に表現する．ARITH で扱う数系とは，重みベクトルと桁集合ベクトルの二項組によって与えられる任意の重み数系である．例えば，従来の符号なし2進数系は，重みを2のべき乗，桁集合を{0,1}で与えられる．ARITH で用いる信号は，全て整数信号であり，属性として数系と取り得る桁の範囲をもつ．一方，整数方程式は整数信号および整数定数を加算，減算，乗算で結合して得られる式である．本研究で開発した言語仕様では，各種加算器，加算器ネットワーク，乗算器，積和演算器およびそれらの複合データパスを設計することができる．

提案する ARITH 記述は，数系を記述するための型定義ブロックと算術アルゴリズムの機能と内部構造を記述するためのモジュール定義ブロックから構成される．型定義ブロックでは，重みベクトルと桁集合ベクトルを宣言することで，非2進数系を含む任意の重み数系を定義する．一方，モジュール定義ブロックでは，型定義ブロックで定義された数系と整数方程式を用いて算術アルゴリズムの機能を記述する．以上の手法で記述された算術アルゴリズムは，シミュレーションなどを用いることなく，形式的に検証される．本研究で開発した ARITH 言語処理系では，従来の検証手法に加えて，数式処理を用いた検証を可能とする．それらの検証手法を組み合わせることにより，ARITH 記述を高速に検証できることを実験的に確認している．

本研究では，ARITH の応用として算術演算器モジュールジェネレータを開発した． ARITH で記述された算術アルゴリズムをデータ形式として用いることで，ジェネレータ内部の演算器ライブラリの信頼性を向上させることができる．開発したモジュールジェネレータは，(i) ARITH/HDL コードジェネレータ，(ii) ARITH 処理系，および(iii) BDD 等価性チェッカーから構成される（図2）．まず，ARITH コードジェネレータは，与えられた設計仕様に従い ARITH 記述を生成する．次に，生成した算術アルゴリズムに応じて，ARITH 処理系および BDD 等価性チェッカーにより ARITH 記述を形式的に検証する．最終的に，それぞれの検証結果を合わせることで，アルゴリズムレベルで完全に機能の保証された演算器モジュールを生成する．



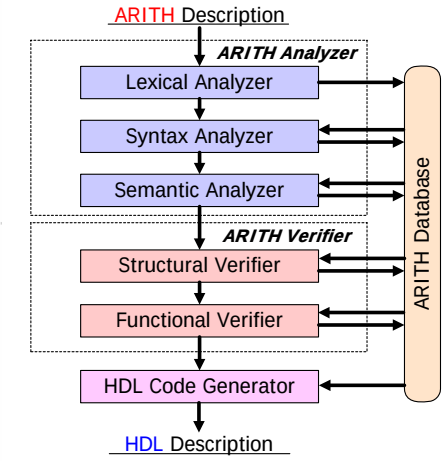


図1左：ARITH を用いたシステム設計フロー（算術アルゴリズムは ARITH により記述され，数学的な正当性を検証される．その後，HDL コードに変換され，システムに組み込まれる）
図1右：ARITH 言語処理系の構成

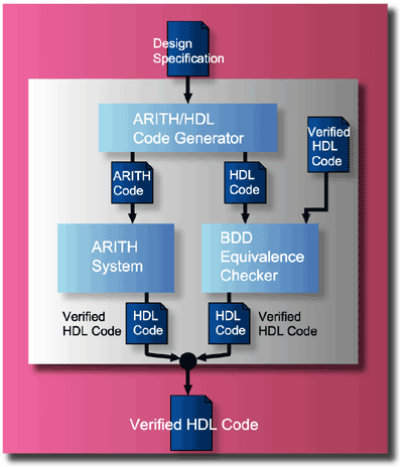
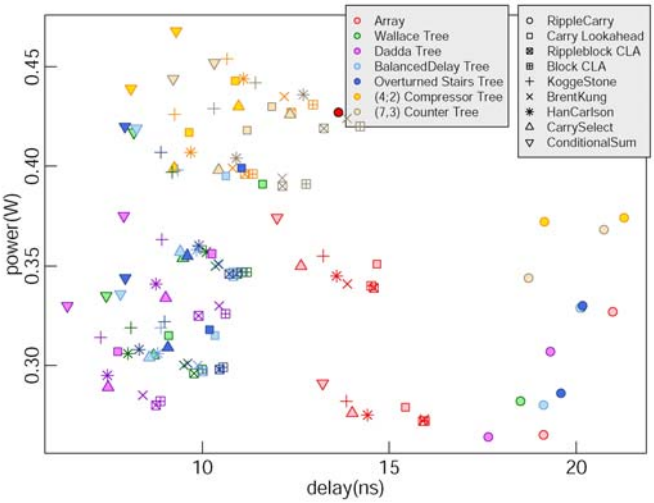


図2：ARITH を用いた乗算器モジュールジェネレータ
入出力の数表現2種類（Unsigned, Two's complement），部分積生成器2種類（PPGs with/without Radix-4 Booth encoding），部分積加算器8種類（Array, Wallace tree, Balanced delay tree, Overturned-stairs tree, Dadda tree, (4;2) compressor tree, (7,3) counter tree, Redundant binary addition tree），最終段加算器11種類（Ripple carry adder, Carry look-ahead adder, Ripple-block carry look-ahead adder, Block carry look-ahead adder, Kogge-Stone adder, Brent-Kung adder, Han-Carlson adder, Conditional sum adder, Carry select adder, Carry skip adder(2 type)）の組み合わせが選択できる.



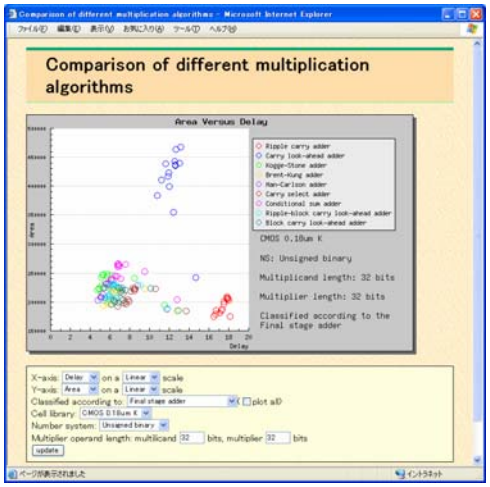


図3左：乗算器モジュールジェネレータにより生成された乗算器の性能
（Rohm 0.35μm CMOS プロセスの例）
図3右：Web 上に公開された性能評価システムのインタフェース

本研究で開発したジェネレータは，並列乗算アルゴリズムを生成の対象とする．ここでは，乗算アルゴリズムを部分積生成アルゴリズム，部分積加算アルゴリズム，最終段加算アルゴリズムの3つに分け，それぞれの算術アルゴリズムを組み合わせることで設計仕様を決定する．各種の数系（非2進数系を含む）に基づく算術アルゴリズムを ARITH によって系統的にライブラリ化しており，352種類の乗算器を任意の語長で生成可能である．生成される乗算器モジュールは，ARITH 処理系の形式的な検証により，アルゴリズムレベルでの機能を完全に保証される．

現在，Web 上(http://www.aoki.ecei.tohoku.ac.jp/arith/mg/)にて，開発したジェネレータを公開している．利用者は，設計仕様の入力フォームから，入力信号幅，数系および乗算アルゴリズムを指定することで，所望の乗算器を得ることができる．また同システムでは生成されたモジュールの性能評価を提供しており（図3），これをアルゴリズム選択の指標として参照することができる．公開したシステムには，2004年秋以降8000件を越える利用がある．

② 加算アルゴリズムの統一的データ構造

加算アルゴリズムを統一的に表現するデータ構造として Counter Tree Diagram（CTD）を開発した.

一般に，算術演算回路は幾つもの単純な基本演算回路から構成され，その中心となるのが加算器もしくはカウンタと呼ばれる回路である．現在までにさまざまな高速加算アルゴリズムが独自の手法により開発されているが，種々の数系に基づく加算アルゴリズムを統一的に取り扱う設計理論は存在しなかった．提案する CTD は，加算機能を抽象化したカウンタノードから構成され，さまざまな抽象度レベルにおける加算器・カウンタのネットワークを表現する．例えば，CTD により，冗長 2 進加算器，Signed-Digit 加算器，Positive-Digit 加算器，Binary Carry-Save 加算器およびその加算器やカウンタによって構成される加算木全般を取り扱うことが可能である.

CTD は，カウンタノードと呼ばれるノードの集合とノード間を結ぶ有向辺の集合からなる．カウンタノードは，加算機能（カウンタ機能）を抽象的に表現する．一方，有向辺は，オペランド同士の依存関係を表現する．すべての有向辺はオペランドの定義域として重みつき区間を持つ．CTD は，カウンタノードのネットワークとして，加算アルゴリズムをさまざまな抽象度レベルで表現する（図4）．分解・結合と呼ばれる操作により抽象度を自由に変更することで，さまざまな加算アルゴリズムの解析や設計に CTD を利用できる.

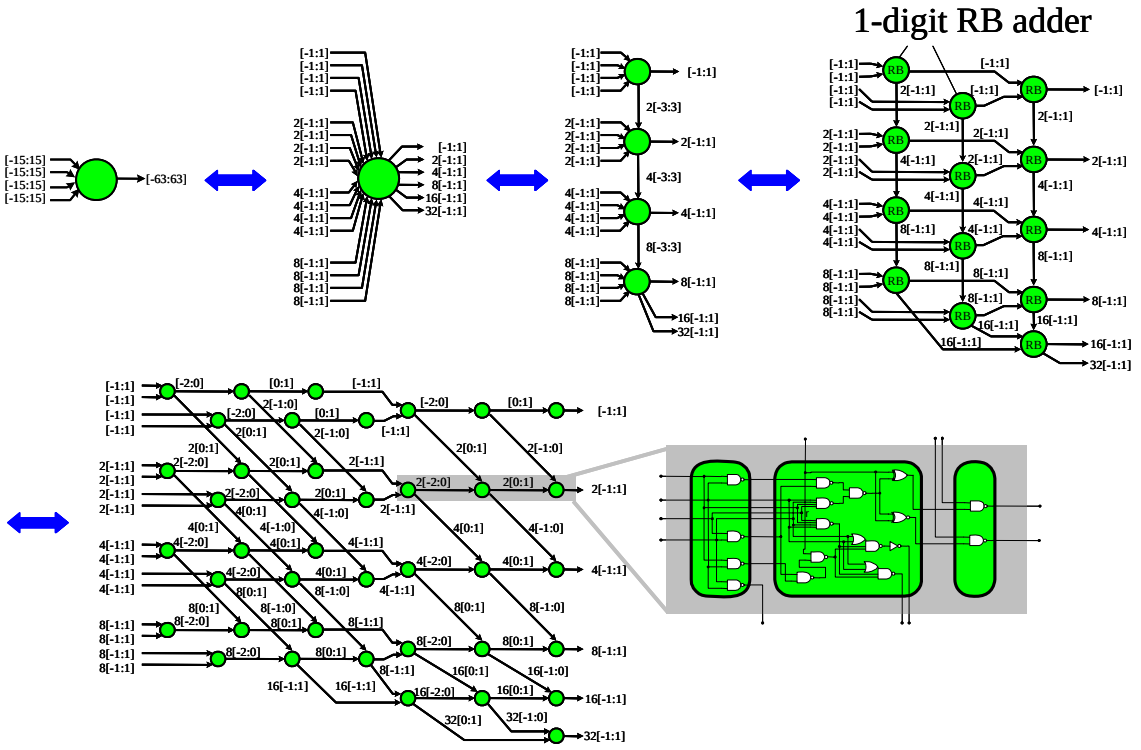
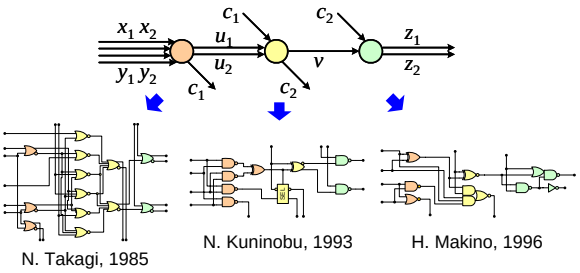


図4：Counter Tree Diagram による冗長加算アルゴリズムの表現
（冗長 2 進加算アルゴリズムの例）



		電力遅延積 (ns・mW)		遅延 (ns)		消費電力 (mW)	
		Hitachi	ROHM	Hitachi	ROHM	Hitachi	ROHM
		0.18	0.35	0.18	0.35	0.18	0.35
本研究で設計した回路		1.46	17.38	0.38	0.90	1.70	11.08
従来の回路	Takagi	2.35	26.62	0.56	1.26	1.98	13.58
	Kuninobu	2.09	24.52	0.50	1.29	1.95	13.52
	Makino	2.15	28.92	0.46	1.17	2.31	14.83

図5：Counter Tree Diagram による冗長加算器の解析とその最適設計への応用
（冗長 2 進加算器の例）

本研究では，CTD の応用として高速な冗長加算器の解析をおこなった．冗長数系を用いることで桁上げ伝搬の制限された高速な加算器を実現できることは広く知られている．しかし，その設計はこれまで冗長数系に関する専門的な知識を持った設計者により経験的に行われてきた．これに対して，CTD を用いることで，算術演算アルゴリズムに関する専門的な知識を用いることなく，冗長数系に基づく高速加算アルゴリズム（多段構造の CTD）を代数的に導出することができる．解析の結果，従来の冗長加算器構造は全て CTD の一つとして解釈できることが明らかになった（図5）．開発した解析手法は，任意の冗長加算アルゴリズムに適用が可能であり，今後の冗長加算器を設計する上でも有用であると考えられる．

さらに本研究では，例として冗長2進加算器を取り上げ，CTD に基づく冗長加算器の設計手法の有効性を確認した．冗長加算アルゴリズムを表現する CTD に対応する論理関数を得るには，まず，整数値をとる CTD 変数上での加算表（機能表）を決定する．次に，CTD 変数を 2 値に符号化し，機能表から真理値表（論理関数）に変換する．そのため，機能表と CTD 変数への符号化の組み合わせにより多様な回路構造が得られる．実験として，冗長2進加算器を網羅的に設計し，得られた加算器の性能を従来の加算器と比較した．その結果，機能表および CTD 変数の符号化を変更することにより，多様な回路構造を系統的に得られることを実証した．また，テクノロジに応じた最適化が可能となり，従来よりも30～40%程度性能の優れた構造が得られることを確認した（図5）．

③ 進化的グラフ生成手法に基づく算術アルゴリズムの自動合成

算術アルゴリズムの自動合成技術として進化的グラフ生成システム（EGG）のフレームワークを開発した．EGG は，本研究者が提案する回路合成に特化した進化的計算手法であり，回路構造を模したグラフによる個体モデルとその構造操作を有する．

本研究では，まず，単一コンピュータでの実装を前提としていた EGG のプロトタイプシステムを並列化して，複数のプロセッサ上で動作する並列 EGG 処理系を開発した．ここでいう並列 EGG は，いわゆるアイランドモデルのように，複数のクライアント（アイランド）上で，それぞれ異なるパラメータによる構造進化を行い，適切なタイミングで個体データを交換（移民）することにより，全体としてグラフ構造の多様性を維持しつつ協調的な進化を実現する．これにより，メモリおよび計算速度の観点から単一プロセッサ上では実装不可能であったような多数の個体を用いた進化プロセスが

実現され，生成される演算回路の質と収束速度を大幅に改善できた．

次に，並列 EGG システムの実現に特化した50ノードの PC クラスタを構築し，開発した並列 EGG を実装した．並列 EGG システムはゆるやかに協調する複数の EGG プロセスから構成されるため，プロセッサ間の通信バンド幅を低く抑えることが可能である．このため，商業ベースの off-the-shelf PC によって構成される PC クラスタ上における効率良い実装が可能である．開発した並列 EGG では，分散処理される各プロセス間の通信（個体集団の移民プロセス）に厳密なスケジューリングが不要なため，汎用のイーサネットとスイッチングハブを用いても十分な台数効果を達成できる．

構築した PC クラスタ上において，多様な演算回路の創発生成実験を実施した．CTD に基づく算術アルゴリズムを種々の抽象度レベルで合成し，EGG の有する本質的な探索能力を明らかにした．具体的には，トランジスタレベルでのアナログ演算回路やワードレベルでの FIR ディジタルフィルタ向け乗算器ブロックを合成対象とした．本研究では，それらの合成結果から EGG の動作について理論的考察をおこない，基本となるデータ構造および操作を定義したオブジェクト指向フレームワークを開発した．現在，Web 上にて，開発したフレームワークを広く公開している (http://www.aoki.ecei.tohoku.ac.jp/egg/)．

5 自己評価：

本研究では，算術アルゴリズムの高水準な設計環境を実現するため，その記述・検証・合成に関する基盤技術を開発した．最終的には，当初計画していた算術アルゴリズムの進化的合成の他にも，算術アルゴリズムの記述や検証において期待以上の大きな成果が得られた．開発した算術アルゴリズムの表現手法により，これまで個別に設計されてきた算術アルゴリズムを統一的に設計することが可能となる．また，提案する算術アルゴリズム記述言語 ARITH を用いて開発した乗算器モジュールジェネレータは，350種類を越える算術アルゴリズムに対応しており，世界でも他に類を見ない規模に達している．

一方で，現在提案する表現手法は，算術式で表現される算術アルゴリズムのみを対象としている．今後は，任意の論理演算を含む算術アルゴリズムへの拡張が課題となる．まず，ARITH の言語拡張では，算術演算と論理演算の混在表現を実現する必要がある．これまでの ARITH のデータ形式を用いて論理演算と算術演算を統一的に記述・検証するためには，論理演算を算術演算として取り扱う枠組みが必要不可欠である．この問題に対して，現在ブール代数特有のべき等律に着目することで論理演算を算術演算とみなす方式を検討している．次に，CTD の拡張では，論理演算に伴う制御信号などを取り扱う必要がある．これは，CTD 変数に媒介変数を導入することで実現可能であると予想される．

今後，拡張した ARITH や CTD を用いて，正当性を100%保証する記述で算術アルゴリズムをライブラリ化し，必要に応じてデータパス設計に利用することが可能になれば，従来の CMOS デバイスのみならず次世代デバイス（単電子デバイス，分子デバイス，スピントロニクスデバイス等）の演算回路設計環境を実現する上で重要なブレークスルーを与えると考えられる．

6　研究総括の見解：

本間研究者は、算術アルゴリズムの高水準な設計環境を実現するため、その記述・検証・合成に関する基盤技術を開発した。本研究の数系・数式表現により、これまで個別に設計されてきた算術アルゴリズムを統一的に設計することが可能となった。具体的には、算術アルゴリズム記述言語 ARITH およびその言語処理系の開発、加算アルゴリズムを統一的に表現するデータ構造として Counter Tree Diagram の開発、進化的グラフ生成手法に基づく算術アルゴリズムの自動合成などにおいて期待以上の大きな成果を挙げた。研究成果は乗算器の生成と性能評価など極めて実用的なものであり、Web 上で公開された算術演算器モジュールジェネレータには数多くの利用があり、多くの実績を残している。今後さらに研究を進め、大きく発展することを期待している。

7　主な論文等：

論文

(1). Naofumi Homma, Takafumi Aoki, and Tatsuo Higuchi, “Multiplier block synthesis using Evolutionary Graph Generation,” The IEEE Computer Society Press in the Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware, pp 79--82, June 2004.

(2). Naofumi Homma, Jun Sakiyama, Taihei Wakamatsu, Takafumi Aoki, and Tatsuo Higuchi, “A systematic approach for analyzing fast addition algorithms using Counter Tree Diagrams,” Proc. of the 2003 IEEE International Symposium on Circuits and Systems, pp. V-197--V-200, May 2004.

(3). 本間尚文, 崎山淳, 若松泰平, 青木孝文, 樋口龍雄, “Counter Tree Diagram に基づく冗長加算器の系統的設計手法 — 冗長 2 進加算器設計の例 —,” 情報処理学会論文誌, Vol. 45, No. 5, pp. 1279--1288, May 2004.

(4). Naofumi Homma, Masanori Natsui, Takafumi Aoki, and Tatsuo Higuchi, “VLSI circuit design using an object-oriented framework of Evolutionary Graph Generation system,” Proc. of 2003 Congress on Evolutionary Computation, pp. 115--122, December 2003.

(5). Jun Sakiyama, Naofumi Homma, Takafumi Aoki, and Tatsuo Higuchi, “Counter Tree Diagrams: A unified framework for analyzing fast addition algorithms,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E86-A, No. 12, pp. 3009--3019, December 2003.

特許

(1). 加算回路合成装置, 加算回路合成方法, 及び加算回路合成プログラム　特開 2005-250704.

受賞

(1). 第 7 回 LSI IP デザイン・アワード完成表彰部門　IP 賞受賞, May 2005.

招待講演等

（1）.Naofumi Homma, Yuki Watanabe, Kazuya Ishida, Takafumi Aoki, and Tatsuo Higuchi, “A multiplier module generator based on arithmetic description language,” Proc. of the IP Based SoC Design Conference & Exhibition, December 2005. （招待講演）

（2）.Takafumi Aoki, Naofumi Homma, and Tatsuo Higuchi, “Evolutionary synthesis of arithmetic circuit structures,” Artificial Intelligence in Logic Design, Edited by S. N. Yanushkevich, Kluwer Academic Publishers, pp. 39--72, 2004 (Reprinted from AI Review, Vol. 20, Nos. 3-4, 2003).（共著）

（3）.Naofumi Homma, Masanori Natsui, Takafumi Aoki, and Tatsuo Higuchi, “A graph-based approach for synthesizing arithmetic circuits,” Proc. of 13th International Workshop on Post-Binary ULSI Systems, pp. 25--32, May 2004.（招待講演）

（4）.Takafumi Aoki, Naofumi Homma, and Tatsuo Higuchi, “Evolutionary synthesis of arithmetic circuit structures,” Artificial Intelligence Review, Kluwer Academic Publishers, Vol. 20, Nos. 3-4, pp. 199--232, December 2003.（招待論文）