計算機代数に基づく算術演算回路の形式的設計に関する研究

氏名（本籍地）	渡邉（わたなべ） 裕樹（ゆうき）
学位の種類	博　士（情報科学）
学位記番号	情　博　第　434　号
学位授与年月日	平成21年　3月25日
学位授与の要件	学位規則第4条第1項該当
研究科、専攻	東北大学大学院情報科学研究科（博士課程）情報基礎科学専攻
学位論文題目	計算機代数に基づく算術演算回路の形式的設計に関する研究
論文審査委員	（主査）東北大学教授　青木　孝文
	東北大学教授　亀山　充隆　　東北大学教授　堀口　進

論文内容の要旨

第1章　緒言

近年，LSIシステムを搭載する機器は著しく増加しており，用途に応じて適切な算術演算回路を設計することが求められている．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．そのため，高性能なLSIシステムを実現することを目的として，これまでに数多くの算術アルゴリズムが考案されてきた．従来の2進数系にとらわれない多進数系や冗長数系に基づく算術アルゴリズムも考案されており，それらが高い性能を発揮することも示されている．

一方で，現在のLSIの設計において標準的に用いられるハードウェア記述言語（HDL: Hardware Description Language）は，論理回路の記述・合成を目的として開発されており，数系や数式を用いて表現される算術アルゴリズムを直接記述することはできない．また，算術演算回路は多入力・多出力であるために，論理シミュレーションを用いた機能検証には膨大な時間がかかる．

これに対して，本論文では，数系や数式を用いた算術アルゴリズムの形式的表現手法および計算機代数に基づく形式的検証手法を提案する．また，提案手法に基づく算術アルゴリズム記述言語を開発するとともに，その言語を用いた演算器モジュールジェネレータを構築し，提案手法の有効性を示す．

第2章　算術演算回路の設計・検証に関する基礎的考察

本章では，LSIの設計に関する基礎的考察を行う．まず，近年のLSIシステム設計において主流となっている高水準設計手法について概説する．次に，算術演算回路の機能検証手法について述べる．特に，高速な検証を実現する方法として近年注目を集めている形式的検証手法に焦点を当て，一般的に使われている手法の限界と問題点について述べる．

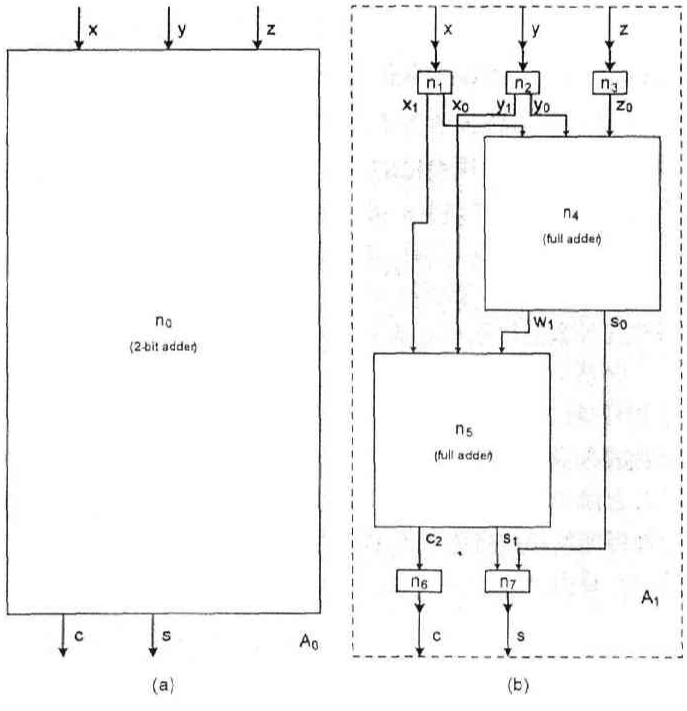
LSIの高水準設計においては，HDLを用いて様々な記述レベルでの回路表現が可能であり，抽象度の高い記述によって設計期間を大幅に短縮することができる．しかし，HDLを用いて算術演算回路を設計する際には，低水準な論理レベルで回路を記述する必要がある．特に，多進数系や冗長数系などの特殊数系に基づく算術アルゴリズムの場合，2値論理への符号化が必要となり，設計者に大きな負担を与えることとなる．このような算術演算回路の記述における問題は，近年開発が進められているSystem CやSystem Verilogにも同様に存在するため，算術アルゴリズムの記述に特化した記述言語が求められる．

LSIの機能検証には一般に論理シミュレーションが用いられるが，入出力数の多い算術演算回路の網羅的な検証は困難である．これに対して，数学的手法を用いて回路機能の正しさを調べる形式的検証手法が研究されている．算術演算回路の形式的検証手法としては，論理関数を一意に表すBDD(Binary

Decision Diagram) やBMD (Binary Moment Diagram) などのグラフ構造の同形判定を用いた手法が知られている．これらの手法は，一定の規模の算術演算回路に対しては有効であることが示されているが，回路規模が大きくなるにつれて計算量とメモリ使用量が著しく増加してしまう．これに対して，回路構造の成り立ちが算術アルゴリズムとして記述され明らかになっていれば，回路を分割して検証することが可能となり，効率的な検証手法が実現可能であると考えられる．

第３章　算術演算回路の形式的表現と計算機代数に基づく形式的検証

本論文では，数系と数式に基づいて算術演算回路を表現するためのデータ構造として，算術回路グラフ（ACG: Arithmetic Circuit Graph）を提案する．算術回路グラフは，ノードと有向辺から構成される有向グラフである．ノードは，整数方程式によって与えられる機能表明と，算術回路グラフによって与えられる内部構造からなる．有向辺は，始点ノード，終点ノード，整数変数からなる．すべての整数変数は，重み数系とレンジ制約によってその値域が制限される．図1に，2ビットの加算器の算術回路グラフとその数式表現を示す．



Arithmmetic Circuit Graph:

$A_0 = (\{n_0\}, \{(\mathrm{nil}, n_0, x), (\mathrm{nil}, n_0, y), (\mathrm{nil}, n_0, z), (n_0, \mathrm{nil}, c), (n_0, \mathrm{nil}, s)\}),$

$A_1 = (\{n_1, n_2, n_3, n_4, n_5, n_6, n_7\},$
$\{(\mathrm{nil}, n_1, x), (\mathrm{nil}, n_2, y), (\mathrm{nil}, n_3, z), (n_1, n_5, x_1), (n_1, n_4, x_0),$
$(n_2, n_5, y_1), (n_2, n_4, y_0), (n_3, n_4, z_0), (n_4, n_5, w_1), (n_5, n_6, c_2),$
$(n_5, n_7, s_1), (n_6, \mathrm{nil}, c), (n_7, \mathrm{nil}, s)\}),$

$n_0 = (\{c + s = x + y + z\}, A_1),$

$n_1 = (\{x_0 + x_1 = x\}, \mathrm{nil}),$

$n_2 = (\{y_0 + y_1 = y\}, \mathrm{nil}),$

$n_3 = (\{z_0 = z\}, \mathrm{nil}),$

$n_4 = (\{w_1 + s_0 = x_0 + y_0 + z_0\}, \mathrm{nil}),$

$n_5 = (\{c_2 + s_1 = x_1 + y_1 + w_1\}, \mathrm{nil}),$

$n_6 = (\{c = c_2\}, \mathrm{nil}),$

$n_7 = (\{s = s_0 + s_1\}, \mathrm{nil}).$

Integer Signal:

$c = (\mathrm{UB}, (2, 2)), s = (\mathrm{UB}, (1, 0)), x = (\mathrm{UB}, (1, 0)),$
$y = (\mathrm{UB}, (1, 0)), z = (\mathrm{UB}, (0, 0)).$

Number System:

$\mathrm{UB} = (W_{\mathrm{UB}}, D_{\mathrm{UB}}),$

$W_{\mathrm{UB}} = (\cdots, 2^{i+1}, 2^i, 2^{i-1}, \cdots),$

$D_{\mathrm{UB}} = (\cdots, \{0, 1\}, \{0, 1\}, \{0, 1\}, \cdots).$

図１　２ビット加算器の算術回路グラフ

算術回路グラフの機能の正しさは，レンジ評価と数式評価と呼ばれる2つの処理によって検証される．レンジ評価では，機能表明の左辺の値域が右辺の値域を包含するかを調べることで，与えられたグラフが回路としての実現であるかどうかを判定する．レンジ評価は，機能表明と入出力変数の値域を用いた総当たり手法によって，容易に行うことができる．数式評価では，内部構造の機能から機能表明が導出可能かを調べることで，内部構造と機能表明の数式的な等価性を判定する．数式評価では，BDDや*BMDを用いた従来の等価性判定手法を利用することができる．一方で，本論文では，内部構造と機能表明が純粋な数式のみで表現されることを利用して，グレブナー基底や多項式簡約といった計算機代数の手法を用いた等価性判定を提案する．提案手法では，まず，内部構造を表す多項式集合からグレブナー基底を導出する．次に，機能表明の多項式のグレブナー基底による多項式簡約を行い，正規形を求める．正規形が0であれば，内部構造の組み合わせによって機能表明が導出可能であるため，数式評価は真を返す．

図２は，乗算器の検証時間である．測定環境は，Intel Core 2 Extreme X6800 2.93 GHz，メモリ 2GBである．数式処理には，Mathematica 5.2を使用した．従来手法との比較として，*BMD等価性判定によ

る機能検証も行った．この結果，大規模の乗算器に対しては，提案手法を用いることで従来手法に比べて高速に検証することができた．さらに，論理演算を含むノードのみを*BMDで検証し，それ以外のノードを数式処理で検証することで，検証時間を大幅に削減することが可能となった(図2のHybrid)．

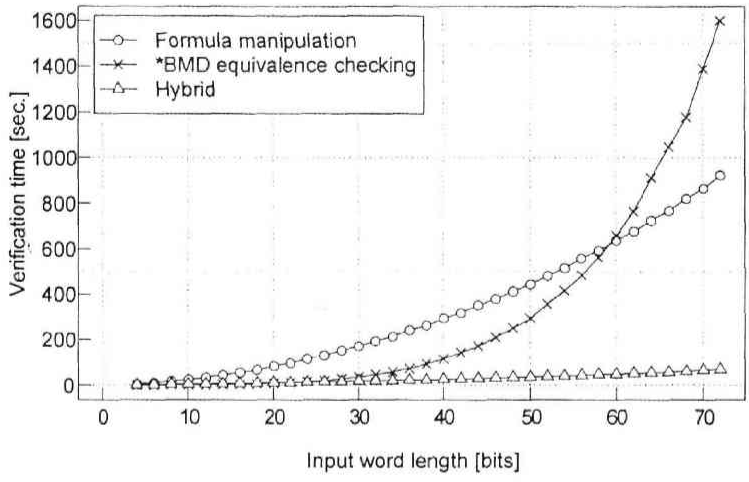


図２　乗算器の検証時間

第４章　算術アルゴリズム記述言語 ARITH

本章では，第2章で提案した表現手法および検証手法を実現する算術アルゴリズム記述言語ARITHを提案する．ARITH記述は，型定義ブロックとモジュールブロックから構成される．

図3(a)は，冗長2進数系の型定義ブロックである．ARITHでは，weight によって各桁の重みを定義し，min, max, step によって各桁の桁集合を定義する．図3(b)は，冗長2進乗算器のモジュールブロックである．モジュールブロックは，モジュール名およびポートリスト（1行目），入出力信号の数系とレンジ制約（2-8行目），機能表明（9行目），内部構造（10-26行目）から構成される．機能表明は整数方程式で与えられ，内部構造はモジュールの組み合わせで与えられる．この例では，P=X*Y という乗算機能を実現するために，4つのモジュールの組み合わせで内部構造を記述している．

ARITH記述は，算術回路グラフに対応しており，第3章で提案した計算機代数に基づく形式的検証手法を適用することで，その機能をアルゴリズムレベルで検証可能である．また，ARITH記述は回路実装によらないアルゴリズム記述であるため，新しい回路方式を対象とした設計でも利用することができる．

```
 1:  typedef SD2_1; /* Radix-2 Signed Digit */
 2:    for(i, SD2_1.low, SD2_1.high) begin
 3:      SD2_1[i].weight = Power(2, i);
 4:      SD2_1[i].min = -1;
 5:      SD2_1[i].max = 1;
 6:      SD2_1[i].step = 1;
 7:    end
 8:  endtypedef
```

(a)

```
 1: module SD_MULTIPLIER(P, X, Y);
 2:   output TC P;
 3:   input TC X, Y;
 4:   constraint begin
 5:     P.high = 16; P.low = 0;
 6:     X.high = 7; X.low = 0;
 7:     Y.high = 7; Y.low = 0;
 8:   end
 9:   assertion P = X * Y;
10:   structure begin
11:     wire SD4_2 B;
12:     wire SD2 PP[];
13:     wire SD2 F;
14:     constraint begin
15:       B.high = 3; B.low = 0;
16:       PP.high = 3; PP.low = 0;
17:       for (i, 0, 3) begin
18:         PP[i].high=i*2+8; PP[i].low=i*2;
19:       end
20:       F.high = 15; F.low = 0;
21:     end
22:     BOOTH_ENCODE U0(B, X);
23:     PPG U1(PP[0],PP[1],PP[2],PP[3],B,Y);
24:     ACCUMULATE U2(F,PP[0],PP[1],PP[2],PP[3]);
25:     SD2TC U3(P, F);
26:   end
27: endmodule
```

(b)

図３　ARITH 記述の例

第５章　ARITH に基づく演算器モジュールジェネレータ

本章では，ARITHの応用として開発した演算器モジュールジェネレータについて述べる．演算器モジュールジェネレータは，設計仕様（演算の種類，算術アルゴリズム，入力信号の数系，入力信号幅など）にしたがって，機能検証済みのHDL記述を自動生成するシステムである．本システムの構成を図4に示す．ARITH記述生成部（ARITH code generator）は，算術アルゴリズムライブラリに保持された生成ルールに従い，ARITH記述を機械的に生成する．ARITH記述検証部（ARITH code verifier）は，数

式処理や*BMD等価性判定を用いた形式的検証により，ARITH記述の機能をアルゴリズムレベルで検証する．ARITH/HDL変換部（ARITH/HDL translator）は，検証済みのARITH記述を等価なHDL記述に変換する．演算器モジュールジェネレータは，図5の算術アルゴリズムをライブラリ化しており，これらの組み合わせで900種類を越える機能保証済みの演算器モジュールを自動生成することができる．構築したジェネレータは，研究室のWebページ（http://www.aoki.ecei.tohoku.ac.jp/arith/mg/）で公開されており，これまでに約108,000件の利用がある．

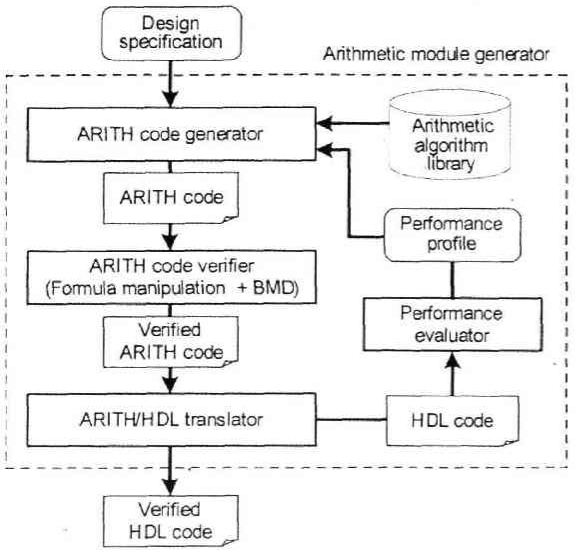


図4　演算器モジュールジェネレータの構成

Number System
Unsigned Binary (UB)
Two' s Complement (TC)

Partial Product Generator
non-Booth
radix-4 modified Booth

Partial Product Accumulator
array
Wallace tree
balanced-delay tree
overturned-stairs tree
Dadda tree
(7,3) counter tree
(4;2) compressor tree
RB addition tree

Final Stage Adder
ripple carry adder
carry look-ahead adder
ripple-block carry look-ahead adder
block carry look-ahead adder
Ladner-Fischer adder
Kogge-Stone adder
Brent-Kung adder
Han-Carlson adder
conditional sum adder
carry select adder
carry skip adder (2 types)

図5　実装した算術アルゴリズム

第6章　結言

本論文では，算術アルゴリズムの形式的設計手法を示した．まず，数系と数式に基づく算術演算回路の形式的表現として算術回路グラフを提案した．また，グレブナー基底や多項式簡約といった計算機代数の手法を用いた算術回路グラフの形式的検証手法を提案した．さらに，提案手法に基づく算術アルゴリズム記述言語 ARITH と，ARITH を用いた演算器モジュールジェネレータを開発し，その有効性を示した．

論文審査結果の要旨

近年、LSIシステムを搭載する機器は著しく増加しており、用途に応じて適切な算術演算回路を設計することが必要になっている。しかし、算術演算回路は、それぞれ固有のハードウェアアルゴリズム（算術アルゴリズム）に基づいて動作するため、論理回路の合成を基本とする従来の設計技術を適用することが困難であった。著者は、算術アルゴリズムの系統的な設計技術を確立するために、数式を用いた算術アルゴリズムの形式的表現法および検証法を考案した。また、これに基づく算術アルゴリズム設計言語を開発するとともに、その言語を用いた演算器モジュールジェネレータを構築した。本論文はこれらの成果をとりまとめたもので、全文6章よりなる。

第1章は緒言である。

第2章では、LSIシステムの設計手法について概説している。特に、現在のハードウェア記述言語とその記述の検証法に関する基礎的考察を与えている。

第3章では、算術アルゴリズムの形式的表現法を提案している。算術演算回路の機能表明と内部構造を数式で記述することによって、重み数系に基づく多様な算術演算回路を統一的に表現できることを明らかにしている。さらに、その内部構造から得られる連立方程式により機能表明の方程式が導出可能かどうかを、計算機代数の手法に基づいて判定し、効率的に機能検証を行う方法を確立している。これは優れた成果である。

第4章では、第3章で提案した表現法および検証法を実現する算術アルゴリズム記述言語ARITHとその処理系を試作した結果を示している。さらに、いくつかの算術演算回路の設計事例を示し、ARITHにより実用的な規模の算術演算回路の形式的設計が可能であることを実証している。これは有用な成果である。

第5章では、ARITHに基づく演算器モジュールジェネレータを構築し、Web上で公開した結果を示している。これは、多入力加算や積和演算などの多様な算術アルゴリズムをライブラリとして有し、機能が完全に保証された900種類を越える演算器モジュールを自動生成することができる。これは実用上重要な成果である。

第6章は結言である。

以上要するに本論文は、計算機代数に基づく算術演算回路の形式的設計法を提案し、算術アルゴリズム記述言語および演算器モジュールジェネレータの実現を通してその有効性を示したものであり、計算機工学および情報基礎科学の発展に寄与するところが少なくない。

よって、本論文は博士（情報科学）の学位論文として合格と認める。