Mizarプルーフチェッカーを用いた冗長2進加算回路の検証

荒井　研一†　　山口真之介††　　和崎　克己†

† 信州大学大学院工学系研究科
†† 九州工業大学情報工学部
E-mail: †arai@security.cs.shinshu-u.ac.jp, ††yamas@el.kyutech.ac.jp, †††wasaki@cs.shinshu-u.ac.jp

あらまし　論理演算器の計算を多ソート代数でモデル化し，証明はプルーフチェッカーを用いて検証する．論理演算子，演算子とハードウェアゲートとの関係，ゲート同士の信号線による接続等の定義・定理を基に，冗長2進加算回路を例とし，演算回路が正しく動作することを検証する．形式検証系としてMizar証明検査システムを用いた．信頼性の高い演算器実装に関する数理的手法を確立することが，本研究の主眼である．

キーワード　プルーフチェッカー、論理演算器、ハードウェア記述、設計検証

A Verification for the Redundant Signed Digit Adder Circuit by using Mizar Proof Checker

Ken'ichi ARAI†, Shin'nosuke YAMAGUCHI††, and Katsumi WASAKI†

† Graduate School of Science and Technology, Shinshu Univeristy
†† Faculty of Computer Science and System Engineering, Kyushu Institute of Technology
E-mail: †arai@security.cs.shinshu-u.ac.jp, ††yamas@el.kyutech.ac.jp, †††wasaki@cs.shinshu-u.ac.jp

Abstract　We propose the calculation models of the logical arithmetic unit based on the many sorted algebra, and verify the circuit's design by using a proof checker. The stability of a circuit (redundant signed digit adder circuit example) is proved based on the definitions and theorems about the logic operator, the hardware gates and the connection by signal lines. For this purpose, we use 'Mizar' proof checking system as formal verification tool. The insistence of this research is to establish the mathematical principle technique concerning the calculation circuits with high reliability.

Key words　Proof Checker, Arithmetic Logical Circuit, Hardware Description, Design Verification

1. はじめに

ディジタル回路を設計する際，最も重要となる点は,「ロジック」と「タイミング」の問題である． 回路の設計検証における従来の研究では，この2つの問題を同時に議論しようとする場合が多かった．これらを同時に解析的に検討するとき，実回路では問題が複雑になりすぎ，回路の範囲を限定した上での小規模回路での議論に留まっている [1] [2] [3] [4] [5].

シミュレーション手法による設計検証 [6] も検討されているが，実回路において，回路要素の遅延時間（最大，最小，平均）の全ての値を調査することは困難である．このため，ゲートアレイ設計の検証法では回路要素や配線の遅延時間をゼロと仮定して上でのシミュレーション（zero-delay simulation）を開発の前段階で行い，設計検証に係る時間の短縮を図る手法がある．しかし，最終的にレイアウトされた実ゲートアレイの動作とは大きな食い違いが生じており，遅延情報を安易に仮定したシミュレーションはあくまで動作の目安に過ぎない，といった報告がある [7]．これらの問題点を解決するためには,「ロジック」と「タイミング」の問題を分けて検討することが有効と考えられる． このうち,「タイミング」の問題に関しては，信号伝搬を「タイミングベルト」という概念によってモデル化し，それを用いた回路のタイミング検証手法が既に提案されている [8] [9] [10].

「ロジック」の問題に関して，演算回路を数学的定義に基づいて設計し，設計検証と動作の正しさをプルーフチェッカー（Proof Checker）を用いて証明する方策について，検討と諸概念の導入が行われた．

回路を定義していくための数学的概念は，回路の構造として，回路内の全信号点を表す状態空間，入出力信号間の写像で定義される演算子，演算に必要な入力信号点を表す演算子から信号点への写像，及び演算結果を表す演算子から出力信号点への写像，といった4つの空間と写像の対として定義する，多ソート

代数構造を用いる [11] [12] [13] [14].

結果，従来のシミュレーション手法などで用いている，狭い条件下での演算子，信号線，遅延情報の組とは異なり，信号点の状態空間を 0/1 の値だけをとり得る 2 値空間だけでなく，ハイインピーダンス状態や連続空間など，様々な空間で回路を定義できることが可能となった．この構造を基に，入力信号により出力信号が一意に決定できる演算子を一つだけ持つような演算器（1 ゲート素子）が定義された [15] [16]．定義に基づいて，演算子が当該回路の入出力信号点の間を写像する関数として正当か，結果が一意に定まる関数が存在するか，入力信号点の状態が変化した場合，演算結果が安定となるか，などの重要な性質が，どのような条件を満たすことで証明可能であるかについて示された．次に，演算回路の合成について定義された [17] [18]．合成後の回路は，上述の構造における各集合・写像の和をとったものであり，この合成回路に関しても，結果が一意に定まるか，演算結果が安定となるか，などの性質が証明可能となった [20].

演算回路の実例として，Montgomery 乗算器の内部演算で使われる，冗長 2 進数表現（RSD : Redundant Signed Digit）[22] [23] を用いたキャリー先送り型の加算演算回路 [24] を取り挙げ，その回路定義と諸定理，更に動作の正しさの証明を行う．2 進数の基本的な演算に関しては定義・証明がなされている [10]．ディジタル回路を定義するための論理演算子，ゲート要素，またそれらの接続に関しての最小限の諸概念については定義・証明がなされている [18]．演算回路を構成するゲート要素のためには多くの論理演算子が用いられるが，必要な場面毎にそれぞれ定義していくことは混乱を招くため，まとまった種類の演算子を用意した [19]．冗長 2 進数表現のための演算器は，3 入力–2 出力で一般化した汎化加算器（Generalized Full Adder Circuit）[24] の定義を改めて行い，これを各ステージで接続することとした [21].

証明は以下のステップで進める． RSD 表現における 1 ビット分の演算回路は，汎化加算器を 2 つ合成した構造で定義する．各々の演算器は，双方ともゲート段数が 2 である回路で定義できる．その定義を基に，回路の入出力信号に関する定理を導く．最後に，入力信号点の状態が決まれば，演算結果が必ず安定することを証明する．合成後の RSD 加算器は，2 ステージパイプライン構成をとるため，全体で 4 ステップで演算結果は安定する．パイプライン終端部では，次段への RSD 表現引き継ぎと先送りしていたキャリーの合算を行うための簡単な AND 回路を構成し，最終的な加算結果を得る．

本稿の定義・定理は，*Mizar proof checking system* [25] [26] [27] により，証明の正しさを検証している．*Mizar* とは，Bialystok 大学（ポーランド）の Andrzej Trybulec 教授を中心とした *Mizar Society* によって進められている，数学をコンピュータを使って形式化するプロジェクトの総称である．*Mizar* プロジェクトは，コンピュータを使って数学を記述するために作られた *Mizar* 言語 によって数学の証明を記述し，それを Windows あるいは Unix 環境下で稼働する *Mizar* プルーフチェッカーによって証明をチェックし，それを *Mizar* ライブラリに登録することによって行われている．このプロジェクトの目的は，数学論文のチェックシステムを作ることである．

Mizar によって数学の証明を形式化し記述したものを *article* と呼ぶ. 新たに *article* を記述する際には, 過去既に証明が検査済みでライブラリに登録されている多くの *article* を参照しながら進めることができる．記述した *article* がライブラリに登録された後, その *article* は他の *article* が参照することができる．*article* は *Mizar* 言語によって記述する．*Mizar* 言語は, 数学の一般的な証明の記述法を基にしているが, *Mizar* 独特の記法もあるので, 文法の詳細に関しては文献 [25] [27] を参照されたい.

2. 回路の構造・定義・定理

本節では, Trybulec, 中村らによってなされた, 一般の回路を定義していくための数学諸概念 [11] [12] [13] [14] [15] [16] [17] について, その構造，演算，入出力信号，合成，状態について取り挙げる．また, Bancerek らによって定義・証明がなされた, ディジタル回路を定義するための論理演算子, ゲート要素, またそれらの接続に関しての諸概念 [18] についても取り挙げる．なお, これらの定義・定理は *Mizar* プルーフチェッカーを利用し, 証明の正しさを検証している.

2.1 回路の構造

回路の構造に関して, *Many Sorted Signature* という構造型を導入する. これは *carrier, operation symbols, arity, result sort* の組である．この概念を回路素子に当てはめた場合, それぞれ, 回路素子の入力信号と演算子を含む内部状態, 演算子, 演算子への入力状態, 出力状態と対応する（図 1）.

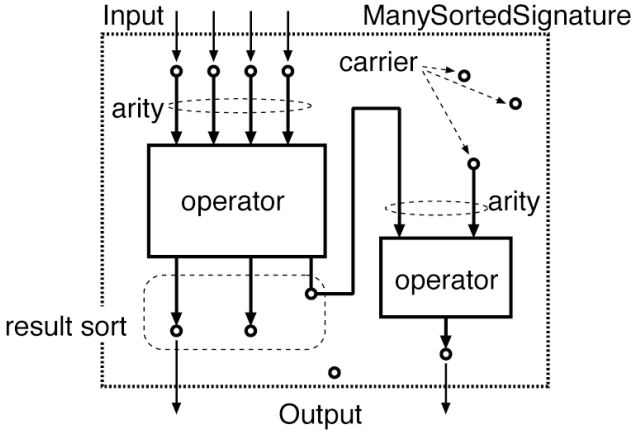

図 1　*Many Sorted Signature* の構造図
Fig 1: A structure of *Many Sorted Signature*.

［定義 2.1］ **(回路の構造)**

Many Sorted Signature とは, 以下に示す *carrier, operation symbols, arity,* 及び *result sort* の組で定義される.

≪ *carrier, operation symbols, arity, result sort* ≫

ここで, *carrier, operation symbols* は一般の集合である．*arity* は *operation symbols* から *carrier** への関数, *result sort* は *operation symbols* から *carrier* への関数である.

2.2 演算回路

演算子（*operation symbol*）を一つだけ持つ回路に関して，$1GateCircStr(p,f)$ という型を定義する．これは入力信号 (*rng* p) を含めた内部状態 (*carrier*) において，演算子 (f) とその入力信号の組 (p) が一つあるような空でない *Many Sorted Signature* である．演算子への入力 *arity* は $rng(p)$, 出力 *result sort* は $\langle p,f\rangle$ となり出力は一意に決まる．

［定義 2.2］ **(*1GateCircStr*(p,f) の諸定義)**

演算子 f を一般の集合，入力信号線の組 p を 有限数列とする．*1GateCircStr*(p,f) とは，*void* ではない *Many Sorted Signature* の下で，以下の様に定義される．

(i) *The carrier of 1GateCircStr*(p,f)
$= rng(p) \cup \{\langle p,f\rangle\}$

(ii) *the operation symbols of 1GateCircStr*(p,f)
$= \{\langle p,f\rangle\}$

(iii) (*the arity of 1GateCircStr*(p,f))$(\langle p,f\rangle) = p$

(iv) (*the result sort of 1GateCircStr*(p,f))$(\langle p,f\rangle)$
$= \langle p,f\rangle$

2.3 回路の入出力信号

回路が用いる入出力信号線に関して，*Input Vertices* と *Inner Vertices* という関数を定義する．*Input Vertices* は内部状態 (*carrier*) から出力状態の要素 *rng*(*result sort of G*) の差をとったものであり，演算子の入力そのものである．他方 *Innner Vertices* は演算出力状態の要素 *rng*(*result sort of G*) である．

［定義 2.3］ **(関数 *Input Vertices*, *Inner Vertices*)**

G を *void* ではない かつ 空ではない *Many Sorted Signature* とする． このとき，関数 *InputVertices*(G) 及び *InnerVertices*(G) は，G における *carrier* の 部分集合の下で，以下の様に定義される．

(i) *InputVertices*(G)
$=$ (*The carrier of G*) $\setminus$ *rng*(*the result sort of G*)

(ii) *InnerVertices*(G) $=$ *rng*(*the result sort of G*)

この定義から，上述の $1GateCircStr(p,f)$ の場合,*InputVertices* と *InnerVertices* に関して以下の定理が得られる．

［定理 2.1］ **(演算回路の入出力)**

演算子 f を一般の集合，入力信号線の組 p を 有限数列とする．関数 *1GateCircStr*(p,f) を *void* ではない *Many Sorted Signature* の下で定義される演算回路とする．このとき，入出力線に関して，以下の定理が成立する．

(i) $(\forall f)(\forall p)(InputVerties(1GateCircStr(p,f))$
$= rng(p))$

(ii) $(\forall f)(\forall p)(InnerVerties(1GateCircStr(p,f))$
$= \{\langle p,f\rangle\})$

2.4 演算回路の合成

Many Sorted Signature 構造に関して，各要素の和（演算 $+\cdot$）をとることで，回路の合成（*combine*）が定義できる．

［定義 2.4］ **(*Many Sorted Signature* の合成)**

空ではない f,g を考える．この時，$f+\cdot g$ は空ではない．一般の集合 A,B を考え，f,g を A,B で *index* された *Many Sorted Set* とするとき，合成 $f+\cdot g$ は以下の様に定義される．

(i) 合成 $f+\cdot g$ は $A \cup B$ で *index* された *Many Sorted Set* である

S_1, S_2 を空ではない *Many Sorted Signature* とするとき，$S_1 +\cdot S_2$ の各要素は 空ではない *Many Sorted Signature* の下で，以下の様に定義される．

(ii) *The carrier of* $S_1 +\cdot S_2$
$=$ (*the carrier of* S_1) $\cup$ (*the carrier of* S_2)

(iii) *the operation symbols of* $S_1 +\cdot S_2$
$=$ (*the operation symbols of* S_1)
$\cup$ (*the operation symbols of* S_2)

(iv) *the arity of* $S_1 +\cdot S_2$
$=$ (*the arity of* S_1) $\cup$ (*the arity of* S_2)

(v) *the result sort of* $S_1 +\cdot S_2$
$=$ (*the result sort of* S_1) $\cup$ (*the result sort of* S_2)

この定義から，上述の $1GateCircStr(p,f)$ の場合,*InputVertices* と *InnerVertices* の回路合成後の状態に関して，以下の定理が得られる．

［定理 2.2］ **(回路合成後の入出力信号)**

S_1, S_2 を，$S_1 \approx S_2$ であるような，空ではない *Many Sorted Signature* とするとき，以下の定理が成立する．

(i) $(\forall S_1)(\forall S_2)(InnerVertices(S_1 + \cdot S_2)$
$= InnerVerties(S_1) \cup InnerVertices(S_2))$

(ii) $(\forall S_1)(\forall S_2)(InputVertices(S_1 + \cdot S_2)$
$\subseteqq InputVerties(S_1) \cup InputVertices(S_2))$

2.5 動作後の回路の状態

回路を動作させた後の回路の状態について *Following* という関数を定義する．回路の状態 (v) に関して，動作後の状態は $s(v)$ で示される．また，回路が安定（*stable*）ということは，すなわち，回路動作の前後において状態 s が変化していないことである．

［定義 2.5］ **(動作後の回路状態)**

I_1 を *void* ではない かつ 空ではない *circuit-like* な *Many Sorted Signature* とする．S_1 を 空ではない I_1 における回路とする．s を S_1 の回路状態とする．このとき，関数 $Following(s)$ は,S_1 の回路状態 の下で，以下の様に定義される．

v を I_1 の *vertex* とするとき，

(i) $v \in InputVertices(I_1)$ ならば
$(Following(s))(v) = s(v)$ とする

(ii) $v \in InnerVertices(I_1)$ ならば $(Following(s))(v) = (Den(the\ action\ at\ v, S_1))((the\ action\ at\ v) depend-on-in\ s)$ とする

(iii) 動作後の S_1の状態 $= Following$(動作前の S_1の状態) ならば S_1は安定（*stable*）とする

例えば，上述の $1GateCircStr(p,f)$ に関して，演算子 f が集合 $Boolean^2$ から *Boolean* への関数である場合，以下の安定性に関する定理が得られる．これは 2 入力の *Boolean* 演算子 f を持つ *1-GateCircuit* の状態は，$Following(s)$（1 ステップ）で安定であることを示している．故に，*1-GateCircuit* を複数個，直列に並べて接続したような合成回路は，その段数 n に応じて，$Following(s,n)$（n ステップ）で安定であることが別途示

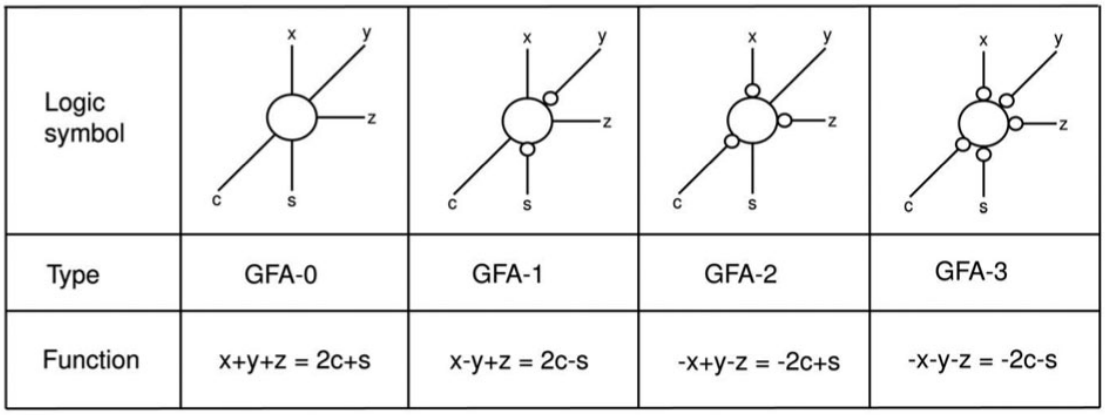

図 2　汎化加算器の 4 つの類形（文献 [24] より引用）
Fig 2: Four type of Generalized Full Adder Circuits.

される．

［定理 2.3］ **(演算回路の安定性判別)**

回路の演算子 $f : Boolean^2 \mapsto Boolean$ の関数とする．x_1,x_2 は $Boolean\{0,1\}$ の要素とする．$\langle x_1, x_2\rangle$ は長さ 2 の有限数列で演算子 f の入力信号線を示す．s を入力信号線 $\langle x_1, x_2\rangle$ であるような演算回路 $1GateCircuit(\langle x_1, x_2\rangle, f)$ の回路状態とする．このとき, 演算回路の動作後の状態に関して, 以下の定理が成立する．

$(\forall f)(\forall s)(\forall x_1, x_2)($
$(Following(s))([\langle x_1, x_2\rangle, f]) = (f)(\langle s(x_1), s(x_2)\rangle)$
かつ $(Following(s))(x_1) = s(x_1)$
かつ $(Following(s))(x_2) = s(x_2)$ ならば動作後の状態 $Following(s)$ は安定 $(stable)$ である)

3. 汎化加算器（Generalized Full Adder Circuit）

本節では, RSD 表現向けの演算素子として「汎化加算器」[24] を取り上げ, その数学定義と回路の安定性について述べる [21]. この汎化加算器を用いて，次節で説明するキャリー保存型の高速加算回路をパイプライン型で構成する．ここでは,**2.** で説明した回路に関する数学諸概念を基に, 証明検査済の *Mizar* ライブラリ [19] で定義した論理演算子を用いて，演算回路の定義を行う．なお，これらの定義・定理は *Mizar* プルーフチェッカーを利用し, 証明の正しさを検証している．

3.1 回路の構造

汎化加算器（Generalized Full Adder Circuit）[24] は，従来からある 2 ビット入力+1 ビットのキャリー入力を有するような全加算器の構成を一般化することにより，3 ビット入力からキャリー出力と後に述べる RSD 表現における中間和の出力を同時に得るように工夫された演算素子の提案である（図 2）．

汎化加算器は，3 ビット入力の中間和を求める加算回路と，キャリー出力を求める桁上がり演算器の 2 つの回路を合成することで構成される．入力は x, y, z であるが，4 つの類形によって各々入力の論理が異なっている．また出力は s, c であるが，これも 4 つの類形によって各々出力の論理が異なっている．以下, 類形タイプ 1（GFA-1）を例とし，演算器の定義を行い, 種々の定理を導いた後, 回路の安定性について示す．他の類形タイプについても，同様に定義が行われ，種々の定理が導かれる．

3.2 汎化加算回路・タイプ 1（GFA-1）の定義

汎化加算回路・タイプ 1（GFA-1）は，入力 x, y, z と出力 s, c の関係が，$x - y + z = 2c - s$ を満たすような演算素子である．加算器 *GFA1AdderStr(x,y,z)* と桁上がり演算器 *GFA1CarryStr(x,y,z)* の定義を行う．この 2 つの回路を合成して，汎化加算回路 *BitGFA1Str(x,y,z)* を構成する．n

［定義 3.1］ **(加算器の定義)**

x,y,z を 集合 とする．このとき, *Many Sorted Signature* の下で, 加算器の構造 *GFA1AdderStr(x,y,z)* は, 以下の様に定義される．

$GFA1AdderStr(x, y, z)$
$= 1GateCircStr(\langle x, y\rangle, xor2c)$
$\quad +\cdot 1GateCircStr(\langle[\langle x, y\rangle, xor2c], z\rangle, xor2c)$

ここで，*2* 入力型論理演算子 *xor2c* は $Boolean^2 \mapsto Boolean$ の関数であって，入力 $\langle x_1, x_2\rangle$ を持つ場合，その演算結果が $x_1 \oplus not(x_2)$ として定義されるものである *[21]*.

回路 *GFA1AdderCirc(x,y,z)* は, 回路の構造が *GFA1AdderStr* である *Many Sorted Signature* の下で, 以下の様に定義される．

$GFA1AdderCirc(x, y, z)$
$= 1GateCircuit(x, y, xor2c)$
$\quad +\cdot 1GateCircuit([\langle x, y\rangle, xor2c], z, xor2c)$

演算出力 *(s)* : $GFA1AdderOutput(x, y, z)$ は, $InnerVertices$ $(GFA1AdderStr(x, y, z))$ の要素の元として, 以下の様に定義される．

$GFA1AdderOutput(x, y, z)$
$= [\langle[\langle x, y\rangle, xor2c], z\rangle, xor2c]$

［定義 3.2］ **(桁上がり演算器の定義)**

x,y,z を 集合 とする．このとき, *Many Sorted Signature* の下で, 桁上がり演算器の構造 *GFA1CarryStr(x,y,z)* は, 以下の様に定義される．

$GFA1CarryStr(x, y, z)$
$= (1GateCircStr(\langle x, y\rangle, and2c)$

$+\cdot\, 1GateCircStr(\langle y,z\rangle, and2a)$
$+\cdot\, 1GateCircStr(\langle z,x\rangle, and2))$
$+\cdot\, 1GateCircStr(\langle[\langle x,y\rangle, and2c],[\langle y,z\rangle, and2a],$
$[\langle z,x\rangle, and2]\rangle, or3)$

ここで，*2* 入力型論理演算子 *and2c, and2a, and2* は $Boolean^2 \mapsto Boolean$ の関数であって，入力 $\langle x_1, x_2\rangle$ を持つ場合，その演算結果が各々， $x_1 \wedge not(x_2)$, $not(x_1) \wedge x_2$, $x_1 \wedge x_2$ として定義されるものである *[19] [21]*. また，*3* 入力型論理演算子 *or3* は $Boolean^3 \mapsto Boolean$ の関数であって，入力 $\langle x_1, x_2, x_3\rangle$ を持つ場合，その演算結果が $x_1 \vee x_2 \vee x_3$ として定義されるものである *[19]*.

回路 *GFA1CarryCirc(x,y,z)* は，回路の構造が *GFA1CarryStr* である *Many Sorted Signature* の下で，以下の様に定義される．

$GFA1CarryStr(x,y,z)$
$= (1GateCircuit(x,y,and2c)$
$+\cdot\, 1GateCircuit(y,z,and2a)$
$+\cdot\, 1GateCircuit(z,x,and2))$
$+\cdot\, 1GateCircuit([\langle x,y\rangle, and2c],[\langle y,z\rangle, and2a],$
$[\langle z,x\rangle, and2], or3)$

演算出力 *(c) : GFA1CarryOutput(x,y,z)* は，*InnerVertices (GFA1CarryStr(x,y,z))* の要素の元として，以下の様に定義される．

$GFA1CarryOutput(x,y,z)$
$= [\langle[\langle x,y\rangle, and2c],[\langle y,z\rangle, and2a],[\langle z,x\rangle, and2]\rangle, or3]$

［定義 3.3］ **(汎化加算回路（タイプ 1）の合成)**

x,y,z を 集合 とする． このとき，*Many Sorted Signature* の下で，合成した汎化加算回路（タイプ *1*）の構造 *BitGFA1Str(x,y,z)* は，以下の様に定義される．

$BitGFA1Str(x,y,z)$
$= GFA1AdderStr(x,y,z) +\cdot GFA1CarryStr(x,y,z)$

回路 *BitGFA1Circ(x,y,z)* は，回路の構造が *BitGFA1Str* である *Many Sorted Signature* の下で，以下の様に定義される．

$BitGFA1Circ(x,y,z)$
$= GFA1AdderCirc(x,y,z) +\cdot GFA1CarryCirc(x,y,z)$

3.3 回路の入出力信号に関する定理

上で定義した演算回路の定義を基に，各回路の内部状態，入出力信号である *carrier, InputVertices, InnerVertices* に関する定理を証明する．

［定理 3.1］ **(加算器に関する諸定理)**

x,y,z を 集合 とする．このとき，加算器に関する以下の定理が成立する．

$(\forall x,y,z)((InnerVertices(GFA1AdderStr(x,y,z))$
は *Relation* である)
かつ *the carrier of* $GFA1AdderStr(x,y,z) =$
$\{x,y,z\} \cup \{[\langle x,y\rangle, xor2c],[\langle[\langle x,y\rangle, xor2c], z\rangle, xor2c]\}$
かつ $InnerVertices(GFA1AdderStr(x,y,z)) =$
$\{[\langle x,y\rangle, xor2c],[\langle[\langle x,y\rangle, xor2c], z\rangle, xor2c]\}$
かつ $InputVertices(GFA1AdderStr(x,y,z)) = \{x,y,z\}$

［定理 3.2］ **(桁上がり演算器に関する諸定理)**

x,y,z を 集合 とする．このとき，桁上がり演算器に関する以下の定理が成立する．

$(\forall x,y,z)((InnerVertices(GFA1CarryStr(x,y,z))$
は *Relation* である)
かつ *the carrier of* $GFA1CarryStr(x,y,z) =$
$\{x,y,z\} \cup \{[\langle x,y\rangle, and2c],[\langle y,z\rangle, and2a],[\langle z,x\rangle, and2]\}$
$\cup \{[\langle[\langle x,y\rangle, and2c],[\langle y,z\rangle, and2a],[\langle z,x\rangle, and2]\rangle, or3]\}$
かつ $InnerVertices(GFA1CarryStr(x,y,z)) =$
$\{[\langle x,y\rangle, and2c],[\langle y,z\rangle, and2a],[\langle z,x\rangle, and2]\}$
$\cup \{[\langle[\langle x,y\rangle, and2c],[\langle y,z\rangle, and2a],[\langle z,x\rangle, and2]\rangle, or3]\}$
かつ $(InputVertices(GFA1CarryStr(x,y,z)) = \{x,y,z\})$

［定理 3.3］ **(汎化加算回路（タイプ 1）に関する諸定理)**

x,y,z を 集合 とする．このとき，汎化加算回路（タイプ *1*）に関する以下の定理が成立する．

$(\forall x,y,z)((InnerVertices(BitGFA1Str(x,y,z))$
は *Relation* である） かつ
the carrier of $BitGFA1Str(x,y,z) =$
$(the\ carrier\ of\ GFA1AdderStr(x,y,z)) \cup$
$(the\ carrier\ of\ GFA1CarryStr(x,y,z))$ かつ
$InnerVertices(BitGFA1Str(x,y,z)) =$
$(InnerVertices(GFA1AdderStr(x,y,z))) \cup$
$(InnerVertices(GFA1CarryStr(x,y,z)))$ かつ
$InputVertices(BitGFA1Str(x,y,z)) = \{x,y,z\}$

3.4 動作の安定性の証明

演算回路の定義と種々の定理を基に，汎化加算回路（タイプ 1）*BitGFA1Str*(x,y,z) に関する，動作の安定性の証明を行う．

［定理 3.4］ **(汎化加算回路（タイプ 1）の 2 ステップ動作後)**

x,y,z を 集合とする．ただし，入力信号点 x,y,z は，回路の内部信号点とは異なることを前提条件とする．このとき，s を 汎化加算回路 *BitGFA1Circ* (x,y,z) の信号点の状態，*Boolean* の要素であるような a_1,a_2,a_3 を $a_1 = s(x)$, $a_2 = s(y)$, $a_3 = s(c)$ とするとき，*2* ステップ動作後の状態のうち出力信号点に着目した場合，以下の値を得る．

$(\forall x,y,z)(\forall s)(\forall a_1,a_2,a_3)($
$Following(s,2)(GFA1AdderOutput(x,y,z))$
$= not(a_1 \oplus not(a_2) \oplus a_3)$, かつ
$Following(s,2)(GFA1CarryOutput(x,y,z))$
$= (a_1 \wedge not(a_2)) \vee (not(a_2) \wedge a_3) \vee (a_3 \wedge a_1))$

［定理 3.5］ **(汎化加算回路（タイプ 1）の安定性)**

x,y,z を 集合とする．ただし，入力信号点 x,y,z は，回路の内部信号点とは異なることを前提条件とする．このとき，s を汎化加算回路 *BitGFA1Circ* (x,y,z) の信号点の状態として，回路は，*2* ステップ動作した後，安定である．つまり，

$(\forall x,y,z)(\forall s)(Following(s,2)\ is\ stable)$

【証明】

入力信号点 x,y,z を，回路の内部信号点とは異なることを前提条件として採用する．S を $BitGFA1Str(x,y,z)$ とする．s を 回路 S の全ての状態，s_1 を $Following(s,2)$（2 ステップ動作後の状態），s_2 を $Following(Following(s,2))$（3 ステップ動作後の状態）とする．このとき，$s_1 = s_2$ であることを示せばよい．

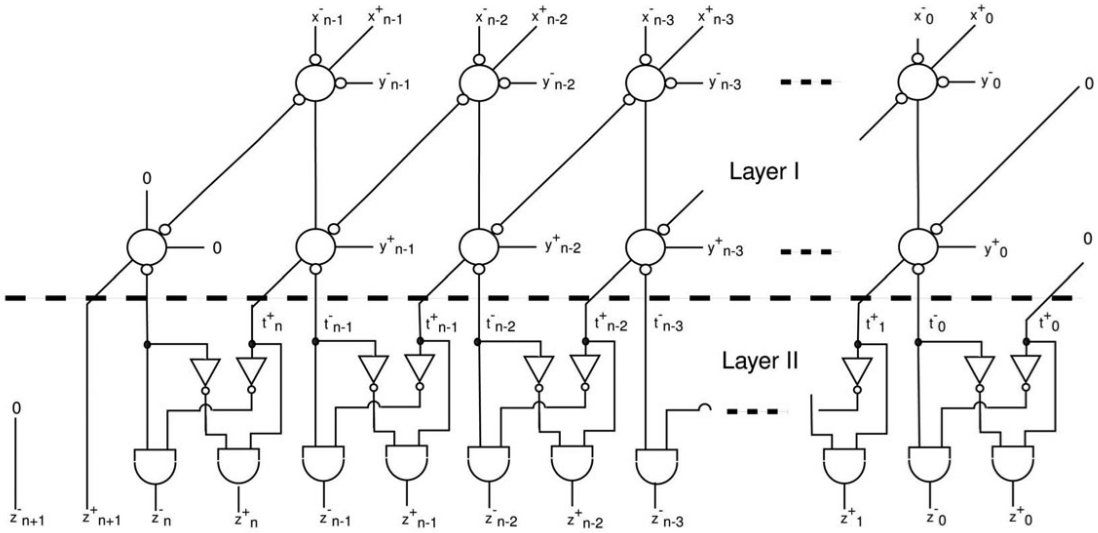

図 3　冗長 2 進加算器の回路構成（文献 [24] より引用）
Fig 3: Circuit Configuration of Redundant Signed Digit Adder.

[A] S の $carrier$ は，$dom(s_1)$ であり かつ $dom(s_2)$ である（[15] 定理 4）．[B] S の $carrier$ は，$InnerVertices($ $GFA1AdderStr\ (x,y,z)) \cup InnerVertices\ (\ GFA1CarryStr\ (x,y,z))$ である（定理 3.3）．　いま，a を S の $carrier$ の集合の要素とすると，$a \in \{x,y,z\}$ または $a \in \{[\langle x,y\rangle, xor2c], [\langle [\langle x,y\rangle, xor2c], z\rangle, xor2c]\}$ または $a \in \{[\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\}$ または $a \in \{[\langle [\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\rangle, or3]\}$ である（[B] ならびに [28] 定理 8）．従って，[C] $a = x$ または $a = y$ または $a = b$ または $a = [\langle x,y\rangle, xor2c]$ または $a = [\langle [\langle x,y\rangle, xor2c], z\rangle, xor2c]$ または $a = [\langle x,y\rangle, and2c]$ または $a = [\langle y,z\rangle, and2a]$ または $a = [\langle z,x\rangle, and2]$ または $a = [\langle [\langle x,y\rangle, and2c], [\langle y,z\rangle, and2a], [\langle z,x\rangle, and2]\rangle, or3]$ である（[29] 定理 8）．先ず，[D] 入力信号線 x,y,z に関して，これはゲート出力ではないから，常に，$s_2(x) = s_1(x)$ かつ $s_1(x) = s(x)$ かつ $s_2(y) = s_1(y)$ かつ $s_1(y) = s(y)$ かつ $s_2(c) = s_1(c)$ かつ $s_1(c) = s(c)$ である．次に，[E] 定義 3.1 を用いて $s_1([\langle [\langle x,y\rangle, xor2c], z\rangle, xor2c]) = s_1(GFA1AdderOutput(x,y,z))$ である．論理の等価性にしたがって，$s_1([\langle [\langle x,y\rangle, xor2c], z\rangle, xor2c]) = not(s_1(x) \oplus not(s_1(y)) \oplus s_1(c))$ である（定理 3.4）．$Following$ を繰り返し適用して，$s_2([\langle [\langle x,y\rangle, xor2], c\rangle, xor2]) = not(s_1(x) \oplus not(s_1(y)) \oplus s_1(c))$ を得る．同様に，[F] $s_1([\langle [\langle x,y\rangle, and2c]\ ,\ [\langle y,z\rangle, and2a]\ ,\ [\langle z,x\rangle, and2]\rangle, or3] = s_1(GFA1CarryOutput(x,y,z))$．これより，$s_1([\langle [\langle x,y\rangle, and2c]\ , [\langle y,z\rangle, and2a]\ , [\langle z,x\rangle, and2]\ \rangle, or3] = (s_1(x) \wedge not(s_1(y))) \vee (not(s_1(y)) \wedge s_1(c)) \vee (s_1(c) \wedge s_1(x))$ であるので，$s_2([\langle [\langle x,y\rangle, and2c]\ ,\ [\langle y,z\rangle, and2a]\ ,\ [\langle z,x\rangle, and2]\rangle, or3] = (s_1(x) \wedge not(s_1(y))) \vee (not(s_1(y)) \wedge s_1(c)) \vee (s_1(c) \wedge s_1(x))$ を得る．以上 [C]〜[F] より $s_2(a) = s_1(a)$ である．故に，[A] ならびに [30] 定理 9 から，$s_1 = s_2$ となる．

4. 冗長 2 進加算器（Redundant Signed Digit Adder）

本節では，演算回路の実例として，Montgomery 乗算器の内部演算で使われる，冗長 2 進数表現（RSD : Redundant Signed Digit）[22] [23] を用いたキャリー先送り型の加算演算回路 [24] について，その回路定義と諸定理，更に動作の正しさの証明を行う．なお，これらの定義・定理は *Mizar* プルーフチェッカーを利用し，証明の正しさを検証している．

4.1 回路の構造

冗長 2 進加算器（Redundant Signed Digit Adder）[24] は，加算器への入出力を行うビット表現を，0, 1 という単一の Boolean から，例えば $\{\langle 0,0\rangle, \langle 1,0\rangle, \langle 0,1\rangle\}$ という 2 つの Boolean 要素を組にした表現（冗長 2 進数）へ拡張し，これによって，加算演算時に発生するキャリー（桁上り）を出来るだけ少なくし，かつ，回路構成を工夫することで，最終的な加算出力を得る段階（レイヤ）まで，キャリーに関する計算を先送りできるように工夫し，キャリー伝搬に伴う速度低下を防止する機構を有する，演算回路の提案である（図 3）．

冗長 2 進数表現を用いた中間和を求める演算器は，3 入力–2 出力で一般化した汎化加算器（Generalized Full Adder Circuit）[21] を用いて，これをレイヤ I で 2 段のパイプラインで接続する．汎化加算器の動作の安定性（2 ステップで全状態が安定化する）については，前節 **3.** にて示した．合成後の RSD 加算器は，2 ステージパイプライン構成をとるため，全体で 4 ステップで演算結果は安定する．パイプライン終端部（レイヤ II）では，次段への RSD 表現引き継ぎと先送りしていたキャリーの合算を行うための簡単な AND 回路（1 ステップで安定化）を構成し，最終的な加算結果を得る．よって，5 ステップで全状態が安定化する．

4.2 冗長 2 進加算器・レイヤ I（中間和）の定義

冗長 2 進加算器・レイヤ I（中間和）の回路構成は，$k-1$ ビッ

ト目から先送りされてきた中間キャリーの入力 c_{in} と，k ビット目の桁の入力 $\langle x_k^+, x_k^- \rangle$ と $\langle y_k^+, y_k^- \rangle$ との RSD 加算演算を行い，中間和出力と先送りされてきたキャリーの組 $\langle t_k^+, t_{k+1}^- \rangle$ を得るようになっている．

レイヤ I の加算器 $BitRSD1Str(x^-, x^+, y^-, y^+, c_{in})$ の定義を行う．これは，汎化加算器のタイプ 2($BitGFA2Str$) とタイプ 1($BitGFA1Str$) を，和出力線で直列合成したものである．

［定義 4.1］ **(レイヤ I 加算器の合成)**

$x^-, x^+, y^-, y^+, c_{in}$ を集合とする．このとき，*Many Sorted Signature* の下で，レイヤ I 加算器の構造 $BitRSD1Str(x^-, x^+, y^-, y^+, c_{in})$ は，以下の様に定義される．

$$BitRSD1Str(x^-, x^+, y^-, y^+, c_{in})$$
$$= BitGFA2Str(x^-, x^+, y^-)$$
$$+\cdot\, BitGFA1Str(GFA2AdderOutput(x^-, x^+, y^-), c_{in}, y^+)$$

回路 $BitRSD1Circ(x^-, x^+, y^-, y^+, c_{in})$ は，回路の構造が *BitRSD1Str* である *Many Sorted Signature* の下で，以下の様に定義される．

$$BitRSD1Circ(x^-, x^+, y^-, y^+, c_{in})$$
$$= BitGFA2Circ(x^-, x^+, y^-)$$
$$+\cdot\, BitGFA1Circ(GFA2AdderOutput(x^-, x^+, y^-), c_{in}, y^+)$$

4.3 回路の入出力信号に関する定理

上で定義した演算回路の定義を基に，各回路の内部状態，入出力信号である *carrier, InputVertices, InnerVertices* に関する定理を証明する．

［定理 4.1］ **(レイヤ I 加算器に関する諸定理)**

$x^-, x^+, y^-, y^+, c_{in}$ を集合とする．このとき，レイヤ I 加算器に関する以下の定理が成立する．

$$(\forall x^-, x^+, y^-, y^+, c_{in})($$
$$(InnerVertices(BitRSD1Str(x^-, x^+, y^-, y^+, c_{in}))$$
$$\text{は } Relation \text{ である})$$
$$\text{かつ } the\ carrier\ of\ BitRSD1Str(x^-, x^+, y^-, y^+, c_{in}) =$$
$$\{x^-, x^+, y^-, y^+, c_{in}\} \cup$$
$$\{[\langle x^-, x^+ \rangle, xor2c], GFA2AdderOutput(x^-, x^+, y^-)\} \cup$$
$$\{[\langle x^-, x^+ \rangle, and2c], [\langle x^+, y^- \rangle, and2a], [\langle y^-, x^- \rangle, and2],$$
$$GFA2CarryOutput(x^-, x^+, y^-)\} \cup$$
$$\{[\langle GFA2AdderOutput(x^-, x^+, y^-), c_{in} \rangle, xor2c],$$
$$GFA1AdderOutput(GFA2AdderOutput(x^-, x^+, y^-),$$
$$c_{in}, y^+)\} \cup$$
$$\{[\langle GFA2AdderOutput(x^-, x^+, y^-), c_{in} \rangle, and2c],$$
$$[\langle c_{in}, y^+ \rangle, and2a],$$
$$[\langle y^+, GFA2AdderOutput(x^-, x^+, y^-) \rangle, and2],$$
$$GFA1CarryOutput(GFA2AdderOutput(x^-, x^+, y^-),$$
$$c_{in}, y^+)\}$$

$$\text{かつ } InputVertices(BitRSD1Str(x^-, x^+, y^-, y^+, c_{in})) =$$
$$\{x^-, x^+, y^-, y^+, c_{in}\}$$

4.4 動作の安定性の証明

演算回路の定義と種々の定理を基に，レイヤ I 加算器 $BitRSD1Str(x^-, x^+, y^-, y^+, c_{in})$ に関する，動作の安定性の証明を行う．

［定理 4.2］ **(レイヤ I 加算器の 4 ステップ動作後)**

$x^-, x^+, y^-, y^+, c_{in}$ を集合とする．入力信号点 $x^-, x^+, y^-, y^+, c_{in}$ は，回路の内部信号点とは異なることを前提条件とする．このとき，s をレイヤ I 加算器 $BitRSD1Circ(x^-, x^+, y^-, y^+, c_{in})$ の信号点の状態，*Boolean* の要素であるような a_1,a_2,a_3,a_4,a_5 を $a_1 = s(x^-)$, $a_2 = s(x^+)$, $a_3 = s(y^-)$, $a_4 = s(y^+)$, $a_5 = s(c_{in})$ とするとき，*4* ステップ（*2+2* ステップ）動作後の状態のうち，ある出力信号点（t^-）に着目した場合，以下の値を得る．

$$(\forall x^-, x^+, y^-, y^+, c_{in})(\forall s)(\forall a_1, a_2, a_3, a_4, a_5)($$
$$Following(s, 4)(BitRSD1Output^-(x^-, x^+, y^-, y^+, c_{in}))$$
$$= not(not(a_1) \oplus a_2 \oplus not(a_3) \oplus a_4 \oplus not(a_5))$$

もうひとつの出力信号点（t^+）に着目した場合にも，同様の値を得る（省略）．

［定理 4.3］ **(レイヤ I 加算器の安定性)**

$x^-, x^+, y^-, y^+, c_{in}$ を集合とする．入力信号点 $x^-, x^+, y^-, y^+, c_{in}$ は，回路の内部信号点とは異なることを前提条件とする．このとき，s をレイヤ I 加算器 $BitRSD1Circ(x^-, x^+, y^-, y^+, c_{in})$ の信号点の状態として，回路は，4 ステップ動作した後，安定である．つまり，

$$(\forall x^-, x^+, y^-, y^+, c_{in})(\forall s)(Following(s, 4)\ is\ stable)$$

【証明は省略. [20] 定理 20 を用いる．2+2 ステップで安定.】

4.5 冗長 2 進加算器・レイヤ II（キャリーとの合算）の定義

冗長 2 進加算器・レイヤ II（キャリーとの合算）の回路構成は，$k-2$ ビット目と $k-1$ ビット目から先送りされてきた中間キャリー t^+ と，k ビット目の桁の中間和を入力として，最終的に求めたい RSD 表現による和出力 $\langle z_k^+, z_{k+1}^- \rangle$ を得るようになっている．

レイヤ II の加算器 $BitRSDStr(x^-, x^+, y^-, y^+, c_{in}, t^+)$ の定義を行う．これは，レイヤ I 加算器 ($BitRSD1Str$) とキャリー合算のための AND 回路 ($1GateCircStr(p, and2c) + \cdot 1GateCircStr(p, and2a)$) を，出力線で直列合成したものである．

［定義 4.2］ **(レイヤ I+II 加算器の合成)**

$x^-, x^+, y^-, y^+, c_{in}, t^+$ を集合とする．このとき，*Many Sorted Signature* の下で，レイヤ I 加算器の構造 $BitRSDStr(x^-, x^+, y^-, y^+, c_{in}, t^+)$ は，以下の様に定義される．

$$BitRSDStr(x^-, x^+, y^-, y^+, c_{in}, t^+)$$
$$= BitRSD1Str(x^-, x^+, y^-, y^+, c_{in})$$
$$+\cdot\, 1GateCircStr(\langle t^+,$$
$$BitRSD1Output^-(x^-, x^+, y^-, y^+, c_{in}) \rangle, and2c)$$
$$+\cdot\, 1GateCircStr(\langle t^+,$$
$$BitRSD1Output^-(x^-, x^+, y^-, y^+, c_{in}) \rangle, and2a)$$

回路 $BitRSDCirc(x^-, x^+, y^-, y^+, c_{in}, t^+)$ は，回路の構造が *BitRSDStr* である *Many Sorted Signature* の下で，以下の様に定義される．

$$BitRSDCirc(x^-, x^+, y^-, y^+, c_{in}, t^+)$$
$$= BitRSD1Circ(x^-, x^+, y^-, y^+, c_{in})$$

$+\cdot 1GateCircuit(t^+,$
$BitRSD1Output^-(x^-,x^+,y^-,y^+,c_{in}),and2c)$
$+\cdot 1GateCircuit(t^+,$
$BitRSD1Output^-(x^-,x^+,y^-,y^+,c_{in}),and2a)$

4.6 回路の入出力信号に関する定理

上で定義した演算回路の定義を基に, 各回路の内部状態, 入出力信号である *carrier, InputVertices, InnerVertices* に関する定理を証明する.

［定理 4.4］ **(レイヤ I+II 加算器に関する諸定理)**

$x^-,x^+,y^-,y^+,c_{in},t^+$ を集合とする．このとき，レイヤ *I+II* 加算器に関する以下の定理が成立する.

$(\forall x^-,x^+,y^-,y^+,c_{in},t^+)($
$(InnerVertices(BitRSDStr(x^-,x^+,y^-,y^+,c_{in},t^+))$
は *Relation* である)
かつ *the carrier of* $BitRSDStr(x^-,x^+,y^-,y^+,c_{in},t^+) =$
$\{x^-,x^+,y^-,y^+,c_{in},t^+\}\cup$
$\{[\langle x^-,x^+\rangle,xor2c],GFA2AdderOutput(x^-,x^+,y^-)\}\cup$
$\{[\langle x^-,x^+\rangle,and2c],[\langle x^+,y^-\rangle,and2a],[\langle y^-,x^-\rangle,and2],$
$GFA2CarryOutput(x^-,x^+,y^-)\}\cup$
$\{[\langle GFA2AdderOutput(x^-,x^+,y^-),c_{in}\rangle,xor2c],$
$GFA1AdderOutput(GFA2AdderOutput(x^-,x^+,y^-),$
$c_{in},y^+)\}\cup$
$\{[\langle GFA2AdderOutput(x^-,x^+,y^-),c_{in}\rangle,and2c],$
$[\langle c_{in},y^+\rangle,and2a],$
$[\langle y^+,GFA2AdderOutput(x^-,x^+,y^-)\rangle,and2],$
$GFA1CarryOutput(GFA2AdderOutput(x^-,x^+,y^-),$
$c_{in},y^+)\}\cup$
$\{[\langle t^+,BitRSD1Output^-(x^-,x^+,y^-,y^+,c_{in})\rangle,and2c],$
$[\langle t^+,BitRSD1Output^-(x^-,x^+,y^-,y^+,c_{in})\rangle,and2a]\}$

かつ $InputVertices(BitRSDStr(x^-,x^+,y^-,y^+,c_{in},t^+)) =$

$\{x^-,x^+,y^-,y^+,c_{in},t^+\}$

4.7 動作の安定性の証明

演算回路の定義と種々の定理を基に，レイヤ I+II 加算器 $BitRSDStr(x^-,x^+,y^-,y^+,c_{in},t^+)$ に関する，動作の安定性の証明を行う.

［定理 4.5］ **(レイヤ I+II 加算器の 5 ステップ動作後)**

$x^-,x^+,y^-,y^+,c_{in},t^+$ を集合とする．入力信号点 $x^-,x^+,y^-,y^+,c_{in},t^+$ は，回路の内部信号点とは異なることを前提条件とする．このとき，s を レイヤ *I+II* 加算器 $BitRSDCirc(x^-,x^+,y^-,y^+,c_{in},t^+)$ の信号点の状態，*Boolean* の要素であるような a_1,a_2,a_3,a_4,a_5,a_6 を $a_1=s(x^-)$, $a_2=s(x^+)$, $a_3=s(y^-)$, $a_4=s(y^+)$, $a_5=s(c_{in})$, $a_6=s(t^+)$ とするとき, *5* ステップ（*4+1* ステップ）動作後の状態のうち，ある出力信号点（z^+）に着目した場合，以下の値を得る.

$(\forall x^-,x^+,y^-,y^+,c_{in},t^+)(\forall s)(\forall a_1,a_2,a_3,a_4,a_5,a_6)($
$Following(s,5)(BitRSDOutput^+(x^-,x^+,y^-,y^+,c_{in},t^+))$
$= a_6\wedge(not(a_1)\oplus a_2\oplus not(a_3)\oplus a_4\oplus not(a_5)$

もうひとつの出力信号点（z^-）に着目した場合にも，同様の値を得る（省略）.

［定理 4.6］ **(レイヤ I+II 加算器の安定性)**

$x^-,x^+,y^-,y^+,c_{in},t^+$ を集合とする．入力信号点 $x^-,x^+,y^-,y^+,c_{in},t^+$ は，回路の内部信号点とは異なることを前提条件とする．このとき，s を レイヤ *I+II* 加算器 $BitRSDCirc(x^-,x^+,y^-,y^+,c_{in},t^+)$ の信号点の状態として，回路は，5 ステップ動作した後，安定である．つまり，

$(\forall x^-,x^+,y^-,y^+,c_{in},t^+)(\forall s)(Following(s,5)\ is\ stable)$

【証明は省略. [20] 定理 20 を用いる．4+1 ステップで安定.】

5. むすび

「ロジック」の問題に関して，演算回路を数学的定義に基づいて設計し，設計検証と動作の正しさをプルーフチェッカー（Proof Checker）を用いて証明する方策を導入した．検証対象となる演算回路の実例としては，冗長 2 進数表現を用いたキャリー先送り型の加算演算回路を取り挙げ，その回路定義と諸定理，更に動作の正しさと安定性に関するの証明を行った．これらの定義・定理の証明を，*Mizar* プルーフチェッカーにより検証した．今後の課題として，n-bit の Carry 先読み型，Carry スキップ型回路などの設計検証，更には Wallence-Tree 型乗算器や Montgomery 高速乗算器などの設計検証も行い，RSA 暗号処理などを高速に行うディジタル回路や DSP プロセッサの ALU に必要な，様々な演算回路について検証を進めていくことが挙げられる.

文　献

[1] 市川, 加藤, 後藤, “循環パイプラインアーキテクチャー Pipelined MIMD の試み,” 情報処理学会第 37 回全国大会講演論文集, no.4, N-4, 1988.

[2] L.Kleeman and A.Cantoni, “Can Redundancy and Masking Improve the Performance of Synchronizers?, ” *IEEE Trans. Computers*, vol.C-35, no.7, pp.643-646, 1986.

[3] K.E.Stoffers, “Test Sets for Combinational logic – The Edge – Tracing Approach,” *IEEE Trans. Computers*, vol.C-29, no.8, pp.741-746, 1980.

[4] C.E.Strangio, “DIGITAL ELECTRONICS: Fundamental Concepts and Applications,” *Prentice-Hall Inc.*, pp.355-385, 1980.

[5] S.H.Unger and C.J.Tan, “Clocking Schemes for High-Speed Digital Systems,” *IEEE Trans. Computers*, vol.C-35, no.10, pp.880-895, 1986.

[6] 石浦, 高橋, 矢島, “論理回路の正確なタイミング検証のための時間記号シミュレーション,” 情報処理学会論文誌, vol.31, no.12, pp.1832-1839, 1990.

[7] 小倉, “CPLD/FPGA の開発手法,” The First Japanese FPGA/PLD Design Conference 技術講座 4, pp.46-62, 1993.

[8] 中村, 西山, 不破, “多段順序回路における信号伝搬のタイミング検証システム,” 電子情報通信学会論文誌 (A) , vol.J75-A, no.12, pp.1826-1836, 1992.

[9] 西山, 不破, 江口, 中村, “クロックモデルによるデジタル回路のタイミング検証法,” 電子情報通信学会技術研究報告, CAS93-65, pp.73-80, 1993.

[10] T.Nishiyama and Y.Mizuhara, “Binary Arithmetics,” *Formalized Mathematics*, vol.4, no.1, pp.83-86, 1993.

[11] A.Trybulec, “Many-sorted Sets,”, *Formalized Mathematics*, vol.4, no.1, pp.15-22, 1993.

[12] A.Trybulec, “Many Sorted Algebras,” *Formalized Mathematics*, vol.5, no.1, pp.37-42, 1996.

[13] Y.Nakamura, P.Rudnicki, A.Trybulec and P. N. Kawamoto,

"Preliminaries to Circuits, I.," *Formalized Mathematics*, vol.5, no.2, pp.167-172, 1996.
[14] Y.Nakamura, P.Rudnicki, A.Trybulec and P. N. Kawamoto : "Preliminaries to Circuits, II.,", *Formalized Mathematics*, vol.5, no.2, pp.215-220, 1996.
[15] Y.Nakamura, P.Rudnicki, A.Trybulec and P. N. Kawamoto, "Introduction to Circuits, I.," *Formalized Mathematics*, vol.5, no.2, pp.227-232, 1996.
[16] Y.Nakamura, P.Rudnicki, A.Trybulec and P. N. Kawamoto, "Introduction to Circuits, II.," *Formalized Mathematics*, vol.5, no.2, pp.273-278, 1996.
[17] Y.Nakamura and G.Bancerek, "Combining of Circuits," *Formalized Mathematics*, vol.5, no.2, pp.283-295, 1996.
[18] G.Bancerek and Y.Nakamura, "Full Adder Circuit. Part I.," *Formalized Mathematics*, vol.5, no.3, pp.367-380, 1996.
[19] K.Wasaki and P.N.Kawamoto, "2's Complement Circuits. Part I.," *Formalized Mathematics*, vol.6, no.2, pp.189-197, 1996.
[20] B.Grzegorz, S.Yamaguchi and Y.Shidama, "Combining of Multi Cell Circuits," *Formalized Mathematics*, vol.10, no.1, pp.47-64, 2002.
[21] S.Yamaguchi, K.Wasaki and N.Shimoi, "Generalized Full Adder Circuits (GFAs). Part I.," *Formalized Mathematics*, vol.13, no.4, pp.549-571, 2005.
[22] A.Avizienis, "Signed-Digit Number Representation for Fast Parallel Arithmetic," *IRE Trans. Electronic Computers*, vol.10, pp.389-400, 1961.
[23] A.Vandemeulebroecke, E.Vanzieleghem, T.Denayer and P.G.A.Jespers, "A New Carry-Free Division Algorithm and its Application to a Single-Chip 1024-b RSA Processor," *IEEE J. Solid-State Circuits*, vol.25, no.3, pp.748-755, 1990.
[24] E.Savas, "A Carry-Free Architecture for Montgomery Inversion," *IEEE Trans. Computers*, vol.54, no.12, pp.1508-1519, 2005.
[25] E.Bonarska, "An Introduction to PC Mizar," *Mizar Users Group, Fondation Philippe le Hodey*, Brussels, 1990.
[26] A.Trybulec and P.Rudnicki, "A Collection of TEXed Mizar Abstracts," *University of Alberta*, Canada, 1989.
[27] Mizar Proof Checker : `http://mizar.org/`
[28] Z.Trybulec and H.Swileczkowska, "Boolean Properties of Sets," *Formalized Mathematics*, vol.1, no.1, pp.17-23, 1990.
[29] A.Trybulec, "Enumerated Sets," *Formalized Mathematics*, vol.1, no.1, pp.25-34, 1990.
[30] C.Byliński, "Functions and Their Basic Properties," *Formalized Mathematics*, vol.1, no.1, pp.55-65, 1990.