論　文

入力名標集合と制御条件を用いたレジスタ転送レベルの設計検証法

吉田たけお[†]　　貴家　仁志[†]　　内藤　祥雄[†]

Design Verification for Register Transfer Level Design by Input Label Sets and Control Conditions

Takeo YOSHIDA[†], Hitoshi KIYA[†], and Sachio NAITO[†]

あらまし　ハードウェア記述言語(HDL)による記述中には，計算機で計算可能な関数として知られている帰納的関数(Church の提唱)が記述されていると考えられる．このとき，設計記述が正しいための必要条件として，仕様記述が表す帰納的関数とこれに対応する設計記述が表す帰納的関数とが等価であることが挙げられる．これらの帰納的関数が等価でない場合，設計記述には，演算の誤りが存在すると言われる．しかし，この必要条件を厳密に調べることは困難である．本論文では，演算の誤りの一部である，システムの入出力の依存関係の誤りを検出する検証法を提案する．そのために，本論文では，HDL 記述が表すシステムの入力名標の集合と制御条件を用いた“機能”という概念を用いて正しい設計記述について考察する．本方法は，レジスタ転送レベル(RTL)を対象とした高レベルの検証法である．

キーワード　ハードウェア記述言語(HDL)，設計検証，帰納的関数，依存入力集合，パス・コンディション

1. まえがき

ディジタルシステムの大規模・複雑化に伴い，記述能力に優れたさまざまなハードウェア記述言語(HDL) [1]〜[6] が開発されてきた．HDLには，以下のように，ディジタルシステムの各設計段階に対応した言語が存在する [7].

方式設計：　システムレベル，インストラクションレベル

機能設計：　レジスタ転送レベル(RTL)

論理設計：　ゲートレベル

また，論理合成技術の急速な進歩により，機能設計の記述から回路のマスクパターンを自動生成することが可能となった [8]．このため，機能設計段階(すなわち，RTL)での設計検証法の確立が重要な課題となっている．本論文では，方式設計段階のHDL記述を仕様記述，機能設計段階のHDL記述を設計記述と呼び，これら記述レベルの異なる二つの記述間で，設計記述が，仕様記述を満たしているか否かを検証する一手法を提案する．

これまで提案されてきた設計検証法 [9]〜[11] のほとんどは，RTLより下位のレベルを対象としたものである．また，RTLを対象とした高レベルの検証法 [12]〜[15] も既にいくつか提案されているが，これらの検証法では，システムの入出力の依存関係を考慮していない．これらの問題を解決するために，筆者らはRTLの設計検証法を検討してきた [16], [17]．更に，システムの入出力関係を論理式で表し，その論理式の真偽を定理証明系(Theorem Prover)を用いて調べる正規検証法 [18] や時相論理の一種であるCTL(Computation Tree Logic)を応用した検証法 [19] 等のように，演算の誤り(入出力の依存関係の誤りを含む)を検出可能な検証法も存在するが，一般に演算の誤りを考慮した検証は時間がかかり，大規模なシステムの設計の正当性を検証することは困難である．

ここで，演算の誤りと入出力の依存関係の誤りについて簡単に説明する．HDL記述が表しているシステムには，p 本の入力線および q 本の出力線があるものとする．これらの入力線がとる値の集合をそれぞれ $X_1, X_2, \cdots, X_p$，同様に，これらの出力線がとる値の集合を $Y_1, Y_2, \cdots, Y_q$ と表す．このとき，HDL記述中には，$Z_j = \prod_{i \in A_j} X_i$ (但し，$A_j \subseteq \{1, 2, \cdots, p\}$，また，

† 東京都立大学工学部，東京都
Faculty of Engineering, Tokyo Metropolitan University, Tokyo, 192-03 Japan

$\prod$は直積を表す）を定義域，$Y_j,\ j=1,2,\cdots,q$ を値域とする q 個の帰納的関数 [20] が記述されていると考えられる．一般に，仕様記述中の帰納的関数と，これに対応する設計記述中の帰納的関数とが等価でない場合，設計記述には演算の誤りが存在すると言う(注1)．これに対して，本論文では，仕様記述中のある帰納的関数の定義域が，この関数に対応する設計記述中の帰納的関数の定義域に包含されていない場合，設計記述中に入出力の依存関係の誤りが存在すると言う．但し，本論文では，仕様記述が表すシステムの各入力線がとる値の集合と，これらの集合に対応する，設計記述が表すシステムの各入力線がとる値の集合は，それぞれ等しいものと考え，これらの集合を変域とする変数の名前の集合を，関数の定義域として扱う．例えば，仕様記述中に "$y=f(a,b,c)$" という関数が記述されており，これに対応する関数として "$y=g(a,b,d)$" という設計をしたとする．ここで，a, b, c, d は，それぞれ，前述の $X_i,\ i=1,2,\cdots,p$ のいずれかを変域とする変数の名前である．同様に，y は，$Y_j,\ j=1,2,\cdots,q$ のいずれかを変域とする変数の名前である．この場合，仕様の関数 f の独立変数名の集合 $\{a,b,c\}$ は，設計の関数 g の独立変数名の集合 $\{a,b,d\}$ に含まれていない．本論文では，このような設計誤りを入出力の依存関係の誤りと言う．この入出力の依存関係が満たされていない場合，仕様記述中の関数が冗長な独立変数を含んでいなければ，演算の誤りが存在することになる．本論文では，入出力の依存関係の誤りを演算の誤りの一部であると考え，入出力の依存関係の誤りを検出する検証法を提案する．本方法は，演算誤りの検証作業の前段階に，演算誤りの検証作業の軽減を目的として行われる．

前述のような入出力の依存関係を調べるために，本論文では，入力名標の集合およびシステムの制御条件を組み合わせた "機能" という概念を用いる．ここで，名標とは，システムの入出力線や記憶要素（レジスタ，メモリ等）の名前（標，ラベル [21]）を言う．また，制御条件とは，記述中の制御文の制御条件である．

本論文では，RTLおよびそれより上位レベルの言語で記述された同期システムを対象とし，有向グラフを用いて，HDL記述をモデル化する．また，本論文では，記述が表すシステムの入出力動作，システム内部の記憶要素に対する書込み，読出し動作および制御条件に着目し，これら各動作の実行順序や並列性に関する情報をモデル化する．このため，本論文で提案するモデルを，より広い意味での演算の誤りや入出力文の実行順序等を考慮した検証法へ適用することも可能であると考えるが，今回は，前述の入出力の依存関係の誤りを検出するための情報（機能）だけをこのモデルから抽出する．また，モデル化する上記各動作は，どのようなHDLを用いた場合でも，明確に記述されている．すなわち，設計および仕様記述が共にRTLあるいはそれより上位レベルであり，かつ，同期システムを表す記述であれば，本論文で提案する検証法を適用できる．

まず **2.** で，検証しようとするシステムの動作を本論文の検証目的に合わせて抽象化したシステム（これを BAS と呼ぶ）を示し，その上で，本論文における検証問題を明確化する．また，HDL記述の検証問題をこの明確化された検証問題に帰着させるために，HDL記述を有向グラフによりモデル化し，この有向グラフとBASとの対応について説明する．この対応に従って，有向グラフ上で，検証対象となる機能を定義する．**3.** では，有向グラフ上で定義した機能の導出方法について述べる．また，本論文で提案する記述のモデル化例，検証例を **4.** に示す．

2. 検証モデルの定義と設計検証

ここでは，まず，検証しようとするシステムの動作を入出力の依存関係に着目して抽象化したシステムBAS を定義する．次に，このBAS上で，本論文における検証問題の明確化を行う．また，HDL記述の検証問題をこの明確化された検証問題に帰着させるために，HDLによる記述を有向グラフを用いてモデル化し，この有向グラフとBASとの対応を説明する．更に，この有向グラフ上で，本論文での検証の基礎となるHDLの機能を定義する．

2.1 動作抽象化システム BAS

まず，本論文で考える動作抽象化システム BAS を定義する．ここで，システムのある入力線の名標を A としたとき，名標 A の入力線がとる値の集合を $\underline{A}$ と表すことにする．

［定義2.1］ 動作抽象化システム BAS(Behavioral Abstract System) の動作は，以下の 7 項組 $S(I,O,O',I_c,f_U,f_C,g)$ で定義される．

（1） I ：入力名標集合

$$I=\{I_1,I_2,\cdots,I_p\}$$

(注1)：実際には，関数の定義域や値域等を考慮する必要があるため，厳密な等価性を調べることは困難である．

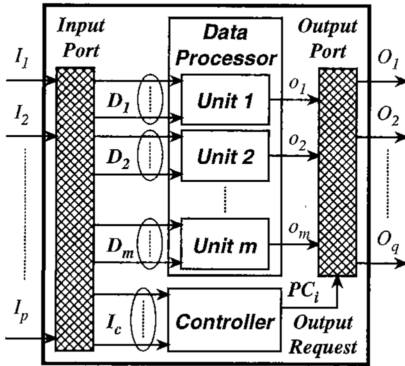


図1 動作抽象化システム (BAS)
Fig. 1 Behavioral abstract system (BAS).

（2） O ：出力名標集合

$O = \{O_1, O_2, \cdots, O_q\}$

（3） O' ：内部出力名標集合

$O' = \{o_1, o_2, \cdots, o_m\}$. 但し，$m \geq q$.

（4） I_c ：制御入力名標集合

$I_c = \{I_{c_1}, I_{c_2}, \cdots, I_{c_r}\} \subseteq I$

（5） f_U ：ユニット関数

$f_U : D_i \mapsto o_i$

但し，$D_i \subseteq I$, $o_i \in O'$, $i = 1, 2, \cdots, m$.

（6） f_C ：条件関数

$$f_C : \left(\prod_{I_{c_h} \in I_c} \underline{I_{c_h}} \right) \rightarrow O'$$

（7） g ：出力関数

$g : O' \rightarrow O$ □

BAS(図1) は，p 本の外部入力線(入力名標は，$I_1, I_2, \cdots, I_p$) と q 本の外部出力線(出力名標は，$O_1, O_2, \cdots, O_q$) を持ち，その内部はデータ処理部，制御部および入出力ポートに分けられる．データ処理部は，更に m 個のユニットに分けられる．以下，入力名標 I_j の外部入力線を単に外部入力線 I_j，同様に出力名標 O_k の外部出力線を外部出力線 O_k と呼ぶ．各外部入力線は，入力ポートでファンアウトされ，いくつかのユニットや制御部に接続される．ユニット i は，記憶要素を含む演算回路であり，演算結果をそのユニットの出力線(内部出力線と言う) o_i 上に出力する．出力ポートは，内部出力線を外部出力線に接続するものであり，その動作は，制御部によって制御される．ここで，各内部出力線は，ただ 1 本の外部出力線に接続され，各外部出力線には，1 本以上の内部出力線が接続される．但し，1 本の外部出力線に，複数の内部出力線が同時に接続されることはない．内部出力線 o_i が，外部出力線 O_k に接続されるとき，$[o_i] = O_k$ と表記する．また，データ処理部に接続されている外部入力線をデータ入力線と言い，制御部に接続されている外部入力線を制御入力線と言う．制御部はその内部に演算回路，記憶要素を含んでおり，各制御入力線はこれらに接続されている．この制御部内の演算回路の出力，すなわち各制御入力線の値によって定まる演算結果が，ユニット i の演算結果をシステムに出力させる制御条件を満たす場合，制御部は出力ポートに出力要求信号を送る．出力要求信号を受けた出力ポートは，内部出力線 o_i を定められた外部出力線 O_k $(= [o_i])$ に接続する．なお，制御部が，各ユニットの演算結果等を用いて分岐判断動作をする必要がある場合，制御部はユニットの演算結果を参照せず，制御部内の演算回路で必要な演算を行って，その結果を参照するものとしている．従って，1 本の外部入力線が，データ入力線，制御入力線の両方で用いられることもある．また，制御部および入出力ポートには遅延がなく，各ユニットが入力を受け取り，演算結果を内部出力線上に出力するまでに要する時間は，ユニットによるものとする．更に，ユニット i において，同一の外部入力線 I_j への異なる時刻における複数の入力信号を用いて演算が行われる場合，これらの入力信号を異なる外部入力線 $I_j^1, I_j^2, \cdots, I_j^{T_{ji}}$ への入力信号として扱う．

次に，本論文における仕様記述および設計記述を定義する．

［定義2.2］ ハードウェアの動作を表している，システムレベルまたはインストラクションレベルの HDL 記述を仕様記述と言う． □

［定義2.3］ 与えられた仕様記述が表しているハードウェアの動作を詳細化して得られる，レジスタ転送レベルの HDL 記述を設計記述と言う． □

以下に，本論文における前提を示す．

［前提2.1］

（1） 仕様および設計記述は，原始帰納的関数(注2) [20] を表しているものとする．

（2） 仕様記述中の各入出力名標に対応する設計記述中の入出力名標は，既知であるもとする．

（3） 仕様記述中の原始帰納的関数は，冗長な独立変数を含んでいないものとする． □

上記前提の（1）は，仕様および設計記述が表している関数は，全域的関数[20]であり，かつ，その関数値を求めるアルゴリズムが存在する必要があることを意味する．この前提により，while文(if-then-else文等の他の制御文で実現されたwhile文も含む)を使用しなければ記述できない帰納的関数をHDL記述中に記述できないことになる．上位レベルの設計検証という立場において，この前提は，大きな欠点となる．しかし，多くの実用的なシステムがwhile文を使用せずに記述可能であること，また，while文を含んだ記述を扱えるように拡張が可能である(注3)ことから，今回この前提を設けた．

また，**1.** で簡単に述べたように，仕様記述中の原始帰納的関数が冗長な独立変数を含んでいる場合は，本検証法を適用できないことになる．このため，（3）の前提を設けた．

次に，BAS上での正しい設計記述について述べる．まず，以下の議論で必要となる二つの用語を定義する．

［定義2.4］ BASのユニット i に接続されているデータ入力線の名標すべての集合，すなわち，BASのユニット関数 f_U において，$o_i(\in O')$ を像とする，I の部分集合 D_i を内部出力線 o_i の依存入力集合と言う． □

［定義2.5］ BASのユニット i の内部出力線 o_i が外部出力線に接続されるための制御条件，すなわち，BASの条件関数 f_C において，$o_i(\in O')$ を像とする，

$\prod_{I_{c_h}\in I_c} \underline{I_{c_h}}$ の要素すべての集合を内部出力線 o_i のパス・コンディションと言い，PC_i と表す． □

本論文では，以下の二つの要件を同時に満たす設計記述を，仕様記述に対して正しいとする．

［要件2.1］(設計記述が満たすべき要件(注4))

（1） 仕様記述が表すシステム中のある内部出力線を os_i ($[os_i]=O_k$) およびその依存入力集合を Ds_i とする．このとき，設計記述が表すシステム中の少なくとも一つの内部出力線 od_g ($[od_g]=O_k$) の依存入力集合が Ds_i を含む．

（2） 上記 os_i のパス・コンディション PCs_i の任意の要素を，設計記述が表すシステムの制御入力線に印加したとき，上記（1）の要件を満たす od_g の一つから，演算結果を外部出力線 O_k に出力する． □

2.2 有向グラフによる記述のモデル化

HDL記述の検証問題を，**2.1** で明確化した検証問題に帰着させるためには，与えられた仕様および設計記述が表すハードウェアの動作をBASの動作に対応させる必要がある．そのために，HDL記述を記述グラフと呼ばれる有向グラフに変換し，この記述グラフをBASに対応させる．

ここでは，まず記述グラフの表現について説明し，その後，HDL記述を記述グラフに変換するための手順を示す．なお，この記述グラフは，グローバルクロックを仮定した同期システムのためのモデルである．

2.2.1 記述グラフの表現

記述中の実行文は，代入文(入出力文を含む)と制御文(if-then-else文，for文，while文，goto文等)に分けられる．本論文では，記述中の各代入文を(有向)弧に，制御文の各制御条件をノードに対応させる．また，一つのHDL記述から得られる記述グラフを一つの連結グラフとして表すために，何も処理を実行しないことを表す弧および何も条件を表していないノードも用い，これらをそれぞれ無処理弧および無条件ノードと呼ぶ．この無処理弧および無条件ノードは，goto文を表すためにも用いられる．図2は，記述グラフの例である．図2中の括弧内の文字は，ノードおよび弧の名前を表している．この記述グラフを説明するために，以下のような表記法を導入する．記述グラフ中のあるノード n_a に対して，n_a へのすべての入力弧の集合を E_{in}，n_a からのすべての出力弧の集合を E_{out} としたとき，これらのノードと弧の集合との接続関係をそれぞれ $In(n_a,E_{in})$, $Out(n_a,E_{out})$ と表す．

記述グラフ中のノード n_c の接続関係を $In(n_c,E_{in})$, $Out(n_c,E_{out})$ と表したとき，E_{in} および E_{out} 中の各弧の(表す)代入文の実行順序は，弧の矢印の向きによって表され，実行の継続と中断は，ノード n_c の(表す)制御条件によって制御される．また，各弧の代入文は，同期システムで用いられるクロックに同期して実行されるものとする．$E_{in}=\{e_{in}\}$ である場合，弧 e_{in} の代入文が実行され，かつ，ノード n_c の制御条件が満たされるときのみ，集合 E_{out} 中の弧の代入文

(注2)：帰納的関数および原始帰納的関数の計算過程は，一つのレジスタ(メモリ)の内容を，何らかの規則に基づいて繰返し更新していく動作として表される．但し，原始帰納的関数の場合は，繰返しの開始時に，その繰返し回数が定まっているのに対し，帰納的関数の場合は，繰返し回数が定まっていないという違いがある．

(注3)：while文を含んだ記述については，**5.** で簡単に述べる．

(注4)：この二つの要件は，設計記述が仕様記述に対して正しいための必要条件であり，十分条件ではない．

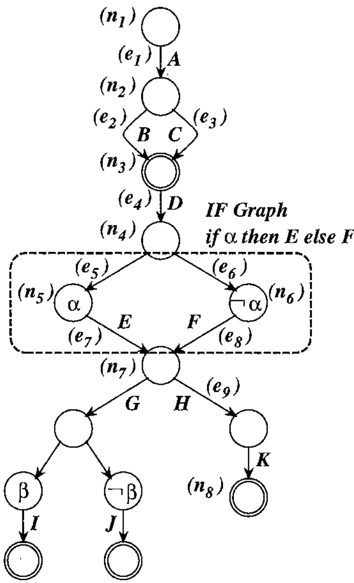


図2 記述グラフの例
Fig. 2 Example of description graph.

表1 弧とノードの種類
Table 1 Edges and nodes.

名称	意味または構造
無処理弧	意味： 何も代入文を実行しない
処理弧	意味： (その弧の表す)代入文を実行する
無条件ノード	意味： 無条件で次の実行文を実行する
条件ノード	意味： (そのノードの持つ)制御条件が満たされるときのみ，次の実行文を実行する
∃-ノード	意味： 少なくとも1本の入力弧の代入文を実行後，次の実行文を実行する
∀-ノード	意味： すべての入力弧の代入文がを実行された後，次の実行文を実行する (二重丸のノードで表す)
開始ノード	意味： 記述の開始点を表すノード 構造： 入力弧数 = 0
終了ノード	意味： 処理の終了を表すノード 構造： 出力弧数 = 0 (二重丸のノードで表す)

の処理が，すべて同時に開始(並列処理)される．例えば，図2の弧 e_5, e_7 およびノード n_5 において，代入文 E は，弧 e_5 (代入動作を実行しない)が実行され，かつ，制御条件 α が満たされるときのみ実行される．同様に，図2において，代入文 H が実行されると，代入文 K は無条件で実行される．また，E_{in} の要素数が複数である場合，E_{in} および E_{out} 中の各弧の代入文に対して，二つの実行形式を考える．一つは，E_{in} 中のすべての弧の代入文が実行されてから，E_{out} 中の各弧の代入文を実行する場合である．この実行形式を ∀-実行と呼ぶ．もう一つの実行形式は，集合 E_{in} 中の弧の代入文のいずれか一つが実行されると，E_{out} 中の各弧の代入文を実行する場合である．この実行形式を ∃-実行と呼ぶ．本論文では，これらの実行形式の違いをノードによって区別する．∀-実行を表すためのノードを2重丸で表し，これを ∀-ノードと呼ぶ．例えば，ノード n_3 は代入文 B, C の両方が実行されなければ，代入文 D を実行できないことを表している．また，∃-実行を表すためのノードを1重丸で表し，∃-ノードと呼ぶ．例えば，ノード n_7 は代入文 E, F の一方が実行されれば，代入文 G および H を実行できることを表している．

記述グラフ中の弧およびノードは，意味論上それぞれ2種類に分類される．弧は，無処理弧(例えば，図2の弧 e_5)と処理弧(例えば，弧 e_1)，ノードは，無条件ノード(例えば，ノード n_2)と条件ノード(例えば，ノード n_5)にそれぞれ分けられる．更に，特別な意味と構造を持つ無条件ノードとして，開始ノードと終了ノードが存在するが，これらについては，後述する．これらの弧およびノードの分類を表1に示す．

2.2.2 記述グラフの導出

次に，HDL記述から記述グラフを導出する手順を示すが，記述に対して以下のことを仮定する．まず，記述中に，冗長な実行文が記述されていないものとする．次に，制御文はすべてif-then-else文(以下，if文と略記)およびgoto文に変換されており，なおかつ，記述中にはいかなるループ[21]も含まれていないものとする．ここで，ループとは，「ある条件が満たされている間，繰り返し実行される文の集合」である．先に示した［前提2.1］の（1）より，与えられた記述をループを含まない記述に変換可能である．例えば，「for $Num := 1$ to X」をif文によって表すためには，X のとる値の上限が既知 (U_X) であれば，U_X 個のif文を並べることによって解決できる．

記述グラフの導出手順は，① 記述中の各実行文を弧またはIFグラフへ変換，② これらの弧およびIFグラフを接続，③ 不必要な無処理弧および無条件ノードを簡約，の三つに分けられる．ここで，IFグラフとは，記述中のif文に対応する有向グラフ(注5)であり，以下で定義される．

［定義2.6］ if文 "if α then M else N" の制御条件 α および $\neg\alpha$ をもつ条件ノードをそれぞれn_{c_1}, n_{c_2} とする．また，代入文 (または,goto文) M, N を表す処理弧 (goto文の場合は，無処理弧) をそれぞれe_{o_1}, e_{o_2} とする．更に，2 本の無処理弧を e_{no_1}, e_{no_2} (これらをIF無処理弧と呼ぶ) とする．このとき，以下のような接続関係をもつ有向グラフをこのif文のIFグラフと言う．

$In(n_{c_1}, \{e_{no_1}\})$, $Out(n_{c_1}, \{e_{o_1}\})$,
$In(n_{c_2}, \{e_{no_2}\})$, $Out(n_{c_2}, \{e_{o_2}\})$ □

IFグラフの例を図2(枠で囲んだ部分)に示す．また，以下の手順で必要となる用語を定義する．

［定義2.7］ $A_1, A_2, \cdots, A_c$ を互いに ∀-実行される HDL 記述中の代入文とし，これらの代入文に対応する記述グラフ中の弧を $e_1, e_2, \cdots, e_c$ と表す．このとき，この中の任意の 2 本の弧 e_i, e_j の間には，∀-収れんの関係があると言う．なお，IF 無処理弧以外の任意の弧 e_h に対して，弧 e_h は，e_h 自身と∀-収れんの関係にあると定義する． □

この ∀-収れんの関係は，明らかに同値関係である．

以上の準備のもとで，記述グラフの導出手順を示す．なお，手順中の表現 $|E|$ は，集合 E の要素数を表す．また，図3 は，［手順2.1］(9) の簡約規則を表す．

［手順2.1］(記述グラフの導出手順)

(1) HDL 記述中の各代入文 A_i を処理弧 e_{A_i} に変換し，この処理弧 e_{A_i} に対して，$Out(n_{A_i}, \{e_{A_i}\})$ となる∃-ノード n_{A_i} を描く．このノード n_{A_i} を代入文 A_i の入力ノードと呼ぶ；

(2) HDL 記述中の各goto文 J_j を無処理弧 e_{J_j} に変換し，この無処理弧 e_{J_j} に対して，$Out(n_{J_j}, \{e_{J_j}\})$ となる ∃-ノード n_{J_j} を描く．このノード n_{J_j} をgoto文 J_j の入力ノードと呼ぶ；

(3) HDL 記述中の各 if 文 C_k を IF グラフに変換する．この IF グラフ中の 2 本のIF無処理弧をそれぞれ，$e_{C_{k_1}}, e_{C_{k_2}}$ としたとき，$Out(n_{C_k}, \{e_{C_{k_1}}, e_{C_{k_2}}\})$ となる ∃-ノード n_{C_k} を描く．このノード n_{C_k} を if 文 C_k の入力ノードと呼ぶ；

(4) IF無処理弧を除くすべての弧の集合を E と表し，集合 E を同値関係 "∀ - 収れん" によって分割する．このとき，それらの各ブロックを $G_1, G_2, \cdots, G_c$ と表す；

(5) ブロック G_r, $r = 1, 2, \cdots, c$ に対して，$In(n_{s_r}, G_r)$ となる ∀-ノード n_{s_r} を描く．このノード n_{s_r} をブロック G_r の出力ノードと呼ぶ；

(6) ブロック $G_r, r = 1, 2, \cdots, c$ に属す各弧の実行文の次に実行されるすべての実行文を求め，これらの実行文すべての入力ノードの集合を N_r とする．各ブロック G_r の出力ノード n_{s_r} から N_r 中のすべてのノード $n_{r_h}, h = 1, 2, \cdots, f$ へ無処理弧e_{r_h} を描く；

(7) 上記(1)～(6)で描いた有向グラフ中のあるノード n_e に対して，$In(n_e, E_{in_e})$ なる接続関係が存在したとする．このとき，E_{in_e} 中の各弧の始点に接続されているすべてのノードの集合を，ノード n_e の前置ノード集合と呼び，N'_e と表す．上記(1)～(3)で描いたすべての入力ノードに対してこの前置ノード集合を求める；

(8) 上記(1)～(3)で描いた入力ノード n_b (接続関係 $In(n_b, E_{in_b})$, $Out(n_b, E_{out_b})$) の前置ノード集合 N'_b と等しい前置ノード集合をもつノードを n_{b_l}, $l = 1, 2, \cdots, g$ (接続関係$In(n_{b_l}, E_{in_{b_l}})$, $Out(n_{b_l}, E_{out_{b_l}})$) とする．このとき，これらのノードを唯一のノードn_b に代表させ，接続関係を $In(n_b, E_{in_b})$, $Out(n_b, E_{out_b} \cup E_{out_{b_1}} \cup \cdots \cup E_{out_{b_g}})$ と変更する；

(9) 無処理弧 e_d の始点にノード n_d が，終点にノード n_{ds} が接続されていたとする．また，ノード n_d が，$In(n_d, E_{in_d})$, $Out(n_d, E_{out_d})$, ノード n_{ds} が，$In(n_{ds}, E_{in_{ds}})$, $Out(n_{ds}, E_{out_{ds}})$ ($e_d \in E_{out_d}, e_d \in E_{in_{ds}}$) なる接続関係をもっていたとする．このとき，以下に従って，グラフの簡約を行う．

(a) n_d が ∀-ノードであり，かつ，$|E_{out_d}| = |E_{in_d}| = 1$ である場合，ノード n_{ds} の接続関係を $In(n_{ds}, (E_{in_{ds}} - \{e_d\}) \cup E_{in_d})$, $Out(n_{ds}, E_{out_{ds}})$ と変更し，弧 e_d およびノード n_d を削除する (図3(a))；

(b) n_{ds} が無条件ノードであり，かつ，$|E_{in_{ds}}| = 1$ である場合，ノード n_d の出力弧の接続関係を $Out(n_d, (E_{out_d} - \{e_d\}) \cup E_{out_{ds}})$ と変更し，弧 e_d およびノードn_{ds} を削除する (図3(b))；

(c) n_d が無条件の ∃-ノード，n_{ds} が ∃-ノードであり，かつ，$|E_{out_d}| = 1$ である場合，ノードn_{ds} の入力弧の接続関係を $In(n_{ds}, (E_{in_{ds}} - \{e_d\}) \cup E_{in_d})$ と変更し，弧 e_d およびノード n_d を削除する (図3(c))； □

ここで，開始ノードおよび終了ノードについて述べる．上記手順によって得られた，記述グラフ (例：図2) において，$In(n_s, \phi)$ となるノードn_s を開始ノー

(注5)：IFグラフは，両端にノードが接続されていない弧を含むので厳密には有向グラフではない．

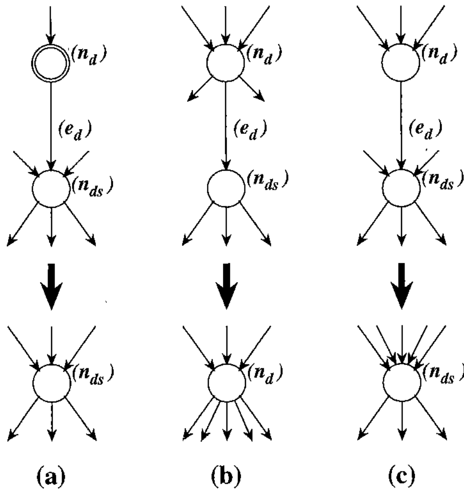


図3 有向グラフの簡約規則
Fig. 3 Reduction rules for directed graph.

ド(ノード n_1)と言う．開始ノードは，記述の開始点を表すノードである．HDL 記述中に冗長な実行文が記述されていない場合，開始ノードは，記述グラフ中にただ一つ存在する．また，$Out(n_e, \phi)$ となるノード n_e を終了ノードと言う．終了ノード(例えば，ノード n_8)は，処理の終了を表すノードであり，一般に複数存在する．この終了ノードは，∀-ノードの一種とみなすことにし，2重丸で表す．

2.3 機能の定義と設計検証

ここでは，**2.1** で示した動作抽象化システム BAS と **2.2** で示した記述グラフとの対応について説明し，検証対象となる“機能”を定義する．更に，HDL記述の検証問題を，**2.1** で明確化した検証問題に帰着させ，記述グラフ上での正しい設計について述べる．

まず，グラフ表現に関する用語を定義する．

［定義2.8］ 記述グラフ中の任意の弧 e_s に対して，開始ノード n_1 から弧e_s へ至るノードと弧の系列 $(n_1\ e_1\ \cdots\ n_s\ e_s)$ を弧 e_s のパスと言う． □

記述グラフ中のある弧 e_s のパスは，一般に複数存在する．しかし，記述中にはいかなるループも含まれていないという仮定より，本論文における記述グラフは有向閉路を含まず，任意の弧 e_s のパスの数は有限となる．

パスは，HDL 記述中の実行文の実行順序を表している．記述グラフ中のある弧 e_s のパス中に ∀-ノードが存在することは，弧 e_s の実行文を処理する前に，並列処理される実行文が存在することに対応する．このように，弧 e_s のパスのうち，並列処理されるパスの集合をパスブロックと呼ぶことにする．一般に，弧 e_s のパスブロックも複数存在する．

以下の手順は，記述グラフ中の弧 e_s のパスブロックを導出するための記号列 S を生成する手順である．以下の手順によって得られる記号列S は，弧およびノードの名前 (文字記号と呼ぶ), “*” および “+” の 4 種類の記号を用いて構成され，$[s_{11} * s_{12} * \cdots * s_{1h_1}] + [s_{21} * s_{22} * \cdots * s_{2h_2}] + \cdots + [s_{u1} * s_{u2} * \cdots * s_{uh_u}]$ のような形をしている．なお，手順中の記号列を見易くするために，“[” および “]” を用いて，記号列 S を区切る．上記の記号列 S の部分記号列 s_{ij} は，文字記号のみからなり，文字部分記号列と呼ばれる．文字部分記号列 s_{ij} に対して，(s_{ij}) は，弧 e_s の一つのパスを表す．また，記号列 S の “+” で区切られた記号列 $[s_{i1} * s_{i2} * \cdots * s_{ih_i}]$ を ∀-部分記号列と呼び，S_i と表す．ここで，記号 “*” は，並列処理されるパスを表すための記号である．例えば，∀-部分記号列 S_i に対して，弧 e_s のパス $(s_{i1}), (s_{i2}), \cdots, (s_{ih_i})$ は，並列処理されることを表す．すなわち，これらのパスの集合 OB_i は，弧 e_s の一つのパスブロックとなる．なお，記号 “+” は，パスブロックを区別するための記号であり，∀-部分記号列 S_i の表すパスブロック中のパスと ∀-部分記号列 S_j $(i \neq j)$ の表すパスブロック中のパスは，並列処理されない．

［手順2.2］（弧 e_s のパスブロック導出手順）

（1） 弧 e_s の始点に接続されているノードを n_s とする．このとき，記号列 $[n_s\ e_s]$ を S とする；

（2） ∀-部分記号列 S_i 中の各文字部分記号列 s_{ij} において，s_{ij} の左端の文字記号が表しているノードを n_{ij} (接続関係$In(n_{ij}, E_{ij})$) とし，$E_{ij} \neq \phi$ であるすべての s_{ij} に対して，以下を行う．

- 集合 E_{ij} の要素を $e_a, e_b, \cdots, e_c$, これらの弧の始点に接続されているノードをそれぞれ，$n_a, n_b, \cdots, n_c$ とし，n_{ij} が ∀-ノードである場合は，(a) を，∃-ノードである場合は，(b) を行う．

(a) ∀-部分記号列 S_i を記号列 $[s_{i1} * s_{i2} * \cdots * n_a\ e_a\ s_{ij} * n_b\ e_b\ s_{ij} * \cdots * n_c\ e_c\ s_{ij} * \cdots * s_{ih_i}]$ に置き換える；

(b) ∀-部分記号列 S_i を記号列 $[s_{i1} * s_{i2} * \cdots * n_a\ e_a\ s_{ij} * \cdots * s_{ih_i}] + [s_{i1} * s_{i2} * \cdots * n_b\ e_b\ s_{ij} * \cdots * s_{ih_i}] + \cdots + [s_{i1} * s_{i2} * \cdots * n_c\ e_c\ s_{ij} * \cdots * s_{ih_i}]$ に置き換える；

（3） すべてのノード n_{ij} の接続関係が $In(n_{ij}, \phi)$

となるまで，上記（２）を繰り返す；

（４） 得られた記号列 S が，$S_1 + S_2 + \cdots + S_u$，各 ∀-部分記号列 S_i が $[s_{i1} * s_{i2} * \cdots * s_{ih_i}]$ であるとする．このとき，各集合 $OB_i = \{(s_{i1}), (s_{i2}), \cdots, (s_{ih_i})\}$, $i = 1, 2, \cdots, u$ を弧 e_s に対する i 番目パスブロックとする； □

例えば，図2 の弧 e_9 のパスブロックは，$\{(n_1\ e_1\ n_2\ e_2\ n_3\ e_4\ n_4\ e_5\ n_5\ e_7\ n_7\ e_9), (n_1\ e_1\ n_2\ e_3\ n_3\ e_4\ n_4\ e_5\ n_5\ e_7\ n_7\ e_9)\}$ および $\{(n_1\ e_1\ n_2\ e_2\ n_3\ e_4\ n_4\ e_6\ n_6\ e_8\ n_7\ e_9), (n_1\ e_1\ n_2\ e_3\ n_3\ e_4\ n_4\ e_6\ n_6\ e_8\ n_7\ e_9)\}$ となる．なお，上記手順（４）において，∀-部分記号列 S_i が $[s_{i1}]$, すなわち，$|OB_i| = 1$ となる可能性がある．以下では，このような OB_i もパスブロックと呼ぶことにする．

記述中の出力文 A_{o_v}, $v = 1, 2, \cdots, t$ に対応する記述グラフ中の弧を e_{o_v} とし，弧 e_{o_v} のパスブロックを特に出力パスブロックと呼ぶ．また，弧 e_{o_v} の w 番目の出力パスブロックを $OB_{(v,w)}$ と表す．ここで，t は，記述中の出力文の数である(注6)．本論文では，この $OB_{(v,w)}$ 中の各パスによって定められた，出力文 A_{o_v} の出力動作を，**2.1** で述べた BAS 上の各内部出力線への出力動作に対応させる．また，この $OB_{(v,w)}$ を内部出力線 $o_{(v,w)}$ の出力パスブロックと呼ぶことにする．

2.1 で説明した BAS における内部出力線，依存入力集合，パス・コンディションには，それぞれ，o_i, D_i, PC_i のように，添字に $i = 1, 2, \cdots, m$ を用いてきた．しかし，BAS 中のユニットと出力パスブロックとの対応をとるために，以下，これらの添字として順序対 (v, w) を用いる．ここで，v, w は，前述の v, w である．

以下，記述グラフ上での依存入力集合，パス・コンディションおよびこれらを考慮した機能を定義する．なお，出力パスブロック $OB_{(v,w)}$ 中に，外部入力線 I_j からの入力を行う入力文に対応する異なる弧が $T_{j(v,w)}$ 個存在する場合，**2.1** で述べたように，それらの入力文に現れる入力名標 I_j に対して，$I_j^1, I_j^2, \cdots, I_j^{T_{j(v,w)}}$ のような番号を付し，これらの入力名標を異なるものとして扱う．これらの準備の下で，先の［定義2.4］，［定義2.5］を出力パスブロックに対応させて定義し直すと以下のようになる．

［定義 2.4′］ 出力パスブロック $OB_{(v,w)}$ 中の各弧が表す代入文に現れる入力名標のうち，$o_{(v,w)}$ の値に影響を与える可能性をもつ入力名標すべての集合を内部出力線 $o_{(v,w)}$ の依存入力集合と言い，$D_{(v,w)}$ と表す． □

［定義 2.5′］ 出力パスブロック $OB_{(v,w)}$ 中の各条件ノードが表す条件を $\alpha_1, \alpha_2, \cdots, \alpha_{c_{(v,w)}}$，とする．このとき，これらすべての条件を論理積 (∧) で結合した論理式 $PC_{(v,w)} = \alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_{c_{(v,w)}}$ を内部出力線 $o_{(v,w)}$ のパス・コンディションと言う． □

［定義2.9］ パス・コンディション $PC_{(v,w)}$ が真となるとき，内部出力線 $o_{(v,w)}$ の値は，依存入力集合 $D_{(v,w)}$ の各要素が表す外部入力線の値から演算されるということを "$f_{(v,w)} = (D_{(v,w)} | PC_{(v,w)})$" と表し，これを内部出力線 $o_{(v,w)}$ の機能と言う．また，順序対 (v, w) を機能番号と言う． □

上記の依存入力集合およびパス・コンディションの導出方法については，**3.** で述べる．

以下，仕様の内部出力線 $os_{(v_s,w_s)}$ の機能を $fs_{(v_s,w_s)}$，設計の内部出力線 $od_{(v_d,w_d)}$ の機能を $fd_{(v_d,w_d)}$ と表す．また，仕様および設計記述中の出力文の数を，それぞれ t_s, t_d とする．更に，仕様の記述グラフ中の弧 $e_{os_{v_s}}$ の出力パスブロックの総数を u_{v_s}，設計の記述グラフ中の弧 $e_{od_{v_d}}$ の出力パスブロックの総数を u_{v_d} とする．

［定義2.10］ 仕様記述が表すすべての機能の集合を Fs，設計記述が表すすべての機能の集合を Fd と表し，これらをそれぞれ，仕様機能，設計機能と言う．

$$Fs = \{fs_{(v_s,w_s)} \mid v_s = 1, 2, \cdots, t_s, w_s = 1, 2, \cdots, u_{v_s}\}$$

$$Fd = \{fd_{(v_d,w_d)} \mid v_d = 1, 2, \cdots, t_d, w_d = 1, 2, \cdots, u_{v_d}\}$$

□

［定義2.9］，［定義2.10］から次の定理が成立する．

［定理2.1］ 以下の条件（１），（２）が成立すれば，［要件2.1］の（１），（２）が共に満たされる．この逆も成立する．

（1） $\forall fs_{(v_s,w_s)} \in Fs$ に対して，$\exists fd_{(v_d,w_d)} \in Fd$ such that $[os_{(v_s,w_s)}] = O_k$, $[od_{(v_d,w_d)}] = O_k$，かつ，$Dd_{(v_d,w_d)} \supseteq Ds_{(v_s,w_s)}$

（2） $\forall fs_{(v_s,w_s)} \in Fs$ に対して，論理式

$$PCs_{(v_s,w_s)} \rightarrow \left(\bigvee_{(v_d,w_d) \in \mathcal{D}_{(v_s,w_s)}} \left(PCd_{(v_d,w_d)} \wedge \right.\right.$$

(注6)：記述中の出力文の数 t と BAS (図1) のユニット数 m，外部出力線数 q との関係は，一般に $q \leqq t \leqq m$ となる．

$$\left(\bigwedge_{(v'_d,w'_d)\in\mathcal{D}_{(v_s,w_s)}-\{(v_d,w_d)\}} \neg PCd_{(v'_d,w'_d)}\right)\left.\right)\left.\right)\left.\right)$$

が恒真となる．ここで，$\mathcal{D}_{(v_s,w_s)}$ は，$fs_{(v_s,w_s)}$ に対して上記の条件（1）を満たしている設計の機能番号の集合．

(証明) 上記条件（1）を満たす $Dd_{(v_d,w_d)}$ が存在するとき，[要件2.1]（1）が満たされることは，明らかである．また，逆に，[要件2.1]（1）が満たされる場合，上記条件（1）を満たすような $Dd_{(v_d,w_d)}$ が存在するはずである．よって，上記条件（1）が成立すれば，[要件2.1]（1）が満たされ，この逆も成立する．

また，上記条件（2）の論理式は，ただ一つのパス・コンディション $PCd_{(v_d,w_d)}\in\mathcal{D}_{(v_s,w_s)}$ が真となるときのみ，→ の右辺が真となる．すなわち，上記条件（2）が満たされる場合，[要件2.1]（2）も成り立つことになる．また逆に，[要件2.1]（2）が成り立つ場合，上記条件（2）の論理式は，常に真となる．よって，上記条件（2）が成立すれば，[要件2.1]（2）が満たされ，この逆も成立する．

以上より，上記条件（1），（2）が成立すれば，[要件2.1] の（1），（2）が共に満たされ，この逆も成立する． □

3. 機能の導出

ここでは，**2.3** で定義した機能（依存入力集合およびパス・コンディション）の導出方法について述べる．

3.1 依存入力集合の導出

まず，いくつかの用語を説明する．記述グラフ中の複数の出力弧をもつノードを分岐ノード，複数の入力弧をもつノードを収れんノードと呼ぶことにする．弧 e_{o_v} の w 番目の出力パスブロック $OB_{(v,w)}$ 中の任意の 2 本のパス op_g, op_h が同じ分岐ノード n_f を含み，その接続関係が $Out(n_f, E_f)$ であるとする．このとき，パス op_g 中の弧 $e_{a_g}\in E_f$ とパス op_h 中の弧 $e_{a_h}\in E_f$ とが異なる場合，パス op_g, op_h は，ノード n_f で分岐すると言う．同様に，パス op_g, op_h が同じ収れんノード n_j を含み，その接続関係が $In(n_j, E_j)$ で，パス op_g 中の弧 $e_{b_g}\in E_j$ とパス op_h 中の弧 $e_{b_h}\in E_j$ とが異なる場合，パス op_g, op_h は，ノード n_j で収れんすると言う．

記述グラフでは，以下の補題が成り立つ．

[補題3.1] 複数の要素をもつ出力パスブロック $OB_{(v,w)}$ 中の異なる 2 本のパス op_g, op_h 中には，これら 2 本のパスが分岐するノード n_{f_1} が存在する．また，ノード n_{f_1} で分岐した 2 本のパスが (∀-) 収れんするノード n_{j_1} が存在する．更に，これらのパスが分岐するノードが複数存在する場合，その分岐ノード n_{f_l}, $l=1,2,\cdots,k$ で分岐したパスが再び収れんするノード $n_{j_l}, l=1,2,\cdots,k$ が存在し，これらの分岐ノードおよび収れんノードは，パス op_g, op_h 中で $(n_1,\cdots,n_{f_1},\cdots,n_{j_1},\cdots,n_{f_2},\cdots,n_{j_2},\cdots,n_{f_k},\cdots,n_{j_k},\cdots,e_{o_v})$ のように交互に並んでいる．

(証明) パス op_g, op_h 中に，分岐ノード n_{f_1} および収れんノード n_{j_1} が存在することは，[手順2.2] から明らかである．従って，このような分岐ノードおよび収れんノードが複数存在する場合について上記の補題が成立することを証明すればよい．

2 本のパス op_g, op_h が，ノード n_{f_l} で分岐するためには，n_{f_l} への入力弧は，パス op_g, op_h で同一でなければならない．同様に，パス op_g, op_h が，ノード n_{j_l} で収れんするためには，n_{j_l} へのパス op_g における入力弧とパス op_h における入力弧は異なっていなければならない．このため，分岐ノードおよび収れんノードが複数存在する場合，分岐ノードと収れんノードは交互に並んでいなければならない． □

本論文では，HDLで記述されたシステムの動作に関して次の前提を置く．

[前提3.1]

（1） 同一の記憶要素の内容を書き換える代入文が並列に実行されない．

（2） 同一の記憶要素に対する書き込みを行う代入文と参照をする代入文が並列に実行されない． □

HDL記述中に上記前提で示したような代入文が存在するか否かを，記述グラフ上で検出することは可能であると考えるが，本論文では，以下の議論を簡単にするために [前提3.1] を設けた．この前提に基づき，内部出力線 $o_{(v,w)}$ の依存入力集合 $D_{(v,w)}$ の導出手順を以下に示す．

以下の手順において，出力パスブロック $OB_{(v,w)}$ 中のパスを op_l, $l=1,2,\cdots,r_{(v,w)}$ と表す．また，記述中のある二つの名標 L_s, L_r に対して，"名標 L_s の外部入力線または記憶要素の値は，名標 L_r の外部出力線または記憶要素の値に影響を及ぼす" という関係を $L_s\mathcal{K}L_r$ と表す．但し，一つの名標 L_r について，パス op_l 中の $h,\cdots,i,j,\cdots,k$ 番目 $(h<\cdots<i<j<\cdots<k)$ の弧の表す代入文で，$L_h\mathcal{K}L_r,\cdots,L_i\mathcal{K}L_r,L_j\mathcal{K}L_r,\cdots,L_k\mathcal{K}L_r$ という

表2　述語とその意味
Table 2　Predicates and their meanings.

述　語	意　　味
$EQ(X,Y)$	X is EQual to Y.　$(X=Y)$
$LT(X,Y)$	X is Larger Than Y.　$(X>Y)$

関係がある場合は，$L_h\mathcal{K}L_r^h, \cdots, L_i\mathcal{K}L_r^i$, $L_j\mathcal{K}L_r^j, \cdots$, $L_k\mathcal{K}L_r^k$ と表し，異なる名標として扱う．この表記法に従って，f 番目 $(i<f\leqq j)$ の弧の $L_r\mathcal{K}L_g$ という関係は，$L_r^i\mathcal{K}L_g$ とする．この関係 $\mathcal{K}$ では，明らかに推移律が成り立つ．

［手順3.1］（依存入力集合 $D_{(v,w)}$ の導出手順）

（1）　パス op_l 中のすべての弧の代入文について，その代入文に現れる名標間の関係 $\mathcal{K}$ を求める；

（2）　推移律を用いて $I_g\mathcal{K}O_h(=[o_{(v,w)}])$ となる入力名標 I_g をすべて求める；

（3）　出力パスブロック $OB_{(v,w)}$ 中のすべてのパス op_l, $l=1,2,\cdots,r_{(v,w)}$ に対して，上記（1），（2）を適用する；

（4）　上記（1）～（3）を適用して得られたすべての入力名標の集合を $D_{(v,w)}$ とし，これを内部出力線 $o_{(v,w)}$ の依存入力集合とする；　□

3.2　パス・コンディションの導出

パス・コンディションは，記述グラフ中の条件ノードの条件を論理積 “∧” で結合した論理式であるため，まず記述グラフ中の各条件ノードの条件を入力名標のみを用いて表現する．これは，以下のような手順で行う．まず，記述グラフ中の条件ノードの条件 α の真理値を変化させる可能性のある外部入力線の入力名標すべてを［手順3.1］と同様な方法により求める．次に，求めた入力名標のみを用いて，条件α を表す．その後，条件 α を，表2 の述語および表4 の関数のみを用いた述語論理式に変換する．これらの手順を記述グラフ中のすべての条件ノードに対して行う．これらの各手順は比較的容易と思われるため，ここでは，使用する述語(表2) および関数(表4) についてのみ簡単に述べる．

本論文では，制御文の制御条件は，変数間の大小関係を表しているものが多いと考え，表2 に示す二つの述語のみを用いる．この二つ以外の大小関係は，表3 に示すように，表2 の述語と論理演算子 $(\neg,\wedge,\vee)$ を用いて表現可能である．また，大小関係を表していない制御条件，すなわち，各外部入力線のレベル(High, Low, enable, disable, 1, 0等)や記憶要素の状態等，

表3　表2 の述語によるその他の関係演算子の表現
Table 3　Relational operators represented by predicates EQ and LT.

述 語 論 理 式	意　味
$\neg EQ(X,Y)$	$X\neq Y$
$\neg LT(X,Y)$	$X\leqq Y$
$EQ(X,Y)\vee LT(X,Y)$	$X\geqq Y$
$\neg EQ(X,Y)\wedge\neg LT(X,Y)$	$X<Y$

表4　定義域 B 上の関数
Table 4　Functions on domain B.

関　　数		
$Sum(x,y)$	$=$	$x+y$
$Dif(x,y)$	$=$	$x-y$
$Not(x)$	$=$	$\overline{x}$
$And(x,y)$	$=$	$x\wedge y$
$Or(x,y)$	$=$	$x\vee y$
$I_i^n(x_1,x_2,\cdots,x_n)$	$=$	x_i

意味として，変数間の大小関係を表していないものも，述語 $EQ(X,Y)$ を用いて表現可能である．ここで，これらの述語に対する引数の定義域は，システムに印加可能な 2 値ベクトルすべての集合とし，これを B と表す．また，定義域 B 上の関数として，表4 に示す六つの関数を用いる．前述のように本論文では，記述が表している関数は，原始帰納的関数に限定している．このため，記述中のすべての関数は，和関数 Sum および射影関数 I_i^n をそれぞれ有限個用いて表現できるが，制御条件を簡潔に表すために他の四つの関数(差関数 Dif, ビット論理否定関数 Not, ビット論理積関数 And, ビット論理和関数 Or) も用いることにする．

4.　例　　題

ここでは，本論文で定義した機能を用いた簡単な検証例を示す．

仕様記述例にはVHDL [2], [3](図4)を使用する．VHDLは，システムレベルからゲートレベルまで一貫して設計を行うことが可能であるが，本方法の一般性を示すために，設計記述にはAHPL [6](図5)を例として用いる．また便宜上，VHDL記述に行番号を付した．また，仕様の入力名標 a, b, c, d, e, f に対して，設計の入力名標 A, B, C, D, E, F がそれぞれ対応する．同様に，仕様の出力名標 g, h に対して，設計の出力名標 G, H がそれぞれ対応する．図4 および図5 の記述に対し［手順2.1］を適用して得られる記述グラフをそれぞれ図6, 図7 に示す．これらの記述グラフ中

```
--Example System (ES)
entity ES is
  port (a, b, c, d, e, f: in integer;
        g, h: out integer;
        i, j: buffer integer);
end ES;

 1  architecture specification of ES is
 2  begin
 3    process
 4    begin
 5      i <= a * b;
 6      if a >= b then
 7        g <= c + i;
 8      else
 9        g <= d - i;
11      endif;
12      j <= e + f
13      if c > d then
14        h <= a * j;
15      else
16        h <= b * j;
17      endif;
18    end process;
19  end specification;
```

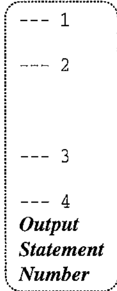


図4 VHDLによる仕様記述例
Fig. 4 Example of specification in VHDL.

```
AHPLMODULE DESIGN_OF_ES
    EXINPUTS: A[4];B[4];C[4];D[4];E[4];F[4].
     OUTPUTS: G[4];H[4].
      MEMORY: MA[4];MB[4];MC[4];MD[4];
              ME[4];MF[4];MG[5];MH[5].

 1 MA <= A; MB <= B; MC <= C; MD <= D;
   ME <= E; MF <= F.
 2 =>(LARGER(MA;MB))/(4).
 3 =>(^EQUAL(MA;MB))/(6).
 4 MG <= ADD(MC;MULT(MA;MB)).
 5 =>(7).
 6 MG <= SUB(MD;MULT(MA;ME)).
 7 =>(^EQUAL(MC;MD))/(10).
 8 MH <= MULT(MA;ADD(ME;MF)).
 9 DEADEND.
10 MH <= MULT(MB;ADD(ME;MF)).
11 DEADEND.

ENDSEQUENCE
  CONTROLRESET(1);
  G = MG[1:4]; H = MH[1:4].
END.
```

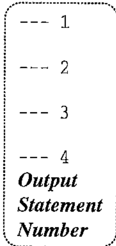


図5 仕様記述(図4)に対する設計記述例(AHPL)
Fig. 5 Example of design in AHPL for specification shown in Fig.4.

の弧に付されている番号は，仕様および設計記述に付されている行番号と対応する．また，この図では，簡単のために表2の述語および表4の関数を使用していない．

図6から仕様記述は6個，図7から設計記述は9個の機能をもっていることがわかる．これらを表5にまとめる．表5中の番号(機能番号)は，出力文の番号 v とその出力文を表す弧の出力パスブロックの番号 w，を用いて表されている．

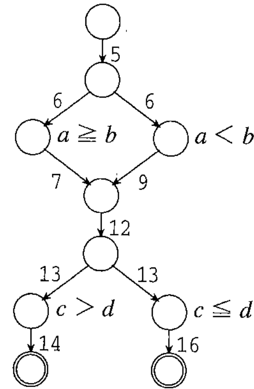


図6 仕様記述(図4)から得られる記述グラフ
Fig. 6 Description graph for specification shown in Fig.4.

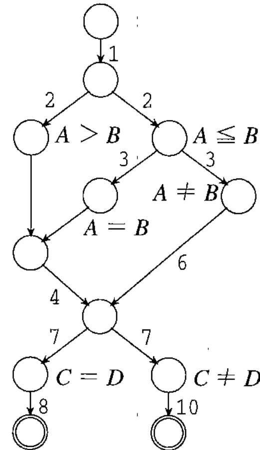


図7 設計記述(図5)から得られる記述グラフ
Fig. 7 Description graph for design shown in Fig.5.

仕様の機能 $fs_{(1,1)}$ について検証を行う．$fs_{(1,1)}$ について，[定理2.1]（1）を満たす設計の機能は，$fd_{(1,1)}$ および $fd_{(1,2)}$ である．これらの機能のパス・コンディションを[定理2.1]（2）の式に適用すると以下のようになる．

$$(EQ(a,b) \vee LT(a,b)) \rightarrow$$
$$(LT(A,B) \wedge \neg(\neg LT(A,B) \wedge EQ(A,B)))$$
$$\vee(\neg LT(A,B) \wedge EQ(A,B) \wedge \neg LT(A,B))$$
$$= (EQ(a,b) \vee LT(a,b)) \rightarrow$$
$$(LT(A,B) \vee EQ(A,B))$$

表5 記述グラフから得られる機能
Table 5 Functions obtained from description graphs.

(a) 仕様(図4)の機能(図6より)

番号	出力：機能 "$D\|PC$"
(1,1)	g ： $\{a,b,c\}$ | $EQ(a,b) \vee LT(a,b)$
(2,1)	g ： $\{a,b,d\}$ | $\neg EQ(a,b) \wedge \neg LT(a,b)$
(3,1)	h ： $\{a,e,f\}$ | $(EQ(a,b) \vee LT(a,b)) \wedge LT(c,d)$
(3,2)	h ： $\{a,e,f\}$ | $\neg EQ(a,b) \wedge \neg LT(a,b) \wedge LT(c,d)$
(4,1)	h ： $\{b,e,f\}$ | $(EQ(a,b) \vee LT(a,b)) \wedge \neg LT(c,d)$
(4,2)	h ： $\{b,e,f\}$ | $\neg EQ(a,b) \wedge \neg LT(a,b) \wedge \neg LT(c,d)$

(b) 設計(図5)の機能(図7より)

番号	出力：機能 "$D\|PC$"
(1,1)	G ： $\{A,B,C\}$ | $LT(A,B)$
(1,2)	G ： $\{A,B,C\}$ | $\neg LT(A,B) \wedge EQ(A,B)$
(2,1)	G ： $\{A,D,E\}$ | $\neg LT(A,B) \wedge \neg EQ(A,B)$
(3,1)	H ： $\{A,E,F\}$ | $LT(A,B) \wedge EQ(C,D)$
(3,2)	H ： $\{A,E,F\}$ | $\neg LT(A,B) \wedge EQ(A,B) \wedge EQ(C,D)$
(3,3)	H ： $\{A,E,F\}$ | $\neg LT(A,B) \wedge \neg EQ(A,B) \wedge EQ(C,D)$
(4,1)	H ： $\{B,E,F\}$ | $LT(A,B) \wedge \neg EQ(C,D)$
(4,2)	H ： $\{B,E,F\}$ | $\neg LT(A,B) \wedge EQ(A,B) \wedge \neg EQ(C,D)$
(4,3)	H ： $\{B,E,F\}$ | $\neg LT(A,B) \wedge \neg EQ(A,B) \wedge \neg EQ(C,D)$

「番号」は，機能番号を表す．

上式は真となるため，$fs_{(1,1)}$ は，設計記述で実現されていることがわかる．この仕様の機能 $fs_{(1,1)}$ は，外部入力線 a,b の値に $a \geqq b$ という関係があるとき，外部出力線 g から $c+a*b$ の値を出力する動作に対応している．一方，設計では，$A>B$ の場合に $C+A*B$ の値が外部出力線 G に送られ，$A=B$ の場合に $C+A*B$ の値が外部出力線 G に送られている．これらの動作に対応する機能が，それぞれ，機能 $fd_{(1,1)}$, $fd_{(1,2)}$ である．このように，設計において，仕様の機能が分けて実現されている場合でも本検証法は有効である．同様に $fs_{(2,1)}$ について検証を行う．$fs_{(2,1)}$ について，[定理2.1]（1）を満たす設計の機能は存在しない．仕様の機能 $fs_{(2,1)}$ は，外部入力線 a,b の値に $a<b$ という関係があるとき，外部出力線 g から $d-a*b$ の値を出力する動作に対応している．一方，設計では，$A<B$ の場合に，$D-A*E$ の値が外部出力線 G に送られており(対応する機能は $fd_{(2,1)}$)，仕様の機能 $fs_{(2,1)}$ の入出力依存関係と異なる．よって，$fs_{(2,1)}$ は設計で実現されていないことがわかる．この時点で，設計に誤りがあると検証されるが，同様にして検証を続けた場合，$fs_{(4,1)}$, $fs_{(4,2)}$ は，[定理2.1]（2）を満たさないため，設計で実現されていないことが確認できる．

5. むすび

本論文ではまず，検証しようとするハードウェアの動作を入出力の依存関係に着目して抽象化したシステム BAS を示し，この BAS 上で，本論文における検証問題を明確化した．次に，HDL記述の検証問題をこの明確化された検証問題に帰着させるために，HDL記述を記述グラフと呼ばれる有向グラフによりモデル化し，この記述グラフと BAS との対応について説明した．また，この対応に従って，記述グラフ上で，検証対象となる機能を定義し，演算の誤りの一部と考える入出力の依存関係の誤りを検出可能な検証法を提案した．

本検証法によって入出力依存関係の誤りを検出することにより，演算の誤りの検出作業を軽減できると考える．今回は，HDL 記述に対して，while 文を使用していないことを前提とした．しかし，HDL 記述中の while 文によって繰り返される文を一つの代入文とみなして記述グラフを得ることにより，帰納的関数を実現しているシステムのHDL記述の検証も可能となる．本検証法は，このように拡張性があり，かつ，論理合成時代に望まれる高レベルの検証法である．

実用的な検証法としては，検出された誤りの位置を特定できることが望まれる．この問題は今後の課題である．更に，今回提案した検証法の，より広い意味での演算の誤りを検出可能な検証法への拡張は，現在検討中である．

文　献

[1] 星野民夫, 唐津　修, “UDL/I,” 情報処理, vol.33, no.11, pp. 1244–1249, Nov. 1992.

[2] R. Lipsett, E. Marschner, and M. Shahdad, “VHDL - The Language,” IEEE Design & Test, pp. 28–42, April 1986.

[3] Z. Navabi, “VHDL : Analysis and Modeling of Digital Systems,” McGraw-Hill, 1993.

[4] 小栗　清, 中村行宏, 野村　亮, 名古屋彰, “SFL,” 情報処理, vol.33, no.11, pp. 1256–1262, Nov. 1992.

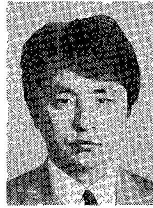
[5] 野地　保, “Verilog HDL,” 情報処理, vol.33, no.11, pp. 1263–1268, Nov. 1992.

[6] F.J. Hill and G.R. Peterson, "DIGITAL SYSTEMS : Hardware Organization and Design," 2nd ed., Wiley, 1978.
[7] 村上道郎, 平川和之, "方式・機能・論理シミュレーション," 情報処理, vol.25, no.10, pp. 1048–1055, Oct. 1984.
[8] 安浦寛人, "論理合成時代のハードウェア記述言語," 情報処理, vol.33, no.11, pp. 1236–1243, Nov. 1992.
[9] A.S. Wojcik, "Formal Design Verification of Digital Systems," Proc. 20th Design Automation Conf., pp. 228–234, June 1983.
[10] V. Pitchumani and E.H. Stabler, "A Formal Method for Computer Design Verification," Proc. 19th Design Automation Conf., pp. 809–814, June 1982.
[11] G.J. Milne, "A Model for Hardware Description and Verification," Proc. 21st Design Automation Conf., pp. 251–257, June 1984.
[12] 高原　厚, 南谷　崇, "高水準設計検証の一方式", 情処学論, vol.27, no.8, pp. 783–792, Aug. 1986.
[13] A.P. Pawlovsky and S. Naito, "Verification of Register Transfer Level(RTL) Designs," IEICE Trans. Inf. & Syst., vol.E75-D, no.6, pp. 785–791, Nov. 1992.
[14] A. Palacios, S. Naito, and M. Tsunoyama, "Buffer a Novel Specification Language for Digital Systems", Trans. IEICE, vol.E 70, no.10, pp. 902–905, Oct. 1987.
[15] アルベルト パラシオス P., 内藤祥雄, "仕様記述言語 Buffer とシステムの一設計検証法," 信学論(D-I), vol.J73-D-I, no.2, pp. 154–160, Feb. 1990.
[16] 吉田たけお, 貴家仁志, 内藤祥雄, "データフローに基づくハードウェア記述言語の検証法," 信学技報, FTS92-50, Feb. 1993.
[17] T. Yoshida, H. Kiya, and S. Naito, "Function Based on Data Flow for Hardware Description Languages and its Functional Verification," Proc. of Pacific Rim International Symposium on Fault Tolerant Computing, pp. 118–122, Dec. 1993.
[18] M.J.C. Gordon and J. Herbert, "Formal hardware verification methodology and its application to a network interface chip," IEE Proc., vol.133, Pt. E, 5, pp. 255–270, Sept. 1986.
[19] 平石裕実, 浜口清治, "論理関数処理に基づく形式的検証手法," 情報処理, vol.35, no.8, pp. 710–718, Aug. 1994.
[20] 福村晃夫, 稲垣康善, "オートマトン・形式言語理論と計算論," 岩波書店, 情報科学-12, May 1982.
[21] 土岐秀雄, "情報処理用語辞典," 日刊理工出版会, 第2版, June 1983.

(平成6年10月28日受付, 7年5月25日再受付)

吉田たけお　(学生員)

平3長岡技科大・電気電子システム工学課程卒. 平5同大学大学院修士課程了. 現在, 都立大大学院博士課程在学中. ハードウェアの設計検証に関する研究に従事. 情報処理学会, 日本ソフトウェア科学会, IEEE各会員.



貴家　仁志　(正員)

昭55長岡技科大・電気電子システム工学課程卒. 昭57同大学大学院修士課程了. 同年都立大・工・電気工学科助手. 現在, 同大学電子・情報工学科助教授. 平7年10月よりオーストラリア, シドニー大学客員研究員. ディジタル信号処理, 画像処理および信号処理アルゴリズムのVLSI実現に関する研究に従事. 著書「高速フーリエ変換とその応用」(共著),「ディジタル信号処理技術入門」,「マルチレート信号処理」等. 画像電子学会, テレビジョン学会, IEEE各会員. 工博.

故 内藤　祥雄　(正員)

昭43東工大大学院修士課程(電気工学専攻)了. 昭51工博. 昭和43日本電気(株)中央研究所, 昭53長岡技科大助教授, 平3同教授, 平4都立大・工・電子・情報工学科教授. ディジタル回路, システムの故障検出, 超高信頼化設計技術に興味をもつ. 著書「順序機械」等. 情報処理学会, IEEE各会員. 平成7年1月3日逝去.