データパスの検証・補正へのアプローチ†

高木 茂††

ディジタルシステムの大規模化，複雑化により，ハードウェア設計の早期の段階から適用できる検証技術が重要になってきている．本論文は，論理仕様記述言語 DDL で記述された論理仕様と，設計された（あるいは設計途中の）データパス系機能ブロック図との間の整合性の検証，整合しない場合の原因の探索および対応策生成，の一手法を提案する．本手法の特徴は，（1）仕様記述をトランスレートし，単純な演算操作に分解するとともに，演算操作相互間の並列動作性を解析・判定する．（2）演算操作とデータパスの整合性検証は解析的に行う．すなわち，1演算操作を，㋑演算操作に現れる演算子を実行しうるサブデータパスの検出，㋺見つかったサブデータパスと，演算操作で参照・代入される資源との間のデータ転送路の探索，により検証する．（3）並列に動作しうる演算操作相互間で資源競合を起こさないことを確認する．（4）検証成功時には，各演算操作がデータパス上でどのように実行されるかを報告する．（5）検証不成功時には，不成功の原因（転送路の欠如，機能の不足，競合等）を解析し，原因ごとに対応策を立てる．（6）データパスの解析，原因解析，対策立案は，設計専門家の知識をルールの形式で定式化，記述，利用することにより実現している．実験システムにより有効性を確認した．本手法は，論理仕様変更時のデータパス系の変更量の見積り，マイクロプログラム，あるいは，制御回路生成等にも応用できる．

1. まえがき

ディジタルシステムの大規模化，複雑化に伴い，設計の早期の段階から適用できる検証技術が重要となってきている．

大規模システム開発では通常次の設計手順をとることが多い．

（1） 論理仕様をレジスタ・トランスファ・レベルの仕様記述言語で記述し，シミュレーションを行って仕様を確認する．

（2） 概略設計を行い，機能ブロック図を作成する．

（3） 各機能ブロックを詳細なゲートに展開する．

機能ブロックレベルでの設計バグは，修正の波及範囲も広くなるため，十分な検証が必要である．

本論文は，仕様記述言語 DDL[1] で記述された仕様とデータパス系機能ブロックとの間の整合性の検証，不整合時の原因の解析・対策立案の一手法を提案するものである．

従来，DDL 仕様記述と，ハードウェアとの整合性を，両レベルでのシミュレーション結果の照合により検証する方法が提案されていた[3]．

シミュレーション主体の検証法では，①テストデータを生成する必要がある，②計算時間がかかる，③設計が完了しないと検証できない，④不整合時の原因解析，対策立案は人間に頼らざるをえない，という問題がある．設計専門家は必要に応じて，データパスの必要箇所を解析することにより，これら問題に対処していると考えられる．本論文は，専門家の解析的手法を次に示すようにモデル化し，検証知識をルールとして定式化，活用する手法を提案する．

（1） DDL 記述をトランスレートし，単純な演算操作に分解するとともに，演算操作相互間の並列動作/非並列動作を解析する．

（2） 1演算操作とデータパスの整合性を，㋑必要とされる演算機能を実行しうるサブデータパスの検出，㋺見つかったサブデータパスと演算操作で参照・代入される資源との間のデータ転送路の探索，により検証する．

（3） 並列に動作しうる複数の演算操作が同じ資源を使用しないことを確認する．

（4） 検証不成功時には，その不成功の原因（転送路の欠落，機能の不足，競合）を解析し，各原因ごとにさらに対策立案のための詳細な解析を実行する．

以下，第2章では，仕様およびデータパスの記述法を，第3章では整合性検証手法を，第4章では，不整合時の原因解析・対策立案手法を，第5章では実験例を示す．

2. 論理仕様およびデータパスの記述

2.1 論理仕様記述

システムの論理仕様は DDL 言語で記述する．DDL は D. Dietmeyer らによって定義されたレジスタ・ト

† The Verification and Compensation of Data Paths by SHIGERU TAKAGI (Musashino Electrical Communication Laboratory, N. T. T.).

†† 日本電信電話公社武蔵野電気通信研究所

```
<SYSTEM> SAMPLE;
  <REGISTER> GR0 GR1 SC BR IR;
  <INPUT> MEMORY-READ-DATA;
  <OUTPUT> MEMORY-ADDRESS;
  <AUTOMATON> CONTROL;
    <STATE>
    INSTRUCTION-FETCH:  CO-BIGIN
                            MEMORY-ADDRESS <- SC.
                            GOTO MEMORY-READ
                        END;
    MEMORY-READ:        CO-BIGIN
                            IR <- MEMORY-READ-DATA.
                            SC <- 1+ SC.
                            GOTO DECODE-EXECUTE
                        END;
    DECODE-EXECUTE: CASE IR  ADD
                                CO-BIGIN
                                    GR0 <- GR0 + GR1.
                                    GOTO INSTRUCTION-FETCH
                                END;
                            SUBTRACT
                                CO-BIGIN
                                    GR0 <- GR0 - GR1.
                                    GOTO INSTRUCTION-FETCH
                                END;
                            EXCLUSIVE-OR
                                CO-BIGIN
                                    GR0 <- GR0 @ GR1.
                                    GOTO INSTRUCTION-FETCH
                                END;
                            LOAD-BR
                                CO-BIGIN
                                    BR <- GR0.
                                    GOTO INSTRUCTION-FETCH
                                END;
                            JUMP
                                CO-BIGIN
                                    SC <- BR.
                                    GOTO INSTRUCTION-FETCH
                                END;
                            JUMP-AND-LINK
                                CO-BIGIN
                                    SC <- GR0 + BR.
                                    GR1 <- SC.
                                    GOTO INSTRUCTION-FETCH
                                END;
                        END CASE;
    END <STATE>;
  END CONTROL;
END SAMPLE;
```

図 1 DDL による論理仕様の定義例 (SAMPLE)
Fig. 1 DDL description example (SAMPLE).

ランスファ・レベルの論理仕様記述言語であり，状態遷移表現に基礎を置いている．DDL の文法の詳細は原論文を参照されたい[1]．

DDL の記述例を**図1**に示す（ただし，仕様を理解しやすくするため，構文は変更してある）．図1は，本論文で検証・補正の例題として使用する簡単な計算機 (SAMPLE) の論理仕様である．5個のレジスタ (GR 0, GR 1, SC, BR, IR) と2個の入出力端子 (MEMORY-ADDRESS, MEMORY-READ-DATA) が基本的資源として宣言されている．三つの状態 (INSTRUCTION-FETCH, MEMORY-READ, DECODE-EXECUTE) を有し，各状態では，指定された条件が成立すれば，演算操作が実行され，次状態へと遷移していく．

演算操作は，レジスタ等の資源を参照し，演算を行い，結果を資源に格納する．たとえば，

GR 0<−GR 0+GR 1

は，GR 0 と GR 1 の内容を加えて GR 0 に格納することを意味する．演算操作は，レジスタ等資源間の論理的演算・代入関係を示すものであり，実際にこの演算がどの転送路，演算器を使ってどのように実行されるかという具体的なことは規定していない．

2.2 データパスの記述

データパスは，いろいろな機能をもつ構成要素が複数結合されてできている．これらデータパスの構造，機能を述語論理の形式で記述する．本論文では，次の構文規則を使用する．

（1） 節をリスト形式で記述する．リストの第1要素は述語であり，残りの要素はその述語に対する引数である．

（2） * で始まる記号は変数である．

（3） * で始まらない記号は，定数，あるいは述語名である．

以下データパスの構造と機能の記述法を示す．

《構造記述》

データパスの構造は，構成要素とその間の結線で規定される．

各構成要素の名前と種類を TYPE 述語で宣言する．第1引数に種類を，第2引数に名前を記述する．たとえば，レジスタ GR 0 は次のように記述する．

(TYPE REGISTER GR 0)

各構成要素は，データ入力端子，データ出力端子，制御入力端子をもつ．これら端子間の結線は PATH 述語で宣言する．たとえば，構成要素 A の OUT 端子と，構成要素 B の IN 端子を信号線 L 1 で結線することは次のように記述する．

(PATH (A OUT) (B IN) L 1)

PATH 述語の第1引数はソース端子であり，第2引数はシンク端子であり，第3引数は信号線名である．

《機能記述》

各構成要素の機能はその入出力端子間の関数関係を

用いて定義できる．一般に，データパス系の構成要素は，制御系からの制御信号を受けると，入力データに演算を施し，これを出力端子に出力する．たとえば，加算および減算機能をもつ ALU 1 は次のように記述する．

```
(FUNCTION(+(ALU 1  OUT)(ALU 1  IN 1)
  (ALU 1  IN 2))(ALU 1  ADD))
(FUNCTION(−(ALU 1  OUT)(ALU 1  IN 1)
  (ALU 1  IN 2))(ALU 1  SUB))
```

FUNCTION 述語の第1引数は機能を表し，第2引数はその機能を選択するための制御入力である．最初の節は，もし ADD 制御信号が印加されれば，ALU 1 は次の演算を実行することを意味する．

```
(ALU 1  OUT)=(ALU 1  IN 1)
            +(ALU 1  IN 2)
```

構成要素が複数の機能をもつならば，各機能は上記例のように別々の節で宣言する．

図2に，構成要素の種類，図形記法，機能記述例を示す．

図3にデータパスの例 (SAMPLE) を，**図4**にその記述を示す．図4において，機能各称 CONN は，入力データを単純に出力端子側に転送する機能（スルー機能と呼ぶ）である．

3. データパスの検証

データパスのバグは2種類に分類できる．

（1） ハードウェア制約条件の違反

（2） DDL 論理仕様との不一致

ハードウェア制約条件の違反とは，たとえば，出力端子同士の結線，ファンアウト負荷制限オーバ，等の論理仕様とは無関係に検出できるバグであり，各制約条件ごとにチェックルールを作ることにより対処できる．

本論文では，後者に焦点をしぼって検証法を示す．

3.1 検証手法の概要

まずシステムの DDL 論理仕様をトランスレータに

構成要素	機能記述（例）
レジスタ	A (FUNCTION (SET A (A IN)) (A SET)) (FUNCTION (CONN (A OUT) A) (A T))
マルチプレクサ	1 0 B (FUNCTION (CONN (B OUT) (B 0)) (B SEL 0)) (FUNCTION (CONN (B OUT) (B 1)) (B SEL 1))
バス	C (FUNCTION (CONN (C OUT) (C IN)) (C T))
演算器	A B ALU 1 + − C (FUNCTION (+ (ALU 1 C) (ALU 1 A) (ALU 1 B)) (ALU 1 ADD)) (FUNCTION (− (ALU 1 C) (ALU 1 A) (ALU 1 B)) (ALU 1 SUB))
トライステートゲート	G (FUNCTION (CONN (G OUT) (G IN)) (G ON))

図 2 データパスの構成要素と機能
Fig. 2 Data paths components.

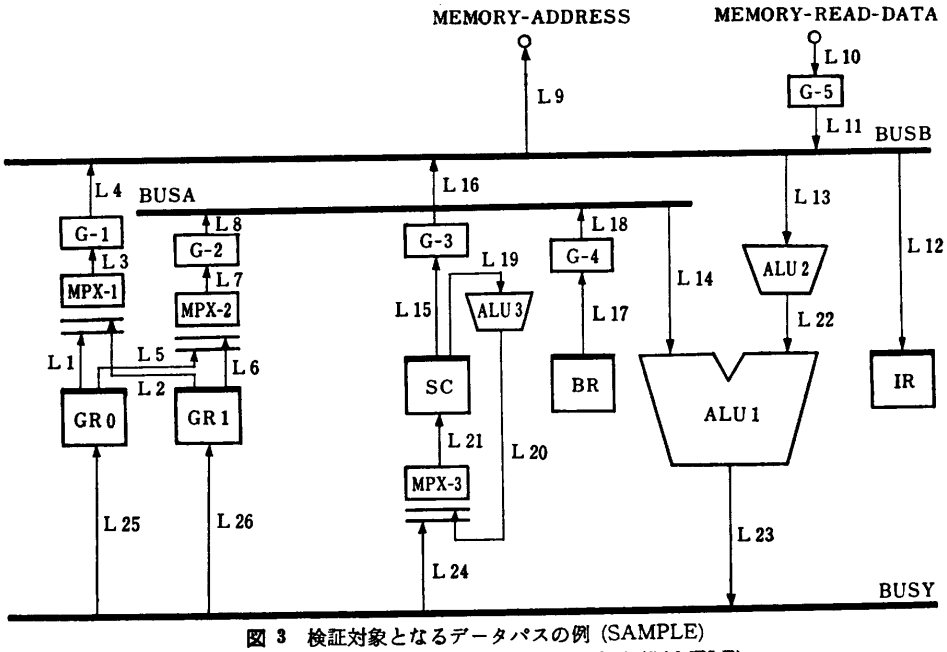


図 3 検証対象となるデータパスの例 (SAMPLE)
Fig. 3 Data paths example to be verified (SAMPLE).

```
(TYPE REGISTER GR0) (TYPE REGISTER GR1) (TYPE REGISTER SC)
(TYPE REGISTER BR)  (TYPE REGISTER IR)
(TYPE INPUT MEMORY-READ-DATA) (TYPE OUTPUT MEMORY-ADDRESS)
(TYPE BUS BUSA) (TYPE BUS BUSB) (TYPE BUS BUSY)
(TYPE MULTIPLEXOR MPX-1) (TYPE MULTIPLEXOR MPX-2) (TYPE MULTIPLEXOR MPX-3)
(TYPE GATE G-1) (TYPE GATE G-2) (TYPE GATE G-3) (TYPE GATE G-4)
(TYPE GATE G-5)
(TYPE ALU ALU1) (TYPE ALU ALU2) (TYPE ALU ALU3)

(PATH (GR0 OUT) (MPX-1 1) L1) (PATH (GR1 OUT) (MPX-1 2) L2)
(PATH (MPX-1 OUT) (G-1 IN) L3) (PATH (G-1 OUT) (BUSB IN) L4)
(PATH (GR0 OUT) (MPX-2 1) L5) (PATH (GR1 OUT) (MPX-2 2) L6)
(PATH (MPX-2 OUT) (G-2 IN) L7) (PATH (G-2 OUT) (BUSA IN) L8)
(PATH (BUSB OUT) MEMORY-ADDRESS L9) (PATH MEMORY-READ-DATA (G-5 IN) L10)
(PATH (G-5 OUT) (BUSB IN) L11)
(PATH (BUSB OUT) (IR IN) L12) (PATH (BUSB OUT) (ALU2 IN) L13)
(PATH (BUSA OUT) (ALU1 A) L14)
(PATH (SC OUT) (G-3 IN) L15) (PATH (G-3 OUT) (BUSB IN) L16)
(PATH (BR OUT) (G-4 IN) L17) (PATH (G-4 OUT) (BUSA IN) L18)
(PATH (SC OUT) (ALU3 IN) L19) (PATH (ALU3 OUT) (MPX-3 2) L20)
(PATH (MPX-3 OUT) (SC OUT) L21)
(PATH (ALU2 OUT) (ALU1 B) L22) (PATH (ALU1 C) (BUSY IN) L23)
(PATH (BUSY OUT) (MPX-3 1) L24)
(PATH (BUSY OUT) (GR0 IN) L25) (PATH (BUSY OUT) (GR1 IN) 26)

(FUNCTION (CONN (BUSA OUT) (BUSA IN)) (BUSA T))
(FUNCTION (CONN (BUSB OUT) (BUSB IN)) (BUSB T))
(FUNCTION (CONN (BUSY OUT) (BUSY IN)) (BUSY T))
(FUNCTION (CONN (MPX-1 OUT) (MPX-1 1)) (MPX-1 SEL1))
(FUNCTION (CONN (MPX-1 OUT) (MPX-1 2)) (MPX-1 SEL2))
(FUNCTION (CONN (MPX-2 OUT) (MPX-2 1)) (MPX-2 SEL1))
(FUNCTION (CONN (MPX-2 OUT) (MPX-2 2)) (MPX-2 SEL2))
(FUNCTION (CONN (MPX-3 OUT) (MPX-3 1)) (MPX-3 SEL1))
(FUNCTION (CONN (MPX-3 OUT) (MPX-3 2)) (MPX-3 SEL2))
(FUNCTION (CONN (G-1 OUT) (G-1 IN)) (G-1 ON))
(FUNCTION (CONN (G-2 OUT) (G-2 IN)) (G-2 ON))
(FUNCTION (CONN (G-3 OUT) (G-3 IN)) (G-3 ON))
(FUNCTION (CONN (G-4 OUT) (G-4 IN)) (G-4 ON))
(FUNCTION (CONN (G-5 OUT) (G-5 IN)) (G-5 ON))
(FUNCTION (1+ (ALU3 OUT) (ALU3 IN)) (ALU3 INCREMENT))
(FUNCTION (^ (ALU2 OUT) (ALU2 IN)) (ALU2 INVERT))
(FUNCTION (CONN (ALU2 OUT) (ALU2 IN)) (ALU2 THROUGH))
(FUNCTION (& (ALU1 C) (ALU1 A) (ALU1 B)) (ALU1 AND))
(FUNCTION (OR (ALU1 C) (ALU1 A) (ALU1 B)) (ALU1 OR))
(FUNCTION (+ (ALU1 C) (ALU1 A) (ALU1 B)) (ALU1 ADD))
(FUNCTION (+ (ALU1 C) (ALU1 A) (ALU1 B) 1) (ALU1 ADD-1))
(FUNCTION (CONN (ALU1 C) (ALU1 A)) (ALU1 SELA))
(FUNCTION (CONN (ALU1 C) (ALU1 B)) (ALU1 SELB))
(FUNCTION (SET GR0 (GR0 IN)) (GR0 SET))
(FUNCTION (SET GR1 (GR1 IN)) (GR1 SET))
(FUNCTION (CONN (GR0 OUT) GR0) (GR0 T))
(FUNCTION (CONN (GR1 OUT) GR1) (GR1 T))
```

図 4 データパスの記述例 (SAMPLE)
Fig. 4 Data paths description example (SAMPLE).

より，次に示すような単純な演算操作と，演算操作ごとの実行条件に変換する．

実行条件　シンク←ソース 1 ⓕ ソース 2
実行条件　シンク←ⓕ ソース
実行条件　シンク←ソース

ここでⓕは演算子である．

たとえば，図1に示した仕様は**図5**のように変換される．変換手法は文献2)，3)，5) を参照されたい．図5において，演算操作の2番目 (OP_2) と3番目 (OP_3) は並列に実行される．また9番目 (OP_9) と10番目 (OP_{10}) も並列に実行される．その他の演算操作の組合せは並列に実行されない．演算操作相互間の並列動作性判定手法は文献5) を参照されたい．

論理仕様とデータパスの整合性検証問題は次の3条件が成立することを確認する問題ととらえることができる．

(1) 各演算操作がデータパス上で実行できる．

(2) 並列に動作しうる複数の演算操作が同じ資源を使用しない．

(3) データパス系資源の並列動作性に関する制約条件に違反しない．

データパス系資源の並列動作性に関する制約条件とは，たとえば，制御系がμプログラム方式で作成されており，一つのμ命令フィールドが複数のデータパス

OPid	OPERATION	CONDITION
OP1	MEMORY-ADDRESS <- SC	INSTRUCTION-FETCH
OP2	IR <- MEMORY-READ-DATA	MEMORY-READ
OP3	SC <- 1+ SC	MEMORY-READ
OP4	GR0 <- GR0 + GR1	DECODE-EXECUTE & (IR = ADD)
OP5	GR0 <- GR0 - GR1	DECODE-EXECUTE & (IR = SUBTRACT)
OP6	GR0 <- GR0 @ GR1	DECODE-EXECUTE & (IR = EXCLUSIVE-OR)
OP7	BR <- GR0	DECODE-EXECUTE & (IR = LOAD-BR)
OP8	SC <- BR	DECODE-EXECUTE & (IR = JUMP)
OP9	SC <- GR0 + BR	DECODE-EXECUTE & (IR = JUMP-AND-LINK)
OP10	GR1 <- SC	DECODE-EXECUTE & (IR = JUMP-AND-LINK)

図 5 DDL のトランスレートの例 (SAMPLE)
Fig. 5 DDL translation result (SAMPLE).

系資源を制御するようにコード化されているため，これら資源が並列に動作できないというような，条件である．以下，具体的判定手法を示す．

3.2 検証手法

《演算操作実行可能性の検証》

データパスが次の条件を満たすならば，演算操作は，そのデータパス上で実行可能である（**図6**）．

（1） 演算操作に必要な機能（演算子）を実行可能なサブデータパスが存在する．

（2） サブデータパスと，演算操作で参照あるいは代入される資源との間にデータ転送路が存在する．

したがって1演算操作の検証は，述語論理を使って，次のように定義できる（2項演算の場合）．

```
(VERIFY (*D<−*S1 *F *S2) (*PATH 0
  (*Z *PATH 3) (*X *PATH 1)
  (*Y *PATH 2)))
    −>(FIND−FUNCTION
        (*Z<−*X *F *Y) *PATH 0)
      (FIND−PATH *S1 *X *PATH 1)
      (FIND−PATH *S2 *Y *PATH 2)
      (FIND−PATH *Z *D *PATH 3)
```

記号 −> の左辺が定義の頭部であり，右辺が本体である．述語 VERIFY の第1引数は演算操作であり，第2引数は検証結果である．すなわち，検証成功時には，演算操作がデータパス上でどのように実行されるかについての情報がセットされる．FIND−FUNCTION は，サブデータパスを見つける述語であり，FIND−PATH はデータ転送路を見つける述語である．変数 *PATH$_i$ は見つけたパスである．まず必要とされる機能 *F を満たすサブデータパスを見つけた (FIND−FUNCTION) 後，このデータパスの入出力端子 (*Z, *X, *Y) と演算操作で参照・代入する資源 (*D, *S1, *S2) との間のデータ転送路を見つける (FIND−PATH).

〈サブデータパスの探索〉

もし，*F の機能を有する ALU がデータパス中に存在するならば，探していたサブデータパスは，この ALU に一致する．すなわち，

```
(FIND−FUNCTION (*Z<−*X *F *Y) *CTL)
    −>(FUNCTION (*F *Z *X *Y) *CTL)
```

もし，*F の機能を有する ALU が存在しないならば，*F を，複数のより基本的な機能の合成として実現する可能性を調べる必要がある（**図7**）．これは，関数合成に関する知識を述語 FIND−FUNCTION に集約することにより実現できる．たとえば，減算は，否定と加算を利用して実現できるという関数合成の知識は次のように記述できる．

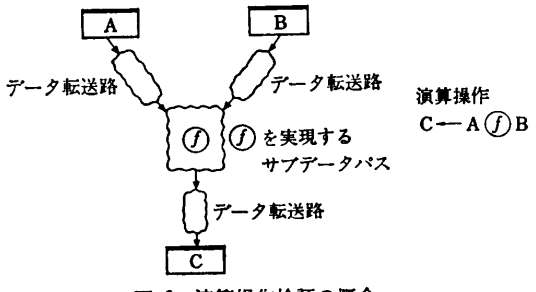


図 6 演算操作検証の概念
Fig. 6 Operation verification concept.

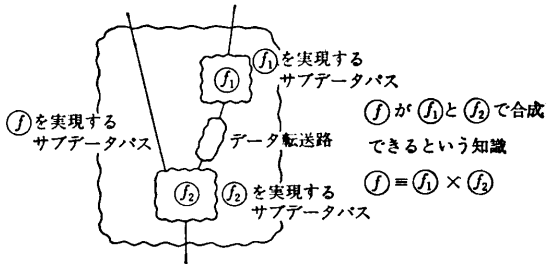


図 7 関数合成による機能検証の概念
Fig. 7 Verification of a function by function synthesis.

```
(FIND−FUNCTION (*Z<−*X − *Y)
  (*CTL 0 *PATH *CTL 1))
    −>(FUNCTION (^ *A *Y) *CTL 0)
        (FUNCTION (+ *X *B 1) *CTL 1)
        (FIND−PATH *A *B *PATH)
```

否定の機能および加算の機能と，これらの間のデータ転送路の存在を確認している．

〈データ転送路の探索〉

データ転送路の構成要素は，信号線およびスルー機能（2.2 節参照）をもつ機能ブロックである．これら構成要素の直列結合でデータ転送路は成っている．したがって FIND−PATH 述語は次のように定義できる．

```
(FIND−PATH *FROM *TO *PATH)
  −>(A−PATH *FROM *TO *PATH)
(FIND−PATH *FROM *TO
  (*PATH 0・*PATH 1))
    −>(A−PATH *FROM *X *PATH 0)
        (FIND−PATH *X *TO *PATH 1)
(A−PATH *FROM *TO (*NET))
    −>(PATH *FROM *TO *NET)
(A−PATH *FROM *TO (*CTL))
    −>(FUNCTION (CONN *TO *FROM)
        *CTL)
```

A−PATH はデータ転送路の構成要素を定義する述語である．

《資源競合，並列動作性制約条件のチェック》

各演算操作が使用するデータパス資源のリストは，述語 VERIFY の第2引数（すなわち変数 *PATH）の値として返されてくる．

```
(VERIFY (*D<−*S1 *F *S2) *PATH)
```

この情報について次のチェックを行う．

（1） 並列に動作しうる演算操作の使用する資源のリストに，同一資源が2度以上出現しないことの確認．

（2） 並列に動作しうる（あるいは単独で動作する）演算操作の使用する資源のリスト中に，互いに並列に動作できないデータパス資源の組が存在していないことの確認．

一般に，一つの演算操作はデータパス上で，複数の方法で実現可能である．資源競合，あるいは，データパスの並列動作性制約違反が存在した場合には，各演算操作の他の実現方法を調べる．それでも回避できないときは，データパスは仕様を満足しないと判断する．

4. データパスの補正

データパスが仕様を満たさないときの原因解析，対応策立案の一手法を示す．

4.1 原因解析

データパスが仕様を満たさない（検証失敗と呼ぶ），原因は次の3種類に分類できる．

（1） 機能の不足

（2） データ転送路の不足

（3） 資源競合，データパスの並列動作性制約違反

（3）については，3.2 節の資源競合，並列動作性制約条件のチェック結果がそのまま原因解析結果として利用できる．

本節では，（1），（2）に焦点を置いて示す．

検証失敗の原因（1），（2）は，一般には，複合して起こる．2項演算形式の演算操作では $2^4-1=15$ 通りの検証失敗パターンが，単項演算形式では $2^3-1=7$ 通りの検証失敗パターンが存在する（図8）．これら失敗パターンごとにパターン検出ルールを設ける．ルールは，機能，転送路の存在，欠落を解析する．これらパターン検出ルールの定義例を示す．

2項演算の場合

デスティネーションへの転送路	ソース1からの転送路	ソース2からの転送路	機能
×	○	○	○
○	×	○	○
○	○	×	○
○	○	○	×
×	×	○	○
×	○	×	○
×	○	○	×
○	×	×	○
○	×	○	×
○	○	×	×
×	×	×	○
×	×	○	×
×	○	×	×
○	×	×	×
×	×	×	×

○：存在

×：欠落

単項演算の場合

デスティネーションへの転送路	ソース1からの転送路	機能
×	○	○
○	×	○
○	○	×
×	×	○
×	○	×
○	×	×
×	×	×

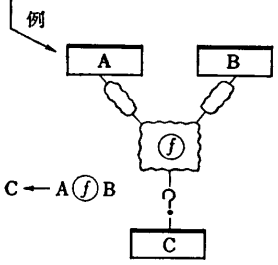


図 8 失敗パターンの概念
Fig. 8 Failure pattern concept.

```
(WHY−FAILED LACK−OF−FUNCTION
  (*D <− *S1 *F *S2) (*ALU))
    −>(TYPE ALU *ALU)
       (DO−NOT−HAVE−FUNCTION
         *ALU *F)
       (FUNCTION (*? *Z *X *Y)
         (*ALU *CTL))
       (FIND−PATH *Z *D *PATH1)
       (FIND−PATH *S1 *X *PATH2)
       (FIND−PATH *S2 *Y *PATH3)
(WHY−FAILED LACK−OF−PATH
  −TO−DESTINATION
  (*D <− *S1 *F *S2) (*Z *D))
    −>(VERIFY (*Z <− *S1 *F *S2)
         *PATH0)
       (NOT (FIND−PATH *Z *D
         *PATH1))
```

WHY−FAILED が原因を解析する述語である．WHY−FAILEDの第1引数は原因を表すキーワードであり，第2引数は検証すべき演算操作であり，第3引数は失敗した箇所を示す．最初の例 LACK−OF−FUNCTION は，演算操作で参照，あるいは，代入する資源と適切に接続された ALU は存在するが，その ALU に，演算操作で必要とされる機能が含まれていない失敗パターンを検出するルールである．この検出ルールは，失敗箇所として，ALU の名前を返す．

第2の例LACK−OF−PATH−TO−DESTINATION は，必要とされる機能を実行しうるサブデータパスおよび，演算操作の二つのソース資源からサブデータパスへの転送路は存在するものの，デスティネーションへの転送路が存在しない失敗パターンの検出ルールである．

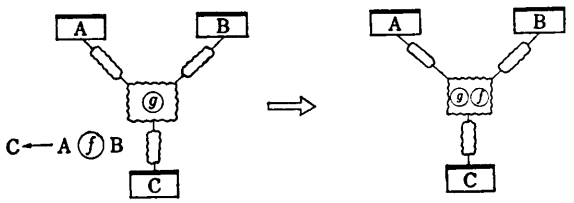
一般には，一つの演算操作に対し，複数の失敗パターンが検出されることがある．この問題は，検出ルールに優先順位を付けることにより解決する．

すなわち，失敗箇所の少ないパターンを検出するルールほど高い優先順位を付与し，最も優先順位の高いルールから順次適用していき，最初に発見した失敗パターンを解析結果とする．

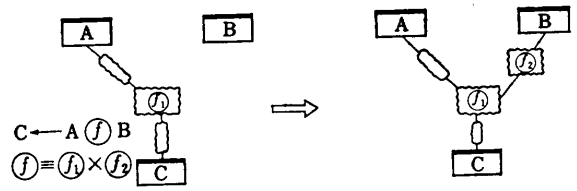
4.2 データパスの補正

失敗の原因の種類に対応し，次の3種類のデータパスの補正が考えられる．

（1） 機能の補正



（イ） 機能の追加による補正



（ロ） 関数合成による補正

図 9 機能の補正の概念
Fig. 9 Function compensation concept.

（2） データ転送路の補正

（3） 資源競合，並列動作性制約違反の補正

〈機能の補正〉

機能の補正の概念を**図9**に示す．最も単純な補正方法は，次に示すように，ALU に機能 (*F) を 追加することである．

```
(SUGGEST LACK−OF−FUNCTION
  (*D <− *S1 *F *S2) (*ALU)
  (ADD−FUNCTION
    (*ALU *OUT) <− *IN1 *F *IN2))
  −> (FUNCTION
       (*X (*ALU *OUT) *IN1 *IN2) *CTL)
```

述語 SUGGEST の第1引数は補正の種類であり，第2引数は演算操作であり，第3引数は，失敗箇所であり，第4引数が補正手法である．

関数合成に関する知識を利用すると，よりきめ細かい補正ルールを作ることも可能である．

すなわち，関数Fをより基本的な関数 $F_1 \cdots F_n$ の組合せで実現することを考え，このなかでデータパス中に含まれていない機能のみを補正（追加）する（図9(ロ)）．

〈転送路の補正〉

データ転送路の補正方法は，転送路のソース，あるいはシンク周辺のトポロジイによって異なってくる．ソース，あるいは，シンク周辺のトポロジイは次のように分類できる（**図 10**）．

（1） ソース周辺のトポロジイ

- ソースから線がでていない．

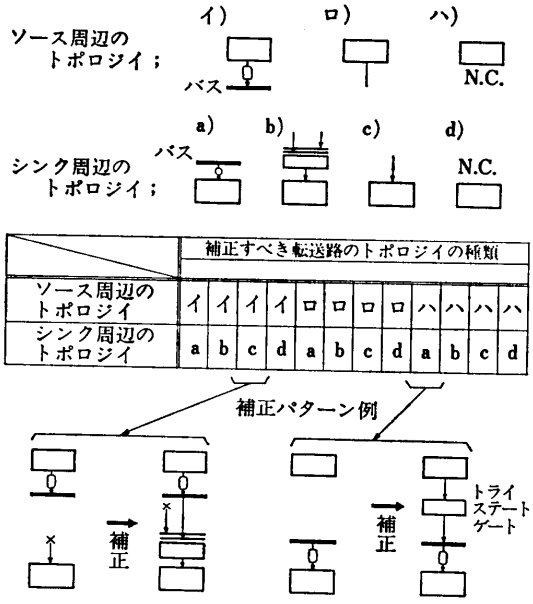


図 10 補正すべき転送路のトポロジイと補正パターンの概念

Fig. 10 Data transfer paths topology and compensation pattern concept.

- ソースがバスにつながっている.
- ソースがバス以外の線につながっている.

(2) シンク周辺のトポロジイ

- シンクに線がつながっていない.
- シンクにバスがつながっている.
- シンクにマルチプレクサがつながっている.
- バス，マルチプレクサ以外の線がシンクにつながっている.

これら，ソース，シンク周辺のトポロジイの組合せ(3×4=12 通り) ごとに，データ転送路補正ルールを設ける．そのうちの 2 例を示す.

```
(SUGGEST−PATH (*S *D)
  (CONNECT *BUS AND
    *D BY NETNAME?))
    −>(CONNECTED−TO−BUS *S *BUS)
      (NO−CONNECTION *D)
(SUGGEST−PATH (*S *D)
  ((INSERT−MULTIPLEXOR
    ?MPX−NAME BETWEEN
    *NET AND *D)
  (CONNECT *F−LEAF AND
    (?MPX−NAME 2) BY ?NETNAME))
    −>(PATH *? *D *NET)
      (NOT (MULTIPLEXOR? *?))
      (FIND−LEAF *S *F−LEAF)
```

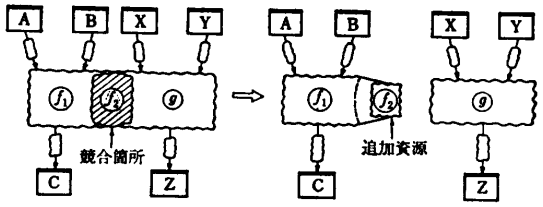


図 11 資源競合補正の概念

Fig. 11 Resource conflict compensation concept.

述語 SUGGEST−PATH の第 1 引数は，ソース (*S) とシンク (*D) の組を示し，第 2 引数は補正手法を示す.

第 1 番目の例は，ソースがバスにつながっていて，シンクに何もつながっていないならば，そのバスをシンクにつなぐようデータパスを補正するルールを示す．第 2 番目の例は，シンクにバス以外の線がつながっている場合には，シンク入力とこの線の間に 2 入力マルチプレクサを挿入し，ソースとこのマルチプレクサの残りの入力端子をつなぐようデータパスを補正するルールを示す.

〈資源競合，並列動作性制約違反の補正〉

資源競合補正の基本的考え方を次に示す (**図 11**).

(1) 競合する箇所に資源を追加する.

(2) 一方の演算操作を，この追加資源を使って実行できるようデータ転送路の補正を施す.

この処理は，機能の補正手法および転送路の補正手法の組合せで実現できる.

並列動作性制約違反はデータパス構成上は実現可能であるが，制御系の構成上実行できない場合である，しかし，データパスの補正か，制御系の補正か選択の余地があるので，本論文では，警告メッセージを出すにとどめている.

5. 実 験 例

図 1 に示した例題計算機 (SAMPLE) の論理仕様と図 3 のデータパスとの整合性検証の実験例を**図 12** に示す．検証は，DDL で記述された論理仕様をトランスレートして得られた演算操作 (図 5 参照) とデータパスの間で行う.

述語 VERIFIER が互いに並列に動作する演算操作の集合を検証する.

(1) は単純なデータ転送 (MEMORY−ADDRESS <−SC) が成功した例である．この演算操作で使用する資源名を検証結果として出力している.

(2) は互いに並列に動作する演算操作が，資源の競

```
(1)--- OP1
-(VERIFIER ((MEMORY-ADDRESS <- SC))):

(MEMORY-ADDRESS <- SC IS VERIFIED AS FOLLOWS :
   SC TO MEMORY-ADDRESS :THROUGH: ((SC T) L15 (G-3 ON) L16 (BUSB T) L9))

(*PROVED)

(2)--- OP2, OP3
-(VERIFIER ((IR <- MEMORY-READ-DATA) (SC <- 1+ SC))):

(IR <- MEMORY-READ-DATA IS VERIFIED AS FOLLOWS :
   MEMORY-READ-DATA TO IR :THROUGH: (L10 (G-5 ON) L11 (BUSB T) L12 (IR SET)))
(SC <- 1+ SC IS VERIFIED AS FOLLOWS :
   1+ :IN: (ALU3 INCREMENT)
   SC TO (ALU3 IN)  :THROUGH: ((ALU3 IN) ((SC T) L19))
   (ALU3 OUT) TO SC :THROUGH: ((ALU3 OUT) (L20 (MPX-3 SEL2) L21 (SC SET))))

(*PROVED*)

(3)--- OP4
-(VERIFIER ((GR0 <- GR0 + GR1))):

(GR0 <- GR0 + GR1 IS VERIFIED AS FOLLOWS :
   + :IN: (ALU1 ADD)
   GR0 TO (ALU1 A) :THROUGH: ((GR0 T) L5 (MPX-2 SEL1) L7 (G-2 ON) L8
                              (BUSA T) L14)
   GR1 TO (ALU1 B) :THROUGH: ((GR1 T) L2 (MPX-1 SEL2) L3 (G-1 ON) L4
                              (BUSB T) L13 (ALU2 THROUGH) L22)
   (ALU1 C) TO GR0 :THROUGH: (L23 (BUSY T) L25 (GR0 SET)))

(*PROVED)

(4)--- OP5
-(VERIFIER ((GR0 <- GR0 - GR1))):

(GR0 <- GR0 - GR1 IS VERIFIED AS FOLLOWS :
   - :AS: COLABORATION (ALU2 INVERT) AND (ALU1 ADD-1) THROUGH (L22)
   GR0 TO (ALU1 A)  :THROUGH: ((GR0 T) L5 (MPX-2 SEL1) L7 (G-2 ON) L8
                              (BUSA T) L14)
   GR1 TO (ALU2 IN) :THROUGH: ((GR1 T) L2 (MPX-1 SEL2) L3 (G-1 ON) L4
                              (BUSB T) L13)
   (ALU1 C) TO GR0  :THROUGH: (L23 (BUSY T) L25 (GR0 SET)))

(*PROVED)

(5)--- OP6
-(VERIFIER ((GR0 <- GR0 @ GR1))):

**CAUTION!! : GR0 <- GR0 @ GR1 CAN NOT BE EXECUTED BECAUSE
(FUNCTION @ IS NOT CONTAINED IN ALU1)
   I RECOMMEND TO :
(ADD-FUNCTION (ALU1 C) := (ALU1 A) @ (ALU1 B) WITH (ALU1 ?FUNC-ID))

NIL

(6)--- OP7
-(VERIFIER ((BR <- GR0))):

**CAUTION!! : BR <- CONN GR0 CANNOT BE EXECUTED BECAUSE
  (THERE IS NO PATH FROM GR0 TO BR)
   I RECOMMEND TO :
  (CONNECT BUSB AND BR BY ?NET-NAME)

NIL

(7)--- OP8
-(VERIFIER ((SC <- BR))):

(SC <- BR IS VERIFIED AS FOLLOWS :
   BR TO SC :THROUGH: ((BR T) L17 (G-4 ON) L18 (BUSA T) L14 (ALU1 SELA)
                       L23 (BUSY T) L24 (MPX-3 SEL1) L21 (SC SET)))

(*PROVED)

(8)--- OP9, OP10
-(VERIFIER ((SC <- GR0 + BR) (GR1 <- SC))):

(SC <- GR0 + BR IS VERIFIED AS FOLLOWS :
   + :IN: (ALU1 ADD)
   GR0 TO (ALU1 B) :THROUGH: ((GR0 T) L1 (MPX-1 SEL1) L3 (G-1 ON) L4
                              (BUSB T) L13 (ALU2 THROUGH) L22)
   BR TO (ALU1 A) :THROUGH: ((BR T) L17 (G-4 ON) L18 (BUSA T) L14)
   (ALU1 A) TO SC :THROUGH: (L23 (BUSY T) L24 (MPX-3 SEL1) L21 (SC SET)))
(GR1 <- SC IS VERIFIED AS FOLLOWS :
   SC TO GR1 :THROUGH: ((SC T) L15 (G-3 ON) L16 (BUSB T) L13 (ALU2 THROUGH)
                        L22 (ALU1 SELB) L23 (BUSY T) L26 (GR1 SET)))
**CONFLICTION!!** : (SC <- GR0 + BR) AND (GR1 <- SC) AT :
  (ALU1 L23 BUSY BUSB L13 ALU2 L22)
((INSERT-MPX ?MPX-NAME BETWEEN L26 AND GR1) (CONNECT SC AND (?MPX-NAME 1)
                    BY ?NET-NAME))

NIL
```

図 12 検証の例 (SAMPLE)
Fig. 12 Verification example (SAMPLE).

合なしで実行できる例である.

(3)は加算が ALU 1 で行われることを示している.

(4)は,減算が,否定と加算を組み合わせて実現できるという関数合成の知識を利用した検証例である.

(5)は,データパスの機能不足のため,検証失敗した例であり,ALU 1 に機能追加することを示唆している.

(6)は,単純データ転送(BR<−GR 0)が,BR への転送路欠落のため,実行できない例である.BUSB と BR を結線することを示唆している.

(7)は単純データ転送の成功例であり,ALU 1 をデータ転送路として使用している.

(8)は二つの演算操作が資源競合のため実行できない例であり,転送路の追加を示唆している.

データパスの補正の仕方は1通りではない.たとえば(6)の例において,BUSB と BR を結線するのでなく,GR 0 と BR の直接結線,あるいは,BUSY と BR の結線でも補正可能である.どの方法が最適かは,トポロジイだけからは判断がむずかしい.

また,データパスと論理仕様の整合性が低い場合,データパスの補正というより,むしろ,設計,あるいは再設計問題に近づいていく.

これらの問題に対処するためには,実験を通じ,補正ルールを高度化していく必要がある.

図12で示したように,検証成功時には,各演算操作が,どの演算器のどの機能を使い,どう実行されるかの情報が収集されている.この情報は,制御回路あるいは,μプログラム設計の基礎データとなる.

6. まとめ

ディジタルシステムの論理仕様とデータパスの整合性を解析的手法で検証し,不整合が存在する場合には,その原因解析・対策を立案する手法を提案した.検証・原因解析・対策立案に関する設計者の知識をルールの形に定式化することにより,システムの高度化を行いやすくした.実験プログラムにより,有効性を確認した.

今後,実験を通じ,ルールの蓄積・高度化を図るとともに,制御系を含む検証へと検討を進める予定である.

謝辞 日頃有益な討論をしていただき,また,プログラム作成の便宜を図って下さった情報通信基礎研究部第一研究室の諸氏に深く感謝いたします.

参考文献

1) Duley, J. R. and Dietmeyer, D. L.: A Digital System Design Language (DDL), *IEEE Trans. Comput.*, Vol. C-17, No. 9, pp. 850-861 (1968).
2) Duley, J. R. and Dietmeyer, D. L.: Translation of a DDL Digital System Specification to Boolean Equation, *IEEE Trans. Comput.*, Vol. C-13, No. 4, pp. 305-313 (1969).
3) Kawato, N., Saito, T., Maruyama, F. and Uehara, T.: Design and Verification of Large Scale Computer by Using DDL, Proc. 16th DA Conference, pp. 360-366 (1979).
4) Maruyama, F., Uehara, T., Kawato, N. and Saito, T.: A Verification Technique for Hardware Designs, Proc. 19th DA Conference, pp. 832-841 (1982).
5) 高木:データパス系機能ブロック自動設計の一手法,設計自動化研究会資料, 21-1 (1984).

(昭和59年4月11日受付)
(昭和59年6月15日採録)