論理装置のデータパス構造自動合成の一手法†

高木 茂††

プロセッサ等論理装置の動作仕様はレジスタ・トランスファ・レベルにおけるデータ転送動作，演算動作等基本動作の系列として記述されうる．本論文は，これら基本動作の系列を状態遷移図の枠組み内で記述する言語 DDL-S (Subset of the Digital System Design Language) に基づき，演算器，バス等の結合からなるデータパス構造を自動合成する手法を提案する．DDL に基づく既存の自動合成法では，動作記述に含まれる基本動作相互間の並列動作性を解析しておらず，ほぼ1演算動作に対し，1演算器を設ける手法を採っているため，合成された資源の使用効率が低くなるという問題がある．これに対し，次の合成法を提案する．(1) 動作記述に含まれる条件文，ステート文に着目し，その構文情報を利用して，基本動作相互間の並列動作／非並列動作を解析する．(2) 互いに並列に動作しない演算動作のグループに対し1演算器を合成する．グループは，(イ) 合成される演算器が非常識な構成にならない，(ロ) 準最小演算器数で全演算動作をカバーできる，の2条件を満たすように選定する．(3) 演算器合成後，転送路の合成を行う．転送路の並列動作／非並列動作を解析し，互いに並列に動作しない転送路のグループに対し，バス，あるいは共同マルチプレクサを合成する．既存アルゴリズムに比較し，演算器の使用効率が2～4倍となる実験結果を得た．

1. まえがき

論理自動合成は古くからの研究課題であり，①ブール式に基づく自動合成，②クロックを陽に指定するレジスタ・トランスファ・レベル動作記述に基づく自動合成，③クロックを指定しない命令セットレベル動作記述に基づく自動合成，等各種レベルで検討されてきた．①のレベルで論理装置全体を記述し自動合成することは非現実的である．③のレベルの代表例としては，ISPS 記述よりデータ依存関係を導き，これをもとに自動合成を行う CMU-DA システム[1)-4)] がある．しかし，論理装置設計では，動作を完了するのに必要なクロック数で性能を陽に指定したいことも多く，②のレベルの自動化が当面望まれる．

本論文は，μプロセッサ，あるいは中大型計算機の演算部，命令取り出し部等一つの状態遷移図で動作仕様を規定できる範囲を対象とし，その動作記述より，データパス構造を合成する手法について述べる．

動作仕様は，レジスタ・トランスファ・レベルの記述言語の一つである DDL[5)] のサブセット DDL-S で記述する．並列動作解析と資源の併合処理を導入した本合成法によれば，既存の手法[6)-8)] に比較し資源の使用効率を向上させうる．

既存の自動合成手順の大筋を次に示す．

(1) 動作記述より，データ転送動作，演算動作等の基本動作（マイクロ操作）と基本動作ごとの実行条件式をもとめる．

(2) 各演算動作に対し，ほぼ1対1対応で演算器を設け，この演算動作で参照，あるいは，代入しているレジスタ類と演算器との間をデータ転送路で結合する．

(3) 実行条件式に基づき制御回路を合成し，データパスと結合する．

(4) データパスの構成要素をゲートに展開する．

(5) 冗長ゲートの除去等，最適化処理を行う．

しかし，この手順に従って合成される回路は人手設計に比較し，冗長な部分が多く，資源の使用効率が低い．また，通常のデータパスでよく使われている多機能演算器の自動合成，バスの自動合成は困難である．これは手順(2)が単純すぎるためであり，本論文は，次の代替手順を提案する．

(1) 動作記述に含まれるステート文，条件文に着目し，その構文情報を利用し，基本動作相互間の並列動作/非並列動作を判定する（並列動作解析）．

(2) ハードウェア量最小化の問題を演算器数最小化の問題ととらえ，互いに並列に動作しない演算動作のグループに対し1演算器を合成する．この演算器は単機能演算器，あるいは多機能演算器となる．グループは，合成される演算器がハードウェア的に妥当な構成になるよう選定する．

(3) 演算器合成後，転送路の合成を行う．転送路の並列動作性を解析し，互いに並列に動作しない転送路のグループに対し，バス，あるいは共用マルチプレクサを合成する．

以下，第2章では並列動作解析法手を示し，第3章

† A Data Path Synthesis Algorithm for Digital System by SHIGERU TAKAGI (Musashino Electrical Communication Laboratory, N. T. T.).

†† 日本電信電話(株)武蔵野電気通信研究所

```
〈DDL-S 構文〉=((NAME システム名){〈宣言文〉}〈オートマトン文〉;
〈宣言文〉=〈INPUT 文〉|〈OUTPUT 文〉|〈REGISTER 文〉|〈REG-FILE 文〉|…;
〈オートマトン文〉=(AUTOMATON オートマトン名{〈状態文〉});
〈状態文〉=(状態名 条件 〈文ブロック〉);
〈文ブロック〉={〈文〉};
〈文〉=〈条件文〉|〈実行文〉|〈遷移文〉;
〈条件文〉=〈IF 文〉|〈CASE 文〉|〈COND 文〉;
〈IF 文〉=(IF 条件 (〈文ブロック〉)  /*Then ブロック*/
              〔(〈文ブロック〉)〕); /*Else ブロック*/
〈CASE 文〉=(CASE 条件 KEY (値1 〈文ブロック〉)
                          ⋮
                       (値n 〈文ブロック〉));
〈COND 文〉=(COND (条件1 〈文ブロック〉)
                    ⋮
                 (条件m 〈文ブロック〉));
〈実行文〉=(<-〈REGISTER 類〉〈式〉)| /*Register 代入*/
        (=〈TERMINAL 類〉〈式〉); /*Terminal 接続*/
〈式〉=〈定数〉|〈REGISTER 類参照〉|〈入力参照〉|(〈演算子〉{〈式〉});
〈演算子〉=&/*and*/|OR|∧/*not*/|@/*exclusive or*/|…/*論理演算*/
        +|1+/*add 1/|-|                          /*算術演算*/
        =|∧=/*not equal*/|>=|<=|…                /*比較演算*/
        SHIFT-RIGHT/*1 bit shift*/|SHIFT-RIGHT*/*n bit shift*/|…/*シフト演算*/;
〈遷移文〉=(->遷移先状態)
```

メタ記号 = 定義　; 定義終了　/*…*/ コメント　| または　{ } 繰り返し　〔 〕 0あるいは1回の出現

図 1 DDL-S の構文（要旨）
Fig. 1 Syntax of DDL-S.

ではデータパス構造の合成手法を示し，第4章では実験例について示す．

2. 並列動作解析

動作記述より基本動作相互間の並列動作性を導く手法を示す．最初に，動作記述について，以後の論議に必要な範囲に限って説明する．

2.1 動作記述

状態遷移表現に基礎を置く DDL 言語[5]を参考にし，そのサブセット*を LISP のS式の構文で記述する言語 DDL-S を使用する．

DDL-S の構文要旨を**図1**に示す．すなわち，

（1） システムの動作記述はレジスタ等資源の宣言文とオートマトン文よりなる．

（2） オートマトン文は複数の状態文よりなる．

（3） 状態文は1状態に対応し，複数の文（実行文，遷移文，条件文）より構成される．

（4） 実行文はレジスタ・トランスファ・レベルの基本動作（マイクロ操作）を指示する．基本動作には，データの単なる転送動作と演算を伴う演算動作がある．

（5） 遷移文は状態遷移を指示する．

（6） 条件文は複数の条件ブロックより構成され，各条件ブロックごとに条件が設定されている．条件ブロックは複数の文（実行文，遷移文，条件文）より構成される．したがって，一般には条件文はネスト構造となる．

オートマトンの中ではつねに唯一の状態のみがアクティブである．1状態は1マシンクロックの間のみ持続し，クロックの進展とともに他（あるいは自）状態へと遷移していく．アクティブになった状態では，条件文の所定の条件が満足されると基本動作が実行される．

状態遷移図，DDL-S 記述の例（簡単な例題計算機 COMP-X の仕様の一部）を**図2**, **図3**に示す．たとえば，図3において，もし，アクティブな状態が命令デコード DEC であり，命令語の命令フィールド OP のビットパターンが SFT (0100) であり，命令語の XR フィールドのビットパターンが (00) であるならば実行文

(<- ACC (SHIFT-RIGHT (GR GR-F)))

が実行され，状態 GRSET へ遷移する．上記実行文の意味は，“汎用レジスタ GR の GR-F アドレスの内容を読み出し，1ビット右シフトし，アキュムレータ ACC にセットする”である．ここで ACC および GR はおのおの16ビット幅である．

* 1システム中に含まれるオートマトン数を1に制限している．

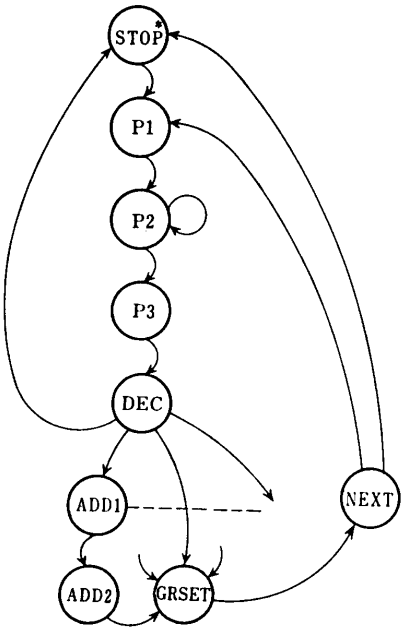


図 2 状態遷移図の例（例題計算器 COMP-X の一部）
Fig. 2 Example of state transition diagram.
STOP*：停止，P1, P2, P3：命令フェッチ，DEC：デコード，ADD 1 / ADD 2 / GRSET / NEXT：命令実行

```
((NAME COMP-X)
 (REGISTER (BR (0 15)) (SC (0 15)) CC (IR (0 15)) (ACC (0 15))  - -)
 (REG-FILE (GR (0 1) (0 15)))
 (INPUT (RPLI (0 1)) (MWRI (0 15)) DSYNC)
 (OUTPUT (RQCO (0 1)) (MARO (0 15)) (MDRO (0 15)) ASYNC)
 (MACRO (OP (IR (0 3))) (GR-F (IR (4 5)))
        (XR (IR (6 7))) (SFT (0 1 0 0)) ——)
 (AUTOMATON  ctl1

   (STOP*  ()
         (COND (RESET (-> RESET*))
               ((& START (^ RESET)) (<- START (0)) (-> p1))
               ((& (^ START) (^ RESET)) (-> stop*))))
   (P1     ()
         (<- MAR SC) (<- SC (1+ SC)) (<- RQC READ) (:= ASYNC (1)) (-> P2))
   (P2     DSYNC
         (<- MWR MWRI) (-> P3))
   (P3     ()
         (<- IR MWR) (-> DEC))
   (DEC    ()
         (CASE OP
               (HJ  (<- SC EA) (-> STOP*))
               (SFT  (CASE XR
                           ((0 0) (<- ACC (SHIFT-RIGHT (GR GR-F))))
                           ((0 1) (<- ACC (SHIFT-LEFT (GR GR-F)))))
                     (-> GRSET))
               (ADD  (<- MAR EA)  —
                     (-> ADD1))
                                 ))

   (ADD1  DSYNC
         (<- MWR MWRI) (-> ADD2))
   (ADD2  ()
         (<- ACC (+ (GR GR-F) MWR))
         (<- CC (= (+ (GR GR-F) MWR) 16D0))
         (-> GRSET))
   (GRSET ()
         (<- (GR GR-F) ACC) (-> NEXT))
   (NEXT  ()
         (IF STOP ((-> STOP*))
                  ((-> P1))))              ))
```

(1)

図 3 DDL-S 記述の例（例題計算機 COMP-X の一部）
Fig. 3 Example of DDL-S description.

なお，本論文では1相クロックを前提としている．また，本文中では論理積(*)，論理和(∨)の記号を使用する．

2.2 基本動作とトランスレーション

動作記述をトランスレートし，基本動作と，各基本動作ごとの実行条件式を得る．

2.2.1 基本動作

動作仕様の観点からすると一つの実行文は一つの基本動作を意味する．しかし，論理合成をしやすくするため複雑な構文をもつ実行文は次に示す規準に従って分解することにする．なお，分解されてできた物も，そうでない実行文と合成アルゴリズム上で区別していないため，同様に基本動作を意味することとする．

(1) メモリ，レジスタファイルへの参照がある場合，アドレス参照系とメモリ参照系に分解する．
たとえば，汎用レジスタ GR の BR アドレスの内容とレジスタAの内容を加算しAにセットする実行文は次のように分解する．

例) (← A (+ A (GR B)))
⇓
(← A (+ A GR)) (データ参照)
(← (GR ADDRESS) B) (アドレス参照)

ここで (GR ADDRESS) はレジスタファイル GR のアドレス入力端子を意味する．

(2) データパスは，たかだか2項演算器で構成されることが多いため複雑な式は2項演算形式に分解する．

(← X (f A B)) (2項演算形式)

もしここで演算子 f を DDL-S で用意されている &, OR 等の基本演算子（図1参照）に限定すると，合成された演算器の各機能もこれら基本演算に限定される．

しかし，たとえば，(OR A (^ B)) のような複数の基本演算子を含む式をそのまま実行する演算器も現実に存在する．これら演算器を合成可能とするため，複合演算子という概念を導入する．複合演算子とは，二つ以上の基本演算子を含む式を，次の例で示すように，2項演算形式で表現する手段である．

例) (OR A (^ B)) ⇒ (OR1 ^ A B)

OR1^ が複合演算子である．複合演算子の種類は多数考えうるが，実際の演算器でよく出てくるものに限定した（**表1**）．

複合演算子を使っても2項演算形式にならない式は中間変数を導入して複数に分解する．

表 1 複合演算子の種類
Table 1 Composite operators.

タイプ	複合演算の種類	複合演算子	備考
I	(& A (^ B))	&1^	A, B はレジスタ等資源を示す
	(& (^ A) B)	&^1	
	(^ (OR A B))	&^^	
	(^ (@ A B))	^@	
	(OR A (^ B))	OR1^	
	(^ (& A B))	OR^^	
II	(+ (ⓘ A B) (ⓙ A B))	ⓘ+ⓙ	+; 加算演算子 ⓘ, ⓙ; タイプ I の複合演算子あるいは基本演算子 ⓘ+ⓙ; 文字的結合を意味する

例) (← A (Shift-left (OR A (^ B))))
⇓
(← A (Shift-left V-1))
(← V-1 (OR1^ A B))

V-1 が中間変数である.

2.2.2 実行条件式

オートマトン文, 条件文の構文解析を行い, 各実行文 (基本動作) が選択される迄の条件の論理積を求めたものが実行条件式である. たとえば図3の(1)の実行文の実行条件は次の論理式となる.

(& DEC (= OP SFT) (= XR (0 0)))

この論理式が真となる, すなわち, DEC ステートであり, 命令レジスタの OP フィールドが SFT コード (0 1 0 0) に一致し, 命令レジスタの XR フィールドが (0 0) ならば実行文(1)が実行される. 実行条件式の求め方は原論文[6]を参照されたい.

2.3 並列動作解析

二つの基本動作 OP_i, OP_j のおのおのの実行条件式を C_i, C_j とすると, 並列動作性判定問題は $C_i * C_j = 0$ (0はつねに偽である論理式) の証明問題に帰着できる. すなわち,

(1) $C_i * C_j = 0$ ならば OP_i, OP_j は並列に動作しない.

(2) $C_i * C_j \neq 0$ ならば OP_i, OP_j は $C_i * C_j$ が真となるとき並列に動作する.

しかし C_i, C_j はレジスタ等の内部記憶, 入力端子参照を含む式であり, $C_i * C_j$ の値を直接かつ厳密に求めることは計算量が多くなる.

計算せずとも, 多くの場合, 並列動作性を推論できることを示す.

まず, 次に示す手続きにより, 各状態文の構文を図4のような木構造で表現する. なお, 木のノードを根, 葉, 内部ノード (並列実行ノード, 排他実行ノードの2種類ある) に細分類する.

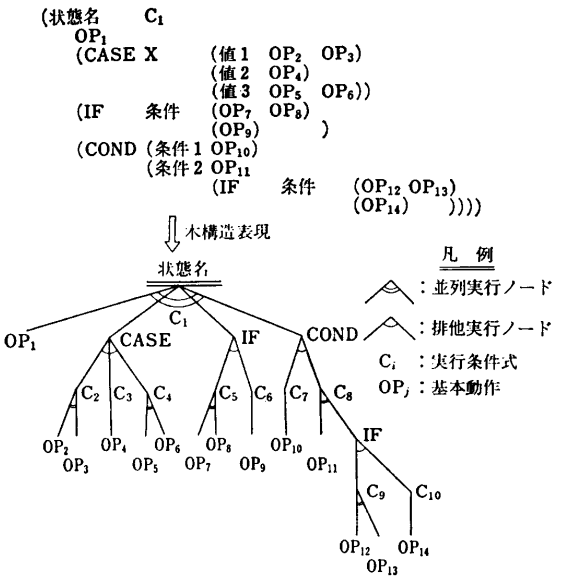


図 4 1状態の構文を表す木
Fig. 4 Tree representing control structure of a state.

(ステップ1) 根を状態に対応付ける. ステップ2に進む.

(ステップ2) ノード N_i に含まれる各実行文, 遷移文, 条件文について,

(イ) 実行文, 遷移文であれば, N_i の下に葉 L_j を生成し, その実行文, 遷移文に対応付ける.

(ロ) 条件文であれば, N_i の下に排他実行ノード N_k を生成する. N_k の下に, 条件文中の各条件ブロックごとに並列ノード $N_{k1}, \cdots, N_{km}$ を生成し対応付ける. $N_{k1}, \cdots, N_{km}$ に対し, (ステップ2) を再帰的に適用する.

DDL-S の構文規則(図1)より, 葉のつながるノードは根, あるいは並列実行ノードのみという性質をもつ. 葉のつながる並列実行ノードが選択されるまでの累積条件 (2.2.2項で示した実行条件式 C_i に一致する) をこのノードに帰属させる.

ここで, 述語 Known-to-0 を導入する. Known-to-0(X) とは, 『論理式 X について $X=0$ であることが知られているならば真(T)であり, さもなくば偽(F)である』ことと定義する.

次に, 得られた木と, (1) 1時刻には1状態のみアクティブである, (2)条件文中の条件ブロックは排他的にしか条件が成立しない, (3)条件ブロック内は並

列に実行される，という構文知識を利用し，次の並列動作推論法を導入する．

〔推論法 1〕 C_i, C_j の対応付けられたノードが別の状態の木に属するならば $C_i * C_j = 0$ であり，

$$\text{Known-to-0}(C_i * C_j) = T$$

である．

〔推論法 2〕 C_i, C_j の対応付けられたノードが同一木に属し，この二つのノードから根のほうへのパスが排他実行ノードで合流するならば

$$C_i * C_j = 0$$

であり，

$$\text{Known-to-0}(C_i * C_j) = T$$

である．

〔推論法 3〕 C_i, C_j の対応付けられたノードが同一木に属し，この二つのノードから根のほうへのパスが並列実行ノードで合流するならば，$C_i * C_j$ の値は計算しなければ決定できず（しかし構文の意味からして，$C_i * C_j \neq 0$ の確率は高いと考えられる），

$$\text{Known-to-0}(C_i * C_j) = F$$

である．

たとえば，図4においては，

$\text{Known-to-0}(C_2 * C_4) = T$,

$$\text{Known-to-0}(C_2 * C_5) = F \cdots$$

となる．

この推論法では $C_i * C_j$ の値を計算していないので，並列動作性を完全には推論できない．しかし，本来 $C_i * C_j \neq 0$ のものを $\text{Known-to-0}(C_i * C_j) = T$ と推論することはない．したがって，この推論結果にもとづいて作られる，並列に動作しない演算動作，あるいは転送路のグループに論理的矛盾（すなわち，互いに並列に動作する演算動作，あるいは転送路を同一グループとすること）が生じないことは保証される．

3. データパス構造の合成手法

演算器，転送路すべてを同時に扱うアルゴリズムは複雑となる．演算器系設計，転送路系設計というサブ問題に分解し，各サブ問題ごとに自動合成アルゴリズムを示す．これらアルゴリズムに共通する思想は「並列に動作しない資源を併合する」である．

3.1 演算器合成

演算器系の設計は，演算動作間で演算器の競合をおこさせないという制約条件のもとに演算器数最小の構成を求める問題とみなせる．この問題は，論理的には，次に示す手続きにより解決できる．

（1） 全演算動作を互いに並列動作しない演算動作のグループに分割する．

（2） グループ数が最小となる分割を見つけ，各演

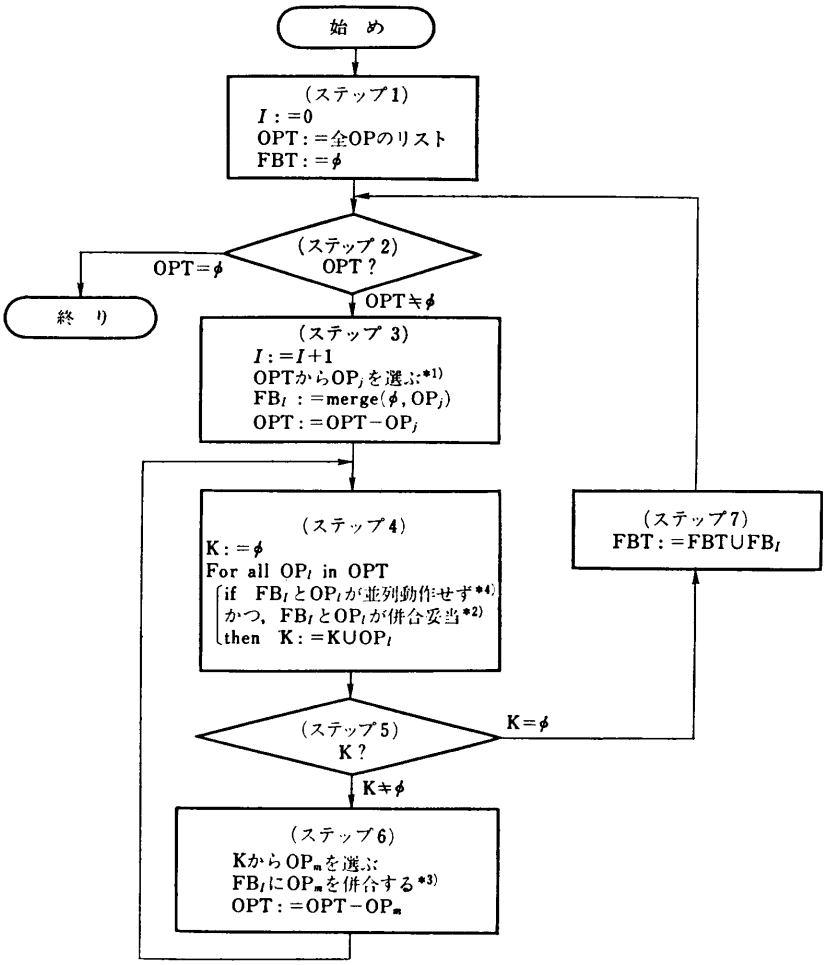


図 5 演算器合成アルゴリズム

Fig. 5 Function block synthesis basic algorithm.

ここで，− は差集合，U は和集合，φ は空集合，OP は演算動作を表す．

*1),*2),*3) 本文参照

*4) FB_I に併合された OP と OP_l が互いに並列動作しないとき，FB_I と OP_l は並列動作しない．

算動作のグループごとに1演算器を合成する.

しかし，現実的な問題として，(1)グループ分割は組合せ問題であり，グループ数最小の分割を見つけるには時間がかかる，(2)各グループがハードウェア的に妥当な演算器になるとは限らない，等が存在する.

そこで，本論文では演算器構成の妥当性判定ルールを設け，これに違反しないようグループを順次求め，発見的にグループ分割をする手法を採る.

図5に具体的アルゴリズムを示す．ステップ4, 5, 6のループが一つの演算動作のグループを構成し，同時に演算器FB_Iを合成するための手続きである．適当な演算動作OP_iを選び，これを核とした演算器を設け，この演算器と並列動作せずかつ，演算器構成の妥当性条件を満たす演算動作を順次この演算器に併合していく．演算動作の数をnとすると，このアルゴリズムの計算量は$O(n^2)$である.

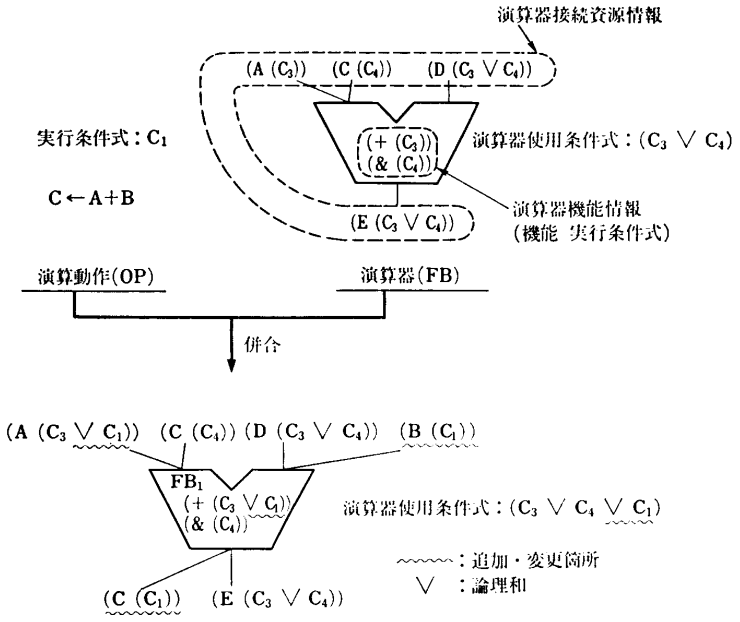


図6 併合処理の例
Fig. 6 Example of merge procedure.

このアルゴリズムのキイポイントである演算器の併合処理，演算器構成の妥当性判定ルール，ヒューリスティクスについて述べる.

1) 併合処理

演算器は次に示す3種類の情報により規定できる.

(1) 演算器接続資源情報：演算器の各入出力ポートに接続される入出力資源名と，各入出力資源ごとの入出力条件式.

(2) 演算器機能情報：演算器で実行すべき機能種別と，各機能ごとの実行条件式.

(3) 演算器使用条件：演算器が使用中である条件式.

併合処理とは，演算器FBに新たに演算動作OPを実行させるため，上記3情報のおのおのに対し，次の追加，変更処理を施すことである.

〔処理1〕 OPで参照・代入する資源がFBの対応する入出力ポートにすでに接続されているならば，その入出力条件式をOPの実行条件式との論理和に変更する．接続されていなければ，入出力資源名，入出力条件(OPの実行条件そのもの)をFBに追加する.

〔処理2〕 FBがOPの機能をすでに含んでいるならばその実行条件式をOPの実行条件式との論理和に変更する．含まれていないならば，機能，実行条件式をFBに追加する.

〔処理3〕 演算器使用条件をOPの実行条件との論理和に変更する.

併合処理の例を**図6**に示す.

2) 演算器構成の妥当性判定ルール

妥当な演算器構成は，自動展開システムや設計者に強く依存するので判定プログラムは変更容易なルール形式にするのが得策である．併合しようとする演算動作OPの機能種別をT_{OP}，演算器にすでに含まれている機能を$T_1 \cdots T_n$とする．本論文では，T_{OP}および$T_1 \cdots T_n$の機能を包含する演算器構成が次の条件のどれかを満足すれば併合妥当と判定するルールを使う.

(条件1) すでにゲート展開済の演算器がDAシステム登録されており，そのなかに存在する.

(条件2) DAシステムにその組合せを自動的にゲートに展開できるプログラムが存在する.

(条件3) 設計者が陽に常識的と宣言した組合せに一致する.

3) ヒューリスティクス

(i) 核演算器設定法

図5のステップ3では，以後の併合処理の核となる演算器を生成する．(i)核演算器の個数，(ii)核にす

表 2 一致度ベクトルタイプ間の優先度
Table 2 Priority of match vector.

一致度ベクトルのタイプ		優先度
基本動作の種類	演算器の種類	
2項演算	2項演算器	1
単項演算	2項演算器	2
単項演算	単項演算器	3

る演算動作の選択法，をどのようにするかによって，最終結果は異なってくる．本論文では，最も単純なヒューリスティクスを使用している．すなわち，

（イ） 核演算器は単数とする．

（ロ） 核とする演算動作はランダムに選択する．ただし，2項演算動作と単項演算動作が存在する場合は2項演算動作を優先して選択する．

（ii） 併合演算動作の選択法

ステップ6で FB_I に併合しうる演算動作 OP の一覧表Kのなかからどれを選択するかによって最終結果は異なってくる．演算動作が FB_I の形に近ければ近いほど，併合時に必要となる付加ハードウェア量が少なく，併合の効果が高いと考えられる．この考えに基づき，演算動作と演算器の一致度ベクトルを定義し，一致度ベクトル間の優先順位による選択法を使用する*．

2項演算動作 OP_i，2項演算器 FB_j の場合の一致度ベクトルの定義を示す．

V_{ij}＝(機能の一致/不一致，出力資源の一致/不一致，ソース1の一致/不一致，ソース2の一致/不一致)

FB_i の入出力資源，あるいは，機能に演算動作 OP_i のそれが含まれていれば一致“1”であり，含まれていなければ不一致“0”である．演算器のタイプ（2項/単項）と演算動作のタイプ（2項/単項）の組合せにより一致度ベクトルをタイプ分けする．

優先順位付けは次による．

（1） タイプ間の優先順位付けは**表2**による．

（2） 同一タイプ内の優先順位は，要素の“1”の多い順，同一“1”数では，演算タイプの一致，入力の一致，出力の一致の順とする．

例) (1011)→(1110)→(1101)→(0111)

3.2 転送路合成

演算器合成後，演算器とレジスタ類との間のデータ転送路を合成する．転送路についての論理的な情報（転送元 転送先 転送条件式）は，演算器の接続資源情報および，データ転送動作より導くことができる．導き方は自明である．

バス，マルチプレクサなど転送路系の物理的資源については，演算器合成時の“ハードウェア構成の妥当性確保”という制約条件は少ない* と考えられるので，転送路の並列動作性のみに着目した併合処理を導入する．

3.2.1 転送路の並列動作性解析法

転送条件式は2.2節で求めた基本動作の実行条件式の論理和となる．二つの転送条件式を T_i, T_j を

$T_i = C_{i1} \vee C_{i2} \cdots \vee C_{im}$

$T_j = C_{j1} \vee C_{j2} \cdots \vee C_{jn}$

とすると，その論理積は次式となる．

* 演算動作と演算器の一致度的考えは，文献3)にも見られる．

* ファンアウト数制限等の細かい制約条件については，詳細論理設計レベルで解決する問題ととらえている．

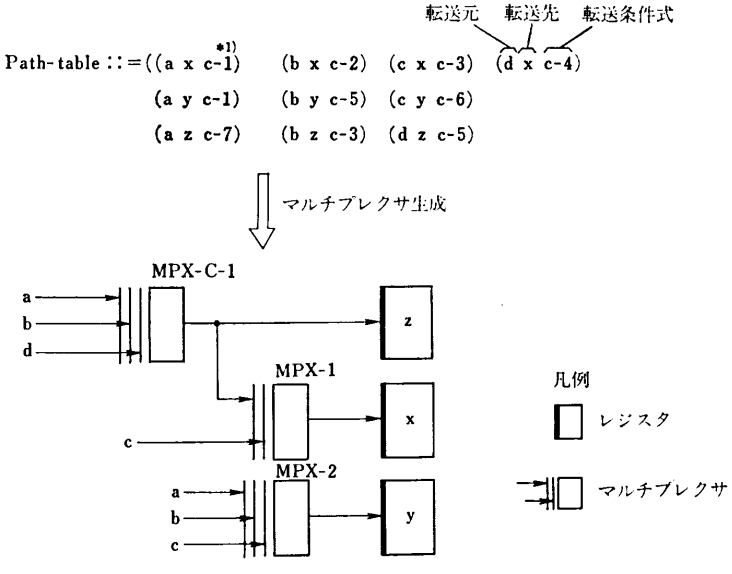


図 7 マルチプレクサ合成例
Fig. 7 Example of multiplexor synthesis.

*1) 見やすくするため，
$\forall i, j\ i \neq j$ について
Known-to-0(C-i*C-j)=F
Known-to-0(C-i*C-j)=T
とする書き方をとっている．

$$T_i * T_j = C_{i1} * C_{j1} \vee C_{i1} * C_{j2} \cdots \cdots \vee C_{i1} * C_{jn} \cdots C_{im} * C_{j1} \vee C_{im} * C_{j2} \cdots \cdots \vee C_{im} * C_{jn}$$

2.2節で述べた並列動作解析により，任意の二つの C_k, C_l に対し，Known-to-0($C_k * C_l$) の T/F が推論されている．これと，次に示す推論規則により，Known-to-0($T_i * T_j$) の T/F を決定する．

〔推論規則 1〕 Known-to-0(X)=F あるいは Known-to-0(Y)=F ならば Known-to-0($X \vee Y$)=F である．

〔推論規則 2〕 Known-to-0(X)=T かつ Known-to-0(Y)=T ならば Known-to-0($X \vee Y$)=T である．

Known-to-0($T_i * T_j$)=T ならば二つの転送路は並列に動作しない．

3.2.2 転送路合成アルゴリズム

（1） バス合成アルゴリズム

次の場合，転送路のグループをバスとする．

（イ） 互いに並列動作しない転送路．

（ロ） ソースを同一とする転送路．

Tseng[4] のアルゴリズムを簡単化したものを使用する（付録1）．この手法では，2本以上の併合可能な転送路はすべてバスとするためあまり意味のないものも合成されうる．バスとして意味のある構成とは，

（イ） バスには複数のソースがつながる

（ロ） バスから複数のデスティネーションにつながる

（ハ） 多数の転送路が併合されている

の場合と考えられる．

そこで本論文では，付録1に示したアルゴリズムにより合成されうるバスのうち，上記(イ),(ロ)の条件を満たし，かつ3本以上の転送路を含むものを選び，バスにすることとした．

（2） マルチプレクサ合成アルゴリズム

バス合成後，次の2種類のマルチプレクサを合成する．

（イ） 資源Aを転送先とする転送路が複数ある場合，資源Aの入力にはマルチプレクサが必要である．

（ロ） 入力にマルチプレクサを必要とする資源 $A_1, \cdots, A_n$ が存在し，各資源 A_i への転送路のサブ集合が転送先 $B_1, \cdots, B_m$ を共通にし，かつサブ集合同士が互いに並列に動作することがなければ，$A_1, \cdots, A_n$ と $B_1, \cdots, B_m$ の間に共用マルチプレクサを設けることができる．

(1)は，ハードウェアを正常に動作させるために必要な措置であり，(2)はハードウェア量を減らす措置

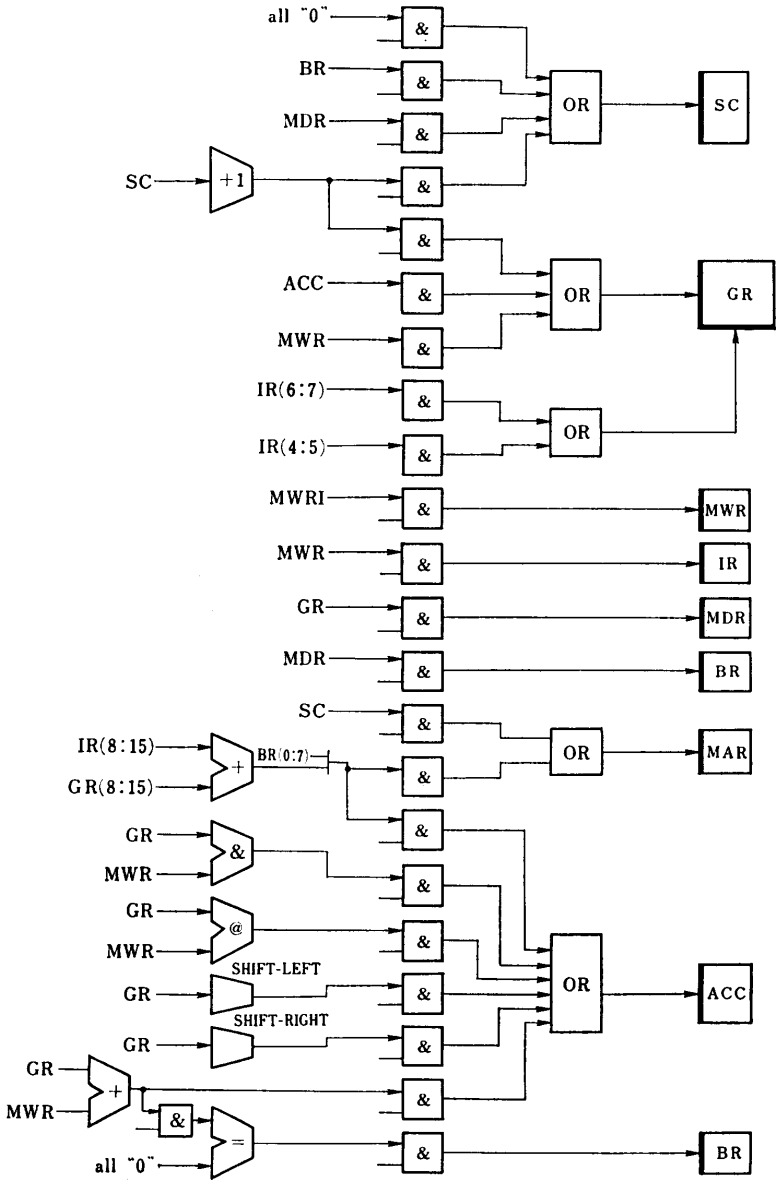


図 8 既存手法による合成例 (COMP-X の主要部)
Fig. 8 Data paths synthesized by conventional method.

である.

マルチプレクサ合成アルゴリズムを付録2に示す. 共用マルチプレクサ合成後, 資源の入力マルチプレクサを合成する. **図7**に合成例を示す.

MPX-C-1 が共用マルチプレクサであり, MPX-1, MPX-2 が資源入力マルチプレクサである.

MPX-C-1 では, 資源 a, b, d から資源 x への転送路と, 資源 a, b, d から資源 z への転送路が併合されている.

バス合成後では共用マルチプレクサの合成される可能性は少ない. バス構成を前提としないハードウェア構成時に共用マルチプレクサは有効と考えられる.

共用マルチプレクサを求める計算時間は $O(n^2)$ であり, 資源入力マルチプレクサを求める計算時間は $O(n)$ である.

4. 適 用 例

図2の DDL-S 記述例で示した例題計算機 COMP-X, について, 既存手法による合成結果を**図8**に, 本論文の手法による合成結果を**図9**に示す. 本例題では複合演算子は出現しなかった. 従来手法では演算器が8個合成されるのに対し, 本論文の手法では2個であり, 演算器の使用効率は4倍になっている. なお, COMP-X の仕様を満足させるために必要な最小演算器数も2個である. 演算器 FB-1 は7種類の機能をもつ. 演算器 FB-2 は all "0" を検出する機能のみをもつ. バスは2本合成され, 共用マルチプレクサは存在しない. バス構成の妥当性はトポロジイだけからは

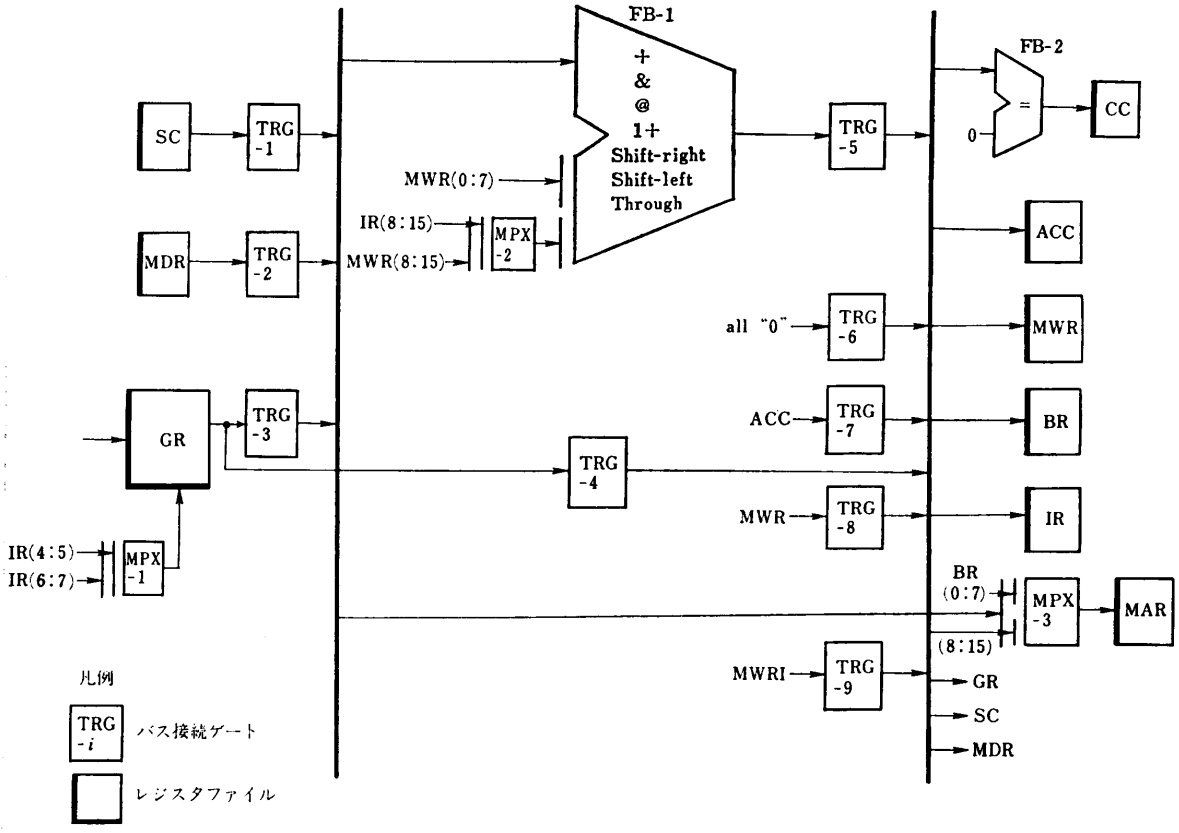


図9 提案手法による合成例 (COMP-X の主要部)
Fig. 9 Data paths synthesized by the proposed method.

表3 処理時間 (COMP-X)
Table 3 CPU time to synthesize COMP-X.

トランスレート 並列動作解析	演算器合成	バス合成	マルチプレクサ合成	バス接続ゲート合成	合計
15 秒	27 秒	16 秒	2 秒	0.5 秒	60.5 秒

判断がむずかしい.

処理時間を**表3**に示す. 使用計算機の性能は約2 MIPS である. COMP-X のソースステートメントは96行であり21状態を含んでいる.

図9では, 任意の二つの機能は併合できるという演算器構成の妥当性判定ルールを使用しているため, 最小演算器数となっている. 妥当性判定ルールを変えた場合演算器数は変化する. たとえば,

(1) SHIFT-RIGHT, SHIFT-LEFT 同士は一緒にできるが他とは一緒にできないというルールを使用すると演算器数は3個になる.

(2) (1)の条件にさらに, 1+(1を加える機能)は他機能と一緒にはできないというルールを追加すると演算器数は4個になる.

(3) どの二つの機能も併合できないとすると, 演算器数は従来手法のそれに一致する.

本論文では1オートマトン内での合成法を示したが, 複数オートマトンを含む装置に拡張するには,

(1) 異なるオートマトンに属する基本動作間は並列に動作する可能性があるとみなすならば, 本手法は, 各オートマトンごとに適用できる.

(2) (1)の仮定を使用しないならば, 異なるオートマトンに属する基本動作間の並列動作判定法を開発する必要がある.

5. ま と め

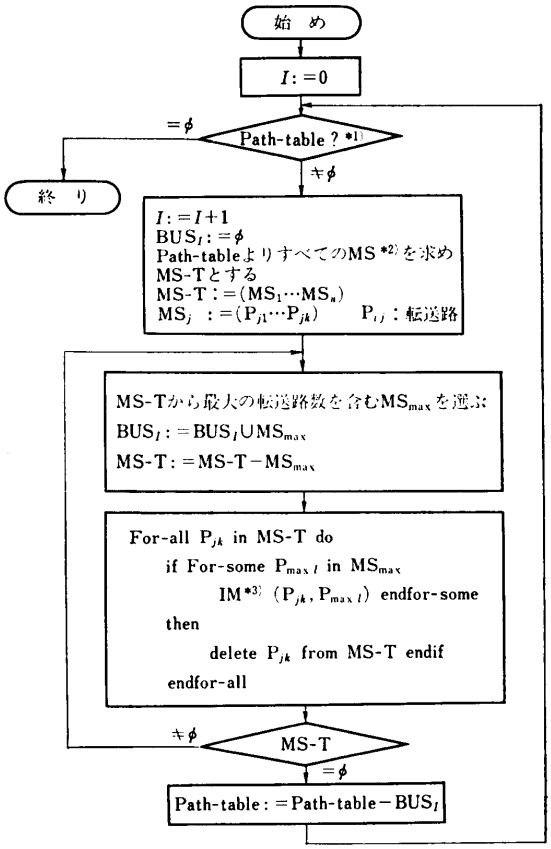
並列動作解析と資源の併合処理を取り入れたデータパス構造の合成手法を提案した. 本手法によれば, 従来手法に比較し資源の使用効率の高いデータパス構造が得られる見通しを得た. また演算器構成の妥当性判定ルールを変化させることにより, 同一仕様に対し様様なデータパス構造の得られることを確認した.

今後, 制御系の自動合成手法およびデータパス系との結合手法の検討を進める予定である.

参 考 文 献

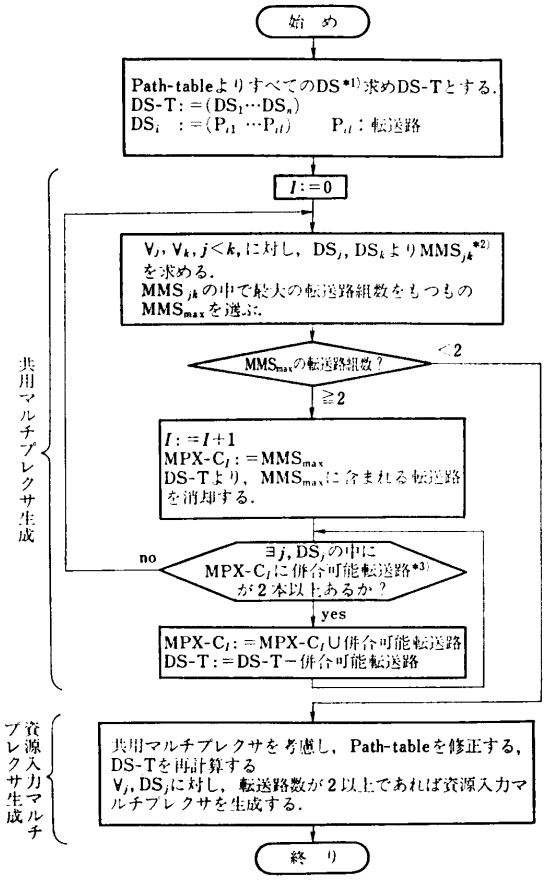
1) Paker, A. et al.: The CMU Design Automation System, Proc. 16th DA Conference, pp. 73-80 (1979).
2) Snow, E. A., Siewjorek, D. P. and Thomas, D. E.: A Technology-Relative Computer-Aided Design System, Proc. 15th DA Conference, pp. 220-226 (1978).
3) Tseng, C. and Siewjorek, D. P.: FACET: A Procedure for the Automated Synthesis of Digital Systems, Proc. 20th DA Conference, pp. 490-496 (1983).
4) Tseng, C. and Siewjorek, D. P.: The Modeling and Synthesis of BUS Systems, Proc. 18 th DA Conference, pp. 471-478 (1981).
5) 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).
6) 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).
7) 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).
8) 星野, 永谷, 中島: VLSI のゲート自動生成, 電気学会電子デバイス研究会資料, EDD-83-37 (1983).

付録 1. バス合成アルゴリズム



ここで, *1) path-table は転送路の集合.
*2) MS はソースを同一とする転送路の集合.
*3) IM (P_x, P_y) は, ソースが異なりかつ互いに並列に動作することがある転送路の組.

付録 2. マルチプレクサ合成アルゴリズム



ここで, *1) DS はデスティネーションを同一とする転送路の集合.

*2) MMS_{jk} は, DS_j の転送路を P_{jl}, DS_k の転送路を P_{km} としたとき, P_{jl} と P_{km} が並列動作せずかつソースが同一である組 (P_{jl}, P_{km}) の集合である. また (P_{jl}, P_{km}), (P_{jx}, P_{ky}) は互いに並列動作しない.

*3) 併合可能転送路は, MPX-C_I と並列動作せず, かつそのソースが MPX-C_I のそれにすでに含まれている DS_j の転送路.

(昭和59年2月 3 日受付)
(昭和60年2月21日採録)