設計自動化　33－2
（１９８６．７．１５）

設計プランを用いたレジスタトランスファ合成法

小林　一夫　　若林　春夫
ＮＴＴ電気通信研究所

ここでは、対象の装置の動作仕様からレジスタトランスファ記述を合成する手法を述べている。抽象度の高い仕様から試行錯誤を繰り返して合成を進めるために、段階的な合成階梯を設定する。さらに、各階梯の設計制約に応じた設計様式（パイプライン制御法、直列制御法等）で合成できるようにするために、各設計様式を実現する合成アルゴリズムを設計プランとして定式化し、設計制約との適合性を評価して与えられた仕様に適用する。簡単なプロッセサの動作仕様を例に、この方法による合成結果も述べている。

AN APROACH TO REGISTER_TRANSFER SYNTHESIS USING DESIGN PLANS

Kazuo KOBAYASHI　　Haruo WAKABAYASHI
NTT Electrical Communication Laoratories,Musashino-Shi, Tokyo 180 Japan

This paper presents a new procedure for computer aided design of digital systems. The procedure uses design plans whitch are composed of logic synthesis algorithms and generates hierarchically register-transfer structure from abstract behavior. Each of those algorithms produces the structure based on specific design style (e.g. pipeline control manner or sequencial control manner) that meets design restrictions given at several levels of abstract. As a result of preliminary experimentation we also show the control structure of simple processer produced by the algorithm for pipeline control manner.

1　まえがき

論理装置の大規模化、高集積化に対応して、論理設計の省力化を図るため、機能設計より上位の仕様設計の階梯からの合成(または展開)が望まれるようになってきている。そのため、具体的なハードウェア構造を意識しない動作仕様で装置を表し、それから下位の機能記述(レジスタトランスファ記述)ないし論理接続記述を合成する手法が研究されるようになってきた[1)-5)]。しかし、それらの研究では、逐次(直列)型の制御構造、バス形式のデータパス構造等の特定の設計様式が前提となっており、必ずしも設計者の意図する設計様式が得られるとは限らない。抽象度レベルが高い動作仕様からの合成に対しては、与えられる設計制約が多様なため、設計制約に対応した設計様式がとれる必要がある。さらに、設計制約には、性能やコストの要求等のように合成の初期段階に与えられるものから、使用可能な素子やファンイン・ファンアウト制限等の合成が進んだ後で求められるものまで合成のフェーズに応じて異なるものがあり、初期からすべてを与えるのは困難である。

これらの課題に対処するために、ここでは、段階的に合成を進め、各合成階梯における設計制約に応じた設計様式でレジスタトランスファ記述を作成する方法を提案する。本稿では、上記の実現法として、段階的に合成を進めるための合成サイクルのモデルを報告する。さらに、設計制約に応じて適当な設計様式を選択可能とするために、各設計様式を実現する合成アルゴリズムを定式化(定式化された合成アルゴリズムを設計プランと呼ぶ)する方法を報告する。

以下、2章では、本手法の前提として、対象の装置を表す動作仕様と合成結果を表すレジスタトランスファ記述の記述規定を明確にする。3章では、本手法の特徴である段階的合成サイクルのモデルと設計プランの構成について述べる。4章では、制御構造の合成アルゴリズムを典型例としてとりあげて、レジスタトランスファ合成手順を述べる。最後に、5章では本手法の適用結果を考察する。

2　前提条件

2.1　動作仕様の記述

装置の仕様には、対象の装置の実現すべき性能と機能の規定が含まれるが、ここでは、実現すべき機能を動作仕様として表したものを扱う。なお、実現すべき性能は設計制約として、合成過程で与えられるものとする(図1参照)。動作仕様は、一般的には装置の基本動作を時系列的に連ねた手続きによって表すことができるので、以下では、次の規約で表されていることを前提とする[6)]。

(1)基本動作

動作仕様は、資源に対する論理的値の転送動作(演算等の値の操作も含む)を記述単位として表す。

(2)資源

ハードウェア構造を意識しない動作を表すため、資源としては対象装置の外部

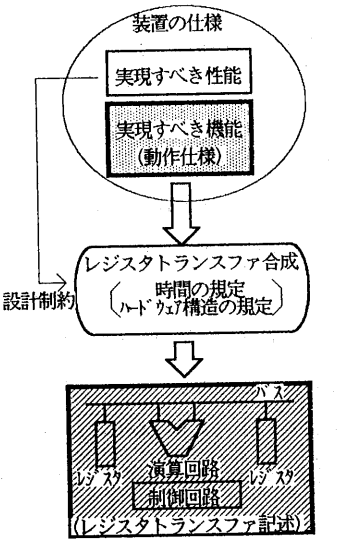


図1　動作仕様とレジスタトランスファ記述

（接続される他の装置または対象装置を利用するソフトウエア）との情報の出入り口となるものを扱い、情報の構造を指定する。

(3)時間

時間の規定は、原則としてハードウェア構造を決めるときに作られるが、装置間信号インタフェースのように動作仕様として時間規定が必要な場合があるので、その場合は、動作間に演算サイクル（クロックの周期）の整数倍の制限時間を指定する。

図２に、上記の規定にもとずいてＣ言語風の構文で動作仕様を記述した例を示す。

```
% 資源の定義
    ads, dac, df.
    adr<0:15>, scc<0:15>, mem<0:15>, dat<0:15>, ir<0:15>.
    f<0:3>:=ir<0:3>, r1<0:1>:=ir<4:5>, b2<0:1>:=ir<6:7>
% 動作の定義                          ,d2:=ir<8:15>.
 exec_ctl() {                        % 実行制御仕様
  while (run=on) {
    ir:=mem(scc);            % 命令フェッチ
    iexec(f);                % 命令実行
    scc:=scc+1; }}           % アドレス更新
 iexec(X) {                          % 命令仕様
  switch (X) {
    case 'L'   :reg[r1]:=mem(a2);
    case 'ST'  :mem(a2):=reg[r1];
    case 'ADD' :reg[r1]:=reg[r1]+mem(a2);
    case 'JL'  :scc:=a2;
    case 'DIAG':switch(r1) {
                case 0:df:=1;
                default:df:=0;} }}
 mem(X) {                            % メモリアクセス仕様(Write)
  ads:=1 {time_restriction};
  adr:=X;
  dat:=mem;
  wait (dac=1) {restriction_end=2e};}
 mem(X) {                            % メモリアクセス仕様(Read)
  ads:=1 {time_restriction}
  adr:=X;
  wait (dac=1) mem:=dat{restriction_end=2e};}
 a2() {                              % アドレス計算仕様
  a2:=reg[x2]+reg[b2]+d2;}
```

注)Ｃ言語風に表しているが、動作の開始から終了までの制限時間×は、{time_restriction}と{restriction_end=x}とで表す。:=は右辺から左辺への値の転送を表す。

図2　動作仕様の記述例

２．２　レジスタトランスファ記述

合成結果を表すレジスタトランスファ記述としては、ハードウェア構造を表すために、演算回路、レジスタ、マルチプレクサ等を基本要素としたデータパスの構造、および演算やデータ転送等の制御系列を記述する必要がある[7]。　ここでは、以下の規約でそれらを記述する。

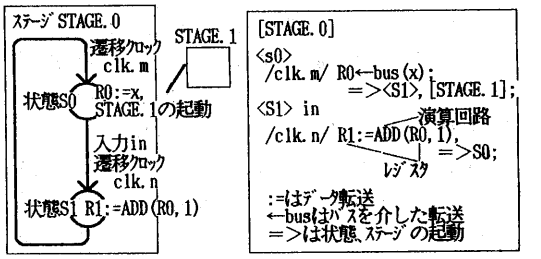
(1)基本要素

レジスタ（記憶回路）、演算回路およびバス回路は陽に表現する。ただし、制御回路、マルチプレクサ、クロック分配系回路、電気条件調整回路はゲート回路合時に合成されるので、陽には表さない。

(2)状態遷移

制御系列は、クロックに同期して状態が変化する状態遷移動作として表す。このときのクロックには、多相を扱う。また、１つの状態遷移の集合（これをステージと呼ぶ）では、その中のトークンは１つ以下とする。

図３にレジスタトランスファ記述の構成を示す。



(1)状態遷移図　(2)レジスタトランスファ記述例

図3　レジスタトランスファ記述の構成

３　基本方針

３．１　段階的合成サイクルモデル

動作仕様からレジスタトランスファ記述への合成過程を複数の階梯に分け、各階梯の設計制約に応じて適当なアルゴリズムを選択して合成を進める方法をとる。その際、上位の合成階梯の設計制約を満たす設計様式が、下位の合成階梯の設計制約と適合しない場合は、上位の設計様式を再選択（合成の試行錯誤）する必要がある。この合成の試行錯誤を短いサイクルで可能とするため、合成階梯を設計様式の選択肢の有無に応じて階層化し、次

のように合成サイクルをモデル化する(図４参照)。なお、以下では、上位の合成階梯から下位の合成階梯への変換を“展開”と呼ぶ。

①解析：動作仕様や上位の合成階梯の展開結果を解析して、展開対象の構造を明確にする。さらに、この階梯で必要な設計制約を明らかにする。
②選択：設計制約と解析結果とをもとに適当な展開アルゴリズムを選択する。再試行時ならば、すでに選択されたことのあるアルゴリズムは選択の対象外とする。適当なアルゴリズムが存在しない場合は、上位の合成階梯に戻り、再試行を行う。なお、最上位の階梯で適当なアルゴリズムが存在しない場合は、①に戻って設計制約を修正する。
③展開：展開アルゴリズムが選択されたならば、それを適用して動作仕様または上位の合成階梯の結果を、この階梯で目的とするハードウェア構造に展開する。
④評価：展開結果が設計目的に叶っているか、また所与の基本素子で実現できるかなどを判断して、次の階梯に移るか否かを決める。不可ならば②選択に戻り、可ならば次の階梯に進む。

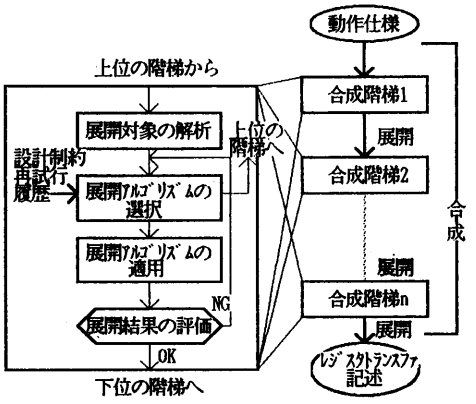


図4　段階的合成サイクルのモデル

３．２　設計プランの構成と選択法

(1)設計プランの構成

展開アルゴリズムとしては、標準的な展開手法と既存の設計事例を規則化したものを採り入れる。これに対して、設計手法や部品技術等の発展に伴い、新たな設計制約の考慮と、設計様式の取り入れが必要となる。既存の構成に影響なくそれらを可能とするため、展開アルゴリズムと、設計制約との適合度の評価式を対にして設計プランとし、次の手順で選択する。

(2)設計プランの選択手順

設計プランは、原則的には与えられた設計制約との適合度の高い順に選択されるが、選択の再試行を考慮して以下のように行う(図５参照)。

すなわち、①設計プラン側では、選択機構から与えられた設計制約をもとに、各自の評価式にもとづいて適合度を計算する。②選択機構側では、各設計プランから適合度をあつめる。③同一合成階梯の設計プランを選択した回数がＮならば、適合度の高い順から第Ｎ+1番目の設計プランの展開アルゴリズムを選択する。

このとき、各設計プランの適合度を次のように定式化する。

(3)適合度の評価式

１つの展開アルゴリズムで作られる回

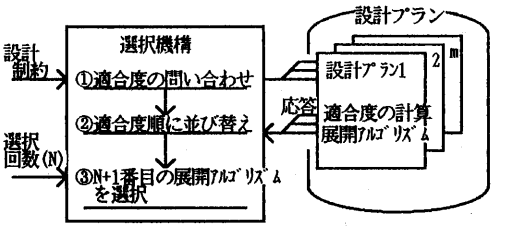


図5　設計プランの構成と選択

路構成が与えられた設計制約のうちのいくつの項目に適合するかを、そのアルゴリズムと設計制約との適合度と定義する。適合度の計算において、ある項目が特定のアルゴリズムを指定する場合と複数を指定する場合とでは、該当の項目の重要さが異なるため重み付けをおこなう。さらに、複数のアルゴリズムが共通の設計制約を満足する場合に、いずれかを優先的に選べるように、各アルゴリズムが満足する項目の重みの総和で該当の項目の重みを割って正規化した結果で比較する。以上を考慮して、次の(1)式を評価式として用いる。

$$M_j=(\Sigma a_i \times p_{ji})/\Sigma p_{ji} \qquad (1).$$

ここで、a_iは設計制約の項目iが与えられたか否かを表し、項目が与えられた場合は'1'、ない場合は'0'の値をとる。p_{ij}($0 \leqq p_{ij} \leqq 1$)は設計プランjを選ぶための設計制約の項目iの重要さの割合(重み)。

4 レジスタトランスファ合成

4.1 合成手順

動作仕様からレジスタトランスファ記述を合成するには、データパスの構造と制御系列を作り出す必要ある。データパスの構造はデータ転送に利用される時間に依存するので、制御系列の合成(転送動作の同期時刻を決めるので時間割付けと呼ぶ)の後、データパスの構造の合成(バス配置と呼ぶ)を行う。各合成階梯において、展開アルゴリズムの選択肢が複数ある場合、そこでさらに階梯を分けて次の手順で行う(図6参照)。

(1)時間割付け

転送動作の実行時刻は、各動作が幾つ並列に実行されるかに依存するため、N(N≧2)段並列から直列までの並列度に対応した時間割付けアルゴリズムを用意する。その参照関係から同時実行可能な動作同士を集めて1クロック周期で実行する動作とする。その結果に、用意された中の適当な時間割付けアルゴリズムを適用する。この手順の詳細は、次の4.2に述べる。

(2)バス配置[8),9)]

レジスタの合成、演算回路の合成、およびバス構造の合成に階梯を分ける。

レジスタの合成では、動作仕様の"資源の定義"で与えられた資源に対し、ライフタイム(対象の資源や操作が転送動作に使われている時間)を解析する。利用可能な記憶素子(ｾｯﾄ・ﾘｾｯﾄ型FF、ﾏｽﾀ・ｽﾚｰﾌﾞ型FF、ｽｸﾗｯﾁﾊﾟｯﾄﾞﾒﾓﾘ等)を設計制約として配置アルゴリズムを選び、1クロック周期以上に亙って情報を保持する資源はレジスタに、1クロック以内のものは端子とする。

演算回路の合成とバス構造の合成では演算機能および資源と演算回路の間の転送についてライフタイムを解析し、同一時刻で使われない演算機能ないし転送をクリーク分割アルゴリズム[2)]にもとずいて統合し、演算回路の機能とバス構造を決める。

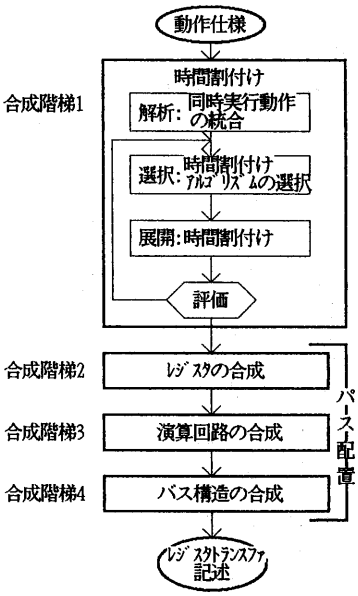


図6 レジスタトランスファ合成手順

4．2　時間割付け手順

(1)同時実行可能な動作の統合

記述に現れる関数(図２の記述例ではiexec(X)のような形式で表されているもの)をすべて組み込む。動作シーケンスの開始時点から終了時点まで(これを経路と呼ぶ)に対して、AEAP(As Early As Possible)法[2]によって次の範囲から同時実行可能な動作を集めた後、各同時実行動作を１クロック周期内の動作とする。

(ｱ)探索の上限は、動作シーケンスの開始時点、時間制限付き動作の先頭および分岐とする。

(ｲ)探索の下限は、動作シーケンスの終了時点、時間制限付き動作の先頭の直前および分岐の直前とする。

(2)時間割付けアルゴリズムの選択

動作の並列度を制約する要因には以下がある。

(ｱ)合成の狙いが、少ないゲート量か高速性か。

(ｲ)メモリインタリーブのように外部装置とのインタフェースに時間規定があり、それが部分的に重畳可能か否か(図７参照)。

(ｳ)使用できるクロックは単相か多相か。

(ｴ)対象の動作仕様の動作シーケンスが繰り返しの構造になっているか否か。

これらを設計制約として時間割付けアルゴリズムを選択する際に、再試行を考慮して以下のように行う。

アルゴリズムの詳細がパラメータで指定できる場合は、再試行時には先ずそのパラメータを更新したものを選択する。さらに、与えられた設計制約に適合しないものの選択を防止するために、アルゴリズムの排他関係を指定(マーク(＊)付け)しておく。そして、適合度の高い順に選択していく際、設計制約に対応する欄にこのマークのあるアルゴリズムは選択の対象外とする。

制御系列の合成アルゴリズムと設計制約との対応例を図８に示す。図２に示した仕様(動作定義の第一行にwhile (run=on)で指定された繰り返し構成がある)が与えられたとき、設計制約を“設計のねらいは性能優先で、時間規定のある動作同士の重畳が可能である”とすると、N段並列制御Ｉが選択される。

図８の例の設計制約に対して、性能重視のアルゴリズム(N段並列制御)では、動作をできるだけ並列に実行させる制御系列を作るので、合成結果を見て多相クロック化が有利か資源の増加が有利かを判断する。一方、コスト重視のアルゴリズム(直列制御)では、単相クロックを用いて動作が重ならない制御系列を作る。

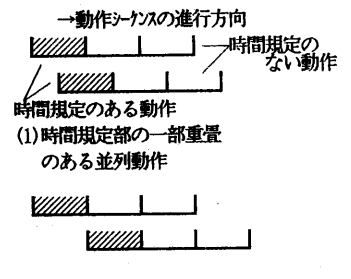


図7　時間規定部の重畳

時間割付けアルゴリズム / 設計制約の項目		直列制御	N段先行制御 Ⅰ	N段先行制御 Ⅱ	与えられた設計制約例(ai)
設計の狙い	ｺｽﾄ	1			0
	性能		0.5	0.5	1
時間規定の重畳	可		1		1
	非		＊	1	0
クロック	単相	1	＊	＊	0
	多相		0.5	0.5	0
繰り返し構成	有		0.5	0.5	1
	無	1			0
ΣPji		3	2.5	2.5	

注) ▨ 内の値はPji

図8　時間割付けアルゴリズムと設計制約の例

いずれも、極端な制御形列から始まるので、他の評価要因を満足したところで再試行を停止する。

(3)時間割付け

N段並列制御Iを採り上げて、再試行を考慮した手順を述べる(図9参照)。

「Step1」並列化対象外の経路(繰り返し構成でない経路等)を除き、各経路の最長のものに合わせて経路の長さを揃える。

「Step2」各経路に共通する動作のパタンを取り出し、先頭の動作開始時刻を順次後ろにずらした場合に、時間規定のある動作同士が重ならない位置を求める。このときに、以前の試行を繰り返さないように、この手順の試行回数がNならば、上からN番目の動作から位置の探索を開始する。Nが系列上のクロック周期数より大なら終了する。

「Step3」経路は、(ア)並列動作までの動作列、(イ)並列動作を含む動作列、(ウ)並列化対象外の動作列のいづれかに分類できるので、それらに対応したステージ(それぞれ(ア)初期ステージ、(イ)並列ステージ、(ウ)例外ステージと呼ぶ)を次の規則で作る。

(ア)初期ステージからは、動作シーケンスの繰り返し条件によって並列ステージに遷移する。

(イ)並列ステージのうち、先頭のステージでは、ステージ内の最終動作から、自ステージと次の並列ステージを起動する。先頭以外の並列ステージでは、次のステージだけを起動する。ただし、最終の並列ステージでは初期ステージを起動する。

(ウ)例外ステージは、この動作の直前の動作の属するステージから起動され、終了後は初期ステージを起動する。

5 合成結果

図2の動作仕様に対して、4章のN段並列制御Iのアルゴリズムを適用した結果を図10に示す。

(1)図10では、STAGE1,STAGE2,STAGE3が並列に実行される。この状態では資源に対するアクセス競合が生じるが、この

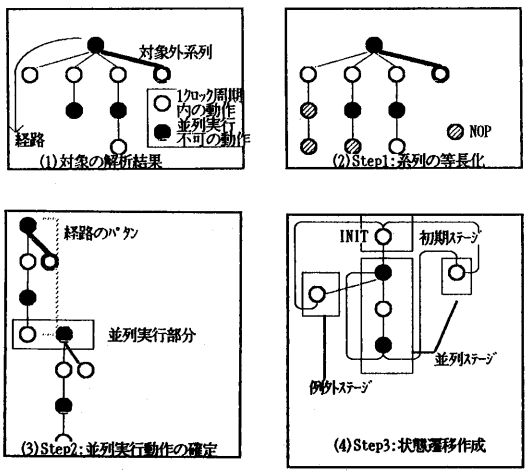


図9 制御構造生成手順

```
[INIT] (run/=on)
     =>[STAGE1];
[STAGE1] (run=on) or (f/='JL')
  <1>
   /clk/ ads:=1,adr:=scc,time_restriction;
      =><2>;
  <2> (dac=1) and restriction_end
   /clk/ ir:=dat;
      =>[EX],<1>,[STAGE2];
[STAGE2]
  <3>
   /clk/ if (f='L') a2:=reg[x2]+reg[b2]+d2;
         if (f='ST') a2:=reg[x2]+reg[b2]+d2,mem:=reg[r1];
         if (f='ADD') a2:=reg[x2]+reg[b2]+d2;
         if (f='DIAG' and r1=0) df:=1;
         if (f='DIAG' and r1=default) df:=0;
      =><4>;
  <4>
   /clk/ if (f='L') ads:=1,time_restriction,adr:=a2;
         if (f='ST') ads:=1,time_restriction,adr:=a2,dat:=mem;
         if (f='ADD') ads:=1,time_restriction,adr:=a2;
         if (f='DIAG') NOP;
      =>[STAGE3];
[STAGE3] (dac=1) and restriction_end
  <5>
   /clk/ if (f='L') reg[r1]:=dat;
         if (f='ST') NOP;
         if (f='ADD') reg[r1]:=reg[r1]+dat;
         if (f='DIAG') NOP;
      =><6>;
  <6>
   /clk/ scc:=scc+1;
      =>[INIT];
[EX] (f='JL')
   /clk/ scc:=reg[x2]+reg[b2]+d2;
      =>[INIT];
```

図10 時間割付け結果

後のバス配置によって、時間調整用のバッファが合成(レジスタ合成結果は付図1に示す)され、競合は回避される。

また、この時間割付けの段階では単相クロックで同期がとられているが、時間調整時に多相クロックを利用する[9]。

(2)本手法では、抽象度レベルの高い動作仕様からの合成を、試行錯誤的に行えることを狙っている。その際、ネックとなるのは処理時間であり、段階的に合成するのはその対策の1つである。さらに応答時間を短くするために、再試行時に以前の合成結果を活用する方法等が今後の課題として挙げられる。

6 むすび

抽象度の高い動作仕様から段階的に試行錯誤を繰り返して、レジスタトランスファ記述を合成する方法を述べ、簡単な仕様を例にこの方法の適用結果を示した。レジスタトランスファ記述から論理接続記述への展開は、多相クロック等の時間的な制約を扱える論理回路合成[10]によって行うことが考えられる。今後は、応答時間の短縮化の検討、合成アルゴリズムの充実を図って行く。

最後に、本検討に当たり、御指導を賜わった、NTT電気通信研究所 基幹交換研究部 処理方式研究室 浜田泰昭室長、脇村慶明主幹研究員および室員各位に深謝いたします。

[参考文献]

[1]T.J.Kowalski,D.E.Thomas:"The Design Automation Assistant:Prototype System",20th DA Conf.pp.479-483(1983)

[2]C.Tseng,D.P.Siewiorek:"Faset:A Procedure for the Automated Synthesis of Digital Systems",20 th DA Conf.,pp.490-496(1983)

[3]林、丸山、真野他:"論理型言語による論理設計支援システム"、信学技報、EC85-28(1985)

[4]高橋隆一:"ハードウェア・アーキテクチャコンパイルの一手法"、第32回情処全大 1U-6(1986)

[5]N.Park,A.Parker:"Synthesis of Clocking Schemes",22nd DA Conf.,pp.489-495(1985)

[6]小林、若林:"非手続き的記述をとりいれた図的アーキテクチャ設計言語";信学技報、EC84-48(1984)

[7]南谷崇:"大特集:論理装置CADの最近の動向-論理合成"、情処学会誌、25、10、pp.1071-1077(1984)

[8]小林、若林:"論理合成エキスパートシステムにおける推論の一手法"、昭和60°信学会部門別全大、46

[9]若林、小林:"レジスタトランスファ合成におけるバス割付けの一手法"、第32回情処全大 1U-8(1986)

[10]小林一夫:"図的言語を用いた装置機能設計支援法";情処論文誌,26,4,pp.643-651(1985)

```
[INIT](run/=on)
  =>[STAGE1];
[STAGE1](run=on) or (f'/='JL')
  <①>
    /clk1/ ads←bus1(1),adr←bus2(scc),time_restriction;
           =><②>;
  <②>(dac=1) and restriction_end
    /clk1/ ir←bus3(dat2);
    /clk3/ reg2[*]←bus2(reg1[*]),scc2←bus3(scc1);
    /clk4/ ir'←bus4(ir);
           =>[EX],<①>,[STAGE2];
[STAGE2]
  <③>
    /clk1/ if (f'='L') a2←bus5(op1([3+],[bus8(reg[x2']),
                          reg[bus10(b2')],bus9(d2')]);
           if (f'='ST') a2←bus5(op1([3+],[bus8(reg[x2']),
                          reg[bus10(b2')],bus9(d2')]),
             mem←bus4(reg[r1']);
           if (f'='ADD') a2←bus5(op1([3+],[bus8(reg[x2']),
                          reg[bus10(b2')],bus9(d2')]);
           if (f'='DIAG' and r1'=0) df←bus4(1);
           if (f'='DIAG' and r1'/=0) df←bus4(0);
           =><④>;
  <④>
    /clk1/ if (f'='L') ads←bus1(1),ads←bus2(a2),time_restri
           if (f'='ST') ads←bus1(1),ads←bus2(a2),
             dat←bus11(mem),time_restriction;
           if (f'='ADD') ads←bus1(1),ads←bus2(a2),time_rest
           if (f'='DIAG') NOP;
    /clk2/ reg3[*]←bus2((reg2[*]),scc3←bus6(scc2);
    /clk3/ ir'←bus1(ir');
    /clk4/ dat1←bus2(dat);
           =>[STAGE3];
[STAGE3] (dac=1) and restrictin_end
  <⑤>
    /clk1/ if (f''='L') reg3[bus6(ri")]←bus4(dat1);
           if (f''='ST') NOP;
           if (f''='ADD') reg4←bus7(reg3[bus6(r1")]);
           if (f''='DIAG') NOP;
    /clk2/ if (f''='L') NOP;
           if (f''='ST') NOP;
           if (f''='ADD') reg3[bus6(r1")]←bus8(op1([2+],
                          [bus4(reg*),bus7(dat1)]));
           if (f''='DIAG') NOP;
    /clk3/ reg[*]←bus2(reg3[*]);
           =><⑥>;
  <⑥>
    /clk1/ scc*←bus5(scc);
    /clk2/ scc←bus4(op1([inc],[bus3(scc*)]));
    /clk3/ dat2←bus4(dat1);
    /clk4/ scc1←bus9(scc),reg[*]←bus3(reg[*]);
           =>[INIT];
[EX] (f'='JL')
    /clk1/ scc2←bus1(op1[3+],[bus3(reg2[x2']),
                  reg2[bus2(b2')],bus9(d2')]);
    /clk2/ scc←bus1(scc2);
           =>[INIT];
```

付図1 レジスタトランスファ合成結果