設計自動化 55－5
(1990.12.12)

局所動作解析に基づく論理生成手法

黒田 理香子，三浦 地平，清水 嗣雄
㈱日立製作所

あらまし　パイプライン計算機の論理回路を自動生成するＡＳＴＲＯのアルゴリズムについて述べる。ＡＳＴＲＯはシーケンシャルなデータ転送に対する制御論理の生成方式と、リソース競合が発生する場合のデータ転送に対する制御論理の生成方式の２種の制御論理生成アルゴリズムを備えている。本報告では２種のアルゴリズムを用いて、パイプライン制御計算機の制御論理を自動生成する方法について示す。さらに、シーケンシャルなデータ転送に対する制御論理生成方式について詳細に述べる。本方式は局所動作解析を用いて、制御論理のフリップフロップ（ＦＦ）数を最小化するという特徴がある。ＣＭＯＳ－ＶＬＳＩの論理をベンチマークとして採用し実験を行った結果、局所動作解析の効果によりＦＦ数を約２０％削減した。

A Logic Synthesis Method Based on Local Behavior Analysis

Rikako Kuroda, Chihei Miura, Tsuguo Shimizu
Hitachi, Ltd.
Kokubunji, Tokyo 185, Japan

Abstract　ASTRO is a system for logic synthesis of pipeline processors. ASTRO has tow control logic synthesis methods. One is a method for sequential data transfers. The other is a method for data transfers that occures resource conflicts. This paper presents how to describe the logic specification of pipeline processor, and how to synthesize the control logic using above two methods. Furthermore a method for sequential data tansfer is presented in detail. The feature of this new method is the ability to minimize the number of control flip-flops. we call this minimizing process local behavior analysis. Adopting CMOS-VLSI logic as the benchmark, we evaluated loval behavior analysis and found that is decrease the number of control flip-flops by approximately 20% decrease.

1．緒言

大型計算機、ＶＬＳＩやマイクロプロセッサ等の設計において、その高性能化を図るためパイプライン制御方式の設計が広く採用されるようになってきた。しかし、その設計は最近まで熟練した設計者によって人手で行われていた。従ってこの設計工数を削減するための、論理生成アルゴリズムの重要性が高まっている。ＡＳＴＲＯ (Associated Structure Oriented Control Logic Synthesis Algorithm)[4]はＲＴレベルの論理記述を入力として、パイプライン制御計算機の論理を自動生成するシステムである。

パイプライン制御計算機に関する最近の提案はパイプラインステージの最適分割に関するもの等がある[1-3]。しかしながらパイプラインステージを制御するための論理を生成する方法については詳細化されていない。ＡＳＴＲＯでは、パイプライン制御計算機のデータ転送動作の性質を明確にし、その性質に応じた記述方法と制御論理生成アルゴリズムを構築することを目指した。

本報告ではＡＳＴＲＯの制御論理生成手法[5][6]の概要について述べ、局所動作解析による制御論理生成手法について詳細に述べる。さらに本手法の評価結果について述べる。局所動作解析は制御系の順序回路を最小化する方法である。現在、組み合わせ回路の論理を最適化するアルゴリズムについては優れたアルゴリズムが幾つか提案されている。しかし、順序回路については効果的な手法が無く、論理生成の段階で、冗長なＦＦを生成しないことが重要な問題となっている。この問題の解決策としてＡＳＴＲＯでは局所動作解析手法を開発した。

以降第２章では論理仕様記述言語について述べる。第３章ではＡＳＴＲＯの制御論理生成方式の概要について述べる。第４章ではシーケンス制御論理生成方式のアルゴリズムについて、第５章ではアルゴリズムの評価結果について述べる。

2．高位論理記述言語Ｂ²ＤＬ

本章ではパイプライン制御計算機のデータ転送動作を言語で記述する方法について述べる。記述には高位論理記述言語Ｂ²ＤＬを利用する。先ず、２．１節でパイプライン制御計算機のデータ転送動作の特徴について述べ、２．２節でそれぞれのデータ転送をＢ²ＤＬで記述する方法を示す。

2．1　パイプライン制御の特徴

パイプライン制御方式の計算機では、一連のデータ転送動作がいくつかのステージに分割されている。そして各ステージを並列に実行することで処理性能を向上している。従って、ＡＳＴＲＯではパイプライン制御計算機のデータ転送動作を次の２種に分類している。

（１）シーケンシャルなデータ転送

ある条件が成立した後に、複数のデータ転送が逐次的に実行されるようなデータ転送。パイプラインの流れには乱れが発生せず、スムーズにデータが流れる。パイプライン制御ではこのようなシーケンシャルなデータ転送を並列に実行している。

（２）リソース競合が発生するデータ転送

リソース競合とはパイプラインの流れに乱れが発生することである。具体的には、並列に転送されたデータがデータ待ち要因の発生点や、データの合流点で衝突する状況のことである。限られたデータ系リソースを用いてスループットを最大に向上させるためには、こうしたリソース競合が発生する場合が多い。リソース競合が発生した場合には競合状態が解消するまでデータを待たせる制御が必要となる。

2．2　高位論理記述言語Ｂ²ＤＬ

高位論理記述言語Ｂ²ＤＬ(Block-diagram and Behavior oriented Description Language)はパイプライン制御方式のデータ転送動作を記述するために開発された言語である。従って２．１節で述べたパイプライン制御のデータ転送動作が記述し易いように設計されている。

Ｂ²ＤＬには構造記述部と動作記述部の２つのパートがある。構造記述部はハードウエアのコンポネント、レジスタ、メモリ、入出力信号を宣言する。

動作記述部には構造記述部で宣言された論理要素に対する一連のデータ転送動作を記述する。２．１節で述べた２種のデータ転送動作の記述方法を以下に述べる。

（１）シーケンシャルなデータ転送

図１（ａ）はシーケンシャルなデータ転送の記述例を示したものである。この論理記述は図１（ｂ）に示したブロック図上のデータ

転送動作を記述したものである。

データ転送動作はシーケンス文によって記述される。シーケンスreadは入力ピンaddrから入力されるアドレスに従ってメモリmemのデータを読みだす論理動作を記述したものである。

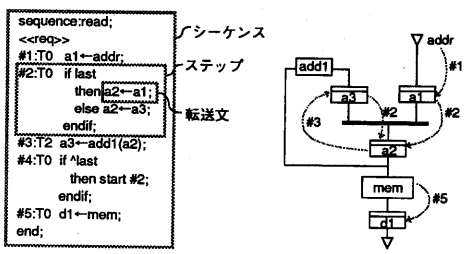
シーケンスはステップ＃１から＃５までの複数のステップによって構成されている。ステップはシーケンス起動条件reqが成立した後に記述された順番に実行される。一つのステップはタイミングとその時刻に起きるレジスタ間のデータ転送動作の集合で構成されている。例えばステップ＃２のタイミングはt0であり、その時刻に起きる動作はレジスタa1からa2への転送か、又はレジスタa3からレジスタa2への転送のどちらかである。どちらの転送が発生するかは条件lastによって異なる。またステップの実行順序を条件にしたがって変更したい場合は、ステップ＃４のような記述になる。条件^lastが成立すると＃２が再度実行される。

ステップはパイプラインのステージの概念に対応する。従って、B²ＤＬでは各ステップが並列に動作可能である。

（２）リソース競合が発生するデータ転送

図２（ａ）は図１（ｂ）のデータ構造上でリソース競合が発生する場合のデータ転送動作を記述したものである。この記述はメモリからのデータの読み出し中に新たなアドレスが、レジスタa1に転送された場合の動作を示している。シーケンスwaitは実行中のデータの読み出し処理が終わるまで、後続するデータを保持する処理を示す。このケースではa2に対するリソース競合が発生する。シーケンスwait、readの動作を図２（ｂ）のタイムチャートに示す。シーケンスwaitではbusy、hold等の特殊関数を利用することで、レジスタa1でのデータセット条件や、データ保持条件を表現している。

条件req&^last&^busy(a1)はリクエストがあり、先行するデータが処理中であり、レジスタa1が空いているという条件のもとでシーケンスが起動されることを示している。ステップ＃１は先行するデータの読み出し処理が終わるまで、a1でデータを保持すること、ステップ#2は先行するデータの処理が終わったら、シーケンスreadのステップ＃２に起動をかけることを示している。



(a)B²DL記述　　(b) ブロック図

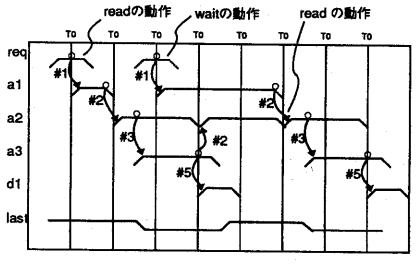
B²DL: Block-diagram and Behavior oriented Description Language

図１ B²DL 記述とブロック図

```
sequence:wait;
<<req&^last&^busy(a1)>>
#1:T0 a1 ← addr hold until(last)
#2:T0 if last then start read #2;
          else goto #2;
      endif;
end;
```

```
sequence:read;
<<req&last>>
#1:T0 a1←addr;
#2:T0 if last then a2←a1;
          else a2←a3;
      endif;
#3:T2 a3←add1(a2);
#4:T0 if ^last then start #2;
      endif;
#5:T0 d1←mem;
end;
```

(a)リソース競合が発生する場合のB²DL記述



(b) タイムチャート

図２　B²DL記述とタイムチャート

３．制御論理生成方式の概要

ＡＳＴＲＯの制御論理生成処理部は２．２節で述べたような２種のデータ転送記述に対応して、２種のアルゴリズムを備えている。一方はシーケンシャルなデータ転送動作の記述に基づいて論理を生成する、シーケンス制御論理生成方式である。もう一方はリソース競合は発生する場合のデータ転送動作の記述に基づいて制御論理を生成するリソース競合制御論理生成方式である。リソース競合制御論理はシーケンス制御論理の出力を用いて論理を構成する。以下に各方式の概要について述べる。

（１）シーケンス制御論理生成方式

シーケンス制御論理はステップの実行順序を管

理する順序回路である。本方式はステップの起動関係をグラフ表現し、それを順序回路に変換することでシーケンス制御論理を生成する。シーケンス制御論理生成方式には以下の特徴がある。

（ｉ）各ステップが並列に動作可能な制御論理を生成する。

（ii）局所解析に基づいて単一のシーケンスを制御する論理のフリップフロップの数を最小化する。

本方式については４章で詳細に述べる。

（２）リソース競合制御論理生成方式

busy,hold untilなどの特殊関数に対応するリソース競合制御論理は規則的な論理の組合せで生成することができる。そこで、生成には論理テンプレート型生成方法を採用している。論理テンプレートは論理回路中に頻出する論理から定型の構造を持った部分を抽出し、ライブラリに登録したものである。ライブラリに格納された論理テンプレートは特殊関数に対応して参照され、展開される。リソース競合制御論理生成にはbusy、hold until、hold forの３種の特殊関数があり、これらの論理の生成には５種の論理テンプレートが利用される。展開された論理テンプレートの入力はシーケンス制御論理の出力である。本生成方式の詳細は[5]で述べた。

リソース競合制御論理生成方式の特徴は以下の２点である。

（ｉ）リソース競合制御用の特殊関数に対応して論理を生成する。

（ii）論理テンプレート型論理生成方法である。

４．シーケンス制御論理生成方式

本章ではシーケンス制御論理生成方式について述べる。B*DLのステップはパイプラインのステージの概念に対応する。従って一つのシーケンス内で２つ以上のステップが同時に実行されることもある。本処理では各ステップが並列に実行できるようにＦＦの割当てを考慮する。

論理自動生成システムの制御論理生成処理方式の構成を図３に示す。主な処理は次の３種である。

１）動作グラフの作成
２）局所動作解析による動作グラフの分割
３）動作グラフの順序回路への変換

以下各処理について詳細に説明する。

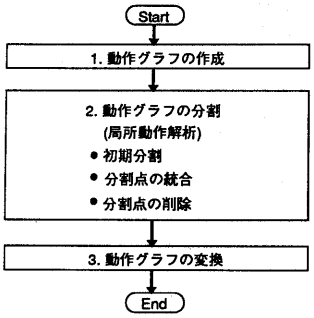


図３ シーケンス制御論理生成処理の構成

４．１　動作グラフの作成

本処理ではB*DLのシーケンス文から各ステップの起動順序を解析する。そして解析した結果は動作グラフとして表現する。図１（ａ）のB*DL記述から解析した動作グラフを図４に示す。図４のグラフの各ノードは一つのステップ、各有向辺は起動順序を示す。reqはシーケンスの起動条件をしめす。^lastはステップ＃４から＃２を起動する場合の条件を示す。このような条件を分岐条件と呼ぶ。この処理で作成した動作グラフを４．２、４．３節の処理によって順序回路に変換する。

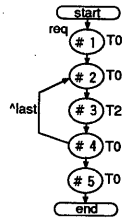


図４　動作グラフ

４．２　局所動作解析による動作グラフの分割

本処理は４．１で生成した動作グラフを分割し、分割点の数を削減する処理である。動作グラフの分割点は、生成される順序回路のＦＦに対応する。従って分割点の数はＦＦ数に対応する。分割処理は以下の３種のサブ処理によって構成される。

１）初期分割
２）分割点の統合
３）分割点の削除

２）３）の処理を局所動作解析と呼ぶ。次に各サブ処理の詳細について説明する。

1）初期分割

動作グラフに対し初期分割を行なう。動作グラフの有向辺の全てが処理の対象となる。有向辺のうち始点と終点にステップが存在する辺には初期分割点を割り当てる。これらの初期分割点に対してＦＦを割当てると各ステップは並列に実行可能となる。さらに各分割点に対してタイミングを与える。分割点のタイミングは最終的に分割点に対して割り当てたＦＦのクロックタイミングになる。分割点のタイミングは、有向辺の始点側のステップのタイミングとする。初期分割結果を図５に示す。この図は図１（ａ）の記述例に対応している。

2）分割点の統合

本処理では動作グラフ上の分割点を統合し、分割点の数を削減する。図５の動作グラフの分割点の全てについてＦＦを割り当てると冗長な順序回路となる、なぜなら幾つかのＦＦは一つに統合しても論理的には等価な順序回路が構成できる。分割点のうち同一のステップを終点とする有向辺上のものは同一のステップに対する起動条件をラッチするＦＦであるから、そのタイミングが同一であれば統合できる。本処理はこのような冗長な分割点を検出し統合する。従って、次の２種の条件を満たす分割点が統合される。

ｉ）分割点を通る有向辺の終点が同一のステップである

ii）分割点のタイミングが同一である

分割点の統合処理を行った結果を図６に示す。図５の分割点ａとｂは終点のステップが双方ともに＃４であり、さらに分割点のタイミングがＴ０と等しいので統合される。

3）分割点の削除

本処理は動作グラフ上の不要な分割点を削除する。例えば図６の動作グラフ上の分割点を順序回路に変換した場合、分割点Ｂ，Ａ，Ｆのそれぞれにｆｆが割り当てられる。それぞれの分割点のタイミングはＴ０，Ｔ２，Ｔ０である。しかも分割点ＡのＦＦは分割点ＢのＦＦ出力をラッチするだけのＦＦである。もし同相のタイミングでもレーシングを起こさないＦＦを使用している場合は、分割点Ａに対して割り当てたＦＦは冗長となる。本処理では分割点ＡのＦＦのような冗長なＦＦを削除する。

本処理では分割点の動作グラフ上の全ての分割点を処理対象とする。任意の分割点Ａについて、動作グラフを前後方向にトレースし次の要素を求める。

（１）前方向の分割点Ｆとそのタイミング。
（２）後方向の分割点Ｂとそのタイミング。
（３）分割点Ａを通る有向辺の分岐条件。

分割点Ａを通る有向辺の分岐条件が存在せず、分割点Ｆのタイミングと分割点Ｂのタイミングの間隔が信号の受渡しに十分である場合、分割点Ａは削除される。

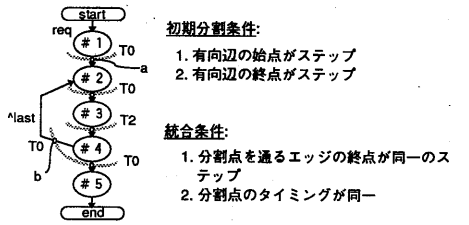


図５　初期分割と分割点の統合

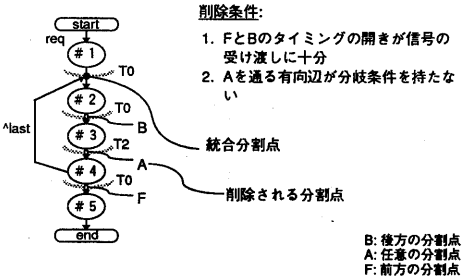


図６　分割点の削除

４．３　動作グラフの順序回路への変換

次に動作グラフの論理回路への変換を行う。動作グラフの変換は各有向辺とステップを論理式に変換する処理と、分割点をフリップフロップに置き換える処理によって構成される。有向辺とステップを論理式に置き換える方法は次のようになる。

有向辺の論理式＝
　（始点側ステップの'ステップの論理式'）
　・（分岐条件）
ステップの論理式＝
$\bigwedge_{i=1}^{m}$ステップを終点とする有向辺の論理式

ここで・は論理積、∧は全要素の論理和を示す。mはステップを終点とする有向辺の本数を示す。

動作グラフを組合せ論理に変換した後に、分割点をＦＦに置き換える。この時統合された分割点については次の組合せ論理と組合せ論理の出力をラッチするＦＦに置き換える。

統合された分割点の論理式＝

$$\bigwedge_{i=1}^{n} \text{分割点を通る'有向辺の論理式'}$$

ここでｎは分割点を通る有向辺の本数を示す。

以上の処理で動作グラフを順序回路に変換することができる。生成結果の論理回路は図７になる。ステップの論理式の出力はそのステップに書かれた転送先レジスタのセット信号やセレクタのセレクト信号として利用する。

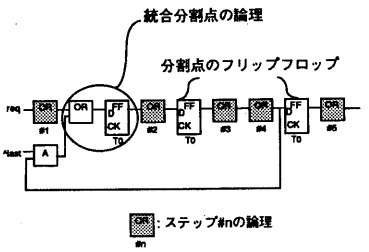


図7　動作グラフの変換

５．評価

ＣＭＯＳ－ＶＬＳＩの論理の一部、約5.6kゲート相当とＥＣＬの論理の一部約1.5kゲート相当を対象として局所動作解析の評価を行った。

B²ＤＬによるＣＭＯＳ論理の記述はread/writeとcancelの２ブロックである。

制御用ＦＦ数に対する局所動作解析の効果として約１８％のＦＦが削減された。詳細なデータは表１に示す。read/writeとcancelのデータ転送動作は１１種のシーケンス、cbusyは４種のシーケンスとして記述されている。ＣＭＯＳ論理の１１種のシーケンスに対する局所動作解析の効果はread/writeでは23％の効果があったが、cancelでは効果が現われなかった。なぜなら局所動作解析はステップ数の多いシーケンスが記述された場合に効果がある。従って短いシーケンスの多いcancelでは効果が認められなかった。

ＣＭＯＳ論理に対する論理生成結果を表２に示す。自動生成された論理を人手設計と比較した結果ゲート数は人手設計の12％増であった。パスディレイについても評価を行った。パスディレイは人手設計論理と自動設計論理からそれぞれ５０パスを抽出し、ディレイ増加率を測定した。ディレイ増加率は５０パスのうち遅延時間が最大となったものを、人手設計と自動設計からそれぞれ選択し、それらの比を採ったものである。ディレイ増加率は1.2倍となった。

６．結論

パイプライン制御計算機のデータ転送に関する制御論理の生成方式と論理生成方式の評価結果について述べた。本方式はシーケンシャルなデータ転送とリソース競合が発生する場合のデータ転送のそれぞれに対する制御論理生成アルゴリズムを備えている。シーケンシャルなデータ転送を制御する論理を生成する方式は、各ステップが並列に動作可能な論理を生成することと、局所動作解析により制御ＦＦ数を削減することに特徴がある。

ＣＭＯＳ－ＶＬＳＩの論理回路の一部を対象とした実験の結果、局所動作解析によりＦＦ数を１８％削減することができた。自動設計ゲート数は人手設計にくらべて１２％程度増加した。ディレイは２０％増となった。

７．参考文献

[1] R.Camposano et al., "Combined Synthesis of Control Logic and Data Path," Proc. ICCAD '87, Nov., 1984, pp.473-378.

[2] G.De.Micheli et al., "HERCULES-A System for High-Level Synthesis," 25th. DAC, June, 1988, pp.483-488.

[3] R.Camposano et al., "Synthesis using Path-based Scheduling: Algorithms and Exercises," 27th. DAC, June, 1990, pp. 450-455.

[4] T.Shimizu et al., "A Control Logic Synthesis and Optimization Algorithm with an Overlap Degree Vector," Proc. ICCAD '87,Nov., 1987, pp.124-129.

[5] 黒田、他"パイプライン制御計算機の制御論理生成方式"電子情報通信学会 VLD-88-113 1988 pp.17-22.

[6] R.Kuroda et al., "Control Logic Synthesis Method Based on Local Behavior Analysis" Proc. SASIMI'90, Oct., 1990, pp.82-87.

表1　局所動作解析の評価

(a) CMOS論理

ブロック名称	シーケンス名称	フリップフロップ数	
		局所動作解析なし	局所動作解析利用
Cancel	in	2	2
	out	1	1
	lock	1	1
	selctl	1	1
Read /Write	mode	3	2
	point	1	1
	in0	2	2
	in1	2	2
	out	7	4
	in2	1	1
	cnt	1	1
計		22	18

(b) ECL論理

ブロック名称	シーケンス名称	フリップフロップ数	
		局所動作解析なし	局所動作解析利用
Cbusy	start	0	0
	count	12	10
	read	7	6
	clear	3	2
計		22	18

表2　ASTROの評価

評価指標	人手設計 (a)	ASTROによる設計 (b)	比率 $\left(\frac{b}{a}\right)$
論理規模 (gates)	5640	6292	1.12
最大パス ディレイ (ns)	7.5	9	1.20

ベンチマーク: CMOS-VLSI 5.6 K gates