設計自動化28−3
(1985・9・17)

並列制御論理の合成

A SYNTHESIS OF CONCURRENT CONTROL LOGIC

大平駿介　伊藤　誠
SYUNSUKE OHHIRA　MAKOTO ITOH

山梨大学工学部
FACULTY OF ENGINEERING YAMANASHI UNIVERSITY

1．はじめに

本研究室では既にフローチャート形式の図面入力による、直列型の制御回路の自動合成システム［1］を開発している。このシステムにより、制御回路設計における図面入力方式の有効性が確認された。このとき残された課題の1つが、より高度で、高速な制御回路設計を可能にする、並列制御への拡張である。

並列で動作するシステムを図面の形で記述するものとして、ペトリネット［2］がある。本システムＣＯＬＤ(COncurrent control Logic Design system)は、このペトリネットを基にして考えたフロー図で制御の流れを図面入力することにより、並列制御回路（直列型を含む）を自動合成する。ＣＯＬＤは図面エディタ、制御論理式生成部、およびフロー図レベル・シミュレータの３つの部分で構成されている。

また本研究室では既に、ＣＯＬＤが自動生成する積和形式の制御論理式を簡単化し、回路を合成するプログラムＬＥＥＸＰ［1］を開発しており、回路合成にはこれを用いているが、ＬＩＳＰを用いたパターンマッチングによる多段論理式の簡単化プログラムＢＥＳ(Boolean Expression Simplification program)を試作したので、本稿ではこれを紹介する。ＢＥＳは人間が与えたルールに従い、論理式を簡単化するプログラムである。

ＬＥＥＸＰまたはＬＩＳＰによる論理式処理プログラムＢＥＳの出力した論理式は、会話型階層図面記述システムＣＡＴＳ［3］の論理式入力コマンドにより回路図面化することもできる。

本稿ではＣＯＬＤ、ＢＥＳ、ＣＡＴＳによる図面化の順に述べる。

2．システム構成

本研究室の論理設計部のシステム構成を図1に示す。

ＣＯＬＤ　：並列制御回路の設計システムであり、図面より制御論理式を自動合成する。

ＬＥＥＸＰ：論理式を簡単化し、簡単化後の論理式、およびＰＬＡパターン、または機能素子を自動合成するプログラムである。

ＢＥＳ　：データ駆動型の論理式処理プログラムである。

ＣＡＴＳ　：図面入力方式の会話型論理回路階層設計システムである。主にデータ処理部の記述に用いるが、ここではＬＥＥＸＰまたはＢＥＳの出力した論理式の回路図面化に使用する。

ＧＳＩＭ　：ゲートレベル・シミュレータである。

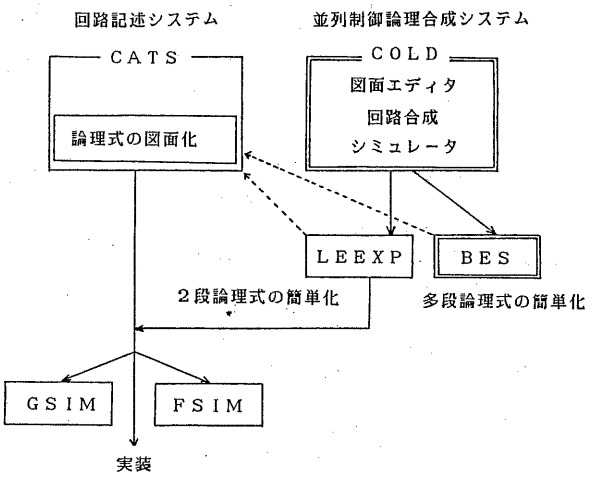


図1　論理設計システム

ＦＳＩＭ　：機能素子レベル・シミュレータである。ＣＯＬＤ、ＣＡＴＳにより設計された論理回路はＧＳＩＭまたはＦＳＩＭにより論理検証できる。

以下、部品割当、部品配置、ＰＣＢルータの各システムを通して、プリント基盤への実装が可能となっている。

３．並列制御図面の記述モデル

ペトリネット［２］はシステムを記述する一般的なモデルであるが、このままでは制御回路を設計するための図面としては使えない。そこでＣＯＬＤではペトリネットに基づき、順序回路間の同期、および並列実行を可能とする記述形式とした。すなわち、

i.順序回路に相当する直列制御部分を構成する状態（プレース）に同一の系列名をつける。

ii.異なる系列の状態はトランジションを介して、同期を取ることができる。

iii.トランジションより、異なる系列の複数の状態に同時に制御を移すことができる。

図２は基本的な記述例である。図形の形状は簡略化してある。図２のａ～ｈは以下の動作を記述している。

ａ．状態Ｓ１から同系列のＳ２へ無条件で遷移する。

ｂ．状態Ｓ１から同系列のＳ２へ条件Ｃで遷移する。

ｃ．状態Ｓ１から同系列のＳ２へ条件Ｃ１＆Ｃ２で遷移する。

ｄ．状態Ｓ１から同系列のＳ２へ条件Ｃ１／Ｃ２で遷移する。

ｅ．状態Ｓ１から条件Ｃ１で同系列のＳ２へ、条件Ｃ２で同系列の状態Ｓ３へ遷移する。ただしＣ１とＣ２は同時に成り立たないとする。

ｆ．同系列の状態Ｓ１、またはＳ２から同系列のＳ３へ遷移する。ただしＳ１とＳ２は同系列のため、同時に発生することはない。

ｇ．系列Ａの状態Ｓ１と系列Ｂの状態Ｓ２の同期を取り、系列ＡのＳ３と、系列ＢのＳ４へ遷移する。

ｈ．系列Ａの状態Ｓ１と系列Ｂの状態Ｓ２の同期を取り、Ｓ１から系列Ａの状態Ｓ３へ遷移する。

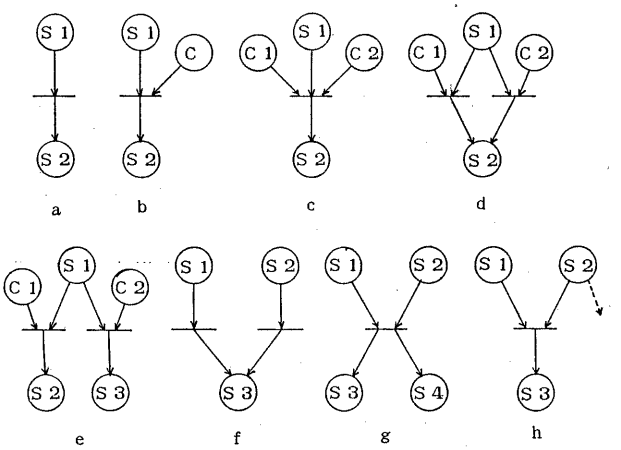


図２　基本記述例

４．記述形式

ＣＯＬＤでは上記のモデルを記述するために、図３に示すような図形を用いる。図３のａ～ｇの各図形は次のような意味を持っている。

ａ．状態プレース

制御の状態を表わす図形である。図形内部のテキストは以下の情報を持っている。

状態名　　：状態プレースの名前である。（ラベルを利用していなければ省略可能）

出力信号名　：その状態で値が１となる出力信号名。

動作系列名　：系列の名前で、同一系列内の２つ以上の状態に同時に制御

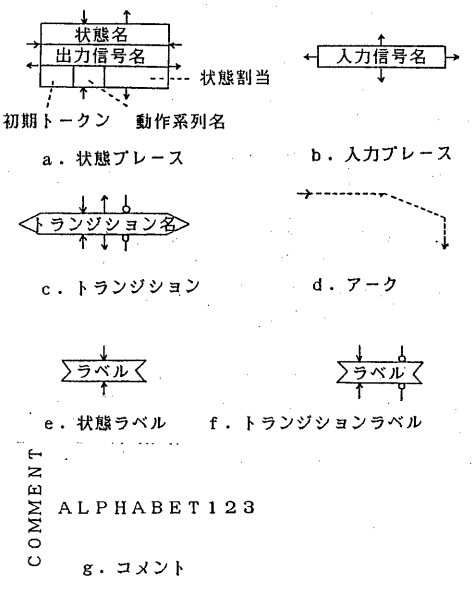


図３　ＣＯＬＤの図形

が移ることはない。

初期トークン：トークンとはペトリネットで、現状態を示すのに使われるマークである。フロー図の初期トークンは、リセット信号で設定される状態を示す。各系列ごとに１つだけ指定する。TOK は初期トークンが設定されていることを示し、/// は初期トークンがないことを示す。

状態割当　　：各系列ごとに設定されるＤ型フリップ・フロップの集合に対する、状態割当符号を示す。８ビットの２進表示で示され、未定義のものはUNDEFINEと表示される。

ｂ．入力プレース

制御回路に対する外部からの入力信号を表わす図形である。図形内には入力信号名を記述する。

ｃ．トランジション

状態遷移条件を表わす図形である。◇は否定条件を表わす。図形内にはトランジション名を記述する。（ラベルを利用していなければ省略可能）

ｄ．アーク

制御や信号の流れを表わす。

ｅ．状態ラベル

図形内のラベル名と同じ状態名を持つ状態プレースと等価な図形である。アークが複雑になって接続し難い場合に、目的の状態プレースの代用とする。

ｆ．トランジションラベル

機能は状態プレースと同じで、トランジションに対して用意されたものである。

ｇ．コメント

図面のドキュメンテーションを良くするために、図面の任意の位置に、任意のサイズで、ＸまたはＹ方向にＡＳＣＩＩ文字列を表示できる。またアークを単なる線として使い、コメントを囲んで表などの作成に用いることもできる。

各図形の矢印はアークを接続する端子で、アークの方向を示している。トランジションの否定端子には矢印がないが、これはトランジションへ入る向きである。各図形で、同方向（図形内に向いているか、図形外かという意味で）の矢印は、図面上でアークを接続し易いように設けたもので、そのいずれに接続しても同じである。ラベルについても同様である。

５．アークの条件とネット

図形間を接続しているアークは条件式とみなすことができる。図形の矢印を端点とし、互いに結線されたアークを「ネット」とすると、それぞれのネット上のどの位置も同じ条件式を満たしている。

状態プレースとトランジション（および、それらのラベル）では、入力方向の矢印の端子上での接触の意味が異なる。状態プレースではＯＲ条件となり、いずれかのネットが満たされればその状態へ遷移することを示す。トランジションではＡＮＤ条件となり、すべてのネットの論理積が遷移条件となる。図４はこれらの例を示している。ネットは矢印の端子上で分離されるので、端子上でアークが接触していてもそれらは別のネットとなる。従って図４のａ～ｄは、

ａ．ネットがトランジションへ入る
　ＡＮＤ条件：Ｃ１＆Ｃ２＆Ｃ３

ｂ．ネットが状態プレースへ入る
　ＯＲ条件　：Ｔ１／Ｔ２

ｃ．ネットがトランジションの端子上で接触する
　ＡＮＤ条件：（Ｃ１１＆Ｃ１２＆Ｃ１３）＆（Ｃ２１＆Ｃ２２）

ただし、それぞれのネット単位では、左のネットがＣ１１＆Ｃ１２＆Ｃ１３、右のネットがＣ２１＆Ｃ２２である。

ｄ．ネットが状態プレース端子上で接触する
　ＯＲ条件：（Ｔ１１／Ｔ１２）／（Ｔ２１／Ｔ２２）

ただし、それぞれのネット単位では、左のネットがＴ１１／Ｔ１２、右のネットがＴ２１／Ｔ２２である。

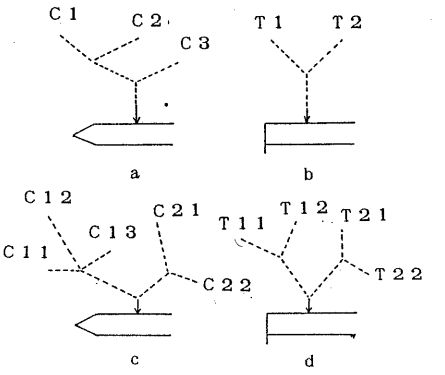


図４　ネットの条件

ＣＯＬＤのフロー図上では、状態プレースの自己ループを記述しなくても良い。ＣＯＬＤはある状態プレースの状態遷移条件を求め、それ以外の場合は自動的に自己ループであるとする。

６．ＣＯＬＤ

ＣＯＬＤは大きく分けて、図面エディタ、図面から制御論理式を自動合成する部分、フローレベル・シミュレータの３つで構成されている。

（１）図面エディタ

ＣＯＬＤの画面構成を図５に示す。マウスを用いた会話型のフロー図の図面エディタで、コマンドメニュー・メッセージ領域の指示に従って図面を編集する。メッセージは日本語表示であり、理解し易い。消去系のコマンドについてはＵＮＤＯ機能を持たせてあり、誤操作により消去してしまった図形や文字列などを元に戻すことが可能である。ほとんどの指示はマウスとそのスイッチで可能であるが、文字はキーボードからの入力となるので、ラインエディタを用いてコメントなどの長い文字列の入力、編集をし易いようになっている。コマンドには次のものがある。

［入力］：状態プレース、入力プレース、トランジション、アーク、状態ラベル、トランジションラベル、テキスト
［移動］：図形、テキスト　　［修正］：テキスト
［消去］：図形、テキスト
［描画］：画面、白黒ハードコピー、カラーハードコピー
［ウィンドゥ変更］　　［シミュレーション］
［ロード］：図面　　［セーブ］：図面、論理式
［ディレクトリ表示］　　［初期化］

（２）制御論理式合成

ＣＯＬＤは図面のエラー・チェックを行なったのち、４．で述べたアークの条件に従い、Ｄ型フリップ・フロップのデータ信号端子に対する論理式と、制御回路からの出力信号の論理式を図面より自動合成する。Ｄ型フリップ・フロップの入出力端子名は自動的に、”Ｄ”＋”動作系列名”＋”状態割当符号ビット番号”、”Ｑ”＋”動作系列名”＋”状態割当符号ビット番号”となり、Ｑの否定は’Ｑ．．となる。ＣＯＬＤは論理式と同時に、すべての未使用の状態割当符号の論理和を冗長項として出力する。これは論理式の簡単化に利用される。

論理式のフォーマットは、
＄ＥＸＰ（信号名，論理式，冗長項）
または冗長項がない場合、
＄ＥＸＰ（信号名，論理式）
である。

論理式には演算子として、＆：ＡＮＤ、／：ＯＲ、％：ＥＸＯＲ、＾：ＮＯＴが利用でき、（）：括弧を多重に含んでも良い。

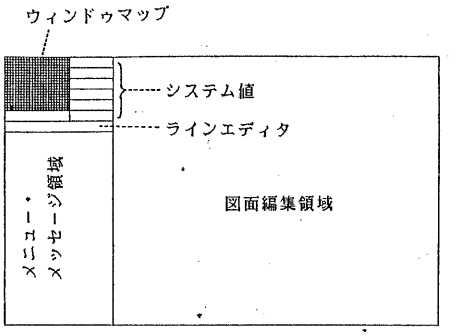
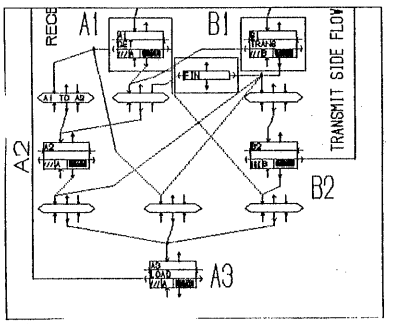
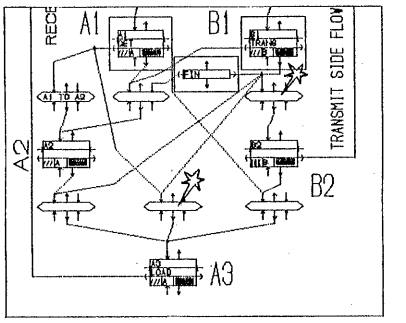


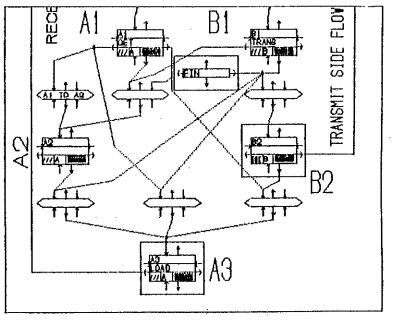
図５　ＣＯＬＤ画面構成



a



b



c

図６　シミュレーション

ＣＯＬＤは、並列制御回路全体を直列の動作系列に分割して合成する。１つの状態プレースに１つのフリップ・フロップを割当るわけではないので、状態割当のときに各直列動作系列の制御回路のフリップ・フロップ数を最小にするようにできる。

（３）フロー図レベル・シミュレータ

ＣＯＬＤでは、図面上でフロー図の動作をインタラクティブにシミュレーションできる。ただしこれはフロー図の正当性を検証するものであり、自動合成される回路の信号の遅延などの検証は、ゲートレベルまたは機能レベル・シミュレータで検証しなければならない。

シミュレーションを始めると、まず以下の「文法的」誤りが検出される。

ラベル名なし：状態ラベル、トランジションラベルの中に該当する名前のないものがある。
系列名なし　：状態プレースの中に、動作系列が記述されていないものがある。
状態割当なし：状態プレースの中に、状態割当されていないものがある。（シミュレーション時には必要ないので無視しても良い。）

次に、現在のアクティブな状態プレースと、入力プレースをマウスで指定するとトークン（図形を囲む枠）が表示される（または、既にトークンのあるものはそれが消える）。そしてシミュレータを起動すると状態遷移を起こす（発火）トランジションが示され、１クロック後の制御状態にトークンが移動する。ここで再び現状態のインタラクティブな指定モードになるので、変更する箇所を指定して、続けてシミュレーションを行なう。ユーザはトークンの移動を見ながらフロー図の動作を追うことにより、フロー図の記述ミスを発見できる。

また、次に示す「本質的な」エラーを検出できる。

トークンなし：ある系列の状態プレースの中で、現状態を示しているものがない。
トークン重複：ある系列の状態プレースの中で、複数のものが現状態を示している。

図６はシミュレーションの１シーンで、ａ．現状態－＞ｂ．発火するトランジションの表示－＞ｃ．次状態、を示している。

７．ＣＯＬＤのデータ構造

（１）図形テーブル

図形テーブルを図７に示す。ＣＯＬＤはＰＡＳＣＡＬで記述されており、図形テーブルは可変レコードを利用して、状態プレース、入力プレース、トランジション、状態ラベル、トランジションラベルのデータを記憶している。

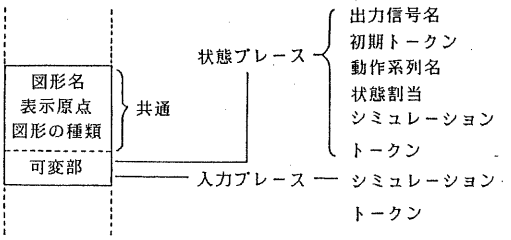


図７　図形テーブル

（２）コメントリスト

コメントリストを図８に示す。コメントは１文ずつ１つのセルに記憶され、それらは表示座標、表示方向、表示色、文字サイズ、文字列の情報を持った線形リストの形でリンクされている。

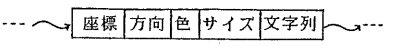


図８　コメントリスト

（３）アークリスト

アークリストを図９に示す。アークは各線分に分解されて記憶される。線分の端点のＹ座標値が等しいものは同一のＹヘダーに、Ｘ座標値で昇順にソートされる。また端点のセルには、線分のその端点の反対側の端点座標値がリンクされている。従って、線分は間接的な双方向リストとなっている。これはアークの条件を調べるときやシミュレーション時に、線分の探索が容易に行なえるようにするためである。

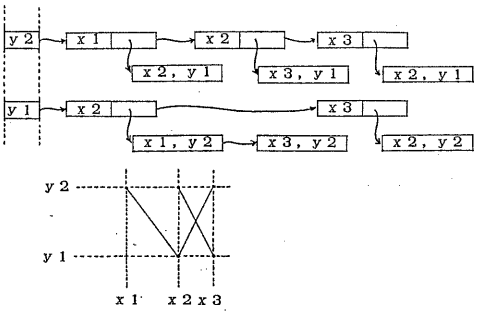


図９　アークリスト

８．ＣＯＬＤの例

図１０にａ．ＣＯＬＤによる設計例を示す。これは図１０ｂに示すような、パラレル・データを受信してシリアル・データとして送信する回路のコントローラである。フロー図で、左のＡ系列は受信側、右のＢ系列は送信側を制御する。図１１は自動合成された論理式、図１２はＬＥＥＸＰにより簡単化された論理式、図１３は合成された機能素子の一部である。この例についてはＢＥＳでも同様の結果を得ている。

図１０は次のような動作を記述している。

Ａ系列（受信側）

状態：動作

Ａ０：ＲＤＹ＝１として、外部からパラレル・データが送られる（ＲＥＣＶ＝１）のを待つ。（初期トークン）

Ａ１：ＧＥＴ＝１として、パラレル・データを受信用のレジスタへ一時的に保存する。

Ａ２：送信側制御回路が作動していたら、送信用のシフトレジスタがあくのを待つ。

Ａ３：ＬＯＡＤ＝１として、パラレル・データを受信用のレジスタから送信用のシフトレジスタへ移す。

Ｂ系列（送信側）

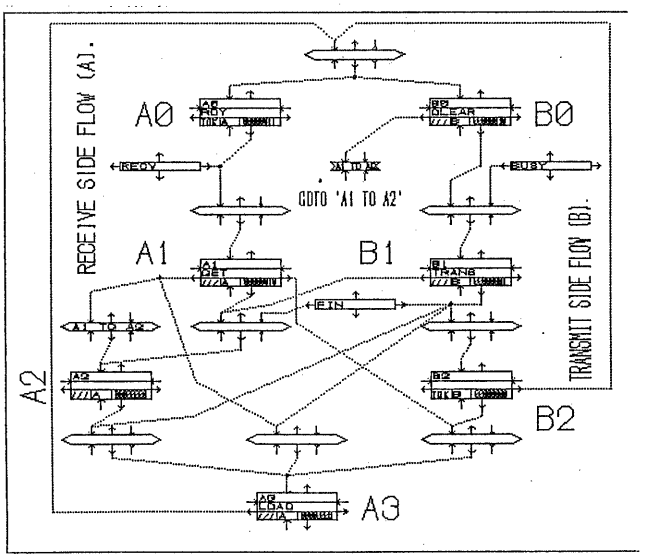
状態：動作

Ｂ０：外部回路がシリアル・データを受信可能になるまで（ＢＵＳＹ＝０）待つ。ＣＬＥＡＲ＝１として、送信ビット数を数えるカウンタをクリアする。

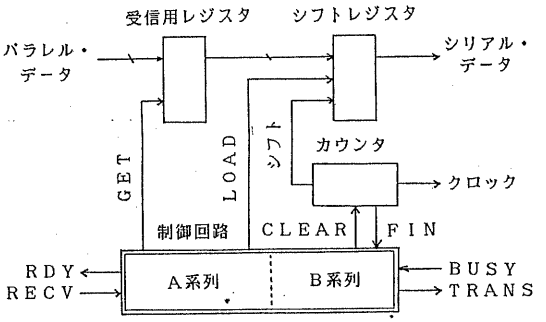
Ｂ１：ＴＲＡＮＳ＝１とし、カウンタを見て全ビットが送信される（ＦＩＮ＝１）のを待つ。

Ｂ２：受信用レジスタにデータがなければ、パラレル・データが受信されるのを待つ。（初期トークン）

Ａ系列とＢ系列はシフトレジスタを共有しているため、制御回路は両系列がこれを同時にアクセスしないように設計しなければならない。Ａ２という状態がこれにあたり、受信側は必ず送信が終わり、シフトレジスタがあくのを待ってからデータを送る。Ａ３の前のトランジションが複雑になっているのは、いかなる状況でもむだなクロックなしでＡ３に状態遷移できるようにしたためである。一般に送信の方に多くのクロックを必要とするが、その間でも受信側は動作できるため、



ａ．ＣＯＬＤによる設計例



ｂ．回路構成

図１０　並列制御回路の設計例

```
$EXP(DA1,^QB1&^QB0&^QA1&QA0/QA1&QA0&^(RECV)
        /QA1&QA0&RECV/QA1&^QA0&^(^QB1&^QB0
        /QB1&^QB0/^QB1&QB0&FIN/^QB1&QB0&^FIN)
        ,QB1&QB0)
$EXP(DA0,^QB1&^QB0&^QA1&QA0/QA1&QA0&^(RECV)
        /^QB1&QB0&FIN&^QA1&^QA0/QA1&^QA0
        &^QB1&QB0&FIN/QA1&^QA0&^QB1&^QB0
        /^QA1&QA0&^(^QB1&^QB0),QB1&QB0)
$EXP(DB1,^QB1&^QB0&^QA1&QA0/QB1&^QB0&^(^BUSY)
        ,QB1&QB0)
$EXP(DB0,QB1&^QB0&^BUSY/^QB1&QB0&^(FIN),QB1&QB0)
$EXP(RDY,QA1&QA0,QB1&QB0)
$EXP(CLEAR,QB1&^QB0,QB1&QB0)
$EXP(GET,QA1&^QA0,QB1&QB0)
$EXP(TRANS,^QB1&QB0,QB1&QB0)
$EXP(LOAD,^QA1&QA0,QB1&QB0)
```

図１１　合成された論理式

同じ周期のクロックであれば、受信と送信を順次行なう制御よりも速く動作する。

９．ＢＥＳ

ＬＥＥＸＰ［１］はＰＬＡに対応させるために、論理式を積和形式の２段回路に展開しているため、多段回路の簡単化をそのまま行なうことは不可能である。また簡単化の手法がプログラム組み込みであるため、簡単化の能力を上げるにはプログラム自体の変更が必要である。

ＢＥＳはＬＩＳＰとＰＲＯＬＯＧで記述されており、論理式に人間が与えたルールを適用して、局所的２段回路の簡単化を行なうプログラムである。ルールを外部から与えられるため、ＢＥＳの能力を上げることは容易である。

ＢＥＳはパターンマッチングによるルール適用部と、ルール検証部から構成されている。

```
$EXP(DA1,^QB1&^QB0&QA0/QA1&QA0)
$EXP(DA0,QA0&^RECV/QB0&^QA0&FIN
        /^QB1&^QB0&QA1&^QA0/^QA1&QA0)
$EXP(DB1,^QB1&^QB0&^QA1&QA0/QB1&BUSY)
$EXP(DB0,QB1&^BUSY/QB0&^FIN)
$EXP(RDY,QA1&QA0)
$EXP(CLEAR,QB1)
$EXP(GET,QA1&^QA0)
$EXP(TRANS,QB0)
$EXP(LOAD,^QA1&QA0)
```

図１２　ＬＥＥＸＰによる簡単化

```
.DFF((DA1,DA0),(@RST,@RST),(@T,@T)
    ,(@CLK,@CLK,@CLK),(QA1,QA0),('QA1,'QA0))
.DFF((DB1,DB0),(@T,@T),(@RST,@RST)
    ,(@CLK,@CLK,@CLK),(QB1,QB0),('QB1,'QB0))
.INV(BUSY,'BUSY)
.ANDS((QA1,QA0),RDY)
.ANDS((QA0,'RECV),@POA5)
.ANDS((GET,@POA1),@POA9)
.ORS((RDY,@POA6),DA1)
.ORS((LOAD,@POA5,@POA7,@POA9),DA0)
.ORS((@POA4,@POA8),DB1)
.ORS((@POA2,@POA3),DB0)
```

図１３　合成された機能素子（一部）

（１）パターンマッチング

論理式は内部で前置き記法で記憶される。例えば、A&B/C&^D&(E/F)は、(/ (& A B) (& C (^ D) (/ E F))) となる。従ってルールも同じ形をしている。信号線にマッチングする記号として、＃、（＊　＊　ｎ）、（＊　ｎ）がある。ここでｎは０以上の整数である。これらは以下の意味を持つ。

ａ．値が未定のとき（第１次マッチング）

＃　　：演算子の次の信号線とマッチングする。

（＊　＊　ｎ）：＃を除いた残りの１本の信号線とマッチングする。

（＊　ｎ）　：＃を除いた残りのすべての信号線とマッチングする。

ｂ．値が既に決まっているとき（第２次マッチング）

＃、（＊　＊　ｎ）、（＊　ｎ）のいずれも位置に関係なく、同一の信号線とマッチングする。

マッチングの例を次に示す。（＝＝でマッチングすることを示す。）

(& A B) と(& # (* * 0)) は、#==A,(* * 0)==B のようにマッチングする。

(/ A B C) と(/ # (* 0)) は、#==A,(* 0)==B C

(& A (/ B C)) と(& # (* * 0)) は、　#==A,(* * 0)==(/ B C)

(/ (^ A) (& B (^ A) C)) と(/ # (& # (* 0))) は、#==(^ A),(* 0)==B C で、このとき、＆内の＃は（^　Ａ）と第２次マッチングする。

マッチングでは最初に＃の値が決まらなければならない。なぜなら第１次マッチングでは、（＊　＊　ｎ）や（＊　ｎ）は「残りのもの」という定義だからである。

また＃のマッチングを入力された論理式のままで行なっていたのではマッチングしない場合がほとんどである。例えば、(& A B (/ B C)) と(& # (/ # (* * 0)) (* * 1)) は、#==Aとなり、(/ B C) の中にはA がないためマッチングしない。ところが式を(& B A (/ B C)) とすると、#==B,(* * 0)==C,(* * 1)==Aにマッチングするのである。

そこで、論理信号の可換性、および負論理表現との等価性を考慮して、以下の２点についてマッチング機能を高めた。

ｉ．第１次マッチングでは、信号がマッチングしない場合、演算子の次の信号（＃に相当する）を別のものに取り替えて再びマッチングを試みる。そしてすべての試みが失敗した時点で、マッチングしないと結論する。この信号線の取り替えを全レベルに対して行うことは、処理時間の上で現実的ではないので、対象の部分論理式の２段目までについて行なっている。（局所的２段回路の簡単化）

ｉｉ．論理の表現によっては等価なものも存在する。例えば、(& A (^ B)) と(^ (/ (^ A) B)) は負論理表現になっているだけで、その値は等しい。また、(& A B C) と(& C A B) は信号線の順番が違っているだけで、実際には等しい。ＢＥＳのマッチング関数は、負論理表現と信号線の順番の違

いを等価なものとしてマッチングさせる。これは２段に限らずに判断される。次の例はマッチングできる。
(& (/ A B) (^ (& (^ B) (^ A))) C) と(& # # (* * 0)) は、(^ (& (^ B) (^ A))) =(/ B A) = (/ A B) なので、　#==(/ A B),(* * 0)==C 。
ＢＥＳのマッチングは論理式の括弧の深いレベルから行なわれ、あるルールが成立すると変換された論理式に対して、再び初めからマッチングを行なう。そして、論理式のあらゆるレベルで、あらゆるルールがマッチングに失敗した時点で処理を終了する。

（２）ルール記述
ルール記述には上記の記号を用いた、プロダクション方式を採用している。プロダクションルールは、第１演算子により分類されて記憶する。これにより、全く違う演算子のルールとマッチングを試みるという無駄を省くことができる。個々のルールの形は次の通りである。
（マッチングさせるパターン　マッチングしたら変換するパターン）
変換するパターンの記号は、マッチングした値に置き換えられる。例えば、＆のルールに((# # (* * 0)) (& # (* * 0))) があれば、(& A A B) を(& A B) に変換することができる。ルールは第１演算子で分類されているので、マッチングさせるパターンの第１演算子は省略されている。各第１演算子ごとのルール群は次のようになっている。
（第１演算子　ルール１　ルール２．．ルールｎ）
パターンマッチングはルール１から順に試みられる。

（３）ルール検証
ルールの記述を人間だけに任せていたのでは、間違いがおこることがあり得る。例えば上の例で、((# # (* * 0)) (/ # (* * 0))) というルールがＢＥＳのデータに存在すると、(& A A B) は(/ A B) に変換されてしまう。このような誤りを防ぐために、ＢＥＳは人間が入力したルールのマッチングさせるパターンと、マッチングしたら変換するパターンが論理的に等価かどうかを検証する。この検証にはＬＩＳＰで記述されたＰＲＯＬＯＧ［４］を用い、マッチングさせるパターンと、マッチングしたら変換するパターンの排他的論理和を取り、その値を１（不一致）として、ＰＲＯＬＯＧのバックトラックにより、入力信号線のすべての組み合わせがこれを”満足しない”ことを調べている。これを満足するものがあった時点で、そのルールを誤りとしてエラー表示する。従って、ルールに論理的な誤りが入り込むことはない。
ルールはＢＥＳシステムとは別のファイルに保存されるため、論理段数の減少、信号線数の減少など、処理の目的に応じたルール群に分類していくつかのルール・ファイルにすることができる。ＢＥＳはこのルール・ファイルの切り換えコマンドを持ち、処理に適当なルール群でＢＥＳを動作させられる。

図１４にＢＥＳにより論理式が処理されて行く過程を示す。

ＢＥＳの問題点はルールの適用順序によって結果が変わる可能性があるということと、処理時間である。プロダクション・ルールによる”しらみつぶし”の方法のため、ルールの増加と論理式の複雑化に伴い処理時間が増加する。現在はＣＰ／Ｍ－８０上のｍｕ－ＬＩＳＰ上で動作させているためメモリに余裕がなく、処理時間の大半をガベージコレクションが占めているため実用的ではない。また、パターンマッチングの能力があまり高くないために、良く類似したルールをいくつも記憶させなければならず、現在の全ルール数が１５０を越えていることも、メモリ効率の悪さの１つの原因である。ただしこれは試験のためにルールを分類せずに記憶させた結果であり、処理の目的に応じてルールを分類して、いくつかのルール群のファイルに分ければ、ある程度減少すると思われる。

```
BES>^(^A&B/^B&A)%((C/D)&^D)

Expr:(% (^ (/ (& (^ A) B) (& (^ B) A))) (& (/ C D) (^ D)))

Befr:(/ (& B (^ A)) (& (^ B) A))
Rule:((& # (* * 0)) (^ (/ # (* * 0))))--->(^ (% # (* * 0)))
Aftr:(^ (% B (^ A)))

Befr:(^ (^ (% B (^ A))))
Rule:((^ #))--->#
Aftr:(% B (^ A))

Befr:(& (^ D) (/ C D))
Rule:((^ #) (/ # (* * 0)))--->(& (^ #) (* * 0))
Aftr:(& (^ D) C)

Befr:(% (& (^ D) C) (% B (^ A)))
Rule:(# (% (* 0)))--->(% # (* 0))
Aftr:(% (& (^ D) C) B (^ A))

Source expression is ^(^A&B/^B&A)%((C/D)&^D)
Final  expression is ^D&C%B%^A
```

図１４　ＢＥＳの処理過程

１０．ＣＡＴＳの論理式入力コマンド

ＣＡＴＳは本研究室で開発した、図面入力方式の会話型論理回路階層設計システムである。ＣＡＴＳは論理式入力コマンドを持ち、組み合わせ回路を論理式の形で入力して、ゲートレベル・シミュレータＧＳＩＭ用のゲートレベル素子を用いた回路図面に展開する。使用する素子は、ＡＮＤ２（２入力ＡＮＤ）～４、ＯＲ２～４、ＮＡＮＤ２～４、ＮＯＲ２～４、ＩＮＶ、ＥＸＯＲ２、ＥＸＮＯＲ２である。

論理式は初め、ＣＡＴＳ内部で図１５のように２分木構造に展開される。その後この２分木は以下のような処理を受ける。

ａ．多入力化

木上で連続した演算子を許される範囲内（ＡＮＤなら４入力まで）でまとめ、木を多分木構造とする。

ｂ．負論理化

負論理化すると、インバータが減少するときに行なう。

例（＆　（＾　Ａ）　Ｂ　（＾　Ｃ）
　　（＾　Ｄ））－＞
（＾　（／　Ａ　（＾　Ｂ）　Ｃ　Ｄ））

ｃ．直列インバータの削除

（＾　（＾　信号Ａ））の不要なインバータを削除し、信号Ａにする。

ｄ．インバータの吸収

ＮＡＮＤ、ＮＯＲ、ＥＸＮＯＲを利用する。

例（＾　（＆　．．．））－＞（＾＆．．．）にする。

処理は、ａ－＞ｂ－＞ｃ－＞ａ－＞ｄの順に行なわれる。図１５の木は図１６のようになる。

木が完成するとＣＡＴＳは木構造の演算子（素子）に座標を割当て、それを仮表示する。ここでインタラクティブな素子の配置修正モードになるので、配置改善をした後、自動配線を行ない図面化する。

図１７は、ＣＯＬＤで設計した図１０の制御回路を論理式入力コマンドを用いて図面化したものである。図１７の回路図面上で、上の２つのＦ．ＦがＡ系列、下の２つのＦ．ＦがＢ系列の順序回路のものである。

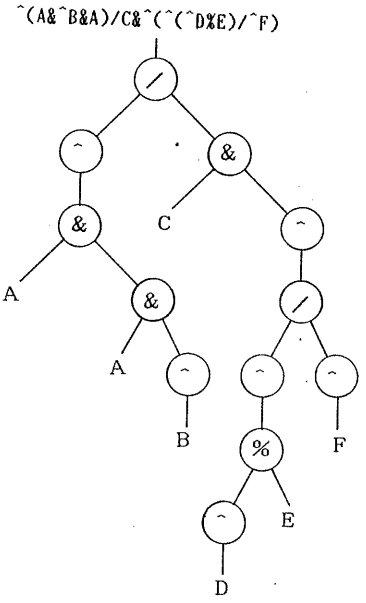


図１５　２分木への展開

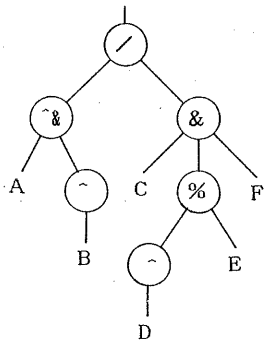


図１６　最終的な木

１１．おわりに

ＣＯＬＤ、ＬＥＥＸＰ、ＣＡＴＳはＣＰ／Ｍ－８６上でＰＡＳＣＡＬ　ＭＴ＋８６で記述されている。ＢＥＳはＣＰ／Ｍ－８０上でｍｕ－ＬＩＳＰで記述されている。

ＣＯＬＤにより、本研究室では図面入力による並列制御論理回路の設計を行なえるようになった。図面入力の利点は、記述の直感的な理解のし易さにある。特にＣＯＬＤのフローレベル・シミュレータは、フロー図の誤りを設計の時点（図面入力時）で発見できるため、素子化した後に素子レベルで検証を行なうよりも修正が容易である。

ＣＯＬＤの今後の課題は、自動状態割当と動作系列の自動割当である。また、ＢＥＳはより大きなＬＩＳＰ上に移植して試験、改良をしていく予定である。

<参考文献>

［１］伊藤誠、大平駿介：「制御フロー図面の入力と論理合成」、設計自動化研究会資料１９８４年６月

［２］J.L.Peterson：「ペトリネット入門」、共立出版

［３］伊藤誠、大平駿介：「パソコンを用いた、制御フロー図の入力と制御論理の自動合成」、情報処理学会全国大会１９８４年後期

［４］中島秀之：「Ｐｒｏｌｏｇ」、産業図書・コンピュータ・サイエンス・ライブラリ

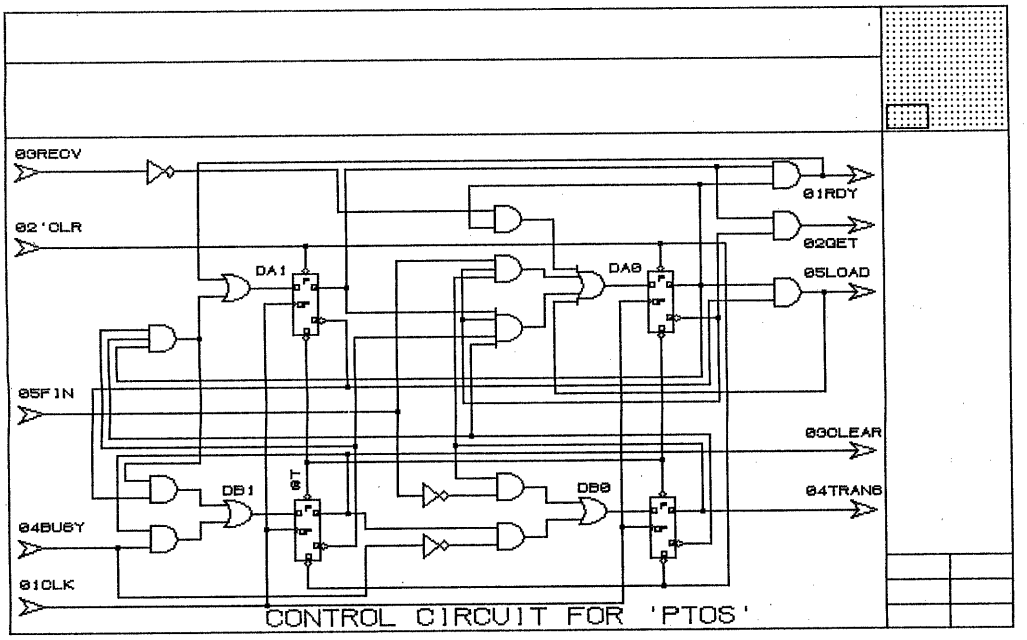


図１７　図面化例