TM-1129

協調型論理設計エキスパートシステム co-LODEX

澤田　秀穂、滝沢　ユカ、笈田　依子、
丸山　文宏（富士通）

October, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

協調型論理設計エキスパートシステム co-LODEX

co-LODEX: Cooperative Logic Design Expert System on a Multiprocessor

澤田秀穂　滝沢ユカ　箕田依子　丸山文宏

Shuho SAWADA　Yuka TAKIZAWA　Yoriko MINODA　Fumihiro MARUYAMA

富士通株式会社

Fujitsu Limited

1．はじめに

半導体技術の進歩により、LSI上に実現される機能は、大規模化、複雑化の一途をたどっている。そこで、高品質な製品を短期間で設計できる設計支援システムの実現が望まれている。

高品質な回路を設計するためには、回路の構成要素である、レジスタ、ALU、マルチプレクサなどの設計（部品に落ちるまでの詳細化）の知識の充実が挙げられる。複数の詳細化の方法により、異なる回路規模、遅延時間などの特性を持つ回路を設計できる。ところが、設計過程では、詳細化の方法の選択は、一意的には決まらない。このために、論理設計では、潜在的な解空間が組合せ的に大きくなるという問題がある。

一方、設計支援システムでは、短い設計のターンアラウンドを活用して、設計者が色々な構成、特性の回路を比較・検討できることが重要である。また、回路全体を対象とした最適化も重要な機能である。

我々は、ハードウェアの機能レベルの仕様から、回路規模と遅延時間に関する制約を満たす回路を設計する、論理設計エキスパートシステムco-LODEX（cooperative Logic Design EXpert system）を開発している。まず、与えられた制約条件に対して設計結果を評価し、違反が検出されると、制約条件を満たすよう設計のやり直し（再設計）を行なう方式の研究を行なった。この研究において、論理式で表現する制約条件違反情報（NJ:Nogood Justification）を考案し、これを利用して再設計を実行するシステムを試作し、NJが探索空間の絞り込みに効果があることを確認した［1］。NJは、最適化に対しても効果的である［6］。現在は、並列協調方式を用いて、このシステムを並列処理システムに拡張している［2］［3］。その特徴は、エージェントと呼ぶ自律的に動作する実行単位を設けて、部分回路を並列に、全体に対する制約条件を満足するように詳細化することである。

本稿では、まず、co-LODEXの概要として設計の流れとNJを利用した再設計機構を紹介し、並列処理システムに拡張するための並列協調方式について説明する。次に、Multi-PSI上にKL1言語を用いて実装した試作システムについて述べ、最適化と並列処理の実験結果を報告する。

2．co-LODEXの概要

2．1　論理設計の流れ

co-LODEXにおける設計の流れを図1に示す。入力は、機能レベルの動作仕様（図2）と、レジスタ、演算器などの機能ブロックと機能ブロック間のデータの流れを示すブロック図（図3）、生成した回路が満たすべき制約条件（ゲート数と遅延時間の上限値）の3種類である。動作仕様は、状態遷移表現に基づくレジスタ転送レベルの仕様記述言語で記述したものであり、これだけでも仕様として十分であるが、設計者が具体的な機能ブロックをイメージしている場合は、ブロック図を入力することにより、設計者の意図を反映させることができる。

初めに、動作仕様から機能レベルのオペレーションを抽出し、これらのすべてがブロック図で表されたデータパス上で実行可能なことを検証する。この過程において、機能ブロックとその接続関係、および制御回路の仕様を決定する。これと平行して、制約条件を、

デフォルトNJと呼ぶ、制約条件と等価な不等式に書き換える。

次に、機能ブロック（制御回路も制御ブロックと呼ぶ機能ブロックのひとつである）とその接続情報に書き換えられた仕様を分割してエージェントに割り当てる。すなわち、担当する機能ブロックの仕様に対応する部分仕様を生成する。

仕様の分割が終了した後、部分仕様とデフォルトNJを各エージェントに通知する。エージェントは協調して、部分仕様の詳細化と評価を繰り返すことにより、制約条件を満たす回路を設計する。制約条件違反を起こした構成方法にはNJを記録しておく。出力は、CMOSスタンダードセルの接続情報である。もし、制約条件を満足する回路を生成することができなかった場合は、違反した制約条件を提示する。ユーザが制約条件を変更することによって、再設計を開始することができる。制約条件の変更による再設計においては、蓄積された設計結果（機能ブロックの属性値）とNJを再利用する。

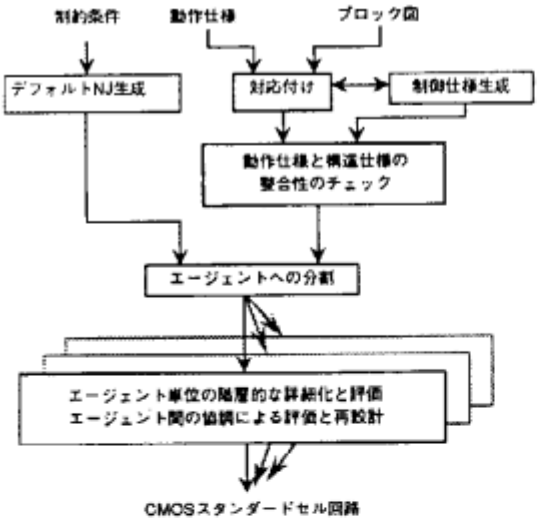

図1　設計の流れ

```
UHDL;
interface_view: interface_example01;
 inputs: .xi(12), .yi(12), .dxi(12), .ui(12), .ai(12);
 outputs: .xo(12), .yo(12);

behavior_view: behavior_example01;
 define: const5 = 5, const3 = 3;
 terminal: u1(12), u2(12), u3(12), u4(12), u5(12), u6(12), y1(12), FF;
 operator: 2stage_pipelined_multiplier(x, y, z) = ( len = 2 ),
          z <- x * y; end_op;
 function: main: clk;
 while (FF) do
  2a: '2stage_pipelined_multiplier'(u, dx, u1);
  3a: '2stage_pipelined_multiplier'(x, const5, u2);
  4a: '2stage_pipelined_multiplier'(const3, y, u3);
  5a: '2stage_pipelined_multiplier'(u2, u1, u4),
      x <- x + dx;
  6a: '2stage_pipelined_multiplier'(u, dx, y1),
      FF <- x < a;
  7a: '2stage_pipelined_multiplier'(u3, dx, u5),
      u6 <- u - u4;
  8a: y <- y1 + y;
  9a: u <- u6 - u5, xo := x, yo := y;
 enddo;
 1a: stop(x<a), x <- xi, y <- yi, dx <- dxi, u <- ui, a <- ai;
endUHDL.
```

図2　動作記述の例

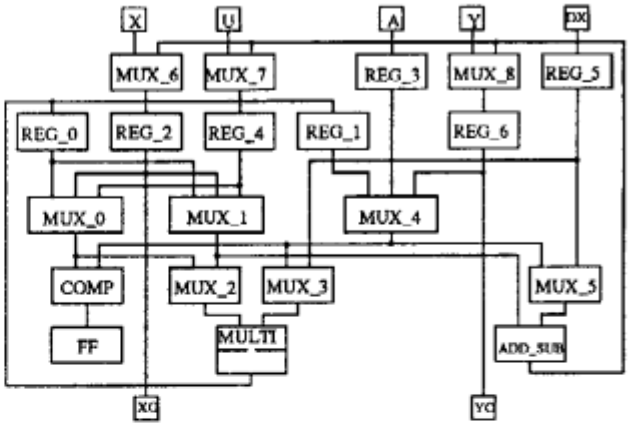

図3　ブロック図の例

2．2　階層的な評価と再設計

詳細化と評価・再設計は、設計の階層と同じ階層構造上に実現する。このとき、NJを階層上のノードに記録することによって、評価・再設計を制御する。

デフォルトNJは、制約条件に関係する属性値を示す変数の和を左辺、制約値を右辺とする、制約条件と等価な不等式である。この不等式が成立することが制約条件違反の必要十分条件になる。デフォルトNJを成立させる構成方法には、その構成方法の属性値でデフォルトNJの対応する変数を具体化したNJを記録することで、組み合わせを禁止する。設計中に生成されるNJは、デフォルトNJを起源とし、その成立が制約条件違反の十分条件になる論理式である。

評価と再設計の制御は、「NJの展開」と「NJの合成」と呼ぶ機構に基づく。例を用いて評価・再設計を説明する。

1. 3つの機能ブロックA、B、Cからなる回路を評価したところ、デフォルトNJ(a)が成立した。

   Aの面積＋Bの面積＋Cの面積＞面積制約値…(a)

2. 成立したNJ(a)の左辺の変数から、Aを再設計要素として選ぶ。
3. 再設計要素Aを構成要素A1とA2で置き換える。この操作を「NJの展開」と呼ぶ。

   （以下“の面積”は省略）

   A1＋A2＋B＋C＞面積制約値…(b)

4. NJ(b)の左辺の変数から、A1を再設計要素として選んだところセルで実現されているため、NJ(b)のA1を属性値（ゲート数）で置き換えたNJ(c)を構成方法に記録する。

   80＋A2＋B＋C＞面積制約値…(c)

5. A1の他の構成方法（90ゲート）でも違反するので、NJ(d)を違反した構成方法に記録する。

   90＋A2＋B＋C＞面積制約値…(d)

6. A1の構成方法を変えても制約条件を満足できない。すべての構成方法で違反したとき、構成方法に記録されたNJの論理積を機能ブロックに記録する。この操作を「NJの合成」と呼ぶ。NJ(c)と同一のNJ(e)をA1に記録する。
7. NJ(e)について1.以降と同様の手順で再設計を行なう。

NJを成立させなくなった時点で、途中すべての構成方法を禁止した機能ブロックからNJを成立させないような構成方法を選ぶ作業を開始する。すべての機能ブロックでNJを成立させてしまうときは、すべての変数が数字で具体化されたNJが生成されているはずである。この値は、設計可能な最小値である。

3．並列協調方式

並列協調アルゴリズムは、以下の二つの要件を満たす並列協調の定義から出発した。

(1) 与えられたゴールをサブゴールに分割し、サブゴールを複数のプロセッサで並列に達成する。

(2) 全体のゴールが達成されないとき（例えば、あるサブゴールが達成できないために全体のゴールが達成できない場合）、各プロセッサにおいて自発的に軌道修正を行なう。

ゴールGをサブゴールg_1, …g_nに分解するには、計算量f(G)に関して、$f(G) \gg f(g_i)$となる分解を選ぶ。つまり、規模の縮小による高速化を図る。一般的にゴールGを満たすためには、サブゴールの調整・再試行の繰り返しが必要である。サブゴールを一括して管理・調整する方式は、管理部に負荷がかかりネックになることが考えられるから、得策でない。従って、協調における各エージェントの自律的な軌道修正が必要であると考える。

この並列協調を論理設計システムに適用すると、次のようになる。

(1) ゴールは与えられた制約条件を満足する回路を設計すること。設計すべき仕様を分割し、各部分回路を並列に設計する。部分回路の設計モジュールをエージェントと呼ぶ。

(2) 全体の制約条件が満足されていないとき、各エージェントが設定した制約条件を変更して再設計する。

回路面積と遅延時間に関して与えられた制約条件は、部分回路に分割して設定することはできない。そこで、部分回路の設計結果（属性値）とNJを交換することにより、他のエージェントの設計値、または、設計不可能な範囲を考慮することで、自身の設計範囲を制限する。すなわち、自身の設計範囲を制限することを、エージェントの制約条件の変更とする。

3．1　並列協調アルゴリズム

図4に沿って説明する。並列協調アルゴリズムは、エージェントに部分仕様とデフォルトNJが通知されたとき始まる。以下の処理は、各エージェントが独立に（並列に）行なう。

【1】蓄積しているNJ（初めはデフォルトNJだけ）を成り立たせないように、部分回路を設計する。他のエージェントが担当している部分の属性値は0として扱う。

【2】少なくともひとつのエージェントが設計に失敗すると、制約を満たす回路は存在しない。すべてが成功したことを確認する。

【3】各エージェントの設計結果（部分回路の属性値）を通知し合う。

【4】ひとつのエージェントでも前回（【4】）と異なる設計結果を出していることを確認する。

【5】通知された設計結果を代入しても、デフォルトNJが成立しなければエージェントは成功。失敗したエージェントがないことを確認する。

【6】蓄積しているNJを成り立たせないように、部分回路を設計する。他のエージェントが担当している部分は【3】で通知された値を使用する。

【7】少なくともひとつのエージェントがNJを成立させない結果を得られたならば、成功した結果で置き換えて、【3】へ。これより、違反が生じている制約の数は減少する。失敗したエージェントには、担当する部分回路に相当する変数を含まないNJが生成されている。

【8】NJを関係するエージェントに通知する。通知されたNJは、「NJの組み合わせ」を行ない、新しいNJとして追加する。

【9】違反している制約のうち、関係するエージェントが最少のものをひとつ選ぶ。この制約を厳しくした一時的制約を与えることにより、強制的に異なる設計結果を導きだす。

【10】部分回路を結合する。

【11】現在の制約条件では設計不可能である。制約条件を緩和することにより、蓄積されたNJのいくつかが成立しなくなり、再設計を開始できる。

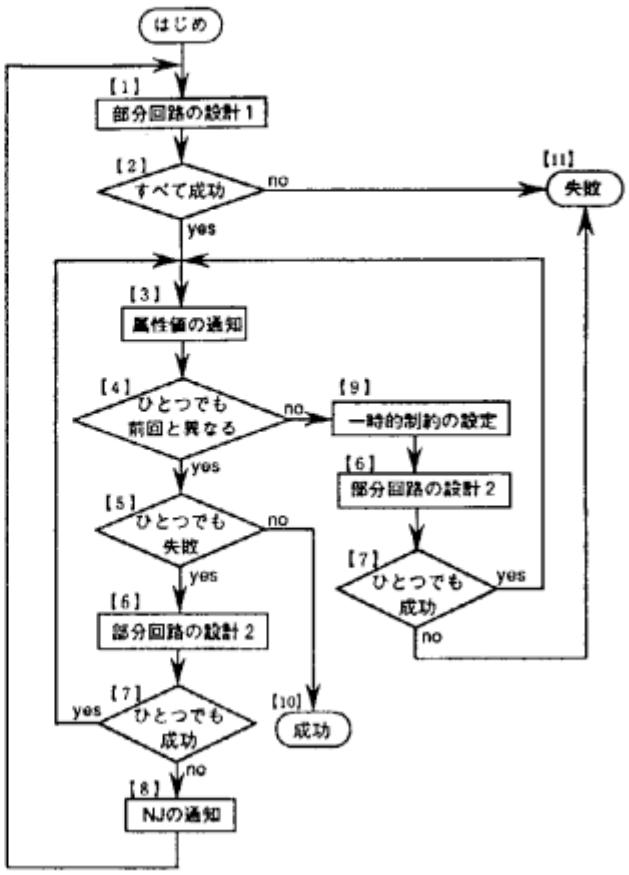

図4　並列協調アルゴリズム

エージェント間の同期は、属性値の通知とNJの通知によって実現されている。例えば、【5】では、自分自身（エージェント）が成功したとき、他の（少なくてもひとつの）エージェントが失敗するか、あるいは、他のすべてのエージェントが成功するまで処理を待つ。

3．2　NJの組み合わせ

NJの組み合わせは、他のエージェントの設計不可能範囲から、自分自身の設計範囲を制限するNJを生成する処理である。組み合わせたNJを追加することにより、他のエージェントが担当している部分を0として扱う部分回路の設計（図4の【1】）でも、他のエージェントの設計不可能範囲を考慮できるようになる。

組み合わせの例を示す。3つのエージェントに関係する制約があり、すべてのエージェントで再設計不可能であるとする。このとき、次のNJが生成されている。A1、A2、A3はそれぞれエージェントの属性値に対応する変数である。

エージェント2：A1＋10＋A3＞制約値

エージェント3：A1＋A2＋8＞制約値

エージェント1は、これらのNJからA1＋18＞制約値というNJを生成する。このNJからA1を除く部分に最低でも18必要なことがわかる。

4．試作システム

試作システムは、co-LODEXのエージェントの並列協調方式と、エージェントが独立に（並列に）行なう設計—評価—再設計の過程を実現したものである。入力は、分割されるべき回路の全体仕様と、エージェントへの割り当て、制約条件と等価なデフォルトNJである。本システムは、並列論理型言語KL1［4］で記述し、Multi-PSI［5］上に実装した。Multi-PSIはMIMD型マシンで、要素プロセッサ（以下、PE）が最大で64台、2次元メッシュ状のネットワークで結合されている。

Multi-PSI上のKL1言語の特徴を考慮して、設計の階層構造に現われる各要素と、NJの合成・組み合わせなどのデータ管理モジュールをプロセスとして実現した。各プロセスはメッセージ通信をしながら処理を進める。メッセージの受信により処理が励起される永久プロセスの表現を使うと、設計の上限値が通知されると評価・再設計を開始する機能ブロックなどが自然に表現できる。PE間のメッセージ通信のコストは、PE内の処理に比べて非常に高い。そこで、エージェントにおいては他のエージェントの詳細な情報を参照しないこと、各機能ブロックには（NJをそのまま与えるのではなく）制約値と他のエージェントの属性値から算出した上限値を与えることにする。並列に実行できる過程は、エージェントにおける部分回路の設計、その中でも各機能ブロックの設計と複数の経路の遅延時間の計算である。逐次的になる過程は、エージェントとコンポーネント設計のオルタナティブにおける上限値の算出と、NJまたは上限値に対する下位の機能ブロックの属性値の評価である。

設計の階層を構成するプロセスの概念図を図5に示す。楕円はプロセスを表し、矢印は入力／出力ストリームを表す。図5は、設計が進行したある状況を表すものであり、設計の初期状態では、入出力のプロセスだけが存在する。入出力以外のプロセスは、設計の進行に伴って生成する。

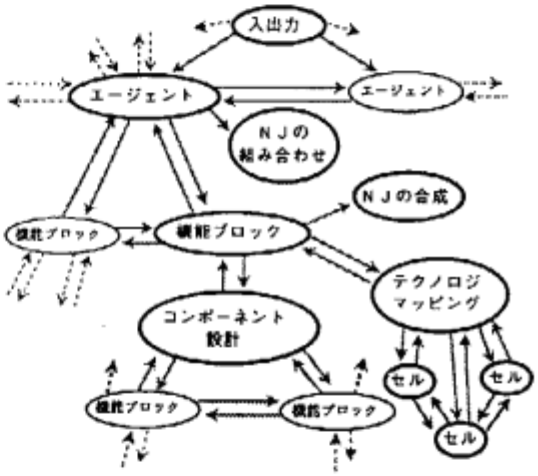

図5　プロセスの概念図

入出力プロセスは、入力された回路の仕様を分割し、分割した回路の部分仕様とデフォルトNJを各エージェントに通知する。

エージェントプロセスは並列協調方式に従う。部分仕様が通知されると、制約条件を考慮しない詳細化を開始する。詳細化のために、機能ブロックプロセスを生成する。デフォルトNJが通知されると、エージェントの属性に関する上限値を示す、エージェントの制約条件（担当する機能ブロックに関する局所的な制約条件）を生成する。設計の進行に伴って、他のエージェントから属性値またはNJが通知されたときも、同様な制約条件を生成する。エージェントの制約条件をもとに、機能ブロックの属性に関する上限値を算出し、

担当する機能ブロックへ一斉に通知する。上限値は各機能ブロックの設計値に応じて変更する。

NJの組み合わせプロセスは、他のエージェントから通知されたNJを組み合わせたNJを生成する。NJの組み合わせプロセスはエージェント毎に生成する。

機能ブロックプロセスは、設計ルールを参照し、詳細化することにより設計を行なう。すなわち、ひとつ下位のオルタナティブ（コンポーネント設計／テクノロジマッピング）のプロセスを生成し、1レベル詳細化した機能ブロック／セルの仕様をオルタナティブプロセスに通知する。上限値が通知されたときは、下位のオルタナティブに上限値を越えない設計を依頼する。設計に成功したときは結果を上位のプロセスに返し、このオルタナティブをIN状態として記録する。失敗したオルタナティブもOUT状態として記録しておく。すべてのオルタナティブが上限値を越えたとき、NJの合成プロセスにより、これ以上厳しい条件のときの再設計を禁止する情報を生成し、設計可能な最小の属性値を返す。NJの合成プロセスは機能ブロック毎に生成する。

コンポーネント設計のオルタナティブプロセスは、機能ブロックを1レベル詳細化した構成方法に対応する。上位の機能ブロックプロセスから与えられた上限値を越えないような、下位の機能ブロックの上限値を算出し、機能ブロックへ一斉に通知する。設計の進行に伴って、生成する上限値は下位の機能ブロックの設計値に応じて変更する。

テクノロジマッピングのオルタナティブのプロセスは、セルへのマッピングの方法に対応する。構成要素のセルが通知されると、セルのプロセスを生成する。与えられた上限値を越えるか否かの結果と、設計結果（ゲート数と遅延時間）を上位の機能ブロックプロセスに返す。セルプロセスへの遅延時間の計算の依頼は、パスの始点となるすべてのセルへ一斉に通知する。セルプロセスには、セルの特性値、接続関係を記録する。

5．実験結果

協調設計機構の動作の確認と、いくつかの例題を用いた実験を行った。

5．1　最適化

最適化は、面積又は遅延時間制約の一方を固定して、もう一方の制約条件を徐々に厳しく（設計された回路より1だけ小さく）なるような制約を設計に失敗するまで与え続けることによって行う。設計に失敗した直前の値が最適値を示す。

「2点間の距離を整数で近似する回路（MAG）」における最適化の様子を図6に示した。まず、面積制約を十分に大きくしておき、遅延時間制約を130 nsとすると、右端にプロットされた面積および遅延時間となるような回路を得た。面積制約を厳しくし続けると、異なる結果が次々と得られ、ついには設計不可能となる。左端にプロットされたものがもっとも小さい面積の回路である。このとき、1224 > CHIPというNJが提示される。このNJを成り立たせるような制約値では、設計できないという意味である。

表1は、最適化と、面積制約を1224と1223とした制約充足の比較である。最適化の時間は、制約充足に比べて短縮されている。これは、ゆるい制約条件での制約充足の時点で生成されたNJが、探索空間の絞り込みに有効に働いているためである。

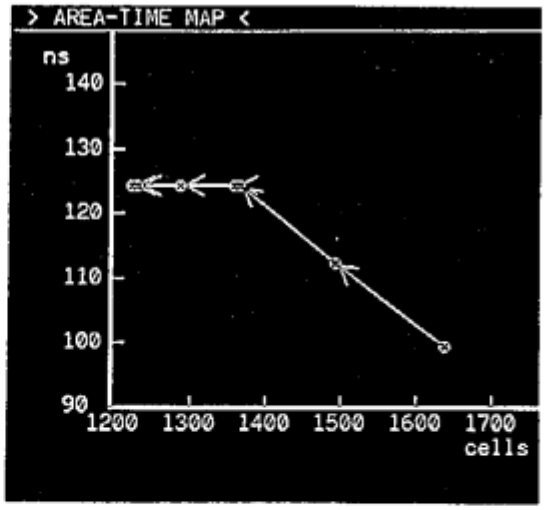

図6　最適化（2点間の距離を整数で近似する回路）

表1．　最適化と制約充足の比較

	面積制約	設計結果	処理時間(sec)
最適化	1639 to 1223	1224	9.8
制約充足	1224	1224	10.7
制約充足	1223	設計不可能	10.5

５．２　台数効果

台数効果は、エージェントを１から１５までの範囲で増加させ、さまざまな制約での時間を測定した。１エージェントを１プロセッサに割り当て、部分回路の分散と計測のためにに１プロセッサを用いたので、最大で１６プロセッサを使用した。

co-LODEX向きの例題

多段加算器（配列加算器）の例を示す。多段加算器は、９個の２の補数表現の整数の和を求める回路であり、122個の１ビット加算器で構成されいる。次項で述べる回路のALUや乗算器の構成要素として使われているものである。１ビット加算器は、二つ以上の１ビット数を加算し、和とキャリーを出力する。１ビット加算器には入力数に応じて、組合せ的な設計方法があり、多段加算器全体では５千万以上の設計方法となる。表２に、設計方法の数と入力数、出力数の関係を示した。１ビット加算器の機能ブロックは、直接CMOSスタンダードセルで実現される。すなわち、１レベルの詳細化が行われるだけで、階層構造をとらない。このような構成の回路を例題にすることによって、co-LODEXの協調設計機構だけを対象とする実験ができる。制約条件として、30個のデフォルトNJを与えた。

図７は、多段加算器の一部である。箱は１ビット加算器を表し、その中に書いてある数字は入力数である。矢はデフォルトNJを表す。上下あるいは、左上、右下の位置関係にある１ビット加算器は、同一のデフォルトNJに関係している。従って、部分回路への分割は、デフォルトNJをなるべく分割しないように、垂直あるいは、左上から右下に向かう境界線で分ける。

本実験では、各エージェントにおける設計処理の量についても考慮した。設計処理の量はエージェントが担当する機能ブロックの設計方法の数によると考えているので、担当する１ビット加算器の設計方法の和を均等にするようにした。図７の網かけの領域は、エージェントの担当する部分回路の例である。多段加算器は、規則的な構造であるため、分割しやすい。

エージェント数と台数効果の関係を図８に示した。破線は、理想的な、エージェント数倍の台数効果を表している。面積制約の変更では、すべてのエージェントが活発に動作する。面積制約の変更では、期待どおりの高い台数効果を得ている。破線で表したエージェント数倍を越えている理由は、協調設計機構によって計算量が減少しているためであると考えられる。一方、遅延時間制約の変更では、クリティカルパスに関係するいくつかのエージェントが活発に動作するだけで、それ以外のエージェントは、ほとんど動作しない。すなわち、稼働しているプロセッサ数はエージェント数より少ない。初期設計の時間は、ほぼ横這いになっている。この理由は、分散によって各エージェントの処

表２．　設計方法の数と入出力数

入力数	和	キャリー	設計方法の数
1	1	0	1
2	1	1	1
3	1	1	1
4	2	1	12
5	2	2	30
6	2	2	15
7	3	2	105
8	3	3	420
9	3	3	84

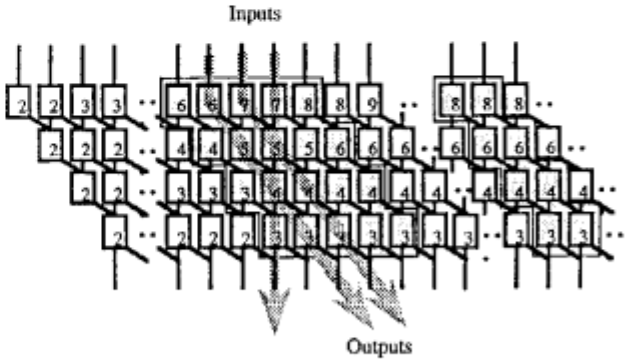

図７　多入力加算器

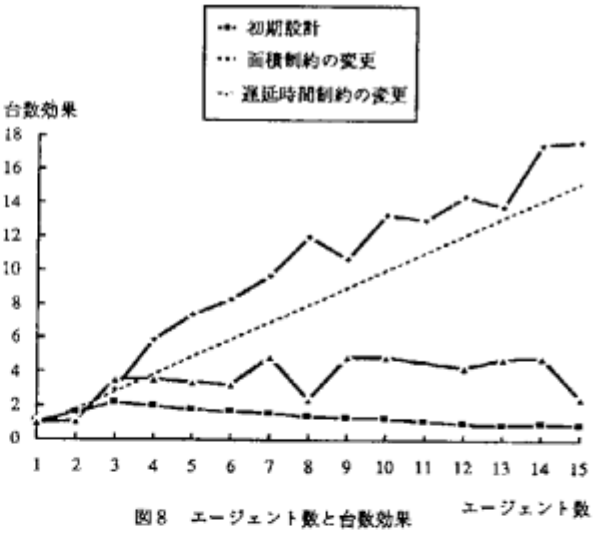

図８　エージェント数と台数効果

理量は減少しているが、設計仕様を分散するためのオーバヘッドも増加しているためであると考えている。

一般的な回路

表3は一般的な例題に対する実験結果である。処理時間が最小を示したエージェント数と、その台数効果を示した。台数効果は我々の期待を下回るものであり、そのエージェント数もプロセッサ数に比べて少ない。

分割されるデータパスは、様々な種類の機能ブロックを含んでいる。例えば、ALUなどのように、階層的な詳細化を繰り返すことが必要な、複雑な機能ブロックと、レジスタのように直接マッピングでき、構成方法の候補も少ない機能ブロックが混在している。プロセッサの稼働状況を観察すると、一つか二つのエージェントが活発に動作し、それ以外のエージェントは、忙しいエージェントからのメッセージ待ちの状態になっている。処理時間は、複雑な機能ブロックを担当するエージェントの設計時間に依存してしまう。

台数効果を得るためには、エージェントへの分割の際にクリティカルパスに注目するだけでなく、①機能ブロックライブラリを先読みして機能ブロックの複雑さを考慮すること、②階層構造上にサブエージェントを設けることが考えられる。

6．おわりに

協調型論理設計エキスパートシステムco-LODEXについて述べた。co-LODEXは、回路を部分回路に分割し、プロセッサごとに部分回路を設計する。プロセッサ間で設計値あるいは、NJを交換する協調設計アルゴリズムにしたがって、面積と遅延時間に関する制約を満足するために、評価・再設計が繰り返される。並列協調動作により、設計時間の短縮が見込まれる。実験では、プロセッサ数倍の台数効果を観測できた。

回路全体の最適化を効率的に行うことも特徴である。例えば、ある遅延時間制約を満足する、面積最小の回路を求めることができる。

co-LODEXでは、論理設計における組合せ的爆発に対して、NJを利用した手法による解空間の絞り込みを行っている。

今後の課題は、エージェント内の設計、評価・再設計の並列化である。並列化に当たっては、負荷の分散が重要である。

表3．　実験結果

回路	機能ブロック数	主な機能ブロック	台数効果	最適なエージェント数
GCD	11	1 subtracter, 1 comparator	1.1	2
DiffEQ	28	1 multiplier, 1 ALU(add/subtract) 1 comparator	1.8	5
MAG(1)	14	1 ALU(add/subtract), 1 comparator 1 two's complementer	1.7	4
MAG(2)	13	1 ALU(add/subtract/compare) 1 two's complementer	1.2	3
MAG(3)	16	1 adder, 1 subtracter, 1 comparator 2 two's complementers	1.8	5
Correlation	22	RAMs, 1 ALU(multiply/add) 1 adder, 1 comparator 1 decrementer, 1 incrementer	1.3	4

GCD: Greatest common devisor [7]

DiffEQ: Differential equation: $y'' + 5xy' + 3y = 0$ [8]

MAG: Approximation of $(a^2 + b^2)^{1/2}$ [9]

Correlation: Correlational function $y[i] = \sum_{j=1}^{N-1-i} x[j] * x[i+j]$ [10]

謝辞

本研究は第五世代コンピュータプロジェクトの一環として行なわれているものであり、御支援頂いたICOTの生駒研究部長代理（現在、NTTデータ通信株式会社）、新田第七研究室長に深く感謝いたします。

参考文献

[1] 丸山他「評価・再設計機構を備えた論理設計支援システム」信学論　A Vol. J72-A No.8 pp.1172-1180 ,1989.8

[2] 笈田他「協調型論理設計エキスパートシステムco-LODEX—概要—」情全国大会 1991.3

[3] 澤田他「協調型論理設計エキスパートシステムco-LODEX—試作—」情全国大会 1991.3

[4] K.Ueda, T.Chikayama, Design of the Kernel Language for the Parallel Inference Machine, The Computer Jurnal, Vol.33, No.6, 1990, pp.494-500

[5] K.Taki, The parallel software reseach and development tool: Multi-PSI system, Programming of Future Generation Computers, North-Holland, 1988, pp.411-426

[6] F.Maruyama et al. , Solving Combinatorial Constraint Satisfaction and Optimization Problems Using Sufficient Conditions for Constraint Violation, Proc. of the Fourth International Conference on Artificial Intelligence, 1991

[7] P. Camposano, Structural Synthesis in The Yorktown Silicon Compiler, Proc. VLSI'87, pp. 29-40

[8] F. D. Brewer, Knowledge Based Control in Micro-Architecture Design, Proc. of the 24th Design Automation Conference, pp.203-209 (1987).

[9] H. Trickey, Flamel: A High-Level Hardware Compiler, IEEE Trans. Computer-Aided Design, Vol. 6, No. 2, March 1987, pp. 259-269

[10] K. Shirai et. al, Development and Evaluation of an Architecture Design System for Application-Specific Integrated Circuits, International Journal of Computer Aided VLSI Design 1, pp. 391-414 (1989)