ICOT Technical Report: TR-0759

TR-0759

並列推論マシン上でのLSI レイアウトシステムCo-HLEX

渡辺　俊典、小松　啓子（日立）、小田　理
山口　英之（インテリジェントシステム）

April, 1992

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

並列推論マシン上でのLSIレイアウトシステムCo-HLEX

渡辺俊典　（株）日立製作所システム開発研究所
小松啓子　（株）日立製作所システム開発研究所
小田理　インテリジエントシステムズ株式会社
山口英之　インテリジエントシステムズ株式会社

を生成する。

set_power_net_props/3は生成した電源配線の最大配線長と合計電位を親の回路モジュールへ報告する。報告方法は、上記generate_power_subproblems節で生成、設定された変数を使用して行う。

```
module([{aggregate_powerlines,Proc,end-Eo}|TMess,LM):-
    get(patent_shape,LM,Shape,LM1),
    get(layoutframe,LM1,LFName,LM2),
    get_and_put(power_nets,LM2,PowerNets,NewPowerNets,NLM),
    planframe(LFName,get_power_route_patterns,PowerRPatterns),
    find_a_good_power_route_pattern(PowerRPatterns,PowerNets,Shape,PowerNets1),
    set_power_net_props(PowerNets1,NewPowerNets,end-Eo),
    module(TMess,NLM).
```

4．9　信号配線

route_signallinesメッセージは、信号線の詳細配線準備を行う。これは、各セルの実コネクタ（トランジスタのベース、コレクタ、エミッタ等）間を結ぶための仮想コネクタをセルの外郭上に作成する。これは、配置処理と極めて類似の再帰構造をしている。配置処理においてラフ配線に使用したgenerate_subproblems,narrow_leaf_connsメッセージが、ここでも使用される。第1節のnarrow_leaf_connsメッセージは、セルの回路モジュールの保持する外郭上コネクタのrange-flagにfinishedというマークを付ける事によって、近接モジュールによる当該コネクタの引き込みアクションを許す。第2節は、分割を行うブロック階層の回路モジュールの処理を行う。generate_subproblemsメッセージにより、子回路モジュール間の配線計画を立て、区画境界に生成した仮想コネクタを各子回路モジュールに外郭上コネクタとして登録する。これにより、各子回路モジュールは親の回路モジュールと同様に計画レイアウトを持つことになり、親の回路モジュールを処理したプログラムで再帰処理出来ることになる。route_signallines_all_sonsメッセージは、その再帰処理を行う。

```
module([{route_signallines,Proc,end-Eo}|TMess],LM):- cell(LM) |
    send_message(TMess,[{narrow_leaf_conns,Proc,end-Eo}]),
    module(TMess,LM).
module([{route_signallines,Proc,end-Eo}|TMess],LM):- not(cell(LM)) |
    send_message(TMess,[{generate_subproblems,Proc,end-E1},
                        {route_signallines_all_sons,Proc,E1-Eo}]),
module(TMess,LM).
```

1　概要

1．1　研究の位置付け

発電設備診断システムに必要な大規模、多種の設備等のレイアウト設計を知的に支援する機能を実証するため、知的レイアウトCADシステムの試作を行う。

周知のようにレイアウト問題は、複数個の事物を限られた空間の内部に配置したり、それらの間を配線する問題であり、発電プラント、ＬＳＩから都市、ビルあるいは新聞紙面の設計に至るまで広く見られる。莫大な組み合わせの可能性の中から、適切な解を探索するという共通の側面から組み合わせ問題と呼ばれる類に属する。探索効率を上げるためには、問題の持つ意味構造を利用した不要探索枝の消去や、並列探索による探索速度の向上が有効である。前者には記号処理言語ＫＬ１による、対象や問題解決知識の表現が有効であり、後者には並列推論マシンの計算力が有効である。

各種のレイアウト問題の内から、本研究では電気回路系統のものを取り挙げた。発電設備設計やＬＳＩ設計がこの系統の問題に属し、電気回路部品を有限平面内に配置・配線する問題となる。特に後者は、ＬＳＩ技術の急速な発展の中で、計算機応用を期待される最も重要な問題と言っても良い。この理由から、本研究ではＬＳＩレイアウト問題を取り挙げた。ＬＳＩレイアウト用ソフトウエアの内容や規模は大変複雑なものとなっており、ソフトウエア生産工学の観点からみてもその開発コストの削減自体が重要な問題である。ＫＬ１言語の得意とする並列オブジェクト指向プログラミング技術は、人間の思考と計算機とを比較的少ないセマンティックギャップで埋めるものであり、ソフトウエア生産性の向上にも有効である可能性が高い。

1．2　開発方針

1．2．1　前期～中期研究の経過

前期～中期においては、逐次型推論マシンＰＳＩに代表される擬似並列計算用の環境が提供されたので、後期の本格的並列処理への足がかりとして比較的小規模のレイアウト問題解決を試みた。対象としては、計算機室への機器のレイアウト問題を取り上げた。

1．2．2　後期研究の経過

ＩＣＯＴ後期３ヶ年間を本格的レイアウト問題解決研究にあてることとし、以下の経過で研究を進めてきた。

（１）平成元年度：基本アルゴリズムの開発を意図し、大規模問題を少ないソフトウエア規模で処理可能とするための階層再帰並列協調算法HRCTL(Hierarchical Recursive Concurrent Theorem prover for Layout)を開発した。HRCTLを核としたレイアウトシステムCo-HLEX(Co-

```
    Rb = [{route_leafcells,Type,Proc,end-E3}],
    Ru = [{route_leafcells,Type,Proc,end-E4}],
    judge_end([E1,E2,E3,E4],Eo),
    module(TMess,NLM).
```

partition_connectors/4は、配線指示Typeによりセル内ネット問題を分析し、配線指示に対応したネットの全コネクタ（外郭上コネクタ、実コネクタ）プロセスのストリームを取得する。collect_routable_netsは、取得したネット問題の中から迷路法に配線依頼出来る状況に成ったものから順次そのコネクタ情報を送信する。routable_netの節で使用されるfixed_conector_range/1が、配線依頼完了の判断を行う。この判定は、全コネクタの中に高々1個でもコネクタ位置が固定されているコネクタが存在する時のみ成功する。この条件は、実コネクタを含むネット問題では常に成功する。反して、実コネクタを含まないネット問題（フィードスルー配線）では常に成功するとは限らず、配線依頼が遅れる事になり出来る限り直線的(最短)に配線する事が期待できる。

```
partition_and_get_connectors(Type,LM,RoutableNets,NLM,end-Eo):-
    partition_connectors(Type,LM,NetsList,NLM),
    collect_routable_nets(NetsList,CName,RoutableNets,end-Eo).

collect_routable_nets([],RoutableNets,Ei-Eo):-
    close([RoutableNets],Ei-Eo).
collect_routable_nets([Net|Rest],RoutableNets,Ei-Eo):-
    merge([RoutableNet,RoutableNets1],RoutableNets),
    routable_net(Net,RoutableNet,Ei-E1),
    collect_routable_nets(Rest,RoutableNets1,E1-Eo).

routable_net(Net,RoutableNet,Ei-Eo):-
    get_and_put(cont_range_layer,Net,ContsProfile,Net1,Ei-E1),
    fixed_connector_range(ContsProfile) |
    RoutableNet = ContsProfile,
    close_connectors_stream(Net1,E1-Eo).
routable_net(Net,RoutableNet,Ei-Eo):-
    get_and_put(cont_range_layer,Net,ContsProfile,Net1,Ei-E1),
    not(fixed_connector_range(ContsProfile)) |
    copy_old_connector_data(ContsProfile),
    routable_net(Net1,RoutableNet,E1-Eo).
```

２．２．３　プロセス条件

トランジスタ、抵抗、容量、配線引き出しコンタクト、異配線層間スルーホール、配線等のレイアウト基本要素の幾何学的形状と、電気的特性や製造時の歩留まりを保障するために設定された要素間の許容間隔。基本要素はシリコン上の拡散層とその上空のアルミ配線層を用いて形成される多層構造物であるため、要素形状も各層について与えられる。また、要素間の距離的制約としては、同一配線層内での配線間の間隔やコンタクトやスルーホール周辺での配線通過禁止域がある。これらは製造時にコンタクト等がシリコン層内に歪を発生させ、周囲に形成する配線等の接続不良の原因となるためである。回路をＬＳＩ化する際に、チップサイズ、消費電力、動作速度によってCMOS,Bipolarといった異なるデバイス系列の中からいずれかを選択する。同じデバイス系列の中でも、実現可能な最小加工寸法や使用可能な配線層数の異なる製造プロセスが存在する。よって製造プロセス名を指定すれば実現可能デバイス、素子形状、レイアウトルール等の諸条件は大略ユニークに決まる。

ここで大略としたのは、抵抗や容量は適当に分割可能であり、分割したものを接続することによって当初の特性を実現できるものは、レイアウト形状が確定しているとは言えないからである。レイアウトに際して、プロセス名は前提として与えられる。

２．２．４　配置配線の拘束条件

回路ネットワークは、素子間の接続状態を表すトポロジーデータであり、基本的には素子の接続関係を与えるに留まる。シリコン上に単にこの接続関係を満たすように素子とそれらの間の配線を形成しても回路は正常に動作はしない。それは実配線の持つ容量、チップ上の異なる場所に素子を形成した場合の電気特性のばらつき、隣接する素子や信号線の動作時の周辺素子への輻射の影響等によって回路設計時に予定していなかった電気的効果が現れるためである。

この現象を一般に寄生素子の発生と呼んでいる。寄生素子の発生防止のためには、特性を揃えたい素子（例えば差動アンプを形成する２つのトランジスタや、回路動作時に両者の比が問題となるような抵抗）や相互間配線容量を少なくしたい素子は近くに配置することが必要となる。

これらの条件は、現場ではペア仕様と呼ばれており、近接同方向配置すべきトランジスタ群、特性を揃えなければならない抵抗群等が指定されている。

２．２．５　Co-HLEXが処理する原レイアウト問題は、次のゴール節で定義できる。

```
:-module solve_a_layoutproblem(+,-,+,+).
?-solve_a_layoutproblem(CirNet,LPlan,Proc,Constr).
```

ここに、

CirNet::=回路モジュールとモジュール間の配線ネットデータ。

LPlan::=回路モジュールとモジュール間配線ネットの幾何形状データ。これがレイアウト結果

AnsWires::=配線結果データ

AnsTHs::=配線層変更スルーホールデータ

MRPデータ

MD::={MRPAddr,RVFlag,MRPAddr,MRPShape,Permits,Streams,PCosts,Costs,Temp}.

RVFlag::=real/virtual,セル内MRPの場合はreal、仮想コネクタMRPの場合はvirtual

MRPAddr::={X,Y}/n/w/s/e,realの場合はMRPの中心位置{X,Y}、virtualの場合はn/w/s/e

MRPShape::={Xs,Xe,Ys,Ye},MRPの形状

Permits::=[NetName|_],障害物の場合、MRPを適用できるネット名称

Streams::=[{wake/sleep,Dir,SendMRP,Cost,Stream}|_],隣接MRPへのストリーム群

wake/sleep::=経路探索メッセージ送出可／否

Dir::=経路探索メッセージ送出方向

SendMRP::=経路探索メッセージ送信MRP

Cost::=経路探索メッセージコスト

Stream::=経路探索メッセージ送信MRPへのストリーム

PCosts::={AL1Cost,AL2Cost,AL3Cost}

AL1Cost::={Straight,Crook,Through_hole},PermitsネットのMRP適用コスト

Staright::=直線通過メッセージコスト

Crook::=曲折通過メッセージコスト

Through_hole::=スルーホール化コスト

ACosts::=Permitsネット以外のMRP適用コスト

Temp::={AL1Path,AL2Path,AL3Path}

AL1Path::={SenderMRP,Dir,Cost},経路探索時一時保存データエリア

SenderMRP::=経路探索メッセージ送信元MRP

Dir::=経路探索メッセージ送信元方向

Cost::=経路探索メッセージ到着コスト

4．11．3　セル内迷路法配線実行

wireメッセージ受信の第1節は、InNetで渡されたコネクタ群を結ぶ配線を行う。

まずコネクタ群から、配線経路点となるMRPをget_conn_mrp/5によって求める。セルの内点コネクタは、セル面上のMRPを求めるが、外郭上コネクタは仮想コネクタMRPを選択する。

次に外郭上コネクタの存在可能範囲より、配線の引き出し点としてはいけない辺上の範囲を求める。

MRPNストリームへrefleshメッセージを送出して各MRPの初期設定を行い、become,sleepメッセージにより引き出し不可範囲にある最外郭MRPをsleep状態化し、引き出し点適用不能とする。

layout_all(Tcs,Tls,Proc).

（3）技術課題

generate_subproblems(CirNet,SonsCirNets,Constr)でネットワークデータCirNetを部分ネットの集合SonsCirNetsに分割し、layout_all(SonsCirNets,SonsLPlist,Proc)で各々のレイアウトSonsLPlistを相互独立に得る場合、部分回路の形状や配線の出し口が整合されず、check_shape_and_wire_abutment(SonsLPlist)のテストを通過するのが困難となる。このテストを甘くすると、部分回路間の形状不整合部分での空きエリアや配線屈曲用のチャネル領域が大きくなり、得られるLPlanのチップ面積は大きくなってしまう。

2．3　問題の解決法

2．3．1　回路ネットデータの事前階層化

与えられたCirNetを4分木に変換する事前処理を設ける。近くに配置するモジュールを4分木の下位階層にまとめることによってCo-HLEXの処理結果においても、近接性を満足させうる。この方法によって、generate_subproblems/3の処理負荷を軽減させる。

2．3．2　モジュール形状の縦方向管理による相互整合化

generate_subproblems/3によって、親ノードの長方形はいくつかの区画に分割され、各々に子回路が埋め込まれる。区画の目標形状は子回路の持つ推定面積と親ノードの形状に従い計画する。layout_all/3によって各子供をレイアウトするとき、各子供が親から与えられた目標形状を遵守することにすれば、各子供を互いに並列に処理しても最後に得られる親のレイアウトは、予定に近いものになると期待できる。図2－3（1）に縦方向の整合を示す。

2．3．3　配線における横方向の協調

異なるブロック間での共通配線の引き出し口の自動整合問題は、コネクタ整合問題としてＬＳＩレイアウトでの有名な難問である。並列走行プロセス間でのこの問題は、現在までのところ未解決である。

並列走行プロセス間で共通配線の引き出し口の位置設定をめぐる協調を行わせることによって、この問題を解決する。これによって、上述のエレガントな再帰式記述が生かせることになる。

前述のダミーコネクタを協調処理におけるプロセス間通信のメール箱として利用する。ダミーコネクタは、任意の時点での配置予定範囲と、アクセスしたプロセス名などを記憶している。generate_subproblems/3は、これらの情報とブロックの内部に埋め込んだ子回路のネットとの接続要否を参照しながらダミーコネクタの配置予定範囲を狭めることを試みる。このgenerate_subproblems/3は範囲を狭める資格が有るか無いかを考慮し、無い場合には他の並列走行プロセスに譲る。前述の内部端子を持つプロセス、または連続したアクセスでないプロセスは、

```
    NeigStr,MD) :-
  MD  = {RVFlag,MRPAddr,MRPShape,Permits,Streams,PCosts,Costs,Temp},
  inside(MRPAddr,ObsRange) |
  purge([PermitNet|Permits],Permits1),
  add_cost(ObsLayer,Costs,AddCost,Costs1),
  MD1 = {RVFlag,MRPAddr,MRPShape,Permits1,Streams,PCosts,Costs1,Temp},
  mrp([def_obs,ObsT,end-Eo|MRPNStr],NeigStr,MD1).
otherwise.
mrp([[def_obs,[_|ObsT],end-Eo]|MRPNStr],NeigStr,MD) :-
  mrp([def_obs,ObsT,end-Eo|MRPNStr],NeigStr,MD).
mrp([[def_obs,[],end-Eo]|MRPNStr],NeigStr,MD) :- Eo = end,
  mrp(MRPNStr,NeigStr,MD).
```

スルーホール敷設禁止登録

全てのセルは南辺上と東辺上をスルーホール敷設禁止とすることで、隣接セル間でのスルーホール近接チェックを不要としている。このためのスルーホール敷設禁止登録を行う要求で、障害化する範囲を管理するMRPにスルーホール化を禁止できるコストを与える。

以降、上記のMRP障害化と同様に第2節、第3節の記述が必要であるが、類似処理であるため省略する。

```
mrp([[def_pobs,[ObsRange|ObsT],end-Eo]|MRPNStr],NeigStr,MD) :-
  MD  = {RVFlag,MRPAddr,MRPShape,Permits,Streams,PCosts,Costs,Temp},
  inside(MRPAddr,ObsRange) |
  add_cost(thorugh_hole,Costs,AddCost,Costs1),
  MD1 = {RVFlag,MRPAddr,MRPShape,Permits,Streams,PCosts,Costs1,Temp},
  mrp([def_pobs,ObsT,end-Eo]|MRPNStr],NeigStr,MD1).
```

MRPリフレシュ

経路探索一時データは、配線層毎に経路探索メッセージ出力側MRPとそのN,W,S,E方向と自MRPへのメッセージ到着時コストを保存するもので、コスト小の経路探索メッセージが到着する度に書き換えられるデータである。この到着時コストに最大値を初期設定することで、新たに経路探索メッセージを受け付け可能にする。

```
mrp([[refresh,end-Eo]|MRPNStr],NeigStr,MD) :- Eo = end,
  MD  = {RVFlag,MRPAddr,MRPShape,Permits,Streams,PCosts,Costs,Temp} |
  init_route_cost(Temp1),
  MD1 = {RVFlag,MRPAddr,MRPShape,Permits,Streams,PCosts,Costs,Temp1},
```

Co-HLEXにおけるレイアウト作成は、回路モジュールプロセスへのメッセージパッシングによる各種機能の実行により行う。回路の階層データや素子データ等が各々プロセスであり、プロセスはメッセージ受信により各レイアウト機能の実行を要求され、その機能実行によりプロセス状態が遷移する。レイアウトはプロセス状態遷移の連鎖で実現される。
以下、各レイアウト機能について概要を述べる。

3．2．1　バックアップメモリー

メモリーには、レイアウトの進捗に伴って生成される下記のデータがバックアップ保持される。
（1）階層回路ネットデータ：回路ネットを回路木生成機能によって階層化したデータ。
（2）階層レイアウトデータ：階層回路ネットに対して各階層ブロックの形状や配線データ等のレイアウトデータを付加したもの。

3．2．2　回路木生成機能

平板な原回路ネットや回路の予定形状と外部端子位置情報を元に、予定形状を2～4のサブブロックに分割する。分割に当たっては、多くの分割代替案の中から予定形状を実現しやすく、部分ブロック間の配線距離が短くなる案を選択する。このために、外部端子とブロック内回路素子との距離を求めて分割時の参考とする。一つの階層の分割が終了したら、各サブブロックのN，W，S，E境界を通過するネットの仮想端子と内部ネットが決まる。これは親ブロックと同一の形式を持っているので、親と同じ処理を再帰的に実施することができ、末端素子に至るまでの階層化を行える。ペア条件を与えられた素子に対しては、ペア間のアーク距離が0に近いとしておくことで自動的に同じ階層ノードに入っていく様にできる。

ブロック内部ネットの各素子の位置は、外端子からネットをたどってその素子までに至る最短距離であると定義する。最小距離の探索には、素子をノードとしネットをアークとするグラフに対するダイクストラの最短パス探索アルゴリズムを使用する（ＩＣＯＴで開発されたその並列版を使用する）。ただし、適用対象が没個性のノードでなく、トランジスタ、抵抗などとなり、信号伝播可能方向が入出力インピーダンスに依存する。このことを考慮し、例えばトランジスタへのベース入力信号はコレクタとエミッタへは短距離で伝わるが、逆方向は遠距離となる等、伝達してきた信号の属性に応じた距離を与えることによって回路動作時の信号伝播状況を素子間距離に反映させるようにした。

素子位置分析の結果、各素子はN，W，S，E（北、西、南、東）辺からの距離指標を持つこととなる。距離指標の大小により2～4サブブロックに分割する。分割テンプレート（辞書）の中から分割後のサブブロックのアスペクト比（縦横比）が極端にならない案を採用する。
回路素子をその位置によって4分区画または2分区画内の適性な部分に所属させる。各々の区画は階層のモジュールプロセスとしてシステム内に生成する。この後、それぞれの区画について分割を再帰実行する。

```
    mrp([collect_route,NetName,EndMRPs,StartMRP,RouteMRPsT,E1-Eo|MRPStr],
    NeigStr,MD1).
otherwise.
mrp([[collect_route,NetName,[_|EndMRPs],StartMRP,RouteMRPs,end-Eo]|MRPStr],
    NeigStr,MD) :-
    mrp([collect_route,NetName,EndMRPs,StartMRP,RouteMRPs,end-Eo]|MRPStr],
    NeigStr,MD).
mrp([[collect_route,NetName,[],StartMRP,RouteMRPs,end-Eo]|MRPStr],NeigStr,MD) :-
    RouteMRPs = [],
    Eo = end,
    mrp(MRPStr,NeigStr,MD).
```

4．11．4　セル内配線物の取得

レイアウトの出力機能から要求され、天元コネクタMRP保存の配線とスルーホールのデータを返送する。

```
top_mrp([[get_wire,WireLayout,end-Eo]|MazeStr],TM) :-
    TM  = {Shape,Distance,Proc,MRPNStr,AnsWires,AnsTHs} |
    WireLayout = {AnsWires,AnsTHs},
    top_mrp(MazeStr,TM).
```

4．12　ライブラリ群

4．12．1　プランフレーム

テンプレートlayoutframe/8に依存した問題解決処理を行う節をまとめてplanframeと言う名のライブラリ群を定義している。これによってレイアウトにおける問題分割や解の統合を行う。いくつかの例を示す。

（1）形状評価

採用候補テンプレートに子回路を埋め込んだとした時に得られる親回路の形状や空きエリア等を求め、最適計画を選択する前用意とする。

```
planframe(LFName,evaluate_shape,OmyPlan,SonsPlace,NmyPlan,NSonsPlace,
    NDeadsp,NAspect) :-
    LFName = r4h_uls |
    get_put_elements(OmyPlan,[X0,_,_,X3,Y0,_,_,Y3],[X0,X1,X2,X3N,Y0,Y1,Y1,Y3N],
    NmyPlan),
    get_sons_area(SonsPlace,[SLu,SLb,SRb,SRu]),
```

原点とする。また、配置時に生じたデッドスペースを各セルに含ませ、これを各セルの実形状とすることでデッドスペース領域も配線可能となる。

次に、トランジスタや抵抗などのセル内配線に迷路法を用いるため、格子状に連結されたプロセス群を生成する。これらを制御する天元プロセスを生成し、通信用ストリームを各セルモジュールとの間に設定する。セル配線において、各セル内の南辺と東辺の最外郭部分のプロセス行、列をスルーホール敷設禁止とすることで隣接セル間でのスルーホール近接チェックが不要となり、計算スピードの向上に役立つ。

最後に、最上位回路ブロックの外周辺コネクタ配置位置をブロックの実周辺位置に合わせる。

３．２．５　配線機能

（１）電源配線

電源線は供給素子が多く、さらに電圧降下や電流密度を考慮した配線レイアウトが必要なため、信号線とは異なる扱いをする。

（ａ）ブロック内電源配線

分割可能なブロックモジュールでの処理を行う。親ブロックの問題分割フェイズでは、子回路が配線した電源ネットの配線長と電位を報告させるための準備のみを行う。この準備を子回路について再帰的に実行する。

セル内電源配線終了後の問題統合フェイズでは、各子回路間をつなぐ電源配線パターンをライブラリより選び出し、報告された各子回路の電源線の配線長と電位から親ブロック内の電源配線幅を決定する。

（ｂ）セル内電源配線

セル内実コネクタと親ブロック電源配線接続用セル内仮想内点コネクタを生成し、これらコネクタの位置と配線引き出し層を取得する。両コネクタ間を接続するセル内配線は、下記の信号配線処理によって行う。

（２）信号配線

（ａ）ブロック内信号配線

信号線の配線経路計画を行う。各セルの実コネクタ（トランジスタのベース、コレクタ、エミッタ等）からの配線と、セル上空通過配線（フィードスルー線と呼ばれる）とを外部に引き出すための、仮想コネクタをセルの北、西、南、東いずれの外周に配置するかまでを計画する。配置処理においてラフ配線に使用した機能が、ここでも使用される。

（ｂ）セル内信号配線

セル内配線は各ネットごとの外周コネクタと実コネクタとの関係によって配線処理する順序を次の様に制御する。

最初、セルの南と東の外周辺上に存在する外周コネクタと実コネクタ間を配線する。次に、セルの北と西の外周辺上に存在する外周コネクタと実コネクタ間を配線する。最後に、外周コネクタ

```
    gen_vector([L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12,L13],
    BLineLayers,BSlice,GoalVec),
    gen_vector([0,0,0,0,0,0,0,0,0,0,0,0,0],BLineLayers,BSlice,InitVec),
    gen_vector([0,0,0,0,0,0,0,0,0,0,0,0,0],BLineLayers,BSlice,InitCost).
```

（4）ダミーコネクタ分配

親ブロック内に子回路を埋め込んで配線計画を作成した後、子回路間の境界線上に得られるダミーコネクタを境界線両側の子回路の外周コネクタとする。

```
planframe(LFName,distribute_connectors,NetData,[NLu,NLb,NRb,NRu],
    AnsSonNWSEConts):- LFNname = r4h_uls !
    like_guard_list(NetData,
    [Net,Nc,Wc,Sc,Ec,Ic,_,_,_,_,_,C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13]),
    cont_str_merge([C9,C10,C11,C12,C13],
    [C9A,C10A,C11A,C12A,C13A],[C9B,C10B,C11B,C12B,C13B]),
    NLu = [{Net,_,C8,C1,C11A,C9A,_}|NLu1],
    NLb = [{Net,_,{C11B,C12A},C2,C3,C10A,_}|NLb1],
    NRb = [{Net,_,C13A,C10B,C4,C5,_}|NRb1],
    NRu= [{Net,_,C7,C9B,{C12B,C13B},C6,_}|NRu1],
    AnsSonNWSEConts = [NLu1,NLb1,NRb1,NRu1].
```

（5）子回路レイアウトの統合

レイアウトが完成した子回路の外周形状を統合して、親回路の形状を求める。

```
planframe(LFName,aggregate_sons_shapes,PShape,
        [{LuW,LuH},{LbW,LbH},{RbW,RbH},{RuW,RuH}],RShape) :-
    LFName = r4h_uls !
    PShape = [PX0,PX1,PX2,PX3,PY0,PY1,PY2,PY3],
    RShape = [RX0,RX1,RX2,RX3,RY0,RY1,RY2,RY3],
    RX0 := PX0 , RY0 := PY0 ,
    RX1 := RX0+LuW , RY1 := RY0+~(big(LuH,RuH)) , RY2 := RY1 ,
    RX2 := RX0+LbW ,
    RX3 := ~(big(~(RX1+RuW),~(RX2+RbW))) ,
    RY3 := ~(big(~(RY1+LbH),~(RY2+RbH))).
```

（6）テンプレート属性の検索

種々のテンプレート情報をライブラリlayoutframe/8から検索する。配線パターン検索の例を示す。

```
planframe(LFName,get_route_patterns,RoutePatterns):-
    layoutframe(LFName,_,_,_,RoutePatterns,_,_,_).
```

（1）プランフレーム

配置配線における問題分割や解の統合を行う述語群であり、次に示す形状テンプレートであるレイアウトフレームの選択適用によりレイアウトを行う。

（2）レイアウトフレーム

（a）ブロックレベル

ブロック分割パターンで、2、3、4個の子回路を持つもの、合計11パターンを保有している。各パターン毎に電源配線パターンや内部区画間を結ぶ配線経路パターンライブラリも保有している。図3－4（1）に分割パターン、同図（2）に電源配線パターン、同図（3）に配線経路パターンの一例を示す。

（b）セルレベル

バイポーラ・アナログ用のトランジスタ、抵抗、コンタクト等について各々を構成する形状要素の集合体として、素子のレイアウトパターンを定義している。同図（4）にトランジスタ、抵抗、逆バイアスをかけるための島つりコンタクトの一例を示す。

（3）レイアウトルール

ＬＳＩチップの製造プロセス名に対応して、レイアウト形状物のサイズや相互間の距離等を保持している。

配置については、トランジスタや抵抗等の素子をシリコンウエハ上に拡散生成するときの、拡散層別のサイズや必要間隔、および配線取り出し口コンタクトが定義されている。

配線については、配線可能層数や配線幅および配線間隔、異配線層間を結ぶためのスルーホールの種類の定義がなされている。

3．2．7　出力機能

Co-HLEXの作成したレイアウト結果の描画を行う。図3－5に出力例を示す。

同図（1）は入力原回路図、同図（2）は回路階層化の実行結果である回路木、同図（3）は階層生成時に作成した仮配置案、同図（4）はレイアウト結果の出力図である。

4　サブシステムの説明

4．1　概要

Co-HLEXを構成するサブシステムについて述べる。記述の明確さ、簡潔さを得るために以下のような、ＧＨＣ記述を用いる。

```
Head:-Body_guards|Body_goals.
```

```
        cont(base,type1,center(20,36),[hole([layer(0),layer(1)],4,4)],
                                        [pad(layer(1),6,6)]),
        cont(emitter,type1,center(20,152),[hole([layer(0),layer(1)],4,4)],
                                        [pad(layer(1),6,6)])].
```

（5）抵抗体の形状

ベース拡散抵抗を拡散領域形状Diffusionとanodeおよびcathode配線引き出し口とで定義する。

```
layout_rule(res_parameter,process100,base_res,simple_res,Diffusions,Contacts):-
   Diffusion=[diff(base,p_diff,rect(0,0,8,48),_)],
   Contacts=[cont(anode,[normal,type1],center(4,4),
                  [hole([layer(0),layer(1)],4,4)],[pad(layer(1),6,6)]),
       cont(cathode,[normal,type1],center(4,44),
                  [hole([layer(0),layer(1)],4,4)],[pad(layer(1),6,6)])].
```

4．13　出力機能

Co-HLEXの作成したレイアウトデータの描画表示を行う。

top_module/5は、draw_layoutメッセージをmodule/2の形成するプロセスネットワークに流し、電源線、素子、配線の描画を要求する。要求を受けたモジュールは配置配線物を収集し、描画データに変換後OutStrストリームに流し込むことで描画出力装置に表示する。

module/2の第1節はセルの場合で、素子と素子内コンタクトをlayoutframe/8により矩形データに変換し描画出力する。また配線と配線層換えスルーホールをメーズルータプロセスネットワークから収集し描画出力する。第2節はブロック階層の場合で、電源線の描画表示を行うと共に下位階層の描画要求メッセージを送出する。

```
top_module([{draw_layout,Proc,end-Eo}|TMess],OutStr,ToModules,FmModules,TM):-
   OutStr = [OutStr1,OutStr2],
   ToModules = [{draw_layout,Proc,OutStr1,end-Eo}|RMS],
   top_module(TMess,OutStr2,RMS,FmModules,TM).
module([{draw_layout,Proc,OutStr,end-Eo}|TMess],LM):- cell(LM) |
   get(patent_shape,LM,Shape,LM1),
   get(layoutframe,LM1,LFName,LM2),
   get(mrpn_stream,LM2,MRPNMessage,LM3),
   take_cell_layout(LFName,Shape,CellLayout),
   MRPNMessage = [{get_wire,WireLayout,end-Eo}],
   OutStr = [{draw,CellLayout},{draw,WireLayout}],
   module(TMess,LM3).
```

top_module(InStr,OutStr,ToModules,FmModules,TM).

4．2．3　通信プロトコル

system/2に現れる各種ストリーム上の伝送メッセージの通信プロトコルを下記のように定める。

{object,Name,Type,Neighbours,Props}::=circuit_fileの出力する回路素子データである。回路ネットモジュールプロセスに変換される。

{net,Name,Type,Neighbours,Props}::=circuit_fileの出力するネットデータである。回路ネットモジュールプロセスに変換される。

{adjust_shape,W,H,Constr,Ei-Eo}::=素子形状整合化要求メッセージ。

{create_highest_module,CircuitData,Ei-Eo}::=最上位回路モジュールプロセス生成・挿入要求メッセージ。

{make_module_net,Ei-Eo}::=回路ネットモジュールプロセス間ストリーム接続要求メッセージ。

{gen_module_tree,Circuit,Ei-Eo}::=自動回路木生成要求メッセージ。

{create_layout_module,Proc,Circuit,Father,PEs,CPE,Ei-Eo}::=レイアウト機能実行用初期回路ネットモジュールプロセス生成要求メッセージ。

{place_by_qtree,Proc,Ei-Eo}::=配置実行要求メッセージ。

{prepare_routing,Proc,Ei-Eo}::=配線準備実行要求メッセージ。

{route_by_qtree,Proc,Ei-Eo}::=電源配線、信号配線実行要求メッセージ。

{draw_layout,Proc,Ei-Eo}::=レイアウト結果をXYプロッタに出力する描画機能実行要求メッセージ。

{disp_layout,Proc,Ei-Eo}::=レイアウト結果をterminalに出力する描画機能実行要求メッセージ。

{save_process_status,Proc,Ei-Eo}::=プロセスネットワーク内部状態のPSFileへの出力要求メッセージ。

4．3　入力データの表現

4．3．1　入力回路ネット

object(Name,Type,Nets,Props)::=部品素子データで、素子名称、素子タイプ、接続ネット情報等を保持する。

Name::=ObjName

Type::=transistor/resistor/{outlet,Dir}/・・・

Nets::=[{collector,NetName},{anode,NetName},/・・・]

Props::=[{npn_tr(1),SubProp},{br(void,void),SubProp},/・・・]

net(Name,Type,Gates,Props)::=配線のネット情報で、ネット名称、ネットタイプ、接続素子情報等を保持する。

```
    format_change(MD,Father,FmBelow,LM),
    module(TMess,LM)@node(CPE).
```

create_layout_moduleメッセージ処理の部分機能であり、負荷分散の中心的役割を果たすのは、assign_processor/4である。assign_processor/4は親モジュールプロセスの保有するプロセッサリストPEsを子プロセスの仕事量に応じて高々4個の部分リストに分割する。仕事量の目安としてはチップエリアの推定面積を用いる。部分リストから代表プロセッサを一つずつ選択してSonsPEsリストを作成し、それに各々の子プロセスを割り付け、並列走行を可能とさせる。また同時に対応する部分リストを各々の子プロセスに渡して、孫以下へのプロセッサ割当処理を再帰可能とする。

プロセッサPEsが分割不可能となってからは、親プロセスが割り当てられたプロセッサCPEを全ての子、孫等のプロセスが使用する。この場合、同一プロセッサを複数のプロセスが使用するために疑似並列モードの処理となる。

```
assign_processor(PEs,SonsAreas,SonsPEs,SonsCurrentPEs):-
    area_ratio(SonsAreas,SonsAreaRatioList),
    divide_pes(PEs,SonsAreaRatioList,SonsPEs),
    current_pe(CPE),
    define_sons_current_pe(CPE,SonsPEs,SonsCurrentPEs).
define_sons_current_pe(CPE,[[PE|_],[],[],[]],[SPE1,SPE2,SPE3,SPE4]):-
    SPE1=PE,SPE2=CPE,SPE3=CPE,SPE4=CPE.
define_sons_current_pe(CPE,[[PE1|_],[PE2|_],[],[]],[SPE1,SPE2,SPE3,SPE4]):-
    SPE1=PE1,SPE2=PE2,SPE3=CPE,SPE4=CPE.
```

5　負荷分散法の解析

（1）問題の定義

PrBPT(R,N)で、高さN、各親ノードがR個の子ノードを持つ完全バランス木を表す。leaf(PrBPR(R,N))でPrBPT(R,N)の末端ノードの総数を表す。no(PEs)でPrBPT(R,N)をHRCTLアルゴリズムによって解く並列プロセッサの総数とする。

（2）問題の仮定

PrBPT(R,N)の全てのノードは同一の計算負荷を持つ。プロセッサ間の通信は計算負荷0で、瞬時に可能である。任意の仕事のPE上での計算時間は、その仕事の計算負荷量に比例する。

（3）負荷分散法の定理

仮定の下では、HRCTLはPrBPT(R,N)に対してO(log(leaf(PrBPT(R,N))))あるいは

SubLFName::=当該スライスをレイアウトするのに用いたレイアウトフレームの名称

Type::={Element,Proc,Kind,Attribute,Prop}

Element::=block/dummyblock/deadspace/transistor/resistor/capacitor

Proc::=プロセス名称。例えばprocess100

Kind::=npn/pnp/br/hr,拡散層接合タイプ

Attribute::=標準セル名称。例えばnpn_tr(1)

Status::=free/place_fixed/fixed,当該回路ブロックが固定形状か可変形状かを指定する。

CProc::={Area,DeadSpace,Aspect}

Father::={FName,FMStr}

FName::=親回路名称

FMStr::=親回路へのストリーム

Sons::={LuSon,LbSon,RbSon,RuSon}

XXSon::={SMStr}/void,子回路へのストリーム

Nets::=[{NetName,PropList}¦_]

NetName::=ネット名称

PropList::=[{PartsName,LayerList}¦_]

PartsName::=collector/base/emitter/anode/cathode/void,接続物総称

LayerList::=[1,2¦_],アルミ配線層番号

Conts::=[{Net,RPattern,NCont,WCont,SCont,ECont,InContList}¦_]

RPattern::={lu_lb_v_v,N1,N2,signal}

XCont::={CPStr}/void,コネクタプロセスへのストリーム

Layers::=[1,2¦_],アルミ配線層番号

Vector::={VPStr}/void/{_,_,_,_}

MazeRouter::={MPStr}/void

Neighbour::=[N,W,S,E]/void

4．3．4　コネクタデータ

CLData::={Name,Net,Range,Type,Layers,NLine,WLine,SLine,ELine,Prop,RangeFlag}

Name::=コネクタ名称

Net::=ネット名称

Range::={Xfrom,Yfrom,Xto,Yto},コネクタ設定範囲

Type::={Element,Proc,UnitCellName,Prop}

Element::=edge/junctor/{transistor/resistor,JoinType}

Proc::=プロセス名称

UnitCellName::=pad/through_hole

３４１個、５８４個、９０１個である。

トランジスタや抵抗素子の形状はシンプルな形のものを使用した。配線については配線可能層数を３層とし、１、２層を信号配線および局所の電源配線に使用し、３層を電源配線専用層に設定した。

６．２．３　実験環境

multi-PSI－６４ＰＥ版を用いて実験した。６４ＰＥのうち、１６ＰＥ、３２ＰＥ、４８ＰＥ使用についても実験し、ＰＥ台数効果の測定を行った。

６．３　結果と考察

１０１９素子の回路を異なる目標形状のもとでレイアウトさせた結果を図６－１（１），(２)に示す。計算速度を図６－２（１），（２）に示す。ＰＥ台数効果を同図（３）に示す。Co-HLEXの大きさを同図（４）に示し、以下、考察を行う。

６．３．１　並列レイアウト問題解決の可能性

実験を通して、適正なレイアウトが得られることを確認した。多くの部分問題に分割して解いたにもかかわらず、並列協調計算によってブロック間で形状と配線の整合が実現できたことが図６－１から判る。これは、世界的にみても新たなものである。

６．３．２　レイアウト品質

図６－１を観察すると、ほぼ良く詰まった配置と屈曲の少ない配線により実現できたことが判る。通常、ブロック間の配線はチャネルエリアを設けて行うが、ブロック間協調処理によるコネクタ位置整合機能が働いて配線チャネルが不要となった。

６．３．３　外形状変更への柔軟な対応

図６－１（１）は横長領域への配置配線であり、同図（２）は縦長領域への配置配線である。これにより、チップの目標外形状の変更にも柔軟に対応できることを確認した。

６．３．４　計算時間

図６－２（１），（２）にmulti-PSI上でのCo-HLEXと汎用大型計算機上の逐次処理による実用レイアウトシステムの比較を示す。Co-HELXの計算オーダは規模に応じてO(N^1.0)である。Nは回路内素子数である。

実験による1019素子のレイアウトを約１５分で得ることができた。従来法に比較して、きわめて高速である。

```
top_module([{net,Name,Type,Neighbours,Props}|TMS],OutStr,ToModules,FmModules,TM):-
    module(ToModules,[],FmNeighbours,{Name,Type,Neighbours,Props,Others,[],ToTop,
     ToNeighbours,Work}),
    TM = {ToTop,TM1},
    top_module(TMS,OutStr,ToModules,FmModules,TM1).
```

4．4．2　素子形状整合化機能

回路素子の配置処理において、素子形状が不揃いだと空きエリアが多く発生し、チップ面積の増加につながる。これを防ぐために抵抗体や容量のように分割可能な素子を、分割不可能なトランジスタの標準的な高さに一致させるように分割する。トップモジュールから上記で生成した全回路ネットモジュールにadjust_shapeメッセージを放送し、分割を実施させる。

素子のモジュールは分割可能であれば自分自身を分割し、分割素子間を接続するネットモジュールを生成する。

分割できる素子は抵抗や容量で、第1節は1セグメントで実現できる抵抗素子の場合であり、放送を無視し、何もしない。第2節は2セグメントで実現する場合である。分割生成される新素子サイズは、標準素子サイズ(W,H)以下とする。分割した場合は、素子のモジュールと、接続配線のためのネットのモジュールを回路ネットモジュールのプロセスとして生成する。

```
top_module([{adjust_shape,W,H,Constr,end-Eo}|TMS],OutStr,ToModules,FmModules,TM):-
    ToModules = [{adjust_shape,W,H,Constr}|RMS],
    top_module(TMS,OutStr,RMS,FmModules,TM).
module([{adjust_shape,Proc,W,H,Constr}|FBrt],FmBelow,FmNets,
          {Name,Type,Nets,Props,Others,ToBroad,ToTop,ToNets,Work}):-
    Type = resistor,
    serial_division(Proc,W,H,Constr,Props,1,Props) |
    module(FBrt,FmBelow,FmNets,{Name,Type,Nets,Props,ToBroad,ToTop,ToNets,Work}).
module([{adjust_shape,Proc,W,H,Constr}|FBrt],FmBelow,FmNets,
          {Name,Type,Nets,Props,Others,ToBroad,ToTop,ToNets,Work}):-
    Type = resistor,
    serial_division(Proc,W,H,Constr,Name,Type,Nets,Props,2,
          [{Name1,Type1,Nets1,Props1},{Name2,Type2,Nets2,Props2}],[AddNet]) |
    MD1 = {Name1,Type1,Nets1,Props1,Others1,[],ToTop1,ToNets1,Work1}),
    MD2 = {Name2,Type2,Nets2,Props2,Others2,[],ToTop2,ToNets2,Work2}),
    MD3 = {AddNet,net,[Name1,Name2],Props3,Others3,[],ToTop3,ToGates,Work3}),
    ToTop = {ToTop1,ToTop2,ToTop3},
    module(FBrt,[],FmNets1,MD1),
```

索アルゴリズムの概要を示し、並列推論マシン上での実現について述べる。７．４節では、電子回路等への応用例について、回路図やレイアウト図のグラフ表現と構造認識例を中心に、それぞれ述べる。

７．２　部分グラフ認識のための探索方式

７．２．１　対象グラフと問題

図７－１のような、９個のノードとノード間を結ぶ１５個の経路からなるネットワークを考える。ｎ１～ｎ９はノード、各ノード間の線分は経路を表わす。各ノードには、その性質を表わす属性を持たせる。各ノードの下の数字は、それぞれのノードの属性を示す。ここでは属性として、仮に、ｎ１、ｎ２、ｎ６、ｎ８に「１」を、ｎ５、ｎ７に「２」を、ｎ３、ｎ４、ｎ９に「３」を、それぞれ与えた。例えば、ｎ４に注目すると、ノードの属性は「３」で、隣接ノードへの経路はｎ２、ｎ３、ｎ５、ｎ７、ｎ８への５個である。同様にｎ８では、属性「１」で、経路はｎ４、ｎ５、ｎ７、ｎ９への４個ということになる。このネットワークに対して、属性の並び（以下、目標リストと呼ぶ）を与えると、ネットワークをたどって解を探索し、該当する属性を持つノードの並びを返す問題を考える。例えば、”属性が「１」－「２」－「３」の並びのノードの連鎖はどれか？”という質問に対しては、ｎ１を始点に選ぶと、ｎ１の属性は「１」なので、次にｎ１に隣接するノードに、属性が「２」のものがあるかどうかを調べる。ｎ１の隣接ノードはｎ２、ｎ７で、そのうち属性「２」のものは、ｎ７のみであるので、次にｎ７の隣接ノードについて同様に調べる。属性「３」のものはｎ４とｎ９である。このようにして、ｎ１－ｎ７－ｎ９とｎ１－ｎ７－ｎ４という２個のノードの連鎖が答として求まる。ｎ２を始点に選ぶとｎ２－ｎ７－ｎ９、ｎ２－ｎ７－ｎ４が、ｎ６だとｎ６－ｎ５－ｎ４、ｎ６－ｎ５－ｎ３が、それぞれ解となる。また、ｎ８を始点とした場合は、ｎ８－ｎ７－ｎ９、ｎ８－ｎ７－ｎ４、ｎ８－ｎ５－ｎ４、ｎ８－ｎ５－ｎ３の４個の連鎖が解となる。つまり、全解が必要な場合は、これら１０個の解をすべて求めなければならない。そのためには、ネットワーク上のすべてのノードを始点ノードとして起動し、各ノードの隣接ノードすべてについて調べなければならない。

以上、問題をまとめると、少なくとも一つ以上の属性を持つノードから構成されるネットワークに対して、目標リストを与え、それに合致する属性を持つノードの連鎖を探索するというものとなる。

７．２．２　探索方針

７．１節で述べた問題を解決するための探索アルゴリズムを考える上で、次の点を考慮する。

（１）全解探索が可能なものとする。

```
module([{make_module_net}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToTop,ToNeighbours,Work},
    Type = block |
    ToBroad = [{make_module_net}|TBrt],
    MD1 = {Name,Type,Neighbour,Props,Others,TBrt,ToTop,ToNeighbours,Work},
    module(FBrt,FmBelow,FmNeighbour,MD1).
otherwise.
module([{make_module_net,end-Eo}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToTop,ToNeighbours,Work} |
    merge(NeigIn,FmNeighbour),
    send_join_message(hand_in_hand,Name,Neighbour,NeigIn,ToTop,ToTop1),
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,ToTop1,ToNeighbours,Work},
    module(FBrt,FmBelow,FmNeighbour,MD1).
```

自回路ネットモジュールへの入力側の回路ネットストリームを、NeigNameで示す相手側回路ネットモジュールの持つ隣接出力ストリームへ登録させる要求を、上位回路モジュールに送出する。

path_costは、回路上の素子位置を調べるための各回路ネットモジュールでの歩数を決めるもので、モジュールの属性に応じた値をあらかじめ設定してあるものを取り出す。

```
send_join_message(hand_in_hand,_,[],NeigIn,ToTop,ToTop1):-
    NeigIn = [], ToTop1 = ToTop.
send_join_message(hand_in_hand,ModName,[Neighbour|Nt],NeigIn,ToTop,ToTop1):-
    NeigIn = {ToNeig,NeigInT},
    Neighbour = {Attr,NeigName},
    path_cost(NeigName,Cost),
    ToTop = [{hand_in_hand,NeigName,{{ModName,Attr},Cost,ToNeig}}|ToTopT],
    send_join_message(hand_in_hand,ModName,NT,NeigInT,ToTopT,ToTop1).
```

回路モジュールが配下送信ストリームにより、隣接出力ストリーム登録要求を受信
受信した要求メッセージを全配下回路ネットモジュールにそのまま放送する中継処理を行う。

```
module(FmBroad,[{hand_in_hand,NeigName,OutMes}|FBet],MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToTop,ToNeighbours,Work} |
    ToBroad = [{hand_in_hand,NeigName,OutMes}|TBrt],
    MD1 = {Name,Type,Neighbour,Props,Others,TBrt,ToTop,ToNeighbours,Work},
    module(FmBroad,FBet,MD1).
```

node([n2,n3],1,n1)

のように書ける。同様に、ｎ２、ｎ３、ｎ４も

node([n1,n4],2,n2)

node([n1,n4],2,n3)

node([n2,n3],3,n4)

と書ける。

これらをマルチＰＳＩ上のプロセスとしてハッシュプールに登録し、各プロセス間を隣接ノードリストを参照して双方向ストリームで結合する。始点ノードに、目標リストを含んだメッセージ（以下、探索メッセージと呼ぶ）を送り、隣接ノードへのストリームに次々に探索メッセージを送ることにより探索を進める。探索アルゴリズムについては次節で述べる。始点ノードとして複数のノードを指定する場合は、該当するノードすべてに探索メッセージを送る。また、全ノードを始点ノードに指定することにより、全解を得ることができる。探索がすべての枝について終了したら、解を出力し、ノード間のストリームを閉じる。

７．３．２　探索アルゴリズムと解の表現

探索アルゴリズムを次の部分より構成する。

（１）すべての隣接ノードに、探索メッセージを送る。

（２）解を含む木構造データで、探索メッセージ内の変数を書き替えていくことにより、解を生成する。

（３）同一ノードを続けて２回以上通らないようにする。

（４）目標リストに一致する解がみつかったら、その旨マークを付け（ここでは＊とした）、その枝の探索を終了する。

（５）解になり得ないとわかった時点（目標リストの要素と属性とが一致しない場合）で枝刈りを行なう。変数を空リストで書き替えて、探索を終了する。

以上の点を考慮し、探索アルゴリズムを以下のステップより構成する。

ステップ１：　探索メッセージを受け取ったノードは、目標リストの先頭要素と自ノードの属性とを比較する。

ステップ２：　ステップ１で双方が一致したら、探索メッセージ内の変数を、隣接ノード数個の、自ノード名と新たな変数とを含んだ木構造データで書き替えて、ステップ４へ進む。

ステップ３：　ステップ１で双方が一致しなければ、探索メッセージ内の変数を、自ノード名と空リストとを含んだ木構造データで書き替えて、探索を終了する。

ステップ４：　隣接ノードに先頭要素を除いた目標リストを含む探索メッセージを渡して、さらに探索を進める。

ステップ５：　目標リストが空になったら、探索メッセージ内の変数を、自ノード名と解がみつ

{decide_sitting,Name}::=モジュール位置決定要求メッセージ

{regist_subblock,Name}::=登録分画決定要求メッセージ

{divide_cir_net,Name}::=ネット分割要求メッセージ

{gen_sub_tree,Name}::=配下分画ブロック回路モジュール生成要求メッセージ

{gen_tree_all,Name}::=配下分画ブロック内回路木生成要求メッセージ

第2節は、単数素子保有回路モジュールの場合であり、分画ブロックの回路モジュールから素子の回路ネットモジュールへの変更を行う。

```
module([{gen_module_tree,Namei,Nameo}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Namei,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Sons},
    or(Sons = void,Sons = [_,_|_]) |
    v_line_cost(Vcost),
    ToBroad = [{add_dir_net,Namei},
               {create_dir_net,Namei,n},
               {create_dir_net,Namei,w},
               {create_dir_net,Namei,s},
               {create_dir_net,Namei,e},
               {investigate_path,Namei},
               {kill_dir_net,Namei},
               {prepare_partitioning,Namei,Vcost},
               {decide_sitting,Namei},
               {regist_subblock,Namei},
               {divide_cir_net,Namei},
               {gen_sub_tree,Namei},
               {gen_tree_all,Namei}|TBt],
    Nameo = Namei,
    MD1 = {Namei,Type,Neighbour,Props,Others,TBt,ToHigher,ToNeighbours,[]},
    module(FBrt,FmBelow,FmNeighbour,MD1).
otherwise.
module([{gen_module_tree,Namei,Nameo}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Namei,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,[EName]}|
    ToBroad = [{report_module_data,EName,EMD},
               {kill,EName}],
    EMD = {EName,EType,ENeighbour,EProps,EOthers,EToBroad,EToHigher,
          EToNeighbours,EWork},
    NMD = {EName,EType,ENeighbour,EProps,EOthers,EToBroad,ToHigher,
```

自ノード名と解がみつかった旨を表わすマーク＊とを含んだ木構造データ t(n4,*) で書き替えて、この枝の探索を終了する。同様に、ｎ３から探索メッセージを受け取ったノードｎ１は、目標リストの先頭要素「３」と自ノードの属性「１」とを比較する。その結果、双方が一致しないので、探索メッセージ内の変数 Tree23 を、自ノード名と空リストとを含んだ木構造データ t(n1,[]) で書き替えて、探索を終了する。一方、ｎ３から探索メッセージを受け取ったノードｎ４は、目標リストの先頭要素「３」と自ノードの属性「３」とを比較する。その結果、双方は一致するが、先頭要素「３」を除いた目標リストは空リスト [] となるので、これが解の１つとなる。そこで、探索メッセージ内の変数 Tree24 を、自ノード名と解がみつかった旨を表わすマーク＊とを含んだ木構造データ t(n4,*) で書き替えて、この枝の探索を終了する（図７－２ (5) ）。これらの探索の結果、解を表わす木構造データは、

t(n1,[t(n2,t(n1,[])),t(n2,t(n4,*))]),t(n1,[t(n3,t(n1,[])),t(n3,t(n4,*))]) となる。 [] で終わっている経路は、途中で枝刈りされたものであり、解ではない。＊マークへ至る経路が解である。この例では、＊で終わる経路は２つあり、それぞれ根元から辿っていくと、ｎ１－ｎ２－ｎ４、ｎ１－ｎ３－ｎ４の２つの解が得られる。

以上の処理を行なった時の木構造データ生成の様子を図７－３に示す。最初に始点ノードを起動する探索メッセージ内に、解を求めるための変数を与える。この変数は、探索が進むにつれて書き替えられ、部分解を蓄えていく。この時、探索の途中経過情報はすべて探索メッセージ上に持たせ、ノードには持たせない。探索が深くなると、探索メッセージは重くなる。少しでも探索メッセージを軽くするために、解を木構造のデータとし、根元の方のデータを共有することとした。図７－３ (2)(3) のように、探索結果得られたノード名を第１引数に保持し、それ以降の探索のために第２引数に変数を持たせる。目標リストに一致する解がみつかったならば、変数を＊に書き替える。枝刈りを行なった場合は、この変数を空リストに書き替える。

以上の処理を、ｎ２、ｎ３、ｎ４についても行ない、全ノードを起動することにより、全解を求めることができる（図７－２ (6) ）。

上記の方式を、ノード数が、５×５（＝２５）、７×７（＝４９）、８×８（＝６４）、９×９（＝８１）、１０×１０（＝１００）、１２×１２（＝１４４）個の構成の正方形の格子状ネットワークに適用して実験した。７×７のネットワーク例を、図７－４に示す。目標リストとしては、 [1,2,3,4,5] を与え、全ノードを始点ノードとして起動した。その結果、全解が予定どおり求まっていることを確認できた。

７．４　回路図への応用例

回路ネットの表現は、原則として、一個の素子や端子に、一個のノードを対応させることとする。配線は、ノード間に双方向ストリームを張ることにより表現する。また、配線の分岐が生じている部分には、ダミーのノードを作ることとする。各ノードには、隣接ノードのリスト、属性、

```
MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work1},
module(FmBroad,FmBelowT,FmNeighbour,MD1).
```

4．5．3　仮想端点モジュール生成

N,W,S,Eの仮想端点（信号伝播経路分析入力点コネクタとそのネット）モジュールを生成し、既に準備された回路ネットモジュールとのストリーム接続を行う。

Inで表した変数は仮想端点ネットモジュールから仮想端点コネクタモジュールへの出力側回路ネットストリームである。Jnで表した変数は仮想端点ネットモジュールから仮想端点コネクタモジュールへの入力側回路ネットストリームである。Knで表した変数は仮想端点コネクタモジュールから仮想端点ネットモジュールへの出力側回路ネットストリームである。Lnで表した変数は仮想端点コネクタモジュールから仮想端点ネットモジュールへの入力側回路ネットストリームである。

仮想端点ネットモジュールからの回路ネットストリームとして、モジュールデータのワーク部に収集済みの出力側回路ネットストリームデータを加えることにより回路ネットストリームの形成を完了する。

図４－１（２）の４方向に記した端点が仮想端点コネクタモジュール(n,w,s,e)と仮想端点ネットモジュール(n_net,w_net,s_net,e_net)である。追加された回路ネットストリームを片方向矢印で示したが、これは信号伝播経路分析メッセージの仮想端点迂回を防ぐため、仮想端点ネットモジュールへの出力ストリームを持っていないためである。

以下、Nモジュールについて記述するが、W,S,Eについても同様に行う。

```
module([{create_dir_net,Name,Place}|FBrt],FmBelow,FmNeighbour,MD):-
    Place = n,
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,{N,W,S,E}},
    merge(In,Jn),
    merge(Kn,Ln),
    merge({ToHighern,ToHighernn,FmBelow},FmBelow1),
    MDn  = {n,{outward,n},[n_net],void,Othersn,[],ToHighern,
           [{{n_net,outward_net},0,Kn}],void},
    MDnn = {n_net,{outward_net,n},[n],void,Othersnn,[],ToHighernn,
           [{{n,outward},0,In}|N],void},
    module(ToBroad,[],Jn,MDn),
    module(ToBroad,[],Ln,MDnn),
    module(FBrt,FmBelow1,FmNeighbour,MD).
```

4．5．4　信号伝播経路分析

{dum11,node([q4,out2,q8],dum,dum11}

{dum12,node([r6,dum13,r5],dum,dum12)}

{dum13,node([r7,dum14,dum12],dum,dum13)}

{dum14,node([r8,gnd,dum13],dum,dum14)}

図７－６の上は、トランジスタと抵抗とが隣接している部分の認識例であり、図７－６の中は、トランジスタ － ダミーノード － トランジスタ という構造の認識例である。

よりきめの細かい認識を行なうために、１個のトランジスタを３つのノードの集合で表現することを考える。ｑ１を３ノードで表現した例を示す。

{q1_b,node([q1_c,q1_e,in1],tr_base,q1)}

{q1_c,node([q1_b,q1_e,dum5],tr_col,q1)}

{q1_e,node([q1_b,q1_c,dum8],tr_emi,q1)}

図７－６の下に、ペアトランジスタの認識例を示す。

8　全体的成果概要

8．1　研究の位置付け

ＬＳＩ技術の進歩と共に計算機アーキテクチュアは従来の逐次型から並列型に進化しつつある。日本が過去１０年余りに渡って進めてきた第５世代計算機プロジェクトは、その最先端を担ってきた。

逐次計算という制御を全ての前提としてきた２０世紀のプログラミング技術は、並列計算機の登場によって大きな変革を被るであろうと予想される。

計算可能性の理論からはＮＰ完全な問題が並列計算機の登場によって処理可能となるわけでは無いという先を見通した議論もあるが、実際の問題を処理させうる並列ハード・ソフト環境が乏しかったことから、その功罪についてのまじめな議論は困難であった。第５世代プロジェクトの成果である汎用並列処理環境によって、やっとこのことが可能になってきたといえる。

本研究の目的は、本格的並列処理プログラムを第５世代プロジェクトの生み出したハード・ソフトの上に実現し、それらの有用性を実証することにある。

8．1．1　取り挙げた問題

並列推論マシンの莫大な計算力に見合う問題として、規模の大きさ、処理の複雑さにおいて特に著名なレイアウト問題を取り挙げた。レイアウト問題は、俗に組み合わせ問題と呼ばれるものに属しており、空間内への物の配置や物の間の配線を作成する問題である。発電設備計画、都市計画、ビルや部屋の間取り、新聞への記事の割り付け、ＬＳＩの配置配線等、多くのレイアウト

信号伝播経路分析実行

信号伝播経路分析メッセージを受け、隣接回路ネットモジュールへのメッセージ伝播を行う。第1節は、信号伝播経路分析の収束処理である。ワーク部に自モジュールの各方向到着コストを保存する。

第2節は、信号伝播経路分析メッセージの到着コストが自モジュール内に保存しているものより大の場合で、新たな最短経路を発見できなかった場合である。

第3節は、新たな最短経路を発見した場合の処理である。到着コストを置換するとともに、隣接回路ネットモジュールに信号伝播経路分析メッセージを送出する。

第2節、第3節はN方向からの信号伝播経路分析処理について記述するが、他方向からについても同様に行う。

```
module(FmBroad,FmBelow,FmNeighbour,MD):-
    MD  = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work},
    Work  = {{_,NC},{_,WC},{_,SC},{_,EC},end} ;
    Work1 = {NC,WC,SC,EC},
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work1},
    module(FmBroad,FmBelow,FmNeighbour,MD1).
alternatively.
module(FmBroad,FmBelow,[reached(n,Sender,Cost,ForwardTo)|FNt],MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work},
    Work = {{D,C},W,S,E,End},
    Cost >= C ;
    ForwardTo = [],
    module(FmBroad,FmBelow,FNt,MD).
module(FmBroad,FmBelow,[reached(n,Sender,Cost,ForwardTo)|FNt],MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work},
    Work = {{D,C},W,S,E,End},
    Cost < C ;
    make_forwarding_list(Name,Sender,ToNeighbours,ForwardTo,ToNeighbours1),
    Work1 = {{Sender,Cost},W,S,E,End,SS},
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours1,Work1},
    module(FmBroad,FmBelow,FNt,MD).
```

隣接する回路ネットモジュールに信号伝播経路分析メッセージを送出

第1節は、全て送出し出力先がなくなった場合、第2節は、トランジスタのベースから、信号であるメッセージを出さないための処理、第3節は、メッセージ送出処理である。

し、探索の場合数を削減している。

（4）粗い評価関数

多数の可能解の中から最適案を選択する方法として、チップ面積、総配線長を評価メジャとしている。このため、配線通過路の細かい制御はむつかしい。

（5）計算モデル

レイアウト状態を記述したテーブルと、テーブルへのポインタから成るデータ構造と、これを書き換えるためのアルゴリズムとして実現される。

（6）ソフトウエア生産性

莫大な手続き型プログラムが開発され、使用されている。強度にデータ構造依存のアルゴリズムであり、変更、保守は難しい。

このような従来技術の難点を第5世代技術によって克服する可能性を探るため、基本的には次の方針に基づくシステム試作を行う。

（1）再帰算法

階層処理を用いたレイアウト方式とする。但し、任意段の階層を処理可能とする。階層の各ノードでは同一のプログラムを繰り返し利用する再帰原理を採用し、プログラム記述量を減らせる。

（2）並列協調処理

各部分問題は相互に並列に処理する。本来、ネットワーク構造である回路を、無理に木構造化するために生じるモジュール形状の不整合や、配線出し口の不整合を、モジュール間相互通信に基づく協調によって解消する。この試みは世界的にみても初めてのものである。

（3）ストリーム並列計算モデル

レイアウト問題を、局所メモリを持った動的プロセスとそれらの間を接続する通信ストリームとから成る計算モデルで表現する。レイアウトの作成はメッセージ送信に基づくプロセスの状態変更によって実現する。別言すれば並列オブジェクト指向プログラミング計算モデルによるレイアウトを行う。

（4）ソフトウエア生産性の向上

分散プロセスネットワークによる対象の記述は、トップダウン設計過程と相性がよい。さらに各プロセスの記述はメッセージに対する内部状態変更過程となり、記述のモジュラリティが高まる。これらの理由により、本研究では分散プロセスネットワークによる対象の記述を行う。

8．2．2　経緯

ＬＳＩのレイアウト問題解決システムを並列推論マシン上に実現するＩＣＯＴ１０年プロジェクトの前、中期7年間の成果は、後期3年間の当初から並列推論マシンmulti-PSIとして結実し、並列プログラミングの研究開発に利用できるようになった。そして最終目標であるPIMの本格的開発も開始された。このような事情を考慮して、以下の研究開発方針とした。

```
    ToModule = [],
    forward_message(Compass,C0,MaxCost,F,T0-T).
```

4．5．5　仮想端点モジュール消去

仮想端点モジュールは、信号伝播経路分析のために設けた仮想モジュールであるため消去する。

```
module([{kill_dir_net,Namei}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work},
    or( Type = {ourward,_},Type = {outward_net,_} ) !
    ToBroad = [],
    close_path(ToNeighbours),
    module([],[],[],MD).
```

4．5．6　回路分割準備

modify_path_value/3は{N,W,S,E}の信号到着値を修正するもので、素子モジュールの電源線経由の信号到着値が修正される。

電源線を通過するコストはVcostであり、電源コネクタ点からの信号到着値は底上げされているため、正規（0から）の値に修正する。修正信号到着値は、保持信号到着値とVcostの剰余を求め、正規に得られている信号到着値の最大値を加えた数である。Vcostは開始値を0としたとき得られる信号到着値の最大値を越える値である。あらかじめ大きな値を与えたのは、電源線への信号パスの回り込みを防ぐためである。

shape_to_type/3は外形状の縦横比により、分画タイプを決定しておく。分画タイプは、上下左右4分割、横並び2分割、縦並び2分割とする。

また、上位回路モジュールに修正信号到着値を報告する。

図4－1（1）に回路分割準備後の素子モジュールが持つ修正信号到着値を示す。

```
module([{prepare_partitioning,Name,Vcost}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,{N,W,S,E}},
    object(Type) , not_connector(Type) !
    modify_path_value({N,W,S,E},Vcost,{Npv,Wpv,Spv,Epv}),
    ToHigher = [{report_path_value,{Npv,Wpv,Spv,Epv}}|TTpt],
    shape_to_type(Shape,Type,Type1),
    MD1 = {Name,Type1,Neighbour,Props,Others,ToBroad,TTpt,ToNeighbours,
          {Npv,Wpv,Spv,Epv}},
    module(FBrt,FmBelow,FmNeighbour,MD1).
```

配下回路ネットモジュールから送られた信号到着値の最大値を求め保存する。

元年度開発のCo-HLEX第１版のアーキテクチュアをデータ、プロセスともに分散化し、並列処理性能を高めたCo-HLEX第２版を開発した。また、入力されるネットワーク形式のデータから、Co-HLEXで受理する階層形式のデータ（回路木）を自動生成する機能を実現した。
レイアウト時に利用する平面分割パターンライブラリ等を充実し、異形状物のレイアウト、配線層の多様化等、レイアウト機能を若干向上させた。

（４）平成３年度

２年度開発のCo-HLEX第２版の構成要素である配線機能を向上させるために、レイアウトルールの厳密な遵守が可能なセル内配線の迷路法化や電源配線機能の追加等を行うとともに、各種サブシステムを有機的に接続し、Co-HLEX第３版を開発した。
バイポーラ・アナログ回路を実験用サンプルとして使用し、レイアウト機能・性能の確認を行った。

８．３．２　成果

前期研究開発を通して下記の成果を得た。

（１）レイアウト問題解決に於ける基礎的知識表現技術の開発

ESP言語の持つオブジェクト指向表現能力を利用して、計算機室、配置機器、あるいは機器配置ルールなどの意味属性の計算機内表現と問題解決での利用技術を開発した。

（２）レイアウト問題解決に於ける並列処理技術の原理開発

ESP言語の持つ擬似並列処理能力を利用して、計算機室をいくつかのゾーンに分割し、それぞれを原問題の部分問題として擬似並列処理させる技術を開発した。ただし、問題解決に於ける階層は固定２段であり、部分問題間の強調処理も初歩的なものであった。

（３）システムの段階的開発過程に於けるオブジェクト指向パラダイムの有効性実証。

中期の開発では、年度毎にシステムを段階的に構築した。研究である為、システムの最終イメージは最初からは与えれないという条件のなかで、インクリメンタルなシステム開発が必要であった。例えば、配置機能を持つシステムへの配線機能の追加や、問題解決モードの逐次処理から並列処理への移行がその例である。定義済みオブジェクトへのスロットやメソッドの追加、継承の変更などによって、従来のプログラミングパラダイムでは多くの困難を伴うインクリメンタルなシステム開発が可能であった。

後期研究開発を通して下記の成果を得た。

（１）ＬＳＩレイアウトに関する階層再帰並列協調新算法HRCTLの開発

長方形区画の高々４個の分画への分割様式とそれらの分画の間の配線経路群とを保持したレイアウトフレーム群をあらかじめ用意しておき、これを用いて階層記述された回路を目標チップ形状内に再帰的に埋め込むことによって配置・配線を生成するという階層再帰形式のレイアウト方式において、分割によって得られた部分問題を相互に並列に走行させるとともに、部分問題間の通信によって相互間の配線を行うという新しいレイアウト算法を開発した。

```
    ToHigher = [{get_path_range,[Nmax,Wmax,Smax,Emax]},
                {report_sitting_point,[I,J]}|TTpt],
    MD1 = {Name,Type,Neighbour,Props,Others1,ToBroad,TTpt,ToNeighbours,Work},
    module(FBrt,FmBelow,FmNeighbour,MD1).
```

回路モジュールは配下回路ネットモジュールの要求により、自モジュール保存の信号到着値の範囲取得結果を取りだす。

```
module(FmBroad,[{get_path_range,PathRange}|FBet],FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work},
    get_path_range(Others,PathRange),
    module(FmBroad,FBet,FmNeighbour,MD).
```

回路モジュールは配下回路ネットモジュールの持つインデックス値の報告を受信し、インデックス値の最大値を求めて最大存在範囲として保存する。

```
module(FmBroad,[{report_sitting_point,[I,J]}|FBet],FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work},
    get_sitting_point(Others,[Imax,Jmax]),
    max(I,Imax,Imax1),
    max(J,Jmax,Jmax1),
    set_sitting_point(Others,[Imax1,Jmax1],Others1),
    MD1 = {Name,Type,Neighbour,Props,Others1,ToBroad,ToHigher,ToNeighbours,Work},
    module(FmBroad,FBet,FmNeighbour,MD1).
```

4．5．8　登録分画決定

インデックス値の範囲を取り出し、これをサイズとした2次元配列を縦横各等2分割し、左上、左下、右下、右上の区画を仮定する。モジュールの持つインデックス値(I,J)がどの区画を指すか調べ、分画名を決定する。

図4－1（1）に、モジュールプロセスの登録すべき分画名を示す。

```
module([{regist_subblock,Namei}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work},
    object(Type), not_connector(Type),
    get_sitting_point(Others,[I,J]),
    ToHigher = [{get_site_range,[Imax,Jmax]}|TTpt],
    Ihalf := Imax/2,
    Jhalf := Jmax/2,
```

マルチPSIおよびPIMは世界で初の並列推論マシンであり、種々の並列アルゴリズムの開発環境として国内外から利用できるようになることが望まれる。本来、非同期並列の事象を基本要素とする現実世界の表現には並列処理の相性がよく、丁度２０世紀においてノイマン型逐次処理が果たしたような役割を２１世紀において並列処理が担うのではないだろうか。人間の思考と計算機の知能レベルとの間のセマンティックギャップを少なくすることがソフトウエア科学の最も重要な課題であると言われていることでもあり、現代の最も代表的な難問の一つであるＬＳＩレイアウト問題解決過程の簡潔な並列再帰記述をひとまず得ることができたことと、この記述をそのまま解釈して実際にレイアウトを生成する並列推論マシンが身近に実在することは、この期待への確信度を相当程度増してくれる胎動のように思えるのである。

参考文献

[Poincare 02] Poincare,H.,La Science et l'Hyopothese (1902).

[Kleene 52] Kleene,S.C.,Introduction to Metamathematics (Van Nostrand,1952).

[Mandelbrot 82] Mandelbrot,B.,The Fractal Geometry of Nature (W.H.Freeman,1982).

[Six 86] Six,P.,Claesen,L.,Rabaey,J.,and De Man,H.,An Intelligent Module Generator Environment,Proc.23rd DAC,pp730-735 (1986).

[Watanabe 89] Watanabe,T. and Shinich,H.,Modelling Layout Problem Solving in Logic,in CAD Systems using AI Techniques,G.Odawara(ed.),pp181-188 (North-Holland, 1989).

[Kowalski 79] Kowalski,R.A.,Logic for Problem Solving (North-Holland,1979).

[Sterling 86] Sterling,L. and Shapiro,E.,The Art of Prolog (The MIT Press,1986).

[Shapiro 87] Shapiro,E.(ed.),Concurrent Prolog:Collected Papers,(Vols.1 and 2) (The MIT Press,1987).

[Samet 80] Samet,H.,Region Representation:Quadtrees from Boundary Codes, C.ACM,Vol.23,No.3,pp163-170 (1980).

[Samet 84] Samet,H.,The Quadtree and Related Hierarchical Data Structures, Computing Survey,Vol.16,No.2,pp187-260 (1984).

[Otten 82] Otten,R,Automatic Floorplan Design,Proc.19th DAC,pp261-267 (1982).

[Luk 86] Luk,W.K.,Tang,D.T.,and Wong,C.K.,Hierarchical Global Wiring for Custom Chip Design,Proc.23rd DAC,pp481-489 (1986).

[Robinson 79] Robinson,J.A.,LOGIC:Form and Function (Edinburgh University Press,1979).

[Chang 73] Chang,C.L. and Lee,R.C.T.,Symbolic Logic and Mechanical Theorem Proving (Academic Press,1973).

[Lloyd 84] Lloyd,J.W.,Foundations of Logic Programming (Springer-Verlag,1984).

```
    number_of(BlockNames,BlockNo),
    or( BlockNo = 1 ->
        set_block_name(MD1,BlockNames,MD2),
        module(FBrt,FmBelow,FmNeighbour,MD2) ,
      BlockNo > 1 ->
        module({{take_net_outlet,Name,PlaceNames,OutletNames},
                {delete_outlet,Name},
                {create_outlet,Name,FBrt,OutletNames},
                {divide_net,Name,BlockNames,Placis,FBrt}},MD1)
    ).
```

接続された素子モジュールに近隣調査メッセージを送り、所在位置名と素子名を返答させ、所在位置毎に素子名を収集する。

```
report_element_place([],Pls,Placis,ToNeighbourso):-
    Placis = Pls,
    ToNeighbourso = [].
report_element_place([{P1,P2,ToPath}|TNit],Pls,Placis,ToNeighbourso):-
    ToPath = [{rep_place,Place,Name}|TPt],
    ToNeighbourso = [{P1,P2,TPt}|TNot],
    add_name_for_place(Place,Name,Pls,Pls1),
    report_element_place(TNit,Pls1,Placis,TNot).
```

所在別素子名設定変数を調べ、所在位置名を得る。

```
take_place_name(N,N,PlaceElems,NamesGuide,PlaceNames):- PlaceNames = [].
take_place_name(N,M,PlaceElems,NamesGuide,PlaceNames):-
        vector_element(PlaceElems,N,[]) !
    take_place_name(~(N+1),M,PlaceElems,NamesGuide,PlaceNames).
otherwise.
take_place_name(N,M,PlaceElems,NamesGuide,PlaceNames):-
        vector_element(NamesGuide,N,Name) !
    PlaceNames = [Name|PNt],
    take_place_name(~(N+1),M,PlaceElems,NamesGuide,PNt).
```

ネット分断コネクタモジュール生成

回路分割により発生するネット分断点となる分画の外郭上に、コネクタモジュールを生成する。

付録

並列協調処理を用いた自動レイアウト実験システム

1．緒言

ＩＣＯＴで開発されたＥＳＰ言語は、オブジェクト指向表現機能による、事物やそれらの間の関係の記述能力、論理型知識表現による推論能力、擬似並列問題解決の実現などの点で、レイアウト問題解決に有効である。特に、従来の最適化手法では取扱いの難しい非数値の意味的制約条件を持つ問題の表現や解探索の効率化に有効と期待されている。ここでは計算機室レイアウト問題解決へのＥＳＰ適用実験について紹介する。

2．並列協調レイアウト問題解決のＥＳＰによる実現

2．1　問題説明および問題解決アプローチ

計算機システムを構成する各種の機器を計算機室に配置すると共に機器間のケーブル配線を行なう。考慮すべき意味的制約条件として、次のようなものを取り上げる。

（ｃ１）機器に応じて出入り口との距離を考慮する。

（ｃ２）類似機器は近くに配置する。

（ｃ３）機器はＣＤ（コンソール装置）に向ける。

（ｃ４）機器間距離の制約を満たす。

（ｃ５）機器の前面線をそろえる。

（ｃ６）機器の保守エリアを確保する。

（ｃ７）ＦＤ（フロッピーディスク装置）等はＣＤの近くに配置する等により使用の便宜を図る。

以下の階層問題解決アプローチを採用する。

（ｓ１）上位階層で問題分割を行なう。機器のグループ化の後、部屋をゾーン分解し、グループのゾーン割り当てを行なう。機器間のケーブル配線よりグループ間配線を調べ、ゾーン境界に仮端子を設ける。全体の問題は各ゾーンでの機器配置および機器や仮端子の間の配線問題からなる部分問題に分割される。

（ｓ２）下位階層で配置制約条件を考慮して部分問題を並列に解く。異なる部分問題の間に、機器の距離や方向に関する制約が存在するため、部分問題間での協調が必要となる。協調の方式として本システムでは二方式を用いた。

（ｃｏ１）他の部分問題の解決を待って後、制約を満たす配置を作る。

（ｃｏ２）他の部分問題の解に応じた制約満足化戦略をあらかじめ各部分問題に持たせておき、

分画外郭上コネクタおよび、分画内素子の隣接出力登録データを取り出し、指示ネット名向けのストリームを新たなストリーム変数と置換する。

```
replace_bname_path(BN,Placis,ToNeighbours,Name,FmPath,ToNeig,OtherPath):-
    get_block_objs(BN,Placis,Objs),
    purge(Objs,ObjNames),
    replace_bname_path_sub(BN,ObjNames,ToNeighbours,Name,FmPath,ToNeig,OtherPath).

replace_bname_path_sub(_,[],ToNeighbours,Name,FmPath,ToNeig,OtherPath):-
    FmPath = [],
    ToNeig = [],
    ToNeighbours = OtherPath.
replace_bname_path_sub(BN,[Obj|Ot],ToNeighbours,Name,FmPath,ToNeig,OtherPath):-
    replace_bname_path_aux(BN,Obj,ToNeighbours,Name,ToNeig,ToNeigT,
    FmPath,FmPathT,RestPath),
    replace_bname_path_sub(BN,Ot,RestPath,Name,ToNeigT,FmPathT,OtherPath).

replace_bname_path_aux(_,_,[],_,ToNeig,ToNeigT,FmPath,FmPathT,RestPath):-
    ToNeigT   = ToNeig,
    FmPathT   = FmPath,
    RestPaath = [].
replace_bname_path_aux(BN,Obj,[ToNeighbours|Tt],Name,ToNeig,ToNeigT,
    FmPath,FmPathT,RestPath):-
    ToNeighbours = [ObjName,P1,OutPath],
    or( ObjName = [Obj,Attr] , ObjName = [BN,Attr] ) !
    FmPath = [FmPath1,FmPath2],
    OutPath = [[replace_out_path,Name,Attr,FmPath1]|OutPath1],
    ToNeig  = [[ObjName,P1,OutPath1]|TNt],
    replace_bname_path_aux(BN,Obj,Tt,Name,TNt,ToNeigT,FmPath2,FmPathT,RestPath).
otherwise.
replace_bname_path_aux(BN,Obj,[ToNeighbours|Tt],Name,ToNeig,ToNeigT,
    FmPath,FmPathT,RestPath):-
    RestPath = [ToNeighbours|RPt],
    replace_bname_path_aux(BN,Obj,Tt,Name,ToNeig,ToNeigT,FmPath,FmPathT,RPt).
```

回路外郭上の旧コネクタモジュールを消去する。

ゾーン境界近辺の仮端子間を配線し、配線全体を完成させる。

（ｏ６）形状処理オブジェクト

機器配置処理などで必要となる計算幾何学演算を実行する。

３．レイアウト実験

３．１　実験結果の一例

約４０個の機器からなる計算機システムのレイアウト過程を図３－１～図３－３に示す。５個の部分問題を並列処理している。協調モードｃｏ１での並列配置処理のタイムチャートを図３－４に示す。

３．２　結果の考察

制約ｃ１－ｃ７を満たす結果を得ている。協調モードｃｏ１では部分問題間の制約満足化のため待ちが発生し（図３－４のｗ）並列化効率の低下やデッドロック現象を招く。機器間距離の様に制約満足化処理に強い前後順序関係が無い場合には協調モードｃｏ２を用いて、後続機器側で制約を満たせることによりこれらの問題を解決できる。本システムではこの方式を用いて並列化効率を向上させている。

```
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work} ;
    get_place(Others,Place),
    Nameo = Name,
    module(FmBroad,FmBelow,FNt,MD).
```

回路ネットモジュールのうちネットのモジュールから回路ネットストリームを通じて素子のモジュールに要求され、指定されたネットモジュールへの隣接出力ストリームを閉じ、新たに渡されたストリーム変数と置換する。

```
module(FmBroad,FmBelow,[{replace_out_path,Namei,Attr,ToPath}|FNt],MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work} ;
    get_out_path(ToNeighbours,Namei,Attr,OutPath,RestToNeig),
    OutPath = {P1,P2,[]},
    NewToNeig = [{P1,P2,ToPath}|RestToNeig],
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,NewToNeig,Work},
    module(FmBroad,FmBelow,FNt,MD1).
```

４．５．１０　配下分画ブロックの回路モジュールプロセス生成

以下、４分割の場合の記述であるが、２、３分割の場合も同様に行う。

Lu,Lb,Rb,Ruブロックについて回路モジュールのプロセスを生成し、分画ブロック回路名、形状を求め設定する。

また、階層化したモジュール間での配下への放送および上位への送信ストリームの接続を行う。これは分画対応に、回路ネットモジュールの上位通信ストリームを自分画の回路モジュールのワーク部に設定させる要求と、回路ネットモジュールの放送受信ストリームを新規置換し分画ブロックの回路モジュールとの接続を行う要求を送出することで行う。

```
module([{gen_sub_tree,Name}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work},
    Type = {block,square} ;
    module(SubBroad,LuFmBelow,LuFmNeighbour,LuMD),
    module(SubBroad,LbFmBelow,LbFmNeighbour,LbMD),
    module(SubBroad,RbFmBelow,RbFmNeighbour,RbMD),
    module(SubBroad,RuFmBelow,RuFmNeighbour,RuMD),
    get_shape(Others,Shape),
    make_block_name(square,Name,[LuName,LbName,RbName,RuName]),
    make_block_shape(square,Shape,[LuOthers,LbOthers,RbOthers,RuOthers]),
    set_sons_name(Others,[LuName,LbName,RbName,RuName],Others1),
```

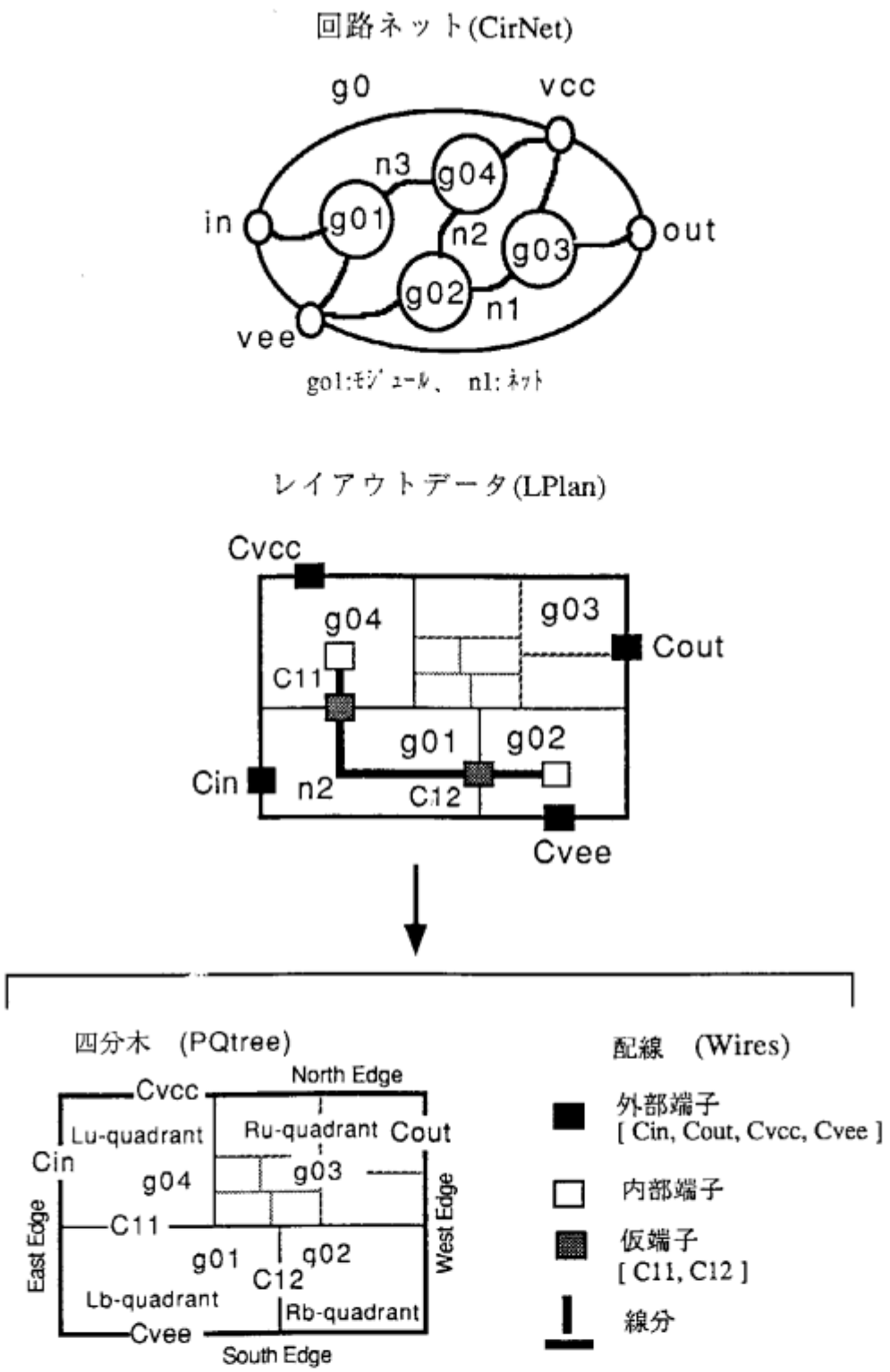


図 2-2 レイアウト状態の 4 分木表現

```
    make_others(ru,RuS,RuA,RuOthers).
```

回路ネットモジュールの上位送信ストリームを新しい変数と置換し、上位分画ブロックの回路モジュールに登録要求を送出。

```
module([{set_tohigher}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work} |
    get_block_name(Others,BN),
    ToHigher = [{link_upper,BN,NewToHigher}],
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,NewToHigher,ToNeighbours,Work},
    module(FBrt,FmBelow,FmNeighbour,MD1).
```

配下から送られた上位送信ストリームを、分画名の一致する分画の回路モジュールが受理し、ワーク部に追加登録する。
最後にToBelow1を閉じる必要があるが、本説明では省略する。

```
module(FmBroad,[{link_upper,BN,BelowToHigher}|FBet],FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,ToBelow},
    find_block_name(Others,BN) |
    ToBelow = {BelowToHigher,ToBelow1},
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,ToBelow1},
    module(FmBroad,FBet,FmNeighbour,MD1).
otherwise.
module(FmBroad,[{link_upper,BN,BelowToHigher}|FBet],FmNeighbour,MD):-
    module(FmBroad,FBet,FmNeighbour,MD).
```

配下回路ネットモジュールの放送受ストリーム置換

配下回路ネットモジュールの放送受信ストリームが、分画ブロックの回路モジュールの放送を受けるように、ストリーム変数の置換を行う。
この処理の実行により、上位の回路モジュール、分画ブロックの回路モジュール、回路ネットモジュールの階層間ストリームの形成が完了する。

```
module([{replace_fmbroad,FmBroads}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,work} |
    get_block_name(Others,BN),
    get_block_broadcast(FmBroads,BN,NewFmBroad),
    module(NewFmBroad,FmBelow,MD).
```

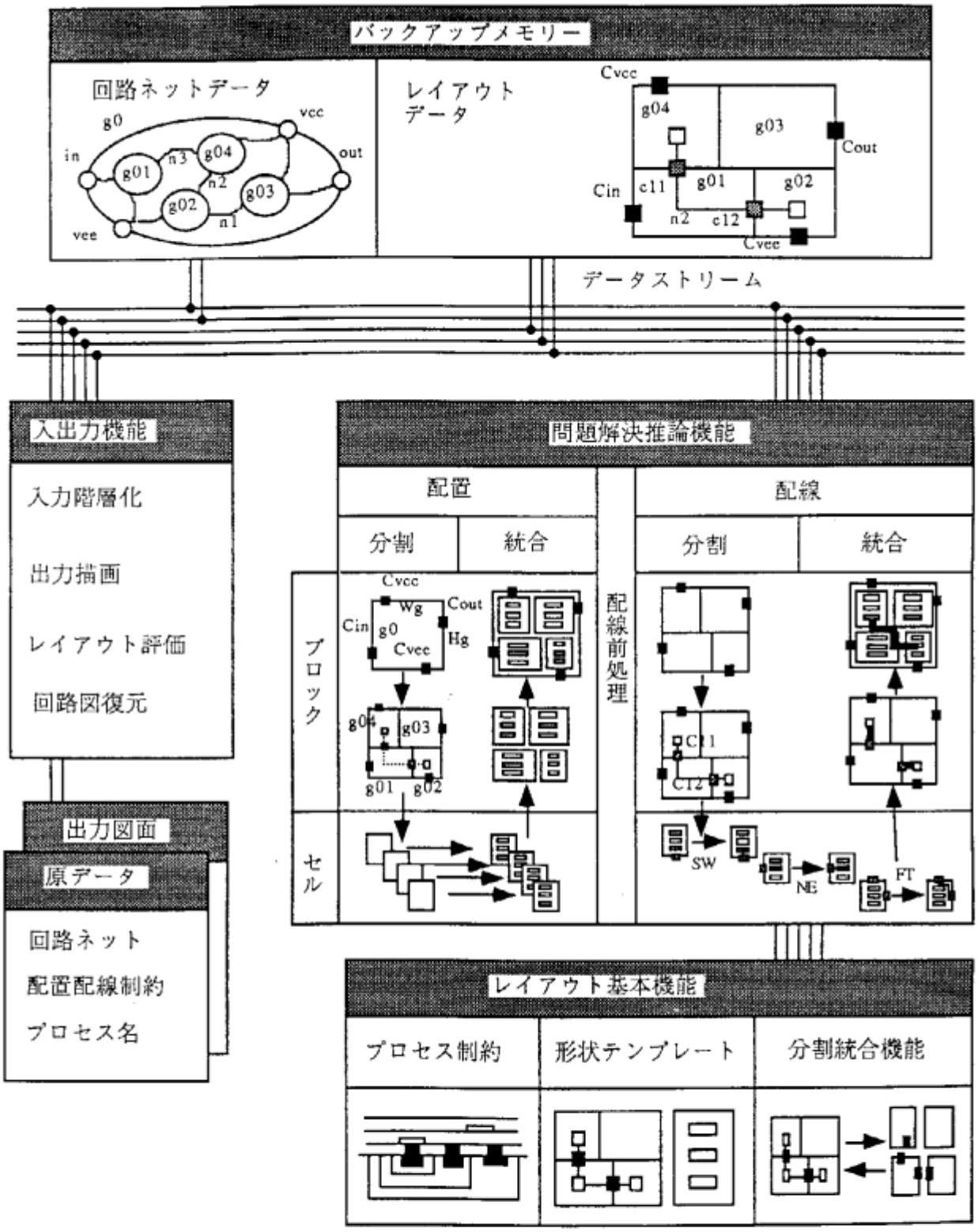


図 3-1 並列レイアウトシステム（Ｃｏ－ＨＬＥＸ）の構成

```
module(FBrt,FmBelow,FmNeighbour,MD1).
```

分画ブロックの回路モジュール配下の素子名称返送受信

分画ブロック配下の素子名をワーク部に収集する。

```
module(FmBroad,[{report_name,RepName}|FBet],FmNeighbour,MD):-
    MD  = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Sons} |
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,
          {RepName|Sons]},
    module(FmBroad,FBet,FmNeighbour,MD1).
```

4．5．12　順序性保存の手法

図4－2に終了監視ストリームを示す。

放送ストリームToBroadを通じ、終了フラグを持ったメッセージ{Message,EachArgument・・・,end-Eo}を全モジュールが受信する。

このとき、ショートスーパバイザであるSSモジュールがメッセージ内のendフラグをSSストリームに流し込み、リレー形式で回帰するendフラグをEoutに取り込み上位モジュールに戻すことで、全モジュールのメッセージ受信完了を通知する。SSモジュール以外の各モジュールは、SSストリームから1つEndフラグ変数を取り出し、実行すべき処理へ制御が渡った時点で、取り出したEndフラグ変数を次のSSストリームに与えることでendが巡回する。

当該プログラムでは、放送メッセージが全ての処理の始動を行うが、放送ストリーム、上位通信ストリーム、隣接モジュールストリームの3入出力ストリームを用いて処理を進める。

モジュールの受信する各々のストリームのメッセージには相互に順序性がないので、リレーすべきSSストリームのEndフラグ変数もメッセージに加えて送出し、処理終結モジュールがEndフラグ変数の送り出しを行うことで、モジュールは処理終了まで次の放送受信を待ち、順序性を保存する。

4．6　配置処理

Co-HLEXの基本アルゴリズムであるHRCTLの最上位が、配置機能である。回路モジュールをplace_by_qtreeメッセージによって最初に配置する。配置機能では、後述の配線機能の一部を用いたラフ配線をモジュールの位置決定時に行う。これにより、配線への考慮を配置時に行えるため、レイアウトの質を高めることができる。トップモジュールより最上位回路モジュールに配置実行を要求する。

```
top_module([{place_by_qtree,Proc,end-Eo}|TMess],OutStr,ToModules,FmModules,TM):-
    ToModules = [{place_by_qtree,Proc,end-Eo}|RMS],
```

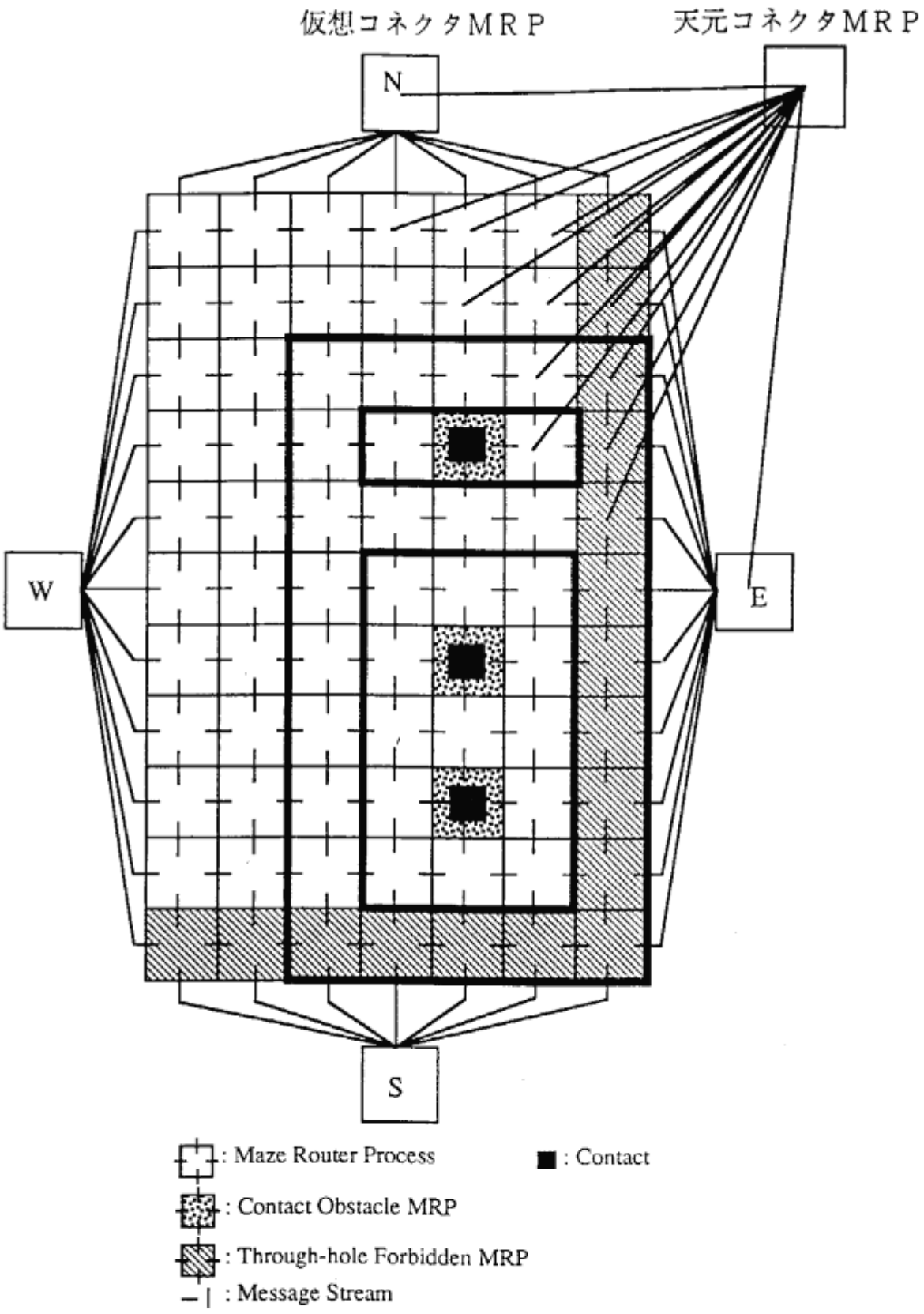


図 3-3 迷路法用プロセスネットワーク
の一例（トランジスタ）

４．６．２　レイアウトフレーム選択

回路モジュールの分割に使用するレイアウトフレームをテンプレートライブラリより選択する。try_all_layoutframes/5によって、可能な全てのレイアウトフレームとその区画への子回路の埋め込み順列に対して配置シミュレーションを行い、find_a_best_layoutframe/3によって、最適なレイアウトフレームとこれへの埋め込み順列における配置計画を選び出す。

```
module([[choose_a_layoutframe,Proc,end-Eo]|TMess],LM):-
    get_and_put(sons_stream,LM,SonsStr1,SonsStr2,LM1),
    get_and_put(parent_plan,LM1,OmyPlan,NmyPlan,LM2),
    get(sons_nets,LM2,SonsNets,NLM,end-Eo),
    separate_nets(NLM,NWSEnets),
    try_all_layoutframes(NWSEnets,OmyPlan,SonsStr1,SonsNets,TriedStr),
    find_a_best_layoutframe(TriedStr,NmyPlan,SonsStr2),
    module(TMess,NLM).
```

４．６．３　周辺コネクタ引き込み終了設定

finished_narrow_down/2は外郭上コネクタの持つrange-flagにfinishedというマークを付ける事によって、周囲のモジュールの当該コネクタの引き込みアクションを許す。

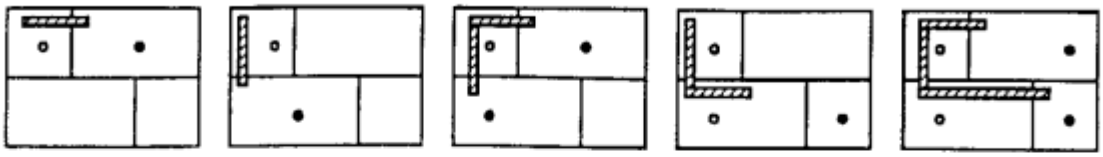
```
module([[narrow_leaf_conns,Proc,end-Eo]|TMess],LM):-
    get(peri_connectors,LM,PConns,NLM),
    finished_narrow_down(PConns,end-Eo),
    module(TMess,NLM).
```

４．６．４　部分問題生成

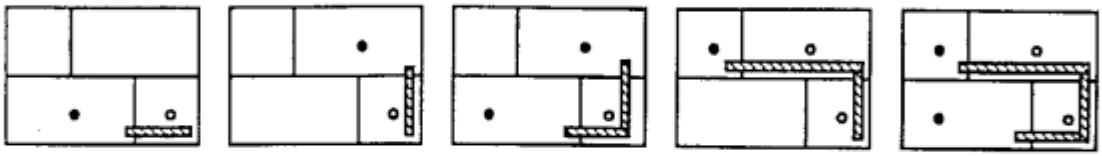
plan_wirings/6によって、選択したレイアウトフレームと、採用した子回路モジュールの各区画への埋め込み計画とを用いて子回路間のラフな配線計画を立て、ダミーコネクタを計画配線と区画分割する境界線との交点に設定する。propagate_plans/4は、それらを各子供に外郭上コネクタとして登録する。ここにおいて、各子回路モジュールは親の回路モジュールと同様に外形状の計画レイアウトを持つことになり、親の回路モジュールを処理したプログラムで再帰処理出来ることになる。

```
module([[generate_subproblems,Proc,end-Eo]|TMess],LM):-
    prepare_generate_sub(LM,NetsList,LM1,end-E1),
    plan_wirings(LM1,NetsList,LM2,NetsList1,Proc,E1-E2),
    propagate_plans(LM2,NetsList1,NLM,E2-Eo),
    module(TMess,NLM).
```

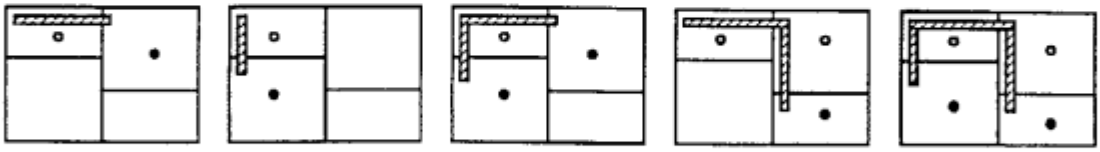
4h,posi



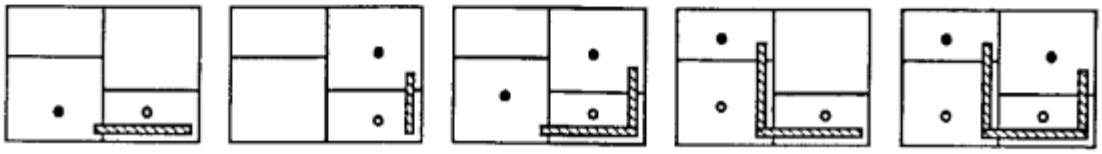
4h,nega



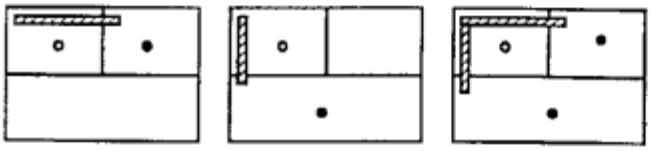
4v,posi



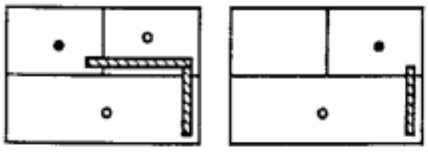
4v,nega



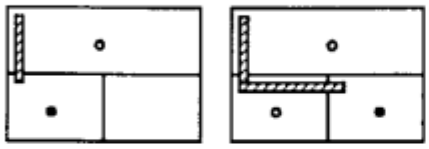
3h_uls,posi



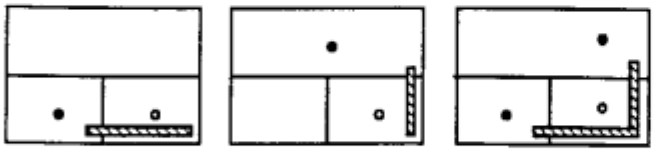
3h_uls,nega



3h_bls,posi



3h_bls,nega



● : posi/nega ネット有り

∘ : posi/nega ネット有りまたは無し

無印 : posi/negaネット無し

図3-4(2) レイアウトフレーム（電源配線パターン）

が送られてきた場合であり、配線結果をAnsNetに返す。narrowメッセージの時には第２、３節が働く。第３節は、配線不可能なフィードスルーネット（内部コネクタを持たない通過配線）の処理を行う。renew_cont_rangeflag/2は、外郭上コネクタの持つrange-flagにinspectedというマークを付けることによって、周囲のモジュールの当該コネクタの引き込みアクションを許す。同時にinspectedというメッセージをwire_manager/1に報告し、次回のnarrowメッセージを待つ。第２節は、配線可能な場合を扱う。外郭上コネクタをpull_in_connectors/4により適切な内部区画に引き込んで仮内部コネクタとした後、find_a_good_route_pattern/7によってネット保有区画間を結ぶ配線パターンを決定した後、回路モジュールの区画境界線上に仮想（ダミー）コネクタを設ける。

```
wire_a_net(_,[finished],Net,_,NWSEContsStr,_,_,VecStr,AnsNet,Ei-Eo):-
    AnsNet = Net,
    close([NWSEContsStr,VecStr],Ei-Eo).
wire_a_net(CName,[[narrow,Ans]|TailMess],Net,InNetProfile,NWSEContsStr,
    Shape,LFName,VecStr,AnsNet,Ei-Eo):-
    get_and_put(cont_range_flag,NWSEContsStr,NWSEConts,NWSEContsStr1,Ei-E1),
    wait_neighbours_action(CName,NWSEConts),
    or(exist_inner_net(InNetProfile),wieable_feed-through(NWSEConts)),!
    renew_cont_range_flag(narrowed,NWSEConts),
    get_and_put(vector,VecStr,[GV,IV,IC],[GV',NewIV,NewIC],VecStr1,E1-E2),
    planframe(LFName,get_route_patterns,RPatterns),
    pull_in_connectors(NWSEConts,Shape,[InNetProfile,IV,IC],NewProfileVec),
    find_a_good_route_pattern(RPatterns,NewProfileVec,GV,NewIV,NewIC,
                              RPattern,NewDummyConts),
    add_crossing_points_on_netdata(Net,NetDummyConts,RPattern,NewNet),
    Ans = finished,
    wire_a_net(CName,TailMess,NewNet,InNetProfile,NWSEContsStr1,
            Shape,LFName,VecStr,AnsNet,E2-Eo).
otherwise.
wire_a_net(CName,[[narrow,Ans]|TailMess],Net,InNetProfile,NWSEContsStr,
        Shape,LFName,VecStr,AnsNet,Ei-Eo):-
    get_and_put(cont_range_flag,NWSEContsStr,NWSEConts,NWSEContsStr1,Ei-E1),
    renew_cont_range_flag(inspected,NWSEConts),
    set_old_range(NWSEConts),
    Ans = inspected,
    wire_a_net(CName,TailMess,Net,InNetProfile,NWSEContsStr1,
```

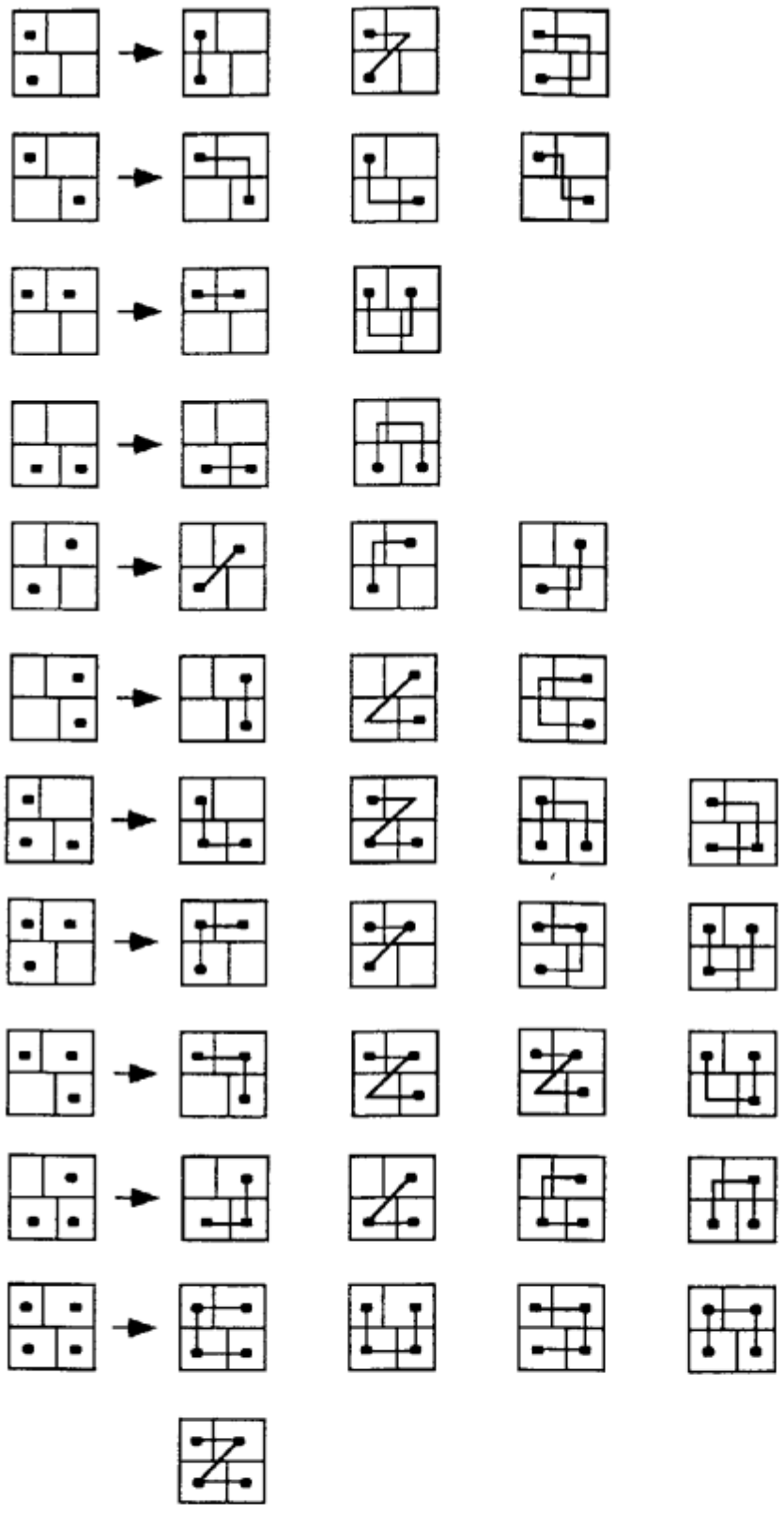


図3-4(3) レイアウトフレーム（配線経路パターン例）

```
                    InNetProfile,AnsInNetProfile):-
    have_narrow_range(ORange,Shape),
    determine_pull_in_slice(ORange,Shape,Slice,NRange) !
    renew_innet_profile(InNetProfile,Slice,Layer,NewInNetProfile),
    pull_in_connectors(Rest,Shape,NewInNetProfile,AnsInNetProfile).
pull_in_connectors([{{ORange,NRange},RFlag,Layer}|Rest],Shape,
                    InNetProfile,AnsInNetProfile):-
    not(have_narrow_range(ORange,Shape)),
    possible_pull_in_slice(ORange,Shape,PSlices) !
    possible_innet_profiles({{ORange,NRange},RFlag,Layer},InNetProfile,PSlices,
                        PossibleInNetProfiles),
    pull_in_connectors(Rest,Shape,PossibleInNetProfiles,AnsInNetProfile).
```

4．6．5　部分問題統合

レイアウトフレームに依存したplanframe/5によって、完成された子回路モジュールを統合して親の回路モジュールの形状を求めてRealShapeとする。トップダウンの計画中に決定したレイアウトフレームPlanLFNameに対応した区画分割構造が、計画時の子回路の形状と実形状とのギャップによって満たされなくなった場合には、適切なレイアウトフレームRealLFNameに変更する。

```
module([{aggregate_subproblems,Proc,end-Eo}|TMess],LM):-
    get_and_put(patent_shape,LM,PlanShape,RealShape,LM1),
    get_and_put(layoutframe,LM1,PlanLFName,RealLFName,LM2),
    get(sons_shapes,LM2,SonsShapes,NLM,end-Eo),
    planframe(PlanLFName,aggregate_sons_shapes,PlanShape,SonsShapes,RealShape),
    renew_lfname(PlanLFName,RealShape,RealLFName),
    module(TMess,NLM).
```

4．7　配線準備

トップモジュールより最上位回路モジュールに各配線前処理の実行を要求する。

get_topmost_planメッセージは、最上位回路の実形状TopShapeとこの段階では古くなっている目標外郭上コネクタExternalConnectorsを取得する。

generate_global_placementメッセージは、配置処理のボトムアップ過程で各回路モジュールの配置が北西端点を原点とした局所座標となっているため、以降の配線のためにこれをグローバルな世界座標系に直す。TopShapeの北西端点が世界座標の原点でとなる。又、配置時に生じたデ

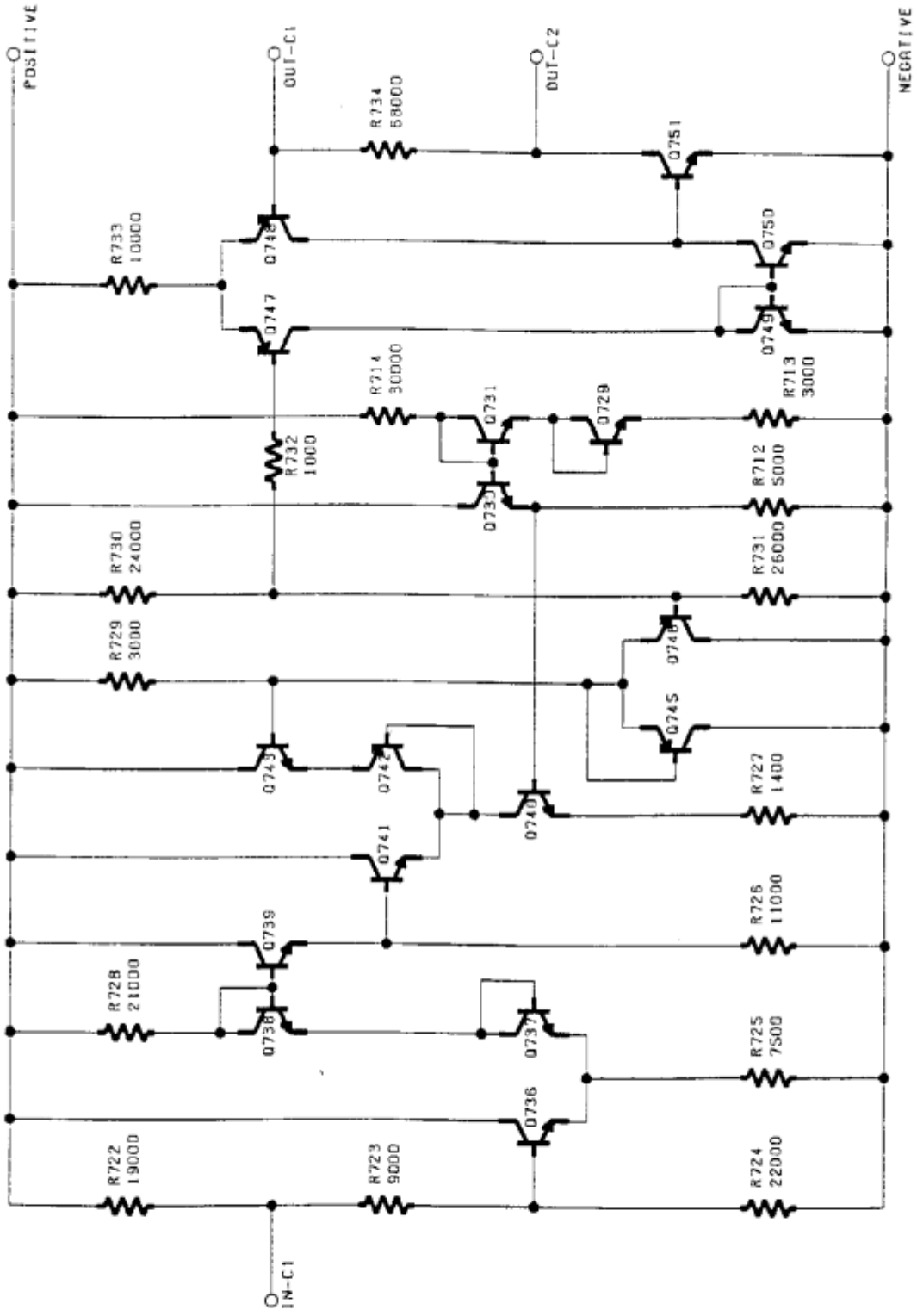


図 3-5(1) 出力例（原回路図）

```
    Ru = [{generate_global_placement,RuS,E2-ERu}],
    judge_end([ELu,ELb,ERb,ERu],Eo),
    module(TMess,NLM).

module([{generate_mazerouter_process,Proc,end-Eo}|TMess],LM):- cell(LM) !
    generate_inner_connectors(LM,Proc,ConnsProfile,LM1,end-E1),
    get(patent_shape,LM1,Shape,LM2),
    take_line_rule(Proc,LayerNo,Distance),
    generate_mazerouter_process(Shape,ConnsProfile,LayerNo,Distance,Proc,MazeStr,
        E1-Eo),
    put(mazerouter,LM2,MazeStr,NLM),
    module(TMess,NLM).
module([{generate_mazerouter_process,Proc,end-Eo}|TMess],LM):- not(cell(LM)) !
    get(sons_stream,LM,[Lu,Lb,Rb,Ru],NLM),
    Lu = [{generate_mazerouter_process,Proc,end-ELu}],
    Lb = [{generate_mazerouter_process,Proc,end-ELb}],
    Rb = [{generate_mazerouter_process,Proc,end-ERb}],
    Ru = [{generate_mazerouter_process,Proc,end-ERu}],
    judge_end([ELu,ELb,ERb,ERu],Eo),
    module(TMess,NLM).

module([{set_topmost_plan,ExternalConnectors,end-Eo}|TMess],LM):-
    get(patent_shape,LM,Shape,LM1),
    put(peri_connectors,LM1,NewExternalConnectors,NLM),
    modify_external_connectors(ExternalConnectors,Shape,NewExternalConnectors,
        end-Eo),
    module(TMess,NLM).
```

4．8　電源配線

route_by_qtreeメッセージにより配線のレイアウト処理を行う。配線はまず電源線配線、次に信号線配線の順で行う。

```
top_module([{route_by_qtree,Proc,end-Eo}|TMess],OutStr,ToModules,FmModules,TM):-
    ToModules = [{route_powerlines,Proc,end-E1},
                 {route_signallines,Proc,E1-E2},
```

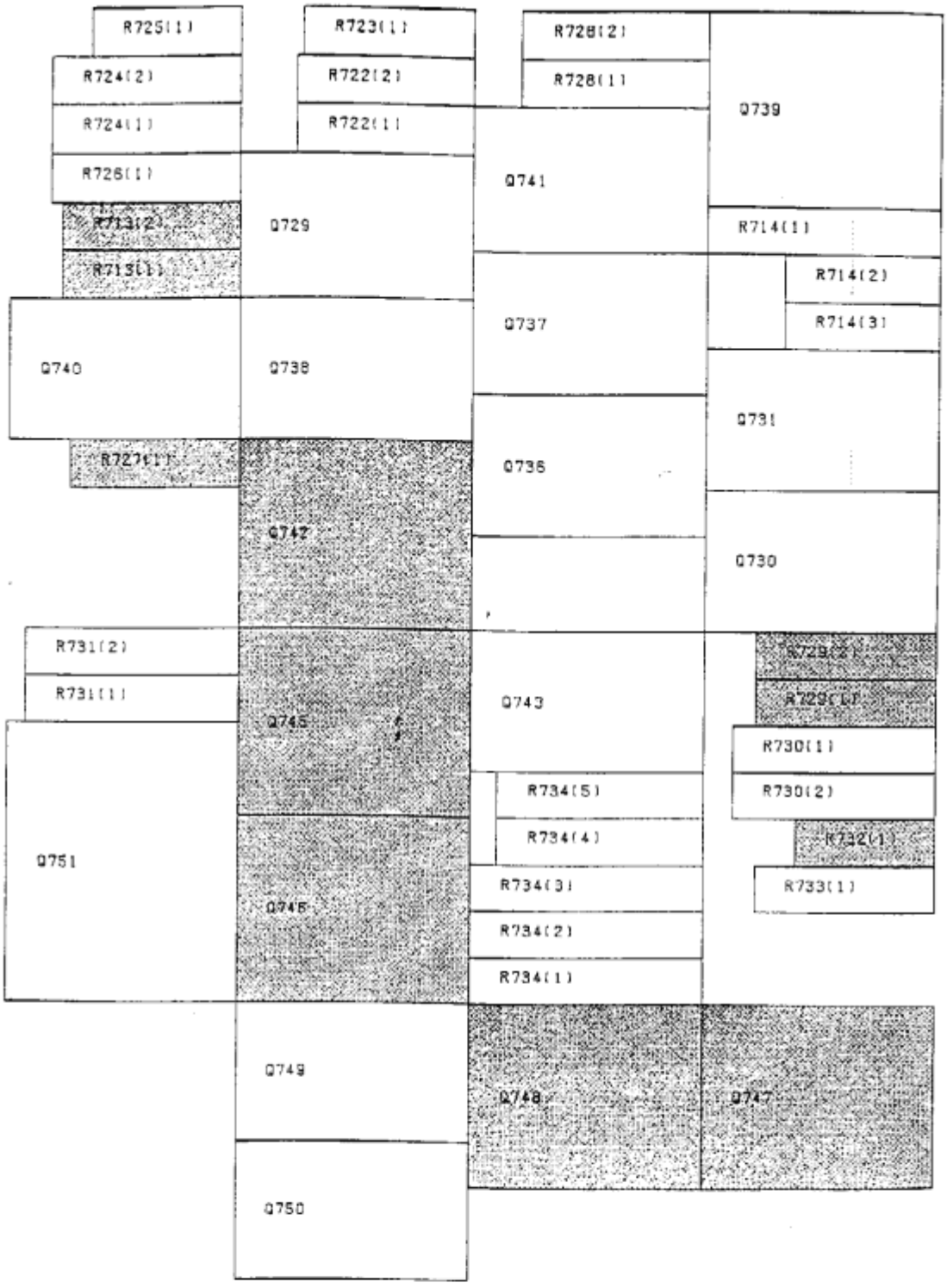


図 3-5(3) 出力例（仮配置案）

```
    Lu = [{route_powerlines,Proc,end-E1}],
    Lb = [{route_powerlines,Proc,end-E2}],
    Rb = [{route_powerlines,Proc,end-E3}],
    Ru = [{route_powerlines,Proc,end-E4}],
    judge_end([E1,E2,E3,E4],Eo),
    module(TMess,NLM).
```

4．8．2　電源配線部分問題生成

plan_power_wiring/3は、各子回路モジュールが配線した電源線の配線長と電位を取得するための変数を生成する。aggregate_powerlines節でこの変数に設定された数値を基に配線幅を決定する。propagate_plans/3は、上記変数を各子回路モジュールに設定する。

```
module([{generate_power_subproblems,Proc,end-Eo}|TMess],LM):-
    get(sons_stream,LM,SonsStream,LM1),
    get_and_put(power_nets,LM1,PowerNets,NewPowerNets,NLM),
    plan_power_wiring(PowerNets,PowerNetsProps,NewPowerNets),
    propagate_plans(PowerNetsProps,SonsStream,end-Eo),
    module(TMess,NLM).
```

prepare_power_nets/6の第1節は、セルの回路モジュールに電源ネットが存在しない場合を処理する。第2節は、電源ネットが存在する場合を処理する。

generate_dummy_inner_connectors/5は、セル内の実コネクタと親の回路モジュールの電源配線とを結ぶ仮想内点コネクタプロセスを生成する。

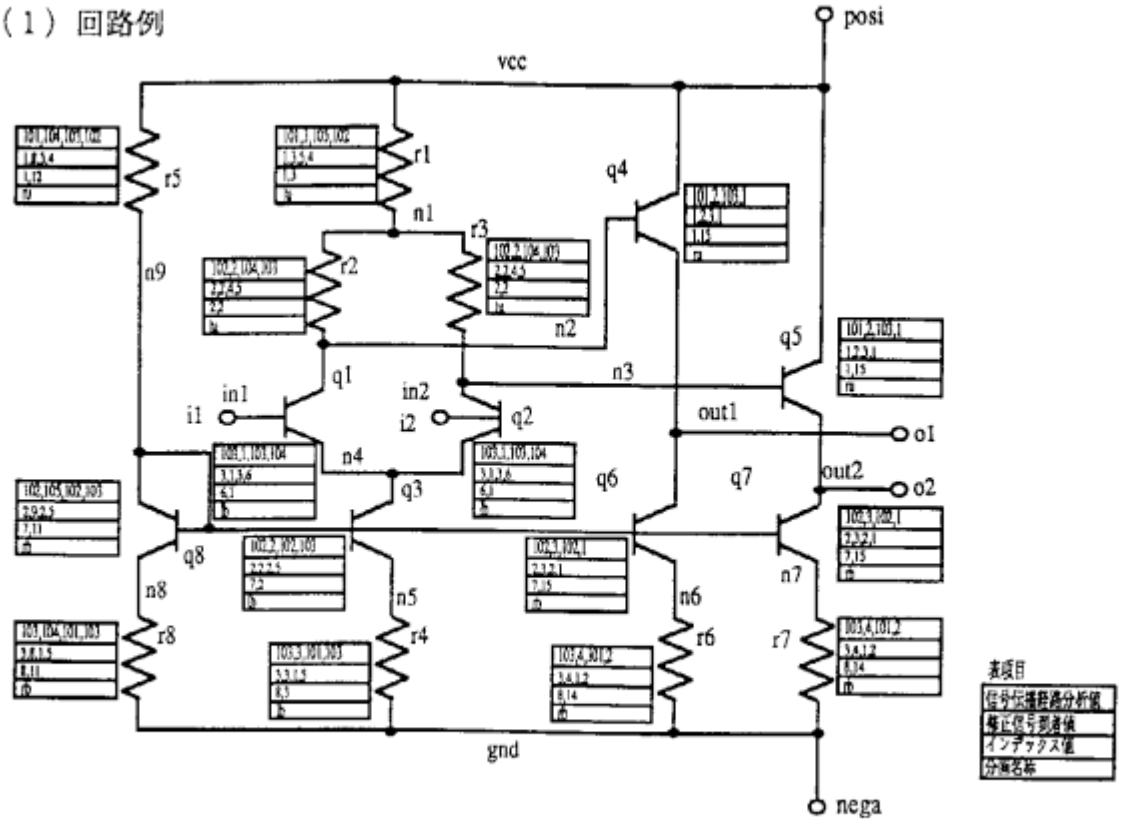
set_power_net_props/3は、電源線の配線長と電位を調べる。get_power_connector_profile/4は、実コネクタと仮想内点コネクタの位置とコネクタ層を取得する。

```
prepare_power_nets([],NewPowerNets,RoutablePowerNets,_,_,Ei-Eo):-
    close([NewPowerNets,RoutablePowerNets],Ei-Eo).
otherwise.
prepare_power_nets(PowerNets,NewPowerNets,RoutablePowerNets,Shape,Proc,Ei-Eo):-
    generate_dummy_inner_connectors(PowerNets,Shape,Proc,PowerNets1,Ei-E1),
    set_power_net_props(PowerNets1,PowerNets2,Shape),
    get_power_connector_profile(PowerNets2,RoutablePowerNets,NewPowerNets,E1-Eo).
```

4．8．3　電源配線部分問題統合

find_a_good_power_route_pattern/4は各子回路モジュールをつなぐ電源配線パターンを選び出し、取得した各子回路モジュールの電源線の配線長と電位から電源配線幅を決定し、電源配線

（1）回路例



（2）プロセスネットワーク

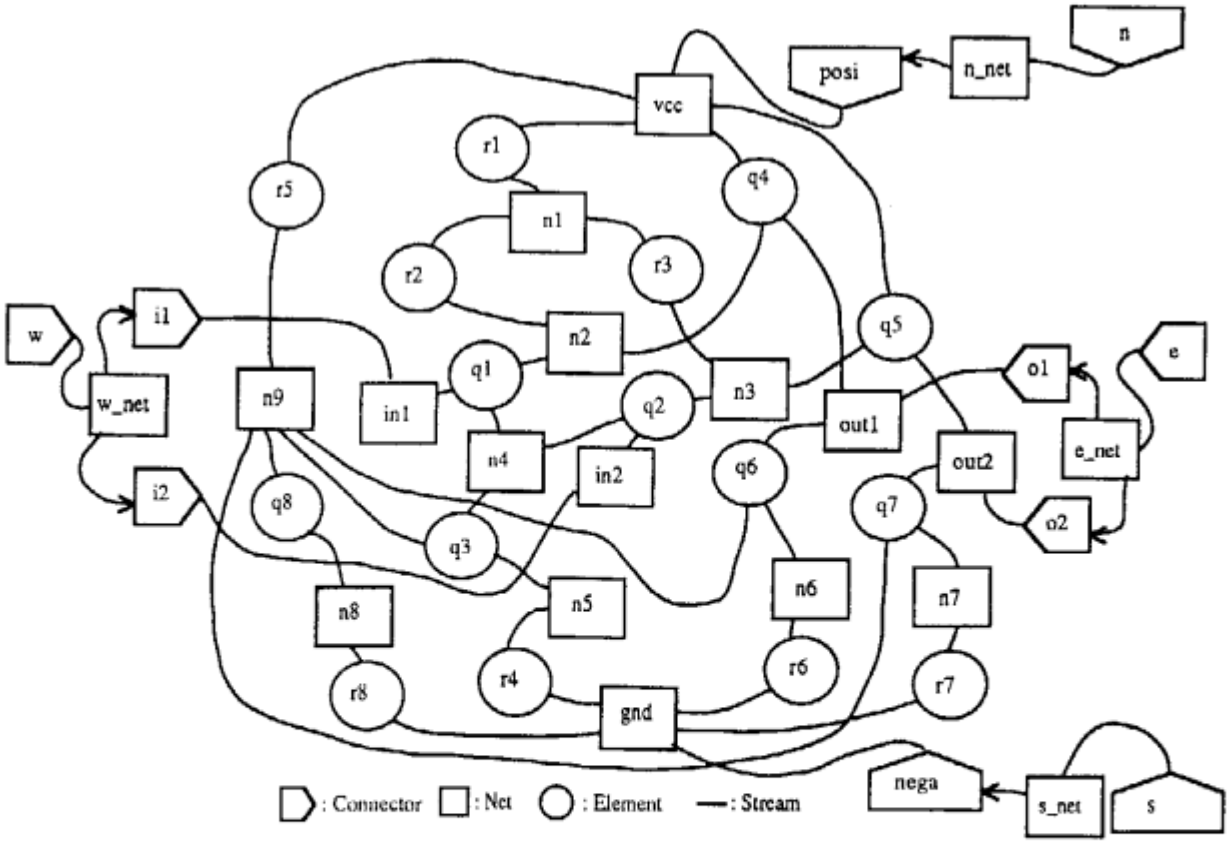


図 4-1 回路ネットのプロセスネットワーク化

```
module([{route_signal_all_sons,Proc,end-Eo}|TMess],LM):-
    get(sons_stream,LM,[Lu,Lb,Rb,Ru],NLM),
    Lu = [{route_signallines,Proc,end-E1}],
    Lb = [{route_signallines,Proc,end-E2}],
    Rb = [{route_signallines,Proc,end-E3}],
    Ru = [{route_signallines,Proc,end-E4}],
    judge_end([E1,E2,E3,E4],Eo),
    module(TMess,NLM).
```

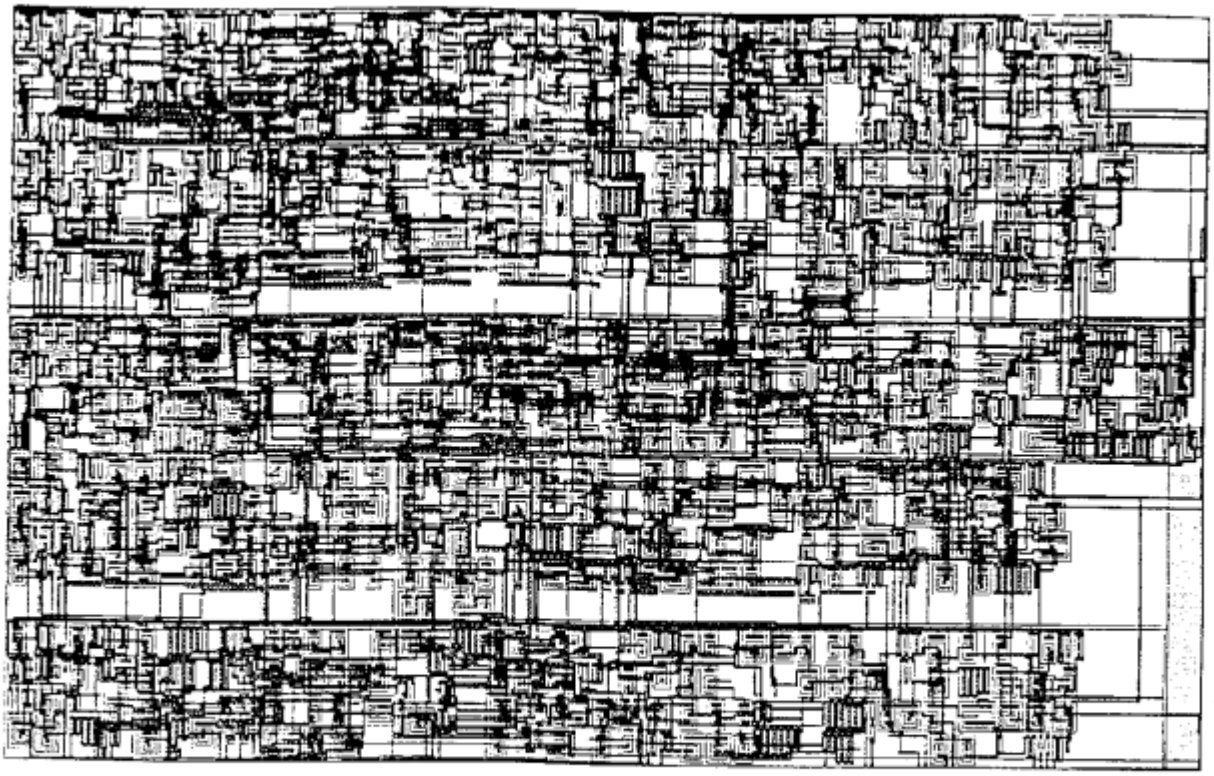
４．１０　セル内配線

４．１０．１　セル内配線制御

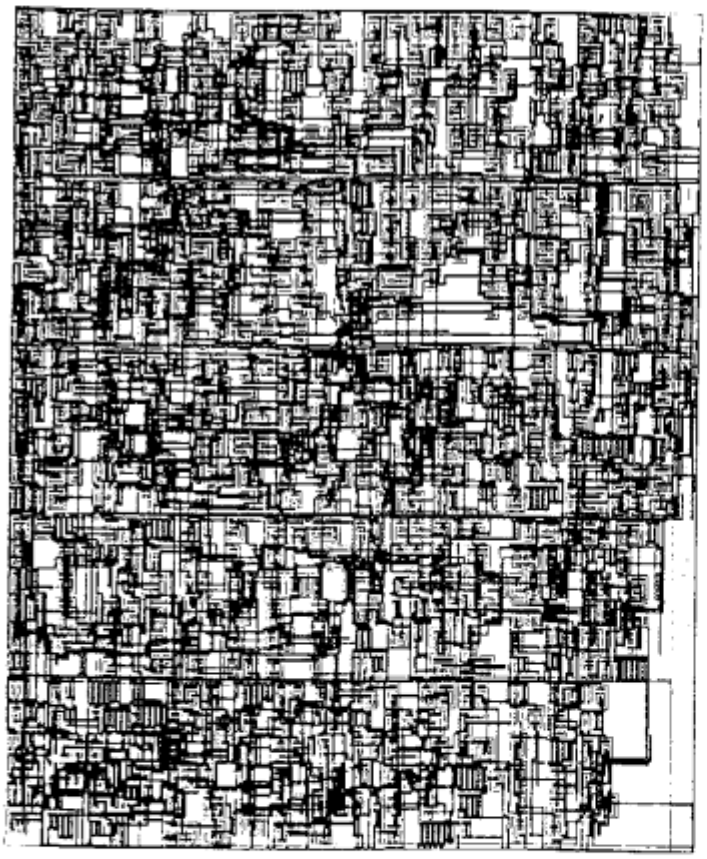
セル内配線は外郭上コネクタと実コネクタとの関係によって配線処理する順序を次の様に制御する。最初は、セルの南辺と東辺に存在する外郭上コネクタと実コネクタ間を配線するもの。次に、セルの北辺と西辺に存在する外郭上コネクタと実コネクタ間を配線するもの。最後に、外郭上コネクタ間（フィードスルー）を配線するもの。以上の順序でセル内配線を行う理由は次の通りである。　セルの外郭上コネクタは、隣接セルの回路モジュールの間で共有されている。仮に各セル回路モジュールが自由に外郭上コネクタにアクセスしコネクタ位置を決定しようとすると、このコネクタを共有する隣接セル回路モジュールでは待ち状態になり、各セル回路モジュールが並列に配線処理を行う上でのタイムロスを生じる。各セル回路モジュールが上記の順序を守ればこれを回避できる。また、実コネクタ配線をフィードスルー配線以前に行う事により、位置不定の外郭上コネクタを実コネクタからより直線的（最短）な位置に固定できる。

第１節は、セルの回路モジュールを処理する。partition_and_get_connectors/5は、配線順序を指定するTypeによって各ネットごとに外郭上コネクタおよび実コネクタ位置とコネクタ層を取得する。この情報を迷路法の天元プロセスにwireメッセージとして送信する。第２節は、分割を行うブロック階層の回路モジュールの処理を行う。

```
module([{route_leafcells,Type,Proc,end-Eo}|TMess],LM):- cell(LM) |
    partition_and_get_connectors(Type,LM,RoutableNets,LM1,end-E1),
    get(mrpn_stream,LM1,MRPNMessage,NLM),
    MRPNMessage = [{wire,RoutableNets,E1-Eo}],
    module(TMess,NLM).
module([{route_leafcells,Type,Proc,end-Eo}|TMess],LM):- not(cell(LM)) |
    get(sons_stream,LM,[Lu,Lb,Rb,Ru],NLM),
    Lu = [{route_leafcells,Type,Proc,end-E1}],
    Lb = [{route_leafcells,Type,Proc,end-E2}],
```



（1）目標形状を横長とした場合



（2）目標形状を縦長とした場合

図 6-1 レイアウト結果の一例

４．１１　セル内迷路法配線

４．１１．１　メーズルータプロセスネットワークの生成

配線準備から要求され、天元コネクタMRPプロセス(top_mrp/2)を生成し、各MRPプロセスを生成するためにcreate_mrpnメッセージをtop_mrp/2に送出する。

MRPNは図３－３に示したもので、セル面をおおうMRP群を敷設し各近接MRP間は双方向メッセージストリームで接続されている。前述の回路ネットの階層化処理の前段処理である、回路ネットモジュール間ストリーム接続で説明したものと類似のストリーム構造を形成している。

次にdef_obsメッセージによりセル内コネクタ位置を管理するMRPへの障害物登録、およびdef_pobsメッセージにより南辺上と東辺上を管理するMRPのスルーホール敷設禁止登録をtop_mrp/2に要求する。

top_mrp/2のMRPNStrは放送ストリームであり、全MRPプロセスが同じものを持っている。

障害登録要求等の天元コネクタMRPから全MRPへの放送の場合、放送メッセージを受信した各MRPは、自MRP位置が障害物エリアで指定された矩形範囲内にあるか調べ、あれば登録を行うが、なければメッセージを無視する。

```
generate_maze_router_process(Shape,ConnsProfile,LayerNo,Distance,Proc,MazeStr,
        Ei-Eo):-
    Shape = {Xs,_,_,Xe,Ys,_,_,Ye},
    XN := (Xe-Xs)/Distance,
    YN := (Ye-Ys)/Distance,
    change_obstacles(ConnProfile,Proc,Obstacles),
    top_mrp(MazeInStr,{{Xs,Xe,Ys,Ye},Distance,Proc,MRPNStr,[],[]}),
    MazeInStr = [{create_mrpn,Xs,Ys,XN,YN,Distance,Ei-E1},
                 {def_obs,Obstacles,E1-E2},
                 {def_pobs,[{Xs,Xe,^(Ye-Distance),Ye},
                 {^(Xe-Distance),Xe,Ys,Ye}E3-Eo}],E2-Eo}|MazeStr].
```

４．１１．２　プロセス内データの表現

天元プロセスMRPデータ

```
TM::={Shape,Distance,Proc,MRPNStr,AnsWires,AnsTHs}.
    Shape::={Xs,Ys,Xe,Ye}
    Distance::=MRPのサイズ
    Proc::=LSI製造プロセス名称
    MRPNStr::=全MRPへの放送ストリーム
```

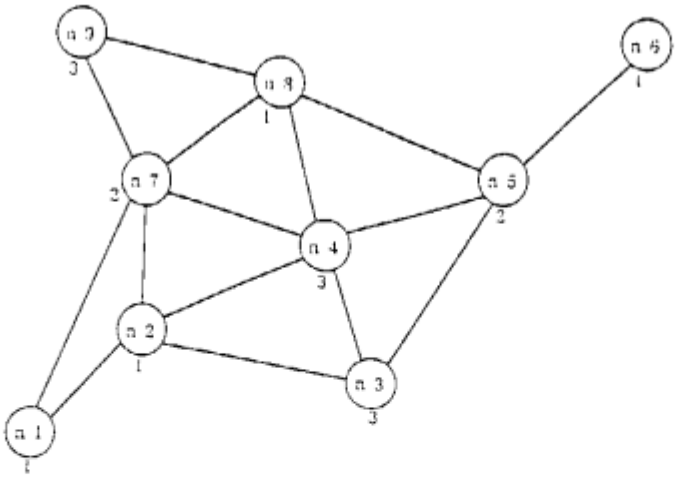


図７－１　探索対象ネットワークの例

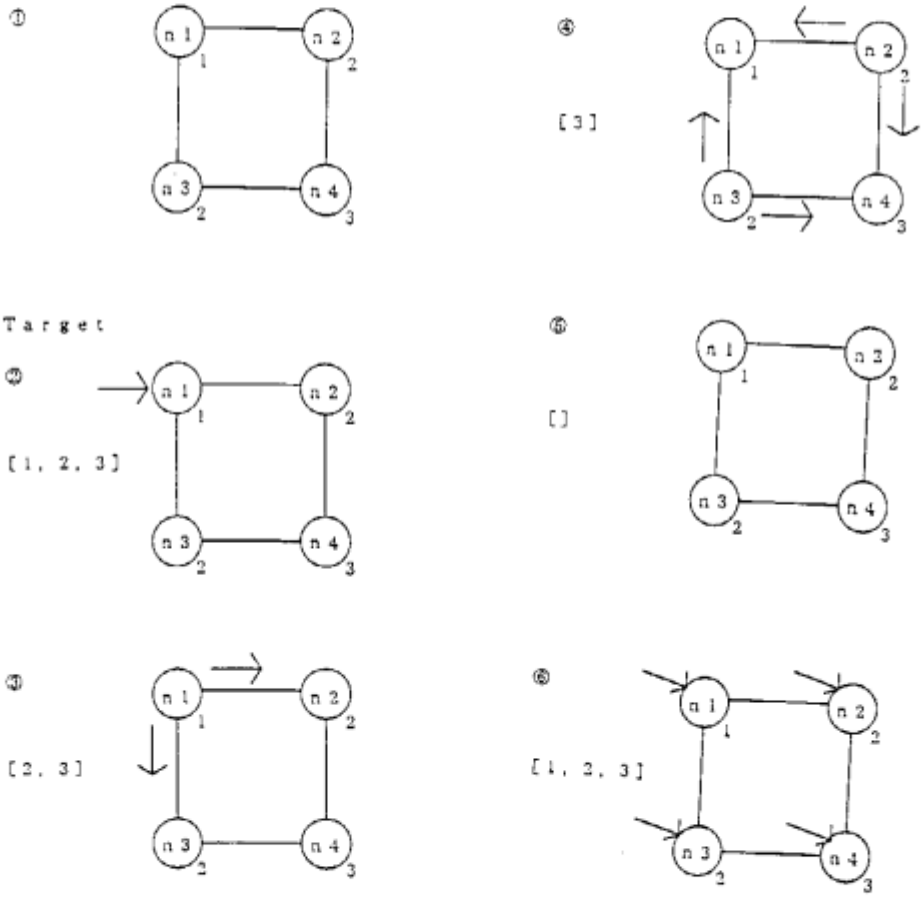


図７－２　探索方式

find_routeメッセージで配線経路を探索し、collect_routeメッセージで終端点から開始端点への配線経路を形成するMRP連鎖を得る。配線経路探索後、become,wakeメッセージでsleep状態化したMRPをwake状態に復帰させる。

change_wires/7で配線経路を配線物に変換する。change_obstacles/4で配線経路を他ネットへの配線障害物に変換し、def_obsメッセージにより配線障害範囲とスルーホール障害範囲を各MRPに放送する。配線結果はtop_mrp/2に保存する。

第2節は配線要求終了時で、配線処理の終了フラグ設定と次のメッセージ処理のための再帰を行う。

```
top_mrp([{wire,[InNet|T],end-Eo}|MazeStr],TM) :-
    InNet = {NetName,Conns} , TM  = {Shape,Distance,Proc,MRPNStr,AnsWires,AnsTHs} |
    get_conn_mrp(Conns,Shape,MRPNStr,[StartMRP|EndMRPs],MRPNStr1),
    get_peri_conn_out_ranges(Conns,Shape,Distance,PeriDirOutRanges),
    MRPNStr1 = [{refresh,end-E1},
                {become,sleep,PeriDirOutRanges,E1-E2},
                {find_route,NetName,StartMRPs,E2-E3},
                {collect_route,NetName,EndMRPs,StartMRP,RouteMRPs,E3-E4},
                {become,wake,PeriDirOutRanges,E4-E5}|MRPNStr2],
    change_wires({NetName,RouteMRPs},Proc,MRPNStr2,Wires,Through_Holes,MRPNStr3,
        E5-E6),
    change_obstacles(Wires,Proc,WireObsts),
    change_obstacles(Through_Holes,Proc,THObsts),
    MRPNStr3 = [{def_obs,WireObsts,E6-E7},
                {def_obs,THObsts,E7-E8}|MRPNStr4],
    top_mrp([{wire,T,E8-Eo}|MazeStr],{Shape,Distance,Proc,MRPNStr4,
    [Wires|AnsWires],[Through_Hols|AnsTHs]}).
top_mrp([{wire,[],end-Eo}|MazeStr],TM) :-
    Eo = end,
    top_mrp(MazeStr,TM).
```

MRP障害化

障害化要求放送受信の処理であり、第1節は障害範囲内に自MRPが位置する場合で、適用可能ネット名称と適用可能ネット以外の場合の適用コストを更新する。

第2節は障害範囲内に自MRPが位置しない場合で、次の障害化要求を再帰実行する。第3節は障害化要求終了で、次の放送受信を再帰実行する。

```
mrp([{def_obs,[{ObsRange,ObsLayer,AddCost,PermitNet}|ObsT],end-Eo}|MRPNStr],
```

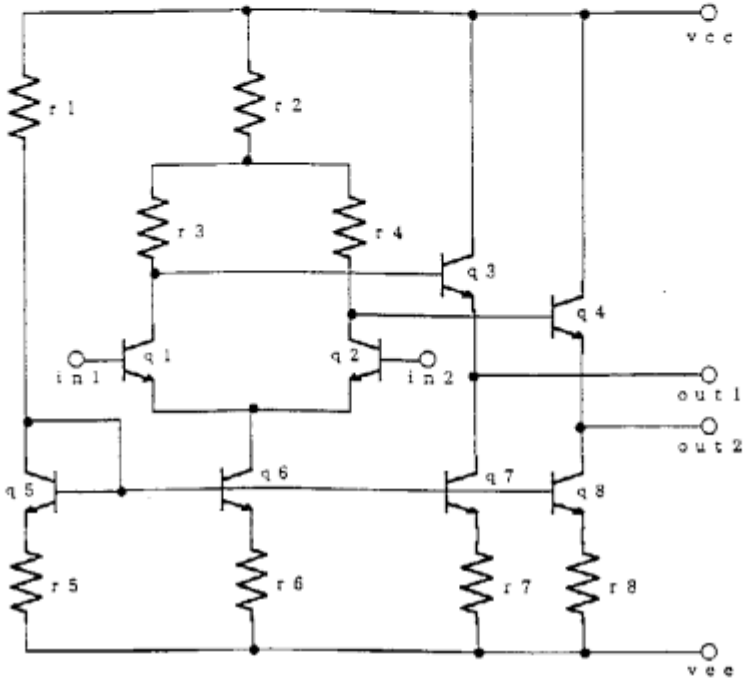

vcc
r1
r2
r3
r4
q3
q4
q1
q2
in1
in2
out1
out2
q5
q6
q7
q8
r5
r6
r7
r8
vee

図７－５　回路図の例

```
mrp(MRPNStr,NeigStr,MD1).
```

最外郭MRP状態遷移

セル外部への配線引き出し不可範囲にある最外郭MRPをsleep状態化し、引き出し点適用不能とするもので、不可範囲にあるMRPの引き出し方向への経路探索メッセージを送る隣接ストリームをsleep登録する。また、復帰時のwake登録も行う。

```
mrp([{become,Kind,[PeriDirOutRange|PeriDirOutRangeT],Ei-Eo}|MRPNStr],
    NeigStr,MD) :-
    MD  = {RVFlag,MRPAddr,MRPShape,Permits,Streams,PCosts,Costs,Temp},
    PeriDirOutRange = {OutRange,Dir},
    inside(MRPAddr,OutRange) |
    set_neigstr_status(Kind,Dir,Streams,Streams1),
    MD1 = {RVFlag,MRPAddr,MRPShape,Permits,Streams1,PCosts,Costs,Temp},
    mrp([{become,Kind,PeriDirOurRangeT}|MRPNStr],NeigStr,MD1).
```

経路探索開始

配線経路探索の開始を、開始端点となるMRPに要求する。前述の回路ネットの階層化処理における信号伝播経路分析で説明したものと類似の探索処理を行うものである。

```
mrp([{find_route,NetName,StartMRP,end-Eo}|MRPNStr],NeigStr,MD) :-
    MD  = {RVFlag,MRPAddr,MRPShape,Permits,Streams,PCosts,Costs,Temp},
    StartMRP = MRPAddr |
    message(MRPAddr,NetName,0,MaxCost,*,Start,end-Eo),
    mrp(MRPNStr,[Start|NeigStr],MD).
```

配線経路MRP収集

配線経路探索結果として探索メッセージの送出側MRPが各MRPに保存されている。終端点から開始端点までの送出側MRPをcollect_route/7によってたどることで、配線経路を収集する。なお仮想コネクタMRPは実体ではないので経路MRPには加えない。

```
mrp([{collect_route,NetName,[EndMRP|EndMRPs],StartMRP,RouteMRPs,
    end-Eo}|MRPStr],NeigStr,MD) :-
    MD  = {RVFlag,MRPAddr,MRPShape,Permits,Streams,PCosts,Costs,Temp},
    EndMRP = MRPAddr |
    collect_route(RVFlag,MRPAddr,Temp,Streams,RouteMRPsPart,Streams1,end-E1),
    RouteMRPs = [RouteMRPsPart|RouteMRPsT],
    MD1 = {RVFlag,MRPAddr,MRPShape,Permits,Streams1,PCosts,Costs,Temp},
```

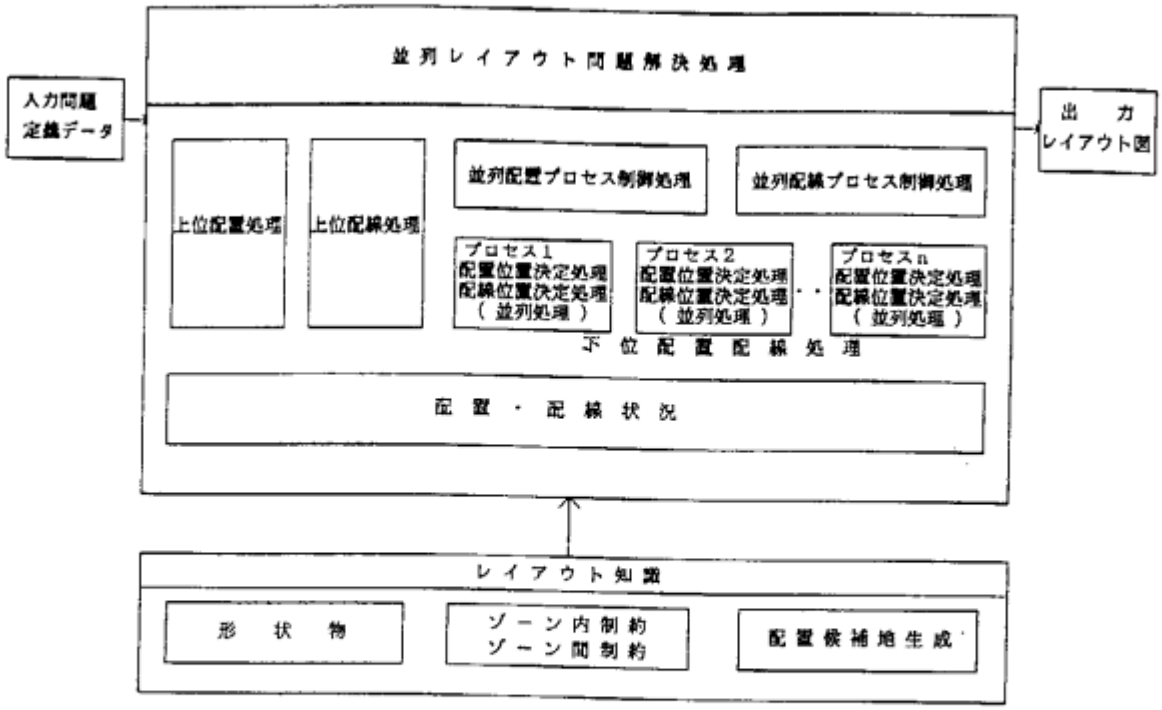


図1 計算機室レイアウトシステムの機能構成

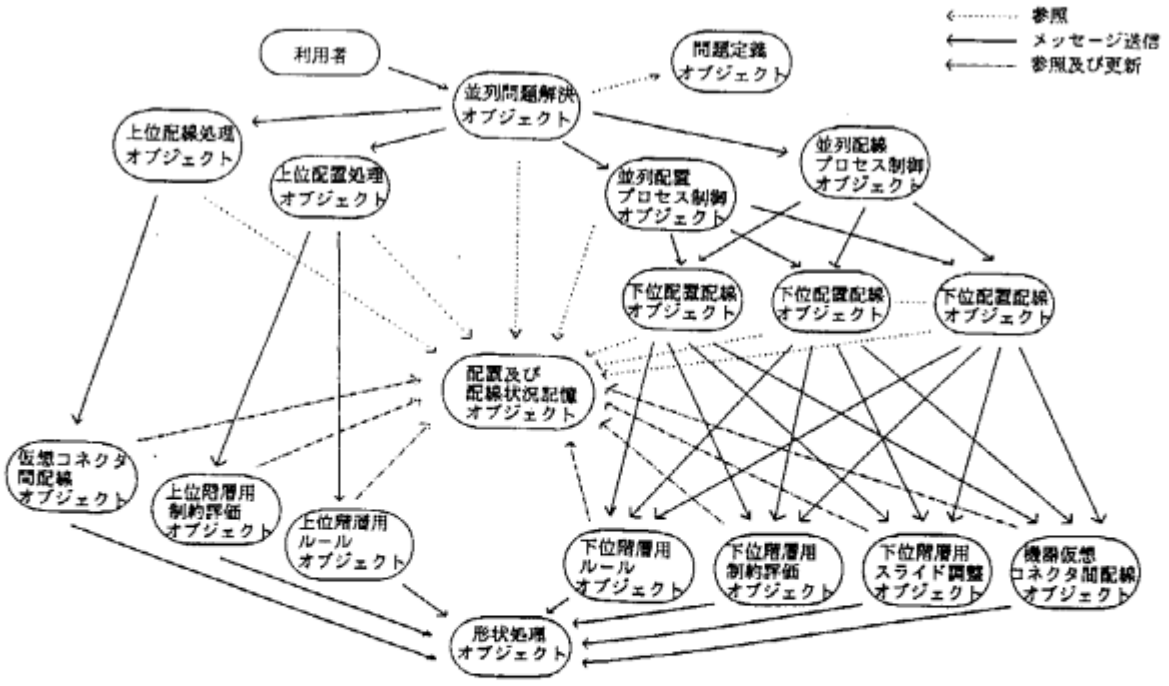


図2 計算機室レイアウトシステムのプロセス構成

```
    put_sons_shape(SonsPlace,
    [X0,X3UpL,Y0,Y1,X0,X3BeL,Y1,Y3N,X3BeL,X3BeR,Y1,Y3N,X1,X3UpR,Y0,Y1],
    NSonsPlace),
    HUp := (SLu+SRu)/(X3-X0),HBe := (Y3-Y0)-HUp,
    Y1 := Y0+HUp,
    Y3N := HUp+HBe,
    X3UpL := X0+SLb/((Y3-Y0)-HUp),
    X3UpR := X3UpL+SRb/((Y3-Y0)-HUp),
    big(X3UpR,X3BeR,X3N),
    cal_deadspace(~(X3N-X0),~(Y3-Y0),~(X3N-X0),~(Y3N-Y0),NDeadsp),
    evaluate_check([~(Y3N-Y3)],NAspect).
```

（2）配線評価

所与の外郭上コネクタ配置のもとで子回路の埋め込み案がもたらすフィードスルー配線（ブロック上空通過配線。配線の悪さの目安）を計算する。

```
planframe(LFName,evaluate_wires,NWSENets,SonsNets,WireGoodness,WireCost) :-
    LFName = r4h_uls , NWSENets = [NN,WN,SN,EN],SonsNets = [SN1,SN2,Sn3,SN4] !
    has_stronger_connection(NN,[SN1,SN4],SN,[SN1,SN4],P1),
    has_stronger_connection(WN,[SN2,SN3],SN,[SN1,SN4],P2),
    has_stronger_connection(SN,[SN1,SN2],EN,[SN1,SN2],P3),
    has_stronger_connection(EN,[SN4,SN3],EN,[SN1,SN2],P4),
    has_stronger_connection(SN1,[SN2,SN4],SN1,SN3,P5),
    has_stronger_connection(SN2,[SN1,SN3],SN2,SN4,P6),
    has_stronger_connection(SN3,[SN2,SN4],SN3,SN1,P7),
    has_stronger_connection(SN4,[SN1,SN3],SN4,SN2,P8),
    check_negative_connections([P1,P2,P3,P4,P5,P6,P7,P8],
    0,WireCost,good,WireGoodness).
```

（3）配線容量（可能本数）評価

回路区画境界辺を通過可能な配線本数を推定する。

```
planframe(LFName,generate_goal_vector,GoalVec,InitVec,InitCost,Shape,Proc) :-
    LFName = r4h_uls , Shape = {X0,X1,X2,X3,Y0,Y1,Y2,Y3} !
    get_basic_slice(Proc,BSlice),
    get_basic_linevector_layers(Proc,BLineLayers),
    L1 := Y1-Y0 , L2 := Y3-Y1 , L3 := X2-X0 , L4 := X3-X2 ,
    L5 := L2 , L6 := L1 , L7 := X3-X1 , L8 := X1-X0 ,
    L9 := L1 , L10 := L2 , L11 := L8 , L12 := X2-X1 , L13 := L4 ,
```

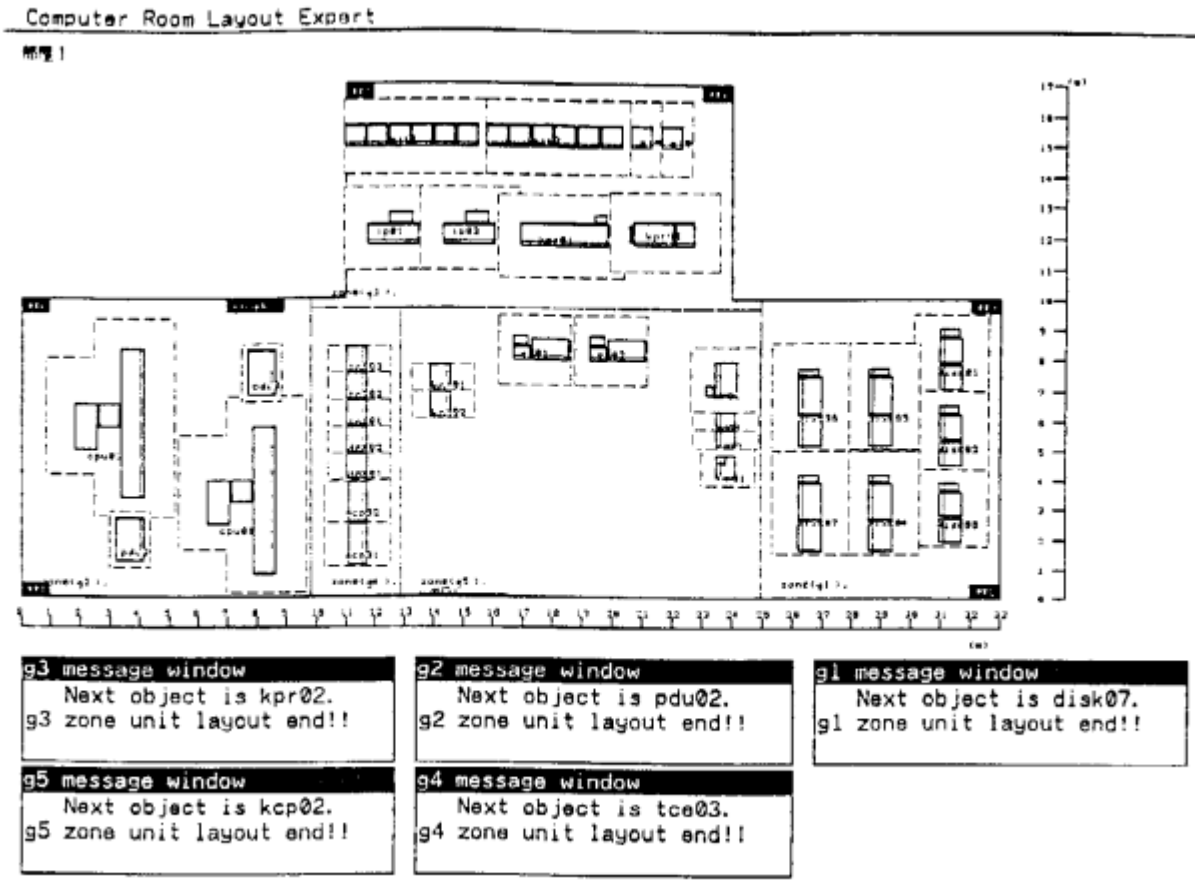


図 3-1 計算機室レイアウト実験画面（機器配置完了）

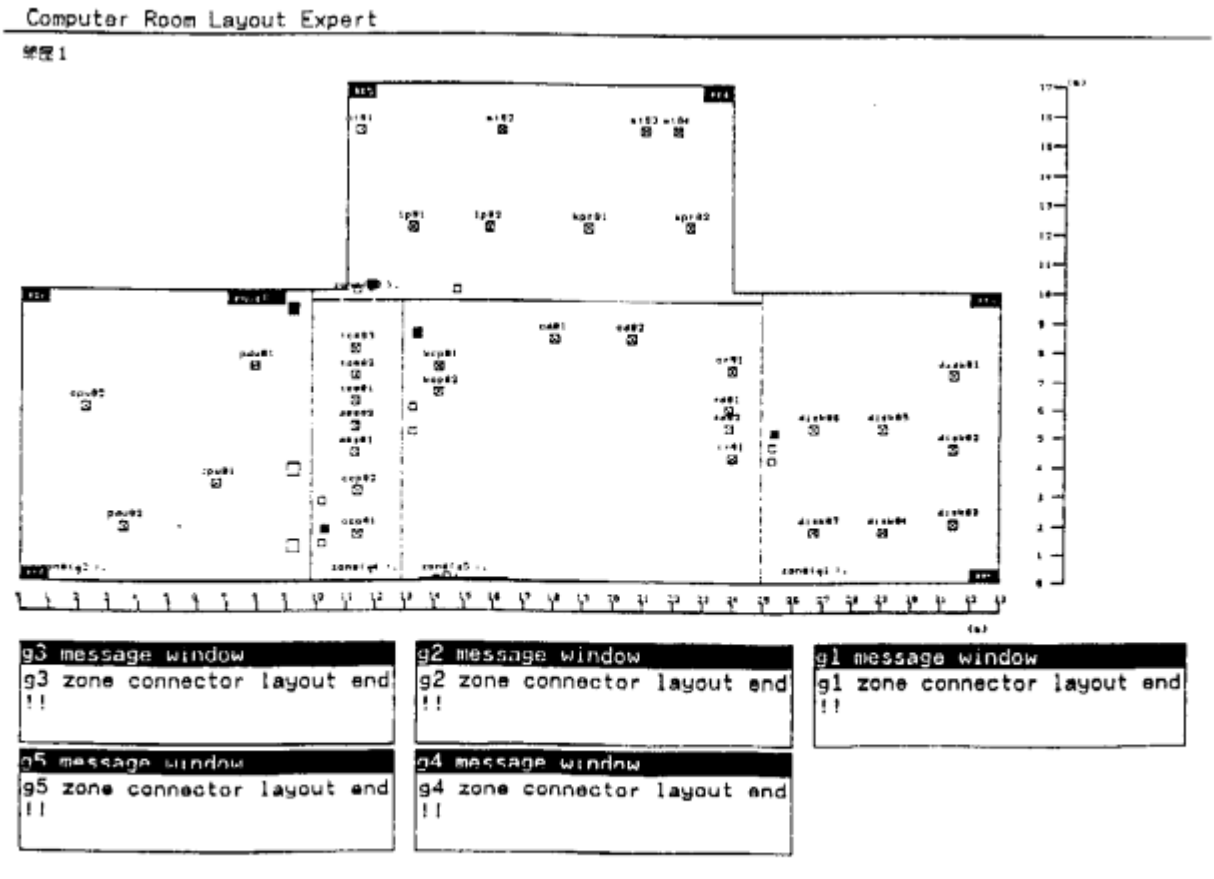


図 3-2 計算機室レイアウト実験画面（配線端子決定）

4．12．2　レイアウトフレーム

Layoutflame/8はブロック、セル等のデバイスを、それらを構成する部品の幾何学的パラメタや電気特性によって定義する。それらは当然バイポーラ、CMOSなどのLSI製造プロセスによって異なるので、各々について別のLayout framesを用いることになる。LFNameは検索キーの役目を果たすLayout framesの名称であり、製造プロセス名称Procもその属性の一つとして保有している。

```
layoutframe(LFName,Blocks,Connectors,Lines,RoutePatterns,
    RouteVector,Obstacles,Others).
```

4．12．3　レイアウトルール

Co-HLEXが現在保有しているレイアウトルールの一部を以下に示す。それらは、バイポーラプロセスに対する例である。

（1）配線可能層

```
layout_rule(usable_layers,process100,Layers):-true|
    Layers=[layer(1),layer(3),layer(3)].
```

（2）配置物間の距離

```
layout_rule(min_line_width,process100,signal,_,_,Width,_,_):-true|Width=4.
layout_rule(min_line_distance,process100,signal,signal,Dist):-true|Dist=4.
layout_rule(isolation_width,process100,_,_,Width):-true|Width=8.
```

（3）配線コネクタの形状

```
layout_rule(connector_params,process100,Layers,pad,Holes,Pads):-
true|Holes=[],Pads=[pad(layer(1),6,6)].
layout_rule(connector_params,process100,Layers,through_hole,Holes,Pads):-true|
    Holes=[hole([layer(1),layer(2)],4,4)],
    Pads=[pad(layer(1),6,6),pad(layer(2),6,6)].
```

（4）トランジスタの形状

NPNトランジスタの平面幾何形状をcollector,base,emitter拡散領域と配線引き出し口(Contacts)とで定義する。

```
layout_rule(tr_parameter,process100,npn,simple_tr,Diffusions,Contacts):-true|
    Diffusions=[diff(collector,n_diff,rect(0,0,40,72),_),
                diff(base,p_diff,rect(8,24,24,40),_),
                diff(emitter,n_diff,rect(16,48,8,8),_)],
    Contacts=[cont(collector,type1,center(20,12),[hole([layer(0),layer(1)],4,4)],
                                                 [pad(layer(1),6,6)]),
```

目次

```
module([{draw_layout,Proc,OutStr,end-Eo}|TMess],LM):- not(cell(LM)) |
    OutStr = {MyOutStr,LuOutStr,LbOutStr,RbOutStr,RuOutStr},
    get(sons_stream,LM,[Lu,Lb,Rb,Ru],LM1),
    get(power_nets,LM1,PowerNets,LM2),
    MyOutStr = [{draw,PowerNets}],
    Lu = [{draw_layout,Proc,LuOutStr,end-E1}],
    Lb = [{draw_layout,Proc,LbOutStr,end-E2}],
    Rb = [{draw_layout,Proc,RbOutStr,end-E3}],
    Ru = [{draw_layout,Proc,RuOutStr,end-E4}],
    judge_end([E1,E2,E3,E4,E5],Eo),
    module(TMess,LM2).
```

4．14　負荷分散法

HRCTLアルゴリズムの実行時にCo-HLEXが行う負荷分散方式を説明する。

入力階層化機能で生成したモジュールプロセスを、レイアウト機能実行用初期回路ネットモジュールプロセス生成{create_layout_module,...}メッセージによりレイアウト表現用分散プロセスネットワークのモジュールに変換する。このとき、モジュールプロセスの最適プロセッサ割り当てを行う。

```
top_module([{create_layout_module,Proc,Name,Father,PEs,CPE}|TMess],OutStr,ToModules,
FmModules,TM):-
    ToModules = [{create_layout_module,Proc,Name,PEs,CPE}|RMS],
    top_module(TMess,OutStr,RMS,FmModules,TM).
module([{create_layout_module,Proc,Name,Fater,PEs,CPE}|TMess],FmBelow,_,MD):-
    MD  = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work}|
    get_sons_name(Others,[LuName,LbName,RbName,RuName]),
    ToBroad = [{take_area,
              [LuName-LuArea,LbName-LbArea,RbName-RbArea,RuName-RuArea]}|TBt1],
    assign_processor(PEs,[LuArea,LbArea,RbArea,RuArea],[LuPEs,LbPEs,RbPEs,RuPEs],
       {LuCPE,LbCPE,RbCPE,RuCPE}),
    TBt1 = [{create_layout_module,Proc,LuName,Name,LuPEs,LuCPE},
          {create_layout_module,Proc,LbName,Name,LbPEs,LbCPE},
          {create_layout_module,Proc,RbName,Name,RbPEs,RbCPE},
          {create_layout_module,Proc,RuName,Name,RuPEs,RuCPE}|TBt2],
    MD1 = {Name,Type,Neighbour,Props,Others,TBt2,ToHigher,ToNeighbours,Work},
```

operative Hierarchical Layout EXpert)の第１版を開発した。
（２）平成２年度：Co-HLEXの並列処理能力を高めるために、第１版で採用していたレイアウト状態の集中メモリーによる記憶方式を分散するとともに、レイアウトに先立って必要となる回路ネットワークの自動階層化機能を開発した。これによって、大規模回路の高速処理を可能化した。配置配線機能の向上を図った。これらを加味してCo-HLEX第２版を実現した。

１．２．３　最終年度の目標・方針

Co-HLEX第２版までに開発した各種機能を改良・統合した第３版の開発と本研究の総合的まとめを行うことを本年度の目標とする。ＩＣＯＴで開発されてきた並列推論マシンＰＩＭの上でCo-HLEXを走行させ、最終的な機能実証としたいと考えている。

１．３　本報告の内容

本報告では、本研究の最終成果であるCo-HLEX第３版の詳細を述べる。中途の成果である中期の計算機室レイアウト問題解決については、その概要を付録に示す。

２　回路レイアウト問題

２．１　概要

本レイアウトシステムはCo-HLEXと呼んでいる。ここでは、Co-HLEXが解くレイアウト問題と問題解決に当たっての課題についてのべる。仕様記述には、論理型言語Prologを用いる。

２．２　問題説明

回路レイアウト問題とは、図２－１に示すように、与えられた回路ネットを与えられたチップ外形条件の中にレイアウトする問題を言う。

２．２．１　回路ネット

モジュール（トランジスタ、抵抗などの部品）、およびモジュール間の接続を定義するネットによる回路ネットワークの記述が回路設計の結果作成される。

２．２．２　チップ目標外形

最終的に得るチップの外形状（幅W、高さH）とチップ周辺上での端子位置（Ｖｃｃ，Ｇｎｄ，信号線の端子）。

O(log(no(PE))+leaf(PrBPT(R,N))/no(PE))の計算時間複雑度を有する。後者は問題規模に比して、プロセッサ数が少ない場合である。

（4）負荷分散法の証明

（場合1）　no(PE)>=leaf(PrBPT(R,N)):PrBPT(Rn,N)の最上位ノードを受理するPEが最も多くの負荷を受けるので、仮定からネックプロセッサとなる。それが処理する回路木の総数はPrBPT(R,N)の深さ、即ちlog(leaf(PrBPT(R,N)))である。

（場合2）　no(PE)=<leaf(PrBPT(R,N)):この場合も、PrBPT(R,N)の最上位ノードを受理するPEが最も多くの負荷を受けるので、仮定からネックプロセッサとなる。PrBPT(R,N)の処理において、回路木の深さlog(no(PE))に到るまでは、場合1が成り立つ。その後、各PEは全ての仕事を他に分散できず、自分で処理する。その仕事量はleaf(PrBPT(R,N))/no(PE)である。ネックとなる最上位プロセッサの総仕事量は両者の合計であり、log(no(PE))+leaf(PrBPT(R,N))/no(PE)となる。

6　レイアウト実験

6．1　概要

Co-HLEXの機能・性能を実証するため、以下に示すレイアウト実験を行い、その結果を考察した。また、汎用大型機上の実用レイアウトシステムとの比較検討も行った。

6．2　実験計画

6．2．1　実証したい事項

（1）並列配置配線能力の確認。

（2）チップ面積および総配線長の縮小が縦型調整と横型協調により実現できることの確認。

（3）異外形状配置配線が可能なことの確認。

（4）高い計算スピードとＰＥ台数効果。

（5）プログラムサイズの縮小と保守性。

6．2．2　実験対象回路の選定

バイポーラ・アナログ回路を実験対象とした。全素子数は１２９９個である。時間計測のために、この一部である２５４個、４８９個、８１０個についても測定した。なおこの個数は素子形状整合での高さ揃え後のもので、原回路素子数は少ないものから２１４個、４１０個、６０９個、１０１９個である。

ネット数はそれぞれ１５４個、２７６個、３９７個、６４９個、素子形状整合後では１８６個、

である。

Proc::=LSI製造プロセス名称。モジュール形状や、配線幅などに影響する。

Constr::=配置時のモジュール間近接条件等、及び、LSIの外形目標データ、外部への配線コネクタ目標位置。

（1）レイアウト状態の表現

solve_a_layoutproblem/4におけるLPlanは、問題解決に用いるレイアウトモデルに応じて、さまざまな表現形式を取りうる。Co-HLEXでは、[Samet 80],[Otten 82],[Luk 86]らによるQtree（Quadratic Tree：4分木構造）を利用する。図2－2にその概要を示す。

LPlan::=[PQtree,Wires].

PQtree::=長方形状区画の再帰的分割構造（スライス）を表す4分木。木の各ノードは、そこに配置するモジュール名称、北、西、南、東辺上の配線コネクタ名称等を持つ。

Wires::=[Conns,Lines].

Conns::=配線端点、Viaと呼ばれる配線層間の貫通穴、および配線が長方形状区画の周辺境界を横切る点に敷設するダミーコネクタ等のリスト。

Lines::=2つのコネクタを結ぶ配線のデータ。線分両端のコネクタ名称、配線幅、走行層名等からなる。

（2）制約付き階層レイアウト問題の再帰解法

レイアウトのスライス構造表現によって、レイアウト問題解決プログラムをエレガントで簡潔な再帰形式で実現する可能性が生じる。そのような実現の一例をProlog記述によって与える。

第1節はスライス階層末端でのリーフセルの外部ライブラリからの取り込み、第2節は階層中間部での問題分割、各部分問題のレイアウト、各部分問題のレイアウト結果のチェック、チェック結果が良好の場合の完成した各部分の統合による全体レイアウトの完成、というステップを踏んでいる。第3、4節は再帰を実現する。

```
solve_a_layoutproblem(CirNet,LPlan,Proc,Constr):-
    leaf(CirNet),!,
    use_library(CirNet,Proc,LPlan).
solve_a_layoutproblem(CirNet,LPlan,Proc,Constr):-
    not(leaf(CirNet)),!,
    generate_subproblems(CirNet,SonsCirNets,Constr),
    layout_all(SonsCirNets,SonsLPlist,Proc),
    check_shape_and_wire_abutment(SonsLPlist),
    aggregate_subproblems(SonsLPlist,LPlan).
layout_all([],[],_).
layout_all([Hcs|Tcs],[Hls|Tls],Proc):-
    solve_a_layoutproblem(Hcs,Hls,Proc),
```

6．3．5　PE台数効果

図6－2（3）より、良好なPE台効果があることが判る。並列処理される部分問題の計算時間に対して、部分問題の通信負荷が少ない場合に台数効果が向上する。Co-HLEXでは配線処理が最も時間を要するが、配線方向のグローバルな制御によるメッセージ削減と、セル内迷路法の計算時間が通信負荷よりも大きいことから、台数効果が良くなったと考えられる。

6．3．6　プログラムサイズと保守性

図6－2（4）に示すように、Co-HLEXのプログラムの配置配線機能は約6,000行の記述量で、従来の10^5行から10^6行に比べてきわめて少ない。これは、HRCTLと呼ぶ再帰アルゴリズムによる効果である。この高いプログラム記述性はKL1の再帰式記述性と、ストリーム並列データフロー計算方式により実現できた。

7　グラフの部分構造認識方法

7．1　概要

レイアウトCADシステムの研究開発の一環として、レイアウト結果得られたレイアウト図から、寄生素子等の部分構造を認識する機能の試作を行なった。計算機による寄生素子の自動抽出の試みは従来から行なわれていたが、配線容量のように配置物を指定すると一意に定まるものの抽出が主で、寄生サイリスタのように複数の配置物が連合して予期せぬ寄生素子を発生させるものの抽出については、その問題の複雑さ故に、まだ解決策が得られていない。その解決可能性をも考慮しながら本研究を開始した。研究を進めていくうちに、問題の核部分は寄生素子抽出に限らず、より広い分野において部分構造認識に適用できる可能性があり、有効な解決手段が発見できれば、様々な形で応用できることがわかった。

問題の本質部分は、原問題をグラフとして表現し、そのグラフに対して探索条件を入力すると、探索を行ない答を出力する、といった一種の質問応答系である。ここで言うグラフとは、属性情報を持つノードと、ノード間を結ぶ経路から構成されるネットワーク状のものを考える。このようなグラフに対して探索アルゴリズムを適用することにより、部分構造の認識を行なう。その際、探索対象となるグラフは一般に大規模なものとなり、探索量はその規模に応じて爆発的に増大する。計算機資源や時間が問題となってくる。そこで我々は、並列推論マシンを用いた問題解決を試みた。並列推論マシン（マルチPSI）と並列論理型言語（KL1）を用いて並列探索実験を行ない、計算モデルの適用性を検証した。以下、7．2節では、簡単な例題を挙げて問題を整理し、探索の方針を示す。7．3節では、部分構造認識のための対象グラフの計算機内部表現と探

範囲を狭める資格を持つ。同図（２）に横方向の協調を示す。

２．３．４　協調オーバヘッド削減のためのグローバル配線制御

横方向の並列協調による配線処理では、各プロセスは周囲のダミーコネクタの配置予定範囲を繰り返し検査する。多数のセルを有する配線の場合、この通信オーバヘッドは顕著な問題となる。通信オーバヘッド削減のために、セル内配線について配線方向のグローバルな管理を行う。

全てのセルについて、まず、内点コネクタを持ちかつ南－東辺上にダミーコネクタを持つネットの配線を行う。次に、内点コネクタを持ちかつ北－西辺上にダミーコネクタを持つネットの配線を行う。最後に、内点コネクタを持たないフィードスルー配線を行う。前２者ではダミーコネクタ位置決定は、空間的には並列、時間的には逐次となり、不要な通信が削減される。同図（３）に配線方向制御を示す。

３　システム全体構成と各要素の概要

３．１　システム全体構成

Co-HLEXのレイアウト機能の概要を図３－１に示す。原データ、入出力機能、バックアップメモリ、問題解決推論機能、レイアウト基本機能等から成る。入出力機能の一部である、入力階層化機能の概要を図３－２に示す。

入出力機能は原データの読み込み、平板な回路ネットプロセスの生成と平板な回路ネットプロセスの４分木構造回路ネットプロセスへの変換を行う。

問題解決推論プロセスは、回路ネットプロセスを用いた配置配線を行う。配置配線は共に、分割、統合の２フェィズで構成されている。分割時は、レイアウト計画の立案を行い、統合フェィズで配置物および配線物の組み上げを行う。配置と配線の間に配線準備を行い、異形状物の配置により発生するデッドスペースを木の末端プロセスであるセルに含ませるなどの処理を行う。
配置配線機能は各々再帰的に実行を繰り返し、末端はセルと呼ぶ単素子である。セルは、配置時には別途用意したテンプレートよりその形状を得、配線時は迷路法によるセル内配線によって配線線分を得る。配置配線機能は各々のフェィズにおいて、レイアウト基本機能に内蔵された形状テンプレートやプロセス制約等を参照し、レイアウト処理を行う。

レイアウト終了後、入出力機能の出力描画機能により、ＬＳＩチップのマスクパターンであるレイアウト図を表示する。

３．２　各要素の概要

目標リストに合致するすべての経路について探索を行ない、複数の解が存在する場合は、それらをすべて求めることが可能なアルゴリズムとする。

（2）探索開始ノードの近傍の探索に有効なものとする。

部分構造の認識を行なうのに効果的な方法とする。目標リストの長い、深い探索は行なわないことを前提とする。

（3）効率的に枝刈りを行なう。

一般に、全件探索を行なうと、探索の複雑さが爆発的に増大する。それを防ぐために、探索の各ステップで毎回、属性の判定を行ない、不要な経路はその都度探索を打ち切ることにより、早目に枝刈りを行なう。

（4）ノードには、最低限の情報のみ保持する。

部分構造認識が目的なので、一度の探索において参照されないノードの数が、参照されるノード数に比べてはるかに多くなるであろうことが容易に予測できる。ノードにローカルメモリ等を持たせて重くすると、参照されないノードのそれが無駄になる。

（5）マルチＰＳＩ及びＫＬ１を用いた効果的な動作が期待できるものとする。

マルチＰＳＩ上で、核言語ＫＬ１を用いてアルゴリズムを設計する。昨年までの研究で、ストリーム並列プログラミング技法が有効であることがわかっている。これらを用いた高速な動作が期待できるものとする。

７．３　並列推論マシン上での実現方式

７．３．１　グラフ表現と部分構造認識アルゴリズムの概要

部分構造認識アルゴリズムを次の部分より構成する。

（1）ネットワークを作成する。

ノードデータを参考にして、ハッシュプール内に、ノードプロセスを作成する。ノード間に双方向ストリーム（通信路）を生成する。

（2）始点ノードに目標リストを渡し、探索アルゴリズムを起動する。

（3）始点ノードが複数ならば、（2）を繰り返す。

（4）探索がすべて終了したら、解を出力し、ノード間のストリームを閉じる。

以下、具体例に沿って説明する。説明を簡単にするために、図７－２ (1) のようなｎ１～ｎ４の４個のノードと、それらを結ぶ４個の経路からなるグラフを考える。ここで、ｎ１の属性として「１」を、ｎ２、ｎ３は「２」を、ｎ４は「３」を、それぞれ与える。グラフの各ノードを、ＫＬ１言語で次のように記述する。

node(隣接ﾉｰﾄﾞﾘｽﾄ,属性,自ﾉｰﾄﾞ名)

例えば、ノードｎ１に注目すると、隣接ノードはｎ２とｎ３、属性は「１」なので、

３．２．３　配置機能

配置機能では、後述の配線機能の一部を用いたラフ配線をモジュールの位置決定時に行い、モジュール間配線長の短いモジュール配置を得る。

（１）セル配置

条件に適合するセルをテンプレートライブラリより選択する。配置の終了したセルの外辺コネクタ設定範囲にレイアウト済みのマークを付ける事によって、周囲のモジュールからの当該コネクタへの引き込みアクションを許し、配置処理の並列化効率を高めている。

（２）ブロック配置

ブロックモジュール形状を回路の推定面積により分割し、各分割に回路ブロックを割り当てることで配置を行う。回路木の親ノードの分割に適用するレイアウトフレームと呼ぶテンプレートをテンプレートライブラリより選択する。

可能な全てのテンプレートとその区画への子回路の埋め込み順列全てについて配置シミュレーションを行い、配線接続性と組み上げ後の推定形状の最適なパターンを調べて使用テンプレートと埋め込み順列を決定する。

親ノードは通常４個の子ノードに分割するが、３、２個に縮退した回路木についてもテンプレートは用意されており、分割可能である。

選択したテンプレートと、採用した子回路の各区画への埋め込み計画とを用いて子回路間のラフな配線計画を立てる。

配線の計画は、親回路の内部区画の各境界辺の配線容量（可能本数）と区画内へのビア（異配線層間の接続コネクタ）敷設可能数とを推定し、リソースベクタと呼ぶ資源管理表を生成し、このリソースベクタをバランスよく消費できる経路による配線パターンを選択することで行う。

計画した配線経路と内部区画境界線の交点にダミーコネクタを生成する。

また、親回路の持つ外周コネクタは、内部区画の外周枠の一片の長さにまで存在範囲を狭められる。ダミーコネクタは各子供の外周コネクタとして登録する。ここにおいて、各子供は親と同様に計画レイアウト（外周形状、外周コネクタ、内部にレイアウトすべき回路）を持つことになり、親を処理したプログラムで再帰処理できる。

配置終了後、テンプレートに依存した統合方針によって、完成された子回路を組み合わせて親回路の形状を求め、親の実形状とする。配置計画中に決定したテンプレート名に対応したスライス構造が、計画時の子回路の形状と実形状とのギャップによって満たされなくなった場合には、適切なスライス構造のテンプレート名に変更する。

３．２．４　配線準備機能

配置処理の統合過程で各モジュールの配置が北西端点を原点とした局所座標となっており、以降の配線のためにこれをグローバルな世界座標系に修正し、最上位回路の北西端点を世界座標の

かった旨を表わすマークとを含んだ木構造データで書き替えて、探索を終了する。

探索メッセージを、次のように構成する。

mes(TargetList,Tree,Switch)

第1引数の TargetList は目標リストで、例えば [1,2,3] のような形の属性のリストである。第2引数の Tree は解を表わす木構造データで、例えば t(NodeName,NewTree) のような形をしており、NodeName は解になりうるノードの名前を、NewTree は隣接ノードへの探索を進めるための新たな変数を、それぞれ表わす。第3引数の Switch は探索の終了を判定するためのスイッチである。

上記の探索アルゴリズムを図7－2 (1) のネットに適用すると次に示すようになる。目標リストを [1,2,3] とする。始点ノードをｎ1とし、目標リストを含んだ探索メッセージを渡して、探索を開始する（図7－2 (2)）。探索メッセージを受け取ったノードｎ1は、目標リストの先頭要素「1」と自ノードの属性「1」とを比較する。その結果、双方が一致するので、探索メッセージ内の変数 Tree を、隣接ノード数個（2個）の、自ノード名と新たな変数とを含んだ木構造データ t(n1,Tree11),t(n1,Tree12) で書き替えて、隣接ノード（ｎ2、ｎ3）に先頭要素「1」を除いた目標リスト [2,3] を含む探索メッセージを渡して、さらに探索を進める。この時、ｎ2への探索メッセージ内の変数を Tree11 に、ｎ3への変数を Tree12 に、それぞれ設定する（図7－2 (3)）。探索メッセージを受け取ったノードｎ2は、目標リストの先頭要素「2」と自ノードの属性「2」とを比較する。その結果、双方が一致するので、探索メッセージ内の変数 Tree11 を、隣接ノード数個（2個）の、自ノード名と新たな変数とを含んだ木構造データ t(n2,Tree21),t(n2,Tree22) で書き替えて、隣接ノード（ｎ1、ｎ4）に先頭要素「2」を除いた目標リスト [3] を含む探索メッセージを渡して、さらに探索を進める。この時、ｎ1への探索メッセージ内の変数を Tree21 に、ｎ4への変数を Tree22 に、それぞれ設定する。同様に、探索メッセージを受け取ったノードｎ3は、目標リストの先頭要素「2」と自ノードの属性「2」とを比較する。その結果、双方が一致するので、探索メッセージ内の変数 Tree12 を、隣接ノード数個（2個）の、自ノード名と新たな変数とを含んだ木構造データ t(n3,Tree23),t(n3,Tree24) で書き替えて、隣接ノード（ｎ1、ｎ4）に先頭要素「2」を除いた目標リスト [3] を含む探索メッセージを渡して、さらに探索を進める。この時、ｎ1への探索メッセージ内の変数を Tree23 に、ｎ4への変数を Tree24 に、それぞれ設定する。これらｎ2の処理とｎ3の処理は、それぞれの結果が他に影響しないので、同時に行なうことができる（図7－2 (4)）。ｎ2から探索メッセージを受け取ったノードｎ1は、目標リストの先頭要素「3」と自ノードの属性「1」とを比較する。その結果、双方が一致しないので、探索メッセージ内の変数 Tree21 を、自ノード名と空リストとを含んだ木構造データ t(n1,[]) で書き替えて、探索を終了する。一方、ｎ2から探索メッセージを受け取ったノードｎ4は、目標リストの先頭要素「3」と自ノードの属性「3」とを比較する。その結果、双方は一致するが、先頭要素「3」を除いた目標リストは空リスト [] となるので、解の1つがみつかったことになる。そこで、探索メッセージ内の変数 Tree22 を、

間のフィードスルー線を配線する。以上の順序でセル内配線を行う理由は次の通りである。

セルの外周コネクタは、隣接セルの間で共有されている。仮に各セルが自由に外周コネクタにアクセスしコネクタ位置を決定しようとすると、このコネクタを共有する隣接セルでは待ち状態になり、各セルが並列に配線処理を行う上でのタイムロスを生じる。各セルが上記の順序を守ればこれを回避できる。また、実コネクタ配線をフィードスルー配線以前に行う事により、位置不定の外周コネクタを実コネクタからより直線的（最短）な位置に固定できる。

（ｃ）セル内配線

セル内配線は、コネクタ間の配線経路を迷路法により確定する機能で、準備と実行のフェイズがある。

（配線準備）：　配線準備機能から要求され、セル内にMRPN（メーズルータプロセスネットワーク）を敷設する。NPNトランジスタ上へのMRP敷設状態を図３－３に示す。MRPNの天元プロセスは全MRPに要求を送出し、その解を得ることができる。

周辺仮想コネクタMRPは外周辺上の仮想コネクタ点を表現する。これらの各MRPは自分の位置の配線層状態として、進入可能ネット名や配線通過コストを保持している。

セル内の南辺と東辺の最外郭MRPをスルーホール敷設禁止とすることで、隣接するセルにスルーホールがある場合のスルーホール近接チェックを不要としている。

（配線実行）：　配線機能から要求され、セル内の配線経路を解として報告する。配線要求は前述のセル内信号配線で記したように、南ー東ネット、北ー西ネット、フィードスルーネットの順序で計画がなされ、迷路法実行開始時には配線すべきネットのコネクタセットと配線結果応答用変数がメッセージとして、セルプロセスより天元プロセスに渡される。

天元プロセスは始点となるコネクタのMRPに、終点となるコネクタのMRPまでの経路探索を要求する。経路探索は、ICOTで開発された最適パス探索アルゴリズムを使用する。ただし、多点間パス探索、配線層考慮、屈曲防止、MRPN再利用可能化等の改良を施した。

仮想コネクタMRPは、外周コネクタが位置不定で辺上に存在可能範囲のみを持つ場合に、存在範囲に対応する辺上のMRP群からのパス探索の終端点の役割を果たす。存在可能範囲からはずれる辺上MRPは、あらかじめ仮想コネクタMRPへパスメッセージを送出できなくすることで、存在可能範囲を指定することができる。

配線すべきコネクタ点群に対応するMRPのひとつから他の全MRPに経路探索メッセージを送出し、最適通過パスを求める。全MRPへの最短パスが求まったとき探索が終了する。これをアドレス変換し配線結果として要求元に報告する。

また既に、配線がある層を通過したMRPにはその層の通過障害コスト値に大きな値を加算し、以降の他ネット経路探索時にメッセージ通過を困難とさせることにより配線短絡を防止している。

３．２．６　テンプレートライブラリ

レイアウト処理中に利用するライブラリ群であり、以下の要素で構成される。

自ノード名を持たせる。属性としては、トランジスタ、抵抗、ダミーノード等が区別できるように、ｔｒ、ｒｅｓ、ｄｕｍ等を与える。このようにすると、図７－５に示す回路のノード表現は、下記のようになる。ここで、ベクタの第１引数はハッシングのキーである。

```
{r1,node([dum1,dum7],res,r1)}
{r2,node([dum1,dum4],res,r2)}
{r3,node([dum4,dum5],res,r3)}
{r4,node([dum4,dum6],res,r4)}
{r5,node([q5,dum12],res,r5)}
{r6,node([q6,dum12],res,r6)}
{r7,node([q7,dum13],res,r7)}
{r8,node([q8,dum14],res,r8)}
{q1,node([dum5,dum8,in1],tr,q1)}
{q2,node([dum6,in2,dum8],tr,q2)}
{q3,node([dum2,dum9,dum5],tr,q3)}
{q4,node([dum3,dum11,dum6],tr,q4)}
{q5,node([dum7,dum10,r5],tr,q5)}
{q6,node([dum8,q7,r6,dum10],tr,q6)}
{q7,node([dum9,q8,r7,q6],tr,q7)}
{q8,node([dum11,r8,q7],tr,q8)}
{vcc,node([dum3],vcc,vcc)}
{gnd,node([dum14],gnd,gnd)}
{in1,node([q1],in,in1)}
{in2,node([q2],in,in2)}
{out1,node([dum9],out,out1)}
{out2,node([dum11],out,out2)}
{dum1,node([dum2,r2,r1],dum,dum1)}
{dum2,node([dum3,q3,dum1],dum,dum2)}
{dum3,node([vcc,q4,dum2],dum,dum3)}
{dum4,node([r4,r3,r2],dum,dum4)}
{dum5,node([q3,q1,r3],dum,dum5)}
{dum6,node([q4,q2,r4],dum,dum6)}
{dum7,node([dum10,q5,r1],dum,dum7)}
{dum8,node([q2,q6,q1],dum,dum8)}
{dum9,node([out1,q7,q3],dum,dum9)}
{dum10,node([q6,q5,dum7],dum,dum10}
```

HeadとBody_guardsの双方がtrueとならないと、'|'の右側のBody_goalsの処理には入れない。即ちそのような節は、親側のゴールのリダクションには利用できないか（HeadとBody_guardsのいずれかがfailの時）、あるいはサスペンドモードに入り新たな情報によって処理の再開が可能となるタイミングを待つ。以下ではBody_guardsに、いかなる述語の定義も可能であると仮定する。ＩＣＯＴの核言語ＫＬ１では指定された述語しか記述可能でない。この規約を守ると記述量が多くなるのでこのような仮定を設ける。

４．２　統括機能

４．２．１　最上位ゴール節

?-system(UserStr,DispStr).

最上位ゴール節は、UserStrストリームとDispStrストリームを持つ。前者はユーザからの入力処理、後者は端末装置への出力表示用ストリームである。

４．２．２　system/2

merge/2はストリームを合流し、top_module/5への入力ストリームInStrを作成する。
select/2はtop_module/5からの出力ストリームを分流する。
terminal/2,xy_plot/1,save_file/2,circuit_file/1'は各々のファイルプロセスである。
top_module/5は、問題解決過程全体をコーディネイトするプロセスである。
circuit_fileプロセスは、回路ネットデータをCDStrストリームに設定することでtop_moduleに送出する。
circuit_fileプロセス動作終了後、UserStrよりtop_moduleへ回路木生成、配置、配線等の実行指示メッセージが各々通信プロトコルに従って送出され、top_moduleはそれらを実行する。レイアウト結果はOutStrを通じて出力され、select/2によってdisplay,xy_plot,save_fileに対する出力に分流され、各々の装置に出力される。
マージされるTMは下位モジュールからの通信ストリーム群で、プロセス生成時に登録される。

```
system(UserStr,DispStr):-
    merge([UserStr,SvStr,CDStr],InStr),
    select(OutStr,[DispStr,GraphStr,StStr]),
    terminal(UserStr,DispStr),
    xy_plot(GraphStr),
    save_file(StStr,SvStr),
    circuit_file(CDStr),
    merge(TM,FmModules),
```

問題が存在する中で、並列推論マシンの莫大な計算力に見合うものとして、ＬＳＩレイアウト問題を取り挙げた。ＬＳＩレイアウトでは、莫大な回路素子をできるだけ面積を少なく配置するとともに、素子間の配線を行う。素子配置場所や配線経路は莫大な可能性を持つとともに、レイアウトルールと呼ばれる素子や配線の相互距離制約が存在し、問題をきわめて難しいものにしている。

８．１．２　第５世代技術への期待

ＬＳＩレイアウト問題解決のために、従来から各種のアルゴリズムが開発されてきた。特に有名なものに配線のための迷路法があるが、製造技術の進歩による配線可能格子数、配線層数、回路素子数の増大のために計算時間が莫大となり、並列を含めた抜本的改良が広く求められている。

次に、レイアウトの質を高めるためには、従来のようにチップ面積とか総配線長といった数値表現可能量のみを目的関数とするのではなくて、素子や属性を考慮したレイアウトを作成する機能が重要である。記号処理の得意な第５世代技術はこの目的に有用である。最後にソフトウエア開発保守に問題がある。ＬＳＩハード技術の進歩に比較してソフトウエア技術の進歩は遅い。レイアウトプログラムもその例外ではなく、莫大な規模のプログラムを、多くの人手をかけて開発保守しているのが現状である。第５世代技術が提供する並列オブジェクト指向言語等の新しいプログラミングパラダイムはこの問題の解決に有効である可能性を持っている。

８．２　開発方針

８．２．１　目標と方針

ＬＳＩレイアウト問題の複雑性を低減させるために各種のアイデアが利用されてきたが、問題点も有している。

（１）階層処理

分割統合原理により、回路を階層的に分割して小問題に分け、各々をレイアウトした後に統合する。階層数は３～５段程度である。

回路は本来ネットワーク構造を持つため、生成された小問題の間で形状や配線接続部を相互整合させることが望まれるが、逐次処理ではこれは難しい問題となる。そこで、子モジュール間に配線チャネルと呼ばれる一種のバッファを設けて、チップ面積の増大によって不整合を吸収している。

（２）ライブラリの利用

使用頻度の高い回路機能ブロックをあらかじめセルとして用意しておき、これを用いて全体のレイアウト作成時間を大幅に短縮する。

（３）自由度の削減

セル配置場所の限定、配線用格子やアルミ層に応じた走行方向限定等によって相互干渉を排除

Name::=NetName

Type::=power/signal/・・・

Gates::=[{Attr,ObjName},/・・・]

Attr::=collector/base/emitter/anode/cathode/・・・

Props::=[vcc/gnd/in/out/・・・]

Constr::=[SubConstr|_],制約条件リスト

SubConstr::={pair,A,[E1,E2,E3,E4],[{pair,A,[E1,E2],[]},{pair,A,[E3,E4],[]}]}/etc...::=E1とE2がランクAのペア、E3とE4がランクAのペアで、それぞれがランクAのペアであることを示す。

4．3．2　入力階層化機能モジュールデータ

MD::={Name,Type,Neighbours,Props,Others,ToBroad,ToHigher,ToNeighbours,Work}.

Name::=ObjName/NetName

Type::=block/transistor/resistor/vcc/gnd/signal/・・・

Neighbours::=[{Attr,NeigName}|_]

Attr::=collector/base/emitter/anode/cathode/・・・

NeigName::=ObjName/NetName

Props::=[{npn_tr(1),SubProp}]/[{br(void,void),SubProp}]/・・・

Others::={Shape,Path,Point,BlcokName,SonsName}

Path::={N,W,S,E},東西南北からの距離指標値

Point::={I,J}，2次平面上の縦横位置

BlockName::=lu/lb/rb/ru

ToBroad::=下位回路への放送ストリーム

ToHigher::=上位回路へのストリーム

ToNeighbours::=隣接回路へのストリーム群

4．3．3　レイアウト機能モジュールデータ

LM::={Name,Shape,LFName,Type,Status,CProc,Father,Sons,Nets,Conts,Layers,
Vector,MazeRouter,Neighbour}

Name::=ModuleName

Shape::={X0,X1,X2,X3,Y0,Y1,Y2,Y3}

Xi/Yi::=スライス構造の分割点座標

LFName::={Prop,SubLFName}

Prop::=block/cell

（1）小規模擬似並列レイアウト問題解決の研究

中期までは、計算機室への機器レイアウト問題を取り上げ、逐次型推論マシンＰＳＩ及び核言語ＥＳＰを用いた問題解決研究を行ない、後期の本格的研究への準備とすること。

（2）並列レイアウト基本アルゴリズムの開発

後期初年度では、並列レイアウト機能をmulti-PSI上に原理レベルで実現し、以降の２年間の研究開発の基礎作りをすること。

（3）問題解決能力の向上

後期２年度では基本アルゴリズムの並列処理能力を高めるとともに、レイアウトシステムとしての諸要素の基本開発を行うこと。

（4）システムの完成

最終年度では、上記で開発した諸要素の機能を高めるとともに、レイアウトシステムとして統合する。統合したシステムによって、実規模の回路を用いたレイアウト実験を行い、機能・性能を把握する。また、プログラム記述をできるだけエレガントな並列処理形式に近づける。

開発方針に従う並列レイアウト基本アルゴリズムの開発において、プログラム動作の不具合を発見しやすいように集中データ分散プロセスアーキテクチュアを採用した。これにより、Co-HLEXの基本機能の完成を効率化できたが、素子数をNとしたときの処理速度はO(N＾2)以上と遅く、４００素子以上の処理は不可能であった．このため、問題解決能力の向上において、分散メモリ分散プロセスアーキテクチュアに変更した結果、O(N)程度の高速性が実現できた。

８．３　成果概要

８．３．１　実施内容

上記の開発方針を、各年度で次のように実施した。

（1）前期～中期

核言語ＥＳＰの上に計算機室レイアウト問題解決システムを開発し、数１０個の機器の配置配線機能を実現した。部屋をいくつかのゾーンに分割してそれらを擬似並列処理させる方式とした。

（2）平成元年度

レイアウトのための階層再帰並列協調算法HRCTLを開発し、これを核としたレイアウトシステムCo-HLEX第１版を実現した。

周知のように並列プログラミングにおいては、並列動作する多数のプロセスの存在によって、プログラム動作の不具合を発見しても、その原因を探ることが大変困難である。デッドロック現象等はその典型である。

並列プログラミング固有のこの問題を回避し、Co-HLEXの基本機能の完成を効率化するために、集中データ分散プロセスアーキテクチュアを採用した。

（3）平成２年度

Prop::={}

RangeFlag::={Flag1,Flag2},

Flagi::={idle/{inspec,CName}/{n_down,CName}/{f_plan,CName}/{f_real,CName}

4．4　回路ネットのプロセス化

回路ネットのプロセス化および回路ネットの階層化処理は、階層化すべきモジュールプロセス間において、配下への要求放送と、上位への報告および要求通信により実現している。
放送は配下の全モジュールが同一のメッセージを受信し、受理するか否かはモジュール自信が決定する。
上位への報告・要求通信は、放送による実行結果等を上位に返送するもので、下位モジュールと上位との間の個別のメッセージ路を通じて行う。
放送と報告・要求通信は順序性があり、ｅｎｄフラグを用いたメッセージの到着管理による順序性保存を行っているが、本項では機能説明に重点を置くため、ｅｎｄフラグの記述は説明から除いた。順序性保存の方式については後述する。

4．4．1　回路ネットデータのプロセス化

top_module/5は、circuit_file/1から読み込まれた回路ネットデータをもとに、module/4という回路ネットモジュールをプロセスとしてシステム内に生成する。
第1節はオブジェクト（素子）、第2節はネットのプロセス化を行う。トップモジュールから放送ストリーム(ToModules)を取り出し、生成モジュールの第1引数として設定する。全ての生成モジュールは共通のToModulesを持ち、トップモジュールからの放送を受け取ることが可能となる。
生成モジュールの第2引数は生成モジュールのさらに下位モジュールからの通信ストリームであるが、この2述語で生成されるのは末端モジュールなので[]となっている。
第3引き数のFmNeighboursは入力側回路ネットストリームであり、近隣モジュールからのメッセージを受信する。ネットワークの双対表現方式を用いるため、素子の近隣はネットとなり、ネットの近隣は素子となる。生成モジュールから上位モジュールへのメッセージストリームToTopをFmModulesに追加するための登録を変数TMに行う。図4－1（1）は説明用回路例である。

```
top_module([{object,Name,Type,Neighbours,Props}|TMS],
        OutStr,ToModules,FmModules,TM):-
   module(ToModules,[],FmNeighbours,{Name,Type,Neighbours,Props,Others,[],ToTop,
    ToNeighbours,Work}),
   TM = {ToTop,TM1},
   top_module(TMS,OutStr,ToModules,FmModules,TM1).
```

本算法は、ネットワーク形式を持つ原問題の木構造上の再帰算法による近似解法となっており、近似によって欠失する隣接ノード間の情報切断をノード間の通信によって回復するものである。並列処理の今後の発展の中で、この計算モデルは広範な適用領域を持つであろう。

（2）新算法の有用性の検証

HRCTL算法を内蔵したＬＳＩレイアウトシステムCo-HLEXを開発し、バイポーラ・アナログ回路を用いたレイアウト実験を行った。並列推論マシン上でのCo-HLEXの速度は回路の素子数Nに対してO(N)程度であり、従来の逐次処理マシン上で常用されてきた迷路法配線を核とするレイアウトプログラムの速度O(N^2)よりも相当高速である。並列走行中の異プロック間通信によって配線用チャネル（一種のバッファ）を不要とした点も従来にない新たな特徴である。

（3）レイアウトＣＡＤを例としたソフトウエア開発への新アプローチの有効性実証

本研究では、ストリーム並列プログラミングモデルによる問題解決システムのモデル化を行った。局所メモリーを有するプロセスが通信路を介してコミュニケーションすることによって計算を進めるものであり、並列分散オブジェクト計算モデルと言ってもよい。この方法によって対象記述のモジュラリティが高まり、またプログラマの思考とプログラミング結果との間のセマンティックギャップが大いに縮小されることを経験できた。プログラムの修正も比較的容易であった。再帰原理による問題解決を採用したこともあって、Co-HLEXのレイアウト機能はKL1で6,000行程度であり、従来の手続き型言語による10^5行から10^6行の規模に対して著しく小さく、思想表現の手段としての有望性を持っている。

（4）並列推論マシンの機能実証

第５世代計算機プロジェクトの成果である並列推論マシン、その上で動作する核言語KL1を用いて、計算機応用を期待される問題の内でももっとも困難なものの一つであるＬＳＩレイアウト問題の解決を試み、上述したような新機能、高性能、高記述性を実現できた。このことから、本研究の目的である並列推論マシンの機能実証に成功したと考える。

８．４　残された重要課題

本報告執筆時点（平成４年３月下旬）ではCo-HLEXのmulti-PSIの16, 32, 48, 64台上での走行実証を行った段階であり、PIMでの実験結果を得てはいないが、multi-PSI上での経験からみて良好な結果を得ると見通している。Co-HLEXの保有するレイアウトフレームすなわち計画用の各種テンプレートは、バイポーラ・アナログ回路用のものである。他の回路系のレイアウトは、それ向きのテンプレートの追加によって可能となると考えられる。

また本研究では、２次元レイアウト問題(詳しくは配線層が３次元方向に層をなす疑似３次元)を取り扱ったが、３次元レイアウト問題への拡張性も備えている。なお、本システムを本格実用化するには、製造プロセス固有の事情をCo-HLEXのテンプレート等に反映させることと、対話処理機能の追加が必要であろう。これらは現在までに得られた成果の延長線上にある課題である。

```
    module(FBrt,[],FmNets2,MD2),
    module(FBrt,[],FmGates,MD3).
```

4．4．3　最上位回路モジュールプロセスの生成・挿入

最上位回路モジュールは回路木の根に対応するプロセスであり、チップ全体のレイアウト計画の外形状を保持している。最上位の回路モジュールを、トップモジュールと上記で作成した回路ネットモジュールの間に挿入するストリーム接続変更を行う。

```
top_module([{create_highest_module,CircuitData,end-Eo}|TMS],
        OutStr,ToModules,FmModules,TM):-
    CircuitData = {Name,Type,Neighbours,Props,Others},
    module(ToHiModule,FmModules,FmNeighbours,
        {Name,Type,Neighbours,Props,Others,ToModules,ToTop,ToNeighbours,void}),
    Eo = end,
    top_module(TMS,OutStr,ToHiModule,ToTop,TM).
```

4．4．4　回路ネットモジュールプロセス間ストリーム接続

回路ネットストリームは、素子とネットの接続関係をストリームで表現したものであり、これにメッセージを流すことで回路の信号伝播経路等を分析する。図4－1（2）にストリーム形成後のモジュールプロセス状態を示す。ここで生成するストリームは全て入りと出の双対となっているが、図では1ストリーム路で簡略表現している。

ストリームを介して、隣接モジュールの情報がどのモジュールでも取得できるが、信号伝播経路分析メッセージについては、抵抗等の受動素子は双方向に可能であるが,トランジスタ等の能動素子は、ベース側からコレクタやエミッタへの伝播は認めるが逆方向は認めないなどの制約を設けている。

以下、top_module/5は配下回路モジュールにモジュール間の回路ネットストリーム生成を指示するものである。module/4の第1節は、当回路モジュールがblockタイプのときは、自分は何もせず、配下回路モジュールにストリーム生成メッセージを伝えることを示す。第2節は当モジュールがblockタイプ以外の場合で、回路ネットストリームの生成を行う。

回路ネットモジュールが素子の場合は接続されるネットの数、回路ネットモジュールがネットの場合は接続される素子数分の入力ストリームが生成される。生成したストリームをマージして自回路ネットモジュールのメッセージ入力ストリームとし、相手側回路ネットモジュールへマージ元ストリームの隣接出力ストリーム登録を上位回路モジュールに要求する。

```
top_module([{make_module_net,end-Eo}|TMS],OutStr,ToModule,FmModule,TM):-
    ToModule = [{make_module_net}|TBrt],
    top_module(TMS,OutStr,TBrt,FmModule,TM).
```

[Watanabe 91] Watanabe,T.and Komatsu,K.,Co-operative Hierarchical Layout Problem Solver on Parallel Inference Machine,LPC'91,pp9-24 (1991).

[Watanabe 92] Watanabe,T.and Komatsu,K.,Co-HLEX:Co-operative Recursive LSI Layout Problem Solver on Japan's Fifth Generation Parallel Inference Machine(to appear), FGCS'92 (1992).

相手側回路ネットモジュールが隣接出力ストリーム登録要求を受信

第１節は要求されたモジュールが受信した場合で、回路ネットストリームの隣接出力ストリーム登録を行う。第２節は名称Nameの異なるモジュールが受信した場合で、要求を無視する。

```
module([{hand_in_hand,Name,OutMes}|TBt],FmBelow,FmNeighbour,MD):-
    MD  = {Name,Type,Neighbour,Props,Others,ToBroad,ToTop,ToNeighbours,Work} ;
    ToNeighbours1 = [OutMes|ToNeighbours],
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,ToTop,ToNeighbours1,Work},
    module(TBt,FmBelow,FmNeighbour,MD1).
otherwise.
module([{hand_in_hand,Name,OutMes}|TBt],FmBelow,FmNeighbour,MD):-
    module(TBt,FmBelow,FmNeighbour,MD).
```

４．５　回路ネットの階層化処理

与えられたチップの外形および外端子のもとで、平板な回路ネットを構成する素子のN,W,S,E（北、西、南、東）辺からの距離を求めた後、回路ネットを２から４個の子回路に分割する。生成した子回路には、外形状を実現できる子回路形状を与える。

分割境界線上を横切るネットには仮想コネクタを設定する。これによって、各々の子回路は、外形状と外端子を持つこととなり、各々を並列かつ再帰的に分割できるようになる。この処理を回路内素子が一つになるまで繰り返す。

トップモジュールは最上位回路モジュールに配下回路木生成を要求する。

```
top_module([{gen_module_tree,Circuit,end-Eo}|TMS],OutStr,ToModule,FmModule,TM):-
    ToModule = [{gen_module_tree,Circuit,_}|TMt],
    top_module(TMS,OutStr,TMt,FmModule,TM).
```

４．５．１　配下回路木生成

第１節は、最上位回路モジュールまたは複数素子保有回路モジュールの場合であり、以下の機能命令メッセージを放送し回路分割を行い回路木生成する。

{add_dir_net,Name}::=仮想端点ネットモジュールからエッジコネクタモジュールへのストリーム追加要求メッセージ

{create_dir_net,Name,Place}::=仮想端点モジュール生成要求メッセージ

{investigate_path,Name}::=信号伝播経路分析要求メッセージ

{kill_dir_net,Name}::=仮想端点モジュール消去要求メッセージ

{prepare_partitioning,Name,Vcost}::=回路分割準備要求メッセージ

待ち無しで部分問題間の制約を満足化させる（一種の事前常識の導入）。
（ｓ３）最後に仮端子間を配線し、全体の解を構成する。

２．２　システム概要

本実験システムの概要を、下記の図、表に示す。
図１　計算機室レイアウトシステムの機能構成
図２　計算機室レイアウトシステムのプロセス構成
図３－１　計算機室レイアウト実験画面（機器配置完了）
図３－２　計算機室レイアウト実験画面（配線端子決定）
図３－３　計算機室レイアウト実験画面（配線完了）
図３－４　機器配置処理タイムチャート
表１　計算機室レイアウトシステムの機能概要
以下、図２を用いて、各オブジェクトによる問題解決過程を説明する。
（ｏ１）並列問題解決オブジェクト
問題解決全体の管理を分担する。利用者の指示により問題定義オブジェクトより部屋や機器のデータを読み込み上位階層処理を実行した後、下位階層処理を並列実行し、再度上位階層配線を実行する。
（ｏ２）上位配置処理オブジェクト
部分問題の生成を行なう。機器のグルーピングと部屋のゾーニングおよびグループのゾーン割り当てを上位階層用ルールオブジェクトを用いて行なう。
上位階層用制約評価オブジェクトはゾーン割り当て時の制約条件を与える。ゾーン形状等は、配置及び配線状況記憶オブジェクトに記憶される。
（ｏ３）並列配置プロセス制御オブジェクト
ゾーン個数分の部分問題を下位配置配線オブジェクトとして生成した後、それらに機器配置を並列実行させる。仮端子配置が最後に行なわれる。下位階層用ルールオブジェクトは配置及び配線状況記憶オブジェクトに記憶された既配置機器の配置状況に応じた機器配置候補地の生成に、下位階層用制約評価オブジェクトは候補地からの適正配置位置の選択にそれぞれ用いられる。下位階層用スライド調整オブジェクトは既配置機器の移動による配置スペース確保を行なう。
（ｏ４）並列配線プロセス制御オブジェクト
下位配置配線オブジェクトを再度並列実行させ、ゾーン内の配置済み機器ケーブルの接続点や仮端子の間に電源線や信号線を敷設する。
機器仮想コネクタ間配線オブジェクトは二点間をＬ字パターン等で結ぶ機能を持つパターンルータである。
（ｏ５）上位配線処理オブジェクト

```
        EToNeighbours,EWork},
    Nameo = EName,
    module(FBrt,FmBelow,FmNeighbour,NMD).
```

4．5．2　仮想端点ネットモジュールからエッジコネクタモジュールへのストリーム追加

N,W,S,Eの仮想端点（信号伝播経路分析入力点コネクタとそのネット）モジュールを生成するための準備を行う。

コネクタを表す所在名(n,w,s,e)を配下回路ネットモジュールに放送し、各モジュールは所在名が一致すれば自モジュールの入力側回路ネットストリームと仮想端点ネットモジュールの出力側回路ネットストリーム用変数とをマージし、この変数と自モジュール名、所在名を上位回路モジュールへ送る。上位回路モジュールは送られたデータをワーク部に保存する。

以下、Nモジュールについて記述するが、W,S,Eについても同様に行う。

```
module([{add_dir_net,Name}|FBrt],FmBelow,FmNeighbour,MD):-
    MD  = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work} ;
    ToBroad = [{checkup_place,n},
               {checkup_place,w},
               {checkup_place,s},
               {checkup_place,e}|ToBroad1],
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad1,ToHigher,
          ToNeighbours,{[],[],[],[]}}.
    module(FBrt,FmBelow,FmNeighbour,MD1).

module([{checkup_place,Place}|FBrt],FmBelow,FmNeighbour,MD):-
    MD  = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work},
    Type = {outlet,Place} ;
    merge({AddIStr,FmNeighbour},FmNeighbour1),
    ToHigher = [{match_place,Place,outlet_net,AddIStr}|ToHigher1],
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher1,ToNeighbours,Work},
    module(FBrt,FmBelow,FmNeighbour1,MD1).

module(FmBroad,[{match_place,Place,Attr,AddOStr}|FmBelowT],FmNeighbour,MD):-
    Place = n,
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work} ;
    Work  = {N,W,S,E},
    Work1 = {[{D,void,Attr,Cost,AddOStr}|N],W,S,E},
```

LSI レイアウト例 （Bipアナログの例）

出典 A.B.Grebene：Bipolar & MOS Analog Integrated Circuit Design

Input Circuit Diagram

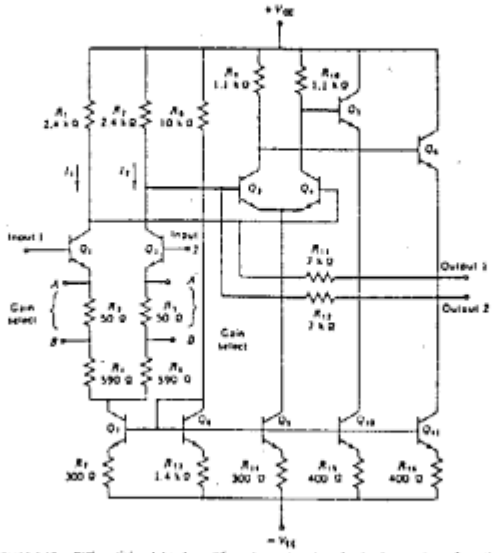


FIGURE 8.17. Differential wideband amplifier using series-shunt feedback cascade configuration (Fairchild μA-733.)

Output Layout Diagram（人手作成）

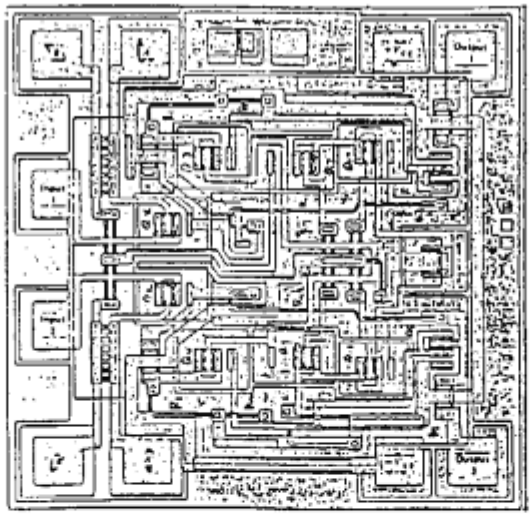


FIGURE 8.19. Photomicrograph of differential-shunt cascade amplifier. Chip size: 43 mils × 43 mils. (Photo: Fairchild Semiconductor.)

図 2-1 回路レイアウト問題

信号伝播経路分析データ初期設定メッセージを放送し、配下モジュールに通過コストの初期設定をさせる。

仮想端点コネクタ(N,W,S,E)モジュールにｇｏメッセージを放送し、信号伝播経路分析を開始する。

メッセージの持つMaxCostは通過コスト最大値で、十分大きな値である。

信号伝播経路分析の終結はショートサーキットにより監視する。

図４－１（１）に回路ネットモジュールのうち素子モジュールが保持している信号伝播経路分析値を示す。

```
module([{investigate_path,Name}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work}|
    ToBroad = [{init_path_dt,End},
              {go,n,MaxCost,end-E1},
              {go,w,MaxCost,E1-E2},
              {go,s,MaxCost,E2-E3},
              {go,e,MaxCost,E3-End}|TBrt],
    MD1 = {Name,Type,Neighbour,Props,Others,TBrt,ToHigher,ToNeighbours,Work},
    module(FBrt,FmBelow,FmNeighbour,MD1).
```

信号伝播経路分析データ初期設定

MaxIntはモジュールの保存する信号到着位置を表すコストの初期値で、整数最大値を用いる。

```
module([{init_path_dt,End}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work}|
    max_interger(MaxInt),
    Work1 = {{void,MaxInt},{void,MaxInt},{void,MaxInt},{void,MaxInt},End},
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work1},
    module(FBrt,FmBelow,FmNeighbour,MD1).
```

信号伝播経路分析起動

N,W,S,Eの仮想端点コネクタモジュールがｇｏメッセージを受理し、信号伝播経路分析メッセージを自モジュールでも受けるための準備を行う。

```
module([{go,Name,MaxCost,end-Eo}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work}|
    message(Name,0,MaxCost,*,Start,end-Eo),
    module(FBrt,FmBelow,[Start|FmNeighbour],MD).
```

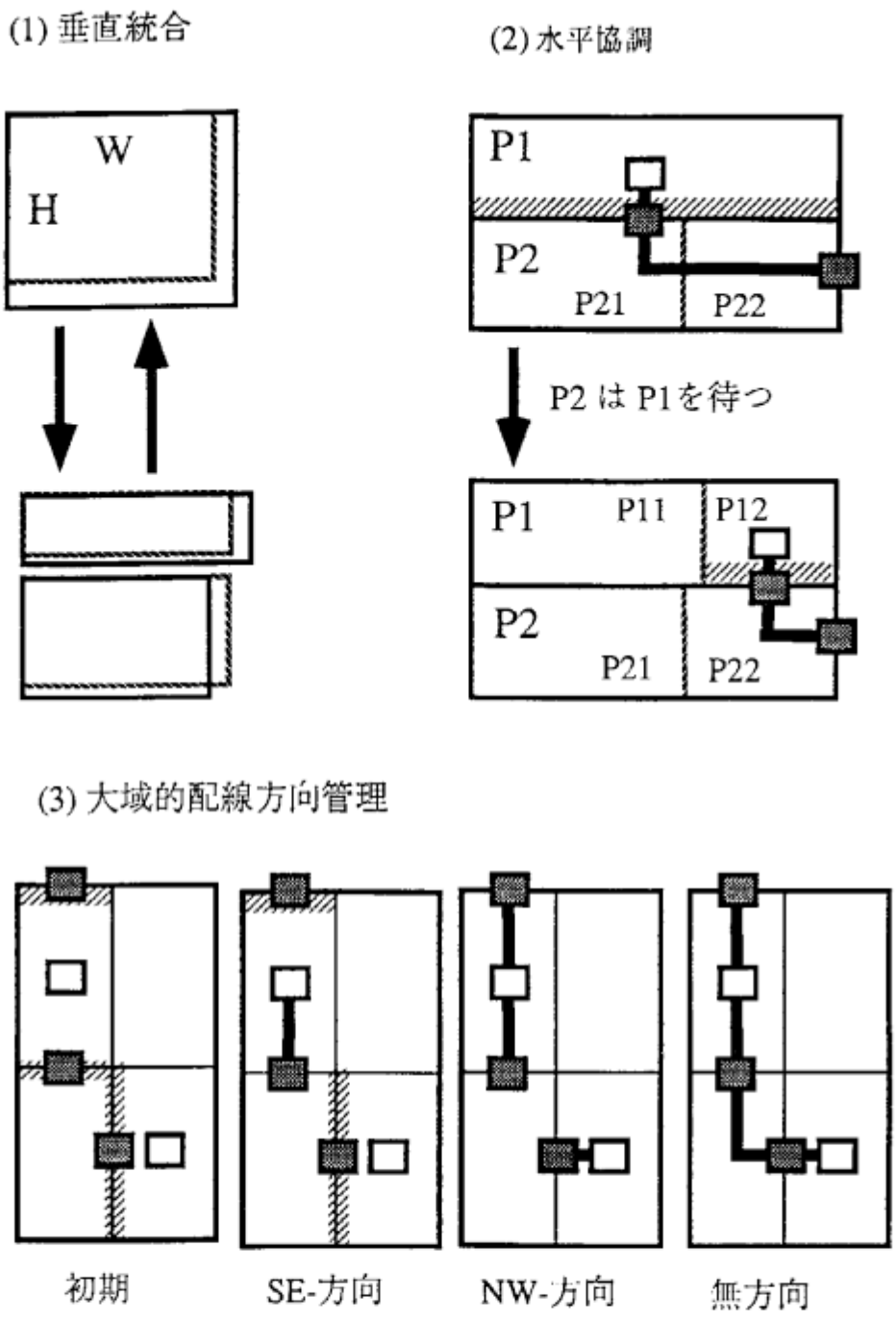


図 2-3 問題解決用ヒューリスティクス

```
make_forwarding_list(_,_,[],ForwardTo,ToNeighbours1):-
    ForwardTo = [],ToNeighbours = [].
make_forwarding_list(Name,{SeName1,Attr},[{SeName2,base},Cost,ToModule]|TNt],
        ForwardTo,ToNeighbours1):-
    Name = SeName1 |
    ToNeighbours1 = [{SeName2,base},Cost,ToModule]|ToNeighbours2],
    make_forwarding_list(Name,{SeName1,Attr},TNt,ForwardTo,ToNeighbours2).
otherwise.
make_forwarding_list(Name,Sender1,[{Sender2,Cost,ToModule}|TNt],
    ForwardTo,ToNeighbours1):-
    ToModule = {ToModule1,ToModule2},
    ForwardTo = [{Sender2,Cost,ToModule1}|ForwardTo1],
    ToNeighbours1 = [{Sender2,Cost,ToModule2}|ToNeighbours2],
    make_forwarding_list(Name,Sender1,TNt,ForwardTo1,ToNeighbours2).
```

回路ネットモジュールに信号伝播経路分析メッセージを送出する機能を優先度を設定し実行する。

```
message(Compass,Cost,maxCost,Sender,ToModule,T0-T,Prio):-
    message(Compass,Cost,MaxCost,Sender,ToModule,T0-T)@priority(*,Prio).

message(Compass,Cost,MaxCost,Sender,ToModule,T0-T):-
    ToModule = [reached(Compass,Sender,Cost,Forwarding)],
    forward_messages(Compass,Cost,MaxCost,Forwarding,T0-T).
```

信号伝播経路分析メッセージを送るためのmessageプロセスを生成する。
このとき、メッセージを送る優先度の決定や、メッセージの持つコストが甚大で無意味であろう信号伝播経路分析メッセージの停止も行う。

```
forward_message(Compass,C0,MaxCost,[Sender,C1,ToModule]|F],T0-T):-
    C := C0+C1 , C =< MaxCost |
    Prio := 4095 * ( MaxCost-C )/MaxCost,
    message(Compass,C,MaxCost,Sender,ToModule,T0-T1,Prio),
    forward_message(Compass,C0,MaxCost,F,T1-T).
forward_message(_,_,_,[],T0-T):- T0 = T.
forward_message(Compass,C0,MaxCost,[Sender,C1,ToModule]|F],T0-T):-
    C := C0+C1 , C > MaxCost |
```

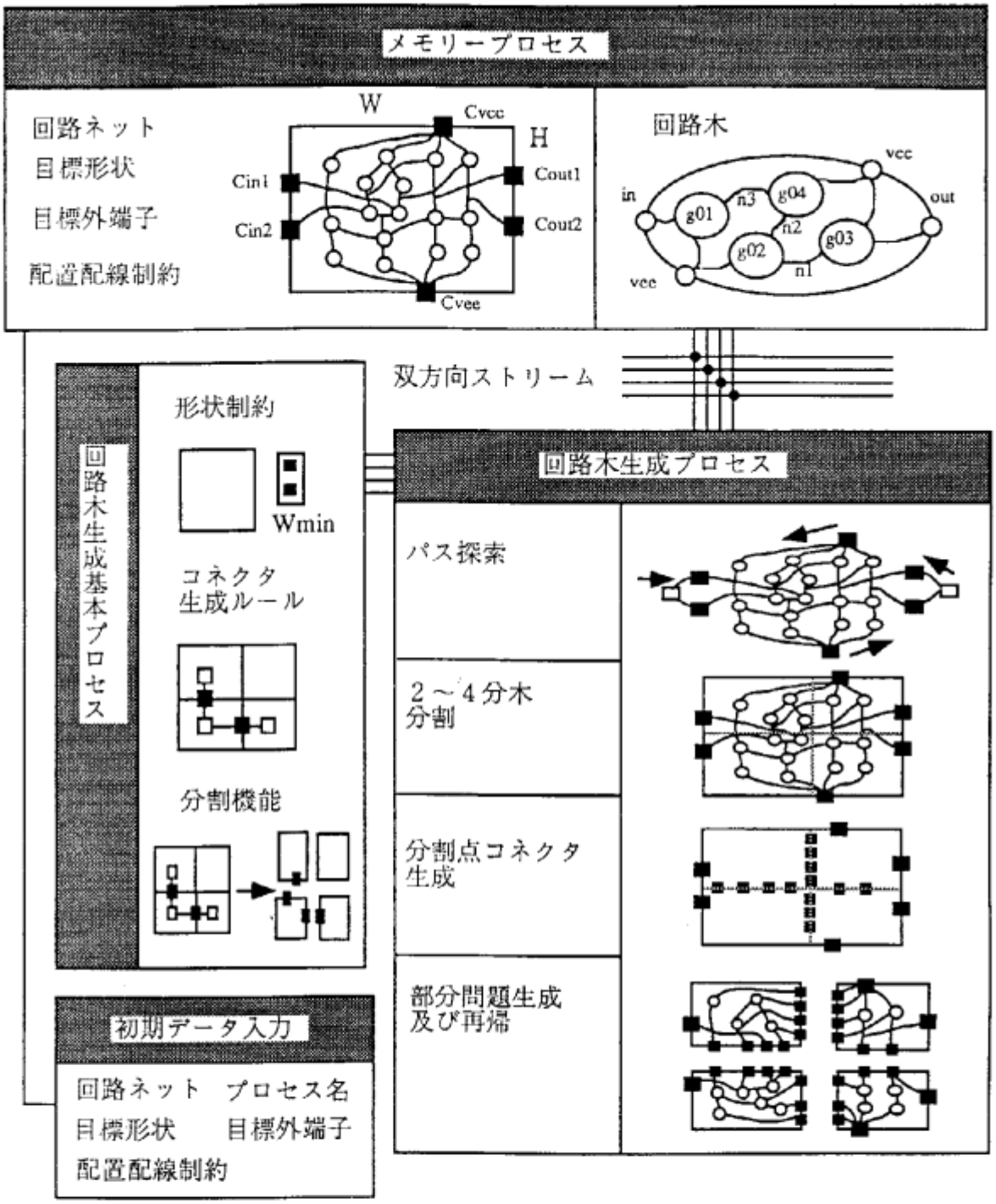


図3-2 回路木生成機能の概要

```
module(FmBroad,[{report_path_value,{Npv,Wpv,Spv,Epv}}|FBet],FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work},
    get_path_range(Others,{Nmax,Wmax,Smax,Emax}) |
    max(Npv,Nmax,Nmax1),
    max(Wpv,Wmax,Wmax1),
    max(Spv,Smax,Smax1),
    max(Epv,Emax,Emax1),
    set_path_range(Others,{Nmax1,Wmax1,Smax1,Emax1},Others1),
    MD1 = {Name,Type,Neighbour,Props,Others1,ToBroad,ToHigher,ToNeighbours,Work},
    module(FmBroad,FBet,FmNeighbour,MD1).
```

4．5．7　モジュール位置決定

素子モジュール位置の決定は、2次元配列面としたときの、縦インデックス(I),横インデックス(J)を求めることで行う。

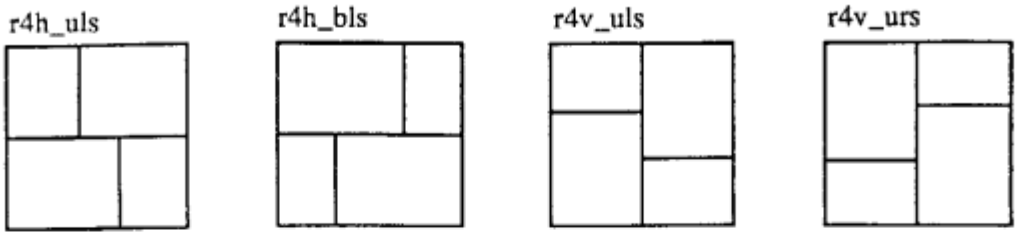
縦インデックスはNとSでモジュールを引き合い、強い側の信号到着値をインデックスとして用いる。横インデックスはWとEでモジュールを引き合い、強い側の信号到着値をインデックスとして用いる。引き合いは各々の方向の信号到着最大値と素子モジュールの持つ信号到着値の比を求め、閾値として対応する方向の値と比較し小さい側を有効とする。

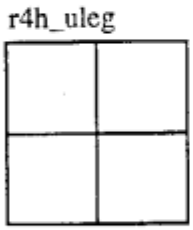
但し、SとEの信号到着値は逆方向インデックスであるので、左／上からのインデックスに変換する。

インデックス値は、自モジュールに保存するとともに、範囲を求めさせるため上位回路モジュールに送信する。

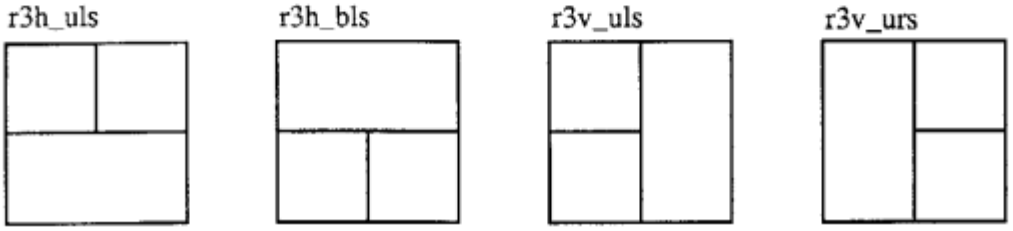
図4－1（1）にモジュール位置決定により得られた、モジュールプロセスの位置インデックス値を示す。

```
module([{decide_sitting,Namei}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,
          {Npv,Wpv,Spv,Epv}},
    object(Type) , not_connector(Type) |
    Nr := Npv/Nmax,
    Wr := Wpv/Wmax,
    Sr := Spv/Smax,
    Er := Epv/Emax,
    or( Nr =< Sr -> I := Npv , Nr  > Sr -> I := Nmax + Smax - Spv + 1 ),
    or( Wr =< Er -> J := Wpv , Wr  > Er -> J := Wmax + Emax - Epv + 1 ),
    set_sitting_point(Others,[I,J],Others1),
```

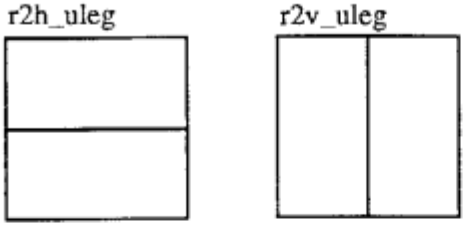




（1） 4分割パターン



（2） 3分割パターン



（3） 2分割パターン

図 3-4(1) レイアウトフレーム（分割パターン）

```
    decide_subblock({I,J},{Ihalf,Jhalf},BlockName),
    set_block_name(Others,BlockName,Others1),
    MD1 = {Name,Type,Neighbour,Props,Others1,ToBroad,TTpt,ToNeighbours,Work},
    module(FBrt,FmBelow,FmNeighbour,MD1).

decide_subblock({I,J},{Ihalf,Jhalf},BlockName):- I =< Ihalf , J =< Jhalf !
    BlockName = lu.
decide_subblock({I,J},{Ihalf,Jhalf},BlockName):- I  > Ihalf , J =< Jhalf !
    BlockName = lb.
decide_subblock({I,J},{Ihalf,Jhalf},BlockName):- I  > Ihalf , J  > Jhalf !
    BlockName = rb.
decide_subblock({I,J},{Ihalf,Jhalf},BlockName):- I =< Ihalf , J  > Jhalf !
    BlockName = ru.
```

配下回路ネットモジュールの要求により、回路モジュール保存の、インデックス値の範囲取得結果を取りだす。

```
module(FmBroad,[{get_site_range,SiteRange}|FBet],FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work} !
    get_sitting_point(Others,SiteRange),
    module(FmBroad,FBet,FmNeighbour,MD).
```

4．5．9　ネット分割

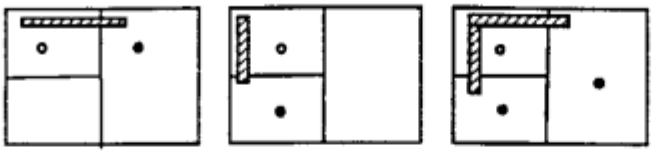
回路ネットモジュールのうちネットのモジュールが放送を受理する。

ネットのモジュールは隣接した素子モジュールに近隣調査メッセージを送り、分画名別に素子名を返答させ、素子の所在位置を調べる。

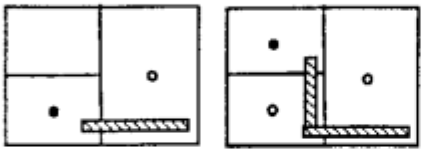
回路分割によりネットの分断が発生するか調べ、分断がなければネットモジュールに素子の存在する分画名を設定する。分断がある場合は、分画の外郭上に設定するコネクタモジュールを生成し、各々の分画でユニークとするためにネットモジュールを分断生成する。

```
module({{divide_cir_net,Namei}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work},
    Type = net !
    report_element_place(ToNeighbours,{[],{},[],[],[],[],[],[]},Placis,ToNeig1),
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeig1,Work},
    take_place_name(0,8,Placis,{n,w,s,e,lu,lb,rb,ru},PlaceNames),
    get(PlaceNames,[lu,lb,rb,ru],BlockNames),
```

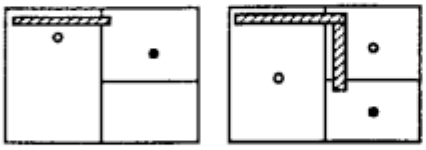
3v_uls,posi



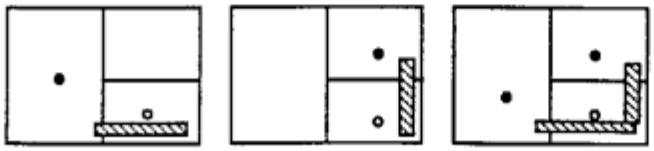
3v_uls,nega



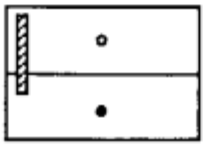
3v_urs,posi



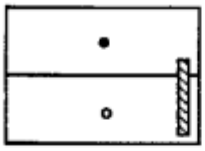
3v_urs,nega



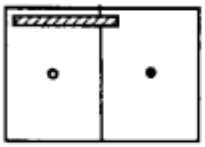
2h,posi



2h,nega



2v,posi



2v,nega

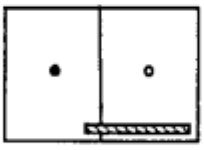


図 3-4(2) レイアウトフレーム（電源配線パターン）

```
module([{create_outlet,Name,FmBroadc,[]}|FBrt],FmBelow,FmNeighbour,MD):-
    module(FBrt,FmBelow,FmNeighbour,MD).
module([{create_outlet,Name,FmBroadc,[BN-Dir|Et]}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work} |
    ToHigher = {ToHigher1,ToHigher2},
    set_block_name(BN,Othersc),
    MDc = {{BN,Dir},{outlet,Dir},[Name],void,Othersc,[],ToHigher1,[{Name,0,_}],
          Workc},
    merge(ToNeigStr,FmNeighbourc),
    module(FmBroadc,[],FmNeighbourc,MDc),
    ToNeighbours1 = [{{BN,Dir},0,ToNeigStr}|ToNeighbours],
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher2,ToNeighbours1,Work},
    module([{create_outlet,Name,FmBroadc,Et}|FBrt],FmBelow,FmNeighbour,MD1).
```

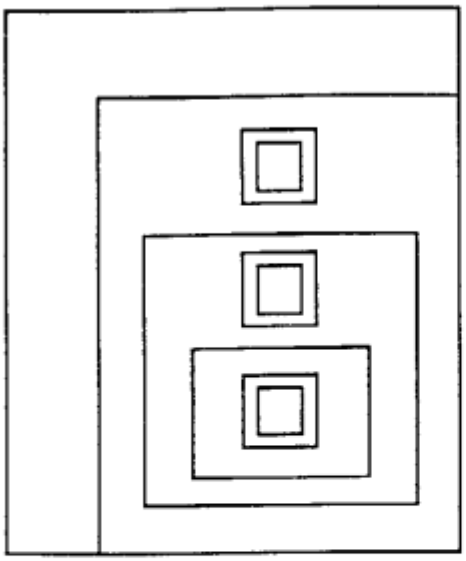
分断ネットモジュール生成

各分画回路内に、独立したネットモジュールを分断生成する。

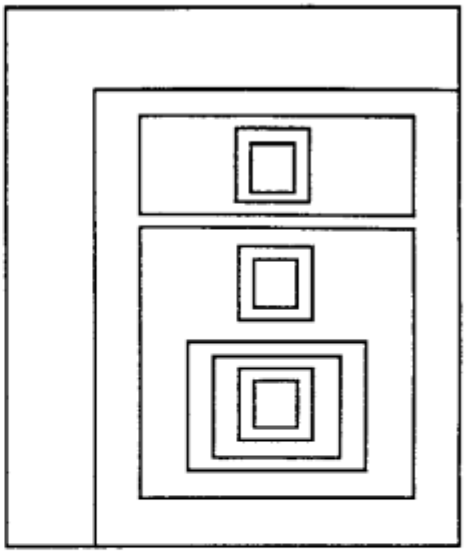
第1節は生成終了の節で、分断元ネットモジュールを自殺させる。

第2節は、各分画回路内に独立したネットモジュールを分断生成するものである。分画内素子モジュールの回路ネットストリームの入力側ストリームを生成ネットモジュールの出力側ストリームと接続させ、分断生成ネットモジュールでは分画内素子モジュールの出力側ストリームを入力ストリームとして登録し、分断ネットモジュールを生成する。

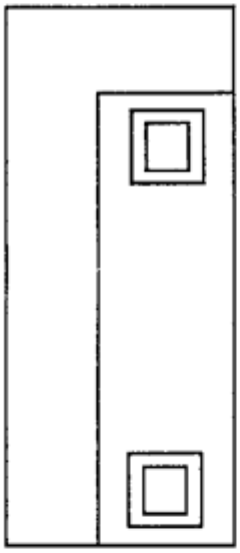
```
module([{divide_net,Name,[],Placis,FBrt}|FBrt],FmBelow,FmNeighbour,MD):- true|true.
module([{divide_net,Name,[BN|Bt],Placis,FBrt}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work} |
    get_block_objs(BN,Placis,Objs),
    purge(Objs,ObjNames),
    merge(FmPath,FmNeigNew),
    replace_bname_path(BN,ObjNames,ToNeighbours,FmPath,ToNeig,OtherPath),
    set_block_name(Others,BN,Others1),
    ToHigher = {ToHigher1,ToHigher2},
    MDn = {Name,net,ObjNames,Props,Others1,ToBroad,ToHigher1,ToNeig,Work},
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher2,OtherPath,Work},
    module(FmBroad,[],FmNeigNew,MDn),
    module([{divide_net,Name,Bt,Placis,FBrt}|FBrt],FmBelow,FmNeighbour,MD1).
```



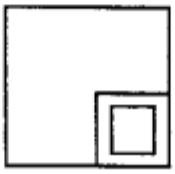
NPNトランジスタ



PNPトランジスタ



抵抗セグメント



島つりコンタクト

図 3-4(4) レイアウトフレーム（素子パターン例）

```
module([{delete_outlet,Name}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work} ;
    delete_outlet(ToNeighbours,ToNeig1),
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeig1,Work},
    module(FBrt,FmBelow,FmNeighbour,MD1).

delete_outlet([],ToNeig):- ToNeig = [].
delete_outlet([{[P1,Name],P2,OutPath}|T1],ToNeig):- Name = or( n,w,s,e ) ;
    OutPath = [{kill,Name}],
    delete_outlet(T1,ToNeig).
otherwise.
delete_outlet([H|T1],ToNeig):-
    ToNeig = [H|T2],
    delete_outlet(T1,T2).
```

所在名リストから分断コネクタ設定位置を求める。

辞書適用し、所在名リストから分断コネクタ設定位置を索引する。例えばPlaceNames = [lu,ru]である場合はOutletNamesに[lu-e,ru-w]が返される。

```
module([{take_net_outlet,Name,PlaceNames,OutletNames}|FBrt],FmBelow,FmNeighbour,
     MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work} ;
    connector_formation(Formations),
    find_outlet(PlaceNames,Formations,OutletNames),
    module(FBrt,FmBelow,FmNeighbour,MD).
```

コネクタ生成辞書

コネクタ生成辞書の一部を示す。

```
connector_formation(Formation):-
    Formation = [{[n,lu,lb],[lu-n,lu-s,lb-n]},
                 {[lu,lb],  [lu-s,lb-n]},
                 {[w,lu,ru],[lu-w,lu-e,ru-w]},
```

回路ネットモジュールのうちネットのモジュールから回路ネットストリームを通じて要求され、素子モジュールプロセスの所在名とモジュール名を返送する。

```
module(FmBroad,FmBelow,[{rep_place,Place,Nameo}|FNt],MD):-
```

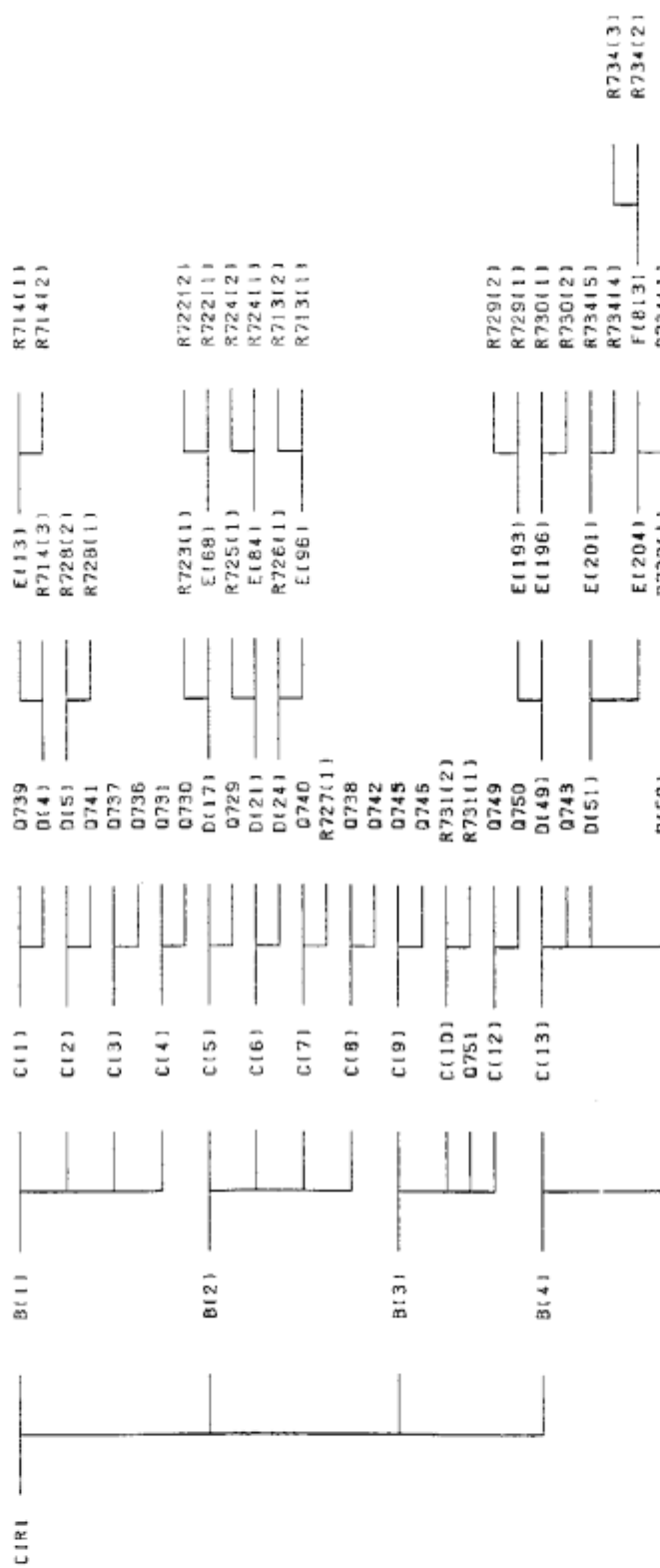

C(R1
B(1)
C(1)
Q739
E(113)
R714(1)
Q(4)
R714(3)
R714(2)
C(2)
D(15)
R728(2)
R728(1)
Q741
C(3)
Q737
Q736
C(4)
Q731
R723(1)
R722(2)
Q730
E(168)
R722(1)
B(2)
C(5)
D(17)
R725(1)
R724(2)
Q729
E(184)
R724(1)
C(6)
D(21)
R726(1)
R713(2)
D(24)
E(196)
R713(1)
C(7)
Q740
R727(1)
C(8)
Q738
Q742
B(3)
C(9)
Q745
Q746
C(10)
R731(2)
R729(2)
Q751
R731(1)
R729(1)
C(12)
Q749
E(193)
R730(1)
C(13)
Q750
E(196)
R730(2)
B(4)
D(49)
Q743
R734(5)
D(51)
E(201)
R734(4)
R734(3)
E(204)
F(8(3)
R734(2)
C(16)
D(52)
R732(1)
R734(1)
Q747
R733(1)
Q748

図 3-5(2) 出力例（回路木）

```
    LuMD = {LuName,LuType,[],LuProps,LuOthers,LuToBroad,LuToHigher,[],LuToBelow},
    LbMD = {LbName,LbType,[],LbProps,LbOthers,LbToBroad,LbToHigher,[],LbToBelow},
    RuMD = {RuName,RuType,[],RuProps,RuOthers,RuToBroad,RuToHigher,[],RuToBelow},
    RbMD = {RbName,RbType,[],RbProps,RbOthers,RbToBroad,RbToHigher,[],RbToBelow},
    merge({FmBelow,LuToBelow},LuFmBelow),
    merge({FmBelow,LbToBelow},LbFmBelow),
    merge({FmBelow,RbToBelow},RbFmBelow),
    merge({FmBelow,RuToBelow},RuFmBelow),
    merge({LuToHigher,LbToHigher,RbToHigher,RuToHigher},NewFmBelow),
    ToBroad  = {{set_tohigher},
                {replace_fmbroad,{LuToBroad,LbToBroad,RbToBroad,RuToBroad}}},
    MD1 = {Name,Type,Neighbour,Props,Others1,SubBroad,ToHigher,ToNeighbours,Work},
    module(FBrt,NewFmBelow,MD1).
```

配下ブロック回路名称および、外形状生成

以下、４分割の場合の記述であるが、２、３分割の場合も同様に行う。

```
make_block_name(square,Name,[LuName,LbName,RbName,RuName]):-
    make_cirname(lu,Name,LuName),
    make_cirname(lb,Name,LbName),
    make_cirname(rb,Name,RbName),
    make_cirname(ru,Name,RuName).

make_block_shape(square,Shape,[LuOthers,LbOthers,RbOthers,RuOthers]):-
    Shape = {X0,X1,X2,X3,Y0,Y1,Y2,Y3},
    LuS = {X0,void,void,X1,Y0,void,void,Y1},
    LbS = {X0,void,void,X2,0,void,void,"(Y3-Y1)},
    RbS = {0,void,void,"(X3-X2),0,void,void,"(Y3-Y2)},
    RuS = {0,void,void,"(X3-X1),0,void,void,Y2},
    LuA := X1 * Y1,
    LbA := X2 * (Y3-Y1),
    RbA := (X3-X2) * (Y3-Y2),
    RuA := (X3-X1) * Y2,
    make_others(lu,LuS,LuA,LuOthers),
    make_others(lb,LbS,LbA,LbOthers),
    make_others(rb,RbS,RbA,RbOthers),
```

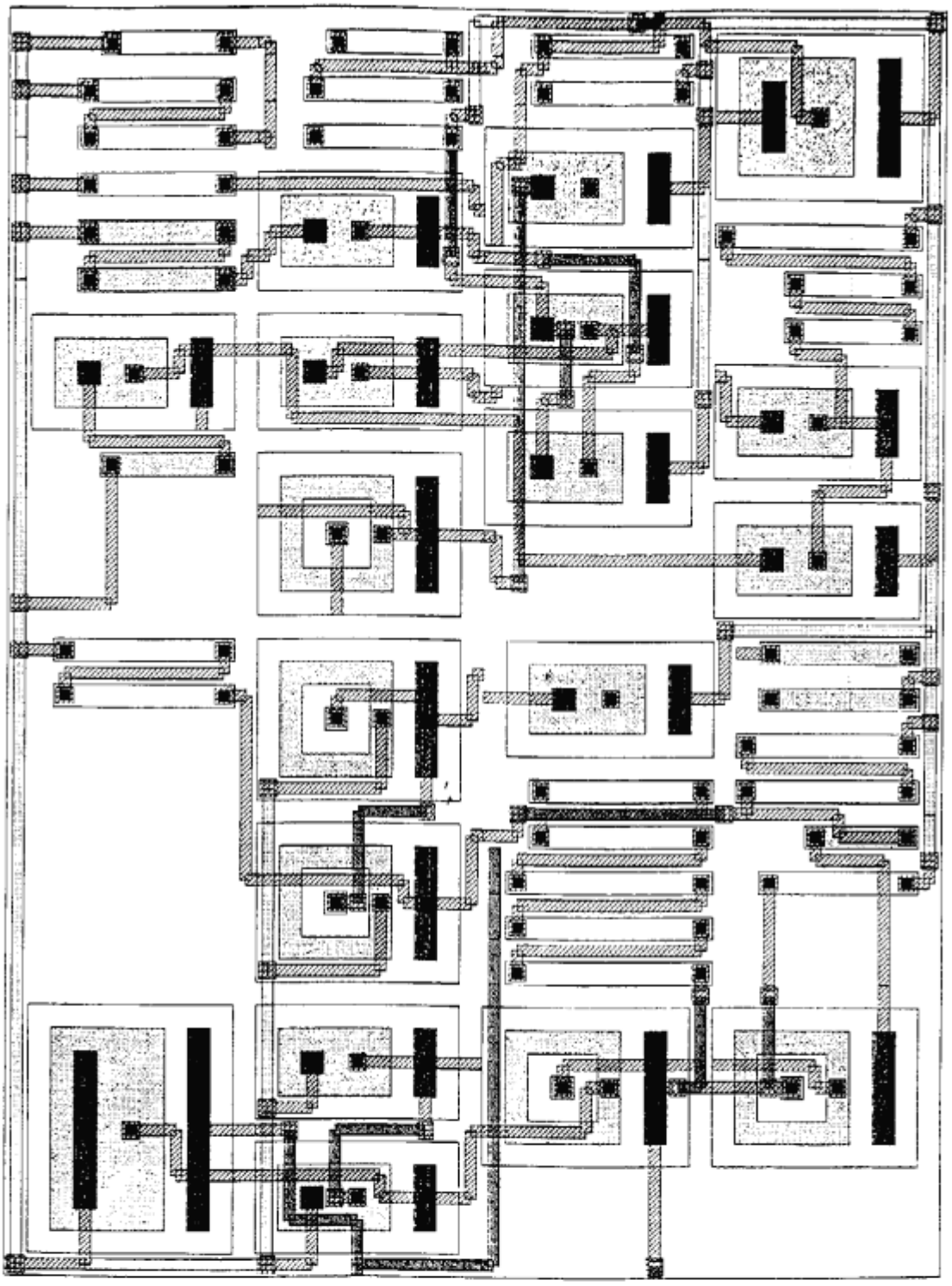


図 3-5(4) 出力例（レイアウト結果）

４．５．１１　配下分画ブロックの回路木生成要求

配下ブロック名称リストを取り出し、各配下分画ブロックの回路モジュールに素子数調査と配下回路木生成要求を送出する。

```
module([{gen_tree_all,Name}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work} ;
    get_sons_name(Others,[Lu1,Lb1,Rb1,Ru1]),
    set_sons_name(Others,[Lu2,Lb2,Rb2,Ru2],Others1),
    ToBroad = [{set_modulality,Lu1},
               {set_modulality,Lb1},
               {set_modulality,Rb1},
               {set_modulality,Ru1},
               {gen_module_tree,Lu1,Lu2},
               {gen_module_tree,Lb1,Lb2},
               {gen_module_tree,Rb1,Rb2},
               {gen_module_tree,Ru1,Ru2}|TBt],
    MD1 = {Name,Type,Neighbour,Props,Others1,TBt,ToHigher,ToNeighbours,Work},
    module(FBrt,FmBelow,FmNeighbour,MD1).
```

分画ブロック内の素子名収集

分画ブロックの回路モジュール配下素子名をワーク部に収集し、その数によりブロック階層か末端かを調べる準備を行うもので、配下回路ネットモジュールに素子名返送を要求する放送を送り、自モジュールのワーク部に素子名を収集する。

```
module([{set_modulality,Name}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work} ;
    ToBroad = [{report_elem_name}|TBt],
    MD1 = {Name,Type,Neighbour,Props,Others,TBt,ToHigher,ToNeighbours,[]},
    module(FBrt,FmBelow,FmNeighbour,MD1).
```

分画ブロックの回路モジュール配下の素子名収集

素子モジュールであれば、素子名を上位モジュールに返送する。

```
module([{report_elem_name}|FBrt],FmBelow,FmNeighbour,MD):-
    MD = {Name,Type,Neighbour,Props,Others,ToBroad,ToHigher,ToNeighbours,Work},
    not( or( Type = {outlet,_} , Type = net ) ) ;
    ToHigher = [{report_name,Name}|TTt],
    MD1 = {Name,Type,Neighbour,Props,Others,ToBroad,TTt,ToNeighbours,Work},
```

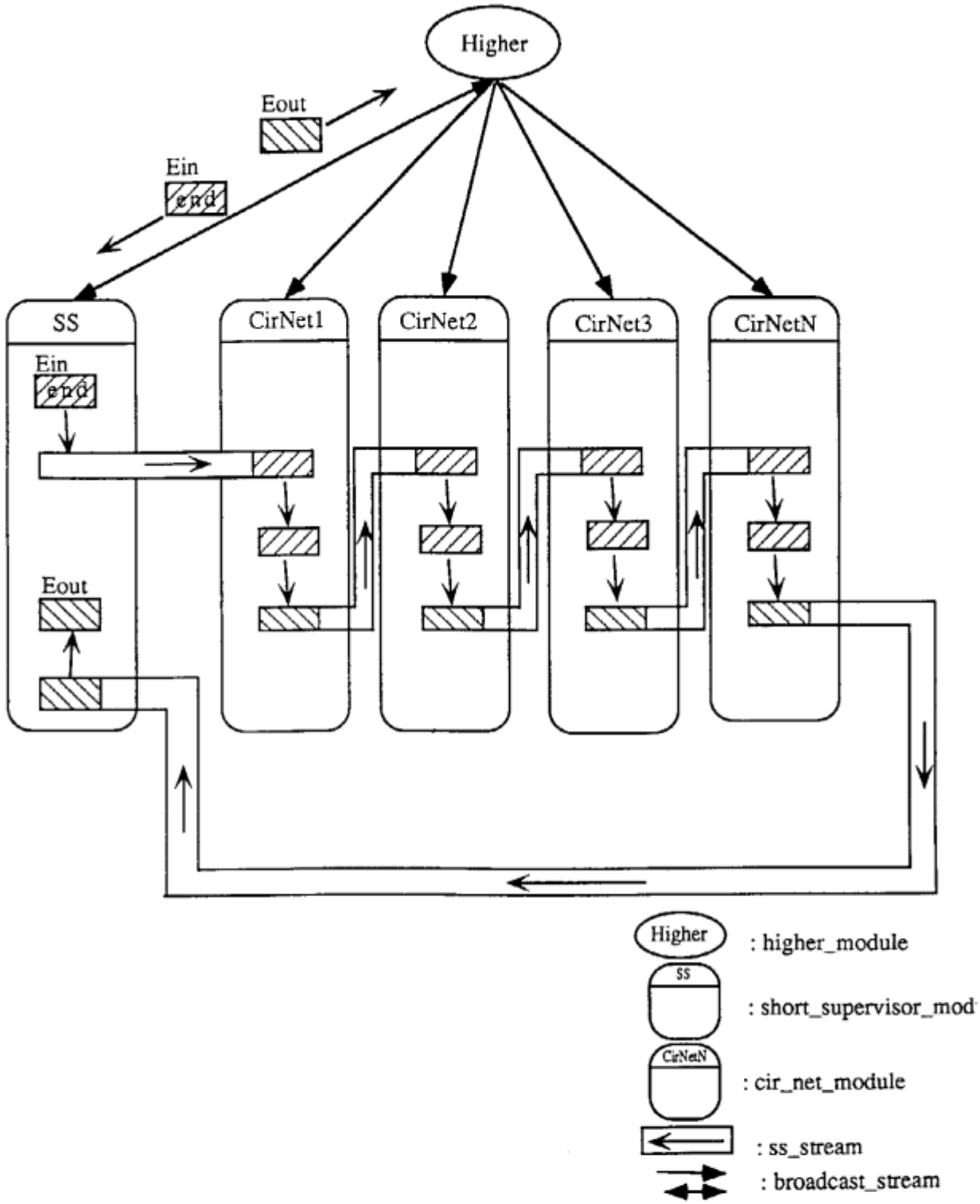


図 4-2 終了監視ストリーム

```
top_module(TMess,OutStr,RMS,FmModules,TM).
```

4．6．1　配置処理制御

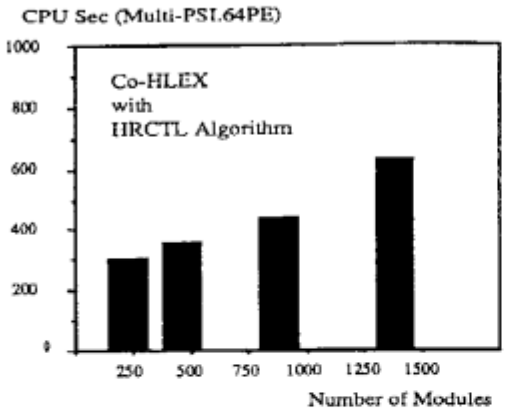
module/2の第1節は、素子モジュールをレイアウトする。choose_a_layoutframeメッセージは、指定された素子に適用可能な標準セルのレイアウトフレームをテンプレートライブラリより選択する。narrow_leaf_connsメッセージは回路の外郭上コネクタの持つrange-flag（コネクタ引き込みアクションのログ）にfinishedというマークを付ける事によって、周囲のモジュールの当該コネクタの引き込みアクションを許す。

第2節は、分割を行うブロック階層の回路モジュールを処理する。choose_a_layoutframeメッセージは回路モジュールの分割に使用するレイアウトフレームをテンプレートライブラリより選択する。generate_subproblemsメッセージによって、回路モジュールは通常4個の部分問題に分割される。3、2個に縮退した回路木（子供にvoidが混じっている）に対してもテンプレートは用意されている。place_all_sonsメッセージは、全ての子回路モジュールを再帰的に処理する。aggregate_subproblemsメッセージは、配置済みとなった子回路モジュールを分割時に使用したレイアウトフレームを用いて統合し、回路モジュールの配置を完成させる。ショートサーキット法によって並列走行する子回路モジュール群の処理完了を検出している。

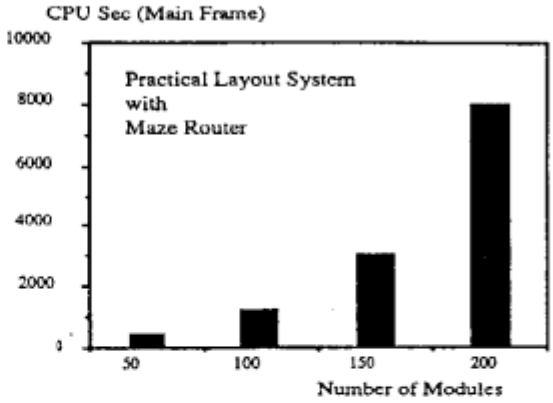
```
module([{place_by_qtree,Proc,end-Eo}|TMess],LM):- cell(LM) |
    send_message(TMess,[{choose_a_layoutframe,Proc,end-E1},
                        {narrow_leaf_conns,Proc,E1-Eo}]),
    module(TMess,LM).
module([{place_by_qtree,Proc,end-Eo}|TMess],LM):- not(cell(LM)) |
    send_message(TMess,[{choose_a_layoutframe,Proc,end-E1},
                        {generate_subproblems,Proc,E1-E2},
                        {place_all_sons,Proc,E2-E3},
                        {aggregate_subprblems,Proc,E3-Eo}]),
    module(TMess,LM).

module([{place_all_sons,Proc,end-Eo}|TMess],LM):-
    get(sons_stream,LM,[Lu,Lb,Rb,Ru],NLM),
    Lu = [{place_by_qtree,Proc,end-E1}],
    Lb = [{place_by_qtree,Proc,end-E2}],
    Rb = [{place_by_qtree,Proc,end-E3}],
    Ru = [{place_by_qtree,Proc,end-E4}],
    judge_end([E1,E2,E3,E4],Eo),
    module(TMess,NLM).
```

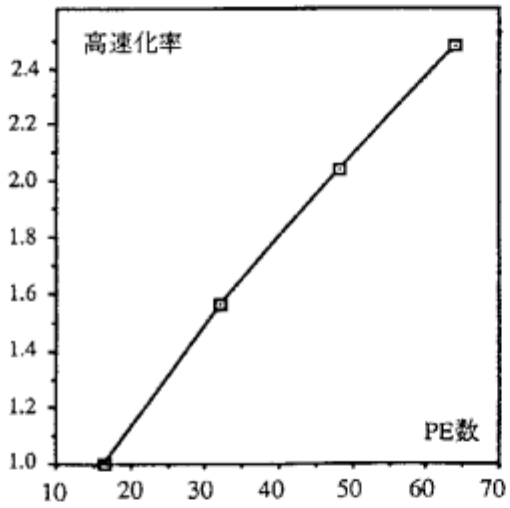
(1) Co-HLEXの計算速度



(2) 従来例(迷路法主体)の速度



(3) Co-HLEXの並列化効果



(4) Co-HLEXの規模（配置配線部）

Subsystems	KL1.Lines
Kernel	620
Planframes	2648
Layoutframes	1180
Layoutrules	684
Utilities	865
Total	5997

図 6-2 Co-HLEXの性能測定

回路モジュールを区画分割するスライスの各境界辺の配線容量（通過可能本数）と区画へのビア（異配線層間の接続コネクタ）敷設可能数とを推定した後、それらをベクトル形式にしてプロセス化する。このプロセスは、VecStrという通信用ストリームを保有している。

```
plan_wirings(LM,NetsList,NLM,AnsNetsList,Proc,end-Eo):-
    get(patent_shape,LM,Shape,LM1),
    get(layoutframe,LM1,LFName,LM2),
    get(circuit_name,LM2,CName,LM3),
    generate_goal_vector(LFName,LM3,Shape,Proc,VecStr,LM4),
    generate_wiring_manager(NetsStr),
    wire_all_nets(CName,NetsList,NetsStr,Shape,LFName,VecStr,AnsNetsList,end-Eo).

gererate_goal_vector(LFName,LM,Shape,Proc,VecStr,NLM):-
    planframe(LFName,gererate_goal_vector,GoalVec,InitVec,InitCost,Shape,Proc),
    create_process(GoalVec,InitVec,InitCost,VecStr0),
    merge([VecStr,VecStr1],VecStr0),
    put(vector,LM,VecStr1,NLM).
```

配線すべきネットの一つを取り挙げ、このネットについてget_inner_net_profile/2を行い各区画での内点コネクタの存在有無をプロファイルInNetProfileから調べる。次に外郭上コネクタへの通信ストリーム束NWSEContsStrをget_pericont_stream/2によって集め、wire_a_net/10によりネットの配線を行う。残りのネットに対しても同様に再帰実行する。

```
wire_all_nets(_,[],NetsStr,_,_,VecStr,AnsNetsList,Ei-Eo):-
    close([NetsStr,VecStr,AnsNetsList],Ei-Eo).
wire_all_nets(CName,[Net|Rest],NetsStr,Shape,LFName,VecStr,AnsNetsList,Ei-Eo):-
    NetsStr = [NetStr|Tail1],
    AnsNetsList = [AnsNet|Tail2],
    get_inner_net_profile(Net,InNetProfile),
    get_pericont_stream(Net,NWSEContsStr),
    wire_a_net(CName,NetStr,Net,InNetProfile,NWSEContsStr,Shape,LFName,VecStr,
    AnsNet,Ei-E1),
    wire_all_nets(CName,Rest,Tail1,Shape,LFName,VecStr,Tail2,E1-Eo).
```

配線中のネットの内点コネクタ間を接続する配線パターンを決定する。

第1節はwire_manager/1（配線マネージャ）からNetStrストリームを通じてfinishedメッセージ

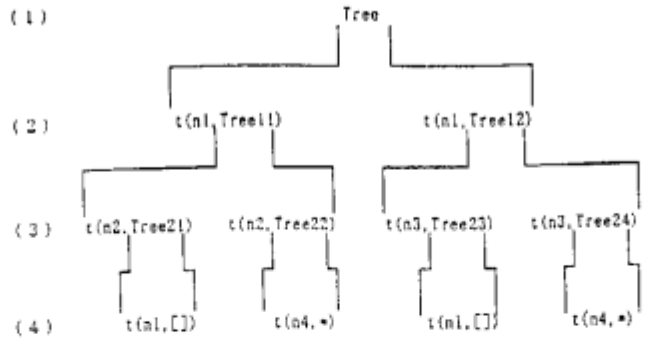


図７－３　解の表現方式

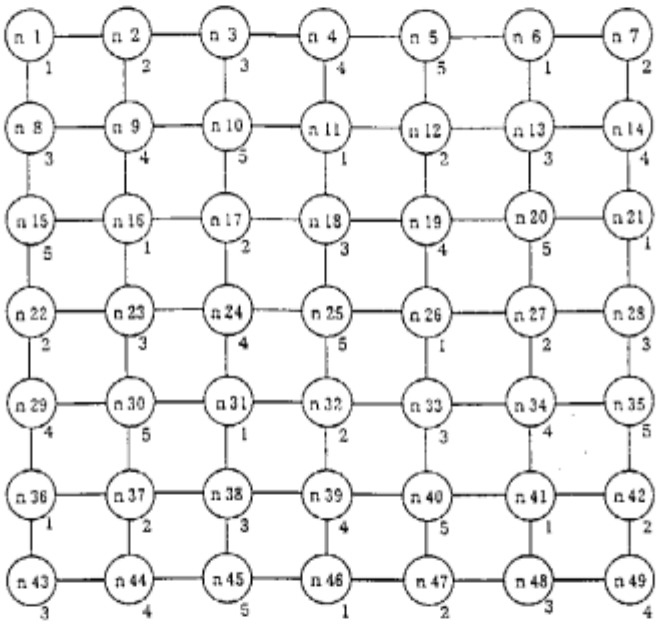


図７－４　７×７ネットワークの例

```
Shape,LFName,VecStr,AnsNet,E1-Eo).
```

wire_a_net/10の第2節をガードするwirable_feed-through/1は、i_can_pull_in/1かi_should_pull_in/1が成功する時に成功する。前者は全ての周辺モジュールが自モジュールで配線中のネットの全外郭上コネクタに対してpull-inアクションを実行した時に成功する。言い換えると、周囲4方向の近隣モジュールの立てたフラグNyf,Wyf,Syf,andEyfが、下記のプログラム内に定義したall_are_members/2条件を満たすとき成功する。
後者は、自分がかつて引き込みを断念した全外辺コネクタが近隣モジュールによって今回も引き込み断念された時成立する。この状況の判定は、外郭上コネクタの持つ近隣モジュールが与えたログがinspectedと設定されていることを見て判断する。

```
wirable_feed-through(NWSEConts):-
    get_pullin_flags(NWSEConts,NWSEFlags),
    or(i_can_pull_in(NWSEFlag),i_should_pull_in(NWSEFlags)) ! true.

i_can_pull_in([{Nmf,Nyf},{Wmf,Wyf},{Smf,Syf},{Emf,Eyf}]):-
    all_are_members([Nyf,Wyf,Syf,Eyf],[void,finished,narrowed]) ! true.

i_should_pull_in(NWSEFlags):-
    flags_i_have_delayed_before(NWSEFlags,DelayedFlags),
    you_have_tried_all(DelayedFlags) ! true.

you_have_tried_all(DelayedFlags):-
    all_are_unifirable([_,inspected],DelayedFlags) ! true.
```

外郭上コネクタをいずれかの内部区画に引き込み、仮内部コネクタとして既に存在する内部コネクタリストに加える。第1節は加えた結果をAnsInNetProfileに返す。第2節は、存在範囲ORangeがいずれかの内部区画境界線に含まれているため、すぐに引き込み可能なコネクタを処理する。引き込まれたコネクタはその時点での内部コネクタリストInNetProfileに追加されNewInNetProfileが新たに定義される。第3節はORangeが複数個の区画の境界線とオーバーラップするため、ただちに引き込み先を決められない場合に対応する。possible_pull_in_slices/3は、引き込み可能な区画PSlicesを求める。次いで各候補についての内点コネクタの分布状況PossibleInNetProfileをpossible_innet_profiles4によって求める。

```
pull_in_connectors([],_,InNetProfile,AnsInNetProfile):-
    AnsInNetProfile = InNetProfile.
pull_in_connectors([{{ORange,NRange},RFlag,Layer}|Rest],Shape,
```

```
[12] cs_circ:s([tr,res],Ans).
 Ans = [[],[],[],[],[],[],[],[t(q1,[]),t(q1,[]),t(q1,[]),t(q1,[])],[t(q2,[]),t(
(q2,[]),t(q2,[]),t(q2,[])],[t(q3,[]),t(q3,[]),t(q3,[]),t(q3,[])],[t(q4,[]),t(q4,[
]),t(q4,[]),t(q4,[])],[t(q5,[]),t(q5,[]),t(q5,[]),t(q5,[t(r5,*)])],[t(q6,[]),t(q6
,[]),t(q6,[]),t(q6,[t(r6,*)]),t(q6,[])],[t(q7,[]),t(q7,[]),t(q7,[]),t(q7,[t(r7,*)
]),t(q7,[])],[t(q8,[]),t(q8,[]),t(q8,[t(r8,*)]),t(q8,[])]]
```

```
[13] cs_circ:s([tr,dum,tr],Ans).
 Ans = [[],[],[],[],[],[],[],[t(q1,[]),t(q1,[t(dum5,[]),t(dum5,[t(q3,*)]),t(du
m5,[t(q1,*)]),t(dum5,[])]),t(q1,[t(dum8,[]),t(dum8,[t(q2,*)]),t(dum8,[t(q6,*)]),t
(dum8,[t(q1,*)])]),t(q1,[])],[t(q2,[]),t(q2,[t(dum6,[]),t(dum6,[t(q4,*)]),t(dum6,
[t(q2,*)]),t(dum6,[])]),t(q2,[]),t(q2,[t(dum8,[]),t(dum8,[t(q2,*)]),t(dum8,[t(q6,
*)]),t(dum8,[t(q1,*)])])],[t(q3,[]),t(q3,[t(dum2,[]),t(dum2,[]),t(dum2,[t(q3,*)])
,t(dum2,[])]),t(q3,[t(dum9,[]),t(dum9,[]),t(dum9,[t(q7,*)]),t(dum9,[t(q3,*)])]),t
(q3,[t(dum5,[]),t(dum5,[t(q3,*)]),t(dum5,[t(q1,*)]),t(dum5,[])])],[t(q4,[]),t(q4,
[t(dum3,[]),t(dum3,[]),t(dum3,[t(q4,*)]),t(dum3,[])]),t(q4,[t(dum11,[]),t(dum11,[
t(q4,*)]),t(dum11,[])]),t(dum11,[t(q8,*)])]),t(q4,[t(dum6,[]),t(dum6,[t(q4,*)]),t(d
um6,[t(q2,*)]),t(dum6,[])])],[t(q5,[]),t(q5,[t(dum7,[]),t(dum7,[]),t(dum7,[t(q5,*
)])]),t(dum7,[])]),t(q5,[t(dum10,[]),t(dum10,[t(q6,*)]),t(dum10,[t(q5,*)]),t(dum10,
[])]),t(q5,[])],[t(q6,[]),t(q6,[t(dum8,[]),t(dum8,[t(q2,*)]),t(dum8,[t(q6,*)]),t(
dum8,[t(q1,*)])]),t(q6,[]),t(q6,[]),t(q6,[t(dum10,[]),t(dum10,[t(q6,*)]),t(dum10,
[t(q5,*)]),t(dum10,[])])],[t(q7,[]),t(q7,[t(dum9,[]),t(dum9,[]),t(dum9,[t(q7,*)])
,t(dum9,[t(q3,*)])]),t(q7,[]),t(q7,[]),t(q7,[])],[t(q8,[]),t(q8,[t(dum11,[]),t(du
m11,[t(q4,*)]),t(dum11,[]),t(dum11,[t(q8,*)])]),t(q8,[]),t(q8,[])]]
```

```
[8] cs_circ:s([dum,tr_col,tr_emi,dum,tr_emi,tr_col],Ans).
 Ans = [[],[],[],[],[],[],[],[t(dum1,[]),t(dum1,[]),t(dum1,[])],[t(
dum2,[]),t(dum2,[]),t(dum2,[t(q3_c,[]),t(q3_c,[]),t(q3_c,[t(q3_e,[]),t(q3_e,[]),t
(q3_e,[]),t(q3_e,[t(dum9,[]),t(dum9,[]),t(dum9,[]),t(dum9,[t(q3_e,[]),t(q3_e,[]),
t(q3_e,[t(q3_c,*)]),t(q3_e,[])])]),t(q3_c,[])]),t(dum2,[])],[t(dum3,[]),t(dum3,
[]),t(dum3,[t(q4_c,[]),t(q4_c,[]),t(q4_c,[t(q4_e,[]),t(q4_e,[]),t(q4_e,[]),t(q4_e
,[t(dum11,[]),t(dum11,[t(q4_e,[]),t(q4_e,[]),t(q4_e,[t(q4_c,*)]),t(q4_e,[])]),t(d
um11,[]),t(dum11,[])])]),t(q4_c,[])]),t(dum3,[])],[t(dum4,[]),t(dum4,[]),t(dum4,[
]),t(dum4,[])],[t(dum5,[]),t(dum5,[]),t(dum5,[t(q1_c,[]),t(q1_c,[]),t(q1_c,[t(q1_
e,[]),t(q1_e,[]),t(q1_e,[]),t(q1_e,[t(dum8,[]),t(dum8,[t(q2_e,[]),t(q2_e,[]),t(q2
_e,[t(q2_c,*)]),t(q2_e,[])]),t(dum8,[]),t(dum8,[t(q1_e,[]),t(q1_e,[]),t(q1_e,[t(q
1_c,*)]),t(q1_e,[])])]),t(q1_c,[])]),t(dum5,[])],[t(dum6,[]),t(dum6,[]),t(dum6,
[t(q2_c,[]),t(q2_c,[]),t(q2_c,[t(q2_e,[]),t(q2_e,[]),t(q2_e,[]),t(q2_e,[t(dum8,[]
),t(dum8,[t(q2_e,[]),t(q2_e,[]),t(q2_e,[t(q2_c,*)]),t(q2_e,[])]),t(dum8,[]),t(dum
8,[t(q1_e,[]),t(q1_e,[]),t(q1_e,[t(q1_c,*)]),t(q1_e,[])])])]),t(q2_c,[])]),t(dum6
,[])],[t(dum7,[]),t(dum7,[]),t(dum7,[t(q5_c,[]),t(q5_c,[]),t(q5_c,[t(q5_e,[]),t(q
5_e,[]),t(q5_e,[]),t(q5_e,[])]),t(q5_c,[])]),t(dum7,[])],[t(dum8,[]),t(dum8,[]),t
(dum8,[t(q6_c,[]),t(q6_c,[]),t(q6_c,[t(q6_e,[]),t(q6_e,[]),t(q6_e,[]),t(q6_e,[])]
),t(q6_c,[])]),t(dum8,[])],[t(dum9,[]),t(dum9,[]),t(dum9,[t(q7_c,[]),t(q7_c,[]),t
(q7_c,[t(q7_e,[]),t(q7_e,[]),t(q7_e,[]),t(q7_e,[])]),t(q7_c,[])]),t(dum9,[])],[t(
dum10,[]),t(dum10,[]),t(dum10,[]),t(dum10,[])],[t(dum11,[]),t(dum11,[]),t(dum11,[
]),t(dum11,[t(q8_c,[]),t(q8_c,[]),t(q8_c,[t(q8_e,[]),t(q8_e,[]),t(q8_e,[]),t(q8_e
,[])]),t(q8_c,[])])],[t(dum12,[]),t(dum12,[]),t(dum12,[]),t(dum12,[])],[t(dum13,[
]),t(dum13,[]),t(dum13,[]),t(dum13,[])],[t(dum14,[]),t(dum14,[]),t(dum14,[]),t(du
m14,[])]]
```

図７－６　回路図の認識例

ッドスペースを分割したものを各セルの回路モジュールに含ませこれを各セルの実形状とすることでデッドスペース領域も配線可能となる。generate_mazerouter_processメッセージは、トランジスタや抵抗などのセルモジュール毎に迷路法を用いるための格子状に連結されたプロセスを生成し、これらを制御する天元プロセスの通信用ストリームを各セルモジュールに設定する。又、各セル面の南と東の再外郭迷路法プロセスをスルーホール敷設禁止領域とすることでセル間でのスルーホール隣接チェックが不要となり、計算スピードの向上に役立つ。

clear_topmost_through-holeメッセージは、最上位回路の南と東に位置する各セルモジュールの迷路法プロセスにスルーホール禁止の解除を指示する。set_topmost_planメッセージは、最後に最上位回路モジュールの外郭上コネクタ配置位置をチップの実外周辺位置に合わせ登録する。

```
top_module([{prepare_routing,Proc,end-Eo}|TMess],OutStr,ToModules,FmModules,TM):-
    ToModules = [{get_topmost_plan,TopShape,ExternalConnectors,Proc,end-E1},
                 {generate_global_placement,TopShape,Proc,E1-E2},
                 {generate_mazerouter_process,Proc,E2-E3},
                 {clear_topmost_through-hole,Proc,E3-E4},
                 {set_topmost_plan,ExternalConnectors,E4-Eo}|RMS],
    top_module(TMess,OutStr,RMS,FmModules,TM).

module([{get_topmost_plan,TopShape,ExternalConnectors,Proc,end-Eo}|TMess],LM):-
    get_and_put(peri_connectors,LM,ExternalConnectors,[],LM1),
    get(patent_shape,LM1,Shape,NLM),
    set_topmost_isolation(Shape,Proc,TopShape,end-Eo),
    module(TMess,NLM).

module([{generate_global_placement,FatherShape,end-Eo}|TMess],LM):-
    get(layoutframe,LM,LFName,LM1),
    get_and_put(patent_shape,LM1,LocalShape,GlobalShape,LM2),
    get_and_put(peri_connectors,LM2,PConns1,PConns2,LM3),
    get_and_put(vector,LM3,VecStr1,VecStr2,LM4),
    get(sons_stream,LM4,[Lu,Lb,Rb,Ru],NLM),
    connectors_process_kill(PConns1,PConns2,end-E1),
    vector_process_kill(VecStr1,VecStr2,E1-E2),
    shift_and_fill(LFName,FatherShape,LocalShape,GlobalShape,[LuS,LbS,RbS,RuS]),
    Lu = [{generate_global_placement,LuS,E2-ELu}],
    Lb = [{generate_global_placement,LbS,E2-ELb}],
    Rb = [{generate_global_placement,RbS,E2-ERb}],
```

表1 計算機室レイアウトシステムの機能概要

入力	問題定義データ	部屋データ	部屋形状・固定物データ
		配置物データ	型紙名・型紙データ
		配線データ	配線系統データ
処理	並列問題解決推論	並列レイアウト問題解決	問題解決処理制御
		上位配置処理	グループごとにゾーン生成
		上位配線処理	配線系統グループ分け・仮想コネクタ作成・仮想コネクタ間配線
		並列配置プロセス制御	並列配置プロセスの制御
		並列配線プロセス制御	並列配線プロセスの制御
		配置位置決定処理	ゾーン内制約・ゾーン間制約満足位置への配置
		配線位置決定処理	機器・仮想コネクタ間配線
		配置配線状況記憶	部屋形状・固定物データ
			配置物・型紙データ
			配線データ
			配置物の配置位置・向き，配線の配線位置・形状，ゾーン位置・形状
	レイアウト知識	形状処理	複合形状物処理
		制約処理	ゾーン内制約満足位置生成処理
			ゾーン間制約満足位置生成処理
			ゾーン内・ゾーン間制約評価処理
		配置候補地生成処理	配置状況認識処理
			配置候補地生成処理
出力	レイアウト図	部屋	部屋及び固定物平面図
		配置物	部屋内での配置状況平面図
		配置処理チャート	機器プロセス動作チャート
		配線	ケーブル配線平面図

```
                {route_leafcells,left-below,Proc,E2-E3},
                {route_leafcells,right-up,Proc,E3-E4},
                {route_leafcells,feed-through,Proc,E4-Eo}|RMS],
    top_module(TMess,OutStr,RMS,FmModules,TM).
```

4．8．1　電源配線制御

電源線は多くの素子が必要とする電流を各々に供給するために、電圧降下や電流密度を考慮した配線形態とする。親回路の持つ電源配線は各子回路が必要とする電流値を供給できる配線幅を求めて生成するものである。

第1節は、電源供給末端であるセルの回路モジュールの配線を行う。prepare_power_nets/6は、セル内の実コネクタと親の回路モジュールの電源配線とを結ぶ仮想内点コネクタを生成し、これらコネクタの位置とコネクタ層を取得する。このコネクタ情報を迷路法の天元プロセスにwireメッセージとして送信し配線を行う。

第2節は、分割を行うブロック階層の回路モジュールの処理を行う。

generate_power_subproblemsメッセージは、子回路モジュールが配線した電源ネットの配線長と電位を取得するための準備を行う。

route_powerlines_all_sonsメッセージは、全ての子回路モジュールを再帰的に処理する。

aggregate_powerlinesメッセージは、各子回路モジュールをつなぐ電源配線パターンをレイアウトフレームから選び出し、取得した各子供の電源線の配線長と電位から電源配線幅を決定する。

```
module([{route_powerlines,Proc,end-Eo}|TMess],LM):- cell(LM) |
    get(patent_shape,LM,Shape,LM1),
    get_and_put(power_nets,LM1,PowerNets,NewPowerNets,LM2),
    prepare_power_nets(PowerNets,NewPowerNets,RoutablePowerNets,Shape,Proc,
    end-E1),
    get(mrpn_stream,LM1,MRPNMessage,NLM),
    MRPNMessage = [{wire,RoutablePowerNets,E1-Eo}],
    module(TMess,NLM).
module([{route_powerlines,Proc,end-Eo}|TMess],LM):- not(cell(LM)) |
    send_message(TMess,[{generate_power_subproblems,Proc,end-E1},
                        {route_powerlines_all_sons,Proc,E1-E2},
                        {aggregate_powerlines,Proc,E2-Eo}]),
    module(TMess,NLM).

module([{route_powerlines_all_sons,Proc,end-Eo}|TMess],LM):-
    get(sons_stream,LM,[Lu,Lb,Rb,Ru],NLM),
```

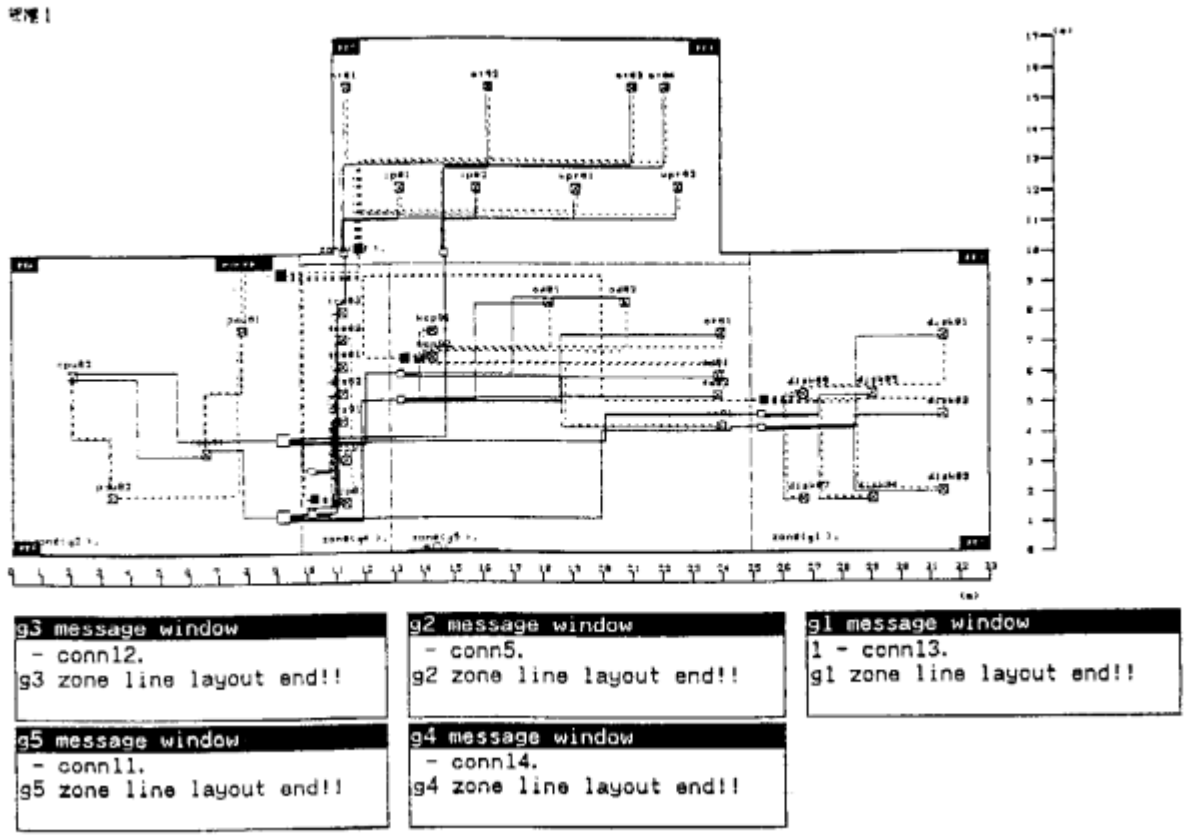


図 3-3 計算機室レイアウト実験画面（配線完了）

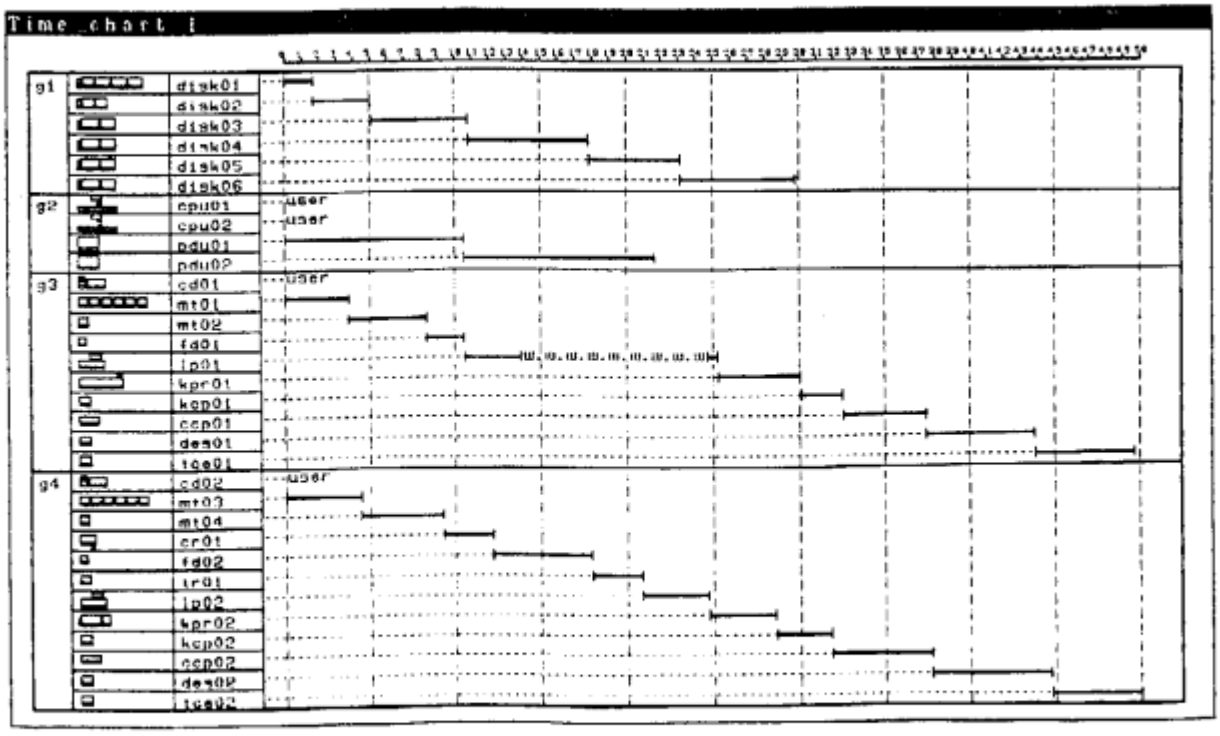


図 3-4 機器配置処理タイムチャート