多重チャネルリングバスに於ける ブロードキャスト伝送制御手順
——機能分散型計算機の内部接続方式——

喜連川 優、田中 英彦、元岡 達（東大 工学部）

1. はじめに

近年、データベースの普及に伴い、ソフトウェアサポートによるDBMSの性能の限界が次第に明らかになり、データベース管理の諸機能をハードウェアでサポートする事を試みたデータベースマシンの研究が着手されている。データベースに於ける検索処理の中には、潜在的に高度の並列性を見出す事が出来、多くのメモリ、プロセッサモジュールを互いに結合したアーキテクチャが採られる場合が多い。更に、検索処理の基本は内容による読み出し（Content Addressing）であるが、連想メモリを用いて大規模なデータベースを格納するにはコストの面から実際的でなく、従って二次記憶のフルスキャンによって擬似的に連想機能を実現する事がなされてきた。我々も同様の考えから、より発展させた形のマシン、“可変構造多重処理データベースマシン”[1,2]の設計、製作をすすめている。図1に示される本マシンはメモリモジュールから複数のプロセッシングモジュールに対してデータをブロードキャストし、並列に処理する事により高速化を狙っている。データのフルスキャンによる連想性は、アーキテクチャ的にはデータブロードキャストにより実現され、これはメモリコンフリクトのない環境を生み出している為、駆動プロセッサ台数に略比例した処理速度の向上が期待できる。しかしながら、多くのプロセッサに対しデータをブロードキャストする場合、一般に一台でもそのデータを読み誤ると当該オペレーションの正当性が保証されない場合がある。従って、この様なアーキテクチャを採用した場合、何らかの伝送制御手順によって、データの送受を正しく実現する事が不可欠となる。

ここではリングバス上でのブロードキャストを実現する際バス上のビットエラー及びバスインターフェイス部での同期障害に対して回復可能な伝送制御手順についての一手法（Broadcast Hand Shaking）及び逆に多くのモジュールから別の一つへ効率よくデータをあつめる為の手順（3 Phase Hand Shaking）について報告する。

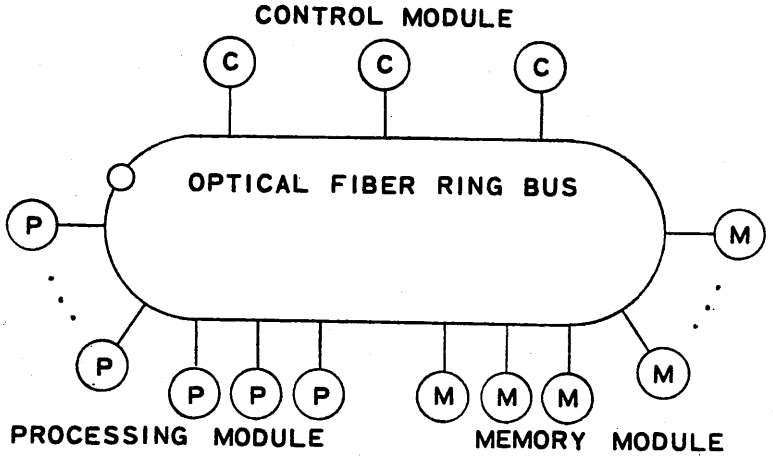


FIG.1 ARCHITECTURE OF DATA MANIPULATION UNIT

2. リングバスデータ形式

リングバスは図2に示される如く固定長チャネル時分割多重方式を採るものとする。ここでプロセッサはメモリから送出されたデータに対し処理を施すが、これはバスの転送レートではなくデータの出力レートによって定まる為、プロセッサはその処理結果を当該データチャネルよりもいくつか後のチャネルに出力する。

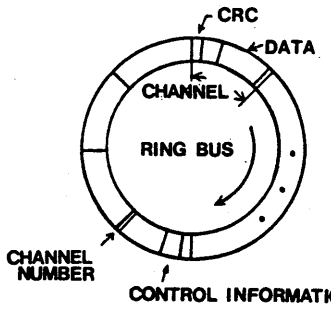


FIG.2 RING BUS DATA FORMAT

3. Broadcast Hand Shaking

この伝送手順を図3に示す。一台のメモリから多数のプロセッサに対してデータをブロードキャストする際の伝送手順であるということでこの様に呼んだ。

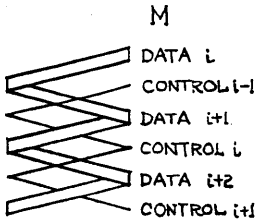


Fig.3. Broadcast Hand Shaking

3－1 設計方針

1) 伝送制御手順の為の制御情報は全てコントロール部に入れ伝送制御の簡素化を計る。

チャネルはデータ、コントロール両パートに分かれプロセッサは前者に対して処理を施し、後者に処理結果を書き込むが、ここに伝送制御用の情報を含め、データパートには一切含めない。両者をまとめ、(データ、コントロール)を一つのユニットとして回復処理を行なう事によって、伝送手続の簡素化を計る事が出来る。

2) プロセッサを群として扱う。

個々のプロセッサを意識してデータを送出するのでは制御できる台数には限度がある。従って任意台数のプロセッサに対してデータ伝送出来る為には、対象を群としてとらえる必要があり、ここでは結合プロセッサのＩＤではなく総数に着目した。

3) 一台のプロセッサの脱落も許さない。

ブロードキャストは常に全プロセッサが正しいデータを受信した事を確認しつつ行なう。従って一台でも読み損じた場合には全プロセッサに対して同一データを再送し、当該プロセッサが回復するまで他を待たせる。

3－2 動作概説

▸ メモリの動作

1) データパートが来るとデータを出力しコントロールパートが来るのを待つ。
2) コントロールパートが来ると、1)で送ったデータを全てのプロセッサが正しく受信したかどうかチェックし、正常なら、ACKを送り、次のデータを用意し、1)へ。異常なら、NACKを送り、リカバリモードに入いり、3)へ。
3) NACKの場合、次のデータパートで前々回のデータを再送し、プロセッサの回復を待つ。
4) プロセッサが回復すると、1)へ。

▸ プロセッサの動作

1) データパートが来るとデータを受信し、正しく受信された場合は処理を施し、コントロールパートが来るのを待つ。
2) コントロールパートが来て、ACKの場合にはデータを受信した事を書き込むと共に処理結果を出力し、1)へ。
3) NACKの場合には、前回及び前々回のデータに対するオペレーションをバックアウトし、1)へ。

以上は正常なプロセッサ及びメモリの振舞であり、障害回復の仕方は省いている。

3－3 総数チェックによる障害検出

メモリは自分に結合している多くのプロセッサをその総数が等しいかどうかという事によって動作監視する。即ち、メモリはコントロールパートの一つのフィールドをカウンタとして用い、結合されたプロセッサの総数に等しい値を書き込んで送出する。プロセッサは1だけデクリメントして値を返す事にすると、カウンタはプロセッサ群を通り、再び戻って来た時には0になっている筈である。'0'によって、プロセッサ群正常とみなす事にする。(総数チェック) このカウンタ長は $\log_2 N$

(N: 結合プロセッサ総数)

3－4　障害回復

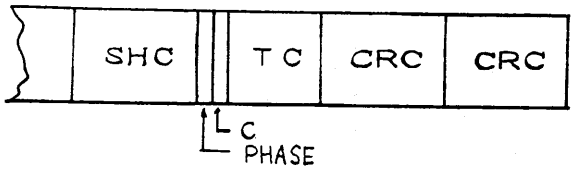
総数チェックにより障害が検出された時の回復手順について考える。プロセッサは当該データに対する処理に一定の時間が必要な為、その処理結果をデータの入っているチャネルよりも幾つか後のチャネルに書込む。その為、メモリにとってはデータパートが戻ってきた時には、次のデータを送るべきか、或は再送すべきかはまだ不明であり、プロセッサ群の動作が正常である事を期待して次のデータを出力する。この後コントロールパートが来、制御情報が得られ、その段階で始めて前々回に送出したデータが正常に受信されたかどうかが判明する。この遅延が制御を複雑にしている。勿論、一つのデータ毎に確認すれば簡単な制御ですむが、データ伝送率は半分になってしまう。障害を検出するとメモリはNACKを送り再送モードである事を示し、次のデータパートで前々回のデータを送出する。この再送データに対する結果も同様に遅れて到着するので、メモリは回復対象データとそれに続くデータの二つを組にして回復するまで送出を繰返す。この際、再送データの前はNACKを次はACKを交互に送る。障害プロセッサ自身の回復動作については、障害を次の二つに分けて考える事にする。

a.　一台のプロセッサが障害を起こした場合
b.　二台以上のプロセッサが引き続いて障害を起こした場合

aの場合、障害を起こしたプロセッサは回復した時点で、メモリ及び他のプロセッサ群が自分の回復を待っている筈であり、NACKの次のデータへリンクすればよい。aでは回復対象が一つに確定している為、制御は容易であるが、実際にはbの場合がある。複数台のプロセッサが全く同時に障害を起こした場合はaに帰着されるが、連続してプロセッサが障害を起こした場合が問題となる。プロセッサP_0が障害を起こし当該チャネルを読みとばし、プロセッサP_1はそれは読んだが次を読みとばしたとする。その時、全システムの回復対象はP_0の読みとばしたデータであり、P_1のそれではない。P_1はシステムが自分の回復を待っていると考えてはならない。即ち障害を起こしたプロセッサにとって自分より一つ前に他のプロセッサが障害を起こしている可能性がある為、回復対象がいずれであるか判定出来ねばならない。そこで、ここではPhase Bit (1bit)を導入し、データに対し0,1を交互に付ける事とした。この1ビットにより、回復対象を決定出来る。例えば$\{(1\ \mathrm{NACK}), D_0, (0\ \mathrm{ACK}), D_1\}$なる系列が送られてくると、プロセッサの最後に受信したフェイズが1の場合には、フェイズ1のNACKが来る事により自分が回復対象である事がわかり、その次のフェイズ0のデータより処理を再開する。0の場合には自分より1つ前に障害を起こしたプロセッサがあり、それに対するNACKを読みとばした事になる。従ってフェイズ0の処理をバックアウトし、処理を再開する。他のプロセッサの障害回復中に障害を起こしたプロセッサの回復も同様にして行なえる。回復対象に対する制御情報を読みとばすと、回復後もやはりこのデータの回復を行なっている筈であり、実質上回復対象の決定は不要となる。

以上、障害を起こしたプロセッサが回復出来る為にはそれだけの情報を与える必要があり、リングバスに於けるブロードキャストではカウンターを用いACK/NACKの他にもう1 bit Phase Bitを付加する事でこれが可能である事が示された。

3－5　コントロールパートフォーマット



SHC: Shake Hand Counter (8 bit)
PHASE: Phase. No. (1 bit)
TC: Tuple Control
C: ACK/NACK (1 bit)

3－6　状態遷移図

説明は略するが、上述の伝送手順を状態遷移図に表わすと、メモリは図6、プロセッサは図7. 図8の如くなる。状態、変数の記述は表1参照。(Phase 0の2)

4.　3-Phase Hand Shaking

多くのプロセッサ群からメモリに対して効率よくデータを出力する為の回復可能な手続として、以下の方式を採用した。図4に示される如く3つのフェイズを互いに重合わせた形になっている為、3 Phase Hand Shakingと呼んだ。

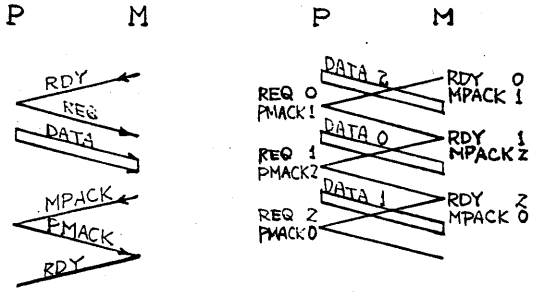


Fig. 4.　3-Phase Hand Shaking

4－1　設計方針

この伝送制御手順を設計する上で、次の点を考慮した。

1) データパートとコントロールパートの分離
Broadcast Hand Shakingとの一貫性を保つため

2) データ伝送効率の向上
データ伝送は3つのPhaseがインタリーブした形になっており、無駄はない。更に一つのPhaseがプロセッサリカバリに使われていても他の二つのPhaseは有効である様、効率向上を計っている。

4－2　動作概説

▸ プロセッサの動作

1) コントロールパートでRDYになっているものを探す.
2) REQを出力するとともに、自分のIDを書き込む。
3) そのコントロールパートに対するデータパートが来ると、データを出力する。
4) メモリがデータを正しく受けとったかどうか、MPACK/MPNACKを待つ。
5) MPACKの場合はメモリから返事を受け取ったという事をメモリへPMACKとして送る
6) メモリが次の入力に移ったのを見て、1)へ戻り新たな出力に取りかかる。
5') MPNACKの場合は、1)へ戻り再出力する。

▸ メモリの動作

1) RDYを送る.
2) RDYがREQとして戻ってくると、データチャネルを読み、特定パタンを出力しておく.
3) データが正常に読み取れるとMPACKを、エラーが検出されるとMPNACKを、プロセッサへ送る.
4) ACKを送った場合、プロセッサからのACKを待ち、戻ってくると1)へ。NACKの場合は、そのまま1)へ。

以上はプロセッサ、メモリが障害を起こさず正常にデータをやりとりする場合である。

4－3　障害回復

ここでは、障害（同期ずれ）回復を考えに入れた場合、先の手順が最小限必要であることを示す。障害の発生する時点は種々考えられるが、論理的に図5の如く、3つ（T_0, T_1, T_2）に分ける事が出来る。プロセッサの動作に着目して考えると、

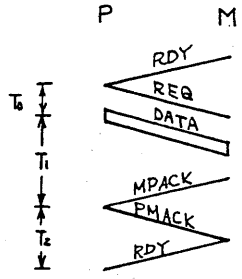


Fig. 5. 3-Phase Hand Shaking における障害分類

T_0：RDYをみつけREQを出したが、データを出す前に障害を起こした。

T_1：データは出力したが、メモリからのMPAckを受け取る前に障害を起こした。

T_2：メモリからのMPAckを受け取り、PMAckを送ったが、それがメモリに届いたかどうか確認する前に障害を起こした。

1) T_0の場合の回復

データパートには制御情報を入れない事にした為、プロセッサはまずコントロールパートでRDYを探し、対応するデータチャネルを確保する。プロセッサがRDY→REQとしたにも拘わらず、データを出力せずに障害を起こした時の検出は、メモリがデータパートに常時特定パタンを出力しておき、それ以外のデータであった時はじめて読みとる事によって実現出来る。メモリはNACKを送り、プロセッサは再出力すればよい。

2) T_1の場合の回復

プロセッサの動作の5)（PMAcK）は、一見冗長に思われるが T_1からの障害回復を行なおうとすると不可欠となる。例えば、PMACKを略し、MPAckをやり捨てにした場合を考えると、プロセッサがデータを送出した後に障害を起こした場合、その後回復した時には自分の送出したデータがメモリ側で正常に受け取られたかどうか知るすべがない。従って回復可能である為には、もしデータをメモリが正常に受信した場合、プロセッサがMPAckを受け取ったという証拠にメモリはPMAckを受けとられねばならないという事になる。従って、T_1でプロセッサが障害を起こし、メモリがMPAckを送ったにも拘わらずPMACKが戻ってこない場合、メモリはプロセッサが回復するのを待つ。この「待つ。」という動作によって不確定時間障害を起こすプロセッサのリカバリが可能になっている。MPNACKの場合も同様にプロセッサの回復を待つ事が出来るがここではNACKの時はやり捨にした。メモリはMPAckを出力したにも拘わらずプロセッサよりPMAckが戻ってこないとダミーサイクルを繰返し、当該Phaseに於て回復中のプロセッサからのPMAckを待つ。PMACKが戻ってくると、次データの入力に取りかかる。プロセッサは回復すると自分が前に出力したチャネルに対して自分の回復を待つてくれているかどうか調べる。もし待ってくれている場合にはデータが正しく転送されているのでPMACKを送り次の仕事に取りかかる。待ってくれていない場合には正しく転送されなかったことになりRDYを探して再出力する必要がある。

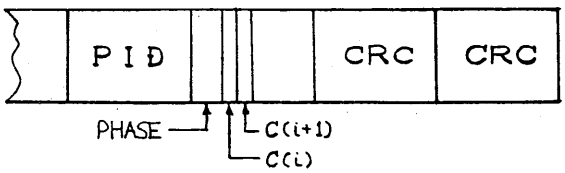
次にプロセッサの識別について考える。メモリはある一台のプロセッサとデータのやり取りを行ない、その回復を行なう必要がある。この為には個々のプロセッサを意識せざるを得ない。つまり引き続いて二台のプロセッサが同一Phaseにリンクし両方共障害を起こしたが、後者のみデータが正常に受け取られた場合、二つの

プロセッサにとってメモリが一体誰の回復を待っているのか識別出来る必要がある。データを正常に受信しなかった場合も回復を行なうならば一つのチャネルの一つのPhaseに結合し得るプロセッサは一台だけであるから、識別する必要はない。しかし、一方でデータの出力効率を上げる事が望まれ、NACKをやり捨てにする事はこれに寄与するが、逆に一つのPhaseに複数の回復中のプロセッサがリニアする可能性が生ずる為、それ等を区別する必要がある。ここではBroadcast Hand Shakingの際の総数チェック用のフィールドをプロセッサIDの為に使い、NACKをやり捨てにして、データを正常に受け取った時のみ回復の対象とする事にした。回復中のメモリはRDYの代わりに、始めからREQにしてコントロールパートを出力する。その際、本来ならばプロセッサ自身がPIDを書き込むのであるが、あらかじめ、回復対象であるプロセッサのIDを書いておく。その後のデータチャネルは読みとばし、MPACKを送り、PMACKが戻って来るのを待つ。プロセッサは回復後、当該Phaseにリンクし、自分のIDが書かれているかどうかを調べる。

3) T_2の場合の回復

プロセッサはメモリよりMPACKを受け取り、PMACKを返送したが、その後障害を起こした為、PMACKがメモリに正しく届いたかどうか不明になる。この場合のメモリの回復方法はT_1の場合に等しい。つまり、PMACKがエラーを起こすと、メモリにとってはプロセッサからデータは正常に受け取ったが、MPACKに対する応答であるPMACKが戻って来ないという状態に変わりはない。プロセッサの回復手法は、T_1のそれと異なる。即ち、プロセッサは既にメモリがデータを受け取った事実は知っており、PMACKがメモリに届いたかどうかという事のみが問題となる。T_1の場合にはデータ自体が届いたかどうかが問題であった。プロセッサはT_2と同様に当該Phaseにリンクし、そこに自分のIDが書かれてあると、PMACKが正しく届かなかった事を示し、それを再送する。書かれていないと、正しく届いた事を示し、次のデータ出力に取りかかる。

4－4　コントロールチャネルフォーマット



PID: Processor ID
PHASE: Phase.No. (0, 1, 2)
C(i): RDY(i) / REQ(i)
C(i+1): MPACK(i+1) / PMACK(i+1)

4－5　状態遷移図

説明は略するが、上述の伝送手順を状態遷移図に表わすと、メモリは図9、プロセッサは図10の如くなる。

但し・Phaseの0～2は各々対等であるから、ここではPhase 0の部分だけ表わすことにする。
・実線は、正常動作中での遷移を表わし、破線は障害が生じた場合の回復動作に伴う遷移を表わす。
・状態、変数の記述は表2参照。(Phase 0のみ)

5. おわりに

多数のプロセッサにデータをブロードキャストし並列処理を試みる場合、データの正しい送受を保証する為、何らかの伝送手続きが必要となる。ここでは、障害の検出だけではなく、回復可能な制御手順について検討した。これによって、障害時点から再び処理を開始することが可能となり、検出と再試行による手法と比べて性能の向上が期待できる。又制御用データが大きなオーバーヘッドとならない様、最小限の情報量によって実現することを念頭に置き、全体として僅かなビット数で押えることができた。

参考文献

1) 喜連川、赤松、田中、元岡 "可変構造多重処理データベースマシンの構成" S.55 情報処理学会全国大会 2F－5

2) 〃 〃 2F－6

3)「高速リングバスシステムハードウェア仕様書」パターン情報処理システム技術研究組合 S.52

Memory
- D^0 : Phase 0のデータ待ち （正常）
- C^0 : Phase 1のコントロール待ち （〃）
- D^1_E : 回復対象データ（Phase 1）の再送待ち
- C^1_E : 回復状態NACK確認

Processor
- D^0 : Phase 0のデータ待ち （正常）
- C^0 : Phase 0のコントロール待ち （〃）
- C^0_{L0} : データ入力異常終了後コントロール待ち
- D^1_{L0} : Phase 1のデータ待ち （回復中）
- C^1_{L0} : NACK待ち （〃）
- D^0_{L0} : Phase 0のデータ再送待ち （〃）
- C^0_{L1} : 再送データ正常入力後コントロール待ち （〃）
- C^0_{L2} : 再送データ異常入力後コントロール待ち （〃）
- R^0 : D^0, C^0, C^0_{L0} での障害回復後の状態
- R^0_L : その他での障害回復後の状態

Memory
- CO : 総数チェック正常
- PH : Phase 0
- ACK : ACK り帰着
- $\overline{CRC}$: データエラー
- D : データパート到着
- C : コントロールパート到着

Processor
- ACK : Memory よりACK
- DN : チャネルNo.一致
- PH : Phase 0
- CRC : データエラー
- D : データパート到着
- C : コントロールパート到着

表1 Broadcast Hand Shaking に於ける状態の記述

Memory
C_1：REQ 待ち
D_1：入力データ待ち
D_0：REQなし データ待ち
C_4：データ正常入力後MPACK送出待ち
C_3：データ異常入力後MPNACK送出待ち
C_6：MPACK後PMACK待ち
C_5：MPNACK確認待ち
C_2：PMNACK受信後コントロール待ち
D_2：ダミーデータ待ち
D_3：REQ異常入力後データ待ち
C_0：D_3での障害後の状態

Processor
C_0：RDY待ち
D_0：データ出力待ち
C_1：Phase 1のコントロール待ち
C_2：MPACK待ち
C_3：PMACK到着確認
R_1：C_1, C_2での障害後の状態
R_2：C_3での障害後の状態
I：アイドル状態

Memory
$\overline{RDY}$(=REQ)：Processorからの出力要求
PMACK：Processorより ACK
PH(0~2)：Memoryの出力したPhase No.
SP：データが特定パタン
WAIT：データ入力禁止
CRC, D, C, DN：表1と同じ

Processor
RDY：当該チャネルが空
MPACK：Memoryより ACK
SID：IDフィールドに自分のIDが書かれている。
DMD：データ出力要求
PH(0~2), CRC, D, C, DN：Memoryと同じ

表2　3-Phase Hand Shakingに於ける状態の記述

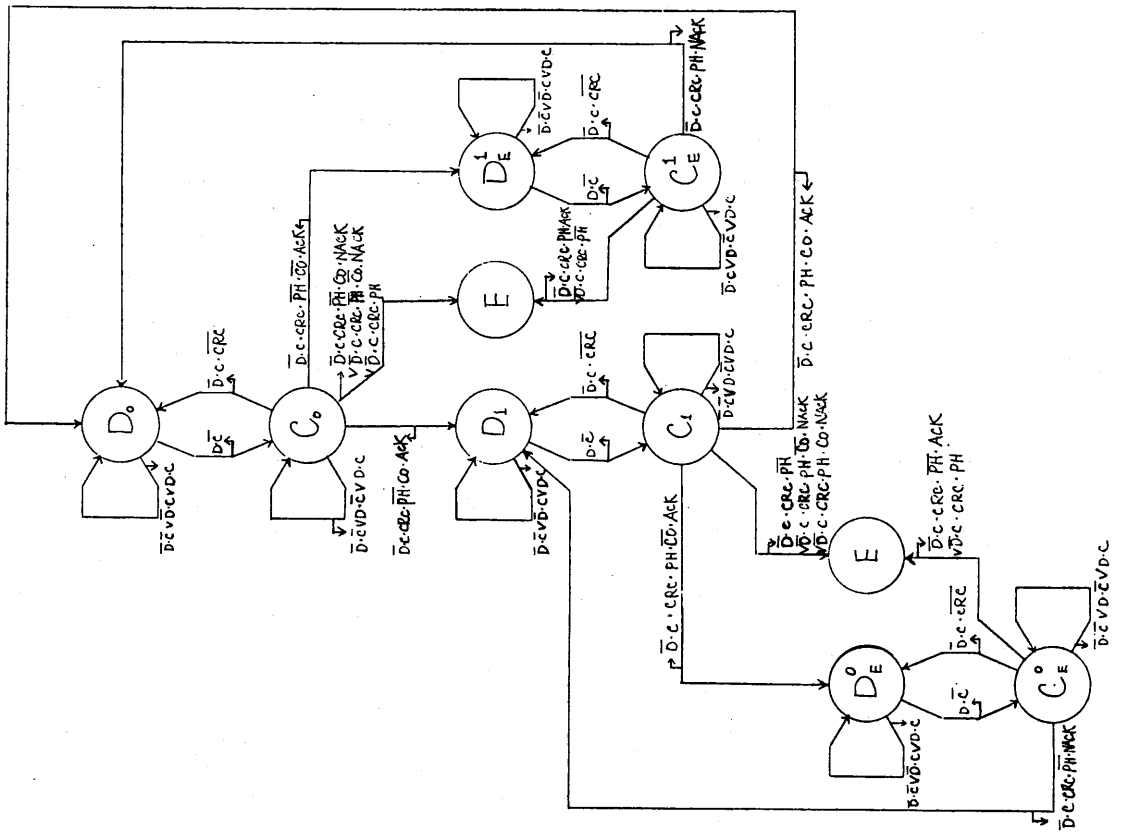


図6

Broadcast Hand Shaking
プロセッシングモジュール状態遷移図 II

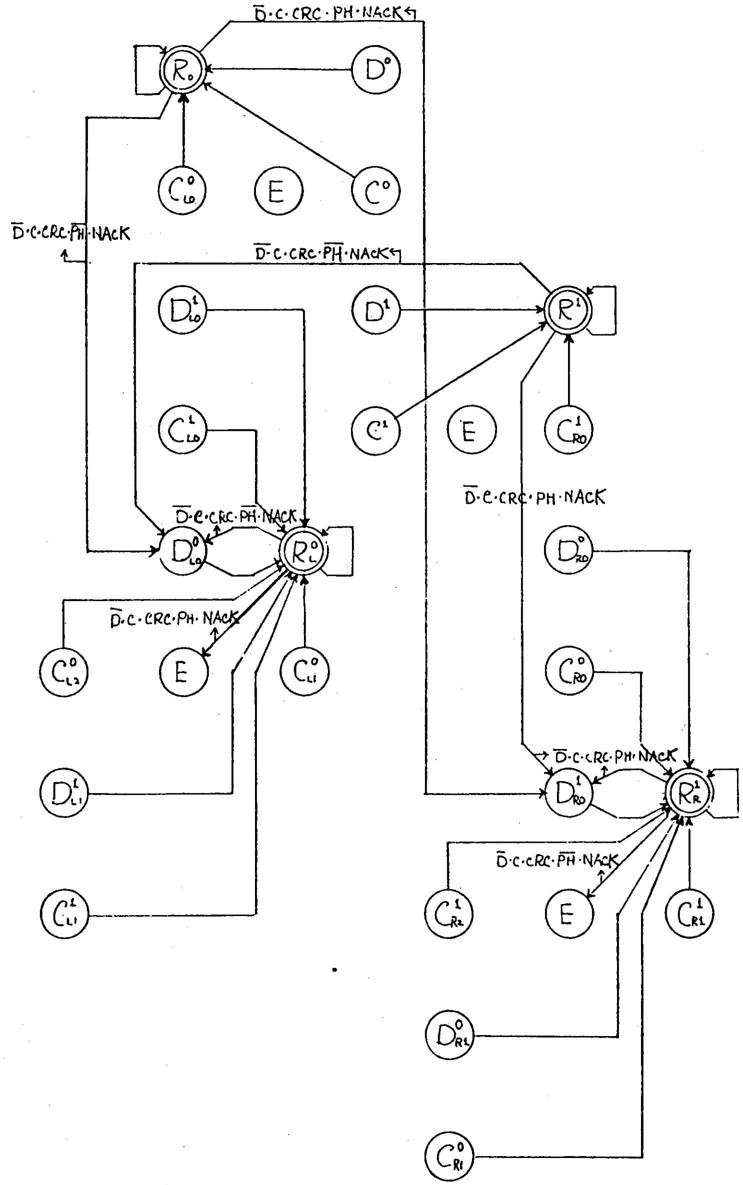


図 7　Broadcast Hand Shaking
プロセッシングモジュール状態遷移図 I

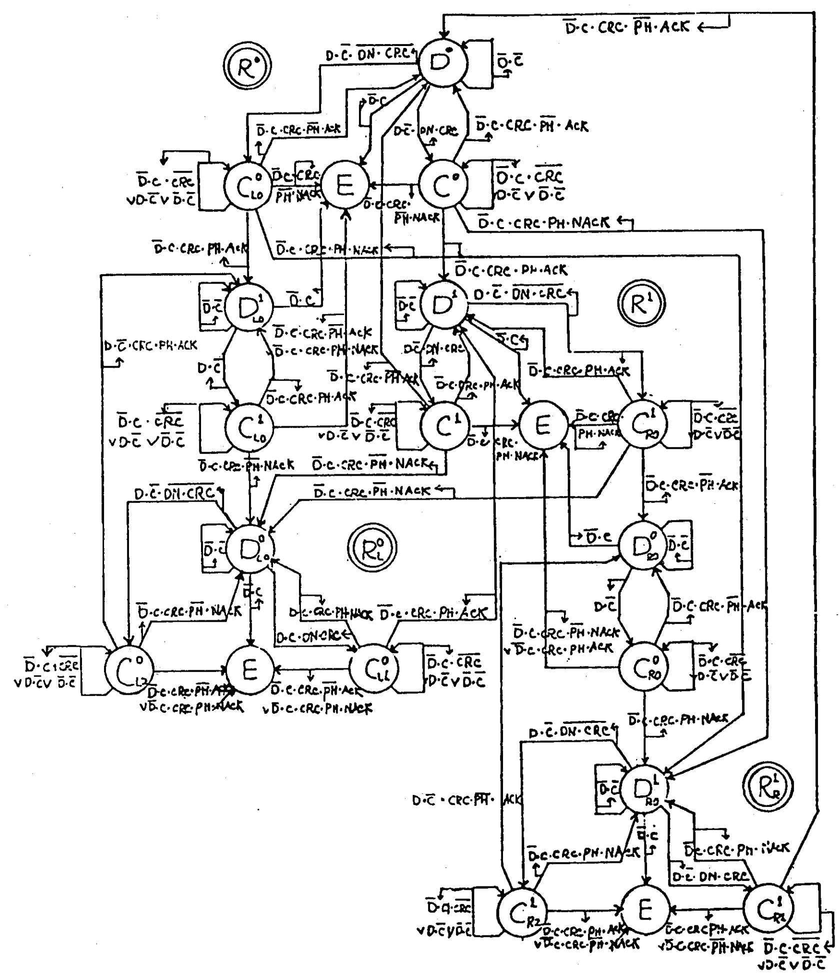


図 10　3-Phase Hand Shaking
プロセッシングモジュール状態遷移図

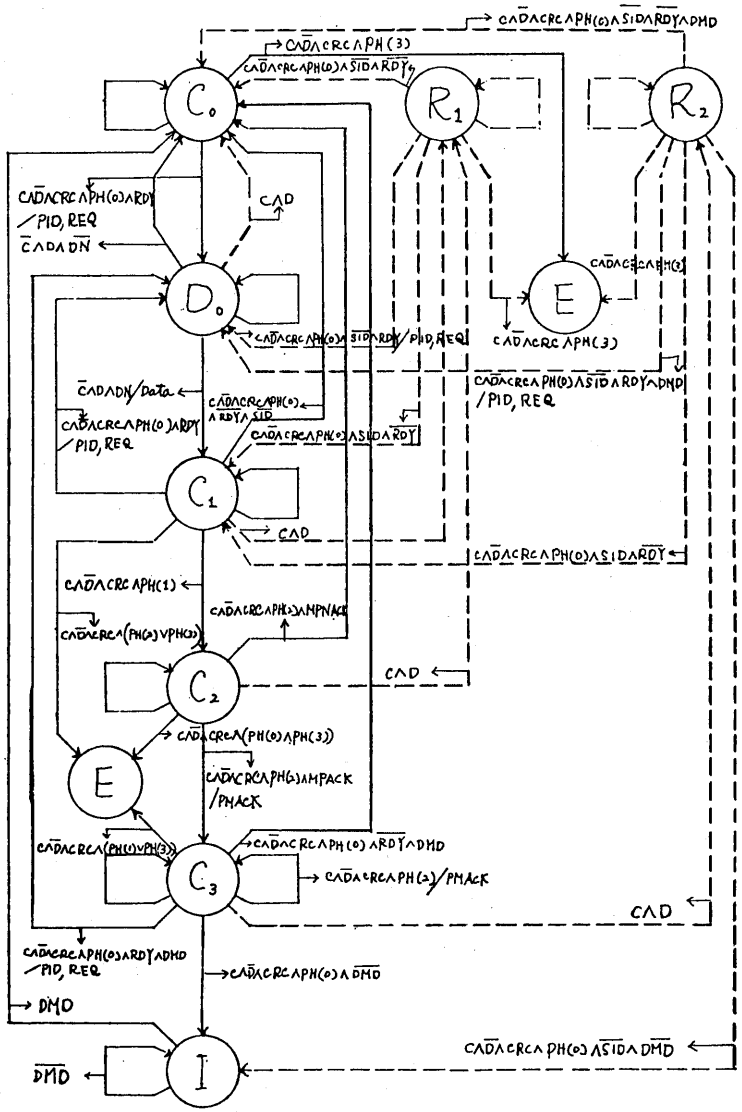


図 9　3-phase Hand Shaking
メモリモジュール状態遷移図