網向きプロセス間通信制御プロセッサの構成

和賀井フミ子　和田哲郎　小森　斉　田中英彦　元岡　達
（東京大学　工学部）

第1章　序

計算機網の構成法に対する研究は各所で行なわれているが、当研究室に於いても、以前から、プロセス間通信機構を主軸とする計算機網TECNETと、網向きのOS(NOS)を開発してきた。[1,2,3]

プロセス間のインタラクションは、メッセージを交換することにより行なわれる。その際、プロセスは相手プロセスの存在ホストを意識することなく通信できるという特徴を持つ。

現在は、網向きのOSの核機能をハードウェアでサポートするサブシステムについて考察し、実装を行なっているので、これについて報告する。

第2章　オペレーティングシステムのハードウェア化について

2－1　OSの機能分散と核の機能の分離

分散処理の一方式として、機能分散型計算機を考えることができる。これは、特定機能に専用化されたサブシステムを組合せて構成されるものである。この方法では、OSを機能別に専用マシンの上に分散する方式になっている。

例えば、当研究室においてはシステム全体を、機能サブシステム，システム管理サブシステム，データサブシステム，I/Oサブシステムの4つに分割した機能分散型計算機を構成し、種々の検討を行なっている。[4,5]

ここでは、この考えを更に進めて、OSの基本となる部分、つまり核の機能分散について考察する。

2－2　核の機能と分離

分散処理向きのOSの核の機能として必要と考えられるものは、一般に、マルチプログラミング，割込処理，仮想記憶，等の基本的な機能の他に、プロセスに対するサービス機能として、1) プロセス間通信（同一システム内及び他システム間）2) プロセス間の同期　3) プロセスの生成・消滅　4) プロセスに対する保護　5) プロセスの核利用インタフェスの提供，等がある。

プロセスには、核プロセス，システムプロセス，ユーザプロセスの3種類がある。

プロセス間のやりとりの道具としては、マクロが幾つか用意されている。

ここでは、上記のサービス機能の分離について考える。すなわち、プロセスにとっては重要だが、必ずしもホスト内で処理する必要のないこれらの機能を専用のハードウェアとともにホストから取り出し、ホスト外で処理するようにする。

このことにより、機能分散構造のサブシステム間インタフェスが明確になり、各サブシステムの制御プログラムが簡略化され、効率も上がる，と思われる。マ、このような特定機能を専用ハードウェアとして設けたことで、これらの間のインタフェスが明確になれば、異種計算機に於いても同一の機能がそのまま使える、という利点が生ずる。

そこで、機能分離を幾つか試み，処理のし易さ、インタフェス，オーバヘッド等の面から比較する。

2－2－1　分離に当たって考慮すべき事。

1) 分散処理向きのプロセス間通信は他システム内のプロセスにも適用する必要があるので、できるだけホストから出した方が良い。

2) プロセスの保護機能は、プロセスの用いるマクロの引数に関して、セキュリティのチェック（通信や同期の可否）や値のバインディング（ユーザ使用論理値からシステム処理値へ）を行なうので、通信や同期機能と同じかホスト寄りに置く。

3) プロセスの同期管理は、プロセス管理テーブルを用いて、プロセスの状態遷移の管理を行なう部分である。

wait状態やready状態の切換は一部は他プロセスによるが、プロセス間通信によって行なわれることも多いので、通信機構と同じ所に置くと良い。

run プロセスの切換は、ディスパッチャによって行なわれるが、この為の機構（レジスタやアドレス退避領域や割込）は、ホスト内に必要である。

同期管理をホストから出した場合、run プロセスの切換の必要がある時だけホスト外から、割込等によってディスパッチャに指示し、それ以外のプロセスの遷移は、ホスト外でプロセス管理テーブルのみを操作するだけでよいので、ホスト内でのオーバヘッドは無くなる。

4) プロセスの生成・消滅については同期管理と同様な事が言える。すなわち、プログラムのロード等の機構はホストに置く必要があるが、その指示やプロセス管理テーブルの操作機能は、必ずしもホストに置く必要はない。

2－2－2 機能の分離法

以上の事に基づいて、下表のように5つのタイプに分けた。

これらのうち、比較的良いのは、H1S4 とH4S1 である。

前者は、プロセスの諸サービス機能とプロセスが分離されている。後者はインタフェスやハードウェアの面からは作り易い。

2－3 機能の分離によって生ずる利点

1) ホストのOSの負担が軽減される。
2) 特定機能の専用ハードウェアによる効率向上
3) ホストから取り出したことにより異種計算機にも、諸特殊ハードウェアの接続が容易になった。
4) 機能の階層化が明白になる。

タイプ	ホスト内		外		特徴
H5S0	IS, CAP PSN, C&D IPC	PT CCB			すべての処理をホストで行なうのでオーバヘッドが大きい。プログラムモジュール間の境が不明確になりやすい。
H4S1	IS, CAP PSN C&D	PT	CCB	IPC	プロセス間通信処理を専用のマシンで行なうのでホストの負担は軽減される。
H3S2	IS CAP C&D		PT CCB	PSN IPC	プロセス間通信に関してはH4S1と同様である。PTが外にあるので生成・消滅の処理に不便が生ずる。
H2S3	IS CAP		PT CCB	C&D PSN IPC	プロセス間通信に関してはH4S1と同様である。プロセスの生成・消滅と同期管理が同じ場所にあるのでPTの操作は便利になる。
H1S4	IS		PT CCB	CAP C&D, PSN IPC	H2S3と同様の特徴をもつ。保護機能もホストから出したことにより、ホストでは割込等の必要最低限の機構を除いて、純粋なプロセス空間となる。

注
IPC：プロセス間通信
PSN：プロセスの同期管理
C&D：プロセス生成・消滅管理
CAP：プロセスの保護
IS：核利用インタフェスサポート
PT：プロセス管理テーブル
CCB：通信制御テーブル

第3章 NOS

3－1 NOSの現状

網向きOS・NOS I版は実装がほぼ終了し、ローカル通信，リモート通信ファイルシステム等の評価が行なわれている。(6)

第I版の構成は、H5S0タイプであり次節で述べるように幾つか改善する必要がある。第II版については、この点を考慮して、H4S1 タイプを採用し、現在実装中である。

3－2

NOS第Ⅰ版のソフトウェア構成上の検討 —— 実装法に関して

1) OSのすべての機能をホストに組み込んだ為プログラムが膨大になり、ミニコン(OKITAC4300C)ではメモリの増設が必要になったこと。(12kwのメモリを3回の増設し、36kwにした)

特にプロセス間通信のプログラムではバッファ領域がかなりの部分(2.5kw)を占め、相対的にユーザ領域が小さくなっている。

2) OSをアセンブラやμ-コード等の低レベル言語で書いたので、理解しにくく、デバッグにも非常に時間がかかったこと。又OS作成中に新たな機能が要求されると、その度追加していったのでプログラムがいびつになり、更に理解にくくなっている。

3－3 NOS第Ⅱ版の実装方針

前節の検討に基づいて、NOSⅡ版を以下の方針で作成している。

3-2の1) に対しては、OSの中で比較的下層に置くことができ、専用マシンの設計やインタフェースが容易であるプロセス間通信機構をホストから取り出す方式(H4S1)を採った。

3-2の2) に対しては、OS記述向きの高級言語で書くことにより改善を画った。具体的には、OS設計段階では、Pascalを用いて書き下し(フローチャートは使用せず)、実装には、PL/Mを用いて記述した。又、プロセスレベルでは、C-Pascalを発展させた言語の開発を行なっている。(7)

第4章 プロセス間通信サブシステムの設計方針

4－1 プロセス間通信の方式

4－1－1 ユーザの立場から

プロセスどおしはメッセージ単位で通信を行なうことができる。すなわちプロセスはSEND, RECEIVE等の通信用マクロを用いて通信要求を出す。

送信要求を出したプロセスと、受信要求を出したプロセスの相手がお互いに一致すれば、そこで通信契約が成立し、メッセージの転送が行なわれる。この際プロセスは、相手プロセスの存在ホストを意識せずに通信できる。

4－1－2 システムの立場から

通信マクロの処理では、プロセスの提出した引数を論理値から、処理値に変換したり、通信の資格を調べたりする。(通信のcapability) そこで通信可、ならば、通信相手の存在ホスト別に処理される。

ローカル通信(同一システム内)ならば通信契約成立以後、メッセージを送信領域から受信領域にコピーする。

リモート通信(他システム間)ならば通信要求をコマンドの形で他ホストのプロセスに伝え、通信契約成立後はメッセージをパケットに分割して転送する。

コマンドやメッセージパケットが到達したことの確認にACKを返している。

実際のパケット伝送には、伝送制御手順を使用し信頼性を向上するとともに、パケットの転送が無い時は、回線が接続されていることの確認の為に周期的に'遊びパケット'を転送している。

4－2 プロセス間通信に必要な機能

プロセス間通信機構では、主にプロセスが出した通信に関する情報を管理する部分と、実際の転送に関する管理をする部分との2つの機能に分けられる。

制御部には、個々の通信対応に通信制御テーブル(CCB)を置き、通信の状態やモードの管理や、ホストや受信コマンドからの通信情報の読み書き、通信制御の管理としてリモート通信の場合の他ホストへのコマンド作成やメッセージのパケット化の作業、ローカル通信の場合のメッセージの転送処理、

ホストに対して通信状況の通知，等の機能が必要である。

後者には、回線処理機能、すなわちパケットのルーティング，到達確認法（ACKや再送），中継処理や，回線伝送手順の管理，等が必要である。

このような事から、プロセス間通信用の専用マシンは、制御部としてのCCP (Communication Control Processor) と回線用としてのLCP (Line Control Processor) との二階層として構成した。

尚、CCPとLCPは、OSIのモデルでは、それぞれ Session, Transport 及び Network, Datalink, Physical 層に対応すると考えられる。

4－2－1

メッセージの転送法について

1) リモート通信

ホストからメッセージを取り出し、CCPでパケット化してLCPに送る方法は、CCPの負荷が大きいことや、ホスト↔CCP，CCP↔LCPと二回コピーを行なうので遅くなること等から得策ではない。そこでメッセージのパケット化はLCPで行なうことにした。すなわち、メッセージはホストとLCP間で直接受渡すこととし、LCPではCCPからのメッセージのパケット化情報に基づいて、パケットの分解及び合成を行なう。

2) ローカル通信

メッセージはホスト内主記憶での移動ですみパケット化の必要はない。そこでこの場合は、通信制御テーブルの効率的利用の面も考え合わせて、一旦ホストからメッセージを取り出し、CCPを経由して再びホストに書き込むという方式を採ることにした。

4－2－2

リモート通信のコマンド処理

ホストからの通信要求はCCPにおいて解釈し、SRQ (Send ReQuest), RRQ (Receive ReQuest) 等の計算機間コマンドを作成してLCPに送る。又、他ホストからの受信コマンドはLCPを経由してCCPに通知され、CCPで解釈して必要があればホストに通知する。

4－2－3

CCPで回復可能なエラー処理

制御テーブル等の領域が不足している場合は、自ホストに対してはその旨の通知，他ホストに対してはRejectコマンドを送り返している。

通信のタイムアウトの場合　メッセージの転送は終了しているのに一定時間経てもMAK (Message ACK) が来ない時はCancelコマンドを両プロセスに発行する。又、通信要求を出したが、一定時間以上経っても相手から通信要求が来ない時は、Rejectを発行する，等のことを行なっている。

4－2－4　LCPの処理

LCPでは、パケットの分解、合成及び伝送制御を行なう。伝送制御手順はNOS I 版では独自のものを使用していたが、効率等を考慮してHDLCを採用し、非同期平衡モードを簡単化したものを考えた。即ち、フレームとしてはRR, RNR, I, SABM, UAの5種類だけで切断モード，初期モードは省略した。

又、再送については、必ず抜けたパケットから後をすべて送る方式を採っているので伝送制御の効率は下がるがこうすることにより、パケット番号は常に順番が保たれるので、組み替えをする必要がなくなり、制御プログラム自体も簡単になるのでエラーが無い場合のスループットを上げる効果がある。

4－2－5

制御手順を用いない転送

ユーザからの通信要求にimmediateの指示があった場合は、SRQ, RRQ等の手続きを踏まずに、いきなり転送を行なうことができる。

4－3　ハードウェア構成

4－3－1　基本アーキテクチャ

基本アーキテクチャとしてマルチマイクロプロセッサを考える。マルチプロセッサにした意味は，論理階層をそのままハードウェアに対応づけることによってシステム全体の階層が明確になり，ソフトウェアも作り易くなるからで，またマイクロプロセッサを使用したのは，価格性能比及び扱い易さからの要請である。構成法の検討の際には、次のような点に留意した。

i)論理階層とハードウェアの対応
ii)転送ネックの回避
iii)モジュール構成，特にモジュール間接続の単純化
iv)全二重，80kbps の回線が8組サポートできること。

以上の方針に基いて，まず論理階層を図4.3.1のようにマッピングした。次に高速回線をサポートするので，回線対応にプロセッサを設け，segmenting，reassembling 及び伝送制御を行なわせる。これを回線制御プロセッサ（Line Control Processor：LCP）と呼ぶ。そしてLCP's の全体制御及びローカル通信を含めたプロセス間通信の管理を行うプロセッサを通信制御プロセッサ（Communication Control Processor：CCP）と呼ぶ。こうして1つのCCPモジュール，複数のLCPモジュールそしてHOST固有モジュールより構成されたシステムをCUPID（Control Processor of Interprocess communication for Distributed system）と呼ぶ。

User's processes		HOST
process management	kernel	
Control of I.P.C.	CCP	CUPID
Network level	LCP's	
Data Link Control		

fig.4.3.1 logocal hierarchy of NOS Ⅱ

4－3－2　結合方式

HOST－CCP－LCPの結合方式は、転送のオーバーヘッドを小さくする上で非常に重要な問題である。次のような方式を検討した。

まず，HOST－CUPID間の結合はDMAチャネルとプログラムチャネルの2本立にして，前者を，メッセージ本体及び制御情報の転送に用い，後者を，HOST－CUPID間のコマンド／レスポンスの転送に用いる。

次に，CUPIDの内部結合であるが，第1案としてCCP－LCP間をプログラム転送によるI/Oポートで接ぐ方式を検討した。しかし，この方式ではオーバーヘッドが大きく，回線速度に間に合わない可能性がある。そこでメッセージ本体の転送にDMACを使用して高速転送を行う方式が第2案である。しかし，この場合メッセージ本体は一度CCPを経由してLCPに転送されるので全体のスループットはCCPの処理能力によっておさえられ，その値は回線速度を下回ってしまう。以上のことから明確にコマンドとメッセージ転送ルートを分離し，LCPも直接HOSTのDMAチャネルに接続するという第3案を検討した。CCPとLCPが8台で9対1のマルチプレクサがデータ分配に必要になるのでHOST固有モジュールに含ませた。

4－3－3　第3案の詳細検討

この検討は，メッセージが処理されるフローを追うことより始めた（図4.3.2）。

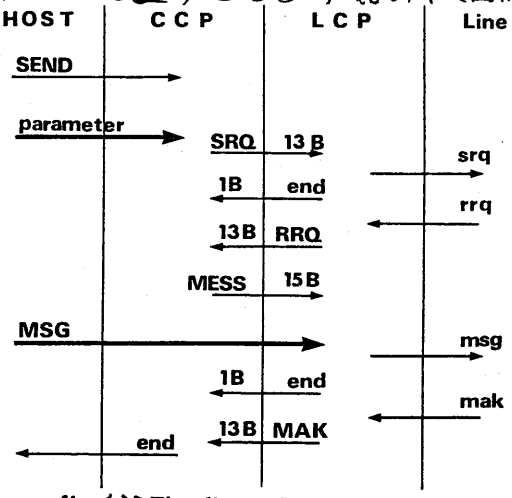

fig. 4.3.2 The flow of msg. processing

それによるとCCP－LCP間の通信量が5KBとかなり多いことがわかる。第3案ではPIOを使用して割込みによるバイト転送を行っているが，CCPのオーバーヘッドを計算すると，回線使用率0.3，LCP 8台の場合，約33%になる。回線使用率が0.5になると50%を越えるわけで，CCPはローカル通信の管理も行うので無視できない。これを小さくするためには割込みの回数を減らしてやればよい。そこでPIOの代わりに共有メモリとFIFOバッファを検討したが，実装の容易さから共有メモリ(shared memory: s.m.)を採用し，PIOは割込みのために残すことにした。これにより割込みは4回ですみ，共有メモリへのアクセス時間を加えてもオーバーヘッドを約1/5にすることができる。

4－3－4　最終案

以上のような検討の結果，図4.3.3の構成を採用した。LCPには，前項の検討にみるように共有メモリが加えられ，またCCPには，第1案から考えられていたものであるが，ローカル通信，即メモリtoメモリのDMAを行うための反射機構(Reflection Mechanism: RM)及び，将来核の他の部分を務す場合のことを考えて，バイト単位でCCPがHOSTの主記憶にアクセスできる機構(Pseudo Memory Mechanism: PMM)を加えている。以上，このような構成をとることにより，見通しが非常に明確になり，制御ソフトウェアも書き易いと思われる。

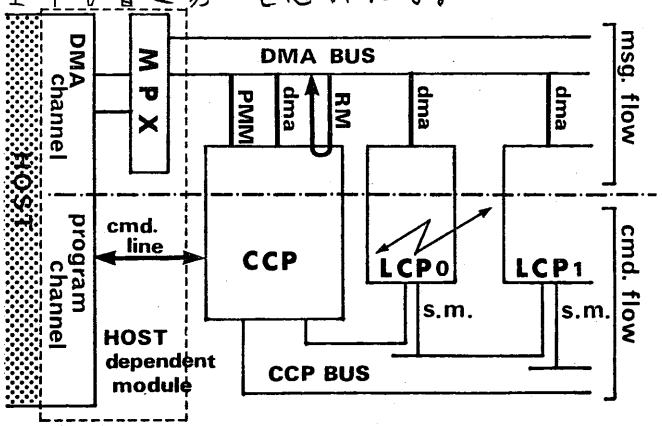

fig.4.3.3 The structure of CUPID

第5章　プロセス間通信サブシステムの実装

5－1　研究室の計算機

一般性を考慮して，4章で設計したプロセス間通信機構は，以下のような研究室内の異種計算機間で結合されている。

PPS-1とOKITAC4300Cとの間で一対
PPS-1とFACOM-U300との間で一対

PPS-1は主記憶を共有する3台のプロセッサから成るポリプロセッサであり，OKITACとUはミニコンである。

各ホストとCCP・LCP間では，DMAと割込の為のインタフェースをークッション置くだけでよい。CCPとLCPは，同一構造のものが使用できる。

5－2　ハードウェア

5－2－1　実装法

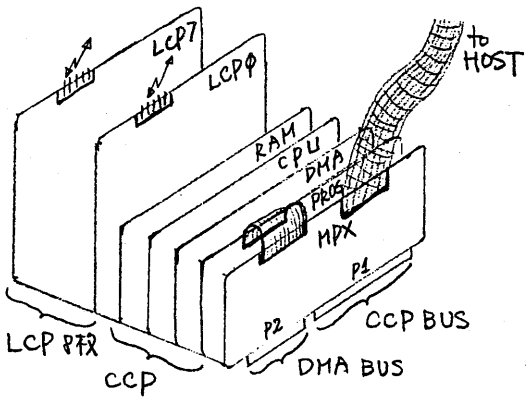

fig.5.2.1　実装図

CUPIDのハードウェアは，HOST固有モジュール，CCPモジュール，LCPモジュールに分けられる。これらモジュール間の接続を簡単にするために，図のようにバス方式を採用した。LCPを1枚の基板におさめることができたので，マザーボードに差すことによって容易に回線の増設ができる。また，HOSTにより固有の部分とマルチプレクサをまとめてMPX基板とし，それ以外の基板は標準化されている。バスとしては，基板の大きさ，コネクタのピン数等を考慮し，MULTIBUS® を採用した

P1をCCP BUS，P2をDMA BUSとして使用している。なお，CPU及びRAMは市販品を利用しているが，その他はすべて自作で，DMA，PROG，MPX はラッピング，LCP及びマザーボードは，アートワークデザインまでを行って，外注したプリント基板である。

CCP及びLCPのCPUとしては，現時点で最も価格性能比のよいZ-80A(4MHz)を採用した。次に各モジュールの内部構成について述べる。

5－2－2　内部構成

1) HOST固有モジュール

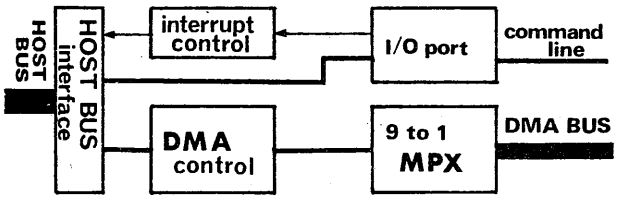

fig.5.2.2

MPX基板にまとめられているこのモジュールは，HOSTとのバスインタフェース，DMA制御，MPXer，コマンド／レスポンス用ポート及び割込み制御の各部より構成されている。MPXerを除いてHOSTに依存した形で実装される。通常TTL 90ケ程度を要する。HOSTとしてPPS-1，OKITAC 4300C，U-300を考えており，U-300については，このまま実装する予定であるが，他の2つについては従来のハードウェアの一部が流用できるため，この基板に実装されるのは，DMA BUSへの配線だけとなっている。

ii) CCPモジュール

CCPは，CPU，RAM，PROG，DMAの各基板から構成されている。CCP BUSのマスターはCPUとDMAであり，その制御にシリアルアービターを用いている。

○ CPU: 割込み優先度の制御のために8259A，時間監視のためにタイマー8253を使用している。(LSI 3ケ，SSI 4ケ)

○ PROG: コマンドライン用のポートとして8255Aを，LCPからの割込みをまとめて制御するために8259Aをスレーブモードで使用している。また，デバッグ用としてモニタROM 4KB及びコンソール用の8251Aがある。将来CP/M®をのせるためにパワーオンジャンプもつけてある。(LSI 5ケ，SSI 3ケ)

○ DMA: この基板は，DMACを除いてTTLで組んであり，最も高密度実装されている。P.M.M.は1BのDMAのために，メモリバンクを選択するメモリバンクレジスタ，アドレスバッファ，データ用双方向バッファ及び制御回路より構成され，CCPより8KBを1バンクとしてHOSTの主記憶にアクセスできるようになっている。反射機構はソースアドレス，ディスティネーション・アドレス及びバイトの各カウンタとテンポラリーレジスタ及び制御回路より構成されHOSTのM.to M.のDMAができるようになっている。

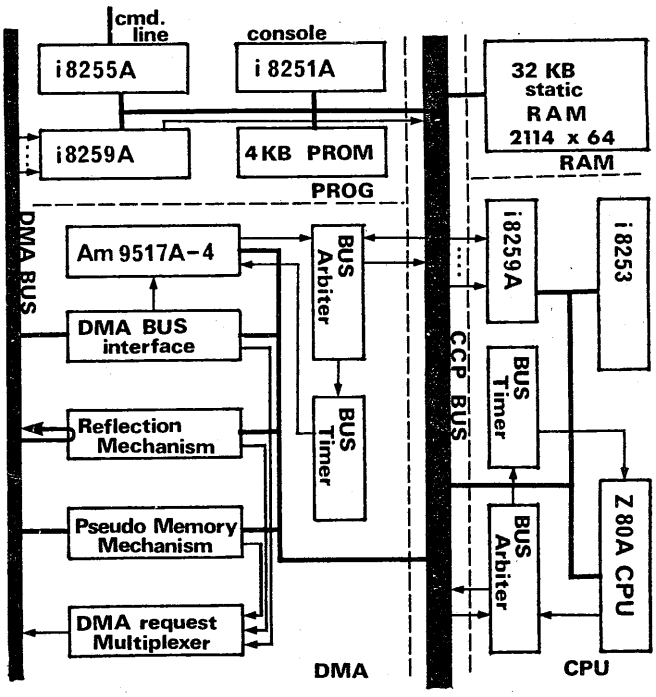

fig.5.2.3 The structure of CCP

DMA BUSインタフェースは，HOST－CCP間でワード単位でDMAが行なえるような制御ロジックである。また，以上3種の機構からDMA req.が出るので，そのMPXをするロジックが付加されている。DMAC としては，8257の改良版であるAm 9517Aを使用し，CH0及び1をCCP内

M. to M. に，CH2 を read，CH3 を write に割当てている。(LSI 1ケ，SSI 10ケ)

◦ RAM: 32KB を 4kbits static RAM 64ケで構成している。(RAM 64ケ，SSI 16ケ)

メモリマップを図5.2.4に，割込み優先度を以下に示しておく。

higher
- N.M.I.: Bus time out (10 msec)
- INT0: Timer
- INT1: LCP 0~7 (0~7間は rotating)
- INT2: HOST (read/write)
- INT3: DMAC (end/Bus time out)

lower
- INT4: RM

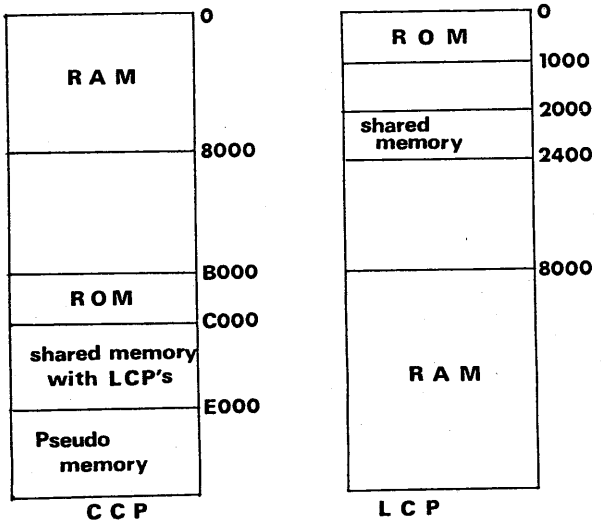

fig. 5.2.4 memory map

iii) LCP モジュール

LCP の構成の主眼は，80Kbps の全二重HDLC の回線のサポートである。そのためデータ量の多いメッセージ本体に関しては，徹底して DMA を使用し，CPU はデータの転送を一切しないですむようにしてある。

また，4-3-3の検討にあるようにCCPとの共有メモリをLCP基板内に設けている。(1kB static RAM) 相互排他の方法は，以下の通りである。制御回路は，LCP のアクセス要求とCCP のアクセス要求をLCPのクロックに同期して交互に検査する。アクセス要求が発見されると，メモリサイクルが開始し，そして終了すると検査は逆側を調べるようになっている。競合に対して LCP には $\overline{\text{wait}}$ ライン，CCP には $\overline{\text{XACK}}$ で打処している。

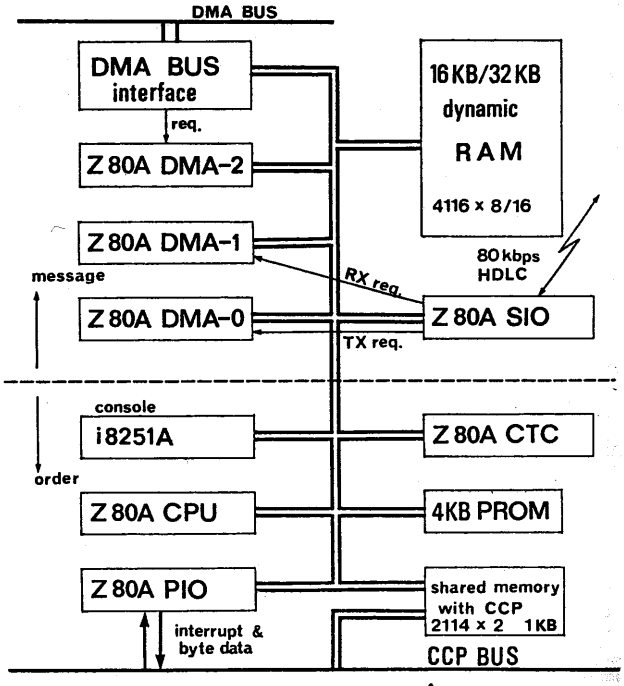

fig. 5.2.5 The structure of LCP

DMA BUS インタフェースは，CCP のそれと全く同じでアドレス・カウンタ(16bits)モード・レジスタ(R/$\overline{\text{W}}$, Byte/$\overline{\text{Word}}$)，入出力用ラッチ，8bits ⟷ 16bits の変換のための順序回路より構成されている。

DMAC としては，Z80A DMA を使用している。SIO の Rx用，Tx用に各1ケ，そしてHOST－LCP 間の転送用に1ケ(両者を切換えて使う)の計3ケを使用している。また，HDLC をサポートする LSI としては，Z80A SIO を使用した。

デバッグ用コンソールのためのボーレート分周及び伝送制御におけるタイマ用に Z80A CTC を，またCCP－LCP 間割込みとバイト転送のために Z80A PIO を使用した。

メモリマップを図5.2.4に，割込み優先度を以下に示す。

higher
- ◦ SIO Rx. (receive one character)
- ◦ SIO Rx. (special receive condition)
- ◦ DMA Tx. (end) DMA-0
- ◦ DMA Host (end) DMA-2
- ◦ PIO in (order request)

lower
- ◦ PIO out (response accepted)

注) 優先度は daisy chain による。

5－3 ソフトウェアの実装

5－3－1 インタフェス

1) ホストからCCPへ (hostparam)

プロセス間通信のユーザからの通信要求に関する引数

2) CCPからホストへ (ccpparam)

通信の状態をプロセスに通知する

3) CCPからLCPへ (lcpparam)

送信コマンドとメッセージ管理情報

4) LCPからCCPへ (lcpparam)

受信コマンドとメッセージ処理済情報

5－3－2 プロトコル

1) ユーザ

実際のシンタクスはプログラミング言語によるが、通信用マクロとして、SEND, RECEIVE, REJECT, CANSEL, RESETを用いている。

マクロの引数は、例えばSENDならば、通信相手名(論理値), 通信の領域や量やタイプ, リンク等を書き、ポートも用いることができる。(7)

2) CCPレベル

リモート通信の場合の制御コマンドには、SRQ, RRQ, CRQ(Continue ReQuest), REJCT(REJeCT), RST(ReSeT), CLOS(=cansel) MAKがあり、相手のCCPとやりとりをして、プロセス間通信の制御を行なう。(図4.3.2)

3) LCPレベル

ヘッダとテキストとから成るパケットを送受の単位とし、HDLCを伝送制御手順として用いる。

5－3－3 制御テーブル

CCPには通信制御テーブルがあり、LCPにはメッセージ管理テーブルがある。

5－3－4 プログラム

1) CCPでは、主にホストからの情報を解釈し、通信制御テーブルの読み書きや、通信の状態制御を行ない、必要ならばLCPにコマンド等を送り出す部分Tと、逆にLCPからのコマンド等を基にして同様な処理をし、必要ならば

```
COMKIND = (SRQ,RRQ,MAK,CRQ,RST,REJCT,CLOS,
           RECANY,RECSELECT,MESS,MESR,MESL,
           LOSRQ,LORRQ,LOCRQ,LORST,LOREJCT,
           LOCLS,SENBROAD)

COMMODE
link : (once,continuous) ;
combuffer : (basic,segmenting) ;
commember : comkind

IDENTITY
host : hostnumber;
processid : processnumber;
portid : portnumber

AREA
blocklength : blockinteger;
addr : addinteger

HOSTPARAM
usermsg : messinteger;
hostmode : commode;
hostid : identity;
hostarea : area;
selofremoteid : array (.1..5.) of identity

CCPPARAM
length : integer;
usermsg : messinteger;
hostmode : commode;
hostid : identity;
remoteid : identity;
infcode : array (.1..3.) of integer;
addcode : array (.1..3.) of blockinteger

LCPPARAM
lcpmode : commode;
hostid : identity;
remoteid : identity;
sysmsgnum : messinteger;
remotearea : area

COMBLOCKTABLE

ownnumber : cominteger;
chainnext : addinteger;
presentstatus : comstatename;
usermsg : messinteger;
hostmode : commode;
remotemode : commode;
hostid : identity;
remoteid : identity;
sysmsgnum : messinteger;
transarea : area;
receivearea : area;
addition : array (.1..5.) of identity


MESSAGETABLE

hostmode : commode;
remotemode : commode;
hostid : identity;
remoteid : identity;
sysmsgnum : messinteger;
messarea : area;
nummesp : array (.1..pnum.) of imep

                                IMEP
PACKET                          meptarea : area;
                                packetnum : integer;
pabmode : commode;              done : boolean
hostid : identity;
remoteid : identity;
sysmsgnum : messinteger;
msglength : blockinteger;
packnum : integer;
textlength : integer;
textcontents : array (.1..maxlength.) of char
```

ホストに知らせる部分R、から成る。他に割込処理などのルーチンがある。

この部分はPL/Mで記述され、Tで800行、Rで500行、コンパイルすると12.6kB程度になる。

2) LCPではCCPからのオーダを解釈し、メッセージ制御テーブルへの書き込みや、計算機間コマンド及び、メッセージのパケット化を行なう部分Ⓣと、受信したパケットをメッセージ化してホストに転送したり、計算機間コマンドの場合には、CCPに知らせる部分Ⓡから成るメッセージ制御プログラムをバックグラウンドで走らせ、伝送制御手順を実行するHDLCプログラムを割込処理に含ませている。

この部分は、現在コーディング中であるが、コード効率が、スループットに非常に影響するので、アセンブラで記述している。

5－4 OS記述言語の比較

NOS I版は、アセンブラやμ-コードで書かれているが、第II版は前述(3-2)のデバッグ効率の点などを考慮して、まず、Pascalを用いて論理構造を記述し、次いでそれをPL/Mに変換する、という方法を採った。これらの言語の使用経験から、言語の比較をすると、以下の通りである。

書き易さ　Pascalが最も良い。arrayやrecordタイプの要素がすべてidentifierで記述でき、ローケーションを全く気にしなくてよい。これに比べてPL/MはarrayのみでArray(n)の形で指定する必要がある。

if-then-elseやcase文を使用することにより場合分けの漏れが防げる。

Pascalでは状態遷移図を見ながら即プログラムに書き下すことができる。

デバッグ　アセンブラでは命令と番地とが一対一に対応しているので動作の場所が直接的にわかるが、PL/Mでは、プログラムのソースの場所と、実際の番地との変換表を見ながら、アセンブラのニモニックコードを解釈しなければならない点でめんどうである。しかし一つのルートについてデバッグできると、他のルートについては、同様の考え方が流用できる点では楽になる。

記述日数　第I版のアセンブラの場合は、設計しながら記述していったのでそのまま比較できないが、約1年かかり、Pascalは状態遷移図を見ながら約2か月/人、PL/MはPascalから変換して約1.5か月/人かかった。

第6章 現状及び評価

現在、ハードウェアは2台が、完成しており、HOSTに接続してデバッグを行なっている。また、ソフトウェアはCCPがデバッグ途中、LCPがコーディング中である。

実装と平行して、スループットの推定を行なっているが、リモート通信の場合には、回線速度で、ローカル通信の場合には、DMA BUSのスループットでそれぞれおさえられるが、ローカル、リモートをあわせると、単純和にはならずに、CCPでのオーバーヘッドによって20%程下がり、約340 KB/sec 程度であろうと思われる。

参考文献

1) 田中・元岡「研究用電子計算機網TECNET」信学研資 EC73-57

2) 和賀井・田中・元岡「網向きオペレーティングシステムについての一考察」信学研資 EC 77-43

3) H.TANAKA & T.MOTO-OKA : Network Oriented Operating System of TECNET J. of the fac of Eng. Univ of Tokyo (B) Vol. XXXV No4 1980

4) 正井・田中・元岡「機能分散型計算機における入出力サブシステムの試作」信学研資 EC77-4

5) 吉田・田中・元岡「機能分散型計算機におけるデータ・サブシステム」分散処理システム研究会5-4. 19

6) 本田公男「分散ファイルシステムにおける重複データ管理方式」東大工学部 修論 1980

7) 大和・田中・元岡「分散処理システム記述言語の設計とそのカーネルの製作」情処56年前期全国大 2E-7