研　究　速　報

UDC 681.32.06:621.382.049.77:800.92

マイクロ・コンピュータ・ネットワークのための システム記述言語MPL

A Language for Micro Computer Networks

浜田　喬*・山口　剛*
Takashi HAMADA and Takeshi YAMAGUCHI

1. はじめに

近年のLSI技術の進歩によって，マイクロ・プロセッサの価格は極めて安くなったが，周辺機器については機械的な部分も多いため，それほど安価ではない．したがって個々のマイクロ・コンピュータ・システムを結合し，周辺機器を共有すると性能価格比の良いシステムが構成できる．さらにこのようなネットワークにI/O制御用プロセッサ，通信用プロセッサなどの専用プロセッサを追加してシステム全体のパフォーマンスを向上させることもできる．

ところで，このマイクロ・コンピュータ・ネットワークのソフトウェアについて考えてみると，個々のノードごとにオペレーティング・システムを記述していたのでは見通しが悪く，作成の能率も良くない．そこでネットワーク全体を統一的に記述できるプログラム言語が必要である．またユーザの利用目的に適合させるため，あるいは要求の増加によってシステムを拡張するためにシステムの構成を変えた場合でも書直しの容易な記述性の良い高級言語が必要となる．

1973年にHoareがmonitorの概念を提案し，それをもとにBrinch Hansenは Concurrent Pascal (1975) をWirthは Modula (1977)を設計した．これらの言語は並列に動作する複数の processとそれらの共有変数を管理するmonitorによって構造的にオペレーティング・システムを記述するものであった．しかしこれらの言語はユニ・プロセッサを対象としており，プロセス間通信を共有変数によって行うため，共有空間をもたないマルチ・プロセッサ・システムを記述するには適さない．Hoareは1978年にプロセス同士が共有変数をもたないメッセージ通信の手法を発表している．

ここでは，以上のようなシステム記述言語の研究成果をもとに設計された，マイクロ・コンピュータ・ネットワークのためのシステム記述言語MPL (Multi-processors Programming Language) の基本仕様と実装方法について述べる．

2. MPLの概要

MPLの特徴は以下のとおりである．

(i) ネットワーク・システム全体を統一的にかつ効率よく記述できる．

(ii) プロセッサ間のメッセージ通信が行える．

(iii) 入出力および割込システムはIntel 8080の方式に合わせている．

MPLプログラムの構造を図1に示す．

processor module (PM)はネットワークの各ノード・プロセッサの動作を記述する．各プロセッサに共通な環境がglobal objectsである．PMの中には一般モジュ

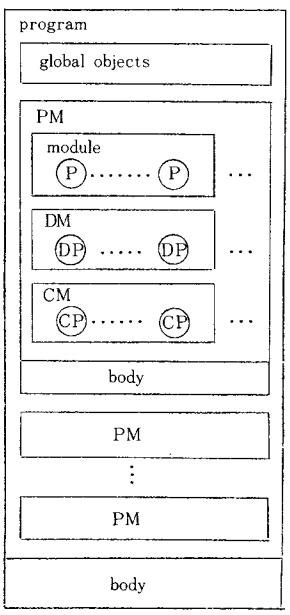

図1　MPLプログラムの構造

*東京大学生産技術研究所　第3部

ールとprocess （P），入出力デバイスを扱うdevice module（DM）とdevice process （DP），プロセッサ間の通信を扱う通信module （CM）と通信プロセス（CP），および各プロセス（P, DP, CP）間の通信・同期を記述するinterface moduleがある．メインプログラムの本体（body）では論理プロセッサと物理プロセッサを対応づける．

PM間の通信はすべてCMによって記述される．CMの中のCPがポートを介したバイト単位の標準通信手続input, outputを呼び出し，それらの要求が一致したときに通信が行われる．（図2）

DPおよびCPにはそれぞれ個有の割込みレベルが指定され，レベル優先割込みが行われる．

各PMに共通な部分を効率よく記述するためにプロセッサ指定と並列文を導入した．プロセッサ指定は種々の宣言がどのプロセッサについて有効であるかを指定する．並列文はプロセッサごとに実行する文を指定する．

3. MPLによる記述例

MPLによるプログラム例として，パケット通信システムを記述する．いま図3に示すような，4つのプロセッサによる単純なループ結合を考える．

各プロセッサ・モジュールの構成を図4に示す．各プロセッサ上にはパケットを生産し，宛名をつけて隣のプロセッサへ送出するプロセスPRODUCERⓅと，自分宛のパケットを消費し，その他はさらに隣へ送るプロセスCONSUMERⒸが存在する．また隣のプロセッサからポートを介して1バイトずつデータを入力する通信プロセスとしてINPORT，隣のプロセッサへ1バイトずつデータを出力する通信プロセスOUTPORTがある．

入出力パケットのバッファとしてIB，OBを用意する．これらのバッファは複数のプロセスによって共有されるので，図5に示すインタフェース・モジュール(IM)によって管理される．

手続きPUTはIBに1バイトずつつめ，手続きGETPCKTはIBがいっぱいになったらそれをパケットとして取り出す．

手続きGETはOBから1バイトずつ取り出し，手続きPUTPCKTはOBにパケットをつめる．

手続きINITIB，INITOBはそれぞれIB，OBを初期化する．IB, OBは仮引数であり，それらの実体はIMの外から与えられる．

以上の手続きはIMの先頭においてdefine宣言されており，これらはIMの外から呼び出される．

IMは通信モジュールによって囲まれており，そのプログラムを図6に示す．

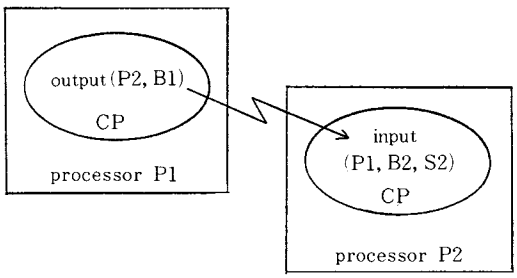

図2　プロセッサ間のメッセージ通信

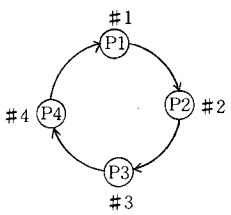

図3　単純なループ結合

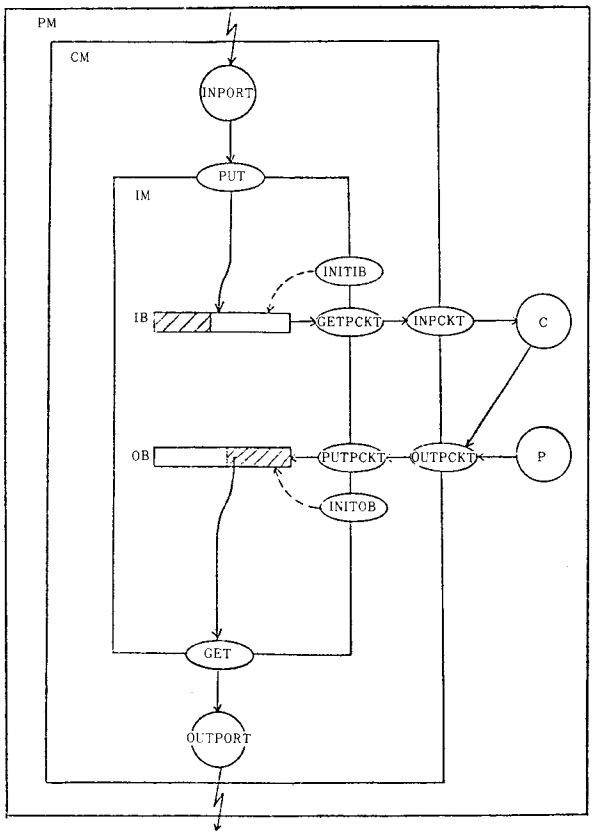

図4　プロセッサ・モジュールの構成

```
interface module PACKET_BUFFER;

  define PCKTBUF,PUT,GET,PUTPCKT,GETPCKT,INITIB,INITOB;
  use PCKTLNG;
  type
    PACKET=array 1:PCKTLNG of char;
    PCKTBUF=record
              BUF:PACKET:
              PTR:integer;
              FULL,EMPTY:signal
            end;

  procedure PUT(DATA:char; var IB:PCKTBUF);
  begin
    with IB do
      if PTR=PCKTLNG+1 then
        send(FULL);
        wait(EMPTY)
      fi;
      BUF(.PTR.):=DATA;
      PTR:=PTR+1;
    htiw
  end PUT;

  procedure GETPCKT(var P:PACKET; var IB:PCKTBUF);
  begin
    with IB do
      if PTR<PCKTLNG+1 then wait(FULL) fi;
      P:=BUF;
      PTR:=1;
      send(EMPTY)
    htiw
  end GETPCKT;

  procedure GET(var DATA:char; var OB:PCKTBUF);
  begin
    with OB do
      if PTR=PCKTLNG+1 then
        send(EMPTY);
        wait(FULL)
      fi;
      DATA:=BUF(.PTR.);
      PTR:=PTR+1
    htiw
  end GET;

  procedure PUTPCKT(P:PACKET; var OB:PCKTBUF);
  begin
    with OB do
      if PTR<PCKTBUF+1 then wait(EMPTY) fi;
      BUF:=P;
      PTR:=1;
      send(FULL)
    htiw
  end PUTPCKT;

  procedure INITIB(var IB:PCKTBUF);
  begin
    with IB do PTR:=1 htiw;
  end INITIB;

  procedure INITOB(var OB:PCKTBUF);
  begin
    with OB do PTR:=PCKTLNG+1 htiw;
  end INITOB;

end PACKET_BUFFER;
```

図5　インターフェース・モジュール

手続きINPCKTはIMの手続きGETPCKTを呼び出してパケットを取り出す．手続きOUTPCKTはIMの手続きPUTPCKTを呼び出してパケットを送出する．これらの手続きはdefine宣言されており，CMの外から呼ばれる．

通信プロセスINPORT，OUTPORTは標準通信手続きinput, output を呼び出して隣のプロセッサと通信を行う．通信する相手プロセッサとそれに接続されているポートのアドレスはportin，portout 宣言によって対応づけられている．これらはプロセッサごとに異なるアドレスなのでプロセッサ指定がされている．また通信の

```
communication module PORT_DRIVER;

  define INPCKT,OUTPCKT;
  use PCKTLNG;

  (:P1:):portin P4=...; (:P1:):portout P2=...;
  (:P2:):portin P1=...; (:P2:):portout P3=...;
  (:P3:):portin P2=...; (:P3:):portout P4=...;
  (:P4:):portin P3=...; (:P4:):portout P1=...;

  (* Here comes
           interface module PACKET_BUFFER
                                              *)

  var INBUF,OUTBUF:PCKTBUF;

  process INPORT(.4.);
    var TEMP,STATUS:char;
  begin
    loop
      parbegin
        (:P1:):input(P4,TEMP,STATUS);
        (:P2:):input(P1,TEMP,STATUS);
        (:P3:):input(P2,TEMP,STATUS);
        (:P4:):input(P3,TEMP,STATUS)
      parend;
      PUT(TEMP,INBUF)
    pool
  end INPORT;

  procedure INPCKT(var P:PACKET);
  begin GETPCKT(P,INBUF) end INPCKT;

  process OUTPORT(.3.);
    var TEMP:char
  begin
    loop
      GET(TEMP,OUTBUF);
      parbegin
        (:P1:):output(P2,TEMP);
        (:P2:):output(P3,TEMP);
        (:P3:):output(P4,TEMP);
        (:P4:):output(P1,TEMP)
      parend
    pool
  end OUTPORT;

  procedure OUTPCKT(P:PACKET);
  begin PUTPCKT(P,OUTBUF) end OUTPCKT;

begin
  INITIB(INBUF); INITOB(OUTBUF);
  INPORT; OUTPORT
end PORT_DRIVER;
```

図6　通信モジュール

相手プロセッサも各プロセッサごとに異なるので，input, output は並列文（parbegin… parend）の中で呼び出される．IB, OBの実体INBUF, OUTBUFはCMにおいて変数宣言されている．CMはプロセッサ・モジュール（PM）に囲まれておりそのプログラムを図7に示す．れている．CMはプロセッサ・モジュール（PM）に囲まれておりそのプログラムを図7に示す．

プロセスCONSUMERはパケットが自分宛ならば消費し，そうでなければ再び隣のプロセッサへ送出する．各プロセッサは自分の名前をSELFという定数として，プロセッサ指定による定数宣言をしている．プロセスPRODUCERは無限にパケットを生産し，隣のプロセッサへ送り出す．4つのプロセッサはこのようにただ一つのPMによってまとめて記述されるので効率が良い．PMを囲むプログラムを図8に示す．

定数PCKTLNG（パケット長）はすべてのPMに共通なのでグローバルに宣言されている．プログラムの本

```
(:P1,P2,P3,P4:):processor module PM;

  (* Here comes
         communication module PORT_DRIVER
                                          *)

  process CONSUMER;
    (:P1:):const SELF='P1      ';
    (:P2:):const SELF='P2      ';
    (:P3:):const SELF='P3      ';
    (:P4:):const SElf='P4      ';
    var P:PACKET; I;integer;
        ADDR:array 1:8 of char;
  begin
    loop
     INPCKT(P);
     I:=1;
     while I<=8 do
       ADDR(.I.):=P(.I.);
       I:=I+1
     elihw;
     if ADDR=SELF then
       (* consume packet data *)
     else OUTPCKT(P) fi
    pool
  end CONSUMER;

  process PRODUCER;
    var P:PACKET;
  begin
    loop
      (* produce packet data *)
      OUTPCKT(P)
    pool
  end PRODUCER;

begin
  CONSUMER;
  PRODUCER
end PM;
```

図7　プロセッサ・モジュール（PM）

```
program PACKET_COMM(P1,P2,P3,P4);

  const PCKTLNG=128;

  (* Here comes
         (:P1,P2,P3,P4:):processor module PM;
                                          *)

begin
  alloc(P1,1);
  alloc(P2,2);
  alloc(P3,3);
  alloc(P4,4)
end PACKET_COMM.
```

図8　プログラム

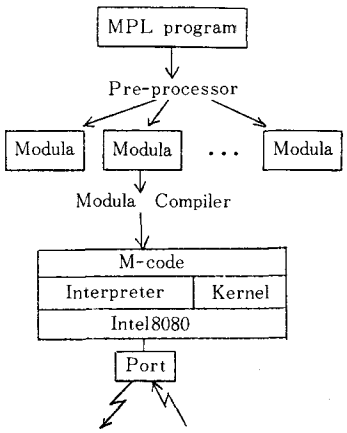

図9　MPLシステム

体においてalloc文によって，各論理プロセッサが物理プロセッサに割り当てられる．

4. MPLの実装

MPLの実装の概略を図9に示す．

MPLソース・プログラムはプリプロセッサによって各プロセッサに対して一つずつのModulaソースプログラムに変換される．変換されたModulaプログラムはコンパイラによってM-codeという仮想スタックマシンコードに落される．M-codeはマイクロ・コンピュータ上にロードされて，仮想計算機MPLマシンによって実行される．MPLマシンはインタプリタと核からなっており，プロセスの生成，信号（signal）に対するsend・waitによるプロセスの切換え，割込み待ち，プロセスのスケジューリングなどは核がサポートする．

5. おわりに

マイクロ・コンピュータ・ネットワークのためのシステム記述言語MPLを設計したが，今後はMPLを用いてマルチ・ユーザ・システムなどの実用的なソフトウェアを記述し，MPLの評価・検討を行いたい．

(1981年3月24日受理)

参 考 文 献

1) N. Wirth, "Modula: A Language for modular multiprogramming" SOFTWARE−PRACTICE AND EXPERIENCE, Vol.7 No.1 pp. 3- 35 (1977)

2) N. Wirth, "The Use of Modula", SOFTWARE−PRACTICE AND EXPERIENCE, Vol.7 No.1 pp. 37-65 (1977)

3) N. Wirth, "Design and Implementation of Modula" SOFTWARE-PRACTICE AND EXPERIENCE, Vol. 7 No. 1 pp. 67- 84 (1977)

4) I. D. Cottam, "Functional Specification of the Modula Compiler", Release 1, Univ. of York, Dept. of Computer Science, 1978

5) I. C. Wand and J. Holde, "MCODE: A Description of the bootstrapping interface of the Univ. of York Modula Compiler", York Computer Science Report No. 14, 1978

6) 山口剛 "マイクロ・コンピュータ・ネットワークとそのシステム記述言語" 修士論文　1981