LUND UNIVERSITY

A Multiprocessor DDC-Package

Breidegard, Björn; Hägglund, Tore; Gutman, Per-Olof; Nyqvist, Jean

1981

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Breidegard, B., Hägglund, T., Gutman, P.-O., & Nyqvist, J. (1981). *A Multiprocessor DDC-Package*. (Technical Reports TFRT-7216). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
4

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Download date: 02. Aug. 2026

CODEN: LUTFD2/(TFRT-7216)/0-065/1981)

A MULTIPROCESSOR DDC-PACKAGE

BJÖRN BREIDEGARD
TORE HÄGGLUND
PER-OLOF GUTMAN
JEAN NYQVIST

DEPARTMENT OF AUTOMATIC CONTROL
LUND INSTITUTE OF TECHNOLOGY
APRIL 1981

LUND INSTITUTE OF TECHNOLOGY
DEPARTMENT OF AUTOMATIC CONTROL
Box 725
S 220 07 Lund 7 Sweden

Document name
Internal report
Date of issue
April 1981
Document number
LUTFD2/(TFRT-7216)/0-065/(1980)

Author(s)
Björn Breidegard
Tore Hägglund
Per-Olof Gutman
Jean Nyqvist

Supervisor

Sponsoring organization

Title and subtitle

A multiprocessor DDC-package

Abstract

A one-processor DDC-package is modyfied to a multiprocessor DDC-package. It can be used e.g. in a hierarchical computer configuration, but other configurations are also possible. The programming language is Concurrent Pascal.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
65

Recipient's notes

Security classification

DOKUMENTDATABLAD RT 3/81

Distribution: The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.

A MULTIPROCESSOR DDC-PACKAGE

Björn Breidegard
Tore Hagglund
Per-Olof Gutman
Jean Nyqvist

Department of Automatic Control
Lund Institute of Technology
May 27, 1980

CONTENTS

CONTENTS

INTRODUCTION

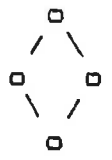
Our work is based on the existing one-processor package DDCPAC, OPCOM, REGUL (see e.g Tommy Essebo et al.: Facilities for concurrent processes in Pascal) which is assumed to be known by the reader.

We decided that any configuration of computers should be allowed, under the following conditions:

1. Each computer is started up by a terminal which is removed after the start-up (this restriction is caused by the limited number of ports).

2. One computer will retain the terminal during the running of the DDC-package. This computer must be called MAIN.

3. Any communication set-up between the computers is allowed, as long as it is consistent (i.e. if a message is received in the left port to be transmitted to another computer, it must not be sent on via the left port). Allowed configurations include:



See also the example of the chapter "How to start up".

The principle of our work was to change as little as possible in the existing DDC-package. As the original DDC-package was not constructed with intercomputer communication in mind, this means that there is a lot of room to improve our packaged. Some ideas are included in the chapter "Suggestions for improvements". These are left with a warm hand to our successors.

Our contribution mainly consists of

a. The code performing the transmission of data and text between the computers (see ch. 6 and 7, and the monitors namemonitortype, buffmontype, outleftmontype, outrightmontype, inlefttype, inrighttype).

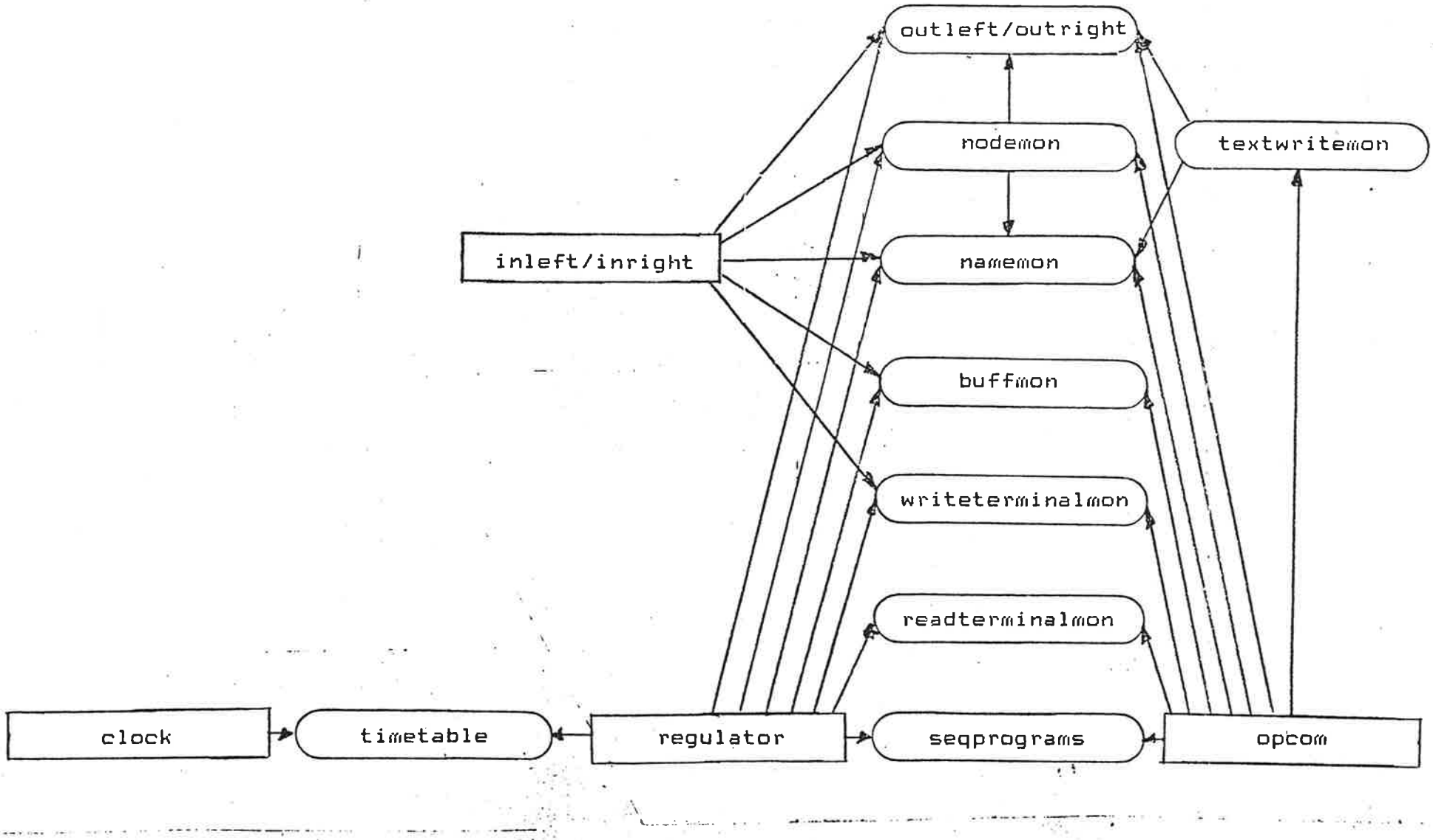
b. Code for initialization. The initialization was messy because the ports have different names if they lead to the terminal or to another computer (see point 1 above). See also the monitors namemonitortype, outleftmontype, outrightmontype, inlefttype, outrighttype, and procedure initialize of OPCOM.

c. Necessary changes in the ddcnodetype.

d. An ad hoc facility for synchronization of nodes in different computers (see ch. 8).

We are indebted to Leif Andersson and Sven Erik Mattson who wrote the kernels that enabled our program, and who advised and aided us in the best possible way.

MONITOR DIAGRAM



DDCPAA DOCUMENTATION

```
{******************
* namemonitortype *
******************}

 >Set of procedures which set a computer's name
 >or return information on a computer's name.

procedure entry setmyname(var thisname:nametype);
 >This procedure, called by OPCOM, sets myname to
 >the name of this computer.

procedure entry getmyname(var thisname:nametype);
 >This procedure returns the name of this computer.

procedure entry getmyname1(var thisname:nametype);
 >Same as getmyname.  Called by the READ procedure
 >in OPCOM.

procedure entry setleftnames(var thisname:nametype;
                             var i:integer);
 >This procedure, called by OPCOM, assigns names to
 >the left computers.

procedure entry setrightnames(var thisname:nametype;
                              var i:integer);
 >This procedure, called by OPCOM, assigns names to
 >the right computers.

procedure entry checkcomp(var thisname:nametype;
                          var answer:answertype);
 >This procedure informs the caller if thisname is
 >the name of this computer, if it is the name of
 >a left or a right computer, or if there is no
 >computer by that name.

{**************
* buffmontype *
**************}

 >This monitor consists of a ring buffer for text to
 >be used by the OPCOM of computers other than MAIN.

procedure entry write(var ch:char);
 >This procedure, called by INLEFT and INRIGHT, puts
 >a character into the ring buffer.

procedure entry read(var ch:char);
 >This procedure, called by OPCOM, fetches a
 >character from the ring buffer.

{*****************
* outleftmontype *
*****************}

 >This monitor sends text and data on the left channel.
```

DDCPAA DOCUMENTATION

```
procedure entry setxleft(var dev:iodevice);
 >This procedure, called by OPCOM, designates one
 >of the output channels as the left channel.

procedure entry getxleft(var dev:iodevice);
 >This procedure, called by INLEFT, returns the
 >channel number of the left channel.

procedure entry dataout(var comp,node:nametype;var outvalue:univ rstring);
 >This procedure sends data on the left channel.

procedure entry asciiout(var comp:nametype;var ch:char);
 >This procedure sends text on the left channel.

{******************
* outrightmontype *
******************}

 >This monitor is the same as OUTLEFTMONTYPE
 >with right substituted for left.

{**********************
* readterminalmontype *
**********************}

Procedure entry read(var character:char);
 >This procedure reads from the keyboard when myname=MAIN.
 >It is not used in the other computers.

{***********************
* writeterminalmontype *
***********************}

Procedure entry write(var ch:char);
 >This procedure writes on the terminal when myname=MAIN.
 >It is not used in the other computers.

{******************
* textwritemontype *
******************}

procedure entry textwrite(var dest:nametype;var ch:char);
 >This procedure, called by OPCOM if myname=MAIN,
 >sends text generated on the terminal to computers
 >other than MAIN.

begin
  namemon.checkcomp(dest,answer);
  case answer of
    left:  outleft.asciiout(dest,ch);
    right: outright.asciiout(dest,ch)
  end;{case}
end;
```

DDCPAA DOCUMENTATION

```
{******************
* seqprogramstype *
******************}

 >This monitor is used at system startup only.

{******************
* nodemonitortype *
******************}

ddcnodetype=record
 >See definition in program REGULATOR.

 >This monitor is a set of procedures which handles
 >operations performed on the nodes.

procedure entry regputgetnode(var node:ddcnodetype;
                              var anyfound:boolean);
 >This procedure, called by REGULATOR, transfers
 >nodes between the active list and the work space
 >of REGULATOR.

procedure entry getoutvalue(var number:integer;
                            var outval:real);
 >This procedure, called by REGULATOR, returns the
 >output value from the node with the given pointer.

procedure lookup(var name:nametype; var ptr:integer);
 >This procedure returns the pointer to the node with
 >the given name.

procedure entry getrealtime(var rtime:integer);

procedure entry putoutvalue(var node:nametype;var value:univ real);
 >This procedure, called by INLEFT and INRIGHT, sets
 >the outvalue of an innode.

procedure entry opgetnode(var node:ddcnodetype;
                          var notfound,notspace:boolean);
 >This procedure, called by OPCOM in response to
 >the command OPEN, copies an existing node into
 >the work space of OPCOM or informs the caller
 >that no node exists by that name.

procedure entry delete(var nodename:nametype;var result:integer);
 >This procedure, called by OPCOM in response to
 >the command DELETE, transfers a node from the
 >active to the inactive list.

procedure entry link(var node:ddcnodetype);
 >This procedure, called by OPCOM in response to
 >the command LINK, copies a node from the work
 >space of OPCOM to a node in the active list.
```

DDCPAA DOCUMENTATION

```
procedure entry checkname(var name:nametype;
                          var ptr,pri:integer);
 >This procedure returns the pointer to and the
 >priority of the node with the given name.

procedure entry datawrite(var comp,node:nametype;var outvalue:real);
 >This procedure, called by REGULATOR, directs data
 >to a node in the appropriate computer.

{************
* timetable *
************}

 >This monitor synchronizes the action of the
 >REGULATOR with the realtime clock.

procedure entry wait;
 >This procedure, called by REGULATOR, delays regqueue.

procedure entry schedule;
 >This procedure, called by CLOCK, continues regqueue.

{*************
* inlefttype *
*************}

 >This process reads continuously on the left
 >channel and sends the information to the
 >appropriate place.
```

DESTINATION	ASCII	NUMERIC
not myself	outright	
myself(=MAIN)	writeterminalmon.write	putoutvalue
myself(<>MAIN)	buffmon.write	

```
inlefttype=process(nodemon:nodemonitortype;namemon:namemonitortype;
                   buffmon:buffmontype;outright:outrightmontype;
                   outleft:outleftmontype;
                   writeterminalmon:writeterminalmontype);

{**************
* inrighttype *
**************}

 >This process is the same as INLEFTTPYE with
 >right substituted for left.
```

DDCPAA DOCUMENTATION

```
{************
* clocktype *
************}

 >This process is the alarmclock for TIMETABLE.

type clocktype=process(timetable:timetabletype);

{****************
* regulatortype *
****************}

regulatortype=process(buffmon:buffmontype;
                      readterminalmon:readterminalmontype;
                      writeterminalmon:writeterminalmontype;
                      seqprograms:seqprogramstype;
                      nodemon:nodemonitortype;
                      timetable:timetabletype;
                      namemon:namemonitortype;
                      outleft:outleftmontype;
                      outright:outrightmontype);

procedure entry read(var character:char);
  if myname=MAIN then readterminalmon.read(character)
  else buffmon.read(character)

procedure entry write(var character:char);
  case MAIN=
    myself: writeterminalmon.write(character);
    left:   outleft.asciiout(MAIN,character);
    right:  outright.asciiout(MAIN,character);

procedure entry wait;
  timetable.wait

procedure entry regputgetnode(var node:ddcnodetype;
                              var anyfound:boolean);
  nodemon.regputgetnode(node,anyfound)

procedure entry getoutvalue(var number:integer;
                            var outvalue:real);
  nodemon.getoutvalue(number,outvalue)

procedure entry getrealtime(var rtime:integer);
  nodemon.getrealtime(rtime)

procedure entry datawrite(var comp,nodename:nametype;
                          var sendvalue:real);
  case comp is to the
   left:        outleft.dataout(comp,nodename,sendvalue);
   right:       outright.dataout(comp,nodename,sendvalue);

procedure entry getmyname(var thisname:nametype);
  namemon.getmyname(thisname)
```

DDCPAA DOCUMENTATION

```
{************
* opcomtype *
************}

opcomtype=process(buffmon:buffmontype;
                  readterminalmon:readterminalmontype;
                  writeterminalmon:writeterminalmontype;
                  textwritemon:textwritemontype;
                  seqprograms:seqprogramstype;
                  nodemon:nodemonitortype;
                  namemon:namemonitortype;
                  outleft:outleftmontype;
                  outright:outrightmontype);

procedure entry read(var character:char);
  if myname=MAIN then readterminalmon.read(character)
  else buffmon.read(character);

procedure entry write(var character:char);
  case MAIN=
    myself: writeterminalmon.write(character);
    left:   outleft.asciiout(MAIN,character);
    right:  outright.asciiout(MAIN,character)

procedure entry opgetnode(var node:ddcnodetype;
                          var notfound,notspace:boolean);
  nodemon.opgetnode(node,notfound,notspace)

procedure entry link(var node:ddcnodetype);
  nodemon.link(node)

procedure entry delete(var nodename:nametype;
                       var result:integer);
  nodemon.delete(nodename,result)

procedure entry checkname(var name:nametype;
                          var ptr,pri:integer);
  nodemon.checkname(name,ptr,pri)

procedure entry textwrite(var dest:nametype;
                          var ch:char);
  textwritemon.textwrite(dest,ch);

procedure entry checkcomp(var thisname:nametype;
                          var answer:answertype);
  namemon.checkcomp(thisname,answer);

procedure entry setmyname(var myname:nametype);
  namemon.setmyname(myname);

procedure entry setleftnames(var thisname:nametype;
                             var i:integer);
  namemon.setleftnames(thisname,i);
```

DDCPAA DOCUMENTATION

```
procedure entry setrightnames(var thisname:nametype;
                              var i:integer);
  namemon.setrightnames(thisname,i);

procedure entry setxleft(var dev:iodevice);
  outleft.setxleft(dev);

procedure entry setxright(var dev:iodevice);
  outright.setxright(dev);

{*******
* main *
*******}

var readterminalmon:readterminalmontype;
    writeterminalmon:writeterminalmontype;
    textwritemon:textwritemontype;
    seqprograms:seqprogramstype;
    nodemon:nodemonitortype;
    timetable:timetabletype;
    clock:clocktype;
    regulator:regulatortype;
    opcom:opcomtype;
    namemon:namemonitortype;
    buffmon:buffmontype;
    outleft:outleftmontype;
    outright:outrightmontype;
    inleft:inlefttype;
    inright:inrighttype;
```

OPCOM DOCUMENTATION

```
program opcom;
entryprocedures=opgetnode,delete,link,checkname,setmyname,
                setleftnames,setrightnames,checkcomp,
                textwrite,setxleft,setxright
 >See comments in main program - DDCPAA.

ddcnodetype=record
 >See definition in program REGULATOR.

procedure error(err:errors);
 >This procedure writes error messages on the terminal.

procedure writeaddress(addr:address);
 >This procedure, called by SHOW, writes an address on the
  terminal.

procedure readname;
 >This procedure reads a node name from the terminal.

procedure open;
 >This procedure copies a node into the work space of
 >OPCOM if a node by that name exists, otherwise
 >a new node is created.

procedure show;
 >This procedure displays on the terminal the values of
 >the node in the work space of OPCOM.

procedure link;
 >This procedure copies the node in the work
 >space of OPCOM into the active list.

procedure delete;
 >This procedure deletes a node from the active list.

procedure readreal(var variable:real;
                   tag:nodetype);
 >This procedure reads a real number from the keyboard
 >if the program is running in MAIN.  If the program is
 >running in a program other than MAIN, a real number
 >is read from the ring buffer.

procedure readint(var variable:integer);
 >This procedure reads an integer.  See READREAL.

procedure readaddr(var signal:address;
                   tag:nodetype;
                   outsignal:boolean);
 >This procedure reads an address.  See READREAL.

procedure readcomp(var comp:nametype);
 >This procedure reads a computer name.  See READREAL.

procedure setvariable;
 >This procedure changes values in the parampart of a node.
```

OPCOM DOCUMENTATION

```
procedure initialize;
 >This procedure initializes the variables of OPCOM
 >and asks for the structure of the communication link
 >relative to this computer (which computers lie to
 >the left, which computers lie to the right, and what
 >is the name of this computer).

begin {opcom}
  initialize;
  if myname=MAIN then                    >If myname=MAIN then read the first
    repeat                                word of a new command from the
      i:=0;                               keyboard.
      while not eoln do begin
        read(ch);
        i:=i+1;
        command[i]:=ch;
        buff[i]:=ch;
      end;{while}
      if command='BYE      ' then
        repeat
          write('>');                    >Write a prompting character.
          read(actcomp);                 >Read a computer name.
          checkcomp(actcomp,exist);      >Check if the computer name exists.
          if exist=nonexistent then writeln('no computer name')
          else write(actcomp,'>')
        until exist<>nonexistent         >Repeat until a legal computer name
                                          is given.
      else if actcomp=MAIN then          >If command is not BYE and communication
        begin                             is with MAIN, then process the command.
          case command of
            open       : open;
            show       : show;
            link       : link;
            delete     : delete;
            other      : setvariable
          end;{case}
          write(myname,'>')
        end
      else begin                         >If communication is with another
        while not eoln do begin           computer, then read the remaining
          i:=i+1;                         part of the command.
          read(buff[i]);
        end;{while}
        for j:=1 to i do textwrite(actcomp,buff[j]); >Send the complete
      end;                                            command to the
                                                      appropriate computer.
      readln;
    until false
  else repeat                            >If myname<>MAIN, then read the
    read(command);                        command from the ring buffer.
    case command of                      >Process the command.
      open       : open;
      show       : show;
      link       : link;
```

OPCOM DOCUMENTATION

```
      delete      : delete;
      other       : setvariable
    end;{case}
    write(myname,'>')
  until false;
end.{opcom}
```

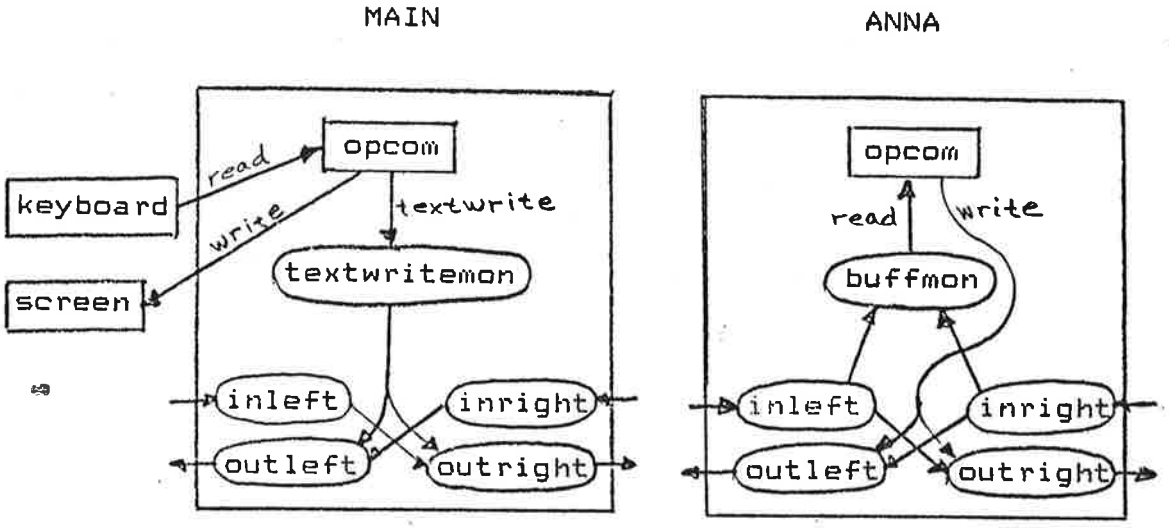
REGULATOR DOCUMENTATION

```
program regulator;

function adin(chan:integer):real; external;
 >Analog-to-digital input.

procedure daout(chan:integer; value:real); external;
 >Digital-to-analog output.

entryprocedures=wait,regputgetnode,getoutvalue,
                getrealtime,datawrite,getmyname
 >See comments in main program - DDCPAA.

DEFINITION OF NODE STRUCTURES

type nametype=array[1..10] of char;

     address=record
       name:nametype;
       number:integer
     end;{address}

     nodetype=(innode, PIDnode, outnode);

     paramtype=record
       goalcomp:nametype;
       intime,outtime,lasttime:integer;
       case tag:nodetype of
         innode:(insig,outplace:address;
                 scale,filter:real);
         PIDnode:(PIDin,PIDref,PIDout:address;
                  k,ti,td,alfa,beta,limit:real);
         outnode:(insig1,insig2,insig3,
                  insig4,output:address;
                  scal1,scal2,scal3,scal4,level:real)
     end;{paramtype}

     statetype=record
       filterstate,yold,ipart,dpart:real;
     end;{statetype}

     ddcnodetype=record
       fwd,bwd:integer;
       name:nametype;
       priority:integer;
       period,counter:integer;
       outvalue:real;
       parampart:paramtype;
       state:statetype
     end;{ddcnodetype}

function ulim(u,lolim,hilim:real):real;
 >This function puts an upper and lower limit
 >on the signal u.
```

REGULATOR DOCUMENTATION

```
function nodevalue(var addr:address):real;
 >This function gets a value from the AD-converter
 >or from the outvalue of the addressed node.

procedure setouttime(var intime,outtime,lasttime,
                         period:integer);
 >This procedure, called by INREG, puts the realtime
 >at which the outvalue is taken from a node into
 >outtime.  (For more information, see the chapter
 >on synchronization.)

procedure inreg;
 >This procedure regulates innodes.

Procedure PIDreg;
 >This procedure regulates PID-nodes.

Procedure outreg;
 >This procedure regulates outnodes.

begin {regul}
  getmyname(myname);                    >Get the name of this computer.
  repeat
    wait;                               >Wait until the next clock tick.
    regputgetnode(node,anyfound);       >Move the node in the work space of
                                         REGULATOR back to the active list, and
                                         move the next node with counter<=0 to
                                         the work space of REGULATOR.
    while anyfound do begin
      case nodetype of
        innode:  inreg;
        PIDnode: PIDreg;
        outnode: outreg
      end;{case}
      regputgetnode(node,anyfound)
    end {while}
  until false;
end {regul}.
```

Transmission of text between two computers.



Suppose that we want to open the node PID in the computer ANNA.

ANNA >OPEN PID

The string OPEN PID is sent from OPCOM in MAIN character by character. E.g. the O is sent by the command

textwritemon.textwrite('ANNA ','O')

In textwritemon, the procedure checkcomp in namemon is called to check whether ANNA is to the left or to the right.

namemon.checkcomp('ANNA ',answer)

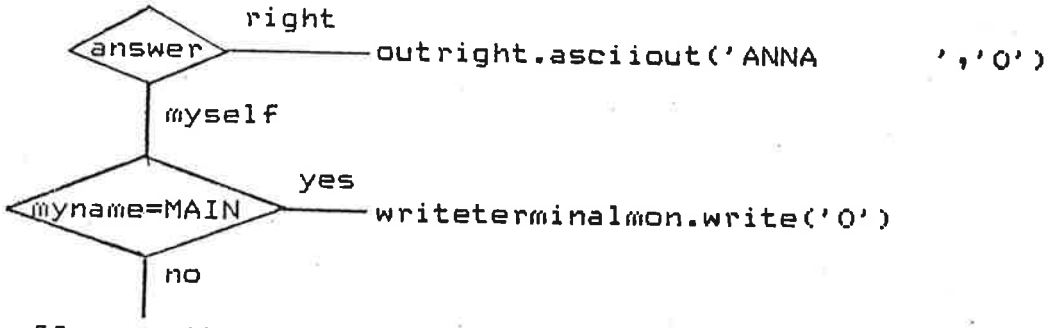
asciiout is then called in outleft or outright depending on if the answer is left or right.

outright.asciiout('ANNA ','O')

In outright, the following 12 characters are sent on the right channel by io-routine: @ (to inform the receiver that text and not data is comming), A, N, N, A, , , , , , , O.

In ANNA the ~~monitor~~ process inleft takes care of the information. The command checkcomp in namemon is called to check if the message is meant for ANNA or another computer.

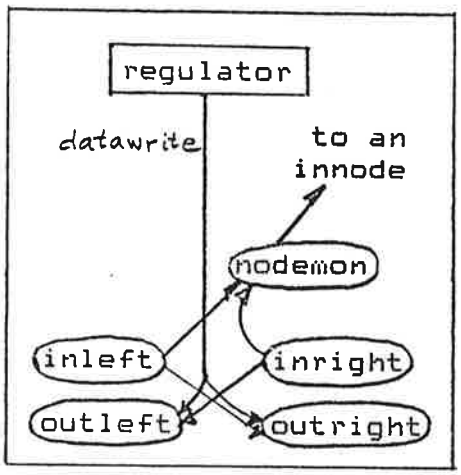
```
namemon.checkcomp('ANNA      '.answer)
```



In this case the character 0 is put into the ringbuffer in buffmon.

Finally the readcommand, that continuously checks if there is something to fetch from buffmon, brings the 0 into OPCOM. At the same time, an 0 is sent back to the screen from ANNA, to make it possible for the operator to check that the message is correctly transferred.

Transmission of data between two computers.



Suppose that we want to send the value 7.35 from the outnode OUT in ANNA to the innode IN in BRITTA. (We are not allowed to send to anything but innodes in other computers).

In the parampart of OUT, GOALCOMP must be set to BRITTA and OUTPUT to IN. In IN, INSIG must be set to EXT.

The transmission is initialized by the command

```
datawrite('BRITTA    ','IN        ',7.35)
```

in ANNA. The command

```
namemon.checkcomp('BRITTA    ',answer)
```

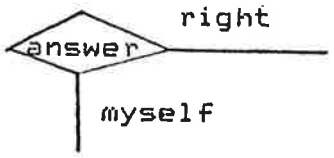
checks whether BRITTA is to the left or to the right. If the answer is right, the procedure

```
outright.dataout('BRITTA    ','IN        ',7.35)
```

will be called. In outright, the last variabel is declared as a universal string, and not as real. The real value is represented by 4 bytes, and they will be sent as characters. The following 25 characters are now sent on the right channel by the io-routine: # (to inform the receiver that data and not text is comming), B, R, I, T, T, A, , , , , I, N, , , , , , , , , , value[1], value[2], value[3], value[4].

In BRITTA, inleft takes eare of the incomming information. The command checkcomp in namemon is called to check if the information is meant for BRITTA.

```
namemon.checkcomp('BRITTA     ',answer)
```




```
outright.dataout('BRITTA     ',
                 'IN         ',7.35)

nodemon.putoutvalue('IN         ',7.35)
```

Finally putoutvalue in nodemon sets outvalue in IN equal to 7.35.

THE NEW COMMAND SYNC

Background

Every node produces, as a result of the computations when it is implemented, a real number: the outvalue (see the definiton of ddcnodetype). The outvalue has one or more destinations - another node in the same computer, the DA-converter of the same computer, or a node in another computer. Moreover the outvalue is always put into the field "outvalue" of the ddcnodetype record.

Also, the AD-converter produces real numbers destined for node(s) in the same computer.

Modes of transport

There are two ways of transporting the outvalue (AD-value) to its destination, an active mode, and a passive mode.

Definition: The active mode of transport (A) means that the producing node actively sends the outvalue to the destination.

Definition: The passive mode of transport (P) means that the producing node (AD-converter) only sets the field outvalue (corresponding) and that the destination has to pick up the outvalue there.

The different ways of transport are used as follows (note that some routes are prohibited: x):

Destination: The same computer	Destination: Another computer	Source:	Source: IN-node PID-node OUT-node	Source: AD-conv
IN-node PID-node OUT-node			P	P
DA-conv			A	x
	IN-node		A	x
	PID-node OUT-node DA-conv		x	x

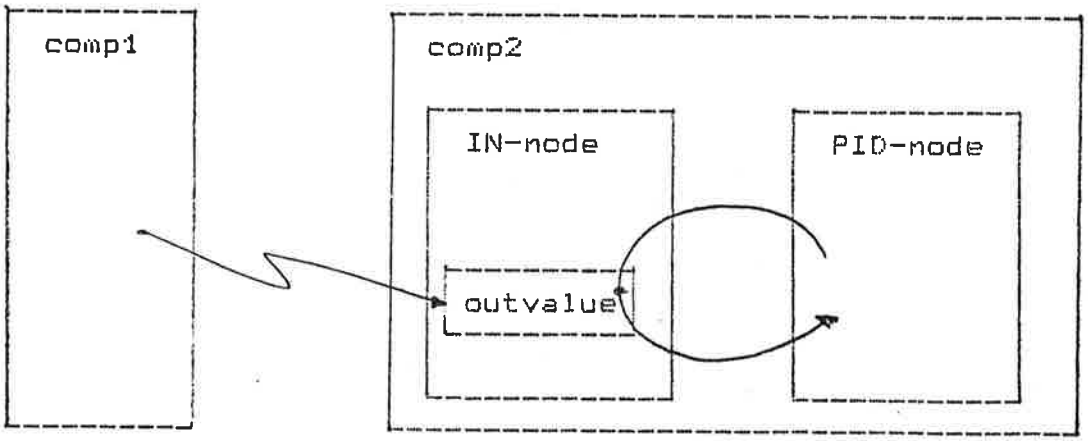
SYNC

When a value is actively sent to an IN-node of another computer, this IN-node solely serves as a station of reception. The IN-node gets the value into the field outvalue, and does not process it (it can, however, send it on to a DA-converter, or to an IN-node of yet another computer). See the procedure inreg of REGULATOR.

The problem

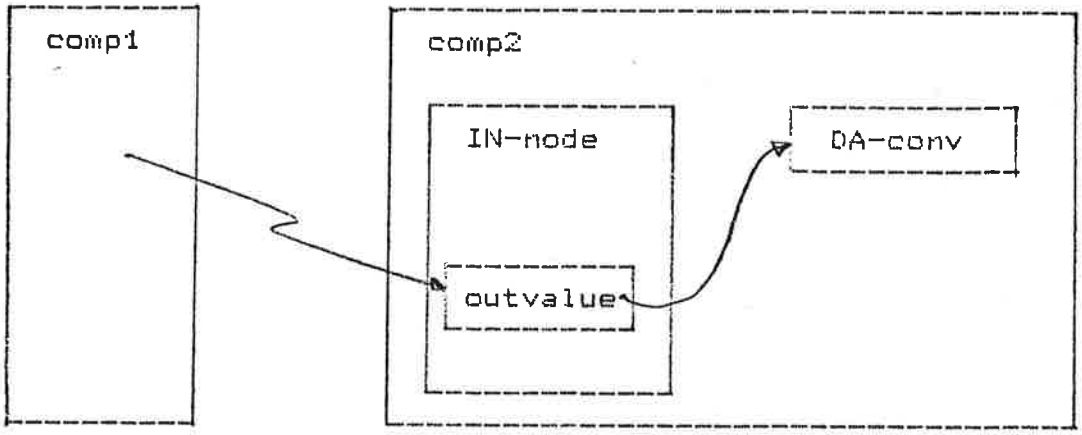
Typically the following synchronization problem occurs:

Case_1: A node in comp1 sends actively a value to an IN-node of comp2. The value now rests in the field outvalue waiting for e.g. a PID-node of comp2 to pick it up. The delay between the time instant when the value was placed in the field outvalue of the IN-node and the time instant it is picked up by the PID-node may cause severe problems. Clearly a synchronization between the time of reception by the IN-node (intime) and the time when the value is picked up (outtime) is needed.



Case_2: The same problem occurs if the IN-node in comp2 is to send its outvalue to a DA-converter. The interval between the time of reception (intime) and the time when the IN-node is interpreted, and thus sends the value to the DA-converter (outtime) should be as short as possible.

SYNC



It is not a trivial task to synchronize the clocks of the computers. If this was done, it would be possible to ensure that the counters (see the record ddcnodetype) of the connected nodes in comp1 and comp2 were synchronized with a predetermined lag, so that the nodes are executed in the correct order with minimum delay. If the clocks are not synchronized it seems to be quite difficult to synchronize the counters.

Another problem is how e.g. a temporary traffic jam on the link connecting comp1 and comp2 which causes a delay in the transfer of the value to the IN-node of comp2, should influence the execution of nodes in comp2 needing the value. Should the execution be delayed, and, if so, for how long? This problem has not been solved, but a field lasttime is included in the ddcnodetype. This field contains the last time instant the regular data transfer went wrong (see below).

Principle of solution

It was decided not to try to synchronize the different computers. As the aim is to minimize the delay between intime and outtime, all actions could be taken in comp2. The following situations may occur both in case 1 and case 2:

```
intime   *---------*---------*---------  /ticks/
outtime  -*---------*---------*--------  /ticks/
```

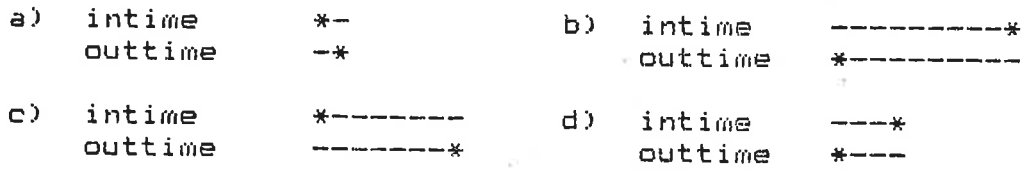
Acceptably small delay
(time between stars = sampling period)

```
intime   *---------*---------*---------  /ticks/
outtime  -------*---------*---------*--  /ticks/
```

Unacceptably long delay

SYNC

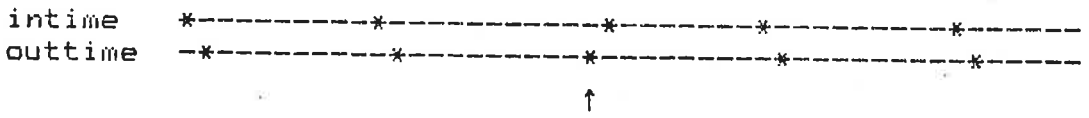
Let the operator have access to the field intime and outtime of the receiving IN-node when commanding OPEN-SHOW. He will then get the last intime and the last outtime, at the occasion the command OPEN was executed. He will get one of the following four possible pictures:



Knowing the sample period, the operator has no trouble to distinguish which of the cases a, b, c, d belong to the case of acceptably small delay respectively the case of unacceptably long delay.

By changing the counter of the node in comp2 that transfers the value (case 1: the PID-node, case 2: the IN-node itself), the time instant of the execution of the node is changed, and the delay between intime and outtime is altered. This is done by the commands OPEN-SYNC-LINK, see below.

The problem of occasional irregularities in intime has been tackled in the following way. Assume that you have been able to synchronize the intime and outtme nicely. Sometimes, however, the transfer of data gets delayed:



In IN-node of comp2 the field lasttime := outtime, if (outtime - intime) >= period/2. See the procedure setouttime in REGULATOR, and procedure entry getoutvalue of nodemonitortype in DDCPAA. In the above figure lasttime will be set at the arrow.

By commanding OPEN-SHOW for the IN-node of comp2 a few times, the operator can get an idea if and how often lasttime is updated. If it occurs too often, he can adjust outtime by the command SYNC, as mentioned above.

Implementation

Ddcnodetype has been changed to include intime, outtime, and lasttime.

Intime is set in procedure entry putoutvalue of nodemonitortype in DDCPAA.

Outtime and lasttime are set by either procedure entry getoutvalue of nodemonitortype in DDCPAA (case 1) or by the procedure setouttime of REGULATOR (case 2).

Intime, outtime, lasttime take on integer values denoting real time in number of ticks.

The procedure show of OPCOM has been changed for the case of innode, to include a display of intime, outtime, and lasttime.

In procedure open of OPCOM the statement 250: node.counter:=0 has been included. This is because the field node.counter is used to store the argument of the command SYNC.

SYNC <integer> is a new command in form of a "subcommand" that can be performed only after and OPEN-command has been executed. It is included in the variable vars of OPCOM. SYNC <integer> causes node.counter:=<integer>. See statement 430 in the procedure setvariable of OPCOM. See also the next paragraph.

Procedure entry link of DDCPAA has got a new statement 516: counter:=counter + node.counter. Thus, if SYNC was used before commanding LINK, the value of the counter of the node in the active list is changed, and the coming instants of execution of the node will be hastened or delayed. As is well known, each time the active list is gone through, counter is decreased by one. The node is executed when counter = 0. At this occasion counter := period. Therefore:

<integer> < 0 [> 0] hastens [delays] the next time instant of execution by <integer> ticks.

Command sequence

Case 1 and case 2 refer to the cases of the section "The problem". Underlined strings are generated by the program. String that are not underlined are generated by the operator.

```
>comp2
comp2>OPEN IN-node
comp2>SHOW

        .
        .
PERIOD_________10
        .
INTIME_______2370
OUTTIME______2374
LASTTIME_____34
        .
```

SYNC

A delay of 4 ticks is unacceptable. Now case 1 and case 2 differ.

Case_1

We must change the counter of the PID-node that fetches the value.

```
comp2>OPEN PID-node
comp2>SYNC -3
comp2>LINK
```

Now we are ready to inspect the result:

```
comp2>OPEN IN-node
comp2>SHOW
        .
        .
INTIME________3010
OUTTIME_______3011
LASTIIME_______34
```

OK!

Case_2

The IN-node itself sends the value to a DA-converter (or another computer). We must change the counter of the IN-node.

```
comp2>SYNC -3
comp2>LINK
```

We are ready to inspect the result. Do not forget to OPEN again in order to get a fresh version of the IN-node of the active list.

```
comp2>OPEN IN-node
comp2>SHOW
        .
        .
INTIME_______3010
OUTTIME______3011
LASTTTIME_____34
```

OK!

SUGGESTIONS FOR IMPROVEMENTS

1) Every time a message is sent, the message must be preceded by the name of the destination computer. If computer names were shortened from ten letters to one letter, there would be less overhead time in sending messages.

2) There is a trade-off between sending one character of text or data at a time and sending a fixed string length every time. At present, only one character of text or data is sent at a time. An implementation which would send a variable string length would be preferable.

3) There are several places in the program where error routines have not been implemented.

4) There is no directory of node names. Once nodes have been created, their names must be remembered.

5) There is no way to check the structure of the communication link (where computers are relative to each other) after the structure has been initialized.

6) It would be desirable to start up the DDC programs in all the computers from the MAIN computer, rather than from a local terminal which is then disabled.

7) The hardware buffer used to send messages between computers might have to be enlarged.

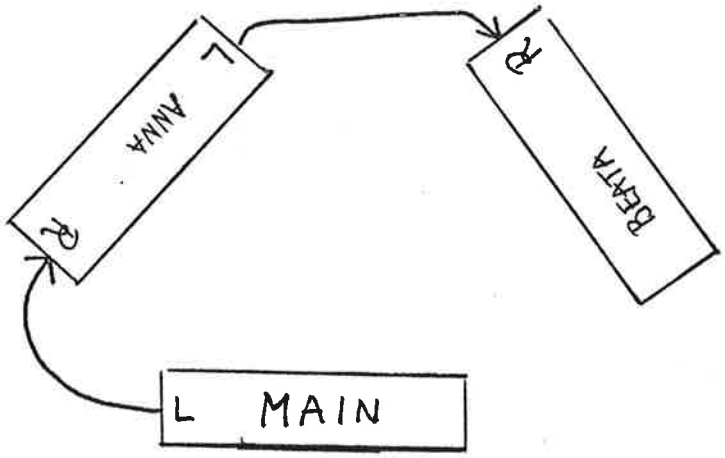
8) There exists no synchronization facility between nodes in the same computer (cf. the PDP-15 version of the DDC-package).

HOW TO START UP

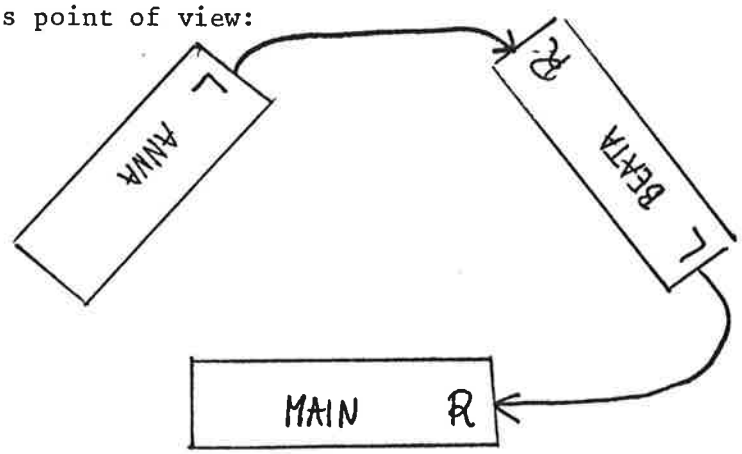
We assume that you want to create the following computer network:

The computer on which the terminal will be attached when you finally run the DDC-package must be called MAIN.

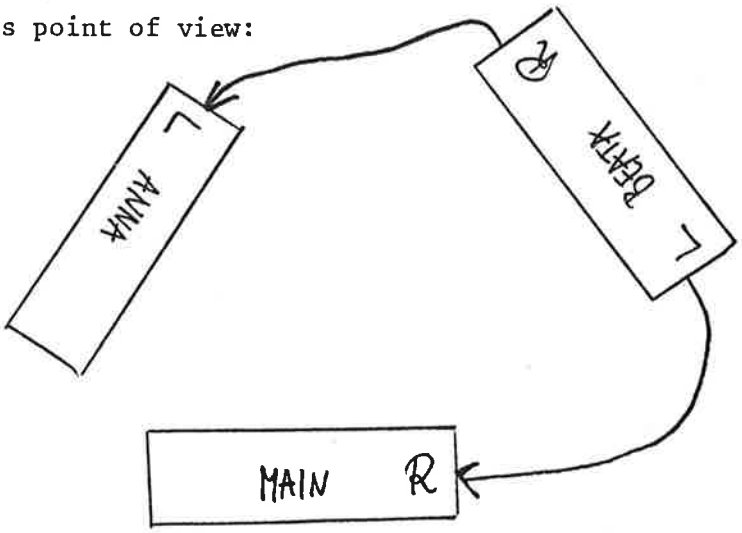
The network from MAIN´s point of view:



From ANNA´s point of view:



From BEATA´s point of view:



L and R stand for Left port and Right port, respectively. The arrows should be interpreted as follows: Messages from MAIN to both ANNA and BEATA will be shipped via the Left port of MAIN. Messages from BEATA to MAIN are sent through the Left port of BEATA, messages from BEATA to ANNA are sent through the Right port of BEATA.

When constructing this communication diagram, make sure that you don't create a "circular mess", i.e. if you want the messages from ANNA to MAIN to go via BEATA, you cannot let the messages from BEATA to MAIN go via ANNA.

The discette must contain the following files:

KERNEL.SAV
KERN3.SAV
OPCOM3.REL
REGUL3.REL
DDCPAA.EMU

Start up of ANNA:

DC OFF, POWER ON, DC ON
SET USR NOSWAP
Load the discette
Press ALPHA LOCK
RUN DX1:KERNEL
DX1:DDCPAA

SYSTEM READY
NAME OF LEFT COMPUTER 1: BEATA (if you want no left computer, press RETURN twice)
NAME OF LEFT COMPUTER 2: MAIN
NAME OF LEFT COMPUTER 3: Press RETURN
NAME OF RIGHT COMPUTER 1: Press RETURN
NAME OF THIS COMPUTER: ANNA

Remove the discette

Start up of BEATA:

Corresponding to the start up of ANNA.

Start up of MAIN:

See start up of ANNA for the first four points. Then

RUN DX1:KERN3
DX1:DDCPAA

SYSTEM READY
NAME OF LEFT COMPUTER 1: ANNA
:
:
NAME OF THIS COMPUTER: MAIN
MAIN >

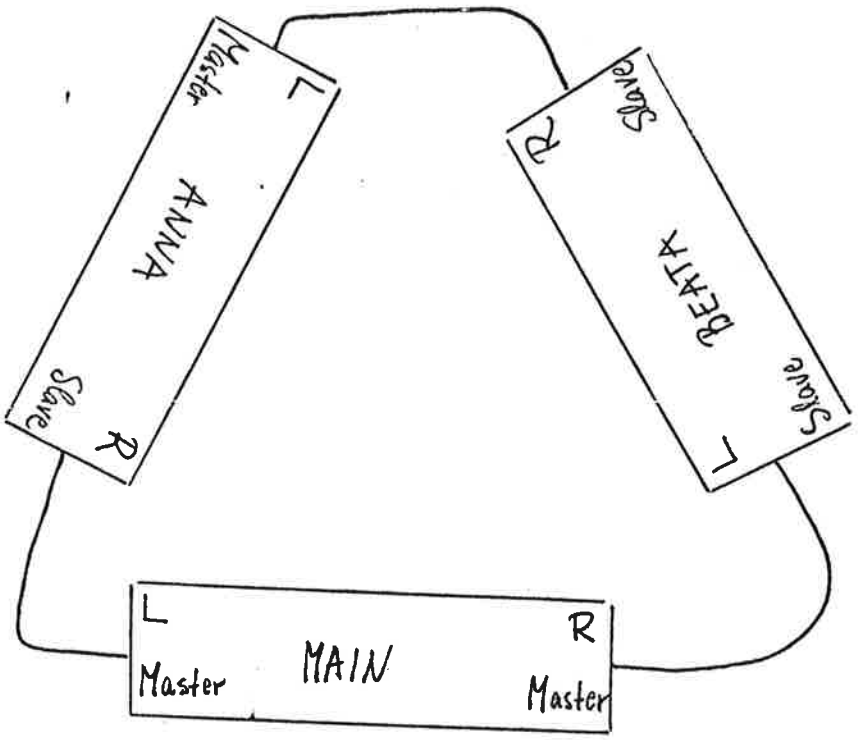
(the underlined strings are the responces of the computer)

Now the computers are ready, but we must first connect them:

The terminal port of MAIN is connected to the terminal in the regular fashion.

All other ports: Set the baud rate = 4800 (you might have to lower this figure later if the transmission load becomes too heavy). Set parity= none.

Every connector must have a MASTER port at one end and a SLAVE port at the other. The following set up is possible, for instance:



Command sequence:

```
MAIN>   OPEN PID7          "Node manipulation commands in MIAN
MAIN>   ...
  :
MAIN>   BYE                "Leave MAIN
>       BEATA              "Choose BEATA
BEATA>   ...               "Node manipulation in BEATA
BEATA>   ...
  :
BEATA>  BYE                "Leave BEATA
>
...etc
```

Close down: DCOFF

REMEMBER: Keep track MANUALLY of the active list and theconnections between the nodes.

```
 1  const maxcomputers=4;
 2  const namelength=10;
 3  type iodevice=(sl0,sl1,sl2);
 4  type ioparam=(input,output);
 5  type nametype=array[1..10] of char;
 6  type answertype=(right,left,myself,nonexistent);
 7  type nodetype=(innode,PIDnode,outnode);
 8  type rstring=array[1..4] of char;
 9
10  {*****************
11  * namemonitortype*
12  *****************}
13
14  type namemonitortype = monitor;
15  type neighbours = array[1..maxcomputers] of nametype;
16  type superqueue = array[1..3] of queue;
17
18  var myname:nametype;
19      leftcomps:neighbours;
20      rightcomps:neighbours;
21      ready:boolean;
22      place:integer;
23      mynamequeue:superqueue;
24
25  procedure entry setmyname(var thisname:nametype);
26  begin
27    myname:=thisname;
28    ready:=true;
29    continue(mynamequeue[1])
30  end;
31
32  procedure entry getmyname(var thisname:nametype);
33  begin
34    if not ready then begin
35                        place:=place+1;
36                        delay(mynamequeue[place]);
37                      end;
38    thisname:=myname;
39    if not empty(mynamequeue[2]) then continue(mynamequeue[2]);
40    continue(mynamequeue[3]);
41  end;
42
43  procedure entry getmyname1(var thisname:nametype);
44  begin
45    thisname:=myname
46  end;
47
48  procedure entry setleftnames(var thisname:nametype;
49                               var i:integer);
50  begin
51    leftcomps[i]:=thisname
52  end;
```

```
53
54  procedure entry setrightnames(var thisname:nametype;
55                                var i:integer);
56  begin
57    rightcomps[i]:=thisname
58  end;
59
60  procedure entry checkcomp(var thisname:nametype;
61                            var answer:answertype);
62  var i:integer;
63      done:boolean;
64  begin
65    answer:=nonexistent;
66    done:=false;
67    i:=1;
68    if (thisname=myname) then answer:=myself
69    else repeat
70      if (thisname=leftcomps[i]) then begin
71                                        answer:=left;
72                                        done:=true
73                                      end
74        else if (thisname=rightcomps[i]) then begin
75                                              answer:=right;
76                                              done:=true;
77                                            end;
78        i:=i+1;
79    until done or (i)maxcomputers)
80  end; {checkcomp}
81
82
83  begin{namemonitortype}
84    myname:='MAIN      ';
85    ready:=false;
86    place:=0
87  end; {namemonitortype}
88
89  {**************
90  * buffmontype *
91  **************}
92
93  type buffmontype=monitor;
94  const bufflength=200;
95  var xchar      :array[1..bufflength] of char;
96      inpos      :integer;
97      outpos     :integer;
98      buffqueue  :queue;
99  procedure entry write(var ch:char);
100 begin
101   inpos:=inpos+1;
102   xchar[inpos]:=ch;
103   if inpos=bufflength then inpos:=0;
104   continue(buffqueue)
105 end;{write}
```

```
106   procedure entry read(var ch:char);
107   begin
108     if inpos=outpos then delay(buffqueue);
109     outpos:=outpos+1;
110     ch:=xchar[outpos];
111     if outpos=bufflength then outpos:=0;
112   end;{read}
113   begin{buffmontype}
114     inpos:=0;
115     outpos:=0;
116   end;{buffmontype}
117
118
119   {*****************
120   * outleftmontype *
121   *****************}
122
123   type outleftmontype=monitor;
124   var no:integer;
125       inparam,outparam:ioparam;
126       ASCII,NUMERIC:char;
127       MAIN:array[1..10] of char;
t↑@8      xleft:iodevice;
129       ready:boolean;
130       leftqueue:queue;
131
132   procedure entry setxleft(var dev:iodevice);
133   begin
134     xleft:=dev;
135     ready:=true;
136     continue(leftqueue);
137   end; {setxleft}
138
139   procedure entry getxleft(var dev:iodevice);
140   begin
141     if ready=false then delay(leftqueue);
142     dev:=xleft;
143   end; {getxleft}
144
145   procedure entry dataout(var comp,node:nametype;var outvalue:univ rstring);
146   begin
147     io(NUMERIC,outparam,xleft);
148     for no:=1 to namelength do io(comp[no],outparam,xleft);
149     for no:=1 to namelength do io(node[no],outparam,xleft);
150     for no:=1 to 4 do io(outvalue[no],outparam,xleft);
151   end;{dataout}
152
153   procedure entry asciiout(var comp:nametype;var ch:char);
154   begin
155     io(ASCII,outparam,xleft);
156     for no:=1 to namelength do io(comp[no],outparam,xleft);
157     io(ch,outparam,xleft);
158   end;{asciiout}
```

```
159   begin
160     inparam:=input;
161     outparam:=output;
162     ASCII:='@';
163     NUMERIC:='#';
164     MAIN:='MAIN      ';
165     ready:=false;
166   end;{outlefttype}
167
168
169   {******************
170   * outrightmontype *
171   ******************}
172
173   type outrightmontype=monitor;
174   var no:integer;
175       inparam,outparam:ioparam;
176       ASCII,NUMERIC:char;
177       MAIN:array[1..10] of char;
178       xright:iodevice;
179       ready:boolean;
180       rightqueue:queue;
181
182   procedure entry setxright(var dev:iodevice);
183   begin
184     xright:=dev;
185     ready:=true;
186     continue(rightqueue);
187   end; {setxright}
188
189   procedure entry getxright(var dev:iodevice);
190   begin
191     if ready=false then delay(rightqueue);
192     dev:=xright;
193   end; {getxright}
194
195   procedure entry dataout(var comp,node:nametype;var outvalue:univ rstring);
196   begin
197     io(NUMERIC,outparam,xright);
198     for no:=1 to namelength do io(comp[no],outparam,xright);
199     for no:=1 to namelength do io(node[no],outparam,xright);
200     for no:=1 to 4 do io(outvalue[no],outparam,xright);
201   end;{dataout}
202
203   procedure entry asciiout(var comp:nametype;var ch:char);
204   begin
205     io(ASCII,outparam,xright);
206     for no:=1 to namelength do io(comp[no],outparam,xright);
207     io(ch,outparam,xright);
208   end;{asciiout}
209   begin
210     inparam:=input;
211     outparam:=output;
```

```
212     ASCII:='@';
213     NUMERIC:='#';
214     MAIN:='MAIN      ';
215     ready:=false;
216   end;{outrighttype}
217
218
219   {**********************
220   * readterminalmontype *
221   **********************}
222
223   type readterminalmontype = monitor;
224   const LF='(:10:)'; del='(:127:)';CR='(:13:)';
225         linelength=133; linelength1=134; {linelength+1}
226   var space,backspace,linefeed:char;
227       inparam,outparam:ioparam;
228       line:array[1..linelength1] of char;
229       pos:0..linelength1;
230       tty:iodevice;
231
232   procedure entry read(var character:char);
233   var ch:char;
234   begin
235     if pos=0 then begin {read a new line}
236       repeat
237          io(ch,inparam,tty);
238          if ch()del then begin
239           io(ch,outparam,tty);
240           if ch)='a'then if ch(='z' then ch:=chr(ord(ch)-32);
241           if pos(linelength then pos:=pos+1;
242           line[pos]:=ch end
243         else begin
244           if pos()0 then begin
245             io(backspace,outparam,tty);
246             io(space,outparam,tty);
247             io(backspace,outparam,tty);
248              pos:=pos-1
249           end
250         end
251       until (ch=CR) or (pos=linelength);
252       if ch=CR then begin
253          line[pos+1]:=LF;
254          io(linefeed,outparam,tty);
255       end;
256       pos:=0
257     end;
258     pos:=pos+1;
259     character:=line[pos];
260     if character=LF then pos:=0
261   end;{read}
262
263   begin {readterminalmontype}
264     tty:=sl0;
```

```
265     inparam:=input;
266     outparam:=output;
267     space:=' ';
268     backspace:=chr(8);
269     linefeed:=LF;
270     pos:=0;
271   end;{readterminalmontype}
272
273
274   {**********************
275   * writeterminalmontype *
276   **********************}
277
278   type writeterminalmontype = monitor;
279   var tty       :iodevice;
280       outparam  :ioparam;
281
282   procedure entry write(var ch:char);
283   begin
284     io(ch,outparam,tty)
285   end;{write}
286   begin
287     tty:=sl0;
288     outparam:=output
289   end;{writeterminalmontype}
290
291
292   {******************
293   * textwritemontype *
294   ******************}
295
296   type textwritemontype = monitor(namemon:namemonitortype;
297                                   outleft:outleftmontype;
298                                   outright:outrightmontype);
299
300   procedure entry textwrite(var dest:nametype;var ch:char);
301   var answer  :answertype;
302   begin
303     namemon.checkcomp(dest,answer);
304     case answer of
305       left:  outleft.asciiout(dest,ch);
306       right: outright.asciiout(dest,ch)
307     end;{case}
308   end;
309   begin
310   end;{textwritemontype}
311
312   type filespectype=array[1..14] of char;
313        seqprogspectype=record
314          filename:filespectype;
315          stacktop:integer;
316          heaptop:integer;
317          startaddress:integer;
```

```
318           loadaddress:integer
319         end;
320         jobprocesstype=(regulprocess,opcomprocess);
321
322   {******************
323   * seqprogramstype *
324   ******************}
325
326   type seqprogramstype=monitor;
327   const regulfilename='DX1:REGUL3.REL';
328         regulspace=1000;
329         opcomfilename='DX1:OPCOM3.REL';
330         opcomspace=1000;
331   var regulfile,opcomfile:seqprogspectype;
332
333   procedure entry seqparms(var seqprogspec:seqprogspectype;
334                            jobprocess:jobprocesstype);
335   begin
336     case jobprocess of
337       regulprocess: seqprogspec:=regulfile;
338       opcomprocess: seqprogspec:=opcomfile
339     end {case}
340   end;{seqparms}
341
342   begin
343     regulfile.filename:=regulfilename;
344     regulfile.loadaddress:=0;
345     Loadseq(regulfile,regulspace);
346     opcomfile.filename:=opcomfilename;
347     opcomfile.loadaddress:=0;
348     Loadseq(opcomfile,opcomspace)
349   end; {seqprogramstype}
350
351   type address=record
352          name:nametype;
353          number:integer
354        end;{address}
355        paramtype= record
356          goalcomp:nametype;
357          intime,outtime,lasttime:integer;
358          tag:nodetype;
359          insig1,insig2,insig3,insig4,output:address;
360          scal1,scal2,scal3,scal4,level:real
361        end;{paramtype}
362        statetype=record
363          filterstate,yold,ipart,dpart:real
364        end;{statetype}
365        ddcnodetype=record
366          fwd,bwd:integer;
367          name:nametype;
368          priority:integer;
369          period,counter:integer;
370          outvalue:real;
```

```
371           parampart:paramtype;
372           state:statetype
373         end;{ddcnodetype}
374
375   {*****************
376   * nodemonitortype *
377   *****************}
378
379   const nodelistsize=20;
380   type nodemonitortype=monitor(namemon:namemonitortype;outleft:outleftmontype;
381                                outright:outrightmontype);
382   var nodelist:array[1..nodelistsize] of ddcnodetype;
383       i,regindex:integer;
384       opqueue:queue;
385
386   procedure entry regputgetnode(var node:ddcnodetype;
387                                 var anyfound:boolean);
388   var done:boolean;
389   begin
390     if regindex<>1 then begin
391       with nodelist[regindex] do begin
392         outvalue:=node.outvalue;
393         state:=node.state;
394         parampart.outtime:=node.parampart.outtime;
395         parampart.lasttime:=node.parampart.lasttime
396       end {with}
397     end;
398     done:=false; anyfound:=false;
399     repeat
400       regindex:=nodelist[regindex].fwd;
401       if regindex=1 then done:=true
402       else begin
403         with nodelist[regindex] do begin
404           if period>0 then begin
405             counter:=counter-1;
406             if counter<=0 then begin
407               counter:=period;
408               anyfound:=true;
409               node:=nodelist[regindex];
410               done:=true
411             end
412           end
413         end {with}
414       end
415     until done;
416     continue(opqueue)
417   end;{regputgetnode}
418
419   procedure entry getoutvalue(var number:integer;
420                               var outval:real);
421   var per:integer;
422   begin
423     per:=nodelist[number].period;
```

```
424     with nodelist[number].parampart do
425     begin
426       outtime:=realtime;
427       if (outtime - intime))=(per div 2) then
428         lasttime:=outtime
429     end; {parampart}
430     outval:=nodelist[number].outvalue
431   end;{getoutvalue}
432
433
434   procedure lookup(var name:nametype; var ptr:integer);
435   begin
436     nodelist[1].name:=name;
437     ptr:=1;
438     repeat
439       ptr:=nodelist[ptr].fwd
440     until nodelist[ptr].name=name
441   end;{lookup}
442
443   procedure entry getrealtime(var rtime:integer);
444   begin
445     rtime:=realtime
446   end; {getrealtime}
447
448   procedure entry putoutvalue(var node:nametype;var value:univ real);
449   var ptr :integer;
450   begin
451     lookup(node,ptr);
452     if ptr=1 then {error(nonode)} else
453        if nodelist[ptr].parampart.tag()innode then {error(noinnode)} else
454          begin
455          nodelist[ptr].outvalue:=value;
456          nodelist[ptr].parampart.intime:=realtime
457          end;
458   end;{putoutvalue}
459
460   procedure entry opgetnode(var node:ddcnodetype;
461                             var notfound,notspace:boolean);
462   var ptr:integer;
463   begin
464     lookup(node.name,ptr);
465     if ptr()1 then begin
466       notfound:=false;
467       node:=nodelist[ptr] end
468     else begin
469       notfound:=true;
470       notspace:=nodelist[2].fwd=2
471     end
472   end;{opgetnode}
473
474   procedure entry delete(var nodename:nametype;var result:integer);
475   var this,ptr:integer;
476   begin
```

```
477     result:=0;
478     lookup(nodename,this);
479     if this=1 then result:=1
480     else begin
481       ptr:=nodelist[this].fwd;
482       while (ptr<>1) and (result<>2) do begin
483         with nodelist[ptr].parampart do begin
484           if insig1.name=nodename then result:=2;
485           if tag<> outnode then begin
486             if insig2.name=nodename then result:=2;
487             if tag=outnode then begin
488               if insig3.name=nodename then result:=2;
489               if insig4.name=nodename then result:=2;
490             end
491           end
492         end;{with}
493         ptr:=nodelist[ptr].fwd
494       end{while}
495     end;
496     if result=0 then begin
497       if this=regindex then delay(opqueue);
498       with nodelist[this] do begin
499         nodelist[fwd].bwd:=bwd;
500         nodelist[bwd].fwd:=fwd;
501         bwd:=2;
502         fwd:=nodelist[2].fwd;
503         nodelist[2].fwd:=this;
504         nodelist[fwd].bwd:=this;
505       end{with}
506     end
507   end;{delete}
508
509   procedure entry link(var node:ddcnodetype);
510   var this,ptr:integer;
511   begin
512     lookup(node.name,this);
513     if this<>1 then begin
514       with nodelist[this] do begin
515         period:=node.period;
516         counter:=counter+node.counter;
517         parampart:=node.parampart
518       end {with} end
519     else begin {a new node}
520       this:=nodelist[2].fwd;
521       ptr:=nodelist[this].fwd;
522       nodelist[ptr].bwd:=2;
523       nodelist[2].fwd:=ptr;
524       nodelist[this]:=node;
525       ptr:=1;
526       with nodelist[this] do begin
527         repeat
528           ptr:=nodelist[ptr].fwd
529         until priority<nodelist[ptr].priority;
```

```
530         fwd:=ptr;
531         bwd:=nodelist[ptr].bwd;
532         nodelist[ptr].bwd:=this;
533         nodelist[bwd].fwd:=this
534       end {with}
535     end
536   end;{link}
537
538   procedure entry checkname(var name:nametype;
539                             var ptr,pri:integer);
540   begin
541     lookup(name,ptr);
542     if ptr<>1 then pri:=nodelist[ptr].priority
543   end;{checkname}
544
545   procedure entry datawrite(var comp,node:nametype;var outvalue:real);
546   var ptr       :integer;
547       answer    :answertype;
548   begin
549     namemon.checkcomp(comp,answer);
550     case answer of
551       left:        outleft.dataout(comp,node,outvalue);
552       right:       outright.dataout(comp,node,outvalue);
553       myself:      {error(longway)};
554       nonexistent: {error(nocomp)}
555     end;{case}
556   end;{datawrite}
557
558   begin {nodemonitortype}
559     regindex:=1;
560     with nodelist[1] do begin
561       fwd:=1; bwd:=1;
562       priority:=32767;
563     end;{with}
564     for i:=2 to nodelistsize do begin
565       with nodelist[i] do begin
566         fwd:=i+1; bwd:=i-1
567       end {with}
568     end;{for}
569     nodelist[2].bwd:=nodelistsize;
570     nodelist[nodelistsize].fwd:=2
571   end;{nodemonitortype}
572
573   {************
574   * timetable *
575   ************}
576
577   type timetabletype=monitor;
578   var lags:integer;
579       regqueue:queue;
580   procedure entry wait;
581   begin
582     if lags=0 then delay(regqueue);
```

```
583     lags:=lags-1
584   end;{wait}
585
586   procedure entry schedule;
587   begin
588     lags:=lags+1;
589     if lags>10 then begin
590       lags:=10;
591       {alarm}
592     end;
593     continue(regqueue)
594   end;{schedule}
595
596   begin
597     lags:=0
598   end;{timetabletype}
599
600
601   {*************
602   * inlefttype *
603   *************}
604
605   inlefttype=process(nodemon:nodemonitortype;namemon:namemonitortype;
606                      buffmon:buffmontype;outright:outrightmontype;
607                      outleft:outleftmontype;
608                      writeterminalmon:writeterminalmontype);
609   const priority=1;
610   var xfertype,ch                          :char;
611       answer                               :answertype;
612       no,nr                                :integer;
613       value                                :array[1..4] of char;
614       comp,myname,node,MAIN                :nametype;
615       inparam,outparam                     :ioparam;
616       xleft                                :iodevice;
617       error                                :array[1..2] of nametype;
618
619   begin
620     setpri(priority);
621     inparam:=input;
622     outparam:=output;
623     outleft.getxleft(xleft);
624     namemon.getmyname(myname);
625     MAIN:='MAIN      ';
626     error[1]:=myname;
627     error[2]:='inL err   ';
628     cycle
629       io(xfertype,inparam,xleft);
630       while (xfertype()'@') and (xfertype()'#') do io(xfertype,inparam,xleft);
631       for no:=1 to namelength do io(comp[no],inparam,xleft);
632       if xfertype='@' then io(ch,inparam,xleft)
633       else begin
634         if xfertype='#' then
635           begin
```

```
636             for no:=1 to namelength do io(node[no],inparam,xleft);
637             for no:=1 to 4 do io(value[no],inparam,xleft);
638           end
639         else
640           begin
641             namemon.checkcomp(MAIN,answer);
642             case answer of
643               left:         for no:=1 to 2 do
644                             for nr:=1 to namelength do
645                             outleft.asciiout(MAIN,error[no,nr]);
646               right:        for no:=1 to 2 do
647                             for nr:=1 to namelength do
648                             outright.asciiout(MAIN,error[no,nr]);
649               myself:       for no:=1 to 2 do
650                             for nr:=1 to namelength do
651                             writeterminalmon.write(error[no,nr])
652             end; {case}
653           end
654         end;
655       namemon.checkcomp(comp,answer);
656       case answer of
657        left:        {error(circmess)};
658        nonexistent: {error(nocomp)};
659        right:       if xfertype='@' then
660                        outright.asciiout(comp,ch) else
661                        outright.dataout(comp,node,value);
662        myself:      if xfertype='@' then
663                        if myname='MAIN      ' then writeterminalmon.write(ch)
664                        else buffmon.write(ch)
665                     else nodemon.putoutvalue(node,value)
666       end;{case}
667     end;{cycle}
668   end;{inlefttype}
669
670   {**************
671   * inrighttype *
672   **************}
673
674   inrighttype=process(nodemon:nodemonitortype;namemon:namemonitortype;
675                       buffmon:buffmontype;outleft:outleftmontype;
676                       outright:outrightmontype;
677                       writeterminalmon:writeterminalmontype);
678   const priority=1;
679   var xfertype,ch                          :char;
680       answer                               :answertype;
681       no,nr                                :integer;
682       value                                :array[1..4] of char;
683       comp,myname,node,MAIN                :nametype;
684       inparam,outparam                     :ioparam;
685       xright                               :iodevice;
686       error                                :array[1..2] of nametype;
687
688   begin
```

```
689     setpri(priority);
690     inparam:=input;
691     outparam:=output;
692     outright.getxright(xright);
693     namemon.getmyname(myname);
694     MAIN:='MAIN      ';
695     error[1]:=myname;
696     error[2]:='inR err   ';
697     cycle
698       io(xfertype,inparam,xright);
699       while (xfertype()'@') and (xfertype()'#') do io(xfertype,inparam,xright);
700       for no:=1 to namelength do io(comp[no],inparam,xright);
701       if xfertype='@' then io(ch,inparam,xright)
702       else begin
703         if xfertype='#' then
704           begin
705             for no:=1 to namelength do io(node[no],inparam,xright);
706             for no:=1 to 4 do io(value[no],inparam,xright);
707           end
708         else
709           begin
710             namemon.checkcomp(MAIN,answer);
711             case answer of
712               left:      for no:=1 to 2 do
713                          for nr:=1 to namelength do
714                          outleft.asciiout(MAIN,error[no,nr]);
715               right:     for no:=1 to 2 do
716                          for nr:=1 to namelength do
717                          outright.asciiout(MAIN,error[no,nr]);
718               myself:    for no:=1 to 2 do
719                          for nr:=1 to namelength do
720                          writeterminalmon.write(error[no,nr])
721             end; {case}
722           end
723         end;
724       namemon.checkcomp(comp,answer);
725       case answer of
726        right:       {error(circmess)};
727        nonexistent: {error(nocomp)};
728        left:        if xfertype='@' then
729                        outleft.asciiout(comp,ch) else
730                        outleft.dataout(comp,node,value);
731        myself:      if xfertype='@' then
732                        if myname='MAIN      ' then writeterminalmon.write(ch)
733                        else buffmon.write(ch)
734                     else nodemon.putoutvalue(node,value)
735       end;{case}
736     end;{cycle}
737   end;{inrighttype}
738
739   {************
740   * clocktype *
741   ************}
```

```
742
743  type clocktype=process(timetable:timetabletype);
744  const priority=1;
745        maxtime=32767;
746  var nexttime:integer;
747  begin
748    setpri(priority);
749    nexttime:=realtime;
750    cycle
751      timetable.schedule;
752      if nexttime=maxtime then nexttime:=0
753      else nexttime:=nexttime+1;
754      sleep(nexttime)
755    end {cycle}
756  end;{clocktype}
757
758  {****************
759  * regulatortype *
760  ****************}
761
762  regulatortype=process(buffmon:buffmontype;
763                        readterminalmon:readterminalmontype;
764                        writeterminalmon:writeterminalmontype;
765                        seqprograms:seqprogramstype;
766                        nodemon:nodemonitortype;
767                        timetable:timetabletype;
768                        namemon:namemonitortype;
769                        outleft:outleftmontype;
770                        outright:outrightmontype);
771
772  const priority=2;
773  var regulfile:seqprogspectype;
774      MAIN     :nametype;
775  program regulator(regulfile:seqprogspectype);
776  entry read,write,wait,
777        regputgetnode,getoutvalue,getrealtime,datawrite,
778        getmyname;
779
780  procedure entry read(var character:char);
781  var name:nametype;
782  begin
783    namemon.getmyname(name);
784    if name=MAIN then readterminalmon.read(character)
785    else buffmon.read(character)
786  end;
787  procedure entry write(var character:char);
788  var answer  :answertype;
789      name    :nametype;
790  begin
791    namemon.checkcomp(MAIN,answer);
792    case answer of
793      myself: writeterminalmon.write(character);
794      left:   outleft.asciiout(MAIN,character);
```

```
795       right:  outright.asciiout(MAIN,character)
796     end{case}
797   end;{write}
798   procedure entry wait;
799   begin
800     timetable.wait
801   end;
802   procedure entry regputgetnode(var node:ddcnodetype;
803                                 var anyfound:boolean);
804   begin
805     nodemon.regputgetnode(node,anyfound)
806   end;
807   procedure entry getoutvalue(var number:integer;
808                               var outvalue:real);
809   begin
810     nodemon.getoutvalue(number,outvalue)
811   end;
812
813   procedure entry getrealtime(var rtime:integer);
814   begin
815     nodemon.getrealtime(rtime)
816   end;
817
818   procedure entry datawrite(var comp,nodename:nametype;
819                             var sendvalue:real);
820   var answer   :answertype;
821   begin
822     namemon.checkcomp(comp,answer);
823     case answer of
824      left:        outleft.dataout(comp,nodename,sendvalue);
825      right:       outright.dataout(comp,nodename,sendvalue);
826      myself:      {error(longway)};
827      nonexistent: {error(nocomp)}
828     end;{case}
829   end;
830
831   procedure entry getmyname(var thisname:nametype);
832   begin
833     namemon.getmyname(thisname)
834   end;
835
836
837   begin
838     setpri(priority);
839     MAIN:='MAIN      ';
840     seqprograms.seqparms(regulfile,regulprocess);
841     regulator(regulfile)
842   end;{regulatortype}
843
844   {************
845   * opcomtype *
846   ************}
847
```

```
848  opcomtype=process(buffmon:buffmontype;
849                    readterminalmon:readterminalmontype;
850                    writeterminalmon:writeterminalmontype;
851                    textwritemon:textwritemontype;
852                    seqprograms:seqprogramstype;
853                    nodemon:nodemonitortype;
854                    namemon:namemonitortype;
855                    outleft:outleftmontype;
856                    outright:outrightmontype);
857  const priority=3; LF='(:10:)';CR='(:13:)';
858  var opcomfile:seqprogspectype;
859      MAIN     :nametype;
860  program opcom(opcomfile:seqprogspectype);
861  entry read,write,opgetnode,delete,
862        link,checkname,setmyname,
863        setleftnames,setrightnames,
864        checkcomp,textwrite,setxleft,setxright;
865  procedure entry read(var character:char);
866  var name  :nametype;
867      answer:answertype;
868      ch:char;
869  begin
870    namemon.getmyname1(name);
871    if name=MAIN then readterminalmon.read(character)
872    else
873     begin
874      buffmon.read(character);
875      namemon.checkcomp(MAIN,answer);
876      case answer of
877       left:       begin
878                     outleft.asciiout(MAIN,character);
879                     if character=LF then begin
880                                            ch:=CR;
881                                            outleft.asciiout(MAIN,ch)
882                                          end
883                   end;
884
885       right:      begin
886                     outright.asciiout(MAIN,character);
887                     if character=LF then begin
888                                            ch:=CR;
889                                            outright.asciiout(MAIN,ch)
890                                          end
891                   end
892      end; {case}
893     end;
894  end;{read}
895  procedure entry write(var character:char);
896  var answer  :answertype;
897      name    :nametype;
898  begin
899    namemon.checkcomp(MAIN,answer);
900    case answer of
```

```
901       myself: writeterminalmon.write(character);
902       left:   outleft.asciiout(MAIN,character);
903       right:  outright.asciiout(MAIN,character)
904     end{case}
905   end;{write}
906   procedure entry opgetnode(var node:ddcnodetype;
907                             var notfound,notspace:boolean);
908   begin
909     nodemon.opgetnode(node,notfound,notspace)
910   end;
911   procedure entry link(var node:ddcnodetype);
912   begin
913     nodemon.link(node)
914   end;
915   procedure entry delete(var nodename:nametype;
916                          var result:integer);
917   begin
918     nodemon.delete(nodename,result)
919   end;
920   procedure entry checkname(var name:nametype;
921                             var ptr,pri:integer);
922   begin
923     nodemon.checkname(name,ptr,pri)
924   end;
925   procedure entry textwrite(var dest:nametype;
926                             var ch:char);
927   begin
928     textwritemon.textwrite(dest,ch);
929   end;
930   procedure entry checkcomp(var thisname:nametype;
931                             var answer:answertype);
932   begin
933     namemon.checkcomp(thisname,answer);
934   end;
935   procedure entry setmyname(var myname:nametype);
936   begin
937     namemon.setmyname(myname);
938   end;
939   procedure entry setleftnames(var thisname:nametype;
940                                var i:integer);
941   begin
942     namemon.setleftnames(thisname,i);
943   end;
944   procedure entry setrightnames(var thisname:nametype;
945                                 var i:integer);
946   begin
947     namemon.setrightnames(thisname,i);
948   end;
949   procedure entry setxleft(var dev:iodevice);
950   begin
951     outleft.setxleft(dev);
952   end;
953   procedure entry setxright(var dev:iodevice);
```

```
  954   begin
  955     outright.setxright(dev);
  956   end;
  957
  958   begin
  959     setpri(priority);
  960     MAIN:='MAIN      ';
  961     seqprograms.seqparms(opcomfile,opcomprocess);
  962     opcom(opcomfile)
  963   end;{opcomtype}
  964
  965
  966   {*******
  967   * main *
  968   *******}
  969
  970   var readterminalmon:readterminalmontype;
  971       writeterminalmon:writeterminalmontype;
  972       textwritemon:textwritemontype;
  973       seqprograms:seqprogramstype;
  974       nodemon:nodemonitortype;
  975       timetable:timetabletype;
  976       clock:clocktype;
  977       regulator:regulatortype;
  978       opcom:opcomtype;
  979       namemon:namemonitortype;
  980       buffmon:buffmontype;
  981       outleft:outleftmontype;
  982       outright:outrightmontype;
  983       inleft:inlefttype;
  984       inright:inrighttype;
  985
  986   begin
  987     init seqprograms,timetable,buffmon,
  988          outleft,outright,namemon,
  989          readterminalmon,
  990          writeterminalmon,
  991          textwritemon(namemon,outleft,outright),
  992          nodemon(namemon,outleft,outright),
  993          clock(timetable),
  994          regulator(buffmon,readterminalmon,writeterminalmon,seqprograms,
  995                    nodemon,timetable,namemon,outleft,outright),
  996          opcom(buffmon,readterminalmon,writeterminalmon,textwritemon,
  997                seqprograms,nodemon,namemon,outleft,outright),
  998          inleft(nodemon,namemon,buffmon,outright,outleft,writeterminalmon),
  999          inright(nodemon,namemon,buffmon,outleft,outright,writeterminalmon)
 1000
 1001   end.

ERRORS DETECTED:    0
```

License site #1-34; Lund Institute of Technology, Lund, Sweden

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

   1                     program opcom;
   2                     label 999;
   3                     type entryprocedures=(opgetnode1,delete1,link1,checkname1,setmyname1,
   4                                           setleftnames1,setrightnames1,checkcomp1,
   5                                           textwrite1,setxleft1,setxright1);
   6                     procedure entprocedure(entryprocedure:entryprocedures); external;
   7
   8                     const maxcomputers=4;
   9                           tickspersecond=50.0;
  10                           blanks='          ';
  11                           AD='AD        ';
  12                           DA='DA        ';
  13                           MAIN='MAIN      ';
  14                           EXT='EXT       ';
  15                     type nametype=array[1..10] of char;
  16                          address=record
  17                            name:nametype;
  18                            number:integer
  19                          end;{address}
  20                          nodetype=(innode,PIDnode,outnode);
  21                          paramtype=record
  22                            goalcomp:nametype;
  23                            intime,outtime,lasttime:integer;
  24                            case tag:nodetype of
  25                              innode:(insig,outplace:address;
  26                                      scale,filter:real);
  27                              PIDnode:(PIDin,PIDref,PIDout:address;
  28                                       k,ti,td,alfa,beta,limit:real);
  29                              outnode:(insig1,insig2,insig3,
  30                                       insig4,output:address;
  31                                       scal1,scal2,scal3,scal4,level:real)
  32                          end;{paramtype}
  33                          statetype=record
  34                            filterstate,yold,ipart,dpart:real
  35                          end;{statetype}
  36                          ddcnodetype=record
  37                            fwd,bwd:integer;
  38                            name:nametype;
  39                            priority:integer;
  40                            period,counter:integer;
  41                            outvalue:real;
  42                            parampart:paramtype;
  43                            state:statetype
  44                          end;{ddcnodetype}
  45                          commandtype=array[1..9] of char;
  46                          opindex=(openx,showx,linkx,deletex,lastopindex);
  47                          variableindex=(goalcompx,periodx,syncx,insigx,outplacex,scalex,
  48                                         filterx,PIDinx,PIDrefx,PIDoutx,kx,tix,
  49                                         tdx,limitx,insig1x,insig2x,
  50                                         insig3x,insig4x,outputx,scal1x,
  51                                         scal2x,scal3x,scal4x,
  52                                         levelx,lastvarindex);
  53                          errors=(fewarg,toomanyarg,illname,nospace,
  54                                  notopen,blankaddr,nonode,referr,ADDAerr,
```

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

  55                                noname,prierr,noop,tagerr,illcomp,desterr);
  56                          texttype=array[1..132] of char;
  57                          answertype=(right,left,myself,nonexistent);
  58                          iodevice=(sl0,sl1,sl2);
  59
  60                        var  op:array[opindex] of commandtype;
  61                             vars:array[variableindex] of commandtype;
  62                             opx:opindex;
  63                             varindex:variableindex;
  64                             node:ddcnodetype;
  65                             name:nametype;
  66                             command:commandtype;
  67                             opened:boolean;
  68                             myname:nametype;
  69                             actcomp:nametype;
  70                             i,j:integer;
  71                             exist:answertype;
  72                             buff:texttype;
  73                             ch:char;
  74                             LF:char;
  75
  76                        procedure opgetnode(var node:ddcnodetype;
  77                                            var notfound,notspace:boolean);
  78                        begin
  79     1      2      1       entprocedure(opgetnode1)
  80                        end;
  81                        procedure delete2(var name:nametype;
  82                                          var result:integer);
  83                        begin
  84     1      2      1       entprocedure(delete1)
  85                        end;
  86                        procedure link2(var node:ddcnodetype);
  87                        begin
  88     1      2      1       entprocedure(link1)
  89                        end;
  90                        procedure checkname(var name:nametype;
  91                                            var nr,pri:integer);
  92                        begin
  93     1      2      1       entprocedure(checkname1)
  94                        end;
  95                        procedure setmyname(var myname:nametype);
  96                        begin
  97     1      2      1       entprocedure(setmyname1)
  98                        end;
  99                        procedure setleftnames(var thisname:nametype;
 100                                               var i:integer);
 101                        begin
 102     1      2      1       entprocedure(setleftnames1)
 103                        end;
 104                        procedure setrightnames(var thisname:nametype;
 105                                                var i:integer);
 106                        begin
 107     1      2      1       entprocedure(setrightnames1)
 108                        end;
```

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

 109                     procedure checkcomp(var thisname:nametype;
 110                                         var answer:answertype);
 111                     begin
 112     1     2     1     entprocedure(checkcomp1)
 113                     end;
 114                     procedure textwrite(var dest:nametype;
 115                                         var ch:char);
 116                     begin
 117     1     2     1     entprocedure(textwrite1)
 118                     end;
 119                     procedure setxleft(var des:iodevice);
 120                     begin
 121     1     2     1     entprocedure(setxleft1)
 122                     end;
 123                     procedure setxright(var des:iodevice);
 124                     begin
 125     1     2     1     entprocedure(setxright1)
 126                     end;
 127
 128                     procedure error(err:errors);
 129                     begin.
 130     1     2     1     case err of
 131     2     2     2       fewarg:     writeln('missing arguments');
 132     3     2     2       toomanyarg: writeln('too many arguments');
 133     4     2     2       illname:    writeln('illegal name');
 134     5     2     2       nospace:    writeln('the nodelist is full');
 135     6     2     2       notopen:    writeln('no node is open');
 136     7     2     2       blankaddr:  writeln('undefined signal');
 137     8     2     2       nonode:     writeln('the node didn''t exist');
 138     9     2     2       referr:     writeln('other nodes use this node');
 139    10     2     2       ADDAerr:    writeln('please don''t send to AD-converter ',
 140                                               'or read from DA-converter');
 141    11     2     2       noname:     writeln('undefined signal');
 142    12     2     2       prierr:     writeln('the value is not ',
 143                                               'available here');
 144    13     2     2       noop:       writeln('illegal operation');
 145    14     2     2       tagerr:     writeln('wrong nodetype');
 146    15     2     2       illcomp:    writeln('no computer name');
 147    16     2     2       desterr:    writeln('don''t send to DA-converter ',
 148                                               'of another computer,stupid!')
 149                       end;{case}
 150    17     2     1     readln;
 151    18     2     1     write(myname,')');
 152    19     2     1     goto 999
 153                     end;{error}
 154
 155                     procedure writeaddress(addr:address);
 156                     begin
 157     1     2     1     with addr do begin
 158     3     2     3       if name=AD then writeln('AD ',number:2)
 159     5     2     4       else if name=DA then writeln('DA ',number:2)
 160     7     2     5         else writeln(name)
 161                       end {with}
 162                     end;{writeadress}
```

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

 163
 164                     procedure readname;
 165                     begin
 166     1     2     1       if eoln then error(fewarg);
 167     3     2     1       read(name);
 168     4     2     1       if not eoln then error(toomanyarg);
 169     6     2     1       if (name=AD) or (name=DA) or
 170     7     2     2         (name=blanks) then error(illname)
 171                     end;{readname}
 172
 173                     procedure open;
 174                     var notfound,notspace,done:boolean;
 175                         nodestring:array[1..8] of char;
 176                     procedure initname(var addr:address);
 177                     begin
 178     1     3     1       addr.name:=blanks;
 179     2     3     1       addr.number:=0
 180                     end;{initname}
 181                     begin
 182     1     2     1       opened:=false;
 183     2     2     1       readname;
 184     3     2     1       node.name:=name;
 185     4     2     1       opgetnode(node,notfound,notspace);
 186     5     2     1       if notfound then begin
 187     7     2     3         if notspace then error(nospace)
 188     9     2     4         else begin
 189    10     2     5           with node do begin
 190    12     2     7             repeat
 191    13     2     8               write('* priority= ');
 192    14     2     8               readln(priority);
 193    15     2     8             until (priority>0) and (priority<32767);
 194    16     2     7             period:=0;
 195    17     2     7             counter:=1;
 196    18     2     7             outvalue:=0.0
 197                               end;{with}
 198    19     2     5           with node.parampart do begin
 199    21     2     7             repeat
 200    22     2     8               done:=true;
 201    23     2     8               write('* nodetype= ');
 202    24     2     8               read(nodestring);
 203    25     2     8               if nodestring='INNODE  ' then
 204    26     2     9                 tag:=innode
 205    27     2     9               else if nodestring='PIDNODE ' then
 206    28     2    10                 tag:=PIDnode
 207    29     2    10                 else if nodestring='OUTNODE ' then
 208    30     2    11                   tag:=outnode
 209    31     2    11                   else begin done:=false; readln end
 210                                 until done;
 211    34     2     7             goalcomp:=myname;
 212    35     2     7             case tag of
 213    36     2     8               innode: begin
 214    37     2     9                         initname(insig);
 215    38     2     9                         scale:=1.0;
 216    39     2     9                         filter:=1.0;
```

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

 217    40     2     9                        initname(outplace);
 218    41     2     9                        intime:=0;
 219    42     2     9                        outtime:=0;
 220    43     2     9                        lasttime:=0;
 221    44     2     9                      end;
 222    45     2     8              PIDnode:begin
 223    46     2     9                        limit:=1.0;
 224    47     2     9                        k:=0.0;
 225    48     2     9                        ti:=0.0;
 226    49     2     9                        td:=0.0;
 227    50     2     9                        initname(PIDin);
 228    51     2     9                        initname(PIDref);
 229    52     2     9                        initname(PIDout)
 230                                         end;
 231    53     2     8              outnode:begin
 232    54     2     9                        initname(insig1); scal1:=0.0;
 233    56     2     9                        initname(insig2); scal2:=0.0;
 234    58     2     9                        initname(insig3); scal3:=0.0;
 235    60     2     9                        initname(insig4); scal4:=0.0;
 236    62     2     9                        initname(output);
 237    63     2     9                        level:=0.0
 238                                         end
 239                                   end {case}
 240                                 end;{with}
 241    64     2     5                 with node.state do begin
 242    66     2     7                   filterstate:=0.0;
 243    67     2     7                   yold:=0.0;
 244    68     2     7                   ipart:=0.0;
 245    69     2     7                   dpart:=0.0;
 246    70     2     7                 end{with}
 247                                  end
 248                               end;
 249    71     2     1               opened:=true;
 250    72     2     1               node.counter:=0;
 251    73     2     1             end;{open}
 252
 253                               procedure show;
 254                               begin
 255     1     2     1               if not opened then error(notopen);
 256     3     2     1               with node do begin
 257     5     2     3                 write('NODENAME: '); writeln(name);
 258     7     2     3                 write('PERIOD   : '); writeln(period);
 259     9     2     3                 write('PRIORITY: '); writeln(priority);
 260    11     2     3                 write('OUTVALUE: '); writeln(outvalue);
 261    13     2     3               end;{with}
 262    14     2     1               with node.parampart,node.state do begin
 263    16     2     4                 case tag of
 264    17     2     5                   innode: begin
 265    18     2     6                             write('INSIG    : '); writeaddress(insig);
 266    20     2     6                             write('FILTER   : '); writeln(filter);
 267    22     2     6                             write('SCALE    : '); writeln(scale);
 268    24     2     6                             write('FILTERSTATE : '); writeln(filterstate);
 269    26     2     6                             write('GOALCOMP: '); writeln(goalcomp);
 270    28     2     6                             write('OUTPLACE: '); writeaddress(outplace);
```

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

 271    30     2     6                    write('INTIME  : '); writeln(intime);
 272    32     2     6                    write('OUTTIME : '); writeln(outtime);
 273    34     2     6                    write('LASTTIME: '); writeln(lasttime);
 274    36     2     6                  end;
 275    37     2     5          PIDnode:begin
 276    38     2     6                    write('PIDREF  : '); writeaddress(PIDref);
 277    40     2     6                    write('PIDIN   : '); writeaddress(PIDin);
 278    42     2     6                    write('PIDOUT  : '); writeaddress(PIDout);
 279    44     2     6                    write('GOALCOMP: '); writeln(goalcomp);
 280    46     2     6                    write('K       : '); writeln(K);
 281    48     2     6                    write('TI      : '); writeln(TI);
 282    50     2     6                    write('TD      : '); writeln(TD);
 283    52     2     6                    write('LIMIT   : '); writeln(limit);
 284    54     2     6                    write('YOLD    : '); writeln(yold);
 285    56     2     6                    write('IPART   : '); writeln(ipart);
 286    58     2     6                    write('DPART   : '); writeln(dpart);
 287    60     2     6                  end;
 288    61     2     5          outnode:begin
 289    62     2     6                    write('INSIG1  : '); writeaddress(insig1);
 290    64     2     6                    write('SCAL1   : '); writeln(scal1);
 291    66     2     6                    write('INSIG2  : '); writeaddress(insig2);
 292    68     2     6                    write('SCAL2   : '); writeln(scal2);
 293    70     2     6                    write('INSIG3  : '); writeaddress(insig3);
 294    72     2     6                    write('SCAL3   : '); writeln(scal3);
 295    74     2     6                    write('INSIG4  : '); writeaddress(insig4);
 296    76     2     6                    write('SCAL4   : '); writeln(scal4);
 297    78     2     6                    write('OUTPUT  : '); writeaddress(output);
 298    80     2     6                    write('GOALCOMP: '); writeln(goalcomp);
 299    82     2     6                    write('LEVEL   : '); writeln(level);
 300    84     2     6                  end
 301                             end {case}
 302                           end {with}
 303                         end;{show}
 304
 305                         procedure link;
 306                         var ts:real;
 307                         begin
 308     1     2     1           if not opened then error(notopen);
 309     3     2     1           with node.parampart do begin
 310     5     2     3             case tag of
 311     6     2     4               innode: begin
 312     7     2     5                         if (myname()goalcomp) and (outplace.name=DA)
 313     8     2     6                         then error(desterr);
 314     9     2     5                         if insig.name=blanks then error(blankaddr)
 315                                     end;
 316    11     2     4               PIDnode:begin
 317    12     2     5                         if (myname()goalcomp) and (PIDout.name=DA)
 318    13     2     6                         then error(desterr);
 319    14     2     5                         if PIDref.name=blanks then
 320    15     2     6                           error(blankaddr);
 321    16     2     5                         if PIDin.name=blanks then
 322    17     2     6                           error(blankaddr);
 323    18     2     5                         ts:=node.period/tickspersecond;
 324    19     2     5                         if ti)0.0 then alfa:=k*ts/ti
```

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

 325    21     2     6                  else alfa:=0.0;
 326    22     2     5                  if (ts)0.0) and (td)0.0) then
 327    23     2     6                    beta:=k*td/ts
 328    24     2     6                  else beta:=0.0
 329                                    end;
 330    25     2     4          outnode:begin
 331    26     2     5                  if (myname()goalcomp) and (output.name=DA)
 332    27     2     6                  then error(desterr);
 333    28     2     5                  if (scal1()0.0) and
 334                                      (insig1.name=blanks) then
 335    29     2     6                    error(blankaddr);
 336    30     2     5                  if (scal2()0.0) and
 337                                      (insig2.name=blanks) then
 338    31     2     6                    error(blankaddr);
 339    32     2     5                  if (scal3()0.0) and
 340                                      (insig3.name=blanks) then
 341    33     2     6                    error(blankaddr);
 342    34     2     5                  if (scal4()0.0) and
 343                                      (insig4.name=blanks) then
 344    35     2     6                    error(blankaddr)
 345                                    end
 346                              end {case}
 347                            end;{with}
 348    36     2     1            link2(node)
 349                          end;{link}
 350
 351                          procedure delete;
 352                          var result:integer;
 353                          begin
 354     1     2     1            readname;
 355     2     2     1            delete2(name,result);
 356     3     2     1            if result=1 then error(nonode);
 357     5     2     1            if result=2 then error(referr)
 358                          end;{delete}
 359
 360                          procedure readreal(var variable:real;
 361                                             tag:nodetype);
 362                          var r:real;
 363                          begin
 364     1     2     1            if node.parampart.tag()tag then
 365     2     2     2              error(tagerr);
 366     3     2     1            if eoln then error(fewarg);
 367     5     2     1            read(r);
 368     6     2     1            if not eoln then error(toomanyarg);
 369     8     2     1            variable:=r
 370                          end;
 371
 372                          procedure readint(var variable:integer);
 373                          var i:integer;
 374                          begin
 375     1     2     1            if eoln then error(fewarg);
 376     3     2     1            read(i);
 377     4     2     1            if not eoln then error(toomanyarg);
 378     6     2     1            variable:=i;
```

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

 379     7     2     1   end;
 380
 381                     procedure readaddr(var signal:address;
 382                                        tag:nodetype;
 383                                        outsignal:boolean);
 384                     var nr,pri:integer;
 385                     begin
 386     1     2     1       if node.parampart.tag()tag then
 387     2     2     2         error(tagerr);
 388     3     2     1       if eoln then error(fewarg);
 389     5     2     1       read(name);
 390     6     2     1       if name()blanks then begin
 391     8     2     3         if outsignal and (name=AD) then
 392     9     2     4           error(ADDAerr);
 393    10     2     3         if (not outsignal) and (name=DA) then
 394    11     2     4           error(ADDAerr);
 395    12     2     3         if (name=DA) or (name=AD) then begin
 396    14     2     5           if eoln then error(fewarg);
 397    16     2     5           read(nr) end
 398    17     2     4         else if (name()EXT) then begin
 399    19     2     6           if node.parampart.goalcomp=myname then begin
 400    21     2     8             checkname(name,nr,pri);
 401    22     2     8             if nr=1 then error(noname);
 402    24     2     8             if node.priority(=pri then error(prierr) end
 403                             end
 404                           end;
 405    26     2     1       if not eoln then error(toomanyarg);
 406    28     2     1       signal.name:=name;
 407    29     2     1       signal.number:=nr
 408                     end;{readaddr}
 409
 410                     procedure readcomp(var comp:nametype);
 411                     begin
 412     1     2     1       if eoln then error(fewarg);
 413     3     2     1       read(comp);
 414     4     2     1       if not eoln then error(toomanyarg);
 415     6     2     1       checkcomp(comp,exist);
 416     7     2     1       if exist=nonexistent then error(illcomp)
 417                     end;
 418
 419                     procedure setvariable;
 420                     begin
 421     1     2     1       with node.parampart do begin
 422     3     2     3         varindex:=goalcompx;
 423     4     2     3         vars[lastvarindex]:=command;
 424     5     2     3         while vars[varindex]()command do
 425     6     2     4           varindex:=succ(varindex);
 426     7     2     3         if not opened then error(notopen);
 427     9     2     3         case varindex of
 428    10     2     4           goalcompx: readcomp(goalcomp);
 429    11     2     4           periodx:   readint(node.period);
 430    12     2     4           syncx:     readint(node.counter);
 431    13     2     4           insigx:    readaddr(insig,innode,false);
 432    14     2     4           outplacex: readaddr(outplace,innode,true);
```

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

 433    15     2     4          scalex:     readreal(scale,innode);
 434    16     2     4          filterx:    readreal(filter,innode);
 435    17     2     4          PIDinx:     readaddr(PIDin,PIDnode,false);
 436    18     2     4          PIDrefx:    readaddr(PIDref,PIDnode,false);
 437    19     2     4          PIDoutx:    readaddr(PIDout,PIDnode,true);
 438    20     2     4          kx:         readreal(k,PIDnode);
 439    21     2     4          tix:        readreal(ti,PIDnode);
 440    22     2     4          tdx:        readreal(td,PIDnode);
 441    23     2     4          limitx:     readreal(limit,PIDnode);
 442    24     2     4          insig1x:    readaddr(insig1,outnode,false);
 443    25     2     4          insig2x:    readaddr(insig2,outnode,false);
 444    26     2     4          insig3x:    readaddr(insig3,outnode,false);
 445    27     2     4          insig4x:    readaddr(insig4,outnode,false);
 446    28     2     4          outputx:    readaddr(output,outnode,true);
 447    29     2     4          scal1x:     readreal(scal1,outnode);
 448    30     2     4          scal2x:     readreal(scal2,outnode);
 449    31     2     4          scal3x:     readreal(scal3,outnode);
 450    32     2     4          scal4x:     readreal(scal4,outnode);
 451    33     2     4          levelx:     readreal(level,outnode);
 452    34     2     4          lastvarindex: error(noop)
 453                           end {case}
 454                         end {with}
 455                       end;{setvariable}
 456
 457                       procedure initialize;
 458                       var done:boolean;
 459                           i:integer;
 460                           xleft,xright:iodevice;
 461                       begin
 462     1     2     1       LF:=chr(10);
 463     2     2     1       op[openx]:=         'OPEN     ';
 464     3     2     1       op[showx]:=         'SHOW     ';
 465     4     2     1       op[linkx]:=         'LINK     ';
 466     5     2     1       op[deletex]:=       'DELETE   ';
 467     6     2     1       vars[periodx]:=     'PERIOD   ';
 468     7     2     1       vars[syncx]:=       'SYNC     ';
 469     8     2     1       vars[insigx]:=      'INSIG    ';
 470     9     2     1       vars[outplacex]:=   'OUTPLACE ';
 471    10     2     1       vars[scalex]:=      'SCALE    ';
 472    11     2     1       vars[filterx]:=     'FILTER   ';
 473    12     2     1       vars[goalcompx]:=   'GOALCOMP ';
 474    13     2     1       vars[PIDinx]:=      'PIDIN    ';
 475    14     2     1       vars[PIDrefx]:=     'PIDREF   ';
 476    15     2     1       vars[PIDoutx]:=     'PIDOUT   ';
 477    16     2     1       vars[kx]:=          'K        ';
 478    17     2     1       vars[tix]:=         'TI       ';
 479    18     2     1       vars[tdx]:=         'TD       ';
 480    19     2     1       vars[limitx]:=      'LIMIT    ';
 481    20     2     1       vars[insig1x]:=     'INSIG1   ';
 482    21     2     1       vars[insig2x]:=     'INSIG2   ';
 483    22     2     1       vars[insig3x]:=     'INSIG3   ';
 484    23     2     1       vars[insig4x]:=     'INSIG4   ';
 485    24     2     1       vars[outputx]:=     'OUTPUT   ';
 486    25     2     1       vars[scal1x]:=      'SCAL1    ';
```

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

 487    26     2     1     vars[scal2x]:=    'SCAL2    ';
 488    27     2     1     vars[scal3x]:=    'SCAL3    ';
 489    28     2     1     vars[scal4x]:=    'SCAL4    ';
 490    29     2     1     vars[levelx]:=    'LEVEL    ';
 491    30     2     1     opened:=false;
 492    31     2     1     actcomp:=         'MAIN     ';
 493    32     2     1     done:=false;     i:=1;
 494    34     2     1     repeat
 495    35     2     2       write('NAME OF LEFT COMPUTER ',i:2,':');
 496    36     2     2       readln(name);
 497    37     2     2       if name=blanks then done:=true
 498    39     2     3       else setleftnames(name,i);
 499    40     2     2       i:=i+1;
 500    41     2     2     until done or (i)maxcomputers);
 501    42     2     1     done:=false;     i:=1;
 502    44     2     1     repeat
 503    45     2     2       write('NAME OF RIGHT COMPUTER ',i:2,':');
 504    46     2     2       readln(name);
 505    47     2     2       if name=blanks then done:=true
 506    49     2     3       else setrightnames(name,i);
 507    50     2     2       i:=i+1;
 508    51     2     2     until done or (i)maxcomputers);
 509    52     2     1     done:=false;
 510    53     2     1     repeat
 511    54     2     2       write('NAME OF THIS COMPUTER:');
 512    55     2     2       readln(myname);
 513    56     2     2       if myname()blanks then done:=true;
 514    58     2     2     until done;
 515    59     2     1     setmyname(myname);
 516    60     2     1     if myname()MAIN then begin
 517    62     2     3       xleft:=s10;
 518    63     2     3       xright:=s11
 519                       end
 520    64     2     2     else begin
 521    65     2     3       xleft:=s11;
 522    66     2     3       xright:=s12;
 523    67     2     3       write(myname,')')
 524                       end;
 525    68     2     1     setxleft(xleft);
 526    69     2     1     setxright(xright)
 527                     end;{initialize}
 528
 529                     begin {opcom}
 530     1     1     1     initialize;
 531     2     1     1   999:if myname=MAIN then
 532     3     1     2         repeat
 533     4     1     3           i:=0;
 534     5     1     3           command:='          ';
 535     6     1     3           while not eoln do begin
 536     8     1     5             read(ch);
 537     9     1     5             if ch)='a' then if ch(='z' then ch:=chr(ord(ch)-32);
 538    12     1     5             i:=i+1;
 539    13     1     5             command[i]:=ch;
 540    14     1     5             buff[i]:=ch;
```

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

 541    15     1     5          if (ch = ' ') or (i >= 9) then exit
 542                           end;{while}
 543    17     1     3          if command='BYE      ' then
 544    18     1     4            repeat
 545    19     1     5              write(')');
 546    20     1     5              read(actcomp);
 547    21     1     5              checkcomp(actcomp,exist);
 548    22     1     5              if exist=nonexistent then writeln('no computer name')
 549    24     1     6              else write(actcomp,')')
 550                               until exist<>nonexistent
 551    25     1     4          else if actcomp=MAIN then
 552    26     1     5            begin
 553    27     1     6              op[lastopindex]:=command;
 554    28     1     6              opx:=openx;
 555    29     1     6              while op[opx]<>command do
 556    30     1     7                opx:=succ(opx);
 557    31     1     6                case opx of
 558    32     1     7                  openx      :open;
 559    33     1     7                  showx      : show;
 560    34     1     7                  linkx      : link;
 561    35     1     7                  deletex    : delete;
 562    36     1     7                  lastopindex: setvariable
 563                                 end;{case}
 564    37     1     6              write(myname,')')
 565                               end
 566    38     1     5          else begin
 567    39     1     6            while (not eoln) and (i<132) do begin
 568    41     1     8              i:=i+1;
 569    42     1     8              read(buff[i]);
 570    43     1     8            end;{while}
 571    44     1     6            for j:=1 to i do textwrite(actcomp,buff[j]);
 572    46     1     6            textwrite(actcomp,LF)
 573                             end;
 574    47     1     3          readln;
 575    48     1     3        until false
 576    49     1     2        else repeat
 577    50     1     3           read(command);
 578    51     1     3           op[lastopindex]:=command;
 579    52     1     3           opx:=openx;
 580    53     1     3           while op[opx]<>command do
 581    54     1     4             opx:=succ(opx);
 582    55     1     3           case opx of
 583    56     1     4             openx      : open;
 584    57     1     4             showx      : show;
 585    58     1     4             linkx      : link;
 586    59     1     4             deletex    : delete;
 587    60     1     4             lastopindex: setvariable
 588                              end;{case}
 589    61     1     3           write(myname,')')
 590                           until false;
 591    62     1     1   end.{opcom}

ERRORS DETECTED:  0
FREE MEMORY:  7166  WORDS
```

License site #1-34; Lund Institute of Technology, Lund, Sweden

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

   1                     program regulator;
   2
   3                     function adin(chan:integer):real; external;
   4
   5                     procedure daout(chan:integer; value:real);
   6                       external;
   7
   8                     type entryprocedures=(wait1, regputgetnode1,
   9                                           getoutvalue1,getrealtime1,
  10                                           datawrite1,getmyname1);
  11
  12                     procedure entprocedure(entryprocedure:
  13                                            entryprocedures); external;
  14
  15                     const AD='AD        ';
  16                           DA='DA        ';
  17                           EXT='EXT       ';
  18
  19                     type nametype=array[1..10] of char;
  20
  21                          address=record
  22                            name:nametype;
  23                            number:integer
  24                          end;{address}
  25
  26                          nodetype=(innode, PIDnode, outnode);
  27
  28                          paramtype=record
  29                            goalcomp:nametype;
  30                            intime,outtime,lasttime:integer;
  31                            case tag:nodetype of
  32                              innode:(insig,outplace:address;
  33                                      scale,filter:real);
  34                              PIDnode:(PIDin,PIDref,PIDout:address;
  35                                       k,ti,td,alfa,beta,limit:real);
  36                              outnode:(insig1,insig2,insig3,
  37                                       insig4,output:address;
  38                                       scal1,scal2,scal3,scal4,level:real)
  39                            end;{paramtype}
  40
  41                          statetype=record
  42                            filterstate,yold,ipart,dpart:real;
  43                          end;{statetype}
  44
  45                          ddcnodetype=record
  46                            fwd,bwd:integer;
  47                            name:nametype;
  48                            priority:integer;
  49                            period,counter:integer;
  50                            outvalue:real;
  51                            parampart:paramtype;
  52                            state:statetype
  53                          end;{ddcnodetype}
  54
```

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

  55                     var node:ddcnodetype;
  56                         anyfound:boolean;
  57                         myname:nametype;
  58
  59                     procedure wait;
  60                     begin
  61     1     2     1       entprocedure(wait1)
  62                     end;
  63
  64                     procedure regputgetnode(var node:ddcnodetype;
  65                                             var anyfound:boolean);
  66                     begin
  67     1     2     1       entprocedure(regputgetnode1)
  68                     end;
  69
  70                     procedure getoutvalue(var number:integer;
  71                                           var outvalue:real);
  72                     begin
  73     1     2     1       entprocedure(getoutvalue1)
  74                     end;
  75
  76                     procedure getrealtime(var rtime:integer);
  77                     begin
  78     1     2     1       entprocedure(getrealtime1)
  79                     end;
  80
  81                     procedure datawrite(var comp, nodename:nametype;
  82                                         var sendvalue:real);
  83                     begin
  84     1     2     1       entprocedure(datawrite1)
  85                     end;
  86
  87                     procedure getmyname(var thisname:nametype);
  88                     begin
  89     1     2     1       entprocedure(getmyname1)
  90                     end;
  91
  92                     function ulim(u,lolim,hilim:real):real;
  93                     begin
  94     1     2     1       ulim:=u;
  95     2     2     1       if u<lolim then ulim:= lolim;
  96     4     2     1       if u>hilim then ulim:= hilim
  97                     end;{ulim}
  98
  99                     function nodevalue(var addr:address):real;
 100                     var value:real;
 101                     begin
 102     1     2     1       if addr.name=AD then value:=adin(addr.number)
 103     3     2     2       else getoutvalue(addr.number,value);
 104     4     2     1       nodevalue:=value;
 105     5     2     1     end;{nodevalue}
 106
 107                     procedure setouttime(var intime,outtime,lasttime,
 108                                          period:integer);
```

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

 109                     var rtime:integer;
 110                     begin
 111     1     2     1     getrealtime(rtime);
 112     2     2     1     outtime:=rtime;
 113     3     2     1     if (outtime - intime))=(period div 2)
 114     4     2     2       then lasttime:=outtime;
 115     5     2     1   end; {setouttime}
 116
 117                     procedure inreg;
 118                     var value,ul:real;
 119                     begin
 120     1     2     1     with node.parampart do begin
 121
 122     3     2     3       if insig.name()EXT
 123     4     2     4         then with node.state do begin
 124     6     2     6           value:=scale*nodevalue(insig);
 125     7     2     6           value:=(1.0-filter)*filterstate+filter*value;
 126     8     2     6           filterstate:=value
 127                             end {then}
 128     9     2     4         else value:=node.outvalue;
 129
 130    10     2     3       ul:=ulim(value,-1.0,1.0);
 131
 132    11     2     3       if (goalcomp=myname) and (outplace.name=DA) then
 133    12     2     4         begin
 134    13     2     5           setouttime(intime,outtime,lasttime,node.period);
 135    14     2     5           daout(outplace.number,ul)
 136                           end
 137    15     2     4       else if (goalcomp()myname)
 138                                   and (outplace.name()DA) then
 139    16     2     5         begin
 140    17     2     6           setouttime(intime,outtime,lasttime,node.period);
 141    18     2     6           datawrite(goalcomp,outplace.name,value)
 142                           end
 143
 144                     end;{with}
 145    19     2     1     node.outvalue:=value
 146                     end;{inreg}
 147
 148
 149                     Procedure PIDreg;
 150                     var e,yr,y,u:real;
 151                     begin
 152     1     2     1     with node.parampart do begin
 153     3     2     3       with node.state do begin
 154     5     2     5         yr:=nodevalue(PIDref);
 155     6     2     5         y:=nodevalue(PIDin);
 156     7     2     5         e:=yr-y;
 157     8     2     5         ipart:=ipart+alfa*e;
 158     9     2     5         dpart:=beta*(yold-y);
 159    10     2     5         u:=k*e+ipart+dpart;
 160    11     2     5         u:=ulim(u,-limit,limit);
 161    12     2     5         yold:=y;
 162    13     2     5         if ti)0.0 then ipart:=u-dpart-k*e
```

```
LINE  STMT LEVEL  NEST   SOURCE STATEMENT

 163                          {anti-reset-windup}
 164                        end; {with node.state}
 165    15     2     3       if (goalcomp=myname)
 166                             and (PIDout.name=DA) then
 167    16     2     4         daout(PIDout.number,u) else if
 168                            (goalcomp()myname)
 169                              and (PIDout.name()DA) then
 170    18     2     5         datawrite(goalcomp,PIDout.name,u);
 171    19     2     3       node.outvalue:=u
 172                         end {with node.parampart}
 173                       end;{PIDreg}
 174
 175
 176                       Procedure outreg;
 177                       var out,out1:real;
 178                       begin
 179     1     2     1       with node.parampart do begin
 180     3     2     3         out:=level;
 181     4     2     3         if scal1()0.0 then
 182     5     2     4           out:=out+scal1*nodevalue(insig1);
 183     6     2     3         if scal2()0.0 then
 184     7     2     4           out:=out+scal2*nodevalue(insig2);
 185     8     2     3         if scal3()0.0 then
 186     9     2     4           out:=out+scal3*nodevalue(insig3);
 187    10     2     3         if scal4()0.0 then
 188    11     2     4           out:=out+scal4*nodevalue(insig4);
 189    12     2     3         out1:=ulim(out,-1.0,1.0);
 190    13     2     3         node.outvalue:=out;
 191    14     2     3         if (goalcomp=myname)
 192                               and (output.name=DA)
 193    15     2     4           then daout(output.number,out1) else if
 194                              (goalcomp()myname)
 195                                and (output.name()DA)
 196    17     2     5           then datawrite(goalcomp,output.name,out);
 197    18     2     3       end {with}
 198                       end;{outreg}
 199
 200                       begin {regul}
 201     1     1     1       getmyname(myname);
 202     2     1     1       repeat
 203     3     1     2         wait;
 204     4     1     2         regputgetnode(node,anyfound);
 205     5     1     2         while anyfound do begin
 206     7     1     4           case node.parampart.tag of
 207     8     1     5             innode:  inreg;
 208     9     1     5             PIDnode: PIDreg;
 209    10     1     5             outnode: outreg
 210                            end;{case}
 211    11     1     4           regputgetnode(node,anyfound)
 212                          end {while}
 213                        until false;
 214    12     1     1   end {regul}.

ERRORS DETECTED:  0
FREE MEMORY:  9000  WORDS
```