CODEN:LUTFD2/(TFRT-5262)/1-126/(1981)

LOGGER -
AN INTERACTIVE PROGRAM FOR DATA LOGGING ON PDP-11

KARL-ERIK ÅRZÉN

INSTITUTIONEN FÖR REGLERTEKNIK
LUNDS TEKNISKA HÖGSKOLA
NOVEMBER 1981

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

Document name
MASTER THESIS

Date of issue
November, 1981

Document number
CODEN:LUTFD2/(TFRT-5262)/1-126/(1981)

Author(s)
Karl-Erik Årzén

Supervisor
Leif Andersson

Sponsoring organization

Title and subtitle

LOGGER - An interactive program for data logging on PDP-11..

Abstract

Logger is an interactive command driven program for data logging on PDP-11. It can also generate different signals, perform PID-control and make signal processing. During the logging it is possible to plot or print signals on the terminal. Commands can be entered via the terminal or in a command file. For conversion of the stored signals to text form the program Convrt can be used.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
126

Recipient's notes

Security classification

LOGGER

An interactive program
for data logging
on PDP-11.

Master thesis in Automatic Control

by

Karl-Erik Arzen

Departement of Automatic Control,
Lund Institute of Technology.
Nov. 1981.

CONTENTS.

LOGGER - User's Manual

1. INTRODUCTION

Logger is a command driven interactive real-time program for data logging on PDP-11. The main idea behind the program is to make something that is extremely easy to learn and to use.

The main task of the program is to log signals in files on a floppy disk. During the logging it is possible to plot the signals or to print their values on the terminal. To facilitate process identification different signal generators, such as squarewave, triangle, sawtooth, sine and PRBS, are provided.

It is possible to control the process with PID-controllers and to make signal processing such as filtering, computing statistics and computing mathematical expressions of signals.

The commands may be entered via the terminal or in a command file.

The logged signals are stored in binary form. The analysis of the logged signals is probably done by the program IDPAC. The input file to IDPAC must be a text file. For the conversion from binary form to text form the program CONVRT can be used.

The program is written in OMSI-Pascal and uses the real-time kernel (Elmqvist, H. and Mattsson S.E. 1981) developed at the Departement of Automatic Control, Lund Institute of Technology. Further information and program listing can be found in LOGGER - Implementation report.

2. RUNNING THE LOGGER

The data structure of the logger is a list of function modules which is interpreted with fixed interval, the so called main period. The function modules are signal generators, filters etc. After the interpretation of the list the signals are logged if a logging instant is due. The signals are logged in a log file. When the log file is full the logging can continue in a new log file. The change to the new log file can also be done from the terminal.

The first decision to make when you are running the Logger is how to choose the main period. It must be chosen as a multiple of 0.020 seconds and it determines the smallest sampling interval. The main period must not be chosen smaller then necessary since then there is a risk that the sampling period will be overrun. There are no safeguards against this. After the program has received the main period it answers with a > and the interpretion of the function module list is started.

The commands of the program can be divided into five groups.

* Log commands.
* Commands acting on the function modules.
* Commands dealing with the output to the terminal.
* Command file commands.
* Miscellaneous.

3. SHORT DESCRIPTION OF THE COMMANDS

Log commands

LOG - Opens the log file and an information file. Writes out the time you can log before the log file is full.
RUNLOG - Starts the logging.
ENDLOG - Ends the logging.
NEWLOGFILE - Opens a new log file.
CHANGE - Change from the old log file to the new log file.
LOGMESS - Makes it possible to write a text line in the information file.

Commands acting on the function modules

NEW - Creates a new module.
PAR - Changes the parameters of a module.
DISP - Displays the parameters of a module.
DELETE - Deletes a module.

Commands for terminal output.

PRINT - Starts the printing of signal values.
ENDPRINT - Ends the printing.
PLOT - Starts the plotting of signals.
ENDPLOT - Ends the plotting.

Command file commands.

COMFILE - Starts the execution of a command file.
BREAK - Breaks the execution.
WAIT - Holds the execution.

Miscellaneous

HELP - Writes a help text.
EXIT - Exits to RT-11.
ADRANGE - Informs the program about the range setting of the AD-converters.
DARANGE - Informs the program about the range setting of the DA-converters.

4. FUNCTION MODULES.

Maximum number of modules is 8.

The program can accept three different types of input signals to the program.

* An AD-converter. It is written as AD0...AD15.
* The name of a previously created module.
* A constant written as CONST value.

When an AD-converter is used as an input signal to more than one module then a new value will be used by all the modules i.e. the AD-converter will be sampled more than one time during the interpretation of the function module list.

Signal_generators.

PULSE - Creates a square-wave.

Parameters:

AMP	- Peak-peak amplitude in volts.
MEANVALUE	- Mean value in volts.
PERIOD	- Period of the square-wave in seconds.
OUTCHANNEL	- The number of the output DA-converter. Negative means no output.
DUTYCYCLE	- Duty cycle in per cent. Initially set to 50.

TRIANGLE - Creates a triangle-wave.

Parameters:

AMP	- Peak-peak amplitude in volts.
MEANVALUE	- Mean value in volts.
PERIOD	- Period in seconds.
OUTCHANNEL	- The number of the output DA-converter. Negative means no output.

SAWTOOTH - Creates a sawtooth-wave.

Parameters:

AMP	- Peak-peak amplitude in volts.
MEANVALUE	- Mean value in volts.
PERIOD	- Period in seconds.
OUTCHANNEL	- The number of the output DA-converter. Negative means no output.

SINE - Creates a sine-wave.

Parameters:
AMP - Peak-peak amplitude in volts.
MEANVALUE - Mean value in volts.
PERIOD - Period in seconds.
OUTCHANNEL - The number of the output DA-converter. Negative means no output.

PRBS - Creates a pseudo-random binary signal.The prbs signal is generated by a 15 stage feedback shift- register implemented as a binary word. The maximum length sequence is 32767.

Parameters:
AMP - Peak-peak amplitude in volts.
MEANVALUE - Mean value in volts.
SAMPPERIOD - Shortest pulse period in seconds.
OUTCHANNEL - The number of the output DA-converter. Negative means no output.

Other functions.

PID - PID-contoller with limits on the control signal and anti reset windup.

```
e:= PIDREF - PIDIN
Ppart:=K*e
Ipart:=Ipart(n-1) + K * (main period)/TI
Dpart:=(TD/(TD+GD*(main period))*Dpart(n-1)
+ (K*TD/main period)*(1-(TD/(TD+GD
*(main period))) * (PIDIN(n-1)-PIDIN)
u:=Ppart+Ipart+Dpart
```

Parameters:

PIDIN	- Input signal.
PIDREF	- Reference signal.
K	- Gain.
TI	- Integral time in seconds. TI <= 0 means no integration.
TD	- Derivation time in seconds.
GD	- Derivation filter factor, i.e. the ratio between TD and the derivation filter time constant. GD must be > 0 when TD > 0.
MINU	- Low limit on the control signal in volts.
MAXU	- High limit on the control signal in volts.
OUTCHANNEL	- The number of the output DA-converter. Negative means no output.

FILTER - Third order low-pass Butterworth filter computed using bilinear transformation and implemented on normal form.

Parameters:

INPUT	- Input signal.
TIMECONST	- The inverted cut-off frequency in seconds. Must not exceed (main period)/2.
OUTCHANNEL	- The number of the output DA-converter. Negative means no output.

ADD - Computes a weighted sum of maximum 4 signals.

ADD:=Weight 1 * Addinput 1 + ...

Parameters:
ADDINPUT nr - Input signal nr.
WEIGHT nr - Weight nr.
NROFADDS - The number of input signals.
OUTCHANNEL - The number of the output DA-converter. Negative means no output.

STAT - Computes statistics on a signal. Mean, variance, lowest and highest signal value and at which sample number it was reached. The outvalues of the module can not be used as input signals to other modules but it is possible to log, plot or print them using the notation Name/MEAN, Name/VAR, Name/MINY, Name/MAXY.

Parameters:
INPUT - Input signal.
OUTCHMEAN - The number of the output DA-converter to the mean of the signal. Negative means no output. Initially -1.
OUTCHVAR - Like the previous but the variance of the signal.
OUTCHMINY - Like the previous but the minimum value of the signal.
OUTCHMAXY - Like the previous but the maximum value of the signal.

5. SYNTAX AND DESCRIPTION OF THE COMMANDS

Explanation of the syntax notation:

[ ] : Optional
... : The previous expression may be repeated.

All commands are terminated with Carriage return.

ADRANGE

Syntax: ADRANGE range

Informs the program about the range setting of the AD-converters. Initially is the program set for bipolar range. Possible ranges are Bipolar and Unipolar.

Bipolar: Voltage range: ±10 V. Resolution: 12 bits, corresponding to 5 mV/bit.

Unipolar: Voltage range: 0 - 10 V. Resolution: 12 bits, corresponding to 2.5mV/bit.

BREAK

Syntax: BREAK

Breaks the execution of a command file. Only valid when a command file is running.

CHANGE

Syntax: CHANGE

Changes the logging from the old log file to the new log file specified in NEWLOGFILE. The old log file is closed.

COMFILE

Syntax: COMFILE filename

Starts the execution of the command file filename.COM. The command file must have the extension .COM. Commands from the file are echoed on the terminal preceeded by a $. The only terminal command valid when a command file is running is BREAK.

DARANGE

Syntax: DARANGE range

Tells the program that the range switches of the DA-converters have been changed. Possible ranges are Bipolar and Unipolar. Initially is the program set for Bipolar range.

Bipolar: Voltage range: ±10 V. Resolution: 12 bits, corresponding to 5 mV/bit.

Unipolar: Voltage range: 0 - 10 V. Resolution: 12 bits, corresponding to 2.5 mV/bit.

Note: All the range switches must be changed.

DELETE

Syntax: DELETE name

Removes the module name from the module list. Invalid if the module is being logged, plotted or printed or if some other module uses the output.

DISP

Syntax: DISP name

Displays the parameters of the module name.

ENDLOG

Syntax: ENDLOG

Ends the logging. The actual log file and the information file are closed.

ENDPLOT

Syntax: ENDPLOT

Ends the plotting.

ENDPRINT

Syntax: ENDPRINT

Ends the printing.

EXIT

Syntax: EXIT

Returns control to RT-11.

HELP

Syntax: HELP [command]

Writes out an explaining help text to each command.

LOG

Syntax: LOG[/log period] filename signal ...

Opens the log file filename and writes out the time you can log, in hours minutes seconds, before the log file is full. Default file extension is .DAT. At the same time the information file filename.INF is opened on DX0:. The size of the information file is 10 blocks. This file initially contains the order of the signals in the log file, the log period and the actual range of the AD-converters. The signals are stored in binary form. Maximum number of signals that can be stored at a time is 8. The conversion of the stored signals to text format is done by the program CONVRT. The log period must be a multiple of the main period. Default log period is main period. The logging is started with RUNLOG and stopped with ENDLOG.

Note: The logged signals are stored according to the range of the AD-converters. That means that if you are logging the output of a module then the signals will be stored according to the range of the AD-converters even if the DA-converter range is different.

Warning: There must be at least 10 free blocks on DX0: for the information file, otherwise the program will crash.

LOGMESS

Syntax: LOGMESS Text to be written.

Writes one text line into the information file. You can write 30 full lines before the information file is full.

NEW

Syntax: NEW type [name]

Creates a new module of the type type and with the name name. Default name is type.

NEWLOGFILE

Syntax: NEWLOGFILE filename

Opens a new log file, filename.DAT, where the logging is continued when the original log file is full. The program writes out how long you can log, in hours.minutes.seconds, in the new log file.

PAR

Syntax: PAR name parameter=value ...

Changes the parameters of the module name. Can take as many parameters as there is space in the line.

PLOT

Syntax: PLOT[/plot period] signal ...

Starts the plotting of maximum 4 signals on the terminal. The signals are plotted with the characters X,O,!,- sequentially. The scale of the screen depends of the range of the AD-converters. Default plot period is main period. The plotting is stopped with ENDPLOT.

PRINT

Syntax: PRINT[/print period] signal ...

Starts the printing of maximum 4 signal values on the terminal. Default print period is main period.

RUNLOG

Syntax: RUNLOG

Starts the logging on the log file specified with command LOG.

WAIT

Syntax: WAIT hours minutes seconds

Holds the execution of the command file the time specified above. Only valid as a command in the command file.

6. CAPACITY ANALYSIS

Since there are no safeguards against overrunning the sample period the user must ensure that this will not be done himself.

Execution time of the different nodes:

Interpretation of empty list	2.8 ms
Execution of pulse	2.2 ms
-"- of Triangle	2.2 ms
-"- of Sawtooth	2.2 ms
-"- of Sine	6.2 ms
-"- of Prbs	3.6 ms
-"- of Pid	4.2 ms
-"- of Add	
2 inputs	6.2 ms
3 -"-	9.2 ms
4 -"-	12.2 ms
-"- of Filter	7.2 ms
-"- of Stat	11.2 ms

Log time including interpretation.

1 signal	9.0 ms
2 signals	11.0 ms

Thereafter + 5 ms per signal.

Plot time including interpretation.

1 signal	35 ms

Thereafter + 10 ms per signal.

Print time including interpretation.

1 signal	20 ms

Thereafter + 20 ms per signal.

All the execution times are maximum times.

When the smallest main period (0.02 seconds) is used is it only possible to log 2 signals. When the main period is smaller than 0.06 seconds is it necessary to be careful with the plotting and the printing.

7. RUNNING EXAMPLE.

The purpose of this running is to send a square-wave signal into a system and to log the input signal and 2 output signals in the file DX1:DATA.

Underlined text is written by the program.

```
.RUN DX1:LOGGER

Mainperiod: 1                          : Fastest sampling period
                                       : is set to 1 second.
>
>NEW PULSE PULS1                       : Creates a new pulse
Pk-pk_amplitude: 10                    : generator with the name
Period: 2                              : Puls1.
Meanvalue: 0                           : Amplitude and meanvalue
Outchannel: 1                          : in volts. period in
                                       : seconds.
>PAR PULS1 PERIOD=5                    : Changes the period.

>DISP PULS1                            : Displays the parameters
Type: PULSE                            : of Puls1.
Outchannel: 1
Pk-Pk_amplitude: 10
Period: 5
Meanvalue: 0
Dutycycle: 50 %

>LOG/2 DX1:DATFIL.DAT PULS1 AD1 AD2    : Opens the log file
You can log 1:12:54                    : and the information
                                       : file. Puls1, AD1 and
                                       : AD2 is to be logged
                                       : every 2 seconds.

>LOGMESS Test nr.1 Plant nr.2          : Text written in the
                                       : information file.

>RUNLOG                                : The logging is started.

>NEWLOGFILE DATA2.DAT                  : New log file where the
You can log 2:43:21                    : logging continues when
                                       : the log file is full.

Change of log file has been made       : After 1:12:54 is the
                                       : logging continued in
                                       : the new log file.

>ENDLOG                                : The logging is ended.

>DELETE PULS1                          : Puls1 is deleted.

>EXIT                                  : Exit to RT-11.
```

8. RUNNING CONVRT

CONVRT is a program that converts the integer file that is the output from the logger. It uses the information file created by the logger in the conversion.

CONVRT is a question-answer program. It starts with asking for the input file.

```
Input file:
```

Default extension is .DAT. When you have written your input file the program starts to search for the file input file.INF that must be either on DX0: or DX1:. Next question is

```
Output file:
```

Default extension of the output file is .TXT. Then the program asks

```
Do you want the whole file converted:
```

If so the program tries to convert all of the file and if there is room on the disk it will succeed. If you answer no to the question the program will ask from which sample you will start to convert and at which sample you will stop. Thus you have the possibility of converting only parts of the file.

The output file will be written as a matrix with the rows corresponding to the sample times and the columns corresponding to the differrent signals sampled at that time.

LOGGER - Implementation Report.

1. PROGRAM STRUCTURE.

The logger is written in OMSI-Pascal and is using the real-time kernel (Elmqvist, H. and Mattsson, S.E. 1981) to achieve concurrency.

The main task for the program is data logging. It can also perform signal generating, filtering and PID control. During the logging there is a need to have an overview of the logged signals and therefore the program has the possibility of printing signal values and plotting signals.

The natural way to handle this is to gather the signal creating operations in some way and this is done by implementing them as different function modules and link them together in a double linked list. The modules consist of a record with variants. The variant part contains the parameters of the module and the fixed part is the name and output value of the module.

The logging, plotting and printing are treated equally since they all can be regarded as output operations on a signal.

The list is interpreted with fixed time interval, the so called main period. The different modules are stored in the list such that if module B uses the output of module A then B is stored after A.

The interpretation of the list is done by the process Listprocess. All the modules are executed every main period. When the intrepretation is ready the different output operations are done. The plotting and printing is done by Listprocess and the logging is done by the process Filehandler which receives the signals to be logged in a message from Listprocess. The process Clock is working as a real time clock and signals to Listprocess every main period.

The logger is command driven and the commands can be entered from a terminal or in a command file.

Commands from the terminal are decoded by the process Opcom. Depending on the command it calls the appropriate global procedure taking care of this command. The commands can either be acting on the function module list or deal with one of the output operations.

The commands in the command file are decoded in the same way by the process Extopcom. The only valid command from the terminal when a command file is running is BREAK. This command ends the execution of the command file. The execution can also be halted for a while with the command WAIT. The waiting is taken care of by the process Waitprocess.

2. COMMUNICATION AND SYNCRONIZATION.

Messages

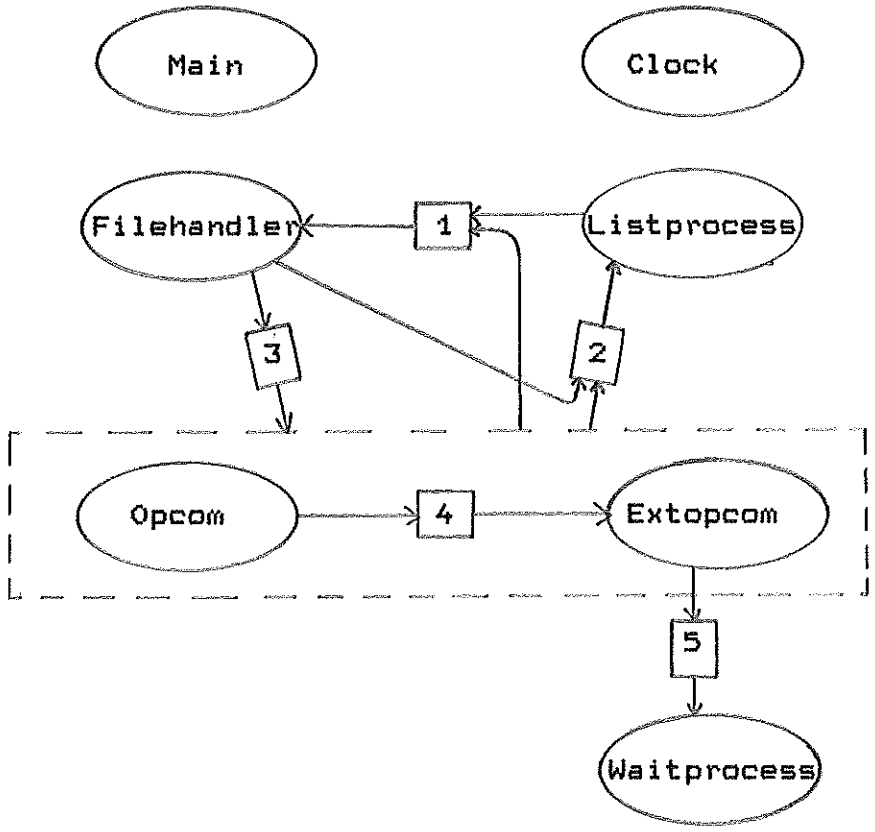
The communication between the processes is mainly done with messages and mailboxes.

The message is a record containing two pointers, data and info. Data points to datamessage which is an array [1..256] of integers which is used in the logging. Info points to a infomessage which is a record with variants, each according to a message type. This division of a message into two parts saves memory since the pointers may point to nil when they are not needed, and since the same message primitives may be used for all kinds of messages.

The mailboxes are

filebox	-	messages to Filehandler.
listbox	-	-"- to Listprocess
opcombox	-	-"- to Opcom or Extopcom.
extbox	-	-"- to Extopcom from Opcom.
waitbox	-	-"- to Waitprocess.
datapoolbox	-	message pool for messages with info pointing to nil.
infopoolbox	-	mesasge pool for messages with data pointing to nil.

It is possible to let both Opcom and Extopcom receive messages from the same mailbox since they never run concurrently.



Processes and mailboxes. 1: Filebox. 2: Listbox
3: Opcombox. 4: Extbox. 5: Waitbox.

The different types of information messages are

startlog	-	Tells Filehandler that a log is to begin.
logmess	-	Tells Listprocess that a log is to begin.
newfile	-	Tells Filehandler that a new log file should be opened and Extopc that a command file should be started.
changemess	-	Tells Filehandler that the logging should change from the old log file to the new.
blockmess	-	Tells Opcom or Extopc how many blocks is available for the logfile.
infomess	-	Text to be written in the information file.

waitmess - Tells Waitprocess how long it should wait.

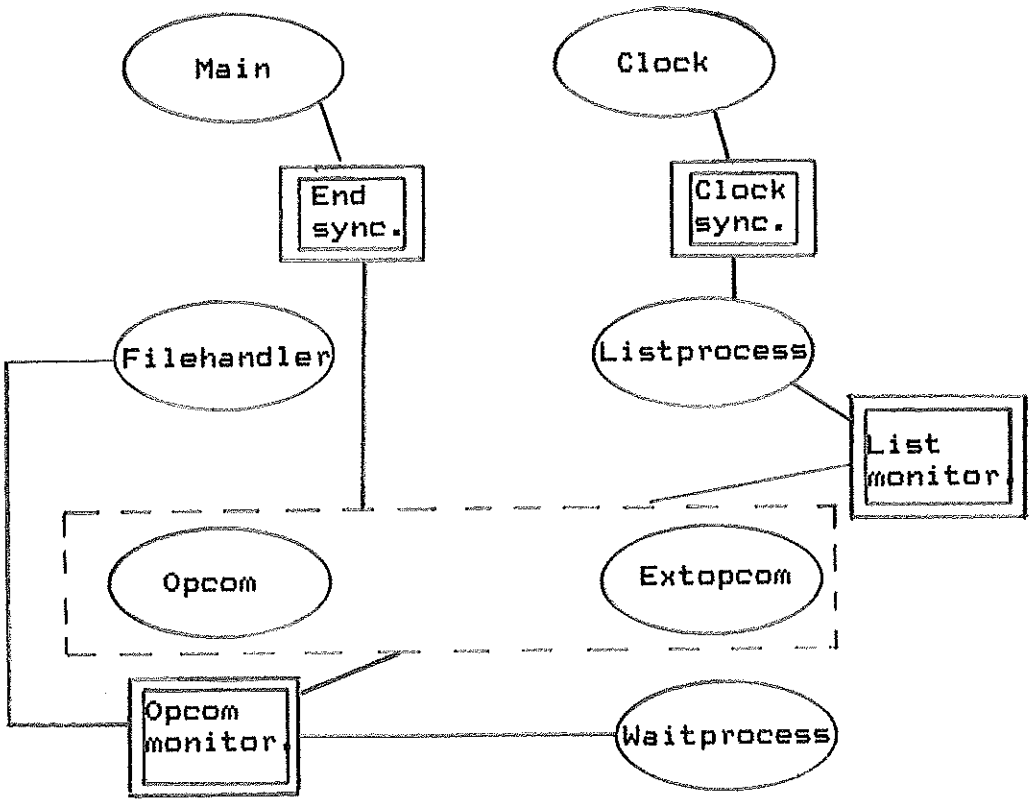
plotmess - Tells Listprocess that a plot or print is
printmess to begin.

endlogmess - Tells that an output operation is over.
endplotmess
endprintmess

okmess - Acknowledgement messages.
errormess

Monitors

There are two monitors in the program, listmonitor and opcommonitor.



Monitors and semaphores for syncronization.

Listmonitor

Listmonitor contains the function module list and information about the actual range of the AD- and DA-converters.

The monitor procedures acting on the listmonitor are as follows.

Procedures called by Opcom or Extopc.

Monnewnode	-	Inserts a new module in its correct place.
Mongetnode	-	Gets the parameter values of a module.

Monsetnode - Sets the parameter values of a module.

Mondelete - Deletes a module from the list unless it is being used by some other module.

Monlevelchange - Changes the actual range of the AD- or DA-converters.

Monname - Checks if a name exists in the list and returns the module type.

Moncheck - Checks if there is room for a new module in the list and that the name is unique.

Monordercheck - Checks the order of two modules.

Procedures called by Listprocess.

Interprete - Performs the interpretation of the module list. Has internal procedures taking care of the different module types. Returns to the Listprocess the signals that are to be logged, plotted or printed.

There also exists

Initlistmonitor- Initiates the listmonitor.

Opcommonitor.

Variabels that are common to Opcom, Extopcom and Waitprocess are stored in the monitor Opcommonitor. It contains different boolean flags, which are used to check if a command is valid or not, a double linked list where the name of the modules that are being logged, plotted and printed are stored, an event start that tells that the execution of a command file shall continue, the range of the AD and DA's and some information about the logging.

The monitor procedures acting on opcommonitor are as follows.

Initopcommonitor - Initiates the opcommonitor.

Checkcommand - Checks if a command is valid.

Logfinished	–	Tells that the logging has stopped.
Removenames	–	Removes the module names from the module name list. Used when a log, plot or print is ended.
Setname	–	Inserts a module name in the module name list.
Opcommoncheck	–	Checks if a module name is in the list i.e. that it is being logged, plotted or printed.
Logstart	–	Tells that the logging has started.
Logtime	–	Function that returns the time in seconds that it is possible to log.
Opmonlevel	–	Function that returns the actual AD range.
Setopmonlevel	–	Changes the AD range.
Setcomfile	–	Tells that a command file has started.
Extcomfile	–	Function that returns if a command file is executing or not.
Causestart	–	Causes the event start. This is done when the waiting time is over.
Waitstart	–	Makes await(start). This is done when a waiting time is to begin.

Global variabels

Some global variables are used by more than one process. There is however no problem with mutual exclusion because the processes only refer to the variables, they do not operate on them.

The most important of these variable is the main period which is set in the initialization and then is used by all the processes. The other variables are mainly arrays of name of the commands, parameters etc..

Semaphores

Two semaphores are used only for syncronization.

Clocksync is signaled every main period by process Clock. When listprocess has interpreted the module list it makes wait(clocksync).

Endsync is signaled either by Opcom or Extopcom when the command EXIT is decoded. After the main program has created the processes it makes wait(endsync). When the program is started again it terminates the execution of the program.

3. THE LOGGING.

The logged signals are stored in binary form in a file of datatype where datatype is an array[1..256] of integers. 256 integers is the size of one block. The signals are written one block at a time. The way of storing signals as integers is the least memory requiring method.

All the operations on the log file must be done by one process, the Filehandler. If the Listprocess itself wrote the signal blocks in the log file then it would be interrupted and the sample period overrun. To avoid this Listprocess sends a message with the signal block over to Filehandler who performs the writing and Listprocess will not be halted.

The logging is ended either by command ENDLOG or when the logfile is full.

4. PROCESSES.

The process Clock must have the highest priority to ensure that it will execute every main period. Waitprocess must also have a high priority. Filehandler must have lower priority than Listprocess else Listprocess would be interrupted when the writing on the log file is done. Opcom has the lowest priority and acts as idle process i.e. it runs when no other process is running. Extopc has higher priority than Opcom. This leads to the fact that the command BREAK cannot stop the execution of the command file unless the command file is waiting. This is however unimportant in practice.

Listprocess

```
{process} procedure listprocess;

begin
setpriority(3);
while true do
  begin
  if log, plot or print then decrement the actual timers;
  interprete the module list;
  if logtimer = 0 then
    begin
    fill the data message with the signals to be logged;
    if the message is full then send it to filehandler;
    end;
  if plottimer = 0 then
    write a plot line on the terminal;
  if printtimer = 0 then
    write the signal values on the terminal;
  if there is a message in the listbox then
    case message of
        logmess     : prepare the logging;
        runlogmess  : start the logging;
        endlogmess  : end the logging;
        plotmess    : start the plotting;
        endplotmess : end the plotting;
        printmess   : start the printing;
        endprintmess: end the printing;
    end;{case}
  wait for semaphore clocksync;
  end;{while}
end;{listprocess}
```

Filehandler

```
{process} procedure filehandler;

begin
setpriority(7);
while true do
  begin
  wait for a startlog message;
  open the log file and the information file;
  while not ready do
    begin
    wait for a message;
    case message of
      data       : write the data block in the logfile;
      infomess   : write a text line in the information file;
      newfile    : open the new log file;
      change     : change to the new log file;
      endlog     : ready:=true;
    end;{case}
  end;{while}
  close the log file and the information file;
  end;{while}
end;{filehandler}
```

Clock

```
{process} procedure clock;

begin
setpriority(1);
while true do
  begin
  waittime(main period);
  signal(clocksync);
  end;{while}
end;{clock}
```

Opcom

```
{process} procedure opcom;

begin
setpriority(12);
while true do
  begin
  decode a command;
  case of the command call appropriate global procedure;
  end;{while}
end;{opcom}
```

Extopcom

```
{process} procedure extopcom;

begin
setpriority(9);
while true do
  begin
  wait for a newfile message;
  open the command file;
  while not break and not eof do
    begin
    decode a command;
    case of the command call the appropriate global procedure;
    end;
  close command file;
  end;{while}
end;{extopc}
```

Waitprocess

```
{process} procedure waitprocess;

begin
setpriority(2);
while true do
  begin
  wait fo a wait message;
  while time left to wait and not break do
    begin
    waittime(one second);
    count down the time left to wait;
    end;
  cause the process Extopc to continue;
  end;{while}
end;{waitprocess}
```

5. IMPORTANT GLOBAL PROCEDURES.

Command_decoding_procedures

Both Opcom and Extopcom use the same global procedes for command decoding. This is possible because in OMSI-pascal procedures are reentrant.

The global procedures all have at least one parameter, mode of type sourcetype. Sourcetype is an enumeration type that can have the values term or fil. It tells from which process the procedure call was done. Instead of using read, readln and eoln they use a set of procedures that, depending on mode, read either from the terminal or from the command file. If the reading is done from the command file the procedures write the value on the terminal.

The set of read procedures contains

Endofline	-	Function similar to eoln.
Readchar	-	Similar to read(char).
Readint	-	Similar to read(integer).
Readrel	-	Similar to read(real).
Readnewline	-	Similar to readln.
Relreadnew	-	Similar to readln(real).
Intreadnew	-	Similar to readln(integer).

The global decoding procedures are as follow.

Command	Procedure

NEW	procedure newnode;
	
	begin
	reads the type and name of the module;
	writes out questions for the parameters;
	inserts the new module in the list;
	end;

PAR	procedure parnode; begin reads the module name; gets the parameters of the module;

```
                    while not endofline do
                      begin
                      reads new parameter value;
                      changes the parameter;
                      end;{while}
                    returns the parameters to the module;
                    end;

DISP                procedure dispnode;

                    begin
                    reads the module name;
                    gets the parameters of the module;
                    writes out the parameters;
                    end;

DELETE              procedure deletenode;

                    begin
                    reads the module name;
                    removes the module from the module list;
                    end;

LOG                 procedure logroutine;

                    begin
                    read log period;
                    read filename;
                    while not endofline and number of signals < 8 do
                      begin
                      read signalname;
                      if module name then
                        insert the name in the opcommonitor list;
                      end;
                    send startlog message to Filehandler;
                    wait for blockmess;
                    compute and write out the max. possible log time;
                    send logmess message to Listprocess;
                    end;

RUNLOG,             procedure listmess;
ENDPLOT,
ENDLOG,             begin
ENDPRINT            send appropriate message to Listprocess.
                    if end of an output operation then
                      remove the nodename from opcommonitor list;
                    end;

NEWLOGFILE          procedure newlogfile;

                    begin
                    read the new filename;
                    send newfile message to Filehandler;
                    wait for answer;
                    if errormess then write error text
```

```
                    else compute and write out the max. possible
                    time to log;
                    end;

CHANGE              procedure change;

                    begin
                    send changemess to Filehandler;
                    wait for answer;
                    if errormess then write error text;
                    end;

LOGMESS             procedure messlog;

                    begin
                    read the text to be written;
                    send infomess to Filehandler;
                    end;

PLOT,               procedure outroutine;
PRINT
                    begin
                    read the period;
                    while not eoln and number of signals < 4 do
                      begin
                      read signal;
                      if node name then insert it in opcommonitor;
                      end;
                    send plotmessage or printmessage to
                    Listprocess;
                    end;

HELP                procedure helproutine;

                    begin
                    read command;
                    call internal procedure according to each
                    command;
                    end;

ADRANGE             procedure adlevel;

                    begin
                    read range;
                    call the monitor procedure monlevelchange;
                    call the monitor procedure setopmonlevel;
                    end;

DARANGE             procedure dalevel;

                    begin
                    read range;
                    call the monitor procedure monlevelchange;
                    end;
```

```
COMFILE         procedure commandfilestart;

                begin
                read filename;
                send newfile message to Extopc;
                end;

BREAK           procedure breakrut;

                begin
                send message to Extopc that the execution
                shall end;
                end;

WAIT            procedure waitroutine;

                begin
                read the waiting time;
                send wait message to Waitprocess;
                do await on event start;
                end;

EXIT            procedure endprogram;

                begin
                make signal on endsync;
                end;
```

Input-output procedures

The possibility of having bipolar and unipolar range of the AD and DA converters makes it impossible to use the two procedures Adin and Daout in Paslib. Instead the program uses the procedures Adin2 and Daout2 which both have the actual range of the converters as input parameter. For program code see Appendix.

6. LINKING.

The program code is to big to allow the whole program to be resident. There are two overlay regions. Overlay region 1 contains the different global command decoding procedures. Naturally they are not called at the same time. Overlay region 2 contains the different procedures that are called by the procedures in region 1. The processes and the procedures that are being called by more than one process simultaneously are in the resident part of the program.

7. REFERENCES.

Elmqvist, H. and Mattsson, S.E. (1981): A Real-time kernel for Pascal. Departement of Automatic Control, Lund Inst. of Technology, Lund, Sweden.

Elmqvist, H., Mattsson, S.E. and Olsson, G. (1981): Datorer i reglersystem - Realtidsprogrammering. Departement of Automatic Control, Lund Inst. of Technology, Lund, Sweden.

OMSI PASCAL-1 Documentation Version 1.1. Oregon Minicomputer Software Inc., 2340 SW Canyon Road, Portland, Oregon 97201.

Appendix

CONTENTS.

LOGGER

```
program datalogger;

const sizekerneldata = 25;
      sizetermdata = 400;
      maxpriority = 1000;
      tick = 1; sec = 50; min = 3000;

type unsignedinteger = 0..65535;
     semaphore = unsignedinteger;
     event = unsignedinteger;

var kerneldata: array [1..sizekerneldata] of unsignedinteger;
    terminaldata: array [1..sizetermdata] of unsignedinteger;

procedure initkernel(memreq:unsignedinteger);external;
procedure createprocess(procedure proced;
          memreq:unsignedinteger);external;
procedure setpriority(priority:integer);external;
procedure initsem(var sem:semaphore;initval:integer);external;
procedure wait(sem: semaphore);external;
procedure signal(sem: semaphore);external;
procedure initevent(var e:event;sem:semaphore);external;
procedure await(e: event);external;
procedure cause(e: event);external;
procedure waittime(t:integer);external;

type messageref = ^message;
     mailbox = unsignedinteger;

procedure initmailbox(var box: mailbox); external;
procedure sendmessage(box:mailbox;var mess:messageref);external;
procedure receivemessage(box:mailbox;var mess:messageref);
                         external;

const  idlength = 10;          maxoutsig = 8;
       AD = 'AD        ';      mark = ',         ';
       nodelistsize = 10;     konst = 'CONST     ';
       blanks = '          '; pi2 = 6.283184;
       maxad = 9;             bipolar = 'BIPOLAR   ';
       infofilesize = 10;     unipolar = 'UNIPOLAR  ';
       LP = 'LP        ';     TT = 'TT        ';
       maxlogsig = 8;         maxprintsig = 4;
       pi = 3.141593;

type   identtype = array [1..idlength] of char;
       datatype = array [1..256] of integer;
       postag = (adsig,nodesig,constsig);
       stattype = (mean,vari,miny,maxy,laststat);
       sourcetype = (term,fil);
       dataref=^datamessage;
       inforef=^infomessage;
       datamessage=datatype;
```

```
 message=record
   nextmess:messageref;
   data:dataref;
   info:inforef;
 end;{message}

 signaldescriptortype = record
   value:real;
   case sigtag:postag of
     adsig       :(chan:integer);
     nodesig     :(name:identtype;
                   ext:stattype);
     constsig    :(dummy:integer);
 end;{signaldescriptortype}

outmodetype = (log,print,plot);
leveltype = (bipolarlev,unipolarlev);
alleveltype = (adbipolar,adunipolar,dabipolar,
               daunipolar);
infomesstype = (startlog,endlogmess,changemess,runlogmess,
            endplotmess,endprintmess,errormess,okmess,newfile,
            blocksize,infomess,logmess,waitmess,plotmess,
            printmess);

infomessage = record
  case infotype:infomesstype of
    startlog  :(nsignals:integer;
                logperiod:real;
                ext,logfile:identtype;
                level:leveltype;
                signalorder:array [1..8] of identtype);

    newfile   :(newext,newlogfile:identtype);
    blocksize :(blocks:integer);
    waitmess  :(hr,minu,sek:integer);
    infomess  :(string:array [1..80] of char);
    logmess   :(nlogsignals,sampleperiod:integer;
                loglevel:leveltype;
                messlogsignals:array [1..maxoutsig]
                               of signaldescriptortype);
    printmess,plotmess
              :(printlevel:leveltype;
                printsampleperiod,nprintsignals:integer;
                messprintsignals:array [1..maxoutsig]
                                 of signaldescriptortype);
    endlogmess,changemess,runlogmess,endplotmess,
    okmess,errormess,endprintmess   :(dummy:integer);
end;{infomessage}

opx = (newx,dispx,parx,deletex,logx,runlogx,logmessx,
       changex,newlogfilex,endlogx,printx,endprintx,plotx,
       endplotx,helpx,comfilex,breakx,adlevelx,dalevelx,
       endx,waitx,lastopindex);
```

```
pars = (pidin,pidref,outchanpar,kpar,tipar,tdpar,gdpar,
        minupar,meanvaluepar,timecon,outchanmaxy,
        maxupar,ptpamppar,periodpar,sampleperiodpar,
        dutycyclepar,input,outchanvar,outchanminy,
        outchanmean,addinputpar,weight,nrinsignalspar,lastpar);
nodetype = (pid,puls,prbs,sinus,triang,sawtooth,stat,
            add,filter,lasttype);
resultcode = (ok,nonode,active,nospace,ahead,referr,
              logging,plotting,printing,runlogresult,nolog,
              logresult,noplot,noprint,printresult,plotresult);
errors = (toomanyarg,toolongid,notin,fewarg,statnode,
          illtype,illname,neggd,badlimit,illpar,noass,
          logfile,changeerr,noslash,nodevice,illstat,
          nolevel,noop,toomanyadd,nyquist);

resulttype = record
  code:resultcode;
  name:identtype;
end;{resulttype}

parset = set of pars;

inputdescriptortype = record
  case intag:postag of
    adsig    :(chan:integer);
    nodesig  :(name:identtype);
    constsig :(value:real);
end;{inputdesciptortype}

nodepart = record
  name:identtype;
  outvalue:real;
  outchan:integer;
  case tag:nodetype of
    pid       :(pidinpdescr,pidrefdescr:inputdescriptortype;
              k,ti,td,gd,minu,maxu,alfa,beta,
              gamma,yold,dpart,ipart:real);
    triang,sawtooth,sinus,puls,prbs
              :(meanvalue,period,ptpamp,inttime
              ,dutycycle:real;
              timer,timervalue,state,
              newstate:integer;
              bit0:boolean);
    stat      :(inpdescr:inputdescriptortype;
              n,maxysample,minysample:integer;
              sum,sum2:real;
              statoutvalue:array [stattype] of real;
              statoutchan:array [stattype] of integer);
    filter    :(timeconst:real;
              a0,a1,a2,a3,b0,b1,b2,b3:real;
              u1,u2,u3,y1,y2,y3:real;
              filtinput:inputdescriptortype);
  add         :(nrofinsignals:integer;
              addinput:array [1..4] of inputdescriptortype;
              addfactor:array[1..4] of real);
end;{nodepart}
```

```
    outsignaltype = record
      tags:array [outmodetype] of boolean;
      level:leveltype;
      outsignals:array [outmodetype] of array [1..maxoutsig]
                 of signaldescriptortype;
      nroutsignals:array [outmodetype] of integer;
      end;{outsignaltype}

var  typename:array [nodetype] of identtype;
     parname:array [pars] of identtype;
     opname:array [opx] of identtype;
     statname:array [stattype] of identtype;
     mainperiod:real;
     inputs:array [0..15] of identtype;
     comfile:text;
     node:nodepart;
     identifier:array [sourcetype] of identtype;
     ops:array [sourcetype] of opx;
     ch:array [sourcetype] of char;
     result:array [sourcetype] of resulttype;
     clocksync:semaphore;
     endsync:semaphore;
     nodeparam:array [nodetype] of parset;
     infopoolbox,datapoolbox,filebox,listbox,opcombox:mailbox;
     extbox,waitbox:mailbox;
type nodeelemptr=^nodeelemtype;

     nodeelemtype=record
       fwd,bwd:nodeelemptr;
       nodename:identtype;
       flags:array [outmodetype] of boolean;
       end;

     listelemptr=^listelemtype;
     listelemtype = record
       fwd,bwd:listelemptr;
       node:nodepart;
       end;
     nodeset=set of nodetype;

var  listmonitor:record
       mutex:semaphore;
       activelist,pool:listelemptr;
       generators,oneinputnodes:nodeset;
       adlevel,dalevel:leveltype;
       end;
     opcommonitor:record
       mutex:semaphore;
       logflag,runlog,plotflag,printflag,extfile:boolean;
       start:event;
       nodelist:nodeelemptr;
       adlevel:leveltype;
       nrlogsignals,logperiod:integer;
       end;
```

```
procedure daout2(mode:leveltype;chan:integer;value:real);

var  ival:integer;
     COBA origin 167772B :integer;

begin
  case mode of
    bipolarlev   : begin
                   if value > 0.9995 then value:=0.9995;
                   if value < -1.0 then value:=-1.0;
                   ival:=trunc(value*2048.0);
                   ival:=ival and 7777B;
                   COBA:=chan*10000B+ival;
                   end;
    unipolarlev  : begin
                   if value > 0.99975 then value:=0.99975;
                   if value < 0.0 then value:=0.0;
                   ival:=trunc(value*4096.0);
                   COBA:=chan*10000B+ival;
                   end;
    end;{case}
end;{daout}


function adin2(mode:leveltype;chan:integer):real;

var  CIBA origin 167774B :integer;
     COBA origin 167772B :integer;
     int:integer;

begin
COBA:=177400B + chan;
while CIBA < 0 do
  begin
  end;
int:=CIBA and 7777B;
case mode of
  unipolarlev  : adin2:=int/4096.0;
  bipolarlev   : if (int and 4000B) <> 4000B then
                   adin2:=int/2048.0
                 else
                   begin
                   int:=int or 170000B;
                   adin2:=int/2048.0;
                   end;
  end;{case}
end;{adin}
```

```
{**************
* Listmonitor *
**************}


function lookup(var name:identtype):listelemptr;

{Points at a node in the nodelist}

var ptr:listelemptr;

begin
with listmonitor do
  begin
  ptr:=activelist;
  repeat
    ptr:=ptr^.fwd;
  until (ptr = activelist) or (ptr^.node.name = name);
  if ptr <> activelist then lookup:=ptr
  else lookup:=nil;
  end;{with}
end;{lookup}

{listmonitor} procedure interprete(var signals:outsignaltype);

{Interpretes the nodelist}


var   listptr:listelemptr;
      i:integer;
      extension:stattype;
      channel:integer;
      index:outmodetype;


function input(inpdescr:inputdescriptortype):real;

{Returns an input signal}

var  v:real;

begin{input}
with inpdescr do
  begin
  case intag of
    adsig     : v:=adin2(listmonitor.adlevel,chan);
    nodesig   : v:=lookup(name)^.node.outvalue;
    constsig  : v:=value;
    end;{case}
  input:=v;
  end;{with}
end;{input}
```

```
procedure pidreg;


{Performs the PID control}

var e,yr,y,ppart,u:real;

begin
with listptr^.node do
  begin
  y:=input(pidinpdescr);
  yr:=input(pidrefdescr);
  e:=yr-y;
  ppart:=k*e;
  ipart:=ipart+alfa*e;
  dpart:=beta*dpart+gamma*(yold-y);
  u:=ppart+ipart+dpart;
  if u < minu then u:=minu;
  if u > maxu then u:=maxu;
  outvalue:=u;
  if outchan >= 0 then daout2(listmonitor.dalevel,outchan,u);
  yold:=y;
  if alfa <> 0.0 then ipart:=u-dpart-ppart;
  end;{with}
end;{pidreg}


procedure pulsgen;

{Creates a square-wave}

var  y:real;

begin
with listptr^.node do
  begin
  y:=ptpamp/2.0;
  inttime:=inttime+mainperiod;
  if inttime > period then inttime:=inttime-period;
  if inttime > (period * dutycycle) then y:=-y;
  outvalue:=y+meanvalue;
  if outchan >= 0 then daout2(listmonitor.dalevel,outchan,outvalue);
  end;{with}
end;{pulsgen}

procedure prbsgen;

{Creates a PRBS}

var  i:integer;
     y:real;

begin
with listptr^.node do
  begin
```

```
  timer:=timer-1;
  if timer = 0 then
    begin
    timer:=timervalue;
    newstate:=state*2;
    state:=state and 60000B;
    bit0:=false;
    for i:=1 to 15 do
      begin
      bit0:=bit0 <> odd(state);
      state:=state div 2;
      end;{for}
    if bit0 then newstate:=newstate+1;
    state:=newstate and 32767;
    if odd(state) then y:=ptpamp/2.0
    else y:=-ptpamp/2.0;
    outvalue:=y+meanvalue;
    if outchan >= 0 then daout2(listmonitor.dalevel,
    outchan,outvalue);
    end;{if}
  end;{with}
end;{prbsgen}

procedure trianglegen;

{Creates a triangle-wave}

var y:real;

begin
with listptr^.node do
  begin
  inttime:=inttime+mainperiod;
  y:=inttime * (ptpamp/(period * 0.5));
  if y >= ptpamp then y:=ptpamp-(y-ptpamp);
  if inttime >= period then inttime:=inttime-period;
  outvalue:=y-(ptpamp/2.0)+meanvalue;
  if outchan >= 0 then daout2(listmonitor.dalevel,outchan,outvalue);
  end;{with}
end;{trianglegen}

procedure sinusgen;

{Creates a sinus-wave}

var  y:real;

begin
with listptr^.node do
  begin
  inttime:=inttime+mainperiod;
  y:=(ptpamp/2.0) * sin(inttime * pi2 / period);
  if inttime >= period then inttime:=inttime-period;
  outvalue:=y+meanvalue;
  if outchan >= 0 then daout2(listmonitor.dalevel,outchan,outvalue);
```

```
  end;{with}
end;{sinusgen}

procedure sawtoothgen;

{Creates a square-wave}

var  y:real;

begin
with listptr^.node do
  begin
  inttime:=inttime+mainperiod;
  y:=inttime *  ptpamp/ period;
  if inttime >= period then inttime:=inttime-period;
  outvalue:=y-(ptpamp/2.0)+meanvalue;
  if outchan >= 0 then daout2(listmonitor.dalevel,outchan,outvalue);
  end;{with}
end;{sawtoothgen}

procedure statistics;

{Computes statistics}

var y:real;
    index:stattype;

begin
with listptr^.node do
  begin
  if n <= maxint then
    begin
    y:=input(inpdescr);
    n:=n+1;
    if n = 1 then statoutvalue[mean]:=y
    else statoutvalue[mean]:=(((n-1)*statoutvalue[mean])+y)/n;
    sum:=sum+y;
    sum2:=sum2+y*y;
    if n = 1 then statoutvalue[vari]:=0.0
    else statoutvalue[vari]:=(sum2+n*statoutvalue[mean]*
         statoutvalue[mean]+2*statoutvalue[mean]*sum)/(n-1);
    if y >= (statoutvalue[maxy]) then
      begin
      statoutvalue[maxy]:=y;
      maxysample:=n;
      end;
    if y <= (statoutvalue[miny]) then
      begin
      statoutvalue[miny]:=y;
      minysample:=n;
      end;
    for index:=mean to maxy do
      if statoutchan[index] >= 0 then daout2(listmonitor.dalevel,

                              statoutchan[index],statoutvalue[index]);
```

```
    end;{if}
  end;{with}
end;{statistics}

procedure addnode;

{performs the addition}

var y:real;
    i:integer;

begin
with listptr^.node do
  begin
  y:=0;
  for i:=1 to nrofinsignals do
  y:=y+(input(addinput[i])*addfactor[i]);
  outvalue:=y;
  if outchan >= 0 then daout2(listmonitor.dalevel,outchan,outvalue);
  end;{with}
end;{addnode}

procedure filtnode;

{Performs the filtering}

var u,y:real;

begin
with listptr^.node do
  begin
  u:=input(filtinput);
  y:=b0*u+b1*u1+b2*u2+b3*u3-a1*y1-a2*y2-a3*y3;
  u3:=u2;
  u2:=u1;
  u1:=u;
  y3:=y2;
  y2:=y1;
  y1:=y;
  outvalue:=y;
  if outchan >= 0 then daout2(listmonitor.dalevel,outchan,outvalue);
  end;{with}
end;{filtnode}


begin{interprete}
{$A-}
with listmonitor do
  begin
  wait(mutex);
  listptr:=activelist^.fwd;
  while listptr <> activelist do
    begin
    case listptr^.node.tag of
      pid          : pidreg;
```

```
        puls          : pulsgen;
        triang        : trianglegen;
        sawtooth      : sawtoothgen;
        prbs          : prbsgen;
        sinus         : sinusgen;
        stat          : statistics;
        add           : addnode;
        filter        : filtnode;
        end;{case}
      listptr:=listptr^.fwd;
      end;{while}
    with signals do
      for index:=log to plot do
      if tags[index] then
        for i:=1 to nroutsignals[index] do
          begin
          if outsignals[index][i].sigtag = adsig then
          begin
          channel:=outsignals[index][i].chan;
          outsignals[index][i].value:=adin2(adlevel,channel);
          end;
          if outsignals[index][i].sigtag = nodesig then
            begin
            listptr:=lookup(outsignals[index][i].name);
            if listptr^.node.tag = stat then
              begin
              extension:=outsignals[index][i].ext;
              outsignals[index][i].value:=listptr^.node.
                 statoutvalue[extension];
              end
            else
              outsignals[index][i].value:=listptr^.node.outvalue;
            end;
          end;
    signal(mutex);
    end;{with}
{$A+}
end;{interprete}
```

```
{listmonitor} procedure monnewnode(node:nodepart);

{Inserts the node in the list in  correct order}

var  ptr,this:listelemptr;

begin
with listmonitor do
  begin {Fetch the first element in the pool}
  wait(mutex);
  this:=pool^.fwd;
  with this^ do
    begin
    ptr:=fwd;
    ptr^.bwd:=pool;
    pool^.fwd:=ptr;
    end;{with}
  {Insert the node in the activelist}
  this^.node:=node;
  ptr:=activelist^.bwd;
  if node.tag in generators then ptr:=activelist;
  if node.tag = stat then
    if node.inpdescr.intag = nodesig then
      ptr:=lookup(node.inpdescr.name);
  if node.tag = filter then
    if node.filtinput.intag = nodesig then
      ptr:=lookup(node.filtinput.name);
  if node.tag = pid then
    begin
    if (node.pidinpdescr.intag = nodesig) and
       (node.pidrefdescr.intag = nodesig) then
      begin
      ptr:=activelist;
      repeat
        ptr:=ptr^.fwd;
      until (ptr^.node.name = node.pidinpdescr.name) or
            (ptr^.node.name = node.pidrefdescr.name);
      repeat
        ptr:=ptr^.fwd;
      until (ptr^.node.name = node.pidinpdescr.name) or
            (ptr^.node.name = node.pidrefdescr.name);
      end
    else
      if node.pidinpdescr.intag = nodesig then
        ptr:=lookup(node.pidinpdescr.name)
      else
        if node.pidrefdescr.intag = nodesig then
          ptr:=lookup(node.pidrefdescr.name);
    end;
  this^.fwd:=ptr^.fwd;
  this^.fwd^.bwd:=this;
  ptr^.fwd:=this;
  this^.bwd:=ptr;
  signal(mutex);
  end;{with}
end;{monnewnode}
```

```
{listmonitor} procedure mondelete(nodename:identtype;
                                  var result:resulttype);

{Removes the node called nodename from the list unless
 1 : There is no node called nodename.
 2 : Other nodes use the output.}

var  this,ptr:listelemptr;
     i:integer;

begin
with listmonitor do
  begin
  wait(mutex);
  result.code:=ok;
  this:=lookup(nodename);
  if this = nil then
    begin
    result.code:=nonode;
    result.name:=nodename;
    end
  else
    begin
    ptr:=this^.fwd;
    while (ptr<>activelist) and (result.code=ok) do
      begin
      with ptr^ do
        begin
        if node.tag = stat then
          if node.inpdescr.intag = nodesig then
            if node.inpdescr.name = nodename then
              begin
              result.code:=referr;
              result.name:=node.name;
              end;
        if node.tag = filter then
          if node.filtinput.intag = nodesig then
            if node.filtinput.name = nodename then
              begin
              result.code:=referr;
              result.name:=node.name;
              end;
        if node.tag = pid then
          if node.pidinpdescr.intag = nodesig then
            if node.pidinpdescr.name = nodename then
              begin
              result.code:=referr;
              result.name:=node.name;
              end
          else
            if node.pidrefdescr.intag = nodesig then
```

```
                if node.pidrefdescr.name = nodename then
                  begin
                  result.code:=referr;
                  result.name:=node.name;
                  end;
          if node.tag = add then
            for i:=1 to node.nrofinsignals do
              if node.addinput[i].intag = nodesig then
                if node.addinput[i].name = nodename then
                  begin
                  result.code:=referr;
                  result.name:=node.name;
                  end;
          end;{with}
        ptr:=ptr^.fwd;
        end;{while}
      end;{if}
    if result.code = ok then
      with this^ do
        begin
        fwd^.bwd:=bwd;
        bwd^.fwd:=fwd;
        bwd:=pool;
        fwd:=pool^.fwd;
        pool^.fwd:=this;
        fwd^.bwd:=this;
        end;{with}
    signal(mutex);
    end;{with}
end;{mondelete}
```

```
{listmonitor} procedure monlevelchange(newlevel:alleveltype);

{Tells the listmonitor that the range of the AD- or DA-
 converters has been changed}

begin
with listmonitor do
  begin
  wait(mutex);
  case newlevel of
    adbipolar  :adlevel:=bipolarlev;
    adunipolar :adlevel:=unipolarlev;
    dabipolar  :dalevel:=bipolarlev;
    daunipolar :dalevel:=unipolarlev;
    end;{case}
  signal(mutex);
  end;{with}
end;{monlevelchange}



{listmonitor} procedure mongetnode(var node:nodepart;
                                   var result:resulttype);
{Gets the node from the list}

var  ptr:listelemptr;

begin
with listmonitor do
  begin
  wait(mutex);
  ptr:=lookup(node.name);
  if ptr = nil then
    begin
    result.code:=nonode;
    result.name:=node.name;
    end
  else
    begin
    result.code:=ok;
    node:=ptr^.node;
    end;
  signal(mutex);
  end;{with}
end;{mongetnode}
```

```
{listmonitor} procedure monsetnode(node:nodepart);

{Copies the node to the list}

var  ptr:listelemptr;

begin
with listmonitor do
  begin
  wait(mutex);
  ptr:=lookup(node.name);
  ptr^.node:=node;
  signal(mutex);
  end;{with}
end;{monsetnode}
```

```
{listmonitor} procedure monname(nodename:identtype;
                                var result:resulttype;
                                var typeindex:nodetype);

{Checks if the nodename exists in the list and returns the
 type}

var  ptr:listelemptr;

begin
with listmonitor do
  begin
  wait(mutex);
  ptr:=lookup(nodename);
  if ptr=nil then
    begin
    result.code:=nonode;
    result.name:=nodename;
    end
  else
    begin
    result.code:=ok;
    typeindex:=ptr^.node.tag;
    end;{if}
  signal(mutex);
  end;{with}
end;{monname}
```

```
{listmonitor} procedure moncheck(nodename:identtype;
                                 var result:resulttype);

{Checks if there is room for a new node and if the new
 nodename is unique}
```

```
var  ptr:listelemptr;

begin
with listmonitor do
  begin
  wait(mutex);
  ptr:=lookup(nodename);
  if ptr <> nil then
    begin
    result.code:=active;
    result.name:=nodename;
    end
  else
  if pool^.fwd = pool then result.code:=nospace
  else result.code:=ok;
  signal(mutex);
  end;{with}
end;{moncheck}


{listmonitor} procedure monordercheck(nodename1,nodename2:
                                      identtype;
                                      var result:resulttype);

{Checks if node2 is ahead of node1}

var  ptr:listelemptr;

begin
with listmonitor do
  begin
  wait(mutex);
  ptr:=lookup(nodename1)^.bwd;
  while (ptr<>activelist) and (ptr^.node.name<>nodename2) do
  ptr:=ptr^.bwd;
  if ptr = activelist then result.code:=ok
  else
    begin
    result.code:=ahead;
    result.name:=nodename2;
    end;
  signal(mutex);
  end;{with}
end;{monordercheck}
```

```
{listmonitor} procedure initlistmonitor;

{Initiates the listmonitor}

var  ptr:listelemptr;
     i:integer;

begin
with listmonitor do
  begin
  initsem(mutex,1);
  new(activelist);
  with activelist^ do
    begin
    fwd:=activelist;
    bwd:=activelist;
    end;{with}
  new(pool);
  ptr:=pool;
  for i:=1 to nodelistsize do
    begin
    new(ptr^.fwd);
    ptr^.fwd^.bwd:=ptr;
    ptr:=ptr^.fwd;
    end;{for}
  pool^.bwd:=ptr;
  ptr^.fwd:=pool;
  generators:=generators + [puls,prbs,sinus,triang,sawtooth];
  oneinputnodes:=oneinputnodes + [stat,filter];
  adlevel:=bipolarlev;
  dalevel:=bipolarlev;
  end;{with}
end;{initlistmonitor}
```

```
{***************
* opcommonitor *
***************}


{opcommonitor} procedure setcomfile(bool:boolean);

{Sets the boolean extfile }

begin
with opcommonitor do
  begin
  wait(mutex);
  extfile:=bool;
  signal(mutex);
  end;{with}
end;{setcomfile }


{opcommonitor} function extcomfile:boolean;

{Returns the value of extfile}

begin
with opcommonitor do
  begin
  wait(mutex);
  extcomfile:=extfile;
  signal(mutex);
  end;{with}
end;{extcomfile}



{opcommonitor} procedure causestart;

{Makes cause(start)}

begin
with opcommonitor do
  begin
  wait(mutex);
  cause(start);
  signal(mutex);
  end;{with}
end;{causestart}
```

```
{opcommonitor} procedure waitstart;

{Makes await(start)}

begin
with opcommonitor do
  begin
  wait(mutex);
  await(start);
  signal(mutex);
  end;{with}
end;{waitstart}



{opcommonitor} procedure checkcommand(command:opx;
                                      var result:resulttype);

{Checks if a command is valid}

begin
with opcommonitor do
  begin
  wait(mutex);
  case command of
    runlogx        : if logflag and not runlog then
                       begin
                       result.code:=ok;
                       runlog:=true;
                       end
                     else
                       if logflag then result.code:=runlogresult
                       else result.code:=nolog;
    logmessx,newlogfilex,changex
                   : if logflag then result.code:=ok
                     else result.code:=nolog;
    logx           : if logflag then result.code:=logresult
                     else
                       begin
                       result.code:=ok;
                       logflag:=true;
                       end;
    endlogx        : if logflag then
                       result.code:=ok
                     else result.code:=nolog;
    endplotx       : if plotflag then
                       begin
                       plotflag:=false;
                       result.code:=ok;
                       end
                     else result.code:=noplot;
    endprintx      : if printflag then
                       begin
                       printflag:=false;
                       result.code:=ok;
```

```
                    end
                  else result.code:=noprint;
    printx        : if printflag then result.code:=printresult
                  else
                    begin
                    printflag:=true;
                    result.code:=ok;
                    end;
    plotx         : if plotflag then result.code:=plotresult
                  else
                    begin
                    plotflag:=true;
                    result.code:=ok;
                    end;
    adlevelx,dalevelx,endx: if logflag  then
                      result.code:=logresult
                      else
                        if plotflag and (command <> endx) then
                      result.code:=plotresult
                      else result.code:=ok;

    end;{case}
  signal(mutex);
  end;{with}
end;{checkcommand}


{opcommonitor} procedure opcommoncheck(nodename:identtype;
                                       var result:resulttype);

{Checks if a nodename is in the nodename list i.e it is
 logged, plotted or printed}

var ptr:nodeelemptr;

begin
with opcommonitor do
  begin
  wait(mutex);
  ptr:=nodelist^.fwd;
  while (ptr <> nodelist) and (ptr^.nodename <> nodename) do
    ptr:=ptr^.fwd;
  if ptr = nodelist then result.code:=ok
  else
    begin
    result.name:=nodename;
    if ptr^.flags[log] then result.code:=logging;
    if ptr^.flags[plot] then result.code:=plotting;
    if ptr^.flags[print] then result.code:=printing;
    end;{if}
  signal(mutex);
  end;{with}
end;{opcommoncheck}
```

```
{opcommonitor} procedure logstart(nroflogsignals;
                                  mpmultiple:integer);

{Tells the opcommonitor the log period and the number of
 signals being logged}

begin
with opcommonitor do
  begin
  wait(mutex);
  nrlogsignals:=nroflogsignals;
  logperiod:=mpmultiple;
  signal(mutex);
  end;{with}
end;{logstart}


{opcommonitor} function logtime(blocks:integer):real;

{Computes the longest possible log time in seconds}

var  nrintegers:integer;

begin
with opcommonitor do
  begin
  wait(mutex);
  if blocks > 0 then
    begin
    nrintegers:=trunc(256/nrlogsignals);
    logtime:=nrintegers*logperiod*mainperiod*blocks;
    end
  else logtime:=0;
  signal(mutex);
  end;{with}
end;{logtime}


{opcommonmitor} function opmonlevel:leveltype;

{Returns the actual range of the AD-converters}

begin
with opcommonitor do
  begin
  wait(mutex);
  opmonlevel:=adlevel;
  signal(mutex);
  end;{with}
end;
```

```
{opcommonitor} procedure setopmonlevel(level:leveltype);

{Tells the opcommonitor the new range of the AD-converters}

begin
with opcommonitor do
  begin
  wait(mutex);
  adlevel:=level;
  signal(mutex);
  end;{with}
end;


{opcommonitor} procedure logfinished;

{Tells the opcommonitor that the logging has ended}

begin
with opcommonitor do
  begin
  wait(mutex);
  logflag:=false;
  runlog:=false;
  signal(mutex);
  end;{with}
end;{logfinished}


{opcommonitor} procedure removenames(mode:outmodetype);

{Removes nodenames from the nodename list in the
 opcommonitor}

var ptr:nodeelemptr;

begin
with opcommonitor do
  begin
  wait(mutex);
  ptr:=nodelist^.fwd;
  while ptr <> nodelist do
    begin
    if ptr^.flags[mode] then
      begin
      ptr^.flags[mode]:=false;
      if (not ptr^.flags[log]) and (not ptr^.flags[print])
and
         (not ptr^.flags[plot]) then ptr^.nodename:=blanks;
      end;
    ptr:=ptr^.fwd;
    end;{while}
  signal(mutex);
  end;{with}
end;{removenames}
```

```
{opcommonitor} procedure setname(nodename:identtype;
                                 mode:outmodetype);

{Inserts a nodename in the nodename list}

var ptr:nodeelemptr;

begin
with opcommonitor do
  begin
  wait(mutex);
  ptr:=nodelist^.fwd;
  while (ptr <> nodelist) and (ptr^.nodename <> nodename) do
    ptr:=ptr^.fwd;
  if ptr <> nodelist then ptr^.flags[mode]:=true
  else
    begin
    ptr:=nodelist^.fwd;
    while ptr^.nodename <> blanks do ptr:=ptr^.fwd;
    ptr^.nodename:=nodename;
    ptr^.flags[mode]:=true;
    end;
  signal(mutex);
  end;{with}
end;{setname}




{opcommonitor} procedure initopcommonitor;

{Initiates the opcommonitor}

var  ptr:nodeelemptr;
     i:integer;
     outindex:outmodetype;

begin
with opcommonitor do
  begin
  initsem(mutex,1);
  initevent(start,mutex);
  logflag:=false;     runlog:=false;
  plotflag:=false;    printflag:=false;
  extfile:=false;
  new(nodelist);
  ptr:=nodelist;
  ptr^.nodename:=blanks;
  for outindex:=log to plot do
    ptr^.flags[outindex]:=false;
  for i:=1 to nodelistsize do
    begin
```

```
    new(ptr^.fwd);
    with ptr^ do
      begin
      fwd^.bwd:=ptr;
      fwd^.nodename:=blanks;
      for outindex:=log to plot do
        fwd^.flags[outindex]:=false;
      end;{with}
    ptr:=ptr^.fwd;
    end;{for}
  nodelist^.bwd:=ptr;
  ptr^.fwd:=nodelist;
  adlevel:=bipolarlev;
  end;{with}
end;{initopcommonitor}
```

```
procedure error(err:errors);

{Writes out an error message on the terminal}

begin
case err of
  toomanyarg   : write('Too many arguments ');
  toolongid    : write('Too long identifier');
  notin        : write('Illegal insignal');
  fewarg       : write('Missing arguments');
  statnode     : write('Statistic signal, not valid');
  illtype      : write('Illegal nodetype');
  illname      : write('Illegal name');
  noop         : write('No operation ');
  neggd        : write('Gd should be > 0 when td is > 0');
  badlimit     : write('Bad limit on the signal');
  toomanyadd   : write('You can only add 4 signals');
  illpar       : write('Illegal parameter');
  noass        : write('= missing');
  logfile      : write('You alredy have a new logfile');
  changeerr    : write('You have no logfile to change to');
  noslash      : write('/ missing');
  nodevice     : write('Illegal device: ');
  illstat      : write('Invalid extension: ');
  nyquist      : write('Bad time constant');
  nolevel      : write('Illegal AD-DA level ');
  end;{case}
writeln;
end;{error}
```

```
procedure checkresult(var result:resulttype;var err:boolean);

{Writes out an error text}

begin
if result.code <> ok then
  begin
  case result.code of
    nonode       : write(result.name,'doesn''t exist');
    active       : write(result.name,'exists already');
    nospace      : write('The nodelist is full');
    ahead        : write(result.name,'is ahead');
    referr       : write(result.name,'uses this node');
    logging      : write('You are logging ',result.name);
    plotting     : write('You are plotting ',result.name);
    printing     : write('You are printing ',result.name);
    nolog        : write('You are not logging');
    runlogresult: write('You have already done RUNLOG ');
    logresult    : write('You are logging');
    noplot       : write('you are not plotting');
    noprint      : write('You are not printing');
    printresult  : write('You are printing');
    plotresult   : write('You are plotting');
    end;{case}
  writeln;
  err:=true;
  end
else err:=false;
end;{checkresult}
```

```
function endofline(mode:sourcetype):boolean;

{Similar to eoln}

begin
if mode = term then endofline:=eoln
else endofline:=eoln(comfile);
end;{endofline}


procedure readchar(var ch:char;mode:sourcetype);

{Similar to read(char)}

begin
if mode = term then read(ch)
else
  begin
  read(comfile,ch);
  if eoln(comfile) then writeln(ch)
  else write(ch);
  end;
end;{readchar}


procedure readint(var int:integer;mode:sourcetype);

{Similar to read(integer)}

begin
if mode = term then read(int)
else
  begin
  read(comfile,int);
  if eoln(comfile) then writeln(int:3)
  else write(int:3);
  end;
end;{readint}


procedure readrel(var rel:real;mode:sourcetype);

{Similat to read(real)}

begin
if mode = term then read(rel)
else
  begin
  read(comfile,rel);
  if eoln(comfile) then writeln(rel:6:2)
  else write(rel:6:2);
  end;
end;{readrel}
```

```
procedure readnewline(mode:sourcetype);

{Similar to readln}

begin
if mode = term then readln
else readln(comfile);
end;{readnewline}


procedure relreadnew(var rel:real;mode:sourcetype);

{Similar to readln(real)}

begin
if mode = term then readln(rel)
else
  begin
  readln(comfile,rel);
  if eoln(comfile) then writeln(rel:6:2)
  else  write(rel:6:2);
  end;
end;{relreadnew}


procedure intreadnew(var int:integer;mode:sourcetype);

{Similar to readln(integer)}

begin
if mode = term then readln(int)
else
  begin
  readln(comfile,int);
  if eoln(comfile) then writeln(int:3)
  else write(int:3);
  end;
end;{intreadnew}
```

```
procedure skipblanks(mode:sourcetype);

{Skips blanks in a line}

begin
while (not endofline(mode)) and (ch[mode] = ' ') do
  readchar(ch[mode],mode);
end;{skipblanks}



procedure checkend(var err:boolean;mode:sourcetype);

{Checks if there is more arguments in the line}

begin
err:=false;
skipblanks(mode);
if ch[mode] <> ' ' then begin error(toomanyarg); err:=true; end
;
end;{checkend}



procedure getident(var err:boolean;mode:sourcetype);

{Reads characters}

label 999;
var i:integer;

function upper(ch:char):char;
{Converts to upper case letters}
  begin
  if (ch >= 'a') and (ch <= 'z') then
    upper:=chr(ord(ch)-32)
  else upper:=ch;
  end;{upper}


begin {getident}
err:=false;
identifier[mode]:=blanks;
skipblanks(mode);
i:=1;
ch[mode]:=upper(ch[mode]);
while ((ch[mode] >= 'A') and (ch[mode] <= 'Z')) or
((ch[mode] >= '0') and (ch[mode] <= '9')) or (ch[mode] = ':') or
      (ch[mode] = ',') do
  begin
  if i > idlength then begin error(toolongid); err:=true; goto
999; end;
```

```
    identifier[mode,i]:=ch[mode];
    i:=i+1;
    if endofline(mode) then ch[mode]:=' '
    else
      begin
      readchar(ch[mode],mode);
      ch[mode]:=upper(ch[mode]);
      end;
  end;{while}
999:
end;{getident}


procedure pidlink(var err:boolean);

{Computes the internal parameters in a PID node}

label 999;

begin
err:=false;
with node do
  begin
  if (td > 0.0) and (gd <= 0.0) then begin error(neggd);
  err:=true; goto 999; end;
  if minu >= maxu then begin error(badlimit);
  err:=true; goto 999; end;
  if ti <= 0.0 then alfa:=0.0
  else alfa:=k*mainperiod/ti;
  if td <= 0.0 then
    begin
    beta:=0.0;
    gamma:=0.0;
    end
  else
    begin
    beta:=1.0/(1.0+gd*mainperiod/td);
    gamma:=k*beta*(1.0-beta)/mainperiod;
    end;
  end;{with}
999:
end;{pidlink}


procedure readdescrinp(var inpdescr:inputdescriptortype;
var err:boolean;mode:sourcetype);

{Reads an input description}

label 999;

var  index:nodetype;
     bool:boolean;
     i:integer;
```

```
begin
err:=false;
getident(bool,mode);
if bool then begin err:=true; goto 999; end;
with inpdescr do
  begin
  if identifier[mode] = blanks then begin error(notin); err:=true;
goto 999;
  end;
  i:=0;
  while (identifier[mode] <> inputs[i]) and (i < 15) do i:=i+1;
  if identifier[mode] = inputs[i] then
    begin
    intag:=adsig;
    chan:=i;
    end
  else
    if identifier[mode] = konst then
      begin
      intag:=constsig;
      readrel(value,mode);
      end
    else
      begin
      intag:=nodesig;
      name:=identifier[mode];
      monname(name,result[mode],index);
      checkresult(result[mode],bool);
      if bool then begin err:=true; goto 999; end;
      if index = stat then begin error(statnode); err:=true; goto
 999; end;
      end;
  end;{with}
999:
end;{readdescrinp}



procedure writeclock(sek:real);

{Writes out the time in hours:minutes:seconds}

var  hours,min,second:integer;

begin
if sek > 0 then
  begin
  hours:=trunc(sek/3600.0);
  min:=trunc((sek-hours*3600.0)/60.0);
  second:=trunc(sek-hours*3600.0-min*60.0);
  write(hours:3,':',min:2,':',second:2);
  end
else write(0:2,':',0:2,':',0:2);
end;{writeclock}
```

```
procedure breakrut;

{Handles the command BREAK}

var mess:messageref;

begin
receivemessage(infopoolbox,mess);
mess^.info^.infotype:=changemess;
sendmessage(extbox,mess);
setcomfile(false);
end;{breakrut}


procedure commandfilestart;

{Handles the command COMFILE filename }

label 999;

var mess:messageref;
    err:boolean;

begin
if eoln then
  begin
  write('Filename: ');
  readln;
  ch[term]:=' ';
  end;
getident(err,term);
if err then goto 999;
setcomfile(true);
receivemessage(infopoolbox,mess);
mess^.info^.infotype:=newfile;
mess^.info^.newlogfile:=identifier[term];
sendmessage(extbox,mess);
999:
end;


procedure waitroutine;

{Handles the command WAIT hours minutes seconds }

label 999;

var mess:messageref;

begin
if eoln(comfile) then goto 999;
```

```
receivemessage(infopoolbox,mess);
with mess^.info^ do
  begin
  infotype:=waitmess;
  readint(hr,fil);
  readint(minu,fil);
  readint(sek,fil);
  end;
sendmessage(waitbox,mess);
waitstart;
999:
end;


procedure filtercount(var node:nodepart);

{Computes the filter constants in the third order
 Butterworth filter}

var wg,wg2,wg3,fnorm:real;

begin
with node do
  begin
  fnorm:=mainperiod/timeconst;
  wg:=2*(sin(pi*fnorm)/cos(pi*fnorm));
  wg2:=wg*wg;
  wg3:=wg2*wg;
  a0:=8+8*wg+4*wg2+wg3;
  a1:=(4*wg2+3*wg3-8*wg-24)/a0;
  a2:=(24-8*wg-4*wg2+3*wg3)/a0;
  a3:=(8*wg-8-4*wg2+wg3)/a0;
  b0:=wg3/a0;
  b1:=3*wg3/a0;
  b2:=b1;
  b3:=b0;
  a0:=1;
  u1:=0;
  u2:=0;
  u3:=0;
  y1:=0;
  y2:=0;
  y3:=0;
  end;
end;


type  unsignedinteger = 0..65535;
      semaphore = unsignedinteger;
      event = unsignedinteger;
      messageref = ^message;
      message = record
        nextmess:messageref;
        end;
      mailbox = ^mail;
```

```
      mail = record
        firstmess : messageref;
        mutex:semaphore;
        send:event;
        end;

procedure wait(sem:semaphore);external;
procedure signal(sem:semaphore);external;

function messageaccept(box:mailbox;var mess:messageref):boolean;


{Exactly as the kernel's acceptmessage}

begin
with box^ do
  begin
  wait(mutex);
  if firstmess = nil then
    begin
    messageaccept:=false;
    mess:=nil;
    end
  else
    begin
    messageaccept:=true;
    mess:=firstmess;
    firstmess:=firstmess^.nextmess;
    end;
  signal(mutex);
  end;{with}
end;{messageaccept}
```

```
procedure newnode(mode:sourcetype);

{Handles the command NEW  type   [name]  }

label 999;

var  typeindex:nodetype;
     err:boolean;
     statindex:stattype;
     invalue:real;
     i:integer;
begin
if endofline(mode) then
  begin
  write('Type: ');
  readnewline(mode);
  ch[mode]:=' ';
  end;
getident(err,mode);
if err then goto 999;
typeindex:=pid;
while (typename[typeindex] <> identifier[mode]) and
      (typeindex <> lasttype) do typeindex:=succ(typeindex);
if typeindex = lasttype then begin error(illtype); goto 999; end;
node.tag:=typeindex;
if endofline(mode) then node.name:=identifier[mode]
else
  begin
  getident(err,mode);
  if err then goto 999;
  if  (identifier[mode] = blanks)
  then begin error(illname); goto 999; end;
  for i:=0 to 15 do
    if identifier[mode] = inputs[i] then
      begin
      error(illname);
      goto 999;
      end;
  node.name:=identifier[mode];
  end;
moncheck(node.name,result[mode]);
checkresult(result[mode],err);
if err then goto 999;
with node do
  begin
  case tag of
    pid       : begin
                write('Pidin: ');
                readnewline(mode);
                ch[mode]:=' ';
                readdescrinp(pidinpdescr,err,mode);
                if err then goto 999;
                write('Pidref: ');
                readnewline(mode);
                ch[mode]:=' ';
```

```
            readdescrinp(pidrefdescr,err,mode);
            if err then goto 999;
            write('Outchannel: ');
            intreadnew(outchan,mode);
            write('K: ');
            relreadnew(k,mode);
            write('Ti: ');
            relreadnew(ti,mode);
            write('Td: ');
            relreadnew(td,mode);
            write('Gd: ');
            relreadnew(gd,mode);
            write('Minu: ');
            relreadnew(invalue,mode);
            minu:=invalue/10.0;
            write('Maxu: ');
            readrel(invalue,mode);
            maxu:=invalue/10.0;
            pidlink(err);
            if err then goto 999;
            yold:=0.0;
            dpart:=0.0;
            ipart:=0.0;
            end;
puls,triang,sinus,sawtooth,prbs
          : begin
            write('Pk-PK Amplitude: ');
            relreadnew(invalue,mode);
            ptpamp:=invalue/10.0;
            if tag <> prbs then
            begin
            write('Period: ');
            relreadnew(period,mode);
            end
            else
            begin
            write('Sample period: ');
            relreadnew(invalue,mode);
            timervalue:=round(invalue/mainperiod);
            timer:=timervalue;
            state:=1;
            end;
            write('Meanvalue: ');
            readrel(invalue,mode);
            meanvalue:=invalue/10.0;
            write('Outchannel: ');
            readnewline(mode);
            if endofline(mode) then outchan:=-1
            else readint(outchan,mode);
            if tag <> prbs then inttime:=0.0;
            if tag = puls then dutycycle:=0.5;
            end;
stat      : begin
            write('Insignal: ');
            readnewline(mode);
```

```
                ch[mode]:=' ';
                readdescrinp(inpdescr,err,mode);
                if err then goto 999;
                n:=0;
                sum:=0;
                sum2:=0;
                for statindex:=mean to maxy do
                  statoutchan[statindex]:=-1;
                statoutvalue[miny]:=1.0;
                statoutvalue[maxy]:=-1.0;
                end;
    add       : begin
                write('Nrofinsignals: ');
                intreadnew(nrofinsignals,mode);
                if nrofinsignals > 4 then
                  begin
                  error(toomanyadd);
                  goto 999;
                  end;
                for i:=1 to nrofinsignals do
                  begin
                  write('Insignal ',i:1,': ');
                  ch[mode]:=' ';
                  readdescrinp(addinput[i],err,mode);
                  if err then goto 999;
                  write('Weighting factor ',i:1,': ');
                  relreadnew(addfactor[i],mode);
                  end;
                write('Outchannel : ');
                readint(outchan,mode);
                end;

    filter    : begin
                write('Insignal: ');
                readnewline(mode);
                ch[mode]:=' ';
                readdescrinp(filtinput,err,mode);
                if err then goto 999;
                write('Time constant: ');
                relreadnew(timeconst,mode);
                if timeconst < (2*mainperiod) then
                  begin
                  error(nyquist);
                  goto 999;
                  end
                else filtercount(node);
                write('Outchannel: ');
                readint(outchan,mode);
                end;
    end;{case}
  end;{with}
  monnewnode(node);
999:
end;{newnode}
```

```
procedure parnode(mode:sourcetype);

{Handles the command PAR name parameter=value ... }

label 999;

var  parindex:pars;
     err:boolean;
     invalue :real;
     addnr:integer;

begin
if endofline(mode) then
  begin
  write('Name: ');
  readnewline(mode);
  ch[mode]:=' ';
  end;
getident(err,mode);
if err then goto 999;
node.name:=identifier[mode];
mongetnode(node,result[mode]);
checkresult(result[mode],err);
if err then goto 999;
with node do
  begin
  repeat
    getident(err,mode);
    if err then goto 999;
    parindex:=pidin;
    while (parname[parindex] <> identifier[mode]) and
          (parindex <> lastpar) do parindex:=succ(parindex);
    if not (parindex in nodeparam[tag]) then
      begin
      error(illpar);
      goto 999;
      end;
    if (parindex = addinputpar) or (parindex = weight) then
    readint(addnr,mode);
    skipblanks(mode);
    if endofline(mode) or (ch[mode] <> '=') then
      begin
      error(noass);
      goto 999;
      end;
    if endofline(mode) then
      begin
      error(fewarg);
      goto 999;
      end;
    ch[mode]:=' ';
    case parindex of
                   pidin    : begin
                              readdescrinp(pidinpdescr,err,mode);
                              if err then goto 999;
```

```
                if pidinpdescr.intag = nodesig
                then
                  begin
                  monordercheck(pidinpdescr.name,
                                name,result[mode]);
                  checkresult(result[mode],err);
                  if err then goto 999;
                  end;
                yold:=0.0;
                ipart:=0.0;
                dpart:=0.0;
                end;
pidref    :     begin
                readdescrinp(pidrefdescr,err,mode);
                if err then goto 999;
                if pidrefdescr.intag = nodesig
                then
                  begin
                    monordercheck(pidrefdescr.name,
                                  name,result[mode]);
                    checkresult(result[mode],err);
                    if err then goto 999;
                  end;
                yold:=0.0;
                ipart:=0.0;
                dpart:=0.0;
                end;
outchanpar  :   readint(outchan,mode);
kpar        :   readrel(k,mode);
tipar       :   readrel(ti,mode);
tdpar       :   readrel(td,mode);
gdpar       :   readrel(gd,mode);
minupar     :   readrel(minu,mode);
maxupar     :   readrel(maxu,mode);
ptpamppar   :   begin;
                readrel(invalue,mode);
                ptpamp:=invalue/10.0;
                end;
periodpar   :   readrel(period,mode);
meanvaluepar:   begin
                readrel(invalue,mode);
                meanvalue:=invalue/10.0;
                end;
dutycyclepar:   begin
                readrel(invalue,mode);
                dutycycle:=invalue/100.0;
                end;
input       :   begin
                if tag = stat then
                readdescrinp(inpdescr,err,mode)
                else readdescrinp(filtinput,err,mode);
                if err then goto 999;
                if tag = stat then
                if inpdescr.intag = nodesig
                then
```

```
                    begin
                    monordercheck(inpdescr.name,
                              name,result[mode]);
                    checkresult(result[mode],err);
                    if err then goto 999;
                    end;
                  if tag = filter then
                  if filtinput.intag = nodesig then
                    begin
                    monordercheck(filtinput.name,
                               name,result[mode]);
                    checkresult(result[mode],err);
                    if err then goto 999;
                    end;
                  if tag = filter then
                    filtercount(node);
                  if tag = stat then
                    begin
                    n:=0;
                    sum:=0;
                    sum2:=0;
                    statoutvalue[miny]:=1.0;
                    statoutvalue[maxy]:=-1.0;
                    end;
                  end;
outchanmean   : readint(statoutchan[mean],mode);
outchanvar    : readint(statoutchan[vari],mode);
outchanminy   : readint(statoutchan[miny],mode);
outchanmaxy   : readint(statoutchan[maxy],mode);
sampleperiodpar: begin
                   readrel(invalue,mode);
                   timervalue:=round(invalue
                                /mainperiod);
                   end;
nrinsignalspar :readint(nrofinsignals,mode);
weight         : readrel(addfactor[addnr],mode);
addinputpar    : begin
                 readdescrinp(addinput[addnr],
                              err,mode);
                 if err then goto 999;
                 if addinput[addnr].intag = nodesig
                 then
                   begin
                   monordercheck(addinput[addnr].name,
                   name,result[mode]);
                   checkresult(result[mode],err);
                   if err then goto 999;
                   end;
                 end;
timecon        : begin
                 readrel(timeconst,mode);
                 if timeconst < (2*mainperiod) then
                   begin
                   error(nyquist);
                   goto 999;
```

```
                                    end
                                  else filtercount(node);
                                  end;
      end;{case}
    if (parindex = kpar) or (parindex = tipar) or
    (parindex = gdpar) or (parindex = minupar) or
    (parindex = tdpar) or (parindex = maxupar)
    then begin pidlink(err); if err then goto 999; end;
    skipblanks(mode);
    until endofline(mode);
  end;{with}
monsetnode(node);
999:
end;{parnode}
```

```
procedure dispnode(mode:sourcetype);

{Handles the command DISP name}

label 999;

var index:stattype;
    err:boolean;
    i:integer;

  procedure writeinpdescr(inpdescr:inputdescriptortype);

  begin
  with inpdescr do
    begin
    case intag of
      adsig    : write('AD',chan:1);
      nodesig  : write(name);
      constsig : write('CONST ',value:6:3);
      end;{case}
    end;{with}
  end;{writeinpdescr}


begin
if endofline(mode) then
  begin
  write('Name: ');
  readnewline(mode);
  ch[mode]:=' ';
  end;
getident(err,mode);
if err then goto 999;
node.name:=identifier[mode];
mongetnode(node,result[mode]);
checkresult(result[mode],err);
if err then goto 999;
writeln('Nodetype: ',typename[node.tag]);
with node do
  begin
  case tag of
    pid       : begin
                write('Pidin: ');
                writeinpdescr(pidinpdescr);
                writeln;
                write('Pidref: ');
                writeinpdescr(pidrefdescr);
                writeln;
                if outchan >= 0 then writeln('Outchannel: ',
                                             outchan:2);
                writeln('K: ',k:6:3);
                writeln('Ti: ',ti:6:3);
                writeln('Td: ',td:6:3);
                writeln('Gd: ',gd:6:3);
                writeln('Minu: ',(minu*10.0):6:3);
```

```
                writeln('Maxu: ',(maxu*10.0):6:3);
                writeln('Yold: ',yold:6:3);
                writeln('Ipart: ',ipart:6:3);
                writeln('Dpart: ',dpart:6:3);
                end;{pid}
  puls,triang,sinus,sawtooth,prbs
              : begin
                if outchan >= 0
                then writeln('Outchannel: ',outchan:2);
                writeln('Peak-Peak Amplitude: ',(ptpamp*10.0):6:3);
                if tag <> prbs then writeln('Period: ',period:6:3)
                else writeln('Sample period: ',
                             (timervalue*mainperiod):6:3);
                writeln('Meanvalue: ',(meanvalue*10.0):6:3);
                if tag = puls then
                  writeln('Dutycycle: ',(dutycycle*100.0):4:1,'%');
                end;
  stat        : begin
                write('Input: ');
                writeinpdescr(inpdescr);
                writeln;
                writeln('Mean value: ',statoutvalue[mean]:6:3);
                writeln('Variance  : ',statoutvalue[vari]:6:3);
                writeln('Lowest value: ',statoutvalue[miny]:6:3,
                '  Samplenumber',minysample:6);
                writeln('Highest value: ',statoutvalue[maxy]:6:3,
                '  Samplenumber',maxysample:6);
                for index:=mean to maxy do
                  if statoutchan[index] >= 0 then
                  writeln('Outchannel',statname[index],':',
                            statoutchan[index]:2);
                end;{stat}
  add         : begin
                write('Outvalue = ');
                for i:=1 to nrofinsignals do
                 begin
                 write(addfactor[i]:5:2);
                 write('*');
                 writeinpdescr(addinput[i]);
                 if i <= (nrofinsignals-1) then write('+ ');
                 end;
                writeln;
                if outchan >= 0 then writeln('Outchannel : ',
                                             outchan:2);
                end;
  filter      : begin
                write('Insignal: ');
                writeinpdescr(filtinput);
                writeln;
                writeln('Time constant : ',timeconst:6:2);
                if outchan >= 0 then writeln('Outchannel : ',
                                             outchan:2);
                end;
  end;{case}
end;{with}
```

```
999:
end;{dispnode}


procedure deletenode(mode:sourcetype);

{Handles the command DELETE name}

label 999;

var err:boolean;

begin
if endofline(mode) then
  begin
  write('Name: ');
  readnewline(mode);
  ch[mode]:=' ';
  end;
getident(err,mode);
if err then goto 999;
opcommoncheck(identifier[mode],result[mode]);
checkresult(result[mode],err);
if err then goto 999;
mondelete(identifier[mode],result[mode]);
checkresult(result[mode],err);
999:
end;{deletenode}


procedure listmess(command:opx;mode:sourcetype);

{Handles the command RUNLOG,ENDPLOT,ENDPRINT,ENDLOG}

label 999;

var mess,filemess:messageref;
    err:boolean;

begin
checkcommand(command,result[mode]);
checkresult(result[mode],err);
if err then goto 999;
receivemessage(infopoolbox,mess);
with mess^.info^ do
case command of
  runlogx  : infotype:=runlogmess;
  endplotx : begin
             infotype:=endplotmess;
             removenames(plot);
             end;
  endprintx: begin
             infotype:=endprintmess;
```

```
                removenames(print);
                end;
  endlogx   : begin
                infotype:=endlogmess;
                removenames(log);
                end;
end;{case}
sendmessage(listbox,mess);
999:
end;{listmess}


procedure messlog(mode:sourcetype);

{Handles the command LOGMESS text }

label 999;


var err:boolean;
    mess : messageref;
    i:integer;
    messch:char;

begin
checkcommand(logmessx,result[mode]);
checkresult(result[mode],err);
if err then goto 999;
receivemessage(infopoolbox,mess);
mess^.info^.infotype:=infomess;
for i:=1 to 80 do mess^.info^.string[i]:=' ';
i:=0;
repeat
  i:=i+1;
  readchar(messch,mode);
  mess^.info^.string[i]:=messch;
until endofline(mode);
sendmessage(filebox,mess);
999:
end;{messlog}


procedure newlogfile(mode:sourcetype);

{Handles the command NEWLOGFILE filename }

label 999;


var mess:messageref;
    newfilename,extension:identtype;
    err:boolean;
    time:real;
begin
```

```
checkcommand(newlogfilex,result[mode]);
checkresult(result[mode],err);
if err then goto 999;
extension:=blanks;
if endofline(mode) then
  begin
  write('Filename: ');
  readnewline(mode);
  ch[mode]:=' ';
  end;
getident(err,mode);
if err then goto 999;
newfilename:=identifier[mode];
if ch[mode] = '.' then
  begin
  ch[mode]:=' ';
  getident(err,mode);
  if err then goto 999;
  end;
receivemessage(infopoolbox,mess);
mess^.info^.infotype:=newfile;
mess^.info^.newext:=identifier[mode];
mess^.info^.newlogfile:=newfilename;
sendmessage(filebox,mess);
receivemessage(opcombox,mess);
case mess^.info^.infotype of
  errormess : error(logfile);
  blocksize : begin
              writeln('You can log ');
              time:=logtime(mess^.info^.blocks);
              writeclock(time);
              writeln;
              end;
  end;{case}
sendmessage(infopoolbox,mess);
999:
end;{newlogfile}


procedure change(mode:sourcetype);

{Handles the command CHANGE}

label 999;


var mess:messageref;
    err:boolean;

begin
checkcommand(changex,result[mode]);
checkresult(result[mode],err);
if err then goto 999;
checkend(err,mode);
```

```
if err then goto 999;
receivemessage(infopoolbox,mess);
mess^.info^.infotype:=changemess;
sendmessage(filebox,mess);
receivemessage(opcombox,mess);
case mess^.info^.infotype of
  errormess :  error(changeerr);
  okmess    : begin
              end;
  end;{case}
sendmessage(infopoolbox,mess);
999:
end;{change}


procedure adlevel(mode:sourcetype);

{Handles the command ADRANGE range }

label 999;

var err:boolean;

begin
checkcommand(adlevelx,result[mode]);
checkresult(result[mode],err);
if err then goto 999;
if endofline(mode) then
  begin
  write('Range: ');
  readnewline(mode);
  ch[mode]:=' '
  end;
getident(err,mode);
if err then goto 999;
if identifier[mode] = bipolar then monlevelchange(adbipolar)
else
  if identifier[mode] = unipolar then monlevelchange(adunipolar)
  else  error(nolevel);
if identifier[mode] = bipolar then setopmonlevel(bipolarlev)
else setopmonlevel(unipolarlev);
999:
end;{adlevel}


procedure dalevel(mode:sourcetype);

{Handles the command DARANGE range }

label 999;

var err:boolean;

begin
checkcommand(dalevelx,result[mode]);
```

```
checkresult(result[mode],err);
if err then goto 999;
if endofline(mode) then
  begin
  write('Range: ');
  readnewline(mode);
  ch[mode]:=' ';
  end;
getident(err,mode);
if err then goto 999;
if identifier[mode] = bipolar then monlevelchange(dabipolar)
else
  if identifier[mode] = unipolar then monlevelchange(daunipolar)
  else error(nolevel);
999:
end;{dalevel}


procedure endprogram(mode:sourcetype);

{Handles the command EXIT}

var err:boolean;

begin
checkcommand(endx,result[mode]);
checkresult(result[mode],err);
if not err then signal(endsync);
end;
```

```
procedure outroutine(mode:sourcetype;command:opx);

{Handles the command PRINT[/period] signal ...
and the command PLOT[/period] signal ...      }

label 999;


var  mess:messageref;
     err:boolean;
     nrofprintsignals,chan,mainperiodmultiple:integer;
     statindex:stattype;
     typeindex:nodetype;
     printperiod:real;
     printsignals:array [1..maxoutsig] of signaldescriptortype;
     i:integer;

begin
nrofprintsignals:=0;
receivemessage(infopoolbox,mess);
if command = printx then
  mess^.info^.infotype:=printmess
else mess^.info^.infotype:=plotmess;
if ch[mode] = '/' then
  begin
  if endofline(mode) then
    begin
    writeln('Period Signals ');
    readnewline(mode);
    ch[mode]:=' ';
    end;
  ch[mode]:=' ';
  readrel(printperiod,mode);
  mainperiodmultiple:=round(printperiod/mainperiod);
  if printperiod < mainperiod then mainperiodmultiple:=1;
  if (printperiod/mainperiod) <> mainperiodmultiple then
    if command = printx then writeln('Print period = ',mainperiod*
                                     mainperiodmultiple:6:2)
    else writeln('Plot period = ',mainperiod*mainperiodmultiple:6:2);
  end
else
  begin
  mainperiodmultiple:=1;
  if command = printx then writeln('Print period = ',mainperiod:6:2)
    else writeln('Plot period = ',mainperiod:6:2);
  end;
if endofline(mode) then
  begin
  writeln('Signals ');
  readnewline(mode);
  ch[mode]:=' ';
  end;
while (not endofline(mode)) and (nrofprintsignals <= maxprintsig) do
  begin
  getident(err,mode);
```

```
if err then begin sendmessage(infopoolbox,mess); goto 999; end;
nrofprintsignals:=nrofprintsignals+1;
if identifier[mode] = blanks then
  begin
  error(notin);
  sendmessage(infopoolbox,mess);
  goto 999;
  end;
i:=0;
while (identifier[mode] <> inputs[i]) and (i < 15) do i:=i+1;
if identifier[mode] = inputs[i] then
  begin
  printsignals[nrofprintsignals].sigtag:=adsig;
  printsignals[nrofprintsignals].chan:=i;
  end
else
  if identifier[mode] = konst then
    begin
    printsignals[nrofprintsignals].sigtag:=constsig;
    if endofline(mode) then
      begin
      error(fewarg);
      sendmessage(infopoolbox,mess);
      goto 999;
      end;
    readrel(printsignals[nrofprintsignals].value,mode);
    end
  else
    begin
    monname(identifier[mode],result[mode],typeindex);
    checkresult(result[mode],err);
    if err then begin sendmessage(infopoolbox,mess); goto 999; end;
    printsignals[nrofprintsignals].sigtag:=nodesig;
    printsignals[nrofprintsignals].name:=identifier[mode];
    if command = printx then
      setname(identifier[mode],print)
    else setname(identifier[mode],plot);
    if typeindex = stat then
      begin
      if ch[mode] <> '/' then begin error(noslash);
      sendmessage(infopoolbox,mess); goto 999; end;
      if endofline(mode) then begin error(fewarg);
      sendmessage(infopoolbox,mess); goto 999; end;
      ch[mode]:=' ';
      getident(err,mode);
      if err then begin sendmessage(infopoolbox,mess);
      goto 999; end;
      statindex:=mean;
      while (statname[statindex] <> identifier[mode]) and
            (statindex <> laststat) do statindex:=succ(statindex);
      if statindex = laststat then begin error(illstat);
      sendmessage(infopoolbox,mess); goto 999; end;
      printsignals[nrofprintsignals].ext:=statindex;
      end;
    end;{if}
```

```
  end;{while}
if nrofprintsignals > maxprintsig then begin error(toomanyarg);
sendmessage(infopoolbox,mess); goto 999; end;
with mess^.info^ do
  begin
  nprintsignals:=nrofprintsignals;
  messprintsignals:=printsignals;
  printlevel:=opmonlevel;
  printsampleperiod:=mainperiodmultiple;
  end;{with}
checkcommand(command,result[mode]);
checkresult(result[mode],err);
if err then sendmessage(infopoolbox,mess)
else sendmessage(listbox,mess);
999:
end;{outroutine}
```

```
procedure logroutine(mode:sourcetype);

{Handles the command LOG[/log period] filename signal ... }

label 999;


var mess:messageref;
    nroflogsignals,mainperiodmultiple:integer;
    err:boolean;
    logsignals:array [1..maxoutsig] of signaldescriptortype;
    time,logperiod:real;
    chan:integer;
    typeindex:nodetype;
    signalnames:array [1..8] of identtype;
    statindex:stattype;
    i:integer;
    logname,extension:identtype;

begin
for i:=1 to 8 do signalnames[i]:=blanks;
extension:=blanks;
nroflogsignals:=0;
if ch[mode] = '/' then
  begin
  if endofline(mode) then
    begin
    write('Log period  Filename  Signal ...');
    readnewline(mode);
    end;
  readrel(logperiod,mode);
  mainperiodmultiple:=round(logperiod/mainperiod);
  if logperiod < mainperiod then mainperiodmultiple:=1;
  if (logperiod/mainperiod) <> mainperiodmultiple then
    writeln('Log period = ',mainperiod*mainperiodmultiple:6:2);
  end
else
  begin
  mainperiodmultiple:=1;
  writeln('Log period = ',mainperiod:6:2);
  end;
if endofline(mode) then
  begin
  write('Filename Signal ...');
  readnewline(mode);
  end;
ch[mode]:=' ';
getident(err,mode);
if err then goto 999;
logname:=identifier[mode];
if ch[mode] = '.' then
  begin
  ch[mode]:=' ';
  getident(err,mode);
  if err then goto 999;
```

```
  extension:=identifier[mode];
  end;
if endofline(mode) then
  begin
  write('Signal ... ');
  readnewline(mode);
  ch[mode]:=' ';
  end;
while (not endofline(mode)) and (nroflogsignals < maxlogsig) do
  begin
  nroflogsignals:=nroflogsignals+1;
  ch[mode]:=' ';
  getident(err,mode);
  if err then goto 999;
  if identifier[mode] = blanks then
    begin
    error(notin);
    goto 999;
    end;
  i:=0;
  while (identifier[mode] <> inputs[i]) and (i < 15) do i:=i+1;
  if identifier[mode] = inputs[i] then
    begin
    logsignals[nroflogsignals].sigtag:=adsig;
    logsignals[nroflogsignals].chan:=i;
    signalnames[nroflogsignals]:=identifier[mode];
    end
  else
    if identifier[mode] = konst then
      begin
      if endofline(mode) then
        begin
        error(fewarg);
        goto 999;
        end;
      logsignals[nroflogsignals].sigtag:=constsig;
      readrel(logsignals[nroflogsignals].value,mode);
      signalnames[nroflogsignals]:=konst;
      end
    else
      begin
      monname(identifier[mode],result[mode],typeindex);
      checkresult(result[mode],err);
      if err then goto 999;
      logsignals[nroflogsignals].sigtag:=nodesig;
      logsignals[nroflogsignals].name:=identifier[mode];
      setname(identifier[mode],log);
      signalnames[nroflogsignals]:=identifier[mode];
      if typeindex = stat then
        begin
        if ch[mode] <> '/' then begin error(noslash); goto 999; end;
        if endofline(mode) then begin error(fewarg); goto 999; end;
        ch[mode]:=' ';
        getident(err,mode);
        if err then goto 999;
```

```
          statindex:=mean;
          while (statname[statindex] <> identifier[mode]) and
                (statindex <> laststat) do statindex:=succ(statindex);
          if statindex = laststat then
            begin
            error(illstat);
            goto 999;
            end;
          logsignals[nroflogsignals].ext:=statindex;
          end;
        end;
    end;{while}
if (nroflogsignals = maxlogsig) and (not endofline(mode)) then
    begin
    error(toomanyarg);
    goto 999;
    end;
receivemessage(infopoolbox,mess);
with mess^.info^ do
    begin
    infotype:=startlog;
    logfile:=logname;
    ext:=extension;
    signalorder:=signalnames;
    nsignals:=nroflogsignals;
    level:=opmonlevel;
    logperiod:=mainperiod*mainperiodmultiple;
    end;{with}
checkcommand(logx,result[mode]);
checkresult(result[mode],err);
if err then begin sendmessage(infopoolbox,mess); goto 999; end
else sendmessage(filebox,mess);
logstart(nroflogsignals,mainperiodmultiple);
receivemessage(opcombox,mess);
write('You can log ');
time:=logtime(mess^.info^.blocks);
writeclock(time);
with mess^.info^ do
    begin
    infotype:=logmess;
    messlogsignals:=logsignals;
    sampleperiod:=mainperiodmultiple;
    loglevel:=opmonlevel;
    nlogsignals:=nroflogsignals;
    end;{with}
sendmessage(listbox,mess);
999:
end;{logroutine}
```

```
procedure helpnew;

begin
writeln('Syntax : NEW type [name] ');
writeln;
write('Creates a new module of type TYPE ');
writeln('with the name NAME (default TYPE)');
writeln('PID - PID regulator');
writeln('PULSE - Squarewave generator');
writeln('TRIANGLE - Trianglewave generator');
writeln('SAWTOOTH - Sawtooth generator');
writeln('SINE  - Sine-wave generator');
writeln('PRBS - Prbs generator');
write('STAT - Computes mean, variance, ');
writeln('highest and lowest sample value');
writeln('FILTER - third order LP Butterworth filter');
writeln('ADD - Computes a mathematic expression of signals');
end;


procedure helppar;

begin
writeln('Syntax : PAR name parameter=value ... ');
writeln;
writeln('Changes the parameters of a module.');
end;

procedure helpdisp;
begin
writeln('Syntax : DISP name ');
writeln;
writeln('Displays the parameters of the module name ');
end;

procedure helpdel;
begin
writeln('Syntax : DELETE name ');
writeln;
write('Removes the module name if it is not ');
writeln('plotted, printed or logged.');
end;


procedure helprunlog;
begin
writeln('Syntax : RUNLOG');
writeln;
writeln('Starts the logging. The logging is stopped with ENDLOG.');
end;

procedure helpmesslog;
begin
writeln('Syntax : LOGMESS Text to be written ');
```

```
writeln;
writeln('Writes messages into the file log file.INF.');
writeln('Accepts only one line a time.');
end;


procedure logendhelp;

begin
writeln('Syntax : ENDLOG');
writeln;
writeln('Stops the logging');
end;

procedure helplog;

begin
writeln('Syntax : LOG[/logperiod] filename  signal ... ');
writeln;
write('Opens the the log file filename and an ');
writeln('information file ');
write('filename.INF. Default log file extension is .DAT. ');
writeln(' Default log');
write('period is main period. The program returns the ');
writeln('time you can log');
write('in  (hr.min.sec) before the file is full. To start ');
writeln('the logging');
write('give command RUNLOG. The logging is stopped ');
writeln('with ENDLOG.');
end;

procedure henewfile;

begin
writeln('Syntax : NEWLOGFILE  filename ');
writeln;
write('Opens a file where the logging continues after the ');
writeln('initial log file');
write('is full. Returns the time you can log (hr.min.sec) ');
writeln('in your new file.');
end;


procedure helpchange;

begin
writeln('Syntax : CHANGE ');
writeln;
write('Changes log file from the initial file to the new ');
writeln('log file. Must be');
writeln('preceeded by the command NEWLOGFILE.');
end;
```

```
procedure helpplot;
begin
writeln('Syntax : PLOT[/plotperiod] signal ... ');
writeln;
write('Starts the plotting of signals on the terminal. ');
writeln('Default plot period');
write('is main period. The signals are plotted with the ');
writeln('characters X,O,|,-');
writeln('sequentially. The plotting is stopped with ENDPLOT.');
end;

procedure plotendhelp;
begin
writeln('Syntax : ENDPLOT');
writeln;
writeln('Stops the plotting.');
end;



procedure helpcomfile;

begin
writeln('Syntax : COMFILE filename ');
writeln;
writeln('Starts the interpretion of the commandfile filename.COM.');
writeln('The interpretion is stopped with BREAK. ');
end;

procedure helpbreak;

begin
writeln('Syntax : BREAK');
writeln;
writeln('Breaks the interpretion of the commandfile');
end;

procedure helpend;
begin
writeln('Syntax : EXIT');
writeln;
writeln('Returns the control to RT-11');
end;

procedure helpadlevel;

begin
writeln('Syntax : ADRANGE range');
writeln;
write('Possible ranges are BIPOLAR and UNIPOLAR. BIPOLAR ');
writeln('changes the range');
write('of all the ad-converters to -10 - +10 volt. UNIPOLAR ');
writeln('changes the');
writeln('range to 0 - +10 volt.');
end;
```

```
procedure helpprint;
begin
writeln('Syntax : PRINT[/printperiod] signal ... ');
writeln;
write('Starts the printing of signals on the terminal. ');
writeln('Default print period');
writeln('is main period.The printing is stopped with ENDPRINT.');
end;

procedure printendhelp;
begin
writeln('Syntax : ENDPRINT');
writeln;
writeln('Stops the printing.');
end;

procedure helpwait;

begin
writeln('Syntax : WAIT hr min sec');
writeln;
write('Holds the interpretion of the command file the time ');
writeln('hr.min.sec.');
writeln('Only valid as external command');
end;

procedure helpdalevel;

begin
writeln('Syntax : DARANGE range');
writeln;
write('Possible ranges are BIPOLAR and UNIPOLAR. BIPOLAR ');
writeln('changes the range');
write('of all the da-converters to -10 - +10 volt. UNIPOLAR ');
writeln('changes the');
writeln('range to 0 - +10 volt.');
end;

procedure helplast;
begin
writeln('This is not a command');
end;

procedure helphelp;
begin
writeln('Syntax : HELP [command]');
writeln;
writeln('Lists a help text ,of which this is a part , for ');
writeln('each command');
end;
```

```
procedure helproutine(mode:sourcetype);

{Handles the command HELP [command]}

label 999;

var command:opx;
    err:boolean;


procedure writehelp;
begin
writeln('The commands of the logger are ');
write('NEW,PAR,DISP,DELETE,LOG,RUNLOG,');
writeln('ENDLOG,CHANGE,NEWLOGFILE,PLOT,');
write('ENDPLOT,PRINT,ENDPRINT,WAIT,');
writeln('COMFILE,BREAK,ADRANGE,DARANGE,EXIT.');
writeln('For further information write HELP command ');
end;

begin
if endofline(mode) then writehelp
else
  begin
  command:=newx;
  getident(err,mode);
  if err then goto 999;
  writeln;
  while (opname[command] <> identifier[mode]) and
        (command <> lastopindex) do command:=succ(command);
  case command of
     newx         : helpnew;
     parx         : helppar;
     dispx        : helpdisp;
     deletex      : helpdel;
     runlogx      : helprunlog;
     logmessx     : helpmesslog;
     endlogx      : logendhelp;
     logx         : helplog;
     plotx        : helpplot;
     endplotx     : plotendhelp;
     printx       : helpprint;
     endprintx    : printendhelp;
     newlogfilex  : henewfile;
     changex      : helpchange;
     waitx        : helpwait;
     comfilex     : helpcomfile;
     breakx       : helpbreak;
     adlevelx     : helpadlevel;
     dalevelx     : helpdalevel;
     endx         : helpend;
     helpx        : helphelp;
     lastopindex  : helplast;
     end;{case}
```

```
  end;
999:
end;{helproutine}
```

```
{***************
* LISTPROCESS *
***************}

{process} procedure listprocess;


var  mess,filemess,listmess:messageref;
     bool:array[outmodetype] of boolean;
     timers,timervalues:array[outmodetype] of integer;
     k,i,col:integer;
     signals:outsignaltype;
     index:outmodetype;
     plotvalue:real;
     plotchar:array [1..4] of char;
     listlevel:leveltype;


begin
{$A-}
setpriority(3);
with signals do
  for index:=log to plot do
  begin
  tags[index]:=false;
  bool[index]:=false;
  end;
plotchar[1]:='X';
plotchar[2]:='O';
plotchar[3]:='|';
plotchar[4]:='-';
listlevel:=bipolarlev;
while true do
  begin
  for index:=log to plot do
    if bool[index] then
      begin
      timers[index]:=timers[index]-1;
      if timers[index] = 0 then
        begin
        timers[index]:=timervalues[index];
        signals.tags[index]:=true;
        end;
      end;
  interprete(signals);
  with signals do
    begin
    if tags[log] then
      begin
      if (256-k) < nroutsignals[log] then
        begin
        for i:=k+1 to 256 do mess^.data^[i]:=9999;
        sendmessage(filebox,mess);
        receivemessage(datapoolbox,mess);
        k:=0;
```

```
    end;
  for i:=1 to nroutsignals[log]  do
    begin
    k:=k+1;
    if listlevel = bipolarlev then
      mess^.data^[k]:=trunc(outsignals[log,i].value*2048.0)
    else mess^.data^[k]:=trunc(outsignals[log,i].value*4096);
    end;
  tags[log]:=false;
  end;
if tags[print] then
  begin
  for i:=1 to nroutsignals[print] do
    if outsignals[print,i].sigtag = nodesig then
      if listlevel = bipolarlev then
      write(outsignals[print,i].name,
              outsignals[print,i].value:7:4,' ')
      else write(outsignals[print,i].name,
                 outsignals[print,i].value:8:5,' ')
    else
      if outsignals[print,i].sigtag = adsig then
        if listlevel = bipolarlev then
        write('AD',outsignals[print,i].chan:1,
                outsignals[print,i].value:7:4,' ')
        else write('AD',outsignals[print,i].chan:1,
                   outsignals[print,i].value:8:5,' ')
      else
        if listlevel = bipolarlev then
        write('CONST ',outsignals[print,i].value:7:4,' ')
        else write('CONST ',outsignals[print,i].value:8:5,' ');
  writeln;
  tags[print]:=false;
  end;
if tags[plot] then
    begin
    write(chr(27),'K');
    for i:= 1 to nroutsignals[plot] do
      begin
      if listlevel = bipolarlev then
        begin
        plotvalue:=outsignals[plot,i].value;
        plotvalue:=plotvalue+1.0;
        col:=round(plotvalue/0.025974);
        end
      else col:=round(outsignals[plot,i].value/0.012658);
      if col > 77 then col:=77;
      if col < 1  then col:=1;
      write(chr(27),'F','7',chr(col+33),plotchar[i]);
      end;
    writeln;
    if listlevel = bipolarlev then
      begin
      write(' -1',chr(27),'F','7',chr(72),'0',
           chr(27),'F','7',chr(109),'+1');
      end
```

```
      else
        write('O',chr(27),'F','7',chr(109),'+1');
      write(chr(13));
      tags[plot]:=false;
      end;
  end;{with}
if  messageaccept(listbox,listmess) then
  begin
  with listmess^.info^ do
  case infotype  of
    runlogmess: bool[log]:=true;
    endlogmess: begin
                bool[log]:=false;
                for i:=k+1 to 256 do mess^.data^[i]:=9999;
                sendmessage(filebox,mess);
                receivemessage(infopoolbox,filemess);
                filemess^.info^.infotype:=endlogmess;
                sendmessage(filebox,filemess);
                end;
    endplotmess : bool[plot]:=false;
    endprintmess : bool[print]:=false;
    logmess    : begin
                 signals.nroutsignals[log]:=nlogsignals;
                 timervalues[log]:=sampleperiod;
                 timers[log]:=timervalues[log];
                for i:=1 to nlogsignals do
                  signals.outsignals[log,i]:=
                  messlogsignals[i];
                 k:=0;
                 receivemessage(datapoolbox,mess);
                 listlevel:=loglevel;
                 end;
    printmess : begin
                signals.nroutsignals[print]:=nprintsignals;
                timervalues[print]:=printsampleperiod;
                timers[print]:=timervalues[print];
                for i:=1 to nprintsignals do
                  signals.outsignals[print,i]:=
                  messprintsignals[i];
                listlevel:=printlevel;
                bool[print]:=true;
                end;
    plotmess   :  begin
                signals.nroutsignals[plot]:=nprintsignals;
                timervalues[plot]:=printsampleperiod;
                timers[plot]:=timervalues[plot];
                for i:= 1 to nprintsignals do
                  signals.outsignals[plot,i]:=
                  messprintsignals[i];
                bool[plot]:=true;
                write(chr(27),'F','7',chr(32));
                listlevel:=printlevel;
                end;
    end;{case}
  sendmessage(infopoolbox,listmess);
```

```
    end;
  wait(clocksync);
  end;{while}
  {$A+}
end;{interpreter}
```

```
{********
* Clock *
********}

{process} procedure clock;

var mainperiodint:integer;

begin
setpriority(1);
mainperiodint:=round(mainperiod/0.02);
while true do
  begin
  waittime(mainperiodint);
  signal(clocksync);
  end;{while}
end;{clock}
```

```
{***************
* Filehandler *
***************}

{process} procedure filehandler;

var  mess:messageref;
     afilesize,bfilesize,int:integer;
     afile,bfile:file of datatype;
     infofile:text;
     afiletag,lognewfile,ready:boolean;
     i,k,nr,actualfilesize:integer;
     answermess:messageref;
     devicetest:array[1..4] of char;
     infofilename:array[1..6] of char;
     logfilename:array[1..14] of char;
     filelevel:leveltype;

begin
setpriority(7);
while true do
  begin
  receivemessage(filebox,mess);
  while (mess^.info = nil) or (mess^.info^.infotype <> startlog)
 do
    begin
    if mess^.info = nil then sendmessage(datapoolbox,mess)
    else sendmessage(infopoolbox,mess);
    receivemessage(filebox,mess);
    end;
  with mess^.info^ do
    begin
    afilesize:=infofilesize;
    for i:=1 to 4 do devicetest[i]:=logfile[i];
    if devicetest = 'DX1:' then
      begin
      for i:=5 to 10 do infofilename[i-4]:=logfile[i];
      rewrite(infofile,infofilename,'INF',afilesize);
      end
    else
      rewrite(infofile,logfile,'INF',afilesize);
    for i:=1 to 8 do write(infofile,signalorder[i]);
    writeln(infofile,'Logperiod       Nr of signals to be logged');
    writeln(infofile,'  ',logperiod:6:2,'                ',nsignals:1);
    filelevel:=level;
    writeln(infofile,'AD-level:');
    if level = bipolarlev then writeln(infofile,bipolar)
    else writeln(infofile,unipolar);
    afilesize:=-1;
    for i:=1 to 14 do logfilename[i]:=' ';
    for i:=1 to 10 do logfilename[i]:=logfile[i];
    if ext <> blanks then
      begin
      i:=1;
      while logfilename[i] <> ' ' do i:=i+1;
```

```
      logfilename[i]:='.';
      for k:=i+1 to i+3 do
      logfilename[k]:=ext[k-i];
      end;
    rewrite(afile,logfilename,'DAT',afilesize);
    afiletag:=true;
    infotype:=blocksize;
    blocks:=afilesize;
    end;{with}
sendmessage(opcombox,mess);
lognewfile:=false;
nr:=0;
ready:=false;
actualfilesize:=afilesize;
while not ready do
    begin
    receivemessage(filebox,mess);
    if mess^.info <> nil then
    begin
    with mess^.info^ do
    case infotype of
      infomess     : writeln(infofile,string);
      newfile      : begin
                     receivemessage(infopoolbox,answermess);
                     if (not lognewfile) then
                       begin
                       for i:=1 to 14 do logfilename[i]:=' ';
                       for i:=1 to 10 do
                         logfilename[i]:=newlogfile[i];
                       if newext <> blanks then
                         begin
                         i:=1;
                         while logfilename[i] <> ' ' do i:=i+1;
                         logfilename[i]:='.';
                         for k:=i+1 to i+3 do
                           logfilename[k]:=newext[k-i];
                         end;
                       lognewfile:=true;
                       if afiletag then
                         begin
                         bfilesize:=-1;
                         rewrite(bfile,logfilename,'DAT',bfilesize);
                         answermess^.info^.infotype:=blocksize;
                         answermess^.info^.blocks:=bfilesize;
                         end
                       else
                         begin
                         afilesize:=-1;
                         rewrite(afile,logfilename,'DAT',afilesize);
                         answermess^.info^.infotype:=blocksize;
                         answermess^.info^.blocks:=afilesize;
                         end;
                       end
                     else
                       begin
```

```
                    answermess^.info^.infotype:=errormess;
                    end;
                  sendmessage(opcombox,answermess);
                  end;
  changemess    : begin
                  receivemessage(infopoolbox,answermess);
                  if (lognewfile) then
                    begin
                    nr:=0;
                    lognewfile:=false;
                    answermess^.info^.infotype:=okmess;
                    if afiletag then
                      begin
                      afiletag:=false;
                      close(afile);
                      actualfilesize:=bfilesize;
                      end
                    else
                      begin
                      afiletag:=true;
                      close(bfile);
                      actualfilesize:=afilesize;
                      end;
                    end
                  else
                    begin
                    answermess^.info^.infotype:=errormess;
                    end;
                  sendmessage(opcombox,answermess);
                  end;
  endlogmess    : begin
                  ready:=true;
                  logfinished;
                  end;
  end;{case}
sendmessage(infopoolbox,mess);
end {then}
else
  begin
  if afiletag then
    begin
    afile^:=mess^.data^;
    put(afile);
    end
  else
    begin
    bfile^:=mess^.data^;
    put(bfile);
    end;
  actualfilesize:=actualfilesize-1;
  if actualfilesize = 0 then
  if lognewfile then
    begin
    lognewfile:=false;
    if afiletag then
```

```
          begin
          afiletag:=false;
          close(afile);
          actualfilesize:=bfilesize;
          end
        else
          begin
          afiletag:=true;
          close(bfile);
          actualfilesize:=afilesize;
          end;
        writeln('Change of logfile has been made');
        writeln('>');
        end
      else
        begin
        ready:=true;
        receivemessage(infopoolbox,answermess);
        mess^.info^.infotype:=endlogmess;
        sendmessage(listbox,answermess);
        logfinished;
        removenames(log);
        writeln('Logging aborted : end of file');
        writeln('>');
        end;
    sendmessage(datapoolbox,mess);
    end;{else}
    end;{while}
  if afiletag then close(afile) else close(bfile);
  close(infofile);
  end;{while}
end;{filehandler}
```

```
{********
* OPCOM *
********}

{process} procedure opcom;

label 999;

var err:boolean;

begin {opcom}
setpriority(12);
while true do
  begin
  writeln;
  ch[term]:=' ';
  write('>');
  getident(err,term);
  if err then goto 999;
  ops[term]:=newx;
  while (opname[ops[term]] <> identifier[term]) and
        (ops[term] <> lastopindex) do ops[term]:=succ(ops[term]);
  if extcomfile then
    if ops[term] <> breakx then writeln('External commandfile')
    else breakrut
  else
  case ops[term] of
    newx            : newnode(term);
    parx            : parnode(term);
    dispx           : dispnode(term);
    deletex         : deletenode(term);
    runlogx         : listmess(runlogx,term);
    logmessx        : messlog(term);
    endplotx        : listmess(endplotx,term);
    endprintx       : listmess(endprintx,term);
    endlogx         : listmess(endlogx,term);
    newlogfilex     : newlogfile(term);
    changex         : change(term);
    logx            : logroutine(term);
    printx          : outroutine(term,printx);
    waitx           : writeln('Only valid as external command');
    comfilex        : commandfilestart;
    breakx          : writeln('No commandfile');
    plotx           : outroutine(term,plotx);
    adlevelx        : adlevel(term);
    endx            : endprogram(term);
    helpx           : helproutine(term);
    dalevelx        : dalevel(term);
    lastopindex     : if identifier[term] <> blanks then error(noop);
    end;{case}
999:
  readln;
  end;{while}
end;{opcom}
```

```
{***********
* EXTOPCOM *
***********}

{process} procedure extopcom;

label 999;
label 888;

var err:boolean;
    k:integer;
    mess:messageref;

begin {extopcom}
setpriority(9);
while true do
  begin
  receivemessage(extbox,mess);
  reset(comfile,mess^.info^.newlogfile,'COM',k);
  if k = -1 then
    begin
    writeln('Non existing file ',mess^.info^.newlogfile,'.COM');
    writeln('>');
    setcomfile(false);
    sendmessage(infopoolbox,mess);
    goto 999;
    end;
  sendmessage(infopoolbox,mess);
  while not messageaccept(extbox,mess) and not eof(comfile)
    do
      begin
      writeln;
      ch[fil]:=' ';
      write('$');
      getident(err,fil);
      if err then goto 888;
      ops[fil]:=newx;
      while (opname[ops[fil]] <> identifier[fil]) and
            (ops[fil] <> lastopindex) do
          ops[fil]:=succ(ops[fil]);
    case ops[fil] of
         newx           : newnode(fil);
         parx           : parnode(fil);
         dispx          : dispnode(fil);
         deletex        : deletenode(fil);
         runlogx        : listmess(runlogx,fil);
         logmessx       : messlog(fil);
         endplotx       : listmess(endplotx,fil);
         endprintx      : listmess(endprintx,fil);
         endlogx        : listmess(endlogx,fil);
         newlogfilex    : newlogfile(fil);
         changex        : change(fil);
         logx           : logroutine(fil);
         printx         : outroutine(fil,printx);
         waitx          : waitroutine;
```

```
          comfilex          : writeln('This is a commandfile');
          breakx            : writeln('Not valid');
          plotx             : outroutine(fil,plotx);
          adlevelx          : adlevel(fil);
          helpx             : helproutine(fil);
          endx              : endprogram(fil);
          dalevelx          : dalevel(fil);
          lastopindex       : if identifier[fil] <> blanks then
                                error(noop);
        end;{case}
      888:
      readln(comfile);
      end;
    if eof(comfile) then begin setcomfile(false);
      writeln('>'); end
    else sendmessage(infopoolbox,mess);
    close(comfile);
  999:
  end;{while}
end;{extopcom}


{***************
* WAITPROCESS *
***************}


{process} procedure waitprocess;

var hours,minutes,seconds:integer;
    mess:messageref;
    ok:boolean;
begin
setpriority(1);
while true do
  begin
  receivemessage(waitbox,mess);
  with mess^.info^ do
    begin
    hours:=hr;
    minutes:=minu;
    seconds:=sek;
    end;
  sendmessage(infopoolbox,mess);
  ok:=false;
  while (not ok) and (extcomfile) do
    begin
    waittime(sec);
    if seconds >  0 then seconds:=seconds-1
    else
      if minutes = 0 then
        begin
        if hours = 0 then ok:=true
        else
          begin
```

```
            hours:=hours-1;
            minutes:=60;
            end;
          end
        else
          begin
          minutes:=minutes-1;
          seconds:=60;
          end;
      end;
      causestart;
    end;{while}
end;
```

```
procedure initnames;

{Initializes all the names }


begin

typename[pid]          :='PID        ';
typename[puls]         :='PULSE      ';
typename[sinus]        :='SINE       ';
typename[prbs]         :='PRBS       ';
typename[triang]       :='TRIANGLE   ';
typename[sawtooth]     :='SAWTOOTH   ';
typename[stat]         :='STAT       ';
typename[add]          :='ADD        ';
typename[filter]       :='FILTER     ';
opname[newx]           :='NEW        ';
opname[parx]           :='PAR        ';
opname[dispx]          :='DISP       ';
opname[deletex]        :='DELETE     ';
opname[logx]           :='LOG        ';
opname[runlogx]          :='RUNLOG     ';
opname[changex]          :='CHANGE     ';
opname[newlogfilex]    :='NEWLOGFILE';
opname[endlogx]        :='ENDLOG     ';
opname[printx]         :='PRINT      ';
opname[endprintx]      :='ENDPRINT   ';
opname[plotx]          :='PLOT       ';
opname[endplotx]       :='ENDPLOT    ';
opname[adlevelx]       :='ADRANGE    ';
opname[dalevelx]       :='DARANGE    ';
opname[endx]           :='EXIT       ';
opname[helpx]          :='HELP       ';
opname[comfilex]       :='COMFILE    ';
opname[breakx]         :='BREAK      ';
opname[logmessx]       :='LOGMESS    ';
opname[waitx]          :='WAIT       ';
parname[pidin]         :='PIDIN      ';
parname[pidref]        :='PIDREF     ';
parname[outchanpar]       :='OUTCHANNEL';
parname[kpar]          :='K          ';
parname[tipar]         :='TI         ';
parname[tdpar]         :='TD         ';
parname[gdpar]         :='GD         ';
parname[minupar]       :='MINU       ';
parname[maxupar]       :='MAXU       ';
parname[ptpamppar]     :='AMP        ';
parname[periodpar]     :='PERIOD     ';
parname[meanvaluepar]:='MEANVALUE  ';
parname[dutycyclepar]:='DUTYCYCLE  ';
parname[input]         :='INPUT      ';
parname[outchanmean]   :='OUTCHMEAN  ';
parname[outchanvar]    :='OUTCHVAR   ';
```

```
parname[outchanminy] :='OUTCHMINY ';
parname[outchanmaxy] :='OUTCHMAXY ';
parname[sampleperiodpar]:='SAMPPERIOD';
parname[addinputpar] :='ADDINPUT  ';
parname[weight]      :='WEIGHT    ';
parname[nrinsignalspar]:='NROFADDS  ';
parname[timecon]     :='TIMECONST ';
statname[mean]       :='MEAN      ';
statname[vari]        :='VARIANCE  ';
statname[miny]       :='MINY      ';
statname[maxy]       :='MAXY      ';
inputs[0]            :='AD0       ';
inputs[1]            :='AD1       ';
inputs[2]            :='AD2       ';
inputs[3]            :='AD3       ';
inputs[4]            :='AD4       ';
inputs[5]            :='AD5       ';
inputs[6]            :='AD6       ';
inputs[7]            :='AD7       ';
inputs[8]            :='AD8       ';
inputs[9]            :='AD9       ';
inputs[10]           :='AD10      ';
inputs[11]           :='AD11      ';
inputs[12]           :='AD12      ';
inputs[13]           :='AD13      ';
inputs[14]           :='AD14      ';
inputs[15]           :='AD15      ';
nodeparam[pid]:=[pidin,pidref,outchanpar,kpar,tipar,tdpar,
gdpar,minupar,maxupar];
nodeparam[puls]:=[ptpamppar,meanvaluepar,periodpar,
outchanpar,dutycyclepar];
nodeparam[sinus]:=nodeparam[puls]-[dutycyclepar];
nodeparam[triang]:=nodeparam[sinus];
nodeparam[sawtooth]:=nodeparam[sinus];
nodeparam[stat]:=[input,outchanmean,outchanvar,
outchanminy,outchanmaxy];
nodeparam[prbs]:=nodeparam[sinus]+[sampleperiodpar]-[periodpar];
nodeparam[add]:=[outchanpar,addinputpar,weight,nrinsignalspar];
nodeparam[filter]:=[input,timecon,outchanpar];
end;{initnames}
```

```
procedure initiate;

{Initiates the program}

var ptr:messageref;
    i:integer;
    invalue:real;

begin
initsem(clocksync,0);
initsem(endsync,0);
initmailbox(infopoolbox);
initmailbox(datapoolbox);
initmailbox(filebox);
initmailbox(opcombox);
initmailbox(listbox);
initmailbox(extbox);
initmailbox(waitbox);
for i:=1 to 3   do
  begin
  new(ptr);
  ptr^.data:=nil;
  new(ptr^.info);
  sendmessage(infopoolbox,ptr);
  end;{for}
for i:=1 to 2 do
  begin
  new(ptr);
  ptr^.info:=nil;
  new(ptr^.data);
  sendmessage(datapoolbox,ptr);
  end;
initnames;
initopcommonitor;
initlistmonitor;
write('Main sample period: ');
read(invalue);
mainperiod:=round(invalue/0.02)*0.02;
if invalue/0.02 <> round(invalue/0.02) then
  writeln('Main sample period = ',mainperiod:6:2);
end;{initiate}

{*******
* MAIN *
*******}

begin {main}
clksav;
initkernel(6100);
initiate;
createprocess(filehandler,2300);
createprocess(listprocess,850);
createprocess(clock,100);
createprocess(opcom,850);
createprocess(extopcom,850);
```

```
createprocess(waitprocess,100);
wait(endsync);
clkrestore;
end.
```

The compilation is done with the command file ALLPAS.COM

```
R PASCAL
FILTER/E=DX1:PREFIX,STRING,FILTER
MACRO FILTER
DEL/NOQ FILTER.MAC
R PASCAL
NEWNOD/E=DX1:PREFIX,OPCEXT,STRING,NEWNOD
MACRO NEWNOD
DEL/NOQ NEWNOD.MAC
R PASCAL
PARNOD/E=DX1:PREFIX,OPCEXT,STRING,PARNOD
MACRO PARNOD
DEL/NOQ PARNOD.MAC
R PASCAL
DISPNO/E=DX1:PREFIX,OPCEXT,STRING,DISPNO
MACRO DISPNO
DEL/NOQ DISPNO.MAC
R PASCAL
PRIRUT/E=DX1:PREFIX,OPCEXT,STRING,PRIRUT
MACRO PRIRUT
DEL/NOQ PRIRUT.MAC
R PASCAL
LOGRUT/E=DX1:PREFIX,OPCEXT,STRING,LOGRUT
MACRO LOGRUT
DEL/NOQ LOGRUT.MAC
R PASCAL
EXTWAI/E=DX1:PREFIX,OPCEXT,STRING,EXTWAI
MACRO EXTWAI
DEL/NOQ EXTWAI.MAC
R PASCAL
NAINIT/E=DX1:PREFIX,STRING,NAINIT
MACRO NAINIT
DEL/NOQ NAINIT.MAC
R PASCAL
INITLI/E=DX1:PREFIX,MONEXT,STRING,INITLI
MACRO INITLI
DEL/NOQ INITLI.MAC
R PASCAL
NACHOR/E=DX1:PREFIX,MONEXT,STRING,NACHOR
MACRO NACHOR
DEL/NOQ NACHOR.MAC
R PASCAL
ERROR/E=DX1:PREFIX,STRING,ERROR
MACRO ERROR
DEL/NOQ ERROR.MAC
R PASCAL
CHLLL/E=DX1:PREFIX,STRING,CHLLL
MACRO CHLLL
DEL/NOQ CHLLL.MAC
R PASCAL
LEGESE/E=DX1:PREFIX,MONEXT,STRING,LEGESE
MACRO LEGESE
DEL/NOQ LEGESE.MAC
R PASCAL
```

```
MONNEW/E=DX1:PREFIX,MONEXT,STRING,MONNEW
MACRO MONNEW
DEL/NOQ MONNEW.MAC
R PASCAL
MONDEL/E=DX1:PREFIX,MONEXT,STRING,MONDEL
MACRO MONDEL
DEL/NOQ MONDEL.MAC
R PASCAL
INITOP/E=DX1:PREFIX,STRING,INITOP
MACRO INITOP
DEL/NOQ INITOP.MAC
R PASCAL
FIRESE/E=DX1:PREFIX,STRING,FIRESE
MACRO FIRESE
DEL/NOQ FIRESE.MAC
R PASCAL
LOGGER=DX1:PREFIX,MAINEX,OPCOM2,EXTOPC,INIT
MACRO LOGGER
DEL/NOQ LOGGER.MAC
R PASCAL
MONITR/E=DX1:PREFIX,STRING,ADDA,LISTMO,OPMON
MACRO MONITR
DEL/NOQ MONITR.MAC
R PASCAL
GETIDE/E=DX1:PREFIX,IDEEXT,STRING,GETIDE
MACRO GETIDE
DEL/NOQ GETIDE.MAC
R PASCAL
FILEPR/E=DX1:PREFIX,LOGEXT,STRING,FILEPR
MACRO FILEPR
DEL/NOQ FILEPR.MAC
R PASCAL
LISTCL/E=DX1:PREFIX,LOGEXT,STRING,LISTCL
MACRO LISTCL
DEL/NOQ LISTCL.MAC
R PASCAL
READFI/E=DX1:PREFIX,STRING,READFI
MACRO READFI
DEL/NOQ READFI.MAC
R PASCAL
ACCMES/E=DX1:ACCMES
MACRO ACCMES
DEL/NOQ ACCMES.MAC
R PASCAL
RESULT/E=DX1:PREFIX,STRING,RESULT
MACRO RESULT
DEL/NOQ RESULT.MAC
R PASCAL
OPMMON/E=DX1:PREFIX,STRING,OPMMON
MACRO OPMMON
DEL/NOQ OPMMON.MAC
R PASCAL
DIVERS/E=DX1:PREFIX,OPCEXT,STRING,DIVERS
MACRO DIVERS
DEL/NOQ DIVERS.MAC
```

```
R PASCAL
HELP/E=DX1:PREFIX,HELEXT,STRING,HELP
MACRO HELP
DEL/NOQ HELP.MAC
R PASCAL
HELP1/E=DX1:PREFIX,HELP1
MACRO HELP1
DEL/NOQ HELP1.MAC
R PASCAL
HELP2/E=DX1:PREFIX,HELP2
MACRO HELP2
DEL/NOQ HELP2.MAC
R PASCAL
HELP3/E=DX1:PREFIX,HELP3
MACRO HELP3
DEL/NOQ HELP3.MAC
R PASCAL
HELP4/E=DX1:PREFIX,HELP4
MACRO HELP4
DEL/NOQ HELP4.MAC
R PASCAL
HELP5/E=DX1:PREFIX,HELP5
MACRO HELP5
DEL/NOQ HELP5.MAC
R PASCAL
HELP6/E=DX1:PREFIX,HELP6
MACRO HELP6
DEL/NOQ HELP6.MAC
```

FILE	CONTENTS
PREFIX	Type and variable declarations.
OPCEXT	External procedure declarations.
MAINEX	-"-
HELEXT	-"-
LOGEXT	-"-
IDEEXT	-"-
STRING	String variables.
ADDA	Adin2, Daout2.
LISTMO	Lookup, Interprete.
MONNEW	Monnewnode.
MONDEL	Mondelete.
LEGESE	Monlevelchange, Mongetnode, Monsetnode.
NACHOR	Monname, Moncheck, Monordercheck.
INITLI	Initlistmonitor.
OPMON	Setcomfile, Extcomfile, Causestart, Waitstart.
OPMMON	Checkcommand.
CHLLL	Opcommoncheck, Logstart, Logtime, Opmonlevel, Setopmonlevel.
FIRESE	Logfinished, Removenames, Setname.
INITOP	Initopcommonitor.
ERROR	Error
RESULT	Checkresult.
READFI	Endofline, Readchar, Readint, Readrel, Readnewline, Relreadnew, Intreadnew.
GETIDE	Skipblanks, Checkend, Getident, Pidlink, Readdescrinp, Writeclock, Breakrut.
FILTER	Filtercount.
EXTWAI	Commandfilestart, Waitroutine.
ACCMES	Messageaccept.
NEWNOD	Newnode.
PARNOD	Parnode.
DISPNO	Dispnode.
DIVERS	Deletenode, Listmess, Messlog, Newlogfile, Change, Adlevel, Dalevel, Endprogram.
PRIRUT	Outroutine.
LOGRUT	Logroutine.
HELP1	Helpnew, Helppar, Helpdisp, Helpdel.
HELP2	Helprunlog, Helpmesslog, Logendhelp, Helplog, Henewfile.
HELP3	Helpchange, Helpplot, Plotendhelp.
HELP4	Helpcomfile, Helpbreak, Helpend, Helpadlevel.
HELP5	Helpprint, Printendhelp, Helpwait.
HELP6	Helpdalevel, Helplast, Helphelp.
HELP	Helproutine.
LISTCL	Listprocess, Clock.
FILEPR	Filehandler.

OPCOM2	Opcom.
EXTOPC	Extopcom, Waitprocess.
NAINIT	Initnames.
INIT	Initiate, Main.

The linking is done with the command file LOGLIN.COM

```
R LINK
LOGGER,DX1:LOGGER=LOGGER,MONITR,GETIDE,DX1:KERNEL/C
DX0:FILEPR,LISTCL,READFI,ACCMES/C
DX0:RESULT,OPMMON,DIVERS,PASLIB/C
NEWNOD/O:1/C
PARNOD/O:1/C
DISPNO/O:1/C
PRIRUT/O:1/C
LOGRUT/O:1/C
EXTWAI/O:1/C
NAINIT/O:1/C
HELP/O:1/C
INITLI/O:2/C
NACHOR/O:2/C
ERROR/O:2/C
CHLLL/O:2/C
LEGESE/O:2/C
HELP1/O:2/C
HELP2/O:2/C
HELP3/O:2/C
HELP4/O:2/C
HELP5/O:2/C
HELP6/O:2/C
MONNEW/O:2/C
FILTER/O:2/C
MONDEL/O:2/C
INITOP/O:2/C
FIRESE/O:2/C
^C
```

```
CONVRT

label 999;

const  DX1 = 'DX1:       ';
       DX0 = 'DX0:       ';
       blanks = '           ';

type identtype = array [1..10] of char;
     leveltype = (bipolarlev,unipolarlev);
     sourcetype = (term,fil);



var  infofile:text;
     confile:text;
     datafile:file of integer;
     extension,dx1filename,identifier,filename:identtype;
     ch:char;
     devicetest:array [1..4] of char;
     infofilename:array [1..6] of char;
     m,k,i,beginsample,endsample,nrsample,nrofsignals:integer;
     all:boolean;
     datafilename,outfilename:array [1..14] of char;
     level:leveltype;
     realval:real;

procedure skipblanks(mode:sourcetype);
begin
if mode = term then
while (not eoln) and (ch = ' ') do read(ch)
else while (not eoln(infofile)) and (ch = ' ') do read(infofile,ch);
end;

procedure getident(mode:sourcetype);
var i:integer;
begin
identifier:=blanks;
skipblanks(mode);
i:=1;
while (((ch >= 'A') and (ch <= 'Z')) or
       ((ch >= '0') and (ch <= '9')) or (ch = ':')) and (i < 11) do
  begin
  identifier[i]:=ch;
  i:=i+1;
  if mode = term then
    begin
    if eoln then ch:=' '
    else read(ch);
    end
  else
    if eoln(infofile) then ch:=' '
    else read(infofile,ch);
  end;{while}
```

```
end;{getident}

begin
write('Input file: ');
extension:=blanks;
ch:=' ';
getident(term);
for i:=1 to 14 do datafilename[i]:=' ';
filename:=identifier;
for i:=1 to 10 do datafilename[i]:=identifier[i];
for i:=1 to 4 do devicetest[i]:=identifier[i];
if ch = '.' then
  begin
  ch:=' ';
  getident(term);
  extension:=identifier;
  end;
if extension <> blanks then
  begin
  i:=1;
  while datafilename[i] <> ' ' do i:=i+1;
  datafilename[i]:='.';
  for k:=i+1 to i+3 do datafilename[k]:=extension[k-i];
  end;
reset(datafile,datafilename,'DAT',k);
if k = -1 then
  begin
  if extension = blanks then
    writeln('Non existing datafile: ',filename,'.DAT')
  else
    writeln('Non existing datafile: ',datafilename);
  goto 999;
  end;
reset(infofile,filename,'INF',k);
if k = -1 then
  begin
  if devicetest = 'DX1:' then
    begin
    for i:=5 to 10 do infofilename[i-4]:=filename[i];
    reset(infofile,infofilename,'INF',k);
    end
  else
    begin
    dx1filename:='DX1:      ';
    for i:=5 to 10 do dx1filename[i]:=filename[i-4];
    reset(infofile,dx1filename,'INF',k);
    end;
  if k = -1 then
    begin
    writeln('Non existing informationfile.');
    goto 999;
    end;
  end;
write('Output file: ');
outfilename:='              ';
```

```
ch:=' ';
readln;
getident(term);
i:=1;
while (i <= 10) do
  begin
  if identifier[i] <> ' '  then
    outfilename[i]:=identifier[i];
  i:=i+1;
  end;
if ch = '.' then
  begin
  ch:=' ';
  getident(term);
  outfilename[i]:='.';
  for m:=i+1 to i+3 do
    outfilename[m]:=identifier[m-i];
  end;
k:=-1;
rewrite(confile,outfilename,'TXT',k);
readln;
write('Do you want all of the file converted?');
readln(ch);
if (ch = 'Y') or (ch = 'y') then
  begin
  all:=true;
  endsample:=-1;
  beginsample:=-1;
  end
  else
    begin
    write('From sample nr: ');
    readln(beginsample);
    write('To sample nr: ');
    read(endsample);
    all:=false;
    end;
readln(infofile);
readln(infofile,realval,nrofsignals);
readln(infofile);
ch:=' ';
getident(fil);
if identifier = 'BIPOLAR    ' then level:=bipolarlev
else level:=unipolarlev;
nrsample:=0;
repeat
  nrsample:=nrsample+1;
  if not all then
    if nrsample = beginsample then all:=true;
  for i:=1 to nrofsignals do
    begin
      if all then
        begin
        if level = bipolarlev then
          write(confile,(datafile^/2048.0):7:4,' ')
```

```
        else
          write(confile,(datafile^/4096.0):8:5,' ');
        end;
        if not eof(datafile) then get(datafile);
    end;{for}
    if all then writeln(confile);
    if all then
      if nrsample = endsample then all:=false;
    while (datafile^ >= 9999) and (not eof(datafile)) do
      get(datafile);
until eof(datafile);
close(datafile);
close(confile);
close(infofile);
999:
end.
```