Motion Control

Modular System - MCS 32 EX
User guide

SERAD SA
271, route des crêtes
44440 TEILLE – France
☎ +33 (0)2 40 97 24 54
📠 +33 (0)2 40 97 27 04
💻 http://www.serad.fr
✉ info@serad.fr

SUMMARY

1- INTRODUCTION

1-1- Description of MCS32 EX

1-1-1- Generality

MCS32 EX is a numeric command specialised in motion control. Many choices of configuration are possible :

↳ alone

↳ with a terminal operator like Dialog 80 or Dialog 640

↳ with a PC

Serial communication ports can connect MCS32EX to other peripheral systems :

↳ touch screen

↳ printer

↳ PLC, etc...

With its modular design, the central unit accepts 10 modules. This advantage facilitates the optimization of your application.

1-1-2- Performance

⇨ Main Board:

↳ 32 bits Processor at 33 MHz

↳ 4Mbits of non-volatile RAM

↳ 8Mbits of Flash memory

↳ 2 serial communication ports - 1200 to 9600 b/s

↳ 1 to 8 axes

↳ 8 to 180 inputs/outputs

↳ real-time clock

↳ watch dog

⇨ Servo module SRV 85:

↳ DSP 40 MHz

↳ 2 high speed registration inputs(0,1µs)

↳ incremental encoder 1,5MHz

↳ +/-10V analogue output (servo cycle time 330µs)

↳ 24 Vdc sensor input

1-1-3- Modularity

MCS32 EX have many choice of modules to adapt of your application.

↳ Encoder, Servo - incremental or SSI

↳ Digital I/O - 8,16 or 24 channels

↳ Analogue I/O - 2 or 4 channels 12 bits

↳ CANOpen communication board

1-2- Description of MCB EX software

1-2-1- Generality

MCB EX is a user program development with MCB (Motion Control Basic) running under MICRODOFT WINDOWS environment.

MCB EX can manage up to 28 basic or PLC tasks and 20 000 user variables.

↳ System configuration with graphic tools

↳ Easy access to advanced instructions with tool box giving

↳ Fastest programming with the PLC tool box

↳ On-line Help and full-screen editor

↳ Debug mode to test your application with a PC

↳ Software oscilloscope captured and displayed up to six simultaneous parameters

1-3- Applications

↳ X, Y Tables - manipulators from 1 up to 8 axes

↳ Product registration

↳ Winding

↳ Electronic cam profiles

↳ Linear and circular interpolation

↳ Flying shears, rotary knives

↳ Software gearbox

↳ Colour synchronisation

↳ Cam box

2- INSTALLATION / STARTING

2-1- Environmental consideration

2-1-1- Environmental consideration

MCS must be installed vertically to have a natural convection cooling. MCS must be sheltered from humidity, liquid projection and dust.

Technical features :

➷ Power supply :100 to 250Vac 30VA

➷ Watchdog : NO Contact liberate from potential - 48Vac maxi 2A maxi

➷ Service temperature : 0 to 45°C

➷ Storage temperature : -10 to 70°C

MCS delivers a power supply of 5Vdc 1.6A for encoders.

2-2- Safety

2-2-1- Safety

➷ The security rules impose a manual restart after a default due to a power supply falling down, a watchdog default or an emergency stop.

➷ MCS's watchdog must be connected in serial with the emergency stop loop

➷ The watchdog must be closed at the beginning of the program. When a fault is detected (Internal fault, power fail, ...), the watchdog is automatically open.

➷ Following error parameter must be setup on each axis.

➷ The sensors that defines axes limits must be connected in serial in the emergency stop loop, or to the corresponding contacts of the servodrive.

➷ It is recommended to use software thrusts on finite axes.

➷ Drive validations should be handled by the user program in MCS

➷ Linked the « Power Electrical cupboard OK » to a PLC input and treated it in a safety basic task.

➷ On a following error detected, MCS sends all the axis to an open loop mode (Analogue command equal to zero) and opens the watchdog. To realize a treatment, you need to use the SECURITY instruction.

2-3- Connections

2-3-1- General explications

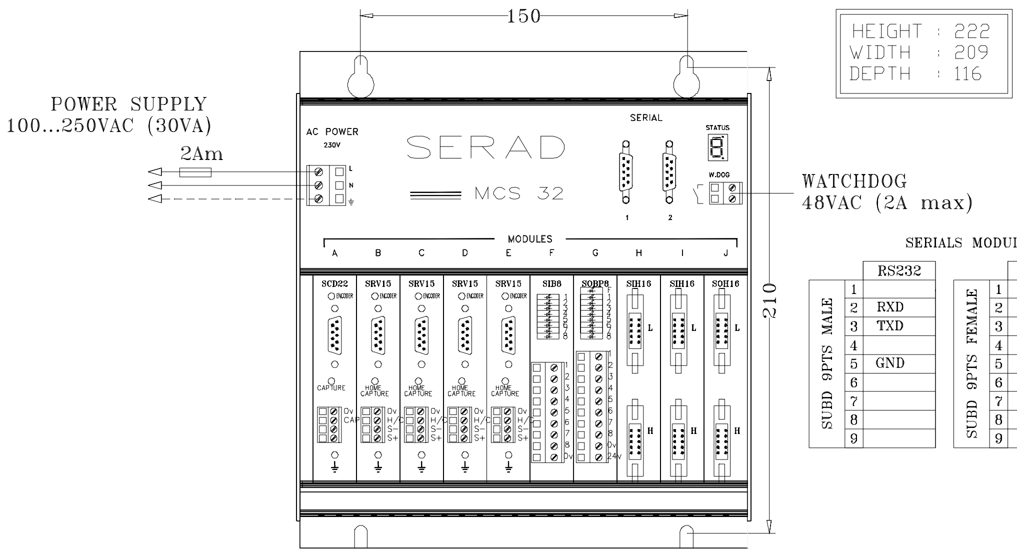
➷ Supply MCS’s power by isolated transformer with shield connected to ground.

➷ The encoder cables, analogue output cables, operator panel cable, must be shielded with shield connected at each end.

➢ MCS/PC cable must be shielded with shield connected at each end. It will have to be disconnected from the MCS when it is not used. All these cables, as well as the inputs/outputs cables will have to be separated and distant of the power modules.

➢ Inductive load must have an interference eliminator diode in DC and filter in AC. Diodes and filters must be placed as close to the charge as possible.

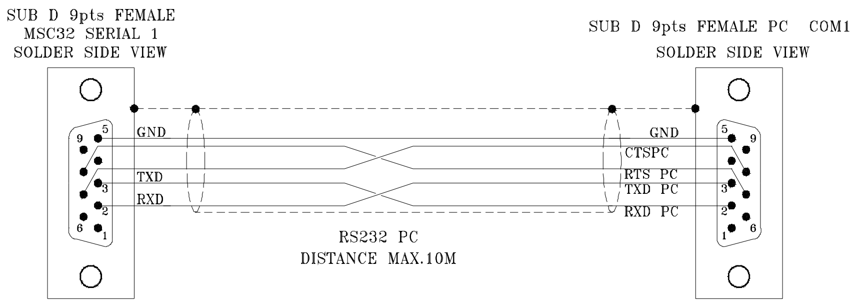
2-3-2- Master unit



SERIALS MODULES

SUBD 9PTS MALE	RS232
1	
2	RXD
3	TXD
4	
5	GND
6	
7	
8	
9	

SUBD 9PTS FEMALE	RS485
1	
2	
3	
4	
5	GND
6	
7	B
8	A
9	



WATCHDOG is a NO contact - 48 VAC maxi - 2A maxi.

2-3-3- Servo module : SRV85

Features

↳ Designed for 5 Volts encoders - line driver - maximal frequency : 1.5 Mhz.

↳ Encoder is powered by MCS 32 (1.6A Max for all encoders).

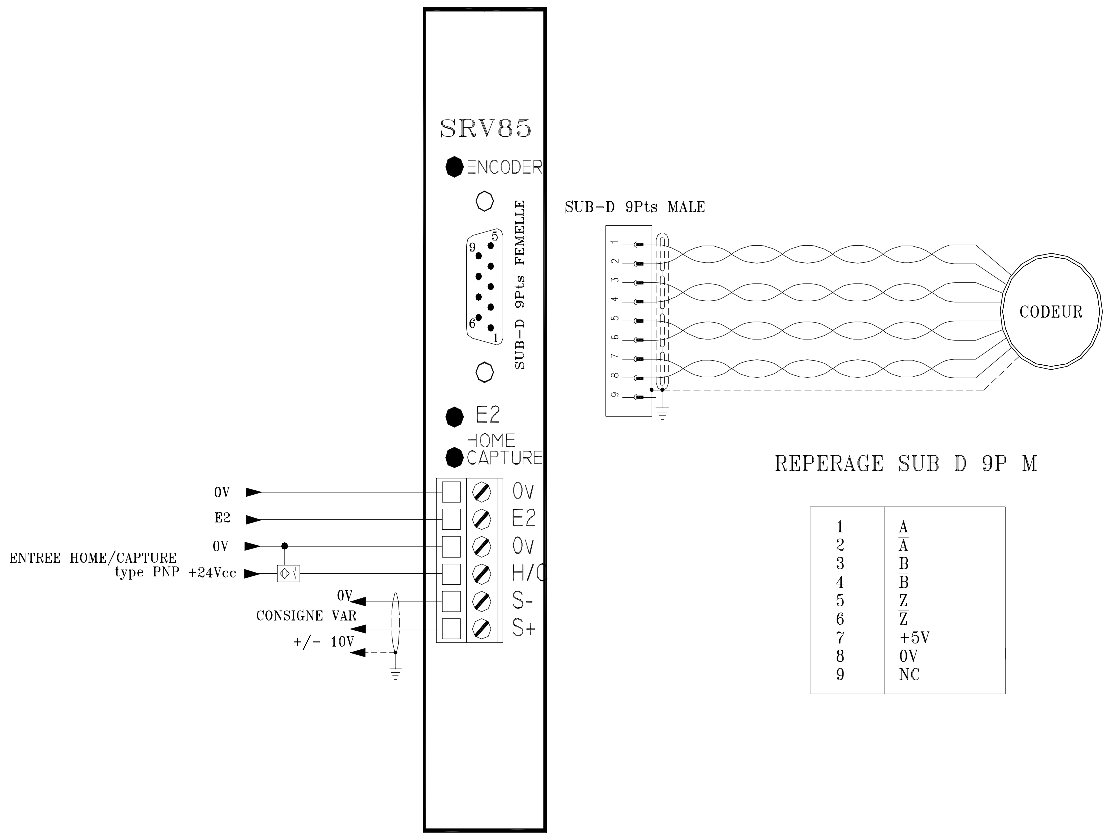
↳ 0,1 µs PNP type capture input : 12 mA/20..33 Vcc

↳ Analogue output +/- 10V / 5mA Max, 12 bits resolution – 0V referenced to the frame with a 470KΩ resistor and a 4,7nF capacitance.

↳ Led for Z encoder signal

↳ Led for Home capture input

Connections



1	A
2	$\overline{A}$
3	B
4	$\overline{B}$
5	Z
6	$\overline{Z}$
7	+5V
8	0V
9	NC

2-3-4- Servo module : SRV15

Features

✧ Designed for 5 Volts encoders - line driver - maximal frequency : 1.5 Mhz.

✧ Encoder is powered by MCS 32 (1.6A Max for all encoders).

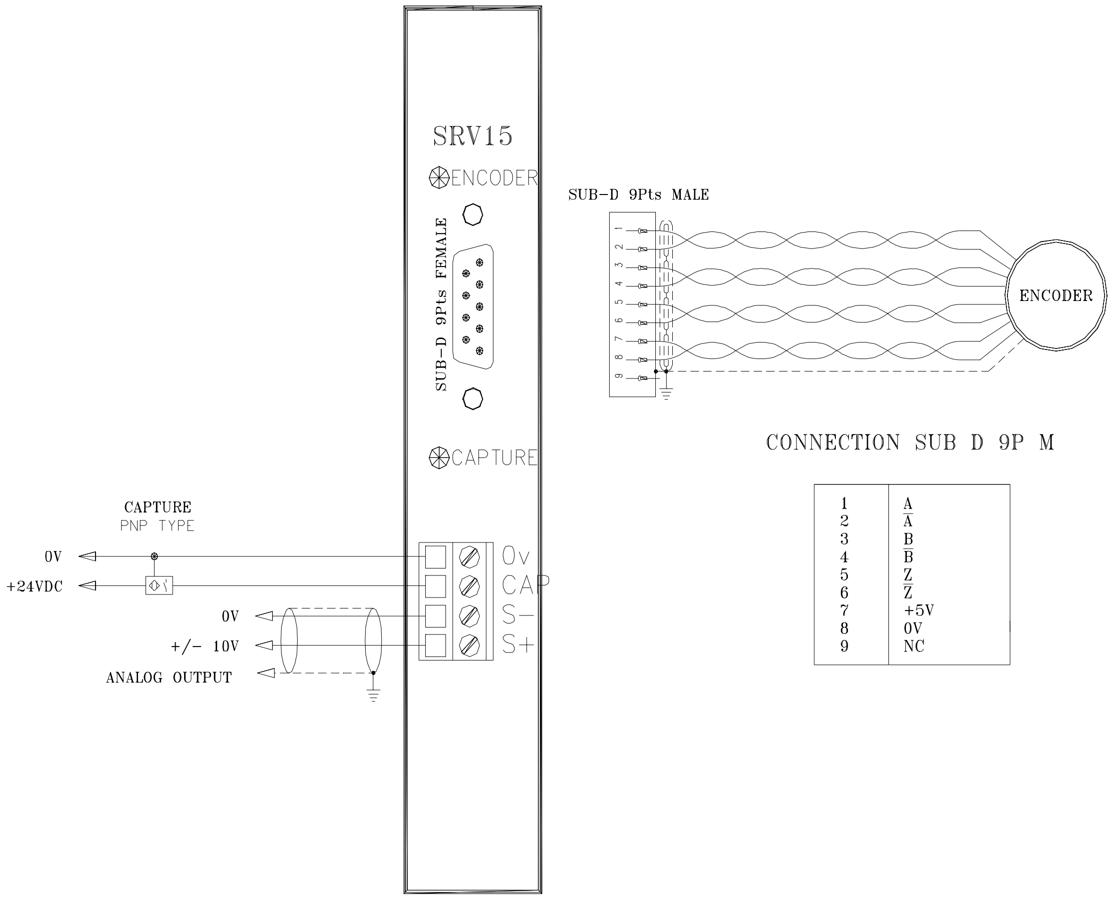
✧ 0,1 µs PNP type capture input : 12 mA/20..33 Vcc

✧ Analogue output +/- 10V / 5mA Max, 12 bits resolution – 0V referenced to the frame with a 470KΩ resistor and a 4,7nF capacitance.

✧ Led for Z encoder signal

✧ Led for Home capture input

Connections



CONNECTION SUB D 9P M

1	A
2	$\overline{A}$
3	B
4	$\overline{B}$
5	Z
6	$\overline{Z}$
7	+5V
8	0V
9	NC

2-3-5- Servo module : SRV1524

Features

✍ Designed for 24 Volts encoders - line driver - maximal frequency : 1.5 Mhz.

✍ Encoder is powered by MCS 32 (1.6A Max for all encoders).

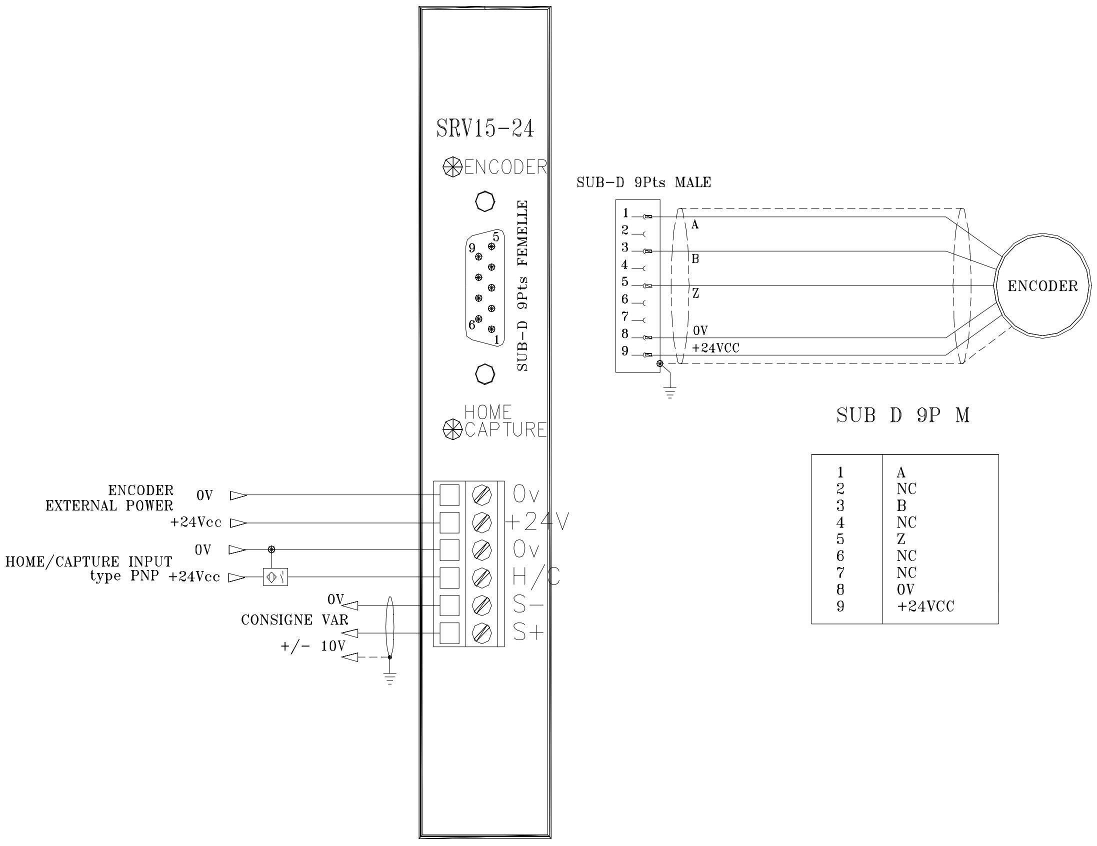
✍ 0,1 µs PNP type capture input : 12 mA/20..33 Vcc

✍ Analogue output +/- 10V / 5mA Max, 12 bits resolution – 0V referenced to the frame with a 470KΩ resistor and a 4,7nF capacitance.

✍ Led for Z encoder signal

✍ Led for Home capture input

Connections



SUB D 9P M

1	A
2	NC
3	B
4	NC
5	Z
6	NC
7	NC
8	0V
9	+24VCC

2-3-6- Servo module : SSI15

Features

✎ Designed for 11...30V SSI encoders - line driver.

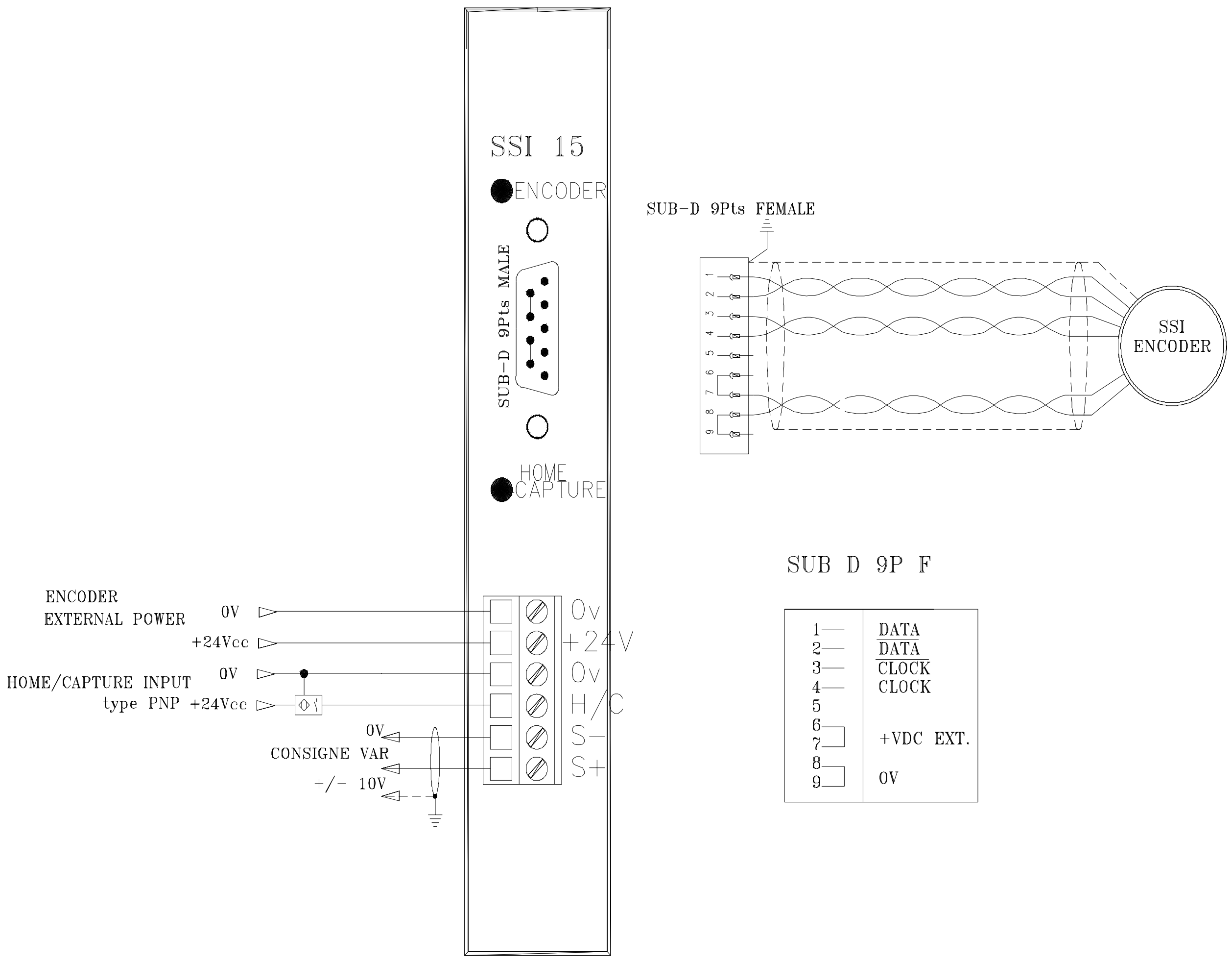
✎ Clock frequency : 100 kHz...1MHz, gray code, no parity bit. External supply 11...30V.

✎ 0,1 µs PNP type capture input : 12 mA/20..33 Vcc

✎ Analogue output +/- 10V / 5mA Max, 12 bits resolution – 0V referenced to the frame with a 470KΩ resistor and a 4,7nF capacitance.

✎ Led for Home capture input

Connections



1	DATA
2	$\overline{\text{DATA}}$
3	$\overline{\text{CLOCK}}$
4	CLOCK
5	
6–7	+VDC EXT.
8–9	0V

2-3-7- Stepper module : STP85

The STP 85 stepper module allows to drive a stepper motor in open loop. It accepts all positionning functions, gearboxes, electronic cams, interpolation…

↳ 4 opto-isolated outputs CLOCK, DIRECTION, ENABLE and ILIMIT

Maximal frequency :

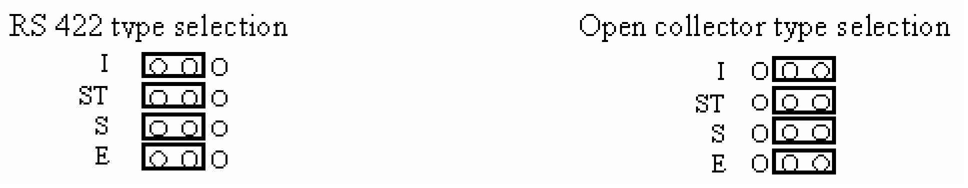
- 500 Khz for CLOCK and DIRECTION

- 500 hz for ENABLE and ILIMIT

Output type selected by on board jumpers :

- RS 422 5v line drivers

- Open collector, 100mA, not protected against short circuits, external power supply from 5 to 33 Vdc



- RS 422 is the default selection

Attention : You must verify that the selected type is compatible with the drive. An incorrect choice may cause damages to the STP85.

↳ 2 PNP type 0,1µs capture or home inputs : 12mA / 20..33 Vdc

- Opto-isolated inputs

- Leds for state visualisation

- ***The '0v' of C1 and C2 inputs are not internaly connected.***

9 points male SUBD pin assignement :

Pin	**RS 422 Type**	**Open collector type**
1	CLOCK +	CLOCK
2	DIRECTION +	DIRECTION
3	ENABLE +	ENABLE
4	ILIMIT +	ILIMIT
5	GND	GND
6	CLOCK -	
7	DIRECTION -	
8	ENABLE -	
9	ILIMIT -	

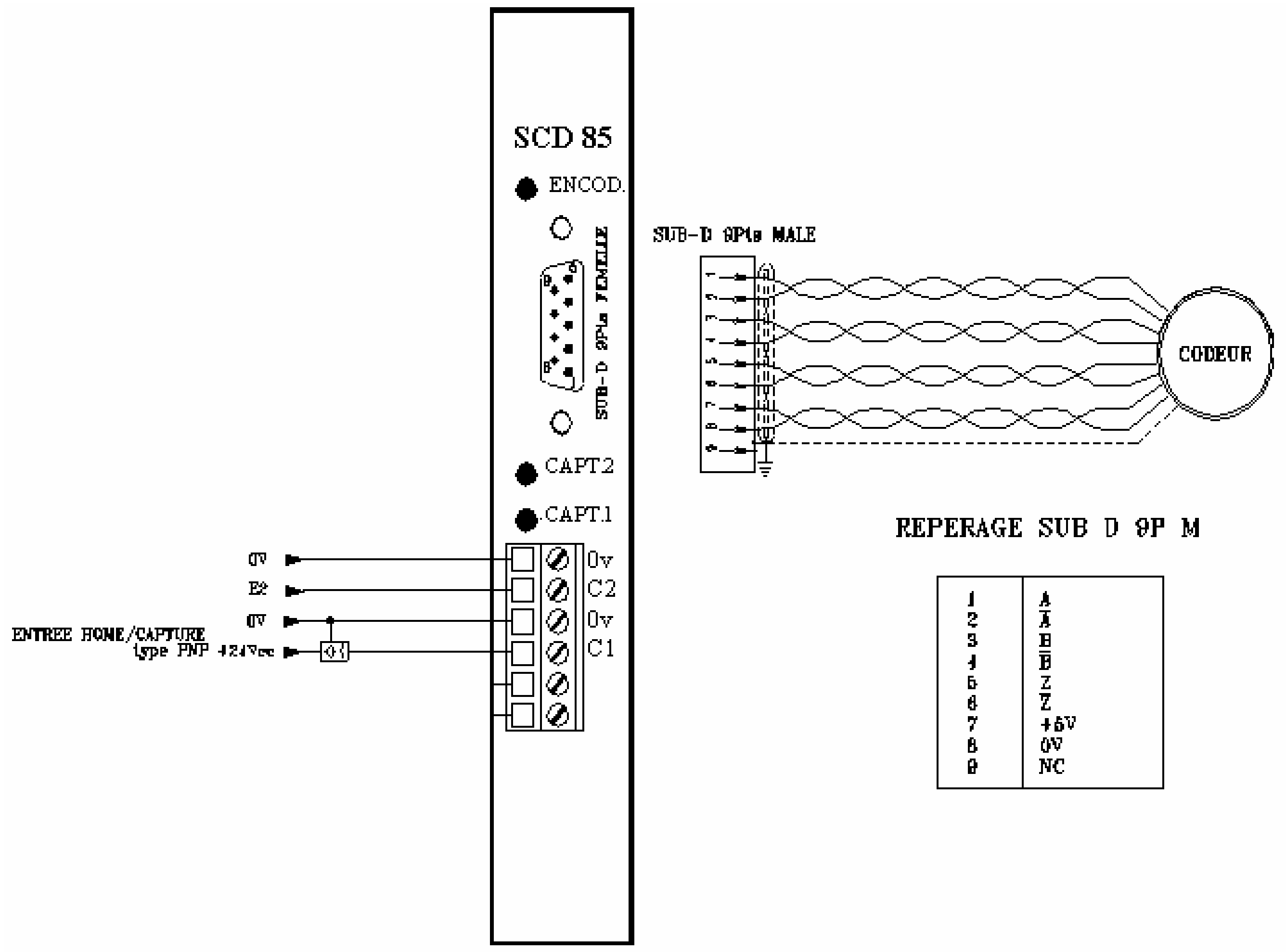
Use a shielded cable with the shield connected at each end.

2-3-8- Encoder module : SCD 85

Features

- ✎ Designed for 5 Volts encoders - line driver - maximal frequency : 6 Mhz.
- ✎ Encoder is powered by MCS 32EX (1.6A Max for all encoders).
- ✎ 0,1 µs PNP type capture input : 12 mA/20..33 Vcc
- ✎ Led for Z encoder signal
- ✎ Led for Home and capture inputs

Connections



REPERAGE SUB D 9P M

1	A
2	$\overline{A}$
3	B
4	$\overline{B}$
5	Z
6	$\overline{Z}$
7	+5V
8	0V
9	NC

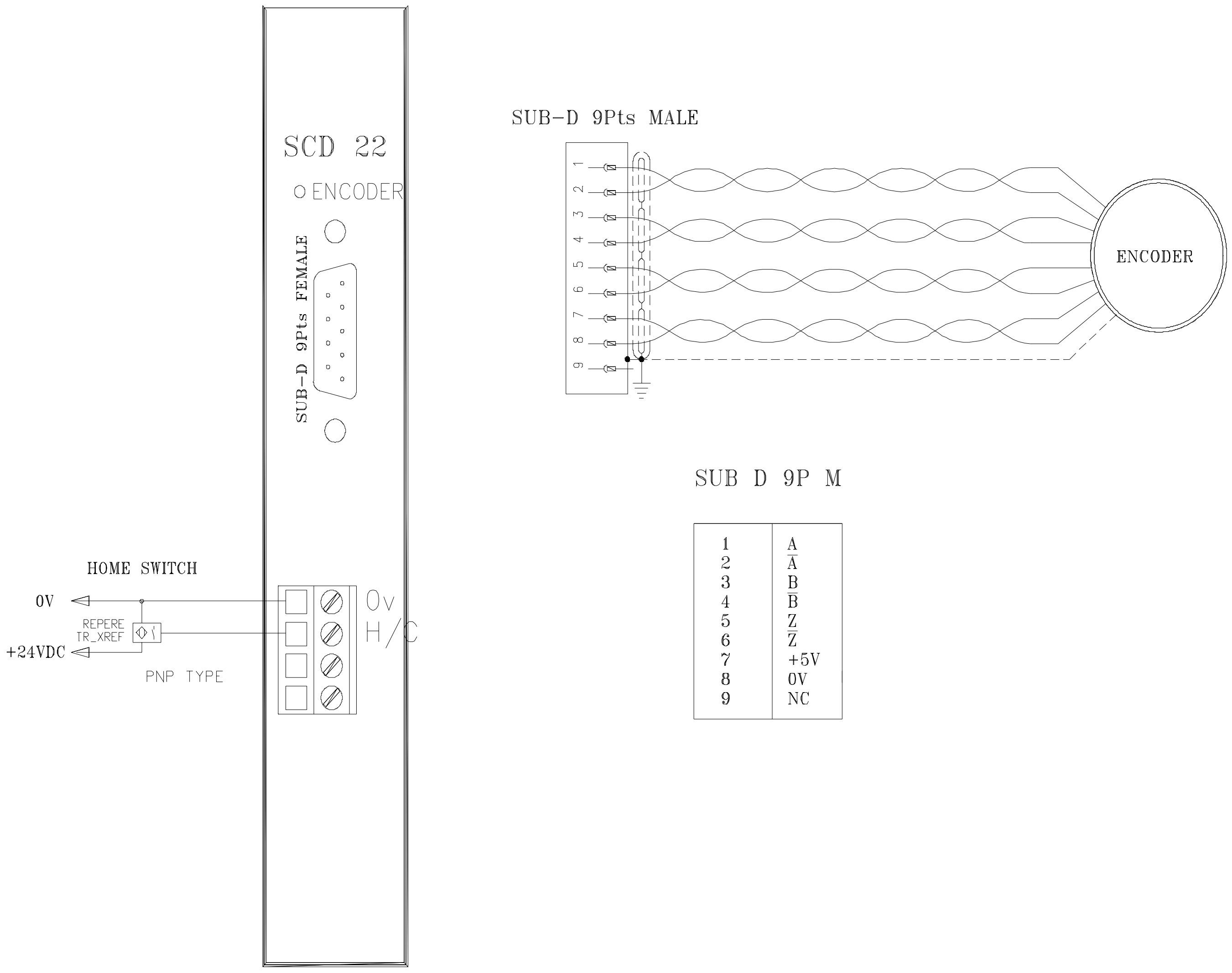
2-3-9- Encoder module : SCD22

Features

- ✎ 5 Volts incremental encoder - line driver - Maximal frequency : 1.5 Mhz.
- ✎ Encoder is powered by MCS 32 (1.6A Max for all encoders).

- 0,1 µs PNP type capture input : 12 mA/20..33 Vcc
- Led for Z encoder signal
- Led for Home capture input

Connections



SUB D 9P M

1	A
2	$\overline{A}$
3	B
4	$\overline{B}$
5	Z
6	$\overline{Z}$
7	+5V
8	0V
9	NC

2-3-10- Encoder module : SCD2224

Features

➩ 24 Volts incremental encoder - line driver - Maximal frequency : 1.5 Mhz.

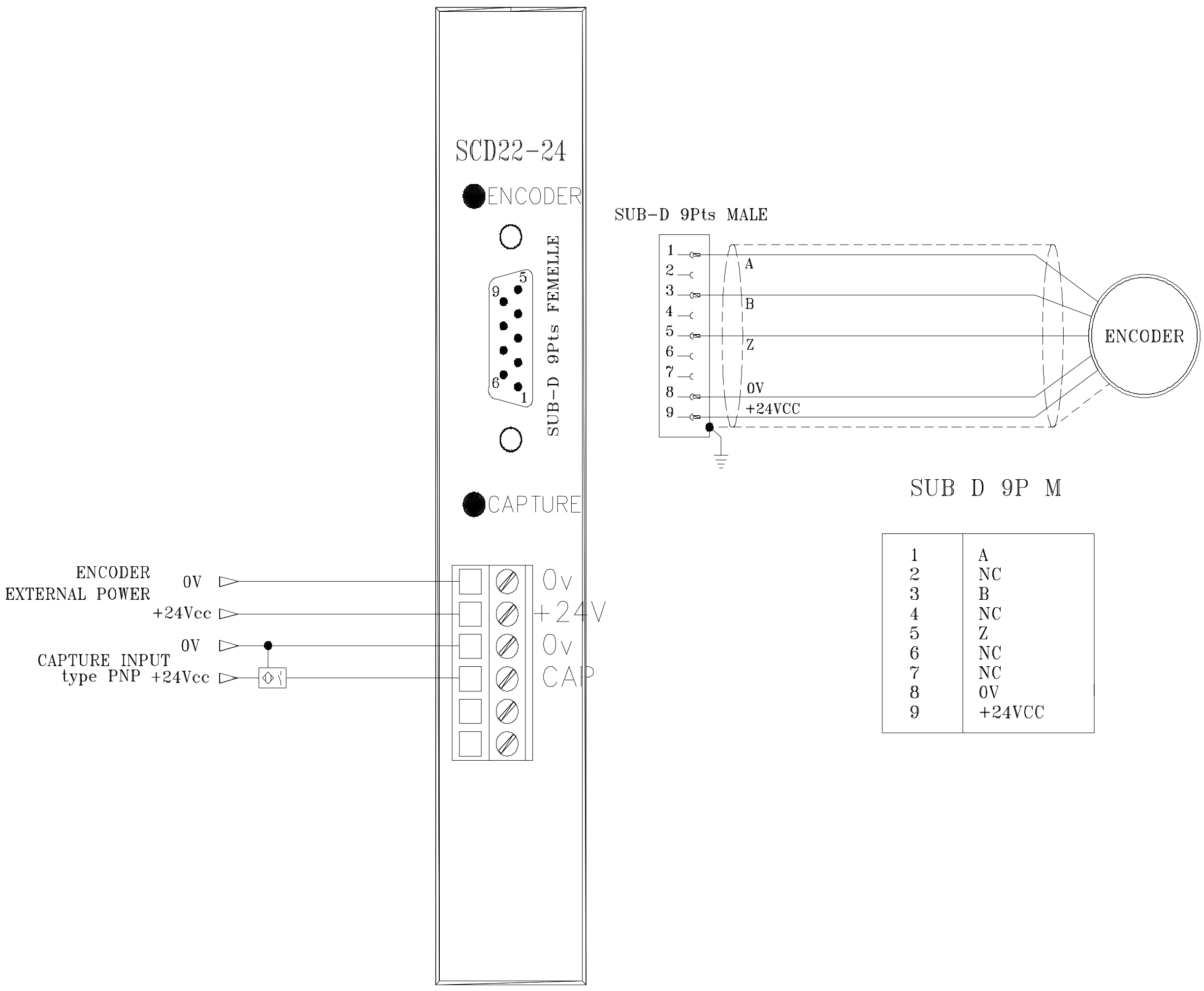
➩ Encoder is powered by MCS 32 (1.6A Max for all encoders).

➩ 0,1 µs PNP type capture input : 12 mA/20..33 Vcc

➩ Led for Z encoder signal

➩ Led for Home capture input

Connections



SUB D 9P M

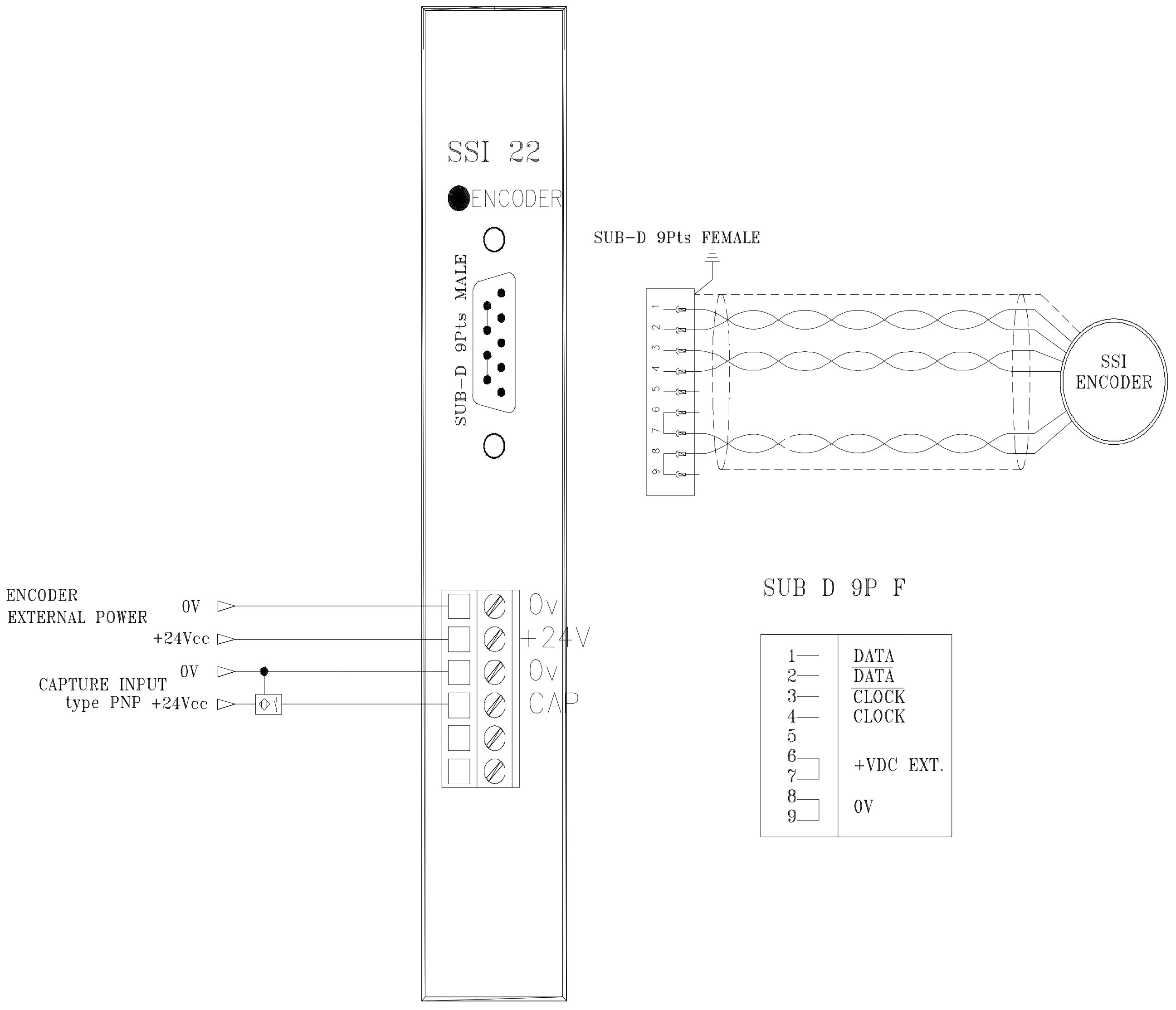
1	A
2	NC
3	B
4	NC
5	Z
6	NC
7	NC
8	0V
9	+24VCC

2-3-11- Encoder module : SSI22

Features

- ↳ Designed for 11...30V SSI encoders - line driver.
- ↳ Clock frequency : 100 kHz...1MHz, gray code, no parity bit.
- ↳ 0,1 µs PNP type capture input : 12 mA/20..33 Vcc
- ↳ Led for Home capture input

Connections



1	DATA
2	$\overline{\text{DATA}}$
3	$\overline{\text{CLOCK}}$
4	CLOCK
5	
6, 7	+VDC EXT.
8, 9	0V

2-3-12- 8 digital inputs module : SIB8

Features

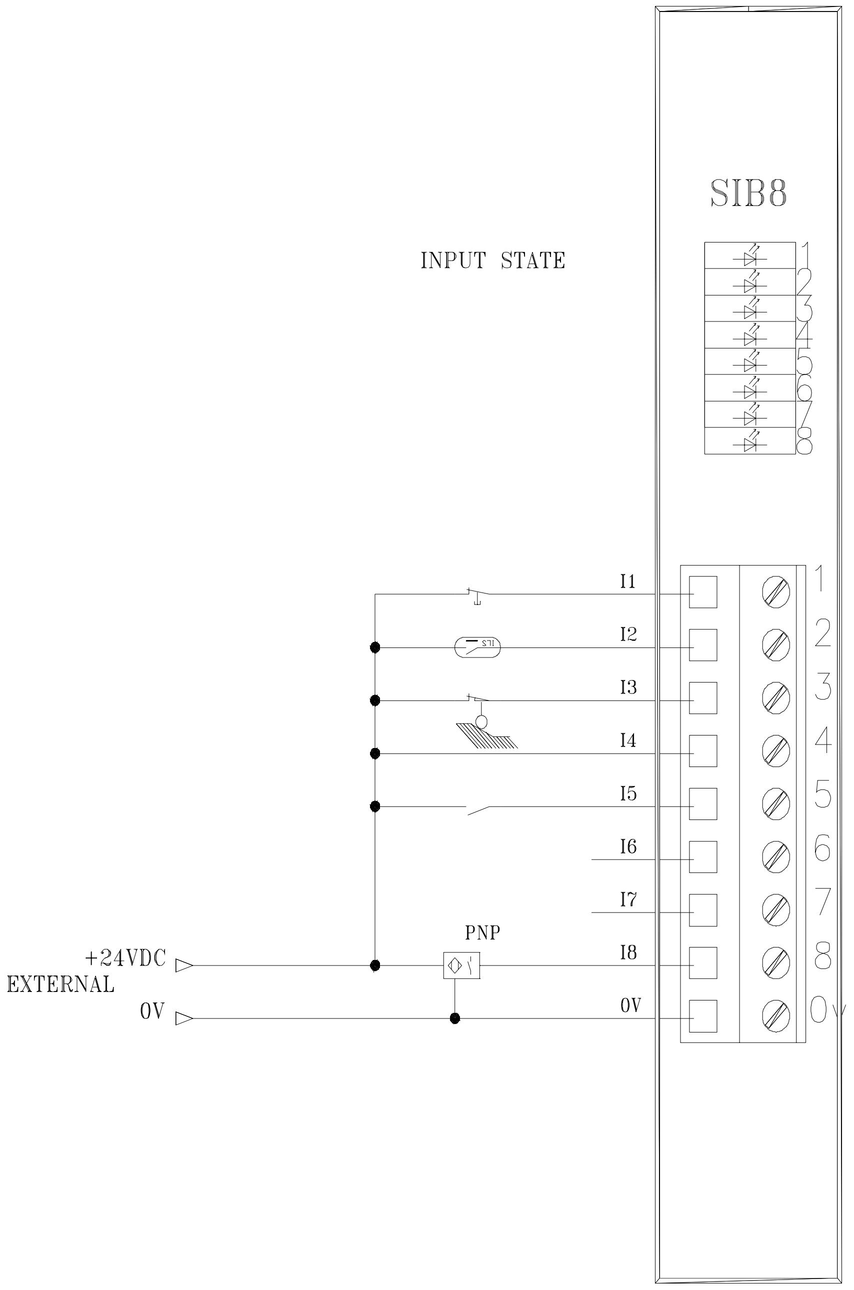
✧ Photo-coupled PNP type inputs.

✧ Input current : 12mA per input.

✧ Power supply voltage between 20 to 33 VDC

✧ Inputs read cycle depends on the value of the software filter parameter. (1 to 255ms)

Connections

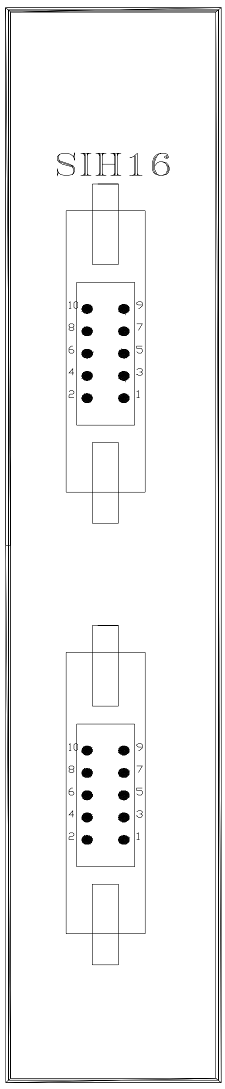


2-3-13- 16 digital inputs module : SIH16

Features

- Photo-coupled PNP type inputs
- Power supply voltage between 20 to 33 VDC
- MIHB8 - 8 inputs interface module directly compatible
- Inputs current : 12mA per input.
- Inputs read cycle depends on the value of the software filter parameter. (1 to 255ms)

Connections



CONNECTOR A HE10

1	INPUT 8
2	INPUT 7
3	INPUT 6
4	INPUT 5
5	INPUT 4
6	INPUT 3
7	INPUT 1
8	INPUT 2
9	0V
10	0V

CONNECTOR B HE10

1	INPUT 16
2	INPUT 15
3	INPUT 14
4	INPUT 13
5	INPUT 12
6	INPUT 11
7	INPUT 9
8	INPUT 10
9	0V
10	0V

2-3-14- 24 digital inputs module : SIH24

Features

↳ PNP type photo-coupled inputs

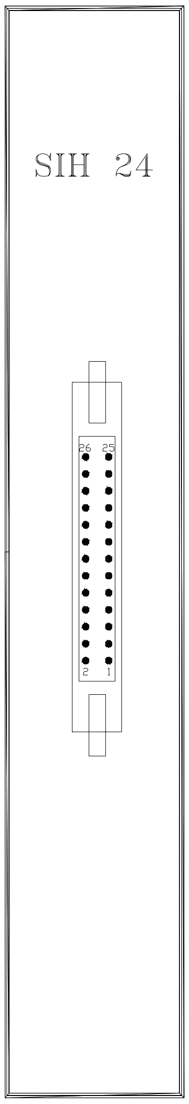
↳ Power supply voltage between 20 to 33 VDC

↳ MIHB24 - 24 inputs interface module directly compatible

↳ Inputs current : 12mA per input.

↳ Inputs read cycle depends on the value of the software filter parameter. (1 to 255ms)

Connections



CONNECTOR HE10

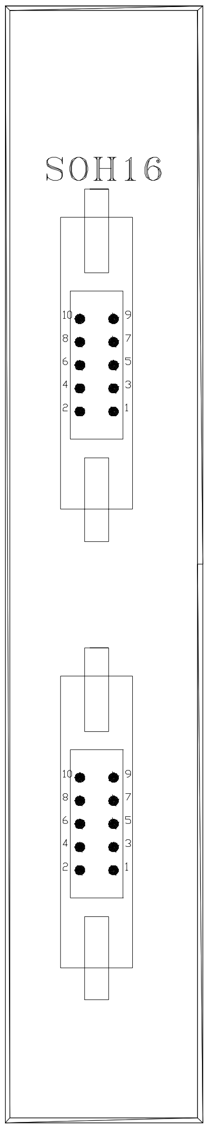
1	0V
2	0V
3	INPUT 12
4	INPUT 24
5	INPUT 11
6	INPUT 23
7	INPUT 10
8	INPUT 22
9	INPUT 9
10	INPUT 21
11	INPUT 8
12	INPUT 20
13	INPUT 7
14	INPUT 19
15	INPUT 6
16	INPUT 18
17	INPUT 5
18	INPUT 17
19	INPUT 4
20	INPUT 16
21	INPUT 3
22	INPUT 15
23	INPUT 2
24	INPUT 14
25	INPUT 1
26	INPUT 13

2-3-15- 16 digital outputs module : SOH16

Features

- Static isolated outputs (unprotected)
- Open collector (NPN type)
- Power supply voltage between 20 to 33 VDC
- MRHB 8 - 8 relays interface module directly compatible
- Outputs refresh cycle : 1ms

Connections



CONNECTOR A HE10

1	0V
2	OUTPUT 6
3	OUTPUT 7
4	+24VDC
5	OUTPUT 8
6	OUTPUT 5
7	OUTPUT 1
8	OUTPUT 4
9	OUTPUT 2
10	OUTPUT 3

CONNECTOR B HE10

1	0V
2	OUTPUT 14
3	OUTPUT 15
4	+24VDC
5	OUTPUT 16
6	OUTPUT 13
7	OUTPUT 9
8	OUTPUT 12
9	OUTPUT 10
10	OUTPUT 11

2-3-16- 8 digital outputs module : SOBP8

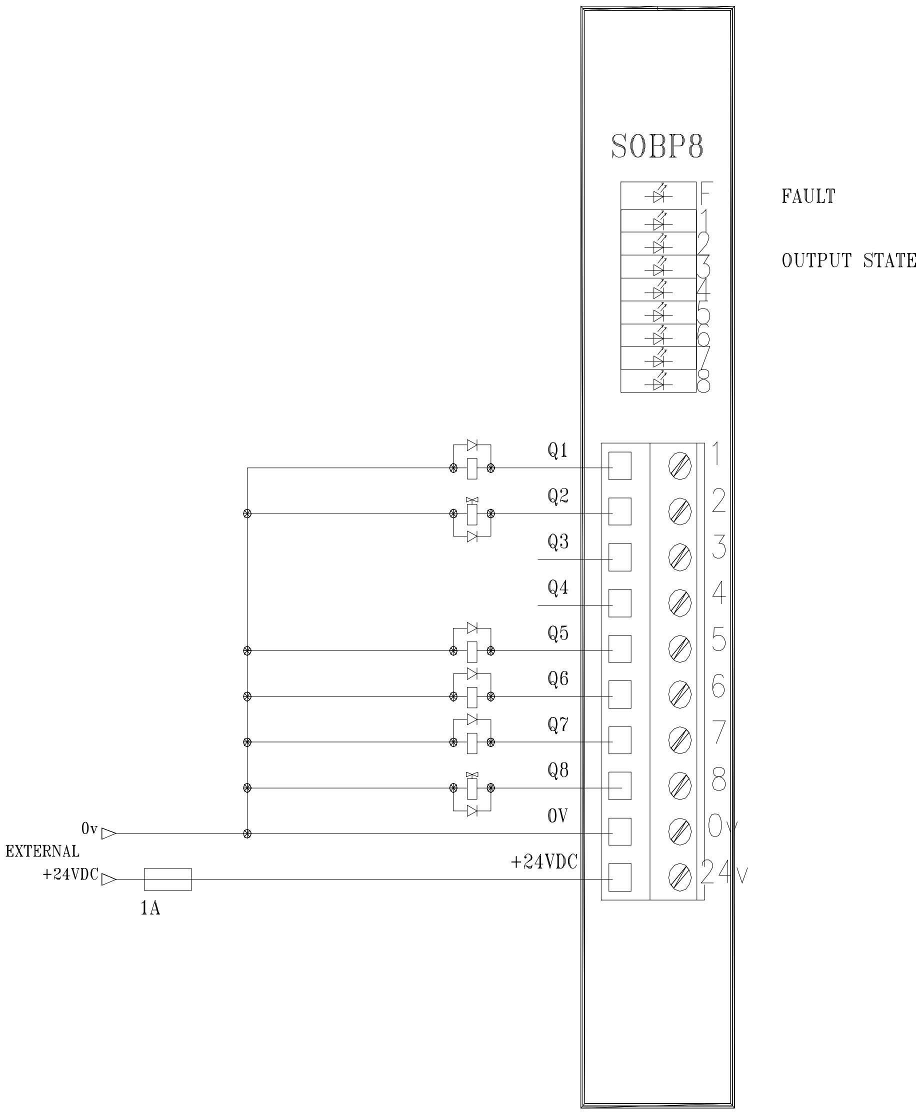
Features

- PNP (PLC compatible) photo-coupled outputs.
- Outputs current : 350mA maxi per output.
- Total outputs current on 8 outputs : 800mA maxi.
- Thermal and short circuit protected
- Maximal peek point : 350mA
- Outputs refresh cycle : 1ms

Remarks :

- If a short circuit is detected, 24 V power supply must be shut down to restart module.
- Add clamping diodes on inductives loads
- Capacitive loads prohibited

Connections

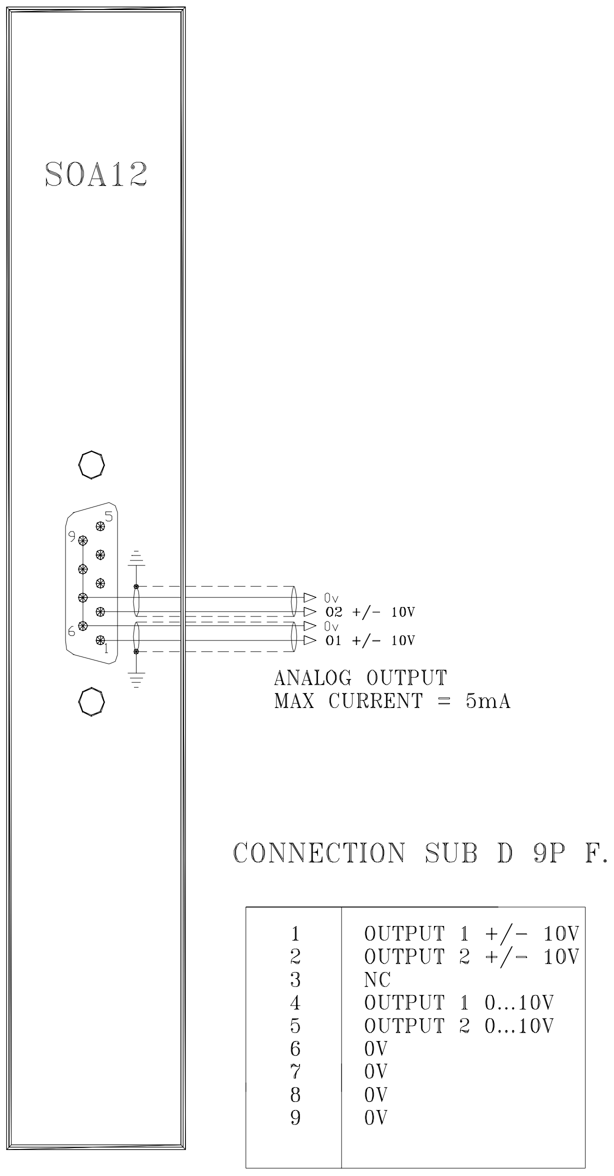


2-3-17- 2 analogue outputs : SOA12

Features

- Photo-coupled outputs
- Output voltage +/-10V or 0...10V
- Resolution (1 LSB) 4.88 mV
- Maximal output current : 5mA
- 12 bits digital to analogue converter
- Outputs refresh cycle : 9ms

Connections



CONNECTION SUB D 9P F.

1	OUTPUT 1 +/- 10V
2	OUTPUT 2 +/- 10V
3	NC
4	OUTPUT 1 0...10V
5	OUTPUT 2 0...10V
6	0V
7	0V
8	0V
9	0V

2-3-18- 4 analogue inputs module : SIA14

Features

⬂ Photo-coupled inputs

⬂ Input voltage +/-10 V

⬂ Maximal input voltage : +/-20V

⬂ Input impedance : 20KΩ

⬂ Resolution (1 LSB) 4.88 mV

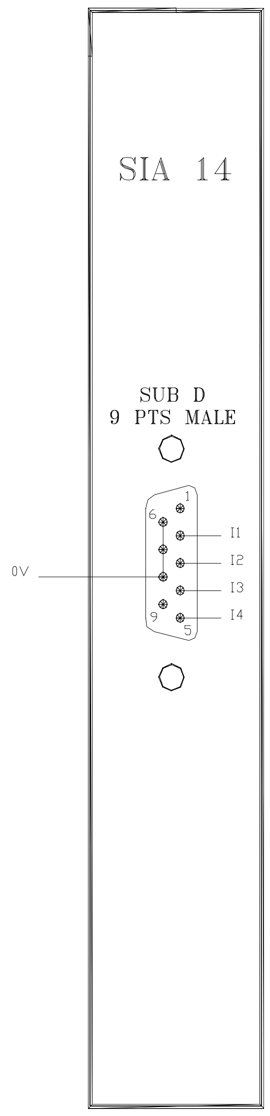
⬂ 12 bits analogue to digital converter

⬂ Inputs cycle read : 9 ms

Connections

CONNECTION SUB D 9P MALE

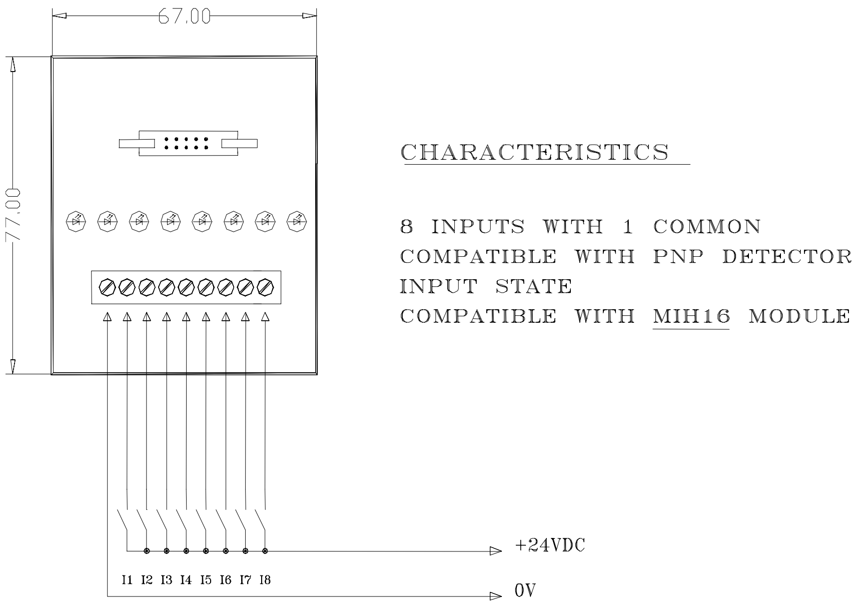
1	NC
2	INPUT 1
3	INPUT 2
4	INPUT 3
5	INPUT 4
6	0v
7	0v
8	0v
9	NC



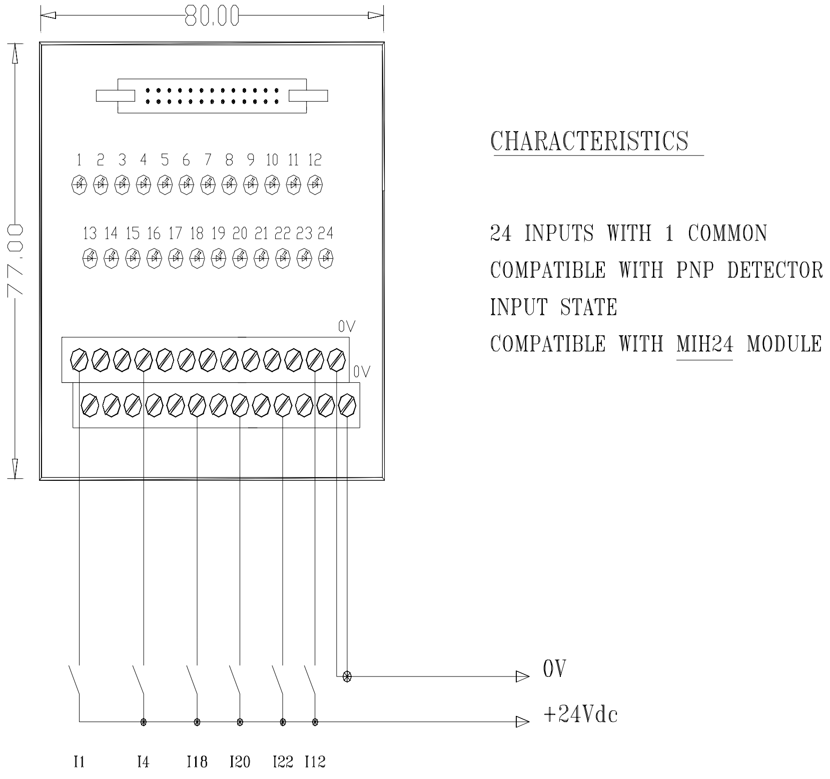
2-3-19- CANopen_board_module_SCAN

For the configuration and the programmation of this board, see the chapter CANopen.

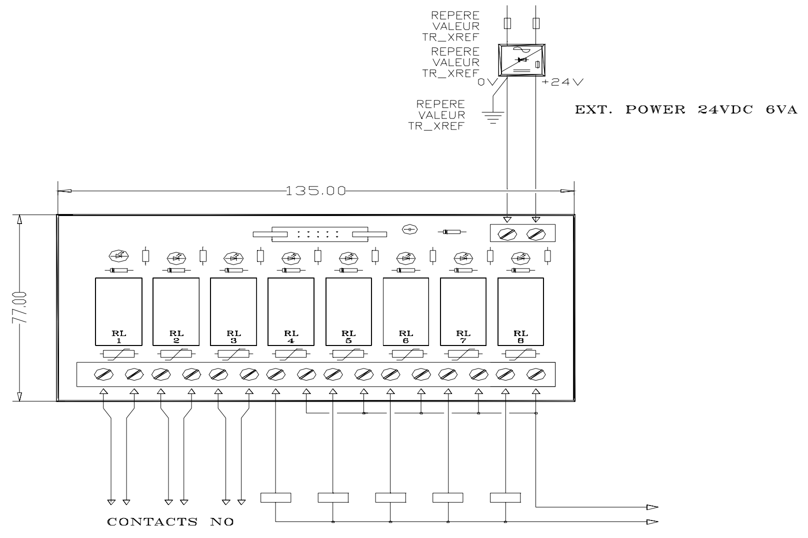
2-3-20- 8 inputs interface module : MIHB8



2-3-21- 24 inputs interface module : MIHB24



2-3-22- 8 relays interface module : MRHB8

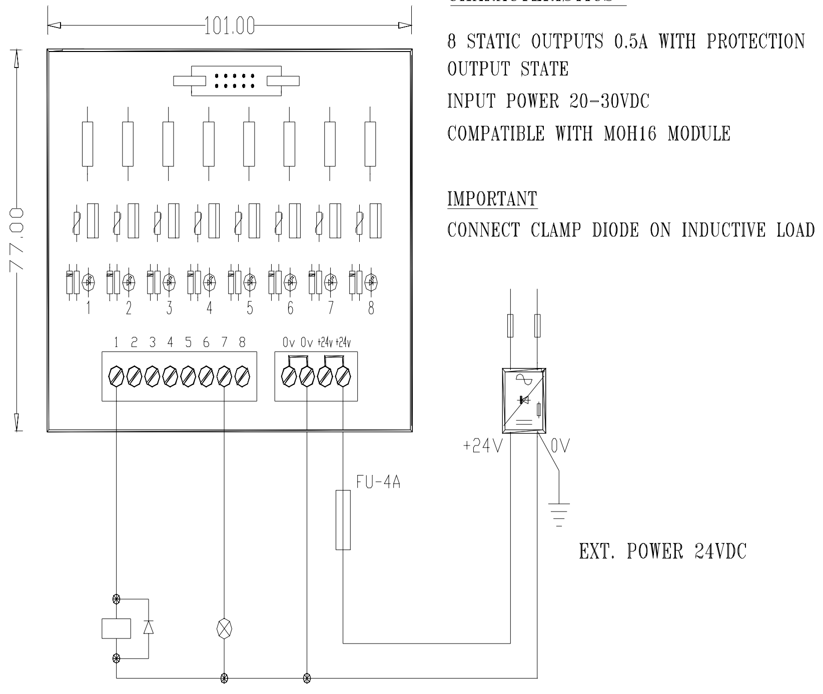


CHARACTERISTICS

8 OUTPUTS RELAY 5A (no) WITH VARISTOR PROTECTION
2 PIN PER OUTPUT
OUTPUT STATE
INPUT POWER 20–30VDC 6VA
COMPATIBLE WITH MOH16 MODULE

IMPORTANT
CONNECT CLAMP DIODE ON INDUCTIVE LOAD (dc voltage)
CONNECT RC FILTER ON INDUCTUVE LOAD (ac voltage)

2-3-23- 8 static outputs interface module : MSHB8

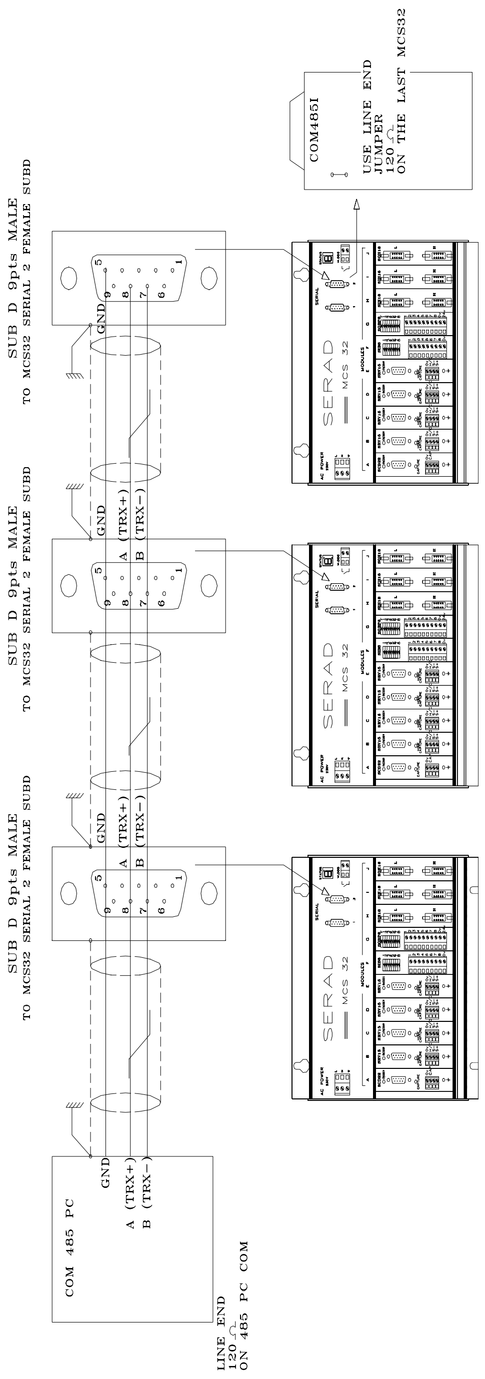


CHARACTERISTICS

8 STATIC OUTPUTS 0.5A WITH PROTECTION
OUTPUT STATE
INPUT POWER 20–30VDC
COMPATIBLE WITH MOH16 MODULE

IMPORTANT
CONNECT CLAMP DIODE ON INDUCTIVE LOAD

2-3-24- RS485 interface module : COM485



2-4- Getting started

2-4-1- Getting started

MCS 32 starting follows this approach :

- ➯ Define board placement in the setup screen.
- ➯ Setup each card.
- ➯ Send setup in MCS using "Send setup" menu.
- ➯ Define the global variables.
- ➯ Send global variables value in MCS.
- ➯ Write each task.
- ➯ Compile and transfer tasks in MCS.
- ➯ If the setup is modified it must be sent one more time.

2-5- Tuning method

2-5-1- Tuning method

There are two parts in a servo-control axis :

1/a : Drive - motor - tachometer (for direct current type motor).

/b : Drive - motor - sensor (for Brushless motor)

2 : MCS - encoder (or position measuring rule).

The encoder gives the axis position to MCS.

The MCS supplies an analogue voltage to drive for control motor.

1) Axis adjustment - no looped system

The part drive-motor-tacho must be, first of all, adjusted.

If the motor axis is turning as soon as power is ON, reverse the tachometer's cables (for direct current type motor).

Connect an adjustable voltage (-9v to +9v) to analogue drive's input and verify working order (Nominal motor's velocity for +9v voltage).

2) Axis adjustment - looped system

Connect the encoder to the MCS.

Connect MCS's analogue voltage output to drive's analogue input.

Fix those parameters values : proportional gain (GPROP_P) = 1000; derived GAIN (GDEV_P) = 0 ; integral gain (GINT_P) = 0 ; encoder resolution ; encoder steps per revolution ; velocity ; acceleration ; deceleration

If the motor axis is turning as soon as power is ON, several solutions are possible :

Adjust the parameter "Encoder direction invert" or the parameter "Voltage command direction invert".

The system must be stable

<u>Counting direction</u>

Adjust the parameters values "Voltage command direction invert" and "Encoder direction invert" to obtain a direction of correct counting.

P.I.D.

Regulate the P.I.D parameters. In most systems, only a proportional gain value is needed. Derived gain, integral gain and integral limit are equal to zero.

Increase gain value for a better accuracy, decrease it for a better stability.

If the axis card is used with sychronized functions, adjust the anticipation gain, velocity gain and acceleration gain to obtain a null error.

Resolution and unit / encoder turn :

The resolution and the unit / encoder turn fits with the points number of the encoder used x 4 and by advance of the axis for an encoder revolution.If these parameters are erroneous, the quotation mentioned on the display will not fit with the real quotation.

Maximal following error : indicate the maximal gap admitted between desired and real position ; overtaking this value involves a system failure. **Increasing this value involves few failure but may be dangerous for persons and machine because of system failure delay.**

Minimum position error : This value defines the position window for which the system has reached position

3- MCB EX SOFTWARE

3-1- Installation

3-1-1- System configuration

Minimal configuration :

⤷ PC 486 DX2 66

⤷ RAM 8 Mb

⤷ Hard disk (35 Mb available)

⤷ Microsoft® Windows™ 95 or Microsoft® Windows™NT 4.0 (service pack 3)

⤷ CD-ROM (2X)

⤷ SVGA colour display

⤷ Mouse or other peripheral pointing system

Required configuration :

⤷ PC Pentium® 75 or greater

⤷ RAM 16 Mb

⤷ Hard disk (35 Mb available)

⤷ Microsoft® Windows™ 95 or Microsoft® Windows™NT 4.0 (service pack 3)

⤷ CD-ROM (4X)

⤷ SVGA colour display

⤷ Mouse or other peripheral pointing system

This software run on Microsoft® Windows NT™. But, it doesn't run on Unix, Mac, MS-DOS and Microsoft® Windows 3.11.

3-1-2- Installation procedure

The Motion Control Basic software is provided on floppy disk (if required) or in a CD-ROM with the MCS system. The installation procedure is described below :

⤷ Verify the required configuration before the software installation

⤷ Insert the floppy disk or the CD-ROM in the appropriate drive.

⤷ In the menu Démarrer, select Exécuter... .

⤷ In the « Execute » dialog box , select Parcourir... .

⤷ In the « Parcourir » dialog box, select the drive where the floppy disk or CD-ROM is.

⤷ Select Setup.exe then Ouvrir in the « Parcourir » dialog box.

⤷ Select OK in the « Execute » dialog box.

⇨ The installation software is running.

⤷ In the beginning of the installation, there are some dialog box to drive the installation :

- Destination folder

- Installation type (Typical, compact or custom)
- Select the program manager

' Warning : only one level of folder can be created.

⇨ **The file installation starts and is indicated by the evolution of a progress bar.**

⇨ **The installation finishes with the adding of MCB icon in the program manager.**

3-1-3- Upgrade from previous versions

A program wrote with a previous version can work on a new version if a compilation is done. MCB software only works with operating system provided in the OS directory of the installation directory of the software. By default this directory is « **C:\Program Files\Serad\Mcb EX** ».

Operation system installation is :

↳ Connect MCS SERIAL1 communication port on COM1 or COM2 of the PC

↳ Run MCBEX's application, go to OPEN PROJET in PROJECT then in OPTIONS -> OPERATING SYSTEM, click on UPDATE. If you want to update by DOS, follow the next instructions.

↳ On Windows 95 or greater, open a DOS windows

↳ With the DOS command, take place in the OS folder

↳ Execute the command : INSTALL < Serial port of PC>

For a serial plug on COM1 : INSTALL COM1

⇨ Installation starts with old operating system erasure. The « Waiting for erasure » message appears on PC screen and 'c' char is displayed on MCS's STATUS.

⇨ Then, programming is starting and 'P' displayed on MCS's STATUS.

⇨ When programming is done, MCS restarts with E23 on STATUS because there is no user program.

↳ Compile tasks and transfer in the MCS.

3-2- Architecture

3-2-1- Folders

↳ Gfx: contains all the chart.

↳ Lib : contains all the file with DLL extensions for the running of the software

↳ Str : contains the language file

↳ OS : contains a copy of the MCS operating system

↳ Help : contains all the help file for the MCS32EX and MCBEX.

↳ Project : contains all the files and folders of the user's project

3-2-2- Project contents

The « project » folder is a reserved folder for the user's project. Each project is composed on a « ProjectName.prj » file and a « ProjectName.rep » folder. This folder have the file below :

↳ a configuration file (ProjectName.cfg)

↳ a global variable definition file (ProjectName.var)

🗲 a global constant definition file(ProjectName.cst)

🗲 a file per basic task (ProjectName.tsk)

🗲 an extra file per ladder task (ProjectName.lad)

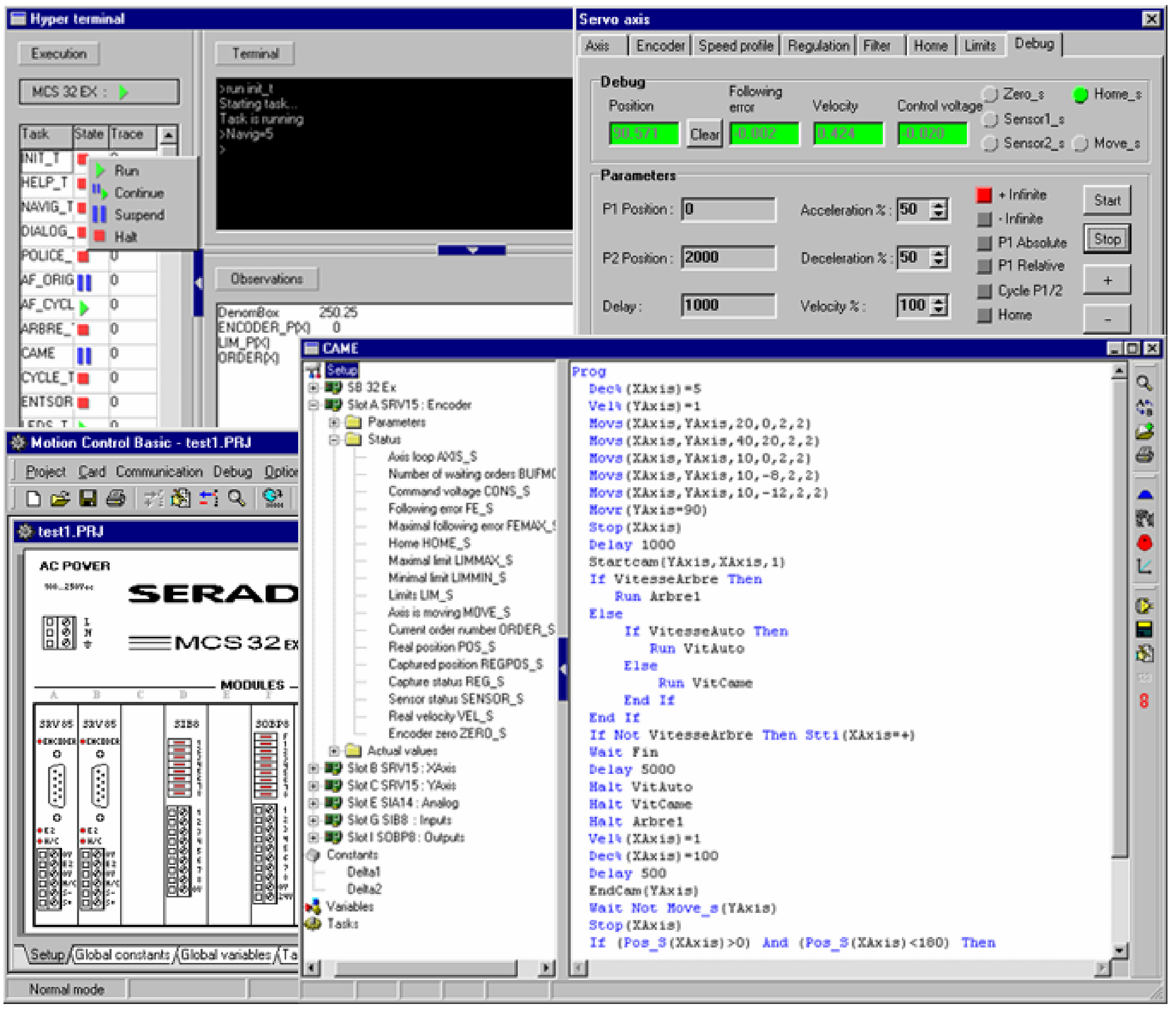
🗲 The result of compilation gives some binary file (ProjectName.bin and ProjectName.b00 to ProjectName.b07). The sum of the task length (b0*) gives the length of the compiled task.

🗲 Other files (.map, .uti) for the MCB internal management

3-3- Description

3-3-1- Initial screen

MCB EX software is defined by a main window with a main menu, an icons bar and the multiwindows. The property of multiwindows provides to users the possibilities to go to another window without to changing it.



Task	State	Trace
INIT_T	🟥	0
HELP_T	🟥	
NAVIG_T	🟥	
DIALOG_	🟥	
POLICE_	🟥	0
AF_ORIG	❚❚	0
AF_CYCL	▶	0
ARBRE_	🟥	0
CAME	❚❚	0
CYCLE_T	🟥	0
ENTSOR	🟥	0
LEDS_T	▶	0

```
>run init_t
Starting task...
Task is running
>Navig=5
>
```

```
DenomBox        250.25
ENCODER_P(X)    0
LIM_P(X)
ORDER(X)
```

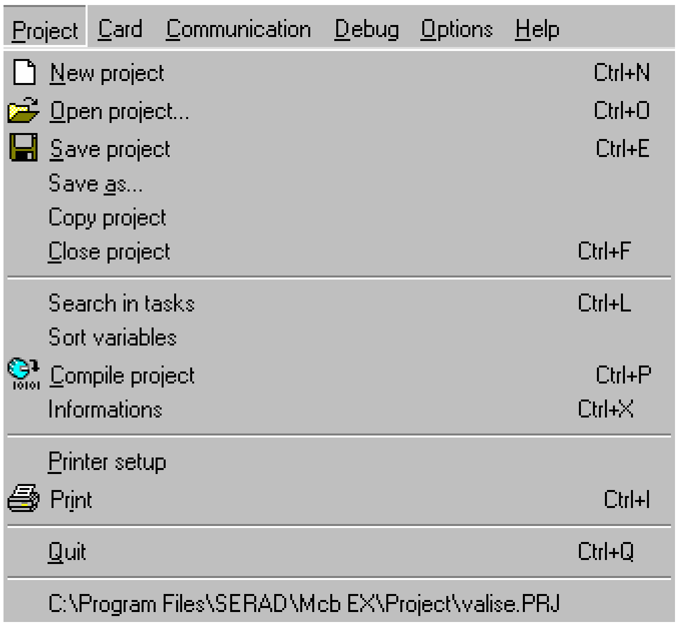
Debug: Position 90.571, Following error -0.002, Velocity 0.424, Control voltage -0.020; Zero_s, Home_s ●, Sensor1_s, Sensor2_s, Move_s

Parameters: P1 Position : 0, Acceleration % : 50, + Infinite 🟥, - Infinite, P2 Position : 2000, Deceleration % : 50, P1 Absolute, P1 Relative, Delay : 1000, Velocity % : 100, Cycle P1/2, Home

```
Prog
  Dec%(XAxis)=5
  Vel%(YAxis)=1
  Movs(XAxis,YAxis,20,0,2,2)
  Movs(XAxis,YAxis,40,20,2,2)
  Movs(XAxis,YAxis,10,0,2,2)
  Movs(XAxis,YAxis,10,-8,2,2)
  Movs(XAxis,YAxis,10,-12,2,2)
  Movr(YAxis=90)
  Stop(XAxis)
  Delay 1000
  Startcam(YAxis,XAxis,1)
  If VitesseArbre Then
     Run Arbre1
  Else
      If VitesseAuto Then
         Run VitAuto
      Else
          Run VitCame
      End If
  End If
  If Not VitesseArbre Then Stti(XAxis=+)
  Wait Fin
  Delay 5000
  Halt VitAuto
  Halt VitCame
  Halt Arbre1
  Vel%(XAxis)=1
  Dec%(XAxis)=100
  Delay 500
  EndCam(YAxis)
  Wait Not Move_s(YAxis)
  Stop(XAxis)
  If (Pos_S(XAxis)>0) And (Pos_S(XAxis)<180) Then
```

3-4- Menus and icons

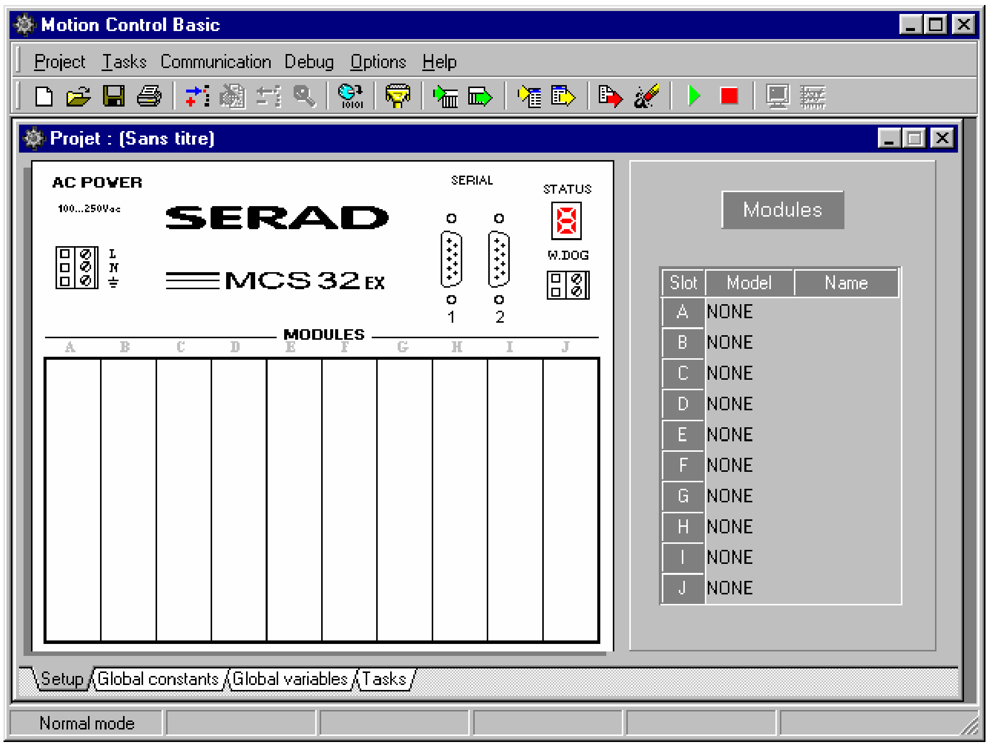
3-4-1- Project menu



New Project

Icon: 🗋

Action: This command defines a new project. The last running project is closed and a new configuration window appears.



Slot	Model	Name
A	NONE	
B	NONE	
C	NONE	
D	NONE	
E	NONE	
F	NONE	
G	NONE	
H	NONE	
I	NONE	
J	NONE	

Open project

Icon:

Action: This command opens the « Open Project » dialog box. The users can indicate the path of the project, he wants to load. This command closes the last running project on the validation of this dialog box.

Save project

Icon:

Action: This command saves the complete running project.

Save as…

Action: This command opens the "Save as" dialog box that allows user to change the path and the name of the project. This command creates a file and a folder with the name of the project and for the first the "prj" extension and the second the "rep" extension.

Copy project

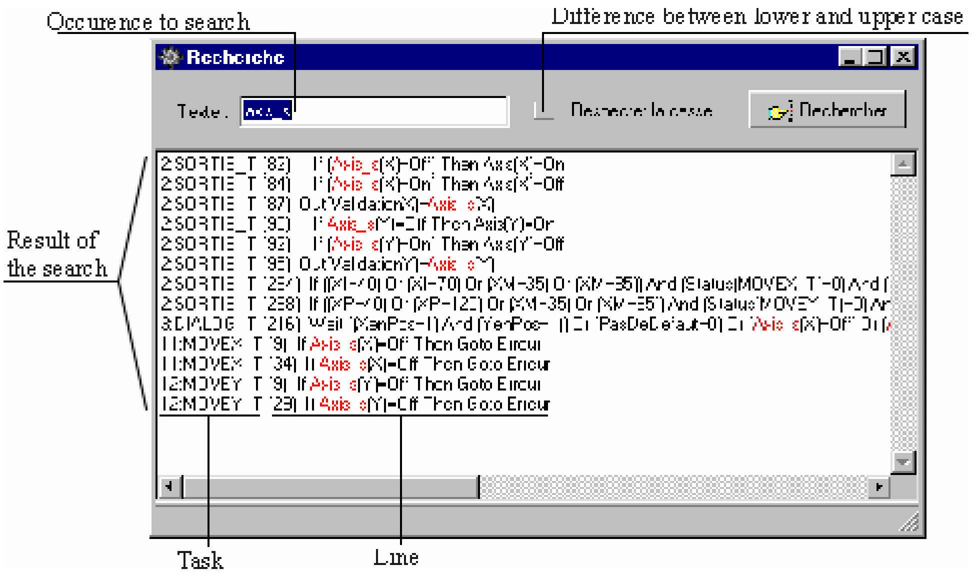
Action: This command has the same properties as Save as… . However, this command doesn't change the date of the project creation.

Close project

Action: This command closes the running project.

Search in tasks

Action: This command searches a text, a word or a part of a word in all the tasks of the project. A dialog box appears and gives all the functions to succeed. A double click on a result line of a search edits the basic task at the right line.



```
2:SORTIE_T (82)   If (Axis_s(X)=Off) Then Axis(X)=On
2:SORTIE_T (84)   If (Axis_s(X)=On) Then Axis(X)=Off
2:SORTIE_T (87) OutValidation(X)=Axis_s(X)
2:SORTIE_T (90)   If Axis_s(Y)=Off Then Axis(Y)=On
2:SORTIE_T (92)   If (Axis_s(Y)=On) Then Axis(Y)=Off
2:SORTIE_T (95) OutValidation(Y)=Axis_s(Y)
2:SORTIE_T (237) If ((XI=70) Or (XI=70) Or (XM=35) Or (XM=35)) And (StatusMOVEX_T=0) And (
2:SORTIE_T (238) If ((XP=70) Or (XP=120) Or (XM=35) Or (XM=35)) And (StatusMOVEY_T=0) An
3:DIALOG_T (216) Wait (XanPos=1) And (YanPos=1) Or (PasDeDefaut=0) Or (Axis_s(X)=Off) Or (
11:MOVEX_T (9) If Axis_s(X)=Off Then Goto Erreur
11:MOVEX_T (34) If Axis_s(X)=Off Then Goto Erreur
12:MOVEY_T (9) If Axis_s(Y)=Off Then Goto Erreur
12:MOVEY_T (23) If Axis_s(Y)=Off Then Goto Erreur
```

Sort variables

Action: This command sort globals variables. At first, we find saves globales variables in growing order of their address, then not saves globales variables in type order (bit, octet, string …). Inside the same type, a alphabetical order is doing.

Compile project

Icon:

Action: This command compiles the project. A first phase verifies the syntax of each task, the configuration of variables, etc…. When a task has a syntax error, the basic task is edited and the error is highlighted with the position of the cursor. It is possible to compile only one task : choose a task in the task's list, right's click on your mouse and select VERIFY SYNTAX.

Informations

Action: This command give detailed informations on project, on programme'memory and memory of uses datas.

Printer setup

Action: This command allows the user to define its print type (printer, paper, etc…). The paper orientation can't be changed.

Impression

Icon:

Action: This command print all of a custom project. The MCS configuration , all the basic's task and the ladder are printed.

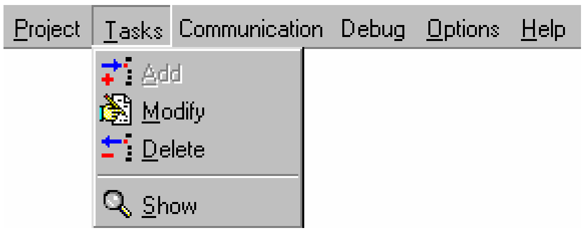
Quit

Action: This command quit the MCB EX software.

Projects list

Action : A click on a project of this list opens it.

3-4-2- Card menu



The four commands of this submenus act on the main windows of the project. The action of each one are different with the selected tab of the windows.

🖎 An adding, suppression or modification of an elements needs the project to be compiled again.

🖎 A modification of a parameter value needs the configuration to be sent to the MCS.

🖎 A modification on a global stored variable value needs the variables to be sent to the MCS.

Add

Icon:

Action : This command adds a board, a global constant, a global variable or a task according to the tab selected.

Modify

Icon:

Action : This command modifies the parameters of a board, a global constant, a global variable or a task according to the tab selected.

Delete

Icon:

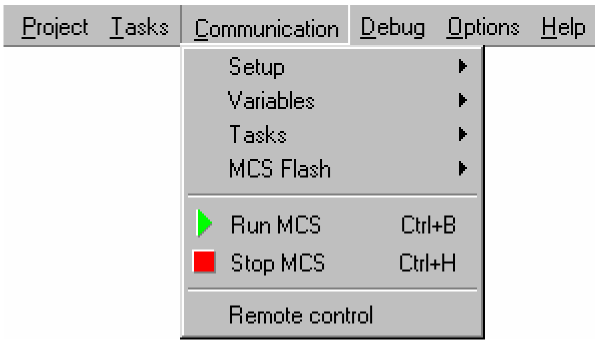
Action : This command deletes a board, a global constant, a global variable or a task according to the tab selected.

Show

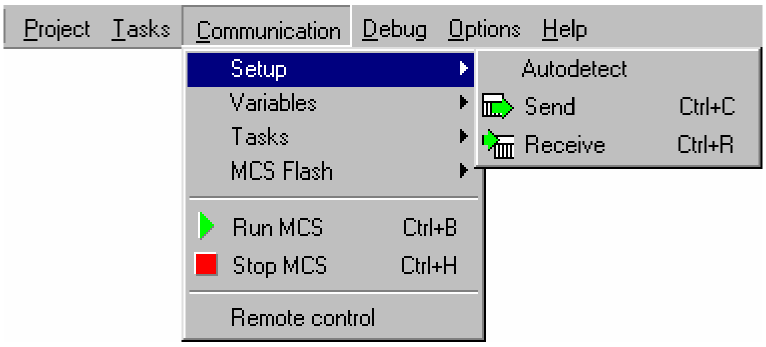
Icon:

Action : This command shows the parameters of a board, a global constant, a global variable or runs the ladder or basic editor according to the tab selected.

3-4-3- Communication menu



Setup



Autodetect

Action: With a new project, this command create automatically the configuration if the PC and the MCS are connected.

Send

Icon:

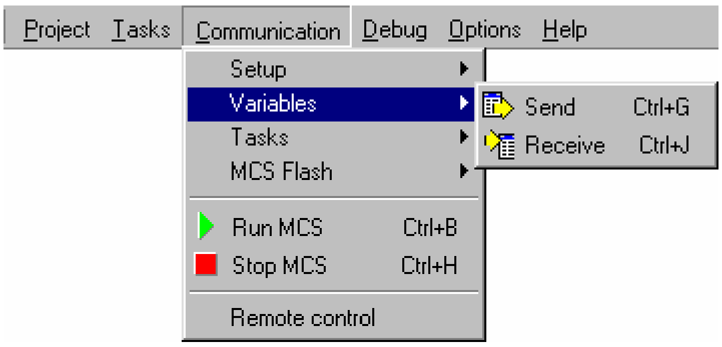
Action: The configuration sending initializes MCS with the parameters defined in the configuration screens of each board. At the beginning, there is a test between the configuration in the MCS and the configuration on the PC. If an error is detected, the transfer is stopped and a message appears with the card where the contents is incorrect. This command is necessary after an adding, deleting or modification of a board…

Receive

Icon:

Action: This command updates the parameters in the screens configuration of boards. The transfer begins with the test between the configuration in MCS and the configuration on the PC. If an error is detected, the transfer is stopped and a message appears with the card where the contents is incorrect. If you want to stored this configuration in the project, you need to execute the « Save as… » command.

Variables



Send

Icon:

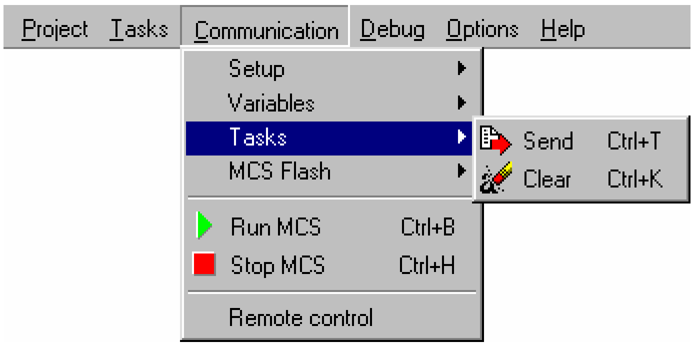
Action: the sending of stored variables initializes the value assigned to this variable in MCS. So, you needn't to initialize them in a program.

Receive

Icon:

Action: This command provides a copy of the stored variables in MCS in the PC. I you want to store this values of variables in the project, you need to execute the « Save as… » command.

Tasks



Send

Icon:

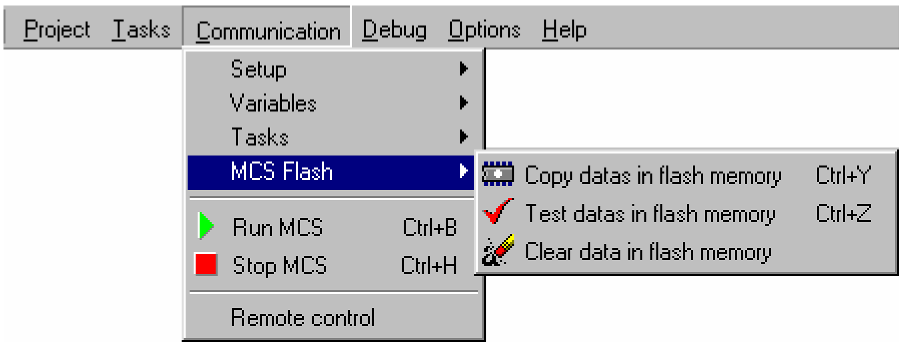
Action: This command sends all the compiled tasks in the MCS. The execution of the tasks is automatically launched after the transfer. The transfer begins with the clearing of the memory. During this phase, the "c" symbol appears on the MCS display status and a bar graph indicates the evolution of the transfer.

Clear

Icon:

Action: This command clears all the tasks in MCS memory. After the execution of this command, MCS indicates an error 23.

MCS Flash



Copy data in flash memory

Action: This command creates a backup in flash memory of the setup parameters and of the first 10000 stored variables in the non-volatile RAM memory. At each MCS power-on, a checksum is made to test the validity of the data in non-volatile RAM. If an error is detected, MCS transfer the flash memory backup in the non-volatile RAM and launches tasks. If there is no backup, MCS indicates an error 20.

Test data in flash memory

Action: This command indicates if a backup memory is present in flash memory.

Clear data in flash memory

Action: This command clear the data's copy in the flash's memory.

Run MCS

Icon: ▶

Action: This command launches the execution of the tasks in MCS.

Stop MCS

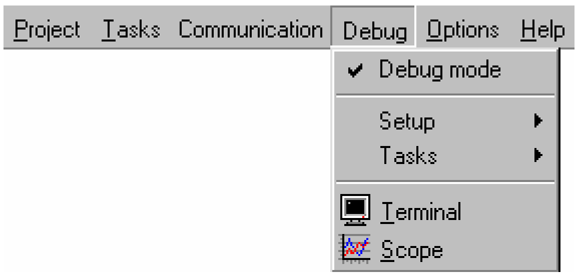
Icon: 🟥

Action: This command stops the execution of the tasks in MCS. WatchDog becomes OFF. All the servo board are in an open loop state (analogue command=0). The Security instruction has no effect on MCS.

Remote control

Action: With this command, you have access in mode Remote Control. You can drive the MCS32EX at distance with a modem and a telephone line (see chapter Remote Control).

3-4-4- Debug menu



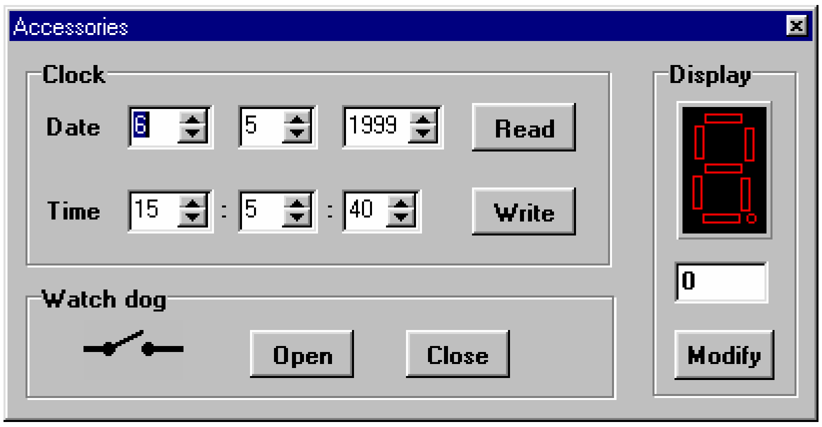
Debug mode

Action: This command allows the working of debug mode. On activate, all the command of this sub-menus are valid.

Setup

Action : This sub-menu displays the SB32EX debug window or the debug window of the slot selected. According to the board in the slot, the dialog box is different :

🖎 A dialog box with the state of the status display, the state of the watchdog and the time and date in MCS appears. All of these parameters can be modified. If one of this parameters is driven by an executed task, the manual modification of its state may be transitory.


Accessories

Clock

Date 6 5 1999 Read

Time 15 : 5 : 40 Write

Watch dog

Open Close

Display

0

Modify

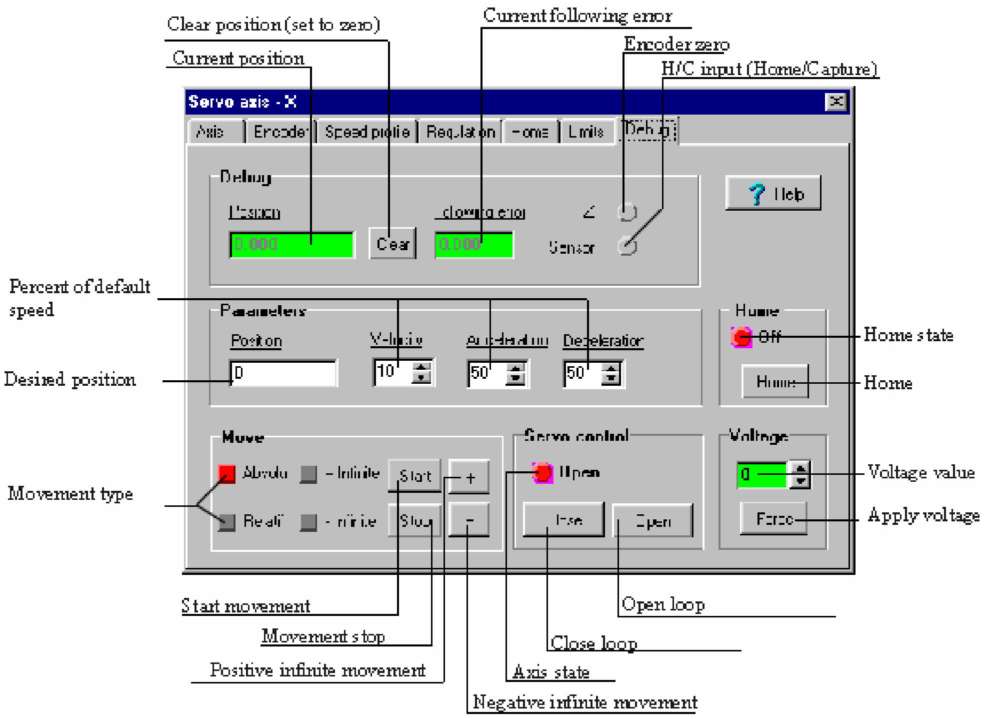
🖎 The debug windows of the digital inputs or outputs boards (SIB8, SIH16, SIH 24, SOBP8, SOH16) shows with leds the state of each input or output of a board. A click on a led modifies the state of the input or output.

🖎 The debug windows of the analogue inputs or outputs boards (SIA14, SOA12) shows the level of voltage on each input or output of a board. Each level of voltage can be modified by user.

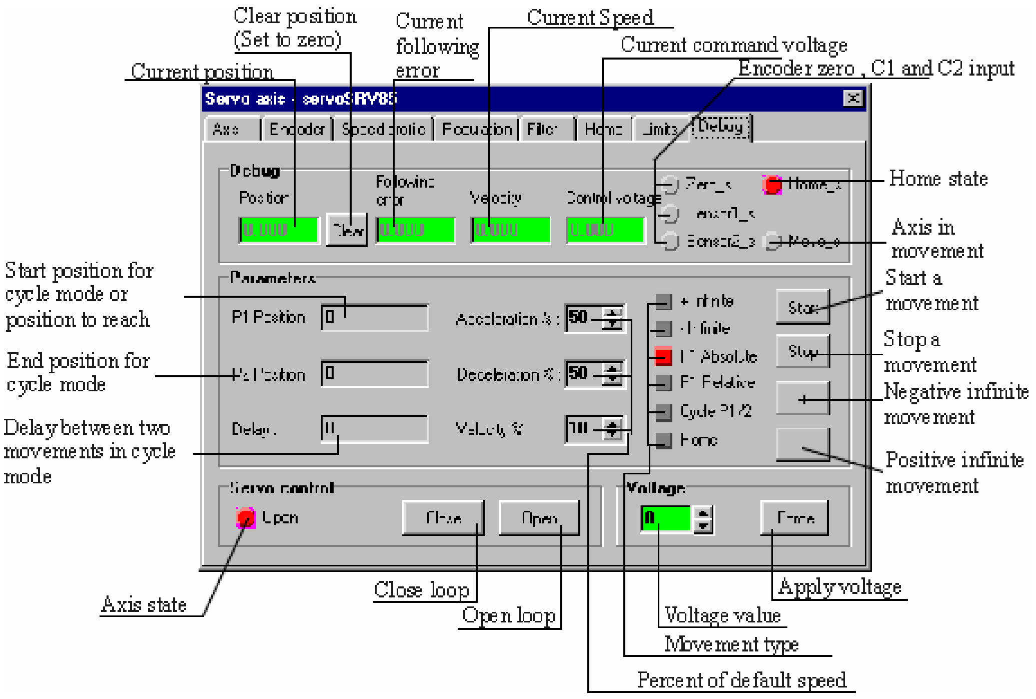
🖎 The debug windows of the encoder boards (SCD22, SCD2224) shows the position, the state of the Z signal and the state of the sensor in order to simplify the control of wiring. The position can be cleared.

🖎 The debug windows of the SSI encoder boards (SSI22) shows the position, the state of the sensor in order to simplify the control of wiring.

🖎 The debug windows of the SRV15 or SRV1524 servo board allows to test the axis position loop. It is better to begin to check the motor/drivebehaviour by applying command voltage between +10V and –10V (The axis must be in open loop mode). Then, we can passed in a close loop mode to adjust the loop parameters. MCS32 takes modifications made on the different pages of setup into account when the Debug page is selected. If a parameter is wrong, the screen with this parameter is displayed and the parameter is selected. When we use the debug mode, the parameters, displayed with a green background, are saved in the MCS. When a parameter is modified, it is automatically sent to MCS. To save the modification, you should save the project before you quit.

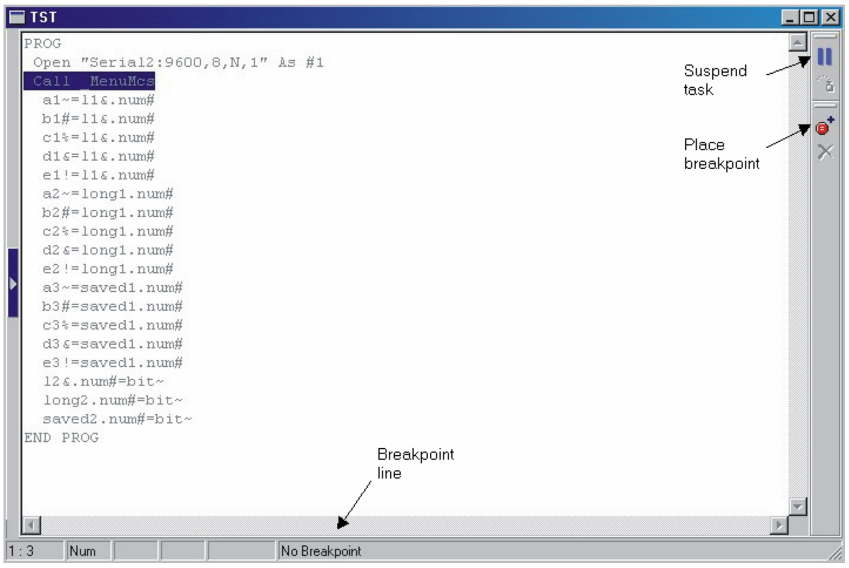


✍ The debug windows of the SRV85 servo board allows to test the axis position loop. It is better to begin to check the motor/drive behaviour by applying command voltage between +10V and –10V (The axis must be in open loop mode). Then, we can passed in a close loop mode to adjust the loop parameters. MCS32 takes modifications made on the different pages of setup into account when the Debug page is selected. If a parameter is wrong, the screen with this parameter is displayed and the parameter is selected. When we use the debug mode, the parameters, displayed with a green background, are saved in the MCS. When a parameter is modified, it is automatically sent to MCS. To save the modification, you should save the project before you quit.



⇘ The debug windows of the SSI15 servo board allows to test the axis position loop. It is better to begin to check the motor/drive behaviour by applying command voltage between +10V and –10V (The axis must be in open loop mode). Then, we can passed in a close loop mode to adjust the loop parameters. MCS32 takes modifications made on the different pages of setup into account when the Debug page is selected. If a parameter is wrong, the screen with this parameter is displayed and the parameter is selected. When we use the debug mode, the parameters, displayed with a green background, are saved in the MCS. When a parameter is modified, it is automatically sent to MCS. To save the modification, you should save the project before you quit. The debug window of the SSI15 board is the same like SRV15 or SRV1524 board without the state of the Z signal.

Tasks

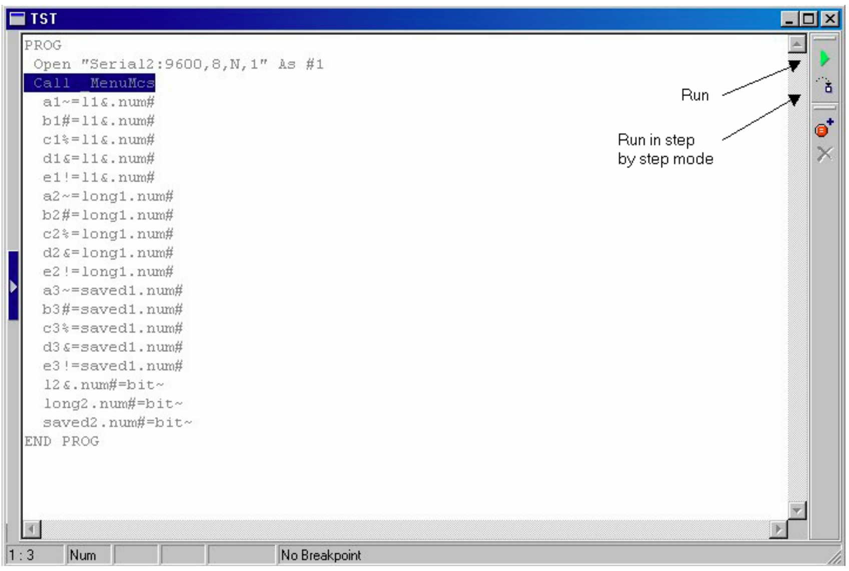


```
PROG
 Open "Serial2:9600,8,N,1" As #1
 Call  MenuMcs
  a1~=l1&.num#
  b1#=l1&.num#
  c1%=l1&.num#
  d1&=l1&.num#
  e1!=l1&.num#
  a2~=long1.num#
  b2#=long1.num#
  c2%=long1.num#
  d2&=long1.num#
  e2!=long1.num#
  a3~=saved1.num#
  b3#=saved1.num#
  c3%=saved1.num#
  d3&=saved1.num#
  e3!=saved1.num#
  l2&.num#=bit~
  long2.num#=bit~
  saved2.num#=bit~
END PROG
```

1 : 3 | Num | No Breakpoint

Action : In this sub-menu, there are all the tasks defined in the tasks tab. The validation of one of the tasks launches the basic editor in a debug mode. The basic code can't be modified. This mode allows the user to show the evolution of code trace if it was validate.

StepByStep

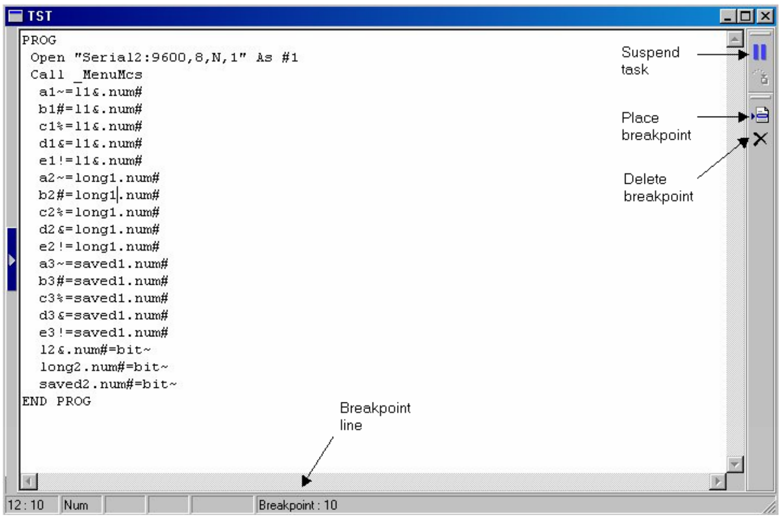


```
PROG
 Open "Serial2:9600,8,N,1" As #1
 Call  MenuMcs
  a1~=l1&.num#
  b1#=l1&.num#
  c1%=l1&.num#
  d1&=l1&.num#
  e1!=l1&.num#
  a2~=long1.num#
  b2#=long1.num#
  c2%=long1.num#
  d2&=long1.num#
  e2!=long1.num#
  a3~=saved1.num#
  b3#=saved1.num#
  c3%=saved1.num#
  d3&=saved1.num#
  e3!=saved1.num#
  l2&.num#=bit~
  long2.num#=bit~
  saved2.num#=bit~
END PROG
```

1 : 3 | Num | No Breakpoint

Action : This command allow to run the programm in step by step mode and control the good functionning of each basic in the task.

Breakpoint



```
PROG
 Open "Serial2:9600,8,N,1" As #1
 Call _MenuMcs
  a1~=l1&.num#
  b1#=l1&.num#
  c1%=l1&.num#
  d1&=l1&.num#
  e1!=l1&.num#
  a2~=long1.num#
  b2#=long1.num#
  c2%=long1.num#
  d2&=long1.num#
  e2!=long1.num#
  a3~=saved1.num#
  b3#=saved1.num#
  c3%=saved1.num#
  d3&=saved1.num#
  e3!=saved1.num#
  l2&.num#=bit~
  long2.num#=bit~
  saved2.num#=bit~
END PROG
```

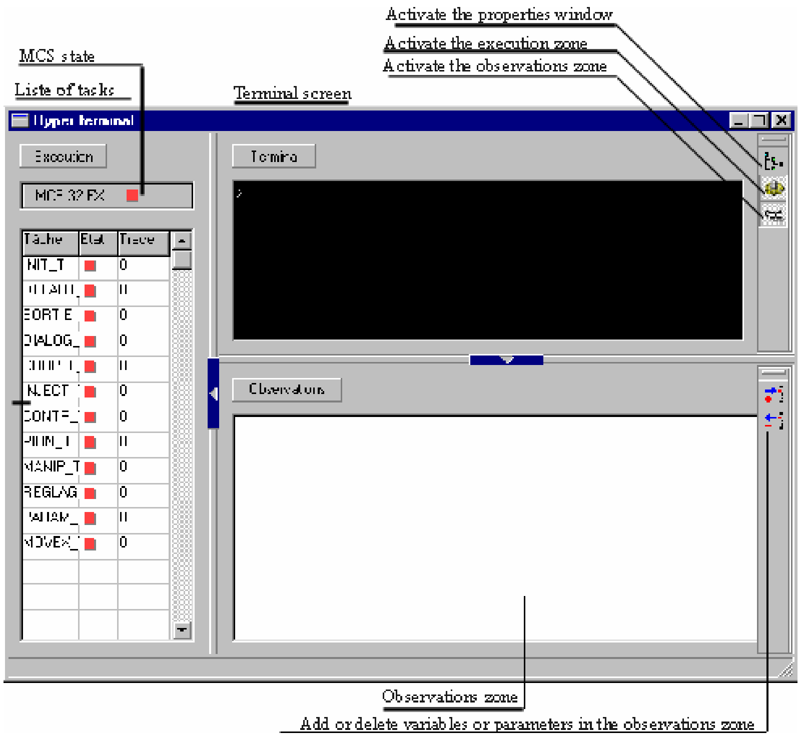
```
12 : 10   Num   Breakpoint : 10
```

Action : This command allow to choose a ligne in the task where you want that the programm stop for control some parameters..

Terminal

Icon:

Action: This command launches the hyper terminal viewer. This tool allows to ask the state of MCS or to read and write the local and global variables, parameters, inputs and outputs.


Activate the properties window
Activate the execution zone
Activate the observations zone
MCS state
Liste of tasks
Terminal screen
Hyper terminal
Execution
MCS 32 EX
Terminal
Observations
Observations zone
Add or delete variables or parameters in the observations zone

Tache	Etat	Trace
NIT_T	🟥	0
[illegible]	🟥	0
EORT E	🟥	0
DIALOG_	🟥	0
[illegible]	🟥	0
N.ECT	🟥	0
CONTE_	🟥	0
[illegible]	🟥	0
MANIP_T	🟥	0
REGLAG	🟥	0
[illegible]	🟥	0
MOVEX_	🟥	0

The terminal window has a main window and two other optional windows : the « observations » window and the « status » window.

⇨ The main window allows the reading and the writing in real-time of all the variables and parameters of MCS. To access to these information, there are some functions :

↳ Print <Variable or Parameter Name> : display the value of a variable or a parameter.

↳ <Variable or Parameter Name>=<Value> : assign a value to a variable or a parameter

↳ STATUS : State of the tasks

↳ RUN <Task name> : execute a task

↳ HALT <Task name> : stop a task

↳ SUSPEND <TaskName> : suspend the execution of a task

↳ CONTINUE <TaskName> : continue the execution of a task

↳ CLS : Clear the dialog zone

↳ RESTART : restart MCS

↳ EXIT : close the terminal

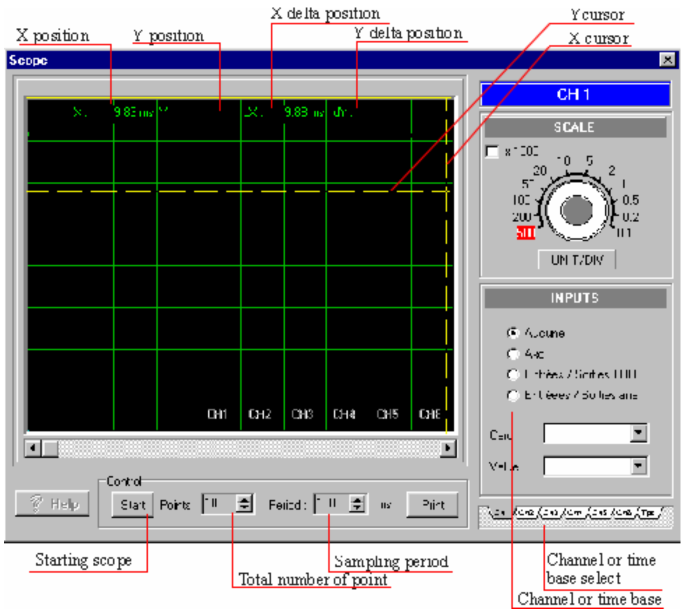
For an easy way to edit the name of variables or parameters, the terminal has a window of MCS properties. In this window, we can find all the parameters of each board, global variables and local variables of each task. The parameter or variable name appears on the terminal window on a double click on one of this variable or parameter.

⇨ The « observations » window show the state of variables in a continuous mode. The maximum of variables or parameters to show is limited to 100. Two commands allow user to add or delete a variable. The adding command launches the execution of the MCS properties window. The variable or parameter must be choose among board, global variables or task. You can save or load this 40 variables as a file.

⇨ The « status » window shows the state of the MCS and the state of the task in a continuous mode. MCS can be remotely driven with a click on the play or stop icon displayed. A click change the icon displayed. The tasks can be remotely driven too separately. The state of each task may be : « Stop », « Start », « Suspend » or « Continue ». The modification of the state is obtained with a click on the state icon of the task. The « trace » row indicates the executed line of a task. Before, the code trace must be validate, the project compiled again and the task sent. You can also have a notion of the system's resources used for each task.

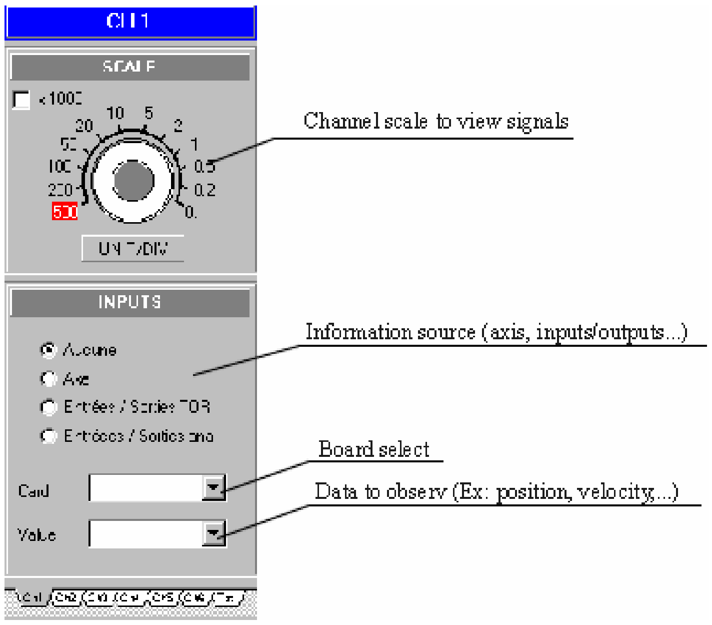
Scope

Icon:	
Action:	This command launches the scope. This tool is able to capture all the information of axis board or input/output board. This tool is able to store six different variable. The scope is configured in three part : the visualization screen, the acquisition configuration zone and the channels configuration zone.

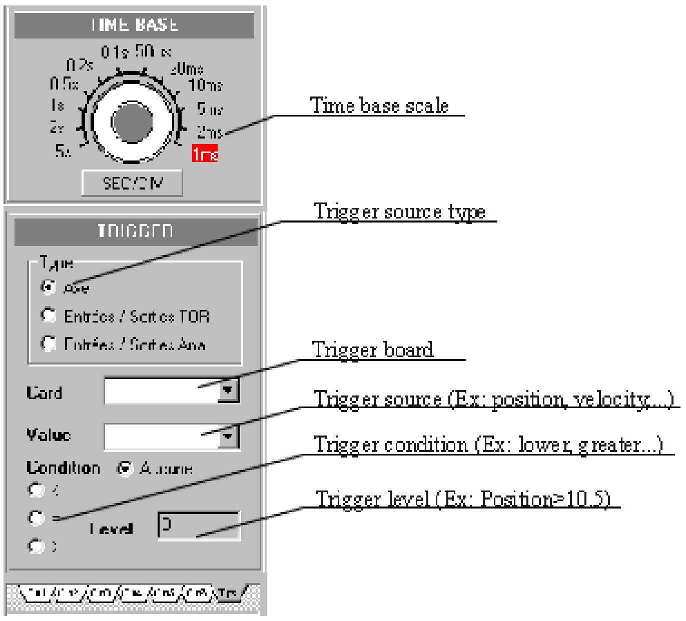


🗬 In the acquisition configuration zone, user can define the number of samples during an acquisition cycle. User can start and print an acquisition.

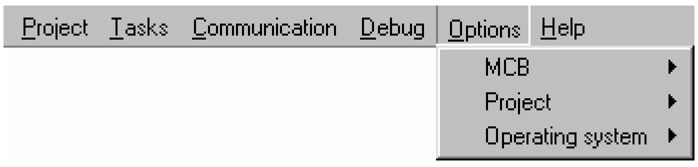
🗬 The channels configuration zone have 6 channels tabs and a time-base tab. For channels tabs, user can define the type of the board, the board, and the acquisition parameter. For example with an axis board, the following error can be chosen.


CH 1
SCALE
x1000
UNIT/DIV
INPUTS
Aucune
Axe
Entrées / Sorties TOR
Entrées / Sorties ana
Card
Value
Channel scale to view signals
Information source (axis, inputs/outputs...)
Board select
Data to observ (Ex: position, velocity...)

🗬 The visualization screen displays the six channels. A double-click on this zone and the window is in full screen. This window gives the position in X and Y of the cursor. We can also define reference position on X and Y. A click on dX or dY shows a moved vertical or horizontal line. The position of the new click defines the reference position. The value indicates in dX or dY is the difference between the cursor position and the reference position.

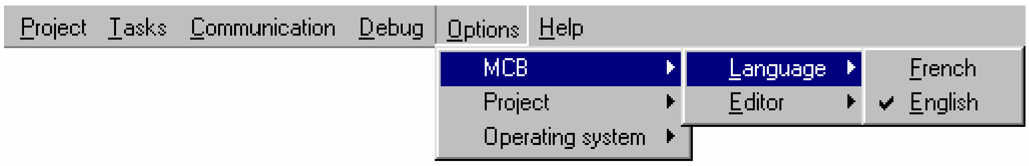

TIME BASE
0.1s 50ms
0.2s
20ms
0.5s
10ms
1s
5ms
2s
2ms
5s
1ms
SEC/DIV
TRIGGER
Type
Axe
Entrées / Sorties TOR
Entrées / Sorties Ana
Card
Value
Condition
Aucune
<
=
>
Level
0
Time base scale
Trigger source type
Trigger board
Trigger source (Ex: position, velocity...)
Trigger condition (Ex: lower, greater...)
Trigger level (Ex: Position>10.5)

3-4-5- Options menu



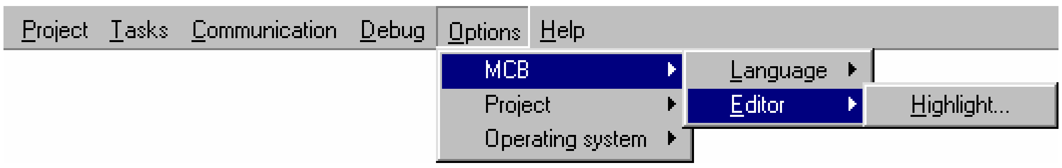
MCB

Language

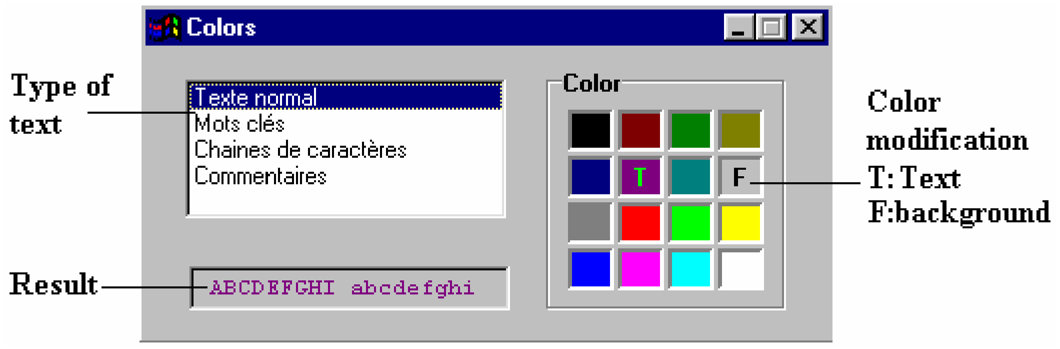


This sub-menu allows the selection of language.

Editor



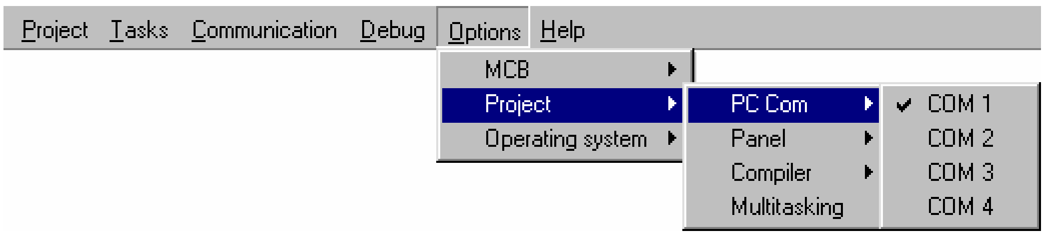
This sub-menu allows to customize the colour of the font and the background of text, key-words… in the editor of basic task.



To modify a colour, first you should select a type of text. Then, you choose one colour and you click on it with the left click to change the font colour or the right click to change the background colour. A screen shows the result of the modification.

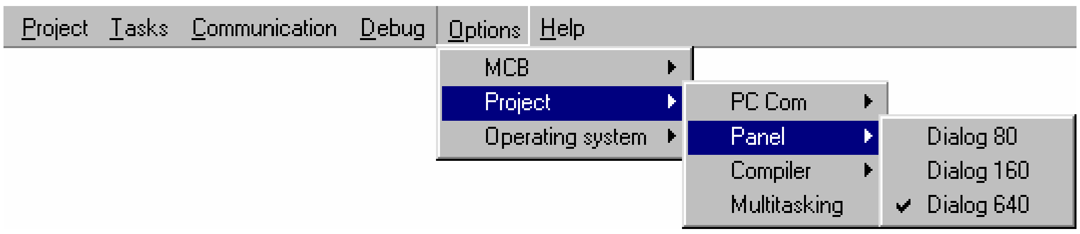
Project

PC Com



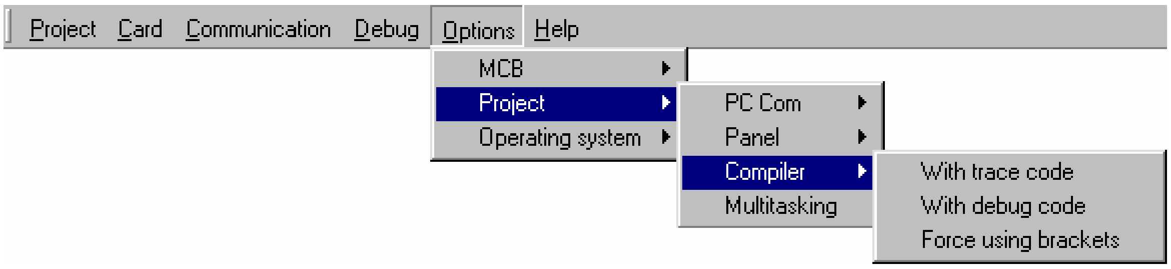
This sub-menu allows the selection of the communication port of the PC which is in link with MCS.

Panel



This sub-menu is only used when MCS manage a SERAD terminal panel. So, you can change the type of terminal panel. In the basic task editor, the panel dialog box will be update with this choice.

Compiler



With trace code

Action: This command adds some information to the compiler to obtain the trace code in task. This command is interesting to test systems but the compiled file are biggest and the execution of task is ran slowly. When it is activate or disable, you need to compile the tasks again.

With debug code

Action: This command adds some information to the compiler to use the step by step and breakpoint functions in the debug mode.

Force using brackets

Action : This command strengthens the test of brackets during the compilation.

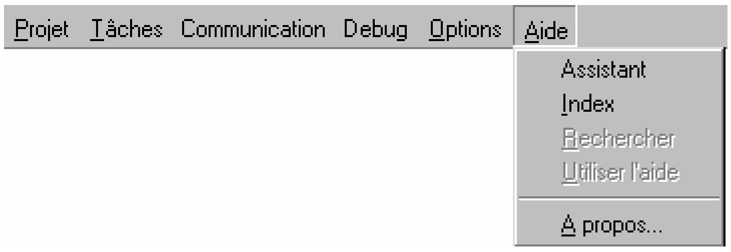
Multitasking

Action : This command allows the modifications of multitasking parameters. A dialog box appears and allows the modification of the ageing time task and the normal slice time.

Operating system

Action : This command update or clear the operating system. Attention, this procedure is reserved to experienced user.

3-4-6- Help menu



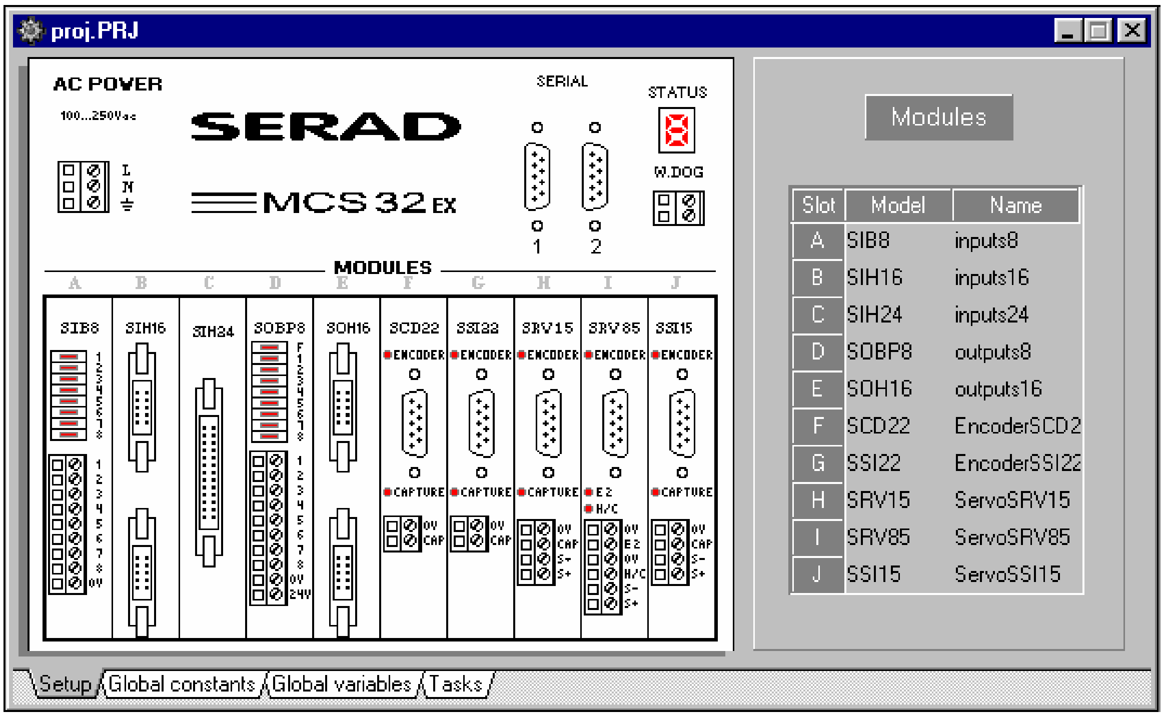
Wizard

Action:	This command allows the displaying of icon information.

About …

Action:	This command launches a dialog box which indicates the software version, etc…

3-4-7- Configuration tab



Slot	Model	Name
A	SIB8	inputs8
B	SIH16	inputs16
C	SIH24	inputs24
D	SOBP8	outputs8
E	SOH16	outputs16
F	SCD22	EncoderSCD2
G	SSI22	EncoderSSI22
H	SRV15	ServoSRV15
I	SRV85	ServoSRV85
J	SSI15	ServoSSI15

On the configuration window, there are two zones. The first zone is on the left with the front of MCS. This zone allows the MCS configuration. We can configure the display machine, the SERIAL1 and SERIAL2 and the A, B, C, D, E, F, G, H, I, J slots. The second zone is on the right with the name « Modules ». This zone sums up the configuration of the 10 slots (Name and type of each modules). On each slots of the MCS (A, B, C, D, E, F, G, H, I, J), we can add, delete, modify or show a board. The add, modify and delete command needs to compile the tasks again.

There are three different ways to obtain the add (a board) command

- In the Card menu and select before the insertion slot
- A double click on the free insertion slot
- A right click opens a menu with the « Add » command (select the slot to insert before).

The modify (a board) command allows the modification of boards features and can be obtained in two different ways :

- In the Card menu and select before the modifying slot
- A right click opens a menu with the « Modify » command (select the board to modify before).

The delete (a board) command allows the suppression of a board in a slot and can be obtained in two different ways :

- In the Card menu and select before the deleted board
- A right click opens a menu with the « Delete » command (select the board to delete before).

The show (a board) command allows the configuration of the board parameters. This command can be obtained in three different ways :

- In the Card menu and select before the board to configure

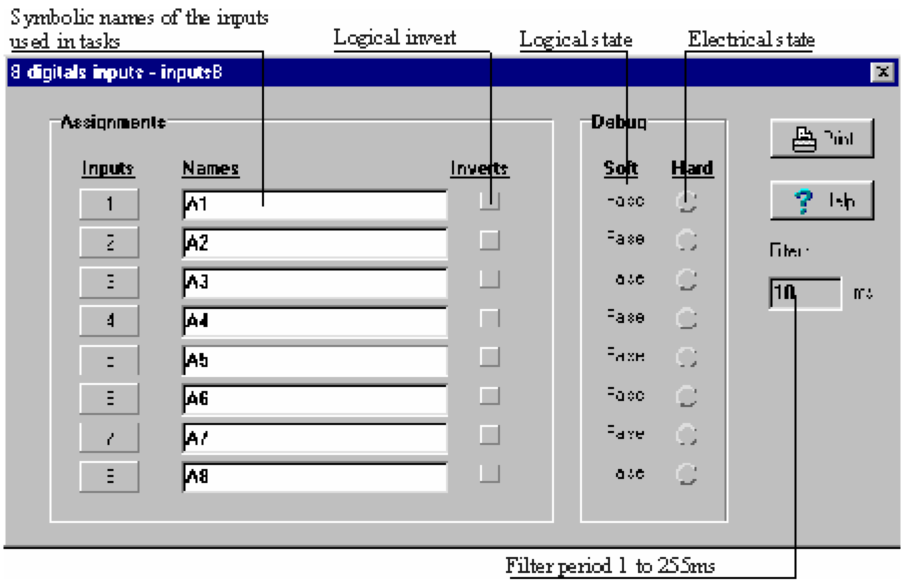
🖎 A double click on the board to configure

🖎 A right click opens a menu with the « Show » command (select the board to configure before).

Configuration of SERIAL1 and SERIAL2

To configure a serial communication port, you must make a double click on the serial to configure. A dialog box with the information of configuration appears. The serial communication port 1 is generally dedicated to a PC. To have a correct communication, these information must be the same between a terminal or a PC and MCS.

Configuration of 8 digital inputs board : SIB8

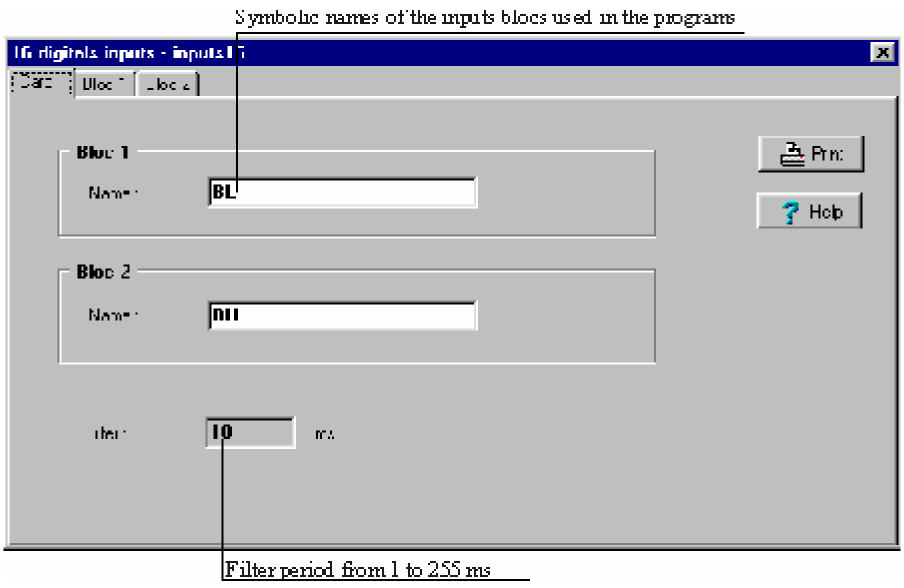


Inputs	Names	Inverts	Soft
1	A1	☐	False
2	A2	☐	False
3	A3	☐	False
4	A4	☐	False
5	A5	☐	False
6	A6	☐	False
7	A7	☐	False
8	A8	☐	False

Filter: 10 ms

Configuration of 16 digital inputs board : SIH16

To configure the bloc1 tab and bloc2 tab, you must refer to the configuration of SIB8 board.


Symbolic names of the inputs blocs used in the programs
16 digitals inputs - inputs16
Card
Bloc 1
Bloc 2
Print
Help
Filter period from 1 to 255 ms

Bloc 1 Name: BL

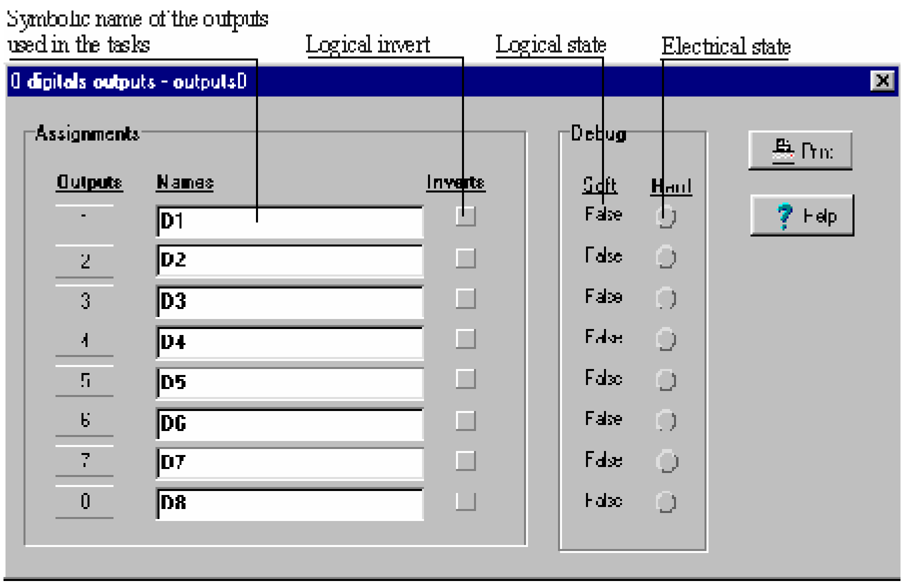
Bloc 2 Name: BII

Filter: 10 ms

Configuration of 24 digital inputs board : SIH24

To configure the bloc1 tab, bloc2 tab and bloc3 tab, you should refer to the configuration of SIB8 board. To configure the card tab, you should refer to the configuration of SIH16 board

Configuration of 8 digital outputs board : SOBP8



8 digitals outputs - outputs8

Assignments

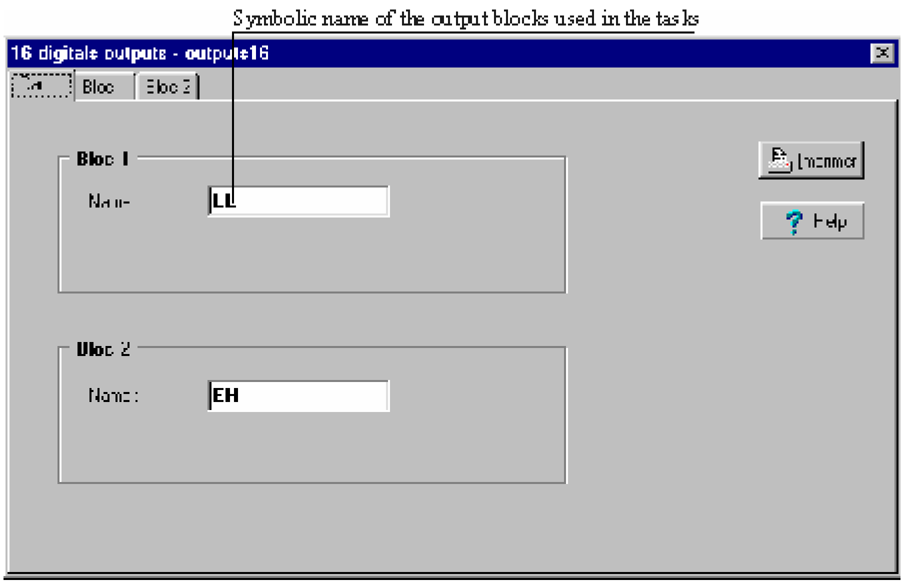
Outputs	Names	Inverts
1	D1	☐
2	D2	☐
3	D3	☐
4	D4	☐
5	D5	☐
6	D6	☐
7	D7	☐
8	D8	☐

Debug

Soft	Hard
False	○
False	○
False	○
False	○
False	○
False	○
False	○
False	○

Configuration of 16 digital outputs board : SOH16

To configure the bloc1 tab and bloc2 tab, you must refer to the configuration of SOBP8 board.

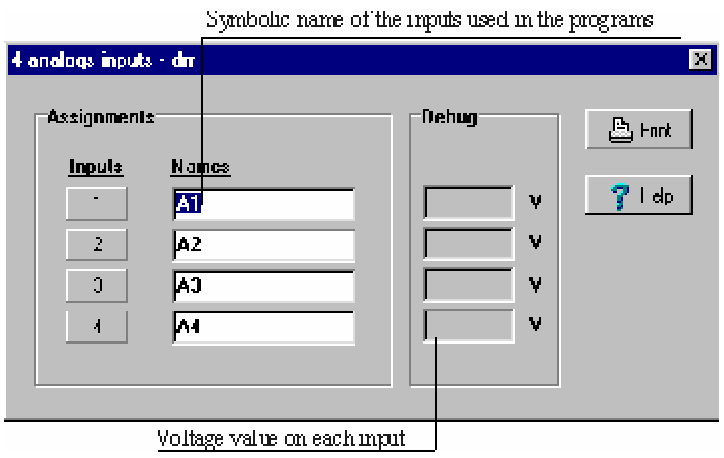


16 digitals outputs - outputs16

Bloc 1
Name: LE

Bloc 2
Name: EH

Configuration of 4 analogue inputs board: SIA14


Symbolic name of the inputs used in the programs
4 analogs inputs - din
Assignments
Inputs
Names
A1
A2
A3
A4
Debug
V
Print
Help
Voltage value on each input

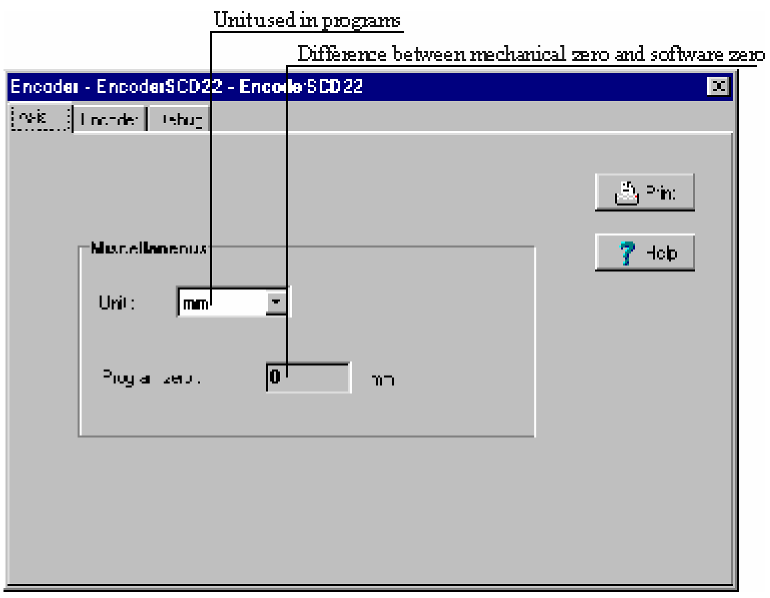
Configuration of 2 analogue outputs board : SOA12

The configuration of analogue outputs is the same as SIA14 board.

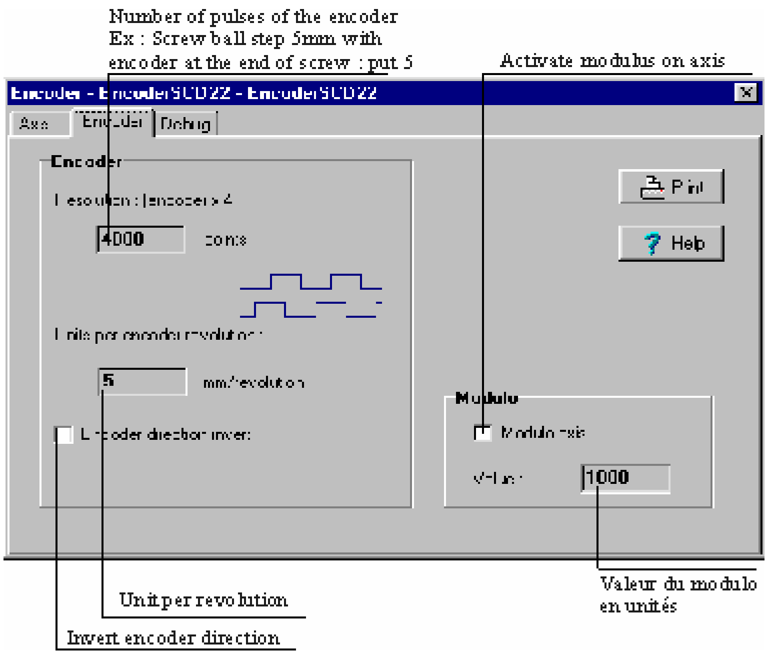
Configuration of encoder board : SCD22

Axis tab configuration :

The choice of units defines the expressed units of this encoder in all the programs. If you choose « mm », all the numeric value will be expressed in millimetres. The program zero parameter is the distance between the program zero found in home capture and the zero program used in programs.


Unit used in programs
Difference between mechanical zero and software zero
Encoder - EncoderSCD22 - EncoderSCD22
Axis
Encoder
Debug
Print
Help
Miscellaneous
Unit :
mm
Program zero :
0
mm

Encoder tab configuration :


Number of pulses of the encoder
Ex : Screw ball step 5mm with
encoder at the end of screw : put 5
Activate modulus on axis
Encoder - EncoderSCD22 - EncoderSCD22
Axis
Encoder
Debug
Print
Help
Unit per revolution
Invert encoder direction
Valeur du modulo
en unités

Encoder

Resolution : (encoder x 4)

4000 points

Units per encoder revolution :

5 mm/revolution

Encoder direction invert

Modulo

Modulo axis

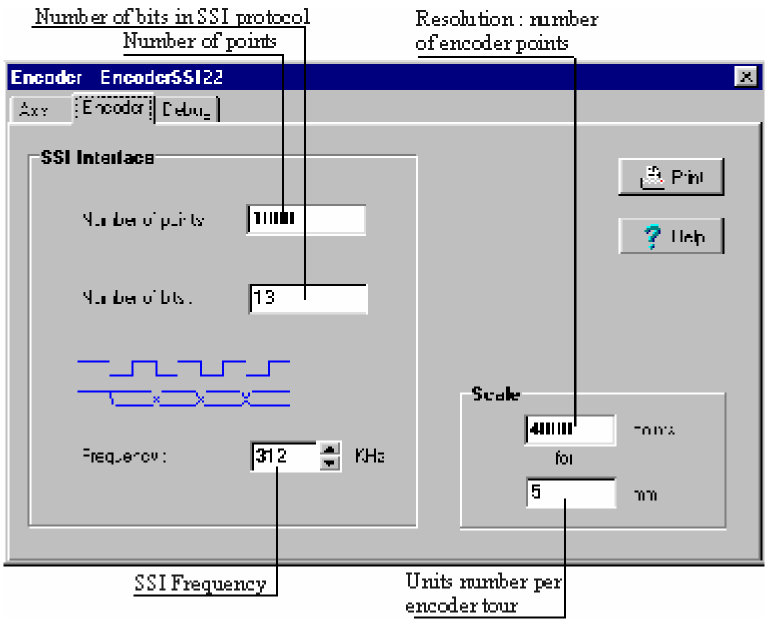
Value : 1000

The explanation of the debug tab is in the Debug menu.

Configuration of SSI encoder board : SSI22

For the Axis tab, you should refer to the encoder SCD22 board.

Encoder tab configuration :



SSI Interface

Number of points : [illegible]

Number of bits : 13

Frequency : 312 KHz

Scale

[illegible] points

for

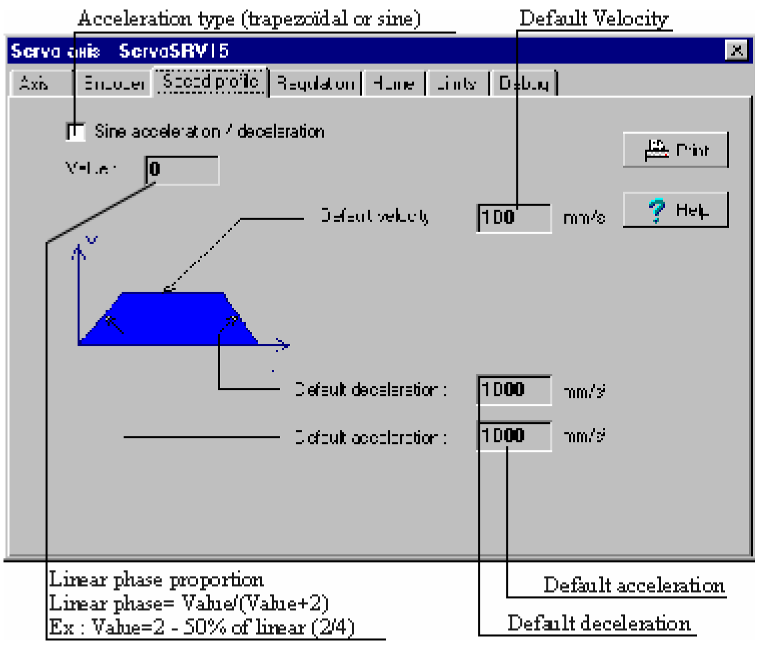
5 mm

The explanation of the debug tab is in the Debug menu.

Configuration of servo board : SRV15, SRV1524

The configuration dialog box of these cards have 7 tabs. To configure Axis and Encoder, you should refer to the SCD22 configuration tab.

Configuration of speed profile tab :



Sine acceleration / deceleration

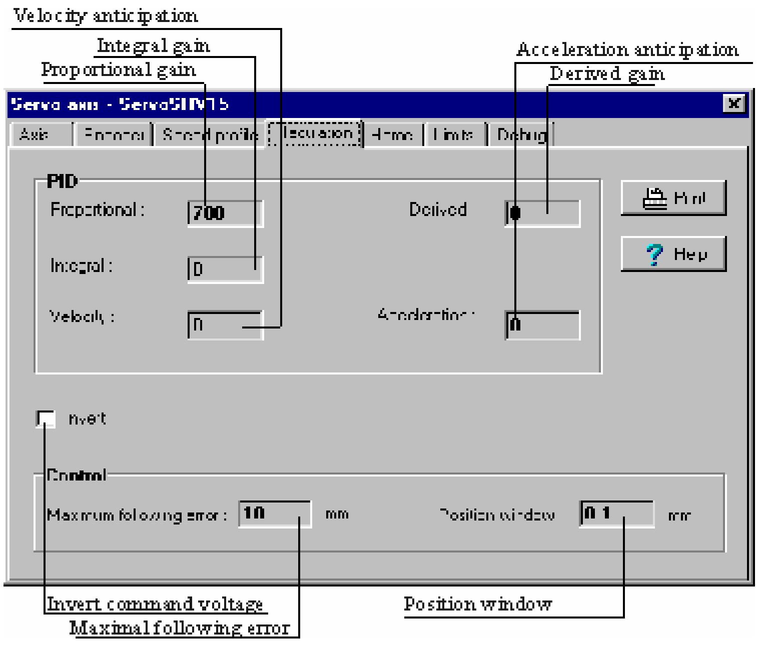
Value : 0

Default velocity 100 mm/s

Default deceleration : 1000 mm/s²

Default acceleration : 1000 mm/s²

Configuration of regulation tab :



PID

Proportional : 700

Integral : 0

Velocity : 0

Derived 0

Acceleration : 0

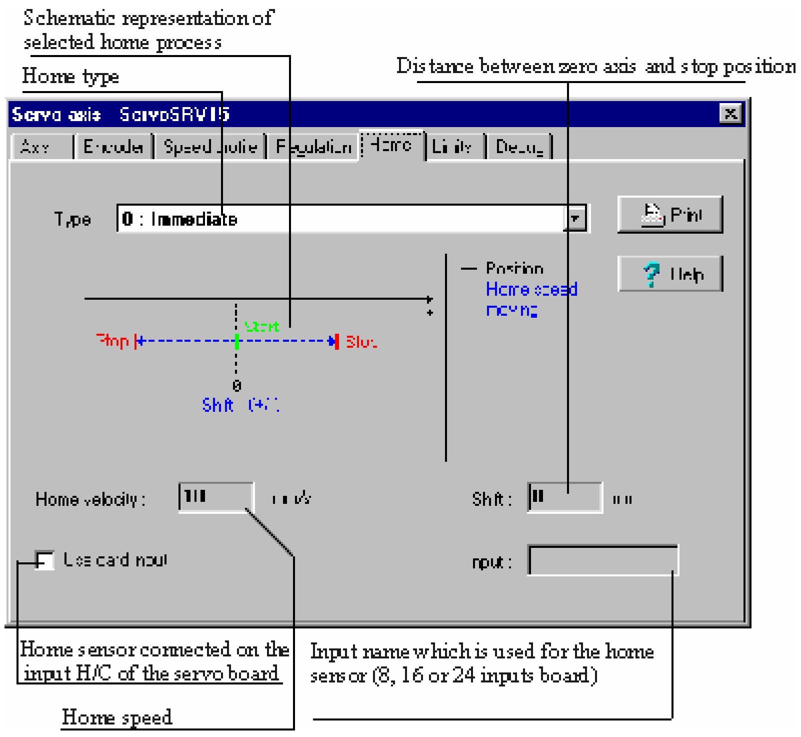
Invert

Control

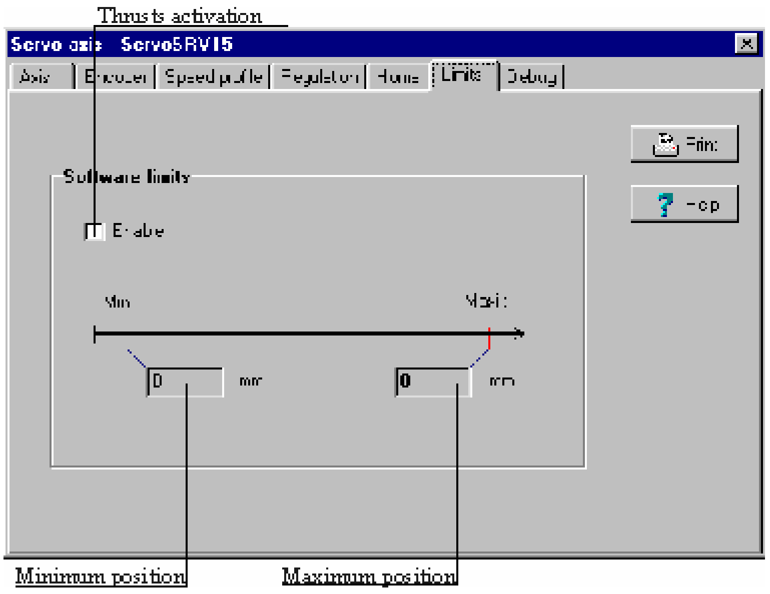
Maximum following error : 10 mm

Position window 0.1 mm

Configuration of home tab :


Schematic representation of selected home process
Home type
Distance between zero axis and stop position
Servo axis ServoSRV15
Axis
Encoder
Speed profile
Regulation
Home
Limits
Debug
Type
0 : Immediate
Print
Help
Position
Home speed
moving
Stop
Start
Stop
0
Shift (+/-)
Home velocity :
mm/s
Shift :
mm
Use card input
Input :
Home sensor connected on the input H/C of the servo board
Input name which is used for the home sensor (8, 16 or 24 inputs board)
Home speed

Configuration of limits tab :

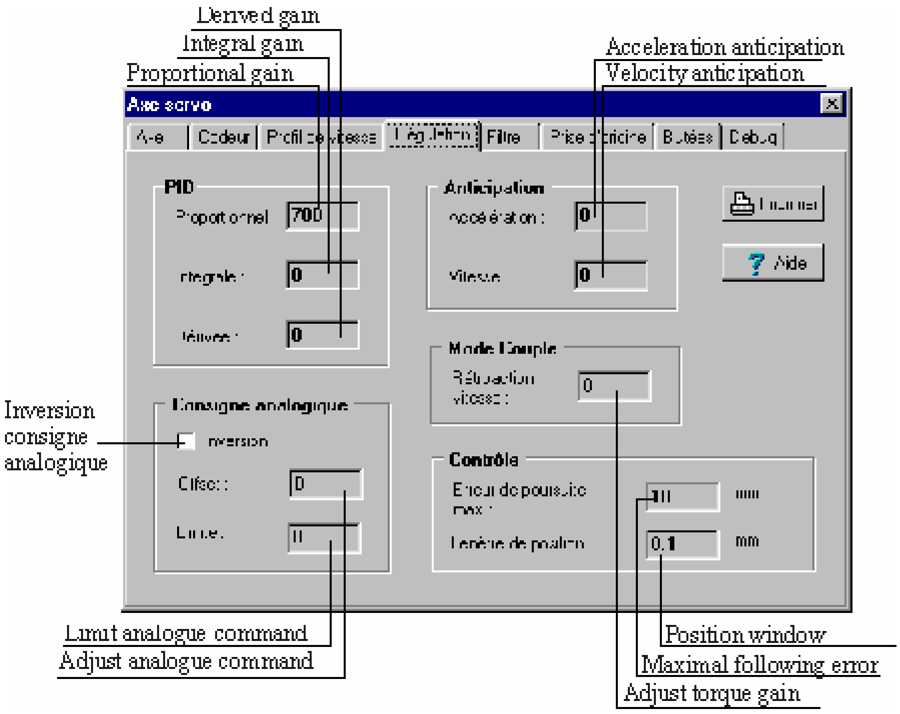


The explanation of the debug tab is in the Debug menu.

Configuration of servo board : SRV85

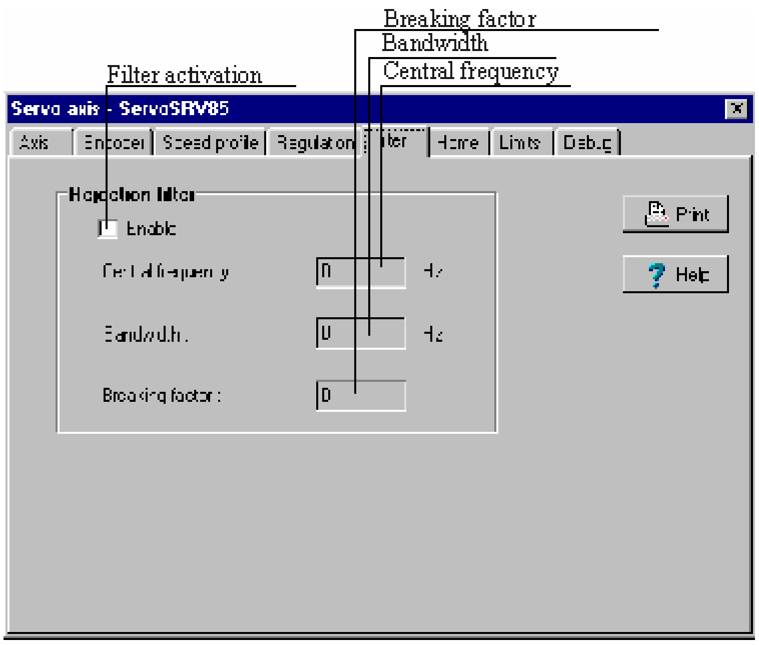
To configure Axis and Encoder, you should refer to the SCD22 configuration tab. To configure the home tab, speed profile tab and limits tab, you should refer to the SRV15 configuration tab.

Configuration of regulation tab :



Proportionnel : 700
Intégrale : 0
Dérivée : 0
Accélération : 0
Vitesse : 0
Réduction vitesse : 0
Inversion
Offset : 0
Limite : 0
Erreur de poursuite max : 10 mm
Fenêtre de position : 0.1 mm

Configuration of filter tab :



Enable
Central frequency : 0 Hz
Bandwidth : 0 Hz
Breaking factor : 0

The explanation of the debug tab is in the Debug menu.

Configuration of SSI servo board : SSI15

To configure the axis tab, regulation tab, speed profile tab and limits tab, you should refer to the SRV15 or SRV85 configuration tabs. To configure the Encoder tab, you should refer to the SCD22 encoder tab.

The explanation of the debug tab is in the Debug menu.

3-4-8- Configuration of STP stepper board : STP85

From the setup menu of the MCB EX software, a STP 85 stepper module can be added.

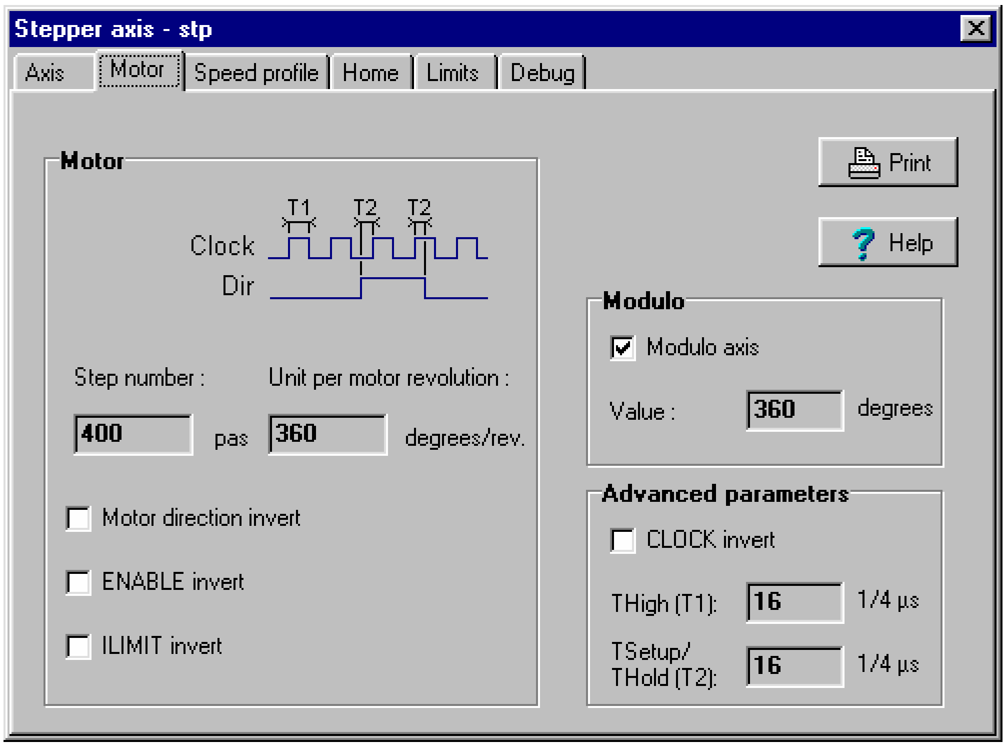
Then a double-click on it allow to view its parameters.

Different part appears :

- ***Axis :***

- Unit (mm, point, degrees, revolution...)

- Program zero



Stepper axis - stp

Axis | Motor | Speed profile | Home | Limits | Debug

Motor

Step number : 400 pas

Unit per motor revolution : 360 degrees/rev.

☐ Motor direction invert

☐ ENABLE invert

☐ ILIMIT invert

Print

Help

Modulo

☑ Modulo axis

Value : 360 degrees

Advanced parameters

☐ CLOCK invert

THigh (T1): 16 1/4 µs

TSetup/ THold (T2): 16 1/4 µs

•

Number of steps, half of steps, quarter of steps or micro steps of the motor

Examples :

For a 200 steps motor, driven by quarter of steps, enter 800

For a 1000 steps motor, driven by steps, enter 1000

- Units per revolution

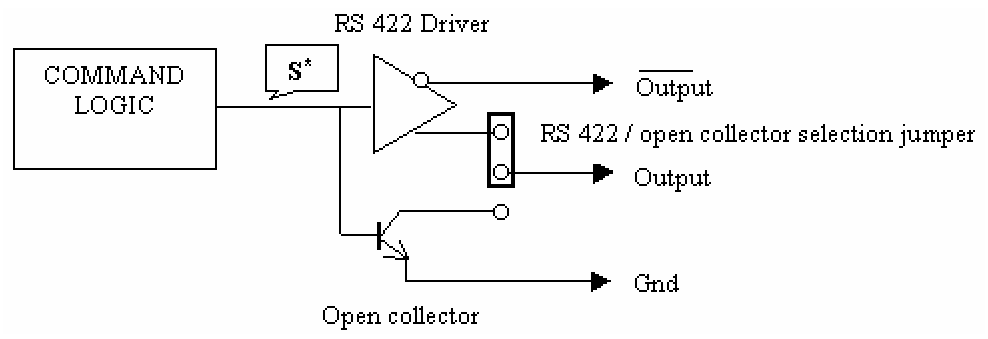
Example :

Motor directly connected on 5mm ballscrew, enter 5

- Signals invert :

- Motor direction invert (see STPINV_P parameter)
- ENABLE signal invert (see ENBINV_P parameter)
- ILIMIT signal invert (see ILMINV_P parameter)

Each signal can be use in positive or negative logic in order to be adapted to all kind of drives.


RS 422 Driver
COMMAND LOGIC
S*
$\overline{\text{Output}}$
RS 422 / open collector selection jumper
Output
Gnd
Open collector

Typical output diagram :

*

Signal invert not selected :	**S** point standby level = 0
Signal invert selected :	**S** point standby level = 1

Example :

ENABLE inverted.

ILIMIT non inverted.

```
.........
Ilimit(Stepper)=Off      ‘ S_Ilimit point  = 0
Enable(Stepper)=On       ‘ S_Enable point = 0
Delay 500
Enable(Stepper)=Off      ‘ S_Enable point = 1
Ilimit(Stepper)=On       ‘ S_Ilimit point = 1
.........
```

- Modulo axis :
- Modulo activation
- Modulo value

- Advanced parameters :

In most applications, let the default values.

The Thigh and Tsetup/hold times can only be modified after reading the drive characteristics.

- CLOCK signal invert (see CLKINV_P parameter)
- Thigh of CLOCK signal (see THIGH_P parameter)
- Tsetup/hold of DIRECTION signal (see THOLD_P parameter)

- ***Speed profile :***

- Default velocity
- Default acceleration
- Default deceleration
- Acceleration type : trapezoidal or sine

- ***Home :***

- Types : immediat, on sensor in positive direction / negative direction, quick on sensor in positive direction / negative direction
- Home velocity
- Shift
- Home input selection

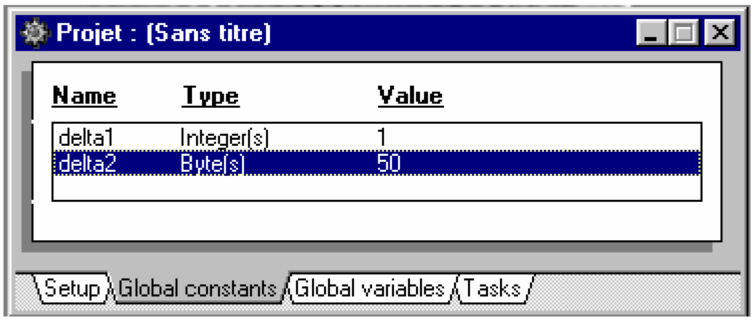
- ***Software limits :***

- Activation
- Minimal limit value
- Maximal limit value

- ***Debug :***

- Visualisation of the position, velocity, sensor1_s, sensor2_s, home_s, move_s
- Button for position clearing
- Infinite, absolute, relative, P1 / P2 position cycle, home mouvements
- Velocity, acceleration, deceleration modification
- Enable and Ilimit command: On / Off
- Enable / disable axis

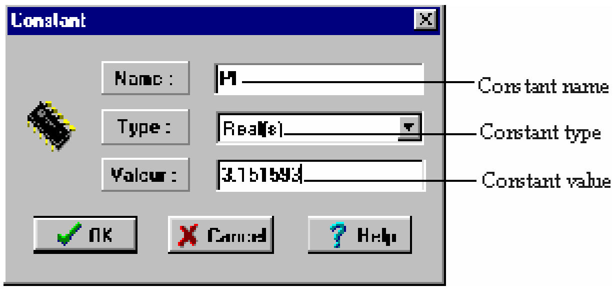
3-4-9- Global constants tab



Name	Type	Value
delta1	Integer(s)	1
delta2	Byte(s)	50

In the « Global Constants » tab, all the constants with their features (name, type and value) are summarized. In this tab, we can add, modify or delete a global constants. The add, delete and modify commands need the compilation of the tasks again and the sending tasks.

The « Add » command create a new global constant to the project. A dialog box appears to configure the parameters of this new constant.



Name : PI
Type : Real(s)
Valeur : 3.141593

The « Add » (a constant) command can be obtained in two different ways :

↳ In the Constants menu

↳ A right click open a menu with the « Add » command

The « Modify » or « Show » command allows the modification of the global constant parameters, except its type. This command can be obtained in three different ways :

↳ In the Constants menu (select the constant to modify before)

↳ A double click on the constant to modify

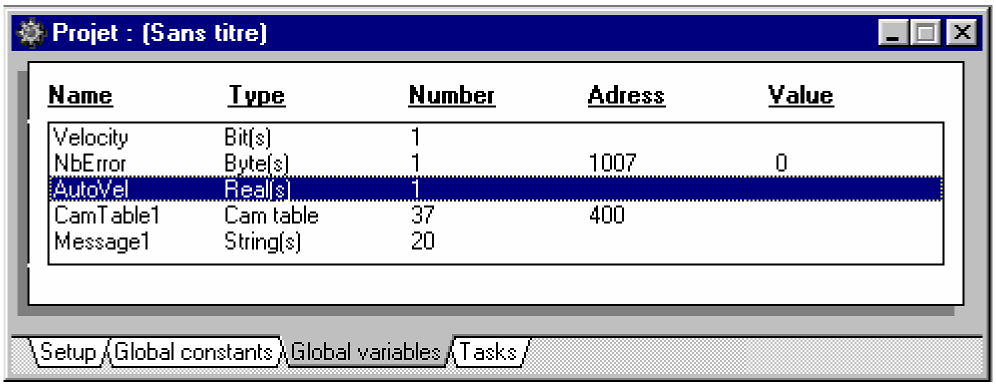
↳ A right click opens a menu with the « Modify » or « Show » command (select the constant to modify before).

The « Delete » command allows to suppress a global constant to the project. This command can be obtained in two different ways :

↳ In the Constants menu (select the constant to suppress before)

↳ A right click opens a menu with the « Modify » or « Show » command (select the constant to modify before)

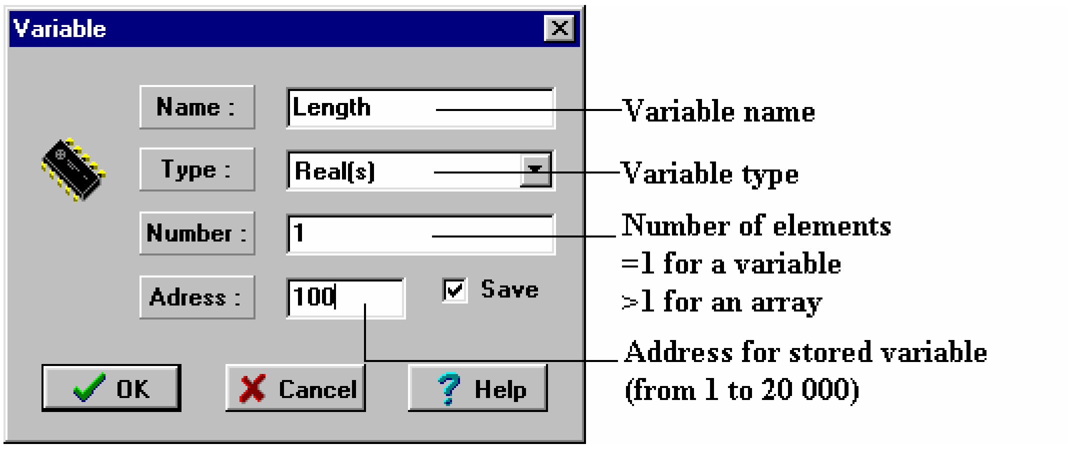
3-4-10- Global variables tab


Projet : (Sans titre)

Name	Type	Number	Adress	Value
Velocity	Bit(s)	1		
NbError	Byte(s)	1	1007	0
AutoVel	Real(s)	1		
CamTable1	Cam table	37	400	
Message1	String(s)	20		

In the « Global variables » tab, all the variables with their features (name, type, number, address and value) are summarized. The number parameter defines the number of elements in this array. The address parameter must be fixed if the variable is a stored variable (address 1 to 20000). In this tab, we can add, modify, delete or show a variable. The « Add », « Modify » or « Delete » command need the compilation of the tasks again and the sending tasks. In the case of a stored variable, you should send the variables too.

The « Add » command defines a new global variable in the project.


Variable

Name : Length
Type : Real(s)
Number : 1
Adress : 100
☑ Save

This command can be obtained in two different ways : :

🖎 In the Variables menu

🖎 A right click opens a menu with the « Add » command

The « modify » command allows the modification of the global variables parameters, except its type. This command can be obtained in two different ways :

🖎 In the Variables menu (select the variable to modify before)

🖎 A right click opens a menu with the « Modify » command (select the variable to modify before)

The « Delete » command suppresses a variable of the project. This command can be obtained in two different ways :

🖎 In the Variables menu (select the variable to delete before)

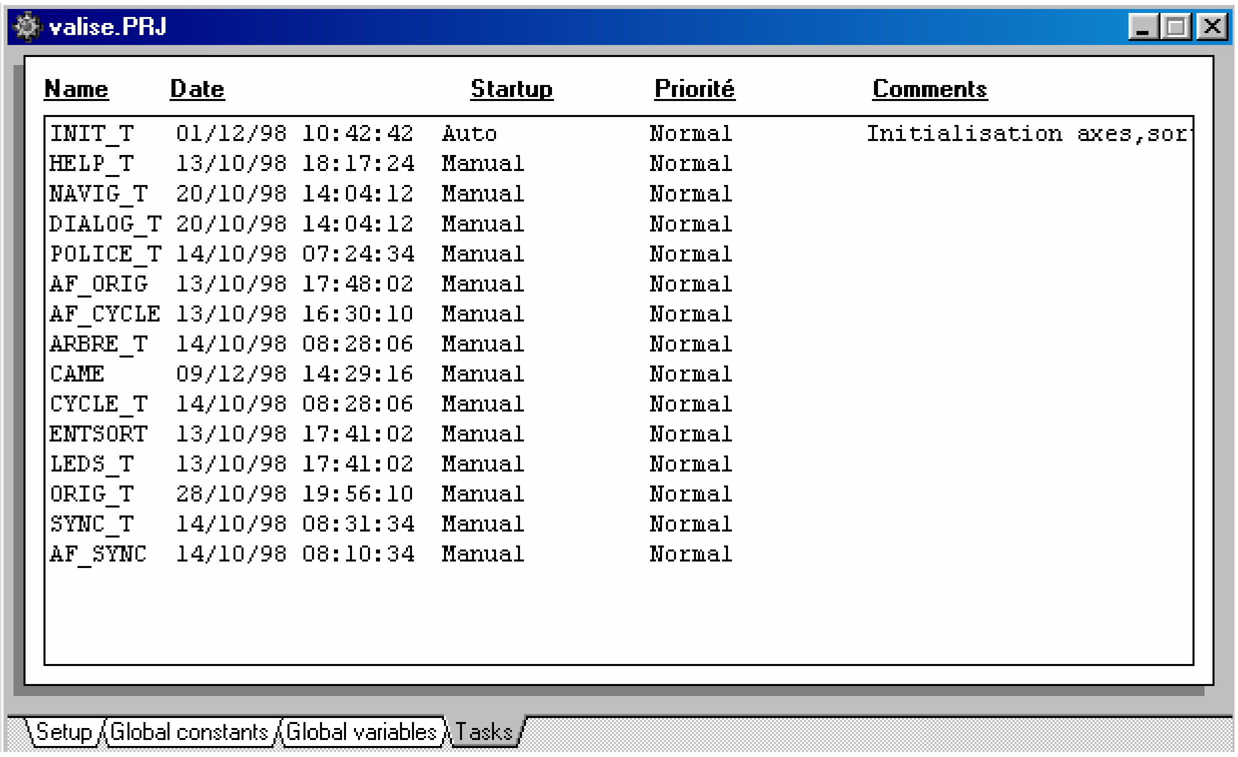
🖎 A right click opens a menu with the « Delete » command (select the variable to delete before)

The « Show » command allows the vizualisation of a variable state. This command shows all the value of an array. It can be obtained in three different ways :

- In the Variables menu (select the variable to show before)
- A double-click on the variable to show
- A right click opens a menu with the « Show » command (select the variable to show before)

When you want to show a camtable variable, the cam editor is launched. In this editor, we can define the profile of a cam.

3-4-11- Tasks tab

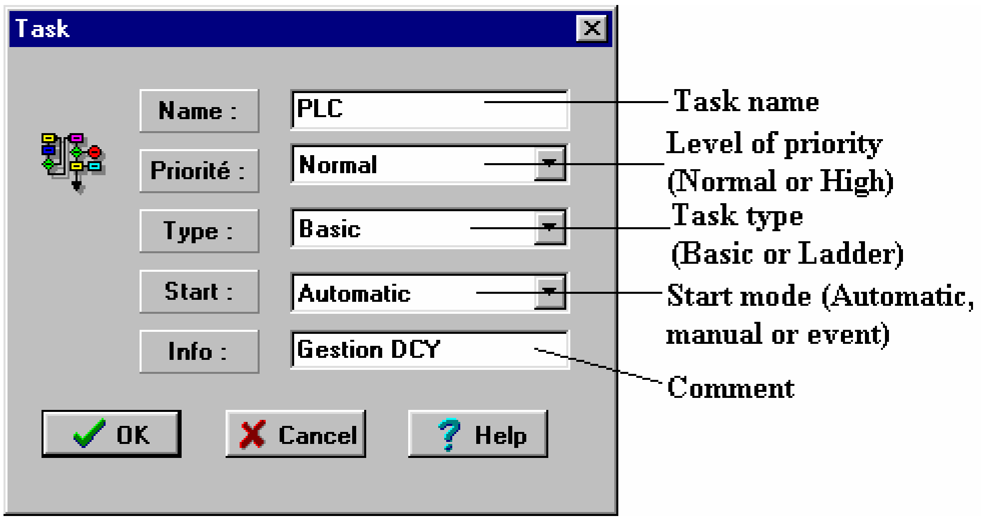


valise.PRJ

Name	Date	Startup	Priorité	Comments
INIT_T	01/12/98 10:42:42	Auto	Normal	Initialisation axes,sor
HELP_T	13/10/98 18:17:24	Manual	Normal	
NAVIG_T	20/10/98 14:04:12	Manual	Normal	
DIALOG_T	20/10/98 14:04:12	Manual	Normal	
POLICE_T	14/10/98 07:24:34	Manual	Normal	
AF_ORIG	13/10/98 17:48:02	Manual	Normal	
AF_CYCLE	13/10/98 16:30:10	Manual	Normal	
ARBRE_T	14/10/98 08:28:06	Manual	Normal	
CAME	09/12/98 14:29:16	Manual	Normal	
CYCLE_T	14/10/98 08:28:06	Manual	Normal	
ENTSORT	13/10/98 17:41:02	Manual	Normal	
LEDS_T	13/10/98 17:41:02	Manual	Normal	
ORIG_T	28/10/98 19:56:10	Manual	Normal	
SYNC_T	14/10/98 08:31:34	Manual	Normal	
AF_SYNC	14/10/98 08:10:34	Manual	Normal	

In the « Tasks » tab, all the tasks with their features (name, date of creation, type of startup and comments) are summarized. In this tab, we can add, modify, delete or show a task. The « Add », « Modify », « Delete » or « Show » command need the compilation of the tasks again and a sending tasks to be done.

The « Add »command defines a new task in the project. A task have different features : priority (normal or high), type (basic or ladder), startup type (manual, automatic, event) and an optional comments. The type of the task defines the editor type of the task.



Task

Name : PLC
Priorité : Normal
Type : Basic
Start : Automatic
Info : Gestion DCY

OK
Cancel
Help
Task name
Level of priority (Normal or High)
Task type (Basic or Ladder)
Start mode (Automatic, manual or event)
Comment

The « Add » (a task) command can be obtained in two different ways :

➾ In the Tasks menu

➾ A right click opens a menu with the « Add » command

The « modify » command allows the modification of the tasks parameters. The « Modify » (a task) command can be obtained in two different ways :

➾ In the Tasks menu (select the task to modify before)

➾ A right click opens a menu with the « Modify » command (select the task to modify before)

The « Show » command launches the editor of task according to the type choose (basic or ladder). This command can be obtained in three different ways :

➾ In the Tasks menu (select the task to show before)

➾ A double-click on the task to show

➾ A right click opens a menu with the « Show » command (select the task to show before)

The « Delete » command suppress a task in the project. This command can be obtained in two different ways :

➾ In the Tasks menu (select the task to delete before)

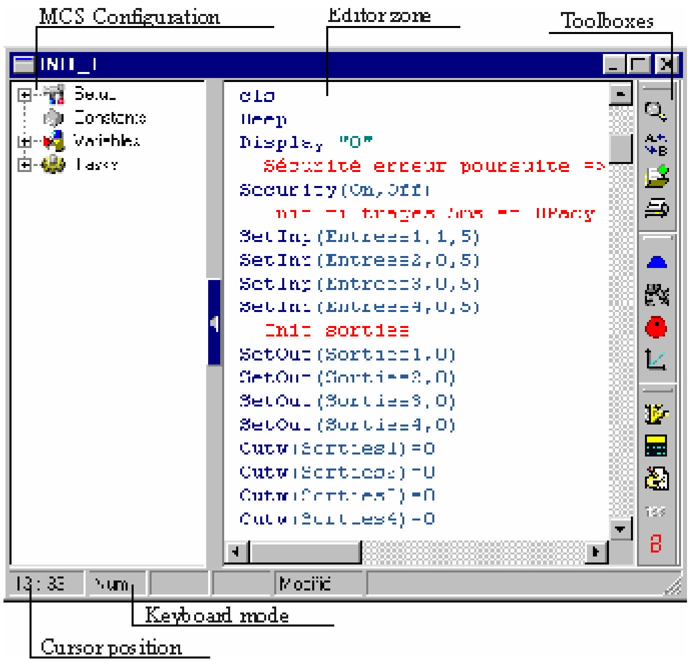
➾ A right click opens a menu with the « Delete » command (select the task to delete before)

Attention : A ladder task is automatically traduct in basic. It's advise to not write a long or complex ladder task, in order to avoid time cycle detoriorations and basic traduction limit.

3-5- Editors

3-5-1- Basic task editor

The basic editor has a zone to edit the program, a toolbox to help the user and another optional zone with all the parameters and variables of MCS. In this last zone, there are all the parameters of each card of the project, all the constants and global variables.



```
cls
Beep
Display "0"
  Sécurité erreur poursuite =>
Security(On,Off)
   [illegible]
SetInp(Entrees1,1,5)
SetInp(Entrees2,0,5)
SetInp(Entrees3,0,5)
SetInp(Entrees4,0,5)
  Init sorties
SetOut(Sorties1,0)
SetOut(Sorties2,0)
SetOut(Sorties3,0)
SetOut(Sorties4,0)
Outw(Sorties1)=0
Outw(Sorties2)=0
Outw(Sorties3)=0
Outw(Sorties4)=0
```

The toolbox helps the user to use movement and other instructions. Tools are in subroup :

⇨ **Editor tools**

↳ A tool to search a word or a group of word . This search is made in the task in confusing or dissociate the upper and lower case.

↳ The replace command search and replace an occurence in a task.

↳ The import command allows to add some file of a form (DXF format – autocad file).

↳ The print command

↳ The next icon display a quick syntax of the check instruction of task editor.

↳ Copy (CTRL+C), paste (CTRL+V) and cut (CTRL+X).

⇨ **Control of movement tools**

↳ The command to edit trajectory (TRAJ) instruction .

↳ The command to edit gearbox (GEARBOX) instruction .

↳ The command to edit cambox (CAM) instruction .

↳ The command to edit movements (MOVS) instruction .

⇨ **Communication tools**

↳ The open communication port command defines the parameter of the basic instruction OPEN.

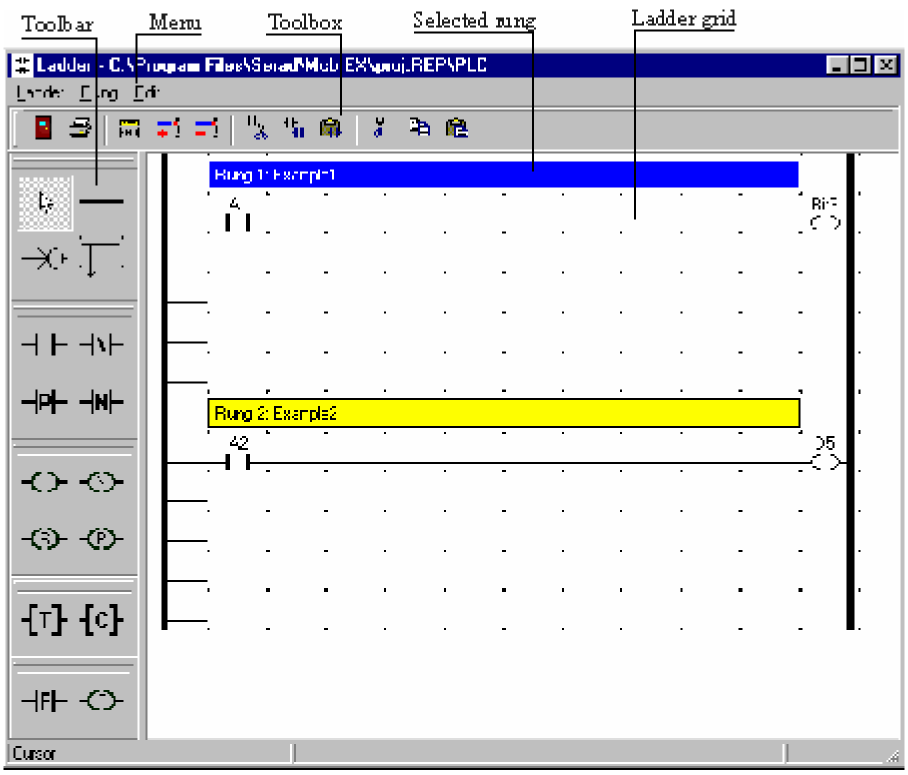
☞ the terminal panel command helps to defines the text on it. Th dialog box is defined with the front of the terminal panel selected in the Option menu. To generate the code corresponding with the text on the terminal panel display, you should choose the Write command. To show the result of a part of code, you should select some text and the Read command. The clear command clears the screen of the terminal panel.

☞ The edit command helps to define an numeric or alphanumeric edit on a terminal panel.

☞ The format command helps to define the code to format a variable.

☞ The display command helps to define the parameter for the DISPLAY instruction.

3-5-2- Ladder task editor



The ladder editor is composed with an editor zone of the ladder program (the grid), a toolbar of chart that can be inserted and a toolbox. A ladder program have lot of rungs limited to 50 in a task. Each rungs have an optional comment, an expression and 1 to 5 outputs.

☞ The tool bar can define the type of chart that can be put on the ladder grid. To select a chart, you just have to click on the button with the wanted chart.

☞ The ladder grid allows to put different chart and so to define the program.

A selected case of the ladder grid is indicated with a black background.

To put a chart on the ladder grid, you just have to click with the left button of the mouse on the ladder grid. The parallel link goes <u>on the top and left corner to the bottom and left corner of the case.</u>

A double-click on a case of a ladder grid where there is a chart allows user to configure it :

⇨ For coils and contacts, a MCS configuration screen appears on the toolbar. In this zone, there are all the bits variables like inputs, outputs and system bits. We can also find 64 bits. By default, there names are like :<bit>+ n)of bit. The name of this bits can be changed by a click or with the menu which appears on a right click. There are three system bits : an init bit which is equal to one during the first cycle of the program and two blink bits. One have a semi period equal to 500 ms and the second to 1s.

⇨ For blocs, there is a specific dialog box. For timer, the name and the delay is configured. For counter, the name, the type and the preset value is configured.

⇨ For free blocs, a dialog box allows to edit the basic code. For free contact, the code will be a condition and for free coil, it will be an action. An error in the code edited will be only detected by the basic compiler.

The selected rungs is the rungs with the background of the comments in blue. To edit a comments, you just have to make a double click on it.

↳ Commands

⇨ A command to quit the ladder editor .

⇨ A command to print .

⇨ Commands to add a rung , to insert a rung before the selected one and to delete the selected rung .

⇨ Commands to cut , copy and paste a chart . The cut or copied chart is the selected one. The paste chart is inserted on the free selected case.

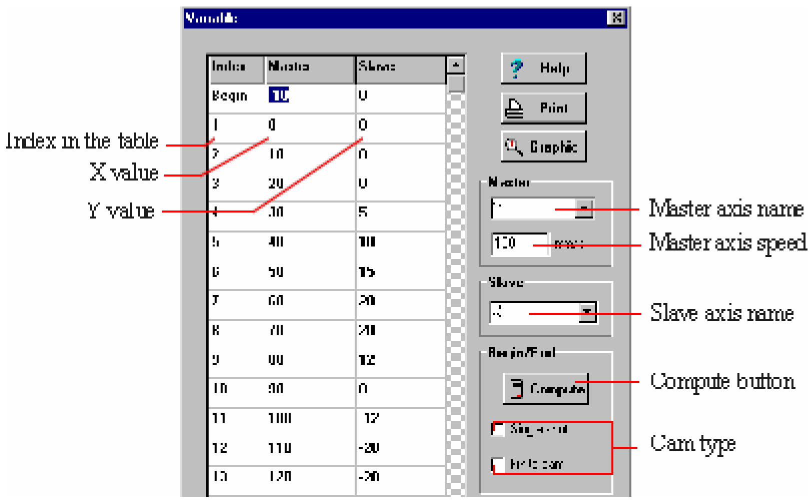
⇨ Commands to cut , copy and paste a rung . The cut or copied rung is the selected one. The paste rung is inserted before the selected one.

⇨ The SUPPR key clears the chart on the selected case. To delete a parallel link, you need to select this chart in the toolbar and to click in the case of the link.

⇨ The «Go at rung» function allows to go to a particular rung.

3-5-3- Cam editor

This editor appears when you want to show a camtable variable in the « global variables » tab. This editor allows to define the element number of a cam and their points. It is composed with the camtable and the help developing tools.

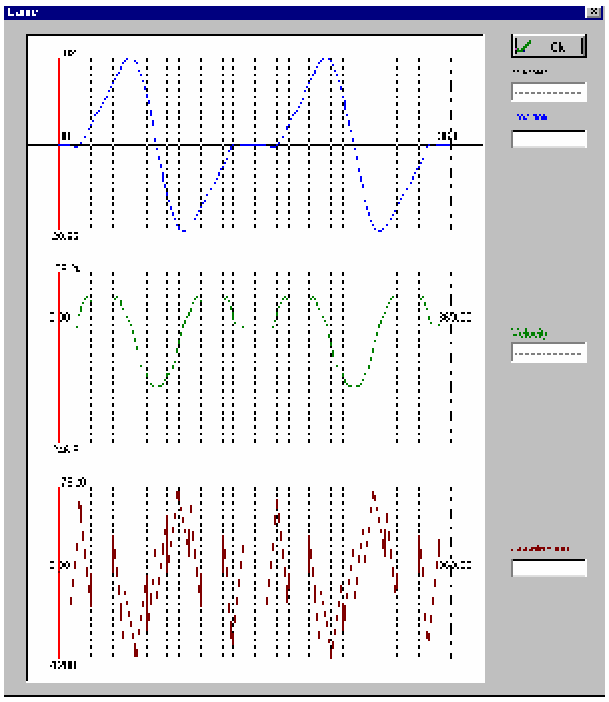


Index	Master	Slave
Begin	10	0
1	0	0
2	10	0
3	20	0
4	30	5
5	40	10
6	50	15
7	60	20
8	70	20
9	80	12
10	90	0
11	100	12
12	110	-20
13	120	-20

The cam table is composed with the different points, which have an index between 1 to N, and two extra points « begin » and « end ». Each point have a master distance and a slave distance. On a right click, a menu with the add or suppress points command appears. This menu only appears if the mouse cursor is on the « master » or « slave » column and if the box is not selected.

In the help developing tools, there are the « Master » and the « Slave » zone. In these zone, we can define the boards which realize the movement. For the « Master », we must define an extra parameter : velocity. This parameter is necessary to compute the coordinates of point in the grapical drawing.

The graphical drawing of the came profile is obtained on a click in the « graphic » button. In this window, there are three graphics with the master distance on Y axis. The three graphics are : the slave distance, velocity and acceleration on the X axis. This curves show the necessary features of motor if the master velocity specified in the previous window is a good choice. If the camtable is modified, you must validate the « graphic » button again to show the new graphics. When the graphic window is on the screen, you must click on the « OK » button before to quit the cam editor.



The « Input/Output » zone allows to define the cam type. With this choice and after a click on the « compute » button, the begin and end points are computed. Thess computation allow to obtain the begin and end features of a cam define in the cam chapter.

➷ Single shot validate : The begin and end slope will be equal to zero.

➷ Finite cam validate : The begin and end slope will be the same and the begin and end position of the cam will be the same too.

➷ Finite cam unvalidate (infinite cam) : The begin and end slope will be the same

The computation define the chaining slope which depends on the type of the came. When the camtable or the type of the came is modified, it is necessary to compute again.

Warning : If you want to realize a specific cam which is not in the cam type area, you don't have to compute.

The value in the master column must be coherent before to quit the editor.

4- PROGRAMMING LANGUAGE

4-1- Introduction

4-1-1- Description

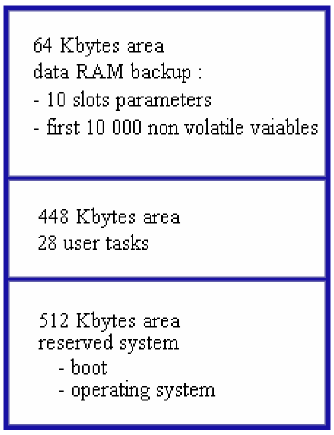
Motion Control Basic is a complete programming language and easy to use with its structured programming constructions as found in most other modern programming languages. This language is built around a multitask kernel and the basic language to ensure flexible and powerful programming. The basic language also contains all the control movement functions and PLC function.

The language manages the constants and variables like global, local or stored variables…

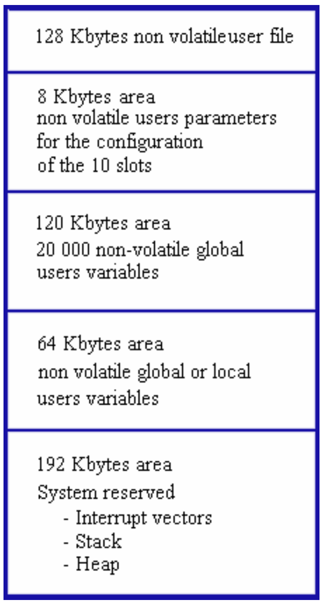
A project developed with MCB software can contain up to 28 tasks working in parallel. Each task have a priority level and can be describe by the basic or ladder editor. An extra task treats the fastest events.

4-1-2- Memory plan of MCS32 Ex

Flash memory (1 Mb)



non-volatile ram memory (512 Kb)



4-2- Data

4-2-1- Global constants

The global constants are defined with the global constant tab of the MCB software. The types accepted are : bit, byte, integer, long integer, real, string char.

Constants are read only data defined in a project. They are stored in the flash memory with the code of the task compiled. A global constant can be used by all the tasks.

4-2-2- Global variables

The global variables are defined with the global variable tab of the MCB software.

A global variable and a constant variable can't have the same name in a project because the compiler can't do the distinction. The global variable types and the global constant types are the same.

A global variable can be used by all the tasks and accessed at every time.

This variable is an array if at its creation, the field « number » is greater than one. The first index of an array is 1. The index can be an immediate value, a byte variable or integer variable.

Example :

```
Position = PositionArray[5]
```

' Warning : A writing at the 0 index in an array is forbidden : this error can make trouble in the operating cycle.

The variable can be a stored variables.

On a power cut, the variable value is preserved. There are 20 000 stored variables at address 1 to 20 000. Then, a stored variable must be assigned to an address.

' Warning : The user must beware of the crossing of variables when he assigns them. For example, an array with 50 elements is assigned at the address 100, the next variables must be assigned at an address greater than 150.

The crossing of variables can be used in one case : to allows the address access with multiple variables.

Example :

TableModbus : array of 50 integer assigned at address 100

DecompteurPiece : integer variable assigned at address 100

```
  If TableModbus[1]=0 Then Goto EndProduction
  If DecompteurPiece=0 Then Goto EndProduction
```

This two last program lines are the same but the last one is the most explicit.

Unlike the local variables, you need to define the global variable before you use it. An non-stored variable will be used as a link between tasks. Whereas the stored variable are used to preserve adjust parameters etc….

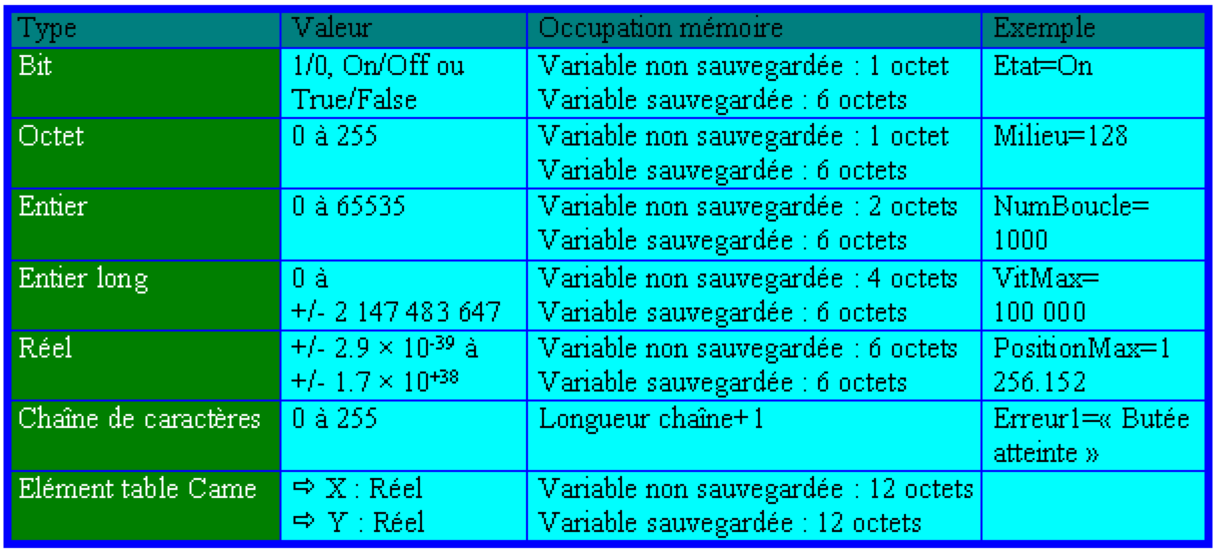
' Warning : The Camtable type have a particular stored mode. Each element is made up with two real variables. The array have also two extra internal elements : the start point and the end point. For example, if a camtable with 10 elements is defined at address 100, the camtable will have (10 + 2) * 2 = 24 real variables at address 100 to 123 include.

address 100 : Start point of the master	address 101 : Start point of the slave
address 102 : point 1 - master	address 103 : point 1- slave
.....	.....

address 121 : point 10 - master address 122 : point 10 - slave

address 122 : exit point - master address 123 : exit point - slave

Defined types are :



Type	Valeur	Occupation mémoire	Exemple
Bit	1/0, On/Off ou True/False	Variable non sauvegardée : 1 octet Variable sauvegardée : 6 octets	Etat=On
Octet	0 à 255	Variable non sauvegardée : 1 octet Variable sauvegardée : 6 octets	Milieu=128
Entier	0 à 65535	Variable non sauvegardée : 2 octets Variable sauvegardée : 6 octets	NumBoucle= 1000
Entier long	0 à +/- 2 147 483 647	Variable non sauvegardée : 4 octets Variable sauvegardée : 6 octets	VitMax= 100 000
Réel	+/- 2.9 × 10^{-39} à +/- 1.7 × 10^{+38}	Variable non sauvegardée : 6 octets Variable sauvegardée : 6 octets	PositionMax=1 256.152
Chaîne de caractères	0 à 255	Longueur chaîne+1	Erreur1=« Butée atteinte »
Elément table Came	⇨ X : Réel ⇨ Y : Réel	Variable non sauvegardée : 12 octets Variable sauvegardée : 12 octets	

4-2-3- Local variables

These variables are accessible only in the task where they are declared(main program and sub-programs). The accepted types are : bit, byte, integer, long integer, real, char string. Their values are not preserved between each power on and are cleared to zero.

You often need to store values temporarily when performing calculations with Basic. You need to preserve the values to compare them but without stocking them in a global variable.

The local variables don't need to be defined before being used. They have an identification character at the end of the name to indicate the data type. The local variables of a task can't be used by an other task. Two variables with the same name, used in two tasks, are two different variables. In a task, the variable can be used in the main program and the subprograms.

The treatment of a local variable is fastest than the global variable. A local array can’t be defined.

' Warning : Don’t use so much local string variables because each local string variable takes 256 bytes in memory !

Example :

```
a%=10                                  ` integer variable
If Position  !>1000 Then Position  !=0  ` real variable
Compteur&=Compteur&+1                  `     long      integer      variable
FormFeed$=Chr$(10)+Chr$(13)            ` string char variable
```

The local variables can have the following types :

Type	Valeur	Occupation mémoire	Déclaration	Exemple
Bit	1/0, On/Off ou True/False	1 octet	~	Capteur~=On
Octet	0 à 255	1 octet	#	Donnée#=128
Entier	0 à 65535	2 octets	%	Nb%=2000
Entier long	0 à +/- 2 147 483 647	4 octets	&	Vitesse&=120 000
Réel	+/- 2.9×10^{-39} à +/- $1.7 \times 10^{+38}$	6 octets	!	Pos!=122.245
Chaîne de caractères	0 à 255	256 octets	$	Mes$='Erreur'

4-2-4- Convert data types

To convert a data type to an other data type, use the functions below:

Destination Source	Bit	Byte	Integer	Long integer	Real	String
Bit	✕	Auto	Auto	Auto		Str$ or Format$
Byte	'.1' to '.8'		Auto	Auto	Auto	Str$ or Format$
Integer	'.1' to '.16'	'.L' or '.H'	✕	Auto	Auto	Str$, Mki$, Mkir$ or Format$
Long integer	'.1' to '.32'	✕	LongToInteger	✕	Auto	Str$, Mkl$, Mklr$ or Format$
Real	✕	RealToByte	RealToInteger	RealToLong	✕	Str$ or Format$
String	✕	✕	Cvi, Cvir	Cvl, Cvlr	VAL	✕

To extract a bit from a byte or integer, the function « .BitNum » can be used.

For a byte, BitNum is between 1 and 8, 1 is the least significant bit.

For an integer, BitNum is between 1 and 16, 1 is the least significant bit.

BitNum maybe a value or a byte variable.

To extract a byte from an integer, the function « .L » or « .H » can be used.

The « .L » function extract the least significant byte and « .H » the most significant byte.

Examples :

```
VarOctet=4
VarBit=VarOctet.3                    ' VarBit=1
VarOctet=16
Index=5
VarBit=VarOctet.Index                ' VarBit=1
VarBit=1
VarOctet=VarBit                      ' VarOctet=1
VarEntier=259
VarOctet=VarEntier.L                 ' VarOctet=3
VarOctet=VarEntier.H                 ' VarOctet=1
VarLong=261
VarReel=38.15
```

```
VarOctet=RealToByte(VarReel)       ' VarOctet=38
VarBit=1
VarEntier=VarBit                   ' VarEntier=1
VarOctet=128
VarEntier=VarOctet                 ' VarEntier=128
VarLong=45200
VarEntier=LongToInteger(VarLong)   ' VarEntier=45200
VarReel=54200.65
VarEntier=RealToInteger(VarReel)   ' VarEntier=54200
VarBit=1
VarLong=VarBit                     ' VarLong=1
VarOctet=128
VarLong=VarOctet                   ' VarLong=128
VarEntier=45200
VarLong=VarEntier                  ' VarLong=45200
VarReel=154200.65
VarLong=RealToLong(VarReel)        ' VarLong=154200
VarOctet=128
VarReel=VarOctet                   ' VarReel=128
VarEntier=45200
VarReel=VarEntier                  ' VarReel=45200
VarEntier=154200
VarReel=VarEntier                  ' VarReel=154200
VarChaîne= « -125.45 »
VarReel=Val(VarChaîne)             ' VarReel=-125 .45
VarReel=1510.55
VarChaîne=Str$(VarReel)            ' VarChaîne= «  1510.55 »
```

4-2-5- Numeric notations

Numeric values can be expressed in decimal, hexadecimal or binary.

Example :

```
VarOctet=254                       ' decimal notation
VarOctet=0FEh                      ' hexadecimal notation
VarOctet=11111110b                 ' binary notation
```

4-3- Tasks

4-3-1- Multitask principles

The real time and multitask kernel can manage 32 tasks in parallel :

- 4 internal tasks reserved to the system
- 27 users tasks defines in pseudo-basic or ladder
- 1 extra task for the management of events

The multitask launches the next task if :

- the executed time of the task is longer than the task ageing time. This time is defined in the Options menu. All the task must be compiled after a modification.
- execute a lock instruction :
  - Wait, Delay
  - Mova, Movap, Movac, Movr, Stop, Traj, Home
  - Beep, Edit
  - ClearFlash, FlashToRam, RamToFlash
- execute a loop or jump instruction :
  - Call

⇨ Goto, Case

⇨ For...Next

⇨ Repeat...Until

⇨ While...End While

⇨ End Prog

The Jump instruction make a jump without launching the next task.

In general, a short task will treat events faster than a big task.

4-3-2- Task priority

Each users task have a priority level : high priority, normal priority.

The multi-task kernel allocates two slices of execution : the high priority slice for the tasks with high priority, a normal priority slice for the tasks with normal priority.

The slice chain during execution is :

| high priority slice | normal priority slice | high priority slice | normal priority slice | ...

➭ High priority slice :

All the tasks with a high priority are executed one after one in this slice. Each task executes its instructions up to the ending condition (executes a locked task, ageing time reached ...).

Maximal execution time of a high priority slice = number of high priority task * ageing time

The ageing time is defined in the Options menus and is the same for the high and normal priority task. All the task must be compiled after a modification.

➭ Normal priority slice:

All the tasks with a normal priority are executed one after one in this slice. Each task executes its instructions up to the ending condition (executes a locked task, ageing time reached ...).

Normal slice execution time = Normal slice time

Maximal execution time of a normal priority slice = normal slice time + ageing time

The normal slice time is defined in the Options menus. All the task must be compiled after a modification.

If the execution time of all the normal priority tasks are lower than the normal slice time, all the tasks are executed one times and the high priority slice is executed.

In the opposite case, the system gives the hand at the high priority slice even if all the normal priority tasks aren't executed. These tasks will be executed in the next normal priority task.

Example :

```
T1, T2  : high priority tasks
T3, T4, T5, T6  : normal priority tasks
Ageing time = 2 ms
Normal slice time = 6 ms
The execution cycle will be | T1,T2 | T3,T4,T5 | T1,T2 | T6,T3,T4 | T1,T2 |
T5,T6,T3 | ...
```

4-3-3- Management of task

Each task can have a starting mode defined at its creation :

➭ Automatic start : At each power on of MCS, the task is launched automatically.

➭ Manual start : The task is not launched automatically.

A project must contain at least a task with automatic starting mode. You should have a task which have the initialization part and the launching task part.

There are 5 types of instructions to manage the tasks :

✎ Run :	launch a task which is stopped.
✎ Suspend :	suspend (pause) a task in execution
✎ Continue :	continue the execution of a suspended task
✎ Halt :	Stop an executed task
✎ Status :	indicates the state of the task

```
Example  :
Menus1 task                  Menus2 task
Prog                         Prog
.....                        .....
Run Menus2                   If Key = @ESC Then Halt Menus2
Wait Status(Menus2)=0        .....
....                         End Prog
End Prog
```

To synchronize the tasks each other, the Signal and Wait Event instructions or global variables can be used.

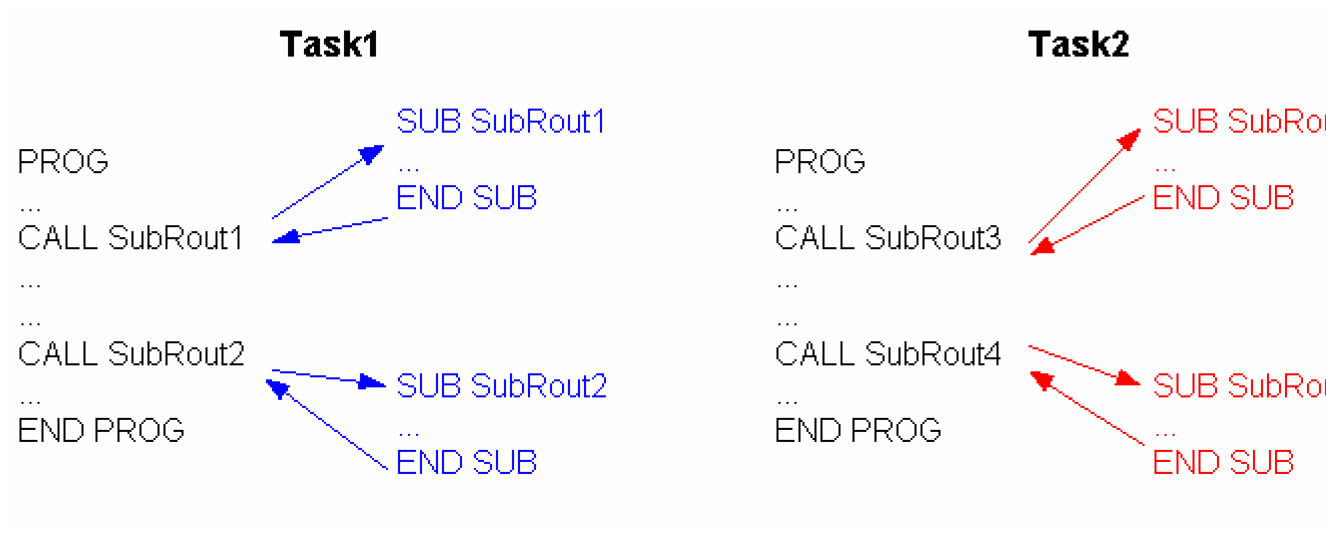
```
Example :
ProcessEnable  : global bit variable
Process1 task                Process2 task
Prog                         Prog
.....                        .....
ProcessEnable=1              Wait ProcessEnable=1
Wait ProcessEnable=0         .....
....                         ProcessEnable=0
End Prog                     .....
                             End Prog
```

' Warning : When you suspend or stop a task, these instructions don't stop the movement lauched by it.

```
Example :
Control task                       Process task
Prog                               Prog
.....                              .....
If ProcessError=1 Then             Mova(X=1000)
   Halt Process                    Out(S1)=1
   Stop(X)                         Mova(X=2000)
End If                             .....
....                               End Prog
End Prog
```

4-3-4- Basic task structure

Each task is constituted with a main program defined with the key-word PROG and END PROG and with subroutine defined with the key-word SUB .. END SUB. For example :



A) Main program

The main program of a task can call all its subroutine but it can't call the subroutine of others tasks. A task is a file. In the last example, the task 1 can call the subroutine 1 and 2 but it can't call the subroutine 3 and 4. A subroutine of a task can call a subroutine of the same task.

One and only one PROG ... END PROG structure must be used by a program and may appear at any place in the program.

During the execution of the task, the execution of the key-word END PROG makes a branch on the key-word PROG.

B) Subroutine

A subprogram must be declared by a procedure SUB...END SUB. This procedure may be before or after the main program.

To call a subroutine, you should use the CALL function. The subroutine called must be in the same task.

After a subprogram call, the execution continues automatically with the instruction that follows the subprogram call. You can stop subprogram executions by using the EXIT SUB instruction. For example :

```
SUB Calcul
Result%=0
IF b%=0 THEN EXIT SUB ' If b% is equal to zero , the division is impossible
Result%=a% DIV b%     ' Division
END SUB
```

A subroutine can be called anywhere in the program but it can't call itself. If datas are used in program and subroutine, you should use some specific variables. In fact, all the variable can be modified by a subroutine, you can assign the specific variables of a subroutine before it was called. For example :

```
...
Diviseur%=a%
Dividende%=b%
CALL Divise
IF Result!>10 THEN ...
...

SUB Divise
Resultat!=0
IF Diviseur%= 0 THEN EXIT SUB
   Resultat!= Dividende% / Diviseur%
END SUB
```

The branch to a subroutine launches the next task.

The instruction ICALL allow also to branch to a subroutine but whithout automatic tipping to next task.

C) Branch to a label

The GOTO instruction makes a branch to a label. A label is a name with at the end « : ». If the GOTO instruction is in a subroutine SUB…END SUB, the label must be in this subroutine.

A branch to a label with the GOTO instruction can be realized before or after the program. For example :

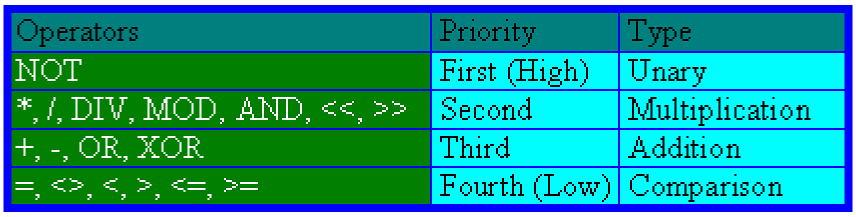
```
GOTO Label1
...
Label1:
...
```

With the GOTO instruction, the multitask kernel launches the next task.

The JUMP instruction have the same features as GOTO but the multitask kernel stays in this task.

D) Operators

The expressions are composed of operators and operands. In Basic, almost operators are binary, this means that they use two operands. Operators that use only one operand are called unary operands. Binary operators use common algebraic form, for example A + B. Unary operators come always before their operand, for example NOT A. In complex expressions precedence rules can suppress all ambiguity in operator order.



Operators	Priority	Type
NOT	First (High)	Unary
*, /, DIV, MOD, AND, <<, >>	Second	Multiplication
+, -, OR, XOR	Third	Addition
=, <>, <, >, <=, >=	Fourth (Low)	Comparison

The three fundamental rules concerning operators priority are :

➪An operand placed between two operators whose one has priority will be linked to the higher priority operator.

➪An operand placed between two operators whose priority are equal will be linked to the left operator.

➪Expressions between brackets are evaluated separately, so results are used as operand.

Operators with same priority are usually used from left to right.

You should used brackets to separate each expression in order to highlight the priority.

```
IF ((INP(E1)=1) AND (FlagRun=1)) OR (InitOk=0) Then ...
```

a) Arithmetical operators

'NOT' operator is an unary operator. + and - operators are used as unary and binary operators. Other operators are only binary operators.

An unary operator has only one parameter. For example :

NOT <Expression>

A binary operator has two parameters. For example :

<Expression1> * <Expression2>

b) Binary operators

Operator	Operation	Operand type	Type
+	Addition	Byte, Integer, Long integer or real	Operand type
-	Substraction	Byte, Integer, Long integer or real	Operand type
*	Multiplication	Byte, Integer, Long integer or real	Operand type
/	Division	Byte, Integer, Long integer or real	Operand type
DIV	Integer division	Byte, Integer, Long integer or real	Operand type
MOD	Modulus	Byte, Integer, Long integer or real	Operand type

c) Unary operators

Operator	Operation	Operand type	Type
+	Same sign	Byte, Integer, Long integer or real	Operand type
-	Invert sign	Byte, Integer, Long integer or real	Operand type

d) Logical operators

Operator	Operation	Operand type	Type
NOT	Binary negation	Byte, Integer	Operand type
AND	Binary AND	Byte, Integer	Operand type
OR	Binary OR	Byte, Integer	Operand type
XOR	Exclusive OR	Byte, Integer	Operand type
>>	Right shift	Byte, Integer	Operand type
<<	Left shift	Byte, Integer	Operand type

e) Bits operators

Operator	Operation	Operand type	Result type
NOT	Binary negation	Bit	Bit
AND	Logical AND	Bit	Bit
OR	Logical OR	Bit	Bit
XOR	Exclusive OR	Bit	Bit

f) String operators

Operator	Operation	Operand type	Result type
+	Concatenation	Char string	Char string

g) Relationship operators

Operator	Operation	Operand type	Result type
=	Equal	Byte, Integer, Long integer, real, string	Bit
<>	Different	Byte, Integer, Long integer, real, string	Bit
<	Lower	Byte, Integer, Long integer, real, string	Bit
>	Greater	Byte, Integer, Long integer, real, string	Bit
<=	Lower or equal	Byte, Integer, Long integer, real, string	Bit
>=	Greater or equal	Byte, Integer, Long integer, real, string	Bit

E) Tests

a) Simple tests

Conditional instructions provide a simple way to choose which part of code will be executed in accordance to a condition. There are two syntax. IF instruction syntax are :

```
IF <Expression> THEN
    <Instruction1>
    ...
[ELSE
   <Instruction2>
    ...]
END IF
```

or

```
IF <Expression> THEN <Instruction1> [ELSE <Instruction2>]
```

<Expression> must be a bit type value. If <Expression> is true then <Instruction1> and following instructions are executed. If <Expression> is false then <Instruction2> and following instructions are executed. In the second syntax form, only one instruction is executed for each condition, all instructions are in the same line and END IF statement is omitted.Nesting if instructions are possible but an ELSE always refers to the nearest IF instruction. For example :

```
VEL%(X)=100                        ' Set normal velocity
STTA(X=2000)                       ' Start absolute move to 2000 position
MOVE_ON:
IF POS_S(X) >1000 THEN VEL%(X)=50  ' Change velocity at middle distance
IF POS_S(X)>1500 THEN OUT(S1)=1 ELSE OUT(S2)=1
IF POS_S(X)>1700 THEN
    FlagInfo1=1
    IF OUT(S2)=1 THEN OUT(S2)=NOT OUT(S2) ` Blink output
 ELSE
    FlagInfo2=1
    OUT(S2)=0                             ` Reset output
END IF
IF MOVE_S(X)=On THEN GOTO MOVE_ON
```

b) Multiple tests

Multiple tests are performed with CASE instruction.

CASE instruction syntax is described below :

```
CASE <Expression> [ GOTO | CALL ] <Subrout1. Identif. > [ { , <Subrout2.
Identif.> } ]
```

<Expression> type must be byte, integer or long integer. With this instruction, subroutines will be called in accordance to <Expression> value. For <Expression>=1 the first subroutine is called, for <Expression>=2 the second subroutine is called ... For example :

```
REPEAT
INPUT #1,Choice%      'Read choice from serial peripheral device
ON Choice% CALL FirstChoice, SecondChoice, ThirdChoice
UNTIL  Choice%=0
GOTO FIN
SUB First Choice      ' Called if the first choice is selected
END SUB
```

```
SUB SecondChoice      ' Called if the second choice is selected
END SUB

SUB ThirdChoice       ' Called if the third choice is selected
END SUB
FIN  :
```

c) Loops

If the loop number is already known when writing your program, it is recommended to use the FOR loop structure, in other case WHILE or REPEAT structures can be used.

FOR instruction

FOR instruction allows the repeated execution of one or more instructions in accordance to a control variable increment or decrement .

FOR instruction syntax is described below :

```
FOR <Counter>=<Start> TO <End> [STEP <Increment>]
          <Instructions>
NEXT <Counter>
```

<Counter> must be a local byte, integer or long integer variable. <Start>, <End> and <Step> are <counter> type compatible expressions. <Start>, <End> and <Step> expressions are computed only one time before starting loop.

<Counter> is affected to <Start> value at the beginning. At each loop <Step> value is added to <Counter> and if <Counter> is greater than <End> then loop is stopped.

For example

```
FOR a%=0 TO 15
OUT(IO1)=1<<a%
NEXT a%
```

At each execution of NEXT instruction, the multitask kernel launches the next task.

WHILE instruction

WHILE instruction allows the repeated execution of one or more instructions in accordance to an expression value.

WHILE instruction syntax is described below:

```
WHILE <Expression> DO
          <Instructions>
END WHILE
```

In this instruction, if <Expression> is false before the WHILE structure beginning there is no loop. While <Expression> is true <Instructions> are executed.

For example :

```
VEL%(X)=100                       ' Set normal velocity 100 %
STTA(X=2000)                      ' Start absolute move to 2000 position
WHILE  MOVE_S(X)  DO              ' Loop while motor is in movement
IF POS_S(X) >1000 THEN VEL%(X)=50  ' Slow velocity at middle distance
END WHILE                         ' End of loop
```

At each execution of END WHILE instruction, the multitask kernel launches the next task.

REPEAT instruction

REPEAT instruction allows the repeated execution of one or more instructions in accordance to an expression value.

REPEAT instruction syntax is described below :

```
REPEAT
<Instructions>
UNTIL <Expression>
```

In this instruction, if <Expression> is right before the REPEAT structure beginning, there is one loop. <Instructions> are executed unit <Expression> is right.

For example :

```
VEL%(X)=100                         ' Fast velocity
STTA(X=2000)                        ' Start absolute move to 2000 position
REPEAT
IF POS_S(X) >1000 THEN VEL%(X)=50   ' Slow velocity at middle distance
UNTIL NOT MOV_S(X)                  ' Loop until motor stop
```

At each execution of UNTIL instruction, the multitask kernel launches the next task.

4-3-5- Ladder task structure

It's a chart form which is composed with rungs. Each rungs can contain contacts, coils, counters and timers.

Free contact or free coil can also be added to the ladder task.

At the compile phase, the ladder task is translated in a basic task. This basic task can be displayed on a windows editor : file « LadderTaskName.tsk ».

4-3-6- Event task structure

Each extra task can manage about 16 events : 7 PLC inputs, 8 capture input and 1 timer.

Extensive events , tie to standard axis's board, are also free (see chapter Enhanced Event Function).

This task is defined once times in a project. When you want to create one, you must chose the event start mode.

A) Events configuration

At each power on of MCS, no events are configured. This configuration is realized in a normal basic task (initialization task) with the MODIFYEVENT instruction.

Syntax : MODIFYEVENT (<Mask>,<Delay>)

<Mask> : Integer expression to select the event

- ⇨ Bits 0...6 : activation of n° 1 to n° 7 input of the first input board detected by system (search in A slot to J slot). A positive edge will generate the event. The input takes account of the invert and filter parameters entered during the card configuration..
- ⇨ Bit 7 : Time base
- ⇨ Bits 8...15 : activate the C1 or C2 capture input associate with the Capture1 instruction of the 1 to 8 SRV 85 card detected by the (search A slot to J slot).

<Delay> : Delay of the time base between 10 ms and 30000 ms. If the time base is unused, the value of delay will be not treated.

After the configuration, the event task is executed if at least one event is detected. The maximum time between the detection of this event and the treatment of this event is the ageing time task.

If you want to modify the events configuration, the MODIFYEVENT instruction must be treated in a normal basic task or in the event task after the GETEVENT instruction.

B) Reading the events detected

The GETEVENT instruction is consumed and read the events detected.

Syntax : <Variable>=GETEVENT

<Variable> is an integer type with the same configuration of bits like the <Mask> parameter of MODIFYEVENT instruction.

Each bit assign to an event is set when the event is detected.

If an event appears during the execution of the event task, it is stored and treated as possible.

C) Clearing the events

The clearing of events is obtained with MODIFYEVENT(0,0) instruction.

D) Warnings

The RUN, HALT, SUSPEND, CONTINUE, STATUS instructions didn't have any effects on event task.

This task don't give the hand to the other task. So, it must be a short task with no locked instructions (ex : WAIT, MOVA ...).

This task mustn't have branch. The END PROG instruction must appear at the end of the task to launch the event detection again.

If the MODIFYEVENT instruction is used in an event task, a new detected event can be changed.

E) Example

```
Init Task
      PROG
      ....
      MODIFYEVENT(0183H,1000)      'events E1, E2, 1s time base,
      .....                        'Capture1 on axis board 1
      END PROG

Tâche EVENEM
      PROG
      Event%=GETEVENT
      IF Event%.1=1 THEN    'événement E1
         .
         .
      END IF
      IF Event%.2=1 THEN    'événement E2
         .
         .
      END IF
      IF Event%.8=1 THEN    'événement base de temps
         .
         .
      END IF
      IF Event%.9=1 THEN    'événement capture 1
         .
         .
         capture1(…         'relance la capture
      END IF
        END PROG
```

5- PROGRAMMATION OF MOTION CONTROL

5-1- Introduction

The MCS can manage from 1 up to 8 servo axis boards. Each board integrates a processor 32 MHz for the control, with an update time of 330 µs, and a FPGA to treat the encoder signals up to 2 MHz.

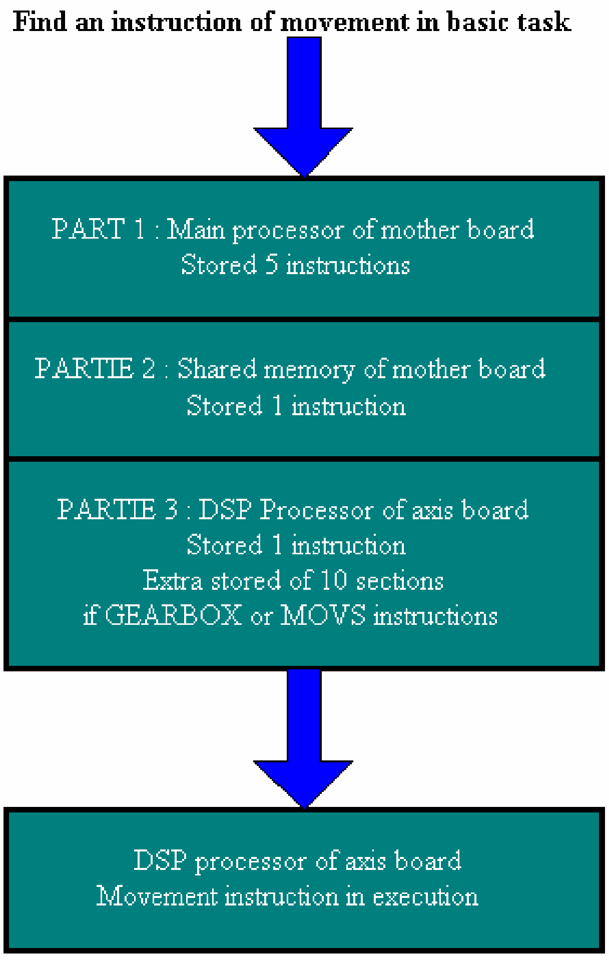
The motion commands are send to the axis boards from the main 32 bits processor, which cracks the pseudo-basic tasks.

The MCB software owns many high-level functions for the motions control: Positioning, gearbox, electronic cam, superposition, compensation, interpolation…

5-2- Motions buffer

The main processor of the MCS, which execute compiled tasks, exchange data with the axis boards.

The movement instructions (MOVA, STTA, MOVS, GEARBOX, STOP ...) are transmitted through a software buffer. Each axis owns his buffer, divided in three parts:


Find an instruction of movement in basic task
PART 1 : Main processor of mother board
Stored 5 instructions
PARTIE 2 : Shared memory of mother board
Stored 1 instruction
PARTIE 3 : DSP Processor of axis board
Stored 1 instruction
Extra stored of 10 sections
if GEARBOX or MOVS instructions
DSP processor of axis board
Movement instruction in execution

When a task have to launch a motion, it sends it to the concerned axis buffer:

✎ If there is no motion instruction running and the buffer is empty, the instruction crosses the parts 1, 2, 3 of the buffer, and is done immediately.

✎ If a motion instruction is already running, the new one is stored in the buffer.

✎ If the buffer is full, the task is waiting until a place is available.

The third part of the buffer has a specific storage zone of 10 parts for the instructions GEARBOX, GEARBOXM, MOVS, MOVSM, MOVSC :

✎ GEARBOX or GEARBOXM are made of one part if there is no acceleration parameter.

✎ GEARBOX or GEARBOXM are made of two parts if there is an acceleration parameter.

✎ MOVS or MOVSM or MOVSC are made of three parts according to the values of "slave distance", acceleration distance", "deceleration distance". For example:

```
MOVS(Y,X,Dmaster,100,0,0)        '1 part
MOVS(Y,X,Dmaster,100,100,0)      '1 part
MOVS(Y,X,Dmaster,100,0,100)      '1 part
MOVS(Y,X,Dmaster,100,20,0)       '2 parts
MOVS(Y,X,Dmaster,100,0,20)       '2 parts
MOVS(Y,X,Dmaster,100,20,20)      '3 parts
```

✎ The movement is stored in the 10 parts zone only when all the elementary parts can be stored there.

✎ The movement number follows the movement (and the parts) until the complete execution of the motion.

Some instructions permit to know the evolution of the elements inside the buffer:

✎ BUFMOV_S shows the number of waiting instructions in the buffer

✎ ORDER can affect a number to each instruction send to the buffer

✎ ORDER_S returns the number of movement is running

✎ MOVE_S equal zero if the buffer is empty and there is no movement running.

✎ Warning :

✎ The instructions STOP and SSTOP stop the running movement, and clear the axis buffer.

✎ If the axis is in open loop (AXIS(Axis)=Off), any motion instruction sawn in a task crosses the buffer and is directly consumed by the axis board without doing the movement.

5-3- Controlled / non controlled mode

5-3-1- Leading in non-controlled mode

An axis leads in non-controlled mode (open loop):

✎ Each time you switch on the MCS

✎ Each time you use the instruction AXIS(Axis)=Off from any task

✎ Each time you use the instruction CONS from any task

✎ For any axis following error (except if the SECURITY instruction is affected)

➭ With a forcing from the debug menus (close, open buttons…), from the communication menu (stop tasks, restart tasks, send tasks.).

The instruction AXIS_S(Axis) permit to read the state in which is the axis board.

If a movement instruction is send to an axis board set in open loop, the instruction will be consumed, but no motion will be done.

For example:

Process Task

```
PROG
...
...           ' The MCS detected a following error on one axis
...           '  => all the axis are leading in non-controlled mode
MOVA(X=1000)  ' The movement is consumed by the axis board
              ' but no done
OUT(S1)=1     ' Output S1=1
MOVA(X=2000)  ' The movement is consumed by the axis board
              ' but no done
OUT(S1)=0     ' Output S1=0
...           ' Output 1 was set for a very few time because the
              ' instruction Mova(X=2000)took very few times to the system
END PROG
```

5-3-2- Leading in controlled-mode

A servo axis can control and drive motions. For that, it must be in controlled mode.

An axis leads in controlled mode (closed loop):

➭ Each time you use the instruction AXIS (Axis)=On from any task

➭ With a forcing from the debug menus (close button)

The instruction AXIS_S(Axis) permit to read the state in which is the axis board and CONS_S(Axis) the output voltage.

For safety, it is necessary to affect a logical output of the MCS to control the “Enable / Disable” of the drives managed by the controller.

This output will be linked to the input “Enable” of the drive. The management of this output will be done in a non-stopping watching task, for example:

Task Fault

```
PROG
....
OUT(EnableDrive)=AXIS_S(Axis)      ' if axis in controlled-mode
....                               ' => OUT(EnableDrive)=1 else =0
END PROG
```

Attention :

➭ The activation of Axis instruction is done after 3ms. For a good servo control :

```
Axis(X)=On
Wait Axis_S(X)=On
```

5-4- Setting an axis

5-4-1- Setting an axis

An axis must be set before using it.

The parameters access is from the MCB configuration screen after chosen the board, or from a basic line looking like “NameParameter_P” written in a task.

The simplest setting method is to input different parameters of each board from the configuration screen of the MCB, and to send them to the MCS with the communication menu “send set-up”. If, later, you have to set a parameter in real-time on the MCS, we can lead to the debug mode on the MCB and adjust it.

5-4-2- User Miscellaneous

Depending to the application, the mechanical (linear or rotation axis), we may affect to each axis a more easy unit: mm, pulse (encoder pulse * 4), degrees, radian, inch, round, or whatever.

Indeed, this unit is only used on the MCB screen, to be easier to understand and practice. This is the reason why you do not find it as the instruction “NameParameter_P”.

For example, if the selected unit is “mm”, in the “speed profile” menu of the MCB, speed’s unit is mm/s, and acceleration and deceleration mm/s^2.

However, the MCS does not see any other unit than pulses or points.

5-4-3- Encoder

Each servo board owns an encoder input to deal with the position feedback. It can be incremental (SRV15 and SRV85 boards), or absolute (SSI15 board).

A) Encoder type

➭ For an incremental encoder

Set in the field ‘resolution’ of the MCB (associated parameter ENCODER_P) the points number * 4 of the encoder.

➭ For a SSI encoder

Set in the field ‘number of points’ of the MCB the total number of points of the encoder (if encoder 1 round 360 points, write 360; if encoder 360 points 10 tours, write 3600).

Set in the field ‘scale’ (associated parameter ENCODER_P), the number of points for 1 round encoder.

Set in the field ‘number of bits’ the total number of bits of the SSI protocol managed by the encoder, and in ‘frequency’ its frequency.

B) Units per encoder revolution

➭ For an incremental encoder

Set in the field ‘units per encoder revolution’ of the MCB (associated parameter UNITREV_P), the distance of the mechanical in user unit for 1 encoder revolution. For example, for a pulley diameter 100mm and an encoder directly fitted on it, input the value 100*3.14 = 314.

➭ For a SSI encoder

Set in the field “scale / for’ (associated parameter UNITREV_P) the distance of the mechanical in user unit for 1 encoder revolution.

All these values for parameters, position, motion,… expressed in user units are translated in pulses inside the MCS, using the formula:

PulsesValueAxis = (UnitValueAxis * ENCODER_P(Axis)) / UNITREV_P(Axis)

C) Reversing the encoder direction (for incremental encoder only)

Tick the box ‘encoder direction invert’ of the MCB (associated parameter ENCINV_P) if you want to invert the counting direction without changing the wires.

D) Modulo axis (for incremental encoder only)

An infinite axis (without mechanical limits) must be declared in modulo axis, otherwise it could cross the counting limits of the encoder pulses (+/- 2^24 pulses), and then make a MCS failure.

Tick the box ‘modulo axis’ of the MCB (associated parameter MODULO_P) to declare the axis in modulo mode, and set its value in the field ‘value’ (associated parameter MODVAL_P).

For example, for a rotation axis expressed in degrees can be 360°.

5-4-4- Speed profile

A trajectory in positioning is made of the phases of acceleration, constant speed and deceleration.

The fields available from the board configuration in the MCB can give default values to these different phases. The values are taken in account every time you switch on the MCS. They are also used in the debug mode, and with the instructions ACC%, DEC%, VEL%.

Set in the field ‘default speed’ (associated parameter VEL_P), the default speed in units/s (range from 5 10^-5 to 1.5 10^6 pulses/s).

Set in the fields ‘default acceleration’ (associated parameter ACC_P) and ‘default deceleration’ associated parameter DEC_P), the default acceleration and deceleration in units/s^2 (range from 5 10^-5 to 1.5 10^6 pulses/s^2).

If you wish to modify these parameters in a basic task, use the instructions ACC, DEC, VEL, ACC%, DEC%, VEL%. The VEL_S permits to read the axis speed.

These values are always true, either with an axis in movement or stopped.

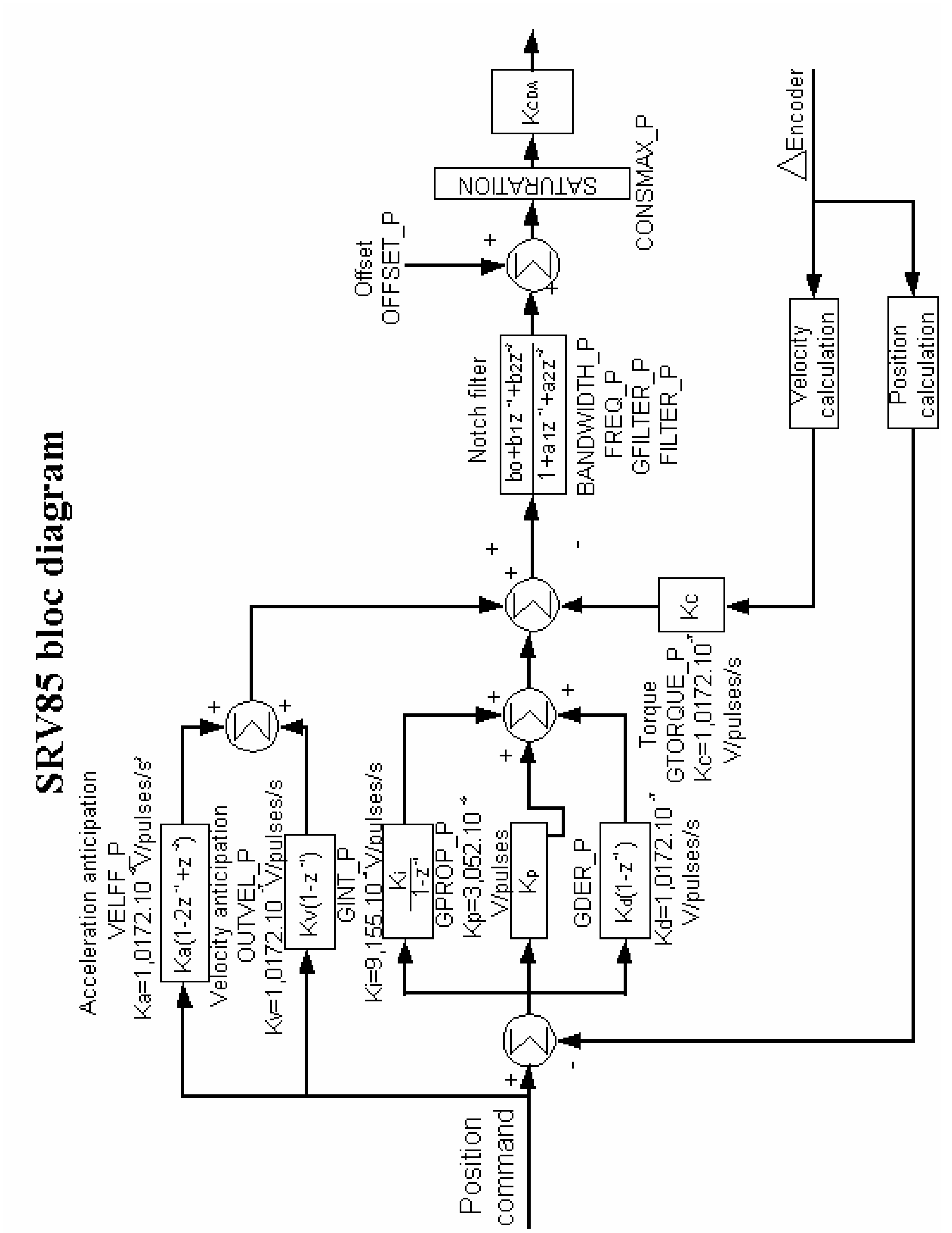
The acceleration / deceleration phases can be linear or sine (associated parameter SIN_P).

If sine is selected, a coefficient (associated parameter SINVAL_P) can insert a linear portion in the sine. For example:

5-4-5- Regulation

The control of an axis is adjustable from the corrector parameters. It can be programmed in torque mode or in speed mode (only for SRV 85, default value: speed mode).

A) General diagram



B) Gains

Adjustment of the controller on a infinite axis:

↳ Set the maximum following error to an important value (for example 2 or 3 rounds motor)

↳ Set the proportional gain (associated parameters GPRO_P) to a low value, and all the other parameters to zero.

↳ Set a value in the torque mode gain (associated parameter GTORQUE_P) if you are using the torque mode.

➷ Launch an infinite movement

➷ Increase the gain 'speed anticipation' (associated parameter OUTVEL_P) until the following error is nearly zero.

➷ Launch again some movements to increase the gain 'acceleration anticipation' (associated parameter VELFF_P) which plays a part in the transitory phases, until the following error is nearly zero.

➷ Increase the proportional gain to have more rigidity and precision in control as long as the system is stable (no oscillations seen)

➷ Increase the torque gain if you use the torque control mode, to have more rigidity and precision.

➷ If necessary, increase the derived gain (associated parameter GDER_P) to have more rigidity, and the integral gain (associated parameter GINT_P) to avoid static error.

➷ Set the maximal following error (associated parameter FEMAX_P) to twice the maximal value you have seen.

' Warning : If the control is not used in torque mode, the torque gain (GTORQUE_P) must be zero.

C) Analogue consign

➷ Tick the box 'invert voltage' of the MCB (associated parameter CONSINV_P) if you wish to invert the analogue consign without making wiring.

➷ Set in the field 'shift' (associated parameter OFFSET_P) the shift value in Volt if the axis drifts when it is in non-controlled mode (example: shift=0.04 V).

➷ Set in the field 'limit' (associated parameter CONSMAX_P) the value of the maximum analogue consign in volts. This can be useful in torque mode, to limit torque. But in general cases, it is better to keep its default value (+10 V)

D) Maximum following error

As soon as an axis leads in controlled mode, it is always controlled, in stop or in motion

If the difference between the calculated theoretic position and the real one given by the encoder feedback is bigger than the maximum following error, all the axis lead in non-controlled mode, and the watchdog contact is getting open (except if you use the instruction SECURITY).

The adjustment of this value is very important: a too small value stops untimely the axis control, a too big one interferes with security of electrical and mechanical devices.

Set in the field 'maximum following error' of the MCB (associated parameter FEMAX_P) the good value.

E) Position window

When we send an axis to a position, the MCS knows that the motion is over when the theoretic profile is achieved, and the real position is equal to the final one +/- the position window. For example, on a piercing machine for which you need an accuracy of +/- 0.1 mm, we take this value for the position window parameter.

Set in the field 'position window' of the MCB (associated parameter POSMIN_P) the required accuracy.

5-4-6- Notch filter (only SRV 85 board)

A frequency notch filter can be integrated in the control loop. It permits to attenuate or to suppress a band of frequency on the axis analogue consign.

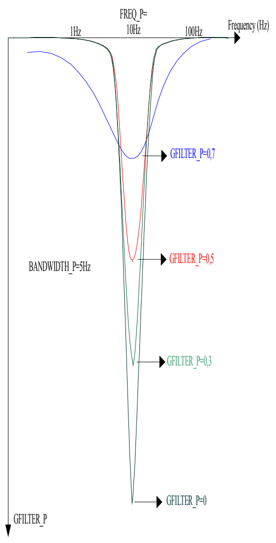
It is made with a central frequency, a bandwidth, and an attenuation gain.

Tick the box ‘Enable’ of the MCB (associated parameter FILTER_P) if you wish to use the filter.

Set in the field ‘central frequency’ (associated parameter FREQ_P) the frequency in Hz.

Set in the field ‘bandwidth’ (associated parameter BANDWIDTH_P) the bandwidth for –3dB in Hz

Set in the field ‘breaking factor’ (associated parameter GFILTER) the value of the attenuation, between 0 and 0.7.



Example :

The analogue consign has a continuous component of 4V and an oscillation period of 30 Hz, amplitude +/- 0.4V.

If FREQ_P=30 and GFILTER_P=0.7 the amplitude of oscillation will be +/- 0.28V.

If FREQ_P=30 and GFILTER_P=0.5 the amplitude of oscillation will be +/- 0.2V.

If FREQ_P=30 and GFILTER_P=0 there will be no oscillation any more.

5-4-7- Homing

A homing cycle is necessary on a servo board after each switch-on of the MCS if the board has an incremental encoder input and it drives a finite axis.

This cycle is to reset the pulses encoder counter of the axis board when some conditions are right.

The instruction HOME (Axis, number of type) launches the cycle. This instruction jams the task for all the cycle long.

Warning : For a modulo axis, the instruction HOME put temporarily to 0 his parameter MODULO_P. If you stop one task who is complying the instruction HOME, forecast to reforce MODULO_P to 1.

The instruction HOME_S shows if the homing cycle has been made. It is forced to zero at the beginning of the cycle, and set at the end.

The instructions ZERO_S and SENSOR_S return the logical states of the zero encoder signal and of the home/capture input of the board.

A) Types

The board can admit about 10 types: direct zero, on a sensor linked to the servo board or to an input board, on sensor and index encoder…

To zero directly the pulses counter, use HOME (Axis, 0) or CLEAR (Axis)

B) Speed

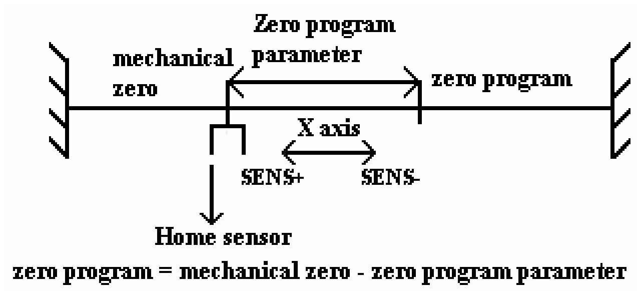
Set in the field 'home velocity' (associated parameter VOLHOM_P) the required speed in units/s. In the case of a fast homing, this will be done first with the default speed, then the homing velocity.

C) Shift

Set in the field 'shift' (associated parameter DISHOME_P) a value in units, if you wish that, after making its homing, the axis goes away for a distance equal to this parameter.

D) Program zero

When the axis meet its homing position, it zeros its counter and stops with repositioning on zero. This position is the zero machine. If you wish to work in another landmark, just set a value in the field 'program zero' of the MCB. (associated parameter ZERO_P). For example:



5-4-8- Software limits

On an axis with limited travel, it is possible to declare software limits maxi and mini. If the system detects that the axis is out of the limits, it sets the flag LIM_S(Axis). This flag can then be treated in a basic or ladder supervising task.

' Warning : The system modifies the state of LIM_S, LIMMIN_S or LIMMAX_S, but has no action on the axis management.

Tick the box 'enable' of the MCB (associated parameter LIM_P) if you wish to use limits.

Fill in the field 'mini' (associated parameter LIMMIN_P) with the value of the mini limit.

Fill in the field 'maxi' (associated parameter LIMMAX_P) with the value of the maxi limit.

The mini value must be lower than the maxi value.

5-5- Declaration of an axis in virtual mode

5-5-1- Declaration of an axis in virtual mode

From a basic task, it is possible to lead an axis board in virtual mode with the instruction LOOP(Axis)=On. In this mode, the board will simulate the encoder pulses in an intern way, so every command send will be made virtually.

' Warning : The analogue output will be even so managed.

This mode is interesting during the program development phase: we can test the global application without motors and drives connected.

It can also be used for some applications with synchronisation, which need a perfect stable master. For example:

X and Y are 2 servo-axis, to link in a gearbox with a ratio ½: X is the master, Y the slave.

Problem: X generates small vibrations, which could be amplified by Y.

Solution: Add a servo-board working in virtual mode, and master for X and Y.

Task Process1

```
PROG
....
AXIS(X)=On
AXIS(Y)=On
LOOP(Master)=On
AXIS(Master)=On
.....
GEARBOX (X,Master,1,1,0)
GEARBOX (Y,Master,1,2,0)
STTI (Master)=+
.....
END PROG
```

5-6- Positionning

5-6-1- Absolute movements

A) Start a movement : STTA

To launch a movement to an absolute position and not wait till it is over to continue the task execution, we must use STTA. This instruction is very useful if the speed or the position to reach changes during the motion. With this function, absolute error is minimal.

This instruction is not jamming or the task (except if the motion buffer is full).

It uses the current values of acceleration, deceleration and speed. The syntax is:

STTA (<Axis1>=<Position1> [{, <Axis2>=<Position2>}])

For example :

```
VEL%(X)=100               ' normal speed
STTA(X=2000)              ' Absolute start to position 2000
WAIT POS_S(X) >200        ' Wait position 200
OUT (A1)=1                ' Set an output
WAIT POS_S(X) >700        ' Wait 700
OUT (A1)=0                ' Reset an output
WAIT  NOT  MOVE_S(X)      ' Wait end of motion
```

In this example, during the movement, we can change outputs because the execution is not stopped.

If the instruction MERGE is activated and you send several STTA instructions in the axis board, movements will be done one after the other without passing through a zero velocity.

If the axis is modulo, a motion to a position will be done in the positive direction if the required position is positive, in the negative direction for a negative required position. For example:

Axis modulo 360°

Axis in initial position at 90°

```
STTA(X=-10) 'Movement in the - direction for a distance de 80°
WAIT  NOT  MOVE_S(X)
STTA(X=350) 'Movement in the + direction for a distance de 340°
```

```
WAIT  NOT  MOVE_S(X)
STTA(X=20) 'Movement in the + direction for a distance de 30°
WAIT  NOT  MOVE_S(X)
```

B) Movement : MOVA

The function MOVA sends one or several axis to an absolute position. It uses the current values of acceleration, deceleration and velocity. The syntax is:

MOVA (<Axis1>=<Position1> [{, <Axis2>=<Position2>}])

This function sends <Axis1> to the absolute position which value is <position1>. The program waits for the end of the motion to continue. The absolute positioning error is minimal.

For example:

```
MOVA(X=100)
CALL Piercing
MOVA(X=0)
```

The instruction MOVA stop the rest of the task until the movement is not over (condition MOVE_S(Axis)=0).

MOVA(X=100)	is equivalent as	STTA (X=100)
		WAIT NOT MOVE_S(X)

To have a better relative precision we can ask for an absolute movement using the current position: MOVA(X=POS_S(X)+100).

If the axis is modulo, a motion to a position will be done in the positive direction if the required position is positive, in the negative direction for a negative required position. For example:

Axis modulo 360°

Axis in initial position at 90°

```
MOVA(X=-10) 'Movement in the - direction for a distance de 80°
MOVA(X=350) 'Movement in the + direction for a distance de 340°
MOVA(X=20) 'Movement in the + direction for a distance de 30°
```

C) Movement triggered on a position : MOVAP

This instruction is equivalent to MOVA, and moreover it includes a condition of starting on a position of another axis.

D) Movement triggered on a capture input : MOVAC (only on SRV 85)

This instruction is equivalent to MOVA, but it includes also a start condition given by a fast registration input.

E) Trajectory : TRAJ

The function Trajectory is made to simplify the definition of complex movements.

To do a movement with a specific velocity, acceleration or deceleration, these parameters must be set before launching the movement. With the function TRAJ, you can launch an absolute movement on one or several axis.

Syntax of the TRAJ function:

TRAJ (<Parameter>=<Value> {, <Parameter>=<Value>})

Example :

```
TRAJ( POS(X)=500, VEL(X)=2000, POS(Y)=POS_S(Y)+200, ACC(Y)=500)
```

This example is similar as:

```
VEL(X)=2000
STTA(X=500)
ACC(Y)=500
STTA(Y=POS_S(Y)+200)
```

The TRAJ instruction don't stops the execution of the.

If the instruction MERGE is activated and we send several TRAJ instructions in the axis board, the movements will be done one after the other without passing through speed zero. For example:

```
MERGE(X)=On
TRAJ(POS(X)=500,VEL(X)=2000)
TRAJ(POS(X)=1000,VEL(X)=50)  ' Change the speed
                             ' to a lower value at position 500
```

5-6-2- Relative movements

A) Start movement : STTR

For launching a movement to a relative position and not waiting its ending for continuing the execution of the task, you have to use STTR. This instruction is useful if the speed or the final position may change during the motion. With this function, the absolute error is minimal.

This instruction does not stop the task (except if the movements buffer is full).

It uses the current values of acceleration, deceleration and velocity. The syntax is:

STTR (<Axis1>=<Distance1> [{, <Axis2>=<Distance2>}])

For example :

```
VEL%(X)=100                  ' Fast speed
P!=POS_S(X)
STTR(X=2000)                 ' Relative start to position 2000
WAIT  (POS_S(X)-P!) >100     ' Wait position +100
VEL%(X)=10                   ' Slow speed
WAIT  NOT  MOVE_S(X)         ' Wait end of motion
```

In this example, during a movement, the speed can be modified because the task execution does not stop.

If the instruction MERGE is activated and we send several STTR instructions in the axis board, the movements will be done one after the other without passing through speed zero.

B) Movement : MOVR

The function MOVR send one or several axis to a relative position. It uses the current values of acceleration, deceleration and velocity. The syntax is:

MOVR (<Axis1>=<Distance1> [{, <Axis2>=<Distance2>}])

This function send <Axis1> to a relative position <Distance1>. The program waits for the end of the movement before going on. The absolute positioning error is minimal.

For example :

```
FOR I#=1 To 10
  MOVR(X=100)
  CALL Piercing
NEXT I#
```

The instruction MOVR stops the task until the movement is finished (condition MOVE_S(Axis)=0).

MOVR(X=100) is equivalent as STTR (X=100)
WAIT NOT MOVE_S(X)

5-6-3- Infinite movements

To launch a continuous movement, you have to use the instruction STTI. The selected axis will move with their current speed.

This instruction does not stop the task execution (except if the movements buffer is full).

The instructions STOP or SSTOP are necessary to stop a continuous movement. The direction of the movement is defined by the characters '+' or '-'.

Syntax :

STTI (<Axis1>=<Direction1> {,<Axis2>=<Direction2>...})

Example :

```
WAIT  INP(PlusKey)=On
STTI(X=+)
WAIT  INP(PlusKey)=Off
STOP(X)
```

5-6-4- Stop a movement

To stop a movement, you have to use the instructions STOP or SSTOP. They stop the selected axis using the programmed deceleration, and then empty the movements buffer.

The STOP instruction stops the task execution until the movement is finished (condition MOVE_S(Axis)=0), and SSTOP does not stop it.

Syntax : STOP (<Axis1> {,<Axis2>...})

Example : movement up to detect a sensor

```
STTI(X=+)
WAIT  INP(Sensor)=On
STOP(X)
```

The instruction AXIS(Axis)=Off stops the movement too, but without any control because the control-mode is inhibited.

5-7- Synchronization

5-7-1- Gearbox

A) Electric shaft : GEARBOX

The GEARBOX function permits to realise an electric shaft between a master axis and a slave axis.

Syntax : GEARBOX(<Slave>, <Master> , <Numerator>, <Denominator>, <Reversible>,

[<Acceleration>])

In a synchronisation function such as an electric shaft, the master axis can be a servo axis or an axis encoder. The slave axis must be a servo axis.

<Numerator> / <Denominator> defines the ratio between the master axis and the slave axis. The ratio can be either positive or negative. The sign of <Numerator> shows the usual direction of the slave and the sign of <denominator> the master axis usual direction.

If the synchronisation have to be reversible, <Reversible> must be set.

➯ non-reversible mode : if the master moves in the opposite direction regarding its usual way, the slave stops. It will go on again as soon as the master will be back in his main direction and reach the position it has when the slave had stopped.

➯ reversible mode : The slave follows the master in both directions.

<Acceleration> is an optional parameter. It represents a distance for the master, where the slave is in acceleration phase before reaching the velocity ratio required.

➭ If it is affected to zero, the master will have to be stopped for executing the GEARBOX instruction.

➭ If it is used, you can allow the case where the master is already in movement during the execution of the instruction GEARBOX.

If the synchronisation must be reversible, <Reversible> must be true.

This instruction does not stop the execution of the task (except if the movement buffer is full). As long as the link between master and slave is not broken, the instruction MOVE_S(Slave) will be equal to 1 (even if the slave is stopped).

➭ If the reversible mode has been selected and a <acceleration> value programmed, if the master changes of direction whereas the slave is in acceleration phase, the slave will change his direction too, so it comes back to its start position. Here stops the slave; it will go on again when the master will come back in the initial direction and reach the master position where the slave has stopped.

➭ The parameter <Acceleration> can be used only with a SRV85 board.

➭ The reversibility is infinite on the SRV85 board. It is limited to several rounds on the other servo boards (SRV 15, SRV 1524, SSI 15).

Example : The slave axis will turn 2 times faster as the master, and on the opposite direction.

```
GEARBOX (Slave, Master, -2, 1, True)
...
STOP(Slave)
```

B) Multi-masters electric shaft : GEARBOXM

This instruction is equivalent as GEARBOX, but the master is here the resultant of 2 master sources. You can use this function for dealing with applications using differential functions for example.

C) Modification of an electrical shaft ratio : GEARBOXRATIO

This function changes the reduction ratio of an electrical shaft. The syntax is :

GEARBOXRATIO(<SlaveAxis>,<Ratio>,[<AccRatio>])

<SlaveAxis> : must be a servo axis board of an electrical shaft link. The shaft ration is defined by : <Ratio> × <Numerator> / <Denominator>. <Numerator> and <Denominator> are the parameters the GEARBOX instruction.

This is an unstopped instruction and it can change the ratio without to stop the electrical shaft. The ratio can be positive or negative.

<AccRation> : Optionnal parameter allowing a progessive modification of an electrical shaft ratio. This acceleration is express in increment / 0,33ms

Ex : For a ration of 1 to 1,2 with a acceleration value of 0,0001 we put ((1,2-1/0,0001)*0.33=660ms

<u>Warning :</u> This function modify a general facto (K1) who is used in all master's distance of synchro functions : MOVS, CORRECTION, CAM …

Example :

```
GEARBOXRATIO (Slave,Master,1.1)

MOVS (Slaver,Master,100, … ) 'intern master distance = 100 *1,1
```

Example :

```
GEARBOX(Esclave, Maître, 1, 2, 1)  'Nominal ratio  : ratio 0.5

GEARBOXRATIO(Esclave,1.2)          '20% increase
```

```
                                        'Ratio  : (1.2×1/2)=0.6
GEARBOXRATIO(Esclave,0.8)               '20% reduction
                                        'Ratio  : (0.8×1/2)=0.4
```

5-7-2- Synchronised movements

A) Movement : MOVS

The instruction MOVS allows making a synchronisation between a slave and a master axis.

This instruction does not stop the task (except if the movements buffer is full).

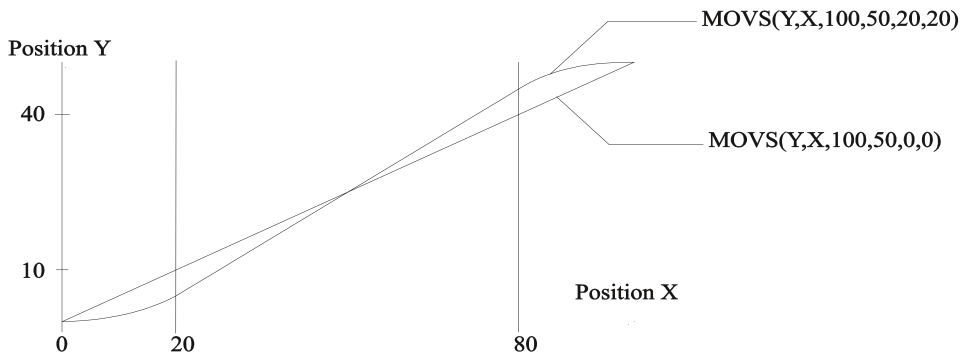
Example :

```
MOVS(Y, X, 10, 20, 0, 0 )    'for a relative movement of 10 units
                             'on X, Y moves of 20
```

Syntax : MOVS(<Slave>, <Master>, <MasterDist>, <SlaveDist>, <AccelDist>, <DecelDis>)

It is used for synchronising the slave and master axis for a precise distance of the master axis, with separately variable phases of acceleration and deceleration on the slave axis. The master axis can be a servo axis or an axis encoder. The slave axis must be a servo axis.

For example :



This example shows 2 synchronised movements with and without the acceleration and deceleration phases. When there is no acceleration and deceleration phases, the master axis and the slave axis must have the same speed to limit the transitory phases. If the speeds are very different, acceleration and deceleration must be adjusted to avoid mechanical diseases.

The speeds are not necessary the same and depend on the acceleration and deceleration phases, just because the system has to respect distances.

The function LOADS and STARTS are also used to make synchronised movements. The movement calculations and the synchronisation are then dissociated, so you can start at the same time several axes synchronised on the same master.

Example LOADS and STARTS :

```
LOADS(Y, X, 10, 20, 0, 0 )   'Loading synchronisation for y
LOADS(Z, X, 10, 40, 0, 0 )   'Loading synchronisation for z
STARTS(Y,Z)                  'Simultaneous launch of the 2 synchronisation
```

B) Movement triggered on position : MOVSP

This instruction is equivalent as MOVS, but it includes a start given by the position of another axis.

Warning : the instruction MOVSP who had the 3 phases (acceleration, velocity, deceleration) can be ignored if the instruction is following by a positionnement movement. In this case, you must decompose MOVSP in several instructions.

C) Multi-masters movement MOVSM

This instruction is equivalent as MOVS. Moreover, the master corresponds to the resultant of two master sources. This can be useful to deal with applications using differential functions.

D) Movement triggered on a capture input : MOVSC (only on SRV 85)

This instruction is equivalent as MOVS, but it includes a start given by a fast input 'capture'.

As long as the link between axis and slave is not broken, the instruction MOVE_S(Slave) will be equal to 1 (even if the slave axis is stopped).

5-7-3- Relative cam

A) Introduction

a) Definition of a cam

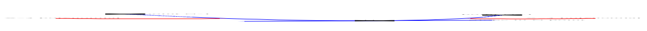
The cam function permits to realise a cam profile on a slave axis linked to a master axis. This profil is defined with a points array. Each servo board can store up to 5 cans and 310 points for the 5 cams.

Each point is defined as a master position and a slave one.

Between two consecutive points, the MCS will execute a trajectory calculated with a 3rd order polynomial.

The values given to the master positions inside the array must be increasing.

For calculating the input and output slopes of the cam, the array will need two other points: the input point, and the output one. They can be calculated by the MCB software or by the user, depending of the user's software level requirements.



The slope in N is the slope of the straight line connecting N-1 and N+1.

b) Finite or infinite came

A mechanical cam corresponds to a finite electronic cam. In the points array, the first and the last values of the slave position are similar. The slave movement will be a linear movement with a finite magnitude.

The electronic cam permits also to create a slave rotation infinite movement : the absolute slave position increases for each new master cycle.

Warning: If the master axis or the slave axis are infinite, they must be declared as modulo axes from the "configuration' tab of the MCB software.

Finite cam

Infinite cam

c) Principles for starting a cam

The co-ordinates values of the different points are set as absolute values. In some cases, you can the have the calibration between the mechanical and the electronic cams.

While starting the cam, the MCS considers that it is already on the first point of the trajectory, and so it executes the cam profile in relative position regarding this starting point. For example:

Cam array

Point	Master	Slave
1	0°	180°
2	20°	230°
3	50°	250°
....	....	....

Before running the cam, the master absolute position is 5°, and the slave one is 10°. The two axes are stopped.

We start the cam execution: the slave axis does not move.

The master axis moves to a relative position of 20 – 0 = 20° (absolute position 5 + 20 = 25°)

➮ The slave axis shifts of 230 – 180 = 50° (absolute position 10 + 50 = 60 °)

The master axis moves to a relative position of 50 – 20 = 30° (absolute position 25 + 30 = 55°)

➮ The slave axis shifts of 250 – 230 = 20° (absolute position 20 + 60 = 80 °)

B) 1er step: simple programming

The first step permits to run a cam profile on an axis very easily. For that we use the CAM instruction.

This profile is made from a points array, created from the data editor. You can access it from the 'global variables' menu of the MCB software (variable type: cam table).

Each point is made of a master and a slave position. After filling all the points in, the input and output points are automatically calculated when you click on the box 'Calculate': these values are influenced by the single-shot or permanent, finite or infinite characteristics of the cam.

The syntax of the CAM function is:

CAM(<Slave>, <Master>, <Table>, <Single-shot>, <Reversible>, <PositiveDirection>, <Gain>)

<Slave> : Slave axis name, on which is applied the cam (servo board : SRV15, SRV85...)

<Master> : Master axis name (servo or encoder board : SCD22, SRV15, SRV85 ...)

<Table> : Cam table name, set from the data editor (variable type 'cam table').

The values given to the master positions inside the array must be increasing.

For calculating the input and output slopes of the cam, the array will need two other points: the input point, and the output one. From the data editor, just tell if the cam is finite or not, and single-shot or not. Then just click on the box 'Input and output points calculation).

<Single-shot> : Define the automatic re-looping of the cam.

Zero value means that the cam will loop on itself until a stop is required, 1 is for a cam to be done once.

<Reversible> : Tells if the <Slave> must follow the <Maitre> in both directions.

✎ Input 0 for a non-reversible cam: if the master moves in the opposite way as the one defined in <PositiveDirection>, the slave stops. It will start off again when the master will go in the right way and pass by the position where the slave stopped.

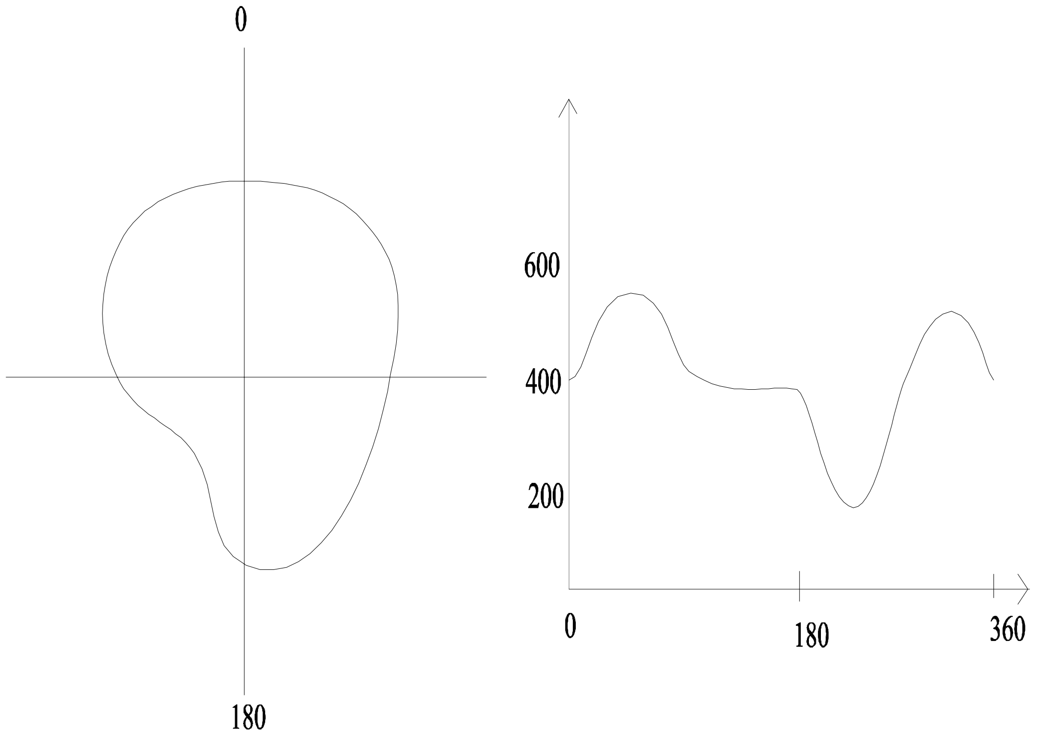
✎ Input 1 for a reversible cam: The slave follows its cam profile whatever is the master direction.

<PositiveDirection> : If the cam is not reversible, you must indicate the usual direction of the master.

Input 0 for a negative direction, 1 for a positive one.

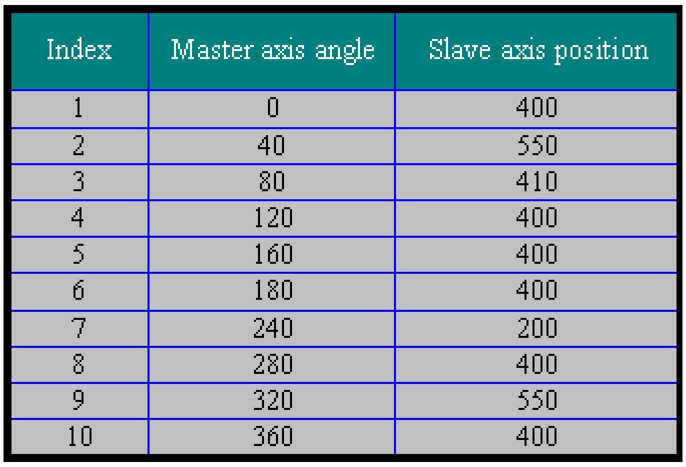
<Gain> : The <Gain> is used to multiply all the slave positions defined in the table with a coefficient.

Example :



In this example, the cam table defines the cam profile shown. Only one line in the program can launch the cam.

```
.....
CAM (Slave,Master,Camtable, 0, 1, 0, 1)   ' non single-shot (re-looping),
...                                       ' reversible
WAIT INP(StopInfo)                  ' Wait stop requirement
ENDCAM(Slave)                       ' Stop at the end of the current
                                    ' profile
```



Index	Master axis angle	Slave axis position
1	0	400
2	40	550
3	80	410
4	120	400
5	160	400
6	180	400
7	240	200
8	280	400
9	320	550
10	360	400

C) 2nd step: advanced programming

The second step permits to launch a cam profile which table points can be input or modified from a basic task of the MCS (for example, table defined on the Dialog 640 terminal).

You can also chain different cam profiles.

a) Creation of a real table

In the MCB project, it is necessary to create a real table, which have the following structure:

element 1 = Input point master

element 2 = Input point slave

element 3 = First point master

element 4 = First point slave

....

element w = Last point master

element x = Last point slave

element y = Exit point master

element z = Exit point slave

The real array contains (NumberCamPoints + 2) * 2 elements. During the declaration of the array from the MCB, it can be interesting to set a higher size, to add some points if necessary.

For example, if you declare a cam table with 10 elements at the address 100, it will be made of (10 + 2) * 2 = 24 real variables in the addresses 100 to including 123.

address 100 : Input point Master	address 101 : Input point slave
address 102 : Point 1 Master	address 103 : Point 1 slave
.....	.....
address 121 : Point 10 Master	address 122 : Point 10 slave
address 122 : Exit point Master	address 123 : Exit point slave

b) Calculating input and output points

In the second programming step, the programmer has to calculate the input and output cam points.

➷ Current cam Cc linked on itself or on a previous cam Cp or next Cn:

InputPointMasterCc = FirstPointMasterCc – (LastPointMasterCp – LastButOnePointMasterCp)

InputPointSlaveCc = FirstPointSlaveCc – (LastPointSlaveCp – LastButOnePointSlaveCp)

OutputPointMasterCc = LastPointMasterCc + (SecondPointMasterCn – FirstPointMasterCn)

OutputPointSlaveCc = LastPointSlaveCc + (SecondPointSlaveCn – FirstPointSlaveCn)

➷ Current cam Cc with stopped start:

InputPointMasterCc = FirstPointMasterCc – (LastPointMasterCc – LastButOnePointMasterCc)

InputPointSlaveCc = SecondPointSlaveCc

➷ Current cam with stop at the end of profile:

OutputPointMasterCc = LastPointMasterCc + (SecondPointMasterCc – FirstPointMasterCc)

OutputPointSlaveCc = LastButOnePointSlaveCc

For example:

Cam table

Point	Master	Slave
Input	Me	Ee
1	M1	E1
2	M2	E2
3	M3	E3
4	M4	E4
5	M5	E5
6	M6	E6
Exit	Ms	Es

Warning: we must have Me < M1 < M2 ... < Ms

In the case of a unique single-shot cam (executed once, beginning and ending stopped)

Me = M1 - (M6 - M5)	<=> 1ast point – (last point – last but one point)
Ms = M6 + (M2 – M1)	<=> last point + (2nd point – 1st point)
Ee = E2	<=> 2nd point
Es = E5	<=> last but one point

In the case of a unique cam non-single-shot (execution N times of the same profile)

Me = M1 - (M6 - M5)	<=> 1st point – (last point – last but one point)
Ms = M6 + (M2 – M1)	<=> last point + (2nd point – 1st point)

Ee = E1 – (E6 – E5)	<=> 1ast point – (last point – last but one point)
Es = E6 + (E2 – E1)	<=> last point + (2nd point – 1st point)

c) Loading a cam

To load a cam in a servo board, use the instruction LOADCAMEX.

Its syntax is : LOADCAMEX(<Slave>,<Master>,<NumberCame>,<Table>,<Number>, <FirstPolynomial>,<SingleShot>,<Reversible>,<PositiveDirection>,<NumberNextCam>, <NumberPreviousCam>)

<Slave> : Slave axis cam for which the cam will be applied (servo board: SRV15, SRV85...)

<Master> : Master axis name (servo or encoder board : SCD22, SRV15, SRV85...)

<NumberCam> : number of the cam (from 1 to 5)

<Table> : Name of the declared table in the global variables of the MCB (rtype real)

<Number> : Number of elements of the table to define the cam

<Number> = (NumberCamPoints + 2) * 2

<FirstPolynomial> : A servo board contains a global table of 310 polynomials for the 5 cams. Input a value from 1 to 310 to tell where the first polynomial of the cam will be stored in the global table of the board. Warning: <FirstPolynomial> + <NumberCamPoints-1> must be lower than 310.

<SingleShot> : Define the automatical re-looping of the cam.

➯ 0: Re-looping cam, it will be stopped only when the stop instruction will be executed.

➯ 1: Single-shot cam

<Reversible> : Tell if the <Slave> must follow the master in both directions.

➯ Input 0 for a non-reversible cam: if the master moves in the opposite way as the one defined in <PositiveDirection>, the slave stops. It will start off again when the master will go in the right way and pass by the position where the slave stopped.

➯ Input 1 for a reversible cam: The slave follows its cam profile whatever is the master direction.

<PositiveDirection> : If the cam is not reversible, you must indicate the usual direction of the master. Input 0 for a negative direction, 1 for a positive one.

<NumberNextCam> : Input 0 if the cam must not be followed by another one. If it is not the case, input the number of the next cam, from 1 to 5.

<NumberPreviousCam> : Input 0 if the cam will not start at the end of another one. If it is not the case, input the number of the previous cam (from 1 to 5).

d) Launching a cam

To launch the execution of a cam, use the instruction STARTCAM.

Its syntax is : STARTCAM(<Slave>,<Master>,<NumberCam>)

<Slave> : Slave axis cam for which the cam will be applied (servo board: SRV15, SRV85...)

<Master> : Master axis name (servo or encoder board : SCD22, SRV15, SRV85...)

<NumberCam> : number of the cam (from 1 to 5) of the servo board <Slave>

e) Example 1 : 6 points cam, infinite, non single-shot

```
PROG
.....
' Definition of the table in the real array TabCam
TabCam[3]=0           'point 1 Master
TabCam[4]=0           'point 1 Slave
TabCam[5]=30          'point 2 Master
TabCam[6]=20          'point 2 Slave
TabCam[7]=90          'point 3 Master
TabCam[8]=76          'point 3 Slave
TabCam[9]=270         'point 4 Master
TabCam[10]=283        'point 4 Slave
TabCam[11]=330        'point 5 Master
TabCam[12]=338        'point 5 Slave
TabCam[13]=360        'point 6 Master
TabCam[14]=360        'point 7 Slave
TabCam[1]=TabCam[3]-(TabCam[13]-TabCam[11])      'Input point Master
TabCam[2]=TabCam[4]-(TabCam[14]-TabCam[12])      'Input point Slave
TabCam[15]=TabCam[13]+TabCam[5]-TabCam[3]        'Output point Master
TabCam[16]=TabCam[14]+TabCam[6]-TabCam[4]        'Output point Slave
' Loading of cam number 1
LOADCAMEX(Slave,Master,1,TabCam,16,1,0,1,0,0,0)
' Launching of the cam number 1
STARTCAM(Slave,Master,1)
.....
WAIT INP(StopInfo)                        ' Wait for stop requirement
ENDCAM(Slave)                             ' Stop at the end of the
.....                                     ' current cycle requirement
END PROG
```

f) Example 2 : Chaining cams

Here is a cycle made of three cams: C1 with an input profile single-shot, C2 repetitive, and C3 with an output profile single-shot.

C1 is chained with C2 and C2 to C3.

```
PROG
.....
' Loading cam n °1  : 10 points, single-shot, followed by cam C2
LOADCAMEX(Slave,Master,1,TabCam1,24,1,1,1,0,2,0)
' Loading cam n °2  : 36 points, non single-shot, followed by cam C3
LOADCAMEX(Slave,Master,2,TabCam2,76,100,0,1,0,3,1)
```

```
' Loading cam n °3  : 6 points, single-shot

LOADCAMEX(Slave,Master,3,TabCam3,16,200,1,1,0,0,3)

' It 's preferable to declare came 3 as previous herself because if during the
passage of cam 2 to cam 3 consequence of End Cam instruction, a small backing
is done, the End Cam instruction is not valided.

' Launching of cam C1 => execution of C1, then C2

STARTCAM(Slave,Master,1)

WAIT CAMNUM_S(Slave)=2            ' Wait execution of C2

.....

WAIT INP(StopInfo)                ' Wait for stop requirement

ENDCAM(Slave)                     ' Stop cam 2 at the end of profile

                                  ' and then cam 3

WAIT NOT CAM_S(Slave)             ' Wait for end of cam 3

.....

END PROG
```

D) State of the cam

Three functions can show the current state of a servo board running a cam.

➭ Instruction CAM_S : permits to know if a cam from the board is running

Example :

```
IF NOT CAM_S(Slave) THEN PRINT «  Stopped cam  »

IF CAM_S(Slave) THEN PRINT «  Running cam  »
```

➭ Instruction CAMNUM_S : permits to know the number of the running cam. The returned value has got a sense only if CAM_S is set.

Example :

```
IF CAMNUM_S(Slave)=1 THEN PRINT «  Cam 1 running  »

IF CAMNUM_S(Slave)=2 THEN PRINT «  Cam 2 running  »
```

➭ Instruction CAMSEG_S : permits to know which equation number of the cam is running. The returned value has got a sense only if CAM_S is set.

Example :

```
IF CAMSEG_S(Slave)=1 THEN PRINT «  Cam between point 1 and point 2  »

IF CAMSEG_S(Slave)=2 THEN PRINT «  Came between point 2 and point 3  »
```

E) Stop of a cam

The function ENDCAM stops the slave movement at the end of the cycle, while the functions STOP stop it immediately. The syntax of the instruction ENDCAM is : ENDCAM (<Axis>)

Warning:

If ENDCAM is applied to a cam which has been declared in non-single shot and linked with another one, the cam ends its profile and goes on to the next.

F) Warning

➭ The values given to the master axis in the table must be crescent.

↳ The difference between 2 successive points of the master or the slave must not be higher than +/- 2^{21} pulses.

↳ This difference must not be too small. It is better for the system to pass through all the points, even at maximal speed (sampling period 330 µs).

G) Movement triggered on capture input : CAMC, STARTCAMC (only for SRV 85)

This instruction is equivalent as CAM or STARTCAM, but moreover it includes a condition of trigger given by a fast input 'capture'.

H) Modification of a point in a servo board : LOADPOINT

This instruction allows to change two polynomials with a camtable point in a servo board. The Syntax of this instruction is :

LOADPOINT(<Slave>,<Master>,<CamTable>,<Number>,<FirstPolynomial>,
<VariableIndex)

<Slave> : Slave axis name where the cam is executed (servo board : SRV15, SRV85...)

<Master> : Master axis name (Servo or encoder board : SRV15, SRV85, SCD22...)

<CamTable> : Cam table name which is defined in the global variables tab of the MCB software (« real » variable type).

<Number> : Number of cam table elements. Number = (NumberCamPoints + 2) × 2

<FirstPolynomial> : A servo baord has a global table of 310 polynomials for all the five cams. Put a value between 1 and 310 to indicate where the first polynomial of the cam will be stored in the global table of the board.

<VariableIndex> : Variable index of the <CamTableau> where the modification is. The modifications of the polynomials depend on the slave and master axis position. You can use LOADPOINT one time to change this two informations.

5-7-4- Absolute cam

A) Presentation

The cam function permits to realise a cam profile on a slave axis linked to a master axis. This profil is defined with a points array. Each servo board can store up to 5 cans and 310 points for the 5 cams.

Each point is defined as a master position and a slave one.

Between two consecutive points, the MCS will execute a trajectory calculated with a 3rd order polynomial.

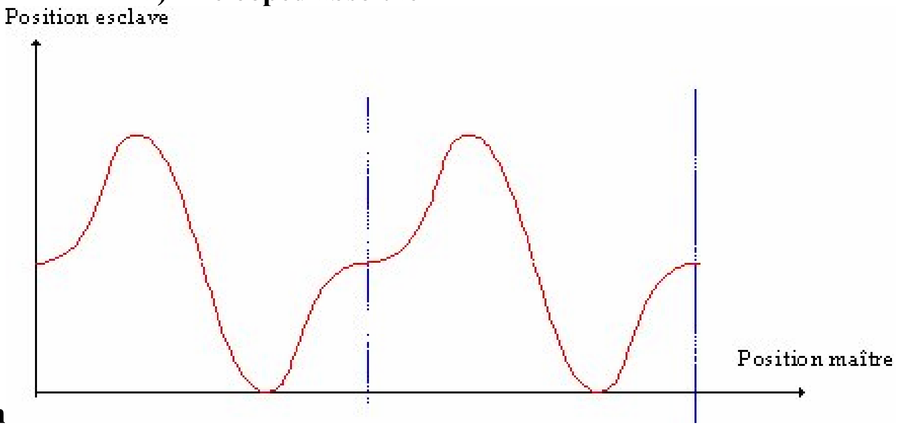
The values given to the master positions inside the array must be increasing.

For calculating the input and output slopes of the cam, the array will need two other points: the input point, and the output one. They can be calculated by the MCB software or by the user, depending of the user's software level requirements.

Contrary to relative cam, in a absolute cam, there 's one and only one position of the slave for each position of the master.

ATTENTION : The master's axis don't must have a ZERO program.

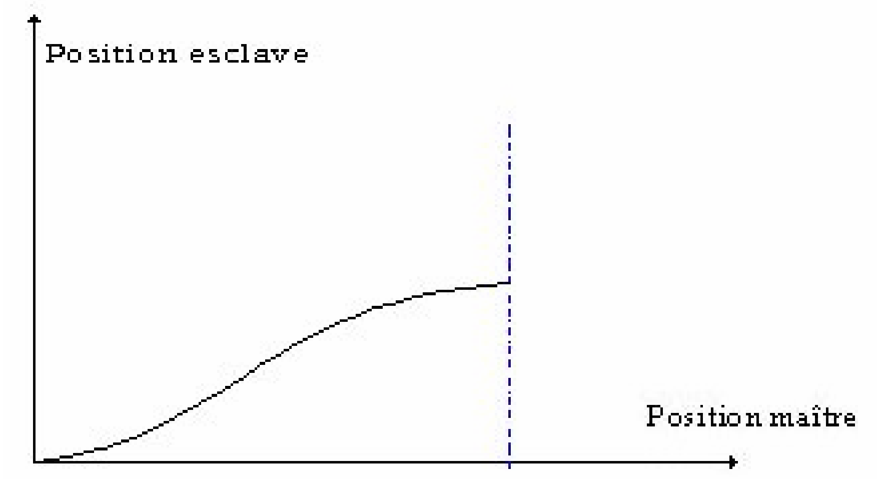
a) Relooped absolute cam



A relooped cam is define on the totality of the modulo of the master's axis. We can use only one cam to describe the cycle at a precise moment. It's possible to embraye the cam anywhere of cycle if the slave is placed previous at the correspoding position in the cam.

In the case where the cam is not reversible, a way of coupling must be define.

b) No relooped absolute cam



In the case of a no relooped cam, it‘s imperious to define a way of coupling which be reversible or not.Like this was define the manner whose the points are read in the table. The table is always define by a master's position stricly growing.

The cam cannot be define on the totality of master's modulo.

We consider that the cam is embrayed when we pass by the first point of this in the define way of coupling. We can only sort of the cam if it was completely cover in the good way. If we sort of the definition range of the cam to a reversilbe cam, in behindrun, we block the slave at the firs point. You need then comeback to the first point of the came for new embraye.

B) Launch of a cam

a) Single shoot cam

The launch off a single shoot cam start at the first define point. Like that, we must move the slave on the first point and the master anywhere. After the launch, this one is embraye when the master pass by the first point in the chosen direction.

Exemple :

```
Tabvar[1]=150
Tabvar[2]=250
Tabvar[3]=200
Tabvar[4]=200
Tabvar[5]=250
Tabvar[6]=250
Tabvar[7]=300
Tabvar[8]=300
Tabvar[9]=360
Tabvar[10]=360
Tabvar[11]=420
Tabvar[12]=420

'Load the cam
  LoadAbsCamEx(Esclave,Maitre,1,TabVar,12,1,1,1,1,0,0)
  ' Calculate the slave positionreally used by slave
SlavePos!=TabVar[4]
SlavePos!=SlavePos!+OffsetEsclave
If SlavePos!>Modval_P(Esclave) Then SlavePos!=SlavePos!-Modval_P(Esclave)
If SlavePos!<0 Then SlavePos!=SlavePos!+Modval_P(Esclave)
' Moving the slave
Mova(Esclave=SlavePos!)
' Applicate offset immediatly
MasterOffset(Esclave,Maitre,OffsetMaitre,1000)
SlaveOffset(Esclave,OffsetEsclave,1000)
' Launch the cam
StartAbsCam(Esclave,Maitre,1)
…
…
          ‘ Load offset during the cycle
```

```
        OffsetEsclave= OffsetEsclave+10
SlaveOffset(Esclave,OffsetEsclave,0.1)
…
…
```

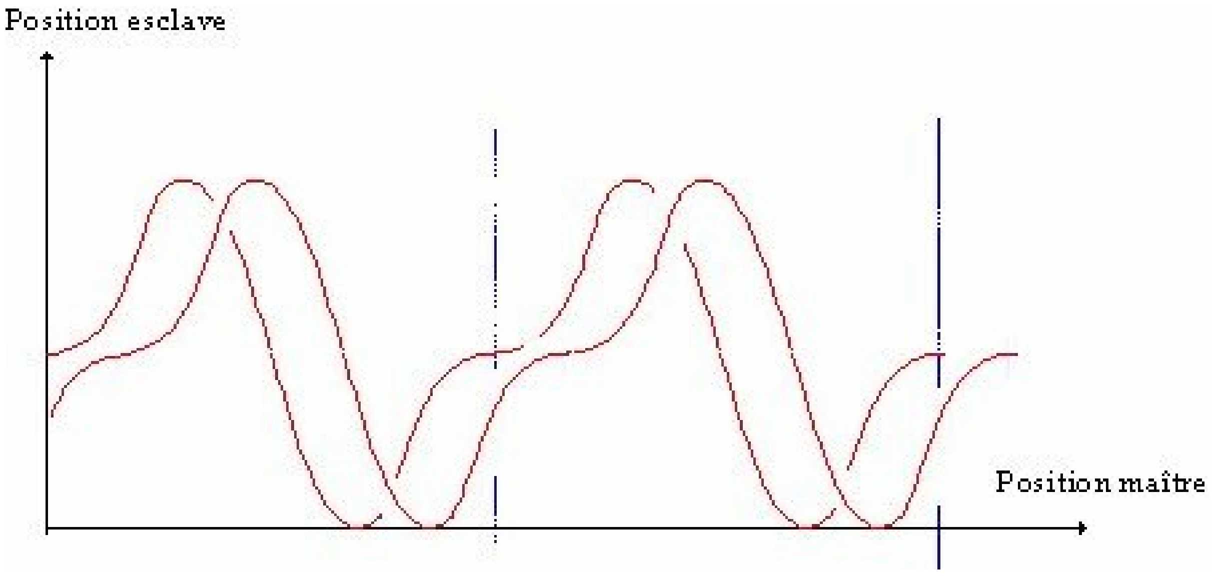
b) Relooped cam

```
Tabvar[1]=-180
Tabvar[2]=-180
Tabvar[3]=0
Tabvar[4]=0
Tabvar[5]=80
Tabvar[6]=80
Tabvar[7]=180
Tabvar[8]=250
Tabvar[9]=360
Tabvar[10]=360
Tabvar[11]=540
Tabvar[12]=540

'Load the cam
  LoadAbsCamEx(Esclave,Maitre,1,TabVar,12,1,0,1,1,0,0)
' Calculate the slave position, really used by slave
MasterPos!=Pos_S(Maitre)
' Calculate the master postion see by slave
MasterPos!=MasterPos!+OffsetMaitre
If MasterPos!>Modval_P(Maitre) Then MasterPos!=MasterPos!-Modval_P(Maitre)
If MasterPos!<0 Then MasterPos!=MasterPos!+Modval_P(Maitre)
' Research the corresponding position in the cam
SlavePos!=CamFromPoint(MasterPos  !,TabVar1,12)
SlavePos!=SlavePos!+OffsetEsclave
If SlavePos!>Modval_P(Esclave) Then SlavePos!=SlavePos!-Modval_P(Esclave)
If SlavePos!<0 Then SlavePos!=SlavePos!+Modval_P(Esclave)

' Moving of the slave
Mova(Esclave=SlavePos!)
' Applicate offset immediatly
MasterOffset(Esclave,Maitre,OffsetMaitre,1000)
SlaveOffset(Esclave,OffsetEsclave,1000)
' Launch of the cam
StartAbsCam(Esclave,Maitre,1)
```

```
…
…
        ‘ Load offset during the cycle
        OffsetEsclave= OffsetEsclave+10
SlaveOffset(Esclave,OffsetEsclave,0.1)
```

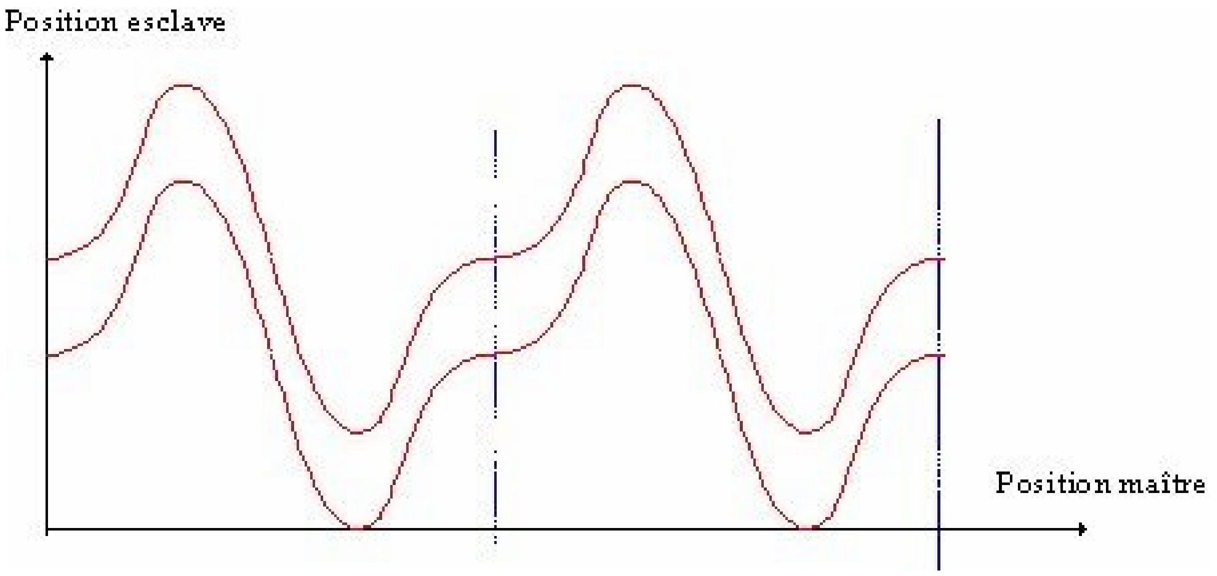
C) Dynamic dephasing

a) Master dephasing



The master dephasing had as effect to dephase the master cycle with regard to the slave. In the case of a relooped cam, it’s necesarry to respect this dephasing for positionnate the sleve with regard to the master. The master dephasing can be do progressively by the application with a acceleration parameter. At the opposite, for applicate a dephasing before the launch, it ‘s neccesary to use a great acceleration so as to assure that dephasing is activate before the starting.

b) Slave dephasing



The master dephasing had as effect to dephase the slave position but guard the phase with the master cycle. It's necessary for all case to take advise of this dephasing to positionning the slave according to the master. The slave dephasing can be do progressively by the application of a acceleration parameter. At the opposite, for applicate the dephasing before the starting of the cam, it's neccesary to use a great acceleration so as to assure that dephasing is activate before starting. You can also make the slave dephasing with compensation functions.

5-7-5- Compensation / uncompensation functions (SRV 85 only)

The compensation/uncompensation permit to make a dynamic phase displacement on a slave axis linked to a master axis. The link between the 2 axis must be an electric shaft (GEARBOX, GEARBOXM), synchronisation (MOVS, MOVSM, MOVSP, MOVSC) or cam (CAM, CAMC).

A) Immediate compensation function : ICORRECTION (SRV 85)

This function applies a correction movement on a slave axis during a distance of the master axis.

The slave must be linked to a master with a synchronisation or a gearbox function.

We add the following movement to the standard synchronisation movement::

While the master runs on a 'MasterDistance', we add a 'SlaveDistance' with acceleration and a deceleration on a master length of 'AccelDistance'.

Syntax :

ICORRECTION(<Slave>,<Master>,<MasterDistance>,<SlaveDistance>,<AccelDistance >)

<Slave> : Slave axis where is made the compensation (SRV 85 only)

<Master> : Master axis (servo board or encoder : SCD 22, SRV 15, SRV 85 ...)

<MasterDistance>: Correction interval (shift distance of the master during the compensation). If the distance is positive, the correction will be applied if the master runs in positive direction. If it is negative, the master must go in the negative direction.

<SlaveDistance> : Correction (shift length applied to the slave). If the sign of this parameter is positive, we will make a compensation. If it is negative, we will make an uncompensation.

<AccelDistancel> : Acceleration / deceleration (portions of acceleration or deceleration of the slave on a master distance).

```
Example  :
.....
GEARBOX(Slave,Master,1,1,1)
STTI(Master=+)
.....
CAPTURE2(Master,24,Off,0,0,Off)    ' Capture position of the master axis    '
                                   on a negative edge of the C2 input       '
                                   of the master axis
WAIT REG2_S(Master)=1              ' Wait capture detection
CorrectionValue!=Phase-REGPOS2_S(X)
ICORRECTION(Slave,Master,Interval!,CorrectionValue!,Dacc  !)
.....
```

B) Compensation function : CORRECTION (SRV 85)

This function permit to apply a correction movement on a slave axis during a distance of the master axis, on a position capture event. As all the axis parameters are stored in the slave axis board, as soon as the event is detected, the compensation is immediately done.

a) Launch the compensation

The compensation can be launched by 2 ways of capture:

➭ capture of the master axis position from a capture input of the slave axis (capture precision: 330 µs).

➭ capture physically made on the master axis and use of the capture diffused on the slave axis (precision capture: 0.1 µs).

Compensation template:

Syntax :

CORRECTION(<Slave>, <Master>, <Mode>, <Capture>, <Configuration>, <Window>, <Mini>, <Maxi>, <Inside>, <Phase>, <MasterDist>, <SlaveMaxVel>, <SlaveMinVel> , <MinAccDist>,[<WayCorrection>])

<Slave> : Slave axis name where is applied the compensation (SRV85 only)

<Master> : Master axis name (servo board or encoder: SCD 22, SRV 15, SRV 85 ...)

<Mode> : 0 for single-shot (only one compensation per instruction CORRECTION), 1 for permanent (a compensation applied on each capture until the program met the instruction STOPCORRECTION).

<Capture> : 0 for master position capture from the register 1 of the master axis (precision 0.1 µs), 1 for master position capture from the register 1 of the slave axis (precision 330 µs)

<Configuration> : Byte for hardware capture configuration

b0 : not used

b1 : capture on top Z

b2 : capture on input n°1 C1

b3 : capture on input n°2 C2

b4 : choice of the edge: 0 for positive, 1 for negative

b5 :7 : not used

<Window> : activation of a window limited by the positions <mini> and <maxi>of <master>

<Inside> defines if the test is made inside or outside of the limits. <Mini> must be lower than<Maxi>.

<Phase> : Theoretical phase. If the captured position is equal to this value, none compensation will be done.

<MasterDist> : Correction interval (shift distance of the master during the correction)

<SlaveMaxVel>: Max velocity added to the slave's current speed to apply the compensation. It is expressed in units/s (example : Vcurrent=1m/s, Vmaxi=0.5m/s => Vcorrection maxi = 1.5m/s).

<SlaveMinVel.>: Min speed retired to the slave's current speed to apply the compensation. It is expressed in units/s (example : Vcurrent=1m/s, Vmini=0.8m/s => Vcorrection mini = 0.2m/s).

<MinAccDist> : Mini acceleration / deceleration (portion of acceleration or deceleration of the slave for a distance of the master).

<WayCorrection> : type byte :

if parameter = 0 or not use => the compensation applicate will be negative or positive (more short)

if parameter = 1 => the compensation applicate will be positive

if parameter = 1 => the compensation applicate will be negative

Parameters Vmaxi and Vmini act as :

Max slave speed with correction = (RatioMasterSlave+SlaveMaxSpeed)*MasterSpeed

Min slave speed with correction = (RatioMasterSlave+SlaveMinSpeed)*MasterSpeed

RatioMasterSlave:

If gearbox with ratio ½ => RatioMasterSlave=0,5 (slave turn twice time less quick than master).

If gearbox with ratio 1 => RatioMasterSlave=1 (Slave & Master have the same speed).

Exemples :

We have a RatioMasterSlave = 1

MasterRunningSpeed = SlaverRunningSpeed = 600°/s

We want a max speed of slave with correction = 660°/s

We want a min speed of slave with correction = 480°/s

Max slave speed with correction = (RatioMasterSlave+SlaveMaxSpeed)*MasterSpeed => 660°=(1+ SlaverMaxSpeed)*600 => SlaveMaxSpeed = 0.1

Min slave speed with correction = (RatioMasterSlave+SlaveMinSpeed)*MasterSpeed
=> 480°=(1+ SlaverMinSpeed)*600 => SlaveMinSpeed = 0.2

CORRECTION(Slave,Master,Mode,Capture,Configuration,Windows,Mini,Maxi,Inside,
Phase,MasterDist,0.1,0.2, MinAccDist)

Warning :

➪ The previous compensation must be over to restart another one.

➪ If the master turns in the opposite direction of his normal one (given by the sign of <MasterDist>, the compensation is not applied and is lost.

➪ The slave must be previously linked to a master with a synchronisation or gearbox function before launching a compensation.

✎ If the compensation forces to go through the user-defined limits, the error is indicated and the max allowed correction is applied.

b) Stop the compensation

To stop the compensation function declared in permanent mode, you have to use the instruction STOPCORRECTION.

Syntax : STOPCORRECTION (<Slave>)

<Slave>: Slave axis name where is applied the compensation (SRV 85 only)

c) State of the compensation

To know the state of the compensation cycle, use the instruction CORRECTION_S.

Syntax : <Variable>=CORRECTION_S(<Slave>)

The returned value is a byte type.

b0 : when set, compensation required and waiting for capture event

b1 : when set, compensation running

b2 : when set, error detected: compensation out of limits (the limit compensation has nevertheless be applied).

To know the applied value of the compensation, use CORRPOS_S instruction

Syntax : <Variable> = CORRPOS_S[<Axe>] with variable type real

✎ In the permanent mode, the value returned by CORRECTION_S varies from « compensation waiting » to « compensation running » (| b0=1 b1=0 | b0=0 b1=1 | b0=1 b1=0 |...).

✎ In single-shot mode, it does: | b0=1 b1=0 | b0=0 b1=1 | b0=0 b1=0 |.

The error bit b2 is automatically reset after each reading of CORRECTION_S or at the beginning of the next correction.

d) Example :

```
.....
GEARBOX(Slave,Master,1,1,1)
STTI(Master=+)
.....                            ' Single-shot compensation
.....                            ' Compensation  condition: positive
.....                            ' edge on input C1 of
.....                            ' the slave axis
CORRECTION(Slave,Master,0,1,4,0,0,0,0,Phase,
          Interval!,0.5,0.5,Interval!/4)
REPEAT                           ' Wait end of compensation
                                 ' or compensation error
C#=CORRECTION_S
UNTIL (C#=0) OR (C#=4)
IF C#=0 THEN GOTO ERROR
.....
```

5-7-6- Movements superposition function(SRV 85 only)

The function ADDMOV superposes on a slave axis its own movements with those of a master axis. The link between the two axis must be an electric shaft (GEARBOX, GEARBOXM), synchronisation (MOVS, MOVSM, MOVSP, MOVSC) or cam (CAM, CAMC) or positioning (MOVA, MOVR, STTA, STTR, STTI).

Syntax : ADDMOV (<Slave>, <Master>, <Coefficient>)

<Slave> : Slave axis name where the superposition is applied (SRV85 only)

<Master> : Master axis name (servo board or encoder : SCD 22, SRV 15, SRV 85 ...)

<Coefficient> : used to scale the master and the slave, in pulses

```
Example  :
↳ Master axis : Linear axis, with ENCODER_P=4000, UNITREV_P=5mm
↳ Slave axis : Linear axis, with ENCODER_P=4000, UNITREV_P=10mm
↳ Coefficient =(ENCODER_P(Esc)*UNITREV_P(Master))/(ENCODER_P(Master)
               *UNITREV_P(Esc))
⇨ Coefficient=5/10=0.5
```

As soon as the instruction ADDMOV is done, the link between the two axis exists until the instruction ADDSTOP is executed.

Syntax : ADDSTOP (<SlaveAxis>)

If the slave axis becomes in non-controlled mode, the link between the master and the slave is broken too.

The master and the slave can do any movement: positioning, synchronisation, interpolation.

```
Example  :
.....
MOVA(X=0,Y=0)
ADDMOV(Y,X,1)         ' activation of the superposition link
STTI(X=+)             ' launch Axis X => Axis Y is moving too
WAIT INP(StartCycle)
MOVR(Y=StepCycle)     ' superposition of a movement on a distance StepCycle
.....
STOP(X)
ADDSTOP(Y)            ' stop the superposition link
.....
```

5-7-7- Stopping a master / slave link

To stop a link between a master and a slave, you have to use the instructions STOP or SSTOP on the slave axis. These instructions are stopping the specified axis too with the current deceleration, and empty the movement buffer.

The stop of the link can be executed whether the slave is moving or not.

The STOP instruction stops the execution of the task until the axis is stationary (condition MOVE_S(Axis)=0), while SSTOP does not.

Syntax : STOP (<Axis1> {,<Axis2>...})

Example :

```
GEARBOX (Slave, Master, 2, 1, 1)   ' Activation of the master / slave link
STTI(Master=+)                     ' Start of the master => the slave
                                   'follows
...
STOP(Master)                       ' Stop of the master => the slave stops
...
STOP(Slave)                        ' Stop of the master / slave link
```

The instruction AXIS(Slave)=Off stops too the link and the movement, but without any control, because the axis is disabled.

5-8- Interpolation

5-8-1- Linear interpolation

The function MOVL makes a linear interpolation between two axis.

Syntax : MOVL(<AxisX>=<DestinationX>, <AxisY>=<DestinationY>[,<Velocity>])

The parameters <DestinationX> and <DestinationY> are expressed in relative.

This instruction does not stop the task execution (except if the movement buffer is full)

If the instruction MERGE is activated and we send several MOVL instructions in the axis board, the movements will be done one after the other without passing through speed zero.

5-8-2- Circular interpolation

The function MOVC makes a circular interpolation between two axis. The circle and arc dimensions are defined with the four following parameters:

MOVC(<AxisX>=<DestinationX>, <AxisY>=<DestinationY>, <CenterX>, <CenterY>, <Clockwise> [,<Velocity>])

The parameters <DestinationX> and <DestinationY> are expressed in relative. The parameters <CenterX> and <CenterY> are defined regarding the current position. <Clockwise> defines the rotation direction.

This instruction does not stop the task execution (except if the movement buffer is full)

If the instruction MERGE is activated and we send several MOVC instructions in the axis board, the movements will be done one after the other without passing through speed zero.

5-8-3- Resultant acceleration / velocity

As a positioning, an interpolated movement is made of an acceleration phase, a constant speed, and a deceleration phase. The acceleration and deceleration phases are the same. These values correspond to the resultant acceleration and speed of the two axis.

The instruction ACC(<AxisX>, <AxisY>)=Value affects a resultant acceleration and deceleration. It is taken in account if the two axis are stopped.

The instruction VEL(<AxisX>, <AxisY>)=Value affects a resultant speed value. It is taken in account at any time.

This instruction does not stop the task execution (except if the movement buffer is full)

If the instruction MERGE (AxisX, AxisY) is activated and we send several MOVL and MOVC instructions in the axis board, the acceleration phase will be effective until the required resultant speed is reached. Then, the movements are chained, we go from one to the next at the same velocity. The deceleration phase will be done onto the last movement(s).

' Warning : For pieces which have some arcs of a circle and for which we use the linked movements (MERGE(X,Y)=On), the real acceleration may become very high:

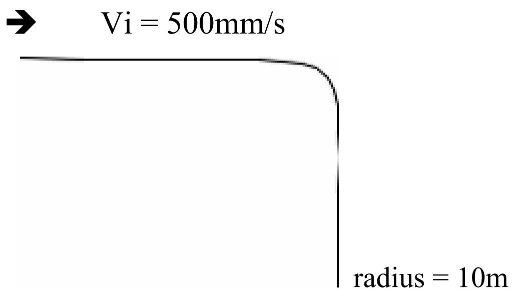
Accel (t) = $- (Vi^2 / r) * \cos (\text{Téta } i + ((Vi / r) * t))$

Accel. maxi = $- (Vi^2 / r)$

with Vi = initial velocity before the arc of circle in m / s

r = radius of the arc en m

Accel in m / s^2



Accel maxi = $- (0.5^2 / 0.01) = -25\ m/s^2$

5-8-4- Stopping a movement

To stop an interpolated movement during the motion, you have to use the instruction STOPI(<AxisX>, <AxisY>). The two axis stop following the interpolated trajectory, and using the deceleration given by the instruction ACC(<AxisX>,<AxisY>). The movement buffer is emptied: we will not be able to go on to the trajectory.

This instruction stops the task until the movement is over (condition MOVE_S(AxisX)=0 and MOVE_S(AxisY)=0). For example:

```
ACC(X,Y)=1000
VEL(X,Y)=100
MERGE(X,Y)=On
MOVL(X=10,Y=5)
MOVL(X=20,Y=0)
...
STOPI(X,Y)
```

If we wish to have a break during the movement, you have to set the resultant speed to zero. For example:

```
ACC(X  ,Y)=1000
VEL(X,Y)=100
MERGE(X,Y)=On
MOVL(X=10,Y=5)
MOVL(X=20,Y=0)
.....
...
IF FlagStop Then
    FlagStop=0
    VEL(X,Y)=0
END IF
IF FlagStart Then
    FlagStart=0
    VEL(X,Y)=100
END IF
...
```

5-8-5- Tools

From the tasks editor, it is possible to bring back a .DXF type file which will be translated in MOVL and MOVC instructions. The .DXF file must have only one looped profile.

An ISO interpret is also available as an option. It can deal with codes G00, G01, G02, G03, G04, G17, G18, G19, G40, G41, G42, G61, G62, G90, G91, G92, M01, P, Q , K, N, E, S.

Ask your retailer for more information.

5-8-6- Examples

A) Glue lifting

Program :

```
AXIS(X)=On
AXIS(Y)=On
VEL(X)=1000                 'Speed in positioning mode
VEL(Y)=1000
ACC(X)=1500                 'Acceleration and deceleration
DEC(X)=1500                 'in positioning mode
ACC(Y)=1500
DEC(Y)=1500
VEL(X,Y)=300                'Interpolated speed
ACC(X,Y)=2000               'Interpolated acceleration
MERGE(X,Y)=On               'Linked movements
```

```
...
MOVA(X = 20, Y = 10)        'Positioning to the firs point
OUT(Lifting)=ON                    'Activation
DELAY LiftingRun            'Temporisation
MOVL(X=110, Y=0)            'Interpolated movement to 130,10
MOVC(X=10, Y=10, 0, 10, 0)  'Move to 140,20 anticlockwise
MOVL(X=0, Y=60)             'Move to en 140,80
MOVC(X=-20, Y=0, -10, 0, 0) 'Move to 120,80 anticlockwise
MOVC(X=-10, Y=-10, -10, 0, 1)      'Move to 110,70 clockwise
MOVL(X=-90, Y=0)                   'Move to 20,70
MOVC(X=-10, Y=-10, 0, -10, 0)      'Move to 10,60 anticlockwise
MOVL(X=0, Y=-40)                   'Move to 10,20
MOVC(X=10, Y=-10, 10, 0, 0)        'Move to 20,10 anticlockwise
WAIT (MOVE_S(X)=Off) And (MOVE_S(Y)=Off) 'Wait for the end of movements
OUT(Lifting)=OFF            'Disable
DELAY  LiftingStop          'Temporisation
MOVA(X = 0, Y = 0)          'Back hone
```

In this example, the X and Y axis are used to lift a glue string. ‘Lifting’ is the output, which commands the glue lifting. After a movement from the origin to the first position (20,10), the lifting begins, following the drawing. When the movement links the starting point, the output is disabled and the axis are going back to the origin.

B) Engraving or cutting

Program :

```
AXIS(X)=On
AXIS(Y)=On
VEL(X)=1000          'Speed in positioning mode
VEL(Y)=1000
ACC(X)=1500          'Acceleration and deceleration in positioning mode
DEC(X)=1500
ACC(Y)=1500
DEC(Y)=1500
VEL(X,Y)=300         'Interpolated velocity
ACC(X,Y)=2000        'Interpolated Acceleration
MERGE(X,Y)=Off       'Movements non linked
...
MOVA(X = 0, Y = 10)   'Moving to the first point
OUT(Engraver)=ON      'Activation
DELAY EngraverOn      'Tempo
MOVL(X=0, Y=20)       'Interpolated movement to 0,20
MOVL(X=10, Y=-10)     'Move to 10,10 - M engraving
MOVL(X=10, Y=10)      'Move to 20,20
MOVL(X=0, Y=-20)      'Move to 20,0
WAIT (MOVE_S(X)=Off) And (MOVE_S(Y)=Off) 'Wait for the end of movements
OUT(Engraver)=OFF     'Stop the engraver
DELAY EngraverOff     'Tempo
MOVA(X=40, Y=0)       'Move to 40,0
OUT(Engraver)=ON      'Activation
DELAY EngraverOn      'Tempo
MOVL(X=-5, Y=0)       'Move to 35,0
MOVC(X=0, Y=20, 0, 10, 1)   'Move to 35,20 - C engraving
MOVL(X=5, Y=0)        'Move to 40,20
WAIT (MOVE_S(X)=Off) And (MOVE_S(Y)=Off) 'Wait for the end of movements
OUT(Engraver)=OFF     'Stop the engraver
DELAY EngraverOff     'Tempo
MOVA(X=60, Y=20)      'Move to 60,20
OUT(Engraver)=ON      'Activation
DELAY EngraverOn      'Tempo
MOVL(X=-10, Y=0)      'Move to 50,20
MOVC(X=0, Y=-10, 0, -5, 0)  'Move to 50,10
MOVL(X=10, Y=0)       'Move to 60,10 - S engraving
MOVC(X=0, Y-10, 0, -5, 1)          'Move to 60,0
```

```
MOVL(X=-10, Y=0)      'Move to 50,0
WAIT (MOVE_S(X)=Off) And (MOVE_S(Y)=Off) 'Wait for the end of movements
OUT(Engraver)=OFF     'Stop the engraver
DELAY EngraverOff     'Tempo
MOVA(X = 0, Y = 0)    'Back to the origin
```

5-9- Capture

5-9-1- Capture

A) CAPTURE (on servo board SRV 15 or SRV 85)

The CAPTURE instruction is used to record the current position of an axis. A specific input (also used for homing) is present on each servo and encoder board.

Syntax : CAPTURE(<Axis>,<Inversion>,<window>,<Mini>,<Maxi>,<Inside>)

With this instruction, the MCS is waiting for a positive edge on the capture input. When the edge is detected, the position is stored in the variable REGPOS_S. The flag REG_S is then set.

<Inversion> permits to deal with the negative edge instead of the positive one.

If <Window> is true then the input is tested only when <Axis> is between <Mini> and <Maxi> positions.

<Inside> tells if the test is made inside or outside of the limits.

<Mini> must be lower than <Maxi>.

```
For example:
CAPTURE(X,Off,On,10,20,On)   'Capture position on positive edge
                             'while X is between 10 and 20
WAIT REG_S(X)=True
MOVA(Y=500+REGPOS_S(X))
```

The captured position on the X axis is used to determinate the target position of the next movement on the Y axis.

B) CAPTURE1 (on servo board SRV 85)

The instruction CAPTURE1 is an extension of the instruction CAPTURE.

It permits to record the position of any servo board SRV 85 from an edge of 1 of the 2 capture inputs.

Syntax :

CAPTURE1(<Axis1>,<Axis2>,<Configuration1>,<Window2>,<Mini2>,<Maxi2>,
<Inside2>)

<Axis1> :	Axis where is located the capture input (SRV 85 only)
<Axis2> :	Axis on which we required to capture the position (SRV 85 only)
<Configuration1> :	Hardware configuration byte of <Axis1> :

b0 : non-used

b1 : capture on top Z

b2 : capture on input n°1 C1

b3 : capture on input n°2 C2

b4 : choice of the edge (0 : positive, 1 : negative)

b5..7 : non-used

<Window2> : activation of a window limited by the positions <Mini2> and <Maxi2> of <Axis2>, in which the capture input is treated.

<Inside2> tells if the test is made inside or outside of the limits.

<Mini2> must be lower than <Maxi2>.

With this instruction the MCS waits for an edge on the capture information. When the edge appears, the position is stored in the variable REGPOS1_S, and the flag REG1_S is set.

```
For example:
CAPTURE1(Y,X,4,On,10,20,On) ' Deportee capture of the position Axis X on  '
                            positive edge of the input C1 of axis Y ' while X
                            axis position is between 10 and 20
WAIT REG1_S(Y)=True
Correction!=Phase-REGPOS1_S(Y,X)
...
```

C) CAPTURE2 (on servo board SRV 85 only)

The CAPTURE2 instruction is equivalent as the CAPTURE one, but it also includes the choice of capture on C1, C2 or the encoder index Z.

Syntax :

CAPTURE2(<Axis>,<Configuration>,<Window>,<Mini>,<Maxi>,<Inside>)

<Axis> : Name of the Axis (SRV 85 only)

<Configuration> : Hardware configuration byte of <Axis>

b0 : non-used

b1 : capture on top Z

b2 : capture on input n°1 C1

b3 : capture on input n°2 C2

b4 : choice of the edge (0 : positive, 1 : negative)

b5..7 : non-used

If <Window> is true then the input is tested only when <Axis> is between <Mini> and <Maxi> positions.

<Inside> tells if the test is made inside or outside of the limits.

<Mini> must be lower than <Maxi>.

With this instruction the MCS waits for an edge on the capture information. When the edge appears, the position is stored in the variable REGPOS2_S, and the flag REG2_S is set.

```
For example :
CAPTURE2(X,24,Off,0,0,Off)       ' Capture position Axis X on negative
                                 ' edge of input C2 of Axis X
WAIT REG2_S(X)=True
Correction!=Phase-REGPOS2_S(X)
...
```

D) Further information

➯ The servo board SRV 15 contains only one register managed by the CAPTURE instruction.

➯ The servo board SRV 85 owns 2 registers: first managed by CAPTURE1, and second by CAPTURE2.

➯ In the case of a SRV 85, the instructions CAPTURE and CAPTURE2 are using the same internal register. It is better to use, on a same board, either CAPTURE or CAPTURE2, but not both together.

↳ An input C1 or C2 of a SRV 85 board can not manage at the same time counting functions (SETUPCOUNTER) and capture functions (CAPTURE, CAPTURE1 or CAPTURE2).

5-10- Time-based encoder

5-10-1- Time-based encoder

The system contains two time-based encoders, which can be used as a master axis in a synchronisation or cambox function. These encoders are called TIMEBASE1 and TIMEBASE2

The common functions to axis and time-based encoder are:

↳ Reading a position : POS_S

↳ Reset : CLEAR

↳ Stop : STOP

↳ Modulo definition : MODVAL_P

A specific function can start a time-based encoder: STARTTIME.

Syntax :

STARTTIME(TIMEBASE1) or STARTTIME(TIMEBASE2)

The time-based encoder position is incremented every 1.0005 ms if it is used on a cambox (ex : CAMBOX).

If a time-based encoder is the master of a synchronisation function, its position is incremented every 0.3335ms. (ex : MOVS, CAM).

These encoders are quite limited because we can not change the time basis. Else, you have to use another servo board declared in virtual mode (LOOP instruction).

5-11- Default on an axis

5-11-1- Default on an axis

As soon as an axis is declared in controlled mode, it is controlled at any time: while moving, stopping.

If the difference between the theoretical calculated position and the real one given by the encoder is bigger than the parameter ‘Maximum following error’, the system’s reaction is:

↳ Set the flag FEMAX_S

↳ All axis in non-controlled mode and open watchdog contact, except if the SECURITY instruction has been used.

↳ All the axis in non-controlled mode if the instruction SECURITY(On,Off) has been executed

↳ Open the watchdog relay if the instruction SECURITY(On,Off) has been executed..

↳ Forcing to zero of the analogue consign, and emptying of all the axis buffers in non-controlled mode.

↳ Reset of the flags MOVE_S of all the axis in non-controlled mode. If some tasks are going on sending movements to the axis, they will be consumed but not done.

↳ Beak of the master / slave links on the non-controlled axis.

The flag FEMAX_S will be set as soon as the axis is back in controlled mode (AXIS(Axis)=On).

' Warning : The instruction SECURITY is automatically forced to SECURITY(On,On) if all the tasks of the MCS are stopped.

5-12- Stepper axis

5-12-1- Stepper Axis

The mouvement commands are handled by the module only when if it's enabled : To do that use « AXIS(<Axe>)=On » instruction.

When the MCS 32EX carry on servo and stepper axes, if a following error appears on a servo axis the stepper axis will also do Axis(<Axe>)=Off. Then the fault must be handled, the axes must be enabled and then the cycle can be restarted.

The STP 85 module don't have a servo control, so the regulation, filter and following error parts are not handled.

Instructions supported by the STP 85 module :

- Axis control :

ACC, ACC%, ADDMOV, ADDSTOP, AXIS, AXIS_S, BUFMOV_S, CLEAR, CORRECTION, CORRECTION_S, DEC, DEC%, HOME, HOME_S, ICORRECTION, LIMMIN_S, LIMMAX_S, LIM_S, MERGE, MOVE_S, ORDER, ORDER_S, POS, POS_S, SENSOR, SENSOR1_S, SENSOR2_S, STOPCORRECTION, VEL, VEL%, VEL_S

- Positionning :

MOVA, MOVAC, MOVAP, MOVR, SSTOP, STOP, STTA, STTI, STTR, TRAJ

- Synchronisation :

CAM, CAMC, CAMNUM_S, CAMSEG_S, CAM_S, GEARBOX, GEARBOXRATIO, LOADCAMEX, LOADPOINT, LOADS, MOVC, MOVL, MOVS, MOVSP, STARTCAM, STARTS, STOPI

- Capture :

CAPTURE1, CAPTURE2, REGPOS1_S, REGPOS2_S, REG1_S, REG2_S

- Counters :

CLEARCOUNTER, COUNTER_S, SETUPCOUNTER

- Camboxes :

CAMBOX, CAMBOXDELAY, CAMBOXSEG, STARTCAMBOX, STOPCAMBOX

Aditionnal parameters list :

CLKINV_P, ENBINV_P, ILMINV_P, STEP_P, STPINV_P, THIGH_P, THOLD_P

Aditionnal instructions list :

ENABLE, ENABLE_S, ILIMIT, ILIMIT_S

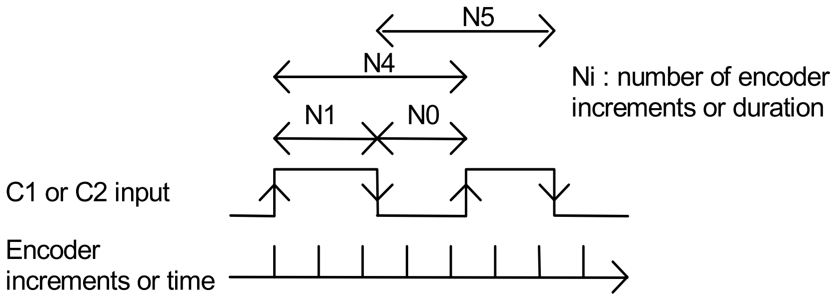
5-13- Enhanced Event Function

5-13-1- Definition

- Introduction

These packaging applications dedicated functions allow to trigger the event task with new conditions. Each SR85 axis card or SCD85 encoder card can generate two events depending on the state of the two capture inputs C1 and C2.

- Triggering conditions :


N5
N4
N1
N0
Ni : number of encoder increments or duration
C1 or C2 input
Encoder increments or time

The system count the number of increments really made between to edges of the selected sensor. Depending on configuration made by user, the SRV85 card will trigger the event task on one or two conditions.

The user can select the input number (C1 or C2), can invert it and filter it. It's also possible to handle the same kind of condition but by counting time between two edges instead of the number of encoder increments.

It's possible to wait for a maximum of 2 events from the same axis at the same time.

An event is generated with the following formula :

((MeasureA TestA LevelA) LogicalOperator (MeasureB TestB LevelB)) = ValidState

MeasureA : measure number

Measures definitions :

- Measure0: distance with the sensor OFF from the last falling edge
(distance between 0 and N0)
- Measure1: distance with the sensor ON from the last rising edge
(distance between 0 and N1)
- Measure2: distance from the last rising edge
(distance between 0 and N4)
- Measure3: distance from the last falling edge
(distance between 0 and N5)
- Measure4: Measure0 captured on the last rising edge
(N0 distance)
- Measure5: Measure1 captured on the last falling edge
(N1 distance)
- Measure6: distance between the two last rising edges
(N4 distance)
- Measure7: distance between the two last falling edges
(N5 distance)

TestA is a '>=' or '<' comparison between MeasureA et LevelA.

LevelA : Level number

The levels are expressed in encoder increments, times in 10µs. The maximal value is 2^31. The level number 0 to 3 are the levels used for the tests are the same for all events of the same axis.

Logical operator :

When the part B is defined (MeasureB, TestB, LevelB), the condition is formed with the combination with 'OR', 'AND' or 'XOR' of the two tests.

MeasureB, TestB, LevelB :

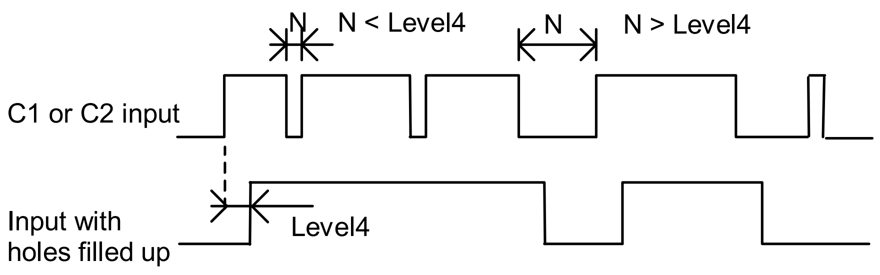
Same behaviour for the setting up as part A.

Valid state :

For each event we define with state is considered as valid and will be used to trigger the event : the valid state can be 0 or 1.

- Filling up holes between products

The filling up of the holes between products means that a high or low level on the sensor is taken into account only if it appears during a displacement greater than a value choose by the user. The level number 4 allow to choose this value.



- Measure of distance and time

A time or distance measure can be done on a sensor of each axis.

The value measured are :

- N0 : measure0 captured on the last rising edge
- N1 : measure1 captured on the last falling edge
- N4 : distance between the two last rising edges
- N5 : distance between the two last falling edges

The distances are expressed in encoder increments, times in 10µs. The maximal value is 2^31.

- Using Fifo

A Fifo memory allow to store distances measured between each sensor state changing. This allow to evaluate by teaching the distances to use to generate events.

These state changes are stored after the filling up functions.

5-13-2- Exemples

- Example 1

The C1 input of the X axis is used to trigger the event task when the sensor is off for a distance between 200 and 400.

```
‘*** Setting up task ***
‘ Events on X axis are activated and generated on a measure of the distance with C1 input non-inverted and not filtered
AXISEVENT(X,True,False,False,False,False)
' level #0= 200
AXISLEVEL(X, 0) = 200
' level #1= 400
AXISLEVEL(X, 1) = 400
' testA : measure #4 >= level #0
' testB : measure #4 < level #1
' logical operator = AND
AXISCONDITION(X, 1, 01011000001001b, 1)
‘Enable the event 1 on the first axis (X)
MODIFYAXISEVENT(0000000000000001b)



‘*** Event task ***
Prog

E%=GetAxisEvent
If E%.1 Then
  ‘Event 1 handling of the first axis
End If
End Prog
```

- Example 2

The C1 input of the X axis is used to trigger the event task when the sensor is off for a distance between 200 and 400. The C2 input is used to trigger the event task with a capture on the rising edge of this one.

```
‘*** Setting up task ***
‘ Events on X axis are activated and generated on a measure of the distance with C1 input non-inverted and not filtered
AXISEVENT(X,True,False,False,False,False)
' level #0= 200
AXISLEVEL(X, 0) = 200
' level #1= 400
AXISLEVEL(X, 1) = 400
' testA : measure #4 >= level #0
' testB : measure #4 < level #1
' logical operator = AND
AXISCONDITION(X, 1, 01011000001001b, 1)
‘Enable the event 1 on the first axis (X)
MODIFYAXISEVENT(0000000000000001b)
‘Enable the capture event on the first axis (X)
MODIFYEVENT (0000000100000000b,0)
‘Setting up the capture of the X axis position on the rising edge of the C2 input of the X axis
CAPTURE1(X,X,8,False,0,0,True)

‘*** Event task ***
Prog

E1%=GetEvent
E2%=GetAxisEvent
If E1%.9 Then
  ‘Capture event 1 handling of the first axis
 ‘ …
‘capture setting up for the next event
```

```
    CAPTURE1(X,X,8,False,0,0,True)
End If
If E2%.1 Then
  'Event 1 handling of the first axis
' …
End If

End Prog
```

6- PROGRAMMATION OF PLC

6-1- Basic task

6-1-1- Digital inputs/outputs

A) Inputs reading

The INP function is used to read 1 bit , INPB a 8 bits bloc and INPW a 16 bits bloc.

The syntax are : INP(<Digital inputs>), INPB(<Digital inputs>), INPW(<Digital inputs>).

<Digital inputs> must represent a valid digital input identifier of 1,8 or 16 bits. This identifier can be either a symbolic name used in the setup module or the hardware name of the bloc. The return data type is :

- Bit for 1 input bloc

- Byte for 8 inputs bloc

- Integer for 16 inputs bloc

For example :

```
A~ = INP(Sensor)      'input reading
B1# = INPB(Bloc1)     'First bloc of eight input reading
B2# = INPB(Bloc2)     'Second bloc of eight input reading
B3# = INPB(Bloc3)     'Third bloc of eight input reading
C%= INPW(A)           'Bloc of sixteen input reading
```

B) Outputs writing

The OUT function is used to write 1 bit , OUTB is used to write a 8 bits bloc and OUTW is used to write a 16 bits bloc .

The syntax are : OUT(<Digital outputs>), OUTB(<Digital outputs>), OUTW(<Digital outputs>)

<Digital outputs>must represent a valid digital output identifier of 1, 8, 16 bits. This identifier can be either a symbolic name used in the setup module or the hardware name of the bloc. The return data type is :

- Bit for 1 output bloc

- Byte for 8 inputs bloc

- Integer for 16 inputs bloc

For example :

```
OUT(Jack)=On          'Output writing
OUT(LAMP)=Defaut.5
OUTB(Data)=00110000b  'Bloc of eight inputs writing
OUTW(B)=0FFFFh        'Bloc of sixteen inputs writing
```

C) Outputs reading

All outputs can also be read. The reading value is the last written value. This feature is very useful when more than one program are using the same output bloc. So, it is possible to write only desired outputs in one operation without changing the others.

For example :

To put 1 on the fourth lower bit of a 8 bits output bloc named IO1, use the following program :

```
OUTB(Bloc1)=OUTB(Bloc1) OR 00001000b      'set of the fourth bit of a eight
                                          'inputs bloc
```

D) Events handling

We can wait for a state change on an input with the function WAIT.

The syntax is : WAIT <Condition>

The WAIT function is used to handle a special state condition during a normal execution.The execution is stopped as long as condition is false. When the state condition is true, the execution continues. This function is very useful to wait for end of movement or mechanical thrusts sensor...

Example :

```
WAIT Lim_S(Cutter)=On       'Waiting for a soft thrust error
Stop(Cutter)                'Axis stop
WAIT Inp(StartButton)=On    'waiting for StartButton pressed
```

E) State test

We can test the input state with the structure IF…THEN…ELSE.

The syntax is : IF (<Condition>) THEN <Action1> ELSE <Action2>

The IF…THEN…ELSE structure is used to test a condition at a given time. If <Condition> is true then the <Action1> is executed otherwise the <Action2> is executed.

Example :

```
IF (Inp(Start)=On) THEN     'Input state test
           Out(StartLed)=On
           RUN Cycle
ELSE
           Out(StartLed)=Off
           HALT Cycle
ENDIF
```

6-1-2- Analog inputs/outputs

A) Inputs reading

The ADC function is used to read an analogue input. Its syntax is : ADC(<AnaInput>)

<Analogue inputs> must represent a valid analogue input identifier.This identifier can be either a symbolic name used in the setup module or the hardware name of the bloc. The returned data by the function are always real type and between -10 and +10.

For example:

```
A! = ADC(Temperature)       'Analogue input reading
```

B) Outputs writing

The DAC function is used to write on an analogue output.

The syntax is : DAC(<Analogue outputs>)=<Real expression>

<Analogue outputs> must represent a valid analogue output identifier.This identifier can be either a symbolic name used in setup module or the hardware name of the bloc. The data used by the instruction are always real type and are between -10 and +10.

For example :

```
DAC(Consign)=5.0       'Write a command value of 5V
```

C) Outputs reading

All outputs can also be read. The reading value is the last written value. This feature is very useful when more than one programs are using the same output. So, you don't have to save current analogue output value.

For example :

```
DAC(Consign)=DAC(Consign)*2 'Multiply by 2 the output command
```

6-1-3- Timing

A) Passive waiting

The DELAY function is designed to make a passive waiting.

Its syntax is :DELAY <Duration>

<Duration> is a long integer expressed in millisecond.It is recommended using this function for a long passive waiting because the waiting program doesn't spend any processor time.

With this function, the program is waiting the indicated duration.

For example:

```
Debut:
WAIT Inp(Start)=ON
...
DELAY 5000            ' 5 seconds delay
...
GOTO Debut
```

B) Active waiting

6-1-4- TIME

The internal global variable TIME is designed to make active waiting of time. This variable is a long integer that represents the number of milliseconds passed since power-on. This variable can then be used as time base for machines which are powered on less than 24 days. At the power on, the variable is equal to zero. Up to 24 days, the variable is at its maximum value 2^31 and passed to its minimal value 2^-31. This overflow can make some timer errors. In that case, you must use the global variable TIMER.

For example :

```
EndDelay& = TIME+5000       'timer of 5s is loaded
WHILE TIME<EndDelay& DO
          ...               'Loop during 5s
END WHILE
EndTimeOut& = TIME+200
WAIT (Inp(Sensor)=On) Or (Time>EndTimeOut&)      'Waiting for sensor or
                                                 '200ms time-out
```

6-1-5- TIMER

The internal global variable TIMER is designed to make active waiting of time. This variable is a real that represents the number of milliseconds passed since power-on. This variable can then be used as time base for machines which are always powered on. The integer part of the global variable is the seconds and the decimal part (3 figures after the point) is the milliseconds.

Par example :

```
EndDelay! = TIMER+5.250     'timer of 5.25s is loaded
WHILE TIMER<EndDelay! DO
          ...               'Loop during 5.25s
END WHILE
EndTimeOut! = TIMER+0.200
WAIT (Inp(Sensor)=On) Or (TIMER>EndTimeOut&)     'Waiting for sensor or
                                                 '200ms time-out.
```

6-1-6- Events

A) Events

In a multi-tasking system, events mechanism are very useful for inter-process communication. Event handling may also provide process control functions. Event handling instruction allows sending, waiting and receiving events. Programs can wait for or sent the same event. In the programming language, there are two mechanisms for events functions.

6-1-7- Signal or Diffuse and Wait Event

➥ To send an event to only one task, there is the SIGNAL function. To send an event to all the tasks, there is the DIFFUSE function.

Syntax : SIGNAL <EventName> or DIFFUSE <EventName>

The <Eventname> can be any non-keyword name but must be used at least once in an event waiting or receiving function.

SIGNAL sends the event to the first task which is waiting it. But, DIFFUSE sends the event to all the tasks which are waiting it.

➥ The WAIT EVENT instruction is used to wait an event.

The syntax of the WAIT EVENT instruction is :

WAIT EVENT <EventName>

After WAIT EVENT instruction, program execution is paused and will be resumed when event is received.

```
Example  with SIGNAL and WAIT EVENT:
‘Master task                              ‘Slave task
PROG                                      PROG
...
RUN SlaveTask                             Beginning:
...                                       ...
WAIT Inp(StartCycle)=On                   ...
...                                       ...
SIGNAL Start                              WAIT EVENT Start
...                                       GOTO Beginning
...                                       END PROG
WAIT Inp(StopCycle)=On
HALT SlaveTask
...
END PROG
```

In this example, there is a master task that controls slave task execution. Master task is waiting for start button pressed state. When this state is reached, the master task starts slave task by sending start event. If stop button is pressed, master task handles this state and stops slave task. Slave task is idle and waiting for the start event. When this event is received, slave task executes a loop.

```
Example  with DIFFUSE and WAIT EVENT:
‘Master Task                              ‘Slave task
PROG                                      PROG
...
RUN SlaveTask                             Beginning:
...                                       ...
WAIT Inp(StartCycle)=On                   ...
...                                       ...
DIFFUSE Start                             WAIT EVENT Start
...                                       GOTO Beginning
...                                       END PROG
WAIT Inp(StopCycle)=On
HALT SlaveTask
...
END PROG
```

This example is the same like the last example but used the DIFFUSE instruction.

6-1-8- Wait

The second mechanism which waits an event is the WAIT instruction. This instruction doesn't allow the execution of the task if the expression is not valid. The Wait instruction used a global variable or an input. To send an event, you must assign a value to the global variable in another task.

The example below is the same like the SIGNAL and WAIT EVENT example but with the mechanism of WAIT :

```
'Master Task                             'Slave Task
WAIT Inp(StartCycle)=On                  ...
...                                      ...
SignalVariable=1                         WAIT SignalVariable=1
...                                      SignalVariable=0
```

This mechanism has an execution time longer than the other mechanism. The initialization of the global variable is an extra time in the execution.

6-1-9- Counters

A) Counters

The SRV85 boards has two 16 bits counters. Each inputs card can be assign to a counter.

' Warning :

- An input can only use a counter function or a position capture function.

- When the counter is at its maximum value, the counter is initialized to zero at the next edge.(maximum value : 65535)

6-1-10- Configuration

SETUPCOUNTER instruction allows the counter configuration.

Syntax : SETUPCOUNTER(<Axis>,<Input>,<Invert>,<DesactivateFilter>)

<Axis> : Servo card name

<Input> : Number of the input (1 for C1 input, 2 for C2 input)

<Invert> : Edge type : 0 for positive edge, 1 for negative edge

<DesactivateFilter> : 0 for a 2ms filter , 1 for without filter,.

If the filter isn't activating, the maximum frequency is 1.5 KHz. Else, the maximum frequency is 200 Hz.

6-1-11- Clear

CLEARCOUNTER instruction initializes the counter to zero.

Syntax : CLEARCOUNTER(<Axis>,<Input>)

<Axis> : Servo card name

<Input> : Number of the input (1 for C1 input, 2 for C2 input)

6-1-12- Read

COUNTER_S allows the reading of the counter.

Syntax : <Variable>=COUNTER_S(<Axis>,<Input>)

<Variable> : integer (0 to 65535)

<Axis> : Servo card name

<Input> : Number of the input (1 for C1 input, 2 for C2 input)

6-1-13- Complements

The state of each SRV85 card inputs (C1 or C2) can be read with the SENSOR1_S or SENSOR2_S instruction.

```
Example  :
OUT(Lamp1)=SENSOR1_S(X)
```

6-1-14- Cambox

A) Cambox

The cambox is used to control outputs according to angular, linear or temporal positions by using optimized instructions. MCB provides an enhanced cambox with 3 levels of complexity.

MCB accepts up to 8 camboxes with up to 16 segments for each cambox. Each cambox controls an output card (8 or 16 outputs) or an internal variable of type integer. For example, on a 16 outputs card, 3,4 and 5 outputs are used per cambox and other outputs are used in an other part of the machine.

If only one cambox is defined, then all the outputs of the cambox are refreshed with a 1ms cycle. (2 camboxes ⇨ 2ms refresh cycle,…., 8 camboxes ⇨ 8ms refresh cycle)

The available functions are :

CAMBOX, CAMBOXSEG, CAMBOXDELAY, CAMBOXVAR, STARTCAMBOX et STOPCAMBOX

When you define a segment, the starting value can be greater than the ending value.

Each segment definition is relative to the zero program.

Case of a cambox who active an internal variable :

An internal variable of integer type is free for each cambox. To use this variable, put 0 for the name of the card in the intruction Cambox : CAMBOX(1,X,0,8,0)

To read this variable in a program, use the function Camboxvar :

Syntax : <Interger Variable>=CAMBOXVAR(<NumCamBox>)

Ex : a%=CamboxVar(1)

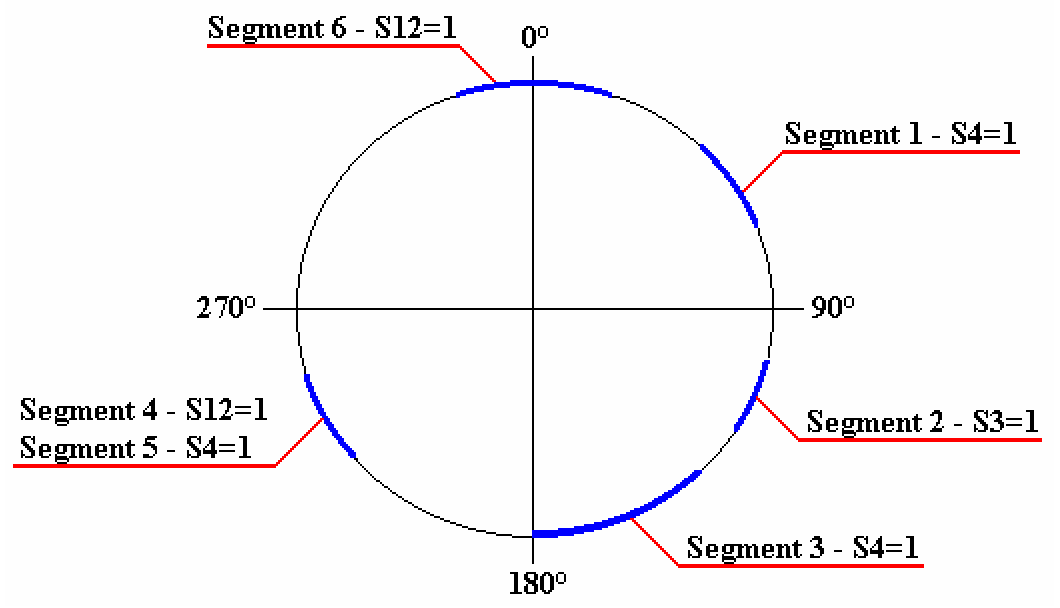
To modify the value of this variable in the program if the cambox is stopped :

Syntax : CAMBOXVAR(<NumCamBox>)=<Integer expression>

Ex : CAMBOXVAR(1)=0

If the box had alreday gone, the used bits by the box are not modifiable by the instruction CAMBOXVAR.

6-1-15- Simple cambox



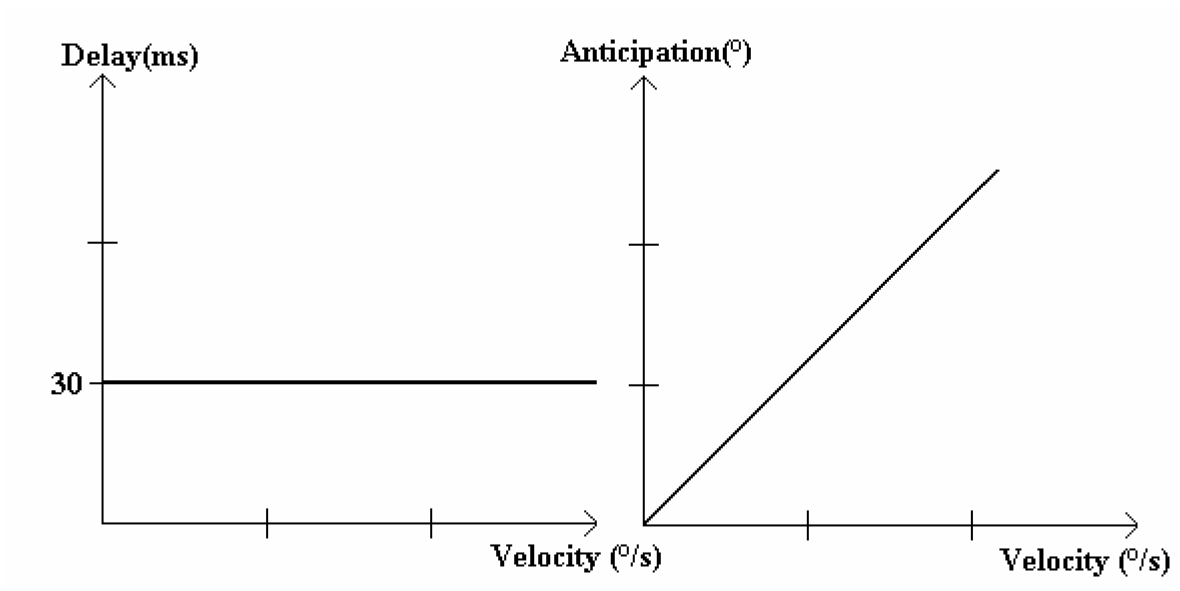
In this example, master axis is 360° modulo and output card name is A.The cambox is wrote like that :

```
CAMBOX (1,Master,A, 6, 0)          'The cambox number is 1 and
                                   'the number of segment is 6
CAMBOXSEG(1,1,4,20,60)             'The segment 1 set the output 4
                                   'between 20° and 60°
CAMBOXSEG(1,2,3,100,135)           'The segment 2 set the output 3
                                   'between 100° and 135°
CAMBOXSEG(1,3,4,135,180)           'The segment 3 set the output 4
                                   'between 135° and 180°
CAMBOXSEG(1,4,12,200,240)          'The segment 4 set the output
                                   '12 between 200° and 240°
CAMBOXSEG(1,5,4,200,240)           'The segment 5 set the output 4
                                   'between 200° and 240°
CAMBOXSEG(1,6,12,350,10)           'The segment 6 set the output 12
                                   'between 350° and 10°
STARTCAMBOX (1)                    'Start the cambox n°1
...
STOPCAMBOX (1)                     'Stop the cambox n°1
```

6-1-16- Compensation of mechanical behaviour

Cambox can also compensate mechanical behaviour. An anticipation on the position proportional to velocity is used to do this compensation.

In this example, a 30 ms anticipation is added for the pneumatic jacks connected on outputs 3,4 and 12 :

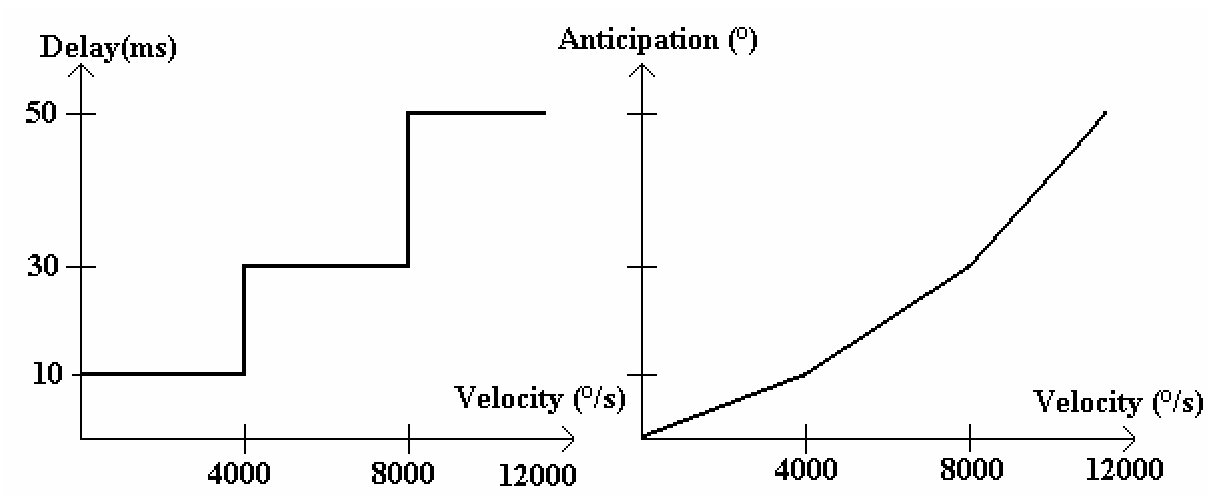


```
CAMBOX (1,Master,A, 6, 1)          'The cambox number is 1 and
                                   'the number of segments is 6
CAMBOXSEG(1,1,4,20,60)             'The segment 1 set the output 4
                                   'between 20° and 60°
CAMBOXSEG(1,2,3,100,135)           'The segment 2 set the output 3
                                   'between 100° and 135°
CAMBOXSEG(1,3,4,135,180)           'The segment 3 set the output 4
                                   'between 135° and 180°
CAMBOXSEG(1,4,12,200,240)          'The segment 4 set the output 12
                                   'between 200° and 240°
CAMBOXSEG(1,5,4,200,240)           'The segment 5 set the output 4
                                   'between 200° and 240°
CAMBOXSEG(1,6,12,350,10)           'The segment 6 set the output 12
                                   'between 350° and 10°
CAMBOXDELAY(1,1,0,30)              'A 30ms anticipation is added
STARTCAMBOX (1)                    'Start the cambox n°1
...
STOPCAMBOX (1)                     'Stop the cambox n°1
```

6-1-17- Compensation of mechanical and products behaviour

On special applications it is possible to use more than 1 anticipation time to compensate products behaviour.

In the last example, we want to add a 10 ms anticipation for velocity up to 4000° / s, 30 ms for velocity from 4000 to 8000° / s and 50ms for velocity from 8000 to 12000° / s and above.



```
CAMBOX (1,Master,A, 6, 3)          'The cambox number is 1 and
                                   'the number of segments is 6
CAMBOXSEG(1,1,4,20,60)             'The segment 1 set the output 4
                                   'between 20° and 60°
CAMBOXSEG(1,2,3,100,135)           'The segment 2 set the output 3
                                   'between 100° and 135°
CAMBOXSEG(1,3,4,135,180)           'The segment 3 set the output 4
                                   'between 135° and 180°
CAMBOXSEG(1,4,12,200,240)          'The segment 4 set the output 12
                                   'between 200° and 240°
CAMBOXSEG(1,5,4,200,240)           'The segment 5 set the output 4
                                   'between 200° and 240°
CAMBOXSEG(1,6,12,350,10)           'The segment 6 set the output 12
                                   'between 350° and 10°
CAMBOXDELAY(1,1,4000,10)           'A 10ms anticipation is added up to
                                   '4000° / s
CAMBOXDELAY(1,2,8000,30)           'A 30ms anticipation is added up to
                                   '8000° / s
CAMBOXDELAY(1,3,12000,50)          'A 50ms anticipation is added up to
                                   '12000° / s
STARTCAMBOX (1)                    'Start the cambox n°1
...
STOPCAMBOX (1)                     'Stop the cambox n°1
```

6-1-18- Enhanced PLC Functions

A) Enhanced PLC Function

a) Présentation

The PLC functions (Enhanced PLC) allow to integrate the functioning of a PLC in a multitasks basic program. Like this, we warrant that the I/O used in this tasks are handle as a PLC. The inputs are memorised in bit's copy before to be treated, the ouputs to modify are memorised before to be update.

b) Utilisation du PLC

The PLC use tables to memorize the status of I/O. Two tables of long integer for inputs and two tables of integer for outputs.

The function PlcReadInputs read the status of inputs, after to have memorized theirs old status, to allow detection of edge.

The function PlcInp, PlcInpb, PlcInpw, PlcInpPe ans PlcInpNe allow to read the status of inputs and detect edges.

The functions PlcOut, PlcOutB and PlcOutW modify bit's copy of outputs.

The function PlcWriteOutputs write the status of bit's copy on physical outputs.

c) Exemple

In this exemple, the outputs's blocks are used to count positive and negative edge of a input.

```
PROG
‘ on utilise toutes les sorties
 Masque[1]=0FFFFh
 Masque[2]=0FFFFh
‘ on initialise le PLC
 PlcInit(Entrees,EntreesOld,Sorties,Masque)
 Repeat
  ‘ lecture des entrées
   PlcReadInputs
  ‘ détection des fronts montants
   If PlcInpPe(I1) Then
     PlcOutB(JL)=PlcOutB(JL)+1

   End If
 ‘ détection des fronts descendants
   If PlcInpNe(I1) Then
     PlcOutB(JH)=PlcOutB(JH)+1
   End If
‘ écriture des sorties
PlcWriteOutputs
 Until False
END PROG
```

6-2- Ladder task

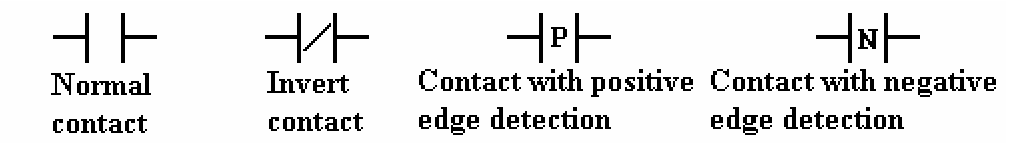
6-2-1- Presentation

Each ladder task is defined with rungs. The number of rungs is limited to 50 for a task. A rung is defined by one or more coils and only one expression. Then, coils of a same rung have the same expression. A rung can have a maximum of 5 coils or contact in parallel and 10 contact in serial.

Attention : A ladder task is automatically traduct in basic. It's advise to not write a long or complex ladder task, in order to avoid time cycle detoriorations and basic traduction limit.

6-2-2- Contacts, coils, timers and counters

6-2-3- Contacts


Normal contact
Invert contact
Contact with positive edge detection
Contact with negative edge detection

An input name, output name or bit name can be assigned to a contact. A system bit name can be assigned only to a normal or invert contact.

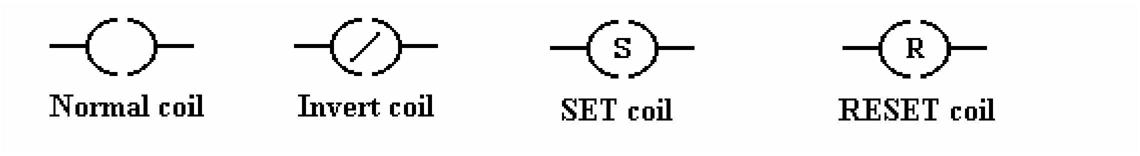
➪ Normal contact : The state of the contact is the state of the variable assigned.

➪ Invert contact : The state of the contact is the invert state of the variable assigned.

➪ Contact with positive edge detection : The state of the contact is true when the assigned variable is in the transition state : false to true.

➪ Contact with negative edge detection: The state of the contact is true when the assigned variable is in the transition state : true to false.

6-2-4- Coils



An output name or a bit name can be assigned to a coil. An input name or a system bit can't be assigned to it.

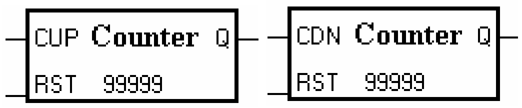
➪ Normal coil : The state of the coil is the state of the expression assigned.

➪ Invert coil : The state of the coil is the invert state of the expression assigned.

➪ Coil with SET action : The state of the coil is true when the expression is true. The state of the coil is false when the Reset coil is activated.

➪ Coil with RESET action: The state of the coil is false when the expression is false. The state of the coil is true when the Set coil is activated.

6-2-5- Counters up or down



The counters up or down have two inputs and an output. Each counters up or down is defined with a name and a pre-selected value. This pre-selected value may be a fixed value or a global variable. With a global variable, you can modify it at any time during the execution. When you used a counter up or down, you must link the counter output to a coil even if the coil is not used.

➪ Up counter : CUP is the counter up input. On a positive edge detected on this input, the counter up variable is incremented. When the value of the counter up variable is greater or equal to the pre-selected value, the counter up output is true. The RST input has priority. When this input is true, the counter up variable is initialize to zero. At the power on, the counter up value is equal to zero.

➪ Down counter : CDN is the counter down input. On a negative edge detected on this input, the counter down variable is decrements. When the value of the counter down variable is lower or equal to zero, the counter down output is true. The RST input has priority. When this input is true, the counter down variable is initialize to the pre-selected value. At the power on, the counter up value is equal to the pre-selected value.

The counter up or counter down variable can be treated and modify in another basic task : <CounterName> + <&>.

```
Example  : Counter name   : Counter1
           Counter up Local Variable  : Counter1&
```

6-2-6- Timer

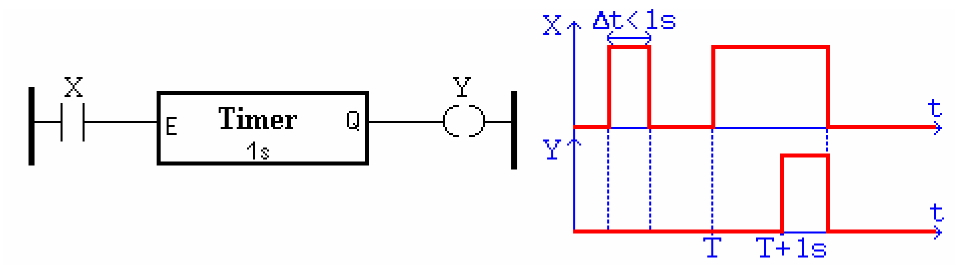
The timers are all on delay timing (TON). The delay may be a fixed value or a global variable. All the timers uses the TIMER instruction. When you used a timer, you must link the timer output to a coil even if the coil is not used.

The variable used with the timer can be treated in another basic task. Its syntax is : <Bloc Name> + <TVAL!>. This variable represents the remaining delay since the activation of timer.

```
Example  : Timer name   : Timer1
           Variable : Timer1Val!
```

➭ On delay timing (TON) :

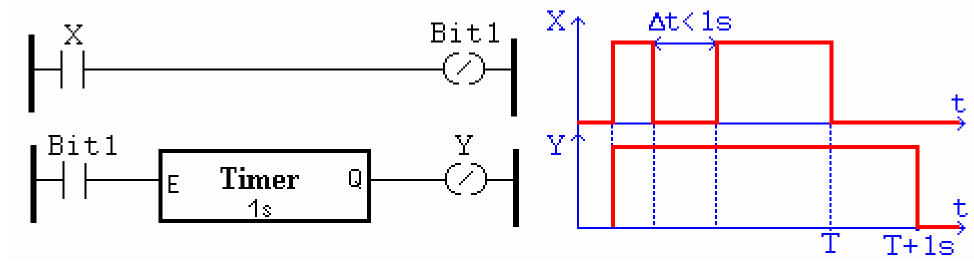
Example :



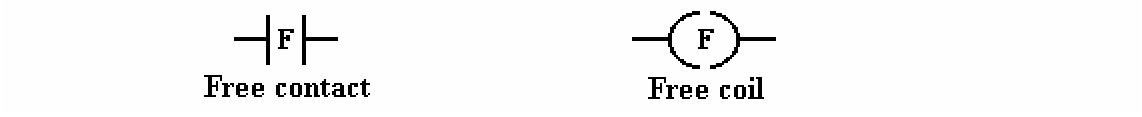
➭ Off delay timing (TOFF) :

To make this type of timer, you must use an invert coil with the triggering expression in a first rung. In the second rung, you use the contact with the same variable as the coil in the last rung and a timer and another invert coil.

Example :



6-2-7- Free contact and coil



This type of contact and coil provides more capabilities for the ladder. This type of contact and coil are : the free contact and the free coil.

➭ Free contact : With this type of contact, you can test all type of variables (Ex :byte, Integer, long integer, real or string). We can make test with movement instructions (Ex :

MOVE_S(X),…). In this contact, you must only edit your expression to test with the bracket. (Ex : (MOVE_S(X)=1) And (POS(X)>2000))

➷ Free coil : This type of coil allows you to execute any sort of instruction like movement instruction... With this coil, you can assign all types of variables. (Ex :byte, Integer, long integer, real or string). In this coil, you must only edit the instruction. (Ex : STTA(X=100,Y=150))

' Warning : Don 't use passive wait instruction. This type of instructions stops anf affects the ladder task evolution (Ex : MOVA, WAIT,…).

6-2-8- System bits

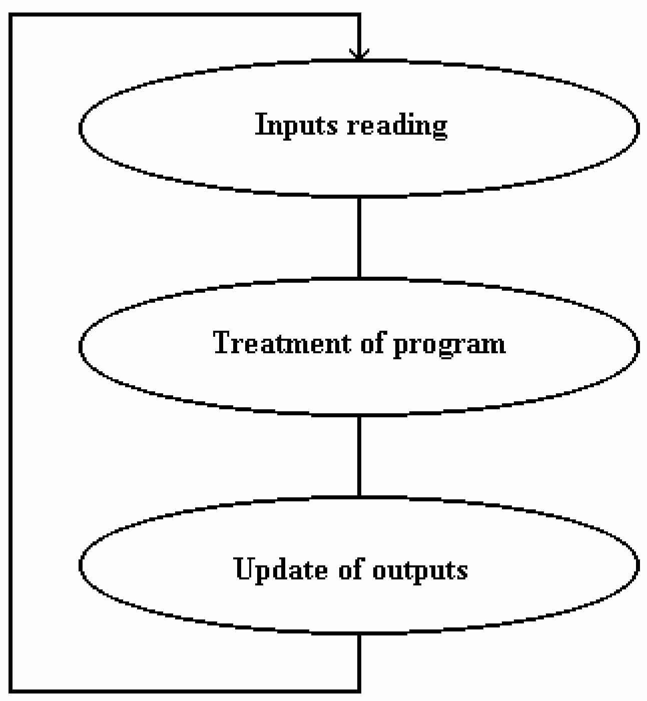
➷ Initialization bit : This bit is true on the first cycle of the ladder task.

➷ Blink 0.5s : The state of this system bit changes between 0 and 1 every 0.5s.

➷ Blink 1s : The state of this system bit changes between 0 and 1 every 1s.

6-2-9- Task architecture

This is the architecture of the ladder task :



With this architecture, the state of inputs is loaded before the treatment of equations. The outputs are updated only once times per cycle.

The multitask allows the ladder task to be suspended at any times of the execution cycle.

7- PROGRAMMATION OF SERIAL1 / SERIAL2 COMUNICATION PORTS

7-1- Introduction

MCS have an RS232 communication port on Serial1. This communication port is used to send or receive the configuration, variables, tasks… between PC and MCS.

A second optional serial port RS232 or RS485 can be installed on Serial2.

These 2 ports can be treated in basic tasks. With the port, we can open or close it or reading or writing data.

The function of conversion like MKI$, CVI, MKL$, CVL... can be used to optimize the coded and decoded of message.

7-2- Opening a communication port

To open a communication port Motion Control Basic provides the OPEN instruction. OPEN instruction has the following syntax :

OPEN <Communication port> AS # <CommNumber>

<Communication port> is a string that identifies physical communication port name and setup. <CommNumber> is the number used to identify the opened communication port . This number will be used by READ, WRITE and CLOSE functions.

<Communication port>string can be decomposed in five parts :

"SERIAL2:[Speed[, Data[, Parity [, Stop]]]]"

➪ Speed : Communication speed (150, 300, 600, 1200, 2400, 4800 or 9600 b/s)

➪ Data : Number of data bits (7 or 8)

➪ Parity : Parity checking mode (E for Event, O for Odd, M for Mask, S for Space or N for None)

➪ Stop : Number of stop bits (1 or 2)

The string must respect the parameters order. Speed, data, parity and stop parameters are optionals. When the task is compiled and if the parameters are not defined, the system takes the default parameter defined in the configuration screen. (double-click on the SUBD of the Serial communication port).

Example :

```
OPEN «SERIAL2  :9600,8,N,1" AS #1  ' SERIAL2 is opened to communicate
                                   ' with a Dialog 640 operator panel
```

When a port is open by a task, this port can't be opened again by another task. But a communication port open can be read or written by any other task.

A communication port must be open before the reading or writing of data.

You should reserve the Serial1 to the downloading between MCS and PC. Otherwise, you need to manipulate the plugs.

If Serial1 is used in a task, you should execute the « Stop task » command before the downloading.

7-3- Reading data

⇨ Received buffer

Each serial port have a received buffer with 500 bytes length

If the buffer is full (500 characters are received and unread), the new received characters clear the first one.

The CLEARIN instruction clears this buffer.

The CARIN instruction returns the number of characters in the buffer.

To read data, there are two instructions : INPUT and INPUT$.

The INPUT instruction waits the data and assigns the received data to variables.

The syntax is : INPUT #<ComNumber>, <Variable> [{, <Variable> }]

<ComNumber> is the number specified in the OPEN instruction. The reading data must come in the order of the variable list and with the same type.

For example :

```
OPEN "SERIAL1:"  AS #1       ' Open the serial port 1 affected to canal 1
INPUT #1, B$, C%             ' Read a string and an integer
CLOSE #1                     ' Close the serial port
```

For all the numeric variables of the list, the beginning of the number is detected when the first character is not a space character. The end of the character is detected with a space, a comma or a carriage return character. An underscore character is a zero. If the numeric variable is not valid, the variable takes the zero value.

For all the string variables of the list, the beginning of the string is detected when the first character is not a space character. The end of the character is detected with a space, a comma or a carriage return character. An underscore character is a string with a string length equal to zero.

The INPUT$ instruction reads some characters on the communication port and stores them in a string char. The syntax is :

<StringcharVariable> = INPUT$ (#<ComNumber>, <LengthOfCharacters>)

This two instructions stop the task as long as the number of received characters is not valid.

7-4- Writing data

⇨ Transmit buffer

Each serial port have a transmit buffer with 500 bytes length

The characters, which are sent by a task with the PRINT instruction, are send to the transmit buffer. These characters are transmitted one after one on the serial link.

If the transmit buffer is full (500 characters in the buffer), the task, which wants to send data, is suspended as long as the transmit buffer is full.

The CLEAROUT instruction clears the buffer.

The CAROUT instruction returns the number of characters in the transmit buffer.

The OUTEMPTY instruction indicates if the buffer is empty and the last character is sent.

The PRINT instruction converts data and send them. The syntax is :

PRINT #<ComNumber>, <Expression> [{ [; | ,] <Expression> }] [; | ,]

<ComNumber> is the number specified in the OPEN instruction.

For example :

```
OPEN "SERIAL1:" AS #1        ' Open the communication port 1
...
PRINT #1, A$, B%;            ' Send a string of char and an integer
PRINT #1, C$,
PRINT #1,CHR$(10)  ;MESSAGE1$  ;   ' ASCII 13D is not sent after MESSAGE1$
PRINT #1,CHR$(10)  ;MESSAGE2$      ' ASCII 13D is sent after MESSAGE2$
...
```

A semicolon between two expressions signifies that the next character is sent immediately after the last character. A semicolon at an end line signifies that the extra ASCII character 13D is not sent.

A comma signifies that the character is sent at the beginning of the next line. If there is no expressions list after the PRINT expression, the ASCII Character 13D is sent.

If the parameter #1 or #2 is not specified, the system send the data on #1.

7-5- Close a communication port

To close a communication port, there is the CLOSE instruction. The syntax is :

CLOSE #<CommNumber>

7-6- RS485 treatment

With a RS232 communication port, MCS can communicate with only one peripheral system. But, with the RS485 communication port, MCS can communicate with more than one peripheral system.

To send a message with a RS485 communication port, MCS must drive the communication line.

The TX485 instruction permits MCS to take the line during a given number of character. When a character is sent, the TX485 value is decrements. When this value reaches zero, the line is automatically given back.

' Warning : Each character sent is received by the MCS as the TX485 value is different to zero.

Example :

```
.....
Message$= « Motion Control System »
TX485(#1)=Len(Message$)
PRINT #1,Message$  ;         ' Take the line during the sending of Message$
CLEARIN #1                   ' Clear the echo characters
```

7-7- Example: RTU Modbus driver

```
SLAVE232 Task
     Prog
      ' ***
      ' *** DRIVER MODBUS ESCLAVE RS232 ***
      ' ***
      '------------------------------------------------------------
      ' *
      ' * INITIALISATION *
      ' *
      '
      ' WARNING!!! =>Defined in global stored variables :
      ' TableModbus   type:integer    number:255
      '
      NumeroMcs#=1 'number of the mcs32
      TimeOut&=10  '10ms maximum delay between 2 received characters
      '
      AdressModBus%=600  'Start address of the table
      NumberModbus%=300   'Number of words in the table
      ' init maintenance counters
```

```
 CmtMessage&=0
 ErrLiaison&=0
 ErrAdresse&=0
 ErrData&=0
 'Open serial2
 Open "Serial2:9600,8,N,1" As #2
 Clearin #2 'Clear the rxd buffer
 TempoRxd&=Time
 '-------------------------------------------------------------
 ' *
 ' * RECEIVE *
 ' *
InitRxd:
 PtrRxd#=0
 Rxd$=""
 '
WaitRxd:
 If Carin(#2)<>0 Then Jump ReadRxd
 If PtrRxd#=0 Then Goto WaitRxd
 If Time>TempoRxd& Then Goto InitRxd
 Goto WaitRxd
 '
ReadRxd:
 TempoRxd&=Time+TimeOut&
 If PtrRxd#>=2 Then Jump MessageRxd
 If PtrRxd#=1 Then Jump Car2Rxd
Car1Rxd:
 CarRxd$=Input$ #2,1
Car1tRxd:
 NumMcs#=Asc(CarRxd$)
 If (NumMcs#<>NumeroMcs#) And (NumMcs#<>0) Then Jump InitRxd
 PtrRxd#=1
 Rxd$=CarRxd$
 Jump WaitRxd
Car2Rxd:
 CarRxd$=Input$ #2,1
 NumFonction#=Asc(CarRxd$)
 If  (NumFonction#<>3)  And  (NumFonction#<>4)  And  (NumFonction#<>16)  Then  Jump
Car1tRxd
 PtrRxd#=2
 Rxd$=Rxd$+CarRxd$
 Jump WaitRxd
MessageRxd:
 CarRxd$=Input$ #2,Carin(#2)
 PtrRxd#=PtrRxd#+len(CarRxd$)
 If PtrRxd#>240 Then Jump InitRxd
 Rxd$=Rxd$+CarRxd$
 If NumFonction#=16 Then
    If PtrRxd#<7 Then Jump WaitRxd
    If PtrRxd#<(Asc(Rxd$,7)+9) Then Jump WaitRxd
    Rxd$=Left$(Rxd$,Asc(Rxd$,7)+9)
   Else
    If PtrRxd#<8 Then Jump WaitRxd
    Rxd$=Left$(Rxd$,8)
 End If
 '
TraitementMessage:
 Sum$=Left$(Rxd$,Len(Rxd$)-2)
 Sum%=Crc(Sum$)
 Sum$=Mki$(Sum%)
 If Sum$<>Right$(Rxd$,2) Then Jump ErreurLiaison
 AdrBus%=Cvir(Mid$(Rxd$,3,2))
 NbrBus#=Asc(Rxd$,6)
 If (NbrBus#=0) Or (NbrBus#>100) Then Jump ErreurAdresse
 If AdrBus%<AdresseModbus% Then Jump ErreurAdresse
 A1%=AdrBus%+NbBus#
 A2%=AdresseModbus%+NombreModbus%
 If A1%>A2% Then Jump ErreurAdresse
 If NumFonction#=16 Then Jump WriteWord
 '------------------------------------------------------------
 '
 ' reading words
 '
ReadWord:
 If NumMcs#<>NumeroMcs# Then Jump ErreurLiaison
 Txd$=""
 I#=1
 A%=(AdrBus%-AdresseModbus%)+1
ReadWordBcl:
 Txd$=Txd$+Mkir$(TableModbus[A%])
 A%=A%+1
 I#=I#+1
 If I#<=NbrBus# Then Jump ReadWordBcl
 Txd$=Chr$(Len(Txd$))+Txd$
 CmtMessage&=CmtMessage&+1
 Jump MessageTxd
 '------------------------------------------------------------
 '
 ' Write Words
 '
WriteWord:
 I#=1
 J#=0
 A%=(AdrBus%-AdresseModbus%)+1
WriteWordBcl:
 TableModbus[A%]=Cvir(Mid$(Rxd$,8+J#,2))
 A%=A%+1
 I#=I#+1
 J#=J#+2
 If I#<=NbrBus# Then Jump WriteWordBcl
 Txd$=Mid$(Rxd$,3,4)
 CmtMessage&=CmtMessage&+1
 Jump MessageTxd
 '------------------------------------------------------------
 ' *
 ' * TRANSMIT *
 ' *
 ' Erreurs
ErreurLiaison:
 ErrLiaison&=ErrLiaison&+1
 Jump InitRxd
ErreurAdresse:
 NumFonction#=NumFonction#+128
 Txd$=Chr$(2)
 ErrAdresse&=ErrAdresse&+1
 Jump MessageTxd
ErreurData:
 NumFonction#=NumFonction#+128
 Txd$=Chr$(3)
 ErrData&=ErrData&+1
 ' Send message
MessageTxd:
 Clearin #2 'clear rxd buffer
 If NumMcs#=0 Then Jump InitRxd
 Txd$=Chr$(NumMcs#)+Chr$(NumFonction#)+Txd$
 Sum%=Crc(Txd$)
 Print #2,Txd$+Mki$(Sum%);
 Jump InitRxd
 '
End Prog
```

8- PROGRAMMATION OF TERMINAL OPERATOR

8-1- Dialog 80 description

Screen

- 4×20 Characters LCD display with backlight
- Display area 74×23 mm
- Characters attributes : normal, blinking
- ASCII protocol

Keyboard

- 28 keys with tactile feedback
- 4 dynamic function keys
- 6 rewriteable function keys with integrated leds
- Control and scrolling keys
- Help and alarm keys
- Numeric and alphanumeric keypad
- Buzzer

Performances

- 16 bits Processor
- 512Kbyte flash memory
- 128Kbyte non-volatile RAM
- RS232 serial communication port
- Optional RS422 or RS485 serial communication port
- Optional fieldbus : CANBUS

Technical features

- 24Vdc supply voltage
- 4W power requirement
- 0 to 45°C operating temperature
- –20 to 70°C storage temperature
- Front panel protection IP65

8-2- Dialog 640 description

Screen

- LCD display with CFL backlight
- Display area 122×66 mm
- Characters attributes : normal, reverse, blinking
- ASCII protocol
- Resolution in graphic mode : 240×128 pixels

- 4 simultaneous sizes of characters in text mode :

⇨ 3×4 mm	16 lines × 40 characters
⇨ 4×7 mm	9 lines × 30 characters
⇨ 5×8 mm	8 lines × 26 characters
⇨ 7×10 mm	6 lines × 17 characters

Keypad

- 33 keys with tactile feedback
- 6 dynamic function keys
- 6 rewriteable function keys with integrated leds
- Control and scrolling keys
- Help and alarm keys
- Numeric and alphanumeric keys
- Buzzer

Performances

- 16 bits processor
- 512Kbyte flash memory
- 128Kbyte non-volatile RAM
- RS232 serial communication port
- Optional RS422 or RS485 serial communication port
- Optional fielbus : CANBUS

Technical features

- 24Vdc supply voltage
- 6W power requirement
- 0 to 45°C operating temperature
- –20 to 70°C storage temperature
- Front panel protection IP65

8-3- Operator functions

8-3-1- Opening a communication

All specific functions to the operator panel use the communication port declared on #1. However if the physical communication port used is "SERIAL2", it is necessary to re-affect the communication port #1 by using the OPEN function or the « Opening communication » command in the toolbox of the basic task editor.

```
Open "Serial2:9600,8,N,1" As #1    ' Operator panel connected to SERIAL2
or
Open "Serial1:9600,8,N,1" As #1    ' Operator panel connected to SERIAL1
```

8-3-2- Screen

Four functions give the access of the operator panel screen.

➷ The CLS function clears partially (Dialog80 and Dialog160 only) or the totality of the screen. To clear a line, the number of the line must be specified. If no number is indicated, the totality of the screen is cleared.

The syntax of this function is : CLS [<LineNumber>].

This instruction have specific extensions for the Dialog 640 :

⇨ CLS B : clear the screen with a black background

⇨ CLS W : clear the screen with a white background

➷ To display the cursor, you need to use the CURSOR (on/off) instruction. This function indicates the beginning of a data capture. CURSOR=<ON/OFF>

➷ To locate the cursor on the screen, there is the LOCATE instruction. The origin of the screen is in the top and left corner of the screen with the 1,1 coordinates. The syntax is : LOCATE <Line>,<Row>.

➷ The PRINT function displays a text or the contents of a variableon the screen. The syntax is : PRINT <Expression>[;/,]<Expression>[;/,]

When a comma is used to separate two expressions, a carriage return is inserted between these expressions. But if a semi-column is used to separate two expressions, the carriage return is not inserted. (character ASCII 13(D))

Example :

```
CLS         'Clear screen
CURSOR=ON   'Display the cursor
LOCATE 2,4  'locate the cursor at the line 2 and row 4
```

For the dialog 640, there are 5 others type of functions :

➷ The FONT function defines the font to use for the text. The syntax is : FONT=<Value>. <Value> is an integer between 1and 8 and defines the type of font.

➷ The PIXEL function switch on or off a pixel on the screen. The syntax is : PIXEL(X,Y,Colour). Colour may be the white (Colour=1) or black (Colour=0).

➷ The BOX function draws a rectangle on the screen. The syntax is :

BOX(X1,Y1,X2,Y2,<BorderColour>,<FillColour>). The X1, Y1 parameters are the top and left corner of the rectangle and X2, Y2 parameters the bottom and right corner of the rectangle. <BorderColor> defines the color of the border and <FillColour> the colour of the filling rectangle.

➷ The HLINE function draws a horizontal line on the screen. The syntax is : HLINE(X1,Y1,X2,<Colour>). The X1,Y1 parameters are the starting point of the line and X2,Y1 parameters are the ending point. <Colour> defines the colour of the line.

➷ The VLINE function draws a vertical line on the screen. The syntax is : VLINE(X1,Y1,Y2,<Color>). The X1,Y1 parameters are the starting point of the line and X1,Y2 parameters the ending point. <Colour> defines the colour of the line.

8-3-3- Keyboard

We have two functions and a system variable to use the keyboard.

➷ The function Inkey allows to read a key and to stock its code in a type byte variable. If no key is pressed before the function call, this one returns 0.

The syntax is the following : <Variable>=INKEY

Example :

```
Waiting:
K#=INKEY
```

```
IF K#=0 Then Goto Waiting
IF K#=@F1 Then Goto MenuF1
IF Inp(StartButton)=On Then Goto Start
Goto Waiting
```

➯ The WAIT KEY function allows to wait for pressing a key and stocking then the key code in the system variable KEY. Contrary to the previous function, this function is locking the task as long as any other key is pressed. The syntax is : WAIT KEY

➯ Endly, the system variable KEY contains the code of the last key pressed in the functions WAIT KEY or EDIT. This variable is local to the task and can't be written.

Example :

```
WAIT  KEY                   ' Key waiting
IF KEY=@F1 THEN GOTO ...
IF KEY=@F2 THEN GOTO ...
...
```

8-3-4- Edit

The DIALOG 80, 160 and 640 allows, via the EDIT function, to type a real with or without sign and point, displaying it on an exact place on the screen.In the instruction's line, we choose the number of characters in the real variable, the line and row number of the first character. we can as well say if yes or no (0 or 1) we use the sign and/or the point.

The syntax is : <Variable> = EDIT(<Line>,<Row>,<Length>,<Sign>, <Point>).

To edit the value on the DIALOG 80, 160 and 640, we use the numerical keys, the DEL keys to clear, ENTER to valid and ESC to stop the editing.

Example :

```
EditRes!=EDIT(1,5,4,0,0)    'Real four numbers edition without point
                            'no sign in line 1 and row 5
If Key=@ESC Then Goto MainMenu
If (EditRes!<10) Or (EditRes!>50) Then
          Beep
          Goto MainMenu
End If
Length=EditRes!
Goto MainMenu
```

The EDIT function have a second syntax. This second syntax allows to type access code with an asterisk displaying (*) on a key pressed. This mode is indicated by the <Code> bit. The syntax is :

<Variable> = EDIT(<Line>,<Row>,<Length>,<Sign>,<Point>,<AccessCode>).

```
EditCode!=EDIT(1,5,4,0,0,1) 'Real four numbers edition without point
                            'no sign in line 1 and row 5 with the access
                            'code mode
If Key=@ESC Then Goto MainMenu
If (EditCode!=AdjustCode) Then
   Goto AdjustMenu
Else
   Beep
   Goto MainMenu
End If
```

To edit a string char displaying it on an exact place on the screen, the Dialog 80 and 640 have the EDIT$ instruction. In this instruction, you must define the string char variable (<Variable>), the line (<Line>) and row (<Row>) of the first editing character and the maximum length of this editing (<Length>).

the syntax is : <Variable>=EDIT$(<Line>,<Row>,<Length>). To edit a character on an operator panel, the numeric and alphanumeric key are used, the DEL key to erase, ENTER key to validate and ESC key to escape. To display an alphanumeric character, you need to press the key one or more times..

```
A$=Edit$(2,9,5)        'Edit in line 2, row 9 of 5 characters
```

8-3-5- Buzzer

Two possibilities are offered to use the buzzer of the DIALOG 80, 160 or 640 :

↳ To produce or to stop a continuous sound - BUZZER instruction

Syntax : BUZZER= <ON/OFF>

↳ To make a brief sound - BEEP instruction - Syntax : BEEP

Example :

```
IF KEY<>@ENTER THEN BEEP    'emit a beep on «  enter  » key press
...
Alarm:
BUZZER=ON                   ' emit a continuous sound during
DELAY 1000                  ' a 1s delay
BUZZER=OFF                  ' Stop the buzzer
DELAY 1000
GOTO Alarm
```

8-3-6- Backlight

The operator panel Dialog 640 have a function to control the backlight : BACKLIGHT. The backlight will become inactive after a delay if user don't push on a key panel. The backlight becomes active when a key pane is pushed. The syntax is : BACKLIGHT= <Delay>. <Delay> defines the active time of the backlight after the last key press. This value is an integer which represents the minutes. The zero value allows the backlight to be always inactive and 1 always active. By default, <Delay> is equal to 15mn.

' Warning : le Backlight have a 10000h life duration.

8-3-7- Leds

To drive the leds of the Dialog 80 and 640, you can use the LED (number)=State instruction.The number parameter is the key name where the led is (@F1 … @F6) or for the specific leds its name (@ALARM or @HELP). The State parameter defines the state of the led : switch off (0), switch on (1) or blink (2).

' Warning : The leds of keys F7…F12 on the Dialog640 are driven by the LED(@F1)…LED(@F6) instruction.

8-4- Keys

8-4-1- Dialog 80

F1 à F6	@F1 à @F6	.	@POINT
0 à 9	@0 à @9	Shift	@SHIFT
Help	@HELP	▲	@UP
Alarm	@ALARM	▼	@DOWN
Esc	@ESC	▶	@RIGHT
Mod	@MOD	◀	@LEFT

+/-	@SIGN	Enter	@RETURN

8-4-2- Dialog 160

F1 à F6	@F1 à @F6	+/-	@SIGN
0 à 9	@0 à @9	.	@POINT
D	@D	Shift	@SHIFT
A	@A	▲	@UP
Esc	@ESC	▼	@DOWN
Mod	@MOD	▶	@RIGHT
Ins	@INS	◀	@LEFT
Del	@DEL	Enter	@RETURN

8-4-3- Dialog 640

F1 à F12	@F1 à @F12	.	@POINT
0 à 9	@0 à @9	▲	@UP
Help	@HELP	▼	@DOWN
Alarm	@ALARM	▶	@RIGHT
Esc	@ESC	◀	@LEFT
Mod	@MOD	Enter	@RETURN
+/-	@SIGN		

8-5- Internals menus

8-5-1- General explications

This internal menus are only used in an operator panel Dialog 160 and 640. There functions are :

- Modify the parameters
- Test axis and inputs / outputs in a manual mode
- Read and write the global stored variables
- Control the storage and restoration of the data in flash

✎ Adjust the date and the time

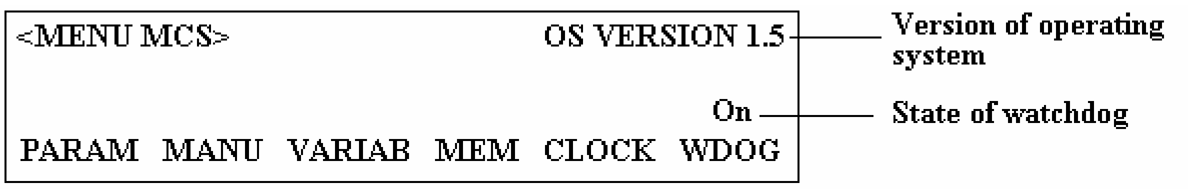
✎ Modify the state of the watchdog

This internal menus are executed with the CALL instruction. These menus are used like a sub-routine. The name of the menus begins with the character '_'. The syntax is : CALL <Name of menu>.

8-5-2- Main menu

Main Menu : MENUMCS

Syntax : CALL _MENUMCS



```
<MENU MCS>                          OS VERSION 1.5

                                               On
PARAM  MANU  VARIAB  MEM  CLOCK  WDOG
```

Function : Gives the access of all the sub-menus.

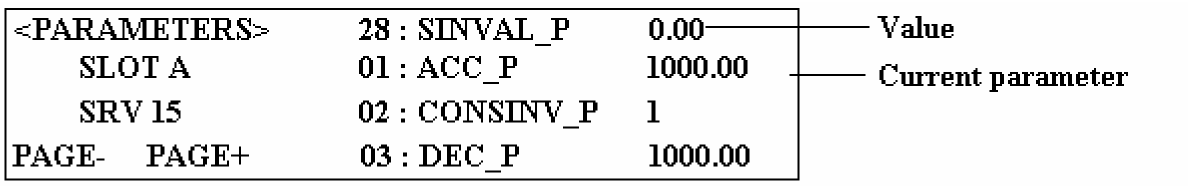
Keys :

F1 : parameters sub-menu	F2 : manual sub-menu
F3 : variables sub-menu	F4 : memory sub-menu
F5 : clock sub-menu	F6 : modify the state of the watchdog
ESC : quit menu	

8-5-3- Parameters sub-menu

Parameter sub-menu : PARAMMCS

Syntax : CALL _PARAMMCS



```
<PARAMETERS>        28 : SINVAL_P      0.00
    SLOT A          01 : ACC_P         1000.00
    SRV 15          02 : CONSINV_P     1
PAGE-   PAGE+       03 : DEC_P         1000.00
```

Keys :

F1 : Previous page	F2 : Next page
← : Previous slot	→ : Next slot
↑ : Previous parameter	↓ : Next parameter
ESC : Exit menu or return in main menu	MOD : modify a parameter

The ACC_P, DEC_P and VEL_P instructions will be used the next ACC%, DEC% and VEL% instructions.

Parameters for SRV85 card:

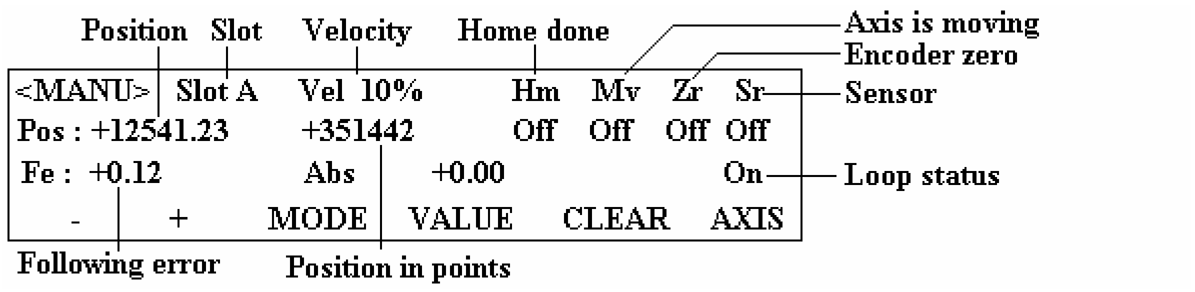
01 : ACC_P	Default acceleration	17 : HOME_P	Home type
02 : BANDWIDTH	Bandwidth	19 : LIMMAX_P	Maximal limit
03 : GFILTER_P	Gain (rejection band filter)	20 : LIMMIN_P	Minimal limit
04 : CONSINV_P	Command invert	21 : LIM_P	Limits activation

05 : CONSMAX_P	Torque limit	22 : MODULO_P	Modulo activation
06 : DEC_P	Default deceleration	23 : MODVAL_P	Modulo value
07 : DISHOME_P	Home shift	24 : OFFSET_P	Offset of analogue command
08 : ENCINV_P	Encoder invert	25 : OUTVEL_P	Output velocity
09 : ENCODER_P	Encoder resolution	26 : POSMIN_P	Position window
10 : FEMAX_P	Max following error	27 : SINVAL_P	Sine value
11 : FILTER_P	reject. band filter activation	28 : SIN_P	Sine acceleration
12 : FREQ_P	Central frequency	29 : UNITREV_P	Units per revolution
13 : GDER_P	Derived gain	30 : VELFF_P	Velocity feed forward
14 : GINT_P	Integral gain	31 : VELHOME_P	Home velocity
15 : GPROP_P	Proportional gain	32 : VEL_P	Default velocity
16 : GTORQUE	Torque constant	33 : ZERO_P	Program zero

8-5-4- Manual sub-menu for axis

Manual sub-menu : MANUMCS for axis

Syntax : CALL _MANUMCS



```
<MANU> | Slot A   Vel 10%     Hm   Mv   Zr   Sr
Pos : +12541.23   +351442     Off  Off  Off  Off
Fe :  +0.12       Abs   +0.00                On
   -      +      MODE  VALUE  CLEAR  AXIS
```

Keys :

F1 : movement in negative dir.	F2 : movement in positive direction
F3 : moving mode	F4 : value or home type
F5 : clear position	F6 : change loop
F7 : start a movement	F8 : stop a movement
← : previous slot	→ : next slot
↑ : increase velocity	↓ : decrease velocity F12 : jump between 5 and 80%

ESC : Exit menu or return in main menu

Modes : Abs : Absolute move to position defined by VALUE (F4)

Rel : Relative movement with distance defined by VALUE (F4)

Inf+ : Infinite movement in positive direction

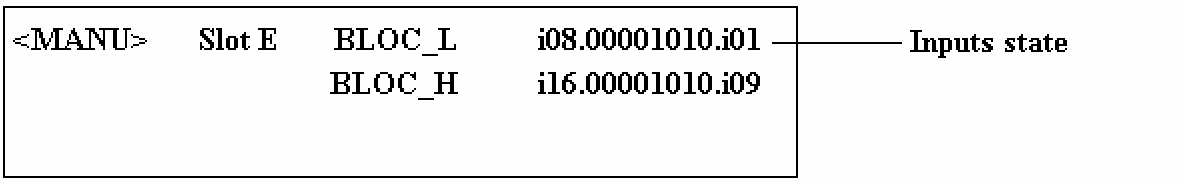
Inf- : Infinite movement in negative direction

Home : Home process using VALUE as home type

8-5-5- Manual sub-menu for digital inputs

Main menu : MANUMCS

Syntax : CALL _MANUMCS



```
<MANU>    Slot E    BLOC_L     i08.00001010.i01
                    BLOC_H     i16.00001010.i09
```

Keys :

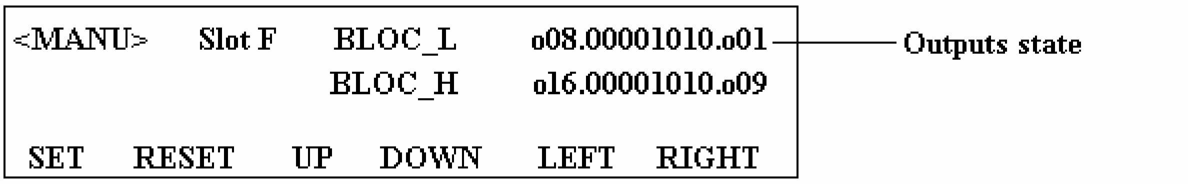
← : Previous slot → : Next slot

ESC : Exit menu or return in main menu

8-5-6- Manual sub-menu for digital outputs

Main menu : MANUMCS

Syntax : CALL _MANUMCS



```
<MANU>    Slot F    BLOC_L     o08.00001010.o01
                    BLOC_H     o16.00001010.o09

 SET    RESET    UP   DOWN    LEFT   RIGHT
```

Keys :

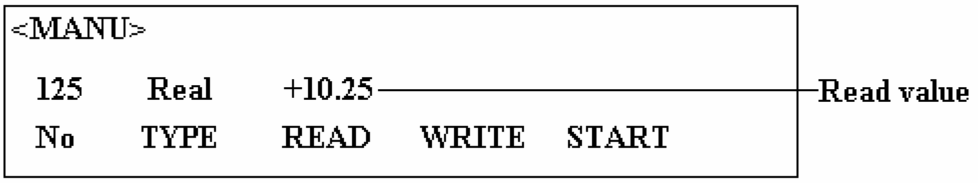
← : Previous slot	→ : Next slot
F1 : Set the bit	F2 : Reset the bit
F3 : Next bloc	F4 : Next bloc
F5 : Next bit	F6 : Previous bit

ESC : Exit menu or return in main menu

8-5-7- Variables sub-menu

Main menu : VARIABMCS

Syntax : CALL _VARIABMCS



```
<MANU>

 125    Real    +10.25
 No     TYPE    READ    WRITE   START
```

Keys :

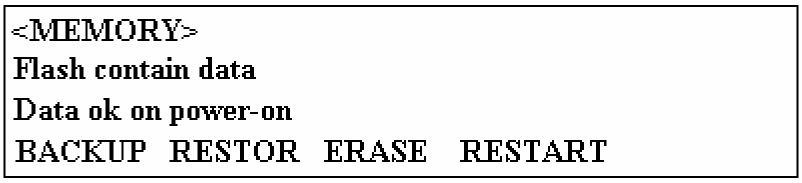
F1 : Variable number	F2 : Variable type	
F3 : Read	F4 : Write	F5 : Real time read
↑ : Next variable	↓ : Previous Variable	

ESC : Exit menu or return in main menu

8-5-8- Memory sub-menu

Main menu : MEMMCS

Syntax : CALL _MEMMCS



```
<MEMORY>
Flash contain data
Data ok on power-on
BACKUP  RESTOR  ERASE   RESTART
```

Keys :

F1 : Backup data in flash   F2 : Restore dat   a from flash

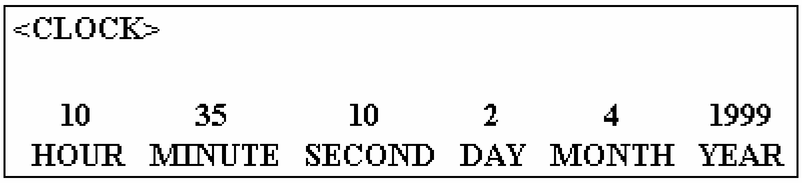
F3 : Clear data in flash   F4 : Restart MCS32 Ex

ESC : Exit menu or return in main menu

8-5-9- Clock sub-menu

Main menu : CLOCKMCS

Syntax : CALL _CLOCKMCS



```
<CLOCK>

  10     35      10      2     4     1999
 HOUR  MINUTE  SECOND   DAY  MONTH  YEAR
```

Keys :

F1 : Set hour   F2 : Set minutes   F3 : Set seconds

F4 : Set day   F5 : Set month   F6 : Set year

ESC : Exit menu or return in main menu

9- PARAMETER LIST

9-1- Axis

ZERO_P	Program zero

9-2- Encoder

ENCODER_P	Encoder resolution
UNITREV_P	Number of units per rotation
ENCINV_P	Encoder invert
MODULO_P	Modulo activation
MODVAL_P	Modulo definition

9-3- Regulation

BANDWIDTH_P	Bandwidth rejection (SRV85)
CONSINV_P	Command voltage invert
CONSMAX_P	Torque limitation (SRV85)
FEMAX_P	Maximal following error
FILTER_P	Band rejection filter activation (SRV85)
FREQ_P	Central frequency of the band rejection filter (SRV85)
GDER_P	Derived gain
GFILTER_P	Band rejection filter gain (SRV85)
GINT_P	Integral gain
GPROP_P	Proportional gain
GTORQUE_P	Torque constant (SRV85)
OFFSET_P	Analogue offset command
OUTVEL_P	Speed anticipation
POSMIN_P	Position window
VELFF_P	Acceleration anticipation

9-4- Speed profile

ACC_P	Default acceleration
DEC_P	Default deceleration
VEL_P	Default speed
SINACC_P	Sine acceleration activation
SINVAL_P	Sine acceleration value

9-5- Home

DISHOME_P	Home distance
INPHOME_P	Sensor input
VELHOME_P	Home velocity

9-6- Step by step driving (STP85)

CLKINV_P	This parameter allows to work in positive or negative logic on the CLOCK signal.
ENBINV_P	This parameter allows to work in positive or negative logic on the ENABLE signal.
ILMINV_P	This parameter allows to work in positive or negative logic on the ILIMIT signal.
STEP_P	This parameter specify the number of steps per motor revolution.
STPINV_P	This parameter allows to work in positive or negative logic on the DIRECTION signal.
THIGH_P	This parameter specifies the high level duration of the CLOCK signal.
THOLD_P	This parameter specifies setup and hold times of the DIRECTION signalThold

.

9-7- Software thrusts

LIMMIN_P	Minimum limit
LIMMAX_P	Maximum limit
LIM_P	Thrusts activation

9-8- Agreement between limits expressed in pulses and limits expressed in users unit

Users unit limits are expressed with this expression :

⇨ Limituser unit= (Limitpulses × Parameter UNITREV_P)/(Parameter ENCODER_P)

The user unit is the selected unit in the board configuration screen « Axis menu » of the axis board (Ex : mm, degree, round…).

Example :

↳ Axis with a 1000 pulses encoder ⇨ ENCODER_P(Axis)=4×1000=4000

↳ Encoder at the end of a ball screw with 5mm by step ⇨ UNITREV_P(Axis)=5

↳ the value of the POSMIN_P parameter is 0 to 32767 pulses, so 0 to 40,95875 mm.

9-9- APPENDIX

9-9-1- ACC_P – Default acceleration

Syntax :	**ACC_P**(<Axis>) = <Expression>
Unit :	User unit per s² (Ex : mm/s², degree/s², …)
Limits :	5.10^-5 to 5,76.10^8 pulses/s²
Accepted types:	<Expression> : real
Description :	This parameter specifies the default acceleration loaded in the axis card at the startup of the MCS.
Remarks :	It is also used by ACC% instruction.

Example :

```
ACC_P(X)=5000
ACC%(X)=50 ' Acceleration  = 2500
```

	`MOVA ...`
See also :	DEC_P

9-9-2- BANDWIDTH_P – Bandwidth rejection (SRV85)

Syntax :	**BANDWIDTH_P**(<Axis>) = <Expression>
Limits :	<Expression> : –32767 to +32767 pulses
Accepted types :	<Expression> : real
Description :	This parameter specifies the bandwidth rejection of the SRV85 band rejection filter.
See also :	GFILTER_P, FREQ_P, FILTER_P, GTORQUE_P

9-9-3- CLKINV_P – CLOCK signal invert

Syntax :	**CLKINV_P**(<Axis>) = ON / OFF
Description :	This parameter allows to work in positive or negative logic on the CLOCK signal.
Remarks :	By default this parameter is OFF (Positive logic). See the drive documentation before any modifications on this parameter. A wrong value may cause worse motor operation !
Example :	CLKINV_P(X)=OFF

9-9-4- CONSINV_P– Command voltage invert

Syntax :	**CONSINV_P**(<Axis>) = ON / OFF
Description :	This parameter allows to invert the analogue command voltage.
Remarks :	This parameter is used with ENCINV_P to change command voltage direction.
Example :	`CONSINV_P(X)=On` *'Command inverted*
See also :	ENCINV_P

9-9-5- CONSMAX_P – Torque limitation (SRV85)

Syntax :	**CONSMAX_P**(<Axis>) = <Expression>
Units :	<Expression> : Volt
Accepted types :	<Expression> : real
Limits :	<Expression> : -10 to +10 Volt
Description :	This parameter specifies the limit value of torque of SRV85 board.
See also :	GTORQUE_P

9-9-6- DEC_P – Default deceleration

Syntax :	**DEC_P**(<Axis>) = <Expression>
Unit :	User unit per s² (Ex : mm/s², degree/s²,…)
Limits :	5.10^-5 to 5,76.10^8 pulses/s²

Accepted types :	<Expression> : real
Description :	This parameter specifies the deceleration loaded in the axis card at the startup of the MCS.
Remarks :	It is also used by DEC% instruction.

Example :	`DEC_P(X)=5000`
	`DEC%(X)=50        'The deceleration is fixed to 50%`
	`                  'then 2500 Units /s²`

See also :	ACC_P

9-9-7- DISHOME_P – Home distance

Syntax :	**DISHOME_P**(<Axis>) = <Expression>
Unit :	<Expression> : User unit (Ex : mm, degree ; …)
Limits :	+/-2^24 pulses
Accepted types :	<Expression> : real
Description :	This parameter specifies home distance disengaging.

Example :	`DISHOME_P(X)=500`
	`HOME(X)  ' Final position  = 500`

See also :	INPHOME_P

9-9-8- ENBINV_P – ENABLE signal invert

Syntax :	**ENBINV_P**(<Axis>) = ON / OFF
Description :	This parameter allows to work in positive or negative logic on the ENABLE signal.
Example :	ENBINV_P(X)=OFF ‘ positive logic ENABLE(X)=OFF ‘ enable signal = 0 ENABLE(X)=ON ‘ enable signal = 1

9-9-9- ENCINV_P – Encoder invert

Syntax :	**ENCINV_P**(<Axis>) = ON /OFF
Description :	This parameter allows to invert the counting direction.
Remarks :	This parameter is used with CONSINV_P, to change the axis movement direction. It can also be used to compensate an invert in the encoder connecting.
Example :	`ENCINV_P(X)=On     'Counter direction invert`
See also :	CONSINV_P

9-9-10- ENCODER_P – Encoder resolution

Syntax :	**ENCODER_P**(<Axis>) = <Expression>
Unit :	<Expression> : pulses
Limits :	<Expression> : 1 to 65535 pulses
Accepted types :	<Expression> : integer

Description :	This parameter specifies the encoder points number x 4 linked to the axis card.
Example :	`ENCODER_P(<Axis>)=2000   '500 points encoder`
See also :	UNITREV_P

9-9-11- FEMAX_P – Maximum following error

Syntax :	**FEMAX_P**(<Axis>) = <Expression>
Unit :	<Expression> : User unit (Ex : mm, degree,…)
Limits :	<Expression> :–32767 to +32767 pulses
Accepted types :	<Expression> : real
Description :	This parameter specify the maximal allowable following error.
Remarks :	<Expression> must be an valid real expression. This parameter is used to modify the following error value beyond the system passes in default state : all the axis passes in a open loop state, watchdog is opened and buffer of movements is cleared.

Example :	`FEMAX_P(X)=5.0 'Max error 5.0 Units`
	`MOVA...`

See also :	POSMIN_P, SECURITY

9-9-12- FILTER_P – Band rejection filter activation (SRV85)

Syntax :	**FILTER_P**(<Axis>) = <Expression>
Accepted types :	<Expression> : bit
Description :	This parameter activates (1) or not (0) the SRV85 band rejection filter.
See also :	BANDWIDTH_P, GFILTER_P, FREQ_P, GTORQUE_P

9-9-13- FREQ_P – Middle frequency of the band rejection filter (SRV85)

Syntax :	**GFILTER_P**(<Axis>) = <Expression>
Limits :	<Expression> : –32767 to +32767 pulses
Accepted types :	<Expression> : real
Description :	This parameter specifies the middle frequency of the SRV85 band rejection filter.
See also :	BANDWIDTH_P, GFILTER_P, FILTER_P, GTORQUE_P

9-9-14- GDER_P – Derivative gain

Syntax :	**GDER_P**(<Axis>) = <Expression>
Unit :	<Expression> : 1.0172 10^-7 Volt/(incr/s)
Limits :	<Expression> : 0 to 32000
Types :	<Expression> : Real
Description :	This parameter specifies the derivative gain of the PID controller.
Remarks :	This parameter is used to modify the derivative gain at any time.
Example :	`GDER_P(X)=100`
See also :	GINT_P, GPROP_P, OUTVEL_P, VELFF_P

9-9-15- GFILTER_P – Band rejection filter gain (SRV85)

Syntax :	**GFILTER_P**(<Axis>) = <Expression>
Limits :	<Expression> : –32767 to +32767 pulses
Accepted types :	<Expression> : real
Description :	This parameter specifies the gain of the SRV85 band rejection filter.
See also :	BANDWIDTH_P, FREQ_P, FILTER_P, GTORQUE_P

9-9-16- GINT_P – Integral gain

Syntax :	**GINT_P**(<Axis>) = <Expression>
Unit :	<Expression> : 9.155 10^-7 Volt/(incr*s)
Limits :	<Expression> : 0 to 32 000 000 for SRV85 board and 0 to 32 000 for other servo board
Accepted types :	<Expression> : Real
Description :	This parameter specifies integral gain of the PID.
Remarks :	This parameter is used to modify the integral gain at any time.
Example :	`GINT_P(X)=50`
See also :	GDER_P, GPROP_P, OUTVEL_P, VELFF_P

9-9-17- GPROP_P – Proportional gain

Syntax :	**GPROP_P**(<Axis>) = <Expression>
Unit :	<Expression> : 3.052 10^-5 Volt/incr
Limits :	<Expression> : 0 to 32000
Accepted types :	<Expression> : Real
Description :	This parameter specifies proportional gain of the PID.
Remarks :	This parameter is used to modify the proportional gain at any time.
Example :	`GPROP_P(X)=1000`
See also :	GINT_P, GDER_P, OUTVEL_P, VELFF_P

9-9-18- GTORQUE_P – Torque constante (SRV85)

Syntax :	**GTORQUE_P**(<Axis>) = <Expression>
Unit :	<Expression> : 1.0172 10^-7 Volt/(incr/s)
Limits :	<Expression> : 0 to 32000
Accepted types :	<Expression> : Real
Description :	This parameter specifies the torque constant of the SRV85 band rejection filter.
See also :	CONSMAX_P

9-9-19- HOME_P – Home type (SRV85)

Syntax 1 :	**HOME_P**(<Axis>)=<Expression>

Syntax 2 : <Variable>=**HOME_P**(<Axis>)

Limits : <Expression>, <Variable> : de 1 à 10

Accepted types : <Expression>, <Variable> : Octet

Description : This parameter specifies the type of axis home. The values 1 to 10 matches with a home type defined in the window configuration of the axis board

9-9-20- INPHOME_P – Sensor input

Syntax : **INPHOME_P**(<Axis>) = <Entrée>

Description : This parameter specifies the input used as home sensor input.

Remarks : <Input> must be a digital input name or an axis card name. If <Axis> is used as the input name, the axis card input will be used.

Example :

```
INPHOME_P(X)=X    ' Axis card input is used
INPHOME_P(X)=SensorHome  ' Digital input named
                             ' SensorHome is used
```

See also : DISHOME_P

9-9-21- ILMIN_P – ILIMIT signal invert

Syntax : **ILMINV_P**(<Axis>) = ON / OFF

Description : This parameter allows to work in positive or negative logic on the ILIMIT signal.

Example : ILMINV_P(X)=ON ‘ negative logique.

ILIMIT(X)=OFF ‘ ilimit signal = 1

ILIMIT(X)=ON ‘ ilimit signal = 0

9-9-22- LIMMAX_P – Maximum position

Syntax : **LIMMAX_P**(<Axis>) = <Expression>

Unit : <Expression> : User unit (Ex : mm, degree,…)

Accepted types : <Expression> : real

Description : This parameter specifies the maximum position.

Remarks : If the position passes LIMMAX_P then LIM_S et LIMMAX_S become true.

Example :

```
LIMMAX_P(X)=100.0 ' If position > 100 then error
IF LIM_S(X)=True Then Goto Error
```

See also : LIMMIN_P, LIM_P

9-9-23- LIMMIN_P – Minimum position

Syntax : **LIMMIN_P**(<Axis>) = <Expression>

Unit : <Expression> : User unit (Ex : mm, degree,…)

Limits : <Expression> : real

Description : This parameter specifies the minimum position.

Remarks : If the position passes LIMMIN_P then LIM_S et LIMMIN_S become true.

Example :

```
LIMMIN_P(X)=-100.0 ' If position < -100 then error
```

```
IF LIM_S(X)=True Then Goto Error
```

See also : LIMMAX_P, LIM_P

9-9-24- LIM_P – Thrusts activation

Syntax : **LIM_P**(<Axis>)= ON / OFF

Description : This parameter defines thrusts activation.

Remarks : This parameter is used to activate or disable software thrusts. When this parameter is false LIMMAX_P and LIMMIN_P have no effects.

Example : LIM_P(X)=ON

See also : LIMMAX_P, LIMMIN_P

9-9-25- MODULO_P – Modulo activation

Syntax : **MODULO_P**(<Axis>) = ON / OFF

Description : This parameter allows to declare a modulo axis. The axis position will be between 0 and the value of the modulo

Remarks : An infinite axis must be define like a modulo axis

Warning : For a modulo axis, the instruction HOME put temporarily to 0 his parameter MODULO_P. If you stop one task who is complying the instruction HOME, forecast to reforce MODULO_P to 1.

Example :

```
MODVAL_P(X)=360   'Modulo axis 360 Units
MODULO_P(X)=On    'Modulo activation
MOVA(X=100)       'Movement of the axis in the positive
                  'direction at the position of 100°
MOVA(X=-20)       'Movement of the axis in the negative sens
                  'direction at the position of 20°
```

See also : MODVAL_P

9-9-26- MODVAL_P – Modulo definition

Syntax : **MODVAL_P**(<Axis>) = <Expression>

Unit : <Expression> : User unit (Ex : mm, degree,…)

Limits : <Expression> : +/-2^24 pulses

Accepted types : <Expression> : real

Description : This parameter allows to declare modulo value.

Remarks : If the modulo is activated the position will go between 0 and the value.

Example :

```
MODVAL_P(X)=360   'Modulo axis 360 Units
MODULO_P(X)=On
```

See also : MODULO_P

9-9-27- OFFSET_P – Analogue offset command (SRV85)

Syntax : **OFFSET_P**(<Axis>) = <Expression>

Units : <Expression> : Volt

Accepted types :	<Expression> : real
Limits :	<Expression> : -10 to +10 Volt
Description :	This parameter specifies the analogue offset command of the SRV85 board.

9-9-28- OUTVEL_P – Speed anticipation

Syntax :	**OUTVEL_P**(<Axis>) = <Expression>
Unit :	<Expression> : 1.0172 10^-7 Volt/(incr/s)
Limits :	<Expression> : 0 to 32000
Accepted types :	<Expression> : Real
Description :	This parameter specifies the speed anticipation coefficient. It allows to decrease following error.
Remarks :	This parameter must be used if the axis is used with a synchronization or interpolation function. The usual values are between 0 and 2000.
Example :	OUTVEL_P(X)=500
See also :	GDER_P, GPROP_P

9-9-29- POSMIN_P – Position window

Syntax :	**POSMIN_P**(<Axis>) = <Expression>
Unit :	<Expression> : User unit (Ex : mm, degree,…)
Limits :	<Expression> : 0 to 32767 pulses
Accepted types :	<Expression> : real
Description :	This parameter specifies the minimum positioning error.
Remarks :	This parameter is used to modify the minimum positioning error between the real position and the theoretical position. After a moving , if the difference between the real position and the required position is lower than POSMIN_P, the system considers that the position is reached.

Example :

```
POSMIN_P(X)=0.1
MOVA(X=100)  ' The real position is between 99.9 and 100.1
```

9-9-30- SINACC_P – Sine acceleration activation

Syntax :	**SINACC_P**(<Axis>) = ON / OFF
Description :	This parameter allows to use the sinus acceleration.
Remarks :	This parameter is used to specify the behaviour during the acceleration and deceleration phases.The sinus acceleration allows to reduce the efforts inflicted to the mechanics during the acceleration and deceleration phases .
Example :	`SINACC_P(X)=On`
See also :	SINVAL_P

9-9-31- SINVAL_P – Sinus acceleration value

Syntax :	**SINVAL_P**(<Axis>) = <Expression>
Limits :	<Expression> : 0 to 10

Accepted types :	<Expression> : real
Description :	This parameter specifies sinus flat zone length. Zone= Expression/(Expression +2)

Example :	`SINVAL_P(X)=0      ' pure sine`
	`SINVAL_P(X)=2      ' 50% flat zone (2/4)`

See also :	SINACC_P

9-9-32- STEP_P – Number of steps per motor revolution

Syntax :	**STEP_P**(<Axis>) = <Expression>
Units :	<Expression> : step
Limits :	<Expression> : from 1 to 65535 steps
Accepted types :	<Expression> : real
Description :	This parameter specify the number of steps per motor revolution.
Remarks :	If the motor is used by half of step, the value is in half of step (same for quarter of step and micro step).
Example :	STEP_P(X)=1000 ‘ 1000 step motor driven by step STEP_P(X)=800 ‘ 200 step motor driver by quarter of step

9-9-33- STPINV_P– Motor direction invert

Syntax :	**STPINV_P**(<Axis>) = ON / OFF
Description :	This parameter allows to work in positive or negative logic on the DIRECTION signal.
Remarks :	When setting up the axis, this parameter allows to change the motor direction without changing the wiring.

9-9-34- THIGH_P – CLOCK signal duration

Syntax :	**THIGH_P**(<Axis>) = <Expression>
Units :	<Expression> : ½ micro-seconde
Limits :	<Expression> : for 1 to 511
Accepted types :	<Expression> : integer
Description :	This parameter specifies the high level duration of the CLOCK signal. Remarks : By default this parameter is equal to 4 µs. See the drive documentation before any modifications on this parameter. A wrong value may cause worse motor operation !
Example :	THIGH_P(X)=8 ‘ 4µs

9-9-35- THOLD_P– DIRECTION signal setup and hold time

Syntax :	**THOLD_P**(<Axis>) = <Expression>

Units :	<Expression> : ½ micro-seconde
Limits :	<Expression> : from 1 to 511
Accepted types :	<Expression> : integer
Description :	This parameter specifies setup and hold times of the DIRECTION signalThold
Remarks :	By default this parameter is equal to 4 µs. The times are indicated in the drive documentation. Always use the highest time. A wrong value may cause worse motor operation !
Example :	THOLD_P(X)=14 ‘on drive guide Tsetup>=7µs Thold=5µs ‘ used value for this parameter = 7µs

9-9-36- UNITREV_P – Number of points per revolution

Syntax :	**UNITREV_P**(<Axis>) = <Expression>
Unit :	<Expression> : User unit (Ex : mm, degree,…)
Limits :	<Expression> : 1 to 2^24 pulses
Accepted types :	<Expression> : real
Description :	This parameter specifies number of units per revolution.

Example :	`UNITREV_P(<Axis>)=5      'Ball screw with a 5mm step and`
	`'an encoder at the end of the screw`

See also :	ENCODER_P

9-9-37- VELFF_P – Acceleration anticipation

Syntax :	**VELFF_P**(<Axis>) = <Expression>
Unit :	<Expression> : 1.0172 10^-7 Volt/(incr/s)
Limits :	<Expression> : 0 to 32000
Accepted types :	<Expression> : Real
Description :	This parameter specifies the acceleration anticipation coefficient. This parameter can decrease the following error during the acceleration and deceleration phases.
Remarks :	This parameter is used when the axis used synchronization or interpolation movements. The usual value are between 0 and 2000.

Example :	`VELFF_P(X)=20`
	`MOVA...`

See also :	GDER_P, GPROP_P, GINT_P, OUTVEL_P

9-9-38- VELHOME_P – Home velocity

Syntax :	**VELHOME_P**(<Axis>) = <Expression>
Unit :	<Expression> : User unit per second (Ex : mm/s, degree/s,…)
Limits :	<Expression> : 5.10^-5 to 1,5.10^6 pulses/s
Accepted types :	<Expression> : real
Description :	This parameter specifies the home velocity in unit per second.
Remarks :	This speed must be low.
Example :	`VELHOME_P(X)=0.2*VEL_P(X)      ' Speed=20% of the default speed`

9-9-39- VEL_P – Default velocity

Syntax :	**VEL_P**(<Axis>) = <Expression>
Unit :	<Expression> : User unit per second (Ex : mm/s, degree/s,…)
Limits :	<Expression> : 5.10^-5 to 1,5.10^6 pulses/s for SRV15 and SSI15
	<Expression> : 5.10^-5 to 2 ,457.10^7 pulses/s for SRV85
Accepted types :	<Expression> : real
Description :	This parameter specifies default velocity loaded at the startup of the MCS
Remarks :	It is also used by VEL% function .

Example :	`VEL_P(X)=2000`
	`VEL%(X)=10 'Speed : 200 Units / s`
	`MOVA ...`

See also :	ACC_P, DEC_P

9-9-40- ZERO_P – Program zero

Syntax :	**ZERO_P**(<Axis>) = <Expression>
Unit :	<Expression> : User unit (Ex : mm, degree,…)
Limits :	<Expression> : +/-2^24
Accepted types :	<Expression> : real
Description :	This parameter specifies program zero position.
Remarks :	This parameter is used to specify the difference between physical zero and software zero. All the position used in the tasks are referenced with this parameter.
Example :	`ZERO_P(X)=10.0 '10 mm between physical zero and program zero`

10- OPERATOR AND INSTRUCTIONS LIST

10-1- Program

CALL	Call a subroutine
ICALL	Call a subroutine
END	bloc end
EXIT SUB	Exit subroutine
GOTO	Branch to label
JUMP	Branch to label
PROG ... END PROG	Program
SUB ... END SUB	Subroutine

10-2- Arithmetical

+	Addition
-	Subtraction
*	Multiplication
/	Division

10-3- Mathematical

ABS	Absolute value
ARCCOS	Cosine invert
ARCSIN	Sine invert
ARCTAN	Tangent invert
COS	Cosine
DIV	Integer divide
EXP	Exponential
FRAC	Fractional part
INT	Integer part
LOG	Logarithm
MOD	Modulus
SGN	Sign
SIN	Sine
SQR	Square root
TAN	Tangent
^	Exponent

10-4- Loops

FOR ... TO ... STEP ... NEXT

REPEAT ... UNTIL
WHILE ... DO ... END WHILE

10-5- Logical

<<	Left shift
>>	Right shift
AND	AND operator
NOT	Complement Operator
OR	OR operator
XOR	Exclusive OR operator

10-6- Test

<	Lower
<=	Lower or equal
<>	Different
=	Equal / affect
>	Greater
>=	Greater or equal
CASE ...	Multiple tests
IF ... THEN ... ELSE ... END IF	Test structure

10-7- Char string

ASC	Convert char to ASCII
CHR$	Convert ASCII to char
FORMAT$	Created a formatted string
INSTR	Search a sub-string
LCASE$	Lowercase
LEFT$	Left part of string
LEN	String length
LTRIM$	Suppress left spaces
MID$	String part
RIGHT$	Right part of string
RTRIM$	Suppress right spaces
SPACE$	Spaces made string
STR$	Convert numeric to string
STRING$	Create a string
UCASE$	Uppercase
VAL	Convert string to numeric

10-8- Motion control

10-8-1- Axis control

ACC	Acceleration
ACC%	Acceleration in percent
ADDMOV	Movements superposition (SRV85)
ADDSTOP	Stop the movements superposition (SRV85)
AXIS	Axis loop control
AXIS_S	Axis loop state
BACKLASH	Taking up of play
BUFMOV_S	Number of waiting orders
CLEAR	Put to zero the axis position
CONS	Command voltage
CONS_S	Command voltage state
CORPOS_S	Value of applied correction
CORRECTION	Correction function triggered on capture input (SRV85)
CORRECTION_S	Correction state (SRV85)
DEC	Deceleration
DEC%	Deceleration in percent
ENABLE	Enable signal driving
ENABLE_S	Enable signal state
FE_S	Following error
FEMAX_S	Following error limit
FILTER	Activate or desactivate a filter
HOME	Home position
HOME_S	Home state
ICORRECTION	Correction function (SRV85)
ILIMIT	Driving ILIMIT state
ILIMIT_S	ILIMIT signal state
LIMMIN_S	Minimum thrust state
LIMMAX_S	Maximum thrust state
LIM_S	Thrusts state
LOOP	Virtual mode
MERGE	Movements merging
MOVE_S	Movement state
ORDER	Movement order number
ORDER_S	Current order number
POS	Reached position
POS_S	Real position
SENSOR_S	Sensor state
SENSOR1_S	C1 Sensor state (SRV85)
SENSOR2_S	C2 Sensor state (SRV85)

STOPCORRECTION	Stop of correction function (SRV85)
THEORICSRCK1	Selection source of synchro
TRIGGER	Launches a movement
VEL	Speed
VEL%	Speed in percent
VEL_S	Real speed
ZERO_S	State of the encoder zero

10-8-2- Positioning

MOVA	Absolute movement
MOVAC	absolute movement triggered on capture input
MOVAP	Launch absolute movement
MOVAT	Absolute triggered movement
MOVR	Relative movement
MOVRT	Relative triggered movement
SSTOP	Without waiting axis stop
STOP	Axis stop
STTA	Start an absolute movement
STTI	Start an infinite movement
STTR	Start a relative movement
TRAJ	Trajectory

10-8-3- Synchronization

ABSCAM	Absolute electronic camebox
CAM	Electronic cam
CAMC	Electronic cam triggered on capture input (SRV85)
CAMFROMPOINT	Slave position in the cam
CAMNUM_S	Number of the running cam
CAMSEG_S	Equation number of the running cam
CAM_S	State of the cam
DISABLERECALE	Desactivation of recale
ENABLERECALE	Automatic fitting encoder function
GEARBOX	Gearbox
GEARBOXRATIO	Change the reduction ratio of an electrical shaft
LOADABSCAMEX	Load an absolute cambox in the servo module
LOADCAMEX	Load a cam in a servo board
LOADPOINT	Point of a cam in a servo board
LOADS	Load a synchronized movement
MASTEROFFSET	Shift dynamically the master positio
MOVC	Circular movement
MOVL	Linear movement
MOVS / MOVSP	Synchronized movement
MOVSC	Synchronized movement triggered on capture input

MOVST	Synchronized movement triggered
SLAVEOFFSET	Shift dynamically the slave position
STARTABSCAM	Start an absolute cambox
STARTCAM	Launch the execution of a cam
STARTCAMC	Launch the execution of a cam on capture input
STARTCAMT	Execution of a cam triggered
STARTS	Start a synchronized movement
STOPI	Stop interpolation moves

10-8-4- Capture

CAPTURE	Starting position capture
CAPTURE1	Starting position capture (SRV85)
CAPTURE2	Starting position capture (SRV85)
REGPOS_S	Captured position
REGPOS1_S	Captured position (SRV85)
REGPOS2_S	Captured position (SRV85)
REG_S	Capture state
REG1_S	Capture state (SRV85)
REG2_S	Capture state (SRV85)

10-8-5- Enhanced Event Funtion

AXISCONDITION	Configuration des conditions
AXISEVENT	Events configuration or read events out of factual tasks
AXISLEVEL	Instructions definition
AXISMEASURE	Read distances and timings
CLEARFIFO	Clear the list of state change
FIFO	Read distances between state change
FIFONUMBER	Number of observed state change
GETAXISEVENT	Reading events out of factual tasks
MODIFYAXISEVENT	Event configuration

10-9- PLC

10-9-1- Logical inputs / outputs

CAMBOX	Cambox
CAMBOXDELAY	Anticipation delay
CAMBOXSEG	Cambox segment
CAMBOXVAR	Cambox on variable
INP	1 digital input reading
INPB	8 digital inputs reading
INPW	16 digital inputs reading
OUT	1 digital output writing

OUTB	8 digital outputs writing
OUTW	16 digital outputs writing
PLCINIT	PLC function initialisation
PLCINP	Read TOR input
PLCINPB	Read a 8 inputs block
PLCINPNE	Read a negative edge on PLC TOR input
PLCINPPE	Read a positive edge on PLC TOR input
PLCINPW	Read a 16 inputs block
PLCOUT	Write a output
PLCOUTB	Write a 8 outputs block
PLCOUTW	Write a 16 outputs block
PLCREADINPUTS	Read the PLC inputs
PLCWRITEOUTPUTS	Write the PLC outputs
SETINP	Inputs filter and invert
SETOUT	Outputs invert
STARTCAMBOX	Start a cambox
STOPCAMBOX	Stop a cambox
WAIT	Condition waiting

10-9-2- Analogue inputs / outputs

ADC	Analogue inputs
DAC	Analogue outputs

10-9-3- Timing

DATE$	Current date in string
DELAY	Passive wait
GETDATE	Current date
GETTIME	Current time
SETDATE	Set date
SETTIME	Set time
TIME	Global time base
TIMER	Global wide time base
TIME$	Current time in string

10-9-4- Event handling

DIFFUSE	Send event
GETEVENT	Read event
MODIFYEVENT	Event configuration
SIGNAL	Send event
WAIT EVENT	Passive event wait

10-9-5- Counter

CLEARCOUNTER	RAZ counter

COUNTER_S	Counter read
SETUPCOUNTER	Counter configuration

10-10- Terminal operator

10-10-1- Dialog 80, 160 and 640

BEEP	Brief sound
BUZZER	Continuous sound
CLS	Clear screen
CURSOR	Clear or display cursor
EDIT	Editing
INKEY	Read a key
KEY	Last key
KEYDELAY	Delay before repeat key
KEYREPEAT	Repeat key period
LOCATE	Cursor position
PRINT	Print a text
WAIT KEY	Key waiting

10-10-2- Dialog 80 et 640

EDIT$	alphanumeric data capture
LED	driving leds

10-10-3- Dialog 640

BACKLIGHT	screen saver control
BOX	Draw box
FONT	Font selection
HLINE	Draw horizontal line
PIXEL	Draw point
VLINE	Draw vertical line

10-11- Task handling

CONTINUE	Continue task execution
HALT	Stop task
RUN	Start task
SUSPEND	Suspend a task
STATUS	Task state

10-12- Communication

CARIN	Input buffer state
CAROUT	Output buffer state
CLEARIN	Clear input buffer

CLEAROUT	Clear output buffer
CLOSE	Close communication port
INPUT	Data reading
INPUT$	Char string reading
OPEN...AS...	Open a communication port
OUTEMPTY	Output buffer status
PRINT	Write on the communication port
TX485	Modify RS485 output state

10-13- Flash, Security and other functions

CRC	Return the checksum value
CLEARFLASH	Clear flash memory
DISPLAY	7 segments display
FLASHOK	Test data in flash memory
FLASHTORAM	Restore from flash memory
POWERFAIL	Power failure detect
RAMTOFLASH	Backup to flash memory
RAMOK	Test ram memory
RESTART	Restart system
SECURITY	Define security actions
VERSION	Operating system version
WATCHDOG	Watchdog

10-14- Conversion

CVL	Convert string to long integer
CVLR	Convert string to reverse long integer
CVI	Convert string to integer
CVIR	Convert string to reverse integer
LONGTOINTEGER	Convert long integer to integer
MKL$	Convert long integer to string
MKLR$	Convert reverse long integer to string
MKI$	Convert integer to string
MKIR$	Convert reverse integer to string
REALTOLONG	Convert real to long integer
REALTOINTEGER	Conversion real to integer
REALTOBYTE	Conversion real to byte

10-15- Agreement between limits expressed in pulses and limits expressed in users unit

Users unit limits are expressed with this expression :

⇨ Limituser unit= (Limitpulses × Parameter UNITREV_P)/(Parameter ENCODER_P)

The user unit is the selected unit in the boardard configuration screen « Axis menu » of the axis board (Ex : mm, degree, round…).

Example :

↳ Axis with a 1000 pulses encoder ⇨ ENCODER_P(Axis)=4×1000=4000

↳ Encoder at the end of a ball screw with 5mm by step ⇨ UNITREV_P(Axis)=5

↳ the value of the POSMIN_P parameter is 0 to 32767 pulses, so 0 to 40,95875 mm.

10-16- Alphabetic list

10-16-1- Addition (+)

Syntax :	<Expression1> + <Expression2>
Accepted types :	Byte, Integer, Long integer, real or string
Description :	This operator adds two numeric expressions and return a value type identical as its operand.
Remarks :	<Expression1> and <Expression2> must be numerical valid expressions. <Expression1> et <Expression2> must have the same type.

Example :	`a%=10`
	`b%=5`
	`c%=a%+b%    'Result : c%=15`

See also :	`-`, `*' and `/'.

10-16-2- Subtraction (-)

Syntax :	<Expression1> - <Expression2>
Accepted types :	Byte, Integer, Long integer or real
Description :	this operator subtract <Expression2> from <Expression1> and return a value type identical as its operand.
Remarks :	<Expression1> and <Expression2> must be numerical valid expressions. <Expression1> and <Expression2> must have the same type.

Example :	`a%=10`
	`b%=5`
	`c%=a%-b%    'Result : c%=5`

See also :	'+', `*' et `/'.

10-16-3- Multiplication (*)

Syntax :	<Expression1> * <Expression2>
Accepted types :	Byte, Integer, Long integer or real
Description :	This operator multiply <Expression1> by <Expression2> and return a value type identical as its operand.
Remarks :	<Expression1> and <Expression2> must be numerical valid expressions. <Expression1> et <Expression2> must have the same type.

Example :	`a%=10`
	`b%=5`

```
c%=a%*b%   'Result : c%=50
```

See also : '+', '-' and `/'.

10-16-4- Division (/)

Syntax : <Expression1> / <Expression2>

Accepted types : Byte, Integer, Long integer or real

Description : This operator divide <Expression1> by <Expression2>

Remarks : <Expression1> and <Expression2> must be numerical valid expressions. <Expression1> et <Expression2> must have the same type. <Expression2> must be different of zero. This operator always return a real value.

Example :

```
a%=10
b%=5
c!=a%/b%   'Result : c!=2.0
```

See also : '+', '-', `*' and DIV.

10-16-5- Lower (<)

Syntax : <Expression1> < <Expression2>

Accepted types : Byte, Integer, Long integer, real or Char string

Description : This operator tests if <Expression1> is lower than <Expression2>.

Remarks : <Expression1> and <Expression2> must be numerical valid expressions. <Expression1> and <Expression2> must have the same type.

Example :

```
a%=10
IF b%<a% THEN ...
```

See also : '=', '>', '>=', '<=', '<>'.

10-16-6- Lower or equal (<=)

Syntax : <Expression1> <= <Expression2>

Accepted types : Byte, Integer, Long integer, real or Char string

Description : This operator tests if <Expression1> is lower or equal than <Expression2>.

Remarks : <Expression1> and <Expression2> must be numerical valid expressions. <Expression1> and <Expression2> must have the same type.

Example :

```
a%=10
IF b%<=a% THEN ...
```

See also : '=', '>', '>=', '<', '<>'.

10-16-7- Left shift (<<)

Syntax : <Expression1> << <Expression2>

Accepted types : Byte or Integer

Description : This operator shifts <Expression2> bits from <Expression1> from right to left.

Remarks : <Expression2> is the number of bits to shift. The shifting is not circular.

Example :

```
a%=100b
```

```
b% =a%<<2  'Result b%=10000b
```

See also : '>>'

10-16-8- Different (<>)

Syntax : <Expression1> <> <Expression2>

Accepted types : Byte, Integer, Long integer, real or Char string

Description : This operator tests if <Expression1> and <Expression2> are different.

Remarks : <Expression1> and <Expression2> must be numerical valid expressions. <Expression1> and <Expression2> must have the same type.

Example :

```
a%=10
IF b%<>a% THEN ...
```

See also : '=', '>', '>=', '<', '<='

10-16-9- Affect/Equal (=)

Syntax : <Expression1> = <Expression2> Or <Variable>=<Expression2>

Accepted types : Bit, Byte, Integer, Long integer, real or Char string

Description : this operator affects <Variable> to <Expression2> or tests if <Expression1> is equal to <Expression2>.

Remarks : <Expression1> and <Expression2> must be numerical valid expressions. <Expression1> and <Expression2> must have the same type.

Example :

```
a%=10
IF b%=5 THEN ...
```

See also : '>', '>=', '<', '<=', '<>'

10-16-10- Greater (>)

Syntax : <Expression1> > <Expression2>

Accepted types : Byte, Integer, Long integer, real or Char string

Description : this operator tests if <Expression1> is greater than <Expression2>.

Remarks : <Expression1> and <Expression2> must be numerical valid expressions. <Expression1> and <Expression2> must have the same type.

Example :

```
IF b%>a% THEN ...
```

See also : '=', '>=', '<', '<=', '<>'

10-16-11- Greater or equal (>=)Diff_rent

Syntax : <Expression1> >= <Expression2>

Accepted types : Byte, Integer, Long integer, real or Char string

Description : This operator tests if <Expression1> is greater or equal than <Expression2>.

Remarks : <Expression1> and <Expression2> must be numerical valid expressions. <Expression1> and <Expression2> must have the same type.

Example :

```
IF b%>=a% THEN ...
```

See also : '=', '>', '<', '<=', '<>'

10-16-12- Right shift (>>)

Syntax :	<Expression1> **>>** <Expression2>
Accepted types :	Byte or Integer
Description :	This operator shifts <Expression2> bits from <Expression1> from left to right.
Remarks :	<Expression2> is the number of bits to shift. The shifting is not circular.

Example :	`a%=11010b`
	`b% =a%>>2  'Result b%=110b`

See also :	'<<'

10-16-13- Exponent (^)

Syntax :	<Expression1> ^ <Expression2>
Accepted types :	Byte, Integer, Long integer or real
Description :	this operator raises <Expression1> to the <Expression2> power.
Example :	`a!=b!^2 ' a=b²`

10-16-14- ABS – Absolute value

Syntax :	**ABS** (<Expression>)
Accepted types :	Byte or Integer
Description :	This function provide the absolute value of <Expression>. A negative number is then converted in a positive number.
Remarks :	<Expression> must be a valid numerical expression. The absolute value of a number is its no-signed value.

Example :	`a%=ABS(-100)      'Result : a%=100`
	`a%=ABS(25) 'Result : a%=25`

10-16-15- ABSCAM – Absolute electronic camebox

Syntax : **ABSCAM** (<Slave>,<Master>, <Table>, <Mono>, <Reverse>, <PositiveDirection>)

Description : This function launch an electronic absolute camebox between two axes.

Remarks : <Slave> : Name of the slave axis where is done the came (servo module: SRV15, SRV85...)

<Master> : Name of the master axis (servo module or encodeur module: SCD22, SRV15, SRV85...)

<Table> : Name of the camebox table given by the data's editor in global variables's tab of MCB software (variable of type « cambox's table »). The values given to the master axis position must be growing.

<Mono> : Define the automatic relooping of the came. Enter the value 0 for a cam who reloops on his profil always until a stop will send, enter 1 for a cam who execute his profile one time.

<Reverse> : Indicate if the slave must follow the master in the two way.

↳ Enter the value 0 for a not reversible cam: the mastter move in the other sens given by <PositiveDirection>, the slave stop et restart when the master go ing in the good sense and reaching the master position where the slave was stop.

	➯ Enter the value 1 for a reversible cam: The slave follow his cam's profil ignorant the master' sense.
	<PositiveDirection> : If the came is not reversilbe, the normal sens of the master must be indicate. Enter 0 for negative sense or 1 for a positve sense.

Exemple :	`ABSCAM(Master,Slave,Table,0,1,1)  ' Non mono-coup, reversible,`
	 `' sens positif, avec un`

See also :	CAMBOX , GEARBOX, MOVS, CAM, CAMC

10-16-16- ACC - Acceleration

Syntax 1 :	**ACC**(<Axis>) = <Expression>
Syntax 2 :	<Variable> = **ACC**(<Axis>)
Syntax 3 :	**ACC**(<AxisX>,<AxisY>) = <Expression>
Units :	Expression, Variable : user unit per s² (Ex : mm/s², degré/s², round/s²…)
Limits :	Expression, Variable : 5.10^-5 to 1,5.10^6 pulses/s
Accepted types :	<Expression> : Byte, Integer, Long integer or real
	<Variable> : real
Description :	This instruction reads or writes the current acceleration.
Remarks :	<Expression> must be a real valid expression. The current acceleration can be read or write at any time. The third form is used for interpolation.
Example :	`ACC(X)=500`
See also :	DEC, POS and VEL

10-16-17- ACC% - Acceleration in percent

Syntax :	**ACC%**(<Axis>) = <Expression>
Limits :	Expression : 1 to 100
Accepted types :	<Expression> : Byte, Integer, Long integer or real
Description :	This function sets current acceleration in percent of acceleration parameter (ACC_P).
Remarks :	The value of ACC_P can be set in the screen "Speed Profile" during the board configuration.

Example :	`ACC_P(X)=500`
	 `ACC%(X)=10 'The current acceleration is 50 units / s²`

See also :	DEC%

10-16-18- ADC – Analogue input

Syntax :	<Variable>= **ADC** (<AnalogInput>)
Units :	Variable : Volt
Limits :	Variable : +/- 10V
Accepted types :	<Variable> : real
Description :	This function returns the voltage of an analogue input.

Remarks :	<AnalogInput> must represent the name of an analogue input.
Example :	`a!=ADC(Temperature)`
See also :	DAC

10-16-19- ADDMOV – Superposition of movements

Syntax :	**ADDMOV** (<Slave>, <Master>, <Coefficient>)
Units :	<Coefficient> : pulse factor between the master and the slave
Accepted types :	<Coefficient> : real
Description :	this function allow the superposition of the master axis movements on the slave axis movements.(only SRV85)
Remarks :	<Slave> and <Master> must be servo axis. The master and the slave can execute any type of movements like positioning, synchronization or interpolation.
See also :	ADDSTOP

10-16-20- ADDSTOP – Stop the superposition of movements

Syntax :	**ADDSTOP** (<Slave Axis>)
Description :	This function cuts off the link of movements superposition of the slave axis.(only SRV85)
Remarks :	<Slave Axis> must be a servo axis. If the slave axis passed in open loop mode (AXIS(Slave)=Off), the link between the master and the slave axis is cutting off.
See also :	ADDMOV

10-16-21- AND – Operator AND

Syntax :	<Expression1> **AND** <Expression2>
Accepted types :	Bit, Byte or integer
Description :	This function makes a binary AND between two expressions and return a value type identical as its operands.
Remarks :	<Expression1> and <Expression2> must have the same type.
Example :	`IF (A% AND 0FF00h)<>0 THEN ...`
See also :	OR, NOT, XOR and IF

10-16-22- ARCCOS – Invert cosine

Syntax :	**ARCCOS** (<Expression>)
Limits :	–1 to +1
Accepted types :	Byte, Integer, Long integer, real
Description :	This function returns the arccosine of <Expression>.
Remarks :	<Expression> must be a numerical valid expression. This function returns an angle expressed in radians.
Example :	`pi!=2*ARCCOS(0)`
See also :	SIN, COS and TAN

10-16-23- ARCSIN – Invert Sine

Syntax :	**ARCSIN** (<Expression>)
Limits :	-1 to +1
Accepted types :	Byte, Integer, Long integer, real
Description :	This function returns the arcsine of <Expression>.
Remarks :	<Expression> must be a numerical valid expression. This function returns an angle expressed in radians.
Example :	`pi!=2*ARCSIN(1)`
See also :	SIN, COS and TAN

10-16-24- ARCTAN – Invert tangent

Syntax :	**ARCTAN** (<Expression>)
Accepted types :	Byte, Integer, Long integer, real
Description :	This function returns the arctangent of <Expression>.
Remarks :	<Expression> must be a numerical valid expression. the function ARCTAN takes the ratio of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side opposite the angle divides by the length of the side adjacent to the angle.

Example :	`a!=ARCTAN(3)`
	`pi!=4*ARCTAN(1)`

See also :	SIN, COS and TAN

10-16-25- ASC – Convert char to ASCII

Syntax :	**ASC**(<String>)
Accepted types :	Char string
Description :	This function returns a numeric value which is the ASCII code for the first character of a string expression.
Remarks :	If the string length <String> is zero, the zero value is returned.

Example :	`a$="A"`
	`b#=ASC(a$) 'Result : b#=65`

See also :	CHR$.

10-16-26- AXIS – Axis loop control

Syntax : **AXIS** (<Axis>) = ON | OFF

Description : This instruction is used to open and close axis loop.

Remarks : When the axis is in closed loop (AXIS=ON), all the movements instructions are transmitted to the axis card by the buffer of movements and executed. If the axis passed in an open loop mode (AXIS=OFF), the buffer of movements is cleared, analogique command is pulled to 0 and the instructions MOVE_S and FE_S return the value 0.

Warning :

➪ This instruction is effective after about 3ms, so for a good servo control :

```
AXIS(X)= ON
WAITAXIS_S(X)=ON
```

➯ All the instruction sent to an axis card, which is in an open loop mode, will be consume but not really executed by the axis card.

➯ If an axis is in a following error, all the axis driven by the MCS go in a open loop mode.

Example :

```
AXIS(X) = ON        ' Closed loop
MOVA(X=1000)        ' Movement to the position 1000
OUT(S1)=1           'Sortie S1=1
MOVA(X=2000)        'Following error detected on another axis
                    'If the position 2000 is not reached yet,
                    'the task continues and the S1 output
                    'pbecomes 0
OUT(S1)=0
```

See also : AXIS_S, SECURITY

10-16-27- AXIS_S – Axis loop state

Syntax : **AXIS_S**(<Axis>)

Description : This function permits to read axis loop state and return an ON or OFF state.

Remarks : This function can be used at any time to see if the axis loop is closed.

Example :

```
MOVA(X=100)
  If AXIS_S(X) = OFF THEN GOTO Error  'Error because the axis is
                                      'passed in a open loop
                                      'mode
OUT(InhibDrive) = Axis_s(X)     'This inhibition drive output
                                'must be used in a security task
```

See also : AXIS

10-16-28- AXISCONDITION – Configuration des conditions

Syntax: AXISCONDITION(<Axis>, <Num event>, <Condition>, <Valid state>)

Accepted types : <Num event> : Byte

<Condition> : Integer

<Valid state > : bit

Description: This instruction allow to define the starting conditions of each events.

<Axis> : Name of the servo or encoder axis

<Num event> : Number of the axis event (1 or 2)

<Condition> : The condition is define by a mask on 16 bits as :

- bit 0: comparison to execute for the first test
  - 0 : measure >= consigne
  - 1 : measure < consigne
- bit 1..3: number of the used measure for the first test (0..7)
- bit 4..5: number of the used consigne for the first test (0..3)
- bit 6: comparison to execute for the second test

0 : measure >= consigne

1 : measure < consigne

- bit 7..9: number of the used measure for the second test (0..7)
- bit 10..11: number of the used consigne for the second test (0..3)
- bit 12..13: Logic operation to execute between the first and second result:

0: PASS (résultat= résultat du premier test)

1: AND

2: OR

3: XOR

<active state> : define the state of the condition use to generate the event.

Exemples: Event if the sensor is ‘0’ during a distance who is between 200 and 400.

```
' consigne N°0= 200
AXISLEVEL(X, 0) = 200
' consigne N°1= 400
AXISLEVEL(X, 1) = 400
' testA : measure N°4 >= consigne N°0
' testB : measure N°4 < consigne N°1
' logic operation = AND
‘ valide state =1
AXISCONDITION(X, 1, 01011000001001b, 1)
```

Event if the sensor is ‘0’ during a distance < 200 or >= 400.

```
' consigne N°0= 200
AXISLEVEL(X, 0) = 200
' consigne N°1= 400
AXISLEVEL(X, 1) = 400
' testA : measure N°4 < consigne N°0
' testB : measure N°0 >= consigne N°1
' opération logique= OR
‘ etat valide =1
AXISCONDITION(X, 1, 10010001001000b, 1)
```

Event when the distance running with the sensor to 1 is >= 400.

```
' consigne N°0= 400
AXISLEVEL(X, 0) = 400
' testA : measure N°1 >= consigne N°0
' logic operation = PASS
‘ valide state =1
AXISCONDITION(X, 1, 00000000000011b, 1)
```

Event if the total running with the sensor to 1 is <200.

```
' consigne N°0= 200
AXISLEVEL(X, 0) = 200
' testA= measure N°5 < consigne N°0
' logic operation= PASS
‘ valid state =1
AXISCONDITION(X, 1, 00000000001010b, 1)
```

10-16-29- AXISEVENT – Events configuration or read events out of factual tasks

a) Events configuration :

Syntax: AXISEVENT(<Axis>, <Active>, <Time maesure>, <Sensor 2>, [<Desactivate filtering >, <Inversion>])

Accepted types: <Actif>, <Mesure Temps>, <Cellule 2>, <Filtrage>,
<Inversion> : Bit

Description: This instruction allow to define the physic configuration necessary to the measurement and events generation.

<Axis> : Name of the servo or encoder modulo

<Active> : Activation of axis events

<Maesuring time> : to 1 maesure in time, to 0 maesure in distance

<Sensor 2> : à 1 mesure sur C2, à 0 mesure sur C1

<Desactivate filtering> : to 0 filtering input at 2 ms, to 1 not filtering input (0,1µs)

<Inversion> : to 1 negative logic, à 0 positive logic

Remarks: The filtering and inversion parameters are optionnals. It’s necessary to use them if the card input was not previously parametering by a capture instruction. When this parameters are forgotten, the input is not configured. Attention, the HOME instruction delete automaticaly the inversion and filtering.

Exemple: AXISEVENT(X,True,False,False) ' Measuring event on sensor in distance

b) Reading events out of factual tasks :

Syntax : <Variable> = AXISEVENT(<Axis>, <Event number>

Accepted types: <Variable> : Bit, Byte, Integer, Long integer, Real.
<Event number> : Byte

Description : This function allow to accomplish and read the events who have been detecte on inputs without use event task.

Exemple :

```
If AXISEVENT(X,1) Then
        ‘…
        ‘…
End If
```

10-16-30- AXISLEVEL – Instructions definition

Syntax: AXISLEVEL(<Axis>, <Instruction number>) = <Instruction>

Accepted types: <Instruction number> : Byte

	<Consigne> : Long interger
Description:	This function allow to define the instructions. <Axis> : Name of the servo or encoder modulo <Instruction number> : number of the instruction to configurate (0 to 4) <Instruction> : distance ou duration.
Remarks:	The number instructions 0 to 3 square to used instructions for the tests. The instruction number 4 square to filling up.
Exemple:	AXISLEVEL(X,0)=400

10-16-31- AXISMEASURE – Read distances and timings

Syntax :	AXISMEASURE(<Axis>,<Event num>,<N0>,<N1>,<N4>,<N5>)
Accepted types:	<Event num> : Byte <N0>, <N1>, <N4>, <N5> : Long integer local variables
Description :	This function allow to read running values or values capture by measure.
Remarks :	The reading values square either to running value (Event num = 0) whether captured values of each event (Event num =1 or 2).

10-16-32- BACKLASH - Taking up of play

Syntax :	BACKLASH(<Axis>)=<Expression>
Accepted types :	Expression : Real
Description :	After to have launch this instruction, on any axis's way change, the system add on PlayValue to moving. The value of axis real position returned by POS_S(Axis) will ignore this play.

Exemple :

```
' init
Jeu=0.05
Backlash(X)=0
Movr(X=Jeu*2)        ' mouvement sens + assurant le rattrapage du jeu
Backlash(X)=Jeu      ' activation de la fonction rattrapage de jeu
                     ' (jeu de 5/100 mm) => toute inversion de déplacement
                     ' integrera un rattrapage de jeu.
' Cycle
Mova(X=20)           ' le jeu n'est pas pris en compte car pas d'inversion
                     ' de sens(toujours sens + )
Mova(X=5)            ' le jeu est pris en compte : la mécanique parcourt
                     ' 20-5+Jeu=15.05mm
Position!=Pos_s(X)   ' la position retournée par la système est égale à 5 ' et
                     non 4.95
Mova(X=0)            ' pas de rattrapage
Movr(X=50)           ' rattrapage
...
```

10-16-33- BACKLIGHT – Dilaog 640 in stand by

Syntax :	**BACKLIGHT**=<duration>
Units :	duration : minutes
Accepted types :	duration : Integer
Description :	this function defines the duration in minute during the backlight of Dialog 640 will stay active if any of the key panel are pressed.
Remarks :	When backlight is switched off, if a key panel is pressed, the backlight is switched on. The default duration is equal to 15 minutes.

Duration : 0→backlight switch off

1→backlight always switch-on.

(Duration>1) →delay in minutes.

' Warning : The backlight life duration is about 10 000 hours.

Example :

```
BACKLIGHT=120      'the backlight Dialog 640 will be switch off
                   'if a key panel is not pressed by the users
                   ' in the two hours last
```

10-16-34- BEEP – Brief sound

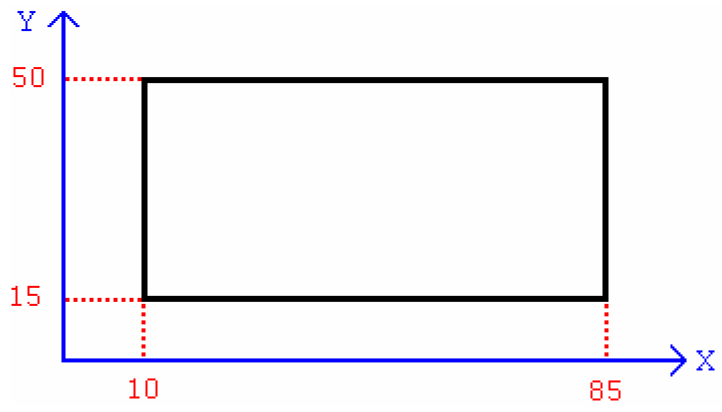
Syntax :	**BEEP**
Description :	this instruction emits a brief sound on the operator panel dialog 80, 160 or 640.
Remarks :	This function uses the communication port #1. By default, the communication port SERIAL1 will be used. If an operator panel Dialog 80, 160 or 640 is connected to SERIAL2, please use the OPEN function to affect #1 to the port SERIAL2.
Example :	`IF KEY<>@ENTER THEN BEEP`
Voir aussi :	BUZZER

10-16-35- BOX – Draw box

Syntax :	**BOX**(X1,Y1,X2,Y2,BorderColour, FillColour)
Units :	X1, Y1, X2, Y2 : pixel
Limits :	X1, X2 : 1 to 240
	Y1, Y2 : 1 to 128
Accepted types :	X1, Y1, X2, Y2, FillColour : Byte
	BorderColour : Bit
Description :	This instruction draws a box with the coordinates X1,Y1 (top left corner) and X2,Y2 (down right corner) on the operator panel Dialog 640.
Remarks :	The BorderColour parameter defines the colour of the border : black (0) or white (1). The FillColour defines the colour of the filling : black (0), white (1) or transparent (2).

Example :

```
BOX(10,50,85,15,0,1)      'Black border with white fil colour
```



10-16-36- BUFMOV_S – Number of waiting orders

Syntax : <Variable>=**BUFMOV_S**(<Axis>)

Accepted types : <Variable> : Byte

Description : this function returns the number of waiting movements in the axis board buffer. The current movement of the axis board is not contained in the result of this function.

Remarks : This function can be used after starting movements, to know if a movement is finished. If the buffer of movements is full, the task is blocked and continue as soon as the buffer is not full.

Example :

```
STTR(X)=100
STTR(X)=50
STTR(X)=50
WAIT BUFMOV_S(X)<2        'Waiting the end of the first movement
```

10-16-37- BUZZER – Continuous sound

Syntax : **BUZZER** = <ON|OFF>

Description : This function activates or desactivates the buzzer of the operator panel dialog 80, 160 or 640.

Remarks : This function uses the communication port #1. By default, the communication port SERIAL1 will be used. If an operator panel Dialog 80, 160 or 640 is connected to SERIAL2, please use the OPEN function to affect #1 to the port SERIAL2.

Example :

```
Alarme:
    BUZZER=ON
    DELAY 1000
    BUZZER=OFF
    DELAY 1000
    GOTO Alarme
```

See also : BEEP

10-16-38- CALL – Subroutine call

Syntax : **CALL** <Name>

Description : This instruction calls a subroutine define by SUB block. <Name> is the block name of the subroutine.

Remarks : A subroutine can't call itself. System contains some predefined subs : _MENUMCS, _PARAMMCS, _MANUMCS, _VARIABMCS, _MEMORYMCS and _CLOCKMCS. The execution of this instruction launches the execution of the next task.

Example : `CALL Move`

See also : SUB, ICALL

10-16-39- CAM – Cam type synchronization

Syntax : CAM (<Slave Axis>,<Master Axis>, <Cam table>,<Mono>,<Reverse>, <PositiveDirection>,<Factor>)

Description : This function starts cam type synchronization between two axis.

Remarks : <Slave> : Slave axis name, on which is applied the cam (servo board : SRV15, SRV85...)

<Master> : Master axis name (servo or encoder board : SCD22, SRV15, SRV85 ...)

<Table> : Cam table name, set from the data editor (variable type 'cam table').

<Factor> is used to multiply all the slave positions defined in the table with a coefficient.

<PositiveDirection> is a slave direction. Input 0 for a negative direction, 1 for a positive one.

<Reverse> is a reversible axis :

➯ Input 0 for a non-reversible cam: if the master moves in the opposite way as the one defined in <PositiveDirection>, the slave stops. It will start off again when the master will go in the right way and pass by the position where the slave stopped.

➯ Input 1 for a reversible cam: The slave follows its cam profile whatever is the master direction.

Example :

```
CAM(Master,Slave,Table,0,1,1,2.5)  ' reversible,
                                   'positive direction, with a
                                   ' Factor 2.5
```

See also : CAMBOX, GEARBOX,, MOVS, CAMC

10-16-40- CAMBOX – Camboxes

Syntax : **CAMBOX** (<Number>,<Master>, <OutputCard>, <Seg Nb>, <Delay Nb>)

Limits : Number : 1 to 8

Seg Nb : 1 to 16

Accepted types : Number : Byte

Seg Nb : Byte

Delay Nb : Byte

Description : This function defines a cambox. All the segments defined before are deleted (CAMBOXSEG).

Remarks : <Number> is the Number of the cambox

<Master> can be a servo axis, an encoder or a time based encoder.

<OutputCard> is the name of a 8 or 16 outputs board.If you give 0 to the name's card, then the box stimulate a variable of integer type.

<Seg Nb> is the maximal number of segments in the cambox. If this value is zero, then the cambox is destroyed and must be defined one more time to be used.

<Delay Nb> is the number of anticipation phases.

Example : `CAMBOX(1,Maitre,S,10,0)  ' 10 segments cambox on S card outputs`

See also : CAMBOXSEG, CAMBOXDELAY, CAMBOXVAR

10-16-41- CAMBOXDELAY – Anticipation delay

Syntax : **CAMBOXDELAY** (<Number>, <Delay Num>,<Velocity>, <Delay>)

Limits : Number : 1 to 8

Delay Num : 1 to 5

Units : Velocity : in user units per second (Ex : mm/s, degree/s…)

Delay : in milliseconds

Accepted types : Number, Delay Num : Byte

Velocity : real

Delay : Long integer

Description : This function defines an anticipation delay.

Remarks : <Number> is cambox number.

<Delay Num> is the delay number.

Example : `CAMBOXDELAY(1,2,100,10)  'A 10 ms delay is up to 100 units/s`

See also : CAMBOX

10-16-42- CAMBOXSEG – Cambox segment

Syntax : **CAMBOXSEG** (<Number>, <Num seg>, <Num Output>, <Begin>,<End>)

Limits : Number : 1 to 8

Num Output : 1 to 16

Units : Begin, End : user unit

Accepted types : Number, Num seg, Num Output : Byte

Begin, End : real

Description : This function defines a cambox segment.

Remarks : <Number> is cambox number. <Num seg> is the segment number. <Num Output> is the number of the modified output or number of bit. The output is set between <Begin> and <End>.

Example :
```
CAMBOXSEG(1,2,4,0,90)    'The second segment of the cambox 1 set
                     ' the fourth output between 0 and 90°(user
                     'unit defined in degree).
```

See also : CAMBOX

10-16-43- CAMBOXVAR – Cambox with contact on variable

Syntax 1: CAMBOXVAR(<Numéro boîte>)=<Expression>

Syntax 2: <Variable>=CAMBOXVAR(<Box number>)

Accepted types : Variable, Expression : Integer.

Description : This function allow to initializate the value used by de cambox or read his status.

Remark : It 's possible to change the value of this variable in a programme if the boxcam is stop ? If the box is already start, it was not possible to change one bit wtih COMBOXVAR.

Exemple :

```
CAMBOX(1,Maitre,0,2,0) 'Boîte à cames de 2 segments sur affectation variable

CAMBOXSEG(1,1,3,10,30) ' Le segment 1 de la boîte 1 met le bit numéro 3 à 1
entre 10 et 30°

CAMBOXSEG(1,2,7,90,180) ' Le segment 2 de la boîte 1 met le bit numéro 7 à 1
entre 90 et 180°

CAMBOXVAR(1)=0 ' initialisation de la variable

STARTCAMBOX(1) ' démarrage de la boîte

...

A%=CAMBOXVAR(1) ' lecture état de la variable

IF A%.3=1 Then ……   ' test du bit 3 de la variable

...
```

See also : CAMBOX , CAMBOXSEG, STARTCAMBOX

10-16-44- CAMC – electronic cambox triggered on capture input

Syntax : CAMC (<Slave Axis>,<Master Axis>,<Cam table>,<Mono>,<Reverse>,
<PositiveDirection>,<Factor>,<Configuration>
[,<TriggeredAxis>,<Window>,<Mini>,<Maxi>,<Inside>])

Accepted types : <Configuration> : Hardware configuration byte of <Axis>

b0 : unused

b1 : capture on Z signal

b2 : capture on C1 input

b3 : capture on C2 input

b4 : choice of edge : 0 for a positive edge, 1 for negative edge

b5..7 : unused

<PositiveDirection>, <Factor>, <Mini>, <Maxi> : real

<Cam table> : camtable

<Window>, <Inside> : bit

Description : This instruction is the same like CAM but it has a triggered condition on a fastest capture input. The triggered can be on a position window. The parameters <TriggeredAxis>, <Window>, <Mini>, <Maxi> and <Inside> are optional.

Remarks : <Master Axis> can be a servo axis, an encoder or a time based encoder.

<Slave Axis> must be a servo axis.

<Slave Axis> must be stopped (MOVE_S(Slave Axis)=0) before the instruction is send because the movement can be obtained even if the condition is not valid.

See also : CAMBOX, GEARBOX,, MOVS, CAM

10-16-45- CAMNUM_S – Number of the running cam

Syntax :	<Variable>=**CAMNUM_S**(<Slave>)
Accepted types :	<Variable> : Integer
Description :	this instruction permits to know the number of the running cam.
Remarks :	<Slave> : Slave axis cam for which the cam will be applied (servo board: SRV15, SRV85...) The returned value has got a sense only if CAM_S is set.

Example :

```
IF CAMNUM_S(Slave)=1 THEN PRINT «  Cam 1 running  »
IF CAMNUM_S(Slave)=2 THEN PRINT «  Cam 2 running  »
```

See also : CAM_S, CAMSEG_S

10-16-46- CAMFROMPOINT – Slave position in the cam

Syntax : <Slave position>=CAMFROMPOINT(<Master position>,
<Table>,<Number>)

Accepted types : <Number> : Integer

<Table> : real table

Description : This intruction allow to calculate the slave positon <Slave position> in the cam, corresponding to the master position <master position>.

Remarks : <Table> : Name of the table declared in the globale variable thumb of the MCB software (Type's variable « real »).

<Number> : size of the table.

Number = (CamPointNumber + 2) *2

10-16-47- CAMSEG_S – Equation number of the running cam

Syntax :	<Variable>=**CAMSEG_S**(<Slave>)
Accepted types :	<Variable> : Integer
Description :	this instruction permits to know which equation number of the cam is running.
Remarks :	<Slave> : Slave axis cam for which the cam will be applied (servo board: SRV15, SRV85...) The returned value has got a sense only if CAM_S is set.

Example :

```
IF CAMSEG_S(Slave)=1 THEN PRINT «  Cam between point 1
and point 2  »
IF CAMSEG_S(Slave)=2 THEN PRINT «  Came between point 2
and point 3  »
```

See also : CAM_S, CAMNUM_S

10-16-48- CAM_S – State of the cam

Syntax :	<Variable>=**CAM_S**(<Slave>)
Accepted types :	<Variable> : Bit
Description :	this instruction permits to know if a cam from the board is running.
Remarks :	<Slave> : Slave axis cam for which the cam will be applied (servo board: SRV15, SRV85...)

Example : `IF NOT CAM_S(Slave) THEN PRINT « Stopped cam »`
`IF CAM_S(Slave) THEN PRINT « Running cam »`

See also : CAMNUM_S, CAMSEG_S

10-16-49- CAPTURE – Position capture

Syntax : **CAPTURE** (<Axis>,<Invert>,<Window>,<Mini>,<Maxi>,<Inside>)

Description : This function activates the position registration .

Remarks : With this instruction, MCS waits a positive edge on the registration input. When the edge is detected, the position is stored in the REGPOS_S variable. REG_S is also set. <Inversion> can change the type of the edge. If <Window> is true, then the capture input is examined only when <Axis> is between <Mini> and <Maxi> position.

<Inside> defines if the test is inside or outside <Mini> and <Maxi> position. <Mini> must be always lower than <Maxi>.

Example :

```
CAPTURE(X,Off,On,10,20,On)     'Position capture on positive
                               'edge when X is between 10 and 20
WAIT REG_S(X)=True
MOVA(Y=500+REGPOS_S(X))
```

See also : REGPOS_S, REG_S, CAPTURE1, CAPTURE2

10-16-50- CAPTURE1 – Position capture

Syntax : CAPTURE1(<Axe1>,<Axis2>,<Configuration1>,<Fenêtre2>,<Mini2>,
<Maxi2>,<Inside2>)

Accepted types : <Configuration> : Hardware configuration byte of <Axis>

b0 : unused

b1 : capture on Z signal

b2 : capture on C1 input

b3 : capture on C2 input

b4 : choice of edge : 0 for a positive edge, 1 for negative edge

b5..7 : unused

<Mini2>, <Maxi2> : real

<Window2>, <Inside2> : bit

Description : This instruction stores the position of any SRV85 board on an edge of one of its two capture inputs.

Remarks : With this instruction, MCS waits an edge on its capture input. When an edge is detected, the position is stored in the REGPOS1_S variable. REG1_S is also set. If <Window2> is true, then the input is examined when <Axis2> is between <Mini2> and <Maxi2> position.

<Interieur2> define if the test is inside or outside <Mini2> and <Maxi2> position. <Mini2> must be always lower than <Maxi2>.

Example :

```
CAPTURE1(Y,X,4,On,10,20,On)    'Capture deportated of the
                        'position on X axis of
                               'C1 input positive edge of Axis Y
                               'when axis X is between 10 and 20
```

```
WAIT REG1_S(Y)=True
Correction  !=Phase-REGPOS1_S(Y,X)
...
```

See also : REGPOS_S, REG_S, CAPTURE, CAPTURE2

10-16-51- CAPTURE2 – Position capture

Syntax : CAPTURE2(<Axis>,<Configuration>,<Window>,<Mini>,<Maxi>,<Inside>)

Accepted types : <Configuration> : Hardware configuration byte of <Axis>

b0 : unused

b1 : capture on Z signal

b2 : capture on C1 input

b3 : capture on C2 input

b4 : choice of edge : 0 for a positive edge, 1 for negative edge

b5..7 : unused

<Mini>, <Maxi> : real

<Window>, <Inside> : bit

Description : This instruction has the same functionality as CAPTURE but the position capture can be made on C1 or C2 capture input or the Z signal of the encoder.

Remarks : With this instruction, MCS waits a capture edge. When the edge is detected, the position is stored in the REGPOS2_ S variable. REG2_S is also set. If <Window> is true, then the input is examined when <Axis> is between <Mini> and <Maxi> positions.

<Inside> defines if the test is inside or outside <Mini> and <Maxi> position. <Mini> must be always lower than <Maxi>.

Example :

```
CAPTURE2(X,24,Off,0,0,Off)      ' Position capture of axis X on
                                ' the negative edge of the C2
                                ' capture input of axis X
WAIT REG2_S(X)=True
Correction!=Phase-REGPOS2_S(X)
...
```

See also : REGPOS_S, REG_S, CAPTURE, CAPTURE1

10-16-52- CASE – Multiple tests

Syntax 1 : **CASE** <Expression> CALL <Label 1> [{ , <Label2> }]

Syntax 2 : **CASE** <Expression> GOTO <Label 1> [{ , <Label2> }]

Accepted types : Expression : Integer

Description : This function allows to make jumps to labels in function of <Expression> values.

Remarks : <Expression> must be an integer valid value. If the Expression value is equal to zero or greater than the number of labels, the task goes on at the next line. The execution of this instruction launches the execution of the next task.

Example :

```
Case  a% GOTO Move1, Move2
Goto Fin    'a%=0 or a%>2
...
Move1:      'a% = 1
```

```
...
Move2:     'a% = 2
...
Fin:
```

10-16-53- CARIN – Input buffer state

Syntax : **CARIN** (<Number>)

Description : This function returns the number of characters in the input buffer of the communication port.

Remarks : <Number> is the number used to open the communication port with OPEN instruction. This function returns an integer.

Example :

```
WAIT CARIN(#1)>=3 ' Wait for at least 3 received characters
A$=Input$ #1,3    ' Read 3 characters
```

See also : CAROUT, CLEARIN

10-16-54- CAROUT – Output buffer state

Syntax : <Expression>=**CAROUT** (<Number>)

Accepted types : <Expression> : integer

Description : This function returns the number of characters in the output buffer of the communication port.

Remarks : <Number> is the number used to open the communication port with OPEN instruction.

Example :

```
WAIT CAROUT(#1)<10       'Waits for place in the buffer
Print A$;                'Characters writing
```

See also : CARIN, CLEAROUT

10-16-55- CHR$ - Convert ASCII to char

Syntax : **CHR$**(<Code>)

Accepted types : Code : Byte

Description : This function returns a one character-string whose ASCII code is the argument.

Example :

```
a#=97
b$=CHR$(a#)       'Result : b$="a"
```

See also : ASASC C

10-16-56- CLEAR – Clear the axis position

Syntax : **CLEAR** (<Axis>)

Description : This instruction clears the axis position.

Remarks : <Axis> can be a servo axis or an encoder.

Example :

```
CLEAR(X)
A!=POS_S(X)       'Result : A!=0.0
```

10-16-57- CLEARCOUNTER – Counter clear

Syntax :	**CLEARCOUNTER**(<Axis>,<Input>)
Accepted types :	<Input> : Byte
Description :	This instruction initialize the counter to zero.
Remarks :	<Axis> : Servo card name
	<Input> : Input number (1 for C1 input, 2 for C2 input)
See also :	COUNTER_S, SETUPCOUNTER

10-16-58- CLEARFLASH – Clear flash memory

Syntax :	**CLEARFLASH**
Description :	This function clears parameters and the first 10000 safe variables in the flash memory.
See also :	RAMOK, FLASHOK, FLASHTORAM

10-16-59- CLEARFIFO – Clear the list of state change

Syntax:	**CLEARFIFO**(<Axis)
Description:	This function allow to clear the state change list of the input.

10-16-60- CLEARIN – Clear input buffer

Syntax : **CLEARIN** <Number>

Description : This instruction suppresses all the characters in the input buffer of the communication port.

Remarks : <Number> is the number used to open the communication port with OPEN instruction.

Example :

```
CLEARIN #1
Wait CARIN (#1)>=3       'Wait for at least 3 characters
A$=Input$ #1,3           'Read 3 characters
```

See also : CARIN

10-16-61- CLEAROUT – Clear output buffer

Syntax : **CLEAROUT** <Number>

Description : This instruction suppresses all the characters in the output buffer of the communication port.

Remarks : <Number> is the number used to open the communication port with OPEN instruction.

Example :

```
CLEAROUT #1
Print A$;          'Write the characters
```

See also : CAROUT

10-16-62- CLOSE – Close communication port

Syntax :	**CLOSE** #Number
Description :	The number argument is the number used in the OPEN instruction to open the communication port.
Remarks :	If you want to change the communication mode, you must close and open once again the communication port.
Example :	`CLOSE #1`
See also :	OPEN, INPUT and PRINT.

10-16-63- CLS – Clear screen

Syntax :	**CLS**
	CLS 1, CLS 2, CLS 3 or CLS 4 (only with Dialog 160 and 80)
	CLS B, CLS W (only with Dialog 640)
Description :	CLS clears the four lines of the operator panel screen. CLS 1, CLS 2, CLS 3, CLS 4 clears respectively the first, second, third and fourth line of the operator panel DIALOG 160 and 80 screen. The function CLS B clears the screen of the Dialog 640 with a black background. The function CLS W clears the screen of the Dialog 640 with a white background
Remarks :	This function uses the communication port #1. By default, the communication port SERIAL1 will be used. If an operator panel is connected to the port SERIAL2, please use OPEN function to affect #1 to the port SERIAL2.

10-16-64- CONS – Command voltage

Syntax :	**CONS**(<Axis>)=<Expression>
Units :	Expression : Volt
Limits :	Expression : –10 to +10
Accepted types :	Expression : real
Description :	This function opens axis loop and set command voltage to a particular value.
Remarks :	Be careful, in this case, the following error is no more controlled. <Axis> must be a servo axis. This instruction is useful to test the drive and the motor. <Expression> represents a voltage between -10 and +10V.
Example :	`CONS(X)=5.0        '5V value`
See also :	CONS_S

10-16-65- CONS_S – Command voltage state

Syntax :	**CONS_S**(<Axis>)
Description :	This function returnsthe image of command voltage.
Remarks :	<Axis> must be a servo axis. This function is used to know the current value of the command voltage.
Example :	`a!=CONS_S(X)`
	`PRINT  #1,a!`
See also :	CONS

10-16-66- CONTINUE – Continue task execution

Syntax : **CONTINUE** <Name>

Description : This instruction is used to continue the execution of a suspended task.

Remarks : <Name> must be the name of a suspended task. This function has no effect on the stopped or executed task.

Example :

```
Wait Inp(Start)
RUN Coupe
Begin:
  Wait Inp(Stop)
  SUSPEND Coupe
  Wait Inp(Start)
  CONTINUE Coupe
Goto Begin
```

See also : RUN, HALT, SUSPEND

10-16-67- CORRECTION – Function of correction

Syntax : **CORRECTION**(<Slave>,<Master>,<Mode>,<Capture>,<Configuration>,
<Window>,<Mini>,<Maxi>,<Inside>,<Phase>,
<Master Dist.>,<Maxi Slave vel.>,<Mini Slave vel.> ,
<Dist.accel.mini>,[<WayCorrection>])

Units : <Master Dist. > : user units
<Maxi Slave vel.>, <Mini Slave vel.> : user unit/s
<Dist.accel.mini> : user unit/s²

Accepted types : <Mode>, <Capture>, <Window>, <Inside> : bit
<Configuration> : Hardware configuration byte of <Axis>
b0 : unused
b1 : capture on Z signal
b2 : capture on C1 input
b3 : capture on C2 input
b4 : choice of edge : 0 for a positive edge, 1 for negative edge
b5..7 : unused
<Phase>, <Maxi>, <Mini>, <Master Dist.>, <Maxi Slave vel.>,
<Mini Slave vel.>,<Dist.accel.mini> : real

Description : This function realizes a correction movement on a slave axis during the distance of the master axis on an event of a capture position.(SRV85 only)

Remarks : <Slave> must be a servo axis (SRV85) to realize the correction and <Master> can be a servo axis or an encoder. Mode defines the single shoot mode (0) or the continuous mode (1). Capture defines the capture register : register 1 of the master (0) or register 1 of the slave (1). Window indicates if the correction must be realized in a window. Maxi must be greater than Mini. Inside defines if the correction is inside(1) or outside(0) the window. Phase defines if the capture position is equal to this phase, no correction will be realized. Because the parameters are stored in the slave axis, when the event is detected, the correction is instantaneously realized. <Master Dist.> is the interval of correction. <Maxi Slave vel.> is the maximal velocity during the correction. <Mini Slave vel.> is the minimal Velocity during the correction.

<Dist.accel.mini> defines the interval of acceleration and deceleration of the slave axis on the master distance. If CORRECTION is already true (in continuous mode), this instruction can be executed again in a real-time to change the parameters (for example the phase). But, it is necessary to indicate all the parameters even if they don't change.

<WayCorrection> : type byte :

if parameter = 0 or not use => the compensation applicate will be negative or positive (more short)

if parameter = 1 => the compensation applicate will be positive

if parameter = 2 => the compensation applicatewill be negative

Parameters Vmaxi and Vmini act as :

Max slave speed with correction = (RatioMasterSlave+SlaveMaxSpeed)*MasterSpeed

Min slave speed with correction = (RatioMasterSlave+SlaveMinSpeed)*MasterSpeed

RatioMasterSlave:

If gearbox with ratio ½ => RatioMasterSlave=0,5 (slave turn twice time less quick than master).

If gearbox with ratio 1 => RatioMasterSlave=1 (Slave & Master have the same speed).

Exemples :

We have a RatioMasterSlave = 1

MasterRunningSpeed = SlaverRunningSpeed = 600°/s

We want a max speed of slave with correction = 660°/s

We want a min speed of slave with correction = 480°/s

Max slave speed with correction = (RatioMasterSlave+SlaveMaxSpeed)*MasterSpeed => 660°=(1+ SlaverMaxSpeed)*600 => SlaveMaxSpeed = 0.1

Min slave speed with correction = (RatioMasterSlave+SlaveMinSpeed)*MasterSpeed
=> 480°=(1+ SlaverMinSpeed)*600 => SlaveMinSpeed = 0.2

CORRECTION(Slave,Master,Mode,Capture,Configuration,Windows,Mini,Maxi,Inside,
Phase,MasterDist,0.1,0.2, MinAccDist)

Warning :

➭ The last correction must be executed completely before this instruction will be executed again.

➭ If the master axis is in an opposite direction, the correction is not realized and is lost.

➭ The slave axis must be linked to the master axis by a gear box function (GEARBOX or GEARBOXM), a synchronized movement (MOVS, MOVSM, MOVSP, MOVSC) or a cambox function (CAM, CAMC) before the execution of the correction instruction.

➭ If the correction is out of the limits, the errors occurs and the maximal allowed correction is applied.

See also : STOPCORRECTION, ICORRECTION

10-16-68- CORRECTION_S – Correction state

Syntax :	<Variable>=**CORRECTION_S**(<Slave>)
Accepted types :	<Variable> : byte
	b0=1 : Wait state correction.

	b1=1 : Execution state correction.
	b2=1 : Error detected : Correction out of limits (The limit correction was applied).
Description :	This function gives the state of the correction cycle.
Remarks :	<Slave> must be a servo axis SRV85. In a closed loop mode, the returned value of CORRECTION_S is between « Wait state correction » and « execution state correction » (| b0=1 b1=0 | b0=0 b1=1 | b0=1 b1=0 |...).
	In single shoot mode, it will be : | b0=1 b1=0 | b0=0 b1=1 | b0=0 b1=0 |.
	The error bit b2 is automatically reset after the reading of CORRECTION_S or at the beginning of the next correction.
See also :	CORRECTION, ICORRECTION

10-16-69- CORRPOS_S– Value of the correction

Syntax :	<Expression>=CORRPOS_S(<Axis>)
Accepted type :	Expression : Real
Description :	This function return the value of applicate correction by the instruction Correction.
Exemple :	C !=CORRPOS_S(X)
See also :	CORRECTION, CORRECTION_S, STOPCORRECTION

10-16-70- COS - Cosine

Syntax :	**COS**(<Expression>)
Accepted types :	Expression : real
Description :	This instruction returns the cosine of the <Expression>.
Remarks :	The argument <Expression> must be a valid numerical expression expressed in radians. The function COS takes an angle and returns the two sides ratio of a rectangle triangle. The ratio is the length of the adjacent side divided by the length of the hypotenuse. The result is between -1 et 1.
Example :	a!=COS(3.14159)
See also :	SIN, ARCTAN et TAN

10-16-71- COUNTER_S – Counter reading

Syntax :	<Variable>=**COUNTER_S**(<Axis>,<Input>)
Accepted types :	<Variable> : Integer
	<Input> : Byte
Description :	This instruction reads the counter
Remarks :	<Axis> : Servo card name
	<Input> : Input number (1 for C1 input, 2 for C2 input)
See also :	SETUPCOUNTER, CLEARCOUNTER

10-16-72- CRC – CRC16

Syntax : CRC Value %=**CRC**(<Expression >)

Accepted types : Expression : Char string

Description : This function return the checksum value in a char string with the modbus RTU format (CRC 16).

Example : `A%=CRC(message$)`

10-16-73- CURSOR – Print or clear the cursor

Syntax : **CURSOR** = <ON | OFF>

Description : This function prints or not the cursor on the operator panel.

Remarks : This function uses the communication port #1.By default, the communication port SERIAL1 will be used. If an operator panel is connected to the SERIAL2 port, please refer to the OPEN function to affect #1 to the port SERIAL2.

10-16-74- CVL – Convert string to long integer

Syntax : <Variable>=**CVL**(<Expression>)

Accepted types : Variable : Long integer

Expression : string of 4 bytes

Description : The CVL function converts a string of 4 bytes, created with the MKL$ instruction, in a long integer value. The least significant word then the most significant word

Example :
```
A&=CVL(A$) 'If  A$=chr$(2)+chr$(3)+chr$(1)+chr$(0)
           'then  A&=2+(3*256)+(1*65536)+(0*16777216)=66306
```

See also : CVLR, MKL$, MKLR$

10-16-75- CVLR – Convert string to long reverse integer

Syntax : <Variable>=**CVLR**(<Expression>)

Accepted types : Variable : Long integer

Expression : string of 4 bytes

Description : The CVL function converts a string of 4 bytes, created with the MKL$ instruction, in a long integer value. The most significant word then the least significant word

Example :
```
A&=CVLR(A$) 'If  A$=chr$(0)+chr$(1)+chr$(3)+chr$(2)  then
            'A&=(0*16777216)+(1*65536)+(3*256)+(2*1)=66306
```

See also : CVL, MKL$, MKLR$

10-16-76- CVI – Convert string to integer

Syntax : <Variable>=**CVI**(<Expression>)

Accepted types : Variable : Integer

Expression : string of 2 bytes

Description : The CVL function converts a string of 2 bytes, created with the MKI$ instruction, in an integer value. The most significant byte then the least significant byte

Example :
```
A&=CVI(A$) 'If  A$=chr$(0)+chr$(1) then A&=0+(1*256)=256
```

See also : CVIR, MKI$, MKIR$

10-16-77- CVIR – Convert string to reverse integer

Syntax : <Variable>=**CVIR**(<Expression>)

Accepted types : Variable : Integer

Expression : string of 2 bytes

Description : The CVL function converts a string of 2 bytes, created with the MKI$ instruction, in an integer value. The least significant byte then the most significant byte

Example : `A%=CVIR(A$) 'If  A$=chr$(3)+chr$(2) then A&=(3*256)+(2*1)=770`

See also : CVI, MKI$, MKIR$

10-16-78- DAC – Analogue outputs

Syntax : **DAC** (<Output>) = <Expression>

Units : Expression : Volts

Limits : Expression : 0 to +10

Accepted types : Expression : real

Description : This function sends a voltage on the analogue output of a SOA12 card.

Remarks : <Output> must represent an analogue output name. The analogue outputs can be read.

' Warning : On the +/- 10V output of the SOA12 card (Ex : pin1 or pin 2 of SUBD), <Expression>=0 gives -10V on the analogue output, <Expression>=5 gives 0V and <Expression>=10 gives 10V.

Example :
```
DAC(Ana2)=5.2
IF ADC(Ana1)>DAC(Ana2) THEN ...
```

See also : ADC

10-16-79- DATE$ - Current Date

Syntax : **DATE$**

Description : This instruction returns a 10 characters string under the form dd/mm/yyyy, where dd is the day (01-31), mm is the month (01-12) et yyyy is the year.

Example : `a$=DATE$ 'Result : a$="01/01/1996"`

See also : TIME$, TIME, TIMER

10-16-80- DEC - Deceleration

Syntax 1 : **DEC**(<Axis>) = <Expression>

Syntax 2 : <Variable> = **DEC**(<Axis>)

Units : user unit/s² (Ex : mm/s², degree/s², round/s²…)

Limits : 5.10^-5 to 1,5.10^6 pulses/s

Accepted types : Variable, Expression : real

Description : This instruction reads or writes the current deceleration in units per s².

Remarks : The current deceleration can be read and modified at any time.

Example : `DEC(X)=500.`

See also :	ACC, VEL

10-16-81- DEC% - Deceleration in percent

Syntax :	**DEC%**(<Axis>) = <Expression>
Limits :	Expression : 0 to 100
Accepted types :	Expression : real
Description :	This function sets the current deceleration in percent of the deceleration parameter(DEC_P).
Remarks :	The value of DEC_P can be entered in the screen "speed profile" during the board configuration

Example :	`DEC_P(X)=500`
	`DEC%(X)=10 'Current deceleration 50 unit / s²`

See also :	ACC% and VEL%

10-16-82- DELAY – Passive waiting

Syntax :	**DELAY** <Duration>
Units :	Duration : milliseconds
Accepted types :	Duration : Integer
Description :	This instruction allows to the system to wait for the time <Duration>. The task continue its execution when the duration is finished. The execution of this instruction launches to the execution of the next task.

Example :	`DELAY 500  '0.5 s. Delay`
	`DELAY Timer1`

10-16-83- DIFFUSE – Event generation

Syntax :	**SIGNAL** <Name>
Description :	This instruction generates an event.
Remarks :	<Name> must be the same name used by WAIT EVENT instruction. Each program which was waiting for this event can then go on.

Example :

```
Program1                 Program2
...                      ...
WAIT EVENT Ready         DIFFUSE Ready
...                      ...
```

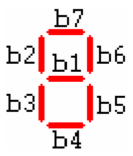
See also :	WAIT EVENT, SIGNAL

10-16-84- DISABLERECALE – Desactivation of recale

Syntax :	DISABLERECALE (<Axis>)
Description :	This instruction cancel the fit automaticaly an axis on a sensor.
See also :	ENABLERECALE

10-16-85- DISPLAY – 7 segments status display

Syntax :	**DISPLAY** "<Car>" or **DISPLAY** <Expression>
Accepted types :	Car : character between '0' and '9'
	Expression : Byte
Description :	This instruction fixes the state of the MCS32Ex status display.
Remarks :	It is possible to use a numerical expression in it, each bit represents a segment. The least significant bit is not used.
Example :	`Display "5"        ' Equivalent to Display 10110110b`



10-16-86- DIV – Integer divide

Syntax :	<Expression1> **DIV** <Expression2>
Accepted types :	Expression1, Expression2 : Integer
Description :	This operator returns the integer divide result.
Remarks :	This operator returns an integer.
Example :	`a%=7`
	`a%=a% DIV 2        'Result : a%=3`
See also :	MOD

10-16-87- EDIT – Editing on operator panel

Syntax 1:	<Variable>=**EDIT**(<Line>,<Row>,<Length>,<Sign>,<Point>)
Syntax 2:	<Variable>=**EDIT**(<Line>,<Row>,<Length>,<Sign>,<Point>, <Code>)
Limits :	Line : 1 to 4 for Dialog 80 and 160 or 1 to 16 for Dialog 640.
	Row : 1 to 20 for Dialog 80 or 1 to 40 for Dialog 160 and 640.
Accepted types :	Variable : real
	Line, Row, Length : Integer
	Sign, Point, Code : bit
Description :	This function allows to edit a real number with the operator panel by using the numerical keys, the DEL key to suppress, the ENTER key to valid and ESC to escape. The second syntax defines the access code mode of editing (Code=1). In this case, all the key press display a star on the operator (*) panel. The execution of this instruction launches the execution of the next task.
Remarks :	<Line> et <Row> are the first character position. <Length> is the maximum number of characters. <Sign> is a boolean value which indicates if the sign can be changed. <Point> is a boolean value which indicates if the point is permitted. The system variable KEY contains the last pressed key. If the edition is aborted then KEY=@ESC and otherwise KEY=@RETURN.
Example :	`A!=EDIT(1,10,4,0,0)       ' Capture in line 1 row 10`
	`                          ' on 4 characters, the sign and the`

```
                                    ' point are not autorised..
          A!=EDIT(1,10,4,0,0,1)     ' Same capture with access code mode
```

10-16-88- EDIT$

Syntax :	<Variable>=**EDIT$**(<Line>,<Row>,<Length>)
Limits :	Line : 1 to 4 for Dialog 80 and 160 or 1 to 16 for Dialog 640.
	Row : 1 to 20 for Dialog 80 or 1 to 40 for Dialog 160 and 640.
Accepted types :	Variable : Char string
	Line, Row, Length : Integer
Description :	This function allows to edit a string with Dialog 80 or Dialog 640 operator panel by using the alphanumeric keys, the DEL key to suppress, the ENTER key to valid and ESC key to escape. For writing an alphanumeric character, push several times on the associated numeric key, to change the character. The record of the character makes itself automatically when you don't push on the associated numeric touch or you push on other touch.
Remarks :	<Line> and <Row> are the first character position. <Length> is the maximum number of characters. The system variable KEY contains the last pressed key. If the edition is aborted then KEY=@ESC and otherwise KEY=@RETURN.

Example :
```
A$=EDIT$(2,9,5)   'capture in line 2, row 9
                  'on 5 characters maxi.
```

10-16-89- ENABLE – ENABLE signal driving

Syntax :	**ENABLE**(<Axis>) = ON / OFF
Description :	This instruction allows to drive the ENABLE output.
Remarks :	This signal must be connected to the ENABLE input of the drive. Il allows to enable or disable the power stage of the drive.
	It can be used in positive or negative logic by using ENBINV_P.
Example :	ENABLE(X)=AXIS_S(X)

10-16-90- ENABLE_S – ENABLE signal state

Syntax :	**ENABLE_S**(<Axe>)
Description :	This function return the state of the ENABLE signal.
Example :	IF ENABLE_S(X) THEN GOTO DriveEnable

10-16-91- ENABLERECALE – Automatic fitting encoder function

Syntax :	ENABLERECALE (<Axis>, <Register Number>, <Config>, <Initial Position>, <Acceleration>, <Min>, <Max>, <Internal>)
Limits :	<Initial Position> : between 0 & axis modulo
Accepted types :	<Initial Position> : Real
	<Acceleration> : Real

	<Min>,<Max> : Real
Description :	This instruction fit automaticaly an axis on a sensor.
Remarques :	<Register Number> define the used register number : 1 ou 2
	<Config> parameters necessary to the capture :
	Bit 0 : Activation of the capture
	Bit 1 : Capture on Top Z
	Bit 2 : Capture on Cellule 1
	Bit 3 : Capture on Cellule 2
	For exemple to fit encoder on Top0 we use the value 3.
	<Initial Position> indicate the position where is theoretically the cellule and we put in the count. For reset, we indicate 0.
	<Acceleration> as the fonction MASTEROFFSET, allow a used acceleration for apply the offset
	<Min>,<Max>,<Internal> define a windows where is appllicate the fitting.
Attention :	The value to put in <Initiale Position>, must don't consider the Zero Programm parameter of the axis.
See also :	DISABLERECALE

10-16-92- END – Block end

Syntax : **END** {PROG | SUB |IF | WHILE}

Description : Bloc end.

Remarks : You must specify a keyword after END

Examples :

```
SUB   Manuel
...
END  SUB
```

See also : PROG, SUB, IF, WHILE

10-16-93- ENDCAM – Stop a cambox

Syntax : **ENDCAM**(<Slave>)

Description : The function ENDCAM stops the slave movement at the end of the cycle, while the functions STOP stop it immediately.

Remarks : Warning : If ENDCAM is applied to a cam which has been declared in non-single shot and linked with another one, the cam ends its profile and goes on to the next.

See also : CAM, STOP

10-16-94- EXIT SUB – Subroutine exit

Syntax : **EXIT SUB**

Description : This instruction allows to exit of a subprogram.

See also : SUB

10-16-95- EXP - Exponential

Syntax :	**EXP** (<Expression>)
Accepted types :	Expression : real
Description :	This function returns *e* (natural logarithms base) raised to <Expression> power.
Remarks :	The argument <Expression> must be a valid numerical expression.
Example :	`a!=EXP(2)`
See also :	LOG

10-16-96- FEMAX_S – Following error limit

Syntax :	**FEMAX_S**(<Axis>)
Description :	This function indicates maximum following error overflow.
Remarks :	<Axis> must be a servo axis. This function is used to know if maximum following error is reached. It will be necessary to use this flag in a security task when the SECURITY(0,0) or SECURITY(0,1) instruction are used.

Example :	`IF FEMAX_S(X) THEN PRINT "Default Axe X"`
	`IF FEMAX_S(Y) THEN PRINT "Default Axe Y"`

See also :	FE_S, SECURITY

10-16-97- FE_S – Current following error

Syntax :	**FE_S**(<Axis>)
Description :	This function returns an image of the current pursuit error.
Remarks :	<Axis> must be a servo axis. This function is used to know the current value of the following error. Thus, we can verify the behaviour of the regulation in real time.

Example :	`a!=FE_S(X)`
	`PRINT  #1,a!`

See also :	FEMAX_S

10-16-98- FLASHOK – Test flash memory

Syntax :	FLASHOK
Description :	This function indicates if parameters and the first 10000 saved variables are backed up in flash memory
See also :	RAMOK, RAMTOFLASH, FLASHTORAM

10-16-99- FIFO – Read distances between state change

Syntax :	**FIFO(**<Axe>, <Distance>, <Etat>)
Accepted types :	<Distance> : long integer local variable <Etat> : bit local variable
Description :	This function allow to read the distance running between each input state change.
Remarks :	Fifo can have 1000 states changes.

10-16-100- FIFONUMBER – Number of observed state change

Syntax :	<Variable> = FIFONUMBER(<Axis>)
Accepted types:	<Variable> : Integer
Description :	This function allow to know the number of observed state change detected by input since the last reading of the FIFO.
Remarks :	The FIFO can only have 1000 states changes.

10-16-101- FILTER – Activate or desactivate a filter

Syntax :	FILTER(<Axis>,<Input>,<Desactivate filter>)
Accepted types:	<Input> : Byte
Description :	This instruction allow to activate or desactivate a filter.
Remarks :	the HOME instruction activate the default filter.
	<Axis> : Name of the servo card
	<Input> : Number of the input (1 -> input C1, 2 -> input C2)
	<Desactivate filter> : Filter validation: 1 without filtering, 0 for a 2ms filter.

10-16-102- FLASHTORAM – Restore saved variables

Syntax :	FLASHTORAM
Description :	This function restore parameters and the first 10000 saved variables from flash memory. This function is automatically called by system if variables are corrupted on start-up.
Voir aussi :	RAMOK, RAMTOFLASH, FLASHOK

10-16-103- FONT – Font selected

Syntax :	**FONT**=**<**Value**>**
Accepted types :	<Value> : byte.
Description :	This function defines the font of the operator panel.
Remarks :	<Value> :
	Font 1 : 16 lines x 40 characters with text and black background, 3x4mm
	Font 2 : 9 l x 30 c with text and black background, 4x7mm
	Font 3 : 6 l x 20 c with text and black background, 12x20mm
	Font 4 : 4 l x 15 c with text and black background, 16x22mm
	Font 5 : 16 l x 40 c with text and white background, 3x4mm
	Font 6 : 9 l x 30 c with text and white background, 4x7mm
	Font 7 : 6 l x 20 c with text and white background, 12x20mm
	Font 8 : 4 l x 15 c with text and white background, 16x22mm

Example :

```
FONT=1
Locate 2,15
```

```
Print "MODE AUTO"
```

10-16-104- FOR – FOR … NEXT loop

Syntax : FOR <Counter>=<Begin> TO <End> [STEP <step>]

...

NEXT <Counter>

Accepted types : Counter : Byte, Integer, Long integer

Description : Repeats an instruction a specified number of time.

Remarks : FOR starts the FOR ... NEXT loop structure. FOR must appear before all the other parts of the structure. <Counter> is a local integer variable used as loop counter. <Counter> is equal to <End>+1 at the end of the loop. <step> must be a positive value. The execution of this instruction NEXT passed to the execution of the next task.

Example :

```
FOR i%=1 TO 10
...
NEXT i%
```

See also : WHILE

10-16-105- FORMAT$

Syntax : **FORMAT$**(<Expression>,<Number>,<Precision>,'<Car>',<Sign>,<LeftAlign>)

Accepted types : Expression : real

Number, Precision : Byte

Car : string char

Sign, LeftAlign : Bit

Description : This function creates a formatted string.

Remarks : The argument <Expression> must be a valid numerical expression. <Number> is the minimum number of characters of the string. <Precision> is the number of character after the decimal point . <Car> is the substitution character used to reach this minimum number if it is necessary. <Sign> indicates if the character "+" or "-" must be added at the beginning of the string. <LeftAlign> indicates if the string is left aligned.

Example :

```
a!=1.2562
b$=FORMAT$(a!,5,2," ",0,1)      ' a$="1.25 "
```

10-16-106- FRAC – Fractional part

Syntax : **FRAC**(<Expression>)

Accepted types : <Expression> : real

Description : This function provides the fractional part of the <Expression>.

Remarks : This fonction returns a real value.

Example :

```
b!=3.0214
a!=FRAC(b!)        'Result a!=0.0214
```

See also : INT

10-16-107- GEARBOX – Gearbox

Syntax : **GEARBOX**(<Slave Axis>,<Master Axis>,<Numerator>,<Denominator>, <Reverse>,[<Acceleration>])

Accepted types : Numerator, Denominator : real

<Acceleration> : Byte, Integer, Long integer or real

Reverse : bit

Description : This function links 2 axis in gearbox.

Remarks : <Master Axis> can be a servo, a time based encoder or an encoder.

<Slave Axis> must be a servo axis.

The ratio is defined by <Numerator> / <Denominator>.

<Reverse> is a boolean which indicates that the gearbox is reversible. <Acceleration> is an optional parameter which defines an acceleration period.

This instruction doesn't stop the task. The link made between master and slave axis is valid as long as STOP(Slave) instruction or AXIS(Slave)=OFF instruction will not be activated.

Example :

```
GEARBOX(Esclave,Maître,-1,2,1) 'Reverse axis and slave axis will have
                               'a velocity two times big as the
                               'master axis which turn in an
                               'opposite direction
```

See also : CAM, CAMBOX, MOVS, STOP, AXIS

10-16-108- GEARBOXRATIO – Change the reduction ratio of an electrical shaft

Syntax : **GEARBOXRATIO**(<SlaveAxis>,<Ratio>,<AccRation>)

Accepted types : <Ratio> : real

Description : This function changes the reduction ratio of an electrical shaft.

Remarks : <SlaveAxis> : must be a servo axis board of an electrical shaft link. The shaft ration is defined by : <Ratio> × <Numerator> / <Denominator>. <Numerator> and <Denominator> are the parameters the GEARBOX instruction.

This is an unstopped instruction and it can change the ratio without to stop the electrical shaft. The ratio can be positive or negative.

<AccRation> : Optionnal parameter allowing a progessive modification of an electrical shaft ratio. This acceleration is express in increment / 0,33ms

Ex : For a ration of 1 to 1,2 with a acceleration value of 0,0001 we put ((1,2-1/0,0001)*0.33=660ms

Warning : This function modify a general facto (K1) who is used in all master's distance of synchro functions : MOVS, CORRECTION, CAM …

Example :

```
    GEARBOXRATIO (Slave,Master,1.1)
MOVS (Slaver,Master,100, … )   'intern master distance = 100*1,1
```

Example :

```
GEARBOX(Esclave, Maître, 1, 2, 1) 'Nominal ratio  : ratio 0.5
GEARBOXRATIO(Esclave,1.2)         '20% increase
```

```
                                   'Ratio   : (1.2×1/2)=0.6
GEARBOXRATIO(Esclave,0.8)          '20% reduction
                                   'Ratio   : (0.8×1/2)=0.4
```

See also :	CAM, CAMBOX, MOVS, STOP, AXIS

10-16-109- GETAXISEVENT – Reading events out of factual tasks

Syntax :	<Variable> = GETAXISEVENT
Accepted types:	<Variable> : Integer
Description :	This function allow to accomplish and read the events who have been detecte on inputs. This instruction must be place at the beginning of the event task.
Remarks :	Each associated bit to a n event is to the state '1' if the event has been detected. If a new event appear during the running of an event task, it is memorised and treated as possible.The event coding is the same as the mask in MODIFYAXISEVENT.

10-16-110- GETDATE – Current date

Syntax :	**GETDATE**(<Year>,<Month>,<Day>,<DayInTheWeek>)
Accepted types :	<Year>, <Month>, <Day>, <DayInTheWeek> : Integer.
Description :	This instruction reads the current date.
See also :	GETTIME

10-16-111- GETEVENT – Events reading

Syntax :	<Variable> = **GETEVENT**
Accepted types :	<Variable> : Integer
Description :	This instruction consumes and reads detected events.
Remarks :	All the event bit are setting if the event is detected. If a new event appears during the execution of the event task, event is stored and is treated as soon as it is possible.
See also :	MODIFYEVENT

10-16-112- GETTIME – Current time

Syntax :	**GETTIME**(<Hour>,<Minute>,<Second>)
Accepted types :	<Hour>, <Minute>, <Second> : Integer.
Description :	This instruction reads the current time.
See also :	GETDATE

10-16-113- GOTO – Branch label

Syntax :	**GOTO** <Label>
Description :	Jumps to a label
Remarks :	The programs with lots of GOTO instructions can become hard to read and to perfect. Use the control structures (FOR...NEXT, REPEAT...UNTIL,

WHILE...END WHILE, IF...THEN...ELSE...END IF) each time it is possible. A label is a name following by character ":". The execution of this instruction passed to the execution of the next task.

Example :

```
GOTO Begin
...
Begin :
```

See also : JUMP, FOR, REPEAT, WHILE, IF, END

10-16-114- HALT – Stop a task

Syntax : **HALT** <Name>

Description : This instruction is used to stop a task which is going to be executed or suspended.

Remarks : This function has no effect on the stopped task, on the movements running and on the buffer of movements.

Example :

```
Begin :
  Wait Inp(Power)=On
  RUN  Cutter
  Wait Inp(Power)=Off
  HALT Cutter
Goto Begin
```

See also : RUN, SUSPEND, CONTINUE

10-16-115- HLINE – Draw horizontal line

Syntax : **HLINE**(X1,Y1,X2,colour)

Units : X1, Y1, X2 : pixel

Limits : X1, X2 : de 1 à 240

Y1 : de 1 à 128

Accepted types : X1, Y1, X2 : byte.

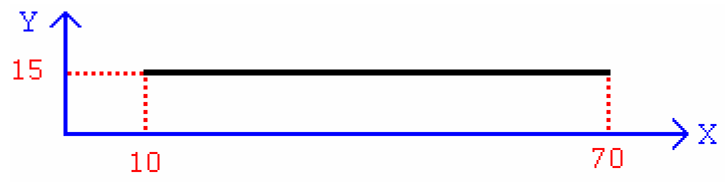
Colour : Bit

Description : This instruction draws a line with its starting point in X1, Y1 and its final point in X2, Y1 on the operator panel Dialog 640.

Remarks : Colour changes the colour of the line : black (0) or white (1)

Example :

```
HLINE(10,15,70,0)
```



10-16-116- HOME – Home

Syntax : **HOME**(<Axis>[, <Type>])

Description : This function forces <Axe> to move towards its home position by using the chosen <Type> of home. This instruction locks the task as long as the home is not finished. The home process uses VELHOME_P parameter form velocity. The input used for the sensor is indicated in INPHOME_P. At the end of the home process axis moves to the position defined by DISHOME_P and the speed is kept to the VELHOME_P value. The execution of this instruction launches the execution of the next task.

<Type> values are :

0 : Immediate home

1 : On encoder zero in positive direction

2 : On encoder zero in negative direction

3 : On sensor zero in positive direction

4 : On sensor zero in negative direction

5 : Fast, on sensor zero in positive direction

6 : Fast, on sensor zero in negative direction

7 : On sensor and encoder zero in positive direction

8 : On sensor and encoder zero in negative direction

9 : Fast, on sensor and encoder zero in positive direction

10 : Fast, on sensor and encoder zero in negative direction

Remarks : Use AXIS(<Axis>)=Off to stop home process. If <Type> is not specified setup screen value will be used. The parameters VELHOME_P, DISHOME_P and INPHOME_P can be chosen in the same screen. They must be sent to MCS with "Send setup" command of the Communication menu.

Warning : For a modulo axis, the instruction HOME put temporarily to 0 his parameter MODULO_P. If you stop one task who is complying the instruction HOME, forecast to reforce MODULO_P to 1.

Example : `HOME(X)`

See also : HOME_S

10-16-117- HOME_S– Home state

Syntax : **HOME_S**(<Axis>)

Description : This function indicates the home state

Remarks : <Axis> must be a servo axis. This function is used to know if the home has already been successfully executed or not. During a home cycle, HOME_S is set. When the home cycle is finished, HOME_S is cleared.

Example : `IF NOT HOME_S(X) THEN HOME(X)`

See also : HOME

10-16-118- ICALL – Call a sub-routine

Syntax : **ICALL** <Name>

Description : This instruction is used to call a sub-routine define by a block SUB. <Name> is the name of the sub-routine's block.

Remarks : A sub-routine can not call himself. The system had predefine sub-routines : _MENUMCS, _PARAMMCS, _MANUMCS, _VARIABMCS, _MEMORYMCS and _CLOCKMCS. The execution of this instruction don't launches the execution of the next task..

Exemple : ICALL Move

Voir aussi : SUB, CALL

10-16-119- ICORRECTION – Function of correction

Syntax : **ICORRECTION**(<Slave>,<Master>,<Dist.master>,<Dist.slave>, <Dist. accel>)

Units : <Dist.master>, <Dist.slave> : user unit (Ex : mm, degree,…)
<Dist.accel> : user unit /s²

Accepted types : <Dist.master>, <Dist.slave>, <Dist.accel> : real

Description : This function applies a correction movement on the slave axis during the distance of master axis.

Remarks : The slave axis must be linked to the master axis by a gear box function (GEARBOX or GEARBOXM), a synchronized movement (MOVS, MOVSM, MOVSP, MOVSC) or a cambox function (CAM, CAMC) before the execution of the correction instruction. With the synchronized movement of the slave axis, the next movement is superposed : During the distance of the master axis, a movement <Dist.slave> is added with an acceleration and a deceleration on a <Dist accel>. <Slave> must be a servo card SRV85 and <Master> can be a servo card or an encoder.

See also : CORRECTION

10-16-120- IF - IF…Then…Else

Syntax 1 :
IF <Condition> **THEN**
{<Instruction1>}
...
ELSE
{<Instruction2>}
...
END IF

Syntax 2 : **IF** <Condition> **THEN** <Instruction1> **ELSE** <Instruction2>

Description : Allows the conditional execution based on the expression evaluation.

Remarks : The keyword IF begins a control structure. IF...THEN...ELSE...END IF . It must appear before all other part of the structure. <Condition> must be a boolean expression.

If <Condition> is right then <Instructions1> are executed.

If <Condition> is false then <Instructions2> are executed.

Example :
```
IF (a%>1) AND (a%<10) THEN
  Locate 1,1
  Print "Length 1"
Else
  Locate 2,1
  Print "Width 1"
END IF
```

See also : END

10-16-121- ILIMIT – driving ILIMIT state

Syntax : **ILIMIT**(<Axe>) = ON / OFF

Description : This instruction allows to drive the ILIMIT output.

Remarks : This signal can be connected to the ILIMIT input of the drive. It allows to limit the current when the axis is stopped.

It can be used in positive or negative logic by using ILMINV_P.

Example :

```
AXIS(X)=ON          ‘ axis enabled
ENABLE(X)=ON
VEL(X)=100          ‘ initialization of the speed profile
ACC(X)=1000
DEC(X)=1000
ILIMIT(X)=OFF       ‘ non current limitation
MOVA(X=360)‘ moving
ILIMIT(X)=ON‘ current limitation when the axis is stopped
....
```

10-16-122- ILIMIT_S – ILIMIT signal state

Syntax : **ILIMIT_S**(<Axis>)

Description : This function return the state of the ILIMIT signal.

10-16-123- INKEY– Read a key on the operator panel

Syntax : <Variable>=**INKEY**

Accepted types : Variable : Byte

Description : This function reads a key from the keyboard of the operator panel and returns its code.

Remarks : This function does not stop the task. Cette fonction est non bloquante pour la tâche. If the input buffer is empty (no key has been pressed) this function returns 0.This function uses the communication port #1. By default, the communication port SERIAL1 will be used. If an operator panel Dialog 160 is connected to the port SERIAL2, please refer yourself to the OPEN function to affect #1 to the port SERIAL2.

Example :

```
REPEAT
A#=INKEY
UNTIL A#<>0
```

10-16-124- INP – Input reading

Syntax :	**INP** (<Input>)
Accepted types :	<Input> : Bit
Description :	This function gives the state of a digital input.
Remarks :	<Input> must represent an input name TOR. The returned data type is Bit.
Example :	`C~=INP(HighCutter)`
See also :	INPB, INPW, OUT, OUTB, OUTW

10-16-125- INPB – 8 digital inputs reading

Syntax :	**INPB** (<Input>)
Accepted types :	<Input> : Byte
Description :	This function gives the state of 8 digital inputs TOR..
Remarks :	<Input> must represent the 8 inputs name. The returned data type is Byte.
Example :	`B#=INPB(Data)`
See also :	INP, INPW, OUT, OUTB, OUTW

10-16-126- INPUT – Data reading

Syntax : **INPUT** #<Number>, <Variable> [{,<Variable>}]

Accepted types : Variable : Bit, Byte, Integer, Long integer, real and Char string

Number : #1 or #2

Description : Reads data from the communication port and assigns the data to the variables. The execution of this instruction passed to the execution of the next task.

Remarks : <Number> is the number used to open a communication port with OPEN function. The read data must appear in the same order that the variables list.

Example :

```
OPEN "SERIAL1:"  AS #1
INPUT #1,A$,B%
CLOSE #1
```

See also : OPEN, PRINT, CLOSE

10-16-127- INPUT$ - Char string reading

Syntax : <Variable> =**INPUT$** <CommNumber>, <NumberOfChar>

Accepted types : Variable : Char string

CommNumber : #1 or #2

NumberOfChar : Byte

Description : Reads <NumberOfChar> characters from the communication port and stores them in a char string. The execution of this instruction launches the execution of the next task.

Remarks : <CommNumber> is the number used to open the communication port with OPEN instruction.<Variable> must be a variable char string type. The task is blocked on this instruction when the number of character received is different than this specified in the instruction.

Example :

```
OPEN "SERIAL1:" AS #1
A$=INPUT$ #1,5    'Read 5 characters from the communication port
CLOSE #1
```

See also : OPEN, PRINT, CLOSE

10-16-128- INPW – 16 digital inputs reading

Syntax : **INPW** (<Input>)

Accepted types : <Input> : Integer

Description : This function gives the state of the16 digital inputs.

Remarks : <Inputs> must represent the name of a 16 digital inputs board.Data type returns is integer.

Example :

```
A%=INP(Bloc)
```

See also : INP, INPB, OUT, OUTB, OUTW

10-16-129- INSTR – Search a sub-string

Syntax : **INSTR**(<string1>,<string2>)

Accepted types : string1, string2 : Char string

Description : This function searches a sub-string in a char string and returns the position of the first occurrence of the sub-string.

Remarks : <String1> is the researched string <String2>.

Example :

```
a$="Press ENTER to start"
EnterPos%=INSTR("ENTER",a$) 'Result : EnterPos%=7
```

See also : LEN

10-16-130- INT – Integer part

Syntax : **INT** (<Expression>)

Accepted types : Expression : real

Description : This function returns the <Expression> integer part.

Remarks : The argument <Expression> must be a valid numerical expression

Example :

```
b!=25.36
a!=INT(b!)         'Result : a!=25
```

See also : FRAC

10-16-131- JUMP – Branch to label

Syntax : **JUMP** <Label>

Description : Jumps to a label

Remarks : The programs with lots of GOTO instructions can become hard to read and to perfect. Use the control structures (FOR...NEXT, REPEAT...UNTIL, WHILE...END WHILE, IF...THEN...ELSE...END IF) each time it is possible. A label is a name following by character ":". The execution of this instruction doesn't launch the execution of the next task.

Example :
```
JUMP Begin
...
Begin :
```
See also : GOTO, FOR, REPEAT, WHILE, IF, END

10-16-132- KEY – Last pressed key

Syntax : **KEY**

Description : This system variable contains the last pressed key.

Remarks : This variable must be used after EDIT et WAIT KEY instructions. The key variable is local for a task.

Example :
```
WAIT KEY
IF KEY=@F1 THEN CALL ...
IF KEY=@F2 THEN CALL ...
IF KEY=@F3 THEN CALL ...
```
See also : EDIT, WAIT KEY

10-16-133- KEYDELAY – Delay before key repeat

Syntax : **KEYDELAY** = <Expression>

Units : Expression : 1/32 of second.

Accepted types : Expression : Byte

Description : This instruction defines the delay before the automatic repetition of a key when this is pressed.

Remarks : The default value is 1 second (32).

Example : `KEYDELAY = 10`

See also : KEYREPEAT

10-16-134- KEYREPEAT – Keyrepeat period

Syntax : **KEYREPEAT**=<Expression>

Units : Expression : 1/32 of second.

Accepted types : Expression : Byte

Description : This instruction defines the delay which separates each automatic key repetition when this is pressed..

Remarks : The default value is 0.3 second (10).

Example : `KEYREPEAT = 5`

See also : KEYDELAY

10-16-135- LCASE$ - Lowercases

Syntax : <Expression> = **LCASE$**(<String>)

Accepted types : String : Char string

Description : This function returns a string in all the letters of the argument have been converted in lowercases.

Remarks : The argument <Expression> must be a char string. Only the uppercases are converted in lowercases, the other letters are not modified.

Example :

```
a$="Sensor1"
b$=LCASE$(a$) 'Result : b$="sensor1"
```

See also : UCASE$

10-16-136- LED – Driving LEDs

Syntax : **LED**(Number)=State

Accepted types : State : bit.

Description : This function allows to drive the LEDs of operator panel Dialog 80 and 640.

Remarks : Definition of the LED : de @F1 to @F6 or @ALARM or @HELP

Definition of their state: switch off (0), light (1), blink (2).

Example :

```
LED(@ALARM)=2   'blink alarm DEL
```

10-16-137- LEFT$ - String left part

Syntax : **LEFT$**(<String>,<Number>)

Accepted types : String : Char string

Number : Integer

Description : This function returns the first <Number> left characters of a string.

Remarks : To find the character numbers in the string <String>, use LEN(<String>).

Example :

```
a$="Sensor1"
b$=LEFT$(a$,6) 'Result : b$="Sensor"
```

See also : RIGHT$, LEN

10-16-138- LEN– String length

Syntax : **LEN**(<String>)

Description : This function returns the number of characters of a string.

Example :

```
a$="Sensor1"
b%=LEN(a$) 'Result : b%=7
```

See also : INSTR

10-16-139- LIMMAX_S – Maximum thrust state

Syntax : **LIMMAX_S**(<Axis>)

Description : This function indicates if the axis position is greater than its maximum thrust.

Remarks : <Axis> must be a servo axis. This function is used to know if the maximum thrust is reached.

Example :

```
SUB Thrusts
IF LIMMAX_S THEN STOP(X) 'if Thrusts maxi reached, then
END SUB                  'axis stopped
```

See also : LIM_S, LIMMIN_S

10-16-140- LIMMIN_S – Minimum thrust state

Syntax :	**LIMMIN_S**(<Axis>)
Description :	This function indicates if the axis position is lower than its minimum thrust.
Remarks :	<Axis> must be a servo axis. This function is used to know if the minimum thrust is reached.

Example :

```
SUB Thrusts
IF LIMMIN_S THEN STOP(X) 'if thrusts mini reached, then          END
SUB                  'axis stopped
```

See also : LIM_S, LIMMAX_S

10-16-141- LIM_S – Thrusts state

Syntax :	**LIM_S**(<Axis>)
Description :	This function indicates if the axis is out of its mini or maxi thrusts.
Remarks :	<Axis> must be a servo axis. This function is used to know if at least one of the thrusts has been reached.

Example :

```
SUB Thrustss
IF LIM_S THEN STOP(X)   'if thrusts mini or maxi reached, then
END SUB                 'axis stopped
```

See also : LIMMAX_S, LIMMIN_S

10-16-142- LOADABSCAMEX– Load an absolute cambox in the servo module

Syntax : **LOADABSCAMEX**(<Slave>, <Master>, <CamNumber>, <Table>, <Number>, <FirstPolynomial>, <Mono>, <Reverse>, <PositiveDirection>, <NextN°Cam>, <PreviousN°Cam>)

Limits : <First Polynome> : 1 to 310

<CamNumber>, <NextN°cam>, <PreviousN°cam> : 1 to 5

Accepted types : <PremierPolynomial>, <Number> : Integer

<CamNumber>, <Number>, <NextN°cam>, <PreviousN°cam> : Byte

<Mono>, <Reverse>, <PositiveDirection> : bit

<Table> : Table of cam

Description : This instruction allow to load a cam in the servo module.

Remarks : <Slave> : Name of the slave's axis where is doing the cam (servo module : SRV15, SRV85...)

<Master> : Name of the master's'axis (servo module or encodeur module: SCD22, SRV15, SRV85...)

<CamNumber> : Number of the cam

<Table> : Name of the camebox table given by the data's editor in global variables's tab of MCB software (variable of type « real »).

<Number> : table'size to definethe cam.

Number = (CamPointNumber + 2) *2

<FirtPolynomial> : A servo module had a global table of 310 polynomials for 5 cams. Enter a value between 1 and 310 to indicate where will be stocked the first polynomial of the cam in the globle table of card.

A table of (CamPointNumber-1) polynomial is given by the table of cam.

Warning : <FirstPolynomial> + <CamPontNomber-1> must be lest than 310.

<Mono> : Define the automatic relooping of the came. Enter the value 0 for a cam who reloops on his profil always until a stop will send, enter 1 for a cam who execute his profile one time.

<Réversible> : Indique si l' <Esclave> doit suivre le <Maitre> dans les deux sens.

☞ Rentrez la valeur 0 pour une came non réversible : si le maître se déplace à l'inverse de son sens normal donné par <DirectionPositive>, l'esclave s'arrête ; il repartira lorsque le maître reprendra son sens normal et atteindra la position maître à laquelle l'esclave s'était arrêté.

☞ Rentrez la valeur 1 pour une came réversible : l'esclave suit son profil de came quel que soit le sens d'avance du maître.

<DirectionPositive> : Si la came n'est pas réversible, le sens normal de l'avance du maître doit être indiqué.Rentrez la valeur 0 pour un sens négatif, 1 pour un sens positif.

<N°came suivante> : Mettez 0 si la came ne doit pas être enchaînée sur une autre came. Dans le cas contraire, rentrez le numéro de la came suivante compris entre 1 et 5.

<N°came précédente> : Mettez 0 si la came n'enchaînera pas sur une came précédente. Dans le cas contraire, rentrez le numéro de la came précédente compris entre 1 et 5.

See also : STARTABSCAM

10-16-143- LOADCAMEX – load a cam

Syntax : **LOADCAMEX**(<Slave>, <Master>, <NumberCame>, <Table>, <Number>, <FirstPolynomial>, <SingleShot>, <Reversible>, <PositiveDirection>, <NumberNextCam>, <NumberPreviousCam>)

Limits : <FirstPolynomial> : 1 to 310

<NumberCame>, <NumberNextCam>, <NumberPreviousCam> : 1 to 5

Accepted types : <FirstPolynomial>, <Number> : Integer

<NumberCame>, <NumberNextCam>, <NumberPreviousCam> : Byte

<SingleShot>, <Reversible>, <PositiveDirection> : bit

Description : this instruction loads a cam in a servo board.

Remarks : <Slave> : Slave axis cam for which the cam will be applied (servo board: SRV15, SRV85...)

<Master> : Master axis name (servo or encoder board : SCD22, SRV15, SRV85...)

<NumberCam> : number of the cam (from 1 to 5)

<Table> : Name of the declared table in the global variables of the MCB (type real)

<Number> : Number of elements of the table to define the cam

<Number> = (NumberCamPoints + 2) * 2

<FirstPolynomial> : A servo board contains a global table of 310 polynomials for the 5 cams. Input a value from 1 to 310 to tell where the first polynomial of the cam will be stored in the global table of the board. Warning: <FirstPolynomial> + <NumberCamPoints-1> must be lower than 310.

<SingleShot> : Define the automatical re-looping of the cam.

↳ 0: Re-looping cam, it will be stopped only when the stop instruction will be executed.

↳ 1: Single-shot cam

<Reversible> : Tell if the <Slave> must follow the master in both directions.

↳ Input 0 for a non-reversible cam: if the master moves in the opposite way as the one defined in <PositiveDirection>, the slave stops. It will start off again when the master will go in the right way and pass by the position where the slave stopped.

↳ Input 1 for a reversible cam: The slave follows its cam profile whatever is the master direction.

<PositiveDirection> : If the cam is not reversible, you must indicate the usual direction of the master. Input 0 for a negative direction, 1 for a positive one.

<NumberNextCam> : Input 0 if the cam must not be followed by another one. If it is not the case, input the number of the next cam, from 1 to 5.

<NumberPreviousCam> : Input 0 if the cam will not start at the end of another one. If it is not the case, input the number of the previous cam (from 1 to 5).

See also : STARTCAM

10-16-144- LOADPOINT – Point of a cam in a servo board

Syntax : **LOADPOINT**(<Slave>,<Master>,<CamTable>,<Number>, <FirstPolynomial>, <VariableIndex)

Limits : <FirstPolynomial> : 1 to 310

Accepted types : <FirstPolynomial>, <VariableIndex>, <Number> : Integer

<CamTable> : Cam table

Description : This instruction allows to change two polynomials with a camtable point in a servo board.

Remarks : <Slave> : Slave axis name where the cam is executed (servo board : SRV15, SRV85...)

<Master> : Master axis name (Servo or encoder board : SRV15, SRV85, SCD22...)

<CamTable> : Cam table name which is defined in the global variables tab of the MCB software (« real » variable type).

<Number> : Number of cam table elements.

Number = (NumberCamPoints + 2) × 2

<FirstPolynomial> : A servo baord has a global table of 310 polynomials for all the five cams. Put a value between 1 and 310 to indicate where the first polynomial of the cam will be stored in the global table of the board.

<VariableIndex> : Variable index of the <CamTableau> where the modification is. The modifications of the polynomials depend on the slave and master axis position. You can use LOADPOINT one time to change this two informations.

See also : LOADCAMEX, STARTCAM

10-16-145- LOADS – Load a synchronized movement

Syntax : **LOADS**(<Slave Axis>, <Master Axis>, <MasterDistance>, <SlaveDistance>,<DistanceAcceleration>, <DistanceDeceleration>)

Units :	MasterDistance, SlaveDistance, DistanceAcceleration, DistanceDeceleration : user unit (Ex : mm)
Accepted types :	MasterDistance, SlaveDistance, DistanceAcceleration, DistanceDeceleration : real
Description :	This instruction is used to link a slave axis to a master axis during a certain distance of the master axis with acceleration and deceleration on the slave axis. <MasterAxis> can be a servo or encoder axis. <SlaveAxis> must be a servo axis.The <Acceleration Distance> and <DecelerationDistance> are expressed in master axis distance and can be zero.
Remarks :	The LOADS instruction is similar to MOVS but the STARTS instruction must be used after LOADS to start the synchronized movement. This feature is very useful to begin simultaneously several synchronized movements.

Example :

```
LOADS(Slave1,Master,90,180,10,10)     'loading of slave1
                                      'movement in the buffer
LOADS(Slave2,Master,90,180,10,10)     'loading of slave2
                                      'movement in the buffer
STARTS(Slave1,Slave2)                 'linked slaves with master
```

See also : MOVS, STARTS

10-16-146- LOCATE – Cursor position

Syntax :	**LOCATE** <Line>,<Row>
Limits :	Line : 1 to 4 for Dialog 80 and 160 or 1 to 16 for Dialog 640. Row : 1 to 20 for Dialog 80 or 1 to 40 for Dialog 160 and 640.
Accepted types :	Line, Row : Byte
Description :	This function is used to select the cursor position on the operator panel.
Remarks :	This function uses the communication port #1. By default, the communication port SERIAL1 will be used. If an operator panel is connected to the SERIAL2 port, please refer you to the OPEN function to affect #1 to the SERIAL2 port.

Example :

```
LOCATE 1,1
PRINT "<MAIN MENU>"
```

10-16-147- LOG - Logarithm

Syntax :	**LOG** (<Expression>)
Accepted types :	Expression : real
Description :	Returns the natural logarithm of <Expression>
Remarks :	<Expression> must be a numerical expression.
Example :	`a!=LOG(1.2)`
See also :	EXP

10-16-148- LONGTOINTEGER – Convert a long integer to integer

Syntax :	**LONGTOINTEGER**(<Expression>)
Accepted types :	Expression : Long integer
Description :	This function converts a long integer type data in integer type data.

Example :

```
A&=Time
B%=LongToInteger(A&)
```

10-16-149- LOOP – Virtual mode

Syntax : **LOOP**(<Axes>)=ON/OFF

Description : This function set axis in virtual mode. In virtual mode axis is virtual, so drives and motor can be disconnected. This instruction can manage a master axis with all the possibilities of a real axis.(Acceleration, Deceleration, Velocity, …).

10-16-150- LTRIM$ - Suppress the left spaces

Syntax : **LTRIM$**(<Expression>)

Description : Returns a string copy without the left spaces.

Remarks : <Expression> must be a char string.

Exemples :

```
a$="     Menu    "
b$=LTRIM$(a$)            ' Result b$="Menu    "
```

See also : RTRIM$

10-16-151- MASTEROFFSET – Shift dynamically the master position

Syntax : MASTEROFFSET(<Slave>,<Master>,<Offset>,<Acceleration>)

Limits : <Offset> : Between 0 ans the master modulo

Accepted types : <Offset> : Real

<Acceleration> : Real

Description : This instruction shift dynamically the master position by an absolute cambox.

Remark : <Slave> : Name of the slave's axis where is donig the cambox (servo module : SRV85)

<Master> : Name of the master's axis (servo or encoder module : SCD22, SRV15, SRV85...)

<Offset> : Offset value to apply

<Acceleration> Used acceleration for apply the offset (increment/T0²).

See also : LOADABSCAMEX, ABSCAM, STARTABSCAM, SLAVEOFFSET

10-16-152- MERGE – Movement merging

Syntax : **MERGE**(<Axis>)=ON | OFF Or **MERGE**(<AxisX>,<AxisY>)

Description : This instruction is used to activate or disable consecutive movements merging. The second form is used for interpolation.

Example :

```
MERGE(X)=ON
TRAJ(POS(X)=1000,VEL(X)=500)    'Movements merge without a
TRAJ(POS(X)=1500,VEL(X)=20)     'crossing with a null Velocity
MERGE(X)=OFF
TRAJ(POS(X)=1800,VEL(X)=1000)   'crossing with a null Velocity
                                'at the position 1500
```

10-16-153- MID$ - String part

Syntax :	**MID$**(<String>, <Begin>, <Length>)
Accepted types :	String : Char string
	Begin, Length : Byte
Description :	This function returns a string part.
Remarks :	<Begin> defines the beginning of the substring extracted and <Length> the number of characters to extract.
Example :	`a$="MAIN MENU    "`
	`b$=MID$(a$,6,4)  ' Result : b$="MENU"`
See also :	LEFT$, RIGHT$

10-16-154- MKI$ - Convert an integer to a string

Syntax :	<string>=**MKI$**(<Variable>)
Accepted types :	<string> : string char of 2 bytes
	Variable : Integer
Description :	This function MKI$ convert long integer value in a string of 2 bytes. Least significant byte and then most significant byte
Example :	`A$=MKI$(A%)       'if A%=256 then A$=01`
See also :	MKIR$, CVI, CVIR

10-16-155- MKIR$ - Conversion Integer reverse / String

Syntax :	<string>=**MKIR$**(<Variable>)
Accepted types :	<string> : string char of 2 bytes
	Variable : Integer
Description :	This function MKI$ convert long integer value in a string of 2 bytes. Most significant byte and then least significant byte
Example :	`A$=MKI$(A%)       'If A%=770 then A$=32`
See also :	MKI$, CVI, CVIR

10-16-156- MKL$ - Convert long integer to string

Syntax :	<string>=**MKL$**(<Expression>)
Accepted types :	<string> : string char of 4 bytes
	Expression : Long integer
Description :	This function MKL$ convert long integer value in a string of 4 bytes. Least significant byte and then most significant byte
Example :	`A$=MKL$(A&)       'if A&=66306 then  A$=2310`
See also :	MKLR$, CVL, CVLR

10-16-157- MKLR$ - Convert long integer reverse to a string

Syntax :	<string>=**MKLR$**(<Expression>)
Accepted types :	<string> : string char of 4 bytes
	Expression : Long integer
Description :	This function MKLR$ convert long integer value in a string of 4 bytes. The most significant word and the least significant word
Example :	`A$=MKL$(A&)        'if A&=66305 then  A$=0132`
See also :	MKL$, CVL, CVLR

10-16-158- MOD - Modulus

Syntax :	<Expression1> **MOD** <Expression2>
Accepted types :	Expression1, Expression2 : Byte, Integer, Long integer
Description :	This operator returns an integer division rest.
Example :	`a%=5`
	`a%=a% MOD 2 'Result : a%=1`
See also :	DIV

10-16-159- MODIFYAXISEVENT – Event configuration

Syntax:	MODIFYAXISEVENT (<Mask>)
Accepted Types:	<Mask> : Integer
Description :	This function allox to configurate the wished event.
Remarks :	<Mask> :

Bit 0 : activation of the 1 event on the first system's detected card (the research is doing from slot A to slot J).

Bit 1 : activation of the 2 vent on the first system's detected card (the research is doing from slot A to slot J).

Bits 2 : activation de the 1 event on the second system's detected card (the research is doing from slot A to slot J).

Bits 3 : activation de the 2 event on the second system's detected card (the research is doing from slot A to slot J).

Bit 4 to 15 : idem for the 6 next axes

After the affectation of the configuratiuon, the event task is launching when an event is detected. The maxi timing between the event appearence and his treatment is egal to the « agering » timing.

If we want to change the event configuration, the MODIFYAXISEVENT instruction must be treated in a normal basic task or in an event task conditionally that it's placed after GETAXISEVENT.

10-16-160- MODIFYEVENT– Events configuration

Syntax : MODIFYEVENT (<Mask>,<Delay>)

Limits : <Delay> : 10ms to 30.000ms

Accepted types : <Mask> : Integer

<Delay> : Integer

Description : This instruction allows to configure events.

Remarks : <Mask> :

↳ Bits 0...6 : Activate the first to seventh input of the first card detected by the system (search A slot to J slot). A positive edge will generate the event. The input take account of the invert and filter parameters entered during the board configuration.

↳ Bit 7 : Time base

↳ Bits 8...15 : activate the C1 or C2 capture input associate with the Capture1 instruction of the 1 to 8 SRV 85 card detected by the system (search A slot to J slot).

<Delay> : Delay of the time base between 10 ms and 30000 ms. If the time base is unused, the value of delay will be not treated.

When the event configuration register is affected, the event task is executed when at least one event is detected. The maxi time between the event detected and its treatment is equal to the task ageing time.

If you want to modify the event configuration register, you'll be treated this instruction in a normal basic task or an event task before the execution of GETEVENT instruction.

See also : GETEVENT

10-16-161- MOVA – Absolute movement

Syntax : **MOVA**(<Axis>=<Distance> {,<Axis>=<Distance> ... })

Limits : Distance : +/- 2^24 pulses

Units : Distance : user unit (Ex : mm, degree,…)

Accepted types : Distance : real

Description : Moves the axis to an absolute position. The execution of this instruction passed the execution of the next task.

Remarks : The system waits for the end movement (Condition MOVE_S(Axe)=0) before executing the next instruction. The axes use the current speed, acceleration and deceleration values. <Axis> must be a servo axis.

Example : MOVA(X=1200.00,Y=-100.00,Z=+550.00,W=Dist!)

See also : MOVR, STTA, STTR, STTI and MOVE_S

10-16-162- MOVAC – Absolute movement triggered on capture input

Syntax : MOVAC(<AxeEsclav>=Distance,<Configuration>
[,<Master Axis>,<Window>, <Mini>,<Maxi>,<Inside>])

Limits : Distance : +/- 2^24 pulses

Accepted types : Distance, Mini, Maxi : real

Configuration : Byte

↳ b0 : unused

↳ b1 : triggering on signal Z

↳ b2 : triggering on capture 1 input (C1)

↳ b3 : triggering on capture 2 input (C2)

↳ b4 : 1 : negative edge ; 0 : positive edge

↳ b5…b7 : unused

Inside : bit

Description : This instruction define a movement on a slave axis and a master axis when the capture input is activated. The execution of this instruction launches the execution of the next task.

Remarks : The system waits for the end movement (Condition MOVE_S(Axe)=0) before executing the next instruction. <Axis> must be a servo axis. <Slave Axis> must be stopped (MOVE_S(Slave Axis)=0) before the instruction is send even the movement is obtained otherwise the condition is false. <Master Axis> and <Slave Axis> can be the same axis. The axis used the current velocity, acceleration and deceleration. <Mini> and <Maxi> define the triggering window of the slave movement. <Mini> must be lower than <Maxi>. <Inside> is used to indicate if the triggering is inside or outside the window. The parameters <Master Axis>, <Window>, <Mini>, <Maxi> and <Inside> are optional.

Example :

```
MOVAC (X=200,X,4,0,0,0,0)     'send  X in 200 on a positive edge
                              ' of input 1
MOVAC (X=500,Y,24,0,0,0,0)    ' send X in 500 on a negative edge
                              ' of input 2 of axis Y
```

See also : MOVA, MOVR, STTA, STTR, STTI, MOVE_S

10-16-163- MOVAP – Absolute triggered movement

Syntax : **MOVAP**(<Slave Axis>=<Distance> ,<Master Axis>,<MiniPosition>,
<MaxiPosition>,<Inside>)

Limits : Distance : +/- 2^24 pulses

Accepted types : Distance, MiniPosition, Maxi Position: real

Inside : bit

Description : This instruction allows to define a movement on a slave axis when a master axis enters in a position window. The execution of this instruction launches the execution of the next task.

Remarks : The system waits for the movement end (Condition MOVE_S(Axe)=0) before executing the next instruction. The axis uses the current speed, acceleration and deceleration. <MiniPosition> and <MaxiPosition> define the trigger window used to start the movement on the slave. <MiniPosition> must be lower than <MaxiPosition>. <Inside> is used to indicate if triggering is made inside or outside the window.

Example : MOVAP (Slave=1200.00,Master,0,0.1,True)

See also : MOVA, MOVR, STTA, STTR, STTI, MOVE_S

10-16-164- MOVAT – Absolute triggered movement

Syntax : MOVAT(<Axis>=<Distance>)

Description : This insruction make a movemebt on axis to an absolute position, launched by the basic instruction TRIGGER. Like that, you can preload the mouvement and launch it after, with instruction TRIGGER in a other task and in a time less than 1 ms.

See also : MOVA, MOVRT, TRIGGER

10-16-165- MOVC– Circular movement

Syntax : MOVC (<AxisX> = <DestinationX>, <AxisY> = <DestinationY>, <CenterX>,<CenterY>, <Clockwise>)

Accepted types : DestinationX, DestinationY, CenterX, CenterY : real

Clockwise : Bit

Description : This instruction is used to make a circular movement with two axes. The movement is send to the buffer of movement. If the buffer of movement is full, the task is blocked until a place is liberated. The execution of this instruction launches the execution of the next task.

Remarks : <AxisX> and <AxisY> must be servo axes. <DestinationX> and <DestinationY> define the position to reach in 2D. <CenterX> and <CenterY> defines the center relative position. <Clockwise> defines rotation direction.

Example : MOVC(X=100,Y=100,100,0,1)

See also : MOVL, MOVE_S

10-16-166- MOVE_S –Movement state

Syntax : **MOVE_S**(<Axis>)

Accepted types : Bit

Description : This function indicates if the axis is in movement.

Remarks : <Axis> must be a servo axis. If the axis is in a open loop state (AXIS(AXE)=OFF), the instruction MOVE_S(Axe)=0. If the axis is in a closed loop state, MOVE_S(Axe) is equal to 0 if the 4 points below are true :

➪ the current movement is finished (theoretic trajectory is finished).

➪ the following error is inside the positioning window (+/-POSMIN_P(Axe)).

➪ the buffer of movement is empty.

➪ In the case of a slave axis linked with a CAM or GEARBOX function, the link will be cut.

If one of this point is false, the instruction MOVE_S(Axe) is set.

Example :

```
STTA (X=100)
WAIT NOT MOVE_S(X) ' Wait the axis is stopped
```

10-16-167- MOVL – Linear movement

Syntax : **MOVL** (<AxisX>=<DestinationX>, <AxisY>=<DestinationY>)

Accepted types : DestinationX, DestinationY : real

Description : This instruction is used to make a linear movement with two axes. The movement is send to the buffer of movement. If the buffer of movement is full, the task is

	blocked until a place is liberated. The execution of this instruction launches the execution of the next task.
Remarks :	<AxisX> and <AxisY> must be servo axes. <DestinationX> and <DestinationY> define the position to reach in 2D.
Example :	`MOVL(X=100,Y=100)`
See also :	MOVC, MOVE_S

10-16-168- MOVR – Relative movement

Syntax :	**MOVR**(<Axis>=<Distance> {,<Axis>=<Distance> ... })
Limits :	Distance : +/- 2^24 pulses
Accepted types :	Distance : real
Description :	Moves axis to a relative position. The execution of this instruction launches the execution of the next task.
Remarks :	The system waits for the movement end (Condition MOVE_S(Axe)=0) before executing the next instruction. The axes use the current speed, acceleration and deceleration values. <Axis> must be a servo axis.
Example :	`MOVR (X=1200.00,Y=-100.00,Z=+550.00,W=Dist!)`
See also :	MOVA, STTA, STTR, STTI, MOVE_S

10-16-169- MOVRT– Relative triggered movement

Syntax :	MOVRT(<Axis>=<Distance>)
Description :	This instruction make a movement on axis to a relative position launched by the basic instruction TRIGGER. Like that, you can preload the movement and launch it after, with the instruction TRIGGER in a other task and in an time less than 1 ms.
See also :	MOVR, MOVAT, TRIGGER

10-16-170- MOVS and MOVSP - Synchronised movement

Syntax 1 :	**MOVS**(<Slave Axis>,<Master Axis>,<MasterDistance>,<SlaveDistance>, <AccelerationDistance>, <DecelerationDistance>)
Syntax 2 :	**MOVSP**(<Slave Axis>,<Master Axis>,<MasterDistance>,<SlaveDistance>, <AccelerationDistance>, <DecelerationDistance>, <TriggerAxis>,<Mini>,<Maxi>,<Inside>)
Accepted types :	MasterDistance, SlaveDistance, AccelerationDistance, DecelerationDistance, Mini, Maxi : real Inside : Bit
Limits :	MasterDistance, SlaveDistance : 0 to +/-2^21 pulses
Description :	This instruction is used to link a slave axis to a master axis during a certain distance of the master axis with acceleration and deceleration phases on the slave axis.This instruction allows to start the synchronisation when a trigger axis enters in a position window. The movement is send to the buffer of movement. If the

buffer of movement is full, the task is blocked up to a place is liberated. The execution of this instruction launches the execution of the next task.

Remarks : <MasterAxis> can be a servo or encoder axis. <SlaveAxis> must be a servo axis. The <AccelerationDistance> and <DecelerationDistance> are expressed in distance on the master axis and can be zero. <Mini> and <Maxi> define the trigger window used to start the movement on the slave. <Mini> must be lower than <Maxi>. <Inside> is used to indicate if triggering is made inside or outside the window.

Example :
```
Flying shears
ORDER(Slave)=0    'initialise the n°+1 of next movement
                  'sends to the buffer
MOVS(Slave,Master,0.8,0.4,0.8,0)      'Acceleration movement 1
MOVS(Slave,Master,0.2,0.2,0,0)        'Synchro phase movement 2
MOVS(Slave,Master,0.8,0.4,0,0.8)      'Deceleration movement 3
MOVS(Slave,Master,8.2,-1.0,0,0.5)     'Back movement 4
WAIT ORDER_S(Slave)>=2   'Wait the execution end of movement 2
OUT(Couteau)=ON          'Activation of cutter
WAIT ORDER_S(Slave)>=3   ' Wait the execution end of movement 3
OUT(Couteau)=OFF         'Stop the cutter
```

See also : CAM, CAMBOX, GEARBOX

10-16-171- MOVSC – Synchronised movement triggered on registration input

Syntax : **MOVSC**(<SlaveAxis>,<MasterAxis>,<MasterDistance>,<SlaveDistance>,
<AcceleretionDistance>,<DecelerationDistance>,<Configuration>
[,<Trigerredaxis>,<Window>,<Mini>,<Maxi>,<Inside>])

Accepted types : MasterDistance, SlaveDistance, AccelerationDistance,
DecelerationDistance, Mini, Maxi : real

Configuration : Byte

- b0 : unused
- b1 : triggering on signal Z
- b2 : triggering on capture 1 input (C1)
- b3 : triggering on capture 2 input (C2)
- b4 : 1 : negative edge ; 0 : positive edge
- b5…b7 : unused

Description : This instruction is used to link the slave axis with the master axis during the distance of the master axis. This link is used with acceleration phases and deceleration phases on the slave axis. This instruction is triggered on a capture input. The movement is send to the buffer of movement. If the buffer of movement is full, the task is blocked up to a place is liberated. The execution of this instruction launches the execution of the next task. The parameters <TriggeredAxis>, <Window>, <Mini>, <Maxi> and <Inside> are optional.

Remarks : <Slave Axis> must be stopped (MOVE_S(Slave Axis)=0) before the instruction is send even the movement is obtained otherwise the condition is false. <Master Axis> can be a servo axis or an encoder. <Slave Axis> must be a servo axis. <AccelerationDistance> and <DecelerationDistance> are defined in a distance on the master axis and can be null. <Mini> and <Maxi> define the triggering window on the <TriggeredAxis>. <Inside> indicate that the triggered position is inside or outside the window. <Mini> must be lower than <Maxi>.

Example :

```
Flying shears
ORDER(Slave)=0     'initialise the n°+1 of next movement
                   'sends to the buffer
MOVSC(Slave,Master,0.8,0.4,0.8,0)     'Acceleration movement 1
MOVS(Slave,Master,0.2,0.2,0,0)        'synchro phase movement 2
MOVS(Slave,Master,0.8,0.4,0,0.8)      'Deceleration movement 3
MOVS(Slave,Master,8.2,-1.0,0.5)       'Back of movement 4
WAIT ORDER_S(Slave)>=2   'Wait execution of movement 2
OUT(Couteau)=ON          'Activate cutter
WAIT ORDER_S(Slave)>=3   'Wait execution end of movement 3
OUT(Couteau)=OFF         'Stop cutter
```

See also : CAM, CAMBOX, GEARBOX

10-16-172- MOVST– Synchronised movement triggered

Syntax : MOVST(<SlaveAxis>, <MasterAxis>, <MasterDistance>, <SlaveDistance>, <AccelerationDistance>, <DecelerationDistance>)

Description : This instruction make a synchronised movement launched by the basic instruction TRIGGER. Like that, you can preload the movement and launch it after, with the instruction TRIGGER in an other task and in a time less than 1 ms.

Voir aussi : MOVS, TRIGGER

10-16-173- NOT – Complement operator

Syntax : **NOT**(<Expression>)

Accepted types : Expression : Bit, Byte, Integer

Description : This function returns the complement.

Remarks : <Expression> must be an integer valid expression.

Example :

```
a%=0FF00h
b%=NOT a%  'Result b%=00FFh
```

See also : AND, OR, XOR

10-16-174- OPEN – Open a communication port

Syntax : **OPEN** <CommunicationPort> **AS #**<Number>

Description : Authorizes the reading/writing operations on a communication port.

Remarks : You must open a communication port before any input/output operation <CommunicationPort> is a char string which defines the parameters with this following syntax :

"SERIALn:[speed [, data[, parity [, stop]]]]"

N: Physical number 1 or 2

Speed: 150, 300, 600, 1200, 2400, 4800 or 9600 bauds.

Data : 7 or 8 bits

Parity : E (even), O (odd), M (mark), S (space) or N (without).

Stop : 1 or 2 bits

<Number> defines the communication canal number used by the functions.

Example :

```
Dialog 80, 160or 640 linked to SERIAL2 : SERIAL2 affected to the canal 1
OPEN "SERIAL2:9600,8,N,1" As #1
PRINT "<MAIN MENU>";
```

See also : INPUT, PRINT, CLOSE

10-16-175- OR – OR operatorr

Syntax :	<Expression1> **OR** <Expression2>
Accepted types :	Expression1, Expression2 : Bit, Byte, Integer
Description :	This function makes a binary OR between two expressions.
Remarks :	<Expression1> and <Expression2> must have the same type. This function returns the same data type as its arguments.
Example :	`A%=A% OR 000FFh`
See also :	AND, NOT, XOR and IF

10-16-176- ORDER – Movement order number

Syntax 1 :	**ORDER**(<Axis>) = <Expression>
Syntax 2 :	**ORDER**(<Axis>)
Accepted types :	Expression : Integer
Description :	This instruction fixes the order number of the next movement or reads the order number of the last deposed movement.
Remarks :	This instruction can be used with the ORDER_S function.

Example :

```
ORDER(X)=0
MOVS(X,Master,50,100,10,10)
A#=ORDER(X)  'Result : A#=1
```

See also : ORDER_S

10-16-177- ORDER_S – Current order number

Syntax :	**ORDER_S**(<Axis>)
Accepted types :	Integer
Description :	This function returns the movement number which is in execution.
Remarks :	This function can be used to know a movement state.

Example :

```
ORDER(X)=0
MOVS(X,Master,50,100,10,10)
MOVS(X,Master,50,100,10,10)
MOVS(X,Master,50,100,10,10)
IF ORDER_S(X)=2 THEN ...'The second movement has begun.
```

See also : ORDER

10-16-178- OUT – Output writing

Syntax : **OUT** (<Output>) = <Expression>

Accepted types :	Expression : Bit
Description :	This function sends a logical state to a digital output.
Remarks :	<Output> must represent an output name.
Example :	`OUT(Cutter)=ON`
See also :	INP, INPB, INPW, OUTB, OUTW

10-16-179- OUTEMPTY – Communication output buffer empty

Syntax :	<Expression>=**OUTEMPTY** (<Number>)
Accepted types :	<Expression> : bit
Description :	This function returns communication output buffer state
Remarks :	<Number> is number used to open communication port with the OPEN function.
Example :	`WAIT  OUTEMPTY(#1)`
See also :	CARIN, CAROUT

10-16-180- OUTB – 8 outputs writing

Syntax :	**OUTB** (<Outputs>) = <Expression>
Accepted types :	Expression : Byte
Description :	This function sends logical states to a 8 logical outputs block
Remarks :	<Outputs>must represent the name of a 8 outputs bloc.
Example :	`OUTB(Bloc1)=0Fh`
See also :	INP, INPB, INPW, OUT, OUTW

10-16-181- OUTW – 16 outputs writing

Syntax :	**OUTW** (<Outputs>) = <Expression>
Accepted types :	Expression : Integer
Description :	This function sends logical states to a 16 logical outputs block
Remarks :	<Outputs> must represent the 16 outputs block name .
Example :	`OUTW(Bloc1)=0FFFFh`
See also :	INP, INPB, INPW, OUT, OUTB

10-16-182- PIXEL – Draw point

Syntax :	**PIXEL**(X,Y,Color)
Units :	X, Y : pixel
Limits :	X : 1 to 240
	Y : 1 to 128
Accepted types :	X,Y : byte.
	Color : Bit
Description :	This function draws a point at coordinates X, Y on the operator panel Dialog 640.
Remarks :	Colour define the colour of the point : black (0) or white (1)

Example : `PIXEL(23,15,0) 'Draw a black pixel at coordinates 23,15`

10-16-183- PLCINIT – PLC function initialisation

Syntax : PLCINIT(<Input table>,< Previous input table>, <Output table>, <Masked output table>)

Description : This function indicate to the system, the variable table to use.

Remark : <Input tablea>, <Previous input table> : Long integer table with any elements as that the system contained the input cards.

<Output table>, <Masked output table> : Integer table with any elements as the system contained the output cards.

<Masked output table> contained the output masks use by the PLC (bit to 1 => output use by the PLC)

exemple : Masque[1]=0FFFFh

Masque[2]=0FFFFh

PlcInit(Entrees,EntreesOld,Sorties,Masque)

See also : PLCINP, PLCINPB, PLCINPW, PLCINPPE, PLCINPNE, PLCOUT, PLCOUTB, PLCOUTW

10-16-184- PLCINP –Read TOR input

Syntax : PLCINP (<Input>) or PLCINP (<Card number>, <Input number>)

Accepted types : <Input> : Bit

<Card number>, <Input number> : Byte

Description : This functionn give the state of PLC TOR input.

Remarks : <Input> must represent a TOR input name. The data type returned is a bit.

Exemple : C~=PLCINP(CouteauEnHaut)

See also : PLCINIT, PLCINPB, PLCINPW, PLCINPPE, PLCINPNE, PLCOUT, PLCOUTB, PLCOUTW

10-16-185- PLCINPB – Read a 8 inputs block

Syntax : PLCINPB (<Inputs>)

Accepted types : Inputs : Byte

Description : This function return the state of a block of 8 TOR inputs.

Remarks : <Inputs> must represente the name of 8 inputs. The data’s type returned is a byte.

Exemple : B#=PLCINPB(Data)

See also : PLCINIT, PLCINP, PLCINPW, PLCINPPE, PLCINPNE, PLCOUT, PLCOUTB, PLCOUTW

10-16-186- PLCINPNE – Read a negative edge on PLC TOR input

Syntax : PLCINPNE (<Input>) or PLCINPNE (<Card number>, <Input number>)

Accepted types : <Input> : Bit

<Card number>, <Input number> : Byte

Description :	This function indicate if a negative edge is make on the PLC TOR input.
Remarks :	<Input> must represente the name of a TOR input. The data's type returned is a Bit.
Exemple :	If PLCINPNE(CouteauEnHaut) Then goto FrontDetecte
See also :	PLCINIT, PLCINP, PLCINPB, PLCINPW, PLCINPPE, PLCOUT, PLCOUTB, PLCOUTW

10-16-187- PLCINPPE – Read a positive edge on PLC TOR input

Syntax :	PLCINPPE (<Input>) or PLCINPPE (<Card number>, <Input number>)
Accepted types :	<Input> : Bit
	<Card number>, <Input number> : Byte
Description :	This function indicate if a positive edge is make on the PLC TOR input.
Remarks :	<Input> must represente the name of a TOR input. The data's type returned is a Bit.
Exemple :	If PLCINPPE(CouteauEnHaut) Then goto FrontDetecte
See also :	PLCINIT, PLCINP, PLCINPB, PLCINPW, PLCINPNE, PLCOUT, PLCOUTB, PLCOUTW

10-16-188- PLCINPW

Syntax :	PLCINPW (<Inputs>)
Accepted types :	Inputs : Byte
Description :	This function return the state of a block of 16 TOR inputs.
Remarks :	<Inputs> must represente the name of 16 inputs. The data's type returned is a integer.
Exemple :	A%=PLCINPW(Data)
See also :	PLCINIT, PLCINP, PLCINPB, PLCINPPE, PLCINPNE, PLCOUT, PLCOUTB, PLCOUTW

10-16-189- PLCOUT – Write a output

Syntax :	PLCOUT (<Output>) = <Expression> or
	PLCOUT (<Card number>, <Ouput number>) = <Expression>
Accepted types :	Expression : Bit
	<Card number>, <Output number> : Byte
Description :	This function change the logic state of image bit.
Remarks :	<Output> must represente the name of an output
Exemple :	PLCOUT(Couteau)=ON
	…
	If PLCOUT(Voyant) Then goto Alarm
See also :	PLCINIT, PLCINP, PLCINPB, PLCINPW, PLCINPPE, PLCINPNE, PLCOUTB, PLCOUTW

10-16-190- PLCOUTB – Write a 8 outputs block

Syntaxe : PLCOUTB (<Output>) = <Expression>

Accepted types : Expression : Byte

Description : This function change the logic state of the 8 associates images outputs.

Remarks : <Outputs> must represente the name of 8 outputs blocks.

Exemple : PLCOUTB(Bloc1)=0Fh

See also : PLCINIT, PLCINP, PLCINPB, PLCINPW, PLCINPPE, PLCINPNE, PLCOUT, PLCOUTW

10-16-191- PLCOUTW – Write a 16 outputs block

Syntaxe : PLCOUTW (<Output>) = <Expression>

Accepted types : Expression : Integer

Description : This function change the logic state of the 16 associates images outputs.

Remarks : <Outputs> must represente the name of 16 outputs blocks.

Exemple : PLCOUTW(Bloc1)=0FFFFh

See also : PLCINIT, PLCINP, PLCINPB, PLCINPW, PLCINPPE, PLCINPNE, PLCOUT, PLCOUTB

10-16-192- PLCREADINPUTS – Read the PLC inputs

Syntaxe : PLCREADINPUTS

Description : This function read the PLC inputs and memorize them into the images bits table.

See also : PLCINIT, PLCINP, PLCINPB, PLCINPW, PLCINPPE, PLCINPNE, PLCOUT, PLCOUTB, PLCOUTW

10-16-193- PLCWRITEOUTPUTS – Write the PLC outputs

Syntax : PLCWRITEOUTPUTS

Description : This function write the PLC ouputs memorized into the images bits.

See also : PLCINIT, PLCINP, PLCINPB, PLCINPW, PLCINPPE, PLCINPNE, PLCOUT, PLCOUTB, PLCOUTW

10-16-194- POS – Position to reach

Syntax 1 : **POS**(<Axis>) = <Expression>

Syntax 2 : **POS**(<Axis>)

Accepted types : Expression : real

Description : This function returns or fixes the current position to reach in the chosen unit.

Remarks : This function is used to change the final position of the current motion. The position can be modified at any time.

Example :

```
STTA(X=5000)           'Axis start
WAIT INP(Cell)=On 'Wait for the cell
POS(X)=POS_S(X)+50.    'Stop 50mm after the sensor
WAIT NOT MOVE_S(X)     'Wait for the axis stop
```

See also : ACC, DEC, VEL

10-16-195- POS_S – Real position

Syntax : <Expression> = **POS_S**(<Axis>)

Accepted types : Expression : real

Description : This function returns the real position of the axis.

Remarks : <Axis> can be a servo axis, an encoder or a time based encoder. Thus, we can obtain the image of the axis position in real time.

Example :

```
STTA (X=100)           'Axis start
REPEAT
    LOCATE 1,1         'Cursor positioning
    PRINT POS_S(X);    'Display the axis position
UNTIL NOT MOVE_S(X)
```

See also : VEL_S

10-16-196- POWERFAIL – Power fail detect

Syntax : **POWERFAIL**= <ON|OFF>

Description : This function activates or inhibits power fail detect.

Remarks : Power fail detect is activated at power-on.

10-16-197- PRINT – Writing on a communication port

Syntax : **PRINT** [#<Number>], <Expression> [{ [; | ,] <Expression>}] [; | ,]

Description : Writes data on a communication port.

Remarks : <Number> is the number used to open the communication port with the OPEN instruction. A semicolon at the end of this instruction means that the previous character is printed immediately after the last character. A comma means that the next character is printed at the next line (by adding a line feed). Print is equal to Print #1. If a real expression is printed then decimal part is not printed and Format$ function must be used. if the transmit buffer is full, the task is blocked and continues when a place in the transmit buffer is liberated.

Example :

```
PRINT #1,A$,B%
PRINT "LENGTH"
```

See also : OPEN, PRINT, CLOSE

10-16-198- PROG – Program start

Syntax : **PROG**

Description : This keyword begins a main program bloc. It is as well used to identify the end of the main program block when it is preceded by END. <Name> is optional.

Remarks : One and only one PROG - END PROG bloc must be defined in a program.

Example :

```
PROG
...
END PROG
```

See also : END

10-16-199- RAMOK – Test ram status

Syntax : **RAMOK**

Description : This function indicates if at the last start-up of the MCS, the RAM data checksum was valid.

Remarks : If RAMOK=1, start-up valid

If RAMOK=0 and data flash copy zone is not blank, the MCS backups the data flash zone in the ram zone and starts the task. If RAMOK=0 and data flash copy zone is blank, the MCS doesn't start the task and indicates an error 20 on the status display.

See also : FLASHOK, RAMTOFLASH, FLASHTORAM

10-16-200- RAMTOFLASH – Backup saved variables

Syntax : **RAMTOFLASH**

Description : This function backups parameters and the first 10000 saved variables in flash memory.

See also : RAMOK, FLASHTORAM, FLASHOK

10-16-201- REALTOLONG – Convert a real to a long integer

Syntax : **REALTOLONG**(<Expression>)

Accepted types : Expression : real

Description : This function converts a real type data in a long integer type data.

Example :

```
A!=Edit(1,1,4,0,0)
B&=RealToLong(A!)
```

10-16-202- REALTOINTEGER – Convert a real to an integer

Syntax : **REALTOINTEGER**(<Expression>)

Accepted types : Expression : real

Description : This function converts a real type data in an integer type data.

Example :

```
A!=Edit(1,1,4,0,0)
B%=RealToInteger(A!)
```

10-16-203- REALTOBYTE – Convert a real to a byte

Syntax : **REALTOBYTE**(<Expression>)

Accepted types : Expression : real

Description : This function converts a real type data in byte type data.

Example :

```
A!=Edit(1,1,4,0,0)
B#=RealToInteger(A!)
```

10-16-204- REGPOS_S– Captured position

Syntax : <Expression>=**REGPOS_S**(<Axis>)

Accepted types : Expression : real

Description : This function returns the last captured position on the axis.

Remarks : <Axis> must be a servo axis.

Example :

```
CAPTURE(X,Off,On,10,20,On)
WAIT REG_S(X)
XPosition! = REGPOS_S(X)
```

See also : REG_S, CAPTURE, REGPOS1_S, REGPOS2_S

10-16-205- REGPOS1_S – Captured position

Syntax : <Expression>=**REGPOS1_S**(<Axis1>,<Axis2>)

Accepted types : Expression : real

Description : This function returns the last captured position on the axis with the execution of the CAPTURE1 instruction.(SRV85 only)

Remarks : <Axis1> where is located the capture entry.

<Axis2> where we want to capture the position.

Example :

```
CAPTURE1(X,X,4,On,10,20,On)    ' Capture position X on a
WAIT REG1_S(X)                 ' positive edge of input C1 on
XPosition!= REGPOS1_S(X)       ' axis  X selection target.
```

See also : REG_S, CAPTURE, REGPOS_S, REGPOS2_S

10-16-206- REGPOS2_S – Captured position

Syntax : <Expression>=**REGPOS2_S**(<Axis>)

Accepted types : Expression : real

Description : This function returns the last captured position on the axis with the execution of the CAPTURE2 instruction.(SRV85 only)

Remarks : <Axis> must be a servo axis.

Example :

```
CAPTURE2(X,24,Off,0,0,Off)
WAIT REG2_S(X)
XPosition!= REGPOS2_S(X)
```

See also : REG_S, CAPTURE, REGPOS_S, REGPOS1_S

10-16-207- REG_S – Capture state

Syntax : **REG_S**(<Axis>)

Description : This function indicates if a capture position was realized.

Remarks : <Axis> must be a servo axis. The return value is true only once by capture. REG_S is automatically cleared on reading operation and when its value is 1. When another capture is made and if REG_S value is 1, then REG_S is cleared.

Example :

```
CAPTURE(X,Off,On,10,20,On)
WAIT REG_S(X)            'REG_S is automatically cleared
XPosition! = REGPOS_S(X)
```

See also : REGPOS_S, CAPTURE, REG1_S, REG2_S

10-16-208- REG1_S – Capture state

Syntax : **REG1_S**(<Axis>)

Description : This function indicates if a capture position was realized. (execution of the CAPTURE1 instruction with SRV85 only)

Remarks : <Axis> must be a servo axis. The return value is true only once by capture. REG1_S is automatically cleared on reading operation and when its value is 1. When another capture is made and if REG1_S value is 1, then REG1_S is cleared.

Example :

```
CAPTURE1(Y,X,4,On,10,20,On)
WAIT REG1_S(X)           'REG1_S is automatically cleared
XPosition!= REGPOS1_S(X)
```

See also : REGPOS_S, CAPTURE, REG_S, REG2_S

10-16-209- REG2_S – Capture state

Syntax : **REG2_S**(<Axis>)

Description : This function indicates if a capture position was realized. (execution of the CAPTURE2 instruction with SRV85 only)

Remarks : <Axis> must be a servo axis. The return value is true only once by capture. REG2_S is automatically cleared on reading operation and when its value is 1. When another capture is made and if REG2_S value is 1, then REG2_S is cleared.

Example :

```
CAPTURE2(X,24,Off,0,0,Off)
WAIT REG2_S(X)           'REG_S is automatically cleared
XPosition!= REGPOS2_S(X)
```

See also : REGPOS_S, CAPTURE, REG_S, REG1_S

10-16-210- REPEAT – Repeat…Until

Syntax : **REPEAT**

{<Instructions>}

UNTIL <Condition>

Description : This structure allows to the system to execute a list of instructions in a loop as long as the given condition is wrong.

Remarks : In the structure REPEAT ... UNTIL the <Instructions> are executed at least once even if the condition is true. The execution of this instruction launches the execution of the next task.

Example :

```
a%=0
REPEAT
    PRINT #1,a%
    a%=a%*2
UNTIL a%>100
```

See also : WHILE

10-16-211- RESTART – Restart system

Syntax : **RESTART**

Description : This function restarts system.

Remarks : This can be used to test system start type : If RESTART function result is false then the system start with power-on and if RESTART function result is true, the system has been restarted by RESTART function.

10-16-212- RIGHT$ - String right part

Syntax : **RIGHT$**(<String>,<Number>)

Accepted types : String : Char string

Number : Integer

Description : This function returns the <Number> right characters of a string.

Remarks : To find the characters number in <String>, use LEN(<String>).

Example :

```
a$="Sensor1"
b$=RIGHT$(a$,1)  'Result : b$="1"
```

See also : LEFT$

10-16-213- RTRIM$ - Remove the right spaces

Syntax : **RTRIM$** (<Expression>)

Accepted types : Expression : Char string

Description : Returns a string copy without the right spaces.

Example :

```
a$="    Menu   "
b$=LTRIM$(a$)           ' Result b$="    Menu"
```

See also : LTRIM$

10-16-214- RUN – Launch a task

Syntax : **RUN** <Name>

Description : This instruction is used to launch a stopped task (ex : task declared with manual start).

Remarks : This function has no effects on a suspended, automatic running tasks or already launched task.

Example :

```
Beginning:
    Wait Inp(Power)=On
    RUN  Cutter
    Wait Inp(Power)=Off
    HALT Cutter
Goto Beginning
```

See also : CONTINUE, HALT, SUSPEND

10-16-215- SECURITY – Define the security actions

Syntax :	**SECURITY**(<OpenLoop>,<OpenWatchDog>)
Description :	This instruction is used to define how the system must react if a following error on an axis is found. <OpenLoop> defines if all the axis loops must be opened. <OpenWatchDog> indicates if the watchdog must be opened. The default power-on values are SECURITY(On,On)
Remarks :	If the SECURITY instruction is used, the security level can be decreased following the program writing. It is advised not using this instruction.
Exemple :	`SECURITY(ON,OFF)  'WatchDog is not opened on a following error`

10-16-216- SEEK – Moving to a save file

Syntax 1 :	SEEK #3,<Long moving>
Syntax 2 :	<Variable> = SEEK #3
Accepted types :	<Long moving>, <Variable> : long integer
Description :	The syntax 1 allow to moving in the save file of <Long moving> characters. The moving start at the current position. The syntax 2 allow to know the current position in the save file.
Remarks :	The first character is at the position 0l.
Exemple :	P&=Seek(#3) ‘Rapport nominal : ratio 0.5
	Seek #3, P&+100 ‘Déplacement sur le 100ème caractère à partir
	‘de la position courante
See also :	OPEN, INPUT$, PRINT

10-16-217- SENSOR_S – Sensor state

Syntax :	**SENSOR_S** (<Axis>)
Description :	This function gives the state of the axis board home/capture input.
Example :	`IF SENSOR_S(X) THEN ...`
See also :	SENSOR1_S, SENSOR2_S

10-16-218- SENSOR1_S – C1 Sensor state (SRV85)

Syntax :	**SENSOR1_S** (<Axis>)
Description :	This function gives the state of the axis board home/capture C1 input of the SRV85 board.
Example :	`IF SENSOR1_S(X) THEN ...`
See also :	SENSOR_S, SENSOR2_S

10-16-219- SENSOR2_S – C2 Sensor state (SRV85)

Syntax :	**SENSOR2_S** (<Axis>)
Description :	This function gives the state of the axis board home/capture C1 input of the SRV85 board
Example :	`IF SENSOR2_S(X) THEN ...`

See also : SENSOR_S, SENSOR1_S

10-16-220- SETDATE – Set the date

Syntax : **SETDATE**(<Year>,<Month>,<Day>,<DayInTheWeek>)

Accepted types : Year, Month, Day, DayInTheWeek : Integer

Description : This instruction set the current date.

See also : GETDATE, SETTIME

10-16-221- SETINP – Input filters and invert

Syntax : **SETINP** (<Name>,<Inversion>,<Filtre>)

Units : Filter : milliseconds

Accepted types : Inversion : Long integer

Filter : Byte

Description : This function defines the inputs invert mask and the filter period.

Remarks : <Invert> is a long integer in which each bit represents the invert of each input. This parameter can be defined during the input card configuration.

Example :

```
SETINP(INPUTS11,4,10)    ' Second input card invert
                         ' and 10 ms filter
```

10-16-222- SETOUT – Outputs invert

Syntax : **SETOUT** (<Name>,<Inversion>)

Accepted types : Inversion : Long integer

Description : This function defines the outputs invert mask.

Remarks : <Invert> is a long integer in which each bit represents the invert of each output. This parameter can be defined during the output card configuration.

Example :

```
SETOUT(OUTPUTS1,3)       ' 2 first outputs card invert
```

10-16-223- SETTIME – Set the hour

Syntax : **SETTIME**(<Hours>,<Minutes>,<Seconds>)

Accepted types : <Hours>, <Minutes> and <Seconds> : Integer.

Description : This instruction set the current hour.

See also : GETTIME, SETDATE

10-16-224- SETUPCOUNTER – Counter configuration

Syntax : **SETUPCOUNTER**(<Axis>,<Input>,<Invert>,<Filter>)

Accepted types : <Input> : Byte

<Invert>, <Filter> : bit

Description : This instruction defines the counter configuration

Remarks : <Axis> : Name of the servo card

<Input> : Input number (1 for C1 input, 2 for C2 input)

	<Inversion> : edge choice : 0 for a positive edge, 1 for a negative edge
	<Filter> : Filter validation : 0 without filter, 1 for a 2ms filter.
See also :	COUNTER_S , CLEARCOUNTER

10-16-225- SGN - Sign

Syntax :	**SGN** (<Expression>)
Accepted types :	Expression : Long integer, real
Description :	This function returns a real equal to –1 for the negative numbers, 1 for the positive numbers and 0 for the zero number.
Example :	`a!=SGN(10) 'Result : a!=1`

10-16-226- SIGNAL – Event generation

Syntax :	**SIGNAL** <Name>
Description :	This instruction generates an event.
Remarks :	<Name> must be the name used by WAIT EVENT instruction. The only first task which was waiting for this event can then continue.

Example :

```
Program1                 Program2
...                      ...
WAIT EVENT Ready         SIGNAL Ready
...                      ...
```

See also : WAIT EVENT, DIFFUSE

10-16-227- SIN - Sine

Syntax :	**SIN** (<Expression>)
Accepted types :	Expression : real
Description :	This instruction returns the sine of <Expression>. <Expression> is expressed in radians.
Example :	<Expression> must be a numerical expression.
See also :	COS, ARCTAN, TAN

10-16-228- SLAVEOFFSET – Shift dynamically the slave position

Syntax :	SLAVEOFFSET(<Slave>,<Master>,<Offset>,<Acceleration>)
Limits :	<Offset> : Between 0 ans the slave modulo
Accepted types :	<Offset> : Real
	<Acceleration> : Real
Description :	This instruction shift dynamically the slave position by an absolute cambox.
Remark :	<Slave> : Name of the slave's axis where is donig the cambox (servo module : SRV85)
	<Master> : Name of the slave's axis (servo or encoder module : SCD22, SRV15, SRV85...)

	<Offset> : Offset value to apply
	<Acceleration> Used acceleration for apply the offset (increment/T0²).
See also :	LOADABSCAMEX, ABSCAM, STARTABSCAM, MASTEROFFSET

10-16-229- SPACE$ - Space made string

Syntax :	**SPACE$**(<Number>)
Limits :	Number : 1 to 255
Accepted types :	Number : Byte
Description :	This function returns a space made string.
Example :	`a$=SPACE$(10) 'Result a$="          "`
See also :	STR$, VAL

10-16-230- SQR – Square root

Syntax :	**SQR** (<Expression>)
Accepted types :	Expression : real
Description :	This function returns the square root of <Expression>.
Example :	`a!=SQR(2)`

10-16-231- SSTOP – Axis stop

Syntax :	**SSTOP**(<Axis>[,<Axis>]...)
Description :	This function stops <Axis> with the current deceleration. The function is not waiting for the end of the deceleration phase and that MOVE_S(Axe) is equal to zero.
Remarks :	If <Axis> is a master axis linked with the CAM, GEARBOX or MOVS functions then <Axis> and its slave are stopped but the link is not broken.
	If <Axis> is a slave axis linked with the CAM, GEARBOX or MOVS function, then <Axis> is stopped and is unlinked with the master axis.
	SSTOP clears the buffer of movements and stops the axis with the current deceleration.
Example :	SSTOP(Master,Slave)
See also :	STTA, STTR, STTI, CAM, GEARBOX, MOVS

10-16-232- STARTABSCAM – Start an absolute cambox

Syntax :	STARTABSCAM(<Slave>,<Master>,<Cam number>)
Limits :	<Cam number> : 1 to 5
Accepted types :	<Com number> : Bytet
Description :	This instruction launch an absolute cambox.
Remarks :	<Slave> : Name of the slave axis where is doing the cambox (servo module: SRV15, SRV85...)
	<Master> : Name of the master axis (servo or encoder module: SCD22, SRV15, SRV85...)

	<Cam number> : Cam number of the slave servo module.
See also :	LOADABSCAMEX, CAM_S

10-16-233- STARTCAM – Launches the execution of a cam

Syntax :	**STARTCAM**(<Slave>,<Master>,<NumberCam>)
Limits :	<NumberCame> : 1 to 5
Accepted types :	<NumberCame> : Byte
Description :	this instruction launches the execution of a cam in a servo board.
Remarks :	<Slave> : Slave axis cam for which the cam will be applied (servo board: SRV15, SRV85...)
	<Master> : Master axis name (servo or encoder board : SCD22, SRV15, SRV85...)
	<NumberCam> : number of the cam (from 1 to 5) of the servo board <Slave>
See also :	LOADCAMEX, CAM_S

10-16-234- STARTCAMBOX – Start a cambox

Syntax :	**STARTCAMBOX**(<Number>)
Description :	This instruction start a defined cambox>
Remarks :	If cambox is not defined this instruction have no effects. <Number> is the number used by CAMBOX function.
Example :	STARTCAMBOX(1)
See also :	CAMBOX

10-16-235- STARTCAMC - Launches the execution of a cam on capture input

Syntax :	STARTCAMC (<SlaveAxis>, <MasterAxis>, <CamNumber>, <Configuration>, [,<Trigerredaxis>, <Windows>, <Mini>, <Maxi>, <Insode>])
Accepted type :	<Configuration> : byte b0 : unused b1 : triggering on signal Z b2 : triggering on capture 1 input (C1) b3 : triggering on capture 2 input (C2) b4 : 1 : negative edge ; 0 : positive edge b5…b7 : unused <Mini>, <Maxi> : réel <Fenêtre>, <Intérieur> : bit
Description :	This instruction equivalent to STARTCAM with a launch condition « capture input». The starting can be do with a window of position. Parameters < Trigerredaxis>, <windows>, <Mini>, <Maxi> et <Inside> are optionals.
Remarks :	<MasterAxis> can be a servo axis, an encoder or temporal encodeur.

<SlaveAxis> must be an servo axis.

<SlaveAxis> must be at stop (MOVE_S(AxeEsclave)=0) before send the order otherwise the movement can be launch even if the condition is not true.

See also : CAM, CAMC, STARTCAM

10-16-236- STARTCAMT - Execution of a cam triggered

Syntax : STARTCAMT(<Esclave>,<Maitre>,<N°came>)

Description : This instruction make a cambox movement movement launched by the basic instruction TRIGGER. . Like that, you can preload the movement and launch it after, with the instruction TRIGGER in an other task and in a time less than 1 ms.

See also : STARTCAM, TRIGGER

10-16-237- STARTS – Launch a synchronized movement

Syntax : **STARTS**(<Axis>,[<Axis>]...)

Description : This instruction launches a synchronized movement already loaded with LOADS in the buffer of movements.

Remarks : LOADS and STARTS must be used instead of MOVS to begin simultaneously some movements.

Example :

```
LOADS(Slave1,Master,90,180,10,10)  'movement 1 loaded
                                   'in the buffer of movement
LOADS(Slave1,Master,90,180,10,10) ' movement 2 loaded
                                   'in the buffer of movement
STARTS(Slave1,Slave2)              'execution of the movement 1
```

See also : MOVS, STARTS

10-16-238- STATUS – Task state

Syntax : **STATUS** (<Name>)

Description : This function returns a task state

Remarks : The possible values are :

0 : The task is stopped

1 : The task is suspended

2 : The task is running

Example :

```
Run Cutter
Wait Status(Cutter)=0
```

10-16-239- STOP – Axis stop

Syntax : **STOP**(<Axis>[,<Axis>]...)

Description : This function stops <Axis> with the current deceleration. The function waits for the end of deceleration phase.

Remarks :	If <Axis> is a master axis linked with the CAM, GEARBOX or MOVS functions then <Axis> and its slave are stopped but the link is not broken. If <Axis> is a slave axis linked with the CAM, GEARBOX or MOVS functions, then <Axis> is stopped and the link with the master axis is broken. This instruction clears the buffer of movements and stops the axis with its current deceleration. STOP is a blocked instruction which is waiting MOVE_S(Axis) different of zero.
Example :	`STOP(Master)`
See also :	STTA, STTR, STTI, CAM, GEARBOX, MOVS

10-16-240- STOPCAMBOX – Stop a cambox

Syntax :	**STOPCAMBOX**(<Number>)
Description :	This instruction stops a cambox.
Remarks :	<Number> is the number used by CAMBOX function. This function doesn't destroy the cambox.
Example :	STOPCAMBOX(1)
See also :	CAMBOX, CAMBOXSEG, STARTCAMBOX

10-16-241- STOPCORRECTION– Stop the correction function

Syntax :	**STOPCORRECTION** (<Slave>)
Description :	This function stops a correction function defined in a continue mode.
Remarks :	<Slave> must be a SRV85 servo card.
See also :	CORRECTION, ICORRECTION

10-16-242- STOPI – Axis stop in interpolation

Syntax :	**STOPI**(<XAxis>,<YAxis>)
Description :	This function stops <XAxis> and <YAxis> with the current interpolation deceleration. The function waits for the end of deceleration phase. The execution of this instruction launches the execution of the next task.
Example :	STOPI(X,Y)
See also :	STTA, STTR, STTI, CAM, GEARBOX, MOVS

10-16-243- STRING$ - String creation

Syntax :	**STRING$**(<Number>, <Code>)
Limits :	Number, Byte : de 0 à 255
Accepted types :	Number, Code : Byte
Description :	This function returns a char string whose characters have the same ASCII code.
Remarks :	We use STRING$ to create a string which is constituted of a repeated character. <Number> is a numerical expression which indicates the length of the returned string. <Code> is the ASCII code of the character used to build a string and a numerical integer expression between 0 and 255.
Example :	`a$=STRING$(10,"0") 'Result a$="0000000000"`

See also : STR$, VAL

10-16-244- STR$ - Char characters convert

Syntax : **STR$**(<Expression>)

Accepted types : Expression : Byte … Real

Description : This function returns a string which represents a numerical expression value.

Remarks : When the numbers are converted in text, a head space is always reserved for the sign of <Expression>. If <Expression> is positive, the string returned by Str$ contains a head space and the sign plus is insinuated.

Warning : This function can send back a value according to the notation of type with exponent. It's preferable to use the instruction FORMAT$ with number=1

Example :

```
a%=10
b$=STR$(a%)  'Result b$=" 10"
```

See also : VAL

10-16-245- STTA – Launch an absolute movement

Syntax : **STTA**(<Axis>=<Distance> {,<Axis>=<Distance>})

Limits : Distance : +/- 2^24 pulses

Accepted types : Distance : real

Description : This instruction launches a movement to an absolute position.

Remarks : The system doesn't wait for the end movement (Condition MOVE_S(Axe)=0) and executes the next instruction. The axis use the current speed, acceleration and deceleration values. <Axis> must be a servo axis.

Example :

```
STTA (X=1200.00,Y=-100.00)
WAIT(NOT MOVE_S(X)) AND (NOT MOVE_S(Y))
```

See also : MOVA, MOVR, STTR, STTI

10-16-246- STTI – Launch an infinite movement

Syntax : **STTI**(<Axis>=[+|-] {,<Axis> =[+|-], ... })

Description : This instruction launches an infinite movement.

Remarks : The system doesn't wait for the movement end and executes the next instruction. You must use the STOP or SSTOP instruction to stop the movement. <Axis> must be a servo axis. The axis use the current speed, acceleration and deceleration.

Example :

```
STTI (X=+, Y=-)   'Begins an infinite movement on X axis
                  'in the positive direction and on the Y axis
                  'in the negative direction
```

See also : MOVA, MOVR, STTA, STTR

10-16-247- STTR – Launch a relative movement

Syntax : **STTR**(<Axis>=<Distance> {,<Axis>=<Distance> ... })

Limits : Distance : +/- 2^24 pulses

Accepted types : Distance : real

Description : This instruction launches a relative movement.

Remarks : The system doesn't wait for the movement end (Condition MOVE_S(Axe)=0) and executes the next instruction. The axis use the current speed, acceleration and deceleration values. <Axis> must be a servo axis.

Example :
```
STTR (X=1200.00,Y=-100.00,Z=+550.00,W=Dist!)
```

See also : MOVA, MOVR, STTA, STTI

10-16-248- SUB – Subroutine

Syntax : **SUB** <Name>

Description : This keyword begins a subprogram block and is also used to define the end of a subprogram block when it is preceded by END.

Remarks : The blocs SUB - END SUB must be outside of a PROG -END PROG bloc .

Example :
```
SUB Move
...
END SUB
```

See also : END

10-16-249- SUSPEND – Suspend a task

Syntax : **SUSPEND** <Name>

Description : This instruction suspends a task in run

Remarks : This instruction has no effects on stopped tasks. All the movements in the buffer of movements are executed.

Example :
```
  Wait Inp(Start)
   RUN  Cutter
Begin:
   Wait Inp(Stop)
   SUSPEND Cutter
   Wait Inp(Start)
   CONTINUE Cutter
Goto Begin
```

See also : RUN, CONTINUE, HALT

10-16-250- TAN - Tangent

Syntax : **TAN** (<Expression>)

Accepted types : Expression : real

Description : This instruction returns the tangent of <Expression>. <Expression>is an angle expressed in radians.

Remarks : This argument <Expression> must be a numerical valid expression. The function TAN takes an angle and returns the ratio of two sides rectangle triangle. The ratio is the length of the opposite side of an angle divided by the length of the adjacent side of the angle.

Example :
```
a!=TAN(PI)
```

See also : SIN, ARCTAN,TAN

10-16-251- THEORICSRCK1 – Selection of the synchro's source

Syntax : TheoricSrcK1(<Axe esclave>)=<Expression bit>

Description : This funtion allow, in the case of the master's axis is a servo's axis, to synchronised on theoretic position and not on real position, so as to avoid to propagate his vibrations on slave's axis.

Exemple :

```
TheoricSrcK1(X)=1   'Sélection de la position théorique de
                    'l'axe X pour la synchro
Movs(Y,X,360,120,90,90)
```

10-16-252- TIME – Time base

Syntax : **TIME**

Description : This instruction returns a long integer which represents the number of milliseconds from the last power-on. This instruction allows to execute no-locking waits. At the start-up of the MCS, TIME is equal to zero and increases up to 2^31. Then, it passed to -2^31 and increases to 0. This cycle is about 24 days long.

Remarks : If the MCS is used more than 24 days, you can use the TIMER instruction to suppress the crossing of 2^31 to –2^31.

Example :

```
Tempo&=Time+5000  'loads 5s delay
WAIT  (INP(Start)=On) Or (Time>Time&)
'if the start input is not activate in the 5s,
'the program continues
```

See also : TIME$, TIMER

10-16-253- TIMER – Wide time base

Syntax : **TIMER**

Description : This instruction returns a real which represents the number of milliseconds from the last power-on. This instruction allows to execute no-locking waits. At the start-up of the MCS, TIMER is equal to zero and increases with a step equal to 0.001(ms).

Example :

```
Timer!=Timer+5.25        'loads 5.25 delay
WAIT  (INP(Start)=On) Or (Timer>Time!)
'if the start input is not activate in the 5.25s,
'the program continues
```

See also : TIME$, TIME, DATE$

10-16-254- TIME$ - Current hour

Syntax : **TIME$**

Description : This instruction returns a 8 chars string with hh:mm:ss form, where hh are the hours (00-23), mm are the minutes (00-59) and ss are the seconds (00-59).

See also : TIME, TIMER, DATE$

10-16-255- TRAJ - Trajectory

Syntax : **TRAJ**(<Parameter>=<Value> {,<Parameter>=<Value> ... })

Accepted types : Value : real

Description : This instruction makes a complex trajectory

Remarks : The system waits for the movement end et launch the next instruction. <Parameter> is POS(<Axis>) for an absolute position, VEL(<Axis>) for a speed, ACC(<Axis>) for an acceleration and DEC(<Axis>) for a deceleration. The axes use the current speed, acceleration and deceleration if they are not defined in the instruction TRAJ. <Axis> must be a servo axis.

Example :

```
MERGE(X)=On                               'Slow speed
TRAJ (POS(X)=1000.00, VEL(X)=Vfast)       'go to position 100,
TRAJ (POS(X)=1500.00, VEL(X)=Vlow)        ' without to stop the axis
MERGE(X)=Off
```

See also : MOVA, MOVR, STTA, STTI

10-16-256- TRIGGER – Launches a movement

Syntax : TRIGGER(<Axe/AxeEsclave>)

Description : This instruction start a preload movement by the instructions MOVAT, MOVRT, MOVST or inst_STARTCAMT with a time less than 1 ms.

Exemple :

```
inst_STARTCAMT(Slave,Master,1)
……
TRIGGER(Slave)
……
```

See also : MOVAT, MOVRT, MOVST, inst_STARTCAMT

10-16-257- TX485 – Modify RS485 output state

Syntax : **TX485**(<Number>)=<Expression>

Accepted types : Expression : Integer

Description : This function enable RS485 port output for a specified number of characters. If number is 0 then output is disabled.

Remarks : <Number> is number used to open communication port with the OPEN function. In RS485 mode all sent characters are also received.

Example :

```
TX485(#1)=10
```

10-16-258- UCASE$ - Uppercase

Syntax : **UCASE$**(<Expression>)

Accepted types : Expression : Char string

Description : This function returns a string, in which all the letters of the argument have been converted in uppercases.

Remarks : The argument <Expression> must be a char string. Only the lowercases letters are converted in uppercases ; the other letters are not modified.

Example :

```
a$="Sensor1"
```

	`b$=UCASE$(a$) 'Result : b$="SENSOR1"`
See also :	LCASE$

10-16-259- VAL – Convert a string in numeric

Syntax :	**VAL**(<Expression>)
Accepted types :	Expression : Char string
Description :	This function returns the numerical value of the string <Expression>.
Remarks :	The argument <Expression> is a char string which can be interpreted as a numerical value. The VAL function stops reading the string when the first character is not known. VAL doesn't know as well the spaces, tabulations and line jumps. The VAL function always returns a real data type.

Example :	`a$="10"`
	`b!=VAL(a$) 'Result b!=10`

See also :	STR$

10-16-260- VEL - Velocity

Syntax 1 :	VEL(<Axis>) = <Expression>
Syntax 2 :	VEL(<XAxis>, <YAxis>) = <Expression>
Units :	Expression : user unit per second (Ex : mm/s, tr/s, degree/s).
Accepted types :	Expression : real
Description :	This value specifies the current velocity in user unit per second.
Remarks :	<Expression> must be a valid real expression. This velocity value can be modified at any time. The second form is used for interpolation. It is not allowed to refresh Velocity more than ten times per second on an axis with a sine acceleration : An unusual behaviour will appear on the acceleration or deceleration phases.
Warning :	It's not advise to always refresh the velocity. Use a temporizing of 20ms minimum.
Example :	`VEL(X)=2000`
See also :	ACC, DEC, POS

10-16-261- VEL% - Set velocity in percent

Syntax :	**VEL%**(<Axis>) = <Expression>
Limits :	Expression : 0 to 100
Accepted types :	Expression : real
Description :	This function adjusts the current velocity in percent of the default velocity parameter VEL_P.
Remarks :	The value of VEL_P can be entered in the speed profile screen during the card configuration. It is not allowed to refresh Velocity more than ten times per second on an axis with a sine acceleration : An unusual behaviour will appear on the acceleration or deceleration phases.

Example :	`VEL_P(X)=2000`
	`VEL%(X)=20 'The current speed is 400 units / s`

See also :	ACC%, DEC%

10-16-262- VEL_S – Current Velocity

Syntax :	**VEL_S**(<Axis>)
Description :	This function returns the current velocity.
Remarques:	<Axis> must be a servo axis. The sending value is the picture of the real velocity for the SRV85 card and the picture of the calculated velocity for all other servo card.

Example :	`STTA (X=100)`
	`WHILE NOT MOVE_S(X) DO PRINT #1,VEL_S(X)`

See also :	POS_S

10-16-263- VERSION – Operating system version

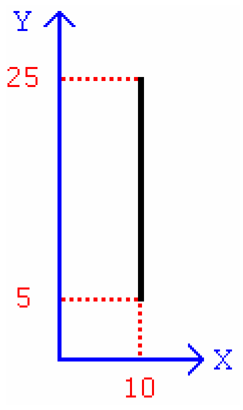
Syntax : <Variable>=VERSION

Accepted type : Variable : chaîne de caractères

Description : This function return a string with the version of the operating system.

10-16-264- VLINE – Draw a vertical line

Syntax :	**VLINE**(X1,Y1,Y2,color)
Units :	X1, Y1, Y2 : pixel
Limits :	X1 : 1 to 240
	Y1, Y2 : 1 to 128
Accepted types :	X1, Y1, Y2 : byte.
	Colour : Bit
Description :	This instruction draws a vertical line with its start point in X1, Y1 and its end point in X1, Y2 on the operator panel Dialog 640.
Remarks :	Colour defines the colour of the line : black (0) or white (1)
Example :	VLINE(10,5,25,0)



10-16-265- WAIT EVENT – Event waiting

Syntax : **WAIT EVENT** <Name>

Description : This instruction allows to the system to wait until an event is received. The execution of this instruction launches the execution of the next task.

Remarks : In the WAIT EVENT instruction, the following instructions are not executed if the event is not received. This instruction provides a passive wait for event.

Example :
```
WHILE Ready=False DO END WHILE        'Active waiting
'This program is similar to :
WAIT EVENT Ready                       'Passive waiting
```

See also : SIGNAL, DIFFUSE, WAIT STATE, DELAY

10-16-266- WAIT KEY – Key waiting

Syntax : **WAIT KEY**

Description : This function waits for a key pressed on the operator panel and record its code in the KEY variable. The execution of this instruction launches the execution of the next task.

Remarks : This function uses the communication port #1. By default, the communication port SERIAL1 will be used.If an operator panel is connected to the port SERIAL2, please refer you to the OPEN function to affect #1 to the port SERIAL2.

Example :
```
WAIT  KEY
IF KEY=@F1 THEN GOTO ...
IF KEY=@F2 THEN GOTO ...
...
```

10-16-267- WAIT – Condition waiting

Syntax : **WAIT** <Condition>

Description : This instruction allows to the system to wait for a condition. The execution of this instruction launches the execution of the next task.

Remarks : The WAIT instruction, the following instructions are not executed if the <Condition> is false. This instruction provides a passive wait for a condition. The STATE keyword is optional.

Example :
```
WHILE INP(Sensor)=Off  DO END WHILE   'Active waiting
'this program is similar to :
WAIT  INP(Sensor)=On                  'Passive waiting
```

See also : WAIT EVENT, DELAY

10-16-268- WATCHDOG – Watchdog

Syntax 1 : WATCHDOG = ON / OFF

Syntax 2 : WATCHDOG

Description : This function allows to the user to read or write the watchdog relay state.

Remarks : The watchdog state under power-on is OFF. Then, it must be set to ON when the program starts.The relay is automatically opened when an axis is in a following error. This function must be tested in a security task. The SECURITY function can also modify its behaviour.

Example :

```
WATCHDOG=ON.
WAIT WATCHDOG=OFF
```

10-16-269- WHILE – While…Do…End While

Syntax : **WHILE** <Condition> **DO**
{<Instructions>}
END WHILE

Description : This instruction allows to the system to execute a list of instructions in a loop as long as the given condition is true. The execution of this instruction launches the execution of the next task.

Remarks : In the WHILE ... DO ... END WHILE instruction, <Instruction>are not executed if the condition is false.

Example :

```
a%=0
WHILE a%<=100
    PRINT #1,a%
    a%=a%*2
END WHILE
```

See also : REPEAT

10-16-270- XOR – Exclusive OR operator

Syntax : <Expression1> **XOR** <Expression2>

Accepted types : Expression1, Expression2 : Bit, Byte, Integer

Description : This function makes a Exclusive Or between the expressions.

Remarks : <Expression1> and <Expression2> must represent a bit, a byte or an integer. <Expression1> and <Expression2> must have the same data type. This function returns the data type of <Expression1>.

Example :

```
IF A% XOR 0FF00h THEN ...
```

See also : AND, OR, NOT, IF

10-16-271- ZERO_S – Zero encoder state

Syntax : **ZERO_S** (<Axis>)

Description : This function indicates the zero encoder state of the axis card.

Example :

```
IF  ZERO_S(X) THEN STOP(X)
```

11- CANopen

11-1- Definition

11-1-1- Introduction

The CAN bus (Controller Area Network) appeared in the middle of the 80ies as an answer for the data transmission in the automotive fields. This kind of bus can have transmission speeds up to 1 Mb/s.

The CAN specifications are defining 3 layers among the ISO/OSI model: the physical one, the data linking one and the application one. The physical layer defines the data transmission mode regarding the transmission support. The data linking layer is the nucleus of the CAN protocol because it deals with the frame to send, the arbitrage, the defaults detection, etc. The last layer is also called CAL (CAN Application Layer). It is a general description of the language for the CAN networks which offers many communication services.

CANopen is a type of network based on the serial bus system and the application layer CAL. CANopen offers only part of the communication services that CAL has at its disposal. Those are the necessary advantages that need small performances computer, without storage capability.

So the CANopen is an application layer standardised by the CIA (CAN In Automation) specifications: DS-201…DS-207

The network manager permits an easier network initialisation. The network can be extended with all the components the user wants to.

The CAN bus is a multi-master bus. The sent messages are identified, instead of the connected modules as in the other field-buses. The network elements are allowed to send their message each time the bus is free. Bus conflicts are solved with a priority level given to messages. The CAN bus emits messages divided among 2032 priority levels. All the network elements have the same rights, so this communication is possible only without master bus.

Each element is deciding itself when it has data to send. However it is possible to send data with another element. This demand is made with the distant frame.

The CANopen specifications (DS-201…DS-207) define the technical and functional characteristics needed by any device to be plugged in the network. The CANopen bus makes a distinction from the server devices and the client devices.

11-1-2- CANopen communication

The CANopen communication profile permits to specify information for data exchange in real time and parameters. The CANopen uses optimised services following the data types:

↳ PDO (Process Data Object)

- ⇨ Data exchange in real time
- ⇨ High priority identifier
- ⇨ Synchronous or asynchronous transmission
- ⇨ 8 bytes (one message) maximum
- ⇨ Pre-defined format

↳ SDO (Service Data Object)

- ⇨ Access to the objects dictionary of a device
- ⇨ Low priority identifier

⇨ Asynchronous transmission

⇨ Data distributed in many telegrams

⇨ Data addressed with an index

The characteristics diffused on the CAN bus are received and evaluated by all the connected devices. Each service of a CAN device is configured by a COBID (Communication OBject IDentifier). The COBID is an identifier which characterises the message. It tells to a device if the message must be taken in account. For each service (PDO or SDO), it is necessary to specify a COBID during the emission (sending a message) and a reception COBID (receiving a message). For the first SDO server, the COBID is fix and can not be modified remotely. Moreover, it is calculated from the NodeID. The NodeID is the parameter which characterises the device and permits the unique access to it.

PDO (Process Data Object)

It is a data exchange arbitrated between 2 modules. The PDO can transfer in turn some synchronisation or controlled events to realise the message sending request. With the controlled events mode, the load of the bus can be very reduced. A device can therefore realise a high performance with a law transfer rate.

The data exchange with the PDO uses the CAN advantages:

⇘ Sending messages can be done from an asynchronous event (controlled event)

⇘ Sending messages can be done from the reception of a synchronisation event.

⇘ Recovery from a remote frame.

SDO (Service Data Object)

It is a data exchange point to point. A device is asking for an access in the objects list of a SDO. This one sends back an information corresponding to the type of request made by the caller. Each SDO can be either client and / or server. A server SDO can not send a request to another SDO, but it can answer any request from another client SDO. Unlike the PDOs, the SDOs must follow a particular communication protocol . The frame to send must have 8 bytes :

⇘ Domain Protocol (Byte 0) : it defines the command (Upload, Download,….)

⇘ Index on 16 bits (Bytes 1 et 2) : It defines the objects dictionary address.

⇘ Sub-index on 8 bits (Octet 3) : It defines the element of the selected object in the dictionary

⇘ Parameter (Octet 4 à 7) : It defines the value of the parameter read or written.

The network manager has a simplified mode to start the network up. The network configuration is not necessary in all the cases. The default configuration of the parameters may be enough. If the user wants to optimise the CANopen network or increase its functionalities, he can the modify himself these parameters. In the CANopen networks, each device has the same rights and the data exchange is directly regulated between each participant device.

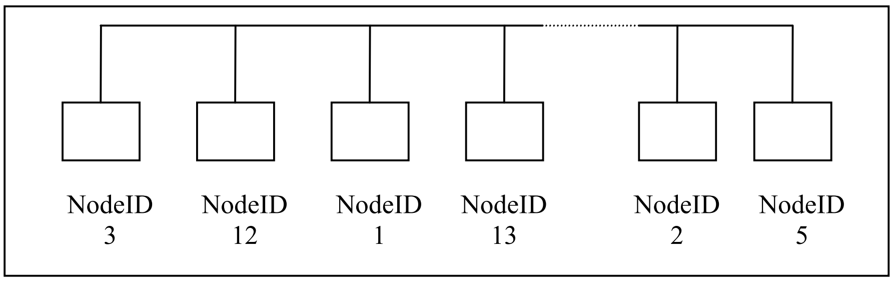
The profile of a device defines the necessary parameters for a communication. The contents of this profile is specified by the constructor. Devices with the same profile are directly interchangeable. Most of the parameters are described by the constructor. The profile has empty places too which are for the future functionality extensions.

In most of the master/slave buses, the efficiency of the master determinates the comportment of the whole network. Moreover, slaves can not communicate directly one with the other. All these characteristics are increasing the transmission errors. CANopen suppress all of these drawbacks. The timing comportment can be specified individually for each respective task of the participant devices. Like that, the whole communication system does not need to have more efficiency if only some of the devices need so. Moreover, an automatic task can be separated for each of the participant devices. So the performances of the network manager can be used in an optimised way and can increase at any time by adding new participant devices.

The variables mapping used during the PDO type exchanges permits to use in an optimal way the current bandwidth of the bus. CANopen determinates default values of all the parameters.

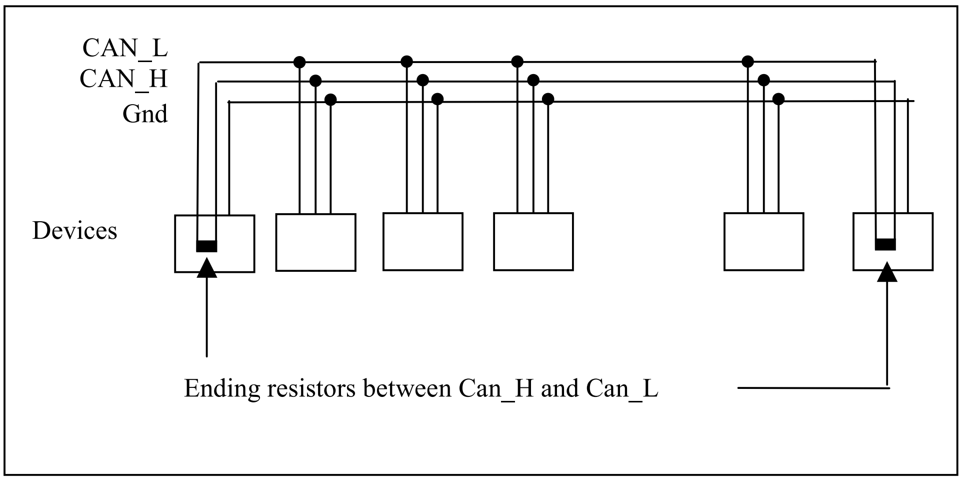
11-1-3- Network configuration

The CanOpen network is made of several devices, each of them can be master and slave. They are identified in the network by an arbitrary number, called Node-Id. This parameter must be unique: two different devices of the Can Open network can not have the same Node-Id. This Node-Id is very important, it is the real identity card of the peripheral on the Can Open network.



Example of CanOpen network configuration

The wiring is as follows:



Wiring of a Can Open network

Warning: Do not forget the ending resistors at each end of the Can Open network. For the SERAD products (SCAN and DIALOG), the resistor is validated if the jumper JP1 is present. If it is not, the resistor is un-validated.

For the other products, see the notice.

11-1-4- Type of send messages

There are two main kinds of messages sent on the Can Open network:

- The SDO are transmitting data
- The PDO are transmitting events

11-2- CANopen board for MCS32EX : SCAN

11-2-1- Presentation - SCAN board

The SCAN co-processor board is included in the rack of a MCS 32 EX. It owns three local tables of 254 data each, for these 3 data formats: 8 bits, 16 bits, 32 bits.

These tables can be read and written by the MCS 32 EX without going into the Can Open network, with the instructions *CanLocal*.

The different parameters of the SCAN board and the data tables are stored in a two-dimensions array, called **dictionary**.

Each data or parameter is defined by an address index, and a sub-index address.

The SCAN board can communicate with another device of the network by different ways. It can let data at other devices disposal by writing them in its local table: any other peripheral can then read and write this local table. It is the way used for example to communicate between an intelligent operator terminal Dialog 80 or 640.

The SCAN board can also read and write a local table of another device. This operation is then done with the instructions *CanRemote*.

11-2-2- Characteristics

➾ A SDO default server to set the parameters of the remote board by a supervisor.

➾ A SDO client to access to variables and peripheral parameters such as displays, PLC, PC boards.

➾ 8 PDO in emission to drive the outputs of the I/Os modules or signal an event to another MCS.

➾ 8 PDO in reception to receive the inputs of the I/Os modules or signal an event to another MCS.

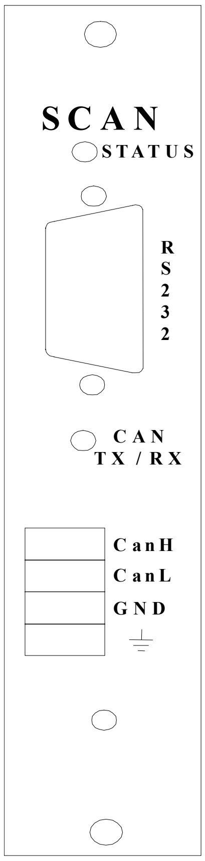
➾ An array of 254 variables « 8 bits non signed » with read and write access for SDO.

➾ An array of 254 variables « 16 bits non signed » with read and write access for SDO.

➾ An array of 254 variables « 32 bits non signed » with read and write access for SDO.

➯ Direct access functions to the bus CAN to send and receive specific messages such as the functions NMT et DBT.

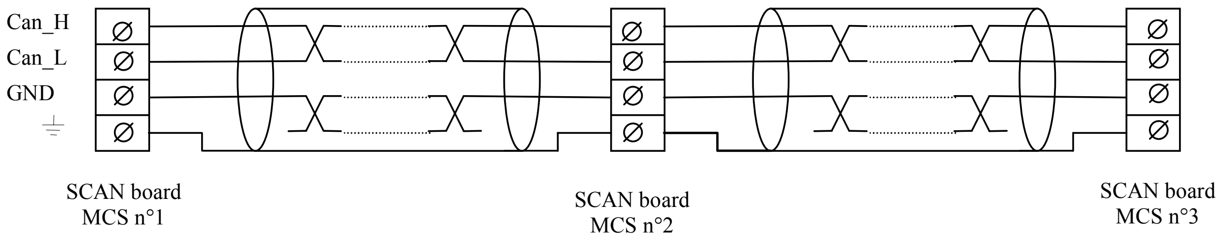
11-2-3- Connections



Use a cable with 2 twisted shielded pairs and a general shielding (type LiY.CY.CY or equivalent) :

- one pair for CAN_L and CAN_H
- one pair for the GND

Link the shieldings to the terminals



Example with 3 MCS 32 EX in a Can Open network :

Warning

At each end of the Can Open network do not forget a 120 Ω ending resistor between CAN_H and CAN_L (for a Dialog 80, Dialog 640 or SCAN board, the installation of the jumper JP1¨can validate this resistor.

For example, in the previous configuration, we have :

SCAN board n°1 : Jumper JP1 ON

SCAN board n°2 : Jumper JP1 OFF

SCAN board n°3 : Jumper JP1 ON

Maximal transmission speed regarding the length of the Can Open network

Maximal transmission speed	Network length
10K to 125 kBauds	500 m
250 kBauds	250 m
500 kBauds	100 m
800 kBauds	50 m
1 Mbauds	25 m

11-2-4- Test and diagnostic of the Can Open network

- LED indicators:

LED status:

Very slow flashing (0,5 second ON each 5 seconds): board in STOP mode

Slow flashing (0,5 second ON each second) : board in STARTED mode (but not sending nor receiving data)

Fast flashing : board in ACTIVE mode, sending or receiving data

LED CAN Rx/Tx:

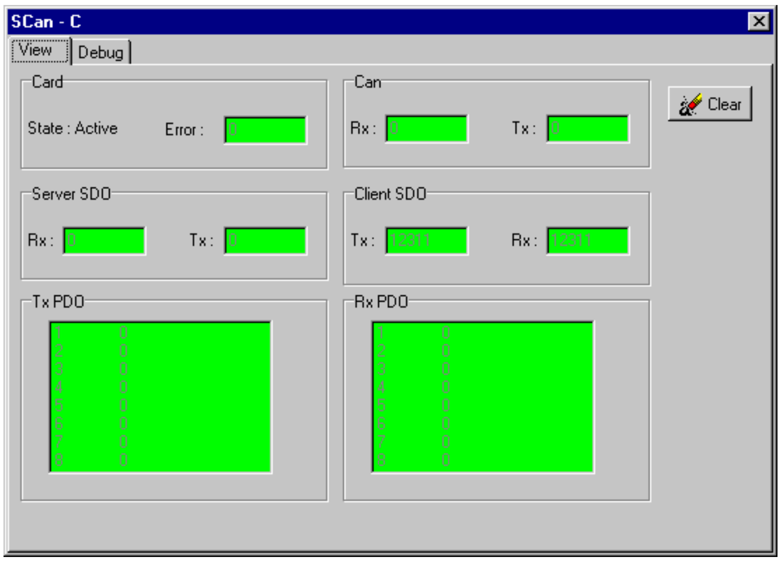
ON: error on the Can Open bus

Flashing: light is function of the traffic on the Can Open bus (its intensity can be very low or high)

- Debug mode SCAN board

From the MCB EX software, activate the debug mode and then double-click on the SCAN board.

VIEW page


SCan - C

View | Debug

Card
State : Active   Error : 0

Can
Rx : 0   Tx : 0

Server SDO
Rx : 0   Tx : 0

Client SDO
Tx : 12311   Rx : 12311

Tx PDO
```
1   0
2   0
3   0
4   0
5   0
6   0
7   0
8   0
```

Rx PDO
```
1   0
2   0
3   0
4   0
5   0
6   0
7   0
8   0
```

Card : visualisation of the communication errors number in the board and its state

3 different states:

STOPPED mode : the SCAN board waits for a StartCan instruction

STARTED mode : the SCAN board is ready to communicate

ACTIVE mode: the SCAN board is communicating

Can : visualisation of the transmission and reception number in free protocol

Server SDO : visualisation of the sent request and correct answers.

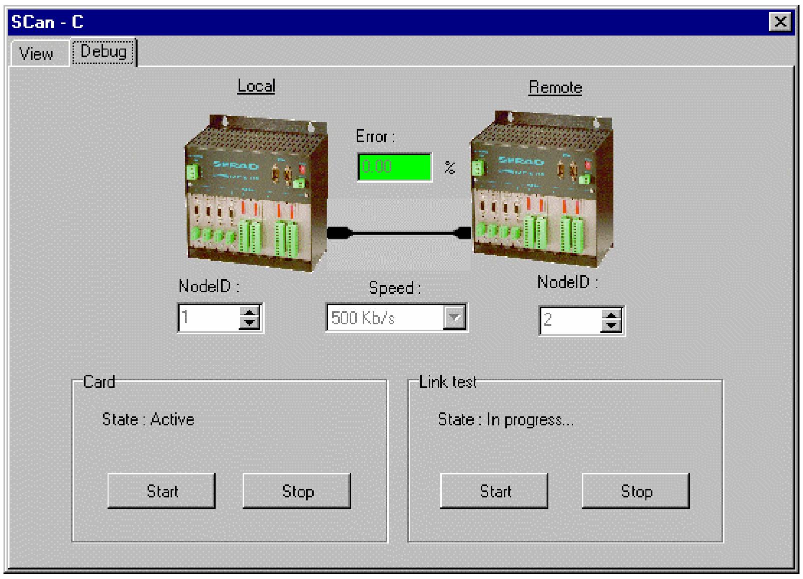
Client SDO : visualisation of the sent correct answers and request

Tx PDO : visualisation of the sent PDO number (sub-total per PDO number)

Rx PDO : visualisation of the received PDO number (sub-total per PDO number)

Clear : click here to clear all the counters of this page

DEBUG page :



This page can validate very easily the good comportment of two SCAN boards inside of a Can Open network.

The procedure is as follows:

On the local board, stop the tasks

Go to the debug menu of the SCAN board

Fill in its Node-Id, the transmission speed and the distant card Node-Id.

For the distant card, there are two different cases :

There is no task in the MCS 32 EX

You have to create one, to start the SCAN board, in automatic mode, called INIT for example:

```
Prog
  Delay 2000
  StartCan (CardName, Speed, Node-Id)
  Halt INIT
End Prog
```

There are already tasks in the MCS 32 EX

In the automatic task, add at the beginning :

```
Prog
  Delay 2000
  StartCan (CardName, Speed, Node-Id)
  Halt INIT
  …
```

```
End Prog
```

Warning: be sure that there is only one task in automatic mode, otherwise pass the others in manual mode.

In both cases, compile and transfer the program

Validate the test by clicking on “START” in the local card. The percentage of errors will tell you very quickly if the Can Open bus is right on a hardware point of view for these two MCS 32 EX.

NB : The errors percentage is calculated with the values printed in the “View” page. Therefore it may be useful to clear these values from time to time.

11-2-5- Dictionary

The dictionary contains the different parameters and variables of the board. They are directly accessible for the MCS with the functions CANSETUP. The variables tables are accessible with the functions CANLOCAL. To access to the other CANopen peripheral parameters, you have to use the functions CANREMOTE.

Index	Sub-idx	Nom	Type	Attr.	Défaut	Description
1000	0	Device type	32 bits non signé	ro	403	type d'appareil
1001	0	Error register	32 bits non signé	ro	0	registre d'erreur interne
1002	0	Manufacturer Status Register	32 bits non signé	ro	0	registre d'etat spécifique au constructeur
1003	0	predefined error field	8 bits non signé	ro	1	nombre d'erreurs apparues
	1	actual error	32 bits non signé	ro	0	dernière erreur apparue
1004	0	number of PDO's supported	32 bits non signé	ro	00080008h	Nombre de PDO supporté
	1	Number of synchronous PDO	32 bits non signé	ro	0	Nombre de PDO synchrone supporté
	2	Number of asynchronous PDO	32 bits non signé	ro	00080008h	Nombre de PDO asynchrone supporté
100B	0	Node ID	32 bits non signé	ro	aucune	
100F	0	Number of SDO's supported	32 bits non signé	ro	00010001h	Nombre de SDO supporté
1200	0	Number of elements	8 bits non signé	ro	2	paramètre du 1er SDO serveur
	1	SDO receive COB-Id	32 bits non signé	ro	600h+node-ID	COB-ID de récéption du 1er SDO serveur
	2	SDO transmit COB-ID	32 bits non signé	ro	580h+node-ID	COB-ID d'envoi du 1er SDO serveur
	3	node ID of the SDO client	8 bits non signé	rw	none	Node ID du SDO client
1280	0	Number of elements	8 bits non signé	ro	2	paramètre du 1er SDO client
	1	SDO transmit COB-ID	32 bits non signé	ro	aucune	COB-ID de récéption du 1er SDO client
	2	SDO receive COB-Id	32 bits non signé	ro	aucune	COB-ID d'envoi du 1er SDO client
	3	node ID of the SDO server	8 bits non signé	rw	none	Node ID du SDO serveur
1400	0	Number of elements	8 bits non signé	rw	2	paramètre de réception du 1er PDO
	1	COB-ID	32 bits non signé	rw	200h + Node Id	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de la réception
1401	0	Number of elements	8 bits non signé	rw	2	paramètre de réception du 2ème PDO
	1	COB-ID	32 bits non signé	rw	300h + Node Id	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de la réception
1402	0	Number of elements	8 bits non signé	rw	2	paramètre de réception du 3ème PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de la réception
1403	0	Number of elements	8 bits non signé	rw	2	paramètre de réception du 4ème PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de la réception
1404	0	Number of elements	8 bits non signé	rw	2	paramètre de réception du 5ème PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de la réception
1405	0	Number of elements	8 bits non signé	rw	2	paramètre de réception du 6ème PDO

	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de la réception
1406	0	Number of elements	8 bits non signé	rw	2	paramètre de réception du 7ème PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de la réception
1407	0	Number of elements	8 bits non signé	rw	2	paramètre de réception du 8ème PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de la réception
1800	0	Number of elements	8 bits non signé	rw	2	paramètre d'émission du 1er PDO
	1	COB-ID	32 bits non signé	rw	180h + Node Id	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de l'émission
1801	0	Number of elements	8 bits non signé	rw	2	paramètre d'émission du 2ème PDO
	1	COB-ID	32 bits non signé	rw	280h + Node Id	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de l'émission
1802	0	Number of elements	8 bits non signé	rw	2	paramètre d'émission du 3ème PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de l'émission
1803	0	Number of elements	8 bits non signé	rw	2	paramètre d'émission du 4ème PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de l'émission
1804	0	Number of elements	8 bits non signé	rw	2	paramètre d'émission du 5ème PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de l'émission
1805	0	Number of elements	8 bits non signé	rw	2	paramètre d'émission du 6ème PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de l'émission
1806	0	Number of elements	8 bits non signé	rw	2	paramètre d'émission du 7ème PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de l'émission
1807	0	Number of elements	8 bits non signé	rw	2	paramètre d'émission du 8ème PDO
	1	COB-ID	32 bits non signé	rw	aucune	COB-ID utilisé par le PDO
	2	Transmission type	8 bits non signé	rw	254	Type de l'émission
7180	from 1 to FEh	Read 32 bits variables	32 bits signé	ro	aucune	
7200	from 1 to FEh	Read 8 bits variables	8 bits non signé	ro	aucune	

7280	from 1 to FEh	Read 16 bits variables	16 bits non signés	ro	aucune	
8180	from 1 to FEh	Write 32 bits variable	32 bits signé	wo	aucune	
8200	from 1 to FEh	Write 8 bits variable	8 bits non signé	wo	aucune	
8280	from 1 to FEh	Write 16 bits variable	16 bits non signés	wo	aucune	

11-3- Instructions list

11-3-1- List of the CANopen instructions

A) Read and write the dictionary

CANSETUP#	Read or write a parameter (byte)
CANSETUP%	Read or write a parameter (word)
CANSETUP&	Read or write a parameter (long integer)

B) Modification of local variables

CANLOCAL#	Read or write a local variable (byte)
CANLOCAL%	Read or write a local variable (word)
CANLOCAL&	Read or write a local variable (long integer)

C) Modification of remote variables

CANREMOTE#	Read or write a remote variable (byte)
CANREMOTE%	Read or write a remote variable (word)
CANREMOTE&	Read or write a remote variable (long integer)

D) Instructions in mode PDO

CAN	Read or write data
CANEVENT	Test of a message arrival
PDOEVENT	Test of a PDO arrival
PDO	Read or write data by a PDO
SETUPCAN	configuration of a message

E) Control instructions

CANERROR	Faults detection
CANERRORCOUNTER	Controls and erases the communication errors
STOPCAN	Starts the CANopen module
STARTCAN	Stops the CANopen module

F) Instructions in mode PDO

SDOEVENT	Allow to know if a wrtting has been done
SDOINDEX	Allow to know the index of the dictionnary's object
SDOSUBINDEX	Allow to know the sub-index of the dictionnary's object

11-3-2- CAN – Read and write a message

Syntax 1 :	**CAN**(<Board>, <Data>)
Syntax 2 :	<Variable> = **CAN**(<Board>)
Accepted types :	<Data>, <Variable> : Characters string
Description :	This function reads or send a message.
Remark :	<Board> must be a CANopen board. You have to tell the parameters of the reception COBID to receive the message.

11-3-3- CANERROR – Faults detection

Syntax :	<Variable> = **CANERROR**(<Board>)
Accepted types :	<Variable> : Boolean
Description :	This function tells if a default occurred.
Remark :	<Board> must be CANopen board.

11-3-4- CANERRORCOUNTER – Controls and erases the communication errors

Syntax 1 :	<Variable> = **CANERRORCOUNTER** (<Board>)
Syntax 2 :	**CANERRORCOUNTER**(<Board>) = 0
Limits :	<Variable> : from 0000h to FFFFh
Accepted types :	<Variable> : integer
Description :	The syntax 1 tells the number of errors which had occurred since the counter has been reset. The second resets the errors counter.
Remark :	<Board> must be CANopen board.

11-3-5- CANEVENT – Test a message arrival

Syntax :	<Variable> = **CANEVENT** (<Board>)
Accepted types :	<Variable> : Boolean
Description :	This function permits to know if a message has been receipted.
Remark :	<Board> must be a CANopen board. You have to tell the parameters of the reception COBID to receive the message.

11-3-6- CANLOCAL – Read or write a local variable

Syntax 1 :	**CANLOCAL#** (<Board>, <Index>, <Expression>)
Syntax 2 :	<Variable> = **CANLOCAL#** (<Board>, <Index>)
Syntax 3 :	**CANLOCAL%** (<Board>, <Index>, <Expression>)
Syntax 4 :	<Variable> = **CANLOCAL%** (<Board>, <Index>)
Syntax 5 :	**CANLOCAL&** (<Board>, <Index>, <Expression>)
Syntax 6 :	<Variable> = **CANLOCAL&** (<Board>, <Index>)
Limits :	<Index> : from 0000h to FFFFh
	Syntax 1 and 2 : <Variable>, <Expression> : from 00h to FFh

	Syntax 3 and 4 : <Variable>, <Expression> : from 0000h to FFFFh
	Syntax 5 and 6 : <Variable>, <Expression> : +/- 7FFFFFFFh
Accepted types :	Syntax 1 and 2 : <Expression>, <Variable> : Byte
	Syntax 3 and 4 : <Expression>, <Variable> : Integer
	Syntax 5 and 6 : <Expression>, <Variable> : Long integer
Description :	This function can read or write a local variable of the CANopen board dictionary of the MCS. The syntax 1 and 2 are giving an access to a table of 8 bits non-signed variables. The syntax 3 and 4 are giving an access to a table of 16 bits non-signed variables. The syntax 5 and 6 are giving an access to a table of 32 bits signed variables.
Remark :	<Board> must be a CANopen board. <Index> must refers to a local variable of the dictionary.

11-3-7- CANREMOTE – Read or write a remote variable

Syntax 1 :	**CANREMOTE#** (<Board>, <Index>, <Sub-Index>, <Expression>)
Syntax 2 :	<Variable> = **CANREMOTE#** (<Board>, <Index>, <Sub-Index>)
Syntax 3 :	**CANREMOTE%** (<Board>, <Index>, <Sub-Index>, <Expression>)
Syntax 4 :	<Variable> = **CANREMOTE%** (<Board>, <Index>, <Sub-Index>)
Syntax 5 :	**CANREMOTE&** (<Board>, <Index>, <Sub-Index>, <Expression>)
Syntax 6 :	<Variable> = **CANREMOTE&** (<Board>, <Index>, <Sub-Index>)
Limits :	<Index> : from 0000h to FFFFh
	<Sub-index> : from 00h to FFh
	Syntax 1 and 2 : <Variable>, <Expression> : from 00h to FFh
	Syntax 3 and 4 : <Variable>, <Expression> : from 0000h to FFFFh
	Syntax 5 and 6 : <Variable>, <Expression> : +/- 7FFFFFFFh
Accepted types :	Syntax 1 and 2 : <Expression>, <Variable> : Byte
	Syntax 3 and 4 : <Expression>, <Variable> : Integer
	Syntax 5 and 6 : <Expression>, <Variable> : Long integer
Description :	This function reads or writes a remote variable in the dictionary of the MCS CANopen board.
Remarks :	<Board> must be a CANopen board. <Index> and <Sub-Index> must refer to an element of the remote dictionary. You have to tell the SDO client and server parameters of the board before sending any remote variable reading or writing.

11-3-8- CANSETUP – Read or write a parameter

Syntax 1 :	**CANSETUP#** (<Board>, <Index>, <Sub-Index>, <Expression>)
Syntax 2 :	<Variable> = **CANSETUP#** (<Board>, <Index>, <Sub-Index>)
Syntax 3 :	**CANSETUP%** (<Board>, <Index>, <Sub-Index>, <Expression>)
Syntax 4 :	<Variable> = **CANSETUP%** (<Board>, <Index>, <Sub-Index>)
Syntax 5 :	**CANSETUP&** (<Board>, <Index>, <Sub-Index>, <Expression>)
Syntax 6 :	<Variable> = **CANSETUP&** (<Board>, <Index>, <Sub-Index>)
Limits :	<Index> : from 0000h to FFFFh

	<Sub-index> : from 00h to FFh
	Syntax 1 and 2 : <Variable>, <Expression> : from 00h to FFh
	Syntax 3 and 4 : <Variable>, <Expression> : from 0000h to FFFFh
	Syntax 5 and 6 : <Variable>, <Expression> : +/- 7FFFFFFFh
Accepted types :	Syntax 1 and 2 : <Expression>, <Variable> : byte
	Syntax 3 and 4 : <Expression>, <Variable> : Integer
	Syntax 5 and 6 : <Expression>, <Variable> : Long integer
Description :	This function reads or writes data in the MCS CANopen board dictionary.
Remark :	<Board> must be a CANopen board. <Index> and <Sub-Index> must refer to elements of the dictionary.

11-3-9- PDOEVENT – Test a PDO arrival

Syntax :	<Variable> = **PDOEVENT** (<Board>, <NumPDO>)
Limits :	<NumPDO> : from 01h to 08h
Accepted types :	<Variable>, <NumPDO> : Byte
Description :	This function tells if a request of a PDO is effective.
Remark :	<Board> must be a CANopen board. You have to tell the transmission parameters of the PDO to receive a PDO.

11-3-10- PDO – Read or write data from a PDO

Syntax 1 :	**PDO** (<Board>, <NumPDO>, <Data>)
Syntax 2 :	<Variable> = **PDO** (<Board>, <NumPDO>)
Limits :	<NumPDO> : from 01h to 08h
	<Data>, <Variable> : characters string
Accepted types :	<NumPDO> : Byte
	<Data>, <Variable> : characters string
Description :	This function reads or writes a PDO.
Remarks :	<Board> must be a CANopen board. You have to tell transmission parameters of the PDO to receive a PDO.

11-3-11- SDOEVENT – Event SDO

Syntax :	<Variable bit> = SDOEvent(<Can Card>)
Description :	This function allow to know if a writting by SPO has been made in the CAN card. The reading of the bit, reset it.

11-3-12- SDOINDEX – Index SDO

Syntax :	<Variable integer> = SDOIndex(<Can Card>)
Description :	This function allow to know the index of the dictionnary's object who has been written.

11-3-13- SDOSUBINDEX – Sub-index SDO

Syntax :	<Variable octet> = SDOSubIndex(<Can Card>)
Description :	This function allow to know the sub-index of the dictionnary's object who has been wrotten.

11-3-14- SETUPCAN – Configuration of a message

Syntax :	**SETUPCAN** (<Board>, <TX COBID>, <RX COBID>)
Accepted types :	<TX COBID>, <RX COBID> : Long integer
Description :	This function configures the reception and transmission COBID before sending a message.
Remark :	<Board> must be a CANopen board.

11-3-15- STARTCAN – Start a CANopen board

Syntax :	**STARTCAN** (<Board>, <Node ID>, <Freq>)
Limits :	<Node ID> : from 01h to FFh
	<Freq> : from 1 to 8
Accepted types :	<Node ID>, <Freq> : Byte
Description :	This function links the CANopen board to the network.
Remark :	<Board> must be CANopen board.

11-3-16- STOPCAN – Stop a CANopen board

Syntax :	**STOPCAN** (<Board>)
Description :	This function puts the corresponding board out of the CANopen network.
Remark :	<Board> must be CANopen board.

11-4- Examples

11-4-1- CANopen kink between two MCS

The communication configuration between two MCS consists of giving a NodeID number to each MCS. Then a communication with SDO is possible when those are configured. You can also exchange events with PDO.

The default COBID of the servers are 600h+NodeID in reception and 580h+NodeID in emission. The default COBID of the first PDO are 200h+NodeID for the reception and 180h+NodeID for the emission. You configures the clients in accordance with that.

➯ Initialisation of the MCS 1

```
'Start the board at 500KBits/s on the node 1
StartCan(Can1,1,5)
'COBID ClientSDO Rx Mcs1= COBID ServerSDO Tx Mcs2
CanSetup&(Can1,1280h,1,582h)
'COBID ClientSDO Tx Mcs1= COBID ServerSDO Rx Mcs2
```

```
CanSetup&(Can1,1280h,2,602h)
'COBID TxPDO1 = COBID RxPDO2
CanSetup&(Can1,1800h,1,202h)
```

➷ Initialisation of the MCS 2

```
'Start the board at 500KBits/s on the node 2
StartCan(Can2,2,5)
'COBID ClientSDO Rx Mcs2= COBID ServerSDO Tx Mcs1
CanSetup&(Can2,1280h,1,581h)
'COBID ClientSDO Tx Mcs2= COBID ServerSDO Rx Mcs1
CanSetup&(Can2,1280h,2,601h)
'COBID TxPDO2 = COBID RxPDO1
CanSetup&(Can2,1800h,1,201h)
```

When this initialisation is over the MCS can exchange data and events. In this example, the MCS2 sends positioning commands to the X axis of the MCS 1. The MCS 1 receives the order by a PDO and tells the end of the command by sending a PDO. The position to reach is read in the variable 5 of the table "read 32 bits variables" of the MCS 2. The MCS 1 also makes the X axis position available in the variable 1 of its table "write 32 bits variable".

```
Wait PDOEvent(Can1,1)              'Waits for the PDO which signals the message
O$=PDO(Can1,1)                     'Reads the PDO
Ordre#=Asc(Left$(O$,1))            'Decoding the command
Pos&=CanRemote&(Can1,7180h,5)      'Reads the position
If Ordre#=1 Then Stta(X=Pos&)             'Execution in absolute
If Ordre#=2 Then Sttr(X=Pos&)             'Execution in relative
...
Repeat
  P&=RealToLong(Pos_S(X))          'Read the position
  CanLocal&(Can1,1,P&)             'Updates the position
Until Move_S(X)=0
O$=Chr(0)                          'Answer
PDO(Can1,1,O$)                     'Acquits the command
```

The MCS 2 sends its commands, reads the X axis position in the variable 1 of the table "read 32 bits variables" and send positions in the variable 5 of the table "write 32 bits variables".

```
CanLocal&(Can2,5,10.25)       'Writes the position
O$=Chr(1)                     'Sends a command for absolute motion
PDO(Can2,1,O$)                'Sends the PDO
Repeat
  P&=CanRemote&(Can2,7180h,1)      'Reads the position
  ...
Until PDOEvent(Can2,1)        'Until the end of motion
```

11-4-2- CANopen linking between a MCS and an I/Os module

The communication configuration between a MCS and an I/Os module consists of giving a NodeID number to each of them. In general cases the NodeID of an I/Os device is configured with switches. Then a communication with SDO and PDO is possible.

The default COBID of the servers are 600h+NodeID in reception and 580h+NodeID in emission. The default COBID of the first PDO are 200h+NodeID for the reception and 180h+NodeID for the emission. You configures the clients in accordance with that.

➷ Initialisation of the MCS

```
'Start the board at 500KBits/s on the node 1
StartCan(Can1,1,5)
'COBID ClientSDO Rx Mcs= COBID ServerSDO Tx I/O
CanSetup&(Can1,1280h,1,582h)
'COBID ClientSDO Tx Mcs= COBID ServerSDO Rx I/O
CanSetup&(Can1,1280h,2,602h)
'COBID TxPDO MCS = COBID RxPDO I/O
CanSetup&(Can1,1800h,1,202h)
'COBID RxPDO MCS = COBID TxPDO I/O
CanSetup&(Can1,1400h,1,182h)
```

The I/Os devices need the sending of the message « NMT Start » so they can be operational. To send this message you use the general CAN functions:

```
SetupCan(Can1, 0, 0)  ' Use the le COBID 0 to access to the NMT server
Nmt$=Chr$(1)+Chr$(2)  ' The module NodeID is 2.
Can(Can1,Nmt$)
```

Read and write I/Os by SDO can be like that:

```
A#=CanRemote#(Can1,6000h,1)         'Read inputs 1 to 8
A#=CanRemote#(Can1,6000h,2)         'Read inputs 9 to 16
CanRemote#(Can1,6200h,1,01000100b) 'Updates outputs 3 and 7
```

It is possible to receive inputs states and to modify outputs states with the PDO. The contents of the PDO is depending on the mapping defined by the construction.

```
Wait PDOEvent(Can1,1)       'Wiats fr a change on the inputs
E$=PDO(Can1,1)              'Reads the PDO
E1#=ASC(MID$(E$,1,1))       'Reads the first inputs bloc
If E1#.3 Then ...           'Uses the 3rd input
S$=Chr$(00010011b)          'Writes outputs 1, 2 and 5
PDO(Can1,1,S$)              'Sends the PDO
```

11-4-3- Communication between an intelligent operator terminal and a SCAN board

- **General principle**

The intelligent operator terminals Dialog 80 and Dialog 640 are working in the same way for the Can Open communication.

We can say they are master, and that all the peripherals (SCAN boards on the MCS 32 EX) are slave. Only SDO can be transmitted.

Data are read and written in the local tables of the SCAN boards.

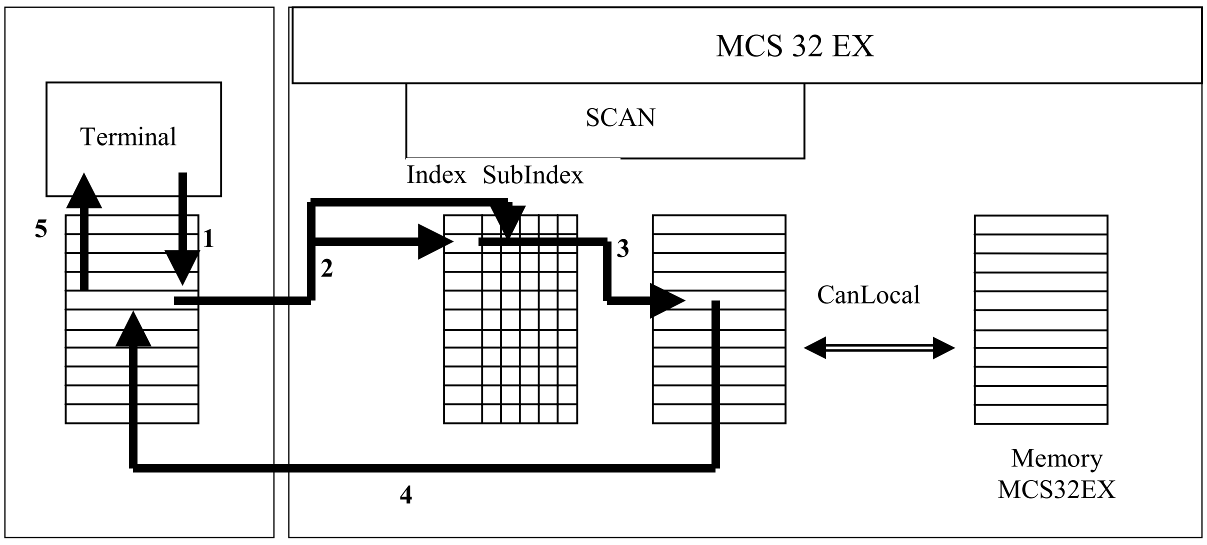
Each variable is affected to a peripheral and a sub index. In this case, the word pepipheral includes two main information, which are:

The Node-ID of the target device

The reception and transmission address of the local target table for this device.

Then, an unique device defined by a single Node-ID can be divided in several peripherals. For example, for a SCAN board, we can define one peripheral for the 32 bits data table, another one for the 16 bits one, and another one for the 8 bits one.

The reading of a data is made as follows:



Step 1: the terminal must print a data, it goes in its own memory to read it.

This data must be updated before printing, so, through the Can Open network, it sends its Index and Sub-Index to the corresponding Node-ID. They are directly written in the dictionary in the specified addresses.

Step 3: parameters which are in very particular places of the dictionary take the concerned data of the local table of the SCAN board.

Step 4: this data is immediately sent on the Can Open bus and goes to the variables table of the terminal.

Step 5: this data can now be printed by the terminal

•

You have to begin to set parameters of the Can Open link.

Different parameters are:

Node-ID : Number of the Dialog in the configuration of the Can Open network

Speed: transmission speed of the bus

Timeout : Maximum time for a transmission before a default

Retry: number of retries for a transmission before a default

Wait for NMT start: wait for a command to launch the network (not used in most cases)

Send NMT start: send a command to launch the network (not used in most cases)

Default SDO: allows to work without modify the internal SDO of the Dialog (choose this option in standard cases)

The state table and the command table have the same functions as in a standard ModBus driver.

Declaration of the peripherals

A peripheral is the association of a target Node-ID and its local target table of variables among the ones it has.

In the case of a SCAN module, there are 3 local tables, one for the 32 bits data, one for the 16 bits data and one for the 8 bits data.

For example, if you want to work with 32 bits data and 8 bits data on the same SCAN board, you will have to declare 2 different peripherals, with the same Node-ID, but different Rx and Tx parameters.

The values to set are elements of the target board SCAN dictionary, and you can find them in this table (in decimal)

	RxIndex : read	TxIndex : write
32 bits variables	29056	33152
16 bits variables	29312	33408

8 bits variables	29184	33280

RxSub and TxSub are the sub-index from which we will start to read and write in the local target table. In the case of a SCAN board, these values can be let to 1. **They must never be set to zero**. Their maximum value is 254.

- **How to set variables parameters**

The parameters of variables which have to be read or written through the Can Open bus must be set as follows:

Name, type, size are the characteristics of the variable.

Device is the number of the device associated to this variable, as it has been defined in the Can Open setting. It is not necessary equal to the Node-ID.

Address is the shift in the local target table from RxSub and TxSub. Warning, if RxSub and TxSub can not be equal to zero, address can be set to zero. For example, for a transmission, if RxSub=1 and address=0, it means that the target variable is the RxSub+Adresse=0+1=1st variable of the local target table.

Initialize : Send the variable value to the concerned peripheral when power-on the system.

Read and Write are the variable read and write modes by the Can Open network.

- **How to set parameters for the SCAN board**

To let at disposal and receive data from the terminal, the SCAN board must be set as follows:

```
delay 2000
StartCan(Can1,5,1)
CanSetup&(Can1,1280h,1,602h)
CanSetup&(Can1,1280h,2,582h)
```

Delay :

Hardware initialisation time of the SCAN board when you power-on the MCS 32 EX

StartCan(Can1,5,1) :

board Can1

5 : bus speed. In this case, 500 kBauds. Here is the table with the different values:

Code	Bus speed (kBauds)
0	10
1	20
2	50
3	125
4	250
5	500
6	800
7	1.000

1 : Node-ID of the SCAN board

CanSetup&(Can1,1280h,1,602h)

CanSetup&(Can1,1280h,2,582h)

Can1 : Name of the SCAN board

602h and 582h : Communication with the Dialog which has a Node-ID is 2. In another case we have to set 600h+Node-ID and 580h+Node-ID.

- **Writing data to be at the terminal disposal from the MCS 32 EX to the SCAN board**

After this procedure, the SCAN board has the right parameters, you just have to read and write data in its local table.

This can be done with the CanLocal instruction in a task.

```
‘ Write the 32 bits variable DATA in the sub-index 1 of the Can1 board
CanLocal&(Can1,1,DATA)
‘ Read the 16 bits variable of the sub-index 23 of the Can1 board, transfered to the MCS32EX variable DATA2
DATA2=CanLocal%(Can1,23)
```

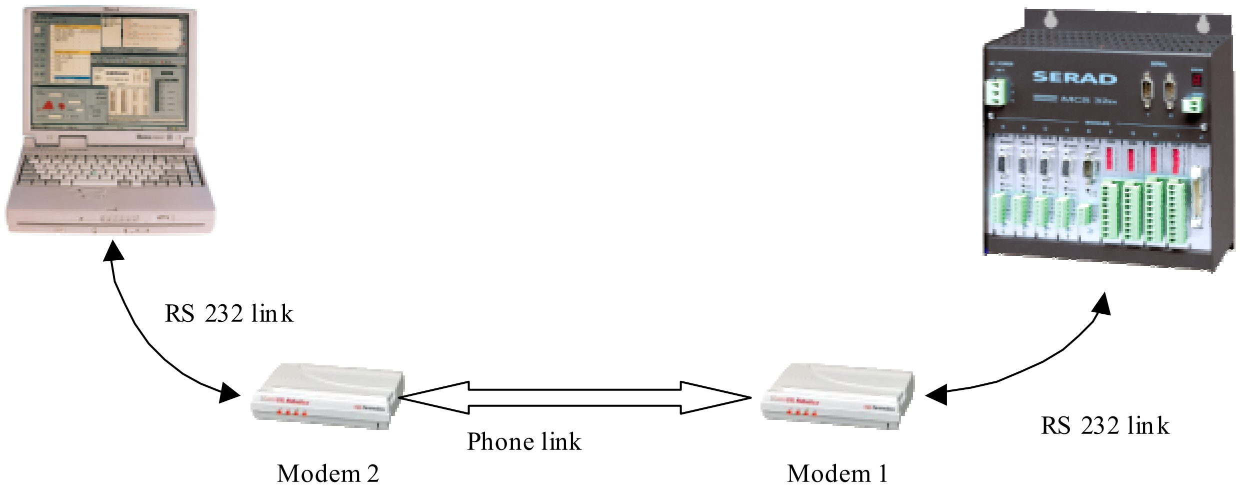
12- REMOTE CONTROL

12-1- Connections

The remote control allows by a simple phone link to remotely control an MCS 32 EX with MCB EX software. The remote control is composed of an integrated dialler and two modems linked by a phone link.

- Structure

The different parts are linked as shown :



- RS 232 link between the modem 1 and the MCS 32 EX

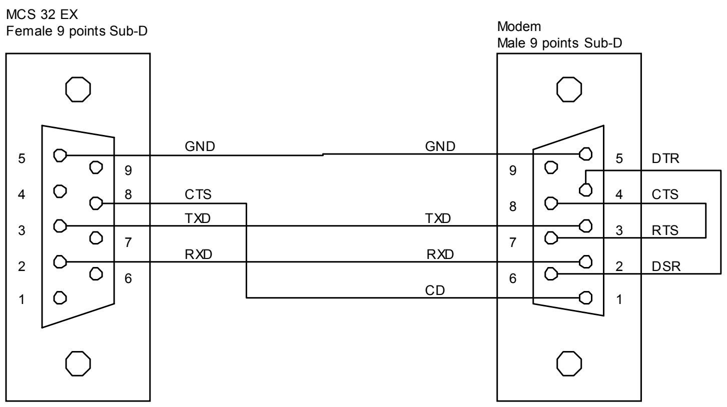
9 points SUBD pin assignment :

Pin	MCS	Modem
1		CD
2	RXD	RXD
3	TXD	TXD
4		DTR
5	GND	GND

6		DSR
7		RTS
8	CTS	CTS
9		

Use a shielded cable with shield connected at each end.

Linking :



- RS 232 link between the modem 2 and the PC

This link between the modem and the PC is made with the cable provided with the modem.

12-2- Link establishment

- **Setting up the modem 1 connected to the MCS**

The set-up of the modem connected to the MCS is made by connecting this modem to a PC. A terminal software is used to send commands to the modem.

This set-up have to following objectives :

- Initialising the modem
- Defining the number of ringing before the modem pick up to allow an automatic establishment of the link.
- Removing all hardware and software flow controls.
- Storing this configuration into the non-volatile memory of the modem.

- Selecting these parameters in the non-volatile memory as parameter to be used at power on.

Example :

Parameters for an « 3Com Us Robotics Sportster » modem type :

-Command : AT&F0

Meaning : Using default factory settings.

-Command : ATS0=3

Meaning : Automatic pick up after 3 ringing.

-Command : AT&H0

Meaning : Disable the flow control when sending

-Command : AT&I0

Meaning : Disable the flow control when receiving

-Command : AT&W0

Meaning : Store current parameters into the non-volatile memory bank #0

-Command : ATY0

Meaning : Selecting these parameters in the non-volatile memory as parameter to be used at power on.

When the modem take these commands into account it answers « OK » .

Parameters for an « Wertermo TD31 or TD32 » modem type :

-Command : AT&F

Meaning : Using default factory settings.

-Command : ATS0=3

Meaning : Automatic pick up after 3 ringing.

-Command : AT&C1

Meaning : Activate DCD when connected

-Command : AT&K0

Meaning : Disable the flow control

-Command : AT&W0

Meaning : Store current parameters into the non-volatile memory bank #0

-Command : AT&Y0

Meaning : Selecting these parameters in the non-volatile memory as parameter to be used at power on.

When the modem take these commands into account it answers « OK » .

- **Setting up the modem 2 connected to the PC**

The setting up of the modem connected to the PC is done by modifying the information in the « Modem » part of the MCB.INI file that is in the Windows directory (C :\Windows or C :\Winnt for example).

This set-up have to following objectives :

- Initialising the modem
- Remove handling of the DSR and DTR signals to avoid automatic hang-up when the communication port is closed.
- Defining the way the calls are made and how to hang-up the line.
- Defining the messages sent by the modem.

Example :

Parameters for an « 3Com Us Robotics Sportster » modem type :

- Parameter : Init1

Value : ATZ

Meaning : Using default factory settings.

- Parameter : Init1TimeOut

Value : 5

Meaning : Maximal waiting delay in 1/10 before the modem answer.

- Parameter : Init2

Value : AT&D0&S0

Meaning : Remove the DTR and DSR handling

- Parameter : Init2TimeOut

Value : 5

Meaning : Maximal waiting delay in 1/10 before the modem answer.

- Parameter : Dial

Value : ATDT for vocal dial. ATDP for a pulse dial

Meaning : Selecting the way to call.

- Parameter : DialTimeOut

Value : 600

Meaning : Maximal waiting delay in 1/10 before the modem connection.

- Parameter : Ok

Value : OK

Meaning : Modem answer if the command have been handled correctly.

- Parameter : Connect

Value : CONNECT

Meaning : Modem answer when connecting.

- Parameter : Busy

Value : BUSY

Meaning : Modem answer if the line is busy.

- Parameter : Hangup

Value : ATH

Meaning : Selecting the way to hang-up.

- Parameter : HangupOk

Value : NO CARRIER

Meaning : Modem answer when hanging-up

- Parameter : CommandTimeOut

Value : 20

Meaning : Maximal waiting delay in 1/10 before the modem going to the command mode.

- Parameter : HangupTimeOut

Value : 20

Meaning : Maximal waiting delay in 1/10 before the hanging up.

All missing parameter is automatically set to the default values indicated on the first using.

Parameters for an « Westermo TD31 or TD32 » modem type :

-Parameter : Init1

Value : ATZ

Meaning : Using default factory settings.

- Parameter : Init1TimeOut

Value : 20

Meaning : Maximal waiting delay in 1/10 before the modem answer.

- Parameter : Init2

Value : AT&F&K0

Meaning : Remove the DTR and DSR handling

- Parameter : Init2TimeOut

Value : 20

Meaning : Maximal waiting delay in 1/10 before the modem answer.

- Parameter : Dial

Value : ATDT for vocal dial. ATDP for a pulse dial

Meaning : Selecting the way to call.

- Parameter : DialTimeOut

Value : 600

Meaning : Maximal waiting delay in 1/10 before the modem connection.

- Parameter : Ok

Value : OK

Meaning : Modem answer if the command have been handled correctly.

- Parameter : Connect

Value : CONNECT

Meaning : Modem answer when connecting.

- Parameter : Busy

Value : BUSY

Meaning : Modem answer if the line is busy.

- Parameter : Hangup

Value : ATH

Meaning : Selecting the way to hang-up.

- Parameter : HangupOk

Value : NO CARRIER

Meaning : Modem answer when hanging-up

- Parameter : CommandTimeOut

Value : 20

Meaning : Maximal waiting delay in 1/10 before the modem going to the command mode.

- Parameter : HangupTimeOut

Value : 20

Meaning : Maximal waiting delay in 1/10 before the hanging up.

The dialler expect that the modem is setup to send an echo for all sent command and to receive a text message as answer. If not the communication is unable. It’s possible to be sure to start with a good set-up for the modem by using the factory settings as default parameters.

A terminal software is used to send commands to the modem.

Parameters for an « 3Com Us Robotics Sportster » modem type :

- Command : AT&F

Meaning : Using default factory settings.

- Command : AT&W0

Meaning : Store current parameters into the non-volatile memory bank #0

- Command : ATY0

Meaning : Selecting these parameters in the non-volatile memory as parameter to be used at power on.

Parameters for an « Wertermo TD31 or TD32 » modem type :

- Command : AT&F

Meaning : Using default factory settings.

- Command : AT&W0

Meaning : Store current parameters into the non-volatile memory bank #0

- Command : AT&Y0

Meaning : Selecting these parameters in the non-volatile memory as parameter to be used at power on.

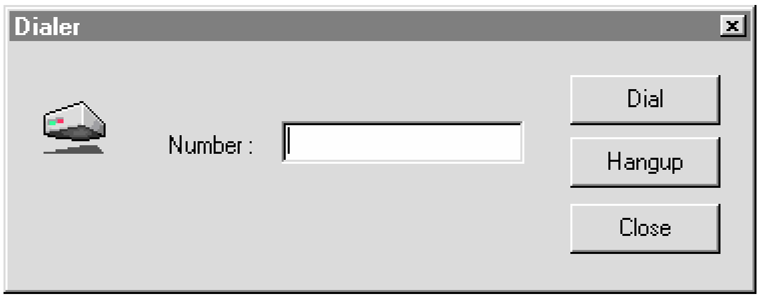
ATTENTION :

- For Westermo modem , it’s also recommended to let the Dips configuration as default (all OFF).

- The Westermo modems are only supported by the version 1.33 are greater of the MCB EX. Software.

- **Call :**

By using the phone dialler integrated in the MCB ex software, we can establish and interrupt the phone link. The phone dialler is accessible form the Communication menu / Remote control.



After entering the phone number, click on «Dial» button to establish the link. The «Hang up » button allows to interrupt the link.

These actions are possible only if the MCB software is not using the same link in debug mode for example. During the connection and disconnection the communication port is not available for the rest of the MCB application.

When the link is established, we can use all the MCB functions including :

- Send and receive the configuration
- Send and receive the variables
- Send the tasks
- Start the tasks
- Stop the tasks
- Access to debug tools : Hyper-terminal, Scope, Trace, Manual mode.

12-3- List of the validated modems

- **3 Com / US Robotics :**

  - Sportster Voice 33600 Fax Modem
  - Sportster 56 K Fax Modem

- **Westermo :**

  - TD 31
  - TD 32

13- APPENDIX

13-1- Compiler error messages

Find <Type1> <Text1> : <Type2> <Text2> Expected

An identifier <Text1> of type <Type1> has been found at the time of the compilation instead of an identifier <Text2> of type <Type2>.

L or H expected

To change an integer in Byte we must use ".L" or ".H".

<Text> unexpected : Prog name expected

The program name must be an identifier no defined previously.

Prog bloc already defined

More than one PROG ... END PROG block is defined in the task.

<Text> unexpected : PROG or SUB expected

A block begins by PROG or SUB. An instruction has been added outwards a block.

No defined PROG

The current block was not finished before end of source file.

Undefined Label

An unknown label has been used in a Goto instruction.

Undefined Sub

An unknown subprogram identifier has been used in a Call instruction.

Undefined Event

An event generated by Signal is waited by any task or a task waits for an event which will be never generated.

Undefined Prog

An unknown task identifier has been used in the Run, Halt, Suspend or Continue instruction.

SRV15 Card Expected

To use axis card home input, in the parameter InpHome_p, home input name must be the same of axis card input.

Instruction expected

An instruction is expected.

Buzzer : Bit constant expected

The Buzzer instruction must be followed by a type bit constant.

Goto or Call instruction expected

A Call or Goto instruction is expected in a Case

Invalid exit instruction

A Exit Sub instruction must be used only in a subprogram.

<Text> Expected

The FOR loop counter variable must be also used in the Next instruction.

➪ If : Instruction expected

An instruction is expected after an If.

➪ Else : Instruction expected

An instruction is expected after an Else.

➪ SERIAL1: or SERIAL2: Expected

In Open instruction, the name of the communication port is either SERIAL1: or SERIAL2:.

➪ POS, VEL, ACC or DEC expected

The TRAJ instruction accepts only POS, VEL, ACC or DEC as parameter.

➪ Undefined variable

The variable contents is used before being defined by an affectation.

➪ String expression expected

A type string expression is expected.

➪ Bit expression expected

A type bit expression is expected.

➪ Comment bloc : Unexpected end of file.

A comments bloc begins by '{{'

➪ Comment bloc : Unexpected char

An other character as '{' has been found.

➪ String constant : Unexpected end of file.

A string constant must finish with quotation marks.

➪ Comment bloc : Unexpected end of line

A comment bloc finishes by '}}'

➪ Bad hex number

An hexadecimal number uses the characters 0 to 9 and A to F

➪ Bad binary number

A binary number uses the characters 0 to 1

➪ Not an hex value

An hexadecimal number uses the characters 0 to 9 and A to F

➪ Not a binary value

A binary number uses the characters 0 to 1

➪ Not a decimal value

A decimal number uses the characters 0 to 9

➪ Real constant : Unexpected end of line

A real constant must finish by a number after the decimal point.

➪ <Text> unexpected : Char from 0 to 9 expected

A real or decimal number uses the characters 0 to 9

➪ System constant : Unexpected end of file

A no complete system constant has been found.

<Text> unexpected : System constant expected

A system constant is expected.

Number : Unexpected end of file

A number finishes by a number

<Text> unexpected : Number from 0 to 9 expected

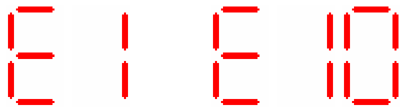
A number finishes by a number

'<Caracter>' unexpected

A no-waited character has been found.

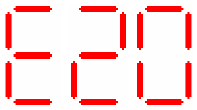
13-2- Execution errors messages

The status display can be treated in a task to indicate its evolution, the state of the MCS and the error message. In a default state, the status displays a ‘G’ like go. If the MCS is connected with MCB software, a ‘S’ like system is displayed. The error messages have the priority.



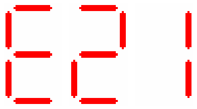
ERROR N°1 to ERROR N°10 :

The errors from 1 to 10 indicate that a card is not well declared or a declared card in the configuration is away or has been replaced by an other type. The number following the E indicates the slot. For example E6 indicates that the card in the slot 6 is not well declared. The system doesn't use the parameters and doesn't start user tasks.



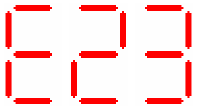
ERROR N°20 :

The error 20 indicates that the data in the saved memory have been corrupted and it is necessary to reload the configuration and the saved variables. The system doesn't use the parameters and doesn't start user tasks.



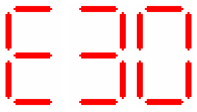
ERROR N°21 :

The error 21 appears under MCS 32 power-on if a parameter of the configuration is wrong. The parameters must be marked and send before starting once again the MCS. The system doesn't start the user tasks.



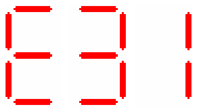
ERROR N°23 :

The error 23 indicates that there are no user tasks in the MCS



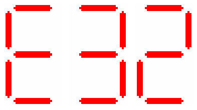
ERROR N°30 :

When a user program makes a divide by zero error n°30 is displayed.



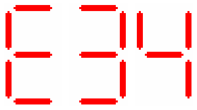
ERROR N°31 :

This error is due to an infinite recursive call of a subprogram and indicates a stack overflow.



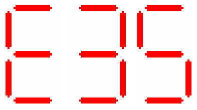
ERROR N°32 :

This error is generated when a floating point overflow is made by a number too high.



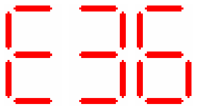
ERROR N°34 :

When an invalid floating point operation has been detected this error is generated. It is produced with the REALTOLONG function if the real number is to big to be stored in long integer.



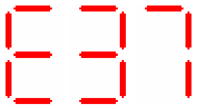
ERROR N°35 :

This error is generated by an arithmetical overflow in a calculus. This is produced when the result of an operation is too great to be stocked in the provided variable receiving it.



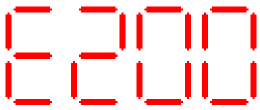
ERROR N°36 :

This error is generated because an index of save variable's table is out of limites.



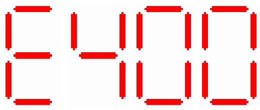
ERROR N°37 :

This error is generated because an index of not save variable's table is out of limites.



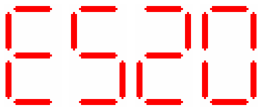
ERROR N°200 + Axis Number :

This error is generated because it is impossible to load an axis card.



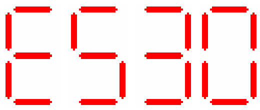
ERROR N°400 + Axis Number :

This error is generated because this axis is not responding.



ERROR N°520 :

This error is generated because it's unable to access to internal global bus. The internal bus is bad defect or freeze by an axis card.



ERROR N°530 :

This error is generated because it's unable to access to internal global bus. The internal bus is bad defect or freeze by an axis card.

These last five errors are generated only during user program execution. When an error is detected, all the task are stopped, the error message is displayed on the MCS status, the watchdog is opened and all the servo axis are in a open loop state.

Index