



ASA-RT srl -- Strada del Lionetto, 16/a – 10146 Torino – ITALY  
Tel: +39 011 796 333r.a.. – Fax: +39 011 712 339 -  
E mail: [info@asa-rt.com](mailto:info@asa-rt.com) – Url: [www.asa-rt.com](http://www.asa-rt.com)

---

# ARP

## Control Programmable System

**SOFTWARE Manual**  
**Ver. 1.30**  
(FW ver. 3.00 on)

---



---

<b>1</b>	<b>INTRODUCTION.....</b>	<b>5</b>
<b>2</b>	<b>CONFIGURATION PARAMETERS.....</b>	<b>7</b>
2.1	Introduction.....	7
2.2	General Parameters.....	7
2.3	Axis Parameters.....	10
<b>3</b>	<b>DEFINITION OF HARDWARE RESOURCES.....</b>	<b>15</b>
3.1	ARP local resources.....	15
3.2	Remote resources on Can Bus.....	15
3.3	CARD ERRORS.....	16
<b>4</b>	<b>VARIABLES AND FLAG.....</b>	<b>19</b>
4.1	Introduction.....	19
4.2	System variables.....	19
4.2.1	System variables concerning the axes.....	20
4.2.2	System variables concerning the general parameters and the axis.....	20
4.2.3	System variables concerning the Flags.....	21
4.2.4	System variables concerning the digital outputs.....	21
4.2.5	System variables of digital inputs.....	21
4.3	User's variables.....	21
4.4	User's Flag.....	22
4.5	Flag of operator panel TR4/TR5 management.....	22
4.6	System Flag.....	25
<b>5</b>	<b>PROGRAMMING.....</b>	<b>27</b>
5.1	Program organization.....	28
5.2	Creation and use of a User's program.....	29
5.3	Logic and mathematical operations on variables and bits.....	30
5.4	Program instruction of the system.....	31
5.5	Organizing instructions.....	34
5.6	General instructions.....	38
5.6.1	ADC – Analog inputs reading.....	38
5.6.2	DAC – Analog outputs writing.....	38
5.7	Instruction for error management.....	39
5.7.1	ERROR – Program error code reading.....	39
5.7.2	WARNING – Program warning code reading.....	39
5.8	Generic regulation instructions.....	40
5.8.1	PID – Regulation from input for load cell.....	40
5.9	Axes movement instructions.....	43
5.9.1	Instruction of speed and acceleration setting.....	43
5.9.2	Instruction of axis initialisation.....	44
5.9.2.1	HOME – Cycle of value zeroing for incremental encoder.....	44
5.9.3	Instruction of independent movement.....	45
5.9.3.1	PAC – Absolute positioning.....	45
5.9.3.2	PRC – Relative positioning.....	45
5.9.3.3	PVC – Positioning in speed.....	46
5.9.3.4	STOP – Stop axis instruction.....	46
5.9.4	Instruction of synchronized movement.....	47
5.9.4.1	PST – Tabulated interpolator electronic cam.....	47
5.10	Timer management instructions.....	52
5.11	Instructions for flag and outputs management.....	53
5.12	Instruction for the management of the TR4 and TR5 operator terminal.....	55
5.13	Programming examples.....	58
5.13.1	Example for the use of the preliminary instructions.....	58
5.13.2	Example of PST instruction on simulated Master.....	59
5.13.3	Example of use of the PID instruction.....	60

---





## 1 INTRODUCTION

The ARP unit is a control system for two axes, suitable to manage simple machine cycles independently from a PLC, thanks to some analog and digital I/O.

Moreover, the presence of two interfaces for load cells, with the correspondent functions of tension regulation functions, allows to manage regulation systems for closed-ring tension.

By the connection on CANbus field it is possible to exchange information among several ARP, allowing the management of multi-axes synchronism and cycles co-ordinated among themselves. On the same bus it is possible to introduce I/O modules to expand the local resources of the system.

The card management is effected by a user's program written in a simil-basic language, integrated by complex functions for the movement management, mathematical calculations with the point fixed at 3 decimals, and the possibility to add routines connected to internal or external events (time interrupt, action on an input edge, etc...). The card on which the program is situated operates in an independent way.

It is important to point out that the ARP card task is to generate movements profiles and to close the regulation rings in position and speed state, using the Driver exclusively as current amplifier (control under current). The performances reached by the system are directly connected to the mechanical dimensioning and the motor-driver system of the machine.



## 2 CONFIGURATION PARAMETERS

### 2.1 Introduction

The parameters are values permanently filed in the Flash-eprom memory; they define the operative modalities of the ARP system and the characteristics of the outstanding application.

They are divided in:

- General Parameters ( @Par[...] ), including some global setting of the control card.
- Axis Parameters ( @ParX[...] , where X indicates the number of the axis ), with specific information for the management of the axis movement.

Some parameters are only in reading mode, while the most part of them are in reading/writing mode and they will have to be carefully compiled for the correct system functioning.

The here-under symbols will be defined in the chapter concerning the systems variables.

### 2.2 General Parameters

The general parameters concern the whole control system; so, they are not connected to the single axis, but, mainly, to the functioning mode of the CPU and to the communications with the external world.

**@Par[01] R/W Sampling period of the control loop (1÷10ms)**

It can be limitedly changed by the system User to optimize its functioning in connection with the kind of application to realize.

A short sampling time (1-2ms) can improve the high dynamic axes control to the detriment of the main program running speed. On the contrary, a higher sampling time (4-5ms) makes the running of the main program faster to the detriment of the particularly fast control axes.

*This parameter is acquired only at the starting of the system.*

**@Par[02] R/W Working Range of the load cell #1 (0 ÷ 2)**

**@Par[03] R/W Working Range of the load cell #2 (0 ÷ 2)**

These parameters indicate the working range used to program the interfaces for the load cell of the ARP unit; in particular

- 0 ⇒ Load cell not present
- 1 ⇒ Load cell operating within the range  $\pm 10$  mV
- 2 ⇒ Load cell operating within the range  $\pm 20$  mV

*These parameters are acquired only at the starting of the system.*

**@Par[04]** R/W **Acquisition filter frequency for load cell inputs** (50, 100, 250, 500, 1000 Hz)

50 ⇒ Output Rate 50 Hz ~ Freq. Stop 7 Hz ~ Freq. -3db 2 Hz  
 100 ⇒ Output Rate 100 Hz ~ Freq. Stop 14 Hz ~ Freq. -3db 4 Hz  
 250 ⇒ Output Rate 250 Hz ~ Freq. Stop 35 Hz ~ Freq. -3db 10 Hz  
 500 ⇒ Output Rate 500 Hz ~ Freq. Stop 70 Hz ~ Freq. -3db 20 Hz  
 1000 ⇒ Output Rate 1000 Hz ~ Freq. Stop 140 Hz ~ Freq. -3db 40 Hz

*This parameter is acquired only when the system is switched on.*

**@Par[05]** R/W **CAN Node Identifier** (1 ÷ 127)

This parameter sets the node identifier of the ARP card used on the CAN link

**@Par[06]** R/W **CAN Baud-Rate** (125, 250, 500, 1000 Kbaud)

This parameter sets the working baud-rate on the CAN link.

**@Par[07]** *Reserved*

**@Par[08]** R/W **CAN Identifier for TR4 / TR5** (1 ÷ 127)

This parameter indicates the terminal node identifier used on the CAN link.

**@Par[09]** *Reserved*

**@Par[10]** R/W **Value displayed on the diagnostic output #1** (1 ÷ 20)

**0** ⇒ Diagnostic disabled; analog output managed by **DAC** instruction.

**1** ⇒ Axis #1 reading encoder

**2** ⇒ Axis #1 positioning counter

**3** ⇒ Axis #1 speed measurement

**4** ⇒ Axis #1 positioning reference

**5** ⇒ Axis #1 speed reference

**6** ⇒ Axis #1 acceleration reference

**7** ⇒ Axis #1 reference towards the Driver

**8** ⇒ Axis #1 reference following error

**9** ⇒ Axis #1 delta reading encoder

**11** ⇒ Axis #2 reading encoder

**12** ⇒ Axis #2 positioning counter

**13** ⇒ Axis #2 speed measurement

**14** ⇒ Axis #2 positioning reference

**15** ⇒ Axis #2 speed reference

**16** ⇒ Axis #2 acceleration reference

**17** ⇒ Axis #2 reference towards the Driver

**18** ⇒ Axis #2 reference following error

**19** ⇒ Axis #2 delta reading encoder

- @Par[11] R/W Gain or attenuation for the diagnostic output #1**  
Multiplicative value for the diagnostic output
- @Par[12] R/W Moving for the diagnostic output #1**  
Additional value for the diagnostic output
- @Par[13] R/W Value displayed on the diagnostic output #2** ( 1 ÷ 20 )  
See description parameter @Par[10]
- @Par[14] R/W Gain or attenuation for the diagnostic output #2**  
Multiplicative value for the diagnostic output
- @Par[15] R/W Movement for the diagnostic output #2**  
Additional value for the diagnostic output
- @Par[16] R/W Period of the internal scheduler of the system** (10 ÷ 100ms)  
The scheduler manages the here-under routines:  
1. Management of the Enable input and anomalies test  
2. Management of the communication with the programming PC  
3. Management of the communication through the can-bus  
*The increasing of the value causes a delay in the above mentioned activities, but an acceleration in the User's program (optimal value 20 ms).*

**BE CAREFUL: These parameters are not accessible by the User's program**

## 2.3 Axis Parameters

For each axis of the system, it is available a set of parameters suitable to describe the functioning limits; so, it is very important to compile them carefully in order to grant the correct system functioning.

These parameters, as the General ones, are filed in a permanent way in the Flash-eprom memory and they are re-read at every starting and resetting of the system. If the User's program changes one of the axis parameters, the modification is done only in the RAM volatile copy, while the filed value in the Flash-eprom memory can be changed only through the WinAxis program in the *Configuration* window.

**BE CAREFUL: The uncorrected setting of the axis parameters can generate heavy malfunctioning of the control system of the axes; the operation must be done by skilful people and in safety conditions for people and things.**

**VERY IMPORTANT FOR THE SAFETY – The axis parameters must be valued and set in the system before doing the installation. It is possible that the first time the motor controlled by the ARP unit is switched on, it could have an abnormal behaviour due to a wrong parameters setting; therefore, it is necessary to be careful that it doesn't cause damages to people or things.**

The parameters available for each axis of the system, all accessible both in reading and in writing mode, are here-under listed

### **@ParX[01] Definition of the positioning measurement unit**

[Increases/Unit]  
(0.001 ÷ 999999.999)

This parameter allows to express the values of the positioning measurement unit suitable for the User; according to the different conditions, it is possible to choose measurement units for the linear systems (millimetres, centimetres, inches, etc...), for the rotating systems (degrees, revolutions, revolution hundredths, etc...)

Known the resolution of the encoder on the revolution, it is possible to define the measurement unit as

$$@\text{ParX}[1] = \frac{\text{Number of impulses on the encoder revolution} \times 4}{\text{Number of units on the encoder revolution}}$$

Example :

a linear system with direct-connected motor to endless screw with 10 mm thread; reduction ratio 1/12; encoder resolution 2000 impulses/revolution; if the output measurement unit is millimetre hundredths, it will be necessary to set:

$$@\text{ParX}[1] = \frac{2000 \times 4}{1000 \times (1/12)} = 96.000$$

BE CAREFUL : When the synchronised axes are working, it is necessary to make the resolutions of the two axes consistent between them (as regards the encoder impulses), in order to avoid noises in the generation of the movement profiles.

**@ParX[02] Number of encoder impulses / revolution** [Increases]  
(1 ÷ 100000)

It must be set the resolution on the revolution of the input encoder, indicated as

$$\text{Number of impulses on the encoder revolution} \times 4$$

**@ParX[03] Hardware management mode** (0 ÷ 3)

This parameter allows to correctly change the encoder reading with the analog output, in order to obtain a correct control of an external Driver; in particular:

- 0 ⇒ Direct encoder, direct analog output
- 1 ⇒ Reversed encoder, direct analog output
- 2 ⇒ Direct encoder, reversed analog output
- 3 ⇒ Reversed encoder, reversed analog output

**@ParX[04] Kind of axis management** (0 ÷ 3)

- 0 ⇒ Encoder reading only (default for both axes)
- 1 ⇒ Axis management in simulated encoder mode without the reading of the physical encoder
- 2 ⇒ Generation of the profile of the axis movement
- 3 ⇒ Control mode from PID instruction

This parameter defines the kind of the axis management; in particular:

- In “0” mode the axis is not controlled but it is read its position
- In “1” mode it is possible to effect on the axis all the movement functions, without generating any external references
- In “2” mode the external Driver is piloted with a current reference on the analog output.
- In “3” mode the axis is not managed by the positioning profiles with feedback from encoder, but by a PID regulation instruction with feedback from load cell.

**@ParX[05] Maximal value of the positioning counter** [Unit]

**@ParX[06] Minimal value of the positioning counter** (-999999.999 ÷ +999999.999 )

These two parameters define the working range of the positioning counter. The above mentioned counter is always managed in cyclic mode between these two values; if it is willing to operate with absolute values, these parameters must be set at their extreme limits.

**@ParX[07] Maximal speed axis** [Unit/second]  
( 0.001 ÷ 999999.999 )

Maximal value of the speed reached by the system motor - mechanics

**@ParX[08] Maximal acceleration and deceleration of the axis** [Unit/second<sup>2</sup>]  
( 0.001 ÷ 999999.999 )

This value is essential to obtain good performance of the controlled system: a too high value will easily cause a tracking error, while a too low value, even if it will cause a milder regulation, won't allow high dynamic performances of the axis itself.

This parameter must be set according to the real physical performances of the driver (provided torque and moment of inertia of the mechanical system), considering a 10-20% margin necessary to grant the regular functioning of the control system.

**@ParX[09] Reserved**

**@ParX[10] Gain on the feed-forward of the regulation PID filter** (0 ÷ 999999.999)

**@ParX[11] Proportional gain of the regulation PID filter (KP)** (0 ÷ 999999.999)

**@ParX[12] Integral gain of the regulation PID filter (KI)** (0 ÷ 999999.999)

**@ParX[13] Derivative gain of the regulation PID filter (KD)** (0 ÷ 999999.999)

These parameters are the factors of the PID filter for the movement control and regulation of the two axes. The filter calculates a current reference toward the Driver starting from a tracking error

**CAUTION : It is important to be very careful during the setting of these parameters, in order to avoid damages to things and people**

**@ParX[14] Maximal admitted tracking error** [ Unit ]  
(0 ÷ 999999.999)

In this parameter it is set the maximal tracking error tolerated by the system before generating the error of the axis control loss (escaping axis)

**@ParX[15] Bonding time for the arrived axis** [ Seconds ]  
(0 ÷ 60.000)

Permanence period of the axis inside the window for the arrival, before the flag check of *arrived axis*

**@ParX[16] Bonding tolerance for the arrived axis** [ Unit ]  
(0 ÷ 999999.999)

Amplitude of the window for the arrived axis

**@ParX[17] Reserved**

**@ParX[18] Reserved**

**@ParX[19] Offset of the axis reference** [ Unit ]  
(-999999.999 ÷ +999999.999)

If the axis is used as Slave for the synchronism instructions, through this parameter it is possible to execute a reference translation; this is an additional parameter and its change through the User's program will cause a synchronism discontinuity, which will be recovered by the control with a sudden acceleration.

The offset correction is an absolute value; so, if it is willing to have incremental changes in correspondence of an initial position obtained with @ParX[11] = 0, it is necessary to accumulate the increases in a variable and each time set it in the parameter itself.

**@ParX[20] Maximal offset of the axis reference admitted for the sampling cycle** [ Unit ]  
(0 ÷ 999999.999)

Through this parameter it is possible to regulate the offset inlet on the present value, to subdivided along the time the error range generated by changing the previous parameter in order to avoid sudden accelerations of the axis.

If this parameter is different from zero, it specifies the maximal correction for each sampling cycle (defined in @Par[01]) to recover the positioning error due to a change of parameter @ParX[11].

**BE CAREFUL : If the parameter is equal to zero, there won't be any offset recovery**

**@ParX[21] Gear ration in reading mode of the Master axis** (-10.000 ÷ 10.000)

Through this parameter it is possible to re-calculate the positioning reading and the Master axis speed in presence of “PST” instructions, without altering the working limits of the counter of the Master axis itself. In this way the virtual Master will be faster and lower than the real one, but there will be a loss of the correspondence between the real values and the ones used by the “PST” instructions.

**@ParX[22] Offset of the Master axis reading** [ Master Unit]  
( -999999.999 ÷ 999999.999 )

In presence of “PST” instructions, the value included in the present parameter will be added to the position read by the Master axis.

**@ParX[23] Maximal offset of the Master axis reading admitted by the sampling cycle** [ Master Unit]  
( 0 ÷ 999999.999 )

Analogous to parameter @ParX[12].

If this parameter is different from zero, it specifies the maximal correction for each sampling cycle (defined in @Par[01]) to recover the positioning error due to a change of parameter @ParX[14].

**BE CAREFUL : If the parameter is equal to zero, there won't be any offset recovery**

**@ParX[24] Maximal speed of Master axis for synchronism** [Master Unit/second]  
( 0.001 ÷ 999999.999 )

In presence of “PST” instructions, the maximal speed reachable by the Master axis will be set in this parameter, in order to size conveniently the **connections of the instruction itself**.

**@ParX[25] Speed for cam entering with the stopped Master** [% di @ParX[07] ]  
( 0.001 ÷ 100.000 )

Speed to execute the cam entering in case of activation of a “PST” instruction with the stopped Master.

### 3 DEFINITION OF HARDWARE RESOURCES

The ARP program uses the here-under symbolic mapping to enter in the local and Can Bus remote hardware resources.

#### *3.1 ARP local resources*

The local ARP hardware resources are mapped as follows:

- The 8 digital inputs are referred as I101 ... I108 (I108 is the “Enable” input).
- The 4 digital outputs are referred as O101 ... O104.
- The two inputs for load cell are read through the instructions **adc 1** and **adc 2**.
- The two analog inputs  $\pm 10$  V are read through the instructions **adc 3** and **adc 4**.
- The analog inputs 0...10 V are read through the instructions **adc 5** and **adc 6**.
- The two analog outputs  $\pm 10$  V are read through the instructions **dac 1** and **dac 2**.

#### *3.2 Remote resources on Can Bus*

The remote resources that can be managed by the ARP unit are mapped as follows:

- I/O node number 1 → inputs from I201 to I232, outputs from O201 to O232
- I/O node number 2 → inputs from I301 to I332, outputs from O301 to O332

### 3.3 CARD ERRORS

In the condition of enabled system (“Enable” input activated), the User’s program is sequentially run starting from the present instruction. The standard functioning can be stopped in one of the following ways:

- The error state (Reset Error) is reached if the “Enable” input is not present any more
- The Reset state is reached if “Ctrl-F2” is pressed by the WinAxis PC User program

The “Enable” input is always activated to allow the system functioning. When the above mentioned input is closed, every eventual critical errors happened during the standard functioning are cancelled, by restoring the initial state of the system.

When the “Enable” input is not activated, the system is in the RESET state with the following conditions:

- The axes are NOT controlled.
- The free digital output are at zero level.
- The analog outputs are at zero tension.
- The User’s program is stopped and at its first instruction.

**BE CAREFUL: If the “Enable” input goes low during an axis positioning, the control on the movement itself will be lost and so, according to the different cases, it will be necessary to stop the motors with external means.**

In Reset condition the control system is stopped, the outputs are not enabled, but there is no error.

The serial communication with the PC or with the Host are always active, whatever the state of the system and the “Enable” input; the remote devices can ask in each moment the state code from which it is possible to take over the system condition in Reset.

If a critical error occurs on the system the outputs are disabled and the control loops on the axes are disabled, but, when possible, the execution of the User program is not stopped. If the error is not critical, the system will generate only a warning message, but it is not granted that the instruction functioning is the one desired.

A part from the blocking errors, the management both of the error condition and of the warning one, is completely carried out by the user’s program, in order to fit at best the different possible operational situations; through the “ERROR” or “WARNING” instructions, a numerical code, univocally associated to each error, is read by the user’s program, allowing a convenient management of the errors themselves.

The Error or Warning condition is zeroed by the opening and closing cycle of the “Enable” input or by the Ctrl-F2 command sent to the PC, through the Debug environment of WinAxis.



The ARP error and warning codes, with the numerical codes indicated by the “ERROR” and “WARNING” instructions, are here-under listed; the blocking errors are identified in the third column of the table, by B letter :

<b>E01</b>	<b>101</b>		<i>Reserved</i>
<b>E02</b>	<b>102</b>		<i>Reserved</i>
<b>E03</b>	<b>103</b>	<b>B</b>	Reading/writing error on permanent EEPROM memory
<b>E04</b>	<b>104</b>		<i>Reserved</i>
<b>E05</b>	<b>105</b>	<b>B</b>	System in Reset state
<b>E06</b>	<b>106</b>		CAN-BUS error
<b>E07</b>	<b>107</b>		<i>Reserved.</i>
<b>E08</b>	<b>108</b>		<i>Reserved.</i>
<b>E09</b>	<b>109</b>		<i>Reserved.</i>
<b>E10</b>	<b>110</b>		<i>Reserved</i>
<b>E11</b>	<b>111</b>	<b>B</b>	Anomalous stop of the User’s program
<b>E12</b>	<b>112</b>	<b>B</b>	Stack error of the User’s program
<b>E13</b>	<b>113</b>		Instruction parameters error
<b>E14</b>	<b>114</b>		<i>Reserved</i>
<b>E15</b>	<b>115</b>		<i>Reserved.</i>
<b>E16</b>	<b>116</b>		Axis control loss error (escaping axis)

<b>W01</b>	<b>201</b>	Division by zero
<b>W02</b>	<b>202</b>	Overcoming of the numerical range permitted ( $\pm 999999.999$ )
<b>W03</b>	<b>203</b>	Overcoming of the instruction parameters range permitted
<b>W04</b>	<b>204</b>	Permanent eeprom memory not initialized



## 4 VARIABLES AND FLAG

### 4.1 Introduction

The data used by the control unit and by the User's program are included in two different fields:

- System variables and flags
- User's variables and flags

The system variables and flags have specific meanings and functions and they allow to receive information and give commands to the control system; some of them are in reading-writing (R/W) state and some only in reading (R) state. On the contrary, the User's variables and flags are freely applied by the user to manage his application.

The variables are 32bit whole values; if they are used in the decimal format, they are automatically re-calculated with 1000 factor and, therefore, they could have a  $\pm 999999.999$  value. If they are used in the hexadecimal format, they can be included between 0 (zero) and HFFFFFFF.

Ex. Introducing 5 in the variables, the recorded value will be 5.000 equal to H1388 in the hexadecimal format.

Normally, the flags, the digital inputs and outputs are considered single bit, but, through the system variables, they be also considered as words, using the normal techniques of bit masking.

Ex.  $V1 == @FLAG1$  ; it reads the first word of flag  
 $V1 == V1 \& H00FF$  ; it masks the highest 24 bits to use the lowest 8 bits

The system variables are volatile and they are continuously updated.

### 4.2 System variables

The system variables are identified by a special prefixed character:

“@” for those in reading – writing mode

“#” for those in reading mode

NB. The “n” parameter provided with the system variables must have no spaces from the name of the variable itself and can't be provided in the symbolic form.

#### 4.2.1 System variables concerning the axes

- **@POS<sub>n</sub>** : [U] Positioning reference of the axis “n” in engineering unit. This value represents the positioning reference (theoretical quote) and it doesn’t include the positioning error almost always present in the real system. The positioning reading and setting effected by the program, is done by giving a variable:

**Ex.**

V05 == @POS1 ;it reads the positioning of the axis 1 in the variable V05

@POS2 == V06 ;it sets the positioning of the axis 2 in the variable value V06

- **#VEL<sub>n</sub>** : [U/s] Speed of the axis “n” in unit/second
- **#LQOSP<sub>n</sub>** : [U] Value of the axis “n” read in correspondence of the positive edge of the latch-quote input (the physical correspondence of the above mentioned input depends on the considered hardware)
- **#LQOSN<sub>n</sub>** : [U] Value of the axis “n” read in correspondence of the negative edge of the latch-quote input (the physical correspondence of the above mentioned input depends on the considered hardware).

#### 4.2.2 System variables concerning the general parameters and the axis

- **@ Par[m]** : m-esimal General parameter
- **@ ParX[m]** : m-esimal “X” Axis parameter

With these system variables, the User’s program can acquire and change the parameters value used by the control system during its functioning.

The parameters writings executed by the User’s program don’t change the basic value of the parameter present in the permanent memory (i.e. the writings do not change the original value of the flash-eprom memory), but only a copy of the value itself used by the control system during the running phase. Each time the system is switched on and reset, the parameters are set in the permanent flash-eprom memory.

### 4.2.3 System variables concerning the Flags

The User and System Flags are usually used in the program as single bit data, but they can also be used in the 32 bit word format.

- **@FLAG1** : 1<sup>a</sup> Flags word (User) from F001 to F032
- **@FLAG2** : 2<sup>a</sup> Flags word (User) from F033 to F064
- **@FLAG3** : 3<sup>a</sup> Flags word (User) from F065 to F096
- **@FLAG4** : 4<sup>a</sup> Flags word (User) from F097 to F128
- **@FLAG5** : 5<sup>a</sup> Flags word (User) from F129 to F160
- **@FLAG6** : 6<sup>a</sup> Flags word (User) from F161 to F192
- **@FLAG7** : 7<sup>a</sup> Flags word (User) from F193 to F224
- **@FLAG8** : 8<sup>a</sup> Flags word (User) from F225 to F256
- **@FLAG9** : 9<sup>a</sup> Flags word (System axis #1) from F257 to F288
- **@FLAG10** : 10<sup>a</sup> Flags word (System axis #2) from F289 to F320

### 4.2.4 System variables concerning the digital outputs

These variables allow the command of the digital outputs in words mode.

- **@OUT1** : 1<sup>o</sup> block (32bit word) of physical outputs ⇒ from O101 to O132
- **@OUT2** : 2<sup>o</sup> block (32 bit word) of physical outputs ⇒ from O201 to O232
- **@OUT3** : 3<sup>o</sup> block (32 bit word) of physical outputs ⇒ from O301 to O332

### 4.2.5 System variables of digital inputs

These variables allow the reading of the digital inputs in words mode

- **#INP1** : 1<sup>a</sup> 32 bit words of the User digital inputs ⇒ from I101 to I132.
- **#INP2** : 2<sup>a</sup> 32 bit words of the User digital inputs ⇒ from I201 to I232
- **#INP3** : 3<sup>a</sup> 32 bit words of the User digital inputs ⇒ from I301 to I332

## 4.3 User's variables

In order to record the data of his application, the User has at his disposal 1000 variables (from V1 to V1000 in the format ±999999.999); they are all permanent and their value is maintained also after the system switching off .

The variables can include every kind of numerical data (ex. speed, acceleration, values, calculations, comparison values, time, etc.). They can be directly recalled indicating the correspondent number, or indirectly recalled through the contents of another variable acting as index (indexing mode). In this case, the indexing variable must be inscribed within two square brackets [].

Ex.

```

V1    ==V100           ;assignment among variables
V1    ==(V100 + V101) / V5 ;mathematical operations with assignment
PAC ASSE1, V100       ;positioning at the V100 value
V1    ==50
V10   ==V[V1]         ;the variable V10 receives the content of the variable 50

```

#### 4.4 User's Flag

For the User's program there are 217 volatile flags available (from F1 to F217), useful to record true-false information; they can be set through the functions "SET *Fxxx*" and "RES *Fxxx*", assigned by the operator "=", and verified by the User's program with the function "IF *condition, routine*"

Exs.

```

If F10 = lo, set F101 ; Test on level and enabling
F112 == I102         ; Assignment through an input
F23  == F[V2]       ; Assignment through the flag addressed by the variable V2

```

#### 4.5 Flag of operator panel TR4/TR5 management

In order to manage the operator program, the User's program uses 39 volatile flags, where 7 of them are reserved for the management of the LED on the operator panel (from F218 to F224) and 32 of them contain the images of the keys pressed on the panel (from F225 to F256). If the operator terminal is not present, all these flags can be used as normal User's flags.

The mapping of these flags is listed here-under

		<b>TR4</b>	<b>TR5</b>
<b>F218</b>	R/W		LED F1 state of the terminal
<b>F219</b>	R/W	LED F12 state of the terminal	LED F2 state of the terminal
<b>F220</b>	R/W	LED F11 state of the terminal	LED F3 state of the terminal
<b>F221</b>	R/W	LED F10 state of the terminal	LED F4 state of the terminal
<b>F222</b>	R/W	LED F9 state of the terminal	LED FA state of the terminal
<b>F223</b>	R/W	LED F8 state of the terminal	LED FB state of the terminal
<b>F224</b>	R/W	LED F7 state of the terminal	LED AUTO state of the terminal

<b>F225</b>	R	Key “1”	Key “1”
<b>F226</b>	R	Key “2”	Key “2”
<b>F227</b>	R	Key “3”	Tasto “3”
<b>F228</b>	R	Key “4”	Key “4”
<b>F229</b>	R	Key “5”	Key “5”
<b>F230</b>	R	Key “6”	Key “6”
<b>F231</b>	R	Key “7”	Key “7”
<b>F232</b>	R	Key “8”	Key “8”
<b>F233</b>	R	Key “9”	Key “9”
<b>F234</b>	R	Key “0”	Key “0”
<b>F235</b>	R	Key “. (point)”	Key “. (point)”
<b>F236</b>	R	Key “- (minus)”	Key “- (minus)”
<b>F237</b>	R	Key “CR (return)”	Key “CR (return)”
<b>F238</b>	R	Key “ESC”	Key “ESC”
<b>F239</b>	R	Key “END”	Key “SHIFT”
<b>F240</b>	R	Key “DEL”	Key “DEL”
<b>F241</b>	R	Key “←”	
<b>F242</b>	R	Key “→”	
<b>F243</b>	R	Key “↓”	Key “↓”
<b>F244</b>	R	Key “↑”	Key “↑”
<b>F245</b>	R	Key “F1”	Key “F1”
<b>F246</b>	R	Key “F2”	Key “F2”
<b>F247</b>	R	Key “F3”	Key “F3”



---

<b>F248</b>	R	Key "F4"	Key "F4"
<b>F249</b>	R	Key "F5"	Key "FA"
<b>F250</b>	R	Key "F6"	Key "FB"
<b>F251</b>	R	Key "F7"	Key "AUTO"
<b>F252</b>	R	Key "F8"	
<b>F253</b>	R	Key "F9"	
<b>F254</b>	R	Key "F10"	
<b>F255</b>	R	Key "F11"	
<b>F256</b>	R	Key "F12"	

## 4.6 System Flag

The system flags are defined in order to provide the User's code with the information concerning the axes state on the control system, and to let the User's code itself to enable or disable specific running options. They can be examined through the instruction "IF *condition, instruction/label*", using as condition the state of one of them, and/or they can be set, if possible, through the instruction "SET *flag*" and "RES *flag*".

The flags which can be set by the User's program are indicated with "R/W", while the flags written only by the system are indicated with "R".

Each axis managed by the ARP unit is joined to a set of 32 flags which allow to determine and change the operative situation of the single axis.

- F257...F288 : **flags of Axis 1 (Controlled Axis)**
- F289...F320 : **flags of Axis 2 (Master Axis)**

NOTE: The flags in grey are the ones that are not managed on the present axis

<b>F257</b>	<b>F289</b>	R	<i>Reserved</i>	
<b>F258</b>	<b>F290</b>	R	<b>Home in process</b>	If at the HI level, it indicates that the zeroing of the values is in process (HOME)
<b>F259</b>	<b>F291</b>	R	<b>Home finished</b>	If at the HI level, it indicates the positive finished process of the values zeroing (HOME).
<b>F260</b>	<b>F292</b>	R	<b>Axis arrived</b>	As soon as a movement instruction begins, the flag is put at LO level while it is at HI level as the axis reaches the assigned position. The flag is enable by the control function (even if there are no movement instructions in that moment) only when it is activated; therefore, if the axis in not controlled, the arrived flag is not managed.
<b>F261</b>	<b>F293</b>	R	<b>Axis in synchronism</b>	In case of "PST" instruction, this flag is put to a high level after the Slave axis has entering in the cam phase and it returns to low level at the end of the instruction itself.
<b>F262</b>	<b>F294</b>	R/W	<b>Stop axis</b>	If it is put at the high level, it stops the movement instruction in process ("PRC", "PAC", "PVC", "PST") keeping the axis at the final speed; therefore, the axis has to be conveniently managed with the instruction of the User's program. The movement instructions automatically zero the axis in order to allow the starting of the movements itself.

<b>F263</b>	<b>F295</b>	R	<b>Movement starting</b>	This flag is put to a high level at the beginning of an execution of a movement function and it is brought to a low level at the end of the instruction itself.
<b>F264</b>	<b>F296</b>	R	<b>Axis in final speed</b>	It is lifted up by the control to the end of an instruction which leaves the axis in movement in final speed because of the end of the instruction itself or following a setting of the stop axis flag by the User's program.
<b>F265</b>	<b>F297</b>	R/W	<b>Enable positive latch-quote</b>	If it is activated by the User's program, this flag enable the function of latch-quote capture <i>on the rising edge of the reserved input</i> . It is disable by the control to communicate to the program that the latch has occurred.
<b>F266</b>	<b>F298</b>	R/W	<b>Enable negative latch-quote</b>	If it is activated by the User's program, this flag enable the function of the latch-quote capture <i>on the falling edge of the reserved input</i> . It is disable by the control to communicate to the program that the latch has occurred.
<b>F267</b>	<b>F299</b>	R/W	<b>Cycle passing on the minimal value</b>	It indicates the passage of the positioning counter under the admitted maximal value and, as a consequence, the bringing of the counter at its minimal value. This flag is put at a high level from the system and it will be brought to the low level by the User's program.
<b>F268</b>	<b>F300</b>	R/W	<b>Cycle passing to the minimal value</b>	It indicates the passage of the positioning counter under the admitted minimal value and, as a consequence, the bringing of the counter at its maximal value. This flag is put at a high level from the system and it will be brought to the low level by the User's program.

Table 1 – System flags concerning the axes

## 5 PROGRAMMING

The ARP unit behaviour must be personalized according to the required specification of the application, by a User's program which will be filed in the permanent memory; the development and the test of the programs must be done with a PC connected to the USER port in serial line.

The program is composed by instructions (or commands) and by comments according to the program Developer's willing. The WinAxis compiler will translate the source program to a running program, which will be sent to the ARP unit and executed in real time.

During the running phase, the program operates with variables, flags, inputs and outputs in order to manage completely the resources connected to the system; moreover, it is possible to effect variously complex mathematical expressions, with brackets to realize the needed calculations

Ex.  $((V01 + V02) / V03) + V04$

In order to optimize the project, the program can be structured with a main routine plus different complementary routines (subprograms reachable by the main program) which realize auxiliary functions. It is also possible to have subprograms activated by asynchronous events (interrupt routines).

Variables, inputs, outputs, flags and program lines can be personalized by a symbolic name (label) in order to make them and their management understandable. However, the assigned labels must be univocal inside the source file for each application.

The numerical data inserted in the instructions or in the expressions of the program, can be supplied in decimal or hexadecimal format; in the hexadecimal format they must begin with "H" or "h".

Ex.            1000.000    ; decimal value,  
                 H1000        ; hexadecimal value (equal to the decimal 4096).

The following predefined constant are also available:

**PIG** = pi  $\pi$  (3.142)      **HI** = **ON** = 1      **LO** = **OFF** = 0



---

## 5.1 Program organization

The ARP program is a file with ASCII printable characters; it is characterized by a main program, i.e. a first routine of the source file which begins with “BEG *program name*” and ends with END”, and a series of subprograms.

It can be written with the WinAxis editor or any other editor that does not enter unprintable control characters which could cause errors during the compiling phase.

The subprograms begin with “BEG *name*” and ends with “END” and they can be recalled by the main program or another subprogram.

The subprograms associated to an asynchronous event are indicated with the initial “BEGA *name*” and ends with ENDA”. They are executed by stopping the main program to have quick answer to sudden events.

Each text line is considered as a User’s program instruction up to an eventual “;” (semi-colon) which identifies the beginning of a comment. The comment ends at the end of the line which must be closed by an ENTER; the maximal length of a program line is 120 characters with spaces included.

Each line of the source file can contain a single instruction (plus an eventual comment); the instruction can’t be written on two lines and they are not considered if they are written on the same line after the comment.

The program line can be identified by a special “*name*” (label) which must end with “:” (colon); the line label is necessary to execute program branches.

```
Ex.          .....          ;other instructions

L00:
    IF VI > 100 L01      ;it jumps to L01 if the condition is verified
    VI += 1              ;
    .....              ;other instructions

L01:
    SET O101            ;
    GOTO L00            ;it jumps to the line beginning with L00:
```

## 5.2 *Creation and use of a User's program*

In order to generate an application for the ARP unit, it is necessary to execute the following operations through the developing environment WinAxis supplied by ASA-RT:

- Write the program with the ASCII editor in the actual environment.
- Compile the program to obtain the “object program”, correcting it in case of syntax errors pointed out by the compiler.
- Send the program to the ARP unit through the “Download” function; if the system is not in program STOP condition, it is necessary to bring them to this condition, by opening the Enable input on the Driver or through the program itself.
- Run the program by closing the Enable input on the Driver or by the developing environment.
- Check the generated program using the environment debug functions.

At each change effected on the source program, it is necessary to repeat the above mentioned operations in order to obtain the result of the change itself.

The source program is filed in the directory selected for the data file with the set name and with a termination including the CPU number to which it refers (ex. “name.I01” for the program of the system with communication address 1). Thus, it won't be possible to send a program to the wrong system, unless the program name is voluntarily changed.

The here-under system files are created in the directory with different terminals (“nn” indicates the system number to which the program is addressed) :

name.Dnn = compiled program file (ASCII)

name.Enn = compiling errors file

name.Inn = source program file

name.Mnn = map file (reserved)

name.Onn = compiled program file (binary - reserved)

name.Snn = file containing the symbols (reserved)

### 5.3 Logic and mathematical operations on variables and bits

All the following logic and mathematical operators are available to execute operations on variables and flags in the User's code:

==	Assignment	Ex. V010 == V001
+, -, *, \	The four basic operations	Ex. V1 == V2 * (V10 + V15)
+=, -=, *=, \=	Operation with assignment	Ex. V2 += V10 $\Rightarrow$ V2 == V2 + V10
&,  , ^	Logic operations And, Or, exclusive Or.	Ex. V10 == V1 & V2 Ex. F12 == I101   F1
&=,  =, ^=	Logic operation And, Or, exclusive Or with assignment	Ex. V33  = V3
%	Module operator	Ex. V1 == 10 % 4 $\Rightarrow$ V1 == 2
%=	Module operator with assignment	
<<, >>	Left or right shift of a variable <sup>1</sup> .	
<<=, >>=	Left/right shift with assignment <sup>1</sup>	
=	Sign of equality	
<>	Sign of inequality	
<, >	Strictly minor, or strictly major	
<=, >=	Minor or equal to, major or equal to	
!	Complementary to 1 (not)	Ex. F1 == !I123
<b>SQR</b>	Square root of a variable	Ex. V1 == SQR(V455)
<b>INT</b>	Whole part of the value of a variable	Ex. V2 == INT(V10) if V10 = 856.56 $\Rightarrow$ V2 = 856
<b>SEN</b>	Angle sine in degrees	Ex. V1 == SEN(V58)
<b>COS</b>	Angle cosine in degrees	Ex. V1 == COS(V23)

<sup>1</sup>The shift instructions are used when the variable content includes a datum that can be valued to single bits (such as the reading of the digital inputs effected by words). On the opposite side of the shift, a 0 (zero) is introduced at the last place of the last bit shifted to the previous one.

## 5.4 Program instruction of the system

The instruction composing the ARP programming language can be divided in the here-under subgroups, according to their function

- Instructions for the organization of the program
- General instructions
- Axes movement instructions
- Timer management instructions
- Flags/Inputs/Outputs instructions

The following tables show all the system instructions; all data indicated in the description of each instruction must be supplied to complete the language syntax, while the data included in the brackets {...} are optional.

The symbol of vertical bar “|” splits two possibilities that can be provided alternatively; ex. *instruction | label*, it means that it is possible to indicate a language instruction OR a label concerning a program line suitable to execute a jump.

The indications written in the round brackets (...) can concern the measurement unit or any other explanation and they are not included in the syntax of the instructions themselves.

<b>Instructions for the organization of the program</b>	
<b>BEG</b> <i>name</i> <b>END</b>	They define the beginning and the end of a program or subprogram
<b>BEGA</b> <i>name</i> <b>ENDA</b>	They define the beginning and the end of an interrupt routine (On – Event).
<b>OnX</b> <i>condition, routine name</i>	X-esimal enabling for the execution of a routine in interrupt
<b>GOTO</b> <i>name</i>	Unconditional jump to the identifier “ <i>name</i> ”.
<b>CALL</b> <i>name</i>	Calling of a subprogram to start its execution
<b>IF</b> <i>condition, instruction/label</i>	Function of conditional execution or conditional jump
Label	Identifier of program line
<i>Name</i> <b>EQU</b> <i>identifier</i>	Assignment of a symbolic name to variables, parameters, inputs, outputs, etc.

<b>General instructions</b>	
<i>Variable == ADC channel</i>	Analog input reading
<b>DAC</b> <i>canale, volt, {settling time}</i>	Analog output setting

<b>Axes movement instructions</b>	
<b>HOME</b> <i>axis</i>	Cycle of value zeroing
<b>VEL</b> <i>axis, value (%)</i>	Maximal speed setting
<b>ACC</b> <i>axis, value (%)</i>	Maximal acceleration setting
<b>DEC</b> <i>axis, value (%)</i>	Maximal deceleration setting
<b>PAC</b> <i>axis, position</i>	Absolute positioning
<b>PRC</b> <i>axis, distance</i>	Relative positioning
<b>PVC</b> <i>axis, final speed</i>	Movement in speed condition
<b>PST</b> <i>axis M, axis S, V_tab</i>	Electric cam between two axes
<b>STOP</b> <i>axis</i>	Immediate stop of an axis

<b>Timer management instructions</b>	
<b>WAIT</b> <i>time</i>	It interrupts the program running for “time” seconds.
<b>STIM</b> <i>n</i>	It zeroes and restarts the timer “n”
<b>STIM</b> <i>n, time, flag</i>	It sets and makes a timer starting through a flag enabling
<b>TIM</b> <i>n</i>	Value reading of the timer in seconds “n”

<b>Flags/Inputs/Outputs instructions</b>	
<b>SET</b> <i>Flag/Output</i>	Flag or output enable (at high level)
<b>RES</b> <i>Flag/Output</i>	Flag or output disable (at low level)
<b>CAM<sub>n</sub></b> <i>Axis, Qi, Qf, Output</i>	Electric cam which enables an output between the values Qi and Qf

<b>Instructions for the error management</b>	
<i>Variable</i> == <b>ERROR</b>	Reading of the error codes of the program
<i>Variable</i> == <b>WARNING</b>	Reading of the warning codes of the program

<b>Instructions for the TR4 / TR5 operator panels management</b>	
<b>MSG</b> <i>nr_msg, "string"</i>	It records a string on the panel
<b>TERM</b> <i>{line, column},{height, nr_msg}, {datum}, {mode}</i>	It displays a string on the panel with specific position and dimension
<b>PRINT</b> <i>nr_msg, {datum}</i>	It transfers a string to the output of the serial print of the panel, with an eventual numerical datum

Each time an instruction requires a numerical value, this value can be directly provided or supplied through a value of a variable; in many cases it is possible to provide this datum by a variable used as index. The index mode is suitable both for the variables and for the Flags, Inputs and Outputs.

**Ex. the here-under instructions are valid:**

V1 == 55	<b>;assignment with constant</b>
V1 == V2	<b>;assignment through variable</b>
V2 == V[V10]	<b>;index assignment through V10</b>
PAC 1, V3	<b>;positioning with data in the variables</b>
WAIT V10	<b>;holding time with variables</b>
V[V1] == 0	<b>;indexed variables zeroing</b>
F[V1] == lo	<b>;assignment to an index flag through V001</b>
F100 == I[V10]	<b>;assignment to a flag by an index input</b>
O[V20] == F20	<b>;assignment to an index output</b>

## 5.5 Organizing instructions

**BEG name ..... END**

BEG e END are compulsory to identify the beginning and the end of the main program and each subprogram. The main program (i.e. the one which runs the starting execution) must be the first file of the application, then, it will be possible to write the subprograms.

Ex.

```
BEG PROG1           (starting of the main program)
.....
CALL SUB01
.....
CALL SUB02
.....
END                 (ending of the main program)

BEG SUB01
.....              (1^ subprogram)
END

BEG SUB02
.....              (2^ subprogram)
END
```

The order of the subprograms inside the source file is not important, but the subprograms must follow the main program.

**BEGA name .....ENDA**

They define the beginning and the end of the routines called in Interrupt in comparison with the main program.

These subprograms **CANNOT** be called by the normal function CALL, but their functioning is prepared by the instruction “Onn ...” (see the following instruction) and they are executed only in interrupt.

## ONn enabling & routine condition

The instruction “ONn ...” enables the routine number “n” (1 ÷ 6), which will be run in interruption in comparison with the main program; it is a preliminary instruction and, therefore, it shall be executed only one time to activate the requested routine (the routine or the function must begin with BEGA and end with ENDA).

The indicated subprogram in the instruction with “routine” is executed by interrupting the main program when the “condition” indicated has occurred; a standard use of this instruction implies that the called subprogram has a limited length, unless the main program won’t be correctly run.

The first parameter is an enabling flag : the “condition” will be tested only if the enabling flag is active. The User’s program can anytime enable and disable the routine in interrupt.

The “condition” can be one of the here-under listed:

- Fyyyy or Innn : the condition is true if the flag or the input are at high level
- !Fyyyy or !Innn : the condition is true if the flag or the input are at low level
- /Fyyyy or /Innn : the condition is true at the rising edge of the flag or input
- \Fyyyy or \Innn : the condition is true at the falling edge of the flag or input
- Axis position > Vxxxx : the condition is true if the axis value is higher than the one set in the variable Vxxxx
- Axis position < Vxxxx : the condition is true if the axis value is lower than the one set in the variable Vxxxx

Vxxxx must include for all the time in which the corresponding “ONn” function is active, the comparing value. The test on the condition will be done on the present value of the variable and not on the one included in it at the time of the setting of instruction “ONn”. This allows to change the variable value, set as parameter on instruction “ONn” during the program running (run-time) without recalling the instruction itself.

It is important to consider that if the condition is set on a level and is verified, the condition flag will have to be disable as soon as the interrupt routine has been entered; otherwise, there will always be the interrupting phase which will cause the main program stopping.

The execution of the instruction “ONn” without any other parameters disables a previous setting of the routine in interrupt.

The execution of an instruction “ONn” with the name number as an active instruction, directly replaces the previous execution by setting the new parameters.

## **GOTO** *name*

Unconditional jump to the identifier (name) of a program line. The program jump can occur both backwards and forwards, but inside the same program or subprogram.

It is not possible to execute jumps between the main program and the subprograms or among the subprograms; only the CALL instruction allows the intervention of the subprograms.

If it is necessary to execute the program jumps according to a condition, the “IF” instruction must be used.

## **CALL** *name*

The CALL instruction recalls the execution of a subprogram, which will start from the first instruction and will always end to its END. When the subprogram has ended, the execution will be brought to its CALL point and will go on from the following instruction.

A subprogram can recall another subprogram up to 10 levels.

See the example proposed for the instructions BEG...END

## **IF** *condition, instruction | name*

Execution of instruction or jump to a program line according to the result of a condition. The condition can be one of the following:

For the bits (inputs or flags):

- *bit1 { & bit2 } { | bit3 } = HI* ;high level test
- *bit1 { & bit2 } { | bit3 } = LO* ;low level test

For the variables:

- *var1 = var2* ;equal sign among the variables.
- *var1 <> var2* ;unequal sign among the variables
- *var1 <, > var2* ;strictly minor or major among the variables
- *var1 <=, >= var2* ;minor or equal, major or equal among the variables

The condition can be set between two variables or between a variable and a direct value, independently from the variables or addressing types; the condition among numerical data are not allowed.



If the condition is verified, the instruction or jump indicated in “*instruction | name*” is executed; if the condition is not verified, the part after the come is ignored and the execution goes on with the instruction of the following line

Ex.

```
IF V100 = V10, VI == 0
IF V5 < 100, CALL SBR01
IF I101 = LO, V10 == 10
IF (I101 & F10) = HI, SET O101
etc ...
```

### label

Identifier of program line; it has to be followed by “:” (colon)

Ex.

```
BEGINNING:           ;label for the program beginning
HOME I               ;other instructions, ex. “home” axis 1
RES O101             ;other instructions, ex. “reset” of an output
LOOP:
IF I101 = LO, LOOP   ;jump to the identifier "LOOP", if I101 is low
etc.
```

### *name EQU identifier*

Assignment of a symbolic name to a variable or flag, suitable for references with mnemonic names in the program code.

```
Ex.   QUOTA1 EQU V100
      VEL1   EQU V22
      F_ARR  EQU F1112
      TAGLIO EQU O101
```

It is necessary to remember that **there is no difference between capital and small letters**; so the here-under words are equivalent:

```
Ex.   VEL1   EQU V22
      Vel1   EQU V22
      VEL1   equ v22
```

## 5.6 General instructions.

### 5.6.1 ADC – Analog inputs reading

*Variable == ADC analog input*

Reading of an analog input and storing in the indicated variable; the meaning and the format of each input are shown in the below table

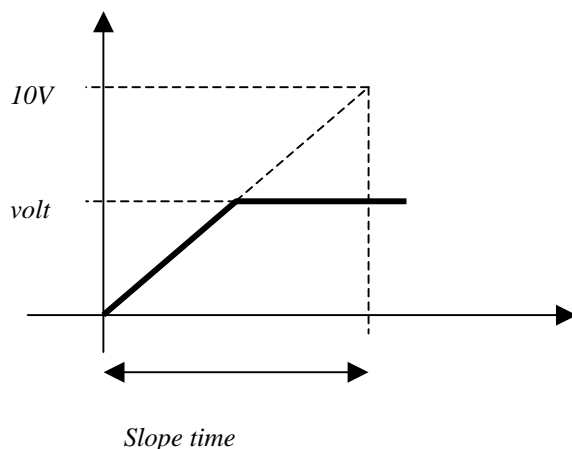
<b>ADC 1</b>	Input value for load cell #1	$\pm 10000.000$ if @Par[02] = 1 $\pm 20000.000$ if @Par[02] = 2
<b>ADC 2</b>	Input value for load cell #2	$\pm 10000.000$ if @Par[03] = 1 $\pm 20000.000$ if @Par[03] = 2
<b>ADC 3</b>	Analog input for differential #1	$\pm 10.000$
<b>ADC 4</b>	Analog input for differential #2	$\pm 10.000$
<b>ADC 5</b>	Analog input for differential #3	$0 \div 10.000$
<b>ADC 6</b>	Analog input for differential #4	$0 \div 10.000$

Ex.  $V001 == \text{ADC } 2$  ; the tension read in the input 2 is written in V001

### 5.6.2 DAC – Analog outputs writing

**DAC analog output, volt (-10V ÷ +10V) {,slope time (0.010 ÷ 10.000s) }**

Setting of an analog output (with an optional slope) through an immediate datum or a variable. The optional parameter generates a slope measured in seconds, with a resolution in milliseconds, corresponding to the variation of the range from 0 to 10 Volt.



**Picture 1 – Definition of “slope time”**

Due to the fact that some analog outputs can be used also for diagnostic purposes, it is necessary to disable this mode, acting on the corresponding general parameter.

## ***5.7 Instruction for error management***

### 5.7.1 ERROR – Program error code reading

*Variable* == **ERROR**

Program error code reading and recording of the mentioned variable

### 5.7.2 WARNING – Program warning code reading

*Variable* == **WARNING**

Program error code reading and recording of the mentioned variable

## 5.8 Generic regulation instructions

### 5.8.1 PID – Regulation from input for load cell

#### **PID** load cell input, analog output, data table

This function implements a PID regulator to be specifically used for the tension regulation on material (ex. winding-unwinding machines), with reaction coming from an input for load cell and command in analog output ( $\pm 10 V_{DC}$ ) towards an actuator, which can be a brake controlled by a proportional valve or a motor torque piloted.

In the instruction syntax it is specified the number of the input for the load cell (1÷2), the number of the output analog channel (1÷2) and the first of the variables V including the table with the instruction data; the definition of the table is here-under indicated.

- REFPID Reference of the tension in regulation
- SGLOFF Reference of the tension to be reached before switching off the regulator
- DELTAR Maximal change of the reference / second which is brought to the regulator by input
- OFFREF Reference offset (to avoid movement inversions on bi-directional actuators)
- KMIS Scale factor for the measure reading
- OFFMIS Offset of the measure reading (calibration)
- KFF Gain for the regulation feed-forward
- KERR Scale factor for the regulation error
- MAXERR Maximal input error to the PID filter (in absolute value)
- MAXPID Maximal output regulation value from PID filter
- MINPID Minimal output regulation value from PID filter
- KREG Scale factor for the regulator output
- KOUT Scale factor for the analog output signal
- MAXDAC Maximal value (in volt) for the analog output
- MINDAC Minimal value (in volt) for the analog output
- OFFDAC Value (in volt) of the analog output in condition of regulator switched off
- SWCVEL Source of the speed measure
- KVEL Scale factor of the speed measure
- DG\_MIS Diagnostic value of the tension measurement

- DG\_REF Diagnostic value of the tension reference of the regulator input
- DE\_ERR Diagnostic value of the tension error after the clamp
- DG\_PID Diagnostic value of the PID regulator output
- DG\_OUT Diagnostic value of the regulation output
- DG\_VEL Diagnostic value of the speed measurement
- DG\_DAC Diagnostic value of the tension on the analog output towards the actuator

The gains of the PID regulation filter are not defined in the table because they have to be set in the parameters of the axis corresponding to the analog output used for the regulation (see example).

To operate on an active PID instruction or screen its state, it is possible to use the here-under system flags:

<b>F271</b>	<b>F303</b>	R/W	<b>Start PID</b>	Brought to HI level, it activates the PID regulation on the analog output.
<b>F272</b>	<b>F304</b>	R/W	<b>Stop PID</b>	Brought to HI level, it causes the abort of the PID instruction
<b>F273</b>	<b>F305</b>	R	<b>Reset PID</b>	At HI level, it indicates the condition of the regulator in reset (Start PID at LO level)
<b>F274</b>	<b>F306</b>	R	<b>Exec PID</b>	At HI level it indicates the presence of a PID instruction in execution

In order to correctly use the PID instruction it is necessary to consider:

- The axis control mode, whose analog output is used for the regulation, must be of type 3.
- The PID instruction is preliminary, i.e. it must be recalled only one time at the beginning of the program; then, it will be possible to operate on it through the system variables and flags.
- All variables are continuously read by the instruction to constantly update the internal data.
- The different scale factors present in the table have been defined in order to allow the use of known measurements during the installation phase.
- The diagnostics variables are updated during each period of the regulation; they can be used to value the behaviour of the intermediate elaborating measurements during the regulation phase.
- If the instruction is used to realize winding-unwinding machines with a driver acting as actuator, the driver itself must be set to receive a torque reference by the ARP unit.

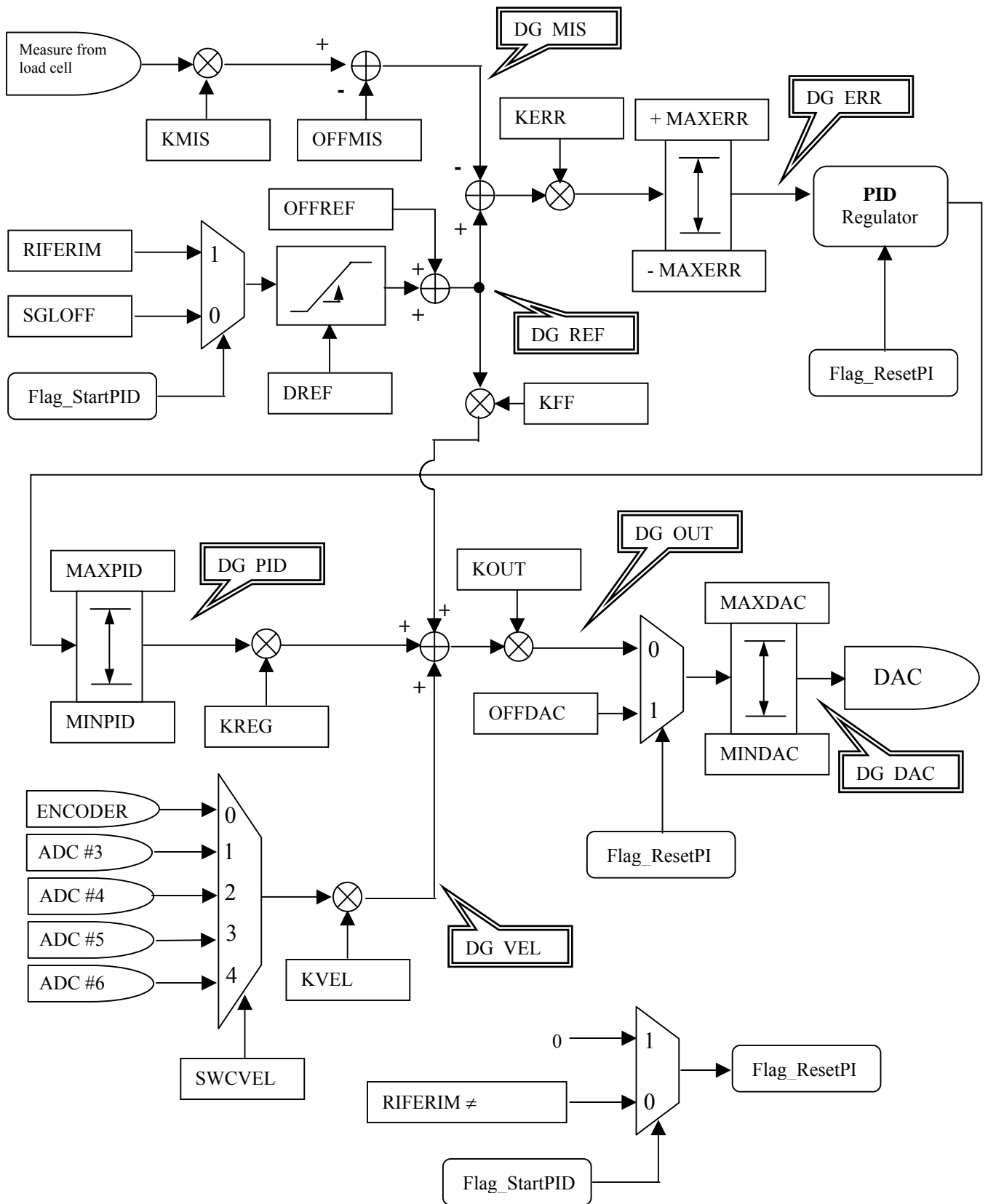


Figure 2 – PID instruction diagram

## 5.9 Axes movement instructions

### 5.9.1 Instruction of speed and acceleration setting

**VEL** axis, value per cent (0÷100%)

When the system is switched on or after a resetting, the maximal speed for each axis is set with the value included in the parameter of the maximal speed of the axis.

The “VEL” instruction ulteriorly limits the maximal speed for the axis indicated with a value per cent included in the above mentioned parameter; the “value per cent” can be an immediate datum or the identifier of a variable containing the willing value.

The maximal speed set with “VEL” is valid until a new instruction of the same kind, or until the switching off and the following switching on of the system.

**ACC** axis, value per cent (0÷100%)

**DEC** axis, value per cent (0÷100%)

Analogously to what happens with the speed, when the system is switched on or after a reset, the maximal acceleration and deceleration of each axis are set with the value included in the parameter of maximal acceleration of the axis.

The “ACC” and “DEC” instructions ulteriorly limit the maximal acceleration and deceleration of the axis at the value per cent contained in the above mentioned parameter; the “value per cent” can be an immediate datum or the identifier of a variable including the willing value.

These limits are valid until new instructions of the same kind or until the switching off and the following switching on of the system.

## 5.9.2 Instruction of axis initialisation

### 5.9.2.1 HOME – Cycle of value zeroing for incremental encoder

#### **HOME** *axis*

During the execution of the Home cycle the flag *Home in process* is enable; at the end of the cycle itself the flag *Home executed* in enable.

**BE CAREFUL** : the function has not been included in the 3.00 version, yet

### 5.9.3 Instruction of independent movement

#### 5.9.3.1 PAC – Absolute positioning

##### **PAC axis, absolute value (Unit)**

This instruction executes a positioning of the axis to the specified “absolute value”, without stopping the User’s program during the execution; the “absolute value” (Unit) can be directly supplied with a numerical datum or through a variable including the willing value.

The instruction ends with the axis stopped at the required position; at the end of the stopping, the flag of *arrived axis* is enable.

The movement can be aborted through the “STOP” instruction or enabling the *stop axis* flag; in this last case, the axis remains at its final speed and the *axis in final speed* flag is enable, and the axis will be conveniently managed by the User’s program.

It is possible to execute this instruction both on real axes and on simulated axes; therefore, it is necessary that the correspondent control parameter is correctly set, otherwise the instruction will not be executed.

It is also important to consider that if a “PAC” instruction is run while another positioning instruction is in process, the User’s program will stop on the line which includes the “PAC” instruction. Therefore, it is better to test the *arrived axis* flag or the *axis in final speed* flag before running a new movement instruction.

The movement profiler is calculated according to what is set in the parameters of maximal acceleration and speed, and to the limits introduced by the instructions “VEL”, “ACC”, “DEC”.

#### 5.9.3.2 PRC – Relative positioning

##### **PRC axis, relative value (Unit)**

This instruction is analogous to the previous one, but for the destination value which is indicated starting from the present value (that can also be negative).

### 5.9.3.3 PVC – Positioning in speed

#### **PVC axis, final speed (0÷100%)**

This instruction brings the axis to the speed indicated by the value per cent in the parameter; as soon as this speed is reached, the instruction ends, enabling the *axis in final speed* flag, and keeping the axis in movement. The User's program will act if it is necessary to stop the axis.

The movement can be aborted through the "STOP" instruction or enabling the *stop axis* flag; in this last case, the axis final speed is the present speed and the *axis in final speed* flag will be enable. The axis will be conveniently managed by the User's program.

It is possible to execute this instruction both on real axes and on simulated axes; therefore, the correspondent control parameter must be correctly set, otherwise the instruction will not be executed.

Also in this case, it is important to consider that if a "PVC" instruction is run while another positioning instruction is in process, the User's program will stop on the line which includes the "PVC" instruction. Therefore, it is better to test the *arrived axis* flag or the *axis in final speed* flag before running a new movement instruction.

The movement profiler is calculated according to what is set in the parameters of maximal acceleration and speed, and to the limits introduced by the instructions "VEL", "ACC", "DEC".

### 5.9.3.4 STOP – Stop axis instruction

#### **STOP axis**

It commands the immediate stop of a moving axis for each previous instruction, or of an axis in final speed through a motion instruction with the continuation of the program. The "STOP" instruction acts also on the synchronism instruction "PST", with the stop axis.

The deceleration occurs according to what is set through the parameter of maximal acceleration and the eventual "DEC" instruction; after the stopping, the axis is in still condition, under control.

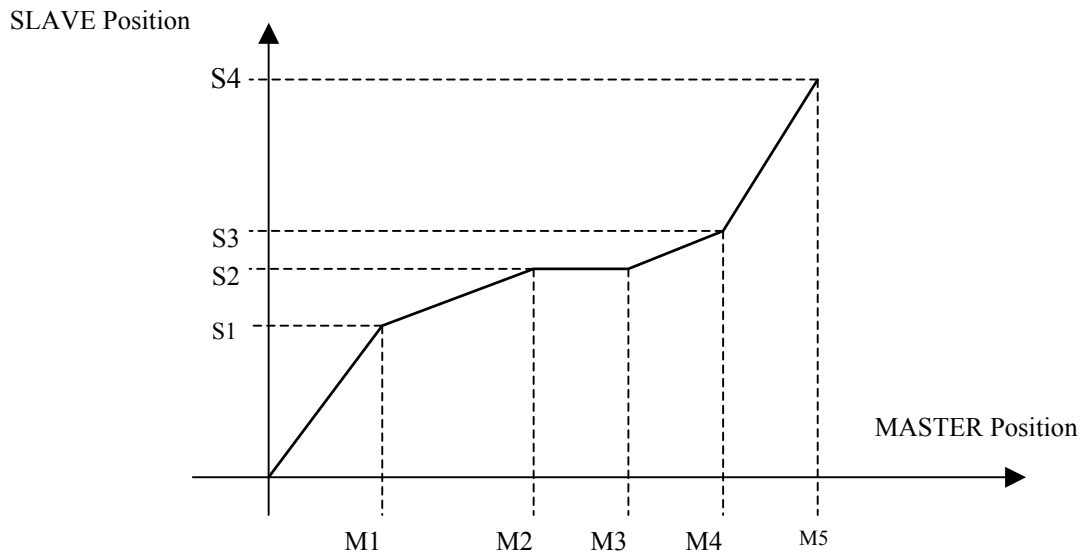
At the end of the stop, the *arrived axis* flag is enable.

## 5.9.4 Instruction of synchronized movement

### 5.9.4.1 PST – Tabulated interpolator electronic cam

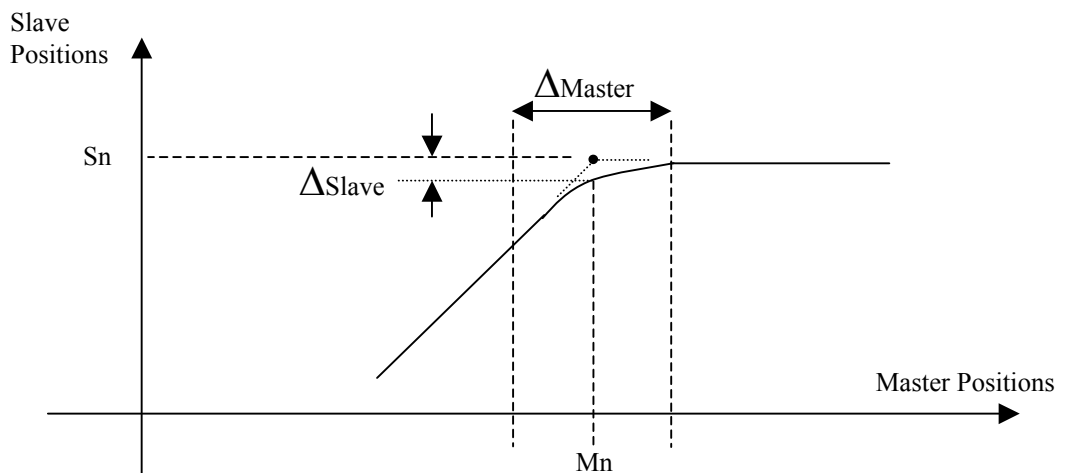
**PST** *master axis, slave axis, variable[n]*

This function forces a slave axis (necessarily axis 1) to move in a synchronic way with a master axis (axis 2), according to a report defined by a point table, where “*variable[n]*” is the first element. The above mentioned table must specify the pairs of values composed by the position of the master axis and the slave axis.



**Figure 3 – Definition of the points for an electronic cam**

As shown in the previous example, a broken line is traced on a plane where the co-ordinates are the positions of the two axes. As this curve can't be physically realized, the segments of the above mentioned broken line are self-calculated and joined through parabolic hyphens, passing by the interpolator points defined in the table.



**Figure 4 – Interpolation on electronic cam**

The hyphen amplitude “ $\Delta_{Master}$ ” is calculated considering the maximal speed necessary to follow the Master axis and the characteristics of maximal speed and acceleration of the Slave axis; its value can be estimated according the here-under relation :

$$\Delta_{Master} = \left\{ \frac{|\Delta m| \cdot VelM}{AccS} \right\} \cdot VelM$$

Where:

$\Delta m$  = Slope variation in [slave unit / master unit], indicated as the difference between the angular coefficients of the two hyphens of the broken line

$$m_i = \frac{S_{i+1} - S_i}{M_{i+1} - M_i}$$

VelM = Maximal speed of the Master axis ( @ParS[24] ) in [master unit / second].

AccS = Maximal acceleration of the Slave axis ( @ParS[08] ) in [slave unit / second<sup>2</sup>], eventually limited by the instructions “ACC”, “DEC”.

### COMPULSORY SETTINGS:

(@ParM indicates the parameters correspondent to the master axis, and @ParS indicates the parameters correspondent to the slave axis)

@ParS[04] = Controlled Axis in control mode

@ParS[24] = Maximal speed of the Master axis used to calculate the interpolarity

@ParS[25] = Speed of the cam input with the Master axis in still position

Variable[n] = Number “m” of points pairs of the cam

Variable[n+1] = first value of the master axis

Variable[n+2] = first value of the slave axis

Variable[n+3] = second value of the master axis

Variable[n+4] = second value of the slave axis.

...

Variable[n+2m-1] = last value of the master axis

Variable[n+2m] = last value of the slave axis

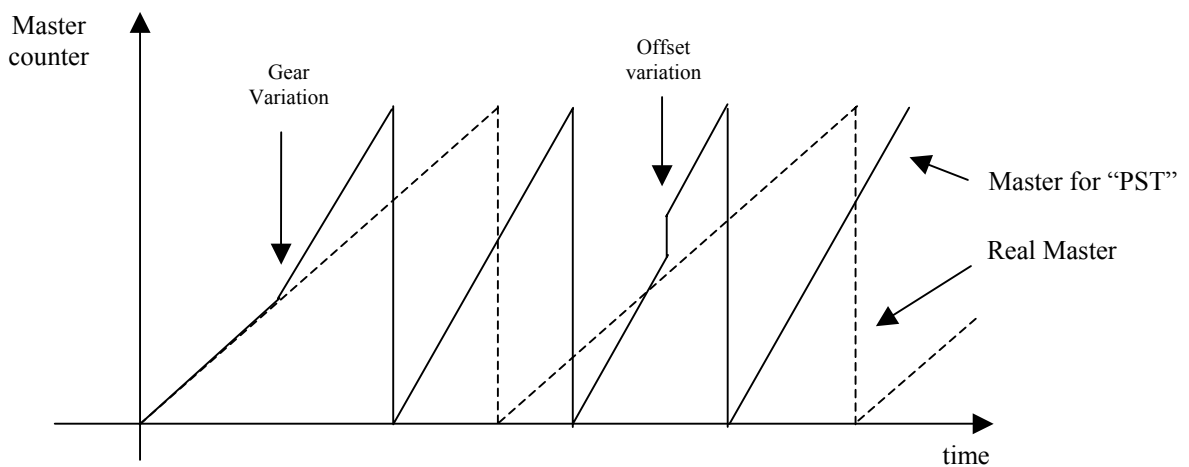
**OPTIONAL SETTINGS :**

@ParS[21] = Gear in reading mode for the Master axis

@ParS[22] = Offset in reading mode for the Master axis (eventually applied in a progressive way through @ParS[23] ).

Through these parameters it is possible to change the vision of the master axis inside the instruction in comparison with the real master; the position and the speed read of the master can be modified without changing the real counter.

It is important to notice that, in this way, it is possible to loose the correspondence between the Master real cycles and the virtual ones.



**Figure 5 – Offset and Gear in reading mode on the Master for “PST”**

@ParS[19] = Output offset for the Slave axis (eventually applied in a progressive way through @ParS[20] ).

This parameter changes the output references generated by the Master – Slave table; its task is to relocate the output references of the Slave axis.

By using the PST instruction it is important to consider that :

1. The table of the variables for the “PST” instruction must be compiled to have the Master positions monotonic strictly growing from the beginning to the end, without points in the same position; on the contrary, as regards the Slave, it is possible to accept points with the same position. This means that for particular movements of the Master, the Slave must be in still condition.
2. If the Master position is not included between the minimal and maximal values of the table when the “PST” instruction starts, the instruction will be aborted without movement of the Slave.

3. If at the beginning of a “PST” instruction, *the Master axis is stopped*, the Slave axis is automatically brought on the cam curve, with a speed specified by the parameter @ParX[25]; when the curve is reached, the synchronism flag is enable and the Master axis can start moving without sudden variations on the Slave axis.
4. If at the beginning of a “PST” instruction, *the Master axis is moving*, the locking between the Master and Slave axes must be conveniently managed by the User’s program in order to avoid sudden unwilling variations.
5. The specified position must be included within the limit of the correspondent positioning cycles of the two axis, otherwise it could be possible that the Slave axis is not managed in the correct way.
6. If there are wrong positions in the table of the cam description, the Slave axis can be abandoned without control and in final speed mode; in this can it has to be correctly managed by other instructions of the User’s program.
7. If the Master exits from the positions of the table, the Slave axis is left in final speed and so it must be managed by other positioning instructions (ex. STOP or PAC...).
8. The instruction can distinguish between the repetitive cam for rotating axes with cyclic and linear values and the non-repetitive cam:
  - **Repetitive cam for cyclic rotating axes:** the first and the last point in the table, both for the Master axis and for the Slave one, must coincide with the correspondent limits of the positioning counters (ParX[05] and ParX[06]). In this way, the last point in the table coincides with the first point, with the correspondent interolarity between the first and the last hyphens of the broken line.
  - **Repetitive cam for linear axes:** the first and the last points of the table for the Master axis must coincide with the limit of the positioning counter (ParM[05] and ParM[06]), while for the Slave axis the first point has to coincide with the last one. Even in this case, the last point of the table coincides with the first point, with the correspondent interolarity between the first and the last hyphens of the broken line.
  - **Non-repetitive cam:** if the conditions indicated in the previous points have not occurred, the electronic cam will be executed until the Master positions remain inside the table and the extreme points of the table are jointed together; when the Master position is outside the table, the cam function ends, the Slave axis is abandoned and it must be managed by other instructions controlling the motion. For example, a slide executing a movement alternatively forwards and backwards, can control a Slave axis in electronic cam in the forwards sense, while it cannot control it in the backwards sense when it has exceeded a position of cam end.

The “PST” instruction can be aborted through the “STOP” instruction or enabling the *stop axis* flag: in the first case, the axis is stopped with the set deceleration slope, while in the second case the axis goes on with the final speed and will be conveniently managed by the User’s program with the following movement instructions.

Considering the example of the previous Figure, the descriptive table of the cam will be:

variable[n]	= 6		
variable[n+1]	= 0	variable[n+2]	= 0
variable[n+3]	= M1	variable[n+4]	= S1
variable[n+5]	= M2	variable[n+6]	= S2
variable[n+7]	= M3	variable[n+8]	= S2
variable[n+9]	= M4	variable[n+10]	= S3
variable[n+11]	= M5	variable[n+12]	= S4

When the “PST” instruction is run, if the standard requisitions have not been respected, the system generates an error “**E13**” – “**INSTRUCTION PARAMETERS ERROR**” on the *Slave axis*. The causes for this above mentioned error could be :

- Slave Axis  $\neq$  1 or Master Axis  $\neq$  2
- $@ParS[24] \leq 0$
- Number of points pairs inferior to 2 or superior to 22
- The points correspondent to the master axis are not monotonic strictly growing.
- The Slave axis should exceed its maximal speed in order to follow the Master axis at the maximal speed, according to what has been defined in the table.
- The acceleration or deceleration of the Slave axis are not enough to execute the cam at the maximal speed of the Master, as it is required a  $\Delta$ Master interpolarity with a too high amplitude to be executed. If this error occurs, it is necessary to limit the interpolarity amplitude acting, if possible, in the following way:
  - Reduce the maximal speed to pursue the Master, through the parameter  $@ParS[24]$ .
  - Increase, if possible, the maximal acceleration of the Slave axis, through the “ACC” and “DEC” instructions or the parameter  $@ParX[08]$

## 5.10 Timer management instructions

### **WAIT** *time* (0.001 ÷ 100.000s)

It interrupts the program running for “*time*” seconds; during the execution of the “WAIT”, the axes continue their movement and are in control condition, but no program instructions is executed until the ending of the assigned time.

### **STIM** *n*

Six timers are available in the system which can be contemporarily used. With the “STIM” instruction, it is possible to activate a timer by zeroing its value and starting the count. This instruction must be executed before reading the value of the timer with the “TIM” instruction described in the following paragraphs.

### **STIM** *n*, *time* (0.01 ÷ 100.0 s), *flag* (F1 ÷ F1000)

All six timers can be used in the extended mode of the “STIM” instruction : in this “*flag*” form is zeroed during the phase of the beginning of the count and activated after a time equal to “*time*”

Ex. STIM 1, 5, F1 ;after 5 seconds from this instruction the F1 flag becomes active, remaining at high level until a following instruction will modify it.

### *variable* == **TIM** *n*

It reads the value of the timer “*n*”, writing what is read in “*variable*”; before effecting this reading, it is necessary to zero the timer with the STIM instruction.

Ex.  
STIM 2 ;it zeroes the timer 2  
*LINEA*:  
VI == TIM 2 ;it reads the value of the timer 2  
IF VI < 1, *LINEA* ;it waits for a second

## 5.11 Instructions for flag and outputs management

**SET** *Flag / Output*

**RES** *Flag / Output*

Enabling at high level (“SET”) or disabling at low level (“RESET”) of the flag or of the output indicated in the instruction.

**CAM $n$**  *axis, position of set( $U$ ), position of reset( $U$ ), Output*

Enabling of a “**digital electric cam**” between an axis and an output; it is possible to execute up to 6 cams contemporarily ( $n = 1 \div 6$ ).

It is necessary to consider that :

- The number of cam “ $n$ ” and the axis controlled by the instruction must be constant
- The running of the “CAM $n$ ” instruction, without other parameters, disable a resident and active cam.
- The flag or the output are updated with a recurrence based on the time of the control loop scanning.
- “CAM $n$ ” is a preliminary instruction; it is therefore enough to execute it once in order to program permanently the enabling and disabling of the willing output.
- If it is necessary to change one of the instruction parameter, the instruction must be re-run.
- It is possible to program a cam with a denied output, by writing the prefix “!” in the identifier of the output itself.
- Outside the specified values, the output is always disable.
- The output is enable when the axis exceed the “set” value towards the “reset” value.

In the following example the six electric cams are programmed in order to obtain the behaviour described in the here-under figure :

Ex.:     **cam1 1, X2, X3, O101**  
          **cam2 1, X3, X4, O102**  
          **cam3 1, X3, X2, O103**  
          **cam4 1, X2, X1, O104**  
          **cam5 1, X2, X3, !O105**  
          **cam6 1, X3, X2, !O106**

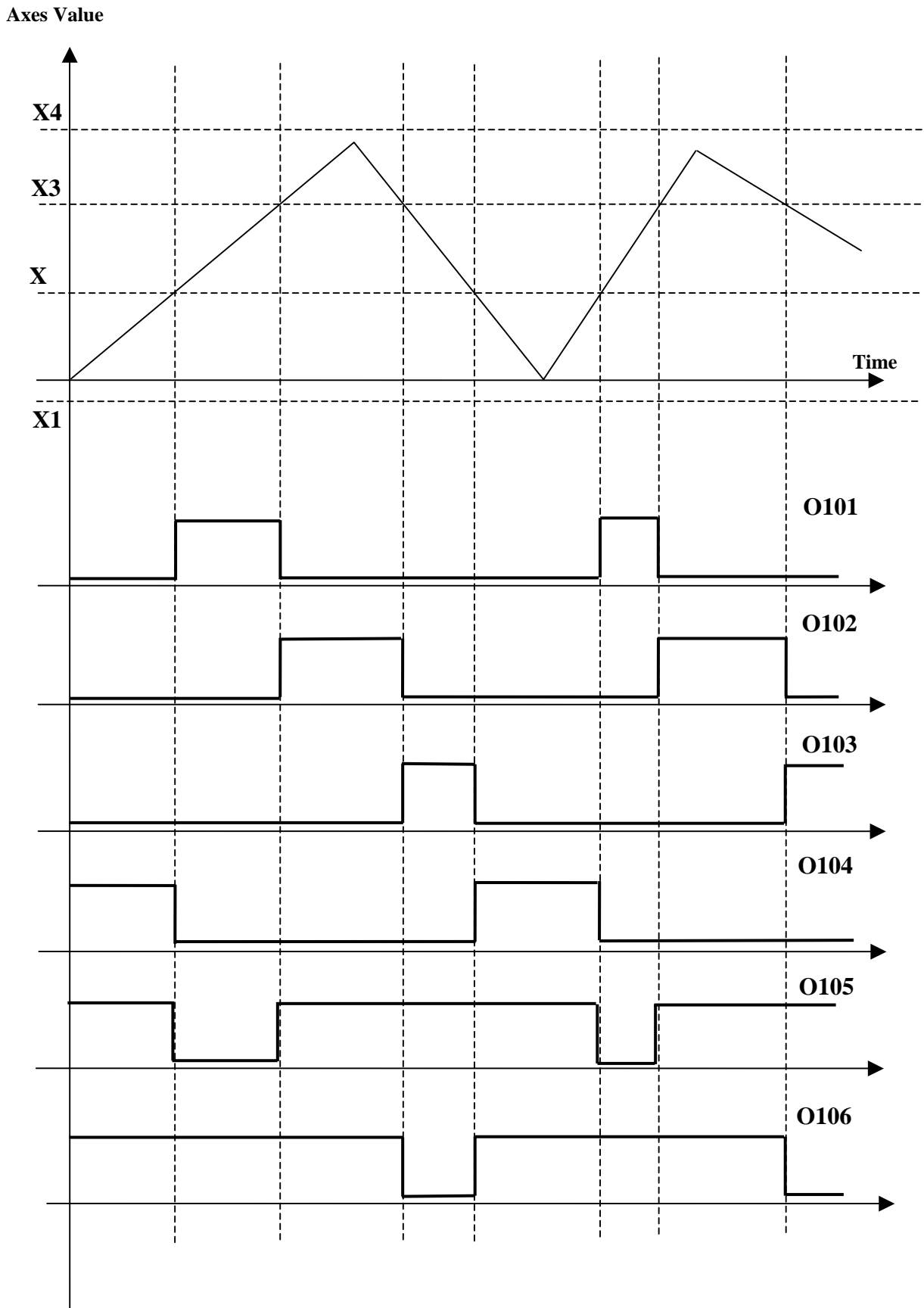


Figure 6 – Example of electric cam use

## ***5.12 Instruction for the management of the TR4 and TR5 operator terminal***

The ARP card can manage a ASA-RT operator terminal, with interface on a Can Bus link; the management of the above mentioned terminal is regulated through some general parameters and some specific functions, and it is realised in conformity with the standard CANOpen field bus.

### **MSG *nr\_msg, "character string"***

It sends an alphanumerical message to the terminal; this message shall be displayed through the TERM instruction. The message, identified with an index (1÷500), is composed by a 16 characters string, which can include a displaying format specification if it must be used to visualize a numerical datum. The format specification is composed by a sequence of “#” with an eventual decimal point ‘.’; the maximal numerical format is #####.###.

All messages sent to the terminal are saved in the eeprom permanent memory and it is possible to display them without writing. It is advisable to generate and execute once a User’s program reserved to the messages writing and then load the working program.

### **TERM {*line , column*}, {*height, number\_msg*}, {*datum*}, {*mode*}**

The “TERM” instruction allows to display or delete a message on the terminal.

The size of the displayed characters can have the following values:

8 x 8 pixels      ( height = 1 )  
16 x 16 pixels    ( height = 2 )  
32 x 32 pixels    ( height = 3 )

The position of the message on the screen, defined with line and column, is always referring to a 8x8 pixels screen with 8 lines x 16 columns, even when a message with a height bigger than 1 is written.

“Line”, “Column”, and “Height” must be expressed as constants; the “message number”, from 1 to 200, can be expressed both as a constant and as a contents in a variable V (index management useful for multilingual screen pages. The text of the message is not displayed here as it should be previously downloaded in the permanent memory of the terminal through the “MSG” instruction.

The “datum” to be displayed must be a variable V, while the “mode”, if indicated, enables the change of the value by the terminal in standard mode (‘E’) or password (‘P’). The displaying format has to be indicated in the string, by using the sequence “#####.###”.

The “TERM” instruction could be used in the following cases :

- “TERM”  
it deletes the whole screen page of TR4 terminal
- “TERM line, column”  
it deletes a message previously displayed in the specified line and column
- “TERM line, column, height, number\_msg”  
it displays the message number “number\_msg”, with a “height” size, in the specified position.
- “TERM line, column, height, number\_msg, datum”  
it displays the message number “number\_msg”, with “height” size, in the specified position, visualizing the value indicated in the “datum” field, instead of ### included in the string.
- “TERM line, column, height, number\_msg, datum, mode”  
it displays the message number “number\_msg”, with “height” size, in the specified position, visualizing the value indicated in the “datum” field, instead of ### included in the string. The displayed value can be changed by the operator, in the specified mode, enabling the mode by the terminal.

**Example:**

```

TERM                                ;it deletes the screen page
MSG 35, “AUTOMATIC CYCLE”
TERM 4, 1, 1, 35                    ;line=4, col.=1, height=1, message=35 (“AUTOMATIC CYCLE”)
MSG 2, “LENGTH: = #####.#”
TERM 2, 22, 3, 2, V100              ;line=2, col.=22, height=3, message = 2, it displays V100 in the format 9999,9
TERM 5, 2, 2, 2, V100, E           ;line=5, col.=2, height=2, it displays V100 with the possibility to change the
                                   value in standard mode
TERM 6, 4, 2, V4                    ;displaying of the index message
TERM 6, 8, 3, 24, V1001, P         ;password setting
TERM 6, 8                           ;it deletes the message in position line=6, col.=8

```

## **PRINT** *number\_msg, {datum}*

PRINT instruction sends to the serial port of the operator terminal a specified message to be printed.

The mandatory parameter “*number\_msg*” is referred to a message which has been previously downloaded on the terminal through the MSG instruction; it can be expressed both as a constant and as a variable. The “*datum*” to be displayed, must be a variable V and its displaying format has to be indicated on the string to be printed similarly to the TERM instruction.

If “*numero\_msg*” is equal to zero, the terminal will send to a two-line print the actual date and hour; although, all messages are printed one for each line.

Examples:

MSG 35, “AUTOMATIC CYCLE”

PRINT 35 ;it prints the message “AUTOMATIC CYCLE”

MSG 2, “LENGTH: = #####.#”

TERM 2, V100 ;it prints V100 in 9999,9 format (“LENGTH: = 9999.9”).

## 5.13 Programming examples

### 5.13.1 Example for the use of the preliminary instructions

```
beg EXEMPLE1

@par1[4] == 0
@par2[4] == 0

cam1 1, 5000, 15000, O101
cam2 1, 15000, 5000, O102
V10 == 5000

on3 F2 & (1 > V10) INTER
on6 F3 & /F4    INTER2

LOOP:
  V1 == @POS1

  goto LOOP
end

bega INTER
res F2
set O103
enda

bega INTER2
res F3
set O104
enda
```

The above mentioned program allows to point out the main characteristics of the ARP unit programming.

It is composed by an initializing process, followed by an infinite cycle identified by the LOOP label; both axes are set in a single reading by zeroing the control parameter.

In the first phase, two electrical cams concerning the controlled axis (#1) are initialized; the first cam enables the output when the axis is moving towards positive value, while the second one enables the output when the axis is moving towards the negative values.

Then, two “on-events” are initialized : the first one detects the exceeding of a value included in the variable V10, while the second one detects a positive edge on the flag #4; these two “on-events” occur only if they are enabled by flag #2 and #3.

In the infinite cycle, the system variable @pos1, which includes the axis 1 reference position, is read and copied in the temporary variable V1.

### 5.13.2 Example of PST instruction on simulated Master

```
beg EXAMPLE2

@par2[4] == 1      ; simulated Master axis
@par1[4] == 2      ; controlled Slave axis

wait 1
V[10] == 3
V[11] == 0
V[12] == 0
V[13] == 5000
V[14] == 5000
V[15] == 10000
V[16] == 0

pst 2,1,V10
WSYNC:
if F261=lo, WSYNC

LOOP:
if F292=lo, LOOP
wait 1
pac 2, 10000
W2:
if F292=lo, W2
wait 1
pac 2, 0

goto LOOP

end
```

This program is an example of synchronization of the controlled axis with a Master axis managed in simulated mode.

In the preliminary phase a table with the cam data is initialized and the “PST” instruction is run, waiting for the synchronization. As in this situation the Master is in still position when the “PST” instruction is run, a profiler is calculated and generated. This profiler will bring the controlled axis on the point of the cam which corresponds to the position of the Master; when the cam entering is ended, the synchronism flag is enabled.

When the preliminary phase of the programming cycle is ended, some forwards-backwards positions of the simulated Master are executed; the controlled axis will consequently move according to the specified table.

It is important to point out that the table is described only for broken lines; the polarity between the broken hyphens will be automatically calculated by the instruction itself.

### 5.13.3 Example of use of the PID instruction

```
REFPID      equ    V50
SGLOFF      equ    V51
DELTAR      equ    V52
OFFREF      equ    V53
KMIS        equ    V54
OFFMIS      equ    V55
KFF         equ    V56
KERR        equ    V57
MAXERR      equ    V58
MAXPID      equ    V59
MINPID      equ    V60
KREG        equ    V61
KOUT        equ    V62
MAXDAC      equ    V63
MINDAC      equ    V64
OFFDAC      equ    V65
SWCVEL      equ    V66
KVEL        equ    V67
DG_MIS      equ    V68
DG_REF      equ    V69
DG_ERR      equ    V70
DG_PID      equ    V71
DG_OUT      equ    V72
DG_VEL      equ    V73
DG_DAC      equ    V74

F_START     equ    F303
F_STOP      equ    F304
F_RESET     equ    F305
F_EXEC      equ    F306

CONTROL     equ    @par2[04]
KP          equ    @par2[11]
KI          equ    @par2[12]
KD          equ    @par2[13]

CELL1       equ    V10
APPO        equ    V11

    beg TEST_PID

    CONTROL == 3

    PID 1,2,REFPID
```



---

LOOP:

CELL1 == adc 1

APPO == (DG\_MIS /50)

dac 1, APPO

goto LOOP

end

In this program the symbolic names of the sizes in the table, of the parameters and system flags are defined.

Then, the PID instruction is enable and the system enter a cycle where diagnostic sizes are updated.



**ASA-RT srl**

Strada del Lionetto, 16/a – 10146 Torino – ITALY

Tel: +39 011 796 333r.a.. – Fax: +39 011 712 339

E\_mail: [info@asa-rt.com](mailto:info@asa-rt.com) – Url: [www.asa-rt.com](http://www.asa-rt.com)