

Solutions in *motion*

IEC programming for nodes

Intelligence Production Movement

IEC programming for nodes

Doc. MS270928
Ed. 13 - English - 28 Oct 2020



IMPORTANT

CMZ SISTEMI ELETTRONICI S.r.l. reserves the right to make changes to the products described in this document at any time without notice.

This document has been prepared by CMZ SISTEMI ELETTRONICI S.r.l. solely for use by its customers, guaranteeing that at the date of issue it is the most up-to-date documentation on the products.

Users use the document under their own responsibility and certain functions described in this document should be used with due caution to avoid danger for personnel and damage to the machines.

No other guarantee is therefore provided by CMZ SISTEMI ELETTRONICI S.r.l., in particular for any imperfections, incompleteness or operating difficulties.

This document contains confidential information that is proprietary to CMZ SISTEMI ELETTRONICI S.r.l.. Neither the document nor the information contained therein should be disclosed or reproduced in whole or in part, without express written consent of CMZ SISTEMI ELETTRONICI S.r.l..

Table of Contents

1. Purpose of the user guide	1
2. What's new	3
3. User Guide	5
3.1. Main characteristics	5
3.1.1. BD series: Program safety condition procedure	11
3.1.2. BD series: Request to go in the Stopped state	12
3.1.3. Running Stopped state notes	12
3.2. Declaration of the variables	12
3.2.1. How the variables are stored into the memory	12
3.2.2. Memory resources of the drive	15
3.2.3. GLOBAL variables	16
3.2.4. RETAIN variables	19
3.2.5. CONSTANT objects	20
3.2.6. LOCAL variables	21
3.3. Instructions	21
3.4. What is a function block?	23
4. IEC reference guide	27
4.1. System functions and function block(SYS_)	27
4.2. Axes management (MC_)	42
4.2.1. Axis status	42
4.2.2. Drive status	44
4.2.3. Axis functionalities	46
4.2.4. Data Type : AXIS_REF	47
4.2.5. Function blocks list	48
4.3. Peripherals management (IO_)	83
4.3.1. Encoder management	83
4.3.2. Digital inputs management	104
4.3.3. Digital outputs management	117
4.3.4. Analog input management	126

4.4. Utility library (Ut_)	135
4.4.1. Functions and function blocks list	135
4.5. Examples	145
4.5.1. Axis management	145
4.5.2. Capture example	149
4.5.3. Example of the management of a program safety condition request	151
A. Error codes list	155
B. Parameters table	157
C. How can I assess the memory usage?	159
D. Objects of the programming	161

Purpose of the user guide

The purpose of this document is to describe how to use the provided functions and function blocks in an IEC application program which runs into a drive.



Important

This manual does not teach how to write an IEC program, the standards and the syntax, therefore, first of all, it is important to study an IEC programmer guide.



Important

This manual refers to the programs that can be written in the drives:

- ISD ;
- SVM ;
- IBD (Hw \geq 15) ;

Chapter 2

What's new

Ed 5

- Description of all the `Io_` function blocks (see *Section 4.3, “Peripherals management (IO_)”*).
- Example for the capture function management (see *Section 4.5.2, “Capture example”*).

Ed 6

- New IBD system.

Ed 7,8

- Description of safety conditions of the program for IBD. There is also a little example.

Ed 9

- some corrections have been made in the `MC_Home`.

Ed 10

- New modes of homing inserted in the `MC_Home`.
- Informations about the resources of the drive updated (differences between ISD, SVM firmware less or greater 38).

Ed 11

- Contents related to the BD drives added.
- *Table A.3* added in the *Appendix A, Error codes list* (Appendix A).
- *CMP_REF* reference added for the *Io_EncComparator* position comparator.
- References used in the *Section 4.3, “Peripherals management (IO_)”* function blocks updated.

Ed 12

- *MC_Gear* function block contents updated.
- *Io_EncEventCaptureValue* two new capture source added.

- *Table A.3* wrong error codes deleted in the IO's function blocks error codes table.
- Various minor corrections.

Ed 13

- *Appendix D, Objects of the programming*: contents updated with the PROFINET addresses and with a note to introduce the "n" variable in the addressing.
- Various minor corrections.

Chapter 3

User Guide

The programming of the drives follows the IEC 61131-3 language standards and in the same way the functions and function blocks dedicated to the motion control respect the PLCopen standards.



Important

In the **Program** page of the SDSetup PC program is possible to write a program, build it, debug it, and download it into the drive.

3.1. Main characteristics

The main characteristics of the programming are:

- IEC language Structural Text (ST);
- type of variables: BOOL, SINT, USINT, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD;
- the STRING variable type is not managed;
- array and struct objects are available;
- some function and function block are provided;
- the application code runs into an unique task and it can be divided into some different programs:

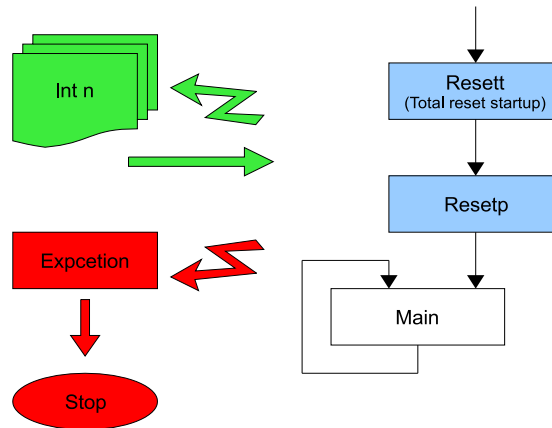


Figure 3.1. Application code structure

- **main**

this is the main program and it must always be present. At the beginning of each execution of the program the %I (input status) are updated with the value of the digital inputs, and at the end of the program the %Q (output status) are copied to the digital outputs;



Important

The measure of the "PROGRAM main" time execution can be read in the *plc period* variable (CAN index 0x4616.00, MODBUS 8773). The read value is the actual measure, not the maximum. The "PROGRAM main" is continuously executed.

- **exception**

when a fatal error happens during the execution of an instruction, the application code cannot run, therefore the *main* program execution is stopped. If the *exception* program is present in the application code, it is called before the stop of the program execution; in this way the programmer can insert in this program all the instructions, functions and functions blocks that are necessary to guarantee a safe stop of the application code execution. The *exception* program is usually executed only once. The function *SYS_Continue* permits to continue the calling of the program. At the end of the execution of this program the application is stopped. The function *SYS_Restart* permits to restart the application with a partial or total reset, or with a firmware restart. An example of fatal error is a division for zero. The object 8709 shows the reason of

the alarm (see *Appendix A, Error codes list*). It is possible to read it with the function *SYS_ReadObject*.

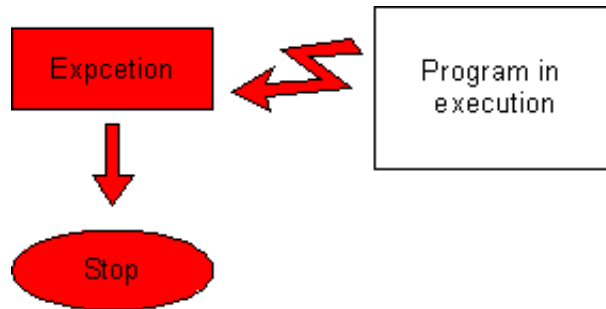


Figure 3.2. exception

- resett**

if the *resett* program is present in the application code, then it is called when a total reset startup is executed. It runs before the *resetp* program. The total reset is usually executed only at the first startup of the application code, then the application has a normal startup. In a normal startup the application skips the *resett* program and starts from the *resetp* program and then calls the main program.

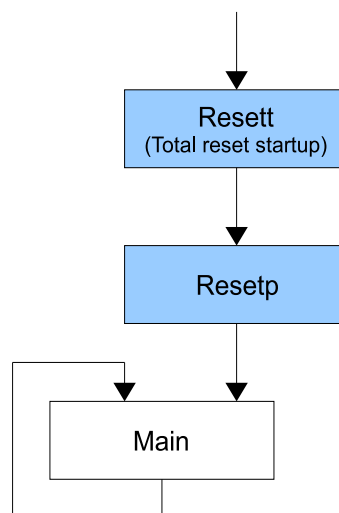


Figure 3.3. Startup



Warning

If the *Section 3.2.4, "RETAIN variables"* management is not active, then at every switch off/switch on of the drive a total reset startup is executed, therefore the *reset* program is always executed.



Note

While commissioning the application program, the code is usually modified and then built and downloaded into the drive. At the end of the download procedure the application program is launched. If the declaration of the *Section 3.2.4, "RETAIN variables"* is changed compared to the previous downloaded application, then it is executed a total reset startup, else it is executed a normal startup.

- **resetp**

if the *resetp* program is present in the application code, then it is executed at every startup.

- **Intn**

with (the second) $n = 0$ to 8, these are a group of programs which are launched when a particular event happens. An event is joined to a program *Intn* by the function *SYS_EnEventInt*.

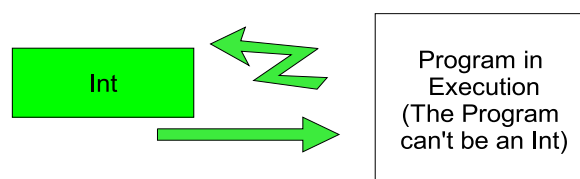


Figure 3.4. Intn program management



Important

When the *Intn* program is running, the drive executes it as fast as possible. The execution of the *Intn* program reduces the free resources of the system, therefore it is important to insert into these programs only the very essential instructions, functions and functions blocks.

The main characteristics of an *Intn* program are:

- an *Intn* program is more important than the other programs, therefore when it is launched it stops the execution of the programs that are running. The only exception is that an *Intn* program cannot break the execution of another *Intn* program.
- when the event happens there is a delay before the called *Intn* program is launched.



Note

When only **one** event happens, then the delay before the start of the execution of the called *Intn* program can be 400 μ s.



Note

The delay can increase when more events happen very close, because more *Intn* programs have to be executed at the same time. The first characteristic of the programming with SDSetup is that it has only one task, therefore it can execute only one program. It means that, while an *Intn* program is executing, the others stand waiting until it is finished.

- When more events happen very close, then more *Intn* programs have to be executed at the same time. The priority of execution is defined by the number *n* of the *Intn* program. The *Intn* program with the lowest *n* will be the first one launched.

While an *Intn* program is running, the other events may anyway happen. The *Intn* programs that is called by these events waits until the running program is executed. When the execution of the *Intn* program finishes, the next launched *Intn* program will be the *Intn* program which *n* value is the lowest. It is important to observe that the priority is not the time of the first happened event, but only the number *n* of the *Intn* program.

- when an *Intn* program ends, the program can wait before restart the execution of the application code.



Note

In the worst case the delay is 400 μ s.

- when an *Intn* program ends, and there is not any *Intn* program still on hold, then the application code execution returns to the program which was stopped, restarting from the instruction where it was been blocked;

- **run**

if the *run* program is present in the application code, then it is executed when the application execution switches from "Stopped" to "Running" status. The execution is started at the beginning of the *main* program. The status of the execution ("Stopped/Running") is commanded from a button in the SDSetup, in the Program management page.

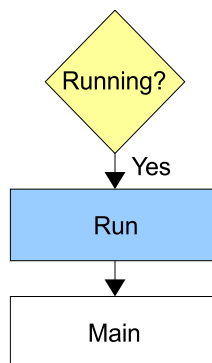


Figure 3.5. When the application returns running

- **stop**

if the *stop* program is present in the application code, then it is executed when the application execution switches from "Running" to "Stopped" status.

The execution is stopped at the end of the *main* program. The status of the execution ("Stopped/Running") is commanded from a button in the SDSetup, in the Program management page.

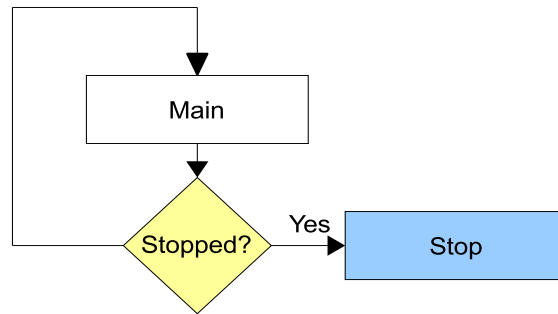


Figure 3.6. When the application is stopped



Warning

If the "PROGRAM main" is executing a loop that cannot be solved, then it will be impossible to stop the program because the program is always stopped at the end of the task.

3.1.1. BD series: Program safety condition procedure

In the BD series there is an internal management of the actions that are requested to the drive: the System Manager (see the product manual). It allows the drive to perform some actions only if all the safety conditions are respected. One of this safety condition is managed from the IEC program: in some application, before to stop the execution of the program, it is necessary to set the application in a well defined situation. This is the safety condition of the application.

The sequence to set the program execution in a safe condition has two steps:

1. the program has to manage the *IECSafeCondition* internal semaphore (see *SYS_WrIECSafeCondition*), which status has to be switched to "green" ;
2. after that, the program execution switches to "Stop". The *main* program arrives at the end of its execution and then the program *stop* is called, if present.

IECSafeCondition

IECSafeCondition is an internal variable. It is a "semaphore" that can block the execution of the action that has been requested by the System Manager.

For example: if an update of the IEC is requested through the SDSetup (download IEC), then System Manager requests to the program to go in its safety condition. Now, if the semaphore is yellow the action is blocked until it is set to "green".

The default value of *IECSafeCondition* is "green". In the program it is possible to change the value of this variable through the *SYS_WrIECSafeCondition* function. See [Section 4.5.3, "Example of the management of a program safety condition request"](#).

3.1.2. BD series: Request to go in the Stopped state

The request to switch to the "Stopped" state is an action controlled by the System Manager, so it needs that the IEC safety condition is respected.

3.1.3. Running Stopped state notes

For SD series the Running/Stopped state management is very simple:

- at the start-up the program is always in "Running" state;
- when the drive is in "Stopped" state and the program is downloaded, then the state returns to "Running". At the end of the download, the IEC program is in Running state.

For BD series the management is quite different:

- at the start-up the program is always in "Running" state;
- When the drive executes an action that needs to put the program in the safety condition, then at the end of this action the IEC state will return to the same status it was before the action.

3.2. Declaration of the variables

This paragraph describes the declaration of the GLOBAL, RETAIN and LOCAL variables. The first sections are important because they explain how the variables are stored in the memory and define the memory limits of the drives.

3.2.1. How the variables are stored into the memory

The memory used by the application program depends on the declaration order of the variables. The declaration order, which guarantees to minimize the usage of memory, is:

1. STRUCT and instances of FUNCTION BLOCK;
2. DINT, UDINT, DWORD;
3. INT, UINT, WORD,
4. SINT, USINT, BYTE,
5. BOOL

Here follow the reasons of the sequence.



Important

The CPU of the drives does not manage data type smaller than a WORD (2 bytes). Bit and byte data type are not managed.

The internal management of the CPU, used in the drive, causes a different storing of the variables in the memory:

- DINT, UDINT and DWORD : these variables always need 4 bytes (2 WORDS) and are memorized at addresses that are multiple of 4 bytes;
- INT, UINT and WORD : these variables always need 2 bytes (1 WORD) and are memorized at even addresses (multiple of 2 bytes);
- SINT, USINT and BYTE : these variables need 1 byte;
- BOOL : these variables need only 1 bit. It is impossible to use only a bit of the memory, the minimum size of usable memory is 1 byte. Therefore 1 BOOL needs 1 byte, but if in the declaration of the variables the BOOL are near than they are packed into the same byte at different bits.

```
a : BOOL ;
b : BYTE ;
c : BOOL ;
```

this situation requires : $a(1 \text{ byte}) + b(1 \text{ byte}) + c(1 \text{ byte}) = 3 \text{ bytes}$.

Instead

```
a : BOOL ;
```

```
c : BOOL ;
b : BYTE ;
```

"a" and "c" are packed into the same BYTE at bit 0 and 1, therefore the used memory is:
a,c(1 byte) + b(1 byte) = 2 bytes.

- the STRUCT and the instance of FUNCTION BLOCK respect the previous rules for their internal variables, but they are always memorized at multiple of 4 bytes addresses.

Now it is simple to understand the next example:

```
a : INT ;
b : DINT ;
c : BYTE ;
d : UDINT ;
```

the total needed memory is 16 bytes:

BYTE	DESCRIPTION
0,1	variable "a"
2,3	no variables, they are lost because "b" is a DINT, therefore it is memorized in a multiple of 4 bytes address
4-7	variable "b"
8	variable "c"
9-11	no variables, they are lost because "d" is a UDINT, therefore it is memorized in a multiple of 4 bytes address
12-15	variable "d"

If the same variables are declared in the next order the memory used is 11 bytes:

```
b : DINT ;
d : UDINT ;
a : INT ;
c : BYTE ;
```

BYTE	DESCRIPTION
0-3	variable "b"
4-7	variable "d"
8,9	variable "a"
10	variable "c"

3.2.2. Memory resources of the drive

The memory resources of the drive are:

- Return stack: it is used for variables that are declared in the "VAR, END_VAR" section into a PROGRAM or FUNCTION. It is also used by the firmware of the drive for the execution of the program;
- Stack: It is used by the firmware of the drive for the execution of the program;
- VAR GLOBAL: it is used for the variables that are declared in the "VAR_GLOBAL" section;
- VAR GLOBAL RETAIN: it is used for all the variables declared in the "VAR_GLOBAL RETAIN" section;
- %M, %I, %Q: it is used for the "VAR_GLOBAL" declared as %M, %I, %Q.



Note

The size of the memory resources depends on the product.

3.2.2.1. ISD, SVM with firmware lower than 38

Memory	ISD (byte)	SVM (byte)
Return stack (VAL LOCAL and internal usage)	1364	1364
Stack	256	256
VAR GLOBAL	768	768
VAR GLOBAL RETAIN	46	46
%M	256	256
%I	8 (6 bit used)	8 (9 bit used)
%Q	4 (4 bit used)	4 (5 bit used)

When the program is built the drive also executes an internal test of the memory usage. See also *Appendix C, How can I assess the memory usage?*.

3.2.2.2. ISD, SVM with firmware greater or equal to 38

Memory	ISD (byte)	SVM (byte)
Return stack (VAL LOCAL and internal usage)	1364	1364

Memory	ISD (byte)	SVM (byte)
Stack	256	256
VAR GLOBAL	1024	1024
VAR GLOBAL RETAIN	46	46
%M	512	512
%I	8 (6 bit used)	8 (9 bit used)
%Q	4 (4 bit used)	4 (5 bit used)

When the program is built the drive also executes an internal test of the memory usage. See also [Appendix C, How can I assess the memory usage?](#).

3.2.2.3. IBD

Memory	IBD (byte)
Return stack (VAL LOCAL and internal usage)	1868
Stack	384
VAR GLOBAL	1076
VAR GLOBAL RETAIN	46
%M	512
%I	128(10 bit used)
%Q	128(7 bit used)

When the program is built the drive also executes an internal test of the memory usage. See also [Appendix C, How can I assess the memory usage?](#).

3.2.3. GLOBAL variables

The GLOBAL variables can be used by all programs of the application, their target is all the application code (that means they can be used in all programs, function blocks, and functions in the application code). They are declared at the top of the file and they are inserted into the section:

```

VAR_GLOBAL
  a : INT ;
  b AT %MW10 : INT ;
END_VAR

```

These variables can be monitored in the watch dialog (Program page) of the SDSetup. A GLOBAL variable can be simple or can be memorized into the %M memory area.

3.2.3.1. Exchange area: %M, %I, %Q variables

The %M variables are named "exchange area" because they represent a memory area used to exchange data between the program into the drive and the Modbus, EtherCAT communication protocols. %I and %Q variables are respectively the images of the status of the digital inputs and digital outputs. The digital inputs image (%I) is updated at the beginning of the MAIN program. The digital output image (%Q) is applied to physical outputs at the end of the MAIN program. The %M, %I, %Q variables are declared in the VAR_GLOBAL section. An example:

```

VAR_GLOBAL(* PLC Input Memory (0 - 127) *)
    Status0           AT %MW0       : WORD ; (* WORD0 = BYTE 0-1 *)
    Drive_Torque      AT %MX1.0     : BOOL ; (* BYTE=1, BIT=0 *)
    Drive_Busy        AT %MX1.7     : BOOL ; (* BYTE=1, BIT=7 *)
    Drive_Alarm       AT %MX0.0     : BOOL ; (* BYTE=0, BIT=0 *)
    Position          AT %MW2       : WORD ; (* WORD1 = BYTE 2-3 *)
(* PLC Output Memory (128 - 255) *)
    Command0         AT %MW128     : WORD ; (* WORD128 = BYTE 128-129
*)
    Enable_Drive     AT %MX129.0   : BOOL ; (* BYTE=129, BIT=0 *)
(* Digital Input *)
    Start            AT %IX0.2     : BOOL ; (* BYTE=0, BIT=2 *)
    Stop             AT %IX0.3     : BOOL ; (* BYTE=0, BIT=3 *)
    Disable          AT %IX0.2     : BOOL ; (* BYTE=0, BIT=2 *)
(* Digital Output *)
    Drive_Ok         AT %QX0.0     : BOOL ; (* BYTE=0, BIT=0 *)
END_VAR
    
```

These variables allow the programmer to fix where to allocate them in the memory. In other words the programmer defines how to distribute the variables into the memory. The memory allocation of the other (GLOBAL and LOCAL) variables is automatically done by the compiler.

In particular the variables are:

- %M : exchange data with Modbus, PROFIBUS, CANopen, EtherCAT;
- %I: image of digital inputs;
- %Q: image of digital outputs;

The declaration of the size of the variable is made by adding one of the following letter after %M, %I, %Q:

- X : bit;
- B : BYTE;
- W : WORD;

- D : DWORD


The exactly distribution of the memory is shown in the following tables:

%MD0											
%MW2						%MW0					
%MB3			%MB2			%MB1			%MB0		
%MX3.7	...	%MX3.0	%MX2.7	...	%MX2.0	%MX1.7	...	%MX1.0	%MX0.7	...	%MX0.0

Table 3.1. Memory struct


16#12345678			
16#1234		16#5678	
16#12	16#34	16#56	16#78
2#00010010	2#00110100	2#01010110	2#01111000

Table 3.2. Data in the memory



Note

The memory allocation of %M, %I, %Q variables has to respect the [Section 3.2.1](#), “*How the variables are stored into the memory*” rules.



Note

A variable can not be both mapped in the exchange memory and defined as RETAIN. If the %M, %I, %Q variables are defined as RETAIN, they cannot be saved into EEPROM memory.

Here follows the description of how the data are exchanged between the drive and a PLC or a PC.

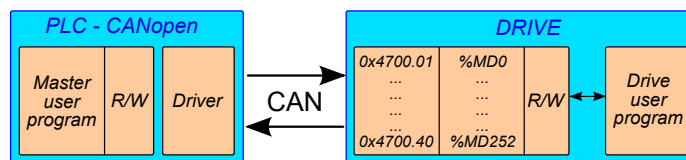


Figure 3.7. Exchange data with CANopen communication

The size of the variables, exchanged through CANopen protocol, must be DWORD. The %M exchange memory can be read or written by the CANopen protocol in this way:

- MD0 corresponds to 0x4700.01,
- ...,

- MD252 corresponds to 0x4700.40.

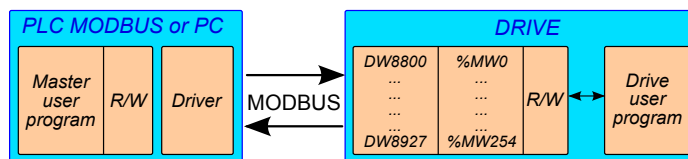


Figure 3.8. Exchange data with MODBUS communication

The size of the variables, exchanged through MODBUS protocol, can be WORD and DWORD. The %M exchange memory can be read or written by the MODBUS protocol in this way:

- MW0 corresponds to DW8800,
-
- MW254 corresponds to DW8927.

3.2.4. RETAIN variables

The RETAIN variables are GLOBAL variables with the important characteristic of to hold the values between a switch off and switch on of the drive. They are declared in a dedicated section, outside of every program:

```
VAR_GLOBAL RETAIN
  r : INT ;
END_VAR
```

The variables are distributed in the memory according with the rules written in [Section 3.2.1](#), “How the variables are stored into the memory”.



Important

In the SD series and IBD the available memory for the RETAIN variables is limited to 46 byte.

When the management of the RETAIN variables is active, then their values are reset only when the start-up of the drive needs the execution of a total reset (see [Section 3.1](#), “Main characteristics” - program reset).

In the SD series the management of the RETAIN variables has to be enabled. It is enabled when the MODBUS parameter 588 (CANopen 5FF0.9) value is 1 or 3. The default value of this parameter is 0, therefore if there are RETAIN variables the program has to write the parameter in the resett program.



Important

In the SD series, when the management of the RETAIN variables is activated, then the minimum value of Logic supply voltage, that is needed to keep alive the drive, is 40V instead of 20V (that is the minimum needed Logic supply voltage when management of the RETAIN variables is not active). When the Logic supply voltage falls under these thresholds, then the application program is stopped, and the drive will shutdown.

In the IBD the management of RETAIN variables is always enabled.

VAR_GLOBAL

```
hmi_var1 AT %MD0 : DINT ;
err : BOOL ;
```

END_VAR

VAR_GLOBAL RETAIN

```
var1 : DINT ;
```

END_VAR

PROGRAM main

```
var1 := hmi_var1;
```

END_PROGRAM

PROGRAM resetp

```
hmi_var1 := var1 ; (* at every startup load the RETAIN
variable into hmi_var1 *)
```

END_PROGRAM

(* Executed only in a reset total startup *)

PROGRAM resett

```
err := SYS_WriteObject(588,1);1 (* it makes active the management of
RETAIN variables *)
```

```
var1 := 123 ; (* first initialization of var1 *)
```

END_PROGRAM

3.2.5. CONSTANT objects

A CONSTANT object is a GLOBAL variable with the characteristic to keep its value constant. It is declared in a VAR_GLOBAL section with the information CONSTANT added.

VAR_GLOBAL CONSTANT

¹This instruction line refers only to SD series products

```
r : INT := 1000 ;
END_VAR
```

3.2.6. LOCAL variables

The LOCAL variables are declared into a PROGRAM, FUNCTION or FUNCTION BLOCK, in a "VAR - END_VAR" section.

```
VAR
  l : INT ;
END_VAR
```

Their characteristics are:

- their domain is only the PROGRAM, FUNCTION or FUNCTION BLOCK where they are declared;
- in a FUNCTION they are initialized at each execution of the function. In a PROGRAM or FUNCTION BLOCK they are initialized at the first call of the PROGRAM or FUNCTION BLOCK and then, since the second cycle, they keep the values of the previous execution;
- they cannot be monitored with SDSetup watch variable tool.

3.3. Instructions

This paragraph describes the instructions that are available to the programmer.

Instruction	Description
Arithmetical operations (ANY_INT)	
+, -, *, /	classical mathematic operations.
MULDIV(p1,p2,p3)	it executes (p1*p2)/p3 and the intermediate product (p1*p2) is internally expressed with a size bigger than the size of the data p1, p2, p3: for example if p1, p2, p3 are DINT, UDINT (32 bit), then the product p1*p2 is on 64bit; if p1,p2,p3 are INT, UINT (16bit) the product is internally stored on 32 bit; if p1, p2, p3 are SINT, USINT (8 bit) the product is internally stored stored on 16 bit.
p1 MOD p2	it returns the remainder of p1/p2.
p1 ** p2	it returns the power of p1 high to p2. The exponent p2 must be INT.
SQRT(p1)	it returns the square root of p1.
MAX(p1, p2)	it returns the biggest value between p1 and p2.
MIN(p1, p2)	it returns the smallest value between p1 and p2.
ABS(p1)	it returns the absolute value of p1.

Instruction	Description
Operations dedicated to the bit elaboration (BOOL, BYTE, WORD, DWORD)	
p1 AND p2 [OR, XOR]	logical operations AND, OR and XOR
SHL(p1, p2)	it shifts the bits of p1 in the left direction, for as many "positions/bits" as declared in p2. When it is executing the shift and the most significant bit "goes out" it is deleted.
SHR(p1, p2)	it shifts the bits of p1 in the right direction, for as many "positions/bits" as declared in p2. When it is executing the shift and the less significant bit "goes out" it is deleted.
ROL(p1, p2)	it rotates the bit of p1 in the left direction for as many "positions/bits" as declared in p2. The "rotate" operation is a special SHL(p1,p2) because when it is executing the shift and the most significant bit "goes out" it is not deleted, as in the shift operation, and it is inserted in the less significant position.
ROR(p1, p2)	it rotates the bit of p1 in the right direction for as many "positions/bits" as declared in p2. The "rotate" operation is a special SHR(p1,p2) because when it is executing the shift and the less significant bit "goes out" it is not deleted, as in the shift operation, and it is inserted in the most significant position.
Other instructions	
RETURN	it is used to jump at the end of the executing program.
EXIT	it is used to finish (abort) the execution of the instructions written inside a flow control section. For example, it is used to exit from the execution of an IF or a loop (WHILE...).
Flow control instructions	
IF	<pre>IF Expression THEN ThenInstructionsList; ELSE ElseInstructionsList; END_IF;</pre> <p>if <i>Expression</i> is TRUE, then it executes <i>ThenInstructionsList</i>, else the <i>ElseInstructionsList</i>.</p>
CASE	<pre>CASE Expression OF Element0: InstructionsList0; Element1: InstructionsList1; ELSE ElseInstructionsList; END_CASE;</pre> <p>If <i>Expression</i> is equal of one of <i>ElementX</i>, then it executes the correspondent <i>InstructionsListX</i>, else it executes <i>ElseInstructionsList</i>. The <i>ElementX</i> can be a single number, or a group of numbers. For example:</p> <pre>CASE n OF 1 : a:=a+1; 2 : b:=b+1; 11,12,13 : c:=c+1; 16..25,14,15,88..89 : d:=d+1; 29..32: e:=e+1; ELSE f:=f+1; END_CASE;</pre>

Instruction	Description
FOR	<p>FOR Variable := StartValue TO EndValue BY Increment DO InstructionsList; END_FOR;</p> <p><i>Variable</i> starts from <i>StartValue</i>. It executes <i>InstructionsList</i>, then adds to <i>Variable</i> the <i>Increment</i>. If <i>Variable</i> is less or equal than <i>EndValue</i> it repeats <i>InstructionsList</i>, else it continues the execution of the program with the instruction after END_FOR.</p>
WHILE	<p>WHILE Condition DO InstructionsList; END_WHILE;</p> <p>While <i>Condition</i> is TRUE it repeats the execution of <i>InstructionsList</i>.</p>
REPEAT	<p>REPEAT InstructionsList; UNTIL Condition END_REPEAT;</p> <p>It executes <i>InstructionsList</i> cyclically until the <i>Condition</i> is FALSE, when <i>Condition</i> becomes TRUE it continues the execution of the program from the instructions after END_REPEAT;</p>

3.4. What is a function block?

A function block is an object written in IEC language. It is necessary in order to manage actions which need a procedure to be executed. The function block permits to launch the procedure and then, at each calling, to test if the procedure is active, executed (or Done), or if it has some errors. A simple example is the function block MC_MoveAbsolute. It is used to manage the movement of an axis to a target position. The function block starts the movement, and then, when the motor arrives to the target position, it sets to TRUE the Done output. In this way, the program is informed about the status of the movement.

In the following table the general rules of a function block management are explained:

output exclusivity	the Done, Error, CommandAborted, Active outputs are mutually exclusive: only one at a time can be TRUE on a FB. If Execute is TRUE, one of these outputs has to be TRUE. Only one of the Done, Error, CommandAborted and Active outputs is set at the same time.
output status	the Done, InVelocity, Error, ErrorID and CommandAborted outputs are reset with the falling edge of Execute. However the falling edge of Execute does not stop or even influence the execution of the actual FB cycle. It must be guaranteed that the corresponding outputs are set for at least one cycle if the situation (that makes them to be set) occurs, even if Execute is reset before the FB execution has been completed. If an instance of a FB receives a new Execute before its cycle ends (as a series of commands on the same instance), the FB execution is interrupted and restarts, so the FB will not return any feedback, like Done or CommandAborted, of the previous action.

input parameters	the parameters are used on the rising edge of the Execute input. To modify an input it is necessary to change the input parameter(s) and to restart the FB.
sign rules	the Velocity, Acceleration, Deceleration and Jerk are always positive values. Position can be both positive and negative.
error handling behavior	all function blocks have two outputs, which deal with errors that can occur while executing that function block. These outputs are defined as follows: Error : the rising edge of Error informs that an error has occurred during the execution of the function block; ErrorID : Error number code. Error types can be for example: parameters out of range, state machine violation attempted. Done, InVelocity mean successful completion, so these signals are logically exclusive with Error.
behavior of Done output	The Done output is set when the commanded action has been successfully completed. When multiple function blocks are working in sequence on the same axis, the following may apply: if the movement on an axis is interrupted with another movement on the same axis without having reached the final goal of the first one, the Done of the first FB will not be set.
behavior of Active output	every FB can have an output Active, reflecting that the FB has not finished its cycle. Active is SET at the rising edge of Execute and RESET when one of the outputs Done, Aborted, or Error is set. It is recommended that this FB is kept in the active loop of the application program for at least the Active TRUE status duration, because the outputs may still change.
behavior of Command-Aborted output	CommandAborted is set, when a commanded motion is interrupted by another motion command. The reset-behavior of CommandAborted is like the one of Done. When CommandAborted occurs, the other output-signals such as InVelocity are reset.
input Enable	the Enable is level sensitive. It means that the function block does not start only at the rising edge detection of this input, but its cycle will be repeated as long as Enable is kept on high level.

Table 3.3. General rules

The behavior between the Execute and the outputs in a MC_MoveAbsolute function block is as follows:

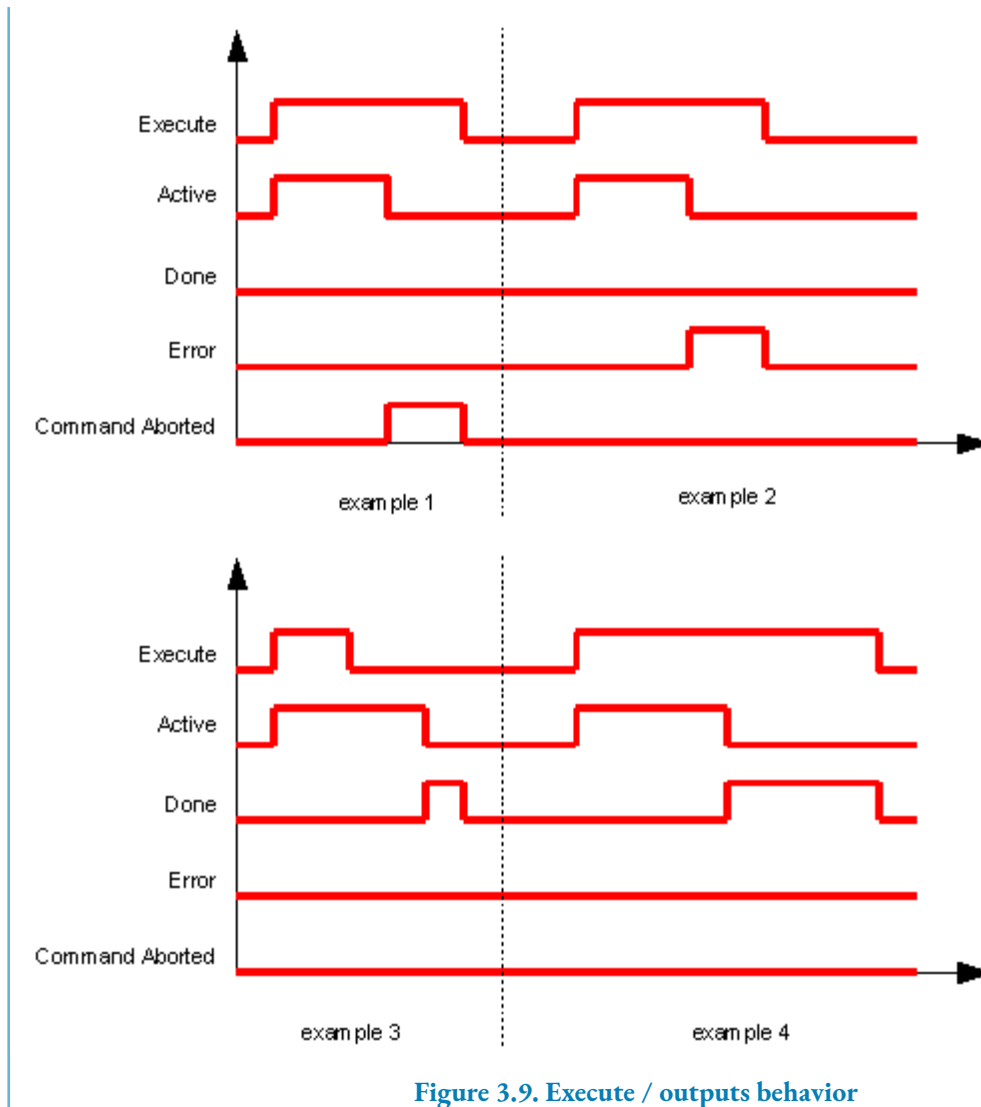


Figure 3.9. Execute / outputs behavior

example 1 : the movement is aborted, therefore the output CommandAborted becomes TRUE;

example 2 : the function block has had an error;

example 3 : the movement has been completed successfully. The Done is TRUE only for one time (cycle) because the input Execute is already FALSE;

example 4 : the movement has been completed successfully. The Done remains TRUE until the input Execute is TRUE.

Chapter 4

IEC reference guide

This chapter includes all the functions and function blocks that are necessary to write an IEC program.

4.1. System functions and function block(SYS_)

This paragraph describes some functions and functions blocks called "system function".

SYS_EnEventInt

It activates the management of an event in an INT PROGRAM.

Synopsis

```
FUNCTION SYS_EnEventInt : BOOL
  VAR_INPUT
    EventType : INT;
    IntNumber : INT;
  END_VAR
```

Return value

Value of type BOOL

If 0 then there are no errors.

Parameter

EventType

Value of type INT

Event type:

- **SYS_EVENT_LOW_VOLTAGE (8)**: it happens when the power supply passes under the low voltage threshold;
- **SYS_EVENT_TIMER (11)**: int called every programmed time. This elapsing time reference is written into 0x461A.02 (8722). This time is written into 461A.01 (8720)¹.
- **SYS_EVENT_GEAR_SYNC (4)**: it happens when the change of ratio is done, while the drive is executing an MC_Gear;
- **SYS_EVENT_GEAR_RAMP (9)**: it happens when the changing of ratio is started, while the drive is executing an MC_Gear;

¹1 = 100µs. See "Description" below.

- SYS_EVENT_CAPTURE_A (6) : it happens when the capture peripheral A captures a new position;
- SYS_EVENT_CAPTURE_B (7) : it happens when the capture peripheral B captures a new position;
- SYS_EVENT_COMPARATOR_0 (10) : event of comparator 0 detected;
- SYS_EVENT_COMPARATOR_1 (0) : event of comparator 1 detected².



Note

For a better programming it is suggested to use the CONSTANT names instead of the related numbers.

IntNumber

Value of type INT

Number of the interrupt program INT, that is called if and when the EventType happens.

Description

it activates the management of a programmed event on an INT PROGRAM. When the event happens, the INT PROGRAM is called.

The timer is set on 100 ms, but the drive approximates it to 500 ms. The result is that the INT PROGRAM intervenes according to this rule:

0..400 ms = 0 ms

500..900 ms = 500 ms

...

²This event is valid only for BD drives.

SYS_DisEventInt

It deactivates the management of an event on an INT PROGRAM.

Synopsis

```
FUNCTION SYS_DisEventInt : BOOL
  VAR_INPUT
    EventType : INT;
  END_VAR
```

Return value

Value of type BOOL

If 0 then there are no errors.

Parameter

EventType

Value of type INT

Event type to be deactivated. See [SYS_EnEventInt](#).

Description

it deactivates the management of a programmed event on an INT PROGRAM.

SYS_ReadTime

It returns the system time in milliseconds.

Synopsis

```
FUNCTION SYS_ReadTime : UDINT
```

Return value

Value of type UDINT

Actual time of the system.

Description

It returns the system time in milliseconds.

SYS_WriteObject

It writes a parameter of the drive.

Synopsis

```
FUNCTION SYS_WriteObject : BOOL
  VAR_INPUT
    ParameterNumber : UDINT ;
    Value : DINT ;
  END_VAR
```

Return value

Value of type BOOL

If 0 then there are no errors

Parameter

ParameterNumber

Value of type UDINT

It is the MODBUS address of the parameter (see [Appendix B, Parameters table](#)).

Value

Value of type DINT

Value to write in the selected parameter.

Description

It writes *Value* in the parameter with address *ParameterNumber*.



Important

Both *SYS_ReadObject* and *SYS_WriteObject* functions can access to the addresses that are related to the exchange area, but only by WORD and not by DWORD data type.

SYS_ReadObject

It reads a parameter of the drive.

Synopsis

```
FUNCTION SYS_ReadObject : DINT
  VAR_INPUT
    ParameterNumber : UDINT ;
  END_VAR
```

Return value

Value of type DINT

it reads the value of a specific parameter.

Parameter

ParameterNumber

Value of type UDINT

It is the MODBUS address of the parameter (see [Appendix B, Parameters table](#)).

Description

It reads the parameter which address is ParameterNumber.



Important

Both SYS_ReadObject and *SYS_WriteObject* functions can access to the addresses that are related to the exchange area, but only by WORD and not by DWORD data type.

SYS_Restart

It permits to restart the application.

Synopsis

```
FUNCTION SYS_Restart : BOOL
  VAR_INPUT
    Type      : INT ;
  END_VAR
```

Return value

Value of type BOOL

If 0 then there are no errors

Parameter

Type

Value of type DINT

Type of restart:

- 0 : the program restarts with the execution of PROGRAM reset;
- 1 : the program restarts with the execution of PROGRAM reset;
- 2 : it simulates a switch off-switch on of the drive. The restart will be applied 10 seconds after the execution of the command. In these 10 seconds the PLC remains on STOP state.



Note

With IBD, before to execute SYS_Restart(2) it is necessary to put the drive in the safety conditions. See the manual of the IBD for further details.

Description

It permits to restart the application. It is usually used into PROGRAM EXCEPTION, to avoid the machine blocking due to the program lock when an exception happens.

SYS_Continue

It requests to continue the execution of program EXCEPTION.

Synopsis

```
FUNCTION SYS_Continue : BOOL
```

Return value

Value of type BOOL

If 0 then there are no errors.

Description

The PROGRAM exception is usually executed only one time, but sometimes when an exception happens, it is necessary to run a safety procedure. For example, to move the axis in a safety position and then switch off the digital outputs: in this case, the PROGRAM exception has to be executed several times. If this function is called, then the PROGRAM exception does not stop its execution but repeat it once again.

SYS_MemoryToEeprom

It manages the saving and restoring of some %M into the EEPROM.

Synopsis

```

FUNCTION_BLOCK SYS_MemoryToEeprom
VAR_INPUT
    Execute      : BOOL ;
    LoadStore   : BOOL ;
    Index        : UINT;
    Lenght       : UINT;
END_VAR
VAR_OUTPUT
    Done         : BOOL ;
    Active       : BOOL ;
    Error        : BOOL ;
    ErrorID      : DINT ;
END_VAR
    
```

Parameter

Execute

Value of type BOOL

At the rising edge it starts to copy the memory.

LoadStore

Value of type BOOL

Direction of the copy:

- 0 : load data from EEPROM to %M;
- 1 : store data from %M to EEPROM.

Index

Value of type UDINT

It is the starting address of the %M to manage.

Lenght

Value of type UINT

It is the length of the area to move.

Done

Value of type BOOL

Flag that is set to TRUE when the copy operation is finished.

Active

Value of type BOOL

Flag that is set to TRUE when the procedure is in progress.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred in the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#)).

Description

The RETAIN area of the drive has few bytes, but sometimes it is important to save into EEPROM some variables. This function block permits to save a part of the %M memory area into the EEPROM, and also to restore a part of the %M area with values saved into EEPROM. The part of the managed %M memory area is declared with its start address (Index) and with its Lenght.

SYS_RdIECSafeCondition

It reads the value of the semaphore *IECSafeCondition*.

Synopsis

```
FUNCTION SYS_RdIECSafeCondition : INT
  VAR_INPUT
    Event   : DWORD;
  END_VAR
```

Return value

Value of type INT

It is the state of the *IECSafeCondition* semaphore (0 = green, 1 = yellow, 2 = red).

Parameter

Event

Value of type DINT

It has to be 0. It means 'all events'.

Description

It reads the state of the semaphore *IECSafeCondition*. See the example [Section 4.5.3, "Example of the management of a program safety condition request"](#).

SYS_RqsIECSafeCondition

It returns TRUE when the event is requesting to the program to go in safe condition.

Synopsis

```
FUNCTION SYS_RqsIECSafeCondition : BOOL
  VAR_INPUT
    Event    : DWORD;
  END_VAR
```

Return value

Value of type BOOL

TRUE means that the system manager is requesting to the program to go in its safety condition before to continue the action referred to the event.

Parameter

Event

Value of type DINT

It is the reference number of the event that is making the request.

Description

It informs when the system manager is requesting to the program to go in its safety condition before to continue the action referred to the event. It is useful when the *IECSafeCondition* is "yellow" (see *SYS_WrIECSafeCondition*).

In this situation when the function *SYS_RqsIECSafeCondition* returns TRUE, the program can execute all the operations to put the application in a well defined condition and then it switches the semaphore to "green". After that, the safety condition for the IEC program is reached and the system manager can continue the execution of the action referred to the event.

See the example *Section 4.5.3, "Example of the management of a program safety condition request"*

SYS_WrIECSafeCondition

It writes the internal semaphore *IECSafeCondition*.

Synopsis

```

FUNCTION SYS_WrIECSafeCondition : DINT
    VAR_INPUT
        Event    : DWORD;
        State    : INT;
    END_VAR
    
```

Return value

Value of type DINT

If 0 then there are no errors

Parameter

Event

Value of type DINT

It has to be 0. It means all events.

State

Value of type INT

It is the value to write in *IECSafeCondition* semaphore:

- [0] this value switches the semaphore to "green" state, therefore the execution of the event is allowed;
- [1] this value switches the semaphore to "yellow" state. When the system manager requests to put the program in safety condition, this condition suspends the execution of the event until the program sets the semaphore to "green" (State = 0).
- [2] this value switches the semaphore to "red" state, therefore the execution of the event is aborted.

Description

It writes the state of the internal semaphore *IECSafeCondition*. When the State is [1], the IEC program wants to suspend the execution of the event that has requested the safety condition.

See the example [Section 4.5.3, “Example of the management of a program safety condition request”](#)

4.2. Axes management (MC_)

This paragraph includes all the informations that are necessary to manage an axis in an IEC program.

4.2.1. Axis status

The axis can be in one of the following status:

- **INITIALIZATION** : when the axis is in this status, the program is initializing its variables;
- **STANDSTILL** : the axis keeps its command position;
- **ERRORSTOP** : the axis has had an error;
- **STOPPING** : the axis is executing a stop ramp (see [MC_Stop](#));
- **EMERGENCY STOPPING** : the axis is executing an emergency stop ramp (see [MC_EmergencyStop](#));
- **DISCRETE MOTION** : the axis is executing a discrete movement. These movements finish automatically and without the execution of others motion function blocks and then the state automatically returns STANDSTILL.
- **CONTINUOUS MOTION** : the axis is executing a continuous movement. These movements are "endless" (e.g. a velocity command) and are interrupted when another motion function block is executed, or if there is an error. The axis state never passes directly from CONTINUOUS MOTION to STANDSTILL.
- **HOMING** : the axis is executing an homing procedure (see [MC_Home](#)).

The function block [MC_ReadStatus](#) returns the status of the axis.

The state diagram describes which motion function blocks or events may change the status of the axis.

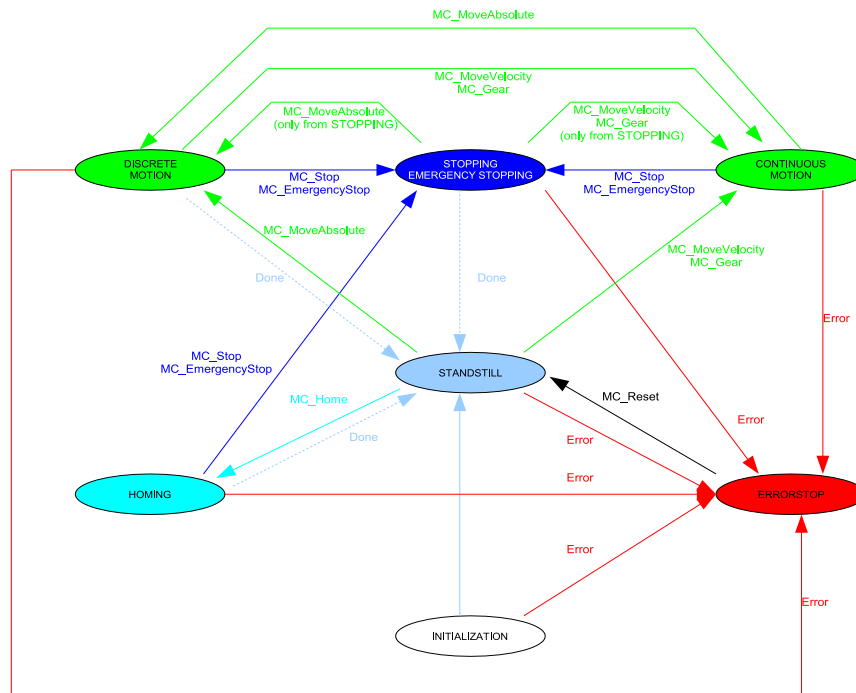


Figure 4.1. Axis status diagram

When the axes program starts the axis is always in **INITIALIZATION** status.

When the drive is "ReadyToSwitchOn" (see [Section 4.2.2.1, "How digital inputs and outputs manage the drive"](#)) the axis status automatically becomes **STANDSTILL**.

When the axis has an error, its status becomes **ERRORSTOP** and the behavior of the motor depends on the drive management.

Note

- when the motor arrives to an hardware or software limit switch, the movement is stopped;
- for all the other alarms, the drive switches off the torque of the motor.

When a motion function block is launched, then the axis changes its status according to the following relations:

- with *MC_Home* the status becomes **HOMING**;
- with *MC_MoveAbsolute* the status becomes **DISCRETE MOTION**;
- with *MC_MoveVelocity* or **MC_Gear** the status becomes **CONTINUOUS MOTION**;
- with *MC_Stop* the status becomes **STOPPING**;

- with *MC_EmergencyStop* the status becomes EMERGENCY STOPPING.

4.2.2. Drive status

The drive status is described by the ReadyToSwitchOn, Run, Fault digital outputs (see *Section 4.2.2.1, “How digital inputs and outputs manage the drive”*). It can be read by *MC_ReadDriveStatus*.

4.2.2.1. How digital inputs and outputs manage the drive

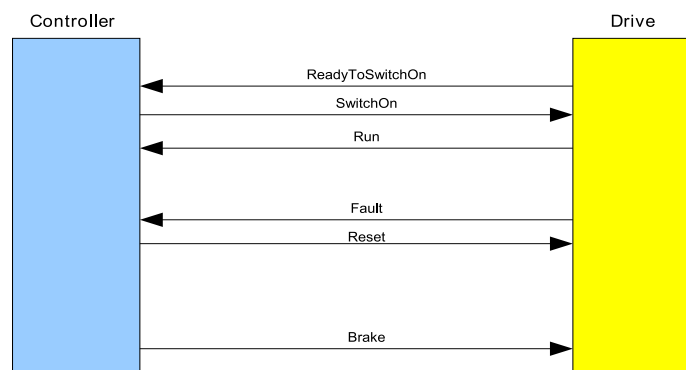


Figure 4.2. Interface between controller and drive

Digital Inputs:

- ReadyToSwitchOn : it is TRUE when the drive is ready to work.
- Run : it is TRUE when the drive is switched on (the power stage of the drive is switched on).
- Fault : it is TRUE when the drive has an alarm. When it is TRUE the digital inputs ReadyToSwitchOn and Run are FALSE.

Digital Outputs:

- SwitchOn : when it becomes TRUE, the controller commands the drive to switch on the power stage;
- Reset : on the rising edge, the axes program tries to recover the drive from a Fault situation;

- Brake : when it is TRUE, the motor can freely run, when it is FALSE the brake is on, therefore the motor is blocked. The brake control can be automatically managed from the axes program or be driven in manual mode.

In the following diagram there is the description of the SwitchOn timing procedure with automatic Brake control:

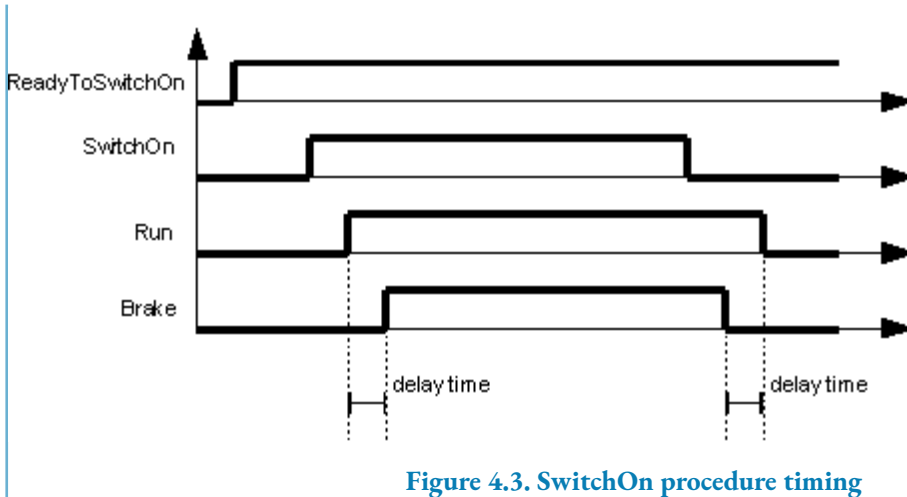


Figure 4.3. SwitchOn procedure timing

The next diagram describes the drive behavior when there is an alarm. The Fault signal becomes TRUE and immediately the Brake, Run and ReadyToSwitchOn I/Os are set to FALSE.

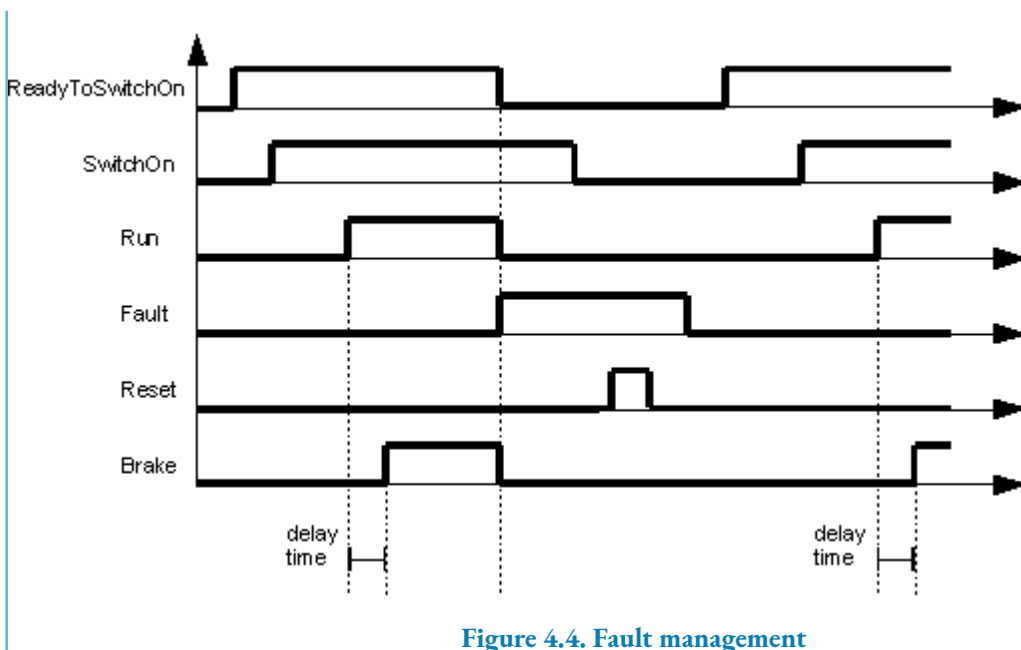


Figure 4.4. Fault management



Important

For SD series there is not the brake management.

4.2.3. Axis functionalities

This section describes some axis functionalities.

4.2.3.1. SW limits management

There are a positive and a negative SW limit:

- when the axis exceeds the positive limit with a positive velocity, then the axis switches to the *ERRORSTOP* state with **Axis is arrived at a SW limit switch** error code;
- when the axis exceeds the negative limit with a negative velocity, then the axis switches to *ERRORSTOP* state with **Axis is arrived at a SW limit switch** error code.

To use a SW limit it is necessary:

1. to configure the position limit (in the SYS_WriteObject the needed parameters numbers are: 4327, 4329);
2. to activate the position limit management (for the SYS_WriteObject the needed parameter number is 4326).

4.2.3.2. HW limits management

There are a positive and a negative HW limit:

- when the axis exceeds the positive limit with a positive velocity then the axis switches to *ERRORSTOP* state with **Axis is arrived at a limit switch** error code;
- when the axis exceeds the negative limit with a negative velocity then the axis switches to *ERRORSTOP* state with **Axis is arrived at a limit switch** error code.

To use a HW limit is necessary to configure which inputs are the positive and negative limits.

4.2.3.3. Emergency ramp

This ramp is executed when the motor arrives to an hardware or limit switch. It is configured with the parameter number 4343.

4.2.4. Data Type : **AXIS_REF**

The data type `AXIS_REF` represents the axis in an IEC program.

AXIS_REF

It is the axis reference.

Synopsis

```
TYPE
  AXIS_REF : STRUCT
    num      : <type>INT</type>;
  END_STRUCT;
END_TYPE
```

Elements

num

Value of type INT

It is an internal number. In a project each axis has an identification number, different from the other axes.

Description

This Data Type is the axis reference. It is used to declare an axis in the project. It is important to initialize the struct with Num = MC_REF_AXIS_MAIN.

4.2.5. Function blocks list

The function blocks can be divided in two groups: motion and administrative (not driving motion).

4.2.5.1. Motion function blocks

This paragraph describes the function blocks dedicated to movement actions.

MC_EmergencyStop

It commands an emergency ramp stop. It is a non-controlled stop.

Synopsis

```

FUNCTION_BLOCK MC_EmergencyStop

VAR_IN_OUT
    Axis                : <type>AXIS_REF</type> ;
END_VAR
VAR_INPUT
    Execute             : <type>BOOL</type> ;
    Deceleration        : <type>DINT</type> ;
    Jerk                : <type>DINT</type> ;
END_VAR
VAR_OUTPUT
    Done                : <type>BOOL</type> ;
    CommandAborted      : <type>BOOL</type> ;
    Error               : <type>BOOL</type> ;
    ErrorID             : <type>DINT</type> ;
END_VAR
    
```

Parameter

Axis

Value of type AXIS_REF

Axis reference

Execute

Value of type BOOL

At the rising edge the axis starts to move

Deceleration

Value of type DINT

Value of deceleration [axis unit per second]

Jerk

Value of type DINT

It must be 100.

Done

Value of type BOOL

Zero velocity is reached.

CommandAborted

Value of type BOOL

Flag that is set to TRUE if the command is aborted by another motion command.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred in the function block

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#))

Description

This function block commands an uncontrolled motion stop and transfers the axis to the *EMERGENCYSTOPPING* status. The program stops the axis in open loop mode. It aborts any motion function block execution in progress. With the Done output set, the state is transferred to *STANDSTILL*. While the axis is in state *EMERGENCYSTOPPING* status, no other FB can perform any motion on the same axis. If the *MC_Stop* is launched and the axis is already in *STANDSTILL*, then the Done output is immediately set to TRUE.

MC_Gear

It manages a gear movement between an encoder and the axis.

Synopsis

```

FUNCTION_BLOCK MC_Gear
VAR_IN_OUT
    Master          : Enc_Ref;
    Slave          : AXIS_REF;
END_VAR
VAR_INPUT
    StartGear      : BOOL ;
    Execute        : BOOL ;
    Mode           : WORD ;
    RatioInNumerator : DINT ;
    RatioInDenominator : DINT := 1000;
    RatioEndNumerator : DINT ;
    RatioEndDenominator : DINT := 1000;
    MasterSpace     : DINT ;
    InpStart       : BOOL ;
    MasterStart     : DINT ;
END_VAR
VAR_OUTPUT
    InGear         : BOOL ;
    Active         : BOOL ;
    CommandAborted : BOOL ;
    Error          : BOOL ;
    ErrorID        : DINT ;
    State          : INT ;
    SlaveStart     : DINT ;
END_VAR
    
```

Parameter

Master

Value of type Enc_Ref

Encoder master reference of the gear movement. The Master has to be IO_REF_ENC_AUXILIARY (see [ENC_REF](#)). In this way the Master is the auxiliary encoder. In order to use the internal simulated master, the auxiliary master configuration must be changed.

Slave

Value of type AXIS_REF

Axis slave reference of the gear movement.

StartGear

Value of type BOOL

At the rising edge it starts the gear movement between the encoder and the axis. When it is TRUE the status of the outputs of the function block are managed. At the rising edge it executes the action of the Execute input, too.

Execute

Value of type BOOL

When the StartGear input is TRUE, at the rising edge it makes active the change of the ratio of the gear movement. The characteristics of the change are defined by the other inputs.

Mode

Value of type WORD

It defines the mode of change of the ratio of the gear movement. One of the bits 1, 2, 3, 4 must be set.

- bit 0 : initial gear movement ratio :
0 = actual ratio;
1 = ratio declared with RatioInNumerator/RatioInDenominator;
- bit 1 : if it is 1 than the change of the ratio from initial to end ratio is started at the rising edge of the Execute input;
- bit 2 : if it is 1 than the change of the ratio from initial to end ratio is started at the rising edge of the InpStart input;
- bit 3 : if it is 1 than the change of the ratio from initial to end ratio is started when the master position is greater than the MasterStart input. It is used when the master has a positive velocity;
- bit 4 : if it is 1 than the change of the ratio from initial to end ratio is started when the master position is less than the MasterStart input. It is used when the master has a negative velocity.

RatioInNumerator

Value of type DINT

It is the numerator of the initial ratio. The range of the admitted values is $1 \div 32768$.

RatioInDenominator

Value of type DINT

It is the denominator of the initial ratio. The range of the admitted values is $1 \div 32768$.

RatioEndNumerator

Value of type DINT

It is the numerator of the gear movement ratio at the end of the change. The range of the admitted values is $1 \div 32768$.

RatioEndDenominator

Value of type DINT

It is the denominator of the gear movement ratio at the end of the change. The range of the admitted values is $1 \div 32768$.

MasterSpace

Value of type DINT

It is the master space within the change of the ratio will be done. The minimum value is 1.

InpStart

Value of type BOOL

When Mode has bit 2 = 1 , then the change of the ratio starts at the rising edge of InpStart.

MasterStart

Value of type DINT

When Mode has bit 3 or 4 = 1 , this input defines the master position threshold for the change of the ratio.

InGear

Value of type BOOL

This flag is set to TRUE when the axis is moving in gear movement with ratio equals to $\text{RatioEndNumerator}/\text{RatioEndDenominator}$.

Active

Value of type BOOL

The function block is active.

CommandAborted

Value of type BOOL

Flag that is set to TRUE if the command is aborted by another motion command.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred in the function block

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#))

State

Value of type INT

State of the procedure for the change of the ratio:

- 0 : not active;
- 1 : it is waiting for the event which triggers the start of the ratio change;
- 2 : the State passes to this number when Mode has bit 3 or 4 = TRUE. In this state it is waiting that the master position becomes higher (or lower) than MasterStart;
- 3 : it is changing the ratio;
- 4 : the ratio has been changed.

SlaveStart

Value of type DINT

The slave position at the beginning of the changing of the gear movement ratio.

Description

This function block manages the gear movement between a master encoder and the axis. It has two functions:

- it activates the gear movement;
- while the axis is moving in gear mode, with this function block it is possible to change the gear ratio between the master and the slave movement. There are several modes to

manage the change. The changing can start from the actual ratio or from a defined ratio, and can also start immediately at the rising edge of the Execute input or when a particular condition happens. All these characteristics are defined by the Mode, InpStart and MasterStart inputs. The initial ratio, when it is necessary, is declared with RatioInNumerator/RatioInDenominator. The gear ratio at the end of the change is RatioEndNumerator/RatioInDenominator. The change of the ratio, from the initial to the end one, is always executed in MasterSpace master units.

MC_Home

It executes a selected homing procedure.

Synopsis

```
FUNCTION_BLOCK MC_Home
VAR_IN_OUT
  Axis : AXIS_REF;
END_VAR
VAR_INPUT
  Execute : BOOL ;
  Position : DINT;
  HomingMode: INT;
  VelocitySearchSwitch : DINT;
  VelocitySearchZero : DINT;
END_VAR
VAR_OUTPUT
  Done : BOOL ;
  CommandAborted : BOOL ;
  Error : BOOL ;
  ErrorID : DINT ;
END_VAR
```

Parameter

Axis

Value of type AXIS_REF

Axis reference

Execute

Value of type BOOL

At the rising edge it starts the action.

Position

Value of type DINT

It is the absolute position when the reference signal is detected.

HomingMode

Value of type INT

It selects the homing type:

- 1: homing against the switch connected to home input (the limit switch detection interrupts the procedure);
- 2: homing against the switch connected to home input and then searching of the "zero mark" encoder reference pulse (the limit switch detection interrupts the procedure); for the SD drives it is allowed only to the ones which feedback is an incremental encoder;
- 3: homing against the switch connected to home input, with the limit switch management;
- 4: homing against the switch connected to home input and then searching of the "zero mark" encoder reference pulse, with the limit switch management; for the SD drives it is allowed only for the ones which feedback is an incremental encoder;
- 5: set the position of the axis equals to the Input Position without any movement;
- 6: the axis searches the encoder reference pulse "zero mark" (the limit switch detection interrupts the procedure); for the SD drives it is allowed only for the ones which feedback is an incremental encoder;
- 7: homing against the switch connected to the hw limit input without "zero mark" reference pulse;
- 8: homing against the switch connected to hw limit input and then searching of the "zero mark" encoder reference pulse; for the SD drives it is allowed only for the ones which feedback is an incremental encoder;
- 101: it is equal to 1, but at the end of the procedure the offsets are saved in the permanent memory so that the position will be correctly restored after a reboot of the drive. It is allowed only for SD drives which feedback is an absolute encoder;
- 103: it is equal to 3, but at the end of the procedure the offsets are saved in the permanent memory so that the position will be correctly restored after a reboot of the drive. It is allowed only for SD drives which feedback is an absolute encoder;
- 105: it is equal to 5, but at the end of the procedure the offsets are saved in the permanent memory so that the position will be correctly restored after a reboot of the drive. It is allowed only for SD drives which feedback is an absolute encoder;
- 107: it is equal to 7, but at the end of the procedure the offsets are saved in the permanent memory so that the position will be correctly restored after a reboot of the drive. It is allowed only for SD drives which feedback is an absolute encoder;

VelocitySearchSwitch

Value of type DINT

It is the velocity used to move the axis toward the limit switch. It has to be a value different from zero.

VelocitySearchZero

Value of type DINT

In the HomingType = (1),(3) it is the velocity used to move the axis out of the micro, while in the modes (2),(4) it is the velocity used to move the axis toward the reference pulse of the encoder. It has to be a value different from zero.

Done

Value of type BOOL

Flag that is set to TRUE when the homing procedure is correctly finished.

CommandAborted

Value of type BOOL

Flag that is set to TRUE if the command is aborted by another motion command.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#)).

Description

This function block starts an homing procedure. An homing procedure is used to get a reference position in a defined real condition. The possible procedures are:

- 1, 3, 101, 103:
 1. the axis moves toward the home switch with `VelocitySearchSwitch` velocity;
 2. when the axis reaches the home switch, the axis starts to move with `VelocitySearchZero` velocity;
 3. when the axis leaves the home switch, the axis position is set to the `Position` value and then the axis is stopped.

the difference between mode 1 and 3 is the axis behavior if the movement reaches a limit (hw limit switch or software limit) while the homing procedure is in progress. Furthermore in modes 1 the axis goes in ERRORSTOP, instead in mode 3 the axis maintains the HOMING state and it reverts the movement in the opposite direction until the axis finds the home switch.

- 2, 4, 102, 104:

1. the axis moves toward home switch with VelocitySearchSwitch velocity;
2. when the axis reaches the home switch, the axis starts to move with VelocitySearchZero velocity;
3. when the axis reaches the reference pulse of the encoder, the axis position is set to the Position value and then it is stopped.

the difference between mode 2 and 4 is the axis behavior if the movement reaches a limit (hw limit switch or software limit) while the homing procedure is in progress. Furthermore in modes 2 the axis goes in ERRORSTOP, instead in mode 4 the axis maintains the HOMING state and it reverts the movement in the opposite direction until the axis finds the home switch.

- 5:

the axis position is set to the Position input value without any movement.

- 6:

1. the axis moves with VelocitySearchZero velocity;
2. when the axis reaches the reference pulse of the encoder, the axis position is set to the Position input value and then it is stopped.

- 7: The procedure is equal to the number 1, but at the beginning the axis moves toward the hw limit;

- 8: The procedure is equal to the number 2, but at the beginning the axis moves toward the hw limit.

When the homing procedure is in progress the axis state is *HOMING*. When the procedure ends, the Done output is set to TRUE and the axis state becomes *STANDSTILL*.



Important

The homing procedure is allowed when the axis state is STANDSTILL.

MC_MoveAbsolute

It moves the axis to an absolute position.

Synopsis

```
FUNCTION_BLOCK MC_MoveAbsolute

VAR_IN_OUT
  Axis          : <type>AXIS_REF </type>;
END_VAR
VAR_INPUT
  Execute       : <type>BOOL</type>  ;
  Position      : <type>DINT</type>  ;
  Velocity      : <type>DINT</type>  ;
  Acceleration  : <type>DINT</type>  ;
  Deceleration  : <type>DINT</type>  ;
  Jerk          : <type>DINT</type>  ;
  Direction     : <type>INT</type>   ;
END_VAR
VAR_OUTPUT
  Done          : <type>BOOL</type>  ;
  CommandAborted : <type>BOOL</type> ;
  Error         : <type>BOOL</type>  ;
  ErrorID       : <type>DINT</type>  ;
END_VAR
```

Parameter

Axis

Value of type AXIS_REF

Axis reference

Execute

Value of type BOOL

At the rising edge it starts to move the axis.

Position

Value of type DINT

It is the target absolute position.

Velocity

Value of type DINT

It is the target velocity.

Acceleration

Value of type DINT

It is the acceleration which the drive uses to start the axis movement in order to reach the target Velocity.

Deceleration

Value of type DINT

It is the deceleration which the drive uses to stop the axis.

Jerk

Value of type DINT

It must be 100.

Direction

Value of type DINT

It is not used.

Done

Value of type BOOL

Flag that is set to TRUE when the movement is finished.

CommandAborted

Value of type BOOL

Flag that is set to TRUE if the command is aborted by another motion command.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see *Table A.1*).

Description

This function block moves the axis from the current position to the target Position.

When the movement is executing the axis state is *DISCRETEMOTION*. When the procedure finishes the output Done is set to TRUE and the axis state becomes *STANDSTILL*. If the MC_MoveAbsolute is launched and the axis is in *STANDSTILL* and it is already in the target position then the output Done is set to TRUE immediately.

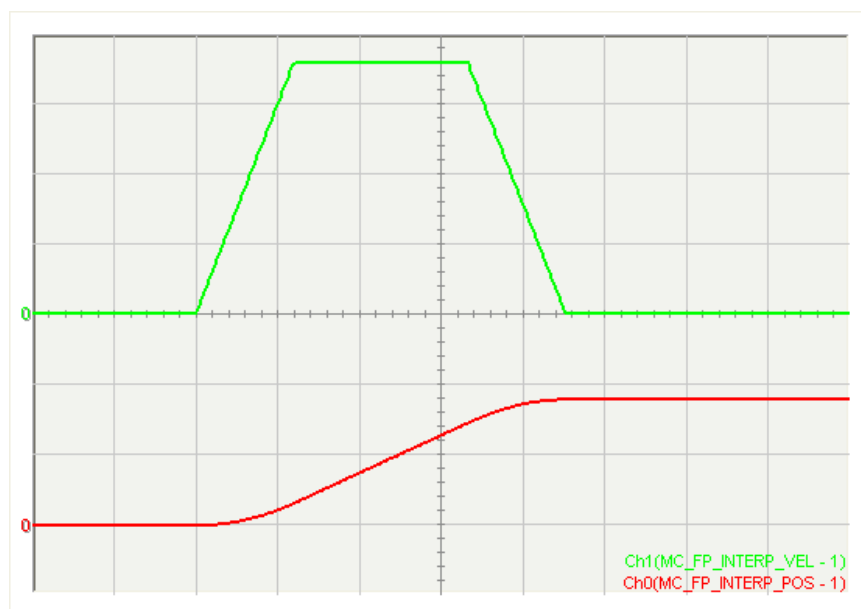


Figure 4.5. position movement

The position movement needs some parameters:

- **end velocity:** it is the velocity in the final part of the movement. It is useful when the end of the movement has to be executed in a "softly mode", that means that the speed near the target position is defined by the programmer and not automatically calculated by the drive;
- **anticipation of end position:** it is the space within the movement has to run with end velocity;

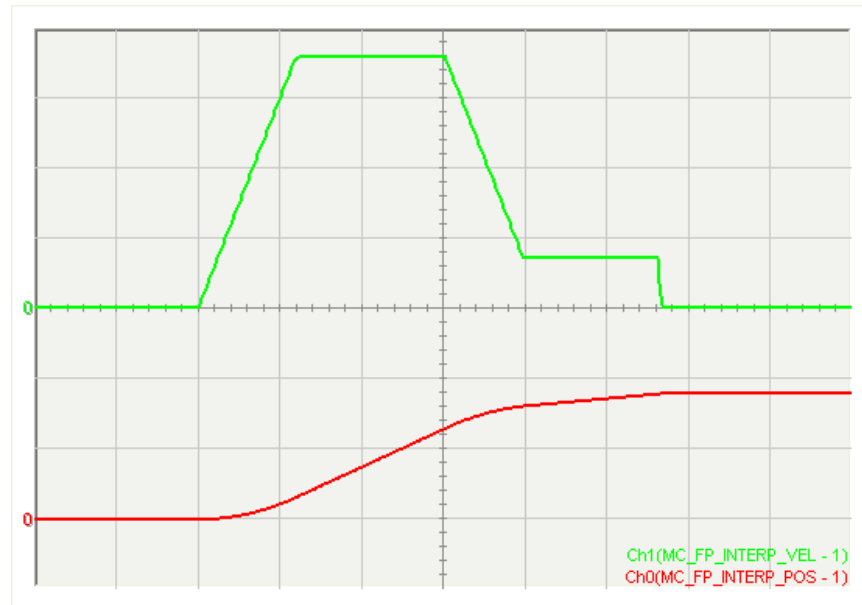


Figure 4.6. position movement with end velocity

- when the axis has a velocity not equal to zero, and the target position is too close to the actual position, the position movement is solved with a profile which guarantees the deceleration and describes an overshoot.

MC_MoveVelocity

It moves the axis with a target velocity.

Synopsis

```
FUNCTION_BLOCK MC_MoveVelocity

VAR_IN_OUT
  Axis          : <type>AXIS_REF</type> ;
END_VAR
VAR_INPUT
  Execute       : <type>BOOL</type> ;
  Velocity      : <type>DINT</type> ;
  Acceleration  : <type>DINT</type> ;
  Deceleration  : <type>DINT</type> ;
  Jerk          : <type>DINT</type> ;
  Direction     : <type>INT</type> ;
END_VAR
VAR_OUTPUT
  InVelocity    : <type>BOOL</type> ;
  CommandAborted : <type>BOOL</type> ;
  Error         : <type>BOOL</type> ;
  ErrorID       : <type>DINT</type> ;
END_VAR
```

Parameter

Axis

Value of type AXIS_REF

Axis reference

Execute

Value of type BOOL

At the rising edge it starts to move the axis.

Velocity

Value of type DINT

It is the target velocity.

Acceleration

Value of type DINT

It is the acceleration which the drive uses to start the axis movement in order to reach the target Velocity.

Deceleration

Value of type DINT

It is the deceleration which the drive uses to stop the axis

Jerk

Value of type DINT

It must be 100.

Direction

Value of type DINT

It is not used.

In Velocity

Value of type BOOL

Flag that is set to TRUE when the theoretical profile arrives to target Velocity.

CommandAborted

Value of type BOOL

Flag that is set to TRUE if the command is aborted by another motion command.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#)).

Description

This function block moves the axis to target Velocity.

When the movement is executing the axis state is *CONTINUOUSMOTION*. If the target velocity is set to 0, when the movement reaches the target velocity (in other words, it stops) the axis changes its state in *STANDSTILL*.

MC_Stop

It commands a motion stop.

Synopsis

```
FUNCTION_BLOCK MC_Stop

VAR_IN_OUT
  Axis          : <type>AXIS_REF</type> ;
END_VAR
VAR_INPUT
  Execute       : <type>BOOL</type> ;
  Deceleration  : <type>DINT</type> ;
  Jerk          : <type>DINT</type> ;
END_VAR
VAR_OUTPUT
  Done          : <type>BOOL</type> ;
  CommandAborted : <type>BOOL</type> ;
  Error         : <type>BOOL</type> ;
  ErrorID       : <type>DINT</type> ;
END_VAR
```

Parameter

Axis

Value of type AXIS_REF

Axis reference

Execute

Value of type BOOL

At the rising edge it starts to move the axis.

Deceleration

Value of type DINT

Value of deceleration [axis's unit per second]

Jerk

Value of type DINT

It must be 100.

Done

Value of type BOOL

Flag that is set to TRUE when the Zero velocity is reached.

CommandAborted

Value of type BOOL

Flag that is set to TRUE if the command is aborted by another motion command.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#))

Description

This function block commands a motion stop and transfers the axis to the *STOPPING* state. It aborts any ongoing motion function block execution. When the Done output is set, the state is transferred to *STANDSTILL*. If the MC_Stop is launched and the axis is in *STANDSTILL*, then the output Done is set to TRUE immediately.

4.2.5.2. Administrative function blocks

MC_Power

It controls the power stage.

Synopsis

```
FUNCTION_BLOCK MC_Power

VAR_IN_OUT
  Axis          : <type>AXIS_REF</type>;
END_VAR
VAR_INPUT
  Enable        : <type>BOOL</type> ;
END_VAR
VAR_OUTPUT
  Status        : <type>BOOL</type> ;
  Error         : <type>BOOL</type> ;
  ErrorID       : <type>DINT</type> ;
END_VAR
```

Parameter

Axis

Value of type AXIS_REF

Axis reference.

Enable

Value of type BOOL

At the rising edge it sets the power on. It does not work like an usual 'Enable' input:

- To enable the axis it is necessary to execute a rising edge of this input, from 0 to 1;
- To disable the axis it is necessary to execute a falling edge of this input, from 1 to 0;

Status

Value of type BOOL

Effective state of power stage. TRUE = axis torque active; FALSE = axis free;

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#)).

Description

This function block controls the power stage (ON or OFF). See an example in [MC_ReadDriveStatus](#).

MC_ReadActualPosition

It returns the actual position of the axis.

Synopsis

```
FUNCTION_BLOCK MC_ReadActualPosition

VAR_IN_OUT
    Axis          : <type>AXIS_REF</type> ;
END_VAR
VAR_INPUT
    Enable        : <type>BOOL</type> ;
END_VAR
VAR_OUTPUT
    Done          : <type>BOOL</type> ;
    Error         : <type>BOOL</type> ;
    ErrorID       : <type>DINT</type> ;
    Position      : <type>DINT</type> ;
    Velocity      : <type>DINT</type> ;
    Acceleration  : <type>DINT</type> ;
END_VAR
```

Parameter

Axis

Value of type AXIS_REF

Axis reference.

Enable

Value of type BOOL

As long as 'Enable' is true, it continuously gets the value of the parameter.

Done

Value of type BOOL

Flag that is set to TRUE when the read actual position value is available.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#)).

Position

Value of type DINT

Absolute actual position in axis's unit.

Velocity

Value of type DINT

Actual velocity in axis's unit per second.

Acceleration

Value of type DINT

Actual acceleration in axis's unit per second².

Description

This function block, while enabled, continuously returns the value of the parameters. The parameters are the actual position, velocity and acceleration of the axis.



Note

Axis's unit is step.

MC_ReadCommandPosition

It returns the commanded position of the axis.

Synopsis

```
FUNCTION_BLOCK MC_ReadCommandPosition

VAR_IN_OUT
  Axis          : <type>AXIS_REF</type> ;
END_VAR
VAR_INPUT
  Enable        : <type>BOOL</type> ;
END_VAR
VAR_OUTPUT
  Done          : <type>BOOL</type> ;
  Error        : <type>BOOL</type> ;
  ErrorID      : <type>DINT</type> ;
  Position     : <type>DINT</type> ;
  Velocity     : <type>DINT</type> ;
  Acceleration : <type>DINT</type> ;
END_VAR
```

Parameter

Axis

Value of type AXIS_REF

Axis reference.

Enable

Value of type BOOL

As long as 'Enable' is true, it continuously gets the value of the parameter.

Done

Value of type BOOL

Flag that is set to TRUE when the read commanded position value is available.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#))

Position

Value of type DINT

Absolute commanded position in axis's unit.

Velocity

Value of type DINT

Commanded velocity in axis's unit per second.

Acceleration

Value of type DINT

Commanded acceleration in axis's unit per second².

Description

This function block, while enabled, continuously returns the value of the parameters. The parameters are the commanded position, velocity and acceleration of the axis.



Note

Axis's unit is step.

MC_ReadDriveStatus

It returns drive status.

Synopsis

```
FUNCTION_BLOCK MC_ReadDriveStatus

VAR_IN_OUT
  Axis      : <type>AXIS_REF</type> ;
END_VAR
VAR_INPUT
  Enable    : <type>BOOL</type> ;
END_VAR
VAR_OUTPUT
  Done      : <type>BOOL</type> ;
  Error     : <type>BOOL</type> ;
  ErrorID   : <type>DINT</type> ;
  ReadyToSwitchOn : <type>BOOL</type> ;
  SwitchedOn : <type>BOOL</type> ;
  Run       : <type>BOOL</type> ;
  Fault     : <type>BOOL</type> ;
END_VAR
```

Parameter

Axis

Value of type AXIS_REF
Axis reference.

Enable

Value of type BOOL
As long as 'Enable' is true, it continuously gets the value of the parameter.

Done

Value of type BOOL
Flag that is set to TRUE when the value is available.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#))

ReadyToSwitchOn

Value of type BOOL

Flag that is set to TRUE when the drive is ready to switch on.

SwitchedOn

Value of type BOOL

Flag that is set to TRUE as long as the drive is switched on. The power stage is on.

Run

Value of type BOOL

Flag that is set to TRUE when the power stage is on and the drive can move the motor. This output is equal to the output 'Status' of MC_Power.

Fault

Value of type BOOL

Flag that is set to TRUE when the drive has an alarm.

Description

This function block describes the drive state. See [Section 4.2.2.1, “How digital inputs and outputs manage the drive”](#)

When the output ReadyToSwitchOn is TRUE then the drive is ready to be switched on, therefore the power stage can be enabled by using [MC_Power](#). When the power stage is on and the drive is ready to move the motor, both the SwitchedOn and Run outputs are TRUE.

If the drive is on alarm state, then the output Fault is TRUE and the other outputs are all FALSE. In this situation the axis state is [ERRORSTOP](#) (see [MC_ReadStatus](#)).

This example is used to show how to give the power to the drive:

The input 'SwitchOn = TRUE' ;

1. first of all it is necessary to start the management of the movement in the IEC program;

2. when the axis management is started, the program reads the drive status;
3. when the drive status output 'ReadyToSwitchOn' is TRUE, the program can enable the stage;
4. when the power is on, then the output 'Status' of 'MC_Power = TRUE' and output 'Run' of 'MC_ReadDriveStatus = TRUE'.

**Note**

At the first switched on procedure some drives execute an alignment procedure.

```
(* it starts the management of the axis movement in the IEC program *)
```

```
MC_Start_inst(Execute:=SwitchOn);
```

```
(* it reads the drive status *)
```

```
MC_DriveStatus_inst(Axis:= Axis, Enable := MC_Start_inst.Done);
```

```
(* when the drive is 'ReadyToSwitchOn' it switches on *)
```

```
MC_Power_inst(Axis:=Axis, Enable:= MC_DriveStatus_inst.ReadyToSwitchOn);
```

Example 4.1. start-up procedure

MC_ReadStatus

It returns, in detail, the status of the axis.

Synopsis

```

FUNCTION_BLOCK MC_ReadStatus

VAR_IN_OUT
    Axis          : <type>AXIS_REF</type> ;
END_VAR
VAR_INPUT
    Enable        : <type>BOOL</type> ;
END_VAR
VAR_OUTPUT
    Done          : <type>BOOL</type> ;
    Error         : <type>BOOL</type> ;
    ErrorID       : <type>DINT</type> ;
    ErrorStop     : <type>BOOL</type> ;
    Stopping      : <type>BOOL</type> ;
    StandStill    : <type>BOOL</type> ;
    DiscreteMotion : <type>BOOL</type> ;
    ContinuousMotion : <type>BOOL</type> ;
    SynchronizedMotion : <type>BOOL</type> ;
    Homing        : <type>BOOL</type> ;
    Initialization : <type>BOOL</type> ;
    CostantVelocity : <type>BOOL</type> ;
    Accelerating  : <type>BOOL</type> ;
    Decelerating  : <type>BOOL</type> ;
END_VAR
    
```

Parameter

Axis

Value of type AXIS_REF
Axis reference.

Enable

Value of type BOOL

As long as 'Enable' is true, it continuously gets the value of the parameter.

Done

Value of type BOOL

Flag that is set to TRUE when the read actual position value is available.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#)).

Errorstop

Value of type BOOL

Is TRUE when the axis is in [ERRORSTOP](#) state.

Stopping

Value of type BOOL

Flag that is set to TRUE when the axis is in [STOPPING](#) or [EMERGENCYSTOPPING](#) state.

StandStill

Value of type BOOL

Flag that is set to TRUE when the axis is in [STANDSTILL](#) state.

DiscreteMotion

Value of type BOOL

Flag that is set to TRUE when the axis is executing a [DISCRETEMOTION](#) movement. (For example a MC_MoveAbsolute)

ContinuosMotion

Value of type BOOL

Flag that is set to TRUE when the axis is executing a [CONTINUOUSMOTION](#) movement (for example a MC_MoveCustom or MC_MoveVelocity)

SynchronizedMotion

Value of type BOOL

Flag that is set to TRUE when the axis is executing a SYNCHRONIZED movement. Actually this output is always FALSE.

Homing

Value of type BOOL

Flag that is set to TRUE when the axis is executing an *HOMING* procedure.

Initialization

Value of type BOOL

Flag that is set to TRUE while the axis program is executing the *INITIALIZATION* procedure of the axis.

CostantVelocity

Value of type BOOL

Flag that is set to TRUE when the axis is moving with a constant theoretical velocity.

Accelerating

Value of type BOOL

It is not managed.

Decelerating

Value of type BOOL

It is not managed.

Description

This function block returns, in detail, the status of the axis according to the motion that is currently in progress. See *Section 4.2.1, "Axis status"* paragraph.

MC_Reset

It recovers the axis from the *ERRORSTOP* state.

Synopsis

```
FUNCTION_BLOCK MC_Reset

VAR_IN_OUT
  Axis          : <type>AXIS_REF</type>;
END_VAR
VAR_INPUT
  Execute       : <type>BOOL</type> ;
END_VAR
VAR_OUTPUT
  Done          : <type>BOOL</type> ;
  Error         : <type>BOOL</type> ;
  ErrorID      : <type>DINT</type> ;
END_VAR
```

Parameter

Axis

Value of type `AXIS_REF`
Axis reference.

Execute

Value of type `BOOL`
At the rising edge it resets the axis.

Done

Value of type `BOOL`
STANDSTILL state is reached.

Error

Value of type `BOOL`

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#)).

Description

This function block makes the transition from the state *ERRORSTOP* to *STANDSTILL* by resetting all internal axis-related errors. If it is tried to execute this function block when the axis state is not *ERRORSTOP*, nothing happens.

```
(* it reads the status of the axis *)
MC_Status_inst(Axis:= Axis, Enable := 1);
IF (MC_Status_inst.ErrorStop = 1) THEN
    (* when there is an error it reset the Execute of MC_Reset_inst *)
    MC_Reset_inst(Axis:=Axis, Execute:=0) ;
END_IF;
(* if there is an error and the input AX_ResetCommand is TRUE then the recovery error procedure is
started *)
MC_Resetinst(Axis:=Axis, Execute:=(MC_Status_inst.ErrorStop AND AX_ResetCommand)) ;
```

When the axis has an error, the *ERRORSTOP* output of *MC_ReadStatus* is TRUE. When *AX_ResetCommand* is TRUE the FB *MC_Reset* recovers the error. If *AX_ResetCommand* is always TRUE this program automatically recovers the axis from an *ERRORSTOP* state.

Example 4.2. Reset error procedure

MC_Start

It starts the axes program management.

Synopsis

```
FUNCTION_BLOCK <function>MC_Start</function>
  VAR_INPUT
    Execute          : <type>BOOL</type> ;
  END_VAR

  VAR_OUTPUT
    Done             : <type>BOOL</type> ;
    CommandAborted  : <type>BOOL</type> ;
    Error            : <type>BOOL</type> ;
    ErrorID         : <type>DINT</type> ;
  END_VAR
```

Parameter

Execute

Value of type BOOL

At the rising edge it starts the management of the axis.

Done

Value of type BOOL

Flag that is set to TRUE when the read actual position value is available.

CommandAborted

Value of type BOOL

Flag that is set to TRUE if the command is aborted by another motion command.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see *Table A.1*).

Description

This function block starts the management of the axis. Before the program launches a movement it is necessary the output Done of MC_Start is TRUE.

4.3. Peripherals management (IO_)

This paragraph describes the function blocks that are necessary to manage the peripherals of the drive: digital inputs, digital outputs, encoder, analog input and output.

4.3.1. Encoder management

In this paragraph there are the descriptions of:

- data type ENC_REF, used in a program that defines the encoder to be managed;
- the function blocks dedicated to the encoder management.

ENC_REF

It is the encoder reference.

Synopsis

```
TYPE Enc_Ref :  
  STRUCT  
    Num    : INT;  
  END_STRUCT;  
END_TYPE
```

Elements

num

Value of type INT

It is an internal number that represents the encoder to be used.

Description

This Data Type is the encoder reference. It is used to declare an encoder in the project. Before to call this function block it is important to initialize the num with the correct value related to the encoder that has to be managed. For this purpose, it is strongly recommended to use the already defined constants:

- IO_REF_ENC_AXIS (1) : it is the encoder used by the axis.



Note

This encoder is equal to IO_REF_ENC_AX_FEEDBACK when the feedback loop is closed, else it is a simulated value (=command position).

- IO_REF_ENC_AX_FEEDBACK (2) : it is the feedback encoder;
- IO_REF_ENC_AX_COMMAND (3) : it is the feedback encoder commanded position¹;

¹This reference is available only on BD drives.

- IO_REF_ENC_AX_FOLLOW_ERR (5) : it is the feedback encoder position including the position following error²;
- IO_REF_ENC_AUXILIARY (11) : it is the auxiliary encoder;

**Note**

It can be selected equal to IO_REF_ENC_AUX_REAL or IO_REF_ENC_AUX_VIRTUAL;

- IO_REF_ENC_AUX_REAL (12) : it is the real auxiliary encoder;
- IO_REF_ENC_AUX_VIRTUAL (13) : it is the internal auxiliary encoder. It is a simulated value.
- IO_REF_ENC_AUX_FIELDBUS (14) : it is the fieldbus auxiliary encoder. It cannot be used for the capture function.

²This reference is available only on BD drives.

CMP_REF

It is the position comparator reference.

Synopsis

```
TYPE Cmp_Ref :  
  STRUCT  
    Num      : INT;  
    EncRef   : Enc_Ref;  
  END_STRUCT;  
END_TYPE
```

Elements

num

Value of type INT

It is an internal number that represents the comparator number.

EncRef

Value of type Enc_Ref

It defines which encoder reference has to be latched. The possible usable encoders are:

- IO_REF_ENC_AXIS (1);
- IO_REF_ENC_AX_COMMAND (3);
- IO_REF_ENC_AUXILIARY (11);

Description

This Data Type is the comparator number. It is used to define the encoder to be related to the comparator. Before to call the function block it is important to initialize the num with the correct value related to the comparator it has to manage. For this purpose, it is strongly recommended to use the already defined constants:

- IO_REF_CMP_0 (0) : it is the peripheral 0 for the position comparison;

- IO_REF_CMP_1 (1) : it is the peripheral 1 for the position comparison;

**Note**

The position comparator cannot use all the encoder references (see [ENC_REF](#)), but only the numbers (1), (3) and (11).

IO_EncGetStatus

It shows the encoder status.

Synopsis

```
FUNCTION_BLOCK Io_EncGetStatus
VAR_IN_OUT
  Reference    : Enc_Ref;
END_VAR
VAR_INPUT
  Enable      : BOOL;
END_VAR
VAR_OUTPUT
  Error       : BOOL;
  ErrorID     : DINT;
  Init       : BOOL;
  Ready      : BOOL;
  Alarm      : BOOL;
  AlarmCode  : DINT;
  AlarmBitCode: DINT;
END_VAR
```

Parameter

Reference

Value of type ENC_REF

Encoder reference.

Enable

Value of type BOOL

As long as 'Enable' is true, it continuously gets the value of the Encoder status.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see *Table A.1*).

Init

Value of type BOOL

As long as it is TRUE, the peripheral is in initialization state.

Ready

Value of type BOOL

As long as it is TRUE, the peripheral is ready.

Alarm

Value of type BOOL

Flag that is set to TRUE when the peripheral is in alarm.

AlarmCode

Value of type DINT

When the encoder has an alarm, it shows the alarm code.

AlarmBitCode

Value of type DINT

When the encoder has an alarm, it shows the alarm code.

Description

It shows the status of the encoder. Its possible statuses are:

- **Init** : the encoder is executing its initialization;
- **Ready** : the encoder is ready to be used;
- **Alarm** : the encoder has an alarm.

When the program starts, the encoder is in Init state and then automatically passes to Ready. If the encoder does not work, it goes in alarm state.

Io_EncManager

It manages the encoder.

Synopsis

```
FUNCTION_BLOCK Io_EncManager
VAR_IN_OUT
  Reference    : Enc_Ref;
END_VAR
VAR_INPUT
  Enable      : BOOL;
  AlarmResume : BOOL;
END_VAR
VAR_OUTPUT
  Error       : BOOL;
  ErrorID     : DINT;
  Init        : BOOL;
  PreReady    : BOOL;
  Ready       : BOOL;
  Alarm       : BOOL;
  AlarmCode   : DINT;
  AlarmBitCode: DINT;
END_VAR
```

Parameter

Reference

Value of type ENC_REF

Encoder reference.

Enable

Value of type BOOL

As long as 'Enable' is true, it continuously gets the value of the encoder status.

AlarmResume

Value of type BOOL

When the encoder is in alarm state, it executes a resume of the alarm. It is necessary to recover the encoder.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#)).

Init

Value of type BOOL

As long as it is TRUE, the encoder is in initialization state.

PreReady

Value of type BOOL

As long as it is TRUE, the encoder is in PreReady state. PreReady is an internal state of the initialization procedure of the encoder.

Ready

Value of type BOOL

Flag that is set to TRUE when the encoder is ready.

Alarm

Value of type BOOL

Flag that is set to TRUE when the encoder is in alarm state.

AlarmCode

Value of type DINT

When the encoder has an alarm, it shows the alarm code.

AlarmBitCode

Value of type DINT

When the encoder has an alarm, it shows the alarm bit code.

Description

It manages the encoder. It allows to recover the encoder from an alarm state.

Io_EncReadPosition

It reads the encoder position.

Synopsis

```
FUNCTION_BLOCK Io_EncReadPosition
VAR_IN_OUT
  Reference    : Enc_Ref;
END_VAR
VAR_INPUT
  Enable      : BOOL;
  RefValidation : BOOL;
END_VAR
VAR_OUTPUT
  Error       : BOOL;
  ErrorID     : DINT;
  Position    : DINT;
  Velocity    : DINT;
  Valid       : BOOL;
  Forced      : BOOL;
  RefValid    : BOOL;
END_VAR
```

Parameter

Reference

Value of type ENC_REF
Encoder reference.

Enable

Value of type BOOL
As long as 'Enable' is true, it continuously reads the encoder position.

RefValidation

Value of type BOOL
It validates the encoder position.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#)).

Position

Value of type DINT

Encoder position.

Velocity

Value of type DINT

Encoder velocity.

Valid

Value of type BOOL

As long as 'Enable' is true, the position and the velocity values are valid.

Forced

Value of type BOOL

It is not used.

RefValid

Value of type BOOL

Flag that is set to TRUE when the position has been validate. After that RefValid has been set to TRUE, this output shows if the position of the encoder is still coherent, or some problem is happened and so the position is not coherent.

Description

It reads the encoder position.

Io_EncReadPositionOnPort

It reads the encoder position.

Synopsis

```
FUNCTION_BLOCK Io_EncReadPositionOnPort
VAR_IN_OUT
    Reference    : Enc_Ref;                (*[IO reference]*)
END_VAR
VAR_INPUT
    Enable       : BOOL;
    RefValidation : BOOL;
END_VAR
VAR_OUTPUT
    Done         : BOOL;
    Active       : BOOL;
    Error        : BOOL;
    ErrorID      : DINT;
    Position     : DINT;
    Velocity     : DINT;
    Valid        : BOOL;
    Forced       : BOOL;
    RefValid     : BOOL;
END_VAR
```

Parameter

Reference

Value of type ENC_REF
Encoder reference.

Enable

Value of type BOOL
As long as 'Enable' is true, it continuously reads the encoder position.

RefValidation

Value of type BOOL

It validates the encoder position.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#)).

Position

Value of type DINT

Encoder position.

Velocity

Value of type DINT

Encoder velocity.

Valid

Value of type BOOL

As long as 'Enable' is true, the position and the velocity values are valid.

Forced

Value of type BOOL

It is not used.

RefValid

Value of type BOOL

Flag that is set to TRUE when the position has been validate. After that RefValid has been set to TRUE, this output shows if the position of the encoder is still coherent, or some problem is happened and so the position is not coherent.

Description

It reads the encoder position. This function block works at the same way of Io_EncReadPosition.

Io_EncTriggerEvent

It configures the trigger event zero mark for the capture function.

Synopsis

```
FUNCTION_BLOCK Io_EncTriggerEvent
VAR_IN_OUT
  Reference      : Enc_Ref;
END_VAR
VAR_INPUT
  Execute       : BOOL;
  Abort         : BOOL;
  TrgStart      : BOOL;
  EdgeType      : BOOL;
  OneShot       : BOOL;
  NEventDriven  : BYTE;
END_VAR
VAR_OUTPUT
  Done          : BOOL;
  Active        : BOOL;
  CommandAborted : BOOL;
  Error         : BOOL;
  ErrorID       : DINT;
  EnablingCapture : BOOL;
  TrgEventHandle : DINT;
END_VAR
```

Parameter

Reference

Value of type ENC_REF

Encoder reference.

- IO_REF_ENC_AX_FEEDBACK (2);
- IO_REF_ENC_AUX_REAL (12)¹.

¹This reference is valid only for BD drives.

Execute

Value of type BOOL

At the rising edge it starts to configure the capture on the zero mark of the encoder.

Abort

Value of type BOOL

At the rising edge it executes an abort of the capture procedure.

TrgStart

Value of type BOOL

At the rising edge it launches the capture function. This command is executed when the configuration is complete.

EdgeType

Value of type BOOL

Edge of the Zero Mark detection that triggers the capture

0 : falling edge;

1 : rising edge.

OneShot

Value of type BOOL

if it is TRUE, then the capture function is executed only once. If it is FALSE, then it is automatically restarted.

NEventDriven

Value of type INT

Selector of the Capture Peripheral:

- 0 : the drive automatically selects the first Capture Peripheral that is not in use (free);
- SYS_EVENT_CAPTURE_A (6) : it selects the Capture Peripheral A;
- SYS_EVENT_CAPTURE_B (7) : it selects the Capture Peripheral B;

Done

Value of type BOOL

The position capture on the Zero Mark detection has been executed.

Active

Value of type BOOL

It is waiting the Zero Mark detection.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the Function Block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#)).

EnablingCapture

Value of type BOOL

When this Function Block completes the configuration of the event, this output is set to TRUE. This output is an input for the Function Block [Io_EncEventCaptureValue](#).

TrgEventHandle

Value of type DINT

It is an internal identification of the event. The Function Block itself calculates it.

NEventActivated

Value of type INT

It indicates which Capture Peripheral has been activated by the Function Block:

- SYS_EVENT_CAPTURE_A (6) : Capture Peripheral A;
- SYS_EVENT_CAPTURE_B (7) : Capture Peripheral B;

Description

This Function Block must be used with [Io_EncEventCaptureValue](#). They configure and manage the capture function of the encoder position when the encoder position passes through the zero mark. The Function Block inputs must be used by following this sequence:

1. to prepare the inputs EdgeType, OneShot;
2. when the Function Block sees a positive edge on the input Execute, it starts to configure the capture function into the drive and also it calculates the TrgEventHandle. When it finishes its internal operations, the output EnablingCapture is set to TRUE. This output is used by the Function Block [Io_EncEventCaptureValue](#) to complete the configuration of the capture function;
3. when this second Function Block finishes its operations, the input TrgStart is set to TRUE. At this moment the capture function starts. The output Active is set to TRUE;

4. when the capture event happens then the output Active returns to FALSE and the output Done is set to TRUE.

See also *Section 4.5.2, "Capture example"*.

Io_EncEventCaptureValue

It completes the configurations of the capture function. It defines which is the value to be captured.

Synopsis

```
FUNCTION_BLOCK Io_EncEventCaptureValue
VAR_IN_OUT
  Reference      : Enc_Ref;
END_VAR
VAR_INPUT
  Execute       : BOOL;
  Abort         : BOOL;
  TrgEventHandle: DINT;
  ValueType     : INT;
END_VAR
VAR_OUTPUT
  Done          : BOOL;
  Active        : BOOL;
  CommandAborted: BOOL;
  Error         : BOOL;
  ErrorID       : DINT;
  CaptureEnabled: BOOL;
  CapturedValue : DINT;
END_VAR
```

Parameter

Reference

Value of type ENC_REF

It defines which encoder reference has to be consider as capture reference. The possible usable encoders are:

- IO_REF_ENC_AXIS (1);
- IO_REF_ENC_AX_FEEDBACK (2);
- IO_REF_ENC_AX_FOLLOW_ERR (5);

- IO_REF_ENC_AUXILIARY (11);
- IO_REF_ENC_AUX_REAL (12)¹;
- IO_REF_ENC_AUX_VIRTUAL (13)²;

Execute

Value of type BOOL

At the rising edge it configures the value for the capture function. The value is the ValueType of the encoder reference (Reference).

Abort

Value of type BOOL

At the rising edge it executes the abort of the capture procedure.

TrgEventHandle

Value of type DINT

It is the internal identification of the event that defines when the capture function has to capture the reference encoder position. This value is calculated by the Function Block which configures the event. See [Io_EncTriggerEvent](#), [Io_DInpTriggerEvent](#).

ValueType

Value of type INT

It defines which type of value the capture function has to capture. The default value is the position of the reference encoder selected. It is not allowed to change it.

Done

Value of type BOOL

The position has been captured.

Active

Value of type BOOL

As long as 'Active' is true, the Function Block keeps waiting the happening of the event.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

¹Available only for BD series.

²Available only for BD series, except for IBD flange 60.

Error identification code (see [Table A.1](#)).

CaptureEnabled

Value of type BOOL

When this function block completes the configuration of the position that the capture function has to capture, this output is set to TRUE. This output is usually connected to the input TrgStart of the function block that configures the event (see [Io_EncTriggerEvent](#) or [Io_DInpTriggerEvent](#)).

CapturedValue

Value of type DINT

When the position has been captured, the output Done is set to TRUE and in this output there is the value of the captured position.

Description

This function block must be used with another one that configures the event for the capture function. (See [Io_EncTriggerEvent](#) or [Io_DInpTriggerEvent](#)).

The capture function is used to capture a position when a particular event happens. To configure the capture function is necessary to define two characteristics:

- the event that causes the capture. The available events are: zero mark ([Io_EncTriggerEvent](#)) and the home digital input ([Io_DInpTriggerEvent](#));
- the position that the function captures when the event happens: this function block is necessary to configure it.

The management for this function block is:

1. when the function block that configures the event sets its EnablingCapture output, then the input Execute of this function block has to detect a rising edge trigger. The input Execute is usually connected to the output EnablingCapture of the [Io_EncTriggerEvent](#) or [Io_DInpTriggerEvent](#). After this rising edge, the function block configures the position to be captured according to the ValueType. When it finishes, the outputs CaptureEnabled and Active are both set to TRUE;
2. the output CaptureEnabled is usually connected to TrgStart of [Io_EncTriggerEvent](#) or [Io_DInpTriggerEvent](#). When TrgStart has a rising edge, the capture function starts. When the event happens and the position has been captured, then the output Done of this function block (Io_EncEventCaptureValue) is set to TRUE and in the output CapturedValue there is the captured value.

See also [Section 4.5.2, "Capture example"](#).

Io_EncComparator

It compares the encoder position.

Synopsis

```
FUNCTION Io_EncComparator
VAR_IN_OUT
  Reference   : Cmp_Ref;
END_VAR
VAR_INPUT
  Enable      : BOOL;
  Direction   : BOOL;
  Position    : DINT;
END_VAR
```

Parameter

Reference

Value of type CMP_REF

Comparator reference

- IO_REF_CMP_0 (0);
- IO_REF_CMP_1 (1).



Note

The encoder references (ENC_REF) that can be linked to the comparator reference (CMP_REF) can be only:

- IO_REF_ENC_AXIS (1);
- IO_REF_ENC_AX_COMMAND (3);
- IO_REF_ENC_AUXILIARY (11).

Enable

Value of type BOOL

Get the value of the status continuously while enabled.

Direction

Value of type BOOL

Defines the positive direction counting of the position comparison. If its value is FALSE the comparator verifies if the reference position is exceeded from lower to higher values. If its value is TRUE the comparator verifies if the reference position is exceeded from higher to lower values.

Position

Value of type DINT

Value of the position to be exceeded.

Description

It sets the position comparator functioning: the encoder reference, the counting direction and the position reference in order to plan the INT PROGRAM through the *SYS_EnEventInt* function.

4.3.2. Digital inputs management

In this paragraph there are the descriptions of:

- data type DINTP_REF, used in a program to define the digital input to be managed;
- the function blocks dedicated to the digital inputs management.

DINP_REF

It is the digital inputs bench reference.

Synopsis

```
TYPE DInp_Ref :  
  STRUCT  
    Num    : INT;  
  END_STRUCT;  
END_TYPE
```

Elements

num

Value of type INT

It is an internal number that represents the digital inputs bench to be used.

Description

This Data Type is the digital inputs bench reference. It is used to declare the digital inputs in the project. Before to call this function block it is important to initialize the num with the correct value related to the digital inputs that has to be managed. For this purpose, it is strongly recommended to use the already defined constants:

- IO_REF_DI_PHYSICAL_0 (1) : first physical bank;
- IO_REF_DI_PHYSICAL_1 (2) : second physical bank¹;

¹This bank is present on SVM and BD series only.

Io_DInpGetStatus

It shows the digital inputs status.

Synopsis

```
FUNCTION_BLOCK Io_DInpGetStatus
VAR_IN_OUT
  Reference   : DInp_Ref;
END_VAR
VAR_INPUT
  Enable     : BOOL;
END_VAR
VAR_OUTPUT
  Error      : BOOL;
  ErrorID    : DINT;
  Init       : BOOL;
  Ready      : BOOL;
  Alarm      : BOOL;
  AlarmCode  : DINT;
  AlarmBitCode: DINT;
END_VAR
```

Parameter

Reference

Value of type DINP_REF
Digital inputs reference.

Enable

Value of type BOOL
As long as 'Enable' is true, it continuously gets the value of the digital inputs status.

Error

Value of type BOOL
Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see *Table A.1*)

Init

Value of type BOOL

Flag that is set to TRUE when the peripheral is in initialization state.

Ready

Value of type BOOL

Flag that is set to TRUE when the peripheral is ready.

Alarm

Value of type BOOL

Flag that is set to TRUE when the peripheral is in alarm.

AlarmCode

Value of type DINT

When the peripheral has an alarm, it shows the alarm code.

AlarmBitCode

Value of type DINT

When the peripheral has an alarm, it shows the alarm code.

Description

It shows the status of the peripheral. The statuses are:

- **Init** : the peripheral is executing its initialization;
- **Ready** : the peripheral is ready to be used;
- **Alarm** : the peripheral has an alarm.

When the program starts, the peripheral is in Init state and then automatically passes to Ready. If the digital inputs do not work, then the peripheral goes in alarm state.

Io_DInpManager

It manages the peripheral digital inputs.

Synopsis

```
FUNCTION_BLOCK Io_DInpManager
VAR_IN_OUT
    Reference    : DInp_Ref;                (*[IO reference]*)
END_VAR
VAR_INPUT
    Enable       : BOOL;
    AlarmResume  : BOOL;
END_VAR
VAR_OUTPUT
    Error        : BOOL;
    ErrorID      : DINT;
    Init         : BOOL;
    PreReady    : BOOL;
    Ready        : BOOL;
    Alarm        : BOOL;
    AlarmCode    : DINT;
    AlarmBitCode: DINT;
END_VAR
```

Parameter

Reference

Value of type DINP_REF

Digital inputs reference.

Enable

Value of type BOOL

As long as 'Enable' is true, it continuously manages the digital inputs.

AlarmResume

Value of type BOOL

When the peripheral is in alarm state, it executes a resume of the alarm. It is necessary to recover the peripheral.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#))

Init

Value of type BOOL

As long as it is TRUE, the peripheral is in initialization state.

PreReady

Value of type BOOL

As long as it is TRUE, the peripheral is in PreReady state. PreReady is an internal state of the initialization procedure of the peripheral.

Ready

Value of type BOOL

As long as it is TRUE, the peripheral is ready.

Alarm

Value of type BOOL

Flag that is set to TRUE when the peripheral is in alarm.

AlarmCode

Value of type DINT

When the peripheral has an alarm, it shows the alarm code.

AlarmBitCode

Value of type DINT

When the peripheral has an alarm, it shows the alarm code.

Description

It manages the peripheral. It permits to recover an alarm in the peripheral.

Io_DInpReadStatus

It reads the status of the digital inputs.

Synopsis

```
FUNCTION_BLOCK Io_DInpReadStatus
VAR_IN_OUT
  Reference    : DInp_Ref;
END_VAR
VAR_INPUT
  Enable      : BOOL;
END_VAR
VAR_OUTPUT
  Error       : BOOL;
  ErrorID     : DINT;
  InpStatus   : BYTE;
  Valid       : BOOL;
  Forced      : BOOL;
END_VAR
```

Parameter

Reference

Value of type DINP_REF
Digital inputs reference.

Enable

Value of type BOOL
As long as 'Enable' is true, it continuously gets the value of the status of the digital inputs.

Error

Value of type BOOL
Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see *Table A.1*).

InpStatus

Value of type BYTE

Digital inputs status when the output Valid is TRUE. Each bit is related to a digital input, according to the bank selected by the DINP_REF.

Valid

Value of type BOOL

Is TRUE when the output InpStatus has a valid value.

Forced

Value of type BOOL

It is not used.

Description

It reads the status of the digital inputs selected in the DINP_REF reference input.

Io_DInpReadStatusOnPort

It reads the status of the digital inputs.

Synopsis

```
FUNCTION_BLOCK Io_DInpReadStatusOnPort
VAR_IN_OUT
  Reference    : DInp_Ref;
END_VAR
VAR_INPUT
  Enable      : BOOL;
END_VAR
VAR_OUTPUT
  Error       : BOOL;
  ErrorID     : DINT;
  InpStatus   : BYTE;
  Valid       : BOOL;
  Forced      : BOOL;
END_VAR
```

Parameter

Reference

Value of type DINP_REF

Digital inputs reference.

Enable

Value of type BOOL

As long as 'Enable' is true, it continuously gets the value of the status of the digital inputs.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see *Table A.1*)

InpStatus

Value of type BYTE

Digital inputs status when the output Valid is TRUE. Each bit is related to a digital input, according to the bank selected by the DINP_REF.

Valid

Value of type BOOL

Is TRUE when the output InpStatus has a valid value.

Forced

Value of type BOOL

It is not used.

Description

It reads the status of the digital inputs selected in the DINP_REF reference input. This function block works at the same way of Io_DInpReadStatus.

Io_DInpTriggerEvent

It configures the capture function trigger event on an edge of a particular digital input.

Synopsis

```

FUNCTION_BLOCK Io_DInpTriggerEvent
VAR_IN_OUT
  Reference    : DInp_Ref;
END_VAR
VAR_INPUT
  Execute      : BOOL;
  Abort        : BOOL;
  TrgStart     : BOOL;
  NBit         : BYTE;
  EdgeType     : BOOL;
  OneShot      : BOOL;
  NEventDriven : BYTE;
END_VAR
VAR_OUTPUT
  Done         : BOOL;
  Active       : BOOL;
  CommandAborted : BOOL;
  Error        : BOOL;
  ErrorID      : DINT;
  EnablingCapture : BOOL;
  TrgEventHandle : DINT;
END_VAR

```

Parameter

Reference

Value of type DINP_REF

Digital inputs reference.

Drive	Reference	NBit	Associated Inputs
SD	IO_REF_DI_PHYSICAL_0	2	In2

Drive	Reference	NBit	Associated Inputs
		3	In3
IBD	IO_REF_DI_PHYSICAL_1	0	In8
		1	In9
NBD	IO_REF_DI_PHYSICAL_0	3	In3
		4	In4
IBD fl.60	IO_REF_DI_PHYSICAL_0	2	In2
		3	In3

Table 4.1. Bit/Input reference

Execute

Value of type BOOL

At the rising edge it starts to configure the trigger event. Its characteristics are in the inputs NBit and EdgeEvent.

Abort

Value of type BOOL

At the rising edge it executes an abort of the capture procedure.

TrgStart

Value of type BOOL

At the rising edge it launches the capture function. This command is executed when the configuration is complete.

NBit

Value of type BYTE

It defines the number of the digital input selected to be the trigger for the capture function. The bit number NBit is related to the bank defined through the DINP_REF reference:

- For Digital input 0..7 the reference bank is IO_REF_DI_PHYSICAL_0, NBit 0..7;
- For Digital input 8..15 the reference bank is IO_REF_DI_PHYSICAL_1, NBit 0..7 (NBit 0 = Dig. input 8, etc.);

EdgeType

Value of type BOOL

0 : falling edge;
1 : rising edge.

OneShot

Value of type BOOL

if it is TRUE, then the capture function is executed only once. If it is FALSE, then it is automatically restarted.

NEventDriven

Value of type BYTE

It is not used.

Done

Value of type BOOL

The capture has been executed.

Active

Value of type BOOL

It is waiting for the event happening.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#)).

EnablingCapture

Value of type BOOL

When this function block completes the configuration of the event, this output is set to TRUE. This output is an input for the function block *Io_EncEventCaptureValue*.

TrgEventHandle

Value of type DINT

It is an internal identification of the event. The function block itself calculates it.

Description

This function block must be used with *Io_EncEventCaptureValue*. They configure and manage the capture of the encoder position function when the event in the digital input happens. The function block inputs has to follow this sequence:

1. it prepares the inputs NBit, EdgeType, OneShot;
2. when the function block detects a positive edge in the input Execute, it starts to configure the capture function into the drive and also to calculate the TrgEventHandle. When it finishes its internal operations, it sets to TRUE the output EnablingCapture. This output is used by the function block *Io_EncEventCaptureValue* to complete the configuration of the capture function;
3. when this second function block finishes its operations, then it sets to TRUE the input TrgStart. In this moment the capture function starts. The output Active is set to TRUE;
4. when the capture has been made, the output Active returns to FALSE and the output Done is set to TRUE.

See also [Section 4.5.2, “Capture example”](#).

4.3.3. Digital outputs management

In this paragraph there are the descriptions of:

- data type DOUT_REF, used in a program to define the digital outputs to be managed;
- the function blocks dedicated to the digital outputs management.

DOUT_REF

It is the digital outputs bench reference.

Synopsis

```
TYPE DOut_Ref :  
  STRUCT  
    Num    : INT;  
  END_STRUCT;  
END_TYPE
```

Elements

num

Value of type INT

It is an internal number that represents the digital outputs bench to be used.

Description

This Data Type is the digital outputs bench reference. It is used to declare the digital outputs in the project. Before to call this function block it is important to initialize the num with the correct value related to the digital outputs that has to be managed. For this purpose, it is strongly recommended to use the already defined constant:

- IO_REF_DO_PHYSICAL_0 (1) : first physical digital output bank;

Io_DOutGetStatus

It shows the digital outputs status.

Synopsis

```

FUNCTION_BLOCK Io_DOutGetStatus
VAR_IN_OUT
    Reference    : DOut_Ref;
END_VAR
VAR_INPUT
    Enable      : BOOL;
END_VAR
VAR_OUTPUT
    Error       : BOOL;
    ErrorID     : DINT;
    Init        : BOOL;
    Ready       : BOOL;
    Alarm       : BOOL;
    AlarmCode   : DINT;
    AlarmBitCode: DINT;
END_VAR
    
```

Parameter

Reference

Value of type DOUT_REF
Digital outputs reference

Enable

Value of type BOOL
As long as 'Enable' is true, it continuously gets the value of the digital outputs status.

Error

Value of type BOOL
Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see *Table A.1*).

Init

Value of type BOOL

As long as it is TRUE, the peripheral is in initialization state.

Ready

Value of type BOOL

As long as it is TRUE, the peripheral is ready.

Alarm

Value of type BOOL

As long as it is TRUE, the peripheral is in alarm.

AlarmCode

Value of type DINT

When the peripheral has an alarm, it shows the alarm code.

AlarmBitCode

Value of type DINT

When the peripheral has an alarm, it shows the alarm code.

Description

It shows the status of the peripheral. The statuses are:

- **Init** : the peripheral is executing its initialization;
- **Ready** : the peripheral is ready to be used;
- **Alarm** : the peripheral has an alarm.

When the program starts, the peripheral is in Init state and then automatically passes to Ready. If the digital outputs do not work, then the peripheral goes in alarm state.

Io_DOutManager

It manages the peripheral digital outputs.

Synopsis

```
FUNCTION_BLOCK Io_DOutManager
VAR_IN_OUT
  Reference   : DOut_Ref;
END_VAR
VAR_INPUT
  Enable      : BOOL;
  AlarmResume : BOOL;
END_VAR
VAR_OUTPUT
  Error       : BOOL;
  ErrorID     : DINT;
  Init        : BOOL;
  PreReady   : BOOL;
  Ready       : BOOL;
  Alarm       : BOOL;
  AlarmCode   : DINT;
  AlarmBitCode: DINT;
END_VAR
```

Parameter

Reference

Value of type DOUT_REF
Digital outputs reference.

Enable

Value of type BOOL
As long as 'Enable' is true, it continuously manages the digital outputs.

AlarmResume

Value of type BOOL

When the peripheral is in alarm state, it executes a resume of the alarm. It is necessary to recover the peripheral.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#)).

Init

Value of type BOOL

Flag that is set to TRUE when the peripheral is in initialization state.

PreReady

Value of type BOOL

Flag that is set to TRUE when the peripheral is PreReady. PreReady is an internal state of the initialization procedure of the peripheral.

Ready

Value of type BOOL

As long as it is TRUE, the peripheral is ready.

Alarm

Value of type BOOL

Flag that is set to TRUE when the peripheral is in alarm.

AlarmCode

Value of type DINT

When the peripheral has an alarm, it shows the alarm code.

AlarmBitCode

Value of type DINT

When the peripheral has an alarm, it shows the alarm code.

Description

It manages the peripheral. It permits to recover an alarm in the peripheral.

Io_DOutWriteStatus

It writes the status of the digital outputs.

Synopsis

```

FUNCTION_BLOCK Io_DOutWriteStatus
VAR_IN_OUT
    Reference    : DOut_Ref;
END_VAR
VAR_INPUT
    Enable      : BOOL;
    OutStatus   : BYTE;
END_VAR
VAR_OUTPUT
    Error       : BOOL;
    ErrorID     : DINT;
    Valid       : BOOL;
    Forced      : BOOL;
END_VAR
    
```

Parameter

Reference

Value of type DOUT_REF

Digital outputs reference.

Enable

Value of type BOOL

As long as 'Enable' is true, it continuously writes the value of the outputs status.

OutStatus

Value of type BYTE

It is the value that has to be written on the digital outputs status.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#)).

Valid

Value of type BOOL

Is TRUE when the digital outputs are valid.

Forced

Value of type BOOL

It is not used.

Description

It writes the value of the input OutStatus on the digital outputs that are selected by the Reference.

Io_DOutWriteStatusOnPort

It writes the status of the digital outputs.

Synopsis

```

FUNCTION_BLOCK Io_DOutWriteStatusOnPort
VAR_IN_OUT
    Reference    : DOut_Ref;
END_VAR
VAR_INPUT
    Enable      : BOOL;
    OutStatus   : BYTE;
END_VAR
VAR_OUTPUT
    Error       : BOOL;
    ErrorID     : DINT;
    Valid       : BOOL;
    Forced      : BOOL;
END_VAR
    
```

Parameter

Reference

Value of type DOUT_REF

Digital outputs reference.

Enable

Value of type BOOL

As long as 'Enable' is true, it continuously writes the value of the outputs status.

OutStatus

Value of type BYTE

It is the value that has to be written on the digital outputs status.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see *Table A.1*).

Valid

Value of type BOOL

Is TRUE when the digital outputs are valid.

Forced

Value of type BOOL

It is not used.

Description

It writes the value of the input OutStatus on the digital outputs that are selected by the Reference.

4.3.4. Analog input management

In this paragraph there are the descriptions of:

- data type AINP_REF used in a program to define the analog inputs to be managed;
- the function blocks dedicated to the analog inputs management.

AINP_REF

It is the analog input reference.

Synopsis

```
TYPE
  AXIS_REF : STRUCT
    num      : <type>INT</type>;
  END_STRUCT;
END_TYPE
```

Elements

num

Value of type INT

It is an internal number that represents the analog input bench to be used.

Description

This Data Type is the analog input bench reference. It is used to declare the analog input in the project. Before to call this function block it is important to initialize the num with the correct value according to the analog input it has to manage. For this purpose, it is strongly recommended to use the already defined constants:

- IO_REF_AI_PHYSICAL_0 (1) : physical analog input;
- IO_REF_AI_TORQUE (9) : torque monitor;
- IO_REF_AI_I2T (10) : I2T monitor;

Io_AInpGetStatus

It shows the analog input status.

Synopsis

```
FUNCTION_BLOCK Io_AInpGetStatus
VAR_IN_OUT
  Reference : AInp_Ref;
END_VAR
VAR_INPUT
  Enable : BOOL;
END_VAR
VAR_OUTPUT
  Error : BOOL;
  ErrorID : DINT;
  Init : BOOL;
  Ready : BOOL;
  Alarm : BOOL;
  AlarmCode : DINT;
  AlarmBitCode: DINT;
END_VAR
```

Parameter

Reference

Value of type AINP_REF
Analog inputs reference.

Enable

Value of type BOOL
As long as 'Enable' is true, it continuously gets the value of the status of the analog input.

Error

Value of type BOOL
Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see *Table A.1*)

Init

Value of type BOOL

As long as it is TRUE, the peripheral is in initialization state.

Ready

Value of type BOOL

As long as it is TRUE, the peripheral is ready.

Alarm

Value of type BOOL

Flag that is set to TRUE when the peripheral is in alarm.

AlarmCode

Value of type DINT

When the peripheral has an alarm, it shows the alarm code.

AlarmBitCode

Value of type DINT

When the peripheral has an alarm, it shows the alarm code.

Description

It shows the status of the peripheral. The statuses are:

- **Init** : the peripheral is executing its initialization;
- **Ready** : the peripheral is ready to be used;
- **Alarm** : the peripheral has an alarm.

When the program starts, the peripheral is in Init state and then automatically passes to Ready. If the analog input does not work, then the peripheral goes in alarm state.

Io_AInpManager

It manages the peripheral analog input.

Synopsis

```
FUNCTION_BLOCK Io_AInpManager
VAR_IN_OUT
    Reference    : AInp_Ref;                (*[IO reference]*)
END_VAR
VAR_INPUT
    Enable      : BOOL;
    AlarmResume : BOOL;
END_VAR
VAR_OUTPUT
    Error       : BOOL;
    ErrorID     : DINT;
    Init        : BOOL;
    PreReady    : BOOL;
    Ready       : BOOL;
    Alarm       : BOOL;
    AlarmCode   : DINT;
    AlarmBitCode: DINT;
END_VAR
```

Parameter

Reference

Value of type AINP_REF
Analog input reference.

Enable

Value of type BOOL
As long as 'Enable' is true, it continuously gets the value of the analog input peripheral status.

AlarmResume

Value of type BOOL

When the peripheral is in alarm state, it executes a resume of the alarm. It is necessary to recover the peripheral.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see [Table A.1](#))

Init

Value of type BOOL

As long as it is TRUE, the peripheral is in initialization state.

PreReady

Value of type BOOL

As long as it is TRUE, the peripheral is in PreReady state. PreReady is an internal state of the initialization procedure of the peripheral.

Ready

Value of type BOOL

As long as it is TRUE, the peripheral is ready.

Alarm

Value of type BOOL

As long as it is TRUE, the peripheral is in alarm.

AlarmCode

Value of type DINT

When the peripheral has an alarm it shows the alarm code

AlarmBitCode

Value of type DINT

When the peripheral has an alarm it shows the alarm code.

Description

It manages the peripheral. It permits to recover an alarm occurred in the peripheral.

Io_AInpReadValue

It reads the value of the analog input.

Synopsis

```
FUNCTION_BLOCK Io_AInpReadValue
VAR_IN_OUT
  Reference    : AInp_Ref;
END_VAR
VAR_INPUT
  Enable       : BOOL;
END_VAR
VAR_OUTPUT
  Error        : BOOL;
  ErrorID      : DINT;
  Value        : DINT;
  Valid        : BOOL;
  Forced       : BOOL;
END_VAR
```

Parameter

Reference

Value of type AINP_REF

Analog input reference.

Enable

Value of type BOOL

As long as 'Enable' is true, it continuously reads the value of the analog input.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification (see *Table A.1*).

Value

Value of type DINT

Analog input value when the output Valid is TRUE.

Valid

Value of type BOOL

If it is TRUE it means that the parameter Value has a valid value.

Forced

Value of type BOOL

It is not used

Description

It reads the value of the analog input selected in the reference.

Io_AInpReadValueOnPort

It reads the value of the analog input.

Synopsis

```
FUNCTION_BLOCK Io_AInpReadValueOnPort
VAR_IN_OUT
  Reference    : AInp_Ref;
END_VAR
VAR_INPUT
  Enable       : BOOL;
END_VAR
VAR_OUTPUT
  Error        : BOOL;
  ErrorID      : DINT;
  Value        : DINT;
  Valid        : BOOL;
  Forced       : BOOL;
END_VAR
```

Parameter

Reference

Value of type AINP_REF

Analog input reference.

Enable

Value of type BOOL

As long as 'Enable' is true, it continuously gets the value of the analog input.

Error

Value of type BOOL

Flag that is set to TRUE when an error has occurred within the function block.

ErrorID

Value of type DINT

Error identification code (see *Table A.1*).

Value

Value of type DINT

Analog input value when the output Valid is TRUE.

Valid

Value of type BOOL

If it is TRUE it means that the parameter Value has a valid value.

Forced

Value of type BOOL

It is not used

Description

It reads the value of the analog input selected in the reference.

4.4. Utility library (Ut_)

This paragraph describes the functions of the utility library. The library contains mathematical functions and features for the management of solar.



Note

These functionalities are not supported in the IBD.

4.4.1. Functions and function blocks list

Ut_ArcCos

It calculates the angle in degrees with resolution E^{-6} .

Synopsis

```
FUNCTION Ut_ArcCos : DINT
  VAR_INPUT
    Cosine : DINT ;
  END_VAR
```

Return value

Value of type DINT

The angle in degrees with resolution E^{-6} ($1000000 = 1$).

Parameter

Cosine

Value of type DINT

It is the cosine of the angle with resolution E^{-8} ($100000000 = 1$). Its range starts from -100000000 to $+100000000$. The value is saturated when it exceeds the range.

Description

This function calculates the angle in degrees with resolution E^{-6} .

Ut_Cos

It calculates the cosine of the angle with resolution E^{-8} .

Synopsis

```
FUNCTION Ut_Cos : DINT
  VAR_INPUT
    Angle : DINT ;
  END_VAR
```

Return value

Value of type DINT

The cosine of the angle with resolution E^{-8} .

Parameter

Angle

Value of type DINT

It is the angle in degree with resolution E^{-6} . Any value is valid.

Description

This function calculates the cosine of the angle with resolution E^{-8} .

Ut_ArcSin

It calculates the angle in degrees with resolution E^{-6} .

Synopsis

```
FUNCTION Ut_ArcSin : DINT
  VAR_INPUT
    Sine : DINT ;
  END_VAR
```

Return value

Value of type DINT

The angle in degrees with resolution E^{-6} ($1000000 = 1$).

Parameter

Sine

Value of type DINT

It is the sine of the angle with resolution E^{-8} ($100000000 = 1$). Its range starts from -100000000 to $+100000000$. The value is saturated when it exceeds the range.

Description

This function calculates the angle in degrees with resolution E^{-6} .

Ut_Sin

It calculates the sine of the angle with resolution E^{-8} .

Synopsis

```
FUNCTION Ut_Sin : DINT
  VAR_INPUT
    Angle : DINT ;
  END_VAR
```

Return value

Value of type DINT

The sine of the angle with resolution E^{-8} .

Parameter

Angle

Value of type DINT

It is the angle in degree with resolution E^{-6} . Any value is valid.

Description

This function calculates the sine of the angle with resolution E^{-8} .

Ut_MTracky

It calculates the angle of the single axis tracker (M-Tracky type).

Synopsis

```
FUNCTION Ut_MTracky : BOOL
  VAR_INPUT
    Altitude : INT ;
    AxisAngle : INT ;
    Azimuth : INT ;
  END_VAR
  VAR_OUTPUT
    Result : INT ;
  END_VAR
```

Return value

Value of type BOOL

If 0 then there are no errors.

Parameter

Altitude

Value of type INT

It is the altitude in hundredths of a degree. Its range starts from -18000 to +18000.

AxisAngle

Value of type INT

It is the angle of rotation axis in hundredths of a degree. Its range starts from -18000 to +18000.

Azimuth

Value of type INT

It is the azimuth in hundredths of a degree. Its range starts from -18000 to +18000.

Result

Value of type INT

The angle of the single axis tracker (M-Tracky type) in hundredths of a degree.

Description

This function calculates the angle of the single axis tracker (M-Tracky type).

Ut_SolarPosition

It calculates the position of the sun.

Synopsis

```
FUNCTION Ut_SolarPosition : BOOL
  VAR_INPUT
    DayOfYear : INT ;
    MinOfDay   : INT ;
    Latitude   : INT ;
  END_VAR
  VAR_OUTPUT
    Altitude  : INT ;
    Azimuth   : INT ;
  END_VAR
```

Return value

Value of type BOOL

If 0 then there are no errors.

Parameter

DayOfYear

Value of type INT

It is the days of the year. Its range starts from 1 to 366.

MinOfDay

Value of type INT

It is the minute of the day. Its range starts from 0 to 1439.

Latitude

Value of type INT

It is the latitude in hundredths of a degree. Its range starts from -9000 to +9000.

Altitude

Value of type INT

Altitude in hundredths of a degree.

Azimuth

Value of type INT

Azimuth in hundredths of a degree.

Description

This function calculates the position of the sun.

Ut_TrueSolarTime

It calculates the local solar time in minutes.

Synopsis

```
FUNCTION Ut_TrueSolarTime : BOOL
  VAR_INPUT
    DayOfYear : INT ;
    MinOfDay  : INT ;
    Longitude : INT ;
  END_VAR
  VAR_OUTPUT
    TrueSolarTime : INT ;
  END_VAR
```

Return value

Value of type BOOL

If 0 then there are no errors.

Parameter

DayOfYear

Value of type INT

It is the day of the year. Its range starts from 1 to 366.

MinOfDay

Value of type INT

It is the minute of the day. Its range starts from 0 to 1439.

Longitude

Value of type INT

It is the longitude in hundredths of a degree. Its range starts from -18000 to +18000.

TrueSolarTime

Value of type INT

Local solar time in minutes.

Description

This function calculates the local solar time in minutes.

4.5. Examples

There are three examples.

4.5.1. Axis management

A simple example of an axis management:

1. **ENABLE** : when the drive becomes **ReadyToSwitchOn** the MC_Power_inst is set to TRUE, then the drive is **SwitchOn**.
2. **RESET** : when the axis has an error and Reset = TRUE, the MC_Reset recovers the error;
3. **HOMING** : it writes the velocities for the homing procedure and then it executes the homing movement;
4. **MOVEMENT** : it continuously moves an axis from position Move1Pos to Move2Pos.

```
(* GLOBAL variables declarations *)VAR_GLOBAL(* STRUCT *)
Axis          : AXIS_REF ;

(* instances of the function block *)
MC_DriveStatus_inst : MC_ReadDriveStatus ;
MC_Status_inst      : MC_ReadStatus ;
MC_Power_inst       : MC_Power ;

(* DINT *)
ActPos          : DINT ;
ActVel          : DINT ;
CmdPos          : DINT ;
CmdVel          : DINT ;
VelFindMicro    : DINT := -4096 ;
VelOutMicro     : DINT := 2048 ;
HomePosition    : DINT := 0 ;
VelJog          : DINT := 8192 ;
Move1Pos        : DINT := 81920 ;
```

```
Move1Vel      : DINT := 4096 ;
Move1Acc      : DINT := 40960 ;
Move2Pos      : DINT := 0 ;
Move2Vel      : DINT := 8192 ;
Move2Acc      : DINT := 40960 ;
EmgDeceleration : DINT := 409600;
```

```
(* INT *)
```

```
HomeMode      : INT := 5 ;
iStep         : INT := 0 ;
```

```
(* BOOL *)
```

```
Reset         : BOOL := FALSE ;
Stop          : BOOL := FALSE ;
StartMove     : BOOL := TRUE ;
Move1Done     : BOOL := 0 ;
Move2Done     : BOOL := FALSE;
err           : BOOL ;
MC_Move1_inst : MC_MoveAbsolute ;
MC_Move2_inst : MC_MoveAbsolute ;
```

```
END_VAR
```

```
PROGRAM main
```

```
VAR
```

```
MC_Start_inst : MC_Start ;
MC_Reset_inst  : MC_Reset ;
MC_ReadActPos_inst : MC_ReadActualPosition ;
MC_ReadPos_inst : MC_ReadCommandPosition ;
MC_Home_inst   : MC_Home;
MC_Stop_inst   : MC_Stop ;
```

```
END_VAR
```

```
Axis.Num := MC_REF_AXIS_MAIN ;
```

```
(* it starts the axis management by IEC program *)
```

```
MC_Start_inst(Execute:=1);
```

```
(* it reads the status of the drive *)
```

```
MC_DriveStatus_inst(Axis:= Axis,
                    Enable := MC_Start_inst.Done);
```

```
(* switch on - switch off *)
```

```
MC_Power_inst(Axis:=Axis,
```



```

        Enable:= MC_DriveStatus_inst.ReadyToSwitchOn) ;

(* it reads the status of the axis *)
MC_Status_inst(Axis:= Axis, Enable := MC_Start_inst.Done);

(* it reads the real and command positions *)
MC_ReadPos_inst(Axis:=Axis, Enable:= TRUE,
                Position=>CmdPos,
                Velocity=>CmdVel);
MC_ReadActPos_inst(Axis:=Axis, Enable:=TRUE,
                   Position=>ActPos,
                   Velocity=>ActVel);

(* it recovers an error *)
IF (MC_Status_inst.ErrorStop = 1) AND Reset THEN
    MC_Reset_inst(Axis:=Axis, Execute:=0) ;
    MC_Reset_inst.Execute := 1 ;
    iStep := 0 ;
END_IF;
MC_Reset_inst(Axis:=Axis) ;

(* it stops the movement *)
IF Stop THEN
    MC_Stop_inst(Axis:=Axis, Execute :=0);
    MC_Stop_inst.Execute := 1 ;
    iStep := 0 ;
END_IF;
MC_Stop_inst(Axis:= Axis,
              Deceleration:= Move1Acc ) ;

CASE iStep OF
    0 :
        (* Homing Procedure *)
        MC_Home_inst.Execute := 0 ;
        MC_Move1_inst.Execute := 0;
        MC_Move2_inst.Execute := 0;
        IF StartMove AND MC_Power_inst.Status THEN
            MC_Home_inst.Execute := 1 ;
            StartMove := 0 ;
            iStep := 1 ;
        END_IF;

    1 :

```

```
(* at the end of the homing it starts the movement *)
IF MC_Home_inst.Done THEN
  MC_Move1_inst.Execute := 1;
  iStep := 2 ;
END_IF;

2 :
IF MC_Move1_inst.Done THEN
  MC_Move1_inst.Execute := 0 ;
  MC_Move2_inst.Execute := 1 ;
ELSE
  IF MC_Move2_inst.Done THEN
    MC_Move1_inst.Execute := 1 ;
    MC_Move2_inst.Execute := 0 ;
  END_IF;
END_IF;
END_CASE ;

(* it manages the homing procedure *)
MC_Home_inst(Axis := Axis,
  Position:=0,
  HomingMode:= 5,
  VelocitySearchSwitch:=VelFindMicro,
  VelocitySearchZero:=VelOutMicro);

(* it executes a continuously
  Move1Pos -> Move2Pos -> Move1Pos
  movement*)
MC_Move1_inst(Axis:=Axis,
  Position:=Move1Pos,
  Velocity:=Move1Vel,
  Acceleration:= Move1Acc,
  Deceleration:=Move1Acc);
MC_Move2_inst(Axis:=Axis,
  Position:=Move2Pos,
  Velocity:=Move2Vel,
  Acceleration:= Move2Acc,
  Deceleration:=Move2Acc);
END_PROGRAM

(* exception management *)
PROGRAM exception
  VAR
```

```

MC_EmergencyStop_inst : MC_EmergencyStop ;
END_VAR
(* it stops the movement because the program has had an error,
   therefore it is stopped *)
MC_EmergencyStop_inst(Axis:= Axis,
    Execute:=0) ;
MC_EmergencyStop_inst(Axis:= Axis,
    Execute:=1,
    Deceleration:= EmgDeceleration ) ;
err:=1 ;
END_PROGRAM
    
```

4.5.2. Capture example

In this section a simple management for the capture of the axis position on the zero mark is shown. To configure the capture function it is necessary to define two characteristics:

- the event that causes the capture. The available events are: zero mark (*Io_EncTriggerEvent*); the home digital input (*Io_DInpTriggerEvent*);
- the position that the function has to capture when the event happens: the function block *Io_EncEventCaptureValue* is necessary to configure it.

In the following example the capture of the axis position on the zero mark is configured and managed. The sequence to manage the capture function is:

1. to prepare the inputs *EdgeType*, *OneShot* in the function block *Io_EncTriggerEvent*;
2. when the function block *Io_EncTriggerEvent* detects a positive edge on the input *Execute*, it starts to configure the capture function into the drive and it also calculates the *TrgEventHandle*. When it finishes its internal operations, it sets to TRUE the output *EnablingCapture*. This output is used from the function block *Io_EncEventCaptureValue* to complete the configuration of the capture function.
3. When this second function block finishes its operations, then it has to set to TRUE the input *TrgStart* of the first one. In this moment the capture function starts. The output *Active* is set to TRUE;
4. When *TrgStart* of *Io_EncTriggerEvent* has a rising edge, the capture function starts. When the event happens and the position is latched, then the output *Done* of *Io_EncEventCaptureValue* is set to TRUE and the captured value is reported in its output *CapturedValue*.

```

VAR_GLOBAL(* STRUCT *)
EventRef      : ENC_REF ;
CptRef        : ENC_REF ;
    
```

```
(* instances of the function blocks *)
```

```
EncTriggerEvent      : Io_EncTriggerEvent;  
EncEventCaptureValue : Io_EncEventCaptureValue;
```

```
END_VAR
```

```
PROGRAM main
```

```
Axis.Num := MC_REF_AXIS_MAIN ;  
EventRef.Num := IO_REF_ENC_AX_FEEDBACK ;    (* event on feedback  
encoder *)  
CptRef.Num := IO_REF_ENC_AXIS ;             (* latch the axis position  
*)
```

```
(* axis management, see previous example *)
```

```
CASE iStep OF
```

```
0 :
```

```
EncTriggerEvent.Execute := 0 ;  
EncEventCaptureValue.Execute := 0;
```

```
IF StartCpt THEN
```

```
EncTriggerEvent.EdgeType := 0 ;  
EncTriggerEvent.OneShot := 1 ;  
EncTriggerEvent.TrgStart := 0;  
EncTriggerEvent.Execute := 1 ;  
iStep := iStep + 1 ;
```

```
END_IF;
```

```
1 :
```

```
EncTriggerEvent.TrgStart := EncEventCaptureValue.CaptureEnabled ;
```

```
IF EncEventCaptureValue.Done THEN
```

```
CptPos := EncEventCaptureValue.CapturedValue ;  
iStep := 0 ;  
StartCpt := 0 ;
```

```
END_IF;
```

```
END_CASE ;
```

```
EncTriggerEvent(Reference:=EventRef);  
EncEventCaptureValue(Reference:=CptRef,  
Execute := EncTriggerEvent.EnablingCapture,
```

```
TrgEventHandle :=  
EncTriggerEvent.TrgEventHandle);  
END_PROGRAM
```

4.5.3. Example of the management of a program safety condition request

This example describes how the IEC program can manage a request to go in its safety condition (*Section 3.1.1, “BD series: Program safety condition procedure”*). The safety condition for an application is a well defined situation. When the application is in this situation the program can stop while the machine is in a safe condition and no problem will happen.

In order to guarantee to the program to arrive in this well defined situation before the program is stopped, it is important to follow this sequence:

1. First of all, in the IEC program the semaphore *IECSafeCondition* has to switch to "yellow" (see *SYS_WrIECSafeCondition(0,1)*). In this way, when the system manager has to execute an action that needs that the program is in safety condition, it has to wait that the program executes the programmed procedure to arrive in the well defined situation;
2. Then it has to check if a request of a safety condition is arrived from the system manager (*SYS_RqsIECSafeCondition*);
3. when it happens: it manages the request, so it executes a sequence of actions to put the program in a well defined situation, that is the safety condition of the application. The procedure reported in the example is to wait until a counter decreases its value from 50000 to 0;
4. at the end, it switches the semaphore to green (*SYS_WrIECSafeCondition(0,0)*), after that the program stops and the application safety condition is valid. The system manager can now continue to test the other safety conditions until all are valid.



Note

SYS_WrIECSafeCondition (0,1) can be thought as a traffic light. When it is 0 then it means it is green, when it is 1 is equal to yellow and 2 equals to red.

In an application the status of the *SYS_WrIECSafeCondition* can change also if there is not a request to switch to the safety condition. In this way if an application is working and it is already in its safety condition, then it can set *SYS_WrIECSafeCondition* to 0.

Or it can set *SYS_WrIECSafeCondition* to 2, in this way the application informs that it can not modify its execution to put the application in the safety condition.

When *SYS_WrIECSafeCondition* is 1 means that the application is not in safety condition in this moment, but if a request arrives (e.g. in case a IEC program download, parameter file or firmware update, ...) then the application (IEC program) will execute a procedure (that must be written by the program developer) in order to put even the machine in its safety condition before to let the IEC safety procedure to start.

VAR_GLOBAL

```
a          : DINT := 10;
d          : DINT := 0;
SemIEC    : INT;
err       : DINT;
event     : DINT;
newrqst   : BOOL;
safeGo    : BOOL;
delay     : DINT;
Exception : INT := 0;
```

END_VAR

PROGRAM resetp

```
a := 0;
delay := 50000;
```

```
(*
```

```
 * it writes the semaphore IECsafeCondition to "yellow" = 1 for all
events
```

```
 * 0: all events
```

```
 * 1: semaphore = yellow
```

```
*)
```

```
err := SYS_WrIECSafeCondition(0, 1);
```

END_PROGRAM

PROGRAM run

```
delay := 50000;
safeGo := FALSE;
```

```
(*
```

```

    * it writes the semaphore IECSafeCondition to "yellow" = 1 for all
    events
    * 0: all events
    * 1: semaphore = yellow
    *)
err := SYS_WrIECSafeCondition(0, 1);
END_PROGRAM

PROGRAM exception
    (* after that only a forced download is allowed *)
    exception := exception + 1;
END_PROGRAM

PROGRAM main
    a := a + 1;

    (*
    * it checks if an event is requesting to program to go in safety
    condition
    *)
    newrqst := SYS_RqsIECSafeCondition(event);
    IF newrqst THEN
        safeGo := TRUE;
    END_IF;

    (*
    * it manages the safety condition procedure in the program
    *)
    IF safeGo THEN
        delay := delay - 1;
        IF delay <= 0 THEN
            (* when the time is elapsed the semaphore is set to green
            because
            * the program is arrived in the safety condition.
            * Then the execution of the program will go in Stopped mode
            *)
            err := SYS_WrIECSafeCondition(0, 0);
        END_IF;
    END_IF;

    (*
    * it reads the semaphore IECSafeCondition
    *)
    SemIEC := SYS_RdIECSafeCondition(0);

```

END_PROGRAM

Appendix A

Error codes list

These are the ErrorID codes:

error codes	description
0x07000201	axis number too big
0x07000202	parameter code does not exist
0x07000203	parameter type mistake
0x07000204	operation not allowed. If the function block is MC_Power this error can mean that the Enable hw in/out is configured but its state is FALSE
0x07000205	the previous operation is not finished
0x07000206	input is not valid
0x07000207	internal error
0x07000208	the axis is not STANDSTILL
0x07000209	homing is not finished
0x0700020A	drive is not ready to switch On
0x0700020B	axis is not compiled
0x0700020C	axes program is not running
0x0700020D	there is not enough memory
0x0700020E	the axis is free, but the command needs the axis in torque
0x0700020F	the axis is executing an emergency ramp, therefore the movement command is denied
0x07000210	the parameter is not local, so it cannot be read with a function, it has to be used the function block
0x07000211	the command is not executed because the axis is in INITIALITAZION state
0x07000212	internal error of MC_MoveCustom
0x07000213	internal calculation error

Table A.1. ErrorID codes

When the program has an exception, the object 8709 returns one of these codes:

error codes	description
0x80000001	Division by 0

error codes	description
0x80000002	power 0^0
0x80000003	sqrt of a number < 0
0x80000004	array out of range
0x000000F0	the SDSetup is too old (assembler old)
0x000000F8	a PROGRAM INT n is called, but it is not declared
0x000000F9	program CRC error
0x000000FA	the SDSetup is too old (compiler old)
0x000000FB	the firmware is too old
0x000000FC	the firmware does not support this program
0x000000FD	the project is too old (header old)
0x000000FE	the project is corrupt (header error)
other	it means there is an internal error, to contact CMZ

Table A.2. Error states

When a function block of the I/O peripheral has an exception, the object 8709 returns one of these codes:

error codes	description
0x05000002	Peripheral index not valid
0x05000003	Trigger event not allowed
0x05000004	Capture event not allowed
0x05010100	Logical input reference not valid
0x05010101	Trigger event handle not valid
0x05010102	Error on event activation
0x05050101	Trigger event handle not valid
0x05050112	Encoder event activation error

Table A.3. IO's function blocks error codes

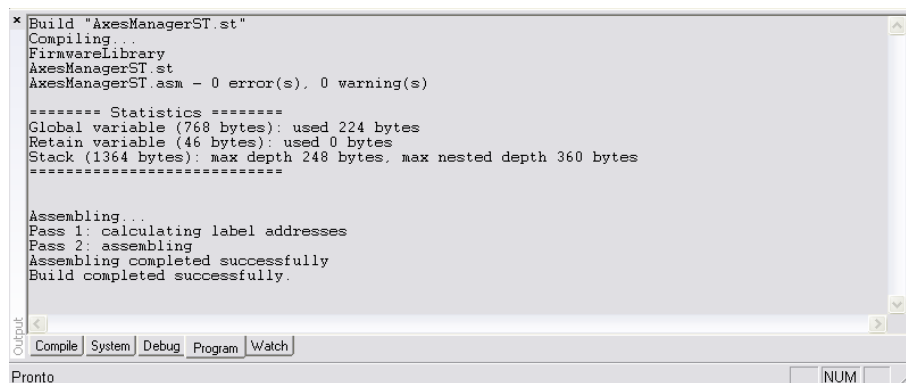
Appendix B

Parameters table

The parameters table is the same of the MODBUS protocol, therefore the address of the desired parameter has to be searched in the table reported in the manual of the drive.

How can I assess the memory usage?

One of the results of the compiling is the list of memory usage:



```
Build "AxesManagerST.st"
Compiling...
FirmwareLibrary
AxesManagerST.st
AxesManagerST.asm - 0 error(s), 0 warning(s)

===== Statistics =====
Global variable (768 bytes): used 224 bytes
Retain variable (46 bytes): used 0 bytes
Stack (1364 bytes): max depth 248 bytes, max nested depth 360 bytes
=====

Assembling...
Pass 1: calculating label addresses
Pass 2: assembling
Assembling completed successfully
Build completed successfully.
```

Output

Compile System Debug Program Watch

Pronto

NUM

Figure C.1. compiling result

In the section *Statistics* there is the report of the memory usage. The numbers written into the parenthesis (nnnn) are the limit values of each type of memory resource (see [Section 3.2.2, “Memory resources of the drive”](#)).

Objects of the programming

There are some objects dedicated to the programming.



Note

In some cells of the tables there is a simplifying formula, used to summarize the addressing rule for the %MD and %MW memory area. The "n" letter in the formula means the number of the byte of the %MD or %MW area (e.g. the %MD12 refers to the Double Word that starts on the byte 12 of the %MD memory area).

CANopen	PROFINET	Modbus	Acc.	Type	Description	Pdo Map
0x461A.01	43025.0	8720	UDINT32	RW	Period of UserTimer INT (11)	NO
0x4617.00	17943	8000	UINT32	RO	Real execution time of the main() program	NO
0x4520.00	17696	8774	UINT16	RW	Timer Free Running (increasing every ms)	NO
0x4616.00	17942	8773	UINT16	RO	Plc period	NO
0x4618.01	43009	8708	UINT16	RO	State of the virtual machine	NO
0x4618.02	43010	8709	UINT32	RO	Error of the virtual machine (see Table A.2)	NO
0x4618.03	43011	8711	UINT32	RO	Program counter of the virtual machine	NO
0x4618.04	43012	8713	UINT16	RO	Operative code executing	NO
0x4618.05	43013	8714	UINT16	RW	Threshold for the IEC event of low voltage (it is expressed in V*10)	NO
Addresses to read or write in %MD area						
Maximum %MD is %MD252 for SD series, %MD508 for BD series						
0x4700.00	-	-	UINT8	RO	Num of %MD, 64 sub indexes ^a , 128 sub indexes ^b	NO
0x4700.01	18176.0	8800	UINT32	RW	Exchange area %MD0	YES
0x4700.02	18176.1	8802	UINT32	RW	Exchange area %MD4	YES
0x4700.03	18176.2	8804	UINT32	RW	Exchange area %MD8	YES

CANopen	PROFINET	Modbus	Acc.	Type	Description	Pdo Map
0x4700.04	18176.3	8806	UINT32	RW	Exchange area %MD12	YES
...						
0x4700. (n/4+1)	18176.(n/4)	8800 + (n/2)	UINT32	RW	Exchange area %MDn	YES
...						
0x4700.40	18176.63	8926	UINT32	RW	Exchange area %MD252 ^a	YES
...						
0x4700.80	18176.127	9054	UINT32	RW	Exchange area %MD508 ^b	YES
Addresses to read or write in %MW area						
Maximum %MW is %MW254 ^a , %MW510 ^b						
0x4720.00	-	-	UINT8	RO	Num of %MW, 128 sub indexes	NO
0x4720.01	18208.0	8800	UINT16	RW	Exchange area %MW0	YES
0x4720.02	18208.1	8801	UINT16	RW	Exchange area %MW2	YES
0x4720.03	18208.2	8802	UINT16	RW	Exchange area %MW4	YES
0x4720.04	18208.3	8803	UINT16	RW	Exchange area %MW6	YES
...						
0x4720. (n/2+1)	18208.(n/2)	8800 + (n/2)	UINT16	RW	Exchange area %MWn	YES
...						
0x4720.80	18208.127	8927	UINT16	RW	Exchange area %MW254	
0x4721.00	-	-	UINT8	RO	Num of %MW, 128 sub indexes ^b	NO
0x4721.01	18209.0	8928	UINT16	RW	Exchange area %MW256 ^b	YES
0x4721.02	18209.1	8929	UINT16	RW	Exchange area %MW258 ^b	YES
0x4721.03	18209.2	8930	UINT16	RW	Exchange area %MW260 ^b	YES
0x4721.04	18209.3	8932	UINT16	RW	Exchange area %MW262 ^b	YES
...						
0x4721. ((n-0x100)/2+1)	18209. ((n-256)/2)	8928 + ((n-256)/2)	UINT16	RW	Exchange area %MWn ^b	YES
...						YES
0x4721.80	18209.127	9055	UINT16	RW	Exchange area %MW510 ^b	YES

^afor SD drives: only drives with firmware version less than 38. For BD drives it is always true.

^bfor SD drives: only drives with firmware version greater and equal than 38. For BD drives it is always true.

Table D.1. Programming objects

**FACTORY AND
HEADQUARTERS**

CMZ SISTEMI ELETTRONICI S.r.l.

Via dell'Artigianato, 21
31050 Vascon (TV) - Italy
Phone 39 (0)422 447411
Fax +39 (0)422 447444

e-mail: sales@cmz.it
web site: www.cmz.it



RESEARCH LABORATORY SINCE 1992