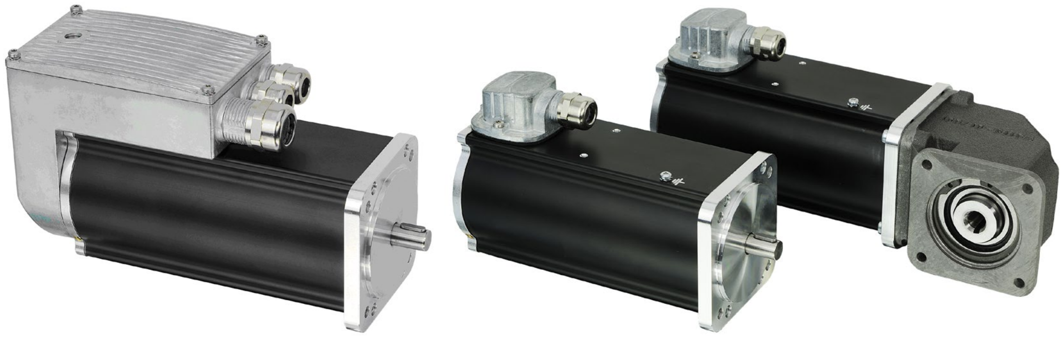
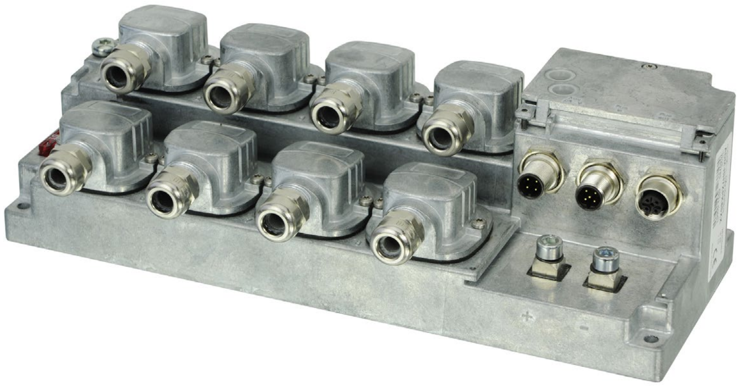
Instruction Manual
HIPERDRIVE with PROFIBUS DP





halstrup-walcher GmbH
Stegener Straße 10

D-79199 Kirchzarten, Germany

Tel.: +49 (0) 76 61/39 63-0
Fax: +49 (0) 76 61/39 63-99

E-Mail: info@halstrup-walcher.de
Internet: www.halstrup-walcher.de

Content

1 General advice

This documentation describes the use of the motion and communication FBs for Siemens STEP 7 in combination with halstrup-walcher HIPERDRIVES. Some of these function blocks are used for the motion control of an axis, while the others can be used for administration and maintenance purposes. The Development of the function blocks is based on the specification „PLCopen TC2 (Motion Control) – Part 1 (Basics)“ in Version 1.1. The communication with halstrup-walcher HIPERDRIVES is based on PROFIBUS.

HIPERDRIVE drives are constructed in accordance with the guidelines which are listed in the CE conformity declaration (see Part 1 of the Mounting and Operating Instructions) and are intended for **set-up and adjustment tasks on machines and parts** which are not subject to any special electrical and mechanical safety requirements. Measures which are used to protect the plant and personnel must be provided by the plant planner and/or operator in accordance with the necessary statutory guidelines.

For identification of the areas of use and safety precautions, please refer to the installation instruction. Please also consult the information there about the proper use of the HIPERDRIVE drives.

In the case of improper use and any damage which may result from this, the manufacturer will bear no responsibility. Proper use also includes compliance with the documentation and operating instructions which apply in each case.

1.1 Documentation and validity

The overall documentation of the HIPERDRIVE comprises the following parts:

- Datasheet HIPERDRIVE HDA..., HRA... family
- Mounting and Operating Instructions Part 1 Hardware description
- Mounting and Operating Instructions Part 2 Software description[1]
- Mounting Instructions[2]

This Mounting and Operating Instruction is intended to give you the necessary information to enable the programming of a control computer (PLC, industrial PC or the like) with which the HIPERDRIVE is controlled via a PROFIBUS-DP connection. Please read through these instructions carefully and take note of the tips and warning remarks.

1.2 Scope

This documentation is valid for:

- HRA08
- HDA45 A,B and E
- HDA30 A,B and E
- HIPERDRIVE-Hub

1.3 Symbols and their meanings

STOP Text which is identified with this symbol contains very important advice, also including advice for averting health risks.

[1] This is the present documentation

[2] HIPERDRIVE and if applicable HIPERDRIVE-HUB

⚠ Text which is identified with this symbol contains very important advice, also including advice for preventing damage to property. Observe this text without fail.

☞ This symbol indicates text which contains comments/advice or tips.

● This bullet identifies the descriptions of actions, which you should carry out.

2 Safety advice

Failure to observe the warning advice may result in bodily injury or damage to machines and plant. Appropriately qualified personnel must be thoroughly familiar with all the warning advice in these operating instructions, the hazards which can arise as a result of the plant specific conditions, as well as the safety regulations to be observed.

2.1 Personal protection

The safety provisions must be read and applied by every person involved with the commissioning, operation, maintenance and repair of the devices, plant or machine.

Please ensure that the voltage or power source with which the HIPERDRIVE is operated is designed in accordance with the technical and statutory guidelines which apply to your plant.
Before carrying out work on the drives themselves, or the components operated by them and the parts of the plant affected, the plant must be switched off in accordance with the regulations.
In addition to the main circuits, attention must be paid to any additional or auxiliary circuits which may be present.

Caution when the drive is touched by personnel. The HIPERDRIVE may assume a high housing temperature, depending on the mode of operation, because of the integrated motor and the power electronics.
Therefore, during installation, ensure that a sufficiently large distance from combustible materials and/or cables is maintained.
The output shaft of the HIPERDRIVE rotates with a rotational speed of up to 280 rev/min. In addition to care which is generally required, please pay attention to the hazards which can result from pieces of clothing, hair and the like becoming entangled.

2.2 Safety advice for mounting, repair, and commissioning

The HIPERDRIVE positioning drive is an electromechanical subassembly. The device must be mounted and connected in the voltage-free/current-free state.
In the event of improper handling, electrical short circuits with permanent consequential damage may occur.
As a result of commissioning the positioning drive, the position of a flange mounted stop/screw or the like - referred to below as an actuator - may be changed.
As a result, the flow of liquids or solid bodies, such as packages etc., may be made possible or interrupted, or other parts may become blocked.

2.2.1 Safety advice for mounting/repair

We wish to point out expressly that the mounting, electrical and mechanical installation and the repair of the HIPERDRIVE may be undertaken only by trained specialist staff with fundamental mechanical, electrical and programming knowledge.

Opening the HIPERDRIVE-HUB DP or the HIPERDRIVE or dismount the adapter is forbidden. Any repairs or inspections must only be carried out by the manufacturer's service department.

- Switch off all the devices/machines/plant involved in the mounting or repair.
- If appropriate, isolate the devices/machines/plant from the mains.

- Check whether switching off devices/machines/plant will cause potential danger.
- In the event of a fault in the HIPERDRIVE on a plant which is in operation, inform the shift manager/safety engineer or the operations manager without delay about the fault, in order to avoid, for example, an outflow/overflow of chemicals or erroneous filling of end products in good time by means of suitable measures.
- Depressurize pneumatic/hydraulic devices/machines/plant before the mounting or repair.
- If necessary, set up warning signs, in order to prevent inadvertent starting up of the devices/machines/plant.
- Carry out the mounting/repair work whilst complying with the relevant profession safety and ccident prevention regulations.
- Before completing the mounting/repair work and/or before the functional test, ensure that all the fixing screws are firmly tightened and that the cable connection is mounted correctly.
- Test the correct functioning of the safety devices (e.g. emergency off switch/safety clutches etc.).

2.2.2 Safety advice for adjustment/commissioning

The adjustment/commissioning may only be undertaken by a person with adequate system knowledge who is aware of the potential hazards.

- Make sure that the HIPERDRIVE is mounted correctly and all the fixing screws are firmly tightened.
- When connecting the drives, the correct polarity both of the voltage supplies for motor and bus electronics and the data lines must be checked without fail. Reversing the polarity of the voltage supply of the motor can destroy the power electronics. The drive does not include reverse polarity protection for the motor supply.
- Make sure that no torques hazardous to personnel or the surrounding area arises from the commissioning or as a result of the test adjustments on the actuating drive.
- If appropriate, set up warning signs in order to prevent devices/machines/plant being started up or shut down inadvertently.
- After completing the adjustments, check the correct functioning and, if necessary, the maintenance of the intended position of the actuator.
- Check the function of the software end position switches and the position feedback.
- Check whether the actuator has actually reached the desired position when the controller indicates the corresponding position.
- By employing suitable measures, prevent elements being pinched by moving actuators.
- Check the correct functioning of any safety devices (e.g. emergency-off buttons/safety couplings etc).
- Carry out the commissioning or the adjustments only in accordance with the instructions described in this documentation.

2.3 Device safety

The HIPERDRIVE positioning drive is a quality product produced in accordance with the recognized industrial regulations and has left the manufacturing plant in perfect condition with regard to safety.
In order to maintain this condition, you – as a user, commissioner, installer - must carry out your task in accordance with this description, technically correctly and with the greatest possible precision.

We assume that, as a trained specialist, you have the necessary knowledge to install, commission and operate the drive in accordance with the rules and regulations that apply to your application.
The positioning drive may be operated only within the values predefined in the technical data.
Commissioning is prohibited until it has been established that the plant/machine in which the positioning motor has beenincorporated complies withthe regulations which have to be applied to the plant/machine.
Make sure that no torques hazardous to persons and environment arise as a result of the mounting, commissioning or as a result of test adjustments.
Opening the drive is forbidden. Any repairs or inspections must only be carried out by the manufacturer's service department.

3 Product Description

3.1 Overview

The table below gives a brief overview on the available function blocks, which are described in detail in the following sections.

Function Block	Description
MC_Power	Controls the power stage of the drive (on or off).
MC_Stop	Implements a controlled motion stop and transfers the axis to the "Stopping" state
MC_Reset	Resetting all axis-internal errors / implements the transition from the "Errorstop" state to the "Standstill" state
MC_MoveAbsolute	Moves an axis to a defined "target position"
MC_MoveVelocity	Controls a "never ending" motion at the defined velocity
MC_ReadAxisError	Calls for the error information of the addressed axis
MC_ReadParameter	Returns the value of a drive parameter specified by the user
MC_WriteParameter	Writes a value to a parameter specified by the user
MC_FaultCheck	Show the "Fault" bit and "Warning" bit from status word

Fig. 1: Function Block overview

3.2 The State Diagram

The following diagram normatively defines the behavior of the axis at a high level when multiple motion control Function Blocks are «simultaneously» activated and work together on the axis.

The basic rule is that motion commands are always taken sequentially, even if the PLC had the capability of real parallel processing. These commands act on the axis' state diagram.

The axis is always in one of the defined states (see diagram below). Any motion command that causes a transition changes the state of the axis and, as a consequence, modifies the way the current motion is computed. The state diagram is an abstraction layer of what the real state of the axis is, comparable to the image of the I/O points within a cyclic (PLC) program.

A change of state is reflected immediately when issuing the corresponding motion command. (Note: the response time of 'immediately' is system dependent, coupled to the state of the axis, or an abstraction layer in the software)

The diagram is focused on a single axis.

Arrows within the state diagram show the possible state transitions between the states. State transitions due to an issued command are shown by full arrows. Dashed arrows are used for state transitions that occur when a command of an axis has terminated or a system related transition (like error related). The motion commands which transfers the axis to the corresponding motion state are listed above the states.

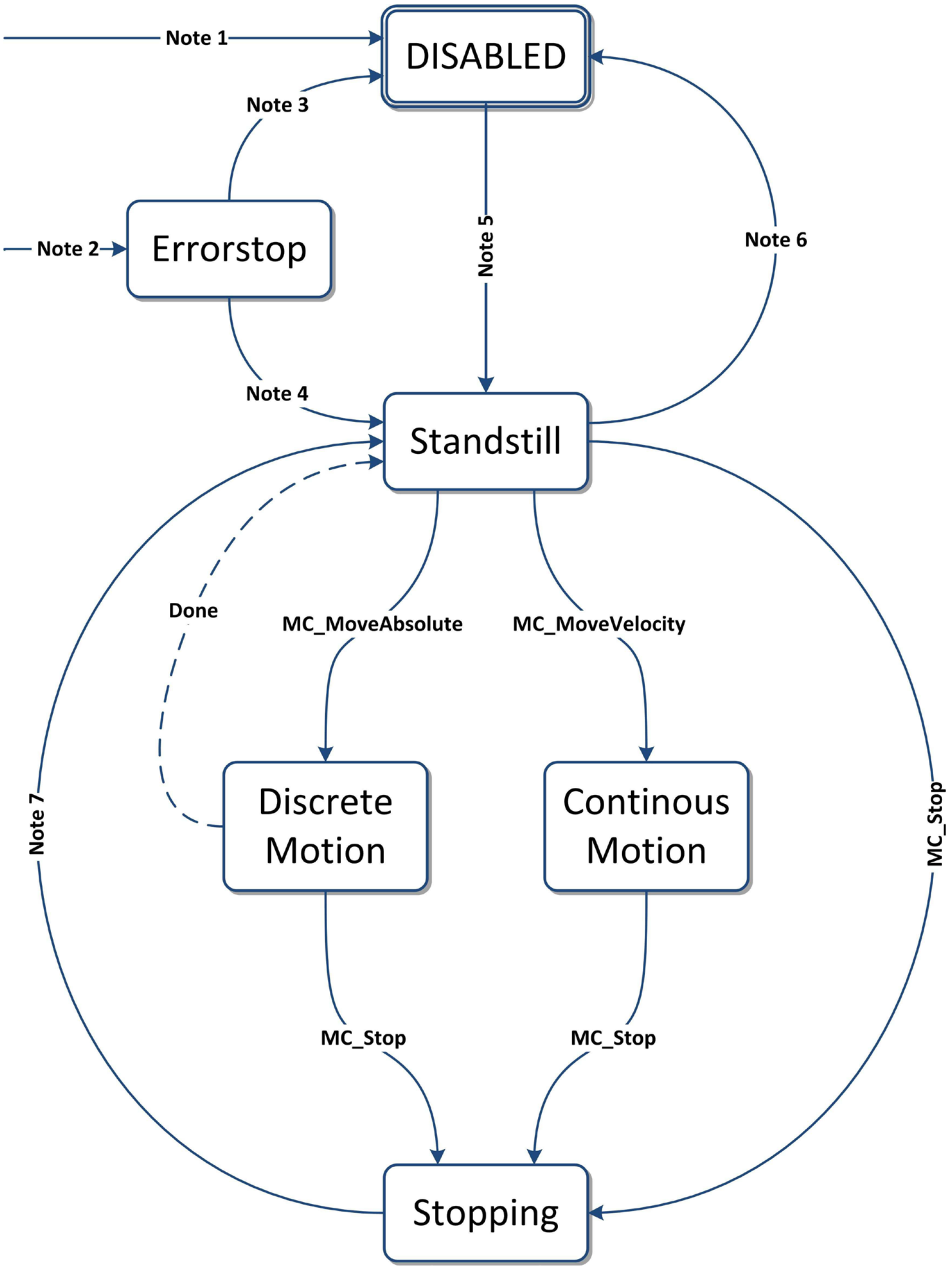
Remarks on states:

Disabled	The state 'Disabled' describes the initial state of the axis. In this state the movement of the axis is not influenced by the FBs. There is no error in the axis. If the MC_Power FB is called with Enable=TRUE while being in 'Disabled', the state changes to 'Standstill'. The axis feedback is operational before entering the state StandStill. Calling MC_Power with Enable=FALSE in any state except ErrorStop transfers the axis to the state 'Disabled', either directly or via any other state. Any on-going motion commands on the axis are aborted (Command Aborted).
ErrorStop	ErrorStop is valid as highest priority and applicable in case of an error. As long as the error is pending the state remains ErrorStop. The intention of the "ErrorStop" state is that the axis goes to a stop, if possible. There are no further FBs accepted until a reset has been done from the ErrorStop state. The transition to ErrorStop refers to errors from the axis and axis control, and not from the Function Block instances.
StandStill	Power is on, there is no error in the axis, and there are no motion commands active on the axis.

Remarks on commands:

MC_Stop	Calling the FB MC_Stop in state "StandStill" changes the state to "Stopping" and back to "Standstill" when "Execute = FALSE". The state "Stopping" is kept as long as the input "Execute" is true. The "Done" output is set when the stop ramp is finished.

Function Blocks which are not listed in the Diagram do not affect the state of the State Diagram, meaning that whenever they are called the state does not change. They are: MC_ReadAxisError, MC_ReadParameter, MC_WriteParameter and MC_FaultCheck.



Note 1: From any state, if MC_Power.Enable = FALSE and there is no error in the axis.
Note 2: From any state, if an error in the axis occurred.
Note 3: MC_Reset AND MC_Power.Status = FALSE
Note 4: MC_Reset AND MC_Power.Status = TRUE AND MC_Power.Enable = TRUE
Note 5: MC_Power.Enable = TRUE AND MC_Power.Status = TRUE
Note 6: MC_Power.Enable = FALSE and there is no error in the axis.
Note 7: MC_Stop.Done = TRUE AND MC_Stop.Execute = FALSE

Fig. 2: FB State Diagram

☞ Note: basic knowledge in SCL is required.
The usage of SCL is recommended
Don’t change the Function Blocks ore Data Block

3.3 MC_Power

3.3.1 Brief Description

The function block **MC_Power** controls the power stage of the drive (ON or OFF). The activation of the block is a precondition for every motion. A special situation is given with the function block **MC_Stop**. With this block, the position is kept active by the drive. That means that the **MC_Power** block cannot be deactivated while MC_Stop is active.

Note: The power of a HIPERDRIVE is always "physically" connected. This function block only switches between the states "Not ready to switch on" and "Operation enabled" of the internal drive control mechanism. It will NOT switch the drive power "physically" ON or OFF.

3.3.2 Interface

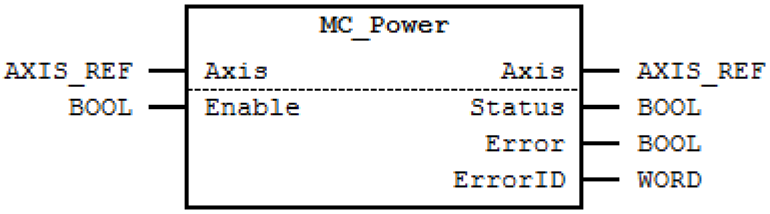


Fig. 3: MC_Power Interface Diagram

I/O Type	Name	Data Type	Description
VAR_IN_OUT	Axis	AXIS_REF	Data structure which contains several information for data exchange with other function blocks and communication settings for the drive. For further information on this structure please see section 3.12
VAR_INPUT	Enable	BOOL	Power is connected as long as "Enable" input level is "TRUE"
VAR_OUTPUT	Status	BOOL	Actual status of power connection
	Error	BOOL	Indicates that an error has occurred while processing the FB
	ErrorID	WORD	Error identification

Fig. 4: MC_Power I/O Interface Description

3.3.3 Min- / Max- and Default-Values of inputs

Name	Type	Min-Value	Max-Value	Default-Value	Takeover
Enable	BOOL			FALSE	Continuous

Fig. 5: Min- / Max- and Default-Values for MC_Power

3.3.4 Signal-Time Diagram

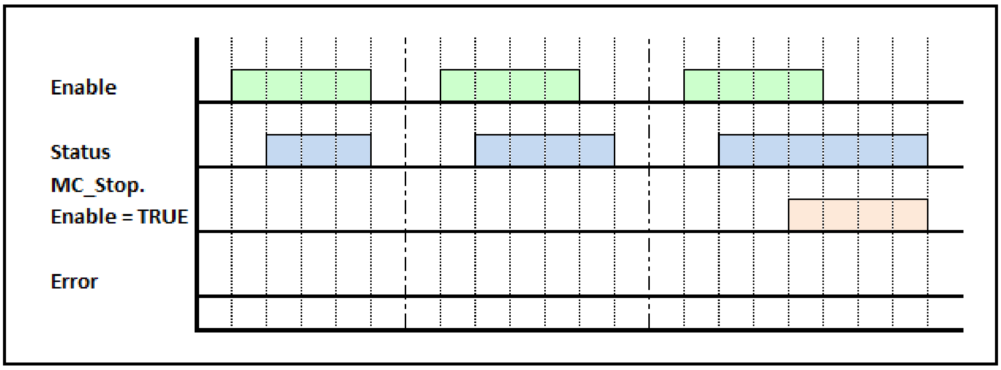


Fig. 6: Signal-Time Diagram for MC_Power – Processing terminated successfully

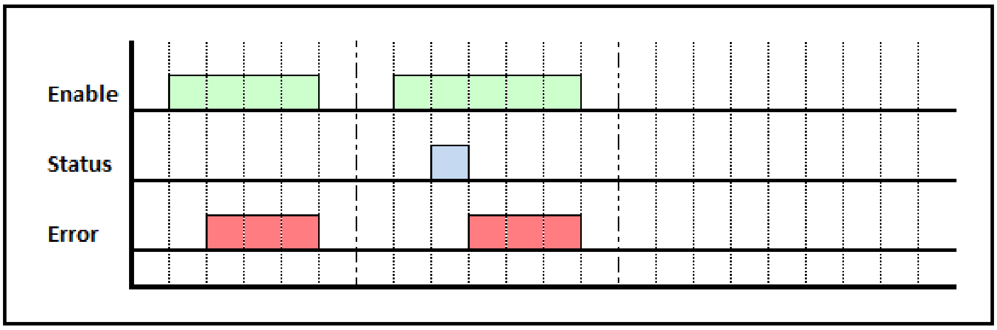


Fig. 7: Signal-Time Diagram for MC_Power – Processing terminated by error

3.3.5 Code example for MC_Power FB call in SCL

The code example below shows one way of calling an instance of **MC_Power** in SCL:

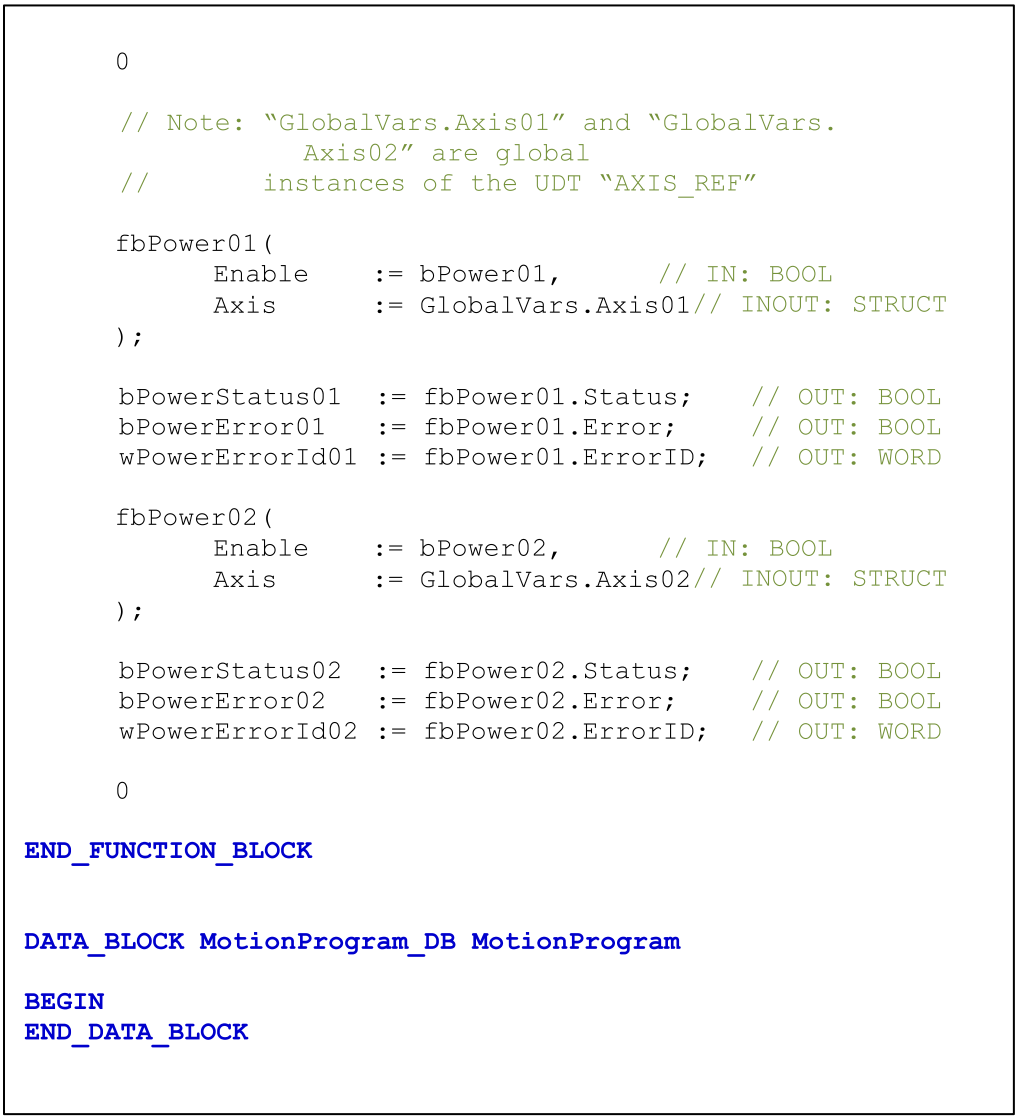
```
FUNCTION_BLOCK MotionProgram

      0

      VAR
        0
        (* in- and output variables for "MC_Power" *)
        bPower          : BOOL := FALSE;
        bPowerStatus    : BOOL := FALSE;
        bPowerError     : BOOL := FALSE;
        wPowerErrorID : WORD := W#16#0000;
        0
      END_VAR

      0

BEGIN

      0

      (* PKW address configuration for 1st axis *)
      GlobalVars.Axis01.PkwAddressIn   := 256;
      GlobalVars.Axis01.PkwAddressOut  := 256;

      (* PZD address configuration for 1st axis *)
      GlobalVars.Axis01.PzdAddressIn   := 264;
      GlobalVars.Axis01.PzdAddressOut  := 264;


      // Note 1: "DBxxx" is an "Instance DB"
                  for this instance of "MC_Power"
      // Note 2: "GlobalVars.Axis01" is a
                  global instance of the UDT "AXIS_REF"

      MC_Power.DBxxx(
            Enable  := bPower,            // IN: BOOL
            Axis    := GlobalVars.Axis01 // INOUT: STRUCT
      );

      bPowerStatus   := DBxxx.Status;    // OUT: BOOL
      bPowerError    := DBxxx.Error;     // OUT: BOOL
      wPowerErrorID := DBxxx.ErrorID;    // OUT: WORD

      0

END_FUNCTION_BLOCK


DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 8: Code example for MC_Power FB call in SCL

3.3.6 Code example for MC_Power FB call in SCL (Multi Instance)

The code example below shows one way of calling multiple instances of MC_Power in SCL:

```
FUNCTION_BLOCK MotionProgram

      0

      VAR

        0

        (* in- and output variables for 1st instance
                  of “fbPower01” *)
        bPower01        : BOOL := FALSE;
        bPowerStatus01  : BOOL := FALSE;
        bPowerError01   : BOOL := FALSE;
        wPowerErrorId01 : WORD := W#16#0000;
        (* in- and output variables for 2nd instance
                  of “fbPower02” *)
        bPower02        : BOOL := FALSE;
        bPowerStatus02  : BOOL := FALSE;
        bPowerError02   : BOOL := FALSE;
        wPowerErrorId02 : WORD := W#16#0000;
        (* instances of “MC_Power” *)
        fbPower01     : MC_Power;
        fbPower02     : MC_Power;

        0

      END_VAR

      0

BEGIN

      0

      (* PKW address configuration for 1st axis *)
      GlobalVars.Axis01.PkwAddressIn  := 256;
      GlobalVars.Axis01.PkwAddressOut := 256;

      (* PZD address configuration for 1st axis *)
      GlobalVars.Axis01.PzdAddressIn  := 264;
      GlobalVars.Axis01.PzdAddressOut := 264;


      (* PKW address configuration for 2nd axis *)
      GlobalVars.Axis02.PkwAddressIn  := 272;
      GlobalVars.Axis02.PkwAddressOut := 272;

      (* PZD address configuration for 2nd axis *)
      GlobalVars.Axis02.PzdAddressIn  := 280;
      GlobalVars.Axis02.PzdAddressOut := 280;

      0
```



```
      0

      // Note: “GlobalVars.Axis01” and “GlobalVars.
                   Axis02” are global
      //       instances of the UDT “AXIS_REF”

      fbPower01(
            Enable    := bPower01,      // IN: BOOL
            Axis      := GlobalVars.Axis01// INOUT: STRUCT
      );

      bPowerStatus01  := fbPower01.Status;    // OUT: BOOL
      bPowerError01   := fbPower01.Error;     // OUT: BOOL
      wPowerErrorId01 := fbPower01.ErrorID;   // OUT: WORD

      fbPower02(
            Enable    := bPower02,      // IN: BOOL
            Axis      := GlobalVars.Axis02// INOUT: STRUCT
      );

      bPowerStatus02  := fbPower02.Status;    // OUT: BOOL
      bPowerError02   := fbPower02.Error;     // OUT: BOOL
      wPowerErrorId02 := fbPower02.ErrorID;   // OUT: WORD

      0

END_FUNCTION_BLOCK


DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 9: Code example for MC_Power multiple instance FB call in SCL

3.3.7 Code example for MC_Power FB call in FBD

Name	Data Type	Initial Value
bPower	Bool	False
bPowerStatus	Bool	False
bPowerError	Bool	False
bPowerErrorID	Bool	W#16#0

Fig. 10: Variable declaration for MC_Power FB call

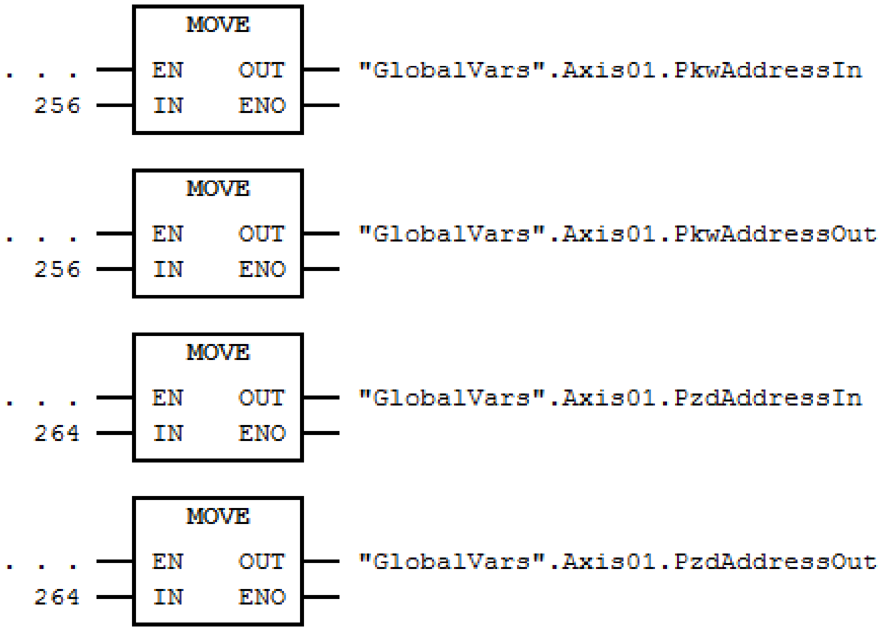

MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressIn
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressOut
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressIn
264
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressOut
264
IN
ENO

Fig. 11: Address configuration for 1st axis in FBD

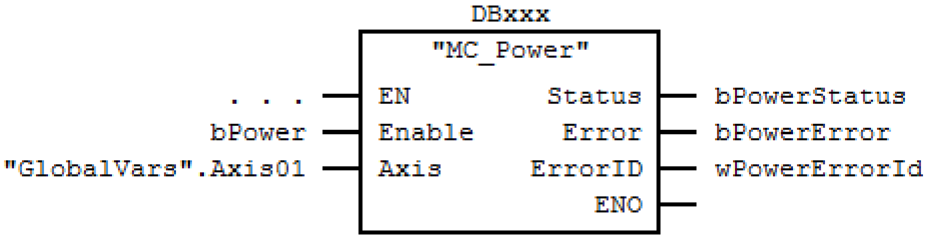
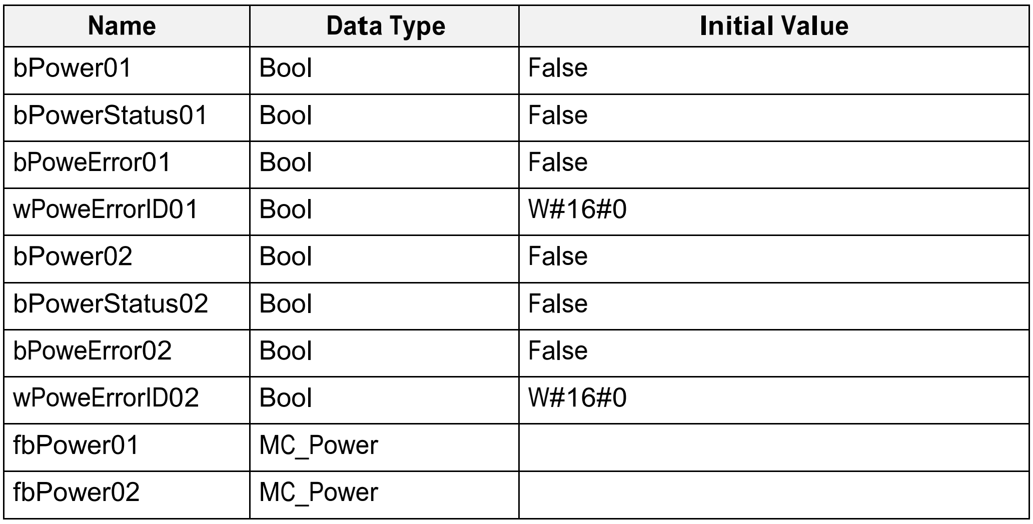


Fig. 12: MC_Power FB call in FBD

Note:
"DBxxx" is an "Instance DB" for this instance of "MC_Power"
"GlobalVars.Axis01" is a global instance of the UDT "AXIS_REF"

3.3.8 Code example for MC_Power FB call in FBD (Multi Instance)



Name	Data Type	Initial Value
bPower01	Bool	False
bPowerStatus01	Bool	False
bPoweError01	Bool	False
wPoweErrorID01	Bool	W#16#0
bPower02	Bool	False
bPowerStatus02	Bool	False
bPoweError02	Bool	False
wPoweErrorID02	Bool	W#16#0
fbPower01	MC_Power	
fbPower02	MC_Power	

Fig. 13: Variable declaration for multiple instance calls of MC_Power FB

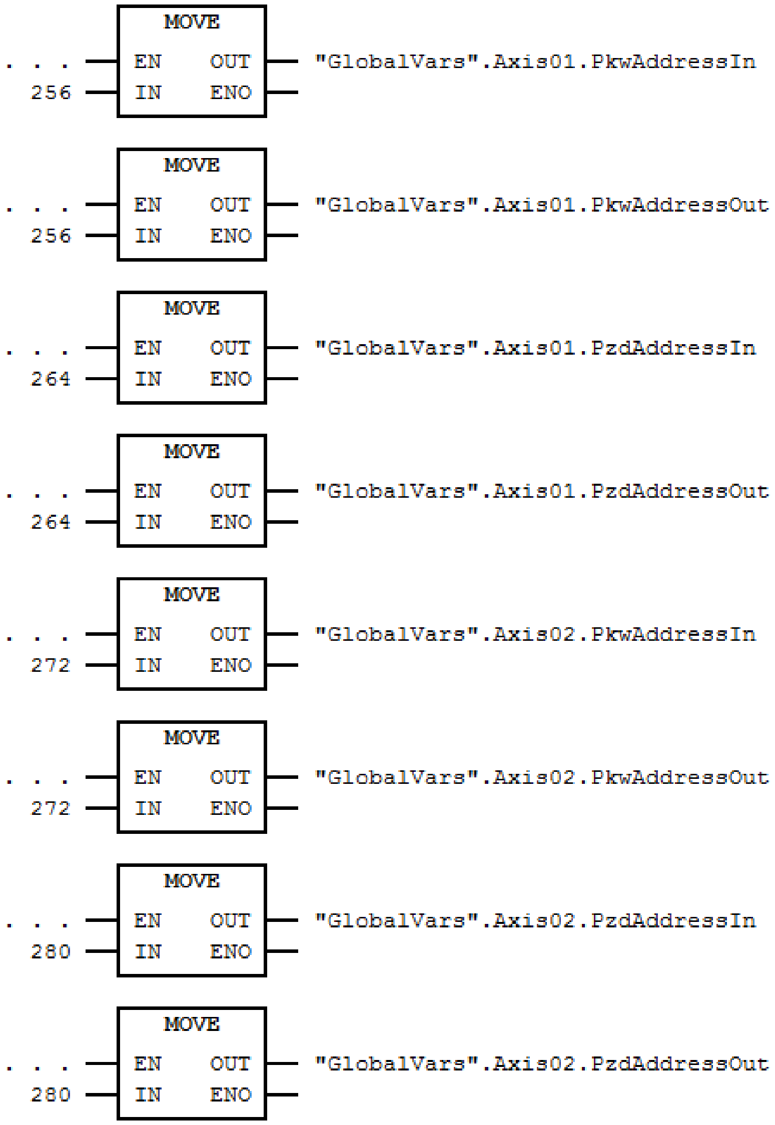

MOVE
. . .
256
EN
IN
OUT
ENO
"GlobalVars".Axis01.PkwAddressIn
MOVE
. . .
256
EN
IN
OUT
ENO
"GlobalVars".Axis01.PkwAddressOut
MOVE
. . .
264
EN
IN
OUT
ENO
"GlobalVars".Axis01.PzdAddressIn
MOVE
. . .
264
EN
IN
OUT
ENO
"GlobalVars".Axis01.PzdAddressOut
MOVE
. . .
272
EN
IN
OUT
ENO
"GlobalVars".Axis02.PkwAddressIn
MOVE
. . .
272
EN
IN
OUT
ENO
"GlobalVars".Axis02.PkwAddressOut
MOVE
. . .
280
EN
IN
OUT
ENO
"GlobalVars".Axis02.PzdAddressIn
MOVE
. . .
280
EN
IN
OUT
ENO
"GlobalVars".Axis02.PzdAddressOut

Fig. 14: Address configuration for 1st and 2nd axis in FBD

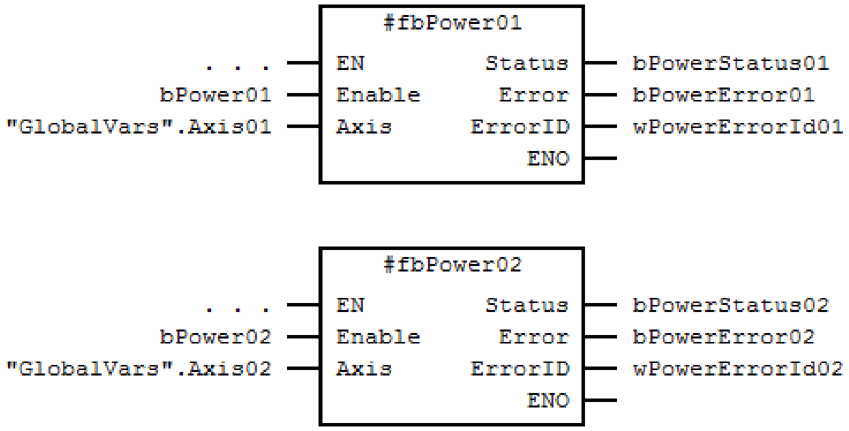


Fig. 15: FB calls of MC_Power (as Multiple Instances) in FBD

Note:
"GlobalVars.Axis01" and "GlobalVars.Axis02" are global instances of the UDT "AXIS_REF"

3.3.9 Error Handling

ErrorID (hex)	ErrorID (bin)	Description
16#1101	2#0001_0001_0000_0001	Error main state machine
16#1102	2#0001_0001_0000_0010	Error "in operation" state machine
16#1201	2#0001_0010_0000_0001	Invalid PZD input address
16#1202	2#0001_0010_0000_0010	Invalid PZD output address
16#1301	2#0001_0011_0000_0001	Error while reading ZSW
16#1302	2#0001_0011_0000_0010	Error while writing STW
16#1401	2#0001_0100_0000_0001	MC_Reset was executed while MC_Power was active

Fig. 16: Error Codes of MC_Power

3.4 MC_Stop

3.4.1 Brief Description

The function block **MC_Stop** implements a controlled motion stop and transfers the axis to the state "Stopping". It aborts any ongoing Function Block execution. While the axis is in state "Stopping", no other FB can perform any motion on the same axis. After the axis has reached velocity zero, the "Done" output is set to TRUE immediately. The axis remains in the state "Stopping" as long as Execute is still TRUE or velocity zero is not yet reached. As soon as "Done" is SET and "Execute" is FALSE the axis switches to state "StandStill".

While Execute is true, MC_Power can't be disabled, because Stop means "stay in position". This is not provided while MC_Power is disabled.

3.4.2 Interface

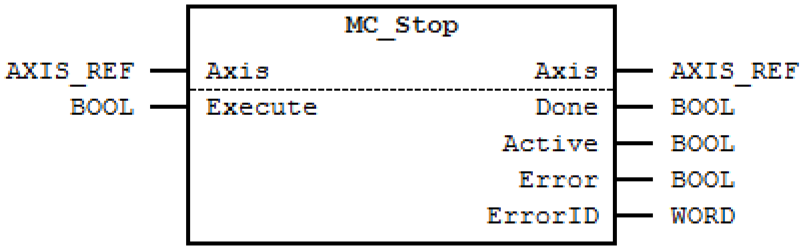


Fig. 17: MC_Stop I/O Interface Diagram

I/O Type	Name	Data Type	Description
VAR_IN_OUT	Axis	AXIS_REF	Data structure which contains several information for data exchange with other function blocks and communication settings for the drive. For further information on this structure please see section 3.12
VAR_INPUT	Execute	BOOL	Rising Edge at Execute stops the axis / transfers the axis into state “Stopping”. The Axis is stopping / kept in the “Stopping” state, as long as Execute is remains “TRUE”
VAR_OUTPUT	Done	BOOL	Standstill (zero velocity) reached
	Active	BOOL	Function Block is processing data
	Error	BOOL	Indicates that an error has occurred while processing the FB
	ErrorID	WORD	Error identification

Fig. 18: MC_Stop I/O Interface Description

3.4.3 Min- / Max- and Default-Values of inputs

Name	Type	Min-Value	Max-Value	Default-Value	Takeover
Execute	BOOL			FALSE	Continuous

Fig. 19: Min- / Max- and Default-Values for MC_Stop

3.4.4 Signal-Time Diagram

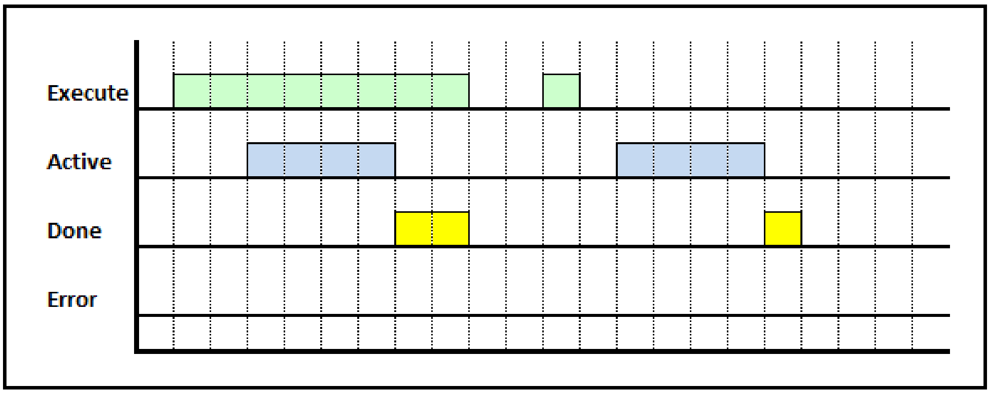


Fig. 20: Signal-Time Diagram MC_Stop for – Processing terminated successfully

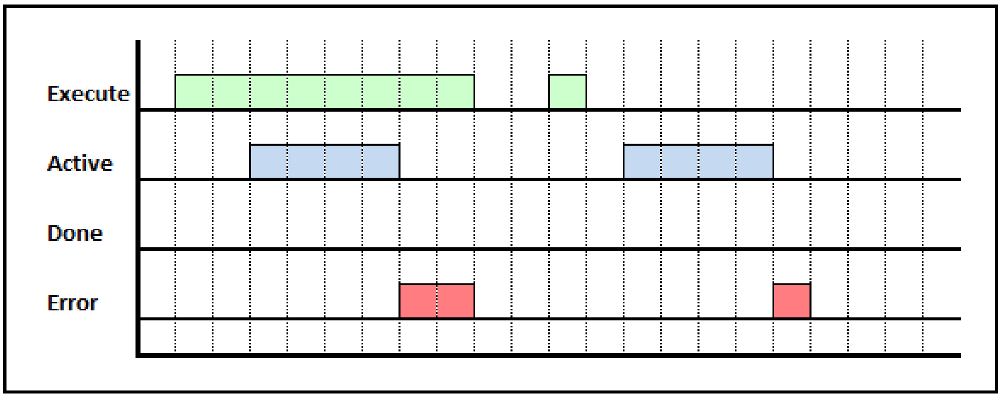


Fig. 21: Signal-Time Diagram for MC_Stop – Processing terminated by error

3.4.5 Code example for MC_Stop FB call in SCL

The code example below shows one way of calling an instance of **MC_Stop** in SCL:

```
FUNCTION_BLOCK MotionProgram
      0
      VAR
        0
        (* in- and output variables for “MC_Stop” *)
        bStop        : BOOL  := FALSE;
        bStopDone    : BOOL  := FALSE;
        bStopActive  : BOOL  := FALSE;
        bStopError   : BOOL  := FALSE;
        wStopErrorID : WORD  := W#16#0000;
        0
       END_VAR
       0
BEGIN
       0

       (* PKW address configuration for 1st axis
       *) GlobalVars.Axis01.PkwAddressIn  := 256;
       GlobalVars.Axis01.PkwAddressOut := 256;

       (* PZD address configuration for 1st axis
       *) GlobalVars.Axis01.PzdAddressIn  := 264;
       GlobalVars.Axis01.PzdAddressOut := 264;


       // Note 1: “DBxxx” is an “Instance DB” for
                  this instance of “MC_Stop”
       // Note 2: “GlobalVars.Axis01” is a global
                  instance of the UDT “AXIS_REF”

       MC_Stop.DBxxx(
       Execute := bStop,      // IN: BOOL
       Axis := GlobalVars.Axis01 // INOUT: STRUCT
       );

      bStopDone     := DBxxx.Done;       // OUT: BOOL
      bStopActive   := DBxxx.Active;     // OUT: BOOL
      bStopError    := DBxxx.Error;      // OUT: BOOL
      wStopErrorID  := DBxxx.ErrorID;    // OUT: WORD

      0

END_FUNCTION_BLOCK


DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 22: Code example for MC_Stop FB call in SCL

3.4.6 Code example for MC_Stop FB call in SCL (Multi Instance)

The code example below shows one way of calling multiple instances of **MC_Stop** in SCL:

```
FUNCTION_BLOCK MotionProgram
      0
      VAR
        0
        (* in- and output variables for “fbStop01” *)
        bStop01        : BOOL := FALSE;
        bStopDone01    : BOOL := FALSE;
        bStopActive01  : BOOL := FALSE;
        bStopError01   : BOOL := FALSE;
        wStopErrorId01 : WORD := W#16#0000;

        (* in- and output variables for “fbStop02” *)
        bStop02        : BOOL := FALSE;
        bStopDone02    : BOOL := FALSE;
        bStopActive02  : BOOL := FALSE;
        bStopError02   : BOOL := FALSE;
        wStopErrorId02 : WORD := W#16#0000;

        (* instances of “MC_Stop” *)
        fbStop01      : MC_Stop;
        fbStop02      : MC_Stop;

        0
      END_VAR
      0
BEGIN
      0

      (* PKW address configuration for 1st axis *)
      GlobalVars.Axis01.PkwAddressIn  := 256;
      GlobalVars.Axis01.PkwAddressOut := 256;

      (* PZD address configuration for 1st axis *)
      GlobalVars.Axis01.PzdAddressIn  := 264;
      GlobalVars.Axis01.PzdAddressOut := 264;


      (* PKW address configuration for 2nd axis *)
      GlobalVars.Axis02.PkwAddressIn  := 272;
      GlobalVars.Axis02.PkwAddressOut := 272;

      (* PZD address configuration for 2nd axis *)
      GlobalVars.Axis02.PzdAddressIn  := 280;
      GlobalVars.Axis02.PzdAddressOut := 280;

      0
      0

      // Note: “GlobalVars.Axis01” and “GlobalVars.Axis02”
                  are global
      //        instances of the UDT “AXIS_REF”

      fbStop01(
            Execute   := bStop01,         // IN: BOOL
            Axis      := GlobalVars.Axis01
                                         // INOUT: STRUCT
      );

      bStopDone01    := fbStop01.Done;         // OUT: BOOL
      bStopActive01  := fbStop01.Active;       // OUT: BOOL
      bStopError01   := fbStop01.Error;        // OUT: BOOL
      wStopErrorId01 := fbStop01.ErrorID;      // OUT: WORD

      fbStop02(
            Execute   := bStop02,              // IN: BOOL
            Axis      := GlobalVars.Axis02
                                           // INOUT: STRUCT
      );

      bStopDone02    := fbStop02.Done;         // OUT: BOOL
      bStopActive02  := fbStop02.Active;       // OUT: BOOL
      bStopError02   := fbStop02.Error;        // OUT: BOOL
      wStopErrorId02 := fbStop02.ErrorID;      // OUT: WORD

      0

END_FUNCTION_BLOCK


DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 23: Code example for MC_Stop multiple instance FB call in SCL

3.4.7 Code example for MC_Stop FB call in FBD

Name	Data Type	Initial Value
bStop	Bool	False
bStopDone	Bool	False
bStopActive	Bool	False
bStopError	Bool	False
wStopErrorID	Bool	W#16#0

Fig. 24: Variable declaration for MC_Stop FB call

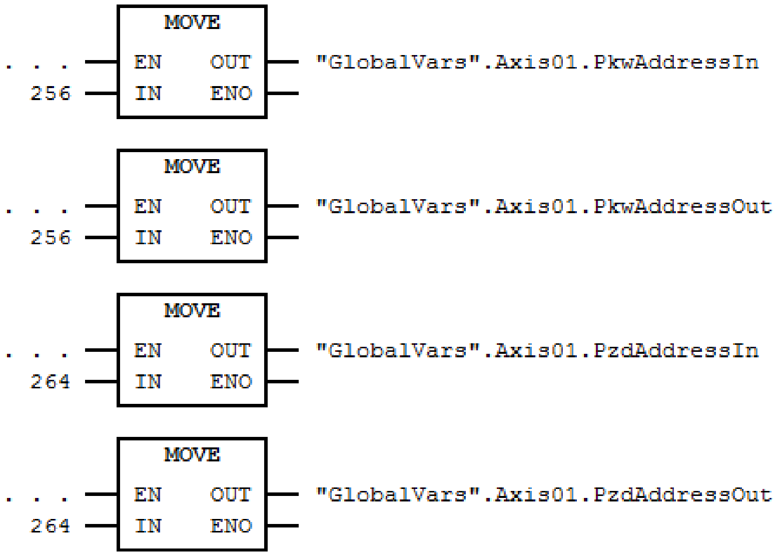

MOVE
EN
OUT
IN
ENO
. . .
256
"GlobalVars".Axis01.PkwAddressIn
MOVE
EN
OUT
IN
ENO
. . .
256
"GlobalVars".Axis01.PkwAddressOut
MOVE
EN
OUT
IN
ENO
. . .
264
"GlobalVars".Axis01.PzdAddressIn
MOVE
EN
OUT
IN
ENO
. . .
264
"GlobalVars".Axis01.PzdAddressOut

Fig. 25: Address configuration for 1st axis in FBD

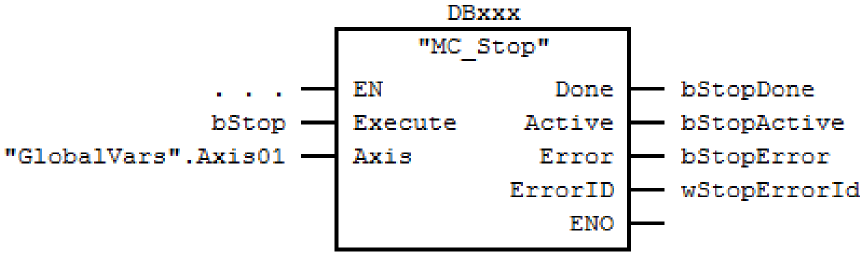


Fig. 26: MC_Stop FB call in FBD

Note:
“DBxxx” is an “Instance DB” for this instance of “MC_Stop”
“GlobalVars.Axis01” is a global instance of the UDT “AXIS_REF”

3.4.8 Code example for MC_Stop FB call in FBD (Multi Instance)

Name	Data Type	Initial Value
bStop01	Bool	False
bStopDone01	Bool	False
bStopActive01	Bool	False
bStopError01	Bool	False
wStopErrorID01	Bool	W#16#0
bStop02	Bool	False
bStopDone02	Bool	False
bStopActive02	Bool	False
bStopError02	Bool	False
wStopErrorID02	Bool	W#16#0

fbStop01	MC_Stop	
fbStop02	MC_Stop	

Fig. 27: Variable declaration for multiple instance calls of MC_Stop FB

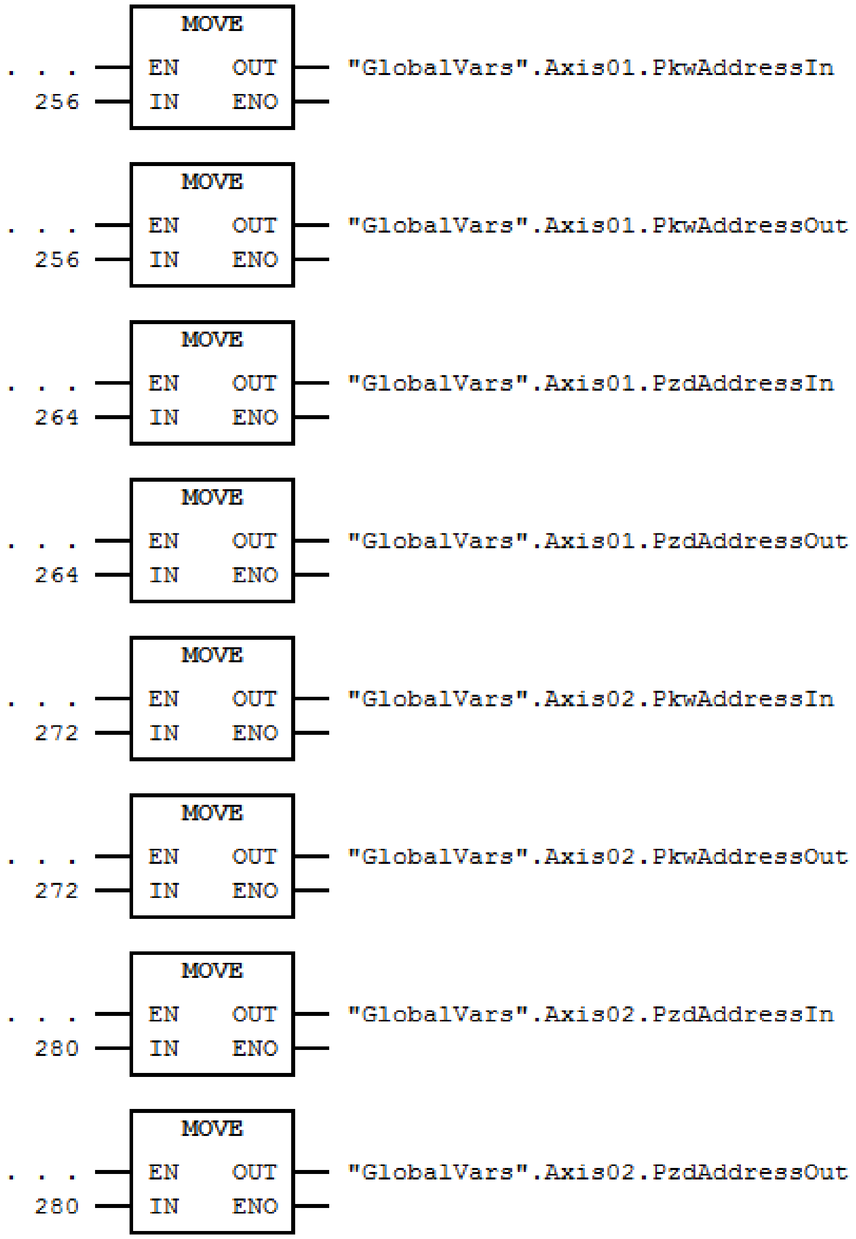

MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressIn
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressOut
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressIn
264
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressOut
264
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PkwAddressIn
272
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PkwAddressOut
272
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PzdAddressIn
280
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PzdAddressOut
280
IN
ENO

Fig. 28: Address configuration for 1st and 2nd axis in FBD

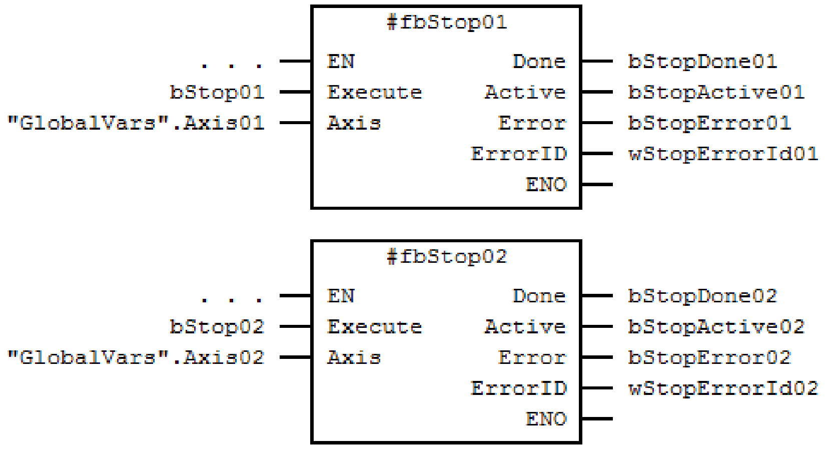


Fig. 29: FB calls of MC_Stop (as Multiple Instances) in FBD

Note:
“GlobalVars.Axis01” and “GlobalVars.Axis02” are global instances of the UDT “AXIS_REF”

3.4.9 Error Handling

ErrorID (hex)	ErrorID (bin)	Descripti on
16#2101	2#0010_0001_0000_0001	Error main state machine
16#2102	2#0010_0001_0000_0010	Error “in operation” state machine
16#2201	2#0010_0010_0000_0001	Invalid PZD input address
16#2202	2#0010_0010_0000_0010	Invalid PZD output address
16#2301	2#0010_0011_0000_0001	Error while reading ZSW
16#2302	2#0010_0011_0000_0010	Error while writing STW

Fig. 30: Error Codes of MC_Stop

3.5 MC_Reset

3.5.1 Brief Description

The Function Block **MC_Reset** implements the transition from the state “ErrorStop” to the state “StandStill” by resetting all axis-internal errors. If there is no axis error, the call of **MC_Reset** will cause, that the MC_Power enter the state disable although the input level is true. The MC_Reset must be disabled and enabled again.

NOTE: This Function Block does NOT affect the (error-) outputs of the FB instances!

3.5.2 Interface

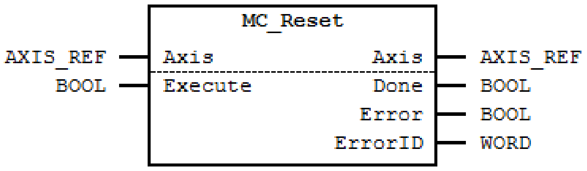


Fig. 31: MC_Reset I/O Interface Diagram

I/O Type	Name	Data Type	Description
VAR_IN_OUT	Axis	AXIS_REF	Data structure which contains several information for data exchange with other function blocks and communication settings for the drive. For further information on this structure please see section 3.12
VAR_INPUT	Execute	BOOL	Starts reset of axis internal errors at rising edge
VAR_OUTPUT	Done	BOOL	Error reset successful
	Error	BOOL	Indicates that an error has occurred while processing the FB
	ErrorID	WORD	Error identification

Fig. 32: MC_Reset I/O Interface Description

3.5.3 Min- / Max- and Default-Values of inputs

Name	Type	Min-Value	Max-Value	Default-Value	Takeover
Execute	BOOL			FALSE	Continuous

Fig. 33: Min- / Max- and Default-Values for MC_Reset

3.5.4 Signal-Time Diagram

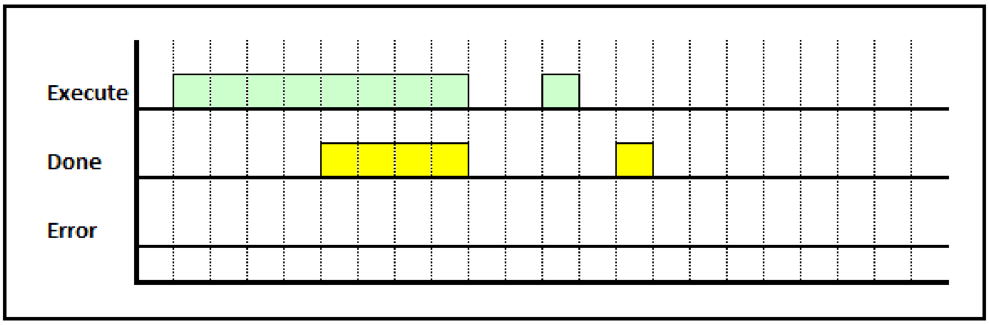


Fig. 34: Signal-Time Diagram MC_Reset for – Processing terminated successfully

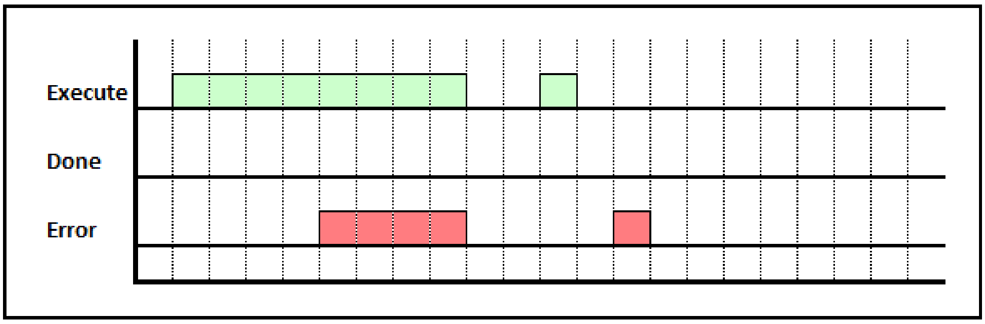


Fig. 35: Signal-Time Diagram for MC_Reset – Processing terminated by error

3.5.5 Code example for MC_Reset FB call in SCL

The code example below shows one way of calling an instance of **MC_Reset** in SCL:

```
FUNCTION_BLOCK MotionProgram
      0
      VAR

        0

        (* in- and output variables for “MC_Reset” *)
        bReset        : BOOL := FALSE;
        bResetDone    : BOOL := FALSE;
        bResetError   : BOOL := FALSE;
        wResetErrorID : WORD := W#16#0000;

        0
       END_VAR
      0
BEGIN
      0

      (* PKW address configuration for 1st axis *)
      GlobalVars.Axis01.PkwAddressIn  := 256;
      GlobalVars.Axis01.PkwAddressOut := 256;

      (* PZD address configuration for 1st axis *)
      GlobalVars.Axis01.PzdAddressIn  := 264;
      GlobalVars.Axis01.PzdAddressOut := 264;


      // Note 1: “DBxxx” is an “Instance DB” for
                              this instance of “MC_Reset”
      // Note 2: “GlobalVars.Axis01” is a global
                              instance of the UDT “AXIS_REF”

      MC_Reset.DBxxx(
            Execute := bReset,            // IN: BOOL
            Axis    := GlobalVars.Axis01 // INOUT: STRUCT
      );

      bResetDone    := DBxxx.Done;       // OUT:  BOOL
      bResetError   := DBxxx.Error;      // OUT:  BOOL
      wResetErrorID := DBxxx.ErrorID;    // OUT:  WORD

      0

END_FUNCTION_BLOCK


DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 36: Code example for MC_Reset FB call in SCL

3.5.6 Code example for MC_Reset FB call in SCL (Multi Instance)

The code example below shows one way of calling multiple instances of **MC_Reset** in SCL:

```
FUNCTION_BLOCK MotionProgram

      0

      VAR

        0

        (* in- and output variables for “fbReset01” *)
        bReset01         : BOOL := FALSE;
        bResetDone01     : BOOL := FALSE;
        bResetActive01   : BOOL := FALSE;
        bResetError01    : BOOL := FALSE;
        wResetErrorId01  : WORD := W#16#0000;

        (* in- and output variables for “fbReset02” *)
        bReset02         : BOOL := FALSE;
        bResetDone02     : BOOL := FALSE;
        bResetActive02   : BOOL := FALSE;
        bResetError02    : BOOL := FALSE;
        wResetErrorId02  : WORD := W#16#0000;

        (* instances of “MC_Reset” *)
        fbReset01       : MC_Reset;
        fbReset02       : MC_Reset;

        0

      END_VAR

      0

BEGIN

      0

      (* PKW address configuration for 1st axis *)
      GlobalVars.Axis01.PkwAddressIn   := 256;
      GlobalVars.Axis01.PkwAddressOut  := 256;

      (* PZD address configuration for 1st axis *)
      GlobalVars.Axis01.PzdAddressIn   := 264;
      GlobalVars.Axis01.PzdAddressOut  := 264;


      (* PKW address configuration for 2nd axis *)
      GlobalVars.Axis02.PkwAddressIn   := 272;
      GlobalVars.Axis02.PkwAddressOut  := 272;

      (* PZD address configuration for 2nd axis *)
      GlobalVars.Axis02.PzdAddressIn   := 280;
      GlobalVars.Axis02.PzdAddressOut  := 280;

      0
```

```
       0

       // Note: "GlobalVars.Axis01" and "GlobalVars.Axis02"
                        are global
       //       instances of the UDT "AXIS_REF"

       fbReset01(
             Execute   := bReset01,       // IN: BOOL
             Axis      := GlobalVars.Axis01
                                         // INOUT: STRUCT
       );

       bResetDone01     := fbReset01.Done;       // OUT: BOOL
       bResetError01    := fbReset01.Error;      // OUT: BOOL
       wResetErrorId01  := fbReset01.ErrorID;    // OUT: WORD

       fbReset02(
             Execute   := bReset02,       // IN: BOOL
             Axis      := GlobalVars.Axis02
                                         // INOUT: STRUCT
       );

       bResetDone02     := fbReset02.Done;       // OUT: BOOL
       bResetError02    := fbReset02.Error;      // OUT: BOOL
       wResetErrorId02  := fbReset02.ErrorID;    // OUT: WORD

       0

END_FUNCTION_BLOCK

DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 37: Code example for MC_Reset multiple instance FB call in SCL

3.5.7 Code example for MC_Reset FB call in FBD

Name	Data Type	Initial Value
bReset	Bool	False
bResetDone	Bool	False
bResetError	Bool	False
wResetErrorID	Bool	W#16#0

Fig. 38: Variable declaration for MC_Reset FB call

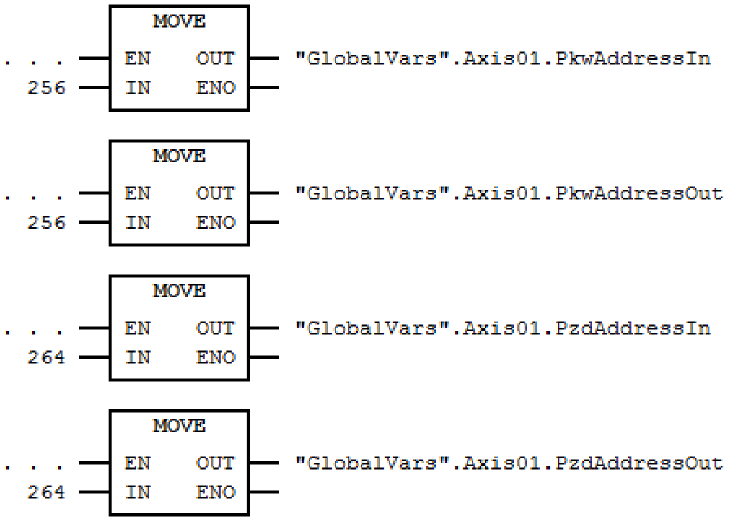

MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressIn
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressOut
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressIn
264
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressOut
264
IN
ENO

Fig. 39: Address configuration for 1st axis in FBD

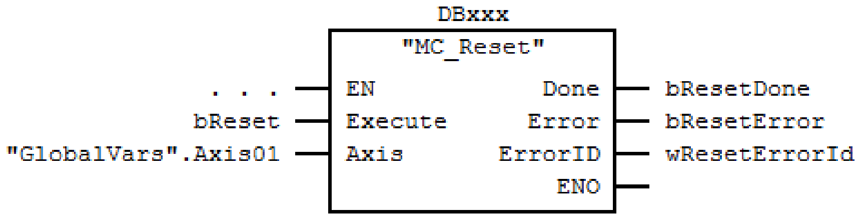


Fig. 40: MC_Reset FB call in FBD

Note:
"DBxxx" is an "Instance DB" for this instance of "MC_Reset"
"GlobalVars.Axis01" is a global instance of the UDT "AXIS_REF"

3.5.8 Code example for MC_Stop FB call in FBD (Multi Instance)

Name	Data Type	Initial Value
bReset01	Bool	False
bResetDone01	Bool	False
bResetError01	Bool	False
wResetErrorId01	Bool	W#16#0
bReset02	Bool	False
bResetDone02	Bool	False
bResetError02	Bool	False
wResetErrorID02	Bool	W#16#0
fbReset01	MC_Reset	
fbReset02	MC_Reset	

Fig. 41: Variable declaration for multiple instance calls of MC_Reset FB

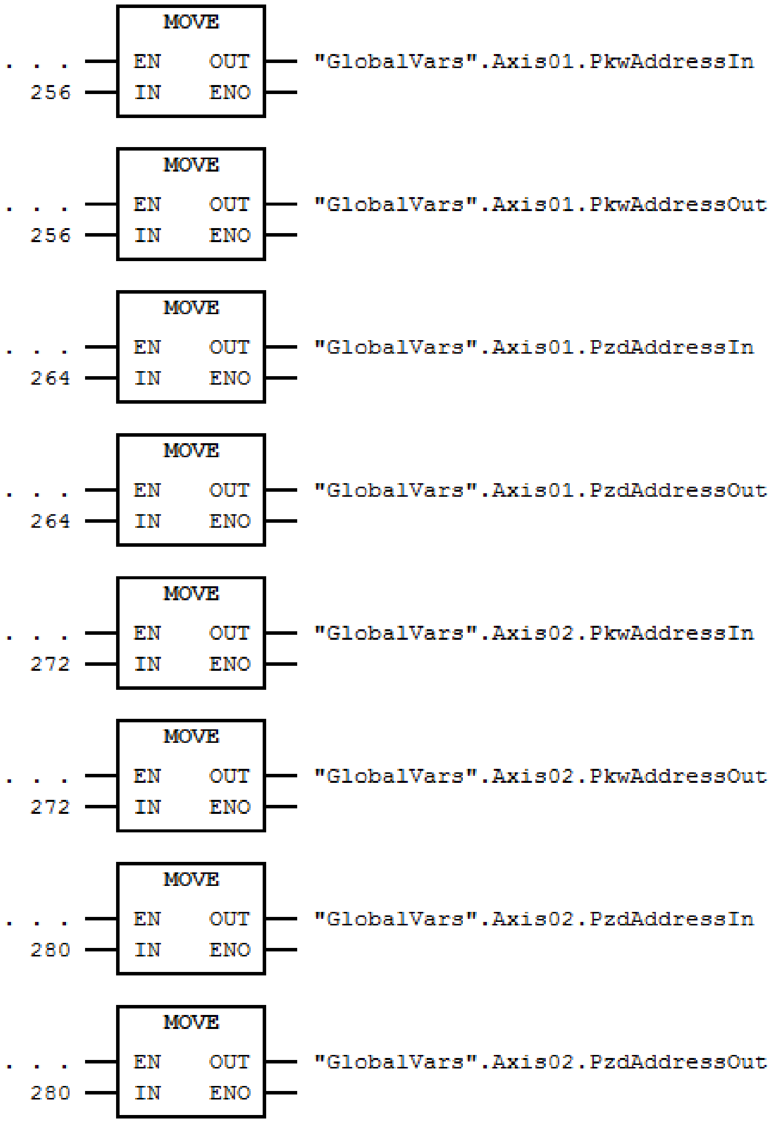

MOVE
EN
OUT
IN
ENO
. . .
256
"GlobalVars".Axis01.PkwAddressIn
256
"GlobalVars".Axis01.PkwAddressOut
264
"GlobalVars".Axis01.PzdAddressIn
264
"GlobalVars".Axis01.PzdAddressOut
272
"GlobalVars".Axis02.PkwAddressIn
272
"GlobalVars".Axis02.PkwAddressOut
280
"GlobalVars".Axis02.PzdAddressIn
280
"GlobalVars".Axis02.PzdAddressOut

Fig. 42: Address configuration for 1st and 2nd axis in FBD

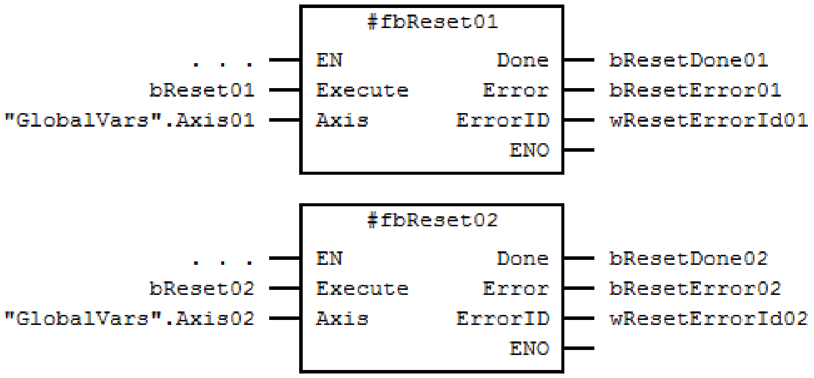


Fig. 43: FB calls of MC_Reset (as Multiple Instances) in FBD

Note: “GlobalVars.Axis01” and “GlobalVars.Axis02” are global instances of the UDT “AXIS_REF”

3.5.9 Error Handling

ErrorID (hex)	ErrorID (bin)	Description
16#3101	2#0011_0001_0000_0001	Error main state machine
16#3102	2#0011_0001_0000_0010	Error “in operation” state machine
16#3201	2#0011_0010_0000_0001	Invalid PZD input address
16#3202	2#0011_0010_0000_0010	Invalid PZD output address
16#3302	2#0011_0011_0000_0010	Error while writing STW

Fig. 44: Error Codes of MC_Reset

3.6 MC_MoveAbsolute

3.6.1 Brief Description

The Function Block **MC_MoveAbsolute** commands a controlled motion to a specified absolute position.

3.6.2 Interface

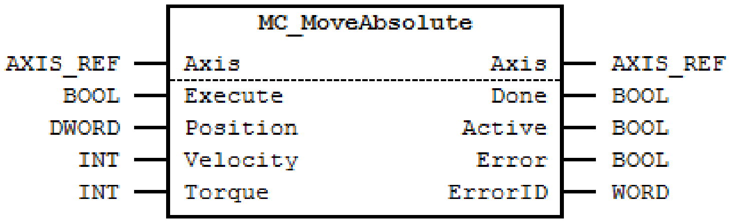


Fig. 45: MC_MoveAbsolute I/O Interface Diagram

Note:
HDA family: the torque parameter is valid for 25%, 50%, 75% and 100%, all other parameter values are rounded off.
HRA08: the torque parameter is ignored.
HDA family: the velocity parameter can be changed during movement
HRA08: changing the velocity parameter during movement will cause a warning

With function block MC_ReadParameter you can check, if the drive is still in position with parameter 1010 [3F2hex] "HIPERDRIVE status bits" Bit 5.
Bit 5=0 => the drive is in position
Bit 5=1 => the drive is not in position
According to the positioning accuracy in the datasheet.

It is not necessary to switch the drive mode (parameter 930) manually. This function block will automatically set parameter 930 to the correct value.

I/O Type	Name	Data Type	Descriptio n
VAR_IN_OUT	Axis	AXIS_REF	Data structure which contains several information for data exchange with other function blocks and communication settings for the drive. For further information on this structure please see section 3.12
VAR_INPUT	Execute	BOOL	Starts the motion at rising edge
	Position	DWORD	Target position
	Velocity	INT	Value of maximum Velocity in % (not necessarily reached)
	Torque	INT	Value of torque in %
VAR_OUTPUT	Done	BOOL	Target position reached
	Active	BOOL	Function Block is processing data
	Error	BOOL	Indicates that an error has occurred while processing the FB
	ErrorID	WORD	Error identification

Fig. 46: MC_MoveAbsolute I/O Interface Description

3.6.3 Min- / Max- and Default-Values of inputs

Name	Type	Min-Value	Max-Value	Default-Value	Takeover
Execute	BOOL			FALSE	Continuous
Position	DWORD	DW#16#0	DW#16#FFFF_FFFF	DW#16#0	Rising edge at “Execute”
Velocity	INT	1	100	0	Rising edge at “Execute”
Torque	INT	1	100	0	Rising edge at “Execute”

Fig. 47: Min- / Max- and Default-Values for MC_MoveAbsolute

3.6.4 Signal-Time Diagram

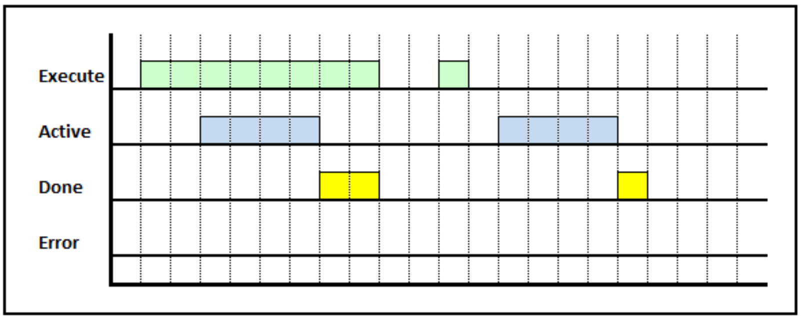


Fig. 48: Signal-Time Diagram MC_MoveAbsolute for – Processing terminated successfully

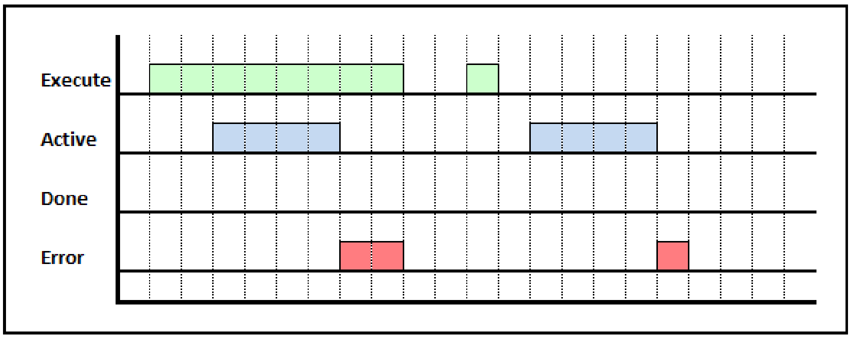


Fig. 49: Signal-Time Diagram for MC_MoveAbsolute – Processing terminated by error

3.6.5 Code example for MC_MoveAbsolute FB call in SCL

The code example below shows one way of calling an instance of **MC_MoveAbsolute** in SCL:

```
FUNCTION_BLOCK MotionProgram
      0
      VAR
        0
        (* in- and output variables for "MC_MoveAbsolute" *)
        bMoveAbsolute         : BOOL  := FALSE;
        dwPosition            : DWORD := DW#16#0000_0000;
        iVelocity             : INT   := 0; iTorque
                              : INT   := 0;
        bMoveAbsoluteDone     : BOOL  := FALSE;
        bMoveAbsoluteActive   : BOOL  := FALSE;
        bMoveAbsoluteError    : BOOL  := FALSE;
        wMoveAbsoluteErrorID  : WORD  := W#16#0000;
        0
      END_VAR
      0
BEGIN
     0
     (* PKW address configuration for 1st axis *)
     GlobalVars.Axis01.PkwAddressIn  := 256;
     GlobalVars.Axis01.PkwAddressOut := 256;

     (* PZD address configuration for 1st axis *)
     GlobalVars.Axis01.PzdAddressIn  := 264;
     GlobalVars.Axis01.PzdAddressOut := 264;

     // Note 1: "DBxxx" is an "Instance DB" for
                this instance of "MC_MoveAbsolute"
     // Note 2: "GlobalVars.Axis01" is a global
                instance of the UDT "AXIS_REF"

     MC_MoveAbsolute.DBxxx(
           Execute  := bMoveAbsolute,         // IN: BOOL
           Position := dwPosition,            // IN: DWORD
           Velocity := iVelocity,             // IN: INT
           Torque   := iTorque,               // IN: INT
           Axis     := GlobalVars.Axis01
                                         // INOUT: STRUCT
      );

      bMoveAbsoluteDone     := DBxxx.Done;     // OUT: BOOL
      bMoveAbsoluteActive   := DBxxx.Active;   // OUT: BOOL
      bMoveAbsoluteError    := DBxxx.Error;    // OUT: BOOL
      wMoveAbsoluteErrorID  := DBxxx.ErrorID;  // OUT: WORD

      0
END_FUNCTION_BLOCK

DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 50: Code example for MC_MoveAbsolute FB call in SCL

3.6.6 Code example for MC_MoveAbsolute FB call in SCL (Multi Instance)

The code example below shows one way of calling multiple instances of **MC_MoveAbsolute** in SCL:

```
FUNCTION_BLOCK MotionProgram
      0
      VAR
         0
         (* in- and output variables for “fbMoveAbsolute01” *)
        bMoveAbsolute01    : BOOL  := FALSE;
        dwPosition01       : DWORD := DW#16#0000_0000;
        iVelocity01        : INT   := 0;
        iTorque01          : INT   := 0;
        bMovAbsDone01      : BOOL  := FALSE;
        bMovAbsActive01    : BOOL  := FALSE;
        bMovAbsError01     : BOOL  := FALSE;
        wMovAbsErrorId01   : WORD  := W#16#0000;

         (* in- and output variables for “fbMoveAbsolute02” *)
        bMoveAbsolute02    : BOOL  := FALSE;
        dwPosition02       : DWORD := DW#16#0000_0000;
        iVelocity02        : INT   := 0;
        iTorque02          : INT   := 0;
        bMovAbsDone02      : BOOL  := FALSE;
        bMovAbsActive02    : BOOL  := FALSE;
        bMovAbsError02     : BOOL  := FALSE;
        wMovAbsErrorId02   : WORD  := W#16#0000;

         (* instances of “MC_MoveAbsolute” *)
        fbMoveAbsolute01 : MC_MoveAbsolute;
        fbMoveAbsolute02 : MC_MoveAbsolute;

        0
        END_VAR
        0
BEGIN
        0

        (* PKW address configuration for 1st axis
         *) GlobalVars.Axis01.PkwAddressIn:= 256;
         GlobalVars.Axis01.PkwAddressOut := 256;

        (* PZD address configuration for 1st axis
         *) GlobalVars.Axis01.PzdAddressIn:= 264;
         GlobalVars.Axis01.PzdAddressOut := 264;


        (* PKW address configuration for 2nd axis
         *) GlobalVars.Axis02.PkwAddressIn:= 272;
         GlobalVars.Axis02.PkwAddressOut := 272;

         (* PZD address configuration for 2nd axis
         *) GlobalVars.Axis02.PzdAddressIn:= 280;
         GlobalVars.Axis02.PzdAddressOut := 280;

        0
```

```
     0

      // Note: "GlobalVars.Axis01" and "GlobalVars.Axis02" are
                  global
      //        instances of the UDT "AXIS_REF"

      fbMoveAbsolute01(
            Execute     := bMoveAbsolute01,          // IN: BOOL
            Position    := dwPosition01,             // IN: DWORD
            Velocity    := iVelocity01,              // IN: INT
            Torque      := iTorque01,                // IN: INT
            Axis        := GlobalVars.Axis01
                                                // INOUT: STRUCT
      );

      bMovAbsDone01     := fbMoveAbsolute01.Done;    // OUT: BOOL
      bMovAbsActive01   := fbMoveAbsolute01.Active;  // OUT: BOOL
      bMovAbsError01    := fbMoveAbsolute01.Error;   // OUT: BOOL
      wMovAbsErrorId01  := fbMoveAbsolute01.ErrorID;// OUT: WORD


      fbMoveAbsolute02(
            Execute     := bMoveAbsolute02,          // IN: BOOL
            Position    := dwPosition02,             // IN: DWORD
            Velocity    := iVelocity02,              // IN: INT
            Torque      := iTorque02,                // IN: INT
            Axis        := GlobalVars.Axis02
                                                // INOUT: STRUCT
      );

      bMovAbsDone02     := fbMoveAbsolute02.Done;    // OUT: BOOL
      bMovAbsActive02   := fbMoveAbsolute02.Active;  // OUT: BOOL
      bMovAbsError02    := fbMoveAbsolute02.Error;   // OUT: BOOL
      wMovAbsErrorId02  := fbMoveAbsolute02.ErrorID;// OUT: WORD

      0

END_FUNCTION_BLOCK


DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 51: Code example for MC_MoveAbsolute multiple instance FB call in SCL

3.6.7 Code example for MC_MoveAbsolute FB call in FBD

Name	Data Type	Initial Value
bMoveAbsolute	Bool	False
dwPosition	DWord	DW#16#0
iVelocity	Int	0
iTorque	Int	0
bMoveAbsoluteDone	Bool	False

bMoveAbsoluteActive	Bool	False
bMoveAbsoluteError	Bool	False
wMoveAbsoluteErrorID	Bool	W#16#0

Fig. 52: Variable declaration for MC_MoveAbsolute FB call

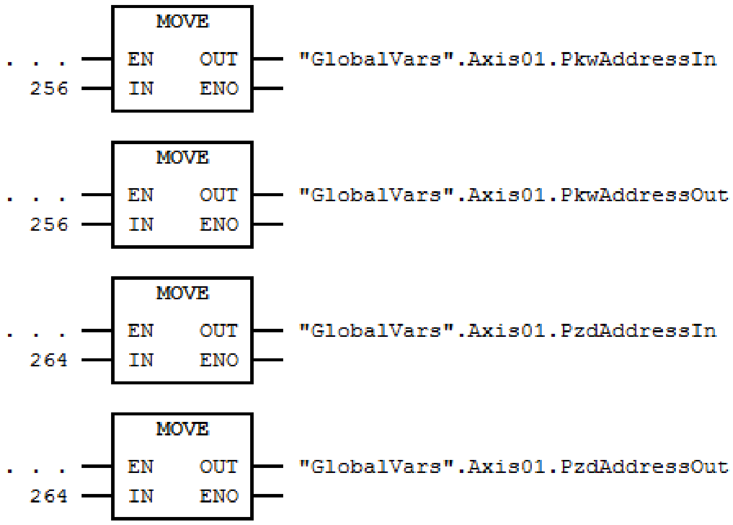

MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressIn
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressOut
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressIn
264
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressOut
264
IN
ENO

Fig. 53: Address configuration for 1st axis in FBD

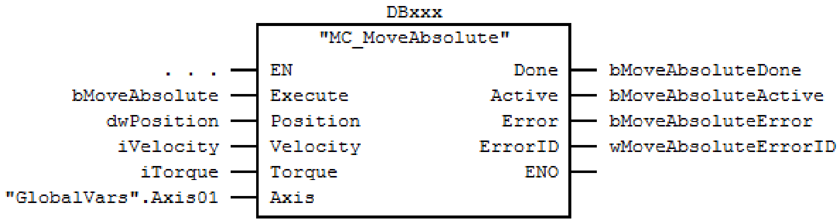


Fig. 54: MC_MoveAbsolute FB call in FBD

Note:
“DBxxx” is an “Instance DB” for this instance of “MC_MoveAbsolute”
“GlobalVars.Axis01” is a global instance of the UDT “AXIS_REF”

3.6.8 Code example for MC_MoveAbsolute FB call in FBD (Multi Instance)

Name	Data Type	Initial Value
bMoveAbsolute01	Bool	False
dwPosition01	DWord	DW#16#0
iVelocity01	Int	0
iTorque01	Int	0
bMoveAbsoluteDone01	Bool	False
bMoveAbsoluteActive01	Bool	False
bMoveAbsoluteError01	Bool	False

wMoveAbsoluteErrorID01	Bool	W#16#0
bMoveAbsolute02	Bool	False
dwPosition02	DWord	DW#16#0
iVelocity02	Int	0
iTorque02	Int	0
bMoveAbsoluteDone02	Bool	False
bMoveAbsoluteActive02	Bool	False
bMoveAbsoluteError02	Bool	False
wMoveAbsoluteErrorID02	Bool	W#16#0
fbMoveAbsolute01	MC_MoveAbsolute	
fbMoveAbsolute02	MC_MoveAbsolute	

Fig. 55: Variable declaration for multiple instance calls of MC_MoveAbsolute FB

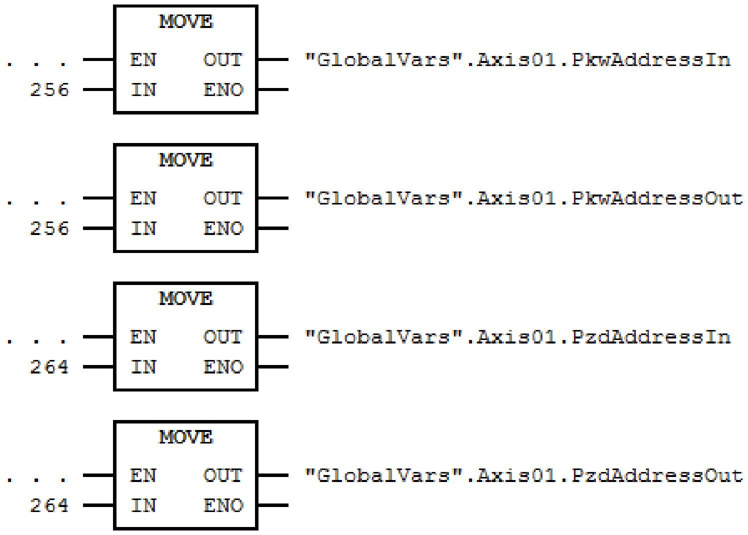

MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressIn
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressOut
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressIn
264
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressOut
264
IN
ENO

Fig. 56: Address configuration for 1st axis in FBD

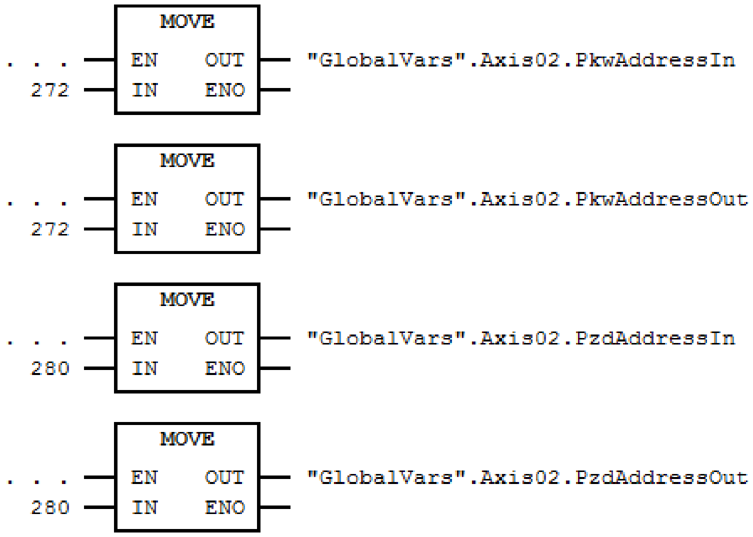

MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PkwAddressIn
272
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PkwAddressOut
272
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PzdAddressIn
280
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PzdAddressOut
280
IN
ENO

Fig. 57: Address configuration for 2nd axis in FBD

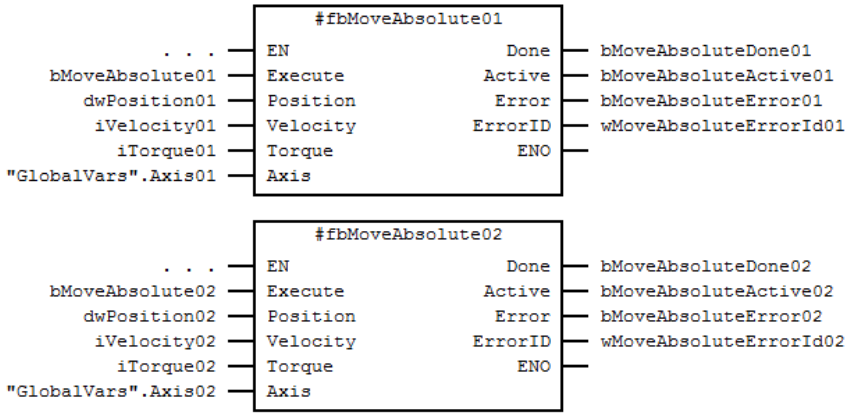


Fig. 58: FB calls of MC_MoveAbsolute (as Multiple Instances) in FBD

Note:
"GlobalVars.Axis01" and "GlobalVars.Axis02" are global instances of the UDT "AXIS_REF"

3.6.9 Error Handling

ErrorID (hex)	ErrorID (bin)	Description
16#4101	2#0100_0001_0000_0001	Error main state machine
16#4102	2#0100_0001_0000_0010	Error “in operation” state machine
16#4201	2#0100_0010_0000_0001	Invalid PZD input address
16#4202	2#0100_0010_0000_0010	Invalid PZD output address
16#4203	2#0100_0010_0000_0011	Invalid velocity
16#4204	2#0100_0010_0000_0100	Invalid torque
16#4301	2#0100_0011_0000_0001	Error while reading ZSW
16#4302	2#0100_0011_0000_0010	Error while writing STW
16#4A01	2#0100_1010_0000_0001	Error while reading drive parameter
16#4A02	2#0100_1010_0000_0010	Error writing drive parameter
16#4B01	2#0100_1011_0000_0001	Motion timeout
16#4D01	2#0100_1101_0000_0001	Wrong drive mode
16#4D02	2#0100_1101_0000_0010	Drive in state “Stopping” / instance of “MC_Stop” active
16#4D03	2#0100_1101_0000_0011	No power (no active instance of "MC_Power")

Fig. 59: Error Codes of MC_MoveAbsolute

3.7 MC_MoveVelocity

3.7.1 Brief Description

The Function Block MC_MoveVelocity commands a never ending controlled motion at a specified velocity.

3.7.2 Interface

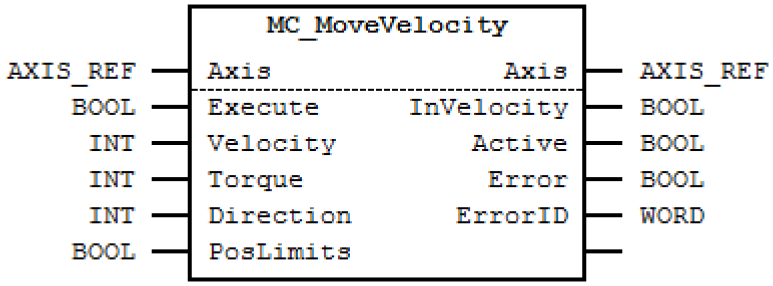


Fig. 60: MC_MoveVelocity I/O Interface Diagram

Note:
HDA family: the torque parameter is valid for 25%, 50%, 75% and 100%, all other parameter values are rounded off.
HRA08: the torque parameter is ignored.
HDA family: the velocity parameter can be changed during movement
HRA08: changing the velocity parameter during movement will cause a warning

It is not necessary to switch the drive mode (parameter 930) manually. This function block will automatically set parameter 930 to the correct value.

I/O Type	Name	Data Type	Description
VAR_IN_OUT	Axis	AXIS_REF	Data structure which contains several information for data exchange with other function blocks and communication settings for the drive. For further information on this structure please see section 3.12
VAR_INPUT	Execute	BOOL	Starts motion at rising edge
	Velocity	INT	Value of maximum velocity in % (not necessarily reached)
	Torque	INT	Value of torque in %
	Direction	INT	Direction of rotation (0 � CCW, 1 � CW)
	PosLimits	Bool	(De-)Activates position limits (parameter 1000 and 1001)
VAR_OUTPUT	InVelocity	BOOL	Target velocity reached
	Active	BOOL	Function Block is processing data
	Error	BOOL	Indicates that an error has occurred while processing the FB
	ErrorID	WORD	Error identification

Fig. 61: MC_MoveVelocity I/O Interface Description

3.7.3 Min- / Max- and Default-Values of inputs

Name	Type	Min-Value	Max-Value	Default-Value	Takeover
Execute	BOOL			FALSE	Continuous
Velocity	INT	1	100	0	Rising edge at “Execute”
Torque	INT	1	100	0	Rising edge at “Execute”
Direction	Int	0	1	0	Rising edge at “Execute”
PosLimits	Bool			False	Rising edge at “Execute”

Fig. 62: Min- / Max- and Default-Values for MC_MoveVelocity

3.7.4 Signal-Time Diagram

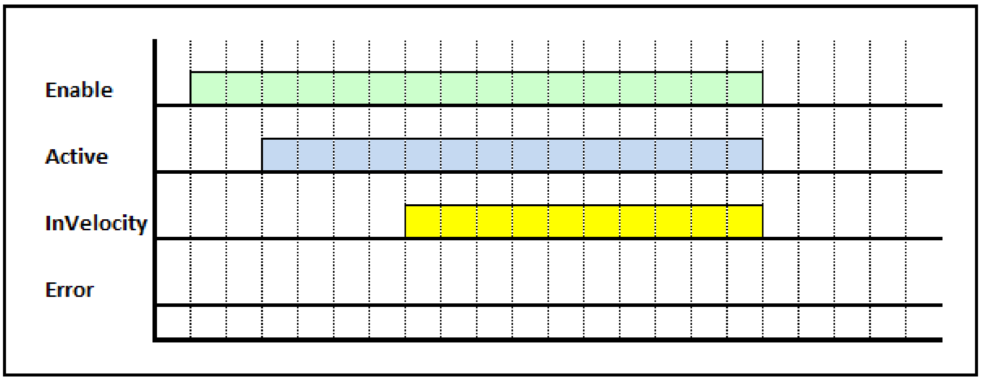


Fig. 63: Signal-Time Diagram MC_MoveVelocity for – Processing terminated successfully

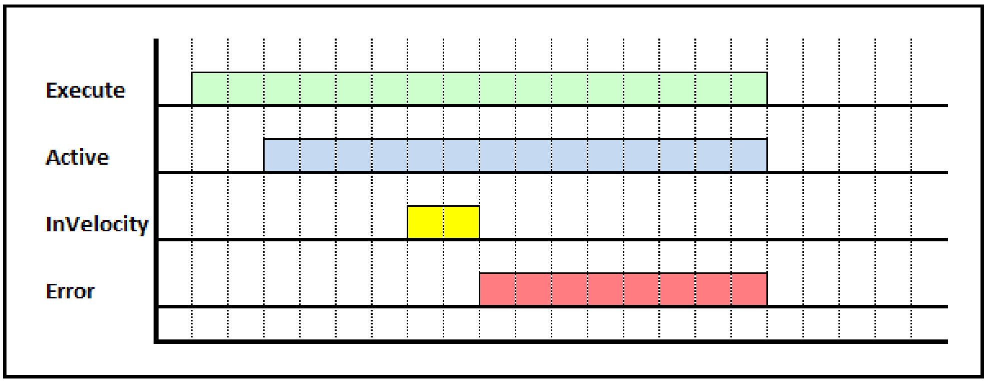


Fig. 64: Signal-Time Diagram for MC_MoveVelocity – Processing terminated by error

3.7.5 Code example for MC_MoveVelocity FB call in SCL

The code example below shows one way of calling an instance of **MC_MoveVelocity** in SCL:

```
FUNCTION_BLOCK MotionProgram
      0
      VAR
        0
        (* in- and output variables for "MC_MoveVelocity" *)
        bMoveVelocity         : BOOL := FALSE;
        iVelocity             : INT  := 0;
        iTorque               : INT  := 0;
        iDirection            : INT  := 0;
        bPosLimits            : BOOL := FALSE;

       bMoveVelocityInVel     : BOOL := FALSE;
       bMoveVelocityActive    : BOOL := FALSE;
       bMoveVelocityError     : BOOL := FALSE;
       wMoveVelocityErrorID : WORD := W#16#0000;
       0
     END_VAR
     0
BEGIN
     0
       (* PKW address configuration for 1st axis
       *) GlobalVars.Axis01.PkwAddressIn  := 256;
       GlobalVars.Axis01.PkwAddressOut := 256;

       (* PZD address configuration for 1st axis
       *) GlobalVars.Axis01.PzdAddressIn  := 264;
       GlobalVars.Axis01.PzdAddressOut := 264;

       // Note 1: "DBxxx" is an "Instance DB" for
                  this instance of "MC_MoveVelocity"
       // Note 2: "GlobalVars.Axis01" is a global
                  instance of the UDT "AXIS_REF"

       MC_MoveVelocity.DBxxx(
              Execute   := bMoveVelocity, // IN: BOOL
              Velocity := iVelocity,      // IN: INT
              Torque    := iTorque,       // IN: INT
              Direction := iDirection,    // IN: INT
              PosLimits := bPosLimits,    // IN: BOOL
              Axis      := GlobalVars.Axis01
                                         // INOUT: STRUCT
       );
       bMoveVelocityInVel   := DBxxx.InVelocity; // OUT: BOOL
       bMoveVelocityActive  := DBxxx.Active;  // OUT: BOOL
       bMoveVelocityError   := DBxxx.Error;   // OUT: BOOL
       wMoveVelocityErrorID := DBxxx.ErrorID; // OUT: WORD
       0
       END_FUNCTION_BLOCK

       DATA_BLOCK MotionProgram_DB MotionProgram

       BEGIN
       END_DAA_BLOCK
```

Fig. 65: Code example for MC_MoveVelocity FB call in SCL

3.7.6 Code example for MC_MoveVelocity FB call in SCL (Multi Instance)

The code example below shows one way of calling an instance of MC_MoveVelocity in SCL:

```
FUNCTION_BLOCK MotionProgram
0
VAR
0
    (* in- and output variables for “fbMoveVelocity01”
*)
     bMoveVelocity01     : BOOL := FALSE;
     iVelocity01         : INT := 0;
     iTorque01           : INT := 0;
     iDirection01        : INT := 0;
     bPosLimits01        : BOOL := FALSE;
     bMovVelInVel01      : BOOL := FALSE;
     bMovVelActive01     : BOOL := FALSE;
     bMovVelError01      : BOOL := FALSE;
     wMovVelErrorId01    : WORD := W#16#0000;

     (* in- and output variables for “fbMoveVelocity02”
*)
     bMoveVelocity02     : BOOL := FALSE;
     iVelocity02         : INT := 0;
     iTorque02           : INT := 0;
     iDirection02        : INT := 0;
     bPosLimits02        : BOOL := FALSE;
     bMovVelInVel02      : BOOL := FALSE;
     bMovVelActive02     : BOOL := FALSE;
     bMovVelError02      : BOOL := FALSE;
     wMovVelErrorId02    : WORD := W#16#0000;

     (* instances of “MC_MoveVelocity” *)
     fbMoveVelocity01 : MC_MoveVelocity;
     fbMoveVelocity02 : MC_MoveVelocity;

     0
  END_VAR
  0
BEGIN
     0
     (* PKW address configuration for 1st axis *)
     GlobalVars.Axis01.PkwAddressIn := 256;
     GlobalVars.Axis01.PkwAddressOut := 256;

     (* PZD address configuration for 1st axis *)
     GlobalVars.Axis01.PzdAddressIn := 264;
     GlobalVars.Axis01.PzdAddressOut := 264;

     (* PKW address configuration for 2nd axis *)
     GlobalVars.Axis02.PkwAddressIn := 272;
     GlobalVars.Axis02.PkwAddressOut := 272;

     (* PZD address configuration for 2nd axis *)
     GlobalVars.Axis02.PzdAddressIn := 280;
     GlobalVars.Axis02.PzdAddressOut := 280;
     0
```

```
      0

      // Note: “GlobalVars.Axis01” and
               “GlobalVars.Axis02” are global
      //       instances of the UDT “AXIS_REF”

      fbMoveVelocity01(
            Execute    := bMoveVelocity01, // IN: BOOL
            Velocity   := iVelocity01,    // IN: INT
            Torque     := iTorque01,      // IN: INT
            Direction  := iDirection01,   // IN: INT
            PosLimits  := bPosLimits01,   // IN: BOOL
            Axis       := GlobalVars.Axis01
                        // INOUT: STRUCT
      );

      bMovVelInVel01   := fbMoveVelocity01.InVelocity;
                 // OUT: BOOL
      bMovVelActive01  := fbMoveVelocity01.Active;
                 // OUT: BOOL
      bMovVelError01   := fbMoveVelocity01.Error;
                 // OUT: BOOL
      wMovVelErrorId01 := fbMoveVelocity01.ErrorID;
                 // OUT: WORD

      fbMoveVelocity02(
            Execute    := bMoveVelocity02,    // IN: BOOL
            Velocity   := iVelocity02,        // IN: INT
            Torque     := iTorque02,          // IN: INT
            Direction  := iDirection02,       // IN: INT
            PosLimits  := bPosLimits02,       // IN: BOOL
            Axis       := GlobalVars.Axis02
                        // INOUT: STRUCT
      );

      bMovVelInVel02   := fbMoveVelocity02.InVelocity;
                 // OUT: BOOL
      bMovVelActive02  := fbMoveVelocity02.Active;
                 // OUT: BOOL
      bMovVelError02   := fbMoveVelocity02.Error;
                 // OUT: BOOL
      wMovVelErrorId02 := fbMoveVelocity02.ErrorID;
                 // OUT: WORD
      0

END_FUNCTION_BLOCK

DATA_BLOCK FB_AxisMoveVelocity_DB FB_AxisMoveVelocity

BEGIN
END_DATA_BLOCK
```

Fig. 66: Code example for MC_MoveVelocity multiple instance FB call in SCL

3.7.7 Code example for MC_MoveVelocity FB call in FBD

Name	Data Type	Initial Value
bMoveVelocity	Bool	False
iVelocity	Int	0
iTorque	Int	0
iDirection	Int	0
bPosLimits	Bool	False
bMoveVelocityInVel	Bool	False
bMoveVelocityActive	Bool	False
bMoveVelocityError	Bool	False
wMoveVelocityErrorID	Word	DW#16#0

Fig. 67: Variable declaration for MC_MoveVelocity FB call

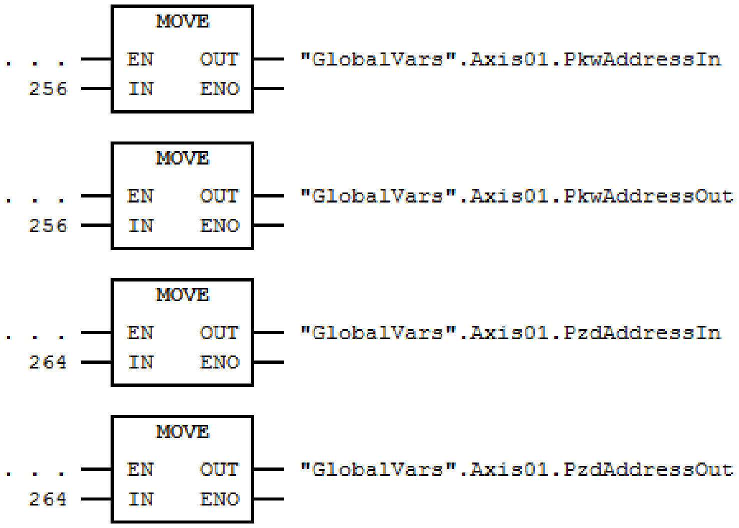

MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressIn
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressOut
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressIn
264
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressOut
264
IN
ENO

Fig. 68: Address configuration for 1st axis in FBD

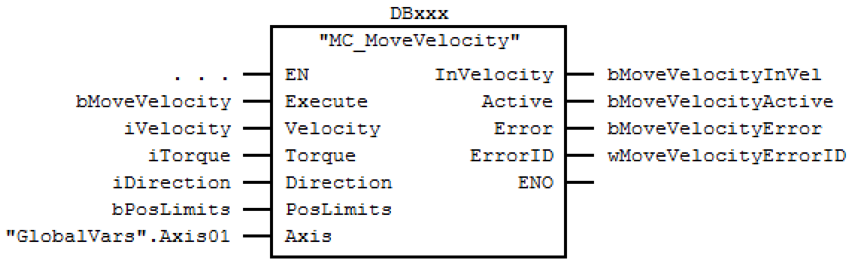


Fig. 69: MC_MoveVelocity FB call in FBD

Note 1:
“DBxxx” is an “Instance DB” for this instance of “MC_MoveVelocity”
“GlobalVars.Axis01” is a global instance of the UDT “AXIS_REF”

3.7.8 Code example for MC_MoveVelocity FB call in FBD (Multi Instance)

Name	Data Type	Initial Value
bMoveVelocity01	Bool	False
iVelocity01	Int	0
iTorque01	Int	0
iDirection01	Int	0
bPosLimits01	Bool	False
bMoveVelocityInVel01	Bool	False
bMoveVelocityActive01	Bool	False
bMoveVelocityError01	Bool	False
wMoveVelocityErrorID01	Word	DW#16#0
bMoveVelocity02	Bool	False
iVelocity02	Int	0
iTorque02	Int	0
iDirection02	Int	0
bPosLimits02	Bool	False
bMoveVelocityInVel02	Bool	False
bMoveVelocityActive02	Bool	False
bMoveVelocityError02	Bool	False
wMoveVelocityErrorID02	Word	DW#16#0
fbMoveVelocity01	MC_MoveVelocity	
fbMoveVelocity01	MC_MoveVelocity	

Fig. 70: Variable declaration for multiple instance calls of MC_MoveVelocity FB

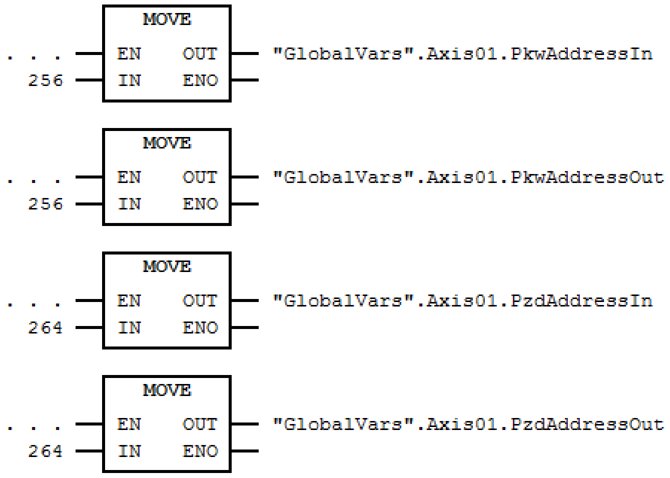

MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressIn
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressOut
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressIn
264
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressOut
264
IN
ENO

Fig. 71: Address configuration for 1st axis in FBD

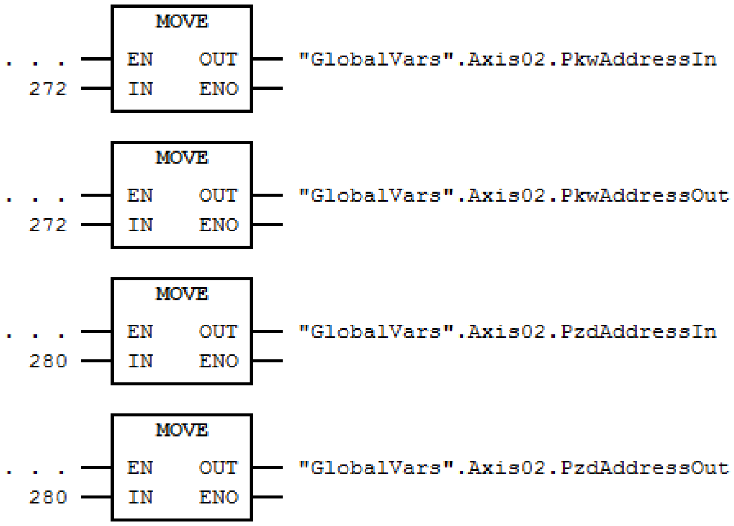

MOVE
. . .
272
EN
IN
OUT
ENO
"GlobalVars".Axis02.PkwAddressIn
MOVE
. . .
272
EN
IN
OUT
ENO
"GlobalVars".Axis02.PkwAddressOut
MOVE
. . .
280
EN
IN
OUT
ENO
"GlobalVars".Axis02.PzdAddressIn
MOVE
. . .
280
EN
IN
OUT
ENO
"GlobalVars".Axis02.PzdAddressOut

Fig. 72: Address configuration for 2nd axis in FBD

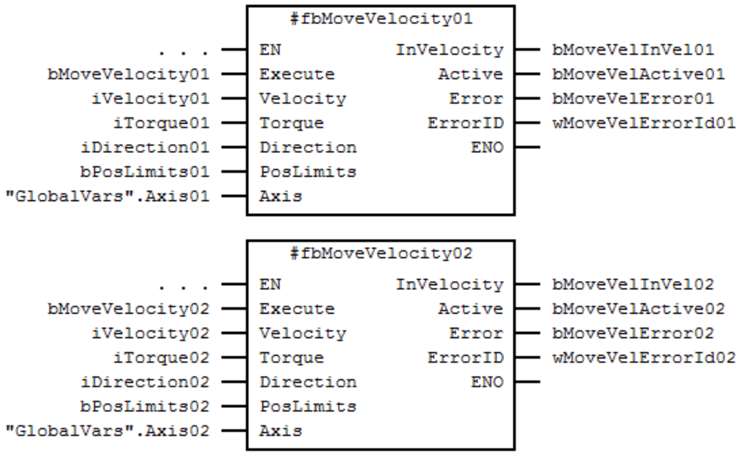


Fig. 73: FB calls of MC_MoveVelocity (as Multiple Instances) in FBD

Note:
"GlobalVars.Axis01" and "GlobalVars.Axis02" are global instances of the UDT "AXIS_REF"

3.7.9 Error Handling

ErrorID (hex)	ErrorID (bin)	Description
16#5101	2#0101_0001_0000_0001	Error main state machine
16#5102	2#0101_0001_0000_0010	Error “in operation” state machine
16#5201	2#0101_0010_0000_0001	Invalid PZD input address
16#5202	2#0101_0010_0000_0010	Invalid PZD output address
16#5203	2#0101_0010_0000_0011	Invalid velocity
16#5204	2#0101_0010_0000_0100	Invalid torque
16#5301	2#0101_0011_0000_0001	Error while reading ZSW
16#5302	2#0101_0011_0000_0010	Error while writing STW
16#5A01	2#0101_1010_0000_0001	Error while reading drive parameter
16#5A02	2#0101_1010_0000_0010	Error writing drive parameter
16#5D01	2#0101_1101_0000_0001	Wrong drive mode
16#5D02	2#0101_1101_0000_0010	Drive in state “Stopping” / instance of “MC_Stop” active
16#5D03	2#0101_1101_0000_0011	No power (no active instance of "MC_Power")

Fig. 74: Error Codes of MC_MoveVelocity

3.8 MC_ReadAxisError

3.8.1 Brief Description

The Function Block **MC_ReadAxisError** reads the error information of the addressed axis.

3.8.2 Interface

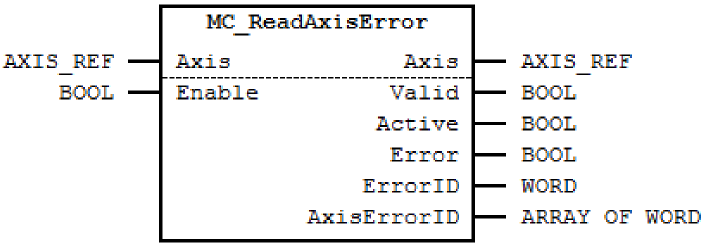


Fig. 75: MC_ReadAxisError I/O Interface Diagram

I/O Type	Name	Data Type	Description
VAR_IN_OUT	Axis	AXIS_REF	Data structure which contains several information for data exchange with other function blocks and communication settings for the drive. For further information on this structure please see section 3.12

VAR_INPUT	Enable	BOOL	Read the value of parameter 945 continuously while “Enable” is TRUE
VAR_OUTPUT	Valid	BOOL	Indicates if a drive error occurred
	Active	BOOL	Function Block is processing data
	Error	BOOL	Indicates that an error has occurred while processing the FB
	ErrorID	WORD	Error identification
	AxisErrorID	ARRAY [0..7] OF WORD	Array with error information (values of drive parameter 945)

Fig. 76: MC_ReadAxisError I/O Interface Description

3.8.3 Min- / Max- and Default-Values of inputs

Name	Type	Min-Value	Max-Value	Default-Value	Takeover
Enable	BOOL			FALSE	Continuous

Fig. 77: Min- / Max- and Default-Values for MC_ReadAxisError

3.8.4 Signal-Time Diagram

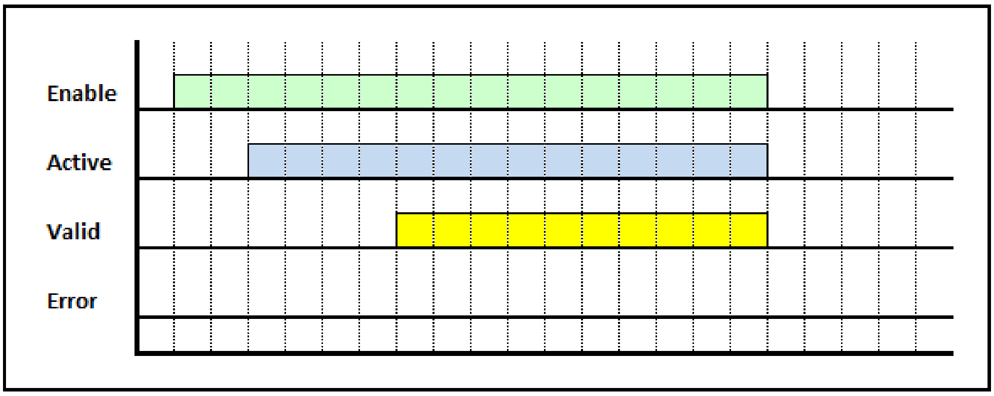


Fig. 78: Signal-Time Diagram MC_ReadAxisError for – Processing terminated successfully

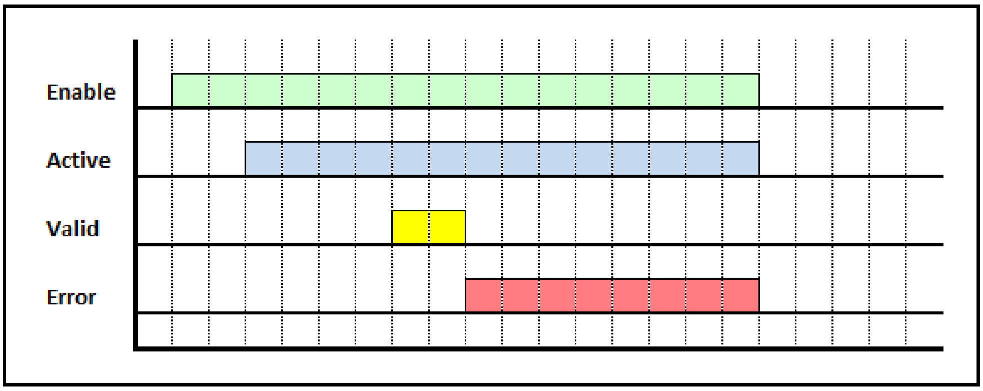


Fig. 79: Signal-Time Diagram for MC_ReadAxisError – Processing terminated by error

3.8.5 Code example for MC_ReadAxisError FB call in SCL

The code example below shows one way of calling an instance of **MC_ReadAxisError** in SCL:

```
FUNCTION_BLOCK MotionProgram
      0
      VAR
        0

        (* in- and output variables for "MC_ReadAxisError" *)
        bReadError    : BOOL := FALSE;

        bValid        : BOOL := FALSE;
        bActive       : BOOL := FALSE;
        bError        : BOOL := FALSE;
        wErrorID      : WORD := W#16#0000;
        arAxisErrorID : ARRAY[0..7] OF WORD;

         0
       END_VAR
       0
BEGIN
      0

      (* PKW address configuration for 1st axis *)
      GlobalVars.Axis01.PkwAddressIn     := 256;
      GlobalVars.Axis01.PkwAddressOut := 256;

      (* PZD address configuration for 1st axis *)
      GlobalVars.Axis01.PzdAddressIn     := 264;
      GlobalVars.Axis01.PzdAddressOut := 264;

      // Note 1: "DBxxx" is an "Instance DB" for
                  this instance of "MC_ReadAxisError"
      // Note 2: "GlobalVars.Axis01" is a global
                  instance of the UDT "AXIS_REF"

      MC_ReadAxisError.DBxxx(
      Enable     := bReadError,    // IN: BOOL
      Axis   := GlobalVars.Axis01    // INOUT: STRUCT
      );

      bValid        := DBxxx.Valid;     // OUT: BOOL
      bActive       := DBxxx.Active;    // OUT: BOOL
      bError        := DBxxx.Error;     // OUT: BOOL
      wErrorID      := DBxxx.ErrorID;   // OUT: WORD
      arAxisErrorID := DBxxx.AxisErrorID;// OUT: ARRAY

      0
END_FUNCTION_BLOCK


DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 80: Code example for MC_ReadAxisError FB call in SCL

3.8.6 Code example for MC_ReadAxisError FB call in SCL (Multi Instance)

The code example below shows one way of calling multiple instances of **MC_ReadAxisError** in SCL:

```
FUNCTION_BLOCK MotionProgram
      0
      VAR
        0
        (*in- and output variables for "fbReadAxisError01"*)
        bReadError01     : BOOL := FALSE;
        bValid01         : BOOL := FALSE;
        bActive01        : BOOL := FALSE;
        bError01         : BOOL := FALSE;
        wErrorID01       : WORD := W#16#0000;
        arAxisErrorID01  : ARRAY[0..7] OF WORD;

        (*in- and output variables for "fbReadAxisError02"*)
        bReadError02     : BOOL := FALSE;
        bValid02         : BOOL := FALSE;
        bActive02        : BOOL := FALSE;
        bError02         : BOOL := FALSE;
        wErrorID02       : WORD := W#16#0000;
        arAxisErrorID02  : ARRAY[0..7] OF WORD;

        (*instances of "MC_ReadAxisError" *)
        fbReadAxisError01 : MC_ReadAxisError;
        fbReadAxisError02 : MC_ReadAxisError;
        0
      END_VAR
      0
BEGIN
      0
      (* PKW address configuration for 1st axis *)
      GlobalVars.Axis01.PkwAddressIn   := 256;
      GlobalVars.Axis01.PkwAddressOut  := 256;

      (* PZD address configuration for 1st axis *)
      GlobalVars.Axis01.PzdAddressIn   := 264;
      GlobalVars.Axis01.PzdAddressOut  := 264;

      (* PKW address configuration for 2nd axis *)
      GlobalVars.Axis02.PkwAddressIn   := 272;
      GlobalVars.Axis02.PkwAddressOut  := 272;

      (* PZD address configuration for 2nd axis *)
      GlobalVars.Axis02.PzdAddressIn   := 280;
      GlobalVars.Axis02.PzdAddressOut  := 280;
      0
```

```
    0

      // Note: "GlobalVars.Axis01" and "GlobalVars.Axis02"
              are global instances of the UDT "AXIS_REF"

      fbReadAxisError01(
            Enable   := bReadError01,      // IN: BOOL
            Axis     := GlobalVars.Axis01 // INOUT: STRUCT
      );

      bValid01         := fbReadAxisError01.Valid;
                                          // OUT: BOOL
      bActive01        := fbReadAxisError01.Active;
                                          // OUT: BOOL
      bError01         := fbReadAxisError01.Error;
                                          // OUT: BOOL
      wErrorID01       := fbReadAxisError01.ErrorID;
                                          // OUT: WORD
      arAxisErrorID01  := fbReadAxisError01.AxisErrorID;
                                          // OUT: ARRAY


      fbReadAxisError02(
            Enable   := bReadError02,       // IN: BOOL
            Axis     := GlobalVars.Axis02
                                        // INOUT:STRUCT
      );

      bValid02         := fbReadAxisError02.Valid;
                                          // OUT: BOOL
      bActive02        := fbReadAxisError02.Active;
                                          // OUT: BOOL
      bError02         := fbReadAxisError02.Error;
                                          // OUT: BOOL
      wErrorID02       := fbReadAxisError02.ErrorID;
                                          // OUT: WORD
      arAxisErrorID02  := fbReadAxisError02.AxisErrorID;
                                          // OUT: ARRAY
      0

END_FUNCTION_BLOCK


DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 81: Code example for MC_ReadAxisError multiple instance FB call in SCL

3.8.7 Code example for MC_ReadAxisError FB call in FBD

Name	Data Type	Initial Value
bReadError	Bool	False
bValid	Bool	False
bActive	Bool	False

bError	Bool	False
wErrorID	Word	W#16#0
arAxisErrorID	Array [0..7] of Word	8 (W#16#0)

Fig. 82: Variable declaration for MC_ReadAxisError FB call

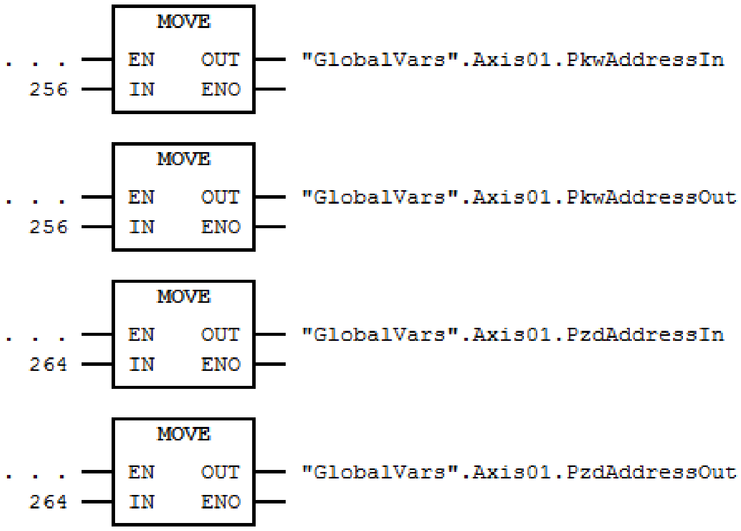

MOVE
. . . EN OUT "GlobalVars".Axis01.PkwAddressIn
256 IN ENO
MOVE
. . . EN OUT "GlobalVars".Axis01.PkwAddressOut
256 IN ENO
MOVE
. . . EN OUT "GlobalVars".Axis01.PzdAddressIn
264 IN ENO
MOVE
. . . EN OUT "GlobalVars".Axis01.PzdAddressOut
264 IN ENO

Fig. 83: Address configuration for 1st axis in FBD

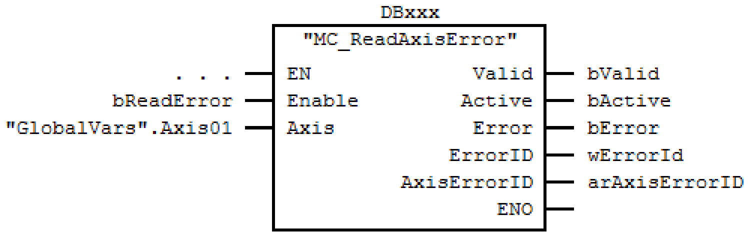

DBxxx
"MC_ReadAxisError"
. . . EN Valid bValid
bReadError Enable Active bActive
"GlobalVars".Axis01 Axis Error bError
ErrorID wErrorId
AxisErrorID arAxisErrorID
ENO

Fig. 84: MC_ReadAxisError FB call in FBD

Note 1:
“DBxxx” is an “Instance DB” for this instance of “MC_ReadAxisError”
“GlobalVars.Axis01” is a global instance of the UDT “AXIS_REF”

3.8.8 Code example for MC_ReadAxisError FB call in FBD (Multi Instance)

Name	Data Type	Initial Value
bReadError01	Bool	False
bValid01	Bool	False
bActive01	Bool	False
bError01	Bool	False
wErrorID01	Word	W#16#0

arAxisErrorID01	Array [0..7] of Word	8 (W#16#0)
bReadError02	Bool	False
bValid02	Bool	False
bActive02	Bool	False
bError02	Bool	False
wErrorID02	Word	W#16#0
arAxisErrorID02	Array [0..7] of Word	8 (W#16#0)
fbReadAxisError01	MC_ReadAxisError	
fbReadAxisError02	MC_ReadAxisError	

Fig. 85: Variable declaration for multiple instance calls of MC_ReadAxisError FB

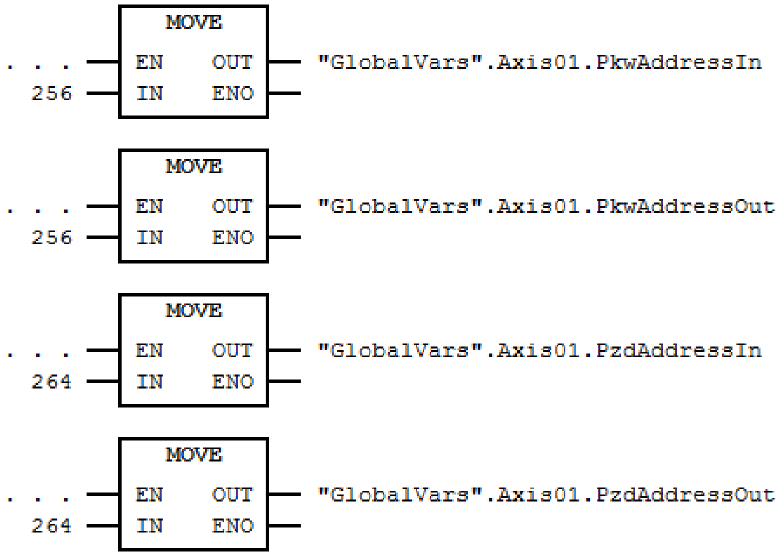

MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressIn
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressOut
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressIn
264
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressOut
264
IN
ENO

Fig. 86: Address configuration for 1st axis in FBD

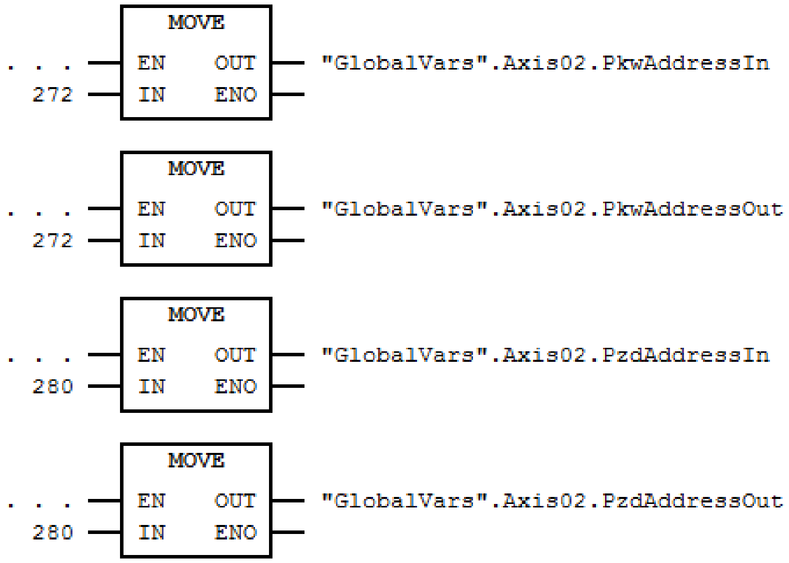

MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PkwAddressIn
272
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PkwAddressOut
272
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PzdAddressIn
280
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PzdAddressOut
280
IN
ENO

Fig. 87: Address configuration for 2nd axis in FBD

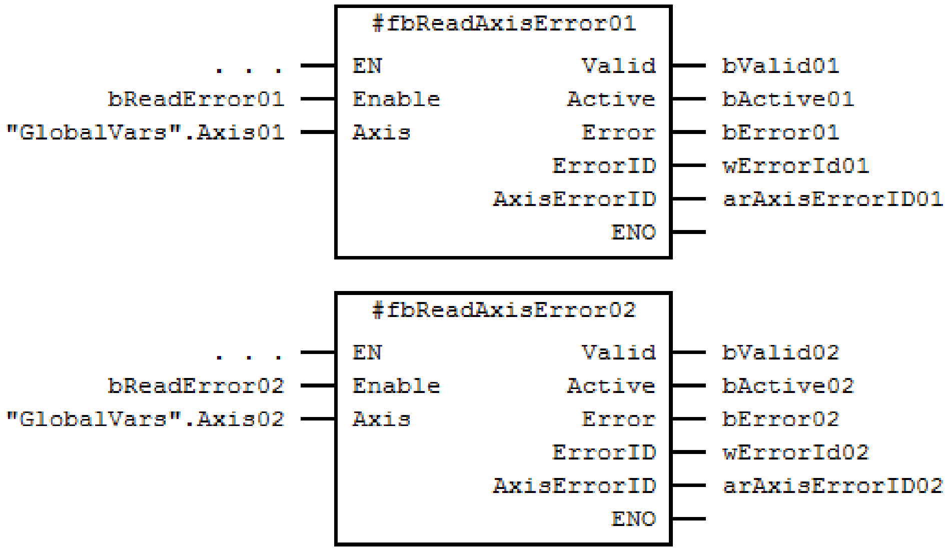


Fig. 88: FB calls of MC_ReadAxisError (as Multiple Instances) in FBD

Note:
“GlobalVars.Axis01” and “GlobalVars.Axis02” are global instances of the UDT “AXIS_REF”

3.8.9 Error Handling

ErrorID (hex)	ErrorID (bin)	Description
16#6101	2#0110_0001_0000_0001	Error main state machine
16#6102	2#0110_0001_0000_0010	Error "in operation" state machine
16#6201	2#0110_0010_0000_0001	Invalid PKW input address
16#6202	2#0110_0010_0000_0010	Invalid PKW output address
16#6303	2#0110_0011_0000_0011	Error while reading PKW
16#6304	2#0110_0011_0000_0100	error while writing PKW
16#6E01	2#0110_1110_0000_0001	Communication timeout
16#6F01	2#0110_1111_0000_0001	PKW channel currently used by another Function Block
16#6F02	2#0110_1111_0000_0010	Internal error (invalid Subindex)

Fig. 89: Error Codes of MC_ReadAxisError

3.9 MC_ReadParameter

3.9.1 Brief Description

The Function Block MC_ReadParameter returns the value of a vendor specific parameter from the specified axis.

3.9.2 Interface

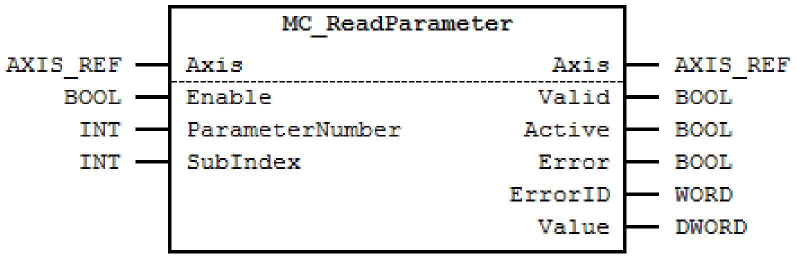


Fig. 90: MC_ReadParameter I/O Interface Diagram

I/O Type	Name	Data Type	Description
VAR_IN_OUT	Axis	AXIS_REF	Data structure which contains several information for data exchange with other function blocks and communication settings for the drive. For further information on this structure please see section 3.12
VAR_INPUT	Enable	BOOL	Get the parameter value continuously while "Enable" is TRUE
	ParameterNumber	INT	Number of the desired parameter

	SubIndex	INT	Number of desired parameter SubIndex
VAR_OUTPUT	Valid	BOOL	Valid parameter value is available
	Active	BOOL	Function Block is processing data
	Error	BOOL	Indicates that an error has occurred while processing the FB
	ErrorID	WORD	Error identification
	Value	DWORD	Value of requested parameter

Fig. 91: MC_ReadParameter I/O Interface Description

3.9.3 Min- / Max- and Default-Values of inputs

Name	Type	Min-Value	Max-Value	Default-Value	Takeover
Enable	BOOL			FALSE	Continuous
ParameterNumber	INT	0	2047	0	Rising edge at “Enable”
SubIndex	INT	0		0	Rising edge at “Enable”

Fig. 92: Min- / Max- and Default-Values for MC_ReadParameter

3.9.4 Signal-Time Diagram

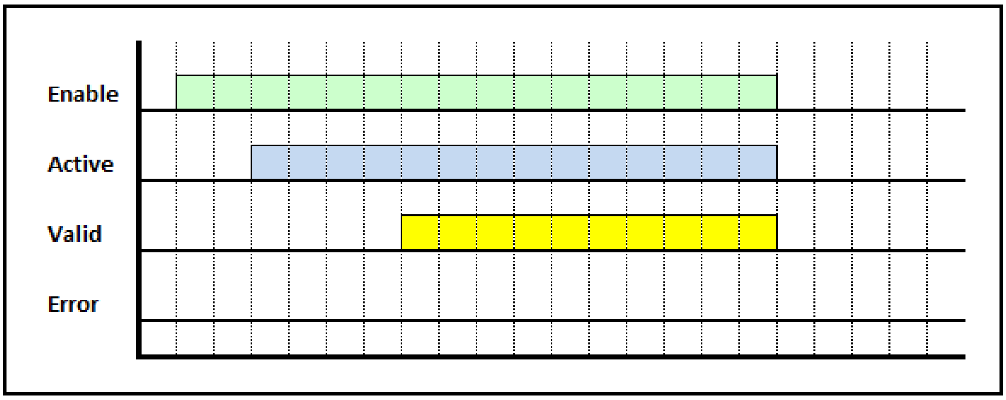


Fig. 93: Signal-Time Diagram MC_ReadParameter for – Processing terminated successfully

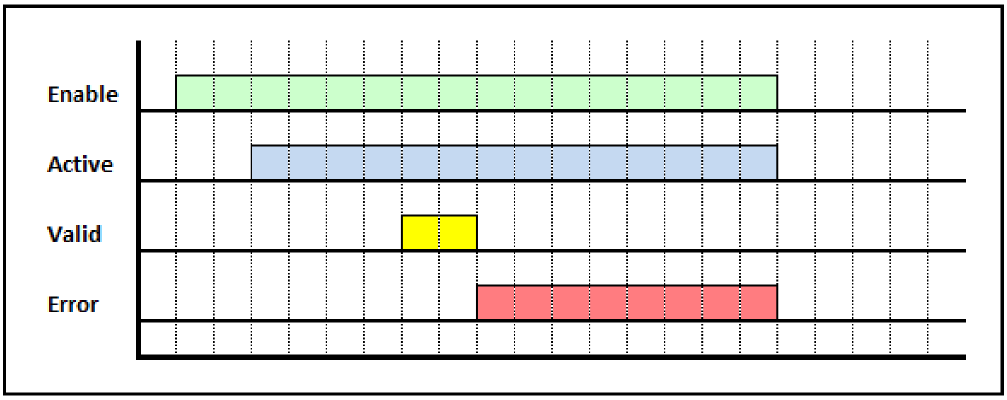
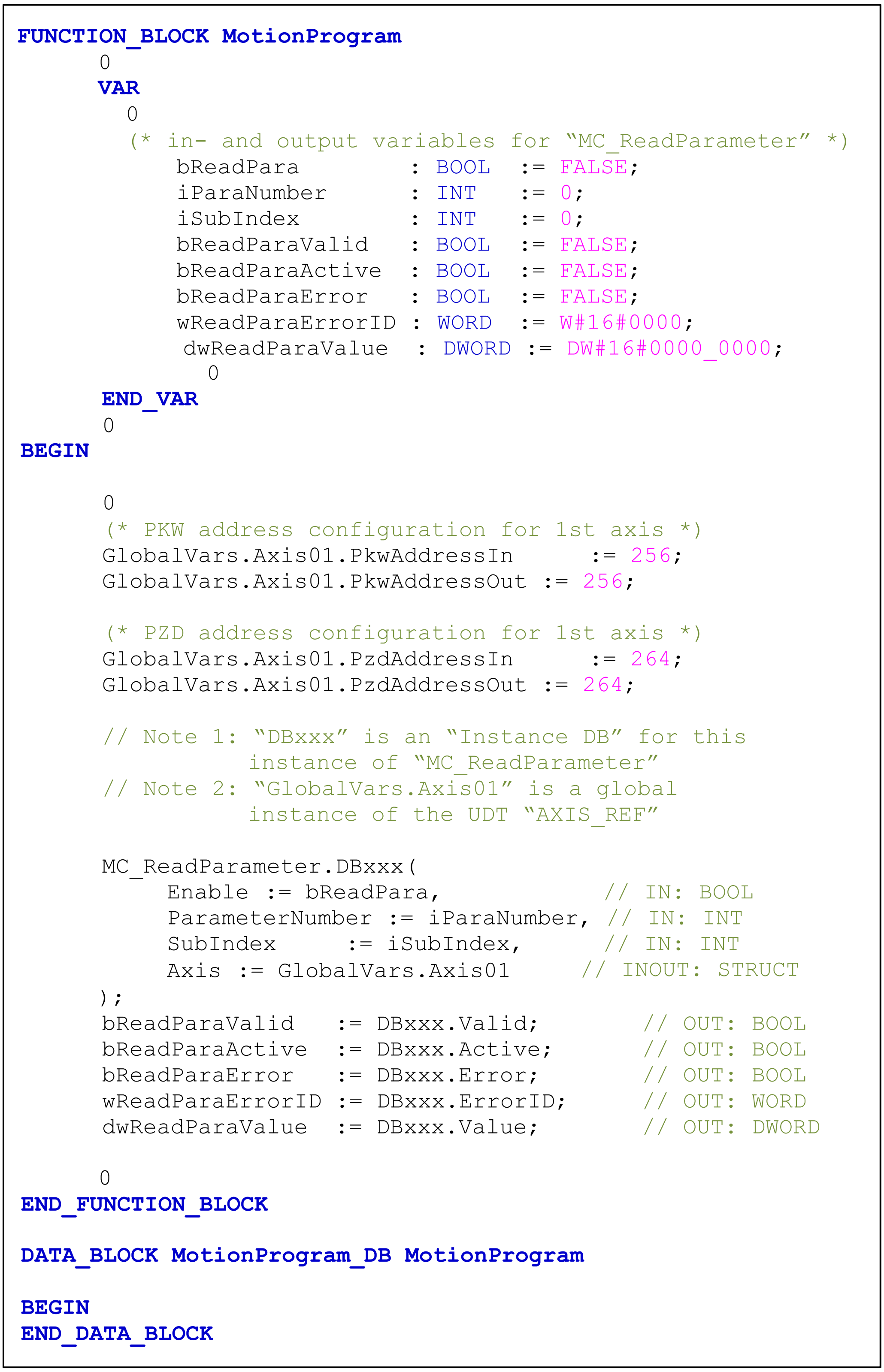


Fig. 94: Signal-Time Diagram for MC_ReadParameter – Processing terminated by error

3.9.5 Code example for MC_ReadParameter FB call in SCL

The code example below shows one way of calling an instance of MC_ReadParameter in SCL:



```
FUNCTION_BLOCK MotionProgram
      0
      VAR
        0
        (* in- and output variables for "MC_ReadParameter" *)
           bReadPara         : BOOL   := FALSE;
           iParaNumber       : INT    := 0;
           iSubIndex         : INT    := 0;
           bReadParaValid    : BOOL   := FALSE;
           bReadParaActive   : BOOL   := FALSE;
           bReadParaError    : BOOL   := FALSE;
           wReadParaErrorID  : WORD   := W#16#0000;
           dwReadParaValue   : DWORD  := DW#16#0000_0000;
             0
      END_VAR
      0
BEGIN

      0
      (* PKW address configuration for 1st axis *)
      GlobalVars.Axis01.PkwAddressIn     := 256;
      GlobalVars.Axis01.PkwAddressOut := 256;

      (* PZD address configuration for 1st axis *)
      GlobalVars.Axis01.PzdAddressIn     := 264;
      GlobalVars.Axis01.PzdAddressOut := 264;

      // Note 1: "DBxxx" is an "Instance DB" for this
                 instance of "MC_ReadParameter"
      // Note 2: "GlobalVars.Axis01" is a global
                 instance of the UDT "AXIS_REF"

      MC_ReadParameter.DBxxx(
           Enable := bReadPara,            // IN: BOOL
           ParameterNumber := iParaNumber, // IN: INT
           SubIndex     := iSubIndex,      // IN: INT
           Axis  := GlobalVars.Axis01     // INOUT: STRUCT
      );
      bReadParaValid    := DBxxx.Valid;        // OUT: BOOL
      bReadParaActive   := DBxxx.Active;       // OUT: BOOL
      bReadParaError    := DBxxx.Error;        // OUT: BOOL
      wReadParaErrorID  := DBxxx.ErrorID;      // OUT: WORD
      dwReadParaValue   := DBxxx.Value;        // OUT: DWORD

      0
END_FUNCTION_BLOCK

DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 95: Code example for MC_ReadParameter FB call in SCL

3.9.6 Code example for MC_ReadParameter FB call in SCL (Multi Instance)

The code example below shows one way of calling multiple instances of MC_ReadParameter in SCL:

```
FUNCTION_BLOCK MotionProgram
      0
      VAR
        0
        (*in- and output variables for “fbReadParameter01”*)
        bReadPara01          : BOOL  := FALSE;
        iParaNumber01        : INT   := 0; iSubIndex01  :
        INT                  := 0; bReadParaValid01     :
        BOOL                 := FALSE; bReadParaActive01
                             : BOOL  := FALSE;
        bReadParaError01     : BOOL  := FALSE;
        wReadParaErrorID01 : WORD  := W#16#0000;
        dwReadParaValue01  : DWORD := DW#16#0000_0000;

        (*in- and output variables for “fbReadParameter02”*)
        bReadPara02          : BOOL  := FALSE;
        iParaNumber02        : INT   := 0; iSubIndex02
        : INT                        := 0;
        bReadParaValid02     : BOOL  := FALSE;
        bReadParaActive02    : BOOL  := FALSE;
        bReadParaError02     : BOOL  := FALSE;
        wReadParaErrorID02 : WORD  := W#16#0000;
        dwReadParaValue02  : DWORD := DW#16#0000_0000;

        (* instances of “MC_ReadParameter” *)
        fbReadParameter01 : MC_ReadParameter;
        fbReadParameter02 : MC_ReadParameter;
        0
      END_VAR
      0
BEGIN
      0
      (* PKW address configuration for 1st axis *)
      GlobalVars.Axis01.PkwAddressIn   := 256;
      GlobalVars.Axis01.PkwAddressOut  := 256;

      (* PZD address configuration for 1st axis *)
      GlobalVars.Axis01.PzdAddressIn   := 264;
      GlobalVars.Axis01.PzdAddressOut  := 264;

      (* PKW address configuration for 2nd axis *)
      GlobalVars.Axis02.PkwAddressIn   := 272;
      GlobalVars.Axis02.PkwAddressOut  := 272;

      (* PZD address configuration for 2nd axis *)
      GlobalVars.Axis02.PzdAddressIn   := 280;
      GlobalVars.Axis02.PzdAddressOut  := 280;

      0
      0

      // Note: "GlobalVars.Axis01" and "GlobalVars.Axis02"
            are global instances of the UDT "AXIS_REF"

      fbReadParameter01(
            Enable          := bReadPara01,    // IN: BOOL
            ParameterNumber := iParaNumber01, // IN: INT
            SubIndex        := iSubIndex01,    // IN: INT
            Axis            := GlobalVars.Axis01
                                         // INOUT: STRUCT
      );

      bReadParaValid01   := fbReadParameter01.Valid;
                                        // OUT: BOOL
      bReadParaActive01 := fbReadParameter01.Active;
                                        // OUT: BOOL
      bReadParaError01   := fbReadParameter01.Error;
                                        // OUT: BOOL
      wReadParaErrorID01 := fbReadParameter01.ErrorID;
                                        // OUT: WORD
      dwReadParaValue01  := fbReadParameter01.Value;
                                       // OUT: DWORD

      fbReadParameter02(
            Enable          := bReadPara02,    // IN: BOOL
            ParameterNumber := iParaNumber02, // IN: INT
            SubIndex        := iSubIndex02,    // IN: INT
            Axis            := GlobalVars.Axis02
                                         // INOUT: STRUCT
      );

      bReadParaValid02   := fbReadParameter02.Valid;
                                        // OUT: BOOL
      bReadParaActive02 := fbReadParameter02.Active;
                                        // OUT: BOOL
      bReadParaError02   := fbReadParameter02.Error;
                                        // OUT: BOOL
      wReadParaErrorID02 := fbReadParameter02.ErrorID;
                                        // OUT: WORD
      dwReadParaValue02  := fbReadParameter02.Value;
                                        // OUT: DWORD
      0

END_FUNCTION_BLOCK


DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 96: Code example for MC_ReadParameter multiple instance FB call in SCL

3.9.7 Code example for MC_ReadParameter FB call in FBD

Name	Data Type	Initial Value
bReadPara	Bool	False
iParaNumber	Int	0
iSubIndex	Int	0
bReadParaValid	Bool	False
bReadParaActive	Bool	False
bReadParaError	Bool	False
bReadParaErrorID	Word	W#16#0
bReadParaValue	DWord	DW#16#0

Fig. 97: Variable declaration for MC_ReadParameter FB call

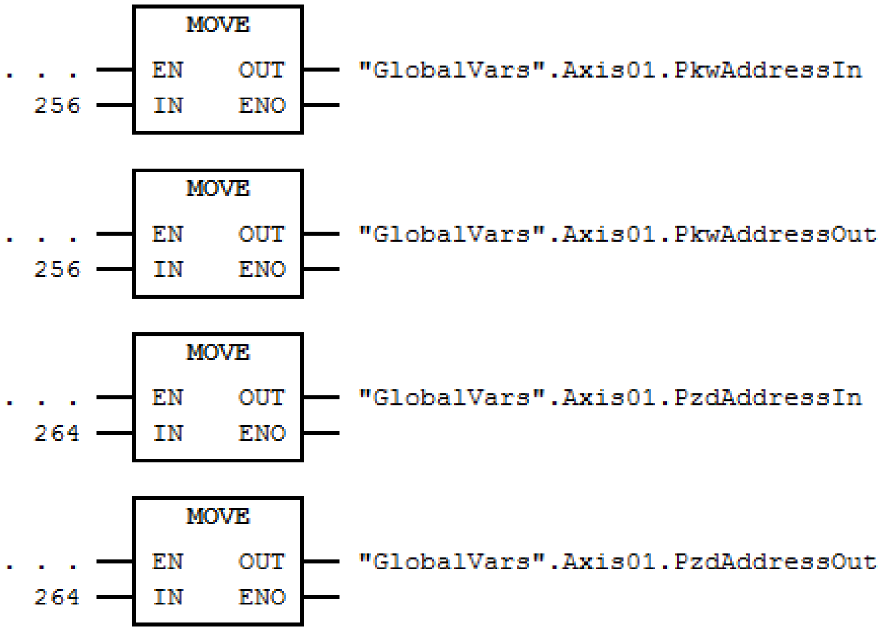

MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressIn
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressOut
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressIn
264
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressOut
264
IN
ENO

Fig. 98: Address configuration for 1st axis in FBD

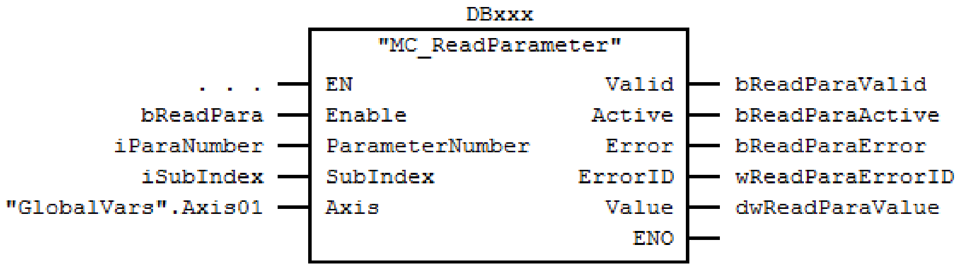


Fig. 99: MC_ReadParameter FB call in FBD

Note 1:

"DBxxx" is an "Instance DB" for this instance of "MC_ReadParameter"
"GlobalVars.Axis01" is a global instance of the UDT "AXIS_REF"

3.9.8 Code example for MC_ReadParameter FB call in FBD (Multi Instance)

Name	Data Type	Initial Value
bReadPara01	Bool	False
iParaNumber01	Int	0
iSubIndex01	Int	0
bReadParaValid01	Bool	False
bReadParaActive01	Bool	False
bReadParaError01	Bool	False
bReadParaErrorID01	Word	W#16#0
bReadParaValue01	DWord	DW#16#0
bReadPara02	Bool	False
iParaNumber02	Int	0
iSubIndex02	Int	0
bReadParaValid02	Bool	False
bReadParaActive02	Bool	False
bReadParaError02	Bool	False
bReadParaErrorID02	Word	W#16#0
bReadParaValue02	DWord	DW#16#0
fbReadParameter01	MC_ReadParameter	
fbReadParameter01	MC_ReadParameter	

Fig. 100: Variable declaration for multiple instance calls of MC_ReadParameter FB

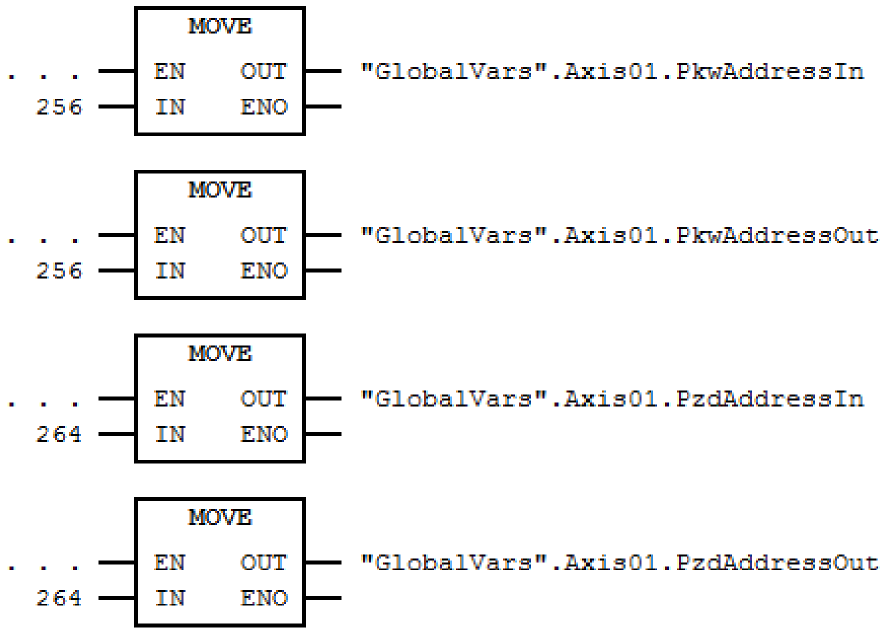

MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressIn
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressOut
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressIn
264
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressOut
264
IN
ENO

Fig. 101: Address configuration for 1st axis in FBD

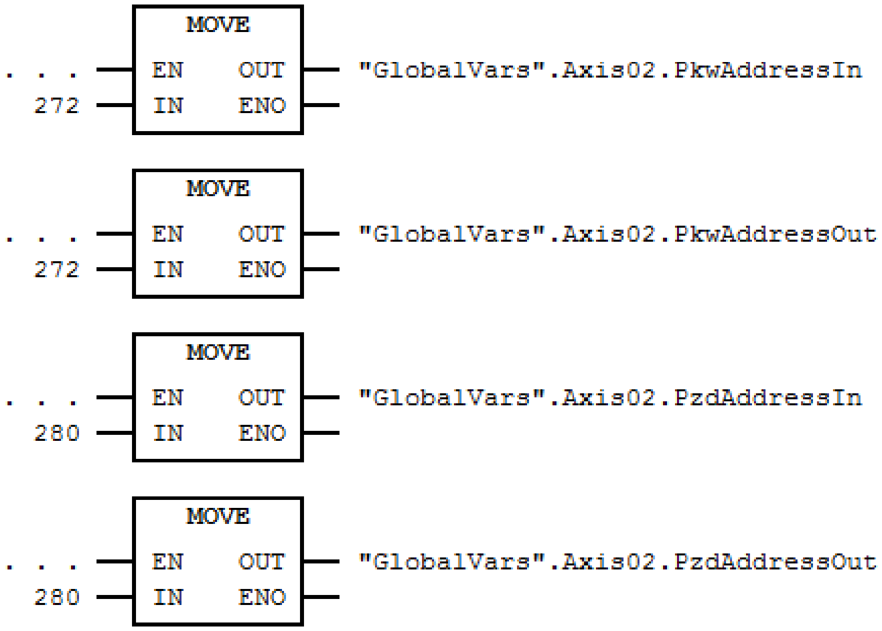

MOVE
. . .
272
EN
IN
OUT
ENO
"GlobalVars".Axis02.PkwAddressIn
MOVE
. . .
272
EN
IN
OUT
ENO
"GlobalVars".Axis02.PkwAddressOut
MOVE
. . .
280
EN
IN
OUT
ENO
"GlobalVars".Axis02.PzdAddressIn
MOVE
. . .
280
EN
IN
OUT
ENO
"GlobalVars".Axis02.PzdAddressOut

Fig. 102: Address configuration for 2nd axis in FBD

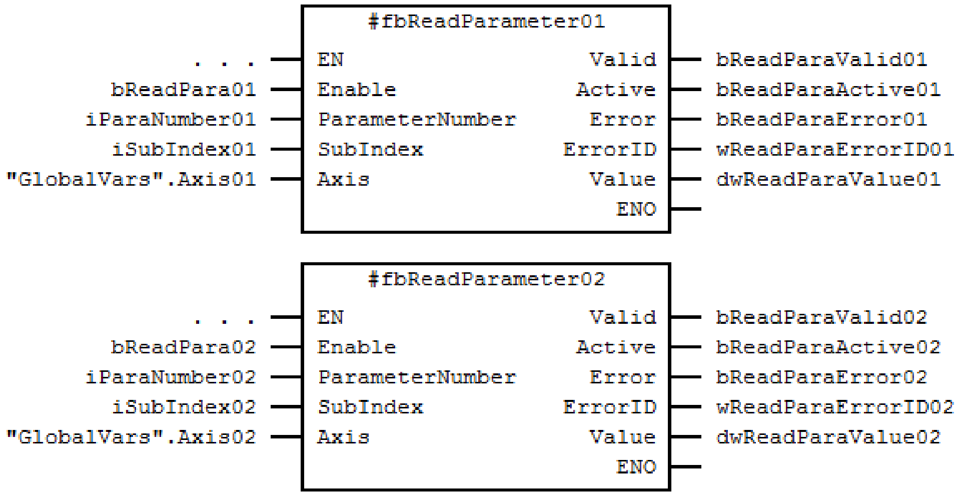


Fig. 103: FB calls of MC_ReadParameter (as Multiple Instances) in FBD

Note:
“GlobalVars.Axis01” and “GlobalVars.Axis02” are global instances of the UDT “AXIS_REF”

3.9.9 Error Handling

ErrorID (hex)	ErrorID (bin)	Description
16#7101	2#0111_0001_0000_0001	Error main state machine
16#7102	2#0111_0001_0000_0010	Error “in operation” state machine
16#7201	2#0111_0010_0000_0001	Invalid PKW input address
16#7202	2#0111_0010_0000_0010	Invalid PKW output address
16#7203	2#0111_0010_0000_0011	Invalid Parameter Number
16#7204	2#0111_0010_0000_0100	Invalid Subindex
16#7205	2#0111_0010_0000_0101	Subindex is greater than parameter array size
16#7301	2#0111_0011_0000_0001	Error while receiving PKW answer from drive
16#7302	2#0111_0011_0000_0010	Error while sending PKW request to drive
16#7A01	2#0111_1010_0000_0001	Communication timeout
16#7B01	2#0111_1011_0000_0001	PKW channel currently used by another Function Block
Forthefollowingerrors: **X** = 4 (**YYYY** = 0100) : Error while Parameter Read Request **X** = 5 (**YYYY** = 0101) : Error while Parameter Array Read Request **X** = 7 (**YYYY** = 0111) : Error while Parameter Array Check		
16#7X00	2#0111_**YYYY**_0000_0000	Impermissible parameter number
16#7X01	2#0111_**YYYY**_0000_0001	Parameter value cannot be changed
16#7X02	2#0111_**YYYY**_0000_0010	Lower or upper value limit violated
16#7X03	2#0111_**YYYY**_0000_0011	Faulty sub-index
16#7X04	2#0111_**YYYY**_0000_0100	Not an array
16#7X05	2#0111_**YYYY**_0000_0101	Wrong data type
16#7X06	2#0111_**YYYY**_0000_0110	Setting not permitted (can only be reset)
16#7X07	2#0111_**YYYY**_0000_0111	Descriptive element cannot be changed
16#7X08	2#0111_**YYYY**_0000_1000	PPO write requested in the IR not present
16#7X09	2#0111_**YYYY**_0000_1001	Descriptive data not present
16#7X0A	2#0111_**YYYY**_0000_1010	Access group wrong
16#7X0B	2#0111_**YYYY**_0000_1011	No operating authority
16#7X0C	2#0111_**YYYY**_0000_1100	Wrong password
16#7X0D	2#0111_**YYYY**_0000_1101	Illegible text in cyclic traffic
16#7X0E	2#0111_**YYYY**_0000_1110	Illegible name in cyclic traffic
16#7X0F	2#0111_**YYYY**_0000_1111	No text array present
16#7X10	2#0111_**YYYY**_0001_0000	Missing PPO write
16#7X11	2#0111_**YYYY**_0001_0001	Job cannot be executed because of operating status
16#7X12	2#0111_**YYYY**_0001_0010	Other error
16#7X13	2#0111_**YYYY**_0001_0011	Illegible date in cyclic traffic

Fig. 104: Error Codes of MC_ReadParameter

3.10 MC_WriteParameter

3.10.1 Brief Description

The Function Block MC_WriteParameter modifies the value of a vendor specific parameter in the specified axis.

3.10.2 Interface

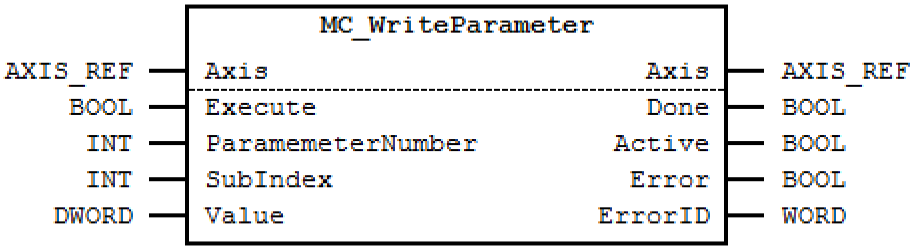


Fig. 105: MC_WriteParameter I/O Interface Diagram

I/O Type	Name	Data Type	Description
VAR_IN_OUT	Axis	AXIS_REF	Data structure which contains several information for data exchange with other function blocks and communication settings for the drive. For further information on this structure please see section 3.12
VAR_INPUT	Execute	BOOL	Writes the new parameter value at rising edge
	ParameterNumber	INT	Number of the desired parameter
	SubIndex	INT	Number of desired parameter subindex
	Value	DWORD	New value to be written to the specified parameter
VAR_OUTPUT	Done	BOOL	The new value has been successfully written to the drive
	Active	BOOL	Function Block is processing data
	Error	BOOL	Indicates that an error has occurred while processing the FB
	ErrorID	WORD	Error identification

Fig. 106: MC_WriteParameter I/O Interface Description

3.10.3 Min- / Max- and Default-Values of inputs

Name	Type	Min-Value	Max-Value	Default-Value	Takeover
Execute	BOOL			FALSE	Continuous
ParameterNumber	INT	0	2047	0	Rising edge at “Execute”
SubIndex	INT	0		0	Rising edge at “Execute”
Value	DWORD			DW#16#0	Rising edge at “Execute”

Fig. 107: Min- / Max- and Default-Values for MC_WriteParameter

3.10.4 Signal-Time Diagram

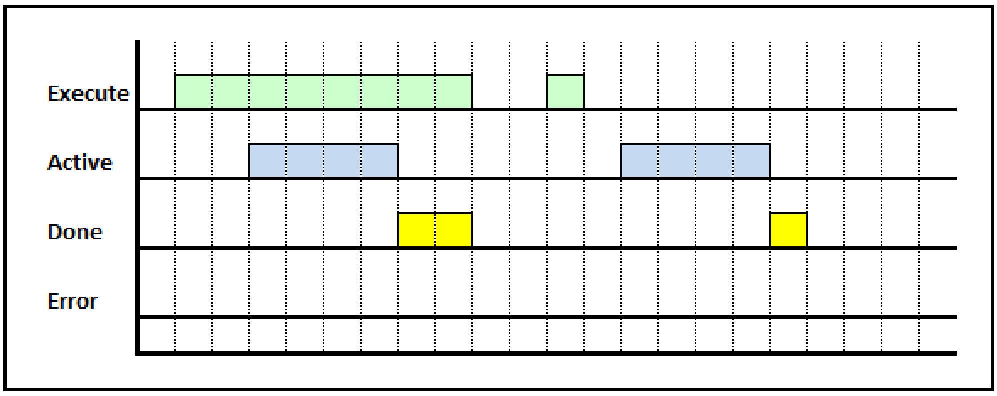


Fig. 108: Signal-Time Diagram MC_WriteParameter for – Processing terminated successfully

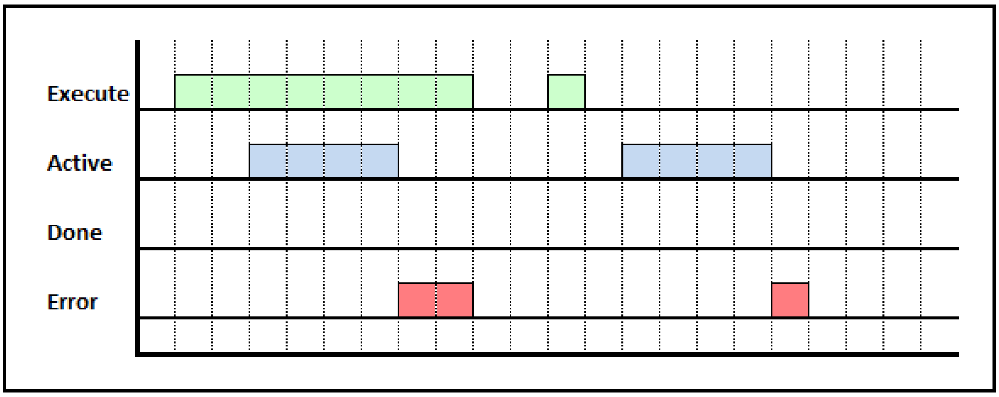


Fig. 109: Signal-Time Diagram for MC_WriteParameter – Processing terminated by error

3.10.5 Code example for MC_WriteParameter FB call in SCL

The code example below shows one way of calling an instance of MC_WriteParameter in SCL:

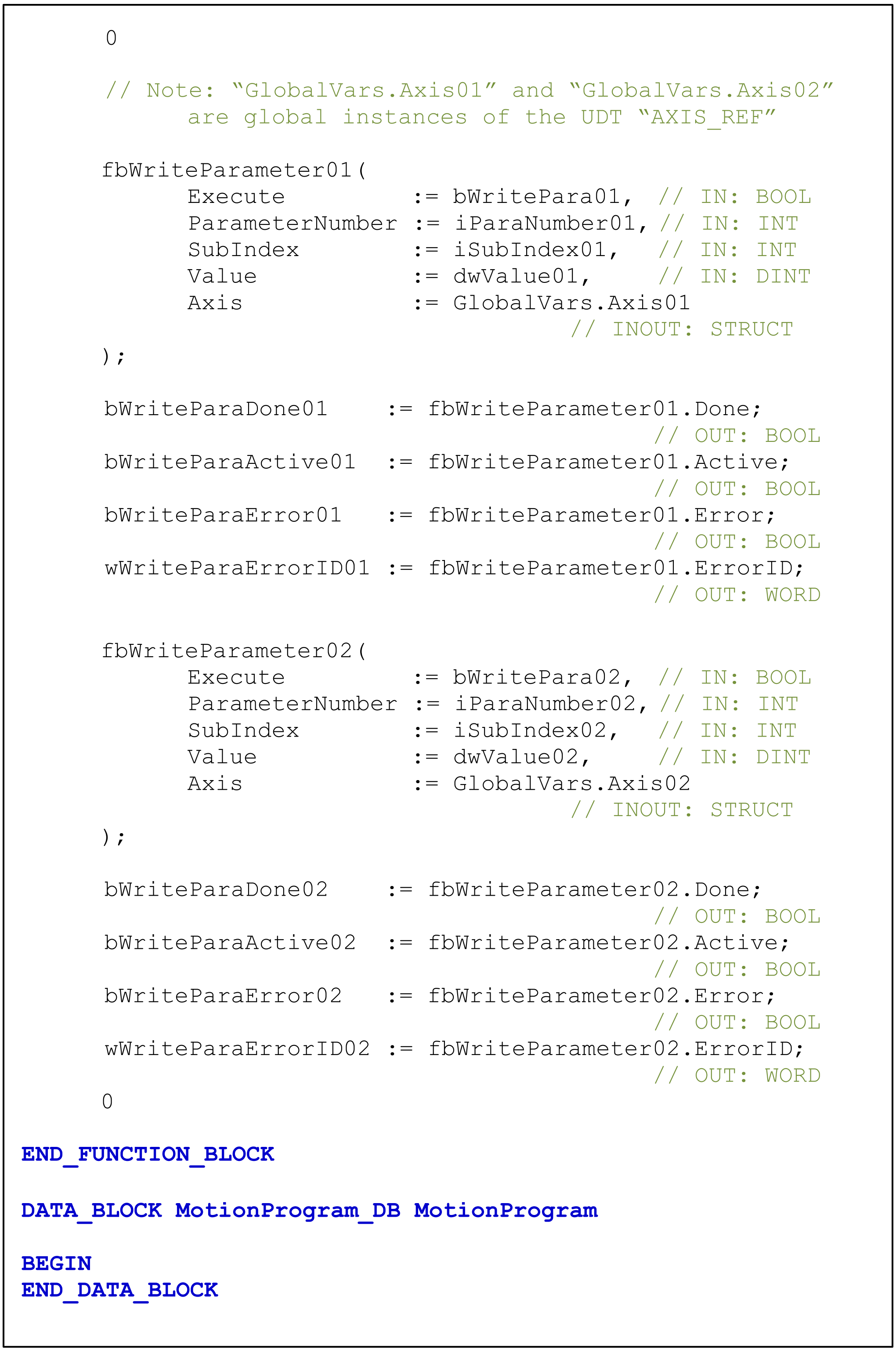
```
FUNCTION_BLOCK MotionProgram
      0
      VAR
        0
        (*in- and output variables for “MC_ReadParameter”*)
            bWritePara         : BOOL  := FALSE;
            iParaNumber        : INT   := 0;
            iSubIndex          : INT   := 0;
            dwValue            : DWORD := DW#16#0000_0000;
            bWriteParaDone     : BOOL  := FALSE;
            bWriteParaActive   : BOOL  := FALSE;
            bWriteParaError    : BOOL  := FALSE;
            wWriteParaErrorID  : WORD  := W#16#0000;
              0
      END_VAR
      0
BEGIN
      0
      (* PKW address configuration for 1st axis *)
      GlobalVars.Axis01.PkwAddressIn      := 256;
      GlobalVars.Axis01.PkwAddressOut  := 256;

      (* PZD address configuration for 1st axis *)
      GlobalVars.Axis01.PzdAddressIn      := 264;
      GlobalVars.Axis01.PzdAddressOut  := 264;

      // Note 1: “DBxxx” is an “Instance DB” for this
                 instance of “MC_WriteParameter”
      // Note 2: “GlobalVars.Axis01” is a global
                 instance of the UDT “AXIS_REF”

      MC_WriteParameter.DBxxx(
          Execute  := bWritePara,          // IN: BOOL
          ParameterNumber := iParaNumber,  // IN: INT
          SubIndex := iSubIndex,           // IN: INT
          Value  := dwValue,               // IN: DWORD
          Axis   := GlobalVars.Axis01      // INOUT: STRUCT
      );

      bWriteParaDone     := DBxxx.Done;        // OUT: BOOL
      bWriteParaActive   := DBxxx.Active;      // OUT: BOOL
      bWriteParaError    := DBxxx.Error;       // OUT: BOOL
      wWriteParaErrorID  := DBxxx.ErrorID;     // OUT: WORD

      0
END_FUNCTION_BLOCK

DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 110: Code example for MC_WriteParameter FB call in SCL

3.10.6 Code example for MC_WriteParameter FB call in SCL (Multi Instance)

The code example below shows one way of calling multiple instances of **MC_WriteParameter** in SCL:

```
FUNCTION_BLOCK MotionProgram
      0
      VAR
        0
        (*in- and output variables for “fbWriteParameter01”*)
        bWritePara01          : BOOL   := FALSE;
        iParaNumber01         : INT    := 0;
        iSubIndex01           : INT    := 0;
        dwValue01             : DWORD  := DW#16#0000_0000;
        bWriteParaDone01      : BOOL   := FALSE;
        bWriteParaActive01    : BOOL   := FALSE;
        bWriteParaError01     : BOOL   := FALSE;
        wWriteParaErrorID01   : WORD   := W#16#0000;

        (*in- and output variables for “fbWriteParameter02”*)
        bWritePara02          : BOOL   := FALSE;
        iParaNumber02         : INT    := 0;
        iSubIndex02           : INT    := 0;
        dwValue02             : DWORD  := DW#16#0000_0000;
        bWriteParaDone02      : BOOL   := FALSE;
        bWriteParaActive02    : BOOL   := FALSE;
        bWriteParaError02     : BOOL   := FALSE;
        wWriteParaErrorID02   : WORD   := W#16#0000;

        (* instances of “MC_WriteParameter” *)
        fbWriteParameter01 : MC_WriteParameter;
        fbWriteParameter02 : MC_WriteParameter;

        0

      END_VAR
      0
BEGIN
      0
      (* PKW address configuration for 1st axis *)
      GlobalVars.Axis01.PkwAddressIn   := 256;
      GlobalVars.Axis01.PkwAddressOut  := 256;

      (* PZD address configuration for 1st axis *)
      GlobalVars.Axis01.PzdAddressIn   := 264;
      GlobalVars.Axis01.PzdAddressOut  := 264;

      (* PKW address configuration for 2nd axis *)
      GlobalVars.Axis02.PkwAddressIn   := 272;
      GlobalVars.Axis02.PkwAddressOut  := 272;

      (* PZD address configuration for 2nd axis *)
      GlobalVars.Axis02.PzdAddressIn   := 280;
      GlobalVars.Axis02.PzdAddressOut  := 280;
      0
```



```
       0

       // Note: “GlobalVars.Axis01” and “GlobalVars.Axis02”
             are global instances of the UDT “AXIS_REF”

       fbWriteParameter01(
             Execute         := bWritePara01,  // IN: BOOL
             ParameterNumber := iParaNumber01, // IN: INT
             SubIndex        := iSubIndex01,   // IN: INT
             Value           := dwValue01,     // IN: DINT
             Axis            := GlobalVars.Axis01
                                            // INOUT: STRUCT
       );

       bWriteParaDone01    := fbWriteParameter01.Done;
                                                  // OUT: BOOL
       bWriteParaActive01  := fbWriteParameter01.Active;
                                                  // OUT: BOOL
       bWriteParaError01   := fbWriteParameter01.Error;
                                                  // OUT: BOOL
       wWriteParaErrorID01 := fbWriteParameter01.ErrorID;
                                                  // OUT: WORD

       fbWriteParameter02(
             Execute         := bWritePara02,  // IN: BOOL
             ParameterNumber := iParaNumber02, // IN: INT
             SubIndex        := iSubIndex02,   // IN: INT
             Value           := dwValue02,     // IN: DINT
             Axis            := GlobalVars.Axis02
                                            // INOUT: STRUCT
       );

       bWriteParaDone02    := fbWriteParameter02.Done;
                                                  // OUT: BOOL
       bWriteParaActive02  := fbWriteParameter02.Active;
                                                  // OUT: BOOL
       bWriteParaError02   := fbWriteParameter02.Error;
                                                  // OUT: BOOL
       wWriteParaErrorID02 := fbWriteParameter02.ErrorID;
                                                  // OUT: WORD
       0

END_FUNCTION_BLOCK

DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 111: Code example for MC_WriteParameter multiple instance FB call in SCL

3.10.7 Code example for MC_ReadParameter FB call in FBD

Name	Data Type	Initial Value
bWritePara	Bool	False
iParaNumber	Int	0
iSubIndex	Int	0
dwValue	DWord	DW#16#0
bWriteParaDone	Bool	False
bWriteParaActive	Bool	False
bWriteParaError	Bool	False
bWriteParaErrorID	Word	W#16#0

Fig. 112: Variable declaration for MC_WriteParameter FB call

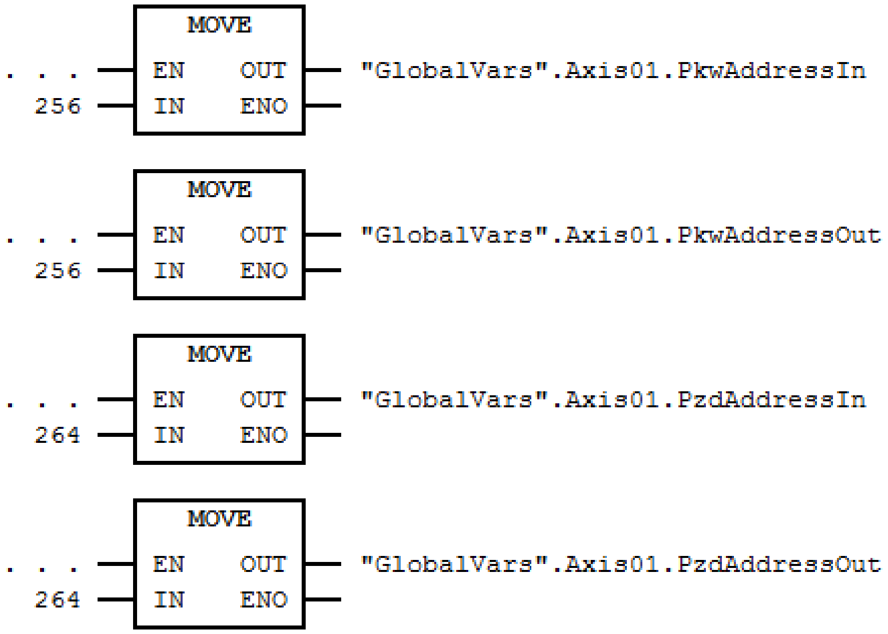

MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressIn
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressOut
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressIn
264
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressOut
264
IN
ENO

Fig. 113: Address configuration for 1st axis in FBD

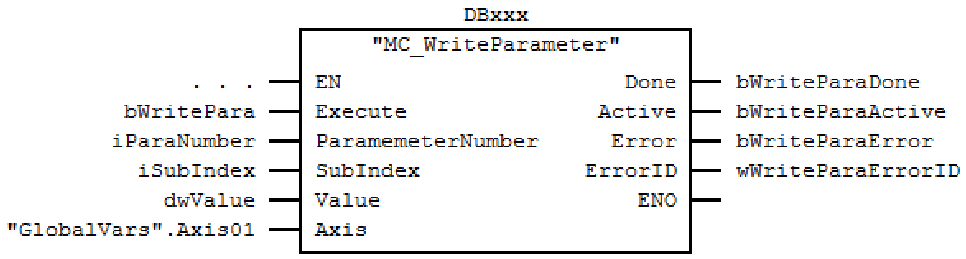


Fig. 114: MC_WriteParameter FB call in FBD

Note 1:
"DBxxx" is an "Instance DB" for this instance of "MC_WriteParameter"
"GlobalVars.Axis01" is a global instance of the UDT "AXIS_REF"

3.10.8 Code example for MC_ReadParameter FB call in FBD (Multi Instance)

Name	Data Type	Initial Value
bWritePara01	Bool	False
iParaNumber01	Int	0
iSubIndex01	Int	0
dwValue01	DWord	DW#16#0
bWriteParaDone01	Bool	False
bWriteParaActive01	Bool	False
bWriteParaError01	Bool	False
bWriteParaErrorID01	Word	W#16#0
bWritePara02	Bool	False
iParaNumber02	Int	0
iSubIndex02	Int	0
dwValue02	DWord	DW#16#0
bWriteParaDone02	Bool	False
bWriteParaActive02	Bool	False
bWriteParaError02	Bool	False
bWriteParaErrorID02	Word	W#16#0
fbWriteParameter01	MC_WriteParameter	
fbWriteParameter01	MC_WriteParameter	

Fig. 115: Variable declaration for multiple instance calls of MC_WriteParameter FB

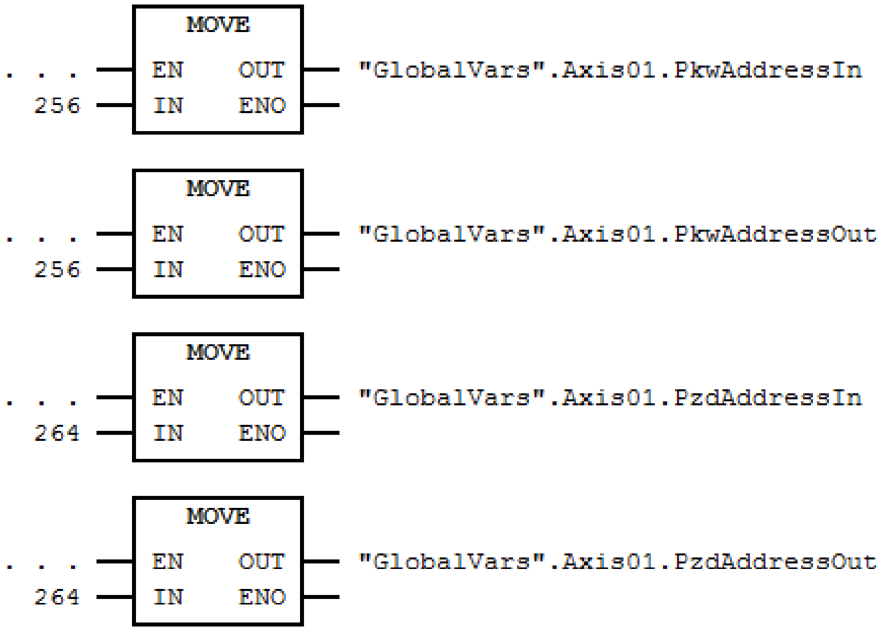

MOVE
...
EN
OUT
"GlobalVars".Axis01.PkwAddressIn
256
IN
ENO
MOVE
...
EN
OUT
"GlobalVars".Axis01.PkwAddressOut
256
IN
ENO
MOVE
...
EN
OUT
"GlobalVars".Axis01.PzdAddressIn
264
IN
ENO
MOVE
...
EN
OUT
"GlobalVars".Axis01.PzdAddressOut
264
IN
ENO

Fig. 116: Address configuration for 1st axis in FBD

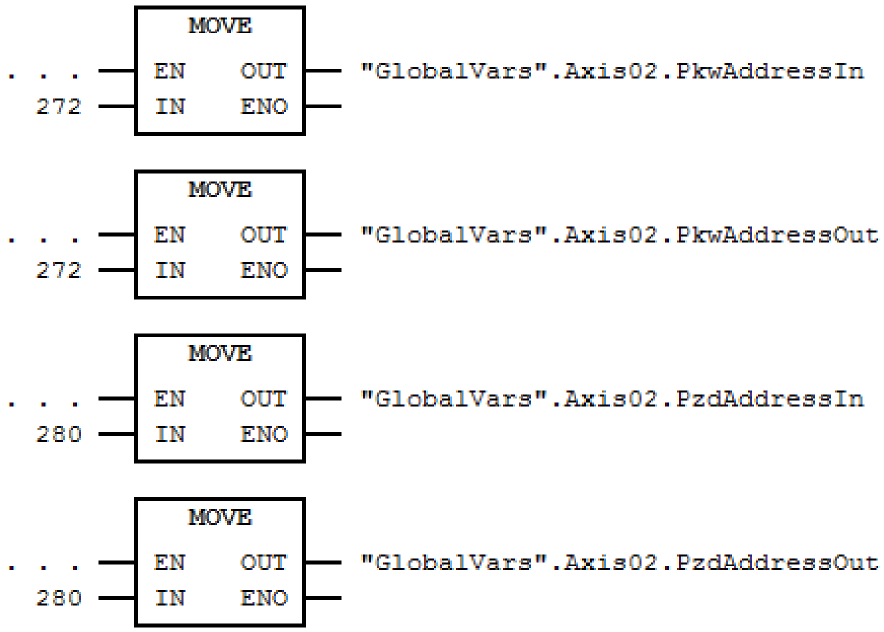

MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PkwAddressIn
272
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PkwAddressOut
272
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PzdAddressIn
280
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis02.PzdAddressOut
280
IN
ENO

Fig. 117: Address configuration for 2nd axis in FBD

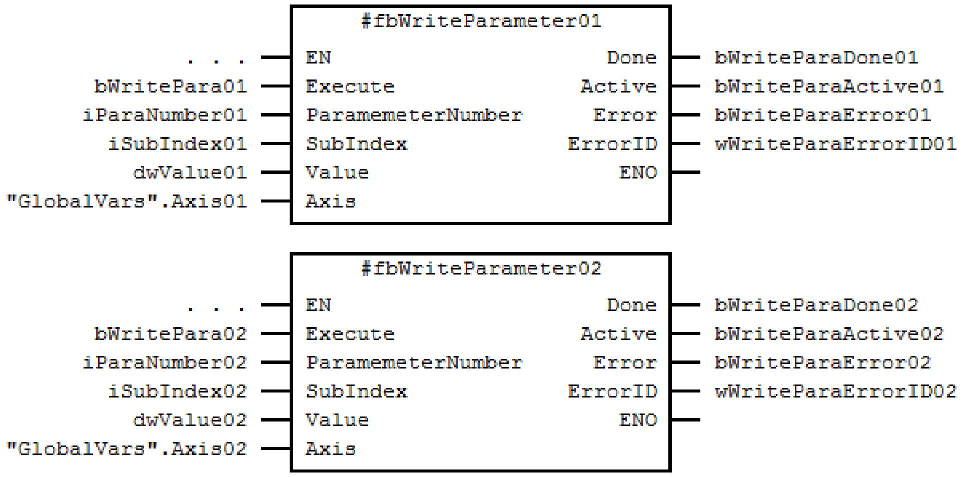


Fig. 118: FB calls of MC_WriteParameter (as Multiple Instances) in FBD

Note:
“GlobalVars.Axis01” and “GlobalVars.Axis02” are global instances of the UDT “AXIS_REF”

3.10.9 Error Handling

ErrorID (hex)	ErrorID (bin)	Description
16#8101	2#1000_0001_0000_0001	Error main state machine
16#8102	2#1000_0001_0000_0010	Error “in operation” state machine
16#8201	2#1000_0010_0000_0001	Invalid PKW input address
16#8202	2#1000_0010_0000_0010	Invalid PKW output address
16#8203	2#1000_0010_0000_0011	Invalid Parameter Number
16#8204	2#1000_0010_0000_0100	Invalid Subindex
16#8205	2#1000_0010_0000_0101	Subindex is greater than parameter array size
16#8301	2#1000_0011_0000_0001	Error while receiving PKW answer from drive
16#8302	2#1000_0011_0000_0010	Error while sending PKW request to drive
16#8A01	2#1000_1010_0000_0001	Communication timeout
16#8B01	2#1000_1011_0000_0001	PKW channel currently used by another Function Block
16#8C01	2#1000_1100_0000_0001	Internal error (parameter type error)
Forthefollowingerrors: **X** = 4 (**YYYY** = 0100) : Error while Parameter Read Request **X** = 5 (**YYYY** = 0101) : Error while Parameter Array Read Request **X** = 6 (**YYYY** = 0110) : Error while Parameter (Array) Write Request **X** = 7 (**YYYY** = 0111) : Error while Parameter Array Check		
16#8X00	2#1000_**YYYY**_0000_0000	Impermissible parameter number
16#8X01	2#1000_**YYYY**_0000_0001	Parameter value cannot be changed
16#8X02	2#1000_**YYYY**_0000_0010	Lower or upper value limit violated
16#8X03	2#1000_**YYYY**_0000_0011	Faulty sub-index
16#8X04	2#1000_**YYYY**_0000_0100	Not an array
16#8X05	2#1000_**YYYY**_0000_0101	Wrong data type
16#8X06	2#1000_**YYYY**_0000_0110	Setting not permitted (can only be reset)
16#8X07	2#1000_**YYYY**_0000_0111	Descriptive element cannot be changed
16#8X08	2#1000_**YYYY**_0000_1000	PPO write requested in the IR not present
16#8X09	2#1000_**YYYY**_0000_1001	Descriptive data not present
16#8X0A	2#1000_**YYYY**_0000_1010	Access group wrong
16#8X0B	2#1000_**YYYY**_0000_1011	No operating authority
16#8X0C	2#1000_**YYYY**_0000_1100	Wrong password
16#8X0D	2#1000_**YYYY**_0000_1101	Illegible text in cyclic traffic
16#8X0E	2#1000_**YYYY**_0000_1110	Illegible name in cyclic traffic
16#8X0F	2#1000_**YYYY**_0000_1111	No text array present
16#8X10	2#1000_**YYYY**_0001_0000	Missing PPO write
16#8X11	2#1000_**YYYY**_0001_0001	Job cannot be executed because of operating status
16#8X12	2#1000_**YYYY**_0001_0010	Other error
16#8X13	2#1000_**YYYY**_0001_0011	Illegible date in cyclic traffic

Fig. 119: Error Codes of MC_WriteParameter

3.11 MC_FaultCheck

3.11.1 Brief Description

The function block **MC_FaultCheck** reads the values of the “Fault” bit and the “Warning” bit from status word.

It is recommended that MC_FaultCheck is running all the time, because the error-state of the FBs only shows FB internal fault

3.11.2 Interface

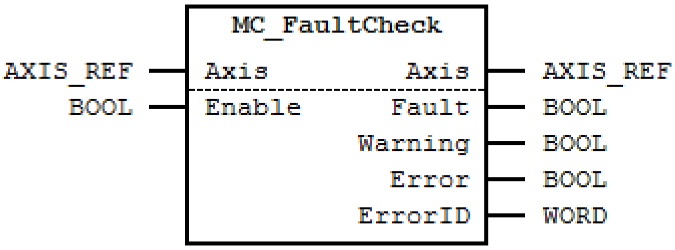


Fig. 120: MC_FaultCheck Interface Diagram

I/O Type	Name	Data Type	Descriptio n
VAR_IN_OUT	Axis	AXIS_REF	Data structure which contains several information for data exchange with other function blocks and communication settings for the drive. For further information on this structure please see section 3.12
VAR_INPUT	Enable	BOOL	Values will be read as long as “Enable” is “TRUE”
VAR_OUTPUT	Fault	BOOL	Value of the “Fault” bit from status word of the drive
	Warning	BOOL	Value of the “Warning” bit from status word of the drive
	Error	BOOL	Indicates that an error has occurred while processing the FB
	ErrorID	WORD	Error identification

Fig. 121: MC_FaultCheck I/O Interface Description

3.11.3 Min- / Max- and Default-Values of inputs

Name	Type	Min-Value	Max-Value	Default-Value	Takeover
Enable	BOOL			FALSE	Continuous

Fig. 122: Min- / Max- and Default-Values for MC_FaultCheck

3.11.4 Signal-Time Diagram

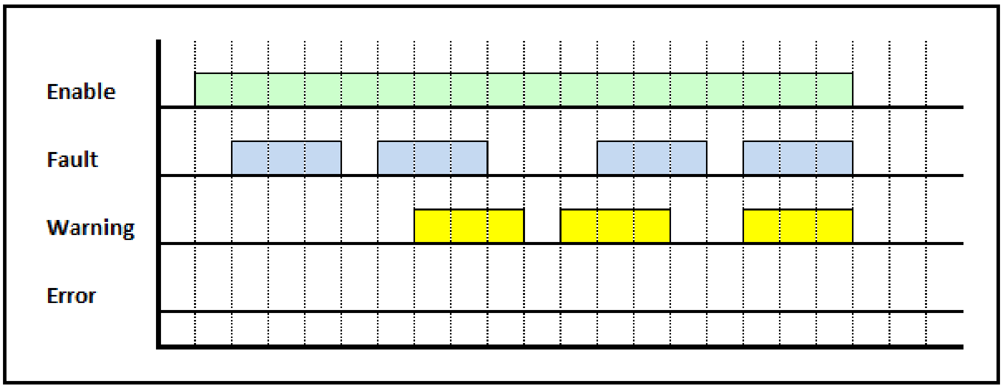


Fig. 123: Signal-Time Diagram for MC_FaultCheck – Processing terminated successfully

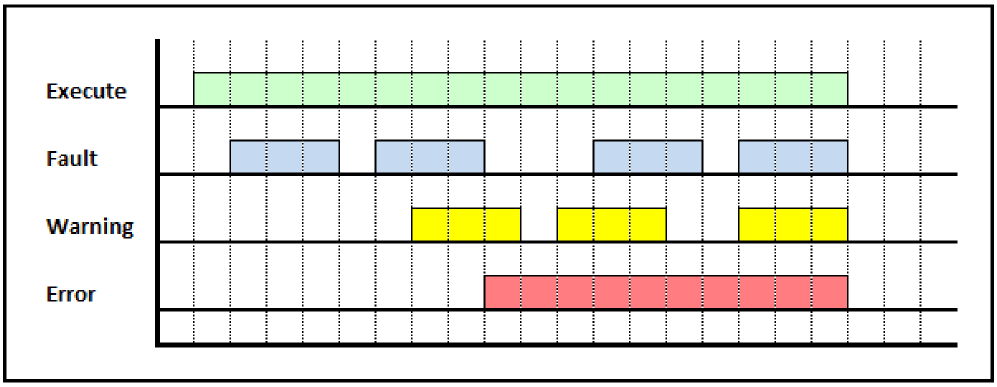


Fig. 124: Signal-Time Diagram for MC_FaultCheck – Processing terminated by error

3.11.5 Code example for MC_FaultCheck FB call in SCL

The code example below shows one way of calling an instance of **MC_FaultCheck** in SCL:

```
FUNCTION_BLOCK MotionProgram

      0

      VAR
        0
        (* in- and output variables for “MC_FaultCheck” *)
        bEnableFaultCheck  : BOOL := FALSE;
        bFault             : BOOL := FALSE;
        bWarning           : BOOL := FALSE;
        bFaultCheckError   : BOOL := FALSE;
        wFaultCheckErrorID : WORD := W#16#0000;
        0
      END_VAR

      0

BEGIN

      0
      (* PKW address configuration for 1st axis *)
      GlobalVars.Axis01.PkwAddressIn  := 256;
      GlobalVars.Axis01.PkwAddressOut := 256;

      (* PZD address configuration for 1st axis *)
      GlobalVars.Axis01.PzdAddressIn  := 264;
      GlobalVars.Axis01.PzdAddressOut := 264;

      // Note 1: “DBxxx” is an “Instance DB” for this
                  instance of “MC_FaultCheck”
      // Note 2: “GlobalVars.Axis01” is a global instance
                  of the UDT “AXIS_REF”

      MC_FaultCheck.DBxxx(
            Enable  := bEnableFaultCheck,     // IN: BOOL
            Axis    := GlobalVars.Axis01      // INOUT: STRUCT
      );

      bFault             := DBxxx.Fault;      // OUT: BOOL
      bWarning           := DBxxx.Warning;    // OUT: BOOL
      bFaultCheckError   := DBxxx.Error;      // OUT: BOOL
      wFaultCheckErrorID := DBxxx.ErrorID;    // OUT: WORD

      0

END_FUNCTION_BLOCK


DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 125: Code example for MC_FaultCheck FB call in SCL

3.11.6 Code example for MC_FaultCheck FB call in SCL (Multi Instance)

The code example below shows one way of calling multiple instances of **MC_FaultCheck** in SCL:

```
FUNCTION_BLOCK MotionProgram
      0
      VAR
      0

        (* in- and output variables for 1st instance of
                  “fbPower01” *)
        bEnableFaultCheck01  : BOOL := FALSE;
        bFault01             : BOOL := FALSE;
        bWarning01           : BOOL := FALSE;
        bFaultCheckError01   : BOOL := FALSE;
        wFaultCheckErrorID01 : WORD := W#16#0000;

        (* in- and output variables for 2nd instance of
                  “fbPower02” *)
        bEnableFaultCheck02  : BOOL := FALSE;
        bFault02             : BOOL := FALSE;
        bWarning02           : BOOL := FALSE;
        bFaultCheckError02   : BOOL := FALSE;
        wFaultCheckErrorID02 : WORD := W#16#0000;

        (* instances of “MC_FaultCheck” *)
        fbFaultCheck01    : MC_FaultCheck;
        fbFaultCheck02    : MC_FaultCheck;
        0
      END_VAR
      0
BEGIN
      0

      (* PKW address configuration for 1st axis *)
      GlobalVars.Axis01.PkwAddressIn  := 256;
      GlobalVars.Axis01.PkwAddressOut := 256;

      (* PZD address configuration for 1st axis *)
      GlobalVars.Axis01.PzdAddressIn  := 264;
      GlobalVars.Axis01.PzdAddressOut := 264;


      (* PKW address configuration for 2nd axis *)
      GlobalVars.Axis02.PkwAddressIn  := 272;
      GlobalVars.Axis02.PkwAddressOut := 272;

      (* PZD address configuration for 2nd axis *)
      GlobalVars.Axis02.PzdAddressIn  := 280;
      GlobalVars.Axis02.PzdAddressOut := 280;

      0
      0

      // Note: “GlobalVars.Axis01” and “GlobalVars.Axis02”
                     are global
      //       instances of the UDT “AXIS_REF”

      fbFaultCheck01(
            Enable    := bEnableFaultCheck01, // IN: BOOL
            Axis      := GlobalVars.Axis01    // INOUT: STRUCT
      );

      bFault01             := fbFaultCheck01.Fault;
                  // OUT: BOOL
      bWarning01           := fbFaultCheck01.Warning;
                  // OUT: BOOL
      bFaultCheckError01   := fbFaultCheck01.Error;
                  // OUT: BOOL
      wFaultCheckErrorID01 := fbFaultCheck01.ErrorID;

                  // OUT: WORD

      fbFaultCheck02(
            Enable    := bEnableFaultCheck02, // IN: BOOL
            Axis      := GlobalVars.Axis02    // INOUT: STRUCT
      );

      bFault02             := fbFaultCheck02.Fault;
                  // OUT: BOOL
      bWarning02           := fbFaultCheck02.Warning;
                  // OUT: BOOL
      bFaultCheckError02   := fbFaultCheck02.Error;
                  // OUT: BOOL
      wFaultCheckErrorID02 := fbFaultCheck02.ErrorID;

                  // OUT: WORD
      0
END_FUNCTION_BLOCK


DATA_BLOCK MotionProgram_DB MotionProgram

BEGIN
END_DATA_BLOCK
```

Fig. 126: Code example for MC_FaultCheck multiple instance FB call in SCL

3.11.7 Code example for MC_FaultCheck FB call in FBD

Name	Data Type	Initial Value
bEnableFaultCheck	Bool	False
bFault	Bool	False
bWarning	Bool	False
bFaultCheckError	Bool	False
wFaultCheckErrorID	Word	W#16#0

Fig. 127: Variable declaration for MC_FaultCheck FB call

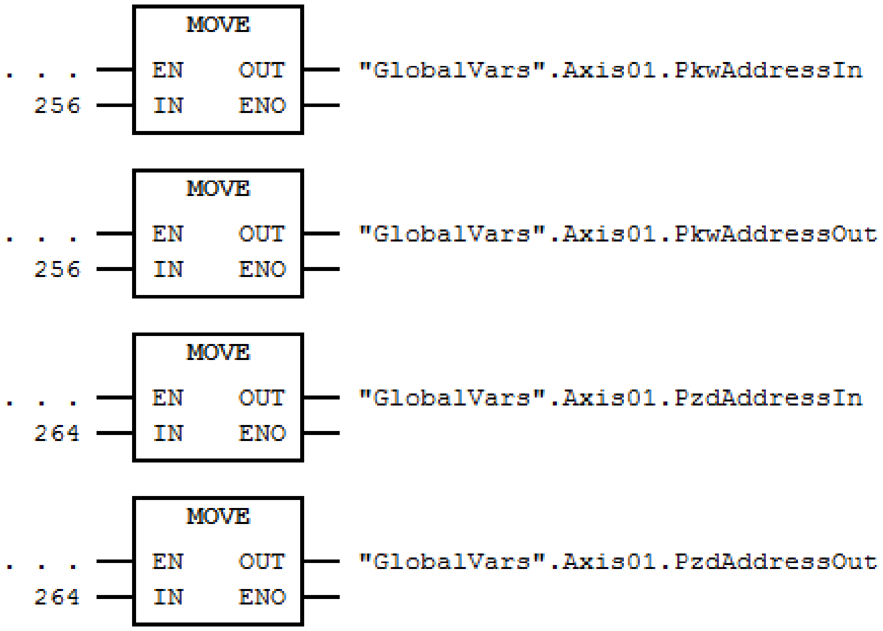

MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressIn
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PkwAddressOut
256
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressIn
264
IN
ENO
MOVE
. . .
EN
OUT
"GlobalVars".Axis01.PzdAddressOut
264
IN
ENO

Fig. 128: Address configuration for 1st axis in FBD

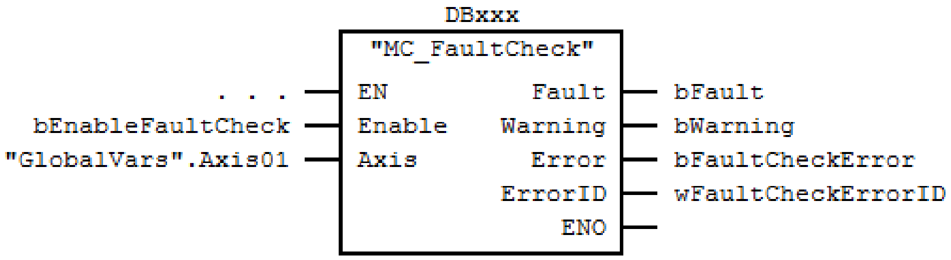


Fig. 129: MC_FaultCheck FB call in FBD

Note:
“DBxxx” is an “Instance DB” for this instance of “MC_FaultCheck”
“GlobalVars.Axis01” is a global instance of the UDT “AXIS_REF”

3.11.8 Code example for MC_FaultCheck FB call in FBD (Multi Instance)

Name	Data Type	Initial Value
bEnableFaultCheck01	Bool	False
bFault01	Bool	False
bWarning01	Bool	False
bFaultCheckError01	Bool	False
wFaultCheckErrorID01	Word	W#16#0
bEnableFaultCheck02	Bool	False
bFault02	Bool	False
bWarning02	Bool	False
bFaultCheckError02	Bool	False

wFaultCheckErrorID02	Word	W#16#0
fbFaultCheck01	My_FaultCheck	
fbFaultCheck02	My_FaultCheck	

Fig. 130: Variable declaration for multiple instance calls of MC_FaultCheck FB

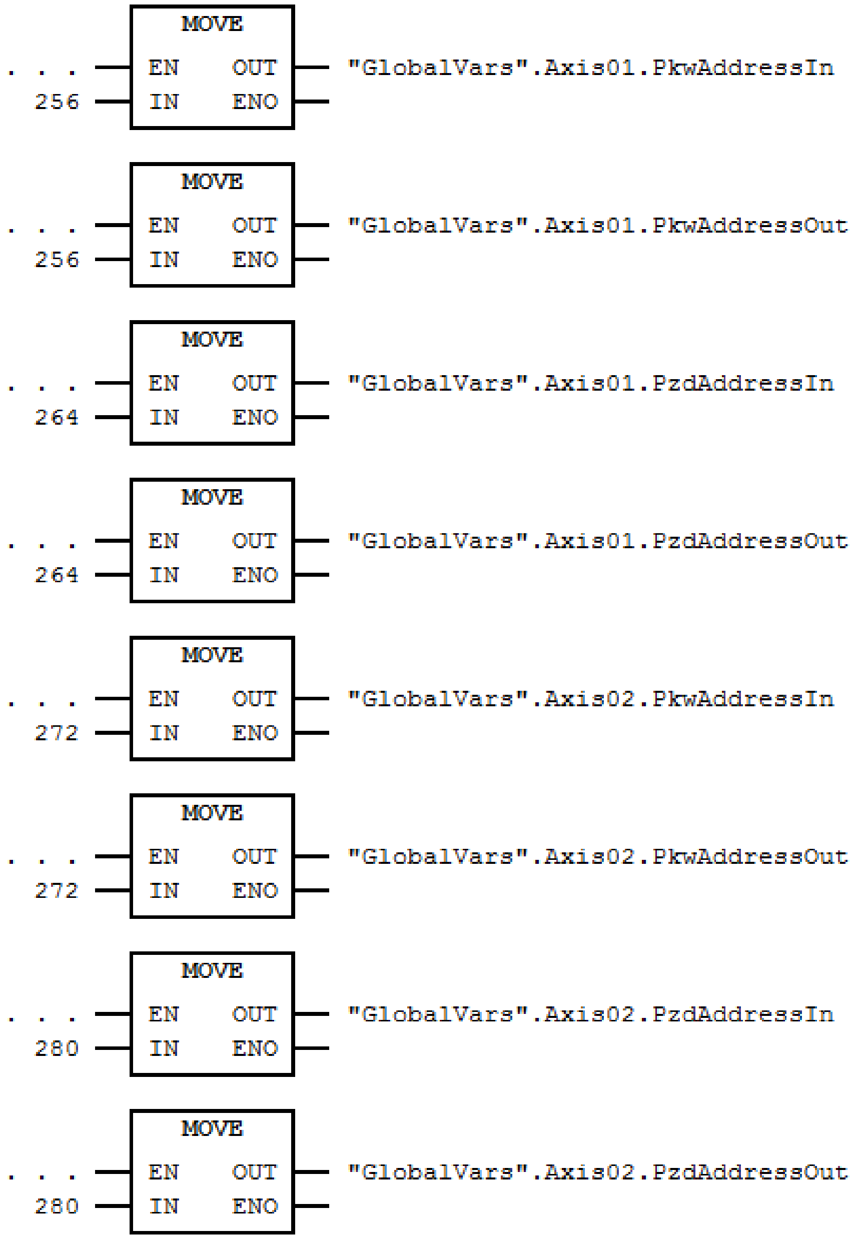

MOVE
EN OUT
IN ENO
. . .
256
"GlobalVars".Axis01.PkwAddressIn
256
"GlobalVars".Axis01.PkwAddressOut
264
"GlobalVars".Axis01.PzdAddressIn
264
"GlobalVars".Axis01.PzdAddressOut
272
"GlobalVars".Axis02.PkwAddressIn
272
"GlobalVars".Axis02.PkwAddressOut
280
"GlobalVars".Axis02.PzdAddressIn
280
"GlobalVars".Axis02.PzdAddressOut

Fig. 131: Address configuration for 1st and 2nd axis in FBD

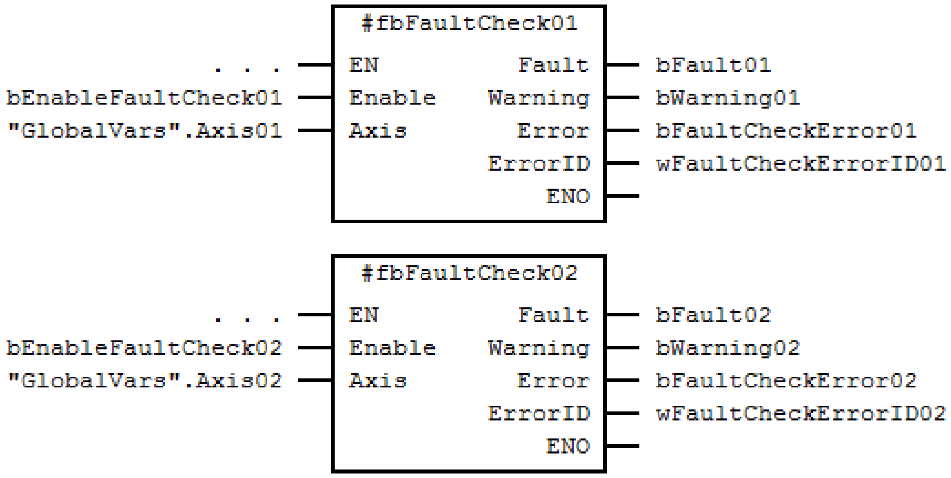


Fig. 132: FB calls of MC_FaultCheck (as Multiple Instances) in FBD

Note: “GlobalVars.Axis01” and “GlobalVars.Axis02” are global instances of the UDT
“AXIS_REF”

3.11.9 Error Handling

ErrorID (hex)	ErrorID (bin)	Description
16#9101	2#1001_0001_0000_0001	Error main state machine
16#9201	2#1001_0010_0000_0001	Invalid PZD input address
16#9202	2#0001_0010_0000_0010	Invalid PZD output address
16#9301	2#0001_0011_0000_0001	Error while reading ZSW

Fig. 133: Error Codes of MC_FaultCheck

3.12 Data Structure AXIS_REF

The Data Structure **AXIS_REF** is used for communication settings of the axis and for internal data transfer between the Function Blocks. You will need **one global instance** of this structure **for every drive / axis** in your project.

Parameter Name	Data Type	Set by	Description
PkwAddressIn	INT	User	Parameter for communication with the axis
PkwAddressOut	INT	User	Parameter for communication with the axis
PzdAddressIn	INT	User	Parameter for communication with the axis
PzdAddressOut	INT	User	Parameter for communication with the axis
AxisNo	INT	User	Number of the Axis
AxisName	STRING [16]	User	Name of the Axis
AxisDescription	STRING [32]	User	Description of the Axis (e.g. function / task)
AxisState	INT	Function Blocks	Actual Axis state
Private	STRUCT	Function Blocks	Data structure for internal use only

Fig. 134: Data Structure AXIS_REF

4 Examples

4.1 How to configure the communication with a drive

- Select station and then double-click on “Hardware” to open up “Hardware Config”

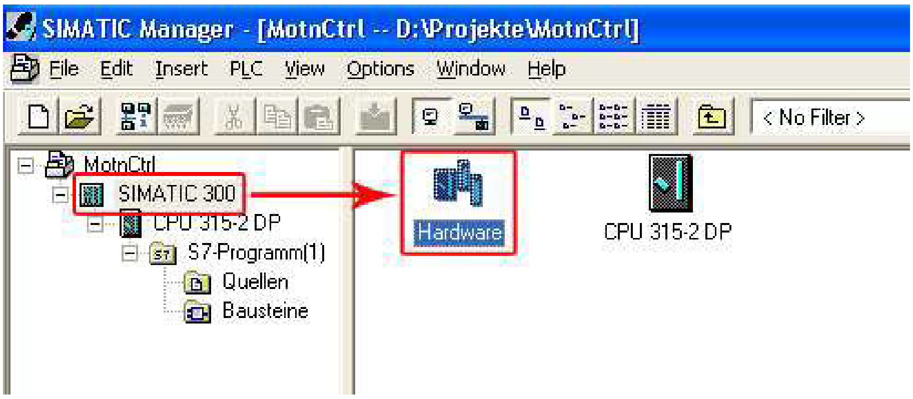
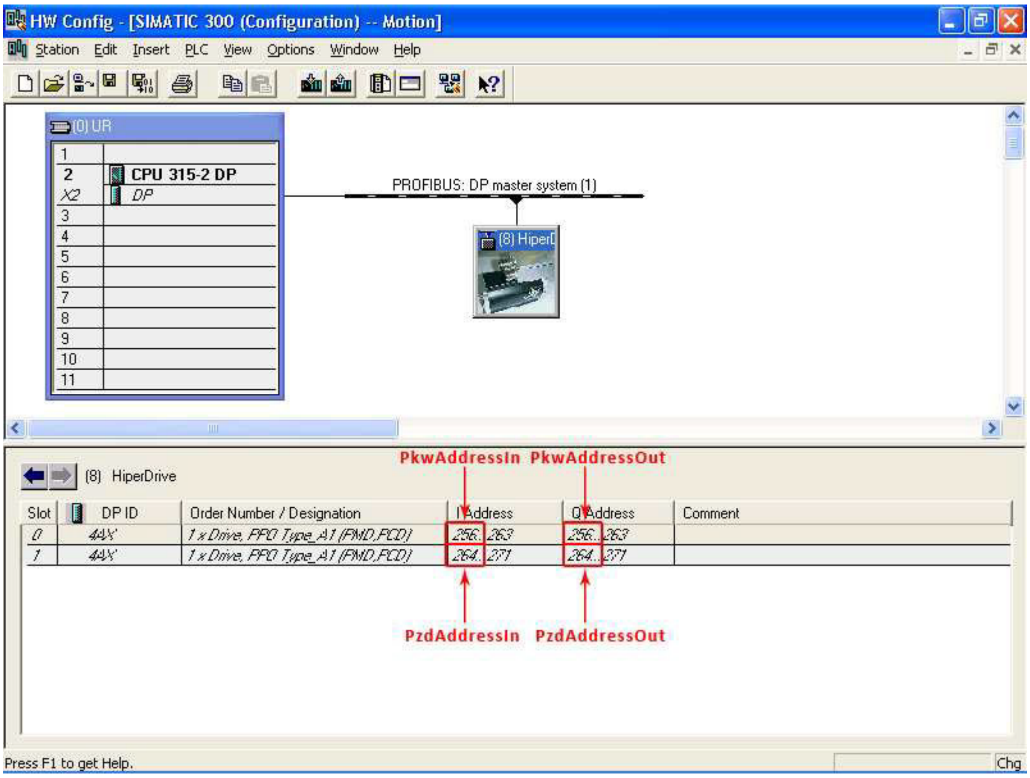

SIMATIC Manager - [MotnCtrl -- D:\Projekte\MotnCtrl]
File Edit Insert PLC View Options Window Help
< No Filter >
MotnCtrl
SIMATIC 300
CPU 315-2 DP
S7-Programm(1)
Quellen
Bausteine
Hardware
CPU 315-2 DP

Fig. 135: Start HW Config from SIMATIC Manager

- Select the desired drive in “HW Config”, to see the address configuration



Slot	DP ID	Order Number / Designation	I Address	Q Address	Comment
0	44X'	1 x Drive, PPO Type_A1 (PMD,PCD)	256..263	256..263	
1	44X'	1 x Drive, PPO Type_A1 (PMD,PCD)	264..271	264..271	

Fig. 136: Drive addresses in HW Config

Note: The number of configured drives must accord to the number of drives on the HUB.
If you have 8 drives on the HUB you have a reaction time of the drive of ca. 0,5 sec

- Create new DB by right-click into window → Insert New Object → Data Block

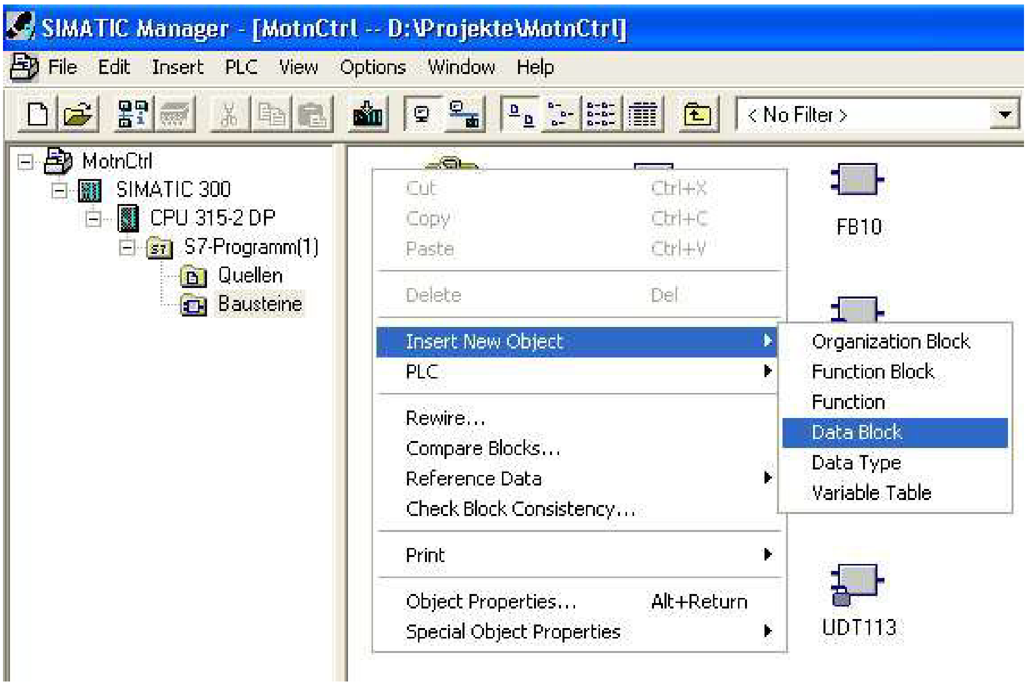


Fig. 137: Create Data Block with Right-Click

- or use the menu Insert → S7 Block → Data Block

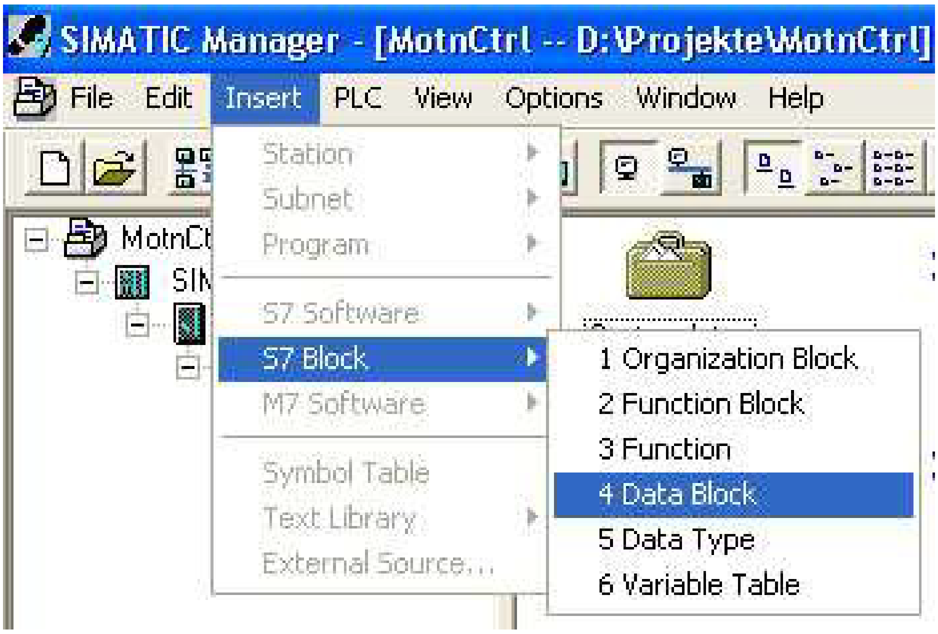
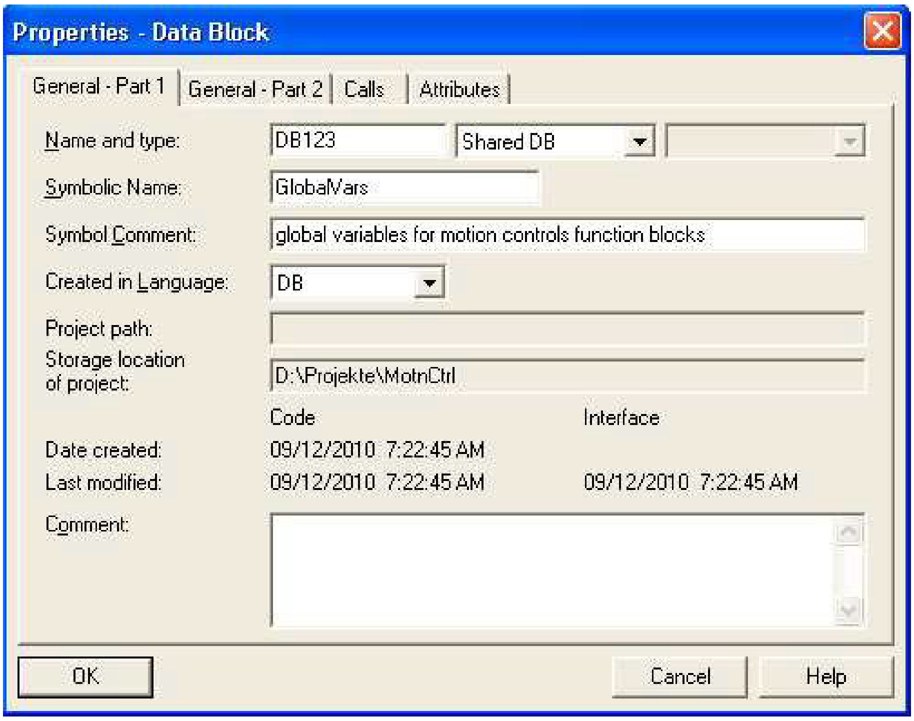


Fig. 138: Create Data Block over the menu

- Fill out all the needed properties for the new Data Block, then click “OK”



Name and type: DB123 Shared DB

Symbolic Name: GlobalVars

Symbol Comment: global variables for motion controls function blocks

Created in Language: DB

Project path:

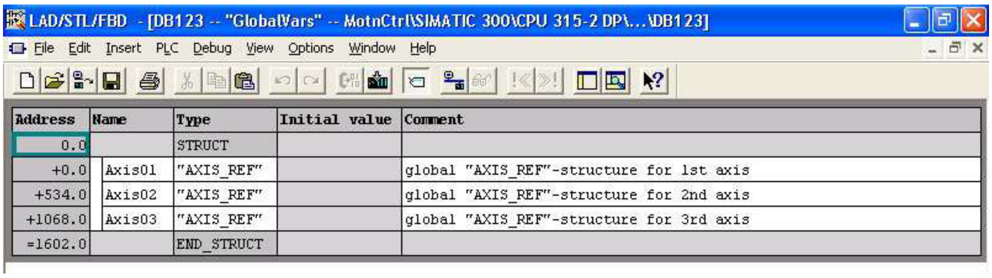
Storage location of project: D:\Projekte\MotnCtrl

	Code	Interface
Date created:	09/12/2010 7:22:45 AM	
Last modified:	09/12/2010 7:22:45 AM	09/12/2010 7:22:45 AM

Comment:

Fig. 139: Data Block properties

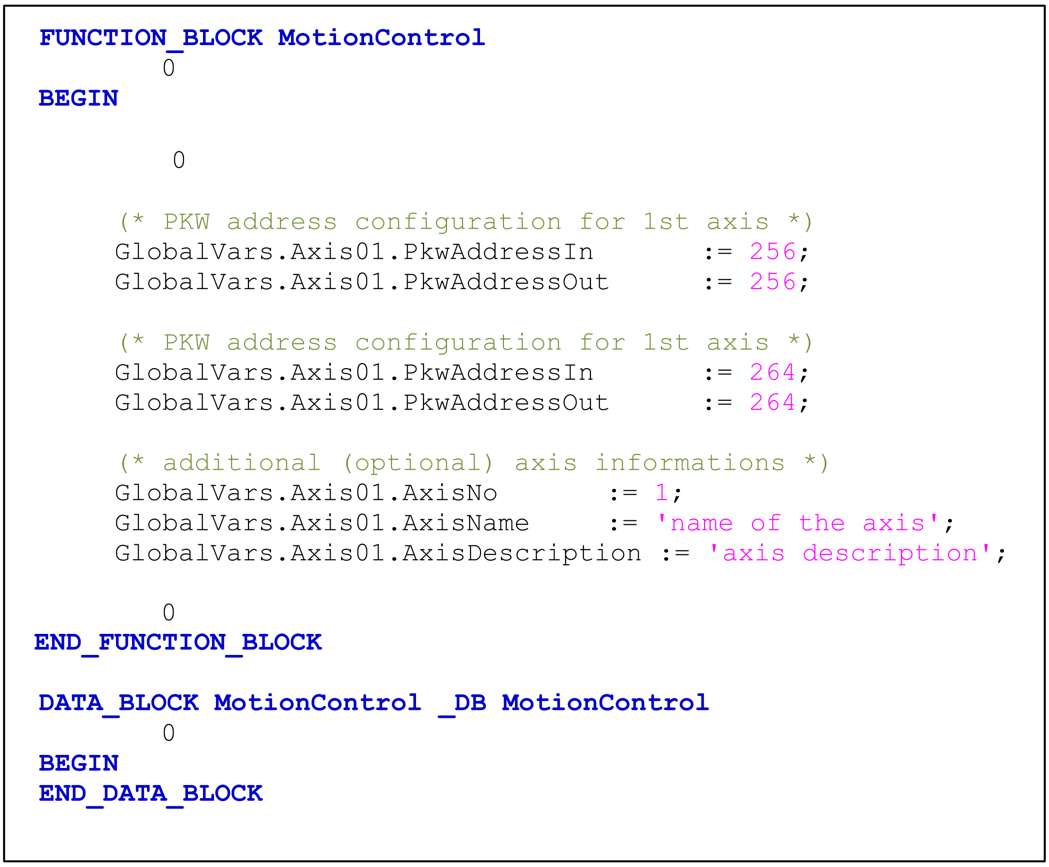
- Open the new data block by double-clicking on it and insert one instance of “AXIS_REF” for every drive that you want to control with the Function Blocks


LAD/STL/FBD - [DB123 -- "GlobalVars" -- MotnCtrl\SIMATIC 300\CPU 315-2 DP\...\DB123]
File Edit Insert PLC Debug View Options Window Help

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	Axis01	"AXIS_REF"		global "AXIS_REF"-structure for 1st axis
+534.0	Axis02	"AXIS_REF"		global "AXIS_REF"-structure for 2nd axis
+1068.0	Axis03	"AXIS_REF"		global "AXIS_REF"-structure for 3rd axis
=1602.0		END_STRUCT		

Fig. 140: Data Block content

- Open your motion program or create a new motion program and set the address values from HW Config to the related instance of “AXIS_REF”, you can also add additional information for maintenance purposes



```
FUNCTION_BLOCK MotionControl
        0
BEGIN

         0

     (* PKW address configuration for 1st axis *)
     GlobalVars.Axis01.PkwAddressIn       := 256;
     GlobalVars.Axis01.PkwAddressOut      := 256;

     (* PKW address configuration for 1st axis *)
     GlobalVars.Axis01.PkwAddressIn       := 264;
     GlobalVars.Axis01.PkwAddressOut      := 264;

     (* additional (optional) axis informations *)
     GlobalVars.Axis01.AxisNo       := 1;
     GlobalVars.Axis01.AxisName     := 'name of the axis';
     GlobalVars.Axis01.AxisDescription := 'axis description';

        0
END_FUNCTION_BLOCK

DATA_BLOCK MotionControl _DB MotionControl
        0
BEGIN
END_DATA_BLOCK
```

Fig. 141: Setting up drive addresses and addition information in SCL

Notes:

- Use same global instance of “AXIS_REF” for all function blocks that should control the same Axis.
- Never use more than one instance of “AXIS_REF” for the same axis.

4.2 How to switch on drive power

- First set up the communication with the drive (section 4.1)
- Add an instance of MC_Power to motion program and set up in- and output variables

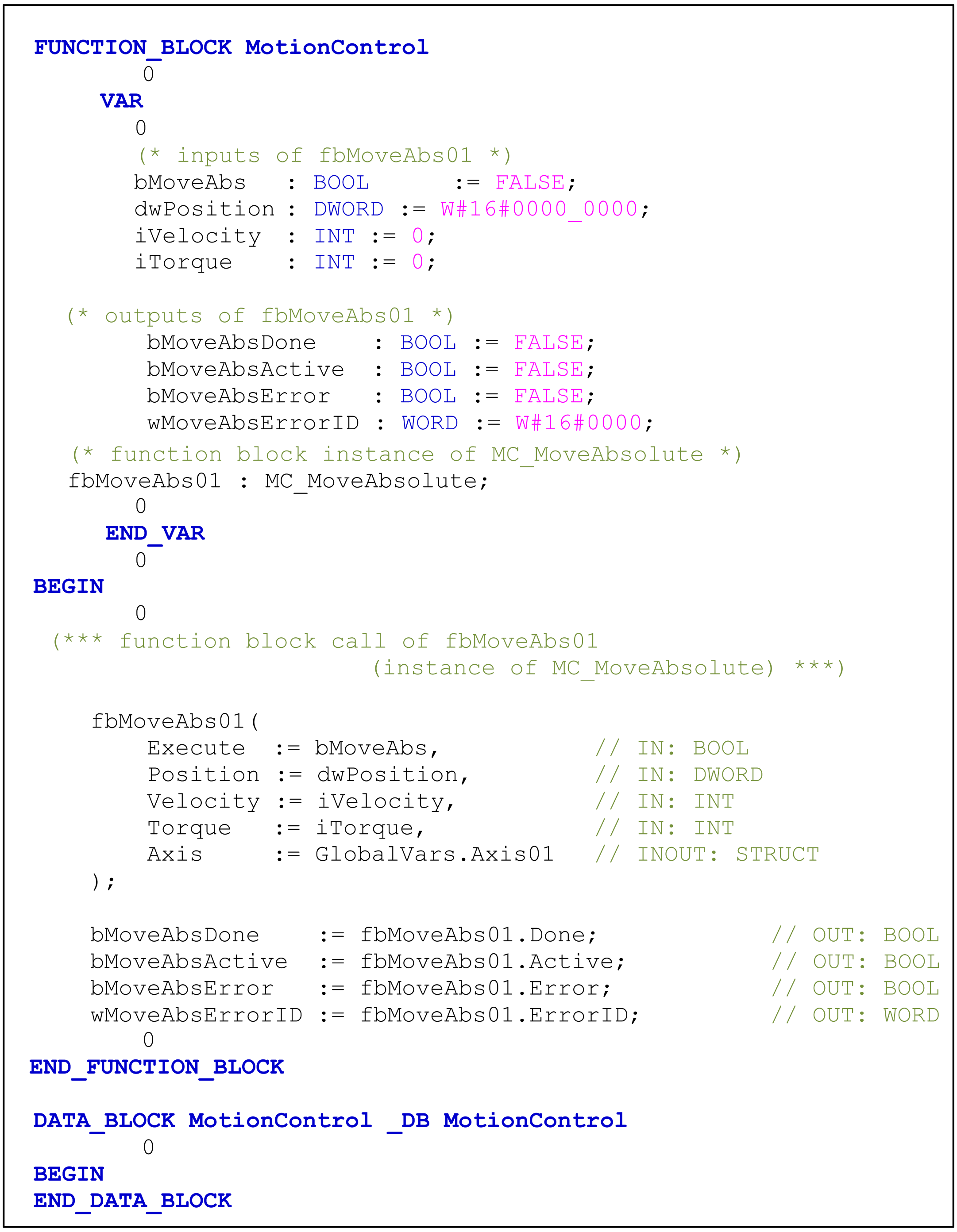
```
FUNCTION_BLOCK MotionControl
        0
   VAR
       0
       (* inputs of fbPower01 *)
       bPower           : BOOL := TRUE;

       (* outputs of fbPower01 *)
        bPowerStatus    : BOOL := TRUE;
        bPowerError     : BOOL := TRUE;
        wPowerErrorID   : WORD := W#16#0000;

       (* function block instance of MC_Power *)
       fbPower01 : MC_Power;
       0
     END_VAR
       0
BEGIN
       0
(*** function block call of fbPower01
                              (instance of MC_Power) ***)

    fbPower01(
        Enable  := bPower,              // IN: BOOL
        Axis    := GlobalVars.Axis01    // INOUT: STRUCT
    );

    bPowerStatus  := fbPower01.Status;   // OUT: BOOL
    bPowerError   := fbPower01.Error;    // OUT: BOOL
    wPowerErrorID := fbPower01.ErrorID;  // OUT: WORD
        0
END_FUNCTION_BLOCK

DATA_BLOCK MotionControl _DB MotionControl
        0
BEGIN
END_DATA_BLOCK
```

Fig. 142: instance call of MC_Power in SCL

In this case the drive power can be switched on, by setting the variable “**bPower**” to TRUE. Setting the same variable back to FALSE will disconnect the power from the drive.

4.3 How to move drive to an absolute position

- set up the communication with the drive (section 4.1)
- Add and set up an instance of MC_Power (section 4.2)
- Add an instance of MC_MoveAbsolute to motion program and set up in- and output variables



```
FUNCTION_BLOCK MotionControl
        0
     VAR
        0
        (* inputs of fbMoveAbs01 *)
        bMoveAbs    : BOOL      := FALSE;
        dwPosition : DWORD := W#16#0000_0000;
        iVelocity  : INT := 0;
        iTorque    : INT := 0;

   (* outputs of fbMoveAbs01 *)
         bMoveAbsDone     : BOOL := FALSE;
         bMoveAbsActive   : BOOL := FALSE;
         bMoveAbsError    : BOOL := FALSE;
         wMoveAbsErrorID : WORD  := W#16#0000;
   (* function block instance of MC_MoveAbsolute *)
   fbMoveAbs01 : MC_MoveAbsolute;
        0
     END_VAR
        0
BEGIN
        0
 (*** function block call of fbMoveAbs01
                       (instance of MC_MoveAbsolute) ***)

    fbMoveAbs01(
        Execute  := bMoveAbs,            // IN: BOOL
        Position := dwPosition,          // IN: DWORD
        Velocity := iVelocity,           // IN: INT
        Torque   := iTorque,             // IN: INT
        Axis     := GlobalVars.Axis01    // INOUT: STRUCT
    );

    bMoveAbsDone     := fbMoveAbs01.Done;            // OUT: BOOL
    bMoveAbsActive   := fbMoveAbs01.Active;          // OUT: BOOL
    bMoveAbsError    := fbMoveAbs01.Error;           // OUT: BOOL
    wMoveAbsErrorID := fbMoveAbs01.ErrorID;          // OUT: WORD
        0
END_FUNCTION_BLOCK

DATA_BLOCK MotionControl _DB MotionControl
        0
BEGIN
END_DATA_BLOCK
```

Fig. 143: instance call of MC_MoveAbsolute in SCL

- **To finally start the motion:**
  - switch on drive power
  - set target position to “**dwPosition**” variable
  - set target velocity to “**iVelocity**” variable (possible values: 0...100 � 0% - 100% of conf. velocity)
  - start the motion with a positive slope at the “**bMoveAbs**” input

- **Drive will stop, when target position is reached**

4.4 How to move drive with a constant velocity

- set up the communication with the drive (section 4.1)
- Add and set up an instance of MC_Power (section 4.2)
- Add an instance of MC_MoveVelocity to motion program and set up in- and output variables

```
FUNCTION_BLOCK MotionControl
       0
   VAR
       0
       (* inputs of fbMoveVel01 *)
       bMoveVel    : BOOL := FALSE;
       iVelocity  : INT  := 0;
       iTorque    : INT  := 0;
       iDirection : INT  := 0;
       bPosLimits : BOOL := FALSE;

       (* outputs of fbMoveVel01 *)
       bMoveInVelocity : BOOL := FALSE;
       bMoveVelActive   : BOOL := FALSE;
       bMoveVelError    : BOOL := FALSE;
       wMoveVelErrorID : WORD := W#16#0000;

       (* function block instance of MC_MoveVelocity *)
       fbMoveVel01 : MC_MoveVelocity;
       0
       END_VAR
       0
BEGIN
       0

 (*** function block call of fbMoveVel01
                            (instance of MC_MoveVelocity) ***)
    fbMoveVel01(
        Execute    := bMoveVel,          // IN: BOOL
        Velocity  := iVelocity,          // IN: INT
        Torque    := iTorque,            // IN: INT
        Direction := iDirection,         // IN: INT
        PosLimits := bPosLimits,         // IN: BOOL
        Axis      := GlobalVars.Axis01  // INOUT: STRUCT
    );

    bMoveInVelocity := fbMoveVel01.InVelocity;      // OUT: BOOL
    bMoveVelActive  := fbMoveVel01.Active;          // OUT: BOOL
    bMoveVelError   := fbMoveVel01.Error;           // OUT: BOOL
    wMoveVelErrorID := fbMoveVel01.ErrorID;         // OUT: WORD
       0
END_FUNCTION_BLOCK

DATA_BLOCK MotionControl _DB MotionControl
       0
BEGIN
END_DATA_BLOCK
```

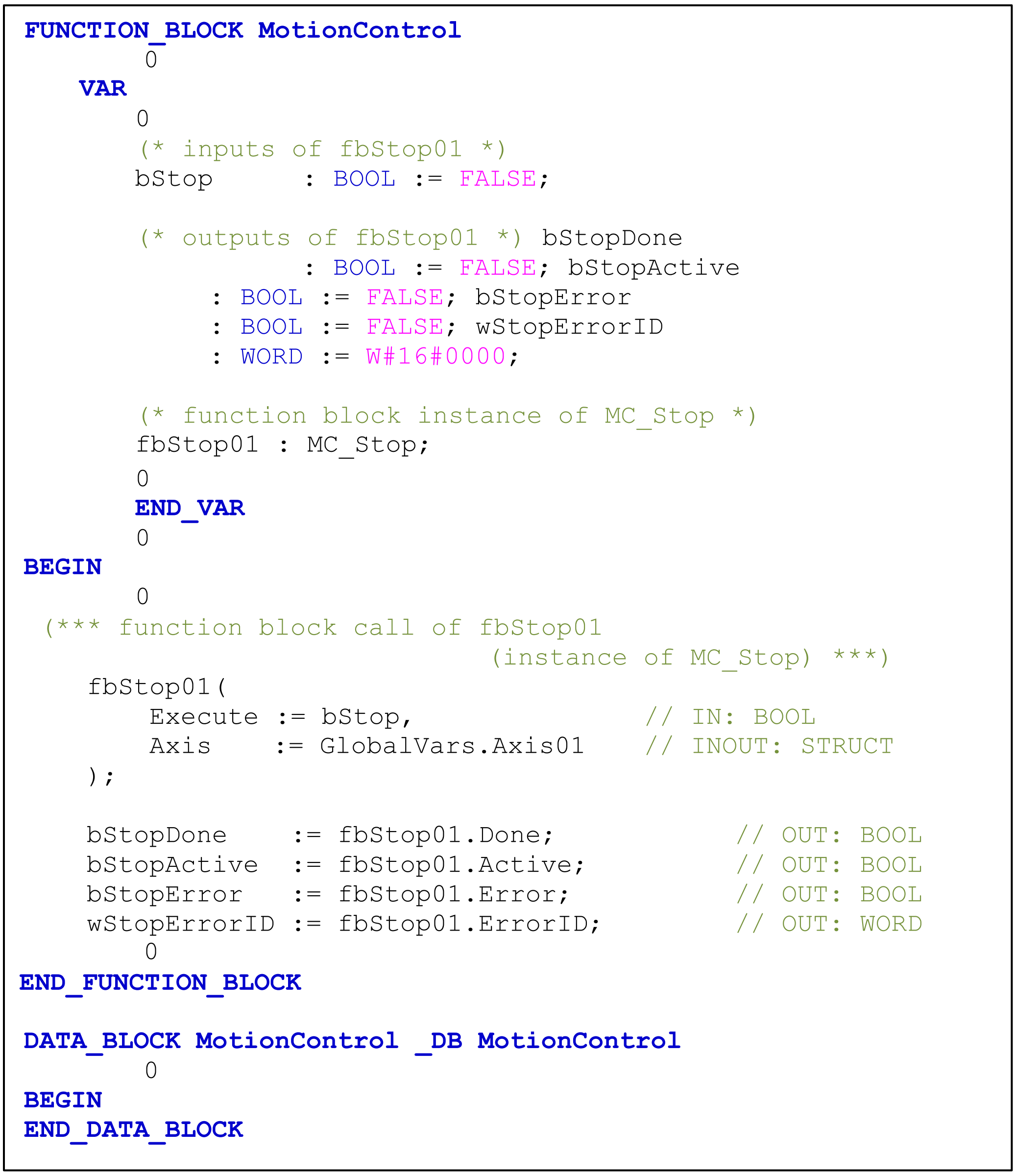
Fig. 144: instance call of MC_MoveVelocity in SCL

- **To finally start the motion:**
  - switch on drive power
  - set target velocity to “**iVelocity**” variable
    (possible values: 0..100 → 0% - 100% of conf. velocity)
  - set torque to “**iTorque**” variable
  - set direction of rotation to “**iDirection**” variable (0 → CCW; 1 → CW)
  - (De-)Activate position limits by setting “**bPosLimits**” variable
  - start the motion with a positive slope at the “**bMoveVel**” input

Caution! In this mode, the drive will NOT stop automatically! To stop the motion, you will need an instance of “MC_Stop”. For further information see next section. (section 4.5)

4.5 How to stop the axis from a continous motion or while moving to an new postion

- Add an instance of MC_Stop to motion program



```
FUNCTION_BLOCK MotionControl
       0
   VAR
       0
       (* inputs of fbStop01 *)
       bStop       : BOOL := FALSE;

       (* outputs of fbStop01 *) bStopDone
                   : BOOL := FALSE; bStopActive
             : BOOL := FALSE; bStopError
             : BOOL := FALSE; wStopErrorID
             : WORD := W#16#0000;

       (* function block instance of MC_Stop *)
       fbStop01 : MC_Stop;
       0
       END_VAR
       0
BEGIN
       0
 (*** function block call of fbStop01
                        (instance of MC_Stop) ***)
    fbStop01(
        Execute := bStop,             // IN: BOOL
        Axis    := GlobalVars.Axis01  // INOUT: STRUCT
    );

    bStopDone    := fbStop01.Done;           // OUT: BOOL
    bStopActive  := fbStop01.Active;         // OUT: BOOL
    bStopError   := fbStop01.Error;          // OUT: BOOL
    wStopErrorID := fbStop01.ErrorID;        // OUT: WORD
       0
END_FUNCTION_BLOCK

DATA_BLOCK MotionControl _DB MotionControl
       0
BEGIN
END_DATA_BLOCK
```

Fig. 145: instance call of MC_Stop in SCL

- positive slope at “bStop” will stop the drive from any motion (axis will transferred to state “Stopping”)

- As long as the “bStop” stays TRUE, it will not be possible to switch of the drive power or to start a new motion. If “fbPower01” will be deactivated while “fbStop01” is active (bStop = TRUE), the drives power will stay switched on until bStop = FALSE.

4.6 How to read drive errors

- To read drive-internal errors we need a set up communication (section 4.1) and an instance of “MC_ReadAxisError” (it is not necessary to switch on the drive power with “MC_Power” in order to read drive errors)

```
FUNCTION_BLOCK MotionControl
       0
   VAR
       0
       (* inputs of fbReadErr01 *)
       bReadErr01 : BOOL := FALSE;

       (* outputs of fbReadErr01 *)
       bReadErrValid    : BOOL := FALSE;
       bReadErrActive   : BOOL := FALSE;
       bReadErrError    : BOOL := FALSE;
       wReadErrErrorID : WORD := W#16#0000;
       arAxisErrorID    : ARRAY[0..7] OF WORD := 0(W#16#0000);

        (* function block instance of MC_ReadAxisError *)
        fbReadErr01 : MC_ReadAxisError;
        0
        END_VAR
        0
BEGIN
        0
(*** function block call of fbReadErr01
                        (instance of MC_ReadAxisError) ***)

    fbReadErr01(
        Enable := bReadError,     // IN: BOOL
        Axis := GlobalVars.Axis01 // INOUT: STRUCT
    );

    bReadErrValid   := fbReadErr01.Valid;       // OUT: BOOL
    bReadErrActive  := fbReadErr01.Active;      // OUT: BOOL
    bReadErrError   := fbReadErr01.Error;       // OUT: BOOL
    wReadErrErrorID := fbReadErr01.ErrorID;     // OUT: WORD
    arAxisErrorID   := fbReadErr01.AxisErrorID; // OUT: ARRAY
        0
END_FUNCTION_BLOCK

DATA_BLOCK MotionControl _DB MotionControl
        0
BEGIN
END_DATA_BLOCK
```

Fig. 146: instance call of MC_ReadError in SCL

- The reading of drive-internal errors will start with a positive slope at "**bReadError**"
- Drive errors will be continuously read, until "bReadError" = FALSE
- **While "fbReadErr01" is active, it will be not possible to read or write parameter or to start a new motion at the configured axis**

4.7 How to read a drive parameter

- To read a drive parameter we need a set up communication (section 4.1) and an instance of "MC_ReadParameter". It is not necessary to switch on the drive power with "MC_Power" in order to read drive parameters.

```
FUNCTION_BLOCK MotionControl
        0
   VAR
       0
       (* inputs of fbReadPara01 *)
       (* inputs of fbReadPara01 *)
        bReadPara          : BOOL   := FALSE;
        iParameterNo       : INT    := 0;
        iSubindex          : INT    := 0;

        (* outputs of fbReadPara01 *)
        bReadParaValid     : BOOL   := FALSE;
        bReadParaActive    : BOOL   := FALSE;
        bReadParaError     : BOOL   := FALSE;
        wReadParaErrorID   : WORD   := W#16#0000;
        dwReadParaValue    : DWORD  := W#16#0000_0000;
         (* function block instance of MC_ReadParameter *)
        fbReadPara01 : MC_ReadParameter;
        0
      END_VAR
        0
BEGIN
        0
 (*** function block call of fbReadPara01
 (instance of MC_ReadParameter) ***)

    fbReadPara01(
        Enable           := bReadPara,            // IN: BOOL
        ParameterNumber  := iParameterNo,         // IN: INT
        SubIndex         := iSubindex,            // IN: INT
        Axis             := GlobalVars.Axis01     // INOUT:
STRUCT
    );

   bReadParaValid    := fbReadPara01.Valid;       // OUT: BOOL
   bReadParaActive   := fbReadPara01.Active;      // OUT: BOOL
   bReadParaError    := fbReadPara01.Error;       // OUT: BOOL
   wReadParaErrorID  := fbReadPara01.ErrorID;     // OUT: WORD
   dwReadParaValue   := fbReadPara01.Value;       // OUT: DWORD
        0
END_FUNCTION_BLOCK

DATA_BLOCK MotionControl _DB MotionControl
        0
BEGIN
END_DATA_BLOCK
```

Fig. 147: instance call of MC_ReadParameter in SCL

- Fill the variables “iParameterNo” and “iSubindex” with the desired values
- The reading of the desired parameter will start with a positive slope at “bReadPara”
- the parameter will be continuously read, until “bReadPara” = FALSE
- “dwReadParValue” will contain the read parameter value, when “bReadParaValid” = TRUE
- While “fbReadPara01” is active, it will be not possible to read or write parameter, to read axis errors or to start a new motion at the configured axis

4.8 How to write a drive parameter

To write a drive parameter we need a set up communication (section 4.1) and an instance of “MC_WriteParameter”. It is not necessary to switch on the drive power with “MC_Power” in order to write drive parameters.

```
FUNCTION_BLOCK MotionControl
      0
   VAR
      0
      (* inputs of fbReadPara01 *)
      bWritePara          : BOOL:= FALSE;
      iParameterNo        : INT := 0;
      iSubindex           : INT := 0;
      dwWriteParaValue    : DWORD := W#16#0000_0000;

      (* outputs of fbReadPara01 *)
      bWriteParaDone      : BOOL       := FALSE;
      bWriteParaActive    : BOOL       := FALSE;
      bWriteParaError     : BOOL       := FALSE;
      wWriteParaErrorID   : WORD       :=
      W#16#0000;
      (* function block instance of MC_ReadParameter *)
      fbWritePara01: MC_WriteParameter;
      0
   END_VAR
      0
BEGIN
      0
      (*** function block call of fbWritePara01
      (instance of MC_WriteParameter) ***)

   fbWritePara01(
        Execute          := bWritePara,          // IN: BOOL
        ParameterNumber  := iParameterNo,        // IN: INT
        SubIndex         := iSubindex,           // IN: INT
        Value            := dwWriteParaValue,    // IN: DWORD
        Axis             := Axis01
                                             // INOUT: STRUCT
   );

   bWriteParaDone    := fbWritePara01.Done;     // OUT: BOOL
   bWriteParaActive  := fbWritePara01.Active;   // OUT: BOOL
   bWriteParaError   := fbWritePara01.Error;    // OUT: BOOL
   wWriteParaErrorID := fbWritePara01.ErrorID;  // OUT: WORD
        0
END_FUNCTION_BLOCK

DATA_BLOCK MotionControl _DB MotionControl
        0
BEGIN
END_DATA_BLOCK
```

Fig. 148: instance call of MC_WriteParameter in SCL

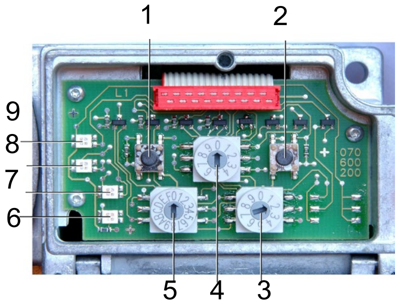
- Fill the variables “iParameterNo” and “iSubindex” with the desired values
- Set “dwWriteParaValue” with the new value for the parameter too
- Start the writing of the new value for the desired parameter with a positive slope at “bWritePara”
- While “fbReadPara01” is active, it will be not possible to read or write parameter, to read axis errors or to start a new motion at the configured axis

5 JOG mode

5.1 Jog-button on the drive

There are two switches on the back of the drive to enable movement of the drives when the control is non-operational. This only requires the supply voltage to be applied and the required switch to be operated (JOG mode). This does not require the presence of control signals.
Once the drive, via the control word, receives the information that the control requires overall control (CW = 04 00_{hex}) the switches will be disabled (release JOG mode).

5.2 Using the Jog-button on the HIPERDRIVE-HUB DP



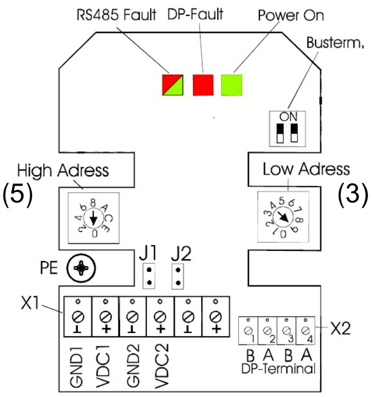
Button:
1 JOG- (counter-clockwise operation)
2 JOG + (clockwise operation) Rotary switch
3 field bus slave address
4 selection of the serial port (drive)
5 field bus slave address
LED
6 Power On
7 DP-Fault
8 diagnostic indicator
9 RS485 Fault

Figure 5-1: Control-Panel, HIPERDRIVE-HUB DP

- Set the field bus slave address to value 129 or set the state machine to STW=0x2B30. This enables the Manual Controlled Work Mode.
- Select the drive with the rotary switch 4 selection of the serial port (drive)
- Push the jog-button 1 for counter-clockwise operation or jog-button 2 for clockwise operation.
- The Jog-buttons on the back of the drive are in this mod usable too.

Danger, risk of injury: Don’t use the jog-button of the HUB and the Jog-button of the drive at the same time.

5.3 Using the Jog-button on the HIPERDRIVE



GND1, VDC1: 24VDC Motor
GND2, VDC2: 24VDC Busadaptor

J1:Conn. GND1-GND2
J2:Conn. VDC1-VDC2

- Set the field bus slave address to value 129. This enables the Manual Controlled Work Mode.
- Use the Jog-button of the drive

6 HIPERDRIVE parameter description

Please note that the guaranteed numbers of write-cycles to the internal E²PROM memory are physically limited to 1 million cycles. So please avoid continuous parameter write operation to the device.

New Parameters can only be written into the HIPERDRIVE when the status word has the following value: 2B30 $_{hex}$, 2B31 $_{hex}$, 2B33$_{hex}$ or 2B37$_{hex}$. resp. 0330 $_{hex}$, 0331 $_{hex}$, 0333$_{hex}$ or 0337$_{hex}$.

6.1 PROFIBUS address (parameter 918 [396$_{hex}$])

Data type: unsigned 16 bit, read-only.
Value range: 1..126
At this parameter number, the set PROFIBUS address can be read.

6.2 Operating mode (parameter 930 [3A2$_{hex}$])

Data type: unsigned 16 bit, read/write
This parameter is used to identify the operating mode. The HIPERDRIVE can be operated in two different modes:

- Operating mode = 1: speed control
- Operating mode = 2: positioning (Default value)

This parameter is stored in a nonvolatile memory. This means that after the supply voltage is connected, the parameter "operating mode" is prefilled with the default value 2 (positioning).
If the operating mode "speed control" is desired, this must be set by changing parameter 930 after the supply voltage has been connected.

6.3 Error buffer (parameter 945 [3B1$_{hex}$])

Data type: array [8], unsigned [16], read-only.
This parameter is defined as an array with 8 entries. Errors which occur in parameter 1009 [3F1 $_{hex}$] are entered in this array with their error code. Unlike Parameter 1009 [3F1 $_{hex}$], the error would not be reseted after the acknowledge of the error. This error code can be read via the parameter number and the specification of the Sub-index. The error codes are not saved and are lost after the supply voltage has been removed.
List of error codes:

Error code	Meaning
8200hex	Motor current or maxim. Torque exceeded
8201hex	Operating voltage below limit
8202hex	Overtemperature
8203hex	dynamic position deviation
8300hex	Drive Blocked (Direction CCW)
8301hex	Drive Blocked (Direction CW)
8304hex	Limit working area exceeded (CCW) - Minim. Position
8305hex	Limit working area exceeded (CW) - Maxim. Position
8401hex	Internal error: EEProm error
8410hex	Internal error: position registration
8501hex	Internal error: Invalid Value of parameter or process data.
8502hex	Internal error: wrong command sequence
8601hex	Timeout for communication drive - gateway

Figure 6-1: Error buffer (parameter number 945 [3B1hex]
Handling of “Limit working area exceeded” see parameter 1009

6.4 Number of errors (parameter 952 [3B8$_{hex}$])

Data type: unsigned 16 bit, read/write
With this parameter, the number of errors which have occurred can be read. The contents are not saved and are lost after the supply voltage has been removed.

If the value 0 is written to the parameter "Number of errors“, then the number of errors and the error codes registered are deleted.

6.5 Warning status of Base Device (parameter 953 [3B9$_{hex}$])

Data type: unsigned 16, read-only.
Warning status of Base Device according to the configuration of PNU 1018. If there is no valid connection to Base Device, a reduced information {0x80.00} will indicated.

Bit	Explanation
00	Motor current or maxim. Torque exceeded
01	Operating Voltage below limit
02	dynamic position deviation
03	Overtemperature, drive stops
04	Drive Blocked (Direction CCW)
05	Drive Blocked (Direction CW)
06	-- Not specified.
07	-- Not specified.
08	Limit working area exceeded (CCW) - Minim. Position
09	Limit working area exceeded (CW) - Maxim. Position
10	-- Not specified.
11	Internal error: EEProm error - correctable
12	Internal error: EEProm error - not correctable
13	Internal error: wrong command sequence
14	Invalid Value of parameter or process data.
15	Timeout for communication drive – gateway

Handling of “Limit working area exceeded” see parameter 1009

6.6 Hardware configuration (parameter 961 [3C1$_{hex}$])

Data type: unsigned 16 Bit, read-only
Via this parameter, the hardware configuration of the HIPERDRIVE can be read, e.g. 0200hex means Version 2.00

6.7 Profile number (parameter 965 [3C5$_{hex}$])

Data type: unsigned 16 Bit, read-only
This parameter indicates the version of the "variable-speed drives" profile. In the case of the HIPERDRIVE, this is 0200hex (Version 2.00).

6.8 CCW operating range limit (parameter 1000 [3E8$_{hex}$])

Data type: signed 32, read/write, Position data mapping
Maximum (=default) value for CCW operation range limit:
Negative value of absolute turns possible with the actual drive unit as written in the data sheet e.g.

HDA45A: -512 revolutions (=0xFE00 0000$_{hex}$), HRA08: -128 revolutions (=0xFF80 0000_{hex})

The position limiting value for counterclockwise running (CCW) predefines the minimum position for the HIPERDRIVE. If the current position of the HIPERDRIVE is outside the predefined limit, then a movement can be made only in the direction of the position limiting value.

Restrictions for setting the parameters:

The parameter "CCW operating range limit" must always be less than the parameter "CW operating range limit".
A change in the value of "CCW operating range limit" beyond the limits of the parameters "CW operating range limit" will be rejected by the HIPERDRIVE.

If you set the range limit to the absolute limit written in the datasheet the error “Limit working area exceeded (CCW) - Minimum Position” can’t be detected. In this case the Position is lost (roll-over).

6.9 CW operating range limit (parameter 1001 [3E9$_{hex}$])

Data type: signed 32, read/write
Maximum (=default) value for CW operating range limit:
Positive value of absolute turns possible with the actual drive unit as written in the data sheet e.g.

HDA45A: 512 revolutions (=0x0200 0000_{hex}),
HRA08: 128 revolutions (=0x0080 $0000_{hex)}$)

The position limiting value for clockwise running predefines the maximum position for the HIPERDRIVE. If the current position of the HIPERDRIVE is outside the predefined limit, then a movement can be made only in the direction of the position limiting value.

Restrictions for setting the parameters:

The parameter "CW operating range limit" must always be greater than the parameter "CCW operating range limit".
A change in the value of "CW operating range limit" beyond the limits of the parameters "CCW operating range limit" will be rejected by the HIPERDRIVE.

If you set the range limit to the absolute limit written in the datasheet the error “Limit working area exceeded (CW) - Maximum Position” can’t be detected. In this case the Position is lost (roll-over).

6.10 Speed limiting value (parameter 1002 [3EA$_{hex}$])

By using this parameter, the maximum possible travel speed of the HIPERDRIVE is predefined. The statement of the speed value in %, in the main setpoint for the HIPERDRIVE, relates to this parameter.
Value range[3]: 0 to maximum drive speed in rpm as written in the data sheet multiplied by factor 10, e.g. HDA45A: 0 to 350 (35.0*10).

6.11 Save new position (parameter 1004 [3EC$_{hex}$])

Data type: signed 32, read/write.
Default value: 00 00 00 00hex

[3] Depending on scaling factors, here: default value

The current absolute position of the HIPERDRIVE can be overwritten with a new arbitrary value, in order to match the motor data to the plant-specific conditions. Writing a value to this parameter has the effect of accepting this value as the current absolute position for the HIPERDRIVE.
Value range: +/- number of revolutions given in the data sheet of the corresponding drive unit, 65536 steps per revolution.

Restrictions for setting the parameter:

The value for the new position must lie only within the parameters "CW operating range limit" and "CCW operating range limit". A change in the value beyond the limits of the parameters "CW operating range limit" or "CCW operating range limit" will be rejected by the HIPERDRIVE.

6.12 HIPERDRIVE error status bits (parameter 1009 [3F1$_{hex}$])

Data type: unsigned 16, read-only.
Error-status of Base Device according to the configuration of PNU 1018.

Bit	Meaning
0	Motor current or maxim. Torque exceeded
1	Operating voltage below limit
2	dynamic position deviation
3	Overtemperature
4	Drive Blocked (Direction CCW)
5	Drive Blocked (Direction CW)
6	Reserved
7	Reserved
8	Limit working area exceeded (CCW) - Minim. Position
9	Limit working area exceeded (CW) - Maxim. Position
10	Internal error: position registration
11	Reserved
12	Internal error: EEProm error – not correctable.
13	Internal error: wrong command sequence
14	Invalid Value of parameter or process data.
15	Timeout for communication drive - gateway

Figure 6-2: HIPERDRIVE error status bits

Error-Handling Limit working area exceeded
- set FB MC_MoveAbsolute or MC_MoveVelocity disable
- toggle MC_Reset
- toggle MC_Power
- enable MC_MoveAbsolute or MC_MoveVelocity with motion direction into the working are

6.13 HIPERDRIVE status bits (parameter 1010 [3F2$_{hex}$])

Data type: unsigned 16, read-only.
This parameter represents the internal status bit of the HIPERDRIVE

Bit	Meaning
0	Reserved
1	Speed mode ready. This bit is "0" when the HIPERDRIVE is in the speed
2	Position mode ready. This bit is "0" when the HIPERDRIVE is in the position

3	Reserved
4	Reserved
5	Bit becomes "0“ when the end position is reached. In speed mode, this bit is not relevant.
6	Bit becomes "1” when speed mode is active, that is to say travel movement Running
7	Position mode active, that is to say travel movement running. Bit is "1“ until the end position is reached.
8	Reserved
9	Bit is “1“, if the JOG mode is active, i.e. drive can be moved via built-in JOG switches.
10-15	Reserved

Figure 6-3: HIPERDRIVE status bits

In case that there is no data connection with the base unit, e.g. no operating voltage or data line is faulty, error code 17, “Job cannot be executed because of operating status”, is returned.

6.14 DP diagnosis (parameter 1011 [3F3$_{hex}$])

Data type:	**Meaning**
00 00$_{hex}$	Diagnostic message OFF (default value after power-on).
00 01$_{hex}$	Diagnostic message ON

Figure 6-4: DP-diagnostics

With this parameter, the exchange of device-specific external diagnostic data can be enabled and disabled, respectively. If the routine is activated, 2 bytes of device-specific diagnostic data (Ext_Diag_Data) are read from the slave. This parameter may have to be reset after power-on.

6.15 Scaling “position values” (parameter 1015 [3F7$_{hex}$])

Scaling value to all attributes, associated to 'positioning'. - Numbers {1000, 1001, 1004}.
specification is {mm.mm.dd.dd }:
mm.mm = multiplier. Default: 01.00 $_{hex}$.
dd.dd = divisor. Default 00.01 $_{hex}$
not implemented in HRA08
(see Chapter 6.17)

6.16 Scaling “speed values” (parameter 1016 [3F8 $_{hex}$])

Scaling speed value.
specification is {mm.mm.dd.dd }:
mm.mm = multiplier. Default: 00.0A $_{hex}$.
dd.dd = divisor. Default 00.01 $_{hex}$
not implemented in HRA08
(see Chapter 6.17)

6.17 Position and velocity scaling, functionality

Some drives of the HIPERDRIVE product family support a scaling function of both, position & velocity. Both parameters are implemented as 32 Bit values consisting of a signed multiplier (High word) and an unsigned divider (low word).

The position factor refers to an internal resolution of [8 Bit / turn];
the velocity factor refers to [rpm].
The result of the scaling must be within
    [±31Bit] for the position value, and
    [±15Bit] for the velocity value
The factory settings are:

position factor:	256
velocity factor:	10

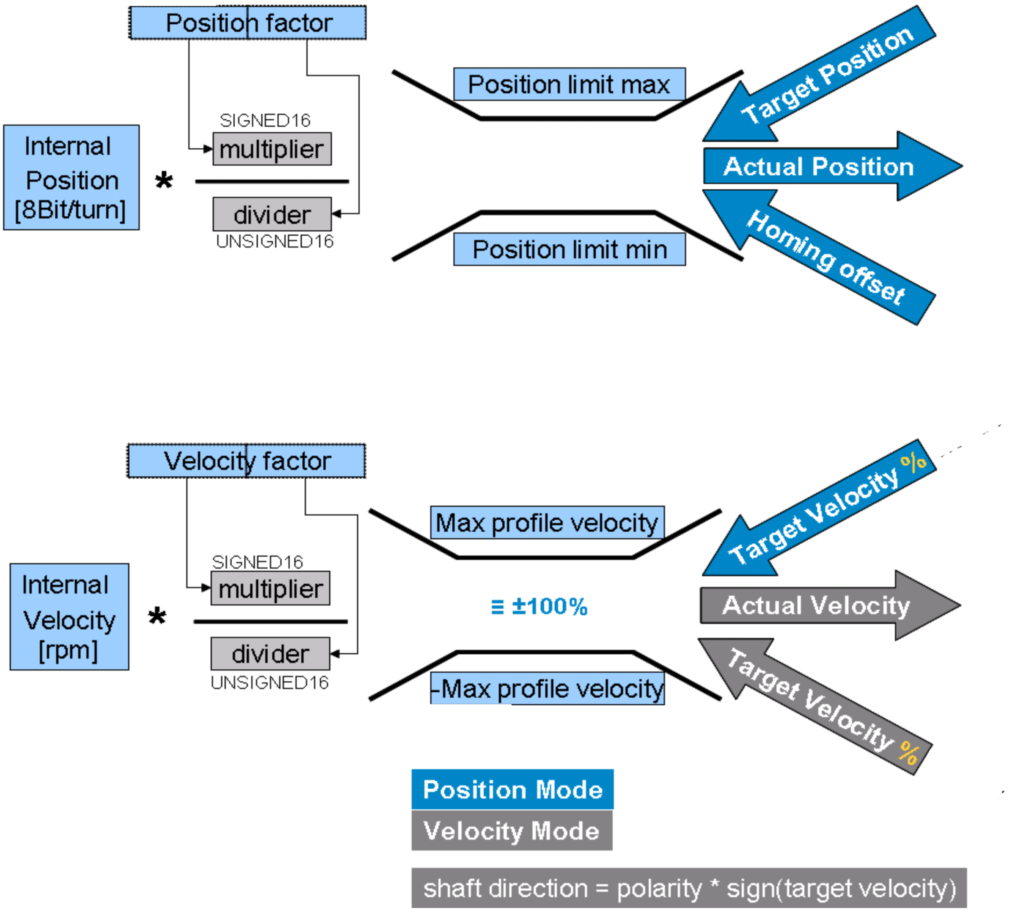


Figure 6-5: Position and velocity scaling

6.18 Holding / Quiescent Torque (parameter 1017 [3F9$_{hex}$])

Holding torque, generated by current during standstill. It is written in % of the maximum torque. Attention to the drive temperature in case of setting up the value, higher than the default value.
Only for HRA... family

6.19 Error/Warning Configuration (parameter 1018 [3FA$_{hex}$])

This parameter defines the behavior of error or warning indication. There is the possibility to show a failure indication as a 'Warning' or as a 'error'.
There are two possibilities to show a error/warning indication:
(1) Error: shown in parameter number 1009.
(2) Warning : shown in parameter 953.

A error, means a severe failure. There is generated an entry into the error stack. Also you have to acknowledge this situation by activating a specific Bit of the process data.
A warning, means an unimportant failure. There is no need for further actions to continue in process. The error stack is not influenced. The warning status disappears when the warning is no more present.
Each bit of this parameter defines the corresponding status of the parameters 1009 / 953.
Setting bit to 'zero': -- failure is shown as a warning.
Setting bit to 'one': -- failure is shown as a error.

Bit-No.	Explanation	Shown as Warning PNU 953	Shownas error PNU 1009
00	Motor current or maxim. Torque exceeded	(val = 0)	(val = 1)
01	Operating Voltage below limit	(val = 0)	(val = 1)
02	dynamic position deviation	(val = 0)	(val = 1)
03	Overtemperature	(val = 0)	(val = 1)
04	Drive Blocked (Direction CCW)	(val = 0)	(val = 1)
05	Drive Blocked (Direction CW)	(val = 0)	(val = 1)
06	-- Not specified.	---	---
07	-- Not specified.	---	---
08[4]	Limit working area exceeded (CCW) - Minim. Position	(val = 0 / 1)	(val = 1)
09[5]	Limit working area exceeded (CW) - Maxim. Position	(val = 0 / 1)	(val = 1)
10:	Internal error: position registration	---	(val = 0 / 1)
11	Internal error: EEProm error - correctable.	(val = 0 / 1)	---
12	Internal error: EEProm error - Not correctable.	(val = 0)	(val = 1)
13	Internal error: wrong command sequence	(val = 0)	(val = 1)
14	Invalid Value of parameter or process data.	(val = 0 / 1)	---
15[6]	Timeout for communication drive – gateway	X	X

Figure 6-6: error Configuration

The Default setting is: $973F_{hex}$ (subject to alterations)

Note: If there is a power failure of the drive power and the drive was not in motion, the HUB acknowledge a motion command, wait until the drive is powered up again and execute the command.

[4] This type of error/warning is shown in addition as a warning (independent the configuration), if Drive is in standstill or moving from outside the boundary towards work area.

[5] This type of error/warning is shown in addition as a warning (independent the configuration), if Drive is in standstill or moving from outside the boundary towards work area.

[6] This type of error/warning is shown as a warning or an error. The definition is not configurable. If there is no drive task active (Positioning or Velocity Mode) the HUB shows a warning. If there is a drive task active, the HUB shows a fault.

6.20 Reset operating parameters to factory default values (Parameter 1019 [3FBhex])

Data type: Data unsigned 16, write only
Via a write procedure to this parameter, all variable parameters are set to factory default values. The value of the written data is without any meaning.

Please note that this service takes approx. 400msec execution time. During this time no further command can be executed.

List of this parameter:

- CCW operating range limit (parameter number 1000 [3E8hex])
- CW operating range limit (parameter number 1001 [3E9hex])
- Scaling "position values" (parameter number 1015 [3F7 hex])
- Scaling "speed values" (parameter number 1016 [3F8 hex])
- Speed limiting value (parameter number 1002 [3EAhex])

6.21 Item number of the bus gateway (Parameter 1020 [$3FC_{hex}$])

Data type: array [3], unsigned 32, read-only
Via this parameter, the item number of the bus gateway can be read. The item number is composed of a total of 12 bytes and is stored in an array[3] with sub-index 0 to subindex 2

Octet 1											Octet 12
MSB			LSB	MSB			LSB	MSB			LSB
Sub-Index 0				Sub-Index 1				Sub-Index 2			

Figure 6-7: Item number of the bus gateway

The characters should be considered as ASCII encoded byte by byte.
Example:

Octet 1											Octet 12
32_{hex}	35_{hex}	37_{hex}	$4F_{hex}$	50_{hex}	31_{hex}	31_{hex}	41_{hex}	53_{hex}	32_{hex}	30_{hex}	31_{hex}
"2"	"5"	"7"	"O"	"P"	"1"	"1"	"A"	"S"	"2"	"0"	"1"

Figure 6-8: example to Figure 6-6

6.22 Serial number of the bus gateway (Parameter 1021 [$3FD_{hex}$])

Serial number Data type: array [2], unsigned 32, read-only
This parameter contains the serial number of the bus gateway. The serial number is composed of a total of 8 bytes and is stored in an array[2] with sub-index 0 to sub-index 1.

Octet 1							Octet 8
MSB			LSB	MSB			LSB
Sub-Index 0				Sub-Index 1			

Figure 6-9: Serial number of the bus gateway

The characters should be considered as ASCII encoded byte by byte. .
Example:

Octet 1							Octet 8
30_{hex}	30_{hex}	30_{hex}	30_{hex}	30_{hex}	30_{hex}	32_{hex}	31_{hex}
"0"	"0"	"0"	"0"	"0"	"0"	"2"	"1"

Figure 6-10: Example to Figure 6-8

6.23 Production date of the bus gateway (Parameter 1022 [3FE$_{hex}$])

Data type: array [2], unsigned 32, read-only
The production date of the bus gateway is composed of a total of 8 bytes and is stored in an array[2] with sub-index 0 to sub-index 1

Octet 1							Octet 8
MSB			LSB	MSB			LSB
Sub-Index 0				Sub-Index 1			

Figure 6-11: Production date of the bus gateway

The characters should be considered as ASCII encoded byte by byte.
Example:

Octet 1							Octet 8
30_{hex}	30_{hex}	30_{hex}	30_{hex}	30_{hex}	37_{hex}	30_{hex}	35_{hex}
“0”	“0”	“0”	“0”	“0”	“7”	“0”	“5”
No meaning				07.05 (week07 / year 2005)			

Figure 6-12: Example to Figure 6-10

6.24 Software version of the bus gateway (Parameter 1023 [3FF$_{hex}$])

Data type: array[3], unsigned 32, read-only
The number of the software version of the bus gateway comprises a total of 12 bytes, which are stored in an array[3] with sub-index 0 to sub-index 2.

Octet 1											Octet 12
MSB			LSB	MSB			LSB	MSB			LSB
Sub-Index 0				Sub-Index 1				Sub-Index 2			

Figure 6-13: Software version of the bus gateway

The characters should be considered as ASCII encoded byte by byte.
Example:

Octet 1											Octet 12
53_{hex}	57_{hex}	$2E_{hex}$	$5F_{hex}$	52_{hex}	45_{hex}	56_{hex}	$5F_{hex}$	31_{hex}	$2E_{hex}$	32_{hex}	30_{hex}
“S”	“W”	“.”	“_”	“R”	“E”	“V”	“_”	“1”	“.”	“2”	“0”

Figure 6-14: Example to Figure 6-12

6.25 Nominal voltage of the HIPERDRIVE (Parameter 1030 [406$_{hex}$])

Data type: unsigned 32, read-only
Via this parameter, the nominal value of the supply voltage of the HIPERDRIVE can be read.
The characters should be considered as ASCII encoded byte by byte
Example:

Octet 1			Octet 4
30_{hex}	30_{hex}	32_{hex}	34_{hex}
“0”	“0”	“2”	“4”
0024 => 24V			

Figure 6-15: Nominal voltage of the HIPERDRIVE

6.26 Nominal current of the HIPERDRIVE (Parameter 1031 [407_{hex}])

Data type: unsigned 32, read-only
Via this parameter, the nominal value of the supply current of the HIPERDRIVE can be read.
The characters should be considered as ASCII encoded byte by byte. See the related datasheet.

Example:

Octet 1			Octet 4
30_{hex}	30_{hex}	34_{hex}	38_{hex}
“0”	“0”	“4”	“8”
0048 => 4,8A			

Figure 6-16: Nominal current of the HIPERDRIVE

6.27 Nominal torque of the HIPERDRIVE (parameter 1032 [408_{hex}])

Data type: unsigned 32, read-only
Via this parameter, the nominal value of the torque of the HIPERDRIVE can be read.
The characters should be considered as ASCII encoded byte by byte. If the Value is e.g.: 1,2 Nm a “A” is written to Octe1, showing, that the value must divided by 10
Example:

Octet 1			Octet 4
30_{hex}	30_{hex}	31_{hex}	35_{hex}
“0”	“0”	“1”	“5”
0015 => 15Nm			
Octet 1			Octet 4
41_{hex}	30_{hex}	31_{hex}	32_{hex}
“A”	“0”	“1”	“2”
0015 => 1,2Nm			

Figure 6-17: Nominal torque of the HIPERDRIVE

6.28 Nominal speed of the HIPERDRIVE (Parameter 1033) [409_{hex}])

Data type: unsigned 32, read-only
Via this parameter, the nominal value of the speed of the HIPERDRIVE can be read.
The characters should be considered as ASCII encoded byte by byte.
Example:

Octet 1			Octet 4
30_{hex}	30_{hex}	32_{hex}	37_{hex}
“0”	“0”	“2”	“7”
0027 => 27 revolution per minute			

Figure 6-18: Nominal speed of the HIPERDRIVE

6.29 Item number of the HIPERDRIVE (Parameter 1035 [40B$_{hex}$])

Data type: array [3], unsigned 32, read-only

Via this parameter, the item number of the HIPERDRIVE can be read. The item number is composed of a total of 12 bytes and is stored in an array[3] with sub- index 0 to subindex 2.

Octet 1											Octet 12
MSB			LSB	MSB			LSB	MSB			LSB
Sub-Index 0				Sub-Index 1				Sub-Index 2			

Figure 6-19: Item Number of the HIPERDRIVE

The characters should be considered as ASCII encoded byte by byte.
Example:

Octet 1											Octet 12
32_{hex}	35_{hex}	37_{hex}	$4F_{hex}$	50_{hex}	31_{hex}	31_{hex}	41_{hex}	53_{hex}	32_{hex}	30_{hex}	31_{hex}
“2”	“5”	“7”	“O”	“P”	“1”	“1”	“A”	“S”	“2”	“0”	“1”

Figure 6-20: Example to Figure 8-20

6.30 Serial number of the HIPERDRIVE (Parameter 1036 [40C$_{hex}$])

Serial number Data type: array [2], unsigned 32, read-only
This parameter contains the serial number of the basic drive unit. The serial number is composed of a total of 8 bytes and is stored in an array[2] with sub-index 0 to sub-index 1.

Octet 1							Octet 8
MSB			LSB	MSB			LSB
Sub-Index 0				Sub-Index 1			

Figure 6-21: Serial number of the HIPERDRIVE

The characters should be considered as ASCII encoded byte by byte.
Example:

Octet 1							Octet 8
30_{hex}	30_{hex}	30_{hex}	30_{hex}	30_{hex}	30_{hex}	32_{hex}	31_{hex}
“0”	“0”	“0”	“0”	“0”	“0”	“2”	“1”

Figure 6-22: Example to the HIPERDRIVE

6.31 Production date of the HIPERDRIVE (Parameter 1037 [40D$_{hex}$])

Data type: array [2], unsigned 32, read-only
The production date of the basic drive unit is composed of a total of 8 bytes and is stored in an array[2] with sub-index 0 to sub-index 1.

Octet 1							Octet 8
MSB			LSB	MSB			LSB
Sub-Index 0				Sub-Index 1			

Figure 6-23: Production Date of the HIPERDRIVE

The characters should be considered as ASCII encoded byte by byte.
Example:

Octet 1							Octet 8
30_{hex}	30_{hex}	30_{hex}	30_{hex}	30_{hex}	37_{hex}	30_{hex}	35_{hex}
“0”	“0”	“0”	“0”	“0”	“7”	“0”	“5”
No meaning				07.05 (week07 / year 2005)			

Figure 6-24: Example to Figure 6-22

6.32 Software version of the HIPERDRIVE (Parameter 1038 [40E$_{hex}$])

Data type: array[3], unsigned 32, read-only
The number of the software version of the basic drive unit comprises a total of 12 bytes, which are stored in an array[3] with sub-index 0 to sub-index 2.

Octet 1											Octet 12
MSB			LSB	MSB			LSB	MSB			LSB
Sub-Index 0				Sub-Index 1				Sub-Index 2			

Figure 6-25: Software version of the HIPERDRIVE

The characters should be considered as ASCII encoded byte by byte.
Example:

Octet 1											Octet 12
53_{hex}	57_{hex}	$2E_{hex}$	$5F_{hex}$	52_{hex}	45_{hex}	56_{hex}	$5F_{hex}$	31_{hex}	$2E_{hex}$	32_{hex}	30_{hex}
“S”	“W”	“.”	“_”	“R”	“E”	“V”	“_”	“1”	“.”	“2”	“0”

Figure 6-26: example to Figure 6-24

6.33 BusCfgControlNode, (Parameter 1090 [442 $_{hex}$])

Read / write
Reset complete Bus-IF Adapter.--Software Reset.
Simulates a Re-Start. Takes current EEProm
configuration. Value only 0x1D

6.34 Reset to HD Base Device, (Parameter 1091 [443$_{hex}$])

'Reset' service (Software Reset) to HD Base Device. Simulates a Re-Start. Takes current EEProm configuration. Service is used to validate parameters which need a restart for update.
Read service: returns value zero(0).

6.35 Ident ProdNumber, (Parameter 1092 [444$_{hex}$])

Meaning: DP-Ident-Number
Value: STEGxxxx

6.36 Ident ModulType, (Parameter 1093 [445$_{hex}$])

Meaning: Adapter Module Type (Hardware Layout).
Value: { 0, 1, 2}, Single (1), Multi (2)

6.37 Ident NumCfgStage, (Parameter 1094 [446$_{hex}$])

Meaning: Size of configuration stage (number of slave units supported by the gateway).
Value: { 1...8}, 8

6.38 Ident NetProtBaudr, (Parameter 1095 [447$_{hex}$])

Protocol specification to RS-485 network.
Meaning: Bit_7..4: Protocol
selection. Value:{ 0...7}, 0
Meaning: Bit_3..0: Baudrate selection.
Value: { 1...3}, 1

6.39 IdentModulFailStatus, (Parameter 1096 [448$_{hex}$])

Flag shows Failure Status of Adapter Module.
Meaning: Bit_15-8: - -
Meaning: Bit_7: number of connected units > configuration stage.
Meaning: Bit_6..2: - -
Meaning: Bit_1: faulted "Eeprom_Save“ operation (write).
Meaning: Bit_0: faulted "Eeprom_Restore“ operation (read).
Set the field bus slave address to value 129. This enables the Manual Controlled Work Mode. Than you can use the Jog-button on the back of the drive

7 Tables

7.1 Parameter

Parameter number [DEC]	meaning	Data type	Access
918	PROFIBUS address	Unsigned 16	R
930	Operating mode	Unsigned 16	R/W
945	Error buffer	Array [8] Unsigned 16	R
952	Number of errors	Unsigned 16	R/W
953	Warning status of Base Device	Unsigned 16	R
961	Hardware configuration	Unsigned 16	R
965	Profile Number	Octet-String 2	R
1000	CCW operating range limit	Signed 32	R/W
1001	CW operating range limit	Signed 32	R/W
1002	Speed limiting value	Unsigned 16	R/W
1003	Maximum torque	Unsigned 16	R/W
1004	Save new position	Signed 32	R/W
1009	HIPERDRIVE status bits	Unsigned 16	R
1010	HIPERDRIVE status information	Unsigned 16	R
1011	DP-diagnostics	Unsigned 16	R/W
1015	Scaling “position values” (not implemented in HRA08)	Signed 32 (see Chapter 6.17)	R/W
1016	Scaling “speed values” (not implemented in HRA08)	Signed 32 (see Chapter 6.17)	R/W
1017	Holding / Quiescent Torque. (only HRA... family)	Unsigned 16	R/W
1018	Error Configuration	Unsigned 16	R/W
1019	Reset parameters to default values	Unsigned 16	W
1020	Item number of the bus gateway	Array[3] Unsigned 32	R
1021	Serial number of the bus gateway	Array[2] Unsigned 32	R

1022	Production date of the bus gateway	Array[2] Unsigned 32	R
1023	Software version of the bus gateway	Array[3] Unsigned 32	R
1030	Nominal voltage of the HIPERDRIV	Unsigned 32	R
1031	Nominal current of the HIPERDRIV	Unsigned 32	R
1032	Nominal torque of the HIPERDRIVE	Unsigned 32	R
1033	Nominal speed of the HIPERDRIVE	Unsigned 32	R
1035	Item number of the HIPERDRIVE	Array[3] Unsigned 32	R
1036	Serial number of the drive	Array[2] Unsigned 32	R
1037	Production date of the drive	Array[2] Unsigned 32	R
1038	Software version of the drive	Array[3] Unsigned 32	R
1090	BusCfgControlNode; Software reset, Value only 0x1D	Unsigned 16	R/W
1091	Reset to HD Base Device	Unsigned 16	R/W
1092	Ident ProdNumber	Unsigned 16	R
1093	Ident ModulType	Unsigned 16	R
1094	Ident NumCfgStage	Unsigned 16	R/W
1095	Ident NetProtBaudr	Unsigned 16	R/W
1096	IdentModulFailStatus	Unsigned 16	R

Figure 7-1: List of Parameter

7.2 Baud rates supported

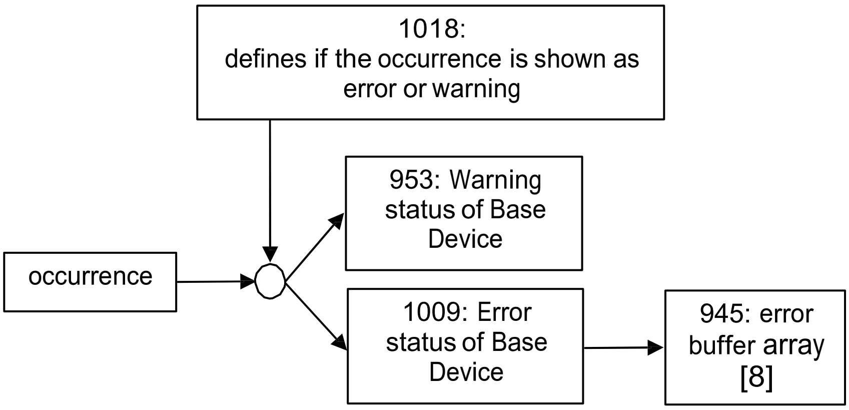
HIPERDRIVE supports the following baud rates:
9.6 kBd, 19.2 kBd, 93.75 kBd, 187.5 kBd, 500 kBd, 1.5 Mbd, 3 Mbd, 6 Mbd, 12 MBd.

7.3 List of abbreviations

AK	task or response ID (for encoding, see tables 3.1. and 3.2) From German: Auftragskennung / Antwortkennung
FB	Function Block
FBD	Function Block Diagram
HW	High Word
LSB	Least Significant Bit / Byte
LW	Low Word
MSB	Most Significant Bit / Byte
PKE	Parameter ID
PKW	Parameter data frame / area From German: Parameter – Kennung – Wert
PNU	Parameter Number
PWE	Parameter Value (From German: Parameterwert)
Pos-act	Position, actual value
Pos-min	Position minimum
PPO	Process Parameter Data Object
PWE	Parameter-value (from German: Parameterwert)
PZD	process data frame / area
SCL	Structured Control Language (aka. ST � Structured Text)
SPM	toggle bit for instantaneous message
CW	Control Word
v-act	speed, actual value
v-max	maximum speed
SW	Status Word

8 LED-status and Error-Handling

An error condition, which can include one or more error messages, leads to the drive stopping. In order to store the error messages, a error buffer is defined. The number of errors is stored in parameter 952. If this is reset to "0“, then the entire error buffer is deleted.
An error condition has been cleared when all the errors that are present have been eliminated and the error has been acknowledged. Acknowledgement is implemented via an edge change in bit 7 in the control word. The error buffer can be read out by means of an increasing Sub-index.
Error / Warning parameter and their meaning:


1018:
defines if the occurrence is shown as error or warning
953: Warning status of Base Device
occurrence
1009: Error status of Base Device
945: error buffer array [8]

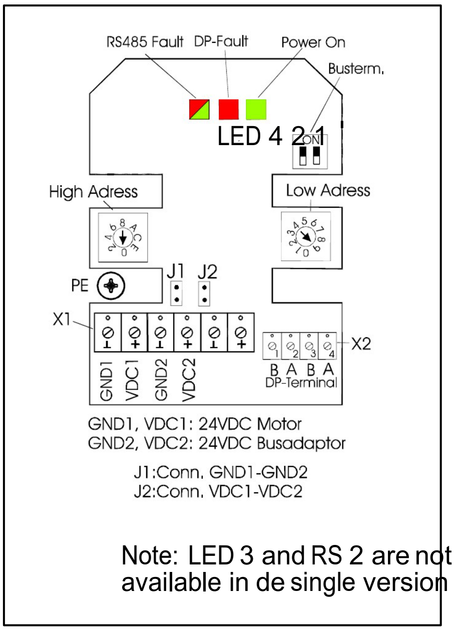
8.1 In general

Check cabling [7] (see figure 6.4)
Check the voltage of the power supply
Please ensure that the termination resistor is inserted at the end of the PROFIBUS DP line.
Check the required address on the PROFIBUS DP in accordance with your plant project engineering. Please note that a change in the switch position becomes effective only after the voltage supply has been switched off and on again.
Look at the LED status and check the application according to the table in chapter 8.2.
If the bus-communication works, read out the error and check the application according to the table in Chapter 8.3.
check your system by means of the diagnostic software of your bus master.
If you need support, please answer the following Questions
Please describe the symptoms of the error
Which drive is used? HDA30, 45, 70, HRA08
Was the drive once working properly? If so, what has changed.
How is the status of the LED
Is the drive running using the jog-button?
Does the drive have Communication with the master
What Error bits are set
How is the proceeding, especially describe the start up sequence
Can you create a Trace / log-file of the bus communication

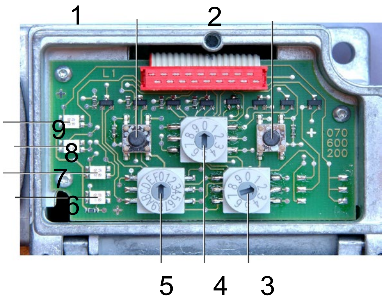
[7] The electronics of the bus Gateway can optionally be fed via separate connection points (X1: VDC2, GND2) or wired to the motor supply (X1: VDC1, GND1) via the jumpers J1 and J2 in the adapter.

8.2 Status indicates by LED

8.2.1 In general:



Note: LED 3 and RS 2 are not available in de single version



Button:
1 JOG- (J1) (counter-clockwise operation)
2 JOG+ (J2)
(clockwise
operation) Rotary
switch
3 field bus slave address (RS1) (low)
4 selection of the serial port (drive) (RS2)
5 field bus slave address
(RS3) (high) LED
6 Power On (LED 1)
7 DP-Fault (LED 2)
8 diagnostic indicator (LED 3)
9 RS485 Fault (LED 4)

8.2.2 Power-on LED 1

LED	Status	Indicates	Action
Off	Not Powered	No power to device	Check cabling Check the voltage of the power Check the max current of the power supply

8.2.3 DP-Error (Status) LED 2

LED	Status	Indicates	Action
Off	Not Powered Not On-Line	No power to device No DP Functionality established.	Check the power-supply Check cabling (especially the polarity)
Flashing green	Ready for cyclic communication	DPC initialized, but NO cyclic communication established. - Ready for configuration data from master.	Check cable connection Check configuration Start Master
Steady green	Online	Valid cyclic communication (data exchange) between master and slave.	No action is needed.
Flashing red	Communication error	DPC initialized. Faulty Configuration telegram. (SET_PRM, CHK_CFG).	Check cable connection Check Address setting Check Master

Steady red	Communication error	Aborted or faulty cyclic data exchange mode. (TimeOut, wrong data frame). Master starts reconfiguration.	Check cable connection Check configuration Start Master.

8.2.4 RS485 drive-error LED 4 and diagnostic Indicator LED 3

LED 3	LED 4	Status	Indicates	Action
X	Off	x	No connected unit is selected	
	Flashing green or flashing red		Self-test period (approx 1 sec. after power on)	No action is needed.
	Steady green	Communication of the selected drive is ok	Connections are established	No action is needed.
	Steady orange	error of the selected drive	Drive indicates an error, the Communication is ok	Error acknowledge, check the status of
	Steady red	Communication error of the selected drive	No response of the drive	Power on drive Check cabling Check the voltage of the power supply
Green	x	No error	Connections to all drives are error and warning free,	none
Flashing Green	x	warning of one or more drive	One of the connected drive reports a warning	check the status of the drive.
Flashing Green-Orange	x	Only if Jog-mode selected	The drives can be selected with rotary switch RS2 and moved in Jog Mode	none
Orange	x	Error of one or more drive	One of the connected drive reports an error	Error acknowledge, check the
Flashing Orange	x	Only if Jog-mode selected	Valid selection of one drive and activation by push-button to start moving	Select the correct drive
Flashing Red-Orange	x	Only if Jog-mode selected	Invalid action, e.g. press more than one button at the same time,...	

red	x	error	Hardware failure	Check cabling Check the voltage of the power supply Check the max current of the
Flashing red	X	Error of one or more drive connections	No connection to at least one connected drive.	With switch RS2 {1...8}, the "delinquent"
Flashing Green	Off	Error of one or more drive	No connected unit is selected	With switch RS2, the drive causing this error
	Green	Error of one or more drive	One of the not selected drive reports error or warning	With switch RS2, the drive causing this error
	Orange	Error of one or more drive	The selected drive reports error or warning	Error acknowledge, check the
Flashing Green-Orange	Green	Only if Jog-mode selected	Valid selection of one drive with the switches RS2 and the selected drive is reporting no error.	
Flashing Green-Orange	Orange	Only if Jog-mode selected Drive error	Valid selection of one drive with the switches RS2 and the selected drive is reporting an error.	
Flashing Green-Orange	red	Only if Jog-mode selected Drive error	The connection to the selected drive is faulty	
Flashing RED	Off	fault of one or more drive connections	no drive is selected	With switch RS2, the drive causing this error
	Green	Fault of one or more drive connections	One of the not selected drive reports error or warning	With switch RS2, the drive causing this error
	Orange	Fault of one or more drive connections	Connection with the selected drive is OK, but the drive reports an error	Error acknowledge, check the status of the drive.
	red	Fault of one or more drive	Connection with the selected drive	Check cabling Check the

		connections	is faulty.	the power supply Check the max current of

Note: check at first LED 3, if an error or warning is signed, search for the drive using RS2 and the state of LED 4

8.3 Error-Handling

8.3.1 Status indicates by “HIPERDRIVE warning / error information“

Refer to Parameter 953 and 1009

Bit	Status	Description	action
0	Motor current or maxim. Torque exceeded	Load is over limit, or the drive is defect. The drive stops	Check if the load is over limit or the drive Compare the torque of the application with the maximum torque of the
1	Operating voltage below limit	Voltage of the drive is below limit The drive stops	Check the voltage of the power Check the power supply for voltage Check the max current of the power supply. Check the cabling (regarding to length, diameter and
2	dynamic position deviation	the drive loses steps (only HRA08) The drive stops	Compare the torque of the application with the maximum torque of the
3	Overtemperature	Temperature of the drive-electronic is over limit The drive stops	Decrease the period of Check the ambient Improve the cooling.
4	Drive Blocked (Direction ccw)	The drive is blocked in direction ccw Drive stops, rotation in CW direction is possible	Check if the load is over limit or the drive is blocked Compare the torque of the application with the maximum torque of the
5	Drive Blocked (Direction cw)	The drive is blocked in direction cw Drive stops, rotation in CCW direction possible	Check if the load is over limit or the drive is blocked Compare the torque of the application with the maximum torque of the
8	Limit working area exceeded (CCW) - Min. Position	Position out of limit. Lost of the position is possible (roll-over) if the limit in parameter 1000 is set to the absolute	Check the readout position, compared with the mechanical Check the limits and change them.

		limit written in the datasheet	Set the position into the Drive the drive into inside the allowed range.
			Check your application, if you drive in Velocity mode with ignoring the limits (MC_MoveVelocity, Check your application, if the load is able do move the drive
9	Limit working area exceeded (CW) - Max. Position	Position out of limit. Lost of the position is possible (roll-over) if the limit in parameter 1001 is set to the absolute limit written in the datasheet	Check the readout position, compared with the mechanical Check the limits and change them. Set the position into the Drive the drive into inside the allowed Check your application, if you drive in Velocity mode with ignoring the limits (MC_MoveVelocity, Check your application, if the load is able do move the drive
10	Internal error: position accuracy	Internal error of the position sense	Send a acknowledge Do a Hardware reset If the error not disappears, change
11	Internal error: EEProm error - correctable	Internal error in data integrity of the drive. The data are automatically restored. The drive is still able to work	Send a acknowledge Do a Hardware reset If the error not disappears, change the drive as soon as
12	Internal error: EEProm error – not correctable	Internal error in data integrity of the drive. The data are lost. The drive is not working	Send a acknowledge Do a Hardware reset If the error not disappears, change
13	Internal error: wrong command sequence	wrong command sequence caused for example by truncated data transmission between gateway and drive	Send a acknowledge Check the cabling Check your application regarding to the sequence of the
14	Invalid Value of parameter or process data.	Try to access not accessible parameter via PROFIBUS DP	Check your application

15	Timeout for communication drive - gateway	Time Out, stated by adapter or Time Out, stated by Drive	Check the Power supply
			Check the Cabling
			Reduce the traffic between master and gateway on the

9 Dimension drawings

For additional specifications and dimension drawings, please visit our website at

www.halstrup-walcher.de/en/produkte/positioniertechnik/positioniersysteme/index.php