JENAER
ANTRIEBSTECHNIK

PROFI PROCESS FIELD BUS


# ECOVARIO® + ECOSTEP®

## Control via Profibus DP

*Published editions:*

| Edition | Comment |
|---|---|
| Nov 2009 | Initial English edition |
| April 2012 | Revision: Integration of 2-axis servo amplifier ECOVARIO 114 D |
| Dec. 2012 | Revision: Details added to „Preconditions“ |
| October 2016 | LOGO JAT new |
| April 2018 | Scope of functions FB100 modified |


Jenaer Antriebstechnik GmbH
Buchaer Straße 1
07745 Jena



The content of this document has been worked out and checked carefully. Nevertheless differences from the real state of the hard and software can never be fully excluded. Necessary corrections will be carried out in the next edition.

## Contents

# 1 About this manual

This manual describes the approach to functionally integrate the servo amplifers of the series ECOVARIO® or ECOSTEP® into a Profibus DP network (Master: Siemens SIMATIC PLC S7) by means of the JAT function blocks (FB Profibus DP). Prerequisite is that a physical connection exists between the servo amplifier and the Profibus master via the interface X22 (ECOVARIO® xxx xx-[G,F]x-xxx-xxx) or X2 (ECOSTEP®), respectively.

Further information:

- Hardware installation: Installation Manual ECOVARIO® or Installation Manual ECOSTEP®
- Documentation of the STEP7 software package (Siemens AG)
- Application Notes ECOVARIO® and ECOSTEP®
- Software commissioning: ECO Studio Operation Manual ECOVARIO®, ECOSTEP®, ECOMPACT®.

Technical requirements for personnel working with ECOVARIO® or ECOSTEP®:

**Transport:** Personnel trained in handling electrostatic sensitive devices
**Installation:** Electrotechnical specialists who are familiar with the safety guidelines for electrical and automation engineering.
**Commissioning:** Experts with extensive knowledge in the field of electrical engineering, automation technology and drive technology, in particular PLC programming.

Only qualified personnel with a broad knowledge of the fields of automation and electrical drives and controls, preferably knowledge in programming Profibus DP interfaces with the STEP7 software package, is allowed to carry out the Profibus integration.

**When dealing with the drive systems the manuals listed above and the safety precautions contained therein have to be observed.**

**The listed programs are example programs and show the basic procedure for using the function blocks and the data blocks. If the example programs are used in customer specific applications the user has to check whether all function and safety relevant conditions are fulfilled.**

# 2 About Profibus DP

PROFIBUS is a manufacturer independent open field bus standard, which is commonly used in many process automation environments. The international standard specification is listed under EN 50170 and describes the three mainly used standards: FMS, DP und PA.

PROFIBUS-DP is the standard optimized for fast data transfer, high efficiency and low implementation costs between an automation system and its decentralized intelligent actuators. Therefore it is suitable to control intelligent drives like ECOVARIO® and ECOSTEP® from a PLC.

PROFIBUS-DP works according to the principle of master-slave communication, together with several masters with an overlaying token-passing network. The master-slave communication is strictly cyclic whereby the failure of a master or slave can be detected immediately by means of a watchdog timer. Furthermore the diagnosis of a slave by the master is standardized and offers numerous possibilities to transmit failure and status information. The user in the network is identified by an adjustable address. In addition, each device class has its own identification number which is the same for all types.

# 3 Preconditions

**Required Compontents**

For controlling the JAT drives via Profibus DP the following components are required (in case of deviations please contact us):

- Servo amplifier ECOVARIO or ECOSTEP with Profibus DP interface
- Siemens SIMATIC PLC S7-300
- Program package STEP7, Version 5.3 to 5.5 (Siemens AG)
- Additional package S7-SCL (Siemens AG) (JAT drive specific blocks are written in in SCL)

  If the additional package is not available, the JAT specific blocks can be used as well, however, there is no possibility of adaptation.
- JAT package „FB Profibus DP“ (function blocks ECOSTEP/ECOVARIO) contains:

| | |
|---|---|
| ctrldjat.zip | Packed STEP7 library |
| jat3achs.zip | Packed example project |
| jatdgsd.zip | Packed GSD files |
| profibus_DP_control_120227.pdf | This description |

In order to establish the Profibus DP cabling between the components please refer to the ECOVARIO or ECOSTEP installation manual and the SIMATIC S7 documentation.

# 4 Procedure

After hardware commissioning has been finished the further procedure is:

1. Start the STEP7 program package
2. Open an existing project or create a new project, respectively
3. Link to the appropriate .GSD files. Two different .GSD files are provided:

| | |
|---|---|
| ECOVARIO.GSD | for all 1-axis servo amplifiers, i.e. ECOSTEP100, ECOSTEP200, ECOVARIO114, ECOVARIO214, ECOVARIO414 |
| ECODUAL.GSD | for 2-axis-servo amplifier ECOVARIO114 D. From the Profibus sight, the 2-axis servo amplifier behaves like any other 1-axis servo amplifier. The second axis is implemented via additional modules which have to be projected. |

4. Configure the Profibus. JAT drives can be found under ***Network Objects\ Other Field Devices\Drives\JAT***.

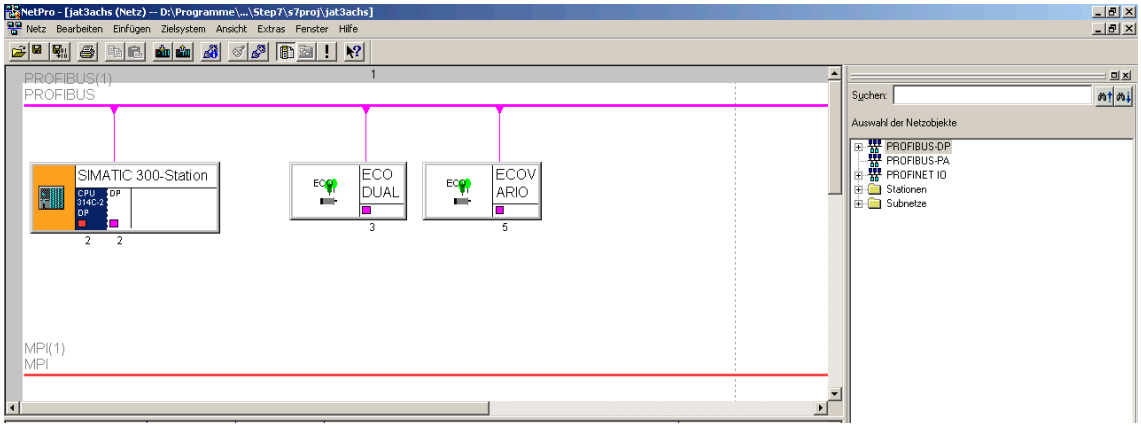



*Fig.1: Representation of the JAT drives*

## 4.1 Single-axis drive

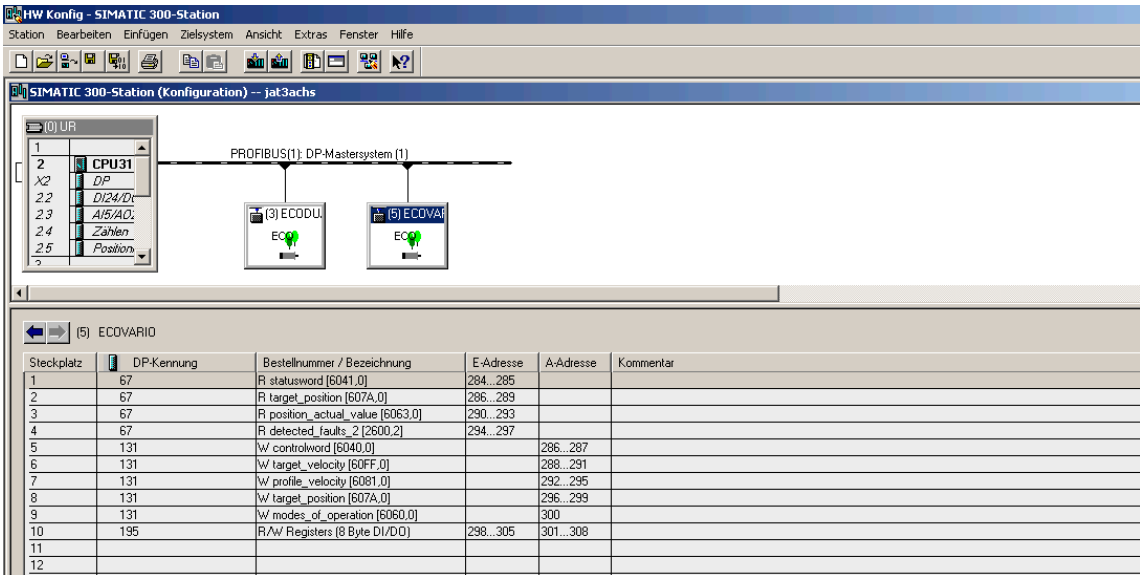
For a single-axis drive, at least the following modules should be configured:

**PROFIBUS MODULES TO BE CONFIGURED**

*Table 1: Profibus modules to be configured for single-axis drive*

| R/W | Module | Description |
|---|---|---|
| W | controlword | Control word (object 6040) |
| W | modes_of_operation | Mode of operation (object 6060) |
| W | target_position | Target position in increments (object 607A) |
| W | profile_velocity | Profile velocity in positioning mode in inc/64 s (object 6081) |
| W | target_velocity | Target velocity in velocity mode in ink/64 s (object 60FF) |
| R | statusword | Status word (object 6041) |
| R | detected_faults | Actual error status (object 2600) |
| R | position_actual_value | Actual position (object 6063) |
| R | target_position | Target position in increments (object 607A) |
| R/W | Registers (8 Byte DI/DO) | optional, see below. |

Fig. 2 shows an example of how the configuration for a 1-axis servo amplifier is displayed in the STEP7 user interface. The addresses are examples and can be different depending on the type of S7 PLC used!
The module „R/W Registers (8Byte DI/DO) is optional. It can be used for the random access to any object of the servo amplifier (read or write). For example, the actual current value (object 6078 sub index 00) of the drive can be provided for visualization or the max. admissible current (object 6073 sub index 00) can be limited. In this case the JAT function block FB110 can be used (cf. chapter 5).




| Steckplatz | DP-Kennung | Bestellnummer / Bezeichnung | E-Adresse | A-Adresse | Kommentar |
|---|---|---|---|---|---|
| 1 | 67 | R statusword [6041,0] | 284...285 | | |
| 2 | 67 | R target_position [607A,0] | 286...289 | | |
| 3 | 67 | R position_actual_value [6063,0] | 290...293 | | |
| 4 | 67 | R detected_faults_2 [2600,2] | 294...297 | | |
| 5 | 131 | W controlword [6040,0] | | 286...287 | |
| 6 | 131 | W target_velocity [60FF,0] | | 288...291 | |
| 7 | 131 | W profile_velocity [6081,0] | | 292...295 | |
| 8 | 131 | W target_position [607A,0] | | 296...299 | |
| 9 | 131 | W modes_of_operation [6060,0] | | 300 | |
| 10 | 195 | R/W Registers (8 Byte DI/DO) | 298...305 | 301...308 | |
| 11 | | | | | |
| 12 | | | | | |

*Fig. 2: Configuration of the Profibus modules for 1-axis servo amplifier*

## 4.2 2-axis drive

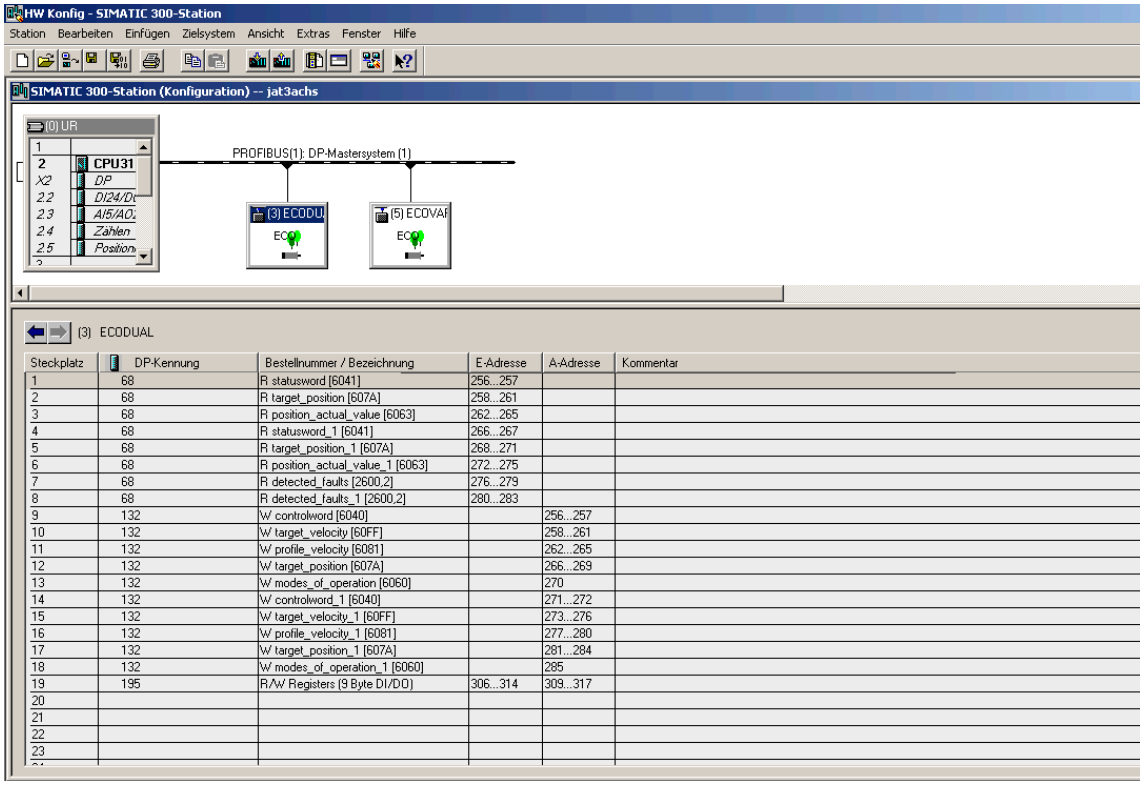
**Profibus Modules to be configured**

For a 2-axis drive (e.g. ECOVARIO 114D) the following modules can to be configured. The modules for axis 2 have the suffix „_1“ at the end of their name.

*Table 2: Profibus modules to be configured for 2-axis drive*

| R/W | Module | Description |
|---|---|---|
| W | controlword | Control word axis 1 (object 6040) |
| W | controlword_1 | Control word axis 2 (object 6040) |
| W | modes_of_operation | Mode of operation axis 1 (object 6060) |
| W | modes_of operation_1 | Mode of operation axis 2 (object 6060) |
| W | target_position | Target position axis 1 in increments (object 607A) |
| W | target_position_1 | Target position axis 2 in increments (object 607A) |
| W | profile_velocity | Profile velocity axis 1 in positioning mode in inc/64 s (object 6081) |
| W | profile_velocity_1 | Profile velocity axis 2 in positioning mode in inc/64 s (object 6081) |
| W | target_velocity | Target velocity axis 1 in velocity mode in inc/64 s (object 60FF) |
| W | target_velocity_1 | Target velocity axis 2 in velocty mode in inc/64 s (object 60FF) |
| R | statusword | Status word axis 1 (object 6041) |
| R | statusword_1 | Status word axis 2 (object 6041) |
| R | detected_faults | Actual error status axis 1 (object 2600) |
| R | detected faults_1 | Actual error status axis 2 (object 2600) |
| R | position_actual_value | Actual position axis 1 (object 6063) |
| R | position_actual_value_1 | Actual position axis 2 (object 6063) |
| R | target_position | Target position in increments axis 1 (object 607A) |
| R | target_position_1 | Target position in increments axis 2 (object 607A) |
| R/W | Registers (8 Byte DI/DO) | optional, see below. |

Fig. 3 shows an example of how the configuration for a 2-axis servo amplifier is displayed in the STEP7 user interface. The addresses are examples and can be different depending on the type of S7 PLC used!
The module „R/W Registers (8Byte DI/DO) is optional. It can be used for the random access to any object of the servo amplifier (read or write). For example, the actual current value (object 6078 sub index 00) of the drive can be provided for visualization or the max. admissible current (object 6073 sub index 00) can be limited. In this case the JAT function block FB104 can be used (cf. chapter 5).




| Steckplatz | DP-Kennung | Bestellnummer / Bezeichnung | E-Adresse | A-Adresse | Kommentar |
|---|---|---|---|---|---|
| 1 | 68 | R statusword [6041] | 256...257 | | |
| 2 | 68 | R target_position [607A] | 258...261 | | |
| 3 | 68 | R position_actual_value [6063] | 262...265 | | |
| 4 | 68 | R statusword_1 [6041] | 266...267 | | |
| 5 | 68 | R target_position_1 [607A] | 268...271 | | |
| 6 | 68 | R position_actual_value_1 [6063] | 272...275 | | |
| 7 | 68 | R detected_faults [2600,2] | 276...279 | | |
| 8 | 68 | R detected_faults_1 [2600,2] | 280...283 | | |
| 9 | 132 | W controlword [6040] | | 256...257 | |
| 10 | 132 | W target_velocity [60FF] | | 258...261 | |
| 11 | 132 | W profile_velocity [6081] | | 262...265 | |
| 12 | 132 | W target_position [607A] | | 266...269 | |
| 13 | 132 | W modes_of_operation [6060] | | 270 | |
| 14 | 132 | W controlword_1 [6040] | | 271...272 | |
| 15 | 132 | W target_velocity_1 [60FF] | | 273...276 | |
| 16 | 132 | W profile_velocity_1 [6081] | | 277...280 | |
| 17 | 132 | W target_position_1 [607A] | | 281...284 | |
| 18 | 132 | W modes_of_operation_1 [6060] | | 285 | |
| 19 | 195 | R/W Registers (9 Byte DI/DO) | 306...314 | 309...317 | |
| 20 | | | | | |
| 21 | | | | | |
| 22 | | | | | |
| 23 | | | | | |

*Fig. 3: Configuration of the Profibus modules for 2-axis servo amplifier*

# 5 Description of the application

By means of the function blocks/functions described in the following a Siemens SIMATIC PLC can control the drives of Jenaer Antriebstechnik GmbH via Profibus DP. In the STEP7 program the axes are programmed with the help of an axis DB. An example for such an axis DB is attached. The data transfer with the Profibus is managed by two functions (FC90 and FC92). The described components are collected in a STEP7 library.

Additionally, a complete STEP7 example project is provided in order to demonstrate the use of the function blocks/functions.

All function blocks/functions are written in SCL. The sources are part of the package, so they can be adapted, if necessary. The compiled FB/FC can also be used in conjunction with other STEP7 programming languages.

### Components of the Step7 Library

The STEP7 library comprises the following components incl. the respective sources in SCL:

*Table 3: Components of the STEP7 library*

| Designation | SCL Source | Description |
|---|---|---|
| FB100 | jatachse.scl | Axis function block. For each axis an instance of this FB with a respective instance DB and the input parameter axis DB (type BLOCK DB) is used. |
| FC90 | pb2axdb.scl | Reads the Profibus data and updates the axis DB data. One call of this function with the input parameter axis DB (type BLOCK DB) is used per axis. |
| FC92 | axdb2pb.scl | Reads the axis DB data and updates the Profibus data. One call of this function with the input parameter axis DB (type BLOCK DB) is used per axis. |
| FB110 | rwvalue.scl | Not required for the axis control itself! Allows to read and write any object on a 1-axis servo amplifier via the universal module. The provided example project shows the use. |
| FB104 | rwvalued.scl | Not required for the axis control itself! Allows to read and write any object on a 2-axis servo amplifier via the universal module. The provided example project shows the use. |
| DB101 | ax1db.scl | Source of axis data block. One block is required per axis (see above). The DB can also be generated otherwise. |

### Structure of the Step7 Program

The STEP7 program could be structured as follows (cf. example in chapter 6):

1. Write actual axis parameters from Profibus into the respective axis DB. To do so, e.g. the function FC90 is called according to the number of axes used and is only parameterized with the projected Profibus addresses.
2. Implementation of the required drive functionality by reading or writing the axis DB (application program).
3. Driving of the axes by calling one or more instance(s) of the function block FB100 with one instance DB and the respective axis DB as parameter.
4. Write actual axis DB data to the Profibus. To do so, the function FC92 is called according to the number of axes used and is only parameterized with the projected Profibus addresses.

### Function block FB100

For each axis a call of the function block „jatachse.scl“ (FB100) is required with a respective instance data block. This function block fulfills the following functions:

- Reset of the axis, i.e. carry out error reset, commutation and homing with the parameters stored in the servo amplifier
- Inching operation in positive and negative direction with the parameter inching speed (L_TARGET_VEL_DINT)
- Relative or absolute positioning to position (L_TARGET_POS_DINT) with the parameter profile velocity (L_PROFIL_VEL_DINT)
- Stopping the axis with STOP (generates error)
- De-energizing the drive with POWER_OFF (switching off the power stage)

Inching operation and positioning are interlocked against each other. The only parameter for this function block is the associated axis data block (see below).

If the integrated Profibus interface is used, the provided function calls „pb2axdb.scl“ (FC90) or „axdb2db.scl“ (FC92) can be used for the respective copy functions. The function calls only need to be parameterized with the peripheral addresses configured before.

**FC90** uses the following parameters:

**FUNCTION CALL FC90**

```
        // Read Profibus data to axis DB axis 1
        fc90 (achs_db    := db101 // IN: BLOCK_DB
,       status           := 256 // IN: ADR. STATUSWORD
,       target_pos_read  := 258 // IN: ADR. TARGET_POSITION READ
,       pos_actual_value := 262 // IN: ADR. POSITION_ACTUAL_VALUE
,       detected_faults  := 276 // IN: ADR. DETECTED_FAULTS
        ); // VOID
```

**FC92** uses the following parameters:

**FUNCTION CALL FC92**

```
        // Write Profibus data axis 1
        fc92 (achs_db    := db101 // IN: ACHS_DB Achse 1
,       control          := 256 // IN: ADR. CONTROLWORD
,       target_velocity  := 258 // IN: ADR. TARGET_VELOCITY
,       target_position  := 266 // IN: ADR. TARGET_POSITION WRITE
,       profile_velocity := 262 // IN: ADR. PROFILE_VELOCITY
,       mode             := 270 // IN: ADR. MODE_OF_OPERATION
        ); // VOID
```

**FB110** can be used for the read and write access to any object of the 1-axis servo amplifier:

**FUNCTION BLOCK FB110 (OPTIONAL) FOR 1-AXIS SERVO AMPL.**

```
// Universal module single axis
fb110.DB110(unimode := 298 // IN: ADR. UNIVERSAL MODULE INPUT (INT)
       ,unimoda  := 301  // IN: ADR. UNIVERSAL MODULE OUTPUT (INT)
       ,object   := rwobj // IN: CANOPEN ID OF THE TARGET OBJECT (WORD)
       ,objectsub := rwobjsubd // IN: CANOPEN SUB ID OF THE TARGET OBJECT (BYTE)
       ,wvalue   := rwvalue // IN: VALUE OF THE TARGET OBJECT (WRITE) (DWORD)
       ,operation := rwoperation // IN: OPERATION (FALSE=READ TRUE=WRITE)
       ,start    := rwstart // IN: START TRIGGER (CHANGE FROM FALSE->TRUE));
rvalue  := DB110.rvalue;  // OUT: RESPONSE VALUE OF THE TARGET OBJECT (READ) (DWORD)
rwrun   := DB110.run;     // OUT: FB IS RUNNING (BOOL)
rwdone  := DB110.done;    // OUT: FB OPERATION DONE
rwerror := DB110.error;   // OUT: FB OPERATION ERROR
```

**FB104** can be used for the read and write access to any object of the 2-axis servo amplifier:

**FUNCTION BLOCK FB104 (OPTIONAL) FOR 2-AXIS SERVO AMPL.**

```
// Universal module 2-axis
fb104.DB104(unimode := 306 // IN: ADR. UNIVERSAL MODULE INPUT (INT)
       ,unimoda  := 309 // IN: ADR. UNIVERSAL MODULE OUTPUT (INT)
       ,object   := rwobjd // IN: CANOPEN ID OF THE TARGET OBJECT (WORD)
       ,objectsub := rwobjsubd // IN: CANOPEN SUB ID OF THE TARGET OBJECT (BYTE)
       ,wvalue   := rwvalued// IN: VALUE OF THE TARGET OBJECT (WRITE) (DWORD)
       ,operation := rwoperationd// IN: OPERATION (FALSE=READ, TRUE=WRITE)
       ,axisnum  := anumd // IN: AXIS NUMBER (FALSE=AXIS_0, TRUE=AXIS_1)
       ,start    := rwstartd // IN: START TRIGGER (CHANGE FROM FALSE->TRUE));
rvalued := DB104.rvalue; // OUT: RESPONSE VALUE OF THE TARGET OBJECT (READ) (DWORD)
rwrund  := DB104.run; // OUT: FB IS RUNNING (BOOL)
rwdoned := DB104.done; // OUT: FB OPERATION DONE
rwerrord:= DB104.error; // OUT: FB OPERATION ERROR
```

In theory, there is an unlimited number of FB100 calls possible.
An arrangement of 12 axes has been tested

The axis data block **DB101** has the following structure:

```
-------------------------------------------------------------------------------------
//////////////////////////////////////////////////////////////////////////////////////
// Simatic S7                                                                       //
// Axis data block DB101 for axis 1                                                 //
//////////////////////////////////////////////////////////////////////////////////////

DATA_BLOCK DB101

AUTHOR:         BOCHER
FAMILY:         JAT
NAME:           AX_DB

STRUCT

   W_CONTROLWORD : WORD ;       // Control word (object 6040)
   W_TARGET_VEL : DWORD ;       // Target velocity in velocity mode in incr/64s (Objekt 60FF)
   W_TARGET_POS : DWORD ;       // Target position in incr. (object 607A)
   W_PROFIL_VEL : DWORD ;       // Target velocity in positioning mode in incr/64s (object 6081)
   W_MODES_OP : BYTE ;          // Mode of operation (object 6060)
   Fuellbyte_Reserve : BYTE ;
   R_STATUSWORD : WORD ;        // Status word (object 6041)
   R_TARGET_POS : DWORD ;       // target position read back (object 607A)
   R_POS_ACT_VALUE : DWORD ;    // actual position (object 6063)
   R_DETECT_FAULTS : DWORD ;    // actual error status (object 2600)
   R_PLOCK_STATUS : BYTE ;      // Positive limit switch status (object 2171)
   R_NLOCK_STATUS : BYTE ;      // Negative limit switch status (object 2172)
   start_reset : BOOL ;         // Commutate and reference axis
   start : BOOL ;               // Position axis
   pos_abs_rel : BOOL ;         // Switching positioning abs=1 ,relative=0
   start_pos_capt : BOOL ;      // Start Capture Mode
   stop : BOOL ;                // Stop axis
   ref : BOOL ;                 // Axis referenced and ready
   axis_error : BOOL ;          // Axis has errors
   tipp_p : BOOL ;              // Inch axis +
   tipp_m : BOOL ;              // Inch axis -
   run : BOOL ;                 // Axis running
   done : BOOL ;                // Positioning finished
   power_off : BOOL ;           // Switch off power stage
   abs_enc : BOOL ;             // Use absolute value encoder (no homing necessary)
   Fuellbit_2 : BOOL ;
   Fuellbit_3 : BOOL ;
   Fuellbit_4 : BOOL ;
   L_DETECT_FAULTS_OLD : DWORD ;// Error status after last movement
   L_TARGET_VEL_DINT : DINT;    // next velocity value in velocity mode in inc/64s
   L_TARGET_POS_DINT : DINT;    // next target position in inc
   L_PROFIL_VEL_DINT : DINT;    // next velocity in positioning mode in inc/64s

END_STRUCT
```

```
BEGIN

   W_CONTROLWORD := W#16#0;
   W_TARGET_VEL := DW#16#0;
   W_TARGET_POS := DW#16#0;
   W_PROFIL_VEL := DW#16#0;
   W_MODES_OP := B#16#0;
   Fuellbyte_Reserve := B#16#0;
   R_STATUSWORD := W#16#0;
   R_TARGET_POS := DW#16#0;
   R_POS_ACT_VALUE := DW#16#0;
   R_DETECT_FAULTS := DW#16#0;
   R_PLOCK_STATUS := B#16#0;
   R_NLOCK_STATUS := B#16#0;
   start_reset := FALSE;
   start := FALSE;
   pos_abs_rel := FALSE;
   start_pos_capt := FALSE;
   stop := FALSE;
   ref := FALSE;
   axis_error := FALSE;
   tipp_p := FALSE;
   tipp_m := FALSE;
   run := FALSE;
   done := FALSE;
   power_off := FALSE;
   abs_enc := FALSE;
   Fuellbit_2 := FALSE;
   Fuellbit_3 := FALSE;
   Fuellbit_4 := FALSE;
   L_DETECT_FAULTS_OLD := DW#16#0;
   L_TARGET_VEL_DINT := L#0;
   L_TARGET_POS_DINT := L#0;
   L_PROFIL_VEL_DINT := L#0;

END_DATA_BLOCK
```

---

The function block FB100 processes incremental values.
A conversion to physical values has to be carried out by the application.

The application interface consists of the following DB values:
Write values:

- L_TARGET_VEL_DINT : next velocity value in velocity mode in inc/64s
- L_TARGET_POS_DINT : DINT next target position in inc.
- L_PROFIL_VEL_DINT : DINT next velocity value in positioning mode in inc/64s
- start_reset : Switch on axis and carry out homing → TRUE
- start : Start positioning → TRUE
- stop : Stop axis during positioning → TRUE
- pos_abs_rel : for absolute positioning → TRUE, for relative positioning → FALSE
- tipp_p : Inching operation in positive direction → TRUE
- tipp_m : Inching operation in negative direction → TRUE
- abs_enc : Absolute value encoder → TRUE, Incremental encoder → FALSE
- power_off : Switch off axis → TRUE

Inching operation and positioning are interlocked against each other.

Read values:

- L_DETECT_FAULTS_OLD : DWORD error status after last movement
- ref : TRUE → Reference has been found
- done : TRUE → Positioning done
- axis_error : TRUE → Positioning error

In principle, also an other structure of the axis data block is possible. Therefore, only the respective addresses in the FB100 have to be adapted.

# 6 Program example

```
////////////////////////////////////////////////////////////////////////////////////////////
// Simatic S7-300                                                                         //
// Example program for an X-Y table with Z axis                                           //
// X- and Y axis: 23S31 each with SINCOS encoder and ECOVARIO 114D                        //
// Z axis: 23S21 mit 5P encoder (160000 Inkr./U) and ECOVARIO 114                         //
////////////////////////////////////////////////////////////////////////////////////////////
// X-Y table travels to 4 positions. At each position a punching movement of the          //
// Z axis is carried out.                                                                 //
// For demonstration of the possibilities of the universal module                         //
// the actual current values of the 3 axes are read out permanently                       //
// and one value is written to the universal variable 2100 sub index 01.                 //
////////////////////////////////////////////////////////////////////////////////////////////
// The example does not claim for completeness and only serves for demonstration          //
// purposes of the possibilities of the provided JAT function blocks/functions.           //
// Machine-specific symbols intentionally have been omitted.                              //
////////////////////////////////////////////////////////////////////////////////////////////


FUNCTION_BLOCK FB200

AUTHOR:       BOCHER
FAMILY:       JAT
NAME:         JATDEMO
VERSION:      '1.0'

CONST

    // State machine for axes
    ax000   :=  000; //
    ax100   :=  100; //
    ax102   :=  102; //
    ax104   :=  104; //
    ax106   :=  106; //
    ax108   :=  108; //
    ax110   :=  110; //
    ax120   :=  120; //
    ax130   :=  130; //
    ax140   :=  140; //
    ax150   :=  150; //
    ax160   :=  160; //
    ax170   :=  170; //
    ax180   :=  180; //
    ax190   :=  190; //
    ax200   :=  200; //
    ax210   :=  210; //
    ax220   :=  220; //
    ax230   :=  230; //
    ax240   :=  240; //
    ax250   :=  250; //
    ax260   :=  260; //
    ax270   :=  270; //
    ax280   :=  280; //
    ax290   :=  290; //
    ax300   :=  300; //
    ax310   :=  310; //
    ax320   :=  320; //
    ax330   :=  330; //
```

```
ax340   :=  340; //
ax350   :=  350; //
ax360   :=  360; //
ax370   :=  370; //
ax380   :=  380; //
ax390   :=  390; //
ax400   :=  400; //
ax410   :=  410; //
ax420   :=  420; //
ax430   :=  430; //
ax440   :=  440; //
ax450   :=  450; //
ax460   :=  460; //
ax470   :=  470; //
ax480   :=  480; //
ax490   :=  490; //
ax500   :=  500; //
ax510   :=  510; //
ax520   :=  520; //
ax530   :=  530; //
ax540   :=  540; //
ax550   :=  550; //
ax560   :=  560; //
ax570   :=  570; //
ax580   :=  580; //
ax590   :=  590; //
// State machine for communication via universal module
rd000   :=  000; //
rd010   :=  010; //
rd020   :=  020; //
rd030   :=  030; //
rd040   :=  040; //
rd050   :=  050; //
rd060   :=  060; //
rd070   :=  070; //
rd080   :=  080; //
rd090   :=  090; //
rd100   :=  100; //
rd110   :=  110; //
rd120   :=  120; //
rd130   :=  130; //
rd140   :=  140; //
rd150   :=  150; //
rd160   :=  160; //
rd170   :=  170; //
rd180   :=  180; //
rd190   :=  190; //

// Logic addresses in axis DB
a_reset             :=  032;    // Address RESET
a_start             :=  032;    // Address START
a_pos_abs_rel       :=  032;    // Address POS_ABS_REL
a_start_pos_capt    :=  032;    // Address START_POS_CAPT
a_stop              :=  032;    // Address STOP
a_tipp_p            :=  032;    // Address TIPP_P
a_tipp_m            :=  033;    // Address TIPP_M
a_power_off         :=  033;    // Address POWER_OFF
ba_reset            :=  000;    // Bit address RESET
ba_start            :=  001;    // Bit address START
ba_pos_abs_rel      :=  002;    // Bit address POS_ABS_REL
ba_start_pos_capt   :=  003;    // Bit address START_POS_CAPT
```

```
    ba_stop             :=  004;    // Bit address STOP
    ba_tipp_p           :=  007;    // Bit address TIPP_P
    ba_tipp_m           :=  000;    // Bit address TIPP_M
    ba_power_off        :=  003;    // Bit address POWER_OFF

    a_ref               :=  032;    // Address REF
    a_axis_error        :=  032;    // Address AXIS_ERROR
    a_run               :=  033;    // Address RUN
    a_done              :=  033;    // Address DONE
    a_absenc            :=  033;    // Address ABSOLUTENCODER
    ba_ref              :=  005;    // Bit address REF
    ba_axis_error       :=  006;    // Bit address AXIS_ERROR
    ba_run              :=  001;    // Bit address RUN
    ba_done             :=  002;    // Bit address DONE
    ba_absenc           :=  004;    // Bit address ABSOLUTENCODER

    // Data addresses in axis DB
    a_target_velocity   :=  038;    // Address velocity inching operation
    a_target_position   :=  042;    // Address target position
    a_profil_velocity   :=  046;    // Address velocity positioning
    a_home_offset       :=  058;    // Address Home Offset
    a_homing_method     :=  062;    // Address Homing Method
    a_max_current       :=  064;    // Address Max Current

    // Maschine values
    enc_aufl_x          :=     128000;  // Resolution of the SIN/COS incremental encoder of the X axis
    enc_aufl_y          :=     128000;  // Resolution of the SIN/COS-incremental encoder of the Y axis
    enc_aufl_z          :=     160000;  // Resolution of the incremental encoder of the Z axis
    geschw_x            :=      +50.0;  // Velocity of the X axis 50mm/s at screw lead 10mm
    geschw_y            :=      +50.0;  // Velocity of the Y axis 50mm/s at screw lead 10mm
    geschw_z            :=     +400.0;  // Velocity of the Z axis 400mm/s at screw lead 10mm
    geschw_hand         :=       +5.0;  // Velocity for manual mode 5mm/s for all axes
    xp0                 :=      +10.0;  // Starting position X axis
    xp1                 :=      +20.0;  // Position1 X axis
    xp2                 :=      +40.0;  // Position2 X axis
    xp3                 :=      +60.0;  // Position3 X axis
    xp4                 :=      +80.0;  // Position4 X axis
    yp0                 :=      +10.0;  // Starting position Y axis
    yp1                 :=      +20.0;  // Position1 Y axis
    yp2                 :=      +40.0;  // Position2 Y axis
    yp3                 :=      +60.0;  // Position3 Y axis
    yp4                 :=      +80.0;  // Position4 Y axis
    zp_aufw             :=       -8.0;  // Z axis up
    zp_abw              :=       +8.0;  // Z axis down

END_CONST

VAR

    zstax1              : INT   := 0;       // Step chain status for axis control
    zstr                : INT   := 0;       // Step chain status for read operation
    rwobjd              : WORD;             // Read/write object ECOVARIO 114D
    rwobjsubd           : BYTE;             // Sub index Read/write object ECOVARIO 114D
    rwstartd            : BOOL  := FALSE;   // Read/write operation START ECOVARIO 114D
    rwrund              : BOOL;             // Read/write operation running ECOVARIO 114D
    rwdoned             : BOOL;            // Read/write operation successfully done ECOVARIO 114D
    rwerrord            : BOOL;            // Read/write operation error ECOVARIO 114D
    rwvalued            : DWORD;            // Write value ECOVARIO 114D
    rvalued             : DWORD;            // Read value ECOVARIO 114D
    anumd               : BOOL;             // Axis number ECOVARIO 114D
    rwoperationd        : BOOL;             // R/W operation ECOVARIO 114D
```

```
    rwobj               : WORD;             // Read/Write object ECOVARIO 114
    rwobjsub            : BYTE;             // Subindex Read/Write object ECOVARIO 114
    rwstart             : BOOL  := FALSE;   // Read/Write operation START ECOVARIO 114
    rwrun               : BOOL;             // Read/Write operation running ECOVARIO 114
    rwdone             : BOOL;            // Read/Write operation successfully done ECOVARIO 114
    rwerror            : BOOL;            // Read/Write operation error ECOVARIO 114
    rwvalue             : DWORD;            // Write value ECOVARIO 114
    rvalue              : DWORD;            // Read value ECOVARIO 114
    rwoperation         : BOOL;             // R/W operation ECOVARIO 114
    cav1                : DWORD;            // actual current axis 1
    cav2                : DWORD;            // actual current axis 2
    cav3                : DWORD;            // actual current axis 3
    refall              : BOOL  := FALSE;   // all axes are referenced

END_VAR

BEGIN

// Read Profibus data to axis DB axis 1
fc90 (achs_db := db101 // IN: BLOCK_DB
 ,status := 256 // IN: INT
 ,target_pos_read := 258 // IN: INT
 ,pos_actual_value := 262 // IN: INT
 ,detected_faults := 276 // IN: INT
 ); // VOID

// Read Profibus data to axis DB axis 2
fc90 (achs_db := db102 // IN: BLOCK_DB
 ,status := 266 // IN: INT
 ,target_pos_read := 268 // IN: INT
 ,pos_actual_value := 272 // IN: INT
 ,detected_faults := 280 // IN: INT
 ); // VOID

// Read Profibus data to axis DB axis 3
fc90 (achs_db := db103 // IN: BLOCK_DB
 ,status := 284 // IN: INT
 ,target_pos_read := 286 // IN: INT
 ,pos_actual_value := 290 // IN: INT
 ,detected_faults := 294 // IN: INT
 ); // VOID

// Restart (OB100)
IF (anl_merker = TRUE) THEN
    refall          := FALSE;
    zstax1          := ax000;
    zstr            := rd000;
    hand_auto       := FALSE;
    anl_merker      := FALSE;
END_IF;

// Reference all 3 axes
IF (ref_taste = TRUE) THEN
    zstax1          := ax100;
    zstr            := rd000;
    ref_taste       := FALSE;
END_IF;

// Switch off power
IF (power_off = TRUE) THEN
    db101.dx[a_power_off,ba_power_off]  := TRUE;
```

```
    db102.dx[a_power_off,ba_power_off]  := TRUE;
    db103.dx[a_power_off,ba_power_off]  := TRUE;
ELSE
    db101.dx[a_power_off,ba_power_off]  := FALSE;
    db102.dx[a_power_off,ba_power_off]  := FALSE;
    db103.dx[a_power_off,ba_power_off]  := FALSE;
END_IF;

// HAND/AUTOMATIC operation
IF (refall = TRUE) THEN
    IF (hand_auto = TRUE) THEN
        // AUTOMATIK
        db101.dx[a_tipp_m,ba_tipp_m]    := FALSE;
        db101.dx[a_tipp_p,ba_tipp_p]    := FALSE;
        db102.dx[a_tipp_m,ba_tipp_m]    := FALSE;
        db102.dx[a_tipp_p,ba_tipp_p]    := FALSE;
        db103.dx[a_tipp_m,ba_tipp_m]    := FALSE;
        db103.dx[a_tipp_p,ba_tipp_p]    := FALSE;
        IF (start_taste = TRUE) THEN
            start_taste                 := FALSE;
            zstax1                      := ax120;
        END_IF;
    ELSE
        // HAND
        zstax1                          := ax500;
    END_IF;
END_IF;

// State machine axes
CASE zstax1 OF
    ax000   :   // wait
        zstax1                                  := ax000;
    ax100   :   // Switch on axes and search for reference, prepare positioning data
        db101.dx[a_ref,ba_ref]                  := FALSE;
        db101.dx[a_reset,ba_reset]              := TRUE;
        db101.dd[a_profil_velocity]  := DINT_TO_DWORD(REAL_TO_DINT(geschw_x * enc_aufl_x * 6.4));
        db101.dx[a_pos_abs_rel,ba_pos_abs_rel]  := TRUE;    // absolute positioning
        db102.dx[a_ref,ba_ref]                  := FALSE;
        db102.dx[a_reset,ba_reset]              := TRUE;
        db102.dd[a_profil_velocity]  := DINT_TO_DWORD(REAL_TO_DINT(geschw_y * enc_aufl_y * 6.4));
        db102.dx[a_pos_abs_rel,ba_pos_abs_rel]  := TRUE;    // absolute positioning
        db103.dx[a_ref,ba_ref]                  := FALSE;
        db103.dx[a_reset,ba_reset]              := TRUE;
        db103.dd[a_profil_velocity]  := DINT_TO_DWORD(REAL_TO_DINT(geschw_z * enc_aufl_z * 6.4));
        db103.dx[a_pos_abs_rel,ba_pos_abs_rel]  := FALSE;   // relative positioning
        zstax1                                  := ax102;
    ax102   :   // Reference found -> X/Y table to starting position
        IF ((db101.dx[a_ref,ba_ref] = TRUE)
        AND (db101.dx[a_ref,ba_ref] = TRUE)
        AND (db101.dx[a_ref,ba_ref] = TRUE)) THEN
            db101.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(xp0 * enc_aufl_x));
            db102.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(yp0 * enc_aufl_y));
            refall                              := TRUE;
            zstax1                              := ax104;
        END_IF;
    ax104   :   // X/Y table to Position 1
        db101.dx[a_start,ba_start]              := TRUE;
        db102.dx[a_start,ba_start]              := TRUE;
        zstax1                                  := ax106;
    ax106   :   // X/Y table Position 1 reached? -> wait
        IF ((db101.dx[a_done,ba_done] = TRUE)
```

```
        AND (db102.dx[a_done,ba_done] = TRUE)) THEN
        zstax1                              := ax110;
    END_IF;
ax110   :   // wait for START
    zstax1                                  := ax110;
ax120   :   // X/Y table set Position 1
    db101.dd[a_target_position]             := DINT_TO_DWORD(REAL_TO_DINT(xp1 * enc_aufl_x));
    db102.dd[a_target_position]             := DINT_TO_DWORD(REAL_TO_DINT(yp1 * enc_aufl_y));
    zstax1                                  := ax130;
ax130   :   // X/Y table to Position 1
    db101.dx[a_start,ba_start]              := TRUE;
    db102.dx[a_start,ba_start]              := TRUE;
    zstax1                                  := ax140;
ax140   :   // X/Y table Position 1 reached? -> Set Z axis down
    IF ((db101.dx[a_done,ba_done] = TRUE)
        AND (db102.dx[a_done,ba_done] = TRUE)) THEN
        db103.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(zp_abw * enc_aufl_z));
        zstax1                              := ax150;
    END_IF;
ax150   :   // Z axis down
    db103.dx[a_start,ba_start]              := TRUE;
    zstax1                                  := ax160;
ax160   :   // Z axis lower position reached? -> Set Z axis up
    IF (db103.dx[a_done,ba_done] = TRUE) THEN
        db103.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(zp_aufw * enc_aufl_z));
        zstax1                              := ax170;
    END_IF;
ax170   :   // Z axis up
    db103.dx[a_start,ba_start]              := TRUE;
    zstax1                                  := ax180;
ax180   :   // Z axis upper Position reached? -> Set next table position
    IF (db103.dx[a_done,ba_done] = TRUE) THEN
        db101.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(xp2 * enc_aufl_x));
        db102.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(yp2 * enc_aufl_y));
        zstax1                              := ax190;
    END_IF;
ax190   :   // X/Y table to Position 2
    db101.dx[a_start,ba_start]              := TRUE;
    db102.dx[a_start,ba_start]              := TRUE;
    zstax1                                  := ax200;
ax200   :   // X/Y table Position 2 reached? -> Set Z axis down
    IF ((db101.dx[a_done,ba_done] = TRUE)
        AND (db102.dx[a_done,ba_done] = TRUE)) THEN
        db103.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(zp_abw * enc_aufl_z));
        zstax1                              := ax210;
    END_IF;
ax210   :   // Z axis down
    db103.dx[a_start,ba_start]              := TRUE;
    zstax1                                  := ax220;
ax220   :   // Z axis lower Position reached? -> Set Z axis up
    IF (db103.dx[a_done,ba_done] = TRUE) THEN
        db103.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(zp_aufw * enc_aufl_z));
        zstax1                              := ax230;
    END_IF;
ax230   :   // Z axis up
    db103.dx[a_start,ba_start]              := TRUE;
    zstax1                                  := ax240;
ax240   :   // Z axis upper Position reached? -> Set next table position
    IF (db103.dx[a_done,ba_done] = TRUE) THEN
        db101.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(xp3 * enc_aufl_x));
        db102.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(yp3 * enc_aufl_y));
```

```
        zstax1                              := ax250;
    END_IF;
ax250   :   // X/Y table to Position 3
    db101.dx[a_start,ba_start]              := TRUE;
    db102.dx[a_start,ba_start]              := TRUE;
    zstax1                                  := ax260;
ax260   :   // X/Y table Position 3 reached? -> Set Z axis down
    IF ((db101.dx[a_done,ba_done] = TRUE)
        AND (db102.dx[a_done,ba_done] = TRUE)) THEN
        db103.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(zp_abw * enc_aufl_z));
        zstax1                              := ax270;
    END_IF;
ax270   :   // Z axis down
    db103.dx[a_start,ba_start]              := TRUE;
    zstax1                                  := ax280;
ax280   :   // Z axis lower Position reached? -> Set Z axis up
    IF (db103.dx[a_done,ba_done] = TRUE) THEN
        db103.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(zp_aufw * enc_aufl_z));
        zstax1                              := ax290;
    END_IF;
ax290   :   // Z axis up
    db103.dx[a_start,ba_start]              := TRUE;
    zstax1                                  := ax300;
ax300   :   // Z axis upper Position reached? -> Set next table position
    IF (db103.dx[a_done,ba_done] = TRUE) THEN
        db101.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(xp4 * enc_aufl_x));
        db102.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(yp4 * enc_aufl_y));
        zstax1                              := ax310;
    END_IF;
ax310   :   // X/Y-Tisch to Position 4
    db101.dx[a_start,ba_start]              := TRUE;
    db102.dx[a_start,ba_start]              := TRUE;
    zstax1                                  := ax320;
ax320   :   // X/Y table Position 4 reached? -> Set Z axis down
    IF ((db101.dx[a_done,ba_done] = TRUE)
        AND (db102.dx[a_done,ba_done] = TRUE)) THEN
        db103.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(zp_abw * enc_aufl_z));
        zstax1                              := ax330;
    END_IF;
ax330   :   // Z axis down
    db103.dx[a_start,ba_start]              := TRUE;
    zstax1                                  := ax340;
ax340   :   // Z axis lower Position reached? -> Set Z axis up
    IF (db103.dx[a_done,ba_done] = TRUE) THEN
        db103.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(zp_aufw * enc_aufl_z));
        zstax1                              := ax350;
    END_IF;
ax350   :   // Z axis up
    db103.dx[a_start,ba_start]              := TRUE;
    zstax1                                  := ax360;
ax360   :   // Z axis upper Position reached? -> Table to starting position
    IF (db103.dx[a_done,ba_done] = TRUE) THEN
        db101.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(xp0 * enc_aufl_x));
        db102.dd[a_target_position]         := DINT_TO_DWORD(REAL_TO_DINT(yp0 * enc_aufl_y));
        zstax1                              := ax370;
    END_IF;
ax370   :   // X/Y table to starting position
    db101.dx[a_start,ba_start]              := TRUE;
    db102.dx[a_start,ba_start]              := TRUE;
    zstax1                                  := ax380;
ax380   :   // X/Y table starting position reached? -> done
```

```
        IF ((db101.dx[a_done,ba_done] = TRUE)
            AND (db102.dx[a_done,ba_done] = TRUE)) THEN
            zstax1                      := ax110;
        END_IF;
    ax500   :   // Manual Mode
        db101.dd[a_target_velocity]     := DINT_TO_DWORD(REAL_TO_DINT(geschw_hand * enc_aufl_x * 64.0));
        db102.dd[a_target_velocity]     := DINT_TO_DWORD(REAL_TO_DINT(geschw_hand * enc_aufl_y * 64.0));
        db103.dd[a_target_velocity]     := DINT_TO_DWORD(REAL_TO_DINT(geschw_hand * enc_aufl_z * 64.0));
        db101.dx[a_tipp_m,ba_tipp_m]    := handx_n;
        db101.dx[a_tipp_p,ba_tipp_p]    := handx_p;
        db102.dx[a_tipp_m,ba_tipp_m]    := handy_n;
        db102.dx[a_tipp_p,ba_tipp_p]    := handy_p;
        db103.dx[a_tipp_m,ba_tipp_m]    := handz_n;
        db103.dx[a_tipp_p,ba_tipp_p]    := handz_p;
        zstax1                          := ax500;
    ELSE        // should not occur!
        zstax1                          := ax000;
END_CASE;

// Read/write axis parameters
CASE zstr OF
    rd000   :   // wait
        IF ((db101.dx[a_ref,ba_ref]) AND (db102.dx[a_ref,ba_ref]) AND (db103.dx[a_ref,ba_ref]))
THEN
            // reference found for all axes -> display current values
            zstr            := rd010;
        END_IF;
    rd010   :   // read axis ID3
        rwobjd              := W#16#6078;   // Object ID
        rwobjsubd           := B#16#00;     // Object SubID
        rwoperationd        := FALSE;       // read = FALSE; write = TRUE
        anumd               := FALSE;       // First axis of ECOVARIO 114 D
        rwstartd            := TRUE;
        rwdoned             := FALSE;
        rwerrord            := FALSE;
        zstr                := rd020;
    rd020   :   // read operation done?
        IF (rwdoned = TRUE) THEN
            cav1            := rvalued;
            rwstartd        := FALSE;
            zstr            := rd030;
        ELSIF (rwerrord= TRUE) THEN
            rwstartd        := FALSE;
            zstr            := rd000;
        END_IF;
    rd030   :   // read axis ID4
        rwobjd              := W#16#6078;   // Object ID
        rwobjsubd           := B#16#00;     // Object SubID
        rwoperationd        := FALSE;       // read = FALSE; write = TRUE
        anumd               := TRUE;        // Second axis of ECOVARIO 114 D
        rwstartd            := TRUE;
        rwdoned             := FALSE;
        rwerrord            := FALSE;
        zstr                := rd040;
    rd040   :   // read operation done?
        IF (rwdoned = TRUE) THEN
            cav2            := rvalued;
            rwstartd        := FALSE;
            zstr            := rd050;
        ELSIF (rwerrord= TRUE) THEN
            rwstartd        := FALSE;
```

```
        zstr            := rd000;
    END_IF;
rd050   :   // read axis ID5
    rwobj               := W#16#6078;   // Object ID
    rwobjsub            := B#16#00;     // Object SubID
    rwoperation         := FALSE;       // read = FALSE; write = TRUE
    rwstart             := TRUE;
    rwdone              := FALSE;
    rwerror             := FALSE;
    zstr                := rd060;
rd060   :   // read operation done?
    IF (rwdone = TRUE) THEN
        cav3            := rvalue;
        rwstart         := FALSE;
        zstr            := rd070;
    ELSIF (rwerrord= TRUE) THEN
        rwstart         := FALSE;
        zstr            := rd000;
    END_IF;
rd070   :   // write axis ID3
    rwobjd              := W#16#2100;   // Object ID
    rwobjsubd           := B#16#01;     // Object SubID
    rwvalued            := DW#16#0F0F0F0F;  // value to be written
    rwoperationd        := TRUE;        // read = FALSE; write = TRUE
    anumd               := FALSE;       // to first axis of ECOVARIO 114 D
    rwstartd            := TRUE;
    rwdoned             := FALSE;
    rwerrord            := FALSE;
    zstr                := rd080;
rd080   :   // write operation done?
    IF (rwdoned = TRUE) THEN
        rwstartd        := FALSE;
        zstr            := rd090;
    ELSIF (rwerrord = TRUE) THEN
        rwstartd        := FALSE;
        zstr            := rd000;
    END_IF;
rd090   :   // write axis ID4
    rwobjd              := W#16#2100;   // Object ID
    rwobjsubd           := B#16#01;     // Object SubID
    rwvalued            := DW#16#23232323;  // value to be written
    rwoperationd        := TRUE;        // write
    anumd               := TRUE;        // to second axis of ECOVARIO 114 D
    rwstartd            := TRUE;
    rwdoned             := FALSE;
    rwerrord            := FALSE;
    zstr                := rd100;
rd100   :   // write operation done?
    IF (rwdoned = TRUE) THEN
        rwstartd        := FALSE;
        zstr            := rd110;
    ELSIF (rwerrord= TRUE) THEN
        rwstartd        := FALSE;
        zstr            := rd000;
    END_IF;
rd110   :   // write axis ID5
    rwobj               := W#16#2100;   // Object ID
    rwobjsub            := B#16#01;     // Object SubID
    rwvalue             := DW#16#45454545;  // value to be written
    rwoperation         := TRUE;        // write
    rwstart             := TRUE;
```

```
        rwdone              := FALSE;
        rwerror             := FALSE;
        zstr                := rd120;
    rd120   :   // write operation done?
        IF (rwdone = TRUE) THEN
            rwstart         := FALSE;
            zstr            := rd010;
        ELSIF (rwerror = TRUE) THEN
            rwstart         := FALSE;
            zstr            := rd000;
        END_IF;
    ELSE        // should not occur!
        zstr                := rd000;
END_CASE;

// FB call axis 1
fb100.db201(achs_db := db101); // IN: ACHS_DB axis 1
// FB call axis 2
fb100.db202(achs_db := db102); // IN: ACHS_DB axis 2
// FB call axis 3
fb100.db203(achs_db := db103); // IN: ACHS_DB axis 3

// Write Profibus data axis 1
fc92 (achs_db := db101 // IN: ACHS_DB axis 1
 ,control := 256 // IN: ADR. CONTROLWORD
 ,target_velocity := 258 // IN: ADR. TARGET_VELOCITY
 ,target_position := 266 // IN: ADR. TARGET_POSITION
 ,profile_velocity := 262 // IN: ADR. PROFILE_VELOCITY
 ,mode := 270 // IN: ADR. MODE_OF_OPERATION
 ); // VOID

// Write Profibus data axis 2
fc92 (achs_db := db102 // IN: ACHS_DB axis 1
 ,control := 271 // IN: ADR. CONTROLWORD
 ,target_velocity := 273 // IN: ADR. TARGET_VELOCITY
 ,target_position := 281 // IN: ADR. TARGET_POSITION
 ,profile_velocity := 277 // IN: ADR. PROFILE_VELOCITY
 ,mode := 285 // IN: ADR. MODE_OF_OPERATION
 ); // VOID

// Write Profibus data axis 3
fc92 (achs_db := db103 // IN: ACHS_DB axis 1
 ,control := 286 // IN: ADR. CONTROLWORD
 ,target_velocity := 288 // IN: ADR. TARGET_VELOCITY
 ,target_position := 296 // IN: ADR. TARGET_POSITION
 ,profile_velocity := 292 // IN: ADR. PROFILE_VELOCITY
 ,mode := 300 // IN: ADR. MODE_OF_OPERATION
 ); // VOID

// Universal module axes 1 and 2
fb104.DB104(unimode := 306 // IN: INT
       ,unimoda := 309 // IN: INT
       ,object := rwobjd // IN: WORD
       ,objectsub := rwobjsubd // IN: BYTE
       ,wvalue := rwvalued// IN: DWORD
       ,operation := rwoperationd// IN: BOOL
       ,axisnum := anumd // IN: BOOL
       ,start := rwstartd // IN: BOOL
       );
rvalued := DB104.rvalue; // OUT: DWORD
rwrund  := DB104.run; // OUT: BOOL
```

```
rwdoned := DB104.done; // OUT: BOOL
rwerrord:= DB104.error; // OUT: BOOL

// Universal module axis 3
fb110.DB110(unimode := 298 // IN: INT
       ,unimoda := 301 // IN: INT
       ,object := rwobj // IN: WORD
       ,objectsub := rwobjsubd // IN: BYTE
       ,wvalue := rwvalue // IN: DWORD
       ,operation := rwoperation // IN: BOOL
       ,start := rwstart // IN: BOOL
       );
rvalue  := DB110.rvalue; // OUT: DWORD
rwrun   := DB110.run; // OUT: BOOL
rwdone  := DB110.done; // OUT: BOOL
rwerror := DB110.error; // OUT: BOOL

END_FUNCTION_BLOCK
```