SIEMENS

# SINUMERIK

# SINUMERIK
# 840D sl/840Di sl/840D/840Di/810D
# Synchronized actions

Function Manual


Foreword

Brief description 1

Detailed description 2

Boundary conditions 3

Signal Descriptions 4

Examples 5

Data lists 6

Appendix A


Valid for

*Control*
SINUMERIK 840D sl/840DE sl
SINUMERIK 840Di sl/840DiE sl
SINUMERIK 840D powerline/840DE powerline
SINUMERIK 840Di powerline/840DiE powerline
SINUMERIK 810D powerline/810DE powerline

| *Software* | *Version* |
|---|---|
| NCU system software for 840D sl/840DE sl | 1.4 |
| NCU system software for 840D sl/DiE sl | 1.0 |
| NCU system software for 840D/840DE | 7.4 |
| NCU system software for 840Di/840DiE | 3.3 |
| NCU system software for 810D/810DE | 7.4 |


11/2006
6FC5397-5BP10-2BA0

## Safety Guidelines

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

**Danger**

indicates that death or severe personal injury **will** result if proper precautions are not taken.

**Warning**

indicates that death or severe personal injury **may** result if proper precautions are not taken.

**Caution**

with a safety alert symbol, indicates that minor personal injury can result if proper precautions are not taken.

**Caution**

without a safety alert symbol, indicates that property damage can result if proper precautions are not taken.

**Notice**

indicates that an unintended result or situation can occur if the corresponding information is not taken into account.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

## Qualified Personnel

The device/system may only be set up and used in conjunction with this documentation. Commissioning and operation of a device/system may only be performed by **qualified personnel**. Within the context of the safety notes in this documentation qualified persons are defined as persons who are authorized to commission, ground and label devices, systems and circuits in accordance with established safety practices and standards.

## Prescribed Usage

Note the following:

**Warning**

This device may only be used for the applications described in the catalog or the technical description and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens. Correct, reliable operation of the product requires proper transport, storage, positioning and assembly as well as careful operation and maintenance.

## Trademarks

All names identified by ® are registered trademarks of the Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

## Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens AG
Automation and Drives
Postfach 48 48
90437 NÜRNBERG
GERMANY

Order No.: 6FC5397-5BP10-2BA0
℗ 11/2006

Copyright © Siemens AG 2006.
Technical data subject to change

# Foreword

## SINUMERIK® Documentation

The SINUMERIK documentation is organized in 3 parts:

- General documentation
- User documentation
- Manufacturer/service documentation

A monthly updated publications overview with respective available languages can be found in the Internet under:

http://www.siemens.com/motioncontrol

Select the menu items "Support" → "Technical Documentation" → "Overview of Publications".

The Internet version of DOConCD (DOConWEB) is available under:

http://www.automation.siemens.com/doconweb

Information about training courses and FAQs (Frequently Asked Questions) can be found in internet under::

http://www.siemens.com/motioncontrol under menu option "Support"

## Target group

This publication is intended for:

- Project engineers
- Technologists (from machine manufacturers)
- System startup engineers (Systems/Machines)
- Programmers

## Benefits

The function manual describes the functions so that the target group knows them and can select them. It provides the target group with the information required to implement the functions.

## Standard version

This documentation only describes the functionality of the standard version. Extensions or changes made by the machine tool manufacturer are documented by the machine tool manufacturer.

Other functions not described in this documentation might be executable in the control. This does not, however, represent an obligation to supply such functions with a new control or when servicing.

Further, for the sake of simplicity, this documentation does not contain all detailed information about all types of the product and cannot cover every conceivable case of installation, operation or maintenance.

## Technical Support

If you have any technical questions, please contact our hotline:

| | Europe / Africa | Asia / Australia | America |
|---|---|---|---|
| **Phone** | +49 180 5050 222 | +86 1064 719 990 | +1 423 262 2522 |
| **Fax** | +49 180 5050 223 | +86 1064 747 474 | +1 423 262 2289 |
| **Internet** | http://www.siemens.com/automation/support-request | | |
| **E-Mail** | mailto:adsupport@siemens.com | | |

**Note**

Country specific telephone numbers for technical support are provided under the following Internet address:

http://www.siemens.com/automation/service&support

## Questions about the manual

If you have any queries (suggestions, corrections) in relation to this documentation, please send a fax or e-mail to the following address:

Fax: +49 (0) 9131 / 98 - 63315
Email: mailto:docu.motioncontrol@siemens.com

A fax form is available at the end of this document.

## SINUMERIK Internet address

http://www.siemens.com/sinumerik

## EC declaration of conformity

The EC Declaration of Conformity for the EMC Directive can be found/obtained

- in the internet:
  http://www.ad.siemens.de/csinfo
  under product/order no. 15257461
- with the relevant branch office of the A&D MC group of Siemens AG.

# Table of contents


**Foreword** ..... **3**

**1 Brief description** ..... **11**

**2 Detailed description** ..... **13**

2.1 Components of synchronized actions ..... 13
2.1.1 Definition of motion-synchronous actions ..... 20
2.1.2 Execution of synchronized actions ..... 20
2.1.3 List of possible actions ..... 20

2.2 Real-time evaluations and calculations ..... 22

2.3 Special real-time variables for synchronized actions ..... 28
2.3.1 Marker/counter variables ..... 29
2.3.2 Timers ..... 30
2.3.3 Synchronized action parameters ..... 31
2.3.4 R parameters ..... 32
2.3.5 Machine and setting data ..... 32
2.3.6 FIFO variables (circulating memory) ..... 34
2.3.7 System variables saved in SRAM (SW 6.3 and later) ..... 36
2.3.8 Determining the path tangent in synchronized actions ..... 37
2.3.9 Determining the current override ..... 38
2.3.10 Capacity evaluation using time requirement for synchronized actions ..... 39
2.3.11 List of system variables relevant to synchronized actions ..... 41

2.4 Actions in synchronized actions ..... 42
2.4.1 Output of M, S and H auxiliary functions to the PLC ..... 44
2.4.2 Setting (writing) and reading of real-time variables ..... 47
2.4.3 Changing of SW cam positions and times (setting data) ..... 48
2.4.4 FCTDEF ..... 49
2.4.5 Polynomial evaluation SYNFCT ..... 51
2.4.6 Overlaid movements $AA_OFF settable (SW 6 and later) ..... 56
2.4.7 Online tool offset FTOC ..... 59
2.4.8 Online tool length offset $AA_TOFF[Index] ..... 61
2.4.9 RDISABLE ..... 66
2.4.10 STOPREOF ..... 66
2.4.11 DELDTG ..... 67
2.4.12 Disabling a programmed axis motion ..... 68
2.4.13 Starting command axes ..... 69
2.4.14 Axial feedrate from synchronized actions ..... 73
2.4.15 Starting/Stopping axes from synchronized actions ..... 74
2.4.16 Axis replacement from synchronized actions ..... 75
2.4.17 Spindle motions from synchronized actions ..... 81
2.4.18 Setting actual values from synchronized actions ..... 85
2.4.19 Activating/deactivating coupled motions and couplings ..... 86
2.4.20 Measurements from synchronized actions ..... 93
2.4.21 Setting and deleting wait markers for channel synchronization ..... 97
2.4.22 Set alarm/error reactions ..... 98
2.4.23 Evaluating data for machine maintenance ..... 99

2.5 Call of Technology Cycles ... 101
2.5.1 Coordination of synchronized actions, technology cycles, part program (and PLC) ... 104
2.6 Control and protection of synchronized actions ... 106
2.6.1 Control via PLC ... 106
2.6.2 Protected synchronized actions ... 108
2.7 Control system response for synchronized actions in specific operational states ... 111
2.7.1 Power On ... 111
2.7.2 RESET ... 111
2.7.3 NC STOP ... 112
2.7.4 Mode change ... 113
2.7.5 End of program ... 113
2.7.6 Response of active synchronized actions to end of program and change in operating mode ... 114
2.7.7 Block search ... 114
2.7.8 Program interruption by ASUB ... 115
2.7.9 REPOS ... 115
2.7.10 Response to alarms ... 115
2.8 Configuration ... 116
2.8.1 Configurability ... 116
2.9 Diagnostics (only with HMI Advanced) ... 119
2.9.1 Displaying status of synchronized actions ... 120
2.9.2 Displaying main run variables ... 121
2.9.3 Logging main run variables ... 121
**3 Boundary conditions ... 125**
**4 Signal Descriptions ... 129**
**5 Examples ... 131**
5.1 Examples of conditions in synchronized actions ... 131
5.2 Reading and writing of SD/MD from synchronized actions ... 132
5.3 Examples of adaptive control ... 134
5.3.1 Clearance control with variable upper limit ... 135
5.3.2 Feedrate control ... 136
5.3.3 Control velocity as a function of normalized path ... 137
5.4 Monitoring a safety clearance between two axes ... 138
5.5 Store execution times in R parameters ... 139
5.6 "Centering" with continuous measurement ... 139
5.7 Axis couplings via synchronized actions ... 143
5.7.1 Coupling to leading axis ... 143
5.7.2 Non-circular grinding via master value coupling ... 144
5.7.3 On-the-fly parting ... 148
5.8 Technology cycles position spindle ... 149
5.9 Synchronized actions in the TC/MC area ... 151
**6 Data lists ... 155**
6.1 Machine data ... 155
6.1.1 General machine data ... 155
6.1.2 Channelspecific machine data ... 155
6.1.3 Axis-specific machine data ... 156

6.2 Setting data ..... 156
6.2.1 Axis/spindle-specific setting data ..... 156
6.3 Signals ..... 157
6.3.1 Signals from channel ..... 157
A Appendix ..... 159
A.1 Publication-specific information ..... 159
A.1.1 Correction sheet - fax template ..... 159
A.1.2 Overview ..... 161
Index ..... 163

# Brief description

# 1

## Definition of synchronized actions

Motion-synchronous actions (or "synchronized actions" for short) are instructions programmed by the user, which are evaluated in the interpolation cycle of the NCK in synchronization with the execution of the part program. If the condition programmed in the synchronized action is fulfilled or if none is specified, then actions assigned to the instruction are activated in synchronism with the remainder of the part program run.

## Applications

The following selection from the wide range of possibilities indicates how actions programmed in synchronized actions can be usefully employed.

- Output of auxiliary functions to PLC
- Writing and reading of main run variables
- Positioning of axes and spindles
- Activation of synchronous procedures, such as:
  - Read-in disable
  - Deletion of distance-to-go
  - End preprocessing stop
- Activation of technology cycles
- Online calculation of function values
- Online tool offsets
- Activation/deactivation of couplings/coupled motion
- Take measurements
- Enabling/disabling of synchronized actions

All possible applications of this function are described in the "Detailed Description" chapter.

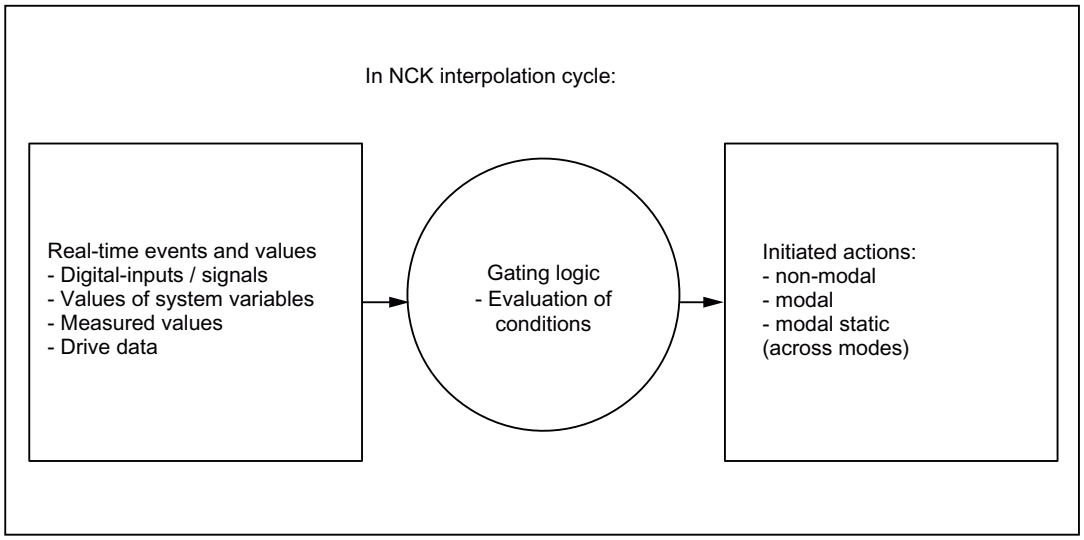



Figure 1-1 Schematic diagram of synchronized actions

For details of how to program synchronized actions, please see:

**References:**

/PGA/Programming Manual Advanced

The following chapters describe:

- functional relationships for synchronized actions in the Chapter "Detailed Description",
- Application examples in the Chapter "Examples".

---

**Note**

This description encompasses the functionality of the current software version. The synchronized actions available in SW version 3 and earlier are described in:

**References:**

/FB2/ Function Manual, Extended Functions; Synchronized Actions (S5).

---

# Detailed description

# 2

## 2.1 Components of synchronized actions

### Structure of a synchronized action

| Component: | Validity, identification number | Frequency | G code for cond. and action | Condition | Action code word (fixed) | G code for action | Action or Tech-no-lo-gy-cy-cle |
|---|---|---|---|---|---|---|---|
| Example | IDS=1 | EVERY | G70 | $AAA_IM[B] > 15 | DO | G71 | POS[X]=100 |

The components of the synchronous actions are explained below individually:

- Validity:
  - With identification number
  - Without identification number
- Frequency
- G code for condition and action (SW 5 and later)
- Condition
- G code for actions (SW 5 and later)
- Action(s)/Technology cycle

### Validity ID number

There are three possible ways for defining the scope of validity of a synchronized action:

1. No specification
2. ID
3. IDS

### No data

Synchronized actions that have no specified validity have a non-modal action, i.e. they apply only to the next block.

Non-modal synchronized actions are operative only in AUTOMATIC mode.

From **SW 6.1** and later, non-modal synchronized actions are active modally for all preprocessing stop blocks (incl. implicitly generated ones) and for implicitly generated intermediate blocks.

### ID

Synchronized actions with validity identifier **ID** act **modally** in the subsequently programmed blocks. They are operative only in **AUTOMATIC** mode.

Limitation:

- ID actions remain operative only until another synchronized action with the same identification number is programmed
- till deletion, CANCEL(i), see Chapter "Coordinations between synchronized actions, Technology cycles, Part program (and PLC)".

### IDS

Statically effective synchronized actions that are programmed with the keyword "**IDS**", are active **in all operating modes**. They are also referred to as static synchronized actions. Option.

Synchronized actions programmed with ID or IDS are deleted from the part program.

### Identification numbers

For modal synchronized actions (ID, IDS) identification numbers between 1 and 255 are allocated. They are important for the functions of mutual coordination of synchronized actions. See Chapter "Coordination of synchronized actions, technology cycles, part program (and PLC)".

Modal/static synchronized actions with identification numbers between 1 and 64 can be disabled and enabled from the PLC. See Chapter "Control via PLC".

Unique identification numbers must be allocated in the channel.

**Applications for static synchronized actions:**

- AC-grinding also active in the JOG mode
- Logic operations for Safety Integrated
- Monitoring functions, responses to machine states in all modes
- Ooptimization of the tool change
- Cyclic machines

Examples:

IDS=1 EVERY $A_IN[1]==1 DO POS[X]=100: all modes

ID=2 EVERY $A_IN[1]==0 DO POS[X]=0: AUTOMATIC

**Note**

The following actions are operative only in AUTOMATIC mode when the program is running:
STOPREOF
DELDTG

## Frequency

Keywords (see table) are programmed to indicate how often the subsequently specified condition must be scanned and the associated action executed if the condition is fulfilled. These keywords are an integral component of the synchronized action condition.

Table 2-1 Effect of frequency keywords

| Keyword | Scanning frequency |
|---|---|
| None | If no scanning frequency is programmed, then the action is executed cyclically in every interpolation cycle. |
| WHENEVER | The associated action/technology cycle is executed cyclically in every interpolation cycle provided that the condition is fulfilled. |
| FROM | If the condition has been fulfilled once, the action/technology cycle is executed cyclically in every interpolation cycle for as long as the synchronized action remains active. |
| WHEN | As soon as the condition has been fulfilled, the action/technology cycle is executed once. Once the action has been executed a single time, the condition is no longer checked. |
| EVERY | The action/technology cycle is activated once if the condition is fulfilled. The action/technology cycle is executed every time the condition changes from the FALSE to the TRUE state.<br>In contrast to keyword WHEN, checking of the condition continues after execution of the action/cycle until the synchronized action is deleted or disabled. |

Details of technology cycles are given in the Chapter "Call of technology cycles".

## Deleting with CANCEL

If an active synchronized action is cancelled (deleted) with CANCELfrom the part program, then the active action is not affected. Positioning motions are completed as programmed. With the command CANCEL a modal or a statically active synchronized action can be deleted.

If a synchronized action is deleted while the positioning axis motion it has initiated is still in progress, the positioning motion continues until properly executed. A channel stop also cancels the positioning movement from synchronized actions/technology cycles.

## G code for condition and action

Starting from **SW-version 5** the G-codes can be programmed in synchronized actions. This allows defined settings to exist for the evaluation of the condition and the action/technology cycle to be executed, independent of the current parts program status. It is necessary to separate the synchronized actions from the program environment, because synchronized actions are required to execute their actions at any time from a defined initial state as a result of fulfilled trigger conditions.

Applications:

Definition of the measurement systems for condition evaluation and action through G codes `G70, G71, G700, G710`.

**Note**

Only one G-code of the G-code group may be programmed for each part of the condition. A G-code specified for the condition is valid for the evaluation of the condition and for the action, if no separate G-code is specified for the action. A G code specified for the condition is valid for the evaluation of the condition **and** for the action if no separate G code is specified for the action.

## Conditions

The execution of actions / technology cycles can be made dependent upon a condition (**logical expression**).

The condition is checked in the interpolation cycle. If no condition is programmed, the action is performed once in every IPO cycle.

Possible components include:

- Comparison of main run variables such as digital or analog inputs/outputs
- Boolean gating of comparison results
- Computation of real-time expressions
- Time/distance from beginning of block
- Measured values, measurement results
- Servo values
- Velocities, axis status

In SW version 3 and earlier, two conditions are permitted, i.e. the comparison of a main run variable with an expression calculated during preprocessing or the comparison of two main run variables.

Examples:

WHENEVER **$AA_IM[X] > 10.5*SIN(45**) DO ...

or

WHENEVER **$AA_IM[X] > $$AA_IM[X1]** DO ...

Additional comparisons using Boolean gating can also be interlinked. Boolean operators of the NC language may be used for this purpose:

`NOT, AND, OR, XOR, B_OR, B_AND, B_XOR, B_NOT`.

Examples:

WHENEVER ($A_IN[1]==1) **OR** ($A_IN[3]==0) DO ...

; while input 1 is applied or input 3 is not applied ...

Two or more real-time expressions may be compared with one another within one condition. Comparisons may be made between variables **of the same type** or between partial expressions.

Examples:

Example

WHEN $AA_IM[X2] <= $AA_IM[X1] +.5 DO $AA_OVR[X1]=0

; Stop, when the safety clearance is exceeded.

Further examples of conditions can be found in Chapter 6.1. The calculations of real-time expressions are described in the Chapter "Evaluations and calculations in real time".

All system variables whose data are read or written by the NC via synchronized actions can be addressed in conditions. So can all machine data and setting data where the value is read in the main run:

Machine data, e.g. $$MN_..., $$MC_..., $$MA_...

Setting data, e.g. $$SN_..., $$SC_..., $$SA_...

**Note**

GUD variables cannot be used

R-parameters are addressed with $R... .

Setting data and machine data, whose value may vary during machining, must be programmed with **$$**S._.../**$$**M._... .

More examples of conditions are given in the Chapter "Examples for conditions in synchronized actions".

## G code for the action

The G code may specify a different G code from the condition for all actions in the block and technology cycles. If technology cycles are contained in the action part, the G code remains modally active for all actions until the next G code, even after the technology cycle has been completed.

Only one G code of the G code group may be programmed for each action part.

## Actions

Every synchronized action contains one or more programmed actions or one technology cycle. These are executed when the appropriate condition is fulfilled. If several actions are programmed in one synchronized action, they are executed within the same interpolation cycle.

Example

WHEN $AA_IM[Y] >= 35.7 DO **M135 $A_OUT[1]=1**

If the actual value of the Y axis is greater than or equal to 35.7, then M135 is output at PLC and at the same time the output 1 is set.

## Program/technology cycle

A program (name) can also be specified as an action. This program may contain any of the actions, which can be programmed individually in synchronized actions. These programs are also referred to as technology cycles below. A technology cycle is a sequence of actions that are processed sequentially in the interpolation cycle, see

Chapter 2.5. "Call of Technology Cycles".

Application: Single axis programs, cyclic machines.

## Machining process

The blocks of a part program are prepared at the program preprocessing stage, stored and then executed sequentially at the interpolation level (main run). Variables are accessed during block preparation. When main-run variables (e.g., actual values) are used, block preparation is interrupted to allow current real-time values up to the preceding block to be supplied.

Synchronized actions are transported to the interpolator in preprocessed form together with the prepared block. The main run variables used are evaluated in the interpolation cycle. Block preparation is not interrupted.

Part program blocks without traversing motion (zero blocks) are normally eliminated by the interpreter. However, if synchronous actions are active, this zero block is included and also executed. Here, an exact stop corresponding to the active function (e.g. `G601`) is triggered. This allows the synchronous action to also switch.

Examples of zero blocks:

```
N1000 G91 X0 Y0 Z0
...
```

```
N10 G90 G64 X100 Y100 Z100
N15 Z100
...
```

Blocks without traversing motion can also be generated using program jumps.

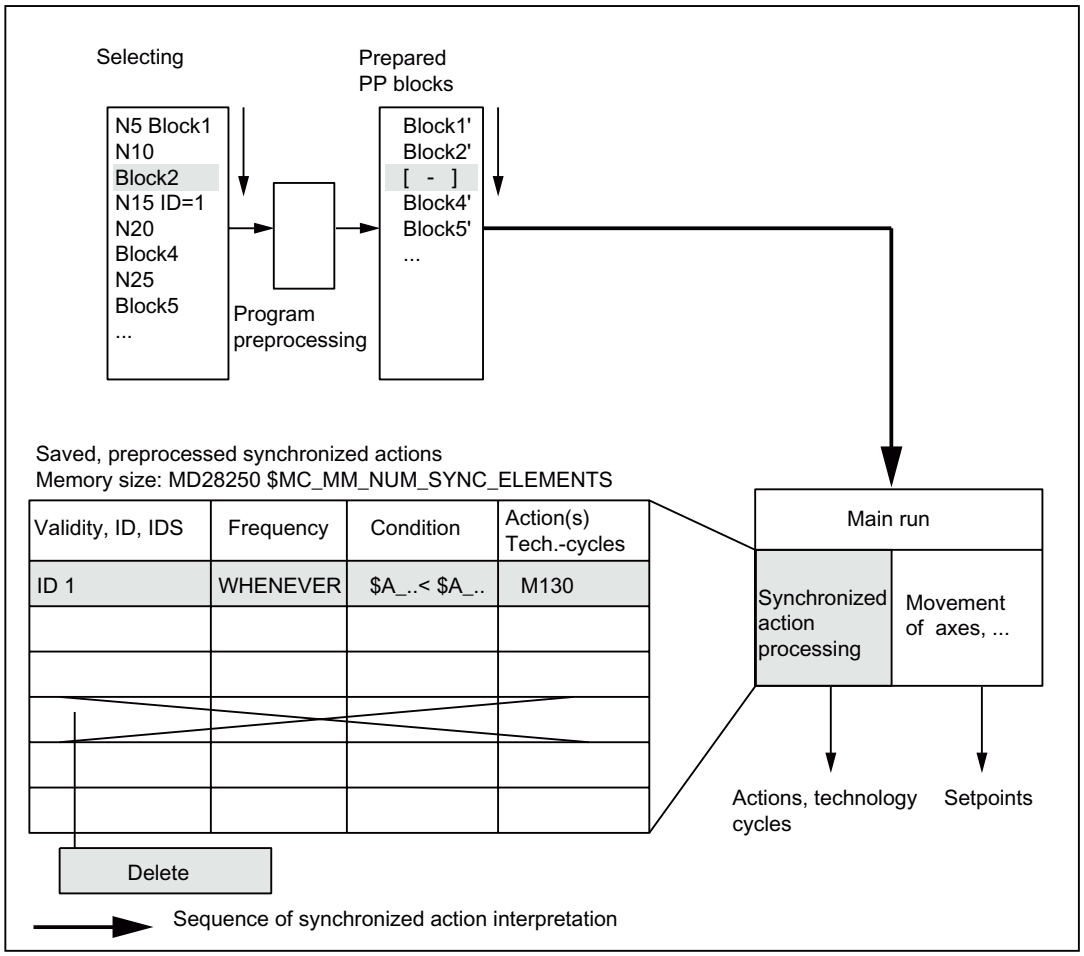



Figure 2-1 Schematic diagram illustrating processing of synchronized actions

## Processing of synchronized actions

Synchronized actions are checked in the interpolation cycle to determine whether they contain actions to be activated.

Action(s) are executed in synchronism with path control if the preconditions programmed on the left of the action(s) are fulfilled.

## Order of execution

Within an interpolation cycle, modal synchronized action instructions are processed in order of their ID number (i.e. block with ID number 1 before block with ID number 2, etc.). Once the modal synchronized action instructions have been executed, non-modal synchronized action instructions are processed in the order in which they are programmed.

## 2.1.1 Definition of motion-synchronous actions

### Defining programs

Motion-synchronous actions can be defined as follows:

- In the part program
- Static synchronized actions in an asynchronous subprogram activated by the PLC

## 2.1.2 Execution of synchronized actions

### Conditions for execution

The actions programmed in motion-synchronous actions are executed, when:

- the synchronized action exists and has not been deselected with CANCEL(ID).
- the synchronous action is not locked, no LOCK(ID), see Chapter 2.5.1 "Coordination of synchronized actions, technology cycles, part programs and PLC".
- evaluation of the action is due as a result of the programmed frequency keyword or
- the appropriate condition is fulfilled.
- For further details, please see the following subsections.

## 2.1.3 List of possible actions

- Output of M, S and H auxiliary functions to the PLC
- Setting (writing) of main run variables enables the following:
  - Overlaid movement ($AA_OFF), option.
  - Applied tool length offsets ($AA_TOFF), option.
  - Feedrate control ($AC_OVR, $AA_OVR), disabling of a programmed axis motion
  - Allocate Servo–data–values $V...=
  - Reading or writing arithmetic variable $R[n]=
  - Writing the SD value in the main run $$SD
- Reading the MD value for the interpretation time $MD...=
- Changing of SW cam positions and times (setting data)
- Modification of coefficients and limits from FCTDEF
- Polynomial evaluation SYNFCT
- Online tool offset FTOC

- Read-in disable RDISABLE
- Preprocessing stop cancellation STOPREOF
- Delete distance-to-go DELDTG
- Calculation of curve table values
- Axial feedrate from synchronized actions
- Axial frame
- Moving/positioning axes/spindles from synchronized actions
- Axis replacement from synchronized actions
- Spindle motions from synchronized actions
- Actual-value setting from synchronized actions (Preset)
- Activation/deactivation of couplings and coupled motion
- Activation/deactivation of coupling modules of the generic coupling
- Measurements from synchronized actions
- Setting and deleting wait markers for channel synchronization
- Set alarm/error reactions
- Travel to fixed stop FXS (FXST, FXSW)
- Travel with restricted moment FOC (FOCON/FOCOF)
- Extended stopping and retraction (Description of Functions M3)
- Reading and, if tagged accordingly, writing of system variables, see /PGA1/ Parameter Manual System Variables.

These actions are described in detail in Section 2.4.

## 2.2 Real-time evaluations and calculations

### Restriction

Calculations carried out in real time represent a subset of those calculations that can be performed in the NC language. It is restricted to data types `REAL, INT, CHAR` and `BOOL`.

Implicit type conversions between REAL, INT, and BOOL in both directions are possible in synchronized actions. During value assignments and parameter transfers, variables of different data types can be assigned or transferred.

### Scope of application

The term "Real-time expression" refers below to all calculations that can be carried out in the interpolation cycle. Real-time expressions are used in conditions and in assignments to NC addresses and variables.

### Main run variables

All main-run variables are evaluated (read) in the interpolation cycle and can be written as part of an action.

The system variables available in synchronized actions during the main run are identified with a field provided for this purpose in the "List of System Variables" in /LHB/ Manual of System Variables.

### Identification of main-run variables

Real-time variables are all variables that begin with:

**$A... ,current main-run variable**

**$V... , Servo values**

**$R... , R parameters.**

NC–machines– and setting data in the main run are interpreted with

**$$M...**

**$$S...**

in synchronized actions.

---

**Note**

You can read the machine– and setting data values during the main run, they are addressed for an online access with $$S... or $$M... and evaluated during the main run, while MD– or SD–values are read with a $–character at the time the synchronized action is interpreted.

Setting data and machine data from the synchronized action are addressed with the $ sign and evaluated at the time of preprocessing.

---

### Data type:

Within an expression in the synchronized actions only the main-run variables **of a data type** may be linked logically. However, in order to process various types of data, you can use the conversion routines provided for type matching (SW 5.2, see conversion routines). In contrast to full expression in the NC language, the calculation is performed in the data type of the main run variables.

```
... DO $R10 = $AC_PARAM[0]          ;  Permissible REAL, REAL
... DO $R10 = $AC_MARKER[0]         ;  not permitted REAL, INT
```

The following examples of real-time evaluations were already available in SW version 3.2 (they employ only real-time variables of this SW version):

### Conversion routines in SW 5.3 and higher

The user can explicitly call two conversion routines RTOI( ) and ITOR( ) for the type conversion in the synchronized action implicitTypeconversion from REAL to INT, and vice versa. These functions are intended for software versions in which an internal type conversion of values was not yet possible. Both conversion routines can be called

- in the part program, and
- from the synchronized action.

can be called.

### Conversion routines (SW 5.2)

In the synchronized action there is no implicit type conversion from `REAL` to `INT` and vice versa. However, the user may explicitly call two conversion routines **RTOI( ) and ITOR( )** for the type conversion. The functions can be called as follows:

- in the part program, and
- from the synchronized action.

### ITOR

`REAL` ITOR( INT ) - Converting integer to real

The function converts the integer value transferred to a real value and returns this value. The transferred variable is not changed.

**Example**

$AC_MARKER[1]=561

ID=1 WHEN TRUE DO $AC_PARAM[1] = **ITOR**( $AC_MARKER[1] )

## RTOI

`INT` RTOI( REAL ) - Converting from real to integer

The function RTOI() converts the transferred Real value in a rounded `INT` value and returns this integer value. If the value transferred lies outside the range that can be unambiguously represented as an integer value, alarm 20145 "Motion-synchronous action: Arithmetic error" is output and no conversion is performed. The transferred variable is not changed.

**Note**

The function RTOI() does not produce an unambiguous result when inverted, i.e. it is not possible to determine the original Real value from the value returned, as the decimal places are lost during conversion!

### Example RTOI:

```
$AC_PARAM[1] = 561.4378
ID=1 WHEN TRUE DO $AC_MARKER[1] =     ;  Result: 561
RTOI( $AC_PARAM[1] )
...
$AC_PARAM[1] = -63.867
ID=1 WHEN TRUE DO $AC_MARKER[1] =     ;  Result: -64
RTOI( $AC_PARAM[1] )
$AC_MARKER[1]=10
$AC_PARAM[1] = -6386798797.29
ID=1 WHEN TRUE DO $AC_MARKER[1] =     ;  Result: Alarm 20145
RTOI( $AC_PARAM[1] )                  ;  $AC_MARKER[1] = 10 (unchanged due to alarm)
```

## Implicit type conversion (SW 6.4)

In SW 6.4 and later, variables of various data types can be assigned to one another in synchronized actions without having to call the RTOI or ITOR function, e.g. REAL to INT and vice versa.

If values outside of the interval [INT_MIN, INT_MAX] would result from the conversion from REAL to INTEGER, alarm 20145 "Motion-synchronous action: Arithmetic error" is output and no conversion is performed.

Examples:

```
previously
$AC_MARKER[1]=561
ID=1 WHEN TRUE DO $AC_PARAM[1] = ITOR( $AC_MARKER[1] )
As of SW 6.4
$AC_MARKER[1]=561
ID=1 WHEN TRUE DO $AC_PARAM[1] = $AC_MARKER[1]

previously
$AC_PARAM[1] = 561.4378
ID=1 WHEN TRUE DO $AC_MARKER[1] = RTOI( $AC_PARAM[1] )                    ; 561
As of SW 6.4
$AC_PARAM[1] = 561.4378
ID=1 WHEN TRUE DO $AC_MARKER[1] = $AC_PARAM[1]                            : 561
```

### Internal type conversion of values in SW 7.4 and higher

Without the call of the RTOI and ITOR function, type conversions can be triggered for value assignments and parameter transfers of multiple implicit type conversions. The following data type conversions are possible:

- REAL to BOOL
- INT to BOOL
- BOOL to REAL or INT

Examples:

implicit conversion from INTEGER to BOOLen

$AC_MARKER[1]=561

ID=1 WHEN $A_IN[1] == TRUE DO $A_OUT[0]=$AC_MARKER[1]

implicit conversion from REAL to BOOLen

R401 = 100.542

WHEN $A_IN[0] == TRUE DO $A_OUT[2]=$R401

implicit conversion from BOOLen to INTEGER

ID=1 WHEN $A_IN[2] == TRUE DO $AC_MARKER[4] = $A_OUT[1]

implicit conversion from BOOLen to REAL

R401 = 100.542

WHEN $A_IN[3] == TRUE DO $R10 = $A_OUT[3]

For type conversions from REAL to INT, a fractional value < 0.5 is rounded off. Otherwise, the value is rounded down as with the ROUND function. Alarm 20145 is output if values are exceeded.

References:

/PGA/ Programming Manual Work Preparation, Motion-Synchronous Actions, Chapter "Main-Run Variables for Synchronized Actions"

### Basic arithmetic operations

Real-time variables of the type `REAL` and `INT` can be linked logically by the following basic arithematic operations:

- Addition
- Subtraction
- Multiplication
- Division
- Integer division
- Modulo division.

Only variables of the same type may be linked by these operations.

### Expressions

Expressions from basic arithmetic operations can be put in brackets and nested. See priorities for operators on the next page.

### Comparisons

The following relational operators may be used:

| | |
|---|---|
| **==** | Equal to |
| **>** | Not equal to |
| **<** | Less than |
| **>** | Greater than |
| **<=** | Less than or equal to |
| **>=** | Greater than or equal to |

### Boolean operators

The following Boolean operators may be used:

| | |
|---|---|
| `NOT` | NOT |
| `AND` | AND |
| `OR` | OR |
| `XOR` | Exclusive OR |

### Bit operators

The following bit operators may be used:

| | |
|---|---|
| `B_OR` | bit OR |
| `B_AND` | bit AND |
| `B_XOR` | bit-serial exclusive OR |
| `B_NOT` | bit negation |

Operands are variables and constants of the `INT` type.

## Priority of operators

In order to produce the desired logical result in multiple expressions, the following operator priorities should be observed in calculations and conditions:

| | | |
|---|---|---|
| 1. | NOT, B_NOT | Negation, bit-serial negation |
| 2. | *, /, DIV, MOD | Multiplication, division |
| 3. | +, - | Addition, subtraction |
| 4. | B_AND | bit AND |
| 5. | B_XOR | bit-serial exclusive OR |
| 6. | B_OR | bit OR |
| 7. | AND | AND |
| 8. | XOR | Exclusive OR |
| 9. | OR | OR |
| 10. | | not used |
| 11. | | Relational operators |
| | == | Equal to |
| | <> | Not equal to |
| | < | Greater than |
| | > | Less than |
| | >= | Greater than or equal to |
| | <= | Less than or equal to |

and parentheses should be used where necessary. The logic operation result for a condition must be a BOOL data type.

Example of a multiple expression:

WHEN ($AA_IM[X] > VALUE) AND ($AA_IM[Y] > VALUE1) DO ...

## Functions

A main-run variable of the REAL type can be used to create function values sine, cosine, etc.

The following functions are possible:

SIN, COS, ABS, ASIN, ACOS, TAN, ATAN2, TRUNC, ROUND, LN, EXP, ATAN, POT, SQRT, CTAB, CTABINV

**Example**

... DO $AC_PARAM[3]=**COS**($AA_IM[X])

For a description of how to use these functions, please see:

**References:**

/PG/ Programming Manual Fundamentals

/PGA/Programming Manual Advanced

### Indexing

The index of a main-run field variable can in turn be a real-time variable.

**Example**

WHEN ... DO $AC_PARAM[ **$AC_MARKER[1]** ] = 3

The index $AC_MARKER[1] is evaluated currently in the interpolation cycle.

Constraints:

- It is not permissible to nest indices with real-time variables.
- A real-time index cannot be generated by a variable that is not generated itself in real time. The following programming would lead to errors:
  $AC_PARAM[1]=$**P_EP[$AC_MARKER[0]**

## 2.3 Special real-time variables for synchronized actions

For a complete list of the system variables that can be addressed, see references:

/PGA1/ Parameter Manual, System Variables.

The system variables that can be addressed in synchronous actions are in the field SA: identified as "Can be used in synchronized actions". A system variable that is updated during the main run is also synchronized in the main run if this is selected in the HL Sync field and then confirmed.

The characteristics of a few special main run variables are described below:

- Marker/counter variables
  - Channel-specific markers
- Timers
- Synchronized action parameters
- R parameters
- Machine and setting data
- `FIFO` variables (circulating memory)

### SW 4

The special real-time variables: Timers, R parameters, machine and setting data and `FIFO` variables are available in SW 4 and later.

## 2.3.1 Marker/counter variables

### Channel-specific markers

The array variable $AC_MARKER[n] is used as a marker or counter and can be read and written in synchronized actions.

Data type: INTEGER

n: Field index of the marker: 0–n

The number of markers per channel is set using machine data

MD28256 $MC_NUM_AC_MARKER.

In the machine data 28256: $MC_NUM_AC_MARKER the

maximum value that can be specified is 20000.

One element requires 4 bytes. You must ensure that sufficient memory of the correct type is available. These markers exist once in each channel under the same name. The memory location for $AC_MARKER[n] can be selected with the machine data

MD28257 $MC_MM_BUFFERED_AC_MARKER with

0: in the dynamic memory (active file system)

1: in the static memory (passive file system)

can be selected.

Markers stored in the static memory can be included in the data backup.

front panel. See 2.3.7

On Power On, NC Reset, and End of Program, the markers are set to 0.

This ensures the same start conditions for every program run.

General procedure

### In addition, for SW 6.3 and higher

It is possible to select the memory location for $AC_MARKER[n] between DRAM and SRAM with the following machine data:

MD28257 $MC_MM_BUFFERED_AC_MARKER ($AC_MARKER[] is saved in SRAM)

0: dynamic memory DRAM, (default)

1: static memory SRAM

20000 can be specified as the highest value in the machine data:

MD28256 $MC_NUM_AC_MARKER (dimension of $AC_MARKER)

One element requires 4 bytes. You must ensure that sufficient memory of the correct type is available.

Markers saved in SRAM can be included in the data backup. See Chapter "System variables saved in SRAM (SW 6.3 and later)"

## 2.3.2 Timers

The system variable $AC_TIMER[n] permits actions to be started after defined periods of delay.

### Definition

Data type: REAL

n: Number of timer variable

Unit: Second

The number of available timer variables is determined via the machine data MD28258 $MC_MM_NUM_AC_TIMER.

### Setting timers

The incrementing of a timer variable is started by means of value assignment:

$AC_TIMER[n]=value

n: Number of time variable

value: Start value (normally 0)

### Stopping timers

Incrementation of a timer variable can be stopped by assigning a negative value:

$AC_TIMER[n]=-1

### Reading timers

The current timer value can be read whether the timer variable is running or has been stopped. After a timer variable has been stopped through the assignment of -1, the current time value remains stored and can be read.

### Example

Output of an actual value via analog output 500 ms after detection of a digital input:

```
WHEN $A_IN[1]==1 DO $AC_TIMER[1]=0       ;  Reset and start timer
WHEN $AC_TIMER[1]>=0.5 DO $A_OUTA[3]=$AA_IM[X] $AC_TIMER[1]=-1
```

## 2.3.3 Synchronized action parameters

### Meaning

The variables $AC_PARAM[n] are used for calculations and also as intermediate memory and can be read and written in synchronized actions.

### Definition

Data type: REAL

n: Number of parameters 0 - n

The number of available AC parameter variables for each channel is determined via the Machine data

MD28254 $MC_MM_NUM_AC_PARAM

fixed.

These parameters exist once in each channel under the same name. $AC_PARAM parameters are stored in the dynamic memory.

In MD28255 $MC_NUM_AC_PARAM, the maximum value that can be specified is 20000.

One element requires 4 bytes. You must ensure that sufficient memory of the correct type is available. The memory location for $AC_PARAM[n] can be selected with the machine data

MD28255 $MC_MM_BUFFERED_AC_PARAM with

0: in the dynamic memory (active file system, default)

1: in the static memory (passive file system)

can be selected.

Synchronized action parameters stored in the static memory can be included in the data backup. See 2.3.7 The parameters are set to 0 on Power On, NC Reset, and End of Program, . identical start conditions for every part program run.

### In addition, for SW 6.3 and higher

In software version 6.3 and later, it is possible to select the memory location for $AC_PARAM[n] between DRAM and SRAM using the machine data:

MD28255 $MC_MM_BUFFERED_AC_PARAM ($AC_PARAM[] is saved in SRAM)

0: dynamic memory DRAM, (default)

1: static memory SRAM

2000 can be specified as the highest value in the machine data MD28255.

One element requires 8 bytes. You must ensure that sufficient memory of the correct type is available.

Synchronization parameters saved in SRAM can be included in the data backup. See Chapter "System variables saved in SRAM (SW 6.3 and later)"

## 2.3.4 R parameters

### Application in synchronized actions

By programming the $ sign in front of R parameters, they can also be used in synchronized actions. The field variables $R[n] or $Rn are used for calculations in synchronized actions R[n] or Rn is used for calculations in part program, which are stored battery-backed in the static memory. Therefore, R parameters retain their values after end of program, RESET, or POWER ON.

Example

Import measured value in the R-parameter

```
WHEN $AC_MEA== 1 DO $R10= $AA_MM[Y]
; if valid measurement is present, import measured value in R-parameter.
```

### Definition

Data type REAL

n: Number of the field variable

### Example

Evaluation of the R-parameter

```
WHEN $A_IN[1] == 1 DO $R10 = $AA_IM[Y]
G1 X100 F150
STOPRE
IF R10 > 50 .... ; Evaluation of the R-parameter.
```

## 2.3.5 Machine and setting data

### Reading and writing MD, SD

It is also possible to read and write machine data and setting data from synchronized actions.

possible.  Access must be differentiated according to the following criteria:

MD, SD that remain unchanged during machining and

MD, SD, whose settings change during machining.

### Reading at the time of preprocessing

Unchangeable machine and setting data are addressed from the synchronized action as in the normal part program commands. They are introduced with a $-sign.

Example

```
ID=2 WHENEVER $AA_IM[z]< $SA_OSCILL_REVERSE_POS2[Z]–6 DO
$AA_OVR[X]=0
; Here the reversal range 2 assumed as unchanging during the processing
; is addressed for oscillation
```

A complete example for oscillation with setting in the reversal range is given in section 6.2 and:

References:

/FB2/Function Manual, Expanded Functions; Oscillation (P5).

### Reading at the time of the main run

Machine data and setting data which change during the processing are addressed from the synchronized action introduced with $$-sign.

Example

```
ID=1 WHENEVER $AA_IM[z]< $$SA_OSCILL_REVERSE_POS2[Z]–6 DO
$AA_OVR[X]=0
```

In this context, it is assumed that the reversal position can be changed any time during the operation.

### Writing at the time of the main run

Requirement:

The currently set access authorization level must allow write access. It is only useful to write MD and SD from the synchronized action when this takes immediate effect. The effectiveness for all machine data and setting data is specified in:

References:

/LIS1/ Lists (Book 1), Section "MD-/SD Lists".

Addressing:

Machine and setting data to be written must be addressed preceded by the $$ sign.

Example

```
ID=1 WHEN $AA_IW[X]>10 DO $$SN_SW_CAM_PLUS_POS_TAB_1[0]=20
$$SN_SW_CAM_MINUS_POS_TAB_1[0]=30
; Changing the switching positions of SW-cams from 20 to 30
```

## 2.3.6 FIFO variables (circulating memory)

### Application

Up to 10 FIF0 variables are provided to allow storage of related data sequences: **$AC_FIFO1[n]** to **$AC_FIFO10[n]**.

### Structure

The memory structure of a FIF0-variable is shown in the figure: Example of FIF0 variable

### Amount

The number of the available AC FIFO variable is specified in machine data

MD28260 $MC_NUM_AC_FIFO

### Size

The number of values that can be stored in a FIFO variable is defined via machine data

MD28264 $MC_LEN_AC_FIFO

All FIFO variables have the same length.

### Data type:

Values in the FIF0-variables have the data type REAL.

### Meaning of index

Index n:

**Indices 0 to 5 have special meanings:**

n=0: When writing with index 0 a new value is stored in the FIFO. When reading with index 0 the oldest element is read and deleted from the FIFO.

n=1: Access to oldest stored element

n=2: Access to latest stored element

n=3: Sum of all FIFO elements

The MD28266 $MC_MODE_AC_FIFO determines the mode

the summation:

Bit 0 = 1 sum upon each writing

Updating

Bit 0 = 0 no summation possible. n=4: Number of elements available in FIF0. Read and write access can be assigned to each element of the FIF0.

FIF0 variables are reset by resetting the number of elements, e.g. for the first FIF0 variable: **$AC_FIFO1[0]=0**

n=5: Current write index relative to beginning of FIF0

n= 6 to 6+nmax: Access to the nth FIF0-element:

**Note**

The FIF0 access is a special form of R parameter access: (see below)

The FIF0-values are stored in the R-parameter range.

The FIF0 values are stored in the static storage area. They are not deleted by end of program, RESET or POWER ON.

The FIF0 values are stored simultaneously when R parameters are archived.

Machine data

MD28262 $MC_START_AC_FIFO

defines the number of the R parameter, which marks the beginning of FIFO variables storage in the R parameter area.

The actual number of R parameters in a channel is programmed in machine data

MD28050 $MC_MM_NUM_R_PARAM

is defined. The following two diagrams show a schematic representation of part lengths of parts on a belt that have been stored in FIF0 variables.

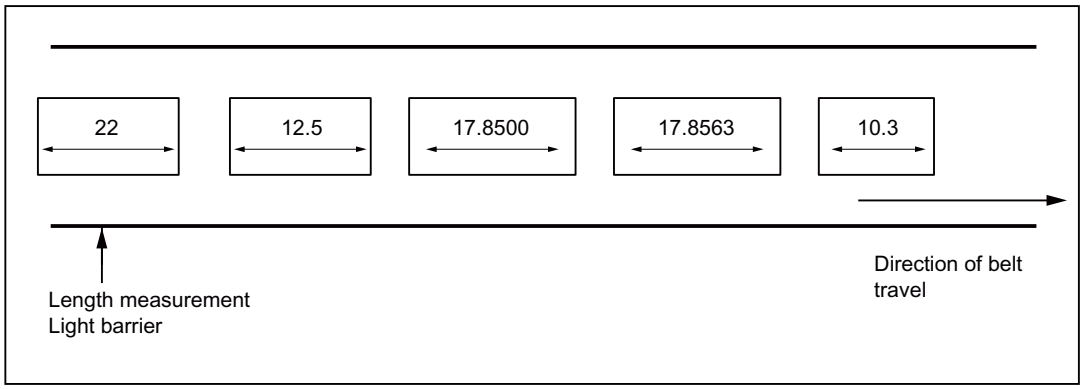



Figure 2-2 Product lengths of sequence of parts on conveyor belt

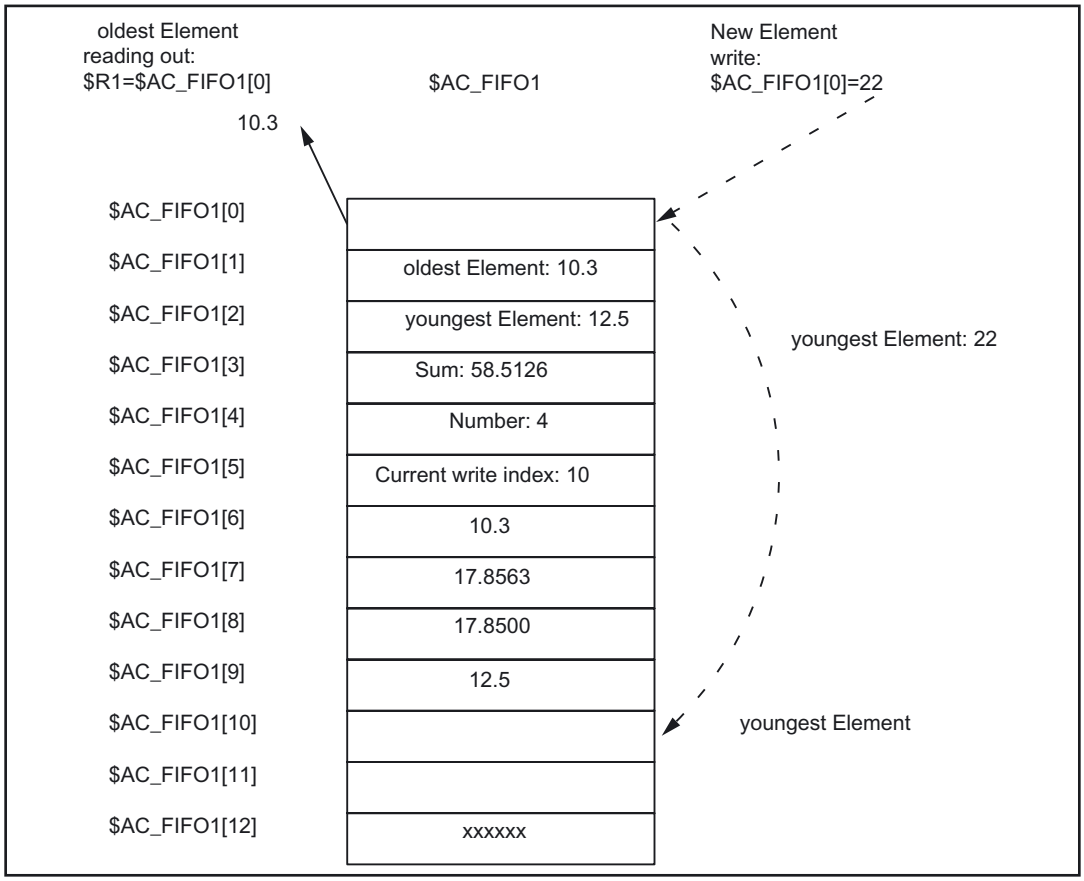



Figure 2-3 FIFO variable

## 2.3.7 System variables saved in SRAM (SW 6.3 and later)

### RESET response

The system variables $AC_MARKER and $AC_PARAM saved in SRAM retain their existing values after RESET and Power On.

**Note**

In the case of part programs and synchronized actions that work with system variables saved in SRAM, you must make sure that the variables are not initialized to 0 after RESET. This may require some adaptation if system variables saved in DRAM have been used previously.

### Data Backup

System variables $AC_MARKER and $AC_PARAM saved in SRAM can be included in the data backup.

The following backup modules are present for each channel:

| | |
|---|---|
| _N_CHi_ACM | for $AC_MARKER values and |
| _N_CHi_ACP | for $AC_PARAM values. |

i denotes the relevant channel number.

### Sequence

The saved modules are entered in the full backup file _N_INITIAL_INI according to R parameters.

**References:**

/IAD/ Commissioning Manual; NCU 840D and CCU 810D.

## 2.3.8 Determining the path tangent in synchronized actions

### $AC_TANEB

The system variable $AC_TANEB (**T**angent **AN**gle at **E**nd of **B**lock), which can be read in synchronized actions, calculates the angle between the path tangent at the end of the current block and the path tangent at the start of the programmed following block.

The tangent angle is always output positive in the range 0.0 to 180.0 degrees. If there is no following block in the main run, the angle -180.0 degrees is output.

The system variable $AC_TANEB should not be read for blocks generated by the system (intermediate blocks). The system variable $AC_BLOCKTYPE is used to tell whether it is a programmed block (main block).

**Programming example:**

```
ID=2 EVERY $AC_BLOCKTYPE==0 DO $R1 = $AC_TANEB;
```

## 2.3.9 Determining the current override

### Current override

The current override (NC-part) can be read and written in the synchronized actions with the following system variables:

| | |
|---|---|
| $AA_OVR | Axial override |
| $AC_OVR | Path override |

### PLC override

The override defined by the PLC is provided for synchronized actions for reading in the following system variables:

| | |
|---|---|
| $AA_PLC_OVR | Axial override |
| $AC_PLC_OVR | Path override |

### Resulting override

The resulting override is provided for synchronized actions for reading in the following system variables:

| | |
|---|---|
| $AA_TOTAL_OVR | Axial override |
| $AC_TOTAL_OVR | Path override |

The resulting override is calculated as follows:

| | | | |
|---|---|---|---|
| $AA_OVR | * | $AA_PLC_OVR | or |
| $AC_OVR | * | $AC_PLC_OVR | |

## 2.3.10 Capacity evaluation using time requirement for synchronized actions

### Description

In a interpolation cycle, synchronized actions have to be both interpreted and motions calculated by the NC. The system variables presented below provide synchronized actions with information about the current time shares that synchronized actions have of the interpolation cycle and about the computation time of the position controllers.

The variables have valid values only when the machine data

MD11510 $MN_IPO_MAX_LOAD is greater than 0. Otherwise the variables for both SINUMERIK powerline and solution line systems always specify the net computing time during which the interrupts caused by HMI are no longer taken into account. The net computing time results from:

- synchronized action time,
- position control time and
- remaining IPO-computing time without interrupts caused by HMI

The variables always contain the values of the

previous IPO cycle.

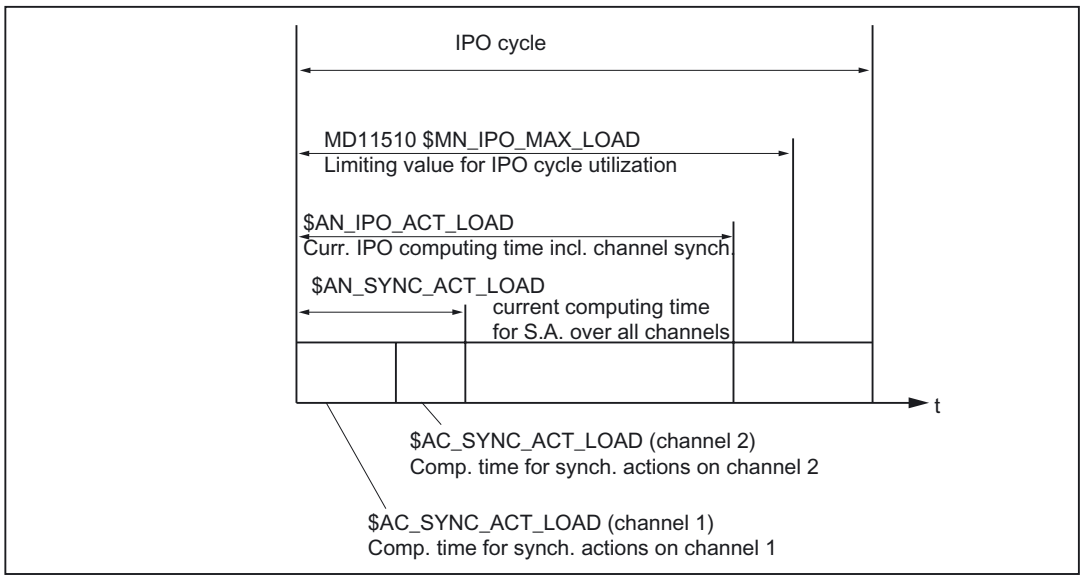



Figure 2-4 Time components of the synchronized actions in the IPO cycle

| System variables | Description |
|---|---|
| $AN_IPO_ACT_LOAD | current IPO computing time (incl. synchronized actions of all channels) |
| $AN_IPO_MAX_LOAD | longest IPO computing time (incl. synchronized actions of all channels) |
| $AN_IPO_MIN_LOAD | shortest IPO computing time (incl. synchronized actions of all channels) |
| $AN_IPO_LOAD_PERCENT | current IPO computing time as percentage of IPO cycle (%) |

| System variables | Description |
|---|---|
| $AN_SYNC_ACT_LOAD | current computing time for synchronized actions over all channels |
| $AN_SYNC_MAX_LOAD | longest computing time for synchronized actions over all channels |
| $AN_SYNC_TO_IPO | percentage share that the synchronized actions have of the complete IPO computer time (over all channels) |
| $AC_SYNC_ACT_LOAD | current computing time for synchronized actions in the channel |
| $AC_SYNC_MAX_LOAD | longest computing time for synchronized actions in the channel |
| $AC_SYNC_AVERAGE_LOAD | average computing time for synchronized actions in the channel |
| $AN_SERVO_ACT_LOAD | current computing time of the position controller |
| $AN_SERVO_MAX_LOAD | longest computing time of the position controller |
| $AN_SERVO_MIN_LOAD | shortest computing time of the position controller |

### Overload information

MD11510 $MN_IPO_MAX_LOAD is used to set the net IPO computing time (in % of IPO cycle) starting from which the system variable $AN_IPO_LOAD_LIMIT is to be set to TRUE.

If the current load falls below this limit, the variable is again set to FALSE.

If the MD is 0, the entire diagnostic function is deactivated. The user can evaluate $AN_IPO_LOAD_LIMIT to define a specific strategy for avoiding plane overflow.

### Writeable system variables

The system variables described above can be **written** from synchronized actions:

| System variables | Description |
|---|---|
| $AN_SERVO_MAX_LOAD | longest computing time of the position controller |
| $AN_SERVO_MIN_LOAD | shortest computing time of the position controller |
| $AN_IPO_MAX_LOAD | longest IPO computing time (incl. synchronized actions of all channels) |
| $AN_IPO_MIN_LOAD | shortest IPO computing time (incl. synchronized actions of all channels) |
| $AN_SYNC_MAX_LOAD | longest computing time for synchronized actions over all channels |
| $AC_SYNC_MAX_LOAD | longest computing time for synchronized actions in the channel |

On every write access, these variables are reset to the current load, regardless of the value written.

### Programming example

```
$MN_IPO_MAX_LOAD = 80                ; MD: Time use limit for IPO cycle
                                     ; As soon as $AN_IPO_LOAD_PERCENT > 80 %,
                                     ; $AN_IPO_LOAD_LIMIT is set to TRUE.

N01 $AN_SERVO_MAX_LOAD=0
N02 $AN_SERVO_MIN_LOAD=0
N03 $AN_IPO_MAX_LOAD=0
N04 $AN_IPO_MIN_LOAD=0
N05 $AN_SYNC_MAX_LOAD=0
N06 $AC_SYNC_MAX_LOAD=0

N10 IDS=1 WHENEVER $AN_IPO_LOAD_LIMIT == TRUE DO M4711 SETAL(63111)
N20 IDS=2 WHENEVER $AN_SYNC_TO_IPO > 30 DO SETAL(63222)
N30 G0 X0 Y0 Z0
...
N999 M30
```

The first synchronized action generates an auxiliary function output and an alarm, if the entire time use limit is exceeded.

The second synchronized action generates an alarm if the share that the synchronized action has of the IPO computing time (over all channels) exceeds 30%.

## 2.3.11 List of system variables relevant to synchronized actions

**Note**

The system variables listed previously, which can be addressed from synchronized actions

are given in the independent print version starting from SW-version 7.1:

/PGA1/ Parameter Manual, System Variables.

All system variables with the corresponding identifier X can be used (read/written) by synchronized actions (SA). For further explanations on the properties of the system variables in the main run, see Section 2.3 "Special main run variables for synchronized actions".

# 2.4 Actions in synchronized actions

## Actions

Each synchronized action contains after the action code **DO ...** one or more (max. 16) actions or a technology cycle, which are executed when the condition is fulfilled. (**Actions** will now be used as a generic term.).

## Several actions

Several actions contained in a synchronized action are activated in the same interpolation cycle if the appropriate condition is fulfilled.

## List of possible actions

The following actions can be programmed in the "Action" section of synchronized actions:

Table 2-2 Actions in synchronized actions

| ... DO ... | Description | Reference |
|---|---|---|
| Mxx<br>Sxx<br>Hxx | Output of auxiliary functions to PLC | 2.4.1 |
| SETAL(no.) | Set alarm, error reactions | |
| $V... = ... | Assign variable (Servo-value) | 2.4.2 |
| $A...= ... | Assign variable (main-run variable) | |
| $AA_OFF = | – Superimposed motion | |
| $AC_OVR = | – Velocity control: | |
| $AA_OVR = | Path velocity | |
| $AC_VC = | Axis velocity | |
| $AA_VC = | add. path-feed offset | 2.4.3 |
| $$SN_SW_CAM_ ... | add. offset value of the axis | |
| $AC_FCT... | Changing the SW-cam positions | 2.4.4 |
| $AA_TOFF = | (Setting data) and all other SD | |
| | Overwriting of FCTDEF parameters | 2.4.8 |
| | – applied tool length offsets | |
| | Synchronized action procedures: | |
| RDISABLE | Activate read-in disable | 2.4.9 |
| STOPREOF | End preprocessing stop | 2.4.10 |
| DELDTG | Delete distance-to-go | 2.4.11 |
| FTOC | Online tool offset | 2.4.7 |
| SYNFCT | Polynomial evaluation | 2.4.5 |
| ZYKL_T1 (e.g.) | Call of Technology Cycles | 2.5 |

| ... DO ... | Description | Reference |
|---|---|---|
| | Control positioning axes: | |
| $AA_OVR[x]= 0 | Disabling an axis motion | 2.4.12 |
| ACHSE_X (e.g.) | Calling an axis program | 2.4.13 |
| POS[u]= ... | Position | 2.4.13 |
| FA[u]= ... | Determine axis feed rate | 2.4.14 |
| | Move command axis continuously: | 2.4.15 |
| MOV[u]= >0 | - forwards | " |
| MOV[u] = <0 | - backwards | " |
| MOV[u] = =0 | - stopping | " |
| AXTOCHAN | Axis replacement from a synchronized action | 2.4.16 |
| GET[axis] | Get axis for axis replacement | " |
| RELEASE(axis) | Release axis for axis replacement | " |
| | Spindles: | |
| SPOS | Position | |
| M3, M4, M5, S = | Direction of rotation, halt, RPM | 2.4.18 |
| $AA_OVR[S1]= 0 | Disabling the spindle motion | |
| PRESETON( , ) | Preset actual-value memory | 2.4.19 |
| | Coupled motion and couplings | 2.4.20 |
| | Activate/deactivate couplings: | |
| LEADON | Couple following axis with leading axis | |
| LEADOF | Disable coupling | 2.4.20 |
| TRAILON | Asynchr. Coupled motion on | |
| TRAILOF | Asynchr. coupled motion off | |
| MEAWA | Measurement without deletion of distance-to-go | 2.4.21 |
| MEAC | Cyclical measurement | |
| | Channel synchronization: | 2.4.22 |
| SETM | Setting a wait mark | |
| CLEARM | Deleting a wait mark | |
| | Coordination of synchronized actions: | 2.5.1 |
| LOCK | - Disable synchronized action/technology cycle | |
| UNLOCK | - Release synchronized action/technology cycle | |
| RESET | - Reset technology cycle | |

## 2.4.1 Output of M, S and H auxiliary functions to the PLC

For general information about auxiliary function outputs, please see:

**References:**

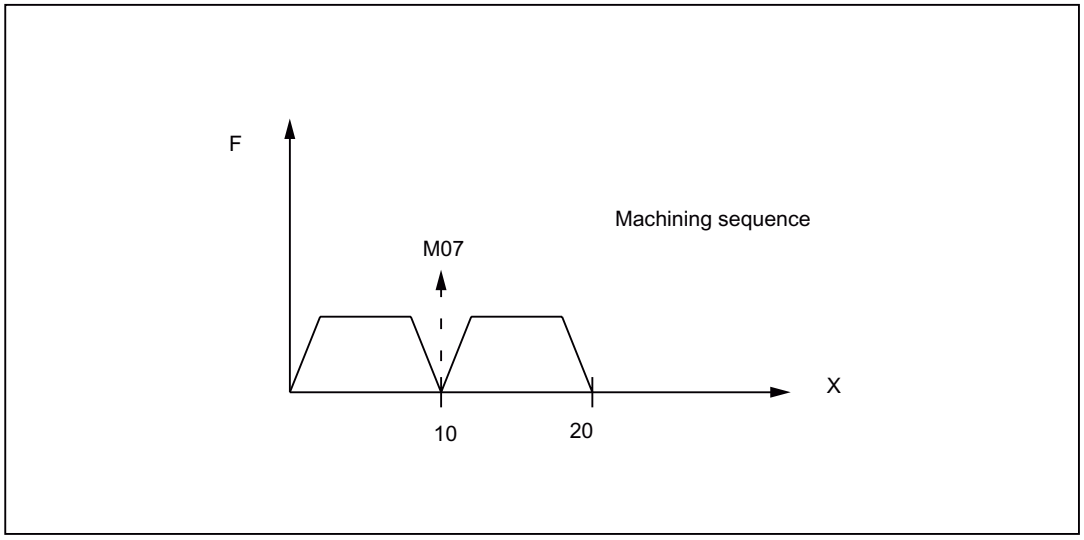
/FB1/ Function Manual Basic Functions; Auxiliary Function Output to PLC (H2)

### Examples

The advantage of implementing auxiliary function outputs in synchronized actions is illustrated by the following example: Switch on coolant at a specific position

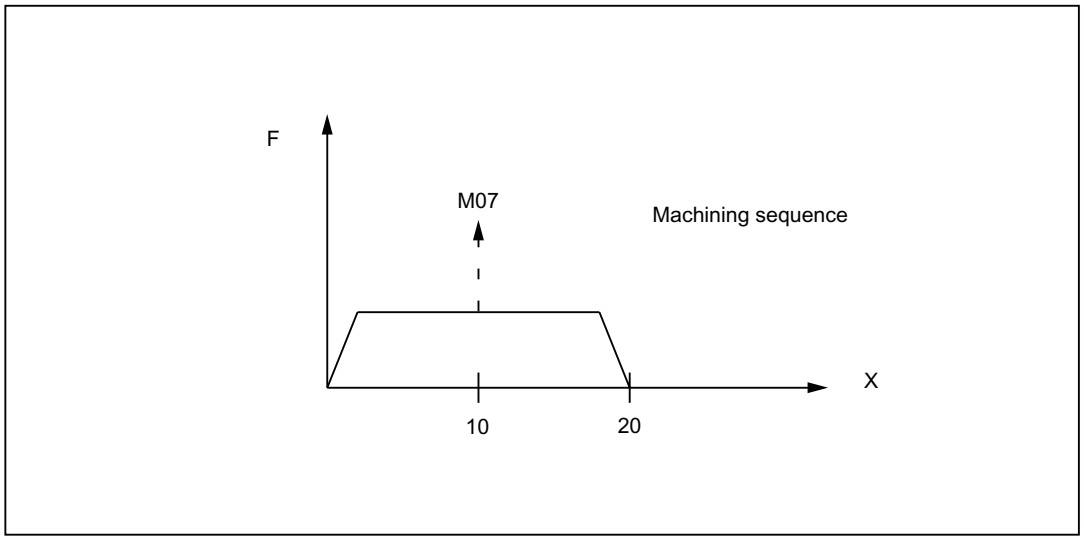
Solution **without** synchronized action: 3 blocks

```
N10 G1 X10 F150
N20 M07
N30 X20
```




Solution **with** synchronized action: 1 set

```
N10 WHEN $AA_IM[X] >= 10 DO M07
N20 G1 X20 F150
```




## Auxiliary function output to the PLC

M, S or H auxiliary functions can be output to the PLC as synchronized actions. The output takes place **immediately** (like an interrupt on the PLC) in the interpolation cycle if the condition is fulfilled.

Output time becomes ineffective, if possibly defined in one of the following machine data:

- MD11110 $MN_AUXFU_GROUP_SPEC (auxiliary function group specification)
- MD22200 $MC_AUXFU_M_SYNC_TYPE (output time of M functions)
- MD22210 $MC_AUXFU_S_SYNC_TYPE (Output time of the S functions)
- MD22230 $MC_AUXFU_H_SYNC_TYPE (Output time of the H functions)

## Programming

Auxiliary functions may be programmed with only frequency keywords WHEN or EVERY in synchronized actions.

## Example

```
WHEN $AA_IM[X] > 50 DO H15 S3000 M03
; if actual value of the X axis is greater than 50, the output is H15, set new spindle speed,
new direction of rotation
```

## Restrictions

No more than 10 auxiliary functions may be output simultaneously (i.e. in an OB 40 cycle of the PLC). The total number of auxiliary function outputs from part programs and synchronized actions must never exceed 10 at any point in time.

Maximum number of auxiliary functions per synchronized action block or technology cycle block:

- 5 M functions
- 3 S functions
- 3 H functions

Predefined M functions cannot be programmed by means of synchronized actions. They will be rejected by an alarm.

WHEN ... DO M0 ;Alarm

However, the spindle-M-functions are allowed: M3, M4, M5 and M17 may be programmed as the end of a technology cycle.

## Message acknowledgment

Technology cycle blocks (see Chapter "Call of technology cycles") containing auxiliary function outputs are not completely processed until all auxiliary functions in the block have been acknowledged by the PLC. The next block in the technology cycle is not processed until all auxiliary functions in the preceding block have been acknowledged by the PLC.

The acknowledgement behavior has been expanded to include other variants:

- Output of auxiliary functions without block change delay

  High-speed auxiliary functions (QUICK) first, as a parallel process in the PLC, then auxiliary function output with anticipated acknowledgment.

The user can choose between `INT` and `REAL` as the data type for H auxiliary functions. The PLC user program must interpret the values in accordance with the definition. The `INT` value range for H auxiliary functions has been increased to: -2 147 483 648 to 2 147 483 647.

**References:**

/FB1/ Function Manual Basic Functions; Auxiliary Function Output to PLC (H2)

## 2.4.2 Setting (writing) and reading of real-time variables

### Write

In synchronized actions the real-time variables can be **written** in actions, which are marked in the list of the system variables in the 8th row in the field "write:" with an X. The following machine and setting data are also written in the main run:

- Machine and setting data, e.g. $$MN_..., $$MC_..., $$MA_...

  or $$SN_..., $$SC_..., $$SA_...

**Note**

Machine data and setting data that are to be written in the main run must be programmed with **$$**.._... .

### Efficiency

Machine data written from synchronized actions must be coded for IMMEDIATE effectiveness. The modified value will not otherwise be available for the remainder of the processing run. Details about the effectiveness of new machine data values after modification can be found in:

**References:**

/LIS/ Lists (Book1)

**Examples:**

... DO **$$MN_MD_FILE_STYLE = 3**;Set machine data

... DO **$$SA_OSCILL_REVERSE_POS1 = 10**;Set setting data

... DO **$A_OUT[1]=1**;Set digital output

... DO **$A_OUTA[1]= 25**; Output analog value

### Read

In synchronized actions, the system variables can be read as main run variables in actions that are labeled with X in the "read" field in the 7th line of the list of synchronous variables. The following machine and setting data are also read:

- Machine data, setting data, e.g. $$SN_..., $$SC_..., $$SA_...

**Note**

Machine and setting data that must be addressed online in the main run must be programmed with **$$**.._... . In the case of variables whose content remains unchanged during the main run, it is sufficient to add a $ sign in front of the identifier.

**Examples:**

WHEN **$AC_DTEB** < 5 DO ... ;Read distance from block end in condition

DO $R5= **$A_INA[2]** ;Read value of the analog input 2 and assign arithmetic variable

## 2.4.3 Changing of SW cam positions and times (setting data)

### Introduction

The "Software cams" function allows position-dependent cam signals to be sent to the PLC or NCK I/Os.

**References:**

/FB2/ Function Manual for Extended Functions, Software Cams, Position-Switching Signals (N3)

### Function

Synchronized actions can be programmed to alter cam positions at which signal outputs are set. Existing setting data are written to change these positions.

The following setting data can be modified via synchronized actions:

$$SN_SW_CAM_MINUS_POS_TAB_1[0..7] ;Positions of the minus cams

$$SN_SW_CAM_MINUS_POS_TAB_2[0..7] ;Positions of the minus cams

$$SN_SW_CAM_PLUS_POS_TAB_1[0..7] ;Positions of the plus cams

$$SN_SW_CAM_PLUS_POS_TAB_2[0..7] ;Positions of the plus cams

### Example 1

Alteration of a cam position:

ID=1 WHEN $AA_IW[x] > 0 DO $$SN_SW_CAM_MINUS_POS_TAB_1[0] = 50.0

Lead or delay times can be changed via the following setting data:

$$SN_SW_CAM_MINUS_TIME_TAB_1[0..7]

; Lead or delay time on minus cams

$$SN_SW_CAM_MINUS_TIME_TAB_2[0..7]

; Lead or delay time on minus cams

$$SN_SW_CAM_PLUS_TIME_TAB_1[0..7]

; Lead or delay time on plus cams

$$SN_SW_CAM_PLUS_TIME_TAB_2[0..7]

; Lead or delay time on plus cams

### Example 2

Alteration of a lead/delay time:

ID=1 WHEN $AA_IW[x] > 0

DO $$SN_SW_CAM_MINUS_TIME_TAB_1[0] = 1.0

**Note**

Software cams must not be set as a function of velocity via synchronized actions immediately in front of a cam. At least 2-3 interpolation cycles must be available between the setting and the relevant cam position.

## 2.4.4 FCTDEF

### Application

The actions of online tool offset FTOC and polynomial evaluation SYNFCT described in the subsections given below require an interrelationship between an input quantity and an output quantity to be defined in the form of an polynomial. FCTDEF defines polynomials of this type.

For special examples of polynomial application for online dressing of a grinding wheel, please see Subsection "Online tool offset FTOC". For examples of load-dependent feedrates and clearance control via polynomials, please see Subsection "Polynonmial evaluation SYNFCT".

### Characteristics of polynomials

The polynomials defined by means of FCTDEF have the following characteristics:

- They are generated through a FCTDEF call in the part program.
- The parameters of defined polynomials are real-time variables.
- Individual polynomial parameters can be overwritten using the same method used to write real-time variables. Permissible generally in part program and in action section of synchronized actions. See Chapter "Setting (writing) and reading of real-time variables".

**Note**

Validity limits and coefficients of existing polynomials can also be changed from synchronized actions.

Example WHEN ... DO $AC_FCT1[1]= 0.5

### Number of polynomials

The number of polynomials that can be defined simultaneously is specified by the following machine data:

MD28252 $MC_MM_NUM_FCTDEF_ELEMENTS (number of FCTDEF elements)

### Block-synchronous polynomial definition

**FCTDEF**(
Polynomial No.
Low limit,
High limit,
a0,
a1,
a2,
a3)

The relationship between the output variable y and the input variable x is as follows:

$y= a_0+ a_1x+ a_2x^2+ a_3x^3$

The parameters specified in the function are stored in the following system variables:

| | |
|---|---|
| $AC_FCTLL[n]: | Lower limit, n: Polynomial number |
| $AC_FCTUL[n]: | Upper limit, n: Polynomial number |
| $AC_FCT0[n]: | a0-coefficient, n: Polynomial number |
| $AC_FCT1[n]: | a1-coefficient, n: Polynomial number |
| $AC_FCT2[n]: | a2-coefficient, n: Polynomial number |
| $AC_FCT3[n]: | a3-coefficient, n: Polynomial number |

On the basis of this relationship, it is also possible to write or modify polynomials directly via the relevant system variables. The validity range of a polynomial is defined via limits $AC_FCTLL[n] and $AC_FCTUL[n].

### Call of polynomial evaluation

Stored polynomials can be used in conjunction with the following functions:

- Online tool offset, `FTOC()`
- Polynomial evaluation, `SYNFCT()`.

**References:**

/PG/ Programming Manual Fundamentals

/PGA/Programming Manual Advanced

/FB2/ Function Manual Extended Functions; Grinding (W4).

## 2.4.5 Polynomial evaluation SYNFCT

### Application

By applying an evaluation function in the action section of a synchronized action, it is possible to read a variable, evaluate it with a polynomial and write the result to another variable in synchronism with the machining process. This functionality can be used, for example, to perform the following tasks:

- Feedrate as a function of drive load
- Position as a function of a sensor signal
- Laser power as a function of path velocity

  ...

### SYNFCT() evaluation function

The function has the following parameters:

**SYNFCT**( Polynomial number,
Real-time variable output,
Real-time variable input)

For definition of a polynomial, please see Chapter "FCTDEF".

### Operating principle of SYNFCT

The polynomial identified by "Polynomial number" is evaluated with the value of the "Real-time variable input". The result is then limited by maximum and minimum limits and assigned to the "Real-time variable output".

**Example**

```
FCTDEF(1,0,100,0,0.8,0,0)                    ;  Definition of polynomial 1 is done
...
```

**Synchronized action:**

```
ID=1 DO SYNFCT(1,$AA_VC[U1], $A_INA[2])      ;  the additive offset value of the axis U1 is
                                                calculated in each interpolation cycle from
                                                the analog input value 2 via polynomial 1
```

For the 'Real-time variable output', it is possible to select variables that are included in the machining process as follows:

- as an additive control factor (e.g. feedrate),
- as a multiplicative control factor (e.g. override),
- as a position offset
- directly

## Additive feedrate control

In case of additive control, the programmed value (F word with respect to Adaptive Control) is compensated by an **additive factor**. $F_{active} = F_{programmed} + F_{AC}$

The following are set, for instance, as 'real-time variable output':

| | |
|---|---|
| $AC_VC | additive path-feed offset, |
| $AA_VC[axis] | additive axial feedrate override |

## Example of additive control of path feedrate

The programmed feedrate (axial- or path-related) must be subject to **additive** control by the (positive) X axis current (e.g. infeed torque). The operating point is set to 5 A. The feedrate may be altered by ±100 mm/min. The magnitude of the axial current deviation may be ±1 A.

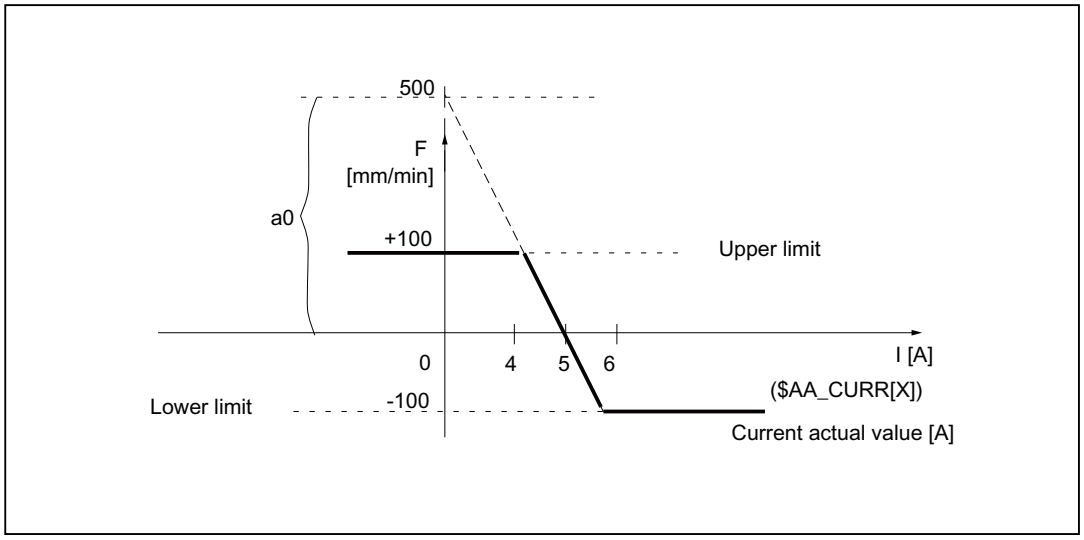



Figure 2-5 Example of additive control

Determination of the coefficients, see also Chapter "FCTDEF":

$y = f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$

$a_1 = 100mm / (1min*A)$

$a_1 = -100 \rightarrow$ Control constant

$a_0 = -(-100)*5 = 500$

$a_2 = 0$ (not a square component)

$a_3 = 0$ (not a cubic component)

Upper limit = 100

Lower limit = -100

The polynomial to be defined (no. 1) is thus as follows:

```
FCTDEF(1, -100, 100, 500, -100, 0, 0)
```

This function completely describes the Figure "Example of additive control".

The *adaptive control* function is activated with the following synchronized action:

```
ID = 1 DO SYNFCT(1, $AC_VC[x],   ;
$AA_LOAD[x])
```

the additive offset value for the feedrate of the x axis is calculated in each interpolation cycle from the percentage utilization value of the drive via the polynomial 1

## Multiplicative control

In case of the multiplicative control, the F Word is **multiplied** with a factor (override in case of adaptive control). $F_{active} = F_{programmed} _ Factor_{AC}$

Variable $AC_OVR that acts as a *multiplicative* factor on the machining process is used as the real-time variable output.

## Example of multiplicative control

The programmed feedrate (axial- or path-related) must be subject to **multiplicative** control as a function of the drive load. The operating point is set to 100% at 30% drive load. The axis(axes) must stop at 80% drive load. An excessive velocity corresponding to the programmed value +20% is permissible.

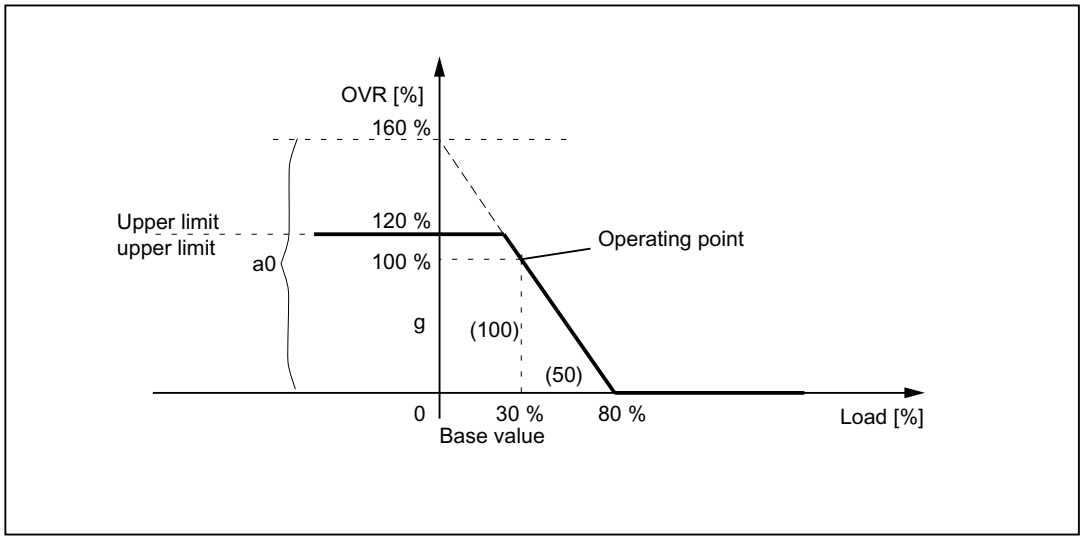



Figure 2-6 Example of multiplicative control

Determination of the coefficients, see also Chapter "FCTDEF":

$y = f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$

a1 = - 100% / (80% -30%) = -2

a0 = 100+ (2 * 30) = 160

a2 = 0 (not a square component)

a3 = 0 (not a square component)

Upper limit = 120

Lower limit = 0

The polynomial (no. 2) can therefore be defined as follows:

```
FCTDEF(2, 0, 120, 160, -2, 0, 0)
```

This function completely describes the Figure "Example of multiplicative control".

The associated synchronized action can be programmed as follows:

```
ID = 1 DO SYNFCT(2, $AC_OVR,                ;  the path override is calculated in each interpolation cycle
$AA_LOAD[x])                                   from the percentage drive load for the x axis via the
                                               polynomial 2
```

## Position offset with limitation

The system variable $AA_OFF controls an axis-specific override that takes immediate effect (basic coordinate system). The type of override is defined by the machine data:

MD36750 $MA_AA_OFF_MODE (Effect of value assignment for axial override in case of synchronized actions)

0: Proportional evaluation

1: integral evaluation

In SW 4 and later, it is possible to limit the value to be compensated absolutely (main-run variable output) to the value stored in the setting data:

SD43350 $SA_AA_OFF_LIMIT (Upper limit of the offset value $AA_OFF in case of clearance control)

Axis-specific system variables can be evaluated in another synchronized action to establish whether the limitation has been reached.

$AA_OFF_LIMIT[axis]

Value -1: Limit of offset value reached in the negative direction.

Value 1: Limit of offset value reached in the positive direction.

Value 0: The offset value is not within the limit range.

### Application:

The SYNFCT function can be used in conjunction with system variable $AA_OFF to implement clearance control in laser machining operations. See below.

## Example

### Task:

**Clearance control** as function of a sensor signal in case of laser machining.

The offset value is limited in the negative Z direction to ensure that the laser head is retracted reliably from finished metal blanks. When the limiting value is reached, the user can trigger the reactions such as Stop axis (by means of override 0, see Chapter "Disabling a programmed axis motion") or set alarm (see Chapter "Set alarms / error reactions").

**Boundary conditions:**

*Integral* evaluation of the input quantity of sensor $A_INA[3]. The offset value is applied in the basic coordinate system, i.e. prior to kinematic transformation. A programmed frame (TOFRAME) has no effect, i.e. the function cannot be used for 3D clearance control in the direction of orientation. The "clearance control" function can be used to implement a clearance control system with high dynamic response or a 3D clearance control system. See:

**References:**

/FB3/ Function Manual Special Functions; Clearance Control (TE1)

/PG/ Programming Manual Fundamentals

The interdependency between input quantity and output quantity is assured through the relationship illustrated in the following diagram.

## Further examples

Please see section "Clearance control with variable upper limit" for an example illustrating dynamic adaptation of a polynomial limit in conjunction with Adaptive Control (clearance control). The chapter "Controlling the feedrate" contains an example of adaptive control of the path feedrate.

## Clearance control

The clearance value is calculated by integrating with the machine data:

MD36750 $MA_AA_OFF_MODE[V]=1 (Effect of value assignment for axial override in case of synchronized actions)

It works in the basic coordinate system, i.e. before transformation. This means that it can be used for clearance control in the orientation direction (after frame selection with TOFRAME).

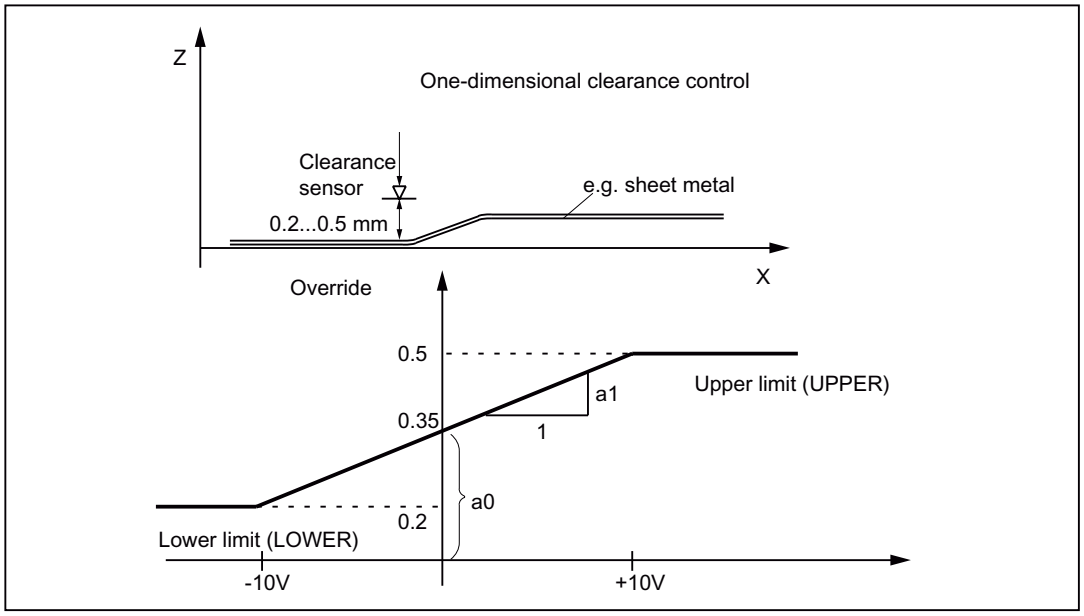



Figure 2-7 Clearance control

```
%_N_AON_SPF
PROC AON                                          ;  Subroutine for clearance control ON
FCTDEF(1, 0.2, 0.5, 0.35, 1.5 EX-5)               ;  Polynomial definition: The offset is applied in
                                                     the range 0.2 to 0.5
ID = 1 DO SYNFCT(1,$AA_OFF[Z], $A_INA[3])         ;  Clearance control active
ID = 2 WHENEVER $AA_OFF_LIMIT[Z]<>0 DO            ;  Disable axis X when limit value is overshot
$AA_OVR[X] = 0
RET
ENDPROC

%_N_AOFF_SPF
PROC AOFF                                         ;  Subroutine for clearance control OFF
CANCEL(1)                                         ;  Cancel clearance control synchronized action
CANCEL(2)                                         ;  Cancel limit range check
RET
ENDPROC

%_N_MAIN_MPF                                      ;  Main program
                                                  ;  MD36750 is set to 1 before Power On for
                                                     integrating machining
$SA_AA_OFF_LIMIT[Z]= 1                            ;  Limiting value for the offset:
AON                                               ;  Clearance control ON
...
G1 X100 F1000
AOFF                                              ;  Clearance control OFF
M30
```

## 2.4.6 Overlaid movements $AA_OFF settable (SW 6 and later)

### Overlaid movements up to SW 5.3

Whatever the current tool and processing level, an overlaid movement is possible for each axis of the channel via the system variable $AA_OFF. The offset is retracted immediately, whether the axis is programmed or not. This allows a clearance control to be implemented.

The type of calculation is defined with the axial machine data as follows:

MD36750 $MA_AA_OFF_MODE (Effect of value assignment for axial override in case of synchronized actions)

Bit0 = 0: proportional calculation (absolute value)

Bit0 = 1: integrating calculation (incremental value)

$AC_VACTB and $AC_VACTW as input variable for synchronized actions and output are disabled via the options bit ("Feedrate-dependent analog value control" → laser power control)!

$AA_OFF, position offset as output variable for synchronized actions for clearance control is disabled via the options bit!

Velocity control with the machine data:

MD32070 $MA_CORR_VELO (axis velocity for override)

### Response of $AA_OFF in SW 6 and later

**After** RESET the position offset can still be retained

Previously, during a RESET the position offset of $AA_OFF was deselected. In the case of static synchronized actions IDS = <Number> DO $AA_OFF = <Value>, this behavior leads to an immediate renewed overlaid movement with the interpolation of a position offset, the behavior of RESET can be set with the following machine data:

MD36750 $MA_AA_OFF_MODE (Effect of value assignment for axial override in case of synchronized actions)

Bit1 = 0: $AA_OFF is deselected in case of RESET

Bit1 = 1: $AA_OFF iis maintained after RESET

### Overlaid movement in the mode JOG

Also in JOG mode, if $AA_OFF changes, an interpolation of the position offset can be set as an overlaid movement via the following machine data:

MD36750 $MA_AA_OFF_MODE (Effect of value assignment for axial override in case of synchronized actions)

Bit2 = 0: no overlaid movement owing to $AA_OFF

Bit2 = 1: an overlaid movement owing to $AA_OFF

If a position offset is interpolated on the basis of $AA_OFF, a mode change can only occur after JOG once the interpolation of the position offset is complete. Otherwise, alarm 16907 is signaled.

### Activation/Deactivation

The programmed conditions of the current motion-synchronous actions are recorded in interpolation time, until the conditions are met or the end of the subsequent block is reached with the machine in operation.

In software version 3.2 and later, the introduction of an $$ main variable approved for synchronized actions results in a comparison of the synchronization conditions in interpolation time in the main run.

## Boundary conditions

- **Interrupt routines/asynchronous subroutines**

  When an interrupt routine is activated, modal motion-synchronous actions are retained and are also effective in the asynchronous subroutine. If the subroutine return is not made with REPOS, the modal synchronized actions changed in the asynchronous subroutine continue to be effective in the main program.

- REPOS

  In the remainder of the block, the synchronized actions are treated in the same way as in an interruption block. Modifications to modal synchronized actions in the asynchronous subprogram are not effective in the interrupted program. Polynomial coefficients programmed with FCTDEF are not affected by ASUB and REPOS.

  The coefficients from the call program are applied in the asynchronous subprogram. The coefficients from the asynchronous subprogram continue to be applied in the call program.

- **End of program**

  Polynomial coefficients programmed with FCTDEF remain active after the end of program.

- **Block search**

  During block search with calculation, these polynomial coefficients are collected, i.e. written to the setting data.

## CORROF

- The part program command CORROF with DRFOF is also collected during block search and output in an action block. In the last block handled by the search run with CORROF or DROF, all the deselected DRF offsets are collected for reasons of compatibility.

  A CORROF with AA_OFF is not collected during a block search and is lost. If a user wishes to continue to use this search run, this is possible by means of block search via "SERUPRO" program testing. More detailed information about these block searches can be found in:

**References:**

/FB1/ Function Manual, Basic Functions; Mode Group, Channel, Program Operation Mode (K1),

- **DRF-axis-specific offsets with**CORROF **deselected**

  With CORROF the DRF-offsets are possible for the individual axes only from the part program.

- **Deselection of the position offset in case of synchronized actions**

  Alarm 21660 is output if a synchronized action is active when the position offset is deselected via parts program command CORROF(axis,"AA_OFF"). $AA_OFF is deselected simultaneously and not set again. If the synchronized action becomes active later in the block after CORROF, $AA_OFF remains set and a position offset is interpolated.

**References:**

/PG/ Programming Manual Fundamentals

**Note**

The coordinate system (BCS or WCS) in which a real-time variable is defined determines whether frames will or will not be included.

Distances are always calculated in the set basic system (metric or inch). A change with `G70` or `G71` has no effect.

`DRF` offsets, zero offsets external, etc., are only taken into consideration in the case of real-time variables that are defined in the machine coordinate system.

## 2.4.7 Online tool offset FTOC

### Online tool offset

Machining of the workpiece and dressing of the grinding wheel for grinding applications can be implemented either in the same or in different channels (machining and dressing channel).

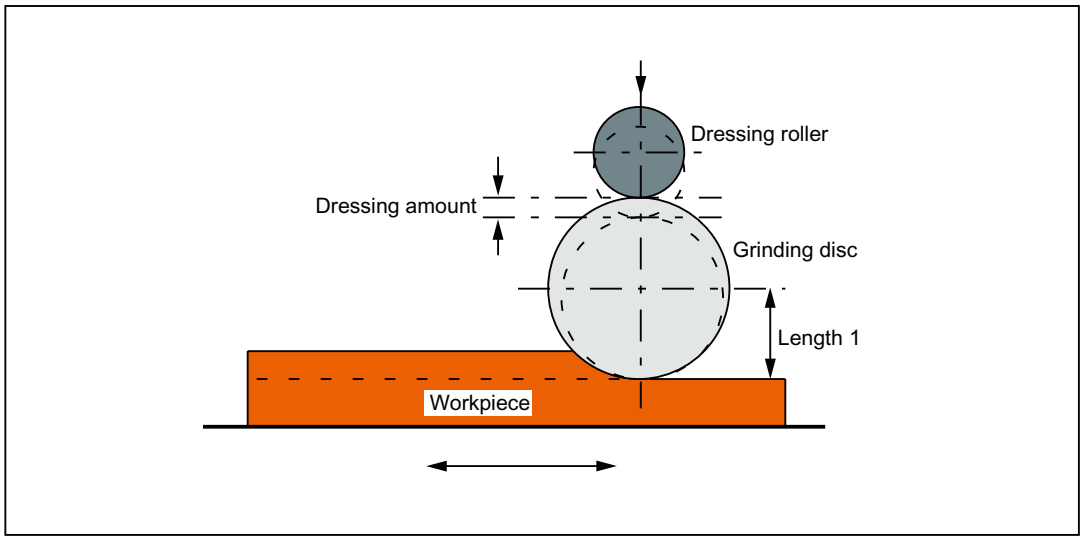



Figure 2-8 Dressing during machining using a dressing roller

**References:**

/FB2/ Function Manual Extended Functions; Grinding (W4).

## Boundary condition

The synchronized action FTOC is available in Software-version 3.2 and later.

An online offset allows an overlaid movement to be implemented for a geometry axis according to a polynomial programmed with FCTDEF (see Subsection "FCTDEF") as a function of a reference value (e.g. actual value of an axis).

The coefficient $a_0$ of the function definition FCTDEF() is evaluated for FTOC. Upper and lower limit are dependent on $a_0$.

## Programming of FTOC

The online offset is specified as follows:

```
FTOC      (Polynomial No.
          Real-main variable_read,   ;   Ref_value
          Length 1_2_3,
          Channel number,
          Spindle number)
```

## Parameter

| | |
|---|---|
| Polynomial No.: | Number of the function parameterized earlier with FCTDEF. |
| Real-main variable_read: | All main variables of the type REAL listed in the chapter "List of system variables relevant for synchronized actions" are permitted. |
| Length 1_2_3: | Wear parameter into which the tool offset value is **added**. |
| Channel number: | Target channel in which the offset must be applied. This enables simultaneous dressing from a parallel channel. If the channel number is missing, the offset is effective in the active channel. Online offset with FTOCON must be activated in the target channel. |
| Spindle number: | The spindle number is programmed if a non-active grinding wheel needs to be dressed. "Constant peripheral speed" or "tool monitoring" must be active for this purpose. If no spindle number is programmed, then the active tool is compensated. |

### Example

Compensate length of an active grinding wheel

```
%_N_DRESS_MPF
FCTDEF(1,-1000,1000,-$AA_IW[V],1)   ;  Definition of the function
ID=1 DO FTOC(1,$AA_IW[V],3,1)       ;  Select online tool offset:
                                       Derived from the motion of the V axis, the length 3 of
                                       the active grinding wheel is compensated in channel 1.
WAITM(1,1,2)                        ;  Synchronization with machining channel
G1 V-0.05 F0.01, G91
G1 V -...
...
CANCEL(1)                           ;  Deselect online offset
...
```

**Note**

No frequency keyword nor any condition is programmed in the synchronized action. The FTOC action is therefore active in every interpolation cycle with no dependencies other than $AA_IW[V].

## 2.4.8 Online tool length offset $AA_TOFF[Index]

### Function

In conjunction with an active orientation transformer or an active tool carrier, tool length offsets can be applied during processing/machining in real time. Changing the effective tool length using online tool length offset produces changes in the compensatory movements of the axes involved in the transformation in the event of changes in orientation. The resulting velocities can be higher or lower depending on machine kinematics and the current axis position.

The **velocity** at which the tool length offset is applied in the appropriate direction via $AA_TOFF[ ], can be set using machine data:

MD21194 $MC_TOFF_VELO (speed online tool offset)

The acceleration can accordingly be modified via machine data:

MD21196 $MC_TOFF_ACCEL (acceleration online tool offset)

**Note**

Online tool length offset is an option and must be enabled beforehand.

For further information regarding the activation of the function in the part program, see:

**References:**

/PGA/Programming Manual Work Preparation; Chapter Transformations "TOFFON, TOFFOF"

## Applications in synchronized actions

The tool length offsets are included via a synchronized action variable $AA_TOFF[ ]. This variable is 3 dimensional corresponding to the three tool axes.

The three geometrical axis labels (tags) X, Y, Z are used as index. This defines the number of active directions of compensation by the geometry axes active at the same time. All offsets can be active at the same time.

For an active orientation transformation or for an active tool carrier that can be oriented, these offsets are effective in the particular tool axes. Before switching on or switching off a transformation, an overlaid motion must be switched out using `TOFFOF()`.

Once the "online tool length offset" has been deselected for a tool direction, the value of system variable $AA_TOFF[ ] or $AA_TOFF_VAL[ ] is zero for this tool direction.

## Mode of operation of the offset in the tool axis

The tool length offsets do not change the tool parameters, but are taken into account within the transformation or the tool carrier that can be orientated, so that offsets are obtained in the tool coordinate system. Machine data MD 21190 MC_TOFF_MODE (mode of operation of offset in the tool axis) and bits 1 to 3 can be used to specify whether the contents of the three components of the synchronized action variable $AA_TOFF[ ]

- Bits 1 to 3 = 0: are to be applied as an **absolute value**, or whether
- Bits 1 to 3 = 1: **an integrating behavior** is to occur.

The integrating behavior of $AA_TOFF[ ] allows 3D remote control. The value that has been reached, integrating, can be read using variable $AA_TOFF_VAL[ ].

## Example

**Selecting** the online tool length offset

Setting the machine data for online tool length offset:

```
MD21190 $MC_TOFF_MODE = 1                ;  Absolute values are approached
MD21194 $MC_TOFF_VEL[0] = 10000
MD21194 $MC_TOFF_VEL[1] = 10000
MD21194 $MC_TOFF_VEL[2] = 10000
MD21196 $MC_TOFF_ACC[0] = 1
MD21196 $MC_TOFF_ACC[1] = 1
MD21196 $MC_TOFF_ACC[2] = 1
```

Activate online tool length offsets in the part program:

```
N5 DEF REAL XOFFSET
N10 TRAORI                               ;  Activate orientation transformation
N20 TOFFON(Z)                            ;  Activating the function for the
                                         ;  Z-tool axis


Query in synchronized action             ;  for the Z-tool axis a
N30 WHEN TRUE DO $AA_TOFF[Z] = 10        ;  Tool length offset = 10
G4 F5                                    ;  interpolated
…
Static synchronized action               ;  for the Z-tool axis a
N50 ID=1 DO $AA_TOFF[X] = $AA_IW[X2]     ;  offset as a function of the
G4 F5                                    ;  position of one axis X2 is executed
…                                        ;  current offset
N100 XOFFSET = $AA_TOFF_VAL[X]           ;  points tool in X direction
N120 TOFFON(X, -XOFFSET)                 ;  for the X-tool axis a
G4 F5                                    ;  tool length offset is reset again to 0
```

## Example

**De-selecting** the online tool length offset

```
N10 TRAORI                              ;  Activate orientation transformation
N20 TOFFON(X)                           ;  Activating the function for the
N30 WHEN TRUE DO $AA_TOFF[X] = 10       ;  X-tool axis a
G4 F5                                   ;  Tool length offset = 10 is interpolated
…
N80 TOFFOF(X)                           ;  Positional offset of the X tool direction is deleted:
                                        ;  ... $AA_TOFF[X] = 0 no axis is traveled,
                                        ;  for the current position in WKS the position offset
                                        ;  is calculated according to the current orientation.
N90 TRAFOOF
```

## Activating and de-activating in the part program

The online tool length offset is activated in the part program with TOFFON( ) and deactivated with TOFFOF( ). When activating for the particular offset direction, an offset value can be specified, e.g. TOFFON(Z, 25), which is then immediately moved through.

During a compensatory movement, the VDI signal NCK → PLC NST "TOFF- motion active" is set to 1:

DB21, ... DBX318.3 (TOFF movement active)

After de-selection, the NST "TOFF active" is set to 0:

DB21, ... DBX318.2 (TOFF active).

**Note**

The online tool length offset remains inactive until it is re-selected using the instruction TOFFON( ) in the part program.

## Tool length offset at RESET and POWER ON

The behavior in case of RESET with Bit 0 can be set using the machine data:

MD21190 $MC_TOFF_MODE (operation of tool offset in tool axis)

The tool length offset $AA_TOFF[ ] is for MD 21190

- Bit 0 = 0: either deselected, or else is
- Bit 0 = 1: retained after RESET.

This is always necessary in case of synchronized actions IDS=number DO $AA_TOFF[n]= value, as otherwise there would be an immediate tool length offset.

Similarly, a transformation or a die carrier with orientation capability can be deselected **after**RESET via the machine data:

MD20110 $MC_RESET_MODE_MASK (definition of initial control system settings after RESET/TP-End)

Also in this case, the tool length offset must be cleared.

If a tool length offset is to remain active extending beyond a RESET, and a transformation change of the tool carrier that can be oriented takes place, then Alarm 21665 "channel %1 $AA_TOFF[ ] reset" is output. In this case, the tool length offset is set to 0.

After POWER ON, all tool length offsets are set to 0.0.

The function is de-activated after POWER ON.

### Behavior for the operating mode change and REPOS

The tool length offset remains active even when the operating mode is changed. The offset can be executed in all modes - with the exception of JOG and REF.

If a tool length offset is interpolated on account of $AA_TOFF[ ] during mode change, the mode change cannot take place until the interpolation of the tool length offset has been completed. Alarm 16907 "Channel %1 action %2 <ALNX> possible only in stop state" is output.

The tool length offset is also active during REPOS.

### Boundary conditions

For an existing tool length offset, to avoid Alarm 21670 "Channel %1 block %2 inadmissible change to the tool direction because $AA_TOFF active", then the following should be observed:

- Deactivate transformation TROFOOF

  A tool length offset must be cancelled **before** a transformation in the part program with TOFFOF( ).

- Change CP after PTP and PTP-travel in the mode JOG

  When changing over from CP to PTP a transformation is deactivated. A tool length offset must be cleared **before** the changeover. If, when changing over to axis-specific manual traverse in the JOG mode, a tool length offset is active, then a change to PTP is not executed and alarm 21670 is issued. CP remains active until the tool length offset was cleared using TOFFOF.

- Geometry axis change and change of plane

  In case of geo-axis replacement, a tool length offset in the direction of the geometry axis must be cleared beforehand via TOFFOF( ).

  If a tool length offset is available when a plane is changed, then this must be cleared beforehand via TOFFOF( ).

- Block search

  The instructions TOFFON( ) and TOFFOF( ) are not collected and output in an action block during block search.

## 2.4.9 RDISABLE

### Programmed read-in disable RDISABLE

The RDISABLE command in the active section causes block processing to be stopped when the relevant condition is fulfilled. Processing of programmed motion-synchronous actions still continues. The read-in disable is canceled again as soon as the condition for the RDISABLE is no longer fulfilled.

An exact stop is initiated at the end of the block containing RDISABLE irrespective of whether or not the read-in disable is still active.

The exact stop is also triggered, when there is control of the continuous-path mode (G64, G641, G642, G643, G644). RDISABLE can be programmed with reference to the block or also modal (ID=, IDS=)!

**Application:** This method can be used, for instance, to start the program in the interpolation cycle as a function of external inputs.

### Example RDISABLE

Programmed read-in disable

```
WHENEVER $A_INA[2]<7000 DO RDISABLE
...
N10 G01 X10                                    ; RDISABLE acts at the end of N10, when
                                                 the condition is fulfilled during its
                                                 processing.
N20 Y20
```

Program processing is halted if the voltage at input 2 drops to below 7 V (assuming that the value 1000 corresponds to 1 V).

Application of this solution, for instance, read-in disable till the obstacle is removed.

## 2.4.10 STOPREOF

### End of preprocessing stop with STOPREOF

A motion-synchronous action containing an STOPREOF command cancels the existing preprocessing stop if the condition is fulfilled.

STOPREOF must always be programmed with keyword `WHEN' and as a non-modal command.

**Application:** Fast program branching at the end of a block.

### Example STOPREOF

Program branches

```
WHEN $AC_DTEB<5 DO STOPREOF
G01 X100
IF $A_INA[7]>5000 GOTOF Label 1
```

If the distance to the end of the block is less than 5 mm, end preprocessing stop. If the voltage at input 7 drops below 5V, jump forwards to label 1 (assuming that the value 1000 corresponds to 1 V).

## 2.4.11 DELDTG

### Deletion of distance-to-go

Synchronized actions can be used to activate deletion of distance-to-go for the **path** and for specified **axes** as a function of a condition.

- High-speed prepared deletion of distance-to-go

### High-speed, prepared DDTG

High-speed/prepared deletion of distance-to-go is used in time-critical applications:

- if the time between deletion of distance-to-go and start of next block needs to be very short,
- if there is a high probability that deletion of distance-to-go will be activated.

### DELDTG

Deletion of distance-to-go is programmed with synchronized action `DELDTG`.

After the distance-to-go has been deleted, the remaining path distance is stored in the system variable $AC_DELT. Continuous-path mode is thus interrupted at the end of the block with high-speed deletion of distance-to-go.

**Constraints:**

Deletion of distance-to-go for the path may only be programmed as a non-modal synchronized action.

If tool radius compensation is active, fast deletion of distance-to-go cannot be used.

**Commands:**

`MOVE`=1:Works on indexing axes with and without Hirth tooth system MOV=0: Same function for both: approaches the next position. Command: `DELDTG`: In the case of indexing axes without Hirth tooth system: Axis stops immediately. In the case of indexing axes with Hirth tooth system: Axis traverses to next position.

### Example DELDTG

```
... DO DELDTG
N100 G01 X100 Y100 F1000
N110 G01 X...
IF $AC_DELT > 50
...
```

### High-speed, prepared DDTG for axes

High-speed, prepared deletion of distance-to-go for axes must be programmed as a non-modal action.

**Application:**

A positioning motion programmed in the part program is halted by means of axial deletion of distance-to-go. Several axes can be stopped simultaneously with one command.

```
... DO DELDTG(axis1, axis2, ...)
```

### Examples of DELDTG(axis)

```
WHEN $A_INA[2]>8000 DO DELDTG(X1)          ; if the voltage at input 2
                                           ; falls below 8 V, distance to go
                                           ; deletion for axis X1
POS[X1] = 100                              ; Next position
R10 = $AA_DELT[X 1]                        ; Import axial distance-to-go in R10
```

After successful deletion of distance-to-go the variable $AA_DELT[axis] contains the axial distance to go.

(Assumption: the value 1000 corresponds to 1 V).

## 2.4.12 Disabling a programmed axis motion

### Task

The axis is programmed within a machining routine and, in particular circumstances, must not be started at the beginning of a block.

### Solution

For each synchronized action the override is maintained at 0 until the starting instant.

**Example**

```
WHENEVER $A_IN[1]==0 DO $AA_OVR[W]=0
G01 X10 Y25 F750 POS[W]=1500 FA[W]=1000    ; The positioning axis is started
                                           ; asynchronous
                                           ; to the path processing;
                                           ; the clearance is done via a digital input
```

**Note**

Axis motion disable can also be programmed for PLC axes (e.g. magazine axis).

## 2.4.13 Starting command axes

### Introduction

Axes can be positioned, started and stopped completely asynchronously to the part program from synchronized actions. This type of programming is advisable for cyclic sequences or sequences that are strongly dependent on events. Axes started from synchronized actions are called **command axes.**

### Control from the PLC

**Autonomous singleaxis operations**

A command axis interpolated from the main run (started by static synchronized actions) reacts independently of the NC program in the event of `NC Stop`, alarm handling, end of program, program control and `RESET`, when control of the command axis has been taken over from the PLC.

The control of the command axis is done via the axial VDI interface (PLC→NCK) by means of the NST:

DB 31, ... DBX28.7 == 1 (PLC controls axis/P5)

For more information about the precise sequence of operations of the various steps for transferring control of the command axis to the PLC, please see:

**References:**

/FB2/ Function Manual, Extended Functions; Positioning Axes (P2)

### Boundary condition

An axis cannot be moved from the part program and from synchronized actions simultaneously. but may be moved from these two sources successively.

Delays may occur if an axis has been moved first from a synchronized action and then programmed again in the part program.

**Note**

MD 30450 $MA_IS_CONCURRENT_POS_AX indicates whether the axis is primarily intended as a command axis or for programming by the part program:

0: no competing axis

1: competing axis (command axis)

### Example 1

...

ID=1 EVERY $A_IN[1]==1 DO **POS[X]=100**

...

### Example 2

An axis motion can be triggered as technology cycle. (See Chapter "Call of technology cycles")

**Main program:**

...

ID=2 EVERY $A_IN[1]==1 DO **AXIS_X**

...

**Axis program:**

AXIS_X:

```
M100
POS[X]=100
M17
```

### Programming

Positioning axis motions are programmed in synchronized actions as they are from the part program:

ID = 1 EVERY $AA_IM[B] > 75 DO **POS[U]=100**

The programmed position is evaluated in inches or in the metric system depending on whether setting `G70` or `G71` is active in the current part program block.

`G70/G71` and `G700/G710` can also be programmed directly in synchronized actions.

This allows the inch/metric evaluation of a command axis movement to be defined independent of programming in the part program.

ID = 1 WHENEVER $A_OUT[1] ==1 DO **G710** POS[X]=10

ID = 2 EVERY **G710** $AA_IM[Z] >100 DO **G700** POS[Z2]=10

---

**Note**

Only `G70, G71, G700, G710` can be programmed in synchronized actions!

See Section 2.1

G functions, which are programmed in the synchronized action block, are only effective for the synchronized action or within the technology cycle. They have no effect on subsequent blocks in the part program.

---

**References:**

/PG/ Programming Manual Fundamentals; Position data

### Absolute/Incremental end position

The end position can be programmed either absolutely or incrementally. The position is approached absolutely or incrementally depending on whether G90 or G91 is active in the main program block currently being processed.

It is possible to explicitly program whether the value must be interpreted as an absolute or incremental setting:

IC: incremental

AC: absolute

DC: direct i.e. position rotary axis via shortest route

ACN: Position modulo rotary axis absolutely in negative direction of motion

ACP: Position modulo rotary axis absolutely in positive direction of motion

CAC: Traverse axis to coded position absolutely

CIC: Traverseaxis to coded position incrementally

CDC: Traverse rotary axis to coded position via shortest route

CACN: Traverse modulo rotary axis to coded position in negative direction

CACP: Traverse modulo rotary axis to coded position in positive direction

Coded positions are settings stored in machine data.

### Example 1 fixed value

```
ID = 1 EVERY $AA_IM[B] > 75 DO POS[U]=IC(10)   ;  When event occurs, position U-axis further by 10
```

### Example 2 current value

The traversing path is generated in real time from a real-time variable:

ID = 1 EVERY $AA_IM[B] > 75 DO **POS[U]=$AA_MW[V]-$AA_IM[W] + 13.5**

### Axial frame

The response of synchronized actions and axial frames is explained in the sections below:

### Activation

When positioning motions are executed from synchronized actions, the axial offsets, scaling and mirroring functions of the programmable and settable frames (G54 etc.) as well as tool length compensations are all operative.

Whichever frame is operative in the current block takes effect. If a rotation is active in the current block, then an alarm is output to reject a positioning motion initiated from a positioning motion.

## Example

```
TRANS X20
IDS= 1 EVERY $A_IN==1 DO POS[X]=40
G1 Y100                                    ; When the input is set, X is positioned to 60

…
TRANS X-10
G1 Y10                                     ; When the input is set, X is positioned to 30
```

## Suppression

The effectiveness of frames and tool lengths can be suppressed with the machine data:

MD32074 $MA_FRAME_OR_CORRPOS_NOTALLOWED (Frame or HL-offset are not permitted)

## Suppressing axial frames

Axial frames that travel incrementally to indexing positions have no effect on a command axis. For this reason, bit 9 = 1 is set and the command axis is positioned with JOG in the following machine data:

MD32074 $MA_FRAME_OR_CORRPOS_NOTALLOWED[AX4] (Frame or HL-offset are not permitted)

### Example

```
RANS A=0,001
POS[A]=CAC(2)                    ; Axis travels to position 180.001 degrees

                                 ; The axial frame is not effective for the
                                   command axis
                                 ; MD32074
                                   $MA_FRAME_OR_CORRPOS_NOTALL
                                   OWED[AX4] = 'H0020'

WHEN TRUE DO POS[A]=CIC(-1)      ; Axis travels to position 180.000 degrees
```

**Note**

If a command axis travels to indexing positions incrementally, axial frames usually have **no** effect on this command axis.

## 2.4.14 Axial feedrate from synchronized actions

### Feedrates

An axial feedrate can be programmed in addition to the end position:

ID = 1 EVERY $AA_IM[B] > 75 DO POS[U]=100 **FA[U]=990**

The axial feed for command axes acts modal. It is programmed under address FA. The default value is allocated via the axial machine data:

MD32060 $MA_POS_AX_VELO (initial setting for positioning axis velocity)

The feedrate value is either preset to a fixed quantity or generated in real time from real-time variables:

### Example of calculated feedrate

ID = 1 EVERY $AA_IM[B] > 75 DO POS[U]=100 **FA[U]=$AA_VACTM[W]+100**

The feedrate value is either programmed as a linear or rotational feed:

The feedrate type is determined by setting data:

SD43300 $SA_ASSIGN_FEED_PER_REV_SOURCE(revolutional feed rate for position axes/spindles)

This data can be altered by an operator input, from the PLC or from the part program. In synchronization with the part program context, the feedrate type can be switched over using the NC commands FPRA0N, FPRA0F. See also:

References:

/FB1/Function Manual Basic Functions; Feedrates (V1)

### Feedrate change

To apply a modified feedrate for command axes, you only have to **specify the end point again**. You do not have to change it.

**Example**

IDS=1 WHENEVER $A_IN[1] 00 1 DO POS[X] = 100 FA[X] = $R1

**Explanation:**

Whenever the analog input changes to 1, position 100 is specified **and** the **feedrate** from the R parameter 1 is simultaneously **applied**. This applies even if an approach to position 100 is already active for command axis X.

**Note**

The axial feedrate from motion-synchronous actions is not output as an auxiliary function to the PLC. Parallel axial technology cycles would otherwise block one another.

## 2.4.15 Starting/Stopping axes from synchronized actions

### Starting/stopping

Command axes can be stopped from synchronized actions even when no end position has been specified. In this case, the axis is traversed in the programmed direction until another motion is set by means of a new motion or positioning command or until the axis is halted by a stop command. This method can be used, for example, to program an endlessly turning rotary axis.

Starting and stopping are programmed using the same method as positioning motions.

**MOV[axis] = value**

The data type for the value is INTEGER.

The sign of the value determines the direction of movement:

>0: Axis motion in the positive direction

<0: Axis motion in the negative direction

==0: Stop axis motion

If a moving indexing axis is halted by command MOV[axis]=0, then the next indexing position is approached in the same way as in JOG mode.

The **feedrate** for the motion can be programmed with **FA[axis]=value** (see above). If no axial feedrate is programmed, the feedrate value is derived from an axis motion that may already be activated from synchronized actions or from the axis velocity set via the machine data:

MD32060 $MA_POS_AX_VELO (initial setting for positioning axis velocity)

### Example

```
... DO MOV[U]=0                    ;  Stop axis motion when the condition is fulfilled.
```

## 2.4.16 Axis replacement from synchronized actions

### Application

For a tool change, the corresponding command axes can be requested as an action of a synchronized action using GET(axis). If the tool has been changed, these command axes can be released again for the channel from the synchronized action using RELEASE(axis).

### Request axis

An axis can be requested with **GET[axis]** as action of a synchronized action.

### Release axis

An axis can be released for replacement with **RELEASE[axis]** as action of a synchronized action

**Note**

Each axis must be assigned to the channel via machine data.

### Axis type and axis status regarding axis replacement

The currently valid axis type and axis status, at the activation instant of synchronized action, can be interrogated using $AA_AXCHANGE_TYP or $AA_AXCHANGE_STAT. Dependent on the channel that has the actual interpolation right of this axis presently has, and from the actual status of the permissible axis replacement, a different sequence is obtained from the synchronized action.

From a synchronized action, an axis can be requested at the request instant with **GET[axis]** , if

- Another channel has the write or interpolation authorization for the axis.
- The requested axis is already assigned the requested channel.
- The axis in the neutral axis state is controlled by the PLC.
- The axis is a command axis, oscillating axis, or concurrent PLC axis.
- The axis is already assigned to the NC program of the channel.

**Note**

**Condition:**An axis controlled exclusively by the PLC cannot be assigned to the NC program. This is also the case for a permanently assigned PLC axis.

From a synchronized action, an axis can be released for axis replacement with `RELEASE[axis]`, if the axis:

- Was previously assigned to the NC program of the channel.
- Is already in the neutral axis state.
- another channel already has the interpolation rights of this axis

### Request axis from another channel

If, when the `GET` action is activated, **another channel**has the interpolation authorization for the axis $AA_AXCHANGE_TYP[axis] == 2, axis replacement is used to fetch the axis from this channel $AA_AXCHANGE_TYP[axis] == 6 and assign it to the requesting channel as soon as possible. The axis then becomes the **neutral axis** ($AA_AXCHANGE_TYP[<axis>]==3).

The state change to a neutral axis does **not** result in reorganization in the requesting channel.

**Requested axis was already requested as neutral axis:**

$AA_AXCHANGE_TYP[axis]==6, the axis is required for the NC program $AA_AXCHANGE_TYP[axis] == 5 and assigned as soon as possible to the NC program of the channel $AA_AXCHANGE_TYP[axis] == 0.

**Note**

This assignments **results in** a reorganization.

### Axis is already assigned to the requested channel

If the requested axis has already been assigned **to this channel**at the point of activation, and its status is that of a neutral axis not controlled by the PLC $AA_AXCHANGE_TYP[axis]==3, it is assigned to the NC program $AA_AXCHANGE_TYP[axis]==0.

This results in **a** reorganization procedure.

### Axis in the state of the neutral axis is controlled from the PLC

If the axis in neutral axis status is controlled **by the PLC**$AA_AXCHANGE_TYP[axis]==4), the axis is requested as a neutral axis $AA_AXCHANGE_TYP[axis] == 8. This disables the axis for automatic axis replacement between channels (Bit 0 == 0) in accordance with the value of bit 0 in machine data:

MD10722 $MN_AXCHANGE_MASK (Parametring the axis replacement behavior)

This corresponds to $AA_AXCHANGE_STAT[axis] == 1.

### Axis is active as command axis / assigned to the PLC

If the axis is active **as command axis** or oscillating axis or concurrent positining axis (PLC-axis) $AA_AXCHANGE_TYP[axis]==1, then the axis is requested as neutral axis $AA_AXCHANGE_TYP[axis] == 8. This locks the axis for automatic axis replacement between channels (Bit 0 == 0) in accordance with the value of bit 0 in machine data:

MD10722 $MN_AXCHANGE_MASK (Parametring the axis replacement behavior)

This corresponds to $AA_AXCHANGE_STAT[axis] == 1.

A new GET action will request the axis for the NC program $AA_AXCHANGE_TYP[axis] changes to == 7.

### Axis already assigned to the NC program of the channel

If the axis is **already assigned to the NC-program** of the channel $AA_AXCHANGE_TYP[axis]==0 or if this assignment is requested, e.g., axis replacement triggered by NC program $AA_AXCHANGE_TYP[axis]==5 or $AA_AXCHANGE_TYP[axis] == 7, then there is **no**state change.

### Release axis for axis replacement RELEASE(Axis)

If the axis is already assigned the NC program of the channel $AA_AXCHANGE_TYP[axis] == 0, then it is transitioned into the status of a neutral axis $AA_AXCHANGE_TYP[axis] == 3 and if required, released for axis replacement in another channel.

This results in **a** reorganization procedure.

**Axis to be released is already a neutral axis:**

If the axis is already in the state of a neutral axis

$AA_AXCHANGE_TYP[<Axis>] == 3 or as command or oscillating axis active or assigned to the PLC for traveling, PLC-Axis == concurrent positioning axis, $AA_AXCHANGE_TYP[Axis] == 1, then the axis is released for an automatic axis interchange between channels. The state of $AA_AXCHANGE_STAT[axis] is reset from 1 to 0 if there is no other reason to link the axis to the channel.

An axis link is involved, e.g. for an axis coupling, active fast lift-off, active transformation, JOG request, rotating frame with possible PLC, command or oscillating axis motion.

### Another channel already has write authorization

If another channel already has the write authorization or interpolation authorization (rights) $AA_AXCHANGE_TYP[axis] == 2, then there is no state change. This also means that waiting for an axis (initiated by NC program $AA_AXCHANGE_TYP[axis] == 5) or through a previous request `GET(axis)` from synchronized action $AA_AXCHANGE_TYP[axis] == 6 **cannot be interrupted by a** `RELEASE(axis)` from a synchronized action.

### Constraints, GET, RELEASE

If several `GET` and `RELEASE` tasks for an axis have been issued in a synchronized action or in a line of a technology cycle, then these tasks mutually cancel one another. Only the last task remains.

**Example**

GET(X,Y) RELEASE(Y,Z) GET(Z) results in GET(X) RELEASE(Y) GET(Z).

Within the actions of a synchronized action, the system does not wait for a `GET` or `RELEASE` request to be fulfilled. This means that `GET[axis] POS[axis]` can result in an alarm message if the axis presently cannot be accessed for the command axis motion.

For a technology cycle, for a sub-division to several lines, the system waits. This means that the system only advances to the next line from line with `GET(axis)`, if the axis - e.g. as neutral axis - was accepted from another channel; refer to the subsequent example, `GET`, `RELEASE` in the technology cycle.

## Example

### GET, RELEASE using synchronized actions

Z axis has been declared in the first and second channels.

**Program sequence in the first channel:**

```
WHEN TRUE DO RELEASE(Z)                    ;  Z axis becomes neutral
WHENEVER $AA_TYP[Z] == 1 DO RDISABLE       ;  Read-in disable as long as
                                           ;  Z axis is program axis
N110 G4 F0.1                               ;
WHEN TRUE DO GET(Z)                        ;  Z axis becomes again
                                           ;  NC program axis
WHENEVER($AA_TYP[Z]<>1) DO RDISABLE        ;  Read-in disable as long as
                                           ;  Z axis is program axis
N120 G4 F0.1                               ;
WHEN TRUE DO RELEASE(Z)                    ;  Z axis becomes neutral
WHENEVER $AA_TYP[Z] == 1 DO RDISABLE       ;  Read-in disable as long as
                                           ;  Z axis is program axis
N130 G4 F0.1                               ;
N140 START(2)                              ;  2. Start channel
```

**Program sequence in the second channel:**

```
WHEN TRUE DO GET(Z)                        ;  ; Move Z axis to second channel (neutral)
WHENEVER $AA_TYP[Z] == 0 DO RDISABLE       ;  Read-in disable as long as
                                           ;  Z axis is in another channel
N210 G4 F0.1                               ;
WHEN TRUE DO GET(Z)                        ;  Z axis becomes NC program axis
WHENEVER($AA_TYP[Z]<>1) DO RDISABLE        ;  Read-in disable till
                                           ;  Z axis is program axis
N220 G4 F0.1                               ;
WHEN TRUE DO RELEASE(Z)                    ;  Z axis in second channel is neutral axis
WHENEVER $AA_TYP[Z] == 1 DO RDISABLE       ;  Read-in disable as long as
                                           ;  Z axis is program axis
N230 G4 F0.1                               ;
N250 WAITM(10,1,2)                         ;  Synchronize with channel 1
N999 M30                                   ;
```

**Program sequence in the first channel continues:**

```
N250 WAITM(10,1,2)                         ;  Synchronize with channel 2
WHEN TRUE DO GET(Z)                        ;  Move Z axis to this channel
WHENEVER $AA_TYP[Z] == 0 DO RDISABLE       ;  Read-in disable as long as
                                           ;  Z axis is in another channel
N160 G4 F0.1                               ;
N199 WAITE(2)                              ;  Wait for end of program in channel 2
N999 M30                                   ;
```

## Example

**GET, RELEASE in the technology cycle**

It has been declared in first and second channel: The axis U, machine data:

MD30552 $MA_AUTO_GET_TYPE = 2 (get automatic GET for axis)

Currently, channel 1 has the interpolation right and the following technology cycle is started in channel 2:

```
GET(U)                    ; Move U axis to channel
POS[U]=100                ; U axis is to be moved to position 100
```

The command-axis-movement line (POS ...) is not executed until the U axis has been fetched to channel 2.

## AXTOCHAN Move axis to another channel

From a synchronous action, an axis for a specific channel can be requested with the NC-language command AXTOCHAN (Axis, channel number)[axis, channel number[, ...]])

This does not have to be its own channel, that currently has the interpolation authorization for the axis. This means that it is possible to shift and axis into another channel.

If the axis **is already assigned to the NC program** of the requested channel $AA_AXCHANGE_TYP[axis] == 0, then there is **no**state change.

In the event of an axis being requested for the same channel, AXTOCHAN from the synchronized action is mapped to a GET from a synchronized action. For the

1. **first** request for the own channel, the axis is assigned to the neutral axis.
2. **second** request assigned to the NC program, which is the case for an GET in the NC program.

## Constraints AXTOCHAN

A PLC controlled corresponds to a "competing positioning axis" where special constraints must be carefully observed.

Traversing using static synchronized actions: Stage 2 is necessary.

**References:**

/FB2/ Function Manual, Extended Functions; Positioning Axes (P2)

---

**Note**

A PLC axis cannot replace the channel. This is also not possible using an entry from the VDI interface.

An axis controlled exclusively by the PLC cannot be assigned to the NC program.

---

## 2.4.17 Spindle motions from synchronized actions

### General

Analogously to positioning axes, it is also possible to start, position and stop **spindles** from synchronized actions. Spindle movements can be started at defined points in time by blocking a spindle motion programmed in the part program or by controlling the axis motion from synchronized actions.

### Starting/stopping

The use of these functions is recommended for cyclic operations or for operations that are predominantly event-controlled.

### Stop until event occurs

**Application:**

A spindle is programmed within a machining routine, but must not be started at the beginning of the block in particular circumstances. A synchronized action is used to maintain a 0 override until the spindle is to start.

**Example**

```
ID=1 WHENEVER $A_IN[1]==0 DO
$AA_OVR[S1]=0
G01 X100 F1000 M3 S1=1000              ; The spindle is started asynchronous
                                       ; to the path processing;
                                       ; the start is done via a digital input
```

### Auxiliary functions, speed, position

These functions are programmed in the action section of the synchronized action by exactly the same method as used in the part program.

Commands: `S= ..., M3, M4, M5, SPOS= ...`

**Example**

```
ID = 1 EVERY $A_IN[1]==1 DO M3 S1000
ID = 2 EVERY $A_IN[2]==1 DO SPOS=270
```

Without numeric extension the commands for the master spindle apply. By specifying a numeric extension, it is possible to activate each spindle individually:

```
ID = 1 EVERY $A_IN[1]==1 DO M1=3 S1=1000 SPOS[2]=90
```

For programming the type of positioning the same rules are applicable as for the positioning axes (see above)

If **concurrent commands** are input via simultaneously active synchronized actions for an axis/spindle, then the commands are applied in the **chronological sequence** in which they are programmed.

**Example**

```
ID=1 EVERY $A_IN[1]==1 DO M3 S300          ; Speed and direction of rotation
ID = 2 EVERY $A_IN[2]==1 DO M4 S500        ; Speed and direction of rotation
ID=3 EVERY $A_IN[3]==1 DO S1000            ; New speed specification
                                           ; for active spindle rotation
ID=4 EVERY ($A_IN[4]==1) AND               ; Position spindle
($A_IN[1]==0) DO SPOS=0
```

## Feed

The feedrate for "Position spindles" can be programmed from a synchronized action with:

**FA[Sn]=** ...

**Note**

Only a modal data item is available for the feedrate of synchronized actions for spindle mode and axis mode. FA[S] and FA[C] are supplied in the same way.

## SW limit switches, working area limitations

The restrictions imposed by SW limit switches and working area limitations also apply to axis/spindle movements activated from synchronized actions.

## Influence of limitations on movements from synchronized actions

Working area limitations programmed with G25/G26 are taken into account as a function of setting data:

SD43400 $SA_WORKAREA_PLUS_ENABLE (Working-area limitation active in the positive direction)

Activation and deactivation of working area limitations by G functions WALIMON/WALIMOF**in the part program** does not affect command axes.

## Correcting acceleration

The acceleration specified in the following machine data can be changed in the range from 0% to 200% with ACC[Axis]=0..200 (**ACC[Axis]=**<Value> (acts in the part program and in synchronized actions)):

MD32300 $MA_MAX_AX_ACCEL (axis acceleration)

### Axis coordination

If a positioning command (POS, MOV) is started from synchronized actions for an axis that is already operating as a path or PLC axis, then processing is aborted with an alarm.

### Axis movement by PP and SA alternately

In typical cases, an axis is either moved from the part program (PP) in motion blocks or as a positioning axis from a synchronized action (SA). However, if the same axis must be traversed alternately from the part program as a path axis or positioning axis and from synchronized actions, then a coordinated transfer takes place between both axis motions.

### Example

**; traverse X axis alternately from part program and from synchronized actions**

```
N10 G01 X100 Y200 F1000                ;  X axis programmed in part program
...
N20 ID=1 WHEN $A_IN[1]==1 DO POS[X]=100 FA[X]=200
                                       ;  Start positioning from synchronized action,
                                       ;  when digital input is present
...
CANCEL(1)                              ;  Select synchronized action
...
N100 G01 X100 Y200 F1000               ;  X: Path axis
                                       ;  Period of delay before motion, when digital
                                       ;  input was 1 and hence X from
                                       ;  Synchronized action was positioned
```

### On-the-fly transitions

Transitions can be made between command axes and spindles.

### Starting point

Since several synchronized actions can be active simultaneously, the situation may arise where an axis motion is started when the axis is already active.

### Response

In this case, the most **recently activated** motion is applicable. POS- and MOV motions can be activated alternately.

When a reversal in the direction of motion is forced in this manner, the axis is first decelerated and then positioned in the opposite direction.

**Examples:**

```
ID=1 EVERY $AC_TIMER[1] >= 5 DO POS[V]=100 FA[V]=560
ID=2 EVERY $AC_TIMER[1] >= 7 DO POS[V]=$AA_IM[V] + 2 FA[V]=790
```

; Owing to the programming with $AC_TIMER[1] the synchronization with ID=2 is the most recently activated, its specifications become effective and release the specifications of ID=1 ... .

The end position and feedrate for a command axis can therefore be adjusted while the axis is in motion.

## Example Activation by signal

```
ID=1 EVERY $A_IN[1]==1 DO POS[U]=$AA_IM[U]+$AA_IM[V]*.5
           FA[U]=$AA_VACTM[U]+10
```

## Legal transitions

| in<br>↓ | To<br>→ | POS | MOV=1<br>MOV = -1 | MOV=0 | SPOS | M3<br>M4 | M5 | LEADON | TRAIL ON |
|---|---|---|---|---|---|---|---|---|---|
| **Axis stationary** | | | | | | | | | |
| Axis mode | | x | x | x | x | x | x | x | x |
| Position-controlled spindle | | x | x | x | x | x | x | | |
| Speed-controlled spindle | | | | | x | x | x | | |
| **Axis in motion** | | | | | | | | | |
| Axis mode | | x | x | x | | | | x | x |
| Position-controlled spindle | | | | | | | | | |
| Speed-controlled spindle | | | | | x | x | x | | |

Transitions marked with x are permitted: Transitions not marked with an x are rejected with an alarm.

**Example: Legal transition**

```
N10 WHEN $AA_IM[Y] >= 5 DO MOV[Y]=-1          ; At position +5 axis in
                                              ; Negative direction
                                              ; start
N20 WHEN TRUE DO POS[Y]=20 FA[Y]=500          ; start Y axis, when
                                              ; Block is changed
```

### On-the-fly transitions in case of axis couplings

Positioning axis motions and movements resulting from axis couplings programmed via synchronized actions can be activated alternately.

See Subsection 2.4.02 "Activating and Deactivating Couple Motions and Couplings" and:

**References:**

/FB3/ Function Manual, Special Functions; Coupled axes and ESR (M3)

Legal transitions in master value couplings are marked with `LEADON` in the above table. Legal transitions in coupled motions are marked with `TRAILON`.

## 2.4.18 Setting actual values from synchronized actions

### Application

The `PRESETON` function can be used to redefine the control zero in the machine coordinate system.

### Function

When Preset is applied, the axis is not moved. A new position value is merely entered for the current axis position.

### Programming

The value for **one** axis can be programmed in each synchronized action.

**Example**

**WHEN $AA_IM[a] >= 89.5 DO PRESETON(a, 10.5)**

with `PRESETON` (axis, value)

Axis: Axis of which the control zero is to be changed

Value: The value by which the control zero is to be changed.

### Permissible applications

`PRESETON` from synchronized actions can be programmed for:

- modulo rotary axes that have been started from the parts program and
- all command axes that have been started from a synchronized action

### Restrictions

`PRESETON` cannot be programmed for axes, which are involved in a transformation.

### Example

You can find an example of how to use PRESETON in conjunction with an "On-the-fly parting" application in the sub-section "On-the-fly parting".

**Note**

Setting of actual values PRESETON may only be done with the key words WHEN or EVERY.

## 2.4.19 Activating/deactivating coupled motions and couplings

### Introduction

**References:**

/FB3/ Function Manual, Special Functions; Coupled axes and ESR (M3)

The following functions are described in detail in the functional description given above.

- Coupled motion

  Slave axis(axes) is(are) linked to a master axis via a coupling factor.

- Curve tables

  Curve tables represent a (complex) relationship between the master and slave values. The following may be applied as master values:

  – Setpoints generated by the control

  – Actual values measured by encoders

  – Externally specified quantities

    Situations, where a following axis is linked to a leading axis by means of a curve table are particularly relevant with respect to synchronized actions.

- Master value coupling

  The following axis master value couplings from the following master value couplings possible for the part programs are available:

  – axis master value coupling

  – path master value coupling,

- Electronic gearbox

  With the help of the "Electronic gearbox" the movement of a following axis FA can be interpolated dependent on up to five leading axes LA. From the parts program a gearbox group can be:

  – is defined.
  – Switched-in
  – Switched off
  – Deleted

be.

- Generic coupling

  With the help of a coupling module, the motion of one axis, (_FA following axis), can be interpolated depending on other (_leading) axes. Coupling modules can also be created and activated in the part program and implicit synchronized actions. The coupling modules created and activated in synchronized actions are designated as main run couplings. The relationships between leading axis/values and the following axis are defined for each leading axis/value by a coupling rule, either a coupling factor or a curve table.

  Each coupling property of the generic coupling can be programmed using keywords. The following keywords are available in synchronized actions:

  – CPLON Switching-on a leading axis of a coupling module
  – CPLOF Switching-off a leading axis of a coupling module
  – CPOF Switching-off a coupling module
  – CPLNUM Counter of the coupling factor
  – CPLDEN Denominator of the coupling factor
  – CPLCTID Number of the curve table
  – CPSETTYPE Coupling type of the existing coupling modes

**Note**

When programming, attention must be paid that the utilized CP key words within a synchronized action are processed **from left to right**.

That is, unlike the programming in the part program, the effect of the various key words depends on their order in the synchronized action.

### TRAILON - Coupled motion from a synchronized action

From a synchronized action it is possible to define and simultaneously activate the assignment of a following axis to a leading axis using a coupling factor:

```
... DO TRAILON(FA, LA, Kf)
```

where:

| | |
|---|---|
| FA | Following axis |
| LA | Leading axis |
| Kf | Coupling factor |

The commands for separating the coupled-axis grouping are as follows:

```
... DO TRAILOF(FA, LA, LA2)
```
or

```
... DO TRAILOF(FA)
```
in short form

where:

| | |
|---|---|
| FA | Following axis |
| LA | Leading axis |
| LA2 | Leading axis 2, optional |

### Curve tables

The relationship between a master quantity and a slave quantity stored in curve tables can be utilized in synchronized actions in the same way as other REAL functions (e.g. SIN, COS).

### Calculate slave value

The slave value calculated from a master value on the basis of curve table n must be assigned to an arithmetic variable.

**Example**

```
... DO $R17=CTAB(LW, n, degree)
```

where:

| | |
|---|---|
| LW | Master value |
| n | Number of curve table |
| degrees | Pitch parameters, result |
| | 2 more options parameters for scaling: |
| | - Following axis |
| | - Leading axis |

**Example**

```
DEF REAL GRADIENT
...
WHEN $A_IN[1] == 1 DO $R17 = CTAB(75.0, 2, GRADIENT)
```

### Calculating master value

From a synchronized action it is possible to calculate a concrete master value for a slave value on the basis of a curve table.

**Example**

```
... DO $R18=CTABINV(FW, aprLW, n, degree)
```

where:

| | |
|---|---|
| FW | Slave value |
| aprLW | approximated master value, with which a unique LW can be determined in case of multi-valued inverse function of the curve table |
| n | Number of curve table |
| degrees | Pitch parameters, result:<br>2 more options parameters for scaling:<br>- Following axis<br>- Leading axis |

The functions `CTAB` and `CTABINV` can be programmed both in conditions and in the action section of synchronized actions.

### LEADON Axis master value coupling from synchronized actions

The coupling between following axis FA and leading axis LA based on the stored curve table with number NR is called in the action section of synchronized actions as follows:

```
... DO LEADON(FA; LA, NR)
```

where:

| | |
|---|---|
| FA | Following axis |
| LA | Leading axis |
| NR | Number of curve table |

### LEADOF Deactivate axis coupling from synchronized action

If the axis master value coupling must be canceled again on the fulfillment of another condition, then the action is:

`... DO LEADOF(FA, LA)` or

`... DO LEADOF(FA)` in abbreviated form

### System variables

The system variables of the master value coupling as specified in the list of system variables can be read/written from the part program and synchronized actions.

**References:**

/PGA1/ Parameter Manual, System Variables

### Detection of synchronism

System variable $AA_SYNC[ax] can be read from the parts program and synchronous action and indicates whether and in what manner the following axis FA is synchronized:

0: Not synchronized

1: Coarse synchronism (according to MD37200 $MA_COUPLE_POS_TOL_COARSE)

2: Fine synchronism (according to MD37210 $MA_COUPLE_POS_TOL_FINE)

### Definition of application

Couplings directly activated in the part program are activated at block limits. With the additional option of activating couplings from synchronized actions, it is possible to implement event-controlled, differential activation, e.g.

- from block beginning for specific axis path
- up to block end for specific distance-to-go
- appearance of digital input signals or
- combinations of these.

  See Chapter "Components of synchronized actions", Conditions

For more information about programming coupling functions and curve tables, please see:

**References:**

/PGA/Programming Manual Advanced

---

**Note**

Axes, which might be in any given motional state at the instant they are coupled via synchronized actions, are synchronized by the control system. You can find details about this in /FB3/ Function Manual Special Functions; Axis Couplings and ESR (M3).

---

### Examples

You can find an example of axis coupling using curve tables in Subsection 6.7.3 "On-the-Fly Parting".

- Generic coupling - activate coupling module in synchronized actions

  It is possible to program keywords in synchronized actions. In this way, coupling modules can also be used in synchronized actions. When the coupling module is activated in a synchronized action, the following axis must already be active in the channel and be in the state “neutral axis ” or “axis already assigned to the NC-program of the channel”.

  When required, this axis state must be provided before the activation of the coupling module. This can also be done in synchronized actions with the help of the command `GET [Axis]`.

- Cross-channel coupling, axis replacement

  For axis replacement, the following and leading axes must be known to the calling channel. Axis replacement of leading axes can be performed independently of the state of the coupling. A defined or active coupling does not produce any other boundary conditions.

  **Note**

  With the activation of the coupling, the following axis becomes the main run axis and is not available for an axis replacement. The following axis is thus logged out of the channel. With this type of coupling, an overlaid movement is therefore not possible.

  For additional information on axis replacement in synchronized actions, see Subsection 2.4.16 Axis replacement from Synchronized Actions; `GET/RELEASE[axis]`

- Activating/deactivating a coupling

  The CP keywords are processed in synchronized actions directly by the coupling module. This means that a CP keyword takes effect immediately. Activation of the coupling of a leading axis to a following axis:

  **CPLON[Fax]=<Leading axis or Leading spindle>**

  Deactivation of the coupling of a leading axis to a following axis:

  **CPLON[Fax]=<Leading axis or Leading spindle>**

  With the following keyword, all the leading axes for slave axes in synchronized actions are deactivated.

  **SPOF=<FAx>**

- Coupling factor

  When programming a coupling factor, a previously activated non-linear coupling relationship (e.g., a curve table) is deactivated. Determining the numerator of the coupling factor in synchronized actions:

  **CPLNUM[FAx,LAx]= <value>**

  Determining the denominator of the coupling factor in synchronized actions:

  **CPLNUM[FAx,LAx]= <value>**

- Curve table

  When programming a table number, a previously activated non-linear coupling relationship e.g., a curve table is deactivated. In synchronized actions, the leading axis/spindle-specific coupling component is calculated for the master value of the leading axis/spindle using the specified curve table:

  **CPLCTID[FAx,LAx]= <value>**

- Example

  Programming with keywords in synchronized actions:

  Example 1:

  Definition of an axis coupling with a leading axis

  DO CPDEF=YCPLDEF[Y]=X CPLNUM[Y,X]=1.5

  Example 2:

  N10 WHEN TRUE DO CPLON[X]=X CPLNUM[X,Y]=2; OK

  N20 WHEN TRUE DO CPLNUM [A,B]=" CPLON [A=B] ; Alarm

  The order in the N20 block is not permitted since CPLNUM is to be set before the coupling module has been created in the part program with CPDEF.

  Example 3:

  N10 WEHN TRUE DO CPLON [X]=Y CPLNUM[X,Y]=3

  N15 Y= 100 F100

  N20= WHEN TRUE DO CPOF=X CPLON[X]=YCPLNUM[X,Y]=3

  In this example, the coupling module active in N10 is recreated and reactivated and is also re-synchronized.

  Example 4:

  N10 WHEN TRUE DO CPLON[X]=Y CPLNUM[X,Y]

  N15 Y=100 F100

  N20 WHEN TRUE DO CPOF0X MOV[X]=1

  In the N20 block, the coupling module is switched off and deleted with CPOF. The following axis is thus again available for the MOV command.

- Use previous coupling types - Existing coupling types TRAII, LEAD, EG and Coup

  If a presetting of the existing coupling types such as coupled motion, master value coupling, electronic gearing, or synchronized spindles is desired, the following key word is also permitted when creating or defining the coupling module

  **CPSETTYPE[FAx]=<value>**

  Permitted in synchronized actions.

  The following value range is possible:

  – "CP" Can be user-programmed (default value)
  – "TRAIL" Coupling type "Coupled motion"
  – "LEAD" Coupling type "Master value coupling"
  – "EG" Coupling type "Electronic gear"
  – "COUP" Coupling type "Synchronized spindle"

## 2.4.20 Measurements from synchronized actions

### Introduction

Measurement functions available for the part programs:
`MEAS, MEAW, MEASA, MEAWA, MEAC`

**References:**

/PGA/Programming Manual Advanced

/FB2/ Function Manual, Extended Functions; Measurements (M5).

Only the following may be used in synchronized actions:

- `MEAWA` Axial measurement without deletion of distance-to-go
- `MEAC` Axial, continuous measurement

While measuring functions are limited to one block at a time in part program motion blocks, they can be activated and deactivated any number of times from synchronized actions:

**Note**

With static synchronized actions, measurements are also available in JOG mode.

### Programming

| | |
|---|---|
| **`MEAWA`**`[Axis]=` | (Mode, Trigger event_1, Trigger event_2, Trigger event_3, Trigger event_4) |
| | ; activate axial measurement without deletion of distance-to-go |
| **`MEAC`**`[Axis]=` | (Mode, Measurement memory, Trigger event_1, Trigger event_2, Trigger event_3, Trigger event_4) |
| | ; axial, activate continuous measurement |
| `Axis:` | Axis for which measurement is taken |

Table 2-3 Mode meanings:

| Tens decade | Units decade | Description |
|---|---|---|
| | 0 | Cancel measuring job |
| | 1 | Up to 4 trigger events can be activated**simultaneously** |
| | 2 | Up to 4 trigger events can be activated **consecutively** |
| | 3 | Up to 4 trigger events can be activated **consecutively**, but with no monitoring of trigger event 1 on START |
| 0 | | active measuring system |
| 1 | | 1. Measuring system |
| 2 | | 2. Measuring system |
| 3 | | Both measuring systems |

Trigger_event_1 to trigger_event_4:

| | | |
|---|---|---|
| 1: | rising edge, probe 1 | |
| - 1: | falling edge, probe 1 | *Optional* |
| 2: | rising edge, probe 2 | *Optional* |
| - 2: | falling edge, probe 2 | *Optional* |
| Measurement memory: | Number of a FIFO variable | |

Measured values are supplied exclusively for the **machine** coordinate system.

## MEAWA

`... DO MEAWA[Axis]=( , , , , )` ; axial measurement without deletion of distance-to-go

If needed, deletion of distance-to-go can be called explicitly in synchronized actions. See Chapter "DELDTG" and the example given below.

GEO axes and axes involved in transformations can be programmed individually.

**Programming:**

The programming method is identical to that used in the part program

**Note**

System variable $AC_MEA does not supply any useful information about the validity of a measurement called from a synchronized action.

Only one measurement job at a time may be active for an axis.

System variables:

| | |
|---|---|
| $AA_MEAACT[Axis] | returns the current measurement status of an axis. |
| 1 | Measurement active |
| 0 | Measurement not active |
| $A_PROBE[Probe] | returns the instantaneous status of the probe. |
| 1 | Probe switched, high signal |
| 0 | Probe not switched, low signal |

Measured values in machine coordinate system with 2 probes (encoders):

| | |
|---|---|
| $AA_MM1[axis] | Trigger event 1, encoder 1 |
| $AA_MM2[axis] | Trigger event 1, encoder 2 |
| $AA_MM3[axis] | Trigger event 2, encoder 1 |
| $AA_MM4[axis] | Trigger event 2, encoder 2 |

## MEAC

```
... DO MEAC[axis]=(mode, No_FIFO, trigger events)
```

The variables $AC_FIFO (see Chapter "FIFO-variables (circulating memory)".) are provided for the purpose of storing measured values from cyclic measuring processes. Mode and trigger event see above

## Examples:

Two FIFOs have been set up in machine data for the following examples.

**Machine data:**

```
MD28050 $MC_MM_NUM_R_PARAM = 300
MD28258 $MC_MM_NUM_AC_TIMER = 1
MD28260 $MC_NUM_AC_FIFO = 2              ;  2 FIFOs
MD28262 $MC_START_AC_FIFO = 100          ;  first FIFO starts from R100
MD28264 $MC_LEN_AC_FIFO = 22             ;  each FIFO can take up 22 values
MD28266 $MC_MODE_AC_FIFO = 0             ;  No summation
```

**Example 1:**

All rising edges of probe 1 must be recorded on a path between X0 and X100. It is assumed that no more than 22 edges will occur.

**Program 1:**

```
DEF INT NUMBER
DEF INT INDEX_R
N0   G0 X0                               ;  Mode = 1, simultaneous
N1   MEAC[X]=( 1, 1, 1) POS[X]=100       ;  No-FIFO = 1
                                         ;  Trigger event 1 = rising edge,
                                         ;  Probe 1
OC2  STOPRE                              ;  Stop preprocessing
N3   MEAC[X]=(0)                         ;  Abort continuous measurement
N4   NUMBER = $AC_FIFO1[4]               ;  Number of measured values received in
                                         ;  of the FIFO variables
N5   NUMBER = NUMBER - 1
N6   FOR INDEX_R= 0 TO NUMBER
N7   R[INDEX_R]= $AC_FIFO1[0]            ;  Enter FIFO content in R0 - ...
N8   ENDFOR                              ;  After reading the FIFO variable is empty
```

Example 2:

All rising and falling edges of probe 1 must be recorded on a path between X0 and X100. The number of trigger events that may occur is unknown. This means: The measured values must be fetched and stored in ascending and descending order in R1 as a parallel operation in a synchronized action. The number of stored measured values is entered in R0.

**Program 2:**

```
N0   G0 X0                                   ;  Rapid traverse to the starting position
N1   $AC_MARKER[1]=1                         ;  Marker 1 as index for arithmetic variable R[..]
OC2  ID=1 WHENEVER $AC_FIFO1[4]>=1           ;  Synchronized action as check:
     DO $R[$AC_MARKER[1]]= $AC_FIFO1[0]
     $AC_MARKER[1]=$AC_MARKER[1]+1
                                             ;  when 1 or more measured values are present in FIFO variable
                                             ;  read the oldest value from FIFO and store in current R[..]
                                             ;  increment index for R by 1
N3   MEAC[X]=( 1, 1, 1, -1) POS[X]=100       ;  Activate continuous measurement, motion
                                             ;  to X = 100
                                             ;  Mode = 1, simultaneous
                                             ;  No_FIFO = 1; trigger event 1= 1,
                                             ;  rising edge probe 1
                                             ;  Trigger event 2= -1, falling edge probe 1
N4   MEAC[X]=(0)                             ;  Deselect measurement
N5   STOPRE                                  ;  Stop preprocessing
N6   R0 = $AC_MARKER[1]                      ;  Number of values recorded in R0
```

Example 3:

Continuous measurement with explicit deletion of distance-to-go after 10 measurements

Program 3:

```
N1   WHEN $AC_FIFO1[4]>=10                   ;  Final condition as synchronized action:
     DO MEAC[X]=(0) DELDTG(X)                ;  When 10 or more measured values are present
                                             ;  in
                                                FIFO variable
                                             ;  are present,
                                             ;  deselct continuous measurement and
                                             ;  Delete distance-to-go
OC2  MEAC[X]=( 1,1,1,-1) G01 X100 F500       ;  continuous measurement active from the
                                             ;  part program.
                                             ;  Mode = 1, simultaneous
                                             ;  No_FIFO = 1, FIFO-Variable 1
                                             ;  Trigger event 1= 1, rising edge,
                                             ;  Probe 1
                                             ;  Trigger event 2= -1, falling edge
                                             ;  probe 1
N3   MEAC[X]=(0)                             ;  Deselect continuous measurement
N4   R0= $AC_FIFO1[4]                        ;  actual number of measurement values
```

### Priority with more than one measurement

Only one measurement task can be active for an axis at any given time.

If a measurement job for the same axis is started, the trigger events are reactivated and the measurement results reset. The system does not react in any special way if Deactivate measurement job (mode 0) is programmed when no measurement job has been activated beforehand. Measurement jobs started from the part program cannot be influenced from synchronized actions. An alarm is generated if a measurement job is started for an axis from a synchronized action when a measurement job from the part program is already active for the same axis. If a measurement job is already in progress from a synchronized action, a measurement job from the part program cannot be started at the same time.

### Measurement tasks and state changes

When a measurement task has been executed from a synchronized action, the control system responds in the following way:

| Status | Response |
|---|---|
| Mode change | A measurement job activated by a modal synchronized action is not affected by a change in operating mode. It remains active beyond block limits. |
| RESET | The measurement task is aborted |
| Block search | Measurement tasks are collected, but not activated until the programmed condition is fulfilled. |
| REPOS | Activated measurement tasks are not affected. |
| End of program | Measurement tasks started from static synchronized actions remain active. |

## 2.4.21 Setting and deleting wait markers for channel synchronization

### Introduction

The coordination of operational sequences in channels is described in:

**References:**

/FB1/ Function Manual, Basic Functions; Mode Group, Channel, Program Operation Mode (K1)

The following of the functions described in this document, may be legally used in synchronized actions:

### Set wait marker

The command SETM (marker number) can be programmed in the part program and the action section of a synchronized action. It sets the marker (marker number) for the channel in which the command is applied (own channel).

### Delete wait marker

The command CLEARM (marker number) can be programmed in the part program and the action section of a synchronized action. It deletes the marker (marker number) for the channel in which the command is applied (own channel).

## 2.4.22 Set alarm/error reactions

### Fault situations

Setting an alarm is one way of reacting to error states.

**Application:**

The SETAL command can be programmed to set cycle alarms from synchronized actions.

The following reactions can also be programmed as a response to errors:

- For stopping the axis, refer to Subsection 2.4.12 "Disabling a Programmed Axis Movement"
- For setting the output, refer to Subsection 2.4.2 "Setting (Writing) and Reading of real-time variables"
- Other actions listed in Section 2.4 "Actions in Synchronized Actions"

### Example set alarm

```
D=67 WHENEVER $AA_IM[X1] - $AA_IM[X2] < 4.567 DO SETAL(61000)
                ;   Set alarm, when distance (actual value of the axis X1 - actual value of the axis X2)
                ;   falls short of the critical value 4.567.
```

### Cycles and cycle alarms

For information about cycles and cycle alarms, please see

**References:**

/PGZ/Programming Manual Cycles

## 2.4.23 Evaluating data for machine maintenance

### Function

Machine operators are able to use system variables in part programs, synchronized actions and via the BTSS interface (even from a PLC or HMI) to access information about the use of the machine.

Maintenance measures can then be taken directly or requested on the basis of the values read out.

### Saving

The system variables for machine maintenance are stored in SRAM. This means that they are retained after POWER ON.

**Note**

In contrast, the lubricant signal is only ever set if an axis path stored in a machine data has been exceeded since POWER ON. See:

**References:**

/FB1/ Function Manual Basic Functions; Various NC/PLC-Interface signals and functions (A2), Chapters "Signals from Axis/Spindle".

### Availability

The values for machine maintenance are available if the global NCK machine data is set:

MD18860 $MN_MM_MAINTENANCE_MON (Activate recording of maintenance data)

And when axis-specific machine data has been used to indicate which data should be provided for each axis involved.

The following machine data is used to activate the function of and prepare the memory for the values indicated in the axis-specific MD. Changes of the following machine data become effective with Power On.

MD18860 $MC_MM_MAINTENANCE_MON

Axis-specific values:

The following can be specified in MD33060 $MA_MAINTENANCE_DATA using bit coding:

| | |
|---|---|
| Bit 0: | Total travel distance, total travel time and travel count of the axis |
| Bit 1: | Total travel distance, total travel time and travel count at **high speeds** of the axis High speeds are = 80% of the maximum axis speed |
| Bit 2: | Total axis jerk, travel time with jerk and travel count with jerk |
| Bits 3 - 15: | Reserved |

### Configuration example

MD18860 $MN_MM_MAINTENANCE_MON = TRUE (Activate recording of maintenance data)

MD33060 $MA_MAINTENANCE_DATA[0]=1 (Config. of the recording of maintenance data)

MD33060 $MA_MAINTENANCE_DATA[1]=1

MD33060 $MA_MAINTENANCE_DATA[2]=1

... Activates the system variables for total travel distance, total travel time and travel count for the first 3 axes

### System variables

The following system variables can be read from the part program and from synchronized actions:

| | |
|---|---|
| $AA_TRAVEL_DIST | Total travel distance in mm or degrees |
| $AA_TRAVEL_TIME | Total travel time in seconds |
| $AA_TRAVEL_COUNT | Total travel count |
| $AA_TRAVEL_DIST_HS | Total travel distance at high speeds in mm or degrees |
| $AA_TRAVEL_TIME_HS | Total travel time in seconds at high speeds |
| $AA_TRAVEL_COUNT_HS | Total travel count at high speeds |
| $AA_JERK_TOT | Total axis jerk in $m/s^3$ |
| $AA_JERK_TIME | Axis travel time with jerk in seconds |
| $AA_JERK_COUNT | Axis travel count with jerk |

### Example Distance during part program processing

Repeat read-outs can be used for example to determine the total travel distance of an axis within an area of a part program.

```
; Start of processing area in part program
R1 = $AA_TRAVEL_DIST[X]
…
…                                   ;   End of processing area
R2 = $AA_TRAVEL_DIST[X]
R3 = R2 -R1                         ;   Total traverse path of the X axis
                                    ;   during the processing of the
                                    ;   processing area in part program
```

# 2.5 Call of Technology Cycles

## Definition

A technology cycle is a sequence of actions that are executed sequentially in the interpolation cycle. The actions listed in the Chapter "Actions in synchronized actions" can be grouped together to programs. From the user's point of view, these programs are subprograms without parameters.

## Parallel processing in channel

Several technology cycles or actions can be processed simultaneously in the same channel. These cycles and actions are processed in parallel in the channel in one interpolation cycle.

## Various processing methods

With respect to processing sequence, the user must select the most suitable method from the following options:

- Several actions in one synchronized action:

  All actions are executed simultaneously in the interpolation cycle in which the condition is fulfilled.

- Actions are grouped together to form a technology cycle:

  The actions in the technology cycle are processed sequentially in the interpolation cycle. One block is processed in each interpolation cycle. A distinction must be made between single-cycle and multi-cycle actions. A technology cycle is ended when its last action has been executed (generally after several interpolation cycles have passed).

Commands such as variable assignments in technology cycles are processed in **one** interpolation cycle. Other commands (e.g. motion of a command axis, see Chapter "Starting of command axes") last for **several** interpolation cycles. If the function is completed (e.g. exact stop on positioning of axis), the next block is executed in the following interpolation cycle.

Each block requires at least one interpolation cycle. If a block contains several single-cycle actions, then these are all processed in one interpolation cycle. Fig. "Multiple technology cycles" provides examples to indicate which actions are single-cycle and which are multi-cycle.

## Application

One possible application of technology cycles is to move each axis using a separate axis program.

## Programming

A technology cycle can be activated as a function of a condition in a modal/static synchronized action.

End of program is programmed with `M02 / M17 / M30 / RET`.

## Search path

The call search path is the same as for subprograms and cycles.

**Example**

```
…
ID=1 EVERY $AA_IM[Y]>=10 DO AX_X                ; AX_X Sub-program-
                                                ; name for axis program for X axis
                      AX_X:                     ; Axis program:
                      POS[X]=$R[7] FA[X]=377
                      $A_OUT[1] = 1
                      POS[X]=R10
                      POS[X]=-90
                      M30
```

**Note**

If the condition is fulfilled again while the technology cycle is being processed, the cycle is not restarted. If a technology cycle has been activated from a synchronized action of the WHENEVER type and the relevant condition is still fulfilled at the end of the cycle, then the technology cycle will be restarted.

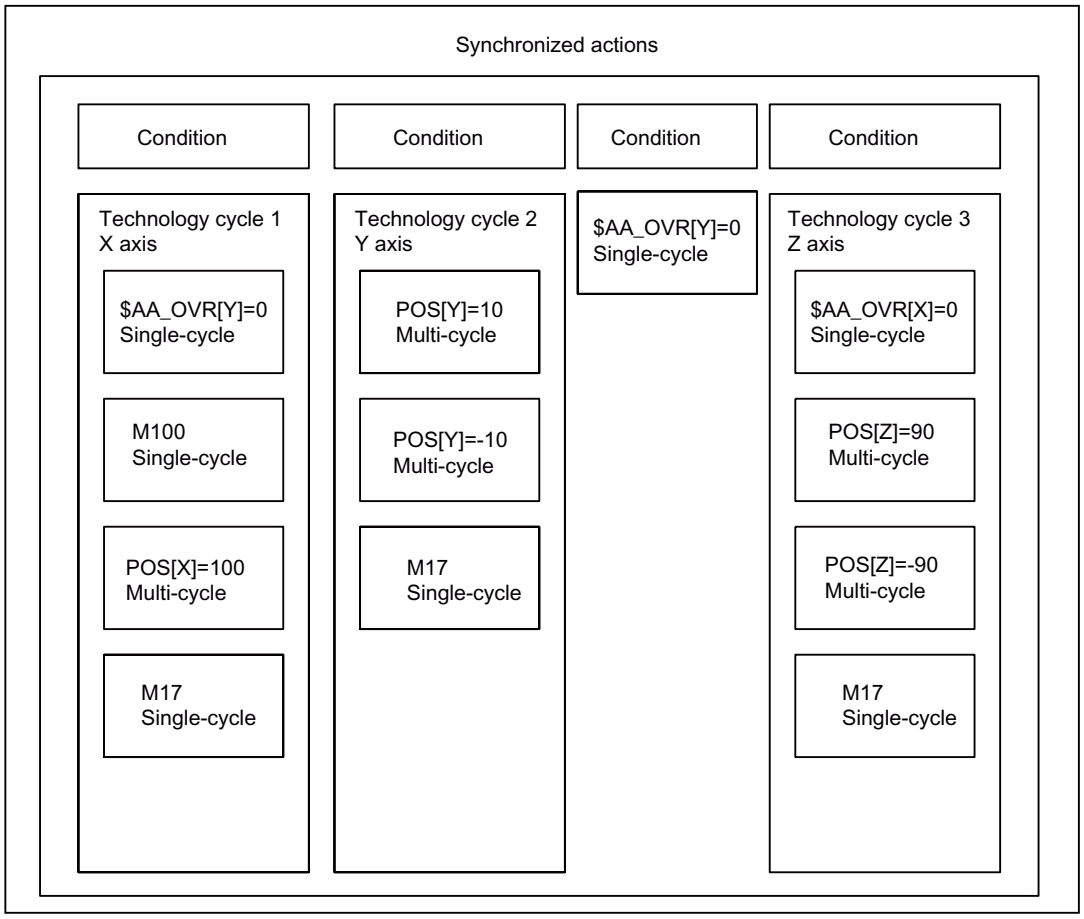



Figure 2-9 Several technology cycles

**Example (2) for coordinated axis motions:**

Different axis programs can be started by setting digital NC inputs.

Main program:

```
...
ID=1 WHEN $A_IN[1]==1 DO AXIS_X
ID=2 WHEN $A_IN[2]==1 DO AXIS_Y
ID=3 WHEN $A_IN[3]==1 DO AA_OVR[Y]=0
ID=4 WHEN $A_IN[4]==1 DO AXIS_Z
M30
```

Axis programs:

AXIS_X:

```
$AA_OVR[Y]=0
M100
POS[X]=100
M17
```

AXIS_Y:

```
POS[Y]=10
POS[Y]=-10
```

```
M17
AXIS_Z:
$AA_OVR[X]=0
POS[Z]=90
POS[Z]=-90
M17
```

## 2.5.1 Coordination of synchronized actions, technology cycles, part program (and PLC)

### Control of technology cycles

Technology cycles/synchronized actions are controlled via the identification number of the synchronized action in which they are programmed as an action:

### Means of coordination

| Keyword | Description | TP | SA |
|---|---|---|---|
| | Call permitted in the part program<br>Call permitted in Synchr. action/technology cycle | + | + |
| LOCK(ID) | Disable technology cycle.<br>An active action is interrupted. | | + |
| UNLOCK(ID) | UNLOCK continues the technology cycle at the point of interruption. An interrupted positioning operation is continued. | | + |
| RESET(ID) | Abort technology cycle. Active positioning operations are aborted. If the technology cycle is restarted, then it is processed from the 1st block in the cycle.<br>Depending on the type of synchronized action, actions are executed once more when the condition is fulfilled again. Already executed synchronized actions of the WHEN type are not processed again after RESET. | | + |
| CANCEL(ID) | Synchronized action is deleted. | + | |

- LOCK(ID), UNLOCK(ID) by PLC, see Subsection 2.6.1 "Control of PLC".

**Note**

A synchronized action contains a technology cycle call. No further actions may be programmed in the same block This ensures that the assignment between ID number and relevant technology cycle is unambiguous.

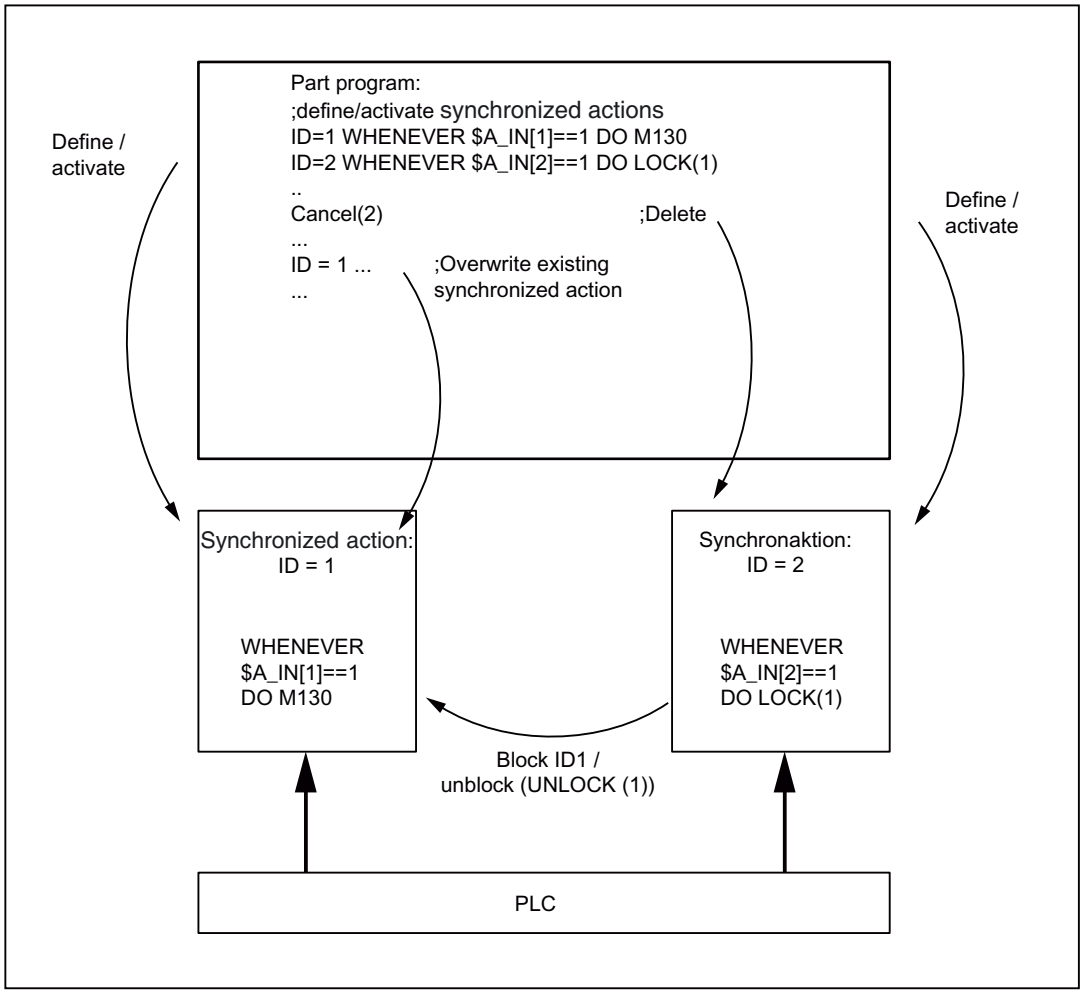



Figure 2-10 Setting up/locking modal synchronized actions/deleting

# 2.6 Control and protection of synchronized actions

## 2.6.1 Control via PLC

### Function

**Modal** synchronized actions (ID, IDS) can be locked or enabled from the PLC.

- Disabling of all modal synchronized actions
- Selective disabling of individual synchronized actions

### Sphere of influence

The PLC can control maximum up to the first 64 modal synchronized actions by applying disables (ID, IDS 1-64). The synchronized actions, that can be **inhibited** by the PLC are designated by the NC with 1 in the 64-bit field of the following interface:

DB21, … DBB308-315

Protected synchronized actions are never tagged as being possible to disable. See Chapter 2.6.2 "Protected synchronized actions".

### Disable all synchronized actions

The PLC-application program can exclude all modal synchronized actions defined and saved in the NC from activation by setting the NC/PLC-interface:

DB21, … DBX1.2 (synchronized action off)

Protected synchronized actions are an exception. Please see Subsection "Protected synchronized actions".

PLC cancels the total disabling again by setting the NC/PLC-interface to 0:

DB21, … DBX1.2.

### Application of selective disabling

One bit is reserved for each of first 64 IDs (1-64) in the PLC interface:

DB21, … DBX300.0 (disable synchronized action No. 1)

to

DB21, … DBX307.7 (disable synchronized action No. 63)

These functions are enabled by default (bits = 0). When the allocated bit is set, evaluation of the condition and execution of the associated function are disabled in the NCK.

### Cancellation of selective disabling

A previously disabled syncronized action is cleared again by the PLC by setting the ID-, IDS-number of the corresponding bit to 0 in the interface:

DB21, … DBX300.0 (disable synchronized action No. 1)

to

DB21, … DBX307.7 (disable synchronized action No. 64)

### Updating the selective disabling

If the PLC-application program has made changes in the range from DB21-30,DBB 300. Bit 0 to DB21-30 BB307 Bit 7, it must activate them with:

DB21, … DBX280.1

### Selective disabling status signal

If selective disablings are activated by the NCK, it is signalled in the interface:

DB21, … DBX.281.1 (disable synchronized action)

**References:**

/LIS2/ Lists (Volume 2); Interface solution line or power line.

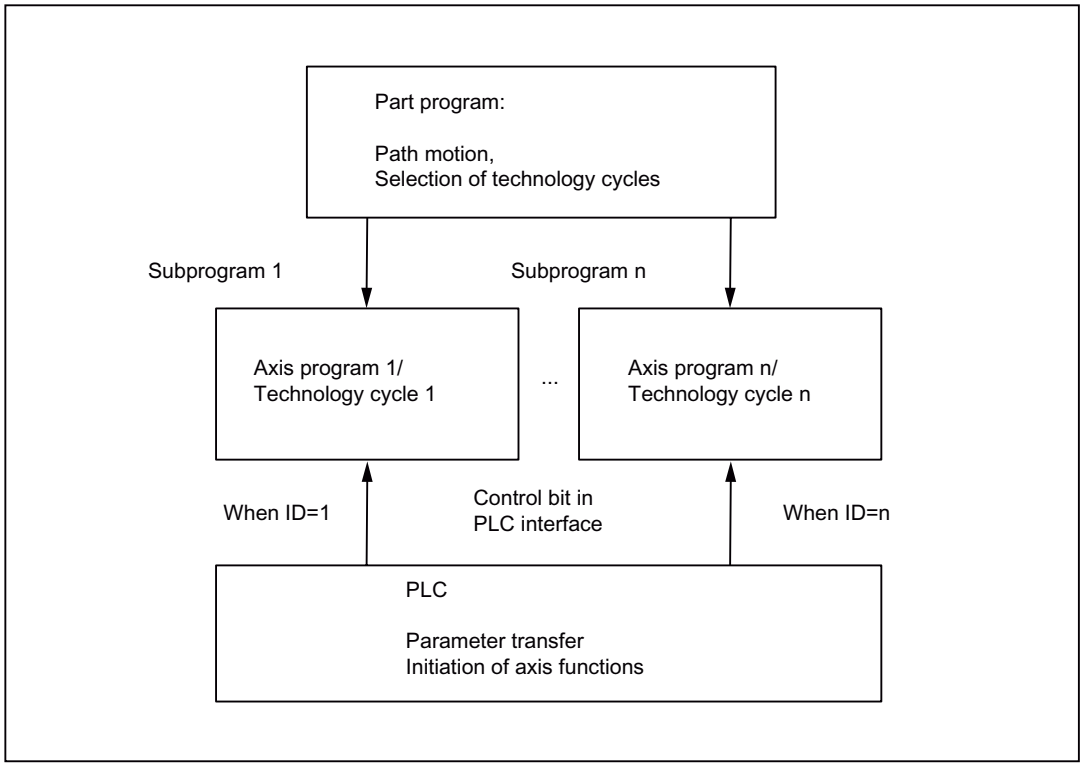



Figure 2-11 Axis programs/technology cycles

### Reading/writing of PLC data

PLC data can also be read and written from the part program by transferring parameters between the NCK and PLC via the VDI interface. This is an option: PLC variables.

**References:**

/FB1/ Function Manual, Basic Functions, Basic PLC Program (P3)

Parameters can also be accessed from synchronized actions, thus allowing PLC data to be transferred to the NCK for parameterization before an axis function is initiated. The system variables to be addressed can be found in

**References:**

/PGA1/ Parameter Manual, System Variables

## 2.6.2 Protected synchronized actions

### Global protection

An area of write-protected synchronized actions can be defined via the machine data:

MD11500 $MN_PREVENT_SYNACT_LOCK (protected synchronized action)

Synchronized actions with ID numbers within the protected area, once they are defined, can **no longer be**:

- Overwritten
- Deleted (`CANCEL`) or
- Disabled (`LOCK`).

Protected synchronized actions cannot be disabled via the PLC either. They are indicated to the PLC as non-lockable in the interface. See Chapter "Control via PLC".

**Note**

The functionality is also used for Safety Integrated systems.

### Applications

The end customer must be prevented from modifying reactions to certain states defined by the machine manufacturer.

The machine is commissioned by the manufacturer without protection. This enables the gating logic to be defined and tested. However, the manufacturer declares the range of synchronized actions he has used as protected before the system is delivered to the end customer, thus preventing the end customer from defining his own synchronized actions within this protected area.

### Notation of the machine data: MD11500 $MN_PREVENT_SYNACT_LOCK

```
MD11500 $MN_$MN_PREVENT_SYNACT_LOCK[0]= i   ; i number of the 1st ID to be disabled
MD11500 $MN_PREVENT_SYNACT_LOCK[1]= j       ; i number of the last ID to be disabled
```

ID i and j can also be inverted.

If i = 0 and j = 0, no synchronized actions are protected.

### Channel-specific protection

An area of write-protected synchronized actions can be defined for the channel via the channel-specific machine data:

MD21240 $MN_PREVENT_SYNACT_LOCK (protected synchronized action)

Synchronized actions with ID numbers within the protected area, once they are defined, can **no longer be**:

- Overwritten
- Deleted (`CANCEL`) or
- Disabled (`LOCK`).

Protected synchronized actions cannot be disabled via the PLC either. They are indicated to the PLC as non-lockable in the interface. See Chapter "Control via PLC".

### Application

See above.

### Notation of the MD21240 $MC_PREVENT_SYNACT_LOCK_CHAN

```
CHANDATA(C)                                  ; with C channel number
MD21240 $MC_PREVENT_SYNACT_LOCK_CHAN[0]= k   ; k number of the first ID to be disabled for the channel
MD21240 $MC_PREVENT_SYNACT_LOCK_CHAN[1]= l   ; l number of the last ID to be disabled for the channel ID
```

k and l can also be inverted.

If k = 0 and l = 0, no synchronized actions are protected.

When k = -1 and l = -1, this specifies that the global range of protected synchronized actions should be applicable for the channel and should be defined with the following machine data:

MD11500 MN_PREVENT_SYNACT_LOCK (protected synchronized action)

**Note**

Protection for synchronized actions must be cancelled while protected static synchronized actions are being defined, otherwise POWER ON will have to be executed for every alteration to allow redefinition of the logic.

The effectiveness of the disable is identical, regardless of whether it is specified as:

- global disabling or
- channel-specific disabling.

## Example

In a system with 2 channels, synchronized actions should be protected as follows:

IDs 20 to 30 should be protected in the first channel and IDs 25 to 35 in the second. A combination of global and channel-specific specifications can be used.

```
MD11500 $MN_PREVENT_SYNACT_LOCK[0] = 25          ;  global specification
MD11500 $MN_PREVENT_SYNACT_LOCK[1] = 35          ;  global specification
CHANDATA(1)
MD21240 $MC_PREVENT_SYNACT_LOCK_CHAN[0] = 20
          ;  in 1st channel only the channel-specific (first ID number to be protected) is effective
             in the first channel
MD21240 $MC_PREVENT_SYNACT_LOCK_CHAN1] = 30
          ;  in 1st channel only the channel-specific MD (last ID number to be protected) is
             effective in the first channel
CHANDATA(2)
MD21240 $MC_PREVENT_SYNACT_LOCK_CHAN[0] = -1
          ;  in 2nd channel the global machine data MD11500 is effective
          ;  $MN_PREVENT_SYNACT_LOCK!
MD21240 $MC_PREVENT_SYNACT_LOCK_CHAN[1] = -1
...
```

# 2.7 Control system response for synchronized actions in specific operational states

## 2.7.1 Power On

No synchronized actions are active during POWER ON. Static synchronized actions that are required to be active immediately after POWER ON must be activated within an ASUB started by the PLC.

**References:**

/FB1/ Function Manual, Basic Functions; Basic PLC Program

/FB1/ Function Manual, Basic Functions; Mode Group, Channel, Program Operation Mode (K1)

This requires ASUB functionality in all operating modes.

Examples:

- Adaptive control
- Safety Integrated, gating logic formulated by means of synchronized actions

## 2.7.2 RESET

### In case of motion of positioning axis

All positioning motions initiated by synchronized actions are aborted on NC reset. Active technology cycles are reset.

### ID

Synchronized actions programmed locally (i.e. with ID=...) are deselected on NC reset.

### IDS

Static synchronized actions (programmed with IDS = ...) remain active after NC reset. Motions can be restarted from static actions after NC reset.

### Other reactions, dependent on actions

RESET continued

| | | |
|---|---|---|
| **Synchronized action/ technology cycle** | Modal and non-modal active action is aborted, synchronized actions are cancelled | Static (IDS)<br>Active action is aborted, technology cycle is reset |
| **Axis/ positioning spindle** | Motion is aborted | Motion is aborted |
| **Speed-controlled spindle** | MD35040 $MA_SPIND_ACTIVE_AFTER_RESET== TRUE:<br>Spindle remains active<br>MD35040 $MA_SPIND_ACTIVE_AFTER_RESET==FALSE:<br>Spindle stops. | MD35040 $MA_SPIND_ACTIVE_AFTER_RESET== TRUE:<br>Spindle remains active<br>MD35040 $MA_SPIND_ACTIVE_AFTER_RESET==FALSE:<br>Spindle stops. |
| **Master value coupling** | MD20110 $MC_RESET_MODE_MASK, Bit13 == 1:<br>Master value coupling remains active<br>MD20110 $MC_RESET_MODE_MASK, Bit13 == 0:<br>Master value coupling is separated | MD20110 $MC_RESET_MODE_MASK, Bit13 == 1:<br>Master value coupling remains active<br>MD20110 $MC_RESET_MODE_MASK, Bit13 == 0:<br>Master value coupling is separated |
| **Measuring operations** | Measuring operations started from synchronized actions are aborted | Measuring operations started from static synchronized actions are aborted |

## 2.7.3 NC STOP

### Motion start from static synchronized actions

Motions that have been started from static synchronized actions remain active in spite of an NC STOP.

Response of a command axis in SW 6.3 and later:

**Note**

In SW 6.3 and later, it is possible to convert a command axis started by a static synchronized action to a PLC-controlled axis. VDI interface:

DB 31, ... DBX28.7 (PLC controls axis/P5)

Such an axis is **no longer** stopped through NC-STOP , but instead through an axial STOP.

### Motion start from non-modal and modal synchronized actions

Axis motions started from non-modal and modal actions are interrupted and then restarted by NC START. Speed-controlled spindles remain active.

Synchronized actions programmed in the current block remain active.

Example

Set output: `... DO $A_OUT[1] = 1`

## 2.7.4 Mode change

The response differs depending on whether the relevant synchronized action is static or programmed locally. Synchronized actions activated by keyword **IDS** remain active after a change in operating mode. All other synchronized actions are deactivated in response to a mode change and reactivated on switchover to AUTO mode for repositioning.

**Example**

```
N10     WHEN $A_IN[1]==1 DO DELDTG
N20     G1      X10 Y 200 F150 POS[U]=350
```

Block N20 contains a STOP command. The operating mode is switched to JOG. If deletion of distance-to-go was not active prior to the interruption, then the synchronized action programmed in block N10 is reactivated when AUTO mode is selected again and the program continued.

## 2.7.5 End of program

Static synchronized actions remain active after the end of program. Modal and non-modal synchronized actions are aborted. Static and modal synchronized actions programmed in M30 blocks remain active. They can be aborted with CANCEL before the M30 block. Polynomial coefficients programmed with FCTDEF remain active after the end of program.

## 2.7.6 Response of active synchronized actions to end of program and change in operating mode

See Chapter 2.7.4 "Change of operation mode" and Chapter 2.7.5 "Program end".

| | | |
|---|---|---|
| **Synchronized action/ technology cycle** | Modal and non-modal actions are aborted | Static actions (IDS) remain active |
| **Axis/positioning spindle** | M30 is delayed until the axis/spindle is stationary. | Motion continues |
| **Speed-controlled spindle** | End of program:<br>MD35040 $MA_SPIND_ACTIVE_AFTER_RESET== TRUE:<br>Spindle remains active<br>MD35040 $MA_SPIND_ACTIVE_AFTER_RESET==FALSE:<br>Spindle stops.<br>Spindle remains active when the operation mode is changed | Spindle remains active |
| **Master value coupling** | MD20110 $MC_RESET_MODE_MASK, Bit13 == 1:<br>Master value coupling remains active<br>MD20110 $MC_RESET_MODE_MASK, Bit13 == 0:<br>Master value coupling is separated | A coupling started from a static synchronized action remains active |
| **Measuring operations** | Measuring operations started from synchronized actions are aborted | Measuring operations started from static synchronized actions remain active |

## 2.7.7 Block search

### General

Synchronized actions in the program, which have been interpreted during the block search, are collected but their conditions are not evaluated. No actions are executed. Processing of synchronized actions does not commence until NC Start.

### IDS

Synchronized actions that are programmed with keyword IDS and are already active remain operative during the block search.

### Polynomial coefficient

Polynomial coefficients programmed with FCTDEF are collected **with calculation** during a block search, i.e. they are written to system variables.

## 2.7.8 Program interruption by ASUB

### ASUB start

Modal and static motion-synchronous actions remain active and are also operative in the asynchronous subprogram.

### ASUB end

If the asynchronous subprogram is not continued with REPOS, then modal and static motion-synchronous actions modified in the subprogram remain operative in the main program.

Positioning motions started from synchronized actions respond in the same way as to operating mode switchover:

Motions started from non-modal and modal actions are stopped and continued with REPOS (if programmed). Motions started from static synchronized actions continue uninterrupted.

## 2.7.9 REPOS

In the remainder of the block, the synchronized actions are treated in the same way as in an interruption block.

Modifications to modal synchronized actions in the asynchronous subprogram are not effective in the interrupted program.

Polynomial coefficients programmed with FCTDEF are not affected by ASUB and REPOS.

The coefficients from the call program are applied in the asynchronous subprogram. The coefficients from the asynchronous subprogram continue to be applied in the call program.

If positioning motions started from synchronized actions are interrupted by the operating mode change or start of the interrupt routine, then they are continued with REPOS.

## 2.7.10 Response to alarms

Axis and spindle motions started by means of synchronized actions are decelerated in response to an alarm involving a motion stop instruction. All other actions (such as Set output) continue to be executed.

If an alarm is activated by a synchronized action, then the action is no longer processed in the next interpolation cycle, i.e. the alarm is output only once. Alarms that respond with an interpreter stop only take effect once the precoded blocks have been processed.

Processing of all other actions continues as normal.

If a technology cycle generates an alarm with motion stop, then processing of the relevant cycle ceases.

# 2.8 Configuration

## 2.8.1 Configurability

### Number of synchronized action elements

The number of programmable synchronized action blocks depends entirely on the configurable number of synchronized action elements. The number of storage elements for motion-synchronous actions (synchronized action elements) is defined via the machine data:

MD28250 $MC_MM_NUM_SYNC_ELEMENTS (Number of elements for expressions in synchronized actions)

This data can be set irrespective of the number of blocks available in the control system, thus enabling the complexity of expressions evaluated in real time as well as the number of actions to be set flexibly.

### Use of elements

Each **one** synchronized action element is needed for:

- A comparison expression in a condition
- An elementary action
- the synchronized action block

**Example**

A total of four elements is needed for the synchronized action block below.

```
WHENEVER ($AA_IM[x] > 10.5) OR ($A_IN[1]==1) DO
|_______|    |_______________|    |_____________|
Element 1     Element 2            Element 3
$AC_PARAM[0]=$AA_im[y]+1
|_____________________|
Element 4
```

The default value of the following machine data is selected such that it is possible to activate the maximum presetting of max. 16 synchronized actions for SW 3 and earlier.

MD28250 $MC_MM_NUM_SYNC_ELEMENTS (Number of elements for expressions in synchronized actions)

**Note**

If the user does not wish to program any synchronized actions, then he can reset the value to 0 in the machine data:

MD28250 $MC_MM_NUM_SYNC_ELEMENTS

In this way, around 16 kByte of DRAM memory can be saved.

### Display

The status display for synchronized actions (see Section 2.9 "Diagnostics only with HMI Advanced") indicates how much of the memory provided for synchronized actions is still available. This status can also be read from synchronized actions in variable $AC_SYNA_MEM.

### Alarm

An alarm is generated if all available elements are used up during program execution. The user can respond by increasing the number of synchronized action elements or by modifying his program accordingly.

### Number of FCTDEF functions

The number of programmable FCTDEF functions for each block can be configured via the following machine data:

MD28252 $MC_MM_NUM_FCTDEF_ELEMENTS (number of FCTDEF elements)

The default value for all types of control is 3. The control-specific maximum value can be found in:

**References:**

/LIS1/ Lists (Book 1); "MD/SD Lists".

### Interpolation cycle

The time required on the interpolation level increases with the number of synchronized actions programmed. It may be necessary for the start-up engineer to lengthen the interpolation cycle accordingly.

## Guide values for lengthening interpolation cycle

As a guide, individual times required to perform operations within synchronized actions (measured on an 840D with NCU 573.x) are given below:

Times may be different for other control types.

| **NC language** | **Time requirement** | |
|---|---|---|
| | Total | **Text in bold print** |
| Base load for a synchronized action, if the condition is not fulfilled:**WHENEVER FALSE** DO $AC_MARKER[0]=0 | 10 µs | **~10 µs** |
| Read variable: WHENEVER **$AA_IM[Y]>10** DO $AC_MARKER[0]=1 | 11 µs | **~1 µs** |
| Write variable: DO *$R2=1* | 11-12 µs | **~1-2 µs** |
| Read/write setting data:<br>DO**$$SN_SW_CAM_MINUS_POS_TAB_1[0]=20** | 24 µs | **~14 µs** |
| Basic arithematic operation, e.g. multiplication: DO $R2=**$R2*2** | 22 µs | **~12 µs** |
| Trigonometric functions (e.g. cos): DO $R2=**COS($R2)** | 23 µs | **~13 µs** |
| Start positioning axis movement: WHEN TRUE DO **POS[z]=10** | 83 µs | **~73 µs** |

# 2.9 Diagnostics (only with HMI Advanced)

## Diagnostic functionality

The following special test tools are provided for diagnosing synchronized actions:

- Status display of synchronized actions in the machine operator area
- System variables display parameters in the operating range

  The current values of all synchronized action variables can be displayed (displaying main run variables)
- System variables log parameters in the operating range

  Characteristics of variables can be recorded in the interpolation cycle grid (logging main run variables)

This functionality is structured in the operator interface in the following way:

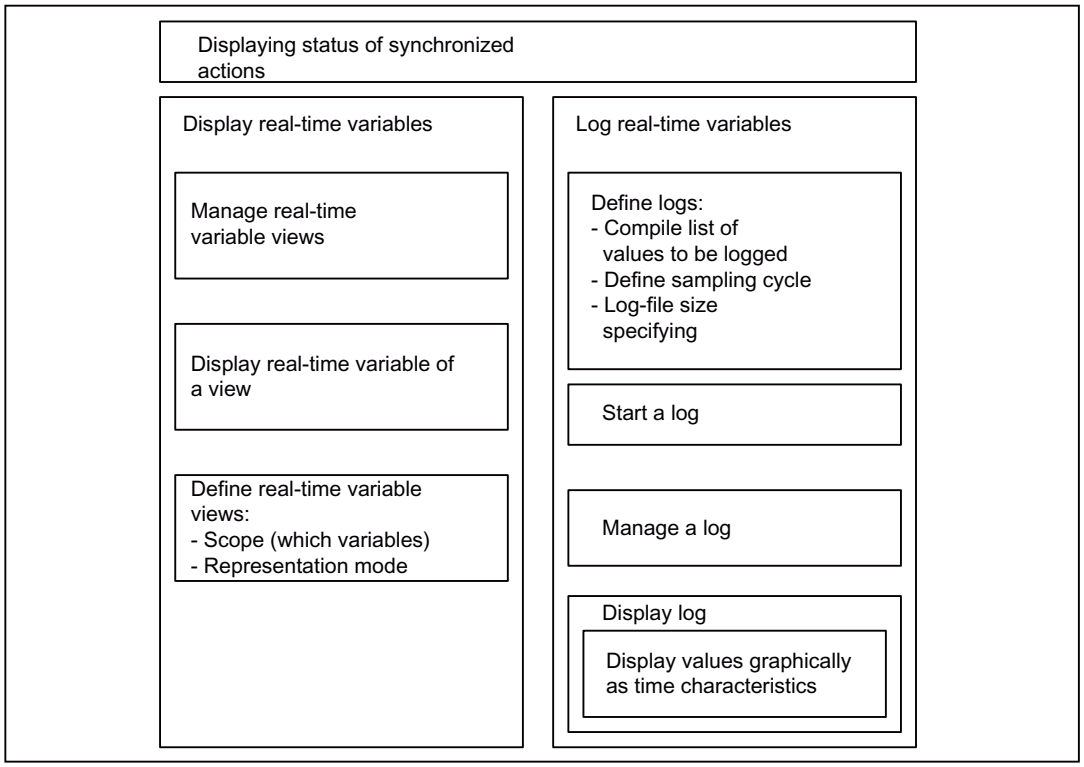



Figure 2-12 Functionality of test tools for synchronized actions

For a description of how to use these functions, please see:

**References:**

/BAD/ Operator's Guide HMI Advanced.

## 2.9.1 Displaying status of synchronized actions

### Status display

The status display shows:

- Current extract of selected program

All programmed synchronized actions according to:

- Line number
- Code denoting synchronized action type
- ID number of synchronized action (for modal actions)
- Status

### Synchronized action type

The following types of auxiliary function are available:

| | |
|---|---|
| ID | Modal synchronized action |
| IDS | Static modal synchronized action |
| | Non-modal synchronized action for next executable block (in AUTOMATIC mode only) |

### Status

The following status conditions might be displayed:

| | |
|---|---|
| No status: | The condition is checked in the interpolation cycle |
| Blocked | LOCK has been set for the synchronized action |
| Active | Action currently being executed. If the action consists of a technology cycle, the current line number in the cycle is also displayed. |

### Complete synchronized actions

A search function can be used to display the originally programmed line in NC language for each displayed synchronized action.

## 2.9.2 Displaying main run variables

### Description

System variables can be monitored for the purpose of monitoring synchronized actions. Variables, which may be used in this way are listed for selection by the user.

A complete list of individual system variables with ID code W for write access and R for read access for synchronized actions can be found in:

**References:**

/PGA1/ Parameter Manual, System Variables

### Views

"Views" are provided to allow the user to define the values, which are relevant for a specific machining situation and to determine how (in lines and columns, with what text) these values must be displayed. Several views can be arranged in groups and stored in correspondingly named files.

### Managing views

A view defined by the user can be stored under a name of his choice and then called again. Variables included in a view can still be modified (Edit View).

### Displaying main run variable of a view

The values assigned to a view are displayed by calling the corresponding user-defined view.

## 2.9.3 Logging main run variables

### Starting point

To be able to trace events exactly in synchronized actions, it is necessary to monitor the action status in the interpolation cycle.

### Method

The values defined in a log definition are written to a log file of defined size in the specified cycle. Special functions for displaying the contents of log files are provided.

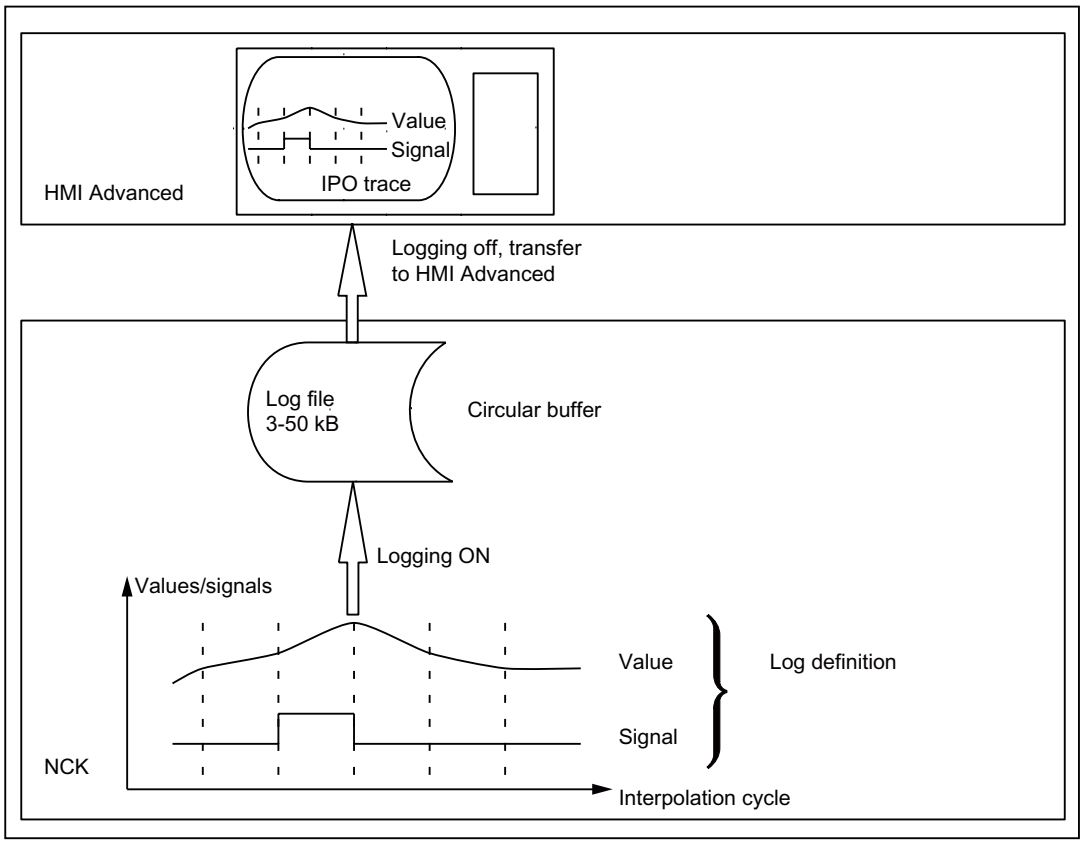



Figure 2-13 Schematic representation of Log main run variables process

## Operation

For information about operating the logging function, please see:

**References:**

/BAD/ Operator's Guide HMI Advanced.

## Log definition

The log definition can contain up to 6 specified variables. The values of these variables are written to the log file in the specified cycle. A list of variables, which may be selected for logging purposes, is displayed. The cycle can be selected in multiples of the interpolation cycle. The file size can be selected in Kbytes. A log definition must be initialized before it can be activated on the NCK for the purpose of acquiring the necessary values.

## Log file size

Values between 3 KB (minimum) and 50 KB (maximum) can be selected as the log file size.

### Storage method

When the effective log file size has been exceeded, the oldest entries are overwritten, i.e. the file works on the circular buffer principle.

### Starting logging

Logging according to one of the initialized log definitions is started by:

- Operation
- Setting system variable $A_PROTO=1 from the part program

The starting instant must be selected such that the variables to be logged are not altered until operations on the machine have been activated. The start point refers to the last log definition to be initialized.

### Stopping logging

This function terminates the acquisition of log data in the NCK. The file containing the logged values is made available on the HMI for storage and evaluation (graphic log). Logging can be stopped by:

- Operation
- Setting system variable $A_PROTO=0 from the part program

### Graphic log function

The measured values (up to 6) of a log are represented graphically as a function of the sampling time. The names of variables are specified in descending sequence according to the characteristics of their values. The screen display is arranged automatically. Selected areas of the graphic can be zoomed.

**Note**

Graphic log representations are also available as text files on the HMI Advanced. An editor can be used to read the exact values of a sampling instant (values with identical count index) numerically.

### Managing logs

Several log definitions can be stored under user-defined names. They can be called later for initialization and start of recording or for modification and deletion.

# Boundary conditions

# 3

## Availability/scope of performance

The scope of performance provided by the "Synchronized actions" function package depends on the following:

- The type of SINUMERIK control system
  - Hardware
  - SW (export/standard versions)
- The availability of functions that can be initiated by "Actions":
  - Standard functions
  - Functions that are available as options

The performance of control systems and their variants as well as functions supplied as options are described in catalogs specific to the SW version:

**References:**

/BU/ Ordering documentation, Catalog NC60 and NC61

/LIS1/ Lists (Book1)

/LIS2/ Lists (Book1)

Further, the functions of the synchronized actions depend on the list of synchronized actions - system variables that can be read/changed - including machine and setting data (also dependent on the SW release). System variables that may be used in conjunction with specific SW versions are described in:

**References:**

/PGA1/ Parameter Manual System Variables (valid for the particular SW release)

## Extensions in SW 4

The following extensions have been introduced with SW 4:

- Diagnostic facilities for synchronized actions
- Availability of additional real-time variables
- Complex conditions in synchronized actions
  - Basic arithmetic operations
  - Functions
  - Indexing with real-time variables
  - Access to setting and machine data
  - Logic operators

- Configurability
  - Number of simultaneously active synchronized actions
  - Number of special variables for synchronized actions
- Activate command axes/axis programs/technology cycles from synchronized actions
- PRESET from synchronized actions
- Couplings and coupled motions from synchronized actions
  - Switch on
  - Switch off
  - Configuring
- Use of measuring functions from synchronized actions
- SW cams
  - Redefinition of position
  - Redefinition of lead times
- Deletion of distance-to-go without preprocessing stop
- Static synchronized actions (modes other than AUTO possible)
- Synchronized actions:
  - Protection against overwriting and deletion
  - Stopping, continuing, deleting
  - Resetting technology cycles
  - Parameterizing, enabling and disabling from PLC
- Overlaid movement/optimized clearance control
- Coordinating channels from synchronized actions
- Starting ASUBs from synchronized actions
- Non-modal auxiliary function output
- All necessary functions for Safety Integrated for formulation of requisite safety-oriented logic operations, protected against changes.
- 16 synchronized actions are included in the basic version

## Expansions in SW 5 and higher

From SW 5 to 7, the following services are additionally provided:

- Synchronized actions, which can be tagged for the PLC
- Availability of additional real-time variables
- Access to PLC I/O (option)
- 255 parallel synchronized actions per channel are possible with the option "Synchronized actions step 2".
- Static synchronized actions IDS that are active beyond the program end and are effective in all operating mode are possible using the option "Inter-mode group actions, ASUBs and synchronized actions".
- Online calculations and online tool offsets (from SW 6).
- Axis replacement using synchronized actions and in the technology cycle (SW 7 onward).
- Creating of coupling modules for the generic coupling (SW 7.4 and higher).

# Signal Descriptions

# 4

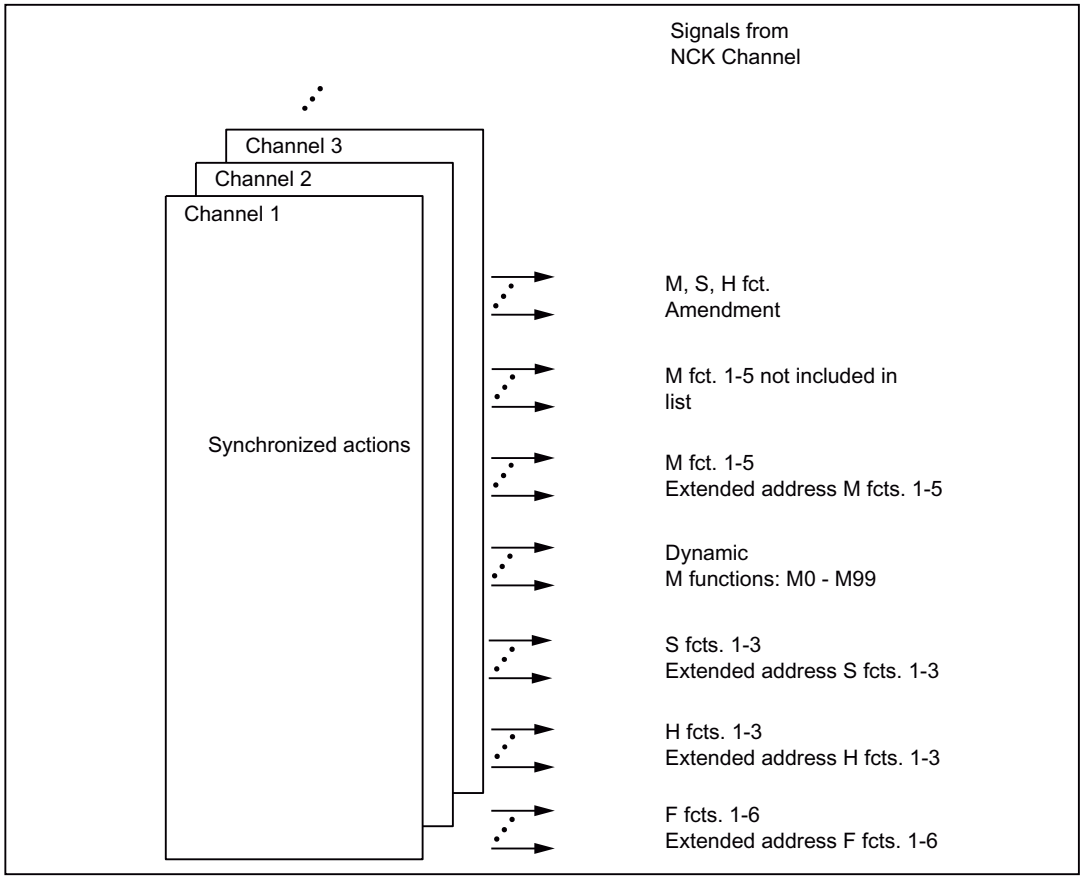



Figure 4-1 PLC interface signals for synchronized actions

For the signals generated by the auxiliary function output from synchronized actions see:
**References:**
/FB1/ Function Manual Basic functions; Auxiliary function output to PLC (H2)

## Signals to channel

With the following signals the PLC application program requests for disabling the assigned synchronized actions:

DB21, … DBX300.0 (disable synchronized action No. 1)

to

DB21, … DBX307.7 (disable synchronized action No. 64)

where:

DBX300.0 of the first modal synchronized action (ID=1/IDS=1) and

DBX307.7 of the 64th modal synchronized action (ID=64/IDS=64).

**Note**

Only the instance (NCK or PLC), which initiated a disable can cancel the disable again.

### Signals from channel

With the following signals the channel shows the PLC application program for the synchronized actions, which may be disabled by PLC:

DB21, … DBX308.0 (disable synchronized action No. 1)

to

DB21, … DBX315.7 (disable synchronized action No. 64)

where:

DBX308.0 of the first modal synchronized action (ID=1/IDS=1) and

DBX315.7 of the 64th modal synchronized action (ID=64/IDS=64).

### Disable all synchronized actions

**All** modal/static synchronized actions, unless protected, are disabled by the global signal:

DB21, … DBX1.2 (synchronized action off)

### Disable selected synchronized actions

DB21, … DBX280.1 (disable synchronized action)

Of the synchronized actions that can be disabled and that are marked in DB21, … DBX308.0 (disable synchronized actions No. 1) to DB21, … DBX315.7 (synchronized actions No. 64), the actions identified in DB 21, … DBX300.0 (disable synchronized actions No. 1) to DB21, … DBX307.7 (synchronized actions No. 64) by the set bit as the ones that can be disabled should only be disabled.

### Synchronized actions disabled

DB21, … DBX281.1 (disable synchronized action)

The required synchronized actions were confirmed by NCK as disabled.

# Examples

# 5

## 5.1 Examples of conditions in synchronized actions

### Path distance from end of block

Axial distance from block end: 10 mm or less (workpiece coordinate system):

```
... WHEN $AC_DTEW <= 10 DO ...
G1 X10 Y20
```

### Axis distance from end of path

```
 ... WHEN $AA_DTEW[X]<= 10 DO ...
POS[X]=10
```

### Path distance from start of block

Path 20 mm or more after start of block in basic coordinate system:

```
...WHEN $AC_PLTBB >= 20 DO ...
```

### Condition with function in comparison

Actual value for axis Y in MCS greater than 10 x sine of value in R10:

```
... WHEN $AA_IM[y] > 10*SIN (R10) DO...
```

### Step-by-step positioning

Every time input 1 is set, the axis position is advanced by one step. The input must be reset again to allow a restart of the system.

```
G91
EVERY $A_IN[1]==1 DO POS[X]= 10
```

### OVR in every interpolation cycle

In order to selectively disable a path motion until a programmed signal arrives, $AC_OVR must be set to zero in every interpolation cycle (keyword WHENEVER).

```
WHENEVER $A_IN[1]==0 DO $AC_OVR= 0
```

### Other system variables

The list of the readable system variables in synchronized actions includes the full set of the values that can be evaluated in the conditions of synchronized actions.

**References:**

/PGA1/ Parameter Manual, System Variables

## 5.2 Reading and writing of SD/MD from synchronized actions

### Infeed and oscillation for grinding operations

Setting data, whose values remain unchanged during machining, are addressed in the part program by their usual names.

**Example**: Oscillation from synchronized actions

```
NC language                                          Comment
N610 ID=1 WHENEVER $AA_IM[Z]>$SA_OSCILL_REVERSE_POS1[Z]
DO $AC_MARKER[1]=0
                                                     ; Always when the current position of the oscillating axis
                                                     ; in the machine coordinate system
                                                     ; Less    the start of reversal area 2
                                                       than
                                                     ; Then    set the axial override of the
                                                     ;         infeed axis to 0
N620 ID=2 WHENEVER $AA_IM[Z]<$SA_OSCILL_REVERSE_POS2[Z]-6
DO $AA_OVR[X]=0 $AC_MARKER[0]=0
                                                     ;; Always when the current position of the oscillating axis
                                                        in the MCS is
                                                     ;         equal to the reversal position 1,
                                                     ; Then    set the axial override of the
                                                     ;         oscillating axis to 0
                                                     ; and     set the axial override of the
                                                     ;         infeed axis to 100% (so that the
                                                     ;         previous synchronized action
                                                     ;         is cancelled!)
N630 ID=3 WHENEVER $AA_IM[Z]==$SA_OSCILL_REVERSE_POS1[Z]
DO $AA_OVR[Z]=0 $AA_OVR[X]=100
                                                     ; Always when the distance-to-go of the part infeed
```

| NC language | Comment |
|---|---|
| | ; Equal to 0, |
| | ; Then set the axial override of the oscillating |
| | ; axis to 100% (so that the previous |
| | ; synchronized action is cancelled!) |
| N640 ID=4 WHENEVER $AA_DTEPW[X]==0 | |
| DO $AA_OVR[Z]=100 $AC_MARKER[0]=1 $AC_MARKER[1]=1 | |
| N650 ID=5 WHENEVER $AC_MARKER[0]==1 DO $AA_OVR[X]=0 | |
| N660 ID=6 WHENEVER $AC_MARKER[1]==1 DO $AA_OVR[X]=0 | |
| | ; if the current position of the oscillating axis in the |
| | ; Workpiece coordinate system |
| | ; Equal to reversal position 1, |
| | ; Then set the axial override of the |
| | ; Oscillating axis to 100% |
| | ; and set the axial override of the |
| | ; infeed axis to 100% (so that the |
| | ; second synchronized action once |
| | ; is cancelled!) |
| N670 ID=7 WHEN $AA_IM[Z]==**$SA_OSCILL_REVERSE_POS1**[Z] | |
| DO $AA_OVR[Z]=100 $AA_OVR[X]=0 | |
| | Setting data, whose value may change during machining (e.g. through an operator input or synchronized action), must be programmed with **$$S**...: |
| | **Example**: Oscillation from synchronized actions with alteration of oscillation position via operator interface |
| N610 ID=1 WHENEVER $AA_IM[Z]>**$$SA_OSCILL_REVERSE_POS1[Z]** | |
| DO $AC_MARKER[1]=0 | |
| | ; Always when the current position of the oscillating axis |
| | ; in the machine coordinate system |
| | ; Less than the start of reversal area 2 |
| | ; Then set the axial override of the |
| | ; infeed axis to 0 |
| N620 ID=2 WHENEVER $AA_IM[Z]<**$$SA_OSCILL_REVERSE_POS2[Z]**-6 | |
| DO $AA_OVR[X]=0 $AC_MARKER[0]=0 | |
| | ; Always when the current position of the oscillating axis |
| | ; in the machine coordinate system |
| | ; Equal to reversal position 1, |
| | ; Then set the axial override of the |
| | ; oscillating axis to 0 |
| | ; and set the axial override of the |
| | ; infeed axis to 100% (so that the |
| | ; previous synchronized action |
| | ; is cancelled!) |

| NC language | Comment |
|---|---|
| N630 ID=3 WHENEVER $AA_IM[Z]==**$$SA_OSCILL_REVERSE_POS1[Z]** | |
| DO $AA_OVR[Z]=0 $AA_OVR[X]=100 | |
| | ; Always when the distance-to-go of the part infeed |
| | ; Equal to 0, |
| | ; Then set the axial override of the |
| | ; oscillating axis to 100% (so that the |
| | ; previous synchronized action |
| | ; is cancelled!) |
| N640 ID=4 WHENEVER $AA_DTEPW[X]==0 | |
| DO $AA_OVR[Z]=100 $AC_MARKER[0]=1 $AC_MARKER[1]=1 | |
| N650 ID=5 WHENEVER $AC_MARKER[0]==1 DO $AA_OVR[X]=0 | |
| N660 ID=6 WHENEVER $AC_MARKER[1]==1 DO $AA_OVR[X]=0 | |
| | If the current position of the oscillating axis in the |
| | Workpiece coordinate system |
| | Equal to reversal position 1, |
| | Then set the axial override of the |
| | Oscillating axis to 100% |
| | and set the axial override of the |
| | ; infeed axis to 100% (so that the |
| | ; second synchronized action once |
| | ; is cancelled!) |
| N670 ID=7 WHEN $AA_IM[Z]==**$$SA_OSCILL_REVERSE_POS1[Z]** | |
| DO $AA_OVR[Z]=100 $AA_OVR[X]=0 | |

## 5.3 Examples of adaptive control

### General procedure

The following examples use the polynomial evaluation function SYNFCT( ).

1. Representation of relationship between input value and output value (real-time variables in each case)
2. Definition of this relationship as polynomial with limitations
3. With position offset: Setting the MD and SD
   - MD36750 $MA_AA_OFF_MODE (Effect of value assignment for axial override in case of synchronized actions)
   - SD43350 $SA_AA_OFF_LIMIT (optional) (Upper limit of the offset value $AA_OFF in case of clearance control)
4. Activation of the control in a synchronized action

## 5.3.1 Clearance control with variable upper limit

### Example of polynomial with dyn. upper limit

For the purpose of clearance control, the upper limit of the output ($AA_OFF, override value in axis V) is varied as a function of the spindle override (analog input 1). The upper limit for polynomial 1 is varied dynamically as a function of analog input 2.

Polynomial 1 is defined directly via system variables:

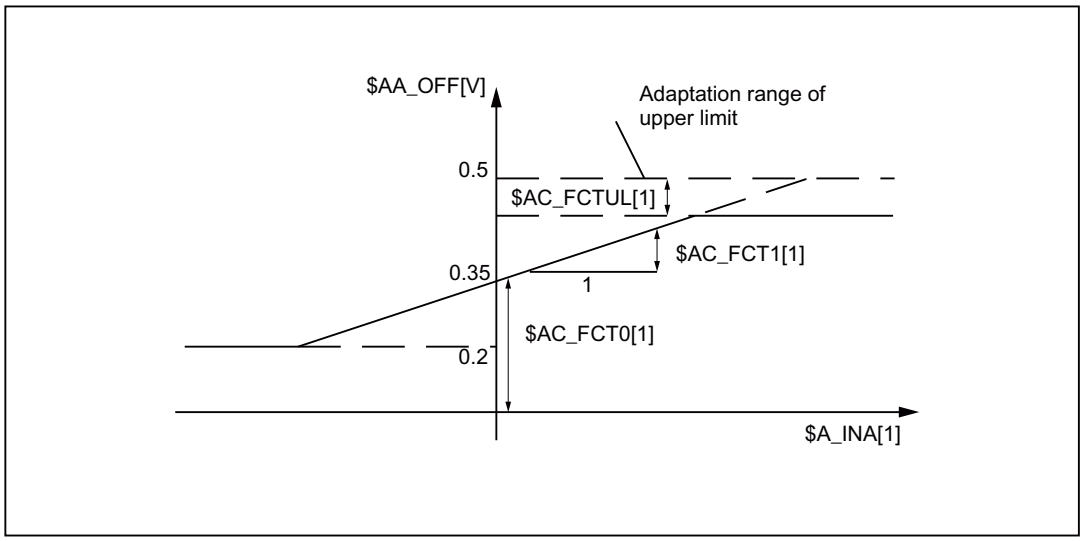



Figure 5-1 Clearance control with variable upper limit

```
$AC_FCTLL[1]=0.2                          ; Lower limit
$AC_FCTUL[1]=0.5                          ; Request Value of upper limit
$AC_FCT0[1]=0.35                          ; Zero passage a0
$AC_FCT1[1]=1.5 EX-5                      ; Pitch a1
STOPRE                                    ; see following note
...
ID=1 DO $AC_FCTUL[1]=$A_INA[2]*0.1+0.35   ; Upper limit
                                          ; adjust dynamically via
                                          ; Analog input 2, no condition
ID=2 DO SYNFCT(1, $AA_OFF[V], $A_INA[1])  ; Clearance control through override
                                          ; no conditions
...
```

**Note**

When system variables are used in the part program, STOPRE must be programmed to ensure block-synchronous writing. The following is an equivalent notation for polynomial definition:

```
FCTDEF(1, 0.2, 0.5, 0.35, 1.5EX-5).
```

## 5.3.2 Feedrate control

### Example of adaptive control with an analog input voltage

A process quantity (measured via $A_INA[1] ) must be regulated to 2 V through an additive control factor implemented by a path (or axial) feedrate override. Feedrate override shall be performed within the range of +100 [mm/min].

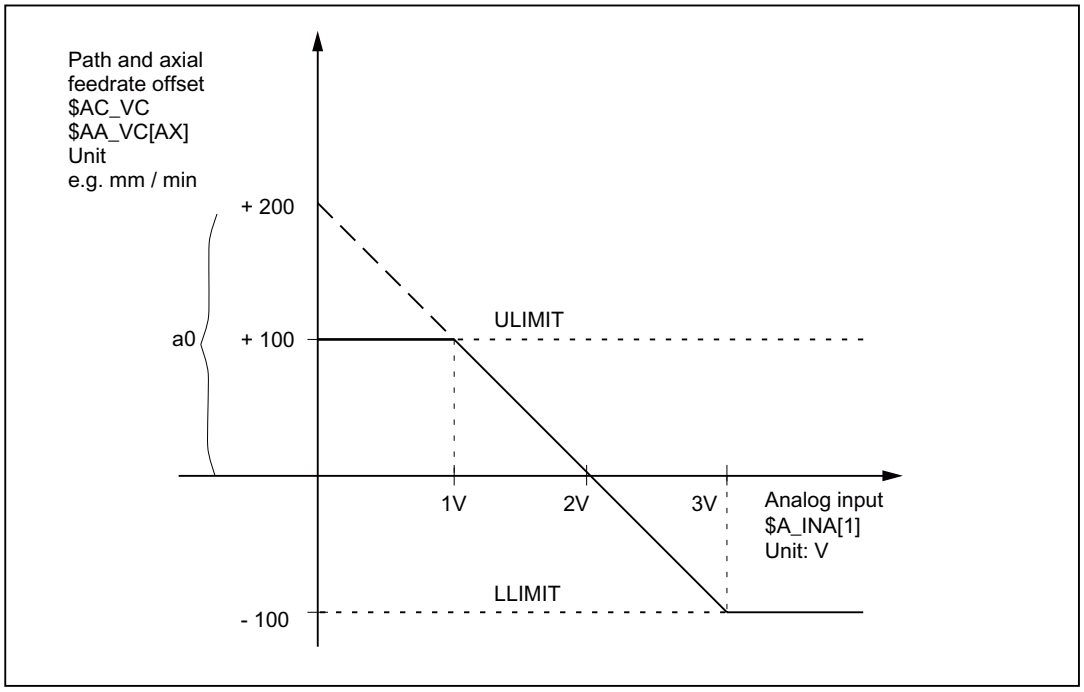



Figure 5-2 Diagram illustrating adaptive control

Determination of coefficients:

$y = f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$

a1 = - 100mm / (1min * 1V)

a1 = - 100% regulation constants, pitch

a0 = - (-100) * 2 = 200

a2 = 0 (not a square component)

a3 = 0 (not a square component)

Upper limit = 100

Lower limit = -100

```
FCTDEF(     Polynomial No.
            LLIMIT
            ULIMIT
            a0              ;   y for x = 0
            a1              ;   Lead
            a2              ;   square component
            a3 )            ;   cubic component
```

With the values determined above, the polynomial is defined as follows:

`FCTDEF(1, -100, -100, 100, 200, 0, 0)`

The following synchronized actions can be used to activate the adaptive control function for the axis feedrate:

```
ID = 1 DO SYNFCT (1, $AA_VC[X], $A_INA[1])
```

or for the path feedrate:

```
ID = 2 DO SYNFCT(1, $AC_VC, $A_INA[1])
```

## 5.3.3 Control velocity as a function of normalized path

### Multiplicative adaptation

The normalized path is applied as an input quantity: $AC_PATHN.

0: At block start

1: at block end

Variation quantity $AC_OVR must be controlled as a function of $AC_PATHN according to a 3rd order polynomial. The override must be reduced from 100 to 1% during the motion.

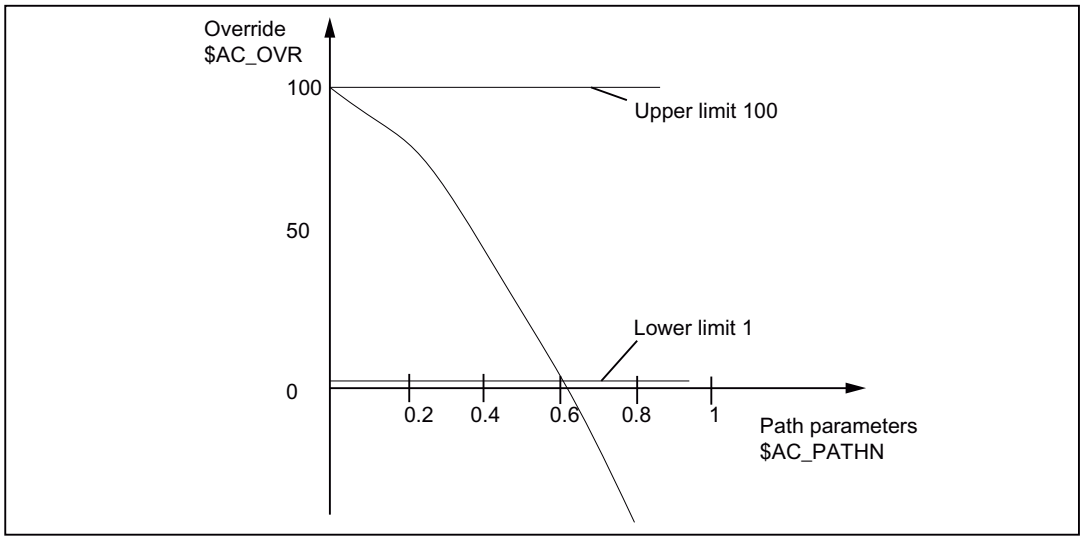



Figure 5-3 Regulate velocity continuously

Polynomial 2:

Lower limit: 1

Hi limit: 100

$a_0$: 100

$a_1$: -100

$a_2$: -100

$a_3$: not used

With these values, the polynomial definition is as follows:

```
FCTDEF(2, 1, 100, 100, -100, -100)
; Activation of the variable override as a function of the path:
ID= 1 DO SYNFCT (2, $AC_OVR, $AC_PATHN)
G01 X100 Y100 F1000
```

## 5.4 Monitoring a safety clearance between two axes

### Task

The axes X1 and X2 operate two independently controlled transport devices used to load and unload workpieces.

To prevent the axes from colliding, a safety clearance must be maintained between them.

If the safety clearance is violated, then axis X2 is decelerated. This interlock is applied until axis X1 leaves the safety clearance area again.

If axis X1 continues to move towards axis X2, thereby crossing a closer safety barrier, then it is traversed into a safe position.

| NC language | Comment |
|---|---|
| ID=1 WHENEVER $AA_IM[X2] - $AA_IM[X1] < 30 DO $AA_OVR[X2]=0 | ; Safety barrier |
| ID=2 EVERY $AA_IM[X2] - $AA_IM[X1] < 15 DO POS[X1]=0 | ; Safe position |

## 5.5 Store execution times in R parameters

### Task

Store the execution time for part program blocks starting at R parameter 10.

| Program | Comment |
|---|---|
| | ; The example is<br>; as follows **without** symbolic programming: |
| IDS=1 EVERY $AC_TIMEC==0 DO $AC_MARKER[0] = $AC_MARKER[0] + 1 | ; Advance R parameter<br>; pointer on block change |
| IDS=2 DO $R[10+$AC_MARKER[0]] = $AC_TIME | ; Write current time<br>; of block start in each case to R parameter |
| | ; The example is<br>; as follows **with** symbolic programming: |
| DEFINE INDEX AS $AC_MARKER[0] | ; Agreements for symbolic<br>; programming |
| IDS=1 EVERY $AC_TIMEC==0 DO INDEX = INDEX + 1 | ; Advance R parameter<br>; pointer on block change |
| IDS=2 DO $R[10+INDEX] = $AC_TIME | ; Write current time<br>; of block start in each case to R parameter |

## 5.6 "Centering" with continuous measurement

### Introduction

The gaps between gear teeth are measured sequentially. The gap dimension is calculated from the sum of all gaps and the number of teeth. The center position sought for continuation of machining is the position of the first measuring point plus 1/2 the average gap size. The speed for measurement is selected in order to enable one measured value to be reliably acquired in each interpolation cycle.

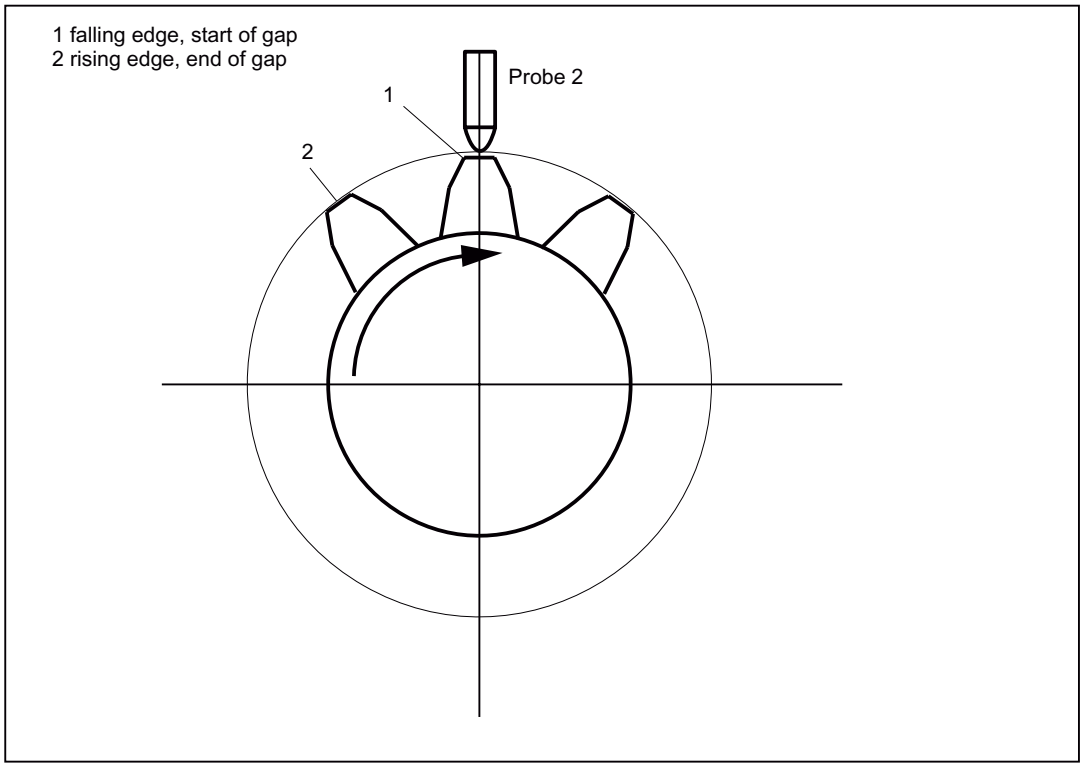



Figure 5-4 Diagrammatic representation of measurement of gaps between gear teeth

%_N_MEAC_MITTEN_MPF

;Measure using rotary axis B (BACH) with display of difference
;between measured values

```
;*** Define local user-defined variables ***
N1 DEF INT ZAEHNEZAHL                ;   Input number of gear teeth
N5 DEF REAL HYS_POS_FLANKE           ;   Hysteresis positive edge probe
N6 DEF REAL HYS_NEG_FLANKE           ;   Hysteresis negative edge probe
;*** Define short names for synchronized action markers ***
define M_ZAEHNE as $AC_MARKER[1]     ;   ID marker for calculation: neg/pos edge per tooth
define Z_MW as $AC_MARKER[2]         ;   Read ID counter MW FIFO
define Z_RW as $AC_MARKER[3]         ;   Calculate ID Counter MW tooth
```

```
*** Input values for ZAHNRADMESSEN ***
N50 ZAEHNEZAHL=26                    ;  Enter number of gear teeth to be measured
N70 HYS_POS_FLANKE = 0.160           ;  Hysteresis positive edge probe
N80 HYS_NEG_FLANKE = 0.140           ;  Hysteresis negative edge probe
```

```
Start:                               ;  *** Assign variables ***
R1=0                                 ;  ID2 calculation result for gap dimension
R2=0                                 ;  ID2 calculation result addition of all gaps
R3=0                                 ;  Contents of the first element read
R4=0                                 ;  R4 corresponds to a tooth distance
R5=0                                 ;  Gap position calculated, final result
R6=1                                 ;  Switch-on ID 3 BACH with MOV
R7=1                                 ;  Switch-on ID 5 MEAC
M_ZAEHNE=ZAEHNEZAHL*2                ;  Calculate ID neg./pos. edge of each teeth
Z_MW=0                               ;  Read ID counter MW FIFO till the number of teeth
Z_RW=2                               ;  Calculate ID counter difference of tooth gap
R13=HYS_POS_FLANKE                   ;  Hysteresis in calculation register
R14=HYS_NEG_FLANKE                   ;  Hysteresis in calculation register
;*** Travel, measure, calculate axis ***
N100 MEAC[BACH]=(0)                  ;  Reset measurement job
;Resetting the FIFO[4] variables and ensuring a defined measurement trace
N105 $AC_FIFO1[4]=0                  ;  Reset FIFO1
STOPRE
;  *** Read FIFO till tooth number reached ***
;  if FIFO1 is not empty and all teeth are still not measured, save measured value from FIFO variable
   in
;  synchronization parameter and increment counter of measured values
```

```
ID=1 WHENEVER ($AC_FIFO1[4]>=1) AND (Z_MW<M_ZAEHNE)
        DO $AC_PARAM[0+Z_MW]=$AC_FIFO1[0] Z_MW=Z_MW+1
;if 2 measured values are present, start calculation, calculate ONLY gap dimension
; and gap sum, increment calculation value counter by 2
ID=2 WHENEVER (Z_MW>=Z_RW) AND (Z_RW<M_ZAEHNE)
        DO $R1=($AC_PARAM[-1+Z_RW]-$R13)-($AC_PARAM[-2+Z_RW]-$R14) Z_RW=Z_RW+2
        $R2=$R2+$R1
;*** Switch-on the axis BACH as endless rotating rotary axis with MOV ***
WAITP(BACH)
ID=3 EVERY $R6==1 DO MOV[BACH]=1     ;  Activate
FA[BACH]=1000
ID=4 EVERY $R6==0 und                ;  Deactivate
($AA_STAT[BACH]==1) DO MOV[BACH]=0
```

```
; Measure sequentially, store in FIFO 1, MT2 neg, MT2 pos edge
;the distance between two teeth is measured
;falling edge-...-rising edge, probe 2
N310 ID=5 WHEN $R7==1 DO MEAC[BACH]=(2, 1, -2, 2)
N320 ID=6 WHEN (Z_MW>=M_ZAEHNE) DO   ;  Cancel measuring job
MEAC[BACH]=(0)
M00
STOPRE
```

```
;*** FIFO Fetch and save values ***
N400 R3=$AC_PARAM[0]                ;  Contents of the first element read
                                    ;  ;Reset the FIFO1[4] variable
                                    ;  ;and ensure a defined measuring trace
                                       ;for the next measurement job
N500 $AC_FIFO1[4]=0
```

```
;*** Calculate difference between the individual teeth ***
N510 R4=R2/(ZAEHNEZAHL)/1000        ;  R4 corresponds to an average
                                    ;  tooth distance
                                    ;  Division "/1000" removed in later SW versions
```

```
;*** Calculate center position ***
N520 R3=R3/1000                     ;  First measurement position converted to degree
N530 R3=R3 MOD 360                  ;  first measurement point modulo
N540 R5=(R3-R14)+(R4/2)             ;  calculate gap position
M00
stopre
R6=0                                ;  Disable axis rotation from BACH
gotob start
M30
```

# 5.7 Axis couplings via synchronized actions

## 5.7.1 Coupling to leading axis

### Task assignment

A cyclic curve table is defined by means of polynomial segments. Controlled by means of arithmetic variables, the movement of the master axis and the coupling process between master and slave (following) axes is activated/deactivated.

%_N_KOP_SINUS_MPF

```
N5 R1=1                                  ;  ID 1, 2 activate/deactivate coupling: LEADON (CACB, BACH)
N6 R2=1                                  ;  ID 3, 4 Move leading axis on/off: MOV BACH
N7 R5=36000                              ;  BACH Feedrate/min
N8 STOPRE
```

```
;*** Define periodic table No. 4 through polynomial segments ***
N10 CTABDEF (YGEO,XGEO,4,1)
N16 G1 F1200 XGEO=0.000 YGEO=0.000       ;  Go to basic position
N17 POLY PO[XGEO]=(79,944.30.420,00.210) PO[YGEO]=(24,634.00.871,-9,670)
N18 PO[XGEO]=(116.059,0.749,-0.656) PO[YGEO]=(22.429,-5.201,0.345)
N19 PO[XGEO]=(243.941,-17.234,11.489) PO[YGEO]=(-22.429,-58.844,39.229)
N20 PO[XGEO]=(280.056,1.220,-0.656) PO[YGEO]=(-24.634,4.165,0.345)
N21 PO[XGEO]=(360.000,-4.050,0.210) PO[YGEO]=(0.000,28.139,-9.670)
N22 CTABEND                              ;  *** End of table definition ***
```

```
; Travel axis leading axis and coupled axis in quick motion in basic position
N80 G0 BACH=0 CACH=0                     ;  Channel axis names
N50 LEADOF(CACH,BACH)                    ;  existing coupling OFF
```

```
N235 ;*** Switch-on the coupling movement for the axis CACH ***
N240 WAITP(CACH)                         ;  Synchronize axis to channel
N245 ID=1 EVERY $R1==1 DO                ;  Coupling via table 4
LEADON(CACH, BACH, 4)
N250 ID=2 EVERY $R1==0 DO                ;  Deactivate coupling
LEADOF(CACH, BACH)
```

```
N265 WAITP(BACH)
N270 ID=3 EVERY $R2==1 DO MOV[BACH]=1 FA[BACH]=R5   ; Rotate leading axis with feedrate endlessly in R5
N275 ID=4 EVERY $R2==0 DO MOV[BACH]=0               ; Stop leading axis
N280 M00
N285 STOPRE
N290 R1=0                                           ; Disable coupling condition
N295 R2=0                                           ; Disable condition for rotating leading axis
N300 R5=180                                         ; New feedrate for BACH
N305 M30
```

## 5.7.2 Non-circular grinding via master value coupling

### Task assignment

A non-circular workpiece that is rotating on axis CACH must be machined by grinding. The distance between the grinding wheel and workpiece is controlled by axis XACH and depends on the angle of rotation of the workpiece. The interrelationship between angles of rotation and assigned movements is defined in curve table 2. The workpiece must move at velocities that are determined by the workpiece contour defined in curve table 1.

### Solution

CACH is designated as the leading axis in a master value coupling. It controls:

- via table 2 the compensatory movement of the axis XACH
- via table 1 the "software axis" CASW.

The axis override of axis CACH is determined by the actual values of axis CASW, thus providing the required contour-dependent velocity of axis CACH.

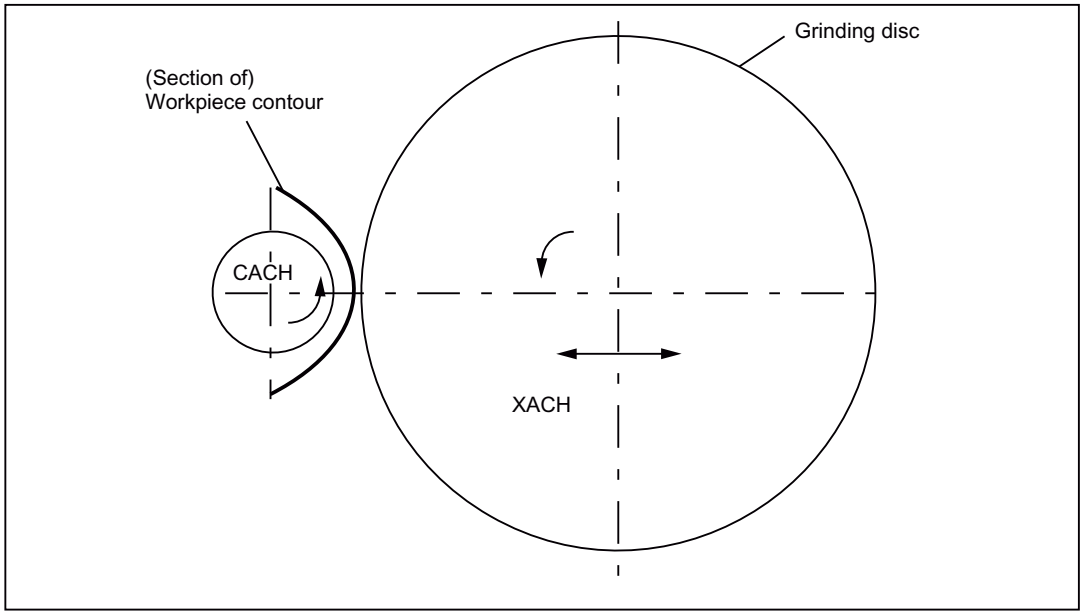



Figure 5-5 Diagrammatic representation of non-circular contour grinding

```
%_N_CURV_TABS_SPF
PROC CURV_TABS
N160 ; *** Define table 1 override ***
N165 CTABDEF(CASW,CACH,1,1)          ;   Table 1 periodic
N170 CACH=0 CASW=10
N175 CACH=90 CASW=10
N180 CACH=180 CASW=100
N185 CACH=350 CASW=10
N190 CACH=359.999 CASW=10
N195 CTABEND
```

```
N160 ; *** Define table 2 linear compensatory movement of XACH ***
CTABDEF(YGEO,XGEO,2,1)               ;   Table 2 periodic
N16 XGEO=0.000 YGEO=0.000
N16 XGEO=0.001 YGEO=0.000
N17 POLY PO[XGEO]=(116.000,0.024,0.012) PO[YGEO]=(4.251,0.067,-0.828)
N18 PO[XGEO]=(244.000,0.072,-0.048) PO[YGEO]=(4.251,-2.937)
N19 PO[XGEO]=(359.999,-0.060,0.012) PO[YGEO]=(0.000,-2.415,0.828)
N16 XGEO=360.000 YGEO=0.000
N20 CTABEND
M17
```

```
%_N_UNRUND_MPF
; Coupling group for a non-circular machining
; XACH is the infeed axis of the grinding disk
; CACH is the workpiece axis as rotary axis and master value axis
; Application: Grind non-circular contours
; Table 1 maps the override for axis CACH as function of the position of CACH
; Overlay of the XGEO axis with handwheel infeed for scratching

N100 DRFOF                              ;  deselect handwheel overlay
N200 MSG(Select "DRF, (Handwheel 1 active) and Select INKREMENT.== Handwheel overlay
AKTIV")
N300 M00
N500 MSG()                              ;  Reset message
N600 R2=1                               ;  LEADON Table 2, Activate with ID=3/4 CACH to
                                           XACH
N700 R3=1                               ;  LEADON Table 1, Activate with ID=5/6 CACH to
                                           CASW, override
N800 R4=1                               ;  Endless rotating axis CACH, start with ID=7/8
N900 R5=36000                           ;  FA[CACH] Endless rotating rotary axis speed


N1100 STOPRE
N1200                                   ;  *** Set axis and leading axis to FA ***
                                        ;  Travel axis leading axis and slave axis
                                        ;   in basic position
N1300 G0 XGEO=0 CASW=10 CACH=0
N1400 LEADOF(XACH,CACH)                 ;  Coupling AUS XACH compensatory movement
N1500 LEADOF(CASW,CACH)                 ;  Coupling AUS CASW override table
N1600 CURV_TABS                         ;  Sub-program with definition of the tables


N1700                                   ;  *** On-off switch of the LEADON compensatory
                                           movement XACH ***
N1800 WAITP(XGEO)                       ;  Synchronize axis to channel
N1900 ID=3 EVERY $R2==1 DO
LEADON(XACH,CACH,2)
N2000 ID=4 EVERY $R2==0 DO
LEADOF(XACH,CACH)
```

```
N2100                                   ; *** On-off switch of the LEADON CASW
                                        ; override table ***
N2200 WAITP(CASW)
N2300 ID=5 EVERY $R3==1 DO              ; CTAB Coupling ON leading axis CACH
LEADON(CASW,CACH,1)
N2400 ID=6 EVERY $R3==0 DO              ; CTAB Coupling OFF leading axis CACH
LEADOF(CASW,CACH)
```

```
N2500                                   ; *** Control override of the CACH from position
                                        ; CASW with ID 10 ***
N2700 ID=11 DO                          ; Assign "axis position" CASW to OVR CACH
$$AA_OVR[CACH]=$AA_IM[CASW]
```

```
N2900 WAITP(CACH)
N3000 ID=7 EVERY $R4==1 DO              ; Start as endless rotating rotary axis
MOV[CACH]=1 FA[CACH]=R5
N3100 ID=8 EVERY $R4==0 DO              ; Stop as endless rotating rotary axis
MOV[CACH]=0
```

```
N3200 STOPRE
N3300 R90=$AA_COUP_ACT[CASW]            ; State of the coupling for CASW for checking
N3400 MSG("Override table CASW activated with LEADON "<<R90<<", further ENDE with
NC-START")
```

```
N3500 M00                               ; *** NC HALT ***
N3600 MSG()
N3700 STOPRE                            ; Preprocessing stop
N3800 R1=0                              ; Stop with ID=2 CASW axis as
                                        ; endless rotating rotary axis
N3900 R2=0                              ; LEADOF with ID=6 FA XACH
                                        ; and leading axis CACH
N4000 R3=0                              ; LEADOF TAB1 CASW with ID=7/8 CACH
                                        ; to CASW override table
N4100 R4=0                              ; Stop axis as endless rotating rotary axis
                                        ; , ID=4 CACH
N4200 M30
```

### Expansion options

The example above can be expanded by the following components:

- Introduction of a Z axis to move the grinding wheel or workpiece from one non-circular operation to the next on the same shaft (cam shaft).
- Table switchovers, if the cams for inlet and outlet have diferent contours.

  ```
  ID = ... <Condition> DO LEADOF(XACH, CACH) LEADON(XACH, CACH, <new table number>)
  ```
- Dressing of grinding wheel by means of online tool offset acc. to Subsection "Online tool offset FTOC".

## 5.7.3 On-the-fly parting

### Task assignment

An extruded material which passes continuously through the operating area of a cutting tool must be cut into parts of equal length.

X axis: Axis in which the extruded material moves, WKS

X1 axis: Machine axis of the extruded material, MKS

Y axis: Axis in which cutting tool "tracks" the extruded material

It is assumed that the infeed and control of the cutting tool are controlled via the PLC. The signals at the PLC interface can be evaluated to determine whether the extruded material and cutting tool are synchronized.

### Actions

Activate coupling, LEADON

Deactivate coupling, LEADOF

Set actual values, PRESETON

```
NC program                                  Comment
%_N_SCHERE1_MPF
;$PATH=/_N_WKS_DIR/_N_DEMOFBE_WPD
N100 R3=1500                          ;  Length of a part to be cut off
N200 R2=100000      R13=R2/300
N300 R4=100000
N400 R6=30                            ;  Start position Y axis
N500 R1=1                             ;  Start condition for conveyor axis
N600 LEADOF(Y,X)                      ;  Delete any existing coupling
N700 CTABDEF(Y,X,1,0)                 ;  Table definition
N800 X=30 Y=30                        ;  Value pairs
N900 X=R13 Y=R13
N1000 X=2*R13 Y=30
N1100 CTABEND                         ;  End of table definition
```

| **NC program** | | Comment |
|---|---|---|
| N1200 PRESETON(X1,0) | ; | PRESET at beginning |
| N1300 Y=R6 G0 | ; | Start position Y axis |
| | ; | Axis is linear |
| N1400 ID=1 EVERY $AA_IW[X]>$R3 DO PRESETON(X1,0) | ; | PRESET according to length R3, PRESTON may |
| | ; | be done only with WHEN and EVERY |
| | ; | new start after parting |
| N1500 WAITP(Y) | | |
| N1800 ID=6 EVERY $AA_IM[X]<10 DO LEADON(Y,X,1) | | ; Couple Y to X via table 1, for X < 10 |
| N1900 ID=10 EVERY $AA_IM[X]>$R3-30 DO LEADOF(Y,X) | | ; > 30 before traversed parting distance, deactivate coupling |
| N2000 WAITP(X) | | |
| N2100 ID=7 WHEN $R1==1 DO MOV[X]=1 FA[X]=$R4 | | ; Set extruded material axis continuously in motion |
| N2200 M30 | | |

# 5.8 Technology cycles position spindle

## Application

Interacting with the PLC program, the spindle which initiates a tool change should be:

- Traversed to an initial position,
- Positioned at a specific point at which the tool to be inserted is also located.

See chapter "Starting of command axes" and chapter "Control via PLC".

## Coordination

The PLC and NCK are coordinated by means of the common data that are provided in SW version 4 and later (see chapter "List of the system variables relevant for synchronized actions")

- $A_DBB[0]: Take up basic position 1,
- $A_DBB[1]: Take up target position 1,
- $A_DBW[1]: value to be positioned +/- , PLC calculates the shortest route.

## Synchronized actions

%_N_MAIN_MPF

```
...
IDS=1 EVERY $A_DBB[0]==1 DO          ;  ; when $A_DBB[0] set by PLC,
NULL_POS                             ;  take up basic position
IDS=2 EVERY $A_DBB[1]==1 DO          ;  when $A_DBB[1] set by PLC,
ZIEL_POS                             ;  position spindle to the value stored in
                                     ;  $A_DBW[1]
...
```

## Technology cycle NULL_POS

%_N_NULL_POS_SPF

```
PROC NULL_POS
SPOS=0                               ;  Bring drive for the tool change
                                     ;  in basic position
$A_DLB[0]=0                          ;  Basic position executed in NCK
```

## Technology cycle ZIEL_POS

%_N_ZIEL_POS_SPF

```
PROC TARGET_POS
SPOS=IC($A_DBW[1])                   ;  Position spindle to the value,
                                     ;  stored in $A_DBW[1]
                                     ;  stored by PLC, incremental dimension
$A_DBB[1]=0                          ;  Target position executed in NCK
```

# 5.9 Synchronized actions in the TC/MC area

## Introduction

The following figure shows the schematic structure of a tool-changing cycle.

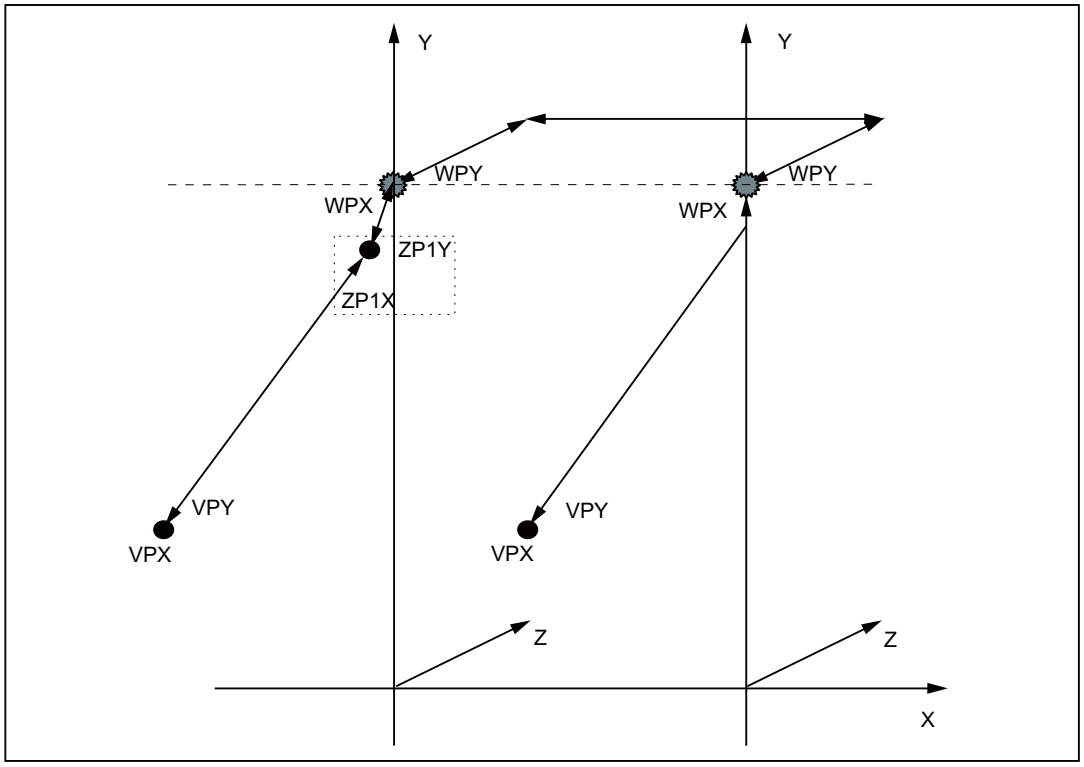



Figure 5-6 Schematic sequence for tool-changing cycle

## Flowchart

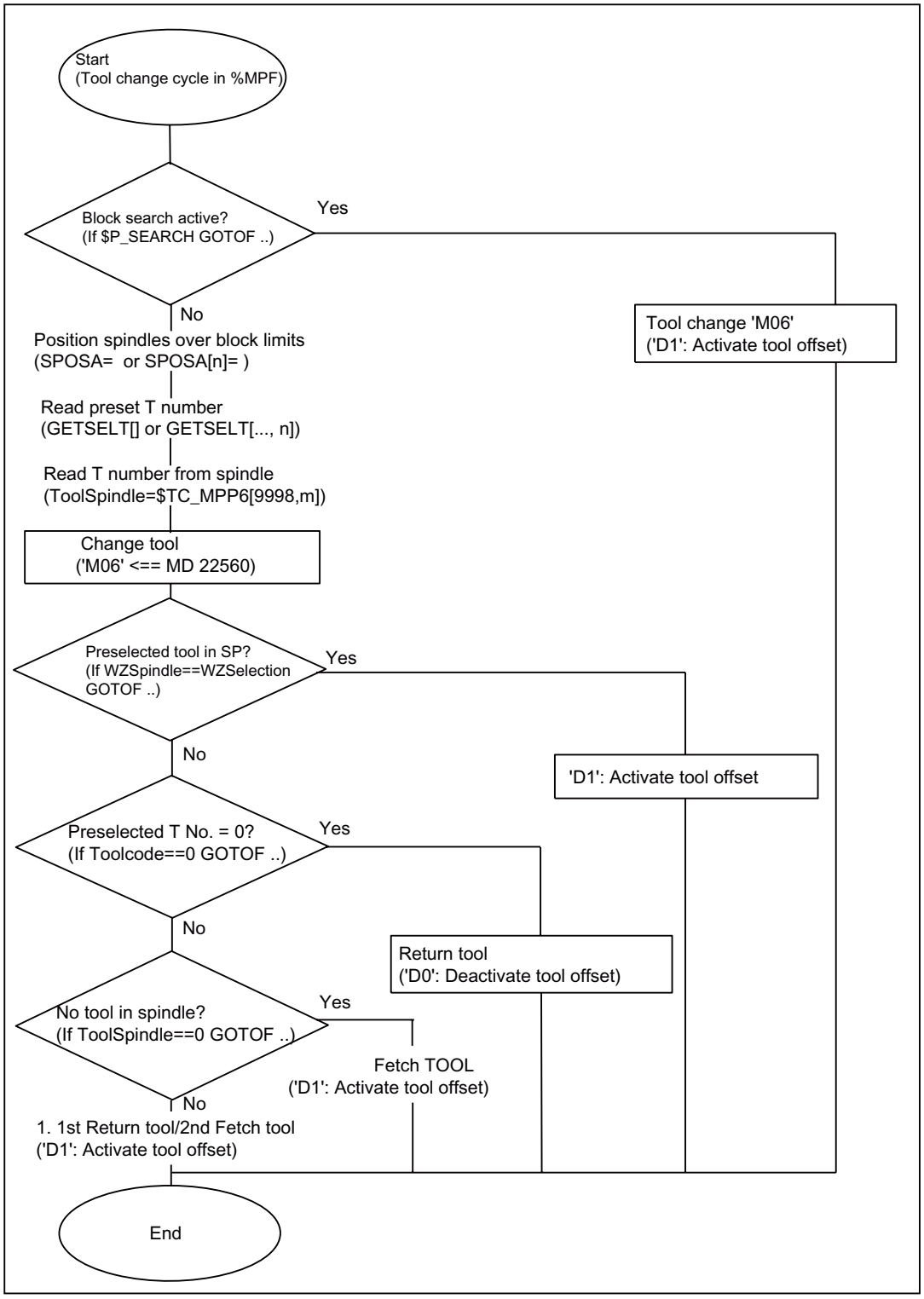



Figure 5-7 Flowchart for tool-changing cycle

| NC program | | Comment |
|---|---|---|
| %_N_WZW_SPF | | |
| ;$PATH=/_N_SPF_DIR | | |
| N10 DEF INT WZPreselection,WZSpindle | ; | Marker on = 1 when MagAxis traversed |
| N15 WHEN $AC_PATHN<10 DO $AC_MARKER[0]=0 $AC_MARKER[1]=0 $AC_MARKER[2]=0 | | |
| N20 ID=3 WHENEVER $A_IN[9]==TRUE DO $AC_MARKER[1]=1 | | |
| N25 ID=4 WHENEVER $A_IN[10]==TRUE DO $AC_MARKER[2]=1 | ; | Marker on = 1 when MagAxis traversed |
| N30 IF $P_SEARCH GOTOF wzw_vorlauf | ; | Block search active ? -> |
| N35 SPOSA=0 D0 | | |
| N40 GETSELT(WZPreselection) | ; | Read preselected T-No. |
| N45 WZSpindle=$TC_MPP6[9998,1] | ; | Read WZ in spindle |
| N50 M06 | | |
| N55 IF WZSpindle==WZPreselection GOTOF wz_in_spindle IF WZPreselection==0 GOTOF store1 IF WZSpindle==0 GOTOF fetch1 | | |
| **;*** Fetch and store tool***** | | |
| store1fetch1: | | |
| N65 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[1]=1 | ; | when MagAxis travels Marker = 1 |
| N70 G01 G40 G53 G64 G90 X=Magazin1VPX Y=Magazin1VPY Z=Magazin1ZGespannt F70000 M=QU(120) M=QU(123) M=QU(9) | | |
| N75 WHENEVER $AA_STAT[S1]<>4 DO $AC_OVR=0 | ; | Spindle in position |
| N80 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[1]=1 | ; | Query MagAxis travel |
| N85 WHENEVER $AC_MARKER[1]==0 DO $AC_OVR=0 | ; | Override=0 when axis not traversed |
| N90 WHENEVER $AA_STAT[C2]<>4 DO $AC_OVR=0 | ;<br>; | Override=0 when MagAxis<br>not in position fine |
| N95 WHENEVER $AA_DTEB[C2]>0 DO $AC_OVR=0 | ; | Override=0 when distance-to-go MagAxis > 0 |
| N100 G53 G64 X=Magazin1ZP1X Y=Magazin1ZP1Y F60000 | | |
| N105 G53 G64 X=Magazin1WPX Y=Magazin1WPY F60000 | | |
| N110 M20 | ; | Release WZ |
| N115 G53 G64 Z=MR_Magazin1ZGeloest F40000 | | |
| N120 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[2]=1; | | |
| N125 WHENEVER $AC_MARKER[2]==0 DO $AC_OVR=0 | | |
| N130 WHENEVER $AA_STAT[C2]<>4 DO $AC_OVR=0 | | |
| N135 WHENEVER $AA_DTEB[C2]>0 DO $AC_OVR=0 | | |
| N140 G53 G64 Z=Magazin1ZGespannt F40000 | | |
| N145 M18 | ; | Clamp tool |
| N150 WHEN $AC_PATHN<10 DO M=QU(150) M=QU(121) | ; | Condition always fulfilled |
| N155 G53 G64 X=Magazin1VPX Y=Magazin1VPY F60000 D1 M17 | | |
| **;*** Store tool***** | | |
| store1: | | |
| N160 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[1]=1 | | |
| N165 G01 G40 G53 G64 G90 X=Magazin1VPX Y=Magazin1VPY Z=Magazin1ZGespannt F70000 M=QU(120) M=QU(123) M=QU(9) | | |
| N170 WHENEVER $AA_STAT[S1]<>4 DO $AC_OVR=0 | | |
| N175 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[1]=1 | | |

| NC program | Comment |
|---|---|
| N180 WHENEVER $AC_MARKER[1]==0 DO $AC_OVR=0 | |
| N185 WHENEVER $AA_STAT[C2]<>4 DO $AC_OVR=0 | |
| N190 WHENEVER $AA_DTEB[C2]>0 DO $AC_OVR=0 | |
| N195 G53 G64 X=Magazin1ZP1X Y=Magazin1ZP1Y F60000 | |
| N200 G53 G64 X=Magazin1WPX Y=Magazin1WPY F60000 | |
| N205 M20 | ; Release tool |
| N210 G53 G64 Z=Magazin1ZGeloest F40000 | |
| N215 G53 G64 X=Magazin1VPX Y=Magazin1VPY F60000 M=QU(150) M=QU(121) D0 M17 | |
| **;*** Fetch tool*** ** | |
| fetch1: | |
| N220 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[2]=1 | |
| N225 G01 G40 G53 G64 G90 X=Magazin1VPX Y=Magazin1VPY Z=Magazin1ZGeloest F70000 M=QU(120) M=QU(123) M=QU(9) | |
| N230 G53 G64 X=Magazin1WPX Y=Magazin1WPY F60000 | |
| N235 WHENEVER $AA_STAT[S1]<>4 DO $AC_OVR=0 | |
| N240 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[2]=1 | |
| N245 WHENEVER $AC_MARKER[2]==0 DO $AC_OVR=0 | |
| N250 WHENEVER $AA_STAT[C2]<>4 DO $AC_OVR=0 | |
| N255 WHENEVER $AA_DTEB[C2]>0 DO $AC_OVR=0 | |
| N260 G53 G64 Z=Magazin1ZGespannt F40000 | |
| N265 M18 | ; Clamp tool |
| N270 G53 G64 X=Magazin1VPX Y=Magazin1VPY F60000 M=QU(150) M=QU(121) D1 M17 | |
| **;***Tool in spindle*** ** | |
| wz_in_spindle: | |
| N275 M=QU(121) D1 M17 | |
| **;***Block search*** ** | |
| wzw_feed: | |
| N280 STOPRE | |
| N285 D0 | |
| N290 M06 | |
| N295 D1 M17 | |

# Data lists

# 6

## 6.1 Machine data

### 6.1.1 General machine data

| Number | Identifier: $MN_ | Description |
|---|---|---|
| 11110 | AUXFU_GROUP_SPEC | Auxiliary function group specification |
| 11500 | PREVENT_SYNACT_LOCK | Protected synchronized actions |
| 18860 | MM_MAINTENANCE_MON | Activate recording of maintenance data |

### 6.1.2 Channelspecific machine data

| Number | Identifier: $MC_ | Description |
|---|---|---|
| 21240 | PREVENT_SYNACT_LOCK_CHAN | Protected synchronized actions for channel |
| 28250 | MM_NUM_SYNC_ELEMENTS | Number of elements for expressions in synchronized actions |
| 28252 | MM_NUM_FCTDEF_ELEMENTS | Number of FCTDEF elements |
| 28254 | MM_NUM_AC_PARAM | Number of $AC_PARAM parameters |
| 28255 | MM_BUFFERED_AC_PARAM | Storage location for $AC_PARAM (SW 6.3 and later) |
| 28256 | MM_NUM_AC_MARKER | Number of $AC_MARKER markers |
| 28257 | MM_BUFFERED_AC_MARKER | Storage location for $AC_MARKER (SW 6.3 and later) |
| 28258 | MM_NUM_AC_TIMER | Number of $AC_TIMER time variables |
| 28260 | NUM_AC_FIFO | Number of $AC_FIFO1, $AC_FIFO2, ... variables |
| 28262 | START_AC_FIFO | Store FIFO variables from R parameter |
| 28264 | LEN_AC_FIFO | Length of $AC_FIFO ... FIFO variables |
| 28266 | MODE_AC_FIFO | FIFO processing mode |

### 6.1.3 Axis-specific machine data

| Number | Identifier: $MA_ | Description |
|---|---|---|
| 30450 | IS_CONCURRENT_POS_AX | Concurrent positioning axis |
| 32060 | POS_AX_VELO | Initial setting for positioning axis velocity |
| 32070 | CORR_VELO | Axial velocity for handwheel, ext. ZO, cont. dressing, clearance control (from SW3) |
| 32074 | FRAME_OR_CORRPOS_NOTALLOWED | Effectiveness of frames and tool length offset |
| 32920 | AC_FILTER_TIME | Filter smoothing time constant for Adaptive Control (SW2 and later) |
| 33060 | MAINTENANCE_DATA | Configuration, recording maintenance data |
| 36750 | AA_OFF_MODE | Effect of value assignment for axial override for synchronized actions (SW3 and later) |
| 37200 | COUPLE_POS_TOL_COARSE | Threshold value for "Coarse synchronism" |
| 37210 | COUPLE_POS_TOL_FINE | Threshold value for "Fine synchronism" |

## 6.2 Setting data

### 6.2.1 Axis/spindle-specific setting data

| Number | Identifier: $SA_ | Description |
|---|---|---|
| 43300 | ASSIGN_FEED_PER_REV_SOURCE | Rotational feedrate for positioning axes/spindles |
| 43350 | AA_OFF_LIMIT | Upper limit of offset value for $AA_OFF clearance control |
| 43400 | WORKAREA_PLUS_ENABLE | Working area limitation in pos. direction |

# 6.3 Signals

## 6.3.1 Signals from channel

| DB number | Byte.Bit | Description |
|---|---|---|
| 21, … | 280.1 | Disable modal synchronized actions acc. to DBX300.0-307.7 |
| 21, … | 300.0 - | Modal synchronized actions disabled acc. to DBX300.0-307.7, acknowledgment from NCK |
| 21, … | 300.0 - | Modal synchronized actions ID or IDS 1 - |
| 21, … | 307.7 | Disable 64. Request to NCK channel |
| 21, … | 308.0 - | Modal synchronized actions ID or IDS 1 - |
| 21, … | 315.7 | 64 can be disabled. Message from NCK. |

# Appendix

# A

## A.1 Publication-specific information

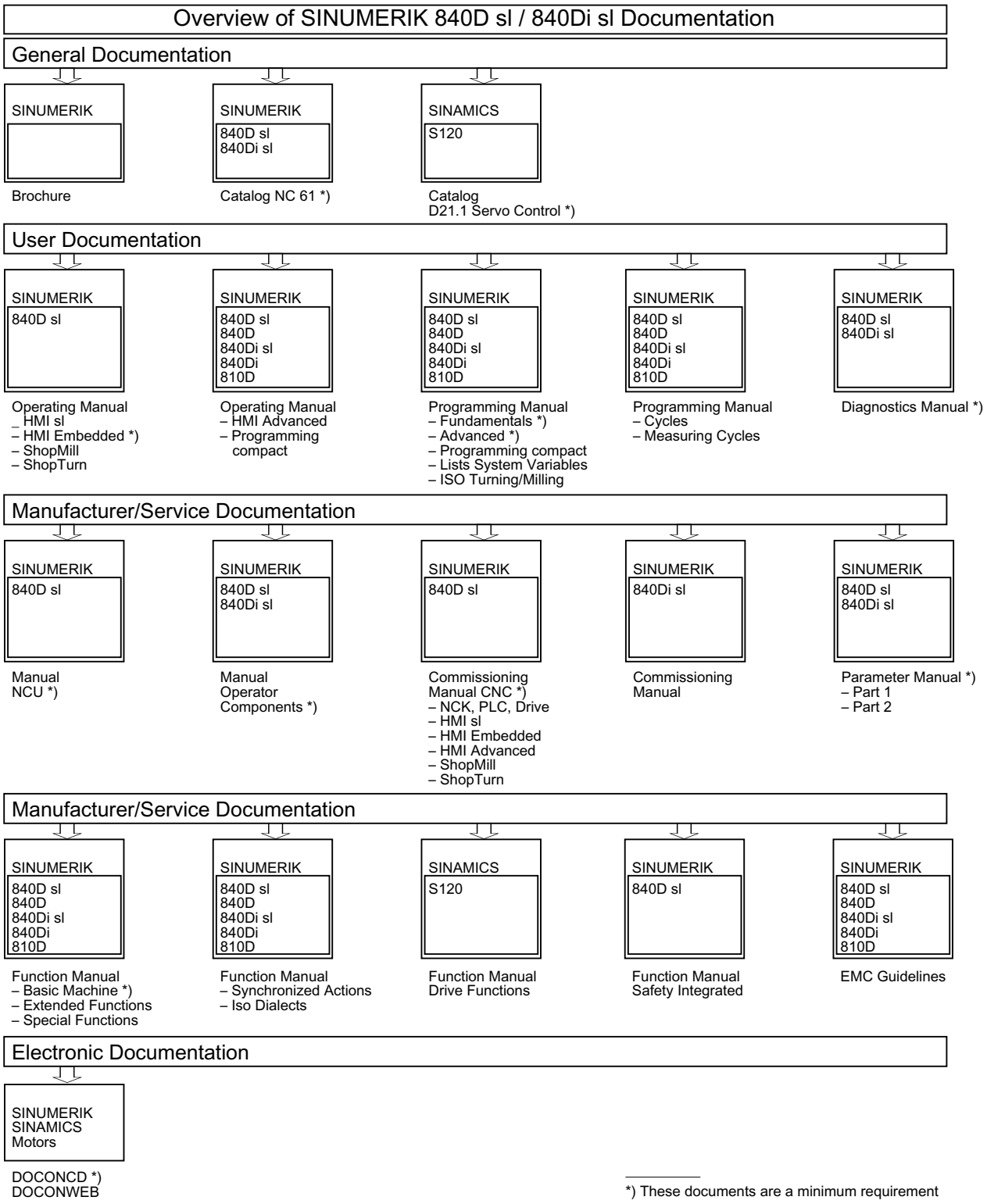
### A.1.1 Correction sheet - fax template

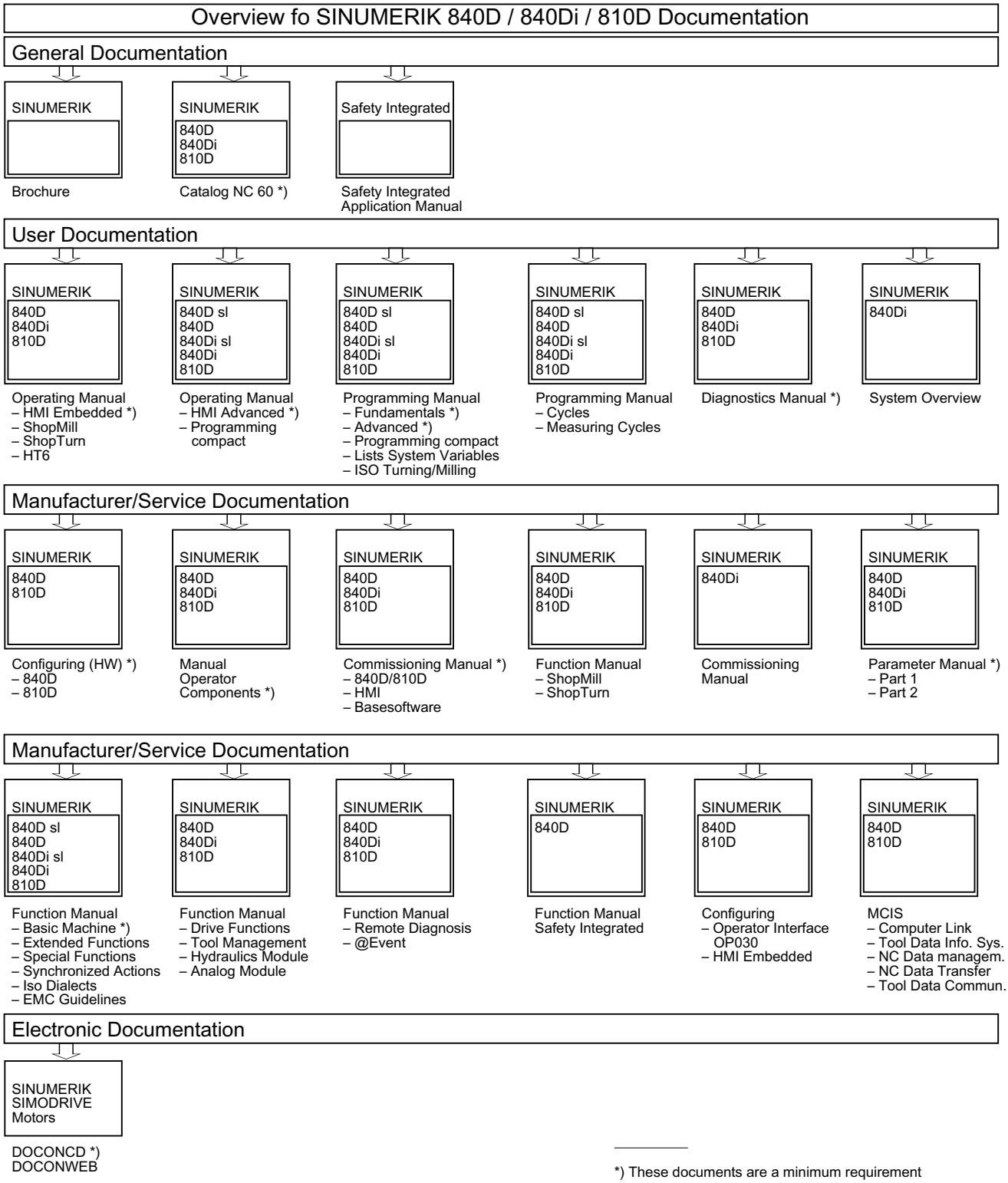
Should you come across any printing errors when reading this publication, please notify us on this sheet. Suggestions for improvement are also welcome.

| To<br>SIEMENS AG<br>A&D MC MS1<br>Postfach 3180<br><br>D-91050 Erlangen<br><br>Fax: +49 (0) 9131 / 98 - 63315 (Documentation)<br>mailto: docu.motioncontrol@siemens.com<br>www.siemens.com/automation/service&support | From<br>Name:<br>Company/Dept.<br>Street:<br>Zip code: Town:<br>Phone: /<br>Fax: / |
|---|---|

Suggestions and / or corrections

## A.1.2 Overview




Overview fo SINUMERIK 840D / 840Di / 810D Documentation
General Documentation
SINUMERIK
Brochure
SINUMERIK
840D
840Di
810D
Catalog NC 60 *)
Safety Integrated
Safety Integrated
Application Manual
User Documentation
SINUMERIK
840D
840Di
810D
Operating Manual
– HMI Embedded *)
– ShopMill
– ShopTurn
– HT6
SINUMERIK
840D sl
840D
840Di sl
840Di
810D
Operating Manual
– HMI Advanced *)
– Programming compact
SINUMERIK
840D sl
840D
840Di sl
840Di
810D
Programming Manual
– Fundamentals *)
– Advanced *)
– Programming compact
– Lists System Variables
– ISO Turning/Milling
SINUMERIK
840D sl
840D
840Di sl
840Di
810D
Programming Manual
– Cycles
– Measuring Cycles
SINUMERIK
840D
840Di
810D
Diagnostics Manual *)
SINUMERIK
840Di
System Overview
Manufacturer/Service Documentation
SINUMERIK
840D
810D
Configuring (HW) *)
– 840D
– 810D
SINUMERIK
840D
840Di
810D
Manual
Operator
Components *)
SINUMERIK
840D
840Di
810D
Commissioning Manual *)
– 840D/810D
– HMI
– Basesoftware
SINUMERIK
840D
840Di
810D
Function Manual
– ShopMill
– ShopTurn
SINUMERIK
840Di
Commissioning
Manual
SINUMERIK
840D
840Di
810D
Parameter Manual *)
– Part 1
– Part 2
Manufacturer/Service Documentation
SINUMERIK
840D sl
840D
840Di sl
840Di
810D
Function Manual
– Basic Machine *)
– Extended Functions
– Special Functions
– Synchronized Actions
– Iso Dialects
– EMC Guidelines
SINUMERIK
840D
840Di
810D
Function Manual
– Drive Functions
– Tool Management
– Hydraulics Module
– Analog Module
SINUMERIK
840D
840Di
810D
Function Manual
– Remote Diagnosis
– @Event
SINUMERIK
840D
Function Manual
Safety Integrated
SINUMERIK
840D
810D
Configuring
– Operator Interface OP030
– HMI Embedded
SINUMERIK
840D
810D
MCIS
– Computer Link
– Tool Data Info. Sys.
– NC Data managem.
– NC Data Transfer
– Tool Data Commun.
Electronic Documentation
SINUMERIK
SIMODRIVE
Motors
DOCONCD *)
DOCONWEB
*) These documents are a minimum requirement

# Index


## $

$AA_OFF, 56

## A

Adaptive control, 134
  Additive control, 52
  Example, 136
  Multiplicative control, 53
Axial feed, 73
Axis replacement from synchronized actions, 75
  AXTOCHAN (axis, channel number)[axis, channel number], 80
  GET[axis], 75
  RELEASE[axis], 75

## B

Block search, 114
Boundary conditions, 125

## C

Calculate slave value, 88
Calculating master value, 89
Command axes, 69
Configurability, 116
Configuration, 116
Control system response, 111
Conversion routines, 23
Coordination, 104
CORROF, 58
Coupled motion, 86
Couplings, 86

## D

DB 21, ...
  DBX300.0, 130
DB 31, ...
  DBX28.7, 69, 112
DB21, ...
  DBX318.2, 64
  DBX318.3, 64
DB21, …
  DBB308-315, 106
  DBX.281.1, 107
  DBX1.2, 106, 130
  DBX280.1, 107, 130
  DBX281.1, 130
  DBX300.0, 106, 107, 129
  DBX307.7, 106, 107, 129, 130
  DBX308.0, 130
  DBX315.7, 130
Detection of synchronism, 90
Diagnostics data, 119

## E

End of program, 113
Extensions in SW 5, 127

## F

FCTDEF, 49
FIFO variable, 34
FTOC
  Online tool offset, 59

## I

Identification number, 14
ITOR, 23

## J

Jerk, 100

## M

Machine maintenance, 99
Main run variables, 22
  Log, 121
MD 37200, 90

MD 37210, 90
MD10722, 77
MD11110, 45
MD11500, 108, 109, 110
MD18860, 99, 100
MD20110, 64, 112, 114
MD21190, 62, 63, 64
MD21194, 61, 63
MD21196, 61, 63
MD21240, 109, 110
MD22200, 45
MD22210, 45
MD22230, 45
MD28050, 95
MD28250, 116
MD28252, 50, 117
MD28255, 31
MD28256, 29
MD28257, 29
MD28258, 95
MD28260, 95
MD28262, 95
MD28264, 95
MD28266, 95
MD30552, 80
MD32060, 73, 75
MD32070, 57
MD32074, 72, 73
MD32300, 82
MD33060, 99, 100
MD35040, 112, 114
MD36750, 54, 55, 56, 57, 134
Measurements from synchronized actions, 93
Mode change, 113

## N

NC STOP, 112

## O

Online tool offset, 59, 61
Output of M, S and H auxiliary functions, 44
Overlaid movements, 56
Overlaid movements up to SW 5.3, 56

## P

Polynomial, 49
Polynomial evaluation, 51
Power On, 111
Preset actual-value memory, 85
Program interruption by ASUB, 115
Protected synchronized actions, 108

## R

Real-time variables
  Advertisements, 121
  Read, 47
  Write, 47
REPOS, 115
RESET, 111
Response to alarms, 115
RTOI, 24

## S

SD43300, 74
SD43350, 54, 134
SD43400, 82
Set alarm, 98
Special real-time variables, 28
Spindle motions, 81
Starting/Stopping axes from synchronized actions, 74
Status of synchronized actions, 120
Synchronized action
  Delete, 15
Synchronized actions
  Actions, 18, 42
  Additive adjustment via SYNFCT, 52
  Alter setting data, 48
  Area of Application, 13
  Availability, 125
  Channel control, 106
  Components, 13
  Conditions, 16
  Control via PLC, 106
  Definition, 20
  Disable axis, 68
  Example Adaptive control, 134
  Example Conditions, 131
  Example Control via dyn. override, 137
  Example Path feedrate control, 136
  Example Presses, coupled axes, 143
  Examples: SD/MD, 132
  Execution of synchronized actions, 20
  Extensions in SW 4, 125
  FIFO variable, 34
  Introduction, 11
  Multiplicative control via SYNFCT, 53
  Order of execution, 19

Real-time calculations, 22
Scanning frequency, 15
Scope of performance, 125
Synchronized actions (FBSY)|Data fields, lists, 157
Synchronized actions (FBSY)|Example, 131
Synchronized actions (FBSY)|Interface signals, 157
Synchronized actions (FBSY)|Supplementary conditions, 125
Synchronous procedure
DELDTG, 67
RDISABLE, 66
STOPREOF, 66
SYNFCT
Examples, 134
Polynomial evaluation, 51

## T

Technology cycle, 101
Technology cycles, 101
Call, 101
TOFFON
Online tool length offset, 61
Total travel count, 100
Total travel time, 100
At high speed, 100
Total traverse path, 100
At high speed, 100

## W

Wait markers
Delete, 97
Setting, 97