

# SIEMENS

## SINUMERIK

### SINUMERIK 840D sl 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

Software  
NCU system software for 840D sl/840DE sl




Version  
2.6

03/2009  
6FC5397-5BP10-4BA0

## Legal information

### Warning notice system

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.

 <b>DANGER</b>
indicates that death or severe personal injury <b>will</b> result if proper precautions are not taken.
 <b>WARNING</b>
indicates that death or severe personal injury <b>may</b> result if proper precautions are not taken.
 <b>CAUTION</b>
with a safety alert symbol, indicates that minor personal injury can result if proper precautions are not taken.
<b>CAUTION</b>
without a safety alert symbol, indicates that property damage can result if proper precautions are not taken.
<b>NOTICE</b>
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.

### Proper use of Siemens products

Note the following:

 <b>WARNING</b>
Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be adhered to. The information in the relevant documentation must be observed.

### 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.

# Foreword

## SINUMERIK® Documentation

The SINUMERIK documentation is organized in three parts:

- General documentation
- User documentation
- Manufacturer/service documentation

Information on the following topics is available at <http://www.siemens.com/motioncontrol/docu>:

- Ordering documentation  
Here you can find an up-to-date overview of publications
- Downloading documentation  
Links to more information for downloading files from Service & Support.
- Researching documentation online  
Information on DOConCD and direct access to the publications in DOConWeb.
- Compiling individual documentation on the basis of Siemens contents with the My Documentation Manager (MDM), refer to <http://www.siemens.com/mdm>.

My Documentation Manager provides you with a range of features for generating your own machine documentation.

- Training and FAQs  
Information on our range of training courses and FAQs (frequently asked questions) are available via the page navigation.

## 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

In case of questions, please contact us through the following hotline:

	<b>Europe / Africa</b>
Phone	+49 180 5050 - 222
Fax	+49 180 5050 - 223
€0.14/min. from German landlines, mobile phone prices may differ	
Internet	<a href="http://www.siemens.de/automation/support-request">http://www.siemens.de/automation/support-request</a>

	<b>America</b>
Phone	+1 423 262 2522
Fax	+1 423 262 2200
E-mail	<a href="mailto:techsupport.sea@siemens.com">mailto:techsupport.sea@siemens.com</a>

	<b>Asia/Pacific</b>
Phone	+86 1064 757575
Fax	+86 1064 747474
E-mail	<a href="mailto:support.asia.automation@siemens.com">mailto:support.asia.automation@siemens.com</a>

---

#### Note

You will find telephone numbers for other countries for technical support on the Internet:  
<http://www.automation.siemens.com/partner>

---

### 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 9131- 98 2176

Email: <mailto:docu.motioncontrol@siemens.com>

A fax form is available in the appendix of this document.

### SINUMERIK Internet address

<http://www.siemens.com/sinumerik>



# Contents

	<b>Foreword .....</b>	<b>3</b>
<b>1</b>	<b>Brief description .....</b>	<b>11</b>
<b>2</b>	<b>Detailed description .....</b>	<b>13</b>
2.1	Definition of synchronized actions .....	13
2.2	Components of synchronized actions .....	13
2.2.1	Validity, identification number .....	14
2.2.2	Frequency .....	15
2.2.3	G code for condition and action .....	16
2.2.4	Condition .....	16
2.2.5	G code for action .....	18
2.2.6	Action or Technology cycle .....	18
2.3	List of possible actions .....	22
2.4	Real-time evaluations and calculations .....	23
2.5	Special main run variables for synchronized actions .....	28
2.5.1	Marker/counter variables .....	29
2.5.2	Timers .....	30
2.5.3	Synchronized action parameters .....	31
2.5.4	R parameters .....	32
2.5.5	Machine and setting data .....	32
2.5.6	FIFO variables (circulating memory) .....	34
2.5.7	SRAM stored system variables .....	36
2.5.8	Determining the path tangent in synchronized actions .....	37
2.5.9	Determining the current override .....	37
2.5.10	Capacity evaluation using time requirement for synchronized actions .....	38
2.5.11	List of system variables relevant to synchronized actions .....	41
2.6	Actions in synchronized actions .....	41
2.6.1	Output of M, S and H auxiliary functions to the PLC .....	43
2.6.2	Setting (writing) and reading of main run variables .....	46
2.6.3	Changing of SW cam positions and times (setting data) .....	47
2.6.4	FCTDEF .....	48
2.6.5	Polynomial evaluation SYNFACT .....	50
2.6.6	Overlaid movements \$AA_OFF settable (SW 6 and later) .....	56
2.6.7	Online tool offset FTOC .....	58
2.6.8	Online tool length offset \$AA_TOFF[Index] .....	60
2.6.9	RDISABLE .....	64
2.6.10	STOPREOF .....	65
2.6.11	DELDTG .....	65
2.6.12	Disabling a programmed axis motion .....	67
2.6.13	Traversing command axes .....	67
2.6.14	Axial feedrate from synchronized actions .....	72
2.6.15	Starting/Stopping axes from synchronized actions .....	73
2.6.16	Axis replacement from synchronized actions .....	74
2.6.17	Spindle motions from synchronized actions .....	79
2.6.18	Setting actual values from synchronized actions .....	83
2.6.19	Activating/deactivating coupled motions and couplings .....	84

2.6.20	Measurements from synchronized actions .....	90
2.6.21	Setting and deleting wait markers for channel synchronization.....	95
2.6.22	Set alarm/error reactions.....	96
2.6.23	Evaluating data for machine maintenance.....	96
2.7	Technology cycles.....	99
2.7.1	Coordination of synchronized actions, technology cycles, part program (and PLC) .....	102
2.8	Control and protection of synchronized actions .....	104
2.8.1	Control by the PLC.....	104
2.8.2	Protected synchronized actions .....	106
2.9	Control behavior in specific operating states .....	109
2.9.1	Power On .....	109
2.9.2	RESET .....	109
2.9.3	NC STOP .....	110
2.9.4	Mode change .....	111
2.9.5	End of program .....	111
2.9.6	Response of active synchronized actions to end of program and change in operating mode .....	111
2.9.7	Block search.....	112
2.9.8	Program interruption by ASUB.....	112
2.9.9	REPOS.....	113
2.9.10	Response to alarms .....	113
2.10	Configuration .....	114
2.10.1	Configurability .....	114
2.11	Diagnostics (only with HMI Advanced) .....	116
2.11.1	Displaying status of synchronized actions .....	117
2.11.2	Displaying main run variables .....	118
2.11.3	Logging main run variables .....	118
<b>3</b>	<b>Boundary conditions .....</b>	<b>121</b>
<b>4</b>	<b>Signal Descriptions .....</b>	<b>123</b>
<b>5</b>	<b>Examples.....</b>	<b>125</b>
5.1	Examples of conditions in synchronized actions.....	125
5.2	Reading and writing of SD/MD from synchronized actions.....	126
5.3	Examples of adaptive control .....	129
5.3.1	Clearance control with variable upper limit .....	129
5.3.2	Feedrate control .....	131
5.3.3	Control velocity as a function of normalized path .....	132
5.4	Monitoring a safety clearance between two axes .....	133
5.5	Store execution times in R parameters.....	134
5.6	"Centering" with continuous measurement.....	135
5.7	Axis couplings via synchronized actions.....	138
5.7.1	Coupling to leading axis .....	138
5.7.2	Non-circular grinding via master value coupling .....	139
5.7.3	On-the-fly parting .....	143
5.8	Technology cycles position spindle.....	144
5.9	Synchronized actions in the TC/MC area .....	145
<b>6</b>	<b>Data lists.....</b>	<b>149</b>

---

6.1	Machine data.....	149
6.1.1	General machine data.....	149
6.1.2	Channelspecific machine data.....	149
6.1.3	Axis-specific machine data .....	149
6.2	Setting data .....	150
6.2.1	Axis/spindle-specific setting data .....	150
6.3	Signals .....	150
6.3.1	Signals from channel .....	150
<b>A</b>	<b>Appendix.....</b>	<b>151</b>
A.1	Feedback on the documentation.....	151
A.2	Overview .....	153
	<b>Index.....</b>	<b>155</b>



## Brief description

### Synchronized actions

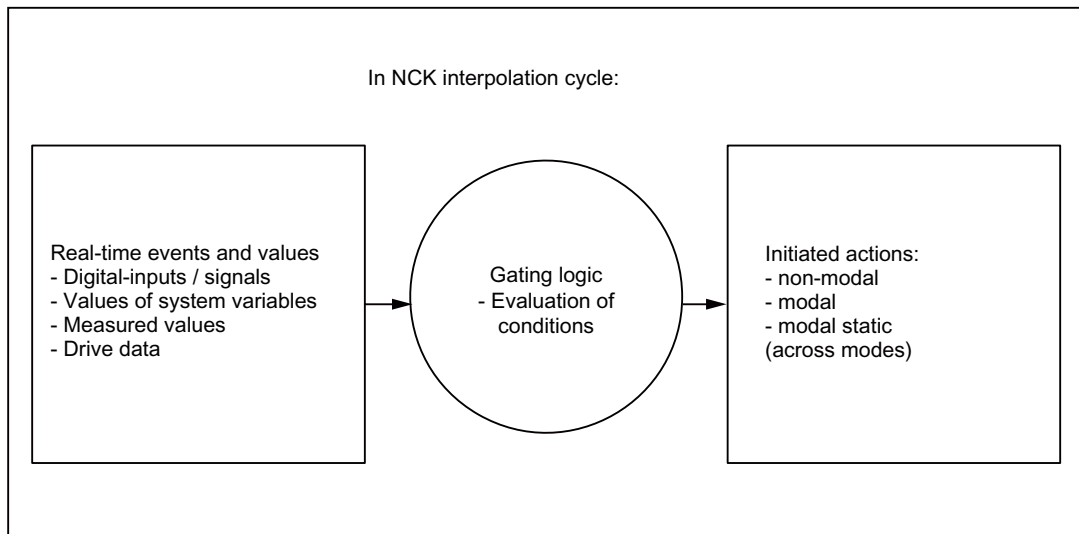
Motion-synchronous actions (or "synchronized actions" for short) are instructions programmed by the user, which are evaluated in the interpolation cycle of the NCK in synchronization with the execution of the part program. If the condition programmed in the synchronized action is fulfilled or if none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

### Applications

Possible actions in synchronized actions are e.g.:

- Output of auxiliary functions to PLC
- Writing and reading of main run variables
- Positioning of axes/spindles
- Activation of synchronous procedures, such as:
  - Read-in disable
  - Delete 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

## Schematic diagram of synchronized actions



## Documentation

The following chapters describe the functional interactions of synchronized actions.

Details of the programming of synchronized actions are available in:

**References:**

Programming Manual Advanced

## Detailed description

### 2.1 Definition of synchronized actions

The definition of synchronized actions is undertaken in the part program.

Static synchronized actions can be defined in an asynchronous subprogram (ASUP) that is activated by the PLC.

### 2.2 Components of synchronized actions

#### Components (overview)

The definition of a synchronized action consists of the following components:

	Components						
	Validity, identification number	Frequency	G code for condition and action	Condition	Action code word (fixed)	G code for action	Action or Technology-cycle
Example	IDS=1	EVERY	G70	\$AAA_IM[B] > 15	DO	G71	POS[X]=100

### 2.2.1 Validity, identification number

#### Validity

Following are the possible ways of defining the scope of validity of a synchronized action:

Specified validity	Effect
No data	<p>Synchronized actions without specified validity have the following effect:</p> <ul style="list-style-type: none"> <li>• a non-modal action, i.e. they apply only to the next block</li> <li>• only in the AUTOMATIC mode.</li> <li>• modally across all preprocessing stop blocks (including implicitly generated ones) and across implicitly-generated intermediate blocks.</li> </ul>
ID	<p>Synchronized actions with validity identifier (ID) have the following effect:</p> <ul style="list-style-type: none"> <li>• modal in the following programmed blocks</li> <li>• only in the AUTOMATIC mode.</li> </ul> <p>Effect limitation:</p> <ul style="list-style-type: none"> <li>• ID actions remain operative only until another synchronized action with the same identification number is programmed</li> <li>• up to deletion with CANCEL (i)</li> </ul> <p>See "Coordination of synchronized actions, technology cycles, part program (and PLC) (Page 102)".</p>
IDS (optional)	<p>Statically effective synchronized actions that are programmed with the validity identifier IDS, are active in all operating modes. They are also referred to as static synchronized actions.</p> <p>Applications:</p> <ul style="list-style-type: none"> <li>• AC-grinding also active in the JOG mode</li> <li>• Logic operations for Safety Integrated</li> <li>• Monitoring functions, responses to machine states in all modes</li> <li>• Optimization of the tool change</li> <li>• Cyclic machines</li> </ul>

#### Note

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 crucial for the functions of the mutual coordination of synchronized actions (refer to "Coordination between synchronized actions, technology cycles, part program (and PLC) (Page 102)").

Modal/static synchronized actions with identification numbers between 1 and 64 can be locked and released from the PLC (See "Control by the PLC (Page 104)").

Unique identification numbers must be allocated in the channel.

Examples:

Program code	Comment
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

## 2.2.2 Frequency

### Keyword of scanning frequency

A keyword (see table) is programmed to indicate how often the subsequently specified condition must be scanned and the associated action executed if the condition is fulfilled. The specified keyword is a component of the synchronized action condition.

Keyword	Scanning frequency
No data	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. In contrast to the keyword WHEN, checking of the condition continues after execution of the action/cycle until the synchronized action is deleted or disabled.

### Note

For information about technology cycles refer to the Chapter "Technology Cycles".

### 2.2.3 G code for condition and action

#### Defined initial state

The option of programming a G-code for the condition and action of a synchronized action is used to ensure that there are defined settings for the evaluation of the condition and the action/technology cycle to be executed, independently of the already active part 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.

---

#### 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.

---

#### Applications

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

### 2.2.4 Condition

#### 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 specified, then the action is executed cyclically in every interpolation cycle.

Possible conditions are e.g.:

- Comparison of main run variables such as digital or analog inputs/outputs
- Calculation of real-time expressions  
(Refer to Chapter "Real-time evaluations and calculations")
- Time/distance from beginning of block
- Measured values, measurement results
- Servo values
- Velocities, axis status

**Examples:**

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

or

Program code
WHENEVER \$AA_IM[X] > \$\$AA_IM[X1] DO ...

More examples of conditions are given in the Chapter "Examples" under the section "Conditions in synchronized actions".

**Boolean gatings**

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

**Example**

Program code	Comment
WHENEVER (\$A_IN[1]==1) OR (\$A_IN[3]==0) DO ...	; while input 1 is applied or input 3 is not applied ...

**Comparison of real-time expressions**

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.

**Example**

Programming	Comment
WHEN \$AA_IM[X2] <= \$AA_IM[X1] +.5 DO \$AA_OVR[X1]=0	; Stop, when the safety clearance is exceeded.

### System variables

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: For example `$$MN_...`, `$$MC_...`, `$$MA_...`
- Setting data: e.g. `$$SN_...`, `$$SC_...`, `$$SA_...`

---

**Note**

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

Setting data and machine data, whose value may vary during machining, must be programmed with `$$S_...`/`$$M_...` .

---

### 2.2.5 G code for 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.

---

**Note**

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

---

### 2.2.6 Action or Technology cycle

#### Actions

You can program one or more actions in each synchronized action. 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

Program code	Comment
WHEN <code>\$\$AA_IM[Y]</code> >= 35.7 DO M135 <code>\$\$A_OUT[1]</code> =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.

The consequence of actions in a technology cycle is processed sequentially in the interpolation cycle (refer to the section "Calling technology cycles").

Application examples:

- Single axis program
- 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.

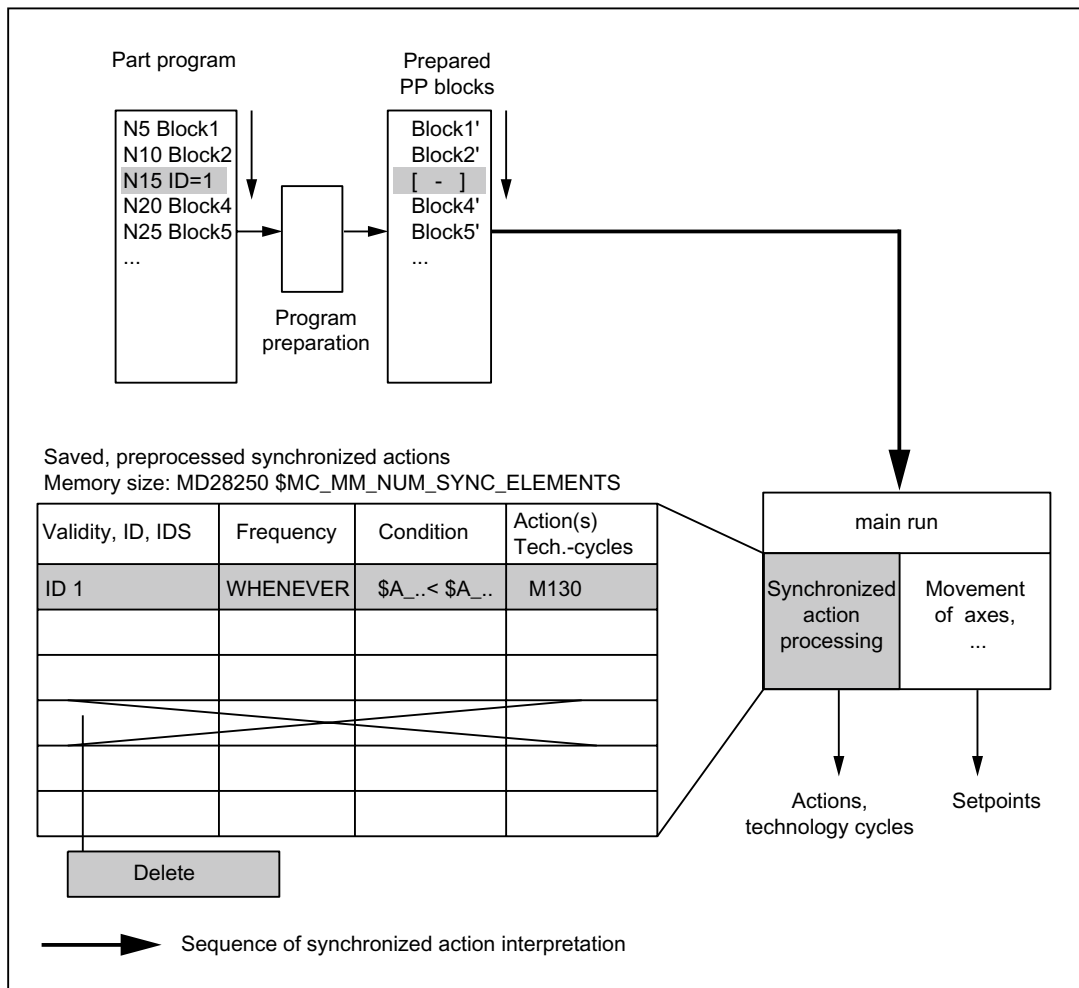


Figure 2-1 Execution of synchronized actions (schematic)

### Response for zero blocks

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:

```

Program code
N1000 G91 X0 Y0 Z0
...

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

---

**Note**

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

---

**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 synchronized action is not disabled: No `LOCK ( ID )`  
(See Chapter "Coordination of synchronized actions, technology cycles, part program and PLC").
- evaluation of the action is due as a result of the frequency keyword
- the condition is fulfilled.

**Time interval**

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.

## 2.3 List of possible actions

Possible actions in synchronized actions are:

- Output of M, S and H auxiliary functions to the PLC
- Setting (writing) of main run variables enables the following:
  - Overlaid movement: `$AA_OFF` (optional)
  - Applied tool length offsets: `$AA_TOFF` (optional)
  - Feedrate control: `$AC_OVR`, `$AA_OVR`  
Disabling a programmed axis motion
  - Assigning servo data values: `$V...=`
  - Read or write arithmetic variable: `$R[n]=`
  - Writing SD value in the main run: `$$SD`
- Read MD value at the interpolation instant in 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
- Traversing to fixed stop: `FXS` (`FXST`, `FXSW`)
- Travel with restricted moment: `FOC` (`FOCON` / `FOCOF`)

- Extended stop and retract

**References:**

Function Manual, Special Functions; Extended Stop and Retract (R3)

- Read and write system variables if appropriately designated  
(refer to manual of system variables)

**Note**

The actions are described in detail in the section "Actions in synchronized actions".

## 2.4 Real-time evaluations and calculations

### Real-time calculations

Calculations carried out in real time represent a subset of those calculations that can be performed in the NC language. They are 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 are assigned or transferred.

### Real-time expression

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 in the main run available in synchronized actions can be detected in the system variables tables (refer to System Variables Manual) from the marked fields under "HL Sync".

### Designation of main run variables

Real-time variables are all variables that begin with the following designation:

\$A...	current main-run variable
\$V...	Servo values
\$R...	R parameters

NC machine and setting data is interpreted in the main run with the following designation in synchronized actions:

- \$\$M...
- \$\$S...

---

**Note**

MD / SD values during the main run are addressed for an online access with \$\$S... or \$\$M... and evaluated during the main run, while MD or SD values 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 conversion**

An internal data type conversion can be used to assign or transfer variables to different data types during value assignments and parameter transfers.

The following data type conversions are possible:

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

**Examples:**

1. Conversion INT to REAL

---

**Program code**

```
$AC_MARKER[1]=561  
ID=1 WHEN TRUE DO $AC_PARAM[1] = $AC_MARKER[1]
```

2. Conversion REAL to INT

---

**Program code**

```
$AC_MARKER[1]=561  
ID=1 WHEN TRUE DO $AC_PARAM[1] = $AC_MARKER[1]
```

3. Conversion INT to BOOL

---

**Program code**

```
$AC_MARKER[1]=561
```

**Program code**

---

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

#### 4. Conversion from REAL to BOOL

**Program code**

---

```
R401=100.542  
WHEN $A_IN[0] == TRUE DO $A_OUT[2]=$R401
```

#### 5. Conversion from BOOL to INT

**Program code**

---

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

#### 6. Conversion from BOOL to REAL

**Program code**

---

```
R401=100.542  
WHEN $A_IN[3]==TRUE DO $R10=$A_OUT[3]
```

---

#### Note

##### Data type conversion from REAL to INT

In conversion from REAL to INT, fractional values that are  $\geq 0.5$  are rounded up, others are rounded down (compare `ROUND` function). An alarm is output and the conversion is not carried out if the values lie outside the value range for INT variables.

---

### Linking main run variables

Main run variables of the REAL and INT type can be interlinked via the following basic arithmetic operations:

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

---

**Note**

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

---

**Expressions from basic arithmetic operations**

Expressions from basic arithmetic operations can be bracketed and nested (refer to the section "Priority of Operators").

**Relational operators**

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 logic operators**

The following logical bit operators may be used:

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

## 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.	Negation, bit-serial negation	NOT, B_NOT
2.	Multiplication, division	*, /, DIV, MOD
3.	Addition, subtraction	+, -
4.	bit AND	B_AND
5.	bit-serial exclusive OR	B_XOR
6.	bit OR	B_OR
7.	AND	AND
8.	Exclusive OR	XOR
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	<=

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

Example of a multiple expression:

---

### Program code

```
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

---

### Program code

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

The meaning of the specified functions are explained in:

**References:**

- Programming Manual fundamentals
- Programming Manual Advanced

## Indexing

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

Example

**Program code**

---

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

The index \$AC\_MARKER[1] is evaluated currently in the interpolation cycle.

Restrictions:

- It is not permissible to nest indices with main run 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]]

## 2.5 Special main run variables for synchronized actions

### List of all system variables

A full list of all addressable system variables is available in:

**References:**

- Manual of System Variables

### System variables in synchronized actions

The system variables that can be addressed in synchronized actions can be detected from the "SA" field in the System Variable tables.

If the system variables table is ticked additionally for the "HL Sync" field, then a system variable updated in the main run is also synchronized in the main run.

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)

## 2.5.1 Marker/counter variables

### Channel-specific marker \$AC\_MARKER[n]

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

Data type: INT  
n: Array index of marker 0-n

### Number

The number of \$AC\_MARKER[n] markers per channel is set using machine data:

MD28256 \$MC\_MM\_NUM\_AC\_MARKER

Max. value: 20000

These markers exist once in each channel under the same name.

### Storage location

The memory location for \$AC\_MARKER[n] can be defined with the machine data:

MD28257 \$MC\_MM\_BUFFERED\_AC\_MARKER

Value	Storage location
0	Dynamic NC memory (standard setting)
1	Static NC memory

One element requires 4 bytes. Please ensure that sufficient memory of correct type is available.

### Note

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

### Response to POWER ON / NC reset / TP end

On Power On, NC reset and end of part program, the markers are set to "0". Thus the same start conditions are given for each program run.

## 2.5.2 Timers

### \$AC\_TIMER[n]

System variable \$AC\_TIMER[n] permits actions to be started after defined periods of delay.

Data type: REAL  
 n : Number of timer variables  
 Unit: Second

### Number

The number of available timer variables is set using machine data:

MD28258 \$MC\_MM\_NUM\_AC\_TIMER

### Setting timers

Incrementation of a timer variable is started by means of value assignment:

\$AC\_TIMER[<n>]=<value>

with: <n> Number of time variables  
 <value> Starting value (generally "0")

### Stopping timers

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

\$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:

Program code	Comment
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	

### 2.5.3 Synchronized action parameters

#### \$AC\_PARAM[n]

The variables \$AC\_PARAM[n] are used for calculations and as buffer storage and can be read and written in synchronized actions.

Data type: REAL  
n : Number of parameters 0 - n

#### Number

The number of available AC parameter variables in each channel is defined via machine data:

MD28254 \$MC\_MM\_NUM\_AC\_PARAM

Max. value: 20000

These parameters exist once in each channel under the same name.

#### Storage location

The memory location for \$AC\_PARAM[n] can be defined with the machine data:

MD28255 \$MC\_MM\_BUFFERED\_AC\_PARAM

Value	Storage location
0	Dynamic NC memory (standard setting)
1	Static NC memory

One element requires 8 bytes. Please ensure that sufficient memory of correct type is available.

#### Note

Synchronized action parameters stored in the static NC memory can be included in the data backup.

#### Response to POWER ON / NC reset / TP end

On Power On, NC reset and end of part program, the parameters are set to "0". Thus the same start conditions are given for each program run.

## 2.5.4 R parameters

### Application in synchronized actions

By programming the \$ sign in front of R parameters, they can also be used in synchronized actions:

- R[n] or Rn: Calculations in part programs
- \$R[n] or \$Rn: Calculations in synchronized actions

Data type: REAL  
n : Number of the field variables

### Storage location

R parameters are stored in static NC memory persistently and they therefore get their values across POWER ON, NC reset and part program end.

### Examples

Example 1: Import measured value in the R-parameter

Program code	Comment
WHEN \$AC_MEA==1 DO \$R10=\$AA_MM[Y]	; If valid measurement is present, import measured value in R- parameter.

Example 2: Evaluation of the R-parameter

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

## 2.5.5 Machine and setting data

### Reading and writing MD and SD

It is also possible to read and write machine data and setting data from synchronized actions. Access must be differentiated according to the following criteria:

- MD and SD that remain unchanged during machining
- MD and SD whose settings change during machining.

### Reading at the time of preprocessing

Machine and setting data whose settings do not vary are addressed from synchronized actions in the same way as in normal part program commands. They are preceded with a \$ sign.

Example:

---

**Program code**


---

```
ID=2 WHENEVER $AA_IM[z]<$$SA_OSCILL_REVERSE_POS2[Z]-6 DO $AA_OVR[X]=0
```

Here, reversal range 2, assumed to remain static during operation, is addressed for oscillation.

### Reading at the time of the main run

Machine data and setting data whose values may change during machining are addressed from a synchronized action preceded by the \$\$ sign.

Example

---

**Program code**


---

```
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 reverse position could be changed at any time through 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:

Lists (Book 1)

Addressing:

Machine data and setting data to be written are preceded by \$\$.

Example

Program code	Comment
<pre>ID=1 WHEN \$AA_IW[X]&gt;10 DO \$\$SN_SW_CAM_PLUS_POS_TAB_1[0]=20 \$\$SN_SW_CAM_MINUS_POS_TAB_1[0]=30</pre>	<pre>; Changing the switching positions of SW cams from 20 to 30.</pre>

## 2.5.6 FIFO variables (circulating memory)

### Application

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

### Structure

The memory structure of a `FIFO`-variable is shown in the figure: Example of `FIFO` 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 `FIFO`-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 `FIFO`. Read and write access can be assigned to each element of the `FIFO`.

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

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

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

---

### Note

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

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

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

The FIFO 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 FIFO variables.

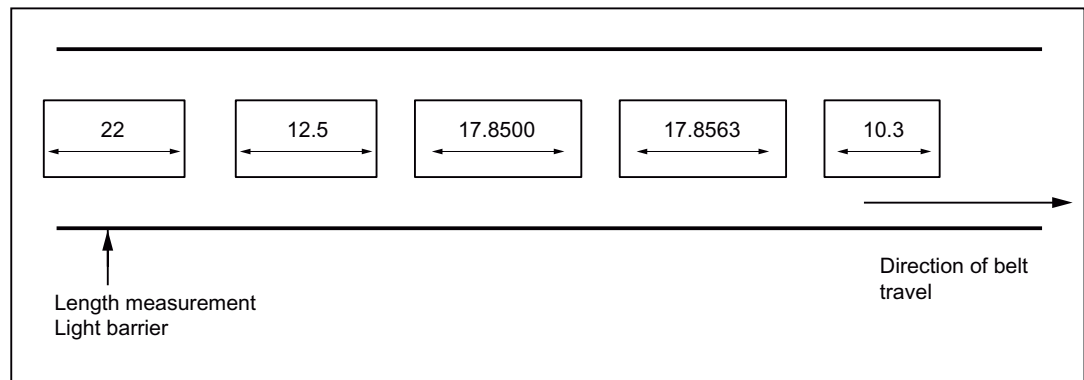


Figure 2-2 Product lengths of sequence of parts on conveyor belt

2.5 Special main run variables for synchronized actions

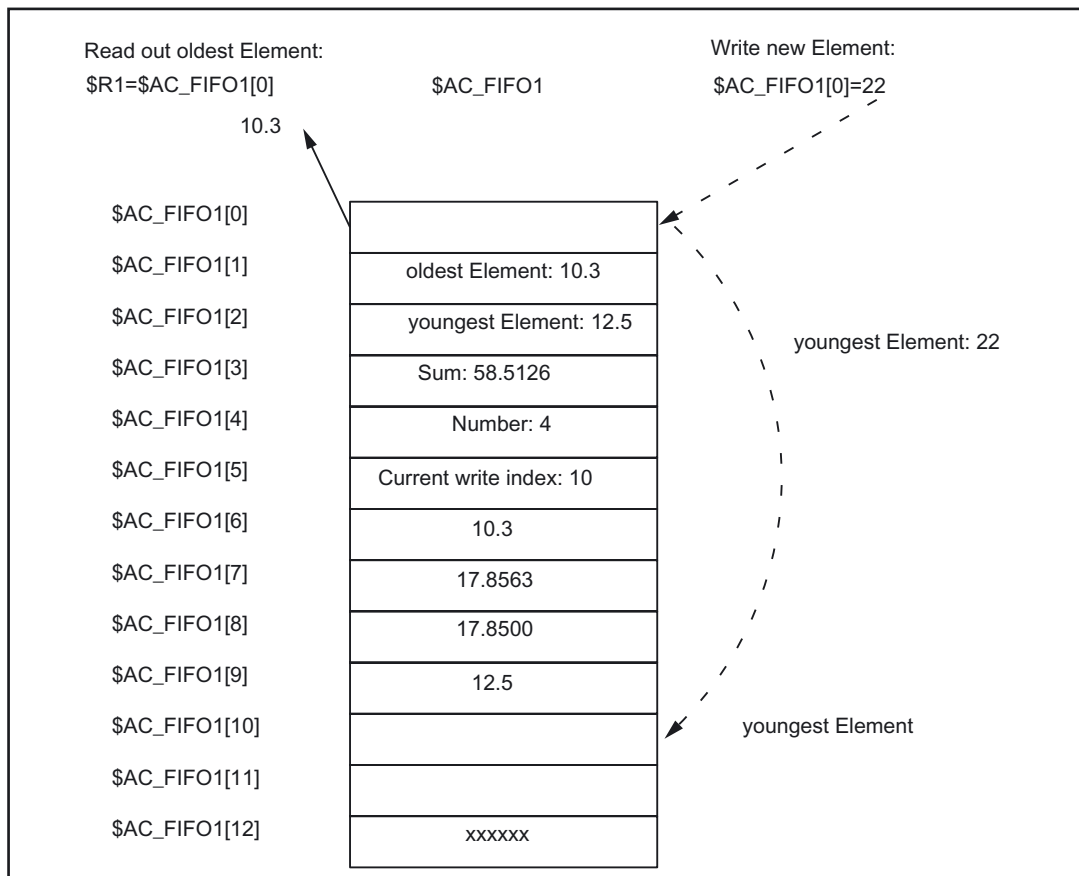


Figure 2-3 FIFO variable

2.5.7 SRAM stored system variables

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

## 2.5.8 Determining the path tangent in synchronized actions

### \$AC\_TANEB

The system variable \$AC\_TANEB (Tangent ANgle at End of Block), 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;
```

## 2.5.9 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:

$$\begin{aligned} & \$AA\_OVR \quad * \quad \$AA\_PLC\_OVR \quad \text{or} \\ & \$AC\_OVR \quad * \quad \$AC\_PLC\_OVR \end{aligned}$$

## 2.5.10 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.

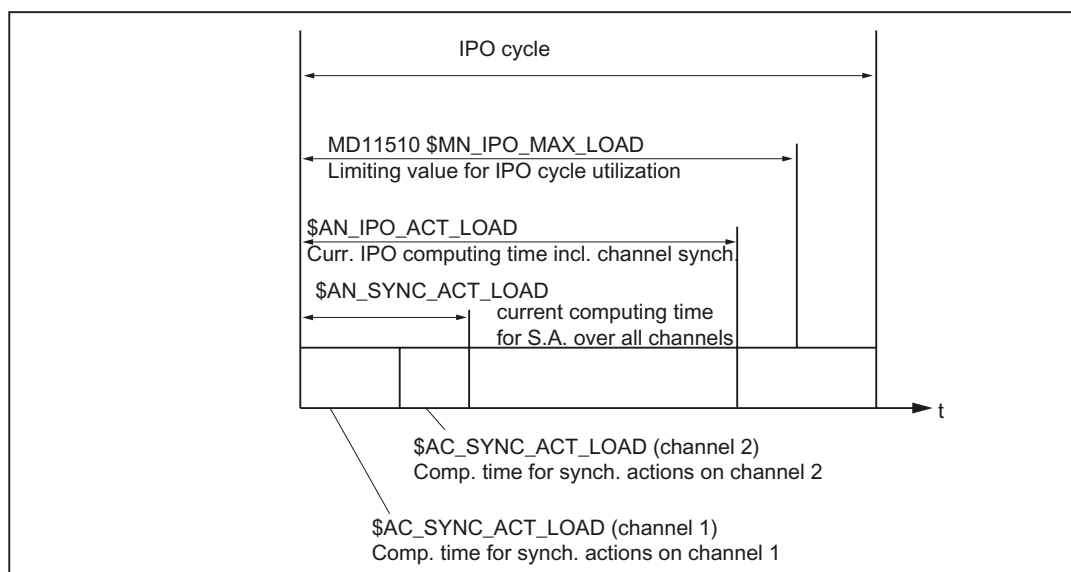


Figure 2-4 Time components of the synchronized actions in the IPO cycle

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 (%)
\$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%.

## 2.5.11 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".

---

## 2.6 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- 1    Actions in synchronized actions

... DO ...	Description	Reference
Mxx Sxx Hxx	Output of auxiliary functions to PLC	2.4.1
SETAL(no.)	Set alarm, error reactions	

2.6 Actions in synchronized actions

... DO ...	Description	Reference
\$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 – applied tool length offsets	2.4.8
	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
	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	

... DO ...	Description	Reference
MEAWA MEAC	Measurement without deletion of distance-to-go Cyclical measurement	2.4.21
SETM CLEARM	Channel synchronization: Setting a wait mark Deleting a wait mark	2.4.22
LOCK UNLOCK RESET	Coordination of synchronized actions: - Disable synchronized action/technology cycle - Release synchronized action/technology cycle - Reset technology cycle	2.5.1

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

### References

For general information about auxiliary function outputs, please see:  
Function Manual Basic Functions; Auxiliary Function Output to PLC (H2)

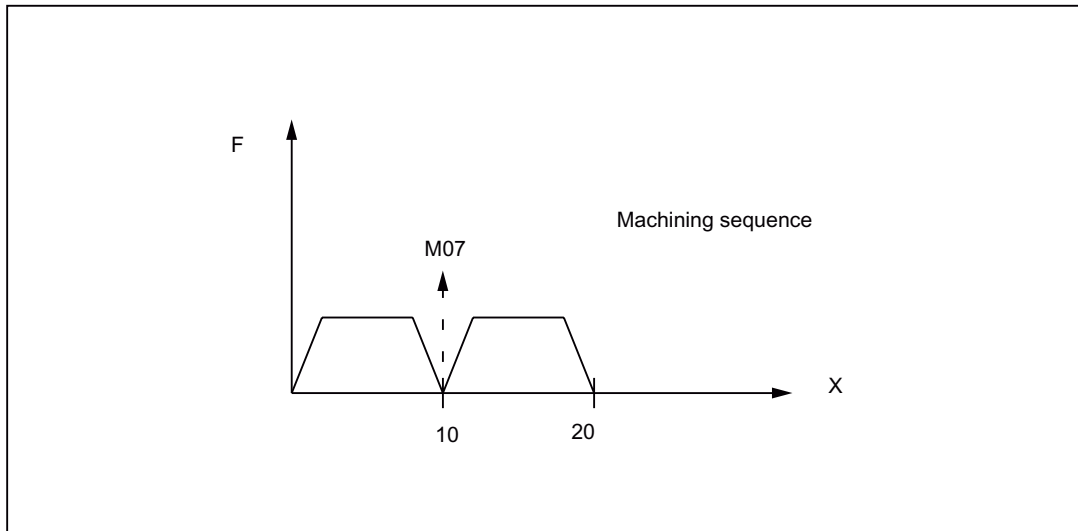
### Auxiliary function output from synchronized actions

The advantage of implementing auxiliary function outputs in synchronized actions is illustrated by the following example:

Example Switch on coolant at a specific position

1. Solution without a synchronized action: 3 blocks

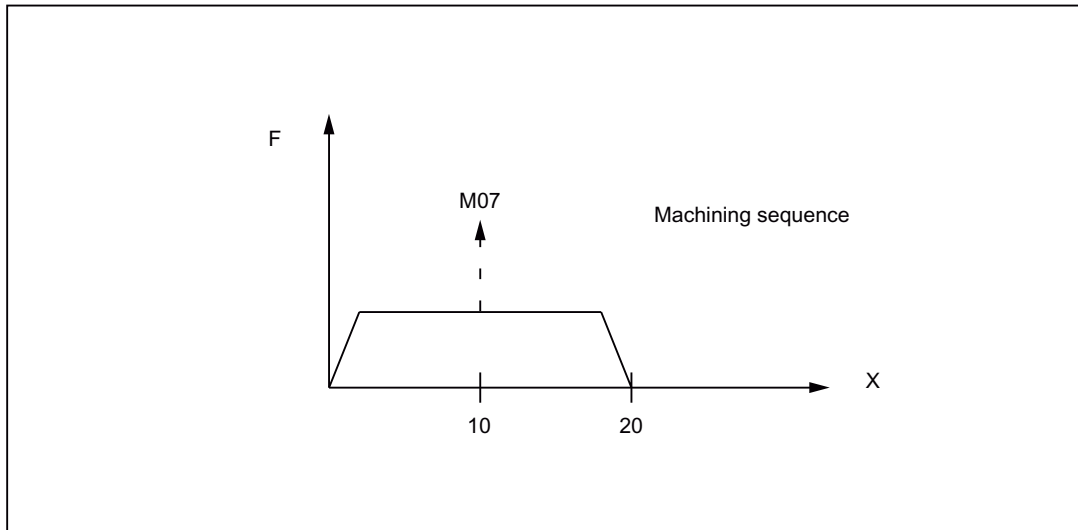
Program code
N10 G1 X10 F150
N20 M07
N30 X20



2. Solution with a synchronized action: 1 set

Program code

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



Output timing

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 times which may have been defined in one of the following machine data become ineffective:

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)

### Maximum number of auxiliary function outputs:

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

### Programming

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

Example

Program code	Comment
<code>WHEN \$AA_IM[X] &gt; 50 DO H15 S3000 M03</code>	; If the actual value of the X axis is greater than 50, the output is H15, set new spindle speed and a new direction of rotation.

#### Note

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

However, the spindle M functions `M3`, `M4`, `M5`, `M40`, `M41`, `M42`, `M43`, `M44`, `M45`, `M70` and `M17` are permitted as an end of a technology cycle.

### Message acknowledgment

Technology cycle blocks (see Section on "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: 2147483648 to 2147483647.

## 2.6.2 Setting (writing) and reading of main run variables

### Write

In synchronized actions the main run 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

### 2.6.3 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.

---

## 2.6.4 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 a 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 "Polynomial 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 main run variables.
- Individual polynomial parameters can be overwritten using the same method used to write main run variables. Permissible generally in part program and in action section of synchronized actions. See Chapter. "Setting (writing) and reading of main run 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).

## 2.6.5 Polynomial evaluation SYNFACT

### 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,  
Main run variable output,  
Main run variable input)

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

### Operating principle of SYNFACT

The polynomial defined with 'Polynomial number' is evaluated with the value of the 'Main run variable input'. The result is then subjected to maximum and minimum limits and assigned to the 'Main run 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 'Main run 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 or
- Direct

### 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 'main run variable output':

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

### Example Additive control of path feed

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  $\pm 100$  mm/min. The magnitude of the axial current deviation may be  $\pm 1$  A.

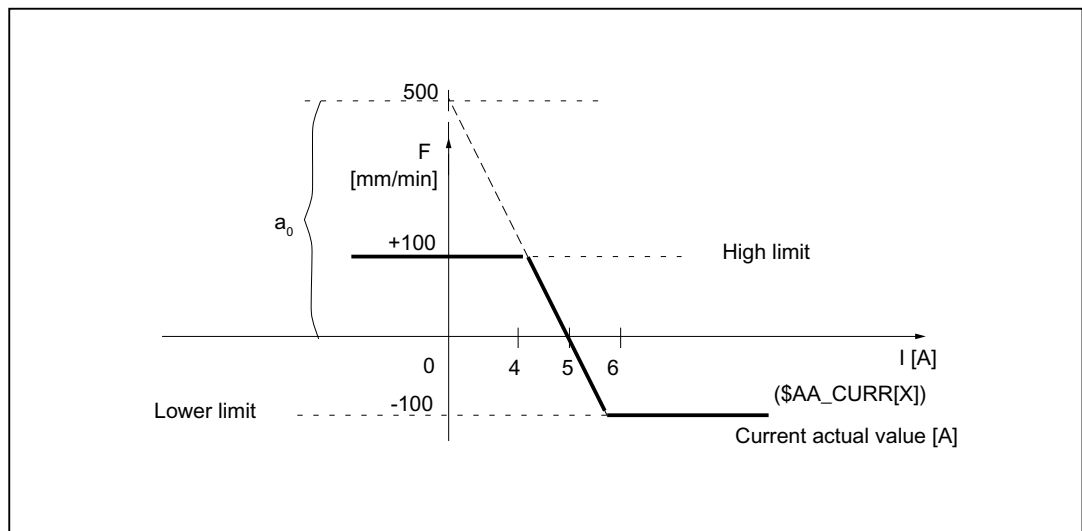


Figure 2-5 Example Additive control of path feed

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

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

$$a_1 = 100\text{mm} / (1\text{min} \cdot \text{A})$$

$$a_1 = -100 \rightarrow \text{Control constant}$$

$$a_0 = -(-100) \cdot 5 = 500$$

$$a_2 = 0 \text{ (not a square component)}$$

$$a_3 = 0 \text{ (not a cubic component)}$$

$$\text{Upper limit} = 100$$

$$\text{Lower limit} = -100$$

2.6 Actions in synchronized actions

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 SYNFACT(1, $AC_VC[x], ; the additive offset value for the feedrate of
$AA_LOAD[x])                    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).

$$F_{active} = F_{programmed} \cdot Factor_{AC}$$

Variable \$AC\_OVR that acts as a *multiplicative* factor on the machining process is used as the main run variable output.

**Example 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(axis) must stop at 80% drive load. An excessive velocity corresponding to the programmed value +20% is permissible.

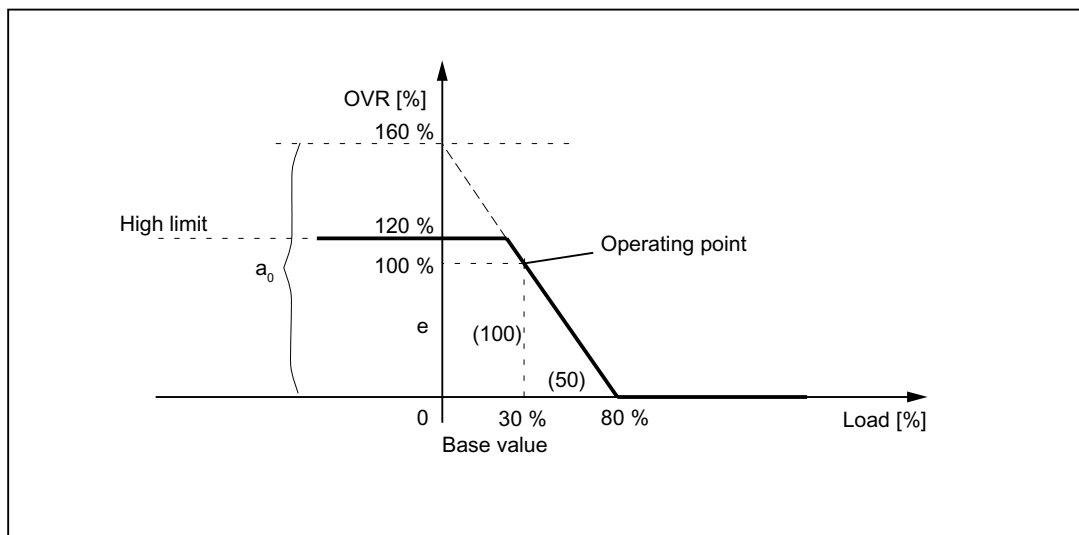


Figure 2-6 Example Multiplicative control

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

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

$$a_1 = - 100\% / (80\% -30\%) = -2$$

$$a_0 = 100+ (2 * 30) = 160$$

$a_2 = 0$  (not a square component)

$a_3 = 0$  (not a cubic 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 SYNFACT(2, $AC_OVR, ; the path override is calculated in each
$AA_LOAD[x])                interpolation cycle 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 action)

0: Proportional evaluation

1: integral evaluation

The value to be offset in absolute mode (main run variable output) can also be limited to the value. It is stored in 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	Meaning
-1	Limit of offset value reached in the negative direction.
1	Limit of offset value reached in the positive direction.
0	The offset value is not within the limit range.

#### Application:

The SYNFACT function can be used in conjunction with system variable \$AA\_OFF to implement clearance control in laser machining operations.

## 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").

### Constraints:

*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).

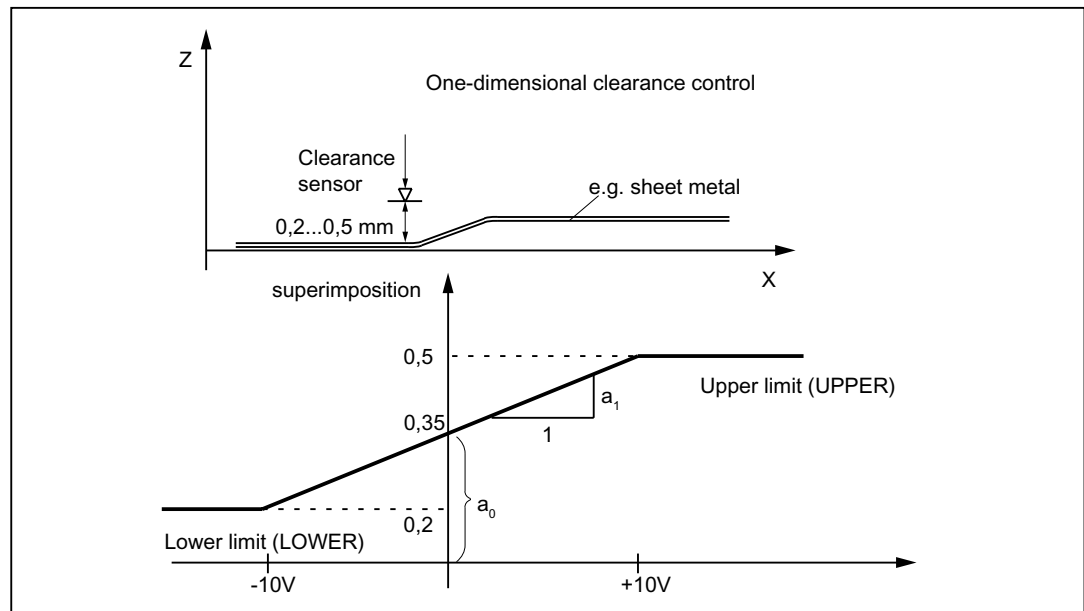


Figure 2-7 Clearance control

```

%_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 SYNFACT(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
$AA_OVR[X] = 0 ; overshoot
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
    
```

## 2.6.6 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 action)

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 action)

Bit1 = 0: \$AA\_OFF is deselected in case of RESET

Bit1 = 1: \$AA\_OFF is 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.

## Constraints

- **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 `ASUP` 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 part 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 main run 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 main run variables that are defined in the MCS.

## 2.6.7 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).

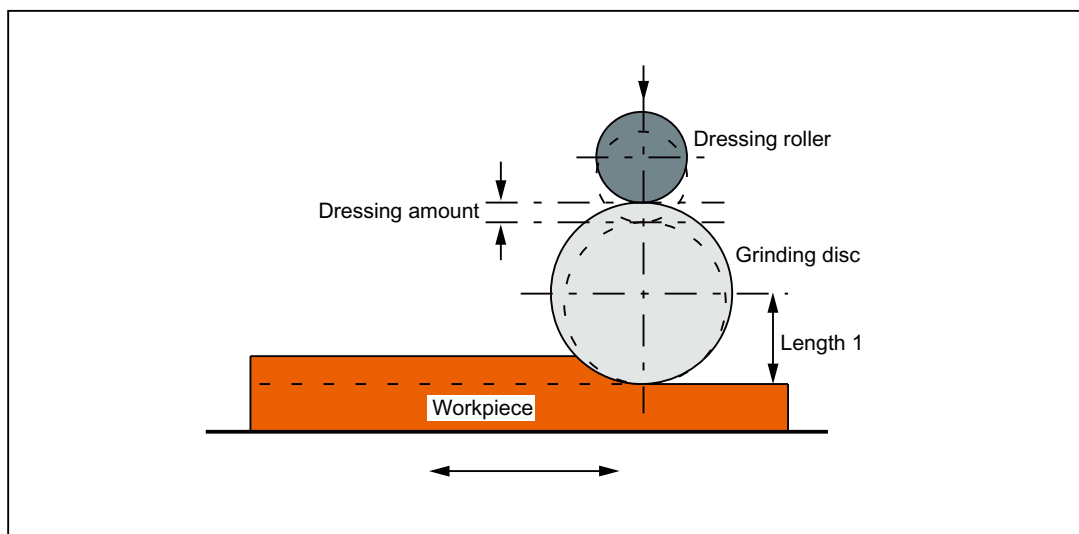


Figure 2-8 Dressing during machining using a dressing roller

### 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 <code>FCTDEF</code> .
Real-main variable_read:	All main variables of the type <code>REAL</code> 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 <b>added</b> .
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 <code>FTOCON</code> 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].

---

## 2.6.8 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:

#### Reference:

/PGA/ Programming Manual Advance; 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. Thus, the number of active directions of offset is determined by the geometry axes that are 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 direction

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 direction) 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 direction

Query in synchronized action ; for the Z-tool direction a
N30 WHEN TRUE DO $AA_TOFF[Z] = 10 ; Tool length offset = 10
G4 F5 ; interpolated
...
Static synchronized action ; for the X-tool direction 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 direction 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 direction 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
; traversed, the position offset is added

```

---

N90 TRAFOOF

to the actual position in WCS in accordance with the current orientation.

### 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 direction)

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 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`.

## Constraints

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.

## 2.6.9 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.

## 2.6.10 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).

## 2.6.11 DELDTG

### Delete distance-to-go

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

- High-speed prepared delete distance-to-go

### High-speed, prepared DDTG

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

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

### DELDTG

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

After executing delete distance-to-go, 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 delete distance-to-go.

#### Restrictions:

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

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

#### Commands:

`MOVE=1`: Works on indexing axes with and without Hirth tooth system  
`MOVE=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 delete 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 delete 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) ; delete distance-to-go for Axis X1,
                                     if the voltage of 8 V is exceeded at
                                     Input 2
POS[X1] = 100 ; Next position
R10 = $AA_DELT[X 1] ; Import axial distance-to-go in R10
```

After successful delete distance-to-go the variable \$AA\_DELT[axis] contains the axial distance to go.

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

## 2.6.12 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).

## 2.6.13 Traversing command axes

In synchronized actions, axes can be positioned, started and stopped asynchronously to the part program. An axis which is moved via synchronized actions is referred to as a **command axis**.

### Alternate traversing from the part program and from synchronized actions

An axis cannot be traversed from the part program and from synchronized actions simultaneously. Alternate traversing from the part program and synchronized actions is possible. Delays may occur if an axis is traversed from the part program again following the end of the traversing motion, as a command axis.

**Axis status following the end of the part program or NC-RESET**

Machine data can be used to set whether, following the end of the part program or NC-RESET, an axis should become a channel axis again or should remain or become a command axis:

Axis status before PP end/NC-RESET <sup>1)</sup>	MD <sup>2)</sup>	Axis status after PP end/NC-RESET <sup>1)</sup>
Channel axis	FALSE	Channel axis
Command axis	FALSE	Channel axis
Channel axis	TRUE	Command axis
Command axis	TRUE	Command axis
1) PP end = End of part program		
2) MD = MD30450 \$MA_IS_CONCURRENT_POS_AX		

**Examples of traversing a command axis**

The command axis's traversing motion is programmed directly in the synchronized action:

Program code	Comment
ID=1 EVERY G710 \$A_IN[1]==1 DO POS[X]=100	; direct programming

The command axis's traversing motion is programmed in an AXIS-X technology cycle, and this is called in the synchronized action:

Part program	Comment
ID=1 EVERY G710 \$A_IN[1]==1 DO POS[X]=100	; indirect programming

Technology cycle: AXIS_X
N10 M100
N20 G71 POS[X]=100
N30 M17

See Section: "Technology cycles (Page 99)"

## Definition of the system of units

Unless a particular system of units (imperial / metric) is specifically programmed in a synchronized action (condition part and/or action part), the system of units which is active in the part program at the time of execution takes effect in the synchronized action which is executed parallel to the part program. This has the following effects on the synchronized action:

- G70/G71 active in the part program:
  - All the **programmed** position values are interpreted in the **programmed** system of units.
  - All the **read** position data are interpreted in the **parameterized basic system**.
- G700/G710 active in the part program:
  - All the **programmed** position values are interpreted in the **programmed** system of units.
  - All the **read** position data are interpreted in the **parameterized basic system**.

To enable a defined system of units to take effect in the synchronized action, the system of units must be explicitly specified with G70/G71/G700/G710 in the condition and action parts. The following rules apply:

- If a system of units is programmed in the condition part, this also takes effect in the action part if a system of units has not been specifically programmed in it.
- If there is only a system of units programmed in the action part, the system which is currently activated in the part program takes effect in the condition part.
- Different systems of units can be programmed in the condition and action parts.
- The system of units programmed in the synchronized action has no impact on the part program.

Program code	Comment
N10 ID=1 EVERY \$AA_IM[Z]>200 DO POS[Z2]=10	; \$AA_IM: ?? ; 200: ?? ; 10: ??
N20 ID=2 EVERY \$AA_IM[Z]>200 DO G70 POS[Z2]=10	; \$AA_IM: ?? ; 200: ?? ; 10: inch
N30 ID=3 EVERY G71 \$AA_IM[Z]>200 DO POS[Z2]=10	; \$AA_IM: ?? ; 200: mm ; 10: mm
N40 ID=4 EVERY G71 \$AA_IM[Z]>200 DO G70 POS[Z2]=10	; \$AA_IM: ?? ; 200: mm ; 10: inch
N50 ID=5 EVERY \$AA_IM[Z]>200 DO G700 POS[Z2]=10	; \$AA_IM: ?? ; 200: ?? ; 10: inch
N60 ID=6 EVERY G710 \$AA_IM[Z]>200 DO POS[Z2]=10	; \$AA_IM: mm ; 200: mm ; 10: mm
N70 ID=7 EVERY G710 \$AA_IM[Z]>200 DO G700 POS[Z2]=10	; \$AA_IM: mm ; 200: mm ; 10: inch

2.6 Actions in synchronized actions

Program code	Comment
??:	depending on the configured basic system or the programmed system of units for the part program

**Note**

**Technology cycles**

If a technology cycle is being used, the system of units can also be programmed in the technology cycle instead of the system of units having to be explicitly assigned in the action part.

**References:**

/PG/ Programming Manual Fundamentals; Position data

**Absolute/incremental traversing motions**

Unless a particular traversing mode (absolute / incremental) is explicitly programmed in a synchronized action (condition part), the traversing mode which is active in the part program at the time of execution takes effect in the synchronized action which is executed parallel to the part program.

Since only the G functions G70/G71/G700/G710 may be programmed in a synchronized action, the operator has to program the traversing mode via specific operations in the action part to enable a defined traversing mode to take effect in the synchronized action.

Operation	Description
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 (...)	Traverse axis 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

**Example: Incremental traversing by a fixed value**

If the condition is met, axis X is traversed incrementally by 10 mm:

Program code
ID=1 EVERY G710 \$AA_IM[B]>75 DO POS[X]=IC(10)

**Example: Absolute traversing to a position calculated in real time**

If the condition is met, axis X is traversed to the position calculated at that time:

Program code
ID=1 EVERY G710 \$AA_IM[B]>75 DO POS[X]=\$AA_MW[V] - \$AA_IM[W] + 13.5

**Behavior with active axial frames**

If programmable and variable frames and tool length corrections are not explicitly calculated via machine data for synchronized actions:

MD32074 \$MA\_FRAME\_OR\_CORRPOS\_NOTALLOWED, Bit9 = 1

deactivated, the frame and/or tool length correction which is active in the part program at the time of execution takes effect in the synchronized action which is executed parallel to the part program.

**Example 1**

Traversing motions in synchronized actions with activated frames / tool length corrections (Bit9 = 0):

Program code	Comment
N100 TRANS X20	; work offset in X: 20 mm
IDS=1 EVERY G710 \$A_IN==1 DO POS[X]=40	; With \$A_IN==1, X travels to position 60 mm
...	
N130 TRANS X-10	; work offset in X: -10 mm
...	; With \$A_IN=1, X travels to position 30 mm

**Example 2**

Traversing motions in synchronized actions with deactivated frames / tool length corrections (Bit9 = 1):

Program code	Comment
N100 TRANS A=0.001	; work offset in X: 0.001 degrees
N120 POS[A]=270	; A travels to position 270.001 degrees
IDS=1 EVERY G710 \$A_IN==1 DO POS[A]=180	; With \$A_IN=1, ; A travels to position 180.000 degrees.
N130 POS[A]=90	; A travels to position 90.001 degrees
N140 POS[A]=CAC(1)	; encoded position 1 = 100 degrees ; A travels to 100.001 degrees
N150 POS[A]=CIC(1)	; encoded position 2 = 200 degrees ; A travels to 200.001 degrees

**Note**

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

## PLC command axis control

A command axis which is started by a static synchronized action (IDS) becomes independent of the status of the part program (NC-STOP, alarm handling, end of program, program controls and NC-RESET) containing the synchronized action if control of the command axis has been taken over by the PLC:

DB31, ... DBX28.7 == 1 (PLC controls axis)

You can find detailed information about PLC command axis control in:

### References:

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

## 2.6.14 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 main run 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 FPRAON, FPRAOF. See also:

### Reference:

/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.

## 2.6.15 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.
```

## 2.6.16 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` (Parametrizing 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 positioning 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  $\$MN\_AXCHANGE\_MASK$  (Parametrizing 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:**

```

N150 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.

**Transfer AXTOCHAN 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 an 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.

### Reference:

/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.

---

## 2.6.17 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 ↓	To →	POS	MOV=1 MOV = -1	MOV=0	SPOS	M3 M4	M5	LEADON	TRAIL ON
<b>Axis stationary</b>									
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		
<b>Axis in motion</b>									
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.

**2.6.18 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.

---

## 2.6.19 Activating/deactivating coupled motions and couplings

### Introduction

#### Reference:

/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(axis) 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 gear unit

With the help of the "Electronic gearbox" the movement of a following axis FA can be interpreted depending on up to five leading axes LA. This way a gearbox group from the part program:

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

- 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)
```

with:

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
```

with:

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)  
with:  
LW        Master value  
n         Number of curve table  
Degree    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)  
with:  
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
Degree	Pitch parameters, result: 2 more options parameters for scaling: - Following axis - 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)
with:
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.

#### Reference:

/PGA1/ Parameter Manual, System Variables

### Detection of synchronism

System variable `$AA_SYNC[ax]` can be read from the part 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:

**Reference:**

/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. Details can be found in: /FB3/ Function Manual, Special Functions; Coupled axes and ESR (M3).

---

### Examples

You can find an example of axis coupling using curve tables in Chapter "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 `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 constraints.

---

**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 Chapter "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 TRAIL, 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"

## 2.6.20 Measurements from synchronized actions

### Introduction

Measurement functions available for the part programs:

MEAS, MEAW, MEASA, MEAWA, MEAC

**Reference:**

/PGA/ Programming Manual Advanced

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

Only the following may be used in synchronized actions:

- MEAWA Axial measurement without delete 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 delete 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- 2 Mode meanings:

Tens decade	Units decade	Meaning
	0	Measurement task is aborted
	1	Up to 4 trigger events can be activated <b>simultaneously</b>
	2	Up to 4 trigger events can be activated <b>consecutively</b>
	3	Up to 4 trigger events can be activated <b>consecutively</b> , 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 delete distance-to-go

If needed, delete 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_MEACTION[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   GO X0                               ; Mode = 1, simultaneous
N1   MEAC[X]=( 1, 1, 1) POS[X]=100      ; No-FIFO = 1
                                           ; Trigger event 1 = rising edge,
                                           ; Probe 1
N2   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   GO X0                               ; Rapid traverse to the starting
                                           position
N1   $AC_MARKER[1]=1                     ; Marker 1 as index for arithmetic
                                           variable R[..]
N2   ID=1 WHENEVER $AC_FIFO1[4]>=1      ; Synchronized action as check: if 1
DO $R[$AC_MARKER[1]]= $AC_FIFO1[0]      ; or more measurement values exist in
$AC_MARKER[1]=$AC_MARKER[1]+1           ; FIFO variable, export oldest value
                                           ; from FIFO and store in current R[..],
```

```

; increase index for R by 1
N3 MEAC[X]=( 1, 1, 1, -1) POS[X]=100 ; Activate continuous measurement,
; movement after X = 100
; Mode = 1, simultaneously
; Nr_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 delete 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,
; deselect continuous measurement and
; Deleting the distance-to-go
N2 MEAC[X]=( 1,1,1,-1) G01 X100 F500 ; continuous measurement active from the
; part program.
; Mode = 1, simultaneously
; Nr_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:

State	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.

### 2.6.21 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).

## 2.6.22 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 main run 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

#### Reference:

/PGZ/ Programming Manual Cycles

## 2.6.23 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 <b>high speeds</b> 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 <sup>3</sup>
\$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
```

## 2.7 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.

### 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.

The search path calls a technology cycle, as in subprograms and cycles.

End of program in technology cycle is programmed with M02 / M17 / M30 / RET.

---

**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.

---

**Examples**

**Example 1: Calling a technology cycle**

Main program:

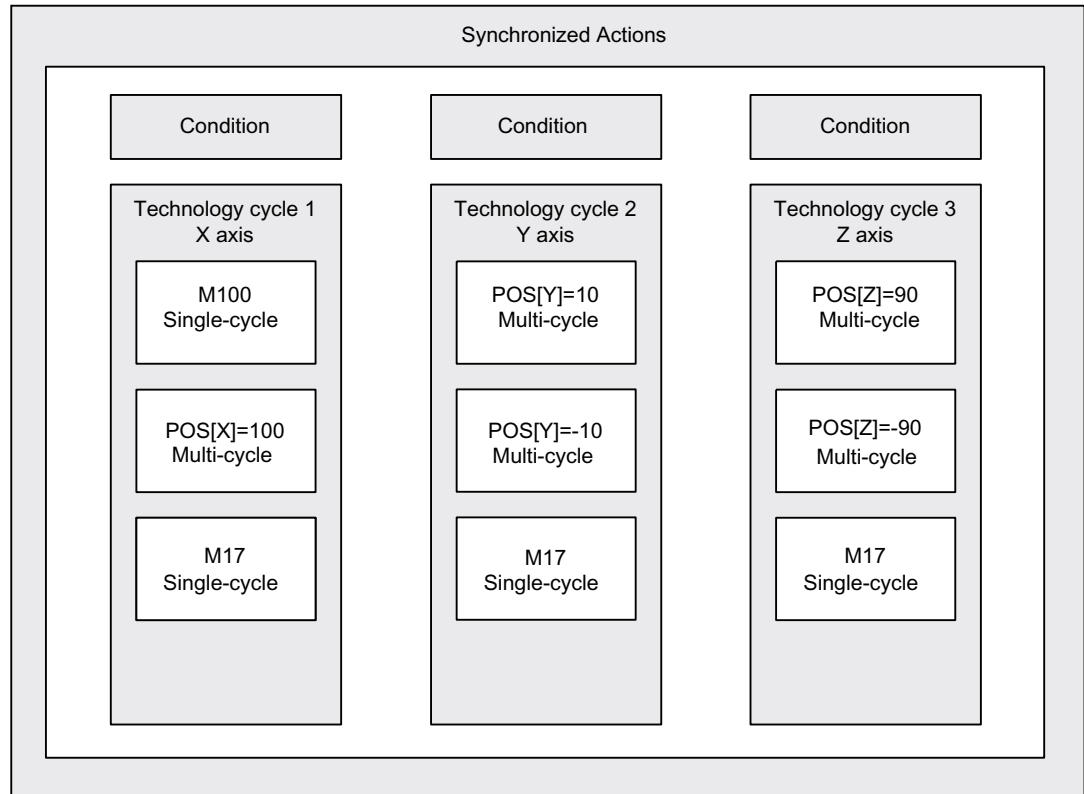
Program code	Comment
...	
ID=1 EVERY \$AA_IM[Y]>=10 DO AX_X	; AX_X: Subprogram name of axis program for X axis
...	

Axis program AX\_X:

Program code
POS[X]=\$R[7] FA[X]=377
\$A_OUT[1] = 1
POS[X]=R10
POS[X]=-90
M30

### Example 2: Coordinated axis movements

Different axis programs can be started by setting digital NC inputs:



Main program:

**Program code**

```
...
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 ACHSE_Z
M30
```

Axis program ACHSE\_X:

**Program code**

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

Axis program ACHSE\_Y:

Program code
POS[Y]=10
POS[Y]=-10
M17

Axis program ACHSE\_Z:

Program code
POS[Z]=90
POS[Z]=-90
M17

## 2.7.1 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.

#### 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.

Besides, the following key words are available for coordinating technology cycles:

Keyword	Meaning	TP <sup>1)</sup>	SA <sup>2)</sup>
LOCK (ID)	Disable technology cycle. 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.  Depending on the type of synchronized action, actions are executed once more when the condition is fulfilled again. Already executed synchronized actions of the type WHEN are not processed again after RESET.		+
CANCEL (ID)	Synchronized action is deleted.	+	
<sup>1)</sup> Call permitted in the part program <sup>2)</sup> Call permitted in synchronized action/technology cycle			

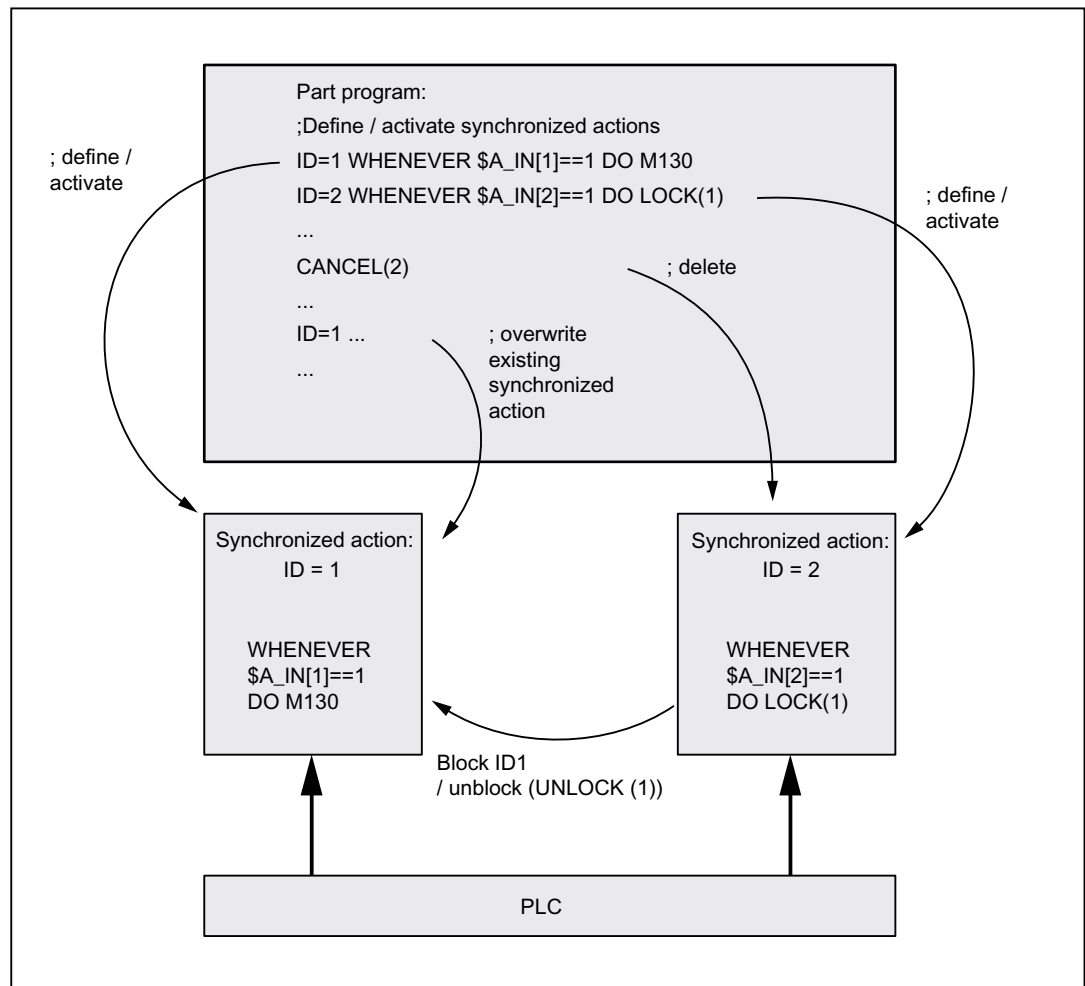


Figure 2-9 Setting up/locking/deleting modal synchronized actions

**Note**

LOCK (ID) , UNLOCK (ID) via PLC, see Chapter "Control from PLC".

## 2.8 Control and protection of synchronized actions

### 2.8.1 Control by the 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 synchronized 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 disabling is 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.

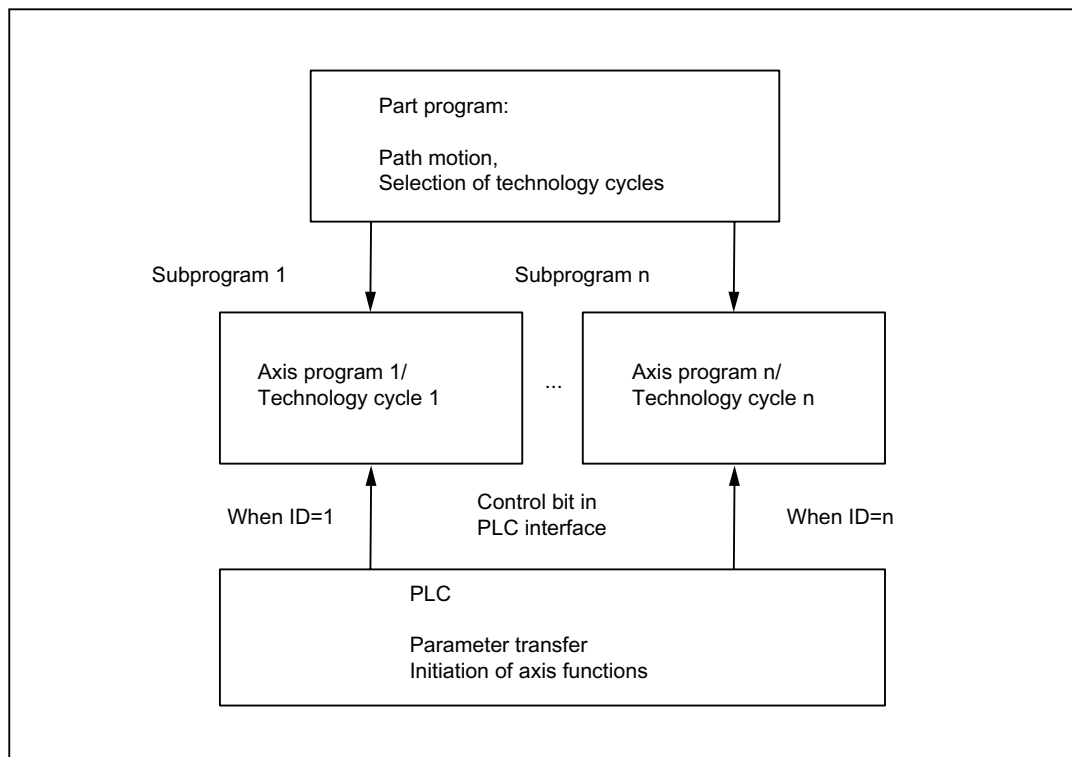


Figure 2-10 Axis programs/technology cycles

### 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

## 2.8.2 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 number of the 1st ID to be disabled
i
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 number of the first ID to be
k disabled for the channel
MD21240 $MC_PREVENT_SYNACT_LOCK_CHAN[1]= ; l number of the last ID to be
l 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
...

```

## 2.9 Control behavior in specific operating states

### 2.9.1 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

### 2.9.2 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

<b>Synchronized action/ technology cycle</b>	Modal and non-modal active action is aborted, synchronized actions are cancelled	Static (IDS) Active action is aborted, technology cycle is reset
<b>Axis/ positioning spindle</b>	Motion is aborted	Motion is aborted

2.9 Control behavior in specific operating states

<b>Speed-controlled spindle</b>	MD35040 \$MA_SPIND_ACTIVE_AFTER_RESET==TRUE:  Spindle remains active MD35040 \$MA_SPIND_ACTIVE_AFTER_RESET==FALSE:  Spindle stops.	MD35040 \$MA_SPIND_ACTIVE_AFTER_RESET==TRUE:  Spindle remains active MD35040 \$MA_SPIND_ACTIVE_AFTER_RESET==FALSE:  Spindle stops.
<b>Master value coupling</b>	MD20110 \$MC_RESET_MODE_MASK, Bit13 == 1: Master value coupling remains active MD20110 \$MC_RESET_MODE_MASK, Bit13 == 0: Master value coupling is separated	MD20110 \$MC_RESET_MODE_MASK, Bit13 == 1: Master value coupling remains active MD20110 \$MC_RESET_MODE_MASK, Bit13 == 0: Master value coupling is separated
<b>Measuring operations</b>	Measuring operations started from synchronized actions are aborted	Measuring operations started from static synchronized actions are aborted

2.9.3 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

## 2.9.4 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.

## 2.9.5 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.

## 2.9.6 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".

<b>Synchronized action/ technology cycle</b>	Modal and non-modal actions are aborted	Static actions (IDS) remain active
<b>Axis/positioning spindle</b>	M30 is delayed until the axis/spindle is stationary.	Motion continues
<b>Speed-controlled spindle</b>	End of program: MD35040 \$MA_SPIND_ACTIVE_AFTER_RESET== TRUE: Spindle remains active MD35040 \$MA_SPIND_ACTIVE_AFTER_RESET==FALSE: Spindle stops. Spindle remains active when the operation mode is changed	Spindle remains active

<p><b>Master value coupling</b></p>	<p>MD20110 \$MC_RESET_MODE_MASK, Bit13 == 1: Master value coupling remains active MD20110 \$MC_RESET_MODE_MASK, Bit13 == 0: Master value coupling is separated</p>	<p>A coupling started from a static synchronized action remains active</p>
<p><b>Measuring operations</b></p>	<p>Measuring operations started from synchronized actions are aborted</p>	<p>Measuring operations started from static synchronized actions remain active</p>

### 2.9.7 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.

### 2.9.8 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.

### 2.9.9 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`.

### 2.9.10 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 a synchronized action itself triggers an alarm, processing will be interrupted and the subsequent actions in this synchronized action will not be executed. If the synchronized action is modal, it is not processed any further in the next interpolation cycle; i.e. the alarm is output only once. Processing of all other synchronized actions continues as normal.

Alarms generating an interpreter stop only take effect once the precoded blocks have been processed.

If a technology cycle generates an alarm with motion stop, then processing of the relevant cycle ceases.

## 2.10 Configuration

### 2.10.1 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	<b>Text in bold print</b>
Base load for a synchronized action, if the condition is not fulfilled: <b>WHENEVER FALSE DO \$AC_MARKER[0]=0</b>	10 µs	<b>~10 µs</b>
Read variable: <b>WHENEVER \$AA_IM[Y]&gt;10 DO \$AC_MARKER[0]=1</b>	11 µs	<b>~1 µs</b>
Write variable: <b>DO \$R2=1</b>	11-12 µs	<b>~1-2 µs</b>
Read/write setting data: <b>DO \$SSN_SW_CAM_MINUS_POS_TAB_1[0]=20</b>	24 µs	<b>~14 µs</b>
Basic arithmetic operation, e.g. multiplication: <b>DO \$R2=\$R2*2</b>	22 µs	<b>~12 µs</b>
Trigonometric functions (e.g. cos): <b>DO \$R2=COS(\$R2)</b>	23 µs	<b>~13 µs</b>
Start positioning axis movement: <b>WHEN TRUE DO POS[z]=10</b>	83 µs	<b>~73 µs</b>

## 2.11 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:

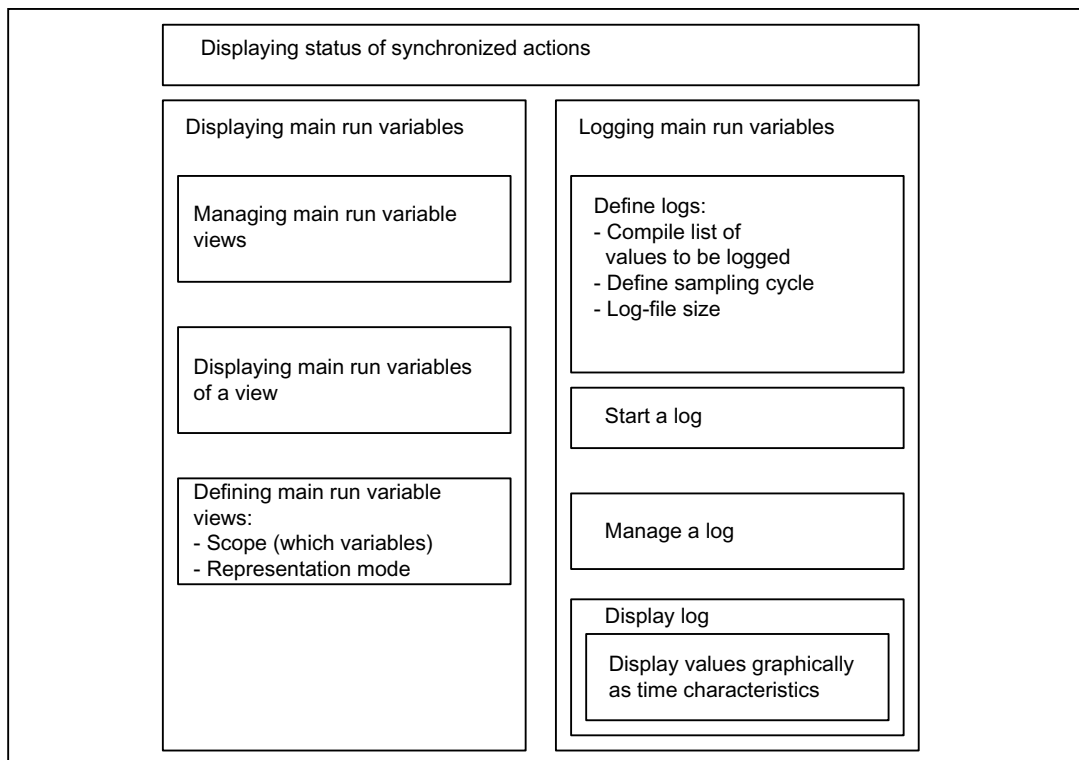


Figure 2-11 Functionality of test tools for synchronized actions

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

**References:**

/BAD/ Operator's Guide HMI Advanced.

## 2.11.1 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	<code>LOCK</code> 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.

## 2.11.2 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.

## 2.11.3 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.

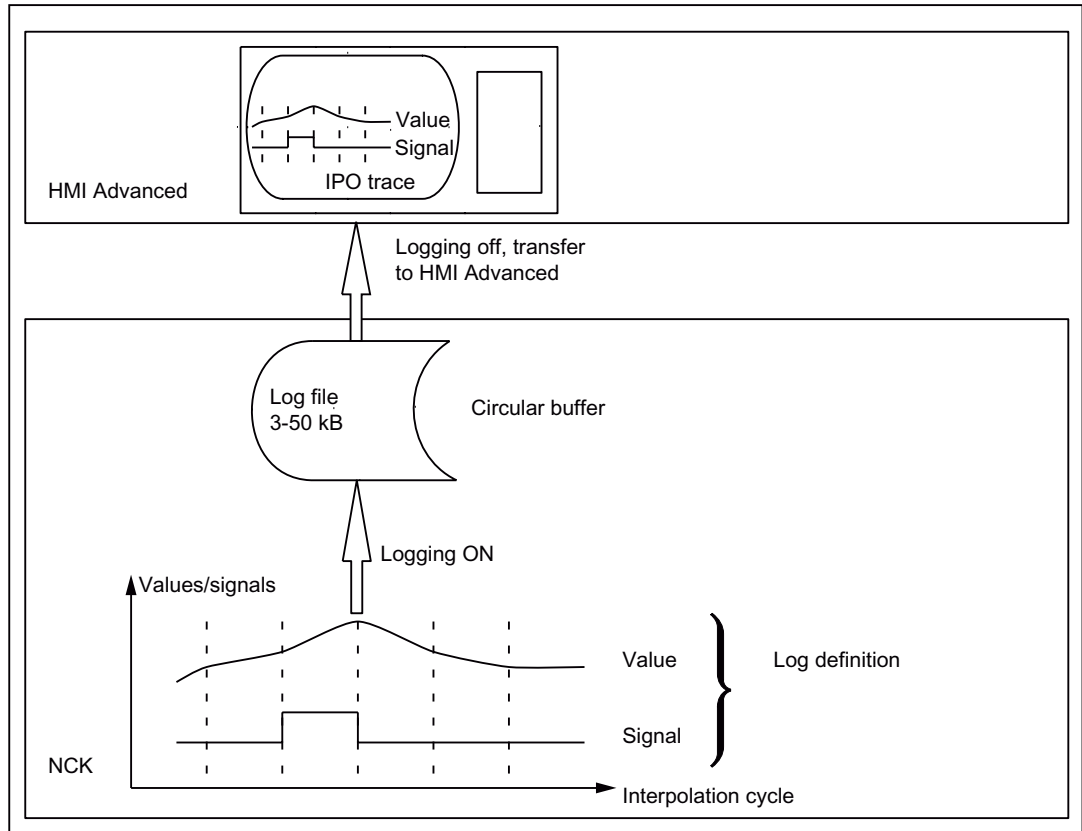


Figure 2-12 Schematic representation of Log main run variables process

**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

### 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:**

Catalog NC60 and NC61

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.

The system data that can be used for a certain software version is described in:

**References:**

Manual of System Variables

Lists (Book 1)

### Extensions

The synchronized actions function package was enhanced continuously in the course of development. The individual extensions are summarized in the following list (with the latest extensions at the end of the list):

- Diagnostic facilities for synchronized actions
- Availability of additional main run variables
- Complex conditions in synchronized actions
  - Basic arithmetic operations
  - Functions
  - Indexing with main run 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
  - Parameterize
- Use of measuring functions from synchronized actions
- SW cams
  - Redefinition of position
  - Redefinition of lead times
- Delete 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
- Synchronized actions, which can be tagged for the PLC
- Availability of additional main run 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
- Exchange via synchronized actions and in technology cycle
- Creation of coupling modules for generic coupling
- Availability of 3-dimensional variables (GUD-, LUD- and system variables)

## Signal Descriptions

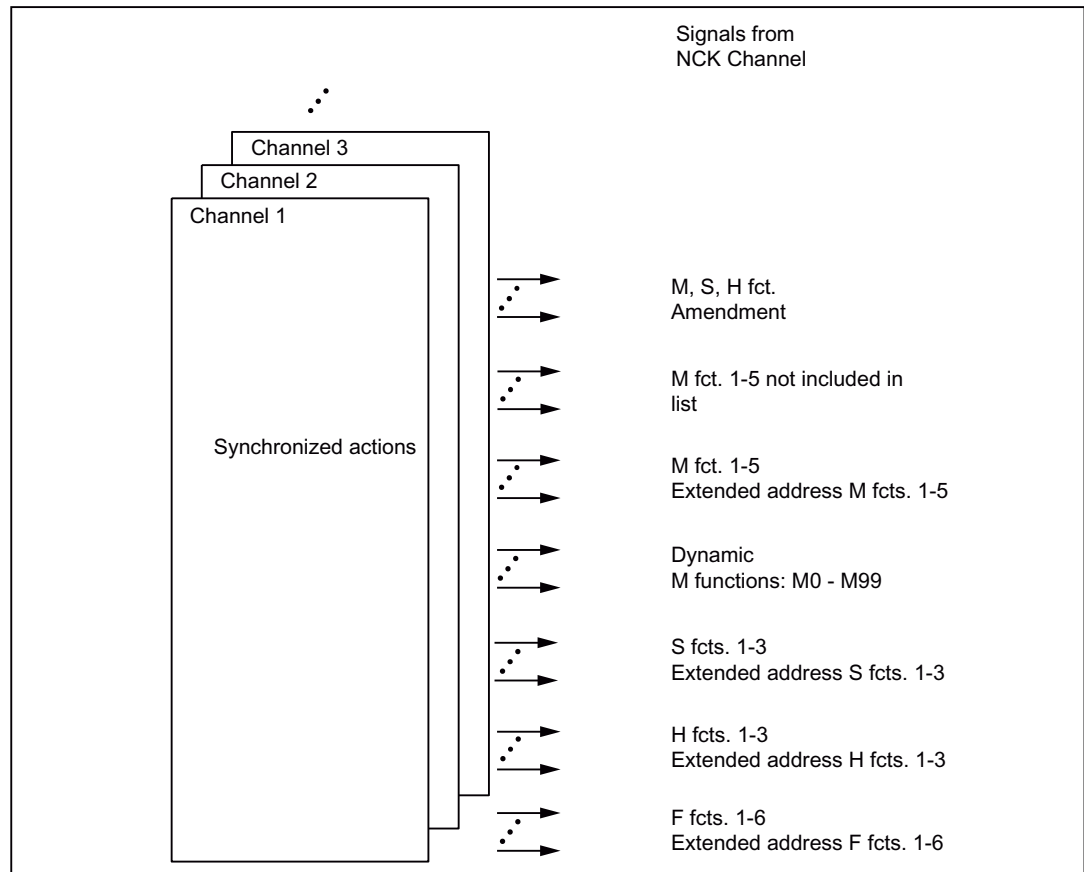


Figure 4-1 NC/PLC interface signals for synchronized actions

For the signals generated by the auxiliary function output from synchronized actions see:

**References:**

Function Manual Basic functions; Auxiliary function outputs to PLC (H2)

### of disabling synchronized actions

With the following signals the PLC application program requests for disabling the assigned synchronized actions:

DB21, ... DBX300.0 (disable synchronized action No. 1)

...

DB21, ... DBX307.7 (disable synchronized action No. 64)

Here DBX300.0 corresponds to the first modal synchronized action (ID=1/IDS=1) and DBX307.7 to 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.

---

**Synchronized actions that can be disabled**

With the following signals the channel shows the PLC application program for the synchronized actions, which may be disabled by PLC:

DB21, ... DBX308.0 (synchronized action No. 1 can be disabled)

...

DB21, ... DBX315.7 (synchronized action No. 64 can be disabled)

Here DBX308.0 corresponds to the first modal synchronized action (ID=1/IDS=1) and DBX315.7 to 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**

The synchronous actions marked in DBB308 to DBB315 as 'can be disabled' and in DBB300 to DBB307 as 'to be disabled' are disabled by the signal:

DB21, ... DBX280.1 (disable synchronized action)

**Synchronized actions disabled**

The NCK confirms the disabling of the requested synchronized actions with the signal:

DB21, ... DBX281.1 (synchronized actions disabled)

## Examples

### 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 ; Equal 0, to ; 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

## 5.2 Reading and writing of SD/MD from synchronized actions

NC language	Comment
	<pre> ; Workpiece coordinate system ; Equal reversal position 1, to ; 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!) </pre>
<pre> N670 ID=7 WHEN \$AA_IM[Z]==\$\$SA_OSCILL_REVERSE_POS1[Z] DO \$AA_OVR[Z]=100 \$AA_OVR[X]=0 </pre>	<pre> Setting data, whose value may change during machining (e.g. through an operator input or synchronized action), must be programmed with <b>\$\$S...</b> <b>Example:</b> Oscillation from synchronized actions with alteration of oscillation position via operator interface </pre>
<pre> N610 ID=1 WHENEVER \$AA_IM[Z]&gt;\$\$SA_OSCILL_REVERSE_POS1[Z] DO \$AC_MARKER[1]=0 </pre>	<pre> ; 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 </pre>
<pre> N620 ID=2 WHENEVER \$AA_IM[Z]&lt;\$\$SA_OSCILL_REVERSE_POS2[Z]-6 DO \$AA_OVR[X]=0 \$AC_MARKER[0]=0 </pre>	<pre> ; Always when the current position of the oscillating axis ; in the machine coordinate system ; Equal reversal position 1, to ; 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!) </pre>
<pre> N630 ID=3 WHENEVER \$AA_IM[Z]==\$\$SA_OSCILL_REVERSE_POS1[Z] DO \$AA_OVR[Z]=0 \$AA_OVR[X]=100 </pre>	<pre> ; Always when the distance-to-go of the part infeed ; Equal 0, to </pre>

Examples

5.2 Reading and writing of SD/MD from synchronized actions

NC language	Comment
<pre> ; Then ; ; ; 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 </pre>	<pre> ; Then    set the axial override of the ;         oscillating axis to 100% (so that the ;         previous synchronized action ;         is cancelled!) </pre>
<pre> If Equal to Then and ; ; ; </pre>	<pre> the current position of the oscillating axis in the Workpiece coordinate system reversal position 1, set the axial override of the Oscillating axis to 100% set the axial override of the infeed axis to 100% (so that the second synchronized action once is cancelled!) </pre>
<pre> N670 ID=7 WHEN \$AA_IM[Z]==\$\$SA_OSCILL_REVERSE_POS1[Z] DO \$AA_OVR[Z]=100 \$AA_OVR[X]=0 </pre>	



```
$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
...
STOPRE ; see following note
ID=1 DO $AC_FCTUL[1]=$A_INA[2]*0.1+0.35 ; Adjust upper limit dynamically via
; analog input 2,
; no condition
ID=2 DO SYNFACT(1, $AA_OFF[V], $A_INA[1]) ; Clearance control by override of no
; condition
...
```

---

**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].

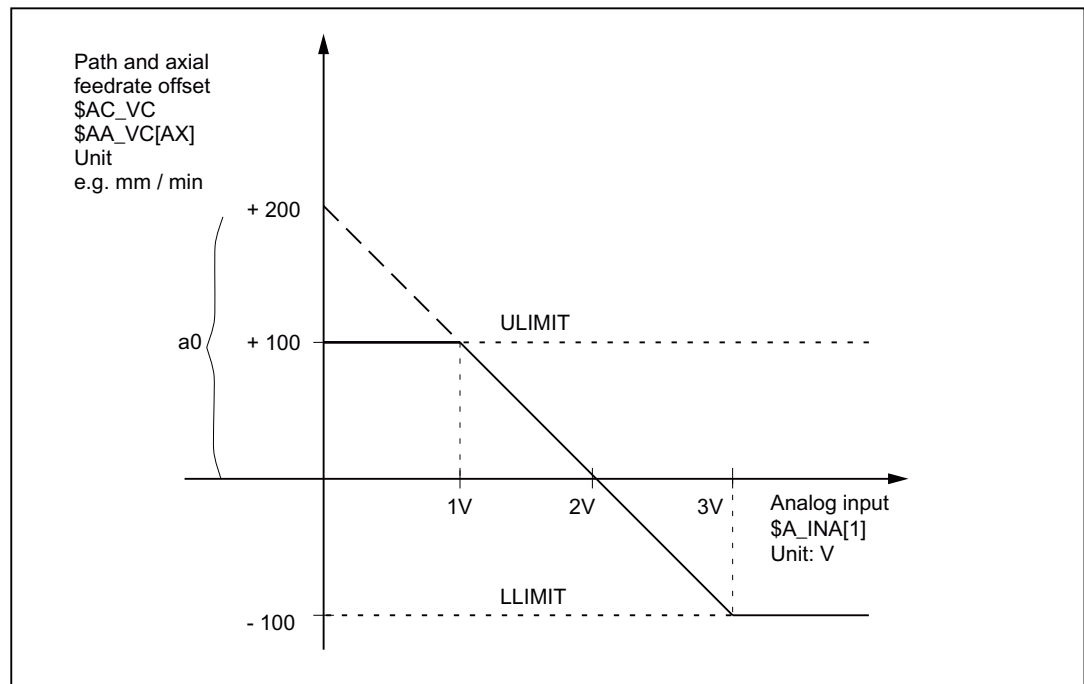


Figure 5-2 Diagram illustrating adaptive control

Determination of coefficients:

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

$$a_1 = -100\text{mm} / (1\text{min} * 1\text{V})$$

$a_1 = -100\%$  regulation constants, pitch

$$a_0 = -(-100) * 2 = 200$$

$a_2 = 0$  (not a square component)

$a_3 = 0$  (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 SYNFACT (1, $AA_VC[X], $A_INA[1])
```

or for the path feedrate:

```
ID = 2 DO SYNFACT(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.

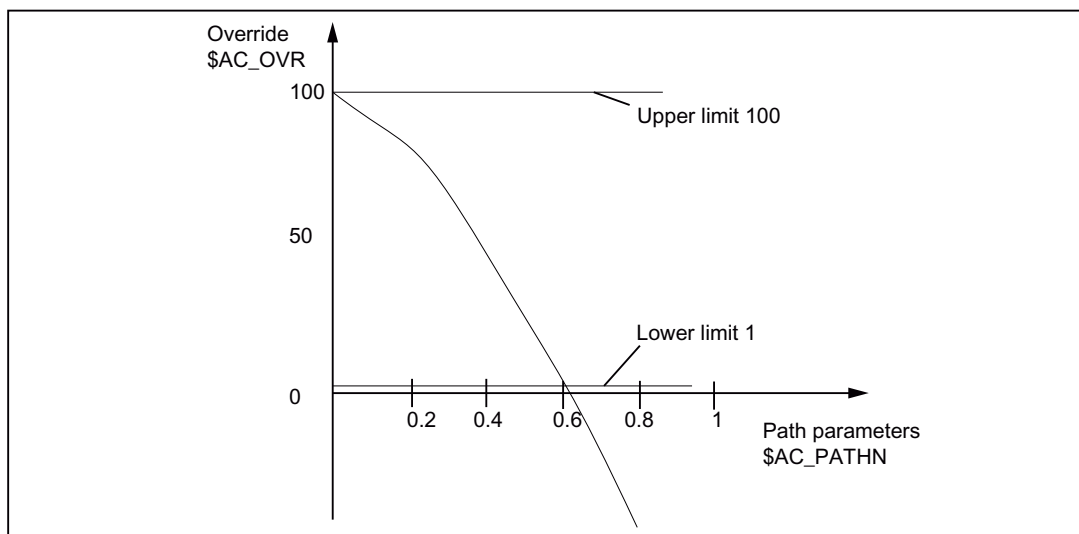


Figure 5-3 Regulate velocity continuously

Polynomial 2:

Lower limit: 1

Hi limit: 100

a<sub>0</sub>: 100

a<sub>1</sub>: -100

a<sub>2</sub>: -100

a<sub>3</sub>: 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 SYNFACT (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
	; as follows <b>without</b> symbolic programming:
IDS=1 EVERY \$AC_TIMEC==0 DO	; Advance R parameter
\$AC_MARKER[0] = \$AC_MARKER[0] + 1	; pointer on block change
IDS=2 DO \$R[10+\$AC_MARKER[0]] =	; Write current time
\$AC_TIME	; of block start in each case to R parameter
	; The example is
	; as follows <b>with</b> symbolic programming:
DEFINE INDEX AS \$AC_MARKER[0]	; Agreements for symbolic
	; programming
IDS=1 EVERY \$AC_TIMEC==0 DO INDEX =	; Advance R parameter
INDEX + 1	; pointer on block change
IDS=2 DO \$R[10+INDEX] = \$AC_TIME	; Write current time
	; 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.

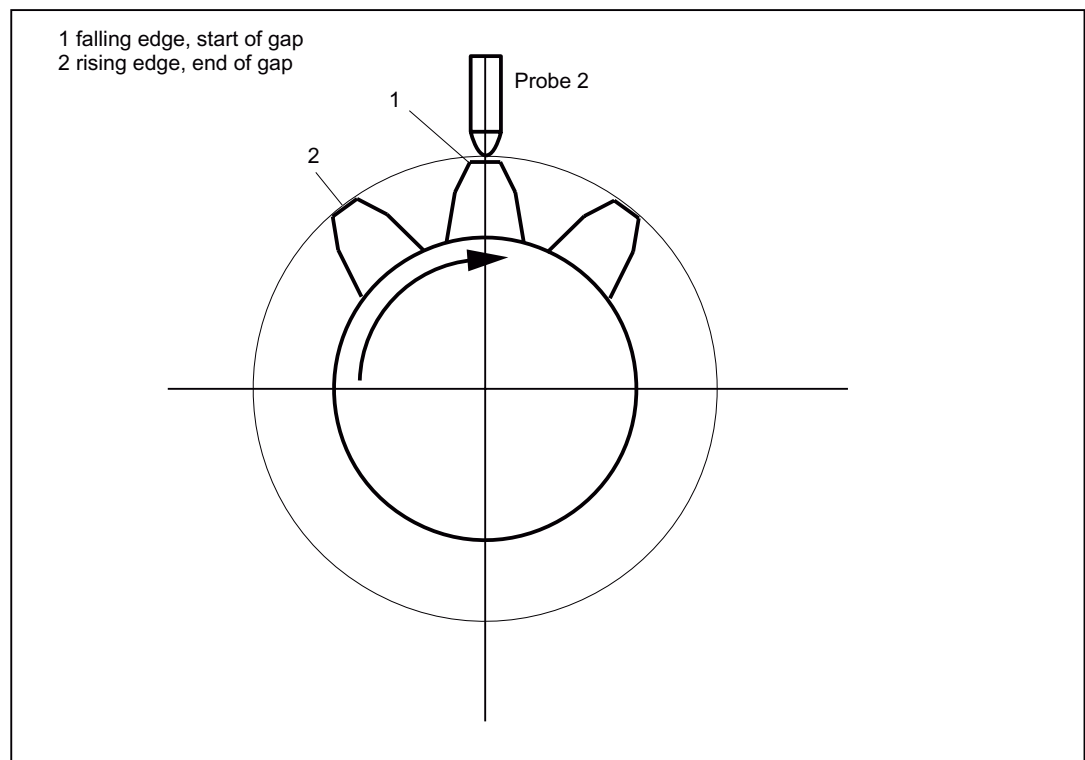


Figure 5-4 Diagrammatic representation of measurement of gaps between gear teeth

```
%_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 gaps  
;*** Input values for ZAHNRADMESSEN ***
```

## 5.6 "Centering" with continuous measurement

```

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           ; Rotate leading axis with feedrate endlessly
MOV[BACH]=1 FA[BACH]=R5           in R5
N275 ID=4 EVERY $R2==0 DO         ; Stop leading axis
MOV[BACH]=0
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.

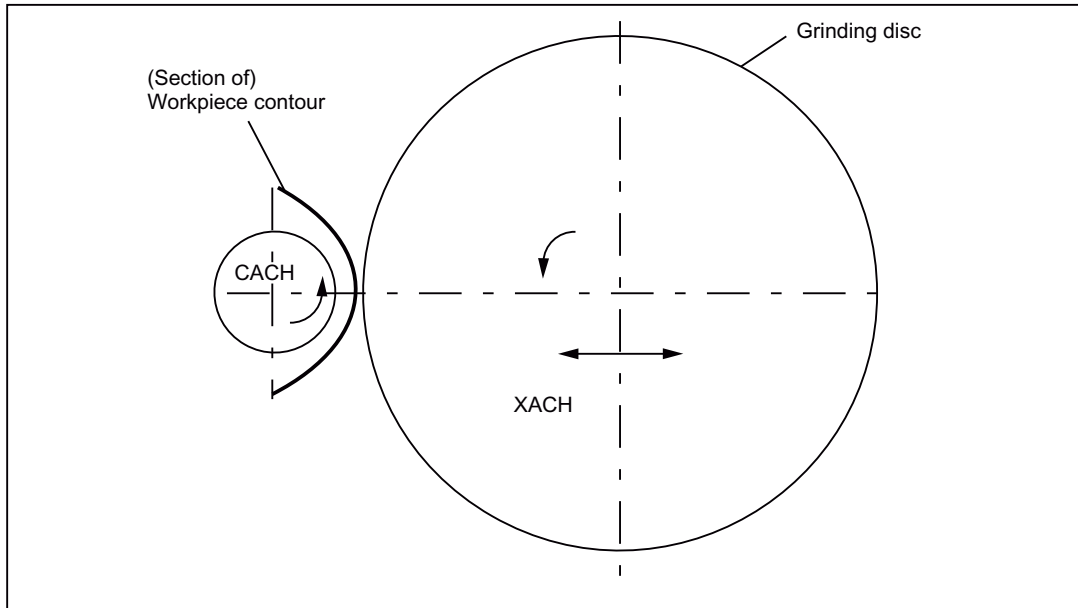


Figure 5-5 Diagrammatic representation of non-circular contour grinding

```

%_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 R9=$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 different 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
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, PRESETON 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

NC program	Comment
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 NULL_POS	; ; when \$A_DBB[0] set by PLC, ; take up basic position
IDS=2 EVERY \$A_DBB[1]==1 DO ZIEL_POS	; when \$A_DBB[1] set by PLC, ; 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_DBW[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.

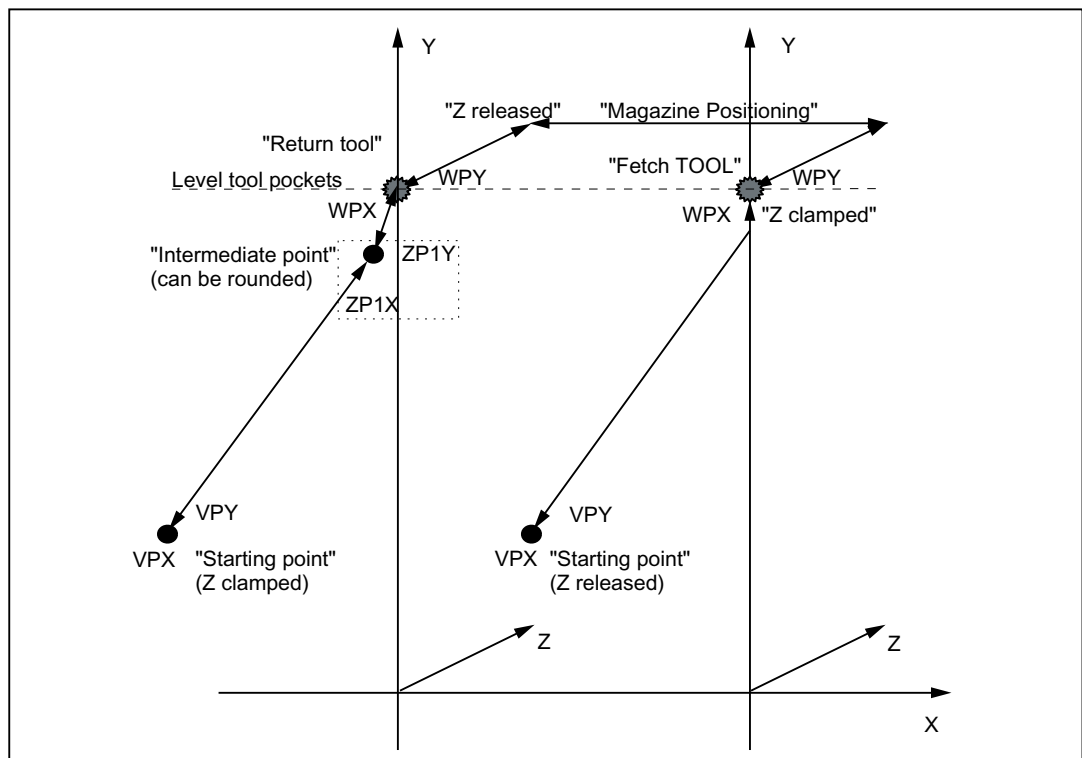


Figure 5-6 Schematic sequence for tool-changing cycle

Flow chart

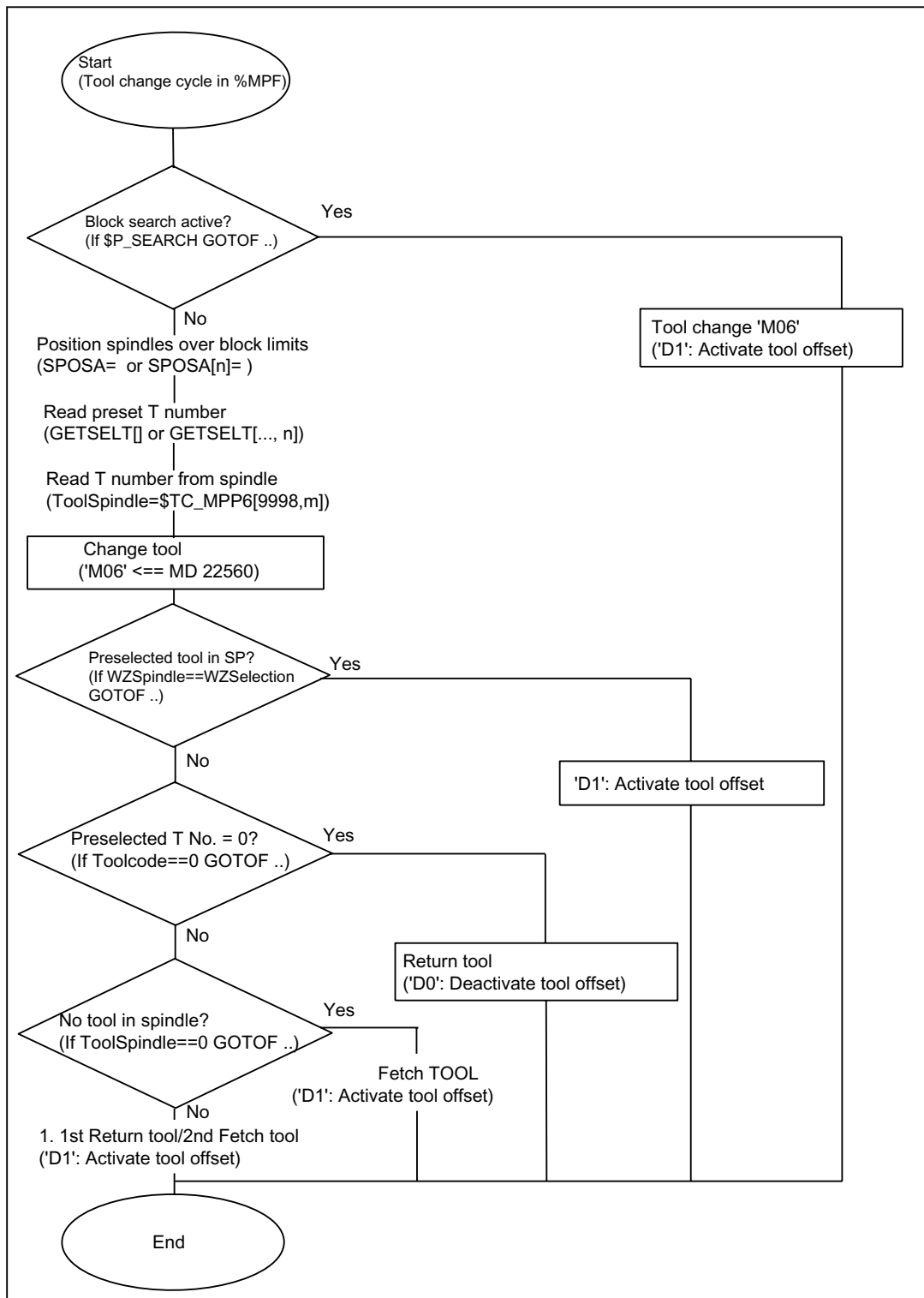


Figure 5-7 Flowchart for tool-changing cycle

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 DO	
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	
<b>*** Fetch and store tool***</b>	
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	; Override=0 when MagAxis ; 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	
<b>*** Store tool***</b>	
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	

## Examples

### 5.9 Synchronized actions in the TC/MC area

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	
<b>;*** Fetch tool***</b>	
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	
<b>;***Tool in spindle***</b>	
wz_in_spindle:	
N275 M=QU(121) D1 M17	
<b>;***Block search***</b>	
wzw_feed:	
N280 STOPRE	
N285 D0	
N290 M06	
N295 D1 M17	

## Data lists

### 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	Memory location of \$AC_PARAM
28256	MM_NUM_AC_MARKER	Number of \$AC_MARKER markers
28257	MM_BUFFERED_AC_MARKER	Memory location of \$AC_MARKER
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. WO (work offset), cont. dressing, clearance control

## 6.2 Setting data

Number	Identifier: \$MA_	Description
32074	FRAME_OR_CORRPOS_NOTALLOWED	Effectiveness of frames and tool length offset
32920	AC_FILTER_TIME	Filter smoothing time constant for Adaptive Control
33060	MAINTENANCE_DATA	Configuration, recording maintenance data
36750	AA_OFF_MODE	Effect of value assignment for axial override with synchronized actions
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.1 Feedback on the documentation

This document will be continuously improved with regard to its quality and ease of use. Please help us with this task by sending your comments and suggestions for improvement via e-mail or fax to:

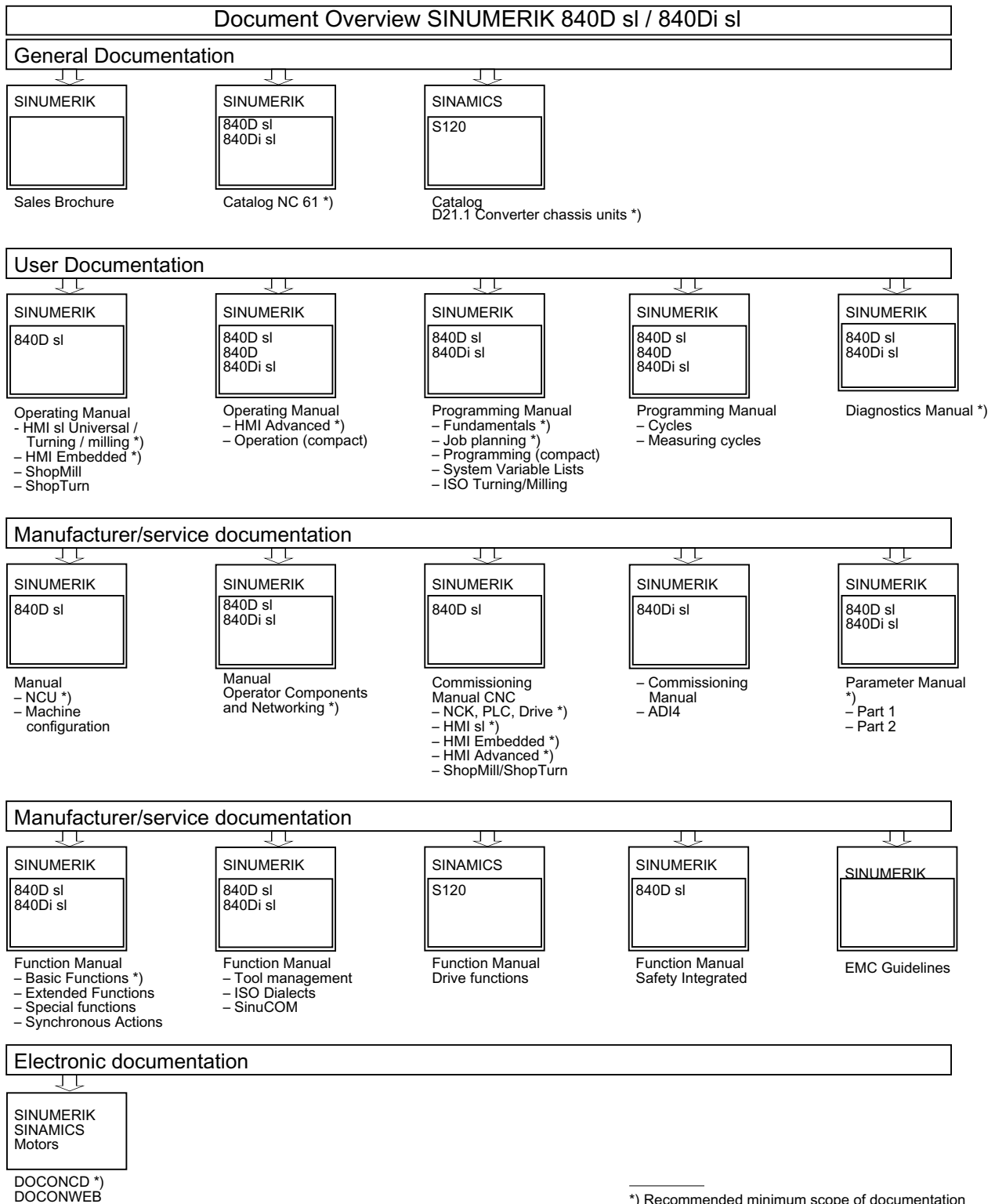
E-mail: <mailto:docu.motioncontrol@siemens.com>

Fax: +49 9131 - 98 2176

Please use the fax form on the back of this page.

<b>To</b> SIEMENS AG I DT MC MS1 P.O. Box 3180  D-91050 Erlangen / Germany  Fax: +49 9131 - 98 2176 (Documentation)	<b>From</b>
	Name:
	Address of your company/department
	Street:
	Zip code:            City:
	Phone:                /
Fax:                    /	
 Suggestions and/or corrections	

## A.2 Overview





# Index

## \$

\$AA\_OFF, 56  
\$AC\_MARKER, 29  
\$AC\_PARAM, 31  
\$AC\_TIMER, 30  
\$R, 32

## A

adaptive control  
    Additive control, 51  
    Multiplicative control, 52  
Adaptive control, 129  
    Example, 131  
Alarm  
    behavior, 113  
Applications, 11  
Area of Application, 14  
Auxiliary function output, 43  
Axial feed, 72  
Axis replacement from synchronized actions, 74  
    AXTOCHAN (axis, channel number)[axis, channel  
    number], 78  
    GET[axis], 74  
    RELEASE[axis], 74

## B

Block search, 112  
Boolean gatings, 17

## C

Calculate slave value, 86  
Calculating master value, 86  
Conditions for execution, 21  
Configurability, 114  
Configuration, 114  
Control system response, 109  
Coordination  
    of technology cycles, 102  
CORROF, 57  
Counter variable, 29  
Coupled motion, 84

Couplings, 84

## D

Data type conversion, 24  
DB 31, ...  
    DBX28.7, 110  
DB21, ...  
    DBX318.2, 63  
    DBX318.3, 63  
DB21, ...  
    DBB308-315, 104  
    DBX.281.1, 105  
    DBX1.2, 104, 124  
    DBX280.1, 105, 124  
    DBX281.1, 124  
    DBX300.0, 104, 105  
    DBX300.0 to 307.7, 123  
    DBX307.7, 104, 105  
    DBX308.0 to 315.7, 124  
DB31, ...  
    DBX28.7, 72  
Definition of synchronized actions, 11  
Detection of synchronism, 87  
Diagnostics data, 116

## E

End of program, 111

## F

FCTDEF, 48  
FIFO variable, 34  
FTOC  
    Online tool offset, 58

## I

Identification number, 14  
Indexing  
    of main run variables, 28

## J

Jerk, 98

- M**
- Machine maintenance, 96
  - Main run variables
    - Special characteristics, 28
  - Main run variables
    - Link, 25
  - Main run variables
    - Write, 46
  - Main run variables
    - Read, 46
  - Main run variables
    - Log, 118
  - Marker variable, 29
  - MD 37200, 87
  - MD 37210, 87
  - MD10722, 75, 76
  - MD11110, 45
  - MD11500, 106, 107, 108
  - MD18860, 97
  - MD20110, 63, 110, 112
  - MD21190, 61, 62, 63
  - MD21194, 60, 62
  - MD21196, 61, 62
  - MD21240, 107, 108
  - MD22200, 45
  - MD22210, 45
  - MD22230, 45
  - MD28050, 93
  - MD28250, 114
  - MD28252, 49, 115
  - MD28254, 31
  - MD28255, 31
  - MD28256, 29
  - MD28257, 29
  - MD28258, 30, 93
  - MD28260, 93
  - MD28262, 93
  - MD28264, 93
  - MD28266, 93
  - MD30552, 78
  - MD32060, 72, 73
  - MD32070, 56
  - MD32300, 81
  - MD33060, 97
  - MD35040, 110, 111
  - MD36750, 53, 54, 56, 129
  - Measurements from synchronized actions, 90
  - Mode change, 111
- N**
- NC STOP, 110
- O**
- Online tool offset, 58, 60
  - Operators
    - Bit-wise logical, 26
    - Boolean, 26
    - Relational, 26
  - Order of execution, 21
  - Overlaid movements, 56
  - Overlaid movements up to SW 5.3, 56
- P**
- Polynomial, 48
  - Polynomial evaluation, 50
  - Power On, 109
  - Preset actual-value memory, 83
  - Program interruption by ASUB, 112
  - Protected synchronized actions, 106
- R**
- R parameters, 32
  - Real time behavior
    - Calculations, 23
    - Expression, 23
  - Real-time variables
    - Advertisements, 118
  - REPOS, 113
  - RESET, 109
- S**
- Scope of performance, 121
  - SD43300, 72
  - SD43350, 53, 129
  - SD43400, 81
  - Set alarm, 96
  - Spindle motions, 79
  - Starting/Stopping axes from synchronized actions, 73
  - Status of synchronized actions, 117
  - Synchronized actions
    - Actions, 18, 41
    - Alter setting data, 47
    - Channel control, 104
    - Components, 13
    - Conditions, 16
    - Control via PLC, 104
    - Disable axis, 67
    - Example Adaptive control, 129
    - Example Conditions, 125
    - Example Control via dyn. override, 132

- Example Path feedrate control, 131
- Example Presses, coupled axes, 138
- Examples: SD/MD, 126
- FIFO variable, 34
- Scanning frequency, 15
- Synchronized Actions
  - Additive adjustment via SYNFACT, 51
  - Multiplicative control via SYNFACT, 52
- Synchronized actions (FBSY)|Data fields, lists, 150
- Synchronized actions (FBSY)|Example, 125
- Synchronized actions (FBSY)|Interface signals, 150
- Synchronous procedure
  - DELDTG, 65
  - RDISABLE, 64
  - STOPREOF, 65
- SYNFACT
  - Examples, 129
  - Polynomial evaluation, 50

## T

- Technology cycle, 19
- Technology cycles, 99
- Time interval, 21
- Timer variable, 30
- TOFFON
  - Online tool length offset, 60
- Total travel count, 98
- Total travel time, 98
  - At high speed, 98
- Total traverse path, 98
  - At high speed, 98

## V

- Variables
  - main run, 23

## W

- Wait markers
  - Delete, 95
  - Setting, 95

## Z

- Zero blocks, 20

