SIEMENS

SINUMERIK 840D sl/
SINUMERIK 840D/840Di/810D

Synchronized Actions

Description of Functions

Valid for

Control	*Software Version*
SINUMERIK 840D sl/840DE sl	1.3
SINUMERIK 840D powerline	7.3
SINUMERIK 840DE powerline (export version)	7.3
SINUMERIK 840Di	2.3
SINUMERIK 840DiE (export version)	2.3
SINUMERIK 810D powerline	7.3
SINUMERIK 810DE powerline (export version)	7.3

08/2005 Edition

SINUMERIK® Documentation

Printing history

Brief details of this edition and previous editions are listed below.

The status of each edition is shown by the code in the “Remarks” column.

Status code in the “Remarks” column:

A . . . . . New documentation.
B . . . . . Unrevised reprint with new order no.
C . . . . . Revised edition with new status.
If factual changes have been made on the page in relation to the same software version, this is indicated by a new edition coding in the header on that page.

06.94	6FC5 297–0AC30–0BP0	**A**
08.94	6FC5 297–0AC30–0BP1	**C**
02.95	6FC5 297–2AC30–0BP0	**C**
04.95	6FC5 297–2AC30–0BP1	**C**
09.95	6FC5 297–3AC30–0BP0	**C**
03.96	6FC5 297–3AC30–0BP1	**C**
08.97	6FC5 297–4AD40–0BP0	**A**[1)]
12.97	6FC5 297–4AD40–0BP1	**C**
12.98	6FC5 297–5AD40–0BP0	**C**
08.99	6FC5 297–5AD40–0BP1	**C**
04.00	6FC5 297–5AD40–0BP2	**C**
10.00	6FC5 297–6AD40–0BP0	**C**
09.01	6FC5 297–6AD40–0BP1	**C**
11.02	6FC5 297–6AD40–0BP2	**C**
03.04	6FC5 297–7AD40–0BP0	**C**
08.05	6FC5 397–5BP10–0BA0	**A**

1) This document replaces the S5 function described for older software versions in the “Description of Functions: Extended Functions” brochure.

Other functions not described in this documentation might be executable in the control. However, no claim can be made regarding the availability of these functions when the equipment is first supplied or for service cases.

Further information is available on the Internet under:
http://www.siemens.com/motioncontrol

We have checked that the contents of this document correspond to the hardware and software described. Nonetheless, differences might exist and therefore we cannot guarantee that they are completely identical. The information contained in this document is, however, reviewed regularly and any necessary changes will be included in the next edition. We welcome suggestions for improvement.

This publication was produced with Interleaf V7.

Subject to changes without prior notice.

Order No. 6FC5 397–5BP10–0BA0
Printed in the Federal Republic of Germany

Siemens Aktiengesellschaft

Foreword

SINUMERIK documentation

The SINUMERIK documentation is subdivided into 3 parts:

- General documentation
- User documentation
- Manufacturer/Service documentation

More detailed information about other SINUMERIK 840D sl/840D/840Di/810D brochures, and brochures for all SINUMERIK controllers (e.g. universal interface, measuring cycles, etc.) can be obtained from your local Siemens representative.

An overview of publications (updated monthly) indicating the language versions available can be found on the Internet at:

http://www.siemens.com/motioncontrol

Follow the menu items"Support" → "Technical Documentation" → "Overview of Documents".

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

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

Target readership of this documentation

This document is designed for machine tool manufacturers. It contains a detailed description of functions from synchronized actions provided in SINUMERIK controls.

Standard version

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

Hotline

Should you have any questions about your controller, please contact the following hotline:

A&D Technical Support Tel.: +49 (180) 5050 222
Fax: +49 (180) 5050–223
E–Mail: mailto:adsupport@siemens.com
Internet: http://www.siemens.com/automation/support–request

If you have any questions about the documentation (suggestions for improvement, corrections), please send a fax to the following number:
Fax: +49 (9131) 98–63315
E–mail: mailto:motioncontrol.docu@siemens.com

Fax form: Refer to the reply form at the end of this manual

Internet address

http://www.siemens.com/motioncontrol

Objective

The Descriptions of Functions provide the information required for configuration and installation.

This documentation describes the function of synchronized actions for SINUMERIK 840D sl from SW 1.3 onwards,
SINUMERIK 840D from SW 4 and for SINUMERIK 810D from SW 2.

The function descriptions are only valid for the specific software version or up to the software version specified. You should request valid function descriptions for new software versions. Old function descriptions are only partly applicable for new software versions.

Target groups

The information contained in the Descriptions of Functions is designed for:

- Design engineers
- PLC programmers creating the PLC user program with the signals listed
- Startup engineers once the system has been configured and set up
- Maintenance personnel inspecting and interpreting status signals and alarms

Layout of manual

This Function Manual is structured as follows:

- General contents
- Descriptions of Functions in alphabetical order in accordance with description of function codes
- Appendix with keyword index

Safety information

This manual contains information which you should observe in order to ensure your own personal safety, as well to avoid material damage. Notices referring to your personal safety are highlighted in the manual by a safety alert symbol; notices referring to property damage only have no safety alert symbol. The warnings appear in descending order of risk as given below.

Danger

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

Warning

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

Caution

with a warning triangle indicates that minor personal injury can result if proper precautions are not taken.

Caution

without a warning triangle means that material damage can occur if the appropriate precautions are not taken.

Notice

indicates that an unwanted result or situation can result if the appropriate advice is not observed.

Intended use

Please note the following:

Warning

The unit may be used only for the applications described in the catalog or the technical description, and only in combination with the equipment, components and devices of other manufacturers where recommended or permitted by Siemens. It is assumed that this product be transported, stored and installed as intended and maintained and operated with care to ensure that the product functions correctly and properly.

Additional notes

!

Important

This notice indicates important facts that must be taken into consideration.

Note

This symbol always appears in this documentation where further, explanatory information is provided.

Technical Information

Notations

The following notations and abbreviations are used in this document:

- PLC interface signals –> IS ”signal name” (signal data)
  E.g.: – IS ”MMC–CPU1 ready” (DB10, DBX108.2), i.e. the signal is stored in data block 10, data byte 108, bit 2.
  – IS ”Feedrate/Spindle speed override” (DB31–48, DBB0), i.e. the signals for each axis/spindle are stored in data blocks 31 to 48, data block byte 0.
- Machine data –> MD: MD_NAME (German name)
- Setting data –> SD: SD_NAME (German name)
- The symbol ” ~~=~~ ” means ”corresponds to”.
- NEW_CONF (cf) – Reconfiguration of the PLC interface
  – ”RESET” key on control unit, or
- RESET (re) ”RESET” key on control unit or
- Immediately (im) after the value has been entered

Data types

The following data types are used in the control:

- DOUBLE
  Real values or integers
  input limits from $+/-4.19*10^{-307}$ to $+/-1.67*10^{308}$
- DWORD
  Integers
  input limits from $-2.147*10^{9}$ to $+2.147*10^{9}$
- BOOLEAN
  Possible input values: true or false/0 or 1
- BYTE
  Integers from –128 to +127
- STRING
  Comprising a max. of 16 ASCII characters (upper case letters, numbers and underscore)

Quantity framework

The explanations of the PLC interface in the individual Descriptions of Functions assume a theoretical maximum number of components:

- Mode groups (associated signals stored in DB11)
- Channels (corresponding signals stored in DB21, ...)
- Axes (corresponding signals stored in DB31, ...)

For details of the actual number of components which can be implemented with each software version, please refer to
References: /BU/, "Order Document", Catalog NC 60

Space for your notes

Table of Contents

Space for your notes

1 Brief Description

Definition 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 synchronism with part program execution. If the condition programmed in the synchronized action is fulfilled or if none is specified, then actions assigned to the instruction are activated in synchronism with the remainder of the part program run.

Applications

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

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

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

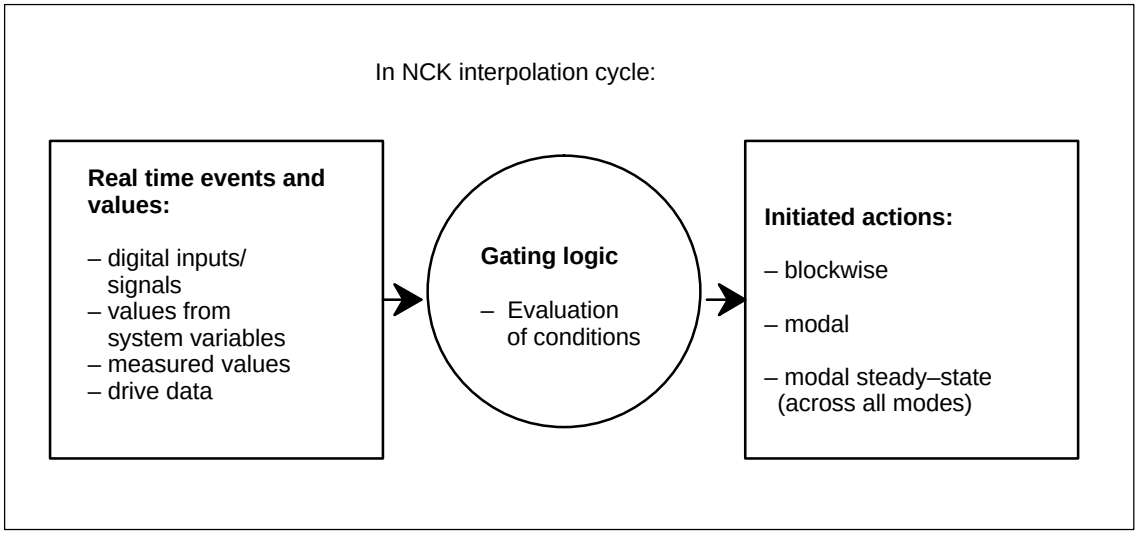


Fig. 1-1 Schematic diagram of synchronized actions

For details of how to program synchronized actions, please see

Reference: /PGA/, Programming Manual Advanced

The following chapters describe:

- Functional conditions for synchronized actions (Chapter 2),
- The required machine data (Chapter 4),
- Example applications (Chapter 6).

Note

This Description of Functions applies to the functionality provided in SW 5.
The functions of synchronized actions in SW versions up to and including 3 are described in:
Reference: /FB/, S5, ”Synchronized actions”

---

2 Detailed Description

2.1 Components of synchronized actions

Structure of a synchronized action

Component:	Validity, identification number	Frequency	G code for cond. and action	Condition	Action code word (fixed)	G code for action	Action or technology cycle refer to 2.5
Example:	IDS=1	EVERY	G70	$AAA_IM[B] > 15	DO	G71	POS[X]=100

The components of a synchronized action, i.e.:

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

are explained individually below.

Validity, ID number

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

- No status
- ID
- IDS

No data

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

Non–modal synchronized actions are operative only in **AUTOMATIC** mode.

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

ID

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

Limitation:

– ID actions remain operative only until another synchronized action with the same identification number is programmed

– Until they are canceled with CANCEL(i), see Subsection 2.5.1.

IDS

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

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

Identification numbers

For modal synchronized actions (ID, IDS) identification numbers between 1 and 255 are allocated. They are important for the functions of mutual coordination of synchronized actions. See Subsection 2.5.1. Modal/static synchronized actions with identification numbers between 1 and 64 can be disabled and enabled from the PLC. See Subsection 2.6.1.

Unique identification numbers must be allocated in the channel.

Application for steady–state (static) synchronized actions:
– AC grinding (also active in JOG mode)
– Gating logic for Safety Integrated
– Monitoring functions, reaction to machine states in all operating modes
– Optimization of tool change
– Cyclic machines

Examples:

IDS=1 EVERY $A_IN[1]==1 DO POS[X]=100	All operating modes
ID=2 EVERY $A_IN[1]==0 DO POS[X]=0	AUTOMATIC

Note

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

Frequency

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

Table 2-1 Effect of frequency keywords

Keyword	Scanning frequency
None	If no scanning frequency is programmed, then the action is executed **cyclically** in every interpolation cycle.
WHENEVER	The associated action/technology cycle is executed cyclically in every interpolation cycle provided that **the condition is fulfilled**.
FROM	If the **condition has been fulfilled once**, the action/technology cycle is executed **cyclically** in every interpolation cycle for as long as the synchronized action remains active.
WHEN	As soon as the **condition has been fulfilled**, the action/technology cycle is executed **once**. Once the action has been executed a single time, the condition is no longer checked.
EVERY	The action/technology cycle is activated **once** if the condition if fulfilled. The action/ technology cycle is re–executed if the **condition changes from the false state into the true state**. Contrary to the keyword WHEN, the condition check remains, after the action has been executed, active until the synchronized action has been cleared or has been de–activated.

For details of technology cycles, please refer to 2.5.

Delete

Deselecting (deleting) an active synchronized action from the part program with **CANCEL** has no effect on the active action. Positioning motions are completed as programmed. A modal or statically effective synchronized action can be cleared using the CANCEL command.
If a synchronized action is cleared, while the positioning axis motion activated from it is still active, then the positioning axis motion is completed. A channel stop also cancels the positioning movement from synchronized actions/technology cycles.

G code for condition and action

In **SW 5** and later, G codes can be programmed in synchronized actions. This allows defined settings to exist for the evaluation of the condition and the action/ technology cycle to be executed, independent of the current 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.

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

Note

In **SW 5**, the use of the G codes in synchronized actions is limited to these 4 G codes.

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.

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

Conditions

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

The condition is checked in the interpolation cycle. If a condition is not specified, then the action is cyclically executed in every IPO clock cycle.
Up to software release 3, the comparison of a real–time variable with an expression calculated in advance or the comparison of two real time variables is permissible.
Examples:
WHENEVER **$AA_IM[X] > 10.5*SIN(45**) DO .... or
WHENEVER **$AA_IM[X] > $$AA_IM[X1]** DO ...

From software release 4, it is additionally possible to combine comparison operations with one another using boolean gating (logic) operations Boolean operators in NC language may be used for this purpose:
NOT, AND, **OR**, **XOR, B_OR, B_AND, B_XOR, B_NOT.**

Examples:
WHENEVER ($A_IN[1]==1) **OR** ($A_IN[3]==0) DO ...
; as long as input 1 is present or input 3 is not present ...

Within a condition, two or several real time expressions can be compared with one another.
Comparisons are possible between variables of the **same type** or partial expressions.
Example:
WHEN $AA_IM[X2] <= **$AA_IM[X1] +.5** DO $AA_OVR[X1]=0
; stop, if the safety clearance is exceeded.

The options for applying real–time expressions are described in the ”Calculations in real time” chapter. When conditions are programmed, all the system variables listed in Subsection 2.3.11 can be addressed. In addition:

- Machine data, e.g. $$MN_..., $$MC_..., $$MA_...
- Setting data, e.g. $$SN_..., $$SC_..., $$SA_...

Note

- GUD variables cannot be used
- R parameters are addressed with $R... .
- Setting data and machine data whose value can change during machining, must be programmed with **$$**S._.../**$$**M._...

Further examples of conditions can be found in Section 6.1.

G code for the action

This G code may specify a G code for all actions in the block and technology cycles, which differs from the one set in the condition. If technology cycles are in the action part, the G code remains modally effective even after the completion of the technology cycle for all subsequent actions until the next G mode.
Only one G code of the G code group may be programmed for each action part.

Actions

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

Example: WHEN $AA_IM[Y] >= 35.7 DO **M135 $A_OUT[1]=1**
If the actual value of the Y axis is greater than or equal to 35.7, then M135 is output to the PLC and output 1 set at the same time.

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. Such programs are also referred to as **technology cycles** below. A technology cycle is a sequence of actions that are processed sequentially in the interpolation cycle. See Section 2.5.

Application: Single axis programs, cyclic machines.

Machining process

The blocks of a part program are prepared at the program preprocessing stage, stored and then executed sequentially on the interpolation level (main run). Variables are accessed during block preparation. When using **real time** variables (e.g. actual values), the block preparation is interrupted so that the actual real time values can be provided up to the previous block.
Synchronized actions are transported to the interpolator in the prepared form with the prepared block. The real–time variables used are evaluated in the **interpolation cycle**. Block preparation is not interrupted.

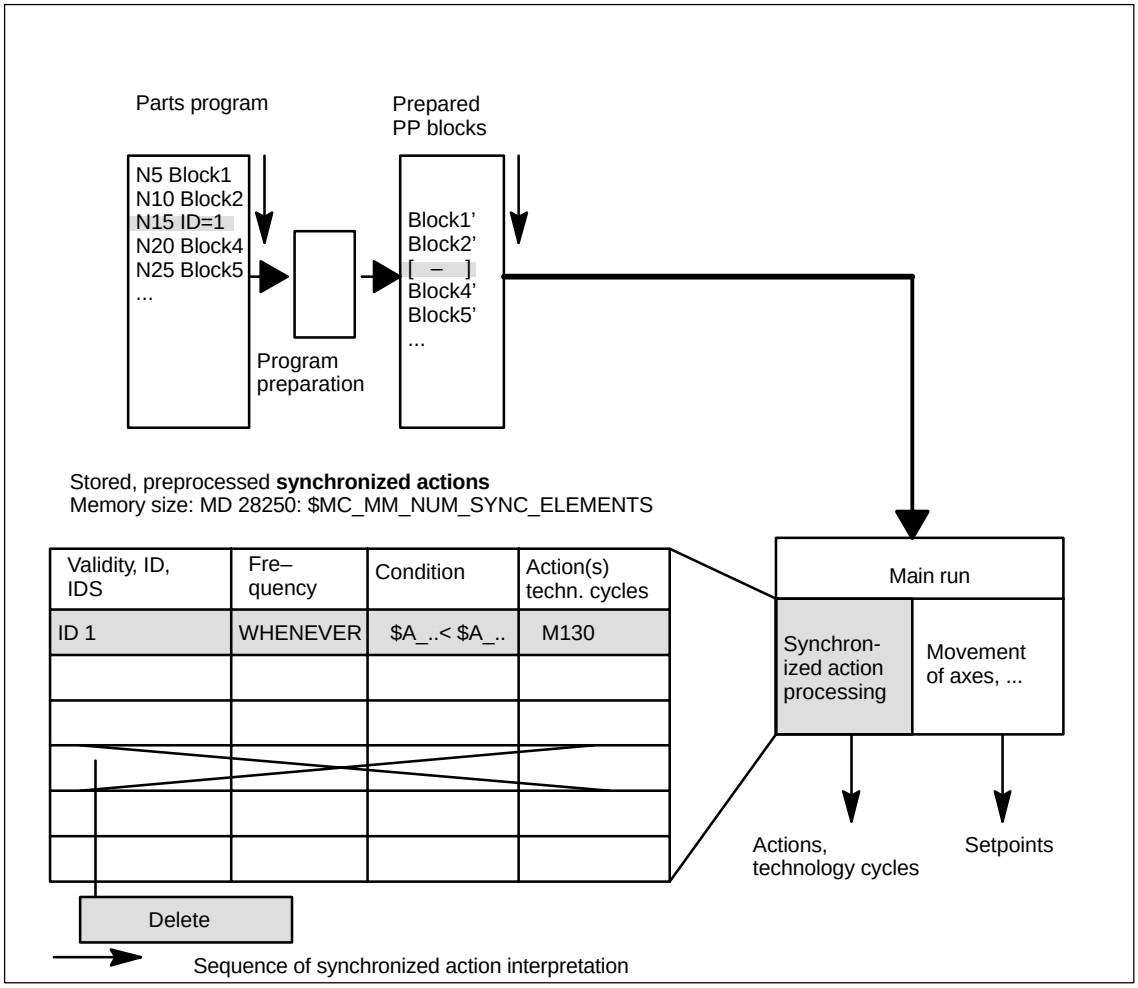


Fig. 2-1 Schematic diagram illustrating processing of synchronized actions

Processing synchronized actions

A check is made in the interpolation clock cycle as to whether actions should be activated in the synchronized actions.
Action(s) is(are) executed in synchronism with the path control if the prerequisites, to the left of the action(s) are fulfilled.

Order of execution

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

2.1.1 Definition of motion–synchronous actions

Defining programs

Motion–synchronous actions can be defined in the following ways:

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

2.1.2 Execution of synchronized actions

Conditions for execution

The actions programmed in motion–synchronous actions are executed if

- The synchronized action exists and has not been deselected with CANCEL(ID) (see Subsection 2.5.1)
- The synchronized action is not disabled, i.e. no LOCK(ID) (see Subsection 2.5.1)
- Evaluation of the action is due as a result of the programmed frequency keyword or
- The appropriate condition is fulfilled

For further details, please see the following subsections.

2.1.3 List of possible actions

- Output of M, S and H auxiliary functions to the PLC
- Real–time variables can be set (written) to obtain the following functionality:
  - Overlaid movement ($AA_OFF), option.
  - Entered tool length offsets ($AA_TOFF), option.
  - Feedrate control ($AC_OVR, $AA_OVR), disabling of a programmed axis motion
  - ...
- Changes to SW cam positions and times (setting data) and alteration of other setting data
- Modification of coefficients and limits from FCTDEF
- SYNFCT (polynomial evaluation)
- FTOC (online tool offsets)
- RDISABLE (read–in disable)
- STOPREOF (preprocessing stop cancellation)
- Delete distance–to–go DELDTG
- Calculation of curve table values
- Axial feedrate from synchronized actions
- Axial frames
- Moving/positioning axes from synchronized actions
- Axis exchange from synchronized actions
- Spindle motions from synchronized actions
- Actual–value setting from synchronized actions (Preset)
- Activation/deactivation of couplings and coupled motion
- Measurements from synchronized actions
- Setting and deletion of wait markers for channel synchronization
- Set alarm/error reactions
- Travel to fixed stop FXS (FXST, FXSW)
- Travel with limited torque FOC (FOCON/FOCOF)
- Extended stopping and retraction (Description of Functions M3)
- Reading and, if tagged accordingly, writing of system variables from the list in Subsection 2.3.11.

These actions are described in detail in Chapter 2.4.

2.2 Real–time evaluations and calculations

Restriction

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

Implicit type conversions, such as in the part program, do not take place. See data type below.

Scope of application

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

Real–time variables

All real–time variables are evaluated (read) in the interpolation cycle and can be written as part of an action.

Real–time variable identifiers

All real–time variables are all variables that start with:
$A... (main run variable) or
$V... (servo values).
So that they can be clearly identified, these variables can be programmed with **$$** in synchronized actions.
e.g. $AA_IM[X] or $$AA_IM[Y]: Actual value for X axis or Y axis in the machine coordinate system.

Note

Setting data and machine data must be programmed with **$$**S.../**$$**M... if their value changes during machining.

Data type

Only real–time variables of the **same data type** may be linked by a logic operation within the same expression. However, in order to process various types of data, you can use the conversion routines provided for type matching (SW 5.2, see conversion routines). In contrast to full expressions in the NC language, calculations are performed in the data type of the real–time variables involved.

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

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

Example 1 for SW 3.2

On the left of the comparison, there is a comparison variable calculated in real time and, on the right, an expression, which is none of the permitted real–time processing variables that begin with $$.

WHEN $AA_IM[X] > **$A_INA**[1] DO M120

M120 is output during execution of the motion programmed in the following block if the X axis actual value exceeds the value applied at analog input 1. With this programming, the actual value is re–evaluated in every interpolation cycle while the value of the analog input is generated at the instant of interpretation.

Example 2 for SW 3.2

On the left of the comparison, there is a comparison variable calculated in real time and, on the right, an expression, which is one of those permitted for the synchronized action (beginning with $$).

WHEN $AA_IM[X] > **$$A_INA**[1] DO M120

The current actual value of the X axis is compared in the IPO cycle with analog input 1 because an $$ variable is programmed on the right.

Both variables are compared to one another in the interpolation cycle.

Example 3 for SW 3.2

$$ variables may also be programmed on the left of the comparison.

WHEN **$$AA_IM[X]** > **$$A_INA[1]** DO M120

Identical to example 2. The left–hand and right–hand sides are always compared in real time.

Extensions in SW 4

The real–time variables available in synchronized actions are listed in Subsection 2.3.11. New system variables, which have been added in subsequent software versions are indicated in the table.

- Machine and setting data

In the case of machine and setting data, $$S... or $$M... must be programmed for online access. The access instruction to be evaluated during interpretation/ decoding must be preceded by a $ sign. Real–time variables that may legally be accessed from synchronized actions are addressed preceded only by a $ sign.

Conversion routines (SW 5.2)

There is no implicit type conversion from REAL to INT and vice versa for synchronized actions. However, the user may explicitly call two conversion routines **RTOI( ) and ITOR( )** for type conversion. The functions can be called

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

ITOR

REAL ITOR(INT) – Converting integer to real

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

Example:

$AC_MARKER[1] = 561
ID=1 WHEN TRUE DO $AC_PARAM[1] = **ITOR**($AC_MARKER[1])

RTOI

INT RTOI(REAL) – Converting from real to integer

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

Note

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

Example RTOI:

```
$AC_PARAM[1] = 561.4378
ID=1 WHEN TRUE DO $AC_MARKER[1] = RTOI( $AC_PARAM[1] )
                                  ; Result: 561

...

$AC_PARAM[1] = –63.867
ID=1 WHEN TRUE DO $AC_MARKER[1] = RTOI( $AC_PARAM[1] )
                                  ; Result:–64

...

$AC_MARKER[1]= 10
$AC_PARAM[1] = –6386798797.29
ID=1 WHEN TRUE DO $AC_MARKER[1] = RTOI( $AC_PARAM[1] )
                  ;Result: Alarm 20145

                  ;$AC_MARKER[1] = 10 (unchanged due to alarm)
```

Implicit type conversion (SW 6.4)

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

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

Examples:
previously

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

SW 6.4 and higher

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

Previously

```
$AC_PARAM[1] = 561.4378
ID=1 WHEN TRUE DO $AC_MARKER[1] = RTOI( $AC_PARAM[1] )  ; 561
```

SW 6.4 and higher

```
$AC_PARAM[1] = 561.4378
ID=1 WHEN TRUE DO $AC_MARKER[1] = $AC_PARAM[1]          : 561
```

Basic arithmetic operations

Real–time variables of the REAL and INT type can be linked logically by the following basic arithmetic operations:

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

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

Expressions

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

Comparisons

The following relational operators may be used:

==	equal to
<>	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

Boolean operators

The following Boolean operators may be used:

NOT	NOT,
AND	AND**,**
OR	OR**,**
XOR	exclusive OR

Bit–serial operators

The following bit operators may be used:

B_OR	Bit–serial OR
B_AND	Bit–serial AND
B_XOR	Bit–serial exclusive OR
B_NOT	Bit–serial negation

Operands are variables and constants of the INT type.

Priority of operators

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

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

and if required, round brackets should be used.
The result of the logic operation of conditions must be a BOOL type.

Example of a multi–element condition:
WHEN ($AA_IM[X] > WERT) AND ($AA_IM[Y] > WERT1) DO ...

Functions

A real–time variable of the REAL type can be used to create function values sine, cosine, etc.
The following functions are possible:

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

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

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

References: /PG/, Programming Manual
/PGA/, Programming Manual Advanced

Indexing

The index of a real–time field variable can in turn be a real–time variable.

Example:
WHEN ... DO $AC_PARAM[**$AC_MARKER[1]**] = 3
The current value of the index $AC_MARKER[1] is evaluated in each interpolation cycle.

Restrictions:

– It is not permissible to nest indices with real–time variables.

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

2.3 Special real–time variables for synchronized actions

A **complete list** of system variables that may be addressed in synchronized actions can be found in Subsection 2.3.11. The characteristics of a few special real–time variables are described below:

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

SW 4

Special real–time variables, i.e. timers, R parameters, machine and setting data and FIFO variables are available in SW 4 and later.

2.3.1 Marker/counter variables

Channel–specific markers

The **$AC_MARKER[n]** variable serves as a marker or counter in the INTEGER data type.

n: Number of marker: 0–n

The number of markers per channel is set via machine data

MD 28256: NUM_AC_MARKER.

The markers are available once per channel under the same name.
The markers are kept in dynamic memories. Markers are set to 0 on POWER ON, NC RESET and End of Program, ensuring identical start conditions for every program run.

Marker variables can be read and written in synchronized actions.

Also available in SW 6.3

In software version 6.3 and later, it is possible to select the memory location for $AC_MARKER[n] between DRAM and SRAM using MD 28257: MM_BUFFERED_AC_MARKER.

0: Dynamic memory DRAM, (default)
1: Static memory SRAM

In MD 28256: NUM_AC_MARKER, 20000 can be entered as maximum value. One element requires 4 bytes. You must ensure that sufficient memory of the correct type is available.

Markers saved in SRAM can be included in the data backup. See 2.3.7

2.3.2 Timers

System variable **$AC_TIMER[n]** permits actions to be started after defined waiting times.

n: Number of timer variable
Unit: Seconds
Data type: REAL

The number of available timer variables is defined in machine data
MD 28258: MM_NUM_AC_TIMER.

Setting timers

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

$AC_TIMER[n]=value
n: Number of timer variable
Value: Start value (normally 0)

Stopping timers

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

$AC_TIMER[n]=–1

Reading timers

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

Example

An actual value is output via analog output 500ms after a digital input has been detected:

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

2.3.3 Synchronized action parameters

$AC_PARAM[n] variables serve as a buffer in synchronized actions.
Data type: REAL
n: Number of parameter 0 – n

The number of available AC parameter variables in each channel is defined via machine data
MD 28254: MM_NUM_AC_PARAM.

These parameters exist once in each channel under the same name. $AC_PARAM parameters are stored in the dynamic memory and reset to 0 on POWER ON, NC RESET and End of Program, ensuring identical start conditions for every part program run. $AC_PARAM variables can be read and written in synchronized actions.

Also available in SW 6.3

In software version 6.3 and later, it is possible to select the memory location for $AC_PARAM[n] between DRAM and SRAM using MD 28255: MM_BUFFERED_AC_PARAM.

0: Dynamic memory DRAM, (default)
1: Static memory SRAM

In MD 28255: NUM_AC_PARAM, 20000 can be entered as maximum value. One element requires 8 bytes. You must ensure that sufficient memory of the correct type is available.

Synchronization parameters saved in SRAM can be included in the data backup. See 2.3.7

2.3.4 R parameters

Definition

R parameters are variables of the REAL time that are stored in battery–backed memory.

For this reason, they retain their settings after end of program, RESET and POWER ON.

Application in synchronized actions

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

Example:
WHEN $AC_MEA== 1 DO **$R10**= $AA_MM[Y]
; if valid measurement available, transfer measured value to R parameter

---

Note

It is advisable to apply a given R variable either normally in the part program or in synchronized actions. If an R variable that has been used in synchronized actions must be later applied ”normally” in the part program, then a STOPRE instruction must be programmed to ensure synchronization. Example:
WHEN $A_IN[1] == 1 DO $R10 = $AA_IM[Y]
G1 X100 F150
STOPRE
IF R10 > 50 .... ; evaluation of R parameter

2.3.5 Machine and setting data

In SW version 4 and later it is possible to read and write machine and setting data from synchronized actions. Access must be programmed according to the following criteria:

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

Reading invariable MD, SD

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 initiated with a **$** character.
Example:
ID=2 WHENEVER $AA_IM[z]< **$**SA_OSCILL_REVERSE_POS2[Z]–6 DO $AA_OVR[X]=0
; Here, reversal range 2, which is assumed to remain static during
; operation, is addressed for oscillation

For a complete example of oscillation with infeed within the reversal range, please see Section 6.2 and:

References: /FB/, P5, Oscillation

Reading variable MD, SD

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

Example:
ID=1 WHENEVER $AA_IM[z]< **$$**SA_OSCILL_REVERSE_POS2[Z]–6 DO $AA_OVR[X]=0
In this situation, it is assumed that the reversal position can be changed at any time by an operator action.

Writing MD, SD

Prerequisite:
The currently set access authorization level must allow write access. It is not meaningful to change MD and SD from synchronized actions unless the change takes immediate effect. The effectiveness of changes is stated individually for all MD and setting data in:

References: /LIS/, Lists

Addressing:
Machine and setting data to be changed must be addressed preceded by the **$$** sign.

Example:

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

2.3.6 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

Fig. 2-3 shows the memory structure of a FIFO variable.

Number

The number of available AC FIFO variables is programmed in machine data
MD 28260: NUM_AC_FIFO.

Size

The number of values that can be stored in a FIFO variable is defined via machine data
MD 28264: LEN_AC_FIFO
All FIFO variables are equal in length.

Data type

Values in FIFO variables are of the REAL data type.

Meaning of index

Index n:

Indices 0 to 5 have special meanings:

n=0: When the variable is written with index 0, a new value is stored in the FIFO.
When it is written with index 0, the oldest element is read and deleted from the FIFO

n=1: Access to oldest stored element

n=2: Access to most recent stored element

n=3: Sum of all FIFO elements
MD 28266: MODE_AC_FIFO defines the mode when generating a sum:
Bit 0 = 1 Sum (total) updated at each write operation
Bit 0 = 0 sum (total) cannot be generated.

n=4: Number of elements available in FIFO.
Every element of the FIFO can be accessed, readng and writing.
FIFO variables are reset by resetting the element number, e.g. for the first FIFO variable: **$AC_FIFO1[4]=0**

n=5 Current write index relative to beginning of FIFO

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

Note

FIFO access is a special form of R parameter access (refer below)
FIFO values are stored in the R parameter area.
FIFO values are stored in the static storage area. They are not deleted by end of program, RESET or POWER ON. FIFO values are stored simultaneously when R parameters are archived.

Machine data
MD 28262: START_AC_FIFO
defines the number of the R parameter, which marks the beginning of FIFO variable storage in the R parameter area.

The current number of R parameters in a channel is programmed in machine data
MD 28050: MM_NUM_R_PARAM

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

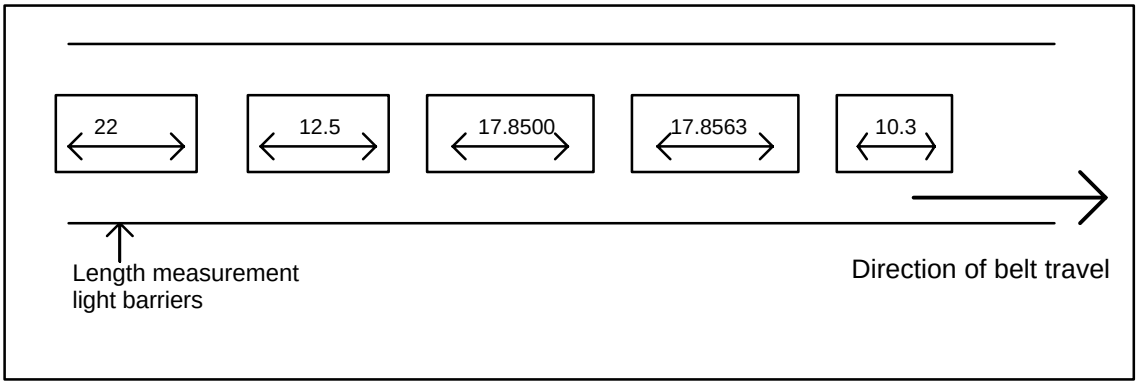


Fig. 2-2 Product lengths of sequence of parts on conveyor belt

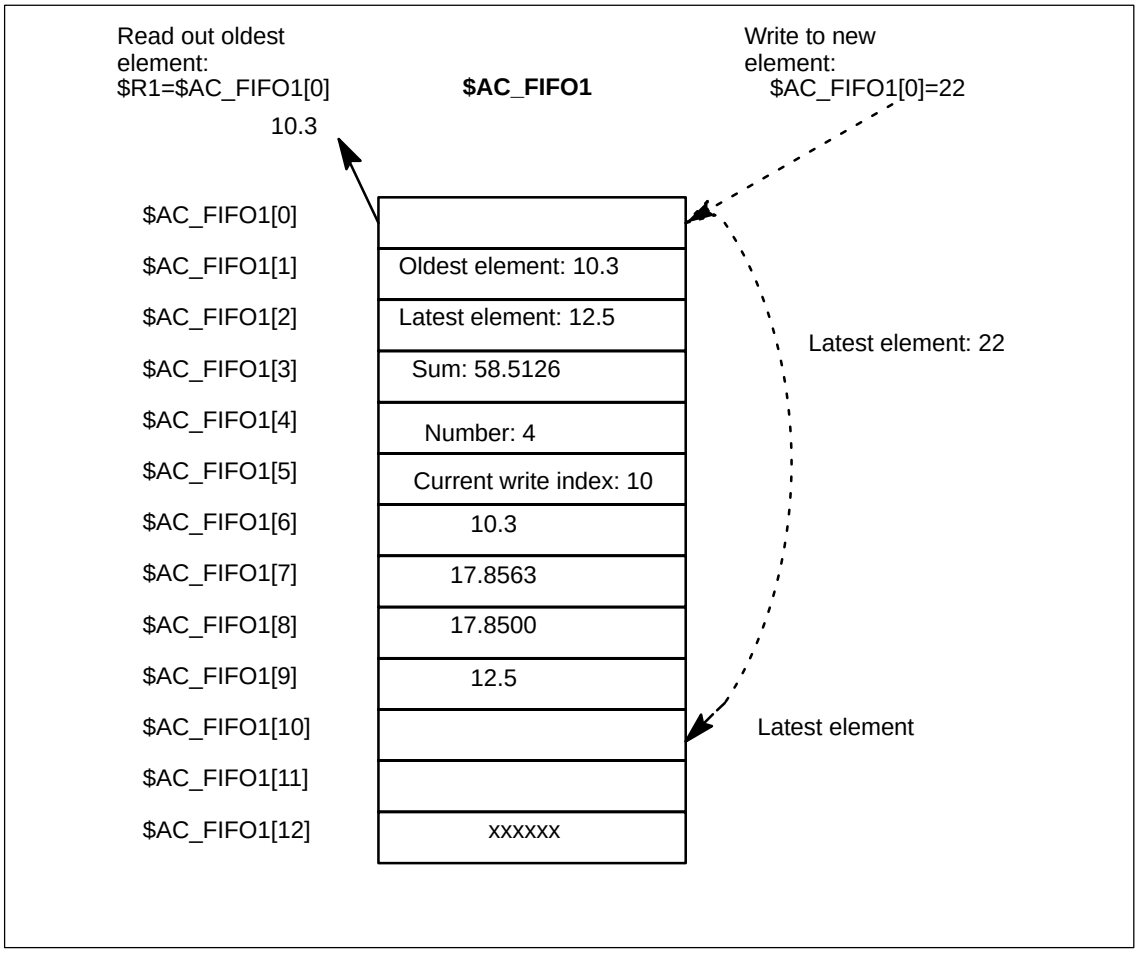


Fig. 2-3 Example of FIFO variables

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

RESET response

System variables $AC_MARKER and $AC_PARAM saved in SRAM retain their values after RESET and POWER ON.

Note

In the case of part programs and synchronized actions that used 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

2.3.8 Determining the path tangent in synchronized actions

$AC_TANEB

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

The tangent angle is always output as a positive value between 0.0 and 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 determine whether the block is a programmed block (main block).

Programming example:

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

2.3.9 Determining the current override

Current override

The current override (NC component) can be read and written to using the system variables:
$AA_OVR axial override
$AC_OVR path override
in synchronized actions.

PLC override

The override specified from the PLC is provided for reading – for synchronized actions in the system variable variable:
$AA_PLC_OVR axial override
$AC_PLC_OVR path override.

Resulting override

The resulting override is, for synchronized actions, provided for reading in the system variables:
$AA_TOTAL_OVR axial override
$AC_TOTAL_OVR path override.

The resulting override is calculated as follows:
$AA_OVR * $AA_PLC_OVR or
$AC_OVR * $AC_PLC_OVR

2.3.10 Capacity evaluation using time requirement for synchronized actions

In an interpolation cycle, synchronized actions must be interpreted and movements, etc. calculated by the NC. The system variables described below can be used by synchronized actions to find out information about the current time components of the synchronized actions in the interpolation cycle and about the computing time of the position controller.

The variables only have valid values if machine data $MN_IPO_MAX_LOAD is greater than 0. Otherwise, the variables always return the gross computing time. This is obtained from:

– synchronized action time,
– position control time and the
– remaining IPO computing time

The variables always contain the values of the **previous** IPO cycle.

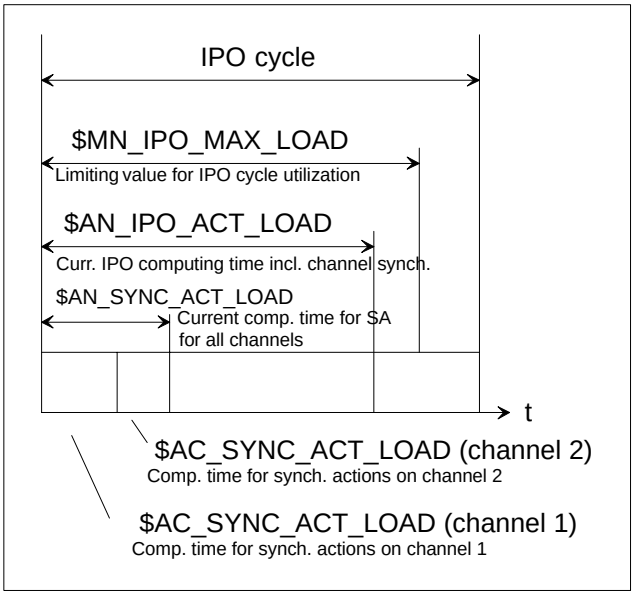


Fig. 2-4 Time components of synch. actions in IPO cycle

System variable	Meaning
$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

System variable	Meaning
$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

The MD 11510: IPO_MAX_LOAD can be used to specify the IPO gross computing time (as a % of the IPO cycle) at and above which the system variable $AN_IPO_LOAD_LIMIT should be set to TRUE.

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

If the MD is 0, then the complete diagnostics function is de–activated.
By evaluating $AN_IPO_LOAD_LIMIT users can define their own strategy in order to avoid level overflow.

System variables that can be written to

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

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

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

Programming example

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

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

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

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

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

2.3.11 List of system variables relevant to synchronized actions

Note

In software version 7.1 and later, the system variables listed previously at this point, which can be accessed by synchronized actions, appear in a separate brochure:

/PGA1/ Lists Manual, System Variables

System variables with the corresponding ID can be used by synchronized actions.

2.4 Actions in synchronized actions

Actions

After action code word **DO ...**, each synchronized action contains:

– One or more (max. 16) actions or a technology cycle
(these two components are referred to generally as **actions** in the following description).

These are executed when the appropriate condition is fulfilled.

Several actions

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

List of possible actions

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

Table 2-2 Actions in synchronized actions

... DO ...	**Meaning**	**Reference**
Mxx Sxx Hxx	Output of auxiliary functions to PLC	2.4.1
SETAL(no.)	Set alarm, error reactions	2.4.22
$A...= ... $V... = ... $AA_OFF = $AC_OVR = $AA_OVR = $AC_VC = $AA_VC = $$SN_SW_CAM_ ...	Writing to real time variables: – Higher level motion – Velocity control: Path velocity Axis velocity add. path feed correction add. correction value of the axis Changing SW cam positions	2.4.2 2.4.3
$AC_FCT... $AA_TOFF =	(setting data) and all other SD Overwriting FCTDEF parameters – Input tool length offsets	2.4.4 2.4.8
RDISABLE STOPREOF DELDTG FTOC SYNFCT ZYKL_T1 (e.g.)	Synchronized action procedures: Activate read–in disable End preprocessing stop Delete distance–to–go Online tool offset Polynomial evaluation Call of technology cycles	 2.4.9 2.4.10 2.4.11 2.4.7 2.4.5 2.5
$AA_OVR[x]= 0 ACHSE_X (e.g.) POS[u]= ... FA[u]= ... MOV[u]= >0 MOV[u] = <0 MOV[u] = =0 AXTOCHAN GET(axis) RELEASE(axis)	Controlling positioning axes: Inhibiting axis motion Calling an axis program Positioning Defining axis feed Command – continually move axes: – forwards – backwards – stop Axis exchange from a synchronized action Request axis for axis exchange Enable axis for axis exchange	 2.4.12 2.4.13 2.4.13 2.4.14 2.4.15 ” ” ” 2.4.16 ” ”
SPOS M3, M4, M5, S = $AA_OVR[S1]= 0	Spindles: Positioning Direction of rotation, stop, speed Disable spindle motion	 2.4.17

Table 2-2 Actions in synchronized actions

... DO ...	**Meaning**	**Reference**
PRESETON(,)	Preset actual value memory	2.4.18
LEADON LEADOF TRAILON TRAILOF	Activating/de–activating couplings: Couple following axis to master axis Disconnect coupling Asynchr. coupled motion on Asynchr. coupled motion off	2.4.19
MEAWA, MEAC	Measurement without deletion of distance–to–go Cyclic measurement	2.4.20
SETM CLEARM	Channel synchronization: Setting a wait marker Deleting a wait marker	2.4.21
LOCK UNLOCK RESET	Coordination between synchronized actions: – synchronized actions/technology cycle inhibit – synchronized actions/technology cycle enable – technology cycle reset	2.5.1

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

For general information about auxiliary function outputs, please see:

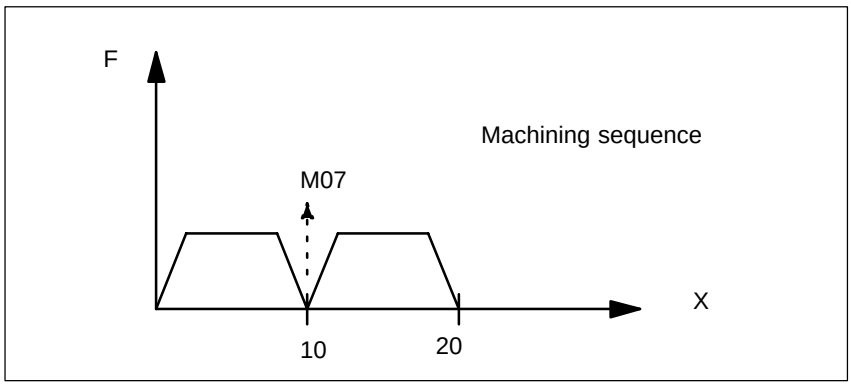
References: /FB/, H2, ”Output of Auxiliary Functions to PLC”

Examples

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

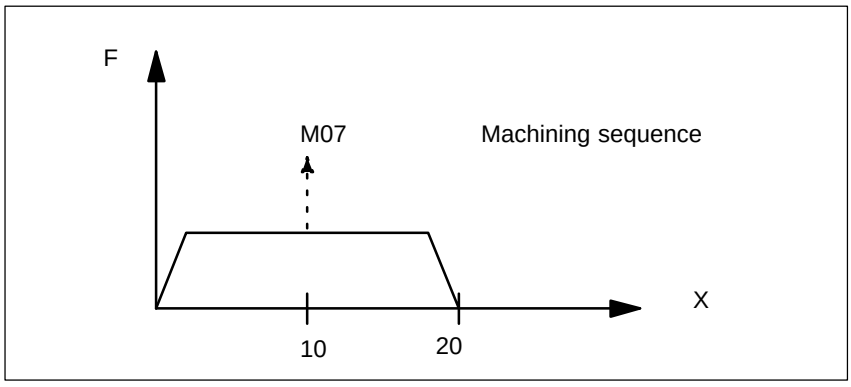
Solution **without** synchronized action: 3 blocks

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



Solution **with** synchronized action: 1 set

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



Auxiliary function output to the PLC

M, S or H auxiliary functions can be output to the PLC as synchronized actions. The output takes place **immediately** (like an interrupt on the PLC) in the interpolation cycle if the condition is fulfilled.
The timing that might be programmed in
MD 11110: AUXFU_GROUP_SPEC (auxiliary function group specification) or
AUXFU_M_SYNC_TYPE (output timing of M functions)/
AUXFU_S_SYNC_TYPE (output timing of S functions)/
AUXFU_H_SYNC_TYPE (output timing of H functions)/
has no effect.

Programming

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

Example

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

Restrictions

No more than 10 auxiliary functions may be output simultaneously (i.e. in an OB 40 cycle of the PLC). The total of auxiliary function outputs from part programs and synchronized actions may never exceed 10 per channel.
Highest number of auxiliary functions for each synchronized action block or technology cycle block:

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

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

```
WHEN ... DO M0                    ; Alarm
```

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

Acknowledgment

Technology cycle blocks (see Section 2.5) 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.

SW 5

Further methods of acknowledgment have been introduced for SW 5 and later:

- Auxiliary function output without block change delay
  High–speed auxiliary functions (QUICK) first, as a parallel process in the PLC, then auxiliary function output with anticipated acknowledgment.

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

References: /FB/, H2, Output of Auxiliary Functions to PLC for SW 5

2.4.2 Setting (writing) and reading of real–time variables

Write

The real–time variables marked with a + sign for access ”Write from synchronized actions” in the list in Subsection 2.3.11 can be **written** in actions contained in synchronized actions.

- Machine and setting data, e.g. $$MN_..., $$MC_..., $$MA_...
  or $$SN_..., $$SC_..., $$SA_...

Note

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

Activation

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

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

The variables in synchronized actions can be **read–accessed** for assignments to real–time variables, as input quantities for functions and for the purpose of formulating conditions. These variables are indicated by the letter **r** for access ”Read from synchronized actions” in the list in Subsection 2.3.11.

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

Note

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

Examples:

```
WHEN $AC_DTEB < 5 DO ...   ; Read distance from end of block in
                           condition
DO $R5= $A_INA[2]          ; Read value of analog input 2 and
                           assign computing variable
```

2.4.3 Alteration of SW cam positions and times (setting data)

Introduction

The ”Software cams” function allows position–dependent cam signals to be output to the PLC or NCK I/Os.

References: /FB/, N3, Software Cams, Position Switching Signals

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:

```
$$SN_SW_CAM_MINUS_POS_TAB_1[0..7]  ; Positions of minus cams
$$SN_SW_CAM_MINUS_POS_TAB_2[0..7]  ; Positions of minus cams
$$SN_SW_CAM_PLUS_POS_TAB_1[0..7]   ; Positions of plus cams
$$SN_SW_CAM_PLUS_POS_TAB_2[0..7]   ; Positions of plus cams
```

Example 1

Changing a cam position:

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

Derivative action or delay times can be changed using the following setting data:

```
$$SN_SW_CAM_MINUS_TIME_TAB_1[0..7] ; Derivative action or delay time
                                                    at the minus cam
$$SN_SW_CAM_MINUS_TIME_TAB_2[0..7] ; Derivative action or delay time
                                                    at the minus cams
$$SN_SW_CAM_PLUS_TIME_TAB_1[0..7] ; Derivative action or delay time
                                                    at the plus cams
$$SN_SW_CAM_PLUS_TIME_TAB_2[0..7] ; Derivative action or delay time
                                                    at the plus cams
```

Example 2

Changing a derivative action/delay time:

```
ID=1 WHEN $AA_IW[x] > 0
                    DO $$SN_SW_CAM_MINUS_TIME_TAB_1[0] = 1.0
```

Note

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

2.4.4 FCTDEF

Application

The Online tool offset FTOC and Polynomial evaluation SYNFCT actions described in the following subsections 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 2.4.7. For examples of load–dependent feedrates and clearance control via polynomials, please see Subsection 2.4.5.

Characteristics of polynomials

Polynomials defined by means of FCTDEF have the following characteristics:

- They are generated through an FCTDEF call in the part program.
- The parameters of defined polynomials are real–time variables.
- Individual polynomial parameters can be overwritten using the same method used to write real–time variables. Permissible generally in part program and in action section of synchronized actions. See Subsection 2.4.2.

Note

In SW 4 and later, it is possible to alter validity limits and coefficients of existing polynomials from synchronized actions. Example: WHEN ... DO $AC_FCT1[1]= 0.5

Number of polynomials

In SW 4 and later, the number of polynomials that can be defined simultaneously can be specified in

MD 28252 : MM_NUM_FCTDEF_ELEMENTS

Block-synchronous polynomial definition

FCTDEF(
Polynomial No.,
Lower limit,
Upper limit,
a0,
a1,
a2,
a3)

The interrelationship between output quantity y and input quantity x is as follows:
$y= a_0+ a_1x+ a_2x^2 + a_3x^3$

The parameters, specified in the function, are saved as follows in the 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
/FB/, W4 ”Grinding”

2.4.5 Polynomial evaluation SYNFCT

Application

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

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

Evaluation function SYNFCT()

The function has the following parameters:

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

For definition of a polynomial, please see Subsection 2.4.4.

Operating principle of SYNFCT

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

Example:
FCTDEF(1,0,100,0,0.8,0,0) ; Polynomial 1, definition made
...
Synchronized action:

ID=1 DO **SYNFCT**(1,$AA_VC[U1], $A_INA[2])
; the additive offset value of axis U1 is computed from the analog input value 2 in every interpolation clock cycle using polynomial 1

For the ’Real–time variable output’, it is possible to select variables that:

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

affect the machining process.

Additive feedrate control

In the 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 examples of ”Real–time variable output” settings:
$AC_VC Additive path feedrate override,
$AA_VC[axis] Additive axial feedrate override

Example of additive control of path feedrate

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

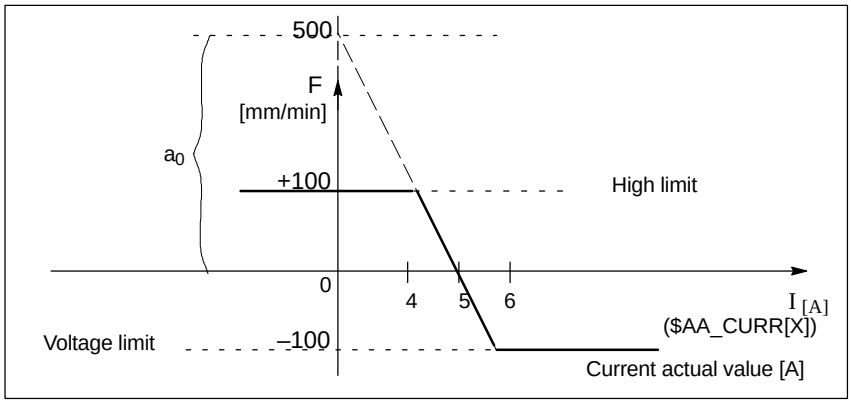


Fig. 2-5 Example of additive control

For definition of coefficients, see also Subsection 2.4.4:

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

$a_1 = -\frac{100\ mm}{1\ min \cdot A}$

$a_1 = -100 \Rightarrow$ control constant

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

$a_2 = 0$ (not a square component)

$a_3 = 0$ (not a cubic component)

Upper limit = 100

Lower limit = –100

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

FCTDEF(1, –100, 100, 500, –100, 0, 0)

The example given in Fig. 2-5 is fully defined with this function.

The *Adaptive Control* is switched–in with the following synchronized action:
ID = 1 DO **SYNFCT**(1, $AC_VC[x], $AA_LOAD[x])
; The additive compensation (override) value for the feedrate of axis x is calculated from the percentage drive load value via polynomial 1 in each interpolation cycle

Multiplicative control

In the case of multiplicative control, the F word is **multiplied** by a factor (override in the 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 real–time variable output.

Example of multiplicative control

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

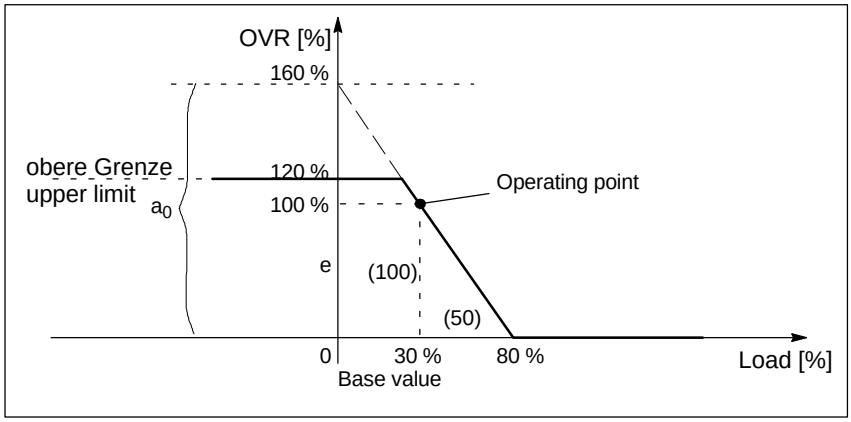


Fig. 2-6 Example of multiplicative control

For definition of coefficients, see also Subsection 2.4.4:

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

$a_1 = -\frac{100\ \%}{(80 - 30)\ \%} = -2$

$a_0 = 100 + (2 \cdot 30) = 160$
$a_2 = 0$ (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)

The example given in Fig. 2-6 is fully defined with this function.

The associated synchronized action can be as follows:
ID = 1 DO **SYNFCT**(2, $AC_OVR, $AA_LOAD[x])
; The path override is calculated from the percentage drive load for the x axis via polynomial 2 in every interpolation cycle.

Position offset with limitation

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

MD 36750: $MA_AA_OFF_MODE

0: Proportional evaluation
1: Integral evaluation

In SW 4 and later, it is possible to limit the value to be compensated absolutely (real–time variable output) to the value stored in setting data

SD 43350 $SA_AA_OFF_LIMIT.

It can be interrogated as to whether the limit is reached by evaluating the axis–specific system variables

$AA_OFF_LIMIT[axis]

in an (additional) synchronized action.

Value –1: Limit of the compensation value was reached in a negative direction.
Value 1: Limit of the compensation value was reached in a positive direction.
Value 0: The compensation value is not in the limit range.

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

Example

Task:
Clearance control as a function of a sensor signal in laser machining operation
The offset value is limited in the negative Z direction to ensure that the laser head is retracted reliably from finished metal blanks. When a limit value is reached, user responses can be initiated such as stopping axes (using override 0, refer to2.4.12) or setting an alarm, refer to 2.4.22.
Limitations/constraints:
Integrating evaluation of the input quantity from the sensor $A_INA[3].
The offset is effective in the basis coordinate system, i.e. before the 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: /FB/, TE1, ”Clearance Control”

References: /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 Subsection 6.3.1 for an example illustrating dynamic adaptation of a polynomial limit in conjunction with Adaptive Control (clearance control). Please see Subsection 6.3.2 for an example of Adaptive Control applied to path feedrate.

Clearance control

The clearance value is applied integrally via MD 36750: AA_OFF_MODE[V]=1. 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).

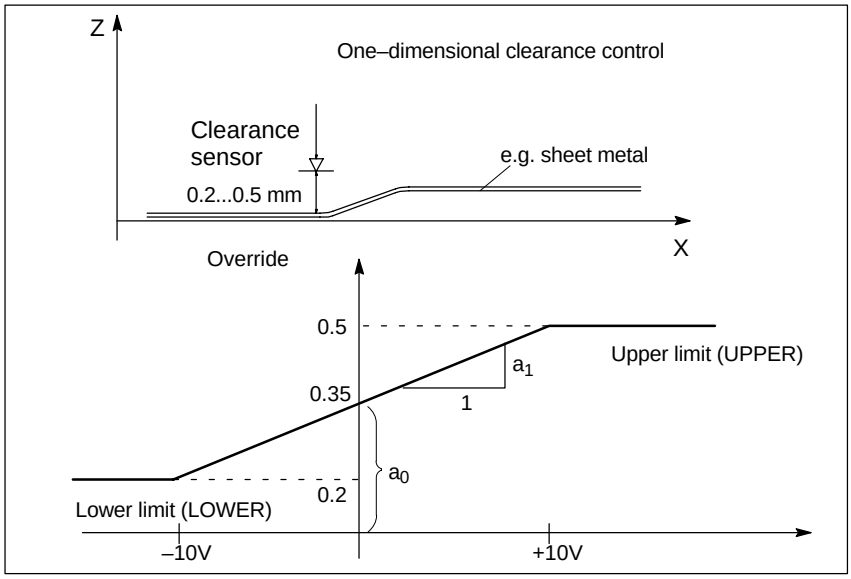


Fig. 2-7 Clearance control

```
%_N_AON_SPF
PROC AON                          ; Subprogram for clearance control on
FCTDEF(1, 0.2,  0.5, 0.35, 1.5 EX–5) ; Polynomial definition: The offset is applied
                                  ;in the range 0.2 to 0.5

ID=1 DO SYNFCT(1,$AA_OFF[Z], $A_INA[3])          ; Clearance control active
ID = 2 WHENEVER $AA_OFF_LIMIT[Z]<>0 DO $AA_OVR[X] = 0
                                  ; Disable if limit range X is
                                  ; exceeded.
RET
ENDPROC


%_N_AOFF_SPF
PROC AOFF                         ; Subprogram for clearance control off
CANCEL(1)                         ; Delete synchronized action for clearance
control
CANCEL(2)                         ; Delete limit range check
RET
ENDPROC


%_N_MAIN_MPF; Main program
                                  ; MD 36750 is set to 1 for
                                  ; integrating processing before power on.
$SA_AA_OFF_LIMIT[Z]= 1            ; Limit value for the offset
AON                               ; Clearance control on
...
G1 X100 F1000
AOFF                              ; Clearance control off
M30
```

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

Overlaid movements up to SW 5.3

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

With axial MD 36750: AA_OFF_MODE, the type of calculation is defined as follows:

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!

Speed limitation with MD 32070: CORR_VELO.

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. As, in the case of static synchronized actions IDS = <number> DO $AA_OFF = <value> this response leads to an immediate renewed overlaid movement with the interpolation of a position offset, machine data MD 36750: AA_OFF_MODE can be used to set the RESET response.

Bit1 = 0: $AA_OFF is deselected in the case of a RESET
Bit1 = 1: $AA_OFF is retained beyond the RESET

In JOG mode, an overlaid movement can take place

Also in JOG mode, if $AA_OFF changes, an interpolation of the position offset can be set as an overlaid movement via machine data MD 36750: AA_OFF_MODE.

Bit2 = 0: No overlaid movement on the basis of $AA_OFF
Bit2 = 1: An overlaid movement on the basis of $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 ASUB and REPOS.

  The coefficients from the call program are applied in the asynchronous subprogram. The coefficients from the asynchronous subprogram continue to be applied in the call program.
- **End of program**
  Polynomial coefficients programmed with FCTDEF remain active after the end of program.
- **Block search**
  During block search with calculation, these polynomial coefficients are collected, i.e. written to the setting data.

CORROF SW 6 and later

- The part program command CORROF with DROF is also collected during a 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/, K1 ”Mode Group, Channel, Program Operation Mode”, Program Testing

- **Axis–specific deselection of DRF offsets with CORROF**
  With CORROF, DRF offsets for individual axes are only possible from the part program.
- **Position offset deselection during active synchronized actions**
  If a synchronized action is active when deselecting the position offset by means of the part program command COROFF(axis,”AA_OFF”), alarm 21660 is signaled. $AA_OFF is deselected simultaneously and not set again. If the synchronized action becomes active later in the block after CORROF, $AA_OFF remains set and a position offset is interpolated.

References: /PG/, ”Programming Manual: Fundamentals”

Note

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

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

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

2.4.7 Online tool offset FTOC

Online tool offset

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

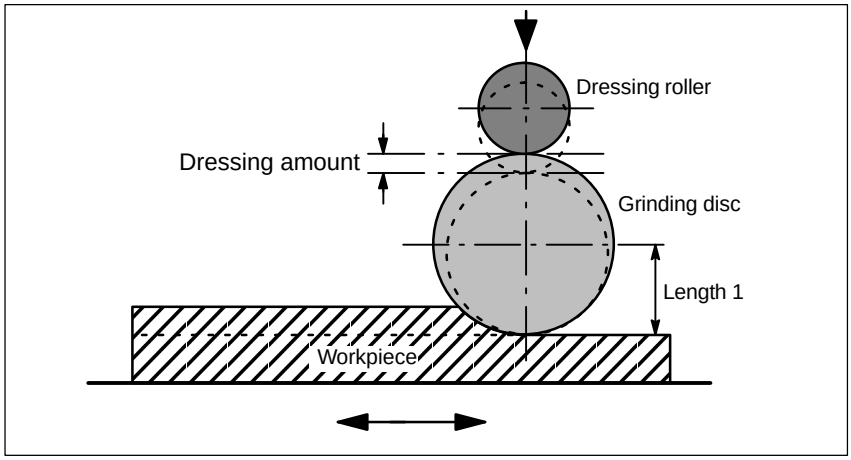


Fig. 2-8 Dressing during machining using a dressing roller

References: /FB/, W4 "Grinding"

Supplementary condition

Synchronized action FTOC is available in SW 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 2.4.4) as a function of a reference value (e.g. actual value of an axis).

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

Programming FTOC

The online offset is specified as follows:

```
FTOC(   Polynomial No.
        Read_real_main_variable     ;Reference value
        Length 1_2_3,
        Channel number
        Spindle number)
```

Parameters

Polynomial No.:	Number of the function previously parameterized with FCTDEF.
Read_real_main_variable:	All of the type REAL main variables listed in 2.3.11 are permissible.
Length 1_2_3:	wear parameter in which the offset value is **added**.

Channel number:	Target channel in which the offset must be applied. This enables simultaneous dressing from a parallel channel. If the channel number is missing, the offset takes effect in the active channel. The online offset with FTOCON must be switched–in in the target channel of the offset.
Spindle number:	The spindle number is programmed if a non–active grinding wheel is 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_ABRICHT_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 corrected in channel 1.
WAITM (1,1,2)                  ; Synchroniz. with the processing channel
G1 V–0.05 F0.01, G91
G1 V –....
...
CANCEL(1)                      ; De–select online offset
...
```

Note

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

2.4.8 Online tool length offset$AA_TOFF[Index]

Function

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

The **velocity** with which the tool length offset is moved through via $AA_TOFF[ ] for the appropriate direction, can be set using machine data MD 21194: TOFF_VELO eingestellt werden.

Correspondingly, using machine data MD 21196: TOFF_ACCEL, the **acceleration** can be set.

Note

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

Additional information to activate the function in the part program, refer to
Reference: /PGA/, 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–in or switching–out a transformation, an overlaid motion must be switched–out using TOFFOF( ).

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

Mode of operation of the offset in the tool axis

The tool length offsets do not change the tool parameters, but are taken into account, within the transformation or the tool carrier that can be orientated, so that offsets are obtained in the tool coordinate systems. The machine data MD 21190: TOFF_MODE can be set using bits 1 to 3 as to whether the contents of the three components of the synchronized action variable $AA_TOFF[ ] are displayed

- Bits 1 to 3 = 0: should be moved to as **absolute value**, or whether
- Bit 1 bis 3 = 1: **an integrating behavior** should be applied.

The integrating behavior of $AA_TOFF[ ] allows a 3D distance 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:
MD 21190: TOFF_MODE = 1 ; absolute values are approached
MD 21194: TOFF_VEL[0] = 10000
MD 21194: TOFF_VEL[1] = 10000
MD 21194: TOFF_VEL[2] = 10000
MD 21196: TOFF_ACC[0] = 1
MD 21196: TOFF_ACC[1] = 1
MD 21196: TOFF_ACC[2] = 1

Activate online tool length offset in the part program
N5 DEF REAL XOFFSET
N10 TRAORI ; activate orientation transformation
N20 TOFFON(Z) ; activate the function for the
; Z tool axis

Interrogate in the synchr. action ; for the Z tool axis, a

N30 WHEN TRUE DO $AA_TOFF[Z] = 10 ; tool length offset = 10
G4 F5 ; interpolated
....

Static synchronized action ; for the X tool axis, an

N50 ID=1 DO $AA_TOFF[X] = $AA_IW[X2] ; offset is applied dependent on the
G4 F5 ; position of axis X2
.... ; actual offset

N100 XOFFSET = $AA_TOFF_VAL[X] ; assign in the X direction
N120 TOFFON(X, –XOFFSET) ; for the X tool axis, the
G4 F5 ; tool length offset is again returned to 0

Example

De–selecting the online tool length offset

N10 TRAORI ; activated orientation transformation
N20 TOFFON(X) activate the function for the
N30 WHEN TRUE DO $AA_TOFF[X] = 10 ; X tool axis, a
G4 F5 ; tool length offset =10 is interpolated
....

N80 TOFFOF(X) ; the position offset of the X tool axis is deleted:
; ... $AA_TOFF[X] = 0 no axis is traversed,
; the position offset is added to the actual
; position in WCS in accordance with the current orientation.
N90 TRAFOOF

Activating and de–activating in the part program

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

During a compensatory movement, the VDI signal NCK → PLC IS ”TOFF motion active” (DB21, ... DBX318.3) is set to 1. After de–selection, the IS ”TOFF active” DB21, ... DBX318.2 is set to 0.

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 machine data MD 21190: TOFF_MODE, the behavior at RESET can be set using bit 0. The tool length offset $AA_TOFF[ ] is, for MD 21190: TOFF_MODE

- Bit 0 = 0 either deselected, or for
- Bit 0 = 1 is kept extending beyond a RESET. The is always, for static

synchronized actions IDS=<number> DO $AA_TOFF[n]= <value> necessary, as otherwise the tool length offset would be immediately received.

Also, using MD 20110: RESET_MODE_MASK, a transformation or a tool carrier that can be oriented, can be de–selected **after** RESET. 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 issued.

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:

- Disable transformation TROFOOF
  A tool length offset must be deleted with TOFFOF( ) **before** a transformation in the part program.
- Change over CP to PTP and move PTP in the JOG mode
  When changing over from CP to PTP, a transformation is switched–out. 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.
- Geometrical axis exchange and level change
  For a geometrical axis exchange, a tool length offset must first be cleared using TOFFOF( ) in the direction of the geometrical axis.
  If, when changing a level, a tool length offset is present, then this must first be cleared using TOFFOF( ).
- Block search
  The instructions TOFFON( ) and TOFFOF( ) are not collected and output in an action block during block search.

2.4.9 RDISABLE

Programmed read–in disable RDISABLE

An RDISABLE command in the active section causes block processing to be stopped if 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.
Application: This method can be used, for example, 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      ;at the end of N10, RDISABLE is effective if, during its
                   processing, the condition is fulfilled
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).
Use of this solution, e.g.: Read–in disable until obstruction is removed from path.

2.4.10 STOPREOF

End of preprocessing stop 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 branch at end of 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 drops below 5V, jump forwards to label 1 (assuming that the value 1000 corresponds to 1 V).

2.4.11 DELDTG

Deletion of distance–to–go

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

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

For the path high speed, prepared DDTG

High–speed/prepared deletion of distance–to–go is used in time–critical applications, i.e.:

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

DELDTG

Programmed using the synchronized action **DELDTG.**
The distance to go along the path is in the system variable $AC_DELT after the delete distance to go has been executed. Continuous–path mode is thus interrupted at the end of the block with high–speed deletion of distance–to–go.

Restrictions:
Delete distance to go for the path can only be programmed as synchronized action that is effective block for block.
If tool radius offset is active, then fast delete distance to go cannot be used.

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

Example DELDTG

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

For axes high speed, prepared DDTG

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

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

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

Examples of DELDTG(axis)

```
WHEN $A_INA[2]>8000 DO DELDTG(X1)
                                  ; If the voltage at input 2 exceeds
                                  ; 8 V, delete distance–to–go
                                  ; for axis X1
POS[X1] = 100                     ; Next position
R10 = $AA_DELT[X 1]               ; Apply axial distance–to–go in R10
```

Once the distance–to–go has been deleted, the variable $AA_DELT[axis] will contain the axial distance–to–go.

(assuming that the value 1000 corresponds to 1 V).

2.4.12 Disabling a programmed axis motion

Task

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

Solution

A synchronized action is used to maintain a 0 override until it is time for the axis to be started.

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 asynchronously
                  ; to path machining;
                  ; the enable is set via a digital input
```

Note

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

2.4.13 Starting command axes

Introduction

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

Control from the PLC

Autonomous individual axis operations (SW 6.3 and later)

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

Control via the command axis occurs via the axial VDI interface (PLC→NCK) with the ”PLC controls axis” IS (DB31, ... DBX28.7) == 1

For more information about the precise sequence of operations of the various steps for transferring control of the command axis to the PLC, please refer to:
Reference: /FB/, P2, ”Positioning axes”

Supplementary condition

An axis cannot be moved from the part program and from synchronized actions simultaneously, This is possible one after the other (consecutively).
Delays may occur if an axis has been moved first from a synchronized action and then programmed again in the part program.

Note

MD 30450: IS_CONCURRENT_POS_AX indicates whether the axis is primarily intended as a command axis or for programming by the part program:
0: not a competing axis
1: competing axis (command axis)

Example 1

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

Example 2

An axis motion can be initiated in the form of a technology cycle (see 2.5)
Main program:

```
...
ID=2 EVERY $A_IN[1]==1 DO ACHSE_X
...
```

Axis program:

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

Programming

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

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

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

G70/G71 and G700/G710 can also be programmed directly in synchronized actions in SW 5 and later.

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

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

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

Note

Only **G70, G71, G700, G710** can be programmed in synchronized actions! Refer to Chapter 2.1.

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

References: /PG/ Chapter 3 ”Positional Parameters”

Absolute/ incremental end position

The end position can be programmed either absolutely or incrementally. Depending on whether G90 or G91 is presently active in the active block of the program, the position is traversed to in absolute or incremental terms.
Further, when programming, it can be explicitly defined as to whether the value is programmed in either absolute or incremental terms.

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

Coded positions are settings stored in machine data.

Example 1 fixed value

ID = 1 EVERY $AA_IM[B] > 75 DO **POS[U]=IC(10)**

; if event occurs, advance U axis by 10

Example 2 current value

The distance to be traversed is formed in real time from real time variables:

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

Axial frame

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

Effect

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

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

Example:

```
TRANS X20
IDS= 1 EVERY $A_IN==1 DO POS[X]=40
G1 Y100          ;if the input is set, then X is positioned to 60
...
TRANS X–10
G1 Y10           ; if the input is set, then X is positioned to 30.
```

Suppression

The effect of frames and tool lengths can be suppressed by means of

MD 32074: FRAME_OR_CORRPOS_NOTALLOWED

Suppressing axial frames

Axial frames that travel incrementally to indexing positions have no effect on a command axis. Therefore, in

MD 32074: FRAME_OR_CORRPOS_NOTALLOWED[AX4], bit 9=1 is set

and the command axis is positioned with JOG.

Example:

```
RANS A=0.001
POS[A]=CAC(2)                    ; Axis travels to position 180.001 degrees

; The axial frame has no effect on the command axis
; MD 32074: FRAME_OR_CORRPOS_NOTALLOWED[AX4] = 'H0020'

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

Note

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

2.4.14 Axial feedrate from synchronized actions

Feedrates

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

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

The axial feedrate for command axes is modal. It is programmed under address FA. The default value is set via axial machine data
MD 32060: POS_AX_VELO.

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

Example of calculated feedrate

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

The feed value is either programmed as linear or rotary feed:
The feed type defines the setting data:
SD 43300: $SA_ASSIGN_FEED_PER_REV_SOURCE.
The setting data can be changed by an operator action, from the PLC, or from the part program. In synchronism with the part program context, the feedrate type can be switched over using the NC commands FPRAON, FPRAOF. See also:

References: /FB/, V1 "Feedrates"

Note

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

2.4.15 Starting/Stopping axes from synchronized actions

Starting/stopping

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

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

MOV[axis]=Value

The value is an INT data type.
The sign of the value determines the direction of motion:
> 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 MD 32060: POS_AX_VELO.

Example

... DO **MOV**[U]=0 ; Stop axis motion as soon as condition has been fulfilled.

2.4.16 Axis replacement from synchronized actions

Application

For a tool change, the command axes available for this purpose can be fetched, using synchronized actions, in a specific channel for the axis replacement. If the tool has been changed, these command axes can be released again for the channel from the synchronized action.

Request axis

As action of a synchronized action, using

GET[axis]

an axis can be requested.

Release axis

RELEASE[axis]

an axis can be released for axis replacement.

Note

The axis concerned 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 in time, 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 authorization (rights) for the axis
- the requested axis is already assigned the requested channel
- the axis in the neutral axis status is controlled by the PLC
- the axis is active as command axis or oscillating axis or is assigned from the PLC to traverse
- the axis is already assigned the NC program of the channel.

Note

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

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 rights 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. It then assumes the state

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.

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 **one** reorganization procedure.

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

If the axis is in neutral axis status **controlled by the PLC** $AA_AXCHANGE_TYP[axis]==4), the axis is requested as a neutral axis $AA_AXCHANGE_TYP[axis] == 8. This locks the axis for automatic axis replacement between channels in accordance with the value of bit 0 in machine data MD 10722: AXCHANGE_MASK (bit 0 == 0).

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

Axis is active as command axis/ assigned to the PLC

If the axis is active **as the command axis**/oscillating axis or assigned to the PLC for travel, PLC axis == concurrent positioning axis, $AA_AXCHANGE_TYP[axis]==1, the axis is requested as a neutral axis $AA_AXCHANGE_TYP[axis] == 8. This locks the axis for automatic axis replacement between channels in accordance with the value of bit 0 in MD 10722: AXCHANGE_MASK (bit 0 == 0).

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,

this means that 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 **one** reorganization procedure.

Axis to be released is already a neutral axis:
If the axis is already in the neutral axis state $AA_AXCHANGE_TYP[<axis>] == 3 or is active as command or oscillating axis or assigned from the PLC to traverse,
PLC axis == competing positioning axis, then the
$AA_AXCHANGE_TYP[axis] == 1, then the axis is released for an automatic axis replacement between channels. The status 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 not state change. This 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.

Supplementary conditions for 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 is neutral
    WHENEVER $AA_TYP[Z] == 1 DO RDISABLE             ; read–in inhibit as long as
                                                     ; the Z axis is the program axis
N110 G4 F0.1
    WHEN TRUE DO GET(Z)                              ; Z axis is again
                                                     ; the NC program axis
    WHENEVER $AA_TYP[Z] <> 1 DO RDISABLE             ; read–in inhibit until
                                                     ; the Z axis is the program axis
N120 G4 F0.1
    WHEN TRUE DO RELEASE(Z)                          ; Z axis is neutral
    WHENEVER $AA_TYP[Z] == 1 DO RDISABLE             ; read–in inhibit as long as
                                                     ;the Z axis is the program axis
N130 G4 F0.1
N140 START(2)                                        ; start the 2nd channel
```

Program sequence in the second channel:

```
    WHEN TRUE DO GET(Z)                              ; fetch the Z axis in the 2nd channel (neutral)
    WHENEVER $AA_TYP[Z] == 0 DO RDISABLE             ; read–in inhibit as long as
                                                     ; the Z axis is in the other channel
N210 G4 F0.1
    WHEN TRUE DO GET(Z)                              ; Z axis is the NC program axis
    WHENEVER $AA_TYP[Z] <> 1 DO RDISABLE             ; read inhibit until
                                                     ; the Z axis is the program axis
N220 G4 F0.1
    WHEN TRUE DO RELEASE(Z)                          ; Z axis neutral in the 2nd channel
    WHENEVER $AA_TYP[Z] == 1 DO RDISABLE             ; read inhibit as long as
                                                     ; the Z axis is the 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)                    ; fetch Z axis in this channel
     WHENEVER $AA_TYP[Z] == 0 DO RDISABLE   ; read–in inhibit as long as
                                            ; the Z axis is in another channel
N160 G4 F0.1
N199 WAITE(2)                               ; wait for the end of the program in channel 2
N999 M30
```

Example

GET, RELEASE in the technology cycle

Axis U (MD 30552: AUTO_GET_TYPE = 2 automatically generated GET issues a GETD) is declared in the 1st and 2nd channels and currently channel 1 has the interpolation authorization,
the following technology cycle is started in channel 2:

```
GET(U)          ; fetch axis U in the channel
POS[U]=100      ; axis U should be traversed to position 100
```

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

AXTOCHAN Move axis to another channel

From a synchronized action, using the NC language command

AXTOCHAN(axis, channel number)[axis, channel number[, ...]])

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

If the axis **is already assigned to the NC program** of the requested channel $AA_AXCHANGE_TYP[axis] == 0

then there is **no** state change.

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

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

Constraints AXTOCHAN

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

Traversing using static synchronized actions: Stage 2 is required, refer to
References: /FB2/, P2 ”Positioning axes”

Note

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

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

2.4.17 Spindle motions from synchronized actions

General

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

Starting/stopping

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

Stop until event occurs

Application:
A spindle is programmed within a machining routine, but should not be started at the beginning of the block in particular circumstances. Using a synchronized action, the override is held at 0 up to the starting time.
Example:

```
ID=1 WHENEVER  $A_IN[1]==0 DO $AA_OVR[S1]=0
G01 X100 F1000 M3   S1=1000
                    ; the spindle is started asynchronously to path processing;
                    ; the start is realized 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
```

If there is no numeric extension, each of the commands applies to the master spindle. Every spindle can be activated by specifying a numerical extension:

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

When programming the positioning type, the same rules apply as for positioning axes (refer 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 ; rotational direction and speed
ID = 2 EVERY $A_IN[2]==1 DO M4 S500; rotational direction and speed
ID=3 EVERY $A_IN[3]==1 DO S1000 ; new speed setting
                                  ; for active spindle rotation
ID=4 EVERY ($A_IN[4]==1 ) AND ($A_IN[1]==0)  DO  SPOS=0
                                                 ; position spindle
```

Feed

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

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 by G25/G26 are taken into account as a function of setting data: SD 43400: WORKAREA_PLUS_ENABLE.

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

Correcting acceleration

ACC[axis]=0..200 can be used to change the acceleration preset in MD 32300: MAX_AX_ACCEL specified acceleration is changed in a range from 0% to 200% with:
ACC[axis]=<value> (is effective in the part program and in synchronized actions).

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 the part program
...
N20  ID=1 WHEN $A_IN[1]==1  DO POS[X]=100  FA[X]=200
                                 ; start positioning from synchronized action,
                                 ; if digital input present
...
CANCEL(1)                        ; de–select synchronized action
...


N100 G01 X100 Y200 F1000         ; X: Path axis
                                 ; Delay before movement if digital
                                 ; input was 1 and X was therefore
                                 ; positioned by a synchronized action
```

On–the–fly transitions

Transitions can be made between command axes and spindles.

Initial situation

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

Procedure

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
; Due to the programming of $AC_TIMER[1], the synchronized action with ID=2 is the most recently activated action. Its commands are applied in place of the commands in 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 Transitions marked with x are legal:

from ↓ to →	POS	MOV=1 MOV= – 1	MOV=0	SPOS	M3 M4	M5	LEADON	TRAIL–ON
Axis stationary								
Axis mode	x	x	x	x	x	x	x	x
Position–controlled spindle	x	x	x	x	x	x		
Speed–controlled spindle				x	x	x		
Axis in motion								
Axis mode	x	x	x				x	x
Position–controlled spindle								
Speed–controlled spindle				x	x	x		

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

Example: Legal transition

```
N10 WHEN $AA_IM[Y] >= 5 DO MOV[Y]=–1      ; In position +5, start
                                          ; axis in negative
                                          ; direction
N20 WHEN TRUE DO POS[Y]=20 FA[Y]=500      ; Start Y axis when
                                          ; block is loaded
```

On–the–fly transitions for axis couplings

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

– See Subsection 2.4.19 and

References: /M3/, Coupled Motions and Master Value Couplings

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

2.4.18 Setting actual values from synchronized actions

Application

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

Function

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

Programming

The value for **one** axis can be programmed in synchronized actions.
e.g.:

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

With PRESETON(axis,value)

Axis:	Axis whose control zero must be altered
Value:	Amount by which control zero must be altered.

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

Please see Subsection 6.7.3 for an example of how to use PRESETON in conjunction with an ”On–the–fly parting” application.

Note

The ”PRESETON” preset actual value memory may only be programmed using the keywords ”WHEN” or ”EVERY”.

2.4.19 Coupled motions and activating/deactivating couplings

Introduction

The following functions are described in detail in:

References: /FB/, M3, Coupled Motions

The following functions are described in detail:

- Coupled motion
  Following axis(axes) is(are) linked to a master axis via a coupling factor.
- Curve tables
  Curve tables represent a (complex) relationship between the master and slave values. The following may be applied as master values:
  - Setpoints generated by the control
  - Actual values measured by encoders
  - Externally specified quantities

  Situations where a following axis is linked to a master axis by means of a curve table are particularly relevant with respect to synchronized actions.
- Master value coupling
  Of the following master value couplings, which may be implemented for part programs:
  - Axis master value coupling
  - Path master value coupling

  only axis master value couplings are available for use in synchronized actions.

Coupled motion

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

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

with:

FA	following axis
LA	master axis
Kf	coupling factor

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

... DO **TRAILOF**(FA, LA, LA2)

with:

FA	following axis
LA	master axis
LA2	master 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, degrees)

with:

LW	Master value
n	Number of the curve table
Degrees	Gradient parameter, result (2 additional opt. Parameters for scaling: – Following axis – Master axis)

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

Calculate 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, degrees)

with:

FW	Slave value
aprLW	Approximate master value, with rich, for a non–unique reversing function of the curve table, a unique master value can be defined
n	Number of the curve table
Degrees	Gradient parameter, result (2 additional opt. Parameters for scaling: – Following axis – Master axis)

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

Axis master value coupling

The coupling between following axis FA and master 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	Master axis
NR	Number of the curve table

Deactivate axis coupling from synchronized action

If the axis master value coupling must be canceled again on the fulfillment of another condition, the action must be programmed as follows:

... DO **LEADOF**(FA, LA)

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.

See 2.3.11.

Detection of synchronism

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

0: Not synchronized

1: Coarse synchronism (acc. to MD 37200: COUPLE_POS_TOL_COARSE)

2: Fine synchronism (acc. to MD 37210: 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

Page 2.1, 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. For more details, please see Description of Functions M3.

Examples

Please see Subsection 6.7.3 for an example illustrating an axis coupling implemented by means of a curve table.

2.4.20 Measurements from synchronized actions

Introduction

From the measuring functions available for part programs:
MEAS, MEAW, MEASA, MEAWA, MEAC

References: /PGA/, Programming Manual Advanced
/FB/, M5, ”Measurements”

Only the following may be used in synchronized actions:

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

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

Note

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

Programming

MEAWA[axis]=(mode, trigger_event_1, trigger_event_2,
trigger_event_3, trigger_event_4)
; activate axial measurem. without deletion of distance–to–go

MEAC[axis]=(mode, measurement memory, trigger_event_1, trigger_event_2,
trigger_event_3, trigger_event_4)
; Activate axial, continuous measurement

Axis: Axis for which measurement is taken

Table 2-3 Mode meanings:

Tens decade	Units decade	Meaning
	0	Abort measurement task
	1	Up to 4 trigger events can be activated **simultaneously**
	2	Up to 4 trigger events can be activated **consecutively**
	3	Up to 4 trigger events can be activated **consecutively**, but with no monitoring of trigger event 1 on START
0		Active measuring system
1		1st measuring system
2		2nd 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 meas. without deletion of distance-to-go

Deletion of distance–to–go can be called explicitly in the synchronized action, see Subsection 2.4.11 and example 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

The system variable $AC_MEA does not provide any information that can be evaluated regarding the validity of the measurement for a measurement called from the synchronized action.

Only one measuring task may be active for each axis.

System variables:

$AA_MEAACT[axis]		supplies the instantaneous measuring status of an axis.
	1	Measurement active
	0	Measurement not active
$A_PROBE[measuring probe]		Supplies the present state of the measuring probe.
	1	Probe switched, high signal
	0	Probe not switched, low signal

Measured values in the machine coordinate system with 2 measuring probes (sensors):

$AA_MM1[axis]	Trigger event1, sensor 1
$AA_MM2[axis]	Trigger event 1, sensor 2
$AA_MM3[axis]	Trigger event 2, sensor 1
$AA_MM4[axis]	Trigger event 2, sensor 2

MEAC

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

$AC_FIFO variables (see Subsection 2.3.6) are provided for the purpose of storing measured values from cyclic measuring processes. See above for mode and trigger events.

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

Machine data

```
MD 28050: MM_NUM_R_PARAM = 300
MD 28258: MM_NUM_AC_TIMER = 1
MD 28260: NUM_AC_FIFO = 2          ; 2 FIFOs
MD 28262: START_AC_FIFO = 100      ; First FIFO starts at R100
MD 28264: LEN_AC_FIFO = 22         ; Each FIFO can store 22 values
MD 28266: 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 ANZAHL
DEF INT INDEX_R
N0    G0 X0
N1    MEAC[X]=( 1, 1, 1) POS[X]=100
                                  ; Mode   = 1, simultaneous
                                  ; No. FIFO       = 1
                                  ; Trigger event 1= rising edge, measuring encoder 1
N2    STOPRE                      ; stop preparation
N3    MEAC[X]=( 0)                ; abort continuous measurement
N4    ANZAHL= $AC_FIFO1[4]        ; number of measured values received in the FIFO variables
N5    ANZAHL= ANZAHL – 1
N6    FOR INDEX_R= 0 TO ANZAHL
N7    R[INDEX_R]= $AC_FIFO1[0]    ; enter FIFO contents in R0 – ...
N8    ENDFOR                      ; after reading–out, the FIFO variable is empty
```

Example 2.
All rising edges 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: that the measured values must be
fetched and stored in ascending order in R1 as a parallel operation in one synchronized action. The number of stored measured values is entered in R0.

Program 2:

```
N0    G0 X0                       ; Rapid traverse to the starting point
N1    $AC_MARKER[1]=1             ; Marker 1 as index for computation variables R[..]
N2    ID=1 WHENEVER $AC_FIFO1[4]>=1
      DO $R[$AC_MARKER[1]]= $AC_FIFO1[0] $AC_MARKER[1]=$AC_MARKER[1]+1
                                  ;Synchronized action as test:
                                  ; if 1 or several measured values are in the FIFO variable
                                  ; read–out the oldest value from FIFO and save in the actual R[ ..]
                                  ; increase index for R by 1
```

```
N3    MEAC[X]=( 1, 1, 1, –1)  POS[X]=100   ; activate continuous measurement, motion
                                           ; after X = 100
                                           ; mode   = 1, simultaneous
                                           ; No_FIFO = 1
                                           ; trigger event 1= 1, rising edge encoder 1
                                           ; trigger event 2= –1, falling edge encoder 1
N4    MEAC[X]=(0)                          ; de–select measurement
N5    STOPRE                               ; stop preprocessing
N6    R0= $AC_MARKER[1]                    ; numbers of values stored R0
```

Example 3:
Continuous measurement with explicit deletion of distance–to–go after 10 measurements

Program 3:

```
N1    WHEN $AC_FIFO1[4]>=10
      DO  MEAC[X]=(0) DELDTG(X)            ; final condition as synchronized action:
                                           ; if 10 or more measured values are in the FIFO
                                           ; variables,
                                           ; de–select continuous measurement and
                                           ; delete distance to go
N2    MEAC[X]=( 1,1,1,–1) G01 X100 F500    ; continuous measurement from the part program active.
                                           ; mode = 1, simultaneously
                                           ; No_FIFO = 1, FIFO variable 1
                                           ; trigger event 1= 1, rising edge measuring probe 1
                                           ; trigger event 2= –1, falling edge 1

N3    MEAC[X]=(0)                          ; de–select continuous measurement
N4    R0= $AC_FIFO1[4]                     ; actual number of measured values
```

Priority with more than one measurement

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

When a measuring task is started for the same axis, this means that trigger events are re–activated and the measurement results are reset.
If switch–out measuring task (mode 0) is programmed without a measuring task having first been activated, then no special response is issued.
Measuring task that were started from the part program, cannot be controlled from synchronized actions.
If, from a synchronized action, a measuring task for an axis is started, and, for this axis, a measuring task from the part program is already active, then an alarm is generated.
If a measuring task from synchronized actions is active, then it is no longer possible to start measurements from the part program.

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	Procedure
Mode change	A measurement task activated by means of a modal synchronized action is not affected by a change in operating mode. It remains active beyond block limits.
RESET	Measurement task is aborted

State	Procedure
Block search	Measurement tasks are collected, but not activated until the programmed condition is fulfilled.
REPOS	Activated measurement tasks are not affected.
End of program	Measurement tasks started from static synchronized actions remain active.

2.4.21 Setting and deletion of wait markers for channel synchronization

Introduction

Coordination of operational sequences in channels is described in:

References: /FB/, K1, Mode Group, Channel, Program Operation

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

Set wait marker

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 mark

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

2.4.22 Setting alarm/error reactions

Fault situations

Setting an alarm is one possibility of responding to fault states.

Application:
Using the SETAL command, cycle alarms can be set from synchronized actions.

Additional possibilities of responding to faults include:

- Stop axis See Subsection 2.4.12
- Set output See Subsection 2.4.2
- Other actions described in Subsection 2.4

Example set alarm

```
ID=67 WHENEVER $AA_IM[X1] – $AA_IM[X2] < 4.567 DO SETAL(61000)

; Set alarm if distance (actual value of axis X1 – actual value of axis X2) ;
exceeds the critical value 4.567.
```

Cycles and cycle alarms

For information about cycles and cycle alarms, please see

References: /PGZ/, Programming Manual Cycles

2.4.23 Evaluating data for machine maintenance

Function

Machine operators are able to use system variables in part programs, synchronized actions and via the OPI 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 Description of Functions – Basic Machine: Interface signals from axis/spindle.

Availability

The values for machine maintenance are available if the global NCK machine data MD ...: MM_MAINTENANCE_MON is set and axis–specific machine data has been used to indicate which data should be provided for each axis involved.

MD: MM_MAINTENANCE_MON is used to activate the function of and prepare the memory for the values indicated in the axis–specific MD. Changes to MD ...: MM_MAINTENANCE_MON take effect at POWER ON.

Axis–specific values:

The following information can be entered in bit–coded format in MD ...: MAINTENANCE_DATA:

Bit 0:	Total travel distance, total travel time and travel count of the axis
Bit 1:	Total travel distance, total travel time and travel count at **high axis speeds**. 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

$MN_MM_MAINTENANCE_MON = TRUE

$MA_MAINTENANCE_DATA[0]=1
$MA_MAINTENANCE_DATA[1]=1
$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

$AA_TRAVEL_DIST	Total travel distance in mm or degrees
$AA_TRAVEL_TIME	Total travel time in seconds
$AA_TRAVEL_COUNT	Total travel count
$AA_TRAVEL_DIST_HS	Total travel distance at high speeds in mm or degrees
$AA_TRAVEL_TIME_HS	Total travel time in seconds at high speeds
$AA_TRAVEL_COUNT_HS	Total travel count at high speeds
$AA_JERK_TOT	Total axis jerk in m/s^3
$AA_JERK_TIME	Axis travel time with jerk in seconds
$AA_JERK_COUNT	Axis travel count with jerk

can be read from the part program and from synchronized actions.

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 distance traveled by X axis during
                    ; processing of the processing area in the part program
```

2.5 Calling technology cycles

Definition

A technology cycle is a sequence of actions that are executed sequentially in the interpolation cycle. The actions described in Section 2.4 can be combined to form 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:
  The actions are executed simultaneously in the interpolation cycle in which the condition is fulfilled.
- Actions are combined to form a technology cycle:
  The actions in the technology cycle are sequentially executed 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. movement of command axis, see Subsection 2.4.13) take **several** interpolation cycles to complete. If the function is complete (e.g. exact stop on positioning of axis), the next block is executed in the following interpolation cycle.

Each block requires at least one interpolation cycle. If a block contains several single–cycle actions, then these are all processed in one interpolation cycle. Fig. 2-9 provides examples to indicate which actions are single–cycle and which are multi–cycle.

Application

One possible application of technology cycles is to move each axis using a separate axis program.

Programming

A technology cycle can be activated as a function of a condition in a modal/static synchronized action.

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

Search path

The call search path is the same as for subprograms and cycles.

Example:

```
...
ID=1 EVERY $AA_IM[Y]>=10  DO AX_X                ; AX_X subprogram
                                                 ; name for axis program for X axis
```

```
AX_X:                ; Axis program
POS[X]=$R[7] FA[X]=377
$A_OUT[1]=1
POS[X]=R10
POS[X]=–90
M30
```

Note

If the condition is fulfilled again while the technology cycle is being processed, the cycle is not restarted. If a technology cycle has been activated from a synchronized action of the WHENEVER type and the relevant condition is still fulfilled at the end of the cycle, then it will be restarted.

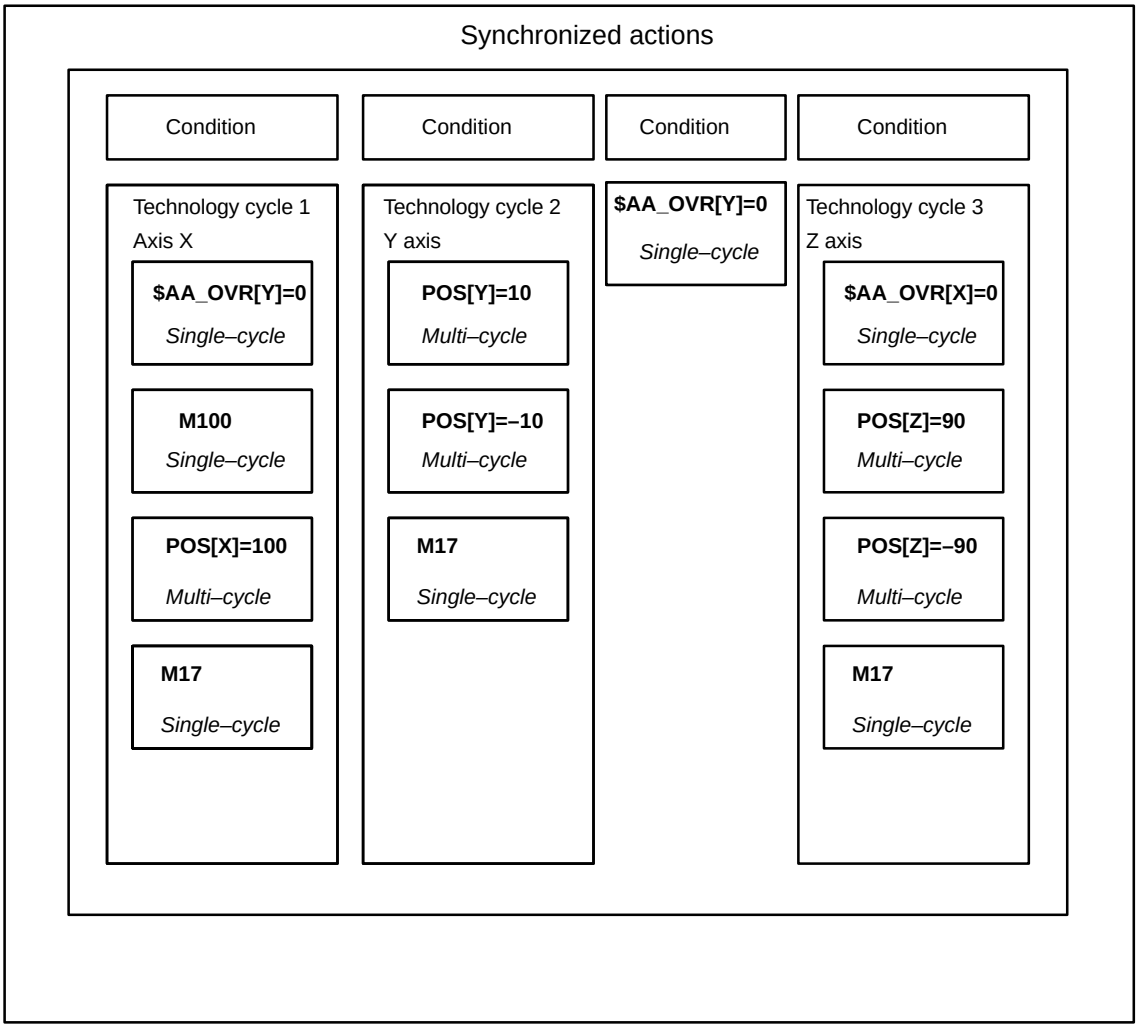


Fig. 2-9 Several technology cycles

Example (2) for coordinated axis motion:
Various axis programs are started by setting digital NC input.

Main program:

```
...
ID=1 WHEN $A_IN[1]==1 DO ACHSE_X
ID=2 WHEN $A_IN[2]==1 DO ACHSE_Y
ID=3 WHEN $A_IN[3]==1 DO AA_OVR[Y]=0
ID=4 WHEN $A_IN[4]==1 DO ACHSE_Z
M30
```

Axis programs:

ACHSE_X:

```
$AA_OVR[Y]=0
M100
POS[X]=100
M17
```

ACHSE_Y:

```
POS[Y]=10
POS[Y]=–10
M17
```

ACHSE_Z:

```
$AA_OVR[X]=0
POS[Z]=90
POS[Z]=–90
M17
```

2.5.1 Coordination of synchronized actions, technology cycles, part program (and PLC)

Control of technology cycles

Technology cycles/synchronized actions are controlled via the identification number of the synchronized action in which they are programmed as an action:

Means of coordination

Keyword	Meaning	PP	SA
	Call legal in part program Call legal in synchr. action/technology cycle	+	 +
LOCK(ID)	Inhibit technology cycle. The positively 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. Completed synchronized actions of the WHEN type are not processed again after RESET.		+
CANCEL(ID)	Synchronized action is deleted.	+	

- LOCK(ID), UNLOCK(ID) by PLC see Subsection 2.6.1

Note

A synchronized action contains a technology cycle call. No further actions may be programmed in the same block in order to ensure that the assignment between ID number and relevant technology cycle is unambiguous.

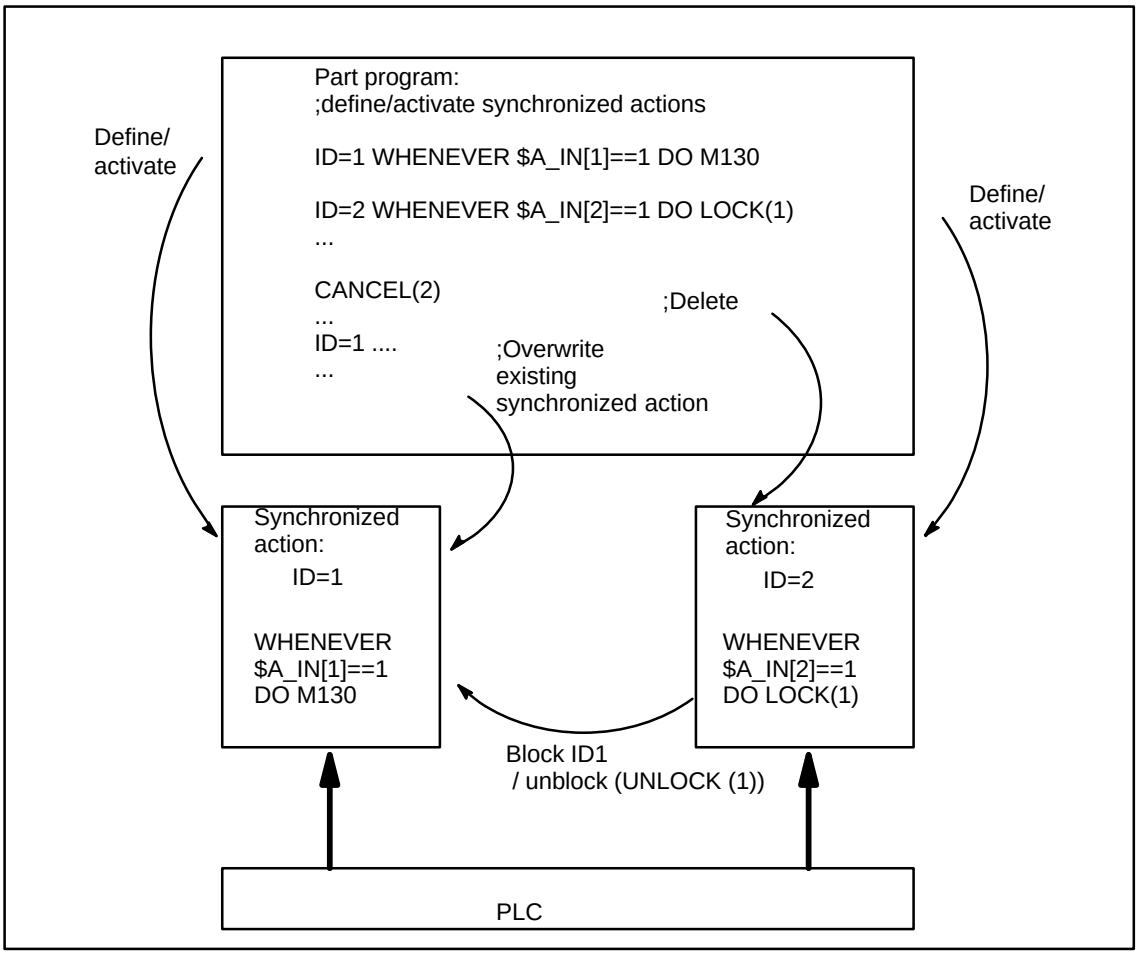


Fig. 2-10 Setting up/locking modal synchronized actions/deleting

2.6 Control and protection of synchronized actions

2.6.1 Control via PLC

Function

Modal synchronized actions (ID, IDS) can be locked or enabled from the PLC.

- Disabling of all modal synchronized actions
- Selective disabling of individual synchronized actions

Control scope

The PLC can control 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 interface:
DB21–30, DBB308–315. Protected synchronized actions are never tagged as being possible to disable. See 2.6.2.

Disable all synchronized actions

The PLC application program can set DB 21–30, DBB1 bit 2 to disable (lock against activation) all modal synchronized actions that are already defined in the NC and stored against activation. In this case, protected synchronized actions are an exception. Please see Subsection 2.6.2.
Setting DB 21–30, DBB1 bit 2 to 0 cancels the general disable by the PLC again.

Using selective disabling

One bit is reserved for each of first 64 IDs (1–64) in the PLC interface (DB 21–30, DBB 300 bit 0 to DB21–30 DBB 307 bit 7).

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

Setting the bits corresponding to the ID, IDS number to 0 in
DB 21–30, DBB 300, bit 0 to DB 21–30, DBB 307 bit 7
causes the PLC to enable a previously disabled synchronized action.

Updating the selective disabling

If the PLC user program has made changes in the range DB 21–30, DBB 300 bit 0 to DB 21–30, DBB 307 bit 7, the changes must be activated with DB 21–30 DBX280.1.

Selective disabling status signal

If selective disabling was activated by the NCK, this is indicated in DB 21–30 DBX.281.1.

Reference: /LIS/, Lists, Interface Signals

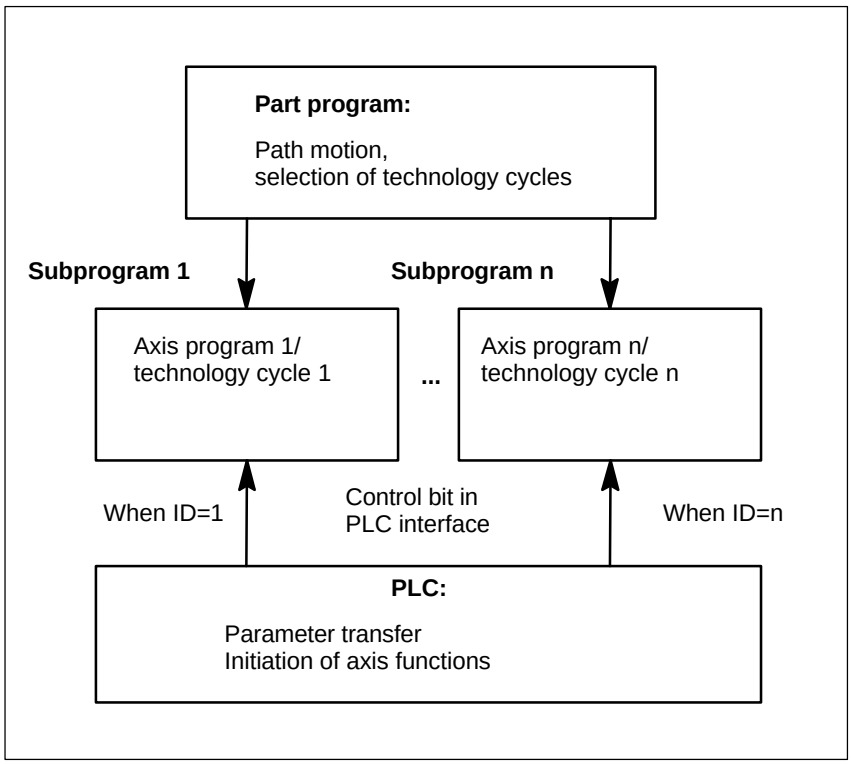


Fig. 2-11 Axis programs/technology cycles

Reading/writing of PLC data

In SW version 4 and later, PLC data can 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 variable

References: /FB/, P3, "Basic PLC Program"

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 Subsection 2.3.11.

2.6.2 Protected synchronized actions

Global protection

Function

Machine data
MD 11500: PREVENT_SYNACT_LOCK
can be programmed to define an area of protected synchronized actions. Synchronized actions with ID numbers within the protected area can **no longer be**:

- Overwritten
- Deleted (CANCEL) or
- Disabled (LOCK)

once they have been defined. Protected synchronized actions cannot be disabled via the PLC either. They are indicated to the PLC as non–lockable in the interface. See Subsection 2.6.1.

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 MD 11500

```
$MN_PREVENT_SYNACT_LOCK[0]= i  ; i Number of the first ID to be disabled
$MN_PREVENT_SYNACT_LOCK[1]= j  ; j Number of the last ID to be disabled
```

i and j can also be specified, interchanged.
If i = 0 and j = 0, no synchronized actions are protected.

Channel–specific protection

Function

The operator can use the channel–specific machine data
MD 21240 : PREVENT_SYNACT_LOCK_CHAN
to define an area of protected synchronized actions for the channel. Synchronized actions with ID numbers within the protected area can **no longer be**:

- Overwritten
- Deleted (CANCEL) or
- Disabled (LOCK)

once they have been defined. Protected synchronized actions cannot be disabled via the PLC either. They are indicated to the PLC as non–lockable in the interface. See Subsection 2.6.1.

Application

See above

Notation of MD 21240

```
CHANDATA(C)    ; with C channel number
$MC_PREVENT_SYNACT_LOCK_CHAN[0]= k
                ; k Number of the first ID to be disabled for the channel
$MC_PREVENT_SYNACT_LOCK_CHAN[1]= l
                ; l Number of the last ID to be disabled for the channel.
```

k and l can also be specified, interchanged.
If k = 0 and l = 0, no synchronized actions are protected.

k = –1 and l = –1 specify that the global area of protected synchronized actions programmed for the channel with
MD 11500: PREVENT_SYNACT_LOCK applies.

Note

Protection for synchronized actions must be canceled 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 inhibits is identical independent of whether they are specified as:
Global inhibit or as
channel–specific inhibit.

Example

In a system with 2 channels, synchronized actions should be protected as follows:
In the 1st channel, IDs 20 to 30 should be protected and
in channel 2, IDs 25 to 35 should be protected. Global and channel–specific protection may be mixed.

```
$MN_PREVENT_SYNACT_LOCK[0] = 25              ; global protection

$MN_PREVENT_SYNACT_LOCK[1] = 35              ; global protection

CHANDATA(1)

$MC_PREVENT_SYNACT_LOCK_CHAN[0] = 20
; Only the channel–specific MD (first ID number to be protected) is effective in
the first channel

$MC_PREVENT_SYNACT_LOCK_CHAN1] = 30
; Only the channel–specific  MD (last ID number to be protected) is effective in
the first channel

CHANDATA(2)

$MC_PREVENT_SYNACT_LOCK_CHAN[0] = –1
                    ; The global machine data
                    ; $MN_PREVENT_SYNACT_LOCK is effective in the 2nd channel!

$MC_PREVENT_SYNACT_LOCK_CHAN[1] = –1
 ...
```

2.7 Control system response for synchronized actions in specific operational states

2.7.1 Power On

No synchronized actions are active during POWER ON. Static synchronized actions that are required to be active immediately after Power On must be activated within an ASUB started by the PLC.

References: /FB/, P3, Basic PLC Program
/FB/, K1, Mode Group, Channel, Program Operation Mode

This arrangement can be used only on condition that SW 4 with ASUBs in all operating modes functionality is installed.

Examples:

- Adaptive Control
- Safety Integrated, gating logic formulated by means of synchronized actions

2.7.2 RESET

For positioning axis motion

All positioning motions initiated from synchronized actions are aborted on NC reset. Active technology cycles are reset.

ID

Synchronized actions programmed locally (i.e. with ID=...) are deselected on NC reset.

IDS

Static synchronized actions (programmed with IDS = ...) remain active after NC reset. Motions can be restarted from static actions after NC reset.

Other reactions, dependent on actions

RESET continued

Synchronized action/ technology cycle	Modal and non–modal Active action is aborted, synchronized actions are canceled	Static (IDS) Active action is aborted, technology cycle is reset
Axis/positioning spindle	Motion is aborted	Motion is aborted
Speed–controlled spindle	$MA_SPIND_ACTIVE_AFTER_RESET== TRUE: Spindle remains active $MA_SPIND_ACTIVE_AFTER_RESET==FALSE: Spindle stops.	$MA_SPIND_ACTIVE_AFTER_RESET== TRUE: Spindle remains active $MA_SPIND_ACTIVE_AFTER_RESET==FALSE: Spindle stops.
Master value coupling	$MC_RESET_MODE_MASK, Bit13 == 1: master value coupling remains active $MC_RESET_MODE_MASK, Bit13 == 0: master value coupling will be resolved	$MC_RESET_MODE_MASK, Bit13 == 1: master value coupling remains active $MC_RESET_MODE_MASK, Bit13 == 0: master value coupling will be resolved
Measuring operations	Measuring operations started from synchronized actions are aborted	Measuring operations started from static synchronized actions are aborted

2.7.3 NC STOP

Motion start from static

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 with ”PLC controlling axis” IS (DB31, ... DBX28.7). This type of access is now stopped with an axial STOP **rather than** an NC STOP.

Motion start from non–modal and modal

Axis motions started from non–modal and modal actions are interrupted and then restarted by NC START. Speed–controlled spindles remain active.

The synchronized actions, belonging to the active block, remain active.
Example:
Set output: ... DO $A_OUT[1] = 1

2.7.4 Mode change

A differentiation is made between program–local and static synchronized actions. Synchronized actions, activated with the keyword**IDS**, remain active after the operating mode has been changed. All other synchronized actions are deactivated in response to a mode change and reactivated on switchover to AUTO mode for repositioning.

Example:

```
N10    WHEN $A_IN[1] == 1 DO DELDTG
N20    G1      X10 Y 200 F150 POS[U]=350
```

Block N20 contains a STOP command. The operating mode is switched to JOG. If deletion of distance–to–go was not active prior to the interruption, then the synchronized action programmed in block N10 is reactivated when AUTO mode is selected again and the program continued.

2.7.5 End of program

Static synchronized actions remain active beyond the end of the program.
Blockwise and modal synchronized actions are interrupted.
Static and modal synchronized actions remain effective in the M30 block.
They can be interrupted with CANCEL before M30. Polynomial coefficients programmed with FCTDEF remain active after the end of program.

2.7.6 Response of active synchronized actions to end of program and change in operating mode

See Sections 2.7.4 and 2.7.5.

Synchronized action/ technology cycle	**Modal and non–modal actions** are aborted	**Static actions (IDS)** remain active
Axis/positioning spindle	M30 is delayed until the axis/spindle is stationary.	Motion continues
Speed–controlled spindle	Program end: $MA_SPIND_ACTIVE_AFTER_RESET== TRUE: Spindle remains active $MA_SPIND_ACTIVE_AFTER_RESET==FALSE: Spindle stops Spindle remains active when the operating mode changes	Spindle remains active
Master value coupling	$MC_RESET_MODE_MASK, Bit13 == 1: master value coupling remains active $MC_RESET_MODE_MASK, Bit13 == 0: master value coupling will be resolved	A coupling started from a static synchronized action remains active
Measuring operations	Measuring operations started from synchronized actions are aborted	Measuring operations started from static synchronized actions remain active

2.7.7 Block search

General

Synchronized actions in the program, which have been interpreted during the block search, are collected, but their conditions are not evaluated. No actions are executed. Processing of synchronized actions does not commence until NC Start.

IDS

Synchronized actions that are programmed with keyword IDS and are already active remain operative during the block search.

Polynomial coefficients

Polynomial coefficients programmed with FCTDEF are collected **with calculation** during a block search, i.e. they are written to system variables.

2.7.8 Program interruption by ASUB

ASUB start

Modal and static motion–synchronous actions remain active and are also operative in the asynchronous subprogram (ASUB).

ASUB end

If the asynchronous sub–program is not continued with repos, then the modal and static motion synchronized actions, changed in the asynchronous sub–program, continue to be effective in the main program.
Positioning motion, started from synchronized actions, behave as for the operating mode change.
Motion started from blockwise and modal actions, are stopped and possibly continued with repos. Motions started from static synchronized actions continue uninterrupted.

2.7.9 REPOS

In the remaining block, the synchronized actions apply just the same as in the interrupted block.
Changes to modal synchronized actions in the asynchronous sub–program are not effective in the interrupted program.
Polynomial coefficients, programmed with FCTDEF, are not influenced by ASUB and REPOS.
The coefficients from the calling program are effective in the sub–program. The coefficients from the asynchronous sub–program continue to be effective in the calling program.

If positioning motion from synchronized actions is interrupted with an operating mode change or the start of an interrupt program, then these are continued with REPOS.

2.7.10 Response to alarms

Axis and spindle motions started by means of synchronized actions are decelerated in response to an alarm involving a motion stop instruction. All other actions (such as Set output) continue to be executed.

If an alarm is activated by a synchronized action, then the action is no longer processed in the next interpolation cycle, i.e. the alarm is output only once.
Alarms that respond with an interpreter stop only take effect once the precoded blocks have been processed.

All other actions are further processed.
If a technology cycle generates an alarm with motion stop, then processing of the relevant technology cycle ceases.

2.8 Configuration

2.8.1 Configurability

Number of synchronized action elements

The number of programmable synchronized action blocks depends entirely on the configurable number of synchronized action elements. The number of storage elements for motion–synchronous actions (synchronized action elements) is defined in machine data

MD 28250: MM_NUM_SYNC_ELEMENTS.

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

Onesynchronized action element is required for each of the following:

- A comparison expression in a condition
- An elementary action
- The synchronized action block

Example:
A total of four elements are used for the subsequent synchronized action block.

```
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 MD 28250: $MC_MM_NUM_SYNC_ELEMENTS is selected such that it is possible to activate the maximum presetting for SW 3 and earlier of 16 synchronized actions.

Note

If the user does not wish to program any synchronized actions, then he can reset the value in MD 28250: MM_NUM_SYNC_ELEMENTS to 0 so as to save approximately 16 KB of DRAM memory.

Indication

The status display for synchronized actions (refer to Chapter 2.9) 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

If the elements are used up in the program sequence, then an alarm is issued. The user can then increase the number of synchronized action elements or can appropriately modify his program.

Number of FCTDEF functions

The number of programmable FCTDEF functions for each block can be configured via machine data
MD 28252: MM_NUM_FCTDEF_ELEMENTS.

The default value for all types of control is 3.
The control–specific maximum value can be found in

References: /LIS/, 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:
Deviations are possible for other control types.

NC language	**Time required**	
	Total	**Text in bold print**
Basic load for a synchronized action, if condition is not fulfilled: **WHENEVER FALSE** DO $AC_MARKER[0]=0	10 µs	**~10 µs**
Read variable: WHENEVER **$AA_IM[Y]>10** DO $AC_MARKER[0]=1	11 µs	**~1 µs**
Write variable: DO ***$R2=1***	11–12 µs	**~1–2 µs**
Read/write setting data: DO **$$SN_SW_CAM_MINUS_POS_TAB_1[0]=20**	24 µs	**~14 µs**
Basic arithmetic operations, e.g. multiplication: DO $R2=**$R2*2**	22 µs	**~12 µs**
Trigonometric functions (e.g. cos): DO $R2=**COS($R2)**	23 µs	**~13 µs**
Start positioning axis motion: WHEN TRUE DO **POS[z]=10**	83 µs	**~73 µs**

2.9 Diagnostics (only with HMI Advanced)

Diagnostic functionality

The following special test tools are provided for diagnosing synchronized actions:

- Status display of synchronized actions in the machine operator area
- Displaying system variables in the Parameter operator area
  Current values of all of the synchronized action variables can be displayed. (display real–time variables).
- Logging system variables in the Parameter operator area
  Variable characteristics can be traced (recorded) in the interpolation cycle grid. (log real–time variables).

This functionality is structured in the operator interface in the following way:

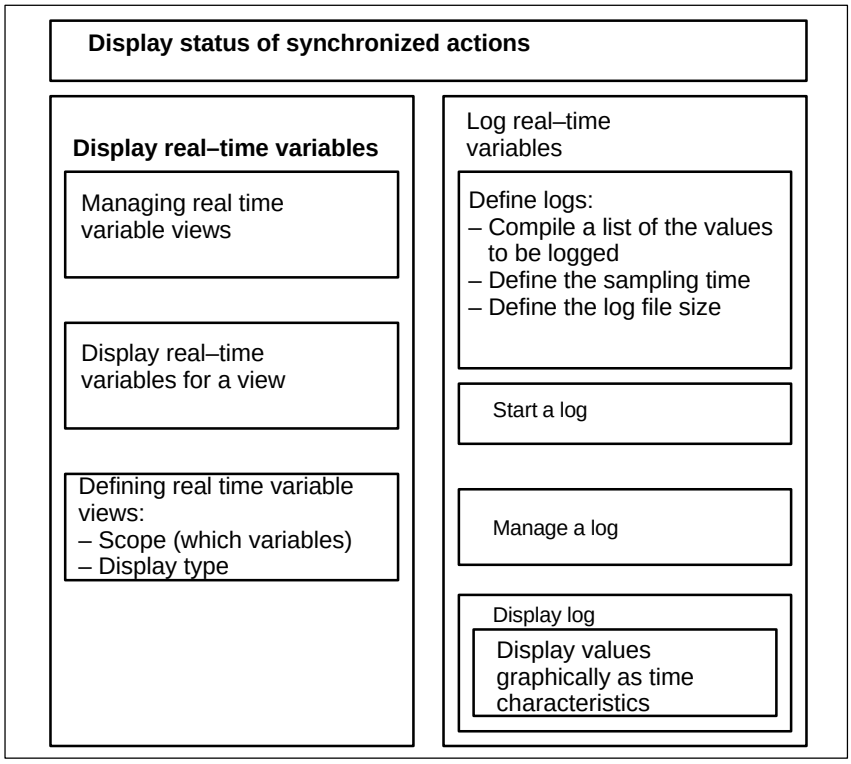


Fig. 2-12 Functionality of test tools for synchronized actions

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

References: /BAD/, Operating Manual HMI Advanced.

2.9.1 Display status of synchronized actions

Status display

The status display contains the following information:

- 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

Synchronized actions are categorized as follows:

– 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
- Disabled LOCK has been set for the synchronized action
- Active Action currently being executed If the action consists of a technology cycle, the current line number in the cycle is also displayed.

Complete synchronized actions

A search function can be used to display the originally programmed line in NC language for each displayed synchronized action.

2.9.2 Display real–time variables

System variables can be monitored for the purpose of monitoring synchronized actions. The permissible variables are listed in a recommended list from where they can be selected.
The complete list of individual system variables with identification of the write access W and read access R for the synchronized actions is provided in:

References: /PGA1/, List 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 real-time variables for a view The values assigned to a view are displayed by calling the corresponding user–defined view.

2.9.3 Log real–time variables

Initial situation To be able to trace events in synchronized actions, it is necessary to monitor the action status in the interpolation cycle.

Method The values selected 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.

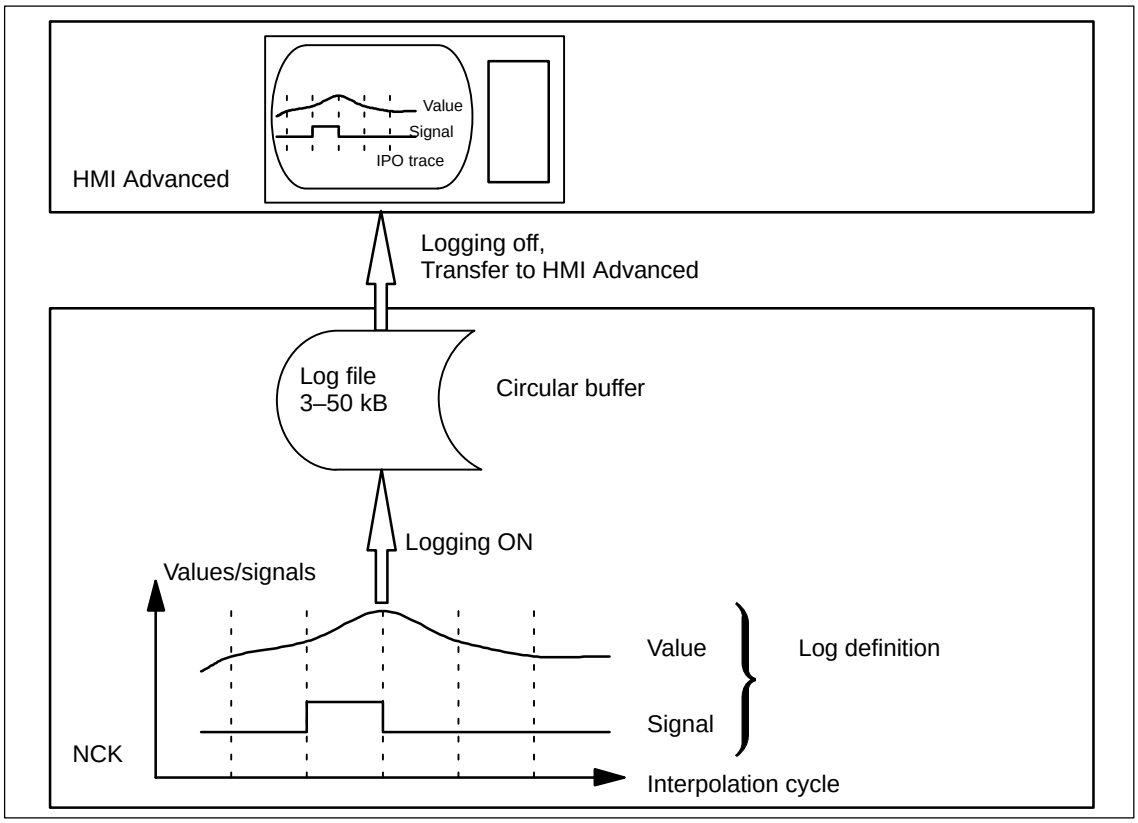


Fig. 2-13 Schematic representation of Log real–time variables process

Operating For information about operating the logging function, please see:

References: /BAD/, Operating Manual 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 KB. 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:

- Operating
- 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 data is made available on the HMI for storage and evaluation (graphic log). Logging can be stopped by:

- Operating
- 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 names of the user's choice. They can be called later for initialization and start of recording or for modification and deletion.

3 Supplementary 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
  - HW
  - SW (export/standard versions)
- The availability of functions that can be initiated by ”Actions”:
  - Standard functions
  - Functions that are available as options

The performance of control systems and their variants as well as functions supplied as options are described in catalogs specific to the SW version:

References: /BU/, Ordering Information, **Catalog NC 60.1** and in /LIS/, Lists (Book 1 and Book 2)

Further, the functions of the synchronized actions depend on the list of synchronized actions – system variables that can be read/changed – including machine and setting data (also dependent on the SW release). System variables that may be used in conjunction with specific SW versions are described in:

References: /PGA1/, List Manual, System Variables (valid for the particular SW release)

Extensions in SW 4

The following extensions have been introduced with SW 4:

- Diagnostic facilities for synchronized actions
- Availability of additional real-time variables
- Complex conditions in synchronized actions
  - Basic arithmetic operations
  - Functions
  - Indexing with real-time variables
  - Access to setting and machine data
  - Logic operators
- Configurability
  - Number of simultaneously active synchronized actions
  - Number of special variables for synchronized actions
- Activate command axes/axis programs/technology cycles from synchronized actions
- PRESET from synchronized actions
- Couplings and coupled motions from synchronized actions

  - Powering on
  - Powering off
  - Parameterization
- Use of measuring functions from synchronized actions
- SW cams
  - Redefinition of position
  - Redefinition of lead times
- Deletion of distance-to-go without preprocessing stop
- Static synchronized actions (modes other than AUTO possible)
- Synchronized actions:
  - Protection against overwriting and deletion
  - Stopping, continuing, deleting
  - Resetting technology cycles
  - Parameterizing, enabling and disabling from PLC
- Overlaid movement/optimized clearance control
- Coordinating channels from synchronized actions
- Starting ASUBs from synchronized actions
- Non-modal auxiliary function outputs
- 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

Extensions in SW 5 and higher

From SW 5 to 7, the following services are additionally provided:

- Synchronized actions, which can be tagged for the PLC
- Availability of additional real-time variables
- Access to PLC I/O (option)
- 255 parallel synchronized actions per channel are possible with the option ”Synchronized actions step 2”.
- Static synchronized actions IDS that are active beyond the program end and are effective in all operating mode are possible using the option ”Inter-mode group actions, ASUBs and synchronized actions”.
- Online calculations and online tool offsets (from SW 6).
- Axis replacement using synchronized actions and the technology cycle (from SW 7).

4 Data Descriptions (MD, SD)

4.1 General machine data

??? Classify **MD number**	**MM_MAINTENANCE_MON** **Activate recording of maintenance data**		
Default setting: 0	Minimum input limit: 0	Maximum input limit: 1	
Changes effective after POWER ON	Protection level: 2 / 7	Unit: –	
Data type: BOOLEAN	Applies as of SW 6.5, 7.1		
Meaning:	This machine data is used to activate the recording of data for machine maintenance. Data to be recorded must be set with the axis–specific MD ... : MAINTENANCE_DATA.		

11500 **MD number**	**PREVENT_SYNACT_LOCK** **Protected synchronized actions**		
Default setting: 0, 0	Minimum input limit: 0	Maximum input limit: 255	
Changes effective after POWER ON	Protection level: 2 / 7	Unit: –	
Data type: DWORD	Applies as of SW 4.1		
Meaning:	First and last ID of a protected synchronized action range. Synchronized actions with IDs in this range cannot be overwritten or inhibited in the program (NC: CANCEL, LOCK). Neither can protected synchronized actions be disabled (LOCK) by the PLC. Typical application: The machine manufacturer defines safety logic in an asynchronous subprogram. This logic is started by the PLC during POWER ON. The range of IDs used is locked out via this machine data, thus preventing the end customer from modifying or deactivating the safety logic integrated by the machine manufacturer. Note: Protection for synchronized actions must be canceled while generating synchronized actions to be protected, otherwise power on will have to be executed at every change to allow the logic to be re–defined. For 0,0, there is no range of protected synchronized actions. the function is not switched on. The values are read as absolute values. Upper and lower values can be specified in any sequence. If necessary, the configuration can be reconfigured using the channel–specific MD 21240: PREVENT_SYNACT_LOCK_CHAN.		
Related to	MD 21240: PREVENT_SYNACT_LOCK_CHAN		

11510 **MD number**	**IPO_MAX_LOAD** **Maximum permissible IPO load**		
Default setting: 0	Minimum input limit: 0	Maximum input limit: 100	
Changes effective after POWER ON	Protection level: 2 / 7	Unit: %	

11510 **MD number**	**IPO_MAX_LOAD** **Maximum permissible IPO load**
Data type:	Applies as of SW
Meaning:	Activate utilization evaluation using synchronized actions. This MD can be used to specify the IPO computing time (as a % of the IPO cycle) at and above which the system variable $AN_IPO_LOAD_LIMIT should be set to TRUE. If, after the value has been exceeded, it is again fallen below, then the variable is again set to FALSE. If the machine data is 0, then this diagnostics function is de–activated.

4.2 Channel-specific machine data

21240 **MD number**	**PREVENT_SYNACT_LOCK_CHAN** **Protected synchronized actions for channel**		
Default setting: –1, –1	Minimum input limit: –1	Maximum input limit: 255	
Changes effective after POWER ON	Protection level: 2 / 7	Unit: –	
Data type: DWORD	Applies as of SW 6.4		
Meaning:	First and last ID of a protected synchronized action range. Synchronized actions with IDs in this range cannot be overwritten or inhibited in the program (NC: CANCEL, LOCK). Neither can protected synchronized actions be disabled (LOCK) by the PLC. The range of IDs used is locked out via this machine data, thus preventing the end customer from modifying or deactivating the safety logic integrated by the machine manufacturer. Note: Protection for synchronized actions must be canceled while generating synchronized actions to be protected, otherwise power on will have to be executed at every change to allow the logic to be re–defined. For 0,0, there is no range of protected synchronized actions. the function is not switched on. The values are read as absolute values. Upper and lower values can be specified in any sequence. –1, –1 indicates that the ID numbers programmed for the channel with MD 11500: PREVENT_SYNACT_LOCK are valid.		
Related to	MD 11500: PREVENT_SYNACT_LOCK		

28250 **MD number**	**MM_NUM_SYNC_ELEMENTS** **Number of elements for expressions in synchronized actions**		
Default setting: 159	Minimum input limit: 0	Maximum input limit: 2000	
Changes effective after POWER ON	Protection level: 2 / 7	Unit: –	
Data type: DWORD	Applies as of SW 4.1		
Meaning:	The components of synchronized actions are stored in elements for storage in the control system. An action requires a minimum of 4 elements. The following are occupied: – every operand in the condition 1 element – every action >= 1 element – every assignment 2 elements – every additional operand in complex expressions 1 element. One element occupies approx. 64 bytes.		
References	Programming Manual Advanced		

28252 **MD number**	**MM_NUM_FCTDEF_ELEMENTS** **Number of FCTDEF elements**		
Default setting: 3	Minimum input limit: 0	Maximum input limit: 100	
Changes effective after POWER ON	Protection level: 2 / 7	Unit: –	
Data type: DWORD	Applies as of SW 4.1		
Meaning:	Storage elements are required to store functions in the control system for use by synchronized actions. This MD determines the number of these elements.		

28254 **MD number**	**MM_NUM_AC_PARAM** **Number of $AC_PARAM parameters**	
Default setting: 50	Minimum input limit: 0	Maximum input limit: 10000, SW 6.3 and higher: 20000
Changes effective after POWER ON	Protection level: 2 / 7	Unit: –
Data type: DWORD	Applies as of SW 4.1	
Meaning:	Number of channel-specific $AC_PARAM parameters for synchronized actions	

28255 **MD number**	**MM_BUFFERED_AC_PARAM** **Storage location for $AC_PARAM**	
Default setting: 0	Minimum input limit: 0	Maximum input limit: 1
Changes effective after POWER ON	Protection level: 2 / 7	Unit: –
Data type: DWORD	Applies as of SW 6.3	
Meaning:	The $AC_PARAM system variables can be saved either: 0: In dynamic DRAM (default) 1: In static SRAM System variables saved in SRAM retain their current values after RESET and POWER ON. They can be included in the data backup.	
Related to	MM_NUM_AC_PARAM	
References	/IAD/, Commissioning Manual	

28256 **MD number**	**MM_NUM_AC_MARKER** **Number of $AC_MARKER markers**	
Default setting: 8	Minimum input limit: 0	Maximum input limit: 10000, SW 6.3 and higher: 20000
Change effective after POWER ON	Protection level: 2 / 7	Unit: –
Data type: DWORD	Applies as of SW 4.1	
Meaning:	Number of channel-specific $AC_MARKER markers for synchronized actions	

28257 **MD number**	**MM_BUFFERED_AC_MARKER** **Storage location for $AC_MARKER**	
Default setting: 0	Minimum input limit: 0	Maximum input limit: 1
Changes effective after POWER ON	Protection level: 2 / 7	Unit: –
Data type:	Applies as of SW 6.3	
Meaning:	The system variables $AC_MARKER can be optionally saved: 0: in the dynamic DRAM memory, default setting 1: in the SRAM static memory System variables, saved in the SRAM, retain their current values extending beyond a RESET and power on. They can be included in the data backup.	
Related to	MM_NUM_MARKER	
References	/IAD/ Commissioning Manual	

28258 MD number	MM_NUM_AC_TIMER Number of $AC_TIMER time variables		
Default setting: 0	Minimum input limit: 0	Maximum input limit: 10000	
Changes effective after POWER ON	Protection level: 2 / 7	Unit: –	
Data type: DWORD	Applies as of SW 4.1		
Meaning:	Number of channel-specific $AC_TIMER time variables for synchronized actions		

28260 MD number	NUM_AC_FIFO Number of $AC_FIFO1, $AC_FIFO2, ... variables		
Default setting: 0	Minimum input limit: 0	Maximum input limit: 10	
Changes effective after POWER ON	Protection level: 2 / /	Unit: –	
Data type: DWORD	Applies as of SW 4.1		
Meaning:	Number of FIFO variables, $AC_FIFO1 to $AC_FIFO10, for synchronized actions.		
Application example(s)	FIFO variables can be used, for example, to track products: Information (e.g. product length) can be buffered for each part on a conveyor belt in a separate FIFO variable.		
Related to	MD 28262: START_AC_FIFO		

28262 MD number	START_AC_FIFO Store FIFO variables from R parameter		
Default setting: 0	Minimum input limit: 0	Maximum input limit: 10000	
Changes effective after POWER ON	Protection level: 2 / 7	Unit: –	
Data type: DWORD	Applies as of SW 4.1		
Meaning:	Number of the R parameter, from which FIFO variables are saved. All R parameters with lower numbers can be used as required in the part program. R parameters above the FIFO range cannot be written to from the part program. The number of R parameters must be set using machine data MD 28050: $MC_MM_NUM_R_PARAM so that from the start of the R parameters, all FIFO variables can be accommodated: $MC_MM_NUM_R_PARAM=$MC_START_FIFO + $MC_NUM_AC_FIFO*($MC_LEN_AC_FIFO+6) The FIFO variables have the names $AC_FIFO1 to $AC_FIFOn. They are set–up as fields. Indices 0 – 5 have special meanings: n= 0: When a variable is written with index 0, a new value is stored in the FIFO. When a variable is read with index 0, the oldest element is deleted from the FIFO n=1: Access to first element to be read in n=2: Access to last element to be read in n=3: Sum of all FIFO elements n=4: Number of elements available in FIFO n=5: Current write index relative to beginning of FIFO		
Related to	MD 28260: NUM_AC_FIFO		

28264 MD number	LEN_AC_FIFO Length of $AC_FIFO ... FIFO variables		
Default setting: 0	Minimum input limit: 0	Maximum input limit: 10000	
Changes effective after POWER ON	Protection level: 2 / 7	Unit: –	
Data type: DWORD	Applies as of SW 4.1		
Meaning:	Length of FIFO variables $AC_FIFO1 to $AC_FIFO10. All FIFO variables of a channel have the same length.		
Related to	MD 28262, MD 28260		

28266 **MD number**	**MODE_AC_FIFO** **FIFO processing mode**		
Default setting: 0	Minimum input limit: 0	Maximum input limit: ***	
Changes effective after POWER ON	Protection level: 2 / 7	Unit: –	
Data type: BYTE	Applies as of SW 4.1		
Meaning:	Mod of the FIFO processing: Bit 0 = 1: The sum of all FIFO contents is regenerated on every write access operation. Bit 0 = 0: No summation		
Related to	MD 28260: NUM_AC_FIFO		

4.3 Axis-specific/spindle-specific machine data

18860 **MD number**	**MAINTENANCE_DATA** **Setting for maintenance data to be recorded**		
Default setting: 1	Minimum input limit: 0	Maximum input limit: 3	
Changes effective after RESET	Protection level: 2 / 7	Unit: –	
Data type:	Applies as of SW		
Meaning:	This machine data is used to configure the recording of data for machine maintenance. It enables data to be recorded to be selected on an axis–specific basis. This can be done as follows: Bit 0: Total travel distance, total travel time and travel count Bit 1: Total travel distance, total travel time and travel count at high axis speeds Bit 2: Total jerk, travel time and travel count with jerk Bits 3 to 15 are reserved.		

30450 **MD number**	**IS_CONCURRENT_POS_AX** **Concurrent positioning axis**	
Default setting: 0	Minimum input limit: 0	Maximum input limit: 1
Changes effective after POWER ON	Protection level: 2 / 7	Unit: 1
Data type: Boolean	Applies as of SW 1	
Meaning:	The axis is a concurrent positioning axis. SW 4.3 and higher (not FM–NC): If FALSE: At RESET a neutral axis becomes a channel axis again. If TRUE: At RESET a neutral axis remains in the neutral axis state, and a channel axis becomes a neutral axis.	
References	Starting the command axes see Subsection 2.4.13	

32070 MD number	**CORR_VELO** Axial velocity for handwheel, ext. WO (work offset), cont. dressing, clearance control	
Default setting: 100	Minimum input limit: 0	Maximum input limit: plus
Changes effective after POWER ON	Protection level: 2 / 7	Unit: %
Data type: DWORD	Applies as of SW 3.2	
Meaning:	Limitation of axis velocity for handwheel override, external zero offset, continuous dressing, clearance control $AA_OFF via synchronized actions referenced to JOG velocity MD: JOG_VELO, MD: JOG_VELO_RAPID, MD: JOG_REV_VELO, MD: JOG_REV_VELO_RAPID. The maximum permissible velocity is the maximum velocity in the MD: MAX_AX_VELO. The limitation is applied at this value. If this value is exceeded, an alarm is output. A conversion is made to either linear or rotary axis velocity MD: IS_ROT_AX.	
Application example(s)	Limitation of velocity for traversal of overlaid movements.	

32074 **MD number**	**FRAME_OR_CORRPOS_NOTALLOWED** **Effectiveness of frames and tool length offset**		
Default setting: 0	Minimum input limit: 0	Maximum input limit: 0xFF	
Changes effective after POWER ON	Protection level: 2 / 7	Unit: –	
Data type: DWORD	Applies as of SW 4.2		
Meaning:	The effectiveness of frames and tool length offsets with respect to indexing axes, PLC axes and command axes started from synchronized actions is programmed in this machine data. Bit == 0: Frame or offset (compensation) values are permitted Bit assignment: Bit 0 == 1: Programmable zero offset (TRANS) for indexing axis prohibited. Bit 1 == 1: Scale change (SCALE) for an indexing axis is forbidden Bit 2 == 1: Direction of rotation reversal (MIRROR) forbidden for an indexing axis Bit 3 == 1: DRF offset for an axis forbidden Bit 4 == 1: External zero offset for an axis forbidden Bit 5 == 1: Online tool offset for an axis forbidden Bit 6 == 1: Synchronized action offset for an axis forbidden Bit 7 == 1: Compile cycles offset for an axis forbidden Bit 8 == 1: Axial frames are taken into account for PLC axes Bit 8 == 0: Axial frames are NOT taken into account for PLC axes (bit evaluation for reasons of compatibility) Bit 9 == 1: Axial frames are NOT taken into account for command axes Bit 9 == 0: Axial frames are taken into account for command axes		

32920 MD number	**AC_FILTER_TIME** Filter smoothing constant for adaptive control		
Default setting: 0.0	Minimum input limit: 0.0	Maximum input limit: plus	
Changes effective after POWER ON	Protection level: 2/7	Unit: s	
Data type: DOUBLE	Applies as of SW 2.1		
Meaning:	The following drive actual values can be sensed using the main run variables $AA_LOAD, $AA_POWER, $AA_TORQUE and $AA_CURR: – drive utilization – active power – instantaneous drive torque setpoint – current actual value of the axis or spindle In order to equalize peaks, the measured values can be smoothed using a PT1 filter. The filter time constant is defined in MD: AC_FILTER_TIME (filter smoothing time constant for Adaptive Control). The PT1 filter acts in addition to the filters integrated in the 611–D with respect to the drive torque setpoint or actual current value. Both filters are connected one after the other if, in the system, it is necessary to have both strongly and weakly smoothed values. The filter is disabled by entering a smoothing time of 0 seconds.		
MD irrelevant for	FM–NC with 611A		
Application example(s)	Smoothing of actual current value for Adaptive Control.		

36750 MD number	**AA_OFF_MODE** Effect of value assignment for axial override with synchronized actions		
Default setting: 0	Minimum input limit: 0	Maximum input limit: 7	
Changes effective after POWER ON	Protection level: 2/7	Unit: –	
Data type: BYTE	Applies as of SW 3.2 (bits 1 and 2 in SW 6 and later)		
Meaning:	Main run variable $AA_OFF allows an overlaid movement for the programmed axis to be implemented within a synchronized action. Using the axial MD: AA_OFF_MODE, the following defines how it is taken into account: Bit0: Effect of tool assignment within a synchronized variable: SW 3.2 and higher Bit0 = 0: Absolute value Bit0 = 1: Incremental value (integrator) Bit1: Response of $AA_OFF in the case of a reset Bit1 = 0: $AA_OFF is deselected in the case of a RESET Bit1 = 1: $AA_OFF is retained beyond the RESET (SW 6 and later) Bit2: $AA_OFF in JOG mode Bit2 = 0: No overlaid movement on the basis of $AA_OFF Bit2 = 1: Overlaid movement is interpolated on the basis of $AA_OFF (SW 6 and later)		
Application example(s)	• Clearance control for laser machining (integral) • Joystick–controlled axis traversal (proportional)		

4.4 Setting data

43350 **MD number**	**AA_OFF_LIMIT** **Upper limit of offset value for $AA_OFF clearance control**		
Default setting: 1.0 Ex+8	Minimum input limit: 0	Maximum input limit: ***	
Changes effective after SOFORT	Protection level: 2 / 7	Unit: mm/degrees	
Data type: DOUBLE	Applies as of SW 4.2		
Meaning:	Upper limit of the offset compensation value that can be entered using synchronized actions via the variable $AA_OFF. The limit value acts on the absolute, effective offset (compensation) value. This is used for clearance control for laser machining: The offset (compensation) value is limited so that the laser head cannot get caught in sheet steel cut–outs. System variable $AA_OFF_LIMIT can be used to interrogate as to whether the offset (compensation) value is in the limit range.		

Space for your notes

5 Signal Descriptions

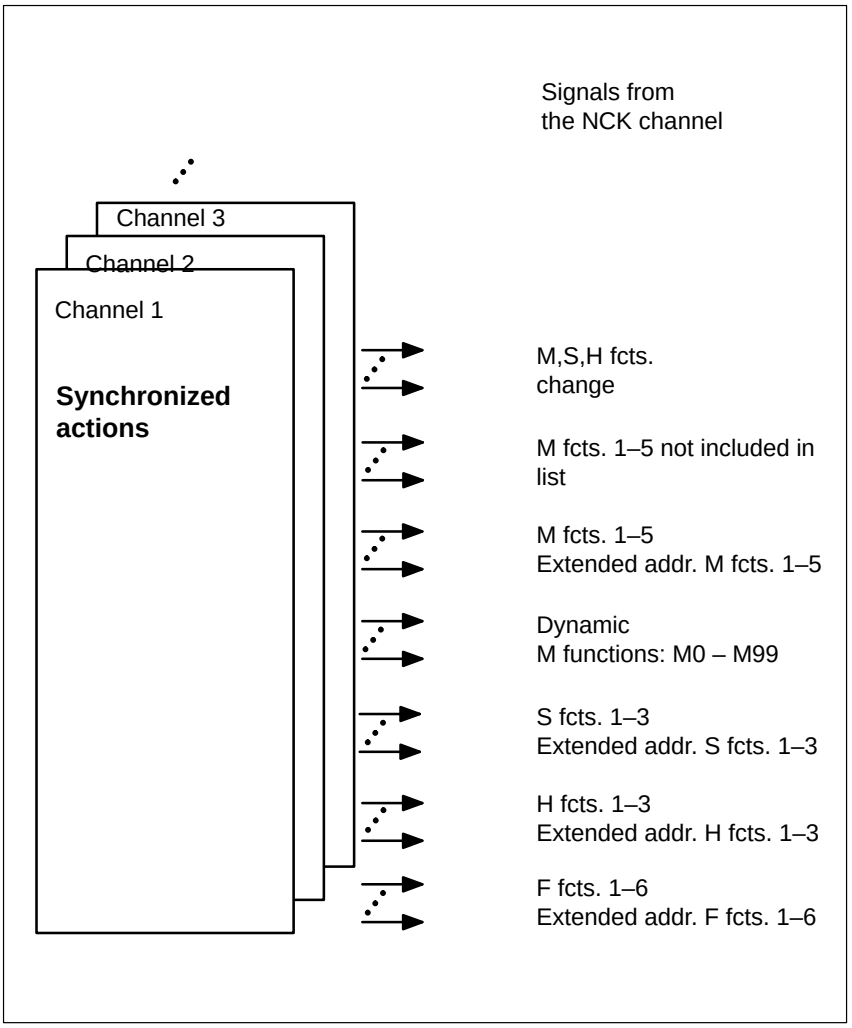


Fig. 5-1 PLC interface signals for synchronized actions

The signals generated as synchronized actions using auxiliary function output correspond to the signals described in

References: /FB/, H2, Output of Auxiliary Functions to PLC.

Signals to channel

The PLC user program uses signals DB 21 – 30 DBB 300 bit 0 to DB 21 – 30 DBB 307 bit 7
to request that the assigned synchronized actions be disabled. The following applies:
DBB 300 bit 0 corresponds to the first modal synchronized action (ID=1/IDS=1) and DBB 307 bit 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.

Signals from channel

The channel uses signals DB 21 – 30 DBB 308 bit 0 to DB 21 – 30 DBB 315 bit 7
to indicate to the PLC user program which synchronized actions can be disabled by the PLC. The following applies:
DBB 308 bit 0 corresponds to the first modal synchronized action (ID=1/IDS=1) and DBB 315 bit 7 to the 64th modal synchronized action (ID=64/IDS=64).

Disable all synchronized actions

The global signal

DB21–30 DBB1 bit 2

disables **all** modal/static synchronized actions, which are not protected.

Disable selected synchronized actions

DB 21 – 30 DBX280.1
From the synchronized actions that are marked as possible to disable in DB 21 – 30 DBB 308 bit 0 to DB 21 – 30 DBB 315 bit 7, disable the ones in DB 21 – 30 DBB 300 bit 0 to DB 21 – 30 DBB 307 bit 7 that have been marked to be disabled by means of the set bit.

Synchronized actions disabled

DB 21 – 30 DBX281.1
The NCK confirms that the requested synchronized actions have been disabled.

6 Examples

6.1 Examples of conditions in synchronized actions

Path distance from end of block

Axial clearance 10 mm or less from the end of the block (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 distance 20 mm or more after the end of the block in the basis coordinate system:
...WHEN $AC_PLTBB >= 20 DO ...

Condition with function in comparison

Actual value for axis y in the MKS greater than 10 x sinusoidal of the 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 then be reset again in order to permit a new start.
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 system variables that can be read in synchronized actions, which appears in

References: /PGA1/, List Manual, System Variables

describes the full range of quantities that can be evaluated in the conditions of synchronized actions.

6.2 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 **Remarks**

```
N610 ID=1 WHENEVER $AA_IM[Z]>$SA_OSCILL_REVERSE_POS1[Z]
 DO $AC_MARKER[1]=0

                                        ;Whenever the current position of the oscillating axis
                                        ;in the machine coordinate system is
                                        ;less     than the start of reversal area 2,
                                        ;then     set the axial override of the
                                        ;         infeed axis to 0

N620 ID=2 WHENEVER $AA_IM[Z]<$SA_OSCILL_REVERSE_POS2[Z]–6
DO $AA_OVR[X]=0 $AC_MARKER[0]=0

                                        ;Whenever the current position of the oscillating axis
                                        ;in the machine coordinate system
                                        ; is the same as reverse position 1,
                                        ; then    set the axial override of the
                                        ;         oscillating axis to 0
                                        ; and     set the axial override of the
                                        ;         infeed axis to 100% (canceling the
                                        ;         previous synchronized
                                        ;         action)

N630 ID=3 WHENEVER $AA_IM[Z]==$SA_OSCILL_REVERSE_POS1[Z]
DO $AA_OVR[Z]=0 $AA_OVR[X]=100

                                        ;Whenever the distance–to–go of the part infeed
                                        ;is equal to 0,
                                        ; then    set the axial override of the oscillating
                                        ;         axis to 100% (canceling the previous
                                        ;         synchronized action!)

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 actual position of the oscillating axis ;in the
                                        workpiece coordinate system
                                        ; is equal to the reversal position 1,
                                        ; then    set the axial override of the
                                        ;         oscillating axis to 100%
                                        ; and     set the axial override of the
                                        ;         feed axis to 0 (canceling the
                                        ;         second synchronized
                                        ;         action once)
```

```
N670 ID=7 WHEN $AA_IM[Z]==$SA_OSCILL_REVERSE_POS1[Z]
 DO $AA_OVR[Z]=100 $AA_OVR[X]=0
```

Setting data whose value may change during machining (e.g. through an operator input or synchronized action) must be programmed with **$$S**...:

Example: Oscillation from synchronized actions with alteration of oscillation position via operator interface

```
N610 ID=1 WHENEVER $AA_IM[Z]>$$SA_OSCILL_REVERSE_POS1[Z] DO $AC_MARKER[1]=0
                                        ;Whenever the current position of the oscillating axis
                                        ;in the machine coordinate system is
                                        ;less     than the start of reversal area 2,
                                        ;then     set the axial override of the
                                        ;         infeed axis to 0
N620 ID=2 WHENEVER $AA_IM[Z]<$$SA_OSCILL_REVERSE_POS2[Z]–6
DO $AA_OVR[X]=0 $AC_MARKER[0]=0
                                        ;Whenever the actual position of the oscillating axis
                                        ;in the machine coordinate system
                                        ; 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
                                        ;         feed axis to 100% (canceling the
                                        ;         previous synchronized
                                        ;         action)
N630 ID=3 WHENEVER $AA_IM[Z]==$$SA_OSCILL_REVERSE_POS1[Z]
DO $AA_OVR[Z]=0 $AA_OVR[X]=100
                                        ;Whenever the distance–to–go of the part infeed
                                        ;is equal to 0,
                                        ; then     set the axial override of the
                                        ;         oscillating axis to 100%
                                        ;         (canceling the previous
                                        ;         synchronized action!)
N640 ID=4 WHENEVER $AA_DTEPW[X]==0
DO $AA_OVR[Z]=100 $AC_MARKER[0]=1 $AC_MARKER[1]=1
N650 ID=5 WHENEVER $AC_MARKER[0]==1 DO $AA_OVR[X]=0
N660 ID=6 WHENEVER $AC_MARKER[1]==1 DO $AA_OVR[X]=0
                                        ;If       the actual position of the oscillating axis
                                        ;         in the workpiece coordinate system
                                        ; is equal to the reversal position 1,
                                        ; then     set the axial override of the
                                        ;         oscillating axis to 100%
                                        ; and      set the axial override of the
                                        ;         feed axis to 0 (canceling the
                                        ;         second synchronized
                                        ;         action once)
N670 ID=7 WHEN $AA_IM[Z]==$$SA_OSCILL_REVERSE_POS1[Z]
DO $AA_OVR[Z]=100 $AA_OVR[X]=0
```

6.3 Examples of adaptive control

General procedure

The following examples use the polynomial evaluation function SYNFCT().

1. Representation of relationship between input value and output value (real-time variables in each case)
2. Definition of this relationship as polynomial with limitations
3. With position offset: Setting of MD and SD
   – MD 36750: $AA_OFF_MODE
   – SA 43350: $SA_AA_OFF_LIMIT (optional)
4. Activation of the control in a synchronized action

6.3.1 Clearance control with variable upper limit

Example of polynomial with dyn. upper limit

For the purpose of clearance control, the upper limit of the output ($AA_OFF, override value in axis V) is varied as a function of the spindle override (analog input 1). The upper limit for polynomial 1 is varied dynamically as a function of analog input 2.
Polynomial 1 is defined directly via system variables:

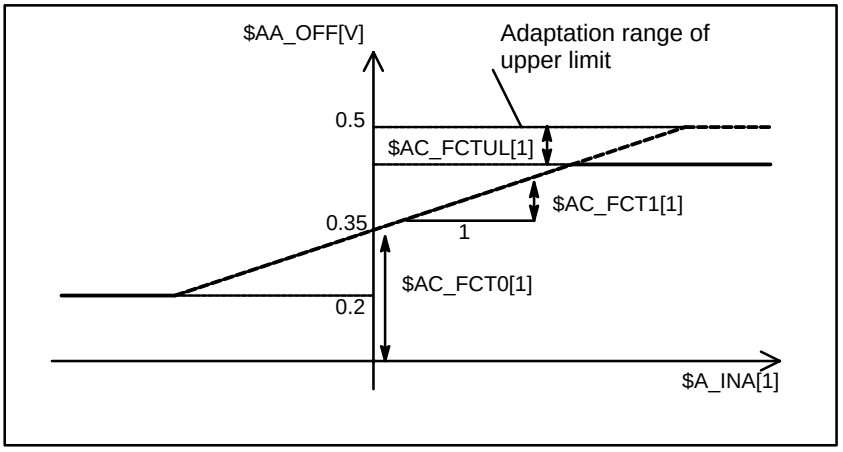


Fig. 6-1 Clearance control with variable upper limit

```
$AC_FCTLL[1]=0.2                    ; Lower limit
$AC_FCTUL[1]=0.5                    ; Init. upper limit value
$AC_FCT0[1]=0.35                    ; zero crossover a0
$AC_FCT1[1]=1.5 EX–5                ; gradient a1
STOPRE                              ; refer to the following note
...
ID=1 DO $AC_FCTUL[1]=$A_INA[2]*0.1+0.35    ;upper limit
                                    ; dynamically  adapt using
                                    ; analog input 2, no condition
ID=2 DO SYNFCT(1, $AA_OFF[V], $A_INA[1])
                                    ; clearance control using superimposition
                                    ; no condition
...
```

Note

When system variables are applied in the part program, STOPRE must be programmed to ensure block-synchronous writing. The following has the same meaning as the notation above to define a polynomial:
FCTDEF(1,0.2, 0.5, 0.35, 1.5EX–5).

6.3.2 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].

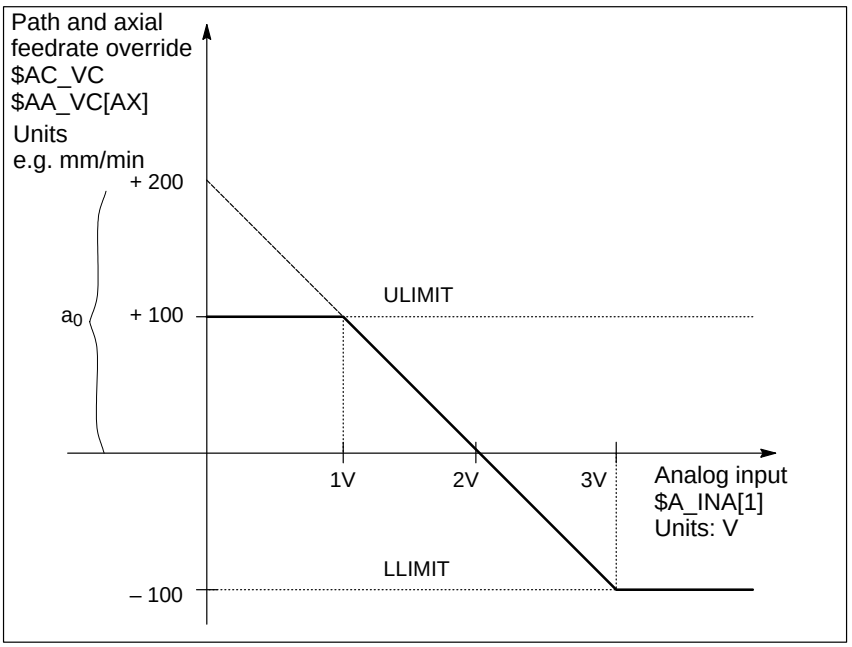


Fig. 6-2 Diagram illustrating adaptive control

Determination of coefficients:

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

$a_1 = -\frac{100\ mm}{1\ min \cdot 1\ V}$

$a_1 = -100 \Rightarrow$ control constant, lead

$a_0 = -(-100) \cdot 2 = 200$

$a_2 = 0$ (not a square component)

$a_3 = 0$ (not a cubic 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, 200, –100, 0, 0)
```

The Adaptive Control can be switched–in with the following synchronized actions:
For the axis feed:

```
ID = 1 DO SYNFCT(1, $AA_VC[X], $A_INA[1])
```

or for the path feed:

```
ID = 2 DO SYNFCT(1, $AC_VC, $A_INA[1])
```

6.3.3 Control velocity as a function of normalized path

Multiplicative adaptation

The normalized path is applied as an input quantity: $AC_PATHN.
0: At block beginning
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.

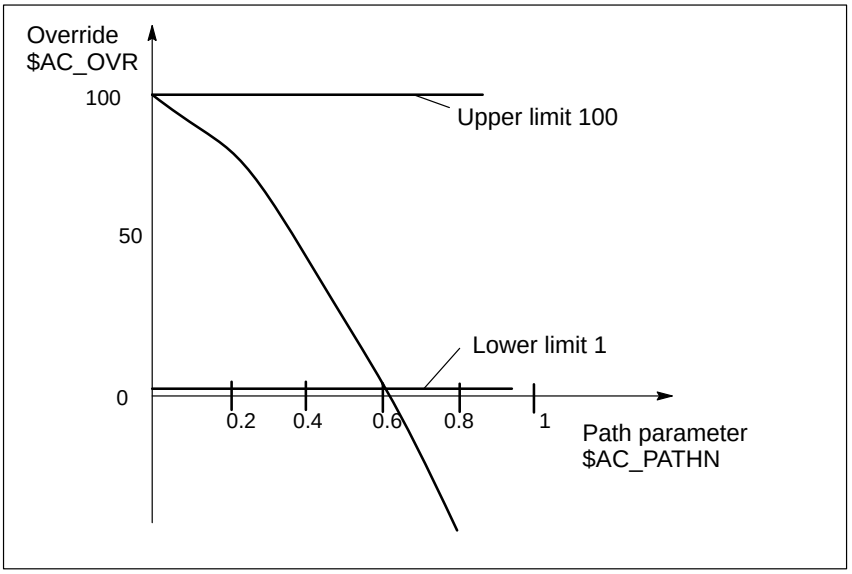


Fig. 6-3 Regulate velocity continuously

Polynomial 2:
Lower limit: 1
Upper limit: 100
a_0: 100
a_1: –100
a_2: –100
a_3: not used

With these values, the polynomial definition is as follows:

```
FCTDEF(2, 1, 100, 100, –100, –100)
; activates the changing override depending on the path distance:

ID= 1 DO SYNFCT(2, $AC_OVR, $AC_PATHN)
G01 X100 Y100 F1000
```

6.4 Monitoring of a safety clearance between two axes

Task The axes X1 and X2 are used for two independently controlled transport devices to load and unload workpieces.
In order to avoid collisions, a safety clearance must be maintained between these two transport devices.
If the safety clearance is fallen below, then axis X2 is braked. This interlocking applies until Axis X1 has again exited the safety range.
If axis X1 continues to approach axis X2 and if a closer safety barrier is fallen below, then it traverses to a safety position.

NC language	Remarks
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

6.5 Store execution times in R parameters

Task Store the execution time for part program blocks starting at R parameter 10.

Program	Remarks
	; The example is as follows **without** symbolic
	; programming:
IDS=1 EVERY $AC_TIMEC==0 DO $AC_MARKER[0] = $AC_MARKER[0] + 1	
	; Advance R parameter pointer on block change
IDS=2 DO $R[10+$AC_MARKER[0]] = $AC_TIME	
	; Write current time of block start in each case
	; to R parameter
	; The example is as follows **with** symbolic
	; programming:
DEFINE INDEX AS $AC_MARKER[0]	; Agreements for symbolic programming
IDS=1 EVERY $AC_TIMEC==0 DO INDEX = INDEX + 1	; advance R parameter pointer on block change
IDS=2 DO $R[10+INDEX] = $AC_TIME	
	; Write current time of block start in each case
	; to R parameter

6.6 ”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.

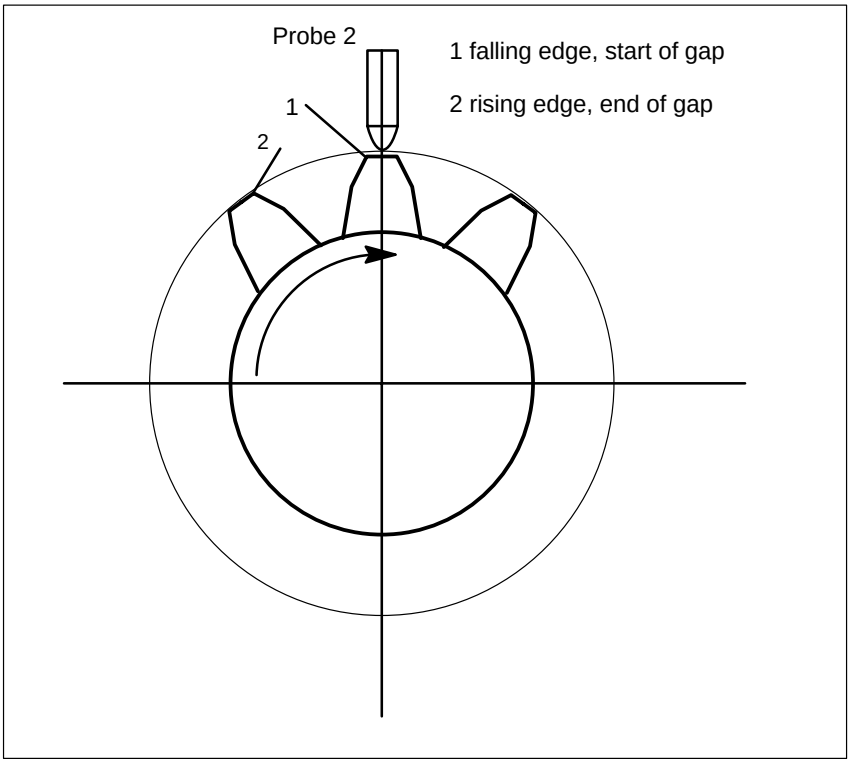


Fig. 6-4 Diagrammatic representation of measurement of gaps between gear teeth

```
%_N_MEAC_MITTEN_MPF

;Measure using rotary axis B (BACH) with display of difference between measured values


;****** Define local user variables ***
N1 DEF INT TOOTH COUNT              ; Enter number of gear wheel teeth
N5 DEF REAL HYS_POS_EDGE            ; Hysteresis positive edge probe
N6 DEF REAL HYS_NEG_EDGE            ; Hysteresis negative edge probe

********* Define code name for synchronization marker **********
define M_TEETH as $AC_MARKER[1]     ; ID marker for computing: Neg/pos edge per tooth
define Z_MW as $AC_MARKER[2]        ; ID counter MW, read out FIFO
define Z_RW as $AC_MARKER[3]        ; ID counter MW, calculate tooth gaps
```

```
;****** Input value for MEASUREGEARWHEEL *******
N50 TEETHNO=26                          ; enter number of gear teeth that have to be measured
N70 HYS_POS_EDGE = 0.160                ; hysteresis, positive edge, probe
N80 HYS_NEG_EDGE = 0.140                ; hysteresis negative edge, probe

start:                                  ;******* Assign variables ************************
R1=0                                    ; ID2 computation result, gap dimension
R2=0                                    ; ID2 computation result, at all gaps
R3=0                                    ; contents of the first element read–in
R4=0                                    ; R4 corresponds to a tooth clearance
R5=0                                    ; gap position calculated, final result
R6=1                                    ; switch–in ID 3 BACH with MOV
R7=1                                    ; switch–in ID 5 MEAC
M_TEETH=TEETHNO*2           ; calculate ID, neg./pos. edge per tooth
Z_MW=0                                  ; ID counter MW, read–out FIFO to tooth number
Z_RW=2                                  ; ID counter, calculate difference, tooth gap
R13=HYS_POS_EDGE                        ; hysteresis in the arithmetic register
R14=HYS_NEG_EDGE                        ; hysteresis in the arithmetic register

;******* Move, measure, calculate axis **********
N100 MEAC[BACH]=(0)                     ; Reset measurement task
;Reset FIFO1[4] variables and ensure a defined measurement trace
N105 $AC_FIFO1[4]=0                     ; Reset FIFO1
STOPRE

****** Read out FIFO until no. of teeth reached *****
; If FIFO1 is not yet empty and not all teeth have been measured, relocate measured value from FIFO variable
; to synchronized action parameter and increase measured value counter

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 available, start to calculate. Calculate gap dimension ONLY and total gap
; increment calculated value counter by 2

ID=2 WHENEVER (Z_MW>=Z_RW) AND (Z_RW<M_TEETH)
        DO $R1=($AC_PARAM[–1+Z_RW]–$R13)–($AC_PARAM[–2+Z_RW]–$R14) Z_RW=Z_RW+2 $R2=$R2+$R1

; ***** Activate axis BACH as endlessly turning rotary axis with MOV *********
WAITP(BACH)
ID=3 EVERY $R6==1 DO  MOV[BACH]=1  FA[BACH]=1000            ; activate
ID=4 EVERY $R6==0 and ($AA_STAT[BACH]==1) DO MOV[BACH]=0    ; deactivate

; Measure in succession, store in FIFO 1, MT2 neg, MT2 pos edge
; the distance between 2 teeth falling edge ... rising edge, probe 2, is measured
N310 ID=5 WHEN $R7==1 DO MEAC[BACH]=(2, 1, –2, 2)
N320 ID=6 WHEN (Z_MW>=M_ZAEHNE) DO MEAC[BACH]=(0)           ; abort measurement
M00
STOPRE


; ******* Fetch FIFO values and store ***
N400 R3=$AC_PARAM[0]                    ; Content of first element to be read in
                                        ; Reset FIFO1[4] variable and ensure
                                        ; a defined measurement trace for next measurement task

N500 $AC_FIFO1[4]=0


; ****** Calculate difference between individual teeth
N510 R4=R2/(NO.TEETH)/1000              ; R4 corresponds to an average distance between teeth
                                        ; ”/1000” division is omitted in later SW versions
```

```
; ******* Calculate center position **********
N520 R3=R3/1000 ; First measurement position converted to degrees
N530 R3=R3 MOD 360 ; First measurement point modulo
N540 R5=(R3–R14)+(R4/2) ; Calculate gap position
M00
stopre
R6=0                                ; deactivate rotation of BACH axis
gotob start
M30
```

6.7 Axis couplings via synchronized actions

6.7.1 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  Master axis movement on/off: MOV BACH
N7 R5=36000                     ; BACH feedrate/min
N8 STOPRE

;**** Define periodic table No. 4 using polynomial segments ****
N10 CTABDEF (YGEO,XGEO,4,1)
N16 G1 F1200  XGEO=0.000 YGEO=0.000    ; approach initial positions
N17  POLY PO[XGEO]=(79.944,3.420,0.210) PO[YGEO]=(24.634,0.871,–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*****


; Traverse master axis and coupled axis in rapid mode to initial position
N80 G0 BACH=0 CACH=0            ; Channel axis names
N50 LEADOF(CACH,BACH)           ; Existing coupling OFF if necessary


N235 ;******* Activation of coupled motion for axis CACH *******
N240 WAITP(CACH)                ; Synchronize axis with channel
N245 ID=1 EVERY $R1==1 DO LEADON(CACH, BACH, 4)     ; Couple using table 4
N250 ID=2 EVERY $R1==0 DO LEADOF(CACH, BACH)        ; Deactivate coupling

N265 WAITP(BACH)

N270 ID=3 EVERY $R2==1 DO MOV[BACH]=1  FA[BACH]=R5  ; Turn master axis endlessly at feedrate in R5
N275 ID=4 EVERY $R2==0 DO MOV[BACH]=0               ; Stop master axis

N280 M00
N285 STOPRE
N290 R1=0                       ; Deactivate coupling condition
N295 R2=0                       ; Deactivate condition for master axis rotation
N300 R5=180                     ; New feedrate for BACH
N305 M30
```

6.7.2 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 inter-relationship 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 coupling. It controls:
– the compensatory motion of axis XACH via table 2 and
– software axis CASW via table 1.

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.

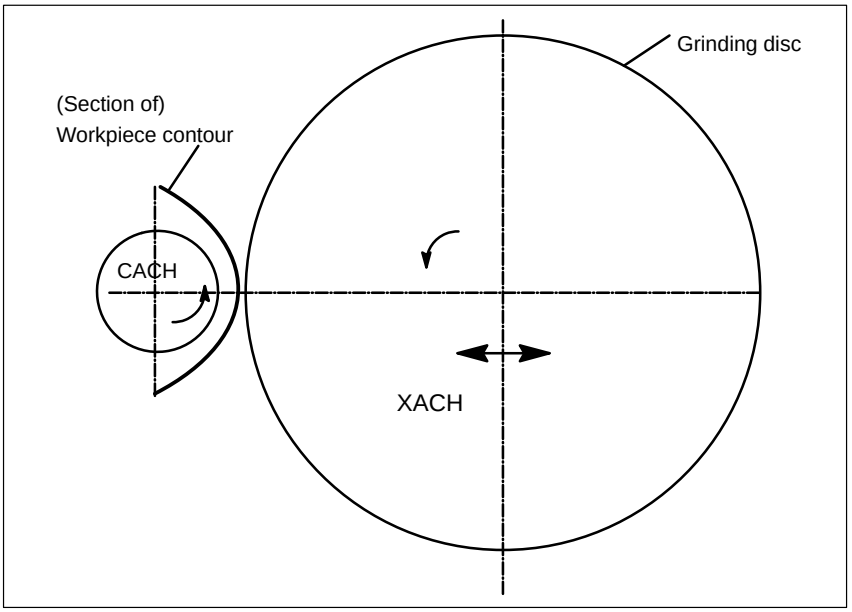


Fig. 6-5 Diagrammatic representation of non-circular contour grinding

```
%_N_CURV_TABS_SPF
PROC CURV_TABS
N160    ; ********** Table 1 Define override ******
N165 CTABDEF(CASW,CACH,1,1)              ; Periodic table 1
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    ; **** Table 2 Define linear compensatory motion of XACH ******
CTABDEF(YGEO,XGEO,2,1)                    ; Periodic table 2
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
; Coupled axis grouping for non–circular machining

; XACH is the infeed axis of the grinding wheel
; CACH is the workpiece axis as a rotary axis and master axis
; Application: Grind non–round contour
; Table 1 emulates the override for axis CACH as a function of the position of CACH
; XGEO axis superimposed with handwheel feed for scratching

N100 DRFOF                          ; de–select handwheel superimposition
N200 MSG("select DRF,  (handwheel 1 active) and select INCREMENT.== handwheel superimposition ACTIV")
N300 M00
N500 MSG()                          ; Reset message
N600 R2=1                           ; LEADON Table 2,  switch–in with ID=3/4  CACH to XACH
N700 R3=1                           ; LEADON Table 1, switch–in with ID=5/6  CACH to CASW, override
N800 R4=1                           ; endlessly rotating rotary axis CACH, start with ID=7/8
N900 R5=36000                       ; FA[CACH]  endlessly rotating rotary axis speed

N1100 STOPRE
N1200                               ; *********  Set axes and master axis to FA *******
                                    ; move axis, master axis and following axis to the initial position
N1300 G0 XGEO=0 CASW=10 CACH=0
N1400 LEADOF(XACH,CACH)             ; coupling OUT XACH equalization motion
N1500 LEADOF(CASW,CACH)             ; coupling OUT CASW override table

N1600 CURV_TABS                     ; Subprogram with definition of tables


N1700                               ; ******* Activate LEADON compensatory motion XACH *******
N1800 WAITP(XGEO)                   ; synchronize axis with channel
N1900 ID=3 EVERY $R2==1 DO LEADON(XACH,CACH,2)
N2000 ID=4 EVERY $R2==0 DO LEADOF(XACH,CACH)

N2100                               ; ************ Activate LEADON CASW override table ****
N2200 WAITP(CASW)
N2300 ID=5 EVERY $R3==1 DO LEADON(CASW,CACH,1)      ; CTAB coupling ON master axis CACH
N2400 ID=6 EVERY $R3==0 DO LEADOF(CASW,CACH)        ; CTAB coupling OFF master axis CACH


N2500                               ; ** Control CASH override from position CASW with ID 10 *
N2700 ID=11 DO $$AA_OVR[CACH]=$AA_IM[CASW]          ; Assign "axis position" CASW to OVR CACH

N2900 WAITP(CACH)

N3000 ID=7 EVERY $R4==1 DO MOV[CACH]=1 FA[CACH]=R5  ; Start as endlessly turning rotary axis
N3100 ID=8 EVERY $R4==0 DO MOV[CACH]=0              ; Stop as endlessly turning rotary axis

N3200 STOPRE
N3300 R90=$AA_COUP_ACT[CASW]                        ; Status of coupling for CASW for checking
N3400 MSG("activate CASW override table with LEADON"<<R90<<", go to END with NC START")
```

```
N3500 M00                ; ********** NC STOP **************
N3600 MSG()
N3700 STOPRE             ; Preprocessing stop
N3800 R1=0               ; Stop with ID=2 CASW axis as endlessly turning 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 endlessly turning 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).
- Switchover between tables if the cams have different contours, e.g. for inlet and outlet.
  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 2.4.7.

6.7.3 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. WCS
X1 axis: Machine axis of extruded material, MCS
Y axis: Axis in which cutting tool ”tracks” the extruded material

For the purpose of this example, it is assumed that the cutting tool infeed is 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 **Remarks**

```
%_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                                            ; starting position Y axis
N500 R1=1                                             ; starting condition for belt axis
N600 LEADOF(Y,X)                                      ; clear a possibly 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 the table definition
N1200 PRESETON(X1,0)                                  ; PRESET to begin
N1300 Y=R6 G0                                         ; starting pos. Y axis

                                                      ; Axis is linear
N1400 ID=1 EVERY $AA_IW[X]>$R3 DO PRESETON(X1,0)      ; PRESET after length R3, PRESTON only permitted
                                                      ; with WHEN and EVERY
                                                      ; New start after material parting
N1500 WAITP(Y)

N1800 ID=6 EVERY $AA_IM[X]<10  DO LEADON(Y,X,1)       ; Couple Y to X via Table 1 when X < 10
N1900 ID=10 EVERY  $AA_IM[X]>$R3–30  DO LEADOF(Y,X)   ; decouple when X > 30 before start of cutting length
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
```

6.8 Technology cycles position spindle

Application

Interacting with the PLC program, the spindle which initiates a tool change must be:

- Traversed to an initial position or
- Positioned at a specific point at which the tool to be inserted is also located.

See Subsections 2.4.13, 2.6.1.

Coordination

The PLC and NCK are coordinated by means of the common data that are provided in SW version 4 and later (see Subsection 2.3.11)

- $A_DBB[0] 1 traverse to initial position
- $A_DBB[1] 1 traverse to target position
- $A_DBW[1] Position value +/– , PLC calculates the shortest route.

Synchronized actions

%_N_MAIN_MPF

```
...
IDS=1 EVERY $A_DBB[0]==1 DO  NULL_POS          ; If $A_DBB[0] is set by PLC, traverse to initial position
IDS=2 EVERY $A_DBB[1]==1 DO  ZIEL_POS          ; If $A_DBB[1] is set by PLC, traverse spindle to
                                               ; position stored in $A_DBW[1]
 ...
```

Technology cycle NULL_POS

%_N_NULL_POS_SPF

```
PROC NULL_POS
SPOS=0                                         ; Move drive for tool change into initial position
$A_DBB[0]=0                                    ; Initial position executed in NCK
```

Technology cycle ZIEL_POS

%_N_ZIEL_POS_SPF

```
PROC TARGET_POS
SPOS=IC($A_DBW[1])                             ; Traverse spindle to position value that has been stored
                                               ; in $A_DBW[1] by the PLC, incremental dimension
$A_DBB[1]=0                                    ; Target position executed in NCK
```

6.9 Synchronized actions in the TC/MC area

Introduction

The following figure shows the schematic structure of a tool-changing cycle.

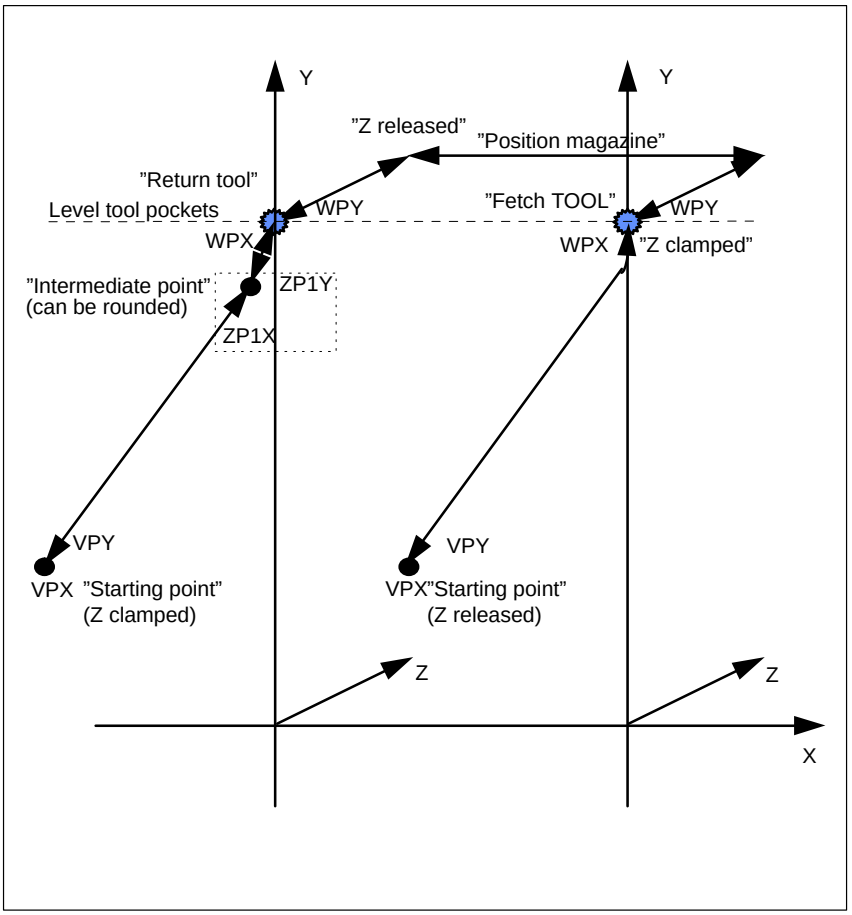


Fig. 6-6 Schematic sequence for tool-changing cycle

Flowchart

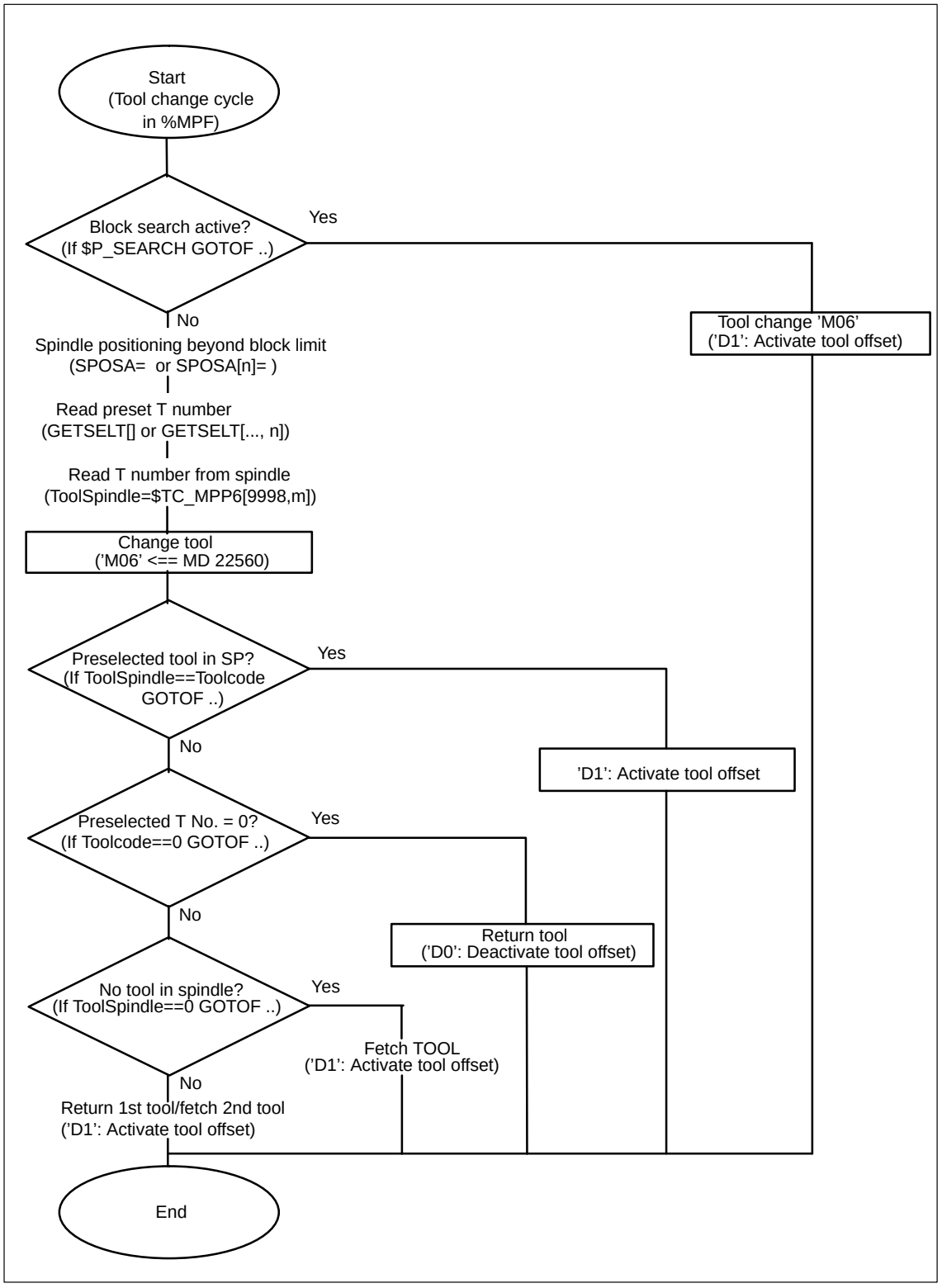


Fig. 6-7 Flowchart for tool-changing cycle

NC program **Remarks**

%_N_WZW_SPF

```
;$PATH=/_N_SPF_DIR
N10 DEF INT ToolCode, ToolSpindle
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          ; Marker to = 1 when MagAxis traversed
N25 ID=4 WHENEVER $A_IN[10]==TRUE DO $AC_MARKER[2]=1         ; Marker to = 1 when MagAxis traversed
N30 IF $P_SEARCH GOTOF tc_preprocessing                      ; Block search active? –>
 N35 SPOSA=0 D0
N40 GETSELT(ToolCode)                                        ; Read preselected T No.
N45 ToolSpindle=$TC_MPP6[9998,1]                             ; Read tool in spindle
N50 M06
N55 IF ToolSpindle==ToolCode GOTOF tool_in_spindle  IF ToolCode==0 GOTOF return1 IF ToolSpindle==0 GOTOF
fetch1

;*****Fetch tool and store*****
store1fetch1:
N65 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[1]=1             ;if MagAxis moves, marker = 1
N70 G01 G40 G53 G64 G90 X=Magazine1VPX Y=Magazine1VPY Z=Magazine1ZclampedF70000 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             ; MagAxis moves, interrogate
N85 WHENEVER $AC_MARKER[1]==0 DO $AC_OVR=0                   ; override=0 if axis is not moved
N90 WHENEVER $AA_STAT[C2]<>4 DO $AC_OVR=0                    ; override=0 if MagAxis not in fine pos
N95 WHENEVER $AA_DTEB[C2]>0 DO $AC_OVR=0                     ; override=0 if distance to go MagAxis > 0
N100 G53 G64 X=Magazine1ZP1X Y=Magazine1ZP1Y F60000
N105 G53 G64 X=Magazine1WPX Y=Magazine1WPY F60000
N110 M20                                                     ; release tool
N115 G53 G64 Z=MR_Magazine1Zreleased 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=Magazine1Zclamped F40000
N145 M18                                                     ; clamp tool
N150 WHEN $AC_PATHN<10 DO M=QU(150) M=QU(121)                ; condition always fulfilled
N155 G53 G64 X=Magazine1VPX Y=Magazine1VPY F60000 D1 M17

;*****Store tool*****
store1:
N160 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[1]=1
N165 G01 G40 G53 G64 G90 X=Magazine1VPX Y=Magazine1VPY Z=Magazine1Zclampled 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
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=Magazine1ZP1X Y=Magazine1ZP1Y F60000
N200 G53 G64 X=Magazine1WPX Y=Magazine1WPY F60000
N205 M20                                                     ; release tool
N210 G53 G64 Z=Magazine1Zreleased F40000
N215 G53 G64 X=Magazine1VPX Y=Magazine1VPY F60000 M=QU(150) M=QU(121) D0 M17

;*****Fetch tool*****
fetch1:
N220 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[2]=1
N225 G01 G40 G53 G64 G90 X=Magazine1VPX Y=Magazine1VPY Z=Magazine1Zreleased F70000 M=QU(120)
M=QU(123) M=QU(9)
N230 G53 G64 X=Magazine1WPX Y=Magazine1WPY 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=Magazine1Zclamped F40000
N265 M18                                                                    ; clamp tool
N270 G53 G64 X=Magazine1VPX Y=Magazine1VPY F60000 M=QU(150) M=QU(121) D1 M17

;*****Tool in spindle *****
tool_in_spindle:
N275 M=QU(121) D1 M17

;*****Block search*****
tc_preprocessing:
N280 STOPRE
N285 D0
N290 M06
N295 D1 M17
```

Space for your notes

7 Data Fields, Lists

7.1 Interface signals

DB number	Bit, byte	Name	Refer-ence
Channel–specific			
21–30	280.1	Disable modal synchronized actions acc. to DBX 300.0–307.7	
21–30	300.0 –	Disable modal synchronized actions acc. to DBX 300.0–307.7, acknowledgment from NCK	
21–30	300.0 –	Modal synchronized actions ID or IDS 1 –	
21–30	307.7	64 disabled. Request to NCK channel	
21–30	308.0 –	Modal synchronized actions ID or IDS 1 –	
21–30	315.7	64 can be disabled. Message from NCK.	

7.2 Machine data

Number	Identifier	Name	Refer-ence
General ($MN_ ...)			
11110	AUXFU_GROUP_SPEC	Auxiliary function group specification	H2
11500	PREVENT_SYNACT_LOCK	Protected synchronized actions	
18860	MM_MAINTENANCE_MON	Activate recording of maintenance data	
Channel-specific ($MC_ ...)			
21240	PREVENT_SYNACT_LOCK_CHAN	Protected synchronized actions for channel	
28250	MM_NUM_SYNC_ELEMENTS	Number of elements for expressions in synchron-ized actions	
28252	MM_NUM_FCTDEF_ELEMENTS	Number of FCTDEF elements	
28254	MM_NUM_AC_PARAM	Number of $AC_PARAM parameters	
28255	MM_BUFFERED_AC_PARAM	Storage location for $AC_PARAM (SW 6.3 and later)	
28256	MM_NUM_AC_MARKER	Number of $AC_MARKER markers	
28257	MM_BUFFERED_AC_MARKER	Storage location for $AC_MARKER (SW 6.3 and later)	
28258	MM_NUM_AC_TIMER	Number of $AC_TIMER time variables	
28260	NUM_AC_FIFO	Number of $AC_FIFO1, $AC_FIFO2, ... variables	
28262	START_AC_FIFO	Store FIFO variables from R parameter	
28264	LEN_AC_FIFO	Length of $AC_FIFO ... FIFO variables	

Channel-specific ($MC_ ...)			
28266	MODE_AC_FIFO	FIFO processing mode	

Axis-specific ($MA_ ...)			
30450	IS_CONCURRENT_POS_AX	Concurrent positioning axis	P2
32060	POS_AX_VELO	Initial setting for positioning axis velocity	P2
32070	CORR_VELO	Axial velocity for handwheel, ext. ZO, cont. dressing, clearance control (from SW3)	H1
32074	FRAME_OR_CORRPOS_NOTALLOWED	Effectiveness of frames and tool length offset	
32920	AC_FILTER_TIME	Filter smoothing time constant for Adaptive Control (SW2 and later)	
33060	MAINTENANCE_DATA	Configuration, recording maintenance data	
36750	AA_OFF_MODE	Effect of value assignment for axial override for synchronized actions (SW3 and later)	
37200	COUPLE_POS_TOL_COARSE	Threshold value for ”Coarse synchronism”	S3
37210	COUPLE_POS_TOL_FINE	Threshold value for ”Fine synchronism”	S3
Setting data ($SA_ ...)			
43300	ASSIGN_FEED_PER_REV_SOURCE	Rotational feedrate for positioning axes/spindles	V1
43350	AA_OFF_LIMIT	Upper limit of offset value for $AA_OFF clearance control	
43400	WORKAREA_PLUS_ENABLE	Working area limitation in pos. direction	A3

7.3 Alarms

For more detailed information about the alarms, see
References: /DA/, ”Diagnostics Manual”
or, for systems with MMC 101/102, the online help.

Space for your notes

Index

Symbols

A

B

C

D

E

F

G

I

J

L

M

W

Space for your notes

To
SIEMENS AG
A&D MC BMS
P.O. Box 3180

D-91050 Erlangen

(Phone: +49 (0)180 5050-222 [Hotline]
Fax: +49 (0)9131 98-63315 [Documentation]
mailto:motioncontrol.docu@siemens.com)

Suggestions

Corrections

For Publication/Manual:

SINUMERIK 840D sl
SINUMERIK 840D/840Di/810D
Description of Functions
Synchronized Actions

Manufacturer/Service Documentation

From

Name

Company/Dept.

Address

Phone: /

Fax: /

Description of Functions

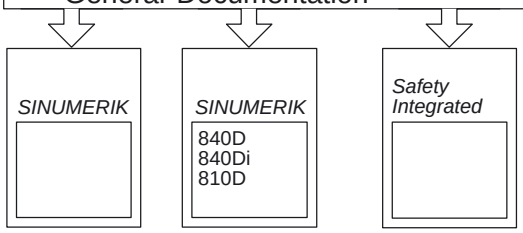
Order No.: 6FC5 397–5BP10–0BA0
Edition: 08/2005

Should you come across any printing errors when reading this publication, please notify us on this sheet.
Suggestions for improvement are also welcome.

Suggestions and/or corrections

Overview of documentation SINUMERIK 840D/840D/810D (08/2005)

General Documentation

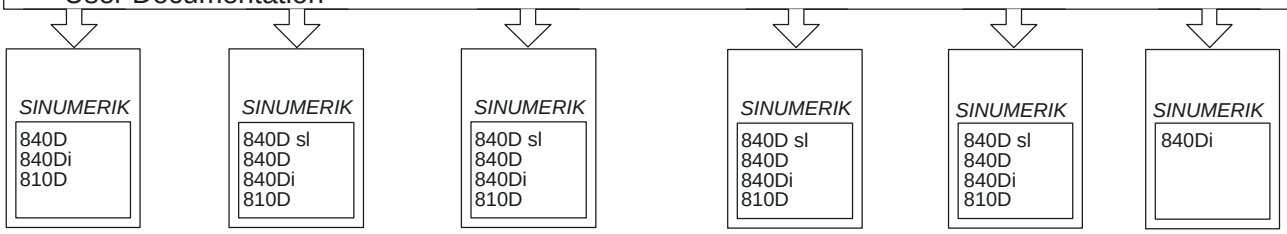


Brochure

Catalog NC 60 ***)**

Safety Integrated Application Manual

User Documentation



Operating Manual
– **HMI Embedded *)**
– **ShopMill**
– **ShopTurn**
– HT6

Operating Manual
– **HMI Advanced *)**
– Product Brief

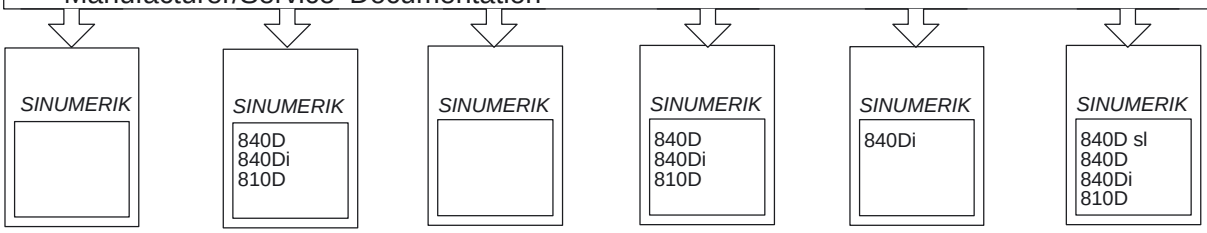
Programming Manual
– Fundamentals ***)**
– **Advanced *)**
– Product Brief
– Lists System Variable
– ISO Turning/Milling

Programming Manual
– **Cycles**
– **Measuring Cycles**

Diagnostics Manual *)

System Overview

Manufacturer/Service Documentation



Configuring (HW) ***)**
– 840D
– 810D

Manual Operator **Components *)**

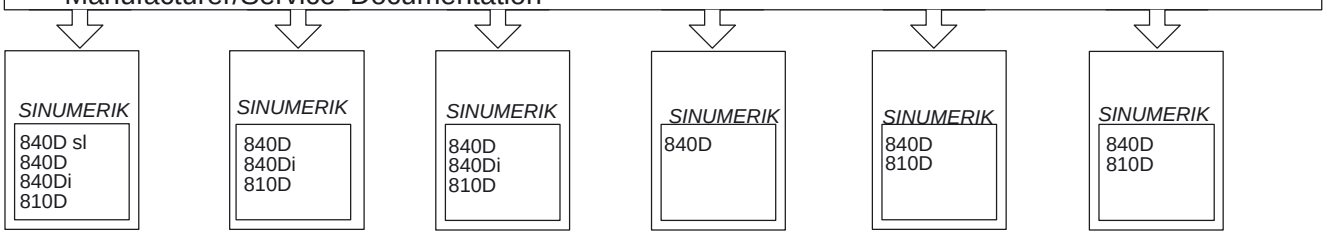
Commissioning Manual *)
– 840D
– 810D
– HMI

Description of Functions
– **ShopMill**
– **ShopTurn**

Commissioning Manual

Lists *)
– 1st Part
– 2nd Part

Manufacturer/Service Documentation



Description of Functions
– Basic Machine ***)**
– Extended Functions
– Special Functions
– **Synchronized Actions**
– **ISO Dialects**
– **EMC Guidelines**

Description of Functions
– **Linear Motor**
– **Tool Mgmt.**
– **Hydraulics Modules**
– **Analog Modules**

Description of Functions
– **Remote Diagnosis**
– **@Event**

Description of Functions
Safety Integrated

Configuring Kit
– **Operator Interface OP030**
– **HMI Embedded**

MCIS
– Computer Link
– Tool Data Information System
– NC Data Mgmt.
– NC Data Transfer
– Tool Data Communication

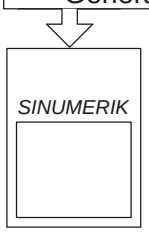
Electronic Documentation

DOCONCD ***)**
DOCONWEB

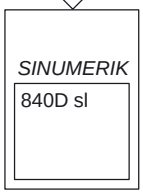
***)** This documents are a minimum requirement

Overview of documentation SINUMERIK 840D sl (08/2005)

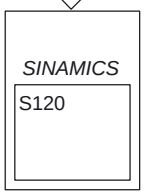
General Documentation



Brochure

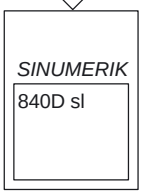


Catalog NC 61 ***)**

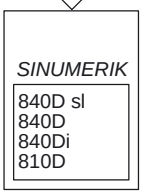


Catalog
D21.2 Servo Control ***)**

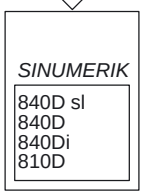
User Documentation



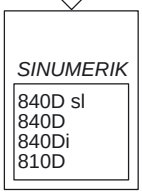
Operating Manual
– **HMI Embedded *)**
– **ShopMill**
– **ShopTurn**



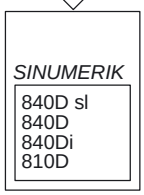
Operating Manual
– **HMI Advanced *)**
– Product Brief



Programming Manual
– Fundamentals ***)**
– Advanced ***)**
– Product Brief
– Lists System Variable
– ISO Turning/Milling

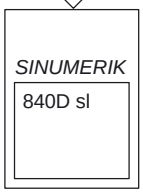


Programming Manual
– **Cycles**
– **Measuring Cycles**

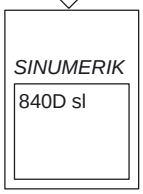


Diagnostics Manual *)

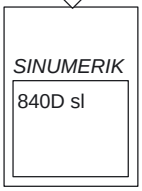
Manufacturer/Service Documentation



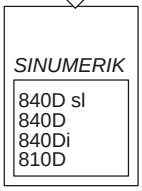
Manual
NCU *)



Manual
Operator Components *)

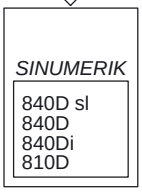


Commissioning Manual CNC ***)**
– 1st Part NCK, PLC, Drive
– 2nd Part HMI
– 3rd Part ShopMill
– 4th Part ShopTurn
– 5th Part Basic Software

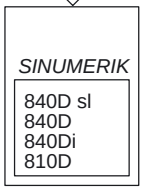


Lists *)
– 1st Part
– 2nd Part

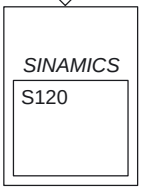
Manufacturer/Service Documentation



Description of Functions
– Basic Machine ***)**
– Extended Functions
– Special Functions

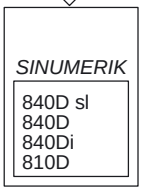


Description of Functions
_ **Synchronized Actions**
– **ISO Dialects**



Description of Functions
Drive Functions

Description of Functions
Safety Integrated



EMC Guidelines

Electronic Documentation

DOCONCD ***)**
DOCONWEB

***)** These documents are a minimum requirement