SIEMENS

SINUMERIK

SINUMERIK 840D sl / 828D Synchronized actions

Function Manual

Valid for

Controls
SINUMERIK 840D sl / 840DE sl
SINUMERIK 828D

Software	Version
CNC software	4.5 SP2

03/2013
6FC5397-5BP40-3BA1

Legal information

Warning notice system

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

⚠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
indicates that minor personal injury can result if proper precautions are not taken.

NOTICE
indicates that property damage can result if proper precautions are not taken.

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

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

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

Trademarks

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

Disclaimer of Liability

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

Siemens AG
Industry Sector
Postfach 48 48
90026 NÜRNBERG
GERMANY

Order number: 6FC5397-5BP40-3BA1
Ⓟ 04/2013 Technical data subject to change

Preface

SINUMERIK documentation

The SINUMERIK documentation is organized in the following categories:

- General documentation
- User documentation
- Manufacturer/service documentation

Additional information

You can find information on the following topics at www.siemens.com/motioncontrol/docu:

- Ordering documentation/overview of documentation
- Additional links to download documents
- Using documentation online (find and search in manuals/information)

Please send any questions about the technical documentation (e.g. suggestions for improvement, corrections) to the following address:

docu.motioncontrol@siemens.com

My Documentation Manager (MDM)

Under the following link you will find information to individually compile OEM-specific machine documentation based on the Siemens content:

www.siemens.com/mdm

Training

For information about the range of training courses, refer under:

- www.siemens.com/sitrain

  SITRAIN - Siemens training for products, systems and solutions in automation technology
- www.siemens.com/sinutrain

  SinuTrain - training software for SINUMERIK

FAQs

You can find Frequently Asked Questions in the Service&Support pages under Product Support. http://support.automation.siemens.com

SINUMERIK

You can find information on SINUMERIK under the following link:

www.siemens.com/sinumerik

Target group

This publication is intended for:

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

Benefits

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

Standard version

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

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

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

Technical Support

You will find telephone numbers for other countries for technical support in the Internet under http://www.siemens.com/automation/service&support

Contents

1 Brief description

General

A synchronized action consists of a series of related statements within a part program that is called cyclically in the interpolation cycle synchronously to the machining blocks.

A synchronized action is essentially divided into two parts, the optional condition and the obligatory action part. The time at which the actions are executed can be made dependent on a specific system state using the condition part. The conditions are evaluated cyclically in the interpolation cycle. The actions are then a reaction to user-definable system states. Their execution is not bound to block limits.

Furthermore, the validity of the synchronized action (non-modal, modal or static) and the frequency of the execution of the actions (once, repeatedly) can be defined.

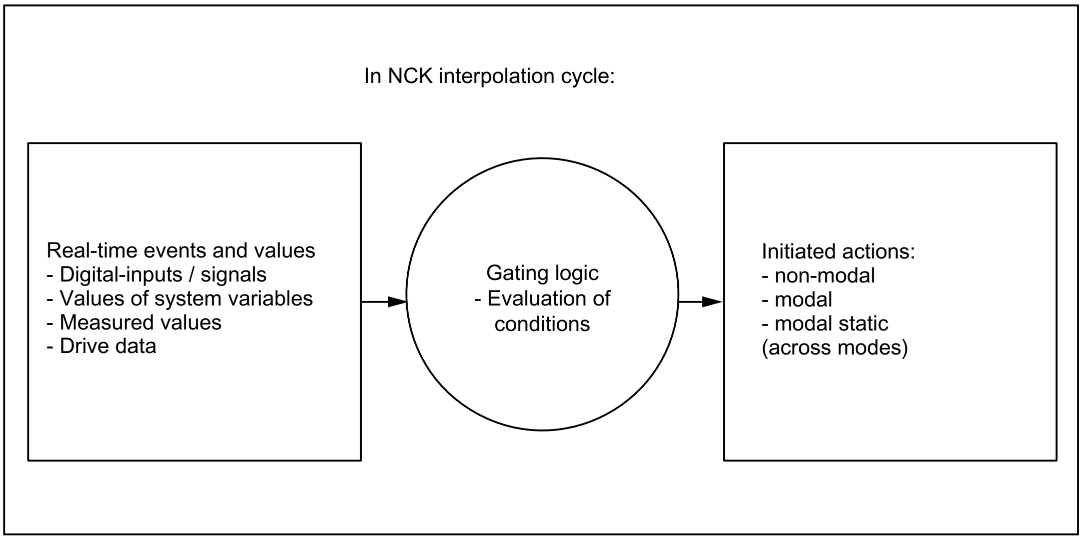
Examples of permissible actions

- Output of auxiliary functions to PLC
- Writing and reading of main run variables
- Traversing of positioning axes
- Activation of synchronous procedures, such as:
  - Read-in disable
  - Delete distance-to-go
  - End preprocessing stop
- Activation of technology cycles
- Calculation of function values
- Tool offsets
- Activating/deactivating couplings
- Measuring
- Enabling/disabling of synchronized actions

Examples of non-permissible actions

- Traversing of path axes

Schematic diagram of synchronized actions



Detailed description 2

2.1 Definition of a synchronized action

A synchronized action is defined in a block of a part program. Any further commands that are not part of the synchronized action, must not be programmed within this block.

Components of a synchronized action

A synchronized action consists of the following components:

Condition part				Action part		
Optional	Optional					
Validity, ID no. (Page 12)	Frequency (Page 13)	Optional G function (Page 14)	Condition (Page 15)	Keyword	Optional: G function (Page 16)	Actions (Page 16)
--- 1)	--- 1)	`G...`	`Logical expression`	`DO`	`G...`	`Action 1` ... `Action n`
`ID=<no.>`	`WHENEVER`					
`IDS=<no.>`	`FROM`					
	`WHEN`					
	`EVERY`					
1) Not programmed						

Syntax

Examples:

1. `DO <action 1...n>`
2. `<frequency> [<G function>] <condition> DO <action 1...n>`
3. `ID=<no.> <frequency> [<G function>] <condition> DO <action 1...n>`
4. `IDS=<no.> <frequency> [<G function>] <condition> DO <action 1...n>`

2.2 Components of synchronized actions

2.2.1 Validity, identification number (ID, IDS)

Validity

The validity defines when and where the synchronized action will be processed:

Validity	Meaning

"No specification"	Non-modal synchronized action
	A non-modal synchronized action applies:
	• As long as the main run block following the synchronized action is active
	• Only in the AUTOMATIC mode
	Example:
	The synchronized action from N10 is effective as long as N20 is active.
	`N10 WHEN $A_IN[1]==TRUE DO $A_OUTA[1]=10`
	`N20 G90 F1000 X100`

ID=<ID number>	Modal synchronized action A modal synchronized action applies: • Until the part program has been completed • Only in the AUTOMATIC mode Range of values: 1 ... 255 Example: `N20 ID=1 EVERY $A_IN[1]==TRUE DO $A_OUTA[1]=10`
IDS=<ID number>	Static synchronized action A static synchronized action applies: • In all operating modes for an unlimited period of time Range of values: 1 ... 255 Example: `N30 IDS=1 EVERY $A_IN[1]==TRUE DO $A_OUTA[1]=10`

Note

Static synchronized actions

Static synchronized actions (`IDS`) can be defined in an ASUB and activated at any time by activation of the ASUB via the PLC user program.

Identification numbers

If several synchronized actions are to be active in parallel in a channel, their identification numbers `ID`/`IDS` must be different. Synchronized actions with the same identification number replace each other within a channel.

Sequence of execution

Modal and static synchronized actions are executed in the order of their identification numbers `ID`/`IDS`.

Non-modal synchronized actions are executed after execution of the modal synchronized actions in the order of their programming.

Coordination via part programs and synchronized actions

Synchronized actions can be coordinated via part programs and synchronized actions based on the identification numbers `ID`/`IDS` (see Section "Coordination via part program and synchronized action (LOCK, UNLOCK, RESET, CANCEL) (Page 110)").

Coordination via PLC

Synchronized actions with identification numbers `ID`/`IDS` in the range from 1 to 64 can be coordinated via the NC/PLC interface from the PLC user program (see Section "Coordination via PLC (Page 111)").

2.2.2 Frequency (WHENEVER, FROM, WHEN, EVERY)

The frequency specifies how often the condition is queried and, when the condition is fulfilled, how often the action should be executed. The frequency is part of the condition.

Frequency	Meaning
"No specification"	If no frequency is specified, the action is executed cyclically in every interpolation cycle.
WHENEVER	If the condition is fulfilled, the action is executed cyclically in every interpolation cycle.
FROM	After the condition has been fulfilled once, the action is executed cyclically in every interpolation clock cycle for as long as the synchronized action is active.
WHEN	If the condition is fulfilled, the action is executed once and then the condition is no longer checked.
EVERY	At every state change of the condition from FALSE to TRUE (rising edge), the action is executed once.

See also

Technology cycles (Page 104)

2.2.3 G function (condition)

Defined initial state

With regard to the part program sequence, synchronized actions can be executed at any time depending on fulfillment of the condition. It is therefore recommended that the measuring system (inch or metric) be defined in a synchronized action **before** the condition and/or in the action part. This generates a defined initial position for the evaluation of the condition and the execution of the action, irrespective of the current part program state.

G functions

The following G functions are permitted:

- `G70` (Inch dimensions for geometric specifications (lengths))
- `G71` (Metric dimensions for geometric specifications (lengths))
- `G700` (Inch dimensions for geometric and technological specifications (lengths, feedrate))
- `G710` (Metric dimensions for geometric and technological specifications (lengths, feedrate))

Note

No other G functions are permitted in synchronized actions except `G70`, `G71`, `G700` and `G710`.

Validity

A G function programmed in the condition part also applies for the action part even if no G function has been programmed in the action part itself.

A G function programmed in the action part only applies within the action part.

2.2.4 Condition

Execution of the action can be made dependent on the fulfillment of a condition. As long as the synchronized action is active, the condition is checked cyclically in the interpolation cycle. If no condition is specified, the action is executed cyclically in every interpolation cycle.

All operations that return a truth value (TRUE/FALSE) as the result can be programmed as a condition:

- Comparisons of system variables with constants
- Comparisons of system variables with system variables
- Comparisons of system variables with results of arithmetic operations
- Linking of comparisons through Boolean expressions

Examples

Comparisons

Program code

```
ID=1 WHENEVER $AA_IM[X] > $$AA_IM[Y] DO ...
ID=2 WHENEVER $AA_IM[X] > (10.5 * SIN(45)) DO ...
```

Boolean operations

Program code

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

See also

Reading and writing (Page 17)

Examples of conditions in synchronized actions (Page 123)

System variables for synchronized actions (Page 17)

2.2.5 G function (action)

Defined initial state

With regard to the part program sequence, synchronized actions can be executed at any time depending on fulfillment of the condition. Therefore, it is advisable to define the required measuring system (inch or metric) in the action part in a synchronized action. This generates a defined initial position for the execution of the action, irrespective of the current part program state.

G functions

The following G functions are permitted:

- `G70` (Inch dimensions for geometric specifications (lengths))
- `G71` (Metric dimensions for geometric specifications (lengths))
- `G700` (Inch dimensions for geometric and technological specifications (lengths, feedrate))
- `G710` (Metric dimensions for geometric and technological specifications (lengths, feedrate))

Validity

A G function programmed in the condition part also applies for the action part even if no G function has been programmed in the action part itself.

A G function programmed in the action part only applies within the action part.

2.2.6 Action (DO)

The action part of a synchronized action is initiated with the keyword `DO`.

One or more actions can be programmed in the action part. These are executed when the appropriate condition is fulfilled. If several actions are programmed in one synchronized action, they are all executed in the same interpolation cycle.

Example:

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

Program code
`WHEN $AA_IM[Y] >= 35.7 DO M135 $A_OUT[1]=1`

Technology cycle

A technology cycle can be called as an action. See Section "Technology cycles (Page 104)".

2.3 System variables for synchronized actions

The system variables of the NCK are listed in the "System Variables" Parameter Manual with their respective properties. System variables that can be read or written in synchronized actions, are marked with an "X" in the corresponding line (Read or Write) of the "SA" (synchronized action) column.

Note

System variables used in synchronized actions are implicitly read and written synchronous to the main run.

References

A comprehensive description of the system variables listed in this function manual can be found in:

- System Variables Parameter Manual

2.3.1 Reading and writing

The reading and writing of variables is performed in the main run in synchronized actions with a few exceptions. Exceptions are:

- User-defined variables: LUD, GUD
- Machine data: $M...
- Setting data: $S...
- R parameters: R<number> or R[<index>]

These variables are already read and written during the preprocessing.

System variables

Generally, all system variables that can be used in synchronized actions are read/written in the main run. These system variables are marked with an "X" in the "Read" and/or "Write" line of the "SA" (synchronized action) column in the "System Variables" Parameter Manual.

References:
System Variables Parameter Manual

System of the identifiers

The identifiers of the system variables that are read/written in the main run have the following system:

$A...	Current main run data
$V...	Servo data
$R...	R parameters to be read/written in the main run
$$M...	Machine data to be read/written in the main run
$$S...	Setting data to be read/written in the main run

2.3.2 Operators and arithmetic functions

Operators

Arithmetic operators

System variables of the REAL and INT type can be linked by the following operators:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division, caution: INT/INT = REAL
DIV	Integer division, caution: INT/INT = INT
MOD	Modulo division (only for type INT) supplies remainder of an INT division Example: 3 MOD 4 = 3

Note

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

Relational operators

Operator	Meaning
==	Equal to
>	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Boolean operators

Operator	Meaning
NOT	NOT
AND	AND
OR	OR
XOR	Exclusive OR

Bit logic operators

Operator	Meaning
B_OR	Bit-by-bit OR
B_AND	Bit-by-bit AND
B_XOR	Bit-by-bit exclusive OR
B_NOT	Bit-by-bit negation

Priority of the operators

The operators have the following priorities for execution in the synchronized action (highest priority: 1):

Prio.	Operators	Meaning
1	NOT, B_NOT	Negation, bit-by-bit negation
2	*, /, DIV, MOD	Multiplication, division
3	+, -	Addition, subtraction
4	B_AND	Bit-by-bit AND
5	B_XOR	Bit-by-bit exclusive OR
6	B_OR	Bit-by-bit OR
7	AND	AND
8	XOR	Exclusive OR
9	OR	OR
10	<<	Concatenation of strings, result type STRING
11	==, <>, <, >, >=, <=	Relational operators

Note

It is strongly recommended that the individual operators are clearly prioritized by setting parentheses "(…)" when several operators are used in an expression.

Example of a condition with an expression with several operators:

Program code

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

Arithmetic functions

Operator	Meaning
Sin()	Sine
COS()	Cosine
TAN()	Tangent
ASIN()	Arc sine
ACOS()	Arc cosine
ATAN2()	Arc tangent 2
SQRT()	Square root
ABS()	Absolute value
POT()	2nd power (square)
TRUNC()	Integer component The accuracy for comparison commands can be set using TRUNC
ROUND()	Round to an integer
LN()	Natural logarithm
EXP()	Exponential function

A detailed description of the functions can be found in:
References
Programming Manual, Job Planning; Section "Flexible NC programming" ff.

Indexing

The index of a system variable of type "Array of …" can in turn be a system variable. The index is also evaluated in the main run in the interpolation cycle.

Example

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

Restrictions

- It is not permissible to nest indices with further system variables.
- The index must not be formed via preprocessing variables. The following example is therefore **not** permitted since $P_EP is a preprocessing variable:
  $AC_PARAM[1] = $P_EP[$AC_MARKER[0]]

2.3.3 Type conversions

An implicit type conversion is performed between the following data types for value assignments and parameter transfers with different data types:

- REAL
- INT
- BOOL

Note

Conversion REAL to INT

For the conversion from REAL to INT, a decimal place value ≧ 0.5 rounded up to the next higher integer. For a decimal place value < 0.5, rounding is to the next lower integer. Behavior in accordance with the `ROUND` function.

If the REAL value is outside the INT value range, an alarm is displayed and a conversion is not performed.

Conversion from REAL or INT to BOOL

- Value <> 0 → TRUE
- Value == 0 → FALSE

Examples

Conversion: **INT** $AC_MARKER → **REAL** $AC_PARAM

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

Conversion: **REAL** $AC_PARAM → **INT** $AC_MARKER

```
Program code
$AC_PARAM[1]=561.0
ID=1 WHEN TRUE DO $AC_MARKER[1] = $AC_PARAM[1]
```

Conversion: **INT** $AC_MARKER → **BOOL** $A_OUT

```
Program code
$AC_MARKER[1]=561
ID=1 WHEN $A_IN[1] == TRUE DO $A_OUT[0]=$AC_MARKER[1]
```

Conversion: **REAL** $R401 → **BOOL** $A_OUT

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

Conversion: **BOOL** $A_OUT → **INT** $AC_MARKER

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

Conversion: **BOOL** $A_OUT → **REAL** $R10

```
Program code
WHEN $A_IN[3] == TRUE DO $R10 = $A_OUT[3]
```

2.3.4 Marker/counter ($AC_MARKER)

The $AC_MARKER[<index>] variables are channel-specific arrays of system variables for use as markers or counters.

Data type:	INT (integer)
<Index>:	Array index: 0, 1, 2, ... (max. number - 1)

Number per channel

The maximum number of $AC_MARKER variables per channel can be set via the machine data:

MD28256 $MC_MM_NUM_AC_MARKER = <maximum number>

Storage location

The storage location of the $AC_MARKER variables can be defined channel-specifically via the machine data:

MD28257 $MC_MM_BUFFERED_AC_MARKER = <value>

Value	Storage location
0	Dynamic memory (default setting)
1	Static memory

Note

Data backup and memory space

- The $AC_MARKER variables created in the static memory can be saved channel-specifically via the data backup. Data block: _N_CH<channel number>_ACM
- Please ensure that sufficient memory is available in the selected memory area. An array element requires 4 bytes of memory space.

Reset behavior

The reset behavior depends on the storage location of the $AC_PARAM variables:

- Dynamic memory: Initialization with the value "0"

Static memory: Retention of the current value

2.3.5 Parameters ($AC_PARAM)

The $AC_PARAM[<index>] variables are channel-specific arrays of system variables for use as general buffers.

Data type:	REAL
<Index>:	Array index: 0, 1, 2, ... (max. number - 1)

Number per channel

The maximum number of $AC_PARAM variables per channel can be set via the machine data:

MD28254 $MC_MM_NUM_AC_PARAM = <maximum number>

Storage location

The storage location of the $AC_PARAM variables can be defined channel-specifically via the machine data:

MD28255 $MC_MM_BUFFERED_AC_PARAM = <value>

Value	Storage location
0	Dynamic memory (default setting)
1	Static memory

Note

Data backup and memory space

- The $AC_PARAM variables created in the static memory can be saved channel-specifically via the data backup. Data block: _N_CH<channel number>_ACP
- Please ensure that sufficient memory is available in the selected memory area. An array element requires 4 bytes of memory space.

Reset behavior

The reset behavior depends on the storage location of the $AC_PARAM variables:

- Dynamic memory: Initialization with the value "0"
- Static memory: Retention of the current value

2.3.6 R parameters ($R)

Whether R-parameters are treated as preprocessing or main run variables depends on whether they are written with or without $ characters. In principle, the notation is freely selectable. For use in synchronized actions, R parameters should be used as main run variables, i.e. with $ characters:

- $R[<index>]
- $R<number>

Data type:	REAL
<Index>:	Array index: 0, 1, 2, ...
<Number>:	Number of the R parameter: 0, 1, 2, ...

The notations with index or number are equivalent.

Parameterizable number per channel

The maximum number of R parameters per channel can be set via the machine data:

MD28254 $MC_MM_NUM_AC_PARAM = <maximum number>

Reset behavior

R parameters are saved persistently in the static memory of the NC. Therefore, R parameters retain their values with all reset types:

- Power on reset
- NC reset
- End of part program reset

Example

Value assignment to R10 in the action part of the synchronized action and subsequent evaluation in the part program

Program code	Comment
`WHEN $A_IN[1]==1 DO $R[10]=$AA_IM[Y]`	`; Assignment`
`G1 X100 F150`	
`STOPRE`	
`IF R[10] > 50 ...`	`; Evaluation in the part program`

2.3.7 Machine and setting data ($$M, $$S)

Reading and writing MD and SD

When machine and setting data is used in synchronized actions, a distinction must be made as to whether this remains unchanged during the execution of the synchronized action, or is changed through parallel processes.

Data that remains **unchanged** can already be read or written by the NC during **preprocessing**.

Data that is **changed** can only be read or written by the NC during the **main run**.

Data access during preprocessing

Machine and setting data that can already be read and written in synchronized actions during preprocessing, is programmed with the same identifiers as in the part program: $M ... or $S ...

Program code
`; The reversal position of the Z axis $SA_OSCILL_REVERSE_POS2[Z]`
`; remains unchanged over the entire machining period`
`ID=2 WHENEVER $AA_IM[z]<$SA_OSCILL_REVERSE_POS2[Z]-6 DO $AA_OVR[X]=0`

Data access during the main run

An additional "$" is added as prefix for machine and setting data that may only be read or written in synchronized actions during the main run: $$M… or $$S…

Program code
`; The reversal position of the Z axis $SA_OSCILL_REVERSE_POS2[Z]`
`; can be changed by operator input at any time`
`ID=1 WHENEVER $AA_IM[z] < $$SA_OSCILL_REVERSE_POS2[Z] DO $AA_OVR[X] = 0`

Writing during the main run

The following requirements must be satisfied for writing during the main run:

- The access authorization at the time of writing must be sufficient for writing.
- The machine or setting data must have the property "Effective immediately".

Program code
`; The switching position of the SW cam $SN_SW_CAM_ ... must,`
`; depending on the current setpoint of the X axis in WCS $AA_IW[X],`
`; only be written during the main run`
`ID=2 WHEN $AA_IW[X] > 10 DO $$SN_SW_CAM_PLUS_POS_TAB_1[0] = 20`
` $$SN_SW_CAM_MINUS_POS_TAB_1[0]=20`

A complete overview of the properties of the machine and setting data can be found in:

References

- Parameter Manual: Lists (Book 1)
- Parameter Manual: Detailed Machine Data Description

2.3.8 Timer ($AC_TIMER)

The $AC_TIMER[<index>] variables are channel-specific arrays of system variables.

Data type:	REAL
<Index>:	Array index: 0, 1, 2, ... (max. number - 1)
Unit:	Seconds

Number per channel

The maximum number of $AC_TIMER variables per channel can be set via the machine data:

MD28258 $MC_MM_NUM_AC_TIMER = <maximum number>

Function

Starting

A timer is started by assigning a value ≥ 0:

$AC_TIMER[<index>] = <starting value>; with starting value ≥ 0

Incrementing

The value of the timer is incremented by the duration of the set interpolation cycle (MD10071 IPO_CYCLE_TIME) each interpolation cycle.

$AC_TIMER[<index>] += <interpolation cycle>

Stopping

A timer is stopped by assigning a value < 0:

$AC_TIMER[<index>] = <stopping value>; with stopping value < 0

When a stopping value is assigned, only the further incrementing of the timer is stopped. The stopping value is not assigned. After the timer is stopped, the last valid value is retained and can still be read.

Note

The current value of a timer can be be read when the timer is running or stopped.

Example

Output the actual value of the X axis as voltage value via analog output $A_OUTA[3], 500 ms after the detection of digital input $A_IN[1]:

Program code	Comment
`WHEN $A_IN[1] == 1 DO $AC_TIMER[1]=0`	`; Start time, starting value 0`
`WHEN $AC_TIMER[1]>=0.5 DO $A_OUTA[3]=$AA_IM[X] $AC_TIMER[1]=-1`	

2.3.9 FIFO variables ($AC_FIFO)

Special data structures managed by the NC are provided via $AC_FIFO variables within the R parameters. These are organized as ring buffers that work according to the FIFO principle (First In, First Out).

Syntax

```
$AC_FIFO<number>[<index>]
$AC_FIFO[<number>, <index>]
```

Data type:	Corresponds to R parameter: REAL
<Number>:	Number of the $AC_FIFO variable: 1, 2, 3, ... max. number
<Index>:	Array index: 0, 1, 2, ... (max. number - 1)

Meaning of the array indices

In addition to the array elements for the user data, a $AC_FIFO variable also contains several array elements to manage the data. Each individual array element can be accessed via the index.

The array elements with the indices 0 … 5 are used to manage the $AC_FIFO variable:

Index	Meaning
0	Index 0 has the following special meaning: Array element 0 is not accessed with index 0. Write: The "most recent" value is stored in the variable Read: The "oldest" value is read from the variable
1	Write/read: The "oldest" array element is addressed
2	Write/read: The "most recent" array element is addressed
3	Read: Returns the sum of the values of all user data Requirement: See paragraph below "Summation of all user data"
4	Read: Returns the number of the existing data items A $AC_FIFO variable is reset to its initial state with: `$AC_FIFO<number>[4] = 0`
5	Read: Returns the current write index, relative to the beginning of the $AC_FIFO variable

The array elements as of index 6 contain the user data:

Index	Meaning
6	Write/read: The 1st array element for user data is addressed
7	Write/read: The 2nd array element for user data is addressed
n	Write/read: The nth array element for user data is addressed

Note

Overwriting of user data

Because of the ring buffer structure, the oldest user data is overwritten as soon as all free array elements of a $AC_FIFO variable have been assigned.

Configuration

Number per channel

Number of array elements

The maximum number of array elements per $AC_FIFO variable can be set via the machine data:

MD28264 $MC_LEN_AC_FIFO = <maximum number of array elements>

Start of $AC_FIFO variable range

The R parameter as of which the $AC_FIFO variable range is to start, can be set via the machine data:

MD28262 $MC_START_AC_FIFO = <number of the start R parameter>

R parameters above the start R parameter cannot be written in the part program.

Total number of R parameters in the channel

The total number of R parameters in the channel can be set via the machine data:

MD28050 $MC_MM_NUM_R_PARAM = <maximum number>

The number of R parameters in the channel set in the machine data must be at least as large as the number of R parameters required for the $AC_FIFO variables:

<Maximum number> ≥ MD28262 $MC_START_AC_FIFO + MD28260 $MC_NUM_AC_FIFO * (MD28264 $MC_LEN_AC_FIFO + 6)

Summation of all user data

The sum of the values of all user data is only provided via the index [4] when the function has been activated via the machine data:

MD28266 MODE_AC_FIFO, bit 0 = <value>

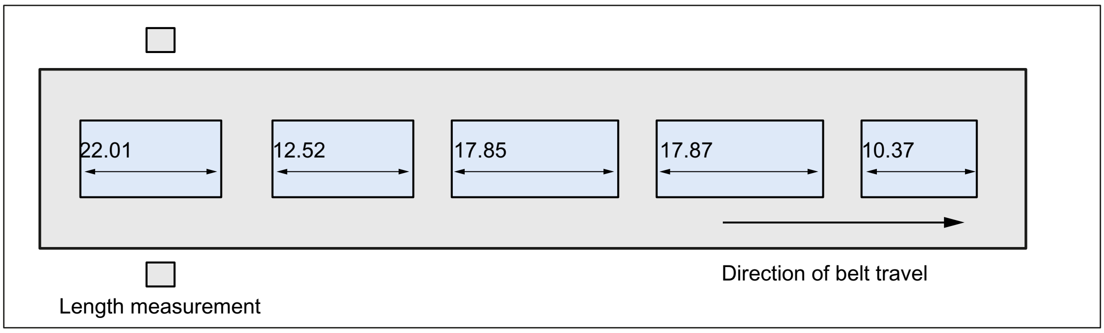
Value	Meaning
0	The sum of the values of all user data is **not** provided
1	The sum of the values of all user data is provided

Storage location and reset behavior

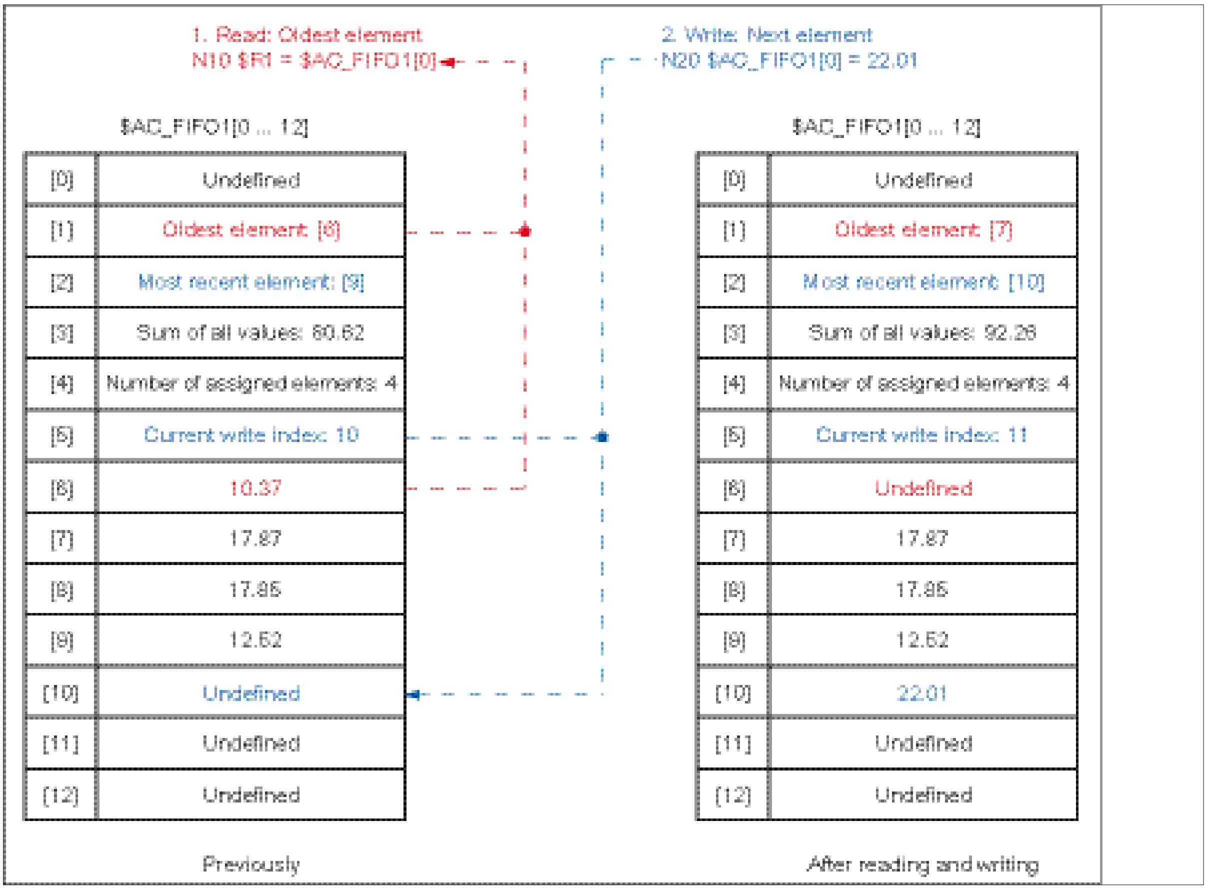
The $AC_FIFO variables are based on the R parameters. The statements made there are therefore also valid for the $AC_FIFO variables. See Section "R parameters ($R) (Page 24)".

Example

Serial determination of the length of workpieces that move past an automatic measuring station on a conveyor belt.



The measurement results are written to or read from the $AC_FIFO1 system variable via synchronized actions.



2.3.10 Path tangent angle ($AC_TANEB)

The angle between the tangent at the end point of the current block and the tangent at the start point of the following block can be read via the channel-specific system variable $AC_TANEB (**T**angent **AN**gle at **E**nd of **B**lock).

Data type: REAL

The tangent angle is always specified positive in the range 0.0 to 180.0°.

If the tangent angle cannot be determined, the value -180.0° is output.

Used only with programmed blocks

It is recommended that the tangent angle only be read for programmed blocks, not for intermediate blocks generated by the system. A distinction can be made via the system variable $AC_BLOCKTYPE:

$AC_BLOCKTYPE == 0 (programmed block)

Example:

Program code

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

2.3.11 Override ($A...OVR)

Current override

Channel-specific override

The path feedrate can be changed via the channel-specific system variable $AC_OVR.

Data type:	REAL
Unit:	%
Range of values:	0.0 to machine data • For **binary**-coded override switch MD12100 $MN_OVR_FACTOR_LIMIT_BIN • For **gray**-coded override switch MD12030 $MN_OVR_FACTOR_FEEDRATE[30]

The system variable $AC_OVR must be written in every interpolation cycle, otherwise the value "100%" is effective.

Channel-specific rapid traverse override

With G0 blocks (rapid traverse), the rapid traverse feedrate can also be influenced via the setting data SD42122 $SC_OVR_RAPID_FACTOR in addition to the system variable $AC_OVR.

Requirement: Release of the rapid traverse override via the user interface.

Axis-specific override

The axial feedrate can be changed via the axis-specific system variable $AA_OVR:

Data type:	REAL
Unit:	%
Range of values:	0.0 to machine data • For **binary**-coded override switch MD12100 $MN_OVR_FACTOR_LIMIT_BIN • For **gray**-coded override switch MD12030 $MN_OVR_FACTOR_FEEDRATE[30]

The system variable $AA_OVR must be written in every interpolation cycle, otherwise the value "100%" is effective.

PLC override

Channel-specific override

The channel-specific override (DB21, ... DBB4) set via the machine control panel can be read via the channel-specific system variable $AC_PLC_OVR:

Data type:	REAL
Unit:	%
Range of values:	0.0 to maximum value

Axis-specific override

The axis-specific override (DB31, ... DBB0) set via the machine control panel can be read via the axis-specific system variable $AA_PLC_OVR:

Data type:	REAL
Unit:	%
Range of values:	0.0 to maximum value

Effective override

Effective channel-specific override

The effective channel-specific override can be read via the channel-specific system variable $AC_TOTAL_OVR:

Data type:	REAL
Unit:	%
Range of values:	0.0 to maximum value

Effective axis-specific override

The effective axis-specific override can be read via the axis-specific system variable $AA_TOTAL_OVR:

Data type:	REAL
Unit:	%
Range of values:	0.0 to maximum value

2.3.12 Capacity evaluation ($AN_IPO ... , $AN/AC_SYNC ... , $AN_SERVO)

The values of the current, maximum and average system utilization due to synchronized actions can be read via the following system variables:

NC-specific system variable	Meaning
$AN_IPO_ACT_LOAD	Current computing time of the interpolator level (incl. synchronized actions of all channels)
$AN_IPO_MAX_LOAD	Longest computing time of the interpolator level (incl. synchronized actions of all channels)
$AN_IPO_MIN_LOAD	Shortest computing time of the interpolator level (incl. synchronized actions of all channels)
$AN_IPO_LOAD_PERCENT	Current computing time of the interpolator level in relation to the interpolator cycle (%)
$AN_SYNC_ACT_LOAD	Current computing time for synchronized actions over all channels
$AN_SYNC_MAX_LOAD	Longest computing time for synchronized actions over all channels
$AN_SYNC_TO_IPO	Percentage share that the synchronized actions have of the total computing time (over all channels)
$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

Channel-specific system variable	Meaning
$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

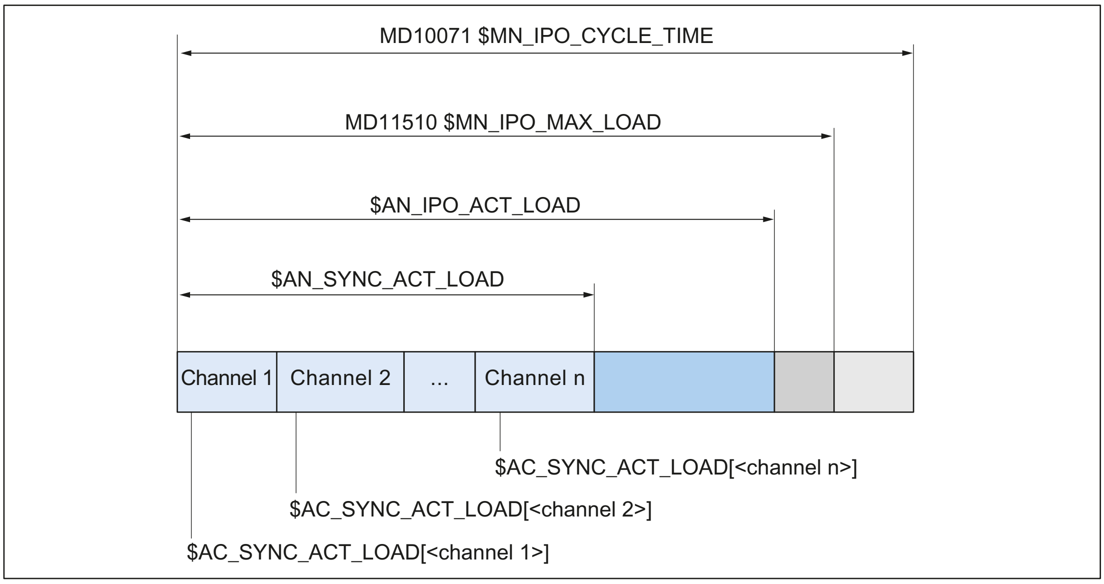


Figure 2-1 Computing time shares of the synchronized actions on the interpolator cycle

Activation

The system variables contain only valid values when the following applies:

MD11510 $MN_IPO_MAX_LOAD > 0 (maximum permissible interpolator utilization)

Note

The variables always contain the values of the previous interpolator cycle.

Overload limit

An overload limit can be set via the following machine data:

MD11510 $MN_IPO_MAX_LOAD = <maximum permissible utilization in %>

If the value set in the machine data is exceeded, the system variable is set:

$AN_IPO_LOAD_LIMIT = TRUE

If the value falls below the set value again, the system variable is reset:

$AN_IPO_LOAD_LIMIT = FALSE

Application

A user-specific strategy to avoid a level overflow can be implemented via the system variable $AN_IPO_LOAD_LIMIT.

Resetting of min./max. values

The following system variables for min./max. values are reset by writing arbitrary values:

System variables	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 computing time of the interpolator level (incl. synchronized actions of all channels)
$AN_IPO_MIN_LOAD	Shortest computing time of the interpolator level (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

Example

Program code	Comment
$MN_IPO_MAX_LOAD=80	; Overload limit
;	
; Initialization of the min./max. values	
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	
;	
; Alarm 63111 when the overload limit is exceeded	
N10 IDS=1 WHENEVER $AN_IPO_LOAD_LIMIT == TRUE DO M4711 SETAL(63111)	
;	
; Alarm 63222 when the computing time share of the	
; synchronized actions over all channels exceeds 30% of the interpolator cycle	
N20 IDS=2 WHENEVER $AN_SYNC_TO_IPO > 30 DO SETAL(63222)	
;	
N30 G0 X0 Y0 Z0	
...	
N999 M30	

2.3.13 Working-area limitation ($SA_WORKAREA_ ...)

Only the activation via the setting data is effective for the traversable command axes in synchronized actions with regard to the programmable working-area limitation `G25`/`G26`:

- $SA_WORKAREA_PLUS_ENABLE
- $SA_WORKAREA_MINUS_ENABLE

Switching the working-area limitation on and off via the commands `WALIMON`/`WALIMOF` in the part program has no effect on the command axes traversable via synchronized actions.

2.3.14 SW cam positions and times ($$SN_SW_CAM_ ...)

The values of the SW cam positions and times can be read and written via the following setting data:

NC-specific setting data	Meaning
$SN_SW_CAM_MINUS_POS_TAB_1[0..7]	Minus cam positions
$SN_SW_CAM_MINUS_POS_TAB_2[0..7]	Minus cam positions
$SN_SW_CAM_PLUS_POS_TAB_1[0..7]	Plus cam positions
$SN_SW_CAM_PLUS_POS_TAB_2[0..7]	Plus cam positions
$SN_SW_CAM_MINUS_TIME_TAB_1[0..7]	Minus cam lead or delay time
$SN_SW_CAM_MINUS_TIME_TAB_2[0..7]	Minus cam lead or delay time
$SN_SW_CAM_PLUS_TIME_TAB_1[0..7]	Plus cam lead or delay time
$SN_SW_CAM_PLUS_TIME_TAB_2[0..7]	Plus cam lead or delay time

Note

The setting of a software cam via synchronized actions must not be performed immediately before the cam is reached. At least three interpolation cycles must be available before the cam is reached.

A detailed description of the "Software cam" function can be found in:

References

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

Examples

```
Program code
; Changing a cam position:
ID=1 WHEN $AA_IW[x] > 0 DO $$SN_SW_CAM_MINUS_POS_TAB_1[0] = 50.0
...
; Changing a lead time
ID=1 WHEN $AA_IW[x] > 0 DO $$SN_SW_CAM_MINUS_TIME_TAB_1[0] = 1.0
```

See also

Machine and setting data ($$M, $$S) (Page 25)

2.3.15 Path length evaluation / machine maintenance ($AA_TRAVEL ... , $AA_JERK ...)

The data of the path length evaluation, e.g. for machine maintenance, can be read via the system variables listed below.

Activation

The activation for the recording of the path length evaluation data is performed via:

MD18860 $MN_MM_MAINTENANCE_MON = 1

The data to be recorded for the specific axis can be selected via the following axis-specific machine data:

MD33060 $MA_MAINTENANCE_DATA[<axis>], bit n = 1

Bit	Meaning
0	Recording of total traversing distance, total traversing time and number of traversing operations of the axis.
1	Recording of total traversing distance, total traversing time and number of traversing operations of the axis at high speed.
2	Recording of total number of axis jerks, the time during which the axis is traversed with jerk and the number of traversing operations with jerk.

System variable

System variable	Meaning	n
$AA_TRAVEL_DIST	Total travel distance: Sum of all set position changes in MCS in [mm] or [deg.]	0
$AA_TRAVEL_TIME	Total travel time: Sum of IPO cycles of set position changes in MCS in [s] (resolution: 1 IPO cycle)	
$AA_TRAVEL_COUNT	Total travel count: A complete machine axis trip is defined by the following succession of states, as based on set position: standstill > traversing > standstill	
$AA_TRAVEL_DIST_HS	Total traversing distance at high axis velocities [1)]	1
$AA_TRAVEL_TIME_HS	Total traversing time at high axis velocities [1)]	
$AA_TRAVEL_COUNT_HS	Total number of traversing operations at high axis velocities [3)]	
$AA_JERK_TOT	Total sum of axis jerks: Sum of all jerk setpoints in [m/s^3] or [$deg./ s^3$]	2
$AA_JERK_TIME	Total travel time with jerk: Sum of IPO cycles from jerk setpoint changes in [s] (solution: 1 IPO cycle)	
$AA_JERK_COUNT	Total number of traversing operations with jerk	
[1)] Actual machine axis velocity ≥ 80% of the maximum parameterized axis velocity (MD32000 MAX_AX_VELO)		

References

For a detailed description of the function, refer to:

Function Manual, Special Functions, Section "Path length evaluation (W6)"

2.3.16 Polynomial coefficients, parameters ($AC_FCT ...)

Function

Polynomials up to the 3rd degree can be defined via the FCTDEF function:

$f(x) = a_0 + a_1 * x + a_2 * x^2 + a_3 * x^3$

Note

The definition must be made in a **part program**.

Syntax

```
FCTDEF(<Poly_No>,<Lo_Limit>,<Up_Limit>,a0,a1,a2,a3)
```

Meaning

Parameter	Meaning
`<Poly_No>`:	Number of the polynomial function
`<Lo_Limit>`:	Lower limit of the function values
`<Up_Limit>`:	Upper limit of the function values
`a0, a1, a2, a3`:	Polynomial coefficient

Note

Polynomial coefficients (a_2, a_3) that are not required can be omitted when programming the `FCTDEF(...)` function.

System variable

Read and write access to polynomial coefficients and parameters is also possible from synchronized actions via the following system variables:

System variable	Meaning
`$AC_FCTLL[<Poly_No>]`:	Lower limit for function value
`$AC_FCTUL[<Poly_No>]`:	Upper limit for function value
`$AC_FCT0[<Poly_No>]`:	a_0
`$AC_FCT1[<Poly_No>]`:	a_1
`$AC_FCT2[<Poly_No>]`:	a_2
`$AC_FCT3[<Poly_No>]`:	a_3
`<Poly_No>`:	The number specified during the definition of the polynomial function (see above: Syntax)

Part program

When writing system variables in the part program, preprocessing stop `STOPRE` must be programmed explicitly for block-synchronous writing.

Note

Block-synchronous writing in the part program

So that the system variables can be written block-synchronously in the part program, the `STOPRE` command (preprocessing stop) must be used after writing the system variables.

Synchronized action

When writing system variables in synchronized actions, they take effect immediately.

Use

The function value f(x) of the polynomial can be used as input value in synchronized actions, e.g. for the following functions:

- "Polynomial evaluation (SYNFCT) (Page 61)"
- "Online tool offset (FTOC) (Page 67)"

Example: Linear dependency

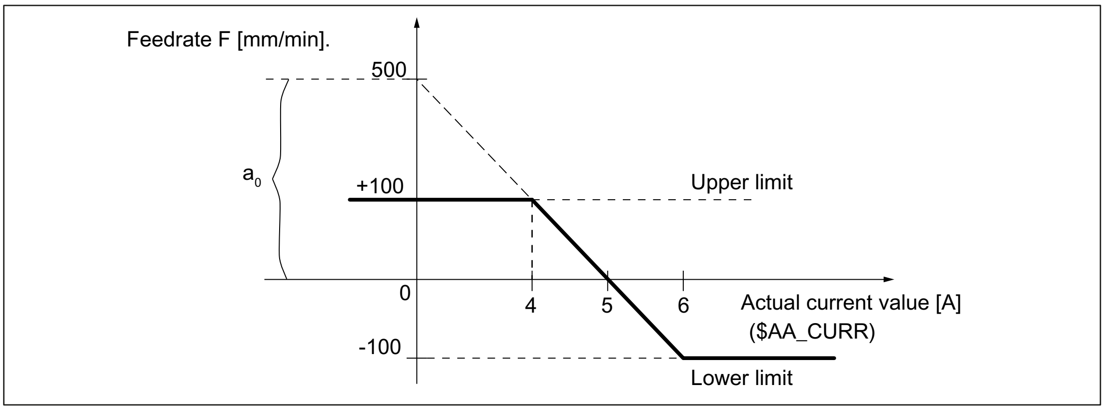


Figure 2-2 Example of linear dependency

Parameter	Meaning
`<Poly_No>`:	Number of the polynomial, e.g. = 1
`<Lo_Limit>`:	Lower limit of the function values = -100
`<Up_Limit>`:	Upper limit of the function values = 100
a_0:	Axis section on the ordinate (feedrate): (5 - 4) / 100 = 5 / a_0 a_0 = 100 * 5 / (5 - 4) = 500
a_1:	Gradient of the straight line: a_1 = 100 / (4 - 5) = -100
a_2:	= 0 (no square component)
a_3:	= 0 (no cubic component)

Program code

```
FCTDEF(1, -100, 100, 500, -100, 0, 0)
; Or in abbreviated notation without parameters a2 and a3
FCTDEF(1, -100, 100, 500, -100)
```

2.3.17 Overlaid movements ($AA_OFF)

Overlaid movements

The system variable $AA_OFF can be used to specify a position offset in a channel axis which is traversed immediately:

$AA_OFF[<channel axis>] = <position offset>

The following machine data can be used to set whether the position offset of the system variable is to be assigned or summed up (integrated):

MD36750 $MA_AA_OFF_MODE, bit 0 = <value>

<value>	Meaning
0	Assignment: $AA_OFF = <position offset>
1	Sum (integration): $AA_OFF += <position offset>

Note

Limitation of the overlay velocity

The maximum permissible velocity with which the position offset can be traversed can be set via the machine data:

MD32070 $MA_CORR_VELO (axis velocity for overlay)

Limitation

The value of $AA_OFF can be limited via the following setting data:

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

The status of the limitation can be read via the following system variable:

$AA_OFF_LIMIT[<axis>] == <value>

Value	Meaning
-1	Offset value is limited in the negative direction
1	Offset value is limited in the positive direction
0	No limitation of the offset value

Reset behavior

With static synchronized actions (`IDS = <number> DO $AA_OFF = <value>`), deselection of the position offset effective in $AA_OFF results in an immediate new overlaid movement. The reset behavior with regard to $AA_OFF can therefore be set via the following machine data:

MD36750 $MA_AA_OFF_MODE, bit 1 = <value>

<value>	Meaning
0	The position offset in $AA_OFF is deselected with RESET
1	The position offset in $AA_OFF is retained after RESET

JOG mode

Execution of an overlaid movement because of $AA_OFF can also be enabled for JOG mode:

MD36750 $MA_AA_OFF_MODE, bit 2 = <value>

<value>	Meaning
0	JOG mode: Overlaid movement because of $AA_OFF disabled
1	JOG mode: Overlaid movement because of $AA_OFF enabled

A mode change to JOG mode is only possible when the current position offset has been traversed. Otherwise the following alarm is displayed:

Alarm "16907 Action ... only possible in stop state"

Supplementary conditions

- Interrupt routines and ASUB

  When an interrupt routine is activated, modal motion-synchronous actions are retained and are also effective in the ASUB. If the subprogram return is not made with `REPOS`, the modal synchronized actions changed in the asynchronous subprogram 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 ASUB are not effective in the interrupted program. Polynomial coefficients programmed with `FCTDEF` are not affected by ASUB and `REPOS`.

  The polynomial coefficients from the calling program are active in the ASUB. The polynomial coefficients from the ASUB continue to be active in the calling program.

- End of program

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

- Block search: Collection of the polynomial coefficients

  During block search with calculation, the polynomial coefficients are collected in the system variables.

- Block search: Deselection of active overlaid movements

  During block search, the `CORROF` and `DRFOF` commands are collected and output in an action block. All the deselected DRF offsets are collected in the last block that contains `CORROF` or `DRFOF`.

  The commands for the deselection of overlaid movements `CORROF(<axis>, "AA_OFF")` are not collected during a block search. If a user wishes to continue to use this search run, this is possible by means of block search via "SERUPRO" program testing.

  Reference:
  Function Manual Basic Functions; Mode Group, Channel, Program Operation (K1)

- Deselection of the position offset in case of synchronized actions

  Alarm 21660 is displayed if a synchronized action is active when the position offset is deselected via the `CORROF(<axis>,"AA_OFF")` command. $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:

Programming Manual, Fundamentals

Note

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

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

`DRF` offsets, zero offsets external, etc., are only taken into consideration in the case of main run variables that are defined in the MCS.

2.3.18 Online tool length compensation ($AA_TOFF)

Function

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

Velocity and acceleration with which specified tool length compensations can be traversed via the system variable $AA_TOFF, can be specified via the following machine data:

- MD21194 $MC_TOFF_VELO (velocity, online offset in tool direction)
- MD21196 $MC_TOFF_ACCEL (acceleration, online offset in tool direction)

For further information regarding the activation of the function, see:

References:

Programming Manual, Job Planning; Section "Transformations "TOFFON, TOFFOF""

Applications in synchronized actions

In synchronized actions, tool length compensations can be applied in all three dimensions via the system variable $AA_TOFF. The three geometry axis names X, Y, Z are used as index. All three offset directions can be active at the same time.

For an active orientation transformation or for an active tool carrier that can be oriented, the offsets are effective in the respective tool axes. An overlaid motion must be switched off with `TOFFOF()` before switching a transformation on or off.

After deselection of the tool length compensation in one dimension, the value of the system variable $AA_TOFF in this dimension is equal to 0.

Mode of operation of the offset in the tool direction

The tool length compensations do not change the tool parameters, but are taken into account within the transformation or the tool carrier that can be orientated, so that offsets are obtained in the tool coordinate system.

For each dimension, it is possible to define whether the tool length compensation specified in $AA_TOFF should be calculated as an absolute or incremental (integrating) value via the following machine data:

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

The current value of the tool length compensation can be read via the system variable $AA_TOFF_VAL.

Note

An evaluation of the variables $AA_TOFF_VAL is only useful in conjunction with an active orientation transformation or an active tool carrier.

Examples

Selecting the online tool length compensation

Machine data for online tool length compensation:

- MD21190 $MC_TOFF_MODE = 1
- MD21194 $MC_TOFF_VEL[0] = 10000
- MD21194 $MC_TOFF_VEL[1] = 10000
- MD21194 $MC_TOFF_VEL[2] = 10000
- MD21196 $MC_TOFF_ACC[0] = 1
- MD21196 $MC_TOFF_ACC[1] = 1
- MD21196 $MC_TOFF_ACC[2] = 1

Activate online tool length compensation in the part program:

Program code

```
N5 DEF REAL XOFFSET
; Activate orientation transformation
N10 TRAORI
; Activate tool length compensation in the Z direction
N20 TOFFON(Z)
; Tool length compensation in the Z direction: 10 mm
N30 WHEN TRUE DO $AA_TOFF[Z] = 10
G4 F5
...
; Static synchronized action: Tool length compensation in the X direction
; corresponds to the position of the X2 axis in the WCS
N50 ID=1 DO $AA_TOFF[X] = $AA_IW[X2]
G4 F5
...
; Note: Current total tool length compensation in the X direction
N100 XOFFSET = $AA_TOFF_VAL[X]
; Retract the tool length compensation in the X direction to 0
N120 TOFFON(X, -XOFFSET)
G4 F5
```

Deselecting the online tool length compensation

Program code
; Activate orientation transformation
N10 TRAORI
; Activate tool length compensation in the X direction
N20 TOFFON(X)
; Tool length compensation in the X direction: 10 mm
N30 WHEN TRUE DO $AA_TOFF[X] = 10
G4 F5
...
; Delete tool length compensation in the X direction
; No axis is traversed. To the current position in the WCS,
; the position offset in accordance with the current orientation
; is added.
N80 TOFFOF(X)
N90 TRAFOOF

Activating and deactivating in the part program

The online tool length compensation is activated in the part program with `TOFFON` and deactivated with `TOFFOF`. When activating for the respective offset direction, an offset value can be specified, e.g. `TOFFON(Z,25)`, which is then immediately traversed. The status of the online tool length compensation is activated at the NC/PLC interface via the following signals:

- DB21, ... DBX318.2 (`TOFF` active)
- DB21, ... DBX318.3 (`TOFF` movement active)

> **Note**
>
> The online tool length compensation remains inactive until it is reselected using via `TOFFON` in the part program.

Behavior at reset and power on

The behavior at reset can be set via the machine data:

MD21190 $MC_TOFF_MODE, bit 0 = <value> (operation of tool offset in tool direction)

Value	Meaning
0	The tool length offset $AA_TOFF is deselected at reset
1	The tool length offset $AA_TOFF is retained at reset

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

Similarly, a transformation or a tool carrier that can be oriented, can be deselected **after** reset via the following machine data:

MD20110 $MC_RESET_MODE_MASK (initial setting after reset)

The tool length compensation must also be deleted in this case.

If a tool length offset is to remain active extending beyond a reset, and a transformation change or a change of the tool carrier that can be oriented takes place, then alarm 21665 "Channel %1 $AA_TOFF[ ] reset" is output. The tool length compensation is set to 0.

After power on, all tool length offsets are set to 0.

The function is deactivated after POWER ON.

Behavior at change of operating mode

The tool length compensation remains active after a change of operating mode. The offset is executed in all operating modes except JOG and REF.

If a tool length compensation is traversed because of $AA_TOFF[ ] at a change of operating mode, the operating mode changeover is only carried out after the traversal of the tool length compensation. Alarm 16907 "Channel %1 action %2 <ALNX> possible only in stop state" is displayed.

Behavior with REPOS

The tool length compensation is active in `REPOS` mode.

Supplementary conditions

With an existing tool length offset, the following supplementary conditions must be taken into account:

- A transformation must be switched off with `TRAFOOF`.
- Before activating a transformation in the part program, an active tool length offset must be deleted with `TOFFOF`.
- A transformation is switched off when changing over from `CP` to `PTP`. A tool length offset must be deleted **before** the changeover. If a tool length compensation is active when changing to axis-specific manual travel in JOG mode, the change to `PTP` is not performed. `CP` remains active until the tool length compensation has been deleted via `TOFFOF`.
- Before a geometry axis interchange, an active tool length offset in the direction of the geometry axis must be deleted via `TOFFOF`.
- Before a change of plane, an active tool length offset must be deleted via `TOFFOF`.
- The `TOFFON` and `TOFFOF` are not collected during a block search and not output in the action block.

2.3.19 Current block in the interpolator ($AC_BLOCKTYPE, $AC_BLOCKTYPEINFO, $AC_SPLITBLOCK)

Information on the block currently being processed in the main run can be read in synchronized actions via the following system variables.

$AC_BLOCKTYPE and $AC_BLOCKTYPEINFO

The system variable $AC_BLOCKTYPE contains the block type or the ID for the function that generated the block.

The system variable $AC_BLOCKTYPEINFO contains, in addition to the block type (thousands position), the function-specific cause for the generation of the intermediate block.

$AC_BLOCKTYPE		$AC_BLOCKTYPEINFO	
Value	Meaning: Current block has been generated because of ...	Value	Meaning
0	Programmed block!	-	-
1	NC as intermediate block	1000	Contains no further information
2	Chamfer/rounding	2001	Straight line
		2002	Circle
3	Smooth approach/retraction (SAR)	3001	Approach with straight line
		3002	Approach with quadrant
		3003	Approach with semicircle
4	Tool offset	4001	Approach block after STOPRE
		4002	Connection blocks if intersection point not found
		4003	Point-type circle on inner corners (on TRACYL only)
		4004	Bypass circle (or conical cut) at outer corners
		4005	Approach blocks for offset suppression
		4006	Approach blocks on repeated TRC activation
		4007	Block split due to excessive curvature
		4008	Compensation blocks for 3D front milling (tool vector parallel to plane vector)
5	Corner rounding	5001	Rounding contour through G641
		5002	Rounding contour through G642
		5003	Rounding contour through G643
		5004	Rounding contour through G644
6	Tangential tracking (TLIFT)	6001	Linear movement of the tangential axis without lift movement
		6002	Non-linear movement of the tangential axis (polynomial) without lift movement
		6003	Lift movement: Tangential axis and lift movement start simultaneously
		6004	Lift movement: Tangential axis does not start until a certain lift position is reached

$AC_BLOCKTYPE		$AC_BLOCKTYPEINFO	
Value	Meaning: Current block has been generated because of ...	Value	Meaning
7	Path segmentation	7001	Programmed path segmentation is active without punching or nibbling
		7002	Programmed path segmentation with active punching or nibbling
		7003	Automatically, internally generated path segmentation
8	Compile cycle	x	x: ID of the compile cycle application that generated the block
9	Path-relative orientation interpolation (ORIPATH/ORIROTC)	9000	Interpolation of the tool orientation with ORIPATH
		9001	Interpolation of the rotation of the tool with ORIROTC
10	Pole handling with orientation transformation	10000	Look-ahead positioning of the pole axis
		10001	Traversal of the pole taper

$AC_SPLITBLOCK

The system variable $AC_SPLITBLOCK can be used to determine whether an internally generated block or a programmed block shortened by the NC is present.

$AC_SPLITBLOCK	
Value	Meaning:
0	Programmed block. A block generated by the compressor is also treated as a programmed block.
1	Internally generated block or a shortened original block
3	Last block in a chain of internally generated blocks or shortened original blocks

Example

Synchronized actions for counting smoothing blocks.

The query of the system variable $AC_TIMEC == 0 (interpolation cycles since start of the block) ensures that the block type is determined only once at the start of the block.

```
Program code                 Comment
$AC_MARKER[0]=0              ;   Counter for all smoothing blocks
$AC_MARKER[1]=0              ;   Counter for G641 smoothing blocks
$AC_MARKER[2]=0              ;   Counter for G642 smoothing blocks
...
; Synchronized action for counting all smoothing blocks
ID=1 WHENEVER ($AC_TIMEC==0) AND ($AC_BLOCKTYPE==5) DO
 $AC_MARKER[0] = $AC_MARKER[0] + 1
...
```

Program code	Comment
; Synchronized action for counting the G641 smoothing blocks	
ID=2 WHENEVER ($AC_TIMEC==0) AND ($AC_BLOCKTYPEINFO==5001) DO	
$AC_MARKER[1] = $AC_MARKER[1]+1	
...	
; Synchronized action for counting the G642 smoothing blocks	
ID=3 WHENEVER ($AC_TIMEC==0) AND ($AC_BLOCKTYPEINFO==5002) DO	
$AC_MARKER[2] = $AC_MARKER[2] + 1	
...	

2.3.20 Initialization of array variables (SET, REP)

Function

Array variables can also be initialized in synchronized actions via the `SET` and `REP` commands.

For a detailed description of the commands, refer to:

References

Programming Manual, Job Planning; Section "Flexible NC programming" > "Variables" > "Definition and initialization of array variables (DEF, SET, REP)"

Example

Program code

```
PROC MAIN
N10 DEF REAL SYG_IS[3,2]
...
WHEN TRUE DO SYG_IS[0,0]=REP(0.0,3)
WHEN TRUE DO SYG_IS[1,1]=SET(3,4,5)
...
```

Supplementary conditions

- Only array variables that can be written in synchronized actions are initialized.

2.4 User-defined variables for synchronized actions

GUD variables capable of synchronized actions

As well as specific system variables, predefined global synchronized-action user variables (synchronized action GUD) can also be used in synchronized actions. The number of synchronized action GUD items available to the user is parameterized for each specific data type and access using the following machine data:

- MD18660 $MM_NUM_SYNACT_GUD_**REAL**[<x>] = <number>
- MD18661 $MM_NUM_SYNACT_GUD_**INT**[<x>] = <number>
- MD18662 $MM_NUM_SYNACT_GUD_**BOOL**[<x>] = <number>
- MD18663 $MM_NUM_SYNACT_GUD_**AXIS**[<x>] = <number>
- MD18664 $MM_NUM_SYNACT_GUD_**CHAR**[<x>] = <number>
- MD18665 $MM_NUM_SYNACT_GUD_**STRING**[<x>] = <number>

The index <x> is used to specify the data block (access rights) and the value <number> to specify the number of synchronized-action GUDs for each data type (REAL, INT, etc.). A 1-dimensional array variable with the following naming scheme is then created in the relevant data block for each data type.: SYG_<data type><access right>[<index>]:

Index <x>		Data type (MD18660 to MD18665)					
	Block	REAL	INT	BOOL	AXIS	CHAR	STRING
0	SGUD	SYG_RS[i]	SYG_IS[i]	SYG_BS[i]	SYG_AS[i]	SYG_CS[i]	SYG_SS[i]
1	MGUD	SYG_RM[i]	SYG_IM[i]	SYG_BM[i]	SYG_AM[i]	SYG_CM[i]	SYG_SM[i]
2	UGUD	SYG_RU[i]	SYG_IU[i]	SYG_BU[i]	SYG_AU[i]	SYG_CU[i]	SYG_SU[i]
3	GUD4	SYG_R4[i]	SYG_I4[i]	SYG_B4[i]	SYG_A4[i]	SYG_C4[i]	SYG_S4[i]
4	GUD5	SYG_R5[i]	SYG_I5[i]	SYG_B5[i]	SYG_A5[i]	SYG_C5[i]	SYG_S5[i]
5	GUD6	SYG_R6[i]	SYG_I6[i]	SYG_B6[i]	SYG_A6[i]	SYG_C6[i]	SYG_S6[i]
6	GUD7	SYG_R7[i]	SYG_I7[i]	SYG_B7[i]	SYG_A7[i]	SYG_C7[i]	SYG_S7[i]
7	GUD8	SYG_R8[i]	SYG_I8[i]	SYG_B8[i]	SYG_A8[i]	SYG_C8[i]	SYG_S8[i]
8	GUD9	SYG_R9[i]	SYG_I9[i]	SYG_B9[i]	SYG_A9[i]	SYG_C9[i]	SYG_S9[i]
Where i = 0 to (<number> - 1) Block: _N_DEF_DIR/_N_ ... _DEF, e.g for SGUD ⇒ _N_DEF_DIR/_N_**SGUD**_DEF							

Properties

Synchronized-action GUD have the following properties:

- Synchronized-action GUD can be read and written in synchronized actions and part programs/cycles.
- Synchronized-action GUD can be accessed via the OPI.
- Synchronized-action GUD is displayed on the HMI user interface in the "Parameters" operating area.
- Synchronized-action GUD can be used on the HMI in the Wizard, in the variables view and in the variables log.
- The array size for STRING type synchronized action GUD is set to a fixed value of 32 (31 characters + \0).
- Even if no definition files have been created manually for global user data (GUD), synchronized-action GUD defined using machine data can be read in the corresponding GUD block from the HMI.

Note

User variables (GUD, PUD, LUD) can only be defined with the same name as synchronized-action GUD (`DEF ... SYG_xy`) if no synchronized-action GUD has been parameterized with the same name (MD18660 - MD18665). These user-defined items of GUD **cannot** be used in synchronized actions.

Access rights

The access rights defined in a GUD definition file remain valid and refer only to the GUD variables defined in this GUD definition file.

Deletion behavior

If the content of a particular GUD definition file is reactivated, the old GUD data block in the active file system is deleted first. The configured synchronized-action GUD is also reset at this point. This process is also possible using the HMI in the operator area "Services" > "Define and activated user data (GUD)".

2.5 Language elements for synchronized actions and technology cycles

The following language elements can be used in synchronized actions and technology cycles:

Fixed addresses	
L	Subprogram number
F	Feed
S 1) 2)	Spindle
M 1) 2)	M function
H1)	H function
1) Section: "Output of M, S and H auxiliary functions to the PLC (Page 60)" 2) Section: "Traversing spindles (M, S, SPOS) (Page 86)"	

Fixed addresses with axis extension Miscellaneous	
POS	Positioning axis Section: "Traversing command axes (POS) (Page 73)"
POSA	Modal positioning axis
SPOS	Spindle positioning Section: "Traversing spindles (M, S, SPOS) (Page 86)"
MOV 1)	Positioning axis Section: "Starting/stopping axes (MOV) (Page 79)"
FA	Axial feed Section: "Axial feedrate (FA) (Page 80)"
OVRA	Axial override
ACC	Axial acceleration
MEASA	Axial measurement with deletion of distance-to-go
MEAWA	Axial measurement without deletion of distance-to-go Section: "Measurement (MEAWA, MEAC) (Page 97)"
MEAC	Cyclic measuring Section: "Measurement (MEAWA, MEAC) (Page 97)"
SCPARA	Parameter set changeover
VELOLIMA	Axial velocity/speed limitation
ACCLIMA	Axial acceleration limitation
JERKLIMA	Axial jerk limitation
1) Not permitted in technology cycles	

Settable addresses: Travel to fixed stop [1)]	
FXS	Activate travel to fixed stop
FXST	Torque limit for travel to fixed stop
FXSW	Monitoring window for travel to fixed stop
FOCON	Activate travel with limited torque/force
FOCOF	Deactivate travel with limited torque/force
1) Section: "Travel to fixed stop (FXS, FXST, FXSW, FOCON, FOCOF, FOC) (Page 100)"	

Settable addresses: Couplings > Generic coupling [1)]	
CPBC	Block change criterion with active coupling
CPDEF	Create coupling module
CPDEL	Delete coupling module
CPFMOF	Behavior of the following axis when switching off the coupling
CPFMON	Behavior of the following axis when switching on the coupling
CPFMSON	Synchronization mode during coupling
CPFPOS	Synchronized position of the following axis when switching on
CPFRS	Reference system for the coupling module of the following axis
CPLCTID	Number of the curve table for the coupling of the following axis
CPLDEF	Definition of the reference: Leading axis to following axis
CPLDEL	Cancellation of the reference: Leading axis to following axis
CPLDEN	Coupling factor: Numerator
CPLNUM	Coupling factor: Denominator
CPLDYPRIO	Priority of the leading axis for the dynamic limitation
CPLDYVLL	Limitation of the overlaid motion of the leading axis: Lower limit
CPLDYVLU	Limitation of the overlaid motion of the leading axis: Upper limit
CPLINSC	Scaling factor for the input value of the leading axis
CPLINTR	Offset value for the input value of the leading axis
CPLOF	Coupling of leading axis to following axis: Switch off
CPLON	Coupling of leading axis to following axis: Switch on
CPLOUTSC	Scaling of the output value
CPLOUTTR	Offset of the output value
CPLPOS	Synchronized position of the leading axis when switching on
CPLSETVAL	Coupling type of the following axis to the leading axis
CPMALARM	Define alarm behavior
CPMPRT	Define start behavior for program test
CPMRESET	Define reset behavior
CPMSTART	Define start behavior
CPMPVDI	Define behavior regarding NC/PLC interface signals
CPOF	Deactivation of the coupling to all defined leading axes
CPON	Activation of the coupling to all defined leading axes
CPSETTYPE	Define basic coupling properties
CPSYNCOP	Position synchronism "coarse"

Settable addresses: Couplings > Generic coupling [1]	
CPSYNCOP2	Position synchronism 2 "coarse"
CPSYNFIP	Position synchronism "fine"
CPSYNFIP2	Position synchronism 2 "fine"
CPSYNCOV	Velocity synchronism "coarse"
CPSYNFIV	Velocity synchronism "fine"
1) Section: "Couplings (CP..., LEAD..., TRAIL..., CTAB...) (Page 92)"	

G functions: Set measuring system [1]	
G70	Inch measuring system
G71	Metric measuring system
G700	Inch measuring system
G710	Metric measuring system
1) Section: "Setting the measuring system (G70, G71, G700, G710) (Page 76)"	

Predefined subprograms: Miscellaneous	
POLFA	Axial retraction position for single axis
POLFC	Axial retraction position for channel axes
STOPREOF	Revoke preprocessing stop Section: "Cancel preprocessing stop (STOPREOF) (Page 70)"
RDISABLE	Read-in disable Programmed read-in disable (RDISABLE) (Page 69)"
DELDTG	Delete distance-to-go Section: "Delete distance-to-go (DELDTG) (Page 71)"
LOCK	Lock synchronized action
UNLOCK	Unlock synchronized action
RESET	Reset technology cycle
ICYCON	Technology cycle: One block per interpolation cycle
ICYCOF	Technology cycle: All blocks in one interpolation cycle
SYNFCT	Evaluate polynomial function Section: "Polynomial evaluation (SYNFCT) (Page 61)"
FTOC	Tool fine compensation Section: "Online tool offset (FTOC) (Page 67)"
SOFTENDSA	Software limit switch
PROTA	Change status of a protection zone
SETM	Set marker of the channel coordination Section: "Channel synchronization (SETM, CLEARM) (Page 102)"
CLEARM	Delete marker of the channel coordination Section: "Channel synchronization (SETM, CLEARM) (Page 102)"
RET	Subprogram return

Predefined subprograms: Miscellaneous	
GET	Request axis Section: "Axis replacement (GET, RELEASE, AXTOCHAN) (Page 81)"
RELEASE	Release axis Section: "Axis replacement (GET, RELEASE, AXTOCHAN) (Page 81)"
AXTOCHAN	Transfer axis to another channel Section: "Axis replacement (GET, RELEASE, AXTOCHAN) (Page 81)"
AXCTSWEC	Withdrawing axis container rotation enable Section: "Withdrawing the enable for the axis container rotation (AXCTSWEC) (Page 88)"
SETAL	Display user-specific alarm Section: "User-specific error reactions (SETAL) (Page 103)"
IPOBRKA	Block change criterion: Deceleration ramp
ADISPOSA	Tolerance window for end-of-motion criterion

Predefined subprograms: Coupling > Coupled motion 1)	
TRAILON	Coupled motion on
TRAILOF	Coupled motion off
1) Section: "Couplings (CP..., LEAD..., TRAIL..., CTAB...) (Page 92)"	

Predefined subprograms: Couplings > Master value coupling 1)	
LEADON	Master value coupling on
LEADOF	Master value coupling off
1) Section: "Couplings (CP..., LEAD..., TRAIL..., CTAB...) (Page 92)"	

Predefined subprograms: Couplings > Torque coupling (master/slave)	
MASLON	Coupling on
MASLOF	Coupling off
MASLDEF	Define coupling
MASLDEL	Delete coupling
MASLOFS	Coupling with slave spindle off

Predefined functions: Coupling > Curve tables 1)	
CTAB	Calculates the following axis position based on the leading axis position using the curve table
CTABINV	Calculates the leading axis position based on the following axis position using the curve table
CTABID	Determines the table number of the curve table
CTABLOCK	Disable curve table
CTABUNLOCK	Enable curve table
CTABISLOCK	Determines the lock status of the curve table
CTABEXISTS	Checks whether the curve table exists
CTABMEMTYP	Determines the storage location of the curve table (static/dynamic memory)
CTABPERIOD	Determines the periodicity of the curve table
CTABNO	Determines the number of curve tables
CTABNOMEM	Determines the number of existing curve tables in a specific storage location
CTABSEG	Determines the number of already used curve segments in a specific storage location
CTABSEGID	Determines the number of already used curve segments in a specific table
CTABFSEG	Determines the number of curve segments that are still possible in a specific table
CTABMSEG	Determines the maximum possible number of curve segments in a specific storage location
CTABPOL	Determines the number of already used polynomials in a specific storage location
CTABPOLID	Determines the number of already used polynomials in a specific table
CTABFPOL	Determines the number of polynomials that are still possible in a specific table
CTABMPOL	Determines the maximum possible number of polynomials in a specific storage location
CTABTSV	Determines the following value at the start of the table
CTABTEV	Determines the following value at the end of the table
CTABTSP	Determines the leading value at the start of the table
CTABTEP	Determines the leading value at the end of the table
CTABTMIN	Determines the minimum following value of the table
CTABTMAX	Determines the minimum following value of the table
CTABFNO	Determines the number of curve tables that are still possible in a specific storage location
CTABSSV	Determines the starting value of a table segment for the following axes
CTABSEV	Determines the end value of a table segment for the following axes
1) Section: "Couplings (CP..., LEAD..., TRAIL..., CTAB...) (Page 92)"	

Predefined functions: Arithmetic	
SIN	Sine
ASIN	Arc sine
COS	Cosine
ACOS	Arc cosine
TAN	Tangent
ATAN2	Arc tangent 2
SQRT	Square root
POT	2. 2nd power (square)
TRUNC	Integer component
ROUND	Round to next integer
ROUNDUP	Rounding up of an input value to the next integer
ABS	Absolute value
LN	Natural logarithm
EXP	Exponential function
MINVAL	Smaller of two values
MAXVAL	Larger of two values
BOUND	Check for defined value range

Predefined functions: Current machine data values	
GETMDACT	Determines the current value of the machine data
GETMDPEAK	Determines the maximum value that has occurred in the machine data since the last RESETPEAK
GETMDLIM	Determines the maximum or minimum limit value of the machine data
RESETPEAK	Resets the maximum value again for GETMDPEAK

Predefined functions: Format conversions	
ITOR	INT → REAL
RTOI	REAL → INT
RTOB	REAL → BOOL
BTOR	BOOL → REAL
ITOB	INT → BOOL
BTOI	BOOL → INT

Predefined functions: Safety Integrated	
SIRELAY	Activation of the safety functions parameterized with SIRELIN, SIRELOUT and SIRELTIME

Predefined functions: Miscellaneous	
POSRANGE	Axis position within the tolerance range around the reference position Section: "Position in specified reference range (POSRANGE) (Page 78)"
PRESETON	Set actual value for an axis Section: "Set actual value (PRESETON) (Page 91)"

References

For detailed descriptions of the language elements not described in this manual, refer to:

- Programming Manual, Fundamentals
- Programming Manual, Job Planning

2.6 Language elements for technology cycles only

The following language elements may only be used in technology cycles:

Jump statements	
IF	Branch
GOTO	Jump to label, search direction forward, then backward
GOTOF	Jump to label, search direction forward
GOTOB	Jump to label, search direction backward

End of program	
M02	End of program
M17	End of program
M30	End of program
RET	End of program

References

For detailed descriptions of the statements not described in this manual, refer to:

- Programming Manual, Fundamentals
- Programming Manual, Job Planning

2.7 Actions in synchronized actions

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

Output timing

Auxiliary functions of the M, S and H type can be output from synchronized actions. The output to the PLC is immediate, i.e. directly in the interpolation cycle in which the action is executed.

Any output times set via the machine data for auxiliary functions have no effect when output from synchronized actions:

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

Maximum number

General

A maximum of 10 auxiliary functions can be output simultaneously from the part program and the active synchronized actions of a channel, i.e. in one OB40 cycle of the PLC.

Synchronized-action-specific

The maximum permissible number of auxiliary functions in the action part of a synchronized action is:

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

Non-modal synchronized actions

In non-modal synchronized actions (without specification of `ID` or `IDS`), auxiliary functions can only be output in conjunction with the scanning frequency `WHEN` or `EVERY`.

Predefined M functions

Predefined M functions generally must not be output in synchronized actions.

Exceptions: `M3`, `M4`, `M5`, `M40`, `M41`, `M42`, `M43`, `M44`, `M45`, `M70` and `M17`

See also

Frequency (WHENEVER, FROM, WHEN, EVERY) (Page 13)

2.7.2 Reading and writing of system variables

The system variables of the NCK are listed in the "System Variables" Parameter Manual with their respective properties. System variables that can be read or written in the action part of synchronized actions are marked with an "X" in the corresponding line (Read or Write) of the "SA" (synchronized action) column.

Note

System variables used in synchronized actions are implicitly read and written synchronous to the main run.

References:
System Variables Parameter Manual

2.7.3 Polynomial evaluation (SYNFCT)

Application

A variable that is evaluated via a polynomial can be read with the SYNFCT function in the main run and the result written to another variable. Application examples:

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

Syntax

```
SYNFCT(<Poly_No>,<SysVar_Out>,<SysVar_In>)
```

Meaning

Parameter	Meaning
`<Poly_No>`:	Number of the polynomial defined with `FCTDEF`: $f(x) = a_0 + a_1 * x + a_2 * x^2 + a_3 * x^3$
`<SysVar_Out>`:	System variable, output: `<SysVar_Out> =` f(x)
`<SysVar_In>`:	System variable, input: x = `<SysVar_In>`
For information on `FCTDEF`, see Section "Polynomial coefficients, parameters ($AC_FCT ...) (Page 38)"	

Example: Additive override of the path feedrate

An override value is added to the programmed feedrate (F word):

$F_{active} = F_{programmed} + F_{AC}$

<SysVar_Out>	Meaning
$AC_VC	Additive path feedrate override
$AA_VC[axis]	Additive axial feedrate override

Input value is the actual current value $AA_CURR of the X axis.

The operating point is set to 5 A.

The feedrate may be altered by ±100 mm/min whereby the axial current deviation may be ±1 A.

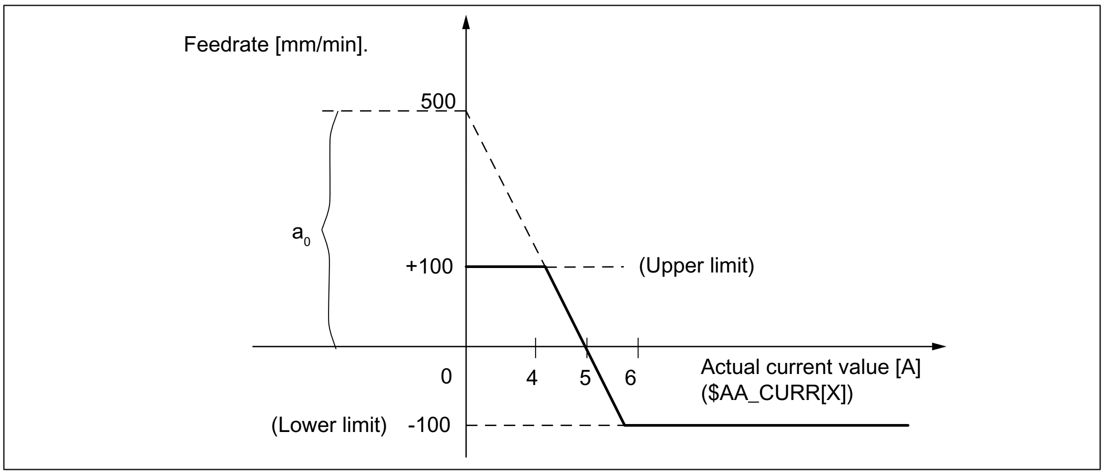


Figure 2-3 Example: Additive control of path feed

Determining the parameters of the `FCTDEF` function:

```
FCTDEF(<Poly_No>,<Lo_Limit>,<Up_Limit>,a0,a1,a2,a3)
```

`<Poly_No>`:	= 1 (example)
`<Lo_Limit>`:	= -100
`<Up_Limit>`:	= 100
	Polynomial: $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$
a_0:	$1 / 100 = 5 / a_0 \Rightarrow a_0 = 500$
a_1	= 100 mm/min / -1 A = -100 [mm/min / A]
a_2	= 0 (no square component)
a_3	= 0 (no cubic component)

Calculation of the override value:

`SYNFCT(<Poly_No>,<SysVar_Out>,<SysVar_In>)`

`<Poly_No>`: = 1

`<SysVar_Out>`: $AC_VC (additive path feedrate override)

`<SysVar_In>`: $AA_CURR (drive actual current value)

Programming:

Program code
`N100 FCTDEF(1, -100, 100, 500, -100)`
`N110 ID=1 DO SYNFCT(1, $AC_VC[X], $AA_CURR[X])`

Example: Multiplicative override of the path feedrate

The programmed feedrate is multiplied by a percentage factor (additional override):

$F_{active} = F_{programmed} * Factor_{AC}$

<SysVar_Out>	**Meaning**
$AC_OVR	Path override can be specified via synchronized action

Input value is the percentage drive load $AA_LOAD of the X axis.

The operating point is set to 100% at 30% drive load.

The axis must stop at 80% load.

An excessive velocity corresponding to the programmed value +20% is permissible.

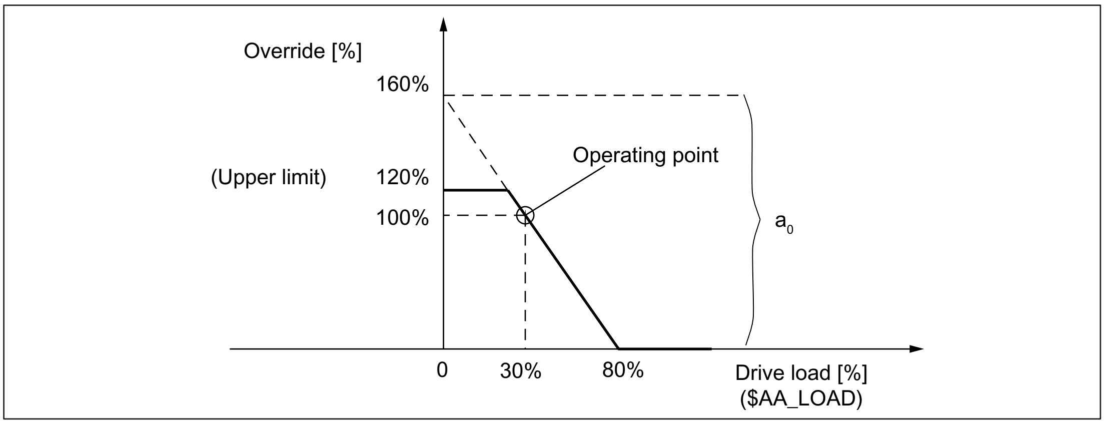


Figure 2-4 Example: Multiplicative control

Determining the parameters of the FCTDEF function:

```
FCTDEF(<Poly_No>,<Lo_Limit>,<Up_Limit>,a0,a1,a2,a3)
```

<Poly_No>:	= 2 (example)
<Lo_Limit>:	= 0
<Up_Limit>:	= 120
	Polynomial: $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$
a_0:	$50 / 100 = 80 / a_0 \Rightarrow a_0 = 160$
a_1	= 100 % / -50 % = -2
a_2	= 0 (no square component)
a_3	= 0 (no cubic component)

Calculation of the override value:

```
SYNFCT(<Poly_No>,<SysVar_Out>,<SysVar_In>)
```

<Poly_No>:	= 2
<SysVar_Out>:	$AC_OVR (path override can be specified via synchronized action)
<SysVar_In>:	$AA_LOAD (drive load)

Programming:

```
Program code
N100 FCTDEF(2, 0, 120, 160, -2)
N110 ID=1 DO SYNFCT(2, $AC_OVR[X], $AA_LOAD[X])
```

Example: Clearance control

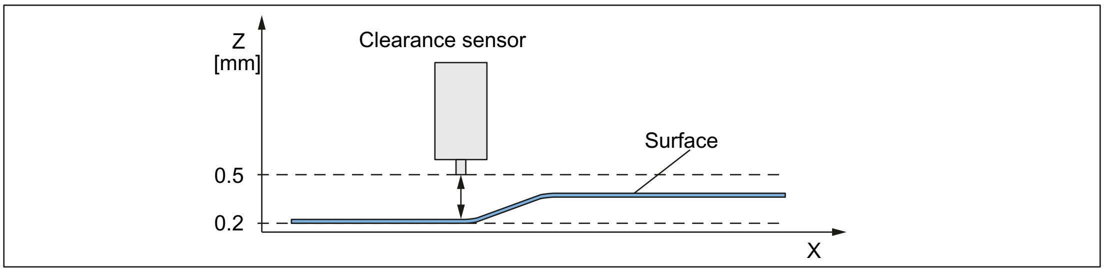


Figure 2-5 Clearance control: Principle

The clearance control of the infeed axis Z is performed via the FCTDEF and SYNFCT functions as well as by the system variables $AA_OFF and $A_INA.

Supplementary conditions:

- The analog voltage of the clearance sensor is connected via the analog input $A_INA[3].
- The position deviations are summed up in $AA_OFF (integrated):
  MD36750 $MA_AA_OFF_MODE, bit 0 = 1
- If the upper limit of the Z axis is exceeded by 1 mm, the X axis is stopped:
  SD43350 $SA_AA_OFF_LIMIT[Z] = 1
  See also Section "Overlaid movements ($AA_OFF) (Page 40)".

Note

$AA_OFF is effective in the basic coordinate system (BCS)

The offset is effective before the kinematic transformation in the basic coordinate system (BCS). The example therefore **cannot** be used for a clearance control in the orientation direction of the tool (workpiece coordinate system WCS).

For clearance control system with high dynamic response or 3D clearance control, see:

References:

Function Manual Special Functions; Clearance Control (TE1)

Customized responses

When the limit value SD43350 $SA_AA_OFF_LIMIT is reached, customized responses can be triggered, for example:

- Section "Override ($A...OVR) (Page 31)"
- Section "User-specific error reactions (SETAL) (Page 103)"

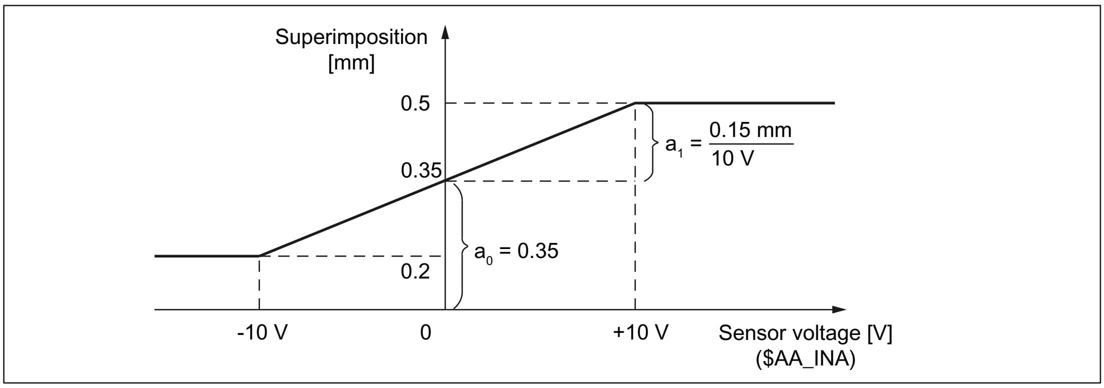


Figure 2-6 Clearance control

Determining the parameters of the `FCTDEF` function:

```
FCTDEF(<Poly_No>,<Lo_Limit>,<Up_Limit>,a0,a1,a2,a3)
```

`<Poly_No>`:	= 1 (example)
`<Lo_Limit>`:	= 0.2
`<Up_Limit>`:	= 0.5
	Polynomial: $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$

a_0:	$10 / x = 20 / 0.3 \Rightarrow a_0 = x + 0.2 = 0.15 + 0.2 = 0.35$
a_1	$= 0.15$ mm / 10 V $= 1.5 * 10^{-2}$ mm/V
a_2	= 0 (no square component)
a_3	= 0 (no cubic component)

Calculation of the override value:

```
SYNFCT(<Poly_No>,<SysVar_Out>,<SysVar_In>)
```

<Poly_No>:	= 1
<SysVar_Out>:	$AA_OFF (overlaid movement of an axis)
<SysVar_In>:	$A_INA (analog input)

Programming:

Program code: %_N_AON_SPF	Comment
PROC AON	; Clearance control "ON"
FCTDEF(1, 0.2, 0.5, 0.35, 1.5 EX-2)	; Polynomial definition
ID=1 DO SYNFCT(1,$AA_OFF[Z],$A_INA[3])	; Clearance control
ID=2 WHENEVER $AA_OFF_LIMIT[Z]<>0 DO $AA_OVR[X] = 0	; Limit value test
RET	
ENDPROC	

Program code: %_N_AOFF_SPF	Comment
PROC AOFF	; Clearance control "OFF"
CANCEL(1)	; Delete clearance control
CANCEL(2)	; Delete limit value check
RET	
ENDPROC	

Program code: %_N_MAIN_MPF	Comment
N100 $SA_AA_OFF_LIMIT[Z]=1	
N110 AON	; Clearance control "ON"
...	
N200 G1 X100 F1000	
N210 AOFF	; Clearance control "OFF"
M30	

See also

Online tool offset (FTOC) (Page 67)

2.7.4 Online tool offset (FTOC)

The FTOC function enables the overlaid movement of a geometry axis for the online tool offset, depending on a reference value, e.g. the actual value of an arbitrary axis. The offset value is calculated on the basis of a polynomial defined with `FCTDEF` (see Section "Polynomial coefficients, parameters ($AC_FCT ...) (Page 38)"). The coefficient a_0 specified in the polynomial definition is also evaluated by `FTOC`.

Example: Machining and dressing in the "Grinding" technology

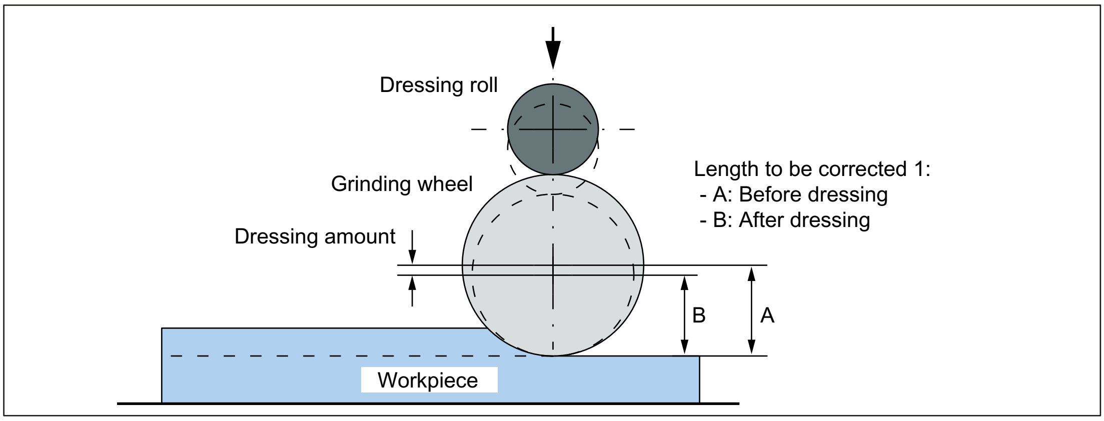


Figure 2-7 Dressing during machining using a dressing roller

References:

Function Manual, Extended Functions; Grinding (W4)

Syntax

```
FTOC(<Poly_No>,<Systemvar>,<Wear>[,<Channel_No>,<Spindle_No>])
```

Meaning

Parameter	Meaning
`<Poly_No>`:	Number of the polynomial defined with `FCTDEF`:
`<Systemvar>`:	Arbitrary system variable of the REAL type that can be used in synchronized actions.
`<Wear>`:	Wear parameter (length 1, 2 or 3) in which the offset value is **added**.

Parameter	Meaning
<Channel_No>:	Target channel in which the offset must be applied. This enables simultaneous dressing from a parallel channel. In the target channel of the offset, the online offset must be switched on with `FTOCON`. If no channel number is programmed, the offset acts in the active channel.
<Spindle_No>:	The spindle number is programmed if a non-active grinding wheel needs to be dressed. Requirement: One of the following functions is active • "Constant grinding wheel peripheral speed" • "Tool monitoring" If no spindle number is programmed, the active tool is compensated.

Example

Compensate length of an active grinding wheel

```
Program code                                Comment
FCTDEF(1, -1000, 1000, -$AA_IW[V], 1)
; FTOC:
; Polynomial no.: 1
; System variable: $AA_IW[V] (axial actual value of the V axis)
; Wear parameter: Length 3
; Target channel: Channel 1
ID=1 DO FTOC(1, $AA_IW[V], 3, 1)
WAITM (1,1,2)                 ;   Synchronization with the machining channel
G1 V-0.05 F0.01 G91           ;   Traversing motion of the V axis
...
CANCEL(1)                     ;   Deselect online offset
...
```

Note

Because no frequency and no condition has been specified in the synchronized action, the action part is executed in every interpolation cycle.

2.7.5 Programmed read-in disable (RDISABLE)

Function

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

An exact stop is initiated at the end of the block containing `RDISABLE` irrespective of whether or not the read-in disable is still active. The exact stop is also triggered if the control is in the continuous-path mode (`G64`, `G641` ... `G645`).

`RDISABLE` can be programmed with reference to the block or also modal (`ID=`, `IDS=`)!

Application

Using `RDISABLE`, for example, the program can be started in the interpolation cycle as a function of external inputs.

Example

Program code	Comment
`WHENEVER $A_INA[2]<7000 DO RDISABLE`	`; Program processing is stopped if the voltage at input 2 drops to below 7 V (assuming that the value 1000 corresponds to 1 V).`
`...`	
`N10 G01 X10`	`; RDISABLE acts at the end of N10, if the condition is fulfilled during its processing.`
`N20 Y20`	
`...`	

Supplementary conditions

Read-in disable RDISABLE in conjunction with axis exchange

Acts via the synchronized actions `RDISABLE` read-in disable and axis exchange (e.g. path axis → positioning axes) together in one block, `RDISABLE` does not act on the action block, but the re-approach block `REPOSA` implicitly generated as a result of the axis exchange:

Program code	Comment
`N100 G0 G60 X300 Y300`	
`N105 WHEN TRUE DO POS[X]=20 FA[X]=20000`	`; Synchronized action → REORG →`
	`; REPOSA`
`N110 WHENEVER $AA_IM[X]<>20 DO RDISABLE`	`; RDISABLE acts on REPOSA`
`N115 G0 Y20`	`; 1st X axis, 2nd Y axis`
`N120 Y-20`	
`N125 M30`	

Path axis X becomes a positioning axis as a result of the synchronized action in the block N105. `REORG` is therefore executed in the channel with `REPOSA`. Therefore, `RDISABLE` in N110 does not act on block N115 – but instead on the internal `REPOSA` block. As a consequence, to start, positioning axis X is traversed to its programmed position and then in block N115, the Y axis to its programmed position.

An explicit release of path axis X before traversing as positioning axis (synchronized action in N105) with `RELEASE(X)` avoids the `REORG` operation, and the X and Y axes traverse together in block N115.

Program code	Comment
N100 G0 G60 X300 Y300	;
N101 RELEASE(X)	; Explicit release
N105 WHEN TRUE DO POS[X]=20 FA[X]=20000	;
...	;

2.7.6 Cancel preprocessing stop (STOPREOF)

With the `STOPREOF` command, an existing preprocessing stop can be cancelled from a synchronized action.

Note

The `STOPREOF` command can only be programmed in non-modal synchronized actions (without specification of ID or IDS) and only in conjunction with the scanning frequency `WHEN`.

Example

- N10: Non-modal synchronized action.
  If the path distance-to-go $AC_DTEB is less than 5 mm, the existing preprocessing stop due to the reading of the analog input $A_INA is cancelled.
- N20: Traversing block whose path distance-to-go is evaluated via $AC_DTEB.
- N30: Branch that triggers the preprocessing stop due to the reading of $A_INA.

Due to the synchronized action, input $A_INA is not evaluated at the end of the N20 block, but already 5 mm before the end of the block. If the voltage is then greater than 5 V at input $A_INA, there is a branch to "MARKE_1".

Program code
N10 WHEN $AC_DTEB < 5 DO STOPREOF
N20 G01 X100
N30 IF $A_INA[7] > 5000 GOTOF MARKE_1

2.7.7 Delete distance-to-go (DELDTG)

The path distance-to-go can be deleted with the `DELDTG` command and axial distances-to-go can be deleted with the `DELDTG (...)` function in synchronized actions.

After deletion of the distance-to-go, the value of the deleted distance-to-go can be read via a system variable:

- Path distance-to-go: $AC_DELT
- Axial distance-to-go: $AA_DELT

Syntax

```
DELDTG

DELDTG(<axis 1>[,<axis 2>, ... ])
```

Meaning

Parameter	Meaning
`DELDTG`	Deletion of the path distance-to-go
`DELDTG(...)`	Deletion of the axial distances-to-go of the specified channel axes
`<Axis n>`:	Channel axis

Supplementary conditions

Path-specific and axial delete distance-to-go

Path-specific and axial delete distance-to-go can only be executed in a **non-modal** synchronized action (without specification of ID or IDS).

Path-specific delete distance-to-go

- The deletion of the path distance-to-go can only be executed in a non-modal synchronized action (without specification of ID or IDS).
- The deletion of the path distance-to-go must **not** be used with active tool radius compensation.

Axial delete distance-to-go

Delete distance-to-go for indexing axes:

- **Without** Hirth tooth system: The axis is braked immediately
- **With** Hirth tooth system: The axis traverses to the next indexing position

Examples

Delete path distance-to-go

If the input $A_IN is set during the traversing block N20, the path distance-to-go is deleted.

Program code

```
N10 WHEN $A_IN[1]==1 DO DELDTG
N20 G01 X100 Y100 F1000
```

Delete axial distances-to-go

N10: If input 1 is set at any time within the part program, the V axis is started as a positioning axis in the positive traversing direction.

N100: Non-modal synchronized action to delete distance-to-go of the V axis, depending on digital input 2.

N110: Non-modal synchronized action to delete distance-to-go of the X1 axis, depending on digital input 3.

N120: The X1 axis is positioned modally. The Y and Z axes are traversed as path axes. The non-modal synchronized actions from N100 and N110 are executed together with N120. The non-modal synchronized actions are also terminated with the end of block N120.

For this reason, the distances-to-go of the X1 and V axes can only be deleted as long as N120 is active.

Program code

```
N10 ID=1 WHEN $A_IN[1]==1 DO MOV[V]=1 FA[V]=700
...
N100 WHEN $A_IN[2]==1 DO DELDTG(V)
N110 WHEN $A_IN[3]==1 DO DELDTG(X1)
N120 POSA[X1]=100 FA[X1]=10 G1 Y100 Z100 F1000
```

2.7.8 Traversing command axes (POS)

Axes can be traversed as command axes via synchronized actions with the POS command. Alternate traversing of the axis via the part program and the synchronized action is thus possible. If a command axis traversed via synchronized actions is subsequently traversed via the part program, a preprocessing stop with reorganization (STOPRE) is executed in the channel of the part program.

Examples:

Example 1: Alternate traversing via part program and synchronized action

Program code	Comment
N10 G01 **X100** Y200 F1000	; Traversing via part program
...	
; Traversing via static synchronized action when input 1 is set	
N20 ID=1 WHEN $A_IN[1]==1 DO **POS[X]=150** FA[X]=200	
...	
CANCEL(1)	; Select synchronized action
...	
; Traversing again via part program => implicit preprocessing stop	
; with reorganization, if the X axis in the meantime has been	
; traversed via synchronized action	
N100 G01 **X240** Y200 F1000	

Example 2: Alternate traversing of the X-axis via two synchronized actions

If the traversing motion of one synchronized action is still active when the traversing motion of the other synchronized action is started, the second traversing motion replaces the first.

Program code
; 1. ; 1st traversing motion
ID=1 EVERY $A_IN[1]>=1 DO POS[V]=100 FA[V]=560
; 2. ; 2nd traversing motion
ID=2 EVERY $A_IN[2]>=1 DO POS[V]=$AA_IM[V] FA[V]=790

Dimensions: Absolute/incremental

The commands G90/G91 to specify the dimensions (absolute/incremental) cannot be programmed in synchronized actions. Therefore by default, the dimensions that were active in the part program at the time of execution of the synchronized action is also effective in the synchronized action.

The following commands can be programmed in the action part to specify the dimensions within a synchronized action:

Command	Meaning
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

Examples:

```
Program code
; Incremental traversing by 10 mm
ID=1 EVERY G710 $AA_IM[B]>75 DO POS[X]=IC(10)
...
; Absolute traversing
ID=1 EVERY G710 $AA_IM[B]>75 DO POS[X]=AC($AA_MW[V]-$AA_IM[W]+13.5)
```

Behavior with active axial frames

If programmable and settable frames and tool length compensations are not explicitly deactivated for inclusion in the calculation for synchronized actions via the following machine data, the frame and/or tool length compensation active in the part program at the time the synchronized action is executed in parallel, takes effect:

MD32074 $MA_FRAME_OR_CORRPOS_NOTALLOWED, bit 9 = 1

Examples

Example 1: Traversing with **active** frames / tool length compensations (bit 9 == 0):

```
Program code                          Comment
N100 TRANS X20                        ;   Zero offset in X: 20 mm.
; Synchronized action: The X axis traverses to position 60 mm
IDS=1 EVERY G710 $A_IN==1 DO POS[X]=40
...
; Zero offset in X: -10 mm. =>
; Synchronized action: The X axis now traverses to position 30 mm
N130 TRANS X-10
...
```

Example 2: Traversing with **deactivated** frames / tool length compensations (bit 9 == 1):

Program code	Comment
N100 TRANS X=0.001	; Zero offset in X: 0.001 degrees
N120 POS[X]=270	; X traverses to position 270.001 degrees
...	
; With $A_IN=1, X traverses to position 180.000 degrees.	
IDS=1 EVERY G710 $A_IN==1 DO POS[X]=180	
...	
; X traverses to position 90.001 degrees	
N130 POS[X]=90	
...	
; Coded position 1 = 100 degrees => X traverses to 100.001 degrees	
N140 POS[X]=CAC(1)	
...	
; Coded position 2 = 200 degrees => X traverses to 200.000 degrees	
N150 POS[X]=CIC(1)	

Note

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

Takeover of the control of a command axis by the PLC

The control of a command axis that has been started via a static synchronized action (`IDS`) is taken over by the PLC irrespective of the status of the part program containing the synchronized action:

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

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

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

Parameterizable axis status

The behavior with regard to the axis status after the end of the part program and NC Reset can be parameterized via the following machine data:

MD30450 $MA_IS_CONCURRENT_POS_AX[<axis>] = <value>

<value>	Axis status before PP end / NC RESET 1)	Axis status after PP end / NC RESET 1)
0	Channel axis	Channel axis
0	Command axis	Channel axis
1	Channel axis	Command axis
1	Command axis	Command axis
1) PP end: Part program end		

See also

Technology cycles (Page 104)

2.7.9 Setting the measuring system (G70, G71, G700, G710)

If a specific measuring system (inch/metric) is not explicitly defined in a synchronized action with `G70`, `G71`, `G700`, `G710`, the measuring system active in the part program at the time the synchronized action is executed takes effect:

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

The following rules apply when defining the measuring system in the synchronized action:

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

Example

Program code	Comment

`N10 ID=1 EVERY $AA_IM[Z]>200 DO POS[Z2]=10`	`; $AA_IM: #`
	`; 200: #`
	`; 10: #`

`N20 ID=2 EVERY $AA_IM[Z]>200 DO` **`G70`** `POS[Z2]=10`	`; $AA_IM: #`
	`; 200: #`
	`; 10: inch`

`N30 ID=3 EVERY` **`G71`** `$AA_IM[Z]>200 DO POS[Z2]=10`	`; $AA_IM: #`
	`; 200: mm`
	`; 10: mm`

`N40 ID=4 EVERY` **`G71`** `$AA_IM[Z]>200 DO` **`G70`** `POS[Z2]=10`	`; $AA_IM: #`
	`; 200: mm`
	`; 10: inch`

`N50 ID=5 EVERY $AA_IM[Z]>200 DO` **`G700`** `POS[Z2]=10`	`; $AA_IM: #`
	`; 200: #`
	`; 10: inch`

`N60 ID=6 EVERY` **`G710`** `$AA_IM[Z]>200 DO POS[Z2]=10`	`; $AA_IM: mm`
	`; 200: mm`
	`; 10: mm`

`N70 ID=7 EVERY` **`G710`** `$AA_IM[Z]>200 DO` **`G700`** `POS[Z2]=10`	`; $AA_IM: mm`
	`; 200: mm`
	`; 10: inch`

#: The unit depends on the parameterized basic system (MD10240 $MN_SCALING_SYSTEM_IS_METRIC) and the measuring system programmed in the part program	

Note

Measuring system and technology cycles

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

2.7.10 Position in specified reference range (POSRANGE)

Function

The `POSRANGE` function can be used to determine whether the current position of an axis is within the tolerance range around a specified reference position.

Note

With modulo axes, the modulo offset is taken into account.

Syntax

```
<Status> POSRANGE(<axis>, <RefPos>, <tolerance>, [<CoordSys>] )
```

Meaning

`<status>`	Function return value Type: BOOL TRUE: The current position of the axis is within the tolerance range. FALSE: The current position of the axis is not within the tolerance range.
`<axis>`	Name of the channel axis Type: AXIS
`<RefPos>`	Reference position Type: REAL
`<Tolerance>`	Permissible tolerance around the reference position Type: REAL The tolerance is specified as an absolute value. The tolerance range results from: Reference position +/- tolerance
`<CoordSys>`	Optional: Coordinate system Type: INT Range of values: 0 = MCS (machine coordinate system) 1 = BCS (basic coordinate system) 2 = SZS (settable zero system) 3 = WCS (workpiece coordinate system)

2.7.11 Starting/stopping axes (MOV)

Function

An axis can be traversed without specifying an end position via the `MOV` command. The axis traverses so long in the specified direction until it is stopped or another traversing direction is specified by a `MOV` command.

Application: Endlessly rotating rotary axes

Syntax

```
MOV[<axis>] = <direction>
```

Meaning

`MOV`	Start traversing motion
`<axis>`	Channel axis name Type: AXIS
`<Direction>`	Traversing direction Type: INT Range of values: `<Direction>` > 0: Positive traversing direction (default: +1) `<Direction>` < 0: Negative traversing direction (default: -1) `<Direction>` = 0: Stop

---

Note

Indexing axis

If an indexing axis is stopped, it stops at the next indexing position.

Technology cycle

The `MOV` command must **not** be used in technology cycles.

See also

Axial feedrate (FA) (Page 80)

2.7.12 Axial feedrate (FA)

An axial feedrate can be specified in a synchronized action via the `FA` command. The axial feedrate is modal.

Examples

Constant feedrate value:

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

Variable feedrate value:

Program code
`ID=1 EVERY $AA_IM[B] > 75 DO POS[U]=100 FA[U]=$AA_VACTM[W]+100`
`IDS=2 WHENEVER $A_IN[1] == 1 DO POS[X]=100 FA[X]=$R1`

Remarks

- The default value for the feedrate of positioning axes is set via axial machine data: MD32060 $MA_POS_AX_VELO (initial setting for positioning axis velocity)
- The axial feedrate can be specified as a linear or revolutional feedrate.
  The feedrate type can be set via the setting data:
  SD43300 $SA_ASSIGN_FEED_PER_REV_SOURCE (revolutional feedrate for positioning axes / spindles)
- The feedrate type can be switched synchronous to the part program via the `FPRAON` and `FPRAOF` commands. Refer to:
  References:
  /FB1/ Function Manual Basic Functions; Feedrates (V1)

> **Note**
>
> So that technology cycles executed in parallel do not obstruct each other, the axial feedrate from synchronized actions is not output as an auxiliary function to the NC/PLC interface.

See also

Starting/stopping axes (MOV) (Page 79)

2.7.13 Axis replacement (GET, RELEASE, AXTOCHAN)

Command axes can be interchanged between channels via the `GET` and `RELEASE` commands.

Note

The command axis must be assigned to the channel via machine data.

Syntax

```
GET(<axis 1> [{, <axis n>}])

RELEASE((<axis 1> [{, <axis n> }])
```

Axis type and axis status regarding axis replacement

The axis type and axis status currently valid at the time of the synchronized action activation, can be queried via the $AA_AXCHANGE_TYP or $AA_AXCHANGE_STAT system variable. Depending on the channel that has the current interpolation authorization for this axis and depending on the status for the permissible axis replacement, a different sequence results from the synchronized action.

An axis can be requested with `GET` from a synchronized action, if

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

Note

Supplementary condition: An "axis controlled exclusively by the PLC" or a "permanently assigned PLC axis" cannot be assigned to the part program.

An axis can be released from a synchronized action with `RELEASE`, if the axis:

- Was previously assigned to the part program of the channel.
- Is already in the neutral axis state.
- Already has another channel that has the interpolation authorization of this axis

Request axis from another channel

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

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

Requested axis was already requested as neutral axis:

$AA_AXCHANGE_TYP[<axis>]==6, the axis is requested for the part program $AA_AXCHANGE_TYP[axis] == 5 and assigned as soon as possible to the part program of the channel $AA_AXCHANGE_TYP[axis] == 0.

Note

This assignment **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 time 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 part program $AA_AXCHANGE_TYP[axis]==0.

This **results in a** reorganization procedure.

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

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

MD10722 $MN_AXCHANGE_MASK (Parametring the axis replacement behavior)

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

Axis is active as command axis / assigned to the PLC

If the axis is active as a command axis or oscillating axis or a concurrent positioning axis (PLC axis) ($AA_AXCHANGE_TYP[<axis>] == 1), the axis is requested as a neutral axis ($AA_AXCHANGE_TYP[<axis>] == 8). Depending on the setting in the following machine data, the axis is blocked for an automatic axis replacement between channels:

MD10722 $MN_AXCHANGE_MASK (Parametring the axis replacement behavior)

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

With a further `GET` request, the axis is then requested for the part program ⇒ $AA_AXCHANGE_TYP[axis] == 7.

Axis already assigned to the NC program of the channel

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

Release axis for axis replacement

If the axis is assigned to the part program at the time of release ($AA_AXCHANGE_TYP[<axis>] == 0), it is transferred to the neutral axis state ($AA_AXCHANGE_TYP[<axis>] == 3) and if required, released for axis replacement in another channel.

This **results in a** reorganization procedure.

Axis to be released is already a neutral axis:

If the axis is already in the neutral axis state ($AA_AXCHANGE_TYP[<axis>] == 3) or active as command or oscillating axis or assigned to the PLC as concurrent positioning axis ($AA_AXCHANGE_TYP[<axis>] == 1), the axis is released for an automatic axis replacement between channels.

$AA_AXCHANGE_STAT[<axis>] is reset from 1 to 0 if there is no other reason to link the axis to the channel. Such a link of the axis is present, for example, with:

- Active axis coupling
- Active fast retraction
- Active transformation
- JOG request
- Rotating frame with PLC, command or oscillating axis motion

Another channel already has the interpolation authorization

If another channel already has the interpolation authorization ($AA_AXCHANGE_TYP[<axis>] == 2), there is no state change. This also means that waiting for an axis, triggered by part program ($AA_AXCHANGE_TYP[<axis>] == 5) or a previous `GET` request from a synchronized action ($AA_AXCHANGE_TYP[<axis>] == 6) cannot be aborted by a `RELEASE` from a synchronized action.

Supplementary conditions

- If several GET and RELEASE requests are programmed for the same axis, they may mutually cancel each other under certain circumstances and only the last respective requests are performed.

  Example:

Programming:	GET(X,Y) RELEASE(Y,Z) GET(Z)
Execution:	GET(X) RELEASE(Y) GET(Z)

- If further commands are programmed in the action part of a synchronized action in addition to GET/RELEASE, there is no waiting period until the GET/RELEASE request is completed before these commands are executed. This can lead to an error if, for example, an axis requested for the positioning motion with GET is not yet available:

  ```
  GET[<axis>] POS[<axis>]
  ```

Example 1: GET and RELEASE as action in synchronized actions in two channels

Requirement: The Z axis must be known in the 1st and 2nd channels

1. Program sequence in the first channel:

Program code	Comment
WHEN TRUE DO RELEASE(Z)	; Z axis becomes neutral
; Read-in disable as long as Z axis is program axis	
WHENEVER $AA_TYP[Z] == 1 DO RDISABLE	
N110 G4 F0.1	
...	
; Z axis returns to status as NC program axis	
WHEN TRUE DO GET(Z)	
; Read-in disable until Z axis is program axis	
WHENEVER($AA_TYP[Z]<>1) DO RDISABLE	
N120 G4 F0.1	
...	
WHEN TRUE DO RELEASE(Z)	; Z axis becomes neutral
; Read-in disable as long as Z axis is program axis	
WHENEVER $AA_TYP[Z] == 1 DO RDISABLE	
N130 G4 F0.1	
...	
N140 START(2)	; 2. Start channel
N150 ; See below: "3. Continuation: Program sequence in the first channel"	

2. Program sequence in the second channel:

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

3. Continuation: Program sequence in the first channel:

Program code	Comment
N150 WAITM(10,1,2)	; Synchronize with channel 2
...	
WHEN TRUE DO GET(Z)	; Move Z axis to this channel
; Read-in disable as long as Z axis is in other channel	
WHENEVER $AA_TYP[Z] == 0 DO RDISABLE	
N160 G4 F0.1	
...	
N199 WAITE(2)	; Wait for end of program in channel 2
N999 M30	

Transfer axis to another channel (AXTOCHAN)

An axis can be requested for a channel from a synchronized action with the AXTOCHAN command. This does not have to be its own channel that currently has the interpolation authorization for the axis. This means that it is possible to shift an axis into another channel.

If the axis is already assigned to the part program of the channel ($AA_AXCHANGE_TYP[<axis>] == 0), there is **no** state change.

If an axis is requested for the same channel from a synchronized action, AXTOCHAN is mapped on the GET command.

- With the **first** request for the same channel, the axis becomes a neutral axis.
- With the **second** request, the axis is assigned to the part program.

Supplementary condition

A "PLC-controlled axis" corresponds to a "concurrent positioning axis" where special supplementary conditions must be carefully observed. See also:

References:

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

Note

A PLC axis cannot replace the channel.

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

2.7.14 Traversing spindles (M, S, SPOS)

Spindles can be started, positioned and stopped via synchronized actions. The programming is performed in the action part of the synchronized action with the same syntax as in the part program. Without numeric extension the commands for the master spindle apply. By specifying a numeric extension, it is possible to program each spindle individually:

Program code	Comment
`ID = 1 EVERY $A_IN[1]==1 DO M3 S1000`	; Master spindle
`ID = 2 EVERY $A_IN[2]==1 DO SPOS=270`	; Master spindle
`ID = 1 EVERY $A_IN[1]==1 DO M1=3 S1=1000 SPOS[2]=90`	

If concurrent commands are specified for a spindle through synchronized actions that are active in parallel, the chronological sequence decides the activation.

User-specific spindle enable

The start of spindle motions at defined times can be achieved via synchronized actions by blocking the motion programmed in the part program.

Example:

The spindle is programmed within a part program and should not start at the beginning of the block, but only when input 1 is set. The synchronized action holds the spindle override at 0% until the enable via input 1. See Section "Override ($A...OVR) (Page 31)".

Program code

```
; As long as input 1 is not set => spindle override = 0%
ID=1 WHENEVER $A_IN[1]==0 DO $AA_OVR[S1]=0
...
; The start of the spindle is triggered
; The spindle is enabled when input 1 is set
G01 X100 F1000 M3 S1=1000
```

Transition between command axis and spindle

Since several synchronized actions can be active simultaneously, the situation may arise where a spindle motion is started when the spindle is already active. In this case, the most recently activated motion is applicable. At a reversal in the direction of motion, the spindle is first braked and then traversed in the opposite direction.

Direction of rotation, speed and position can also be changed during the motion.

Examples

Program code	Comment
`ID=1 EVERY $AC_TIMER[1] >= 5 DO M3 S300`	`; Speed and direction of rotation`
`ID=2 EVERY $AC_TIMER[1] >= 7 DO M4 S500`	`; Speed and direction of rotation`
`ID=3 EVERY $A_IN[1]==1 DO S1000`	`; Speed`
`ID=4 EVERY ($A_IN[4]==1) AND ($A_IN[1]==0) DO SPOS=0`	`; Spindle positioning`

Transitions between axis and spindle

In state ↓	To →	POS	MOV<>0	MOV=0	SPOS	M3/M4	M5	LEADON	TRAIL ON
during traversing									
	Axis	x	x	x	x	x	x	x	x
	Position-controlled spindle	x	x	x	x	x	x	-	-
	Speed-controlled spindle	-	-	-	x	x	x	-	-
in motion									
	Axis	x	x	x	-	-	-	x	x
	Position-controlled spindle	-	-	-	-	-	-	-	-
	Speed-controlled spindle	-	-	-	x	x	x	-	-
Transitions marked with x are permitted: The transitions marked with - are rejected with an alarm.									

See also

Couplings (CP..., LEAD..., TRAIL..., CTAB...) (Page 92)

2.7.15 Withdrawing the enable for the axis container rotation (AXCTSWEC)

Function

Using the command AXCTSWEC an already issued enable signal to rotate the axis container can be withdrawn again. The command triggers a preprocessing stop with reorganization (STOPRE).

The following conditions must be fulfilled so that in the channel, the enable signal to rotate the axis container is withdrawn again:

- In the channel, the axis container rotation must already have been enabled:
  - AXCTSWE(<container>)
  - $AC_AXCTSWA[<container>] == 1
- Axis container rotation was still not started:
  - $AN_AXCTSWA[<container>] == 0

As feedback signal for the successful withdrawal of the enable signal, the following channel-specific system variable is reset:

$AC_AXCTSWA[<container>] == 0

For a detailed description of the system variables, refer to:

References:
Parameter Manual System Variables

Syntax

```
DO AXCTSWEC(<container>)
```

Meaning

AXCTSWEC:	Withdrawing the enable for the axis container rotation for the channel
<Container>:	Name of axis container: Possible data include: • CT<container number>: The number of the axis container is attached to the CT letter combination. Example: CT3 • <container name>: Individual name of the axis container set using MD12750 $MN_AXCT_NAME_TAB. Example: A_CONT3 • <Axis name>: Axis name of a container axis known in the channel.

Example

Program code	Comment
; Initialization of the global counter for the technology cycle CTSWEC	
N100 $AC_MARKER[0]=0	
N110 ID=1 DO CTSWEC	; For technology cycle CTSWEC, see below.
NEXT:	
N200 G0 X30 Z1	
N210 G95 F.5	
N220 M3 S1000	
N230 G0 X25	
N240 G1 Z-10	
N250 G0 X30	
N260 M5	
; Enable of the axis container rotation for container spindle S1.	
N270 AXCTSWE(S1)	
N200 GOTO NEXT	

Program code	Comment
PROC CTSWEC(STRING _ex_CT="CT1"	
INT _ex_CTsl_BITmask=1H	
INT _ex_CT_SL_Number=1	
INT _ex_WAIT_number_of_IPOs=1000	
) DISPLOF ICYCOF	
DEFINE _ex_number_of_IPOs AS $AC_MARKER[0]	
IF ($AC_STOP_COND[0] + $AC_STOP_COND[1] + $AC_STOP_COND[2] + $AC_STOP_COND[3] +	
$AC_STOP_COND[4] + $AC_STOP_COND[5] + $AC_STOP_COND[6] + $AC_STOP_COND[7] +	
$AC_STOP_COND[8] + $AC_STOP_COND[9] + $AC_STOP_COND[10]) > 0)	
; Increment IPO cycle counter	
_ex_number_of_IPOs = _ex_number_of_IPOs + 1	
; If a stop condition for longer than "_ex_WAIT_number_of_IPOs"	
; IPO cycles is present AND its own slot has not been enabled	
IF (_ex_number_of_IPOs >= _ex_WAIT_number_of_IPOs) AND	
($AN_AXCTSWEC[_ex_CT] == _ex_CTsl_BITmask)	
AXCTSWEC	; Cancel the enable of the axis container rotation.
ENDIF	
ELSE	
; Reset IPO cycle counter	
_ex_number_of_IPOs = 0	
ENDIF	
RET	

Supplementary condition

Time of execution of synchronized actions

Program code
; Enable of the axis container rotation.
N10 AXCTSWE(CT3)
; Traversing of the container axis AX_A => before the axis is traversed, there
; is a waiting period for the end of the axis container rotation: $AN_AXCTSWA[CT3]==0
N20 AX_A = 10
; Cancellation of the enable. **No effect!**
WHEN <condition> DO AXCTSWEC(AX_A)
N30 G4 F1

Because after the enable of the axis container rotation in block `N10`, an axis of the axis container (`AX_A`) is used in block `N20` and this use leads to the system waiting for the end of the axis container rotation, the synchronized action only comes together with the program block `N30` in the main run and has therefore no effect.

Remedy:

Program code	Comment
; Enable of the axis container rotation.	
N11 AXCTSWE(CT3)	
; Cancellation of the enable.	
WHEN <condition> DO AXCTSWEC(AX_A)	
N21 ...	; Executable NC block
; Traversing of the container axis AX_A => before the axis is traversed, there	
; is a waiting period for the end of the axis container rotation: $AN_AXCTSWA[CT3]==0	
N31 AX_A = 10	

Note

Without the executable block `N21`, the synchronized action would only be implemented after the end of the axis container rotation with the next executable program block `N31` in the main run and would therefore have no effect, just the same as in the example above.

2.7.16 Set actual value (PRESETON)

Function

The actual value can be redefined in the machine coordinate system (MCS) for machine axes with `PRESETON`. In this mode, the axes are not traversed.

`PRESETON` is possible in synchronized actions for the following axis types:

- Modulo rotary axes that were started from the part program
- Command axes that were started from synchronized actions

> **CAUTION**
>
> **Loss of machine coordination**
>
> By setting a new actual value in the machine coordinate system, the reference point of the machine axis becomes invalid. For this reason it is recommended that `PRESETON` only be used for axes that do not require a reference point.
>
> To restore the original machine coordinate system, the measuring system of the machine axis must be referenced again, e.g. through active referencing from the part program (`G74`).

Syntax

```
WHEN | EVERY ... DO PRESETON(<axis>,<value>)
```

Meaning

`PRESETON`:	Preset actual-value memory	
`<axis>`:	Machine axis name	
	Range of values:	Machine axis names defined in the channel
`<value>`:	New actual value of the machine axis in the machine coordinate system (MCS)	

Example

Program code

```
...
N10 G1 X=10 Y=12 F5000
N20 WHEN TRUE DO G71 POS[X]=200
; IF setpoint position (MCS) of the X axis (command axis) >= 80 mm
; THEN actual-value offset by +70 mm => axis traverses to 270 mm
N22 WHEN G71 $AA_IM[X] >= 80 DO PRESETON(X, $AA_IM[X]+70)
N24 G4 F3
...
```

Supplementary conditions

- `PRESETON` must **not** be applied to axes involved in a transformation.
- `PRESETON` must only be used in conjunction with `WHEN` or `EVERY`.
- In contrast to the programming in the part program, `PRESETON` can only be programmed for one axis in each synchronized action.

Operating modes

- JOG mode: `PRESETON` is only possible for stationary axes.
- JOGREF mode: `PRESETON` is **not** possible.

Axis replacement (only 840D sl)

`PRESETON` must only be executed in a channel that has the interpolation authorization for this axis. The axis is **not** requested from another channel via axis replacement.

See also

On-the-fly parting (Page 137)

2.7.17 Couplings (CP..., LEAD..., TRAIL..., CTAB...)

The commands listed in Section "Language elements for synchronized actions and technology cycles (Page 52)" can be programmed in synchronized actions for the functions coupled motion (`TRAIL...`), curve tables (`CTAB...`), master value coupling (`LEAD...`) and generic coupling (`CP...`):

Note

Generic coupling

Note that the "generic coupling" `CP ...` commands are always executed in synchronized actions in the sequence of the programming from left to right. This means that in contrast to the programming in the part program, the effect of the various commands depends on their sequence in the synchronized action.

Curve tables

The `CTAB` and `CTABINV` commands can be used in the condition and in the action.

References

Detailed information on coupling commands can be found in:

- Coupled motion, curve tables, master value coupling:

  Programming Manual, Job Planning; Section "Axis couplings"

- Generic coupling

  Description of Functions, Special Functions, Section "Axis couplings (M3)" > "Generic coupling"

Coupled motion

When the coupling is activated from the synchronized action, the leading axis can be in motion. In this case the following axis is accelerated up to the set velocity. The position of the leading axis at the time of synchronization of the velocity is the starting position for coupled-axis motion.

Master value coupling

Syntax

```
... DO LEADON(<FA>, <LA>, <NO>, <OVW>)
```

Meaning

`<FA>`:	Name of the following axis Type: AXIS
`<LA>`:	Name of the leading axis Type: AXIS
`<NO>`:	Number of the curve table Type: INT
`<OVW>`:	Status of the overwrite permission Type: BOOL 0: Overwriting of the table is **not** permitted 1: Overwriting of the table is permitted

- Synchronized actions can be used to change the basic curve table without a resynchronization even during an active master value coupling.

  The following axis attempts as fast as possible to follow the position values specified by the new curve table.

- In order to be able to program an axis to be coupled via synchronized actions, the axis must first be released with the `RELEASE` command.

  Example:

Program code
`...`
`N60 RELEASE(X)`
`N50 ID=1 EVERY SR1==1 DO LEADON(C, X, 1)`

Example: On-the-fly parting

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 the extruded material (MCS)
- Y axis: Axis in which the cutting tool "tracks" the extruded material

It is assumed that the infeed and control of the cutting tool are controlled via the PLC user program. The signals at the PLC interface can be evaluated to determine whether the extruded material and cutting tool are synchronized.

Program code	Comment
N100 R3=1500	; Length of a part to be cut off
N200 R2=100000 R13=R2/300	
N300 R4=100000	
N400 R6=30	; Start position Y axis
N500 R1=1	; Start condition for conveyor axis
N600 LEADOF(Y,X)	; Delete coupling
N700 CTABDEF(Y,X,1,0)	; Table definition
N800 X=30 Y=30	; Value pairs
N900 X=R13 Y=R13	
N1000 X=2*R13 Y=30	
N1100 CTABEND	; End of table definition
N1200 PRESETON(X1,0)	; PRESET at beginning
N1300 Y=R6 G0	; Start position Y axis, axis is linear
; PRESET after length R3, new start after parting	
N1400 ID=1 WHENEVER $AA_IW[X]>$R3 DO PESETON(X1,0)	
N1500 RELEASE(Y)	
; Couple Y to X via table 1, for X < 10	
N1800 ID=6 EVERY $AA_IM[X]<10 DO LEADON(Y,X,1)	
; > 30 before traversed parting distance, deactivate coupling	
N1900 ID=10 EVERY $AA_IM[X]>$R3-30 DO LEADOF(Y,X)	
N2000 WAITP(X)	
; Set extruded material axis continuously in motion	
N2100 ID=7 WHEN $R1==1 DO MOV[X]=1 FA[X]=$R4	
N2200 M30	

Generic coupling

- When a coupling module is activated in a synchronized action, the following axis **must** already be active in the channel and be in the state “neutral axis ” or “axis already assigned to the part program of the channel”. The corresponding axis state can be generated, if necessary, in the synchronized action by programming `GET[<following axis>]`.
- The commands of the generic coupling `CP ...` are processed directly in synchronized actions by the coupling module. The command therefore takes effect immediately.
- With the programming of a coupling factor (`CPLNUM`, `CPLDEN`) or table number (`CPLCTID`), a previously activated non-linear coupling relationship, e.g. a curve table, is deactivated.

Generic coupling: Using the TRAIL, LEAD, EG or COUP coupling type.

If in the framework of the generic coupling, a behavior corresponding to one of the known coupling types "Coupled motion", "Master value coupling", "Electronic gear" or "Synchronous spindle" is required, the command `CPSETTYPE` is also possible in synchronized actions when creating or defining the coupling module:

```
CPSETTYPE[FAx] = <coupling type>
```

<Coupling type>	Meaning
CP	Freely programmable
TRAIL	"Coupled motion" coupling type
LEAD	"Master value coupling" coupling type
EG	"Electronic gearbox" coupling type
COUP	"Synchronous spindle" coupling type

Supplementary conditions

Synchronism status of a following axis

The system variable $AA_SYNC[<axis>] can be used to read the synchronism status of a following axis in the part program or synchronized action.

Axis replacement with cross-channel coupling

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

Note

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

See also Section "Axis replacement (GET, RELEASE, AXTOCHAN) (Page 81)"

Conflict prevention when changing from following axis to channel axis

In order to be able to traverse a following axis traversed via synchronized actions as a channel axis again, you must ensure that the coupling is deactivated before the channel requests the relevant axis.

The following example shows an **error case**:

Program code
`...`
`N50 WHEN TRUE DO TRAILOF(Y, X)`
`N60 Y100`

The Y axis is not released early enough in `N50` because `TRAILOF` only becomes active with `N60` through the non-modal synchronized action.

Corrected example:

Program code	Comment
`...`	
`N50 WHEN TRUE DO TRAILOF(Y, X)`	
`N55 WAITP(Y)`	`; Wait for end of travel of the positioning axis`
`N60 Y100`	

Examples

Define coupling: Y = following axis, X = leading axis

Program code
`... DO CPLDEF[Y]=X CPLNUM[Y,X]=1.5`

Activate coupling and define coupling relationship.

- N10 with the correct sequence: First `CPLON` then `CPLNUM`
- N20 with **incorrect** sequence: First `CPLNUM` then CPLON

Program code	
`N10 ... DO CPLON[Y]=X CPLNUM[X,Y]=1.5`	
`N20 ... DO CPLNUM[X,Y]=2 CPLON[Y]=X`	`; Error`

Activate coupling, deactivation/activation with implicit resynchronization

Program code
`N10 ... DO CPLON[X]=Y CPLNUM[X,Y]=3`
`N20 Y100 F100`
`N30 ... DO CPLOF=X CPLON[X]=Y CPLNUM[X,Y]=3`

Activate coupling, deactivate and traverse as a command axis

Program code
`N10 ... DO CPLON[X]=Y CPLNUM[X,Y]=3`
`N20 Y100 F100`
`N30 ... DO CPLOF=X MOV[X]=10`

2.7.18 Measurement (MEAWA, MEAC)

The following commands can be used in synchronized actions for measurement:

- `MEAWA` (measurement without delete distance-to-go)
- `MEAC` (continuous measurement without delete distance-to-go)

While the measuring function in the part program is limited to one motion block, the measuring function can be switched on and off any number of times from synchronized actions.

Note

Measurement can also be performed in JOG mode via static synchronized actions `IDS ....`

References

Detailed information on measuring commands can be found in:

- Coupled motion, curve tables, master value coupling:

  Programming Manual, Job Planning; Section "Axis couplings"
- Generic coupling

  Description of Functions, Special Functions, Section "Axis couplings (M3)" > "Generic coupling"

Measurement tasks and state changes

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

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

Remarks

System variables

The following system variables can be used in conjunction with synchronous actions:

- $AA_MEAACT (axial measuring active)
- $A_PROBE (probe state)
- $AA_MM1 ... 4 (probe position 1st to 4th trigger (MCS))

The following system variable **cannot** be used in conjunction with synchronized actions:

- $AC_MEA (probe has responded)

Measurement job

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

Priority with more than one measurement

A new measurement task for the same axis has the effect that the trigger events are reactivated and the measurement results reset.

Measurement jobs started from the part program cannot be influenced from synchronized actions. If a measurement task is started from a synchronized action for an axis for which a measurement task is already active from the part program, an alarm is displayed.

If a measurement task is already active from a synchronized action, measurement can no longer be started from the part program.

Saving measurement results

A FIFO memory is set up in the $AC_FIFO system variables to save the measurement results. See Section "FIFO variables ($AC_FIFO) (Page 27)".

Examples

In the following examples, two FIFO memories are set up via machine data:

- MD28050 $MC_MM_NUM_R_PARAM = 300
- MD28258 $MC_MM_NUM_AC_TIMER = 1
- MD28260 $MC_NUM_AC_FIFO = 1 (set up FIFO memory)
- MD28262 $MC_START_AC_FIFO = 100 (FIFO memory starts from R100)
- MD28264 $MC_LEN_AC_FIFO = 28 (22 variables + 6 management data)
- MD28266 $MC_MODE_AC_FIFO = 0 (no summation)

Example 1

All rising edges of probe 1 are to be recorded between 0 and 100 mm for the X axis. It is assumed that no more than 22 measuring edges occur.

Program code	Comment
DEF INT NUMBER	; Number of current measured values
DEF INT INDEX_R	; Loop index
N10 G0 X0	; Approach starting point for the measurement
;Measurement: Mode = 1 (simultaneously), FIFO memory = 1,	
; trigger event = 1 (rising edge of probe 1)	
N20 MEAC[X]=(1, 1, 1) POS[X]=100	
N30 STOPRE	; Stop preprocessing
N40 MEAC[X]=(0)	; Cancel measuring job
N50 ANZAHL=$AC_FIFO1[4]	; Number of saved measured values
N60 ANZAHL = ANZAHL - 1	
N70 FOR INDEX_R=0 TO ANZAHL	
N80 R[INDEX_R]=$AC_FIFO1[0]	; Save measured value in R parameter
N90 ENDFOR	

Example 2

All rising and falling edges of probe 1 are to be recorded between 0 and 100 mm for the X axis. The number of measurements is not known. Therefore, the measured values must be fetched parallel to the measurement and stored in ascending order as of $R1. The number of stored measured values is entered in $R0.

Program code

```
$AC_MARKER[1]=1                              ;   Initialize index for R parameter index
N10 G0 X0                                    ;   Approach starting point for the measurement
; If measured values are available in the FIFO memory, the oldest value is read and
; stored in the current R parameter[$AC_MARKER[1]].
; The R parameter index is then incremented.
N20 ID=1 WHENEVER $AC_FIFO1[4] >= 1 DO $R[$AC_MARKER[1]] = $AC_FIFO1[0]
 $AC_MARKER[1] = $AC_MARKER[1] + 1
; Continuous measurement: Mode = 1 (simultaneously), FIFO memory = 1,
; trigger event 1 = 1 (rising edge of probe 1),
; trigger event 2 = -1 (falling edge of probe 1)
N30 MEAC[X]=(1, 1, 1, -1) POS[X]=100
N40 MEAC[X]=(0)                              ;   Turn measurement off
N50 STOPRE                                   ;   Stop preprocessing
N60 R0 = $AC_MARKER[1]                       ;   Number of recorded measured values
```

Example 3

Rising and falling edges of probe 1 are to be recorded between 0 and 500 mm for the X axis. The number of measurements is limited to 10.

The distance-to-go of the X axis is then deleted.

Program code

```
N10 G0 X0                                 ;   Approach starting point for the measurement
; Abort condition: Deselect continuous measurement after 10 or more measurements
; and perform "delete distance-to-go"
N10 WHEN $AC_FIFO1[4] >= 10 DO MEAC[X]=(0) DELDTG(X)
; Continuous measurement: Mode = 1 (simultaneously), FIFO memory = 1,
; trigger event 1 = 1 (rising edge of probe 1),
; trigger event 2 = -1 (falling edge of probe 1)
N20 MEAC[X]=(1, 1, 1, -1) G01 X100 F500
N30 MEAC [X]=(0)                          ;   Turn measurement off
N40 R0 = $AC_FIFO1[4]                     ;   Number of recorded measured values
```

2.7.19 Travel to fixed stop (FXS, FXST, FXSW, FOCON, FOCOF, FOC)

Function

Travel to fixed stop

The function "Travel to fixed stop" can be controlled via synchronized actions with the `FXS`, `FXST` and `FXSW` commands.

The activation can also be performed without traversing motion of the relevant axis. The torque is immediately limited. The fixed stop is monitored as soon as the axis is traversed.

Travel with limited torque/force

Travel with limited torque/force can be controlled via synchronized actions with the `FOCON`, `FOCOF` and `FOC` commands.

Syntax

```
FXS[<axis>]=<request>
FXST[<axis>]=<clamping torque>
FXSW[<axis>] = <window width>
FOCON[<axis>]
FOCOF[<axis>]
FOC[<axis>]
```

Meaning

Parameter	Meaning
`FXS`:	Travel to fixed stop
`<Request>`:	Request to the "Travel to fixed stop" function: 0 = switch off 1 = switch on
`FXST`:	Set clamping torque
`<Clamping torque>`:	Clamping torque as % of the maximum drive torque
`FXSW`:	Set monitoring window
`<Window width>`:	Width of the tolerance window around the fixed stop Unit: mm, inch or degrees
`FOCON`:	Switch on modal torque/force limitation
`FOCOF`:	Switch off modal torque/force limitation
`FOC`:	Non-modal torque/force limitation
`<axis>`:	Name of the channel axis on which the command will be applied

Remarks

Avoidance of multiple selection

The "Travel to fixed stop" function must only be switched on once per axis. In the event of an error, alarm 20092 is displayed and the corresponding alarm response takes effect.

To avoid multiple selections, it is recommended that a selection marker be used in the synchronized action.

Example:

```
Program code                                Comment
N10 R1=0                                    ;   Initialize selection marker
...
N20 IDS=1 WHENEVER ($R1==0 AND $AA_IW[AX3] > 7) DO $R1=1 FXS[AX1]=1
```

Switching on during the approach motion

"Travel to fixed stop" can also be switched on during the approach motion through a non-modal synchronized action.

Example:

```
Program code                                Comment
N10 G0 G90 X0 Y0                            ;   Approach initial setting
...
; "Travel to fixed stop" is switched on for the X axis,
; as soon as the position setpoint in the WCS is > 20 mm
; Execution of the non-modal synchronized action: With N30
N20 WHEN G71 $AA_IW[X] > 20 DO FXS[X]=1
N30 G1 F200 X100                            ;   Traversing block of the X axis
```

Example: Travel to fixed stop completely via synchronized actions

Program code	Comment
; IF selection request $R1==1 AND state of the Y axis == "not to fixed stop"	
; THEN: For the Y axis:	
; - Switch on FXS	
; - Traverse to position 150 mm	
; - Reduce drive torque to 10%	
IDS=1 WHENEVER G71 (($R1==1) AND $AA_FXS[y]==0)) DO $R1=0 FXS[Y]=1 FXST[Y]=10 FA[Y]=200 POS[Y]=150	
...	
; IF state of the Y-Axis == "Fixed stop has been detected"	
; THEN: Increase drive torque to 30%	
IDS=2 WHENEVER ($AA_FXS[Y]==4) DO FXST[Y]=30	
...	
; IF state of the Y axis == "Successful travel to fixed stop"	
; THEN: Set drive torque in accordance with setting $R0	
IDS=3 WHENEVER ($AA_FXS[Y]==1) DO FXST[Y]=$R0	
...	
; Deselection depending on R3 and retract.	
IDS=4 WHENEVER (($R3==1) AND $AA_FXS[Y]==1)) DO FXS[Y]=0 FA[Y]=1000 POS[Y]=0	
...	
N10 R1=0 FXS[Y]=0 G0 G90 Y0	; Initialization
N30 RELEASE(Y)	; Enable Y axis for traversing in synchronized actions
N50 ...	
N60 GET(Y)	; Include Y axis in the path group again

2.7.20 Channel synchronization (SETM, CLEARM)

Synchronization markers can be set and deleted in the channel in which the synchronized action runs with the `SETM` and `CLEARM` commands.

Syntax

```
SETM(<No_marker 1> [,<No_marker 2> {, ... < No_marker n>} ] )
CLEARM(<No_marker 1> [,<No_marker 2> {, ... < No_marker n>} ] )
```

Meaning

A detailed description of the `SETM` and `CLEARM` commands can be found in:

References

Programming Manual, Job Planning; Section "Flexible NC programming" > "Program coordination (INIT, START, WAITM, WAITMC, WAITE, SETM, CLEARM)"

2.7.21 User-specific error reactions (SETAL)

Synchronized actions can be used to react user-specifically to application-specific error states. Possible reactions are:

- Axis with stop via override = 0%
- Display user-specific alarm
- Set digital output

Display alarm

Syntax

```
SETAL(<Alarm_no>[,"Alarm text"])
```

Meaning

Parameter	Meaning
`<Alarm_no>`:	Alarm number from the range: 65000 - 69999

A complete description of the configuration of user alarms can be found in:

References

Base Software and HMI Advanced Commissioning Manual,
Section "HMI Advanced" > "Configuring the HMI system" > "Configuring user alarms"

Examples

```
; If the distance between axes X1 and X2 is less than 5 mm =>
; stop axis X2
ID=1 WHENEVER G71 ($AA_IM[X1]-$AA_IM[X2])<5.0 DO $AA_OVR[X2]=0

; If the distance between axes X1 and X2 is less than 5 mm =>
; display alarm 65000
ID=1 WHENEVER G71 ($AA_IM[X1]-$AA_IM[X2])<5.0 DO SETAL(65000)
```

2.8 Technology cycles

2.8.1 General

Definition

A technology cycle is a subprogram that is called in the action part of a synchronized action. All language elements and system variables that are also used in the action part of a synchronized action can be used in a technology cycle. In addition, there are also the following language elements that may only be used within a technology cycle:

- Section "System variables for synchronized actions (Page 17)"
- Section "User-defined variables for synchronized actions (Page 50)"
- Section "Language elements for synchronized actions and technology cycles (Page 52)"
- Section "Language elements for technology cycles only (Page 59)"
- Section "Actions in synchronized actions (Page 60)"

End of program

The following commands are permitted as end of program: `M02`, `M17`, `M30`, `RET`

Search path

When calling a technology cycle, the same search path is used as for subprograms and cycles.

References
Programming Manual, Job Preparation, Section "Flexible NC programming" > "Subprogram technique" > "General" > "Search path"

Multiple calls

If a condition is fulfilled again while the technology cycle is being executed, the technology cycle is **not** restarted.

If a technology cycle is started because of a fulfilled `WHENEVER` condition and the condition is still fulfilled after completion of the technology cycle, then the technology cycle is started again.

Behavior with non-modal synchronized actions

A non-modal synchronized action is always linked to the next main run block. If the execution time of the technology cycle is longer than the processing time of the associated main run block, the technology cycle is aborted with the block change.

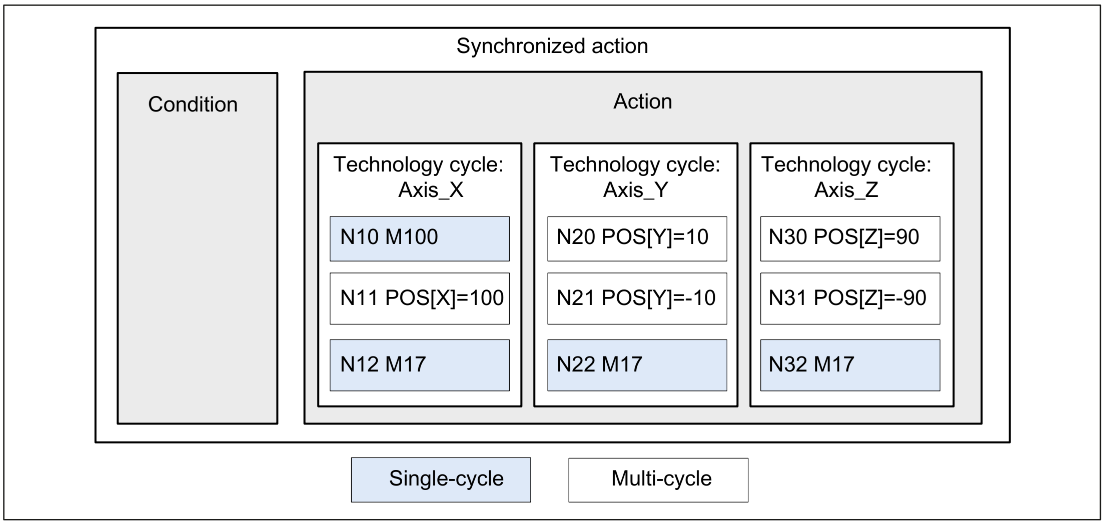
Execution sequence of technology cycles

If several technology cycles are programmed in the action part of a synchronized action, they are executed in the sequence from left to right.

Example:

Call of three technology cycles in the action part of a synchronized action

Program code
`ID=1 <condition part> DO AXIS_X AXIS_Y AXIS_Z`



Execution sequence of the technology cycle blocks: `N10`, `N11`, `N12`, `N20`, `N21`, `N22`, `N30`, `N31`, `N32`

Note

Supplementary conditions

- A maximum of eight technology cycles may be called in the action part of a synchronized action.
- Except for the call of further technology cycles, no other action may be programmed in the action part of a synchronized action in which a technology cycle is called.

See also

Processing mode (ICYCON, ICYCOF) (Page 106)

2.8.2 Processing mode (ICYCON, ICYCOF)

Function

The ICYCOF and ICYCON commands can be used to control the processing mode of the actions within technology cycles.

Per default, the processing mode ICYCON is active.

Processing mode: ICYCON

A non-modal technology cycle is executed in the ICYCON processing mode. The execution of all actions programmed in a block is initiated in the same interpolation cycle. As soon as all initiated actions are completed, the next block is processed in the following interpolation cycle.

A distinction is made between single-cycle and multi-cycle actions. Examples are:

- Single-cycle actions: Auxiliary function output, value assignments
- Multi-cycle actions: Traversing motions of axes and spindles

Each block of a technology cycle requires at least one interpolation cycle.

Processing mode: ICYCOF

All actions of all blocks of a technology cycle are initiated in parallel in the ICYCOF processing mode.

Subprogram as part program

If a subprogram is executed as a part program, the ICYCOF and ICYCON commands have no effect.

Syntax

In the action part of a synchronized action

```
ID=1 <condition part> DO [ICYCOF] <technology cycle 1> [ICYCOF |
ICYCON] <technology cycle 2> ...
```

As a property of a subprogram

```
PROC <name> [ICYCOF | ICYCON]
```

Within a subprogram

```
PROC <name>
  N10 ...
  N20 [ICYCOF | ICYCON]
  N90 ...
  N100 [ICYCOF | ICYCON]
  N110 ...
RET
```

Example

Program code	Effective processing mode	Interpolation cycle
`PROC TECHNOCYC`	ICYCON	
`$R1=1`	ICYCON	1
`POS[X]=100`	ICYCON	2 ... 25
`ICYCOF`	ICYCOF	26
`$R1=2`	ICYCOF	26
`$R2=$R1+1`	ICYCOF	26
`POS[X]=110`	ICYCOF	26
`$R3=3`	ICYCOF	26
`RET`	ICYCOF	26

2.8.3 Definitions (DEF, DEFINE)

If a subprogram is used as a technology cycle that contains commands for the (`DEF`) variables and/or (`DEFINE`) macro definition, these have **no effect** when executing the technology cycle.

Although variables and macro definitions have no effect within a technology cycle, they must nevertheless have the correct syntax. In the event of an error, the execution of the technology cycle is aborted and an alarm displayed.

As the variables and macros are not available in the technology cycle, special measures may have to be taken in the program code. See Section "Context variable ($P_TECCYCLE) (Page 108)".

2.8.4 Parameter transfer

All types of parameter transfer and parameter definition that are possible in subprograms can also be used when the subprogram is used as a technology cycle:

- Call-by-value
- Call-by-reference
- Default parameters

References

A detailed description of the parameter transfer and parameter definition in subprograms can be found in:

Programming Manual, Job Planning, Section "Flexible NC programming" > "Subprogram technique" > "Definition of a subprogram" or "Call of a subprogram"

2.8.5 Context variable ($P_TECCYCLE)

Function

The $P_TECCYCLE system variable can be used to determine within a subprogram whether the subprogram is currently being executed as a part program or technology cycle:

- $P_TECCYCLE == TRUE: Execution as a technology cycle
- $P_TECCYCLE == FALSE: Execution as a part program

If a subprogram is used as a part program and also as a technology cycle, it is therefore possible to determine which program sections are executed as a part program and which as a technology cycle.

Application

The (`DEF`) variables and (`DEFINE`) macro definitions have no effect in technology cycles. If a subprogram is used as a technology cycle that contains the appropriate definitions, a differentiation of cases is required in the program code because the variables and macros are then not available.

Example:
Travel parameters via user variables in the part program and R parameters in the technology cycle

Program code	Comment: Use in
`PROC UP_1`	
`DEF REAL POS_X=100.0`	Part program
`DEF REAL F_X=250.0`	Part program
`IF $P_TECCYCLE==TRUE`	
`$R1=100.0`	Technology cycle
`$R2=250.0`	Technology cycle
`ENDIF`	
`IF $P_TECCYCLE==TRUE`	
`N100 POS[X]=$R1 FA[X]=$R2`	Technology cycle
`ELSE`	
`N200 POS[X]=POS_X FA[X]=F_X`	Part program
`ENDIF`	
`RET`	

See also

Definitions (DEF, DEFINE) (Page 107)

2.9 Protected synchronized actions

Each synchronized action is clearly identified via its `ID`.

The following machine data can be used to define an NC global or channel-specific range of identification numbers with which a synchronized action can be protected against overwriting, deletion (`CANCEL(ID)`) and locking (`LOCK(ID)`):

- NC global:
  MD11500 $MN_PREVENT_SYNACT_LOCK (protected synchronized actions)
- Channel-specific:
  MD21240 $MN_PREVENT_SYNACT_LOCK (protected synchronized actions)

Behavior is the same in both cases.

Protected synchronized actions cannot be locked via the NC/PLC interface or are displayed as non-lockable:

- DB21, ... DBB300 ... 307 (lock synchronized actions)
- DB21, ... DBB308 ... 315 (synchronized actions that can be disabled locked)

Application

The synchronized actions defined by the machine manufacturer to react to certain machine states should not be changed after commissioning.

Note

It is recommended that the protection of synchronized actions should not be activated during the commissioning phase as otherwise a Power on reset is required at each change to the synchronized action.

Example

In a system with two channels, the synchronized actions of the following identification number areas should be protected:

Channel 1: 20 ... 30

Channel 2: 25 ... 35

Machine data configuration

NC-global protection area:

- MD11500 $MN_PREVENT_SYNACT_LOCK[0] = 25
- MD11500 $MN_PREVENT_SYNACT_LOCK[1] = 35

Channel-specific protection area for channel 1:

- MD21240 $MC_PREVENT_SYNACT_LOCK_CHAN[0] = 20
- MD21240 $MC_PREVENT_SYNACT_LOCK_CHAN1] = 30

Channel-specific protection area for channel 2:

- MD21241 $MC_PREVENT_SYNACT_LOCK_CHAN[0] = -1
- MD21241 $MC_PREVENT_SYNACT_LOCK_CHAN[1] = -1

A separate protection was not defined in channel 2 and therefore the NC-global protection area applies.

2.10 Coordination via part program and synchronized action (LOCK, UNLOCK, RESET, CANCEL)

Each modal and static synchronized action must be assigned a unique identification number during the definition:

Program code
`ID=<number> condition part DO action part`
`IDS=<number> condition part DO action part`

By specifying the identification number, synchronized actions from part programs and from synchronized actions can be coordinated via the following commands:

Keyword	Meaning	TP[1)]	SA[2)]
`LOCK(<number>)`:	Lock synchronized action An active positioning action is interrupted.	-	x
`UNLOCK(<number>)`:	Continue interrupted synchronized action An interrupted positioning operation is continued.	-	x
`RESET(<number>)`:	Cancel synchronized action An active positioning action is cancelled. If a technology cycle is restarted, then it is processed from the 1st block in the technology cycle. Depending on the type of the condition, the actions are performed again when the condition is fulfilled again. Already executed synchronized actions with condition `WHEN` are not processed again after RESET.	-	x
`CANCEL(<number>)`:	Delete synchronized action An active positioning action is terminated.	x	-
[1)] Can be programmed in the part program [2)] Can be programmed in a synchronized action / technology cycle			

2.11 Coordination via PLC

With regard to their execution by the NC, synchronized actions that are not protected can be locked. Either all synchronized actions in the channel can be locked together or individually in the ID/IDS 1 - 64 area.

All, channel-specific

Lock all synchronized actions in the channel:

DB21, … DBX1.2 = 1 (synchronized action off)

Individually, channel-specific

Synchronized actions that can be locked

The synchronized actions that can be locked are displayed via:

DB21, … DBX308.0 - 315.7 == 1 (synchronized actions ID/IDS can be locked)

The update of the display must be triggered actively via the following signal from the PLC user program:

DB21, … DBX281.1 = 1 (request: Update synchronized actions that can be locked)

The NC then updates the display of the synchronized actions that can be locked and acknowledges the update by resetting the trigger signal:

DB21, … DBX281.1 = 0 (acknowledgement: Synchronized actions that can be locked updated)

Lock synchronized actions

The corresponding lock bit must be set by the PLC user program for each synchronized action that is to be locked in the channel:

DB21, … DBX300.0 - 307.7 = 1 (lock synchronized action ID/IDS 1 - 64)

The following trigger signal must be set by the PLC user program as a request to the channel to accept the current lock bits:

DB21, … DBX280.1 = 1 (request: Accept synchronized actions to be locked)

The NC then accepts the synchronized actions to be locked in the channel and acknowledges this by resetting the trigger signal:

DB21, … DBX280.1 = 0 (acknowledgement: Synchronized actions to be locked accepted)

See also

Protected synchronized actions (Page 109)

2.12 Configuration

Number of synchronized action elements

The number of synchronized action elements that can be provided per channel is set via the machine data:

MD28250 $MC_MM_NUM_SYNC_ELEMENTS (number of elements for expressions in synchronized actions)

At least four synchronized action elements are required per synchronized action. Further synchronized action elements are required for:

Operation	Number of required elements
Operator in the condition	1
Action	>= 1
Assignment	2
Further operands in complex expressions	1

The number of programmable synchronized actions therefore depends on the number of available synchronized action elements and the complexity of the synchronized actions.

Memory utilization

The status display for synchronized actions can be used to track the memory utilization of the synchronized action memory (see Section "Diagnostics (HMI Advanced only) (Page 118)").

The number of free synchronized action elements can also be read via the system variable $AC_SYNA_MEM.

If more synchronized action elements are required during operation than are available, alarm "14751 Resources for motion synchronous actions not sufficient" is displayed.

Number of FCTDEF elements

The number of `FCTDEF` elements per channel is set via the machine data:

MD28252 $MC_MM_NUM_FCTDEF_ELEMENTS (number of FCTDEF elements)

Synchronized actions and interpolation cycle

If there are a large number of simultaneously active synchronized actions, the interpolation cycle may have to be increased:

MD10070 $MN_IPO_SYSCLOCK_TIME_RATIO

Time required by individual operations

Synchronized action commands	Time required 1)	
	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: WHEN TRUE DO **POS[z]=10**	83 µs	73 µs
1) Measured with SINUMERIK 840D with NCU 573.x		

2.13 Control behavior in specific operating states

2.13.1 Power On

No synchronized actions are active during ramp-up of the NC (Power On).

Synchronized actions that are to be active immediately after the ramp-up of the NC (Power On), must be event-driven as static synchronized actions within an ASUB or activated via the PLC user program.

References

Detailed information on the activation of synchronized actions after ramp-up of the NC (Power On) can be found in:

PLC user program

Function Manual, Basic Functions; PLC Basic Program for SINUMERIK 840D sl
Section "Structure and functions of the basic program" > "Functions of the basic program with call from the user program"

Event-driven

Function Manual, Basic Functions; Mode Group, Channel, Program Operation (K1)
Section "Program operation" > "Event-controlled program calls"

2.13.2 NC reset

State after NC reset:

From:	Modal and non-modal synchronized action (ID)	Static synchronized action (IDS)
Synchronized action	Aborted or inactive	Active
Traversing motion	The traversing motions are aborted	
Speed-controlled spindle	MD35040 $MA_SPIND_ACTIVE_AFTER_RESET = <value> TRUE ⇒ The spindle remains active FALSE ⇒ The spindle is stopped	
Master value coupling	MD20110 $MC_RESET_MODE_MASK, bit 13 = <value> 1 ⇒ The coupling remains active 0 ⇒ The coupling is released	
Measuring	Aborted	

2.13.3 NC stop

Non-modal and modal synchronized actions (ID)

Traversing motions from non-modal and modal synchronized actions are stopped by NC stop.

A non-modal or modal synchronized action also remains active while the channel is in the "interrupted" state:

DB21, ... DBX35.6 == 1 (channel state "interrupted")

If the condition is fulfilled during this time, the actions are executed except for traversing motions.

Stopped traversing motions are continued with NC start.

Static synchronized actions (IDS)

Traversing motions from static synchronized actions are **not** stopped by NC stop.

2.13.4 Operating mode change

Status after operating mode change:

From:	Modal and non-modal synchronized action (ID)	Static synchronized action (IDS)
Synchronized action	Aborted or inactive 1)	Active
Traversing motion	Aborted 2)	Active
Speed-controlled spindle	Active	Active
Master value coupling	MD20110 $MC_RESET_MODE_MASK, bit 13 = <value> 1 ⇒ The coupling remains active 0 ⇒ The coupling is released	Active
Measuring	Aborted	Active
1) The synchronized actions become active again after changing back to the AUTOMATIC mode. 2) End of program `M30` is delayed until the axis is at standstill.		

2.13.5 End of program

State after end of program:

From:	Modal and non-modal synchronized action (ID)	Static synchronized action (IDS)
Synchronized action	Aborted or inactive	Active
Traversing motion	Aborted 1)	Active
Speed-controlled spindle	MD35040 $MA_SPIND_ACTIVE_AFTER_RESET = <value> TRUE ⇒ The spindle remains active FALSE ⇒ The spindle is stopped	Active
Master value coupling	MD20110 $MC_RESET_MODE_MASK, bit 13 = <value> 1 ⇒ The coupling remains active 0 ⇒ The coupling is released	Active
Measuring	Aborted	Active
1) End of program `M30` is delayed until the axis is at standstill.		

2.13.6 Block search

Non-modal and modal synchronized actions (ID)

Synchronized actions are collected during the block search but not activated. I.e. the conditions are not evaluated, the actions are not executed.

The synchronized actions only become active with NC start. I.e. the conditions are evaluated and the actions executed if necessary.

Static synchronized actions (IDS)

Static synchronized actions that are already active remain effective during the block search.

2.13.7 Program interruption by ASUB

Non-modal and modal synchronized actions (ID)

Active modal synchronized actions also remain active during the ASUB.

Traversing motions started from non-modal and modal synchronized actions are interrupted. If at the end of the ASUB, positioning is at the interruption point of the part program (`REPOS`), then the interrupted traversing motions are continued.

Static synchronized actions (IDS)

Static synchronized actions also remain active during the ASUB.

Traversing motions started from static synchronized actions are not interrupted by the ASUB.

Synchronized actions of the ASUB

If the ASUB is `not` continued with REPOS, the modal and static synchronized actions from the ASUB remain effective in the part program.

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

If positioning motions started from synchronized actions are interrupted by the operating mode change or start of the interrupt routine, then they are continued with `REPOS`.

2.13.9 Response to alarms

- If an action of a synchronized action triggers an alarm, this action will be aborted. Other actions of the synchronized action are processed.
- If a modal synchronized action triggers an alarm, it will be inactive after the interrupt time.
- If a technology cycle generates an alarm with motion stop, it will then be aborted and no longer processed.
- If an alarm is triggered with motion stop, all axis/spindle motions, which were started by synchronized actions, will be stopped. Actions without traversing motion are still executed.
- If an alarm is triggered with interpreter stop, it will only have an effect on synchronized actions after complete execution of the predecoded blocks.

2.14 Diagnostics (HMI Advanced only)

Diagnostic functionality

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

- Status display of synchronized actions in the machine operator area
- System variables display parameters in the operating range

  The current values of all synchronized action variables can be displayed (displaying main run variables)
- System variables log parameters in the operating range

  Characteristics of variables can be recorded in the interpolation cycle grid (logging main run variables)

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

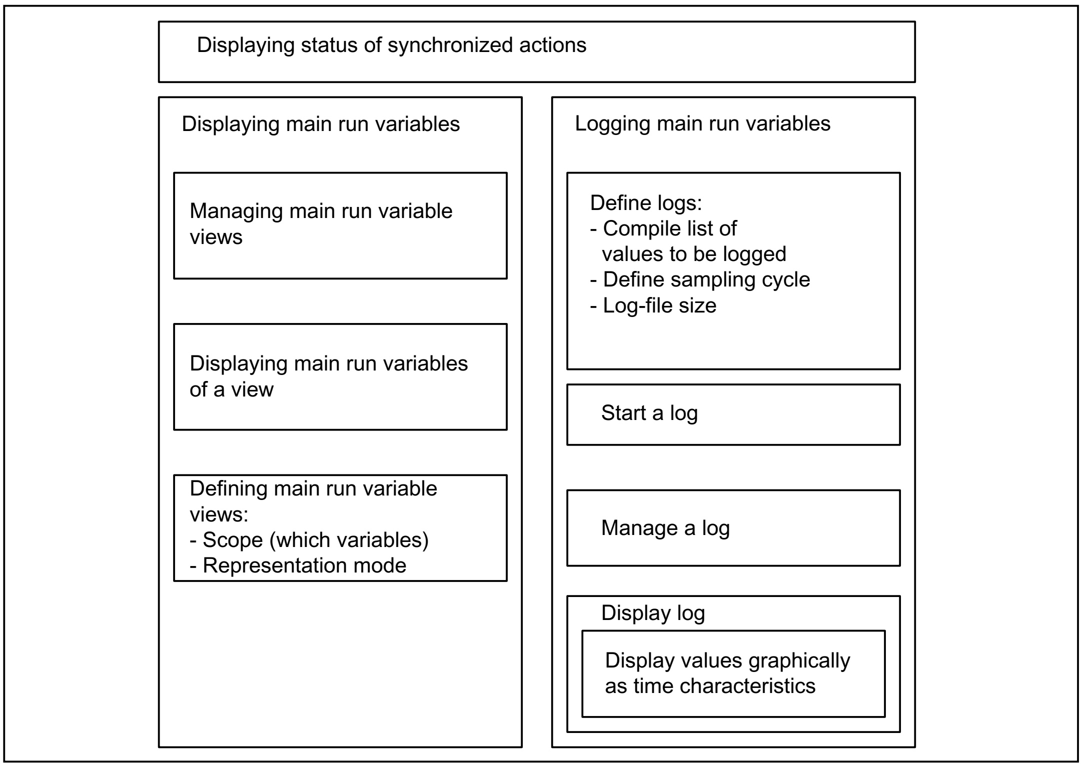


Figure 2-8 Functionality of test tools for synchronized actions

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

References:

/BAD/ Operator's Guide HMI Advanced.

2.14.1 Displaying the status of synchronized actions

The following information is shown on the status display of the synchronized actions:

- Overview of the programmed synchronized actions
- Validity and identification number (only for modal synchronized actions)
  See Section "Validity, identification number (ID, IDS) (Page 12)"
- Status of the synchronized action

Status

Status	Meaning
No status	The condition is being checked in the interpolation cycle
Locked	The synchronized action is locked. See Section: • Coordination via part program and synchronized action (LOCK, UNLOCK, RESET, CANCEL) (Page 110) • Coordination via PLC (Page 111)
Active	The action part of the synchronized action is being executed. If the action consists of a technology cycle, the current block number in this is displayed.

References

Operating Manual, HMI Advanced, Section "Machine operating area" > "General functions and displays" > "Status of the synchronized actions"

2.14.2 Displaying main run variables

Description

System variables can be monitored for the purpose of monitoring synchronized actions. Variables, which may be used in this way are listed for selection by the user.

A complete list of individual system variables with ID code W for write access and R for read access for synchronized actions can be found in:

References:

/PGA1/ Parameter Manual, System Variables

Views

"Views" are provided to allow the user to define the values, which are relevant for a specific machining situation and to determine how (in lines and columns, with what text) these values must be displayed. Several views can be arranged in groups and stored in correspondingly named files.

Managing views

A view defined by the user can be stored under a name of his choice and then called again. Variables included in a view can still be modified (Edit View).

Displaying main run variable of a view

The values assigned to a view are displayed by calling the corresponding user-defined view.

2.14.3 Logging main run variables

Starting point

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

Method

The values defined in a log definition are written to a log file of defined size in the specified cycle. Special functions for displaying the contents of log files are provided.

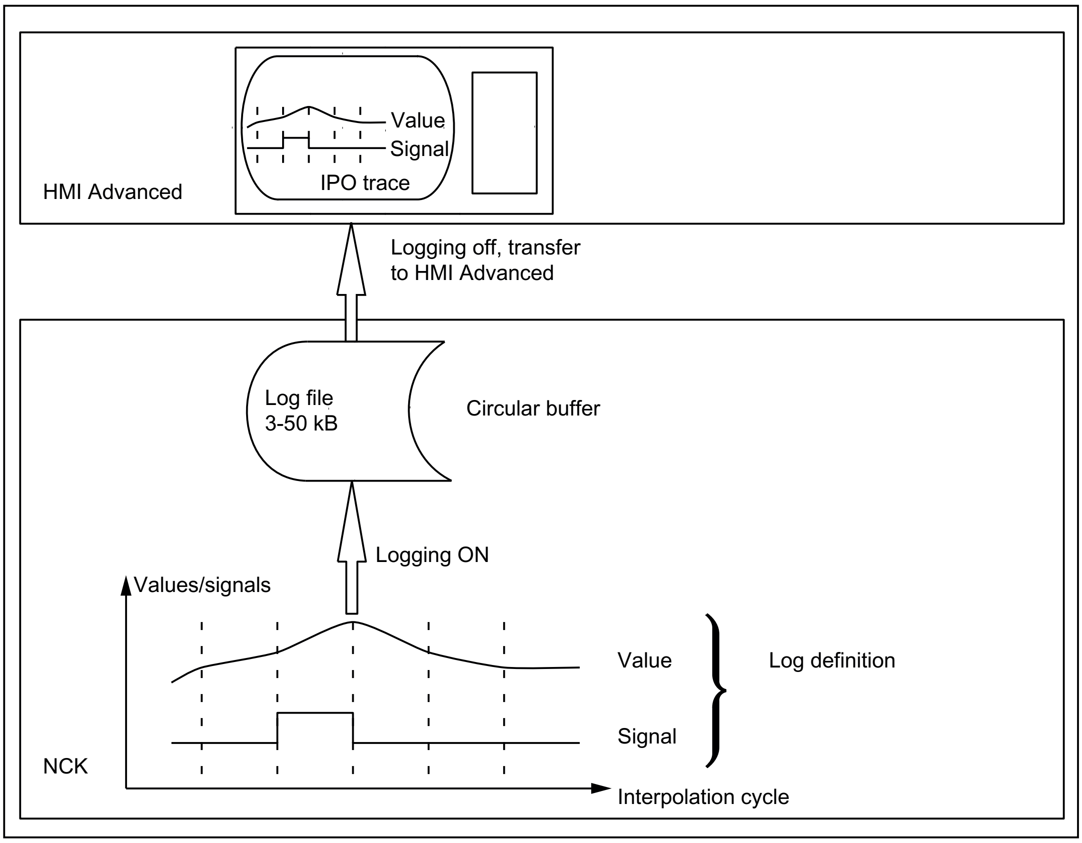


Figure 2-9 Schematic representation of Log main run variables process

Operation

For information about operating the logging function, please see:

References:

/BAD/ Operator's Guide HMI Advanced.

Log definition

The log definition can contain up to 6 specified variables. The values of these variables are written to the log file in the specified cycle. A list of variables, which may be selected for logging purposes, is displayed. The cycle can be selected in multiples of the interpolation cycle. The file size can be selected in Kbytes. A log definition must be initialized before it can be activated on the NCK for the purpose of acquiring the necessary values.

Log file size

Values between 3 KB (minimum) and 50 KB (maximum) can be selected as the log file size.

Storage method

When the effective log file size has been exceeded, the oldest entries are overwritten, i.e. the file works on the circular buffer principle.

Starting logging

Logging according to one of the initialized log definitions is started by:

- Operation
- Setting system variable $A_PROTO=1 from the part program

The starting instant must be selected such that the variables to be logged are not altered until operations on the machine have been activated. The start point refers to the last log definition to be initialized.

Stopping logging

This function terminates the acquisition of log data in the NCK. The file containing the logged values is made available on the HMI for storage and evaluation (graphic log). Logging can be stopped by:

- Operation
- Setting system variable $A_PROTO=0 from the part program

Graphic log function

The measured values (up to 6) of a log are represented graphically as a function of the sampling time. The names of variables are specified in descending sequence according to the characteristics of their values. The screen display is arranged automatically. Selected areas of the graphic can be zoomed.

Note

Graphic log representations are also available as text files on the HMI Advanced. An editor can be used to read the exact values of a sampling instant (values with identical count index) numerically.

Managing logs

Several log definitions can be stored under user-defined names. They can be called later for initialization and start of recording or for modification and deletion.

3 Examples

3.1 Examples of conditions in synchronized actions

Condition	Programming
Path distance-to-go ≤ 10 mm (WCS)	`... WHEN $AC_DTEW <= 10 DO ...`
Distance-to-go of the X axis ≤ 10 mm (WCS)	`... WHEN $AA_DTEW[X]<= 10 DO ...`
Path distance to start of block ≥ 20 mm (BCS)	`...WHEN $AC_PLTBB >= 20 DO ...`
Actual value of the Y axis (MCS) > 10 * SIN(R10)	`... WHEN $AA_IM[y] > 10*SIN (R10) DO...`
Input 1 changes from 0 to 1	`... EVERY $A_IN[1]==1 DO ...`
Input 1 == 1	`... WHENEVER $A_IN[1]==0 DO ...`

3.2 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 by name as in the part program.

Example: Oscillation from synchronized actions

Program code

```
N610 ID=1 WHENEVER $AA_IM[Z] > $SA_OSCILL_REVERSE_POS1[Z] DO $AC_MARKER[1]=0
...
; ALWAYS WHEN current position of the oscillating axis in the MCS < start of reversal area 2,
; THEN override of the infeed axis = 0%
N620 ID=2 WHENEVER $AA_IM[Z] < $SA_OSCILL_REVERSE_POS2[Z] - 6 DO
 $AA_OVR[X]=0 $AC_MARKER[0]=0
...
; ALWAYS WHEN current position of the oscillating axis in the MCS == reversal position 1,
; THEN override of the oscillation axis = 0%, override of the infeed axis = 100%
; This cancels 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
...
; ALWAYS WHEN distance-to-go of the partial infeed == 0,
; THEN override of the oscillation axis = 100%
; This cancels 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
```

Program code

```
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
...
; WHEN current position of the oscillating axis in the WCS == reversal position 1,
; THEN override of the oscillation axis = 100%, override of the infeed axis = 0%
; This cancels 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 changes during machining (e.g. by means of
; operator input or synchronized action) must be programmed with $$S...:
; Example: Oscillation from synchronized actions with change of the oscillation
; position via the user interface
N610 ID=1 WHENEVER $AA_IM[Z] > $$SA_OSCILL_REVERSE_POS1[Z] DO $AC_MARKER[1]=0
...
; ALWAYS WHEN current position of the oscillating axis in the MCS < start of reversal area 2,
; THEN override of the infeed axis = 0%
N620 ID=2 WHENEVER $AA_IM[Z] < $$SA_OSCILL_REVERSE_POS2[Z]-6 DO
 $AA_OVR[X]=0 $AC_MARKER[0]=0
...
; ALWAYS WHEN current position of the oscillating axis in the MCS == reversal position 1,
; THEN override of the oscillation axis = 0%, override of the infeed axis = 100%
; This cancels the previous synchronized action!
N630 ID=3 WHENEVER $AA_IM[Z]==$$SA_OSCILL_REVERSE_POS1[Z] DO
 $AA_OVR[Z]=0 $AA_OVR[X]=100
...
; ALWAYS WHEN distance-to-go of the partial infeed == 0,
; THEN override of the oscillation axis = 100%
; This cancels 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
...
; WHEN current position of the oscillating axis in the WCS == reversal position 1,
; THEN override of the oscillation axis = 100%, override of the infeed axis = 0%
; This cancels 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
```

3.3 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 (main run variables in each case)
2. Definition of this relationship as polynomial with limitations
3. With position offset: Setting the MD and SD
   - MD36750 $MA_AA_OFF_MODE (Effect of value assignment for axial override in case of synchronized actions)
   - SD43350 $SA_AA_OFF_LIMIT (optional) (Upper limit of the offset value $AA_OFF in case of clearance control)
4. Activation of the control in a synchronized action

3.3.1 Clearance control with variable upper limit

Example of polynomial with dynamic upper limit

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

Polynomial 1 is defined directly via system variables:

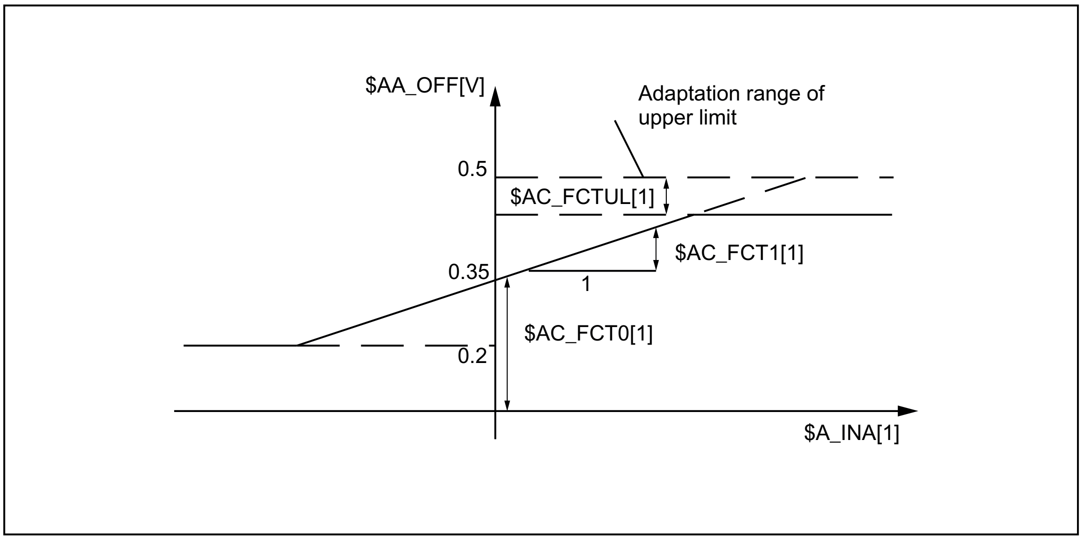


Figure 3-1 Clearance control with variable upper limit

```
$AC_FCTLL[1]=0.2                        ;  Lower limit
$AC_FCTUL[1]=0.5                        ;  Request Value of upper limit
$AC_FCT0[1]=0.35                        ;  Zero passage a0
$AC_FCT1[1]=1.5 EX-5                    ;  Pitch a1
STOPRE                                  ;  see following note
...
STOPRE                                  ;  see following note
ID=1 DO $AC_FCTUL[1]=$A_INA[2]*0.1+0.35 ;  Adjust upper limit dynamically via
                                        ;  analog input 2,
                                        ;  no condition
ID=2 DO SYNFCT(1, $AA_OFF[V], $A_INA[1]) ;  Clearance control by override of no
                                        ;  condition
...
```

Note

When system variables are used in the part program, `STOPRE` must be programmed to ensure block-synchronous writing. The following is an equivalent notation for polynomial definition:

`FCTDEF(1,0.2, 0.5, 0.35, 1.5EX-5).`

3.3.2 Feedrate control

Example of adaptive control with an analog input voltage

A process quantity (measured via $A_INA[1]) must be regulated to 2 V through an additive control factor implemented by a path (or axial) feedrate override. Feedrate override shall be performed within the range of +100 [mm/min].

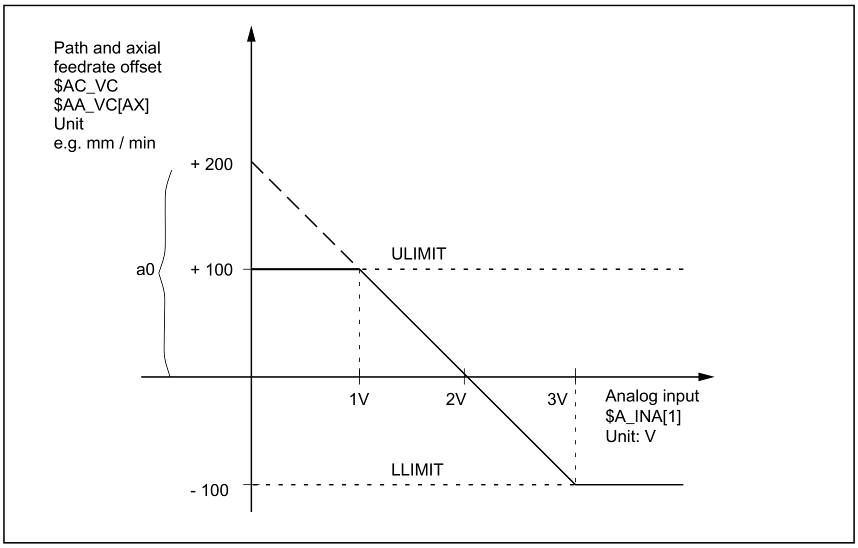


Figure 3-2 Diagram illustrating adaptive control

Determination of coefficients:

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

a1 = - 100mm / (1min * 1V)

a1 = - 100% regulation constants, pitch

a0 = - (-100) * 2 = 200

a2 = 0 (not a square component)

a3 = 0 (not a square component)

Upper limit = 100

Lower limit = -100

```
FCTDEF(          Polynomial No.
                 LLIMIT
                 ULIMIT
                 a0                  ;  y for x = 0
                 a1                  ;  Lead
                 a2                  ;  square component
                 a3 )                ;  cubic component
```

With the values determined above, the polynomial is defined as follows:

`FCTDEF`(1, -100, -100, 100, 200, 0, 0)

The following synchronized actions can be used to activate the adaptive control function for the axis feedrate:

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

or for the path feedrate:

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

3.3.3 Control velocity as a function of normalized path

Multiplicative adaptation

The normalized path is applied as an input quantity: $AC_PATHN.

0: At block start

1: at block end

Variation quantity $AC_OVR must be controlled as a function of $AC_PATHN according to a 3rd order polynomial. The override must be reduced from 100 to 1% during the motion.

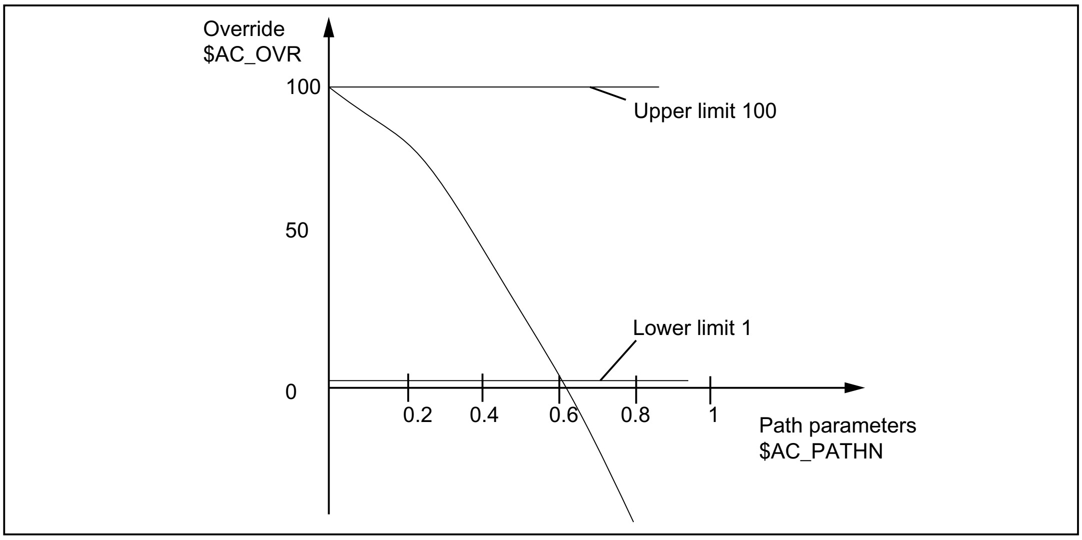


Figure 3-3 Regulate velocity continuously

Polynomial 2:

Lower limit: 1

Hi limit: 100

a_0: 100

a_1: -100

a_2: -100

a_3: not used

With these values, the polynomial definition is as follows:

`FCTDEF`(2, 1, 100, 100, -100, -100)

; Activation of the variable override as a function of the path:

```
ID= 1 DO SYNFCT (2, $AC_OVR, $AC_PATHN)

G01 X100 Y100 F1000
```

3.4 Monitoring a safety clearance between two axes

Task

The axes X1 and X2 operate two independently controlled transport devices used to load and unload workpieces.

To prevent the axes from colliding, a safety clearance must be maintained between them.

If the safety clearance is violated, then axis X2 is decelerated. This interlock is applied until axis X1 leaves the safety clearance area again.

If axis X1 continues to move towards axis X2, thereby crossing a closer safety barrier, then it is traversed into a safe position.

NC language	**Comment**
`ID=1 WHENEVER $AA_IM[X2] - $AA_IM[X1] < 30 DO $AA_OVR[X2]=0`	; Safety barrier
`ID=2 EVERY $AA_IM[X2] - $AA_IM[X1] < 15 DO POS[X1]=0`	; Safe position

3.5 Store execution times in R parameters

Task

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

Program	**Comment**
	; The example is ; as follows **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

3.6 "Centering" with continuous measurement

Introduction

The gaps between gear teeth are measured sequentially. The gap dimension is calculated from the sum of all gaps and the number of teeth. The center position sought for continuation of machining is the position of the first measuring point plus 1/2 the average gap size. The speed for measurement is selected in order to enable one measured value to be reliably acquired in each interpolation cycle.

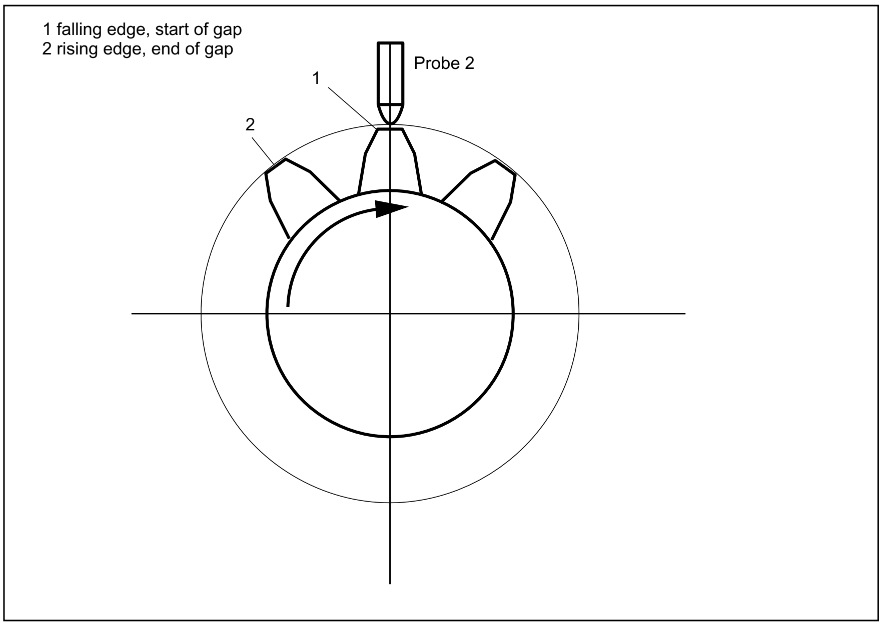


Figure 3-4 Diagrammatic representation of measurement of gaps between gear teeth

%_N_MEAC_MITTEN_MPF

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

```
;*** Define local user-defined variables ***
N1 DEF INT ZAEHNEZAHL              ;  Input number of gear teeth
N5 DEF REAL HYS_POS_FLANKE         ;  Hysteresis positive edge probe
N6 DEF REAL HYS_NEG_FLANKE         ;  Hysteresis negative edge probe
;*** Define short names for synchronized action markers ***
define M_ZAEHNE as $AC_MARKER[1]   ;  ID marker for calculation: neg/pos edge per
                                      tooth
define Z_MW as $AC_MARKER[2]       ;  Read ID counter MW FIFO
define Z_RW as $AC_MARKER[3]       ;  Calculate ID Counter MW tooth gaps
;*** Input values for ZAHNRADMESSEN ***
N50 ZAEHNEZAHL=26                  ;  Enter number of gear teeth to be measured
N70 HYS_POS_FLANKE = 0.160         ;  Hysteresis positive edge probe
N80 HYS_NEG_FLANKE = 0.140         ;  Hysteresis negative edge probe
```

```
Start:                              ;  *** Assign variables ***
R1=0                                ;  ID2 calculation result for gap dimension
R2=0                                ;  ID2 calculation result addition of all gaps
R3=0                                ;  Contents of the first element read
R4=0                                ;  R4 corresponds to a tooth distance
R5=0                                ;  Gap position calculated, final result
R6=1                                ;  Switch-on ID 3 BACH with MOV
R7=1                                ;  Switch-on ID 5 MEAC
M_ZAEHNE=ZAEHNEZAHL*2               ;  Calculate ID neg./pos. edge of each teeth
Z_MW=0                              ;  Read ID counter MW FIFO till the number of
                                       teeth
Z_RW=2                              ;  Calculate ID counter difference of tooth gap
R13=HYS_POS_FLANKE                  ;  Hysteresis in calculation register
R14=HYS_NEG_FLANKE                  ;  Hysteresis in calculation register
;*** Travel, measure, calculate axis ***
N100 MEAC[BACH]=(0)                 ;  Reset measurement job
;Resetting the FIFO[4] variables and ensuring a defined measurement trace
N105 $AC_FIFO1[4]=0                 ;  Reset FIFO1
STOPRE
;  *** Read FIFO till tooth number reached ***
;  if FIFO1 is not empty and all teeth are still not measured, save measured value
   from FIFO variable in
;  synchronization parameter and increment counter of measured values
```

```
ID=1 WHENEVER ($AC_FIFO1[4]>=1) AND (Z_MW<M_ZAEHNE)
        DO $AC_PARAM[0+Z_MW]=$AC_FIFO1[0] Z_MW=Z_MW+1
;if 2 measured values are present, start calculation, calculate ONLY gap dimension
; and gap sum, increment calculation value counter by 2
ID=2 WHENEVER (Z_MW>=Z_RW) AND (Z_RW<M_ZAEHNE)
        DO $R1=($AC_PARAM[-1+Z_RW]-$R13)-($AC_PARAM[-2+Z_RW]-$R14) Z_RW=Z_RW+2
        $R2=$R2+$R1
;*** Switch-on the axis BACH as endless rotating rotary axis with MOV ***
WAITP(BACH)
ID=3 EVERY $R6==1 DO MOV[BACH]=1     ;  Activate
FA[BACH]=1000
ID=4 EVERY $R6==0 und                ;  Deactivate
($AA_STAT[BACH]==1) DO MOV[BACH]=0
; Measure sequentially, store in FIFO 1, MT2 neg, MT2 pos edge
;the distance between two teeth is measured
;falling edge-...-rising edge, probe 2
N310 ID=5 WHEN $R7==1 DO MEAC[BACH]=(2, 1, -2, 2)
N320 ID=6 WHEN (Z_MW>=M_ZAEHNE) DO   ;  Cancel measuring job
MEAC[BACH]=(0)
M00
STOPRE
```

```
;*** FIFO Fetch and save values ***
N400 R3=$AC_PARAM[0]                 ;  Contents of the first element read
                                     ;  ;Reset the FIFO1[4] variable
                                     ;  ;and ensure a defined measuring trace
                                        ;for the next measurement job
N500 $AC_FIFO1[4]=0
```

```
;*** Calculate difference between the individual teeth ***
N510 R4=R2/(ZAEHNEZAHL)/1000         ;  R4 corresponds to an average
                                     ;  tooth distance
                                     ;  Division "/1000" removed in later SW
                                        versions
```

```
;*** Calculate center position ***
N520 R3=R3/1000                      ;  First measurement position converted to
                                        degree
N530 R3=R3 MOD 360                   ;  first measurement point modulo
N540 R5=(R3-R14)+(R4/2)              ;  calculate gap position
M00
stopre
R6=0                                 ;  Disable axis rotation from BACH
gotob start
M30
```

3.7 Axis couplings via synchronized actions

3.7.1 Coupling to leading axis

Task assignment

A cyclic curve table is defined by means of polynomial segments. Controlled by means of arithmetic variables, the movement of the master axis and the coupling process between master and slave (following) axes is activated/deactivated.

%_N_KOP_SINUS_MPF

```
N5 R1=1                              ;  ID 1, 2 activate/deactivate coupling: LEADON
                                        (CACB, BACH)
N6 R2=1                              ;  ID 3, 4 Move leading axis on/off: MOV BACH
N7 R5=36000                          ;  BACH Feedrate/min
N8 STOPRE
```

```
;*** Define periodic table No. 4 through polynomial segments ***
N10 CTABDEF (YGEO,XGEO,4,1)
N16 G1 F1200 XGEO=0.000 YGEO=0.000    ;  Go to basic position
N17 POLY PO[XGEO]=(79,944.30.420,00.210) PO[YGEO]=(24,634.00.871,-9,670)
N18 PO[XGEO]=(116.059,0.749,-0.656) PO[YGEO]=(22.429,-5.201,0.345)
N19 PO[XGEO]=(243.941,-17.234,11.489) PO[YGEO]=(-22.429,-58.844,39.229)
N20 PO[XGEO]=(280.056,1.220,-0.656) PO[YGEO]=(-24.634,4.165,0.345)
N21 PO[XGEO]=(360.000,-4.050,0.210) PO[YGEO]=(0.000,28.139,-9.670)
N22 CTABEND                           ;  *** End of table definition ***
```

```
; Travel axis leading axis and coupled axis in quick motion in basic position
N80 G0 BACH=0 CACH=0                  ;  Channel axis names
N50 LEADOF(CACH,BACH)                 ;  existing coupling OFF
```

```
N235 ;*** Switch-on the coupling movement for the axis CACH ***
N240 WAITP(CACH)                      ;  Synchronize axis to channel
N245 ID=1 EVERY $R1==1 DO             ;  Coupling via table 4
LEADON(CACH, BACH, 4)
N250 ID=2 EVERY $R1==0 DO             ;  Deactivate coupling
LEADOF(CACH, BACH)
```

```
N265 WAITP(BACH)
N270 ID=3 EVERY $R2==1 DO             ;  Rotate leading axis with feedrate endlessly
MOV[BACH]=1 FA[BACH]=R5                  in R5
N275 ID=4 EVERY $R2==0 DO             ;  Stop leading axis
MOV[BACH]=0
N280 M00
N285 STOPRE
N290 R1=0                             ;  Disable coupling condition
N295 R2=0                             ;  Disable condition for rotating leading axis
N300 R5=180                           ;  New feedrate for BACH
N305 M30
```

3.7.2 Non-circular grinding via master value coupling

Task assignment

A non-circular workpiece that is rotating on axis CACH must be machined by grinding. The distance between the grinding wheel and workpiece is controlled by axis XACH and depends on the angle of rotation of the workpiece. The interrelationship between angles of rotation and assigned movements is defined in curve table 2. The workpiece must move at velocities that are determined by the workpiece contour defined in curve table 1.

Solution

CACH is designated as the leading axis in a master value coupling. It controls:

- via table 2 the compensatory movement of the axis XACH
- via table 1 the "software axis" CASW.

The axis override of axis CACH is determined by the actual values of axis CASW, thus providing the required contour-dependent velocity of axis CACH.

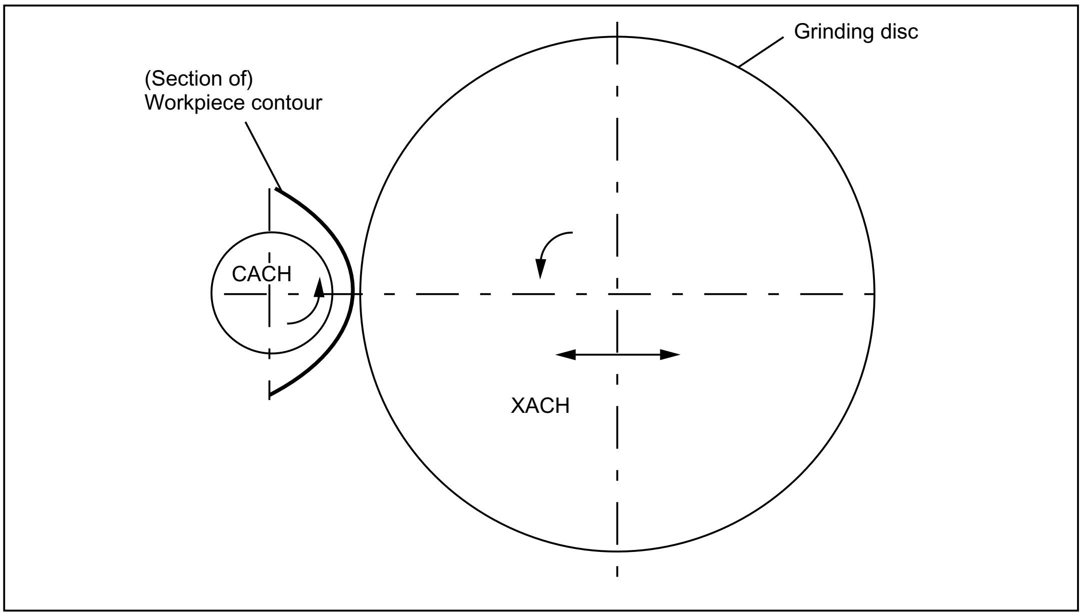


Figure 3-5 Diagrammatic representation of non-circular contour grinding

%_N_CURV_TABS_SPF		
PROC CURV_TABS		
N160 ; *** Define table 1 override ***		
N165 CTABDEF(CASW,CACH,1.1)	;	Table 1 periodic
N170 CACH=0 CASW=10		
N175 CACH=90 CASW=10		
N180 CACH=180 CASW=100		

N185 CACH=350 CASW=10		
N190 CACH=359.999 CASW=10		
N195 CTABEND		

N160 ; *** Define table 2 linear compensatory movement of XACH ***		
CTABDEF(YGEO,XGEO,2.1)	;	Table 2 periodic
N16 XGEO=0.000 YGEO=0.000		
N16 XGEO=0.001 YGEO=0.000		
N17 POLY PO[XGEO]=(116.000,0.024,0.012) PO[YGEO]=(4.251,0.067,-0.828)		
N18 PO[XGEO]=(244.000,0.072,-0.048) PO[YGEO]=(4.251,-2.937)		
N19 PO[XGEO]=(359.999,-0.060,0.012) PO[YGEO]=(0.000,-2.415,0.828)		
N16 XGEO=360.000 YGEO=0.000		
N20 CTABEND		
M17		

%_N_UNRUND_MPF

; Coupling group for a non-circular machining

; XACH is the infeed axis of the grinding disk

; CACH is the workpiece axis as rotary axis and master value axis

; Application: Grind non-circular contours

; Table 1 maps the override for axis CACH as function of the position of CACH

; Overlay of the XGEO axis with handwheel infeed for scratching

N100 DRFOF	;	deselect handwheel overlay
N200 MSG(Select "DRF, (Handwheel 1 active) and Select INKREMENT.== Handwheel overlay AKTIV")		
N300 M00		
N500 MSG()	;	Reset message
N600 R2=1	;	LEADON Table 2, Activate with ID=3/4 CACH to XACH
N700 R3=1	;	LEADON Table 1, Activate with ID=5/6 CACH to CASW, override
N800 R4=1	;	Endless rotating axis CACH, start with ID=7/8
N900 R5=36000	;	FA[CACH] Endless rotating rotary axis speed

N1100 STOPRE		
N1200	;	*** Set axis and leading axis to FA ***
	; ;	Move Axis, master axis and following axis to the initial position
N1300 G0 XGEO=0 CASW=10 CACH=0		
N1400 LEADOF(XACH,CACH)	;	Coupling AUS XACH compensatory movement
N1500 LEADOF(CASW,CACH)	;	Coupling AUS CASW override table
N1600 CURV_TABS	;	Sub-program with definition of the tables

N1700	;	*** On-off switch of the LEADON compensatory movement XACH ***
N1800 WAITP(XGEO)	;	Synchronize axis to channel
N1900 ID=3 EVERY $R2==1 **DO** LEADON(XACH,CACH,2)		
N2000 ID=4 EVERY $R2==0 **DO** LEADOF(XACH,CACH)		

N2100	; ;	*** On-off switch of the LEADON CASW override table ***
N2200 WAITP(CASW)		
N2300 ID=5 EVERY $R3==1 **DO** LEADON(CASW,CACH,1)	;	CTAB Coupling ON leading axis CACH
N2400 ID=6 EVERY $R3==0 **DO** LEADOF(CASW,CACH)	;	CTAB Coupling OFF leading axis CACH

N2500	; ;	*** Control override of the CACH 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 endless rotating rotary axis
N3100 ID=8 EVERY $R4==0 **DO** MOV[CACH]=0	;	Stop as endless rotating rotary axis

N3200 STOPRE		
N3300 R90=$AA_COUP_ACT[CASW]	;	State of the coupling for CASW for checking
N3400 MSG("Override table CASW activated with LEADON "<<R90<<", further ENDE with NC-START")		

N3500 M00	;	*** NC HALT ***
N3600 MSG()		
N3700 STOPRE	;	Preprocessing stop
N3800 R1=0	; ;	Stop with ID=2 CASW axis as endless rotating rotary axis
N3900 R2=0	; ;	LEADOF with ID=6 FA XACH and leading axis CACH
N4000 R3=0	; ;	LEADOF TAB1 CASW with ID=7/8 CACH to CASW override table
N4100 R4=0	; ;	Stop axis as endless rotating rotary axis , ID=4 CACH
N4200 M30		

Expansion options

The example above can be expanded by the following components:

- Introduction of a Z axis to move the grinding wheel or workpiece from one non-circular operation to the next on the same shaft (cam shaft).
- Table switchovers, if the cams for inlet and outlet have diferent contours.

```
ID = ... <Condition> DO LEADOF(XACH, CACH) LEADON(XACH, CACH, <new table number>)
```

- Dressing of grinding wheel by means of online tool offset acc. to Subsection "Online tool offset FTOC".

3.7.3 On-the-fly parting

Task assignment

An extruded material which passes continuously through the operating area of a cutting tool must be cut into parts of equal length.

X axis: Axis in which the extruded material moves, WKS

X1 axis: Machine axis of the extruded material, MKS

Y axis: Axis in which the cutting tool "tracks" the extruded material

It is assumed that the infeed and control of the cutting tool are controlled via the PLC. The signals at the PLC interface can be evaluated to determine whether the extruded material and cutting tool are synchronized.

Actions

Activate coupling, `LEADON`

Deactivate coupling, `LEADOF`

Set actual values, `PRESETON`

Program code	Comment
%_N_SCHERE1_MPF	
;$PATH=/_N_WKS_DIR/_N_DEMOFBE_WPD	
N100 R3=1500	; Length of a part to be cut off
N200 R2=100000 R13=R2/300	
N300 R4=100000	
N400 R6=30	; Start position Y axis
N500 R1=1	; Start condition for conveyor axis
N600 LEADOF(Y,X)	; Delete any existing coupling
N700 CTABDEF(Y,X,1,0)	; Table definition
N800 X=30 Y=30	; Value pairs
N900 X=R13 Y=R13	
N1000 X=2*R13 Y=30	
N1100 CTABEND	; End of table definition
N1200 PRESETON(X1,0)	; Preset offset at the beginning
N1300 Y=R6 G0	; Start position Y axis
	; The axis is a linear axis
N1400 ID=1 EVERY $AA_IW[X]>$R3 DO PRESETON(X1,0)	; Preset offset after length R3, PRESETON must only be executed with WHEN and EVERY
	; New start after parting
N1500 WAITP(Y)	
N1800 ID=6 EVERY $AA_IM[X]<10 DO LEADON(Y,X,1)	; For X < 10, couple Y to X via table 1
N1900 ID=10 EVERY $AA_IM[X]>$R3-30 DO LEADOF(Y,X)	; > 30 before traversed parting distance, deactivate coupling
N2000 WAITP(X)	
N2100 ID=7 WHEN $R1==1 DO MOV[X]=1 FA[X]=$R4	; Set extruded material axis continuously in motion
N2200 M30	

3.8 Technology cycles position spindle

Application

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

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

See chapter "Starting of command axes" and chapter "Control via PLC".

Coordination

The PLC and NCK are coordinated by means of the common data that are provided in SW version 4 and later (see chapter "List of the system variables relevant for synchronized actions")

- $A_DBB[0]: Take up basic position 1,
- $A_DBB[1]: Take up target position 1,
- $A_DBW[1]: value to be positioned +/- , PLC calculates the shortest route.

Synchronized actions

%_N_MAIN_MPF

```
...
IDS=1 EVERY $A_DBB[0]==1 DO NULL_POS    ; when $A_DBB[0] set by PLC,
                                        ; take up basic position
IDS=2 EVERY $A_DBB[1]==1 DO ZIEL_POS    ; when $A_DBB[1] set by PLC,
                                        ; position spindle to the value stored in
                                        ; $A_DBW[1]
...
```

Technology cycle NULL_POS

%_N_NULL_POS_SPF

```
PROC NULL_POS
SPOS=0                ; Bring drive for the tool change in basic position
$A_DLB[0]=0           ; Basic position executed in NCK
```

Technology cycle ZIEL_POS

%_N_ZIEL_POS_SPF

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

3.9 Synchronized actions in the TC/MC area

Introduction

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

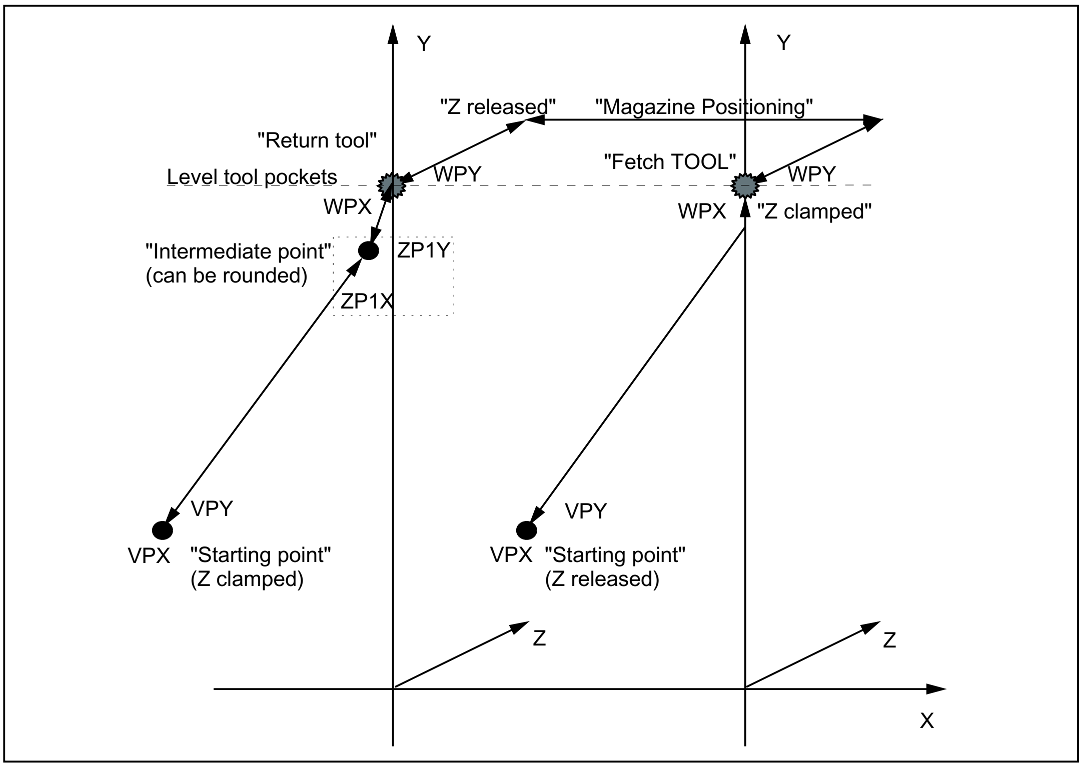


Figure 3-6 Schematic sequence for tool-changing cycle

Flow chart

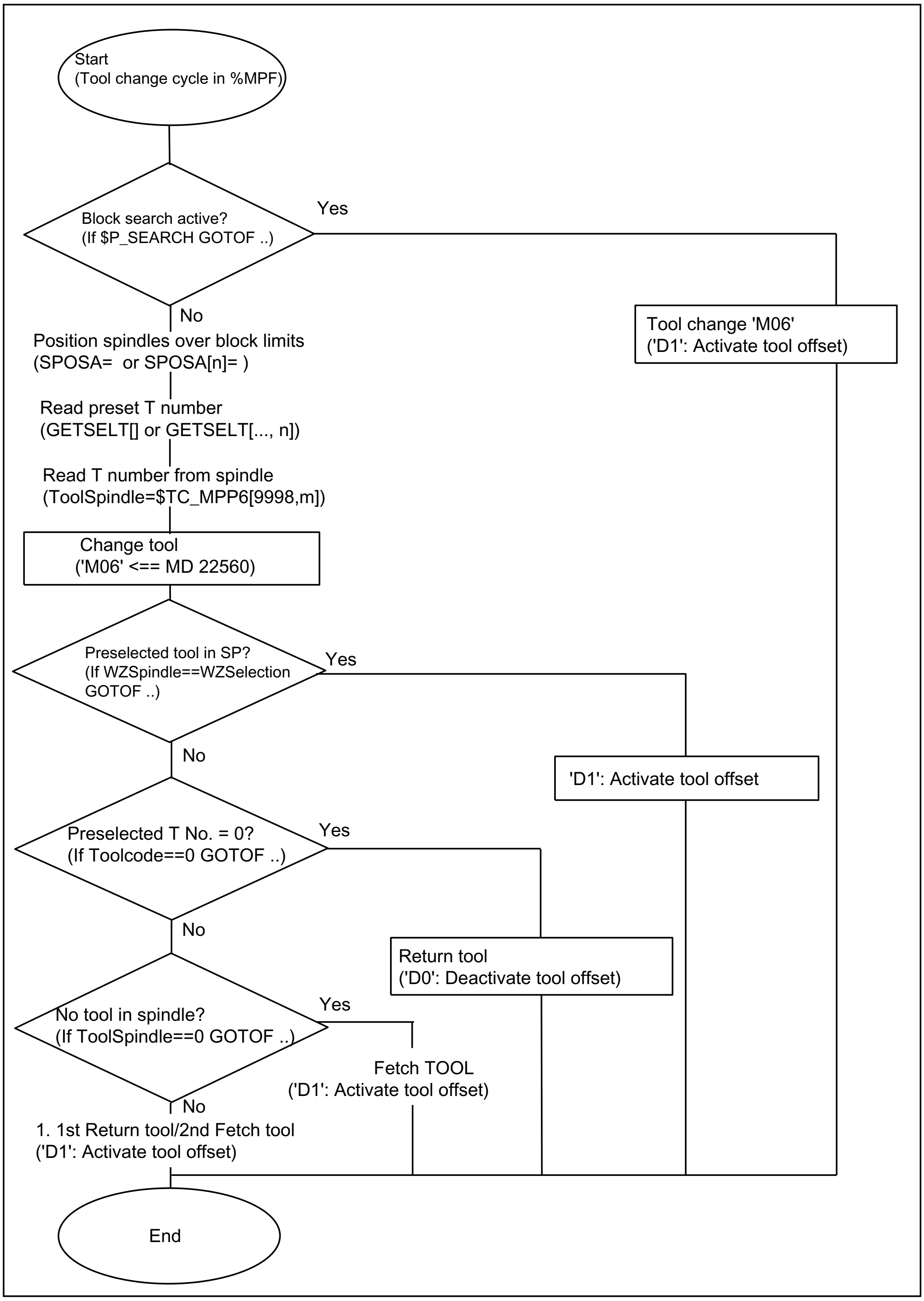


Figure 3-7 Flowchart for tool-changing cycle

NC program	Comment
%_N_WZW_SPF	
;$PATH=/_N_SPF_DIR	
N10 DEF INT WZPreselection,WZSpindle	; Marker on = 1 when MagAxis traversed
N15 WHEN $AC_PATHN<10 DO $AC_MARKER[0]=0 $AC_MARKER[1]=0 $AC_MARKER[2]=0	
N20 ID=3 WHENEVER $A_IN[9]==TRUE DO $AC_MARKER[1]=1	
N25 ID=4 WHENEVER $A_IN[10]==TRUE DO $AC_MARKER[2]=1	; Marker on = 1 when MagAxis traversed
N30 IF $P_SEARCH GOTOF wzw_vorlauf	; Block search active ? ->
N35 SPOSA=0 D0	
N40 GETSELT(WZPreselection)	; Read preselected T no.
N45 WZSpindle=$TC_MPP6[9998,1]	; Read WZ in spindle
N50 M06	
N55 IF WZSpindle==WZPreselection GOTOF wz_in_spindle IF WZPreselection==0 GOTOF store1 IF WZSpindle==0 GOTOF fetch1	
;* Fetch and store tool*****	
store1fetch1:	
N65 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[1]=1	; when MagAxis travels Marker = 1
N70 G01 G40 G53 G64 G90 X=Magazin1VPX Y=Magazin1VPY Z=Magazin1ZGespannt F70000 M=QU(120) M=QU(123) M=QU(9)	
N75 WHENEVER $AA_STAT[S1]<>4 DO $AC_OVR=0	; Spindle in position
N80 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[1]=1	; Query MagAxis travel
N85 WHENEVER $AC_MARKER[1]==0 DO $AC_OVR=0	; Override=0 when axis not traversed
N90 WHENEVER $AA_STAT[C2]<>4 DO $AC_OVR=0	; Override=0 when MagAxis ; not in position fine
N95 WHENEVER $AA_DTEB[C2]>0 DO $AC_OVR=0	; Override=0 when distance-to-go MagAxis > 0
N100 G53 G64 X=Magazin1ZP1X Y=Magazin1ZP1Y F60000	
N105 G53 G64 X=Magazin1WPX Y=Magazin1WPY F60000	
N110 M20	; Release WZ
N115 G53 G64 Z=MR_Magazin1ZGeloest F40000	
N120 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[2]=1;	
N125 WHENEVER $AC_MARKER[2]==0 DO $AC_OVR=0	
N130 WHENEVER $AA_STAT[C2]<>4 DO $AC_OVR=0	
N135 WHENEVER $AA_DTEB[C2]>0 DO $AC_OVR=0	
N140 G53 G64 Z=Magazin1ZGespannt F40000	
N145 M18	; Clamp tool
N150 WHEN $AC_PATHN<10 DO M=QU(150) M=QU(121)	; Condition always fulfilled
N155 G53 G64 X=Magazin1VPX Y=Magazin1VPY F60000 D1 M17	
;* Store tool*****	
store1:	
N160 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[1]=1	
N165 G01 G40 G53 G64 G90 X=Magazin1VPX Y=Magazin1VPY Z=Magazin1ZGespannt F70000 M=QU(120) M=QU(123) M=QU(9)	
N170 WHENEVER $AA_STAT[S1]<>4 DO $AC_OVR=0	
N175 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[1]=1	

NC program	Comment
N180 WHENEVER $AC_MARKER[1]==0 DO $AC_OVR=0	
N185 WHENEVER $AA_STAT[C2]<>4 DO $AC_OVR=0	
N190 WHENEVER $AA_DTEB[C2]>0 DO $AC_OVR=0	
N195 G53 G64 X=Magazin1ZP1X Y=Magazin1ZP1Y F60000	
N200 G53 G64 X=Magazin1WPX Y=Magazin1WPY F60000	
N205 M20	; Release tool
N210 G53 G64 Z=Magazin1ZGeloest F40000	
N215 G53 G64 X=Magazin1VPX Y=Magazin1VPY F60000 M=QU(150) M=QU(121) D0 M17	
;* Fetch tool*****	
fetch1:	
N220 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[2]=1	
N225 G01 G40 G53 G64 G90 X=Magazin1VPX Y=Magazin1VPY Z=Magazin1ZGeloest F70000 M=QU(120) M=QU(123) M=QU(9)	
N230 G53 G64 X=Magazin1WPX Y=Magazin1WPY F60000	
N235 WHENEVER $AA_STAT[S1]<>4 DO $AC_OVR=0	
N240 WHENEVER $AA_VACTM[C2]<>0 DO $AC_MARKER[2]=1	
N245 WHENEVER $AC_MARKER[2]==0 DO $AC_OVR=0	
N250 WHENEVER $AA_STAT[C2]<>4 DO $AC_OVR=0	
N255 WHENEVER $AA_DTEB[C2]>0 DO $AC_OVR=0	
N260 G53 G64 Z=Magazin1ZGespannt F40000	
N265 M18	; Clamp tool
N270 G53 G64 X=Magazin1VPX Y=Magazin1VPY F60000 M=QU(150) M=QU(121) D1 M17	
;*Tool in spindle*****	
wz_in_spindle:	
N275 M=QU(121) D1 M17	
;*Block search*****	
wzw_feed:	
N280 STOPRE	
N285 D0	
N290 M06	
N295 D1 M17	

Data lists 4

4.1 Machine data

4.1.1 General machine data

Number	Identifier: $MN_	Description
11110	AUXFU_GROUP_SPEC	Auxiliary function group specification
11500	PREVENT_SYNACT_LOCK	Protected synchronized actions
18860	MM_MAINTENANCE_MON	Activate recording of maintenance data

4.1.2 Channelspecific machine data

Number	Identifier: $MC_	Description
21240	PREVENT_SYNACT_LOCK_CHAN	Protected synchronized actions for channel
28250	MM_NUM_SYNC_ELEMENTS	Number of elements for expressions in synchronized actions
28252	MM_NUM_FCTDEF_ELEMENTS	Number of FCTDEF elements
28254	MM_NUM_AC_PARAM	Number of $AC_PARAM parameters
28255	MM_BUFFERED_AC_PARAM	Memory location of $AC_PARAM
28256	MM_NUM_AC_MARKER	Number of $AC_MARKER markers
28257	MM_BUFFERED_AC_MARKER	Memory location of $AC_MARKER
28258	MM_NUM_AC_TIMER	Number of $AC_TIMER time variables
28260	NUM_AC_FIFO	Number of $AC_FIFO1, $AC_FIFO2, ... variables
28262	START_AC_FIFO	Store FIFO variables from R parameter
28264	LEN_AC_FIFO	Length of $AC_FIFO ... FIFO variables
28266	MODE_AC_FIFO	FIFO processing mode

4.1.3 Axis-specific machine data

Number	Identifier: $MA_	Description
30450	IS_CONCURRENT_POS_AX	Concurrent positioning axis
32060	POS_AX_VELO	Initial setting for positioning axis velocity
32070	CORR_VELO	Axial velocity for handwheel, ext. WO (work offset), cont. dressing, clearance control
32074	FRAME_OR_CORRPOS_NOTALLOWED	Effectiveness of frames and tool length offset
32920	AC_FILTER_TIME	Filter smoothing time constant for Adaptive Control
33060	MAINTENANCE_DATA	Configuration, recording maintenance data
36750	AA_OFF_MODE	Effect of value assignment for axial override with synchronized actions
37200	COUPLE_POS_TOL_COARSE	Threshold value for "Coarse synchronism"
37210	COUPLE_POS_TOL_FINE	Threshold value for "Fine synchronism"

4.2 Setting data

4.2.1 Axis/spindle-specific setting data

Number	Identifier: $SA_	Description
43300	ASSIGN_FEED_PER_REV_SOURCE	Rotational feedrate for positioning axes/spindles
43350	AA_OFF_LIMIT	Upper limit of offset value for $AA_OFF clearance control
43400	WORKAREA_PLUS_ENABLE	Working area limitation in pos. direction

4.3 Signals

4.3.1 Signals to channel

DB number	Byte.Bit	Description
21, ...	21.2	Disable all synchronized actions
21, …	280.1	Disable synchronized actions ID/IDS 1 - 64 (general request)
21, …	300.0 - 307.7	Disable synchronized action ID/IDS 1 - 64

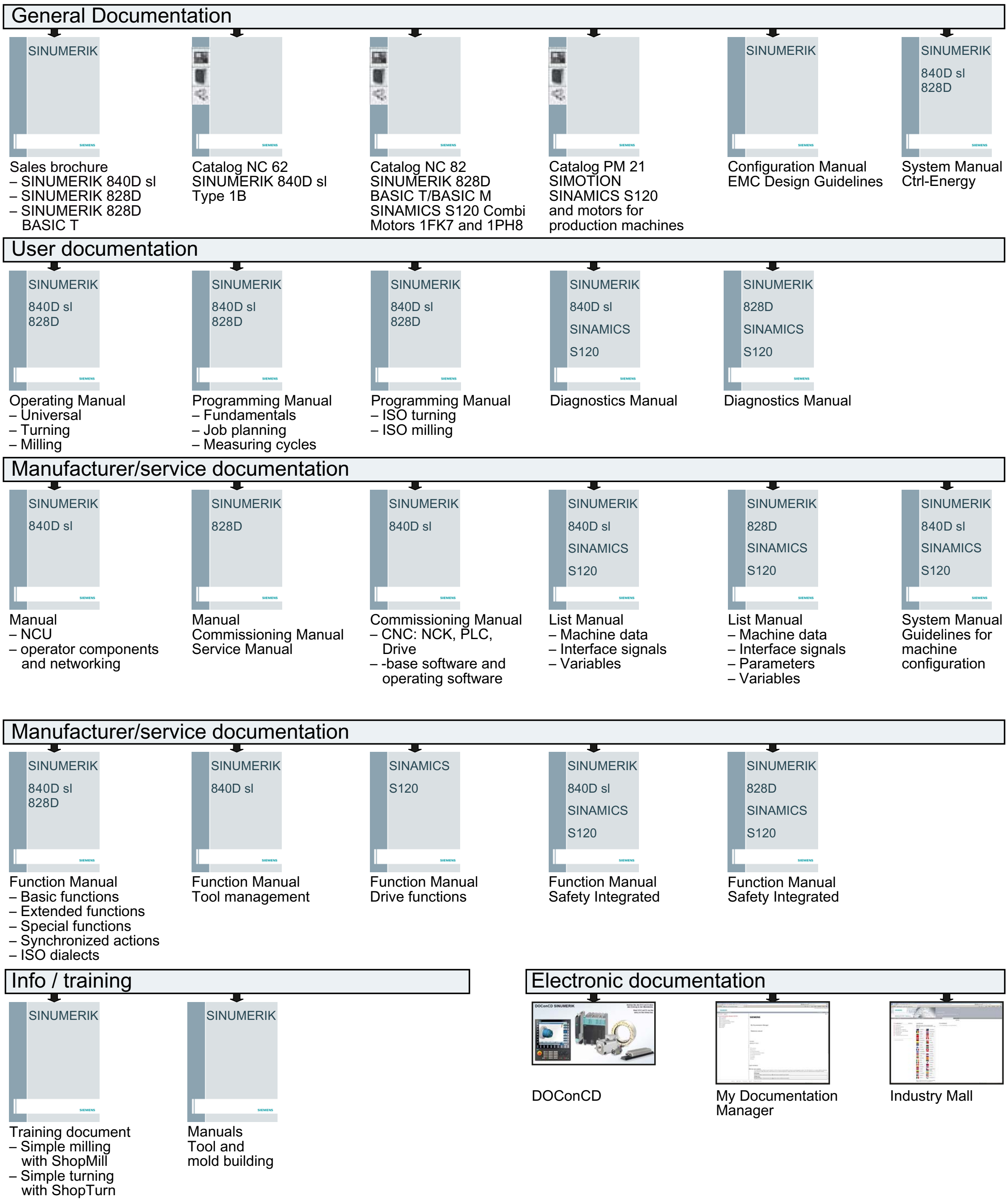
4.3.2 Signals from channel

DB number	Byte.Bit	Description
21, …	281.1	Synchronized actions ID/IDS 1 - 64 disabled (general feedback signal)
21, …	308.0 - 315.7	Synchronized action ID/IDS 1 - 64 can be disabled

Appendix

A

A.1 Overview



Index

D

E

F

G

I

L

M

N

P

R

S

T

W