Tool for Control Technology

applications & TOOLS

Tool collection of functions for programming tasks involving mathematical operations

SIEMENS

Tool collection for bit, number and mathematical operations

Note

The application examples and Tools are not binding and do not claim to be complete regarding the circuits shown, equipping and any eventuality. The application examples and Tools do not represent customer-specific solutions. They are only intended to provide support for typical applications. You are responsible in ensuring that the described products are correctly used. These application examples and Tools do not relieve you of the responsibility in safely and professionally using, installing, operating and servicing equipment. When using these application examples, you recognize that we cannot be made liable for any damage/claims beyond the liability clause described. We reserve the right to make changes to these application examples at any time without prior notice. If there are any deviations between the recommendations provided in these application examples and other Siemens publications - e.g. Catalogs - then the contents of the other documents have priority.

Warranty, Liability and Support

We do not accept any liability for the information contained in this document.

Any claims against us - based on whatever legal reason - resulting from the use of the examples, information, programs, engineering and performance data etc., described in this application example shall be excluded. Such an exclusion shall not apply in the case of mandatory liability, e.g. under the German Product Liability Act (“Produkthaftungsgesetz”), in case of intent, gross negligence, or injury of life, body or health, guarantee for the quality of a product, fraudulent concealment of a deficiency or breach of a condition which goes to the root of the contract (“wesentliche Vertragspflichten”). However, claims arising from a breach of a condition which goes to the root of the contract shall be limited to the foreseeable damage which is intrinsic to the contract, unless caused by intent or gross negligence or based on mandatory liability for injury of life, body or health The above provisions does not imply a change in the burden of proof to your detriment.

For questions about this document, please use the following e-mail address:

online-support.automation@siemens.com

Preface

In this example, we introduce fully functional and tested automation configurations based on Siemens Industry Sector standard products and individual function blocks or tools, for simple, fast and inexpensive implementation of automation tasks.

Apart from a list of all required hardware and software components and a description of the way they are connected to each other, the examples include the tested tools or function blocks. This ensures that the functionalities described here can be reset in a short period of time and thus also be used as a basis for individual expansions.

Industry Automation and Drives Technologies Service & Support Portal

This entry is from the internet service portal of Siemens AG, Industry Automation and Drives Technologies. Clicking the link below directly displays the download page of this document.

http://support.automation.siemens.com/WW/view/en/29851674

Table of Contents

1 Bit and Number Operations

1.1 Random number generator

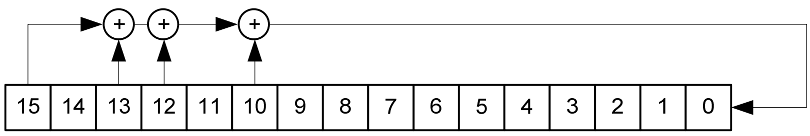
Description

In order to create random numbers, random number generators are used. These can be hardware- or software-implemented in IT systems. It should be noted that these random number generators are generally deterministic. Accordingly, the sequences of numbers created by such random number generators only appear to be random, but are actually determined by an algorithm. This means that the same initial value creates the same sequence of numbers.

In practical use, you can easily implement deterministic random number generators by means of a linear feedback shift register.

Figure 1-1 illustrates the basic structure of such a register.

Figure 1-1



In each cycle, the register is shifted to the left by one bit and the bits 15, 13, 12 and 10 are operated on by XOR. The result of the XOR operation is the input bit of the shift register. In the case of a 16-bit shift register, the range of possible values is 1 – 65,535. The algorithm excludes the zero value.

Function “RANDOM” (FC 45)

The RANDOM function is a random number generator implemented as 16-bit feedback shift register. The random numbers are in the range of -32,768 to +32,767. It should be noted that the zero number never occurs. For the 16-bit shift register, you have to assign a static variable to the INOUT parameter “RND”. At the same time, RND contains the random number, which is recalculated on each call of the RANDOM function.

The random number generator is initialized with a new seed by means of the “Init” input. The initial value is determined by means of system function SFC1 (READ_CLK). The initialization is level-triggered; edge detection does not take place. If the INOUT variable RND is zero on function call, an initialization takes place automatically.

Note

- The initialization via the “Init” input is not edge-triggered. ➔ As long as this input is selected, no random numbers are created.
- The algorithm excludes the zero value.
- If the random numbers Z are desired to be in the range between “0” and N, you have to perform a modulo operation (Z MOD (N + 1)).

Block parameters of function “RANDOM” (FC 45)

Table 1-1

Parameters	Declaration	Data type	Area	Description
Init	IN	BOOL	I, Q, M, D, L	Initialization of the random number generator
RND	INOUT	INT	Q, M, D, L	Contains the random number

Example

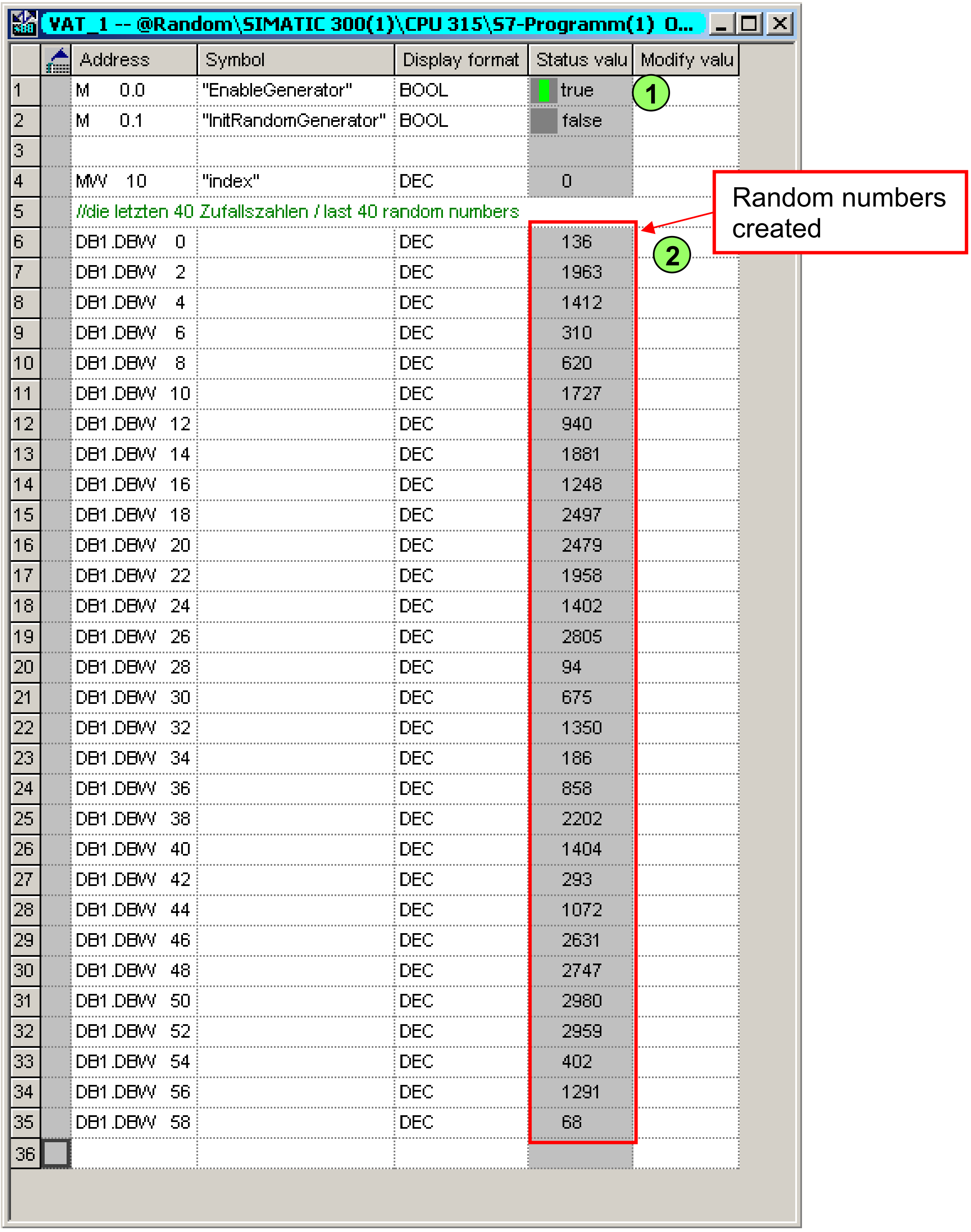
In the example project, the RANDOM function is called in the OB1 block if the M0.0 flag (EnableGenerator) is set. The random numbers range between 0 and 3000. The last 30 random numbers are saved to DB1. The variable table “VAT_1” contains the results.

In order to test the example project, proceed as follows:

Table 1-2

Step	Action / Event
1.	✓ Load the complete station into the CPU or to the S7-PLCSIM.
2.	✓ Open the variable table “VAT_1” in online mode. ①
3.	✓ Set the M0.0 flag (EnableGenerator) to signal status “1” **Result:** The simulation is activated and the random numbers are generated. ②

Figure 1-2



	Address	Symbol	Display format	Status valu	Modify valu
1	M 0.0	"EnableGenerator"	BOOL	true	
2	M 0.1	"InitRandomGenerator"	BOOL	false	
3					
4	MW 10	"index"	DEC	0	
5	//die letzten 40 Zufallszahlen / last 40 random numbers				
6	DB1.DBW 0		DEC	136	
7	DB1.DBW 2		DEC	1963	
8	DB1.DBW 4		DEC	1412	
9	DB1.DBW 6		DEC	310	
10	DB1.DBW 8		DEC	620	
11	DB1.DBW 10		DEC	1727	
12	DB1.DBW 12		DEC	940	
13	DB1.DBW 14		DEC	1881	
14	DB1.DBW 16		DEC	1248	
15	DB1.DBW 18		DEC	2497	
16	DB1.DBW 20		DEC	2479	
17	DB1.DBW 22		DEC	1958	
18	DB1.DBW 24		DEC	1402	
19	DB1.DBW 26		DEC	2805	
20	DB1.DBW 28		DEC	94	
21	DB1.DBW 30		DEC	675	
22	DB1.DBW 32		DEC	1350	
23	DB1.DBW 34		DEC	186	
24	DB1.DBW 36		DEC	858	
25	DB1.DBW 38		DEC	2202	
26	DB1.DBW 40		DEC	1404	
27	DB1.DBW 42		DEC	293	
28	DB1.DBW 44		DEC	1072	
29	DB1.DBW 46		DEC	2631	
30	DB1.DBW 48		DEC	2747	
31	DB1.DBW 50		DEC	2980	
32	DB1.DBW 52		DEC	2959	
33	DB1.DBW 54		DEC	402	
34	DB1.DBW 56		DEC	1291	
35	DB1.DBW 58		DEC	68	
36					

Technical data

Table 1-3

Block	Data
RANDOM (FC 45) Random number generator	Required local data: 24 bytes Load memory requirement: 252 bytes Main memory requirement : 180 bytes

The respective download file is available in chapter "Overview of the download files".

1.2 Determination of the parity of data elements

Description

The PARITY function determines the parity bit of an input memory area. The memory areas that may contain the data are:

- Inputs
- Flags
- Data blocks

Function “PARITY” (FC 12)

The PARITY function (FC 12) allows the parity determination of byte, word, double word and DB blocks for the memory areas “input”, “flag” and “data blocks”. The data is transferred via an ANY pointer. Transfer parameters, such as MB, IW, DB12.DBD, or data blocks (e.g. P#DB12.DBX 0.0 BYTE 13) are permitted for this. The function checks the data types and memory areas listed above, generating an error bit if they do not correspond. The parity bit is available as output parameter. It is coded as follows:

- Parity even = '0'
- Parity odd = '1'

Block parameters of function “PARITY” (FC 12)

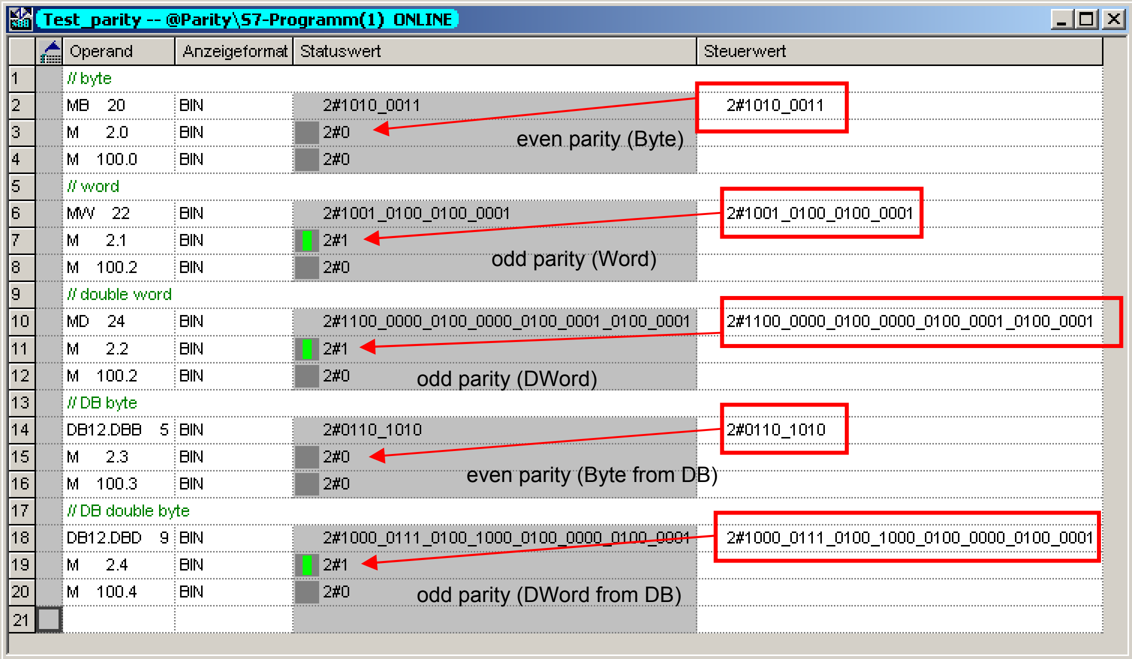
Table 1-4

Parameters	Declaration	Data type	Area	Description
ParityTest	IN	ANY	I, M, D	Input of the area to be checked
Parity	Out	BOOL	Q, M, D, L	Result: Even = 0, Odd = 1
Failure	Out	BOOL	Q, M, D, L	Error in case of wrong transfer parameters

Example

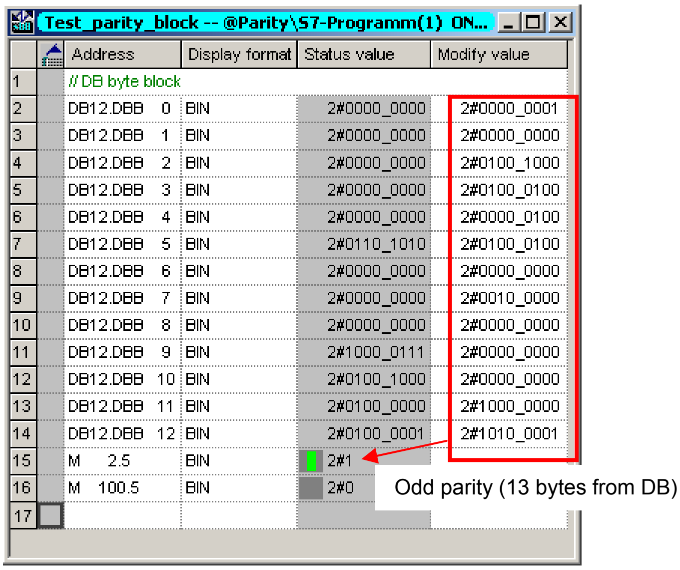
There is an example appended to the project. In this example, the PARITY function is called with various parameters in FC 11. You can check the results in the variable tables “Test_parity” and “Test_parity_block”.

Figure 1-3



	Operand	Anzeigeformat	Statuswert	Steuerwert
1	// byte			
2	MB 20	BIN	2#1010_0011	2#1010_0011
3	M 2.0	BIN	2#0	
4	M 100.0	BIN	2#0	
5	// word			
6	MW 22	BIN	2#1001_0100_0100_0001	2#1001_0100_0100_0001
7	M 2.1	BIN	2#1	
8	M 100.2	BIN	2#0	
9	// double word			
10	MD 24	BIN	2#1100_0000_0100_0000_0100_0001_0100_0001	2#1100_0000_0100_0000_0100_0001_0100_0001
11	M 2.2	BIN	2#1	
12	M 100.2	BIN	2#0	
13	// DB byte			
14	DB12.DBB 5	BIN	2#0110_1010	2#0110_1010
15	M 2.3	BIN	2#0	
16	M 100.3	BIN	2#0	
17	// DB double byte			
18	DB12.DBD 9	BIN	2#1000_0111_0100_1000_0100_0000_0100_0001	2#1000_0111_0100_1000_0100_0000_0100_0001
19	M 2.4	BIN	2#1	
20	M 100.4	BIN	2#0	
21				

Figure 1-4



	Address	Display format	Status value	Modify value
1	// DB byte block			
2	DB12.DBB 0	BIN	2#0000_0000	2#0000_0001
3	DB12.DBB 1	BIN	2#0000_0000	2#0000_0000
4	DB12.DBB 2	BIN	2#0000_0000	2#0100_1000
5	DB12.DBB 3	BIN	2#0000_0000	2#0100_0100
6	DB12.DBB 4	BIN	2#0000_0000	2#0000_0100
7	DB12.DBB 5	BIN	2#0110_1010	2#0100_0100
8	DB12.DBB 6	BIN	2#0000_0000	2#0000_0000
9	DB12.DBB 7	BIN	2#0000_0000	2#0010_0000
10	DB12.DBB 8	BIN	2#0000_0000	2#0000_0000
11	DB12.DBB 9	BIN	2#1000_0111	2#0000_0000
12	DB12.DBB 10	BIN	2#0100_1000	2#0000_0000
13	DB12.DBB 11	BIN	2#0100_0000	2#1000_0000
14	DB12.DBB 12	BIN	2#0100_0001	2#1010_0001
15	M 2.5	BIN	2#1	
16	M 100.5	BIN	2#0	
17				

Technical data

Table 1-5

Block	Data
PARITY (FC 12) Parity determination of a data area	Required local data: 12 bytes Load memory requirement: 370 bytes Main memory requirement : 284 bytes

The respective download file is available in chapter “Overview of the download files”.

1.3 Determination of the active bit position in a flag word

Description

In STEP 7, the process control for a sequential process is to be implemented by means of a flag word. In a step sequence flag word, there is always just one bit active, which corresponds to the currently active step in the step sequence. Bit counting makes it possible to visualize the active step as plain text – e.g. in ProTool – by means of symbol lists. For the visualization, it is of advantage to have the bit position of the currently active step as integer value. If only one bit is set in every case, the following formula applies:

$$2^x = Y$$

$$\Rightarrow X = \frac{\ln(Y)}{\ln(2)}$$

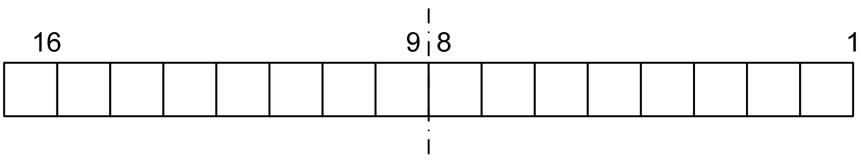
Function "CalcBitPos" (FC 1)

The "CalcBitPos" function determines the position of the set bit in an input data word (16 bit) and returns the position as an INT value via the OUT parameter "bit_pos". If no bit is set, the function returns "0" (zero); if the most significant bit (msb) is set, the function returns "16". If more than one bit is set, the result is undefined.

Note

- The bit counting starts with "1" (see Figure 1-5
- If more than one bit is set, the result is undefined.

Figure 1-5



Block parameters of function “CalcBitPos”

Table 1-6

Parameters	Declaration	Data type	Area	Description
m_word	IN	WORD	I, Q, M, D, L	Data word to be analyzed
bit_pos	OUT	INT	Q, M, D, L	Position of the set bit

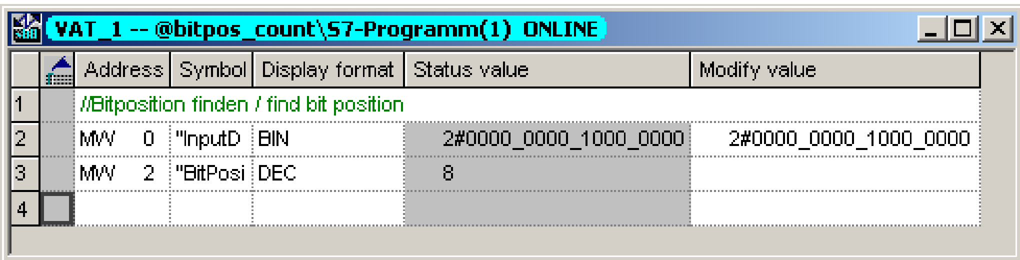
Example

In the appended example project, you can test the functionality of the “CalcBitPos” function. For this purpose, proceed as follows:

Table 1-7

Step	Action / Event
1.	✓ Load the complete station into the CPU or to the S7-PLCSIM.
2.	✓ Open the variable table “VAT_1” in online mode.
3.	✓ Activate the control value (“control variable”) **Result:** MW 2 (“BitPosition”) returns “8” because the 8th bit of flag word “MW 0” (“InputDataWord”) is set.

Figure 1-6



Technical data

Table 1-8

Block	Data
CalcBitPos (FC 1) Determine bit position	Required local data: 24 bytes Load memory requirement: 252 bytes Main memory requirement : 180 bytes

1.4 Edge detection in a 32-bit field

Description

In order to monitor a 32-bit field for any coming or going events, you can check every bit subject to monitoring for positive or negative edges.

For filtering – from any number of bits – the one that changed its status in the cycle, an XOR operator is used on the value of the last cycle and that of the current one. The XOR operator only returns the bit found exclusively in one of the two values. A further AND query determines whether the bit represents a coming or going signal.

The bit position results from incrementing: X + 1, whereby the X-value is determined by means of the following conversion:

$$2^x = Y$$

$$\Rightarrow X = \frac{\ln(Y)}{\ln(2)}$$

Function “Monitor32” (FB 1)

The function “Monitor32” checks whether a bit has changed in a 32-bit data word since the last function call. The function provides the following information as result:

- Bit coming / going
- Bit position (bit counting starts with “1”)

If several bits of the data word have changed since the last function call, the result is undefined.

Block parameters of function “Monitor32”

Table 1-9

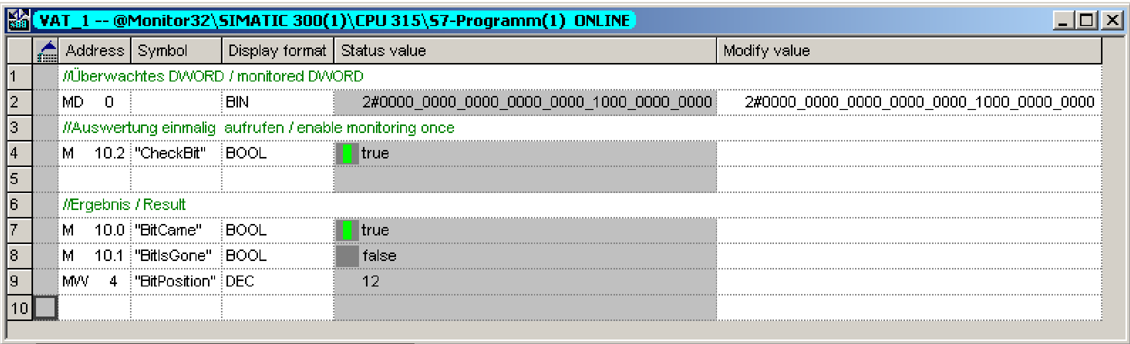
Parameters	Declaration	Data type	Area	Description
value_in	IN	DWORD	I, Q, M, D, L	Input double word the bits of which are to be checked for status changes
come	OUT	BOOL	Q, M, D, L	Bit has been set.
go	OUT	BOOL	Q, M, D, L	Bit has been reset.
bit	OUT	INT	Q, M, D, L	No. of bit come / gone 0: No bit has changed since last function call.

Example

Table 1-10

Step	Action / Event
1.	✓ Load the complete station into the CPU or to the S7-PLCSIM.
2.	✓ Open the variable table “VAT_1” in online mode.
3.	✓ Enter 2#0000_0000_0000_0000_0000_1000_0000_0000 for the “BitField” variable. (12th bit, coming) ✓ Set “CheckBit” to “1” Result: “BitCame” has become “1”, variable “BitPosition” = 12.

Figure 1-7


VAT_1 -- @Monitor32\SIMATIC 300(1)\CPU 315\S7-Programm(1) ONLINE

	Address	Symbol	Display format	Status value	Modify value
1	//Überwachtes DWORD / monitored DWORD				
2	MD 0		BIN	2#0000_0000_0000_0000_0000_1000_0000_0000	2#0000_0000_0000_0000_0000_1000_0000_0000
3	//Auswertung einmalig aufrufen / enable monitoring once				
4	M 10.2	"CheckBit"	BOOL	true	
5					
6	//Ergebnis / Result				
7	M 10.0	"BitCame"	BOOL	true	
8	M 10.1	"BitIsGone"	BOOL	false	
9	MW 4	"BitPosition"	DEC	12	
10					

Technical data

Table 1-11

Block	Data
Monitor32 (FB 1) Edge detection in a 32-bit field	Required local data: 4 bytes Load memory requirement: 318 bytes Main memory requirement : 228 bytes

1.5 Incremental counter with limit of 2,147,483,647

Description

With a STEP 7 counter, you can only count in the range of 0 to 999. Cascading the counters quickly uses up the internal counters. The system function “CTU” (SFB0) also provides a counter, but it can only count to 32,767. In the example described here, you can count up to 2,147,483,647.

Function “Counter” (FB 2)

The function block “Counter” (FB 2) enables you to implement the incremental counting function. A rising edge at the “CU” input (as compared to the last FB call) causes the counter to be incremented by “1”. If the counter reaches the upper limit of **2,147,483,647**, it is not incremented anymore and the “Overflow” output is set. In this case, any further rising edge at the “CU” input has no result. A rising edge at the “R” input causes a counter reset to zero, no matter which value is set at the “CU” input.
A rising edge at the “S” input causes the count value “CV” to be set with the value at the “SW” input.
The “Q” output indicates whether the current count value is greater than or equal to the comparative value “PV”.

Note

- In each CPU cycle, you cannot count more than one counter pulse. In this case, the edges are not detected
- The number **2,147,483,647** is the maximum positive number you can represent with a double word.

Block parameters of function “Counter”

Table 1-12

Parameters	Declaration	Data type	Area	Description
CU	IN	BOOL	I, Q, M, D, L	Counter input
R	IN	BOOL	I, Q, M, D, L	Reset input
S	IN	BOOL	I, Q, M, D, L	Set input
SW	IN	DINT	I, Q, M, D, L	Value to be set (possible values: 0 to 2,147,483,647)
PV	IN	DINT	I, Q, M, D, L, const.	Comparative value See parameter Q for relevance of PV (possible values: 0 to 2,147,483,647).
Q	OUT	BOOL	I, Q, M, D, L	Status of counter: Q has value 1 if CV >= PV

Parameters	Declaration	Data type	Area	Description
				0 otherwise
CV	OUT	DINT	I, Q, M, D, L	Current count value (possible values: 0 to 2,147,483,647)
Overflow	OUT	BOOL	I, Q, M, D, L	Overflow has value 1 if CV >= 2,147,483,647 0 otherwise

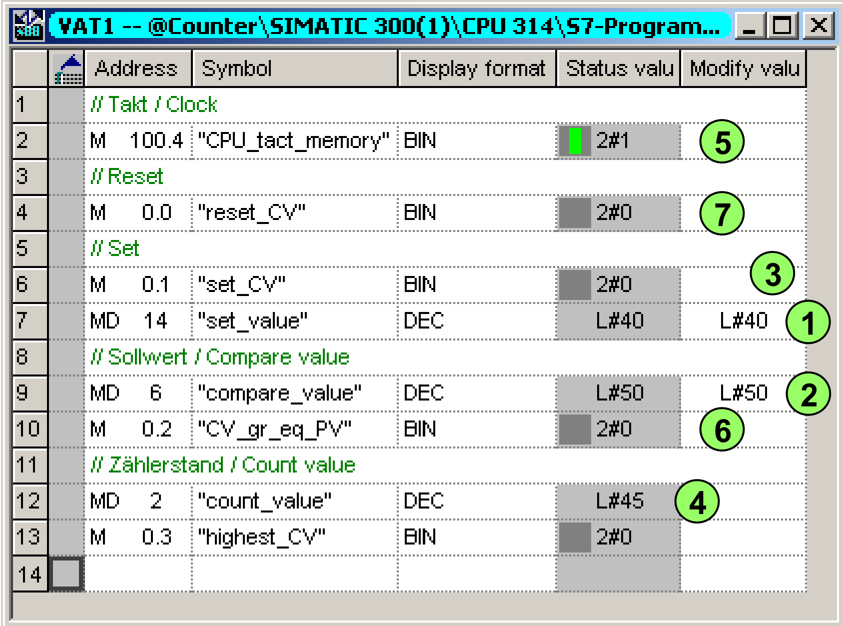
Example

In the example project contained in the download, the function block "Counter" is called. The incremental counting is triggered by the 4th bit of the cycle flag word MW100 parameterized in the CPU. To test the example, proceed as follows:

Table 1-13

Step	Action / Event
1.	✓ Load the complete station into the CPU or to the S7-PLCSIM.
2.	✓ Open the variable table "VAT1" in online mode.
3.	✓ As an example, enter 40 for "set_value". (1)
4.	✓ As an example, enter 50 for "compare_value" and apply "control variable". (2) **Result:** The values in the variable table are updated.
5.	✓ Set the variable "set_CV" to "1" and then reset it to "0". (3) **Result:** The counter ("count_value") is set to 40. (4)
6.	✓ Let a period of 10 cycles elapse. (5) **Result:** After 10 cycles, the counter is set to 50 and the output "CV_gr_eq_PV" is "1". (6)
7.	✓ Set "reset_CV" to "1" (7) **Result:** The counter is reset to "0" ("count_value").

Figure 1-8



Technical data

Table 1-14

Block	Data
Counter (FB 2) Incremental counter with 2,147,483,647 limit	Required local data: 0 bytes Load memory requirement: 264 bytes Main memory requirement : 172 bytes

2 Mathematical Operations

2.1 Calculate the xth root of a REAL number

Description

Since there is no direct command for calculating the xth root ($\sqrt[x]{a}$) in the set of commands provided in STEP 7, the calculation must be performed by means of the “EXP” and “LN” commands. “EXP” calculates the exponential value of a floating-point number to the base “e” and “LN” determines the natural logarithm (logarithm to the base “e”) of a floating-point number.

The following formula describes the relevant mathematical conversion:

$$C = \sqrt[x]{a} = a^{\frac{1}{x}} = e^{\frac{1}{x} \cdot \ln a}$$

Function “X-ROOT”

The X-ROOT function calculates the xth root from an input floating-point number (REAL). The result – returned via an OUT parameter (result) – is of type REAL.

Note

The X-ROOT function provides one possible calculation of the xth root. There are no additional checks of the input values for correctness with respect to mathematical conventions and limits. Therefore, there is no error status.

Block parameters of function “X-ROOT” (FC 23)

Table 2-1

Parameters	Declaration	Data type	Area	Description
number	IN	REAL	I, M, D, L	Number the xth root of which is to be calculated
exponent	IN	REAL	I, M, D, L	Root exponent x
result	OUT	REAL	Q, M, D, L	Result of root calculation

Example

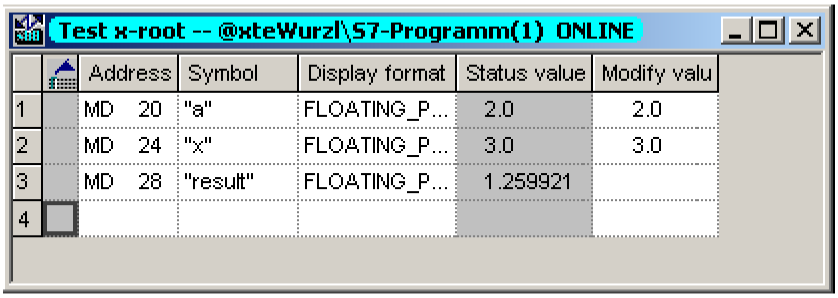
In the program example, the function “CALL_XROOT” (FC 24) is called. In turn, CALL_XROOT calls the function “X-ROOT” (FC 23), which calculates the xth root of “a”. The function has the following formal parameters:

Table 2-2

Symbol	Type	Address	Description
a	REAL	MD20	Number the xth root of which is to be calculated
x	REAL	MD24	Root exponent “x” (xth root)
result	REAL	MD28	Result of $\sqrt[x]{a}$

For testing and monitoring the results, the variable table “Test x-root” is available (see Figure 2-1). In this example $\sqrt[x]{a} = \sqrt[3]{2} \approx 1{,}2599$ is calculated.

Figure 2-1



	Address	Symbol	Display format	Status value	Modify valu
1	MD 20	"a"	FLOATING_P...	2.0	2.0
2	MD 24	"x"	FLOATING_P...	3.0	3.0
3	MD 28	"result"	FLOATING_P...	1.259921	
4					

Technical data

Table 2-3

Block	Data
X-ROOT (FC 23) Calculates the xth root of a REAL number	Required local data: 0 bytes Load memory requirement: 120 bytes Main memory requirement : 64 bytes

The respective download file is available in chapter “Overview of the download files”.

2.2 Calculation of statistical values in automation systems

Description

As a basic requirement, a quality management system must provide a method to check how well a product conforms to the required standards.

The available function FB 20 (SPC01) enables you to perform basic SPC calculations (Statistical Process Control). The function provides the following statistical values:

- Highest value
- Lowest value
- Arithmetic mean
- Standard deviation

For the standard deviation, the following formula is used:

$$\hat{\sigma} = \sqrt{\frac{\sum (x - \bar{x})^2}{n}}$$

With:

x : Measured values

$\bar{x}$: Arithmetic mean

n : Number of measured values

You can use the function for both continuous and discontinuous processes.

With a feedback loop, you can use these data to set control parameters (not implemented in the download).

Function “SPC01” (FB 20)

The SPC01 function calculates statistical values, namely the highest value, the lowest value, the arithmetic mean and the standard deviation.

If the function block is called with “Rec_On” = “false”, the outputs

- Hi
- Lo
- Mean

follow the value at the “PV” input. The outputs

- StdDev
- Count
- Total

- SqrsTotal

are set to 0.0.

If the function block is called with “Rec_On” = “true”, the values of the instance data block and the output parameters are calculated and updated in accordance with the value at “PV”.

For a continuous process, use the function as follows:

Step	Action / Event
1.	✓ Perform a conditional function call.
2.	✓ Set the “Rec_On” input to “true”. **Result:** Data recording starts.
3.	✓ To end the recording, you do not call the function again. Note: If you set “Rec_On” to “false”, all results will be deleted.

For a discontinuous process, use the function as follows:

Step	Action / Event
1.	✓ Set the “Rec_On” input to “true”.
2.	✓ Call the function block conditionally via a one-time call or by means of a rising edge-trigger for the batch period.
3.	✓ When the batch measurement is finished, set “RecOn” to “false”.
4.	✓ Set “RecOn” to “true” again to restart collecting data.

Block parameters of function “SPC01” (FB 20)

Table 2-4

Parameters	Declaration	Data type	Area	Description
PV	IN	REAL	I, M, D, L	Recorded process variable
Rec_On	IN	REAL	I, M, D, L	If “true”, value recording takes place; if “false”, the output values follow the current PV.
Hi	OUT	REAL	Q, M, D, L	The highest value recorded since “Rec_On” was last toggled from “false” to “true”
Lo	OUT	REAL	Q, M, D, L	The lowest value recorded since “Rec_On” was last toggled from “false” to “true”
Mean	OUT	REAL	Q, M, D, L	The arithmetic mean of the PV values since “Rec_On” was last toggled from “false” to “true”
StdDev	OUT	REAL	Q, M, D, L	The standard deviation of the PV values since “Rec_On” was last

Parameters	Declaration	Data type	Area	Description
				toggled from "false" to "true"
Count	OUT	REAL	Q, M, D, L	The number of recorded PV measurements (auxiliary value) since "Rec_On" was last toggled from "false" to "true"
Total	OUT	REAL	Q, M, D, L	The total of the PV values (auxiliary value) since "Rec_On" was last toggled from "false" to "true"
SqrsTotal	OUT	REAL	Q, M, D, L	The total of the squared PV values (auxiliary value) since "Rec_On" was last toggled from "false" to "true"

Note This function block is not suitable for logging individual PV values. In order to use the CPU memory sparingly, the function block calculates the results of a data collection immediately and only stores such data as are necessary to establish the output values. The analysis takes place at runtime. If you need to store the process variables for a subsequent statistical evaluation, the process values should be archived outside the CPU (e.g. in WinCC).

Example

In the organization block OB1, the function SPC01 (FB 20) is called every 15 seconds. The "PV" parameter gets its data via the flag double word MD2. The results are written to the following flags:

Hi ➔ MD6

Lo ➔ MD10

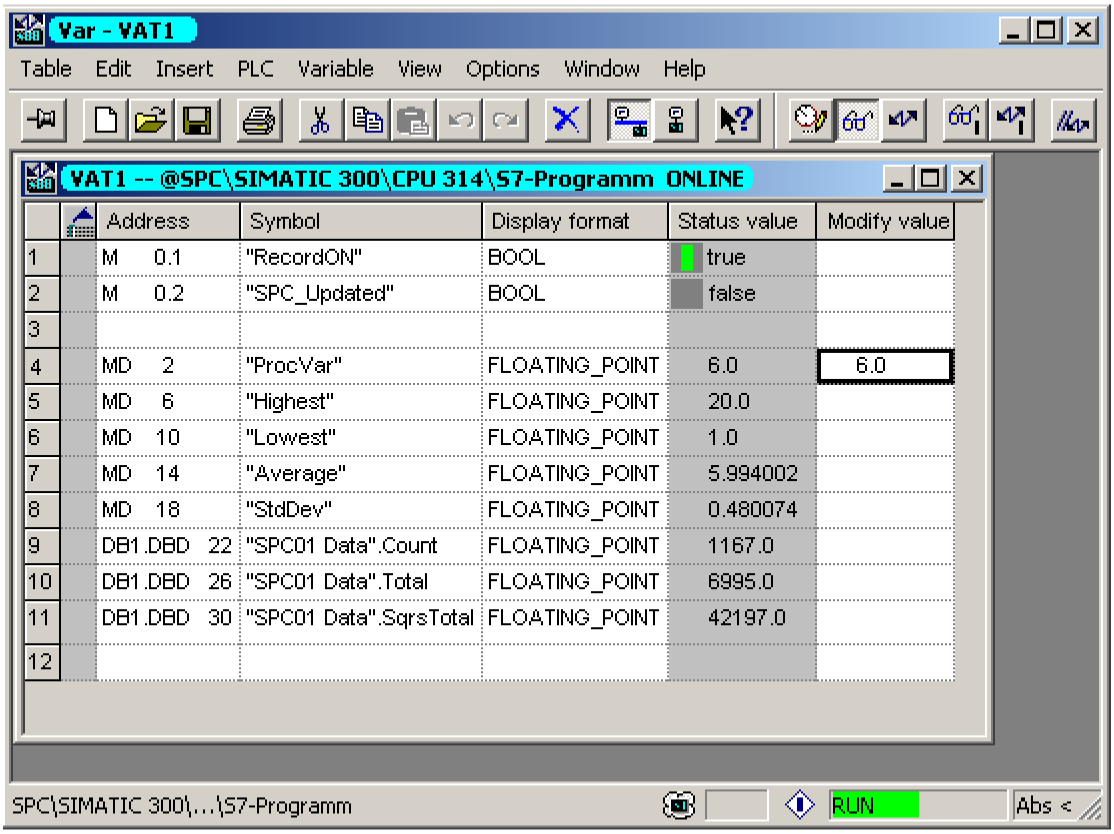
Mean ➔ MD14

StdDev ➔ MD18

In order to test the example, proceed as follows:

Step	Action / Event
1.	✓ Load the complete station into the CPU or to the S7-PLCSIM. ✓ Open the “VAT1” variable table.
2.	✓ Set M0.1 (“RecordON”) to “true”. **Result**: The recording of measured values starts.
3.	✓ Enter a value in MD2 (“ProcVar”) within 15 seconds. Result: Every 15 seconds, the variables • Hi • Lo • Mean • StdDev • "SPC01 Data".Count • "SPC01 Data".Total • "SPC01 Data".SqrsTotal are updated.
4.	✓ Repeat step 3 as often as you like.
5.	✓ When you are finished, set M0.1 (“RecordON”) to “false”. Result: Within 15 seconds, all values are reset or set to the value at the “PV” input.
6.	✓ If you want to start a new series of measurements, go back to step 2.

Figure 2-2



Technical data

Table 2-5

Block	Data
SPC01 (FB20) Calculation of statistical values	Required local data: 6 bytes Load memory requirement: 360 bytes Main memory requirement : 284 bytes

The respective download file is available in chapter "Overview of the download files".

2.3 Matrix operations in SIMATIC systems

Various high-level functions for process control deal with several in- and output values and therefore matrices and matrix operations are required to define them in the technical literature. Application examples include state controllers, monitoring devices, Kalman filters, predictive controllers and process simulators.

Such functions are quite easy to implement on a PC by including ready-made numerical libraries while it is rather difficult to implement them in SIMATIC systems (S7 or PCS7) because you have to program all matrix operations manually by means of interlaced loops in the high-level language SIMATIC SCL.

The library “MatrixOperations” offers a solution to this problem by providing ready-made functions for processing matrices in SCL. To define a matrix, you use a UDT (user-defined data type), which requires header information on the number of rows and columns and contains the matrix elements in form of a two-dimensional array. Vectors are included in the definition as special matrices consisting of only one column. For all variables of this data type, at least the following ready-made functions are available as “FCs”:

- Matrix addition
- Subtraction
- Matrix multiplication
- Transposition
- Inversion

In addition, there are functions available for creating a zero matrix or an identity matrix of specified dimensions.

Examples

In the download, there is an example of each matrix function with the exception of the functions “MxEin” for creating an identity matrix and “MxNull” for creating a zero matrix. Each example has its own variable table and a test FB, which is called cyclically in OB1. To test the examples, proceed as follows:

Table 2-6

Step	Action / Event
1.	✓ Create a new project in the SIMATIC Manager. ✓ Configure a SIMATIC station.
2.	✓ Open the “MatrixOperations” library and copy all elements (including the variable tables) to your project.
3.	✓ Load the complete station into the CPU or to the S7-PLCSIM.
4.	✓ Open the relevant variable table in online mode. **Note:** You can find the relevant variable tables in the section of the respective functions (see below).

User-defined data type (UDT): Matrix

The user-defined data type MATRIX has the following data structure:

```
STRUCT
    No. of rows: INT: = 0;
    No. of columns: INT: = 0;
    Elements: ARRAY[1..4,1..4] OF REAL;
END_STRUCT
```

The integer variables “No. of rows” and “No. of columns” define the dimension of the matrix; the two-dimensional array contains the matrix elements of type “REAL”. At the same time, the dimension of the array represents the maximum permissible size of a matrix.

Note

- Since it is not possible to adjust the dimension of the array dynamically, you have to specify the size of the field in advance when declaring the variable. For this purpose, you have to make sure that the selected dimension can accommodate the maximum-sized matrix of the S7 program. The default size of the array is 4 x 4. You can adjust the dimension in the source file “Matrix_UDT”. To implement the change, you have to compile the source file.
- For the sake of simplicity, the MATRIX data type includes the definition of vectors so that no extra data type is necessary.

Function for matrix addition: “MxAdd” (FC 501)

The “MxAdd” function serves for the addition of two $m \times n$ matrices of data type MATRIX using the following calculation formula:

$A + B = (a_{ij} + b_{ij})$ with $i = 1,...,m; j = 1,...,n$

Before the calculation takes place, the function checks whether both matrices have the same dimension. In case of an error, it returns a matrix of dimension 0×0 as result.

Calculation example:

$$\begin{pmatrix} 1 & 3 & 2 \\ 1 & 2 & 2 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 5 \\ 2 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 7 \\ 3 & 3 & 3 \end{pmatrix}$$

In SCL, you call the function with:

```
MxAdd(MxA:= Am, MxB:= Bm, MxC:= AplusB);
```

Block parameters of function “MxAdd”

Table 2-7

Parameters	Declaration	Data type	Area	Description
MxA	IN	MATRIX	L	Matrix A
MxB	IN	MATRIX	L	Matrix B
MxC	OUT	MATRIX	L	Sum of matrices A+B

Example

In the variable table “VAT_MxAdd”, you can see the result of the matrix addition exemplified above (see Figure 2-3). As to how to apply the example, see 2.3.

Figure 2-3

VAT_MxAdd -- @TestMx\SIMATIC 300(1)\CPU 315\S7-Programm(1) ONLI...

	Operand	Symbol	Anzeigeformat	Statuswert	Steuerwert
1	//Addition zweier Matrizen / addition of two matrices				
2					
3	//Matrix Am				
4	//erste Zeile / first row				
5	DB51.DBD 4	"DITestMxAdd".Am.elemente[1, 1]	GLEITPUNKT	1.0	
6	DB51.DBD 8	"DITestMxAdd".Am.elemente[1, 2]	GLEITPUNKT	3.0	
7	DB51.DBD 12	"DITestMxAdd".Am.elemente[1, 3]	GLEITPUNKT	2.0	
8	//zweite Zeile / second row				
9	DB51.DBD 20	"DITestMxAdd".Am.elemente[2, 1]	GLEITPUNKT	1.0	
10	DB51.DBD 24	"DITestMxAdd".Am.elemente[2, 2]	GLEITPUNKT	2.0	
11	DB51.DBD 28	"DITestMxAdd".Am.elemente[2, 3]	GLEITPUNKT	2.0	
12					
13	//Matrix Bm				
14	//erste Zeile / first row				
15	DB51.DBD 72	"DITestMxAdd".Bm.elemente[1, 1]	GLEITPUNKT	0.0	
16	DB51.DBD 76	"DITestMxAdd".Bm.elemente[1, 2]	GLEITPUNKT	0.0	
17	DB51.DBD 80	"DITestMxAdd".Bm.elemente[1, 3]	GLEITPUNKT	5.0	
18	//zweite Zeile / second row				
19	DB51.DBD 88	"DITestMxAdd".Bm.elemente[2, 1]	GLEITPUNKT	2.0	
20	DB51.DBD 92	"DITestMxAdd".Bm.elemente[2, 2]	GLEITPUNKT	1.0	
21	DB51.DBD 96	"DITestMxAdd".Bm.elemente[2, 3]	GLEITPUNKT	1.0	
22					
23	// Am + Bm = Cm				
24	//Ergebnis nach Addition / result after addition				
25	DB51.DBD 140	"DITestMxAdd".Cm.elemente[1, 1]	GLEITPUNKT	1.0	
26	DB51.DBD 144	"DITestMxAdd".Cm.elemente[1, 2]	GLEITPUNKT	3.0	
27	DB51.DBD 148	"DITestMxAdd".Cm.elemente[1, 3]	GLEITPUNKT	7.0	
28	DB51.DBD 156	"DITestMxAdd".Cm.elemente[2, 1]	GLEITPUNKT	3.0	
29	DB51.DBD 160	"DITestMxAdd".Cm.elemente[2, 2]	GLEITPUNKT	3.0	
30	DB51.DBD 164	"DITestMxAdd".Cm.elemente[2, 3]	GLEITPUNKT	3.0	
31					

Technical data

Table 2-8

Block	Data
MxAdd (FC 501) Addition of two matrices	Required local data: 20 bytes Load memory requirement: 916 bytes Main memory requirement : 790 bytes

Function for matrix subtraction: "MxSub" (FC 505)

The "MxSub" function subtracts two matrices of data type MATRIX from each other. Before the calculation takes place, the function checks whether both matrices have the same dimension and returns a 0x0 matrix in case of error in this case as well.

Calculation example:

$$\begin{pmatrix} 1 & 3 & 2 \\ 1 & 2 & 2 \end{pmatrix} - \begin{pmatrix} 0 & 0 & 5 \\ 2 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 3 & -3 \\ -1 & 1 & 1 \end{pmatrix}$$

In SCL, you call the function with:

```
MxSub(MxA:= Am, MxB:= Bm, MxC:= AminusB);
```

Block parameters of function "MxSub"

Table 2-9

Parameters	Declaration	Data type	Area	Description
MxA	IN	MATRIX	L	Matrix A
MxB	IN	MATRIX	L	Matrix B
MxC	OUT	MATRIX	L	Difference of matrices A-B

Example

In the variable table "VAT_MxSub", you can see the result of the matrix subtraction exemplified above (see Figure 2-4). As to how to apply the example, see 2.3.

Figure 2-4

VAT_MxSub -- @TestMx\SIMATIC 300(1)\CPU 315\S7-Programm(1) ONLINE

	Operand	Symbol	Anzeigeformat	Statuswert	Steuerwert
1	//Subtraktion zweier Matrizen / subtraction of two matrices				
2					
3	//Matrix A				
4	//erste Zeile / first row				
5	DB55.DBD 4	"DITestMxSub".Am.elemente[1, 1]	GLEITPUNKT	1.0	
6	DB55.DBD 8	"DITestMxSub".Am.elemente[1, 2]	GLEITPUNKT	3.0	
7	DB55.DBD 12	"DITestMxSub".Am.elemente[1, 3]	GLEITPUNKT	2.0	
8	//zweite Zeile / second row				
9	DB55.DBD 20	"DITestMxSub".Am.elemente[2, 1]	GLEITPUNKT	1.0	
10	DB55.DBD 24	"DITestMxSub".Am.elemente[2, 2]	GLEITPUNKT	2.0	
11	DB55.DBD 28	"DITestMxSub".Am.elemente[2, 3]	GLEITPUNKT	2.0	
12					
13	//Matrix Bm				
14	//erste Zeile / first row				
15	DB51.DBD 72	"DITestMxAdd".Bm.elemente[1, 1]	GLEITPUNKT	0.0	
16	DB51.DBD 76	"DITestMxAdd".Bm.elemente[1, 2]	GLEITPUNKT	0.0	
17	DB51.DBD 80	"DITestMxAdd".Bm.elemente[1, 3]	GLEITPUNKT	5.0	
18	//zweite Zeile / second row				
19	DB51.DBD 88	"DITestMxAdd".Bm.elemente[2, 1]	GLEITPUNKT	2.0	
20	DB51.DBD 92	"DITestMxAdd".Bm.elemente[2, 2]	GLEITPUNKT	1.0	
21	DB51.DBD 96	"DITestMxAdd".Bm.elemente[2, 3]	GLEITPUNKT	1.0	
22					
23	//Am - Bm = Cm				
24	//Ergebnis nach Subtraktion / result aftersubtraction				
25	DB55.DBD 140	"DITestMxSub".Cm.elemente[1, 1]	GLEITPUNKT	1.0	
26	DB55.DBD 144	"DITestMxSub".Cm.elemente[1, 2]	GLEITPUNKT	3.0	
27	DB55.DBD 148	"DITestMxSub".Cm.elemente[1, 3]	GLEITPUNKT	-3.0	
28	DB55.DBD 156	"DITestMxSub".Cm.elemente[2, 1]	GLEITPUNKT	-1.0	
29	DB55.DBD 160	"DITestMxSub".Cm.elemente[2, 2]	GLEITPUNKT	1.0	
30	DB55.DBD 164	"DITestMxSub".Cm.elemente[2, 3]	GLEITPUNKT	1.0	
31					

Technical data

Table 2-10

Block	Data
MxSub (FC 505) Subtraction of two matrices	Required local data: 20 bytes Load memory requirement: 916 bytes Main memory requirement : 790 bytes

Function for matrix multiplication: "MxMul" (FC 503)

For multiplying two matrices, the "MxMul" function is available. The calculation only takes place if the number of columns of the first matrix corresponds to the number of rows of the second one. As before, the function returns a matrix of dimension 0x0 if this condition does not apply. The calculation formula for multiplying two matrices A and B with each other is:

$$A = (a_{ij}) \quad \text{with} \quad i = 1,...,l;\ j = 1,...,m$$

$$B = (a_{ij}) \quad \text{with} \quad i = 1,...,m;\ j = 1,...,n$$

$$A \cdot B = (c_{ij}) \quad \text{with} \quad i = 1,...,l;\ j = 1,...,n \ and\ c_{ij} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \bullet \begin{pmatrix} 6 & -1 \\ 3 & 2 \\ 0 & -3 \end{pmatrix} = \begin{pmatrix} 12 & -6 \\ 39 & -12 \end{pmatrix}$$

In SCL, you call the function with:

```
MxMul(MxA:= Am, MxB:= Bm, MxC:= Cm);
```

Block parameters of function "MxMul"

Table 2-11

Parameters	Declaration	Data type	Area	Description
MxA	IN	MATRIX	L	Matrix A to be multiplied
MxB	IN	MATRIX	L	Matrix B to be multiplied
MxC	OUT	MATRIX	L	Matrix product A*B

Example

In the variable table "VAT_MxMul", you can see the result of the matrix multiplication exemplified above (see Figure 2-5). As to how to apply the example, see 2.3.

Figure 2-5

VAT_MxMul -- @TestMx\SIMATIC 300(1)\CPU 315\S7-Programm(1) ONLI...

	Operand	Symbol	Anzeigeformat	Statuswert	Steuerwert
1	//Subtraktion zweier Matrizen / subtraction of two matrices				
2					
3	//Matrix Am				
4	//erste Zeile / first row				
5	DB53.DBD 4	"DITestMxMul".Am.elemente[1, 1]	GLEITPUNKT	1.0	
6	DB53.DBD 8	"DITestMxMul".Am.elemente[1, 2]	GLEITPUNKT	2.0	
7	DB53.DBD 12	"DITestMxMul".Am.elemente[1, 3]	GLEITPUNKT	3.0	
8	//zweite Zeile / second row				
9	DB53.DBD 20	"DITestMxMul".Am.elemente[2, 1]	GLEITPUNKT	4.0	
10	DB53.DBD 24	"DITestMxMul".Am.elemente[2, 2]	GLEITPUNKT	5.0	
11	DB53.DBD 28	"DITestMxMul".Am.elemente[2, 3]	GLEITPUNKT	6.0	
12					
13	//Matrix Bm				
14	//erste Zeile / first row				
15	DB53.DBD 72	"DITestMxMul".Bm.elemente[1, 1]	GLEITPUNKT	6.0	
16	DB53.DBD 76	"DITestMxMul".Bm.elemente[1, 2]	GLEITPUNKT	-1.0	
17	//zweite Zeile / second row				
18	DB53.DBD 88	"DITestMxMul".Bm.elemente[2, 1]	GLEITPUNKT	3.0	
19	DB53.DBD 92	"DITestMxMul".Bm.elemente[2, 2]	GLEITPUNKT	2.0	
20	//dritte Zeile / third row				
21	DB53.DBD 104	"DITestMxMul".Bm.elemente[3, 1]	GLEITPUNKT	0.0	
22	DB53.DBD 108	"DITestMxMul".Bm.elemente[3, 2]	GLEITPUNKT	-3.0	
23					
24	//Am * Bm = Cm				
25	//Ergebnis nach Subtraktion / result aftersubtraction				
26	DB53.DBD 140	"DITestMxMul".Cm.elemente[1, 1]	GLEITPUNKT	12.0	
27	DB53.DBD 144	"DITestMxMul".Cm.elemente[1, 2]	GLEITPUNKT	-6.0	
28	DB53.DBD 156	"DITestMxMul".Cm.elemente[2, 1]	GLEITPUNKT	39.0	
29	DB53.DBD 160	"DITestMxMul".Cm.elemente[2, 2]	GLEITPUNKT	-12.0	
30					

Technical data

Table 2-12

Block	Data
MxMul (FC 503) Multiplication of two matrices	Required local data: 26 bytes Load memory requirement: 956 bytes Main memory requirement : 820 bytes

Function for matrix transposition: "MxTrans" (FC 506)

The "MxTrans" function returns the transposed matrix of an input matrix.

The formula for determining the transposed matrix A^T of matrix A is:

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

The transpose of matrix A is:

$$A^T = \begin{pmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{pmatrix}$$

Calculation example:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

In SCL, you call the function with:

```
MxTrans(MxA:= Am, MxAT:= Atrans);
```

Block parameters of function "MxTrans"

Table 2-13

Parameters	Declaration	Data type	Area	Description
MxA	IN	MATRIX	L	Matrix A to be transposed
MxAT	OUT	MATRIX	L	Transpose of matrix A

Example

In the variable table "VAT_MxTrans", you can see the result of the matrix transposition exemplified above (see Figure 2-6). As to how to apply the example, see 2.3.

Figure 2-6

VAT_MxTrans -- @TestMx\SIMATIC 300(1)\CPU 315\S7-Programm(1) ONL...

	Operand	Symbol	Anzeigeformat	Statuswert	Steuerwert
1	//Transponieren einer Matrix / transpose a matrix				
2					
3	//Matrix A				
4	//erste Zeile / first row				
5	DB56.DBD 4	"DITestMxTrans".Am.elemente[1, 1]	GLEITPUNKT	1.0	
6	DB56.DBD 8	"DITestMxTrans".Am.elemente[1, 2]	GLEITPUNKT	2.0	
7	DB56.DBD 12	"DITestMxTrans".Am.elemente[1, 3]	GLEITPUNKT	3.0	
8	//zweite Zeile / second row				
9	DB56.DBD 20	"DITestMxTrans".Am.elemente[2, 1]	GLEITPUNKT	4.0	
10	DB56.DBD 24	"DITestMxTrans".Am.elemente[2, 2]	GLEITPUNKT	5.0	
11	DB56.DBD 28	"DITestMxTrans".Am.elemente[2, 3]	GLEITPUNKT	6.0	
12					
13					
14	//Am^T				
15	//Ergebnis nach dem Transponieren / result after transpose				
16	//erste Zeile / first row				
17	DB56.DBD 72	"DITestMxTrans".Cm.elemente[1, 1]	GLEITPUNKT	1.0	
18	DB56.DBD 76	"DITestMxTrans".Cm.elemente[1, 2]	GLEITPUNKT	4.0	
19	//zweite Zeile / second row				
20	DB56.DBD 88	"DITestMxTrans".Cm.elemente[2, 1]	GLEITPUNKT	2.0	
21	DB56.DBD 92	"DITestMxTrans".Cm.elemente[2, 2]	GLEITPUNKT	5.0	
22	//dritte Zeile / third row				
23	DB56.DBD 104	"DITestMxTrans".Cm.elemente[3, 1]	GLEITPUNKT	3.0	
24	DB56.DBD 108	"DITestMxTrans".Cm.elemente[3, 2]	GLEITPUNKT	6.0	
25					

Technical data

Table 2-14

Block	Data
MxTrans (FC 506) Transposition of a matrix	Required local data: 16 bytes Load memory requirement: 588 bytes Main memory requirement : 486 bytes

Function for matrix inversion: FC "MxInv" (FC 502)

The "MxInv" function serves for inverting a non-singular square matrix. Since the dimension of the matrix to be inverted within the estimation algorithm is low, meaning that the efficiency differences between the various inversion algorithms are hardly significant, a very simple algorithm, the so-called Shipley-Coleman algorithm is applied. Besides its simplicity, this inversion algorithm stands out in that it is an "in-place" algorithm so there is no additional auxiliary variable of type MATRIX required for the inversion. If the function receives a non-square matrix as input, it returns a 0x0 matrix as result.

In SCL, you call the function with:

```
MxInv(MxA:= Am, MxAI:= Ainv);
```

Block parameters of function "MxInv"

Table 2-15

Parameters	Declaration	Data type	Area	Description
MxA	IN	MATRIX	L	Matrix to be inverted
MxAI	OUT	MATRIX	L	Contains inverse matrix

Example

In this example, the following matrix is inverted:

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 3 & 0 \\ 3 & 4 & 1 \end{pmatrix}$$

The inverse matrix A^{-1} is:

$$A^{-1} = \begin{pmatrix} -3 & 2 & 0 \\ 2 & -1 & 0 \\ 1 & -2 & 1 \end{pmatrix}$$

You can find the result in variable table "VAT_MxInv" (Figure 2-7). As to how to apply the example, see 2.3.

Figure 2-7

VAT_MxInv -- @TestMx\SIMATIC 300(1)\CPU 315\S7-Programm(1) ONLINE

	Operand	Symbol	Anzeigeformat	Statuswert	Steuerwert
1	//Inverieren einer Matrix / invert a matrix				
2					
3	//Matrix A				
4	//erste Zeile / first row				
5	DB52.DBD 4	"DITestMxInv".Am.elemente[1, 1]	GLEITPUNKT	1.0	
6	DB52.DBD 8	"DITestMxInv".Am.elemente[1, 2]	GLEITPUNKT	2.0	
7	DB52.DBD 12	"DITestMxInv".Am.elemente[1, 3]	GLEITPUNKT	0.0	
8	//zweite Zeile / second row				
9	DB52.DBD 20	"DITestMxInv".Am.elemente[2, 1]	GLEITPUNKT	2.0	
10	DB52.DBD 24	"DITestMxInv".Am.elemente[2, 2]	GLEITPUNKT	3.0	
11	DB52.DBD 28	"DITestMxInv".Am.elemente[2, 3]	GLEITPUNKT	0.0	
12	//dritte Zeile / third row				
13	DB52.DBD 36	"DITestMxInv".Am.elemente[3, 1]	GLEITPUNKT	3.0	
14	DB52.DBD 40	"DITestMxInv".Am.elemente[3, 2]	GLEITPUNKT	4.0	
15	DB52.DBD 44	"DITestMxInv".Am.elemente[3, 3]	GLEITPUNKT	1.0	
16					
17	//Am^-1				
18	//Ergebnis nach dem Invertieren / result after invert				
19	//erste Zeile / first row				
20	DB52.DBD 72	"DITestMxInv".Cm.elemente[1, 1]	GLEITPUNKT	-3.0	
21	DB52.DBD 76	"DITestMxInv".Cm.elemente[1, 2]	GLEITPUNKT	2.0	
22	DB52.DBD 80	"DITestMxInv".Cm.elemente[1, 3]	GLEITPUNKT	0.0	
23	//zweite Zeile / second row				
24	DB52.DBD 88	"DITestMxInv".Cm.elemente[2, 1]	GLEITPUNKT	2.0	
25	DB52.DBD 92	"DITestMxInv".Cm.elemente[2, 2]	GLEITPUNKT	-1.0	
26	DB52.DBD 96	"DITestMxInv".Cm.elemente[2, 3]	GLEITPUNKT	0.0	
27	//dritte Zeile / third row				
28	DB52.DBD 104	"DITestMxInv".Cm.elemente[3, 1]	GLEITPUNKT	1.0	
29	DB52.DBD 108	"DITestMxInv".Cm.elemente[3, 2]	GLEITPUNKT	-2.0	
30	DB52.DBD 112	"DITestMxInv".Cm.elemente[3, 3]	GLEITPUNKT	1.0	
31					

Technical data

Table 2-16

Block	Data
MxInv (FC 502) Inversion of a matrix	Required local data: 26 bytes Load memory requirement: 3182 bytes Main memory requirement : 3046 bytes

Function for creating zero matrices: “MxNull” (FC 504)

The “MxNull” function serves for generating square zero matrices. The function gets the dimension of the zero matrix via the input parameter “dim”. If the specified “dim” value is less than or equal to zero, the function returns a matrix of dimension 0x0 as result.

In SCL, you call the function with:

```
MxNull(Dim:= n, MxN:= Nm);
```

Block parameters of function “MxNull”

Table 2-17

Parameters	Declaration	Data type	Area	Description
Dim	IN	INT	L	Dimension of matrix
MxN	OUT	MATRIX	L	Contains the square zero matrix of dimension “dim”

Technical data

Table 2-18

Block	Data
MxNull (FC 504) Creation of a square zero matrix	Required local data: 20 bytes Load memory requirement: 462 bytes Main memory requirement : 374 bytes

Function for creating identity matrices: “MxEin” (FC 507)

The “MxEin” function serves for creating square identity matrices. As above, you can specify the dimension by means of the input parameter “dim”. The error output is identical to that of the “MxNull” function.

In SCL, you call the function with:

```
MxEin (Dim:= n, MxI:= Im);
```

Block parameters of function “MxEin”

Table 2-19

Parameters	Declaration	Data type	Area	Description
Dim	IN	INT	L	Dimension of matrix
MxI	OUT	MATRIX	L	Contains the square identity matrix of dimension “dim”

Technical data

Table 2-20

Block	Data
MxEin (FC 507) Creation of a square identity matrix	Required local data: 20 bytes Load memory requirement: 594 bytes Main memory requirement : 502 bytes

2.4 Multidimensional interpolation

Description

In process automation, it is often necessary to calculate values that are difficult or even impossible to express in formulas. In this case, it is suitable to interpolate the function values.

However, a simple linear interpolation between two data points is often insufficient. It is often reasonable to represent the function in a table by means of several interpolation points and – if required – to select the best pair of data points in question for the interpolation.

In other cases, the function depends on several parameters so that a multidimensional interpolation is required. For instance, the response time does not only depend on the temperature. The pressure and concentration of the reagents are relevant as well.

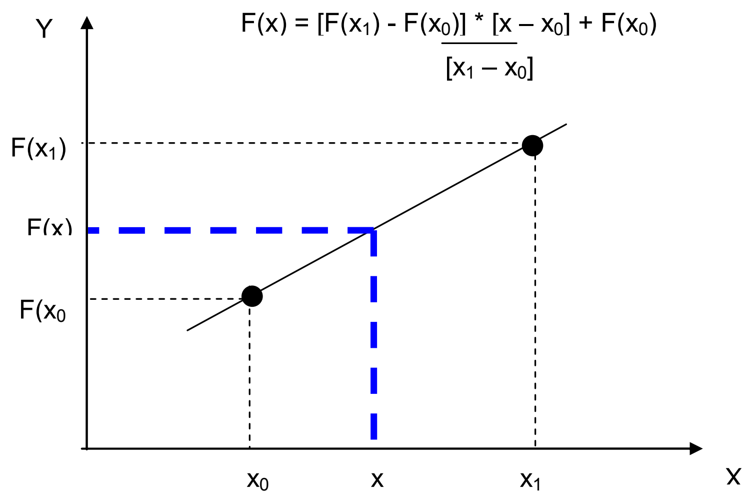
Function “INTERP_2POINT” FC 100

The function “INTERP_2POINT” performs a simple linear interpolation between two points if the function follows a line ($F(x) \sim a \cdot x + b$). The following calculation formula applies to the interpolation:

$$F(x) = \left(F(x_1) - F(x_0)\right) \cdot \frac{x - x_0}{x_{i+1} - x_0} + F(x_0)$$

See also Figure 2-8

Figure 2-8



Block parameters of function "INTERP_2POINT" FC 100

Table 2-21

Parameters	Declaration	Data type	Area	Description
X	IN	REAL	I, Q, M, D, L	Input value
Xn	IN	REAL	I, Q, M, D, L	X-value of max. interpolation point
Xo	IN	REAL	I, Q, M, D, L	X-value of min. interpolation point
Fxn	IN	REAL	I, Q, M, D, L	F(xn) of max. interpolation point
Fxo	IN	REAL	I, Q, M, D, L	F(xo) of min. interpolation point
FX	OUT	REAL	Q, M, D, L	Interpolated value F(x)
ERROR	OUT	BOOL	Q, M, D, L	1: Error, 0: No error
STATUS	OUT	WORD	Q, M, D, L	Status: 0000: no error 7001: X<Xo 7002: X>Xn 8191: Xn = Xo -> Division by zero!

Example

In the project folder of "Interpol.zip", you can find the variable table "VAT_1" to test the functionality of function "INTERP_2POINT". The function is called cyclically in OB 1, where it is provided with the required parameters. The example illustrates a temperature conversion from degrees Celsius to degrees Fahrenheit. To test the example, proceed as follows:

Table 2-22

Step	Action / Event
1.	✓ Load the complete station into the CPU or to the S7-PLCSIM and start the CPU.
2.	✓ Open the variable table "VAT_1" in online mode.
3.	✓ Enter, for instance, 30 as value in the variable table and transfer it to control. (1) Result: The function returns value 86 as result. This corresponds to a conversion from 30°Celcius to 86°Fahrenheit. (2)

Technical data

Figure 2-9

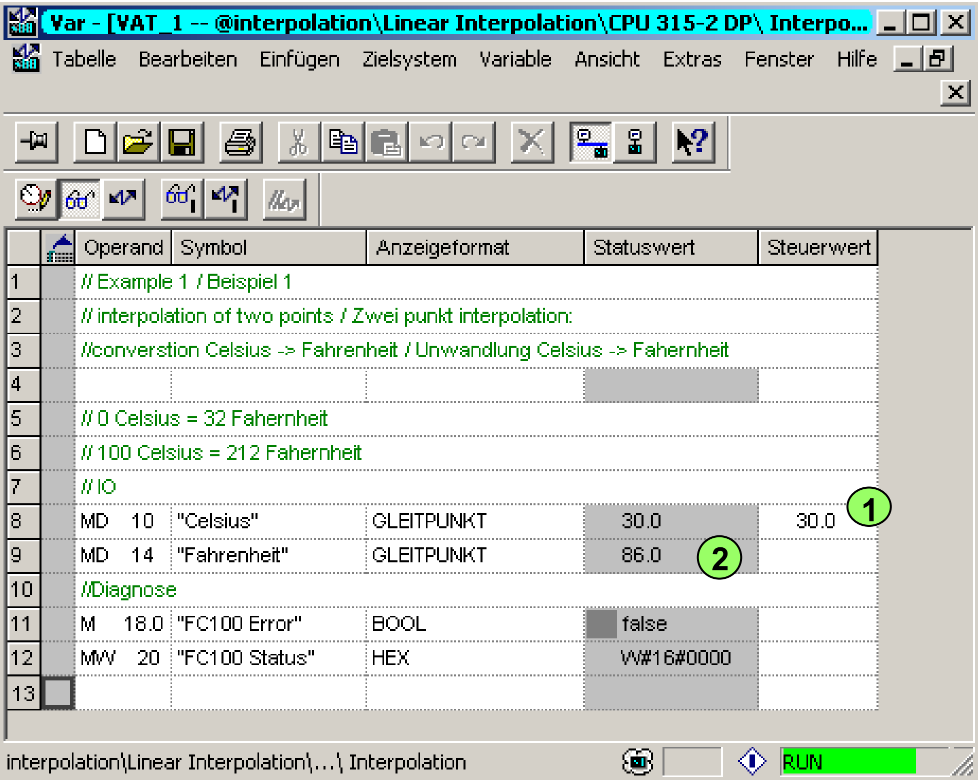


Table 2-23

Block	Data
INTERP_2POINT FC 100 Linear interpolation between two data points	Required local data: 6 bytes Load memory requirement: 300 bytes Main memory requirement : 222 bytes

Function “INTERP_1D” FB 1

The function “INTERP_1D” FB 1 enables you to perform an interpolation of a one-dimensional function F(x). You approximate the function by means of several interpolation points. The interpolation points are saved to the instance data block of the function and must be manually written (see example). You can modify the number of possible interpolation points in the SCL source file “INTERP_1D”. The default value is 10 interpolation points.

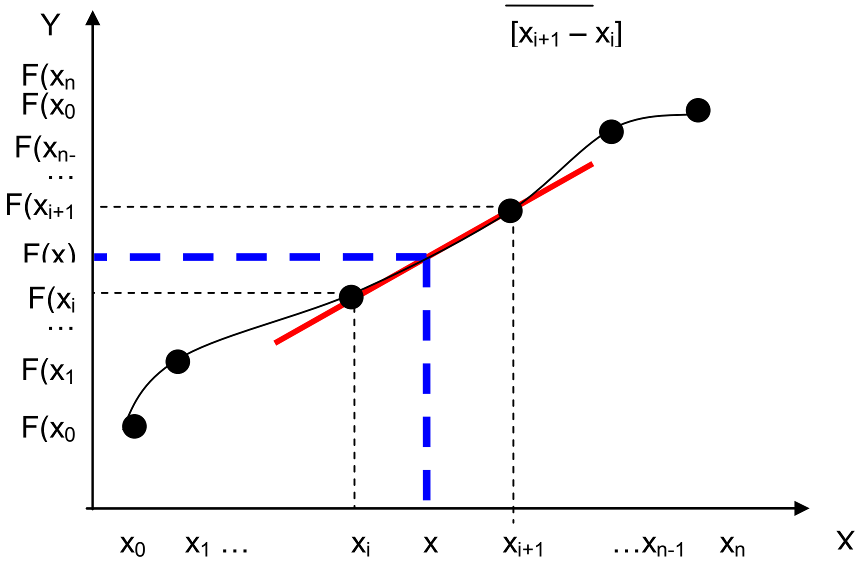
Note

- Each X-value has a corresponding function value F(x).
- You can modify the number of interpolation points by means of the "Nx" constant in the SCL source.
- If you change the number of possible interpolation points, you have to compile the SCL source anew.

The following formula applies to the interpolation:

Figure 2-10

$$F(x) = \frac{[F(x_{i+1}) - F(x_i)] * [x - x_i]}{[x_{i+1} - x_i]} + F(x_i)$$



Block parameters of function "INTERP_1D" FB 1

Table 2-24

Parameters	Declaration	Data type	Area	Description
X	IN	REAL	I, Q, M, D, L	Input value
FX	OUT	REAL	Q, M, D, L	Interpolated value F(x)
ERROR	OUT	BOOL	Q, M, D, L	1: Error, 0: No error
STATUS	OUT	WORD	Q, M, D, L	Status: 0000: no error 7001: X<Xo or X>Xn 8191: Xn = Xo -> Division by zero

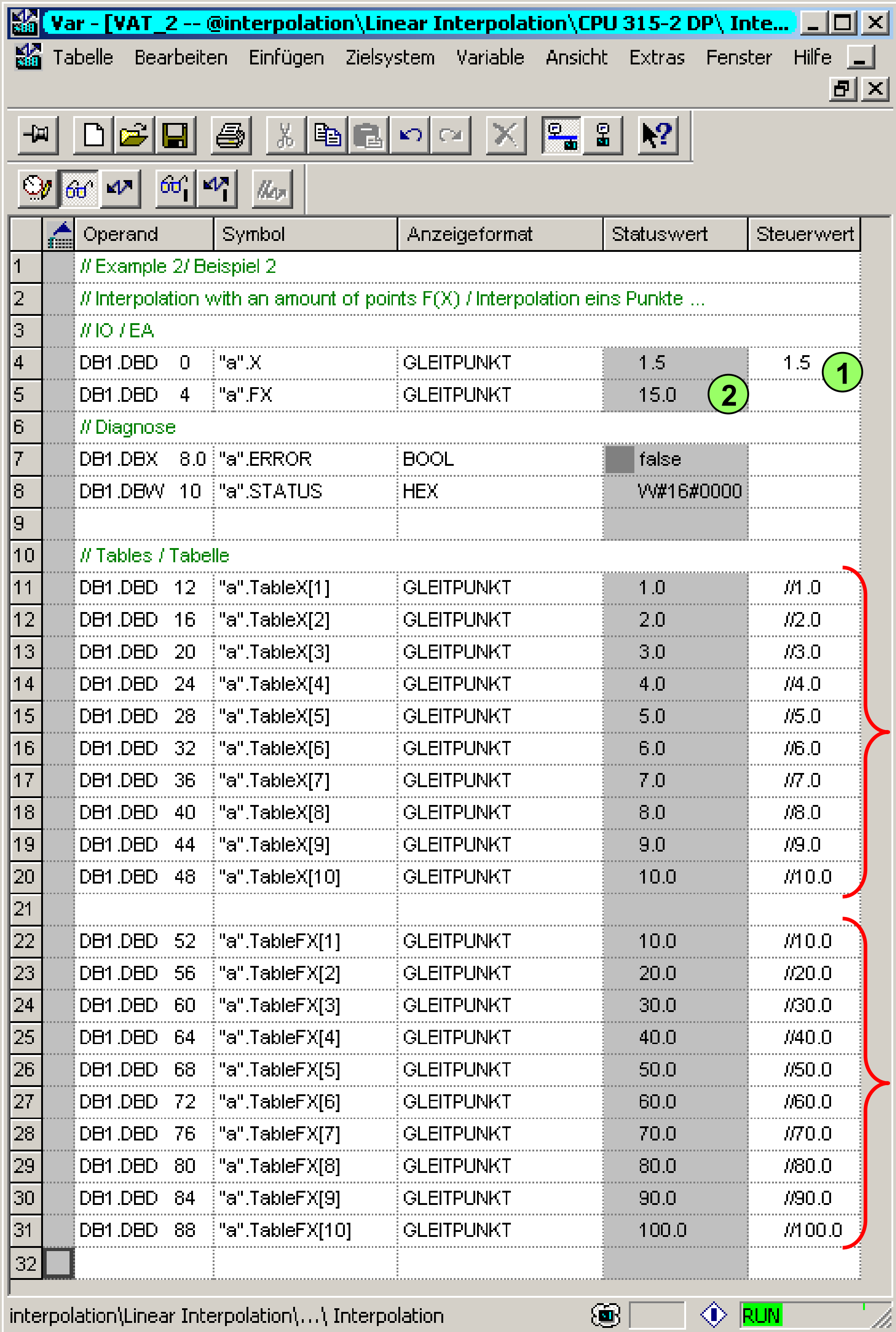
Example

In the project folder of “Interpol.zip”, you can find the variable table “VAT_2” to test the functionality of function “INTERP_1D”. The function is called cyclically in OB 1.After restart, the function values (interpolation points) are allocated in OB 100. To test the example, proceed as follows:

Table 2-25

Step	Action / Event
1.	✓ Load the complete station into the CPU or to the S7-PLCSIM and start the CPU. Result: OB 100 is called and initializes the function values (interpolation points) in the instance data block of the function.
2.	✓ Open the variable table “VAT_2” in online mode.
3.	✓ Enter, for instance, 1.5 as value in the variable table and transfer it to control. (1) Result: The function returns the interpolated value 15 as a result. (2)

Figure 2-11



	Operand	Symbol	Anzeigeformat	Statuswert	Steuerwert
1	// Example 2/ Beispiel 2				
2	// Interpolation with an amount of points F(X) / Interpolation eins Punkte ...				
3	// IO / EA				
4	DB1.DBD 0	"a".X	GLEITPUNKT	1.5	1.5
5	DB1.DBD 4	"a".FX	GLEITPUNKT	15.0	
6	// Diagnose				
7	DB1.DBX 8.0	"a".ERROR	BOOL	false	
8	DB1.DBW 10	"a".STATUS	HEX	W#16#0000	
9					
10	// Tables / Tabelle				
11	DB1.DBD 12	"a".TableX[1]	GLEITPUNKT	1.0	//1.0
12	DB1.DBD 16	"a".TableX[2]	GLEITPUNKT	2.0	//2.0
13	DB1.DBD 20	"a".TableX[3]	GLEITPUNKT	3.0	//3.0
14	DB1.DBD 24	"a".TableX[4]	GLEITPUNKT	4.0	//4.0
15	DB1.DBD 28	"a".TableX[5]	GLEITPUNKT	5.0	//5.0
16	DB1.DBD 32	"a".TableX[6]	GLEITPUNKT	6.0	//6.0
17	DB1.DBD 36	"a".TableX[7]	GLEITPUNKT	7.0	//7.0
18	DB1.DBD 40	"a".TableX[8]	GLEITPUNKT	8.0	//8.0
19	DB1.DBD 44	"a".TableX[9]	GLEITPUNKT	9.0	//9.0
20	DB1.DBD 48	"a".TableX[10]	GLEITPUNKT	10.0	//10.0
21					
22	DB1.DBD 52	"a".TableFX[1]	GLEITPUNKT	10.0	//10.0
23	DB1.DBD 56	"a".TableFX[2]	GLEITPUNKT	20.0	//20.0
24	DB1.DBD 60	"a".TableFX[3]	GLEITPUNKT	30.0	//30.0
25	DB1.DBD 64	"a".TableFX[4]	GLEITPUNKT	40.0	//40.0
26	DB1.DBD 68	"a".TableFX[5]	GLEITPUNKT	50.0	//50.0
27	DB1.DBD 72	"a".TableFX[6]	GLEITPUNKT	60.0	//60.0
28	DB1.DBD 76	"a".TableFX[7]	GLEITPUNKT	70.0	//70.0
29	DB1.DBD 80	"a".TableFX[8]	GLEITPUNKT	80.0	//80.0
30	DB1.DBD 84	"a".TableFX[9]	GLEITPUNKT	90.0	//90.0
31	DB1.DBD 88	"a".TableFX[10]	GLEITPUNKT	100.0	//100.0
32					

x-values (rows 11–20)

Function values F(x) (rows 22–31)

Technical data

Table 2-26

Block	Data
INTERP_1D FB1 Linear interpolation of a one-dimensional function with several interpolation points	Required local data: 26 bytes Load memory requirement: 728 bytes Main memory requirement : 616 bytes

Function “INTERP_2D” FB 2

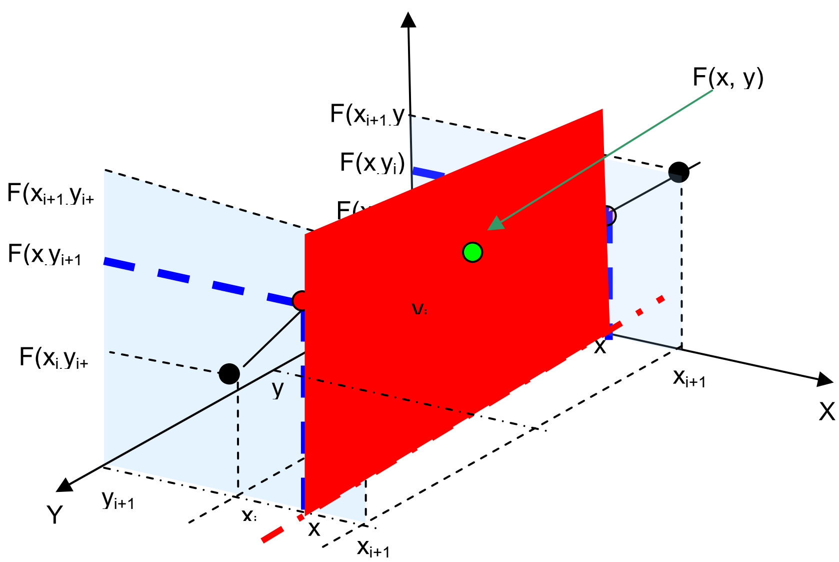
The function “INTERP_2D” FB 2 enables you to perform an interpolation of a two-dimensional function F(x, y). You approximate the function by means of several interpolation points. The interpolation points are saved to the instance data block of the function and must be manually written (see example). You can modify the number of possible interpolation points in the SCL source file “INTERP_2D”. The default value for each variable (x, y) is three interpolation points.

Note

- Each pair of values (x, y) has a corresponding function value F(x, y).
- You can modify the number of interpolation points by means of the constants “Nx” and “Ny” in the SCL source.
- If you change the number of possible interpolation points, you have to compile the SCL source anew.

The following figure illustrates the calculation of interpolated values:

Figure 2-12



Block parameters of function “INTERP_2D” FB 2

Table 2-27

Parameters	Declaration	Data type	Area	Description
X	IN	REAL	I, Q, M, D, L	Input value
Y	IN	REAL	I, Q, M, D, L	Input value
FXY	OUT	REAL	Q, M, D, L	Interpolated value (F(x, y))
ERROR	OUT	BOOL	Q, M, D, L	1: Error, 0: No error
STATUS	OUT	WORD	Q, M, D, L	Status: 0000: no error 7001: X<Xo or X>Xn or Y<Yo or Y>Yn 8191: Xn = Xo or Yn = Yo --> Division by zero

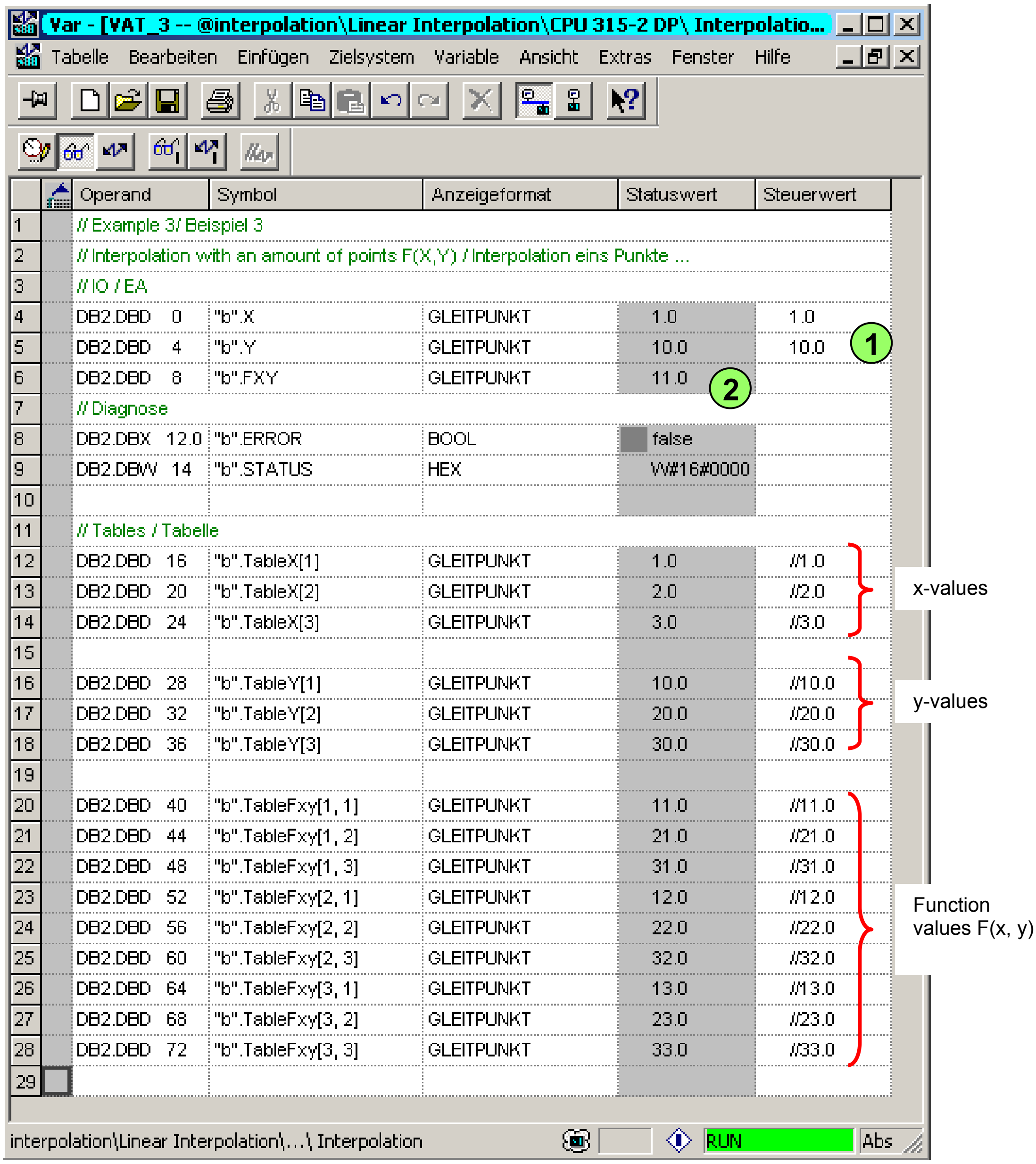
Example

In the project folder of “Interpol.zip”, you can find the variable table “VAT_3” to test the functionality of function “INTERP_2D”. The function is called cyclically in OB 1. After restart, the function values (interpolation points) are allocated in OB 100. To test the example, proceed as follows:

Table 2-28

Step	Action / Event
1.	✓ Load the complete station into the CPU or to the S7-PLCSIM and start the CPU. Result: OB 100 is called and initializes the function values (interpolation points) in the instance data block of the function.
2.	✓ Open the variable table “VAT_3” in online mode.
3.	✓ Enter, for instance, the value 1.0 and 10.0 in the variable table and transfer this to control. (1) Result: The function returns the interpolated value 11 as a result. (2)

Figure 2-13



Var - [VAT_3 -- @interpolation\Linear Interpolation\CPU 315-2 DP\ Interpolatio...

	Operand	Symbol	Anzeigeformat	Statuswert	Steuerwert
1	// Example 3/ Beispiel 3				
2	// Interpolation with an amount of points F(X,Y) / Interpolation eins Punkte ...				
3	// IO / EA				
4	DB2.DBD 0	"b".X	GLEITPUNKT	1.0	1.0
5	DB2.DBD 4	"b".Y	GLEITPUNKT	10.0	10.0
6	DB2.DBD 8	"b".FXY	GLEITPUNKT	11.0	
7	// Diagnose				
8	DB2.DBX 12.0	"b".ERROR	BOOL	false	
9	DB2.DBW 14	"b".STATUS	HEX	W#16#0000	
10					
11	// Tables / Tabelle				
12	DB2.DBD 16	"b".TableX[1]	GLEITPUNKT	1.0	//1.0
13	DB2.DBD 20	"b".TableX[2]	GLEITPUNKT	2.0	//2.0
14	DB2.DBD 24	"b".TableX[3]	GLEITPUNKT	3.0	//3.0
15					
16	DB2.DBD 28	"b".TableY[1]	GLEITPUNKT	10.0	//10.0
17	DB2.DBD 32	"b".TableY[2]	GLEITPUNKT	20.0	//20.0
18	DB2.DBD 36	"b".TableY[3]	GLEITPUNKT	30.0	//30.0
19					
20	DB2.DBD 40	"b".TableFxy[1, 1]	GLEITPUNKT	11.0	//11.0
21	DB2.DBD 44	"b".TableFxy[1, 2]	GLEITPUNKT	21.0	//21.0
22	DB2.DBD 48	"b".TableFxy[1, 3]	GLEITPUNKT	31.0	//31.0
23	DB2.DBD 52	"b".TableFxy[2, 1]	GLEITPUNKT	12.0	//12.0
24	DB2.DBD 56	"b".TableFxy[2, 2]	GLEITPUNKT	22.0	//22.0
25	DB2.DBD 60	"b".TableFxy[2, 3]	GLEITPUNKT	32.0	//32.0
26	DB2.DBD 64	"b".TableFxy[3, 1]	GLEITPUNKT	13.0	//13.0
27	DB2.DBD 68	"b".TableFxy[3, 2]	GLEITPUNKT	23.0	//23.0
28	DB2.DBD 72	"b".TableFxy[3, 3]	GLEITPUNKT	33.0	//33.0
29					

Technical data

Table 2-29

Block	Data
INTERP_2D FB2 Linear interpolation of a two-dimensional function with several interpolation points	Required local data: 54 bytes Load memory requirement: 1466 bytes Main memory requirement : 1308 bytes

Function “INTERP_3D” FB 3

The function “INTERP_3D” FB 3 enables you to perform an interpolation of a three-dimensional function F(x, y, z). You approximate the function by means of several interpolation points. The interpolation points are saved to the instance data block of the function and must be manually written (see example). You can modify the number of possible interpolation points in the SCL source file “INTERP_3D”. The default value for each variable (x, y, z) is three interpolation points.

Note

- Each triple of values (x, y, z) has a corresponding function value F(x, y, z).
- You can modify the number of interpolation points by means of the constants “Nx”, “Ny” and “Nz” in the SCL source.
- If you change the number of possible interpolation points, you have to compile the SCL source anew.

The calculation is analogous to the calculation in case of “INTERP_2D”, but with three variables x, y and z.

Block parameters of function “INTERP_3D” FB 3

Table 2-30

Parameters	Declaration	Data type	Area	Description
X	IN	REAL	I, Q, M, D, L	Input value
Y	IN	REAL	I, Q, M, D, L	Input value
Z	IN	REAL	I, Q, M, D, L	Input value
FXYZ	OUT	REAL	Q, M, D, L	Interpolated value (F(x, y, z))
ERROR	OUT	BOOL	Q, M, D, L	1: Error, 0: No error
STATUS	OUT	WORD	Q, M, D, L	Status: 0000: no error 7001: X<Xo or X>Xn or Y<Yo or Y>Yn or Z<Zo or Z>Zn 8191: Xn = Xo or Yn = Yo or Zn = Zo --> Division by zero

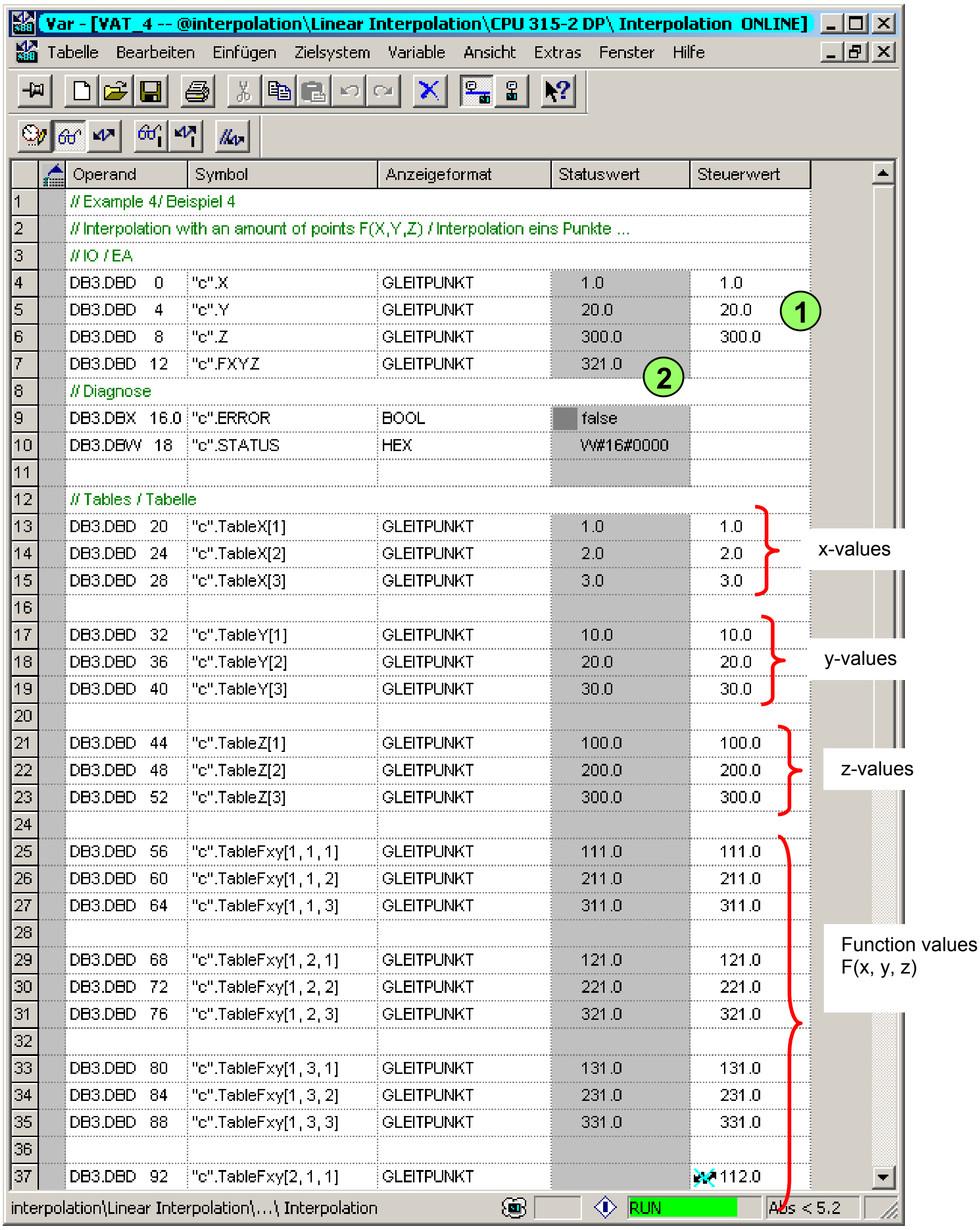
Example

In the project folder of “Interpol.zip”, you can find the variable table “VAT_4” to test the functionality of function “INTERP_3D”. The function is called cyclically in OB 1. After restart, the function values (interpolation points) are allocated in OB 100. To test the example, proceed as follows:

Table 2-31

Step	Action / Event
1.	✓ Load the complete station into the CPU or to the S7-PLCSIM and start the CPU. Result: OB 100 is called and initializes the function values (interpolation points) in the instance data block of the function.
2.	✓ Open the variable table "VAT_4" in online mode.
3.	✓ Enter, for instance, the values 1.0, 20.0 und 300.0 in the variable table and transfer this to control. (1) Result: The function returns the interpolated value 321 as a result. (2)

Figure 2-14



Technical data

Table 2-32

Block	Data
INTERP_3D FB3 Linear interpolation of a three-dimensional function with several interpolation points	Required local data: 98 bytes Load memory requirement: 2618 bytes Main memory requirement : 2404 bytes

3 Overview of the Download Files

In download file “29851674_Operationen_V10.zip”, you can find the following ZIP files for the respective function examples.

Table 3-1

No.	Data block	ZIP file
1.	Random number generator	Random.zip
2.	Determination of the parity of data elements	Parity.zip
3.	Determination of the active bit position in a 16-bit data word	bitpos_c.zip
4.	Edge detection in a 32-bit field	Monitor.zip
5.	Incrementing counter with 2,147,483,647 limit	Counter.zip
6.	Calculate the xth root of a REAL number	xroot.zip
7.	Calculation of statistical values in an automation system	spc_example.zip
8.	Matrix operations in SIMATIC systems	MatrixOp.zip
9.	Multidimensional interpolation	Interpol.zip

4 History

Table 4-1 History

Version	Date	Modifications
V1.0	2009-04-06	First version