Tool for Control Technology

applications
& TOOLS

Tool collection of conversion blocks for data type conversions

SIEMENS

Tool collection for conversion blocks

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 und 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/25629271

Table of Contents

1 Converter for floating point and fixed-point numbers

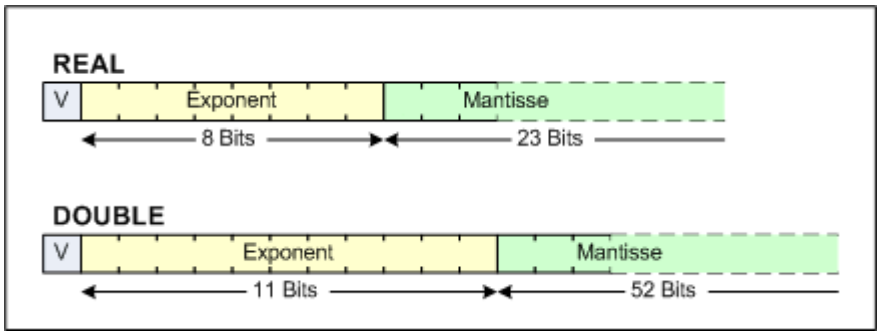
1.1 Conversion: 64 bit floating point number ⇔ 32 bit floating point number

Description

If SIMATIC controllers shall process or output 64 bit floating point numbers (DOUBLE), a prior format conversion to or from the 32 bit data type “REAL” becomes necessary. Working with DOUBLE in SIMATIC requires, that the calculation precision of the 32 bit floating point arithmetic is sufficient for the set task.

Both IEEE 754 floating point formats DOUBLE and REAL consist of sign bit, exponent and mantissa and are represented as follows:

Figure 1-1



The program example contains a STEP 7 library with the functions “32TO64” (FC21) and “64TO32” (FC22), which can be called in the organization block OB1. You can test the function with the added variable table.

Function “32TO64”

Function “32TO64” (FC21) transforms a 32 bit floating point number into a 64 bit floating point number. The REAL sign bit is adopted in the highest-order DOUBLE bit. The difference = 896 is added to the REAL exponent to receive the DOUBLE exponent. The REAL mantissa is written to the DOUBLE mantissa left-aligned, the non-assigned low-order bits are filled up with zeros.

Block parameters of the “32TO64” (FC21) function

Table 1-1

Parameter	Declaration	Data type	Range	Description
float	Input	REAL	I, M, D, L	REAL number to be converted into DOUBLE
double	Input	POINTER	M, D, L	Target address for the value in DOUBLE format. The target area must comprise 8 bytes
RET_VAL	Return	INT	Q, M, D, L	The value gives information on the format conversion "float" ⇨ "double": 0000_H: "float" is a normalized number number[1] "double" is a normalized number[2] (normal case). 0001_H: "float" = NULL[3] or a denormalized number[4] "double" = NULL[3]. 0002_H: "float" = ±∞[5] or no valid floating point number (NaN)[6] "double" = NULL[3].

Note

[1] normalized number: exponent = 1...254
[2] normalized number: exponent = 1...2046
[3] NULL: exponent = 0 and mantissa = 0
[4] denormalized number: exponent = 0 and mantissa > 0
[5] ±∞ exponent = 255, mantissa = 0
[6] NaN: exponent = 255, mantissa = > 0

Function “64TO32”

Function “64TO32” (FC22) transforms a 64 bit floating point number into a 32 bit floating point number. The DOUBLE sign bit is adopted in the highest-order REAL bit. From the DOUBLE exponent the difference = 896 is subtracted to receive the REAL exponent. From the DOUBLE mantissa the 29 low-order bits are cut off.

Block parameters of the 64TO32 (FC22) function

Table 1-2

Parameter	Declaration	Data type	Range	Description
double	Input	POINTER	M, D, L	Source address for the value in DOUBLE format. The source area must comprise 8 bytes
float	Output	REAL	Q, M, D, L	DOUBLE number transformed to REAL
RET_VAL	Return	INT	Q, M, D, L	The value gives information on the format conversion " double" ⇨ " float": 0000_H: "double" is a normalized number[1] "float" is a normalized number[2] (normal case) 0001_H: "double" = NULL[3] or a denormalized number[4] or too small [5] for the format conversion ⇨ "float" =NULL[3] 0002_H: "double" = too large for the format conversion[6] or = ±∞[7] or no valid floating point number (NaN)[8] ⇨ "real" = $\pm 3.402823 \cdot 10^{38}$ (maximum)

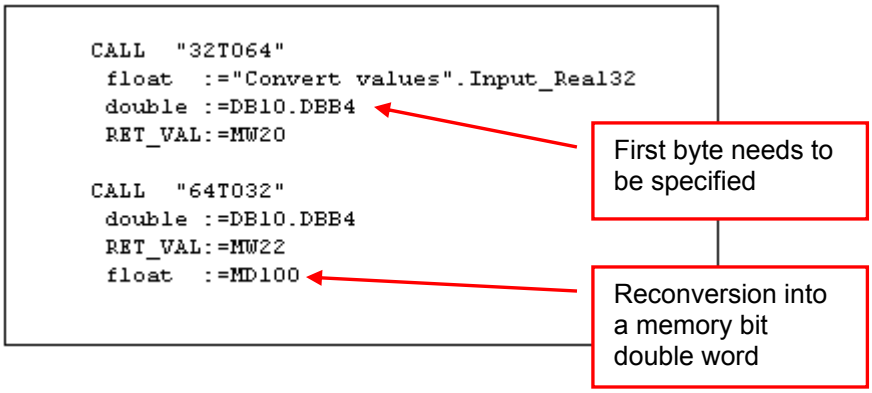
Note

[1] normalised number: exponent = 1...2046
[2] normalized number: exponent = 1...254
[3] NULL: exponent = 0 and mantissa = 0
[4] denormalized number: exponent = 0 and mantissa > 0
[5] too small: exponent < 879
[6] too large: exponent > 1150
[7] ±∞ exponent = 2047, mantissa = 0
[8] NaN: exponent = 2047, mantissa = > 0

Example

In the organization block OB1 the functions are called with the following parameters.

Figure 1-2



The next figure shows a VAT table with the conversion of the 32 bit floating point number (real number =11.25) into a 64 bit floating point number and the reconversion into the memory bit double-word MD100.

The 64 bit floating point number can be used to check the following formula.
An IEEEE 64 bit floating point number from the binary code is calculated as follows:

$$Z = (-1)^{VZ} * (1{,}0 + M/2^{52}) * 2^{E-1023}$$

Legend

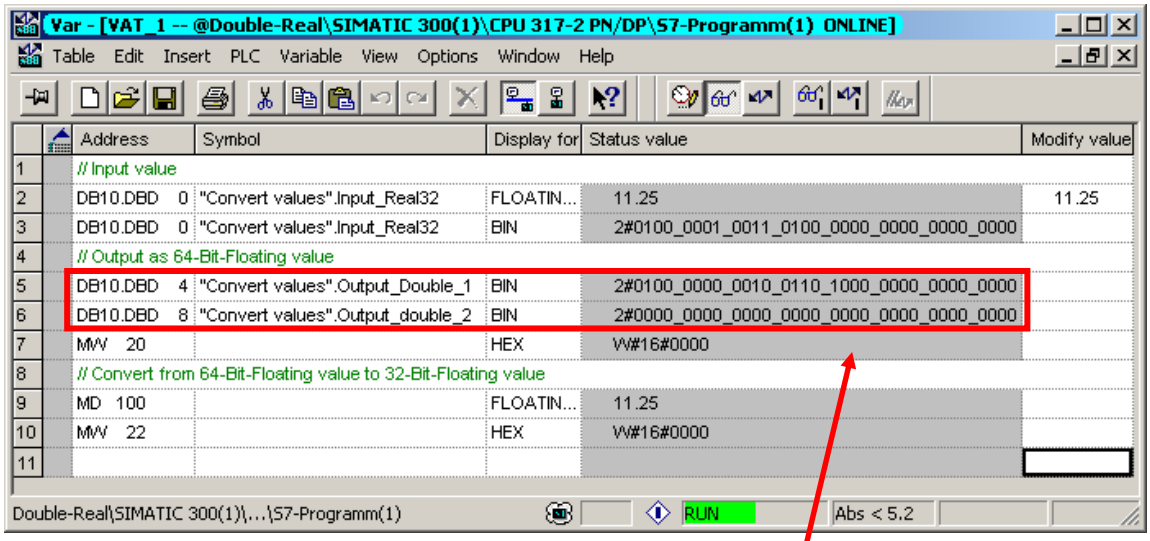
Z := decimal number

Vz := sign

E := exponent

M := mantissa

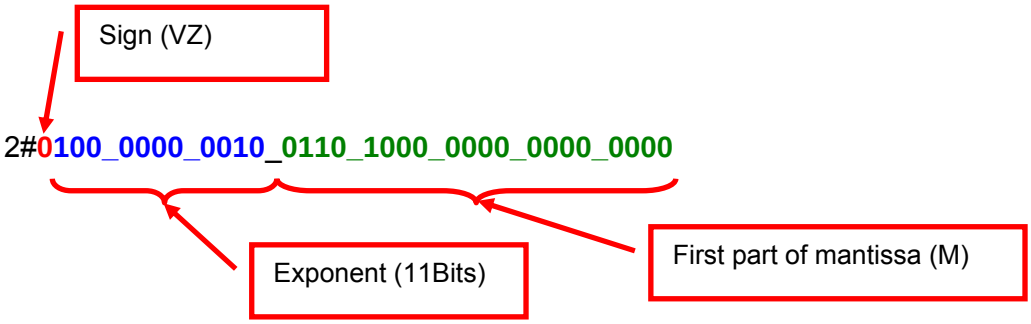
Figure 1-3



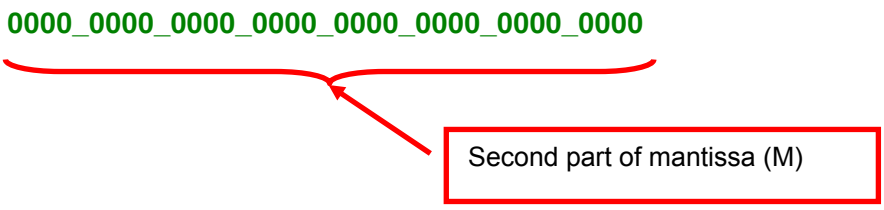
Binary code as 64 bit floating point number

To better understand the converted 64 bit floating point number (real number 11.25) is decoded and calculated using the binary sample from the VAT table.

Decoding of the data double-word: DB10.DBD4



Decoding of the data double-word: DB10.DBD8



Control calculation

The decimal numbers for the calculation formula are calculated using the decoded data words.

Sign: VZ = $0_{binary} = 0 \Rightarrow +$

Exponent: E = $2^{10} + 2^{1} = 1026$

Mantissa: M = $2^{50} + 2^{49} + 2^{47} = 1.829587^{15}$

Substituted into the formula:

$$Z = (-1)^{VZ} * (1.0 + M/2^{52}) * 2^{E-1023}$$

$$Z = (-1)^{0} * \left(1.0 + \frac{1.829587^{15}}{2^{52}}\right) * 2^{1026-1023} = 11.25$$

The floating point number “11.25” is displayed correctly in the 64 bit data format!

Technical data

Table 1-3

Block	Data
32TO64 (FC21) Converts REAL to DOUBLE	Required local data : 20 bytes Load memory requirements : 383 bytes Main memory requirement : 284 bytes
64TO32 (FC22) Converts DOUBLE to REAL	Required local data : 24 bytes Load memory requirements : 386 bytes Main memory requirement : 286 bytes

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

1.2 Converting floating point numbers to S5 or S7 format

Description

The floating point numbers in the S5 CPU have a different format than in an S7 CPU. You cannot take the real numbers directly from the S5 controller. You must first convert the numbers to S7 format. This naturally also applies in the reverse case if you wish to use S7 floating point values in your S5 CPU.

The program example contains a STEP 7 project with the “S7ToS5-Floating” (FC1) and “S5ToS7-Floating” (FC2) functions which are called in the organization block OB1. You can test the function with the added variable table.

Since the permitted value range at S7 (standardized) differs from that of S5, the following permitted data range results here:

- The value of the floating point number must not fall short of 1.175495e-38 (S7 limit).
- The value of the floating point number must not exceed 1.701412e+38 (S5 limit).

Function “S7ToS5-Floating”

In the “S7ToS5 floating” function the real numbers in the S7 format are converted to floating point numbers in the S5 format. At the input "S7_R_Number" please enter the S7 real number. The converted floating point number is output at the output parameter "S5_F_Number".

When exceeding the S5 limit the "Overflow" bit is set in the function. With this bit you check the validity of the transformed REAL number.

Block parameters of the “S7ToS5 floating” (FC1) function

Table 1-4

Parameter	Declaration	Data type	Range	Description
S7_R_Number	Input	REAL	ID, QD, MD, DBD, LD	Real number in S7 format
S5_F_Number	Output	DWORD	QD, MD, DBD, LD	Floating point number in S5 format
Overflow	Output	BOOL	Q, M, D, L	S7 real number too large

"S7ToS5 floating" function

In the "S7ToS5 floating" function the floating point numbers in the S5 format are converted to real numbers in the S7 format. At the input "S5_F_Number" you enter the S5 floating point number. The then converted real number is output at the output parameter "S5_F_Number".

When falling short of the S7 limit the "Underflow" bit in the function is set. With this bit you check the validity of the transformed REAL number.

Block parameters of the "S5ToS7-Floating" (FC2) function

Table 1-5

Parameter	Declaration	Data type	Range	Description
S5_F_Number	Input	DWORD	ID, QD, MD, DBD, LD	Floating point number in S5 format
S7_R_Number	Output	REAL	QD, MD, DBD, LD	Real number in S7 format
Underflow	Output	BOOL	Q, M, D, L	S5 floating point number too small

Technical data

Table 1-6

Block	Data
"S7ToS5-Floating" (FC1) converts floating point numbers in S7 format to S5 format	Required local data : 10 bytes Load memory requirements : 264 bytes Main memory requirement : 198 bytes
"S5ToS7-Floating" (FC2) converts floating point numbers in S5 format to S7 format	Required local data : 10 bytes Load memory requirements : 320 bytes Main memory requirement : 246 bytes

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

1.3 Converting a variable from data type DINT to INT

Description

Since in programming language STL no explicit conversion command from DINT to INT exists, the following function is available to you in the programming languages LAD/STL/FBD. Function “DINT_TO_INT” converts a 32 bit integer into a 16 bit integer and indicates in the Boolean variable whether the conversion was successful.

Function “DINT_TO_INT”

The function first checks whether the input value of data type “DINT” lies between -32768 and 32767. If this is the case, the lowest-order word of the input value is loaded and transferred to the output value of data type “INT”. Furthermore the variable “OK” (conversion successful) is set to “1”.
If the input value is outside the valid range, the value “0” is transferred on the output value “INTEGER” and variable “OK” is additionally set to “0”.

Block parameters of the “DINT_TO_INT” (FC25) function

Table 1-7

Parameter	Declaration	Data type	Range	Description
DOUBLE	Input	DINT	ID, QD, MD, DBD, LD	Transferring a variable from data type DINT
INTEGER	Output	INT	Q, M, D, L	Conversion result form data type “INT”
OK	Output	BOOL	Q, M, D, L	Conversion OK = '1' (transferred area lies between -32768 and 32767)

Technical data

Table 1-8

Block	Data
“DINT_TO_INT” (FC25) Converts a 32 bit integer into a 16 bit integer	Required local data : 0 bytes Load memory requirements : 182 bytes Main memory requirement : 112 bytes

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

2 Converter for String Data Types

2.1 Converting an integer to ASCII characters

Description

An integer is converted to ASCII code without using data type “STRING”. Function “INT_TO_ASCII” divides an integer number into individual ASCII characters.

Function “INT_TO_ASCII”

The function converts the 16 bit integer (decimal number) temporarily into a seven digit binary coded decimal number (BCD number) and adds an offset of 30hex (corresponds to digit “0” in ASCII) for displaying the individual numbers (ones-tens-hundreds ...) in ASCII format.
The sign is also evaluated and entered as ASCII sign ’+’ or ’-’ in the target data area a the first position.
The target area in the data block must be declared as “ARRAY of CHAR” with a length of 6.

Block parameters of the “INT_TO_ASCII” (FC1) function

Table 2-1

Parameter	Declaration	Data type	Range	Description
VALUE	Input	INT	I, Q, M, D, L	Integer to be divided
DB_NO	Input	BLOCK_DB	DB	Target block in which the ASCII character shall be saved
START_ADD	Input	INT	I, Q, M, D, L	Target address in the data block for the ASCII characters

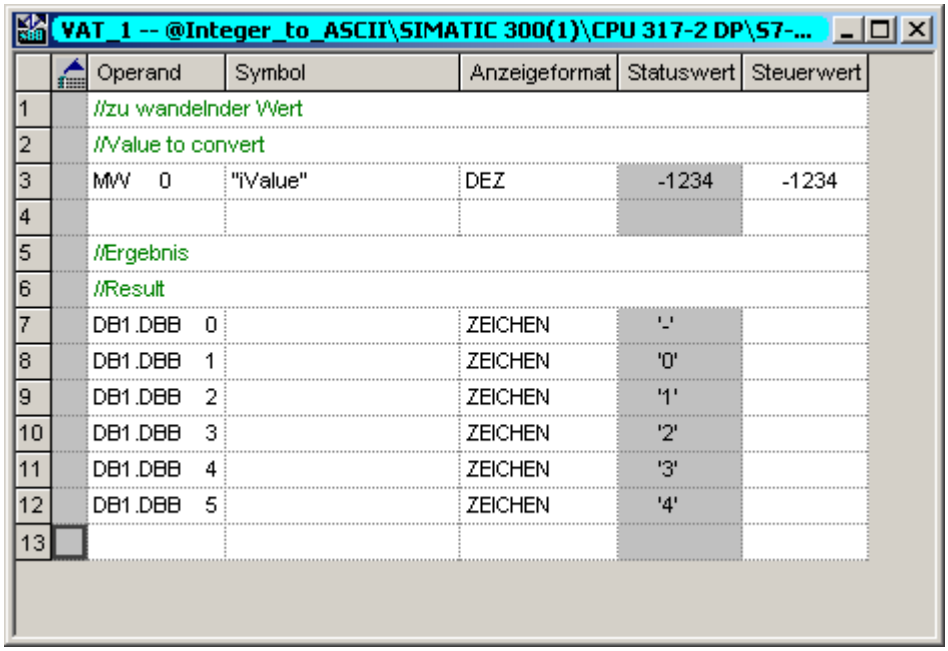
Example

Function FC1 is called in the organization block OB1 with the following parameters:

```
CALL  "INT_TO_ASCII"
 VALUE    :="iValue"
 DB_NO    :=DB1
 START_ADD:=0
```

In the added S7 program you find data bock DB1 with an array of type “Character” with the length 6. From target address “0” on the individual digits of the integer, which is transferred to variable “iValue” (MW0) in variable table “VAT_1”, are entered as ASCII character.

Figure 2-1 Number to be converted and the respective result in the data block



	Operand	Symbol	Anzeigeformat	Statuswert	Steuerwert
1	//zu wandelnder Wert				
2	//Value to convert				
3	MW 0	"iValue"	DEZ	-1234	-1234
4					
5	//Ergebnis				
6	//Result				
7	DB1.DBB 0		ZEICHEN	'-'	
8	DB1.DBB 1		ZEICHEN	'0'	
9	DB1.DBB 2		ZEICHEN	'1'	
10	DB1.DBB 3		ZEICHEN	'2'	
11	DB1.DBB 4		ZEICHEN	'3'	
12	DB1.DBB 5		ZEICHEN	'4'	
13					

Technical data

Table 2-2

Block	Data
“INT_TO_ASCII” (FC1) Converts a 32 bit integer into ASCII characters	Required local data : 10 bytes Load memory requirements : 258 bytes Main memory requirement : 186 bytes

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

2.2 Converting string variables to ASCII text format

Description

A field with string variables is completely converted to ASCII text format. The first two bytes (length specification of the string) are also converted to ASCII text characters.

This is required for certain applications, i.e. if a STRING variable must be output directly via an ASCII or FTP protocol on a printer or at a PC terminal.

The function therefore replaces the first two bytes with the ASCII code for line break (0D Hex) and line feed (0A Hex). Thereafter the string variable only contains ASCII text characters.

In a variable of data type “STRING” an ASCII character chain can be saved at most with 254 characters. The length in bytes results from the number of characters in the character chain "n" and 2 bytes for the header.

The following table shows the order of bytes when specifying the data type “STRING[4]” with the output value 'AB'. (Representation of the values in Hex format).

Table 2-3 Structure of a string in Hex format

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
4	2	41	42	0	0

The bytes have the following meaning:

- Byte 0: maximum length of the character chain
- Byte 1: actual length of the character chain
- Byte 2: ASCII value for 'A'
- Byte 3: ASCII value for 'B'
- Byte 4: not assigned
- Byte 5: not assigned

Note The string functions (e.g. from the IEC library) can no longer be applied with the converted variables!

Function “RPGEN”

In the organization block the function “RPGEN” OB 1 is called via an edge memory bit (one-shot trigger). The function replaces the header information in byte 0 and byte 1 with the ASCII code for “Carrige Return” (0D HEX) and “Line Feed” (0A HEX).

The character chains are saved equal length in the data block DB 1 in a field of string variables with equal length.
Data block DB 1, created as an example, contains 35 character chains, each character chain has a length of 76 characters.

After the function was called once, all header information is replaced with “0D” and “0A”. A new call of function “RPGEN” causes no further changes in the data block.

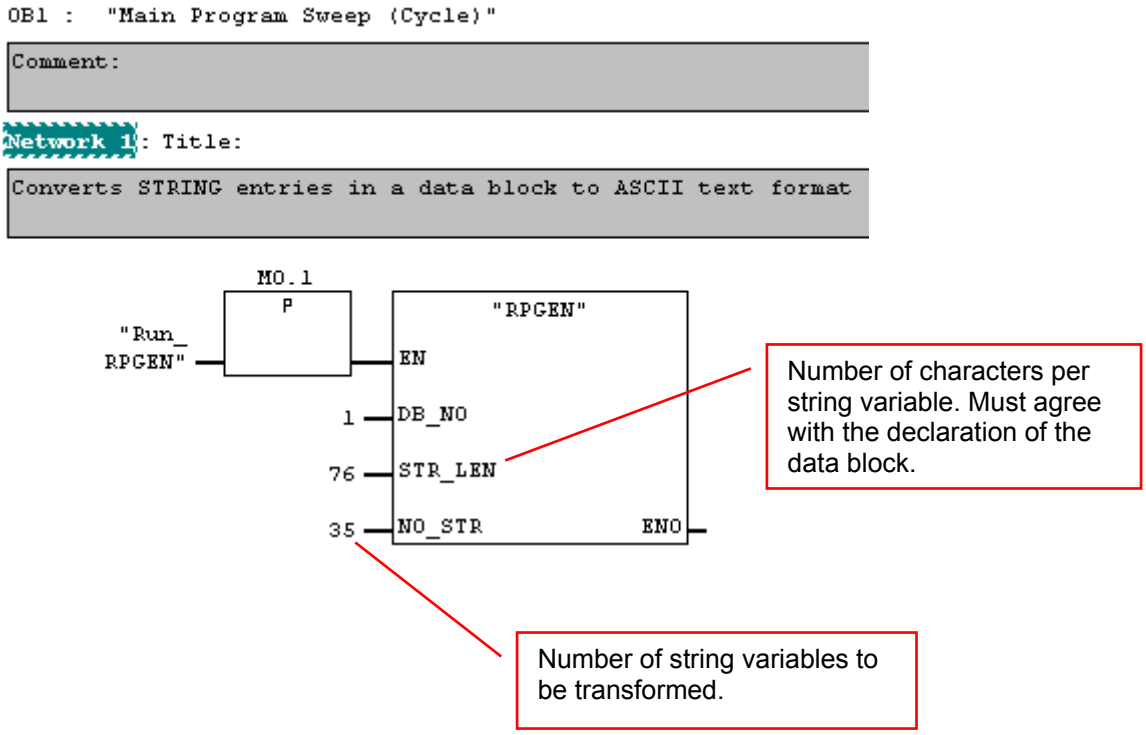
Block parameters of the “RPGEN” (FC1) function

Table 2-4

Parameter	Declaration	Data type	Range	Description
DB_NO	Input	INT	I, Q, M, D, L	Number of the data block which contains the character chains to be converted (“DB1” in the example)
STR_LEN	Input	INT	I, Q, M, D, L	Length of the character chain (as defined for the string variables, 76 characters the example)
NO_STR	Input	INT	I, Q, M, D, L	Number of character chains (as defined in the field, “35” in the example)

Example

Figure 2-2



Note To test this function a VAT table is available to you.

Technical data

Table 2-5

Block	Data
"RPGEN" (FC1) Converts a string variable to ASCII text format (incl. the length information).	Required local data : 12 bytes Load memory requirements : 210 bytes Main memory requirement : 142 bytes

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

3 Converter for Date/Time and Time Operations

3.1 Converting from Time to DINT and vice versa

Description

To convert the data type TIME into DINT values and vice versa the functions “TIME_TO_DINT” and “DINT_TO_TIME” are available to you. The following four tables show the conversion formula to convert input value (in milliseconds) to days, hours, minutes, seconds and milliseconds (Tage, Stunden, Minuten Sekunden, and Millisekunden).

Table 3-1 Conversions for the days

Variable	Description	Conversion
Tage	Days	Tage = TIME_Wert / 86400000 ms 1 day = 86400000 ms
TageinStd	Days in hours	TageinStd = Tage * 24 h
TageinMin	Days in minutes	TageinMin = Tage * 1440 min.
TageinSek	Days in seconds	TageinSek = Tage * 86400 sec.
TageinMSek	Days in milliseconds	TageinMSek = Tage * 86400000 ms

Table 3-2 Conversions for the hours

Variable	Description	Conversion
Stunden	Hours from TIME_Wert	Stunden = (TIME_Wert / 3600000) - TageinStd
StdinMin	Hours in minutes	StdinMin = Stunden * 60 min.
StdinSek	Hours in seconds	StdinSek = Stunden * 3600 sec.
StdinMSek	Hours in milliseconds	StdinMSek = Stunden * 3600000 ms

Table 3-3 Conversions for the minutes

Variable	Description	Conversion
Minuten	Minutes from TIME_Wert	Minuten = (TIME_Wert / 60000) - TageinMin - StdinMin
MininSek	Minute in seconds	MininSek = Minuten * 60 sec.
MininMSek	Minute in milliseconds	MininMSek = Minuten * 60000 ms

Table 3-4 Conversions for seconds and milliseconds

Variable	Description	Conversion
Sekunden	Seconds from TIME_Wert	Sekunden = (TIME_Wert / 1000) - TageinSek - StdinSek – MininSek
SekinMSek	Minutes in milliseconds	SekinMSek = Sekunden * 1000 ms
Millisekunden	Milliseconds from TIME_Wert	Millisekunden = TIME_Wert - TageinMSek - StdinMSek - SekinMSek

"TIME_TO_ DINT" function

The "TIME_TO_DINT" converts a variable of data type "TIME" to data type "DINT", where the individual values (Tage, Stunden, Minuten, Sekunden, Millisekunden) are provided respectively as individual variable.

The content of the variable of the "TIME" type is interpreted as "Milisekunden" and filed as 32 bit fixed-point number with sign. To convert data type "TIME" (number of milliseconds) to the individual "DINT" variables (Tage, Stunden, Minuten, Sekunden, Millisekunden) the calculation formulae listed above are used.

Block parameters of the "TIME_TO_DINT" (FC12) function

Table 3-5

Parameter	Declaration	Data type	Range	Description
TIME_WERT	Input	TIME (IEC time)	ID, QD, MD, DBD, LD	Time to be converted. Example: T#12d13h14m25s100ms
TAGE	Output	DINT	ID, MD, DBD, LD	Value in days
STUNDEN	Output	DINT	QD, MD, DBD, LD	Value in hours
MINUTEN	Output	DINT	QD, MD, DBD, LD	Value in minutes
SEKUNDEN	Output	DINT	QD, MD, DBD, LD	Value in seconds
MILLISEKUNDEN	Output	DINT	QD, MD, DBD, LD	Value in milliseconds

Function "DINT_TO_TIME"

Function "DINT_TO_TIME" converts the variables with the content of days, hours, minutes, seconds and milliseconds (Tage, Stunden, Minuten, Sekunden and Millisekunden) back to a variable with data type "Time". The value is first converted to milliseconds, then added and subsequently

transferred to output parameter “TIME_WERT”. The result is a 32 bit fixed-point number with sign, which in the data view is displayed as follows with display format “Time”: T#12d13h14m25s100ms.

Block parameters of the “DINT_TO_TIME” (FC11) function

Table 3-6

Parameter	Declaration	Data type	Range	Description
TAGE	Input	DINT	ID, QD, MD, DBD, LD	Value in days
STUNDEN	Input	DINT	ID, QD, MD, DBD, LD	Value in hours
MINUTEN	Input	DINT	ID, QD, MD, DBD, LD	Value in minutes
SEKUNDEN	Input	DINT	ID, QD, MD, DBD, LD	Value in seconds
MILLISEKUNDEN	Input	DINT	ID, QD, MD, DBD, LD	Value in milliseconds
TIME_WERT	Output	TIME (IEC time)	QD, MD, DBD, LD	Converted number with data type “TIME“

Example

The example contains a STEP 7 project with the functions listed above. The call is made in organization block OB1 via FC10. To test the functions a VAT table is available to you.

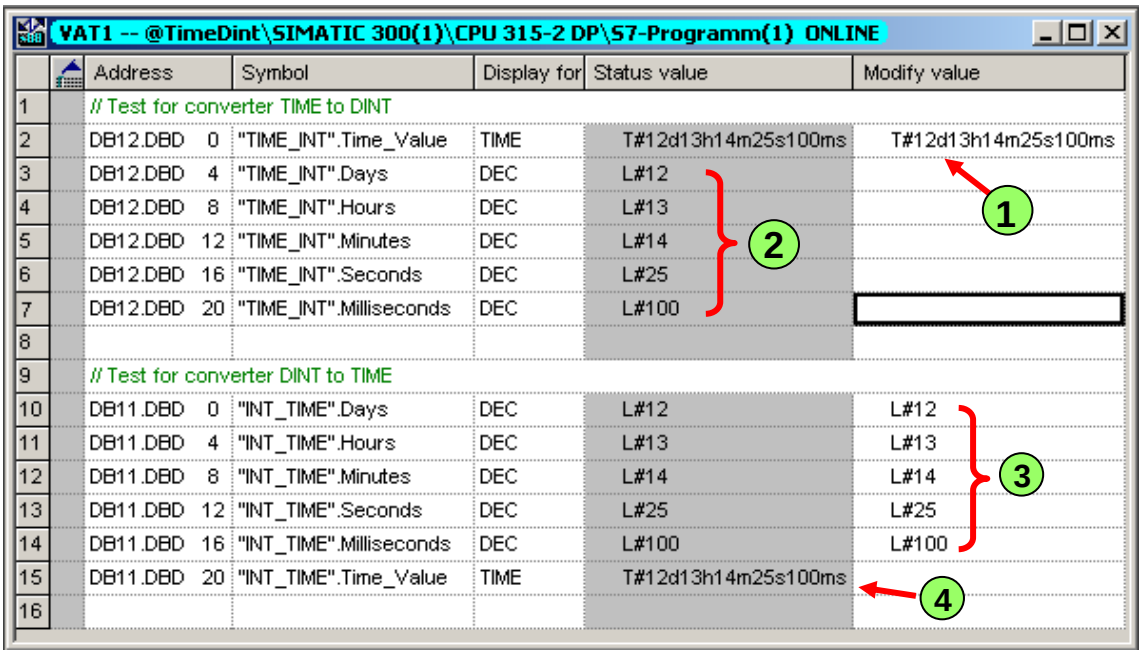
The following table lists the most important steps for testing the function.

Table 3-7

Step	Action / Event
1.	✓ Load the complete station into the CPU or the S7-PLCSIM
2.	✓ Open the VAT table “VAT1” in “Monitor variable” mode
3.	✓ Enter an input value in “TIME” format and activate the control value for converting “TIME” to “DINT”. See point 1 The result of the conversion can be seen in point 2
4.	✓ Enter the value for Tage, Stunden, Minuten, Sekunden and Millisekunden, and activate the control value for the conversion of “DINT” to “TIME”. See point 3

Step	Action / Event
	The result of the conversion can be seen in point 4

Figure 3-1



Technical data

Table 3-8

Block	Data
"TIME_TO_DINT" (FC12) Converts a variable with data type "TIME" into five variables for days, hours, minutes, seconds and milliseconds (Tage, Stunden, Minuten, Sekunden and Millisekunden)	Required local data : 40 bytes Load memory requirements : 408 bytes Main memory requirement : 318 bytes
"DINT_TO_TIME" (FC11) Conversion of the above mentioned function. From five DINT variables one variable of data type "TIME" is formed	Required local data : 16 bytes Load memory requirements : 204 bytes Main memory requirement : 132 bytes

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

3.2 Converting DATE_AND_TIME to STRING and vice versa

Description

Function “DT_TO_STRING” converts a DATE_AND_TIME variable to a STRING with a date and time character sequence.

Function “STRING_TO_DT” converts the STRING with the data and time character sequence back to a variable with the composite data type “DATE_AND_TIME”.

The example contains a STEP 7 project with the functions listed above. In the example the temporary local data variable “OB1_DATE_TIME” is read in the organization block OB1 and converted to a STRING variable. Subsequently there is a reconversion. To test the functions a VAT table is available to you.

To represent a DATE_AND_TIME variable as data and time character sequence (STRING), the individual bytes of the DATE_AND_TIME variable are read in, converted to individual numbers in ASCII code, and then copied into the string.

Date and time are saved with data type “DATE_AND_TIME” in 8 bytes (BCD format). The following table shows the content and the validity range of this data type.

Table 3-9

Byte	Content	Range
0	Year "YY"	1990...1999 → 90...99 2000...2089 → 00...89
1	Month "MM"	01...12
2	Day "DD"	1...31
3	Hour "hh"	0...23
4	Minute "mm"	0...59
5	Second "ss"	0...59
6	First two digits of the milliseconds "cc." (both highest-order numbers)	00...99
7 (4MSB)	Last position of milliseconds "..c"	0...9
7 (4LSB)	Weekday	1...7 (1 = Sunday)

Legend:

MSB: Most Significant Bit
LSB: Least Significant Bit

Function “DT_TO_STRING”

Function “DT_TO_STRING” converts the DATE_AND_TIME variable to a STRING variable of the following format.
DD/MM/JJ hh:mm:ss.ccc

The separators in the data representation are set via the block parameter “w” (freely selectable).

The generated string has a length of 23 bytes. During the declaration of the STRING variable the declared length is automatically stored in byte 0 as well as the current length of the string in byte 1. From byte 2 on the actual STRING elements are saved.

If the STRING variable lies in the local data area, byte 0 (declaration length of the string) must be initialized prior to calling the function.

Block parameters of function “DT_TO_STRING” (FC90)

Table 3-10

Parameter	Declaration	Data type	Range	Description
X_DATE_TIME	Input	DATE_AND_TIME	P#	DT variable
W	Input	CHAR	’ ’	Separator between date
Y_DT_STRING	Output	STRING	P#	STRING variable with 21 elements

Function “STRING_TO_DT”

Function “STRING_TO_DT” converts the STRING character chain with format **DD/MM/JJ hh:mm:ss.ccc** into a variable with data type “DATE_AND_TIME”.

If the STRING variable lies in the local data area, byte 0 (declaration length of the string) must be initialized prior to calling the function.

Block parameters of function “STRING_TO_DT” (FC91)

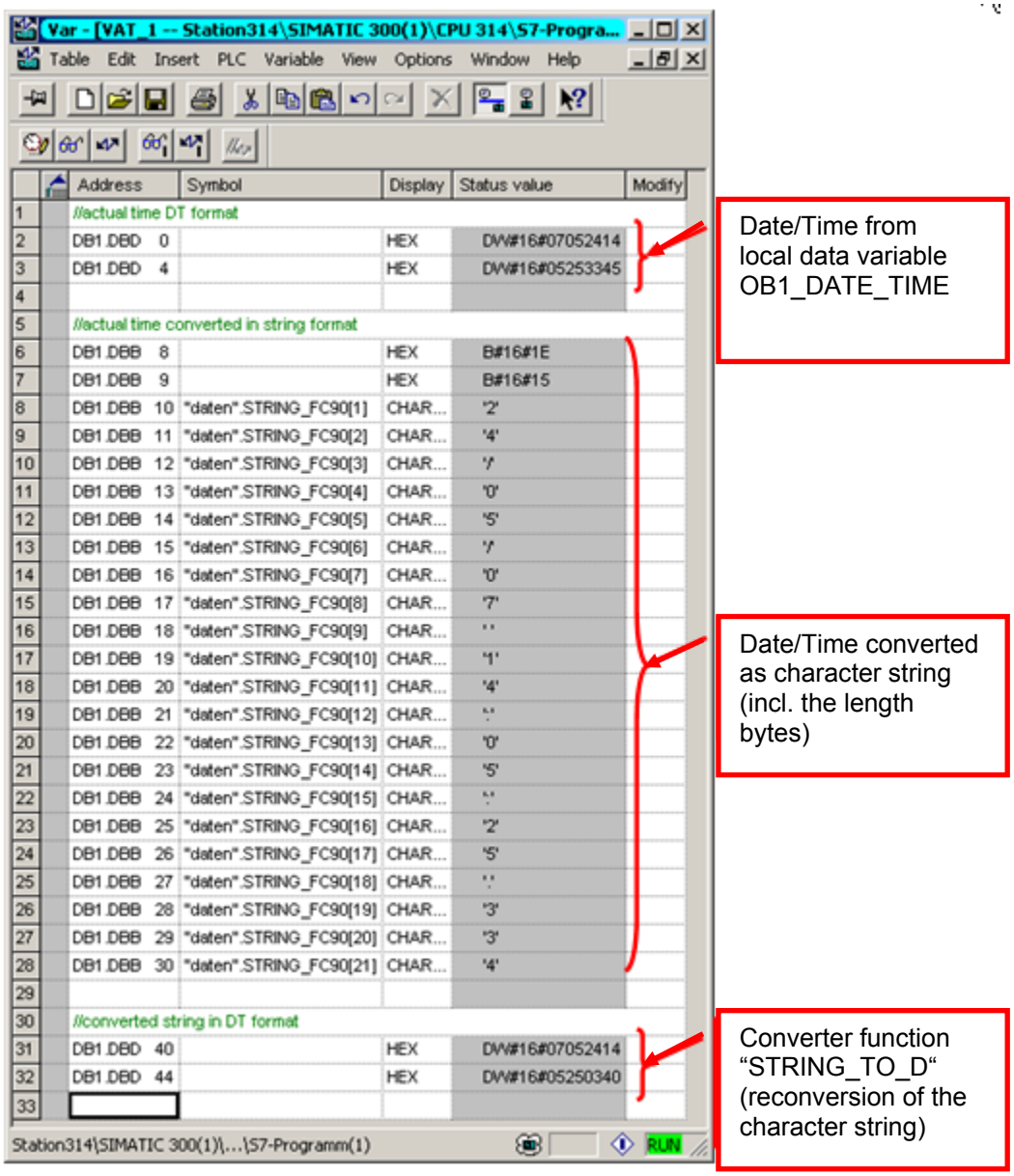
Table 3-11

Parameter	Declaration	Data type	Range	Description
X_DT_STRING	Input	STRING	P#	STRING variable with 21 elements
Y_DATE_TIME	Output	DATE_AND_TIME	P#	DT variable

Example

The following VAT table shows the results of the conversion for date and time from 24/05/2007 14:05:25.334 .

Figure 3-2



Note Details of the week day are not considered in the STRING and during reconverting.

Technical data

Table 3-12

Block	Data
"TIME_TO_DINT" (FC90) Converts a variable with data type "DATE_AND_TIME" back to a STRING variable	Required local data : 16 bytes Load memory requirements : 450 bytes Main memory requirement : 370 bytes
"DINT_TO_TIME" (FC91) Reconverting Converts the STRING variable to a variable with data type "DATE_AND_TIME"	Required local data : 16 bytes Load memory requirements : 412 bytes Main memory requirement : 334 bytes

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

4 Converter for various Conversion Functions

4.1 Converting numbers in binary code ⇔ gray code

Description

The gray code is a symmetrical way of generating a code where the neighboring numbers differ in only one bit. It serves as coding method for robust transmission of digital parameters via analog signal paths (e.g. temperature sensors, travel measurement).
Six functions are available for converting binary code to gray code and vice versa.

- from binary code to gray code
  - FC103 for 8 bit gray code (Byte)
  - FC104 for 16 bit gray code (Word)
  - FC105 for 32 bit gray code (DWord)

- from gray code to binary code
  - FC100 8 bit gray code to 8 bit binary (Byte)
  - FC101 16 bit gray code to 16 bit binary (Word)
  - FC102 32 bit gray code to 32 bit binary (DWord)

This example contains a library with the above listed functions, a VAT table and an organization block OB1 for testing the functions FC103 and FC100.

4 bit gray code representation

The following table shows the binary values and the gray code of the decimal numbers 0 to 15. The gray code is a different representation form of the binary code. It is based on the notion that two neighboring gray numbers must not differ in more than one bit (i.e. 0 or 1).

Table 4-1

Decimal number	Binary code	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100

Decimal number	Binary code	Gray code
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Functions “binary code to gray code”

The following functions convert binary coded numbers to gray coded numbers. The functions only differ in code width (8, 16 or 32 bit) which they convert.

- Function “BINAER_to_GRAY_BYTE” (FC103)
- Function “BINAER_to_GRAY_ WORD” (FC104)
- Function “BINAER_to_GRAY_ DWORD” (FC105)

The following conversion formula was used for the conversion:

$$G(n) = B(n)\ XOR\ B(n+1)$$

Legend:

G = bit of the gray code
B = bit of the binary code
n = number of the bit

Block parameter of the functions FC103, FC104 and FC105

Table 4-2

Parameter	Declaration	Data type	Range	Description
BINAERCODE	INPUT	FC103: BYTE FC104: WORD FC105: DWORD	I, Q, M, D, L	Binary code coded number
GRAYCODE	OUTPUT	FC103: BYTE FC104: WORD FC105: DWORD	Q, M, D, L	Gray code coded number

Functions “gray code to binary code”

The following functions convert gray coded numbers to binary coded numbers. The only difference between the functions is that the variables have a different format.

- Function “GRAY_to_BINAER_ BYTE” (FC100)
- Function “GRAY_to_BINAER_ WORD” (FC101)
- Function “GRAY_to_BINAER_ DWORD” (FC102)

The following conversion formula was used for the conversion:
The highest-order bit is calculated first:

$$B(n) = G(n)\ XOR\ 0$$

The following bits are calculated as follows:

$$B(n-1) = G(n-1)\ XOR\ B(n)$$

$$B(n-2) = G(n-2)\ XOR\ B(n-1)$$

$$B(n-3) = G(n-3)\ XOR\ B(n-2)$$

etc.

The formula for calculating the bits from (n-1) to 1 is:

$$B(n-x) = G(n-x)\ XOR\ B(n-(x-1)\ )$$

Legend:

x = number from 1 to (n-1)
G = bit of the gray code
B = bit of the binary code
n = number of the bit

Block parameter of the functions FC100, FC101 and FC102

Table 4-3

Parameter	Declaration	Data type	Range	Description
GRAYCODE	INPUT	FC100: BYTE FC101: WORD FC102: DWORD	I, Q, M, D, L	Gray code coded number
BINAERCODE	OUTPUT	FC100: BYTE FC101: WORD FC102: DWORD	Q, M, D, L	Binary code coded number

Example

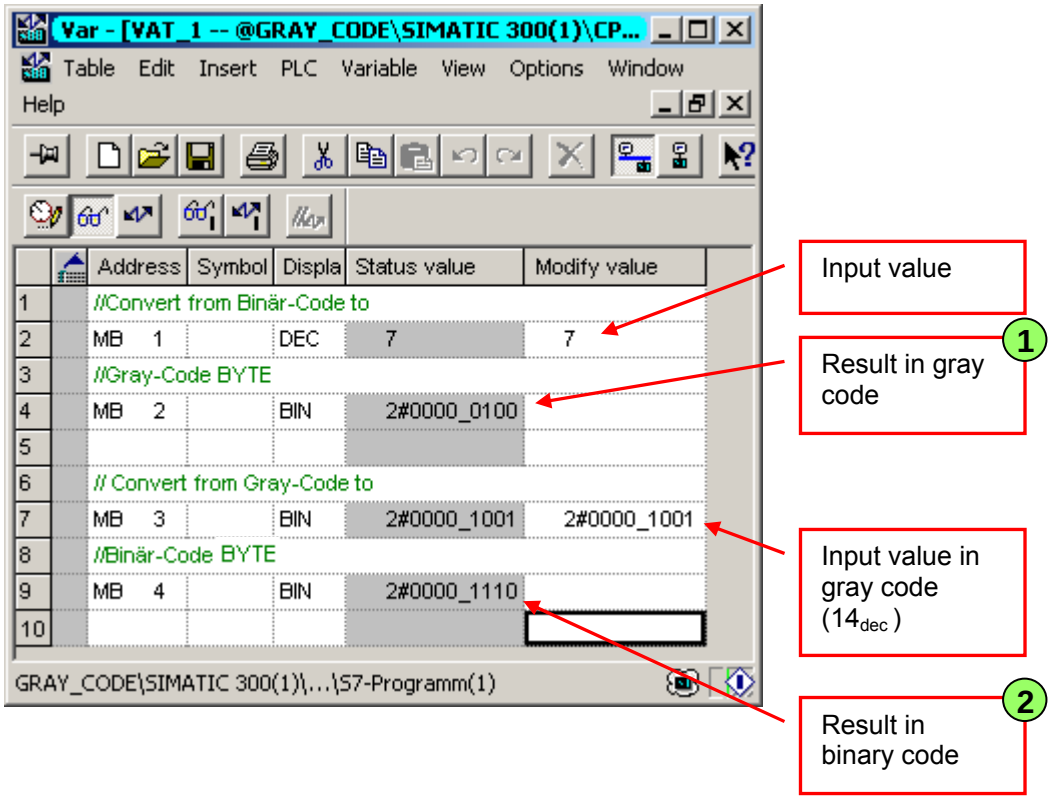
In the following table the most important steps for testing the functions are listed. In organization block OB1 the following functions are called in network 1.

- Function “BINAER_to_GRAY_BYTE” (FC103)
- Function “GRAY_to_BINAER_ BYTE” (FC100)

Table 4-4

Step	Action / Event
1.	✓ Generate a new STEP 7 project and copy all blocks and objects from the library to the program folder.
2.	✓ Load the complete station into the CPU or the S7-PLCSIM
3.	✓ Open the VAT table “VAT1” in “Monitor variable” mode
4.	✓ Enter the input value in “7” and activate the control value for converting binary to gray code. The result of the conversion can be seen in point (1)
5.	✓ Enter the input value in 2#000_1001 (decimal 14) and activate the control value for converting gray to binary code. The result of the conversion can be seen in point (2)

Figure 4-1



Technical data

Table 4-5

Block	Data
Functions: binary code to gray code “BINAER_to_GRAY_BYTE” (FC103) “BINAER_to_GRAY_WORD” (FC104) “BINAER_to_GRAY_DWORD” (FC105)	Required local data : 0 bytes Load memory requirements : 108 bytes Main memory requirement : 54 bytes
Function “GRAY_to_BINAER_ BYTE” (FC100)	Required local data : 4 bytes Load memory requirements : 174 bytes Main memory requirement : 112 bytes
Function “GRAY_to_BINAER_ WORD” (FC101)	Required local data : 6 bytes Load memory requirements : 174 bytes Main memory requirement : 112 bytes
Function “GRAY_to_BINAER_ DWORD” (FC102)	Required local data : 10 bytes Load memory requirements : 174 bytes Main memory requirement : 112 bytes

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

4.2 Temperature conversion from degree Celsius ⇔ degree Fahrenheit or from degree Celsius ⇔ degree Kelvin

Description

Four functions are available for converting temperature values into different units.

- Converting degree Celsius to degree Fahrenheit
- Converting degree Fahrenheit to degree Celsius
- Converting degree Kelvin to degree Celsius
- Converting degree Celsius to degree Kelvin

The example contains a STEP 7 project with the functions listed above. The call is made in organization block OB1. To test the functions a VAT table is available to you.

Function “Degree Celsius to degree Fahrenheit” (Tmp_C_F)

To define the zero point of the Fahrenheit temperature scale a cold mixture of water, ice and ammonium chloride was used. The freezing point of the water for this scale lies at 32 degree Fahrenheit, the boiling point of the water at 212 degree Fahrenheit. Between freezing and boiling point of the water there are 180 degrees, while on the Celsius temperature scale this temperature difference is divided into 100 parts.

The following conversion formula results:

Degree Celsius to degree Fahrenheit: $F = \frac{9}{5} * (C + 32)$

Block parameters of function “Tmp_C_F” (FC1)

Table 4-6

Parameter	Declaration	Data type	Range	Description
Tmp_Celsius	INPUT	REAL	ID, QD, MD, DBD, LD	Temperature in degrees Celsius as real number
Tmp_Fahrenheit	OUTPUT	REAL	QD, MD, DBD, LD	Temperature in degrees Fahrenheit as real number

Function “Degree Fahrenheit to degree Celsius” (Tmp_F_C)

The following conversion formula is used for converting degree Fahrenheit to degree Celsius:

Degree Fahrenheit to degree Celsius: $C = \frac{5}{9} * (F - 32)$

Block parameters of function “Tmp_F_C” (FC2)

Table 4-7

Parameter	Declaration	Data type	Range	Description
Tmp_Fahrenheit	INPUT	REAL	ID, QD, MD, DBD, LD	Temperature in degrees Fahrenheit as real number
Tmp_Celsius	OUTPUT	REAL	QD, MD, DBD, LD	Temperature in degrees Celsius as real number

Function “Kelvin to degree Celsius” (Tmp_K_C)

Kelvin is the SI unit for temperature. The temperature units Kelvin and Celsius differ in the definition of the zero point. 0 K refers to the absolute zero point of the temperature at minus 273,15° C, at which no thermal movement takes place even on atom level. The Celsius scale refers to the melting point of ice as zero point.

Daraus ergibt sich folgende Umrechnungsformel:

$$Tmp\_Celsius\,[°C] = Tmp\_Kelvin\,[K] - 273{,}15$$

Block parameters of function “Tmp_K_C” (FC3)

Table 4-8

Parameter	Declaration	Data type	Range	Description
Tmp_Kelvin	INPUT	REAL	ID, QD, MD, DBD, LD	Temperature in Kelvin as real number
Tmp_Celsius	OUTPUT	REAL	QD, MD, DBD, LD	Temperature in degrees Celsius as real number

Function “Degree Celsius to Kelvin” (Tmp_C_K)

The following conversion formula is used for reconverting degree Celsius to Kelvin:

$$Tmp\_Kelvin\left[K\right] = Tmp\_Celsius\left[°C\right] + 273{,}15$$

Block parameters of function “Tmp_C_K” (FC4)

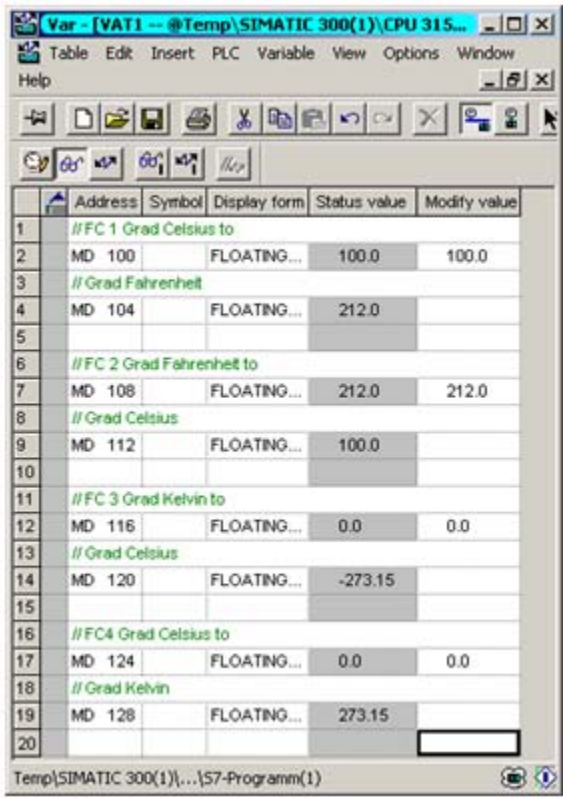
Table 4-9

Parameter	Declaration	Data type	Range	Description
Tmp_Celsius	INPUT	REAL	ID, QD, MD, DBD, LD	Temperature in degrees Celsius as real number
Tmp_Kelvin	OUTPUT	REAL	QD, MD, DBD, LD	Temperature in Kelvin as real number

Example

With the added VAT table you test the various conversion functions.

Figure 4-2



Technical data

Table 4-10

Block	Data
Function “Tmp_C_F” Converting degree Celsius to degree Fahrenheit	Required local data : 0 bytes Load memory requirements : 124 bytes Main memory requirement : 70 bytes
Function “Tmp_F_C” Converting degree Fahrenheit to degree Celsius	Required local data : 0 bytes Load memory requirements : 124 bytes Main memory requirement : 70 bytes
Function “Tmp_K_C” Converting degree Kelvin to degree Celsius	Required local data : 0 bytes Load memory requirements : 108 bytes Main memory requirement : 54 bytes
Function “Tmp_C_K” Converting degree Celsius to degree Kelvin	Required local data : 0 bytes Load memory requirements : 108 bytes Main memory requirement : 54 bytes

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

5 Overview of the Download Files

In download file “25629271_Converter_V10.ZIP” you find the below ZIP files for the respective converter blocks.

Table 5-1

No.	Converter	ZIP file
1.	Conversion: 64 bit floating point number ⟺ 32 bit floating point number	double_real.zip
2.	Converting floating point numbers to S5 or S7 format	S5_S7-Floating-Converter.zip
3.	Converting a variable from data type “DINT” to “INT”	DINT_to_INT.zip
4.	Converting an integer to ASCII characters	INTEGER_to_ASCII.zip
5.	Converting a string variable to ASCII text format including two bytes for length specification	STRING_to_ASCII-Text.zip
6.	Converting a variable of data type “TIME” into five DINT variables with the values for days, hours, minutes, seconds and milliseconds (Tage, Stunden, Minuten, Sekunden and Millisekunden) Reconverting DINT variables to data type “TIME”. From five DINT variables one variable of data type “TIME” is formed	TIME_to_DINT.zip
7.	Converts a variable of data type “DATE_AND_TIME” back to a STRING variable Reconverting a STRING variable to data type “DATE_AND_TIME”	DT_to_STRING.zip
8.	Converting numbers in binary code ⟺ gray code	Gray_code.zip
9.	Temperature conversion in various units (degree Celsius, degree Fahrenheit, Kelvin)	Temperatur.zip

6 History

Table 6-1 History

Version	Date	Modification
V1.0	02.02.2009	First issue