Software Description

Advant Controller 31
Intelligent Decentralized Automation System

Function Block Library
40/50 Series

TP S40

IN
Q

T
T
Contents

The Base_S40 function block library 3
Considerable changes in ABB library 907 AC 1131 compared to CE library 907 PC 331 ......................... 3
Special characteristics in the ABB library .................................................. 4
Functions .................................................................................. 4
Function blocks ............................................................................ 4
Special characteristics of individual blocks ........................................... 4

Overview of blocks arranged according to their call names 5

ADDITION DOUBLE WORD ADDD S40 ........................................ 8
OFF DELAY ASV S40 ............................................................ 10
SELECTION GATE, WORD AWT S40 ........................................ 12
BINARY SELECTION GATE AWTB S40 ..................................... 13
BCD TO DUAL CONVERSION, WORD BCDDUAL S40 ................ 14
LIMITER, WORD BEG S40 .......................................................... 17
BINARY VALUE CHANGE ANNUNCIATOR BMELD(8, 127) S40 ....... 19
ANALOG CHANNEL CONFIGURATION CONFIO(1, 8) S40 ............ 22
COPYING MEMORY AREAS COPY S40 ........................................ 25
CONFIGURE AC31 MODULES CS31CO S40 ............................ 27
ACKNOWLEDGE AC31 ERRORS CS31QU S40 ............................. 35
UP COUNTER CTU S40 ................................................................. 37
HIGH SPEED COUNTER CTUH S40 ........................................... 39
READ DIRECT INPUTS DIN S40 .................................................. 42
DIVISION DOUBLE WORD DOUT S40 ........................................... 46
OUTPUT OF ASCII CHARACTERS DRUCK S40 ............................. 48
DUAL TO BCD CONVERSION, WORD DUALBCD S40 .............. 52
AND COMBINATION, DOUBLE WORD DWAND S40 ..................... 54
OR COMBINATION, DOUBLE WORD DWOR S40 .......................... 55
DOUBLE WORD TO WORD CONVERSION DWW S40 .............. 56
XOR COMBINATION, DOUBLE WORD DXOR S40 ......................... 57
RECEPTION OF ASCII CHARACTERS EMAS S40 ......................... 58
ON DELAY ESV S40 ................................................................. 62
FUNCTION GENERATOR FKG(2..256) S40 .............................. 64
FALLING EDGE DETECTION I_MINUS S40 .................................. 66
RISING EDGE DETECTION I_PLUS S40 ...................................... 68
READ BINARY VARIABLE, INDEXED IDLB S40 .............................. 70
READ WORD VARIABLE, INDEXED IDLM S40 .............................. 72
WRITE BINARY VARIABLE, INDEXED IDSB S40 ............................ 74
WRITE WORD VARIABLE, INDEXED IDSM S40 .............................. 76
LIST ALLOCATOR LIZU(8..256) S40 .......................................... 78
MONOSTABLE ELEMENT »ABORT« MOA S40 ................................. 80
MONOSTABLE ELEMENT »ABORT« MOAT S40 ................................. 82
OPERATION MODE MODBUS MASTER MODMAST/B/W S40 ........... 84
MONOSTABLE ELEMENT »CONSTANT« MOK S40 ............................. 92
MULTIPLICATION BY 2 TO THE POWER OF N, WORD MUL2N S40 ........ 94
MULTIPLICATION DOUBLE WORD MULD S40 .................................. 96
MULTIPLICATION WITH DIVISION MULDI S40 .............................. 98
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEGATION WORD</td>
<td>100</td>
</tr>
<tr>
<td>PULSES GENERATOR</td>
<td>101</td>
</tr>
<tr>
<td>PACK BINARY VARIABLES IN WORD</td>
<td>103</td>
</tr>
<tr>
<td>PULSE DURATION MODULATOR</td>
<td>105</td>
</tr>
<tr>
<td>PROPORTIONAL INTEGRAL CONTROLLER</td>
<td>107</td>
</tr>
<tr>
<td>PIDT1 CONTROLLER</td>
<td>112</td>
</tr>
<tr>
<td>RESET MEMORY , DOMINATING</td>
<td>118</td>
</tr>
<tr>
<td>SERIAL LINE INITIALIZATION</td>
<td>119</td>
</tr>
<tr>
<td>SQUARE ROOT, DOUBLEWORD</td>
<td>122</td>
</tr>
<tr>
<td>SQUARE ROOT, WORD</td>
<td>124</td>
</tr>
<tr>
<td>SET MEMORY, DOMINATING</td>
<td>126</td>
</tr>
<tr>
<td>SUBTRACTION DOUBLE WORD</td>
<td>127</td>
</tr>
<tr>
<td>TIME WORD CONVERSION</td>
<td>129</td>
</tr>
<tr>
<td>OFF DELAY</td>
<td>131</td>
</tr>
<tr>
<td>ON DELAY</td>
<td>133</td>
</tr>
<tr>
<td>MONOSTABLE ELEMENT « CONSTANT »</td>
<td>135</td>
</tr>
<tr>
<td>DISPLAY AND SET CLOCK</td>
<td>137</td>
</tr>
<tr>
<td>UNPACKING A WORD INTO BINARY VARIABLES</td>
<td>142</td>
</tr>
<tr>
<td>UP-DOWN COUNTER</td>
<td>144</td>
</tr>
<tr>
<td>INTERRUPTION TASK VALIDATION</td>
<td>146</td>
</tr>
<tr>
<td>AND COMBINATION, WORD</td>
<td>147</td>
</tr>
<tr>
<td>WORD TIME CONVERSION</td>
<td>148</td>
</tr>
<tr>
<td>WORD TO DOUBLE WORD CONVERSION</td>
<td>150</td>
</tr>
<tr>
<td>READ WORD WITH ENABLING</td>
<td>151</td>
</tr>
<tr>
<td>OR COMBINATION, WORD</td>
<td>153</td>
</tr>
<tr>
<td>XOR COMBINATION, WORD</td>
<td>154</td>
</tr>
</tbody>
</table>

**Glossary**

155

**Index**

157
The Base_S40 function block library

Considerable changes in ABB library 907 AC 1131 compared to CE library 907 PC 331

All blocks and functions are divided into groups of operators, functions and function blocks according to the IEC norm. (The respective definition can be found in the programming software manual.)

- All operators listed in the table »Overview of blocks« are part of the programming software.

- The standard library contains all blocks described in the programming manual.
- The ABB library for PLC series 40 and series 50 contains all ABB specific blocks (functions and function blocks, no operators).

Note: The blocks defined in the ABB library cannot be used in simulation mode.

In order to use the ABB specific blocks the ABB library (Base_S40_Vxx) must be included in the library window.
Special characteristics in the ABB library

In the ABB library all manufacturer functions and function blocks are integrated. The assignment of the block types is listed in the table «Overview of blocks».

The functionality of the blocks was identically transformed in accordance to the known blocks defined in 07 KR 51 - EBS. However, some special characteristics have to be observed when calling and selecting the blocks.

Functions

The FBD blocks are identically constructed compared to the blocks available in the previous programming software. Some input and output names have been changed as not all characters are allowed. In all programming languages the order of the inputs and outputs is identical. The library window displays the assignment of the blocks (order and variable type).

Function blocks

A main characteristic of the function blocks is that an instance has to be defined when calling them. In this case it has to be observed that instance names may not be defined several times.

Example:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VRZ1</td>
<td>VRZ</td>
</tr>
<tr>
<td>VRZ2</td>
<td>VRZ</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The instance name can be defined without any restrictions. The type is preset and identical to the function block name.

Special characteristics of individual blocks

1) Negation of inputs and outputs:

In the 907 AC 1131 the negation of inputs and outputs is only possible for bit operands (variable types: BOOL, BYTE, WORD, DWORD). The known operands word flag %MW (MW) and double word flag %MD (MD) are of the variable type INT and DINT.

2) Duplication of inputs:

The operators (ADD, DIV, MUL, SUB) can be used and duplicated as often as required.

3) Duplication of inputs and outputs:

Blocks: e.g. PACK, UNPACK

In the FBD editor the number of duplications is not permitted and is fixed to 4, 8 or 16. Input n is assigned accordingly and the block is displayed.

4) Insert Inline ABB Instruction List with Inline Pragma:

ABB Instruction List may be add in a project as a Program POU.

The ABB Instruction list must be inserted into Pragma in a IL Program POU.

The syntax is:

```
LD 0 necessary, if POU include only Inline Pragma

{S40Inline
.
.
ABB ASCII-Instruction List
.
.
}
```

Blocks: DRUCK, EMAS

The DRUCK and EMAS block are only available under this format.

Caution:
No syntax control is done inside Pragma !!!

5) Special solutions:

Blocks: LIZU8, LIZU16, LIZU32, LIZU64, LIZU256

The number assigned to LIZU represents the number of direct constants in the list. This number is a direct constant value (#8, #16, #32, #64, #256)
Blocks: FKG2, FKG4, FKG16, FKG32, FKG256

The number assigned to FKG represents the number of interpolation points. This number is a direct constant value (#2, #4, #16, #32, #256)

Blocks: BMELD8, BMELD16, BMELD32, BMELD64, BMELD127

The number assigned to BMELD represents the number of input values. This number is a direct constant value (#8, #16, #32, #64, #127)

Overview of blocks arranged according to their call names

Character description:

FBmV ... Function block with historical values
FBoV ... Function block without historical values
F ... Function

<table>
<thead>
<tr>
<th>CE name</th>
<th>Type</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDD</td>
<td>FBoV</td>
<td>Addition double word</td>
<td>8</td>
</tr>
<tr>
<td>ASV</td>
<td>FBmV</td>
<td>Off delay</td>
<td>10</td>
</tr>
<tr>
<td>AWT</td>
<td>FBoV</td>
<td>Selection gate word</td>
<td>12</td>
</tr>
<tr>
<td>AWTB</td>
<td>FBoV</td>
<td>Binary selection gate</td>
<td>13</td>
</tr>
<tr>
<td>BCDDUAL</td>
<td>F</td>
<td>BCD → DUAL conversion word</td>
<td>14</td>
</tr>
<tr>
<td>BEG</td>
<td>F</td>
<td>Limiter word</td>
<td>17</td>
</tr>
<tr>
<td>BMELD8</td>
<td>FBmV</td>
<td>Binary value change annunciator with a maximum of 8 comparison values, binary</td>
<td>19</td>
</tr>
<tr>
<td>BMELD16</td>
<td>FBmV</td>
<td>Binary value change annunciator with a maximum of 16 comparison values, binary</td>
<td>19</td>
</tr>
<tr>
<td>BMELD32</td>
<td>FBmV</td>
<td>Binary value change annunciator with a maximum of 32 comparison values, binary</td>
<td>19</td>
</tr>
<tr>
<td>BMELD64</td>
<td>FBmV</td>
<td>Binary value change annunciator with a maximum of 64 comparison values, binary</td>
<td>19</td>
</tr>
<tr>
<td>BMELD127</td>
<td>FBmV</td>
<td>Binary value change annunciator with a maximum of 127 comparison values, binary</td>
<td>19</td>
</tr>
<tr>
<td>CONFIO1</td>
<td>FBmV</td>
<td>1 Analog Channel Configuration</td>
<td>22</td>
</tr>
<tr>
<td>CONFIO4</td>
<td>FBmV</td>
<td>4 Analog Channels Configuration</td>
<td>22</td>
</tr>
<tr>
<td>CONFIO8</td>
<td>FBmV</td>
<td>8 Analog Channels Configuration</td>
<td>22</td>
</tr>
<tr>
<td>COPY</td>
<td>FBoV</td>
<td>Copying memory areas</td>
<td>25</td>
</tr>
<tr>
<td>CS31CO</td>
<td>FBmV</td>
<td>Configure AC31 modules</td>
<td>27</td>
</tr>
<tr>
<td>CS31QU</td>
<td>FBoV</td>
<td>Acknowledge AC31 errors</td>
<td>35</td>
</tr>
<tr>
<td>CTU</td>
<td>FBmV</td>
<td>Up counter</td>
<td>37</td>
</tr>
<tr>
<td>CTUH</td>
<td>FBmV</td>
<td>High speed counter</td>
<td>39</td>
</tr>
<tr>
<td>DIN</td>
<td>FBoV</td>
<td>Read direct inputs</td>
<td>42</td>
</tr>
<tr>
<td>DIVD</td>
<td>FBoV</td>
<td>Division double word</td>
<td>44</td>
</tr>
<tr>
<td>DOUT</td>
<td>FBoV</td>
<td>Write direct outputs</td>
<td>46</td>
</tr>
</tbody>
</table>
### OVERVIEW OF BLOCKS ARRANGED ACCORDING TO THEIR CALL NAMES

<table>
<thead>
<tr>
<th>CE name</th>
<th>Type</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRUCK</td>
<td>FBmV</td>
<td>Output of ASCII characters</td>
<td>48</td>
</tr>
<tr>
<td>DUALBCD</td>
<td>F</td>
<td>DUAL → BCD conversion word</td>
<td>52</td>
</tr>
<tr>
<td>DWAND</td>
<td>F</td>
<td>AND combination, double word</td>
<td>54</td>
</tr>
<tr>
<td>DWW</td>
<td>FBoV</td>
<td>Double word to word conversion</td>
<td>56</td>
</tr>
<tr>
<td>DWOR</td>
<td>F</td>
<td>OR combination, double word</td>
<td>55</td>
</tr>
<tr>
<td>DXOR</td>
<td>F</td>
<td>XOR combination, double word</td>
<td>57</td>
</tr>
<tr>
<td>EMAS</td>
<td>FBmV</td>
<td>Reception of ASCII characters</td>
<td>58</td>
</tr>
<tr>
<td>ESV</td>
<td>FBmV</td>
<td>On delay</td>
<td>62</td>
</tr>
<tr>
<td>FKG2</td>
<td>FBoV</td>
<td>Function generator with a maximum of 2 nodes</td>
<td>64</td>
</tr>
<tr>
<td>FKG4</td>
<td>FBoV</td>
<td>Function generator with a maximum of 4 nodes</td>
<td>64</td>
</tr>
<tr>
<td>FKG16</td>
<td>FBoV</td>
<td>Function generator with a maximum of 16 nodes</td>
<td>64</td>
</tr>
<tr>
<td>FKG32</td>
<td>FBoV</td>
<td>Function generator with a maximum of 32 nodes</td>
<td>64</td>
</tr>
<tr>
<td>FKG64</td>
<td>FBoV</td>
<td>Function generator with a maximum of 64 nodes</td>
<td>64</td>
</tr>
<tr>
<td>FKG256</td>
<td>FBoV</td>
<td>Function generator with a maximum of 256 nodes</td>
<td>64</td>
</tr>
<tr>
<td>I_MINUS</td>
<td>FBoV</td>
<td>Falling edge detection</td>
<td>66</td>
</tr>
<tr>
<td>I_PLUS</td>
<td>FBoV</td>
<td>Rising edge detection</td>
<td>68</td>
</tr>
<tr>
<td>IDLB</td>
<td>FBoV</td>
<td>Read binary variable, indexed</td>
<td>70</td>
</tr>
<tr>
<td>IDLm</td>
<td>FBoV</td>
<td>Read word variable, indexed</td>
<td>72</td>
</tr>
<tr>
<td>IDSB</td>
<td>FBoV</td>
<td>Write binary variable, indexed</td>
<td>74</td>
</tr>
<tr>
<td>IDSm</td>
<td>FBoV</td>
<td>Write word variable, indexed</td>
<td>76</td>
</tr>
<tr>
<td>LIZU8</td>
<td>FBoV</td>
<td>List allocator with a maximum of 8 word inputs</td>
<td>78</td>
</tr>
<tr>
<td>LIZU16</td>
<td>FBoV</td>
<td>List allocator with a maximum of 16 word inputs</td>
<td>78</td>
</tr>
<tr>
<td>LIZU32</td>
<td>FBoV</td>
<td>List allocator with a maximum of 32 word inputs</td>
<td>78</td>
</tr>
<tr>
<td>LIZU64</td>
<td>FBoV</td>
<td>List allocator with a maximum of 64 word inputs</td>
<td>78</td>
</tr>
<tr>
<td>LIZU256</td>
<td>FBoV</td>
<td>List allocator with a maximum of 256 word inputs</td>
<td>78</td>
</tr>
<tr>
<td>MOA</td>
<td>FBmV</td>
<td>Monostable element »abort«</td>
<td>80</td>
</tr>
<tr>
<td>MOAT</td>
<td>FBmV</td>
<td>Monostable element »abort«</td>
<td>82</td>
</tr>
<tr>
<td>MODUSB</td>
<td>FBmV</td>
<td>MODBUS master for COM1 and BOOL data</td>
<td>84</td>
</tr>
<tr>
<td>MODBUSW</td>
<td>FBmV</td>
<td>MODBUS master for COM1 and INT data</td>
<td>84</td>
</tr>
<tr>
<td>MODMASTB</td>
<td>FBmV</td>
<td>MODBUS master with interface identifier and BOOL data</td>
<td>84</td>
</tr>
<tr>
<td>MODMASTW</td>
<td>FBmV</td>
<td>MODBUS master with interface identifier and INT data</td>
<td>84</td>
</tr>
<tr>
<td>MOK</td>
<td>FBmV</td>
<td>Monostable element »constant«</td>
<td>92</td>
</tr>
<tr>
<td>MUL2N</td>
<td>F</td>
<td>Word multiplication by 2 to the power of n</td>
<td>94</td>
</tr>
<tr>
<td>MULD</td>
<td>FBoV</td>
<td>Multiplication double word</td>
<td>96</td>
</tr>
<tr>
<td>MULDI</td>
<td>F</td>
<td>Multiplication with division</td>
<td>98</td>
</tr>
<tr>
<td>NEGW</td>
<td>F</td>
<td>Negation word</td>
<td>100</td>
</tr>
<tr>
<td>NPULSE</td>
<td>FBmV</td>
<td>Pulses generator</td>
<td>101</td>
</tr>
</tbody>
</table>
### OVERVIEW OF BLOCKS ARRANGED ACCORDING TO THEIR CALL NAMES

<table>
<thead>
<tr>
<th>CE name</th>
<th>Type</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PACK4</td>
<td>FBoV</td>
<td>Pack binary variables in word (max 4 inputs)</td>
<td>103</td>
</tr>
<tr>
<td>PACK8</td>
<td>FBoV</td>
<td>Pack binary variables in word (max 8 inputs)</td>
<td>103</td>
</tr>
<tr>
<td>PACK16</td>
<td>FBoV</td>
<td>Pack binary variables in word (max 16 inputs)</td>
<td>103</td>
</tr>
<tr>
<td>PDM</td>
<td>FBmV</td>
<td>Pulse duration modulator</td>
<td>105</td>
</tr>
<tr>
<td>PI</td>
<td>FBmV</td>
<td>PI controller</td>
<td>107</td>
</tr>
<tr>
<td>PIDT1</td>
<td>FBmV</td>
<td>PIDT1 controller</td>
<td>112</td>
</tr>
<tr>
<td>SINIT</td>
<td>FBmV</td>
<td>Serial line Initialization</td>
<td>119</td>
</tr>
<tr>
<td>SUBD</td>
<td>FBoV</td>
<td>Subtraction double word</td>
<td>127</td>
</tr>
<tr>
<td>TIME_W</td>
<td>FBoV</td>
<td>Time word conversion</td>
<td>129</td>
</tr>
<tr>
<td>TOF</td>
<td>FBmv</td>
<td>Off delay</td>
<td>131</td>
</tr>
<tr>
<td>TON</td>
<td>FBmv</td>
<td>On delay</td>
<td>133</td>
</tr>
<tr>
<td>TP</td>
<td>FBmV</td>
<td>Monostable element &quot;constant&quot;</td>
<td>135</td>
</tr>
<tr>
<td>UHR</td>
<td>FBmV</td>
<td>Display and set clock</td>
<td>137</td>
</tr>
<tr>
<td>UNPACK4</td>
<td>FBoV</td>
<td>Unpacking a word into binary variables (max 4 outputs)</td>
<td>142</td>
</tr>
<tr>
<td>UNPACK8</td>
<td>FBoV</td>
<td>Unpacking a word into binary variables (max 8 outputs)</td>
<td>142</td>
</tr>
<tr>
<td>UNPACK16</td>
<td>FBoV</td>
<td>Unpacking a word into binary variables (max 16 outputs)</td>
<td>142</td>
</tr>
<tr>
<td>VRZ</td>
<td>FBmV</td>
<td>Up-down counter, word</td>
<td>144</td>
</tr>
<tr>
<td>VTASK</td>
<td>FBoV</td>
<td>Interruption task validation</td>
<td>146</td>
</tr>
<tr>
<td>WAND</td>
<td>F</td>
<td>AND combination, word</td>
<td>147</td>
</tr>
<tr>
<td>W_TIME</td>
<td>FBoV</td>
<td>Word time conversion</td>
<td>148</td>
</tr>
<tr>
<td>WDW</td>
<td>F</td>
<td>Word to double word conversion</td>
<td>150</td>
</tr>
<tr>
<td>WOL</td>
<td>F</td>
<td>Read word with enabling</td>
<td>151</td>
</tr>
<tr>
<td>WOR</td>
<td>F</td>
<td>OR combination, word</td>
<td>153</td>
</tr>
<tr>
<td>WXOR</td>
<td>F</td>
<td>XOR combination, word</td>
<td>154</td>
</tr>
</tbody>
</table>
ADDITION DOUBLE WORD

The value of the operand at input E1 is added to the value of the operand at input E2 and the result is assigned to the operand at output A.

The result is limited to the maximum or minimum value of the number range. If limiting occurred, a TRUE signal is assigned to the binary operand at output Q. If no limiting occurred, a FALSE signal is assigned to the binary operand at output Q.

**Block type**

Function block without historical values

**Parameters**

<table>
<thead>
<tr>
<th>Instance</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>DINT</td>
<td>Summand 1</td>
</tr>
<tr>
<td>E2</td>
<td>DINT</td>
<td>Summand 2</td>
</tr>
<tr>
<td>A</td>
<td>DINT</td>
<td>Total</td>
</tr>
<tr>
<td>Q</td>
<td>BOOL</td>
<td>Total, limited</td>
</tr>
</tbody>
</table>

**Description**

The value of the operand at input E1 is added to the value of the operand at input E2 and the result is assigned to the operand at output A.

The result is limited to the maximum or minimum value of the number range (-2147483647 … 2147483647). If limiting occurred, a TRUE signal is assigned to the binary operand at output Q. If no limiting occurred, a FALSE signal is assigned to the binary operand at output Q.

The inputs and outputs can neither be duplicated nor negated.
Example

Declaration:

```
ADDD_1 : ADDD;
ADDD_E1 AT %MD2000.0 : DINT;
ADDD_E2 AT %MD2000.1 : DINT;
ADDD_A AT %MD2000.2 : DINT;
ADDD_Q AT %MX0.0 : BOOL;
```

Translation in ABB IL:

```
!BA 0
ADDD
E1
E2
A
Q
```

FBD:

![FBD Diagram]

Function call in IL

```
CAL ADDD_1(E1 := ADDD_E1,
            E2 := ADDD_E2)
LD ADDD_1.Q
ST ADDD_Q
LD ADDD_1.A
ST ADDD_A
```

Function call in ST

```
ADDD_1 (E1 := ADDD_E1,
         E2 := ADDD_E2);
ADDD_Q:=ADDD_1.Q;
ADDD_A:=ADDD_1.A;
```
OFF DELAY

The TRUE/FALSE edge of input E is delayed by the time T and is output as a TRUE/FALSE edge at output A.

If input E returns to the TRUE level before the time T is expired, output A remains in the TRUE level.

Block type

Function block with historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>ASV</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>BOOL</td>
<td>Input signal</td>
</tr>
<tr>
<td>T</td>
<td>TIME</td>
<td>Delay time</td>
</tr>
<tr>
<td>A</td>
<td>BOOL</td>
<td>Delayed signal, operands M, A (not E, S)</td>
</tr>
</tbody>
</table>

Description

The TRUE/FALSE edge of input E is delayed by the time T and is output as a TRUE/FALSE edge at output A.

If input E returns to the TRUE level before the time T is expired, output A remains in the TRUE level.

Maximum time offset at the output: < 1 cycle time

Reasonable range for T: > 1 cycle time

The inputs and the output can neither be duplicated nor inverted.

General behavior

- Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program. An appropriate message of the operating system is not issued to the affiliated timer block in the PLC program until the timer has elapsed.
- Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:
  - Abort PLC program
  - RUN/STOP switch from RUN -> STOP
  - Warm or cold start
Example

Declaration:

\[
\begin{align*}
\text{ASV}_1 & : \text{ASV}; \\
\text{ASV}_E & \text{ AT } \%\text{MX}0.0 : \text{BOOL}; \\
\text{ASV}_T & \text{ AT } \%\text{MD}4001.0 : \text{TIME} := t\#2s500ms; (* 2500ms*) \\
\text{ASV}_A & \text{ AT } \%\text{MX}0.1 : \text{BOOL}; \\
\end{align*}
\]

Translation in ABB IL:

\[
\begin{align*}
!\text{BA} & 0 \\
\text{ASV} & \\
\text{E} & \\
\text{T} & := \text{t}#2s500ms; (* 2500ms*) \\
\text{A} & \\
\end{align*}
\]

FBD:

\[
\begin{array}{c}
\begin{array}{c}
\text{ASV}_1 \\
\text{ASV}_E \\
\text{ASV}_T \\
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\text{E} \\
\text{T} \\
\text{A} \\
\text{ASV}_A \\
\end{array}
\end{array}
\]

ABB IL of example:

\[
\begin{align*}
!\text{BA} & 0 \\
\text{ASV} & \\
\text{E} & \\
\text{T} & := \text{t}#2s500ms; \quad \text{; 2500} \\
\text{M0,1} & \\
\end{align*}
\]

Function call in IL

\[
\begin{align*}
\text{CAL} & \; \text{ASV}_1(E := \text{ASV}_E, \\
& \quad \text{T} := \text{ASV}_T) \\
\text{LD} & \; \text{ASV}_1.A \\
\text{ST} & \; \text{ASV}_A \\
\end{align*}
\]

Function call in ST

\[
\begin{align*}
\text{ASV}_1 & (E := \text{ASV}_E, \\
& \quad \text{T} := \text{ASV}_T); \\
\text{ASV}_A & := \text{ASV}_1.A; \\
\end{align*}
\]
SELECTION GATE, WORD

The AWT copy input1 (E1) or input2 (E2) in output (A1) depending of input switch (SWI) values.

Block type
Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>AWT</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWI</td>
<td>BOOL</td>
<td>Switchover input</td>
</tr>
<tr>
<td>E1</td>
<td>INT</td>
<td>Word input for SWI = FALSE</td>
</tr>
<tr>
<td>E2</td>
<td>INT</td>
<td>Word input for SWI = TRUE</td>
</tr>
<tr>
<td>A1</td>
<td>INT</td>
<td>Word output</td>
</tr>
</tbody>
</table>

Description
A FALSE signal at the binary input SWI allocates the value of the word operand at the input E1 to the word operand at the output A1. A TRUE signal at the binary input SWI allocates the value of the word operand at the input E2 to the word operand at the output A1.

Example

Declaration:

AWT1 : AWT;
AWT_SWI AT %MX0.0 : BOOL;
AWT_E1 AT %MW1000.0 : INT;
AWT_E2 AT %MW1000.1 : INT;
AWT_A1 AT %MW1000.2 : INT;

Translation in ABB IL:

!BA 0
AWT
SWI
E1
E2
A1

ABB IL of example:

!BA 0
AWT
M0,0
MW0,0
MW0,1
MW0,2

Function call in IL

CAL AWT_1(SWI := AWT_SWI,
E1 := AWT_E1,
E2 := AWT_E2)

LD AWT_1.A1
ST AWT_A1

Function call in ST

AWT_1 (SWI := AWT_SWI,
E1 := AWT_E1,
E2 := AWT_E2);

AWT_A1 := AWT_1.A1;
**Binary Selection Gate**

The AWTB copy input1 (E1) or input2 (E2) in output (A1) depending of input switch (SWI) values.

**Block type**
- Function block without historical values

**Parameters**

- **Instance**: AWTB
- **SWI**: BOOL - Switchover input
- **E1**: BOOL - Binary input for SWI = FALSE
- **E2**: BOOL - Binary input for SWI = TRUE
- **A1**: BOOL - Binary output

**Description**

A FALSE signal at the binary input SWI allocates the status of the operand at the input E1 to the operand at the output A1. A TRUE signal at the binary input SWI allocates the status of the operand at the input E2 to the operand at the output A1.

**Example**

**Declaration:**

```plaintext
AWTB1 : AWTB;
AWTB_SWI AT %MX0.0 : BOOL;
AWTB_E1 AT %MX0.1 : BOOL;
AWTB_E2 AT %MX0.2 : BOOL;
AWTB_A1 AT %MX0.3 : BOOL;
```

**Translation in ABB IL:**

```plaintext
!BA 0
AWTB
SWI
E1
E2
A1
```

**ABB IL of example:**

```plaintext
!BA 0
AWTB
M0,0
M0,1
M0,2
M0,3
```

**Function call in IL**

```plaintext
CAL  AWTB_1(SWI := AWTB_SWI,
E1 := AWTB_E1,
E2 := AWTB_E2)
LD  AWTB_1.A1
ST  AWTB_A1
```

**Function call in ST**

```plaintext
AWTB_1  (SWI := AWTB_SWI,
E1 := AWTB_E1,
E2 := AWTB_E2);
AWTB_A  := AWTB_1.A1;
```
The positive BCD coded number at input E is converted to a binary number and is assigned to the operand at output A.

**Block type**

Function

**Parameters**

- E INT BCD coded number
- (A) INT Binary number

**Description**

The positive BCD coded number at input E is converted to a binary number and is assigned to the operand at output A.

The input and the output can neither be duplicated nor negated.

**Definition:**

The significance of the digits in a BCD coded number and a hexadecimal number is defined as follows:

<table>
<thead>
<tr>
<th>BCD NUMBER</th>
<th>HEX NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 11 7 3 0</td>
<td>15 11 7 3 0</td>
</tr>
<tr>
<td>Z4 Z3 Z2 Z1</td>
<td>Z4 Z3 Z2 Z1</td>
</tr>
</tbody>
</table>

Numerical value:

| Z1 * 1 |
| Z2 * 10 |
| Z3 * 100 |
| Z4 * 1000 |

0 ≤ Zi ≤ 9

Remark:

At the BCD input, the block additionally accepts digits to which the following applies:

0 ≤ Zi ≤ F
Example 1

<table>
<thead>
<tr>
<th>BCD NUMBER</th>
<th>HEX NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 11 7 3 0</td>
<td>0 4 D 2</td>
</tr>
</tbody>
</table>

| Z1 = 4 * 1 | Z1 = 2 * 1 |
| Z2 = 3 * 10 | Z2 = 13 * 16 |
| Z3 = 2 * 100 | Z3 = 4 * 256 |
| Z4 = 1 * 1000 | Z4 = 0 * 4096 |

Example 2

<table>
<thead>
<tr>
<th>BCD NUMBER</th>
<th>HEX NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 2 F 4</td>
<td>2 8 7 2</td>
</tr>
</tbody>
</table>

| Z1 = 4 * 1 | Z1 = 2 * 1 |
| Z2 = 15 * 10 | Z2 = 7 * 16 |
| Z3 = 2 * 100 | Z3 = 8 * 256 |
| Z4 = 10 * 1000 | Z4 = 2 * 4096 |

Representation of a negative BCD number

A negative BCD number can be represented in the PLC by separate representation of the value and the sign. In doing so, the value of the BCD number is stored in a word variable and the information about the sign is stored in a binary variable.
Example

Declaration:

```
BCDDUAL_E AT %MW1002.0 : INT;
BCDDUAL_A AT %MW1002.1 : INT;
```

Translation in ABB IL:

```
!BA 0
BCDDUAL
E
A
```

FBD:

```
BCDDUAL E MW2,0
BCDDUAL A MW2,1
```

ABB IL of example:

```
!BA 0
BCDDUAL
MW2,0
MW2,1
```

Function call in IL

```
LD  BCD_E
BCDDUAL
ST  BCD_A
```

Function call in ST

```
BCD_A:=BCDDUAL(BCD_E);
```
The value of the operand at input E is limited to the range between the upper and lower limits.

The upper limit is specified by the operand at the OG input and the lower limit is specified by the one at the UG input.

The following applies:

\[ A = UG \text{ for } E < UG \]
\[ A = E \text{ for } UG \leq E < OG \]
\[ A = OG \text{ for } E > OG \]

The inputs and the output can neither be duplicated nor negated.
Example

Declaration:

BEG_E AT %MW1000.0 : INT;
BEG_OG AT %MW3001.0 : INT := 20000;
BEG_UG AT %MW3001.1 : INT := 10000;
BEG_A AT %MW1000.1 : INT;

Translation in ABB IL:

IBA 0
BEG
E
OG
UG
A

FBD:

```
BEG
  BEG_E
  BEG_OG
  BEG_UG
```

ABB IL of example:

IBA 0
BEG
MW0,0
KW1,0 ; 20000
KW1,1 ; 10000
MW0,1

Function call in IL

LD BEG_E
BEG BEG_OG, BEG_UG
ST BEG_A

Function call in ST

BEG_A := BEG(BEG_E, BEG_OG, BEG_UG);
This block monitors the binary values present at input E0 capable of duplication for a change. The BMELD number indicates the maximum number of input values. The following change annunciators are available:

- **BMELD8**: Binary value change annunciator with a maximum of 8 input values
- **BMELD16**: Binary value change annunciator with a maximum of 16 input values
- **BMELD32**: Binary value change annunciator with a maximum of 32 input values
- **BMELD127**: Binary value change annunciator with a maximum of 127 input values

**Block type**

Function block with historical values

**Parameters**

- **Instance**: BMELD(8..32) — Instance name
- **FREI**: BOOL — Block enabling
- **RESET**: BOOL — Reset
- **N**: INT — Number of input values
- **E0 .. En-1**: BOOL — Input values
- **NR**: INT — Number of the input where the change was recognized
- **A**: INT — Current input value
- **AEND**: BOOL — Change detected

**Description**

This block monitors the binary values present at the input E0 ... En-1 capable of duplication for a change. The inputs and outputs cannot be negated/inverted.

**Recognition of a change**

Each time the block is processed, the current input values at the inputs E0 ... En-1 are successively compared against the historical values (input values from the previous processing of the block). If a change is recognized at one of the inputs E0 ... En-1:

- this is indicated at output AEND
- the number of the input where the change was recognized is applied at output NR
- the changed input value is applied at output A

Each time the block is processed, a change at only one input is recognized. If a change is recognized, the inputs following the one where the change was previously determined are monitored the next time the block is processed.

**Initialization of historical values**

The first time the block is processed after PLC initialization (FREI = TRUE) or enabling of processing after it has been disabled (FREI changes from FALSE to TRUE), all current input values are assumed once as historical values and all outputs are set to the value 0. These initialized historical values now represent the starting basis for recognition of changes.
FREI  BOOL
Processing of the block is enabled with the FREI input.
FREI = FALSE → Block is not processed
FREI = TRUE → Processing of the block is enabled
If FREI = FALSE, the outputs of the block are also no longer updated.

RESET  BOOL
The block can be reset with the RESET input.
RESET = FALSE → No reset
RESET = TRUE → Reset of the block
A reset means:
  – Adoption of the current values at the inputs E0 ... En-1 as historical values.
  – All outputs are set to the value FALSE (0).

n  INT
The number of values to be monitored at the inputs E0 ... En-1 is specified at input n.
Range for n: 1 ≤ n ≤ max. number (8..127)

E0 .. En-1  BOOL
The operands to be monitored for a change are specified at the inputs E0 ... En-1.

NR  INT
The serial number of the input E0 ... En-1 where a change has been determined is applied at output NR. If no change is determined during processing of the block, the number of the last changed input is still applied at output NR.
The following allocation applies:
Change determined at E0 → NR = 0
Change determined at E1 → NR = 1
Change determined at En-1 → NR = n-1

A  BOOL
If a change is determined at one of the inputs E0 ... En-1, the changed input value is assigned to output A. If no change is determined at the inputs E0 ... En-1 during processing of the block, the value of the last changed input is still applied at output A.

AEND  BOOL
The output AEND indicates whether or not a change has been determined at the inputs E0 ... En-1.
AEND = FALSE → No change determined
AEND = TRUE → Change determined
Example

Declaration:

```plaintext
BMELD_8 : BMELD8;
BMELD_FREI AT %MX0.0 : BOOL;
BMELD_RESET AT %MX0.1 : BOOL;
BMELD_E0 AT %MX1.0 : BOOL;
BMELD_E1 AT %MX1.1 : BOOL;
BMELD_E2 AT %MX1.2 : BOOL;
BMELD_E3 AT %MX1.3 : BOOL;
BMELD_E4 AT %MX1.4 : BOOL;
BMELD_E5 AT %MX1.5 : BOOL;
BMELD_E6 AT %MX1.6 : BOOL;
BMELD_E7 AT %MX1.7 : BOOL;
BMELD_NR AT %MW1010.0 : INT;
BMELD_A AT %MX0.2 : BOOL;
BMELD_AEND AT %MX0.3 : BOOL;
```

Translation in ABB IL:

```plaintext
!BA 0
BMELD
FREI
RESET
#n
E0
E1
E2
E3
E4
E5
E6
E7
NR
A
AEND
```

Function call in IL

```plaintext
CAL BMELD_1(FREI := BMELD_FREI,
RESET := BMELD_RESET, n := 8,
E0 := BMELD_E0, E1 := BMELD_E1,
E2 := BMELD_E2, E3 := BMELD_E3,
E4 := BMELD_E4, E5 := BMELD_E5,
E6 := BMELD_E6, E7 := BMELD_E7 )

LD BMELD_1.NR
ST BMELD_NR
LD BMELD_1.A
ST BMELD_A
LD BMELD_1.AEND
ST BMELD_AEND
```

Note: In IL, the function call has to be performed in one line.

Function call in ST

```plaintext
BMELD_1 ( FREI := BMELD_FREI,
RESET := BMELD_RESET, n := 8,
E0 := BMELD_E0, E1 := BMELD_E1,
E2 := BMELD_E2, E3 := BMELD_E3,
E4 := BMELD_E4, E5 := BMELD_E5,
E6 := BMELD_E6, E7 := BMELD_E7 );

BMELD_NR := BMELD_1.NR;
BMELD_A := BMELD_1.A;
BMELD_AEND := BMELD_1.AEND;
```
ANALOG CHANNEL CONFIGURATION

The function block CONFIO1 is used to:

- configure the type (voltage, current or BALCO500/NI1000/PT100/PT1000) of one analog channel on the AC31 extensions.
- change the filtering time of the analog input.
- change the scale of the display value.
- lock or unlock the configuration for all analog channels of one analog extension.

CONFIO1 1 Analog Channel Configuration
CONFIO4 4 Analog Channels Configuration
CONFIO8 8 Analog Channels Configuration

Block type

Function block with historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>CONFIO(1..8)</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENA</td>
<td>BOOL</td>
<td>Block enabling</td>
</tr>
<tr>
<td>CHAN0..8</td>
<td>INT</td>
<td>Channel identification</td>
</tr>
<tr>
<td>TYPE0..8</td>
<td>INT</td>
<td>Type of analog channel</td>
</tr>
<tr>
<td>DOT0..8</td>
<td>INT</td>
<td>Position of dot of the display value</td>
</tr>
<tr>
<td>OFFS0..8</td>
<td>INT</td>
<td>Value of the offset for the display value</td>
</tr>
<tr>
<td>MULT0..8</td>
<td>INT</td>
<td>Value of the multiplication for the display value</td>
</tr>
<tr>
<td>FILT0..8</td>
<td>INT</td>
<td>Filtering time</td>
</tr>
<tr>
<td>RDY</td>
<td>BOOL</td>
<td>Processing of the configuration is completed</td>
</tr>
<tr>
<td>ERR</td>
<td>BOOL</td>
<td>An error is detected</td>
</tr>
</tbody>
</table>

Description

The function block CONFIO is used to:

- configure the type (voltage, current or BALCO500/NI1000/PT100/PT1000) of one analog channel on the AC31 extensions.
- change the filtering time of the analog input.
- change the scale of the display value.
- lock or unlock the configuration for all analog channels of one analog extension.

The analog channel configuration is set through the function block instead of the pushbutton on the front plate of the analog extension.

The configuration is stored in an internal EEPROM in the analog extension.

The scale of the display value is set according to the formula:

\[ \text{Display value} = \text{int value} \times \text{MULT0} / 32767 + \text{OFFS0} \]

The latest configured channel number of one analog extension is displayed.

<table>
<thead>
<tr>
<th>ENA</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>The function block is processed when ENA is on the rising edge FALSE-&gt;TRUE</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAN0</th>
<th>INT</th>
</tr>
</thead>
<tbody>
<tr>
<td>The analog channel to be configured is directly set. For example, %IW1000.0 for the analog input 0 on the analog extension at the address 1000, %QW1065.01 for the analog output 1 on the analog extension at the address 65</td>
<td></td>
</tr>
</tbody>
</table>
### ANALOG CHANNEL CONFIGURATION

<table>
<thead>
<tr>
<th>TYPE0</th>
<th>INT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of analog signal:</td>
<td></td>
</tr>
<tr>
<td>0: the channel is set to +/- 10 V</td>
<td></td>
</tr>
<tr>
<td>1: the channel is set to 0-20mA</td>
<td></td>
</tr>
<tr>
<td>2: the channel is set to 4-20mA</td>
<td></td>
</tr>
<tr>
<td>3: the channel is set to Pt100</td>
<td></td>
</tr>
<tr>
<td>4: the channel is set to Pt1000</td>
<td></td>
</tr>
<tr>
<td>5: the channel is set to Pt1000 3 wires</td>
<td></td>
</tr>
<tr>
<td>6: the channel is set to Pt1000 3 wires</td>
<td></td>
</tr>
<tr>
<td>8: the channel is set to Ni1000</td>
<td></td>
</tr>
<tr>
<td>9: the channel is set to BALCO500</td>
<td></td>
</tr>
<tr>
<td>14: unlock the configuration through the pushbutton on the analog extension front plate at the address xx when CHAN0 is %IW10xx.yy or %QW10xx.yy</td>
<td></td>
</tr>
<tr>
<td>15: lock the configuration through the pushbutton on the analog extension front plate at the address xx when CHAN0 is %IW10xx.yy or %QW10xx.yy</td>
<td></td>
</tr>
</tbody>
</table>

The configuration is automatically unlocked after a power supply ON.

<table>
<thead>
<tr>
<th>DOT0</th>
<th>INT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position of the dot on the display:</td>
<td></td>
</tr>
<tr>
<td>0: 4 digits are displayed without dot</td>
<td></td>
</tr>
<tr>
<td>Example: value=1234 display: 1234</td>
<td></td>
</tr>
<tr>
<td>1: 4 digits are displayed with dot on position 1</td>
<td></td>
</tr>
<tr>
<td>Example: value=1234 display: 123.4</td>
<td></td>
</tr>
<tr>
<td>2: 4 digits are displayed with dot on position 2</td>
<td></td>
</tr>
<tr>
<td>Example: value=1234 display: 12.34</td>
<td></td>
</tr>
<tr>
<td>3: 4 digits are displayed with dot on position 3</td>
<td></td>
</tr>
<tr>
<td>Example: value=1234 display: 1.234</td>
<td></td>
</tr>
</tbody>
</table>

If DOT0 < 0 or DOT0 > 3, the bit ERR is TRUE and the function is not processed.

<table>
<thead>
<tr>
<th>OFFS0</th>
<th>INT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value of the offset:</td>
<td></td>
</tr>
<tr>
<td>-32767 ( \leq ) OFFS0 ( \leq ) 32767</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MULT0</th>
<th>INT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value of the multiplication:</td>
<td></td>
</tr>
<tr>
<td>-32767 ( \leq ) MULT0 ( \leq ) 32767</td>
<td></td>
</tr>
</tbody>
</table>

If MULT0 = 0, the parameters OFFS0 and DOT0 are not used. In this case the scale is set to factory setting scale.

The parameter MULT0 can be used to set a channel value to the display.

<table>
<thead>
<tr>
<th>FILT0</th>
<th>INT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filtering time:</td>
<td></td>
</tr>
<tr>
<td>0: internal filter according to the documentation of analog extension</td>
<td></td>
</tr>
<tr>
<td>1 - 127: integration number</td>
<td></td>
</tr>
<tr>
<td>160: fast refresh time (50 ms instead of 120 ms in standard)</td>
<td></td>
</tr>
<tr>
<td>192: 60 Hz filter</td>
<td></td>
</tr>
<tr>
<td>224: 50 Hz filter</td>
<td></td>
</tr>
</tbody>
</table>

All channels of one extension will be affected by this parameter.

<table>
<thead>
<tr>
<th>RDY</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>This bit is set to FALSE during the function processing.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ERR</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>This bit is set to TRUE during one cycle (the bit RDY is set to TRUE in the same time).</td>
<td></td>
</tr>
</tbody>
</table>

An error is detected if:
- one parameter value is wrong
- the analog channel doesn't exist
- communication problem between central unit and analog extension

**Note:** One historical value is used by the function CONFIO1
Example

Declaration:

CONFI01 : CONFI01;
CONFI01_ENA AT %MX0.0 : BOOL;
CONFI01_CHAN0 AT %MW1000.0 : INT;
CONFI01_TYPE0 AT %MW1000.1 : INT;
CONFI01_DOT0 AT %MW1000.2 : INT;
CONFI01_OFFS0 AT %MW1000.3 : INT;
CONFI01_MULT0 AT %MW1000.4 : INT;
CONFI01_FILT0 AT %MW1000.5 : INT;
CONFI01_RDY AT %MX0.1 : BOOL;
CONFI01_ERR AT %MX0.2 : BOOL;

Translation in ABB IL:

!BA 0
CONFIO
#1
ENA
CHAN0
TYPE0
DOT0
OFFS0
MULT0
FILT0
RDY
ERR

FBD:

ABB IL of example:

!BA 0
CONFIO
#1
M0,0
MW0,0
MW0,1
MW0,2
MW0,3
MW0,4
MW0,5
M0,1
M0,2

Function call in IL

CAL CONFI01(ENA := CONFI01_ENA,
CHAN0 := CONFI01_CHAN0,
TYPE0 := CONFI01_TYPE0,
DOT0 := CONFI01_DOT0,
OFFS0 := CONFI01_OFFS0,
MULT0 := CONFI01_MULT0,
FILT0 := CONFI01_FILT0)

LD CONFI01.RDY
ST CONFI01_RDY
LD CONFI01.ERR
ST CONFI01_ERR

Note: In IL, the function call has to be performed in one line.

Function call in ST

CONFIO1 (ENA := CONFI01_ENA,
CHAN0 := CONFI01_CHAN0,
TYPE0 := CONFI01_TYPE0,
DOT0 := CONFI01_DOT0,
OFFS0 := CONFI01_OFFS0,
MULT0 := CONFI01_MULT0,
FILT0 := CONFI01_FILT0);

CONFIO1_RDY := CONFI01.RDY;
CONFIO1_ERR := CONFI01.ERR ;
COPYING MEMORY AREAS

This block copies n words from a source memory area into a target memory area.

Block type

Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>Instance name</td>
</tr>
<tr>
<td>FREI</td>
<td>BOOL Block enabling</td>
</tr>
<tr>
<td>ANZ</td>
<td>INT Number of word(s) to be copied</td>
</tr>
<tr>
<td>QOFF</td>
<td>INT Offset address of the start of the source area</td>
</tr>
<tr>
<td>QSEG</td>
<td>INT Segment address of the start of the source area</td>
</tr>
<tr>
<td>ZOFF</td>
<td>INT Offset address of the start of the target area</td>
</tr>
<tr>
<td>ZSEG</td>
<td>INT Segment address of the start of the target area</td>
</tr>
</tbody>
</table>

Description

This function block copies n words from a source memory area into a target memory area.

The contents of the source memory area are not changed.

In each case, the start of the source and target memory areas is specified at the block's inputs by means of the offset and segment addresses.

The inputs and outputs can neither be duplicated nor negated/inverted.

<table>
<thead>
<tr>
<th>FREI</th>
<th>BOOL Block enabling</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREI = FALSE</td>
<td>The block is not processed</td>
</tr>
<tr>
<td>FREI = TRUE</td>
<td>The block is processed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ANZ</th>
<th>INT Number n of words to be copied.</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 0</td>
<td>No copying</td>
</tr>
<tr>
<td>n = 8000H</td>
<td>A complete segment (64 kBytes) is copied</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>QOFF</th>
<th>INT Offset address of the start of the source area</th>
</tr>
</thead>
<tbody>
<tr>
<td>QSEG</td>
<td>INT Segment address of the start of the source area</td>
</tr>
<tr>
<td>ZOFF</td>
<td>INT Offset address of the start of the target area</td>
</tr>
<tr>
<td>ZSEG</td>
<td>INT Segment address of the start of the target area</td>
</tr>
</tbody>
</table>
Example

Declaration:

COPY1 : COPY;
COPY_FREI AT %MX0.0 : BOOL;
COPY_ANZ AT %MW3002.0 : INT := 16;
COPY_QOFF AT %MW3002.1 : INT := 16#1C00; (*MW00,00*)
COPY_QSEG AT %MW3002.2 : INT := 0;
COPY_ZOFF AT %MW3002.3 : INT := 16#1C40; (*MW03,00*)
COPY_ZSEG AT %MW3002.4 : INT := 0;

Translation in ABB IL:

IBA 0
COPY
FREI
ANZ
QOFF
QSEG
ZOFF
ZSEG

FBD:

```fbd
COPY1
  COPY
  COPY_FREI := FREI
  COPY_ANZ := ANZ
  COPY_QOFF := QOFF
  COPY_QSEG := QSEG
  COPY_ZOFF := ZOFF
  COPY_ZSEG := ZSEG
```

Function call in IL

CAL COPY1(FREI := COPY_FREI,
         ANZ := COPY_ANZ,
         QOFF := COPY_QOFF,
         QSEG := COPY_QSEG,
         ZOFF := COPY_ZOFF,
         ZSEG := COPY_ZSEG)

Note: In IL, the function call has to be performed in one line.

ABB IL of example:

IBA 0
COPY
M0,0
KW2,0 ; 16
KW2,1 ; #H1C00
KW2,2 ; 0
KW2,3 ; #H1C40
KW2,4 ; 0

Function call in ST

COPY1 (FREI := COPY_FREI,
       ANZ := COPY_ANZ,
       QOFF := COPY_QOFF,
       QSEG := COPY_QSEG,
       ZOFF := COPY_ZOFF,
       ZSEG := COPY_ZSEG);
CONFIGURE AC31 MODULES

The function block is used to configure the AC31 remote modules. The block can both send configuration parameters to the remote modules and also scan their currently set configuration.

**Block type**

Function block with historical values

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>CS31CO</td>
</tr>
<tr>
<td>FREI</td>
<td>BOOL</td>
</tr>
<tr>
<td>GRN</td>
<td>INT</td>
</tr>
<tr>
<td>CODE</td>
<td>INT</td>
</tr>
<tr>
<td>D1</td>
<td>INT</td>
</tr>
<tr>
<td>D2</td>
<td>INT</td>
</tr>
<tr>
<td>D3</td>
<td>INT</td>
</tr>
<tr>
<td>D4</td>
<td>INT</td>
</tr>
<tr>
<td>D5</td>
<td>INT</td>
</tr>
<tr>
<td>D6</td>
<td>INT</td>
</tr>
<tr>
<td>D7</td>
<td>INT</td>
</tr>
<tr>
<td>D8</td>
<td>INT</td>
</tr>
<tr>
<td>RDY</td>
<td>BOOL</td>
</tr>
<tr>
<td>OK</td>
<td>BOOL</td>
</tr>
<tr>
<td>ERR</td>
<td>INT</td>
</tr>
<tr>
<td>A1</td>
<td>INT</td>
</tr>
<tr>
<td>A2</td>
<td>INT</td>
</tr>
<tr>
<td>A3</td>
<td>INT</td>
</tr>
<tr>
<td>A4</td>
<td>INT</td>
</tr>
<tr>
<td>A5</td>
<td>INT</td>
</tr>
<tr>
<td>A6</td>
<td>INT</td>
</tr>
<tr>
<td>A7</td>
<td>INT</td>
</tr>
</tbody>
</table>

**Description**

The function block is used to configure the AC31 remote modules. The block can both send configuration parameters to the remote modules and also scan their currently set configuration.

Apart from configuration of the AC31 remote modules, the function block can also process further jobs (see list of jobs).

Enable for processing a job once is triggered by a FALSE/TRUE edge at input FREI.

The required job identification is specified at input CODE.

The parameters required for the job are planned at the inputs D1 ... D8.

Status messages are signalized at the outputs RDY, OK and ERR.

The response data of the job are available at the outputs A1 ... A7.

It may take several PLC cycles to process the job.
FREI 
Processing of the block is controlled via input FREI.

FREI = FALSE:
All block outputs are set to the value »FALSE«.
However, this is not valid, if a job is currently being processed, i.e. processing of a job which is currently being processed, is not affected by FREI = FALSE.

FREI = FALSE/TRUE edge:
Processing of the job is enabled.
Input FREI is no longer evaluated during processing of the job.

FREI = TRUE:
The block is not processed, i.e. it no longer changes its outputs. However, this is not valid, if a job is currently being processed.

GRN 
Group number with which the remote module is addressed by the PLC program.
Range: 0 … 63
Example:
On binary input E 12,08, »12« is the group number and »08« is the channel number.

CODE 
The identification of the job to be executed is specified at input CODE (see list of jobs on the next page).

D1…D8 
The parameters required for the job are preset at the inputs D1 … D8. The number of parameters depends on the job to be executed. There are also jobs requiring no parameters (see list of jobs on the next page).

RDY 
The output RDY indicates that processing of the job currently being processed is completed. This output does not indicate whether processing of the job was successful or not. The output RDY has therefore always to be considered together with the output OK.

RDY = FALSE and OK = TRUE:
Processing of the job is completed without errors. A new job can be started with a FALSE/TRUE edge at input FREI.

RDY = TRUE and OK = FALSE:
During processing of the job an error has been detected. The corresponding error identification is present at output ERR. A new job can be started with a FALSE/TRUE edge at input FREI.

RDY = TRUE:
Processing of an enabled job has not yet been completed (job is still running) or output RDY has been reset with FREI = FALSE.

OK 
Output OK indicates whether the job has been handled successfully or whether an error has been detected during processing. In case of an error, an error number is indicated at output ERR. The output OK is not valid until the job has been completed, i.e. if RDY = TRUE.
The following applies:
If RDY = TRUE and OK = TRUE: The job has been processed successfully.
OK = FALSE: During processing of the job an error has been detected.

ERR 
At the output ERR status and error identifications are output. The status identifications are output during processing of a job in order to signalize in what stage of processing the job currently is. After enabling a job, status identifications are signalized only for as long as RDY = FALSE.
The error identifications are output after completion of the job processing if an error has occurred. Error identifications are thus not signalized until
RDY = TRUE and OK = FALSE
Error identifications
ERR = 1: An illegal job identification has been specified at input CODE.
ERR = 2: Incorrect parameters have been specified at the inputs D1 … D8 (e.g. a group number for which there is no remote module on the CS31 system bus).
ERR = 3: The addressed AC31 remote module does not accept the job.
Status identifications
ERR = 8: The function block is waiting since a job of another user is currently being processed.
ERR = 10: The job has been sent to the receiver and the block is waiting for its response.

A1…A7 
After completion of job processing, the response is available at the outputs A1 … A7. The number of response parameters depends on the job performed (see list of jobs).
List of jobs
Processing a job consists of:
– transferring the job and
– supplying the OK response or not-OK response
The OK response is described in connection with the corresponding job.
The not-OK response of the individual jobs always looks as follows:

RDY: TRUE
OK: FALSE
ERR: 1. inadmissible job identification
2. wrong parameter; e.g. group number to which there exists no remote module
3. remote module does not accept the job

A1 ... A7: 0

• Updating of the maximum number of remote modules detected

The input INT EW 07,15 contains, amongst other things, the maximum number of remote modules detected in the past. The actual number of remote modules which exist at the moment may be less. This command is used to update this value. The modules which exist are counted and the value is stored. The user can inquire this value in the PLC program (EW 07,15, bit 8 ... 15).

– Job
  GRN: 255 (Master PLC with bus)
  CODE: 132
  D1 ... D8: not used

– OK response
  RDY: TRUE
  OK: TRUE
  A1 ... A7: 0

• Inquiring the open-circuit monitoring of an input to determine whether it is activated or deactivated

– Job
  GRN: group number 0 ... 63
  CODE: 32
  D1: channel number
  D2 ... D8: not used

– OK response
  RDY: TRUE
  OK: TRUE
  A1: 47. open-circuit monitoring ON
  32. open-circuit monitoring OFF
  A2 ... A7: 0

• Inquiring the open-circuit monitoring of an output to determine whether it is activated or deactivated

– Job
  GRN: group number 0 ... 63
  CODE: 33
  D1: channel number
  D2 ... D8: not used

– OK response
  RDY: TRUE
  OK: TRUE
  A1: 47. open-circuit monitoring ON
  32. open-circuit monitoring OFF
  A2 ... A7: 0

• Deactivating or activating the open-circuit monitoring of an input

– Job
  GRN: group number 0 ... 63
  CODE: 224. open-circuit monitoring ON
  160. open-circuit monitoring OFF
  D1: channel number
  D2 ... D8: not used

– OK response
  RDY: TRUE
  OK: TRUE
  A1 ... A7: 0

• Deactivating or activating the open-circuit monitoring of an output

– Job
  GRN: group number 0 ... 63
  CODE: 225. open-circuit monitoring ON
  161. open-circuit monitoring OFF
  D1: channel number
  D2 ... D8: not used

– OK response
  RDY: TRUE
  OK: TRUE
  A1 ... A7: 0

• Inquiring a channel to determine whether it is configured as input or input/output

– Job
  GRN: group number 0 ... 63
  CODE: 34
  D1: channel number
  D2 ... D8: not used

– OK response
  RDY: TRUE
  OK: TRUE
  A1: 34. input
  35. input/output
  A2 ... A7: 0
Configuration of a channel as input or input/output

- Job
  GRN: group number 0 ... 63
  CODE: 162. Input
        163. input/output
  D1: channel number
  D2 ... D8: not used
- OK response
  RDY: TRUE
  OK: TRUE
  A1 ... A7: 0

Inquiring the input delay of a channel

- Job
  GRN: group number 0 ... 63
  CODE: 38
  D1: channel number
  D2 ... D8: not used
- OK response
  RDY: TRUE
  OK: TRUE
  A1: input delay:
     2. 2 ms
     4. 4 ms
     ...
     30. 30 ms
     32. 32 ms
  A2 ... A7: 0

Setting the input delay of a channel

- Job
  GRN: group number 0 ... 63
  CODE: 166
  D1: channel number
  D2: input delay
     2. 2 ms
     4. 4 ms
     ...
     30. 30 ms
     32. 32 ms
- OK response
  RDY: TRUE
  OK: TRUE
  A1 ... A7: 0

Acknowledging errors on remote module

This command can be used to reset the error messages registered on the selected remote module. A reset is possible only if the cause of the error is no longer operative.

- Job
  GRN: group number 0 ... 63
  CODE: 232
  D1: lowest channel number on the module:
     0. lowest channel number on the module is 0 (<7)
     8. lowest channel number on the module is 8 (>7)
  D2: module type:
     0. binary input
     1. analog input
     2. binary output
     3. analog output
     4. binary input/output
     5. analog input/output
  D3 ... D8: not used
- OK response
  RDY: TRUE
  OK: TRUE
  A1 ... A7: 0

Acknowledging errors on remote module and resetting configuration values to default setting

In addition to the job «Acknowledging errors on remote module», all configurable settings are reset to the default setting.

- Job
  GRN: group number 0 ... 63
  CODE: 233
  D1: first channel number on the module:
     0. first channel number on the module is 0 (<7)
     8. first channel number on the module is 8 (>7)
  D2: module type:
     0. binary input
     1. analog input
     2. binary output
     3. analog output
     4. binary input/output
     5. analog input/output
  D3 ... D8: not used
- OK response
  RDY: TRUE
  OK: TRUE
  A1 ... A7: 0
● Inquiring the configuration of an analog input
   – Job
     GRN: group number 0 ... 63
     CODE: 42
     D1: channel number
     D2 ... D8: not used
   – OK response
     RDY: TRUE
     OK: TRUE
     A1: 50. input 0 ... 20 mA
     49. input 4 ... 20 mA
     A2 ... A7: 0

● Inquiring the configuration of an analog output
   – Job
     GRN: group number 0 ... 63
     CODE: 43
     D1: channel number
     D2 ... D8: not used
   – OK response
     RDY: TRUE
     OK: TRUE
     A1: 50. output 0 ... 20 mA
     49. output 4 ... 20 mA
     51. output +10 V
     A2 ... A7: 0

● Configuration of an analog input
   – Job
     GRN: group number 0 ... 63
     CODE: 170
     D1: channel number
     D2: 50. input 0 ... 20 mA
     49. input 4 ... 20 mA
     D3 ... D8: not used
   – OK response
     RDY: TRUE
     OK: TRUE
     A1 ... A7: 0

● Configuration of an analog output
   – Job
     GRN: group number 0 ... 63
     CODE: 171
     D1: channel number
     D2: 50. output 0 ... 20 mA
     49. output 4 ... 20 mA
     51. output +10 V
     D3 ... D8: not used
   – OK response
     RDY: TRUE
     OK: TRUE
     A1 ... A7: 0

● Inquiring the bus configuration
   The bus interface of the Master PLC has a list which stores specific data of the remote modules. The remote modules are numbered in this list in the order in which they can be found on the CS31 system bus. The internal number of the modules must be specified with this command. The response to this command is the group number stored under this number and status information on the corresponding module.
   – Job
     GRN: not evaluated
     CODE: 80
     D1: number from the module list (1 ... 31)
     D2 ... D8: not used
   – OK response
     RDY: TRUE
     OK: TRUE
     A1: status of the remote module:
     Bit 0 ... 3: number of process data bytes (binary module) or words (word module), which the module sends to the master.
     Bit 4 ... 7: number of process data bytes (binary module) or words (word module), which the master sends to the module
     A2: group number (0 ... 63)
     A3: Bit 0: 0. lowest channel number <7
         1. lowest channel number >7
         Bit 1: 0. binary module
         1. word module
     A4 ... A7: 0

● Read 1 ... 6 bytes
   – Job
     GRN: group number 0 ... 63
     CODE: 49.
     50. read 1 byte
     51. read 2 bytes
     52. read 3 bytes
     53. read 4 bytes
     54. read 5 bytes
     55. read 6 bytes
     D1: first channel number on the module:
     0. first channel number on the module is 0 (<7)
     8. first channel number on the module is 8 (>7)
     D2: module type:
     0. binary input
     1. analog input
     2. binary output
     3. analog output
     4. binary input/output
     5. analog input/output
     Note:
     Bit: even number (0, 2, 4)
     Word: odd number (1, 3, 5)
D3: byte start address (Low Byte)  
D4: byte start address (High Byte)  
D5 ... D8: not used  

- **OK response**  
  RDY: TRUE  
  OK: TRUE  
  A1: value of 1st byte  
  A2: value of 2nd byte or 0  
  A3: value of 3rd byte or 0  
  A4: value of 4th byte or 0  
  A5: value of 5th byte or 0  
  A6: value of 6th byte or 0  
  A7: 0  

- **Read 1 bit from 1 byte**  
  - **Job**  
    GRN: group number 0 ... 63  
    CODE: 63  
    D1: first channel number on the module:  
        0. first channel number on the module is 0 (<7)  
        8. first channel number on the module is 8 (>7)  
    D2: module type:  
        0. binary input  
        1. analog input  
        2. binary output  
        3. analog output  
        4. binary input/output  
        5. analog input/output  
    Note:  
    Bit: even number (0, 2, 4)  
    Word: odd number (1, 3, 5)  
    D3: byte start address (Low Byte)  
    D4: byte start address (High Byte)  
    D5: bit position within bytes 0 ... 7  
    D6 ... D8: not used  
    - **OK response**  
      RDY: TRUE  
      OK: TRUE  
      A1 ... A7: 0  

- **Write 1 bit of 1 byte**  
  - **Job**  
    GRN: group number 0 ... 63  
    CODE: 79  
    D1: first channel number on the module:  
        0. first channel number on the module is 0 (<7)  
        8. first channel number on the module is 8 (>7)  
    D2: module type:  
        0. binary input  
        1. analog input  
        2. binary output  
        3. analog output  
        4. binary input/output  
        5. analog input/output  
    Note:  
    Bit: even number (0, 2, 4)  
    Word: odd number (1, 3, 5)  
    D3: byte start address (Low Byte)  
    D4: byte start address (High Byte)  
    D5: bit position within bytes 0 ... 7  
    D6: bit value (0 or 1)  
    D7 ... D8: not used  
    - **OK response**  
      RDY: TRUE  
      OK: TRUE  
      A1 ... A7: 0  

- **Write 1 ... 4 bytes**  
  - **Job**  
    GRN: group number 0 ... 63  
    CODE: 65. write 1 byte  
    66. write 2 bytes  
    67. write 3 bytes  
    68. write 4 bytes  
    D1: first channel number on the module:  
        0. first channel number on the module is 0 (<7)  
        8. first channel number on the module is 8 (>7)  
    D2: module type:  
        0. binary input  
        1. analog input  
        2. binary output  
        3. analog output  
        4. binary input/output  
        5. analog input/output  
    Note:  
    Bit: even number (0, 2, 4)  
    Word: odd number (1, 3, 5)  
    D3: byte start address (Low Byte)  
    D4: byte start address (High Byte)  
    D5: bit position within bytes 0 ... 7  
    D6: bit value (0 or 1)  
    D7 ... D8: not used  
    - **OK response**  
      RDY: TRUE  
      OK: TRUE  
      A1 ... A7: 0
Example

Declaration:

```plaintext
CS31CO_1 : CS31CO;
CS31CO_FREI AT %MX0.0 : BOOL;
CS31CO_GRN AT %MW1001.0 : INT;
CS31CO_CODE AT %MW1001.1 : INT;
CS31CO_D1 AT %MW1002.0 : INT;
CS31CO_D2 AT %MW1002.1 : INT;
CS31CO_D3 AT %MW1002.2 : INT;
CS31CO_D4 AT %MW1002.3 : INT;
CS31CO_D5 AT %MW1002.4 : INT;
CS31CO_D6 AT %MW1002.5 : INT;
CS31CO_D7 AT %MW1002.6 : INT;
CS31CO_D8 AT %MW1002.7 : INT;
CS31CO_RDY AT %MX0.1 : BOOL;
CS31CO_OK AT %MX0.2 : BOOL;
CS31CO_ERR AT %MW1001.2 : INT;
CS31CO_A1 AT %MW1002.8 : INT;
CS31CO_A2 AT %MW1002.9 : INT;
CS31CO_A3 AT %MW1002.10 : INT;
CS31CO_A4 AT %MW1002.11 : INT;
CS31CO_A5 AT %MW1002.12 : INT;
CS31CO_A6 AT %MW1002.13 : INT;
CS31CO_A7 AT %MW1002.14 : INT;
```

Translation in ABB IL:

```plaintext
!BA 0
CS31CO
FREI
GRN
CODE
D1
D2
D3
D4
D5
D6
D7
D8
RDY
OK
ERR
A1
A2
A3
A4
A5
A6
A7
```

FBD:

```plaintext
CS31CO_1
CS31CO_FREI
CS31CO_GRN
CS31CO_CODE
CS31CO_D1
CS31CO_D2
CS31CO_D3
CS31CO_D4
CS31CO_D5
CS31CO_D6
CS31CO_D7
CS31CO_D8

FREI
GRN
CODE
D1
D2
D3
D4
D5
D6
D7
D8

RDY
OK
ERR
A1
A2
A3
A4
A5
A6
A7
```

ABB IL of example:

```plaintext
!BA 0
CS31CO
M0.0
MW1.0
MW1.1
MW2.0
MW2.1
MW2.2
MW2.3
MW2.4
MW2.5
MW2.6
MW2.7
M0.1
M0.2
MW1.2
MW2.8
MW2.9
MW2.10
MW2.11
MW2.12
MW2.13
MW2.14
```
Function call in IL

CAL CS31CO1 (FREI := CSCO_FREI,
GRN := CSCO_GRN,
CODE := CSCO_CODE,
D1 := CSCO_D1,
D2 := CSCO_D2,
D3 := CSCO_D3,
D4 := CSCO_D4,
D5 := CSCO_D5,
D6 := CSCO_D6,
D7 := CSCO_D7,
D8 := CSCO_D8)

LD CS31CO1.OK
ST CS31CO_OK
LD CS31CO1.ERR
ST CS31CO_ERR
LD CS31CO1.A1
ST CS31CO_A1
LD CS31CO1.A2
ST CS31CO_A2
LD CS31CO1.A3
ST CS31CO_A3
LD CS31CO1.A4
ST CS31CO_A4
LD CS31CO1.A5
ST CS31CO_A5
LD CS31CO1.A6
ST CS31CO_A6
LD CS31CO1.A7
ST CS31CO_A7
LD CS31CO1.RDY
ST CS31CO_RDY

Function call in ST

CS31CO1 (FREI := CSCO_FREI,
GRN := CSCO_GRN,
CODE := CSCO_CODE,
D1 := CSCO_D1,
D2 := CSCO_D2,
D3 := CSCO_D3,
D4 := CSCO_D4,
D5 := CSCO_D5,
D6 := CSCO_D6,
D7 := CSCO_D7,
D8 := CSCO_D8);

CS31CO_OK := CS31CO1.OK;
CS31CO_ERR := CS31CO1.ERR;
CS31CO_A1 := CS31CO1.A1;
CS31CO_A2 := CS31CO1.A2;
CS31CO_A3 := CS31CO1.A3;
CS31CO_A4 := CS31CO1.A4;
CS31CO_A5 := CS31CO1.A5;
CS31CO_A6 := CS31CO1.A6;
CS31CO_A7 := CS31CO1.A7;
CS31CO_RDY := CS31CO1.RDY;

Note: In IL, the function call has to be performed in one line.
ACKNOWLEDGE AC31 ERRORS

This block allows to acknowledge automatically error messages of AC31 remote modules.

Block type
Function block without historical values

Parameters
Instance CS31QU Instance name
FREI BOOL Enabling of the block processing

Description
This function block allows to acknowledge automatically FK3 and FK4 error messages of AC31 remote modules. Error messages are stored on the AC31 remote modules until they are acknowledged. Even if the error has been removed, the error message is still pending on the module until acknowledgement and is also signalized to the PLC until the message is acknowledged.

Processing of the block is enabled with a TRUE signal at input FREI, and the block then acknowledges AC31 errors continuously until the signal at the input FREI changes from TRUE to FALSE.

It may take several PLC cycles to acknowledge an error on an AC31 module.

If the function block is enabled, it constantly checks whether an AC31 error of class 3 or 4 has occurred and acknowledges this error.

1. An AC31 error of class 3 has occurred:

   The block acknowledges the error on the AC31 remote module which signalizes the error and also clears the error message on the PLC, i.e. the error flag M255.13 / MX255.13 is reset and LED FK3 is deactivated.

   Example of a FK3 error:
   – a remote module is disconnected from the CS31 system bus.

2. An AC31 error of class 4 has occurred:

   The block acknowledges the error on the AC31 remote module which signalizes the error and also clears the error message on the PLC, i.e. the error flag M255.14 / MX255.14 is reset.

   Example of a FK4 error:
   – a remote module signalizes an open circuit
Example

Declaration:
CS31QU_1 : CS31QU;
CS31QU_FREI AT %X62.0 : BOOL;

FBD:

Translation in ABB IL:
IBA 0
CS31QU
FREI

ABB IL of example:
IBA 0
CS31QU
E62,0

Function call in IL
CAL CS31QU_1(FREI := CS31QU_FREI)

Function call in ST
CS31QU_1(FREI := CS31QU_FREI);
UP COUNTER

This function block serves to count pulses.

Block type

Function block with historical values

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>CTU</td>
<td>Instance name</td>
</tr>
<tr>
<td>CU</td>
<td>BOOL</td>
<td>Pulse input</td>
</tr>
<tr>
<td>RESET</td>
<td>BOOL</td>
<td>Counter reset input</td>
</tr>
<tr>
<td>PV</td>
<td>INT</td>
<td>High counter limit</td>
</tr>
<tr>
<td>Q</td>
<td>BOOL</td>
<td>Limit indicator</td>
</tr>
<tr>
<td>CV</td>
<td>INT</td>
<td>Counter value</td>
</tr>
</tbody>
</table>

Description

Each positive edge (FALSE->TRUE edge) at the input CU increases the current value specified at output CV by 1.

CU BOOL

The pulse signal is allocated to the input CU. The positive edge of the pulse is evaluated in each case.

R BOOL

A 1 signal at the input R sets the counter content to the value 0. The reset input R has the highest priority.

PV INT

The high limit of the counter is specified at the input PV.

Q BOOL

The output Q indicates if the counter value is higher or not than the value at the input PV.

CV >= PV -> Q = TRUE
CV < PV  -> Q = FALSE

CV INT

The current counter value is available at the output CV. If the counter reaches the positive or negative limit of number range, the counter is limited to this value.

Number range

Low limit: 0
High limit: 7FFF_H +32767
**Example**

**Declaration:**

CTU1: CTU;
CTU_CU AT %MX1.0: BOOL;
CTU_RESET AT %MX0.0: BOOL;
CTU_PV %MW3002.0 : INT := 1000;
CTU_Q AT %MX1.1 : BOOL;
CTU_CV AT %MW1002.2 : INT;

**Translation in ABB IL:**

IBA 0
CTU
CU
RESET
PV
Q
CV

**FBD:**

![FBD Diagram]

**ABB IL of example:**

IBA 0
CTU
M1,0
M0,0
KW2,0 ; 1000
M1,1
MW2,2

**Function call in IL**

CAL CTU_1(CU := CTU_CU,
RESET := CTU_RESET,
PV := CTU_PV)

LD CTU_1.Q
ST CTU_Q
LD CTU_1.CV
ST CTU_CV

**Function call in ST**

CTU ( CU= CTU_CU,  
RESET:= CTU_RESET,  
PV := CTU_PV)

CTU_Q := CTU_1.Q;
CTU_CV := CTU_.CV;

Note: In IL, the function call has to be performed in one line.
The CTUH function block allows the counting of high-speed counters of units series 40 and 50.

**Block type**

Function block with historical values

**Parameters**

<table>
<thead>
<tr>
<th>Instance</th>
<th>CTUH</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM</td>
<td>INT</td>
<td>Counter mode</td>
</tr>
<tr>
<td>RE</td>
<td>BOOL</td>
<td>Counter reset input</td>
</tr>
<tr>
<td>SE</td>
<td>BOOL</td>
<td>Counter set input</td>
</tr>
<tr>
<td>INIT</td>
<td>INT</td>
<td>Set value</td>
</tr>
<tr>
<td>RPI</td>
<td>BOOL</td>
<td>Reset point indicator</td>
</tr>
<tr>
<td>CATCH</td>
<td>BOOL</td>
<td>Catch counter value</td>
</tr>
<tr>
<td>R_Q</td>
<td>BOOL</td>
<td>Reset bit overflow</td>
</tr>
<tr>
<td>Q</td>
<td>BOOL</td>
<td>Overflow</td>
</tr>
<tr>
<td>CV</td>
<td>INT</td>
<td>Counter value</td>
</tr>
<tr>
<td>CATV</td>
<td>INT</td>
<td>Caughed counter value</td>
</tr>
</tbody>
</table>

**Description**

Units series 40..50 have two high-speed counters that can be used in the following **modes**:

- **C1 : counting on the %IX62.00 input**
  - Counting start: on positive edge (FALSE->TRUE edge)
  - Representation: 1 INT (16 bits)
  - Overflow: when passing from -1 to 0.
  - Capture: on positive edge of the %IX62.02 input

- **C2 : counting on the %IX62.01 input**
  - Counting start: on positive edge (FALSE->TRUE edge)
  - Representation: 1 INT (16 bits)
  - Overflow: when passing from -1 to 0.
  - Capture: on positive edge of the %IX62.03 input

- **Incremental encoder : counting on the %IX62.00 and %IX62.01 inputs**
  - Counting start: on positive edge (FALSE/TRUE edge)
  - Representation: 1 INT (16 bits)
  - Overflow: when passing from -1 to 0 or from 0 to -1
  - Capture: on positive edge of the %IX62.02 input

In case of defective channel (i.e., one input not connected) the value increases of +1 and decreases of -1.

**NUM \hspace{1cm} DIRECT CONSTANT**

The counting mode is specified at the input NUM.

1 = counter mode C1
2 = counter mode C2
3 = incremental encoder
>3 = the block is not processed

**R \hspace{1cm} BOOL**

A TRUE signal at the input R resets the counter value and the capture register to the value 0. The reset input R has the highest priority.
If R = TRUE then CV = 0 and CATV = 0

**S \hspace{1cm} BOOL**

A TRUE signal at the input S loads the counter value with the preset value specified at the input INIT.
If S = TRUE then CV = INIT
**INIT**
The preset value is specified at the input INIT.

**RPI**
A TRUE signal at the input RPI validates the counter value capture and the counter reset during the capture. The RPI input has a higher priority than CATCH.

RPI = TRUE  Capture is valid on all the counters.

If there is a positive edge on %IX62,02 or %IX62,03, there is a hard capture of the counter. The counter is reset to 0.

**CATCH**
A TRUE signal at the input CATCH validates the counter value capture.

CATCH = 0  Capture not valid
CATCH = 1  Capture is valid on all the counters.

If there is a positive edge on %IX62,02 or %IX62,03, there is a hard capture of the counter. The counter is not reset to 0.

**R_Q**
A TRUE signal at the input R_Q resets the overflow to the value FALSE.

If R_Q = TRUE then Q = FALSE

**Q**
The overflow is specified at the output Q.

Q = TRUE when CV passes from -1 to 0 or from 0 to -1.

**CV**
The current counter value is available at the output CV.

**CATV**
The counter value when CATCH = TRUE is available at the output CATV.

**Number range:**
Integer word (16 bits)
Low limit : 8000H - 32768
High limit : 7FFFH + 32767
Example

Declaration:

CTUH_1: CTUH;
CTUH_RE AT %MX1.0: BOOL;
CTUH_SE AT %MX1.1: BOOL;
CTUH_INIT AT %MW3002.0: INT := 100;
CTUH_RPI AT %MX1.2: BOOL;
CTUH_CATCH AT %MX1.3: BOOL;
CTUH_R_Q AT %MX1.4: BOOL;
CTUH_Q AT %MX1.5: BOOL;
CTUH.CV AT %MW1010.0: INT;
CTUH.CATV AT %MW1010.1: INT;

Translation in ABB IL:

!BA 0
CTUH #NUM
RE
SE
INIT
RPI
CATCH
R_Q
Q
CV
CATV

ABB IL of example:

!BA 0
CTUH #1
M1,0
M1,1
MW2,0 ; 100
M1,2
M1,3
M1,4
M1,5
MW10,0
MW10,1

Function call in IL

CAL CTUH1 ( NUM := 01,
RE := CTUH_RE,
SE := CTUH_SE,
INIT := CTUH_INIT,
RPI := CTUH_RPI,
CATCH := CTUH_CATCH,
R_Q := CTUH_R_Q)
LD CTUH1.Q
ST CTUH_Q
LD CTUH1.CV
ST CTUH.CV
LD CTUH1.CATV
ST CTUH.CATV

Function call in ST

CTUH1 ( NUM := 01,
RE := CTUH_RE,
SE := CTUH_SE,
INIT := CTUH_INIT,
RPI := CTUH_RPI,
CATCH := CTUH_CATCH,
R_Q := CTUH_R_Q);
CTUH_Q := CTUH1.Q;
CTUH.CV := CTUH1.CV;
CTUH.CATV := CTUH1.CATV;
READ DIRECT INPUTS

The function block DIN reads ONE direct input of the central unit and its extensions. The direct input to be read is specified at the input INP.

Block type

Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>DIN</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>BOOL</td>
<td>Enable</td>
</tr>
<tr>
<td>INP</td>
<td>BOOL</td>
<td>Direct Input</td>
</tr>
</tbody>
</table>

Description

This function can only be used with series 40..50 central units.

The function block DIN reads ONE direct input of the central unit and its extensions. The direct input to be read is specified at the input INP.

The function block is useful:
- if the cycle time is long
- if the capacity utilization of the central unit is high

The PLC processor automatically builds an updated process image of the inputs at the start of each program cycle.

The DIN function block is able to get the physical input value specified at the input INP within a program cycle. This may be required in conjunction with specific applications, in order to detect and process the input signal changes more than once per program cycle.

- If the input to be read is on the central unit:
The new input value can be read instantaneously.

- If the input to be read is on an central extension unit:
The DIN function reads the present input value and generates at the same time a refreshment of the extension bus.

The DIN function block is useful if it is used in a time interruption knowing that the present input value read in one interruption is the input value refreshed by the DIN function of the previous interruption.

Note:
Remote inputs are refreshed every cycle time. The DIN function block can not be used for remote units on the CS31 bus.
Example

Declaration:
DIN_1 : DIN;
DIN_EN AT %MX0.0 : BOOL;
DIN_INP AT %IX62.0 : BOOL;

Translation in ABB IL:
!BA 0
DI
EN
INP

ABB IL of example:
!BA 0
DI
M0,0
E62,0

Function call in IL
CAL  DIN_1(EN := DIN_EN,
INP := DIN_INP)

Function call in ST
DIN_1  (EN := DIN_EN,
INP := DIN_INP);
DIVISION DOUBLE WORD

The value of the operand at input E1 is divided by the value of the operand at input E2 and the result is assigned to the operand at output A. The remainder is assigned to the operand at the output REST. If a remainder is produced, the result will always be rounded down. If the result lies outside of the permissible number range, it will be limited to the maximum or minimum value of the number range. If a limiting has been performed, a TRUE signal is assigned to the binary operand at output Q and the value 0 is assigned to output REST. If no limiting occurred, a FALSE signal is assigned to the binary operand at output Q.

Block type

Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>DIVD Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>DINT Dividend</td>
</tr>
<tr>
<td>E2</td>
<td>DINT Divisor</td>
</tr>
<tr>
<td>A</td>
<td>DINT Result (quotient)</td>
</tr>
<tr>
<td>REST</td>
<td>DINT Remainder</td>
</tr>
<tr>
<td>Q</td>
<td>BOOL Result limited</td>
</tr>
</tbody>
</table>

Description

The value of the operand at input E1 is divided by the value of the operand at input E2 and the result is assigned to the operand at output A. The remainder is assigned to the operand at output REST. If a remainder is produced, the result will always be rounded down. If the result lies outside of the permissible number range, it will be limited to the maximum or minimum value of the number range: -2147483647 (8000 0001H) ... 2147483647 (7FFF FFFFH). If a limiting has been performed, a TRUE signal is assigned to the binary operand at output Q and the value 0 is assigned to output REST. If no limiting occurred, a FALSE signal is assigned to the binary operand at output Q.

Division by »zero« is therefore also signalized at the binary output Q.

The inputs and outputs can neither be duplicated nor negated.

Limitation of Divisor

The divisor E2 is limited to the range –32767...+32767

Remainder handling

If the division results in a remainder, this is available at the double word output REST. The result of the division is always rounded down if a remainder occurs.

Example:

<table>
<thead>
<tr>
<th>3 : 3</th>
<th>4 : 3</th>
<th>5 : 3</th>
<th>6 : 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Remainder 0</td>
<td>1 Remainder 1</td>
<td>1 Remainder 2</td>
<td>2 Remainder 0</td>
</tr>
</tbody>
</table>

As the remainder is available at the output REST, the user can compare this to the divisor and can round the result at output A according to his own requirements.

Example:

Remainder > divisor / 2
→ round up the result at output A
Division by »zero«

If the divisor has the value »zero«, the positive or negative limit of the number range is assigned to output A.

For the division by »zero« the following applies:

A = -2147483647 (8000 0001H) if dividend is negative
A = +2147483647 (7FFF FFFFH) if dividend is positive

REST = 0  Output for the remainder
Q = TRUE  Output to signalize that the value at output A has been limited

Invalid result value

If the invalid value 8000 0000H is the result of the division, this will be corrected to the permissible limit 8000 0001H (-2 147 483 647), the binary output Q will be set to the value TRUE and the output REST will be set to the value 0.

Example

Declaration:

DIVD_1 : DIVD;
DIVD_E1 AT %MD2002.0 : DINT;
DIVD_E2 AT %MD2002.1 : DINT;
DIVD_A AT %MD2002.2 : DINT;
DIVD_REST AT %MD2002.3 : DINT;
DIVD_Q AT %MX0.0 : BOOL;

Translation in ABB IL:

!BA 0
DIVD
E1
E2
A
REST
Q

ABB IL of example:

!BA 0
DIVD
MD2,0
MD2,1
MD2,2
MD2,3
M0,0

Function call in IL

CAL  DIVD1(E1 := DIVD_E1,
         E2 := DIVD_E2)
LD  DIVD1.REST
ST  DIVD1.REST
LD  DIVD1.Q
ST  DIVD Q
LD  DIVD1.A
ST  DIVD_A

Function call in ST

DIVD1   (E1 := DIVD_E1,
         E2 := DIVD_E2);
DIVD _REST :=DIVD1.REST;
DIVD _Q :=DIVD1.Q;
DIVD _A :=DIVD1.A;
WRITE DIRECT OUTPUTS

The function block DOUT writes ONE direct output of the central unit and its extensions. The direct output to be write is specified at the input OUT.

Block type

Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>DOUT</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>BOOL</td>
<td>Enable</td>
</tr>
<tr>
<td>OUT</td>
<td>BOOL</td>
<td>Direct Output</td>
</tr>
</tbody>
</table>

Description

The function block DOUT writes ONE direct output of the central unit or its extension. The direct output to be written is specified at the output OUT.

The function block is useful:
- if the cycle time is long
- if the capacity utilization of the central unit is high

The PLC processor automatically outputs a process image of the direct outputs, which was updated during the program cycle, at the end of each program cycle. The DOUT function block is able to set the physical output specified at the OUT parameter within program cycles. This may be required in conjunction with specific applications, in order to have available the output signal changes more than once per program cycle.

If the output to be written is on the central unit:
The new output value can be write instantaneously.

If the output to be written is on an extension unit:
The new output value can be write with a delay of max 2 ms.

Note:
Remote outputs are refreshed every cycle time. The DOUT function block can not be used for remote outputs.
Example

Declaration:

DOUT_1 : DOUT;
DOUT_EN AT %MX0.0 : BOOL;
DOUT_OUT AT %QX62.0 : BOOL;

Translation in ABB IL:

!BA 0
DO
EN
OUT

ABB IL of example:

!BA 0
DO
M0,0
A62,0

FBD:

Function call in IL

CAL DOUT_1(EN := DOUT_EN)
LD DOUT_1.OUT
ST DOUT_OUT

Function call in ST

DOUT_1 (EN := DOUT_EN);
DOUT_OUT := DOUT_1.OUT;
A central unit of S40..50 can send ASCII messages through its RS232 serial interface with the DRUCK function block.

Each message has an identification number and can be composed of ASCII texts and operands values.

The texts and operand identifiers to be output are stored in the user program in the PLC directly by the DRUCK block. The numerical values to be output are conditioned for a diversity of representations by specifying a format identifier.

### Block type

ABB IL Program block inside Pragma

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>ABB IL Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREI</td>
<td>BOOL</td>
<td>Enable signal for output of one text (FALSE/TRUE edge)</td>
</tr>
<tr>
<td>SSK</td>
<td>INT</td>
<td>Serial interface identification</td>
</tr>
<tr>
<td>TXNR</td>
<td>INT</td>
<td>Number of the text to be output</td>
</tr>
<tr>
<td>RDY</td>
<td>BOOL</td>
<td>Ready</td>
</tr>
<tr>
<td>TX</td>
<td>Texts</td>
<td>Texts/operands capable of duplication</td>
</tr>
</tbody>
</table>

### Description

**IMPORTANT NOTE :**

**Initialization of the serial interface**

Before the DRUCK block can communicate with the serial interface, it must be initialized with the SINIT block.

**Communication between several DRUCK blocks and the same serial interface:**

Several DRUCK blocks can use the same serial interface. If the serial interface is engaged by one of the DRUCK blocks, the other DRUCK blocks automatically wait until it is free again. The priority access to the serial interface when several DRUCK blocks simultaneously have access to the same interface, corresponds to the sequence in which the DRUCK blocks are called in the user program: the first DRUCK block located at the beginning of the user program is given access first. The processing sequence must be planned by an appropriate mutual interlocking of the DRUCK blocks.

**Communication by a DRUCK and EMAS block with the same serial interface:**

A DRUCK block and a EMAS block (receiving of ASCII messages) can use the same serial interface without any special precaution.

**FREI**

If the block is ready (RDY = TRUE) and a FALSE/TRUE edge appears at the FREI input, the text identified at the input TXNR is sent through the serial interface specified at the SSK input.

If a FALSE/TRUE edge appears at the FREI input although the RDY output is equal to FALSE (i.e. the block is not ready yet for a new transfer), the FALSE/TRUE edge is ignored. Therefore, no new text transfer is started as long as the RDY signal is FALSE.
SSK

Central units can have one or two RS232 serial interface. The number of the interface through which the text is to be output is specified at the SSK input:
- SSK = 1 for COM1
- SSK = 2 for COM2

TXNR

The number of the text to be output is specified at the TXNR input: \(1 \leq \text{TXNR} < 99\)

The number of the text to be output must be present at the TXNR input until the block indicates the end of text transfer with a TRUE signal at its RDY output.

RDY

In the program cycle in which the block is called for the first time, and during the time when a text is output, RDY is equal to FALSE. As long as RDY is equal to FALSE, no new text output can be activated and all FALSE/TRUE edge present at the FREI input are ignored and lost.

After the first call of the block or after termination of a text output, RDY is equal to TRUE and the block is ready again for output of a new text.

<table>
<thead>
<tr>
<th>FREI</th>
<th>RDY</th>
<th>TXNR</th>
<th>SSK</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE/TRUE edge</td>
<td>TRUE</td>
<td>2</td>
<td>1</td>
<td>The text with the number 2 is output through interface 1</td>
</tr>
<tr>
<td>FALSE/TRUE edge</td>
<td>FALSE</td>
<td>2</td>
<td>1</td>
<td>FALSE/TRUE edge is ignored because the block is not ready</td>
</tr>
<tr>
<td>No FALSE/TRUE edge</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>No output of a new text</td>
</tr>
</tbody>
</table>

=> The RDY signal can be used for example at the FREI input to activate a new text transfer.

TX

Texts and operands to be output are directly written at the TX inputs. The way to write the messages is described here below.

Storage of texts in the central unit:

- Each text character counts as 1 character
- Each word operand counts as 1 to 6 characters
- Each double word operand counts as 10 to 11 characters

When the program is started, the PLC checks the texts to determine whether or not the maximum length is exceeded.

Syntax of texts:

- The text number
- One or several subtexts (optional)
- Operands with format identifier (optional)

Format identifier:

The PLC is capable of presenting the numerical values to be output on a screen or a printer in diverse ways. The format identifier specifies the type of the operand and the display format of its value.

This is planned directly before the operand to be output and consists of three digits. The 1st digit from the left specifies the operand type. There are 3 operand types:

- BOOL : 1
- INT : 2
- DINT : 3

Numbers 2 and 3 define the display format.

Examples:

- Format identifier 103
  Digit 1 : 1 : BOOL operand
  Digits 2 and 3 : 03 : Display format 03 (see table)

- Format identifier 204
  Digit 1 : 2 : INT operand
  Digits 2 and 3 : 04 : Display format 04 (see table)
- Format identifier 341
  Digit 1 : 3 : DINT operand
  Digits 2 and 3 : 41 : Display format 41 (see table)

**Possible display formats:**
All possible display formats are listed in the following table:
- identifiers applicable to word data types:
  01 to 16, 21 to 26, 33 to 36, 42 to 51, and 99
- identifiers applicable to double word data types:
  05 to 16 and 41 to 62

There are formats
- with leading zeros and
- with leading zeros substituted by blanks which are indicated in the table here below by `-`

<table>
<thead>
<tr>
<th>Format identifier</th>
<th>ASCII output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerical example=0012345678</td>
<td></td>
</tr>
<tr>
<td>01     x    8</td>
<td></td>
</tr>
<tr>
<td>02     xx   78</td>
<td></td>
</tr>
<tr>
<td>03     xxx  678</td>
<td></td>
</tr>
<tr>
<td>04     xxxx 5678</td>
<td></td>
</tr>
<tr>
<td>05     xxxxx 45678</td>
<td></td>
</tr>
<tr>
<td>Numerical example=1102215</td>
<td></td>
</tr>
<tr>
<td>06     xxxxxxx0221,5</td>
<td></td>
</tr>
<tr>
<td>07     xxxxx 022,15</td>
<td></td>
</tr>
<tr>
<td>08     xx,xxx 02,215</td>
<td></td>
</tr>
<tr>
<td>09     x,xxxx 0,2215</td>
<td></td>
</tr>
<tr>
<td>10     ,xxxxx ,02215</td>
<td></td>
</tr>
<tr>
<td>Numerical example=00331</td>
<td></td>
</tr>
<tr>
<td>11     +/- xxxx +00331</td>
<td></td>
</tr>
<tr>
<td>12     +/- xxxx,x +0033,1</td>
<td></td>
</tr>
<tr>
<td>13     +/- xxxx,xx +003,31</td>
<td></td>
</tr>
<tr>
<td>14     +/- xx,xxx +00,331</td>
<td></td>
</tr>
<tr>
<td>15     +/- x,xxxx +0,0331</td>
<td></td>
</tr>
<tr>
<td>16     +/- ,xxxxx +00331</td>
<td></td>
</tr>
<tr>
<td>Numerical example=00234</td>
<td></td>
</tr>
<tr>
<td>21     xxxxx  --234</td>
<td></td>
</tr>
<tr>
<td>Numerical example=00347</td>
<td></td>
</tr>
<tr>
<td>22     xxxx,x  --347</td>
<td></td>
</tr>
<tr>
<td>23     xx,xx   --3,47</td>
<td></td>
</tr>
<tr>
<td>24     xx,xxx  --,347</td>
<td></td>
</tr>
<tr>
<td>25     x,xxxx  -.0347</td>
<td></td>
</tr>
<tr>
<td>26     ,xxxxx  ,00347</td>
<td></td>
</tr>
<tr>
<td>Numerical example=00347</td>
<td></td>
</tr>
<tr>
<td>33     +/-xxx.xx  --3,47</td>
<td></td>
</tr>
<tr>
<td>34     +/-xx,xxx  --,347</td>
<td></td>
</tr>
<tr>
<td>35     +/-x,xxxx  -.0347</td>
<td></td>
</tr>
<tr>
<td>36     +/-xxxxx  +0,0347</td>
<td></td>
</tr>
</tbody>
</table>

**Numerical example:**
0012345678
41

- Possible display formats:
  All possible display formats are listed in the following table:
  - identifiers applicable to word data types:
    01 to 16, 21 to 26, 33 to 36, 42 to 51, and 99
  - identifiers applicable to double word data types:
    05 to 16 and 41 to 62

There are formats
- with leading zeros and
- with leading zeros substituted by blanks which are indicated in the table here below by `-`

<table>
<thead>
<tr>
<th>Format identifier</th>
<th>ASCII output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerical example=0012345678</td>
<td></td>
</tr>
<tr>
<td>01     x    8</td>
<td></td>
</tr>
<tr>
<td>02     xx   78</td>
<td></td>
</tr>
<tr>
<td>03     xxx  678</td>
<td></td>
</tr>
<tr>
<td>04     xxxx 5678</td>
<td></td>
</tr>
<tr>
<td>05     xxxxx 45678</td>
<td></td>
</tr>
<tr>
<td>Numerical example=1102215</td>
<td></td>
</tr>
<tr>
<td>06     xxxxxxx0221,5</td>
<td></td>
</tr>
<tr>
<td>07     xxxxx 022,15</td>
<td></td>
</tr>
<tr>
<td>08     xx,xxx 02,215</td>
<td></td>
</tr>
<tr>
<td>09     x,xxxx 0,2215</td>
<td></td>
</tr>
<tr>
<td>10     ,xxxxx ,02215</td>
<td></td>
</tr>
<tr>
<td>Numerical example=00331</td>
<td></td>
</tr>
<tr>
<td>11     +/- xxxx +00331</td>
<td></td>
</tr>
<tr>
<td>12     +/- xxxx,x +0033,1</td>
<td></td>
</tr>
<tr>
<td>13     +/- xxxx,xx +003,31</td>
<td></td>
</tr>
<tr>
<td>14     +/- xx,xxx +00,331</td>
<td></td>
</tr>
<tr>
<td>15     +/- x,xxxx +0,0331</td>
<td></td>
</tr>
<tr>
<td>16     +/- ,xxxxx +00331</td>
<td></td>
</tr>
<tr>
<td>Numerical example=00234</td>
<td></td>
</tr>
<tr>
<td>21     xxxxx  --234</td>
<td></td>
</tr>
<tr>
<td>Numerical example=00347</td>
<td></td>
</tr>
<tr>
<td>22     xxxx,x  --347</td>
<td></td>
</tr>
<tr>
<td>23     xx,xx   --3,47</td>
<td></td>
</tr>
<tr>
<td>24     xx,xxx  --,347</td>
<td></td>
</tr>
<tr>
<td>25     x,xxxx  -.0347</td>
<td></td>
</tr>
<tr>
<td>26     ,xxxxx  ,00347</td>
<td></td>
</tr>
<tr>
<td>Numerical example=00347</td>
<td></td>
</tr>
<tr>
<td>33     +/-xxx.xx  --3,47</td>
<td></td>
</tr>
<tr>
<td>34     +/-xx,xxx  --,347</td>
<td></td>
</tr>
<tr>
<td>35     +/-x,xxxx  -.0347</td>
<td></td>
</tr>
<tr>
<td>36     +/-xxxxx  +0,0347</td>
<td></td>
</tr>
</tbody>
</table>

**Format identifier**

**ASCII output**

- Possible display formats:
  All possible display formats are listed in the following table:
  - identifiers applicable to word data types:
    01 to 16, 21 to 26, 33 to 36, 42 to 51, and 99
  - identifiers applicable to double word data types:
    05 to 16 and 41 to 62

There are formats
- with leading zeros and
- with leading zeros substituted by blanks which are indicated in the table here below by `-`

<table>
<thead>
<tr>
<th>Format identifier</th>
<th>ASCII output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerical example=0012345678</td>
<td></td>
</tr>
<tr>
<td>01     x    8</td>
<td></td>
</tr>
<tr>
<td>02     xx   78</td>
<td></td>
</tr>
<tr>
<td>03     xxx  678</td>
<td></td>
</tr>
<tr>
<td>04     xxxx 5678</td>
<td></td>
</tr>
<tr>
<td>05     xxxxx 45678</td>
<td></td>
</tr>
<tr>
<td>Numerical example=1102215</td>
<td></td>
</tr>
<tr>
<td>06     xxxxxxx0221,5</td>
<td></td>
</tr>
<tr>
<td>07     xxxxx 022,15</td>
<td></td>
</tr>
<tr>
<td>08     xx,xxx 02,215</td>
<td></td>
</tr>
<tr>
<td>09     x,xxxx 0,2215</td>
<td></td>
</tr>
<tr>
<td>10     ,xxxxx ,02215</td>
<td></td>
</tr>
<tr>
<td>Numerical example=00331</td>
<td></td>
</tr>
<tr>
<td>11     +/- xxxx +00331</td>
<td></td>
</tr>
<tr>
<td>12     +/- xxxx,x +0033,1</td>
<td></td>
</tr>
<tr>
<td>13     +/- xxxx,xx +003,31</td>
<td></td>
</tr>
<tr>
<td>14     +/- xx,xxx +00,331</td>
<td></td>
</tr>
<tr>
<td>15     +/- x,xxxx +0,0331</td>
<td></td>
</tr>
<tr>
<td>16     +/- ,xxxxx +00331</td>
<td></td>
</tr>
<tr>
<td>Numerical example=00234</td>
<td></td>
</tr>
<tr>
<td>21     xxxxx  --234</td>
<td></td>
</tr>
<tr>
<td>Numerical example=00347</td>
<td></td>
</tr>
<tr>
<td>22     xxxx,x  --347</td>
<td></td>
</tr>
<tr>
<td>23     xx,xx   --3,47</td>
<td></td>
</tr>
<tr>
<td>24     xx,xxx  --,347</td>
<td></td>
</tr>
<tr>
<td>25     x,xxxx  -.0347</td>
<td></td>
</tr>
<tr>
<td>26     ,xxxxx  ,00347</td>
<td></td>
</tr>
<tr>
<td>Numerical example=00347</td>
<td></td>
</tr>
<tr>
<td>33     +/-xxx.xx  --3,47</td>
<td></td>
</tr>
<tr>
<td>34     +/-xx,xxx  --,347</td>
<td></td>
</tr>
<tr>
<td>35     +/-x,xxxx  -.0347</td>
<td></td>
</tr>
<tr>
<td>36     +/-xxxxx  +0,0347</td>
<td></td>
</tr>
</tbody>
</table>

**Special format:**
Output of a word operand’s HEX value:
The value of a INT operand is output directly as a hexadecimal value. Therefore, the value is not converted to ASCII before output.

98 Only the LOW BYTE (8 bits) of the word operand is output.
99 The LOW BYTE of the word operand is output first, followed by its HIGH BYTE.

**Important:** This special format is only permissible for the "word" data type.

**Inadmissible format:**
298 and 299
398 and 399
- Input of texts:
The following parts of the overall message are treated as independent operands:
- the text number e.g. # 1
- a subtext e.g. #" Text1 "#
- a format identifier e.g. # 203
- an operand e.g. MW002,03
- a further subtext e.g. #" Text2 "#

Input of special characters for screen or printer control:
Control characters such as "line feed" <LF> or "carriage return" <CR> are needed to arrange the message when output to a screen or printer. These special characters can be placed anywhere within a subtext. In the programming system, these special characters are entered by means of:

\Numerical value of the character

The numerical value of the character is specified as a three-digit decimal number.

Example:

```
#n Subtext
```

The following output is to be made on a printer:
First line
Blank line
Second line

To do this, the following text must be planned:
First line <CR> <LF> <LF> second line

The following applies:
<CR> = 013
<LF> = 010

The text input in the programming system is as follows:

```
#"First line\013\010\010second line"
```

Notes:
- The characters with the ASCII code >20H (=32D) must be entered with the keyboard. As an example, the character "!" can be entered with the keyboard, and not as \033.
- Characters, which are not special characters and which can not be entered with the keyboard, can be generated in the following way:
  Press and hold down the <ALT> key, now type the numerical code (decimal code) on the numerical keypad of the keyboard, then release the <ALT> key.
- The character with the ASCII code 255 is reserved for internal use of the programming software and must not be used otherwise.

- Example:
Text 1: The machine is ready
Text 2: The machine is not ready
Text 3: Level is (%MW1001.1=>MW01,01)m and temperature is (%MW1001,0=>MW01,00)°C.

The Program will be in Instruction List:
LD 0
(S40Inline
!BA0
DRUCK
M0,0
MW00,00
MW00,01
A00,00
#01
#"\010\013 The machine is ready : "#
#02
#"\010\013 The machine is not ready. "#
#03
#"\010\013 It is "#
#202
MW01,01
#"m and temperature is "#
#203
MW01,00
#"°C. "#
}`
DUAL TO BCD CONVERSION, WORD

The binary number at input E is converted to a BCD coded number and assigned to the operand at output A.

The binary number is represented in 16 bits and must lie within the range 0 ≤ E ≤ 270FH (corresponding to BCD 9999). The BCD number is limited to 9999 if it lies outside of this range. The BCD number is stored in a 16 bit INT.

**Block type**

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>E</th>
<th>INT</th>
<th>Binary number</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>INT</td>
<td>BCD coded number</td>
</tr>
</tbody>
</table>

**Description**

The binary number at input E is converted to a BCD coded number and assigned to the operand at output A.

The binary number is represented in 16 bits and must lie within the range 0 ≤ E ≤ 270FH (corresponding to BCD 9999). The BCD number is limited to 9999 if it lies outside of this range. The BCD number is stored in a 16 bit INT.

The input and the output can neither be duplicated nor negated.

**Definition:**

The significance of the digits in a hexadecimal number and a BCD coded number is defined as follows:

<table>
<thead>
<tr>
<th>HEX NUMBER</th>
<th>BCD NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 11 7 3 0 BIT</td>
<td>15 11 7 3 0</td>
</tr>
</tbody>
</table>

**Numerical value:**

<table>
<thead>
<tr>
<th>Z4</th>
<th>Z3</th>
<th>Z2</th>
<th>Z1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z1 * 1</td>
<td>Z2 * 16</td>
<td>Z3 * 256</td>
<td>Z4 * 4096</td>
</tr>
</tbody>
</table>

0 ≤ Z1 ≤ F

**Numerical value:**

<table>
<thead>
<tr>
<th>Z4</th>
<th>Z3</th>
<th>Z2</th>
<th>Z1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z1 * 1</td>
<td>Z2 * 10</td>
<td>Z3 * 100</td>
<td>Z4 * 1000</td>
</tr>
</tbody>
</table>

0 ≤ Z1 ≤ 9
Example

<table>
<thead>
<tr>
<th>HEX NUMBER</th>
<th>BCD NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 4 D 2</td>
<td>1 2 3 4</td>
</tr>
</tbody>
</table>

Z1 = 2 * 1 = 2
Z2 = 13 * 16 = 208
Z3 = 4 * 256 = 1024
Z4 = 0 * 4096 = 0

1234

Conversion of a negative binary number to a BCD number
A negative binary number with an amount less than 270FH can be converted to a BCD number, whereby the value and the sign of the BCD number are each stored in one flag.

Converting a binary number with an amount higher than 270FH
Not available for S40..50.

Example

Declaration:
DUALBCD_E AT %MW1002.1 : INT;
DUALBCD_A AT %MW1002.2 : INT;

Translation in ABB IL:
!BA 0
BINBCD
E
A

FBD:

FUNCTION DUALBCD

DUALBCD_E

DUALBCD_A

ABB IL of example:
!BA 0
BINBCD
MW2,1
MW2,2

Function call in IL
LD DUAL_E
DUALBCD
ST DUAL_A

Function call in ST
DUAL_A:= DUALBCD(DUAL_E);
AND COMBINATION, DOUBLE WORD

This function block generates, bit-by-bit, the AND combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output.

Block type
Function

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>DINT</td>
<td>Operand 1</td>
</tr>
<tr>
<td>E2</td>
<td>DINT</td>
<td>Operand 2</td>
</tr>
<tr>
<td>A</td>
<td>DINT</td>
<td>Result of the AND combination</td>
</tr>
</tbody>
</table>

Description
This function block generates, bit-by-bit, the AND combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output. The inputs and outputs can neither be duplicated nor negated.

Example

Declaration:

```
DWAND_E1 AT %MD2000.0 : DINT;
DWAND_E2 AT %MD2000.1 : DINT;
DWAND_A AT %MD2000.2 : DINT;
```

Translation in ABB IL:

```
IBA 0
DWAND
E1
E2
A
```

FBD:

```
!BA 0
DWAND
MD0,0
MD0,1
MD0,2
```

Function call in IL

```
LD DWAND_E1
DWAND DWAND_E2
ST DWAND_A
```

Function call in ST

```
DWAND_A := DWAND(E1 := DWAND_E1,
E2 := DWAND_E2);
```
OR COMBINATION, DOUBLE WORD

This function block generates, bit-by-bit, the OR combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output.

**Block type**

Function

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>DINT</td>
<td>Operand 1</td>
</tr>
<tr>
<td>E2</td>
<td>DINT</td>
<td>Operand 2</td>
</tr>
<tr>
<td>A</td>
<td>DINT</td>
<td>Result of the OR combination</td>
</tr>
</tbody>
</table>

**Description**

This function block generates, bit-by-bit, the OR combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output. The inputs and outputs can neither be duplicated nor negated.

**Example**

**Declaration:**

```
DWOR_E1 AT %MD2000.0 : DINT;
DWOR_E2 AT %MD2000.1 : DINT;
DWOR_A AT %MD2000.2 : DINT;
```

**Translation in ABB IL:**

```
!BA 0
DWOR
E1
E2
A
```

**ABB IL of example:**

```
!BA 0
DWOR
MD0,0
MD0,1
MD0,2
```

**Function call in IL**

```
LD DWOR_E1
DWOR DWOR_E2
ST DWOR_A
```

**Function call in ST**

```
DWOR_A := DWOR(E1 := DWOR_E1,
                E2 := DWOR_E2);
```
DOUBLE WORD TO WORD CONVERSION

The value of the double word operand at input E is converted to a word variable and the result is assigned to the word operand at output A1.

**Block type**

Function block without historical values

**Parameters**

<table>
<thead>
<tr>
<th>Instance</th>
<th>DWW instance name</th>
<th>E</th>
<th>DINT</th>
<th>Double word variable to be converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>INT</td>
<td></td>
<td></td>
<td>Result of conversion, word variable</td>
</tr>
<tr>
<td>A2</td>
<td>BOOL</td>
<td></td>
<td></td>
<td>Result limited</td>
</tr>
</tbody>
</table>

**Description**

The value of the double word operand at input E is converted to a word variable and the result is assigned to the word operand at output A1.

The result is limited to the maximum or minimum number range.

- max. number range: +32767 (7FFFH)
- min. number range: -32767 (8001H)

If limiting occurred, a TRUE signal is assigned to the binary operand at output A2. If no limiting occurred, a FALSE signal is assigned to the binary operand at output A2.

The input and the outputs can neither be duplicated nor negated.

**Example**

**Declaration:**

DWW_1 : DWW
DWW_E AT %MD2000.0 : DINT;
DWW_A1 AT %MW1000.0 : INT;
DWW_A2 AT %MW1000.1 : INT;

**Translation in ABB IL:**

!BA 0
DWW
E
A1
A2

**ABB IL of example:**

!BA 0
DWW
MD0,0
MW0,0
MW0,1

**Function call in IL**

CAL DWW_1(E := DWW_E)
LD DWW_1.A2
ST DWW_A2
LD DWW_1.A1
ST DWW_A1

**Function call in ST**

DWW_1 (E := DWW_E);
DWW_A2 := DWW_1.A2;
DWW_A1 := DWW_1.A1;
XOR COMBINATION, DOUBLE WORD

This function block generates, bit-by-bit, the XOR combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output.

Block type
Function

Parameters

| E1 | DINT | Operand 1 |
| E2 | DINT | Operand 2 |
| A  | DINT | Result of the XOR combination |

Description
This function block generates, bit-by-bit, the XOR combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output. The inputs and outputs can neither be duplicated nor negated.

Example

Declaration:

DWXOR_E1 AT %MD2000.0 : DINT;
DWXOR_E2 AT %MD2000.1 : DINT;
DWXOR_A AT %MD2000.2 : DINT;

Translation in ABB IL:

!BA 0
DWXOR
E1
E2
A

ABB IL of example:

!BA 0
DWXOR
MD0,0
MD0,1
MD0,2

Function call in IL

LD DWXOR_E1
DWXOR DWXOR_E2
ST DWXOR_A

Function call in ST

DWXOR_A := DWXOR(E1 := DWXOR_E1,
E2 := DWXOR_E2);
RECEPTION OF ASCII CHARACTERS

The function block EMAS:
- receives telegram through a serial interface of the PLC
- compares these telegrams to comparison telegrams stored in the user program
- and, if these agree, provides the user data of the telegram received at the block's outputs.

Block type
ABB IL Program block inside Pragma

Parameters

<table>
<thead>
<tr>
<th>ABB IL Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUIT</td>
<td>BOOL</td>
</tr>
<tr>
<td>SSK</td>
<td>INT</td>
</tr>
<tr>
<td>#ANU</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>constant</td>
</tr>
<tr>
<td>MEUN</td>
<td>BOOL</td>
</tr>
<tr>
<td>RDY</td>
<td>BOOL</td>
</tr>
<tr>
<td>TELN</td>
<td>INT</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>MW0</td>
<td>INT</td>
</tr>
<tr>
<td>VT0</td>
<td>Texts</td>
</tr>
</tbody>
</table>

Description

ASCII communication function block.

A AC31 central unit can receive ASCII messages through its RS232 serial interface with the EMAS function block.

The EMAS function block:
- receives telegrams through a serial interface of the PLC
- compares these telegrams to comparison telegrams stored in the user program
- and, if these agree, provides the user data of the telegram received at the block's outputs.

The received telegrams are fetched from the serial interface by an interface driver and are provided in a BUFFER for further processing by EMAS. The driver recognizes the end of the telegram by the end of telegram character. This end of telegram character is planned in the SINIT block.

IMPORTANT NOTE:
Initialization of the serial interface.

Before using the EMAS block, the serial interface used has to be initialized with the SINIT block.

Communication between several EMAS blocks and the same serial interface:
- EMAS blocks of a user program which access the same serial interface must be interlocked so that only ever one EMAS block is active. If this is not done, telegrams may be processed by the wrong EMAS and declared invalid.

- If both user program 1 and also user program 2 contain EMAS blocks which access the same serial interface, they must be interlocked so that only ever one EMAS block is active. If this is not done, telegrams may be processed by the wrong EMAS and declared invalid.

A telegram loss can be avoided by interlocking of the EMAS blocks. Interlocking must be planned so that only the EMAS block is enabled for which the telegram arriving through the interface is intended.
Communication by an EMAS block and a DRUCK block with the same serial interface:

An EMAS and a DRUCK block can use the same serial interface without special precautions having to be taken.

QUIT \hspace{1cm} BOOL

The input QUIT controls reception of telegrams and also serves the purpose of acknowledgement in the event of an error occurring.

QUIT = FALSE : Reception of telegrams enabled.
QUIT = TRUE : Reception of telegrams not enabled. Acknowledgment after reception of an invalid telegram.

If agreement with none of the stored comparison telegrams is ascertained on comparison of a received telegram, the EMAS automatically assumes the "error" state. In this case, EMAS no longer processes any new telegrams until the error is acknowledged with a 1 signal at the input QUIT and reception of telegrams is enabled again (next cycle) with a 0 signal at the input QUIT.

SSK \hspace{1cm} INT

The number of the interface through which the block receives its telegrams is specified at the SSK input (interface identifier).

The following applies:

COM1 : number = 1
COM2 : number = 2

#ANU \hspace{1cm} DIRECT CONSTANT

The number of outputs MW0 at which the block provides the received user information is specified at the input #ANU (number of user information items). This is specified as a direct constant.

I. e. numer = 10 => #ANU = #10 or #H0A

MEUN \hspace{1cm} BOOL

The output MEUN (flag invalid) indicates whether or not the data at the outputs MW is valid or invalid.

If a telegram is received and processed properly, the data at the outputs MW is declared valid.

The data at the outputs MW is declared invalid if the received telegram does not agree with any of the stored comparison telegrams or if the received telegram cannot be processed properly.

MEUN = FALSE -> Data at the outputs MW is valid
MEUN = TRUE -> Data at the outputs MW is invalid

RDY \hspace{1cm} BOOL

The output RDY (ready) indicates that a telegram has been received and processed.

The output RDY does not provide any information as to whether or not a valid or invalid telegram has been received.

RDY = FALSE -> Still no telegram has been received
RDY = TRUE -> A telegram has been received and processed

<table>
<thead>
<tr>
<th>QUIT</th>
<th>MEUN</th>
<th>RDY</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>The EMAS is disabled by QUIT=True. In doing so, the outputs MEUN and RDY are permanently set to FALSE.</td>
</tr>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>EMAS has received a valid telegram and is ready to receive a new telegram.</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>EMAS has received an invalid telegram. An acknowledgement at the QUIT input is necessary in order to be able to receive a new telegram. QUIT: FALSE-&gt;TRUE edge</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>Acknowledgement after reception of an invalid telegram. After acknowledgement, the EMAS is enabled again by QUIT=False</td>
</tr>
</tbody>
</table>

TELN \hspace{1cm} INT

If a valid telegram is received, the number of the affiliated comparison telegram is output through the output TELN (telegram number).

MW0 \hspace{1cm} INT

The output MW0 can be duplicated. The user data communicated in the telegram currently received is output through the MW0 parameters. This user data may consist of numerical values or any characters. This depends on which kind of dummy parameters have been planned in the comparison telegram. The user data of a telegram is stored beginning with the first MW parameter and in the sequence in which they are planned in the comparison telegram. As many outputs MW0 must be provided as are sufficient for the telegram with the most user data.

=> the operands must be written in ABB IL format.
VT0
The comparison telegrams to be stored in the PLC program are specified at the inputs VT0. The block is capable of processing from 1 to 99 telegrams. One telegram occupies 2 inputs, each telegram number being specified at one input and the actual telegram text being specified at the next one. The exact syntax and handling of the comparison telegrams are described here below.
The way to write the messages is described here below and an example is given at the end.

Detailed description of Comparison telegrams
1...99 comparison telegrams are stored directly after the EMAS block.
The comparison telegrams serve to identify
- the current telegrams received
- and the user data contained in the telegrams received
Each stored comparison telegram has a telegram number to identify it and may comprise up to 255 characters.
The comparison telegrams consist of :
- ASCII characters serving only to identify the telegram received,
- Dummy parameters for the user information to be received and to be output through the block’s outputs.

As regards the dummy parameters for the user information, EMAS function block distinguishes between dummy parameters for digits and dummy parameters for characters.

Dummy parameter for digits :
# (1 # per digit)

Dummy parameter for characters :
* (1 * per character/byte)

Dummy parameter for digits :
For each dummy digit parameter (#) of the comparison telegram, EMAS expects precisely one ASCII coded decimal digit in the telegram to be received. Up to 5 dummy digits constitute one dummy parameter group. Such a group of dummy digits represents the numerical value of a decimal number comprising up to 5 digits.

No dummy parameter is specified for the decimal number’s sign because EMAS takes it into account automatically. The EMAS allocates one user information output to each numerical value belonging to a dummy parameter group.

E.g. :
<table>
<thead>
<tr>
<th>Decimal number</th>
<th>Dummy parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>####</td>
</tr>
<tr>
<td>+1234</td>
<td>####</td>
</tr>
<tr>
<td>-1234</td>
<td>####</td>
</tr>
</tbody>
</table>

The EMAS block checks the received decimal number in relation to its significant range. Only numbers within the +32767 range can be processed in the PLC. If the received decimal number exceeds the significant range, the EMAS will automatically insert the maximum respective limit. The limit for a positive number is +32767 and the limit is -32767 for a negative number.

Dummy parameter for characters :
For each dummy character (*) of the comparison telegram, EMAS expects any one character/byte in the telegram to be received. These may comprise ASCII characters of letters, but also all other hex values from 0...FF.
The length of a dummy character group is up to 255. If this were the case, the complete comparison telegram would consist of dummy characters only. EMAS allocates the characters/bytes received without change and successively to its user information outputs MW0.

Syntax diagram : Structure of comparison telegrams

#n : Successive telegram number (direct constant 1...99)
#* : Start identifier for text input
“#” : Stop identifier for text input
* : Dummy parameter for character/byte
# : Dummy parameter for digits
TEXT : All ASCII characters 01 to FF except * and #
Input of comparison telegrams
- Each comparison telegram consists of:
  - the telegram number
  - the telegram text

The telegram number and the telegram text are two separate operands. This is why the telegram number and the telegram text occupy separate inputs in the FBD symbol of the EMAS block. Therefore, two inputs are needed for one comparison telegram.

Example:
First TEXT Input:   #1
(No. of the first comparison telegram)
Second TEXT Input:   "PRINT###
IDENTIFIER**** (Text of the first comparison telegram)

- Apart from the ASCII characters for * and #, all ASCII characters are possible in the telegram text.
- When entering special ASCII characters such as "start of line" <CR>, the following must be observed: special characters are entered by:
  \Numerical value of the character
The character's numerical value is specified as a three-digit decimal number.

Example for the EMAS block:
The following telegram text is to be compared:
Temperature <CR> boiler 1
The following applies: <CR> = 013

The text input in the programming system is as follows:
   *temperature\013boiler 1

Note:
- The characters with the ASCII code >20H or >32D must be entered with the keyboard. As an example, the character "!" must be entered with the keyboard, and not as \033.
- Characters, which are no special characters and which also could not be entered with the keyboard, can be generated in the following way:
  Press and hold down the <ALT> key, now type the numerical code (decimal code) on the numeric keypad of the keyboard, then release the <ALT> key.
- The character with the ASCII code 255 is reserved for internal use of the programming software and must not be used otherwise.

The Program will be in Instruction List:
LD 0
(S40Inline
!BA 0
EMAS
M00,00
MW00,00
#1
AW00,00
AW00,01
#1
"###"
ON DELAY

The FALSE/TRUE edge of input E is delayed by the time T and is output as a FALSE/TRUE edge at output A.

If input E returns to the FALSE level before the time T is expired, output A remains in the FALSE level.

Block type

Function block with historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>ESV</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>BOOL</td>
<td>Input signal</td>
</tr>
<tr>
<td>T</td>
<td>TIME</td>
<td>Delay time</td>
</tr>
<tr>
<td>A</td>
<td>BOOL</td>
<td>Delayed signal, operands M, A (not E, S)</td>
</tr>
</tbody>
</table>

Description

The FALSE/TRUE edge of input E is delayed by the time T and is output as a FALSE/TRUE edge at output A.

If input E returns to the FALSE level before the time T is expired, output A remains in the FALSE level.

Maximum time offset at the output: < 1 cycle time

Reasonable range for T: > 1 cycle time

The inputs and the output can neither be duplicated nor inverted.

General behavior

- Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program. An appropriate message of the operating system is not issued to the affiliated timer block in the PLC program until the timer has elapsed.
- Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:
  - Abort PLC program
  - RUN/STOP switch from RUN -> STOP
  - Warm or cold start
Example

Declaration:

ESV_1 : ESV;
ESV_E AT %MX0.0 : BOOL;
ESV_T AT %MD4002.0 : DINT := t#3s; (* 3000 ms*)
ESV_A AT %MX0.1 : BOOL;

Translation in ABB IL:

!BA 0
ESV
EI
A
Q

FBD:

\[
\begin{array}{c}
\text{ESV_1} \\
\text{ESV_E} \quad E \\
\text{ESV_T} \quad T
\end{array}
\]

ABB IL of example:

!BA 0
ESV
M0,0
KD2,0 ; 3000
M0,1

Function call in IL

CAL ESV_1(E := ESV_E, 
\( T := \text{ESV_T} \))
LD ESV_1.A
ST ESV_A

Function call in ST

ESV_1 \((E := \text{ESV_E}, 
\( T := \text{ESV_T} \));
ESV_A := ESV1.A;
FUNCTION GENERATOR

In an x/y coordinate system, a polygon is defined by n coordinate points X0/Y0...Xn-1/Yn-1. For each value at input X, the function block outputs the assigned y value of the polygon at output Y.

The FKG number indicates the maximum number of nodes. The following function generators are available:
- FKG2: Function generator with max. 2 nodes
- FKG4: Function generator with max. 4 nodes
- FKG16: Function generator with max. 16 nodes
- FKG32: Function generator with max. 32 nodes
- FKG64: Function generator with max. 64 nodes
- FKG256: Function generator with max. 256 nodes

<table>
<thead>
<tr>
<th>Block type</th>
<th>Function block without historical values</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>FKG(2..256) Instance name</td>
</tr>
<tr>
<td>X</td>
<td>INT Input for the x value of the polygon</td>
</tr>
<tr>
<td>n</td>
<td>INT Number of nodes</td>
</tr>
<tr>
<td>XC0..XCn-1</td>
<td>INT Input for x-values of the nodes</td>
</tr>
<tr>
<td>YC0..YCn-1</td>
<td>INT Input for y-values of the nodes</td>
</tr>
<tr>
<td>Y</td>
<td>INT Output for the y value of the polygon</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>The FKG number indicates the maximum number of nodes. The following function generators are available:</td>
<td></td>
</tr>
<tr>
<td>FKG2</td>
<td>Function generator with max. 2 nodes</td>
</tr>
<tr>
<td>FKG4</td>
<td>Function generator with max. 4 nodes</td>
</tr>
<tr>
<td>FKG16</td>
<td>Function generator with max. 16 nodes</td>
</tr>
<tr>
<td>FKG32</td>
<td>Function generator with max. 32 nodes</td>
</tr>
<tr>
<td>FKG64</td>
<td>Function generator with max. 64 nodes</td>
</tr>
<tr>
<td>FKG256</td>
<td>Function generator with max. 256 nodes</td>
</tr>
</tbody>
</table>

In an x/y coordinate system, a polygon is defined by n coordinate points X0/Y0...Xn-1/Yn-1. For each value at input X, the function block outputs the assigned y value of the polygon at output Y.

To the x-coordinate is valid:
X0 < X1 < X2 ... < Xn-1
2 ≤ n ≤ FKG number

Example:
For FKG 16, the following applies: 2 ≤ n ≤ 16

The block performs a linear interpolation between the nodes. The resulting polygon represents the connection between the input value x and the output value y.

For the interpolation between two nodes the following applies:

\[ y = \frac{(x - X_{i-1}) \cdot (Y_i - Y_{i-1})}{X_i - X_{i-1}} + Y_{i-1} \]

Note:
At the division it is always rounded down, i.e. a remainder with the division is not considered.

For the range outside of the nodes the following applies:

for x < X0, y = Y0
for x > Xn-1, y = Yn-1
**FUNCTION GENERATOR**

**FKG(2..256) S40**

### X

The current x coordinate is specified at input X. The block then defines the y coordinate assigned by the polygon.

### n

The number of nodes which are necessary to define the polygon is specified at input n.

### XC0-...-XCn-1

The x coordinates of the n nodes are specified at the inputs XC0 ... XCn-1.

### YC0-...-YCn-1

The y coordinates of the n nodes are specified at the inputs YC0 ... YCn-1.

### Y

The y coordinate assigned by the polygon of the specified x coordinate is output at output Y.

---

**Example**

**Declaration:**

FKG_2 : FKG;
FKG_X AT %MW1000.0 : INT;
FKG_XC0 AT %MW3002.0 : INT := 0;
FKG_YC0 AT %MW3002.1 : INT := 10000;
FKG_XC1 AT %MW3002.2 : INT := 0;
FKG_YC1 AT %MW3002.3 : INT := 20000;
FKG_Y AT %MW1000.1 : INT;

**ABB IL of example:**

!BA 0
FKG
MW0,0
#4 (* 2 * n *)
KW2,0
KW2,1
KW2,2
KW2,3
MW0,1

**FBD:**

![FBD Diagram]

**Function call in IL**

CAL FKG_1(X := FKG2_X, n := 2, XC0 := FKG2_XC0, YC0 := FKG2_YC0, XC1 := FKG2_XC1, YC1 := FKG2_YC1)

LD FKG_1.Y
ST FKG2_Y

**Note:** In IL, the function call has to be performed in one line.

**Function call in ST**

FKG_1(X := FKG2_X, n := 2, XC0 := FKG2_XC0, YC0 := FKG2_YC0, XC1 := FKG2_XC1, YC1 := FKG2_YC1);

FKG2_Y := FKG_1.Y;
FALLING EDGE DETECTION

A negative edge (TRUE/FALSE) at the input CLK generates a pulse at the output PULS which has the duration of 1 PLC program cycle.

Block type

Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>I_MINUS</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>BOOL</td>
<td>Input for TRUE/FALSE edge</td>
</tr>
<tr>
<td>Q</td>
<td>BOOL</td>
<td>Output for interrogation of the direct flag, operand M (not E, S)</td>
</tr>
<tr>
<td>PULS</td>
<td>BOOL</td>
<td>Pulse output, operand M, A (not E, S)</td>
</tr>
</tbody>
</table>

Description

A negative edge (TRUE/FALSE) at the input CLK generates a pulse at the output PULS which has the duration of 1 PLC program cycle.

The output Q is needed for edge detection. This flag must not be used again in the PLC program.

Duration of the pulse:
From recognition of the TRUE/FALSE edge by the connection element up to renewed processing of this connection element in the next program cycle.
Example

Declaration:

\[
\begin{align*}
\text{I\_MINUS\_1} & : \text{I\_MINUS}; \\
\text{I\_MINUS\_CLK} & @ \%\text{IX62.0} : \text{BOOL}; \\
\text{I\_MINUS\_Q} & @ \%\text{MX80.0} : \text{BOOL}; \\
\text{I\_MINUS\_PULS} & @ \%\text{MX0.0} : \text{BOOL};
\end{align*}
\]

Translation in ABB IL:

\[
\begin{align*}
\neg \text{CLK} &= \text{Q} \\
\neg \text{CLK} \land \text{Q} &= \text{R} \\
\text{R} &= \text{PULS}
\end{align*}
\]

FBD:

![FBD Diagram]

ABB IL of example:

\[
\begin{align*}
\neg \text{E62.0} &= \text{S} \text{M80.0} \\
\neg \text{E62.0} \land \text{M80.0} &= \text{R} \text{M80.0} \\
\text{R} \text{M80.0} &= \text{M0.0}
\end{align*}
\]

Function call in IL

\[
\begin{align*}
\text{CAL} & \quad \text{I\_MINUS\_1(\text{CLK} := \text{I\_MINUS\_CLK})} \\
\text{LD} & \quad \text{I\_MINUS\_1.Q} \\
\text{ST} & \quad \text{I\_MINUS\_Q} \\
\text{LD} & \quad \text{I\_MINUS\_1.PULS} \\
\text{ST} & \quad \text{I\_MINUS\_PULS}
\end{align*}
\]

Function call in ST

\[
\begin{align*}
\text{I\_MINUS\_1(\text{CLK} := \text{I\_MINUS\_CLK}, )};
\end{align*}
\]

\[
\begin{align*}
\text{I\_MINUS\_Q} & := \text{I\_MINUS\_1.Q}; \\
\text{I\_MINUS\_PULS} & := \text{I\_MINUS\_1.PULS};
\end{align*}
\]
RISING EDGE DETECTION

A positive edge (FALSE/TRUE) at the input CLK generates a pulse with the duration of one PLC program cycle at the PULS output.

**Block type**

Function block without historical values

**Parameters**

<table>
<thead>
<tr>
<th>Instance</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>BOOL</td>
<td>Input for FALSE/TRUE edge</td>
</tr>
<tr>
<td>Q</td>
<td>BOOL</td>
<td>Output for interrogation of the direct flag, operand M (not E, S)</td>
</tr>
<tr>
<td>PULS</td>
<td>BOOL</td>
<td>Pulse output, operand M, A (not E, S)</td>
</tr>
</tbody>
</table>

**Description**

A positive edge (FALSE/TRUE) at the input CLK generates a pulse with the duration of one PLC program cycle at the PULS output.

The output Q is needed for edge detection. This flag **must not be used again** in the PLC program.

**Duration of the pulse:**
From recognition of the FALSE/TRUE edge by the connection element until renewed processing of this connection element in the next program cycle.
Example

Declaration:

```
I_PLUS_1 : I_PLUS;
I_PLUS_CLK AT %I62.1 : BOOL;
I_PLUS_Q AT %MX80.1 : BOOL;
I_PLUS_PULS AT %MX0.1 : BOOL;
```

Translation in ABB IL:

```
!N CLK =R Q
! CLK &N Q =S Q = PULS
```

FBD:

```
I_PLUS_1

I_PLUS_CLK

I_PLUS

CLK

Q

I_PLUS_Q

PULS

I_PLUS_PULS
```

ABB IL of example:

```
!N E62.1 =R M80,1
! E62,1 &N M80,1
=S M80,1
= M0,1
```

Function call in IL

```
CAL I_PLUS_1(CLK := I_PLUS_CLK)
LD I_PLUS_1.Q
ST I_PLUS_1.Q
LD I_PLUS_1.PULS
ST I_PLUS_PULS
```

Function call in ST

```
I_PLUS_1(CLK := I_PLUS_CLK, );
I_PLUS_Q := I_PLUS_1.Q;
I_PLUS_PULS := I_PLUS_1.PULS;
```
READ BINARY VARIABLE, INDEXED

The function block serves for the purpose of indexed reading of binary variables.

**Block type**

Function block without historical values

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>IDLB</td>
<td>Instance name</td>
</tr>
<tr>
<td>FREI</td>
<td>BOOL</td>
<td>Block enabling</td>
</tr>
<tr>
<td>INDEX</td>
<td>INT</td>
<td>The index and the basic variable result in the source variable</td>
</tr>
<tr>
<td>BASIS</td>
<td>BOOL</td>
<td>Basic address of the binary variable</td>
</tr>
<tr>
<td>ZIEL</td>
<td>BOOL</td>
<td>Target variable, operand M, A (not E, S)</td>
</tr>
</tbody>
</table>

**Description**

This function block serves the purpose of indexed reading of binary variables.

The source variable to be read is obtained from indexing the basic variable.

The value of the source variable read is allocated to the target variable.

The group and channel numbers of the source flag (source variable) are determined from the basic flag and the index INDEX.

The source flag is:

\[ M (G\_Basis + A) , (K\_Basis + B) \]

where:

- \( G\_Basis \): Group number of the basic flag
- \( K\_Basis \): Channel number of the basic flag

The source flag is:

\[ %MX(000+A).(00+B) \]

Formula:

\[
\text{INDEX} \ \ \ \ \ \ \ \ \ \ \ \ 16 = A \text{ Remainder B}
\]

Group No. of the source flag:

- Group No. of the basic flag + A

Channel No. of the source flag:

- Channel No. of the basic flag + B

**Example:**

Basic variable: \( M00,00 \) AT \( %MX000.00 \)

\[ \text{INDEX} = 10 \]

\[ \rightarrow A = 10 : 16 \rightarrow A = 0, \text{Remainder B} = 10 \]

\[ \rightarrow \text{Source variable:} \ %MX000.10 \]

**Further examples:**

<table>
<thead>
<tr>
<th>Basic variable</th>
<th>INDEX</th>
<th>Source variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>( %MX000.00 )</td>
<td>0</td>
<td>( %MX000.00 )</td>
</tr>
<tr>
<td>( %MX000.00 )</td>
<td>2</td>
<td>( %MX001.00 )</td>
</tr>
<tr>
<td>( %MX000.02 )</td>
<td>18</td>
<td>( %MX001.04 )</td>
</tr>
</tbody>
</table>

**FREI**

Enable block

- FREI = FALSE \( \rightarrow \) Block is not processed
- FREI = TRUE \( \rightarrow \) The value of the source variable is read and allocated to the target variable ZIEL.
INDEX

The index value is specified at the input INDX. The source variable (see above for a calculation) results from the index and the basic variable.

Value range: -16383 < INDEX < +16383

BASIS

The basic variable is specified at the input BASIS. The source variable (see above for a calculation) results from the index and the basic variable.

ZIEL

The target variable is specified at the output ZIEL. The value of the selected source variable is allocated to the target variable ZIEL.

Example

Declaration:

```
IDLB_1 : IDLB;
IDLB_FREI AT %MX0.0 : BOOL;
IDLB_INDEX AT %MW1000.0 : INT;
IDLB_BASIS AT %MX20.0 : BOOL;
IDLB_ZIEL AT %MX0.1 : BOOL;
```

Translation in ABB IL:

```
!BA 0
IDLB
FREI
INDEX
BASIS
ZIEL
```

FBD:

```
```

Function call in IL

```
CAL IDLB_1(FREI := IDLB_FREI,
INDEX := IDLB_INDEX,
BASIS := IDLB_BASIS)
LD IDLB_1.ZIEL
ST IDLB_ZIEL
```

Note: In IL, the function call has to be performed in one line.

Function call in ST

```
IDLB_1 (FREI := IDLB_FREI,
INDEX := IDLB_INDEX,
BASIS := IDLB_BASIS);
IDLB_ZIEL:=IDLB_1.ZIEL;
```

ABB IL of example:

```
!BA 0
IDLB
M0,0
MW0,0
M20,0
M0,1
```
READ WORD VARIABLE, INDEXED

The function block serves for the purpose of indexed reading of word variables.

Block type

Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>IDLm</td>
<td>Instance name</td>
</tr>
<tr>
<td>FREI</td>
<td>BOOL</td>
<td>Block enabling</td>
</tr>
<tr>
<td>INDEX</td>
<td>INT</td>
<td>The index and the basic variable result in the source variable</td>
</tr>
<tr>
<td>BASIS</td>
<td>INT</td>
<td>Basic variable</td>
</tr>
<tr>
<td>ZIEL</td>
<td>INT</td>
<td>Target variable</td>
</tr>
</tbody>
</table>

Description

The function block serves for the purpose of indexed reading of word variables.

The source variable to be read is obtained from indexing the basic variable. The value of the source variable read is assigned to the target variable.

The inputs and outputs can neither be duplicated nor inverted nor negated.

The group and channel numbers of the source flag (source variable) are determined from the basic flag and the index.

The source flag is:

\[ MW (G_{\text{Basis}} + A), (K_{\text{Basis}} + B) \]

where:

- \( G_{\text{Basis}} \): Group number of the basic flag
- \( K_{\text{Basis}} \): Channel number of the basic flag

Formula:

\[ \frac{\text{INDEX}}{16} = A \quad \text{Remainder B} \]

Example:

Basic variable: \( MW00.00 \) AT \( %MW1000.00 \)

\( \text{INDEX} = 10 \)

\( \rightarrow A = 10 : 16 \rightarrow A = 0, \text{Remainder B} = 10 \)

\( \rightarrow \text{Source variable:} \)

\( %MW(1000+A).00+B) = %MW(1000+0).00+10) = %MW1000.10 \)

Further examples:

<table>
<thead>
<tr>
<th>Basic variable</th>
<th>INDEX</th>
<th>Source variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>%MW1000.00</td>
<td>0</td>
<td>%MW 1000.00</td>
</tr>
<tr>
<td>%MW1000.00</td>
<td>2</td>
<td>%MW 1001.00</td>
</tr>
<tr>
<td>%MW1000.02</td>
<td>16</td>
<td>%MW 1001.04</td>
</tr>
</tbody>
</table>
READ WORD VARIABLE, INDEXED

FREI
Block enabling
FREI = FALSE
→ Block is not processed
FREI = TRUE
→ The value of the source variable is read and assigned to the target variable ZIEL.

INDEX
The index value is specified at input INDEX. The source variable results from the index and the basic variable.
Value range: -32767 ≤ INDEX ≤ +32767
If INDEX is not within this range, the function block is not processed.

BASIS
The basic variable is specified at input BASIS. The source variable results from the index and the basic variable.

ZIEL
The target variable is specified at output ZIEL. The value of the selected source variable is assigned to the target variable ZIEL.

Example

Declaration:

IDLm_1 : IDLm;
IDLm_FREI AT %MX0.0 : BOOL;
IDLm_INDEX AT %MW1002.0 : INT;
IDLm_BASIS AT %MW1020.0 : INT;
IDLm_ZIEL AT %MW1002.1 : INT;

Translation in ABB IL:

!BA 0
IDL
FREI
INDEX
BASIS
ZIEL

ABB IL of example:

!BA 0
IDL
M0,0
MW2,0
MW20,0
MW2,1

Function call in IL

CAL IDLM_1(FREI := IDLM_FREI,
INDEX := IDLM_INDEX,
BASIS := IDLM_BASIS)
LD IDLM_1.ZIEL
ST IDLM_ZIEL

Note: In IL, the function call has to be performed in one line.

Function call in ST

IDLM_1 (FREI := IDLM_FREI,
INDEX := IDLM_INDEX,
BASIS := IDLM_BASIS);
IDLM_ZIEL := IDLM_1.ZIEL;
WRITE BINARY VARIABLE, INDEXED

The function block serves for the purpose of indexed writing of binary variables.

Block type

Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>IDSB</td>
<td>Instance name</td>
</tr>
<tr>
<td>FREI</td>
<td>BOOL</td>
<td>Block enabling</td>
</tr>
<tr>
<td>QUELL</td>
<td>BOOL</td>
<td>Source variable</td>
</tr>
<tr>
<td>INDEX</td>
<td>INT</td>
<td>The current target variable results from the index and the basic variable</td>
</tr>
<tr>
<td>BASIS</td>
<td>BOOL</td>
<td>Basic variable, operands M, A (not E, S)</td>
</tr>
</tbody>
</table>

Description

The function block serves for the purpose of indexed writing of binary variables.

When the block is enabled, the value of the source variable is read and assigned to the target variable. The target variable is defined by indexing the basic variable.

The inputs and outputs can neither be duplicated nor inverted nor negated.

The group and channel numbers of the target flag (target variable) are determined on the basic flag and the index INDEX.

The target flag is called:

M (G_Basis + A), (K_Basis + B)

where:

G_Basis : Group number of the basic flag
K_Basis : Channel number of the basic flag

Formula:

INDEX

-------- = A  Remainder B
16

Example:

Basic variable : M00,00 AT %MX000.00
INDEX = 10
-> A = 10 : 16 -> A = 0, Remainder B = 10
-> Target variable :
%MX(000+A).(00+B) =
%MX(000+0).(00+10) =
%MX000.10

Further examples:

<table>
<thead>
<tr>
<th>Basic variable</th>
<th>INDEX</th>
<th>Target variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>%MX000.00</td>
<td>0</td>
<td>%MX000.00</td>
</tr>
<tr>
<td>%MX000.00</td>
<td>2</td>
<td>%MX000.02</td>
</tr>
<tr>
<td>%MX000.00</td>
<td>16</td>
<td>%MX001.00</td>
</tr>
<tr>
<td>%MX000.02</td>
<td>18</td>
<td>%MX001.04</td>
</tr>
</tbody>
</table>

Group No. of the target flag:

Group No. of the basic flag + A

Channel No. of the target flag:

Channel No. of the basic flag + B
WRITE BINARY VARIABLE, INDEXED

IDSB S40

FREI
Block enabling
FREI = FALSE → Block is not processed
FREI = TRUE → The value of the source variable is read and assigned to the target variable.

QUELL
The source variable is specified at input QUELL. The value of this variable is read and assigned to the target variable.

INDEX
The index value is specified at input INDEX. The source variable results from the index and the basic variable.
Value range: \(-16383 < INDEX < +16383\)
If INDEX is not within this range, the function block is not processed.

BASIS
The basic variable is specified at the output BASIS.
The target variable (see above for a calculation) results from the index INDEX and the basic variable.

Example

Declaration:

```
IDSB_1 : IDSB;
IDSB_FREI AT %MX0.0 : BOOL;
IDSB_QUELL AT %MX1.0 : INT;
IDSB_INDEX AT %MW1001.1 : INT;
IDSB_BASIS AT %MX20.0 : INT;
```

Translation in ABB IL:

```
!BA 0
IDSB
FREI
QUELL
INDEX
BASIS
```

FBD:

```
IDSB
IDSB_FREI  FREI  BASIS
IDSB_QUELL QUELL
IDSB_INDEX INDEX
```

ABB IL of example:

```
!BA 0
IDSB
M0,0
M1,0
MW1,1
M20,0
```

Function call in IL:

```
CAL IDSB_1(FREI := IDSB_FREI,
QUELL := IDSB_QUELL,
INDEX := IDSB_INDEX)
LD IDSB_1.BASIS
ST IDSB_BASIS
```

Note: In IL, the function call has to be performed in one line.

Function call in ST:

```
IDSB_1 (FREI := IDSB_FREI,
QUELL := IDSB_QUELL,
INDEX := IDSB_INDEX);
IDSB_BASIS := IDSB_1.BASIS;
```
WRITE WORD VARIABLE, Indexed

The function block serves for the purpose of indexed writing of word variables.

**Block type**

Function block without historical values

**Parameters**

<table>
<thead>
<tr>
<th>Instance</th>
<th>IDSm</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREI</td>
<td>BOOL</td>
<td>Block enabling</td>
</tr>
<tr>
<td>QUELL</td>
<td>INT</td>
<td>Source variable</td>
</tr>
<tr>
<td>INDEX</td>
<td>INT</td>
<td>The index and the basic variable result in the source variable</td>
</tr>
<tr>
<td>BASIS</td>
<td>INT</td>
<td>Basic variable</td>
</tr>
</tbody>
</table>

**Description**

The function block serves for the purpose of indexed writing of word variables. When the block is enabled, the value of the source variable is read and assigned to the target variable. The target variable is defined by indexing the basic variable.

The inputs and outputs can neither be duplicated nor inverted nor negated.

The group and channel numbers of the target flag (target variable) are determined on the basic flag and the index INDEX.

The target flag is called: MW (G_Basis + A), (K_Basis + B)

where:

- G_Basis : Group number of the basic flag
- K_Basis : Channel number of the basic flag

**Example:**

Basic variable : MW00,00 AT %MW1000.00

INDEX = 10

-> A = 10 : 16 -> A = 0, Remainder B = 10

->Target variable :

%MW(1000+A).00+B) =
%MW(1000+0).00+10) =
%MW1000.10

**Further examples:**

<table>
<thead>
<tr>
<th>Basic variable</th>
<th>INDEX</th>
<th>Target variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>%MW1000.00</td>
<td>0</td>
<td>%MW1000.00</td>
</tr>
<tr>
<td>%MW1000.00</td>
<td>2</td>
<td>%MW1000.02</td>
</tr>
<tr>
<td>%MW1000.00</td>
<td>16</td>
<td>%MW1001.00</td>
</tr>
<tr>
<td>%MW1000.02</td>
<td>18</td>
<td>%MW1001.04</td>
</tr>
</tbody>
</table>

**Formula:**

\[
\text{INDEX} \mod 16 = A \quad \text{Remainder} \quad B
\]

**Group No. of the target flag:**

Group No. of the basic flag + A

**Channel No. of the target flag:**

Channel No. of the basic flag + B
WRITE WORD VARIABLE, INDEXED

FREI
Block enabling
FREI = FALSE → Block is not processed
FREI = TRUE → The value of the source variable is read and assigned to the target variable.

BOOL

INDEX
The index value is specified at input INDEX. The source variable results from the index and the basic variable.
Value range: -32767 < INDEX < +32767
If INDEX is not within this range, the function block is not processed.

INT

QUELL
The source variable is specified at input QUELL. The value of this variable is read and assigned to the target variable.

INT

INDEX

INT

The basic variable is specified at output BASIS. The target variable (see above for a calculation) results from the index INDEX and the basic variable.

Example

Declaration:
IDSm_1 : IDSm
IDSm_FREI AT %MX0.0 : BOOL;
IDSm_QUELL AT %MW1020.0 : INT;
IDSm_INDEX AT %MW1002.1 : INT;
IDSm_BASIS AT %MW1002.2 : INT;

Translation in ABB IL:
!BA 0
IDSm
FREI
QUELL
INDEX
BASIS

ABB IL of example:
!BA 0
IDSm
M0,0
MW20,0
MW2,1
MW2,2

Function call in IL
CAL  IDSm_1(FREI := IDSM_FREI,
QUELL := IDSM_QUELL,
INDEX := IDSM_INDEX)
LD  IDSM_1.BASIS
ST  IDSM_BASIS

Note: In IL, the function call has to be performed in one line.

Function call in ST
IDSM_1 (FREI := IDSM_FREI,
QUELL := IDSM_QUELL,
INDEX := IDSM_INDEX);
IDSM_BASIS := IDSM_1.BASIS;
LIST ALLOCATOR

The function block has a list of word data at its inputs. With a list pointer, it selects a value out of this list and applies it at its output.

The LIZU number indicates the maximum number of word data. The following list allocators are available:

- LIZU8 List allocator for max. 8 word data
- LIZU16 List allocator for max. 16 word data
- LIZU32 List allocator for max. 32 word data
- LIZU64 List allocator for max. 64 word data
- LIZU256 List allocator for max. 256 word data

Block type

Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>LIZU(8..256)</td>
<td>Instance name</td>
</tr>
<tr>
<td>ZEIG</td>
<td>INT</td>
<td>Pointer to the list of word data</td>
</tr>
<tr>
<td>E0..En-1</td>
<td>INT</td>
<td>List of direct constants</td>
</tr>
<tr>
<td>A_E</td>
<td>BOOL</td>
<td>0 &lt; ZEIG &lt; n, i.e. pointer in the valid range, operands M, A (not E, S)</td>
</tr>
<tr>
<td>A</td>
<td>INT</td>
<td>Selected word value</td>
</tr>
</tbody>
</table>

Description

The function block has a list of word data at its inputs. With a list pointer, it selects a value out of this list and applies it at its output.

The inputs and outputs cannot be inverted and duplicated.

ZEIG INT

The pointer to the word value to be selected from the list is specified at input ZEIG.

The following allocation applies:

\[
\begin{align*}
\text{ZEIG} = 0 & \quad \rightarrow \quad \text{Word value at E0} \\
\text{ZEIG} = 1 & \quad \rightarrow \quad \text{Word value at E1} \\
\vdots & \quad \vdots \\
\text{ZEIG} = n-1 & \quad \rightarrow \quad \text{Word value at En-1}
\end{align*}
\]

The value at input ZEIG is subjected to a validity check. The result of this range check is signalized at output A_E.

Allowed range:

\[0 \leq \text{ZEIG} < n-1\]

With n: Number of the inputs E0..En-1.

No allocation to output A takes place if the value at input ZEIG is outside the allowed range.

E0 .. En-1 INT

The word values out of which one is selected with the value at input ZEIG and assigned to output A are specified at the inputs E0 .. En-1.

A_E BOOL

Output A_E indicates whether the value of the list pointer (input ZEIG) is within the allowed range.

Allowed range:

\[0 \leq \text{ZEIG} < n-1\]

With n: Number of the inputs E0..En-1.

The following applies:

\[
\begin{align*}
\text{ZEIG in the allowed range} & \quad \rightarrow \quad \text{A_E} = \text{TRUE} \\
\text{ZEIG in the forbidden range} & \quad \rightarrow \quad \text{A_E} = \text{FALSE}
\end{align*}
\]

If the list pointer has a forbidden value, no allocation of a value to output A takes place because no word value can be selected. In this case, output A is not updated.

A INT

The value of the selected word value is assigned to the operand at output A.
Example

Declaration:

LIZU8_1 : LIZU8;
LIZU_ZEIG AT %MW1000.0: INT;
LIZU_A_E AT %MX0.0 : BOOL;
LIZU_A AT %MW1000.1 : INT;

Translation in ABB IL:

!BA 0
LIZU
ZEIG
#n
#E0
#E1
#E2
#E3
#E4
#E5
#E6
#E7
A_E
A

FBD:

<table>
<thead>
<tr>
<th>LIZU0_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIZU8</td>
</tr>
<tr>
<td>ZEIG</td>
</tr>
<tr>
<td>A_E</td>
</tr>
<tr>
<td>LIZU_A</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Function call in IL

CAL LIZU_8(ZEIG := LIZU8_ZEIG,
            n := 8,
            E0 := 1,
            E1 := 2,
            E2 := 3,
            E3 := 4,
            E4 := 5,
            E5 := 6,
            E6 := 7,
            E7 := 8)

LD LIZU8_1.A_E
ST LIZU_A_E
LD LIZU8_A
ST LIZU8_A

Note: In IL, the function call has to be performed in one line.

Function call in ST

LIZU_8
ZEIG := LIZU8_ZEIG,
       n := 8,
       E0 := 1,
       E1 := 2,
       E2 := 3,
       E3 := 4,
       E4 := 5,
       E5 := 6,
       E6 := 7,
       E7 := 8);

LIZU8_A_E := LIZU8_1.A_E;
LIZU8_A := LIZU8_1.A;
MONOSTABLE ELEMENT »ABORT«  

MOA S40

A FALSE/TRUE edge at input E causes a FALSE/TRUE edge at output A. If input E remains at TRUE level, a TRUE/FALSE edge is applied at output A after the duration T has elapsed.

Output A is also immediately set back to FALSE level if input E returns to FALSE level before the time period T has expired.

Maximum time offset at the output: < 1 cycle time

Block type

Function block with historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>MOA</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>BOOL</td>
<td>Input signal</td>
</tr>
<tr>
<td>T</td>
<td>TIME</td>
<td>Pulse length</td>
</tr>
<tr>
<td>A</td>
<td>BOOL</td>
<td>Output signal, operands M, A (not E, S)</td>
</tr>
</tbody>
</table>

Description

A FALSE/TRUE edge at input E causes a FALSE/TRUE edge at output A. If input E remains at TRUE level, a TRUE/FALSE edge is applied at output A after the duration T has elapsed.

Output A is also immediately set back to FALSE level if input E returns to FALSE level before the time period T has expired.

Valid time range: 5 ms .. 24,8 days.

Maximum time offset at the output: < 1 cycle time

Reasonable range for T: > 1 cycle time

The inputs and the output can neither be duplicated nor inverted.

General behavior

- Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program. An appropriate message of the operating system is not issued to the affiliated timer block in the PLC program until the timer has elapsed.

- Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:
  - Abort PLC program
  - RUN/STOP switch from RUN -> STOP
  - Warm or cold start

```
E
A
\[ t \]
\[ T \]
\[ t \]
\[ < T \]
```
Example

Declaration:

\[
\begin{align*}
\text{MOA}_1 &: \text{ MOA}; \\
\text{MOA}_E &: \text{ AT } \%\text{MX0.0} : \text{ BOOL}; \\
\text{MOA}_T &: \text{ AT } \%\text{MD4001.0} : \text{ TIME} := \text{ t}#1\text{m}; (\text{"60000 ms"}) \\
\text{MOA}_A &: \text{ AT } \%\text{MX0.1} : \text{ BOOL};
\end{align*}
\]

Translation in ABB IL:

\[
\begin{align*}
\text{!BA 0} \\
\text{MOA} \\
\text{E} \\
\text{T} \\
\text{A}
\end{align*}
\]

FBD:

\[
\begin{array}{c}
\text{MOA}_1 \\
\text{MOA}_E := \text{ E} \\
\text{MOA}_T := \text{ T} \\
\text{MOA}_A := \text{ A}
\end{array}
\]

ABB IL of example:

\[
\begin{align*}
\text{!BA 0} \\
\text{MOA} \\
\text{M0,0} \\
\text{KD1,0} & \quad \text{; 60000} \\
\text{M0,1}
\end{align*}
\]

Function call in IL

\[
\begin{align*}
\text{CAL} & \quad \text{MOA}_1(\text{E := MOA}_E, \\
& \quad \text{T := MOA}_T) \\
\text{LD} & \quad \text{MOA}_1.A \\
\text{ST} & \quad \text{MOA}_A
\end{align*}
\]

Note: In IL, the function call has to be performed in one line.

Function call in ST

\[
\begin{align*}
\text{MOA}_1 & \quad (\text{E := MOA}_E, \\
& \quad \text{T := MOA}_T); \\
\text{MOA}_A & \quad := \text{ MOA}_1.A;
\end{align*}
\]
A FALSE/TRUE edge at input E causes a FALSE/TRUE edge at output A. If input E remains at TRUE level, a TRUE/FALSE edge is applied at output A after the duration T has elapsed.

Output ET displays the current time.

Output A is also immediately set back to FALSE level if input E returns to FALSE level before the time period T has expired.

**General behavior**

- Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program. An appropriate message of the operating system is not issued to the affiliated timer block in the PLC program until the timer has elapsed.

- Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:
  - Abort PLC program
  - RUN/STOP switch from RUN -> STOP
  - Warm or cold start

---

**Block type**

Function block with historical values

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>Instance name</td>
</tr>
<tr>
<td>IN</td>
<td>Input signal</td>
</tr>
<tr>
<td>PT</td>
<td>Pulse length</td>
</tr>
<tr>
<td>Q</td>
<td>Output signal, operands M, A (not E, S)</td>
</tr>
<tr>
<td>ET</td>
<td>Time visualization</td>
</tr>
</tbody>
</table>

**Description**

A FALSE/TRUE edge at input E causes a FALSE/TRUE edge at output A. If input E remains at TRUE level, a TRUE/FALSE edge is applied at output A after the duration T has elapsed.

Output A is also immediately set back to FALSE level if input E returns to FALSE level before the time period T has expired.

The output ET indicates the current time.

Valid time range: 5 ms .. 24.8 days.

Maximum time offset at the output: < 1 cycle time

Reasonable range for T: > 1 cycle time

The inputs and the output can neither be duplicated nor inverted.
## Example

### Declaration:

```plaintext
MOAT_1 : MOAT
MOAT_IN AT %MX0.0 : BOOL;
MOAT_PT AT %MD4002.0 : TIME := t#2s500ms; (*2500 ms*)
MOAT_Q AT %MX0.1 : BOOL;
MOAT_ET AT %MD2002.0 : TIME;
```

### Translation in ABB IL:

```plaintext
!BA 0
MOAT
IN
PT
Q
ET
```

### FBD:

```
MOAT_1
MOAT
MOAT_IN IN Q
MOAT_PT PT ET
MOAT_ET
```

### ABB IL of example:

```plaintext
!BA 0
MOAT
M0,0
KD2,0 ; 2500
M0,1
MD2,0
```

### Function call in IL

```plaintext
CAL MOAT_1(E := MOAT_E,
             T := MOAT_T)
LD MOAT_1.A
ST MOAT_A
```

Note: In IL, the function call has to be performed in one line.

### Function call in ST

```plaintext
MOAT_1 (E := MOAT_E,
         T := MOAT_T);
MOAT_A := MOAT_1.A;
```
OPERATION MODE MODBUS MASTER

MODBUS master communication:

**MODUSB**
- COM1 by default
- and DATA is a BOOL

**MODBUSW**
- COM1 by default
- and DATA is an INT

**MODMASTB**
- COM1 or COM2
- and DATA is a BOOL

**MODMASTW**
- COM1 or COM2
- and DATA is an INT

### Block type
Function block with historical values

### Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>MODBUS B/W</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>BOOL</td>
<td>Enabling of the block processing</td>
</tr>
<tr>
<td>COM</td>
<td>INT</td>
<td>Interface identifier (COM1, COM2)</td>
</tr>
<tr>
<td>SLAVE</td>
<td>INT</td>
<td>Slave address (1...255)</td>
</tr>
<tr>
<td>FCT</td>
<td>INT</td>
<td>Function code</td>
</tr>
<tr>
<td>TIMEOUT</td>
<td>INT</td>
<td>Telegram timeout in milliseconds</td>
</tr>
<tr>
<td>ADDR</td>
<td>INT</td>
<td>Operand/register address in the slave</td>
</tr>
<tr>
<td>NB</td>
<td>INT</td>
<td>Number of data</td>
</tr>
<tr>
<td>DATA</td>
<td>BOOL / INT</td>
<td>Data to send to the slave or to write with the data received from a slave MODBUS</td>
</tr>
<tr>
<td>RDY</td>
<td>BOOL</td>
<td>Ready message, communication in progress, operands M, A</td>
</tr>
<tr>
<td>ERR</td>
<td>BOOL</td>
<td>Error message, operands M, A (not E, S)</td>
</tr>
<tr>
<td>ERNO</td>
<td>INT</td>
<td>Error identifier</td>
</tr>
</tbody>
</table>

### Description

MODBUS master communication function block

The central unit is a master on a MODBUS network and can communicate with other products with MODBUS protocol.

The function MODBUS MASTER in the central unit is valided by:

- system constant KW 00,06 / %MW1000.6 = 100
- and connection between pins 7 and 6 on the connector of the serial interface

Several function blocks MODBUS/MODMAST can be used in one user program.

The MODBUS protocol is a master/slave protocol. The master sends a frame to a slave and waits for the answer (a timeout is defined). Binary or numeric data can be read or written in a slave.

The area of data in the master is chosen by the address of the first variable. The size of area is necessary for sending or receiving. Reading or writing data are done automatically from these areas.

**EN**

A rising edge at the FREI input leads to an output of a request to a MODBUS slave, provided that the block is ready to do this (RDY = TRUE).

If a rising edge appears at the FREI input although the RDY output is equal to FALSE, i.e. the block is not ready for a new MODBUS communication, the rising edge will be ignored. Therefore, no new MODBUS communication can be started as long as the RDY output is FALSE.
COM INT
At input COM the MODBUS interface number is specified.
COM = 1: COM1 (PLC)
COM = 2: COM2 (PLC)

SLAV INT
Address of the slave which receives the request.
Value : 0 < ADDR < 255
In case of address 0 (ADDR = 0), all slaves on the MODBUS network will read the frame.

FCT INT
The function depends on the type of parameters and if it is reading or writing.
Value : 1 (01H): reading n bits
2 (02H): reading n words
3 (03H): reading n words
4 (04H): reading n words
5 (05H): writing one bit
6 (06H): writing one word
7 (07H): fast reading of 8 bits
8 (08H): Diagnosis/initialization
15 (0FH): writing n bits
16 (10H): writing n words

The other function codes are not supported by the central units series 40..50. In case of a wrong function code, an error 1 is generated in the word ERNO.

TIME INT
Timeout for the communication (maximum time for an answer of the MODBUS slave).
The value is given in milliseconds.
Cycle time (KD00,00 / %MD4000.0) < TIME < 32767
In case of a timeout, the output ERN provides the value of 9.

ADDR INT
Address of data in the slave memory to read or write.

NB INT
Number of data to read or write in the slave.
This number defines also the size of the data area in the master to send to the slave or to receive from the slave.

DATA INT, BOOL
INT: MODBUSW or MODMASTW
BOOL:
DATA defines the first variable of the data area in the master. The size of this area depends on the NB parameter.

Different cases are possible according to the function code and the operand:

<table>
<thead>
<tr>
<th>ADDR</th>
<th>DATA</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>INT</td>
<td>Idem</td>
</tr>
<tr>
<td>BOOL</td>
<td>BOOL</td>
<td>Idem</td>
</tr>
</tbody>
</table>

Example: data = %MX00.07 :
The first bit of the read word will be written in %MX 00.00.

Writing:

<table>
<thead>
<tr>
<th>ADDR</th>
<th>DATA</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>INT</td>
<td>Idem</td>
</tr>
<tr>
<td>BOOL</td>
<td>BOOL</td>
<td>Idem</td>
</tr>
</tbody>
</table>

RDY BOOL
The output RDY (ready) indicates whether a MODBUS communication is in progress or not. As long as a communication is in progress, the output RDY is equal to FALSE. The function block can only be used if RDY = TRUE.

ERR BOOL
The output ERR indicates an error occurred during communication. The word output ERN indicates the details of the error.
If ERR = 1 -> error,
ERR = 0 -> no error or communication in progress.
The error is clear after one cycle time.

ERNO INT
Details of error:
0 : no error
1 : unknown function code
2 : address error
3 : data error
9 : timeout
10 : checksum error
The error is clear after one cycle time.
<table>
<thead>
<tr>
<th>Operands (symbolic)</th>
<th>Operands (IEC)</th>
<th>MODBUS address (HEX)</th>
<th>Operand description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E000_00</td>
<td>%IX000.00</td>
<td>0000_00</td>
<td>Binary inputs, CS31 bus</td>
</tr>
<tr>
<td>E061_15</td>
<td>%IX061.15</td>
<td>03DF_15</td>
<td></td>
</tr>
<tr>
<td>E062_00</td>
<td>%IX062.00</td>
<td>03E0_00</td>
<td>Binary inputs, local</td>
</tr>
<tr>
<td>E068_15</td>
<td>%IX068.15</td>
<td>044F_15</td>
<td></td>
</tr>
<tr>
<td>A000_00</td>
<td>%QX000.00</td>
<td>1000_00</td>
<td>Binary outputs, CS31 bus</td>
</tr>
<tr>
<td>A061_15</td>
<td>%QX061.15</td>
<td>13DF_15</td>
<td></td>
</tr>
<tr>
<td>A062_00</td>
<td>%QX062.00</td>
<td>13E0_00</td>
<td>Binary outputs, local</td>
</tr>
<tr>
<td>A068_15</td>
<td>%QX068.15</td>
<td>144F_15</td>
<td></td>
</tr>
<tr>
<td>M000_00</td>
<td>%MX0000.00</td>
<td>2000_00</td>
<td>Binary flags</td>
</tr>
<tr>
<td>M099_15</td>
<td>%MX0099.15</td>
<td>263F_15</td>
<td></td>
</tr>
<tr>
<td>M230_00</td>
<td>%MX0000.00</td>
<td>2000_00</td>
<td>Binary flags</td>
</tr>
<tr>
<td>M254_15</td>
<td>%MX0254.15</td>
<td>2FF6_15</td>
<td></td>
</tr>
<tr>
<td>M255_00</td>
<td>%MX0255.00</td>
<td>2FF0_00</td>
<td>Binary flags (system)</td>
</tr>
<tr>
<td>M255_15</td>
<td>%MX0255.15</td>
<td>2FFF_15</td>
<td></td>
</tr>
<tr>
<td>S000_00</td>
<td>%MX5000.00</td>
<td>3000_00</td>
<td>Steps</td>
</tr>
<tr>
<td>S125_15</td>
<td>%MX5125.15</td>
<td>37DF_15</td>
<td></td>
</tr>
<tr>
<td>EW000_00</td>
<td>%IW1000.00</td>
<td>0000_00</td>
<td>Analog inputs, CS31 bus</td>
</tr>
<tr>
<td>EW061_15</td>
<td>%IW1061.15</td>
<td>03DF_15</td>
<td></td>
</tr>
<tr>
<td>EW062_00</td>
<td>%IW1062.00</td>
<td>03E0_00</td>
<td>Analog inputs, local</td>
</tr>
<tr>
<td>EW068_15</td>
<td>%IW1068.15</td>
<td>044F_15</td>
<td></td>
</tr>
<tr>
<td>AW000_00</td>
<td>%QW1000.00</td>
<td>1000_00</td>
<td>Analog outputs, CS31 bus</td>
</tr>
<tr>
<td>AW061_15</td>
<td>%QW1061.15</td>
<td>13DF_15</td>
<td></td>
</tr>
<tr>
<td>AW062_00</td>
<td>%QW1062.00</td>
<td>13E0_00</td>
<td>Analog outputs, local</td>
</tr>
<tr>
<td>AW068_15</td>
<td>%QW1068.15</td>
<td>144F_15</td>
<td></td>
</tr>
<tr>
<td>Operands (symbolic)</td>
<td>Operands (IEC)</td>
<td>MODBUS address (HEX)</td>
<td>Operand description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td>----------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>MW000.00</td>
<td>%MW1000.00</td>
<td>2000&lt;sub&gt;hex&lt;/sub&gt;</td>
<td>Word flags</td>
</tr>
<tr>
<td>MW099.15</td>
<td>%MW1099.15</td>
<td>263F&lt;sub&gt;hex&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>MW230.00</td>
<td>%MW1230.00</td>
<td>2E60&lt;sub&gt;hex&lt;/sub&gt;</td>
<td>Word flags</td>
</tr>
<tr>
<td>MW253.15</td>
<td>%MW1253.15</td>
<td>2FCF&lt;sub&gt;hex&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>MW254.00</td>
<td>%MW1254.00</td>
<td>2FD0&lt;sub&gt;hex&lt;/sub&gt;</td>
<td>Word flags (error message)</td>
</tr>
<tr>
<td>MW255.15</td>
<td>%MW1255.15</td>
<td>2FFF&lt;sub&gt;hex&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>KW000.00</td>
<td>%MW3000.00</td>
<td>3000&lt;sub&gt;hex&lt;/sub&gt;</td>
<td>Word constants (system)</td>
</tr>
<tr>
<td>KW000.15</td>
<td>%MW3000.15</td>
<td>300F&lt;sub&gt;hex&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>KW001.00</td>
<td>%MW3001.00</td>
<td>3010&lt;sub&gt;hex&lt;/sub&gt;</td>
<td>Word constants</td>
</tr>
<tr>
<td>KW031.15</td>
<td>%MW3031.15</td>
<td>31FF&lt;sub&gt;hex&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>MD000.00</td>
<td>%MD2000.00</td>
<td>4000&lt;sub&gt;hex&lt;/sub&gt;</td>
<td>Double word flags</td>
</tr>
<tr>
<td>MD007.15</td>
<td>%MD2007.15</td>
<td>40FE&lt;sub&gt;hex&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>KD000.00</td>
<td>%MD4000.00</td>
<td>5000&lt;sub&gt;hex&lt;/sub&gt;</td>
<td>Double word constants</td>
</tr>
<tr>
<td>KD000.15</td>
<td>%MD4000.15</td>
<td>501E&lt;sub&gt;hex&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>KD001.00</td>
<td>%MD4000.00</td>
<td>5020&lt;sub&gt;hex&lt;/sub&gt;</td>
<td>Double word constants</td>
</tr>
<tr>
<td>KD007.15</td>
<td>%MD4007.15</td>
<td>50FE&lt;sub&gt;hex&lt;/sub&gt;</td>
<td></td>
</tr>
</tbody>
</table>
Example 1: MODBUS – MODBUS master only for COM1 and BOOL - data

Declaration:

MOD_1: MODBUS;
MOD_EN AT %MX13.0: BOOL;
MOD_SLAV AT %MW3002.1: INT := 10;
MOD_FCT AT %MW3002.2: INT := 3;
MOD_TIMEOUT AT %MW3002.3: INT := 5000;
MOD_ADDR AT %MW3002.4: INT := 16#2000; (*8192*)
MOD_NB AT %MW3002.5: INT := 16;
MOD_DATA_B AT %MX40.0: BOOL;
MOD_RDY AT %MX30.0: BOOL;
MOD_ERR AT %MX30.1: BOOL;
MOD_ERNO AT %MW1039.0: INT;

Translation in ABB IL:

!BA 0
CALLUP
EN
#H0300
#H0000
#H0000
#00009
SLAV
FCT
TIMEOUT
ADDR
NB
DATA
RDY
ERR
ERNO

FBD:

MOD_1
MODBUS

MOD_EN
MOD_SLAV
MOD_FCT
MOD_TIMEOUT
MOD_ADDR
MOD_NB
MOD_DATA_B

MOD_RDY
MOD_ERR
MOD_ERNO

ABB IL of example:

!BA 0
CALLUP
M0,0
#H0300
#H0000
#H0000
#00009
KW2,1 ; 10
KW2,2 ; 3
KW2,3 ; 5000
KW2,4 ; 8192
KW2,5 ; 16
M40,0
M30,0
M30,1
MW39,0

Function call in IL

CAL MOD_1(EN := MOD_EN,
SLAV := MOD_SLAV,
FCT := MOD_FCT,
TIMEOUT := MOD_TIMEOUT,
ADDR := MOD_ADDR,
NB := MOD_NB,
DATA := MOD_DATA_B)

LD MOD_1.RDY
ST MOD_RDY
LD MOD_1. ERR
ST MOD_ERR
LD MOD_1.ERNO
ST MOD_ERNO

Function call in ST

MOD_1 (EN := MOD_EN,
SLAV := MOD_SLAV,
FCT := MOD_FCT,
TIMEOUT := MOD_TIMEOUT,
ADDR := MOD_ADDR,
NB := MOD_NB,
DATA := MOD_DATA_B)

MOD_RDY := MOD_1.RDY;
MOD_ERR := MOD_1. ERR;
MOD_ERNO := MOD_1.ERNO;
Example 2: MODBUSW – MODBUS master only for COM1 and INT - data

Declaration:

MOD_1: MODBUSW;
MOD_EN AT %MX13.0: BOOL;
MOD_SLAV AT %MW3002.1: INT := 10;
MOD_FCT AT %MW3002.2: INT := 3;
MOD_TIMEOUT AT %MW3002.3: INT := 5000;
MOD_ADDR AT %MW3002.4: INT := 16#2000; (*8192*)
MOD_NB AT %MW3002.5: INT := 16;
MOD_DATA AT %MW1030.0: INT;
MOD_RDY AT %MX30.0: BOOL;
MOD_ERR AT %MX30.1: BOOL;
MOD_ERNO AT %MW1039.0: INT;

Translation in ABB IL:

!BA 0
CALLUP
EN
#H0300
#H0000
#H0000
#00009
SLAV
FCT
TIMEOUT
ADDR
NB
DATA
RDY
ERR
ERNO

FBD:

MOD_1
MODBUSW
MOD_EN
MOD_SLAV
MOD_FCT
MOD_TIMEOUT
MOD_ADDR
MOD_NB
MOD_DATA
MOD_RDY
MOD_ERR
MOD_ERNO

ABB IL of example:

!BA 0
CALLUP
M0.0
#H0300
#H0000
#H0000
#00009
KW2.1 ; 10
KW2.2 ; 3
KW2.3 ; 5000
KW2.4 ; 8192
KW2.5 ; 16
MW30.0
M30.0
M30.1
MW39.0

Function call in IL

CAL MOD_1(EN := MOD_EN,
SLAV := MOD_SLAV,
FCT := MOD_FCT,
TIMEOUT := MOD_TIMEOUT,
ADDR := MOD_ADDR,
NB := MOD_NB,
DATA := MOD_DATA)

LD MOD_1.RDY
ST MOD_RDY
LD MOD_1.ERR
ST MOD_ERR
LD MOD_1.ERNO
ST MOD_ERNO

Function call in ST

MOD_1 (EN := MOD_EN,
SLAV := MOD_SLAV,
FCT := MOD_FCT,
TIMEOUT := MOD_TIMEOUT,
ADDR := MOD_ADDR,
NB := MOD_NB,
DATA := MOD_DATA)

MOD_RDY := MOD_1.RDY;
MOD_ERR := MOD_1.ERR;
MOD_ERNO := MOD_1.ERNO;
Example 3: MODMASTB – MODBUS master for COM1/COM2 and BOOL - data

Declaration:

MOD_1: MODMASTB;
MOD_EN AT %MX13.0: BOOL;
MOD_COM AT %MW3002.0: INT := 1;
MOD_SLAV AT %MW3002.1: INT := 10;
MOD_FCT AT %MW3002.2: INT := 3;
MOD_TIMEOUT AT %MW3002.3: INT := 5000;
MOD_ADDR AT %MW3002.4: INT := 16#2000; (*8192*)
MOD_NB AT %MW3002.5: INT := 16;
MOD_DATA_B AT %MX40.0: INT;
MOD_RDY AT %MX30.0: BOOL;
MOD_ERR AT %MX30.1: BOOL;
MOD_ERNO AT %MW1039.0: INT;

Translation in ABB IL:

!BA 0
CALLUP
EN
#H0306
#H0000
#H0000
#00010
COM
SLAV
FCT
TIMEOUT
ADDR
NB
DATA
RDY
ERR
ERNO

FBD:

Function call in IL

CAL  MOD_1(EN := MOD_EN,
COM := MOD_COM,
SLAV := MOD_SLAV,
FCT := MOD_FCT,
TIMEOUT := MOD_TIMEOUT,
ADDR := MOD_ADDR,
NB := MOD_NB,
DATA := MOD_DATA_B)
LD  MOD_1.RDY
ST  MOD_RDY
LD  MOD_1.ERR
ST  MOD_ERR
LD  MOD_1.ERNO
ST  MOD_ERNO

Function call in ST

MOD_1  (EN := MOD_EN,
COM := MOD_COM,
SLAV := MOD_SLAV,
FCT := MOD_FCT,
TIMEOUT := MOD_TIMEOUT,
ADDR := MOD_ADDR,
NB := MOD_NB,
DATA := MOD_DATA_B)
MOD_RDY := MOD_1.RDY;
MOD_ERR := MOD_1.ERR;
MOD_ERNO := MOD_1.ERNO;
Example 4: MODMASTW – MODBUS master for COM1/COM2 and INT - data

**Declaration:**

MOD_1: MODMASTW;
MOD_EN AT %MX13.0: BOOL;
MOD_COM AT %MW3002.0: INT := 1;
MOD_SLAV AT %MW3002.1: INT := 10;
MOD_FCT AT %MW3002.2: INT := 3;
MOD_TIMEOUT AT %MW3002.3: INT := 5000;
MOD_ADDR AT %MW3002.4: INT := 16#2000; (*8192*)
MOD_NB AT %MW3002.5: INT := 16;
MOD_DATA AT %MW1030.0: INT;
MOD_RDY AT %MX30.0: BOOL;
MOD_ERR AT %MX30.1: BOOL;
MOD_ERNO AT %MW1039.0: INT;

**Translation in ABB IL:**

!BA 0
CALLUP
M0,0
#H0306
#H0000
#00010
#H0000
#H0306
#H0000
#00010
KW2,0 ; 1
KW2,1 ; 10
KW2,2 ; 3
KW2,3 ; 5000
KW2,4 ; 8192
KW2,5 ; 16
MW30,0
M30,0
M30,1
MW39,0

**ABB IL of example:**

!BA 0
CALLUP
MOD_1 (EN := MOD_EN,
COM := MOD_COM,
SLAV := MOD_SLAV,
FCT := MOD_FCT,
TIMEOUT := MOD_TIMEOUT,
ADDR := MOD_ADDR,
NB := MOD_NB,
DATA := MOD_DATA)

LD MOD_1.RDY
ST MOD_RDY
LD MOD_1.ERR
ST MOD_ERR
LD MOD_1.ERNO
ST MOD_ERNO

**Function call in ST**

MOD_1 (EN := MOD_EN,
COM := MOD_COM,
SLAV := MOD_SLAV,
FCT := MOD_FCT,
TIMEOUT := MOD_TIMEOUT,
ADDR := MOD_ADDR,
NB := MOD_NB,
DATA := MOD_DATA)

MOD_RDY := MOD_1.RDY;
MOD_ERR := MOD_1.ERR;
MOD_ERNO := MOD_1.ERNO;
MONOSTABLE ELEMENT »CONSTANT«

A FALSE/TRUE edge at input E causes a FALSE/TRUE edge at output A. If input E remains at TRUE level, a TRUE/FALSE edge is applied at output A after the duration T has elapsed.

A possible second FALSE/TRUE edge of input E which occurs before the time period T has elapsed is ignored.

Maximum time offset at the output: < 1 cycle time

Block type

Function block with historical values

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>MOK</td>
<td>Instance name</td>
</tr>
<tr>
<td>E</td>
<td>BOOL</td>
<td>Input signal</td>
</tr>
<tr>
<td>T</td>
<td>TIMER</td>
<td>Pulse length</td>
</tr>
<tr>
<td>A</td>
<td>BOOL</td>
<td>Output signal, operands M, A (not E, S)</td>
</tr>
</tbody>
</table>

Description

A FALSE/TRUE edge at input E causes a FALSE/TRUE edge at output A. If input E remains at TRUE level, a TRUE/FALSE edge is applied at output A after the duration T has elapsed.

A possible second FALSE/TRUE edge of input E which occurs before the time period T has elapsed is ignored.

Valid time range: 5 ms .. 24,8 days

Maximum time offset at the output: < 1 cycle time

Reasonable range for T: > 1 cycle time

The inputs and the output can neither be duplicated nor inverted.

General behavior

- Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program. An appropriate message of the operating system is not issued to the affiliated timer block in the PLC program until the timer has elapsed.

- Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:
  - Abort PLC program
  - RUN/STOP switch from RUN -> STOP
  - Warm or cold start

![Diagram](image)
Example

Declaration:

```
MOK_1 : MOK;
MOK_E AT %MX0.0 : BOOL;
MOK_T AT %MD4002.1 : TIME := 16#1s; (*1000 ms*)
MOK_A AT %MX0.1 : BOOL;
```

Translation in ABB IL:

```
!BA 0
MOK
E
T
A
```

FBD:

```
 MOK_1

 MOK_E=E
 MOK_T=T
```

ABB IL of example:

```
!BA 0
MOK
M0,0
KD2,1 ; 1000
M0,1
```

Function call in IL

```
CAL MOK_1(E := MOK_E,
          T := MOK_T)
LD MOK_1.A
ST MOK_A
```

Note: In IL, the function call has to be performed in one line.

Function call in ST

```
MOK_1 (E := MOK_E,
       T := MOK_T);
MOK_A := MOK_1.A;
```
MULTIPLICATION BY 2 TO THE POWER OF N, WORD

The value of the operand at input E1 is shifted bitwise N times.

If the value at input N is positive, the value is shifted to the left. Each shift by 1 bit position corresponds to a multiplication of the current value by 2.

If the value at input N is negative, the value is shifted to the right. Each shift by 1 bit position corresponds to a division of the current value by 2.

The result is assigned to the operand at output A1.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1 INT</td>
<td>Input operand</td>
</tr>
<tr>
<td>N INT</td>
<td>Quantity</td>
</tr>
<tr>
<td>A1 INT</td>
<td>Result</td>
</tr>
</tbody>
</table>

Description

The value of the operand at input E1 is shifted bitwise N times.

If the value at input N is positive, the value is shifted to the left. Each shift by 1 bit position corresponds to a multiplication of the current value by 2.

If the value at input N is negative, the value is shifted to the right. Each shift by 1 bit position corresponds to a division of the current value by 2.

The result is assigned to the operand at output A1.

The inputs and the output can neither be duplicated nor negated.

Reasonable range for N: -14 ≤ N ≤ +14

If N = 0, the value at input E1 is passed directly to output A1.

Sign of the value at input E1:

The sign of value E1 is not influenced by the shift operation. I.e. the sign of the output value is always identical with the sign of the input value.

Shift to the left (Multiplication):

When the value at the input is shifted to the left, the released bit 0 is filled with 0. The sign bit (bit 15) is not changed because a limiting to the limit of the number range is performed before.

Limiting the value at output A1 when shifting to the left:

- The following applies to positive values at input E1: If bit 14 has a »1« and if shift operations still have to be carried out on the basis of the value at input N, these are no longer executed. Instead, the output is set to the limit of the positive number range. I.e. the limit has been reached in any case at the latest after 14 shifts.
  Limit value: Output A1 = +32767 (7FFFH).

- The following applies to negative values at input E1: If bit 14 has a »0« and if shift operations still have to be carried out on the basis of the value at input N, these are no longer executed. Instead, the output is set to the limit of the negative number range. I.e. the limit has been reached in any case at the latest after 14 shifts.
Shift to the right (Division):

When shifting to the right, every bit moves to the right by one position. At the same time, the sign bit (bit 15) always retains its value. The released bit (bit 14) is filled in each case with the value of the sign bit.

Limiting the value at the output when shifting to the right:

- The following applies to positive values at input E1:
  If now only bit 0 has a »1« and shift operations still have to be carried out because of the value at input N, the output will be set to the value 0. I.e. value 0 has been reached in any case at the latest after 14 shifts.

  Output A1 = 0.

- The following applies to negative values at input E1:
  If bit 0 ... bit 15 has a »1« as the result of the shift, the limit value (-1) has been reached. Further shifts have no effect. I.e. the value -1 has been reached at the latest after 15 shifts.


The inputs and the output can neither be duplicated nor negated.

**Examples**

1. Input value E1 = 5498 (157A<sub>H</sub>)
   Exponent N = 2 -> 2 * Left shift

   ![Diagram 1]
   
   Input value before shifting left
   Result after shifting left twice

2. Input value E1 = 32612 (7F64<sub>H</sub>)
   Exponent N = -3 -> 3 * Right shift

   ![Diagram 2]
   
   Input value before shifting right
   Result after shifting right three times

3. Input value E1 = -32612 (8008<sub>H</sub>)
   Exponent N = -3 -> 3 * Right shift

   ![Diagram 3]
   
   Input value before shifting right
   Result after shifting right three times

**Translation in ABB IL:**

```plaintext
!BA 0
MUL2N
E1
N
A
```

**ABB IL of example:**

```plaintext
!BA 0
MUL2N
MW0,0
MW0,1
MW0,2
```

**Function call in IL**

```plaintext
LD MUL2N_E1
MUL2N MUL2N_N
ST MUL2N_A
```

**Function call in ST**

```plaintext
MUL2N_A:=MUL2N(MUL2N_E1, MUL2N_N);
```
MULTIPLICATION DOUBLE WORD

The value of the operand at input E1 is multiplied by the value of the operand at input E2 and the result is assigned to the operand at output A.

The result is limited to the maximum or minimum value of the number range. If limiting occurred, a TRUE signal is assigned to the binary operand at output Q. If no limiting has occurred, a FALSE signal is assigned to the binary operand at output Q.

Block type
- Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>MULD</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>DINT</td>
<td>Multiplicand</td>
</tr>
<tr>
<td>E2</td>
<td>DINT</td>
<td>Multiplier</td>
</tr>
<tr>
<td>A</td>
<td>DINT</td>
<td>Result (product)</td>
</tr>
<tr>
<td>Q</td>
<td>BOOL</td>
<td>Result limited, operands M, A (not E, S)</td>
</tr>
</tbody>
</table>

Description

The value of the operand at input E1 is multiplied by the value of the operand at input E2 and the result is assigned to the operand at output A.

The result is limited to the maximum or minimum value of the number range (Number range: \(-2147483647 \ldots 2147483647\)). If limiting occurred, a TRUE signal is assigned to the binary operand at output Q. If no limiting occurred, a FALSE signal is assigned to the binary operand at output Q.

The inputs and outputs can neither be duplicated nor negated.
Example

Declaration:

MULD_1 : MULD;
MULD_E1 AT %MD2000.0 : DINT;
MULD_E2 AT %MD2000.1 : DINT;
MULD_A AT %MD2000.2 : DINT;
MULD_Q AT %M0.0 : BOOL;

Translation in ABB IL:

!BA 0
MULD
E1
E2
A
Q

ABB IL of example:

!BA 0
MULD
MD0.0
MD0.1
MD0.2
M0.0

FBD:

```
MULD_1
MULD_E1-E1 A-------------------------MULD_A
MULD_E2-E2 Q-------------------------MULD_Q
```

Function call in IL

CAL MULD_1(E1 := MULD_E1,
E2 := MULD_E2)
LD MULD_1.Q
ST MULD_1.Q
LD MULD_1.A
ST MULD_A

Note: In IL, the function call has to be performed in one line.

Function call in ST

MULD_1 (E1 := MULD_E1,
E2 := MULD_E2);
MULD_A := MULD_1.A;
MULD_Q := MULD_1.Q;
MULTIPLICATION WITH DIVISION

The operand value at input E1 is multiplied by the operand value at input E2, the intermediate result is divided by the operand value at E3 and then the result is assigned to output A.

The result is limited to the maximum or minimum value of the number range.

Block type

| Function |

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>INT</td>
<td>Multiplicand</td>
</tr>
<tr>
<td>E2</td>
<td>INT</td>
<td>Multiplier</td>
</tr>
<tr>
<td>E3</td>
<td>INT</td>
<td>Divisor</td>
</tr>
<tr>
<td>A</td>
<td>INT</td>
<td>Result</td>
</tr>
</tbody>
</table>

Description

The operand value at input E1 is multiplied by the operand value at input E2, the intermediate result is divided by the operand value at E3 and then the result is assigned to output A.

Internal, the block performs the multiplication and division with the accuracy of a double word (32 bit). Only when assigning the result to output A, the limiting to the accuracy of a word (16 bit) is carried out. If the remainder of the division is > 0.5, the result is rounded up. If a numerical overflow occurs during the division (e.g. division by zero), the correct signed limit of the number range is applied at output A.

The result is limited to the maximum value 32767 and the minimum value -32767.

The inputs and the output can neither be duplicated nor negated.
Example

Declaration:
MULDI_E1 AT %MW1000.0 : INT;
MULDI_E2 AT %MW1000.1 : INT;
MULDI_E3 AT %MW1000.2 : INT;
MULDI_A AT %MW1000.3 : INT;

Translation in ABB IL:
!BA 0
MULDI
E1
E2
E3
A

FBD:

```
  MULDI
   E1
   E2
   E3
```

ABB IL of example:
!BA 0
MULDI
MW0,0
MW0,1
MW0,2
MW0,3

Function call in IL
LD MULDI_E1
MULDI MULDI_E2,MULDI_E3
ST MULDI_A

Function call in ST
MULDI_A := MULDI (MULDI_E1,
                      MULDI_E2,
                      MULDI_E3);
NEGATION WORD

NEGATION WORD

The value of the operand at input E is negated and the result is assigned to the operand at output A.

Block type
Function

Parameters

<table>
<thead>
<tr>
<th>E1</th>
<th>INT</th>
<th>Input value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>INT</td>
<td>Negated value</td>
</tr>
</tbody>
</table>

Description
The value of the operand at input E is negated and the result is assigned to the operand at output A

The input and the output can neither be duplicated nor negated.

Number range:
Low limit: 8000_{H} - 32768
High limit: 7FFF_{H} + 32767
Note: if E1 = -32768 then A = +32767.

Example

Declaration:
NEGW_E1 AT %MW1000.0 : INT;
NEGW_A AT %MW1000.1 : INT;

Translation in ABB IL:
IBA 0
NEGW
E1
A

ABB IL of example:
IBA 0
NEGW
MW0.0
MW0.1

Function call in IL
LD NEGW_E1
NEGW
ST NEGW_A

Function call in ST
NEGW_A := NEGW(NEGW_E1);
PULSES GENERATOR

The function block generates pulse sequences with specific frequencies and specific number of pulses. Thus, it can be used for the activation of stepper motors.

The pulse sequences are available at output A62.00 / %QX62.00.

**Block type**

Function block with historical values

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>NPULSE</td>
</tr>
<tr>
<td>VAL</td>
<td>BOOL</td>
</tr>
<tr>
<td>RESET</td>
<td>BOOL</td>
</tr>
<tr>
<td>FREQ</td>
<td>INT</td>
</tr>
<tr>
<td>NB</td>
<td>INT</td>
</tr>
<tr>
<td>RDY</td>
<td>BOOL</td>
</tr>
<tr>
<td>VAL_P</td>
<td>INT</td>
</tr>
</tbody>
</table>

**Description**

The function block generates pulse sequences with specific frequencies and specific number of pulses. Thus, it can be used for the activation of stepper motors.

The pulse sequences are available at output A62.00 / %QX62.00.

At output VAL_P, the currently generated number of pulses is displayed. The rising edge at the input VAL starts the pulse generator from the beginning. The first period starts with a high signal.

**VAL**

Enabling the output of the impulse sequence at the fixed output A62.00 / %QX62.0.

**RESET**

The pulse mode is validated. A62.00 / %QX62.00 can not be used normally.

**FREQ**

Frequency of the pulse sequence:

Pulse frequency: 10 Hz to 2,604 kHz

Formula for frequency calculation:

\[
\text{Frequency [Hz]} = \frac{1000000}{(256 - \text{FREQ}) \times 384}
\]

**NB**

Number of pulses to be generated.

If NB < 0 then the function block NPULSE generates continuously pulses till RESET input is validated.

**Reset = FALSE** stops the pulse generator, the output A62.00 / %QX62.0 is set to FALSE till RESET = TRUE.
RDY       BOOL
Ready bit
RDY = FALSE  Pulses are generated
RDY = TRUE   Pulse sequence is finished

VAL_P       INT
Number of pulses elapsed.
This number is estimated by the function block every cycle time. It doesn't represent the exact value.
The internal memory bit %QX62.00 is disabled during pulses are generating.

Example

Declaration:
NPULSE1 : NPULSE;
NPULSE_VAL AT %MX0.0 : BOOL;
NPULSE_RESET AT %MX0.1 : BOOL;
NPULSE_FREQ AT %MW1000.0 : INT;
NPULSE_NB AT %MW1000.1 : INT;
NPULSE_RDY AT %MX0.2 : BOOL;
NPULSE_VAL_P AT %MW1000.2 : INT;

Translation in ABB IL:
!BA 0
NPULSE
VAL
RESET
FREQ
NB
RDY
VAL_P

FBD:

Function call in IL
CAL NPULSE1(VAL := NPULSE_VAL,
RESET := NPULSE_RESET,
FREQ := NPULSE_FREQ,
NB := NPULSE_NB)
LD NPULSE1.RDY
ST NPULSE_RDY
LD NPULSE1.VAL_P
ST NPULSE_VAL_P

Function call in ST
NPULSE1  (VAL := NPULSE_VAL,
RESET := NPULSE_RESET,
FREQ := NPULSE_FREQ,
NB := NPULSE_NB);
NPULSE_RDY  := NPULSE1.RDY;
NPULSE_VAL_P  := NPULSE1.VAL_P;

Note: In IL, the function call has to be performed in one line.
PACK BINARY VARIABLES IN WORD

This block packs n binary variables into one word variable.

Block type
Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>PACK</td>
<td>Instance name</td>
</tr>
<tr>
<td>N</td>
<td>INT</td>
<td>Number of binary variables to be packed</td>
</tr>
<tr>
<td>B0..B15</td>
<td>BOOL</td>
<td>Binary variable to be packed</td>
</tr>
<tr>
<td>A</td>
<td>INT</td>
<td>Word variable</td>
</tr>
</tbody>
</table>

Description

The PACK number indicates the maximum number of bits to pack. The following PACK blocks are available:

- PACK4: PACK for max. 2 binary variables
- PACK8: PACK for max. 8 binary variables
- PACK16: PACK for max. 16 binary variables

This block packs n binary variables into one word variable.

At input n, the number of binary variables to be packed is specified.

The following applies:

- $1 < n < 4, (8, 16)$
- $n = 0$ is not allowed! (default = 1)

B0...B15

The binary variables to be packed are specified at the inputs B0 ... Bn-1.

A

The value of each binary variable at the inputs B0 ... Bn-1 is loaded into the corresponding bit (bit 0 ... bit 15) of the variable at output A.

Assignment:

- $B0 \rightarrow \text{bit0 of the output variable}$
- $B1 \rightarrow \text{bit1 of the output variable}$
- $\ldots$
- $B15 \rightarrow \text{bit15 of the output variable}$

Note:
If the user plans less than 16 binary input variables, the unnecessary bits of the output variable are set to the value FALSE.
Example

Declaration:

PACK_16 : PACK;
PACK_B0 AT %MX0.0 : BOOL;
PACK_B1 AT %MX0.1 : BOOL;
PACK_B2 AT %MX0.2 : BOOL;
PACK_B3 AT %MX0.3 : BOOL;
PACK_A AT %MW1000.0 : INT;

Translation in ABB IL:

!BA 0
PACK
#n
B0
B1
B2
B3
A

ABB IL of example:

!BA 0
PACK
#4
M0,0
M0,1
M0,2
M0,3
MW0,0

Function call in IL

CAL PACK_1(n := 4,
    B0 := PACK_B0,
    B1 := PACK_B1,
    B2 := PACK_B2,
    B3 := PACK_B3)
LD PACK_1.A
ST PACK_A

Function call in ST

PACK_1 (n := 16,
    B0 := PACK_B0,
    B1 := PACK_B1,
    B2 := PACK_B2,
    B3 := PACK_B3);
PACK_A:=PACK_1.A;

Note: In IL, the function call has to be performed in one line.
PULSE DURATION MODULATOR

This function block generates a pulse duration-modulated binary signal at its PULS output.

The pulse duty ratio is specified at the t_TA input and the period duration for the output signal is specified at the TA_T input.

Block type

Function block with historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>PDM</th>
<th>Pulse duty name</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_TA</td>
<td>INT</td>
<td>Pulse duty ratio</td>
</tr>
<tr>
<td>TA_T</td>
<td>INT</td>
<td>Period duration referred to the cycle time</td>
</tr>
<tr>
<td>PULS</td>
<td>BOOL</td>
<td>Pulse duration modulated signal</td>
</tr>
</tbody>
</table>

Description

This function block generates a pulse duration-modulated binary signal at its PULS output.

The pulse duty ratio is specified at the t_TA input and the period duration for the output signal is specified at the TA_T input.

The inputs and the output can neither be duplicated nor negated nor inverted.

**t_TA** INT

The required pulse duty ratio for the output signal PULS is specified at input t_TA. At the same time, TA is the period duration of the signal at output PULS and t is the time within the period duration TA during which the output signal assumes a TRUE level. The specified value for the required pulse duty ratio at input t_TA must be specified in scaled form. To do this, the required pulse duty ratio must be multiplied by the value 32767 and rounded to an integer number. The resulting numerical value is then specified at input t_TA.

Boundary condition for t: t > T

I.e. the required duty cycle of the output signal must be higher than the cycle time of the PLC program (T).

**TA_T** INT

The required period duration TA for the signal at output PULS is specified at input TA_T. At the same time, the period duration TA must be scaled to the cycle time T.

Boundary condition for TA:

- TA must be an integer multiple of T (TA = n * T)
- TA >> T > 0; the higher TA is in relation to T, the more exactly the required pulse duty ratio is kept

Example:

TA > 10 * T → inaccuracy of the pulse duty ratio at output PULS < 10%.

If a value TA_T < 0 is specified for TA_T, the function block automatically replaces this meaningless value by 32767.
The pulse duration modulated signal is available at output PULS.

Combination of the PDM block with a controller

If the PDM function block is connected to the output of a controller in order to realize a »switching« controller, the following boundary conditions apply:

- Period duration $T_A$ of the PDM = sampling time of the controller.
- Period of the pulse signal must be synchronous with the period of the controller sampling time.

These boundary conditions are fulfilled by planning the controller in the same PLC program as the PDM, but within one run number block. By means of the run number block, the sampling time of the controller is prolonged by an integer multiple of the cycle time. Therefore, the controller is processed less frequently within the run number block than the PDM outside of the run number block.

Example

**Declaration:**

```
PDM_1 : PDM;
PDM_t_TA AT %MW1000.0 : INT;
PDM_TA_t AT %MW1000.1 : INT;
PDM_PULS AT %MX0.0 : BOOL;
```

**FBD:**

```
PDM_1  
PDM
PDM_t_TA AT T_TA  PULS
PDM_TA_t AT T_A_T  PDM_PULS
```

**Translation in ABB IL:**

```
!BA 0
PDM
PDM_t_TA
PDM_TA_t
PDM_PULS
```

**ABB IL of example:**

```
!BA 0
PDM
MW0,0
MW0,1
M0,0
```

**Function call in IL**

```
CAL PDM_1(t_TA := PDM_t_TA, TA_T := PDM_TA_t)
LD PDM_1,PULS
ST PDM_PULS
```

Note: In IL, the function call has to be performed in one line.

**Function call in ST**

```
PDM_1 (t_TA := PDM_t_TA, TA_T := PDM_TA_t);
PDM_PULS:=PDM_1.PULS;
```
PROPORTIONAL INTEGRAL CONTROLLER

The PI controller changes the value at its output Y (manipulated variable) until the value at input X (controlled variable) is equal to the value at input W (command variable).

### Block type
Function block with historical values

### Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>PI</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>INT</td>
<td>Command variable (set point)</td>
</tr>
<tr>
<td>X</td>
<td>INT</td>
<td>Controlled variable (actual value)</td>
</tr>
<tr>
<td>KP</td>
<td>INT</td>
<td>Proportional coefficient, specified as a percentage value</td>
</tr>
<tr>
<td>TN_TZ</td>
<td>INT</td>
<td>Integral action time scaled to the PLC cycle time</td>
</tr>
<tr>
<td>OG</td>
<td>INT</td>
<td>High limit for the manipulated variable Y</td>
</tr>
<tr>
<td>UG</td>
<td>INT</td>
<td>Low limit for the manipulated variable Y</td>
</tr>
<tr>
<td>SET</td>
<td>BOOL</td>
<td>Enabling for setting the manipulated variable Y to the initial value INIT</td>
</tr>
<tr>
<td>INIT</td>
<td>INT</td>
<td>Initial value for the manipulated variable Y</td>
</tr>
<tr>
<td>RES</td>
<td>BOOL</td>
<td>Reset of the manipulated variable Y to 0</td>
</tr>
<tr>
<td>OG_MELD</td>
<td>BOOL</td>
<td>High limit has been reached, operands M, A (not E, S)</td>
</tr>
<tr>
<td>UG_MELD</td>
<td>BOOL</td>
<td>Low limit has been reached, operands M, A (not E, S)</td>
</tr>
<tr>
<td>Y</td>
<td>INT</td>
<td>Output for the manipulated variable Y</td>
</tr>
</tbody>
</table>

### Description

The PI controller changes the value at its output Y (manipulated variable) until the value at input X (controlled variable) is equal to the value at input W (command variable).

Control algorithm: Simple rectangle rule

\[
Y = \frac{KP}{100} \cdot \frac{W-X}{100} + YI(z-1) + \frac{KP}{100} \cdot (W-X)
\]

With: \(YI(z-1)\) is the integral component from the previous program cycle

Transfer function:

\[
F(s) = KP \cdot \left(1 + \frac{1}{s \cdot TN}\right)
\]

The inputs and outputs can neither be duplicated nor negated/inverted.

**PI controller:** Surge-free transition from the specified value to control operation

**PI controller:** Surging transition from the specified initial value to control operation
The command variable (set point) is specified at input W.

The controlled variable (actual value) is specified at input X.

The proportional coefficient is specified at input KP. This value is specified as a percentage value. Positive or negative values are possible.

Example:
1 = 1 percent
55 = 55 percent
100 = 100 percent
1000 = 1000 percent
-500 = -500 percent

1 percent means that the block multiplies the system deviation by the factor 0.01 (see also control algorithm)

100 percent means that the block multiplies the system deviation by the factor 1 (see also control algorithm)

1000 percent means that the block multiplies the system deviation by the factor 10 (see also control algorithm)

Generally, proportional coefficients of more than 1000 percent are not meaningful in control systems.

The integral action time TN is scaled to the PLC cycle time TZ and is specified at input TN_TZ.

Value range: 0 ≤ TN_TZ ≤ 328

If values are specified which are not within the admissible value range the PLC generally uses the value 328.

A large integral action time TN can be achieved by choosing a large cycle time T, too. If the block is used within a run number block, the cycle time of the run number block is valid for block INTK and not the cycle time (KD 0,0) of the PLC program.

The controller output Y (manipulated variable) can be limited to a maximum value by specifying a limit at input OG (high limit)

The high and low limits also apply to the controller’s internal I component. I.e. the I component can only assume values between the high and low limits. If the manipulated variable Y reaches one of the two limits, the I component of the controller is no longer changed. This prevents the I component from drifting in the event of limiting of the controller output Y, assuming meaningless values from the point of view of control and, in certain circumstances, not returning to the operating range until after a very long time. This response of a controller is also referred to as a »special anti-reset windup measure«.

Setting and resetting the controller

Setting the controller to an initial value

The output Y of the controller is set to the initial value specified at the INIT input by means of a TRUE signal at input SET.

A TRUE signal at input RES (reset) is equivalent to specifying the initial value 0 (see above).

Surge-free setting/resetting

The output Y of the controller is set to the initial value specified at the INIT input by means of a TRUE signal at the binary input SET.

A TRUE signal at input RES (reset) is equivalent to specifying the initial value 0.

In doing so, adjustment to the initial value is performed internal to the controller. This adjustment is a shift of the controller output from the momentary value to the required initial value. Now, the controller continues operating from this initial value precisely as it would have done at the old operating point before the shift, i.e. without surges. The I component of the controller is defined so that the sum of the P component and the I component just results in the initial value.

Advantage of surge-free setting:
Control as from the new initial value is devoid of surges.

Disadvantage of surge-free setting:
The equation applies: I_component = INIT - P_component

In certain circumstances, the I component is set to high values and may take very long before this »wrong« I component from the point of view of control is dissipated again.
Surging setting/resetting

- The output $Y$ of the controller is set to the initial value specified at the INIT input by means of a TRUE signal at input SET.

- A TRUE signal at input RES (reset) is equivalent to specifying the initial value 0.

In the event of surging setting or resetting of the controller, the I component is set equal to the initial value. To do this, the P component must be suppressed during the setting procedure.

The following applies: I component = INIT

Surging setting to an initial value is achieved by the following measure during the setting procedure:

- Specifying the value 0 at input KP.

This measure renders the controller's P component inactive. The controller output $Y$ assumes the initial value during the set cycle.

The P component is enabled again after the set cycle. From the initial value, the controller output $Y$ jumps according to the P component of the controller.

Advantage of surging setting:

- The I component is not set to »wrong« values from the point of view of control.

Disadvantage of surging setting:

- Not surging-free

$Y \quad \text{INT}$

The manipulated variable $Y$ of the controller is applied at output $Y$.

$\text{OG\_MELD} \quad \text{BOOL}$

Output $\text{OG\_MELD}$ indicates whether the value at output $Y$ has reached the specified top limit.

$\text{OG\_MELD} = \text{FALSE} \quad $ limit has not been reached.

$\text{OG\_MELD} = \text{TRUE} \quad $ limit has been reached.

$\text{UG\_MELD} \quad \text{BOOL}$

Output $\text{UG\_MELD}$ indicates whether the value at output $Y$ has reached the specified low limit.

$\text{UG\_MELD} = \text{FALSE} \quad $ limit has not been reached.

$\text{UG\_MELD} = \text{TRUE} \quad $ limit has been reached.
Example

Declaration:

PI_1: PI;
PI_W AT %MW1000.0: INT;
PI_X AT %MW1000.1: INT;
PI_KP AT %MW3001.0: INT := 100;
PI_TN_TZ AT %MW3001.1: INT := 20;
PI_OG AT %MW3001.2: INT := 20000;
PI_UG AT %MW3001.3: INT := 0;
PI_SET AT %MX0.0: BOOL;
PI_INIT AT %MW3001.4: INT := 10000;
PI_OG_MELD AT %MX0.1: BOOLEAN;
PI_UG_MELD AT %MX0.2: BOOLEAN;
PI_Y AT %MW1000.2: INT;

Translation in ABB IL:

!BA 0
PI
W
X
KP
TN_TZ
INIT
SET
OG
UG
Y
OG_MELD
UG_MELD

FBD:

<table>
<thead>
<tr>
<th>Function call in IL</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD PI_1.OG_MELD</td>
</tr>
<tr>
<td>ST PI_1.UG_MELD</td>
</tr>
<tr>
<td>LD PI_1.UG_MELD</td>
</tr>
<tr>
<td>ST PI_1.Y</td>
</tr>
<tr>
<td>ST PI_Y</td>
</tr>
</tbody>
</table>

Note: In IL, the function call has to be performed in one line.

<table>
<thead>
<tr>
<th>Function call in ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI_OG_MELD := PI_1.OG_MELD;</td>
</tr>
<tr>
<td>PI_UG_MELD := PI_1.UG_MELD;</td>
</tr>
<tr>
<td>PI_Y := PI_1.Y;</td>
</tr>
</tbody>
</table>
The PI controller changes its output Y (manipulated variable) until input X (controlled variable) is equal to input W (command variable).

Transfer function:
\[ F(s) = KP \times \left(1 + \frac{1}{s \times TN} + \frac{s \times TV}{1 + (s \times T1)} \right) \]

Control algorithm: Simple rectangle rule:
\[ Y = \frac{KD \times XD + YI(z-1) + \frac{1}{1 + (T1 \times T1)}}{100} \]

Where:
- \(YI(z-1)\): The integral portion from the previous program cycle
- \(KD\): Control system difference from the previous program cycle
- \(XD\): Control system difference from the current program cycle

The inputs and outputs can neither be duplicated nor negated/inverted.
PIDT1 controller: Surge-free transition from the specified value to control operation

PIDT1 controller: Surging transition from the specified initial value to control operation
The command variable (set point) is specified at input W.

The controlled variable (actual value) is specified at input X.

The proportional coefficient is specified at input KP. This value is specified as a percentage value. Positive or negative values are possible.

Example:
- 1 = 1 percent
- 55 = 55 percent
- 100 = 100 percent
- 1000 = 1000 percent
- -500 = -500 percent

1 percent means that the block multiplies the system deviation by the factor 0.01 (see also control algorithm).

100 percent means that the block multiplies the system deviation by the factor 1 (see also control algorithm).

1000 percent means that the block multiplies the system deviation by the factor 10 (see also control algorithm).

Generally, proportional coefficients of more than 1000 percent are not meaningful in control systems.

The integral action time TN is scaled to the PLC cycle time TZ and is specified at input TN_TZ.

Value range: 0 ≤ TN_TZ ≤ 328

If values are specified which are not within the admissible value range, the PLC generally uses the value 328.

A large integral action time TN can be achieved by choosing a large cycle time T, too. If the block is used within a run number block, the cycle time of the run number block is valid for block INTK and not the cycle time (KD00,00 / MD4000.0) of the PLC program.

The derivative action time TV is scaled to the PLC cycle time TZ and is specified at input TV_TZ:

0 ≤ TN_TZ ≤ 32767.

The returning time T1 is scaled to the PLC cycle time TZ and is specified at input T1_TZ:

0 ≤ TN_TZ ≤ 32767.

The returning time is the time in which the DT1 component has decreased to approximately 37% of its initial value.

Invalid time parameters:
Every time value is set to the maximum positive value 32767 if the time value at the input is erroneously specified as less than or equal to »0«.

The DT1 component of the controller can be activated or deactivated by means of the D-FR input.

In the following cases, from the control point of view it is often disturbing and not meaningful for the DT1 component to be active:

- During activations
- In the event of large system deviations
- When setting the controller to a specified initial value
- When resetting the controller to the value 0

The command and controlled variables can be compared outside of the controller. Depending on this comparison, the DT1 component can be activated or deactivated specifically using the D-FR input.

For example, the activation can be restricted to the condition that the system deviation is within a required bandwidth. I.e. the DT1 component is only active if the controlled variable fluctuates around the set point within a specific bandwidth. The DT1 component is deactivated if the controlled variable leaves this tolerance band.
Limiting the manipulated variable Y

OG INT
UG INT

The controller output Y (manipulated variable) can be limited
– to a maximum value by specifying a limit at input OG (high limit)
– to a minimum value by specifying a limit at input UG (low limit).

The high and low limits also apply to the controller’s internal I component. I.e the I component can only assume values between the high and low limits. If the manipulated variable Y reaches one of the two limits, the controller’s I component is no longer changed.

This prevents the I component from drifting in the event of limiting of the controller output Y, assuming meaningless values from the point of view of control and, in certain circumstances, not returning to the operating range until after a very long time. This response of a controller is also referred to as a »special anti-reset windup measure«.

Setting and resetting the controller

SET BOOL
INIT INT
RES BOOL

Setting the controller to an initial value
– The output Y of the controller is set to the initial value specified at the INIT input by means of a TRUE signal at input SET.
– A 1 signal at input RES (reset) is equivalent to specifying the initial value 0 (see above).

Surge-free setting/resetting

– Output Y of the controller is set to the initial value specified at the INIT input by means of a TRUE signal at the binary input SET.
– A 1 signal at input RES (reset) is equivalent to specifying the initial value 0.

In doing so, adjustment to the initial value is performed internal to the controller. This adjustment is a shift of the controller output from the momentary value to the required initial value. Now, the controller continues operating from this initial value precisely as it would have done at the old operating point before the shift, i.e. without surges. The controller’s I component is defined in a way that the sum of the P, I and the DT1 component just results in the initial value.

Advantage of surge-free setting:
– Control as from the new initial value is devoid of surges.

Disadvantage of surge-free setting:
– The equation applies:
  I component = INIT – P component – DT1 comp.

In certain circumstances, the I component is set to high values and may take very long before this »wrong« I component from the point of view of control is dissipated again.

Surging setting/resetting

– The output Y of the controller is set to the initial value specified at the INIT input by means of a TRUE signal at input SET.
– A 1 signal at input RES (reset) is equivalent to specifying the initial value 0.
– In the event of surging setting or resetting of the controller the I component is set equal to the initial value. To do this, the P component and the DT1 component must be suppressed during setting procedure.

The following applies: I component = INIT

Surging setting to an initial value is achieved by the following measures during setting procedure:
– Deactivation of the DT1 component via the D-FR control input and
– Specifying the value 0 at the input KP.

These measures render the P component and the DT1 component inactive during setting of the controller.

The controller output assumes the initial value during the set cycle.

The P component and the DT1 component are enabled again after the set cycle. From the initial value, the controller output Y jumps according to the P component and the DT1 component of the controller.

Advantage of surging setting:
– The I component is not set to »wrong« values from the point of view of control.

Disadvantage of surging setting:
– Not surging-free

Y INT

The manipulated variable Y of the controller is applied at output Y.
**OG_MELD**

Output `OG_MELD` indicates whether the value at output Y has reached the specified top limit.

- `OG_MELD = FALSE` limit has not been reached.
- `OG_MELD = TRUE` limit has been reached.

**UG_MELD**

Output `UG_MELD` indicates whether the value at output Y has reached the specified low limit.

- `UG_MELD = FALSE` limit has not been reached.
- `UG_MELD = TRUE` limit has been reached.

### Example

#### Declaration:

```plaintext
PIDT1_1 : PIDT1;
PIDT1_W AT %MW1000.0: INT;
PIDT1_X AT %MW1000.1: INT;
PIDT1_KP AT %MW3001.0: INT := 100;
PIDT1_TN_TZ AT %MW3001.1: INT := 20;
PIDT1_TV_TZ AT %MW3001.2 : INT := 0;
PIDT1_T1_TZ AT %MW3001.3 : INT := 0;
PIDT1_DT1_FR AT %MX0.4 : BOOL;
PIDT1_OG AT %MW3001.5: INT := 20000;
PIDT1_UG AT %MW3001.6: INT := 0;
PIDT1_SET AT %MX0.0: BOOL;
PIDT1_INIT AT %MW3001.7: INT := 10000;
PIDT1_RES AT %MX0.1: BOOL;
PIDT1_OG_MELD AT %MX0.2: BOOL;
PIDT1_UG_MELD AT %MX0.3: BOOL;
PIDT1_Y AT %MW1000.2: INT;
```

#### Translation in ABB IL:

```plaintext
!BA 0
PIDT1
W
X
KP
TN_TZ
TV_TZ
T1_TZ
DT1_FR
INIT
SET
RES
OG
UG
Y
OG_MELD
UG_MELD
```

#### FBD:

```
```

#### ABB IL of example:

```plaintext
!BA 0
PI
MW0,0
MW1,1
KW1,0 ; 100
KW1,1 ; 20
KW1,2 ; 0
KW1,3 ; 0
M0,4
M0,0
M0,1
KW1,5 ; 10000
M0,0
M0,1
KW1,6 ; 20000
KW1,7 ; 0
MW0,2
M0,2
M0,3
```
Function call in IL

CAL PIDT1_1(W := PIDT1_W,  
   X := PIDT1_X,  
   KP := PIDT1_KP,  
   TN_TZ := PIDT1_TNTN,  
   TV_TZ := PIDT1_TVTZ,  
   T1_TZ := PIDT1_T1_TZ,  
   DT1_FR := PIDT1_DT1_FR,  
   OG := PIDT1_OG,  
   UG := PIDT1_UG,  
   SET := PIDT1_SET,  
   INIT := PIDT1_INIT,  
   RES := PIDT1_RES)

LD PIDT1_1.OG_MELD  
ST PIDT1_1.OG_MELD  
LD PIDT1_1.UG_MELD  
ST PIDT1_1.UG_MELD  
LD PIDT1_1.Y  
ST PIDT1_1.Y

Note: In IL, the function call has to be performed in one line.

Function call in ST

PIDT1_1  
   W := PIDT1_W,  
   X := PIDT1_X,  
   KP := PIDT1_KP,  
   TN_TZ := PIDT1_TNTN,  
   TV_TZ := PIDT1_TVTZ,  
   T1_TZ := PIDT1_T1_TZ,  
   DT1_FR := PIDT1_DT1_FR,  
   OG := PIDT1_OG,  
   UG := PIDT1_UG,  
   SET := PIDT1_SET,  
   INIT := PIDT1_INIT,  
   RES := PIDT1_RES);

PIDT1_OG_MELD := IDT1_1.OG_MELD;  
PIDT1_UG_MELD := IDT1_1.UG_MELD;  
PIDT1_Y := IDT1_1.Y;
RESET MEMORY, DOMINATING

A status TRUE at the input SET sets the operand Q1 to the status TRUE.

A status TRUE at the input RESET1 resets the operand Q1 to the status FALSE.

=> dominating RESET

Block type
Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>RS</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>BOOL</td>
<td>Set input</td>
</tr>
<tr>
<td>RESET1</td>
<td>BOOL</td>
<td>Reset input</td>
</tr>
<tr>
<td>Q1</td>
<td>BOOL</td>
<td>Flip-flop output</td>
</tr>
</tbody>
</table>

Description

A status TRUE at the input SET sets the operand Q1 to the status TRUE.

A status TRUE at the input RESET1 resets the operand Q1 to the status FALSE.

A simultaneous TRUE status at the inputs SET and RESET1 sets operand Q1 to a status FALSE (dominating reset)

A status FALSE at the input SET or RESET1 has no influence on the operand Q1

Example

Declaration:

RS_1 : RS;
RS_SET AT %MX0.0 : BOOL;
RS_RESET1 AT %MX0.1 : BOOL;
RS_Q1 AT %MX0.2 : BOOL;

Translation in ABB IL:

! SET =S Q1
! RESET1 =R Q1

ABB IL of example:

! M0,0 =S M0,2
! M0,1 =R M0,2

Function call in IL

CAL RS_1(SET = RS_SET,
       RESET1 := RS_RESET)

LD RS_1.Q1
ST RS_Q

Function call in ST

RS_1 ( SET := RS_SET,
       RESET1 := RS_RESET);

RS_Q := RS_1.Q1;
SERIAL LINE INITIALIZATION

Serial interface initialization function block.

The function block SINIT is processed once with every FALSE/TRUE edge at the FREI input. It initializes the serial interface specified at the SSK input.

The interface can be operated by the PLC program (e.g. with the DRUCK and EMAS blocks).

Block type

Function block with historical values

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREI</td>
<td>BOOL</td>
<td>Enable processing of the block, FALSE-&gt;TRUE edge</td>
</tr>
<tr>
<td>SSK</td>
<td>INT</td>
<td>Interface identifier (1 or 2)</td>
</tr>
<tr>
<td>BAUD</td>
<td>INT</td>
<td>Baud rate; 300 ... 9600 Baud</td>
</tr>
<tr>
<td>STOP</td>
<td>INT</td>
<td>Number of stop bits; input has no effect</td>
</tr>
<tr>
<td>ZL</td>
<td>INT</td>
<td>Character length, 7 or 8 data bits per character</td>
</tr>
<tr>
<td>PTY</td>
<td>BOOL</td>
<td>Parity, enable/disable</td>
</tr>
<tr>
<td>E_O</td>
<td>BOOL</td>
<td>Parity even/odd</td>
</tr>
<tr>
<td>ECHO</td>
<td>BOOL</td>
<td>Echo, on/off</td>
</tr>
<tr>
<td>SBRK</td>
<td>BOOL</td>
<td>Send break character</td>
</tr>
<tr>
<td>FEND</td>
<td>BOOL</td>
<td>Enable end of text character for transmitting direction</td>
</tr>
<tr>
<td>ENDS</td>
<td>INT</td>
<td>End of text character for transmitting direction</td>
</tr>
<tr>
<td>ENDE</td>
<td>INT</td>
<td>End of text character for receiving direction</td>
</tr>
</tbody>
</table>

Description

Serial interface initialization function block.

Before using one of these interfaces, the interface must be initialized. The function block SINIT is available for this purpose.

The function block SINIT is processed once with every FALSE->TRUE edge at the FREI input.

It initializes the serial interface specified at the SSK input (COM1, COM2).

FREI | BOOL
The block is run through once when a FALSE/TRUE edge is specified at the FREI input. As the result of this, the serial interface whose number is specified at the SSK input is initialized and the interface is then operable.

SSK | INT
The number of the interface to be initialized is specified at the input SSK.

The following applies:

COM1 : number = 1
COM2 : number = 2
BAUD
The value for the baud rate is specified at the BAUD input.
Baud rate: 300 ... 9600 Baud

STOP
The number of stop bits is set to 1 and can not be modified. The value for the number of stop bits specified at the input STOP has no significance.

ZL
The input ZL specifies the required character length. The character length signifies the number of data bits per character. 7 or 8 data bits per character are possible.

PTY
The input PTY specifies whether a character is transferred with or without a parity bit.
PTY = 0 -> Transfer without parity bit
PTY = 1 -> Transfer with parity bit

E_O
The input E/O specifies whether an even or odd parity bit is required.
E/O = 0 -> Odd parity bit
E/O = 1 -> Even parity bit

ECHO
The input ECHO specifies whether the characters received through the applicable interface are to be reflected (echoed) by the PLC. In this way, the sender of a character, for example, can determine whether or not it has arrived correctly in the PLC.
ECHO = 0 -> No echo, character is not reflected
ECHO = 1 -> Echo, character is reflected

SBRK
The state of the transmit line TxD can be influenced at the input SBRK (send break character).
SBRK = 0 -> Normal state of the transmit line TxD for transfer of characters
SBRK = 1 -> Transmit line TxD set to "0"

FEND
The input FEND specifies whether or not the end of text character planned at input ENDS is output at the same time (enable end character).
FEND = 0 -> End of text character in transmitting direction is not output
FEND = 1 -> End of text character in transmitting direction is output

ENDS
A freely selectable end of text character for the transmitting direction can be specified at the ENDS input. This end character is then appended automatically to every text (telegram) which the DRUCK block sends to the outside world through the serial interface. However, a precondition is that input FEND is enabled.
The end of text character is specified as a numeric value.
Example:
3 or 03H signifies <ETX>
4 or 04H signifies <EOT>
13 or 0DH signifies <CR>
10 or OAH signifies <LF>
32 or 20H signifies <SP>
...

ENDE
A freely selectable end of text character for the receiving direction can be specified at the ENDE input. When a telegram is received through the serial interface, the PLC recognizes the end of the telegram by virtue of this end character. The end character is specified in the same way as in the case of the ENDS input.
Example

Declaration:

SINIT1 : SINIT;
SINIT_FREI AT %MX0.0 : BOOL;
SINIT_SSK AT %MW3001.1 : INT := 1;
SINIT_BAUD AT %MW3001.2 : INT := 9600;
SINIT_STOP AT %MW3001.3 : INT := 1;
SINIT_ZL AT %MW3001.4 : INT := 8;
SINIT_PTY AT %MX0.1 : BOOL;
SINIT_E_O AT %MX0.2 : BOOL;
SINIT_ECHO AT %MX0.3 : BOOL;
SINIT_SBRK AT %MX0.4 : BOOL;
SINIT_FEND AT %MX0.5 : BOOL;
SINIT_ENDS AT %MW3001.5 : INT := 13;
SINIT_ENDE AT %MW3001.6 : INT := 13;

Translation in ABB IL:

!BA 0
SINIT
FREQ
SSK
BAUD
STOP
ZL
PTY
E_O
ECHO
SBRK
FEND
ENDS
ENDE

FBD:

```plaintext
SINIT1
  SINIT
    SINIT_FREI := FREI
    SINIT_SSK := SSK
    SINIT_BAUD := BAUD
    SINIT_STOP := STOP
    SINIT_ZL := ZL
    SINIT_PTY := PTY
    SINIT_E_O := E_O
    SINIT_ECHO := ECHO
    SINIT_SBRK := SBRK
    SINIT_FEND := FEND
    SINIT_ENDS := ENDS
    SINIT_ENDE := ENDE
```

ABB IL of example:

!BA 0
SINIT
M0,0
KW1,1 ; 1
KW1,2 ; 9600
KW1,3 ; 1
KW1,4 ; 8
M0,1
M0,2
M0,3
M0,4
M0,5
KW1,5 ; 13
KW1,6 ; 13

Function call in IL

CAL SINIT1(FREI := SINIT_FREI,
            SSK := SINIT_SSK,
            BAUD := SINIT_BAUD,
            STOP := SINIT_STOP,
            ZL := SINIT_ZL,
            PTY := SINIT_PTY,
            E_O := SINIT_E_O,
            ECHO := SINIT_ECHO,
            SBRK := SINIT_SBRK,
            FEND := SINIT_FEND,
            ENDS := SINIT_ENDS,
            ENDE := SINIT_ENDE)

Note: In IL, the function call has to be performed in one line.

Function call in ST

SINIT1 (FREI := SINIT_FREI,
        SSK := SINIT_SSK,
        BAUD := SINIT_BAUD,
        STOP := SINIT_STOP,
        ZL := SINIT_ZL,
        PTY := SINIT_PTY,
        E_O := SINIT_E_O,
        ECHO := SINIT_ECHO,
        SBRK := SINIT_SBRK,
        FEND := SINIT_FEND,
        ENDS := SINIT_ENDS,
        ENDE := SINIT_ENDE)
The function block SQRTD generates the square root of the value at the input E. The result is available at the output A and is always rounded down to an integral number. If the value at the input E must be a positive number. If the value at the input is negative, the value ‘zero’ is output through the output A and the value TRUE is output through the ERR output.

**Parameters**

<table>
<thead>
<tr>
<th>Instance</th>
<th>SQRTD</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>DINT</td>
<td>Input</td>
</tr>
<tr>
<td>A</td>
<td>DINT</td>
<td>Square root of the input value</td>
</tr>
<tr>
<td>ERR</td>
<td>BOOL</td>
<td>Error if the input value is negative, operands M, A (not E, S)</td>
</tr>
</tbody>
</table>

**Description**

The function block SQRTD generates the square root of the value at the input E. The result is available at the output A and is always rounded down to an integral number. If the value at the input E must be a positive number. If the value at the input is negative, the value ‘zero’ is output through the output A and the value TRUE is output through the ERR output.

**E**

The square root of the value at the input E is generated and is available as a value of the output operand A.

**A**

The value of the square root is available at the output A.

**ERR**

The ERR output indicates whether the value of the input operand E is positive (>= 0) or negative (< 0).

Input E >= 0 -> ERR = FALSE and A = square root
Input E < 0  -> ERR = TRUE and A = 0
Example

Declaration:

```plaintext
SQRTD_1 : SQRTD;
SQRTD_E AT %MD2000.0 : DINT;
SQRTD_A AT %MD2000.1 : DINT;
SQRTD_ERR AT %MX0.0 : BOOL;
```

Translation in ABB IL:

```plaintext
!BA 0
SQRT
K0,1
E
A
ERR
```

FBD:

```
SQRTD_1
SQRTD_E = E
A
ERR = SQRTD_ERR
```

ABB IL of example:

```plaintext
!BA 0
SQRT
K0,1
MD0,0
MD0,1
M0,0
```

Function call in IL

```
CAL SQRTD_1(E := SQRTD_E)
LD SQRTD_1.A
ST SQRTD_A
LD SQRTD_1.ERR
ST SQRTD_ERR
```

Function call in ST

```
SQRTD_1(E := SQRTD_E);
SQRTD_A := SQRTD_1.A;
SQRTD_ERR := SQRTD_1.ERR;
```
SQUARE ROOT, WORD

The function block SQRTW generates the square root of the value at the input E. The result is available at the output A and is always rounded down to an integral number. If the value at the input E must be a positive number. If the value at the input is negative, the value 'zero' is output through the output A and the value TRUE is output through the ERR output.

Block type
Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>SQRTW</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>INT</td>
<td>Input</td>
</tr>
<tr>
<td>A</td>
<td>INT</td>
<td>Square root of the input value</td>
</tr>
<tr>
<td>ERR</td>
<td>BOOL</td>
<td>Error if the input value is negative, operands M, A (not E, S)</td>
</tr>
</tbody>
</table>

Description

The function block SQRTW generates the square root of the value at the input E. The result is available at the output A and is always rounded down to an integral number. If the value at the input E must be a positive number. If the value at the input is negative, the value 'zero' is output through the output A and the value TRUE is output through the ERR output.

E INT
The square root of the value at the input E is generated and is available as a value of the output operand A.

A INT
The value of the square root is available at the output A.

ERR BOOL
The ERR output indicates whether the value of the input operand E is positive (>= 0) or negative (< 0).

Input E >= 0 -> ERR = FALSE and A = square root
Input E < 0  -> ERR = TRUE and A = 0
Example

Declaration:

SQRTW_1 : SQRTD;
SQRTW_E AT %MW1000.0 : DINT;
SQRTW_A AT %MW1000.1 : DINT;
SQRTW_ERR AT %MX0.0 : BOOL;

Translation in ABB IL:

!BA 0
SQRT
K0,0
E
A
ERR

FBD:

Function call in IL

CAL SQRTW_1(E := SQRTD_E)
LD SQRTW_1.A
ST SQRTW_A
LD SQRTW_1.ERR
ST SQRTW_ERR

Function call in ST

SQRTW_1(E := SQRTW_E);
SQRTW_A := SQRTW_1.A;
SQRTW_ERR := SQRTW_1.ERR;
SET MEMORY , DOMINATING

A status TRUE at the input SET1 sets the operand Q1 to the status TRUE.

A status TRUE at the input RESET resets the operand Q1 to the status FALSE.

=> dominating SET

Block type

Function block without historical values

Parameters

Instance | SR | Instance name
---|---|---
SET1  | BOOL | Set input
RESET | BOOL | Reset input
Q1  | BOOL | Flip-flop output

Description

A status TRUE at the input SET1 sets the operand Q1 to the status TRUE.

A status TRUE at the input RESET resets the operand Q1 to the status FALSE.

A simultaneous TRUE status at the inputs SET1 and RESET sets operand Q1 to a status TRUE (dominating set).

A status FALSE at the input SET1 or RESET has no influence on the operand Q1.

Example

Declaration:

SR_1 : SR;
SR_SET1 AT %MX0.0 : BOOL;
SR_RESET AT %MX0.1 : BOOL;
SR_Q1 AT %MX0.2 : BOOL;

Translation in ABB IL:

! RESET
=R Q1
! SET1
=S Q1

ABB IL of example:

! M0,1
=R M0,2
! M0,0
=S M0,2

Function call in IL

CAL SR_1 (SET1 := SR_SET, RESET := SR_RESET)
LD SR_1.Q1
ST SR_Q

Function call in ST

SR_1 (SET1 := SR_SET, RESET := SR_RESET);
SR_Q := SR_1.Q1;
SUBTRACTION DOUBLE WORD

The value of the operand at input E2 is subtracted from the value of the operand at input E1 and the result is assigned to the operand at output A.

The result is limited to the maximum or minimum value of the number range (Number range: -2147483647 … 2147483647). If limiting occurred, a TRUE signal is assigned to the binary operand at output Q. If no limiting occurred, a FALSE signal is assigned to the binary operand at output Q.

Block type

Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>SUBD</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>DINT</td>
<td>Minuend</td>
</tr>
<tr>
<td>E2</td>
<td>DINT</td>
<td>Subtrahend</td>
</tr>
<tr>
<td>A</td>
<td>DINT</td>
<td>Result (difference)</td>
</tr>
<tr>
<td>Q</td>
<td>BOOL</td>
<td>Result limited</td>
</tr>
</tbody>
</table>

Description

The value of the operand at input E2 is subtracted from the value of the operand at input E1 and the result is assigned to the operand at output A.

The result is limited to the maximum or minimum value of the number range (Number range: -2147483647 … 2147483647). If limiting occurred, a TRUE signal is assigned to the binary operand at output Q. If no limiting occurred, a FALSE signal is assigned to the binary operand at output Q.

The inputs and outputs can neither be duplicated nor negated.
Example

**Declaration:**

| SUBD_1 : SUBD; |
| SUBD_E1 AT %MD2000.0 : DINT; |
| SUBD_E2 AT %MD2000.1 : DINT; |
| SUBD_A AT %MD2000.2 : DINT; |
| SUBD_Q AT %MX0.0 : BOOL; |

**Translation in ABB IL:**

| IBA 0 |
| SUBD |
| E1 |
| E2 |
| A |
| Q |

**FBD:**

```
SUBD_1

SUBD_E1 -- E1
A

SUBD_E2 -- E2
Q -- SUBD_Q
```

**ABB IL of example:**

| IBA 0 |
| SUBD |
| MD0.0 |
| MD0.1 |
| MD0.2 |
| M0.0 |

**Function call in IL**

```
CAL SUBD_1(E1 := SUBD_E1, E2 := SUBD_E2)
LD SUBD1.A
ST SUBD_A
LD SUBD1.Q
ST SUBD_Q
```

**Function call in ST**

```
SUBD_1(E1 := SUBD_E1, E2 := SUBD_E2);
SUBD_A:=SUBD1.A;
SUBD_Q:=SUBD1.Q;
```
TIME WORD CONVERSION

This function is used to manage time value with word variables.

The time in [ms] TIMEE is converted in hours, minutes, seconds and milliseconds.

Block type

Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>TIME_W</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMEE</td>
<td>DINT</td>
<td>Time value</td>
</tr>
<tr>
<td>H</td>
<td>INT</td>
<td>Hours value</td>
</tr>
<tr>
<td>M</td>
<td>INT</td>
<td>Minute value</td>
</tr>
<tr>
<td>SEC</td>
<td>INT</td>
<td>Second value</td>
</tr>
<tr>
<td>MS</td>
<td>INT</td>
<td>Millisecond value</td>
</tr>
</tbody>
</table>

Description

This function is used to manage time value with word variables.

The time value TIMEE is converted in hours, minutes, seconds and milliseconds.

TIMEE

The time value is in millisecond.

Value range:

\[0 \leq \text{TIMEE} \leq 986399999\]

IF TIMEE is greater than 986399999 then H, M, SEC, MS are set to the maximum value.

IF TIMEE is negative then H, M, SEC, MS are set to zero.

H

Output for the hours.

Value range:

\[0 \leq H \leq 273\]
Example

Declaration:

```plaintext
TIME_W_1 : TIME_W;
TIME_W_TIMEE AT %MD2000.0 : DINT;
TIME_W_H AT %MW1000.0 : INT;
TIME_W_M AT %MW1000.1 : INT;
TIME_W_SEC AT %MW1000.2 : INT;
TIME_W_MS AT %MW1000.3 : INT;
```

Translation in ABB IL:

```plaintext
!BA 0
TIME_W
H
M
SEC
MS
```

FBD:

```
TIME_W_1

TIME_W_TIMEE

TIME_W_H

TIME_W_M

TIME_W_SEC

TIME_W_MS
```

ABB IL of example:

```plaintext
!BA 0
TIME_W
MD0,0
MW0,0
MW0,1
MW0,2
MW0,3
```

Function call in IL

```plaintext
CAL TIME_W_1(TIMEE := TIME_W_TIMEE);
LD TIME_W_1.H
ST TIME_W_H
LD TIME_W_1.M
ST TIME_W_M
LD TIME_W_1.SEC
ST TIME_W_SEC
LD TIME_W_1.MS
ST TIME_W_MS
```

Function call in ST

```plaintext
TIME_W_1(TIMEE := TIME_W_TIMEE);
TIME_W_H := TIME_W_1.H;
TIME_W_M := TIME_W_1.M;
TIME_W_SEC := TIME_W_1.SEC;
TIME_W_MS := TIME_W_1.MS;
```
OFF DELAY

The TRUE/FALSE edge of the input E is delayed by the time T at the output A. If the input E returns to TRUE level before expiry of the time T, the output A retains TRUE level.

Block type

Function block with historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>TOF</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>BOOL</td>
<td>Input signal</td>
</tr>
<tr>
<td>T</td>
<td>TIME</td>
<td>Preset time</td>
</tr>
<tr>
<td>A</td>
<td>BOOL</td>
<td>Delayed signal, operands M, A (not E, S)</td>
</tr>
<tr>
<td>ET</td>
<td>TIME</td>
<td>Time visualization</td>
</tr>
</tbody>
</table>

Description

The TRUE/FALSE edge of the input E is delayed by the time T at the output A. If the input E returns to TRUE level before expiry of the time T, the output A retains TRUE level.

The time elapsed can be consulted at the output ET and the preset time value at the input T can be modified when the timer is running. The preset time is specified in milliseconds. The time range which can be specified is : 1 ms ... 24,8 days.

Maximum time offset at the output : < 1 cycle time

Meaningful range for T : > 1 cycle time.

General behavior

- Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program. An appropriate message of the operating system is not issued to the affiliated timer block in the PLC program until the timer has elapsed.

- Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:
  - Abort PLC program
  - RUN/STOP switch from RUN -> STOP
  - Warm or cold start
Example

Declaration:

```plaintext
TOF_1 : TOF;
TOF_IN AT %MX0.0 : BOOL;
TOF_PT AT %MD40001.0 : TIME := t#10s; (* 10000 ms *)
TOF_Q AT %MX0.1 : BOOL;
TOF_ET AT %MD2000.0 : TIME;
```

Translation in ABB IL:

```plaintext
IBA 0
TOF
IN
PT
Q
ET
```

FBD:

![FBD diagram]

Function call in IL

```plaintext
CAL TOF_1(E := TOF_E, T := TOF_T)
LD TOF_1.A
ST TOF_A
LD TOF_1.ET
ST TOF_ET
```

Abb IL of example:

```plaintext
IBA 0
TOF
M0,0
KD1,0 ; 10000
M0,1
MD0,0
```

Function call in ST

```plaintext
TOF_1(E := TOF_E, T := TOF_T);
TOF_A := TOF_1.A;
TOF_ET := TOF_1.ET;
```
ON DELAY

The FALSE/TRUE edge of the input E is delayed by the time T at the output A. The output A retains FALSE level if the input E returns to FALSE level before the time T has elapsed.

Block type

Function block with historical values

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>TON</td>
<td>Instance name</td>
</tr>
<tr>
<td>E</td>
<td>BOOL</td>
<td>Input signal</td>
</tr>
<tr>
<td>T</td>
<td>TIME</td>
<td>Preset time</td>
</tr>
<tr>
<td>A</td>
<td>BOOL</td>
<td>Delayed signal</td>
</tr>
<tr>
<td>ET</td>
<td>TIME</td>
<td>Time visualization</td>
</tr>
</tbody>
</table>

Description

The FALSE/TRUE edge of the input E is delayed by the time T at the output A. The output A retains FALSE level if the input E returns to FALSE level before the time T has elapsed.

The time elapsed can be consulted at the output ET and the preset time value at the input T can be modified when the timer is running. The preset time is specified in milliseconds. The time range which can be specified is: 1 ms ... 24.8 days.

Maximum time offset at the output: < 1 cycle time

Meaningful range for T: > 1 cycle time

General behavior

- Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program. An appropriate message of the operating system is not issued to the affiliated timer block in the PLC program until the timer has elapsed.

- Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:
  - Abort PLC program
  - RUN/STOP switch from RUN -> STOP
  - Warm or cold start
Example

Declaration:

TON_1 : TON;
TON_IN AT %MX0.0 : BOOL;
TON_PT AT %MD40001.0 : TIME := t#10s; (* 10000 ms *)
TON_Q AT %MX0.1 : BOOL;
TON_ET AT %MD2000.0 : TIME;

Translation in ABB IL:

IBA 0
TON
IN
PT
Q
ET

FBD:

ABB IL of example:

IBA 0
TON
M0,0
KD1,0 ; 10000
M0,1
MD0,0

Function call in IL

CAL TON_1(E := TON_E, T := TON_T)
LD TON_1.A
ST TON_A
LD TON_1.ET
ST TON_ET

Function call in ST

TON_1(E := TON_E, T := TON_T);
TON_A := TON_1.A;
TON_ET := TON_1.ET;
**MONOSTABLE ELEMENT « CONSTANT »**

The FALSE/TRUE edge at the input E produces a FALSE/TRUE edge at the output A. The output A is reset to FALSE level after expiry of the period T.

A second FALSE/TRUE Edge of the input E before the time T has elapsed is ignored.

### Block type

Function block with historical values

### Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>ASV</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>BOOL</td>
<td>Input signal</td>
</tr>
<tr>
<td>T</td>
<td>TIMER</td>
<td>Delay time</td>
</tr>
<tr>
<td>A</td>
<td>BOOL</td>
<td>Delayed signal</td>
</tr>
<tr>
<td>ET</td>
<td>DINT</td>
<td>Time visualization</td>
</tr>
</tbody>
</table>

### Description

The FALSE/TRUE edge at the input E produces a FALSE/TRUE edge at the output A. The output A is reset to FALSE level after expiry of the period T.

A second FALSE/TRUE Edge of the input E before the time T has elapsed is ignored.

The time elapsed can be consulted at the output ET and the preset time value at the input E can be modified when the timer is running.

The preset time is specified in milliseconds. The time range which can be specified is: 1 ms ... 24.8 days.

Maximum time offset at the output: < 1 cycle time

Meaningful range for PT: > 1 cycle time.

### General behavior

- Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program. An appropriate message of the operating system is not issued to the affiliated timer block in the PLC program until the timer has elapsed.
- Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:
  - Abort PLC program
  - RUN/STOP switch from RUN -> STOP
  - Warm or cold start

---

![Diagram of the timer block](image-url)
Example

Declaration:

```plaintext
TP_1 : TOF;
TP_IN AT %MX0.0 : BOOL;
TP_PT AT %MD4001.0 : TIME := t#10s; (* 10000 ms *)
TP_Q AT %MX0.1 : BOOL;
TP_ET AT %MD2000.0 : TIME;
```

Translation in ABB IL:

```plaintext
IBA 0
TP
IN
PT
Q
ET
```

FBD:

![FBD Diagram]

ABB IL of example:

```plaintext
IBA 0
TP
M0,0
KD1,0 ; 10000
M0,1
MD0,0
```

Function call in IL

```plaintext
CAL TP_1(E := TP_E, T := TP_T)
LD TP_1.A
ST TP_A
LD TP_1.ET
ST TP_ET
```

Function call in ST

```plaintext
TP_1(E := TP_E, T := TP_T);
TP_A := TP_1.A;
TP_ET := TP_1.ET;
```
DISPLAY AND SET CLOCK

This function block allows to set and display the current time and date.

The inputs and outputs can neither be duplicated nor inverted nor negated.

The clock is set by means of the set inputs for the time and date. The values present at the set inputs are adopted with the occurrence of a FALSE/TRUE edge at input SET. As long as a TRUE signal is present at the FREI input, the current date and time are indicated at the block’s outputs.

**Parameters**

<table>
<thead>
<tr>
<th>Instance</th>
<th>UHR</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREI</td>
<td>BOOL</td>
<td>Enable block processing</td>
</tr>
<tr>
<td>SET</td>
<td>BOOL</td>
<td>FALSE/TRUE edge sets the time and date</td>
</tr>
<tr>
<td>ESEC</td>
<td>INT</td>
<td>Set input for seconds</td>
</tr>
<tr>
<td>EMIN</td>
<td>INT</td>
<td>Set input for minutes</td>
</tr>
<tr>
<td>EH</td>
<td>INT</td>
<td>Set input for hours</td>
</tr>
<tr>
<td>ETAG</td>
<td>INT</td>
<td>Set input for days</td>
</tr>
<tr>
<td>EMON</td>
<td>INT</td>
<td>Set input for months</td>
</tr>
<tr>
<td>EJHR</td>
<td>INT</td>
<td>Set input for years</td>
</tr>
<tr>
<td>EWO</td>
<td>INT</td>
<td>Set input for weekdays</td>
</tr>
<tr>
<td>AKT</td>
<td>BOOL</td>
<td>Topicality (usefulness) of the data at the outputs</td>
</tr>
<tr>
<td>ERR</td>
<td>INT</td>
<td>Error identifier</td>
</tr>
<tr>
<td>ASE</td>
<td>INT</td>
<td>Output seconds</td>
</tr>
<tr>
<td>AMIN</td>
<td>INT</td>
<td>Output minutes</td>
</tr>
<tr>
<td>AH</td>
<td>INT</td>
<td>Output hours</td>
</tr>
<tr>
<td>ATAG</td>
<td>INT</td>
<td>Output days</td>
</tr>
<tr>
<td>AMON</td>
<td>INT</td>
<td>Output months</td>
</tr>
<tr>
<td>AJHR</td>
<td>INT</td>
<td>Output years</td>
</tr>
<tr>
<td>AWO</td>
<td>INT</td>
<td>Output weekday no.</td>
</tr>
</tbody>
</table>

**Description**

This function block allows to set and display the current time and date.

Enabling the block

FREI = FALSE:
The block is not processed. The AKT and ERR outputs are set to FALSE or 0. The time and date outputs are no longer changed by the block.

FREI = TRUE: Block is processed
DISPLAY AND SET CLOCK

SET

FALSE/TRUE edge:
The clock is set to the values present at the time and date inputs.
During the setting procedure the time and date at the block outputs are invalid (output AKT = FALSE).

Set inputs for date and time

In the event of a FALSE/TRUE edge at input SET, the clock is set to the values present at the set inputs. If the specified set values are inadmissible, the AKT output is set to FALSE and an error message appears at the ERR output. In this case the values present at the time and date outputs are invalid. The clock has to be set again.

ESEC

Set input for seconds.
Value range: 0 ... 59.

EMIN

Set input for minutes.
Value range: 0 ... 59.

EH

Set input for hours.
The clock operates in 24 hour mode, i.e. it changes from 23:59:59 h to 0:0:0 h.
Value range: 0 ... 23.

ETAG

Set input for days (which day of the month).
The clock knows the number of days depending on the months and leap years. For the clock, a leap year exists when the year number is an integral multiple of 4. The maximum value for the days (28, 29, 30, 31) depends on the month.
Value range: 1...28, 29, 30, 31.

EMON

Set input for months.
Value range: 1 ... 12.

EJHR

Set input for years.
The clock only indicates the years and decades.
Value range: 0 ... 99.

EWO

Set input for the number of the weekday.
This input specifies on which day of the week the input is made. That is to say, it is possible to determine which day of the week the day with the number 1 is to be (e.g. Sunday or Monday).
Value range: 1 ... 7.

Example:
The clock is set on Friday, 01.07.88. If the value 6 is entered for the weekday number, Friday is now the 6th day of the week and Sunday is defined as the 1st day of the week.

AKT

Indication of the topicality (usefulness) of the outputs.
AKT is TRUE if:
– The date and time outputs were updated in the current cycle.
– The values at the outputs are consistent, i.e. none of the values at the date or time outputs has changed during updating. They all originate from the same clock pulse.
– The clock was set correctly.

AKT = TRUE → ERR = 0:
Date/time are valid
AKT = FALSE → ERR > 0:
Date/time are invalid
The reason is displayed at output ERR as an error identifier.
**ERR INT**

In the event of an error, the relevant error identifier is available at output ERR.

Meanings of the error identifiers:

**• No error has occurred:**

ERR=0: No error has occurred or FREI = FALSE, i.e. block disabled.

**• Error when setting the clock:**

<table>
<thead>
<tr>
<th>ERR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR=1</td>
<td>0 &lt; SEC &lt; 59 has not been met</td>
</tr>
<tr>
<td>ERR=2</td>
<td>0 &lt; MIN &lt; 59 has not been met</td>
</tr>
<tr>
<td>ERR=3</td>
<td>0 &lt; H &lt; 23 has not been met</td>
</tr>
<tr>
<td>ERR=4</td>
<td>1 &lt; TAG &lt; 28, 29, 30, 31 (depending on the month) has not been met</td>
</tr>
<tr>
<td>ERR=5</td>
<td>1 &lt; MON &lt; 12 has not been met</td>
</tr>
<tr>
<td>ERR=6</td>
<td>0 &lt; JHR &lt; 99 has not been met</td>
</tr>
<tr>
<td>ERR=7</td>
<td>1 &lt; WTG &lt; 7 has not been met</td>
</tr>
<tr>
<td>ERR=8</td>
<td>The transmission mailbox is currently occupied by another user. The block waits until the mailbox is free and afterwards date/time are set.</td>
</tr>
<tr>
<td>ERR=9</td>
<td>Date/time at the outputs are invalid. Output seconds. Value range: 0 … 59.</td>
</tr>
<tr>
<td>ERR=10</td>
<td>Setting of date/time is currently performed; this may take several PLC cycles.</td>
</tr>
<tr>
<td>ERR=11</td>
<td>Setting was not successful, please repeat (unknown request code).</td>
</tr>
<tr>
<td>ERR=12</td>
<td>Setting was not successful, please repeat (invalid mail parameter).</td>
</tr>
<tr>
<td>ERR=13</td>
<td>Setting was not successful, please repeat (request code cannot be executed).</td>
</tr>
</tbody>
</table>

**• Error when displaying date and time:**

<table>
<thead>
<tr>
<th>ERR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR=9</td>
<td>Date/time at the outputs are invalid. Outputs for date and time</td>
</tr>
</tbody>
</table>

The outputs are updated whenever a TRUE signal is present at the FREI input and the clock has been set once. During the setting the outputs for date and time are invalid.

If the AKT output is equal to TRUE, the outputs for the date and time are valid. In case of an error, an error identifier is output at output ERR.
Example

Declaration:

UHR_1 : UHR;
UHR_FREI AT %MX0.0 : BOOL;
UHR_SET AT %MX0.1 : BOOL;
UHR_ESEC AT %MW1000.0 : INT;
UHR_EMIN AT %MW1000.1 : INT;
UHR_EH AT %MW1000.2 : INT;
UHR_ETAG AT %MW1000.3 : INT;
UHR_EMON AT %MW1000.4 : INT;
UHR_EJHR AT %MW1000.5 : INT;
UHR_EWO AT %MW1000.6 : INT;
UHR_AKT AT %MX0.2 : BOOL;
UHR_ERR AT MW1000.7 : INT;
UHR_ASEC AT MW1000.8 : INT;
UHR_AMIN AT MW1000.9 : INT;
UHR_AH AT MW1000.10 : INT;
UHR_ATAG AT MW1000.11 : INT;
UHR_AMON AT MW1000.12 : INT;
UHR_AJHR AT MW1000.13 : INT;
UHR_AWO AT MW1000.14 : INT;

Translation in ABB IL:

!BA 0
IBA 0
UHR
M0,0
MW1,0
UHR
M0,1
MW1,1
UHR_ESEC
ESEC
UHR_ASEC
UHR_EMIN
EMIN
UHR_AMIN
UHR_AH
AH
UHR_ETAG
ETAG
UHR_ATAG
UHR_EJHR
EJHR
UHR_AJHR
UHR_EWO
EWO
UHR_AWO
AWO

FBD:

ABB IL of example:

!BA 0
IBA 0
UHR
M0,0
MW1,0
UHR
M0,1
MW1,1
UHR
M0,2
MW1,2
UHR
M0,3
MW1,3
UHR
M0,4
MW1,4
UHR
M0,5
MW1,5
UHR
M0,6
MW1,6
UHR
M0,7
MW1,7
UHR
M0,8
MW1,8
UHR
M0,9
MW1,9
UHR
M0,10
MW1,10
UHR
M0,11
MW1,11
UHR
M0,12
MW1,12
UHR
M0,13
MW1,13
UHR
M0,14
MW1,14
Function call in IL


LD UHR_1.ERR
ST UHR_ERR
LD UHR_1.ASEC
ST UHR_ASEC
LD UHR_1.AMIN
ST UHR_AMIN
LD UHR_1.AH
ST UHR_AH
LD UHR_1.ATAG
ST UHR_ATAG
LD UHR_1.AMON
ST UHR_AMON
LD UHR_1.AJHR
ST UHR_AJHR
LD UHR_1.AWO
ST UHR_AWO
LD UHR_1.AKT
ST UHR_AKT

Note: In IL, the function call has to be performed in one line.

Function call in ST


UHR_ERR := UHR_1.ERR;
UHR_ASEC := UHR_1.ASEC;
UHR_AMIN := UHR_1.AMIN;
UHR_AH := UHR_1.AH;
UHR_ATAG := UHR_1.ATAG;
UHR_AMON := UHR_1.AMON;
UHR_AJHR := UHR_1.AJHR;
UHR_AWO := UHR_1.AWO;
UHR_AKT := UHR_1.AKT;
UNPACKING A WORD INTO BINARY VARIABLES

This function block unpacks the word variable at input E. Each bit of the input variable is assigned to one binary variable each (B0 ... B15) at the output.

UNPACK4 – max. 4 output variables
UNPACK8 – max. 8 output variables
UNPACK16 – max. 16 output variables

Block type

Function block without historical values

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>UNPACK</td>
<td>Instance name</td>
</tr>
<tr>
<td>E</td>
<td>INT</td>
<td>Word variable to be unpacked</td>
</tr>
<tr>
<td>N</td>
<td>INT</td>
<td>Number of bit</td>
</tr>
<tr>
<td>B0..B15</td>
<td>BOOL</td>
<td>Binary output variable</td>
</tr>
</tbody>
</table>

Description

This function block unpacks the word variable at input E. Each bit of the input variable is assigned to one binary variable each (B0 ... Bn-1) at the output.

E

The variable to be unpacked is specified at input E. Each bit (bit0 ... bit15) of this input variable is assigned to the assigned output variable (B0 ... Bn-1).

B0...B15

The assigned bits of the variable at input E are assigned to the binary outputs B0 ... B15.

Assignment:

Input variable Bit0 → B0
Input variable Bit1 → B1
... ... ...
Input variable Bit15 → B15
Example

Declaration:

UNPACK_4: UNPACK4;
UNPACK_E AT %MW1001.0: INT;
UNPACK_B0 AT %MX1.0: BOOL;
UNPACK_B1 AT %MX1.1: BOOL;
UNPACK_B2 AT %MX1.2: BOOL;
UNPACK_B3 AT %MX1.3: BOOL;

Translation in ABB IL:

!BA 0
UNPACK
E
#n
B0
B1
B2
B3

FBD:

```
UNPACK_4

UNPACK_E

4

E

B0

UNPACK_B0

B1

UNPACK_B1

B2

UNPACK_B2

B3

UNPACK_B3
```

ABB IL of example:

!BA 0
UNPACK
M0,0
#4
M1,0
M1,1
M1,2
M1,3

Function call in IL

```
CAL UNPACK_4(E := UNPACK_E, n := 4)
LD UNPACK_4.B0
ST UNPACK_B0
LD UNPACK_4.B1
ST UNPACK_B1
LD UNPACK_4.B2
ST UNPACK_B2
LD UNPACK_4.B3
ST UNPACK_B3
```

Function call in ST

```
UNPACK_4 (E := UNPACK_E, n := 4);
UNPACK_B0 :=UNPACK_4.B0
UNPACK_B1 :=UNPACK_4.B1
UNPACK_B3 :=UNPACK_4.B3;
```
UP-DOWN COUNTER

This function block is used to count pulses. During counting, the positive edge of the pulse is evaluated in each case. The counter is capable of counting up and down and the counting increment can be specified. It is possible to preset the counter content to an intermediate value.

Block type

Function block with historical values

Parameters

<table>
<thead>
<tr>
<th>Instance</th>
<th>VRZ</th>
<th>Instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREI</td>
<td>BOOL</td>
<td>Enabling of the block processing</td>
</tr>
<tr>
<td>ZV</td>
<td>BOOL</td>
<td>Pulse input, up counting</td>
</tr>
<tr>
<td>ZR</td>
<td>BOOL</td>
<td>Pulse input, down counting</td>
</tr>
<tr>
<td>DIFF</td>
<td>INT</td>
<td>Counter content change per positive edge (increment)</td>
</tr>
<tr>
<td>SET</td>
<td>BOOL</td>
<td>Set counter to an intermediate value</td>
</tr>
<tr>
<td>ZW</td>
<td>INT</td>
<td>Intermediate value</td>
</tr>
<tr>
<td>RESET</td>
<td>BOOL</td>
<td>Reset counter</td>
</tr>
<tr>
<td>A</td>
<td>INT</td>
<td>Output for counter content</td>
</tr>
</tbody>
</table>

Description

This function block is used to count pulses. During counting, the positive edge of the pulse is evaluated in each case. The counter is capable of counting up and down and the counting increment can be specified. It is possible to preset the counter content to an intermediate value.

FREI

Counting is enabled or disabled by means of the FREI input.

The following applies:

FREI = FALSE → Counting disabled
FREI = TRUE → Counting enabled

ZV

Each positive edge (FALSE → TRUE edge) at the input ZV increases the current counter content by the increment specified at input DIFF.

ZR

Each positive edge (FALSE → TRUE edge) at the input ZR decreases the current counter content by the increment specified at input DIFF.

DIFF

The increment for the counting operation is specified at input DIFF. The increment is the value by which the counter is changed at the input ZV or ZR with each positive edge.

SET

By means of a TRUE signal at input SET, the counter content is set to the value specified at input ZW. Counting is blocked as long as a TRUE signal is present at input SET. Setting is also effective when a TRUE signal is present at input FREI.

ZW

The value to which the counter content is set by a TRUE signal at input SET is specified at input ZW.

RESET

A TRUE signal at input RESET sets the counter content to the value 0. The reset input RES has the highest priority of all inputs.
The current counter content is available at output A.

If the counter reaches the positive or negative limit of the number range, the counter is limited to this value.

The inputs and the output can neither be duplicated nor negated/inverted.

Example

Declaration:

VRZ_1 : VRZ;
VRZ_FREI_AT %MX0.0 : BOOL;
VRZ_ZV_AT %MX0.1 : BOOL;
VRZ_ZR_AT %MX0.2 : BOOL;
VRZ_DIFF_AT %MW1000.0 : INT;
VRZ_SET_AT %MX0.3 : BOOL;
VRZ_ZW_AT %MW1000.1 : INT;
VRZ_RESET_AT %MX0.4 : BOOL;
VRZ_A_AT %MW1000.2 : INT;

Translation in ABB IL:

!BA 0
VRZ
RESET
DIFF
SET
ZW
FREI
ZV
ZR
A

FBD:

Function call in IL

CAL VRZ_1(FREI := VRZ_FREI,
ZV := VRZ_ZV,
ZR := VRZ_ZR,
DIFF := VRZ_DIFF,
SET := VRZ_SET,
ZW := VRZ_ZW,
RESET := VRZ_RESET)

LD VRZ_1.A
ST VRZ_A

Function call in ST

VRZ_1(FREI := VRZ_FREI, ZV := VRZ_ZV,
ZR := VRZ_ZR, DIFF := VRZ_DIFF,
SET := VRZ_SET, ZW := VRZ_ZW,
RESET := VRZ_RESET);

VRZ_A:=VRZ_1.A;
INTERRUPTION TASK VALIDATION

Enable or disable (interrupt) task routine.

**Block type**

Function block without historical values

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>VTASK</td>
<td>Instance name</td>
</tr>
<tr>
<td>Ene</td>
<td>BOOL</td>
<td>Interrupt validation</td>
</tr>
<tr>
<td>INTNUM</td>
<td>INT</td>
<td>Interrupt number</td>
</tr>
</tbody>
</table>

**Description**

The VTASK function validates the execution of an interruption whose name is specified in NAME variable.

In function block diagram, the block has the name of the interruption.

**Example**

**Declaration:**

VTASK_1 : VTASK;
VTASK_EN AT %MX0.0 : BOOL;

**Translation in ABB IL:**

!BA 0
VTASK
EN #INTNUM

**ABB IL of example:**

!BA 0
VTASK
M0,0 #1
... IPE (* Programmende *)
IBA 0
TASK #1
#1
... (* Code der Interrupttask *)
RET NOP

**Function call in IL**

CAL VTASK_1(enable := VTASK_enable, N := VTASK_N )

**Function call in ST**

VTASK_1 (enable := VTASK_enable, N := VTASK_N );
This function block generates, bit-by-bit, the AND combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output.

### Block type
Function

### Parameters
- **E1**: INT Operand 1
- **E2**: INT Operand 2
- **A**: INT Result of the AND combination

### Description
This function block generates, bit-by-bit, the AND combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output. The inputs and outputs can neither be duplicated nor negated.

### Example

#### Declaration:
- `WAND_E1 AT %MW1000.0 : INT;
- `WAND_E2 AT %MW1000.1 : INT;
- `WAND_A AT %MW1000.2 : INT;

#### FBD:

```
WAND_E1 := E1
WAND_E2 := E2
WAND_A := WAND
```

#### ABB IL of example:

```
!BA 0
WAND
E1
E2
A
```

#### Function call in IL

```
LD WAND_E1
WAND WAND_E2
ST WAND_A
```

#### Function call in ST

```
WAND_A := WAND(E1 := WAND_E1, E2 := WAND_E2);
```
**WORD TIME CONVERSION**

This function is used to set a TIME value through words. The words values are converted in a double word for timer functions.

**Block type**
- Function block without historical values

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>W_TIME</td>
<td>Instance name</td>
</tr>
<tr>
<td>H</td>
<td>INT</td>
<td>Hour value</td>
</tr>
<tr>
<td>M</td>
<td>INT</td>
<td>Minute value</td>
</tr>
<tr>
<td>SEC</td>
<td>INT</td>
<td>Second value</td>
</tr>
<tr>
<td>MS</td>
<td>INT</td>
<td>Millisecond value</td>
</tr>
<tr>
<td>TIMEE</td>
<td>DINT</td>
<td>Time value</td>
</tr>
</tbody>
</table>

**Description**

This function is used to set a time value through words. The words values are converted in a double word for timer functions. The time value is in millisecond.

**Input for the hours.**
- Value range: $0 \leq H \leq 273$

**Input for the minutes.**
- Value range: $0 \leq M \leq 32767$

**Input for the seconds.**
- Value range: $0 \leq SEC \leq 32767$

**Input for the milliseconds.**
- Value range: $0 \leq MS \leq 32767$

**Output for the converted time value is in milliseconds.**
- Value range: $0 \leq TIMEE \leq 986399999$

If one parameter is set to a negative value, the value used for the internal calculation is 0.
Example

Declaration:

W_TIME_H AT %MW1000.0 : INT;
W_TIME_M AT %MW1000.1 : INT;
W_TIME_SEC AT %MW1000.2 : INT;
W_TIME_MS AT %MW1000.3 : INT;
W_TIME_TIMEE AT %MD2000.0 : DINT;

Translation in ABB IL:

!BA 0
W_TIME
H
M
SEC
MS
TIMEE

ABB IL of example:

!BA 0
W_TIME
MW0,0
MW0,1
MW0,2
MW0,3
MD0,0

Function call in IL

CAL W_TIME1 ( H := W_TIME_H,
M := W_TIME_M,
SEC := W_TIME_SEC,
MS := W_TIME_MS)
LD W_TIME1.TIMEE
ST W_TIME_TIMEE

Function call in ST

W_TIME1 ( H := W_TIME_H,
M := W_TIME_M,
SEC := W_TIME_SEC,
MS := W_TIME_MS);
W_TIME_TIMEE := W_TIME1.TIMEE;
WORD TO DOUBLE WORD CONVERSION

The value of the word operand at input E is converted to a double word variable and the result is assigned to the double word operand at output A.

Block type
Function

Parameters

<table>
<thead>
<tr>
<th>E</th>
<th>INT</th>
<th>Word variable to be converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>DINT</td>
<td>Result of the conversion, double word variable</td>
</tr>
</tbody>
</table>

Description

The value of the word operand at input E is converted to a double word variable and the result is assigned to the double word operand at output A. The input and the output can neither be duplicated nor negated.

Value range for \( E_1 \):

\[ 8000_{16} \leq E_1 \leq 7\text{FFFF}_{16} \]
\[ -32768 \leq E_1 \leq 32767 \]

Example

Declaration:

WDW_E AT %MW1000.0 : INT;
WDW_A AT %MD2000.0 : DINT;

FBD:

```
WDW_E := WDW(E);
WDW_A := WDW(A);
```

Translation in ABB IL:

```
IBA 0
WDW
E
A
```

ABB IL of example:

```
IBA 0
WDW
MW0,0
MD0,0
```

Function call in IL

```
LD WDW_E
WDW
ST WDW_DW
```

Function call in ST

\[
WDW\_DW := WDW(WDW\_E);
\]
READ WORD WITH ENABLING

If there is a TRUE signal at input FREI, the value of the specified physical address is read and assigned to the operand at output A.

<table>
<thead>
<tr>
<th>Block type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>FREI</td>
<td>BOOL</td>
</tr>
<tr>
<td>OFF</td>
<td>INT</td>
</tr>
<tr>
<td>SEG</td>
<td>INT</td>
</tr>
<tr>
<td>A</td>
<td>INT</td>
</tr>
</tbody>
</table>

**Description**

If there is a TRUE signal at input FREI, the value of the specified physical address is read and assigned to the operand at output A.

No reading and no assignment is performed, if there is a FALSE signal at input FREI.

The inputs and the output can neither be duplicated nor negated.

**FREI**  
Processing of the block is enabled or disabled with the operand at input FREI.

The following applies:

FREI = FALSE → Processing disabled  
FREI = TRUE → Processing enabled

**OFF**  
The offset address to be read is specified at input OFF. This is specified as 16 bit address.

**SEG**  
The segment address to be read is specified at input ADRESSE. This is specified as 16 bit address.

**A**  
The value read is assigned to the operand at output A.
Example

Declaration:
WOL_FREI AT %MX0.0 : BOOL;
WOL_A AT %MW1000.0 : INT

Translation in ABB IL:
!BA 0
WOL
FREI
OFF
SEG

FBD:

<table>
<thead>
<tr>
<th>WOL_FREI</th>
<th>WOL_A</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>Seg</td>
</tr>
</tbody>
</table>

ABB IL of example:
!BA 0
WOL
M0,0
#0
#H197C

Function call in IL
LD  WOL_FREI
WOL  WOL_OFF, WOL_ADR
ST  WOL_A

Function call in ST
WOL_A := WOL(WOL_FREI,
             WOL_OFF,
             WOL_SEG);
OR COMBINATION, WORD

This function block generates, bit-by-bit, the OR combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output.

**Block type**
Function

**Parameters**
- E1 INT Operand 1
- E2 INT Operand 2
- A INT Result of the OR combination

**Description**
This function block generates, bit-by-bit, the OR combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output. The inputs and outputs can neither be duplicated nor negated.

**Example**

**Declaration:**
- WOR_E1 AT %MW1000.0 : INT;
- WOR_E2 AT %MW1000.1 : INT;
- WOR_A AT %MW1000.2 : INT;

**Translation in ABB IL:**
- !BA 0
- WOR
- E1
- E2
- A

**FBD:**

```
!BA 0
WOR
E1
E2
A
```

**ABB IL of example:**

```
!BA 0
WOR
MW0,0
MW0,1
MW0,2
```

**Function call in IL**

```
LD WOR_E1
WOR WOR_E2
ST WOR_A
```

**Function call in ST**

```
WOR_A := WOR(E1 := WOR_E1, E2 := WOR_E2);
```
XOR COMBINATION, WORD
WXOR S40

This function block generates, bit-by-bit, the XOR combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output.

Block type
Function

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>INT</td>
<td>Operand 1</td>
</tr>
<tr>
<td>E2</td>
<td>INT</td>
<td>Operand 2</td>
</tr>
<tr>
<td>A</td>
<td>INT</td>
<td>Result of the XOR combination</td>
</tr>
</tbody>
</table>

Description
This function block generates, bit-by-bit, the XOR combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output.

The inputs and outputs can neither be duplicated nor negated.

Example

Declaration:
```
WXOR_E1 AT %MW1000.0 : INT;
WXOR_E2 AT %MW1000.1 : INT;
WXOR_A AT %MW1000.2 : INT;
```

Translation in ABB IL:
```
!BA 0
WXOR
E1
E2
A
```

FBD:
```
WXOR
WXOR_E1 := E1
WXOR_E2 := E2
WXOR_A
```

ABB IL of example:
```
!BA 0
WXOR
MW0.0
MW0.1
MW0.2
```

Function call in IL
```
LD WXOR_E1
WXOR WXOR_E2
ST WXOR_A
```

Function call in ST
```
WXOR_A := WXOR(E1 := WXOR_E1,
               E2 := WXOR_E2);
```
Glossary

BOOL
Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit memory space are reserved.

DINT
DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

<table>
<thead>
<tr>
<th>Type</th>
<th>Lower limit</th>
<th>Upper limit</th>
<th>Memory space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DINT</td>
<td>-2147483648</td>
<td>2147483647</td>
<td>32 bit</td>
</tr>
</tbody>
</table>

Due to this, it is possible that information are lost during the conversion of greater data types to smaller data types.

DWORD
DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

<table>
<thead>
<tr>
<th>Type</th>
<th>Lower limit</th>
<th>Upper limit</th>
<th>Memory space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD</td>
<td>0</td>
<td>4294967295</td>
<td>32 bit</td>
</tr>
</tbody>
</table>

Due to this, it is possible that information are lost during the conversion of greater data types to smaller data types.

INT
INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

<table>
<thead>
<tr>
<th>Type</th>
<th>Lower limit</th>
<th>Upper limit</th>
<th>Memory space</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT:</td>
<td>-32768</td>
<td>32767</td>
<td>16 bit</td>
</tr>
</tbody>
</table>

Due to this, it is possible that information are lost during the conversion of greater data types to smaller data types.
WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

<table>
<thead>
<tr>
<th>Type</th>
<th>Lower limit</th>
<th>Upper limit</th>
<th>Memory space</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>0</td>
<td>65535</td>
<td>16 bit</td>
</tr>
</tbody>
</table>

Due to this, it is possible that information are lost during the conversion of greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can derived:

- Within functions, global variables can neither be read nor written.
- Within functions, it is not allowed to read or write absolute operands.
- Within functions, it is not allowed to call function blocks.

Function blocks

Function blocks are subroutines which can have as much inputs, outputs and internal variables as required. They are called by a program or by another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. This allows for example to realize counters which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters. A function has maximally one output parameter. Note that the output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

For all ABB function blocks it has to be observed that instance names may not be defined several times if different data sets should be called.

Example:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VRZ1</td>
<td>VRZ</td>
</tr>
<tr>
<td>VRZ2</td>
<td>VRZ</td>
</tr>
</tbody>
</table>
## Index

<table>
<thead>
<tr>
<th>A</th>
<th>IDLM 72</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDD S40 8</td>
<td>IDSB 74</td>
</tr>
<tr>
<td>ASV S40 10</td>
<td>IDSM 76</td>
</tr>
<tr>
<td>AWT S40 12</td>
<td>LIZU 78</td>
</tr>
<tr>
<td>AWTB S40 13</td>
<td>MOA 80</td>
</tr>
<tr>
<td></td>
<td>MOAT 82</td>
</tr>
<tr>
<td></td>
<td>MODMAST_B/W 84</td>
</tr>
<tr>
<td>B</td>
<td>MOK 92</td>
</tr>
<tr>
<td>Base S40</td>
<td>MUL2N 94</td>
</tr>
<tr>
<td>ADDD 8</td>
<td>MULD 96</td>
</tr>
<tr>
<td>AWT 12</td>
<td>MULDI 98</td>
</tr>
<tr>
<td>AWTB 13</td>
<td>NEGW 100</td>
</tr>
<tr>
<td>BCDDUAL 14</td>
<td>NPULSE 101</td>
</tr>
<tr>
<td>BEG 17</td>
<td>PACK 103</td>
</tr>
<tr>
<td>BMELD 19</td>
<td>PDM 105</td>
</tr>
<tr>
<td>CONFIO 22</td>
<td>PI 107</td>
</tr>
<tr>
<td>COPY 25</td>
<td>PIDT1 112</td>
</tr>
<tr>
<td>CS31CO 27</td>
<td>RS 118</td>
</tr>
<tr>
<td>CS31QU 35</td>
<td>SINIT 119</td>
</tr>
<tr>
<td>CTU 37</td>
<td>SQRTD 122</td>
</tr>
<tr>
<td>CTUH 39</td>
<td>SQRTW 124</td>
</tr>
<tr>
<td>DIN 42</td>
<td>SR 126</td>
</tr>
<tr>
<td>DIVD 44</td>
<td>SUBD 127</td>
</tr>
<tr>
<td>DOUT 46</td>
<td>TIME_W 129</td>
</tr>
<tr>
<td>DRUCK 48</td>
<td>TOF 131</td>
</tr>
<tr>
<td>DUALBCD 52</td>
<td>TON 133</td>
</tr>
<tr>
<td>DWAND 54</td>
<td>TP 135</td>
</tr>
<tr>
<td>DWOR 55</td>
<td>UHR 137</td>
</tr>
<tr>
<td>DWW 56</td>
<td>UNPACK 142</td>
</tr>
<tr>
<td>DWXOR 57</td>
<td>VRZ 144</td>
</tr>
<tr>
<td>EMAS 58</td>
<td>VTASK 146</td>
</tr>
<tr>
<td>ESV 62</td>
<td>W_TIME 148</td>
</tr>
<tr>
<td>FKG 64</td>
<td>WAND 147</td>
</tr>
<tr>
<td>I_MINUS 66</td>
<td>WDW 150</td>
</tr>
<tr>
<td>I_PLUS 68</td>
<td>WOL 151</td>
</tr>
<tr>
<td>IDLB 70</td>
<td>WOR 153</td>
</tr>
</tbody>
</table>
WXOR 154
BCDDUAL S40 14
BEG S40 17
BME LD S40 19

C
CONFIO S40 22
COPY S40 25
CS31CO S40 27
CS31QU S40 35
CTU S40 37
CTUH S40 39

D
DIN S40 42
DIVD S40 44
DOUT S40 46
DRUCK S40 48
DUALBCD S40 52
DWAND S40 54
DWOR S40 55
DWW S40 56
DWXOR S40 57

E
EMAS S40 58
ESV S40 62

F
FKG S40 64
Function blocks 3
Functions 3

I
I_MINUS S40 66
I_PLUS S40 68
IDLB S40 70
IDLM S40 72
IDSB S40 74

L
IDSM S40 76
LIZU S40 78

M
MOA S40 80
MOAT S40 82
MODMAST_B/W S40 84
MOK S40 92
MUL2N S40 94
MULD S40 96
MULDI S40 98

N
NEGW S40 100
NPULSE S40 101

P
PACK S40 103
PDM S40 105
PI S40 107
PIDT1 S40 112

R
RS S40 118

S
SINIT S40 119
SQR TD S40 122
SQR TW S40 124
SR S40 126
SUBD S40 127

T
TIME_W S40 129
TOF S40 131
TON S40 133
TP S40 135
U
UHR S40 137
UNPACK S40 142

V
VRZ S40 144
VTASK S40 146

W
W_TIME S40 148
WAND S40 147
WDW S40 150
WOL S40 151
WOR S40 153
WXOR S40 154