Unity Pro

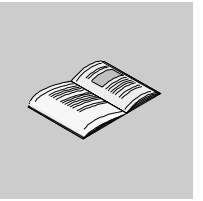
Safety Block Library
Enhanced Description

07/2011

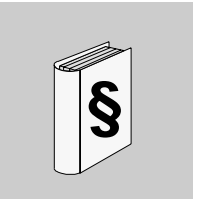
33004225.02

www.schneider-electric.com

Table of Contents



Safety Information



Important Information

NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of this symbol to a Danger or Warning safety label indicates that an electrical hazard exists, which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠ DANGER

DANGER indicates an imminently hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING

WARNING indicates a potentially hazardous situation which, if not avoided, **can result in** death or serious injury.

⚠ CAUTION

CAUTION indicates a potentially hazardous situation which, if not avoided, **can result in** minor or moderate injury.

CAUTION

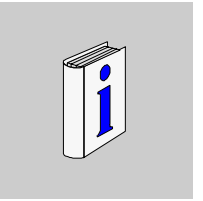
CAUTION, used without the safety alert symbol, indicates a potentially hazardous situation which, if not avoided, **can result in** equipment damage.

PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

About the Book



At a Glance

Document Scope

This document describes the following new function blocks of the Safety library:

- S_DISIL2
- S_AISIL2
- S_HSBY_SWAP
- S_SMOVE_WORD
- S_SMOVE_BIT

NOTE: The Unity Pro Safety library is only available in Unity Pro XLS.

This document is valid for Unity Pro XLS.

Validity Note

The data and illustrations found in this document are not binding. We reserve the right to modify our products in line with our policy of continuous product development. The information in this document is subject to change without notice and should not be construed as a commitment by Schneider Electric.

Related Documents

You can download these technical publications and other technical information from our website at *www.telemecanique.com*

Title of Documentation	Reference Number
Unity Pro Safety Block Library	33003873
Unity Pro Safety Manual	33003879
Unity Pro XLS Specifics	33003885
Unity Pro Operating Modes Manual	33003101
Unity Pro - Languages and Program Structure Reference Manual	35006144

Quantum with Unity Pro Discrete and Analog I/O Reference Manual	33002447
Modicon Quantum Hot Standby with Unity User Manual	35010533

You can download these technical publications and other technical information from our website at www.schneider-electric.com.

Product Related Information

Schneider Electric assumes no responsibility for any errors that may appear in this document. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of Schneider Electric.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When controllers are used for applications with technical safety requirements, please follow the relevant instructions.

Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this product related warning can result in injury or equipment damage.

User Comments

We welcome your comments about this document. You can reach us by e-mail at techcomm@schneider-electric.com.

General information

I

Overview

This section contains general information about the Safety library.

What's in this Part?

This part contains the following chapters:

Block Types and Their Applications

1

Overview

This chapter describes the different block types and their applications.

What's in this Chapter?

This chapter contains the following topics:

Block Types

Block Types

Different block types are used in Unity Pro. The general term for all block types is FFB.

There are the following types of block:
- Elementary Function (EF)
- Elementary Function Block (EFB)

Elementary Function

Elementary functions (EF) have no internal status. If the input values are the same, the value at the output is the same for all executions of the function, e.g. the addition of two values gives the same result at every execution.

An elementary function is represented in the graphical languages (FBD and LD) as a block frame with inputs and an output. The inputs are always represented on the left and the outputs always on the right of the frame. The name of the function, i.e. the function type, is shown in the center of the frame.

The number of inputs can be increased with some elementary functions.

Elementary Function Block

Elementary function blocks (EFB) have an internal status. If the inputs have the same values, the value on the output can have another value during the individual executions. For example, with a counter, the value on the output is incremented.

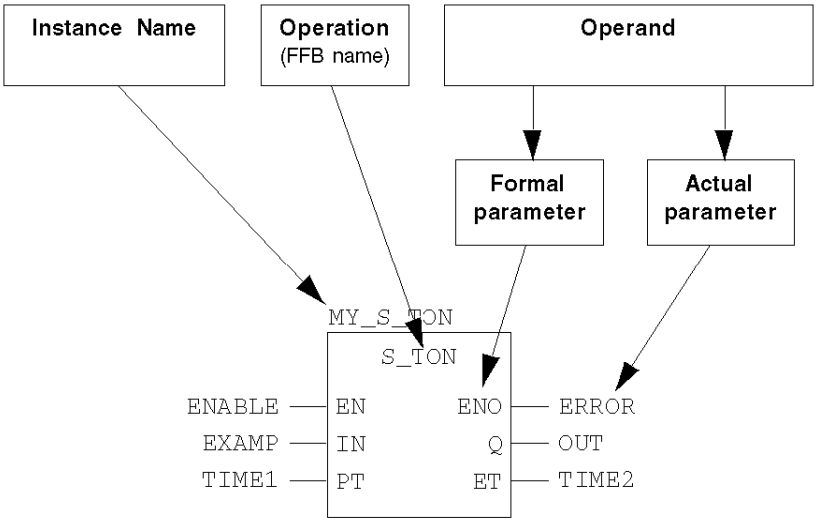
An elementary function block is represented in the graphical languages (FBD and LD) as a block frame with inputs and outputs. The inputs are always represented on the left and the outputs always on the right of the frame. The name of the function block, i.e. the function block type, is shown in the center of the frame. The instance name is displayed above the frame.

FFB Structure

Structure

Each FFB is made up of an operation (name of the FFB), the operands required for the operation (formal and actual parameters) and an instance name for elementary/derived function blocks.

Call of a function block in the FBD programming language:



Operation

The operation determines which function is to be executed with the FFB, e.g. shift register, conversion operations.

Operand

The operand specifies what the operation is to be executed with. With FFBs, this consists of formal and actual parameters.

Formal/Actual Parameters

Inputs and outputs are required to transfer values to or from an FFB. These are called formal parameters.

Objects are linked to formal parameters; these objects contain the current process states. They are called actual parameters.

At program runtime, the values from the process are transferred to the FFB via the actual parameters and then output again after processing.

The data type of the actual parameters must match the data type of the input/output (formal parameters). The only exceptions are generic inputs/outputs whose data type is determined by the actual parameter. If all actual parameters consist of literals, a suitable data type is selected for the function block.

EN and ENO

Description

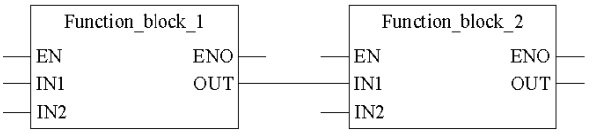
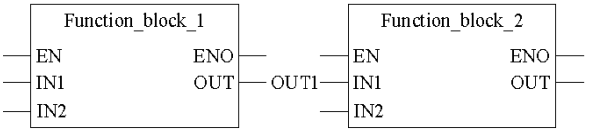
An EN input and an ENO output can be configured for all FFBs.

If the value of EN is ...	then ...
0 when the FFB is called up,	the algorithms defined by the FFB are not executed and ENO is set to 0.
1 when the FFB is called up,	the algorithms defined by the FFB are executed. After the algorithms have been executed successfully, the value of ENO is set to 1. **Note:** If an error occurs when executing these algorithms, ENO is set to 0.

Examples

The following tables describe examples if ENO is set to 0 (caused by EN=0 or an error during execution).

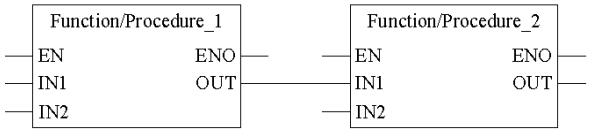
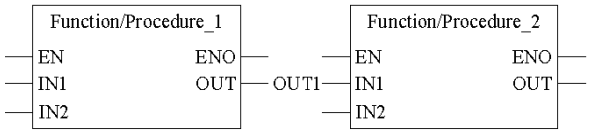
Function blocks

Example	Description
EN/ENO handling with function blocks that (only) have 1 link as an output parameter: 	If EN from FunctionBlock_1 is set to 0, the output connection OUT from FunctionBlock_1 retains the status it had in the last correctly executed cycle.
EN/ENO handling with function blocks that have 1 variable and 1 link as output parameters: 	If EN from FunctionBlock_1 is set to 0 • the output connection OUT from FunctionBlock_1 retains the status it had in the last correctly executed cycle; • the variable OUT1 on the same pin, either retains its previous status or can be changed externally without influencing the connection; • the variable and the link are saved independently of each other.

Functions/Procedures

As defined in IEC61131-3, the outputs from deactivated functions (EN-input set to 0) is undefined. The same applies to procedures.

Here nevertheless an explanation of the output statuses in this case:

Example	Description
EN/ENO handling with function/procedure blocks that (only) have 1 link as an output parameter:  	If EN from Function/Procedure_1 is set to 0, the output connection OUT from Function/Procedure_1 is also set to 0.
EN/ENO handling with function/procedure blocks that have 1 variable and 1 link as output parameters:  	If EN from Function/Procedure_1 is set to 0 • the output connection OUT from Function/Procedure_1 is also set to 0; • the variable OUT1 on the same pin retains its previous value. **Note:** In this way it is possible for the variable and the link to have different values.

The output behavior of the FFBs does not depend on whether the FFBs are called up without EN/ENO or with EN=1.

Conditional/Unconditional FFB Call

Unconditional or conditional calls are possible with each FFB. The condition is realized by pre-linking the input EN.

- EN connected
  conditional calls (the FFB is only processed if EN = 1)
- EN shown, hidden, and marked TRUE, or shown and not occupied
  unconditional calls (FFB is always processed)

Availability of the Blocks on Different Hardware Platforms

2

Availability of the Block on the Various Hardware Platforms

Introduction

Not all blocks are available on all hardware platforms. The blocks available on your hardware platform can be found in the following tables.

Comparison

Availability of the blocks

Block Name	Block Type	Defined in IEC 61131-3	Premium/M340	Quantum	Safety Quantum
S_EQ_***	EF	+	-	-	+
S_GE_***	EF	+	-	-	+
S_GT_***	EF	+	-	-	+
S_LE_***	EF	+	-	-	+
S_LT_***	EF	+	-	-	+
S_NE_***	EF	+	-	-	+
Legend:					
+	Yes				
-	No				

High Availability

Availability of the blocks

Block Name	Block Type	Defined in IEC 61131-3	Premium/M340	Quantum	Safety Quantum
S_DISIL2	EF	-	-	-	+
S_AISIL2	EF	-	-	-	+
Legend:					
+	Yes				
-	No				

Hot Standby

Availability of the blocks

Block Name	Block Type	Defined in IEC 61131-3	Premium/M340	Quantum	Safety Quantum
S_HSBY_SWAP	EF	-	-	-	+
Legend:					
+	Yes				
-	No				

Logic

Availability of the blocks

Block Name	Block Type	Defined in IEC 61131-3	Premium/M340	Quantum	Safety Quantum
S_AND_***	EF	+	-	-	+
S_F_TRIG	EFB	+	-	-	+
S_NOT	EF	+	-	-	+
S_OR_***	EF	+	-	-	+
S_R_TRIG	EFB	+	-	-	+
S_ROL_***	EF	+	-	-	+
S_ROR_***	EF	+	-	-	+
S_RS	EFB	+	-	-	+
S_SHL_***	EF	+	-	-	+
S_SHR_***	EF	+	-	-	+

Block Name	Block Type	Defined in IEC 61131-3	Premium/M340	Quantum	Safety Quantum
S_SR	EFB	+	-	-	+
S_XOR_***	EF	+	-	-	+
Legend:					
+	Yes				
-	No				

Mathematics

Availability of the blocks

Block Name	Block Type	Defined in IEC 61131-3	Premium/M340	Quantum	Safety Quantum
S_ABS_***	EF	+	-	-	+
S_ADD_***	EF	+	-	-	+
S_DIV_***	EF	+	-	-	+
S_MOVE_***	EF	+	-	-	+
S_SMOVE_BIT	EF	-	-	-	+
S_SMOVE_WORD	EF	-	-	-	+
S_MUL_***	EF	+	-	-	+
S_NEG_***	EF	+	-	-	+
S_SIGN_***	EF	+	-	-	+
S_SUB_***	EF	+	-	-	+
Legend:					
+	Yes				
-	No				

Statistical

Availability of the blocks

Block Name	Block Type	Defined in IEC 61131-3	Premium/M340	Quantum	Safety Quantum
S_LIMIT_***	EF	+	-	-	+
S_MAX_***	EF	+	-	-	+
S_MIN_***	EF	+	-	-	+

Block Name	Block Type	Defined in IEC 61131-3	Premium/M340	Quantum	Safety Quantum
S_MUX_***	EF	+	-	-	+
S_SEL	EF	+	-	-	+
Legend:					
+	Yes				
-	No				

Timers and Counter

Availability of the blocks

Block Name	Block Type	Defined in IEC 61131-3	Premium/ M340	Quantum	Safety Quantum
S_CTD_***	EFB	+	-	-	+
S_CTU_***	EFB	+	-	-	+
S_CTUD_***	EFB	+	-	-	+
S_TOF	EFB	+	-	-	+
S_TON	EFB	+	-	-	+
S_TP	EFB	+	-	-	+
Legend:					
+	Yes				
-	No				

Type to Type

Availability of the blocks

Block Name	Block Type	Defined in IEC 61131-3	Premium/ M340	Quantum	Safety Quantum
S_BIT_TO_BYTE	EF	+	-	-	+
S_BIT_TO_WORD	EF	+	-	-	+
S_BOOL_TO_***	EF	+	-	-	+
S_BYTE_TO_BIT	EF	+	-	-	+
S_BYTE_TO_***	EF	+	-	-	+
S_DINT_TO_***	EF	+	-	-	+
S_DWORD_TO_***	EF	+	-	-	+

Block Name	Block Type	Defined in IEC 61131-3	Premium/ M340	Quantum	Safety Quantum
S_INT_TO_***	EF	+	-	-	+
S_TIME_TO_UDINT	EF	+	-	-	+
S_UDINT_TO_***	EF	+	-	-	+
S_UINT_TO_***	EF	+	-	-	+
S_WORD_TO_BIT	EF	+	-	-	+
S_WORD_TO_***	EF	+	-	-	+
Legend:					
+	Yes				
-	No				

High Availability

II

Introduction

This section describes the elementary functions and elementary function blocks of the `High Availability` family.

What's in this Part?

This part contains the following chapters:

S_DISIL2: High Availiability for Safety Digital Inputs

3

Introduction

This chapter describes the `S_DISIL2` block.

What's in this Chapter?

This chapter contains the following topics:

Description

Function Description

This function block is used in high-availability architecture with redundant safety digital input (140 SDI 953 00S) modules. It continuously compares the integrity of both SDI modules and selects the data to be retrieved based on that comparison. By default, the data of module 1 are used as long as they are healthy.

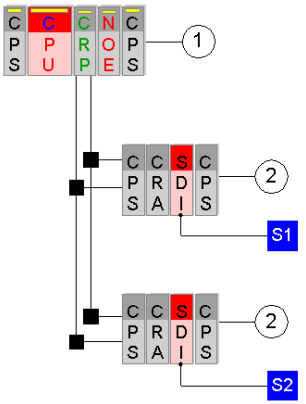
To improve availability by using the `S_DISIL2` safety function blocks, it is recommended to respect the following rules:

- Use 2 SDI modules 140 SDI 953 00S. (For details on this module see the Quantum with Unity Pro Discrete and Analog I/O Reference Manual.)
- Use 2 remote drops, each containing one of the SDI modules.
- Use 1 or 2 sensors.

Examples of High-Availability Architectures

The following drawings represent examples of high-availability architectures with 1 or 2 sensors.

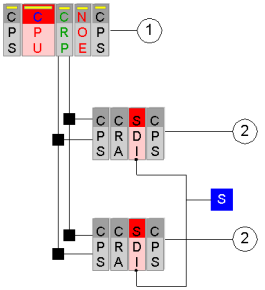
Architecture A: 2 sensors, 2 safety input modules, 2 remote drops



1 local rack
2 remote drops

In architecture A one SDI module processes the signal from sensor S1 whereas the other SDI module processes the signal from sensor S2.

Architecture B: 1 sensor, 2 safety input modules, 2 remote drops



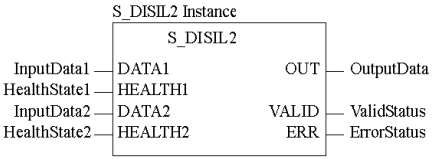
1 local rack
2 remote drops

In architecture B both SDI modules process the same signal from sensor S.

Should one of the SDI modules fail, the other will feed the signal from the sensor to the CPU.

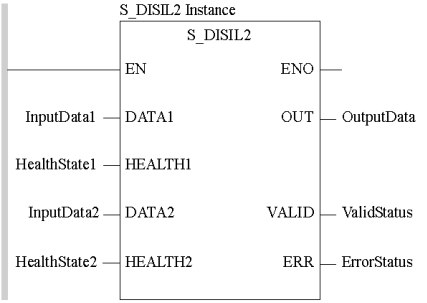
Representation in FBD

Representation



Representation in LD

Representation



Parameter Description

The S_DISIL2 function block consists of the following input and output parameters.

Input parameters:

- DATA
- HEALTH

Output parameters:

- OUT
- VALID
- ERR

Input Parameters

The function block retrieves the data and the health information out of each SDI module.

General description of the DATA and HEALTH parameters

Parameter	Data Type	Meaning
DATA	WORD	Each bit represents one of the 16 channels of the SDI module.
HEALTH	WORD	Each bit represents the health of one channel of the SDI module. 1: channel is healthy or valid 0: channel is unhealthy or invalid The health information is a combination of: • open circuit • safety state • process power supply failures

Retrieving DATA and HEALTH Information from the SDI Modules

The DATA1, DATA2 and HEALTH1, HEALTH2 parameters can be retrieved from the first and seventh word of the respective SDI module.

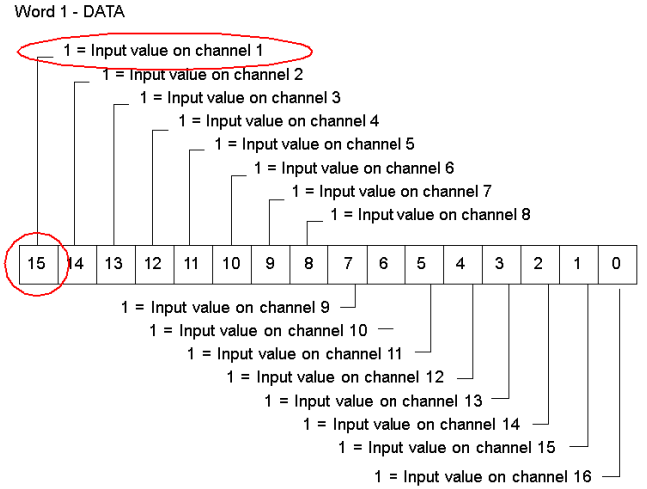
Parameter	X^{th} Word of SDI Module	Meaning
DATA1	first	This word contains all 16 channel data of SDI module 1.
HEALTH1	seventh	This word contains all 16 channel health data of the SDI module 1.
DATA2	first	This word contains all 16 channel data of SDI module 2.
HEALTH2	seventh	This word contains all 16 channel health data of the SDI module 2.

NOTE: There is a difference in the HEALTH parameter between S_DISIL2 and S_AISIL2 function blocks: Since S_AISIL2 works with only 1 channel of the SAI module, the HEALTH parameter consists of only 1 bit that represents the health status of 1 analog channel.

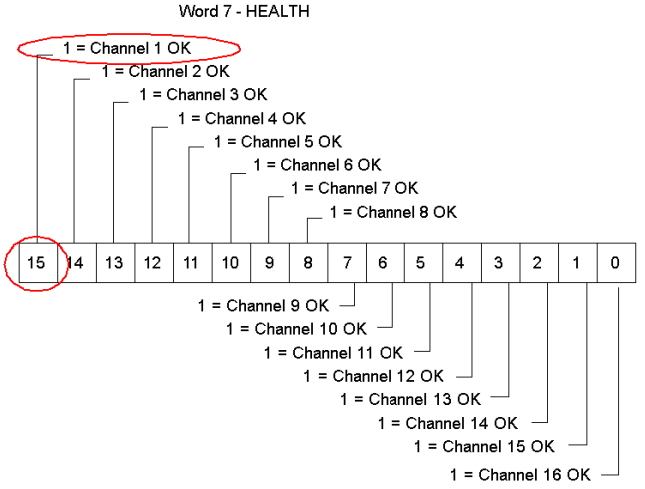
Graphical Representations of DATA and HEALTH Addressing

The S_DISIL2 safety function block parameters DATA and HEALTH are associated to respectively word 1 and word 2 on 1 SDI module and represent the 16 data bits and 16 health bits of the 16 channels of the module.

SDI module: addressing word 1 (DATA)



SDI module: addressing word 2 (HEALTH)



Bit=1 channel is healthy

Bit=0 channel is unhealthy (broken wire, invalid channel, process supply failed, CRC failed, incorrect exchange number

NOTE: Channel 1 is mapped to bit 15.

Output Parameters

Output parameter OUT

Parameter	Data Type	Meaning
OUT	WORD	The parameter OUT contains the evaluation of all 16 channels from the 2 SDI modules. By default, the function block first of all evaluates the channels of module 1. If these are healthy their data values will be used for OUT. If any of the channels of module 1 is not healthy, the respective channel on module 2 is checked. If this channel on module 2 is healthy, its data value will be used for OUT. **Result**: The OUT value may be a mixture of values from modules 1 and 2 if some channels of module 1 are not healthy. If a channel is neither healthy in module 1 nor in module 2, the bit of this channel in OUT will be set to 0 (safe state). If the complete module 1 is defect, the data of all channels will be taken from module 2 (if module 2 is healthy).

Example: OUT parameter for channel 1

Module 1, Channel 1	Module 2, Channel 1	OUT Data Value
healthy	not evaluated	value of module 1, channel 1
not healthy	healthy	value of module 2, channel 1
not healthy	not healthy	= 0 (safe state)

Output parameters VALID and ERR

Parameter	Data Type	Meaning
VALID	WORD	The parameter VALID shows the validity of the 16 channels individually in the 16 bits of the word VALID. A bit VALID is 1 if at least 1 of the 2 module channels is healthy.
ERR	WORD	This word will return a value between 0 and 7 depending on the health of the 2 modules and if there is a discrepancy between data of the 2 modules.

NOTE: The ERR word does not indicate which channel on the unhealthy module is faulty or in which channel there is a discrepancy. To discover that, just check the binary value of the corresponding DATA and HEALTH word.

Block Behavior Description

Summary: For each bit of the parameters DATA1/2 and HEALTH1/2 the following rules apply:

If...	Then ...
HEALTH1 is valid	DATA1 will be used for OUT.
HEALTH1 is not valid	HEALTH2 is checked and if valid, DATA2 will be used for OUT.
HEALTH1 and HEALTH2 are not valid	OUT is set to 0 (safe state).
HEALTH1 and HEALTH2 are valid	The bit of DATA1 will be used for OUT, regardless whether it is different from DATA2 or not. The error word ERR informs on the discrepancy between DATA1 and DATA2.

Conclusion: Module1, when healthy, is dominant.

If the bit in the health word HEALTH1 of SDI module 1 is valid, then the corresponding bit in DATA1 will be copied to the output word OUT of S_DISIL2.

State Table

This table shows the states for only 1 channel. Each channel is treated individually. Only the error code ERR indicates the global state of the modules, see below:

DATA1 **Data1. Channel X**	**HEALTH1** **Health1. Channel X**	**DATA2** **Data2. Channel X**	**HEALTH2** **Health2. Channel X**	**OUT** **OUT.Channel X**	**VALID** **VALID. Channel X**	**ERR**	**Remark**
0	0	0	0	0	0	Mod1, Mod2 defective	safe state
0	0	0	1	0	1	Mod1 defective	use Mod2 value
0	0	1	0	0	0	Mod1, Mod2 defective	safe state
0	0	1	1	1	1	Mod1 defective	use Mod2 value
0	1	0	0	0	1	Mod2 defective	use Mod1 value
0	1	0	1	0	1	OK	consistent values
0	1	1	0	0	1	Mod2 defective	use Mod1 value
0	1	1	1	0	1	discrepancy	use Mod1 value
1	0	0	0	0	0	Mod1, Mod2 defective	safe state
1	0	0	1	0	1	Mod1 defective	use Mod2 value
1	0	1	0	0	0	Mod1, Mod2 defective	safe state
1	0	1	1	1	1	Mod1 defective	use Mod2 value
1	1	0	0	1	1	Mod2 defective	use Mod1 value

DATA1 Data1. Channel X	HEALTH1 Health1. Channel X	DATA2 Data2. Channel X	HEALTH2 Health2. Channel X	OUT OUT.Channel X	VALID VALID. Channel X	ERR	Remark
1	1	0	1	1	1	discrepancy	use Mod1 value
1	1	1	0	1	1	Mod2 defective	use Mod1 value
1	1	1	1	1	1	OK	consistent values

Mod1 Digital input module 1
Mod2 Digital input module 2

Error Code Table

The following table explains the error codes:

Error Code	Meaning
0	OK All health bits of module 1 are valid and all health bits of module 2 are valid. All data bits of module 1 and module 2 are equal.
1	At least 1 health bit of module 1 is invalid and all health bits of module 2 are valid. All data bits of module 1 and module 2 are equal.
2	All health bits of module 1 are valid and at least 1 health bit of module 2 is invalid. All data bits of module 1 and module 2 are equal.
3	At least 1 health bit of module 1 is invalid and at least 1 health bit of module 2 is invalid. All data bits of module 1 and module 2 are equal.
4	There is a discrepancy between the input data. All health bits of module 1 are valid and all health bits of module 2 are valid. At least 1 data bit of module 1 is different to the data bit of module 2.
5	Discrepancy and 1 or more health bits in module 1 are invalid. At least 1 health bit of module 1 is invalid and all health bits of module 2 are valid. At least 1 data bit of module 1 is different to the data bit of module 2 (not necessarily the data bit of the unhealthy channel.
6	Discrepancy and 1 or more health bits in module 2 are invalid. All health bits of module 1 are valid and at least 1 health bit of module 2 is invalid. At least 1 data bit of module 1 is different to the data bit of module 2 (not necessarily the data bit of the unhealthy channel.
7	Discrepancy and 1 or more invalid health bits in module 1 and module 2. At least 1 health bit of module 1 is invalid and at least 1 health bit of module 2 is invalid. At least 1 data bit of module 1 is different to the data bit of module 2 (not necessarily the data bit of the unhealthy channel).

Fault Bits

The following table describes the meaning of each significant fault bit:

Fault Bit	Meaning
bit0	at least 1 channel of module 1 is invalid
bit1	at least 1 channel of module 2 is invalid
bit2	discrepancy between `DATA1` and `DATA2`

Hardware Configuration

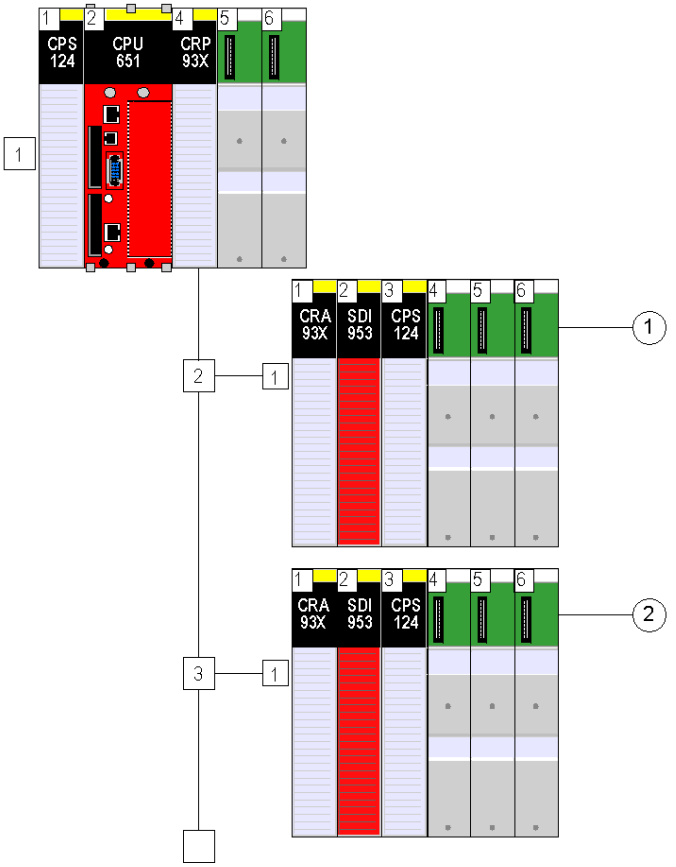
Overview

This section provides hardware configuration examples.

Application Example

As an example, the following application can be created and built upon.

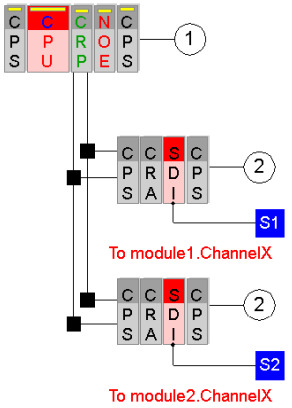
Configuration example



1 SDI module 1
2 SDI module 2

Hardware Configuration Example using 2 Sensors

The drawing below shows an application example with 2 sensors:



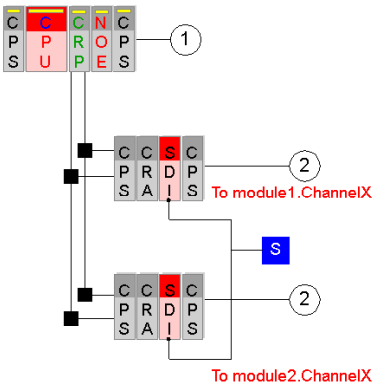
1 local rack
2 remote drops

In applications that use 2 sensors to measure the process value, these 2 sensors have to be consistently wired to the same channel on both modules.

Example: Wire sensor 1 to channel X on module 1 AND sensor 2 to channel X on module 2.

Hardware Configuration Example using 1 Sensor

The drawing below shows an application example with 1 sensor:



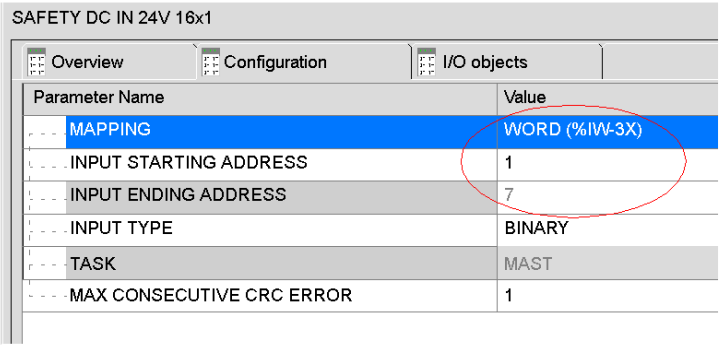
1 local rack
2 remote drops

In applications that use 1 sensor to measure the process value, this sensor has to be consistently wired to the same channel on both modules.

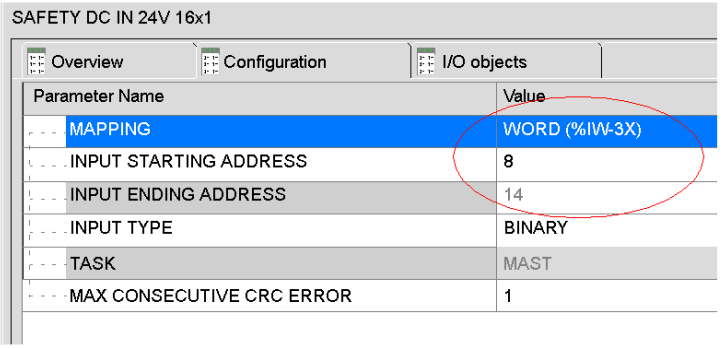
Retrieving Physical Addresses via Unity XLS Pro

Both SDI modules are mapped to integer values (`%IW`). You can display the physical addresses for each SDI module in the configuration window of Unity XLS Pro.

Physical addresses for SDI module 1: `%IW1...%IW7`



Physical addresses for SDI module 2: `%IW8...%IW14`



In the hardware example presented in this document, the data needed from each module is as follows:

SDI Module	**DATA**	**HEALTH**
SDI module 1	DATA1: %IW1	HEALTH1: %IW7
SDI module 2	DATA2: %IW8	HEALTH2: %IW14

After you have created the hardware configuration proceed with creating the necessary variables in the Unity Pro XLS **Data Editor** as described in the following section.

Software Configuration

Overview

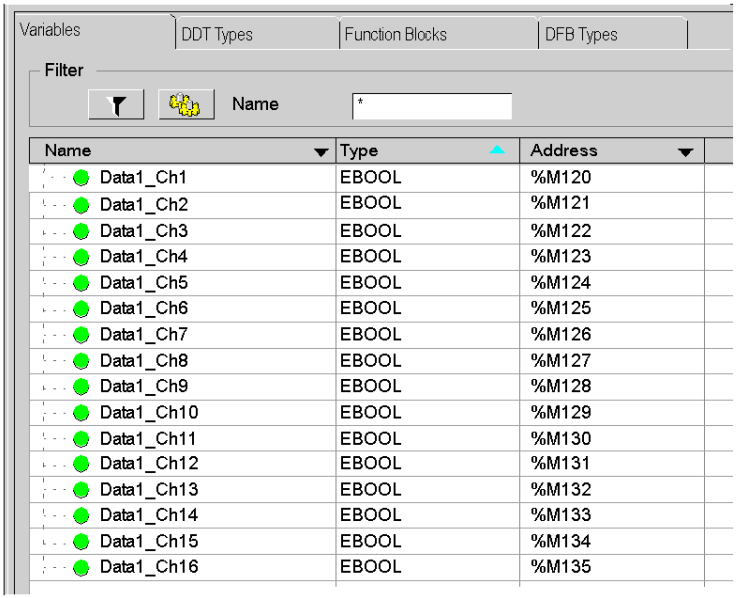
This section provides software configuration examples.

Creating Variables

After you have created the hardware configuration as described in the previous section, open the Unity Pro XLS **Data Editor** to create the necessary variables.

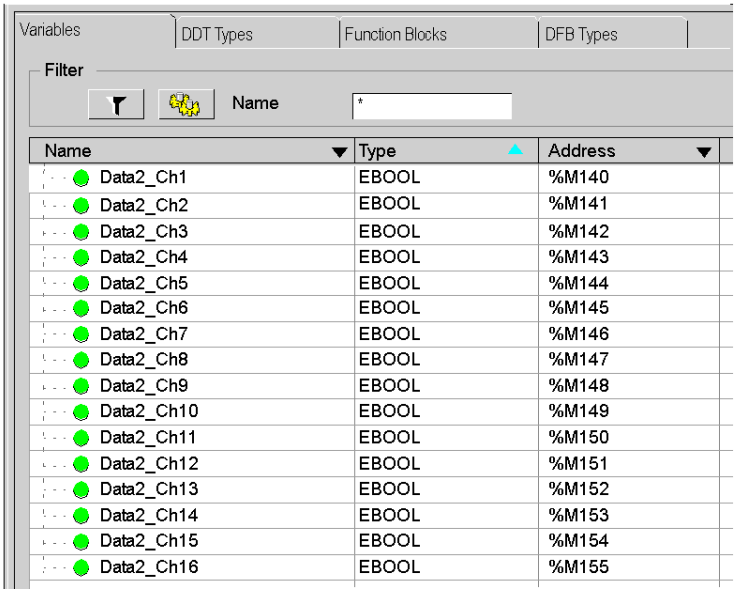
To be able to use the individual bits in the logic and to be able to force them, if necessary, the 16 data bits of each SDI module must be declared as `EBOOL` data types.

Creating variables for SDI module 1:



Name	Type	Address
Data1_Ch1	EBOOL	%M120
Data1_Ch2	EBOOL	%M121
Data1_Ch3	EBOOL	%M122
Data1_Ch4	EBOOL	%M123
Data1_Ch5	EBOOL	%M124
Data1_Ch6	EBOOL	%M125
Data1_Ch7	EBOOL	%M126
Data1_Ch8	EBOOL	%M127
Data1_Ch9	EBOOL	%M128
Data1_Ch10	EBOOL	%M129
Data1_Ch11	EBOOL	%M130
Data1_Ch12	EBOOL	%M131
Data1_Ch13	EBOOL	%M132
Data1_Ch14	EBOOL	%M133
Data1_Ch15	EBOOL	%M134
Data1_Ch16	EBOOL	%M135

Creating variables for SDI module 2:



Name	Type	Address
Data2_Ch1	EBOOL	%M140
Data2_Ch2	EBOOL	%M141
Data2_Ch3	EBOOL	%M142
Data2_Ch4	EBOOL	%M143
Data2_Ch5	EBOOL	%M144
Data2_Ch6	EBOOL	%M145
Data2_Ch7	EBOOL	%M146
Data2_Ch8	EBOOL	%M147
Data2_Ch9	EBOOL	%M148
Data2_Ch10	EBOOL	%M149
Data2_Ch11	EBOOL	%M150
Data2_Ch12	EBOOL	%M151
Data2_Ch13	EBOOL	%M152
Data2_Ch14	EBOOL	%M153
Data2_Ch15	EBOOL	%M154
Data2_Ch16	EBOOL	%M155

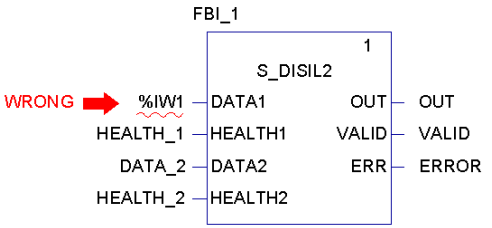
NOTE: `%IW` is interpreted as `INT` and therefore cannot be connected to the `S_DISIL2` function block. You must therefore declare a variable of type `WORD` at the correct address. Use flat addressing to be able to set `WORD` type. The `WORD` type is forbidden on topological addressing.

Example:

`Test : WORD : %IW\2.2\1.3.1.2` is not accepted

`Test : WORD : %IW8` is accepted

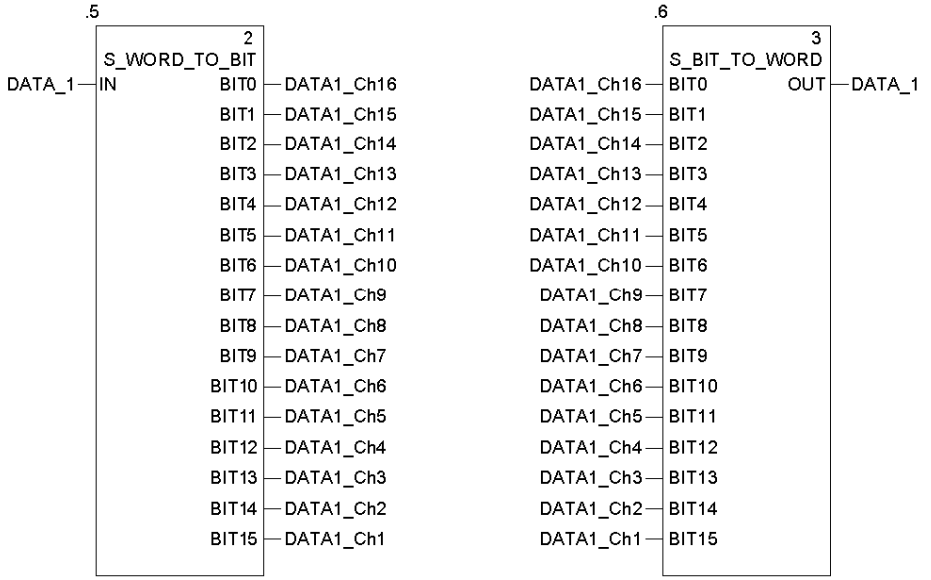
`%IW1` being interpreted as data type `INT`:



Example of Forcing Bits

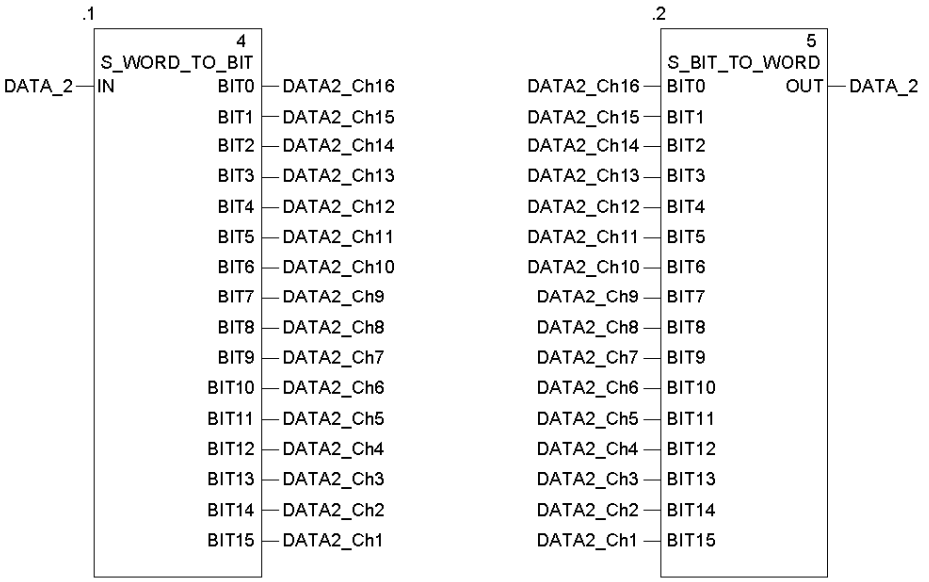
If the individual bits are available you can force each bit of the input words in order to simulate the behavior of the block.

The example below illustrates how to force the bits of word input DATA1:



Use the function block S_WORD_TO_BIT followed by S_BIT_TO_WORD to create a word DATA_1 of which you can force bits.

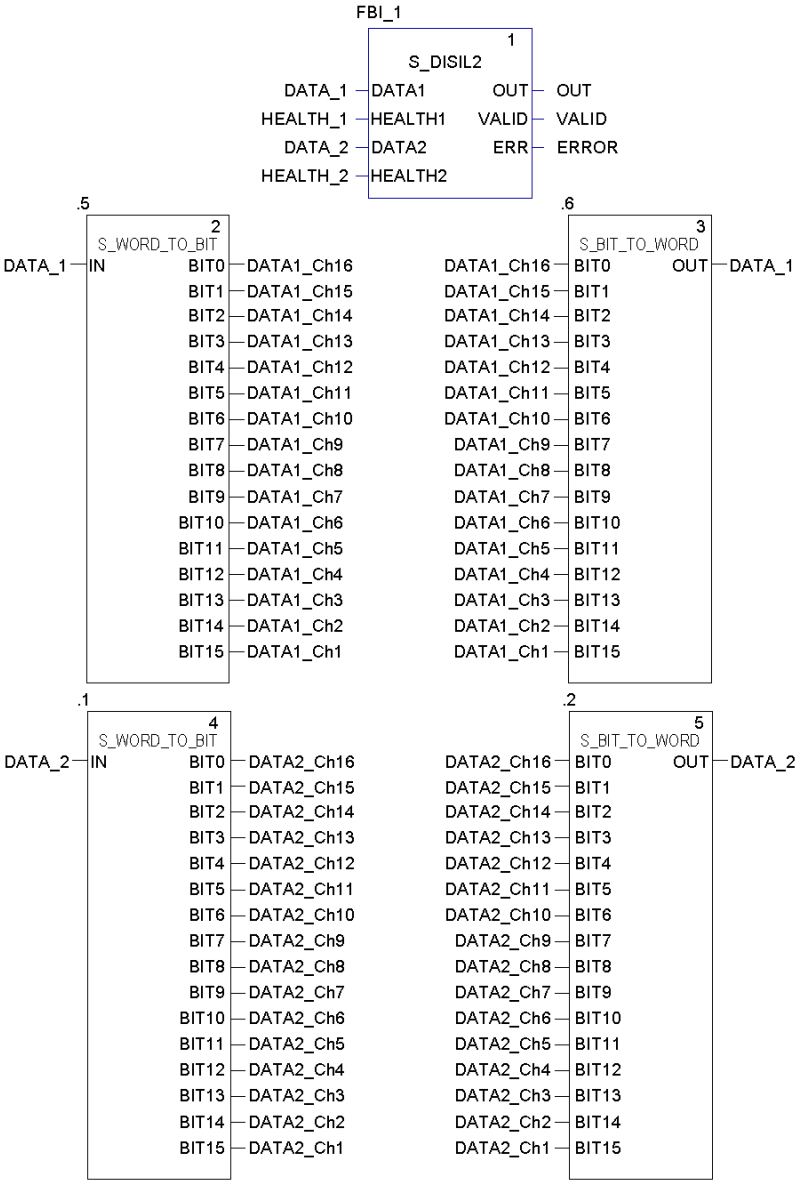
The example below illustrates how to force the bits of word input DATA2:



Use the function block S_WORD_TO_BIT followed by S_BIT_TO_WORD to create a word DATA_2 of which you can force bits.

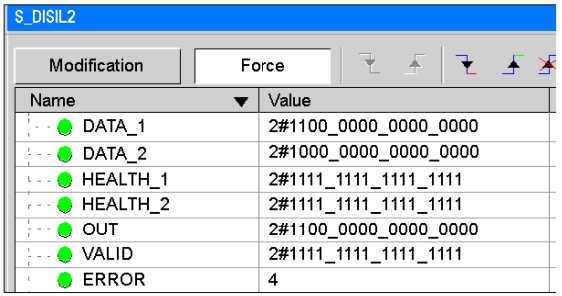
Animation Table Result

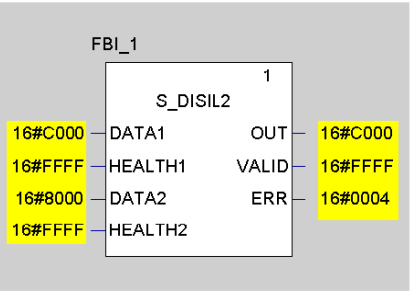
The complete program that will be used with the animation table is as follows:



Examples of S_DISIL2 Input and Output Values

Example 1:

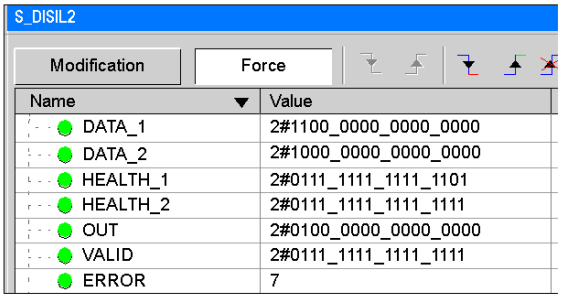


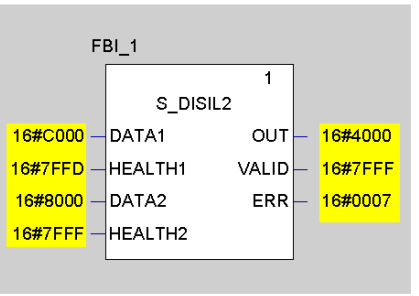


The above example indicates the following information:

- Both SDI modules are healthy (all bits of HEALTH1 and HEALTH2 are 1).
- Bit 14 of DATA1 is 1 while bit 14 of DATA2 is 0.
- The resulting error code is 4 which indicates that both modules are healthy but that there is a discrepancy between the 2 data words.
- The output value is set to DATA1.

Example 2:

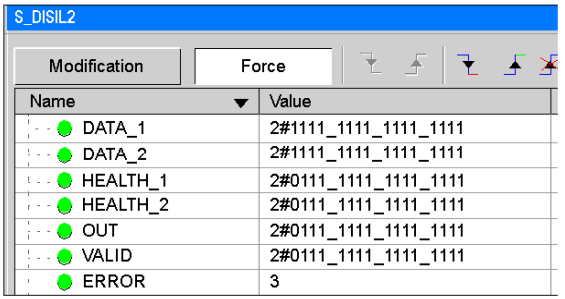


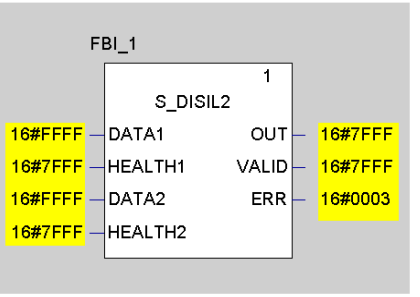


The above example indicates the following information:

- Bit 15 of HEALTH1 is 0 and bit 15 of HEALTH2 is 0.
- Bit 14 of DATA1 is 1 while bit 14 of DATA2 is 0.
- The resulting error code is 7 which indicates that there is a discrepancy on data bits between both modules (in this example: bit 14) in addition to the presence of at least 1 unhealthy bit in module 1 and in module 2.
- Output bits 0 to 14 are equal to DATA1, only unhealthy bit 15 is set to the safe state 0.

Example 3:





The above example indicates the following information:

- Bit 15 of `HEALTH1` is `0` and bit 15 of `HEALTH2` is `0` which means that channel 1 on both SDI modules is unhealthy.
- `DATA1` and `DATA2` are equal, so there is no discrepancy.
- The error code is `3` which indicates that at least 1 health bit of module 1 is invalid and at least 1 bit of module 1 is invalid and that all data bits of module 1 and module 2 are equal.
- Output bit 15 is `0` (safe state) because bit 15 on both modules is unhealthy.

S_AISIL2: High Availiability for Safety Analog Inputs

4

Introduction

This chapter describes the S_AISIL2 block.

What's in this Chapter?

This chapter contains the following topics:

Description

Function Description

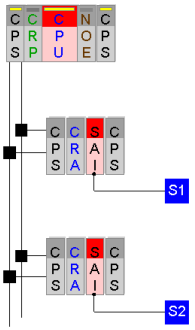
This function block is used in high-availability architectures with redundant safety analog input (140 SAI 940 00S) modules. It continuously compares the integrity of the 2 channels stemming from both SAI modules and selects the data to be retrieved based on that comparison.

To improve availability by using the `S_AISIL2` safety function blocks, it is recommended to respect the following rules:

- Use 2 SAI modules 140 SAI 940 00S. (For details on this module see the Quantum with Unity Pro Discrete and Analog I/O Reference Manual.)
- Use 2 remote drops, each containing one of the SAI modules.
- Use 2 sensors.

High-Availability Architecture

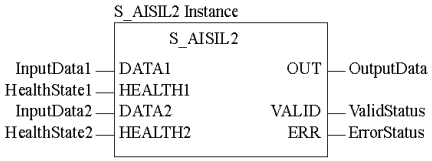
Example of a high-availability architecture with 2 sensors, 2 SAI modules, 2 remote drops



In this architecture 1 SAI module processes the signal from sensor S1 whereas the other SAI module processes the signal from sensor S2.

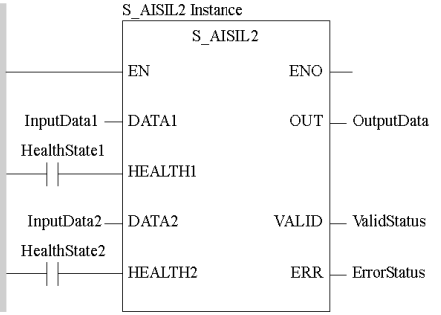
Representation in FBD

Representation



Representation in LD

Representation



Parameter Description

The S_AISIL2 function block consists of the following input and output parameters.

Input parameters:

- DATA
- HEALTH

Output parameters:

- OUT
- VALID
- ERR

Input Parameters

The function block retrieves the data and the health information out of each SAI module.

General description of the DATA and HEALTH parameters

Parameter	Data Type	Meaning
DATA	WORD	Each bit represents one of the 8 channels of the SAI module.
HEALTH	BIT	It represents the health of 1 channel of the SAI module. 1: channel is healthy or valid 0: channel is unhealthy or invalid The health information is a combination of: • open circuit • safety state • process power supply failures

Retrieving DATA and HEALTH Information from the SAI Modules

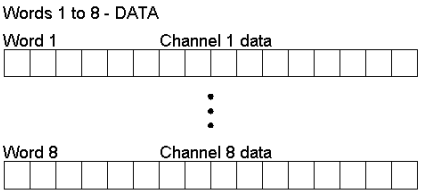
The DATA1 and DATA2 variables are mapped to one of the first 8 words of SAI module 1 and SAI module 2.

The HEALTH1 and HEALTH2 variables are each one bit located in the thirteenth word of SAI module 1 respectively SAI module 2.

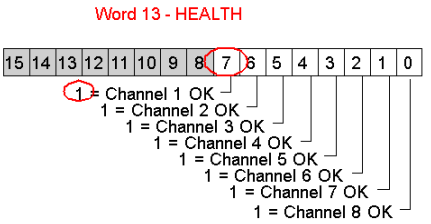
NOTE: The S_AISIL2 safety function block DATA is associated to 1 of the words 1 to 8 whereas the parameter HEALTH is associated to a bit in word 13 of the SAI module.

Graphical Representations of DATA and HEALTH Addressing

SAI module: addressing words 1 to 8 (DATA)



SAI module: addressing word 13 (HEALTH)



Bit 8 to 15: unused
Bit 0 to 7: used
Bit=1 channel is healthy
Bit=0 Channel is unhealthy (invalid channel, out of range, CRC failed, incorrect exchange number)

NOTE: Channel 1 is mapped to bit 7.

Output Parameters

Output parameter OUT

Parameter	Data Type	Meaning
OUT	WORD	=DATA1: The OUT word will always contain the value of DATA1 as long as the channel of SAI module 1 is healthy (HEALTH1 is valid). =DATA2: The OUT word will only contain the value of DATA2 if the channel of SAI module 1 is defective (HEALTH1 is invalid). =0: The value of the OUT word is set to 0 (safe state) when the channels of both SAI modules are defective (HEALTH1 and HEALTH2 are invalid).

Example: OUT parameter for channel of module 1

Channel of Module 1	Channel of Module 2	OUT Data Value
healthy	not evaluated	value of channel of module 1
not healthy	healthy	value of channel of module 2
not healthy	not healthy	= 0 (safe state)

Output parameters VALID and ERR

Parameter	Data Type	Meaning
VALID	WORD	When at least 1 of the HEALTH bits is valid, VALID will be 1.
ERR	WORD	This word will return a value between 0 and 3 depending on the states of the 2 modules.

Block Behavior Description

Summary: The following rules apply for the parameters DATA1/2 and HEALTH1/2 :

If...	Then ...
HEALTH1 is valid	DATA1 will be used for OUT.
HEALTH1 is not valid	HEALTH2 is checked and if valid, DATA2 will be used for OUT.
HEALTH1 and HEALTH2 are not valid	OUT is set to 0 (safe state).
HEALTH1 and HEALTH2 are valid	DATA1 will be used for OUT, regardless whether it is different from DATA2 or not. The error word ERR informs on the discrepancy between DATA1 and DATA2.

Conclusion: Module1, when healthy, is dominant.

If the health bit HEALTH1 of the channel of the SAI module 1 is 1, then DATA1 will be copied to the output word OUT of S_AISIL2.

State Table

Health1	Health2	Output	Valid	Error	Remark
0	0	0	0	channel in module 1 defective channel in module 2 defective	safe state
0	1	Data2	1	channel in module 1 defective	use value of channel in module 2
1	0	Data1	1	channel in module 2 defective	use value of channel in module 1
1	1	Data1	1	Both channels on both modules are OK	use value of channel in module 1

Error Code Table

The following table explains the error codes:

Error Code	Fault Description
0	valid
1	Module 1 is defective.
2	Module 2 is defective.
3	Module 1 and module 2 are defective.

Fault Bits

The following table describes the meaning of the 2 fault bits:

Fault Bit	Meaning
bit0	channel of module 1 is invalid
bit1	channel of module 2 is invalid

Hardware Configuration

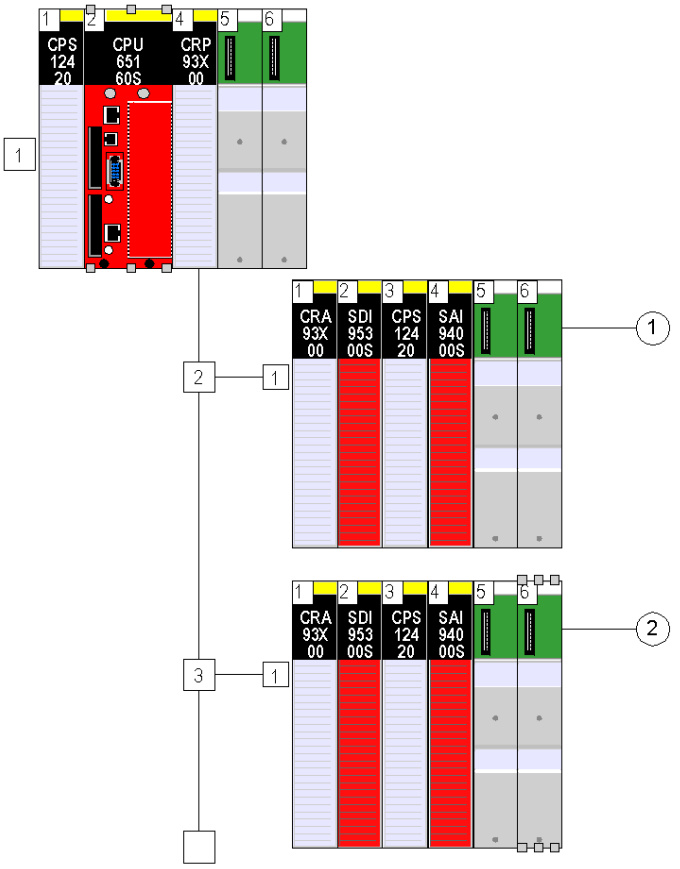
Overview

This section provides hardware configuration examples.

Application Example

As an example, the following application can be created and built upon.

Configuration example

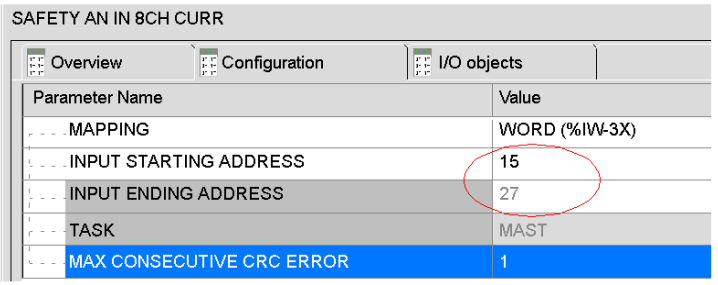


1 SAI module 1: sensor 1 connected to channel 1
2 SAI module 2: sensor 2 connected to channel 1

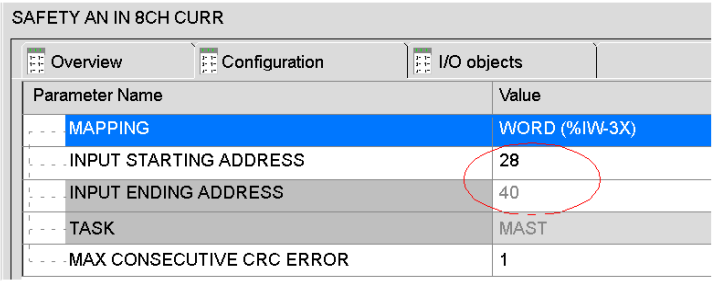
Retrieving Physical Addresses via Unity XLS Pro

Both SAI modules are mapped to integer values (%IW). You can display the physical addresses for each SAI module in the configuration window of Unity XLS Pro.

Physical addresses for SAI module 1: %IW15...%IW27



Physical addresses for SAI module 2: %IW28...%IW40



In the hardware example presented in this document, the data needed from each module is the data from channel 1 of each SAI module:

SAI Module	DATA	HEALTH
SAI module 1	DATA1: %IW15	HEALTH1: %IW27.7
SAI module 2	DATA2: %IW28	HEALTH2: %IW40.7

After you have created the hardware configuration proceed with creating the necessary variables in the Unity Pro XLS **Data Editor** as described in the following section.

Software Configuration

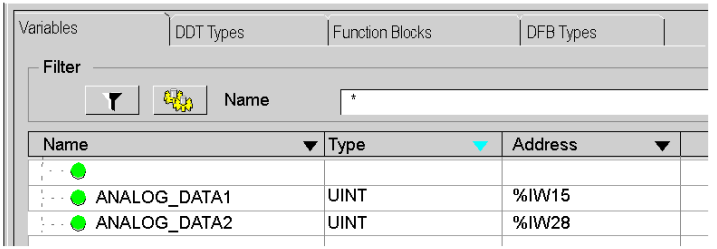
Overview

This section provides software configuration examples.

Creating Variables

After you have created the hardware configuration as described in the previous section, open the Unity Pro XLS **Data Editor** to create the necessary variables.

These variables are needed to feed the pins of the `S_AISIL2` safety function block.

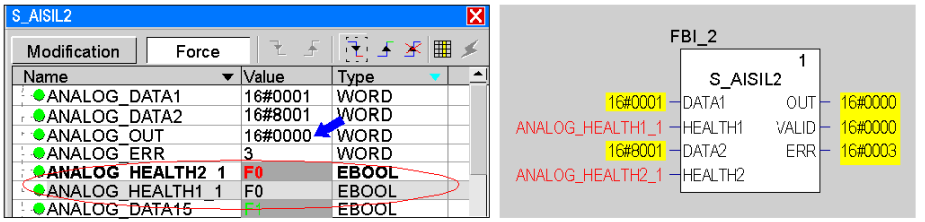


Animation Table Result

The following paragraphs show values forced in the Unity Pro XLS **Animation table** and their results.

Examples of `S_AISIL2` Input and Output Values

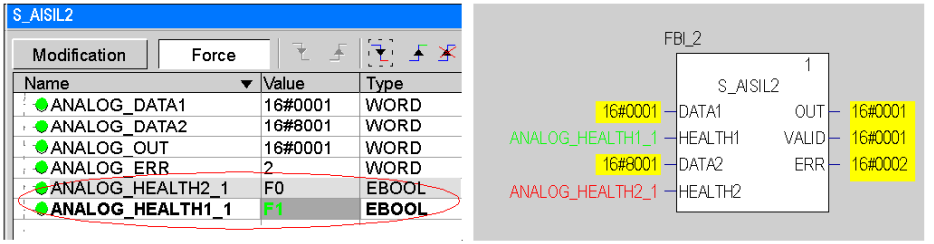
Example 1:



The above example indicates the following information:

- Both SAI modules are defective (`HEALTH1` and `HEALTH2` are set to `0`).
- The resulting error code is `3`.
- The output value is set to `0`. (Remember: In case both modules are defective, `OUT` switches to the safe state.)

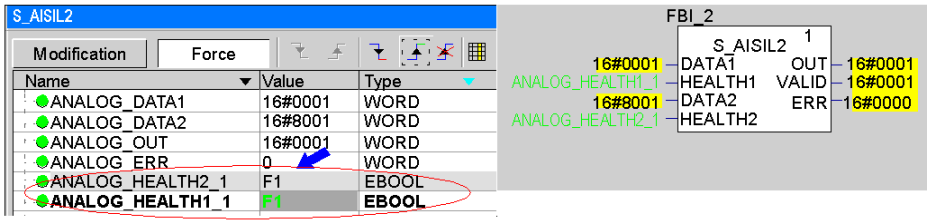
Example 2:



The above example indicates the following information:

- SAI module 2 is defective (bit 0 of `HEALTH2` is set to `0`).
- The resulting error code is `2`.
- The output value `OUT` is set to the value of `DATA1` (`16#0001`).

Example 3:



The above example indicates the following information:

- Both SAI modules are healthy (`HEALTH1` is `1` and `HEALTH2` is `1`).
- The resulting error code is `0`.
- The output value `OUT` is set to the value of `DATA1` (`16#0001`).

Mathematics

III

Introduction

This section describes the function blocks S_MOVE_WORD and S_MOVE_BIT of the Mathematics family.

What's in this Part?

This part contains the following chapters:

S_SMOVE_WORD: Assignment

5

Description

Function Description

In general, the memory of the Safety PLC is write protected, i.e. write requests are not executed.

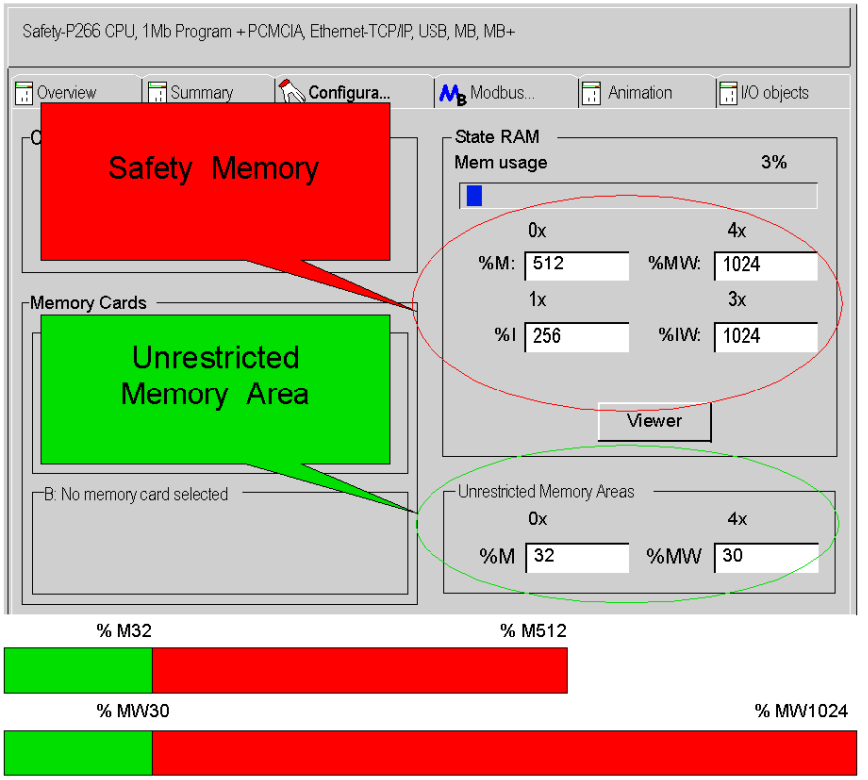
To allow exceptions from this rule (which are fully in the responsibility of the customer), e.g. to change set points from an HMI, the UMA (unrestricted memory area) for %M and %MW values has been introduced.

Definition of UMA

The UMA is characterized as follows:

- The UMA is a part of the Safety PLC's memory for %M and %MW values that is not write protected.
- Its addresses can be written from other PLCs or HMIs.
- Its configuration is performed in the **CPU property dialog** of Unity Pro XLS.

Configuring the memory of the Safety PLC (safety memory and UMA):



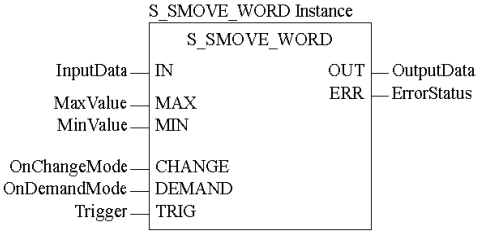
NOTE: `%M` and `%MW` in the UMA are located at the beginning of the memory range. You should thus make sure to reserve UMA space of the memory at the very beginning of the project.

Purpose of `S_SMOVE_WORD`

You cannot use variables of type `WORD` from the UMA ranges directly in the safety logic. To use these variables, the safety function block `S_SMOVE_WORD` is required. For variables of type `BOOL` from the UMA range of `%MW` use the `S_SMOVE_BIT` function block in the safety logic.

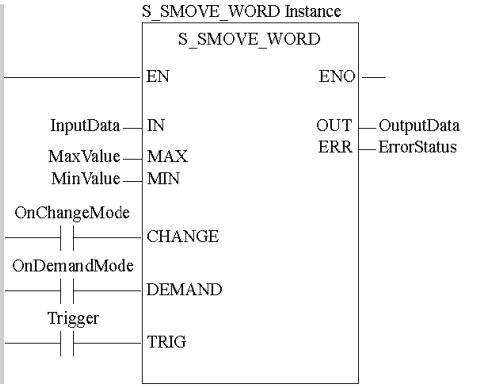
Representation in FDB

Representation



Representation in LD

Representation



Parameter Description

Description of the input parameters

Parameter	Data Type	Meaning
IN	WORD	A variable of the unrestricted word range.
MAX	WORD	The maximum value permitted for IN. A literal can be connected or if a variable is used it must be mapped to a word in the safety memory range.
MIN	WORD	The minimum value permitted for IN. A literal can be connected or if a variable is used it must be mapped to a word in the safety memory range.

Parameter	Data Type	Meaning
CHANGE	BOOL	=1: The input data is transferred to the output if the input data has changed (as long as IN is in the valid range (MIN...MAX)). =0 or no data change: The output value is not written. =0 and DEMAND=0: called transparent mode, for more information see below. A literal can be connected or if a variable is used it must be mapped to a word in the safety memory range.
DEMAND	BOOL	=1 and rising edge at the TRIG input detected: The input data is transferred to the output value OUT (as long as IN is in the valid range (MIN...MAX)). Otherwise the output value remains unchanged. A literal can be connected or if a variable is used it must be mapped to a Boolean in the safety memory range.
TRIG	BOOL	Only used in demand mode. A rising edge at this input triggers the transfer of the input value to OUT (as long as IN is in the valid range (MIN...MAX)). A literal can be connected or if a variable is used it must be mapped to a Boolean in the safety memory range.

Description of output parameters

Parameter	Data Type	Meaning
OUT	WORD	This is the controlled word to be used in the safety logic. It is mapped to a word in the safety memory range.
ERR	WORD	If the actual value at the input is not within the range, the value of output OUT will not be changed and the output ERR will indicate an out of range (8A19h) error. It is mapped to a word in the safety memory range.

Block Behavior

The S_SMOVE_WORD function block may work according to the following 3 modes:

- transparent mode
- demand mode
- change mode

These modes will be described in the following paragraphs.

Transparent Mode

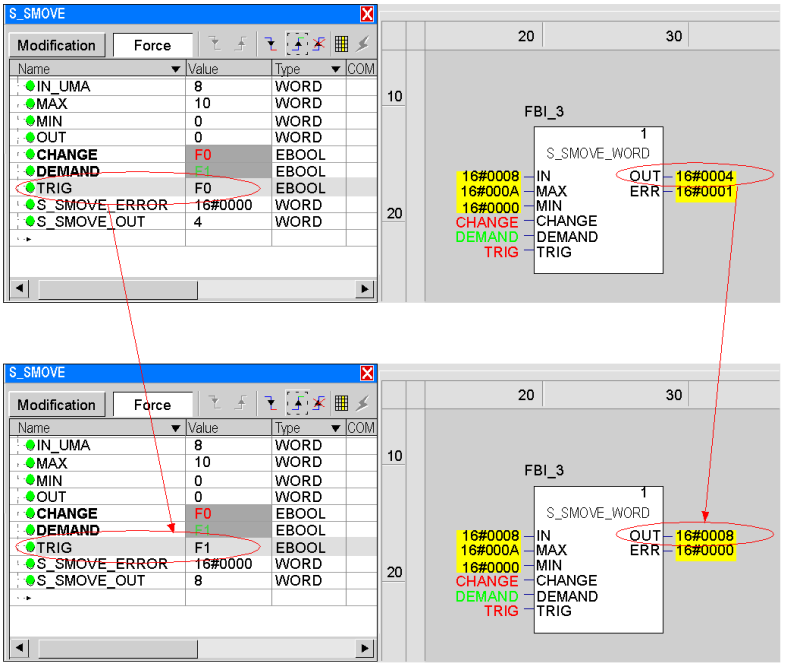
If...	Then...
CHANGE mode bit = 0 and DEMAND mode bit = 0 and the input value IN is within the valid range (MIN...MAX)	the value of the input word will be written to the output word in the safety memory.

NOTE: If no signals are connected to DEMAND, CHANGE and TRIG the function block is in transparent mode because non-connected inputs are treated as connected with 0.

Demand Mode

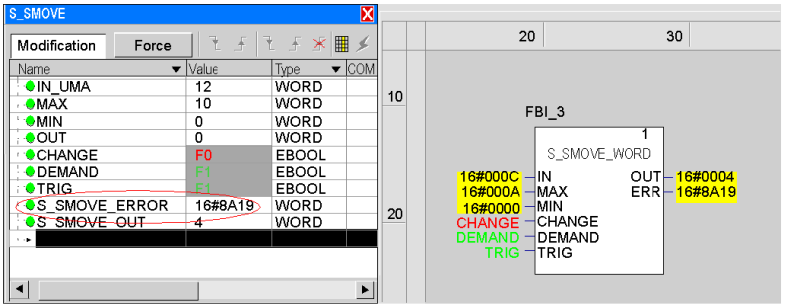
If...	Then...
there is a rising edge detected at the TRIG input and the input value IN is within the valid range (MIN...MAX)	the value of the input word will be written to the output word in the safety memory.

Example of demand mode with IN being within the defined range:



If...	Then...
the input value IN is out of range	the output is unchanged and error code 16#8A19 is shown in the error status word.

Example of demand mode with IN being out of the defined range:



Change Mode

In this mode only CHANGE is set to 1 (DEMAND is 0 and TRIG has no influence).

If...	Then...
the input value IN is within the valid range (MIN...MAX) and the input value IN is changing	the value of the input word will be written to the output word in the safety memory.

Notable Differences between Change and Transparent Mode

In change mode the function block only changes the value of the variable connected to the OUT if the input IN is changed. In transparent mode the value of the input IN will always be written to the output OUT. When doing modifications in online mode consider the following:

If...	Then...
you replace a variable connected to the output OUT by another variable	the newly connected variable will not change its current value until the input IN changes.
you replace a variable connected to the output OUT by another variable	the value of a link which is also connected to the output will become equal to the current value of the newly connected variable.
you replace a variable connected to the output OUT by a link	the value of the link will be 0 until the input IN changes.

Note for Change and Demand Mode

NOTE: Both DEMAND and CHANGE bits are set to 1. In this case the input will be written to the output when the input changes or when there is a rising edge on the trigger input, unless the input value is out of range.

State Table

Pin states and the respective mode

CHANGE	DEMAND	TRIG	IN	OUT	ERR	Mode
-	-	-	out of range (min, max)	old OUT value	E_Input_Valu_Out_Of_Range (16#8A19)	-
0	0	-	valid range	= IN	0	transparent
0	1	0->1	valid range	= IN	0	demand
0	1	not 0->1	valid range	old IN value	0	demand
1	0	-	valid range, changed	= IN	0	change
1	0	-	valid range, no change	OUT unchanged	0	change

CHANGE	DEMAND	TRIG	IN	OUT	ERR	Mode
1	1	0->1	valid range, no change	= IN	0	demand + change
1	1	not 0->1	valid range, no change	OUT unchanged	0	demand + change
1	1	0->1	valid range, changed	= IN	0	demand + change
1	1	not 0->1	valid range, changed	= IN	0	demand + change

S_SMOVE_BIT: Assignment

6

Description

Function Description

In general, the memory of the Safety PLC is write protected, i.e. write requests are not executed.

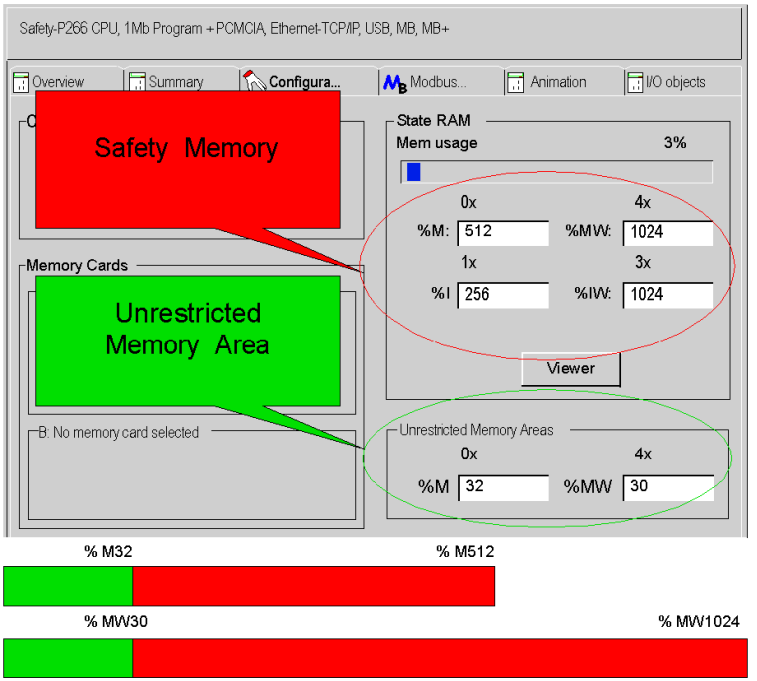
To allow exceptions from this rule (which are fully in the responsibility of the customer), e.g. to change set points from an HMI, the UMA (unrestricted memory area) for %M and %MW values has been introduced.

Definition of UMA

The UMA is characterized as follows:

- The UMA is a part of the Safety PLC's memory for %M and %MW values that is not write protected.
- Its addresses can be written from other PLCs or HMIs.
- Its configuration is performed in the **CPU property dialog** of Unity Pro XLS.

Configuring the memory of the Safety PLC (safety memory and UMA):



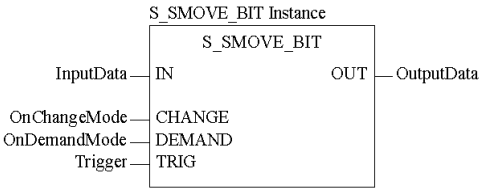
NOTE: `%M` and `%MW` in the UMA are located at the beginning of the memory range. You should thus make sure to reserve UMA space of the memory at the very beginning of the project.

Purpose of `S_SMOVE_BIT`

You cannot use variables of type `BOOL` from the UMA range of `%MW` use the `S_SMOVE_BIT` function block in the safety logic.

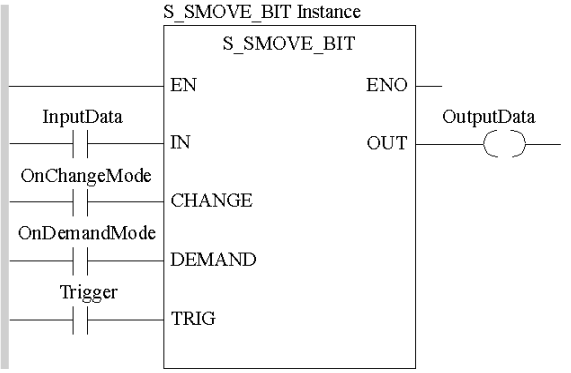
Representation in FDB

Representation



Representation in LD

Representation



Parameter Description

Description of the input parameters

Parameter	Data Type	Meaning
IN	BOOL	A variable or direct address from the unrestricted bit range.
CHANGE	BOOL	If the input data has changed, the input data is transferred to the output. Otherwise the output value remains unchanged. A literal can be connected or if a variable is used it must be mapped to a bit in the safety memory range.
DEMAND	BOOL	If a rising edge at the TRIG input is detected, the input data is transferred to the output. Otherwise the output value remains unchanged. A literal can be connected or if a variable is used it must be mapped to a bit in the safety memory range.
TRIG	BOOL	Only used in Demand Mode: A rising edge at this input triggers the transfer of the input value to OUT. A literal can be connected or if a variable is used it must be mapped to a bit in the safety memory range.

Description of output parameters

Parameter	Data Type	Meaning
OUT	BOOL	This is the controlled bit to be used in safety logic. It is mapped to a bit in the safety memory range.

Block Behavior

The S_SMOVE_BIT function block may work according to the following 3 modes:

- transparent mode
- demand mode
- change mode

These modes will be described in the following paragraphs.

Transparent Mode

If...	Then...
CHANGE mode bit = 0 and DEMAND mode bit = 0	the value of the input bit will be written to the output bit in the safety memory.

NOTE: If no signals are connected to DEMAND, CHANGE and TRIG the function block is in transparent mode because non-connected inputs are treated as connected with 0.

Demand Mode

If...	Then...
there is a rising edge detected at the TRIG input	the value of the input bit will be written to the output bit in the safety memory.

Change Mode

In this mode only CHANGE is set to 1 (DEMAND is 0 and TRIG has no influence).

If...	Then...
the input value IN is changing	the value of the input bit will be written to the output bit in the safety memory.

Notable Differences between Change and Transparent Mode

In change mode the function block only changes the value of the variable connected to the OUT if the input IN is changed. In transparent mode the value of the input IN will always be written to the output OUT. When doing modifications in online mode consider the following:

If...	Then...
you replace a variable connected to the output OUT by another variable	the newly connected variable will not change its current value until the input IN changes.
you replace a variable connected to the output OUT by another variable	the value of a link which is also connected to the output will become equal to the current value of the newly connected variable.
you replace a variable connected to the output OUT by a link	the value of the link will be 0 until the input IN changes.

Note for Change and Demand Mode

NOTE: Both DEMAND and CHANGE bits are set to 1. In this case the input will be written to the output when the input changes or when there is a rising edge on the trigger input.

State Table

Pin states and the respective mode

CHANGE Pin	DEMAND Pin	TRIG Pin	IN Pin	OUT Pin	Mode
0	0	-	-	= IN	transparent
0	1	0->1	-	= IN	demand
0	1	Not 0->1	-	old IN value	demand
1	0	-	has changed	= IN	change
1	0	-	no change	OUT unchanged	change
1	1	0->1	no change	= IN	demand + change
1	1	Not 0->1	no change	OUT unchanged	demand + change
1	1	0->1	has changed	= IN	demand + change
1	1	Not 0->1	has changed	= IN	demand + change

Hot Standby

IV

S_HSBY_SWAP: Hot Standby Swapping Function

7

Description

Function Description

The S_HSBY_SWAP function block is used to initiate the swapping between primary and standby CPU.

EN and ENO can be configured as additional parameters.

This function block is convenient to be used to activate a swap by program logic. This swap between primary and standby CPU can be performed in the Hot Standby Safety mode only.

Please refer to the Modicon Quantum Hot Standby with Unity - User Manual for information regarding safety hot standby CPUs.

This means that when the HSBY is running, the standby PLC becomes the primary PLC, and the old primary PLC becomes the standby PLC activated by the program logic.

NOTE: It is not mandatory to use this function for performing a hot standby swap in safety mode. Indeed, you may rely on register %SW60 and follow the procedure described in the Modicon Quantum Hot Standby with Unity - User Manual .

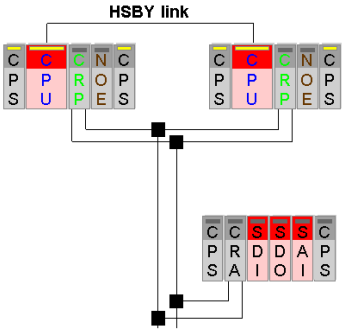
Advantages of the Swapping Function

The advantages of swapping could be the following:

- The health of the standby PLC is monitored. It is checked that the standby PLC can take over.
- The switch over could be tested at regular intervals.

Example of a Hot Standby Safety Application

The illustration below shows an example of a hot standby safety application:



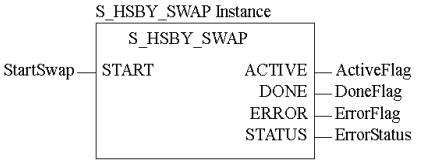
Steps for Changing Status

A Hot Standby swapping function is processed as follows:

Step	Action
1	**Status:** PLC-A is the primary controller, PLC-B is the standby controller. PLC-A sets itself to offline. **Result:** PLC-B becomes the primary controller.
2	**Status:** PLC-A is offline, PLC-B is the primary controller. PLC-B sets PLC-A to run-mode **Result:** PLC-A is the standby controller.
3	**Status:** PLC-A is the standby controller, PLC-B is the primary controller. EFB outputs will be set. **Result:** Hot Standby swapping function is completed.

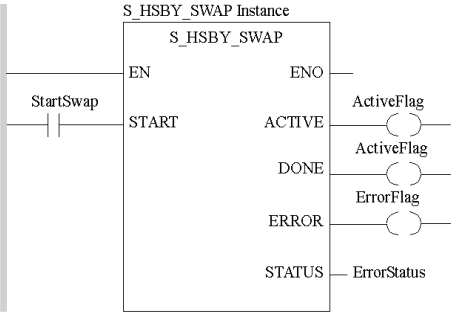
Representation in FBD

Representation



Representation in LD

Representation



Parameter Description

Description of the input parameter

Parameter	Data Type	Meaning
START	BOOL	START = 1 starts the S_HSBY_SWAP operation. The value of 1 must be applied until the operation has finished or until an error has occurred.

Description of output parameters

Parameter	Data Type	Meaning
ACTIVE	BOOL	ACTIVE = 1 indicates that an S_HSBY_SWAP operation is in progress.
DONE	BOOL	DONE = 1 indicates that the S_HSBY_SWAP operation has been completed successfully.
ERROR	BOOL	ERROR = 1 indicates that an error has occurred, or that the current S_HSBY_SWAP operation has been aborted.
STATUS	INT	An error code (STATUS) is generated by the S_HSBY_SWAP block. A complete list is shown in the error code table.

Error Status Table

The following table explains the error codes:

Error Codes	Fault Description
0	OK
1	The function S_HSBY_SWAP has been aborted.
2	Hot Standby not activated (%SW61.15=0).

Error Codes	Fault Description
3	Standby does not exist.
4	Primary controller is not in Safety mode.
5	The swap was unsuccessful.

NOTE: The system words %SW60 and %SW61 reflect the status of the primary and the secondary PLC.

Switchover Using Command Register System Bit `%SW60.1` or `%SW60.2`

Another way of forcing a switchover is setting the bits in the Command Register. To achieve this, do the following:

Step	Action
1	Open file 1.
2	Connect to the primary,
3	Ensure the controller order of the primary is A or B.
4	Access • Command Register system bit `%SW60.1` If the connected controller order is A. • Command Register system bit `%SW60.2` If the connected controller order is B.
5	Set bit to `0`. **Note**: Ensure that the standby switched to primary.
6	Open file 2.
7	Connect to the new primary controller.
8	Access the Command Register system bit used in step 4.
9	Set bit to `1`. **Note**: Ensure that the standby controller is now online.
10	Ensure both primary and standby controllers are in Run Primary and in Run Standby mode.

Appendices

System Objects

A

Introduction

This chapter describes the system bits and words of the Quantum Safety PLC.

Note: The symbols, associated with each bit object or system word, mentioned in the descriptive tables of these objects, are not implemented as standard in the software, but can be entered using the data editor.

They are proposed in order to ensure the homogeneity of their names in the different applications.

What's in this Chapter?

This chapter contains the following sections:

A.1 System Bits

Introduction

This section describes the system bits of the Quantum Safety PLC.

For your convenience, all system bits of standard Quantum PLCs are listed but only explained further if used in the Quantum Safety PLC.

What's in this Section?

This section contains the following topics:

System Bit Introduction

General

The Quantum PLCs use %Si system bits which indicate the state of the PLC, or they can be used to control how it operates.

These bits can be tested in the user program to detect any functional development requiring a set processing procedure.

Some of these bits must be reset to their initial or normal state by the program. However, the system bits that are reset to their initial or normal state by the system must not be reset by the program or by the terminal.

Description of the System Bits %S0 to %S13

Detailed Description

NOTE: Not all of the system bits can be used in the Quantum Safety PLC. The unusable system bits are marked in the **Quant. Safety** column with no.

The following table gives a description of the system bits %S0 to %S13:

Bit **Symbol**	**Function**	**Description**	**Initial State**	**Write Access**	**Quant. Safety**
%S0 COLDSTART	cold start	Normally at 0, this bit is set to 1 by: • power restoral with loss of data (battery fault), • the user program, • the terminal, • a change of cartridge, • program loading. This bit is set to 1 during the first complete restored cycle of the PLC either in RUN or in STOP mode. It is reset to 0 by the system before the following cycle.	1 (1 cycle)	no	yes
%S1 WARMSTART	warm restart	see chapter "System Bits" in the *Unity Pro Languages and Program Structure Reference Manual*	0	no	no
%S4 TB10MS	time base 10 ms	see chapter "System Bits" in the *Unity Pro Languages and Program Structure Reference Manual*	-	no	no
%S5 TB100MS	time base 100 ms	see chapter "System Bits" in the *Unity Pro Languages and Program Structure Reference Manual*	-	no	no
%S6 TB1SEC	time base 1 s	see chapter "System Bits" in the *Unity Pro Languages and Program Structure Reference Manual*	-	no	no
%S7 TB1MIN	time base 1 min	see chapter "System Bits" in the *Unity Pro Languages and Program Structure Reference Manual*	-	no	no
%S10 IOERR	input/output fault	Normally at 1, this is set to 0 when an I/O fault on an in-rack module or device on Fipio is detected (e.g. non-compliant configuration, exchange fault, hardware fault, etc.). The %S10 bit is reset to 1 by the system as soon as the fault disappears.	1	no	yes
%S11 WDG	watchdog overflow	Normally at 0, this is set to 1 by the system as soon as the task execution time becomes greater than the maximum execution time (i.e. the watchdog) declared in the task properties.	0	no	yes

Bit Symbol	Function	Description	Initial State	Write Access	Quant. Safety
%S12 PLCRUNNING	PLC in RUN	This bit is set to 1 by the system when the PLC is in RUN. It is set to 0 by the system as soon as the PLC is no longer in RUN (STOP, INIT, etc.).	0	no	yes
%S13 1RSTSCANRUN	first cycle after switching to RUN	Normally set to 0, this is set to 1 by the system during the first cycle of the master task after the PLC is set to RUN.	-	no	yes

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

On Quantum Safety PLCs, communication errors from modules NOE, CRA and CRP are not reported on bit %S10.

It is entirely your responsibility to ensure that these system bits are used correctly.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Description of the System Bits %S15 to %S21

Detailed Description

NOTE: Not all of the system bits can be used in the Quantum Safety PLC. The unusable system bits are marked in the **Quant. Safety** column with no.

The following table gives a description of the system bits %S15 to %S21:

Bit Symbol	Function	Description	Initial State	Write Access	Quant. Safety
%S15 STRINGERROR	character string fault	see chapter "System Bits" in the *Unity Pro Languages and Program Structure Reference Manual*	0	yes	no
%S16 IOERRTSK	task input/output fault	Normally set to 1, this bit is set to 0 by the system when a fault occurs on an in-rack I/O module or a Fipio device configured in the task. This bit must be reset to 1 by the user.	1	yes	yes
%S17 CARRY	rotate or shift output	normally at 0 During a rotate or shift operation, this bit takes the state of the outgoing bit.	0	no	yes
%S18 OVERFLOW	overflow or arithmetic error	Normally set to 0, this bit is set to 1 in the event of a capacity overflow if there is • a result greater than + 32 767 or less than - 32 768, in single length, • result greater than + 65 535, in unsigned integer, • a result greater than + 2 147 483 647 or less than - 2 147 483 648, in double length, • result greater than +4 294 967 296, in double length or unsigned integer, • real values outside limits, • division by 0, • the root of a negative number, • forcing to a non-existent step on a drum, • stacking up of an already full register, emptying of an already empty register. It must be tested by the user program after each operation where there is a risk of overflow, and then reset to 0 by the user if there is indeed an overflow. When the %S18 bit switches to 1, the application stops in error state if the %S78 bit has been set to 1.	0	yes	yes
%S19 OVERRUN	task period overrun (periodical scanning)	Normally set to 0, this bit is set to 1 by the system in the event of a time period overrun (i.e. task execution time is greater than the period defined by the user in the configuration or programmed into the %SW word associated with the task). The user must reset this bit to 0. Each task manages its own %S19 bit.	0	yes	yes

Bit Symbol	Function	Description	Initial State	Write Access	Quant. Safety
%S20 INDEXOVF	Index overflow	Normally set to 0, this bit is set to 1 when the address of the indexed object becomes less than 0 or exceeds the number of objects declared in the configuration. In this case, it is as if the index were equal to 0. It must be tested by the user program after each operation where there is a risk of overflow, and then reset to 0 if there is indeed an overflow. When the %S20 bit switches to 1, the application stops in error state if the %S78 bit has been set to 1.	0	yes	no
%S21 1RSTTASKRUN	first task cycle	Tested in a task (Mast, Fast, Aux0, Aux1, Aux2 Aux3), the bit %S21 indicates the first cycle of this task. %S21 is set to 1 at the start of the cycle and reset to zero at the end of the cycle. **Notes**: The bit %S21 does not have the same meaning in PL7 as in Unity Pro.	0	no	yes

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

On Quantum Safety PLCs, communication errors from modules NOE, CRA and CRP are not reported on bit %S16.

It is entirely your responsibility to ensure that these system bits are used correctly.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Description of the System Bits %S30 to %S51

Detailed Description

NOTE: Not all of the system bits can be used in the Quantum Safety PLC. The unusable system bits are marked in the **Quant. Safety** column with no.

The following table gives a description of the system bits %S30 to %S51:

Bit Symbol	Function	Description	Initial State	Write Access	Quant. Safety
%S30 MASTACT	activation/deactivation of the master task	see chapter "System Bits" in the *Unity Pro Languages and Program Structure Reference Manual*	1	yes	no
%S31 FASTACT	activation/deactivation of the fast task	see chapter "System Bits" in the *Unity Pro Languages and Program Structure Reference Manual*	0	yes	no
%S32 **%S33** **%S34** **%S35**	activation/deactivation of the auxiliary tasks 0-3	see chapter "System Bits" in the *Unity Pro Languages and Program Structure Reference Manual*	0	yes	no
%S38 ACTIVEVT	enabling/inhibition of events	see chapter "System Bits" in the *Unity Pro Languages and Program Structure Reference Manual*	1	yes	no
%S39 EVTOVR	saturation in event processing	see chapter "System Bits" in the *Unity Pro Languages and Program Structure Reference Manual*	0	yes	no
%S50 RTCWRITE	updating of time and date via words %SW50 to %SW53	Normally set to 0, this bit is set to 1 by the program or the terminal: • set to 0: update of system words %SW50 to %SW53 by the date and time supplied by the PLC real-time clock, • set to 1: system words %SW50 to %SW53 are no longer updated, therefore making it possible to modify them. • The switch from 1 to 0 updates the real-time clock with the values entered in words %SW50 to %SW53.	0	yes	yes
%S51 RTCERR	time loss in real-time clock	This system-managed bit set to 1 indicates that the real-time clock is missing or that its system words (%SW50 to %SW53) are meaningless. If set to 1, the clock must be reset to the correct time.	-	no	yes

Description of the System Bits %S59 to %S122

Detailed Description

NOTE: Not all of the system bits can be used in the Quantum Safety PLC. The unusable system bits are marked in the **Quant. Safety** column with no.

The following table gives a description of the system bits %S59 to %S122:

Bit Symbol	Function	Description	Initial State	Write Access	Quant. Safety
%S59 RTCTUNING	incremental update of the time and date via word %SW59	Normally set to 0, this bit can be set to 1 or 0 by the program or the terminal: ● set to 0: the system does not manage the system word %SW59, ● set to 1: the system manages edges on word %SW59 to adjust the date and current time (by increment).	0	yes	yes
%S67 PCMCIABAT0	state of the application memory card battery	This bit is used to monitor the status of the main battery when the memory card is in the upper PCMCIA slot (all the Atriums, Premiums, and on the Quantums (140 CPU 671 60, 140 CPU 651 60 and 140 CPU 651 50)): ● set to 1: main voltage battery is low (application is preserved but you must replace the battery following the so-called predictive maintenance procedure), ● set to 0: main battery voltage is sufficient (application always preserved). Bit %S67 is managed: ● on the PV06 small and medium capacity RAM memory cards (product version written on the card label), i.e. offering memory size under Unity =#768K: TSX MRP P 128K, TSX MRP P 224K TSX MCP C 224K, MCP C 512K, TSX MRP P 384K, TSX MRP C 448K, TSX MRP C 768K, ● under Unity whose version is ≥ 2.02.	-	no	yes
%S68 PLCBAT	state of the processor battery	This bit is used to check the operating state of the backup battery for saving data and the program in RAM: ● set to 0: battery present and operational, ● set to 1: battery missing or non-operational.	-	no	yes
%S75 PCMCIABAT1	state of the data storage memory card battery	This bit is used to monitor the status of the main battery when the memory card is in the lower PCMCIA slot, see chapter "System Bits" in the *Unity Pro Languages and Program Structure Reference Manual*. **Note:** Data stored on a memory card in slot B is not processed in safety projects.	-	no	no

Bit Symbol	Function	Description	Initial State	Write Access	Quant. Safety
%S76 DIAGBUFFCONF	configured diagnostics buffer	This bit is set to 1 by the system when the diagnostics option has been configured. Then, a diagnostics buffer for storage of errors found by diagnostics DFBs is reserved. This bit is read-only.	0	no	yes
%S77 DIAGBUFFFULL	full diagnostics buffer	This bit is set to 1 by the system when the buffer that receives errors from the diagnostics function blocks is full. This bit is read-only.	0	no	yes
%S78 HALTIFERROR	stop in the event of error	Normally at 0, this bit can be set to 1 by the user, to program a PLC stop on application fault: %S15, %S18, %20.	0	yes	yes
%S80 RSTMSGCNT	reset message counters	Normally set to 0, this bit can be set to 1 by the user to reset the message counters %SW80 to %SW86.	0	yes	yes
%S94 SAVECURRVAL	saving adjustment values	see chapter "System Bits" in the *Unity Pro Languages and Program Structure Reference Manual*	0	yes	no
%S118 REMIOERR	general Fipio I/O fault	Normally set to 1, this bit is set to 0 by the system when a fault occurs on a device connected to the RIO and Fipio remote input/output bus. This bit is reset to 1 by the system when the fault disappears.	-	no	yes
%S119 LOCIOERR	general inrack I/O fault	Normally set to 1, this bit is set to 0 by the system when a fault occurs on an I/O module placed in 1 of the racks. This bit is reset to 1 by the system when the fault disappears.	-	no	yes
%S120 **%S121** **%S122**	DIO bus faults	see chapter "System Bits" in the *Unity Pro Languages and Program Structure Reference Manual*	-	no	no

A.2 System Words

Introduction

This section describes the system words of the Quantum Safety PLC.

For your convenience, all system words of standard Quantum PLCs are listed but only explained further if used in the Quantum Safety PLC.

What's in this Section?

This section contains the following topics:

Description of the System Words %SW0 to %SW21

Detailed Description

NOTE: Not all of the system words can be used in the Quantum Safety PLC. The unusable system words are marked in the **Quant. Safety** column with no.

The following table gives a description of the system words %SW0 to %SW21:

Word **Symbol**	**Function**	**Description**	**Initial State**	**Write Access**	**Quant. Safety**
%SW0 MASTPERIOD	master task scanning period	see chapter "System Objects" in the *Unity Pro Languages and Program Structure Reference Manual*	0	yes	no
%SW1 FASTPERIOD	fast task scanning period	see chapter "System Objects" in the *Unity Pro Languages and Program Structure Reference Manual*	0	yes	no
%SW2, **%SW3**, **%SW4**, **%SW5**	auxiliary task scanning period	see chapter "System Objects" in the *Unity Pro Languages and Program Structure Reference Manual*	0	yes	no
%SW8 TSKINHIBIN	acquisition of task input monitoring	see chapter "System Objects" in the *Unity Pro Languages and Program Structure Reference Manual*	0	yes	no
%SW9 TSKINHIBOUT	monitoring of task output update	see chapter "System Objects" in the *Unity Pro Languages and Program Structure Reference Manual*	0	yes	no
%SW10 TSKINIT	first cycle after cold start	see chapter "System Objects" in the *Unity Pro Languages and Program Structure Reference Manual*	0	no	no
%SW11 WDGVALUE	watchdog duration	Reads the duration of the watchdog. The duration is expressed in milliseconds (10...1500 ms). This word cannot be modified.	-	no	yes
%SW12 APMODE	mode of application processor	This word indicates the operating mode of the application processor. Possible values are • 16#A501: application processor is in Maintenance Mode. • 16#5AFE: application processor is in Safety Mode. Any other value is interpreted as an error. This system word is not available for the standard Quantum CPU.	16#A501	no	yes

Word Symbol	Function	Description	Initial State	Write Access	Quant. Safety
%SW13 INTELMODE	mode of Intel processor	This word indicates the operating mode of the Intel Pentium processor. Possible values are • 16#501A: application processor is in Maintenance Mode. • 16#5AFE: application processor is in Safety Mode. Any other value is interpreted as an error. This system word is not available for the standard Quantum CPU.	16#501A	no	yes
%SW14 OSCOMMVERS	commercial version of PLC processor	This word contains the commercial version of the PLC processor. **Example:** 16#0135 version: 01; issue number: 35	-	no	yes
%SW15 OSCOMMPATCH	PLC processor patch version	This word contains the commercial version of the PLC processor patch. It is coded onto the least significant byte of the word. coding: 0 = no patch, 1 = A, 2 = B... **Example:** 16#0003 corresponds to patch C.	-	no	yes
%SW16 OSINTVERS	firmware version number	This word contains the Firmware version number in hexadecimal of the PLC processor firmware. **Example:** 16#0017 version: 2.1; VN: 17	-	no	yes
%SW17 FLOATSTAT	error status on floating operation	see chapter "System Objects" in the *Unity Pro Languages and Program Structure Reference Manual*	0	yes	no
%SW18 **%SW19** 100MSCOUNTER	absolute time counter	%SW18 is the low and %SW19 the high word for calculating durations. Both are incremented every 1/10th of a second by the system (even when the PLC is in STOP, they are no longer incremented if it is powered down). They can be read and written by the user program or by the terminal.	0	yes	yes
%SW20 **%SW21** MSCOUNTER	absolute time counter	The low word %SW20 and the high word %SW21 are incremented every 1/1000th of a second by the system (even when the PLC is in STOP, they are no longer incremented if it is powered down). They can be read by the user program or by the terminal. %SW20 and %SW21 are reset on a cold start, but not on a warm start.	0	no	yes

Description of the System Words %SW30 to %SW59

Detailed Description

NOTE: Not all of the system words can be used in the Quantum Safety PLC. The unusable system words are marked in the **Quant. Safety** column with no.

The following table gives a description of the system words %SW30 to %SW59:

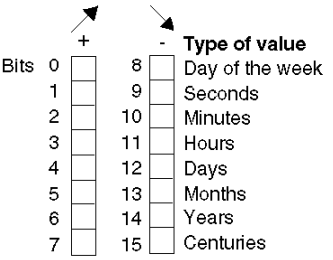
Word **Symbol**	**Function**	**Description**	**Initial State**	**Write Access**	**Quant. Safety**
%SW30 MASTCURRTIME	master task execution time	This word indicates the execution time of the last master task cycle (in ms).	-	no	yes
%SW31 MASTMAXTIME	maximum master task execution time	This word indicates the longest master task execution time since the last cold start (in ms).	-	no	yes
%SW32 MASTMINTIME	minimum master task execution time	This word indicates the shortest master task execution time since the last cold start (in ms).	-	no	yes
%SW33 **%SW34** **%SW35**	fast task execution times	see chapter "System Objects" in the *Unity Pro Languages and Program Structure Reference Manual*	-	no	no
%SW36 to %SW47	auxiliary tasks execution times	see chapter "System Objects" in the *Unity Pro Languages and Program Structure Reference Manual*	-	no	no
%SW48 IOEVTNB	number of events	see chapter "System Objects" in the *Unity Pro Languages and Program Structure Reference Manual*	0	yes	no

Word Symbol	Function	Description	Initial State	Write Access	Quant. Safety

%SW49
DAYOFWEEK
 	
%SW50
SEC
 	
%SW51
HOURMIN
 	
%SW52
MONTHDAY
 	
%SW53
YEAR
	real-time clock function	System words containing date and current time (in BCD):
		• %SW49: day of the week:
		  • 1 = Monday,
		  • 2 = Tuesday,
		  • 3 = Wednesday,
		  • 4 = Thursday,
		  • 5 = Friday,
		  • 6 = Saturday,
		  • 7 = Sunday,
		• %SW50: Seconds (16#SS00),
		• %SW51: Hours and Minutes (16#HHMM),
		• %SW52: Month and Day (16#MMDD),
		• %SW53: Year (16#YYYY).
		
		These words are managed by the system when the bit %S50 is set to 0.
		These words can be written by the user program or by the terminal when the bit %S50 is set to 1.
			-	yes	yes

Word Symbol	Function	Description	Initial State	Write Access	Quant. Safety

%SW54
STOPSEC
 	
%SW55
STOPHM
 	
%SW56
STOPMD
 	
%SW57
STOPYEAR
 	
%SW58
STOPDAY
	real-time clock function on last stop	System words containing date and time of the last power outage or PLC stop (in Binary Coded Decimal):
		• %SW54: Seconds (00SS),
		• %SW55: Hours and Minutes (HHMM),
		• %SW56: Month and Day (MMDD),
		• %SW57: Year (YYYY),
		• %SW58: the most significant byte contains the day of the week (1 for Monday through to 7 for Sunday), and the least significant byte contains the code for the last stop:
		  • 1 = change from RUN to STOP by the terminal or the dedicated input,
		  • 2 = stop by watchdog (PLC task or SFC overrun),
		  • 4 = power outage or memory card lock operation,
		  • 5 = stop on hardware fault,
		  • 6 = stop on software fault. Details on the type of software fault are stored in %SW125.
			-	no	yes

%SW59 ADJDATETIME	adjustment of current date	Contains 2 8-bit series to adjust the current date. The action is performed on the rising edge of the bit. This word is enabled by bit %S59=1. In the following illustration, bits in the left column increment the value, and bits in the right column decrement the value:  	0	yes	yes

Description of the System Words %SW60 to %SW127

Detailed Description

NOTE: Not all of the system words can be used in the Quantum Safety PLC. The unusable system words are marked **no** in the **Quant. Safety** column.

The following table gives a description of the system words %SW60 to %SW127:

Word **Symbol**	**Function**	**Description**	**Initial State**	**Write Access**	**Quant. Safety**

%SW60
HSB_CMD
	Quantum Hot Standby command register	Meaning of the different bits of the word %SW60:
		• %SW60.0=1 invalidates the commands entered in the display (keypad).
		• %SW60.1
		  • =0 sets PLC A to OFFLINE mode.
		  • =1 sets PLC A to RUN mode.
		• %SW60.2
		  • =0 sets PLC B to OFFLINE mode.
		  • =1 sets PLC B to RUN mode.
		• %SW60.3=0 forces Standby PLC to OFFLINE mode if the applications are different.
		• %SW60.4
		  • =0 authorizes an update of the firmware only after the application has stopped.
		  • =1 authorizes an update of the firmware without the application stopping.
		• %SW60.5=1 application transfer request from the Standby to the primary.
		• %SW60.8
		  • =0 address switch on Modbus port 1 during a primary swap.
		  • =1 no address switch on Modbus port 1 during a primary swap.
			0	no	yes

Word Symbol	Function	Description	Initial State	Write Access	Quant. Safety

%SW61
HSB_STS
	Quantum status register	Meaning of the different bits of the word %SW61:
		• %SW61.0 und %SW61.1 PLC operating mode bits:
		  • %SW61.1=0, %SW61.0=1: OFFLINE mode.
		  • %SW61.1=1, %SW61.0=0: primary mode.
		  • %SW61.1=1, %SW61.0=1: secondary mode (Standby).
		• %SW61.2 and %SW61.3 operating mode bits from the other PLC
		  • %SW61.3=0, %SW61.2=1: OFFLINE mode.
		  • %SW61.3=1, %SW61.2=0: primary mode.
		  • %SW61.3=1, %SW61.2=1: secondary mode (Standby).
		  • %SW61.3=0, %SW61.2=0: the remote PLC is not accessible (switched off, no communication).
		• %SW61.4=0 the applications are identical on both PLCs.
		• %SW61.5
		  • =0 the PLC is used as unit A.
		  • =1 the PLC is used as unit B.
		• %SW61.7
		  • =0 same PLC OS version.
		  • =1 different PLC OS version.
		• %SW61.8
		  • =0 same Copro OS version.
		  • =1 different Copro OS version.
		• %SW61.12
		  • =0 information given by bit 13 is not relevant.
		  • =1 information given by bit 13 is valid.
		• %SW61.13
		  • =0 NOE address set to IP.
		  • =1 NOE address set to IP + 1.
		• %SW61.15
		  • =0 Hot Standby not activated.
		  • =1 Hot Standby activated.
			0	no	yes

Word Symbol	Function	Description	Initial State	Write Access	Quant. Safety
%SW62 HSBY_REVERSE0 **%SW63** HSBY_REVERSE1	transfer word	These 2 words may be written by the user in the first section of the master task. They are then transferred automatically from the Standby processor to update the primary PLC. They may be read on the primary PLC and be used as primary application parameters.	0	yes	no
%SW70 WEEKOFYEAR	real-time clock function	System word containing the number of the week in the year: 1 to 52.	–		yes
%SW71 KEY_SWITCH	position of the switches on the Quantum front panel	This word provides the image of the positions of the switches on the front panel of the Quantum processor. This word is updated automatically by the system. • %SW71.0 = 1 switch in the "Memory protected" position, • %SW71.1 = 1 switch in the "STOP" position, • %SW71.2 = 1 switch in the "START" position, • %SW71.8 = 1 switch in the "MEM" position, • %SW71.9 = 1 switch in the "ASCII" position, • %SW71.10 = 1 switch in the "RTU" position, • %SW71.3 to 7 and 11 to 15 are not used.	0	no	yes
%SW75 TIMEREVTNB	timer-type event counter	see chapter "System Objects" in the *Unity Pro Program Languages and Structure Reference Manual*	0		no
%SW76 DLASTREG	diagnostics function: recording	Result of the last registration: • = 0 if the recording was successful, • = 1 if the diagnostics buffer has not been configured, • = 2 if the diagnostics buffer is full.	0		yes
%SW77 DLASTDEREG	diagnostics function: non-recording	Result of the last deregistration: • = 0 if the non-recording was successful, • = 1 if the diagnostics buffer has not been configured, • = 21 if the error identifier is invalid, • = 22 if the error has not been recorded.	0		yes
%SW78 DNBERRBUF	diagnostics function: number of errors	Number of errors currently in the diagnostics buffer.	0		yes

Word Symbol	Function	Description	Initial State	Write Access	Quant. Safety
%SW80 MSGCNT0 **%SW81** MSCNT1	message management	These words are updated by the system, and can also be reset using %S80. • %SW80: Number of Modbus messages sent by the system as client on all communication ports except USB and Ethernet copro. **NOTE:** Modbus messages sent by the system as Master are not counted in this word. • %SW81: Number of Modbus messages received by the system as client on all communication ports except USB and Ethernet copro. **NOTE:** Modbus messages received as response to the requests sent by the system, as Master, are not counted in this word.	0	yes	yes
%SW87 MSTSERVCNT	communication flow management	Number of requests processed by synchronous server per master (MAST) task cycle.	0		yes
%SW90 MAXREQNB	maximum number of requests processed per master task cycle	This word is used to set a maximum number of requests which can be processed by the PLC per master task cycle. When the CPU is the server: This number of requests must be between 2 (minimum) and N+4 (maximum). N: number differs depending on the model. When the CPU is the client: N: number differs depending on the model. The value 0 does not work. If a value that is outside of the range is entered, it is the value N that is taken into account. See also chapter "System Objects" in the *Unity Pro Program Languages and Structure Reference Manual*.	0	yes	yes
%SW108 FORCEDIOIM	number of forced I/O module bits	This system word counts the number of forced I/O module bits. This word is incremented for every forcing, and decremented for every unforcing.	0	no	yes
%SW110	number of unrestricted memory area for %M	This system word gives information on the size of the unrestricted memory area for %M. This system word is not available for the standard Quantum CPU.	–	no	yes
%SW111	number of unrestricted memory area for %MW	This system word gives information on the size of the unrestricted memory area for %MW. This system word is not available for the standard Quantum CPU.	–	no	yes

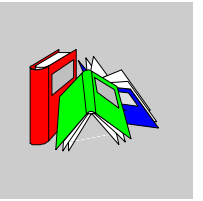
Word Symbol	Function	Description	Initial State	Write Access	Quant. Safety
%SW124 CPUERR	type of system fault	This system word is updated if the PLC is set to error state. The possible values are as follows: ● 0x0065: execution of HALT instruction impossible ● 0x0080: system watchdog If the PLC is set to Safety error state, the content of %SW125 is updated and can be read after the next restart of the PLC (see below).	–	no	yes

Word Symbol	Function	Description	Initial State	Write Access	Quant. Safety
%SW125 BLKERRTYPE	last fault detected	The code of the last fault detected is given in this word. The following error codes cause the PLC to stop if %S78 is set to 1. %S15, %S18 and %S20 are activated independently of %S78: ● 16#0002: PCMCIA signature not verified ● 16#2258: execution of HALT instruction ● 16#2302: call to a not supported system function in a user function block ● 16#9690: error of application CRC detected in background ● 16#DE87: calculation error on floating-point numbers (%S18, these errors are listed in the word %SW17) ● 16#DEB0: watchdog overflow (%S11) ● 16#DEF1: character string transfer error (%S15) ● 16#DEF2: arithmetic or division by 0 error (%S18) ● 16#DEF3: index overflow (%S20) **Note:** The codes 16#8xxx and 16#7xxx do not stop the application and indicate an error on function blocks. In case of a SIL3 related error, the PLC stops. After power off and restart of the PLC, %SW 125 contains the code of the cause of the error: ● 0x5AF1: sequence check error (unpredictable execution in CPU) ● 0x5AF2: error in memory (corrupt address) ● 0x5AF3: comparison error (execution results of Intel and application processor differ) ● 0x5AF4: real-time clock error ● 0x5AF5: error initializing double code execution ● 0x5AF6: watchdog activation error ● 0x5AF7: error during memory check (takes more than 8 hours) ● 0x5AF8: error in memory check (corrupt RAM) **Note:** %SW125 is only reset after init or complete download or restart (it always contains the last fault detected).	–	no	yes

Word Symbol	Function	Description	Initial State	Write Access	Quant. Safety
%SW126 ERRADDR0 **%SW127** ERRADDR1	blocking fault instruction address	Address of the instruction that generated the application blocking fault. For 16 bit processors: • %SW126 contains the offset for this address, • %SW127 contains the segment number for this address. For 32 bit processors: • %SW126 contains the least significant word for this address, • %SW127 contains the most significant word for this address. The content of %SW126 and %SW127 is for Schneider Electric use only.	0	no	yes

For the description of the system words %SW128 to %SW339 and %SW535 to %SW640, see the chapter "Quantum Specific System Words" in the *Unity Pro Program Languages and Structure Reference Manual*. The system words %SW340 to %SW534 are not used in Quantum Safety PLCs.

Glossary



0-9

%

Prefix that identifies internal memory addresses in the controller that are used to store the value of program variables, constants, I/O, etc.

%I

According to the IEC standard, %I indicates a discrete input-type language object.

%IW

According to the IEC standard, %IW indicates an analog input -type language object.

%KW

According to the IEC standard, %KW indicates a constant word-type language object.

%M

According to the IEC standard, %M indicates a memory bit-type language object.

%MW

According to the IEC standard, %MW indicates a memory word-type language object.

%Q

According to the IEC standard, %Q indicates a discrete output-type language object.

%QW

According to the IEC standard, %QW indicates an analog output-type language object.

A

ARRAY

An ARRAY is a table of elements of the same type.

The syntax is as follows: ARRAY [<terminals>] OF <Type>

Example:

ARRAY [1..2] OF BOOL is a one-dimensional table made up of 2 BOOL-type elements.

NOTE: Only one-dimensional ARRAYs are allowed for safety applications.

B

base 10 literals

A literal value in base 10 is used to represent a decimal integer value. This value can be preceded by the signs + and -. If the character _ is employed in this literal value, it is not significant.

Example:

-12, 0, 123_456, +986

base 16 literals

An literal value in base 16 is used to represent an integer in hexadecimal. The base is determined by the number 16 and the sign #. The signs + and - are not allowed. For greater clarity when reading, you can use the sign _ between bits.

Example:

16#F_F or 16#FF (in decimal 255)

16#E_0 or 16#E0 (in decimal 224)

base 2 literals

A literal value in base 2 is used to represent a binary integer. The base is determined by the number 2 and the sign #. The signs + and - are not allowed. For greater clarity when reading, you can use the sign _ between bits.

Example:

2#1111_1111 or 2#11111111 (in decimal 255)

2#1110_0000 or 2#11100000 (in decimal 224)

base 8 literals

A literal value in base 8 is used to represent an octal integer. The base is determined by the number 8 and the sign #. The signs + and - are not allowed. For greater clarity when reading, you can use the sign _ between bits.

Example:

8#3_77 or 8#377 (in decimal 255)

8#34_0 or 8#340 (in decimal 224)

BCD

binary coded decimal format

BCD is used to represent decimal numbers between 0 and 9 using a group of 4 bits (half-byte).

In this format, the 4 bits used to code the decimal numbers have a range of unused combinations.

Example of BCD coding:
- the number 2450
- is coded: 0010 0100 0101 0000

BOOL

BOOL is the abbreviation of boolean type. This is the elementary data item in computing. A BOOL type variable has a value of either: 0 (FALSE) or 1 (TRUE).

A BOOL type word extract bit, for example: %MW10.4.

BYTE

When 8 bits are put together, this is called a BYTE. A BYTE is either entered in binary, or in base 8.

The BYTE type is coded in an 8 bit format, which, in hexadecimal, ranges from 16#00 to 16#FF

D

DBCD

representation of a double BCD-format double integer

The binary coded decimal (BCD) format is used to represent decimal numbers between 0 and 9 using a group of 4 bits.

In this format, the 4 bits used to code the decimal numbers have a range of unused combinations.

Example of DBCD coding
- number `78993016`
- is coded: `0111 1000 1001 1001 0011 0000 0001 0110`

DINT

double integer format (coded on 32 bits).

The lower and upper limits are as follows: -(2 to the power of 31) to (2 to the power of 31) - 1.

Example:

`-2147483648`, `2147483647`, `16#FFFFFFFF`.

DWORD

double word

The `DWORD` type is coded in 32 bit format.

This table shows the lower/upper limits of the bases which can be used:

Base	Lower Limit	Upper Limit
Hexadecimal	16#0	16#FFFFFFFF
Octal	8#0	8#37777777777
Binary	2#0	2#11111111111111111111111111111111

Representation examples

Data Content	Representation in One of the Bases
00000000000010101101110011011110	16#ADCDE
00000000000000010000000000000000	8#200000
00000000000010101011110011011110	2#10101011110011011110

E

EBOOL

extended boolean type

A EBOOL type variable brings a value 0 (FALSE) or 1 (TRUE) but also rising or falling edges and forcing capabilities.

An EBOOL type variable takes up 1 byte of memory.

The byte split up into

- 1 bit for the value,
- 1 bit for the history bit (each time the state's object changes, the value is copied inside the history bit),
- 1 bit for the forcing bit (equals to 0, if the object isn't forced, equal to 1 if the bit is forced.

The default type value of each bit is 0 (FALSE).

EF

elementary function

This is a block which is used in a program, and which performs a predefined software function.

A function has no internal status information. Multiple invocations of the same function using the same input parameters always supply the same output values. Details of the graphic form of the function invocation can be found in the "[Functional block (instance)] ". In contrast to the invocation of the function blocks, function invocations only have a single unnamed output, whose name is the same as the function. In FBD each invocation is denoted by a unique [number] via the graphic block, this number is automatically generated and can not be altered.

You position and set up these functions in your program in order to carry out your application.

You can also develop other functions using the SDKC development kit.

EFB

elementary function block

This is a block which is used in a program, and which performs a predefined software function.

EFBs have internal statuses and parameters. Even where the inputs are identical, the output values may be different. For example, a counter has an output which indicates that the preselection value has been reached. This output is set to 1 when the current value is equal to the preselection value.

elementary function

see EF

EN

EN means **EN**able, this is an optional block input.

When EN is activated, an ENO output is automatically drafted.

If...	**then...**
If EN = 0,	• the block is not activated, • its internal program is not executed, • and ENO is set to 0.
If EN = 1,	• the internal program of the block is executed, • and ENO is set to 1 by the system. **Note:** If an error occurs, ENO is set to 0.
If EN is not connected,	it is automatically set to 1.

ENO

ENO means **E**rror **NO**tification, this is the output associated to the optional input EN.

If ENO is set to 0 (caused by EN=0 or in case of an execution error),
- the outputs of function blocks remain in the status they were in for the last correct executed scanning cycle, and
- the output(s) of functions and procedures are set to 0.

F

FBD

function block diagram

FBD is a graphic programming language that operates as a logic diagram. In addition to the simple logic blocks (AND, OR, etc.), each function or function block of the program is represented using this graphic form. For each block, the inputs are located to the left and the outputs to the right. The outputs of the blocks can be linked to the inputs of other blocks to form complex expressions.

FFB

collective term for EF (elementary function), EFB (elementary function block) and DFB (derived function block)

function

see EF

function block diagram

see FBD

G

GRAY

Gray or "reflected binary" code is used to code a numerical value being developed into a chain of binary configurations that can be differentiated by the change in status of one and only one bit.

This code can be used, for example, to avoid the following random event: in pure binary, the change of the value 0111 to 1000 can produce random numbers between 0 and 1000, as the bits do not change value altogether simultaneously.

Equivalence between decimal, BCD and Gray

Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
Gray	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101

I

IEC 61131-3

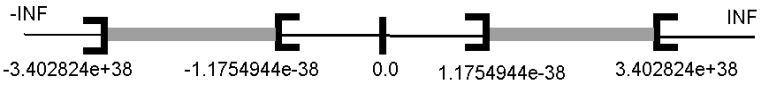
international standard: programmable logic controls

Part 3: Programming Languages

INF

Used to indicate that a number overruns the allowed limits.

For a number of Integers, the value ranges (shown in gray) are as follows:



When a calculation result is

- less than -3.402824e+38, the symbol `-INF` (for -infinite) is displayed,
- greater than +3.402824e+38, the symbol `INF` (for +infinite) is displayed.

INT

single integer format (coded on 16 bits)

The lower and upper limits are as follows: -(2 to the power of 15) to (2 to the power of 15) - 1.

Example

```
-32768, 32767, 2#1111110001001001, 16#9FA4.
```

integer literals

Integer literal are used to enter integer values in the decimal system. The values can have a preceding sign (+/-). Individual underlines (_) between numbers are not significant.

Example

```
-12, 0, 123_456, +986
```

IODDT

input/output derived data type

The term IODDT designates a structured data type representing a module or a channel of a PLC module. Each application expert module possesses its own IODDTs.

K

keyword

A keyword is a unique combination of characters used as a syntactical programming language element. (See annex B definition of the IEC standard 61131-3. All the keywords used in Unity Pro and of this standard are listed in annex C of the IEC standard 61131-3. These keywords cannot be used as identifiers in your program (names of variables, sections, DFB types, etc.))

L

LD

ladder diagram

LD is a programming language, representing the instructions to be carried out in the form of graphic diagrams very close to a schematic electrical diagram (contacts, coils, etc.).

located variables

A located variable is a variable for which it is possible to know its position in the PLC memory. For example, the variable `Water_pressure`, is associated with `%MW102`. `Water_pressure` is said to be localized.

NOTE: All variables used in safety applications must be located.

N

naming conventions (identifier)

An identifier is a sequence of letters, numbers and underlines beginning with a letter or underline (e.g. name of a function block type, an instance, a variable or a section). Letters from national character sets (e.g: ö,ü, é, õ) can be used except in project and DFB names. Underlines are significant in identifiers; e.g. `A_BCD` and `AB_CD` are interpreted as different identifiers. Multiple leading underlines and consecutive underlines are invalid.

Identifiers cannot contain spaces. Not case sensitive; e.g. `ABCD` and `abcd` are interpreted as the same identifier.

According to IEC 61131-3 leading digits are not allowed in identifiers. Nevertheless, you can use them if you activate in dialog **Tools** →**Project settings** in tab **Language extensions** the check box **Leading digits**.

Identifiers cannot be keywords.

NAN

Used to indicate that a result of an operation is not a number (NAN = Not A Number).

Example: calculating the square root of a negative number.

NOTE: The IEC 559 standard defines two classes of NAN: quiet NAN (QNAN) and signaling NaN (SNaN) QNAN is a NAN with the most significant fraction bit set and a SNAN is a NAN with the most significant fraction bit clear (Bit number 22). QNANs are allowed to propagate through most arithmetic operations without signaling an exception. SNAN generally signal an invalid-operation exception whenever they appear as operands in arithmetic operations (See %SW17 and %S18).

network

There are 2 meanings for network.

Used...	Meaning
in LD	A network is a set of interconnected graphic elements. The scope of a network is local to the program organization unit (section) in which the network is located.
with communication expert modules	A network is a group of stations which communicate among one another. The term network is also used to define a group of interconnected graphic elements. This group forms then a part of a program which may be composed of a group of networks.

P

procedure

Procedures are functions view technically.

The only difference to elementary functions is that procedures can take up more than 1 output and they support data type VAR_IN_OUT. To the eye, procedures are no different than elementary functions.

Procedures are a supplement to IEC 61131-3.

S

SAI module

safety analog input module

SDI module

safety digital input module

T

TIME

The type TIME expresses a duration in milliseconds. Coded in 32 bits, this type makes it possible to obtain periods from 0 to 2^{32}-1 milliseconds.

The units of type TIME are the following: the days (d), the hours (h), the minutes (m), the seconds (s) and the milliseconds (ms). A literal value of the type TIME is represented by a combination of previous types preceded by T#, t#, TIME# or time#.

Examples: T#25h15m, t#14.7S, TIME#5d10h23m45s3ms

time literals

The units of type TIME are the following: the days (d), the hours (h), the minutes (m), the seconds (s) and the milliseconds (ms). A literal value of the type TIME is represented by a combination of previous types preceded by T#, t#, TIME# or time#.

Examples: T#25h15m, t#14.7S, TIME#5d10h23m45s3ms

U

UDINT

UDINT is the abbreviation of Unsigned Double Integer format (coded on 32 bits) unsigned. The lower and upper limits are as follows: 0 to (2 to the power of 32) - 1.

Example

0, 4294967295, 2#11111111111111111111111111111111, 8#37777777777, 16#FFFFFFFF.

UINT

unsigned integer format (coded on 16 bits)

The lower and upper limits are as follows: 0 to (2 to the power of 16) - 1.

Example

0, 65535, 2#1111111111111111, 8#177777, 16#FFFF.

UMA

UMA is the abbreviation of unrestricted memory area. The UMA is a part of the Safety PLC's memory for %M and %MW that is not write protected. Its addresses can be written from other PLCs or HMIs. Its configuration is performed in the **CPU property dialog** of Unity Pro XLS.

unlocated variable

An unlocated variable is a variable for which it is impossible to know its position in the PLC memory. A variable which have no address assigned is said to be unlocated.

NOTE: It is not allowed to use unlocated variables in safety applications.

V

variable

Memory entity of the type BOOL, WORD, DWORD, etc., whose contents can be modified by the program during execution.

W

WORD

The WORD type is coded in 16 bit format and is used to carry out processing on bit strings.

This table shows the lower/upper limits of the bases which can be used:

Base	Lower Limit	Upper Limit
Hexadecimal	16#0	16#FFFF
Octal	8#0	8#177777
Binary	2#0	2#1111111111111111

Representation examples

Data Content	Representation in One of the Bases
0000000011010011	16#D3
1010101010101010	8#125252
0000000011010011	2#11010011

Index

Symbols

0-9

A

B

C

D

E

F

H

I