# EcoStruxure Machine Expert

## Toolbox

## Library Guide

EIO0000000096.09
12/2023

www.se.com

# Legal Information

The Schneider Electric brand and any trademarks of Schneider Electric SE and its subsidiaries referred to in this guide are the property of Schneider Electric SE or its subsidiaries. All other brands may be trademarks of their respective owners.

This guide and its content are protected under applicable copyright laws and furnished for informational use only. No part of this guide may be reproduced or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), for any purpose, without the prior written permission of Schneider Electric.

Schneider Electric does not grant any right or license for commercial use of the guide or its content, except for a non-exclusive and personal license to consult it on an "as is" basis. Schneider Electric products and equipment should be installed, operated, serviced, and maintained only by qualified personnel.

As standards, specifications, and designs change from time to time, information contained in this guide may be subject to change without notice.

To the extent permitted by applicable law, no responsibility or liability is assumed by Schneider Electric and its subsidiaries for any errors or omissions in the informational content of this material or consequences arising out of or resulting from the use of the information contained herein.

As part of a group of responsible, inclusive companies, we are updating our communications that contain non-inclusive terminology. Until we complete this process, however, our content may still contain standardized industry terms that may be deemed inappropriate by our customers.

# Table of Contents

# Safety Information

## Important Information

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, service, 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 a hazardous situation which, if not avoided, **will result in** death or serious injury.

**⚠ WARNING**

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

**⚠ CAUTION**

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

***NOTICE***

***NOTICE*** is used to address practices not related to physical injury.

## 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.

# QUALIFICATION OF PERSONNEL

A qualified person is one who has the following qualifications:

- Skills and knowledge related to the construction and operation of electrical equipment and the installation.
- Knowledge about providing machine functionality in software implementation.
- Received safety-related training to recognize and avoid the hazards involved.

The qualified person must be able to detect possible hazards that may arise from parameterization, modifying parameter values and generally from mechanical,

electrical, or electronic equipment. The qualified person must be familiar with the standards, provisions, and regulations for the prevention of industrial accidents, which they must observe when designing and implementing the system.

# PROPER USE

This product is a library to be used together with the control systems and servo amplifiers intended solely for the purposes as described in the present documentation as applied in the industrial sector.

Always observe the applicable safety-related instructions, the specified conditions, and the technical data.

Perform a risk evaluation concerning the specific use before using the product. Take protective measures according to the result.

Since the product is used as a part of an overall system, you must ensure the safety of the personnel by means of the design of this overall system (for example, machine design).

Any other use is not intended and may be hazardous.

# Before You Begin

Do not use this product on machinery lacking effective point-of-operation guarding. Lack of effective point-of-operation guarding on a machine can result in serious injury to the operator of that machine.

**⚠WARNING**

**UNGUARDED EQUIPMENT**

- Do not use this software and related automation equipment on equipment which does not have point-of-operation protection.
- Do not reach into machinery during operation.

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

This automation equipment and related software is used to control a variety of industrial processes. The type or model of automation equipment suitable for each application will vary depending on factors such as the control function required, degree of protection required, production methods, unusual conditions, government regulations, etc. In some applications, more than one processor may be required, as when backup redundancy is needed.

Only you, the user, machine builder or system integrator can be aware of all the conditions and factors present during setup, operation, and maintenance of the machine and, therefore, can determine the automation equipment and the related safeties and interlocks which can be properly used. When selecting automation and control equipment and related software for a particular application, you should refer to the applicable local and national standards and regulations. The National Safety Council's Accident Prevention Manual (nationally recognized in the United States of America) also provides much useful information.

In some applications, such as packaging machinery, additional operator protection such as point-of-operation guarding must be provided. This is necessary if the operator's hands and other parts of the body are free to enter the pinch points or other hazardous areas and serious injury can occur. Software products alone cannot protect an operator from injury. For this reason the software cannot be substituted for or take the place of point-of-operation protection.

Ensure that appropriate safeties and mechanical/electrical interlocks related to point-of-operation protection have been installed and are operational before

placing the equipment into service. All interlocks and safeties related to point-of-operation protection must be coordinated with the related automation equipment and software programming.

**NOTE:** Coordination of safeties and mechanical/electrical interlocks for point-of-operation protection is outside the scope of the Function Block Library, System User Guide, or other implementation referenced in this documentation.

# Start-up and Test

Before using electrical control and automation equipment for regular operation after installation, the system should be given a start-up test by qualified personnel to verify correct operation of the equipment. It is important that arrangements for such a check are made and that enough time is allowed to perform complete and satisfactory testing.

**⚠WARNING**

**EQUIPMENT OPERATION HAZARD**

- Verify that all installation and set up procedures have been completed.
- Before operational tests are performed, remove all blocks or other temporary holding means used for shipment from all component devices.
- Remove tools, meters, and debris from equipment.

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

Follow all start-up tests recommended in the equipment documentation. Store all equipment documentation for future references.

**Software testing must be done in both simulated and real environments.**

Verify that the completed system is free from all short circuits and temporary grounds that are not installed according to local regulations (according to the National Electrical Code in the U.S.A, for instance). If high-potential voltage testing is necessary, follow recommendations in equipment documentation to prevent accidental equipment damage.

Before energizing equipment:

- Remove tools, meters, and debris from equipment.
- Close the equipment enclosure door.
- Remove all temporary grounds from incoming power lines.
- Perform all start-up tests recommended by the manufacturer.

# Operation and Adjustments

The following precautions are from the NEMA Standards Publication ICS 7.1-1995:

(In case of divergence or contradiction between any translation and the English original, the original text in the English language will prevail.)

- Regardless of the care exercised in the design and manufacture of equipment or in the selection and ratings of components, there are hazards that can be encountered if such equipment is improperly operated.

- It is sometimes possible to misadjust the equipment and thus produce unsatisfactory or unsafe operation. Always use the manufacturer’s instructions as a guide for functional adjustments. Personnel who have access to these adjustments should be familiar with the equipment manufacturer’s instructions and the machinery used with the electrical equipment.
- Only those operational adjustments required by the operator should be accessible to the operator. Access to other controls should be restricted to prevent unauthorized changes in operating characteristics.

# About the Book

## Document Scope

This document describes the functions of the EcoStruxure Machine Expert Toolbox Library.

## Validity Note

This document has been updated for the release of EcoStruxure™ Machine Expert V2.2.

The characteristics that are described in the present document, as well as those described in the documents included in the Related Documents section below, can be found online. To access the information online, go to the Schneider Electric home page www.se.com/ww/en/download/.

The characteristics that are described in the present document should be the same as those characteristics that appear online. In line with our policy of constant improvement, we may revise content over time to improve clarity and accuracy. If you see a difference between the document and online information, use the online information as your reference.

## Related Documents

| Document title | Reference |
|---|---|
| Cybersecurity Best Practices | CS-Best-Practices-2019-340 |
| Cybersecurity Guidelines for EcoStruxure Machine Expert, Modicon and PacDrive Controllers and Associated Equipment | EIO0000004242 |
| EcoStruxure Machine Expert, Functions and Libraries User Guide | EIO0000002829 (ENG);<br>EIO0000002830 (FRE);<br>EIO0000002831 (GER);<br>EIO0000002832 (ITA);<br>EIO0000002833 (SPA);<br>EIO0000002834 (CHS) |
| EcoStruxure Machine Expert Programming Guide | EIO0000002854 (ENG);<br>EIO0000002855 (FRE);<br>EIO0000002856 (GER);<br>EIO0000002857 (ITA);<br>EIO0000002858 (SPA);<br>EIO0000002859 (CHS) |

## Product Related Information

**NOTE:** Schneider Electric adheres to industry best practices in the development and implementation of control systems. This includes a "Defense-in-Depth" approach to secure an Industrial Control System. This approach places the controllers behind one or more firewalls to restrict access to authorized personnel and protocols only.

## ⚠WARNING

**UNAUTHENTICATED ACCESS AND SUBSEQUENT UNAUTHORIZED MACHINE OPERATION**

- Evaluate whether your environment or your machines are connected to your critical infrastructure and, if so, take appropriate steps in terms of prevention, based on Defense-in-Depth, before connecting the automation system to any network.
- Limit the number of devices connected to a network to the minimum necessary.
- Isolate your industrial network from other networks inside your company.
- Protect any network against unintended access by using firewalls, VPN, or other, proven security measures.
- Monitor activities within your systems.
- Prevent subject devices from direct access or direct link by unauthorized parties or unauthenticated actions.
- Prepare a recovery plan including backup of your system and process information.

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

For more information on organizational measures and rules covering access to infrastructures, refer to ISO/IEC 27000 series, Common Criteria for Information Technology Security Evaluation, ISO/IEC 15408, IEC 62351, ISA/IEC 62443, NIST Cybersecurity Framework, Information Security Forum - Standard of Good Practice for Information Security and refer to Cybersecurity Guidelines for EcoStruxure Machine Expert, Modicon and PacDrive Controllers and Associated Equipment.

## ⚠WARNING

**LOSS OF CONTROL**

- Perform a Failure Mode and Effects Analysis (FMEA), or equivalent risk analysis, of your application, and apply preventive and detective controls before implementation.
- Provide a fallback state for undesired control events or sequences.
- Provide separate or redundant control paths wherever required.
- Supply appropriate parameters, particularly for limits.
- Review the implications of transmission delays and take actions to mitigate them.
- Review the implications of communication link interruptions and take actions to mitigate them.
- Provide independent paths for control functions (for example, emergency stop, over-limit conditions, and error conditions) according to your risk assessment, and applicable codes and regulations.
- Apply local accident prevention and safety regulations and guidelines.[1]
- Test each implementation of a system for proper operation before placing it into service.

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

[1] For additional information, refer to NEMA ICS 1.1 (latest edition), *Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control* and to NEMA ICS 7.1 (latest edition), *Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems* or their equivalent governing your particular location.

Before you attempt to provide a solution (machine or process) for a specific application using the POUs found in the library, you must consider, conduct and complete best practices. These practices include, but are not limited to, risk analysis, functional safety, component compatibility, testing and system validation as they relate to this library.

**⚠WARNING**

**IMPROPER USE OF PROGRAM ORGANIZATION UNITS**

- Perform a safety-related analysis for the application and the devices installed.
- Ensure that the Program Organization Units (POUs) are compatible with the devices in the system and have no unintended effects on the proper functioning of the system.
- Ensure that the axis is homed and that the homing is valid before usage of absolute movements or POUs using absolute movements.
- Use appropriate parameters, especially limit values, and observe machine wear and stop behavior.
- Verify that the sensors and actuators are compatible with the selected POUs.
- Thoroughly test all functions during verification and commissioning in all operation modes.
- Provide independent methods for critical control functions (emergency stop, conditions for limit values being exceeded, etc.) according to a safety-related analysis, respective rules, and regulations.

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

**⚠WARNING**

**UNINTENDED EQUIPMENT OPERATION**

- Only use software approved by Schneider Electric for use with this equipment.
- Update your application program every time you change the physical hardware configuration.

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

## Terminology Derived from Standards

The technical terms, terminology, symbols and the corresponding descriptions in this manual, or that appear in or on the products themselves, are generally derived from the terms or definitions of international standards.

In the area of functional safety systems, drives and general automation, this may include, but is not limited to, terms such as *safety*, *safety function*, *safe state*, *fault*, *fault reset*, *malfunction*, *failure*, *error*, *error message*, *dangerous*, etc.

Among others, these standards include:

| Standard | Description |
|---|---|
| IEC 61131-2:2007 | Programmable controllers, part 2: Equipment requirements and tests. |
| ISO 13849-1:2015 | Safety of machinery: Safety related parts of control systems.<br>General principles for design. |
| EN 61496-1:2013 | Safety of machinery: Electro-sensitive protective equipment.<br>Part 1: General requirements and tests. |
| ISO 12100:2010 | Safety of machinery - General principles for design - Risk assessment and risk reduction |
| EN 60204-1:2006 | Safety of machinery - Electrical equipment of machines - Part 1: General requirements |
| ISO 14119:2013 | Safety of machinery - Interlocking devices associated with guards - Principles for design and selection |
| ISO 13850:2015 | Safety of machinery - Emergency stop - Principles for design |
| IEC 62061:2015 | Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems |
| IEC 61508-1:2010 | Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements. |
| IEC 61508-2:2010 | Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems. |
| IEC 61508-3:2010 | Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements. |
| IEC 61784-3:2016 | Industrial communication networks - Profiles - Part 3: Functional safety fieldbuses - General rules and profile definitions. |
| 2006/42/EC | Machinery Directive |
| 2014/30/EU | Electromagnetic Compatibility Directive |
| 2014/35/EU | Low Voltage Directive |

In addition, terms used in the present document may tangentially be used as they are derived from other standards such as:

| Standard | Description |
|---|---|
| IEC 60034 series | Rotating electrical machines |
| IEC 61800 series | Adjustable speed electrical power drive systems |
| IEC 61158 series | Digital data communications for measurement and control – Fieldbus for use in industrial control systems |

Finally, the term *zone of operation* may be used in conjunction with the description of specific hazards, and is defined as it is for a *hazard zone* or *danger zone* in the *Machinery Directive* (*2006/42/EC*) and *ISO 12100:2010*.

**NOTE:** The aforementioned standards may or may not apply to the specific products cited in the present documentation. For more information concerning the individual standards applicable to the products described herein, see the characteristics tables for those product references.

# Function Requirements

## What’s in This Part



## Overview

The part describes the Function Requirements.

# Function Requirements

## What's in This Chapter



## Overview

This chapter describes the function requirements.

# Toolbox Function Library Requirements

## Hardware Requirements

The Toolbox Library can be used with all Schneider Electric controllers managed with EcoStruxure Machine Expert software.

## Using the Library

**⚠WARNING**

**UNINTENDED EQUIPMENT OPERATION**

- Verify the EcoStruxure Machine Expert libraries contained in your program are the correct version after updating EcoStruxure Machine Expert software.
- Verify that the library versions updated are consistent with your application specifications.

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

For more detailed information, see Schneider Electric Libraries (see EcoStruxure Machine Expert, Libraries Overview).

# Bit Functions

## What's in This Part



## Overview

This part describes the bit function family.

# Bit Organization

## What's in This Chapter



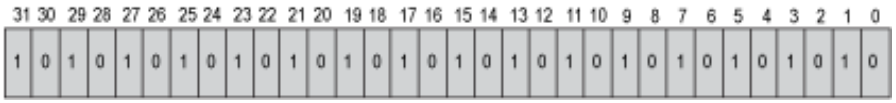
## Overview

This chapter describes the Bit Organization.

# Bit Organization for DWORD

## Bit Organization for DWORD

This figure shows the codage rule of the bit position in a DWORD. An example is given for the 16#AAAAAAAA corresponding to the value of 2863311530.




## Integer Data Types Description

This table shows the integer data types. Each of the different number types covers a different range types.

| Type | Lower limit | Upper limit | Memory space |
|---|---|---|---|
| BYTE | 0 | 255 | 8 Bit |
| WORD | 0 | 65535 | 16 Bit |
| DWORD | 0 | 4294967295 | 32 Bit |
| LWORD | 0 | $2^{64}$-1 | 64 Bit |
| SINT | -128 | 127 | 8 Bit |
| USINT | 0 | 255 | 8 Bit |
| INT | -32768 | 32767 | 16 Bit |
| UINT | 0 | 65535 | 16 Bit |
| DINT | -2147483648 | 2147483647 | 32 Bit |
| UDINT | 0 | 4294967295 | 32 Bit |
| LINT | $-2^{63}$ | $2^{63}$-1 | 64 Bit |
| ULINT | 0 | $2^{64}$-1 | 64 Bit |

## REAL/LREAL Data Types Description

This table shows the REAL/LREAL data types. REAL and LREAL are called floating-point types. They are required to represent rational numbers.

| Type | Range | Resolution | Memory space |
|---|---|---|---|
| REAL uses 4 bytes | -3.402e+38...3.402e+38<br>(-2^128...2^128) | 1,175e-38<br>(2^-126) | 32 Bit |
| LREAL uses 8 bytes | -1.797e+308...1.797e+308<br>(-2^1024...2^1024) | 2,225e-308<br>(2^-1022) | 64 Bit |

**NOTE:** The support of data type LREAL depends on the target device. Please see in the corresponding documentation whether the 64 bit type LREAL gets converted to REAL during compilation (possibility with a loss of information) or persists.

# SetBitTo: Setting/Resetting One Bit

## What's in This Chapter



## Overview

This chapter describes the SetBitTo function.

# SetBitTo Function

## Pin Diagram

This figure shows the pin diagram of the SetBitTo function:


## Functional Description

The SetBitTo function sets/resets (according to set) one bit specified by the bit in the given DWORD input. The bits are counted from low to high starting with 0.

The valid range is 0 to 31.

## Input Pin Description

This table describes the input pins of the SetBitTo function:

| Input | Data Type | Description |
|---|---|---|
| i_dwIput | DWORD | Input value<br>Range: 0...4294967295 |
| i_iPos | INT | Bit position<br>Range: 0...31 |
| i_xSet | BOOL | TRUE: Set<br>FALSE: Reset. |

## Output Pin Description

This table describes the output pins of the SetBitTo function:

| Output | Data Type | Description |
|---|---|---|
| SetBitTo | DWORD | Output value<br>Range: 0...4294967295 |

# Limitations

If the `i_iPos` input is not within the valid range, the input will be interpretated in modulo mode.

# TestBit: Testing One Bit

## What's in This Chapter



## Overview

This chapter describes the TestBit function.

# TestBit Function

## Pin Diagram

This figure shows the pin diagram of the TestBit function:


## Functional Description

The TestBit function tests one bit specified by the bit in the given DWORD input. The bits are counted from low to high starting with 0.

The output shows the status of presence of bit in that specified position. The valid range is 0 to 31.

## Input Pin Description

This table describes the input pins of the TestBit function:

| Input | Data Type | Description |
|---|---|---|
| i_dwIput | DWORD | Input value<br>Range: 0...4294967295 |
| i_iPos | INT | Bit position<br>Range: 0...31 |

## Output Pin Description

This table describes the output pins of the TestBit function:

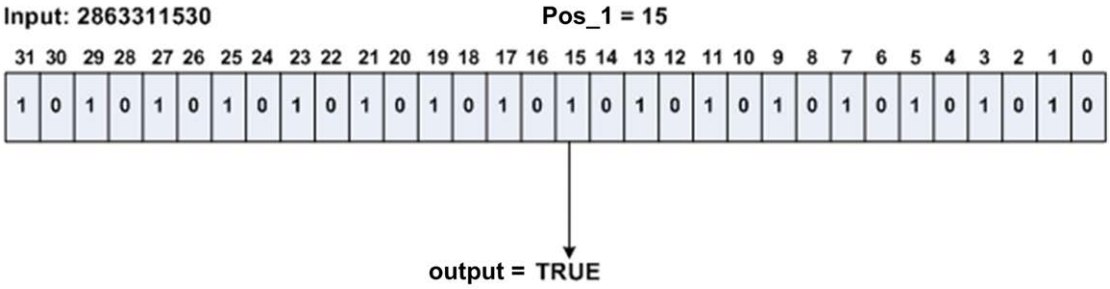
| Output | Data Type | Description |
|---|---|---|
| TestBit | BOOL | Result is True or False |

# Limitations

If the `i_iPos` input is not within the valid range, the input will be interpretated in modulo mode.

# Usage Example


This figure shows an example of the `TestBit` function:



# Closed Loop Functions

## What's in This Part



## Overview

The part describes the closed loop function family.

# FB_2points: Closed Loop 2 Point Transfer Element

## What's in This Chapter



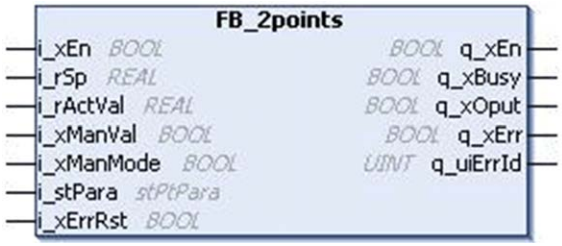
## Overview

This chapter describes the FB_2points transfer element function block.

# FB_2points Function Block

## Pin Diagram

This figure shows the pin diagram of the FB_2points function block:




## Functional Description

The FB_2points function block provides a 2 point transfer function based on hysteresis input.

# Operating Modes

## Automatic Mode

In automatic mode (i_xEn is TRUE and i_xManMode is FALSE), if the calculated process error is greater than 50% of the hysteresis in the positive direction, then q_xOput is TRUE.

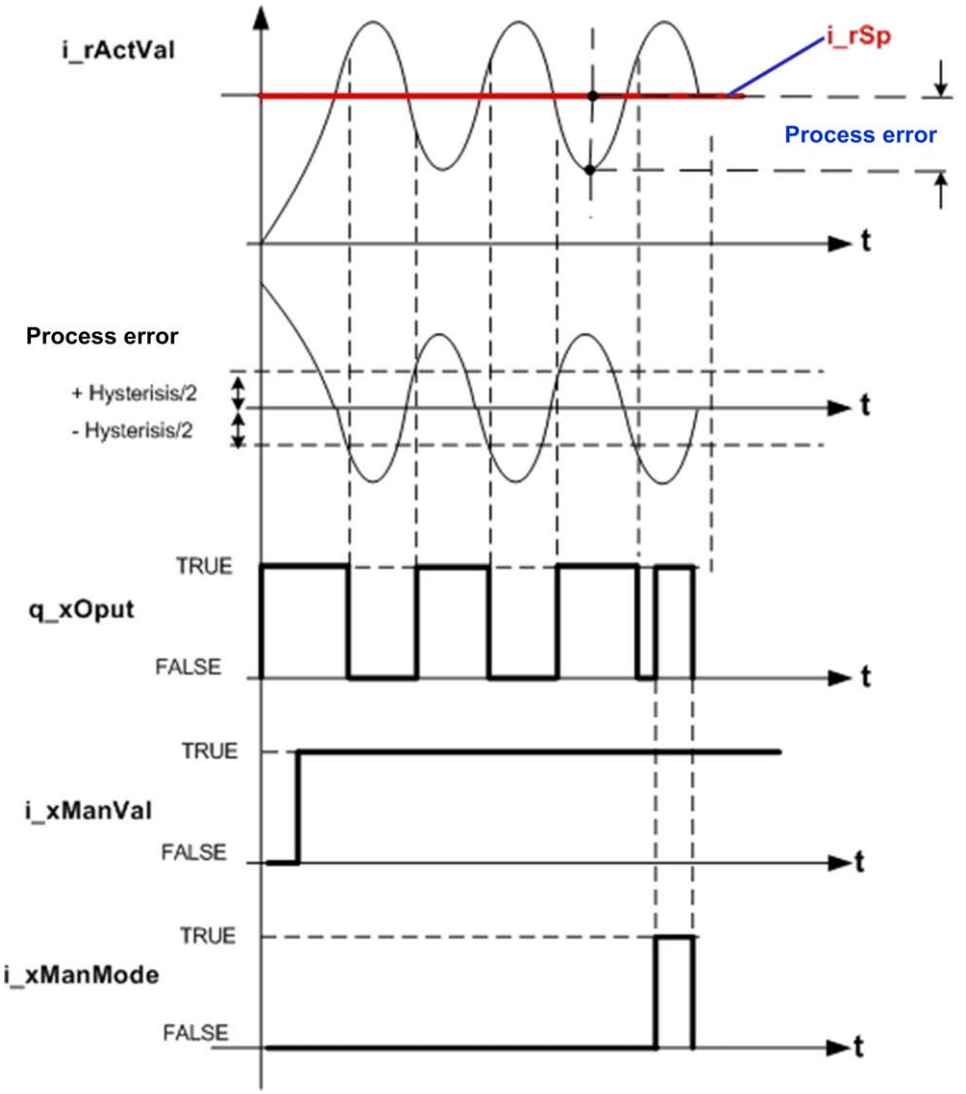
q_xOput is reset only if the calculated process error goes below 50% of hysteresis in negative direction.

Process error = Setpoint value – Actual value.

## Manual Mode

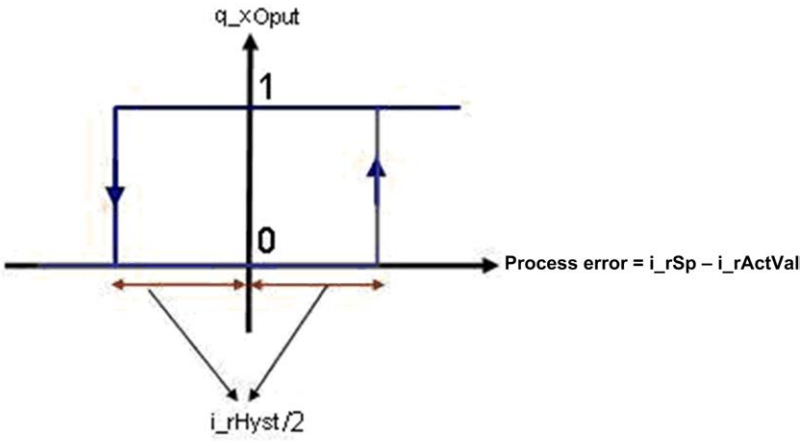
If i_xManMode is TRUE regardless of the current inputs, the output is set to the state of i_xManVal.

This figure shows the timing diagram for the `FB_2points` function block.




## Mode Timing Diagram

This figure shows the transfer function for the `FB_2points` function block.



# Detected Error State

An invalid parameter at the function block inputs results in a detected error and the corresponding detected error ID is generated.

During the error detected state, the output value is set to zero. Detected error can be reset only through rising edge of `i_xErrRst` input.

The output `q_xBusy` is TRUE, whenever the function block is enabled and when there is no detected error.

# Input Pin Description

## Input Pin Table

This table describes the input pins of the `FB_2points` function block.

| Input | Data Type | Description |
|---|---|---|
| `i_xEn` | `BOOL` | TRUE: Enables the function block<br>FALSE: Disables the function block |
| `i_rSp` | `REAL` | Set point value<br>Range: ±3.4e$^{+38}$ |
| `i_rActVal` | `REAL` | Actual value<br>Range: ±3.4e$^{+38}$ |
| `i_xManVal` | `BOOL` | Manual value<br>(Optional) |
| `i_xManMode` | `BOOL` | Manual mode<br>(Optional) |
| `i_stPara` | `STRUCT stPtPara` | Structure parameter<br>(Please refer the table below). |
| `i_xErrRst` | `BOOL` | Reset for detected error<br>(Rising edge resets detected error.)<br>(Optional) |

# Structure Used

## *stPtPara*

| Structure Element | Type | Description |
|---|---|---|
| *tCyclTime* | TIME | Task cycle time<br>Range: 1...1e$^{32}$ ms |
| *rHyst* | REAL | Range of the hysteresis should be greater than 0.0<br>Range: ±3.4e$^{+38}$ |

# Output Pin Description

## Output Pin Table

This table describes the output pins of the `FB_2points` function block.

| Output | Data Type | Description |
|---|---|---|
| `q_xEn` | `BOOL` | TRUE: function block enabled<br>FALSE: Disabled |
| `q_xBusy` | `BOOL` | TRUE: function block active and no detected error.<br>FALSE: function block disabled or function block detected error |
| `q_xOput` | `BOOL` | TRUE: If process error is greater than 50% of hysteresis<br>FALSE: If `q_xOput` is TRUE and detected error goes below 50% of hysteresis in negative direction. |
| `q_xErr` | `BOOL` | TRUE: During detected error<br>FALSE: If no detected error |
| `q_uiErrId` | `UNIT` | 0 = No detected error<br>1 = Invalid parameter task Cycle time = 0<br>2 = Invalid parameter `i_rHyst` <= 0. |

# FB_3points: Closed Loop 3 Points Transfer Element

## What's in This Chapter



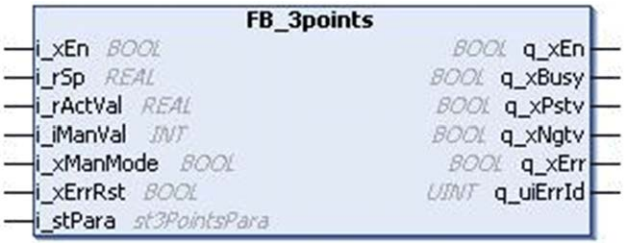
## Overview

This chapter describes the FB_3points transfer element function block.

# FB_3points Function Block

## Pin Diagram

This figure shows the pin diagram of the FB_3points function block:




## Functional Description

The FB_3points function block provides a 3-point transfer element in the functional diagram.

# Operating Modes

## Automatic mode

This function block checks the value of process error (set point - actual value).

If the process error is positive and more than the upper threshold value for positive side rPstvOutOn, it sets the q_xPstv output signal.

q_xPstv output is reset if the process error decreases below the lower threshold value for positive side rPstvOutOff.

If process error is negative, q_xNgtv output signal is set upon overshooting upper threshold value for negative side rNgtvOutOn.

q_xNgtv is reset upon decreasing below lower threshold value for negative side rNgtvOutOff.

# Manual Mode

The function block output is set manually according to the value of the `i_iManVal` input pin.

If

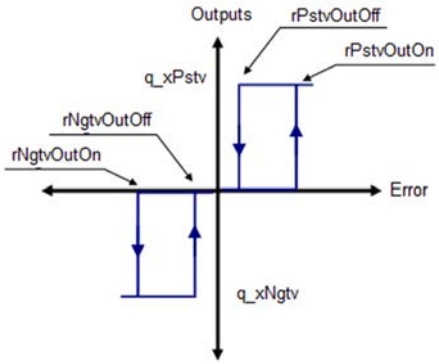
`i_iManVal` >= 1 then `q_xPstv` = TRUE, `q_xNgtv` = FALSE.

`i_iManVal` <= -1 then `q_xPstv` = FALSE, `q_xNgtv` = TRUE.

Else

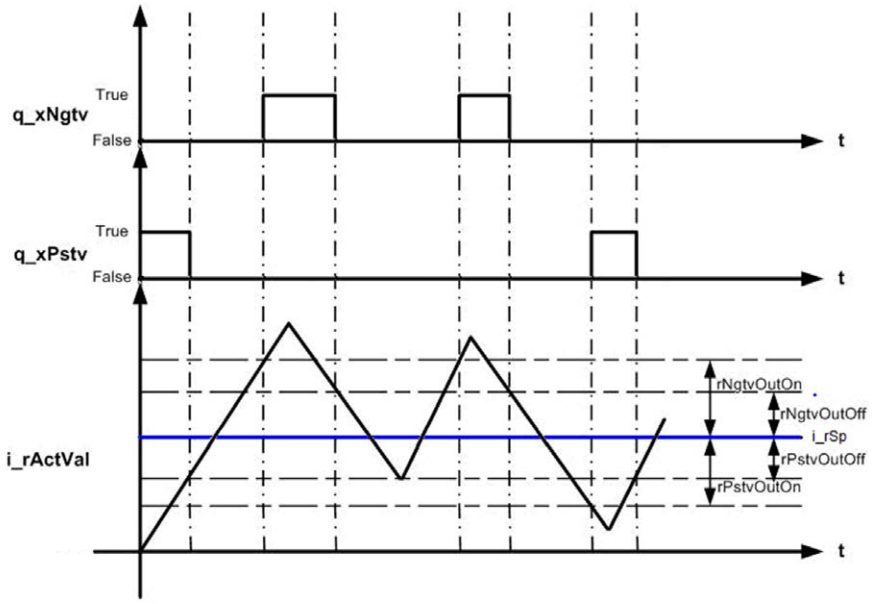
`q_xPstv` = FALSE, `q_xNgtv` = FALSE

This figure shows the transfer function for the `FB_3points` function block:




# Timing Diagram

This figure shows the timing diagram for the `FB_3points` function block:




# Detected Error State

An invalid parameter at the function block inputs results in a detected error and corresponding detected error ID is generated.

During the error detected state the output values are set to zero. Detected error can be reset only through rising edge of `i_xErrRst` input.

The output `q_xBusy` is TRUE, whenever the function block is enabled and when there is no detected error.

# Input Pin Description

## Input Pin Table

This table describes the input pins of the `FB_3points` function block:.

| Input | Data Type | Description |
|---|---|---|
| `i_xEn` | `BOOL` | TRUE: Enables the function block<br>FALSE: function block disabled |
| `i_rSp` | `REAL` | Set point value of the process<br>Range: ±3.4e$^{+38}$ |
| `i_rActVal` | `REAL` | Actual value of the process<br>Range: ±3.4e$^{+38}$ |
| `i_iManVal` | `INT` | Input value for manual mode.<br>Range: -32768...32767 |
| `i_xManMode` | `BOOL` | TRUE: function block in manual mode<br>FALSE: function block in automatic mode<br>(Optional) |
| `i_xErrRst` | `BOOL` | Reset for detected error<br>(Rising edge triggered.)<br>(Optional) |
| `i_stPara` | `STRUCT st3PointsPara` | Function block structure variable |

# Structure Used

## *st3PointsPara*

| Structure Element | Type | Description |
|---|---|---|
| *rPstvOutOn* | REAL | Upper threshold value for positive side of process error<br>Range: 0...1e$^{38}$ |
| *rPstvOutOff* | REAL | Lower threshold value for positive side of process error<br>Range: 0...1e$^{38}$ |
| *rNgtvOutOn* | REAL | Upper threshold value for negative side of process error<br>Range: 0...1e$^{38}$ |
| *rNgtvOutOff* | REAL | Lower threshold value for negative side of process error<br>Range: 0...1e$^{38}$ |

# Output Pin Description

## Output Pin Table

This table describes the output pins of the `FB_3points` function block:.

| Output | Data Type | Description |
|---|---|---|
| `q_xEn` | `BOOL` | TRUE: Function block enabled<br>FALSE: Disabled |
| `q_xBusy` | `BOOL` | TRUE: Function block active and no detected error.<br>FALSE: Function block disabled or detected error |
| `q_xPstv` | `BOOL` | This output from the 3-points element is TRUE if the upper branch of the hysteresis curve is active. |
| `q_xNgtv` | `BOOL` | This output from the 3-points element is TRUE if the lower branch of the hysteresis curve is active. |
| `q_xErr` | `BOOL` | Detected error signal |
| `q_uiErrId` | `UINT` | Supplies the detected error Id when `q_xErr` output is set.<br>Range: 0...3 |

## q_uiErrId

This unique integer value indicates some particular detected error:

| Detected error ID | Description |
|---|---|
| 0 | No detected error |
| 1 | Invalid parameter values (If `rPstvOutOn` < 0 or `rPstvOutOff` < 0 or `rNgtvOutOff` < 0 or `rNgtvOutOn` < 0) |
| 2 | Invalid parameter values (`rPstvOutOn` < `rPstvOutOff` or `rNgtvOutOn` < `rNgtvOutOff`) |
| 3 | Internal detected error (function block in unknown state) |

# FB_3points_Ext: Closed Loop Extension For 3 Points Transfer Element

## What's in This Chapter



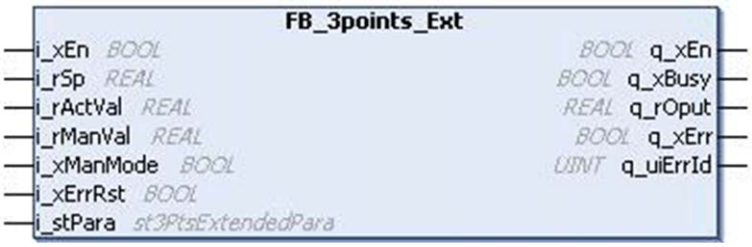
## Overview

This chapter describes the FB_3points_Ext function block.

# FB_3points_Ext Function Block

## Pin Diagram

This figure shows the pin diagram of the FB_3points_Ext function block:




## Functional Description

The FB_3points_Ext function block provides a 3-point transfer element in the functional diagram.

This function block is an extension for FB_3points function block. It produces a control output q_rOput in the form of 3-point hysteresis loop. The output depends upon the process error and the gain and offset value that are given by the user.

# Operating Modes

## Automatic Mode

This function block checks the value of process error (difference between set point and actual value). If the process error is positive and greater than the upper threshold value rOutOn, it calculates the control output as below,

q_rOput = Process error x rGain + rOfst

If the process error decreases below the lower threshold value rOutOff, it resets the control output to zero.

Similarly if process error is negative, and its absolute value is more than upper threshold value rOutOn, it calculates the control output as below,

`q_rOput` = Process error x `rGain` - `rOfst`

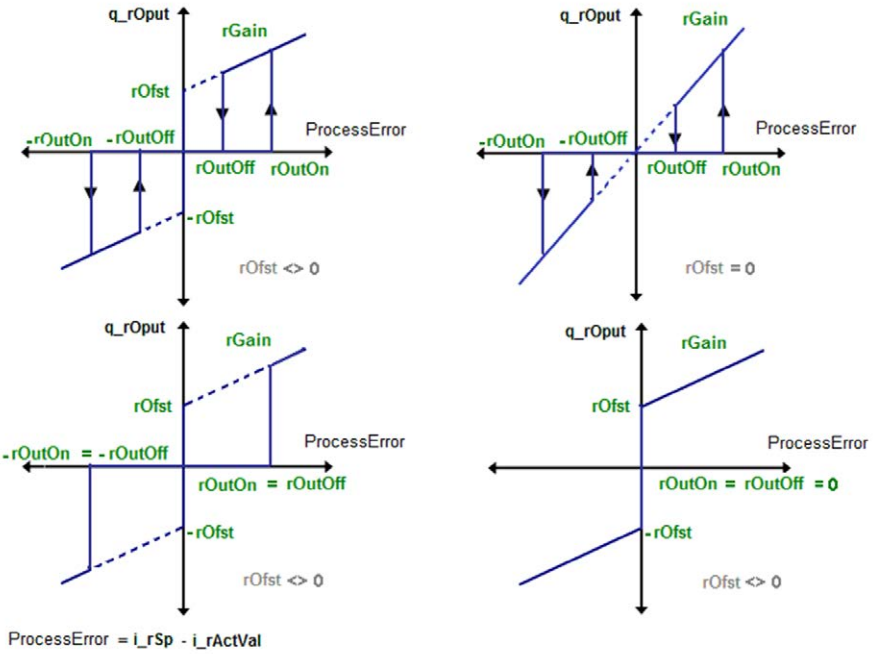
`q_rOput` is reset to zero if absolute value of process error becomes less than lower threshold value `rOutOff`.

# Manual Mode

The function block output is set manually as according to the value of input pin `i_rManVa`:.

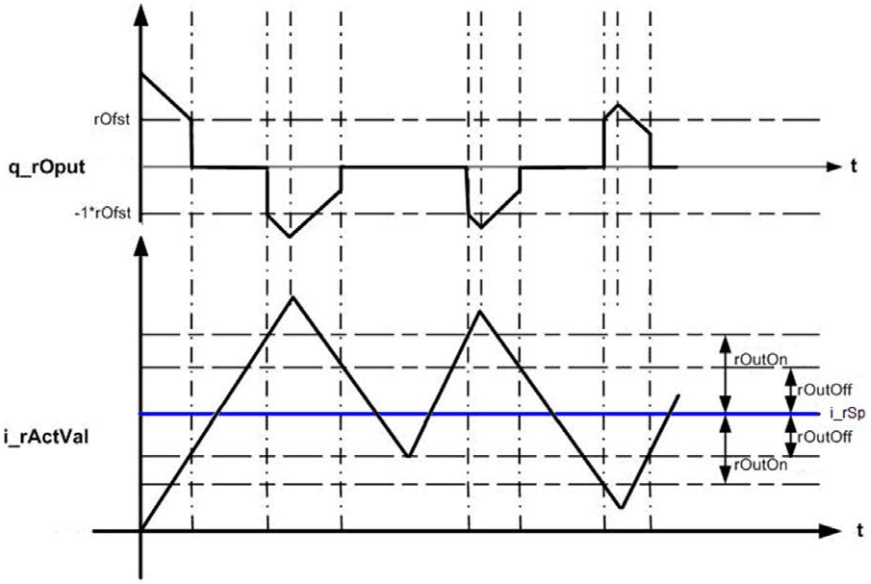
| IF | AND IF | THEN |
|---|---|---|
| `Abs(i_rManVal)` < 1 | - | `q_rOput` = 0.0 |
| `Abs(i_rManVal)` >= 1 | `rE` > 0 | `q_rOput` = `rE` x `rGain` + `rOfst` |
| `Abs(i_rManVal)` >= 1 | `rE` < 0 | `q_rOput` = `rE` x `rGain` - `rOfst` |
| `Abs(i_rManVal)` >= 1 | `rE` = 0 | `q_rOput` = 0.0 |
| **rE** = `i_rSp` - `i_rActVal`<br>**Abs()** Absolute value function. | | |

This figure shows the transfer function for the `FB_3points_Ext` function block:



# Timing Diagram

This figure shows the timing diagram for the `FB_3points_Ext` function block:




# Detected Error State

An invalid parameter at the function block input results in a detected error state and a corresponding detected error ID is generated.

During the error detected state the output values are set to zero. Detected error can be reset only through rising edge of `i_xErrRst` input.

The output `q_xBusy` is TRUE whenever the function block is enabled and when there is no detected error.

# Input Pin Description

## Input Pin Table

This table describes the input pins of the `FB_3points_Ext` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xEn` | `BOOL` | TRUE: Enables the function block<br>FALSE: function block disabled |
| `i_rSp` | `REAL` | Set point value of the process<br>Range: ±3.4e$^{+38}$ |
| `i_rActVal` | `REAL` | Actual value of the process<br>Range: ±3.4e$^{+38}$ |
| `i_rManVal` | `REAL` | Input value for manual mode.<br>Range: ±3.4e$^{+38}$<br>(Optional) |
| `i_xManMode` | `BOOL` | TRUE: function block in manual mode<br>FALSE: function block in automatic mode |

| Input | Data Type | Description |
|---|---|---|
| | | (Optional) |
| i_xErrRst | BOOL | Reset for detected error<br>(Rising edge triggered)<br>(Optional) |
| i_stPara | STRUCTst3PtsExtendedPara | Function block structure variable |

# Structure Used

## *st3PtsExtendedPara*

| Structure Element | Type | Description |
|---|---|---|
| *rOutOn* | REAL | Upper threshold value<br>Range: 0...1e$^{38}$ |
| *rOutOff* | REAL | Lower threshold value<br>Range: 0...1e$^{38}$ |
| *rGain* | REAL | Gain<br>Range: 0...1e$^{38}$ |
| *rOfst* | REAL | Offset<br>Range: 0...1e$^{38}$ |

# Output Pin Description

## Output Pin Table

This table describes the output pins of the FB_3points_Ext function block:

| Output | DataType | Description |
|---|---|---|
| q_xEn | BOOL | TRUE: Function block enabled<br>FALSE: Disabled |
| q_xBusy | BOOL | TRUE: Function block active and no detected error.<br>FALSE: Function block disabled or detected error |
| q_rOput | REAL | Output from the function block.<br>Range: ±3.4e$^{+38}$ |
| q_xErr | BOOL | Detected error signal |

## q_uiErrId

This unique integer value indicates some particular detected error:

| Detected Error ID | Description |
|---|---|
| 0 | No detected error |
| 1 | Invalid parameter values (rOutOn < 0 OR rOutOff < 0) |

| Detected Error ID | Description |
|---|---|
| 2 | Invalid parameter values (`rOutOn` < `rOutOff`) |
| 3 | Invalid parameter values (`rGain` < 0) |
| 4 | Invalid parameter values (`rOfst` < 0) |
| 5 | Internal detected error (function block in unknown state) |

# FB_P: Control Loop With Proportional Only Algorithm

## What's in This Chapter



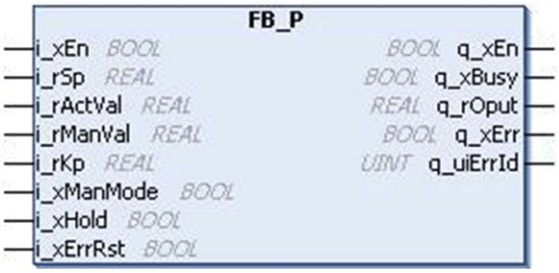
## Overview

This chapter describes the FB_P function block.

# FB_P Function Block

## Pin Diagram

This figure shows the pin diagram of the FB_P function block:




## Functional Description

This FB_P function block is developed to provide a control loop with proportional only algorithm.

# Operation Modes

## Automatic Mode

This function block provides the proportional response i.e. the output is process error times the gain.

$$G(s) = Kp$$

This equation shows the transfer function:

Where:

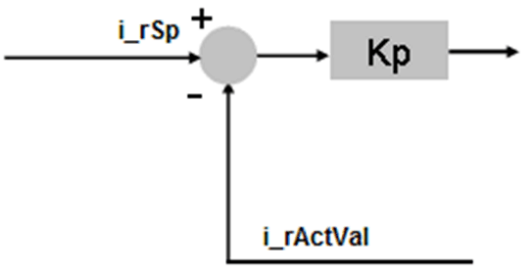
| | |
|---|---|
| Kp | = Proportional gain |

| | |
|---|---|
| q_rOput | = G(s) * Process error |

## Manual Mode

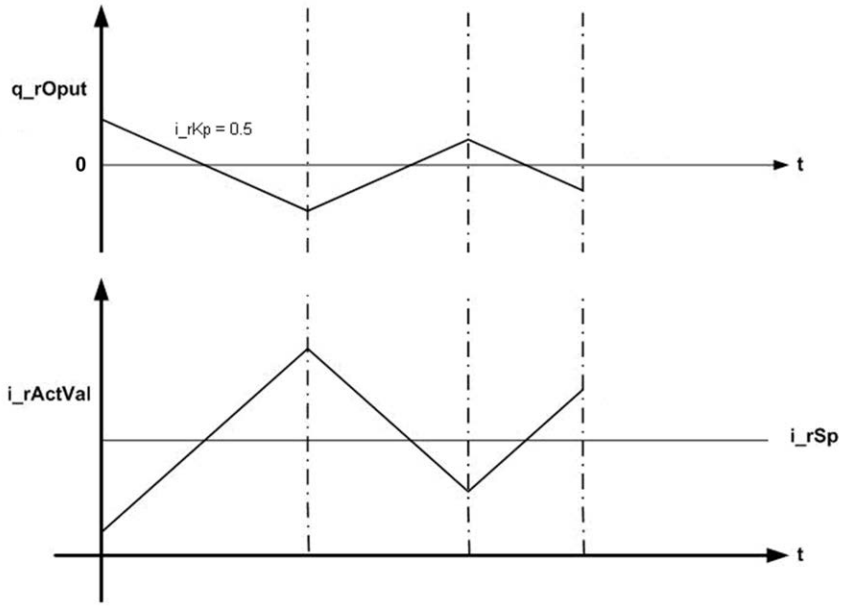
The `q_rOput` function block output is set equal to `i_rManVal`.

This figure shows the function diagram for the `FB_P` function block




## Timing Diagram

This figure shows the timing diagram for the `FB_P` function block




## Detected Error State

An invalid parameter at the function block inputs results in detected error and corresponding detected error ID is generated.

During detected error state, the output values are set to zero. Detected error can be reset only through rising edge of `i_xErrRst` input.

The output `q_xBusy` is TRUE whenever the function block is enabled and there is no detected error.

# Input Pin Description

## Input Pin Table

This table describes the input pins of the `FB_P` function block:

| Input | Data Type | Description |
|---|---|---|
| i_xEn | BOOL | TRUE: Enables the function block<br>FALSE: function block disabled |
| i_rSp | REAL | Set point value of the process<br>Range: ±3.4e+38 |
| i_rActVal | REAL | Actual value of the process<br>Range: ±3.4e+38 |
| i_rManVal | REAL | Input value for manual mode.<br>Range: ±3.4e+38<br>(Optional) |
| i_rKp | REAL | Proportional gain<br>Range: 0...3.4e+38 |
| i_xManMode | BOOL | TRUE: Function block in manual mode<br>FALSE: Function block in automatic mode<br>(Optional) |
| i_xHold | BOOL | TRUE: Hold the internal state and output of the function block constant at its current value.<br>FALSE: Disabled<br>(Optional) |
| i_xErrRst | BOOL | Reset for detected error (rising edge triggered.)<br>(Optional) |

# Output Pin Description

## Output Pin Table

This table describes the output pins of the FB_P function block:

| Output | Data Type | Description |
|---|---|---|
| q_xEn | BOOL | TRUE: Function block enabled<br>FALSE: Disabled |
| q_xBusy | BOOL | TRUE: Function block active and no detected error.<br>FALSE: Function block disabled or detected error |
| q_rOput | REAL | Output from the function block<br>Range: ±3.4e+38 |
| q_xErr | BOOL | Detected error signal |
| q_uiErrId | UINT | Supplies the detected error Id when q_xErr output is set.<br>0: No detected error<br>1: Invalid parameter values (If i_rKp < 0)<br>2: Internal detected error (function block in unknown state) |

# q_uiErrId

This unique integer value indicates some particular detected error

| Detected Error ID | Description |
|---|---|
| 0 | No detected error |
| 1 | Invalid parameter values (`i_rKp` < 0) |
| 2 | Internal detected error (function block in unknown state) |

# FB_PI: Proportional and Integration Process Control

## What's in This Chapter



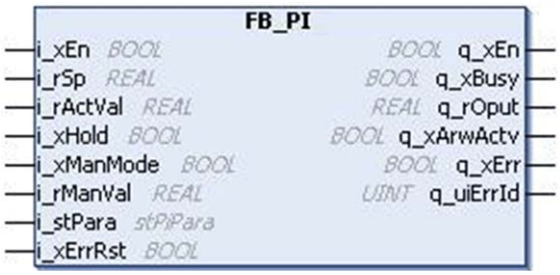
## Overview

This chapter describes the FB_PI Closed Loop process control function block.

# FB_PI Function Block

## Pin Diagram

This figure shows the pin diagram of the FB_PI function block:




## Functional Description

The FB_PI function block is a standard PI function block with manual tuning, hold function, and anti reset wind-up.

The PI controller generates a control output based on the process error in the system (Process error = Set Point – Actual Value). Using the setting of function block parameters, control output can be tuned to reduce the process error.

The proportional and integral value for the process calculated continuously based on the actual value, set point and input parameters. The function block also limits the control output based on limit settings.

## Transfer Function

The following equation is the transfer function for the FB_PI function block:

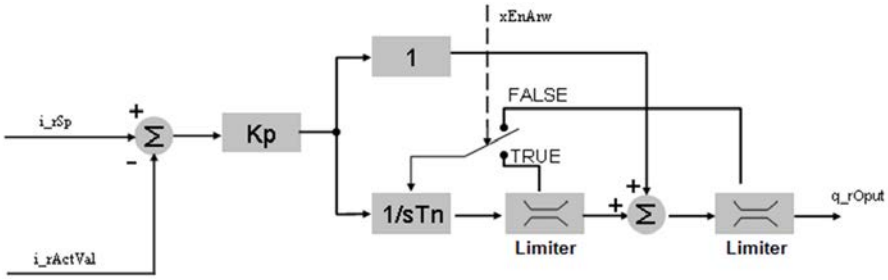
$$G(s) = Kp\left(1 + \frac{1}{sT_n}\right)$$

Where:

| | |
|---|---|
| $K_p$ | = Proportional gain. |
| $sT_n$ | = Integral time |

# Functional Diagram

This figure shows the functional diagram of the `FB_PI` function block:




This function block is used to control the Closed Loop processes with continuous input and output variables.

# Detected Error State

An invalid parameter at the function block inputs results in a detected error and a corresponding detected error ID is generated.

During the error detected state, the output value is set to zero.

Detected error can be reset only through rising edge of `i_xErrRst` input. The output `q_xBusy` is TRUE, whenever the function block is enabled and when there is no detected error.

# Input Pin Description

## Input Pin Table

This table describes the input pins of the `FB_PI` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xEn` | `BOOL` | TRUE: Enables the function block<br>FALSE: Disables the function block |
| `i_rSp` | `REAL` | Set point value<br>Range: ±3.4e$^{+38}$ |
| `i_rActVal` | `REAL` | Actual value<br>Range: ±3.4e$^{+38}$ |
| `i_rManVal` | `REAL` | Manual value<br>Range: ±3.4e$^{+38}$<br>(Optional) |
| `i_xManMode` | `BOOL` | Manual value<br>(Optional) |
| `i_xHold` | `BOOL` | Hold<br>(Optional) |

| Input | Data Type | Description |
|---|---|---|
| i_xErrRst | BOOL | Reset for detected error (rising edge reset detected error)<br>(Optional) |
| i_stPara | STRUCT stPiPara | Structure parameter<br>(Refer to the stPiPara description, page 47.) |

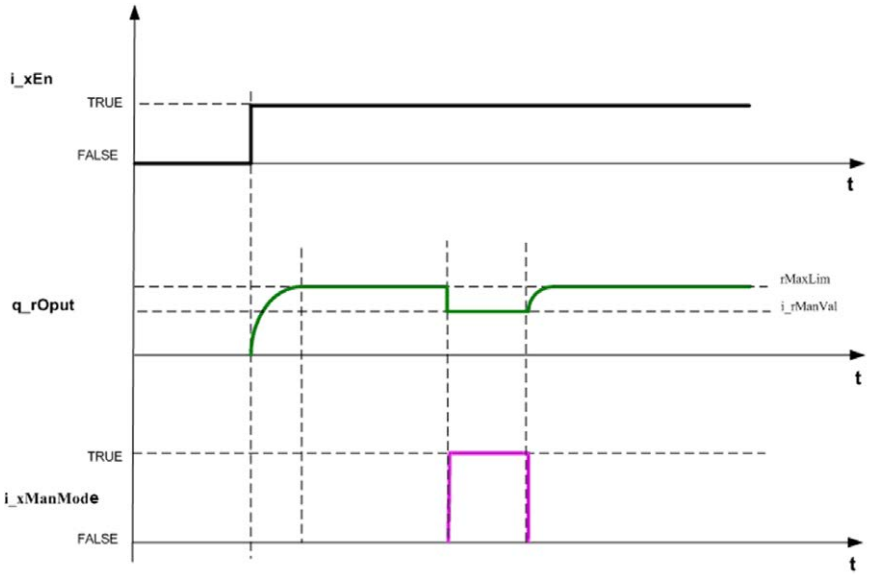
## i_xManMode

i_xManMode decides the manual mode of the FB_PI function block.

If the function block is enabled and manual is set to TRUE, then the function block will set the manual value (i_rManVal) as the PI output and stops the PI algorithm as shown in block diagram function block in manual mode.

If the auto mode is enabled, then PI Algorithm executes continuously.

This figure shows the time diagram of the function block in manual mode:



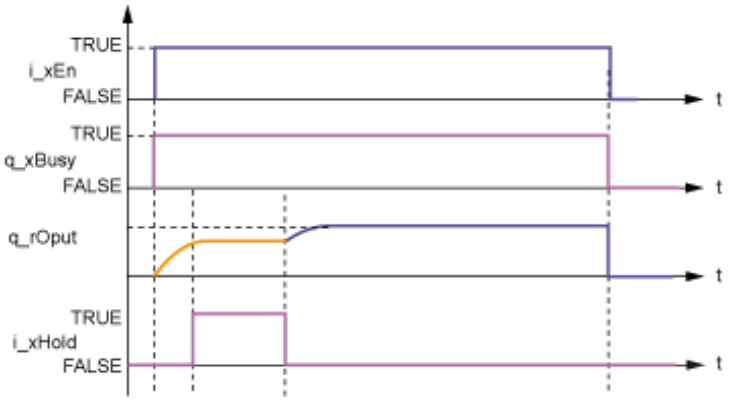

## i_xHold

i_xHold will hold the PI output at current level.

If this input is TRUE, then the PI output will be maintained at last value and internal calculations of PI algorithm will be stopped as shown in block diagram function block on hold.

If this input is FALSE, then PI algorithm is executed cyclically. The new PI output will be calculated from the last value.

Time diagram of the function block on hold:




# Structure Used

## *stPiPara*

| Structure Element | Type | Description |
|---|---|---|
| *tCyclTime* | TIME | Task cycle time<br>Range: 10 ms...60 s |
| *xEnArw* | BOOL | Enable anti reset wind-up |
| *tTn* | TIME | Integral action time<br>Range: 1...1e$^{32}$ ms |
| *rKp* | REAL | Proportional gain Value<br>Range: ±3.4e$^{+38}$ |
| *rMaxLim* | REAL | Maximum Output Limit<br>Range: ±3.4e$^{+38}$ |
| *rMinLim* | REAL | Minimum Output Limit<br>Range: ±3.4e$^{+38}$ |

### `tCyclTime`

`tCyclTime` is the time between the two executions of the function block. If the task is assigned as cyclic, then it is equal to the task cycle time of the cyclic task.

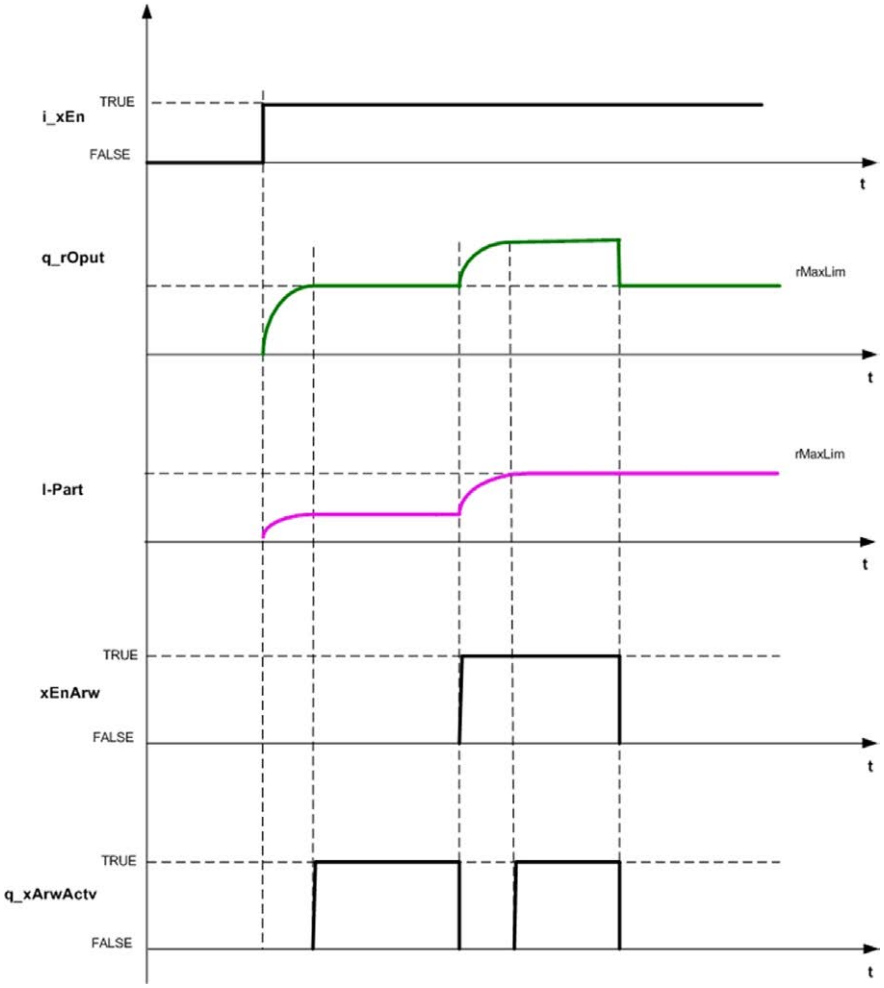
### `xEnArw`

`xEnArw` will enable anti reset wind-up (ARW) operation.

If FALSE, hold the integral part if the entire control output reaches a limit.

If TRUE, the function block holds the integral part only if the integral part reaches a limit. The output is equal to the sum of limit value and proportional part if integral part reaches a limit as shown in block diagram function block on enable ARW.

This figure shows the function block on Enable ARW mode:




## tTn

Integral time for PI loop

## rKp

Proportional gain for PI loop

## rMaxLim

Output greater than this limit is limited to `rMaxLim` value.

## rMinLim

Output less than this limit is limited to `rMinLim` value.

**NOTE:** If the `rMinLim` is greater than 0 then PI then operation starts from the `rMinLim` value.

# Output Pin Description

## Output Pin Table

This table describes the output pins of the `FB_PI` function block.

| Output | Data Type | Description |
|---|---|---|
| `q_xEn` | `BOOL` | TRUE: Function block enabled |
| `q_xBusy` | `BOOL` | TRUE: function block active and no detected error.<br>FALSE: function block disabled or function block detected error |
| `q_rOput` | `REAL` | Calculated PI output<br>Range: ±3.4e$^{+38}$ |
| `q_xArwActv` | `BOOL` | TRUE: Output is limited, anti reset wind-up is active. |
| `q_xErr` | `BOOL` | Detected error signal |
| `q_uiErrId` | `UNIT` | Detected error ID of particular detected error<br>Range: 0...5 |

## `q_uiErrId`

This unique integer value indicates the particular detected error.

| Detected error ID | Description |
|---|---|
| 0 | No detected error |
| 1 | Invalid task cycle time = zero |
| 2 | Invalid parameter `i_rOputMaxLim` < `i_rOputMinLim` |
| 3 | Invalid parameter `rKp` < zero |
| 4 | Invalid parameter `tTn` = zero |
| 5 | Internal detected error (function block in unknown state) |

# FB_PID: Manual Mode Operation Process Control

## What's in This Chapter



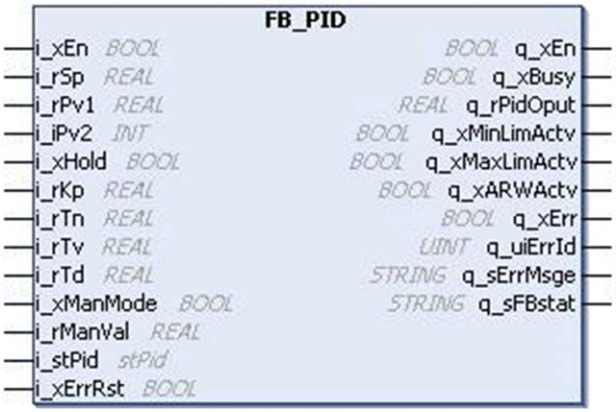
## Overview

This chapter describes the FB_PID function block.

# FB_PID Function Block

## Pin Diagram

This figure shows the pin diagram of the FB_PID function block:




## Functional Description

The FB_PID function block is a standard PID function block with manual tuning, hold function, bumpless transfer and damping time for derivative action.

This function block provides following features:

- Different modes such as P, PI, PD, and PID.
- Manual mode operation to control the PID output in manual mode.
- Anti reset wind-up to avoid the saturation or wind-up in integral action: If the control variable reaches actuator limit, process error will continue to integrate very large integral term is called as windup.
- Damping time (Td) to filter the overshoot due to derivative action.
- Bump less transfer is activated when mode changes from manual to auto.

  Bump less transfer avoids sudden change in PID output when mode changes.
- Detected error status is generated by function block to display detected errors.

- Inner and outer window functions are used in integral calculations.

  If absolute value of process error is less than inner window then integral part is scaled with a factor [ABS (err)/Inner Window].

  This minimizes the overshoot in the PID output.

  If absolute value of process error is greater than inner window and less than outer window then normal integral calculations are done.

  If absolute value of process error is greater than outer window then anti reset windup is active and integral output will hold the last value.

# PID Output

The following equation shows the PID output:

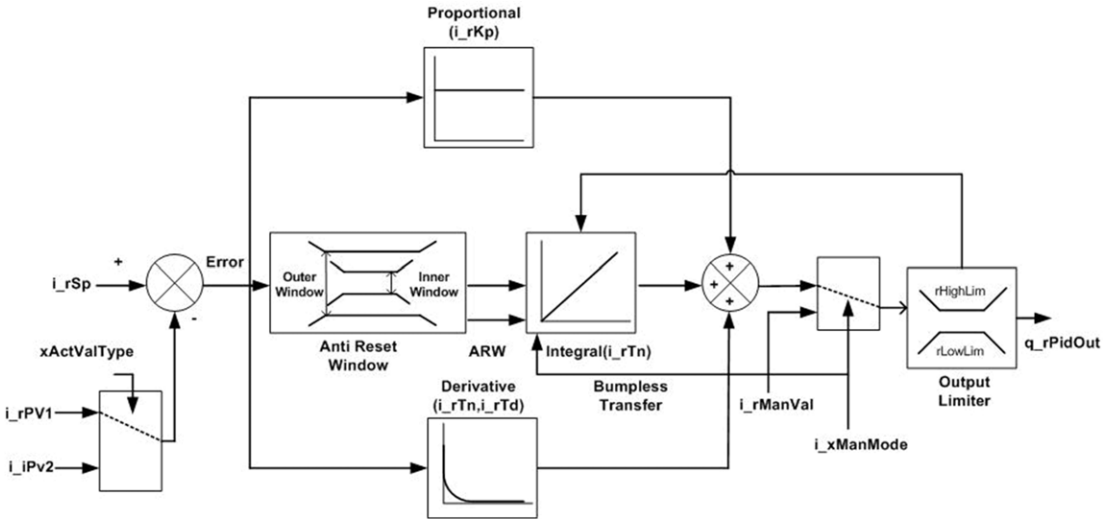
$$y(t) = Kp\left(e(t) + \frac{1}{Tn}\int e(t)dt + \frac{Tv}{1+Td}\frac{de(t)}{dt}\right)$$

Where:

| | |
|---|---|
| y (t) | = PID Output |
| Kp | = Proportional gain |
| Tn | = Integral time |
| Tv | = Derivative time |
| Td | = Filter time for derivative |
| e (t) | = Process error between set point and feedback value. |

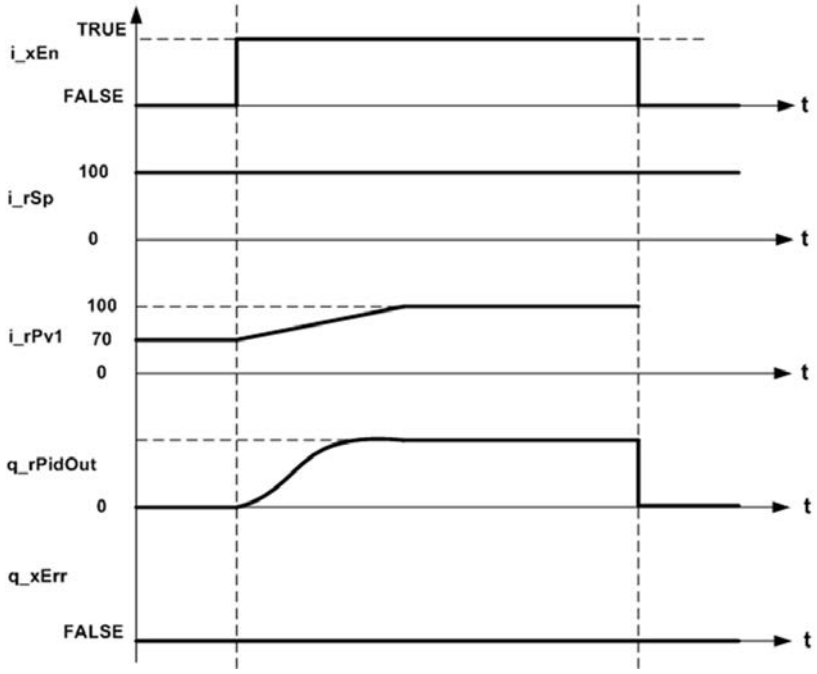
# Schematic Diagram

This figure shows the block diagram for the `FB_PID` function block:



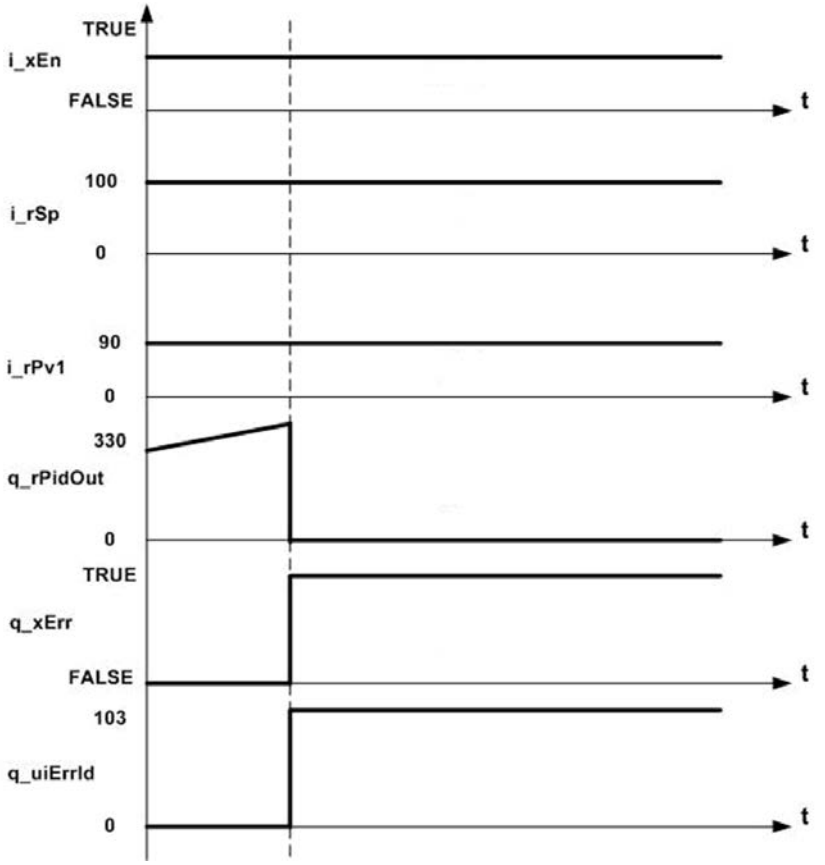
# Normal Behavior Diagram

This figure shows normal behavior diagram of the `FB_PID` function block:




# Detected Error Diagram

This figure shows the `FB_PID` function block diagram with detected error:



# Input Pin Description

## Input Pin Description

This table describes the input pins of the FB_PID function block:

| Input | Data Type | Description |
|---|---|---|
| i_xEn | BOOL | TRUE: Enables FB_PID function block<br>FALSE: Disables FB_PID function block |
| i_rSp | REAL | Set point<br>Range: ±3.4e+38 |
| i_rPv1 | REAL | Feedback value from the process.<br>Range: ±3.4e+38 |
| i_iPv2 | INT | Feedback value from the process.<br>Range: -32768...32767 |
| i_xHold | BOOL | TRUE: Holds the PID output at current level and stops the PID calculations.<br>FALSE: Normal running of PID Mode.<br>(Optional) |
| i_rKp | REAL | Proportional gain for PID<br>Range: 0.0...3.4e+38 |
| i_rTn | REAL | Integral time for PID<br>Range: 0.0...60000 milliseconds |
| i_rTv | REAL | Derivative time for PID control<br>Range: 0.0...60000 milliseconds |
| i_rTd | REAL | Damping time for derivative action<br>Range: 60000.0 > i_rTd > i_stPid.rTargCyclTime |
| i_xManMode | BOOL | TRUE: Manual mode<br>FALSE: Auto mode (factory setting)<br>(Optional) |
| i_rManVal | REAL | Manual PID output value when i_xManMode is TRUE.<br>Range: ±3.4e+38<br>(Optional) |
| i_stPid | STRUCT st_Pid | Parameter structure<br>(Refer to the , page 54i_stPid description.) |
| i_xErrRst | BOOL | TRUE: Reset detected error<br>(Optional)<br>(Rising edge resets the detected error) |

### i_xEn

If TRUE, enables the function block.

If FALSE, then PID output set to zero and detected error status (q_xErr) is cleared and detected error id (q_uiErrId) is set to zero.

### i_xHold

If TRUE, then PID output is maintained at last value and internal calculations of PID algorithm are stopped.

If FALSE, then PID algorithm is executed cyclically.

New PID output is calculated from last value.

### i_xErrRst

This resets the current detected error.

On rising edge of this input, detected error is reset.

The PID output is set to zero.

### i_xManMode

If TRUE, then PID output is equal to `i_rManVal`.

If FALSE then as per bump less transfer, PID output starts from `i_rManVal` and PID algorithm starts execution.

## Structure Used

### *stPid*

| Structure Element | Type | Description |
|---|---|---|
| *xActValType* | BOOL | TRUE: `i_rPv1` is process value.<br>FALSE: `i_iPv2` is process value. |
| *rDbnd* | REAL | Dead band value for internal process error calculation.<br>Range: 0.0...100.0<br>(Optional) |
| *rTargCyclTime* | REAL | Target cycle time<br>Range: 60000.0 > `rTargCyclTime` > 0.0<br>Unit: ms |
| *rLowLim* | REAL | Lower limit of PID output<br>Range: ±3.4e$^{+38}$ |
| *rHighLim* | REAL | High limit of PID output<br>Range: ±3.4e$^{+38}$ |
| *rInerWndo* | REAL | Inner window for reduced Integral part.<br>Range: 0.0...3.4e$^{+38}$ |
| *rOterWndo* | REAL | Outer window for disabling Integral part.<br>Range: 0.0...3.4e$^{+38}$ |

**NOTE:** *xActValType*, *rLowLim*, *rHighLim*, *rInerWndo*, *rOterWndo* accept the new state or value change on rising edge of *i_xEn*.

## xActValType

Selects the process value.

## rDbnd

Dead band value is used for process error calculation.

If absolute value of process error is greater than dead band then process error value is calculated as: Process error = Set point - Actual Feedback Value.

If absolute value of process error is less than dead band then process error = 0.0.

## rTargCyclTime

Target cycle time for PID.

Cycle time can be measured and entered as fix value or current cycle time can be measured in each controller scan as target cycle time.

Parameter *stPid.rTargCyclTime* = 0 results detected error with ID20: Invalid CycleTime

Parameter *stPid.rTargCyclTime* greater than derivative dumping time (Td) results detected error with ID201: Incorrect Td parameter

## rLowLim

Lower limit of PID output.

If internal PID output is less than *rLowLim* then PID, output is clamped to *rLowLim*.

## rHighLim

Higher limit of PID output.

If internal PID output is greater than *rHighLim* then PID output is clamped to *rHighLim*.

## rInerWndo

If absolute value of process error is less than *rInerWndo* then Integral calculations are scaled by factor [ABS (detected error) / Inner Window].

Process error is difference between set point and actual process value.

If process error is greater than *rInerWndo* and less than *rOterWndo* then normal integral calculations are done.

This window is use to reduce overshoot in PID output.

Process error = Set Point - Process value

## rOterWndo

If absolute value of process error is greater than *rOterWndo* then Integral calculations are stop and integral output is held at last value.

In the PID output maximum contribution is due to integral action.

If process error is not reducing even though PID output is changing, then PID output saturate due to windup in integral action.

To avoid saturation and windup, *rOterWndo* parameter is required.

# Output Pin Description

## Output Pin Table

This table describes the output pins of the `FB_PID` function block:

| Output | Data Type | Description |
|---|---|---|
| `q_xEn` | `BOOL` | TRUE: function block enabled.<br>FALSE: function block disabled. |
| `q_xBusy` | `BOOL` | TRUE: PID is active and no internal detected error.<br>FALSE: PID is not active or detected error. |
| `q_rPidOput` | `REAL` | The output of PID controller.<br>Range: `i_stPid.rLowLim`...`i_stPid.rHighLim` |
| `q_xMinLimActv` | `BOOL` | TRUE: If PID output Less than or equal to `i_stPid.rLowLim` (Minimum limit).<br>FALSE: If PID output greater than `i_stPid.rLowLim` (Minimum limit). |
| `q_xMaxLimActv` | `BOOL` | TRUE: If PID output greater than or equal to `i_stPid.rHighLim` (Maximum limit).<br>FALSE: If PID output less than `i_stPid.rHighLim` (Maximum limit). |
| `q_xARWActv` | `BOOL` | TRUE: Anti reset windup is active.<br>FALSE: Anti reset windup is not active. |
| `q_xErr` | `BOOL` | TRUE: function block detected error<br>FALSE: No detected error. |
| `q_uiErrId` | `UNIT` | Indicates the detected error number when the detected error output is set.<br>Range: 0, 100, 103, 104, 105, 106, 107 |
| `q_sErrMsge` | `STRING` | Detected error message |
| `q_sFBstat` | `STRING` | Function block status |

### q_xEn

True to indicate the enable status of function block.

### q_xBusy

TRUE if function block is started without any detected error.

If any detected error occurs, then busy output goes low.

## q_rPidOput

The FB_PID starts calculating the output each cycle, if there is no detected error.

## q_xARWActv

Indicate status of anti reset windup.

If integral time is greater than zero, integral action is active.

If TRUE, then integral action is stop and integral output is held at last value.

TRUE:

- Case 1: Integral time > 0.0 and ((PID Output >=Maximum limit) and (Process error > 0.0))
- Case 2: Integral time > 0.0 and ((PID Output <=Minimum limit) and (Process error < 0.0))

## q_xErr

TRUE indicates the detected error and PID output is set to zero.

## q_uiErrId & q_sErrMsge

This gives detected error number and detected error message when q_xErr is TRUE.

| Detected Error ID | Description |
|---|---|
| 0 | No detected error |
| 1 | Internal detected error |
| 20 | Invalid cycle time |
| 114 | Invalid limit parameter |
| 115 | Invalid dead band limit |
| 200 | Incorrect PID parameter |
| 201 | Incorrect Td parameter |
| 202 | Incorrect I window |

## q_sFBstat

| | |
|---|---|
| FB Active: | Function block is active and working without detected error. |

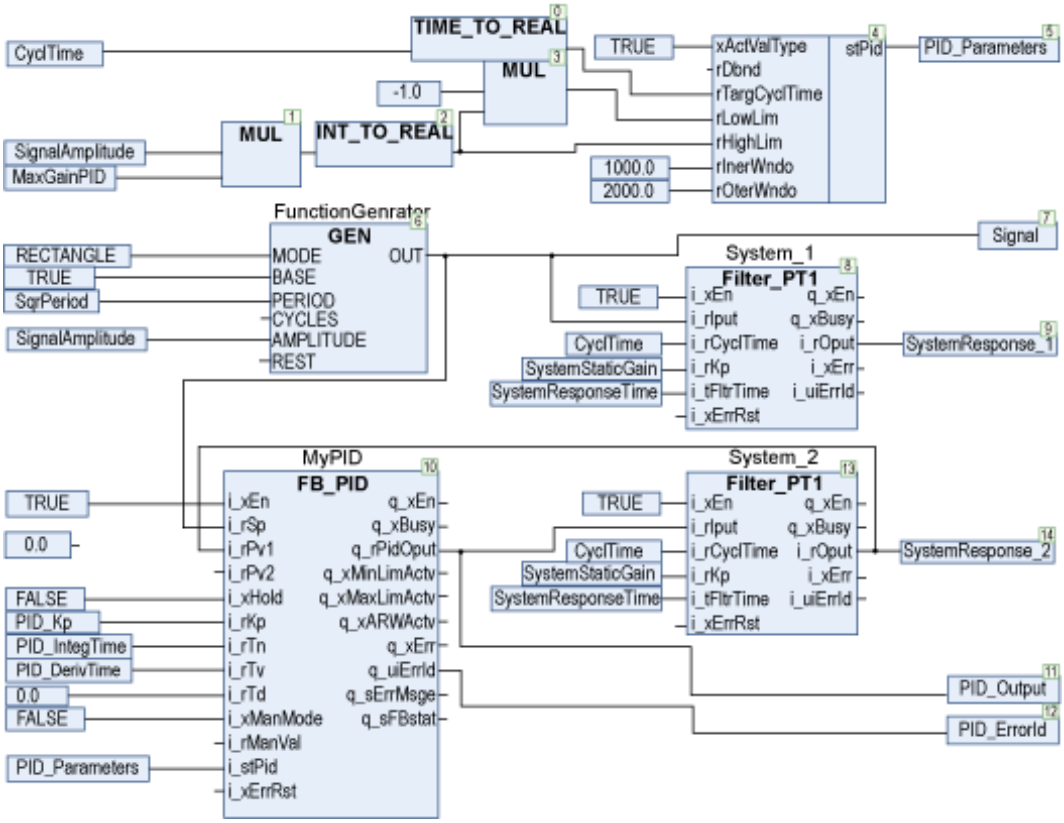
| | |
|---|---|
| FB Detected error: | Function block is active and there is a detected error. |

| | |
|---|---|
| FB Disabled: | Function block is disabled. |

# Instantiation and Usage Example

## Instantiation and Usage Example

This figure shows an instance of the `FB_PID` function block:



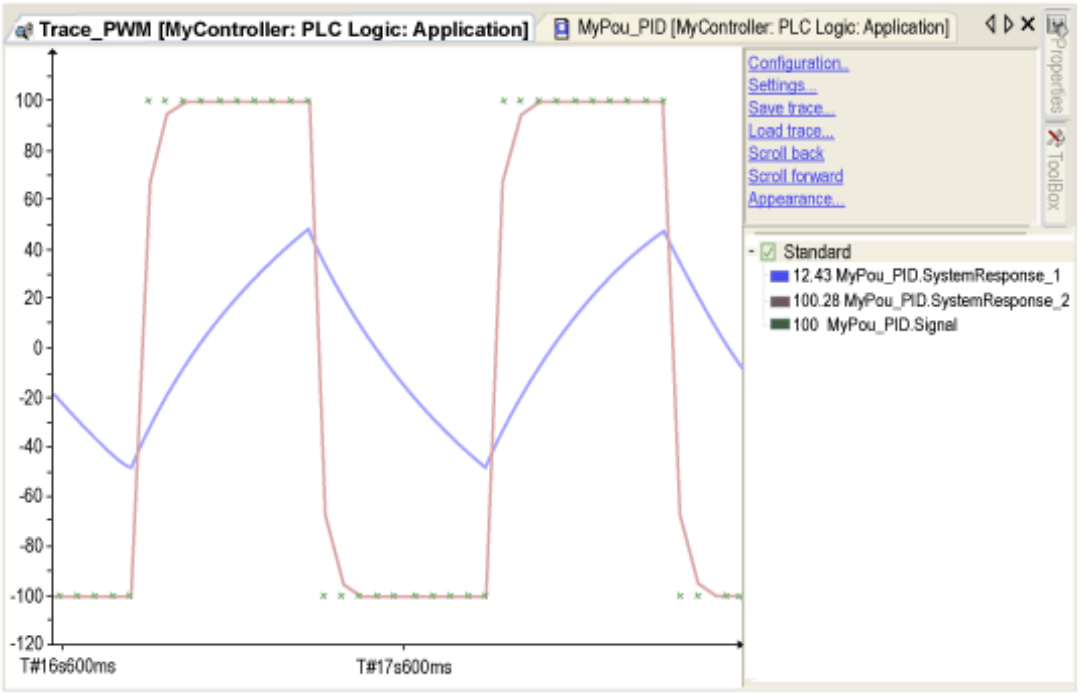

- A square signal is generated using the GEN, key parameters are `SqrPeriod` and `SignalAmplitude`.
- The system to control is a simple first order filter, key parameters are `SystemResponseTime` and `SystemStaticGain`.
- A trace is done in open loop `SystemResponse_1` and in closed loop using `FB_PID` function block.

Data of this example are:

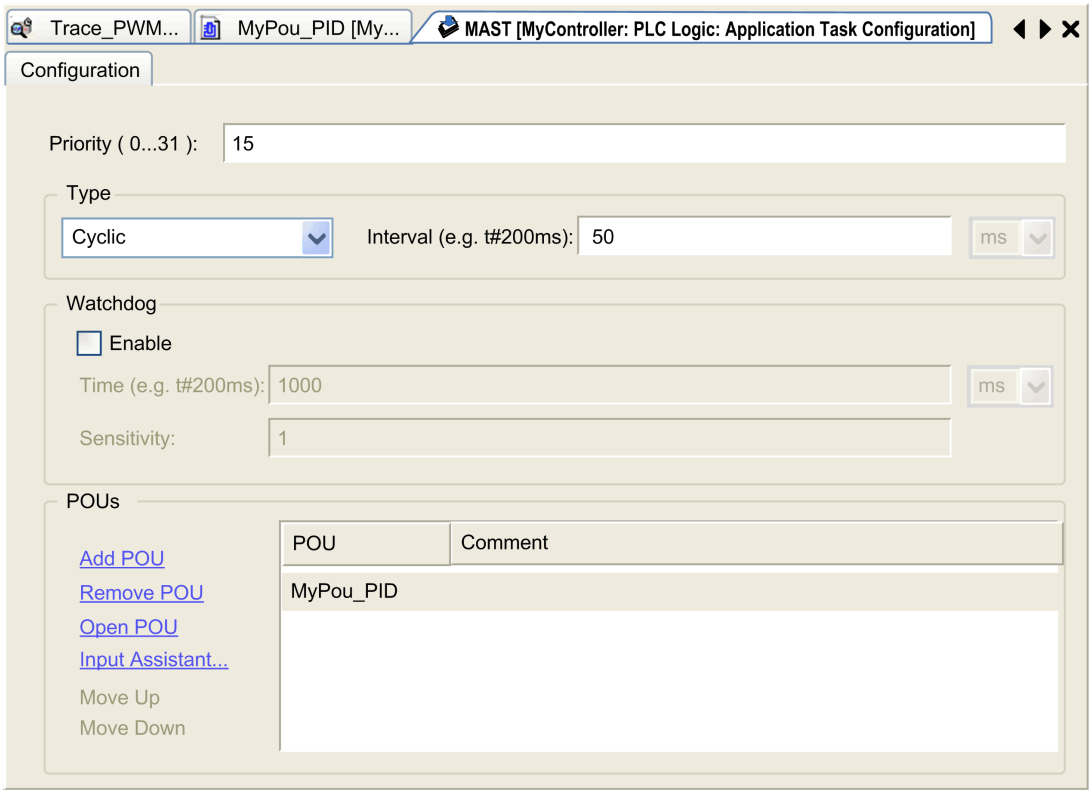
Trace_PWM [MyController: PLC Logic: Application] | **MyPou_PID [MyController: PLC Logic: Application]**

```
PROGRAM MyPou_PID
VAR
  FlagDemarrer        : BOOL;
  FunctionGenerator   : GEN;
  SqrPeriod           : TIME := T#1000MS;
  SignalAmplitude     : INT := 100;
  Signal              : INT;
  System_1            : Filter_PT1;
  System_2            : Filter_PT1;
  CyclTime            : TIME := T#50MS;(* Should have the same value than the periodicity of the POU in the MAST*)
  SystemStaticGain    : REAL := 1.0;
  SystemResponseTime  : TIME := T#500MS;
  SystemResponse_1    : REAL;
  SystemResponse_2    : REAL;

  MyPID               : FB_PID;
  PID_Parameters      : stPid;
  PID_Kp              : REAL := 7.5;
  PID_IntegTime       : REAL := 44.0;
  PID_DerivTime       : REAL := 0.0;
  PID_ErrorId         : UINT;
  PID_Output          : REAL;
  MaxGainPID          : INT := 20;
END_VAR
```

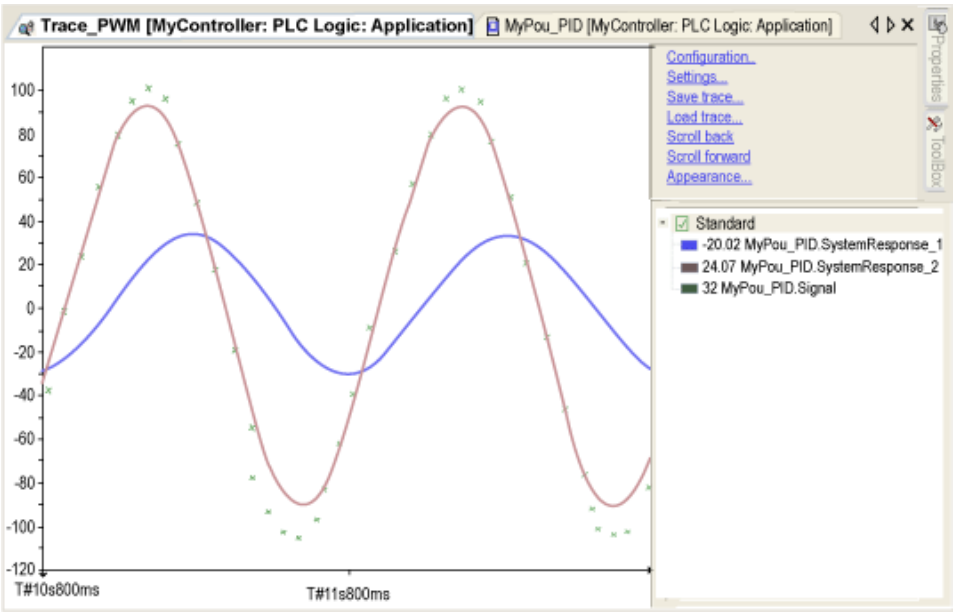
Using the previous setting, the setpoint/open loop/closed loop answer is:



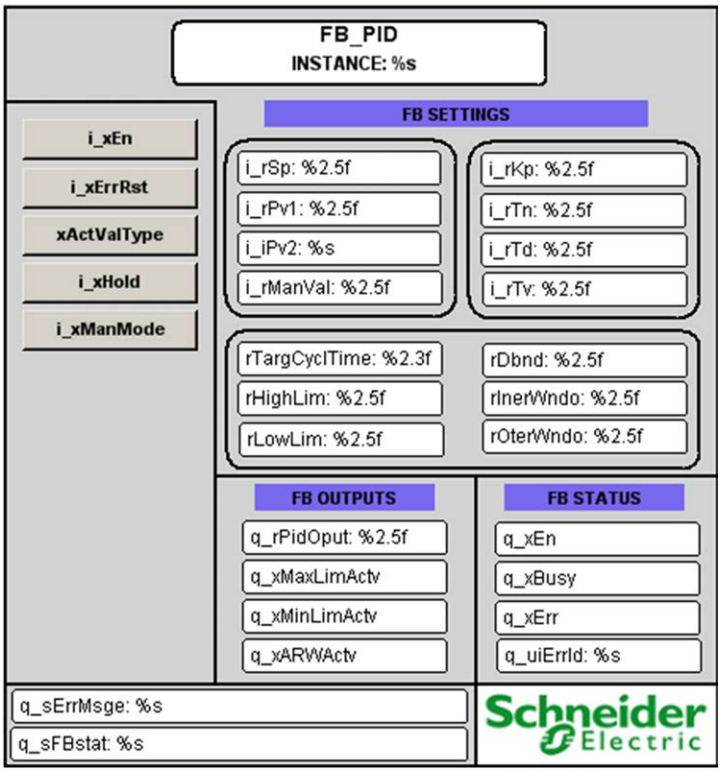

The input `i_tCyclTime` of the first order filters System_1 and System_2 (dataCyclTime) must have exactly the same value as the period of the POU in the MAST, here 50 milliseconds.



When the GEN MODE is changed from RECTANGLE to SINUS with the same other parameters, the Sinus answer is:




This figure shows the visualization of the `FB_PID` function block:




## Detected Error State

This table describes some general detected errors:

| Issue | Cause | Solution |
|---|---|---|
| Detected error state | Invalid input parameter | Enter valid parameter, then reset detected error |

**NOTE:** If the function block is disabled, outputs are set to zero.

# FB_PI_PID: Cascaded PI_PID Control Loop

## What's in This Chapter



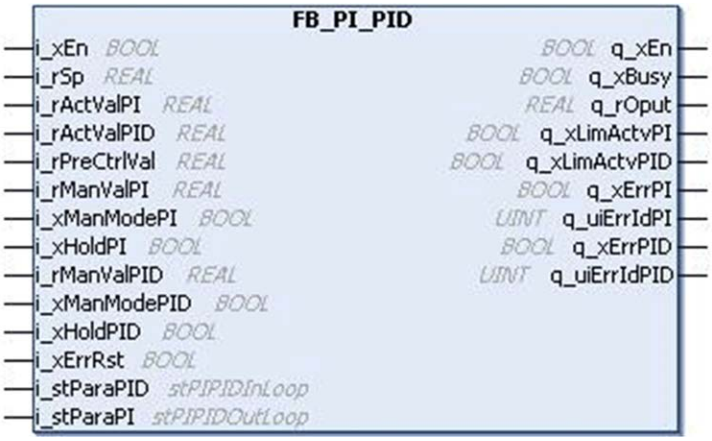
## Overview

This chapter describes the FB_PI_PID function block.

# FB_PI_PID Function Block

## Pin Diagram

This figure shows the pin diagram of the FB_PI_PID function block:




## Functional Description

The FB_PI_PID function block provides a cascaded operation of FB_PI, FB_Limiter and FB_PID.

This function block consists of a PI, a control limiter, and a PID element.

# Operation Modes

## Automatic Mode

The function block calculates the PI response for set point i_rSp and outer loop actual value i_rActValPI. This PI response added to pre control value i_rPreCtrlVal and limited by maximum and minimum threshold inputs, serves as set point to inner PID loop.

The inner loop calculates a PID response with i_rActValPID as the inner loop actual value.

This equation shows the transfer function of PI element:

$$G(s) = Kp(1 + \frac{1}{sT_n})$$

Where:

| | |
|---|---|
| $K_p$ | = Proportional gain |
| $T_n$ | = Integral time |

This equation shows the transfer function of PID element:

$$G(s) = Kp(1 + \frac{1}{sT_n} + \frac{sT_v}{1 + sT_d})$$

Where

| | |
|---|---|
| $K_p$ | = Proportional gain |
| $T_n$ | = Integral time |

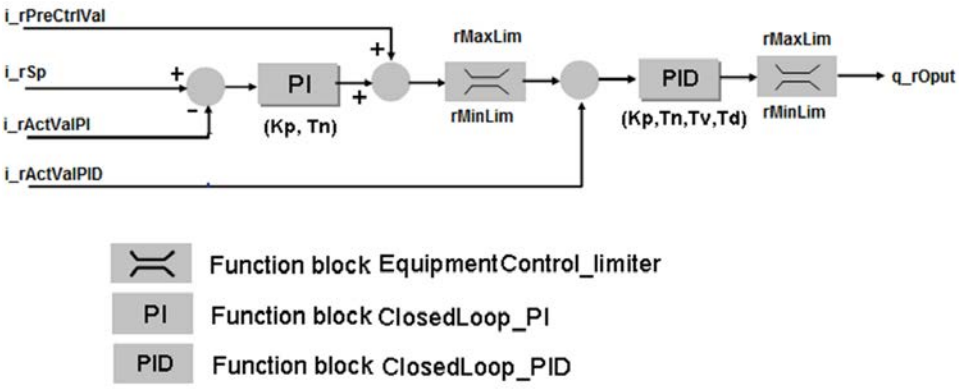
Automatic mode:

| | |
|---|---|
| $T_d$ | = Derivative time |
| $T_v$ | = Filter time |

## Manual Mode

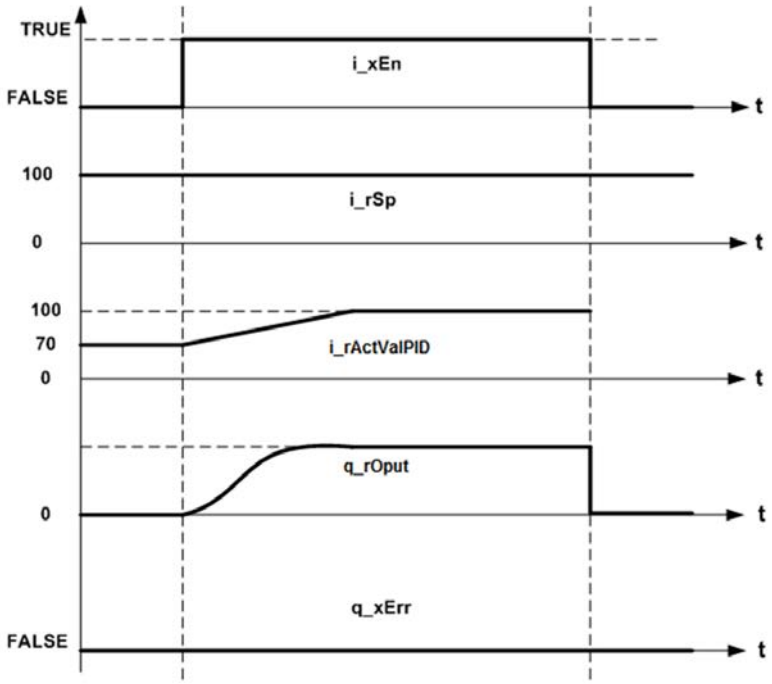
The PI loop and the PID loop can be setup individually in manual modes using input pins `i_xManModePI` and `i_xManModePID` respectively. In manual mode, PI loop output and PID loop outputs are substituted by values at input pins `i_rManValPI` and `i_rManValPID` respectively.

This figure shows the transfer function for `FB_PI_PID` function block



# Timing Diagram

This figure shows the timing diagram for the FB_PI_PID function block




# Detected Error State

An invalid parameter at the function block inputs results in a detected error and a corresponding detected error ID is generated.

During the error detected state the output values are set to zero.

Detected error can be reset only through rising edge of i_xErrRst input.

The output q_xBusy is TRUE whenever the function block is enabled and when there is no detected error.

# Input Pin Description

## Input Pin Table

This table describes the input pins of the FB_PI_PID function block:

| Input | Data Type | Description |
|---|---|---|
| i_xEn | BOOL | TRUE: Enables the function block<br>FALSE: Disables the function block |
| i_rSp | REAL | Set point value of the process<br>Range: ±3.4e$^{+38}$ |
| i_rActValPI | REAL | Actual value of the process to PI outer loop<br>Range: ±3.4e$^{+38}$ |
| i_rActValPID | REAL | Actual value of the process to PID inner loop<br>Range: ±3.4e$^{+38}$ |

| Input | Data Type | Description |
|---|---|---|
| i_rPreCtrlVal | REAL | Pre control value added to the output from PI outer loop<br>Range: ±3.4e+38 |
| i_rManValPI | REAL | Manual input for PI outer loop<br>Range: ±3.4e+38<br>(Optional) |
| i_xManModePI | BOOL | TRUE: Operate PI outer loop in manual mode.<br>FALSE: Operate PI outer loop in auto mode<br>(Optional) |
| i_xHoldPI | BOOL | TRUE: Hold the PI outer loop output and internal state constant<br>FALSE: Disabled<br>(Optional) |
| i_rManValPID | REAL | Manual input for PID inner loop<br>Range: ±3.4e+38<br>(Optional) |
| i_xManModePID | BOOL | TRUE: Operate PID inner loop in manual mode.<br>FALSE: Operate PID inner loop in auto mode<br>(Optional) |
| i_xHoldPID | BOOL | TRUE: Hold the PID inner loop output and internal state constant<br>FALSE: Disabled<br>(Optional) |
| i_xErrRst | BOOL | Reset for detected error<br>(Rising edge resets detected error.)<br>(Optional) |
| i_stParaPID | STRUCT stPIPIDOutLoop | Control parameters for PID inner loop |
| i_stParaPI | STRUCT stPIPIDInLoop | Reset for detected error |

# Structures Used

## *stPIPIDOutLoop*

| Structure Element | Type | Description |
|---|---|---|
| *rKp* | REAL | Proportional gain<br>Range: 0.0...1e$^{38}$ |
| *tTn* | TIME | Integral time<br>Range: 0...60000 ms |
| *tTv* | TIME | Derivative time<br>Range: 0...60000 ms |
| *tTd* | TIME | Filter time<br>Range: 0...60000 ms |
| *rMaxLim* | REAL | Maximum output limit for PID inner loop<br>Range: ±3.4e$^{+38}$ |
| *rMinLim* | REAL | Maximum output limit for PID inner loop<br>Range: ±3.4e$^{+38}$ |
| *rInerWndo* | REAL | Inner window for reduced I-part.<br>Range: ±3.4e$^{+38}$ |
| *rOterWndo* | REAL | Outer window for disabling I-part.<br>Range: ±3.4e$^{+38}$ |

## *stPIPIDInLoop*

| Structure Element | Type | Description |
|---|---|---|
| *tCyclTime* | TIME | Task cycle time.<br>Range: 1...60000 ms |
| *rKp* | REAL | Proportional gain<br>Range: 0...1e$^{38}$ |
| *tTn* | TIME | Integral time<br>Range: 0...60000 ms |
| *rMaxLim* | REAL | Maximum output limit for PI outer loop<br>Range: ±3.4e$^{+38}$ |
| *rMinLim* | REAL | Minimum output limit for PI outer loop<br>Range: ±3.4e$^{+38}$ |

# Output Pin Description

## Output Pin Table

This table describes the output pins of the `FB_PI_PID` function block.

| Output | Data Type | Description |
|---|---|---|
| q_xEn | BOOL | TRUE: Function block is enabled<br>FALSE: Disabled |
| q_xBusy | BOOL | TRUE: Function block is active and no error is detected.<br>FALSE: function block disabled or detected error |
| q_rOput | REAL | Output from the cascaded PI PID Loop.<br>Range: ±3.4e+38 |
| q_xLimActvPI | BOOL | TRUE: Output of the PI outer loop is being limited<br>FALSE: Output from PI outer loop not being limited |
| q_xLimActvPID | BOOL | TRUE: Output of the PID inner loop is being limited<br>FALSE: Output from PID inner loop not being limited |
| q_xErrPI | BOOL | Detected error in PI loop |
| q_uiErrIdPI | UNIT | Displays the detected error Id for the PI loop when q_xErrPI becomes TRUE<br>Range: 0...4 |
| q_xErrPID | BOOL | Detected error in PID loop |
| q_uiErrIdPID | UNIT | Displays the detected error Id for the PID loop when q_xErrPID becomes TRUE<br>Range: 0...4 |

## q_uiErrIdPI

This unique integer value indicates some particular detected error:

| Detected Error ID | Description |
|---|---|
| 0 | No error is detected |
| 1 | i_stParaPI.tCyclTime out of range |
| 2 | i_stParaPI.rMaxLim < i_stParaPI.rMinLim |
| 3 | i_stParaPI.rKp less than zero |
| 4 | i_stParaPI.tTn out of range |

## q_uiErrIdPID

This unique integer value indicates some particular detected error:

| Detected Error ID | Description |
|---|---|
| 0 | No error is detected |
| 1 | i_stParaPID.tTv out of range or i_stParaPID.tTn out of range or i_stParaPID.tTd out of range or i_stParaPID.rKp less than zero. |
| 2 | i_stParaPID.tTd < (i_stParaPI.tCyclTime / 2) |
| 3 | i_stParaPID.rMaxLim < i_stParaPID.rMinLim |
| 4 | i_stParaPID.rOterWndo<i_stParaPID.rInerWndo or<br>i_stParaPID.rOterWndo<0 or<br>i_stParaPID.rInerWndo<0 |

# Equipment Control Functions

## What’s in This Part



## Overview

This part describes the functionality and implementation of equipment control function blocks.

# FB_Cyclic_Monitoring: Cyclic Monitoring

## What's in This Chapter



## Overview

This chapter explains the FB_Cyclic_Monitoring function block.

# FB_Cyclic_Monitoring Function Block

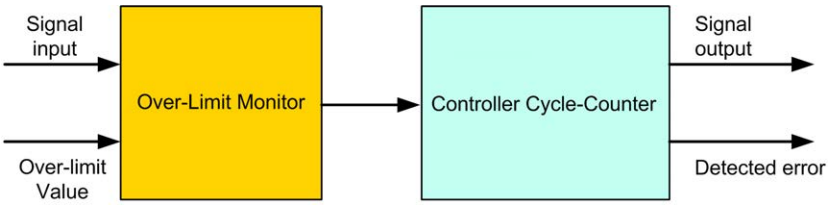
## Pin Diagram

This figure shows the pin diagram of the FB_Cyclic_Monitoring function block:


## Functional Description

The FB_Cyclic_Monitoring function block monitors an input signal for a maximum value (percentage of the absolute maximum value), over a pre-defined number of controller cycles before an inoperable over-limit is detected.

This function block is used to monitor a real input signal and to transfer the input signal to the output only if the input is within limits. It causes the input value to remain above a predefined limiting value for more than a predefined number of consecutive cycles.

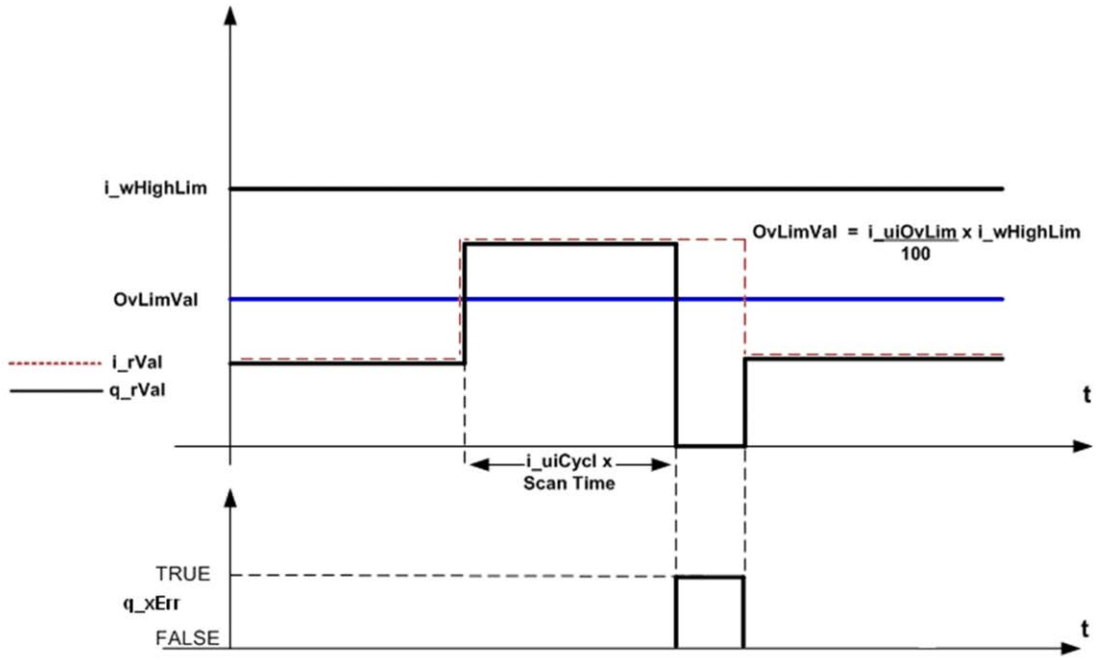



In normal operation, the input signal is transferred to the output based on following conditions

- If the input signal is less than or equal to over-limit (%) of high limit.
- If the input signal is exceeding the high limit for "n" number of consecutive cycle which is less than cycle input.

## Timing Diagram

This figure shows the timing diagram for the FB_Cyclic_Monitoring function block:




## Detected Error State

If the input signal is exceeding over-limit (%) of high limit for n number of cycle which is greater or equal to cycle input, then output is set to zero and detected error output is set to TRUE. The detected error is reset automatically if the input signal is within the limit.

# Input Pin Description

## Input Pin Description

This table describes the input pins of the `FB_Cyclic_Monitoring` function block.

| Input | Data Type | Description |
|---|---|---|
| `i_wHighLim` | `WORD` | High limit of signal<br>Range: 0...65535 |
| `i_uiOvlim` | `UINT` | % of `i_wHighLim` which defines the over-limit range.<br>Range: 0...100<br>Input value exceeding 100% is limited to 100%. |
| `i_rVal` | `REAL` | The input signal to be monitored<br>Range: $\pm 3.4e^{+38}$ |
| `i_uiCycl` | `UINT` | The number of controller cycles the FB allows input to output when input exceeds the over limit range.<br>Range: 0...65535 |

# Output Pin Description

## Output Pin Description

This table describes the output pins of the `FB_Cyclic_Monitoring` function block

| Output | Data Type | Description |
|---|---|---|
| `q_rVal` | `REAL` | Monitored output value.<br>Range: ±3.4e+38 |
| `q_xErr` | `BOOL` | Detected error bit |
| `q_rPerc` | `REAL` | Monitored input value (`i_rVal`) in % of `i_wHighLim`.<br>Range: ±3.4e+38 |

# Instantiation and Usage Example

## Instantiation and Usage Example

This figure shows an instance of the the `FB_Cyclic_Monitoring` function block:


## Example

This table shows and example for the `FB_CyclicMonitoring` function block operation:

| Example | Inputs | Outputs | Remarks |
|---|---|---|---|
| 1 | i_wHighLim = 200,<br>i_uiOvLim = 50,<br>i_rVal = 40,<br>i_uiCycle = 10 | q_rVal = 40,<br>q_rPerc = 20,<br>q_xErr = FALSE | Input value (i_rVal) is less than or equal to calculated over limit value (That is [(i_uiOvLim/100) x i_wHighLim]).<br>Output: Input i_rVal is assigned to output q_rVal & q_xErr output is FALSE. |
| 2 | i_wHighLim = 200,<br>i_uiOvLim = 50,<br>i_rVal = 110,<br>i_uiCycle = 10 | q_rVal = 110<br>q_rPerc = 55<br>q_xErr = FALSE | Input value (i_rVal) is greater than calculated over limit value (That is [(i_uiOvLim/100) x i_wHighLim]) and, **completed scan cycles are less than set no of cycles**.<br>Output: Input i_rVal is assigned to output q_rVal & q_xErr output is FALSE. |
| 3 | i_wHighLim = 200,<br>i_uiOvLim = 50,<br>i_rVal = 110,<br>i_uiCycle = 10 | q_rVal = 0<br>q_rPerc = 55<br>q_xErr = TRUE | Input Value (i_rVal) is greater than calculated over limit value (That is [(i_uiOvLim/100) x i_wHighLim)] and **completed scan cycles are equal to or greater than set no of cycles**.<br>Output: Output q_rVal is equal to zero and q_xErr output is TRUE. |

# Detected Error state

This table shows some general issues and their solution:

| Issue | Cause | Solution |
|---|---|---|
| Detected error state | Input value(i_rVal) is not within the over limit (%) of i_wHighLim for n number of cycles | Input value (i_rVal) less than Over Limit value automatically resets detected error. |

# FB_DeadBand: Suppressing Amplitude Oscillations

## What's in This Chapter



## Overview

This chapter describes the FB_DeadBand function block.

# FB_DeadBand Function Block

## Pin Diagram

This figure shows the pin diagram of the FB_DeadBand function block:


## Functional Description

The FB_DeadBand function block is a Deadband function block which allows the input to pass on to the output only if the input is greater than the Deadband limit.

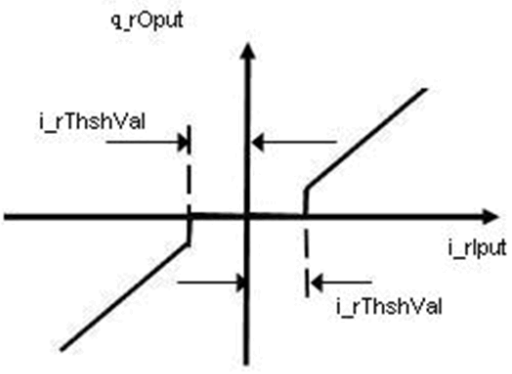
This function block suppresses small amplitude oscillations that are caused by noise, quantization or parameter calculation. It suppresses an input signal if it is within the threshold as shown in transfer function figure below.

With reference to the timing diagram:

- If the i_rIput is lower than the defined threshold range, q_rOput is set to zero and q_xSigUndThsh is TRUE.
- If the input value (i_rIput) greater or equal to the threshold range, q_rOput is equal to the i_rIput value.

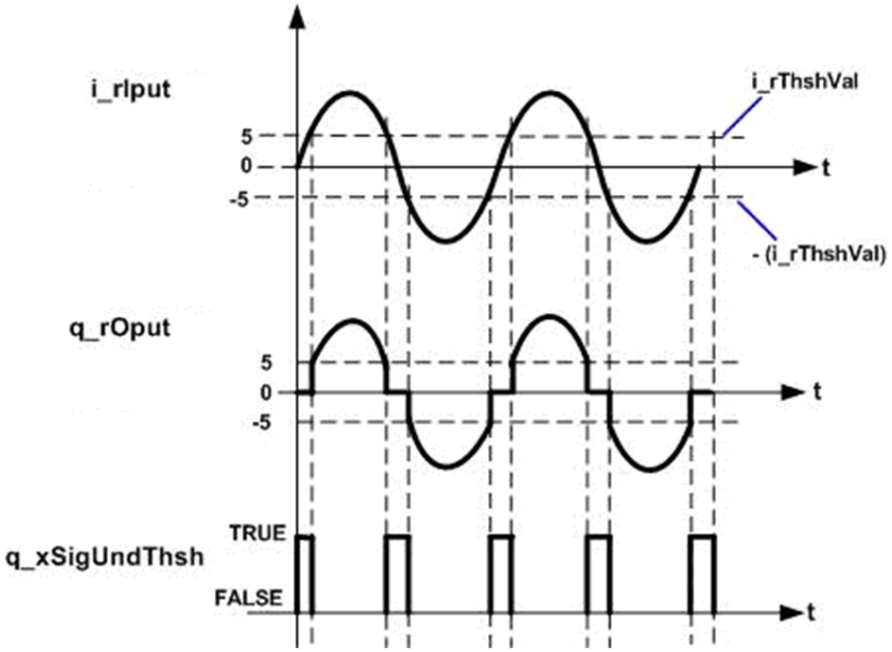
The q_xEn is TRUE as long as i_xEn is TRUE regardless of the detected error.

This figure shows the transfer function for FB_DeadBand function block:



# Timing Diagram

This figure shows the timing diagram for `FB_DeadBand` function block:




# Detected Error State

An invalid parameter at the function block inputs results in a detected error and corresponding detected error ID is generated. During the error detected state, the output value is set to zero.

The detected error can be reset only through rising edge of `i_xErrRst` input.

The `q_xBusy` is TRUE, whenever the function block is enabled and when there is no detected error.

# Input Pin Description

## Input Pin Description

This table describes the input pins of the `FB_DeadBand` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xEn` | `BOOL` | TRUE: Enables the function block.<br>FALSE: Disables the function block. |
| `i_rIput` | `REAL` | Input value<br>Range: ±3.4e$^{+38}$ |
| `i_rThshVal` | `REAL` | Threshold value<br>Range: 0.0...3.4e$^{+38}$ |
| `i_xErrRst` | `BOOL` | Reset for detected error (on rising edge)<br>(Optional) |

# Output Pin Description

## Output Pin Description

This table describes the output pins of the `FB_DeadBand` function block.

| Output | Data Type | Description |
|---|---|---|
| `q_xEn` | `BOOL` | TRUE if function block is enabled. |
| `q_xBusy` | `BOOL` | TRUE if function block is enabled and no detected error. |
| `q_rOput` | `REAL` | Output of the given input.<br>Range: $\pm 3.4e^{+38}$ |
| `q_xSigUndThsh` | `BOOL` | TRUE if input is under threshold limit. |
| `q_xErr` | `BOOL` | Detected error |
| `q_uiErrId` | `UINT` | 0 = No detected error<br>1 = Invalid parameter `i_rThsh` < 0<br>Range: 0...1 |

# FB_Limiter: Limiting Input Signals

## What's in This Chapter



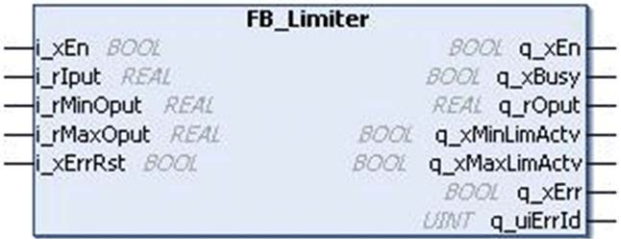
## Overview

This chapter describes the FB_Limiter function block.

# FB_Limiter Function Block

## Pin Diagram

This figure shows the pin diagram of the FB_Limiter function block:




## Functional Description

The FB_Limiter function block is a limiter function block to limit an input signal within a defined range.

The input signal is limited to a defined range based on the i_rMaxOput and i_rMinOput as shown in the transfer function figure below.

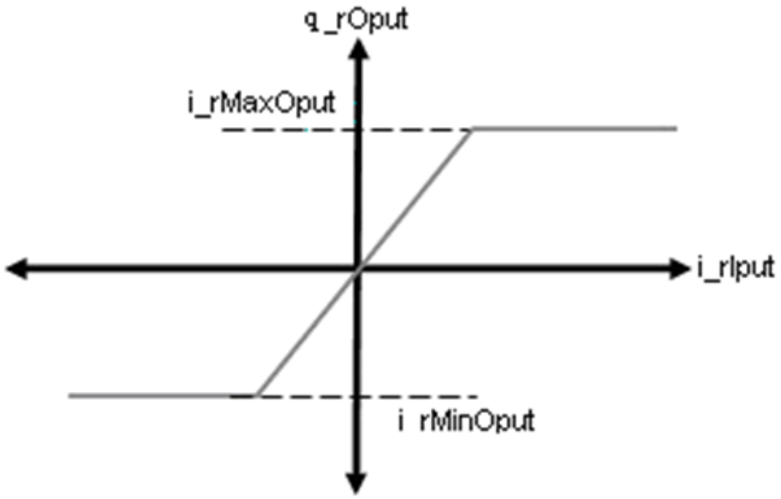
If the input exceeds the upper or lower limit, the output is limited to maximum or minimum values respectively.

With reference to the timing diagram below:

- If the input is within the defined range, the output is equal to input value.
- If input value exceeds the maximum limit, the output is limited to maximum output value.
- Similarly, if the input goes below the minimum output value, the output is limited to the minimum output value.
- If the function block limits the output, then q_xMinLimActv or q_xMaxLimActv is TRUE, based on the type of limit.

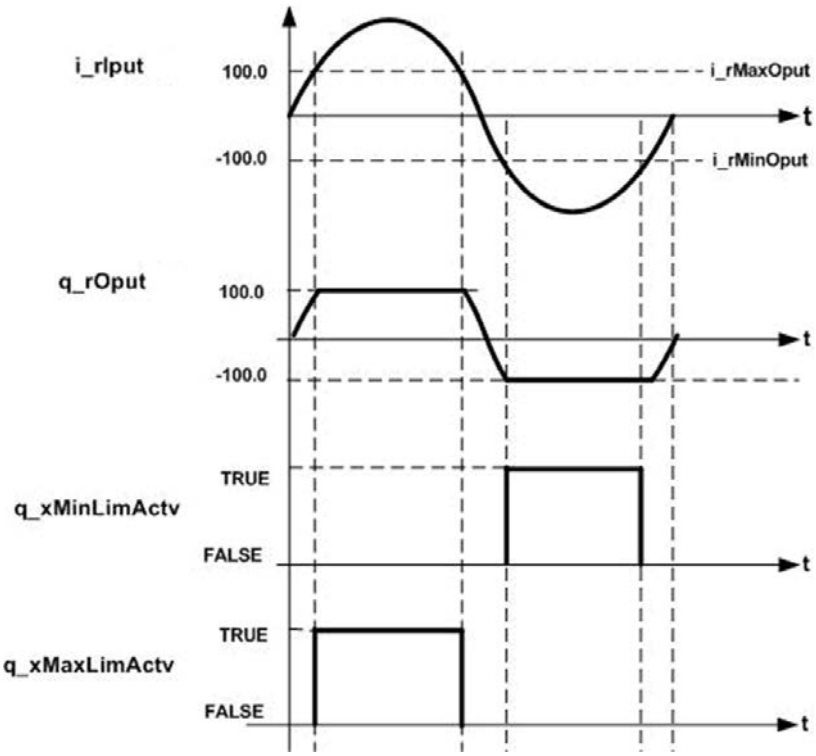
The q_xEn is TRUE as long as i_xEn is TRUE, regardless of detected error.

This figure shows the transfer function of the `FB_Limiter` function block:




## Timing Diagram

This figure shows the timing diagram of the `FB_Limiter` function block:




## Detected Error State

An invalid parameter at the function block inputs results in detected error, and a corresponding detected error ID will be generated.

During a detected error state, the output will be set to zero.

Detected error can be reset only through the rising edge of `i_xErrRst` input. The output `q_xBusy` is TRUE whenever the function block is enabled and there is no detected error.

# Input Pin Description

## Input Pin Description

This table describes the input pins of the `FB_Limiter` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xEn` | `BOOL` | TRUE: Enables the function block.<br>FALSE: Disables the function block. |
| `i_rIput` | `REAL` | Input value which has to be limited.<br>Range: ±3.4e+38 |
| `i_rMinOput` | `REAL` | Minimum output value.<br>Range: ±3.4e+38 |
| `i_rMaxOput` | `REAL` | Maximum output value.<br>Range: ±3.4e+38 |
| `i_xErrRst` | `BOOL` | Reset for detected error (on rising edge)<br>(Optional) |

# Output Pin Description

## Output Pin Description

This table decscribes the input pins of the `FB_Limiter` function block:

| Output | Data Type | Description |
|---|---|---|
| `q_xEn` | `BOOL` | TRUE if function block is enabled. |
| `q_xBusy` | `BOOL` | TRUE if function block is enabled and no detected error. |
| `q_rOput` | `REAL` | Output of the given input.<br>Range: ±3.4e+38 |
| `q_xMinLimActv` | `BOOL` | TRUE if input is equal or under the minimum output value. |
| `q_xMaxLimActv` | `BOOL` | TRUE if input is equal or above maximum output value. |
| `q_xErr` | `BOOL` | Detected error |
| `q_uiErrId` | `UINT` | 0 = No detected error<br>1 = Invalid parameter `i_rMaxOput` < `i_rMinOput`<br>Range: 0 ... 1 |

# FB_PWM: Providing a PWM Output

## What's in This Chapter



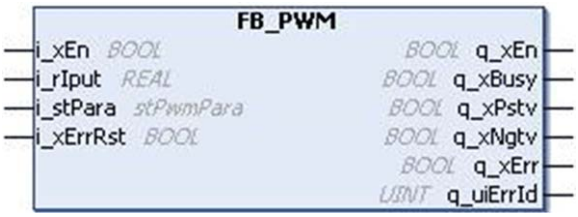
## Overview

This chapter describes the FB_PWM function block.

# FB_PWM Function Block

## Pin Diagram

This figure shows the pin diagram of the FB_PWM function block:




## Functional Description

The FB_PWM function block is developed to provide a PWM output based on the input parameter.

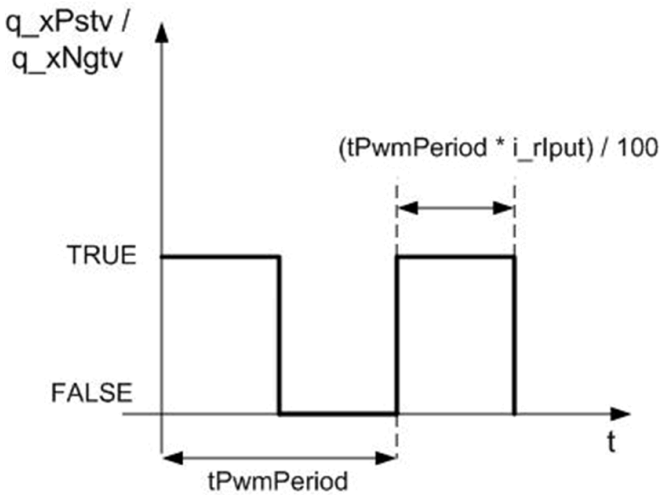
The PWM output is generated with defined ON time and OFF time as per the input shown in the first timing diagram below.

With reference to the second timing diagram:

- If i_rIput is a positive value, then the PWM output is available in the q_xPstv. The input i_rIput should be in a range of -100 to 100. The ON time of PWM is determined as given below: PWM ON time = (i_rIput x tPwmPeriod) / 100.
- If i_rIput is a negative value, then the PWM output is available in q_xNgtv.
- If i_rIput is greater than 100, then it is limited to 100 and if i_rIput is less than –100, then it is limited to –100.
- If i_xPwmInstUpdt is TRUE, the change in input parameter is updated in the current PWM cycle itself as shown in the timing diagram.
- If i_xPwmInstUpdt is FALSE, the change in input is updated only during a start of a new PWM cycle.

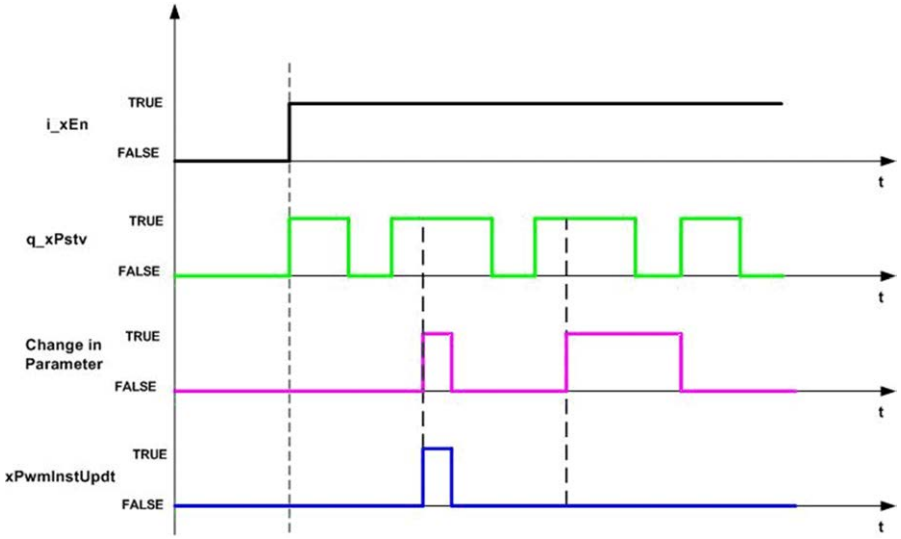
The q_xEn is TRUE as long as the input i_xEn is TRUE, regardless of detected error.

This figure shows the timing diagram for `FB_PWM` calculation:




## Timing Diagram

This figure shows the timing diagram for `FB_PWM` function block:

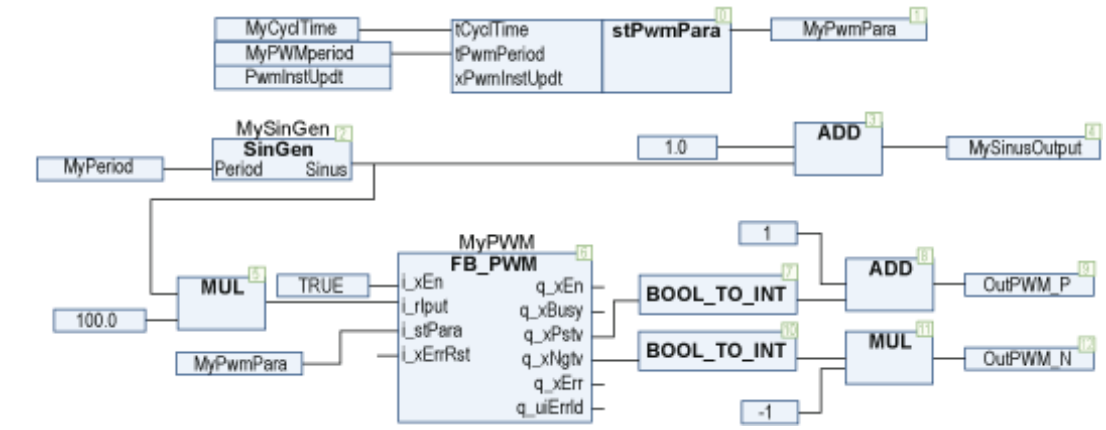


# Example with a Frequency Signal

The program creates a Sinus signal at a certain period (5 seconds/0.2 Hz). This Sinus signal is the input of the `FB_PWM`.

MyPou_PWM [MyController: PLC Logic: Application] | Trace_PWM [MyController: PLC Logic: Application]

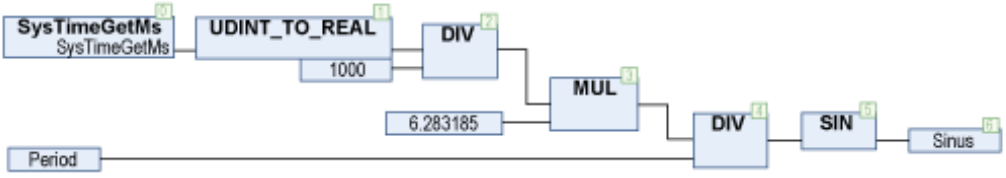
```
PROGRAM MyPou_PWM
VAR
    FlagDemarrer        : BOOL;
    MyPWM               : FB_PWM;
    MyPwmPara           : stPwmPara;
    MySinGen            : SinGen;
    MyCyclTime          : TIME := T#10MS;
    MyPWMperiod         : TIME := T#200MS;
    PwmInstUpdt         : BOOL;
    MyPeriod            : REAL := 5.0;
    MySinusOutput       : REAL;
    OutPWM_P            : INT;
    OutPWM_N            : INT;
END_VAR
```



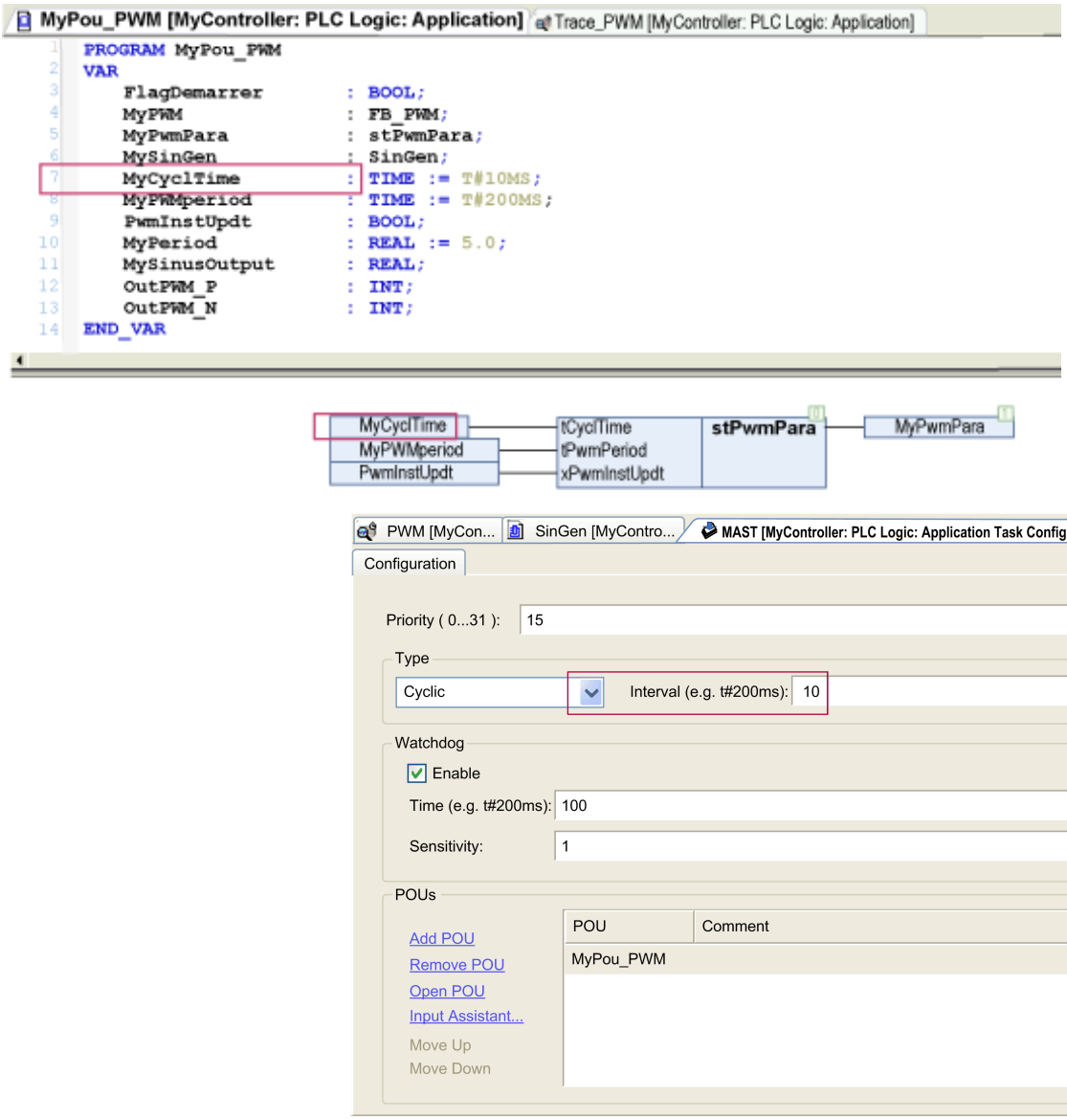

Definition of the `SinGen` function block:

MyPou_PWM [MyController: PLC Logic: Application] | Trace_PWM [MyController: PLC Logic: Application] | SinGen [MyContro...

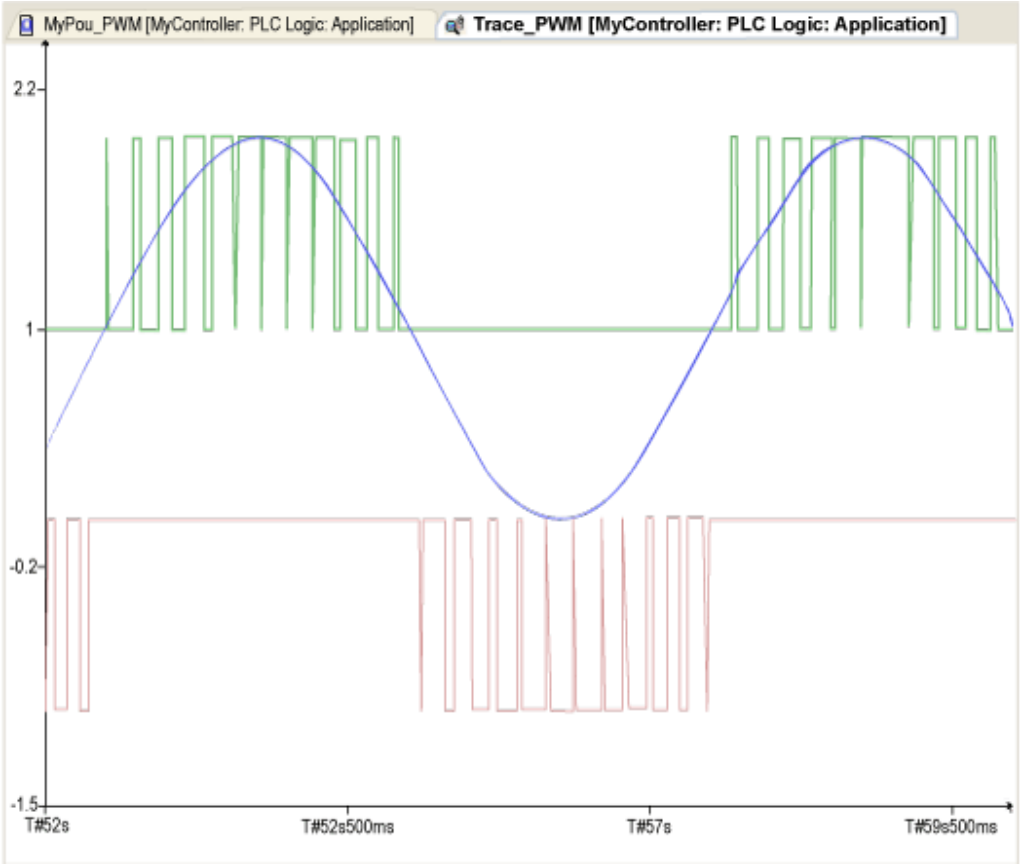
```
FUNCTION_BLOCK SinGen
VAR_INPUT
    Period : REAL;
END_VAR
VAR_OUTPUT
    Sinus : REAL;
END_VAR
VAR
END_VAR
```



The input `stPwmPara.tCycTime` of the `FB_PWM` function block must have exactly the same value as the period of the POU in the MAST, here 10 milliseconds (see the red bordered area).



The result of the previous POU:




**Blue** `i_rIput` sinus signal at 0.2 Hz (function block `My_Filter_PT1_1`).

**Green** `q_xPstv` (an offset is added for the trace).

**Red** `q_xNgtv` (the signal is inversed for the trace).

## Detected Error State

An invalid parameter at the function block inputs results in detected error and corresponding detected error ID is generated.

During detected error state, the output is set to zero.

The detected error can be reset only through rising edge of `i_xErrRst` input. The output `q_xBusy` is TRUE whenever the function block is enabled and there is no detected error.

# Input Pin Description

## Input Pin Description

This table shows the input pins of the `FB_PWM` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xEn` | `BOOL` | TRUE: Enables the function block.<br>FALSE: Disables the function block. |
| `i_rIput` | `REAL` | PWM Input value<br>Range: -100...100 |

| Input | Data Type | Description |
|---|---|---|
| i_stPara | STRUCT stPwmPara | Structure parameter |
| i_xErrRst | BOOL | Reset for detected error (on rising edge)<br>(Optional) |

# Structure Used

## *stPwmPara*

| Structure Element | Type | Description |
|---|---|---|
| *tCyclTime* | TIME | Task cycle time<br>Range: 1...1e32 ms |
| *tPwmPeriod* | TIME | PWM time period<br>Range: 1...1e32 ms |
| *xPwmInstUpdt* | BOOL | TRUE: a new input value is immediately adopted, even in the present PWM cycle.<br>(Optional) |

# Output Pin Description

## Output Pin Description

This table describes the input pins of the FB_PWM function block:

| Output | Data Type | Description |
|---|---|---|
| q_xEn | BOOL | TRUE if function block is enabled. |
| q_xBusy | BOOL | TRUE if function block is enabled and no detected error. |
| q_xPstv | BOOL | PWM output is positive if PWM input >0. |
| q_xNgtv | BOOL | PWM output is negative if PWM Input <0. |
| q_xErr | BOOL | Detected error |
| q_uiErrId | UINT | 0 = No detected error<br>1 = Invalid parameter i_tCyclTime = 0<br>2 = Invalid parameter i_tPwmPeriod <= i_tCyclTime<br>Range: 0...2 |

# FB_Redundant_Sensor_Monitoring: Redundant Sensor Monitoring

## What's in This Chapter



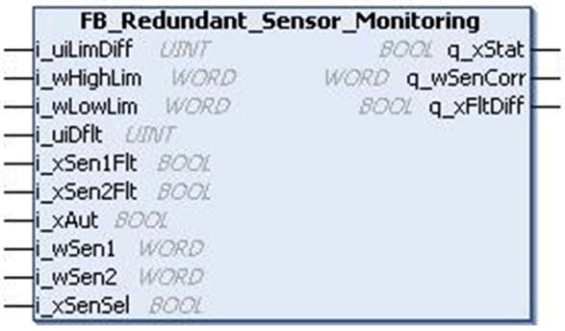
## Overview

This chapter describes the FB_Redundant_Sensor_Monitoring function block.

# FB_Redundant_Sensor_Monitoring Function Block

## Pin Diagram

This figure shows the pin diagram of the FB_Redundant_Sensor_Monitoring function block:



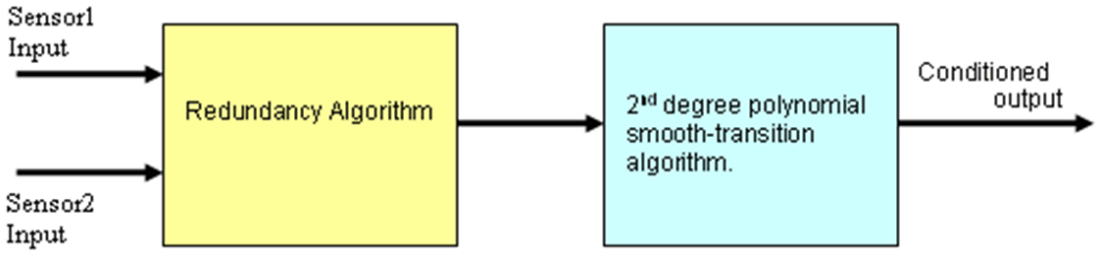

## Functional Description

The FB_Redundant_Sensor_Monitoring function block monitors signals coming in from two redundant analog signal sources or field sensors which have the same range and characteristics.

The function block does the following functions:

- Manipulates the output as per sensor(s) readings (average value).
- Monitors if the 2 sensor readings are within the specified difference limit, otherwise takes corrective action.
- Performs predefined action in case of any inoperable sensor is reported.
- Automatic selection of healthy sensor for output in case of other inoperable sensor.
- Provision to select any sensor manually.
- A second-degree polynomial based 'smooth-transition algorithm' eliminates any sudden step variation in the output of the block.
- The output for a detected difference between the two sensors is set to TRUE when the difference between the two sensor values is not within the limits. The difference between sensor input values is reset immediately after the difference between the sensor inputs are within the limits.

This figure shows the block diagram of the `FB_Redundant_Sensor_Monitoring` function block:




This table contains the second degree polynomial smooth-transition algorithm conditions:

| Example | Condition | FB Output |
|---|---|---|
| 1 | Absolute difference between calculated output and previous FB output is greater than or equal to 15, calculated output is greater than previous FB output. | Previous FB output - ((0.07 * (Calculated output - Previous FB output)) - 23) |
| 2 | Absolute difference between calculated output and previous FB output is greater than or equal to 15, calculated output is less than or equal to previous FB output. | Previous FB output - ((0.07 * (Calculated output - Previous FB output)) + 23) |
| 3 | Absolute difference between calculated output and previous FB output is less than 15. | Calculated output |

# Input Pin Description

## Input Pin Description

This table describes the input pins of the `FB_Redundant_Sensor_Monitoring` function block:

| Output | Data Type | Description |
|---|---|---|
| `i_uiLimDiff` | `UINT` | The difference between sensor-1 and sensor-2 values in % of the sensor range within which the redundant sensors shall work.<br>Range: 0...100<br>Value exceeding 100% is limited to 100%.<br>0%: There should not be any difference between sensors.<br>100%: Accept any value that comes in or always average Sen1_ip and Sen2_ip. |
| `i_wHighLim` | `WORD` | High limit of sensor inputs<br>Range: 0...65535 |
| `i_wLowLim` | `WORD` | Low limit of sensor inputs<br>Range: 0...65535 |
| `i_uiDflt` | `UINT` | Default value in terms of % of the sensor input range<br>Range: 0...100 |
| `i_xSen1Flt` | `BOOL` | TRUE: Sensor is in detected error<br>FALSE: Sensor is healthy.<br>(Optional) |
| `i_xSen2Flt` | `BOOL` | TRUE: Sensor is in detected error. |

| Output | Data Type | Description |
|---|---|---|
| | | FALSE: Sensor is healthy.<br>(Optional) |
| i_xAut | BOOL | TRUE: Auto mode<br>FALSE: Manual mode. |
| i_wSen1 | WORD | Sensor-1 raw value<br>Range: 0...65535<br>(It is expected that sensor-1 and 2 shall be of same range and characteristics.) |
| i_wSen2 | WORD | Sensor-2 raw value<br>Range: 0...65535<br>(It is expected that sensor-1 and 2 shall be of same range and characteristics.) |
| i_xSenSel | BOOL | TRUE: Sensor 2 selected<br>FALSE: Sensor 1 selected<br>Applicable only for manual mode. |

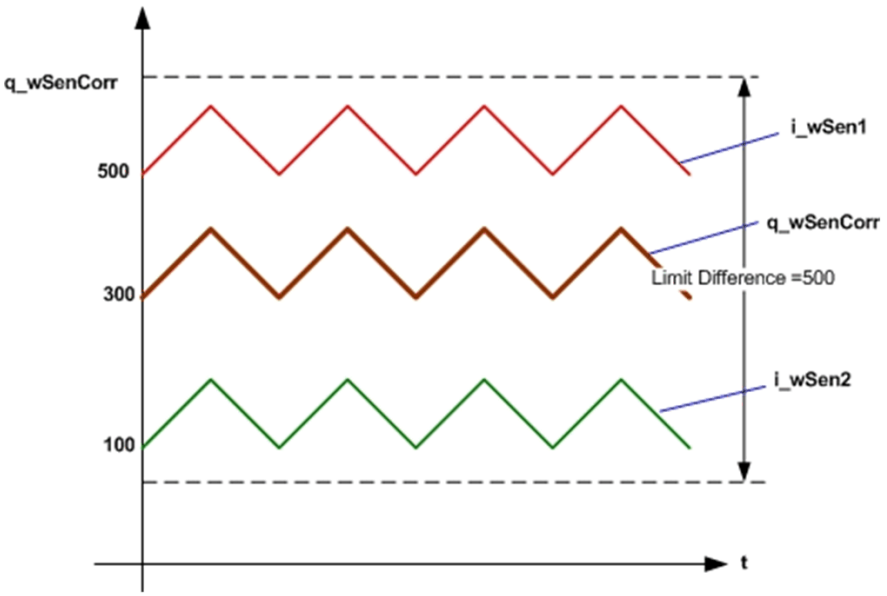
## i_uiLimDiff

This is Limit difference in terms of %. Averaged output (q_wSenCorr) is generated only if the difference between two sensor inputs is less than the Limit Difference.

- Limit difference is calculated based on the below equation

  Limit Difference = (i_uiLimDiff x (i_wHighLim - i_wLowLim))/100
- If the difference between Sensor 1 input and sensor 2 input is less than the Limit Difference, then output is average of two sensors as shown in the following figure.

This figure shows the averaging function in FB_Redundant_Sensor_Monitoring function block:



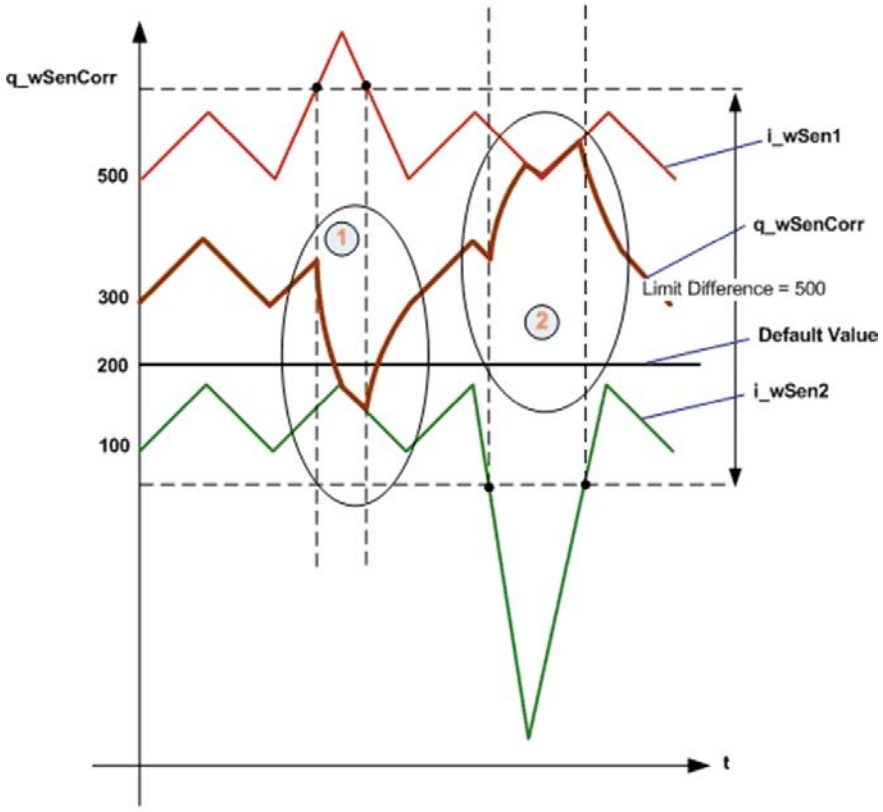
## i_uiDflt

This input is default value in terms of % which is used to generate most appropriate output the difference is not within the limit and both the sensors are healthy.

- Default value is calculated based on the below equation:

  Default value = (`i_uiDflt` x (`i_wHighLim` - `i_wLowLim`))/100
- If the difference between 2 sensor input is out of limit, the function block gives an output of anyone sensor which is closer to the default value as shown in the following figure.

This figure shows the default value function in `FB_Redundant_Sensor_Monitoring` function block:




1. Difference between the sensors is not within the limit & Sensor 2 input is closer to default value, so Output follows the Sensor 2 regardless of sensor1 This transition is gradual due to Smooth transition algorithm.
2. Difference between the sensors is not within the limit & Sensor 1 input is closer to default value, so Output follows the Sensor 1 regardless of sensor 2. This transition is gradual due to Smooth transition algorithm.

## i_xSen1Flt and i_xSen2Flt

These inputs are used to detect whether the two sensors are healthy or not healthy.

- If both the sensors are not healthy, the output (`q_wSenCorr`) is set to zero.
- If sensor 1 is not healthy, then the output is set to sensor 2 input. Similarly if sensor 2 is not healthy, then the output is set to sensor 1 input.

### i_xAut

This input is used to select auto mode or manual mode.

- If this input is TRUE, the function block operates in auto mode. In auto mode, based on sensor inputs and difference between the inputs, function block generates an appropriate output.
- If this input is FALSE, the function block operates in manual mode. In manual mode, based `i_xSenSel` input the output is forced to either sensor 1 or sensor 2.
- If `i_xSenSel` is FALSE, sensor 1 input is selected. Similarly if `i_xSenSel` is TRUE, sensor 2 input is selected.

# Output Pin Description

## Output Pin Description

This table describes the output pins of the `FB_Redundant_Sensor_Monitoring` function block:

| Output | Data Type | Description |
|---|---|---|
| `q_xStat` | `BOOL` | TRUE: Auto mode<br>FALSE: Manual mode |
| `q_wSenCorr` | `WORD` | Redundant sensor manipulated value<br>Range: 0...65535 |
| `q_xFltDiff` | `BOOL` | TRUE: Detected error<br>FALSE: Normal<br>This output is TRUE when difference between the sensors is not within limit for more than 3 consecutive controller cycles in auto mode. |

**NOTE:** The function block output `q_wSenCorr` will not change abruptly based on input. Instead a second-degree polynomial is applied to avoid such sudden step variation in the output of the function block.

# FB_Scaling: Scaling Input Signals

## What's in This Chapter



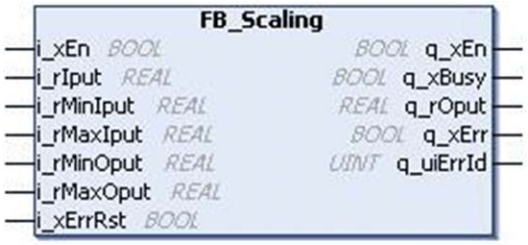
## Overview

This chapter describes the `FB_Scaling` function block.

# *FB_Scaling* Function Block

## Pin Diagram

This figure shows the pin diagram of the *FB_Scaling* function block:



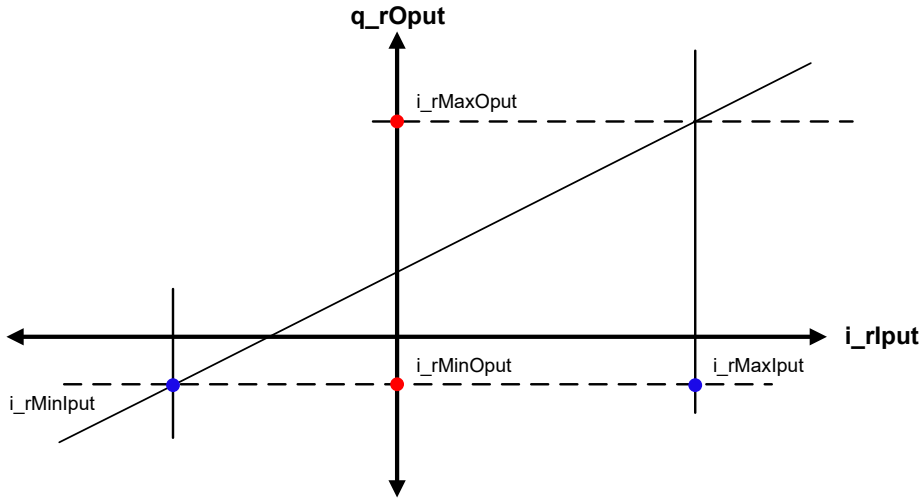
# Functional Description

The *FB_Scaling* function block is developed to convert an input value to a specified output range linearly and an error is detected in case of an invalid parameter.

This function block scales an input signal to a linear output, relative to a defined maximum and minimum range. The minimum and maximum values used for scaling are not limiting the output value.

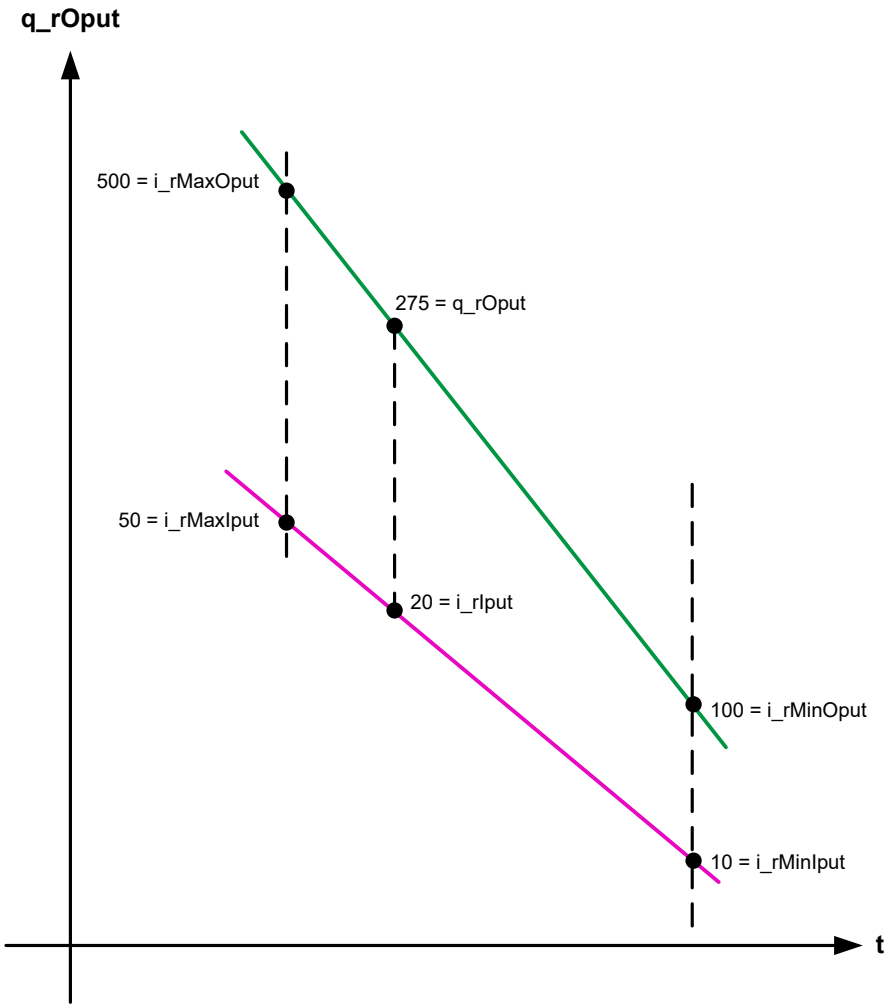
**NOTE:** For a limitation of the scaled value provided by the function block, use *FB_Limiter*.

The input signal is scaled in a linear manner with reference to two value ranges as shown in the figure below:




The output changes dynamically based on the change in input:

- Slope = (*i_rMaxOput*-*i_rMinOput*) / (*i_rMaxIput* - *i_rMinIput*)
- Offset = *i_rOutMax* - (Slope * *i_rMaxIput*)
- *q_rOput* = (Slope * *i_rIput*) + Offset




The *q_xEn* is TRUE as long as the input *i_xEn* is TRUE, regardless of a detected error.

## Detected Error State

An invalid parameter at the function block inputs results in a detected error and a corresponding detected error ID is generated. The output is set to zero during a detected error. The detected error can be reset only through a rising edge of *i_xErrRst*. The input *q_xBusy* is TRUE whenever the function block is enabled and there is no detected error.

# Input Pin Description

## Input Pin Description

This table describes the input pins of the `FB_Scaling` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xEn` | `BOOL` | TRUE: Enables the function block.<br>FALSE: Disables the function block. |
| `i_rIput` | `REAL` | Input value which has to be scaled.<br>Range: $\pm 3.4e^{+38}$ |
| `i_rMinIput` | `REAL` | Minimum input value |

| Input | Data Type | Description |
|---|---|---|
| | | Range: ±3.4e+38 |
| i_rMaxIput | REAL | Maximum input value<br>Range: ±3.4e+38 |
| i_rMinOput | REAL | Minimum output value<br>Range: ±3.4e+38 |
| i_rMaxOput | REAL | Maximum output value<br>Range: ±3.4e+38 |
| i_xErrRst | BOOL | Reset for detected error (on rising edge)<br>(Optional) |

# Output Pin Description

## Output Pin Description

The following table includes the different outputs of the function block along with the description of the identifiers or commands.

| Output | Data Type | Description |
|---|---|---|
| q_xEn | BOOL | TRUE if function block is enabled. |
| q_xBusy | BOOL | TRUE if function block is enabled and no detected error. |
| q_rOput | REAL | Scaled output of the given input.<br>Range: ±3.4e+38 |
| q_xErr | BOOL | Detected error |
| q_uiErrId | UINT | 0 = No detected error<br>1 = Invalid parameter i_rMinIput = i_rMaxIput<br>Range: 0...1 |

# FB_Sensor_Monitoring: Sensor Monitoring

## What's in This Chapter



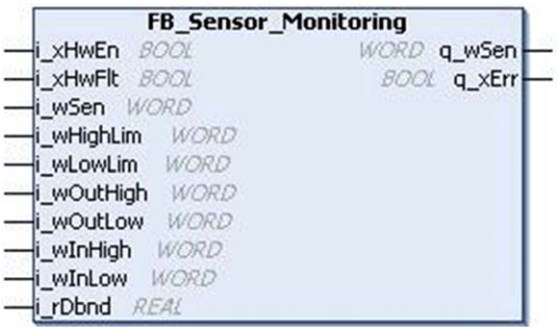
## Overview

This chapter describes the FB_Sensor_Monitoring function block.

# FB_Sensor_Monitoring Function Block

## Pin Diagram

This figure shows the pin diagram of the FB_Sensor_Monitoring function block:



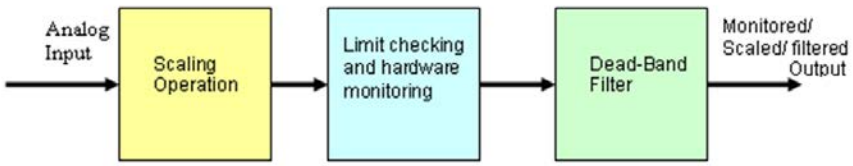

## Functional Description

The FB_Sensor_Monitoring function block monitors and/or scales and/or Deadband filters an input analog signal.

This function block performs the following operations on an analog input signal:

- Monitor if the sensor reading is within the operator specified range, otherwise an error output is detected if not within the range.
- Monitor I/O hardware and generate an alarm if an error is detected.
- Provision to enable/disable the I/O hardware monitoring feature in the block.
- Scale the input value to the desired output range.
- Pass the final output through a dead-band filter. Deadband suppresses the relative oscillation based on the previous input and present input, and then generates an output.

This figure shows the FB_Sensor_Monitoring block diagram:



# Input Pin Description

## Input Pin Description

This table describes the input pins of the `FB_Sensor_Monitoring` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xHwEn` | `BOOL` | TRUE: Hardware monitoring enabled<br>FALSE: Hardware monitoring disabled |
| `i_xHwFlt` | `BOOL` | TRUE: Hardware detected error<br>FALSE: No hardware detected error<br>(Optional) |
| `i_wSen` | `WORD` | Sensor value<br>Range: 0...65535 |
| `i_wHighLim` | `WORD` | High limit of sensor input.<br>Range: 0...65535<br>`i_wHighLim` should be greater than `i_wLowLim` |
| `i_wLowLim` | `WORD` | Low limit of sensor input.<br>Range: 0...65535 |
| `i_wOutHigh` | `WORD` | Output range high limit for scaling function.<br>Range: 0..65535<br>For scaling function, `i_wOutHigh`, `i_wOutLow`, `i_wInHigh` & `i_wInLow` are used |
| `i_wOutLow` | `WORD` | Output range low limit for scaling function.<br>Range: 0...65535 |
| `i_wInHigh` | `WORD` | Input range high limit for scaling function.<br>Range: 0...65535 |
| `i_wInLow` | `WORD` | Input range low limit for scaling function.<br>Range: 0...65535 |
| `i_rDbnd` | `REAL` | Defines the width of the 'dead-band' in the DeadBand filter, in terms of % of range.<br>Range: 0.0...100.0<br>0.0: No deadband filtering.<br>100.0: Block all signals.<br>So ideally the value is less than 100.0<br>(Optional) |

# Output Pin Description

## Output Pin Description

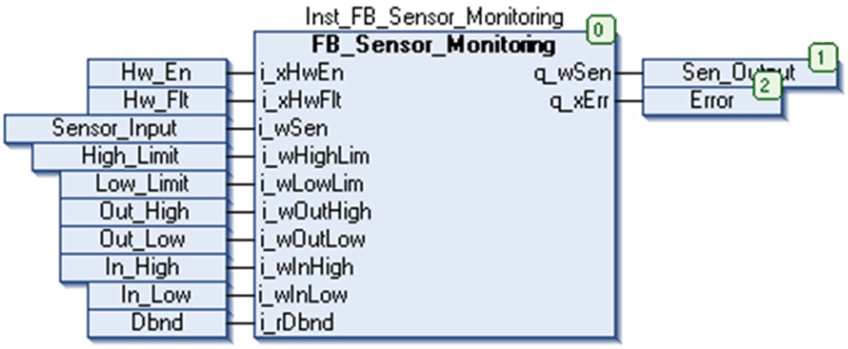
This table describes the output pins of the `FB_Sensor_Monitoring` function block:

| Output | Data Type | Description |
|---|---|---|
| q_wSen | WORD | Valid sensor output<br>Range: 0...65535 |
| q_xErr | BOOL | TRUE: If there is a Hardware detected error or if the scaled output is not within the sensor limits for more than 3 consecutive controller scan cycles.<br>FALSE: No detected error |

# Instantiation and Usage Example

## Instantiation and Usage Example

This figure shows an instance of the FB_Sensor_Monitoring function block pin diagram:




## Example

This example illustrates the functionality of different features in FB_Sensor_Monitoring function block:

| Example | Steps | Inputs | Outputs |
|---|---|---|---|
| 1 | Scaling:<br>Based on the scaling input parameters, sensor input i_wSen is scaled linearly and scaled output value is passed for limit checking. | i_wSen = 1000, i_wOutHigh = 2000, i_wOutLow = 100, i_wInHigh = 1000, i_wInLow = 10. | Internal calculated scaled output = 2000 |
| 2 | Limit checking:<br>• If calculated scaled output is greater than maximum limiting Input, i_wHighLim than output is limited to i_wHighLim<br>• If calculated scaled output is less than minimum limiting input, i_wLowLim than output is limited to i_wLowLim<br>• If calculated scaled output is within the limit, the calculated scaled output is processed further for deadband functions.<br>q_xErr will be TRUE if calculated scaled output is out of range for more than 3 | i_wSen = 1000, i_wHighLim = 20000, i_wLowLim = 4000, i_wOutHigh = 2000, i_wOutLow = 100, i_wInHigh = 1000, i_wInLow = 10, i_rDbnd = 10. | Internal output after limit check = 4000<br>q_xErr = FALSE/ TRUE. |

| Example | Steps | Inputs | Outputs |
|---|---|---|---|
| | consecutive controller scan cycles. | | |
| 3 | Hardware detected error monitoring: `q_xErr` output is TRUE and `q_wSen` holds its last value when hardware detected error monitoring is enabled and hardware (`i_xHwFlt`) detected error input is TRUE. | `i_xHwEn`=1, `i_xHwFlt`=1, `i_wSen`=1000, `i_wOutHigh`=2000, `i_wOutLow`=100, `i_wInHigh`=1000, `i_wInLow`=10, `i_rDbnd`=10. | `q_wSen` = 4000, `q_xErr` = TRUE. |
| 4 | Deadband filtering:<br>• If difference between calculated scaled output and previous FB output is less or equal to calculated deadband difference value (That is [(`i_rDbnd`/100) x (`i_wHighLim`-`i_wLowLim`)]), final FB output is equal to previous FB output.<br>• If difference between calculated scaled output and previous FB output is greater than calculated deadband difference value (That is [(`i_rDbnd`/100) x (`i_wHighLim`-`i_wLowLim`)]), final FB output is equal to calculated scaled output.<br>`q_xErr` status output can be equal to 0 or 1 depending on limit checking and hardware detected error monitoring functionality explained above.<br>Note: `i_rDbnd` = 0: No deadband filtering. `i_rDbnd` >= 100: Block all signals. | `i_wSen` = 1000, `i_wHighLim` = 20000, `i_wLowLim` = 4000, `i_wOutHigh` = 2000, `i_wOutLow` = 100, `i_wInHigh` = 1000, `i_wInLow` = 10, `i_rDbnd` = 10. | `q_wSen` = 4000, `q_xErr` = False. |

# Filtering Functions

## What's in This Part



## Overview

This part describes the functionality and implementation of filtering function blocks.

# Global Parameter List

## What's in This Chapter



# Global Parameter List (GPL)

## Overview

| | |
|---|---|
| Type: | Global parameters |
| Available as of: | 3.0.1.0 |

## Description

The global parameter list (GPL) contains global constants which are used by filtering function blocks of this library. The parameters can be edited individually for each application where the library is used. The modification must be done within the **Library Manager** of the project where the library is referenced.

## Global Parameters

| Variable | Data type | Default value | Range | Description |
|---|---|---|---|---|
| *Gc_uiMaxAvgeSmpl* | UINT | 100 | 100... 1000 | The maximum number of samples counted by the *i_uiSmplCnt* input , page 107. |

# Filter_AnalogInput: Checking Analog Input Varaibility

## What's in This Chapter



## Overview

This chapter describes the `Filter_AnalogInput` function block.

# `Filter_AnalogInput` Function Block

## Pin Diagram

This figure shows the pin diagram of the `Filter_AnalogInput` function block:


## Functional Description

The `Filter_AnalogInput` function block checks the plausibility on a measured analog input.

In normal state of operation, if the difference between the present and previous input value:

- is less than or equal to the specified value `i_rMaxIputDiff`, then the output follows the input value.
- is greater than the specified value `i_rMaxIputDiff`, then the output is overwritten by the previous output value for the maximum of three controller scan cycles. Output overwritten status bit `q_xOputOvwr` is TRUE in this condition.
- exceeds the specified value `i_rMaxIputDiff` for more than three consecutive controller scan cycles, the output again follows the input value.

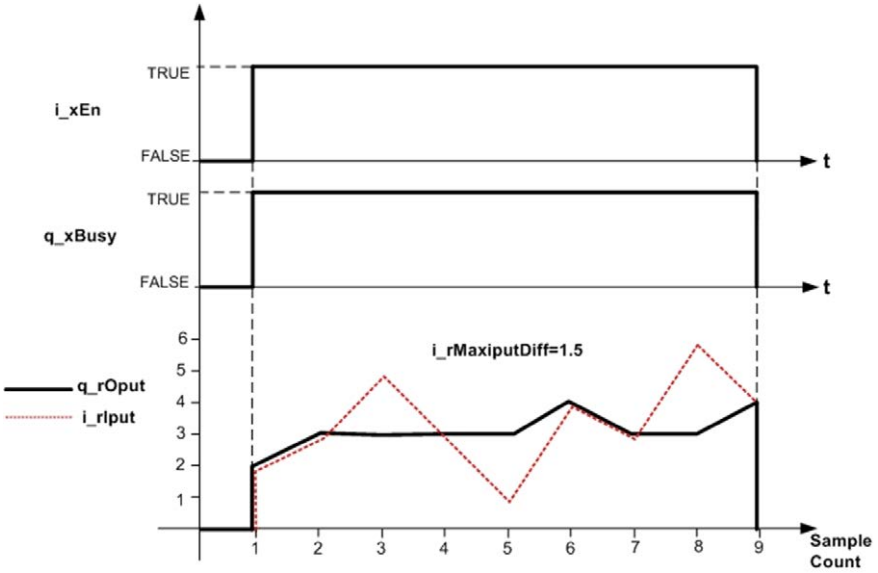
  **NOTE:** When the function block is enabled, during the first scan cycle the input is assigned to the output.

## Example

Maximum difference between present and previous inputs (`i_rMaxIputDiff`) = 1.5:

| Scan Cycle | Input Value (i_rIput) | Output Value (q_rOput) | Output Overwritten Bit (q_xOputOvwr) |
|---|---|---|---|
| First | 2.0 | 2.0 | FALSE |
| Second | 3.0 | 3.0 | FALSE |
| Third | 5.0 | 3.0 | TRUE |
| Fourth | 3.0 | 3.0 | TRUE |
| Fifth | 1.0 | 3.0 | TRUE |
| Sixth | 4.0 | 4.0 | FALSE |

This figure shows the normal behavior of the `Filter_AnalogInput` function block:




## Detected Error State

Invalid parameter such as `i_rMaxIputDiff` < 0 results in a detected error and corresponding detected error ID is generated. During the error detected state, the output is set to zero.

Detected error can be reset only through the rising edge of `i_xErrRst` input.

As shown in the behavior of output figure above, `q_xBusy` is TRUE whenever the function block is enabled and when there is no detected error.

# Input Pin Description

## Input Pin Description

This table describes the input pins of the `Filter_AnalogInput` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xEn` | `BOOL` | TRUE: Enabled<br>FALSE: Disabled |
| `i_rIput` | `REAL` | Analog input variable<br>Range: $1.17e^{-38}$...$3.4e^{+38}$ |

| Input | Data Type | Description |
|---|---|---|
| i_rMaxIputDiff | REAL | Maximum difference between current and previous inputs for plausibility check<br>Range: 0...3.4e+38<br>i_rMaxIputDiff < 0 generates detected error |
| i_xErrRst | BOOL | TRUE: Reset the detected error (on rising edge).<br>(Optional) |

# Output Pin Description

## Output Pin Description

This table describes the output pins of the Filter_AnalogInput function block:

| Output | Data Type | Description |
|---|---|---|
| q_xEn | BOOL | TRUE: Enabled<br>FALSE: Disabled |
| q_xBusy | BOOL | TRUE: Active and no error detected.<br>FALSE: Disabled or detected error. |
| q_xOputOvwr | BOOL | TRUE: If present output is overwritten by previous output.<br>FALSE: If output follows present input. |
| q_rOput | REAL | Analog output value from<br>Range: ±3.4e+38. |
| q_xErr | BOOL | TRUE: Detected error.<br>FALSE: No detected error. |
| q_uiErrId | UINT | Indicates the detected error number when detected error output is set.<br>0: No detected error.<br>1: Invalid parameter i_rMaxIputDiff < 0. |

# Filter_Arithmetic: Giving Arithmetic Mean Value

## What's in This Chapter



## Overview

This chapter describes the `Filter_Arithmetic` function block.

# `Filter_Arithmetic` Function Block

## Pin Diagram

This figure shows the pin diagram of the `Filter_Arithmetic` function block:


## Functional Description

The `Filter_Arithmetic` function block calculates the arithmetic average value for the user defined number of input samples.

Once the function block is enabled, the output calculation begins.

When the number of samples recorded is equal to the specified value `i_uiSmplCnt`, the function block gives an averaged output and the output valid bit `q_xOputVld` becomes TRUE.

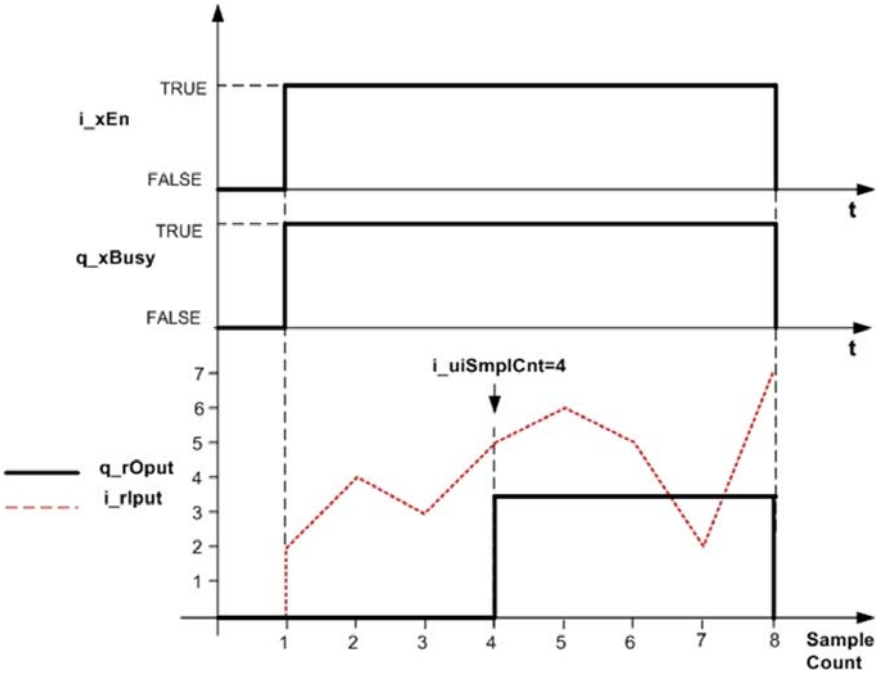
The function block output holds this value until the function block is disabled or is in the detected error state.

## Example

Number of samples to average (`i_uiSmplCnt`) = 4:

| Scan Cycle | Input Value (`i_rIput`) | Output Value (`q_rOput`) | Output Valid Bit (`q_xOputVld`) |
|---|---|---|---|
| First | 2.0 | 0 | FALSE |
| Second | 4.0 | 0 | FALSE |
| Third | 3.0 | 0 | FALSE |
| Fourth | 5.0 | 3.5 | TRUE |
| Fifth | 6.0 | 3.5 | TRUE |
| Sixth | 5.0 | 3.5 | TRUE |

This figure shows normal output behavior:




## Mathematical Background

This equation shows the generalized arithmetic mean value:

$$\bar{x} = \frac{1}{n}\left(\sum_{n} x_n\right)$$

Where:

n = Number of samples entered by user for mean value calculation,

Xn = Input samples,

$\bar{x}$ = Calculated output.

## Detected Error State

Invalid parameter such as `i_uiSmplCnt` = 0 results in a detected error and corresponding detected error ID is generated. During the detected error state, the output is set to zero.

Detected error can be reset only through the rising edge of `i_xErrRst` input.

As shown in output behavior figure above, `q_xBusy` is TRUE whenever the function block is enabled and when there is no detected error.

# Input Pin Description

## Input Pin Description

This table describes the input pins of the `Filter_Arithmetic` function block:

| Input | Data Type | Description |
|---|---|---|
| i_xEn | BOOL | TRUE: Enabled<br>FALSE: Disabled |
| i_rIput | REAL | Input value<br>Range: ±3.4e+38 |
| i_uiSmplCnt | UINT | Number of samples (Input values) to average<br>Range: 1...65535<br>i_uiSmplCnt = 0 generates detected error. |
| i_xErrRst | BOOL | TRUE: Reset the detected error (on rising edge).<br>(Optional) |

# Output Pin Description

## Output Pin Description

This table describes the output pins of the Filter_Arithmetic function bloc:

| Output | Data Type | Description |
|---|---|---|
| q_xEn | BOOL | TRUE: Enabled<br>FALSE: Disabled |
| q_xBusy | BOOL | TRUE: Active and no detected error.<br>FALSE: Disabled or detected error. |
| q_xOputVId | BOOL | TRUE: If number of samples recorded is greater than or equal to i_uiSmplCnt input. |
| q_rOput | REAL | Output value from ±3.4e+38. |
| q_xErr | BOOL | TRUE: Detected error.<br>FALSE: No detected error. |
| q_uiErrId | UINT | Detected error number when error output is set.<br>0: No detected error.<br>1: Invalid parameter i_uiSmplCnt = 0. |

# Filter_MovingAverage: Giving Moving Mean Value

## What's in This Chapter



## Overview

This chapter describes the `Filter_MovingAverage` function block.

# `Filter_MovingAverage` Function Block

## Pin Diagram

This figure shows the pin diagram of the `Filter_MovingAverage` function block:


## Functional Description

The `Filter_MovingAverage` function block calculates the moving average value for the user-defined number of input samples.

When the number of recorded samples are:

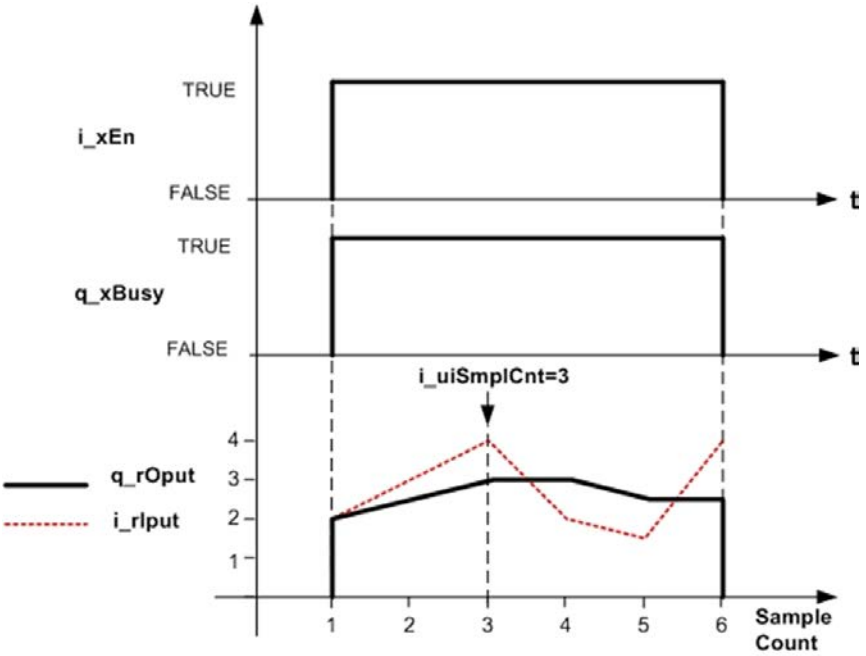
- Less than the specified value `i_uiSmplCnt`, the function block calculates the average value with the available number of inputs and gives the corresponding output.
- Equal to or greater than the specified value `i_uiSmplCnt`, the function block calculates the average value with `i_uiSmplCnt` number of inputs and gives the corresponding output. It operates like the moving average filter.
- For `i_uiSmplCnt` = 0, the input value is assigned to output.

## Example

Number of Samples to average (`i_uiSmplCnt`) = 3:

| Scan Cycle | Input Value (`i_rIput`) | Output Value (`q_rOput`) |
|---|---|---|
| 1 | 2.0 | 2.0 |
| 2 | 3.0 | 2.5 |
| 3 | 4.0 | 3.0 |
| 4 | 2.0 | 3.0 |
| 5 | 1.5 | 2.5 |
| 6 | 4.0 | 2.5 |

This figure shows normal behavior:




## Mathematical Background

This equation shows the generalized equation for the `Filter_MovingAverage` function:

$$\bar{x}_k^n = \frac{1}{n}\left(\sum_{i=k-n}^{k} x_i\right)$$

n = Number of samples,

$x_i$ = Input samples,

k = *GPL.Gc_uiMaxAvgeSmpl*, internal constant,

$\bar{x}_k^n$ = Calculated output.

## Note

In the event of a decrease in the number of sample counts (`i_uiSmplCnt`), the output (`q_rOput`) in the consequent scans is calculated by decreasing the number of samples by one in every consecutive scan.

## Detected Error State

Invalid parameter such as `i_uiSmplCnt` > *GPL.Gc_uiMaxAvgeSmpl* results in detected error and corresponding detected error ID is generated.

During the detected error state, the output is set to zero.

Detected error can be reset only through the rising edge of `i_xErrRst` input.

As shown in behavior of output figure above, `q_xBusy` is TRUE whenever the function block is enabled and when there is no detected error.

# Input Pin Description

## Input Pin Description

This table describes the input pins of the `Filter_MovingAverage` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xEn` | `BOOL` | TRUE: Enabled<br>FALSE: Disabled |
| `i_rIput` | `REAL` | Input value<br>Range: ±3.4e+38 |
| `i_uiSmplCnt` | `UINT` | Number of samples (Input values) to average<br>Range: 0...*GPL.Gc_uiMaxAvgeSmpl*<br>`i_uiSmplCnt` > *GPL.Gc_uiMaxAvgeSmpl* generates detected error. |
| `i_xErrRst` | `BOOL` | Reset the detected error (on rising edge).<br>(Optional) |

# Output Pin Description

## Output Pin Description

This table describes the output pins of the `Filter_MovingAverage` function block:

| Output | Data Type | Description |
|---|---|---|
| `q_xEn` | `BOOL` | TRUE: Enabled<br>FALSE: Disabled |
| `q_xBusy` | `BOOL` | TRUE: Active and no detected error.<br>FALSE: Disabled or detected error. |
| `q_rOput` | `REAL` | Output value<br>Range: ±3.4e+38 |
| `q_xErr` | `BOOL` | TRUE: Detected error.<br>FALSE: No detected error. |
| `q_uiErrId` | `UINT` | Detected error number when error output is set.<br>0: No detected error.<br>1: Invalid parameter `i_uiSmplCnt` > *GPL.Gc_uiMaxAvgeSmpl*. |

# Filter_PT1: Providing PT1 Transfer Function

## What's in This Chapter



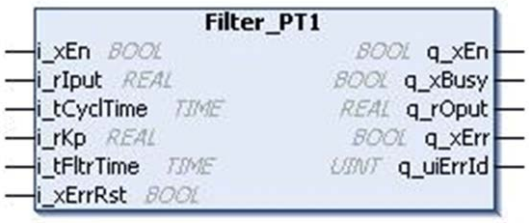
## Overview

This chapter describes the `Filter_PT1` function block.

# `Filter_PT1` Function Block

## Pin Diagram

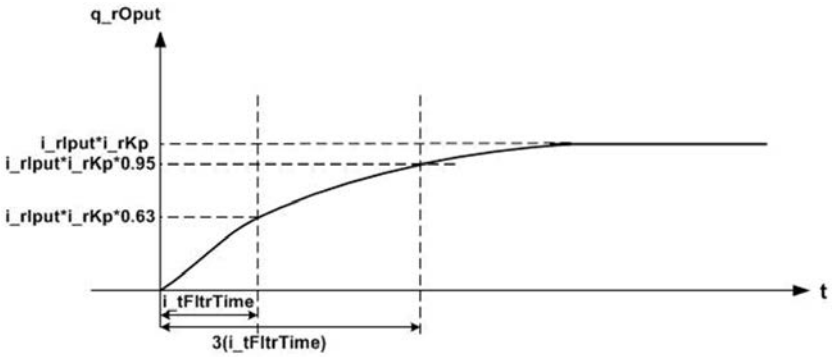
This figure shows the pin diagram of the `Filter_PT1` function block:




## Functional Description

The `Filter_PT1` function block provides a PT1 transfer function. The output value increases to 63% of input value within the time period equal to filter time constant. The output value reaches to 95% of input value after the time period equal to 3 * Filter time constant and then reaches gradually to 100% of the input value.

This figure shows the output profile functionality of the `Filter_PT1` function block:




When the period is equal to:

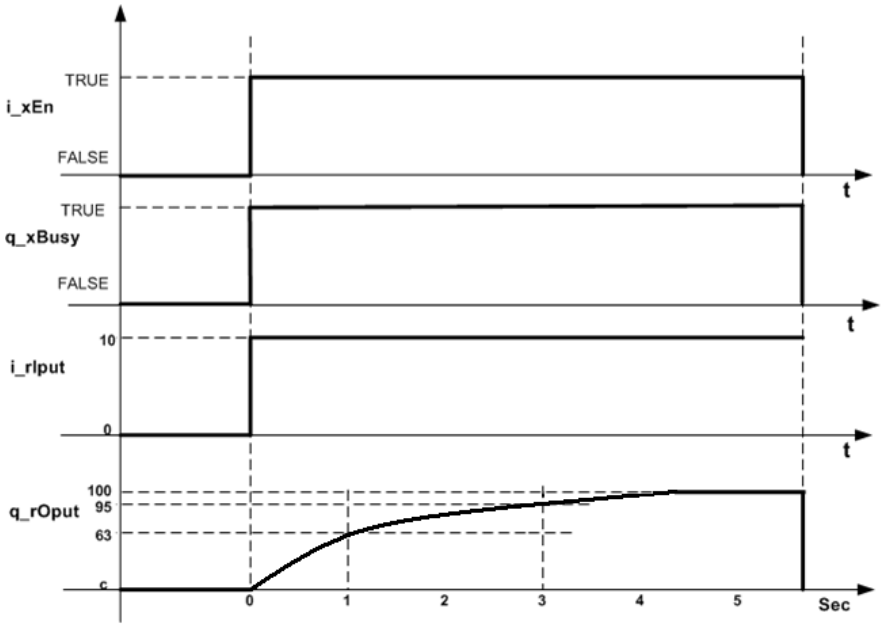
- The filter time constant, the output value increases to 63% of the input value,
- Three times the filter time constant, the output value increases to 95% of the input value and then gradually reaches to 100% of the input.

# Example

If the input value (`i_rIput`) equals 10 and the filtering time constant (`i_tFltrTime`) is one second for a filtering gain of 10, then the output value (`q_rOput`) will be equal to 63 after a time period of one second.

The output value will be equal to 95 after a time period of three seconds (three times the filter time constant), and then the output will gradually reach to 100.

This figure shows normal behavior:




# Mathematical Background

This equation shows the transfer function:

$$G(s) = K_p \frac{1}{1 + T_s}$$

Where:

| | |
|---|---|
| Kp | = Function PT1 gain or amplification |
| Ts | = Function PT1 filter time constant |
| G(s) | = transfer function |

The equation shown above is a Laplace notation for the first-order low pass filter.

In digital-time systems, this function is often referred to as the pulse-transfer function (PT1 function).

# Detected Error State

Invalid parameter such as `i_tCyclTime` = 0 or `i_tFltrTime` < `i_tCyclTime` results in a detected error and corresponding detected error ID is generated. During the detected error state, the output is set to zero.

Detected error can be reset only through the rising edge of `i_xErrRst` input.

As shown in function block the output figure, `q_xBusy` is TRUE whenever the function block is enabled and when there is no detected error.

# Input Pin Description

## Input Pin Description

This table describes the output pins of the `Filter_PT1` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xEn` | `BOOL` | TRUE: Enabled<br>FALSE: Disabled |
| `i_rIput` | `REAL` | Input value for processing<br>Range: ±3.4e$^{+38}$ |
| `i_tCyclTime` | `TIME` | Task cycle time<br>Range: 0...4294967295 ms<br>`i_tCyclTime` < 0 generates detected error. |
| `i_rKp` | `REAL` | Function PT1 gain/amplification<br>Range: ±3.4e$^{+38}$ |
| `i_tFltrTime` | `TIME` | Filter time constant<br>Range: 0...4294967295 ms<br>Factory setting = t#0ms,<br>`i_tFltrTime` < `i_tCyclTime` generates detected error. |
| `i_xErrRst` | `BOOL` | TRUE: Reset the detected error (On rising edge).<br>(Optional) |

# Output Pin Description

## Output Pin Description

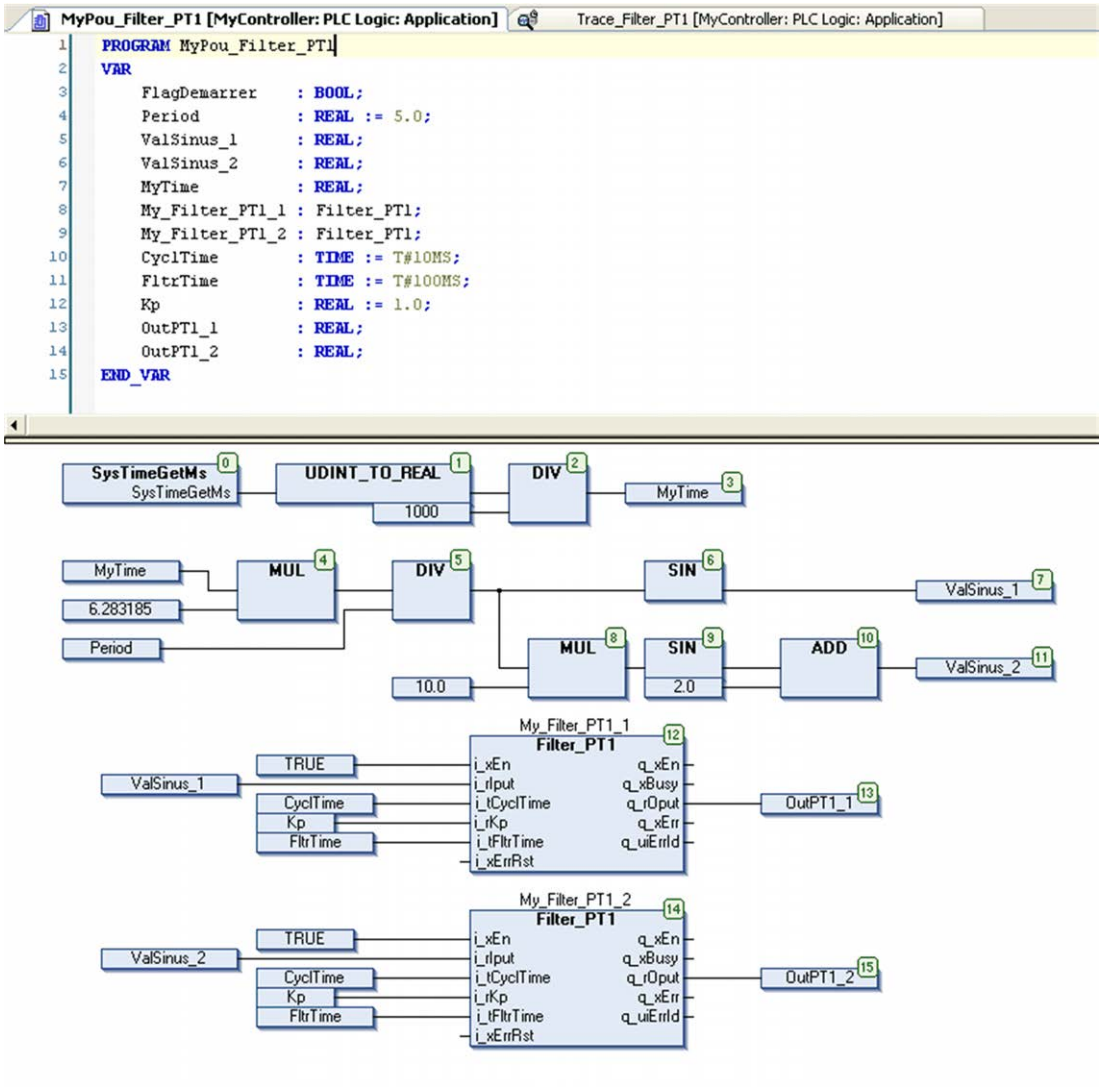
This table describes the output pins of the `Filter_PT1` function block:

| Output | Data Type | Description |
|---|---|---|
| `q_xEn` | `BOOL` | TRUE: Enabled<br>FALSE: Disabled |
| `q_xBusy` | `BOOL` | TRUE: Active and no detected error.<br>FALSE: Disabled or detected error. |
| `q_rOput` | `REAL` | Output value<br>Range: ±3.4e$^{+38}$ |
| `q_xErr` | `BOOL` | TRUE: Detected error.<br>FALSE: No detected error. |
| `q_uiErrId` | `UINT` | Detected error number when error output is set.<br>0: No detected error.<br>1: Invalid parameter `i_tCyclTime` < 0.<br>2: Invalid parameter `i_tFltrTime` < `i_tCyclTime`. |

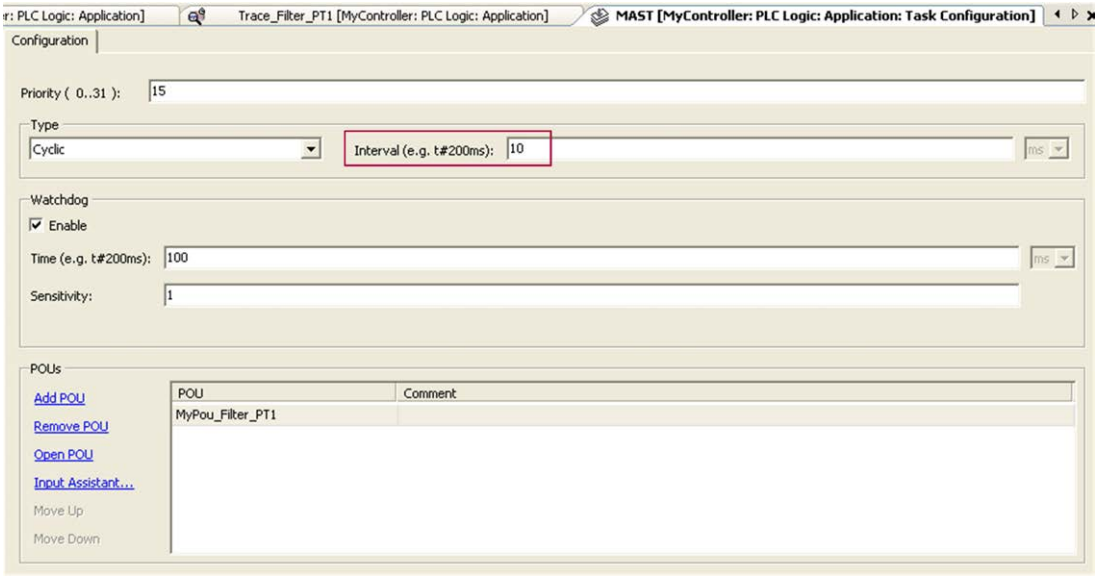
# Instantiation and Usage Example

## Example with a Frequency Signal

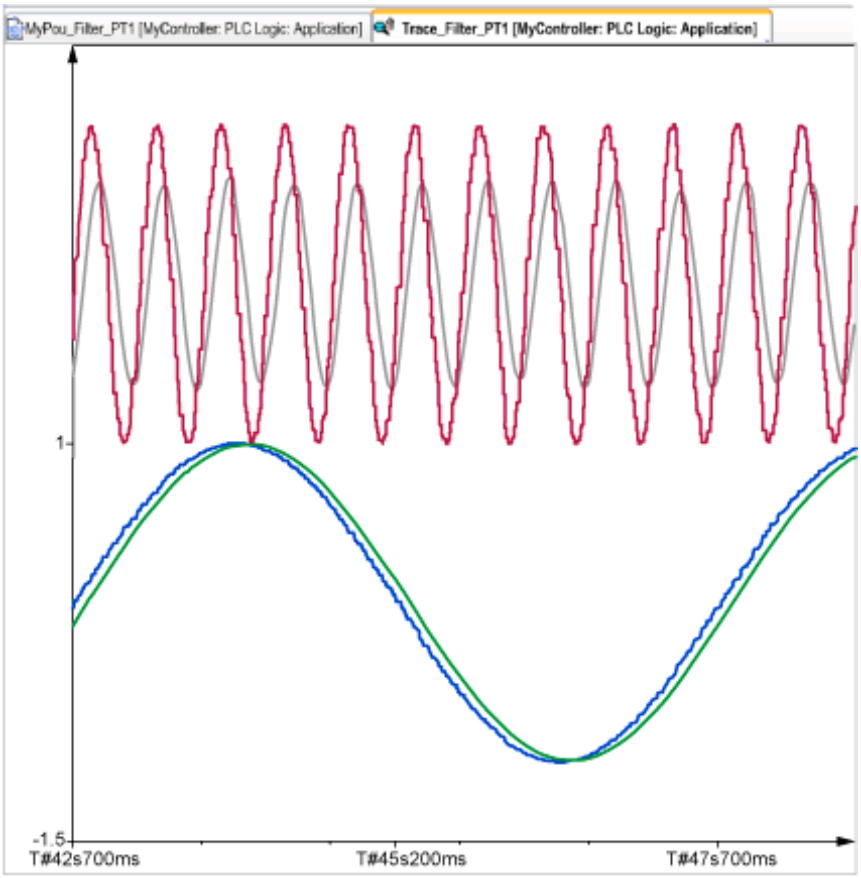
The program creates a Sinusoidal signal at a certain period (5 seconds/0.2 Hz) and a Sinusoidal signal one decade high (0.5 seconds/2 Hz).



The input `i_tCyclTime` of the `Filter_PT1` function block must have exactly the same value as the period of the POU in the MAST, here 10 milliseconds (See the red bordered area).




The result of the previous POU when the input `i_tFltrTime` equals 100 ms:



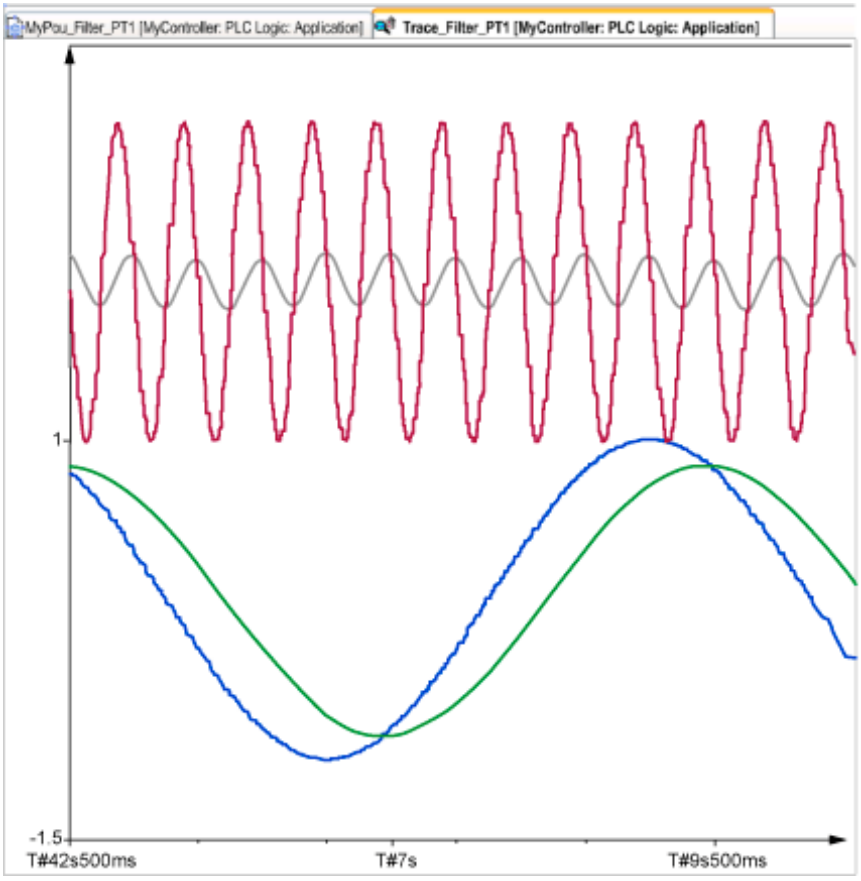

**Blue** `i_rIput` sinusoidal signal at 0.5 Hz (function block `My_Filter_PT1_1`)

**Green** `q_rOput` filtered signal (function block `My_Filter_PT1_1`)

**Red** `i_rIput` sinusoidal signal at 5 Hz (function block `My_Filter_PT1_2`)

**Gray** `q_rOput` filtered signal (function block `My_Filter_PT1_2`)

The result of the previous POU when the input `i_tFltrTime` equals 500 ms:




**Blue** `i_rIput` sinusoidal signal at 0.5 Hz (function block `My_Filter_PT1_1`)

**Green** `q_rOput` filtered signal (function block `My_Filter_PT1_1`)

**Red** `i_rIput` sinusoidal signal at 5 Hz (function block `My_Filter_PT1_2`)

**Gray** `q_rOput` filtered signal (function block `My_Filter_PT1_2`)

# Flip-Flops

## What's in This Part



## Overview

This part describes the Flip-Flops family.

# JK_FlipFlop: Resetting/Setting Input to Flip Flop Output

## What's in This Chapter



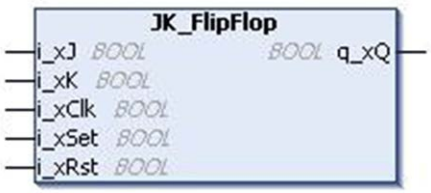
## Overview

This chapter describes the JK_FlipFlop function block.

# JK_FlipFlop Function Block

## Pin Diagram

This figure shows the pin diagram of the JK_FlipFlop function block:




## Functional Description

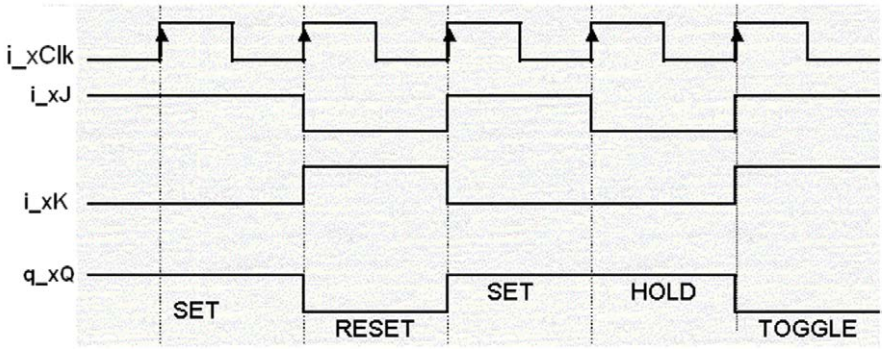
The JK_FlipFlop function block implements the truth table for JK flip-flop.

This function block refers to a flip-flop that obeys this truth table:

| i_xClk | i_xJ | i_xK | q_xQ(n) | q_xQ(n+1) | Operation |
|---|---|---|---|---|---|
| 0 | X | X | X | Q(n) | Hold |
| RE | 0 | 0 | 0 | 0 | Hold |
| RE | 0 | 0 | 1 | 1 | Hold |
| RE | 0 | 1 | 0 | 0 | Reset |
| RE | 0 | 1 | 1 | 0 | Reset |
| RE | 1 | 0 | 0 | 1 | Set |
| RE | 1 | 0 | 1 | 1 | Set |
| RE | 1 | 1 | 0 | 1 | Toggle |
| RE | 1 | 1 | 1 | 0 | Toggle |
| **n** 'n' is the present state and (n+1) is the next state.<br>**RE** Rising Edge | | | | | |

The Reset input (i_xRst) resets the flip flop output q_xQ, whereas Set input (i_xSet) sets the flip flop output q_xQ.

Truth table represented as a time diagram:




## Input Pin Description

This table describes the input pins of the `JK_FlipFlop` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xJ` | `BOOL` | TRUE: `i_xJ` input active.<br>FALSE: Disabled (factory setting) |
| `i_xK` | `BOOL` | TRUE: `i_xK` input active.<br>FALSE: Disabled (factory setting) |
| `i_xClk` | `BOOL` | TRUE: Clock signal active.<br>FALSE: Disabled (factory setting) |
| `i_xSet` | `BOOL` | TRUE: Sets the flip-flop output.<br>FALSE: Disabled (factory setting) |
| `i_xRst` | `BOOL` | TRUE: Resets the flip-flop output.<br>FALSE: Disabled (factory setting) |

## Output Pin Description

This table describes the output pins of the `JK_FlipFlop` function block:

| Output | Data Type | Description |
|---|---|---|
| `q_xQ` | `BOOL` | Flip-flop output |

## Limitations

In JK flip-flop the inputs `i_xSet` and `i_xRst` have higher priority than `i_xJ` and `i_xK` inputs. When both the inputs `i_xSet` and `i_xRst` are either FALSE /TRUE then the output of the FB `q_xQ`, depends upon the inputs `i_xJ` and `i_xK` and `i_xClk`.

# JK_FlipFlop_MasterSlave: Resetting/Setting Input to Flip Flop Output

## What's in This Chapter



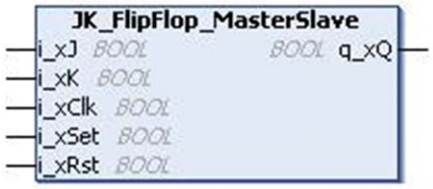
## Overview

This chapter describes the JK_FlipFlop_MasterSlave function block.

# JK_FlipFlop_MasterSlave Function Block

## Pin Diagram

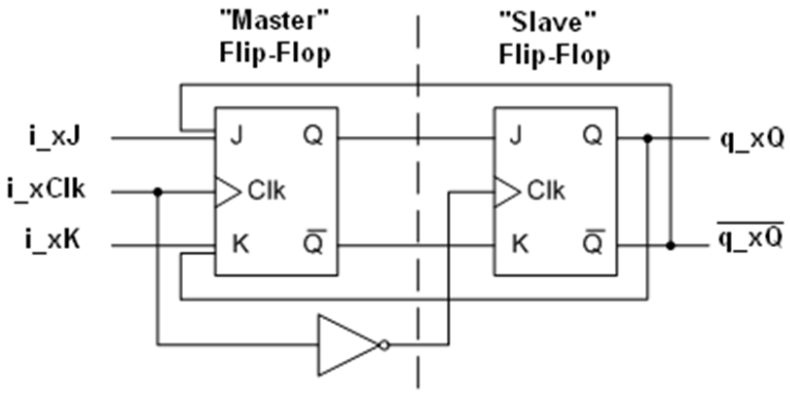
This figure shows the pin diagram of the JK_FlipFlop_MasterSlave function block:




## Functional Description

The JK_FlipFlop_MasterSlave function block implements the truth table for master slave JK flip-flop. The master ouput is captured at rising edge of clock signal and output of slave is updated at falling edge of clock signal.

This diagram represents the internal construction of the JK_FlipFlop_MasterSlave function block.




**NOTE:** The complementary output \q_xQ is not an output of the FB.

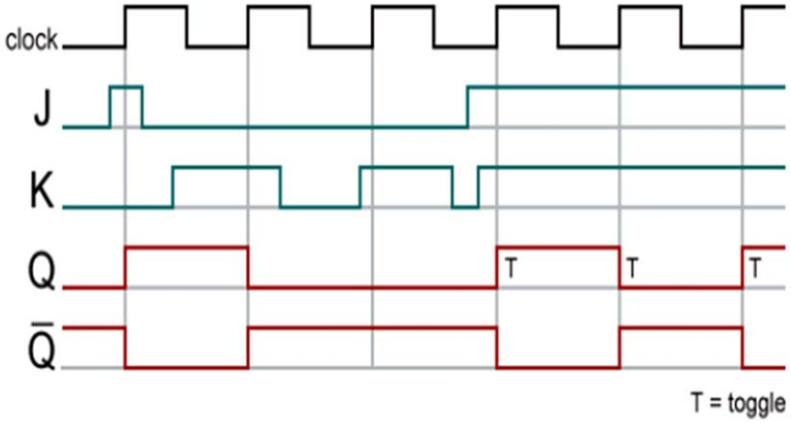
The JK_FlipFlop_MasterSlave refers to a flip-flop that obeys this truth table:

| Set | Reset | CLK | J | K | Q |
|---|---|---|---|---|---|
| 1 | 0 | X | X | X | 1 |
| 0 | 1 | X | X | X | 0 |
| 1 | 1 | X | X | X | 1* |

| Set | Reset | CLK | J | K | Q |
|---|---|---|---|---|---|
| 0 | 0 | ↑ | 0 | 0 | Unv. |
| 0 | 0 | ↑ | 1 | 0 | 1 |
| 0 | 0 | ↑ | 0 | 1 | 0 |
| 0 | 0 | ↑ | 1 | 1 | Toggle |
| 0 | 0 | 0 | X | X | Unv. |

The Reset input (`i_xRst`) resets the flip flop output `q_xQ`, whereas Set input (`i_xSet`) sets the flip flop output `q_xQ`.

Truth table represented as a time diagram:




## Input Pin Description

This table describes the input pins of the `JK_FlipFlop_MasterSlave` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xJ` | `BOOL` | TRUE: `i_xJ` input active.<br>FALSE: Disabled (factory setting) |
| `i_xK` | `BOOL` | TRUE: `i_xK` input active.<br>FALSE: Disabled (factory setting) |
| `i_xClk` | `BOOL` | TRUE: Clock signal active.<br>FALSE: Disabled (factory setting) |
| `i_xSet` | `BOOL` | TRUE: Sets the flip-flop output.<br>FALSE: Disabled (factory setting) |
| `i_xRst` | `BOOL` | TRUE: Resets the flip-flop output.<br>FALSE: Disabled (factory setting) |

## Output Pin Description

This table describes the output pins of the `JK_FlipFlop_MasterSlave` function block:

| Output | Data Type | Description |
|---|---|---|
| `q_xQ` | `BOOL` | Flip-flop output (True / False) |

# Limitations

In JK Master Slave flip-flop the inputs `i_xSet` and `i_xRst` have higher priority than `i_xJ` and `i_xK` inputs. When both the inputs `i_xSet` and `i_xRst` are either FALSE /TRUE then the output of the FB `q_xQ` depends upon the inputs `i_xJ` and `i_xK` and `i_xClk`.

# RS_FlipFlop: Resetting/Setting of Flip Flop Input/ Output

## What's in This Chapter



## Overview

This chapter describes the RS_FlipFlop function block.

# RS_FlipFlop Function Block

## Pin Diagram

This figure shows the pin diagram of the RS_FlipFlop function block:


## Functional Description

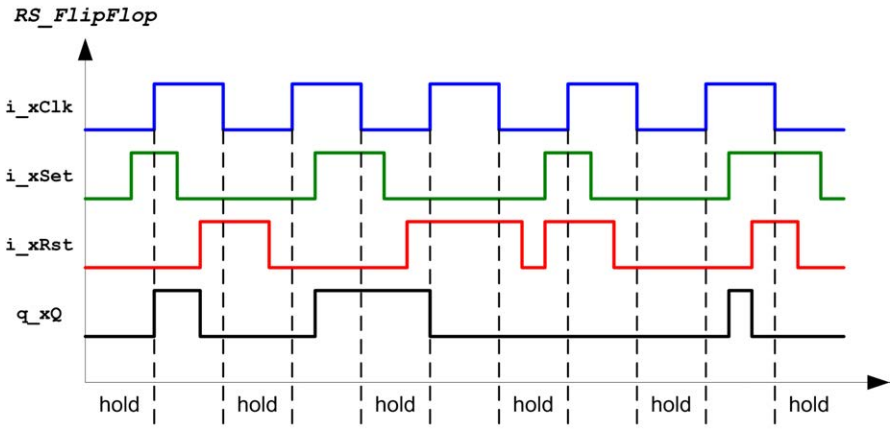
The RS_FlipFlop function block implements the truth table for RS flip-flop with reset priority.

The RS_FlipFlop refers to a flip-flop that obeys this truth table:

| i_xClk | i_xSet | i_xRst | q_xQ(n+1) |
|---|---|---|---|
| 0 | X | X | Q(n) |
| 1 | 0 | 0 | Q(n) |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |
| **n** 'n' is the present state and (n+1) is the next state. | | | |

It has two inputs namely, a Set input or i_xSet, and a Reset input or i_xRst. It also has one output q_xQ. When both set and reset inputs are high, priority is given to Reset input (i_xSet=1 and i_xRst=1).

Truth table represented as a time diagram:




## Input Pin Description

This table describes the input pins of the `RS_FlipFlop` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xClk` | `BOOL` | TRUE: Clock signal active.<br>FALSE: Disabled (factory setting) |
| `i_xSet` | `BOOL` | TRUE: Sets the flip-flop output.<br>FALSE: Disabled (factory setting) |
| `i_xRst` | `BOOL` | TRUE: Resets the flip-flop output.<br>FALSE: Disabled (factory setting) |

## Output Pin Description

This table describes the output pins of the `RS_FlipFlop` function block:

| Output | Data Type | Description |
|---|---|---|
| `q_xQ` | `BOOL` | Flip-flop output |

# SR_FlipFlop: Resetting/Setting of Flip Flop Input/Output

## What's in This Chapter



## Overview

This chapter describes the SR_FlipFlop function block.

# SR_FlipFlop Function Block

## Pin Diagram

This figure shows the pin diagram of the SR_FlipFlop function block:


## Functional Description

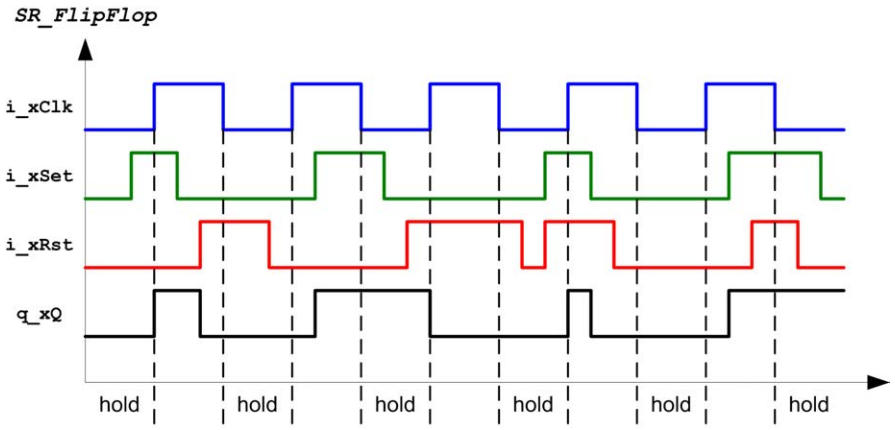
The SR_FlipFlop function block implements the truth table for SR flip-flop with set priority.

The SR_FlipFlop refers to a flip-flop that obeys this truth table:

| i_xClk | i_xSet | i_xRst | q_xQ(n+1) |
|---|---|---|---|
| 0 | X | X | Q(n) |
| 1 | 0 | 0 | Q(n) |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |
| **n** 'n' is the present state and (n+1) is the next state. | | | |

It has two inputs, namely, a Set input, or i_xSet, and a Reset input, or i_xRst. It also has one output, q_xQ. When both set and reset inputs are high, priority is given to Set input (i_xSet=1 and i_xRst=1).

Truth table represented as a time diagram:




## Input Pin Description

This table describes the input pins of the `SR_FlipFlop` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xClk` | `BOOL` | TRUE: Clock signal active.<br>FALSE: Disabled (factory setting) |
| `i_xSet` | `BOOL` | TRUE: Sets the flip-flop output.<br>FALSE: Disabled (factory setting) |
| `i_xRst` | `BOOL` | TRUE: Resets the flip-flop output.<br>FALSE: Disabled (factory setting) |

## Output Pin Description

This table describes the output pins of the `SR_FlipFlop` function block:

| Output | Data Type | Description |
|---|---|---|
| `q_xQ` | `BOOL` | Flip-flop output |

# Toggle_FlipFlop: Toggling of Flip Flop Input/Output

## What's in This Chapter



## Overview

This chapter describes the Toggle_FlipFlop function block.

# Toggle_FlipFlop Function Block

## Pin Diagram

This figure shows the pin diagram of the Toggle_FlipFlop function block:


## Functional Description

The Toggle_FlipFlop function block implements the truth table for T(Toggle) flip-flop with set priority.

The Toggle_FlipFlop is a type of Flip-Flop which obeys this truth table:

| i_xRst | $i\_xT_{(n-1)}$ | $i\_xT_{(n)}$ | $q\_xQ_{(n)}$ | $q\_xQ_{(n+1)}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | X | $Q_{(n)}$ |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | x | x | $Q_{(n)}$ |
| 1 | x | x | x | 0 |
| **n** 'n' is the present state and (n+1) is the next state. | | | | |

It has two inputs, namely, i_xT input, and a Reset input, or i_xRst. It also has an output q_xQ.

## Input Pin Description

This table describes the input pins of the Toggle_FlipFlop function block:

| Input | Data Type | Description |
|---|---|---|
| i_xT | BOOL | Rising edge 0...1 toggles the flip-flop.<br>Factory setting: FALSE |
| i_xRst | BOOL | TRUE: Resets the flip-flop output.<br>FALSE: Disabled (factory setting) |

# Output Pin Description

This table describes the output pins of the `Toggle_FlipFlop` function block:

| Output | Data Type | Description |
|---|---|---|
| `q_xQ` | `BOOL` | Flip-flop output |

# Mathematical Functions

## What's in This Part



## Overview

This part describes the Mathematical function family.

# Analysis: Calculating Integral and Derivative Values

## What's in This Chapter



## Overview

This chapter describes the `Analysis` function block.

# `Analysis` Function Block

## Pin Diagram

This figure shows the pin diagram of the `Analysis` function block:


## Functional Description

The `Analysis` function block calculates the integral and derivative values of a series of input. The output starts with zero at the rising edge of `i_xEn`. The integral value increases in multiples of interval input.

In each scan both the integral output and the derivative output is updated based on the interval value.

An error is detected if the interval value is equal to/less than zero or if the input is out of range or if the integral or derivative outputs exceed $3.4e^{+38}$.

Integral = Integral + (Current Input + Previous Input) / 2 * Interval.

Derivative = (Current Input - Previous Input)/ Interval.

## Example

Input = 10 (Previous input: 0), Interval = 10, then the outputs after the first cycle of execution are as below:

- Integral = 0 + (10+0) / 2 * 10 = 50
- Derivative = (10-0)/ 10 = 1

## Input Pin Description

This table describes the input pins of the `Analysis` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xEn` | `BOOL` | TRUE: FB enabled<br>FALSE: FB disabled |
| `i_rIput` | `REAL` | Input value |

| Input | Data Type | Description |
|---|---|---|
| | | Range: $1.17e^{-38}$...$3.4e^{38}$ |
| i_rItvl | REAL | Input value<br>Range: $1.17e^{-38}$...$3.4e^{38}$ |
| i_xErrRst | BOOL | TRUE: Reset the detected error. (On rising edge)<br>(Optional) |

## Output Pin Description

This table describes the output pins:

| output | Data Type | Description |
|---|---|---|
| q_xActv | BOOL | function block status output |
| q_rItgr | REAL | Integral value<br>Range: $1.17e^{-38}$...$3.4e^{38}$ |
| q_rDrvt | REAL | Derivative value<br>Range: $1.17e^{-38}$...$3.4e^{38}$ |
| q_xErr | BOOL | TRUE: i_rItvl input <= 0<br>or i_rIput < $1.17e^{-38}$<br>or i_rIput > $3.4e^{+38}$<br>or q_rItgr > $3.4e^{+38}$<br>or q_rDrvt > $3.4e^{+38}$<br>FALSE: No detected error |

# Frequency_Multiplier: Implementing 32 Blinkers

## What's in This Chapter



## Overview

This chapter describes the Frequency_Multiplier function block.

# Frequency_Multiplier Function Block

## Pin Diagram

This figure shows the pin diagram of the Frequency_Multiplier function block:


## Functional Description

The Frequency_Multiplier function block implements 32 blinkers represented by the bits of output.

On every rising edge of enable signal, the blinker output starts with zero. The lowest bit changes its state after a time base period. The second bit blinks with half the frequency of the initial one. The third bit blinks with half the frequency of the second one and so on, until enable signal is reset. If i_xHold input is set, then current state of blinkers is Hold. If blinkers of type BOOL are required the function block DWORD_AS_BIT (Util Library) can be used.

The output is reset on the rising edge of Enable input.

## Example

Frequencies (Enable = TRUE, Timebase = t#100ms, Hold = FALSE)

DWORD_AS_BIT (Input = Frequency ouput)

DWORD_AS_BIT.B00 is blinking with 100 ms

DWORD_AS_BIT.B01 is blinking with 200 ms

DWORD_AS_BIT.B02 is blinking with 400 ms

## Input Pin Description

This table describes the input pins of the Frequency_Multiplier function block:

| Input | Data Type | Description |
|---|---|---|
| i_xEn | BOOL | TRUE: FB enabled<br>FALSE: Disabled |
| i_tBase | TIME | Time period<br>Range: 1...4294967295 ms (≥ cycle time of controller) |
| i_xHold | BOOL | TRUE: Active<br>FALSE: Disabled |

## Output Pin Description

This table describes the output pins of the Frequency_Multiplier function block:

| output | Data Type | Description |
|---|---|---|
| q_xActv | BOOL | TRUE: FB enabled<br>FALSE: Disabled |
| q_dwOput | DWORD | Output status<br>Range: 0...4294967295 |

# Without Hold Condition Description

## Without Hold Condition

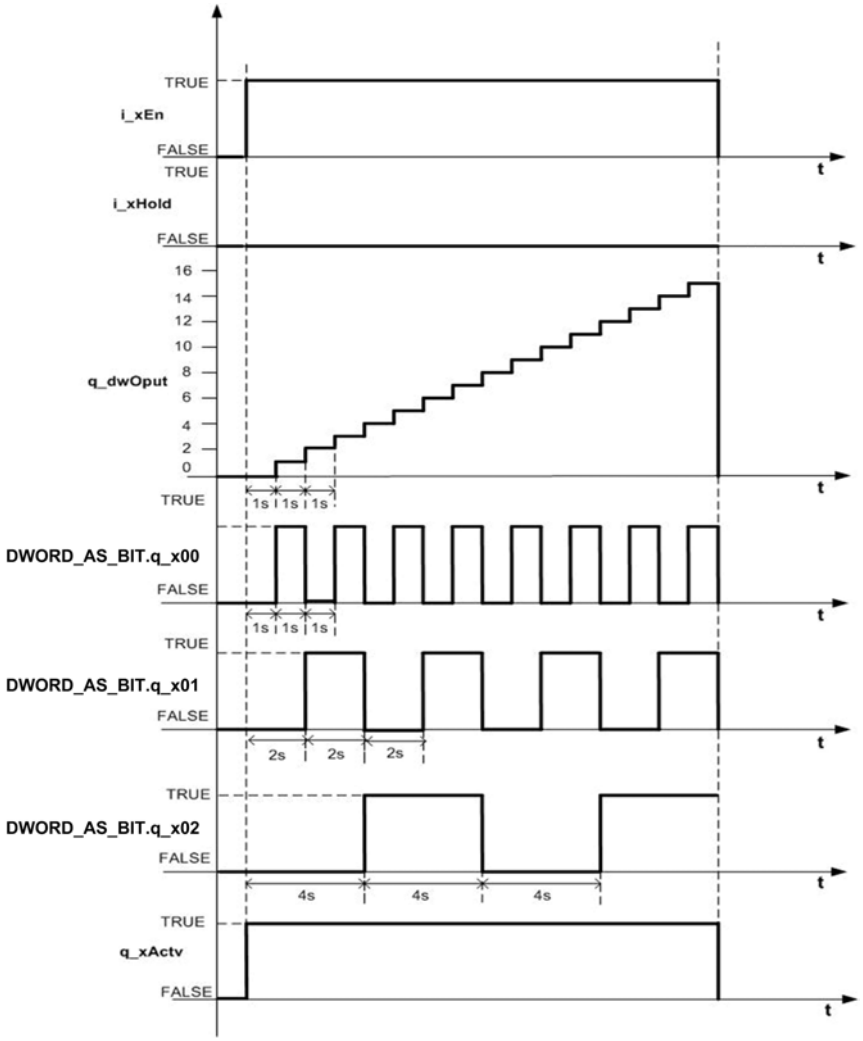
If input at pin i_xEn is high, i_tBase time is set to 1s and i_xHold input is FALSE, then q_dwOput is increased by one, after the completion of i_tBase time period.

DWORD_AS_BIT (Input: = Frequency_Multiplier. q_dwOput):

- DWORD_AS_BIT.q_x00 is ON after 1 s of Enabling FB for Time base period
- DWORD_AS_BIT.q_x01 is ON after 2 s of Enabling FB for 2 * Time base period
- DWORD_AS_BIT.q_x02 is ON after 4 s of Enabling FB for 4 * Time base period

# Timing Diagram

This figure shows the timing diagram for the `Frequency_Multiplier` function block without hold input:



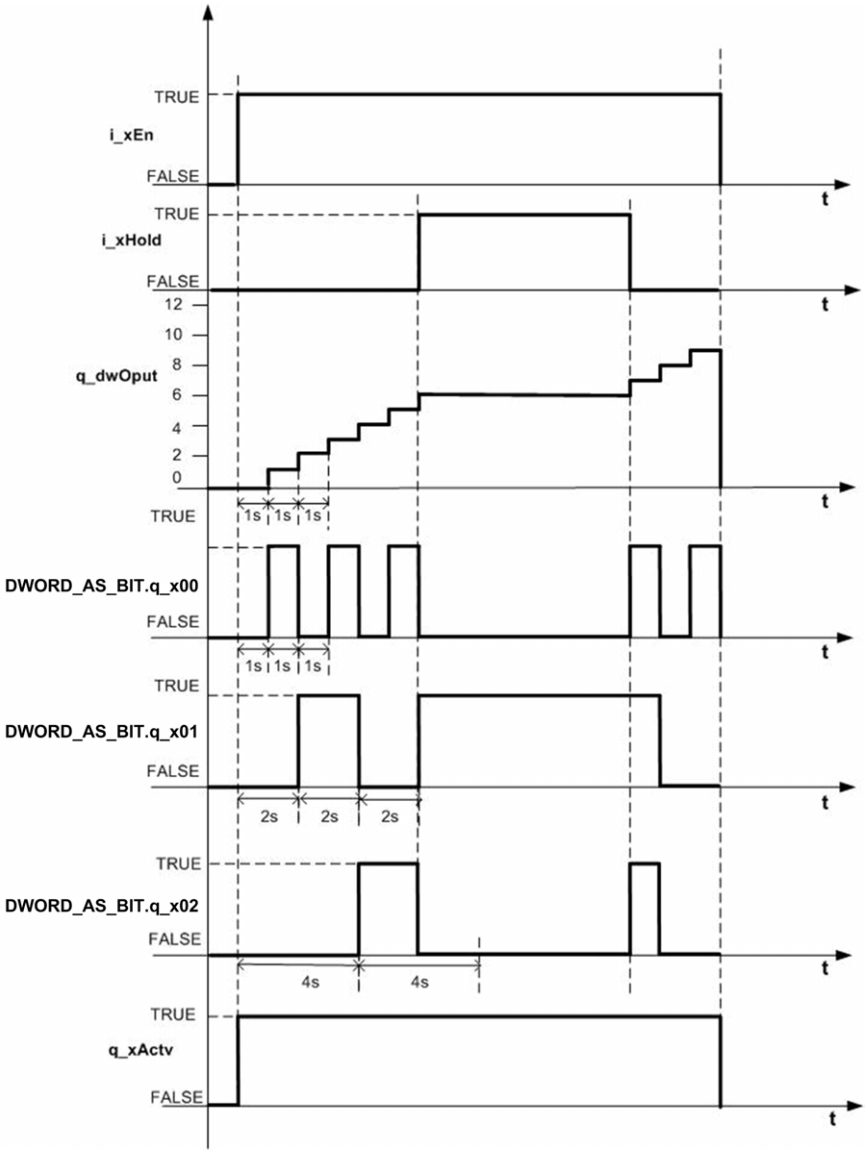

# Functionality With Condition Description

## Functionality Description

If input at pin `i_xEn` is high, `i_tBase` time is set to 1s and `i_xHold` input is TRUE, then `q_dwOput` HOLD the previous status of Blinkers. FB resumes with its normal operation again when `i_xHold` input goes to FALSE.

# Timing Diagram

This figure show the timing diagram for the `Frequency_Multiplier` function block with hold input:



# Frequency_Output: Implementing a Frequency

## What's in This Chapter



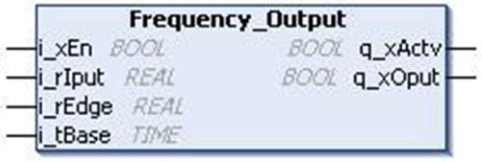
## Overview

This chapter describes the Frequency_Output function block.

# Frequency_Output Function Block

## Pin Diagram

This figure shows the pin diagram of the Frequency_Output function block:




## Functional Description

The Frequency_Output function block implements frequencies.

A positive signal at i_xEn activates the block. The output signal starts with TRUE and holds this signal for a duration of (TimeBase * i_rIput) and then it changes to FALSE for a duration of TimeBase * (1-i_rIput).

If the input signal enters the edge range 0 to i_rEdge, the output signal no longer alternates and is continually FALSE. If it enters the range 1-i_rEdge to1 then the output signal is continuously TRUE.

If the signal at i_xEn is removed, the output signal remains at its current value until the function block is restarted through a positive signal at i_xEn.

## Input Pin Description

This table describes the input pins of the Frequency_Output function block:

| Input | Data Type | Description |
|---|---|---|
| i_xEn | BOOL | TRUE: FB enabled<br>FALSE: Disabled |
| i_rIput | REAL | Input value<br>Range: 0.0...1.0 |
| i_rEdge | REAL | Input value<br>Range: 0.0...1.0 |
| i_tBase | TIME | Time period<br>Range: 0...4294967295 ms |

# Output Pin Description

This table describes the output pins of the `Frequency_Output` function block:

| output | Data Type | Description |
|---|---|---|
| `q_xActv` | `BOOL` | function block active status |
| `q_xOput` | `BOOL` | Output value |

# Instantiation and Usage Example

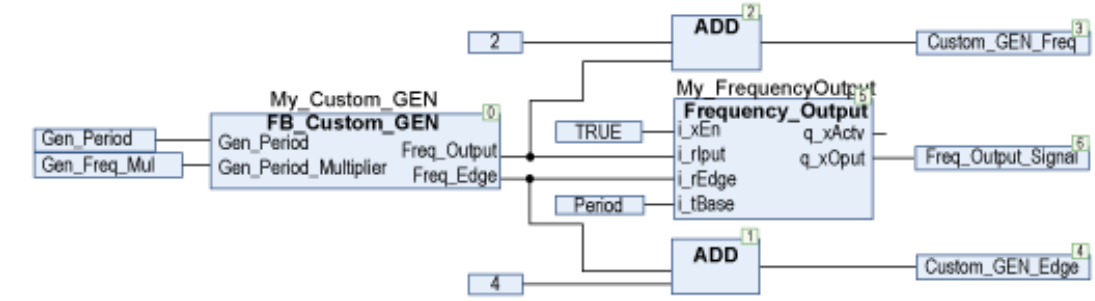
The example use the following custom signal generators that creates 2 signals :

- A 5 seconds period signal that will be used as the input for the `Frequency_Output` function block.
- A 50 seconds period signal that will be used as the edge for the `Frequency_Output` function block.

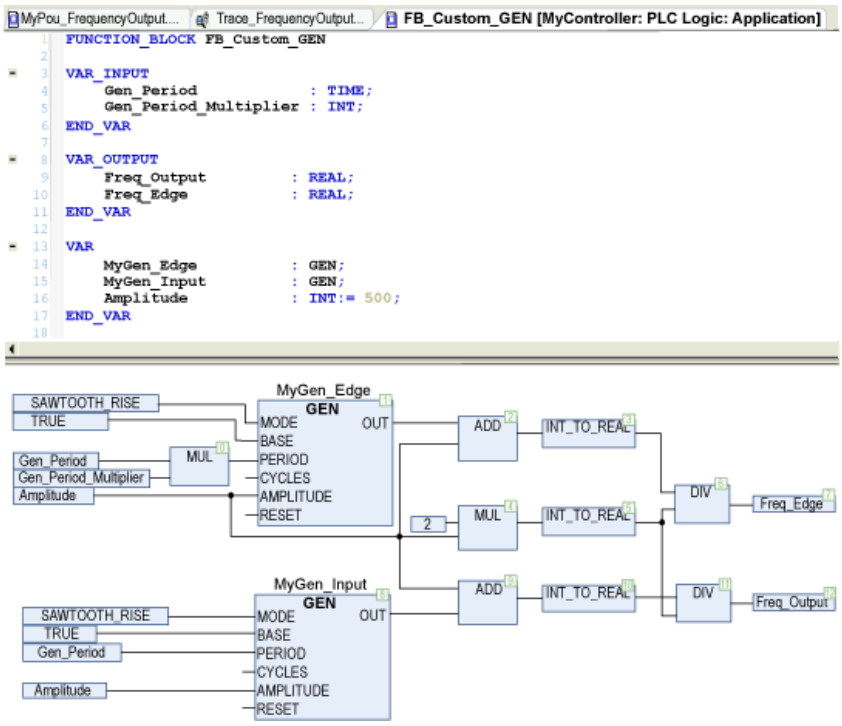
This POU has a period of 10 ms in the MAST.

MyPou_FrequencyOutput [MyController: PLC Logic: Application] | Trace_FrequencyOutput [MyController: PLC Logic: A...

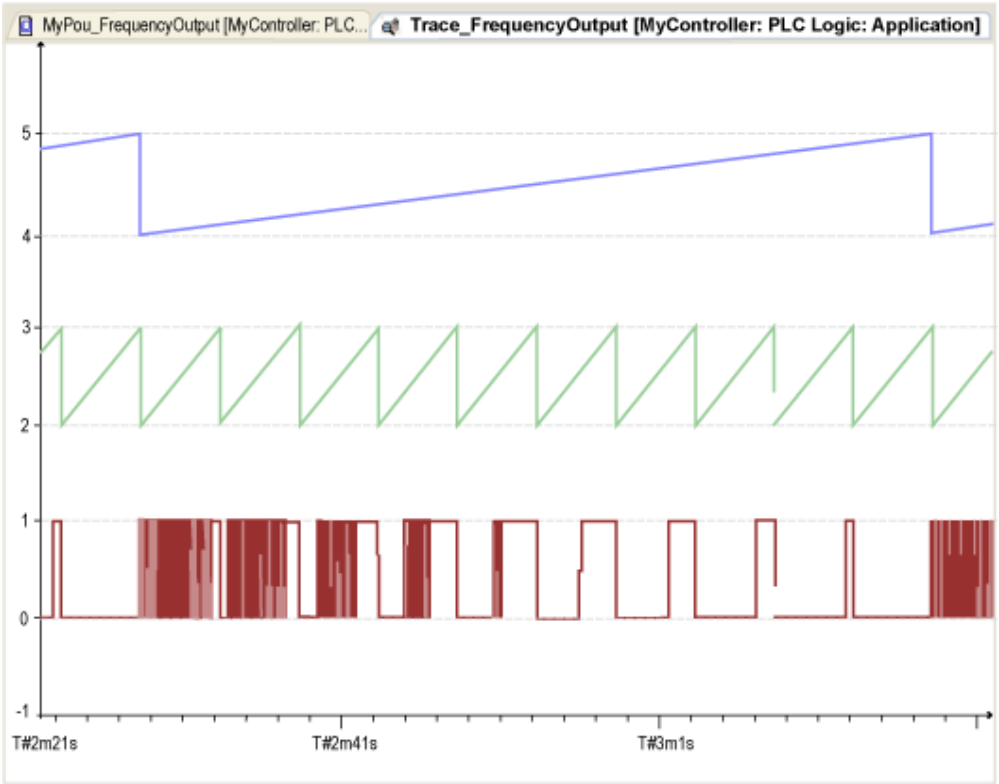
```
PROGRAM MyPou_FrequencyOutput
VAR
 FlagDemarrer          : BOOL;

 My_Custom_GEN         : FB_Custom_GEN;
 Gen_Period            : TIME := T#5000MS;
 Gen_Freq_Mul          : INT := 10;

 My_FrequencyOutput    : Frequency_Output;
 Period                : TIME := T#100MS;
 Freq_Output_Signal    : BOOL;
 Custom_GEN_Freq       : REAL;
 Custom_GEN_Edge       : REAL;
 Grid_Draw_1: REAL;
END_VAR
```



This figure shows the custom signal generator:




This example gives the following results:




**Blue** `i_rEdge` signal, period of 50 seconds.

**Green** `i_rIput` signal, period of 5 seconds.

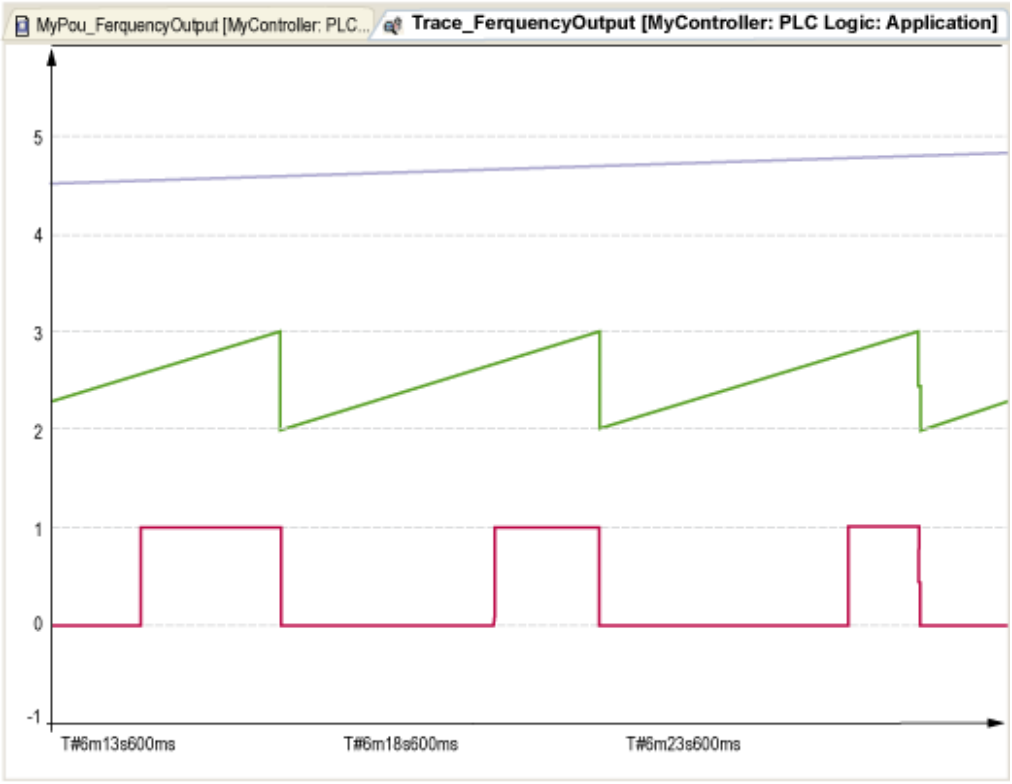
**Red** `q_xOput`, output of the `Frequency_Output` function block.

This table shows the truth table:

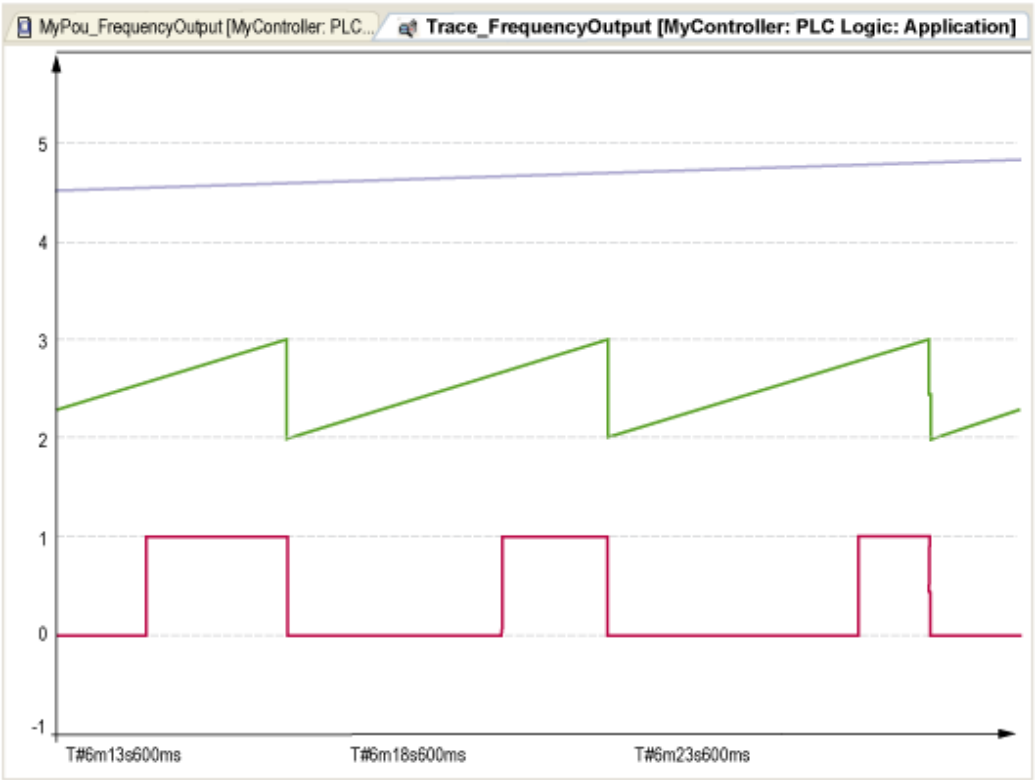
| Edge level | Input level | Output |
|---|---|---|
| `i_rEdge` < 0.5 | `i_rIput` < `i_rEdge` | FALSE |
| | `i_rEdge` < `i_rIput` < (1-`i_rEdge`) | PWM; duty = `i_rIput` |

| Edge level | Input level | Output |
|---|---|---|
| | (1 - `i_rEdge`) < `i_rIput` | TRUE |
| `i_rEdge` >= 0.5 | `i_rIput` < `i_rEdge` | FALSE |
| | `i_rEdge` <= `i_rIput` | TRUE |

Example with `i_rEdge` < 0.5:

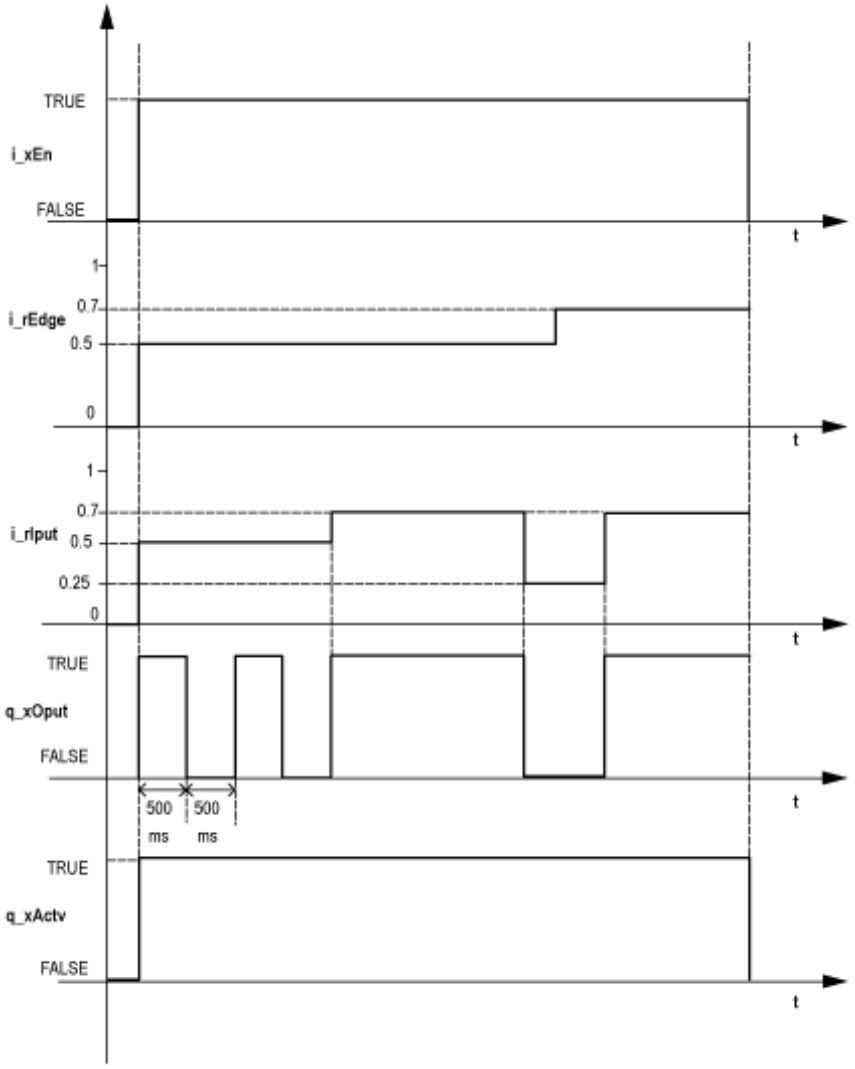



Example with `i_rEdge` >= 0.5:



Example: If the input at pin `i_xEn` is high, `i_rIput` input and initially `i_rEdge` input is set to 0.5 and `i_tBase` input is set to 1sec, then `q_xOput` is set for 500ms and reset for 500ms time period. After some time the input `i_rEdge` is changed to 0.7.

This figure shows the timing diagram of the above example:



# Normalizer_With_Limiter: Scaling the Minimum and Maximum Input

## What's in This Chapter



## Overview

This chapter describes the Normalizer_With_Limiter function block.

# Normalizer_With_Limiter Function Block

## Pin Diagram

This figure shows the pin diagram of the Normalizer_With_Limiter function block:


## Functional Description

The Normalizer_With_Limiter function block scales the minimum and maximum input value to the range of minimum and maximum output value. The input value can be limited to minimum and maximum output values.

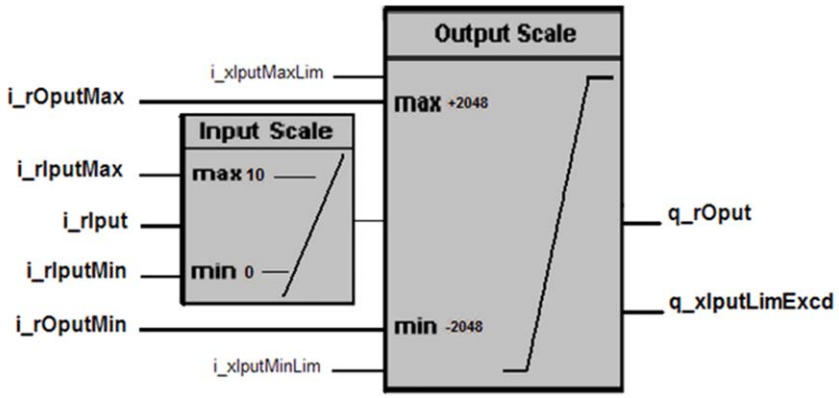
The output value can be limited to minimum and maximum output values using i_xIputMinLim and i_xIputMaxLim.

When the input exceeds the maximum/minimum input limit value q_xIputLimExcd will be TRUE.

## Mathematical Backround

$$q\_rOput = \frac{(i\_rIput - i\_rIputMin) \times (i\_rOputMax - i\_rOputMin)}{(i\_rIputMax - i\_rIputMin)} + i\_rOputMin$$

# Example




| i_xIputMaxLim | i_xIputMinLim | i_rIput | q_rOput | q_xIputLimExcd |
|---|---|---|---|---|
| FALSE | FALSE | 12.5 | 3072 | TRUE |
| TRUE | FALSE | 12.5 | 2048 | TRUE |
| TRUE | FALSE | 10 | 2048 | FALSE |
| TRUE | FALSE | 7.5 | 1024 | FALSE |
| TRUE | TRUE | 5 | 0 | FALSE |
| FALSE | TRUE | 2.5 | -1024 | FALSE |
| FALSE | TRUE | 0 | -2048 | FALSE |
| FALSE | TRUE | -2.5 | -2048 | TRUE |
| FALSE | FALSE | -2.5 | -3072 | TRUE |

# Input Pin Description

This table describes the input pins of the `Normalizer_With_Limiter` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_rIput` | `REAL` | Input value<br>Range: ±3.4e+38 |
| `i_rIputMin` | `REAL` | Minimum input<br>Range: ±3.4e+38 |
| `i_rIputMax` | `REAL` | Maximum input value<br>Range: ±3.4e+38 |
| `i_rOputMin` | `REAL` | Minimum output value<br>Range: ±3.4e+38 |
| `i_rOputMax` | `REAL` | Maximum output value<br>Range: ±3.4e+38 |
| `i_xIputMinLim` | `BOOL` | TRUE: Limit enabled for minimum input value<br>FALSE:Limit disabled |
| `i_xIputMaxLim` | `BOOL` | TRUE: Limit enabled for maximum input value<br>FALSE:Limit disabled |

# Output Pin Description

This table describes the output pins of the `Normalizer_With_Limiter` function block:

| Output | Data Type | Description |
|---|---|---|
| `q_rOput` | `REAL` | Scale output value<br>Range: ±3.4e$^{+38}$ |
| `q_xIputLimExcd` | `BOOL` | Input value exceed limit value |

# ONE_SEC_PULSE: Providing Pulses Every One Second

### What's in This Chapter



## Overview

This chapter describes the ONE_SEC_PULSE function block.

## ONE_SEC_PULSE Function Block

## Pin Diagram

This figure shows the pin diagram of the ONE_SEC_PULSE function block:


## Functional Description

The ONE_SEC_PULSE function block generates pulses of one second duration at the output q_xOput.

## Output Pin Description

This table describes the output pins of the ONE_SEC_PULSE function block:

| output | Data Type | Description |
|---|---|---|
| q_xOput | BOOL | Output Pulse<br>Output state = TRUE during 1 cycle of task<br>Frequency = 1 Hz |

## Instantiation and Usage Example

The above figure shows an instance of ONE_SEC_PULSE function block:

# Quantizer: Digitalizing the Input Value for the Interval

## What's in This Chapter



## Overview

This chapter describes the `Quantizer` function block.

# `Quantizer` Function Block

## Pin Diagram

This figure shows the pin diagram of the `Quantizer` function block:


## Functional Description

The `Quantizer` function block discretes the input value (–32768 to 32767) for a given interval.

If the input is more than the input range then the detected error output is TRUE and the output displays a zero value.

## Mathematical Background

$$q\_rOput = i\_rItvl \times ROUND\left(\frac{i\_rIput}{i\_rItvl}; 0\right)$$

With Itvl = interval.

## Example

| Input | Interval | Output |
|---|---|---|
| 32766.7 | 1 | 32767<br>Detected error = FALSE |
| 32768 | 15 | 0<br>Detected error = TRUE |
| -32768 | 20 | -32760<br>Detected error = FALSE |
| 36.89 | 15 | 30<br>Detected error = FALSE |

| Input | Interval | Output |
|---|---|---|
| -47.98 | -10 | -50<br>Detected error = FALSE |
| -42.14 | -10 | -40<br>Detected error = FALSE |
| 3456.78 | 80 | 3440<br>Detected error = FALSE |
| Between -4.99 to 4.99 | 10 | 0 |
| Between 5 to 14.99 | 10 | 10 |

## Input Pin Description

This table describes the input pins of the `Quantizer` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_rIput` | `REAL` | Input value<br>Range: -32768...32767 |
| `i_rItvl` | `REAL` | Quantization interval input value<br>Range: $\pm 3.4e^{+38}$ |
| `i_xErrRst` | `BOOL` | Reset the detected error. (On rising edge)<br>(Optional) |

## Output Pin Description

This table describes the output pins of the `Quantizer` function block:

| output | Data Type | Description |
|---|---|---|
| `q_rOput` | `REAL` | Output value<br>Range: -32768...32767 |
| `q_xErr` | `BOOL` | TRUE: Input limit exceed<br>FALSE: No detected error |

# Signal_Saturation: Limiting to Upper and Lower Saturation Limit

## What's in This Chapter



## Overview

This chapter describes the Signal_Saturation function block.

# Signal_Saturation Function Block

## Pin Diagram

This figure shows the pin diagram of the Signal_Saturation function block:


## Functional Description

The Signal_Saturation function block limits the input signal to upper and lower saturation limit.

When low input value is more than the high value then the detected error output is TRUE and the output displays zero.

When the input exceeds the low/high input limit value, then q_xIputLimExcd is TRUE.

## Input Pin Description

This table describes the input pins of the Signal_Saturation function block:

| Input | Data Type | Description |
|---|---|---|
| i_rIput | REAL | Input value<br>Range: ±3.4e+38 |
| i_rLow | REAL | Lower input value<br>Range: ±3.4e+38 |
| i_rHigh | REAL | Higher input value<br>Range: ±3.4e+38 |
| i_xErrRst | BOOL | Reset the detected error. (On rising edge)<br>(Optional) |

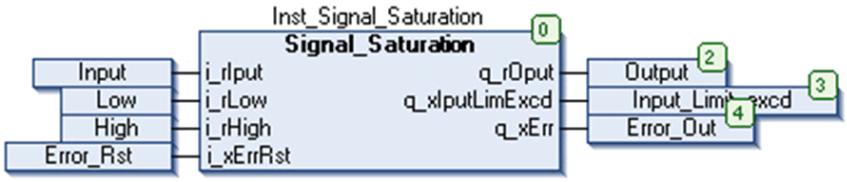
# Output Pin Description

This table describes the output pins of the `Signal_Saturation` function block:

| Output | Data Type | Description |
|---|---|---|
| `q_rOput` | `REAL` | Output value ±3.4e$^{+38}$ |
| `q_xIputLimExcd` | `BOOL` | TRUE: Input value exceed limit value. |
| `q_xErr` | `BOOL` | TRUE: Input limit wrong<br>FALSE: No detected error |

# Instantiation and Usage Example

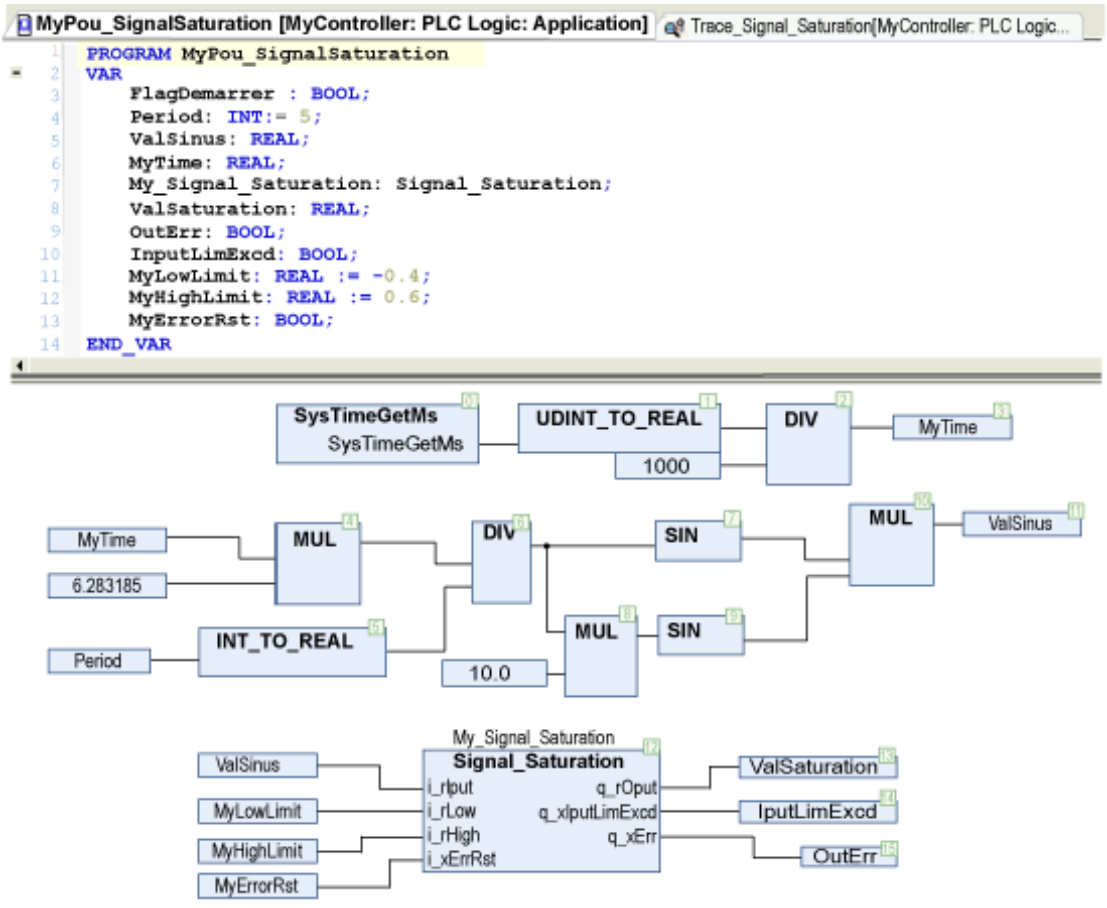
This figure shows an instance of the `Signal_Saturation` function block:




If input `i_rIput` is set to 4, `i_rLow` is set to 5 and `i_rHigh` is set to 10, then saturation output `i_rLow` value is 5 & `q_xIputLimExcd` is TRUE.

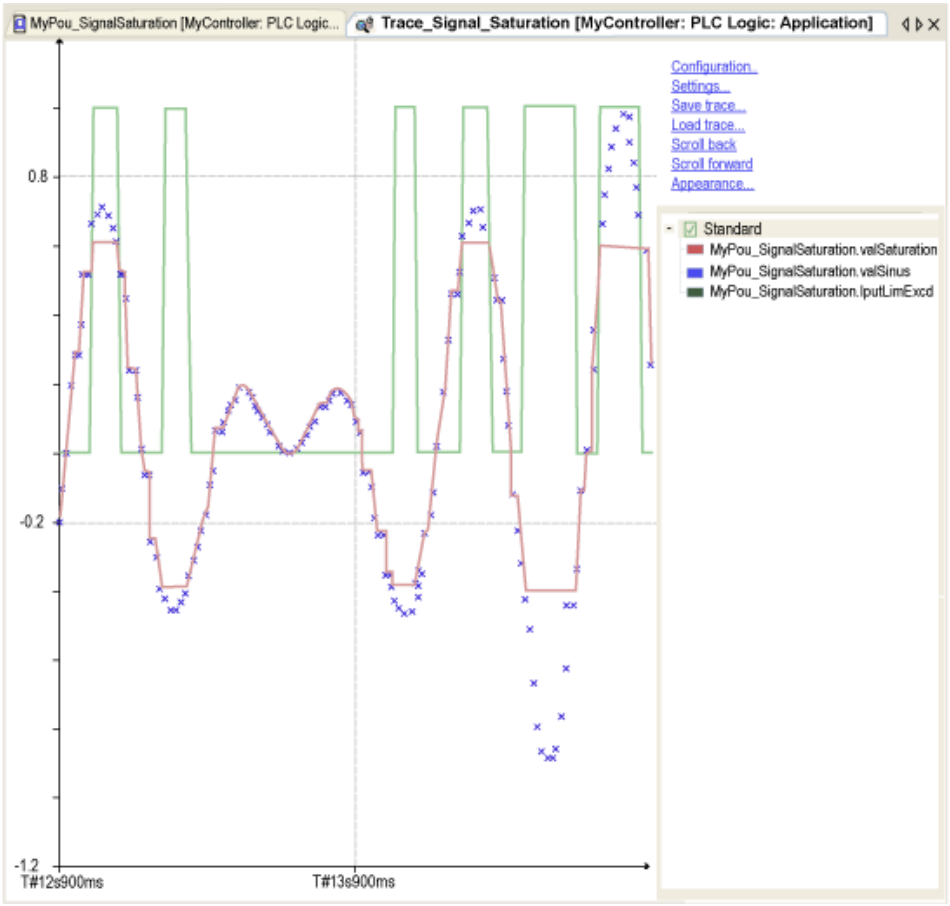
# CFC Example

This figure shows the CFC example on `Signal_Saturation` implementation:



# Timing Diagram

This figure shows the timing diagram for the `Signal_Saturation` function block:

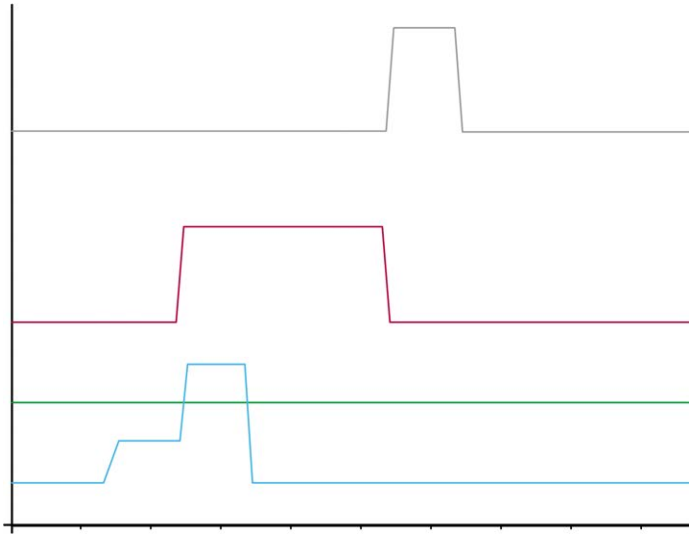



**Blue** input signal

**Red** output signal, limited in the range [Low ; High]/[-0,4 ; +0,6].

**Green** `IputLimExcd`, TRUE when the input signal is out of the range.

This figure shows the timing diagram of `q_xErr` output:



**Blue:** `i_rLow` input signal

**Green:** `i_rHigh` input signal

**Red:** `q_xErr` output signal, TRUE as soon as `i_rLow` is higher than `i_rHigh`.

**Gray:** `i_xErrRst` input signal, reset the `q_xErr` output signal on rising edge while `i_rLow` is lower than `i_rHigh`.

# Signal_Statistics: Calculating Maximum, Minimum, Average and Variance

## What's in This Chapter



## Overview

This chapter describes the `Signal_Statistics` function block.

# `Signal_Statistics` Function Block

## Pin Diagram

This figure shows the pin diagram of the `Signal_Statistics` function block:


## Functional Description

This function block calculates the maximum, minimum, average and variance of a series of input values.

This function block considers the input value in each controller scan cycle as a Sample.

## Minimum Value

Minimum Output value is the value which is minimum amongst all the recorded samples.

## Maximum value

Maximum Output value is the value which is maximum amongst all the recorded samples.

## Average Value

The average value is equal to the sum of the observations (samples) divided by the number of observations (samples).

$$\bar{x} = \frac{1}{n}\left(\sum_{n} x_n\right)$$

Where:

- n = Number of samples recorded
- Xn = Input samples
- $\bar{x}$ = Calculated output

## Variance Value

The variance is equal to the mean of the squares of samples minus the square of the mean (Average output value).

$$\bar{x} = \frac{1}{n}\left(\sum_{1}^{n} (x_n)^2\right) - \left(\frac{1}{n}\sum_{1}^{n} x_n\right)^2$$

Where:

- n = Number of samples recorded
- Xn = Input samples
- $\bar{x}$ = Calculated output

## Example

- Statistics (Enable: = TRUE, Input: = 1, 2
- Minimum Output =1
- Maximum Output = 2
- Average = (1 + 2) / 2 = 1.5
- Variance = ((1 * 1 + 2 * 2) / 2) - (1.5 * 1.5) = 2.5 - 2.25 = 0.25

## Input Pin Description

This table describes the input pins of the `Signal_Statistics` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xEn` | `BOOL` | TRUE: FB enabled<br>FALSE: FB Disabled |
| `i_rIput` | `REAL` | Bit position<br>Range: ±3.4e$^{+38}$ |

## Output Pin Description

This table describes the output pins of the `Signal_Statistics` function block:

| output | Data Type | Description |
|---|---|---|
| q_xActv | BOOL | FB status output |
| q_rMin | REAL | Minimum value<br>Range: ±3.4e+38 |
| q_rMax | REAL | Maximum value<br>Range: ±3.4e+38 |
| q_rAvge | REAL | Average value<br>Range: ±3.4e+38 |
| q_rVrnc | REAL | Variance value<br>Range: ±3.4e+38 |

# Check_Divisor: Checking for Zero Division Condition

## What's in This Chapter



## Overview

This chapter describes the Check_Divisor function block.

# Check_Divisor Function Block

## Pin Diagram

This figure shows the pin diagram of the Check_Divisor function block:


## Functional Description

The Check_Divisor function block checks for zero division condition.

If the divisor i_rDvsr is zero, then the output q_rChkDiv is 1, else if divisor not equal to 0, then output is equal to divisor.

## Input Pin Description

This table describes the input pins of the Check_Divisor function block:

| Input | Data Type | Description |
|---|---|---|
| i_rDvsr | REAL | Divisor output<br>Range: ±3.4e+38 |

## Output Pin Description

This table describes the output pins of the Check_Divisor function block:

| output | Data Type | Description |
|---|---|---|
| q_rChkDiv | REAL | Check division output<br>Range: ±3.4e+38<br>**NOTE:** This output is not valid at the value 0. |

# Numerical Conversion Functions

## What's in This Part



## Overview

This part describes the Numerical Conversion family.

# ArrayOfByte_TO_String: Converting Array in Byte Format to String Format

## What's in This Chapter



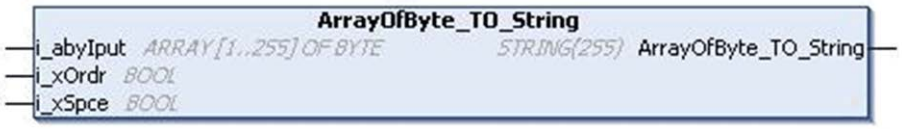
## Overview

This chapter describes the `ArrayOfByte_TO_String` function.

# `ArrayOfByte_TO_String` Function

## Pin Diagram

This figure shows the pin diagram of the `ArrayOfByte_TO_String` function:




## Functional Description

The output String [255] is the set of string characters which corresponds to the ASCII value of input array given in Byte format.

If Order input is TRUE, then the order of characters in the output string corresponds to the order of bytes at the input array. This means there is a 1:1 correspondence between order of input bytes and order of string characters returned at output as explained in example 1.

If the Order input is FALSE, then the order of characters in the output string will be such that, the string character corresponding to ASCII value at input[1] is displayed in output position[2] of output[1..255]. The string character corresponding to ASCII value at input[2] will be displayed in output position[1] of output[1..255].

Similarly the string character corresponding to ASCII value at input[3] will be displayed in output position[4] of output[1..255]. The string character corresponding to ASCII value at input[4] will be displayed in output position[3] of output[1..255] as explained in example 2.

Only if Order input is FALSE and Space input is TRUE and the number of input bytes at the input is odd, then a space character will be added prior to the last string character of the output string as shown in example 3 and 4.

But if the Order input is TRUE, then the Space input has no impact on the output as shown in example 6.

## Example 1

Input: ARRAY [1..255] OF BYTE = 72, 69, 76, 76, 79;

Order: TRUE

Space: FALSE

String: 'HELLO'

As shown above, the order of characters in the output string corresponds to the order of input bytes, that is the byte value at the first position of Array[1..255] is 72 which corresponds to the string value at the first position of the output which is H. The byte value at second position of Array[1..255] is 69 which corresponds to string value at second position of the output which is E and so on.

## Example 2

Input: ARRAY [1..255] OF BYTE = 65, 66, 67, 68, 69, 70, 71;

ByteOrder: FALSE

InsertSpace: FALSE

String: ‘BADCFEG’

As shown above, the order of characters in the output string is changed, that is the byte value at first position of Array [1...255] is 65 which corresponds to the string value at second position of the output which is A. The byte value at second position of Array[1..255] is 66 which corresponds to the string value at first position of the output which is B. Similarly the byte value at third position of Array[1..255] is 67 which corresponds to the string value at fourth position of the output which is C. The byte value at fourth position of Array[1..255] is 68 corresponds to the string value at third position of the output which is D and so on.

## Example 3

Input: ARRAY [1..255] OF BYTE = 72, 69, 76, 76, 79;

Order: FALSE

Space: TRUE

String: ‘EHLL O’

## Example 4

Input: ARRAY [1..255] OF BYTE = 65, 66, 67, 68, 69, 70, 71;

Order: FALSE

Space: TRUE

String: ‘BADCFE G’

As shown above in examples 3 & 4, the number of inputs is 5 in example 3 and 7 in example 4. Since 5 and 7 are odd numbers, the Order input is FALSE and Space input is TRUE. Hence the string outputs are 'EHLL O' AND 'BADCFE G' respectively.

**NOTE:** However if the number of bytes at the input are 255, Order input is FALSE and Space input is TRUE. The Space input becomes insignificant as explained in example 5 below.

## Example 5

Input: ARRAY [1..250] OF BYTE = 65 and ARRAY [251..255] OF BYTE = 66, 67, 68, 69, 70;

Order: TRUE

Space: TRUE/FALSE

String[1..250]: 'A' and String[251..255] = ‘CBEDF’

As shown in the above example, if the number of bytes at the input is 255, the string output remains unaffected by the Space input

## Example 6

Input: ARRAY [1..255] OF BYTE = 65, 66, 67, 68, 69, 70, 71;

Order: TRUE

Space: TRUE

String: ‘ABCDEFG’

As shown above, if the Order input is TRUE, the space input becomes insignificant.

## Input Pin Description

This table describes the input pins of the `ArrayOfByte_TO_String` function:

| Input | Data Type | Description |
|---|---|---|
| `i_abyIput` | `ARRAY [1..255] OF BYTE` | Input value<br>Range: 1...255 |
| `i_xOrdr` | `BOOL` | TRUE: In order of input<br>FALSE: Swaps higher and lower byte |
| `i_xSpce` | `BOOL` | TRUE: Inserts space when `i_xOrdr` is low<br>FALSE: No space is inserted. |

**NOTE:** `I_xSpce` will insert a space character just before the last output string character when the Order input is FALSE and the number of input bytes is odd.

## Output Pin Description

This table describes the output pins of the `ArrayOfByte_TO_String` function:

| Output | Data Type | Description |
|---|---|---|
| `ArrayOfByte_TO_String` | `STRING(255)` | Output of string characters |

**NOTE:** It is mandatory for the user to define the size of the string output as [255], else the size is taken as 80 by default.

## Instantiation and Usage Example

This figure shows an instance of the `ArrayOfByte_TO_String` function:

## With Order Input and Without Space Input

If input is:

- `i_abyIput` [255]
  - Input [1] = 65
  - Input [2] = 66
  - Input [3] = 67
  - Input [4] = 68
  - Input [5] = 69
- `i_xOrdr`: TRUE
- `i_xSpce`: FALSE

The `ArrayOfByte_TO_String` displays ‘ABCDE’.

## With Order Input and With Space Input

If input is:

- `i_abyIput` [255]
  - Input [1] = 65
  - Input [2] = 66
  - Input [3] = 67
  - Input [4] = 68
  - Input [5] = 69
- `i_xOrdr`: FALSE
- `i_xSpce`: TRUE

The `ArrayOfByte_TO_String` displays ‘BADC E’

# DT_AS_WORD: Converting Date and Time as Word

## What's in This Chapter



## Overview

This chapter describes the DT_AS_WORD function block.

# DT_AS_WORD Function Block

## Pin Diagram

This figure shows the pin diagram of the DT_AS_WORD function block:


## Functional Description

The DT_AS_WORD function block extracts data from date and converts to equivalent words.

The DATE_AND_TIME input is converted to output in the form of WORD having year, month, date, hour, minute and second separately.

## Example

With the input DT#2008-08-15-11:05:30, outputs are:

- Output year: 2008
- Output month: 8
- Output day: 15
- Output hours: 11
- Output minutes: 5
- Output seconds: 30

## Input Pin Description

This table describes the input pins of the DT_AS_WORD function block:

| Input | Data Type | Description |
|---|---|---|
| i_dtDttm | DT | Date and time input<br>Range: 1970-01-01-00:00:00...<br>2106-02-07-06:28:15 |

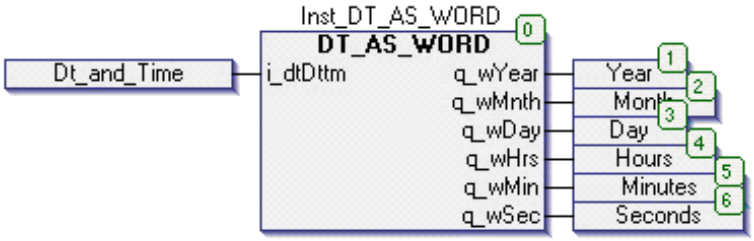
# Output Pin Description

This table describes the output pins of the DT_AS_WORD function block:

| Output | Data Type | Description |
|---|---|---|
| q_wYear | WORD | Year output<br>Range: 1970...2106 |
| q_wMnth | WORD | Month output<br>Range: 1...12 |
| q_wDay | WORD | Date output<br>Range: 1...31 |
| q_wHrs | WORD | Hours output<br>Range: 1...23 |
| q_wMin | WORD | Minutes output<br>Range: 1...59 |
| q_wSec | WORD | Seconds output<br>Range: 1...59 |

# Instantiation and Usage Example

This figure shows an instance of the DT_AS_WORD function block:




Operation of the function block is given in the below example:

- i_dtDttm: DT#2008-08-15-11:05:30
- q_wYear: 2008
- q_wMnth: 8
- q_wDay: 15
- q_wHrs: 11
- q_wMin: 5
- q_wSec: 30

# DWORD_AS_WORD: Splitting the Double Word into Two Words

## What's in This Chapter



## Overview

This chapter describes the DWORD_AS_WORD function block.

# DWORD_AS_WORD Function Block

## Pin Diagram

This figure shows the pin diagram of the DWORD_AS_WORD function block:


## Functional Description

The DWORD_AS_WORD function block converts an input value of data type DWORD to lower and higher outputs of type WORD.

The input double word i_dwIput is split into two words, higher q_wHigh and lower q_wLow words.

## Input Pin Description

This table describes the input pins of the DWORD_AS_WORD function block:

| Input | Data Type | Description |
|---|---|---|
| i_dwIput | DWORD | Input value<br>Range: 0...4294967295 |

## Output Pin Description

This table describes the output pins of the DWORD_AS_WORD function block:

| Output | Data Type | Description |
|---|---|---|
| q_wLow | WORD | Lower word output value<br>Range: 0...65535 |
| q_wHigh | WORD | Higher word output value<br>Range: 0...65535 |

# String_TO_ArrayOfByte: Output Array and ASCII Value of the Input String

### What's in This Chapter



## Overview

This chapter describes the `String_TO_ArrayOfByte` function.

# `String_TO_ArrayOfByte` Function

## Pin Diagram

This figure shows the pin diagram of the `String_TO_ArrayOfByte` function:


## Functional Description

The `String_TO_ArrayOfByte` function is an output Array [255] of bytes is the ASCII value of the input String.

If the Order input is TRUE, then the order of the output values corresponds to the order of the string characters at the input. This means there is a 1:1 correspondence between the order of inputs and the order of ASCII value returned at the output as explained in example 1.

If the Order input is False, then the output is such that the ASCII value of string character at input[1] of input array[1..255] is displayed in position 2 of the output. The ASCII value of string character at input[2] of input array[1..255] is displayed in position 1 of the output. Similarly the ASCII value of string character at input[3] of input array[1..255] is displayed in position 4 of the output and the ASCII value of string character at input[4] of input array[1..255] is displayed in position 3 of the output as explained in example 2.

## Example 1

If the Order input is TRUE then only output array is displayed in the order of string input as shown:

`i_sIput`='ABCDE'

`i_xOrdr`= TRUE

Then the string to array of byte output is:

- output [1] = 65
- output [2] = 66
- output [3] = 67
- output [4] = 68
- output [5] = 69

- output [6] = 0

As shown in the above example, Input [1] = A, its corresponding ASCII code is 65, displayed in output [1] position.

Similarly input [2] = B, its corresponding ASCII code is 66, displayed in output [2] position and so on.

## Example 2

`i_sIput`='ABCDE'

`i_xOrdr`= FALSE

Then the string to array of byte output is:

- output [1] = 66
- output [2] = 65
- output [3] = 68
- output [4] = 67
- output [6] = 0
- output [5] = 69

As shown in the above example,

Input [1] = A, its corresponding ASCII code is 65, displayed in output [2] position.

Input[2] = B, and its corresponding ASCII code is 66, displayed in output [1] position.

Similarly input [3] = C, its corresponding ASCII code is 67, displayed in output [4] position.

Input [4] = D, its corresponding ASCII code is 68, displayed in output [3] position.

Similarly Input [5] = E, its corresponding ASCII code is 69, displayed in output [6] position.

Input [6] = (space), its corresponding ASCII code is " " (that is one blank space character) is displayed in output [5] position.

> **NOTE:** However if the number of bytes at the input is 255, Order input is FALSE. Then the last ASCII value remains in the same position (refer example 3 below).

Input:

- `i_sIput` [1...250]='A'
- `i_sIput` [251...255]='BCDEF'

Order: FALSE

Output

- Output [1...250]:='65'
- Output [251...255]='CBEDF'

## Input Pin Description

This table describes the input pins of the `String_TO_ArrayOfByte` function:

| Input | Data Type | Description |
|---|---|---|
| `i_sIput` | `STRING [1...255]` | Input string value (1...255) |
| `i_xOrdr` | `BOOL` | TRUE: Output in order of input<br>FALSE: Output swaps higher and lower byte. |

**NOTE:** It is mandatory for the user to define the size of the string input[255], else the size is taken as 80 by default.

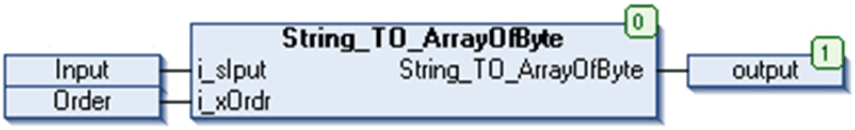
## Output Pin Description

This table describes the output pins of the `String_TO_ArrayOfByte` function:

| Output | Data Type | Description |
|---|---|---|
| String_TO_Array OfByte | `ARRAY [0...255] OF BYTE` | Array of ASCII values<br>Range: 0...255 |

## Instantiation and Usage Example

This figure shows an instance of the the `String_TO_ArrayOfByte` function:




## With Order Input

`i_sIput` [255]:

- Input [1] = A
- Input [2] = B
- Input [3] = C
- Input [4] = D
- Input [5] = E

`i_xOrdr`: TRUE

The `String_TO_ArrayOfByte` displays '65, 66, 67, 68, 69'

## Without Order Input

`i_sIput` [255]:

- Input [1] = A
- Input [2] = B
- Input [3] = C
- Input [4] = D
- Input [5] = E

`i_xOrdr`: FALSE

The `String_TO_ArrayOfByte` displays '66, 65, 68, 67, 69'

# WORD_AS_DWORD: Shifting Higher Word and Adding Lower Word

## What's in This Chapter



## Overview

This chapter describes the WORD_AS_DWORD function block.

# WORD_AS_DWORD Function Block

## Pin Diagram

This figure shows the pin diagram of the WORD_AS_DWORD function block:


## Functional Description

The WORD_AS_DWORD function block merges two input values of data type WORD into a single output of type DWORD.

The higher word input i_wHigh is shifted to the left by 4 nibbles and adds the lower word input i_wLow to obtain a Dword output q_dwOput.

## Input Pin Description

This table describes the input pins of the WORD_AS_DWORD function block:

| Input | Data Type | Description |
|---|---|---|
| i_wLow | WORD | Lower word input value<br>Range: 0...65535 |
| i_wHigh | WORD | Higher word input value<br>Range: 0...65535 |

## Output Pin Description

This table describes the output pins of the WORD_AS_DWORD function block:

| Output | Data Type | Description |
|---|---|---|
| q_dwOput | DWORD | Output Dword value<br>Range: 0...4294967295 |

# Physical Conversion

## What's in This Part



## Overview

This part describes the Physical Conversion family.

# Celsius_TO_Fahrenheit: Converting Celsius to Fahrenheit

### What's in This Chapter



## Overview

This chapter describes the Celsius_TO_Fahrenheit function block.

# Celsius_TO_Fahrenheit Function

## Pin Diagram

This figure shows the pin diagram of the Celsius_TO_Fahrenheit function:


## Functional Description

The Celsius_TO_Fahrenheit function converts temperature in Celsius to Fahrenheit.

Use Fahrenheit_TO_Celsius for the reverse process.

Formula: T_Fahrenheit = [(T_Celcius * 1.8) + 32]

## Input Pin Description

This table describes the input pins of the Celsius_TO_Fahrenheit function:

| Input | Data Type | Description |
|---|---|---|
| i_rIput | REAL | Input value in Celsius<br>Range: ±1.89e38 (higher values result INFINITY at the output). |

## Output Pin Description

This table describes the output pins of the Celsius_TO_Fahrenheit function:

| Output | Data Type | Description |
|---|---|---|
| Celsius_TO_Fahrenheit | REAL | Output value in Fahrenheit |

# Celsius_TO_Kelvin: Converting Celsius to Kelvin

## What's in This Chapter



## Overview

This chapter describes the Celsius_TO_Kelvin function.

# Celsius_TO_Kelvin Function Block

## Pin Diagram

This figure shows the pin diagram of the Celsius_TO_Kelvin function block:


## Functional Description

The Celsius_TO_Kelvin function block converts the value of Celsius unit of REAL type to Kelvin unit. This result will be a REAL number.

The i_rIput pin is used to enter Celsius.

The q_rOput pin returns the equivalent Kelvin value in REAL data type.

Formula: Kelvin = Celsius + 273.15

## Input Detected Error

The q_xErr pin becomes TRUE if invalid Celsius value is entered in pin i_rIput and the pin q_rOput returns to 0 as the temperature in Kelvin unit cannot be less than 0.

This detected error pin is reset on valid input entry.

## Input Pin Description

This table describes the input pins of the Celsius_TO_Kelvin function block:

| Input | Data Type | Description |
|---|---|---|
| i_rIput | REAL | Input value in Celsius<br>Range: -273.15...3.4e$^{+38}$ |

## Output Pin Description

This table describes the output pins of the Celsius_TO_Kelvin function block:

| Output | Data Type | Description |
|---|---|---|
| q_xErr | BOOL | TRUE: Invalid input<br>FALSE: Valid input |
| q_rOput | REAL | Output value in Kelvin<br>Range: 0...3.4e+38 |

The i_rIput input cannot be set to less than -273.15 because the equivalent Kelvin value is less than 0 which is theoretically not possible.

# Fahrenheit_TO_Celsius: Converting Fahrenheit to Celsius

### What's in This Chapter



## Overview

This chapter describes the Fahrenheit_TO_Celsius function block.

## Fahrenheit_TO_Celsius Function

## Pin Diagram

This figure shows the pin diagram of the Fahrenheit_TO_Celsius function:


## Functional Description

The Fahrenheit_TO_Celsius function converts temperature in Fahrenheit to Celsius.

Use Celsius_TO_Fahrenheit for the reverse process.

Formula: T_Celcius = [(T_Fahrenheit - 32) / 1.8]

## Input Pin Description

This table describes the input pins of the Fahrenheit_TO_Celsius function:

| Input | Data Type | Description |
|---|---|---|
| i_rIput | REAL | Input value in Fahrenheit<br>Range: ±3.4e$^{+38}$ |

## Output Pin Description

This table describes the output of the Fahrenheit_TO_Celsius function:

| Output | Data Type | Description |
|---|---|---|
| Fahrenheit_TO_Celcius | REAL | Output value in Celsius<br>Range: ±1.89e$^{+38}$ |

# Frequency_TO_Period: Calculating Period of Time of Given Frequency

## What's in This Chapter



## Overview

This chapter describes the `Frequency_TO_Period` function.

# `Frequency_TO_Period` Function

## Pin Diagram

This figure shows the pin diagram of the `Frequency_TO_Period` function:


## Functional Description

The `Frequency_TO_Period` function converts frequency (Hertz) value of type `REAL` to time. This result is of `TIME` type.

The period of time is calculated with the given frequency. The frequency is set in `i_rIput` pin in `REAL` data format. The equivalent time value is returned in `Frequency_TO_Period` pin in `TIME` data format.

Period = 1 / Frequency

**NOTE:** If the input is not in the previous range, the output is zero.

## Input Pin Description

This table describes the input pins of the `Frequency_TO_Period` function:

| Input | Data Type | Description |
|---|---|---|
| `i_rIput` | `REAL` | Input Frequency 0.0...1000.0Hz |

## Output Pin Description

This table describes the output pins of the `Frequency_TO_Period` function:

| Output | Data Type | Description |
|---|---|---|
| `Frequency_TO_Period` | `TIME` | Period of time of the frequency input. 0...4294967295 ms |

# Kelvin_TO_Celsius: Converting Kelvin to Celsius

## What's in This Chapter



## Overview

This chapter describes the `Kelvin_TO_Celsius` function block.

# `Kelvin_TO_Celsius` Function Block

## Pin Diagram

This figure shows the pin diagram of the `Kelvin_TO_Celsius` function block:


## Functional Description

The `Kelvin_TO_Celsius` function block converts the value in Kelvin unit of type `REAL` to Celsius unit. The result is a `REAL` number.

The pin `i_rIput` is used to enter Kelvin.

The pin `q_rOput` returns the equivalent Celsius value in `REAL` data type.

Formula: Celsius = Kelvin – 273.15

## Input Detected Error

The `q_xErr` pin of type `BOOL` becomes TRUE if the invalid Kelvin value (ie < 0) is entered in the pin `i_rIput` and the pin `q_rOput` returns –273.15, because the equivalent Celsius value for the minimum Kelvin value is –273.15.

This detected error pin is reset on valid input entry.

## Input Pin Description

This table describes the input pins of the `Kelvin_TO_Celsius` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_rIput` | `REAL` | Input value in Kelvin<br>Range: 0...3.4e$^{+38}$ |

## Output Pin Description

This table describes the output pins of the `Kelvin_TO_Celsius` function block:

| Output | Data Type | Description |
|---|---|---|
| q_xErr | BOOL | TRUE: Invalid input.<br>FALSE: Valid input. |
| q_rOput | REAL | Output value in Celsius<br>Range: -273.15...3.4e+38 |

# Limitations

The i_rIput input cannot be set to less than 0 because, theoretically Kelvin value cannot be less than 0.

# Period_TO_Frequency: Calculating Frequency of Given Time

## What's in This Chapter



## Overview

This chapter describes the Period_TO_Frequency function.

# Period_TO_Frequency Function

## Pin Diagram

This figure shows the pin diagram of the Period_TO_Frequency function:


## Functional Description

The Period_TO_Frequency function converts time of type TIME to frequency (Hertz).This result is a REAL number.

This function calculates the frequency of given period of time. Time is set in i_rIput pin in TIME data format. The equivalent frequency value is returned in Period_TO_Frequency pin in REAL data format.

Frequency = 1 / Period

## Input Pin Description

This table describes the input pins of the Period_TO_Frequency function:

| Input | Data Type | Description |
|---|---|---|
| i_tIput | TIME | Input time value<br>Range: 0...4294967295 ms |

**NOTE:** If the input is not in the previous range, the output will be zero.

## Output Pin Description

This table describes the output pins of the Period_TO_Frequency function:

| Output | Data Type | Description |
|---|---|---|
| Period_TO_Frequency | REAL | Equivalent frequency of the time input<br>Range: 0...1000 Hz |

# Utilities

## What's in This Part



## Overview

This part describes the utilities function block family.

# Hour_Meter: Accumulating Operating Hours

## What's in This Chapter



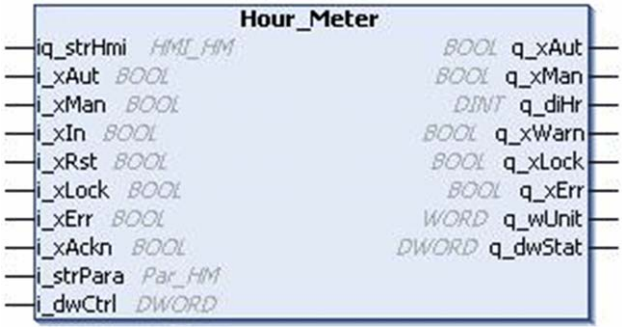
## Overview

This chapter describes the `Hour_Meter` function block.

# `Hour_Meter` Function Block

## Pin Diagram

This figure shows the pin diagram of the `Hour_Meter` function block:




## Functional Description

The `Hour_Meter` function block is used for accumulating the operating hours of various devices.

## Limitations

- If an error is detected, time is not calculated for the period till the detected error is acknowledged after which time calculation resumes from the value at the arrival of detected error. Hence, if the device for which UP time is being calculated is active during the error detected state, then the accumulated time is lesser than the actual time.
- For a wrong value of time unit specified in the variable `i_strPara.wTypeTime`, the function block displays the previous value of calculated time when the time unit input is correct, but the function block does not inform the user about this erroneous condition.
- Even if the block is locked by the external Interlock input, the function block can receive, generate and display a detected error.
- In Locked mode, the function block can reset the detected error by receiving the acknowledgement input.

- The calculated time value at the output is only in seconds or minutes or hours. The user should convert it into a format like HH:MM:SS if required.
- The structure variable `i_strPara` holds the warning time value and the time unit. The user must take special precaution not to accidentally change the time units as this can result in false alarm generation.

## Operation Modes

The accumulation can take place either in manual mode or automatic mode:

- **Automatic Mode:** The automatic mode is selected through the input pin `i_xAut`. When the input `i_xIn` is set to TRUE the block accumulates the time and stops when the `i_xIn` is set to FALSE. The accumulated time is available at the output pin `q_diHr`.
- **Manual Mode:** The Manual mode is activated by the pin `i_xMan`. When the input `i_xIn` is set to TRUE the block accumulates the time and stops when the `i_xIn` is set to FALSE.The accumulated time is available at the output pin `q_diHr`. The accumulation can be inhibited by the command bit `i_dwCtrl`.

The block is de-activated on controller start and remains in the specified mode until a new one is selected. If both inputs are set to 1, then the operation mode is invalid.

## Resetting Value

The reset of the output `q_diHr` is executed by a rising edge at the input `i_xRst` in automatic mode or by a command bit in manual mode.

The reset value of the output `q_diHr` is set to the value `i_strPara.diSp` (Set Point). Additionally the detected signal `q_xWarn` is set, when the output `q_diHr` exceeds the detected error limit specified by the parameter `i_strPara.diWaitTime`.

## Setting Output Value Type

The parameter `i_strPara.wTypeTime` sets the unit of the output value. The selectable are seconds, minutes and hours. The accumulation function does not depend on this value, but is always done on the base of seconds.

## Running Conditions

The counting takes place only, if the interlock input `i_xLock` is set to FALSE. An active interlock signal inhibits the operation of the hour meter. An active interlock is indicated at the output `q_xLock`.

The function block sets the detected error signal, if the detected error input `i_xErr` is set to TRUE (external detected error) or if the operation mode is invalid (internal detected errors). The detected errors are indicated in the HMI. To reset the detected error output the detected error has to be acknowledged by a rising edge on the input `i_xAckn` or by using bit 16 of the signal `i_dwCtrl`.

# Input Pin Description

## Input Pin Description

This table describes the input pins of the `Hour_Meter` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xAut` | `BOOL` | TRUE: Auto mode enabled<br>FALSE: Disabled (factory setting) |
| `i_xMan` | `BOOL` | TRUE: Manual mode enabled<br>FALSE: Disabled (factory setting) |
| `i_xIn` | `BOOL` | TRUE: Block accumulates the time<br>FALSE: Disabled (factory setting) |
| `i_xRst` | `BOOL` | TRUE: Sets the output back to the preset value entered at input `i_strPara`.<br>FALSE: Disabled (factory setting)<br>(Optional) |
| `i_xLock` | `BOOL` | Interlock input for operation<br>TRUE: Interlock is active<br>FALSE: No Interlock. (factory setting)<br>(Optional) |
| `i_xErr` | `BOOL` | TRUE: External detected error is active<br>FALSE: No external detected error (factory setting) |
| `i_xAckn` | `BOOL` | Acknowledgement is done with a rising edge.<br>Input to acknowledge internal and external detected errors indicated at the output `q_xErr`. |
| `i_strPara` | `STRUCT Par_HM` | Structure with parameters for this block.<br>Refer to the , page 177`Par_HM` description. |
| `i_dwCtrl` | `DWORD` | Command bits to interact from the HMI<br>Range: 0...4294967295<br>Refer to the control word bit description, page 178. |

# Output Pin Description

## Output Pin Description

This table describes the output pins of the `Hour_Meter` function block:

| Output | Data Type | Description |
|---|---|---|
| `q_xAut` | `BOOL` | TRUE: Automatic mode enabled<br>FALSE: Disabled |
| `q_xMan` | `BOOL` | TRUE: Manual mode enabled<br>FALSE: Disabled |
| `q_diHr` | `DINT` | Accumulated operating time in seconds, minutes or hours.<br>Range: -2147483648...2147483647 |

| Output | Data Type | Description |
|---|---|---|
| q_xWarn | BOOL | TRUE: Signal that the time at hour has exceeded a warn time value<br>FALSE: Disabled. |
| q_xLock | BOOL | TRUE: Interlock is active.<br>FALSE: No interlock<br>Indicates that the operation is blocked by an interlock (input i_xLock) |
| q_xErr | BOOL | TRUE: detected error is active<br>FALSE: No detected error |
| q_wUnit | WORD | Indicates the unit of the displayed operating time at the output hour:<br>• 16#01: Seconds<br>• 16#02: Minutes<br>• 16#04: Hours |
| q_dwStat | DWORD | Status bits to be displayed in the HMI<br>Range: 0...4294967295 |

# Input – Output Pin

## Input - Output from HMI

This table describes the Input – Output pin of the Hour_Meter function block:

| Input - Output | Data Type | Description |
|---|---|---|
| iq_strHmi | STRUCT HMI_HM | Structure to interface with the HMI<br>Refer to used structures, page 177. |

# Structures Used

## *Par_HM*

| Structure Element | Type | Description |
|---|---|---|
| *diSp* | DINT | Preset value for the hour meter time output used on reset |
| *diWaitTime* | DINT | Time value for activating the detected error signal |
| *wTypeTime* | WORD | Set the accumulated time at the output in seconds, minutes or hours:<br>• 16#01: Seconds<br>• 16#02: Minutes<br>• 16#04: Hours |

## *HMI_HM*

| Structure Element | Type | Description |
|---|---|---|
| *diVal* | DINT | Accumulated operating time in seconds, minutes or hours. |
| *diSp* | DINT | Preset value for the hour meter time output used on reset |
| *diWaitTime* | DINT | Time value for activating the detected error signal |
| *wTypeTime* | WORD | Indicating the unit of the displayed operating time at the output Hour |

# Control Word Bit Description

## Functionality

This table describes the control word bits:

| Bit Position | Description |
|---|---|
| 0...3 | Not used |
| 4 | Reset the output `q_diHr` to value at `i_strPara.diSp` |
| 5 | Pause time counting |
| 6...15 | Not used |
| 16 | Change in this bit will acknowledge detected error. |
| 17...31 | Not used |

# Status Word

## Functionality

This table describes the status word bits:

| Bit Position | Description |
|---|---|
| 0 | Auto mode is active |
| 1 | Manual mode is active |
| 2 | No mode is selected |
| 3 | Function block is locked by interlock input `i_xLock` |
| 4 | Not used |
| 5 | Time counting is paused |
| 6 | Operational time at `i_strPara.diWaitTime` is exceeded. |
| 7...15 | Not used |
| 16 | Detected error is reset |
| 17 | Detected error is present |
| 18...23 | Not used |
| 24 | Internal detected error is present |
| 25 | External detected error is present |
| 26...31 | Not used |

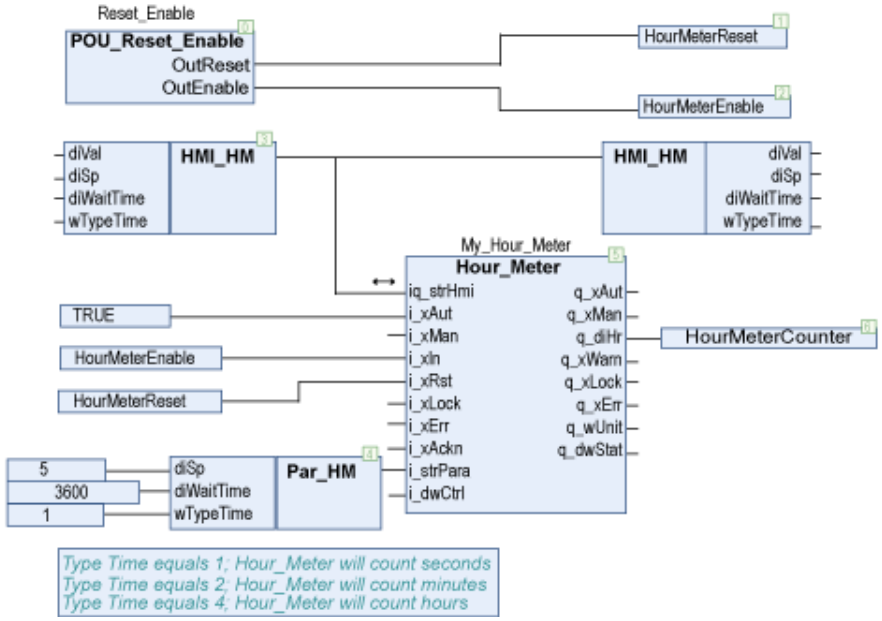
# Instantiation and Usage Example

## Instantiation and Usage Example

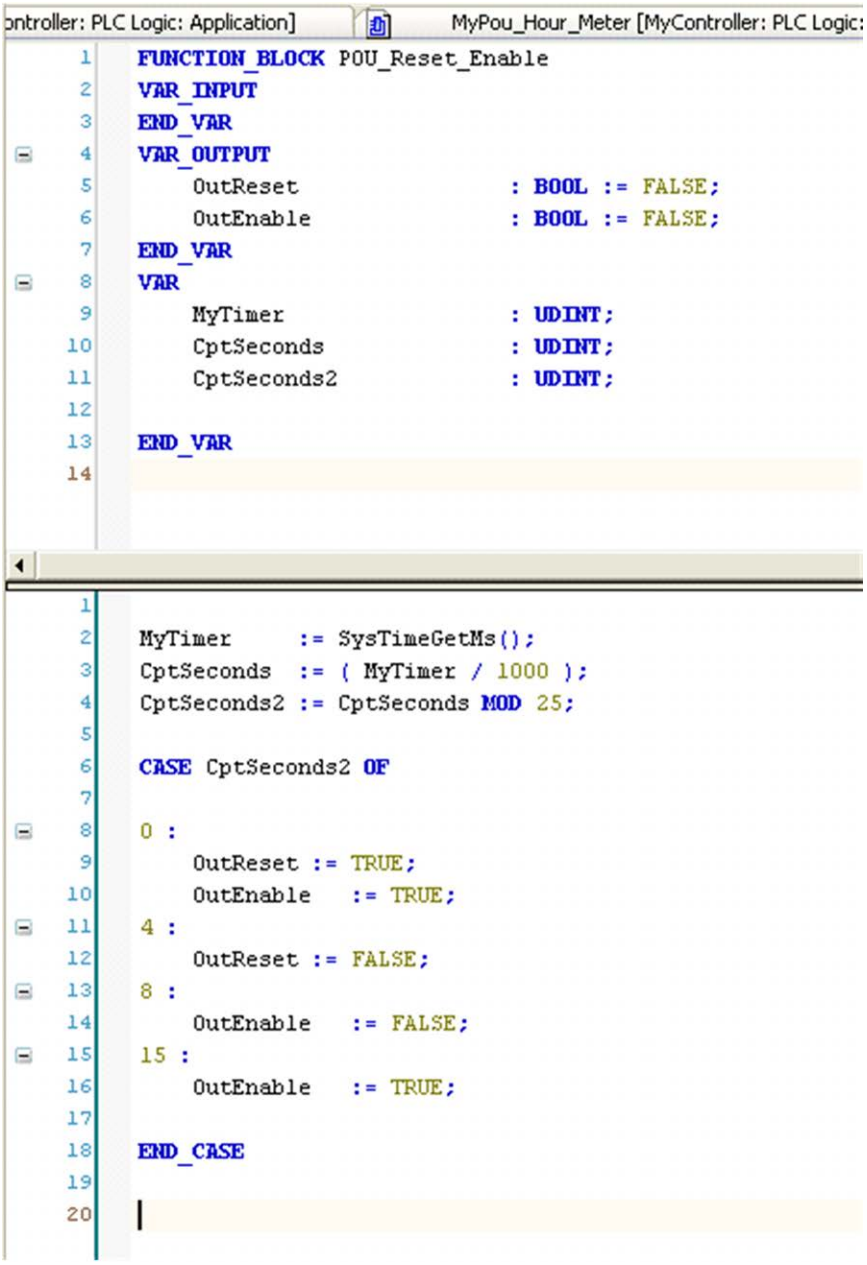
The following program periodically resets and enables the input of `Hour_Meter` function block:

Trace_Hour_Meter [MyController: PLC...] MyPou_Hour_Meter [MyController: PLC Logic: Application]

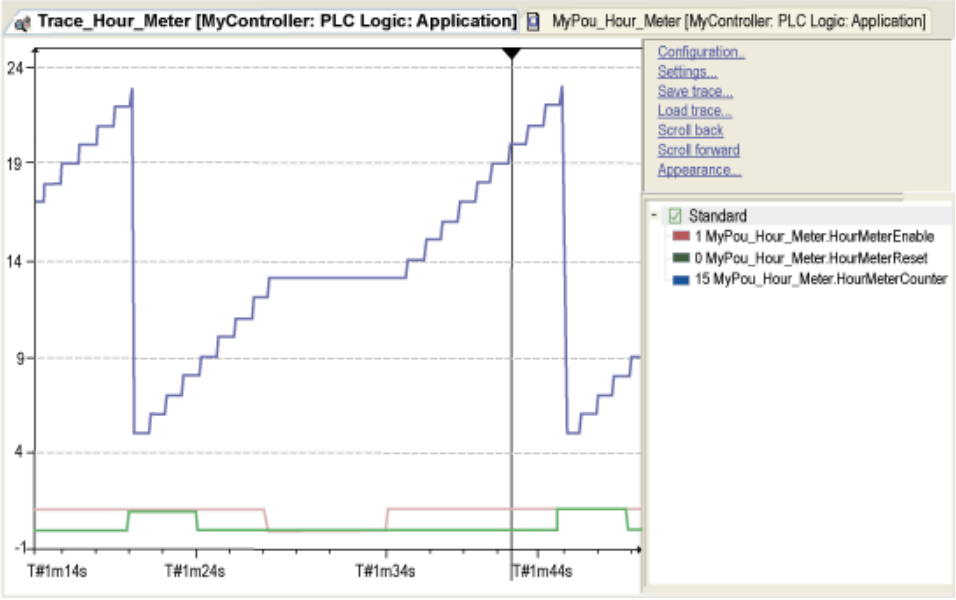
```
PROGRAM MyPou_Hour_Meter
VAR
    FlagStart           : BOOL;
    Reset_Enable        : POU_Reset_Enable;
    My_Hour_Meter       : Hour_Meter;
    HourMeterEnable     : BOOL;
    HourMeterReset      : BOOL;
    HourMeterCounter    : DINT;
END_VAR
```



`HourMeterReset` and `HourMeterEnable` are managed with the following POU:



Every 25 seconds, the data `HourMeterCounter` is reset with an initial value of 5. When `HourMeterReset` is FALSE, the counter holds its current value.




**Blue** `HourMeterCounter`

**Green** `HourMeterReset`

**Red** `HourMeterEnable`

In this sample, the cycle time of the POU in the MAST has no impact. For this sample, the periodicity is 100 milliseconds.

# Operation_Mode: Selecting Operating Mode

## What's in This Chapter



## Overview

This chapter describes the Operation_Mode function block.

# Operation_Mode Function Block

## Pin Diagram

This figure shows the pin diagram of the Operation_Mode function block:


## Functional Description

The Operation_Mode function block is used for selecting between auto and manual operating modes from two different sources:

- Switches / controller program
- HMI

## Limitations

- If no operation mode is currently selected, then the previous active mode persists. For example, if auto mode with local control was set previously, then upon resetting the auto input persists till another mode is set.
- The local mode has higher priority than the HMI control. A switch in operating mode does not take place automatically when local mode is reset and HMI control is previously set along with local mode.

## Local Mode

The local mode can be activated with auto or manual mode. The local mode is activated by using the input i_xLoc and prohibits manual interaction from the HMI via control word input i_dwCtrl.

With local mode active, the operating mode can be activated by using the inputs i_xAut and i_xMan:

- If `i_xAut` is set the automatic mode is activated and indicated at the output `q_xAut`.
- If `i_xMan` is set the manual mode is activated and indicated at the output `q_xMan`.

If local mode is not active, then by setting the bits of `i_dwCtrl` the operating modes can also be activated from the HMI.

**NOTE:** When `q_xHmiCtrl` is set, the inputs `i_xAut` and `i_xMan` are ignored

## Priority

The `i_xLoc` has a higher priority than the `i_dwCtrl` command word. So that once the `i_xLoc` is set, the operating mode is again activated by the inputs `i_xAut` and `i_xMan`.

## Resetting a Detected Error

The block generates an invalid operation mode, if both auto and manual modes are selected (internal detected error) and displays at output `q_xErr`. It also sets the detected error signal, if the detected error input `i_xErr` is set to 1 (external detected error). Detected errors are indicated in the bits of status word `q_dwStat`. To reset the detected error output the detected error has to be acknowledged by a rising edge on the **input Ack** or by using the acknowledgement bit of the input `i_dwCtrl`.

# Input Pin Description

## Input Pin Description

This table describes the input pins of the `Operation_Mode` function block:

| Input | Data Type | Description |
|---|---|---|
| `i_xAut` | `BOOL` | TRUE: Automatic mode enabled<br>FALSE: Disabled (factory setting) |
| `i_xMan` | `BOOL` | TRUE: Manual mode enabled.<br>FALSE: Disabled. (factory setting) |
| `i_xLoc` | `BOOL` | TRUE: Local mode enabled<br>FALSE: Disabled. (factory setting) |
| `i_xErr` | `BOOL` | TRUE: External detected error is active.<br>FALSE: No external detected error. (factory setting) |
| `i_xAckn` | `BOOL` | Acknowledgement is done with a rising edge.<br>Input to acknowledge internal and external detected errors indicated at the output `q_xErr`. |
| `i_dwCtrl` | `DWORD` | Command bits to interact from the HMI<br>Range: 0...4294967295<br>Refer to the status word description, page 184 |

# Output Pin Description

## Output Pin Description

### Output Pin Description

This table describes the output pins of the `Operation_Mode` function block:

| Output | Data Type | Description |
|---|---|---|
| `q_xAut` | `BOOL` | TRUE: Automatic mode enabled<br>FALSE: Disabled |
| `q_xMan` | `BOOL` | TRUE: Manual mode enabled<br>FALSE: Disabled |
| `q_xLoc` | `BOOL` | TRUE: Local mode enabled<br>The operating mode is not changeable by the HMI and function blocks cannot be operated from the HMI.<br>FALSE: Disabled |
| `q_xHmiCtrl` | `BOOL` | TRUE: Operating mode is given by the HMI.<br>`q_xHmiCtrl` is overwritten by local mode.<br>FALSE: Disabled |
| `q_xErr` | `BOOL` | TRUE: Detected error is active.<br>FALSE: No detected error |
| `q_dwStat` | `DWORD` | Status bits to be displayed in the HMI<br>Range: 0...4294967295<br>Refer to the status word description, page 185. |

# Control Word Bit Description

## Functionality

This table describes the control word bits:

| Bit Position | Description |
|---|---|
| 0 | Set automatic mode |
| 1 | Set manual mode |
| 2, 3 | Not used |
| 4 | Set HMI Control active. Only after this bit is set, selections via bit 0 and bit 1 will take place |
| 5...15 | Not used |
| 16 | Rising edge of this bit gives acknowledgment to detected error. |
| 17...31 | Not used |

This table shows the truth table:

| i_xAut | i_xMan | i_xLoc | i_dwCtrl | | | q_xAut | q_xMan | q_xLoc | q_xHmi-Ctrl | q_xErr |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | bit0 Aut | bit1 Man | bit4 HMI | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | X | X | X | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | X | X | X | 0 | 0 | 1 | 0 | 1 |

| i_xAut | i_xMan | i_xLoc | i_dwCtrl | | | q_xAut | q_xMan | q_xLoc | q_xHmi-Ctrl | q_xErr |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | bit0 Aut | bit1 Man | bit4 HMI | | | | | |
| 1 | 0 | 1 | X | X | X | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | X | X | X | 0 | 1 | 1 | 0 | 0 |
| X | X | 1 → 0 | X | X | 0 | PS | PS | 0 | 0 | 0 |
| X | X | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| X | X | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| X | X | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| X | X | 0 | X | X | 1 → 0 | PS | PS | 0 | 0 | 0 |
| X | X | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| **PS** Previous State | | | | | | | | | | |

# Status Word

## Functionality

This table describes the status word bits:

| Bit Position | Description |
|---|---|
| 0 | Auto mode is active |
| 1 | Manual mode is active |
| 2 | Local mode is active |
| 3 | Not used |
| 4 | HMI Control mode is active |
| 5...15 | Not used |
| 16 | Indicates reset of detected error |
| 17 | Detected error is present |
| 18...23 | Not used |
| 24 | Internal detected error is present |
| 25 | External detected error is present |
| 26...31 | Not used |

# Valve Control

## What's in This Part



## Overview

This part describes the valve control function library.

# Bistable_Valve: Controlling the Bistable Valves

## What's in This Chapter



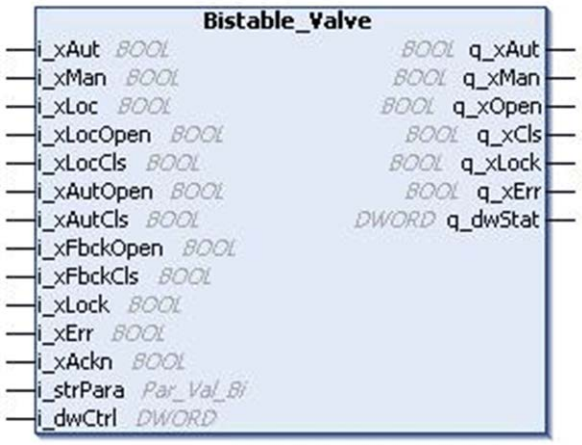
## Overview

This chapter describes the `Bistable_Valve` function block.

# `Bistable_Valve` Function Block

## Pin Diagram

This figure shows the pin diagram of the `Bistable_Valve` function block:




## Functional Description

The `Bistable_Valve` function block is used for controlling the bistable valve.

## Operation Modes

The `Bistable_Valve` function block supports three operating modes:

- **Automatic Mode:** The automatic mode is activated by the input pin `i_xAut`. In this mode, the valve is opened and closed through the inputs `i_xAutOpen` and `i_xAutCls` respectively, regardless of the local mode being activated or not.
- **Manual Mode:** The manual mode is activated by the pin `i_xMan`.

  *Case 1:* Local mode is not activated. The valve is opened and closed through the bit commands of the signal `i_dwCtrl`.

  *Case 2:* Local mode is activated. The valve is opened and closed through the input signals `i_xLocOpen` and `i_xLocCls` respectively.

- **Local Mode:** The local mode is activated by an input pin `i_xLoc` and is set additionally to the automatic or manual mode. The local mode does not influence the automatic mode, but changes the source for manual operation.

## Output Behavior

The output `q_xOpen` remains active as long as the feedback signal `i_xFbckOpen` remains low. Also, the output `q_xCls` remains active as long as the feedback signal `i_xFbckCls` remains low. This output behavior is valid for manual and local modes.

## Controller Start Up Behavior

The block is de-activated on controller start and remains in the same operation mode, unless a new one is selected. If both automatic and manual modes are selected simultaneously (inputs `i_xAut` and `i_xMan` are set to 1), the operation mode is invalid which is indicated at the `q_xErr` output.

## Supervising the Valve

The position of the valve is supervised by the feedback signals `i_xFbckOpen` and `i_xFbckCls`. Once the operation is started, the feedback inputs must signal the right position of the valve within a defined time. If this time exceeds, then the block indicates a detected error. The time can be set through the structure element iFbckDly at input `i_strPara`.

When both open and close feedback signals are missing (`i_xFbckOpen` and `i_xFbckCls` are set to 0), and the position of the valve is unknown (`QOpen_bi` and `QClose_bi` set to 0), an unknown position error is detected.

When the input `xFbckEn` is FALSE, then the time supervising is NOT enabled. Refer to the Output Behavior, page 188.

## Operating the Valve

The valve can be operated only if the input `i_xLock` is set to 0. An active interlock signal inhibits the operation of the valve. An active interlock is indicated by the output `q_xLock`.

The valve can only be operated if the output `q_xErr` is set to 0. An active detected error signal inhibits the operation of the valve.

## Detected Error Management

The output `q_xErr` is high, only if an error is detected. The detected error can be:

- Internal detected error (invalid operation mode, missing feedback signal or unknown position).
- External detected error

The detected errors are indicated in the HMI as alarms. If an interlock or an error is detected during the operation of the valve, the behaviour of the function block depends on the structure element `i_strPara.xFrceEn` at the input `i_strPara`. If this element is set to 1, the block enforces the valve to move in to the default position, and the corresponding output is high (`q_xOpen` or `q_xCls`) for the duration of `i_strPara.iFbckDly` seconds. Otherwise the operation is stopped and has to be restarted after the interlock is gone.

To reset the `q_xErr`, the detected error has to be acknowledged by a rising edge on the input `i_xAckn` or by using bit 16 of the signal `i_dwCtrl`.

## Setting Default Position

The default position of the valve can be set by `i_strPara.xPosDflt`. This description assumes Close as the default. If `i_strPara.xPosDflt` is set to 1, Open is the default position.

# Input Pin Description

## Input Pin Description

This table describes the input pins of the `Bistable_Valve` function block:

| Input | Data Type | Description | Remarks |
|---|---|---|---|
| `i_xAut` | `BOOL` | TRUE: Automatic mode enabled<br>FALSE: Disabled | |
| `i_xMan` | `BOOL` | TRUE: Manual mode enabled<br>FALSE: Disabled | |
| `i_xLoc` | `BOOL` | TRUE: Local mode enabled<br>FALSE: Disabled | |
| `i_xLocOpen` | `BOOL` | Rising edge from 0 to 1 manually opens the valve in local mode | Manual and local mode to be set to 1 simultaneously. |
| `i_xLocCls` | `BOOL` | Rising edge from 0 to 1 manually closes the valve in local mode. | Manual and local mode to be set to 1 simultaneously. |
| `i_xAutOpen` | `BOOL` | Rising edge from 0 to 1 opens the valve in automatic mode. | |
| `i_xAutCls` | `BOOL` | Rising edge from 0 to 1 closes the valve in automatic mode. | |
| `i_xFbckOpen` | `BOOL` | TRUE: Open feedback signal is active.<br>FALSE: No open feedback | |
| `i_xFbckCls` | `BOOL` | TRUE: Close feedback signal is active.<br>FALSE: No close feedback | |
| `i_xLock` | `BOOL` | TRUE: Interlock is active<br>FALSE: No interlock<br>(Optional) | |
| `i_xErr` | `BOOL` | TRUE: External detected error is active.<br>FALSE: No external detected error | |
| `i_xAckn` | `BOOL` | Acknowledgement is done with a rising edge. | Input to acknowledge internal and external detected errors indicated at the output `q_xErr`. |

| Input | Data Type | Description | Remarks |
|---|---|---|---|
| i_strPara | STRUCT Par_Val_Bi | Structure with parameters for this block. | Refer to the used structures, page 190 description. |
| i_dwCtrl | DWORD | Command bits to interact from the HMI<br>Range: 0...4294967295 | Refer to the control word, page 191 description |

# Output Pin Description

## Output Pin Description

This table describes the output pins of the Bistable_Valve function block:

| Output | Data Type | Description | Remarks |
|---|---|---|---|
| q_xAut | BOOL | TRUE: Automatic mode enabled<br>FALSE: Disabled | - |
| q_xMan | BOOL | TRUE: Manual mode enabled<br>FALSE: Disabled | - |
| q_xOpen | BOOL | TRUE: Open command active<br>FALSE: Disabled | - |
| q_xCls | BOOL | TRUE: Close command active<br>FALSE: Disabled | - |
| q_xLock | BOOL | TRUE: Interlock is active<br>FALSE: No interlock | Indicates that the operation is blocked by an interlock (input i_xLock) |
| q_xErr | BOOL | TRUE: Detected error is active<br>FALSE: No detected error) | - |
| q_dwStat | DWORD | Status bits to be displayed in the HMI<br>Range: 0...4294967295 | Refer to the status word description, page 191. |

# Structure Used

## *Par_Val_Bi*

| Structure Element | Type | Description |
|---|---|---|
| *xFbckEn* | BOOL | Enable feedback signal supervision |
| *iFbckDly* | INT | Delay time in seconds to get the feedback signal from the valve. |
| *iRevDly* | INT | Delay time in seconds to operate the valve in opposite direction. |
| *xPosDflt* | BOOL | Default position of the valve<br>(FALSE: closed, TRUE: opened). |
| *xFrceEn* | BOOL | Enable enforcement of default position at interlock or detected error. |

# Control Word Bit Description

## Functionality

This table describes the control word bits:

| Bit Position | Description |
|---|---|
| 0...7 | Not used |
| 8 | Rising edge from 0 to 1 manually opens the valve (output `q_xOpen` set to TRUE) in manual mode. |
| 9 | Rising edge from 0 to 1 manually closes the valve (output `q_xCls` set to TRUE) in manual mode. |
| 10...15 | Not used |
| 16 | Acknowledges internal and external detected errors indicated at the output `q_xErr`. This is rising edge triggered. |
| 17...31 | Not used |

# Status Word

## Functionality

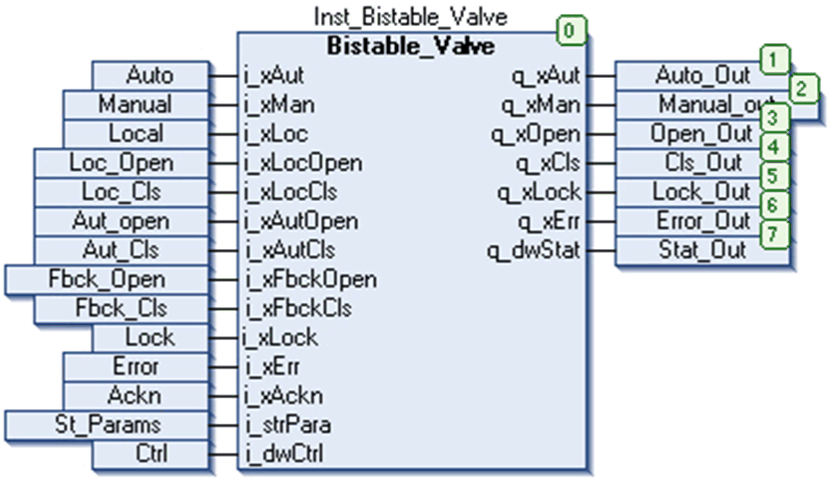
This table describes the status word:

| Bit Position | Description |
|---|---|
| 0 | Auto mode is active. |
| 1 | Manual mode is active. |
| 2 | Local mode is selected. |
| 3 | Indicates that the operation is blocked by an interlock (input `i_xLock`). |
| 4...7 | Not used |
| 8 | Signal to open the valve. |
| 9 | Signal to close the valve. |
| 10...13 | Not used |
| 14 | Feedback signal from opened valve. |
| 15 | Feedback signal from closed valve. |
| 16 | Resets the detected error (`q_xErr`). |
| 17 | Indicates that the operation is blocked by an internal or external (Input `i_xErr`) detected error, which is not acknowledged. |
| 18 | Not used |
| 19 | Enable feedback signal supervision. |
| 20 | Default Position of the valve (FALSE: closed, TRUE: opened). |
| 21...23 | Not used |
| 24 | Invalid Operating mode detected error. |
| 25 | External detected error. |
| 26 | Missing feedback detected error. |
| 27 | Unknown position detected error. |
| 28...31 | Not used |

# Instantiation and Usage Example

## Instantiation and Usage Example

This figure shows an instance of the `Bistable_Valve` function block:




## Limitations

When forced enable is active (`xFrceEn`), the valve is forced in to the default position (`xPosDflt`) only for the time `iFbckDly` seconds. It is reset if an appropriate feedback signal is activated or interlock signal is removed or the `q_xErr` output is acknowledged.

Ensure that appropriate feedback signal is activated before executing the block; else an unknown position error is detected after a time delay of `iFbckDly` seconds.

# Monostable_Valve: Controlling Monostable Valves

## What's in This Chapter



## Overview

This chapter describes the Monostable_Valve function block.

# Monostable_Valve Function Block

## Pin Diagram

This figure shows the pin diagram of the Monostable_Valve function block:


## Functional Description

The Monostable_Valve function block is used for controlling the monostable valve.

## Operation Modes

The Monostable_Valve function block supports three operating modes:

- **Automatic Mode:** The automatic mode is activated by the input pin i_xAut. In this mode the valve is opened and closed through the inputs i_xAutOpen (default position is Closed) and i_xAutCls (default position is Open) respectively, regardless of the local mode being activated or not. The output q_xValvPos remains active as long as the inputs i_xAutOpen/i_xAutCls remain active.

- **Manual Mode:** The manual mode is activated by the pin `i_xMan`.

  *Case 1:* Local mode is not activated. The valve is opened and closed through the bit commands of the signal `i_dwCtrl`.

  *Case 2:* Local mode is activated. The valve is opened and closed through the input signals `i_xLocOpen` and `i_xLocCls` respectively.
- **Local Mode:** Local mode is activated by an input pin `i_xLoc` and is set additionally to automatic or manual mode. Local mode does not influence the automatic mode, but changes the source for manual operation.

**NOTE:** If both automatic and manual modes are selected simultaneously (inputs `i_xAut` and `i_xMan` are set to 1), the operation mode is invalid, which is indicated at the `q_xErr` output.

## Controller Start Up Behavior

The block is de-activated on controller start and remains in the same operation mode, unless a new one is selected.

## Supervising the Valve

The position of the valve is supervised by the feedback signals `i_xFbckOpen` and `i_xFbckCls`. Once the operation is started, the feedback inputs must signal the right position of the valve within a defined time. If this time exceeds, then the block indicates a detected error (missing feedback detected error).The time can be set through the structure element `iFbckDly` at input `i_strPara`. The supervision can be switched Off by the structure element `xFbckEn` at the input `i_strPara`.

## Operating the Valve

The valve can be operated only if the input `i_xLock` is set to 0. An active interlock signal inhibits the operation of the valve and is indicated by the output pin `q_xLock`.

The valve can only be operated if the output `q_xErr` is set to 0. An active detected error signal inhibits the operation of the valve.

## Detected Error Management

The output `q_xErr` is high, only if an error is detected. The detected error can be:

- Internal detected error (invalid operation mode, missing feedback signal or unknown position).
- External detected error

Detected errors are indicated in the HMI as alarms. If an interlock or an error is detected during the operation of the valve, the behaviour of the function block depends on the structure element `i_strPara.xFrceEn` at input `i_strPara`. If this element is set to 1, the block enforces the valve to move in to the default position, and the corresponding output is high (`q_xOpen` or `q_xCls`) for the duration of `i_strPara.iFbckDly` seconds. Otherwise the operation is stopped and has to be restarted after the interlock is gone.

To reset `q_xErr`, the detected error has to be acknowledged by a rising edge on the input `i_xAckn` or by using bit 16 of the signal `i_dwCtrl`.

## Setting Default Position

The default position of the valve can be set by `i_strPara.xPosDflt`. This description assumes Close as default. If `i_strPara.xPosDflt` is set to 1, Open is the default position.

# Input Pin Description

## Input Pin Description

This table describes the input pins of the `Monostable_Valve` function block:

| Input | Data Type | Description | Remarks |
|---|---|---|---|
| `i_xAut` | `BOOL` | TRUE: Automatic mode enabled<br>FALSE: Disabled | - |
| `i_xMan` | `BOOL` | TRUE: Manual mode enabled<br>FALSE: Disabled | - |
| `i_xLoc` | `BOOL` | TRUE: Local mode enabled<br>FALSE: Disabled | - |
| `i_xLocOpen` | `BOOL` | Rising edge from 0 to 1 manually opens the valve in local mode. | Manual and Local Mode to be set to 1 simultaneously. |
| `i_xLocCls` | `BOOL` | Rising edge from 0 to 1 manually closes the valve in local mode. | Manual and Local Mode to be set to 1 simultaneously. |
| `i_xAutOpen` | `BOOL` | TRUE: Valve open command is enabled.<br>FALSE: Disabled | - |
| `i_xAutCls` | `BOOL` | TRUE: Valve close command is enabled.<br>FALSE: Disabled | - |
| `i_xFbckOpen` | `BOOL` | TRUE: Open feedback signal is active.<br>FALSE: No open feedback | - |
| `i_xFbckCls` | `BOOL` | TRUE: Close feedback signal is active.<br>FALSE: No close feedback | - |
| `i_xLock` | `BOOL` | TRUE: Interlock is active<br>FALSE: No interlock | Interlock input for valve operation. Valve operation is inhibited, when the input is set to 1. |
| `i_xErr` | `BOOL` | TRUE: External detected error is active.<br>FALSE: No external detected error | - |
| `i_xAckn` | `BOOL` | Acknowledgement is done with a rising edge. | Input to acknowledge internal and external detected errors indicated at the output `q_xErr`. |

| Input | Data Type | Description | Remarks |
|---|---|---|---|
| i_strPara | STRUCT PAR_Val_Mono | Structure with parameters for this block. | Refer to the used structure, page 196 description. |
| i_wCtrl | WORD | Command bits to interact from the HMI.<br><br>Range: 0...65535 | Refer to the command word description, page 197. |

# Output Pin Description

## Output Pin Description

This table describes the output pins of the Monostable_Valve function block:

| Identifiers | Output Type | Description | Remarks |
|---|---|---|---|
| q_xAut | BOOL | TRUE: Automatic mode enabled<br><br>FALSE: Disabled | - |
| q_xMan | BOOL | TRUE: Manual mode enabled<br><br>FALSE: Disabled | - |
| q_xValvPos | BOOL | TRUE: Open/Close command active<br><br>FALSE: Disabled | Open command active if default position is closed/Close command active if default position is open. |
| q_xLock | BOOL | TRUE: Interlock is active.<br><br>FALSE: No interlock | Indicates that the operation is blocked by an interlock (input i_xLock) |
| q_xErr | BOOL | TRUE: Detected error is active.<br><br>FALSE: No detected error | - |
| q_wStat | WORD | Status bits to be displayed in the HMI<br><br>Range: 0...65535 | Refer to the status word description, page 197. |

# Structure Used

## *PAR_Val_Mono*

| Structure Element | Type | Description |
|---|---|---|
| *xFbckEn* | BOOL | Enable feedback signal supervision |
| *iFbckDly* | INT | Delay time in seconds to get the feedback signal from Valve |
| *xPosDflt* | BOOL | Default position of the valve (0: closed, 1: opened) |

# Control Word Bit Description

## Functionality

This table describes the control word bits:

| Bit Position | Description |
|---|---|
| 0 | Rising edge from 0 to 1 manually opens the valve in manual mode. |
| 1 | Rising edge from 0 to 1 manually closes the valve in manual mode. |
| 2 | Input to acknowledge internal and external detected errors indicated at the output `q_xErr`. Acknowledgement is done with a rising edge |
| 3...15 | Not used |

# Status Word

## Functionality

This table describes the status word:

| Bit position | Description |
|---|---|
| 0 | Auto mode is active. |
| 1 | Manual mode is active. |
| 2 | Local mode is active. |
| 3 | Valve is opened/closed depending upon the default position. |
| 4 | Default position of the Valve. |
| 5 | Interlock signal is active. |
| 6 | Feedback signal from opened valve. |
| 7 | Feedback signal from closed valve. |
| 8 | Detected error is active. |
| 9 | Invalid operating mode detected error. |
| 10 | External detected error. |
| 11 | Missing feedback detected error. |
| 12 | Enable feedback supervision. |
| 13-15 | Not used. |

# Proportional_Valve: Controlling Proportional Valves

## What's in This Chapter



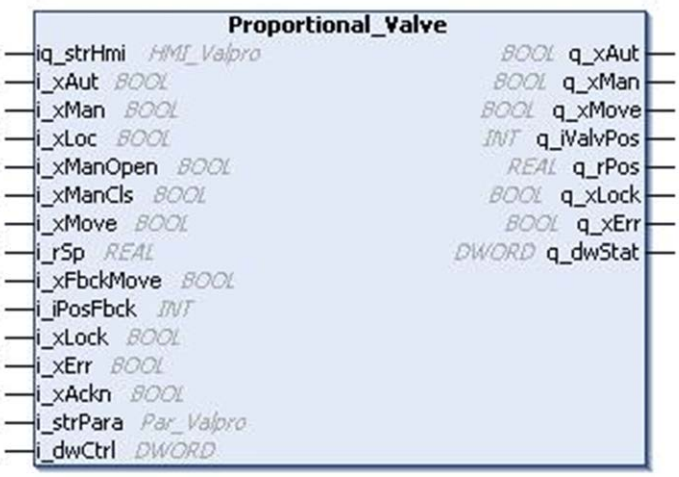
## Overview

This chapter describes the Proportional_Valve function block.

# Proportional_Valve Function Block

## Pin Diagram

This figure shows the pin diagram of the Proportional_Valve function block:




## Functional Description

The Proportional_Valve function block is used for controlling proportional valve.

## Operation Modes

The Proportional_Valve function block supports three operating modes:

- **Automatic Mode:** The automatic mode is activated by the input pin i_xAut. In this mode, the valve is opened and closed via the input i_xMove, regardless if the local mode is activated or not. The new set point is given at the input i_rSp.

- **Manual Mode:** The manual mode is activated by the pin `i_xMan`.

  *Case 1:* Local mode is not activated. The valve is opened and closed through a bit command in the variable `i_dwCtrl` and the value of the set point is given by `rSP` on the input-output `iq_strHmi`

  *Case 2:* Local mode is activated. The valve is operated through the inputs `i_xManOpen` and `i_xManCls`.

- **Local Mode:** The local mode is activated by an input pin `i_xLoc` and is set additionally to the automatic or manual mode. The local mode does not influence the automatic mode, but changes the source for manual operation. The output `q_iValvPos` is automatically set to `i_strPara.rMinSp` when using `i_xManCls` or `i_strPara.rMaxSp` when using `i_xManOpen`.

  **NOTE:** If the operation mode is changed from manual or HMI mode to automatic mode, a movement of the valve is stopped. Any other change of the operation mode does not affect the valve movement, but the setpoint is adjusted to the value of the actual operation mode.

## Output Behavior

The output `q_xMove` remains active as long as the new position given by the setpoint is not reached.

## Setting a Deadband

A deadband can be set by `i_strPara.rBnd`, so that `q_xMove` is switched Off, when the deviation of actual position and setpoint is less than the deadband.

When using the inputs `i_xManOpen` and `i_xManCls` in local mode, the output `q_xMove` is active as long as the inputs are active or if the maximum or minimum position is reached (taking dead band into consideration).

## Supervising the Valve

The position of the valve is supervised by the feedback signals `i_xFbckOpen` and `i_xFbckCls`. Once the operation is started, the feedback inputs must signal the right position of the valve within a defined time. If this time exceeds, then the block indicates a detected error. The time can be set through the structure element `iFbckDly` at input `i_strPara`. The supervision can be switched Off by the structure element `xFbckEn` at the input `i_strPara`.

## Operating the Valve

The valve can be operated if the input `i_xLock` is set to 0. An active interlock signal inhibits the operation of the valve. An active interlock is indicated by the output `q_xLock`.

The valve can only be operated, if the output `q_xErr` is set to 0. An active detected error signal inhibits the operation of the valve.

## Detected Error Management

The output `q_xErr` is high if an error is detected. The detected error can be:

- Internal detected error (invalid operation mode, missing feedback signal or unknown position.
- External detected error

The detected errors are indicated in the HMI as alarms. If an interlock or an error is detected during the operation of the valve, the behaviour of the function block depends on the structure element `i_strPara.xFrceEn` at input `i_strPara`. If this element is set to 1, the block enforces the valve to move in to the default position, and the output is high (`q_xMove`) for the duration of `i_strPara.iFbckDly` seconds. Otherwise the operation is stopped and has to be restarted after the interlock is gone.

To reset the `q_xErr`, the detected error has to be acknowledged by a rising edge on the input `i_xAckn` or by using bit 16 of the signal `i_dwCtrl`.

## Setting Default Position

The default position of the valve can be set by `i_strPara.xPosDfltSet`. This description assumes Close as default. If `i_strPara.xPosDflt` is set to 1, Open is the default position.

# Input Pin Description

## Input Pin Description

This table describes the input pins of the `Proportional_Valve` function block:

| Input | Data Type | Description | Remarks |
|---|---|---|---|
| `i_xAut` | `BOOL` | TRUE: Automatic mode enabled<br>FALSE: Disabled | - |
| `i_xMan` | `BOOL` | TRUE: Manual mode enabled<br>FALSE: Disabled | - |
| `i_xLoc` | `BOOL` | TRUE: Local mode enabled<br>FALSE: Disabled | - |
| `i_xManOpen` | `BOOL` | TRUE: Manually opens the valve (output `q_xMove`) in local mode as long as the signal is activated.<br>FALSE: Disabled | - |
| `i_xManCls` | `BOOL` | TRUE: Manually closes the valve (output `q_xMove`) in local mode as long as the signal is activated.<br>FALSE: Disabled | - |
| `i_xMove` | `BOOL` | TRUE: Move the valve into a new position in automatic mode<br>FALSE: Disabled | - |
| `i_rSp` | `REAL` | New position for movement of valve in automatic mode<br>Range: $1.17e^{-38}$...$3.4e^{+38}$ | - |
| `i_xFbckMove` | `BOOL` | TRUE: Feedback for movement of the valve<br>FALSE: Disabled | - |
| `i_iPosFbck` | `INT` | Feedback of Valve position.<br>Range: 0...31 | - |
| `i_xLock` | `BOOL` | TRUE: Interlock enabled.<br>FALSE: Disabled | Interlock input for valve operation. Valve operation is inhibited, when the input is set to 1. |

| Input | Data Type | Description | Remarks |
|---|---|---|---|
| i_xErr | BOOL | TRUE: External detected error is active.<br>FALSE: No external detected error | - |
| i_xAckn | BOOL | Acknowledgement is done with a rising edge. | Input to acknowledge internal and external detected errors indicated at the output q_xErr. |
| i_strPara | STRUCT Par_Valpro | Structure with parameters for this block. | Refer to the used structure descriptions, page 202. |
| i_dwCtrl | DWORD | Command bits to interact from the HMI.<br>Range: 0...4294967295 | Refer to the command word descriptions, page 202. |

# Output Pin Description

## Output Pin Description

This table describes the output pins of the Proportional_Valve function block:

| Identifiers | Output Type | Description | Remarks |
|---|---|---|---|
| q_xAut | BOOL | TRUE: Automatic mode enabled<br>FALSE: Disabled | - |
| q_xMan | BOOL | TRUE: Manual mode enabled<br>FALSE: Disabled | - |
| q_xMove | BOOL | TRUE: Open Command Active<br>FALSE: Disabled | - |
| q_iValvPos | INT | Signal to indicate the new position of the valve. The value is given by the set point in automatic or manual mode. If the valve is operated manually in local mode the value is set to completely opened or completely closed.<br>Range: 0...31 | If enforcement of default position is active, the signal will be set to the default position of the valve in case of interlock or detected error. |
| q_rPos | REAL | Actual position of the valve.<br>Range: ±3.4e$^{+38}$ | - |
| q_xLock | BOOL | TRUE: Interlock is active.<br>FALSE: No interlock | Indicates that the operation is blocked by an interlock (input i_xLock) |
| q_xErr | BOOL | TRUE: Detected error is active.<br>FALSE: No detected error | - |
| q_dwStat | DWORD | Status bits to be displayed in the HMI<br>Range: 0...4294967295 | Refer to the status word description, page 203. |

# Input - Output Pin

## Description

This table describes the `iq_strHmi` input/output:

| Identifier | Type | Description |
|---|---|---|
| `iq_strHmi` | `STRUCT HMI_Valpro` | Structure to interface with the HMI<br>Refer to used structures, page 202. |

# Structures Used

## *Par_Valpro*

| Structure Element | Type | Description |
|---|---|---|
| *xFbEn* | BOOL | Enable feedback signal supervision |
| *iFbTime* | INT | Delay time in seconds to get the feedback signal and valve to reach the set point |
| *xPosDflt* | BOOL | Default Position of the valve (0: closed, 1: opened) |
| *xPosDfltSet* | BOOL | Enable enforcement of default position at interlock or detected error |
| *rMinSp* | REAL | Minimum value for setpoint: close position |
| *rMaxSp* | REAL | Maximum value for setpoint: open position |
| *rCnvrFact* | REAL | Conversion factor for position output and feedback position input |
| *rBnd* | REAL | Allowed deviation between setpoint and position |

## *HMI_Valpro*

| Structure Element | Type | Description |
|---|---|---|
| *rVal* | REAL | Actual position of the valve |
| *rSp* | REAL | Setpoint for manual mode |
| *rHighLim* | REAL | Low limit for generating a detected error and alarm |
| *rLowLim* | REAL | High limit for generating a detected error and alarm |

# Control Word Bit Description

## Functionality

This table describes the control word bits:

| Bit Position | Description |
|---|---|
| 0...9 | Not used |
| 10 | Move the valve towards the set point. Bit is Edge Triggered |
| 11...15 | Not used |

| Bit Position | Description |
|---|---|
| 16 | Rising edge of this bit gives acknowledgment to detected error. |
| 17...31 | Not used |

**NOTE:** On receiving an Edge trigger (1->0 or 0->1) at this bit(Control Word bit 10), Move command to the valve is active and is SET to TRUE as long as the valve does not reach set point value. Move command becomes inactive and is RESET to FALSE once the valve reaches the set point value and is monitored using `i_iPosFbck` input OR if `q_xErr` is TRUE.

# Status Word

## Functionality

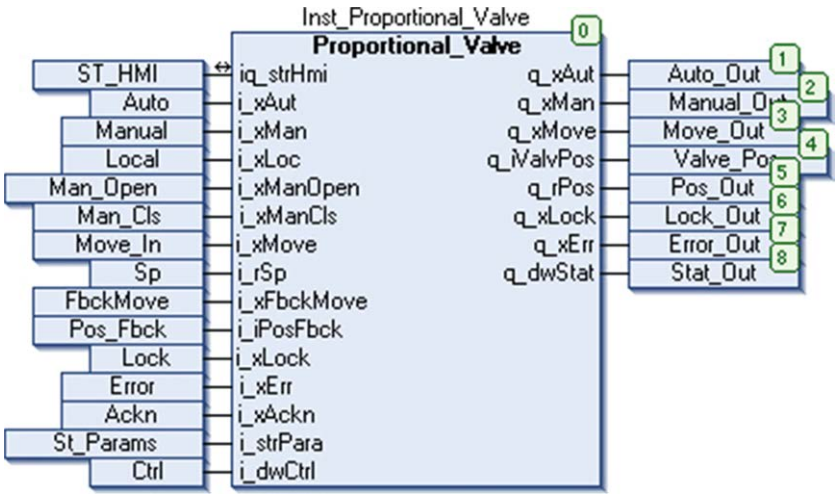
This table describes the status word:

| Bit Position | Description |
|---|---|
| 0 | Auto mode is active. |
| 1 | Manual mode is active. |
| 2 | Local mode is active. |
| 3 | Function block is locked by interlock input `i_xLock`. |
| 4...7 | Not used |
| 8 | Maximum set point reached by valve(read at input `i_iPosFbck`. |
| 9 | Minimum set point reached by valve(read at input `i_iPosFbck`. |
| 10 | Indicates movement of valve. |
| 11...13 | Not used |
| 14 | This bit is set to 1,<br>if `i_strPara.xFbEn` is true and there is a Valve move feedback<br>or<br>if `i_strPara.xFbEn` is false and there is a Valve move command |
| 15 | Not used |
| 16 | Indicates reset of detected error. |
| 17 | Detected error is present. |
| 18 | Not used |
| 19 | Equivalent to `i_strPara.xFbEn` (Feedback enable). |
| 20 | Equivalent to `i_strPara.xPosDflt` (Default position). |
| 21...23 | Not used |
| 24 | Invalid operating mode. |
| 25 | External detected error is present. |
| 26 | Feedback missing. |
| 27 | Valve has reached the low limit specified at `i_strPara.rMinSp`. |
| 28 | Valve has reached the high limit specified at `i_strPara.rMaxSp`. |
| 29...31 | Not used |

# Instantiation and Usage Example

## Instantiation and Usage Example

This figure shows an instance of the `Proportional_Valve` function block:




## Limitations

If feedback is enabled by setting `i_strPara.xFbEn` = 1, then not only the movement feedback has to come at the input `i_xFbckMove` but also the valve has to reach the setpoint within the specified time `i_strPara.iFbTime`.

If `i_strPara.rMinSp` = `i_strPara.rMaxSp` = `i_rSP` and the block is set in auto mode, then both the bits of status word `q_dwStat` for valve open and valve close is set to 1.

While operating with feedback enabled, even if no move feedback is received by the block but the valve reaches setpoint within the specified time, then no error is detected.

## Priority Management Limitations

While operating in manual mode with local control, inputs `i_xManOpen` and `i_xManCls` given together does not produce any result until one of them is withdrawn. However in this case, the value of output `q_iValvPos` is unpredictable as long as both inputs are high.

# Glossary

## A

**%:**

According to the IEC standard, % is a prefix that identifies internal memory addresses in the logic controller to store the value of program variables, constants, I/O, and so on.

**analog input:**

Converts received voltage or current levels into numerical values. You can store and process these values within the logic controller.

**analog output:**

Converts numerical values within the logic controller and sends out proportional voltage or current levels.

**ARRAY:**

The systematic arrangement of data objects of a single type in the form of a table defined in logic controller memory. The syntax is as follows: `ARRAY [<dimension>] OF <Type>`

Example 1: `ARRAY [1..2] OF BOOL` is a 1-dimensional table with 2 elements of type `BOOL`.

Example 2: `ARRAY [1..10, 1..20] OF INT` is a 2-dimensional table with 10 x 20 elements of type `INT`.

**ARW:**

(*anti-reset windup*) The feature that shuts off the reset action when the measurement is outside the proportional band to help prevent the reset circuit from overloading. Reset action begins again when the measurement returns to within the proportional band. Anti-reset windup is standard in most quality PID controllers.

**ASCII:**

(*American standard code for Information Interchange*) A protocol for representing alphanumeric characters (letters, numbers, certain graphics, and control characters).

## B

**byte:**

A type that is encoded in an 8-bit format, ranging from 00 hex to FF hex.

## C

**CFC:**

(*continuous function chart*) A graphical programming language (an extension of the IEC 61131-3 standard) based on the function block diagram language that works like a flowchart. However, no networks are used and free positioning of graphic elements is possible, which allows feedback loops. For each block, the inputs are on the left and the outputs on the right. You can link the block outputs to the inputs of other blocks to create complex expressions.

**closed loop:**

A closed loop control is a motion control system that used both positional feedback and velocity feedback to generate a correction signal. It does this by comparing its position and velocity to the values of specified parameters. The devices providing the feedback are typically encoders, resolvers, LVTDs, and tachometers.

See also: *open loop*

**control network:**

A network containing logic controllers, SCADA systems, PCs, HMI, switches, ...

Two kinds of topologies are supported:

- flat: all modules and devices in this network belong to same subnet.
- 2 levels: the network is split into an operation network and an inter-controller network.

These two networks can be physically independent, but are generally linked by a routing device.

**cyclic task:**

The cyclic scan time has a fixed duration (interval) specified by the user. If the current scan time is shorter than the cyclic scan time, the controller waits until the cyclic scan time has elapsed before starting a new scan.

**NOTE:**

## D

**DWORD:**

(*double word*) Encoded in 32-bit format.

## E

**element:**

The short name of the ARRAY element.

**equipment:**

A part of a machine including sub-assemblies such as conveyors, turntables, and so on.

## F

**FB:**

(*function block*) A convenient programming mechanism that consolidates a group of programming instructions to perform a specific and normalized action, such as speed control, interval control, or counting. A function block may comprise configuration data, a set of internal or external operating parameters and usually 1 or more data inputs and outputs.

**function:**

A programming unit that has 1 input and returns 1 immediate result. However, unlike FBs, it is directly called with its name (as opposed to through an instance), has no persistent state from one call to the next and can be used as an operand in other programming expressions.

Examples: boolean (AND) operators, calculations, conversions (BYTE_TO_INT)

## H

**HMI:**

(*human machine interface*) An operator interface (usually graphical) for human control over industrial equipment.

## I

**ID:**

(*identifier/identification*)

**input/output:**

The index of the ARRAY.

**I/O:**

(*input/output*)

## M

**MAST:**

A processor task that is run through its programming software. The MAST task has 2 sections:

- **IN:** Inputs are copied to the IN section before execution of the MAST task.
- **OUT:** Outputs are copied to the OUT section after execution of the MAST task.

**NOTE:**

**ms:**

(*millisecond*)

## N

**nibble:**

A half-byte (representing 4 bits of a byte).

## O

**open loop:**

Open loop control refers to a motion control system with no external sensors to provide position or velocity correction signals.

See also: *closed loop*.

## P

**PID:**

(*proportional, integral, derivative*) A generic control loop feedback mechanism (controller) widely used in industrial control systems.

**POU:**

(*program organization unit*) A variable declaration in source code and a corresponding instruction set. POUs facilitate the modular re-use of software programs, functions, and function blocks. Once declared, POUs are available to one another.

**program:**

The component of an application that consists of compiled source code capable of being installed in the memory of a logic controller.

**PWM:**

(*pulse width modulation*) A fast output that oscillates between off and on in an adjustable duty cycle, producing a rectangular wave form (though you can adjust it to produce a square wave).

## S

**scan:**

A function that includes:

- reading inputs and placing the values in memory
- executing the application program 1 instruction at a time and storing the results in memory
- using the results to update outputs

**setpoint:**

In a PID controller, the target value set by the user. The main objective of the PID controller is to ensure that the process value reaches the setpoint.

See also *process value*.

**STOP:**

A command that causes the controller to stop running an application program.

**string:**

A variable that is a series of ASCII characters.

## T

**task:**

A group of sections and subroutines, executed cyclically or periodically for the MAST task or periodically for the FAST task.

A task possesses a level of priority and is linked to inputs and outputs of the controller. These I/O are refreshed in relation to the task.

A controller can have several tasks.

**NOTE:**

## V

**variable:**

A memory unit that is addressed and modified by a program.

# Index

## W

Schneider Electric
35 rue Joseph Monier
92500 Rueil Malmaison
France

+ 33 (0) 1 41 29 70 00

www.se.com

As standards, specifications, and design change from time to time, please ask for confirmation of the information given in this publication.



EIO0000000096.09