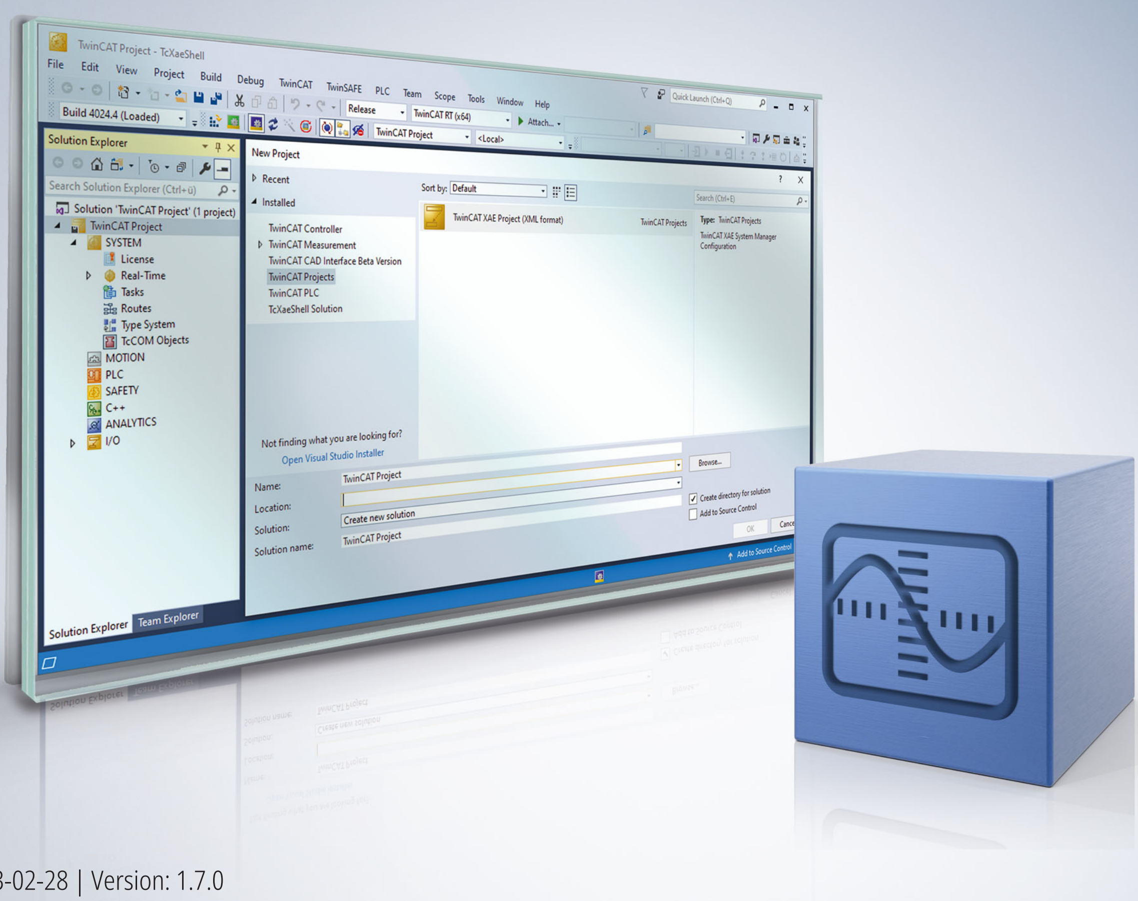
BECKHOFF New Automation Technology

Manual | EN

TF3680

TwinCAT 3 | Filter



2023-02-28 | Version: 1.7.0

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the requirements for safety, including all the relevant laws, regulations, guidelines and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without prior announcement.
No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.

EtherCAT.

EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

1.2 Safety instructions

Safety regulations

Please note the following safety instructions and explanations!
Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

Exclusion of liability

All the components are supplied in particular hardware and software configurations appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation and drive engineering who are familiar with the applicable national standards.

Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

⚠ DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

⚠ WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

⚠ CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

Tip or pointer

This symbol indicates information that contributes to better understanding.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

2 Overview

The TwinCAT 3 function TF3680 TC3 Filter provides various function blocks for implementing digital filters in a PLC library. Digital filters are used to manipulate digitalized (time-discrete and value-quantized) signals and to emphasize or suppress certain components of a signal in the frequency range, for example.

Applications:

- Suppression of frequency bands, in which the wanted signal plays a subordinate role compared to the noise signal. One example is the conventional low-pass filter for suppressing high-frequency noise.
- Targeted elimination of interfering frequency components. An example is a 50 Hz signal that is superimposed on the wanted signal, which influences the measuring signal through electromagnetic coupling of the mains voltage.

Function blocks:

Function block	Filter
FB_FTR_IIRCoeff [▶ 30] FB_FTR_IIRSos [▶ 32]	Enables the implementation of a custom filter through the specification of filter coefficients, so that, in principle, any filter characteristics can be used.
FB_FTR_IIRSpec [▶ 35]	Enables the implementation of a filter of type "Butterworth", "Chebyshev" or "Bessel".
FB_FTR_PT1 [▶ 41] FB_FTR_PT2 [▶ 44] FB_FTR_PT3 [▶ 47] FB_FTR_PTn [▶ 50] FB_FTR_PT2oscillation [▶ 58] FB_FTR_LeadLag [▶ 55] FB_PTt [▶ 61]	Enables the implementation of different delay elements of different order.
FB_FTR_MovAvg [▶ 38]	Enables the implementation of a moving average filter for smoothing.
FB_FTR_Median [▶ 64]	Median filter for smoothing signals with outliers.
FB_FTR_Notch [▶ 53]	Enables the implementation of a band-stop filter to suppress a narrow frequency band.
FB_FTR_AcutalValue [▶ 67]	Filter for the treatment of outliers.
FB_FTR_Gaussian [▶ 71]	Smoothing filter with minimal group delay.

3 Installation

3.1 System requirements

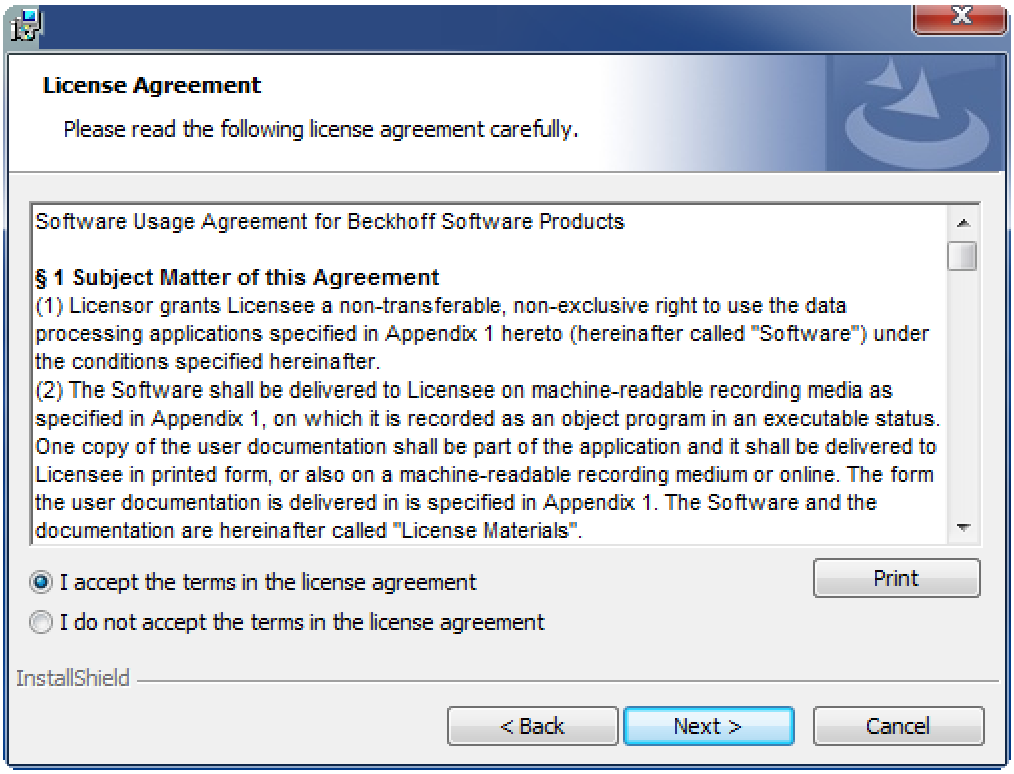
Technical data	Description
Operating system	Windows 7/10, Windows Embedded Standard 7, Windows CE 7
Target platform	PC architecture (x86, x64 or ARM)
TwinCAT version	TwinCAT 3.1 build 4022.25 or higher
Required TwinCAT setup level	TwinCAT 3 XAE, XAR
Required TwinCAT license	TF3680 TC3 Filter or TF3600 TC3 Condition Monitoring

3.2 Installation

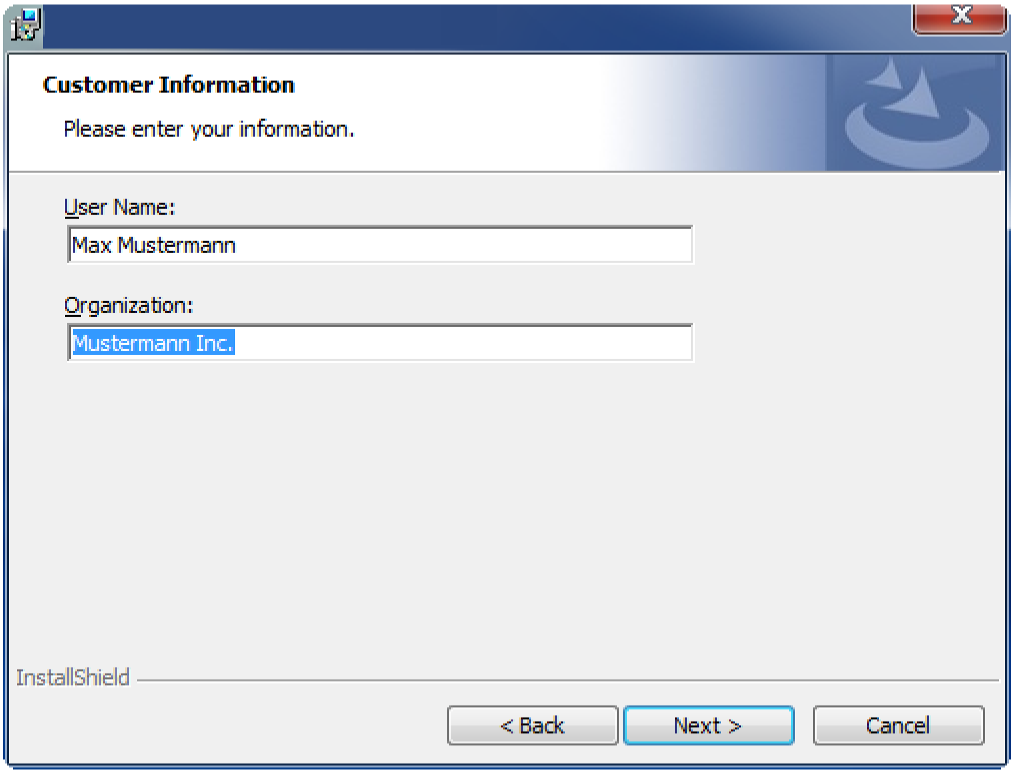
The following section describes how to install the TwinCAT 3 Function for Windows-based operating systems.

✓ The TwinCAT 3 Function setup file was downloaded from the Beckhoff website.

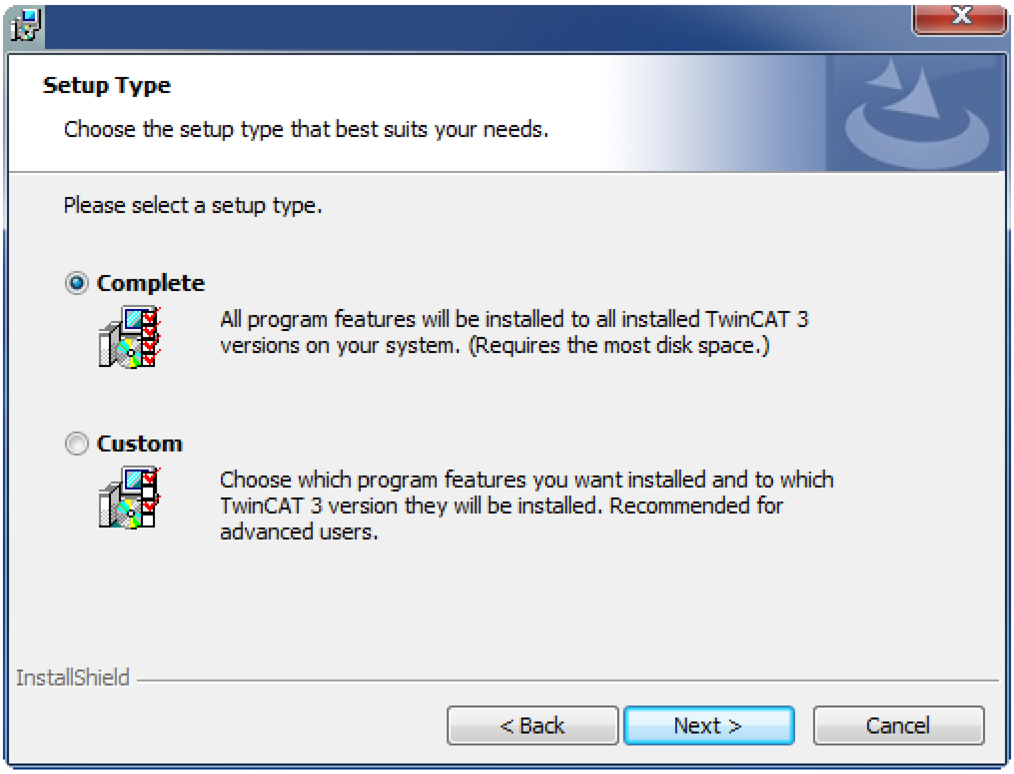
1. Run the setup file as administrator. To do this, select the command **Run as administrator** in the context menu of the file.
   ⇨ The installation dialog opens.
2. Accept the end user licensing agreement and click **Next**.



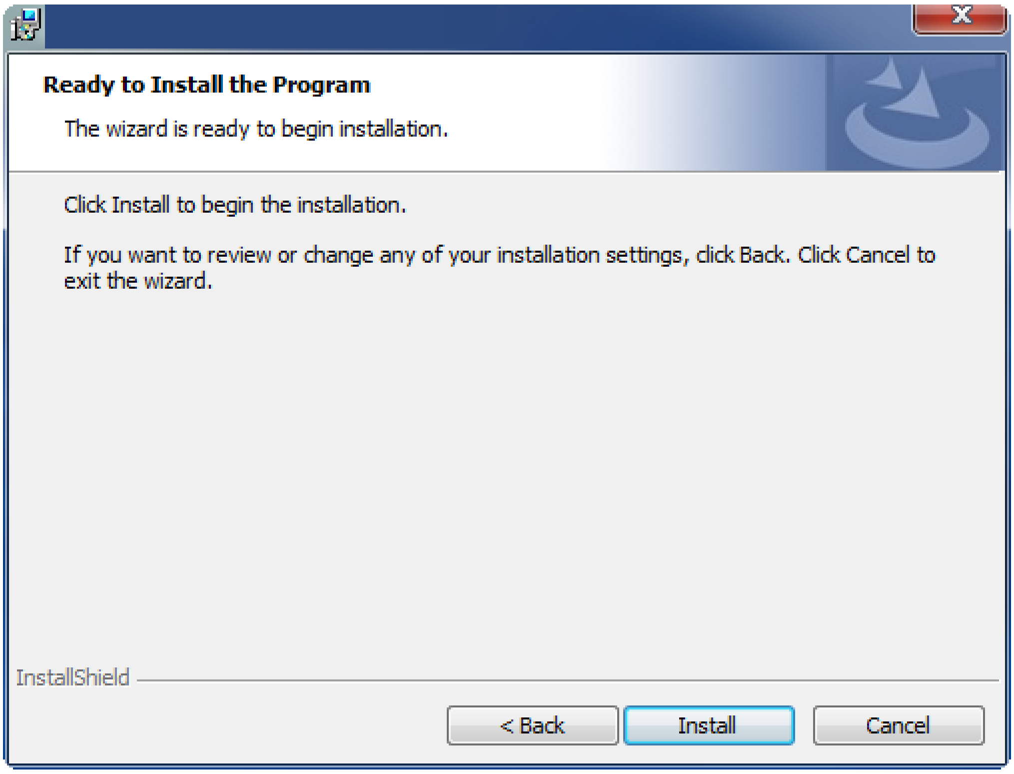
3. Enter your user data.



4. If you want to install the full version of the TwinCAT 3 Function, select **Complete** as installation type. If you want to install the TwinCAT 3 Function components separately, select **Custom**.

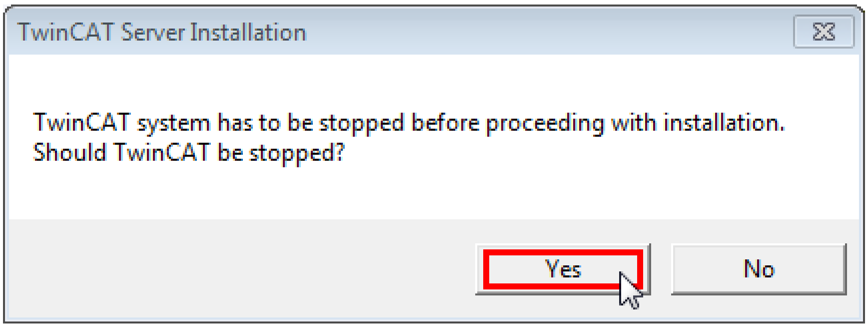


5. Select **Next**, then **Install** to start the installation.

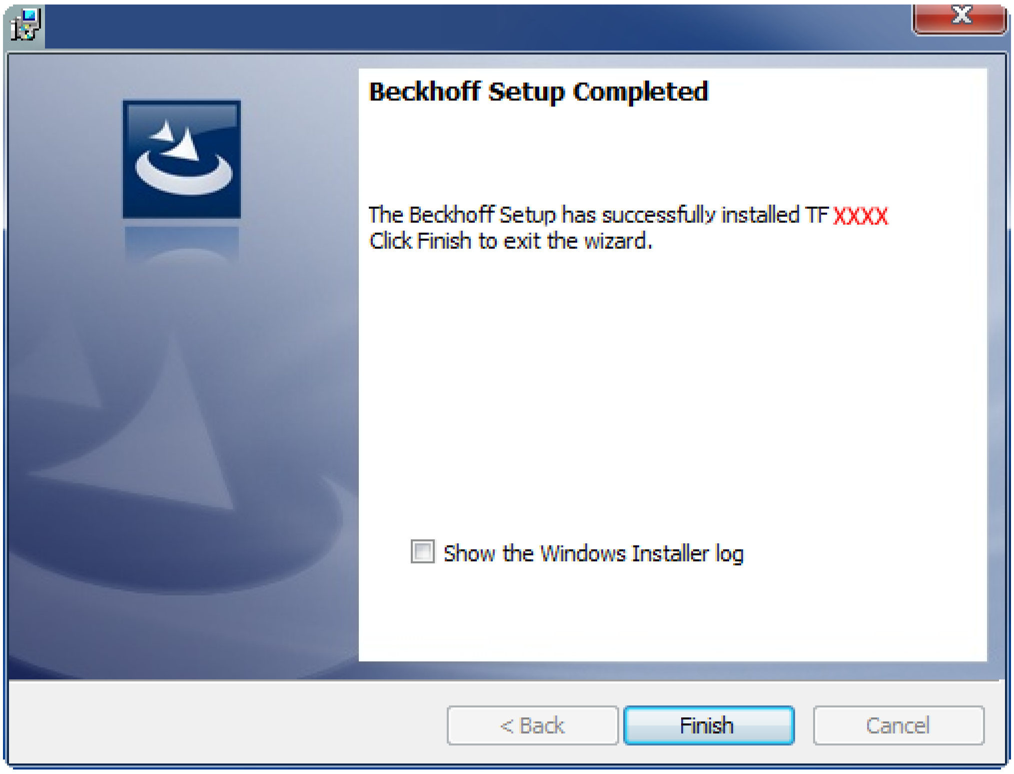


⇨ A dialog box informs you that the TwinCAT system must be stopped to proceed with the installation.

6. Confirm the dialog with **Yes**.



7. Select **Finish** to exit the setup.



⇨ The TwinCAT 3 Function has been successfully installed and can be licensed (see Licensing [▶ 12]).

3.3 Licensing

The TwinCAT 3 function can be activated as a full version or as a 7-day test version. Both license types can be activated via the TwinCAT 3 development environment (XAE).

Licensing the full version of a TwinCAT 3 Function

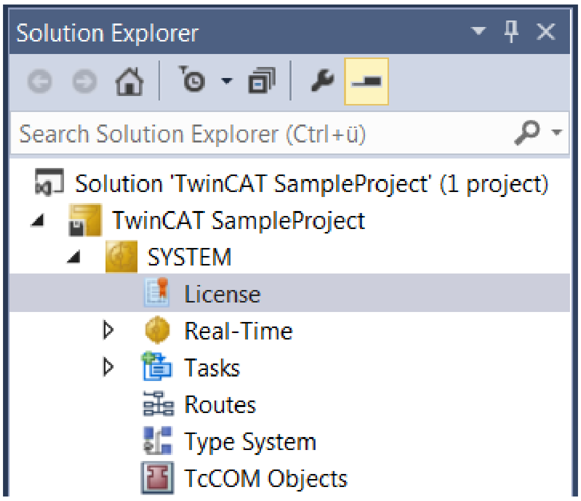
A description of the procedure to license a full version can be found in the Beckhoff Information System in the documentation "TwinCAT 3 Licensing".

Licensing the 7-day test version of a TwinCAT 3 Function

A 7-day test version cannot be enabled for a TwinCAT 3 license dongle.

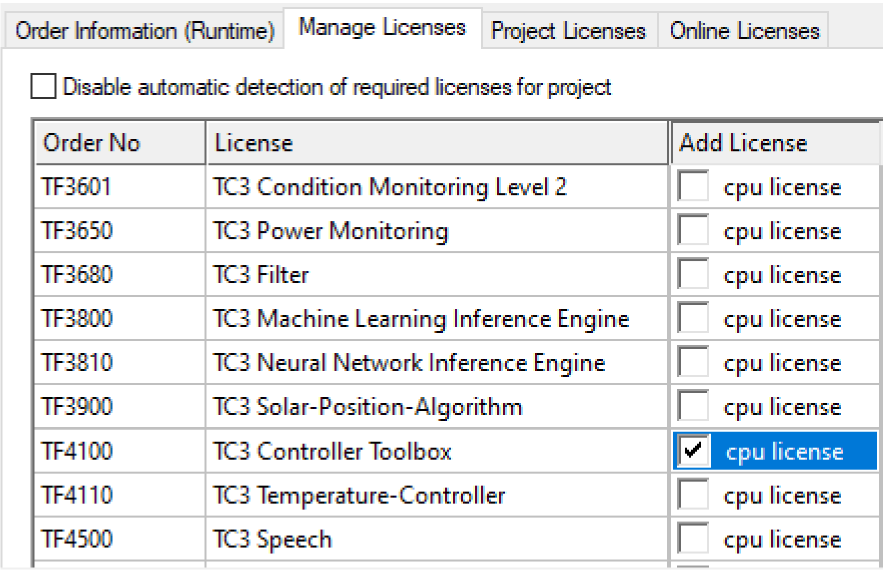
1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.
3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the toolbar.
   ⇨ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.

4. In the **Solution Explorer**, double-click **License** in the **SYSTEM** subtree.



⇨ The TwinCAT 3 license manager opens.

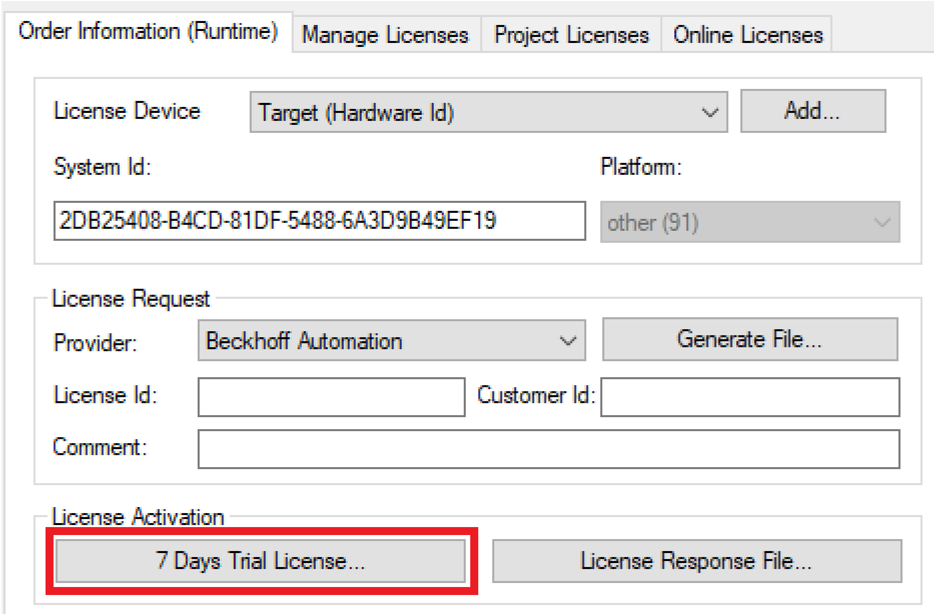
5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. "TF4100 TC3 Controller Toolbox").



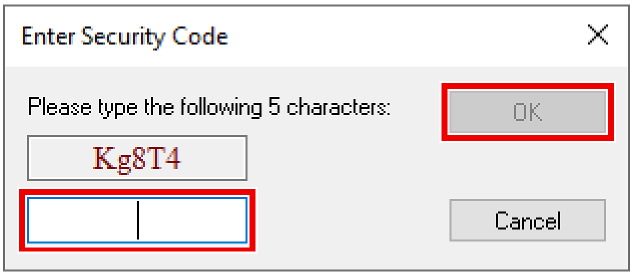
6. Open the **Order Information (Runtime)** tab.

⇨ In the tabular overview of licenses, the previously selected license is displayed with the status "missing"**.**

7. Click **7-Day Trial License...** to activate the 7-day trial license.



⇨ A dialog box opens, prompting you to enter the security code displayed in the dialog.



8. Enter the code exactly as it is displayed and confirm the entry.
9. Confirm the subsequent dialog, which indicates the successful activation.
   ⇨ In the tabular overview of licenses, the license status now indicates the expiry date of the license.
10. Restart the TwinCAT system.

⇨ The 7-day trial version is enabled.

4 Technical introduction

4.1 Digital filters

Digital filters are used to manipulate digitalized (time-discrete and value-quantized) signals. The manipulation is evident in the frequency domain, where certain components of a signal are emphasized or suppressed.

Properties

Digital filters can differ, among other things, in the frequency domain that may pass through the filter.

Filter type	Description	Area of application (examples)
Low-pass	Frequencies below a cut-off frequency can pass through the filter.	Anti-aliasing filter or filter for smoothing a signal.
High-pass	Frequencies above a cut-off frequency can pass through the filter.	Elimination of an interfering DC component in the signal.
Band-pass	Frequencies within a certain frequency interval can pass through the filter.	Useful for amplitude-modulated signals (radio technology, optical measuring signals, ultrasound signals, ...), i.e. the wanted signal is spectrally distributed around a carrier frequency, so that low and high frequencies outside the wanted signal worsen the SNR (signal-to-noise ratio) and are suppressed.
Band-stop	Frequencies out of a certain frequency interval can pass through the filter.	Suppression of an inductively coupled frequency, e.g. the mains frequency.

The specific implementation of the filter determines the transition behavior from the passband to the stopband.

See also: Filter types and parameterization [▶ 17]

Digital signals

An analog signal x(t) is converted by an analog-to-digital converter, e.g. in an EL3xxx or ELM3xxx, to a time-discrete and value-quantized signal x[n]. The time discretization takes place with the sampling period T (inverse of the sampling rate f_s).

$$x[n] = x(t = nT)$$

Difference equation

The general difference equation for an input signal x[n] (input to a discrete system, in this case a filter) and a corresponding output signal y[n] is:

$$a_0 y[n] + \sum_{k=1}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k]$$

a_k und b_k are usually real-valued coefficients (filter coefficients). The current output value y[n] of a system is thus calculated as a linear combination of past filter inputs x[n-k] with k > 0, past filter outputs y[n-k] with k > 0 and the current filter input x[n] (k = 0).

The inclusion of past filter outputs in the calculation of a current output value represents a feedback and therefore requires verification to ensure system stability. Filters with feedback are called "IIR filters" (Infinite Impulse Response filters). Filters without feedback are called "FIR filters" (Finite Impulse Response filters). The advantage of IIR filters is that "good" manipulations of the signal x[n] can be achieved with low filter orders. By definition, FIR filters can never be unstable.

Transfer function

By z-transforming the difference equation and using the linearity and the time shift property, the following general representation of the filter transfer function is obtained:

$$G(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_M z^{-M}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

The denominator coefficients a_k belong to the coefficients in the feedback. In order for the filter to be stable in conjunction with the transfer function G(z), care must be taken when calculating these coefficients that the poles of G(z) lie within the unit circle in the complex level.

IIR filters with a high filter order can become instable due to quantification effects during the calculation of the coefficients. In order to overcome these challenges, IIR filters are often implemented in cascaded biquad filters, normally called second-order sections (SOS). The overall transfer function is expressed by a multiplication of several 2^{nd} order filters. The transfer function G(z) is then described with:

$$G(z) = \prod_{m=1}^{M} G_m(z) = \prod_{m=1}^{M} \frac{b_{0m} + b_{1m} z^{-1} + b_{2m} z^{-2}}{a_{0m} + a_{1m} z^{-1} + a_{2m} z^{-2}}$$

The frequency response of a system can be determined from the transfer function G(z) by transitioning to the frequency range (frequency f) with $z = \exp(j2\pi f T)$. The amplitude response then corresponds to the magnitude of the frequency response, and the phase response corresponds to the argument of the frequency response.

Implementation in the PLC library

The PLC library Tc3_Filter provides various function blocks for implementing digital filters.

The function block FB_FTR_IIRCoeff [▸ 30] can be used to implement a free filter. The filter coefficients a_k and b_k can be calculated individually and transferred to the function block using a configuration structure. You are responsible for the stability of your filter.

With the function block FB_FTR_IIRSos [▸ 32] you can implement a free filter structured in SOS. The filter coefficients a_k and b_k can be calculated individually and transferred to the function block using a configuration structure. You are responsible for the stability of your filter.

The function block FB_FTR_IIRSpec [▸ 35] can be used to implement ready-made filters of type Butterworth, Chebyshev or Bessel through simple parameterization. The filter coefficients are thereby calculated internally as biquads.

The function block FB_FTR_MovAvg [▸ 38] and FB_FTR_Median [▸ 64] can be used to implement an average filter or median filter, which is used in many applications for smoothing signals.

Use the function block FB_FTR_Gaussian [▸ 71] to create a smoothing filter with minimal group delay so that the shape of your signal is only minimally affected as it passes through the filter and only the interfering signal components are removed.

You can use the function block FB_FTR_Notch [▸ 53] to implement a band-stop filter that is used to suppress a narrow frequency band.

You can use the function block FB_FTR_ActualValue [▸ 67] to perform a plausibility check of a measured input value.

In addition, further filters that are commonly used in system theory and control technology are made available to you: PTt [▶ 61], PT1 [▶ 41], PT2 [▶ 44], PT3 [▶ 47], PTn [▶ 50], PT2oscillation [▶ 58] and LeadLag [▶ 55] elements.

A PT1 element and a Butterworth 1st order low-pass filter can be converted equivalently to each other, but the featured parameters of the filters are different.

Bilinear transformation

The parameterization of the predefined filters takes place in the Laplace space. The implementation of the time-continuous system representation in the time-discrete z-space takes place internally with the help of the bilinear transformation.

$$s = \frac{2}{T}\frac{z-1}{z+1}$$

The effect of "frequency warping" is taken into account in the filter design.

4.2 Filter types and parameterization

This description is limited to low-pass filters. However, the concepts can be applied to other filter types (high-pass, band-pass and stop-band filters).

TwinCAT Filter Designer

The graphics created in the following were created with the TwinCAT Filter Designer. The Filter Designer makes it possible to create filters graphically and then use them in the PLC with TwinCAT 3 Filter (or as a filter directly on the EtherCAT Terminal or Drive).

The Butterworth filter, the Chebyshev filter and the Bessel filter are common implementations of a digital filter. Each filter has certain properties that are beneficial in different situations.

Butterworth filter vs. Chebyshev filter:

The difference between the two implementations essentially consists of the balance between the permissible ripple of the amplitude response in the passband and the slope of the amplitude response in the transition between the passband and the stopband. While the Butterworth filter has a maximally flat amplitude response in the passband, for the Chebyshev filter the permissible ripple of the amplitude response in the passband is specified as a parameter. The advantage of the Chebyshev filter is a steeper decrease of the amplitude response in the transition from the passband to the stopband.

Bessel filter vs. Chebyshev and Butterworth filters:

In the Bessel filter, the focus is on a constant group delay or a linear phase response in the passband of the filter. As a result, the shape of signals with spectral components in the passband is not changed when passing through the filter. Compared to the Butterworth or Chebyshev filter, the transition from the passband to the stopband is less sharp in the Bessel filter, i.e. amplitudes near the cut-off frequency are comparatively less attenuated. In the passband itself, the Bessel filter exhibits a monotonically decreasing amplitude response.

The filter types are compared and described in more detail below. First, some basic terms are explained briefly.

Transfer function in the amplitude/frequency diagram

The filter is described mathematically by the transfer function (see Digital filters [▶ 15]). The transfer function can be displayed in the form of an amplitude and a phase response.

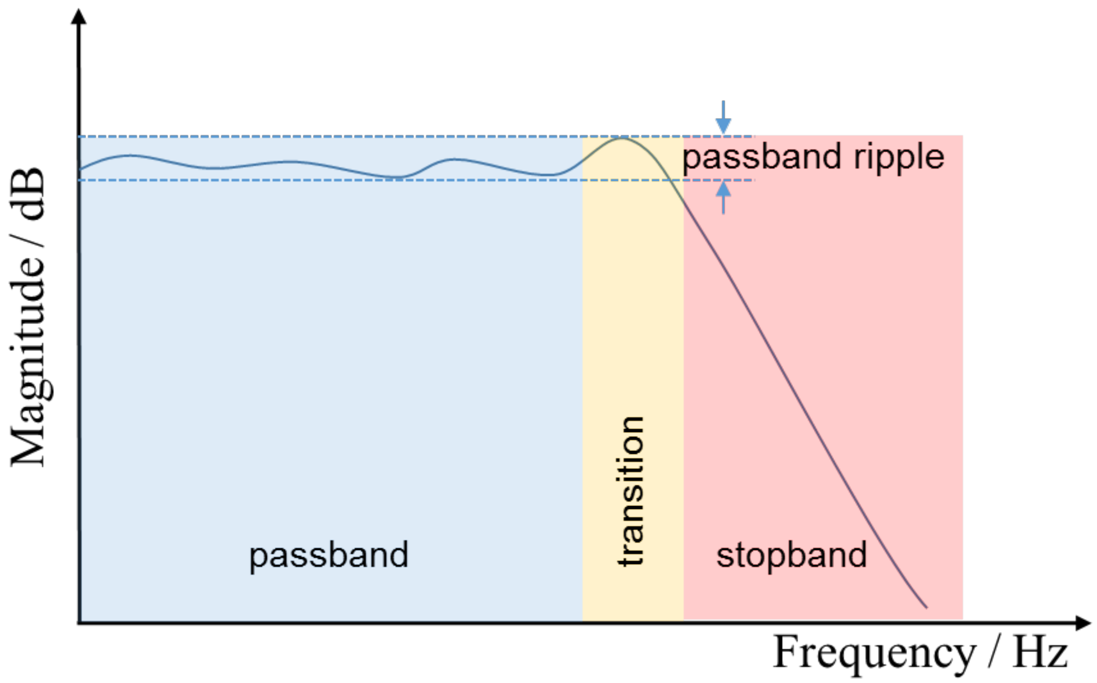


Fig. 1: Graphical representation of the amplitude response of a low-pass filter

Passband

The passband (blue zone) allows spectral components of a signal to pass through. Modification of the signal in this frequency range should be avoided.

Stopband

In the stopband (red zone), the filter attenuates the corresponding frequency components of the signal.

Transition

The transition (yellow zone) separates the passband and the stopband. It should normally be as small as possible. The design of the transition phase is a defining criterion for the selection of the filter type and its parameterization.

Passband ripple

The ripple in the passband describes the waviness of the amplitude response in the passband.

Parameterization of the Butterworth filter

Properties

The amplitude response of the Butterworth filter is maximally flat in the passband, so that the wanted signal in this range is only minimally manipulated. In addition, the entire course of the amplitude response is monotonous, i.e. without passband ripple. This filter type is one of the most frequently used filter types.

Parameter

The transfer function of the Butterworth filter contains only two parameters to be defined: the cut-off frequency and the filter order.

Filter order

The filter order determines how steeply the amplitude response decreases in the transition. The higher the filter order, the steeper the amplitude response decreases and the smaller the transition. For the slope of the amplitude response for a Butterworth filter, -n * 20 dB/decade, with n = order, i.e. -20 dB/decade for filter order 1, -40 dB/decade for filter order 2, and so on.

Cut-off frequency

The cut-off frequency of the Butterworth filter is defined as the frequency at which the normalized amplitude response assumes the value 1/sqrt(2) ≈ -3 dB. This applies to all filter orders. Accordingly, when designing the filter, care must be taken to ensure that the spectral components of a signal are already attenuated by 3 dB at the cut-off frequency. This parameter causes a parallel shift of the amplitude response along the frequency axis (distortion due to the logarithmic frequency axis).

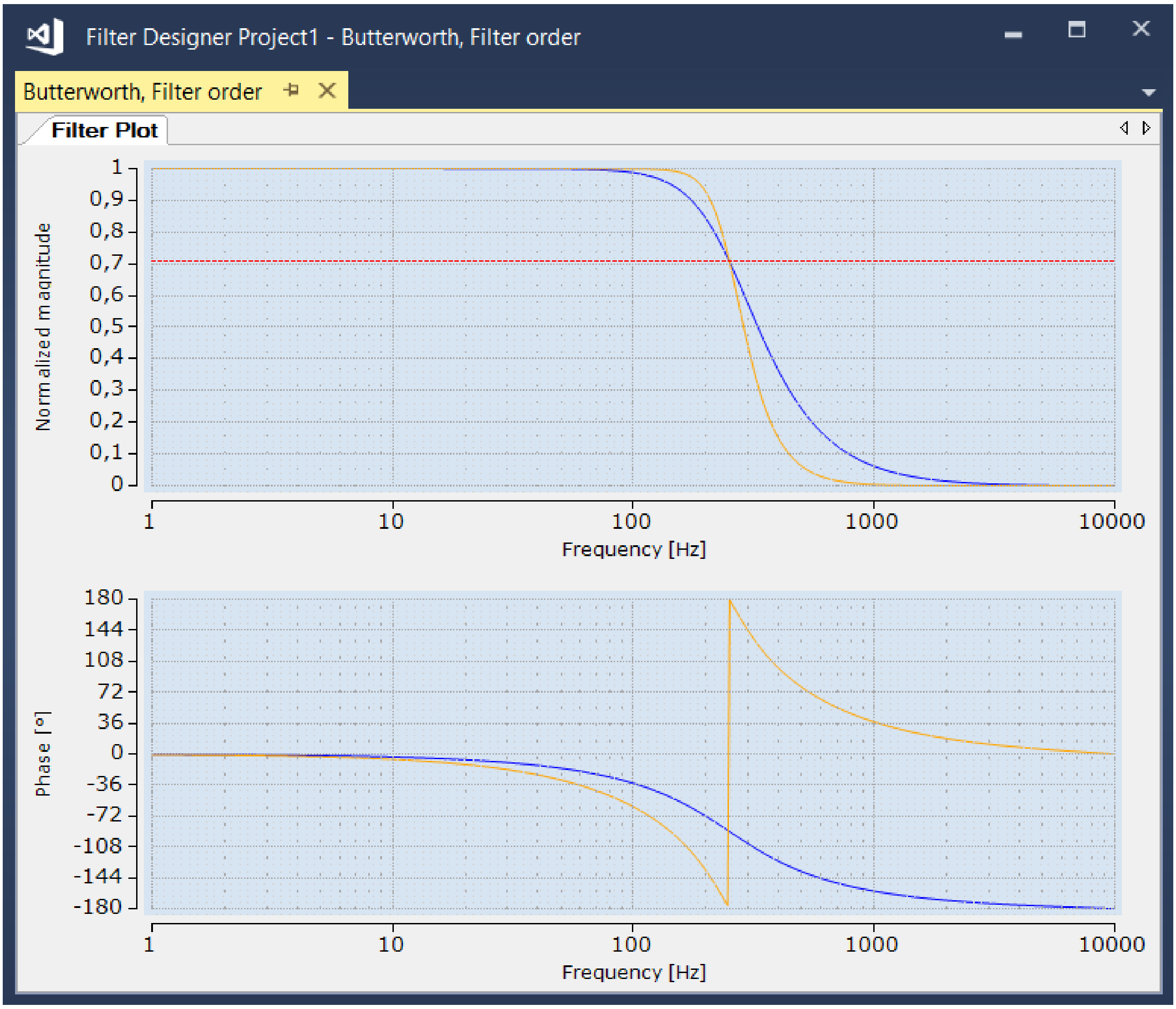


Fig. 2: Graphical representation of the amplitude and phase response of a Butterworth filter with identical cut-off frequency (blue: filter order 2, yellow: filter order 4)

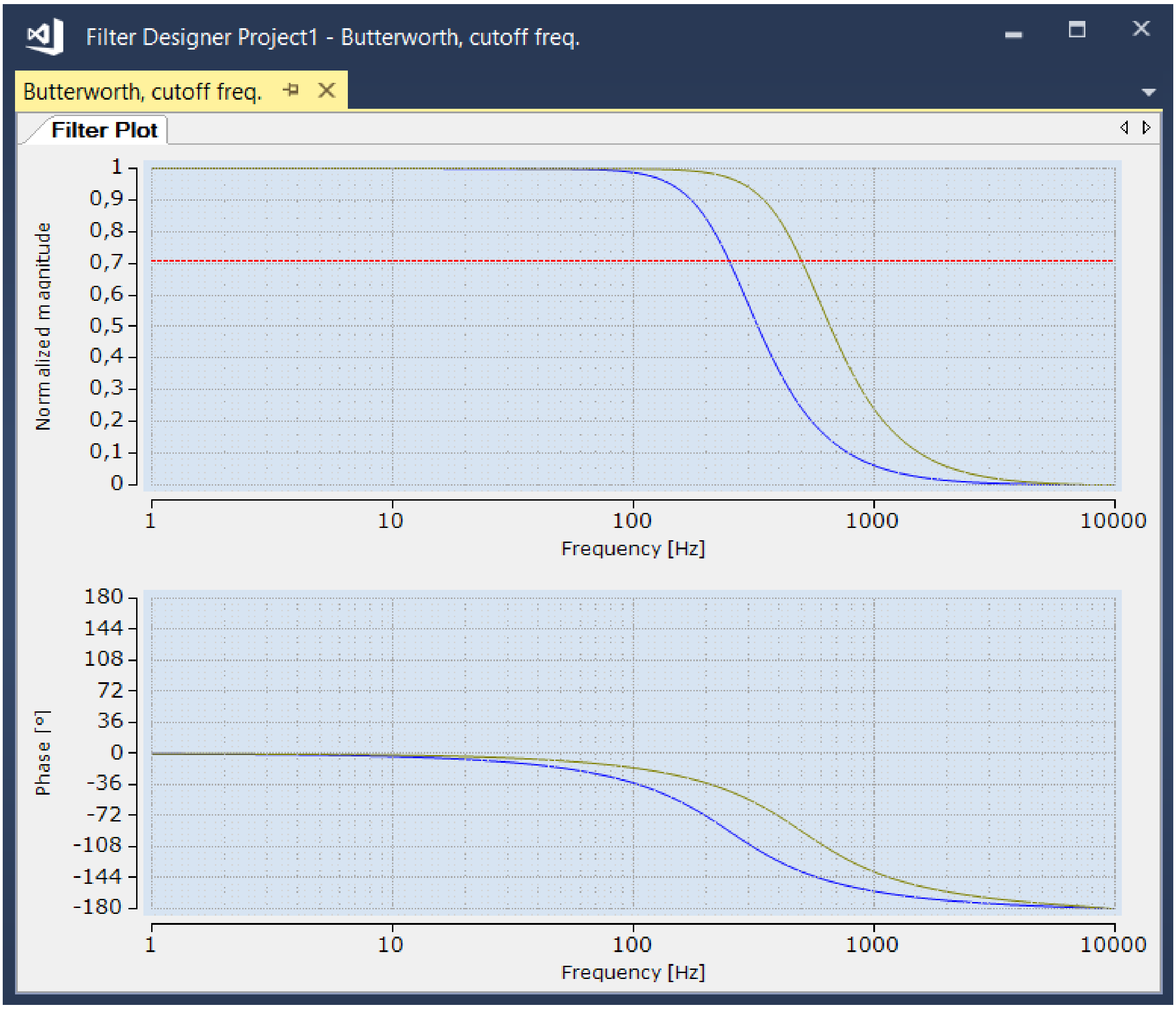


Fig. 3: Graphical representation of the amplitude and phase response of a Butterworth filter with identical filter order (blue: cut-off frequency 400 Hz, green: cut-off frequency 700 Hz)

Parameterization of the Chebyshev filter

Properties

The amplitude response of the Chebyshev filter has a parameterizable passband ripple. However, the amplitude response decreases steeply in the transition even at low filter order. The following applies: the greater the permissible passband ripple, the shorter the transition.

Parameter

In addition to the filter order and the cut-off frequency as parameters to be defined, the transfer function of the Chebyshev filter contains a "passband ripple" parameter.

Passband ripple

The parameter specifies the permissible ripple of the amplitude response in the passband of the filter. By allowing a passband ripple, a short transition between passband and stopband, and thus a steep decrease of the amplitude response, can be achieved with a significantly lower filter order.

Cut-off frequency

The cut-off frequency of the Chebyshev filter is defined as the frequency at which the amplitude response passes downwards through the defined "passband ripple". The position of the transition on the frequency axis is thus associated not only with the cut-off frequency, but also with the settings for the filter order and passband ripple.

The following diagram shows three different Chebyshev filters with different filter order and passband ripple, but the same cut-off frequency.

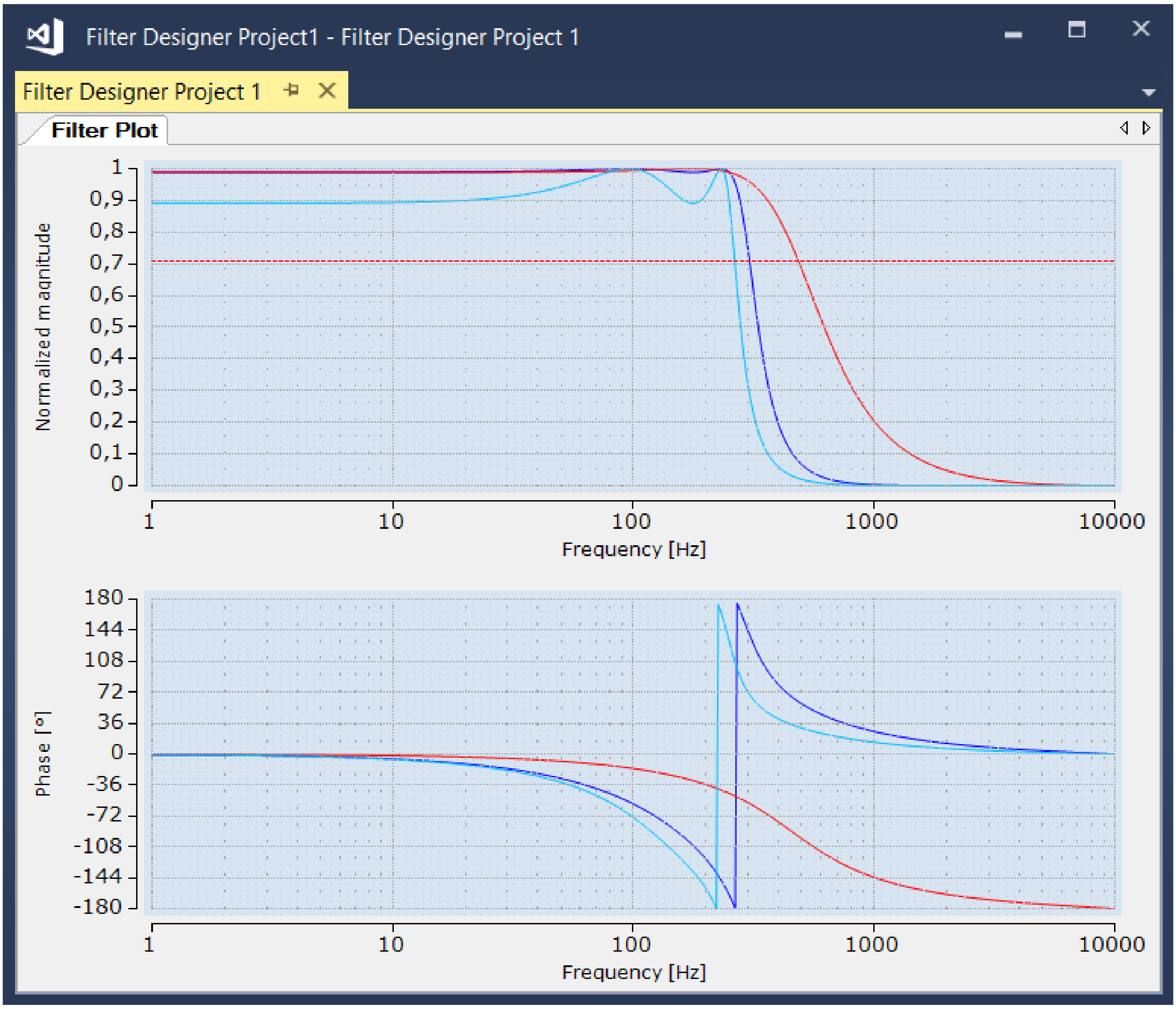


Fig. 4: Graphical representation of the amplitude and phase response of a Chebyshev filter with identical cut-off frequency (blue: filter order 4, passband ripple 0.1 dB, red: filter order 2, passband ripple 0.1 dB, cyan: filter order 4, passband ripple 1 dB)

Filter order

The filter order determines how steeply the amplitude response decreases in the transition. The higher the filter order, the steeper the amplitude response decreases and the smaller the transition. For the slope of the amplitude response for a Butterworth filter, -n * 20 dB/decade, with n = order, i.e. -20 dB/decade for filter order 1, -40 dB/decade for filter order 2, and so on.

The filter order does not influence the cut-off frequency according to the above definition, as can be seen in the following graph. The amplitude response curves intersect at the cut-off frequency of 250 Hz.

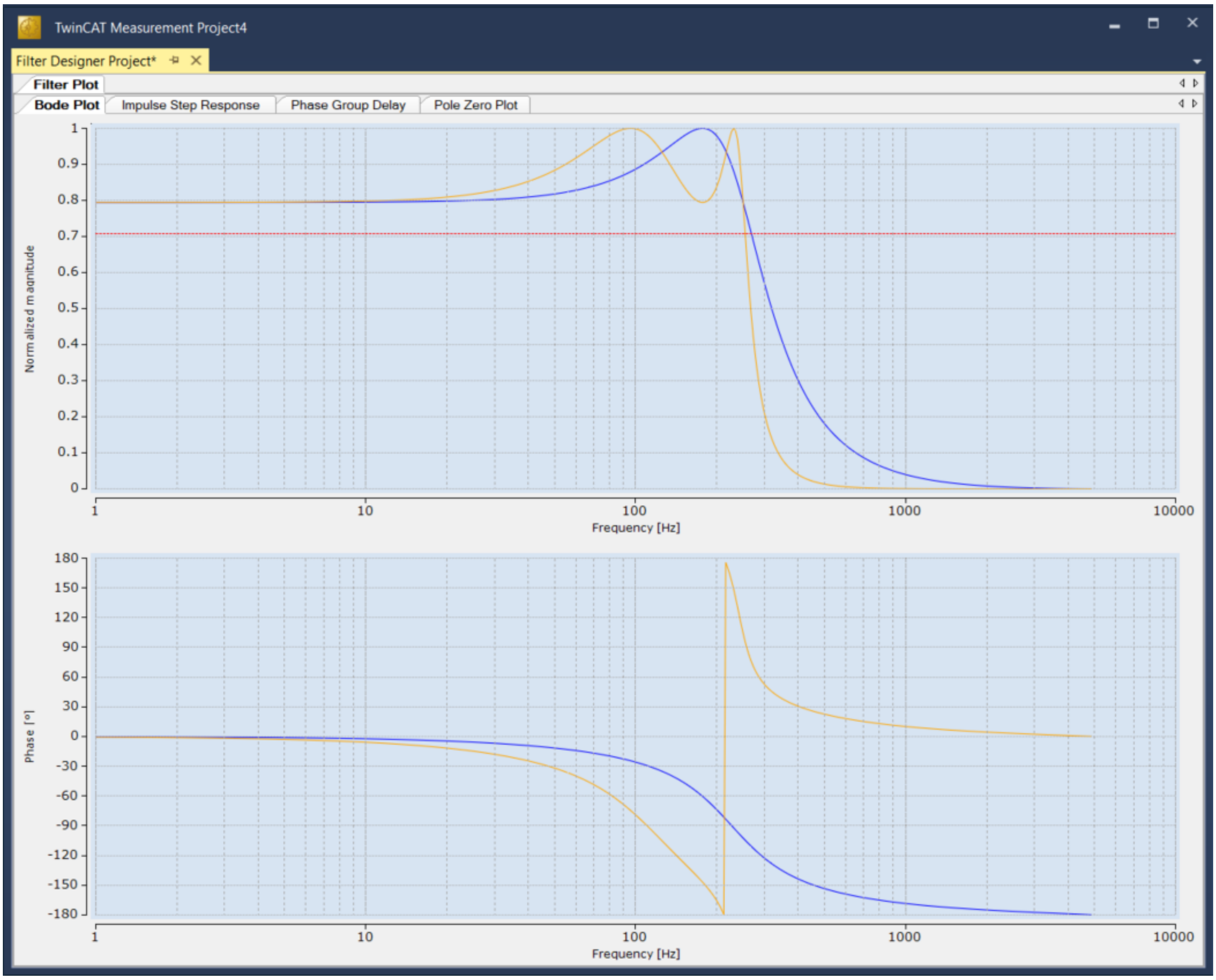


Fig. 5: Graphical representation of the amplitude and phase response of a Chebyshev filter with identical cut-off frequency (250 Hz) and identical passband ripple (2 dB): (blue: filter order 2, yellow: filter order 4)

Comparison of Butterworth and Chebyshev filters

The following diagram shows a direct comparison of the amplitude and phase response of a Butterworth filter and a Chebyshev filter. Both filters are parameterized so that their amplitude responses intersect at the cut-off frequency of the Butterworth filter at a normalized amplitude of 1/sqrt(2). Both filters are defined as fifth order filters. The passband ripple parameter of the Chebyshev filter is 0.5 dB.

The weighing up referred to above between the permissible passband ripple of the amplitude response and the slope in the transition with the same filter order becomes apparent. With the same filter order, the amplitude response of the Chebyshev filter decreases more sharply in the transition than that of the Butterworth filter. On the other hand, its amplitude response is not smooth in the passband, so that the wanted signal is manipulated more strongly here than with the Butterworth filter.

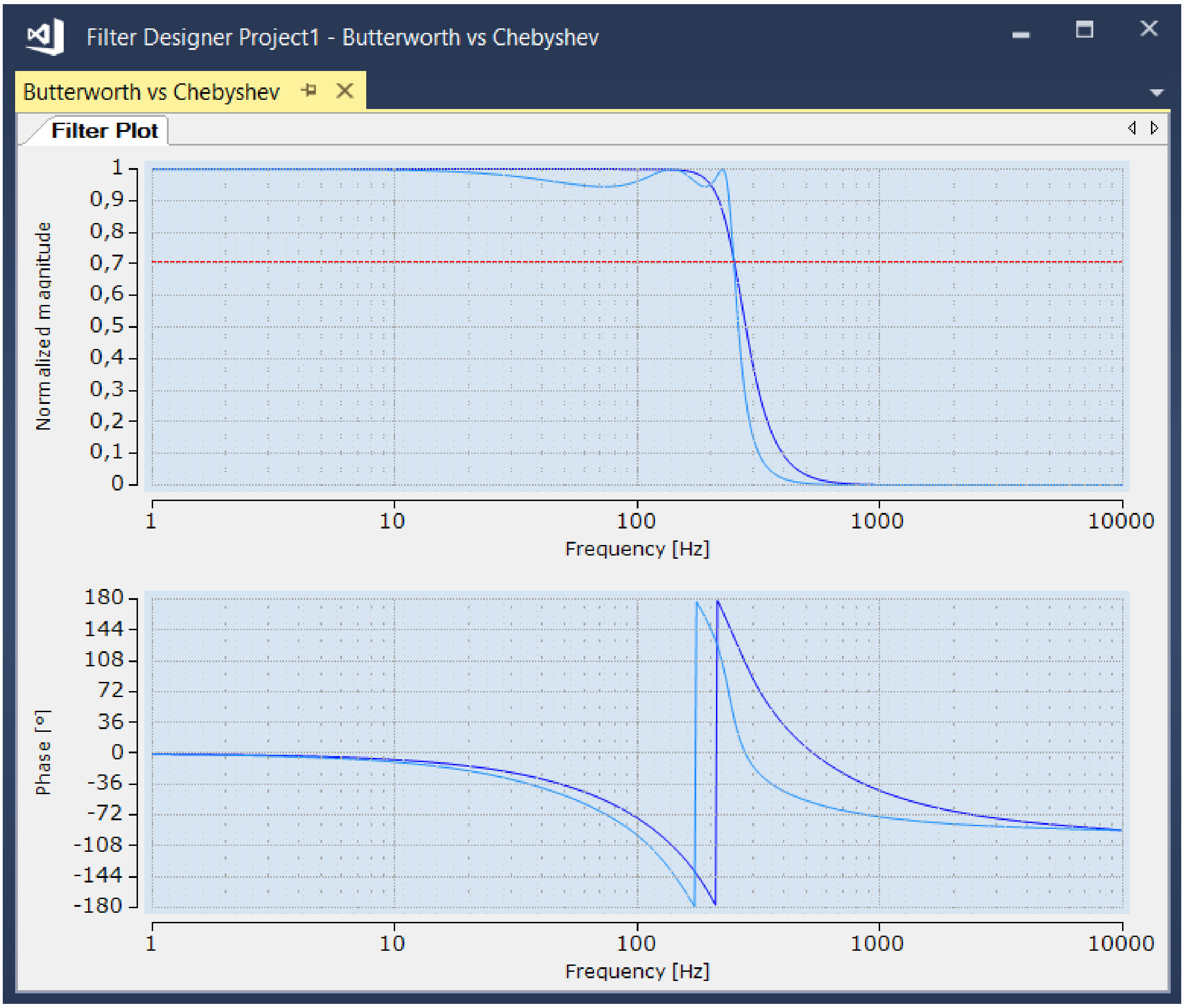


Fig. 6: Graphical representation of the amplitude and phase response of a Butterworth filter (blue) and a Chebyshev filter (cyan)

Parameterization of the Bessel filter

Properties

The Bessel filter has a constant group delay in the passband. The amplitude response is monotonically slightly decreasing. Due to these properties, a signal that has only spectral components in the passband will not change in its signal shape when passing through the filter.

Parameter

Like the Butterworth filter, the Bessel filter is parameterized via the cut-off frequency and filter order.

Cut-off frequency

The cut-off frequency f_c defines the amount of group delay of the Bessel filter in the passband $\tau_{gd,pass}$:

$$\tau_{\mathrm{gd,pass}} = \frac{1}{2 f_s \tan\left(\pi f_c / f_s\right)}$$

Here f_s is the sampling frequency.

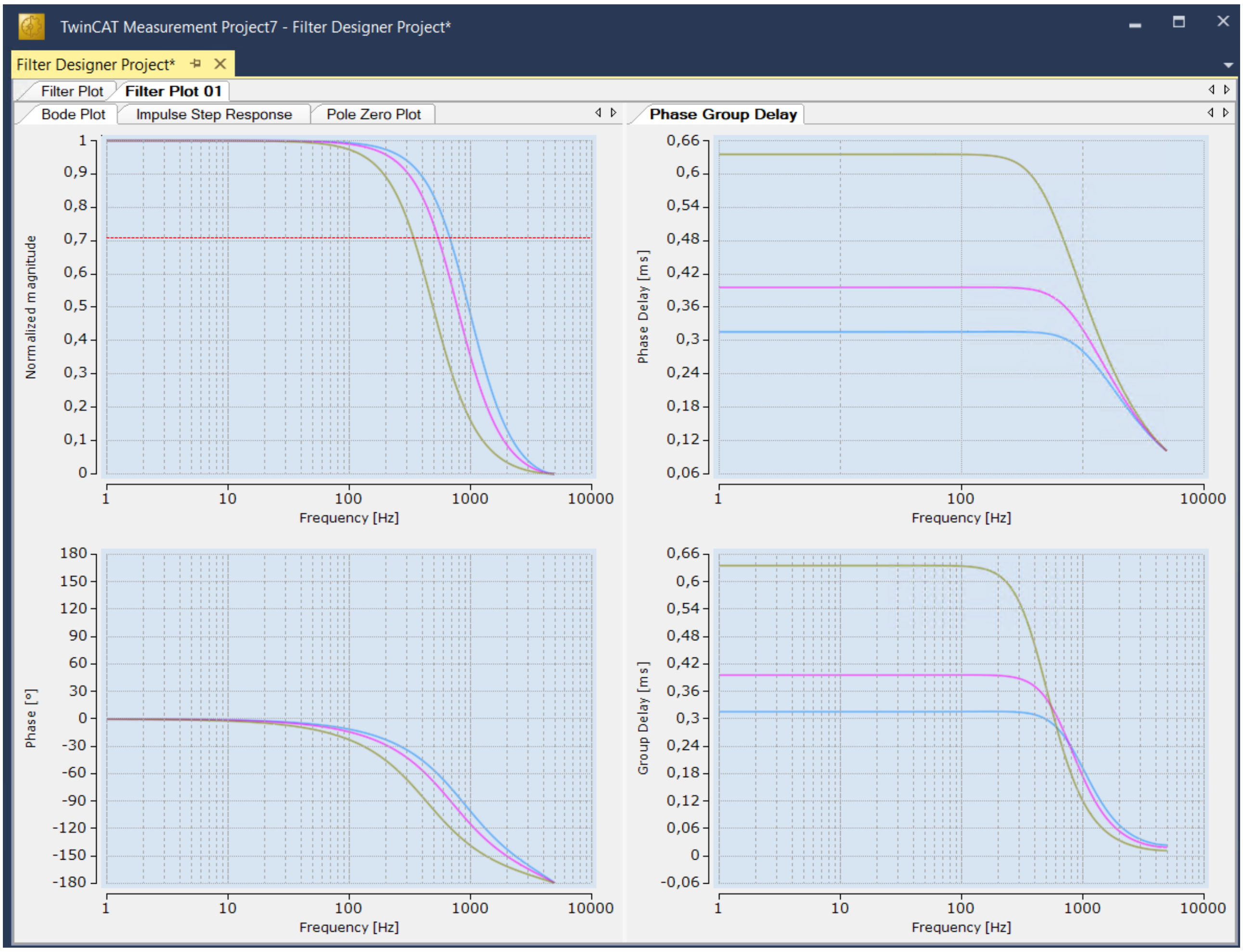


Fig. 7: Graphical representation of the amplitude and phase response of a Bessel filter (blue: cut-off frequency 500 Hz, magenta: cut-off frequency 400 Hz, olive: cut-off frequency 250 Hz)

Filter order

The filter order influences the slope of the amplitude response in the stopband. It should be noted with the Bessel filter that an increase in the filter order is accompanied by an increase in the passband. Accordingly, it is advisable to select the order first and then use the cut-off frequency to define the passband.

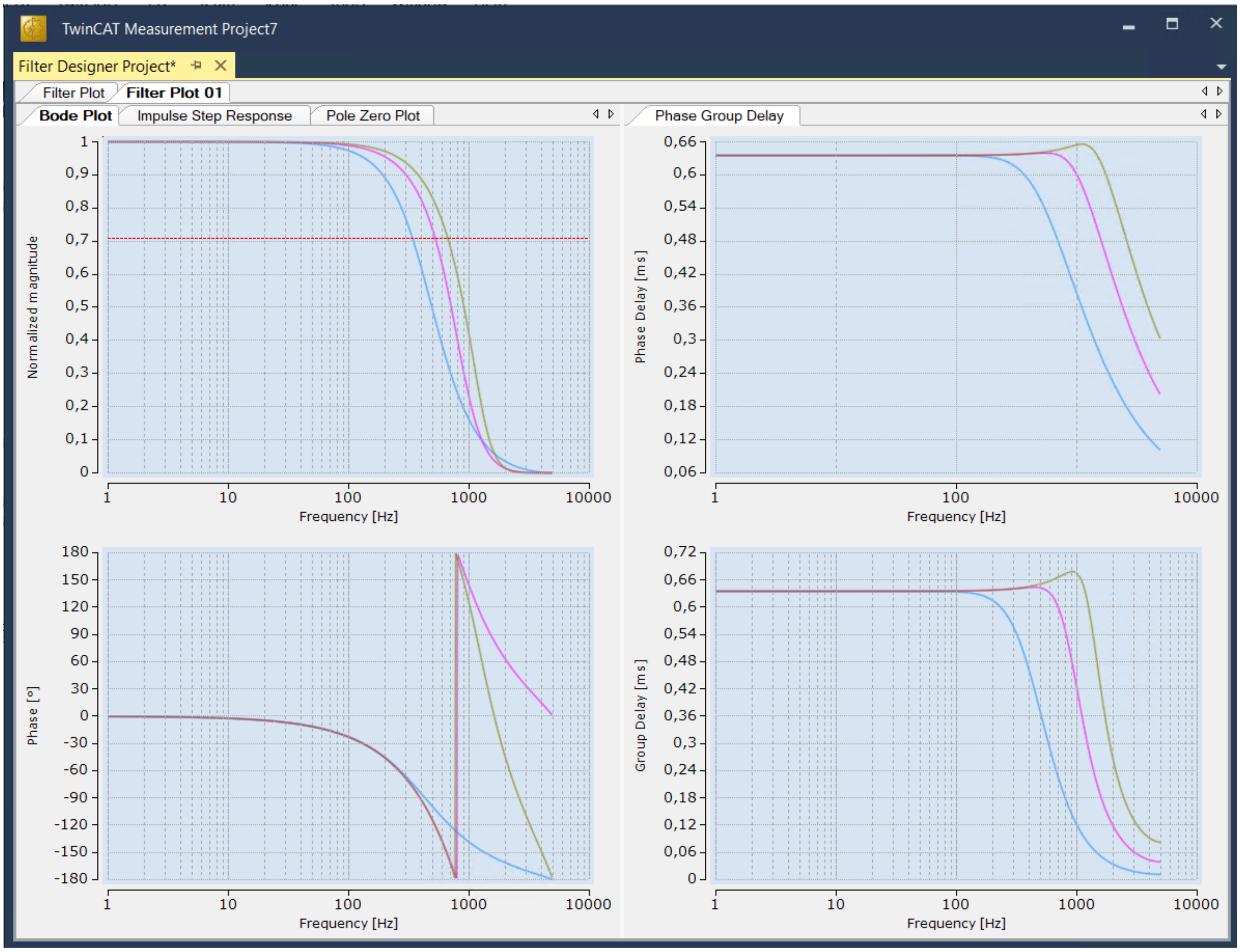


Fig. 8: Graphical representation of the amplitude and phase response of a Bessel filter (blue: filter order 2, magenta: filter order 4, olive: filter order 6)

Comparison of Butterworth, Chebyshev and Bessel filters

The main characteristics of the Bessel filter can be seen in the time domain or in the phase and group delay. As shown in the graph below, the impulse response and step response of the Bessel filter do not require much settling. In addition, phase delay and group delay are almost constant in the passband of the filter, which means that signals with spectral components in the passband are not changed in shape.

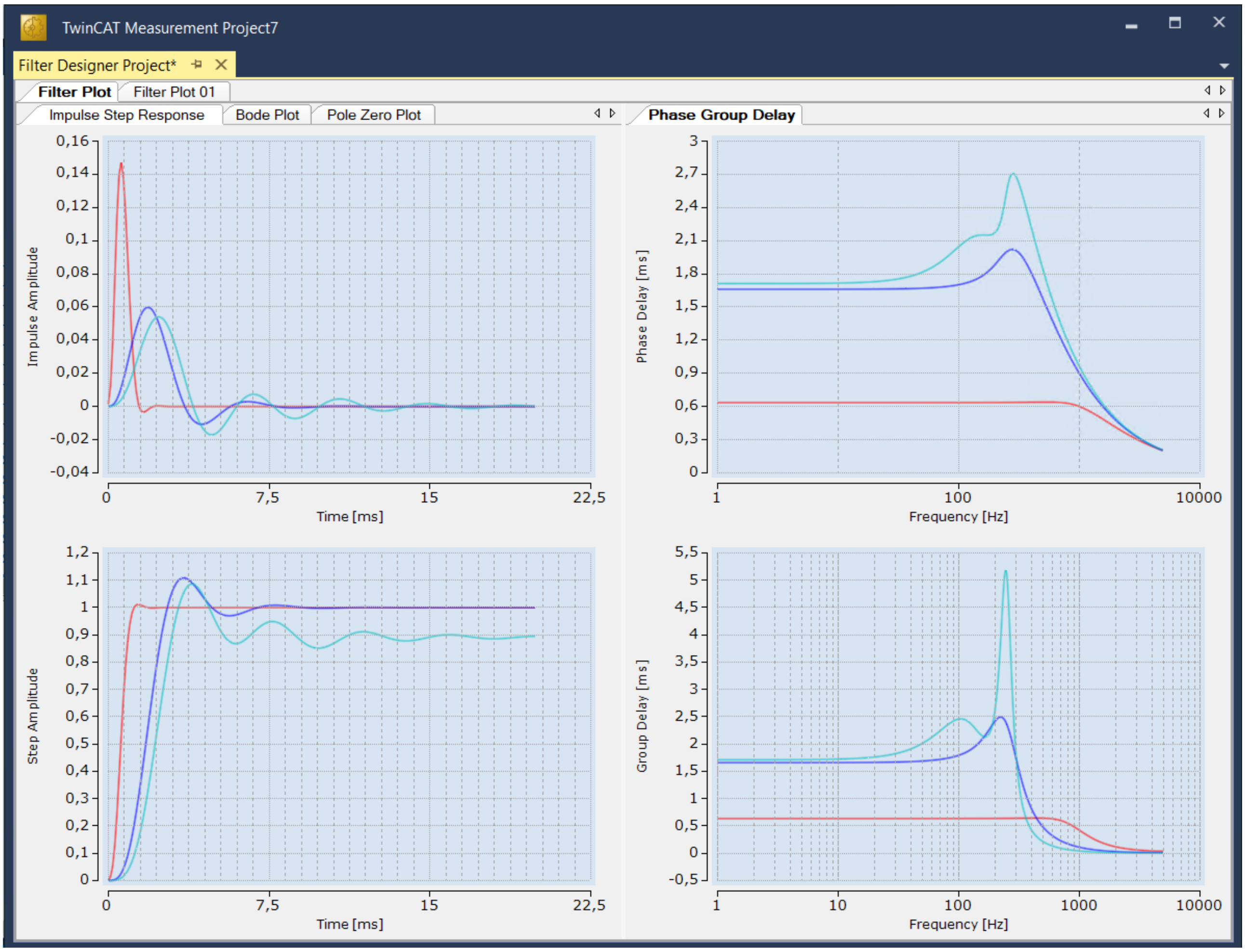
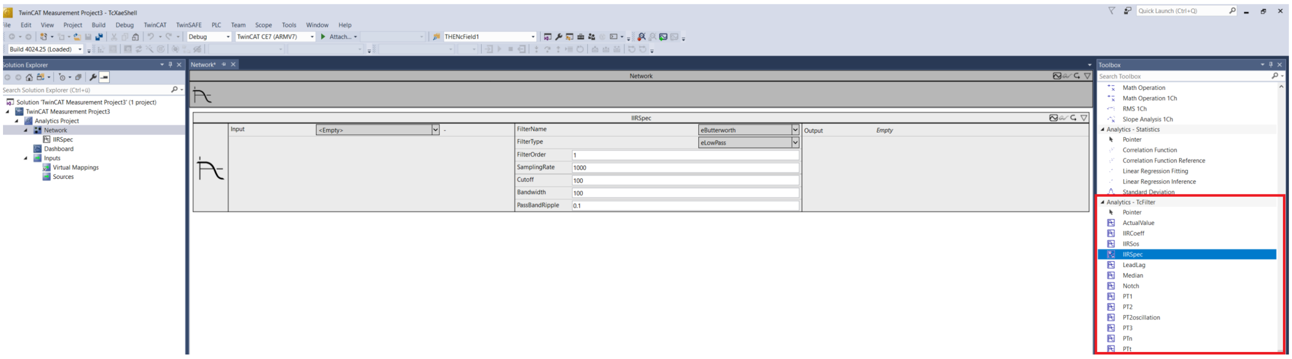


Fig. 9: Graphical representation of the impulse and step response as well as the phase and group delay of a Butterworth filter (blue), a Chebyshev filter (cyan) and a Bessel filter (red).

4.3 TwinCAT Analytics

The library can also be used in the TwinCAT Analytics Engineering products to create continuous data analyses from different, decentralized machine controllers. In addition, Analytics Workbench can be used to automatically generate a PLC application, and the generated code can be downloaded to Analytics Runtime, where it runs 24/7 in parallel with applications in the field.



Further information can be found in the TwinCAT Analytics documentation.

5 PLC API

5.1 Function blocks

Basic structure of the function blocks

All function blocks of the library Tc3_Filter are based on the same basic structure. This simplifies the engineering process when changing from one filter type to another.

Syntax

```
FUNCTION_BLOCK FB_FTR_<type>
VAR_INPUT
    stConfig        : ST_FTR_<type>;
END_VAR
VAR_OUTPUT
    bError          : BOOL;
    bConfigured     : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

To configure the filter, a configuration structure of type ST_FTR_<type> is transferred to the function blocks during instantiation. The configuration structure can be assigned in the declaration or via the `Configure()` method at runtime.

See also: Data types [▸ 74] > Configuration structures [▸ 74]

Sample of configuration in the declaration:

```
(* define configure structure - exemplary for IIRSpec *)
stParams : ST_FTR_IIRSpec := (
    eFilterName    := E_FTR_Name.eButterworth,
    eFilterType     = E_FTR_Type.eLowPass,
    nFilterOrder   := nFilterOrder,
    fSamplingRate  := fSampleRate,
    fCutoff        := fCutoff,
    nOversamples   := cOversamples,
    nChannels      := cChannels);
```

```
(* create filter instance with configure structure *)
fbFilter : FB_FTR_IIRSpec := (stConfig := stParams);
```

Outputs

All function blocks have an error flag `bError` and a flag `bConfigured` of type `BOOL` as output parameters. These indicate whether an error has occurred and whether the corresponding function block instance has been successfully configured. The output `ipResultMessage` of type `I_TcMessage` provides various properties for explaining the cause of an event and methods for processing the message (event list [▸ 95]).

See also: I_TcEventBase und I_TcMessage

Methods

All function blocks of the library Tc3_Filter have three methods. They return a positive value if they were executed without errors.

Configure()

This method can be used at runtime to initially configure the instance of a filter function block (if it was not already configured in the declaration) or to reconfigure it.

```
METHOD Configure : BOOL
VAR_INPUT
     stConfig : ST_FTR_<type>;
END_VAR
```

Call()

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

```
METHOD Call : BOOL
VAR_INPUT
     pIn       : POINTER TO LREAL; (*address of input array*)
     nSizeIn   : UDINT;            (*size of input array*)
     pOut      : POINTER TO LREAL; (*address of output array*)
     nSizeOut  : UDINT;            (*size of output array*)
END_VAR
```

Reset()

The method resets the internal status of a filter. The influence of the past values on the current output value is eliminated.

```
METHOD Reset : BOOL
```

Properties

The library Tc3_Filter references the TwinCAT 3 EventLogger and thus ensures that information (events) is provided via the standardized interface I_TcMessage.

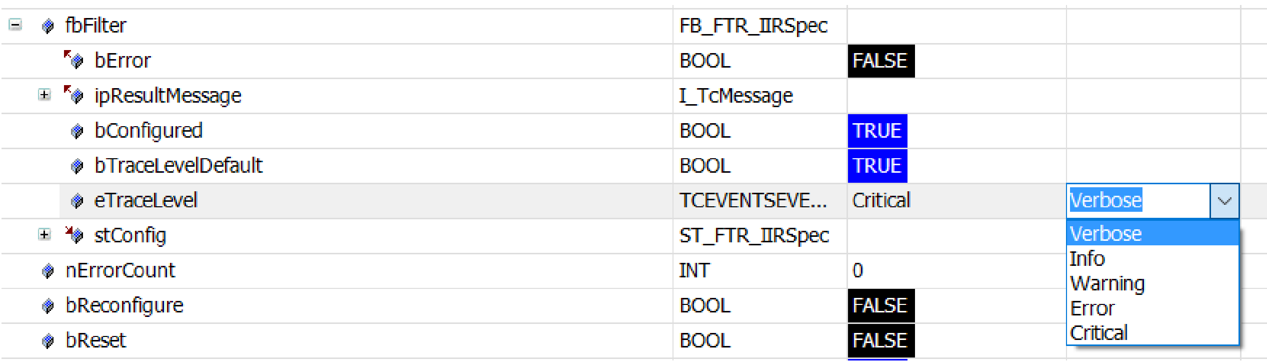
Each function block has the properties `eTraceLevel` of type `TcEventSeverity` and `eTraceLevelDefault` of type `BOOL`.

The trace level determines the severity of an event (verbose, info, warning, error, critical) and is set using the `eTraceLevel` property.

```
(* Sample of setting fbFilter to trace level info *)
fbFilter.eTraceLevel := TcEventSeverity.Info;
```

The property `eTraceLevelDefault` can be used to reset the trace level to the default value (`TcEventSeverity.Critical`). The property can be read and written, i.e. the property `eTraceLevelDefault` can be used to query whether the default value is set.

The properties can also be set in Online view.



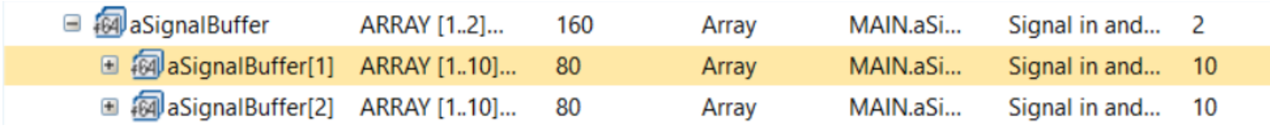
Dealing with oversampling and multiple channels

All function blocks are oversampling- and multi-channel-capable. They can be used in different ways. The declaration of the filter function block instance `fbFilter` is always the same.

Multi-channel with two-dimensional signal arrays

The definition of a two-dimensional array has the advantage that it is universally valid, and the parameter `cChannels` can also be set to 1 in other projects. The channels in TwinCAT 3 Scope can also be selected individually, so that all channels can be viewed independently of each other.

Display of the multidimensional array in the "Target Browser":



Sample:

```
VAR CONSTANT
    cChannels     : UINT := 2;
    cOversamples : UINT := 10;
END_VAR
VAR
    aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
    aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
END_VAR
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

Multi-channel with one-dimensional signal arrays

Alternatively, the sampling values of the different channels can be stored in a one-dimensional array. However, in this case recording the individual channels with TwinCAT 3 Scope is more difficult.

Sample:

```
VAR CONSTANT
     cChannels     : UINT := 2;
     cOversamples : UINT := 10;
END_VAR
VAR
     aInput   : ARRAY [1..cChannels*cOversamples] OF LREAL;
     aOutput : ARRAY [1..cChannels*cOversamples] OF LREAL;
END_VAR
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

One-channel application with oversamples

If only a single channel is considered, the input and output arrays can be declared as one-dimensional quantities.

```
VAR CONSTANT
     cChannels     : UINT := 1;
     cOversamples : UINT := 10;
END_VAR
VAR
     aInput   : ARRAY [1..cOversamples] OF LREAL;
     aOutput : ARRAY [1..cOversamples] OF LREAL;
END_VAR
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

One-channel application without oversamples

If only a single channel is considered and no oversampling is applied, the input and output variables can be declared as `LREAL`.

```
VAR CONSTANT
     cChannels     : UINT := 1;
     cOversamples : UINT := 1;
END_VAR
VAR
     fInput   : LREAL;
     fOutput : LREAL;
END_VAR
```

```
bSucceed := fbFilter.Call(ADR(fInput), SIZEOF(fInput), ADR(fOutput), SIZEOF(fOutput));
```

5.1.1 FB_FTR_IIRCoeff

The function block implements an IIR filter (Infinite Impulse Response filter). The general transfer function is:

$$G(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + ... + b_M z^{-M}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + ... + a_N z^{-N}}$$

The coefficients a_k and b_k are freely definable in the configuration structure. The numerator degree and denominator degree may be different. The denominator can be set to a_0 = 1 and a_k = 0 (for all k > 0), so that a FIR filter is configured. The filter specification is transferred with the structure ST_FTR_IIRCoeff.

See also: Digital filters [▸ 15]

Syntax

Declaration:

```
fbFilter : FB_FTR_IIRCoeff(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_IIRCoeff
VAR_INPUT
    stConfig         : ST_FTR_IIRCoeff;
END_VAR
VAR_OUTPUT
    bError           : BOOL;
    bConfigured      : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_IIRCoeff [▸ 76]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

5.1.1.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
    stConfig : ST_FTR_IIRCoeff;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_IIRCoeff [▶ 76]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Sample

```
(*Declaration without configuration*)
fbFilter : FB_FTR_IIRCoeff();
```

```
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.pCoefficientArrayAdr_A := ADR(aNewArray); (*change coefficients of denominator*)
    stParams.nCoefficientArraySize_A := SIZEOF(aNewArray);
    bSucceed          := fbFilter.Configure(stConfig := stParams);
    bReconfigure      := FALSE;
END_IF
```

5.1.1.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn         : POINTER TO LREAL;
     nSizeIn     : UDINT;
     pOut        : POINTER TO LREAL;
     nSizeOut    : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.1.3 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▸ 15]

Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.1.2 FB_FTR_IIRSos

The function block implements an IIR filter (**I**nfinite-**I**mpulse-**R**esponse filter). The transfer function is:

$$G(z) = \prod_{m=1}^{M} G_m(z) = \prod_{m=1}^{M} \frac{b_{0m} + b_{1m}z^{-1} + b_{2m}z^{-2}}{a_{0m} + a_{1m}z^{-1} + a_{2m}z^{-2}}$$

The coefficients of the transfer function are freely definable and can be transferred with the structure ST_FTR_IIRSos.

See also:

Digital filters [▶ 15]

Syntax

Declaration:

```
fbFilter : FB_FTR_IIRSos(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_IIRSos
VAR_INPUT
    stConfig         : ST_FTR_IIRSos;
END_VAR
VAR_OUTPUT
    bError           : BOOL;
    bConfigured      : BOOL;
    ipResultMessage  : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_IIRSos [▶ 77]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

See also:

FB_FTR_IIRSpec [▸ 35]

Filter types and parameterization [▸ 17]

5.1.2.1 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn        : POINTER TO LREAL;
     nSizeIn    : UDINT;
     pOut       : POINTER TO LREAL;
     nSizeOut   : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.2.2 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▸ 15]

Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.1.2.3 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
    stConfig : ST_FTR_IIRSos;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_IIRSos [▸ 77]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Sample

```
(*Declaration without configuration*)
fbFilter : FB_FTR_IIRSos();
```

```
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.pCoefficientArrayAdr_Sos := ADR(aNewArray); (*change coefficients*)
    stParams.nCoefficientArraySize_Sos := SIZEOF(aNewArray);
    bSucceed          := fbFilter.Configure(stConfig := stParams);
    bReconfigure      := FALSE;
END_IF
```

5.1.3 FB_FTR_IIRSpec

The function block implements an IIR filter (Infinite Impulse Response filter).

The filter coefficients of the transfer function are calculated internally on the basis of the filter specification transferred in the form of biquads. The filter specification is transferred with the structure ST_FTR_IIRSpec. Filters of type Butterworth, Chebyshev or Bessel can be specified. Low-pass, high-pass, band-pass and band-stop filters can be defined in each case.

See also:

Digital filters [▸ 15], Filter types and parameterization [▸ 17]

Syntax

Declaration:

```
fbFilter : FB_FTR_IIRSpec(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_IIRSpec
VAR_INPUT
    stConfig        : ST_FTR_IIRSpec;
END_VAR
VAR_OUTPUT
    bError          : BOOL;
    bConfigured     : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_IIRSpec [▸ 77]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition lo-cation	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

See also:

Filter types and parameterization [▸ 17]

FB_FTR_IIRSos [▸ 32]

5.1.3.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
     stConfig : ST_FTR_IIRSpec;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_IIRSpec [▸ 77]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Sample

```
(*Declaration without configuration*)
fbFilter : FB_FTR_IIRSpec();
```

```
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.fCutoff := 50; (*change cutoff freq.*)
    bSucceed         := fbFilter.Configure(stConfig := stParams);
    bReconfigure     := FALSE;
END_IF
```

5.1.3.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn        : POINTER TO LREAL;
     nSizeIn    : UDINT;
```

```
    pOut        : POINTER TO LREAL;
    nSizeOut    : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.3.3 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▶ 15]

Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.1.4 FB_FTR_MovAvg

The function block implements a moving average filter. The filter calculates the mean value of M input values x[n-k] with k = 0 .. M - 1.

$$y[n] = \frac{1}{M}(x[n] + x[n-1] + ...x[n-M+1])$$

The filter specification is transferred with the structure ST_FTR_MovAvg.

Syntax

Declaration:

```
fbFilter : FB_FTR_MovAvg(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_MovAvg
VAR_INPUT
    stConfig        : ST_FTR_MovAvg;
END_VAR
VAR_OUTPUT
    bError          : BOOL;
    bConfigured     : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_MovAvg [▶ 78]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition lo-cation	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

5.1.4.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
     stConfig : ST_FTR_MovAvg;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_MovAvg [▶ 78]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Sample

```
(*Declaration without configuration*)
fbFilter : FB_FTR_MovAvg ();
```

```
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.nSamplesToFilter := 11; (*change filter order*)
    bSucceed         := fbFilter.Configure(stConfig := stParams);
    bReconfigure     := FALSE;
END_IF
```

5.1.4.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn        : POINTER TO LREAL;
     nSizeIn    : UDINT;
     pOut       : POINTER TO LREAL;
     nSizeOut   : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array

Name	Type	Description
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.4.3 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▶ 15]

Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.1.5 FB_FTR_PT1

The function block FB_FTR_PT1 implements a first-order delay element (PT1 element) with the complex transfer function (Laplace space):

$$G(s) = K_p \frac{1}{1 + T_1 s}$$

The filter specification is transferred with the structure ST_FTR_PT1.

Syntax

Declaration:

```
fbFilter : FB_FTR_PT1(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_PT1
VAR_INPUT
    stConfig        : ST_FTR_PT1;
END_VAR
VAR_OUTPUT
    bError          : BOOL;
    bConfigured     : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_PT1 [▸ 78]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

5.1.5.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
     stConfig : ST_FTR_PT1;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_PT1 [▶ 78]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Sample

```
(*Declaration without configuration*)
fbFilter : FB_FTR_PT1 ();
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.fKp := 2; (*change gain*)
    bSucceed        := fbFilter.Configure(stConfig := stParams);
    bReconfigure    := FALSE;
END_IF
```

5.1.5.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn        : POINTER TO LREAL;
     nSizeIn    : UDINT;
     pOut       : POINTER TO LREAL;
     nSizeOut   : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.5.3 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▶ 15]

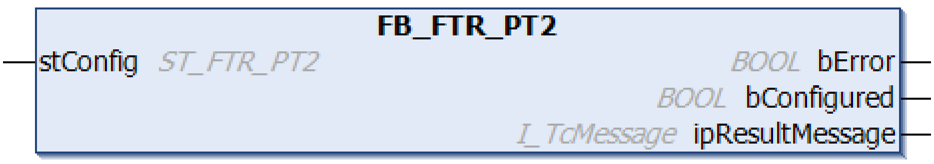
Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.1.6 FB_FTR_PT2



The function block FB_FTR_PT2 implements a second-order delay element (PT2 element) with the complex transfer function (Laplace space):

$$G(s) = K_p \frac{1}{1+T_1 s} \frac{1}{1+T_2 s}$$

The filter specification is transferred with the structure ST_FTR_PT2.

Syntax

Declaration:

```
fbFilter : FB_FTR_PT2(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_PT2
VAR_INPUT
    stConfig        : ST_FTR_PT2;
```

```
END_VAR
VAR_OUTPUT
    bError          : BOOL;
    bConfigured     : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_PT2 [▶ 79]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

5.1.6.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
     stConfig : ST_FTR_PT2;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_PT2 [▶ 79]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Sample

```
(*Declaration without configuration*)
fbFilter : FB_FTR_PT2 ();
```

```
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.fKp := 2; (*change gain*)
    bSucceed         := fbFilter.Configure(stConfig := stParams);
    bReconfigure     := FALSE;
END_IF
```

5.1.6.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn         : POINTER TO LREAL;
     nSizeIn     : UDINT;
     pOut        : POINTER TO LREAL;
     nSizeOut    : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.6.3 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▶ 15]

Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.1.7 FB_FTR_PT3

The function block FB_FTR_PT3 implements a third-order delay element (PT3 element) with the complex transfer function (Laplace space):

$$G(s) = K_p \frac{1}{1+T_1 s} \frac{1}{1+T_2 s} \frac{1}{1+T_3 s}$$

The filter specification is transferred with the structure ST_FTR_PT3.

Syntax

Declaration:

```
fbFilter : FB_FTR_PT3(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_PT3
VAR_INPUT
    stConfig          : ST_FTR_PT3;
END_VAR
VAR_OUTPUT
    bError            : BOOL;
    bConfigured       : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_PT3 [▶ 80]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

5.1.7.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
     stConfig : ST_FTR_PT3;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_PT3 [▸ 80]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Sample

```
(*Declaration without configuration*)
fbFilter : FB_FTR_PT3 ();
```

```
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.fKp := 2; (*change gain*)
    bSucceed         := fbFilter.Configure(stConfig := stParams);
    bReconfigure     := FALSE;
END_IF
```

5.1.7.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn         : POINTER TO LREAL;
     nSizeIn     : UDINT;
     pOut        : POINTER TO LREAL;
     nSizeOut    : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.7.3 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▶ 15]

Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.1.8 FB_FTR_PTn

The function block FB_FTR_PTn implements a nth-order delay element with the same time constants. The complex transfer function (Laplace space):

$$G(s) = K_p \frac{1}{(1 + T_1 s)^n}$$

The filter specification is transferred with the structure ST_FTR_PTn.

Syntax

Declaration:

```
fbFilter : FB_FTR_PTn(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_PTn
VAR_INPUT
    stConfig        : ST_FTR_PTn;
END_VAR
VAR_OUTPUT
    bError          : BOOL;
    bConfigured     : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_PTn [▶ 80]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

5.1.8.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
     stConfig : ST_FTR_PTn;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_PTn [▶ 80]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Sample

```
(*Declaration without configuration*)
fbFilter : FB_FTR_PTn ();
```

```
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.fKp := 2; (*change gain*)
```

```
    bSucceed          := fbFilter.Configure(stConfig := stParams);
    bReconfigure      := FALSE;
END_IF
```

5.1.8.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn         : POINTER TO LREAL;
     nSizeIn     : UDINT;
     pOut        : POINTER TO LREAL;
     nSizeOut    : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.8.3 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▸ 15]

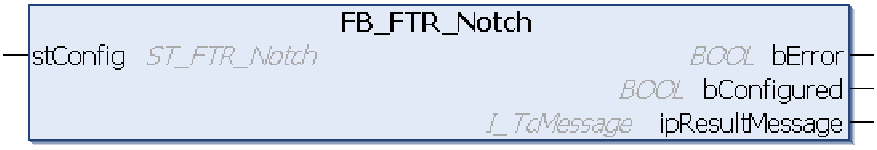
Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.1.9 FB_FTR_Notch



The function block FB_FTR_Notch implements a band-stop filter with narrow bandwidth. The complex transfer function (Laplace space):

$$G(s) = \frac{s^2 + \Omega_{\text{Notch}}^2}{(s^2 + \Omega_{\text{Notch}}/Q)^2 + \Omega_{\text{Notch}}^2}$$

$$\Omega_{\text{Notch}} = \frac{2}{T} \tan\left(\pi \frac{f_{\text{Notch}}}{f_{\text{sample}}}\right)$$

The filter specification is transferred with the structure ST_FTR_Notch.

Syntax

Declaration:

```
fbFilter : FB_FTR_Notch(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_Notch
VAR_INPUT
    stConfig         : ST_FTR_Notch;
END_VAR
VAR_OUTPUT
    bError           : BOOL;
    bConfigured      : BOOL;
    ipResultMessage  : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_Notch [▶ 81]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

5.1.9.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
     stConfig : ST_FTR_Notch;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_Notch [▸ 81]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Example

```
(*Declaration without configuration*)
fbFilter : FB_FTR_Notch ();
```

```
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE;
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.fQ := 20; (*change quality factor*)
    bSucceed        := fbFilter.Configure(stConfig := stParams);
    bReconfigure    := FALSE;
END_IF
```

5.1.9.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn         : POINTER TO LREAL;
     nSizeIn     : UDINT;
     pOut        : POINTER TO LREAL;
     nSizeOut    : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.9.3 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▶ 15]

Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.1.10 FB_FTR_LeadLag

The function block FB_FTR_LeadLag implements a first order phase correction element with the complex transfer function (Laplace space):

$$G(s) = \frac{1 + T_1 s}{1 + T_2 s}$$

The filter specification is transferred with the structure ST_FTR_LeadLag.

Syntax

Declaration:

```
fbFilter : FB_FTR_LeadLag(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_LeadLag
VAR_INPUT
    stConfig         : ST_FTR_LeadLag;
END_VAR
VAR_OUTPUT
    bError           : BOOL;
    bConfigured      : BOOL;
    ipResultMessage  : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_LeadLag [▶ 81]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

5.1.10.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
     stConfig : ST_FTR_LeadLag;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_Notch [▸ 81]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Example

```
(*Declaration without configuration*)
fbFilter : FB_FTR_LeadLag ();
```

```
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE;
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.fT1      := 0.0002; (*change time constant T1*)
    bSucceed          := fbFilter.Configure(stConfig := stParams);
    bReconfigure      := FALSE;
END_IF
```

5.1.10.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn        : POINTER TO LREAL;
     nSizeIn    : UDINT;
     pOut       : POINTER TO LREAL;
     nSizeOut   : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array

Name	Type	Description
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.10.3 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▶ 15]

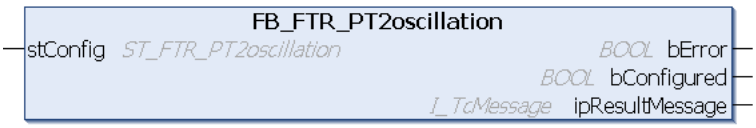
Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.1.11 FB_FTR_PT2oscillation



The function block FB_FTR_PT2oscillation implements an oscillating second-order delay element with the complex transfer function (Laplace space):

$$G(s) = \frac{K_p}{1 + 2\vartheta T_1 s + (T_1 s)^2}$$

The filter specification is transferred with the structure ST_FTR_PT2oscillation.

Syntax

Declaration:

```
fbFilter : FB_FTR_PT2oscillation(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_PT2oscillation
VAR_INPUT
    stConfig        : ST_FTR_PT2oscillation;
END_VAR
VAR_OUTPUT
    bError          : BOOL;
    bConfigured     : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_PT2oscillation [▸ 82]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition lo-cation	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

5.1.11.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
     stConfig : ST_FTR_PT2oscillation;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_PT2oscillation [▸ 82]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Example

```
(*Declaration without configuration*)
fbFilter : FB_FTR_PT2oscillation ();
```

```
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE;
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.fTheta   := 0.7; (*change damping factor*)
    bSucceed          := fbFilter.Configure(stConfig := stParams);
    bReconfigure      := FALSE;
END_IF
```

5.1.11.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn         : POINTER TO LREAL;
     nSizeIn     : UDINT;
     pOut        : POINTER TO LREAL;
     nSizeOut    : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.11.3 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▸ 15]

Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.1.12 FB_FTR_PTt

The function block FB_FTR_PTt implements a dead time element (PTt element) with the complex transfer function (Laplace space):

$$G(s) = K_p \cdot e^{-T_t s}$$

The filter specification is transferred with the structure ST_FTR_PTt.

Syntax

Declaration:

```
fbFilter : FB_FTR_PTt(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_PTt
VAR_INPUT
    stConfig        : ST_FTR_PTt;
END_VAR
VAR_OUTPUT
    bError          : BOOL;
```

```
    bConfigured     : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_PTt [▶ 82]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

5.1.12.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
     stConfig : ST_FTR_PTt;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_PTt [▶ 82]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Sample

```
(*Declaration without configuration*)
fbFilter : FB_FTR_PTt ();
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.fKp := 2; (*change gain*)
    bSucceed         := fbFilter.Configure(stConfig := stParams);
    bReconfigure     := FALSE;
END_IF
```

5.1.12.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn         : POINTER TO LREAL;
     nSizeIn     : UDINT;
     pOut        : POINTER TO LREAL;
     nSizeOut    : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.12.3 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▶ 15]

Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.1.13 FB_FTR_Median

The function block implements a median filter. The median is the mean value of a list of values ordered by size. This means that one half of the collected data values is smaller than the median value, the other half is larger.

The filter specification is transferred with the structure ST_FTR_Median.

Syntax

Declaration:

```
fbFilter : FB_FTR_Median(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_Median
VAR_INPUT
    stConfig         : ST_FTR_Median;
END_VAR
VAR_OUTPUT
    bError           : BOOL;
    bConfigured      : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_Median [▶ 83]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

5.1.13.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
     stConfig : ST_FTR_Median;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_Median [▶ 83]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Sample

```
(*Declaration without configuration*)
fbFilter : FB_FTR_Median ();
```

```
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.nSamplesToFilter := 11; (*change filter order*)
    bSucceed          := fbFilter.Configure(stConfig := stParams);
    bReconfigure      := FALSE;
END_IF
```

5.1.13.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn         : POINTER TO LREAL;
     nSizeIn     : UDINT;
     pOut        : POINTER TO LREAL;
     nSizeOut    : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.13.3 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▸ 15]

Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.1.14 FB_FTR_ActualValue

The function block allows a measured input value to be checked for plausibility and filtered.

If the difference between two sampling values in sequence is larger than the specified window, *fDeltaMax*, then the current input value is suppressed for a maximum of three cycles. During this time the output value is linearly extrapolated from the previous input values. If the specified window is exceeded for more than three cycles, the output will again follow the input value.

$$y_k = \begin{cases} x_k & : \ |x_k - x_{k-1}| <= fDeltaMax \\ y_{temp} & : \ |x_k - x_{k-1}| > fDeltaMax \end{cases}$$

$$y_{temp} = \begin{cases} y_{k-1} + \frac{(y_{k-1} - y_{k-2}) + (y_{k-2} - y_{k-3})}{2} & : \ n <= 3 \\ y_{k-1} + \frac{(x_k - y_{k-1})}{2} & : \ n = 4 \\ y_k & : \ n = 5 \end{cases}$$

Variable n is incremented if $|x_k - x_{k-1}| > fDeltaMax$.

The filter specification is transferred with the structure ST_FTR_ActualValue [▶ 83].

Syntax

Declaration:

```
fbFilter : FB_FTR_ActualValue(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_ActualValue
VAR_INPUT
    stConfig        : ST_FTR_ActualValue;
END_VAR
VAR_OUTPUT
    bError          : BOOL;
    bConfigured     : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_ActualValue [▶ 83]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling.

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.
GetFilterActive()	Local	Indicates which elements of the input signal have been changed.
GetFilterActiveTimestamps()	Local	Shows when the filter was last active.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical.
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event.
nTimestamp	UINT	Set	Local	0	Timestamp of the oldest input value of the next Call().
tSamplePeriod	LTIME	Set	Local	0	Time difference between two input values.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

5.1.14.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
     stConfig : ST_FTR_Median;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_Median [▸ 83]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Sample

```
(*Declaration without configuration*)
fbFilter : FB_FTR_Median ();
```

```
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.nSamplesToFilter := 11; (*change filter order*)
    bSucceed         := fbFilter.Configure(stConfig := stParams);
    bReconfigure     := FALSE;
END_IF
```

5.1.14.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn        : POINTER TO LREAL;
     nSizeIn    : UDINT;
     pOut       : POINTER TO LREAL;
     nSizeOut   : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.14.3 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▶ 15]

Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.1.14.4 GetFilterActive

The method calculates which elements of the input signal were changed, starting from the last Call().

Syntax

```
METHOD GetFilterActive: BOOL
VAR_INPUT
     pFilterActive         : POINTER TO BOOL;
     nSizeGetFilterActive : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pFilterActive	POINTER TO BOOL	Address of the input array
nSizeGetFilterActive	UDINT	Size of the input array

Return value

Name	Type	Description
GetFilterActive	BOOL	Returns TRUE if active state was calculated.

Sample

```
aFilterActive : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF BOOL;
```

```
bSucceed := fbFilter.GetFilterActive(ADR(aFilterActive),SIZEOF(aFilterActive));
```

5.1.14.5 GetFilterActiveTimestamps

The method calculates when the filter was last active, based on the last Call().

Syntax

```
METHOD GetFilterActiveTimestamps: BOOL
VAR_INPUT
     pFilterActiveTimestamps : POINTER TO BOOL;
     nSizeFilterActiveTimestamps : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pFilterActiveTimestamps	POINTER TO ULINT	Address of the input array
nSizeFilterActiveTimestamps	UDINT	Size of the input array

Return value

Name	Type	Description
GetFilterActiveTimestamps	BOOL	Returns TRUE if the timestamps have been calculated.

Sample

```
aFilterActiveTimeStamps : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF ULINT;
```

```
bSucceed := fbFilter. GetFilterActiveTimestamps
(ADR(aFilterActiveTimeStamps),SIZEOF(aFilterActiveTimeStamps));
```

5.1.15 FB_FTR_Gaussian

The function block FB_FTR_Gaussian implements a Gaussian filter. The filter calculates the weighted average of M input values. The weights are calculated using a Gaussian function, where the standard deviation σ determines the cut-off frequency of the filter.

The cut-off frequency f_c is defined by the standard deviation σ and the sampling rate f_s such that the normalized amplitude response takes the value 1/sqrt(2) ≈ -3dB at the cut-off frequency.

$$f_c = \frac{f_s}{2\pi\sigma} \cdot \sqrt{\ln(2)}$$

The filter specification is transferred with the structure ST_FTR_Gaussian [▶ 84].

Syntax

Declaration:

```
fbFilter : FB_FTR_Gaussian(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_FTR_Gaussian
VAR_INPUT
    stConfig         : ST_FTR_Gaussian;
```

```
END_VAR
VAR_OUTPUT
    bError           : BOOL;
    bConfigured      : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_Gaussian [▶ 84]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and filter configuration.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE if eTraceLevel = Critical
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.25	PC or CX (x64, x86)	Tc3_Filter

5.1.15.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
     stConfig : ST_FTR_PT1;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_FTR_PT1 [▶ 78]	Structure for configuring the filter behavior

Return value

Name	Type	Description
Configure	BOOL	TRUE if the filter instance was configured successfully.

Sample

```
(*Declaration without configuration*)
fbFilter : FB_FTR_PT1 ();
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.fKp := 2; (*change gain*)
    bSucceed         := fbFilter.Configure(stConfig := stParams);
    bReconfigure     := FALSE;
END_IF
```

5.1.15.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn        : POINTER TO LREAL;
     nSizeIn    : UDINT;
     pOut       : POINTER TO LREAL;
     nSizeOut   : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
aOutput : ARRAY [1..cChannels] OF ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

5.1.15.3 Reset

The method resets the internal status of the filter. The current output value y[n] consists of the current input value x[n] and past input and output values (x[n-k] and y[n-k], k > 0). Resetting the function block returns the filter to its original state, i.e. without any influence from the past. The filter is thus reset to the last configuration state.

See also: Digital filters [▸ 15]

Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the filter was successfully reset.

5.2 Data types

5.2.1 Configuration structures

General description

An individual configuration structure ST_FTR_<type> exists for each function block FB_FTR_<type>. In the configuration structure all parameters are defined that are required for the calculation of the transfer function, the input and output variables (size and form of the arrays) as well as the internal states.

Common parameters

All configuration structures ST_FTR_<type> contain the following parameters:

Parameter	Type	Description
`nOversamples`	UDINT	Number of oversamples > 0
`nChannels`	UDINT	Number of channels > 0 & < 101
`pInitialValues`	POINTER TO LREAL	Pointer to array with initial values (optional)
`nInitialValuesSize`	UDINT	Size of the array with initial values in BYTE (optional)

nOversamples and nChannels

The parameters `nOversamples` and `nChannels` describe the size of the input and output arrays that are transferred when the `Call()` method is called. If, for example, 10 oversamples and 3 channels are used, the `Call()` method expects an array with 30 elements at the input and output (see Oversampling and channels [▸ 28]).

The parameter `nChannels` describes the number of parallel signal channels to be processed with a call. The parameter `nOversamples` describes the number of samples that occur for each individual channel and each call of the `Call()` method.

Inital Values

The optional parameters `pInitialValues` and `nInitialValuesSize` are used to define the internal state of the filter after its configuration. In practice, the parameters are used to reduce the settling time of the configured filter by introducing prior knowledge. For this purpose, the historical values y[n-k] and x[n-k] with k > 0 in the difference equation must be set in a targeted manner.

The following structure of the difference equation applies to the function blocks FB_FTR_IIRCoeff [▶ 30], FB_FTR_MovAvg [▶ 38], FB_FTR_PT1 [▶ 41], FB_FTR_PT2 [▶ 44] and FB_FTR_PT3 [▶ 47]:

$$a_0 y[n] + \sum_{k=1}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k]$$

For a more compact illustration, the following notations are used for the historical values: $x_{i,k}$ and $y_{i,k}$, where channel i and time delay k > 0. Sample: Channel i = 2 and delay k = 3 corresponds to $y_{2,3} = y_2[n-3]$.

Application options:

- `InitialValues = nullptr`
  All historical values $y_{i,k}$ and $x_{i,k}$ with k > 0 are set to zero for all channels i. This corresponds to the behavior if the entries of the structure are not used.
- `InitialValues = P`
  All historical values $x_{i,k}$, where k > 0, are set for all channels i to the value P. All historical values $y_{i,k}$ with k > 0 are set to V*P for all channels i, where V corresponds to the DC gain of the filter.
  Accordingly, the filter is in steady state for all channels with respect to a constant input signal x[n] = P.
- `InitialValues =` $[x_{1,1}, x_{1,2}, \ldots, x_{1,M}, y_{1,1}, y_{1,2}, \ldots, y_{1,N}]$
  All historical values $x_{i,k}$ with k > 0 and $y_{i,k}$ with k > 0 are set individually, but to the same values for all channels i.
- `InitialValues =` $[x_{1,1}, x_{1,2}, \ldots, x_{1,M}, y_{1,1}, y_{1,k}, \ldots, y_{1,N},$
  $x_{2,1}, x_{2,2}, \ldots, x_{2,M}, y_{2,1}, y_{2,2}, \ldots, y_{2,N},$
  ....
  $x_{I,1}, x_{I,2}, \ldots, x_{I,M}, y_{I,1}, y_{I,2}, \ldots, y_{I,N}]$
  All historical values $x_{i,k}$ and $y_{i,k}$ are set individually for every channel i = 1..I.

For the function blocks FB_FTR_IIRSos [▶ 32] and FB_FTR_IIRSpec [▶ 35], the transfer function is calculated in second-order sections (SOS):

$$G(z) = \prod_{m=1}^{M} G_m(z) = \prod_{m=1}^{M} \frac{b_{0m} + b_{1m}z^{-1} + b_{2m}z^{-2}}{a_{0m} + a_{1m}z^{-1} + a_{2m}z^{-2}}$$

The compact illustration is extended accordingly by m, which specifies the biquad: $x^{(m)}_{i,k}$ and $y^{(m)}_{i,k}$. Sample: Channel i = 2 and delay k = 1 of the fourth biquad (m = 4) corresponds to $y^{(4)}_{2,1} = y^{(4)}_2[n-1]$. Each biquad can use only two historical values here; accordingly, k = 1.2.

Therefore, the structuring for the input and output values is different there.

Application options:

- `InitialValues = nullptr`
  Same behavior as described above.
- `InitialValues = P`
  Same behavior as described above.
- `InitialValues =` $[x^{(1)}_{1,1}, x^{(1)}_{1,2}, x^{(2)}_{1,1}, x^{(2)}_{1,2}, \ldots, x^{(M)}_{1,1}, x^{(M)}_{1,2},$
  $y^{(1)}_{1,1}, y^{(1)}_{1,2}, y^{(2)}_{1,1}, y^{(2)}_{1,2}, \ldots, y^{(M)}_{1,1}, y^{(M)}_{1,2}]$
  All historical values $x^{(m)}_{i,k}$ and $y^{(m)}_{i,k}$ are individually set, but the same for all channels i.
- `InitialValues =` $[x^{(1)}_{1,1}, x^{(1)}_{1,2}, x^{(2)}_{1,1}, x^{(2)}_{1,2}, \ldots, x^{(M)}_{1,1}, x^{(M)}_{1,2},$
  $y^{(1)}_{1,1}, y^{(1)}_{1,2}, y^{(2)}_{1,1}, y^{(2)}_{1,2}, \ldots, y^{(M)}_{1,1}, y^{(M)}_{1,2},$
  $x^{(1)}_{2,1}, x^{(1)}_{2,2}, x^{(2)}_{2,1}, x^{(2)}_{2,2}, \ldots, x^{(M)}_{2,1}, x^{(M)}_{2,2},$
  $y^{(1)}_{2,1}, y^{(1)}_{2,2}, y^{(2)}_{2,1}, y^{(2)}_{2,2}, \ldots, y^{(M)}_{2,1}, y^{(M)}_{2,2},$
  ....
  $x^{(1)}_{I,1}, x^{(1)}_{I,2}, x^{(2)}_{I,1}, x^{(2)}_{I,2}, \ldots, x^{(M)}_{I,1}, x^{(M)}_{I,2},$
  $y^{(1)}_{I,1}, y^{(1)}_{I,2}, y^{(2)}_{I,1}, y^{(2)}_{I,2}, \ldots, y^{(M)}_{I,1}, y^{(M)}_{I,2}]$
  All historical values $x^{(m)}_{i,k}$ and $y^{(m)}_{i,k}$ are set individually for every channel i = 1..I.

The function block FB_FTR_PTn [▸ 50] is also described as a cascaded filter, but as a first-order cascaded filter. Accordingly, the same consideration as for the SOS blocks applies here, but with k = 1. For a PTn filter where n = M, the following then applies:

- `InitialValues = nullptr`
  Same behavior as described above.
- `InitialValues = P`
  Same behavior as described above.
- `InitialValues` = $[x^{(1)}_{1,1}, x^{(2)}_{1,1}, \ldots, x^{(M)}_{1,1}, y^{(1)}_{1,1}, y^{(2)}_{1,1}, \ldots, y^{(M)}_{1,1}]$
  All historical values $x^{(m)}_{i,k}$ and $y^{(m)}_{i,k}$ are individually set, but the same for all channels i.
- `InitialValues` = $[x^{(1)}_{1,1}, x^{(2)}_{1,1}, \ldots, x^{(M)}_{1,1},$
  $y^{(1)}_{1,1}, y^{(2)}_{1,1}, \ldots, y^{(M)}_{1,1},$
  $x^{(1)}_{2,1}, x^{(2)}_{2,1}, \ldots, x^{(M)}_{2,1},$
  $y^{(1)}_{2,1}, y^{(2)}_{2,1}, \ldots, y^{(M)}_{2,1},$
  ....
  $x^{(1)}_{I,1}, x^{(2)}_{I,1}, \ldots, x^{(M)}_{I,1},$
  $y^{(1)}_{I,1}, y^{(2)}_{I,1}, \ldots, y^{(M)}_{I,1}]$
  All historical values $x^{(m)}_{i,k}$ and $y^{(m)}_{i,k}$ are set individually for every channel i = 1..I.

5.2.1.1 ST_FTR_IIRCoeff

Configuration structure for the function block FB_FTR_IIRCoeff [▸ 30].

Syntax

Definition:

```
TYPE ST_FTR_IIRCoeff :
STRUCT
    pCoefficientArrayAdr_A  : POINTER TO LREAL;
    nCoefficientArraySize_A : UDINT;
    pCoefficientArrayAdr_B  : POINTER TO LREAL;
    nCoefficientArraySize_B : UDINT;
    bReset                  : BOOL := TRUE;
    nOversamples            : UDINT;
    nChannels               : UDINT;
    pInitialValues          : POINTER TO LREAL;
    nInitialValuesSize      : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
pCoefficientArrayAdr_A	Pointer to LREAL	Pointer to an array with filter coefficients a_k (denominator) $[a_0, a_1, a_2, \ldots, a_N]$
nCoefficientArraySize_A	UDINT	Size of the array $[a_0, a_1, a_2, \ldots, a_N]$ in BYTE
pCoefficientArrayAdr_B	Pointer to LREAL	Pointer to an array with filter coefficients b_k (numerator) $[b_0, b_1, b_2, \ldots, b_M]$
nCoefficientArraySize_B	UDINT	Size of the array $[b_0, b_1, b_2, \ldots, b_M]$ in BYTE
bReset	BOOL	If TRUE, a reset is performed when the filter is configured. If FALSE, the historical values x[n-k] and y[n-k] are not reset.
nOversamples	UDINT	Number of oversamples (greater than zero)
nChannels	UDINT	Number of channels (greater than zero and less than 101)
pInitialValues	Pointer to LREAL	Pointer to array with initial values (optional)
nInitialValuesSize	UDINT	Size of the array with initial values in BYTE (optional)

5.2.1.2 ST_FTR_IIRSos

Configuration structure for the function block FB_FTR_IIRSos [▸ 32].

Syntax

Definition:

```
TYPE ST_FTR_IIRSos :
STRUCT
    pCoefficientArrayAdr_Sos  : POINTER TO LREAL;
    nCoefficientArraySize_Sos : UDINT;
    bReset                    : BOOL := TRUE;
    nOversamples              : UDINT;
    nChannels                 : UDINT;
    pInitialValues            : POINTER TO LREAL;
    nInitialValuesSize        : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
pCoefficientArrayAdr_Sos	Pointer to LREAL	Pointer to an array with filter coefficients [b_{01}, b_{11}, b_{21},a_{01}, a_{11}, a_{21}, b_{02}, b_{12}, b_{22},a_{02}, a_{12}, a_{22}, …. b_{0M}, b_{1M}, b_{2M},a_{0M}, a_{1M}, a_{2M}]
nCoefficientArraySize_Sos	UDINT	Size of the array with filter coefficients in BYTES
bReset	BOOL	If TRUE, a reset is performed when the filter is configured. If FALSE, the historical values x[n-k] and y[n-k] are not reset.
nOversamples	UDINT	Number of oversamples (greater than zero)
nChannels	UDINT	Number of channels (greater than zero and less than 101)
pInitialValues	Pointer to LREAL	Pointer to array with initial values (optional)
nInitialValuesSize	UDINT	Size of the array with initial values in BYTE (optional)

An example of a filter with 3 biquads might look like this:

```
aCoeffs_Sos : ARRAY [1..3] OF ARRAY [1..6] OF LREAL :=
              [ [1, 1, 1, 1, 1, 1],   // [b01,b11,b21,a01,a11,a21]
              [ [1, 1, 1, 1, 1, 1],   // [b02,b12,b22,a02,a12,a22]
              [ [1, 1, 1, 1, 1, 1]];  // [b03,b13,b23,a03,a13,a23
```

5.2.1.3 ST_FTR_IIRSpec

Configuration structure for the function block FB_FTR_IIRSpec [▸ 35].

Syntax

Definition:

```
TYPE ST_FTR_IIRSpec :
STRUCT
    eFilterName        : E_FTR_Name;
    eFilterType        : E_FTR_Type;
    nFilterOrder       : UDINT;
    fCutoff            : LREAL;
    fBandwidth         : LREAL;
    fPassBandRipple    : LREAL;
    fSamplingRate      : LREAL;
    nOversamples       : UDINT;
    nChannels          : UDINT;
    pInitialValues     : POINTER TO LREAL;
    nInitialValuesSize : UDINT;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Description
eFilterName	E_FTR_Name [▶ 84]	Describes the filter implementation (Butterworth, Chebyshev, Bessel)
eFilterType	E_FTR_Type [▶ 85]	Describes the filter type (high-pass, low-pass, …)
nFilterOrder	UDINT	Filter order (max. 20 for high-pass and low-pass, max. 10 for band-pass and band-stop)
fCutoff	LREAL	Cut-off frequency in Hz (greater than 0 and less than fSamplingRate/2)
fBandwidth	LREAL	Bandwidth in Hz with respect to band-pass and band-stop.
fPassbandRipple	LREAL	Passband ripple of the amplitude response in the passband of the filter in dB (greater than 0)
fSamplingRate	LREAL	Sampling rate f_s in Hz
nOversamples	UDINT	Number of oversamples (greater than zero)
nChannels	UDINT	Number of channels (greater than zero and less than 101)
pInitialValues	Pointer to LREAL	Pointer to array with initial values (optional)
nInitialValuesSize	UDINT	Size of the array with initial values in BYTE (optional)

Notes on the influence of the parameters on the Butterworth, Chebyshev and Bessel type filters can be found here: Filter types and parameterization [▶ 17].

5.2.1.4 ST_FTR_MovAvg

Configuration structure for the function block FB_FTR_MovAvg [▶ 38].

Syntax

Definition:

```
TYPE ST_FTR_MovAvg :
STRUCT
    nSamplesToFilter   : UDINT:
    nOversamples       : UDINT;
    nChannels          : UDINT;
    pInitialValues     : POINTER TO LREAL;
    nInitialValuesSize : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
nSamplesToFilter	UDINT	Number of samples for calculating the moving average (often referred to as window size)
nOversamples	UDINT	Number of oversamples (greater than zero)
nChannels	UDINT	Number of channels (greater than zero and less than 101)
pInitialValues	Pointer to LREAL	Pointer to array with initial values (optional)
nInitialValuesSize	UDINT	Size of the array with initial values in BYTE (optional)

5.2.1.5 ST_FTR_PT1

Configuration structure for the function block FB_FTR_PT1 [▶ 41].

Syntax

Definition:

```
TYPE ST_FTR_PT1 :
STRUCT
    fKp                : LREAL;
    fT1                : LREAL;
    fSamplingRate      : LREAL;
    nOversamples       : UDINT;
    nChannels          : UDINT;
    pInitialValues     : POINTER TO LREAL;
    nInitialValuesSize : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
fKp	LREAL	Gain factor (greater than zero)
fT1	LREAL	Time constant T_1 in seconds (greater than zero)
fSamplingRate	LREAL	Sampling rate f_s in Hz (greater than zero)
nOversamples	UDINT	Number of oversamples (greater than zero)
nChannels	UDINT	Number of channels (greater than zero and less than 101)
pInitialValues	Pointer to LREAL	Pointer to array with initial values (optional)
nInitialValuesSize	UDINT	Size of the array with initial values in BYTE (optional)

5.2.1.6 ST_FTR_PT2

Configuration structure for the function block FB_FTR_PT2 [▶ 44].

Syntax

Definition:

```
TYPE ST_FTR_PT2 :
STRUCT
    fKp                : LREAL;
    fT1                : LREAL;
    fT2                : LREAL;
    fSamplingRate      : LREAL;
    nOversamples       : UDINT;
    nChannels          : UDINT;
    pInitialValues     : POINTER TO LREAL;
    nInitialValuesSize : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
fKp	LREAL	Gain factor (greater than zero)
fT1	LREAL	Time constant T_1 in seconds (greater than zero)
fT2	LREAL	Time constant T_2 in seconds (greater than zero)
fSamplingRate	LREAL	Sampling rate f_s in Hz (greater than zero)
nOversamples	UDINT	Number of oversamples (greater than zero)
nChannels	UDINT	Number of channels (greater than zero and less than 101)
pInitialValues	Pointer to LREAL	Pointer to array with initial values (optional)
nInitialValuesSize	UDINT	Size of the array with initial values in BYTE (optional)

5.2.1.7 ST_FTR_PT3

Configuration structure for the function block FB_FTR_PT3 [▸ 47].

Syntax

Definition:

```
TYPE ST_FTR_PT3 :
STRUCT
    fKp                 : LREAL;
    fT1                 : LREAL;
    fT2                 : LREAL;
    fT3                 : LREAL;
    fSamplingRate       : LREAL;
    nOversamples        : UDINT;
    nChannels           : UDINT;
    pInitialValues      : POINTER TO LREAL;
    nInitialValuesSize  : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
fKp	LREAL	Gain factor (greater than zero)
fT1	LREAL	Time constant T_1 in seconds (greater than zero)
fT2	LREAL	Time constant T_2 in seconds (greater than zero)
fT3	LREAL	Time constant T_3 in seconds (greater than zero)
fSamplingRate	LREAL	Sampling rate f_s in Hz (greater than zero)
nOversamples	UDINT	Number of oversamples (greater than zero)
nChannels	UDINT	Number of channels (greater than zero and less than 101)
pInitialValues	Pointer to LREAL	Pointer to array with initial values (optional)
nInitialValuesSize	UDINT	Size of the array with initial values in BYTE (optional)

5.2.1.8 ST_FTR_PTn

Configuration structure for the function block FB_FTR_PTn [▸ 50].

Syntax

Definition:

```
TYPE ST_FTR_PTn :
STRUCT
    fKp                 : LREAL;
    fT1                 : LREAL;
    nOrder              : UDINT;
    fSamplingRate       : LREAL;
    nOversamples        : UDINT;
    nChannels           : UDINT;
    pInitialValues      : POINTER TO LREAL;
    nInitialValuesSize  : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
fKp	LREAL	Gain factor (greater than zero)
fT1	LREAL	Time constant T_1 in seconds (greater than zero)
nOrder	UDINT	Order of the filter (1..10)
fSamplingRate	LREAL	Sampling rate f_s in Hz (greater than zero)

Name	Type	Description
nOversamples	UDINT	Number of oversamples (greater than zero)
nChannels	UDINT	Number of channels (greater than zero and less than 101)
pInitialValues	Pointer to LREAL	Pointer to array with initial values (optional)
nInitialValuesSize	UDINT	Size of the array with initial values in BYTE (optional)

5.2.1.9 ST_FTR_Notch

Configuration structure for the function block FB_FTR_Notch [▸ 53].

Syntax

Definition:

```
TYPE ST_FTR_Notch :
STRUCT
    fNotchFrequency    : LREAL;
    fQ                 : LREAL;
    fSamplingRate      : LREAL;
    nOversamples       : UDINT;
    nChannels          : UDINT;
    pInitialValues     : POINTER TO LREAL;
    nInitialValuesSize : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
fNotchFrequency	LREAL	Notch frequency in Hz (greater than 0 and less than fSamplingRate/2)
fQ	LREAL	Q-factor = fNotchFrequency/bandwidth (greater than zero)
fSamplingRate	LREAL	Sampling rate f_s in Hz (greater than zero)
nOversamples	UDINT	Number of oversamples (greater than zero)
nChannels	UDINT	Number of channels (greater than zero and less than 101)
pInitialValues	Pointer to LREAL	Pointer to array with initial values (optional)
nInitialValuesSize	UDINT	Size of the array with initial values in BYTE (optional)

Configuration example

For example, you want to create a notch filter that strongly attenuates the frequency 100 Hz. In doing so, you allow a one-sided bandwidth of 10 Hz around these 100 Hz, i.e. you expect the -3 dB limit at 90 Hz and 110 Hz. Define for this:

```
fNotchFrequency := 100;
```

```
fQ := 10; (* B = fNotchFrequency/fQ = 10 Hz *)
```

5.2.1.10 ST_FTR_LeadLag

Configuration structure for the function block FB_FTR_PT2 [▸ 44].

Syntax

Definition:

```
TYPE ST_FTR_PT2 :
STRUCT
    fT1                : LREAL;
    fT2                : LREAL;
    fSamplingRate      : LREAL;
    nOversamples       : UDINT;
    nChannels          : UDINT;
```

```
    pInitialValues     : POINTER TO LREAL;
    nInitialValuesSize : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
fT1	LREAL	Time constant T_1 in seconds (greater than zero)
fT2	LREAL	Time constant T_2 in seconds (greater than zero)
fSamplingRate	LREAL	Sampling rate f_s in Hz (greater than zero)
nOversamples	UDINT	Number of oversamples (greater than zero)
nChannels	UDINT	Number of channels (greater than zero and less than 101)
pInitialValues	Pointer to LREAL	Pointer to array with initial values (optional)
nInitialValuesSize	UDINT	Size of the array with initial values in BYTE (optional)

5.2.1.11 ST_FTR_PT2oscillation

Configuration structure for the function block FB_FTR_PT2oscillation [▶ 58].

Syntax

Definition:

```
TYPE ST_FTR_PT2 :
STRUCT
    fKp                : LREAL;
    fT1                : LREAL;
    fTheta             : LREAL;
    fSamplingRate      : LREAL;
    nOversamples       : UDINT;
    nChannels          : UDINT;
    pInitialValues     : POINTER TO LREAL;
    nInitialValuesSize : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
fKp	LREAL	Gain factor (greater than zero)
fT1	LREAL	Time constant T_1 in seconds (greater than zero)
fTheta	LREAL	Damping factor (greater than zero)
fSamplingRate	LREAL	Sampling rate f_s in Hz (greater than zero)
nOversamples	UDINT	Number of oversamples (greater than zero)
nChannels	UDINT	Number of channels (greater than zero and less than 101)
pInitialValues	Pointer to LREAL	Pointer to array with initial values (optional)
nInitialValuesSize	UDINT	Size of the array with initial values in BYTE (optional)

5.2.1.12 ST_FTR_PTt

Configuration structure for the function block FB_FTR_PTt [▶ 61].

Syntax

Definition:

```
TYPE ST_FTR_PTt :
STRUCT
    fKp                : LREAL;
```

```
    fTt                 : LREAL;
    fSamplingRate       : LREAL;
    nOversamples        : UDINT;
    nChannels           : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
fKp	LREAL	Gain factor (greater than zero)
fTt	LREAL	Dead time T_t in seconds (greater than zero)
fSamplingRate	LREAL	Sampling rate f_s in Hz (greater than zero)
nOversamples	UDINT	Number of oversamples (greater than zero and multiple of 1/fSamplingRate)
nChannels	UDINT	Number of channels (greater than zero and less than 101)

5.2.1.13 ST_FTR_Median

Configuration structure for the function block FB_FTR_Median [▶ 64].

Syntax

Definition:

```
TYPE ST_FTR_Median :
STRUCT
    nSamplesToFilter    : UDINT;
    nOversamples        : UDINT;
    nChannels           : UDINT;
    pInitialValues      : POINTER TO LREAL;
    nInitialValuesSize : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
nSamplesToFilter	UDINT	Number of samples for calculating the moving average (often referred to as window size)
nOversamples	UDINT	Number of oversamples (greater than zero)
nChannels	UDINT	Number of channels (greater than zero and less than 101)
pInitialValues	Pointer to LREAL	Pointer to array with initial values (optional)
nInitialValuesSize	UDINT	Size of the array with initial values in BYTE (optional)

5.2.1.14 ST_FTR_ActualValue

Configuration structure for the function block FB_FTR_ActualValue [▶ 67].

Syntax

Definition:

```
TYPE ST_FTR_ActualValue :
STRUCT
    fDeltaMax              : LREAL;
    nOversamples        : UDINT;
    nChannels           : UDINT;
    pInitialValues      : POINTER TO LREAL;
    nInitialValuesSize : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
fDeltaMax	LREAL	Maximum difference between two input values in sequence (greater than or equal to zero)
nOversamples	UDINT	Number of oversamples (greater than zero)
nChannels	UDINT	Number of channels (greater than zero and less than 101)
pInitialValues	Pointer to LREAL	Pointer to array with initial values (optional)
nInitialValuesSize	UDINT	Size of the array with initial values in BYTE (optional)

5.2.1.15 ST_FTR_Gaussian

Configuration structure for the function block FB_FTR_Gaussian [▶ 71].

Syntax

Definition:

```
TYPE ST_FTR_Gaussian :
STRUCT
    fCutoff            : LREAL;
    fSamplingRate      : LREAL;
    nSamplesToFilter   : UDINT;
    nOversamples       : UDINT;
    nChannels          : UDINT;
    pInitialValues     : POINTER TO LREAL;
    nInitialValuesSize : UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
fCutoff	LREAL	Cut-off frequency in Hz (greater than 0 and less than fSamplingRate/2)
fSamplingRate	LREAL	Sampling rate f_s in Hz (greater than zero)
nSamplesToFilter	UDINT	Number of samples for calculating the sliding average value (often referred to as window size). At null, the number is calculated automatically (optional)
nOversamples	UDINT	Number of oversamples (greater than zero)
nChannels	UDINT	Number of channels (greater than zero and less than 101)
pInitialValues	Pointer to LREAL	Pointer to array with initial values (optional)
nInitialValuesSize	UDINT	Size of the array with initial values in BYTE (optional)

5.2.2 E_FTR_Name

ENUM for filter names.

Syntax

Definition:

```
TYPE ST_FTR_Name :
 (
    eButterworth := 1,
    eChebyshev   := 2,
    eBessel      := 3
 ) UDINT
END_TYPE
```

5.2.3 E_FTR_Type

ENUM for filter types.

Syntax

Definition:

```
TYPE ST_FTR_Type :
 (
    eLowPass     := 1,
    eHighPass    := 2,
    eBandPass    := 3,
    eBandStop    := 4
 ) UDINT
END_TYPE
```

6 Samples

6.1 Configuration of a filter with FB_FTR_IIRSpec

This sample shows how to configure a filter of type Butterworth with the function block FB_FTR_IIRSpec.

Download: https://infosys.beckhoff.com/content/1033/TF3680_TC3_Filter/Resources/5906979595/.zip (*.tnzip)

Description:

- The sample project consists of a TwinCAT PLC project and a measurement project.
- In the measurement project, two input signals and two output signals are configured in two axes.
- The two input signals are generated synthetically via a function generator called in the PLC. They are harmonic signals (sine) with a frequency of 250 Hz for the first channel and 300 Hz for the second channel. The signals are processed by a Butterworth type filter. The implemented filter has a cut-off frequency of 250 Hz.
- The MAIN PLC program is called by a task with a cycle time of 1 ms.

Implementation:

- First, a structure `stParams` of type `ST_FTR_IIRSpec` is declared and initialized. The structure is used for parameterizing the function block `FB_FTR_IIRSpec` or for configuring the filter.

```
stParams : ST_FTR_IIRSpec := ( … )
```

- An instance `fbFilter` of the function block `FB_FTR_IIRSpec` is then created. The structure `stParams` is transferred during instantiation.

```
fbFilter : FB_FTR_IIRSpec := (stConfig := stParams);
```

- In the source code, the `Call()` method is called to execute the filter, and error checking variants are listed.

```
IF NOT fbFilter.bError THEN
    fbFilter.Call(ADR(aSignalBuffer), SIZEOF(aSignalBuffer),
                  ADR(aFilteredSignal), SIZEOF(aFilteredSignal));
ELSE
    // if on error state, do something
    nErrorCount := nErrorCount + 1;
    // e.g. check if filter is configured
    // IF fbFilter.bConfigured THEN
    // or use ipResultMessage to check root cause
    // fbFilter.ipResultMessage
END_IF;
```

- The `Configure()` and/or `Reset()` methods are also called to change the filter configuration during runtime (`fbFilter.Configure`) or to perform a reset (`fbFilter.Reset`). The call of the methods is controlled by the flags `bReconfigure` and `bReset`. They can be manually set to `TRUE` or `FALSE` in the Online Watch.

Observation:

Recording with TwinCAT 3 Scope shows that the amplitude of the output signals is attenuated and phase-shifted by the filter. The signal in the second channel (green) is attenuated more strongly than that in the first channel (blue, attenuation by -3 dB).

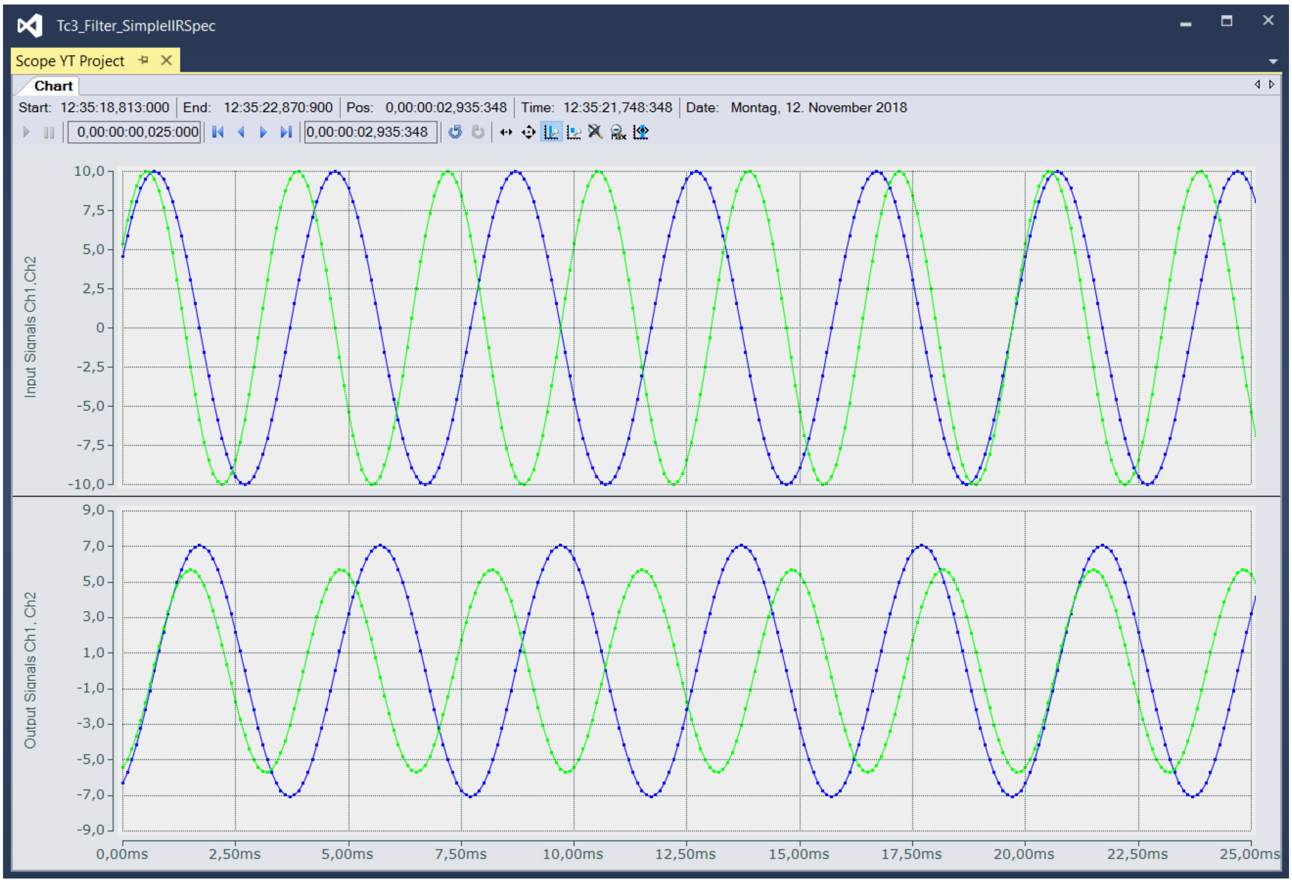


Fig. 10: Signal curves of the input signals (top) and the output signals (bottom) (blue: channel 1, green: channel 2)

If the flag `bReconfigure` is set, the cut-off frequency is shifted to 50 Hz, which makes the amplitude of both output signals even smaller.

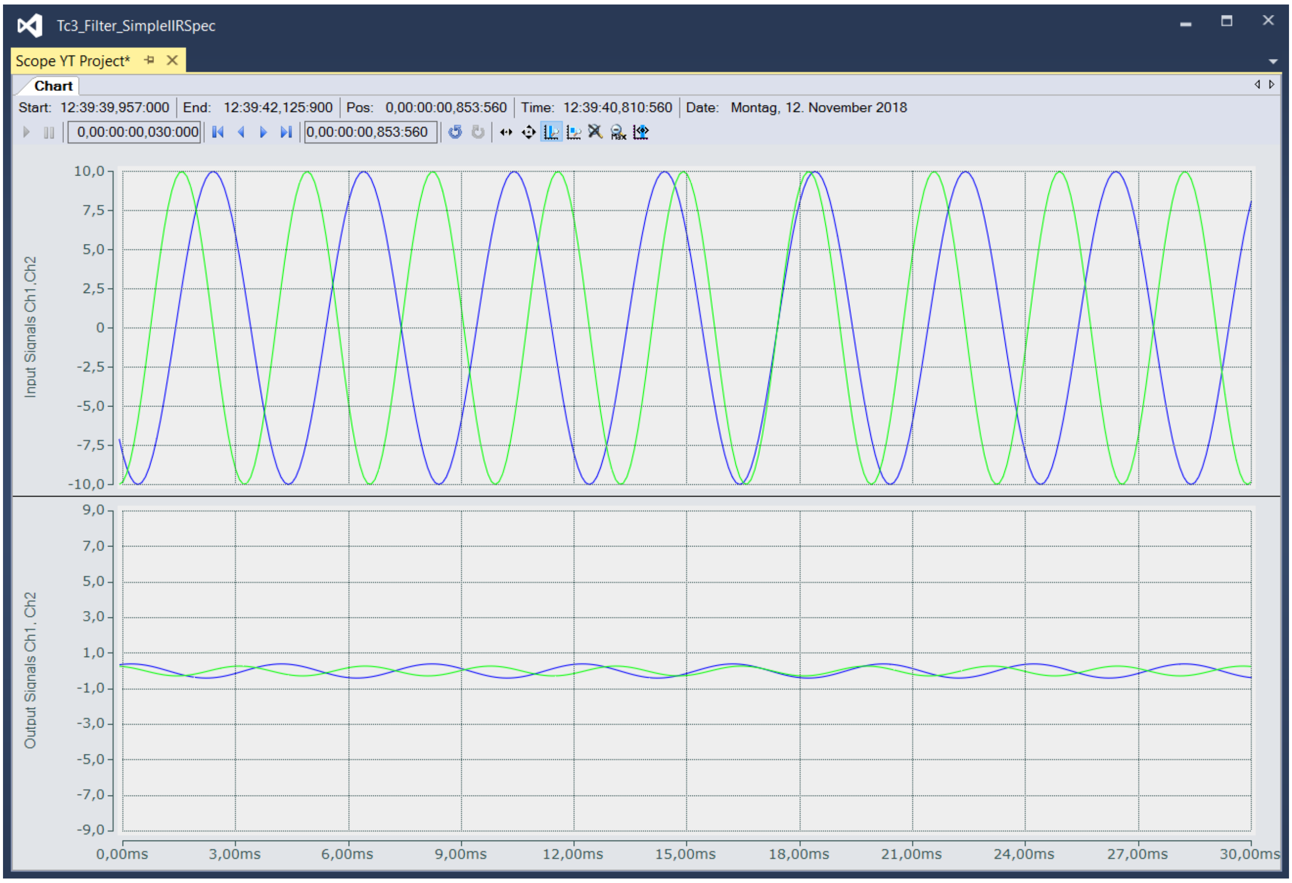


Fig. 11: Signal curves of the input signals (top) and the output signals (bottom), if the flag bReconfigure is set (blue: channel 1, green: channel 2)

If the flag `bReset` is set, the internal state of the filter is deleted, i.e. past input and output values are no longer taken into account. The continuous signal curve is interrupted (at approx. 27 ms), and the filter settles again.

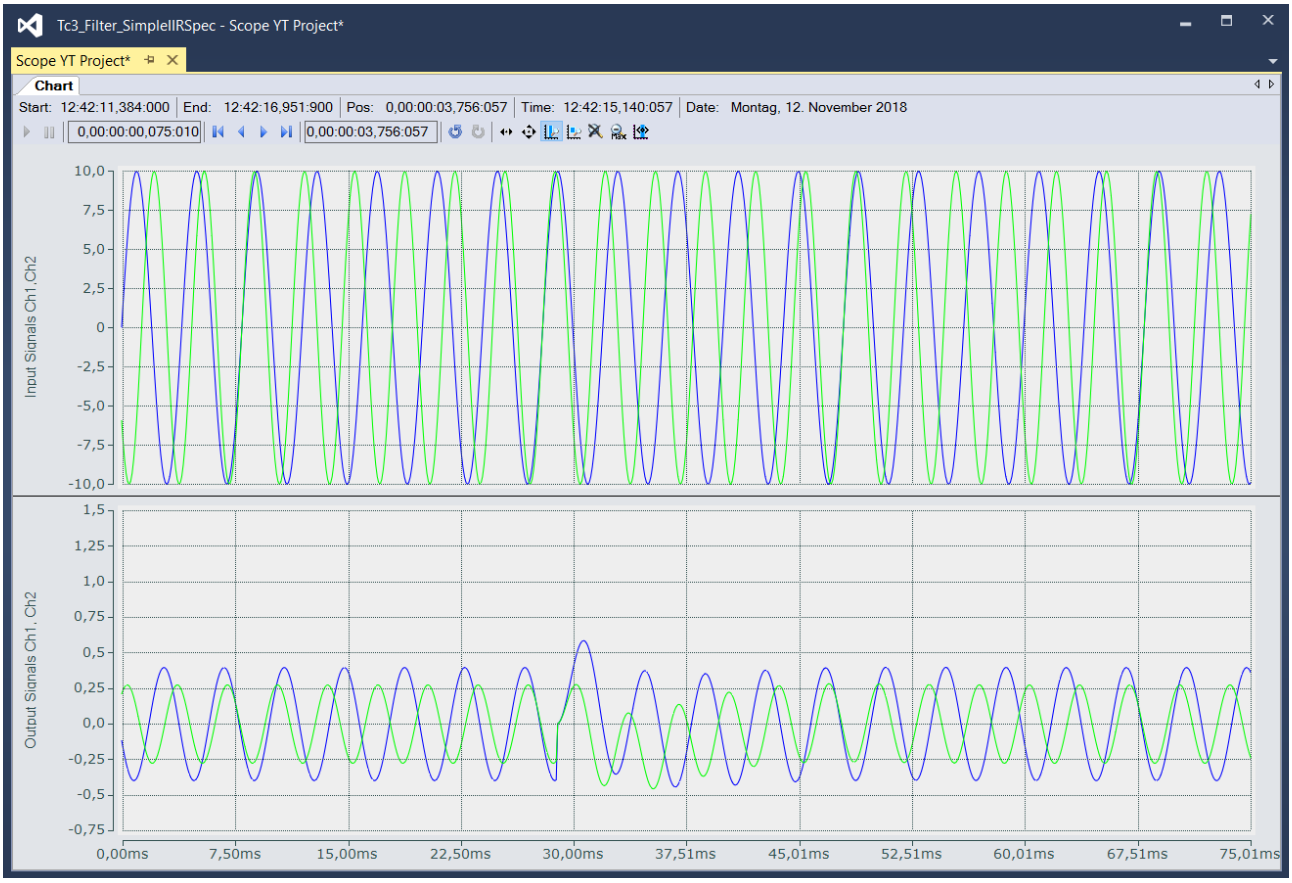


Fig. 12: Signal curves of the input signals (top) and the output signals (bottom) when the flag bReset is set (blue: channel 1, green: channel 2)

See also:

- Filter types and parameterization [▸ 17]
- FB_FTR_IIRSpec [▸ 35]

6.2 Declaring and calling filters with FB_FTR_<type>

This sample shows how the individual filter function blocks of the PLC library Tc3_Filter are declared and called.

Download: https://infosys.beckhoff.com/content/1033/TF3680_TC3_Filter/Resources/5906974603/.zip (*.tnzip)

Description:

- The sample project consists of a TwinCAT PLC project.
- In the MAIN PLC program, a function generator generates a harmonic signal that is used as input signal for the various filters.
- Each filter is described in a separate function block, which is instantiated in the MAIN PLC program.
- The MAIN PLC program is called by a task with a cycle time of 1 ms.

Implementation:

- The declarations and calls of the filter function blocks provided in the Tc3_Filter library are shown as examples in the function blocks.
- The variable `nExampleSelector` can be used to change the filter during runtime.

Basic principle:

```
FUNCTION_BLOCK FB_PT1
VAR_INPUT
    aBuffer    : ARRAY [1..MAIN.cChannels] OF ARRAY [1..MAIN.cOversamples] OF LREAL;
END_VAR
VAR_OUTPUT
    aOutput    : ARRAY [1..MAIN.cChannels] OF ARRAY [1..MAIN.cOversamples] OF LREAL;
END_VAR
VAR
    stParams: ST_FTR_PT1 := (fSamplingRate := 10000,
    fKp := 1,
    fT1 := 6.3661977236758134307553505349006E-4,
    nOversamples := MAIN.cOversamples,
    nChannels := MAIN.cChannels);
// fT1 correlates to fc=250 Hz
    fbFilter : FB_FTR_PT1:=(stConfig:=stParams);
END_VAR
```

See also:

- FB_FTR_IIRCoeff [▸ 30]
- FB_FTR_IIRSpec [▸ 35]
- FB_FTR_MovAvg [▸ 38]
- FB_FTR_PT1 [▸ 41]
- FB_FTR_PT2 [▸ 44]
- FB_FTR_PT3 [▸ 47]
- FB_FTR_PTn [▸ 50]
- FB_FTR_Notch [▸ 53]
- FB_FTR_LeadLag [▸ 55]
- FB_FTR_PT2oscillation [▸ 58]
- FB_FTR_PTt [▸ 61]
- FB_FTR_Median [▸ 64]
- FB_FTR_ActualValue [▸ 67]
- FB_FTR_Gaussian [▸ 71]

6.3 Reconfiguration with initial values

This sample shows how a filter can be reconfigured or dynamically adapted during runtime. It also describes how previous knowledge of a signal can be used to shorten the settling time of a filter after reconfiguration.

Download: https://infosys.beckhoff.com/content/1033/TF3680_TC3_Filter/Resources/5906977931/.zip (*.tnzip)

Description:

- The sample project consists of a TwinCAT PLC project and a measurement project.
- Two output signals and a counter variable are configured in the measurement project.
- The input signal is generated synthetically via a function generator that is called in the PLC. The signal is processed by two Butterworth low-pass filters that are configured in different ways.
- Each filter is described in a separate function block, which is instantiated in the MAIN PLC program.
- The MAIN PLC program is called by a task with a cycle time of 1 ms.

Observation:

The recording of the TwinCAT 3 Scope shows a signal curve that is very noisy and jumps between two levels.

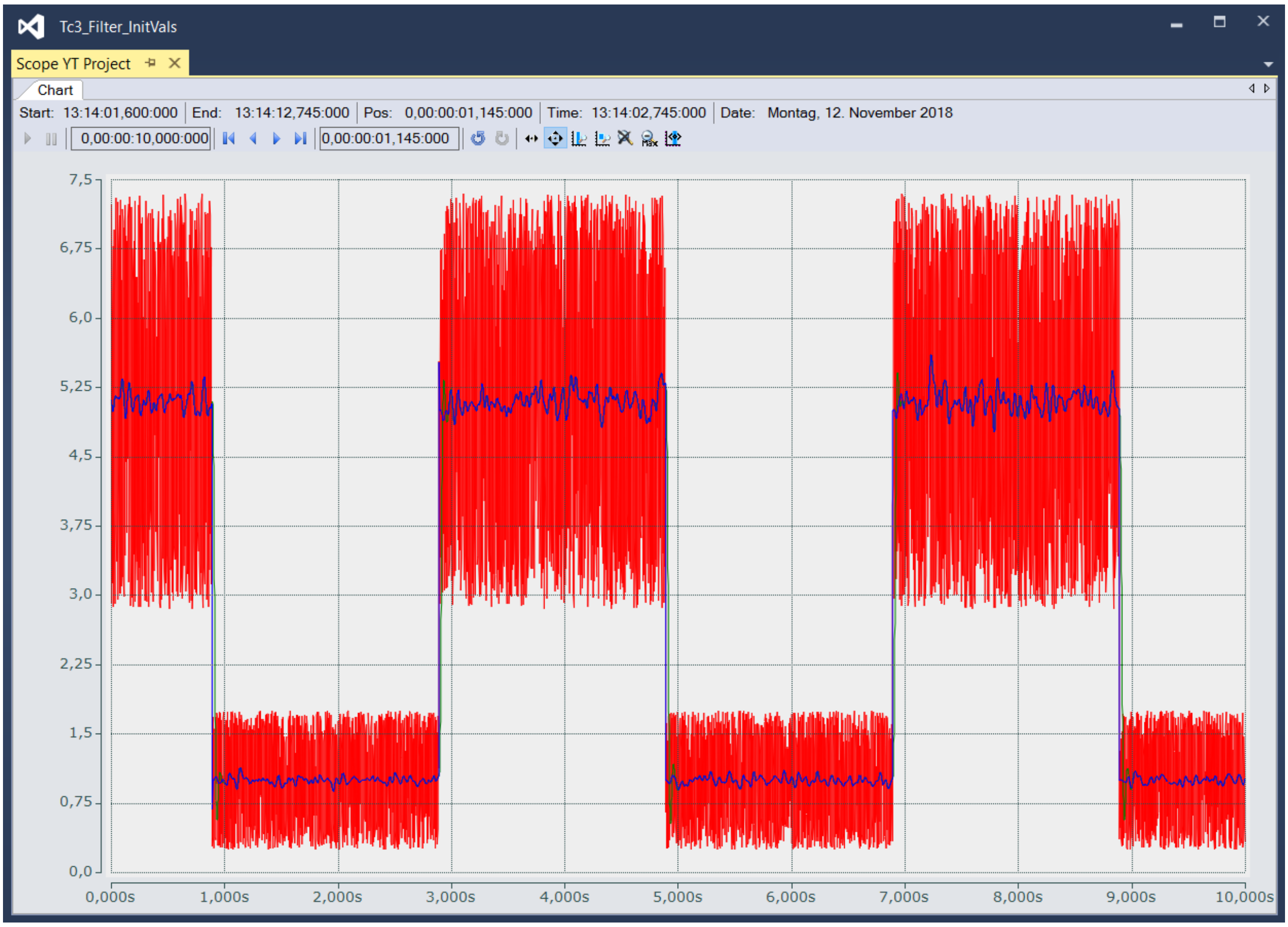


Fig. 13: Signal curve of the input signal (red) and signal curves of the output signals (green FB_ContinuousFilter, blue: FB_DynamicFiltering)

The first filter described in the function block `FB_ContinousFilter` remains unchanged during runtime. The output signal of this filter is limited in its dynamics due to the low-pass effect.

The second filter, which is described in the function block `FB_DynamicFiltering`, responds dynamically to the variations of the input signal during runtime.

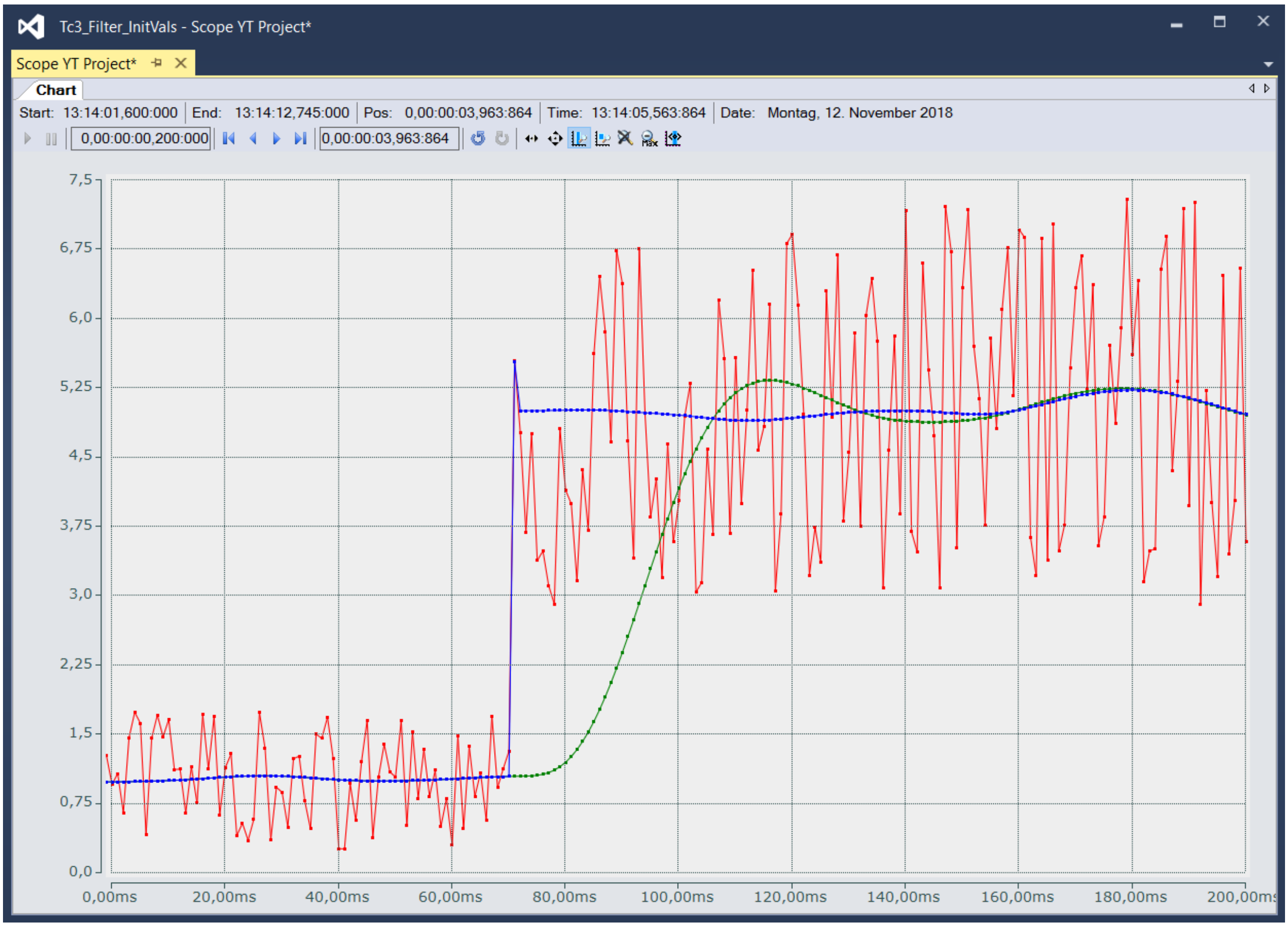


Fig. 14: Signal curve of the input signal (red) and signal curves of the output signals (green FB_ContinuousFilter, blue: FB_DynamicFiltering) (zoom into an edge)

The `bTrigger` input informs the `fbDynamicLowPass` instance of the `FB_DynamicFiltering` function block that there is a jump in the signal. In practice, this can be indicated by the signal itself or by an action initiated by the PLC.

If `bTrigger` is set to `TRUE`, the `Configure()` method is called, which sets the low-pass cut-off frequency to the maximum permitted value. This means that the low-pass effect is lost the next time the `Call()` method is called. However, the output signal can quickly follow the jump in the input signal.

Finally, the filter effect is reactivated by calling the `Configure()` method and greatly reducing the cut-off frequency.

In order to reduce the settling time of the filter considerably, previous knowledge about the two plateaus is used. To this end, the parameter `pInitialValues` of the configuration structure `stParams` (type `ST_FTR_IIRSpec`) is assigned the initial value 1 or 5. The values are defined in the MAIN program (`fPreKnowledgeHigh`, `fPreKnowledgeLow`) and reflect the user's knowledge that the signal to be evaluated will jump approximately between 1 and 5. The filter is thus already in the steady state around this value when the `Call()` method is called for the first time.

```
stParams.fCutoff := Main.fCutOff; // reduce cutoff freq.

IF bRisingEdge THEN
    // on rising edge apply preknowledge (Signal around 5)
    bRisingEdge := FALSE;
    stParams.pInitialValues := ADR(fInitValHighLevel);
    stParams.nInitialValuesSize := SIZEOF(fInitValHighLevel);
ELSE
    // on falling edge
    stParams.pInitialValues := ADR(fInitValBaseLevel);
    stParams.nInitialValuesSize := SIZEOF(fInitValBaseLevel);
END_IF;

fbFilter.Configure(stConfig := stParams);
```

The time it takes to settle to the new plateau can be reduced significantly by event-based switching of the filter characteristics.

See also:

- FB_FTR_IIRSpec [▶ 35]

6.4 Reconfiguration with and without reset

This sample shows how a filter can be reconfigured during runtime and which options exist to suppress a reset.

Download: https://infosys.beckhoff.com/content/1033/TF3680_TC3_Filter/Resources/5907283339/.zip (*.tnzip)

Description:

- The sample project consists of a TwinCAT PLC project and a measurement project
- Two output signals and a counter variable are configured in the measurement project.
- A signal generator generates a harmonic signal with a frequency of 700 Hz. The signal is processed by two filters with identical filter coefficients.
- During runtime, the system alternates between two filter configurations (parameterization of the filter coefficients). For one filter instance (`fbFilterReset)`, in contrast to the other filter instance (`fbFilterNoReset`), the `Reset()` method is called when the filter is reconfigured to reset the internal status of the filter.
- The MAIN PLC program is called by a task with a cycle time of 1 ms.

Implementation:

- The parameter `bReset` of the configuration structure ST_FTR_IIRCoeff can be used to control whether a reset is executed when the `Configure()` method is executed (default value is `TRUE`).

```
stParams : ST_FTR_IIRCoeff := ( … )
```

```
stParams.bReset := FALSE;
fbFilterNoReset.Configure(stConfig := stParams);
```

Observation:

If you activate the project, you can observe the output signals of the two filter instances in the measurement project. The red signal shows the output signal of the function block instance `fbFilterNoReset`. The time curve of the signal shows no jump through the reconfiguration. The output signal of the function block instance `fbFilterReset`, on the other hand, jumps when the reconfiguration is executed, since the filter returns to zero at the time of the reconfiguration (blue signal).

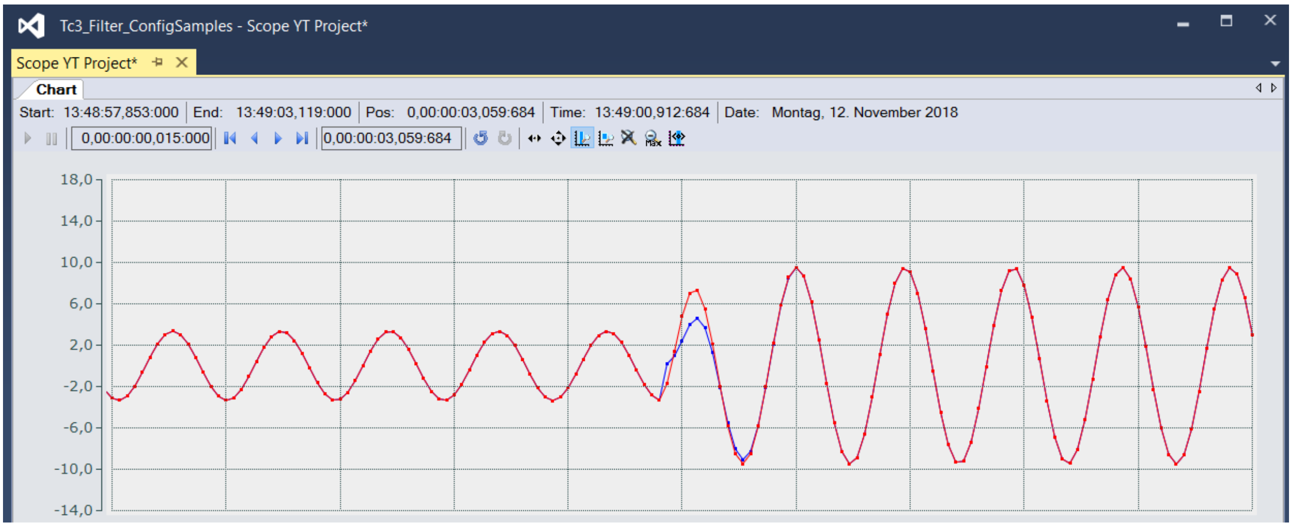


Fig. 15: Signal curves of the output signals during reconfiguration without reset (red) and with reset (blue)

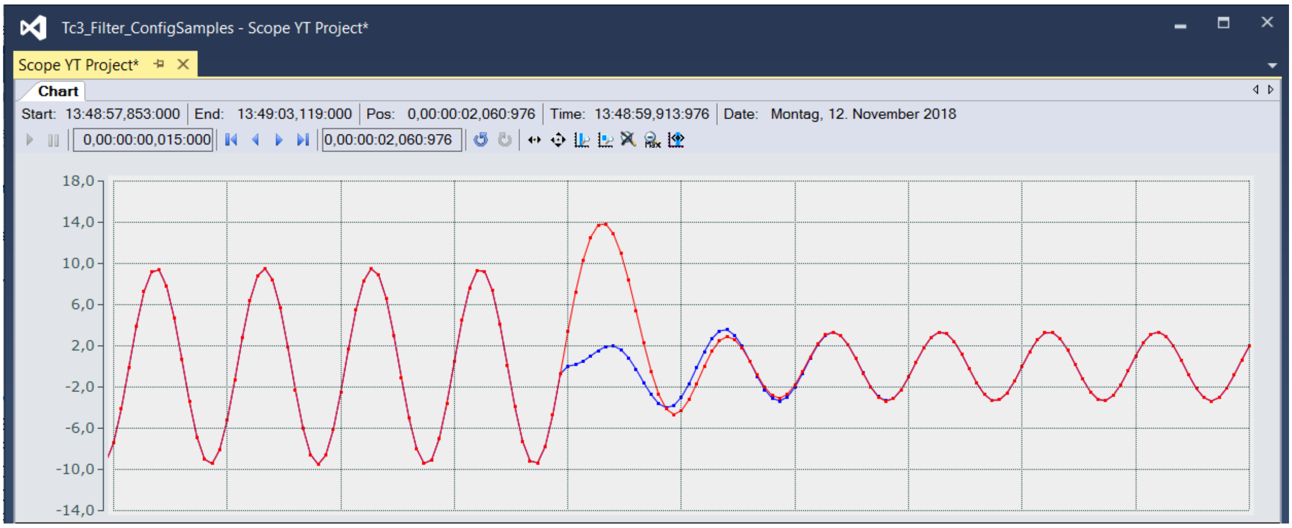


Fig. 16: Signal curves of the output signals during reconfiguration without reset (red) and with reset (blue)

See also:

- FB_FTR_IIRCoeff [▶ 30]

7 Appendix

7.1 Return codes

Return codes of the ipResultMessage.

Online Watch:

fbFilter	FB_FTR_IIRSpec	
bError	BOOL	TRUE
bConfigured	BOOL	FALSE
ipResultMessage	I_TcMessage	
eSeverity	TCEVENTSEVERITY	Error
ipSourceInfo	I_TcSourceInfo	
nEventId	UDINT	8197
sEventClassName	STRING(255)	'TcFilter'
sEventText	STRING(255)	'Error during configuration. fCutoff must be greater than zero and smaller than fSamplingrate/2.'

Defined events:

nEventId (hex)	sEventText
16#1001	Error during initialization. No router memory available. Check size of router memory.
16#1002	Error during access to object archive.
16#1004	Error in Transition PREOP->SAFEOP.
16#1005	Error in Transition SAFEOP->OP
16#1006	Error in Transition SAFEOP->OP: No Task assigned. Module will not be executed cyclically.
16#1007	Error in Transition OP->SAFEOP
16#1008	Error in Transition SAFEOP->PREOP
16#2001	Error during configuration. A nullpointer has been allocated.
16#2002	Error during configuration. A nullpointer has been allocated.
16#2003	Error during configuration. fT1 must be greater than zero.
16#2004	Error during configuration. fSamplingrate must be greater than zero.
16#2005	Error during configuration. fCutoff must be greater than zero and smaller than fSamplingrate/2.
16#2006	Error during configuration. fBandwidth must be greater than zero and smaller than fSamplingrate/2 - fCutoff.
16#2007	Error during configuration. fPassBandRipple must be greater than zero.
16#2008	Error during configuration. fStopBandRipple must be greater than zero.
16#2009	Error during configuration. nChannels must be greater than zero and smaller than 101.
16#200A	Error during configuration. nOversamples must be greater than zero.
16#200B	Error during configuration. nFilterOrder has to be between 1 and 10/20 (band pass and band stop/ low pass and high pass)
16#200C	Error during configuration. nCoefficientArraySize_A and nCoefficientArraySize_B must be equal and greater than seven.
16#200D	Error during configuration. pCoefficientArrayAdr_A is an invalid pointer.
16#200E	Error during configuration. A0 coefficient cant be zero due zero devision.
16#200F	Error during configuration. pCoefficientArrayAdr_B is an invalid pointer.
16#2010	Error during configuration. fKp must be greater than zero.
16#2011	Error during configuration. nSamplesToFilter must be greater than zero.
16#2012	Error during configuration. fT2 must be greater than zero.
16#2013	Error during configuration. fT3 must be greater than zero.
16#2014	Error during configuration.nOrder must be greater than zero and smaller than eleven.

nEventId (hex)	sEventText
16#2015	Error during configuration. nInitialValuesSize must be 0, 8, (OrderB+OrderA)*8 or (OrderB+OrderA)*nChannels*8.
16#2016	Error during configuration. Invalid FilterName.
16#2017	Error during configuration. Invalid FilterType.
16#2018	Error during configuration. bReset = false is only allowed if the following filter structure member don't change: nCoefficientArraySize_A, nCoefficientArraySize_B, nChannels and nOversamples.
16#2019	Error during configuration. nCoefficientArraySize_Sos must be a multiple of 48.
16#201A	Error during configuration. pCoefficientArrayAdr_Sos is an invalid pointer.
16#201B	Error during configuration. Filter parameter lead to an unstable filter, please choose other filter parameter.
16#201C	Error during configuration. bReset = false is only allowed if the following filter structure member don't change: nCoefficientArraySize_Sos, nChannels and nOversamples.
16#201D	Error during configuration. nInitialValuesSize must be 0, 8, (number of biquads)*4*8 or (number of biquads)*4*nChannels*8.
16#201E	Error during configuration. fNotchfrequency must be greater than zero and smaller than fSamplingrate/2.
16#201F	Error during configuration. fQ must be greater than zero.
16#2020	Error during configuration. fTheta must be greater than zero.
16#2021	Error during configuration. fTt must be greater than zero and a multiple of 1/ fSamplingRate.
16#2022	Error during configuration. fDeltaMax must be equal or greater than zero.
16#3001	Error during runtime. No router memory available. Check size of router memory.
16#3002	Error during runtime. Missing configuration.
16#3003	Error during runtime. Cyclic caller is assigned. Methods can not be called.
16#3004	Error during runtime. Size of fIn Array doesnt match with nOversamples*nChannels.
16#3005	Error during runtime. Size of fOut Array cant be smaller than fIn Array Size.
16#3006	Error during runtime. A nullpointer has been allocated @ pIn.
16#3007	Error during runtime. A nullpointer has been allocated @ pOut.

7.2 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Download finder

Our download finder contains all the files that we offer you for downloading. You will find application reports, technical documentation, technical drawings, configuration files and much more.

The downloads are available in various formats.

Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on our internet page: www.beckhoff.com

You will also find further documentation for Beckhoff components there.

Beckhoff Support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline:	+49 5246 963-157
e-mail:	support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline:	+49 5246 963-460
e-mail:	service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone:	+49 5246 963-0
e-mail:	info@beckhoff.com
web:	www.beckhoff.com

More Information:
www.beckhoff.com/TF3680

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com