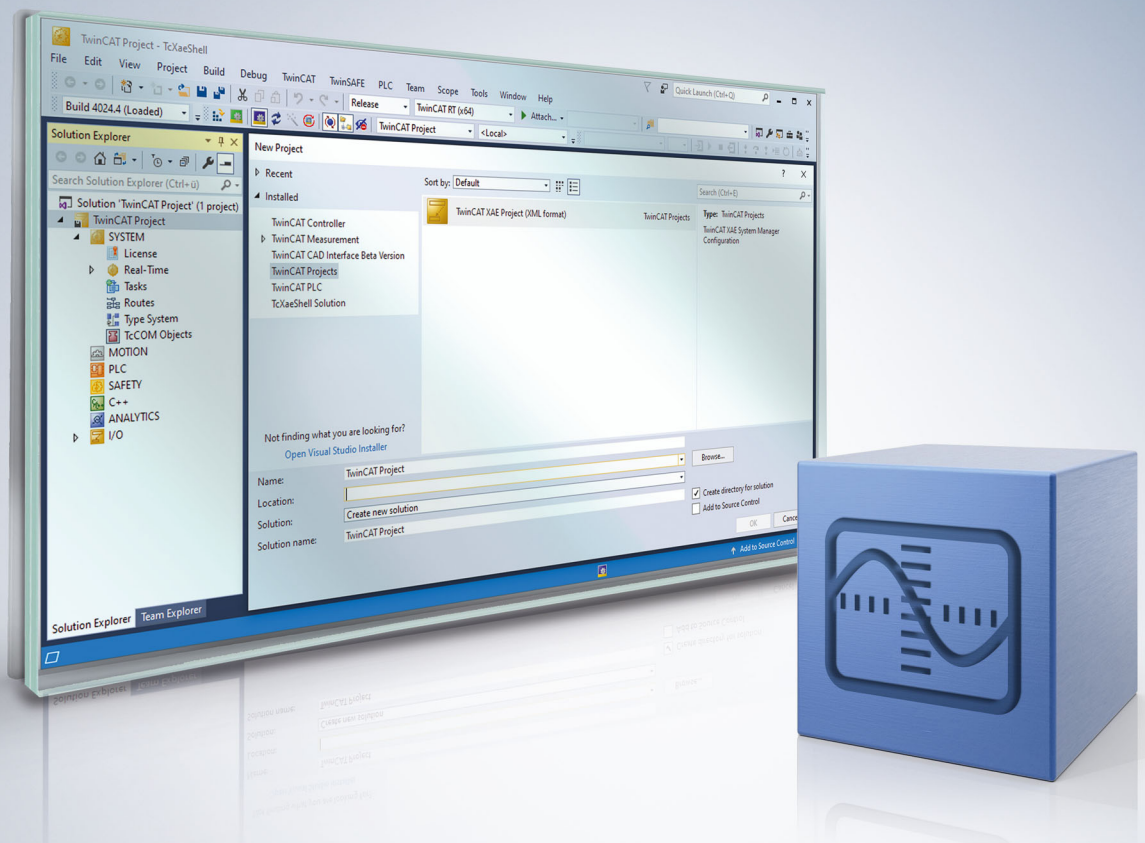
BECKHOFF New Automation Technology

Handbuch | DE

TF3600

TwinCAT 3 | Condition Monitoring



09.12.2024 | Version: 1.11.0

Inhaltsverzeichnis

1 Vorwort ... 7
1.1 Hinweise zur Dokumentation ... 7
1.2 Zu Ihrer Sicherheit ... 8
1.3 Hinweise zur Informationssicherheit ... 9
2 Übersicht ... 10
2.1 Einführende Konzepte ... 10
2.1.1 Fourier-Analyse ... 11
2.1.2 Analyse von Daten-Streams ... 18
2.1.3 Getriggerte Analyse eines Zeitabschnitts ... 24
2.1.4 Skalierung von Spektren ... 26
2.1.5 Statistische Auswertung ... 29
2.2 Anwendungskonzepte ... 34
2.2.1 Schwingungsbeurteilung ... 35
2.2.2 Frequenzanalyse ... 37
2.2.3 Wälzlagerüberwachung ... 41
2.2.4 Getriebeüberwachung ... 48
2.2.5 Lebensdaueranalyse und Schädigungsrechnung ... 53
2.3 Literaturhinweise ... 64
3 Installation ... 66
3.1 Systemvoraussetzungen ... 66
3.2 Kompatibilität ... 66
3.3 Installation ... 67
3.4 Installation TwinCAT 3.1. Build 4026 ... 70
3.5 Lizenzierung ... 70
4 Technische Einführung ... 73
4.1 Speicherverwaltung ... 73
4.2 Task Einstellung ... 74
4.3 NaN-Werte ... 76
4.4 Parallelverarbeitung mit Transfer Tray ... 77
4.5 Umgang mit MultiArray ... 79
5 SPS-API ... 84
5.1 Funktionsbausteine ... 88
5.1.1 FB_CMA_AnalyticSignal ... 89
5.1.2 FB_CMA_ArgSort ... 93
5.1.3 FB_CMA_BufferConverting ... 96
5.1.4 FB_CMA_CrestFactor ... 99
5.1.5 FB_CMA_CrestFactorPlus ... 104
5.1.6 FB_CMA_ComplexDataHandling ... 108
5.1.7 FB_CMA_ComplexFFT ... 112
5.1.8 FB_CMA_Correlation ... 115
5.1.9 FB_CMA_DiscreteClassification ... 121
5.1.10 FB_CMA_Downsampling ... 125
5.1.11 FB_CMA_EmpiricalExcess ... 126

5.1.12 FB_CMA_EmpiricalMean ... 131
5.1.13 FB_CMA_EmpiricalSkew ... 136
5.1.14 FB_CMA_EmpiricalStandardDeviation ... 141
5.1.15 FB_CMA_Envelope ... 146
5.1.16 FB_CMA_EnvelopeSpectrum ... 150
5.1.17 FB_CMA_HistArray ... 154
5.1.18 FB_CMA_InstantaneousFrequency ... 160
5.1.19 FB_CMA_InstantaneousPhase ... 164
5.1.20 FB_CMA_IntegratedRMS ... 167
5.1.21 FB_CMA_MagnitudeSpectrum ... 172
5.1.22 FB_CMA_MeanStressCorrection ... 176
5.1.23 FB_CMA_MinersRule ... 179
5.1.24 FB_CMA_MomentCoefficients ... 183
5.1.25 FB_CMA_MultiBandRMS ... 189
5.1.26 FB_CMA_OrderPowerSpectrum ... 194
5.1.27 FB_CMA_PowerCepstrum ... 198
5.1.28 FB_CMA_PowerSpectrum ... 202
5.1.29 FB_CMA_Quantiles ... 206
5.1.30 FB_CMA_RainflowCounting ... 211
5.1.31 FB_CMA_ReadCsvFile ... 215
5.1.32 FB_CMA_RealFFT ... 216
5.1.33 FB_CMA_RMS ... 220
5.1.34 FB_CMA_SlidingDFT ... 224
5.1.35 FB_CMA_SparseSpectrum ... 229
5.1.36 FB_CMA_SpikeEnergySpectrum ... 233
5.1.37 FB_CMA_Sink ... 238
5.1.38 FB_CMA_Source ... 245
5.1.39 FB_CMA_SourcePaired ... 252
5.1.40 FB_CMA_VibrationAssessment ... 258
5.1.41 FB_CMA_WatchUpperThresholds ... 262
5.1.42 FB_CMA_ZoomFFT ... 266
5.1.43 FB_CMA_MovingMean ... 270
5.1.44 FB_CMA_MovingStandardDeviation ... 274
5.1.45 FB_CMA_MovingSkew ... 278
5.1.46 FB_CMA_MovingExcess ... 283
5.2 Funktionen ... 287
5.2.1 F_MA_IsNAN ... 287
5.2.2 F_CM_CalculateRecommendedOverlap ... 287
5.2.3 F_CM_CalculateWoehlerCurve ... 288
5.2.4 F_CM_CalculateWindow ... 288
5.2.5 F_CM_ApplySpectralScaling ... 289
5.3 Datentypen ... 289
5.3.1 E_CM_ComplexDataHandling ... 289
5.3.2 E_CM_CorrelationMode ... 289
5.3.3 E_CM_MCoefOrder ... 290
5.3.4 E_CM_MeanStressCorrection ... 290

5.3.5 E_CM_ScalingType ... 290
5.3.6 E_CM_SlidingDFTType ... 290
5.3.7 E_CM_SpectrumType ... 291
5.3.8 E_CM_UnwrapMethod ... 291
5.3.9 E_CM_WindowMode ... 291
5.3.10 E_CM_WindowType ... 291
5.3.11 E_MA_ElementTypeCode ... 292
5.3.12 T_CM_WindowParameters ... 292
5.3.13 Fehlercodes ... 292
5.3.14 InitPars Strukturen ... 297
5.4 Globale Konstanten ... 322
5.4.1 GVL_CM ... 322
5.4.2 GVL_CM_Base ... 323
5.4.3 Global_Version ... 323
5.4.4 Param ... 324

6 Beispiele ... 325
6.1 FFT mit reell-wertigem Eingangssignal ... 325
6.2 FFT mit komplex-wertigem Eingangssignal ... 327
6.3 Magnitudenspektrum ... 329
6.4 Mehrkanal-Magnitudenspektrum ... 332
6.5 Berechnung einzelner Spektralwerte ... 333
6.6 Fensterfunktionen ... 336
6.7 Skalierung von Spektren ... 338
6.8 Zeitbasierter RMS ... 339
6.9 Multiband-RMS ... 341
6.10 Histogramm ... 343
6.11 Statistische Methoden ... 345
6.12 Schwingungsbeurteilung nach ISO 10816-3 ... 346
6.13 Schwingungsbeurteilung nach ISO 10816-3 (kompakt) ... 349
6.14 Schwingungsbeurteilung nach ISO 10816-3 (erweitert) ... 351
6.15 Condition Monitoring mit Frequenzanalyse ... 353
6.16 Schwellwertbetrachtung gemittelte Betragsspektren ... 359
6.17 Crest Faktor ... 360
6.18 Einhüllendenspektrum ... 363
6.19 Leistungscepstrum ... 365
6.20 Eventbasierte Frequenzanalyse ... 366
6.21 Ordnungsanalyse ... 368
6.22 Schädigungsüberwachung ... 370
6.23 Spitzenwert Spektrum ... 373
6.24 Zoom FFT ... 376
6.25 Sliding DFT ... 378
6.26 Korrelationsfunktion ... 381
6.27 Gleitender Mittelwert ... 383

7 Anhang ... 386
7.1 Fehlercodes Übersicht ... 386

7.2 ADS Return Codes ... 387
7.3 Optionen der Spektrumsskalierung ... 390
7.4 Support und Service ... 392

Glossar ... 394

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, stets die aktuell gültige Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiterentwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

© Beckhoff Automation GmbH & Co. KG, Deutschland.
Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet.
Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten.

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

⚠ GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

⚠ WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

⚠ VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt

Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

2 Übersicht

Beckhoff bietet eine Toolbox aus Hardware- und Software-Komponenten an, um ein in die Steuerung integriertes Condition Monitoring System zu realisieren. Der Vorteil der Beckhoff-Lösung ist dabei die Integration in die Standard Maschinensteuerung. So werden zusätzliche Subsysteme mit aufwendiger Querkommunikation vermieden. Maschinensteuerung und Condition Monitoring laufen auf derselben Plattform, können mit den gleichen Engineering-Tools programmiert werden und haben mit EtherCAT ein gemeinsames Feldbussystem.

Die TwinCAT Condition Monitoring Bibliothek bildet einen erheblichen Teil der Software-Toolbox. Es stehen darin verschiedene mathematische Algorithmen als SPS-Funktionsbausteine zur Verfügung.

Produktinformationen

Die aktuelle Version der TwinCAT 3 Condition Monitoring Bibliothek ist als Download unter http://www.beckhoff.com/software verfügbar. Die SPS-Bibliothek bietet verschiedene Algorithmen für die Datenanalyse. Multitask Applikationen werden empfohlen, um Datenaufzeichnung und Verarbeitung zu trennen. Die Kommunikation zwischen unterschiedlichen Tasks und CPU-Kernen erfolgt durch die Mechanismen der Condition Monitoring Bibliothek automatisch.

Produktkomponenten

Das Produkt **TF360x Condition Monitoring** besteht aus den folgenden Komponenten:

- **SPS Bibliotheken:** Tc3_CM.compiled-library, Tc3_CM_Base.compiled-library and Tc3_MultiArray.compiled-library
- **Treiber:** TcCM.tmx und TcMultiArray.tmx

Zusätzlich beinhaltet das Produkt die Komponenten von **TF3680 TC3 Filter**. In der Lizenz TF3600 Condition Monitoring Level 1 ist die Lizenz TF3680 enthalten, sodass alle Bestandteile des Produkts TF3680 TwinCAT 3 Filter verwendet werden können.

2.1 Einführende Konzepte

Beim ersten Kontakt mit dem Thema Condition Monitoring und Signalverarbeitung wird dringend die Einbeziehung von Literaturquellen zusätzlich zu dieser Dokumentation empfohlen. Literaturhinweise finden Sie am Ende dieses Abschnitts.

Im Folgenden werden einführende Konzepte zur Signalverarbeitung, im Speziellen Fourier-Analyse, sowie Statistik vorgestellt. Es werden keine Details zur Programmierung erläutert, sondern lediglich Bezüge zu den Schnittstellen und Funktionsweisen der in der Condition Monitoring Bibliothek verwendeten Algorithmen aufgeführt.

Sie erfahren hier:

- Wie funktioniert eine Frequenzanalyse?
- Wie funktioniert eine lückenlose Analyse eines kontinuierlichen Daten-Streams?
- Wie analysiere ich ein Zeitsegment, bzw. wie triggere ich eine Analyse?
- Wie skaliere ich ein Spektrum und warum ist das wichtig?
- Wie erhalte ich statistisch belastbare Ergebnisse bei verrauschten oder gestörten Messsignalen?

2.1.1 Fourier-Analyse

Einführung

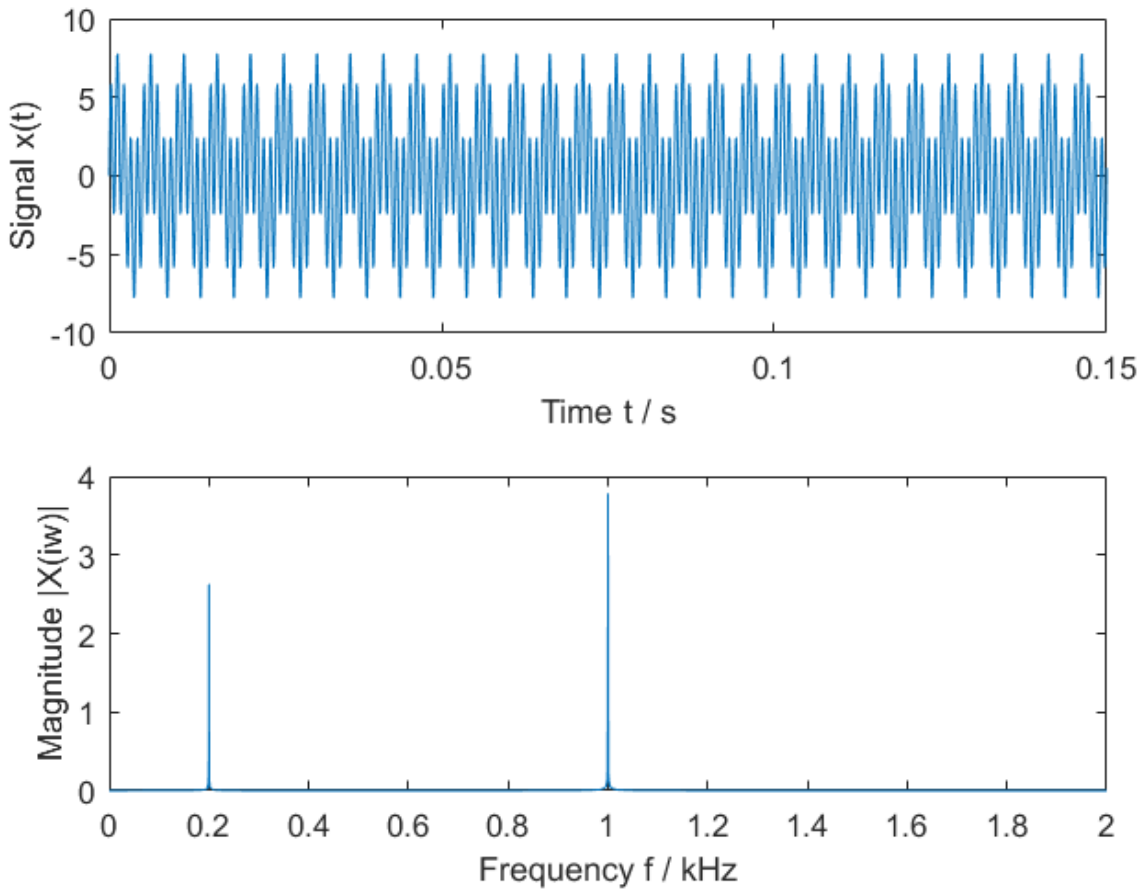
Die wohl wichtigste Methode der Frequenzanalyse ist die Fourier-Analyse. Die grundlegende Bedeutung der Fourier-Analyse ergibt sich daraus, dass sie ein Signal *x(t)* in eine Darstellung zerlegt, die einer Überlagerung von Sinus- und Cosinusschwingungen entspricht. Das Ergebnis dieser Transformation wird als Signalspektrum oder kurz Spektrum bezeichnet. Definition der Fourier-Transformation:

$$X(w) = \int_{-\infty}^{\infty} x(t) e^{-i\omega t}\, \mathrm{d}t = \int_{-\infty}^{\infty} x(t) \left[\cos(\omega t) - i \sin(\omega t)\right]\, \mathrm{d}t$$

Das Signalspektrum ist vom Informationsgehalt gleichwertig mit dem ursprünglichen Signal und darüber hinaus wird es dem Entstehungsmechanismus beispielsweise von zu analysierenden Schwingungen gerecht. Wenn von zwei verschiedenen Bauteilen Schwingungen mit unterschiedlicher Periode (Frequenz) ausgehen, die sich additiv überlagern, sind in der Fourier-Transformierten zwei verschiedene Komponenten sichtbar. Die Kombination von Sinus und Cosinus für jede Frequenz ermöglicht es außerdem beliebige Phasenlagen abzubilden.

Werden beispielsweise zwei Sinusschwingungen mit unterschiedlicher Frequenz und Amplitude überlagert, ergeben sich die in der unteren Darstellung aufbereiteten Grafiken. Aus dem Zeitverlauf lässt sich kaum sagen, wie sich das Gesamtsignal zusammensetzt. Aus dem Betragsspektrum $|X(w)|$, dem Betrag der Fourier-Transformierten, mit entsprechender Skalierung (siehe Skalierung von Spektren [▶ 26]) lässt sich hingegen leicht erkennen, dass es sich um zwei Schwingungen handelt – eine mit der Frequenz 0,2 kHz und Amplitude 2,6 und eine mit der Frequenz 1 kHz und Amplitude 3,8. Die Phaseninformation ist aufgrund der Betragsbildung ausgeblendet.

In dem Beispiel Magnitudenspektrum [▶ 329] wird für ein Signal dieser Form exemplarisch das Magnitudenspektrum berechnet und dargestellt.



Es gibt zwei Prozesse, welche die in einer Maschine entstehenden Schwingungssignale bei der Schallübertragung beeinflussen. Erstens eine Übertragung über Maschinenteile, die Schwingungen je nach Frequenz unterschiedlich stark dämpfen und zweitens die Überlagerung mit Schwingungen von anderen Maschinenkomponenten, wobei sich die Amplituden ohne gegenseitige Beeinflussung addieren. Aufgrund der Eigenschaften der Fourier-Transformation werden beide Faktoren voneinander getrennt:

- Zeitverzögerungen wirken sich ausschließlich auf die Phase der Fourier-Transformierten aus
- Frequenzselektive Dämpfung sowie konstruktive Überlagerung der Schwingungsamplituden werden im Betrag der Fourier-transformierten sichtbar.

Verarbeitung zeitdiskreter Signale und die Diskrete Fourier-Transformation

Ein sehr wichtiger Aspekt bei der Anwendung der Fourier-Analyse ist die zeitliche Abtastung des Signals. Die *Fourier-Tansformierte* ist mathematisch definiert für kontinuierliche, zeitlich unbeschränkte Signale.

Praktisch eingesetzt wird jedoch die *Diskrete Fourier-Transformation* (DFT). Diese ist definiert für ein *diskretes, periodisches Signal* mit einer *endlichen Zahl von diskreten Frequenzkomponenten*. "Diskret" bedeutet hierbei, dass das Signal in zeitlich gleichen Schritten abgetastet wird; dies geschieht in der Regel direkt bei der Aufnahme mit einem Analog-Digital-Umsetzer, z.B. in einer EL3xxx oder ELM3xxx.

Wird ein zeitkontinuierliches Signal mit einer Periode *T* abgetastet so ergibt sich eine Wertfolge:

$$x[n] = x(t = nT),\ n = 0..N-1$$

Diese aus *N* Werten bestehende Folge kann mittels DFT in ein diskretes Spektrum transformiert werden.

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-2\pi i n k}{N}} = \sum_{n=0}^{N-1} x[n] \left[\cos(\tfrac{2\pi n k}{N}) - i \sin(\tfrac{2\pi n k}{N})\right]$$

Die Variable *k* repräsentiert nun einen Frequenzkanal, auch Frequenz-bin genannt, und läuft wie die Variable *n* von 0 bis *N* —1.

Diskretisierung der Zeit und Quantisierung von Werten (Digitalisierung)

Für die digitale Verarbeitung analoger Signale sind zwei Operationen notwendig: Quantisierung von der analogen zur digitalen Wertdarstellung und Abtastung vom zeitlich kontinuierlichen physikalischen Signal zu diskreten Sequenz von quantisierten Werten.

Die Digitalisierung der Messwerte erfolgt in der I/O Klemme durch den Analog-Digital-Umsetzer. Üblicherweise erfolgt eine Quantisierung der Werte auf ein ganzzahliges Signal mit vorzeichenbehafteter 16-Bit Darstellung oder 24-Bit Darstellung. Die Verarbeitung in der TwinCAT 3 Condition Monitoring Library erfolgt durchgängig im 64-Bit IEEE Double-Fließkommaformat, welches moderne Prozessoren hardwaremäßig implementieren. Die zeitliche Abtastung erfolgt ebenfalls in der I/O Klemme durch die Abtastung des Eingangssignals mit einer definierten Abtastfrequenz. Diese berechnet sich für eine Task-Zyklus-Zeit T_c und einen Oversampling-Faktor c_s zu:

$$f_s = c_s \cdot \frac{1}{T_c}$$

Beispiel: Wählt man eine Task-Zyklus-Zeit von 1 ms und ein Oversampling von 10, so ergibt sich eine Abtastrate von $f_s = 10 * 1 / (10^{-3} * 1s) = 10$ kHz.

Die Abtastfrequenz ist unbedingt zu beachten

Die Abtastfrequenz ergibt sich in TwinCAT aus der Task Cycle Time und dem Oversampling-Faktor der genutzten Klemme: `fs := Oversamples*1000.0/TaskCycleTime_ms`. Vorsicht: die Task Cycle Time wird hier mit der Einheit Millisekunden, wie in TwinCAT üblich, verwendet.
Die Task Cycle Time kann in der SPS über den globalen Datentypen `PlcTaskSystemInfo` abgefragt werden. Siehe: INFOSYS Global Data Types

Abtasttheorem

Die wichtigste praktische Einschränkung bei der Anwendung der DFT ist die Beschränkung der eindeutig darstellbaren Frequenzen. Nach dem Nyquist-Theorem oder Abtasttheorem können (leicht vereinfacht beschrieben) nur Signale eindeutig dargestellt werden, deren höchste im Signal vorkommende Frequenz f_{max} maximal der halben Abtastfrequenz f_s entspricht. Entsprechend ist die Abtastfrequenz größer als die höchste im analogen Signal vorkommende Frequenz zu wählen.

$$f_s > 2 f_{max}$$

Die Anwesenheit höherer Frequenzen im analogen Signal führt zu sogenannten Aliasing. Das analoge Signal wird dann nicht mehr korrekt im diskreten Signal wiedergegeben. Höhere Frequenzen müssen deswegen noch vor der Analog-Digital-Umsetzung durch einstellbare Analogfilter aus dem analogen Signal entfernt werden.

Anti-Aliasing-Filter

Die EtherCAT-Klemmen EL3xxx bzw. ELM3xxx stellen je nach Klemmentyp verschiedene Filter zur Verfügung. Die EtherCAT-Klemme EL3632 verfügt z.B. über einen analogen parametrierbaren Tiefpassfilter 5. Ordnung, welcher zur Vermeidung von Aliasing eingesetzt werden soll. Die EL3751 sowie die ELM3xxx Module verfügen über mehrere Filterstufen, die zur Anti-Aliasing-Filterung sowie zur Nutzsignalfilterung ausgelegt werden können, siehe dazu auch den TC3 Filter Designer.

Frequenzauflösung

Da die Frequenzauflösung (diskrete Auflösung nach Frequenzkomponenten im Signal) es ermöglicht, verschiedene Signalanteile bestimmten Maschinenelementen und Schadstellen zuzuordnen, wird es in vielen Fällen von Interesse sein, eine möglichst hohe Auflösung der diskreten Frequenzachse zu erzielen.

Grundsätzlich bestimmt die Länge der Fourier-Transformation N die Schrittweite Δf auf der diskreten Frequenzachse $k \cdot \Delta f$. Eine grundlegende Überlegung dazu erleichtert das Verständnis: Um die Frequenz einer Sinusschwingung im Frequenzbereich darstellen zu können, muss die Messzeit mindestens eine volle Periode dieser Schwingung betragen. Daraus ergibt sich folgender Zusammenhang zwischen Auflösung Δf und Messzeit T_m:

$$\Delta f = \frac{1}{T_{\mathrm{m}}} = \frac{f_{\mathrm{s}}}{N}$$

Typischer PLC Code Syntax, siehe bspw. in der MAIN Routine im Beispiel Magnitudenspektrum [▶ 329]:

```
fSampleRate := cOversamples * (1000.0 / fTaskCycleTime);

fResolution := fSampleRate / cFFTLength;
```

Für eine hohe Frequenzauflösung ist demnach eine lange Messzeit erforderlich. Es ist möglich, die Eingangsdaten für die DFT durch symmetrisches Hinzufügen von Nullen vor und hinter dem Eingangssignal zu verlängern (Zero Padding). Dadurch erhöht sich die Länge N der Signalfolge bei gleichbleibender Abtastrate f_s , wodurch sich die diskrete Auflösung Δf verfeinert. Das Zero Padding fügt allerdings keine zusätzlichen Informationen zum Signal hinzu. Man unterscheidet bei Verwendung von Zero Padding zwei verschiedene Arten von Auflösung: Einerseits die Schrittweite zwischen einem Frequenz-bin zum nächsten auf der diskreten Frequenzachse, d.h. dem Übergang von $k \cdot \Delta f$ zu $(k+1) \cdot \Delta f$, und andererseits die Auflösung zur messtechnischen Unterscheidung von zwei nahe benachbarten Frequenzen des Eingangssignals.

Durch das Zero Padding wird zwar die diskrete Auflösung Δf kleiner, die messtechnische Auflösung verändert sich dadurch hingegen nicht. Eine Verfeinerung der messtechnischen Auflösung kann nur durch eine entsprechend lange Messzeit (in der Condition Monitoring Bibliothek für FFT-basierte Algorithmen oft als *WindowLength* bezeichnet) realisiert werden. In der Anwendung ist in der Regel die messtechnische Frequenzauflösung entscheidend, welche die Unterscheidbarkeit zwischen zwei eng benachbarten Signalfrequenzen beeinflusst.

Zero Padding

Durch Zero Padding wird keine Information zu dem zu analysierenden Signal hinzugefügt. Die Frequenzauflösung zur Unterscheidung von zwei nahe benachbarten Frequenzen des Signals wird demnach nicht verfeinert, sondern nur die numerische Auflösung der Frequenzachse.

Veranschaulichung an einem Beispiel:

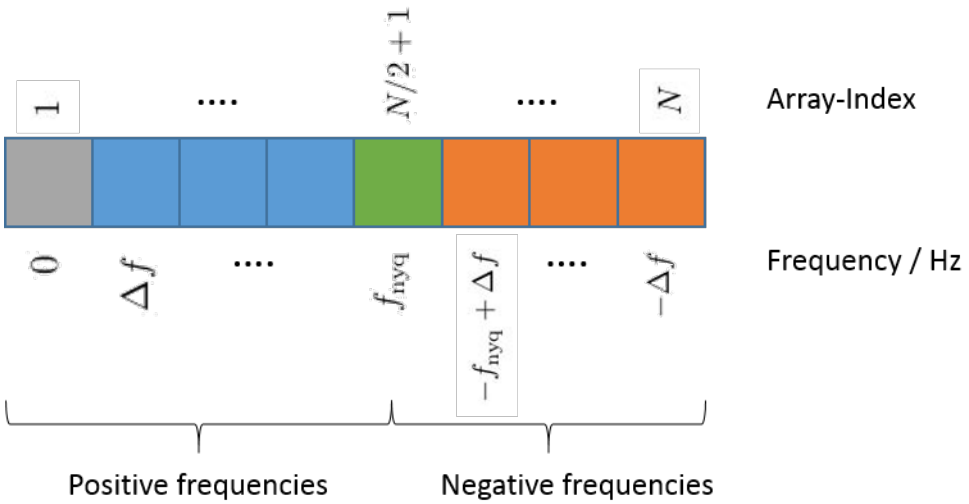
Bei einer Task Cycle Time T_c = 1 ms und einem Oversampling-Faktor von 10 (d.h. f_s = 10 kHz) werde ein Buffer der Länge 3200 gefüllt. Hieraus ergibt sich eine Messzeit von $T_m = T_c$ * 3200 / 10 = 320 ms und somit eine Auflösung im messtechnischen Sinn von Δf = 1 / 320 ms = 3,125 Hz. Wird für weiterführende Analysen/ Berechnungen eine FFT herangezogen, wird der Buffer symmetrisch um 2*448 Nullen erweitert um eine Länge N von 2^12 = 4096 > 3200 zu erreichen (N muss eine Potenz von 2 sein, siehe nächster Abschnitt). Die Numerische Auflösung verfeinert sich durch das Zero Padding somit auf Δf = 10 kHz / 4096 = 2,44140625 Hz.

Die diskrete Frequenzachse wird limitiert durch die Frequenz Null (Gleichanteil) und der Nyquist-Frequenz f_{nyq}, welche der halben Abtastfrequenz entspricht. Nach dem Nyquist-Theorem entspricht sie also der höchsten darstellbare Frequenz des aufgenommenen Signals. Wird das diskrete Spektrum *X[k]* in einem SPS Array mit dem Index m, welcher von 1..N läuft, gespeichert, so ergibt sich folgende Frequenzachse zu *X[k]*

```
fFrequency := (m-1) * fResolution; // m = 1..N/2+1
```

Hier repräsentiert $m = 1$ den Gleichanteil und $m = N/2+1$ die Nyquist-Frequenz. Die Indizes für m von $N/2+2$ bis N bilden die sogenannten negativen Frequenzen, welche praktisch allerdings nur relevant sind, wenn das Eingangssignal $x[n]$ zur FFT komplexwertig ist. Sehen Sie dazu den Abschnitt Spiegelfrequenzen [▶ 15].

Folgende Abbildung veranschaulicht den Aufbau der Frequenzachse zu einer DFT der Länge N (wobei N eine gerade Zahl sei).



Effiziente Berechnung durch FFT-Algorithmen

Die Schnelle Fourier-Transformation (FFT) bezeichnet genau genommen eine Familie von Algorithmen zur Diskreten Fourier-Transformation (DFT), die besonders effizient implementiert sind, sich im numerischen Ergebnis jedoch nicht von der DFT unterscheiden. Während die Komplexität einer naiv implementierten DFT mit *N* Zeitwerten O(N^2) beträgt, beträgt sie für eine FFT nur O (*N* * log*N*). Wenn *N* größere Werte erreicht, ist der erzielte Unterschied beträchtlich. Für *N*=1024 macht er beispielsweise schon einen Faktor von etwa Einhundert aus. Üblicherweise sind FFT-Algorithmen für Werte von *N* (mit N = Länge der FFT) definiert, die eine Zweierpotenz darstellen, also 256, 512, 1024 und so weiter.

Komplexwertiges Ergebnis

Die FFT (sowie die DFT) zerlegt das eingehende Signal *x*[*n*] in eine Reihe von Sinus- und Cosinusschwingungen. Für jede Frequenz gibt es je einen Koeffizienten für die Sinus- und Cosinuskomponenten. Beide Faktoren zusammen werden dargestellt als eine komplexe Zahl. Die Zerlegung findet sich in der *Eulerschen Formel* wieder:

$$e^{i\omega t} = \cos\omega t + i\sin\omega t$$

$$\mathrm{Re}\{e^{i\omega t}\} = \cos\omega t, \quad \mathrm{Im}\{e^{i\omega t}\} = \sin\omega t$$

Der Realteil Re{..} jedes Fourier-Koeffizienten entspricht der Cosinus-Komponente, der Imaginärteil Im{..} entsprechend der Sinus-Komponente. Dabei spiegelt das Verhältnis der beiden Komponenten die Phasenlage der Frequenzkomponente wieder:

$$\Phi(k) = -\mathrm{atan}\frac{\mathrm{Im}\{X(k)\}}{\mathrm{Re}\{X(k)\}}$$

In vielen Fällen interessiert nicht die genaue zeitliche Charakteristik des Signals, sondern nur das Betragsspektrum. Dieses lässt sich aus der Fourier-Transformierten durch Betragsbildung der komplexen Zahl ermitteln:

$$|X(k)| = \sqrt{\mathrm{Re}\{X(k)\}^2 + \mathrm{Im}\{X(k)\}^2}$$

Komplexer Datentyp

Das Ergebnis der FFT eines reellwertigen oder komplexwertigen Eingangssignals ist komplexwertig. Als Datentypen werden für die Signaldarstellung der Typ `LREAL` und `LCOMPLEX` verwendet. Wird ein Funktionsbaustein zur Berechnung des Betragsspektrums [▶ 172] oder Leistungsspektrums [▶ 202] genutzt ist das Ergebnis direkt wieder reellwertig.

Spiegelfrequenzen

Für die Fourier-transformierte eines reellen Signals gilt, dass die Koeffizienten für negative Frequenzen gleich den komplex konjugierten für positive Frequenzen sind. Ist *X[k]* die Fourier-transformierte von *x*[*n*] und *X**[*k*] die komplex konjugierte, so gilt für eine Fourier-Transformation mit *N* Punkten:

$$X[k] = X^*[N-k]$$

Weiterhin entspricht bei reellwertigen Signalen eine Zeitumkehr des Eingangssignals der komplexen Konjugation der Fourier-transformierten. Daraus folgt, dass die Spektralwerte für Frequenzen unterhalb der Nyquist-Frequenz in den Werten oberhalb der Nyquist-Frequenz gespiegelt auftreten. Da die Werte mit *k* > *N/2* für reelle Eingangssequenzen somit redundant sind, wird die Fourier-transformierte für reelle Sequenzen normalerweise auf die ersten *N/2* Werte beschränkt (gilt für *k* = 0..*N*—1).

Übersicht zur Berechnung von Spektralwerten

Die Condition Monitoring Bibliothek bietet eine Vielzahl an Möglichkeiten zur Berechnung von Spektralwerten. Folgende Tabelle bietet einen Überblick. Die Zusammenhänge der FFT mit (überlappenden) Fenstern werden im nächsten Kapitel näher betrachtet.

Funktionsbaustein	Input			Output			Anmerkung
	dimension	real	cplx	dimension	Real	cplx	
FB_CMA_ComplexFFT [▶ 112]	N	-	✓	N	-	✓	• FFT für komplex-wertige Eingangssignale • Keine Fensterung, Überlappungen und Skalierung • Berechnung des gesamten Spektrums
FB_CMA_MagnitudeSpectrum [▶ 172]	L	✓	-	N/2+1	✓	-	• Absolutbetrag der Spektralwerte • Fensterung und Skalierung einstellbar • Berechnung des gesamten Spektrums
FB_CMA_OrderPowerSpectrum [▶ 194]	V,P	✓	-	N/2+1	✓	-	• Ordnungsspektrum • Fensterung und Skalierung einstellbar • Berechnung des gesamten Spektrums
FB_CMA_PowerSpectrum [▶ 202]	L	✓	-	N/2+1	✓	-	• Leistungsbezogene Spektralwerte • Fensterung und Skalierung einstellbar • Berechnung des gesamten Spektrums
FB_CMA_PowerCepstrum [▶ 198]	L	✓	-	N/2+1	✓	-	• Leistungscepstrum • Fensterung und Skalierung einstellbar • Berechnung des gesamten Cepstrums
FB_CMA_RealFFT [▶ 216]	N	✓	-	N/2+1, N	-	✓	• FFT für reell wertige Eingangssignale • Keine Fensterung, Überlappungen und Skalierung • Berechnung des gesamten Spektrums
FB_CMA_SlidingDFT [▶ 224]	*	✓	-	nBins	-	✓	• Online DFT, d. h. Sample-weises Update

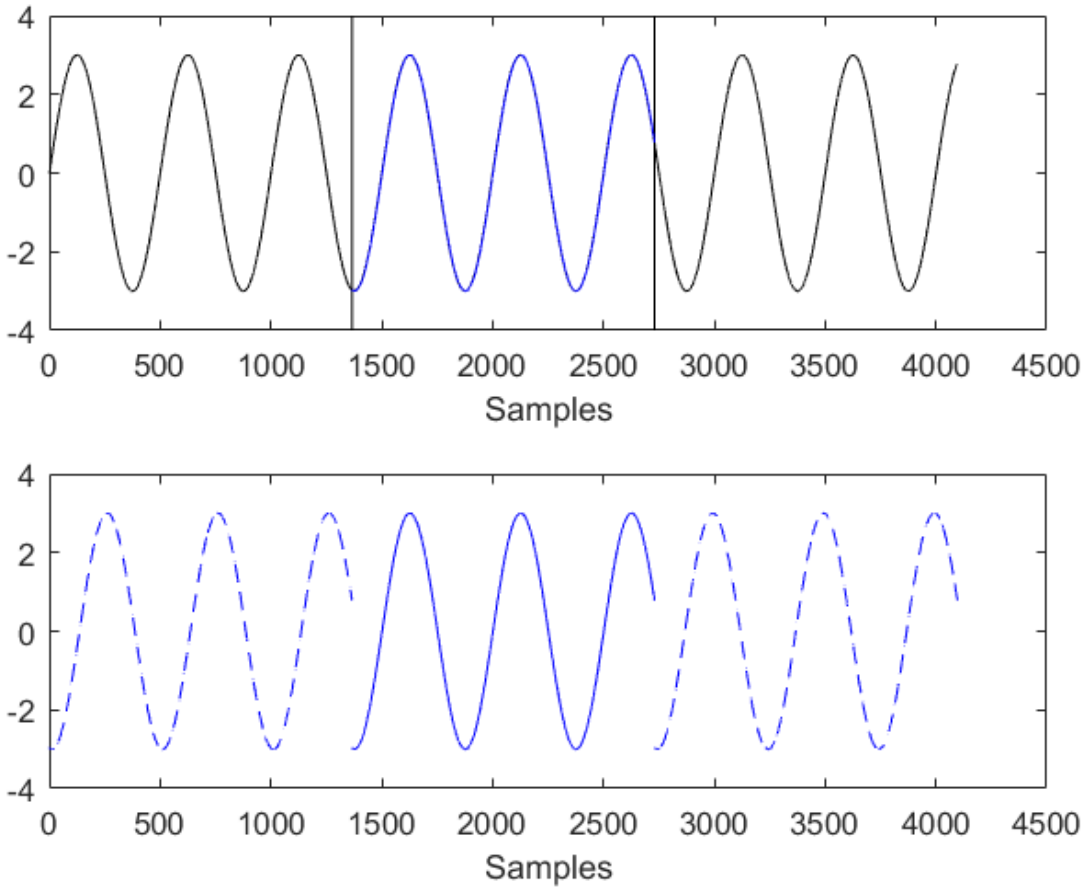
Funktionsbaustein	Input			Output			Anmerkung
	di-men-sion	real	cplx	di-mensi-on	Real	cplx	
							• Für minimale Latenz • Zu berechnende Frequenzbereiche konfigurierbar
FB_CMA_SparseSpectrum [▶ 229]	M	✔	-	nBins	✔	✔	• Berechnung einzelner, konfigurierbarer Spektrallinien • Fensterung und Skalierung einstellbar
FB_CMA_SpikeEnergySpectrum [▶ 233]	L	✔	-	N/2+1	✔	-	• Detektieren periodisch auftretender, hochfrequenter Stöße • Fensterung und Skalierung einstellbar
FB_CMA_ZoomFFT [▶ 266]	N	✔	✔	N/(2D)+1	-	✔	• Berechnung eines Frequenz-Ausschnittes des Spektrums • Keine Fensterung, Überlappungen und Skalierung

Hierbei entspricht `N` der FFT Länge, `L` der Fensterlänge (respektive Überlappung) und `nBins` der Anzahl konfigurierter Spektralwerte. Ferner bezeichnet `D` den Dezimierungsfaktor. Die Länge des Eingangssdatenpuffers des Funktionsbausteins FB_CMA_SlidingDFT ist nicht auf eine feste Größe beschränkt. Die Längen `L`, `V` des Ordnungsspektrums müssen spezifisch auf Vibrations- und Positionsdaten angepasst werden.

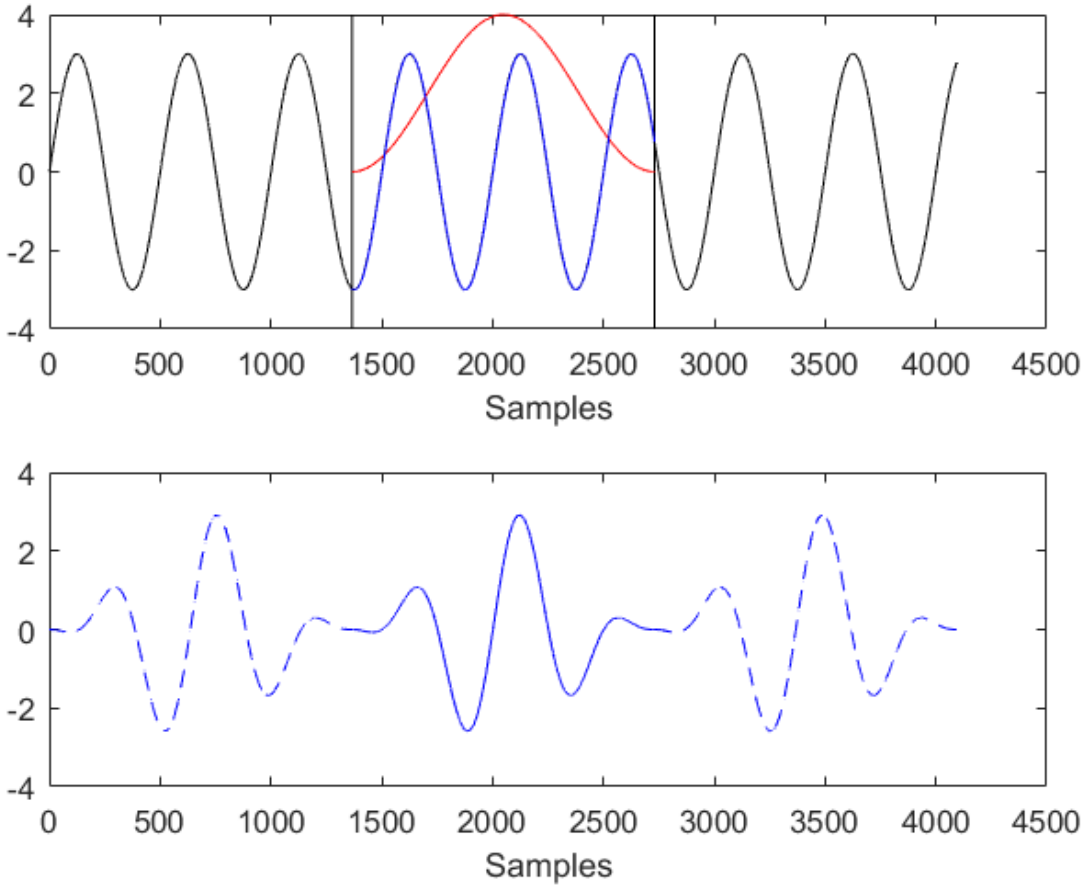
2.1.2 Analyse von Daten-Streams

Blockweise FFT-Analyse aus einem Daten-Stream

Die DFT/FFT ist definiert auf einem zyklisch fortgesetzten, periodischen Signal. Dies führt zu einer zunächst überraschenden Folgerung: Wenn eine FFT-Analyse eines langen Signals benötigt wird, kann nicht einfach das Eingangssignal in Abschnitte unterteilt und mit der DFT transformiert werden. Denn sobald der letzte Wert in einem solchen Abschnitt nicht mit dem ersten übereinstimmt, behandelt die FFT dies als Sprung in der zyklischen Fortsetzung, der sich im Spektrum deutlich abbildet (*Spectral Leakage*). Das Prinzip wird in folgender Abbildung deutlich. Es wird aus dem Gesamtsignal (schwarz) ein Teilsignal (blau) ausgeschnitten. Die FFT impliziert eine zyklische Fortsetzung des Teilsignals (untere Abbildung) und geht von Sprüngen im zu transformierenden Signal aus, was sich im Spektrum deutlich niederschlägt.

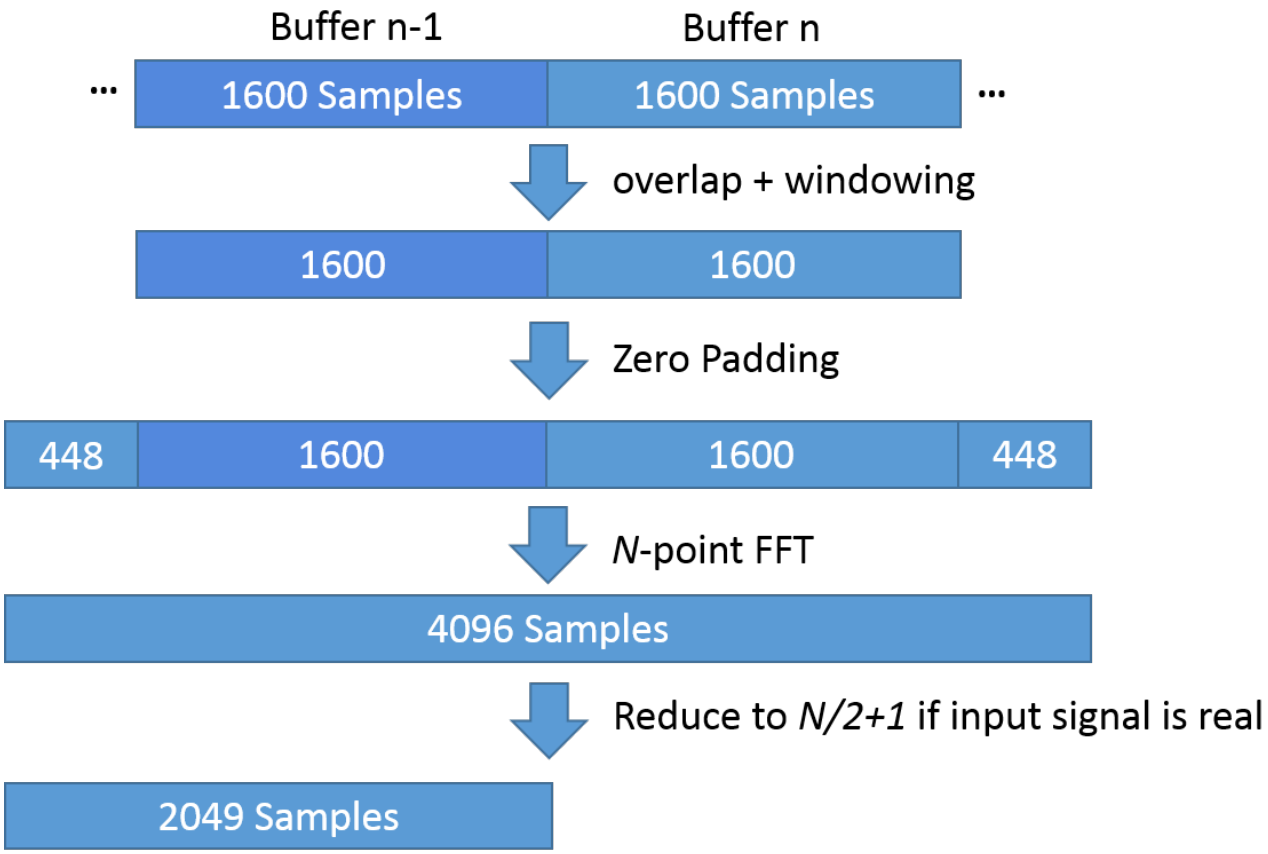


Abhilfe schafft hier die Gewichtung der Signalabschnitte vor der Transformation mit einer geeigneten Fensterfunktion (für Details siehe nächster Abschnitt). Ein geeignetes Fenster ist so beschaffen, dass sowohl Zeitwerte ganz am Anfang als auch solche ganz am Ende mit einem Faktor Null bzw. nahezu Null multipliziert werden. Folgende Abbildung zeigt selbiges Szenario wie gerade beschrieben, allerdings nun mit einer Fensterfunktion (rot). Die Sprünge in der zyklischen Fortsetzung verschwinden durch die Fensterung, allerdings ist zu beachten, dass die Eigenschaften des Fensters natürlich im Spektrum des gefensterten Teilsignals zu sehen sind. Die Fenstereigenschaft führt jedoch zu in der Regel deutlich geringeren Einflüssen im Spektrum.



Problematisch bei der Fensterung ist, dass Werte am Rand eines Fensters kaum Berücksichtigung im Spektrum finden. Sollten sich gerade in diesem Bereich Signalmerkmale, die auf mögliche Schädigungen schließen lassen, befinden, könnte es zu einem wesentlichen Informationsverlust kommen. Damit keine Informationen verloren gehen, wird diese Fensterung in der TwinCAT Condition Monitoring Bibliothek mit überlappenden Signalabschnitten vorgenommen. Bei Verwendung des als Standard verwendeten Hann-Fensters wird eine 50% Überlappung empfohlen. Dadurch befinden sich Samples, die sich in einem Fensterabschnitt am Rand befinden, im nächsten Fensterabschnitt in der Mitte.

Folgende Abbildung zeigt schematisch die Vorgehensweise einer FFT-Analyse aus einem Daten-Stream mit 50% Überlappung.



Aus dem Daten-Stream werden zunächst Puffer mit definierter Länge gefüllt, hier 1600 Werte. Bei der Auswertung der Daten aus Puffer n wird der vorangegangene Puffer hinzugezogen und das nun 3200 Werte lange Datenpaket gefenstert. Das Maximum der Fensterfunktion liegt dabei genau zwischen den beiden Puffern und fällt in Richtung Rand der beiden Puffer auf den Wert Null. Durch Zero Padding wird das Datenpaket auf eine Länge von 4096 Werten verlängert, so dass die Länge eine Potenz von 2 ist und somit durch einen FFT-Algorithmus effizient berechnet werden kann. Das Ergebnis der FFT ist dann ein Datenpaket mit 4096 Werten, welches ggf. auf 2049 Werte reduziert werden kann, falls die Eingangsdaten alle reellwertig sind (siehe Spiegelfrequenzen [▶ 15]).

Während der Auswertung von Puffer n, wird Puffer n+1 aufgefüllt bei dessen Auswertung dann wiederum Puffer n hinzugezogen wird. Durch dieses Vorgehen entsteht immer ein 50%iger Überlappungsbereich der gefensterten Zeitbereiche.

Analyse eines Daten-Stream in TwinCAT 3

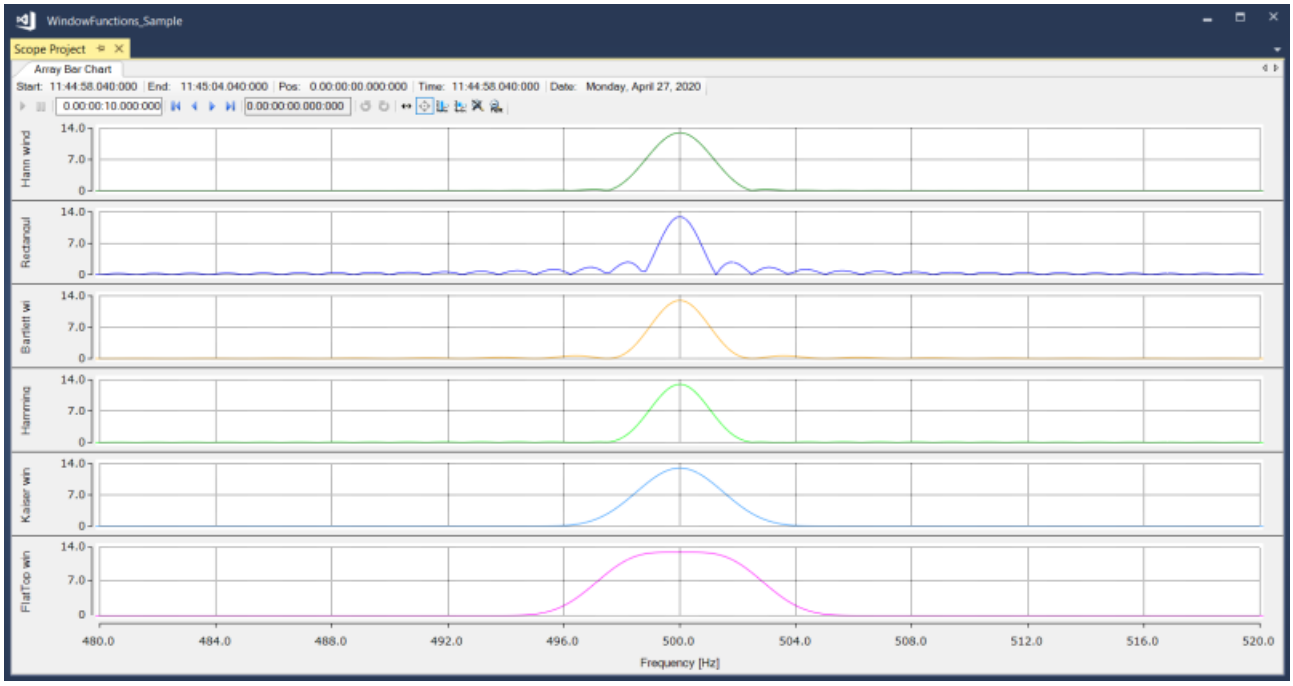
Das in der obigen Abbildung gezeigte Schema der Signalanalyse wird in der Condition Monitoring Bibliothek durch den FB_CMA_MagnitudeSpectrum sowie FB_CMA_PowerSpectrum realisiert. Als Anwender ist nur eine Konfiguration der Parameter (Länge der Puffer, Länge der FFT, Fensterfunktion, Überlappung, …) sowie die Bereitstellung der Daten-Puffer notwendig. Siehe z. B.: ST_CM_PowerSpectrum_InitPars [▶ 313].

Fensterfunktionen

Die Eigenschaften der verwendeten Fensterfunktionen sind im Ergebnis der Transformation wiederzufinden. Es wird nicht das Signal $x[n]$ Fourier-transformiert, sondern das Signal $x[n] \cdot w[n]$, mit $w[n]$ als Zeitwerte der Fensterfunktion. Beachten Sie hierzu die grundlegenden Eigenschaften von Fensterfunktionen.

Wird „keine“ Fensterfunktion verwendet, also ein Signalabschnitt aus einem längeren Gesamtsignal herausgenommen entspricht dies der Verwendung eines Rechteckfensters. Zum Vergleich der Eigenschaften von Fensterfunktionen gegenüber einer Rechteckfensterung ein Beispiel: Ein harmonischer Sinus mit Amplitude 13 und Frequenz 500 Hz wird mit einer Abtastrate von 10 kHz abgetastet, mit einer Fensterfunktion der Länge 3200 Samples gefenstert und anschließend das Magnitudenspektrum (mit der Skalierungsoption [▶ 390] `eCM_PeakAmplitude`) gebildet. In folgender Darstellung wird das Magnitudenspektrum bei Verwendung der verfügbaren Fensterfunktionen gegenübergestellt.

Ein Beispiel zur Rekonstruktion der nachfolgenden Grafik finden Sie hier: Fensterfunktionen [▶ 336]



In der Grafik sind zwei wesentliche Eigenschaften von Fensterfunktionen zu erkennen:

- Die Breite des *mainlobe* (auch Hauptzipfel), hier die Breite des Zipfels um 500 Hz.
- Die Dämpfung der *sidelobes* (auch Nebenmaxima) bezogen auf das Maximum des *mainlobe*.
- Die Amplitudentreue harmonischer Signale, vgl. die Maximalwerte bei Vorgabe einer Schwingung mit Amplitude von 13.

Die verschiedenen Fenstertypen bilden jeweils einen trade-off zwischen diesen 3 wesentlichen Eigenschaften.

Die Breite des *mainlobe* beeinflusst die erreichbare Frequenzauflösung. Die Höhe der *sidelobes* zeigt die spektralen Verfälschungen (*spektral leakage*) auf, da diese nur durch das Fenster entstehen und nicht durch das zu analysierende Signal. Die *Krümmung* des *mainlobes* im Bereich -0,5 bins bis +0,5 bins um das Maximum charakterisiert den maximalen Amplitudenfehler, welcher bei harmonischen Signalen auftreten kann (*scalloping losses*).

Es ist zu erkennen, dass das **Rechteckfenster** eine sehr gute Frequenzauflösung erlaubt, jedoch starke spektrale Verfälschungen nach sich zieht, was z.B. problematisch wird, wenn nun neben dem Peak bei 500 Hz ebenfalls eine Frequenzkomponente bei 550 Hz mit einer Amplitude von 0,5 auftritt. Außerdem der maximal mögliche Amplitudenfehler mit -36,34% sehr hoch. Das **Hann-Fenster** verringert die *sidelobes* erheblich, verringert jedoch auch die erreichbare Frequenzauflösung, was dennoch einen guten Kompromiss darstellt solange keine exakte Bestimmung von Amplituden von sinusförmigen Signalen benötigt wird. Der maximal mögliche Amplitudenfehler liegt hier bei -15,12%. Das Hann-Fenster ist eines der bekanntesten Fensterfunktionen und ist deshalb als Standard in der Condition Monitoring Bibliothek eingestellt. Ist eine Amplitudentreue bei harmonischen Signalen gefordert ist ein **Flat-top-Fenster** (SFT5M) zu verwenden, welches eine möglichst flache Krümmung im zentralen *mainlobe* Bereich aufweist (maximaler Amplitudenfehler -0,045%). Allerdings ist der *mainlobe* hier sehr breit, wodurch sich dieses Fenster nur bei der Analyse rein harmonischer Signale empfiehlt.

Eine wichtige Kenngröße der Frequenzauflösung bei Nutzung einer Fensterfunktion ist die Equivalent Noise Bandwidth (ENBW).

$$ENBW = N\,\frac{\sum w^2[n]}{(\sum w[n])^2}\Delta f = \frac{\sum w^2[n]}{(\sum w[n])^2} f_s$$

Die Größe Δf ergibt sich, über die FFT-Länge N und die Abtastrate f_s (siehe Fourier-Analyse [▶ 11]). Der Ausdruck in der Gleichung vor dem Δf wird definiert über die Eigenschaften des Fensters und ist für ein Rechteckfenster 1 und z.B. für das Hann Fenster 1,5 und für das SFT5M 3,885.

Die Wahl des verwendeten Fensters, der zugehörigen Parameter sowie die zu verwendende Überlappung wird für die betreffenden Funktionsbausteine über die jeweilige Baustein-spezifische Struktur mit Initialisierungsparametern realisiert, z.B. ST_CM_MagnitudeSpectrum_InitPars [▶ 306].

Verfügbare Fensterfunktionen

Für ein Fenster der Länge N, stehen in der Condition Monitoring Bibliothek die nachfolgenden Fensterfunktionen zur Verfügung. Hierbei bezeichnet I_0 die modifizierte Bessel-Funktion erster Gattung und Ordnung Null. Die Parametrierung des `eCM_KaiserWindow` sowie des `eCM_FlatTopWindow` erfolgt über T_CM_WindowParameters [▶ 292] in den initPars Strukturen. Die letzte Spalte gibt die empfohlene Überlappung in Prozent der Fensterlänge an. Exemplarisch sind für `eCM_KaiserWindow` das Fenster Kaiser-4 mit

$$a_0 = \beta = 4\pi$$

und für `eCM_FlatTopWindow` das Fenster SFT5M mit

$$a_0 = 0.209671, a_1 = -0.407331, a_2 = 0.281225, a_3 = -0.092669, a_4 = 0.0091036$$

gewählt worden.

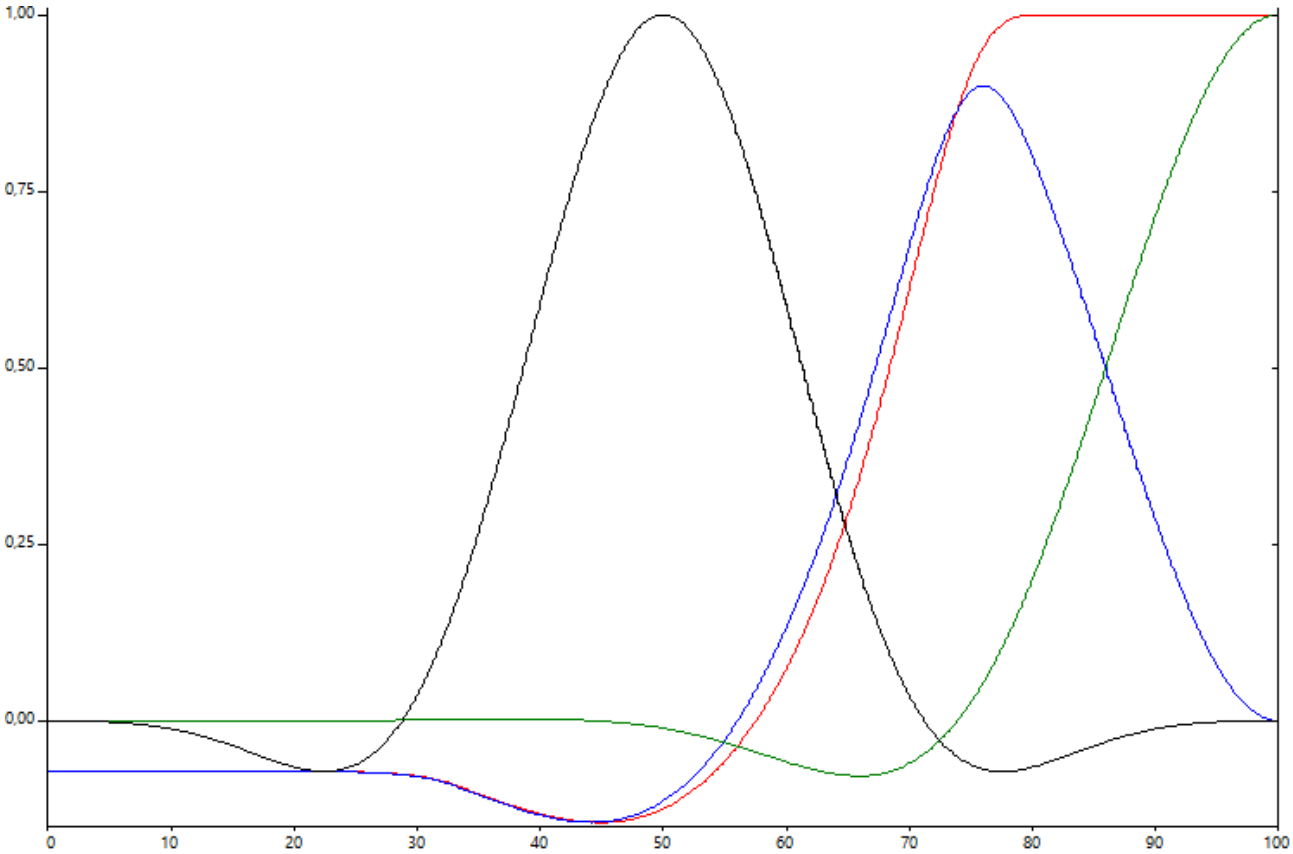
Bezeichner	PID	Definition	Überlappung
`eCM_HannWindow`	16#05300901	$w[n] = 0.5 \cdot (1 - \cos(2\pi n/N))$	50%
`eCM_RectangularWindow`	16#05300902	$w[n] = 1$	0%
`eCM_BartlettWindow`	16#05300905	$w[n] = 1 - \lvert 2n/N - 1 \rvert$	50%
`eCM_HammingWindow`	16#05300906	$w[n] = 0.54 - 0.45\cos(2\pi n/N)$	50%
`eCM_KaiserWindow`	16#05300907	$w[n] = I_0\left(\beta\sqrt{1-(2n/N-1)^2}\right)/I_0(\beta)$	67% (Kaiser-4)
`eCM_FlatTopWindow`	16#05300917	$w[n] = a_0 + \sum_{k=1}^{4}\left\{(-1)^k a_k \cos(2k\pi n/N)\right\}$	76% (SFT5M)

Berechnung der empfohlenen Überlappung

Die empfohlene Überlappung der gefensterten Signalabschnitte ergibt sich aus der Betrachtung folgender Eigenschaften der Fensterfunktion:

- *amplitude flatness*: Flachheit der überlappenden Fensterfunktion, d.h. die Gewichtung der einzelnen Samples.
- *overlap correlation*: Korrelation zwischen den einzelnen Samples der überlappenden Funktionen.

Die empfohlene Überlappung liegt an der Stelle, wo die Differenz beider Eigenschaften maximal wird. D.h. die bestmögliche Flachheit bei möglichst geringer Korrelation erreicht wird. Dies wird in der folgenden Grafik für das SFT5M Fenster (`eCM_FlatTopWindow`) gezeigt.



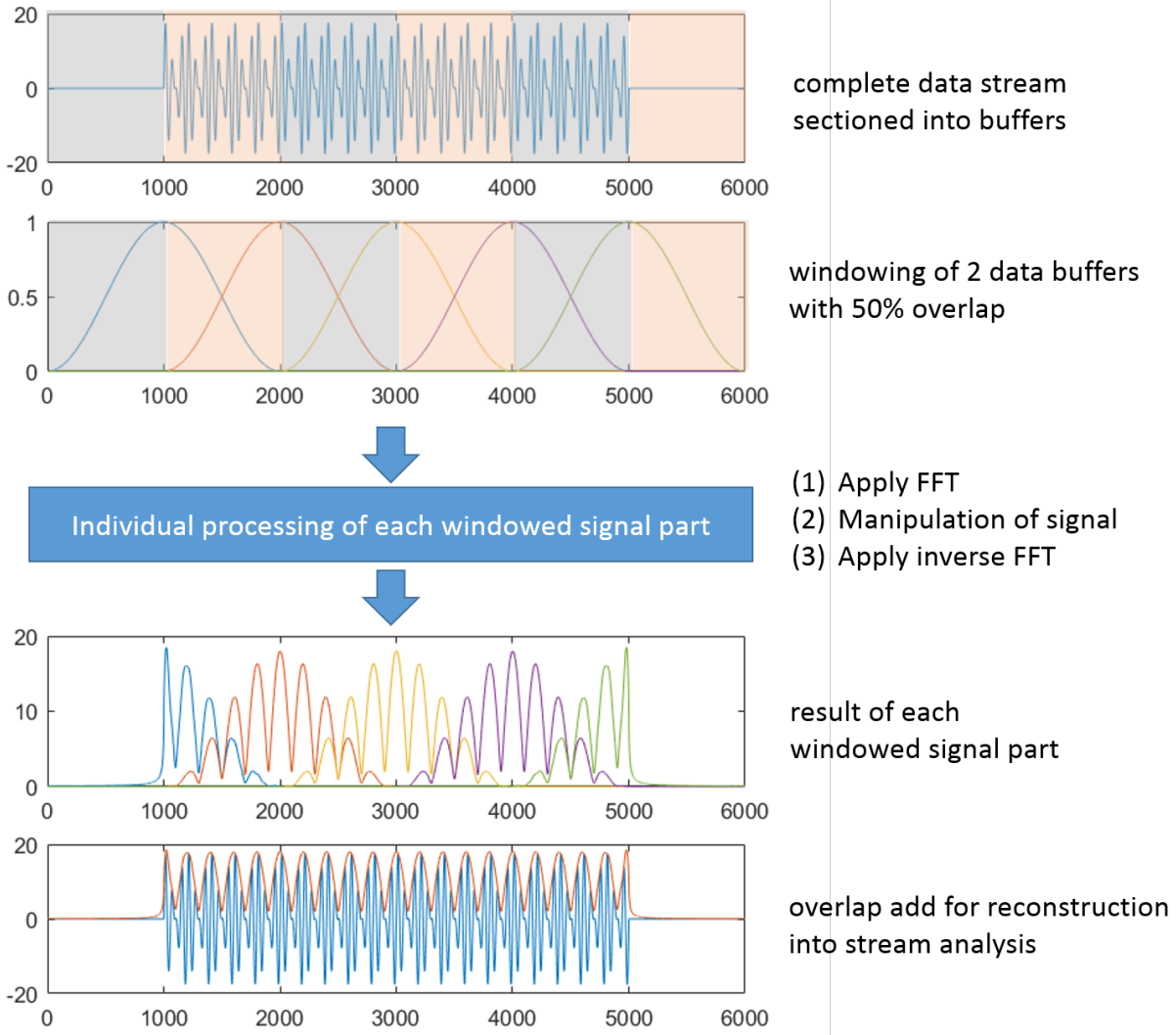
Fensterfunktion (schwarz), *amplitude flatness* (rot), *overlap correlation* (grün), empfohlene Überlappung (blau).

Für weitere Informationen siehe:

- G. Heinzel et.al.: Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions and some new flat-top windows. 2002.
- In der Condition Monitoring Bibliothek erfolgt die Berechnung der empfohlenen Überlappung über F_CM_CalculateRecommendedOverlap [▶ 287]. Die Funktion kann explizit genutzt werden, die Algorithmen verwenden diese bei entsprechender Konfiguration auch implizit in Abhängigkeit der eingestellten Fensterfunktion. Natürlich kann auch eine frei wählbare Überlappung eingestellt werden. Vgl. z.B. eWindowType, eWindowTypeParameters und nOverlap in ST_CM_PowerSpectrum_InitPars [▶ 313].

Overlap-Add Methode

Einige Funktionsbausteine der Condition Monitoring Bibliothek arbeiten über Manipulation des Spektrums des Eingangssignals, d.h. wie oben beschrieben, wird zunächst das Eingangssignal in überlappende Teilsignale zerlegt und Fourier-transformiert. Dann wird eine Manipulation des Spektrums vorgenommen und eine inverse Fourier-Transformation berechnet. Je nach verwendeter Fensterfunktion ist eine Korrekturfunktion notwendig, um den Einfluss des Fensters zu kompensieren. Die einzelnen ebenfalls überlappenden Ergebnisse werden dann am Ausgang des Bausteins summiert, so dass wieder ein Daten-Stream am Ausgang des Bausteins entsteht. Dieses Vorgehen wird als overlap-add Methode bezeichnet und ist beispielhaft anhand der Berechnung der Signaleinhüllenden (FB_CMA_Envelope [▶ 146]) in folgender Grafik gezeigt.



Overlap-Add in TwinCAT 3

Das Verfahren wird innerhalb einiger Funktionsbausteine der Bibliothek genutzt und muss vom Anwender nicht selber implementiert werden. Als Anwender ist nur eine Konfiguration der Parameter (Länge der Puffer, Länge der FFT, …) sowie die Bereitstellung der Daten-Puffer notwendig.

2.1.3 Getriggerte Analyse eines Zeitabschnitts

Motivation

Neben der zeitlich durchgängigen Analyse eines Prozesses, z. B. das Vibrationsverhalten einer sich kontinuierlich drehenden Welle, ist ein ebenso häufig auftretender Anwendungsfall das Analysieren eines definierten Zeitfensters. Anwendungsfälle sind z. B. die Analyse von Vibrationssignalen an einem Bohrkopf, einem Fräsaggregat oder an einer Welle, welche sich nur über bestimmte Zeitabschnitte dreht.

Der Vorteil einer in die Steuerung integrierten Analyse wird hier besonders deutlich. Die Steuerung veranlasst in der Regel einen bestimmten Prozessschritt, z. B. das Bohren. Entsprechend kann mit der Maschinenablaufsteuerung nicht nur der Prozessschritt, sondern auch der dazu passende Analyseschritt getriggert werden.

Umsetzung in der Condition Monitoring Bibliothek

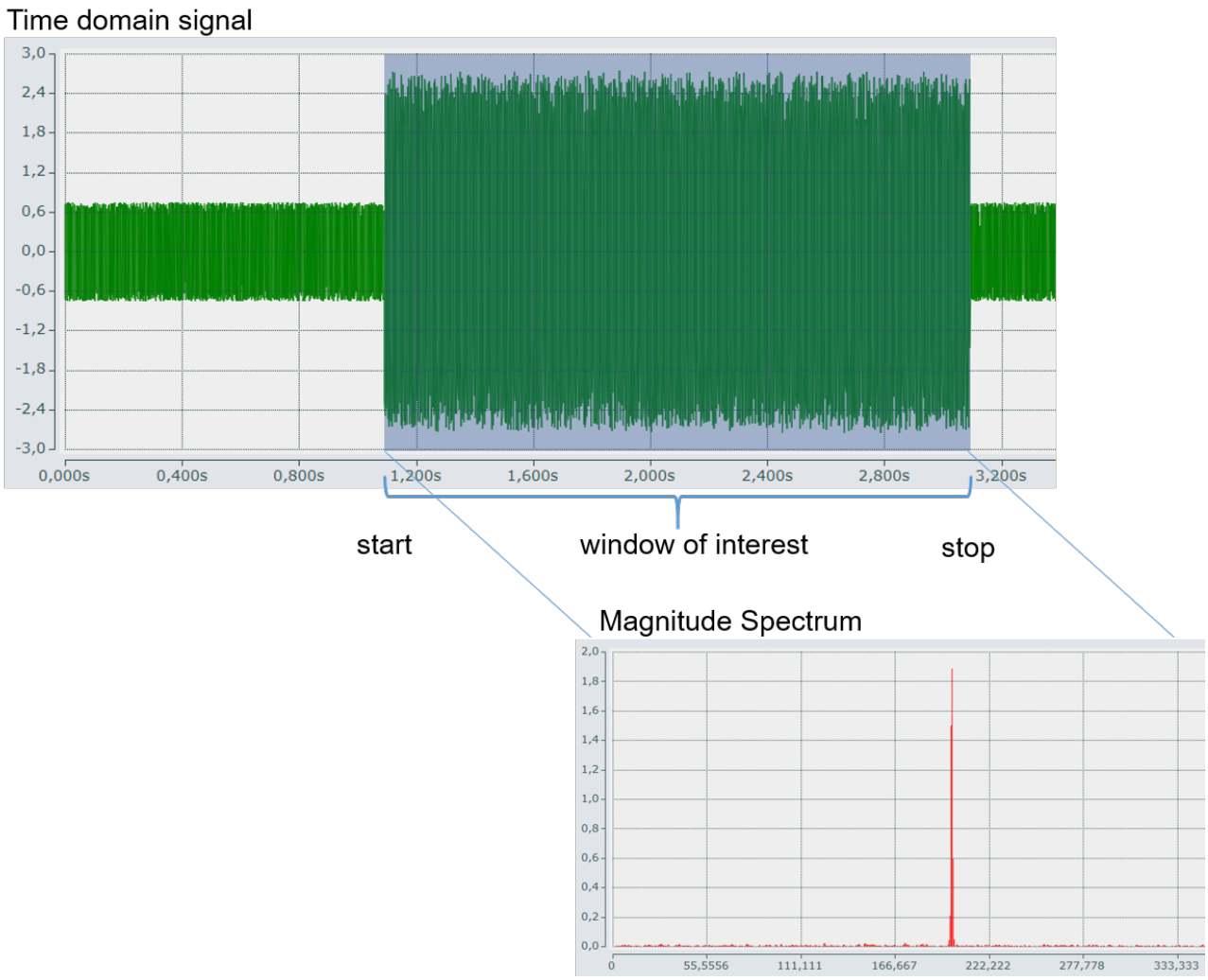
Die Auswertung eines definierten Zeitfensters unterscheidet sich hinsichtlich der Analysefunktionen nicht von der kontinuierlichen Analyse eines Daten-Streams. Einziger Unterschied ist, dass jede getriggerte Analyse für sich allein und nicht in einem kontinuierlichen Zusammenhang steht. Entsprechend können alle Analysebausteine der TC3 Condition Monitoring Bibliothek für kontinuierliche und auch für getriggerte Zeitfensteranalysen genutzt werden. Es sind nur zwei Punkte bei der Konfiguration der Analysekette zu beachten:

- Über den FB_CMA_Source müssen genügend Daten (ein genügend großes Fenster an Daten) an die Analysekette gesendet werden, damit ein gültiges Ergebnis berechnet werden kann. Beachtet werden muss bspw. die WindowLength eines FFT-basierten Algorithmus.
- Um die Einzelanalysen voneinander sauber zu trennen ist lediglich dafür Sorge zu tragen, dass alle Analysebausteine mit Gedächtniseigenschaft (siehe dazu die jeweilige Dokumentation der einzelnen Algorithmen; Abschnitt Gedächtniseigenschaften) nach einer abgeschlossenen Analyse zurückgesetzt werden. Dazu bietet der FB_CMA_Source die Methode `ResetAnalysisChain()`. Alternativ kann auch an jedem einzelnen Analysebaustein die Methode `ResetData()` genutzt werden, welche dann nur auf den einzelnen Analysebaustein wirkt.

Beispiel-Implementierung

Ein Beispiel anhand eines synthetischen Signals sei im Folgenden beschrieben. Das synthetische Signal besteht aus einem Grundrauschen und einem additiv überlagerten Sinussignal mit der Frequenz 200 Hz und Amplitude 2. Das Sinussignal wird im Wechsel immer 2 Sekunden ein- und wieder ausgeschaltet.

Wird bei einem solchen Signal eine kontinuierliche Auswertung gewählt, ist nicht sicherzustellen in welchen Zeitabschnitten die Signalabschnitte liegen, welche zur Auswertung genutzt werden. Entsprechend ist es hier ratsam immer dann ein Auswertefenster für eine definierte Messzeit zu starten, wenn das Sinussignal eingeschaltet wird. Untenstehende Abbildung zeigt schematisch das beschriebene synthetische Signal sowie das Amplitudenspektrum auf Basis des angedeuteten Auswertefensters.



Den Quellcode sowie eine detailliertere Beschreibung des Samples finden Sie hier: Eventbasierte Frequenzanalyse [▶ 366].

2.1.4 Skalierung von Spektren

Magnituden- und Powerspektrum

Es gibt mehrere gängige Arten, das Spektrum auszuwerten:

- Das Magnitudenspektrum [▶ 172], das linear skalierte Beträge der komplexwertigen Spektralwerte $|X[k]|$ verwendet. Es wird auch als Betragsspektrum oder Amplitudenspektrum bezeichnet.
- Das Powerspektrum [▶ 202] (auch Leistungsspektrum), dessen Werte die Quadrate der Magnitudenwerte $|X[k]|^2$ darstellen.

Die Verwendung des Leistungsspektrum ist dann sinnvoll, wenn Leistungswerte aufsummiert bzw. zusammengefasst werden, da die quadrierten Spektralwerte $|X[k]|^2$ über das Parseval'sche-Theorem exakt mit dem Effektivwert des Zeitsignals zusammenhängen.

Das Parseval'sche Theorem besagt, dass die Leistung des Signals $x[n]$ in der Zeitdarstellung gleich der Leistung des Signals in der Fourier-transformierten ist:

$$\sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2$$

Wird nun der RMS-Wert (Effektivwert) des Signals $x[n]$ berechnet, kann man dies im Zeitbereich oder im Frequenzbereich realisieren, da beide Darstellungen bezüglich der Leistung identisch sind:

$$\mathrm{RMS}\{x[n]\} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}|x[n]|^2} = \sqrt{\frac{1}{N^2}\sum_{k=0}^{N-1}|X[k]|^2}$$

In der Praxis können so z. B. RMS-Werte für eingeschränkte Frequenzbereiche eines Signals berechnet werden. Praktische Skalierungsoptionen [▶ 390] der Condition Monitoring Bibliothek die sich auf die benannten Eigenschaften in diesem Abschnitt beziehen sind z. B. `eCM_ROOT_POWER_SUM` sowie `eCM_RMS`.

Die Spektrale Leistungsdichte

Ein weiterer für die Auswertung von Spektren wesentlicher Begriff ist die Spektrale Leistungsdichte oder *Power Spectral Density* (PSD). Sie bezeichnet den Leistungswert bezogen auf die effektive Frequenzauflösung, welche durch die *Equivalent Noise Bandwidth* (ENBW) angegeben wird

$$\mathrm{PSD} = \frac{1}{ENBW}|X[k]|^2$$

Ein Blick auf die physikalischen Einheiten von Signal, Betragsspektrum und PSD erläutert anschaulich die Zusammenhänge. Wird ein Signal $x[n]$ in Volt (V) gemessen, so wird das diskrete Betragsspektrum $|X[k]|$ ebenfalls in Volt angegeben. Durch das Quadrieren wird das Leistungsspektrum in V^2 dargestellt. Das Leistungsdichtespektrum soll nach Definition eine Leistungsgröße (V^2) bezogen auf die Frequenz in Hz darstellen. Durch Beziehung des Leistungsspektrums auf die effektive Frequenzauflösung in Hertz (Hz) ergibt sich die Einheit V^2/Hz.

Ebenfalls kann diese Darstellung für Magnituden-Werte genutzt werden. Es ergibt sich entsprechend die sogenannte Linear Spectral Density (LSD) zu

$$\mathrm{LSD} = \sqrt{\frac{1}{ENBW}}|X[k]|$$

Dezibel-Skala

In der Schwingungsanalyse und Maschinenakustik ist die Umrechnung von Werten von der linearen in die logarithmische „Dezibel-Skala“ gebräuchlich. Die Dezibel-Skala ermöglicht eine gute Interpretation, wenn sowohl sehr große Werte als auch sehr kleine Werte in einem Spektrum vorkommen und sowohl in Bereichen großer sowie kleiner Werte ausgewertet werden soll. Die Umrechnung des Betragsspektrums in die Dezibel-Skala erfolgt über:

$$|X[k]|_{\mathrm{dB}} = 10\log_{10}|X[k]|^2 = 20\log_{10}|X[k]|$$

Die Dezibel-Skala kann sowohl über 10 Mal den Logarithmus des Leistungsspektrums als auch 20 Mal den Logarithmus des Magnitudenspektrums berechnet werden. Das Ergebnis einer Berechnung aus FB_CMA_MagnitudeSpectrum [▶ 172] und FB_CMA_PowerSpectrum [▶ 202] ist also in der Dezibel-Skala identisch.

Die Umrechnung von Ergebnissen in die Dezibel-Skala wird von der Condition Monitoring Bibliothek komfortabel über eine Bool’sche Variable in den Funktionsbaustein-Initialisierungsparametern aktiviert, siehe z. B. ST_CM_PowerSpectrum_InitPars [▶ 313].

Skalierungsoptionen nach Signaltyp

Durch die Auswahl einer passenden Skalierungsoption [▶ 390] können die durch den Powerspektrum [▶ 202] oder Magnitudenspektrum [▶ 172] Baustein berechneten Spektralwerte automatisch auf eine gewünschte Referenzgröße angepasst werden. Die richtige Interpretation der Referenzgröße ist hierbei von besonderer Bedeutung.

Bei den Skalierungsoptionen ist in der Praxis und unter der Annahme eines stationären Signals zunächst wichtig zu unterscheiden zwischen deterministischen und stochastischen Signalen.

Deterministische Signale bestehen aus periodischen Schwingungen mit definierter Frequenz. Entscheidend ist, dass die Frequenzauflösung (ENBW) breiter ist als eine harmonische Frequenz. Also ist die gesamte Leistung dieser Frequenzkomponente des Signals in diesem Frequenzkanal zusammengefasst.

Daher sind die Spektralwerte direkt auf eine Amplitude (Skalierungsoption [▶ 390] `eCM_PeakAmplitude`) oder einen RMS-Wert eines äquivalenten Sinussignals skalierbar. Wenn das Signal nicht in die Mitte des Frequenzkanals fällt, treten sogenannte Scalloping Losses auf, vgl. Abschnitt Fensterfunktionen [▶ 20], welche die beobachtete maximale Amplitude vermindern. Neben der Verwendung eines Flat-top-Fensters, kann dies nachträglich bei Verwendung eines bspw. Hann-Fensters ausglichen werden, indem die Power-Werte aus benachbarten Frequenzkanälen summiert ausgewertet werden, siehe Skalierungsoption [▶ 390] `eCM_ROOT_POWER_SUM` sowie `eCM_RMS`.

Stochastische oder breitbandige Signale erfordern die Auswertung von Power Spectral Densities (PSD) bzw. Linear Spectral Densities (LSD), da über einen definierten Frequenzbereich alle Frequenzen Signalleistung beinhalten. In diesem Fall sind die ermittelten Leistungswerte abhängig von der effektiven Breite der Frequenzkanäle der FFT. Sie müssen sinnvollerweise auf diese Bandbreite bezogen werden, um Resultate zu erhalten, die von den Parametern der Auswertung unabhängig sind. Weil die effektive Breite der Frequenzkanäle bei der Verwendung von Fensterfunktionen von der Länge und Form der Fensterfunktion abhängt, muss in diesem Fall die oben erwähnte Equivalent Noise Bandwidth (ENBW) verwendet werden, siehe Skalierungsoption [▶ 390] `eCM_PowerSpectralDensity`.

Die Skalierung anhand der PSD ermöglicht keine konsistente Skalierung des "Gleichspannungsanteils". Falls benötigt, sollte dieser durch Tiefpassfilterung oder Mittelung bestimmt werden.

Falls ein Signal sowohl deterministische Anteile als auch breitbandige Anteile enthält, müssen beide Skalierungen unabhängig voneinander verwendet werden, um von den Verarbeitungsparametern unabhängige Werte zu erhalten. Ein Beispiel wäre z.B. die Auswertung eines Signals, das zusammengesetzt ist aus einem harmonischen Sinus und einem bandbegrenzten Rauschen. Soll ausgewertet werden, wie groß die Amplitude des harmonischen Sinus ist, ist eine Skalierung für deterministische Signale vorzunehmen. Ist eine Beurteilung des stochastischen Grundrauschens angestrebt, ist eine Skalierung als PSD oder LSD vorzunehmen.

Skalierung von Spektren mit der Condition Monitoring Bibliothek

Verschiedene Skalierungsmöglichkeiten sind in der Condition Monitoring Bibliothek bereits implementiert und können über die Baustein-spezifische Struktur mit Initialisierungsparametern parametrisiert werden. Siehe E_CM_ScalingType [▶ 290] und Optionen der Spektrumsskalierung [▶ 390]. Ein Tutorial dazu finden Sie hier: Skalierung von Spektren [▶ 338].

Referenzieren

Einordnung der Skalierung

Während in der Messtechnik, der Schwingungsbeurteilung [▶ 35] nach ISO 10816-3 und dem Maschinenschutz der Vergleich absolut gemessener Größen sehr wichtig ist, ist eine absolute Kalibrierung im Bereich des Trend-basierten bzw. vergleichenden Condition Monitoring nicht notwendig.

Unabhängig von einer konkreten Maschine sind festgelegte Grenzwerte häufig für eine diagnostische Schadensfrüherkennung weniger geeignet. Da die Wahl des Messpunktes (die Stelle der Messung, die Ankopplung des Sensors, …) einen großen Einfluss auf die Dämpfungsfaktoren der Übertragungsstrecke hat, ist es für **Trendüberwachungen** im Verhältnis dazu viel wichtiger, den einmal gewählten Messpunkt sowie die Ankoppelbedingungen konsequent beizubehalten. Auch sind Signalanteile mit anfangs geringem Pegel häufig durchaus wichtig. Falls sie periodisch sind, treten sie bei einer Verwendung möglichst schmalbandiger, hochauflösender FFT-Spektren und geeigneter statistischer Funktionen besonders deutlich und früh hervor. Deswegen spielen für den Bereich Condition Monitoring Trendbeobachtungen über lange Zeiträume und relative, auf der Dezibel -Skala durchgeführte Vergleiche normalerweise eine viel wichtigere Rolle, als die Betrachtung einzelner absoluter Werte. Für die Sensoren folgt daraus, dass eine kostenaufwendige hochgenaue absolute Kalibrierung und glatter Frequenzgang in der Regel geringere Bedeutung haben als eine hohe Langzeitstabilität und ausreichend kleine Temperaturabhängigkeit, was nicht bedeutet, dass eine Kalibrierung vollständig vernachlässigt werden kann.

Skalierung anhand von Referenzsignalen

Die rechnerische Referenzierung (Skalierung anhand einer Referenz) von Messwerten kann sich oft wesentlich komplexer gestalten, als man auf den ersten Blick vermuten möchte. Sobald die Verarbeitung mehrere Schritte enthält, die von diversen Parametern nichtlinear abhängen, ist es in vielen Fällen einfacher und vor allem weniger fehleranfällig, die Skalierung mit Hilfe eines Kalibriergerätes vorzunehmen. Hierbei macht man sich zunutze, dass die Magnitudenwerte der berechneten Spektren immer linear zu den

Eingangswerten sind. Um das Signal korrekt zu skalieren, muss man also nur den zugehörigen linearen Faktor anhand eines bekannten Referenz-Eingangswertes ermitteln. Dies wird professionell so vorgenommen, dass man mit einem Kalibriergerät ein physisches Signal mit definierter Amplitude (bzw. definiertem RMS-Wert) erzeugt und durch Messung des Ausgangswertes den benötigten Korrekturfaktor als Quotient von Eingang und Ausgang bestimmt. Die Skalierung anhand von Referenzsignalen hat den großen Vorteil, dass physikalische Defekte, wie beispielsweise ein Schaden an einem Beschleunigungsaufnehmer, sowie Fehlkonfigurationen am Messsystem zuverlässig entdeckt werden können. Ihre Grenzen hat diese Methode, wenn bei Auswertungen eine Vielzahl von Parameterkombinationen untersucht werden sollen.

2.1.5 Statistische Auswertung

Condition Monitoring wird zur Überwachung von Grenzwerten eingesetzt. Wertüberschreitungen verursachen Meldungen und Warnungen. In der Praxis schwanken die Einzelwerte der FFT oft stark, so dass eine Mittelung oder andere statistische Auswertung notwendig ist. Eine Auswertung von einzelnen Werten würde dazu führen, dass z.B. ein hoher Wert eine Überschreitung der Grenzwerte verursacht.

Grundbegriffe

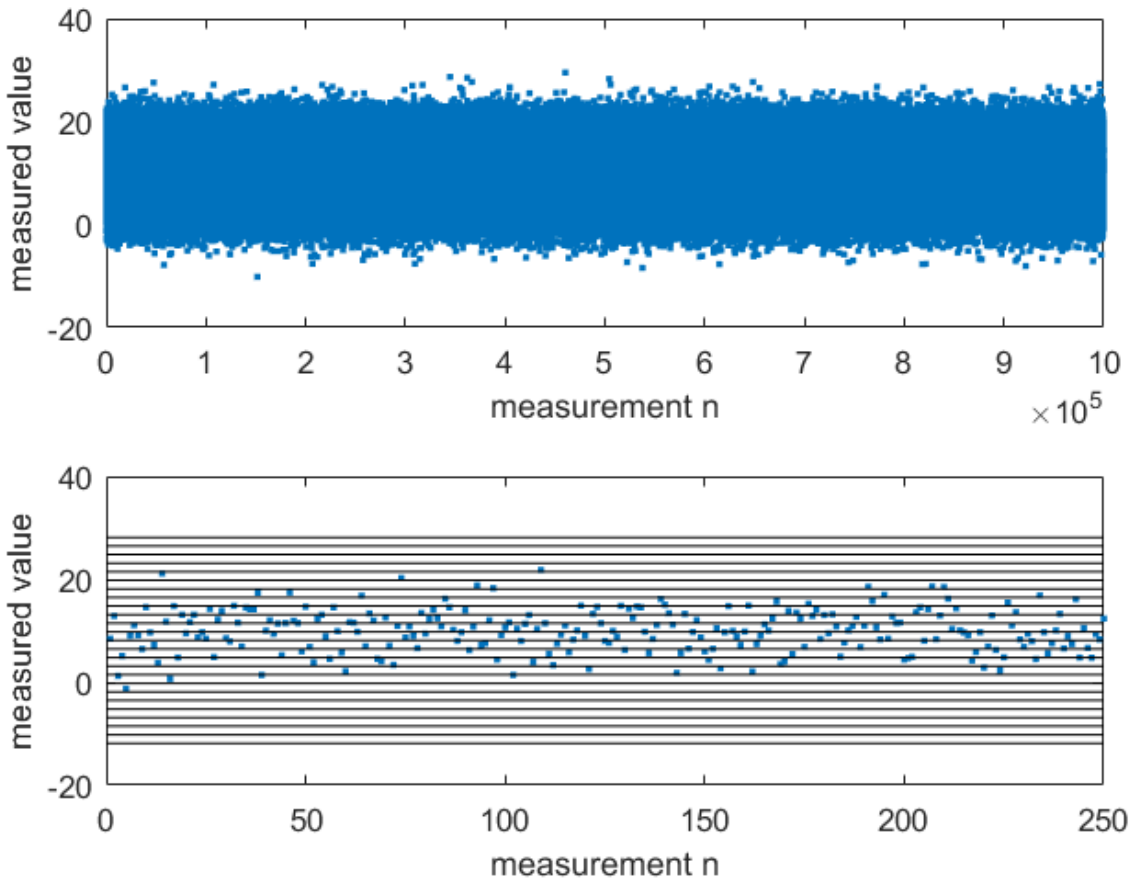
Wird eine Größe (z.B. Temperatur, Druck, Spannung, …) in einem realen Prozess gemessen, wird bei wiederholter Messung mit sehr hoher Wahrscheinlichkeit der zuvor bestimmte Messwert nicht mit dem der wiederholten Messung übereinstimmen. Da sich der Verlauf von regellos schwankenden Größen nicht deterministisch (über eine konkrete Gleichung) beschreiben lässt, werden zur Beschreibung dieser Signale statistische Kenngrößen verwendet. Dabei ist es unerheblich, dass häufig eine Überlagerung von deterministischem und stochastischem Signal vorliegt (z.B. eine Gleichspannung überlagert mit Messrauschen). In Summe ist das Ergebnis zufällig und somit ein stochastisches Signal.

Eine einzelne Messung einer zufällig schwankenden Größe ist ein *zufälliges Ereignis*. Jede einzelne Messung wird als *Realisierung* aus einem *Zufallsexperiment* bezeichnet. Werden *N* Stichproben aus dem Zufallsexperiment gezogen, beschreibt diese Menge an Realisierungen den *Stichprobenumfang*.

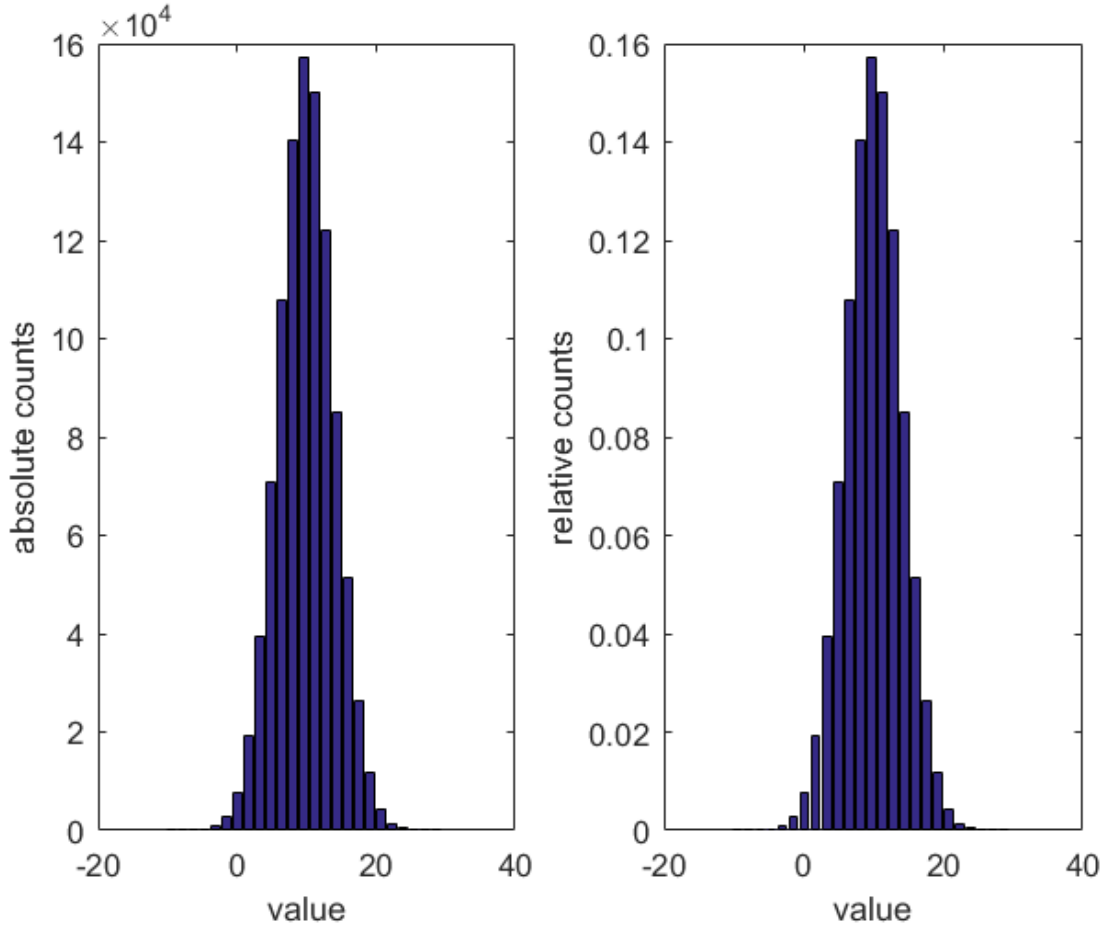
Histogramme

Für zufällige Ereignisse ist eine zentrale Eigenschaft die Wahrscheinlichkeit, dass die gemessene Größe einen bestimmten Wert annimmt. Dies wird über die absolute oder relative *Häufigkeitsverteilung* beschrieben, welche in einem *Histogramm* dargestellt wird.

Einfaches Beispiel: Angenommen eine Messgröße von 10 Volt sei überlagert mit einem Normalverteilten Rauschen (Mittelwert 0 V und Standardabweichung 4 V). Wird der Messvorgang dieser Größe nun 1 Mio. Mal wiederholt, ergibt sich die untenstehende Grafik (oberer Teil). Die 1 Mio. Realisierungen des Zufallsexperiments können zur besseren Übersicht in einem Histogramm dargestellt werden. Die absolute Häufigkeitsverteilung kann so generiert werden, dass der Wertebereich der aufgenommenen Messgröße in Klassen (Bins) unterteilt wird. Im oberen Teil der Grafik ist die Messgröße über jede Einzelmessung aufgetragen, im unteren Teil sind nur die ersten 250 Messungen sowie die Klassengrenzen für das Histogramm gezeigt.



Die absolute Häufigkeitsverteilung ergibt sich dann ganz einfach aus der Anzahl von Messwerten die innerhalb einer Klasse (Bin) liegen, siehe folgende Grafik, links. Parametrisiert wird die Verteilung entsprechend durch die Anzahl der berücksichtigten Klassen – je mehr Klassen, desto feiner die Verteilung. Aus der absoluten Häufigkeitsverteilung kann durch Referenzieren auf den Stichprobenumfang die relative Häufigkeitsverteilung berechnet werden, siehe folgende Grafik, rechts. Diese ist dann unabhängig von der Anzahl der Messungen und zeigt die Wahrscheinlichkeit an, mit der ein Wert gemessen wurde, z.B. wurden Werte in der Klasse um 10 V mit einer Wahrscheinlichkeit von 0,157 = 15,7% gemessen.



Anhand einer Häufigkeitsverteilung lässt sich ein experimentell untersuchter Prozess zunächst visuell recht einfach einschätzen. Es können Fragen erörtert werden wie:

- Wie stark streut die Messgröße?
- Streut die Messgröße um einen Wert (wie oben um 10 V), oder streut die Messgröße auch um weitere Werte?
- Wie ist die Verteilung der Werte? - Normalverteilung, Student-*t* -Verteilung, Chi-Quadrat-Verteilung?

Berechnung der absoluten Häufigkeitsverteilung in TwinCAT 3

Mit der Condition Monitoring Bibliothek lässt sich die absolute Häufigkeitsverteilung einfach über den Funktionsbaustein FB_CMA_HistArray [▶ 154] berechnen. Zur Parametrisierung des Bausteins werden lediglich der betrachtete Wertebereich und die Anzahl von Klassen benötigt. Eine grafische Darstellung ist mit dem Array-Bar-Chart im TwinCAT Scope View möglich. Dazu kann ein Sample [▶ 343] heruntergeladen werden.
In dem Beispiel Statistische Methoden [▶ 345] werden weitere Möglichkeiten der Condition Monitoring Bibliothek zur statistischen Auswertung von Daten exemplarisch vorgestellt.

Gewöhnliche und zentrale Momente

Aus einer gegebenen Stichprobe von zufällig schwankenden Größen kann ein Wert geschätzt werden, der dem wahren Wert möglichst nahe kommt – dieser wird als *Schätzwert* bezeichnet. Für diesen Vorgang kommen verschiedene *Schätzer* (z.B. der arithmetische Mittelwert) mit unterschiedlichen Eigenschaften in Frage. Neben der Berechnung eines Schätzwerts ist häufig auch eine Aussage über die *Unsicherheit* dieses Schätzwertes von Bedeutung, welcher i.d.R. über die *Stichprobenstreuung* (auch empirische Standardabweichung genannt) berechnet wird.

Zur Berechnung von statistischen Größen aus einer gegebenen Stichprobe eignen sich beispielsweise sehr gut die Momente: Mittelwert, Varianz, Schiefe, Kurtosis, … usw. Während der Mittelwert einen geeigneten Schätzwert der Stichprobe liefert, liefern die weiteren Momente Erkenntnisse über die Verteilung der Werte um diesen Schätzwert.

Erläuterung an einem Beispiel:

Die oben unter dem Punkt Histogramm beschriebene Stichprobe hat einen „wahren Wert“ von 10 V und wurde nachträglich verrauscht. Aus der gegebenen Stichprobe von 1 Mio. Realisierungen lässt sich der Mittelwert zu 9,9977 V berechnen und bildet den Schätzwert des wahren Werts. Die Varianz um diesen Mittelwert beträgt 16,01 V^2. Die Wurzel aus der Varianz entspricht der Standardabweichung und beträgt 4,0013 V. Ist die Verteilung der Messwerte, wie in diesem Fall, Normalverteilt, ist die Verteilung der Messwerte mit diesen beiden Momenten komplett beschrieben, d.h. die Schiefe und Kurtosis sind (theoretisch) Null. Die Schiefe beschreibt die Symmetrie der Verteilung um den Mittelwert, hingegen beschreibt die Kurtosis die Steilheit (Spitzenhaltigkeit) einer Verteilungsfunktion.

Beurteilung der Unsicherheit eines Schätzergebnisses:

Das Joint Committee for Guides in Metrology (JCGM) hat 1995 einen Leitfaden zur Angabe der Unsicherheit beim Messen publiziert. Das JCGM setzt sich aus zentralen Dachverbänden wie dem BIPM, IEC; IFCC, ISO, usw. zusammen, die diesem gemeinsamen Leitfaden entwickelt haben. Das Basispapier „Guide to the Expression of Uncertainty in Measurement” – kurz GUM – kann z.B. auf den Seiten des BIPM frei heruntergeladen werden. Ein kurzer Einblick in die zentrale Idee wird im Folgenden gegeben.

Aus einer gegebenen Stichprobe mit *N* Werten lässt sich, wie oben beschrieben, ein Schätzwert (Mittelwert = Stichprobenmittelwert) berechnen. Als Unsicherheitsmaß wird nun nicht die Stichprobenstreuung (Standardabweichung = Streuung der Stichprobe) genutzt, sondern die Streuung des Stichprobenmittelwerts berechnet. Dies ist natürlich sinnvoll, denn es soll die Unsicherheit des Schätzwerts beurteilt werden und nicht die der Stichprobe. Die Streuung des Stichprobenmittelwerts lässt sich einfach aus der Stichprobenstreuung berechnen, indem dieser Wert durch die Wurzel aus *N* dividiert wird. Bei genügend großer Stichprobenmenge kann die Streuung des Stichprobenmittelwerts mit dem Faktor 2 (ansonsten größer) multipliziert werden, um die erweiterte Unsicherheit zu berechnen. Der Mittelwert plus/minus dieser erweiterten Unsicherheit enthält dann mit 95%-iger Wahrscheinlichkeit den wahren Wert der Messgröße.

Entsprechend ist es möglich mit den Algorithmen der Condition Monitoring Bibliothek GUM-konforme Aussagen über die Messunsicherheit zu liefern.

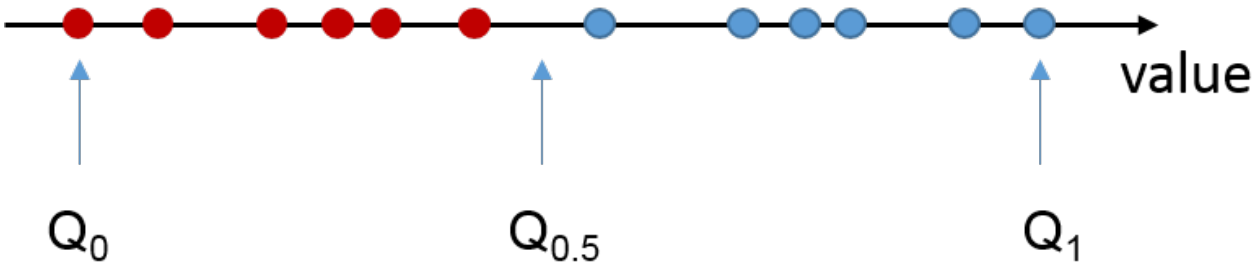
Berechnung der Momente in TwinCAT 3

Mit der Condition Monitoring Bibliothek lassen sich mit dem Funktionsbaustein FB_CMA_MomentCoefficients [▶ 183] die Momente erster bis vierter Ordnung (Mittelwert, Varianz, Schiefe, Kurtosis) einer Stichprobe berechnen. Der Baustein muss nur hinsichtlich der genutzten Stichprobenmenge parametrisiert werden.

Quantile

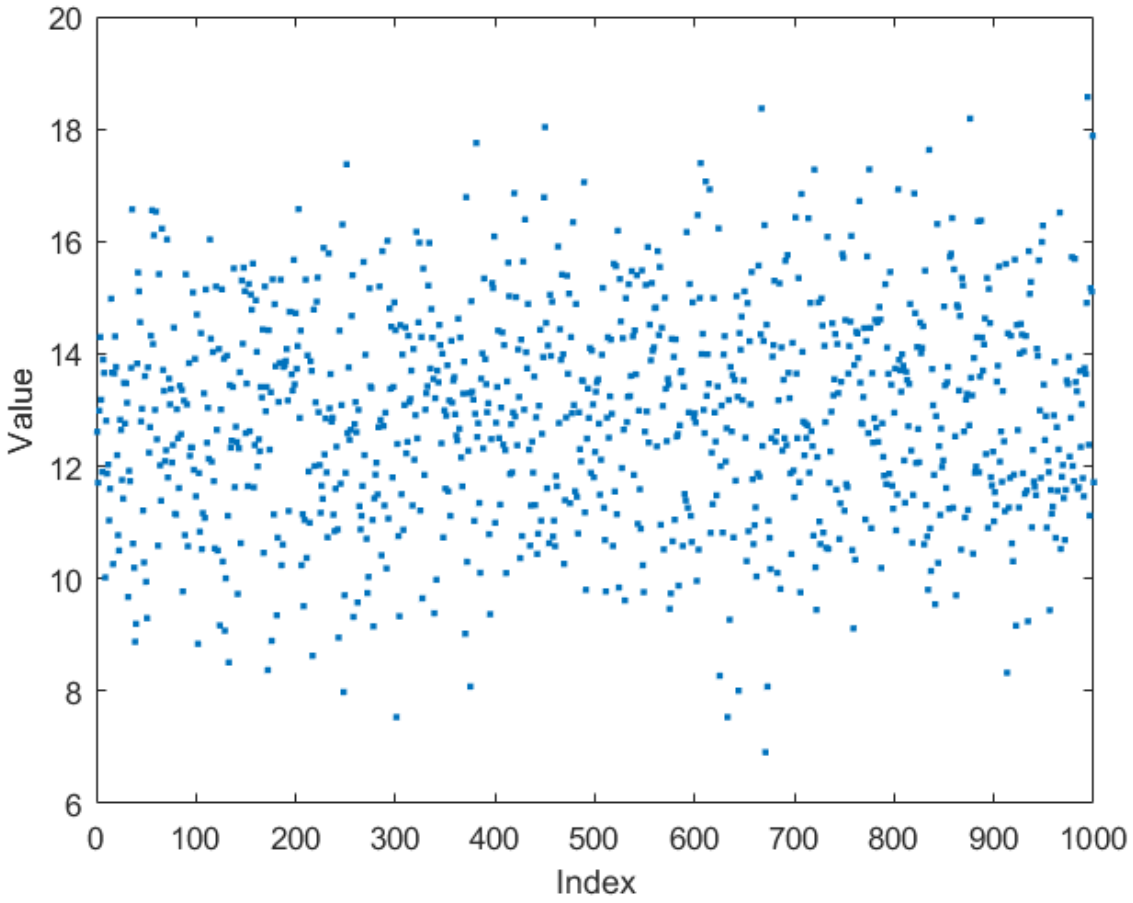
Das p-Quantil Q_p einer Zufallsvariablen *x* ist der Wert, für den $Q_p > x$ für den Anteil *p* aller Realisierungen von *x* ist. Etwas anschaulicher formuliert: Ist eine endliche Anzahl von Werten gegeben, so teilt das p-Quantil die Daten in zwei Bereiche ein. Das 50%-Quantil (auch Median) markiert z.B. den Wert unterhalb dessen mindestens 50% der in den Gesamtdaten vorkommenden Werte liegen. Dieser Wert ist nicht mit dem Mittelwert einer Stichprobe zu verwechseln.

Der Wert von *p* kann zwischen Null und Eins liegen. Wenn *p* in Prozent angegeben wird, handelt es sich um Perzentilen. So entspricht $Q_{0.5}$ genau dem Median, während $Q_{0.9}$ das 90-Prozent-Perzentil und Q_1 das Maximum einer beobachteten Werte-Folge darstellt.

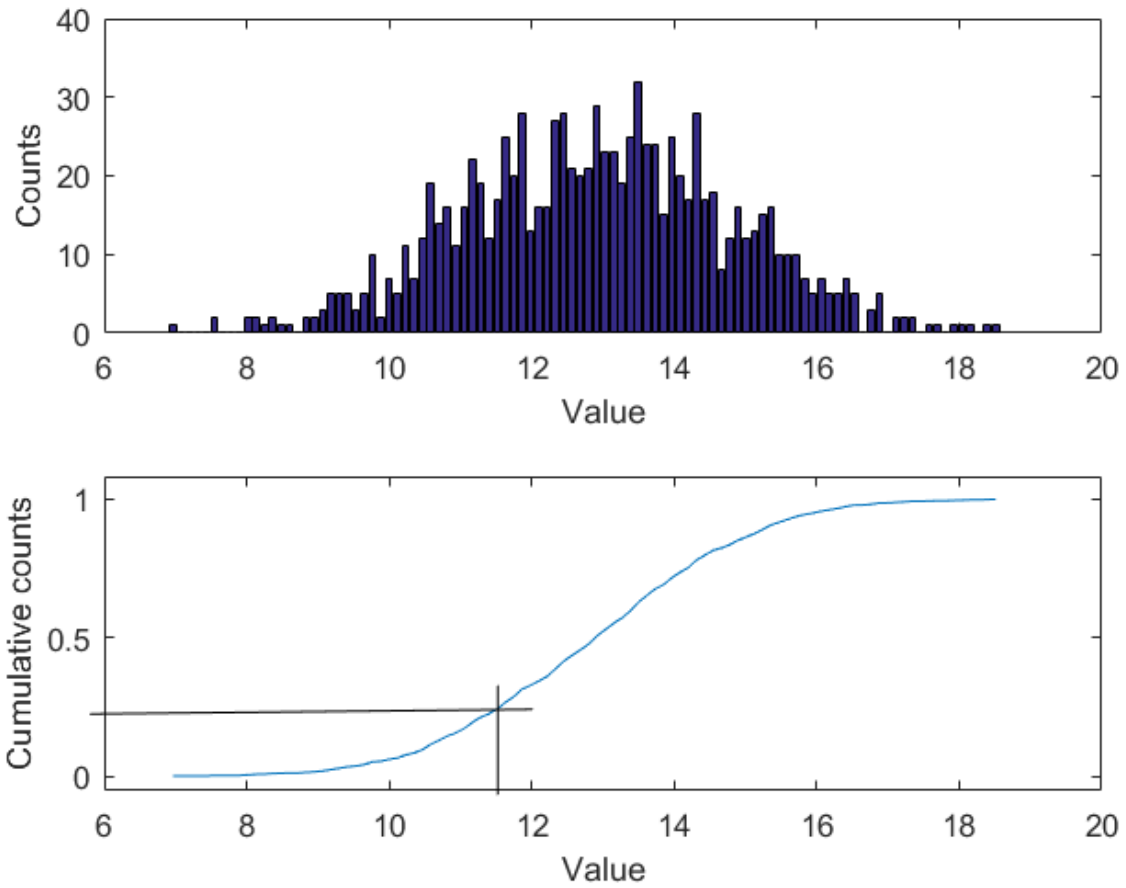


Je näher *p* an den Wert Eins heranrückt, desto stärker ist Q_p durch Ausreißer und extreme Einzelwerte bestimmt, je näher umgekehrt *p* an den Wert 0.5 heranrückt, desto mehr nähert sich Q_p dem Median an, welcher sehr robust gegenüber Ausreißern ist. Über den Wert von *p*, der in TwinCAT zur Laufzeit konfiguriert werden kann, lässt sich die Empfindlichkeit der Auswertung einer Stichprobe gegenüber von Einzelwerten dynamisch verändern.

Zur Veranschaulichung des Grundgedankens von Quantilen, zeigt folgende Grafik eine Folge von 1000 Werten, welche um einem Mittelwert von 13 streuen.



Aus der Wertfolge lässt sich das Histogramm berechnen, welches angibt wie häufig ein Wert in der betrachteten Folge (Stichprobe) vorkommt. Durch Integration der im Histogramm dargestellten absoluten Häufigkeit und Referenzieren auf die Gesamtzahl der Werte in der betrachteten Folge (hier 1000), kann die empirische Summenhäufigkeitsverteilung berechnet werden, anhand derer die Quantile leicht abzulesen sind. In diesem Fall liegt z.B. das 25%-Quantil bei 11.8, d.h. mindestens 25% der Einzelwerte der betrachteten Stichprobe von 1000 Werten liegen unterhalb von diesem Wert.



Der Bibliotheksbaustein zur Berechnung von Quantilen [▶ 206] arbeitet in zwei Teilschritten, die gemeinsam oder in separaten Teilschritten aufgerufen werden können. Im ersten Schritt werden Werte einem internen Histogramm hinzugefügt, dessen Parameter vorab konfiguriert werden können. Dieser Schritt benötigt sehr wenig Rechenaufwand. Im zweiten Schritt werden aus dem gespeicherten Histogramm die zuvor ausgewählten Quantile berechnet. Diese zweite Operation ist je nach Konfiguration deutlich rechenintensiver, da sie durch aufwendigere Operationen definiert ist, muss aber viel seltener ausgeführt werden.

Berechnung von Quantilen in TwinCAT 3

Zur Berechnung von Quantilen kann der Funktionsbaustein FB_CMA_Quantiles [▶ 206] genutzt werden. Dabei können mehrere Quantile mit nur einem Bausteinaufruf berechnet werden. Parametrisiert wird der Baustein wie der Histogramm-Baustein sowie zusätzlich die zu berechnenden Quantile und der zu nutzende Stichprobenumfang.

2.2 Anwendungskonzepte

Dieser Teil der Einführung gibt einen Überblick zu Grundmustern von Anwendungen und Lösungen zu Aufgabenstellungen im Condition Monitoring. Dabei wird noch nicht auf die Details der Programmierung und Schnittstellen eingegangen, sondern es werden einige zugrundeliegende Strategien und Konzepte erklärt. Am Ende jedes Konzepts wird ein Umsetzungsschema mit der Condition Monitoring Bibliothek aufgeführt, so dass ein Überblick über die Möglichkeiten der Bibliothek entsteht.

Sie erfahren hier:

- Wie funktioniert eine Schwingungsüberwachung nach ISO 10816-3?
- Wie funktioniert eine Schwellwertüberwachung im Frequenzbereich?
- Wie ist ein Condition Monitoring für Wälzlager aufgebaut?
- Wie ist ein Condition Monitoring für Getriebe aufgebaut?

- Wie realisiert man eine Schädigungsüberwachung?

2.2.1 Schwingungsbeurteilung

Einleitende Begriffsklärung

Die Schwingungsbeurteilung zielt auf einen zuverlässigen und sicheren Betrieb einer Maschine und somit auf die Bewertung des Maschinenbetriebszustands anhand von Vibrationsmessungen. Es wird demnach nicht auf eine lokale Diagnose/Analyse von Maschinenkomponenten eingegangen. Konzepte zur diagnostischen Zustandsüberwachung von Komponenten wie Wälzlager und Getriebe werden im Folgenden separat beschrieben.

Hinweise auf gängige Normen

Hinsichtlich der Beurteilung von Maschinenschwingungen existiert eine erhebliche Anzahl von Normen, die folgende Auflistung erhebt keinen Anspruch auf Vollständigkeit:

- DIN ISO 5348, Mechanische Schwingungen und Stöße – Mechanische Ankopplung von Beschleunigungsaufnehmern
- DIN ISO 10816, Mechanische Schwingungen – Bewertung der Schwingungen von Maschinen durch Messung an nicht-rotierenden Teilen (vorher VDI-Richtlinie 2056). Die Norm besteht aus mehreren Bestandteilen
  - DIN ISO 10816-3 bezieht sich auf industrielle Maschinen mit einer Nennleistung über 15 kW und Nenndrehzahlen zwischen 120 U/min und 15000 U/min bei Messung am Aufstellungsort.
  - DIN ISO 10816-7 bezieht sich auf Kreiselpumpen für den industriellen Einsatz
  - DIN ISO 10816-21 bezieht sich auf Windenergieanlagen mit horizontaler Drehachse und Getriebe
- DIN ISO 7919, Mechanische Schwingungen - Bewertung der Schwingungen von Maschinen durch Messungen an rotierenden Wellen. Die Norm besteht aus mehreren Teilen
  - DIN ISO 7919-3 bezieht sich auf Gekuppelte industrielle Maschinen
  - DIN ISO 7919-2 bezieht sich auf Stationäre Dampfturbinen und Generatoren über 50 MW mit Nenn-Betriebsdrehzahlen von 1500 min^{-1}, 1800 min^{-1}, 3000 min^{-1} und 3600 min^{-1}
- DIN ISO 20816-1, Mechanische Schwingungen – Messung und Bewertung der Schwingungen von Maschinen. Zusammenfassung von DIN ISO 7919-1 und DIN ISO 10816-1.

Beurteilung von Maschinenschwingungen in Anlehnung an DIN ISO 10816-3

Der Anwendungsbereich dieser Norm erstreckt sich von Dampfturbinen bis 50 MW über Elektromotoren bis hin zu Gebläsen und Lüftern. Aufgrund dieser Reichweite wird diese Norm im Folgenden näher erläutert. Ziel der Norm ist die Klassifizierung des Maschinenzustands in vier unterschiedliche Klassen anhand von Schwingungsdaten für Abnahmemessungen und die Betriebsüberwachung.

Als Bewertungskriterien sind nach der Norm der Effektivwert der Schwinggeschwindigkeit sowie der Effektivwert des Schwingungswegs geeignet. In der Regel reicht es aus die Schwinggeschwindigkeit zu messen. Nur bei Auftreten niedriger Frequenzkomponenten ist die zusätzliche Auswertung des Schwingungswegs empfohlen. Bei Erfassung und Auswertung beider Schwingungsgrößen gilt effektiv die schlechtere beider ermittelten Klassen.

Der zu erfassende Frequenzbereich der Schwingungen richtet sich nach der Drehzahl der Maschine:

- 10 Hz bis 1000 Hz für Drehzahlen über 600 min^{-1}
- 2 Hz bis 1000 Hz für Drehzahlen unter 600 min^{-1}

Geeignete Messorte kennzeichnen sich dadurch aus, dass sie die dynamischen Kräfte der Maschine möglichst unverfälscht wiedergeben, beispielsweise sind Orte an denen lokale Resonanzen auftreten nicht geeignet. Als geeignet werden in der Regel Lagerständer und Lagerdeckel bezeichnet, wobei Messung in zwei orthogonal zueinander stehenden Richtungen üblich sind.

Die Klassifizierung berücksichtigt des Weiteren die Maschinenunterbauten, gegliedert in starre und elastische Unterbauten. Liegt die tiefste Eigenfrequenz des Gesamtsystems aus Maschine und Unterbau mindestens 25 % über der wesentlichen Anregungsfrequenz (in der Regel die Drehfrequenz), kann der Unterbau als starr bezeichnet werden – ansonsten entsprechend als elastisch. Diese Beurteilung ist für jede Messrichtung (zwei orthogonale Richtungen, siehe oben) individuell durchzuführen.

Zur Bewertung werden in der DIN ISO 10816-3:2009 vier Zonen (A, B, C, D) mit den in der folgenden Tabelle aufgeführten Grenzwerten beschrieben.

Maschinengruppe		1		2	
Aufstellung		starr	elastisch	starr	elastisch
Effektivwert der Schwing-geschwindigkeit in mm/s	11,00 .. ∞	D	D	D	D
	7,10 .. 11,00	D	C	D	D
	4,50 .. 7,10	C	B	D	C
	3,50 .. 4,50	B	B	C	B
	2,80 .. 3,50	B	A	C	B
	2,30 .. 2,80	B	A	B	B
	1,40 .. 2,30	A	A	B	A
	0,00 .. 1,40	A	A	A	A
Maschinengruppe		1		2	
Aufstellung		starr	elastisch	starr	elastisch
Effektivwert des Schwingwegs in µm	140 .. ∞	D	D	D	D
	113 .. 140	D	C	D	D
	90 .. 113	D	C	D	C
	71 .. 90	C	B	D	C
	57 .. 71	C	B	C	B
	45 .. 57	B	B	C	B
	37 .. 45	B	A	B	B
	29 .. 37	B	A	B	A
	22 .. 29	A	A	B	A
	0 .. 22	A	A	A	A

Zone A	Die Schwingungen neu in Betrieb gesetzter Maschinen liegen gewöhnlich in dieser Zone.
Zone B	Maschinen, deren Schwingungen in dieser Zone liegen, werden üblicherweise als geeignet angesehen, ohne Einschränkungen im Dauerbetrieb zu laufen.
Zone C	Maschinen, deren Schwingungen in dieser Zone liegen, werden üblicherweise nicht als geeignet angesehen, ständig im Dauerbetrieb zu laufen. Im Allgemeinen darf die Maschine aber für eine begrenzte Zeit in diesem Zustand betrieben werden, wenn sich eine günstige Gelegenheit für Abhilfemaßnahmen ergibt.
Zone D	Schwingungswerte innerhalb dieser Zone werden üblicherweise als so gefährlich angesehen, dass Schäden an der Maschine entstehen können.
Maschinengruppe 1	Große Maschinen mit einer Nennleistung größer 300 kW bis 50 MW sowie elektrische Maschinen mit einer Achshöhe ≥ 315 mm
Maschinengruppe 2	Mittelgroße Maschinen mit einer Nennleistung größer 15 kW bis 300 kW sowie elektrische Maschinen mit einer Achshöhe zwischen 160 mm und 315 mm

Die Umsetzung der beschriebenen Schwingungsbeurteilung nach ISO 10816-3 ist in drei unterschiedlichen Beispielen umgesetzt. Siehe hierzu:

- Schwingungsbeurteilung nach ISO 10816-3 [▶ 346]
- Schwingungsbeurteilung nach ISO 10816-3 (kompakt) [▶ 349]
- Schwingungsbeurteilung nach ISO 10816-3 (erweitert) [▶ 351]

2.2.2 Frequenzanalyse

Motivation

Eine der wichtigsten Methoden bei der diagnostischen/analytischen Maschinenüberwachung ist die Aufnahme von Schwingungen mit Beschleunigungsaufnehmern und eine darauf aufbauende Frequenzanalyse. Dies liegt daran, dass Maschinen aus Metall und damit aus elastisch federnden Strukturen bestehen, die nahezu immer periodischen Kräften ausgesetzt sind. Diese führen zu Schwingungen, in denen sich sowohl die Frequenzen und anregenden Kräfte als auch die charakteristischen Frequenzen der betreffenden Strukturen widerspiegeln. Die Messung der Schwingungen ermöglicht Rückschlüsse auf Strukturen und Kräfte in der Maschine. Schäden und Strukturveränderungen von Maschinenelementen, wie z. B. einem Lager, führen zu Veränderungen des Schwingungsbildes.

Die Schwingungen breiten sich in Form von Schallwellen (Körperschall) in den Maschinenbauteilen aus. Da Maschinen aus einer Vielzahl von Teilen bestehen, welche einerseits Schwingungen anderer Bauteile elastisch übertragen und andererseits selber oszillieren, kommt es zu Filterungen und Überlagerungen der einzelnen Schwingungskomponenten. Ein Schwingungssignal besteht entsprechend aus mehreren Signalanteilen, welche sich mit unterschiedlichen Zeitverzögerungen und vom zurückgelegten Weg abhängigen Dämpfungen zum Gesamtsignal addieren. Einzelne gesuchte Schwingungskomponenten sind deswegen im zeitlichen Verlauf des Gesamtsignals nicht mehr ohne weiteres erkennbar. Die Leistung der Frequenzanalyse besteht nun darin, die Vielzahl der vorhandenen linear überlagerten Schwingungen in Frequenzkomponenten aufzutrennen. Diese Frequenzkomponenten können dann wesentlich leichter einem besonderen Maschinenzustand, einem Bauteil oder einem Ablauf zugeordnet werden.

Das Konzept zur frequenzselektiven Überwachung von Komponenten trennt sich auf in:

- Berechnung des Spektrums
- Statistische Beurteilung des Ergebnisses
- Schwellwertüberwachung

Praktische Elemente der Frequenzanalyse

Die wesentlichen Aspekte der Fourier-Analyse sind bereits im Abschnitt Fourier-Analyse [▶ 11] behandelt worden. An dieser Stelle werden nochmal die wichtigsten praktischen Aspekte aufgegriffen.

Bei der Auslegung der Parameter des Funktionsbausteins zur Fourier-Analyse (z. B. FB_CMA_MagnitudeSpectrum [▶ 172] oder FB_CMA_PowerSpectrum [▶ 202]) sind folgende Fragen von zentraler Bedeutung.

- Wie hoch ist die höchste zu analysierende Frequenz?
  Entsprechend ist die Abtastfrequenz über den Oversampling-Faktor der Klemme und die zugehörige Task Cycle Time einzustellen. Ebenfalls ist auf die Einstellung eines Anti-Aliasing-Filters zu achten. Siehe Abschnitt Abtasttheorem [▶ 13].
- Wie sind die Anforderungen an die Frequenzauflösung?
  Entsprechend lang ist die Messzeit (Größe des Eingangs-Arrays) zu wählen, siehe Frequenzauflösung [▶ 13]. Die Verschlechterung der Frequenzauflösung durch die Verwendung einer Fensterfunktion ist dabei ebenfalls zu berücksichtigen, siehe Fensterfunktionen [▶ 20].
- Die FFT-Länge muss größer sein als die Größe des Eingangs-Arrays und muss eine Zweierpotenz sein. Die überschüssigen Elemente werden als Nullen aufgefüllt, siehe Zero Padding bzw. Frequenzauflösung [▶ 13].
- Eine geeignete Skalierung des Spektrums ist auszuwählen, siehe Skalierung von Spektren [▶ 26].

> Obige Punkte beziehen sich insbesondere auf die Verwendung des FB_CMA_MagnitudeSpectrum und FB_CMA_PowerSpectrum. Bei der Verwendung von speziellen Varianten der Spektralberechnung lassen sich einige Punkte optimieren. Siehe dazu Abschnitt Übersicht Berechnung von Spektralwerten [▶ 16].

Statistische Beurteilung

Das Fourier-Spektrum ist sehr empfindlich gegenüber Rauschen und Störungen im Signal. Deswegen ist die Fourier-transformierte realer verrauschter Signale für eine direkte Analyse oder Begutachtung normalerweise nicht gut geeignet. Um dies auszugleichen, wird in der Regel das Betragsspektrum gemittelt oder durch Quantile ausgewertet, siehe Statistische Auswertung [▶ 29]. Diese Herangehensweise setzt die zeitliche Stabilität oder zyklische Wiederholung des zu analysierenden Signals voraus. Die so ermittelten Größen sind wesentlich robuster gegenüber Störungen und auch visuell viel besser zu beurteilen. Im Folgenden wird eine Auswertung basierend auf dem Mittelwert von mehreren Spektren exemplarisch betrachtet.

Statistische Bewertung des Betragsspektrums

Es ist sinnvoll, mehrere Betragsspektren zu bilden und diese statistisch auszuwerten, z.B. über Mittelwertbildung oder Quantile. Dadurch verringert sich die Unsicherheit der ermittelten Werte und eine Schwellwertanalyse wird zuverlässiger.

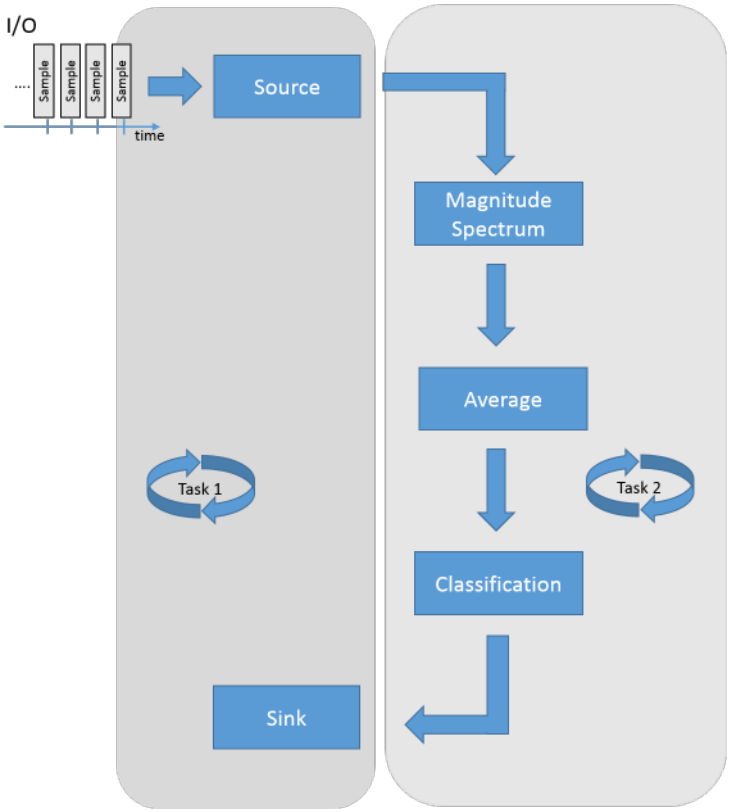
Eine alternative Methode ist die Mittelung der errechneten Fourier-Koeffizienten über die Frequenz, also die Mittelung benachbarter Frequenz-Bins. Dazu ist verständlicherweise die FFT mit höherer Frequenzauflösung zu berechnen als bei der oben beschriebenen Methode der sukzessiven Mittelung von Spektren über die Zeit. Die Mittelung benachbarter Frequenz-Bins ist weitgehend gleichwertig zur Mittelung von Spektren über die Zeit, jedoch Rechenaufwändiger.

Schwellwertüberwachung

Der letzte Schritt des hier erläuterten Konzepts besteht in einer automatischen Schwellwertüberwachung. Dabei werden für jeden Frequenzkanal Schwellwerte definiert, die mehreren Kategorien unterschiedlicher Priorität zugeordnet sind, wie z.B. "Normalbetrieb", "Warnung" und "Alarm". Diese Schwellwerte können basierend auf Erfahrungen gesetzt und auch im Betrieb angepasst werden.

Verarbeitungskonzept

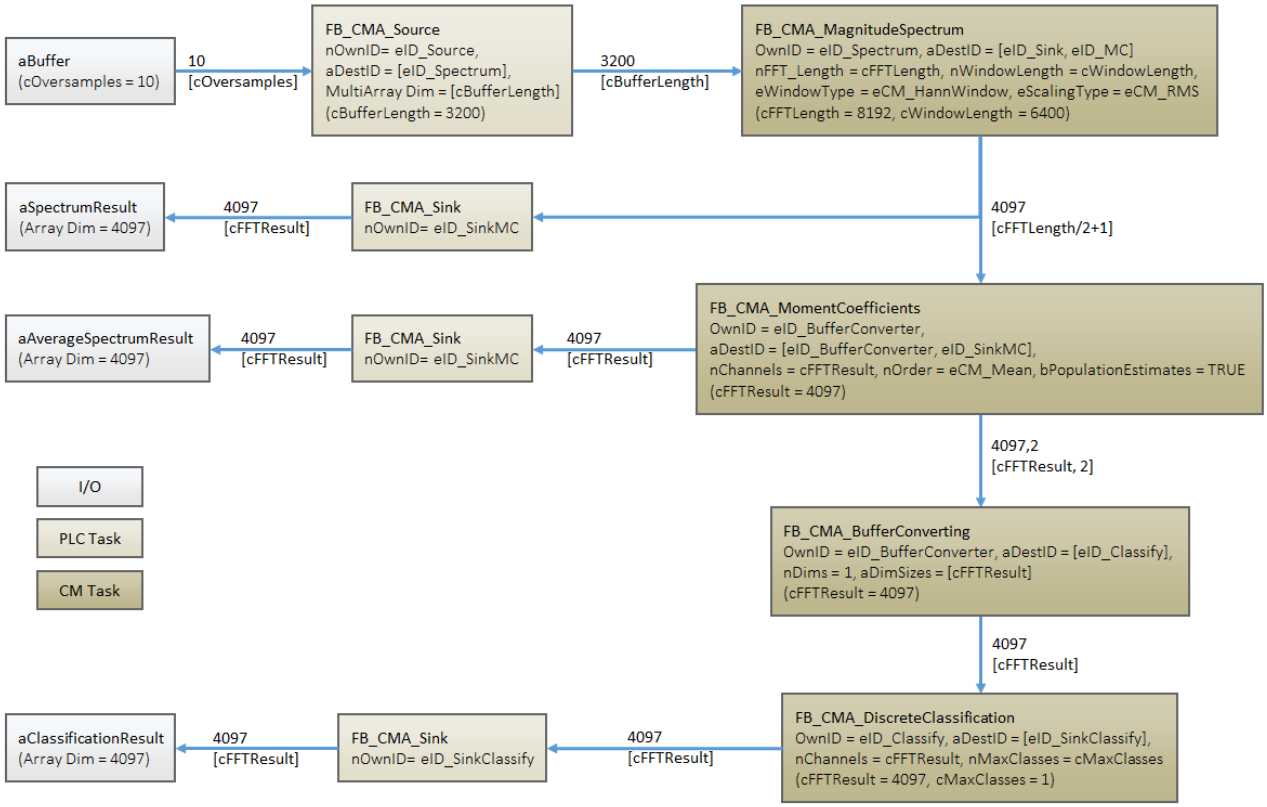
Das oben abstrakt beschriebene Konzept kann mit der TwinCAT Condition Monitoring Bibliothek durch Parametrierung der zur Verfügung gestellten Bausteine einfach umgesetzt werden. Im Folgenden dazu eine Beispielkonfiguration.



Es soll eine Schwellwertüberwachung auf gemittelten Betragsspektren umgesetzt werden. Dazu werden folgende Komponenten der Condition Monitoring Bibliothek mit den beschriebenen Funktionen verwendet:

- FB_CMA_Source
  - Puffern der Eingangsdaten
- FB_CMA_MagnitudeSpectrum
  - Reihung der Eingangspuffer in überlappende Abschnitte
  - Fenstern der Berechnungsabschnitte
  - Berechnung der Fourier-Transformation
  - Berechnung des Absolutbetrags der Fourier-Koeffizienten
  - Skalierung des Ergebnisses (im folgenden RMS)
- FB_CMA_MomentCoefficients
  - Bildung des arithmetischen Mittelwerts
- FB_CMA_BufferConverting
  - Anpassung der Buffer-Dimensionen zur Übergabe zwischen FB_CMA_MomentCoefficients und FB_CMA_DiscreteClassification
- FB_CMA_DiscreteClassification
  - Überwachung jeder berechneten Frequenz auf eine Schwellwertüberschreitung

In der Quellcode-näheren Darstellung des obigen Schaubildes stellt sich eine mögliche Umsetzung wie folgt dar:



Das Beispiel-Projekt zu dem hier dargestellten Konzept kann hier heruntergeladen werden: download [▶ 359]**.**

Parametrierung der Berechnung des Betragsspektrums

Die Cycle Time und der Oversampling Faktor sind so eingestellt, dass sich eine Abtastrate von 10 kHz ergibt. Für die Parametrierung des Bausteins MagnitudeSpectrum sowie der Source-Bausteins sind folgende Einstellungen im Beispiel angesetzt

```
// constant for input
cOversamples  : UDINT     := 10;    // number of oversamples
cFSample      : UDINT     := 10000; // 1ms task with 10 oversamples = 10kHz

// constants for FFT (Magnitude Spectrum)
cBufferLength : UDINT  := 3200;             // buffer size
cWindowLength : UDINT  := 2*cBufferLength;  // 50% overlap
cFFTLength    : UDINT  := 8192;             // length of FFT for mag. spectrum, power of 2
cFFTResult    : UDINT  := cFFTLength/2+1;   // result of mag. spectrum
```

Die numerische Frequenzauflösung ergibt sich demnach zu 10 kHz / 8192 = 1,22 Hz. Wie im Kontext zum Zero Padding bzw. Frequenzauflösung [▶ 13] beschrieben, entspricht dies jedoch nicht der Frequenzauflösung, die die Unterscheidung von zwei dicht benachbarten Frequenzen ermöglicht. Diese ist im hier dargestellten Fall 10 kHz / (2*3200) * 1,5 = 2,34 Hz, wobei 2*3200 die Länge des zur Berechnung der FFT genutzten Signalstücks entspricht (Messzeit in Abtastwerten). Der Erweiterungsfaktor 1.5 ist durch die Wahl des Hanning-Fensters definiert (Fensterung im MagnitudeSpectrum-Baustein). Die FFT-Länge ist mit 8192 die kleinste Zahl größer als 2*3200 welche eine Zweierpotenz darstellt. Die Länge des Ergebnis-Arrays der Berechnung des Betragsspektrums ist mit 4097 durch die Symmetrieeigenschaft der FFT fest definiert.

Mittelung der Betragsspektren

Das Ergebnis des MagnitudeSpectrum-Bausteins wird übergeben an den FB_CMA_MomentCoefficients, welcher so konfiguriert wird, dass er als Ergebnis den Mittelwert (erstes zentrales Moment) liefert. Standardmäßig liefert der Baustein zusätzlich die Größe der Stichprobenmenge, welche zur Berechnung des zentralen Moments genutzt wurde. Aus diesem Grund erhöht sich die Dimension der Ergebnis-Arrays auf zwei Dimensionen. Im hier betrachteten Beispiel wird die CallEx() Methode des Bausteins genutzt, um jeweils 25 Betragsspektren zu Mitteln und dann den Baustein wieder zurückzusetzen. Da bei dieser Parametrierung des Bausteins die Stichprobenmenge immer 25 ist, wird diese Information nicht weiter benötigt. Der zugehörige Sink-Baustein wird deshalb so parametriert, dass er aus dem Ergebnis des Bausteins nur die Mittelwerte in den Kontext der PLC Task kopiert. Außerdem wird zwischen dem Klassifikationsbaustein und dem MomentCoefficients-Baustein eine Buffer-Konvertierung (FB_CMA_BufferConverting) durchgeführt, welche ebenfalls die Spalte mit der Information der Stichprobenmenge entfallen lässt.

Klassifikation

Der Baustein FB_CMA_DiscreteClassification wird im vorliegenden Fall als einfacher Schwellwert-Klassifikator genutzt. Entsprechend werden nur 2 Klassen definiert (nMaxClasses = 1). Die Konfiguration der Schwellwerte, welche für jede diskrete Frequenz einzeln gesetzt wird, erfolgt zur Laufzeit. Im bereitgestellten Beispiel wird der Schwellwert für die Array-Indizes 30 bis 50 (entspricht ca. 36 Hz bis 61 Hz) auf 6 V_{RMS} und für die restlichen Frequenzen auf 2 V_{RMS} gesetzt. Bei Unterschreitung des Schwellwerts wird eine -1 als Ergebnis für die betroffene Frequenz zurückgegeben. Bei Überschreitung wird eine 0 ausgegeben.

Weiteres zum Beispielcode

Im Projekt eingebunden ist ein Measurement Projekt, welches ein Scope Array Projekt mit drei Achsen beinhaltet. Die obere Darstellung zeigt das Ergebnis des FB_CMA_MagnitudeSpectrum, also das Betragsspektrum des Eingangssignals. Das Eingangssignal wird durch einen Funktionsgenerator erzeugt und stellt einen verrauschten Sinus mit 50 Hz und einer Amplitude von 25 V. Entsprechend ist das Ergebnis des nicht-gemittelten Betragsspektrums mit der Zeit veränderlich (unsicher). Durch die Mittelung stabilisiert sich das Ergebnis merklich. Das gemittelte Betragsspektrum wird in der mittleren Darstellung des Scope Array Projects angezeigt. Die untere Darstellung zeigt das Ergebnis Klassifikation, also für jede Frequenz eine -1 für Unterschreitung und eine 0 für Überschreitung des jeweils definierten Schwellwerts.

Weiteres Beispiel zum Condition Monitoring mit Frequenzanalyse

Im Abschnitt Beispiele sind mehrere Code-Beispiele zu finden. Unter anderem finden Sie unter Condition Monitoring mit Frequenzanalyse [▶ 353] ein ähnliches Beispiel wie hier in diesem Abschnitt beschrieben wurde. Dies soll die Flexibilität Ihrer individuellen Lösung unterstreichen, die Sie mit der Condition Monitoing Bibliothek erstellen können.

2.2.3 Wälzlagerüberwachung

Motivation

Wälzlager zählen zu den häufigsten und am stärksten belasteten Maschinenelementen. In vielen Fällen können sie eine für den Betrieb einer Anlage kritische Bedeutung haben. Während bei großen Lagern schon die Ausfallzeiten für eine Ersatzbeschaffung hohe Kosten verursachen können, können auch Ausfälle von kleinen Lagern Kosten verursachen, welche weit über den Kosten des Ersatzteils liegen.

Schadensursachen

Die möglichen Ursachen für Ausfälle von Wälzlagern sind vielfältig:

- Die "natürliche" Ursache für Ausfälle von Wälzlagern ist Materialermüdung durch die hohen Spannungen, die beim Betrieb an den Kontaktflächen der Wälzkörper auftreten. Diese führen nach einer gewissen Zeit zu Rissen im Material und zu Ausbrüchen an der Lauffläche. Es entstehen kleine Defekte, die zunächst sehr langsam wachsen und sich gegen Ende der Lebensdauer mit zunehmender Geschwindigkeit vergrößern. Die Mechanismen der Materialermüdung sind theoretisch gut verstanden und lassen sich statistisch beschreiben; sie sind Bestandteil der normalen Abnutzung. Bei der Auslegung eines normalen Lagers werden die Dimensionen so gewählt, dass die Wahrscheinlichkeit eines gravierenden Schadens innerhalb der Lebensdauer der Maschine gering ist. Von korrekt dimensionierten und gewarteten Lagern kann also unter normalen Umständen eine sehr lange Lebensdauer erwartet werden. Die tatsächlich erreichte Lebensdauer ist oft wesentlich geringer, jedoch nicht genau vorhersagbar und kann beträchtlich schwanken, was an den folgenden Ursachen liegt.
- Die Beanspruchung von Wälzkörpern und Laufflächen wird erheblich vergrößert durch fehlerhafte Schmierung, da das Schmiermittel einen Teil der Spannung verteilt und auch ein Heißlaufen des Lagers verhindert.
- Eine weitere Ursache von Schäden sind Verschmutzungen, beispielsweise aufgrund fehlerhafter Dichtungen, oder durch Metallspäne. Auch das Eindringen von Wasser kann zu einem Versagen der Schmierung führen, da schon geringe Mengen Wasser Schmiermittel untauglich machen.
- Eine nicht unwichtige Fehlerquelle sind weiterhin Ungenauigkeiten bei der Ausrichtung oder schädigende Belastungen beim Einbau.
- Zu hohe Belastungen führen zu plastischen Verformungen der Lauffläche (Brinelling). Ähnliches kann auch durch Vibrationen bei Stillstand des Lagers entstehen, die nicht durch einen Schmierfilm gemildert werden (False Brinelling).
- In elektrischen Maschinen kann Stromfluss die Laufflächen zerstören.
- Korrosion kann die initiale Schädigung der Oberfläche bewirken.

Allen diesen Ursachen ist gemeinsam, dass Beschädigungen der Laufflächen des Wälzlagers schon früh im Schadensverlauf detektierbar sind. Aus der Tatsache, dass Ausfälle bei Lagern in der großen Mehrzahl der Fälle nicht durch Materialermüdung verursacht werden, folgt, dass eine frühe Erkennung von Schäden und die Analyse und Rückverfolgung auf die primären Ursachen (Root Cause Failure Analysis (RCFA)) es – neben der Reduktion von Ausfallkosten – mittelfristig möglich macht, viele Schäden vorbeugend zu vermeiden und die praktische Lebensdauer von Lagern erheblich zu erhöhen.

Schadensfolgen

Nach der ersten Beschädigung der Laufflächen kommt es infolge von anwachsenden Spannungen zu einer Ausbreitung von Defekten. Neben der Lauffläche können auch andere Komponenten, wie der Käfig der Wälzkörper betroffen sein. Vibrationen zeigen nicht unbedingt die ersten Stufen des Schadensprozesses an, da sie meist nicht die Ursache von Schäden darstellen, sondern ein Symptom. Jedoch führen alle Schadensverläufe früher oder später zu Defekten an den Kontaktstellen, die sich in zunehmenden Vibrationen äußern.

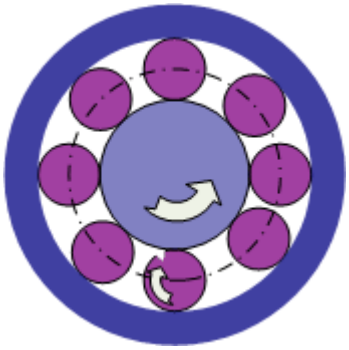
Überwachungsstrategien

Da eine direkte Erkennung der ersten Ursachen möglicherweise schwierig ist, liegt der Fokus auf einer Früherkennung der Folgeschäden in der Lauffläche des Lagers. Je früher diese bemerkt und untersucht werden, desto größer sind die Chancen, die Spuren der initialen Schäden zu finden und anhand dieser die Ursachen zu beheben – eine Strategie, die langfristig oft zu nachhaltigen Einsparungen führt. Weiterhin

erleichtert eine Früherkennung eine Planung der Wartung, was vor allem für Anlagenbetreiber einen Vorteil darstellt. Eine zweite Strategie ist eine Identifikation der betroffenen Elemente durch Analyse der Schwingungssignale.

Zum Verständnis der folgenden Möglichkeiten zur Signalanalyse erfolgt zunächst ein kurzer phänomenologischer Einblick in die Entstehung von Vibrationen bei defekten Wälzlagern.

Schematischer Aufbau eines Wälzlagers im Querschnitt:



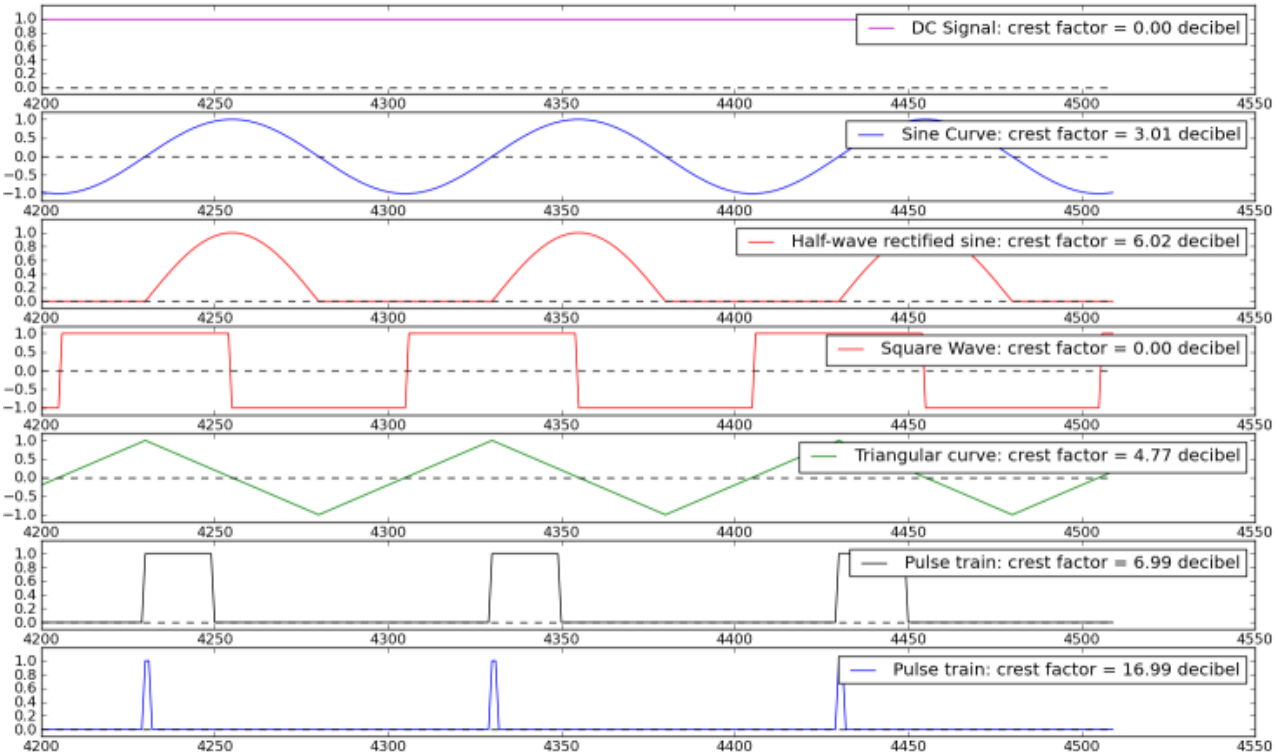
Die kritischen Teile eines Wälzlagers sind diejenigen Flächen, welche aufeinander laufen. Dies sind die Wälzkörperoberfläche, sowie die Lauffläche des Innenrings und die Lauffläche des Außenrings. Bei lokal beschädigter Lauffläche kommt es beim Überrollen der Beschädigung zu einem Stoßimpuls, welcher über einen Beschleunigungssensor aufgenommen werden kann. Je größer die Beschädigung, desto kräftiger fällt der Stoßimpuls aus.

Bewertung der Vibration im Zeitbereich

Eine einfache Methode zur Bewertung des Zustands eines Wälzlagers ist die Bewertung des Vibrationssignals hinsichtlich seiner Impulshaltigkeit. Gängige Verfahren dazu sind die Berechnung des Crestfaktors sowie des Kurtosiswerts.

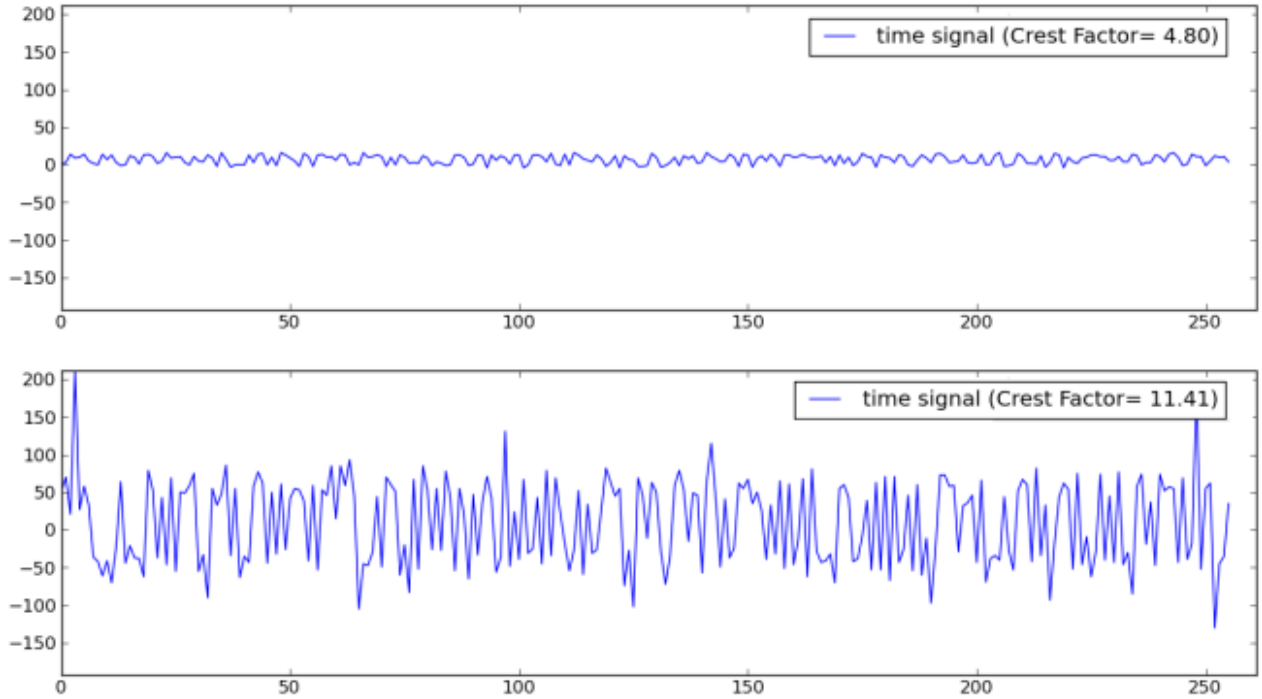
Der Crestfaktor

Der Crestfaktor ist definiert als das Verhältnis vom Betrag der maximalen Amplitude zum Effektivwert des Signals. Er wird angegeben in Dezibel und ist eine Zahl größer gleich Null. Der Crestfaktor bestimmt somit das Verhältnis zwischen Maximalamplitude und der effektiven mittleren gemessenen Schwingungsamplitude. Stoßimpulse aus einem beginnenden Schaden führen zu einer Zunahme des Crestfaktors. Folgende Grafik zeigt die Erhöhung des Crestfaktors mit steigender Impulshaltigkeit der Signale.



Die unterste Kurve zeigt den typischen starken Anstieg des Crestfaktors beim Auftreten von spitzen Stoßimpulsen. Eine Zunahme des Crestfaktors zeigt normalerweise an, dass sehr wahrscheinlich eine Beschädigung vorliegt. Dies macht diese Größe zu einem geeigneten Werkzeug für die Früherkennung von Schäden und für Trendanalysen.

Signale aus einem Wälzlager können wie folgt interpretiert werden.



Die obige Abbildung zeigt zwei Schwingungssignale aus unterschiedlich abgenutzten Lagern, jeweils mit dem zugehörigem Crestfaktor. Die aus dem Schaden resultierenden Spitzen sind im unteren Signalverlauf deutlich zu erkennen. Während das unbeschädigte Lager einen Crestfaktor von 4,8 dB aufweist, beträgt dieser beim beschädigten Teil 11,4 dB, und zeigt so das Vorliegen einer Beschädigung klar an.

Der Crestfaktor hat den Vorteil, dass er sehr effizient zu berechnen und leicht interpretierbar ist. Zudem lässt er sich sehr einfach in einem Diagramm über der Zeit darstellen. Für eine korrekte Anwendung ist es wichtig, auch die prinzipiellen Grenzen dieser Auswertung zu verstehen:

- Der Crestfaktor ist stark durch das Signalmaximum bestimmt und daher im statistischen Sinne keine robuste Größe.
- Der Crestfaktor steigt bei zunehmenden lokalen Schädigungen an. Die Spitzenwerte der Stoßimpulse werden aber ab einem bestimmten Schadensfortschritt nicht mehr signifikant ansteigen, jedoch nimmt die Anzahl der lokalen Schädigungen zu. Das führt dazu, dass die Signalmaxima nicht zunehmen, der Effektivwert des Signals jedoch ansteigt. Dadurch verringert sich der Crestfaktor bei stark beschädigten Lagern wieder.

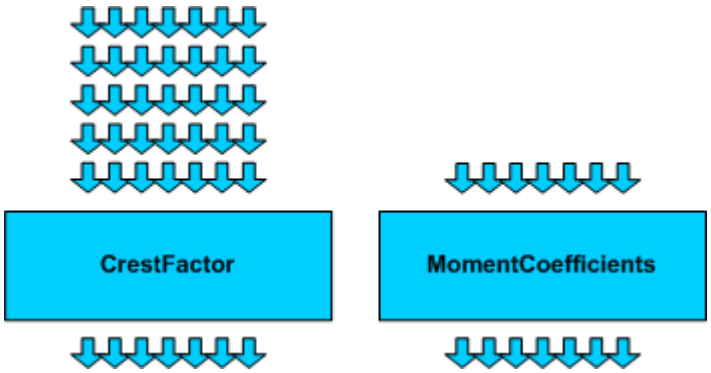
Aus diesem Grund ist es ratsam bei Verwendung des Crestfaktors möglichst kontinuierlich zu messen und die Ergebnisse hinsichtlich des Trends auszuwerten.

Der Kurtosiswert

In manchen Fällen ist die begrenzte statistische Robustheit des Crestfaktors problematisch. Robuster, aber etwas rechenaufwändiger, ist der Kurtosiswert (auch Kurtosis, Wölbung, viertes zentrales Moment). Die Kurtosis gehört, wie Mittelwert und Varianz, zu den sogenannten Momentenkoeffizienten, mit denen sich Größen statistisch beschreiben lassen. Die Kurtosis beschreibt das Verhältnis zwischen den extremen (weitab vom Mittelwert liegenden) Werten einer Verteilung und der mittleren Schwankungsbreite. Da vereinzelte Ausreißer in einer Messreihe aber nicht bestimmend für das Ergebnis sind, ist die Kurtosis statistisch wesentlich robuster als der Crestfaktor.

Die praktische Benutzung der Kurtosis erfolgt weitgehend analog zum Crestfactor. Die Berechnung der Kurtosis (bzw. des Exzess) und weiterer gängiger statistischer Momentenkoeffizienten erfolgt in der TwinCAT 3 Condition Monitoring Library mit dem Baustein MomentCoefficients [▶ 183].

Verarbeitungskonzepte



Die obige Abbildung benennt die verfügbaren Bausteine zur Berechnung von Crestfaktor und Kurtosis. Der Baustein CrestFactor [▶ 99] kann Daten von mehreren Sensoren gleichzeitig auswerten, vorausgesetzt die Zahl der Einzelwerte ist für jeden Kanal gleich. Der Rückgabewert besteht aus einem Einzelwert für jeden Kanal. Die Einzelwerte werden in einem Vektor zurückgegeben. Dies soll in obiger Grafik durch die Verwendung der Pfeile in horizontale und vertikale Richtung angedeutet werden. Der Crestfaktor-Baustein erhält hier jeweils 5 Einzelwerte (vertikal) für 7 Kanäle (horizontal) und gibt entsprechend für jeden der 7 Kanäle den Crestfaktor aus.

Die Kurtosis kann durch den Baustein MomentCoefficients [▶ 183] ausgewertet werden. Hierbei werden die Werte wahlweise für alle Kanäle und einzelne Zeitschritte übergeben, oder blockweise für mehrere Zeitschritte, was für einkanalige Signale aufgrund des geringeren Overheads effizienter ist.

Einhüllendenspektrum

Theorie

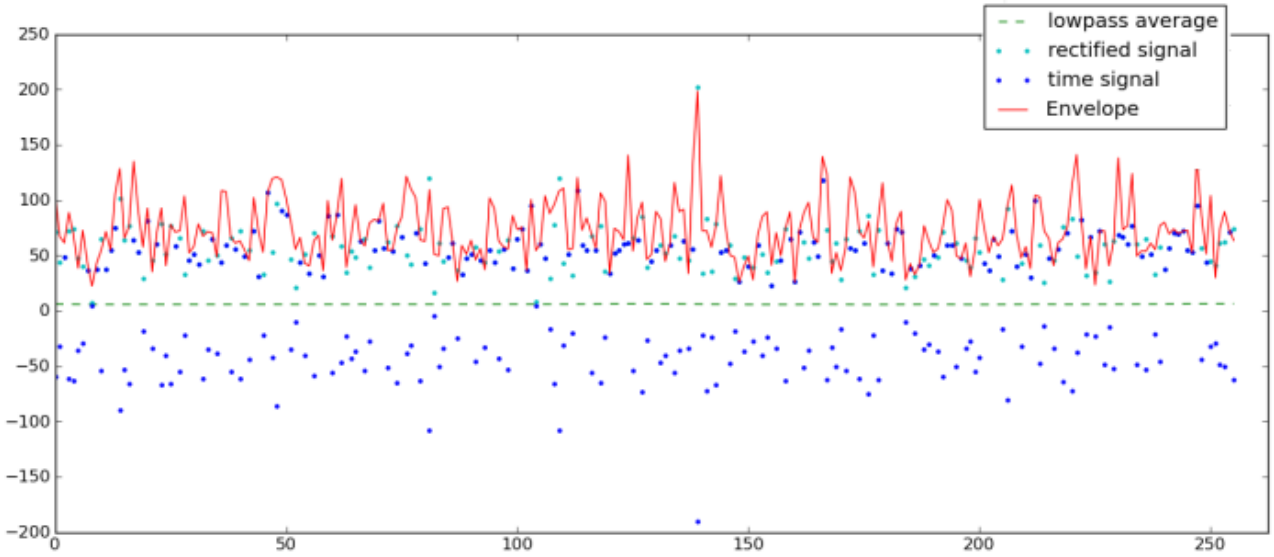
Die Bestimmung von Crestfaktor oder Kurtosis erzielt mit sehr geringem Aufwand frühe Hinweise auf das Vorliegen eines Schadens. Da die Zerlegung und Inspektion von Anlagenteilen immer einen – unter Umständen erheblichen – Aufwand bedeutet und gegebenenfalls eine Vielzahl von Lagern vorhanden sind, sind zusätzliche Diagnosemöglichkeiten von Interesse, mit denen sich beschädigte Lager oder sogar Einzelelemente genauer identifizieren lassen. Die Defekterkennung beruht auf der Auswertung von Stößen,

die auf Beschädigungen der Kontaktflächen zurückgehen. Bei einem Schaden an einem rotierenden Teil entstehen die Stoßimpulse periodisch, wobei die Länge der Periode von der Frequenz abhängt, mit der ein Defekt eine Kontaktfläche berührt. Diese Stoßimpulsperiode hängt einerseits von der Rotationsgeschwindigkeit des Lagers, darüber hinaus jedoch auch von der Geometrie der Elemente ab. Somit kann die Periode der Stoßimpulse das defekte Bauteil identifizieren.

Die Stoßimpulse beinhalten sowohl einen hochfrequenten Signalanteil, welcher auf die Schwingungen des angestoßenen Maschinenteils zurückzuführen ist, als auch einen damit überlagerten (gefalteten) und eventuell auch modulierten niederfrequenten Anteilen, welcher Information über die periodische Wiederholungen der Stößen enthält. Diese niederfrequenten Signalanteile kann man durch die Berechnung der Einhüllenden (Envelope) ermitteln. Die Einhüllende kann durch Anwendung der Hilbert-Transformation im Frequenzbereich effizient berechnet werden. Eine vorherige Filterung durch einen Hochpassfilter, wie z.B. von der TwinCAT 3 Controller Toolbox bereitstellt, kann sinnvoll sein, ist jedoch nicht zwingend notwendig. Anschließend an die Berechnung der Einhüllenden wird das Leistungsspektrum des Einhüllendensignals ermittelt. Die markanten Frequenzen dieses Einhüllendenspektrums identifizieren die Stoßperioden.

Anwendung

Das Einhüllendenspektrum ist insbesondere ein Hilfsmittel für eine Diagnose, welche Aggregate oder welche Elemente eines Lagers möglicherweise defekt sind. Daneben ist für die Früherkennung von Schäden die Möglichkeit interessant, gezielt wichtige Signalanteile auszuwerten und Störanteile auszuklammern. Soll es für eine Früherkennung genutzt werden, so müssen die in Frage kommenden Schadfrequenzen aus der Lagergeometrie ermittelt und überwacht werden.



Das obige Diagramm zeigt die Einhüllende des schon zuvor verwendeten Signals eines beschädigten Wälzlagers. Das Zeitsignal wird durch die blauen Punkte markiert, die Einhüllende durch die rote Linie. Zum besseren Verständnis ist außerdem grün gestrichelt der gleitende Mittelwert des Zeitsignals geplottet, der nicht exakt Null ist, sowie hellblau die Beträge der negativen Werte im Zeitsignal. Man sieht, dass die Einhüllende immer sehr nahe an den maximalen Werten des Zeitsignals bzw. des Betrages der Zeitwerte liegt. Spitzen oder negative Ausschläge im Zeitsignal führen zu Spitzen in der Einhüllenden, während ein "Grundrauschen" im Zeitsignal durch die Einhüllendenbildung nur wenig verändert wird.

Analyse des Hüllkurvenspektrums

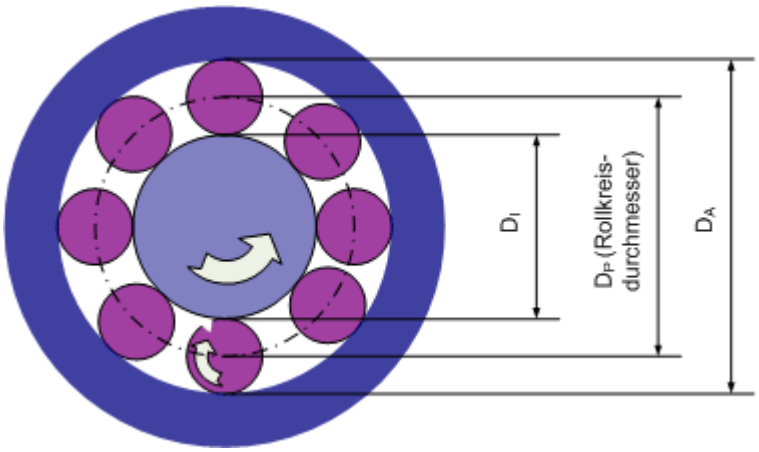
Da eine Sequenz von periodischen Stößen (Pulse Train) einem Signal mit vielen Oberschwingungen entspricht, treten im Hüllkurvenspektrum einerseits die Grundfrequenz und andererseits die ganzzahligen Vielfachen der Grundfrequenz auf. Dabei bestimmen sich – genauso wie bei einem Leistungs- oder Magnitudenspektrum – die den Spektralwerten zugeordneten Frequenz aus dem Index des Ergebnis-Array multipliziert mit der Frequenzauflösung der FFT, siehe Frequenzauflösung [▶ 13]. Mit der Länge der FFT N und der Abtastrate f_s folgte: $\Delta f = f_s / N$ und damit für die Frequenz des Frequenz-bin mit Index m : $f_m = (m-1)\,\Delta f$ (unter der Annahme, dass der Array-Index m mit 1 beginnt).

Zur Diagnose muss die Grundfrequenz der Stoßfolge identifiziert werden. Die Harmonischen sind zu erkennen an einer kammartigen Abfolge von spitzen Maxima, die einen gleichmäßigen Abstand haben. Die Grundfrequenz ist der Abstand zwischen diesen Maxima, in der Regel also die Frequenz des ersten Maximums dieser Reihe. Ihr Kehrwert ergibt die Periode der Stöße, die Einheit des Kehrwerts ist also eine Zeitdifferenz. Zusammen mit der Rotationsgeschwindigkeit der Achse, die zu messen ist, und den aus der Lagergeometrie ermittelbaren Drehzahlverhältnissen der Schadfrequenzen lassen sich dann die für die Ursache eines Defekts in Frage kommenden Bauteile ermitteln.

Charakteristische Schadfrequenzen in Wälzlagern

Die untenstehende Abbildung zeigt exemplarisch, welche Drehzahlverhältnisse in einem einfachen Wälzlager auftreten können. Grundsätzlich entstehen Stoßimpulse mit der Frequenz, mit der die Kontaktstelle zwischen zwei Lagerelementen eine Stelle mit beschädigter Oberfläche (im Bild an der Oberseite des Wälzkörpers ganz unten) passieren. Diese Kontaktstelle bewegt sich ebenfalls aufgrund der relativen Bewegung der Elemente zueinander. Die Drehzahl- bzw. Winkelgeschwindigkeit der Kontaktstelle kann basierend auf der Regel ermittelt werden, dass in einem korrekt funktionierenden Lager kaum Schlupf vorhanden ist, die Elemente also fast vollständig aufeinander abrollen.

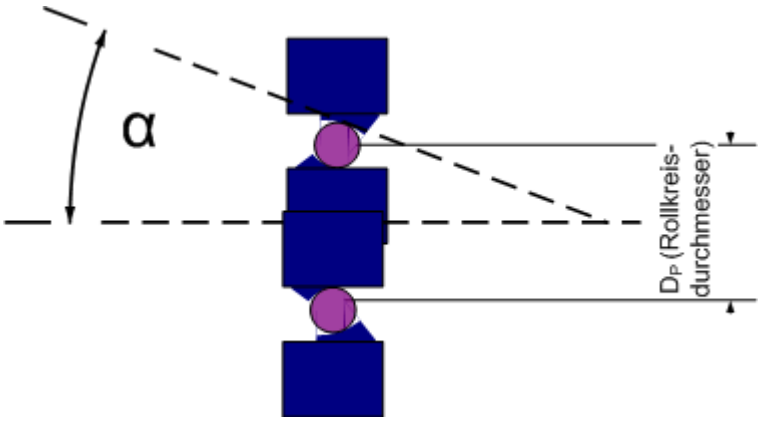
Wälzlager-Geometrieparameter



Angenommen die Drehzahl f_{rot} einer Achse, die mit dem Innenring verbunden ist, wird gemessen, und die Durchmesser der Lagerteile verhalten sich wie folgt: Durchmesser des Innenrings D_I, Durchmesser der Kugeln D_B, Durchmesser Außenring D_A. Die Zahl der Kugeln sei Z. Aus D_I und D_A ergibt sich der **Pitch Diameter** oder **Rollkreisdurchmesser**: $D_P = (D_I + D_A)/2$. Dreht sich nun der Innenring mit einer Drehzahl von f_{rot}, so können hieraus die Impulsfrequenzen ermittelt werden. Zur Bezeichnung der Frequenzen sind die folgenden Akronyme gebräuchlich:

- **BPFO** (Rolling element pass frequency outer race): Frequenz, mit der die Wälzelemente den Außenring passieren.
- **BPFI** (Rolling element pass frequency inner race): Frequenz, mit der die Wälzelemente den Innenring passieren.
- **BSF** (Bearing spin frequency rolling elements): Frequenz, mit der die Kugeln/Wälzelemente relativ zu einer Lauffläche rollen.
- **BPF** (Ball pass frequency): Wälzelementfrequenz, die Frequenz, mit der ein Defekt auf einer Kugel eine Lauffläche passiert.
- **FTF** (Fundamental Train frequency): Rotationsgeschwindigkeit des Käfigs bzw. die Lagerelement-Modulationsfrequenz

Kontaktwinkel:



Für eine genaue Rechnung bei Lagern, die axiale Lasten aufnehmen, ist der Durchmesser der Kugeln mit dem Kontaktwinkel α zu korrigieren, mit dem die Kugeln die Lauffläche berühren: $D_b = \cos(\alpha)\ D_B$. Bei Radiallagern ist dieser Winkel 0°.Dann ergeben sich die folgenden praktisch genutzten Formeln:

BPFO = $Z * f_{rot} /2 * (1 - D_b / D_P)$
BPFI = $Z * f_{rot} /2 * (1 + D_b / D_P)$
BSF = $f_{rot} /2 * D_P/D_B * (1 - (D_b/D_P)^2)$
BPF = 2 * BSF
Rotierender Innenring:
FTF = $f_{rot} /2 * (1 - D_b/D_P)$
Rotierender Außenring:
FTF = $f_{rot} /2 * (1 + D_b/D_P)$ Dabei ist (BPFI + BPFO) / f_{rot} immer gleich der Anzahl der Wälzelemente Z. Von diesen Formeln ergeben sich in der Praxis leichte Abweichungen, da beispielsweise unter Belastung der Kontaktwinkel α variieren kann. Als einfache Faustregel wird häufig der Wert

$f_{BPFI} = 0.6 * f_{rot} * Z$

als Indikatorfrequenz für einen Defekt des Innenrings verwendet, sowie

$f_{BPFO} = 0.4 * f_{rot} * Z$

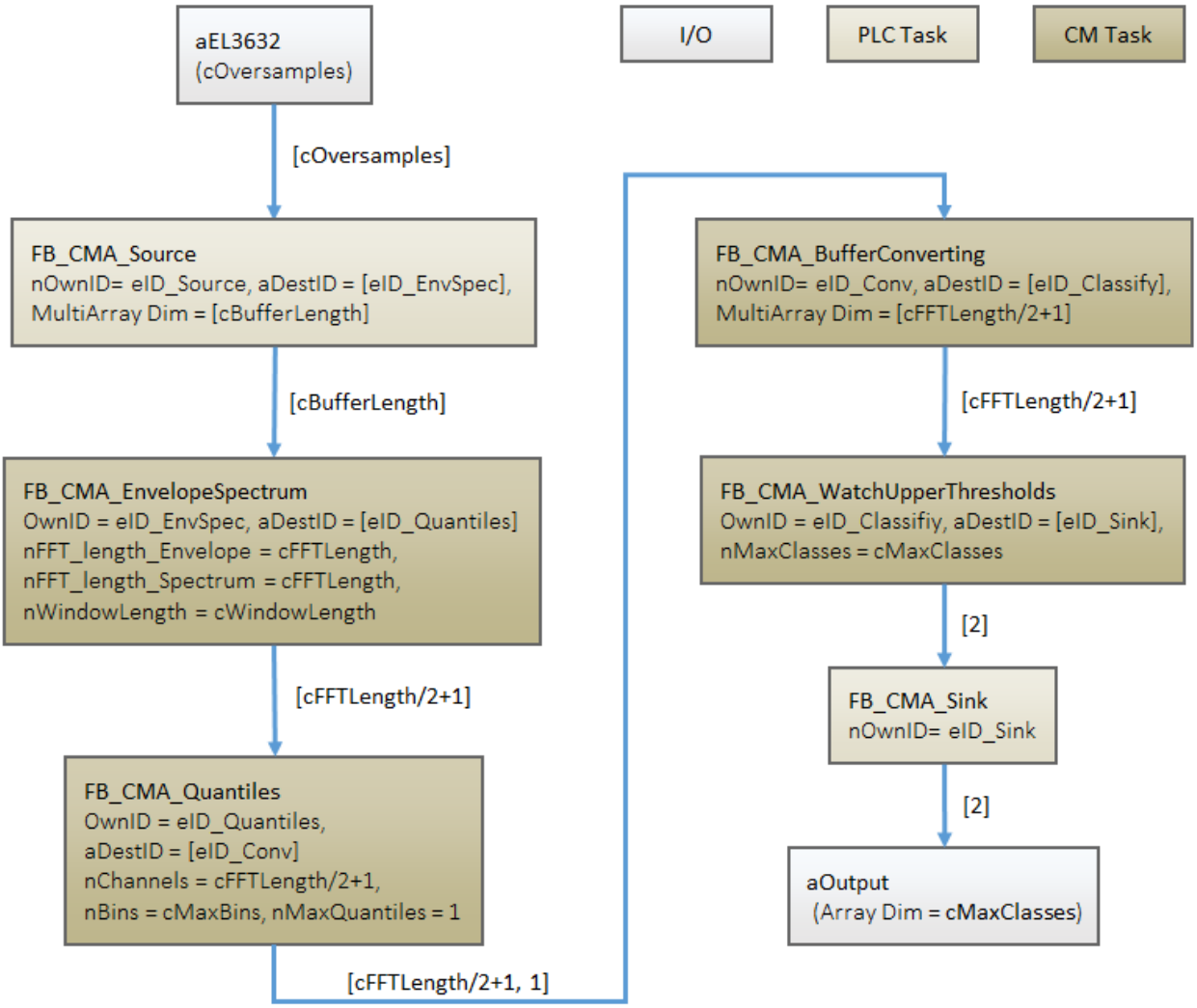
für einen Defekt des Außenrings. Zur Ermittlung der Lagergeometrie ist es sinnvoll Daten des Lagerherstellers hinzuzuziehen. Ebenfalls kann es hilfreich sein, Berechnungsprogramme zu nutzen, welche die Hersteller teilweise im Internet anbieten.

Die Typnummer eines Wälzlagers erlaubt keinen eindeutigen Rückschluss auf die Lagergeometrie; Parameter wie die Anzahl der Wälzkörper können sich durchaus ändern.

Verarbeitungskonzept

Verarbeitungsschritte der Frequenzanalyse:

Analyseschritte:



Das obige Diagramm zeigt die Verarbeitungsschritte für das Einhüllendenspektrum sowie die Bausteine, die hier verwendet werden können. Dabei wird zunächst mit dem Baustein Envelope [▶ 146] die Einhüllende berechnet. Anschließend wird, genauso wie bei der Berechnung des Spektrums eines beliebigen Zeitsignals, das Leistungsspektrum berechnet (Baustein PowerSpectrum [▶ 202]). Da das erhaltene Hüllkurvenspektrum bei nicht stationären Signalen relativ stark fluktuiert, empfiehlt sich, dieses wie zuvor im Abschnitt Frequenzanalyse [▶ 37] erläutert, mit der Methode der Quantilsberechnung (Quantiles) statistisch auszuwerten. Die erhaltenen Werte können mit einer Grenzwertüberwachung durch den Baustein WatchUpperThreshold [▶ 262] automatisch auf die Einhaltung bestimmter Schwellwerte kontrolliert werden.

2.2.4 Getriebeüberwachung

Motivation

Dieser Abschnitt erläutert das Konzept einer Überwachung von Getrieben. Wie Wälzlager zählen Getriebe zu den häufigsten Maschinenelementen. Da sie unter anderem in Antrieben aller Art eingesetzt werden, haben sie zumeist eine entscheidende Rolle für die zuverlässige Funktion einer Anlage. Typische Getriebeschäden unterscheiden sich von Schäden in Wälzlagern. Dies liegt zum einen daran, dass in Getrieben hoch belastete Teile direkt aufeinander gleiten, was besondere Anforderungen sowohl an die Schmierung als auch an die Qualität der Oberfläche stellt. Aufgrund der zur Aufnahme der im normalen Betrieb entstehenden Kräfte sind Getriebe deswegen relativ groß und damit teuer und ein Austausch während der Lebensdauer der Maschine kann auch bei einwandfreier Wartung notwendig werden. Ausreichende Schmierung und korrekte Montage sind auch hier wichtig. Die auftretenden Schadensbilder sind jedoch keineswegs ausnahmslos auf Fehler in diesen Punkten zurückzuführen. Zu hohe Spannungen an den Kontaktstellen ebenso wie ein Zusammenwirken mit Korrosion und Überhitzung können von

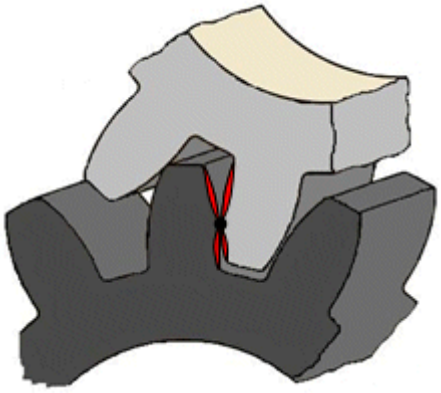
anfänglichen Oberflächenschäden (Pitting, Graufleckigkeit / Micropitting, Oberflächenausbrüche / Spalling, Abrieb) bis hin zu Absplitterungen und Deformationen der Zahnflächen führen. Mechanische Schocks und Überlast können den unmittelbaren Bruch von Zahnrädern verursachen. Des Weiteren führt ein Versagen von Getrieben, eher als bei Wälzlagern, zu abrupten Ausfällen und signifikanten Folgekosten. Dies liegt daran, dass bei Zahnrädern die größten Spannungen am Ansatz der Zähne liegen, vergleiche untenstehende Abbildung (rote Flächen). Folglich treten dort früh Ermüdungserscheinungen auf, die im Laufe der Zeit zu tiefen Rissen und schließlich zum Herausbrechen von Zähnen führen. Letzteres führt im Extremfall dazu, dass das ganze Getriebe scheinbar ohne Vorwarnung blockiert und umfangreiche Folgeschäden z.B. durch den Bruch von Achsen entstehen. Aus den gerade genannten Ursachen und dem daraus folgenden Verhalten ergeben sich zwei Zielrichtungen für eine Überwachung von Getrieben:

- Erstens ist es von Interesse, Abnutzungserscheinungen langfristig zu überwachen und durch Trendbeobachtungen Probleme frühzeitig zu erkennen und zu beheben, bevor Folgeschäden auftreten.
- Zweitens lassen sich akute Schäden durch eine Überwachung sofort erkennen, wodurch Reparaturmaßnahmen früher eingeleitet und Ausfälle und Stillstandzeiten verringert werden können.

Theorie

Im Folgenden werden nun die theoretischen Grundlagen zu einer frühen Erkennung von Getriebeschäden kurz skizziert.

Zahneingriffsschwingungen

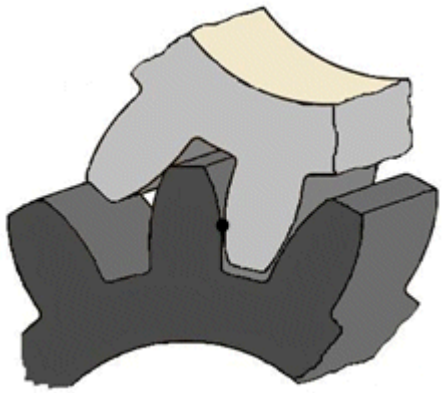


In einem Getriebe rollen Zahnräder aufeinander ab, wobei periodisch die einzelnen Zähne in Kontakt kommen, Kraft übertragen und sich wieder voneinander lösen. Während es in einem neuen, gut ausgelegten Getriebe möglich ist, dass dies mit exakt konstantem Übersetzungsverhältnis und weitgehend konstanter Kraft geschieht (Evolventenverzahnung), ist es nicht machbar, dass dieses Abrollen ohne einen Anteil an gleitender Bewegung erfolgt. Wie das obige Bild zeigt, erfolgt in der Mitte der Zahnfläche vorwiegend eine Drehbewegung und mit zunehmender Entfernung von der Mitte eine wachsender Anteil Gleitbewegung. Bei einem solchen Zahnkontakt ist zudem zwar das Drehzahlverhältnis weitgehend konstant, das übertragene Drehmoment variiert jedoch. Da die Zähne aus hartem, elastischem Material bestehen und sich somit geringfügig verformen, werden diese zu einer Schwingung mit der Periode des Zahneingriffs angeregt, der sogenannten Zahneingriffsfrequenz.

Oberschwingungen der Zahneingriffsfrequenz

Da die Zahneingriffsschwingung eine erzwungene Schwingung ist, die nicht sinusförmig aussieht, sondern auf einem vergleichsweise plötzlichen Auftreten und Nachlassen der Kräfte basiert, besteht sie im Spektrum aus zahlreichen Oberschwingungen, deren Frequenzen ganzzahlige Vielfache der Zahneingriffsfrequenz sind. Die Schwingungen hängen von der Belastung des Zahnrads ab, da das Drehmoment die Zähne elastisch verformt. Zahnradschwingungen sind also lastabhängig.

Abnutzungsfolgen



Bei zunehmender Abnutzung weichen die Zahnprofile mehr und mehr von der idealen Form ab, da durch das Gleiten der Flächen aufeinander Material abgetragen wird. Dies geschieht umso stärker, je weiter die Fläche vom Mittelpunkt der Zahnflanke entfernt ist, wie die Grafik oben zeigt. Die Gleitbewegung nimmt deswegen ihrerseits zu und das Drehmoment variiert stärker, wodurch sich die Zahneingriffsschwingungen und besonders die enthaltenen harmonischen Oberschwingungen verstärken. *Die Analyse der Oberschwingungen ist daher der Schlüssel zur Auswertung des Getriebezustands.* Zu beachten ist, dass ein plötzlicher Rückgang von Oberschwingungen in einem schon deutlich beschädigten Getriebe als Alarmzeichen zu werten ist: Möglicherweise ist der Bruch einer Zahnflanke so stark fortgeschritten, dass sich die Elastizität der Verzahnung erhöht hat. In diesem Fall ist mit einem kurzfristigen Totalausfall des Getriebes zu rechnen.

Das Cepstrum

Das Cepstrum ist das wichtigste Werkzeug zur Analyse von Getriebeschwingungen sowie Oberschwingungen und Modulationen. Es stellt eine Operation dar, welche Periodizitäten im Spektrum eines Signals hervorhebt.

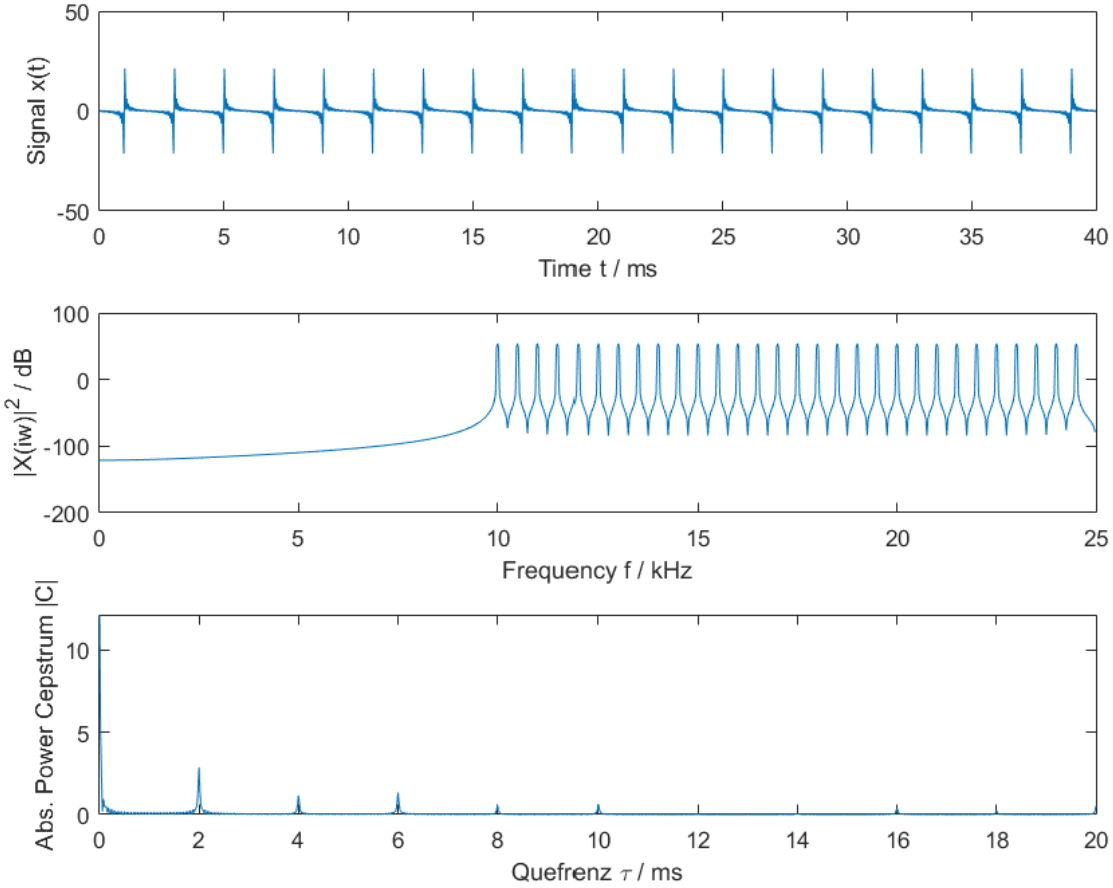
Die Definition des Leistungs-Cepstrums für ein Signal *x*(*t*) lautet:

$$C_{\mathrm{p}}(\tau) = X^{-1}\left\{\log(|X(\omega)|^2)\right\} = X^{-1}\left\{2\log|X(\omega)|\right\}, \qquad \text{Quefrency } \tau$$

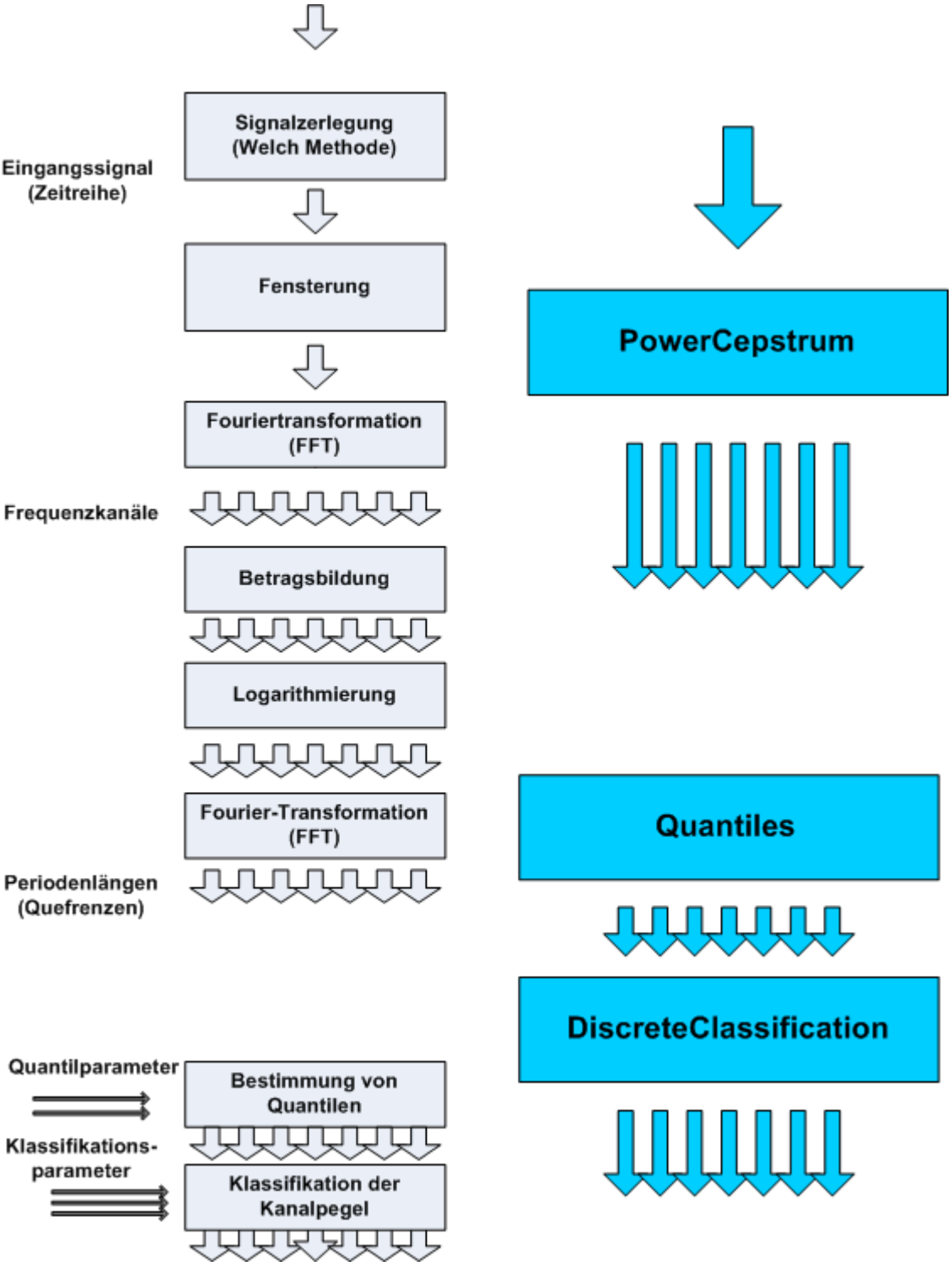
Interpretation

Ebenso wie die Fourier-Analyse Periodizitäten im Zeitbereich eines Signals sichtbar macht, wertet das Cepstrum Periodizitäten im Frequenzbereich aus. Durch die inverse Fourier-Transformation wird das Ergebnis wieder in einen Zeitbereich abgebildet. Der assoziierte Index der Werte stellt jedoch nicht die ursprüngliche Zeitachse bzgl. *t* dar, sondern die auftretenden Perioden im Spektrum. Diese Größe, deren Einheit die Zeiteinheit ist, wird aufgrund der Kombination von Umkehrung und Rücktransformation als Quefrency bezeichnet. Ähnliche unterscheidende Bezeichnungen gibt es z.B. für Entitäten und Operationen wie Harmonische, Filterung und Phasenanalyse. Je größer die Länge *N* der beiden verwendeten Fourier-Transformationen ist, desto mehr Eingangswerte werden zur Berechnung des Cepstrums herangezogen, was den Einfluss von Rauschen und (nicht systematischen) Schwankungen verringert. Vergrößert werden kann die Zeitauflösung nur, wenn die Abtastrate erhöht wird.

Als Beispiel zeigt die folgende Abbildung das Leistungsspektrum und Leistungs-Cepstrum eines sogenannten harmonischen Tonkomplexes. Im Zeitbereich des Signals ist eine Wiederholung eines Pulses alle 2 ms erkennbar. Jeder Einzelpuls wurde aus einer Überlagerung von Oberschwingungen zusammengesetzt, das heißt die Situation ist ähnlich (grobes Modell) wie im oben beschriebenen Fall eines Getriebeschadens. Das Leistungsspektrum ist in der mittleren Darstellung aufgezeigt. Deutlich zu erkennen ist die Periodizität des Leistungsspektrums, wobei die Maxima jeweils 0,5 kHz auseinanderliegen. Der Betrag des Leistungs-Cepstrums ist in der unteren Darstellung zu sehen. Das größte (globale) Maximum liegt bei einer Quefrenz von 0 ms, was praktisch keine Relevanz hat (es zeigt nur den Mittelwert des Leistungsspektrums). Abgesehen von diesem Maximum ist das größte Maximum bei 2 ms zu erkennen, was genau der zeitlichen Wiederholung des Zeitsignals entspricht bzw. dem Kehrwert des Abstandes der lokalen Maxima im Leistungsspektrum 1/0,5 kHz = 2 ms.


Signal x(t)
50
0
-50
0 5 10 15 20 25 30 35 40
Time t / ms
|X(iw)|^2 / dB
100
0
-100
-200
0 5 10 15 20 25
Frequency f / kHz
Abs. Power Cepstrum |C|
10
5
0
0 2 4 6 8 10 12 14 16 18 20
Quefrenz τ / ms

Verarbeitungskonzept (Rechenschritte)



Berechnung des Leistungs-Cepstrums

Die Berechnung des Cepstrums basiert, wie aus der Definition hervorgeht, auf der "normalen" Frequenzanalyse. Dementsprechend ist, wie im Abschnitt Analyse von Daten-Streams [▶ 18] beschrieben, zunächst eine Zerlegung des Signals in Abschnitte und eine anschließende Multiplikation mit einer Fensterfunktion bzw. "Fensterung " notwendig. Anschließend wird entsprechend den oben angegebenen Rechenschritten Fourier-Transformation, Betragsbildung, Logarithmierung und erneute Fourier-Transformation das Leistungs-Cepstrum berechnet. Wichtig ist hierbei, dass Wertbereichsüberschreitungen vermieden werden, denn der Logarithmus von Null ist ähnlich wie die Division durch Null nicht definiert.

Das Ergebnis der Berechnung ist zunächst komplex-wertig. Typischerweise wird zur weiteren Analyse der Betrag oder auch das Betrags-Quadrat verwendet.

Ein Beispiel kann hier heruntergeladen werden: Leistungscepstrum [▶ 365]

Berechnung von Quantilen

Die kurzzeitigen Werte des Cepstrums fluktieren, ähnlich wie die der FFT von der sie abgeleitet sind, meistens recht stark. Deswegen ist der nächste empfohlene Verarbeitungsschritt die Berechnung von Quantilen für jede erhaltene Periode, also jede Quefrency. Für Überwachungsaufgaben wird oft beispielsweise die 95-% Quantile, also jenen Wert, der von den Messwerten in 95 % aller Fälle nicht überschritten wird, bestimmt. Diese Berechnung geschieht ebenso wie bei der Frequenzanalyse mit dem Baustein Quantiles [▶ 206].

Schwellwertüberwachung

Die weitere Verarbeitung hängt von der konkreten Zielsetzung ab:

- Für Trendanalysen ist es sinnvoll, die erhaltenen Werte zu speichern und ihre Entwicklung über lange Zeiträume darzustellen.
- Für die automatische Maschinenüberwachung ist eine Klassifikation in konfigurierbare Schwellen oder Grenzwerte angezeigt. Diese leistet der hier eingezeichnete Baustein DiscreteClassification [▶ 121].
- Für Aufgabenstellungen wie einen Maschinenschutz, bei dem für eine individuelle Analyse keine Zeit vorhanden ist, kann der Baustein WatchUpperThreshold [▶ 262] verwendet werden, der automatisch die Nummer der höchsten Grenzwert-Kategorie berechnet. Wird also beispielsweise der Kategorie 0 der Zustand "alles in Ordnung", der Kategorie 1 der Zustand "Warnung" und der Kategorie 2 der Zustand "Alarm" zugeordnet, so kann bei Ausgabe des Levels 1 eine Warnung per SMS verschickt und bei Level 2 die Anlage automatisch abgeschaltet werden.

2.2.5 Lebensdaueranalyse und Schädigungsrechnung

Motivation

Wie lange hält eine strukturelle Komponente, wie z. B. ein Turm einer Windkraftanlage, Brückenträger, Lastenkräne oder auch Maschinenkomponenten, die zyklischen Lasten ausgesetzt sind?

Die Antwort auf diese Fragen liefern rechnerische Lebensdaueranalysen. Sie verwenden Betriebslasten, Geometrie- und Materialdaten, um die Ermüdungsfestigkeit einer Konstruktion zu ermitteln. Ist das Bauteil oder die Konstruktion in Betrieb, kann mit einem kontinuierlichen Monitoring der realen Betriebslasten eine Abschätzung der Restlebensdauer realisiert werden.

Vor allem der Leichtbau hat den ressourcensparenden Einsatz von Rohstoffen und Energie bei Produktion und Betrieb zum Ziel. Durch leichtere Konstruktionen lassen sich nicht nur Kosten reduzieren, sondern auch neuartige Produkte mit komplexeren Geometrien realisieren. Diese Zielstellungen sind natürlich nicht nur auf den Bereich Leichtbau beschränkt, sondern erstrecken sich mehr und mehr über alle Bereiche mechanischer Konstruktionen. Vormals überdimensionierte Bauteile werden den voraussichtlich im Betrieb auftretenden Lasten angepasst. Entsprechend reduzieren sich auch die Sicherheitsmargen gegenüber Belastungen – die Bauteile sind also nicht mehr als *dauerfest* konstruiert. Eine *betriebsfeste* Konstruktion hingegen versucht, ein Bauteil so auszulegen, dass es für den definierten Einsatzzweck und bei definiertem Belastungsprofil eine gewünschte Lebensdauer ohne Versagen übersteht.

Wie im Bereich Schädigungsüberwachung [▶ 370] im Abschnitt Beispiele [▶ 325] aufgezeigt, ist die Software zur Lebensdaueranalyse mit TwinCAT 3 Condition Monitoring einfach realisierbar. Darüber hinaus bietet das Beckhoff-System eine hervorragende Plattform, die notwendigen Messgrößen aufzunehmen (bspw. ELM35xx), sie per EtherCAT auch über weite Strecken zu kommunizieren, sie auf einem verlässlichen IPC zu verarbeiten und schlussendlich an überlagerte Systeme weiterzureichen, zu speichern oder anzuzeigen.

Der folgende Abschnitt vermittelt Hintergrundwissen und dient der Einordnung des Themengebiets. Die Beschreibung der relevanten Funktionsbausteine sowie das aufbereitete Beispiel dienen exemplarisch der praktischen Umsetzung.

Hintergrund zur Lebensdaueranalyse

In diesem Abschnitt lernen Sie:

- Wie berechnet man die Ermüdungslebensdauer eines Bauteils und was ist eine Schadensakkumulation?
- Was sind Rainflow-Zyklenzählungen?
- Was ist eine Wöhler-Kurve?
- Wie werden Wöhler-Kurven ermittelt?

Grundlegende Konzepte

In den Anfängen der Ermüdungsrechnung vor ca. 200 Jahren, weit vor dem heutigen standardmäßigen Einsatz von Simulationstools und Messtechnik, wurden die ersten Fälle von Bauteilversagen näher untersucht: Bereits im Jahre 1837 hat Wilhelm Albert, Ingenieur aus Clausthal, erste Untersuchungen zur Ermüdung von Stahlketten durchgeführt. Seine Beobachtung, dass die Ketten nicht durch Überlastung, sondern auch bei häufigen, zyklischen Belastungen mit kleinerer Amplitude versagen, hat den Grundstein für die systematische Untersuchung der Ermüdungsfestigkeit gelegt. Um 1860 hat Albrecht Wöhler, ein deutscher Eisenbahn-Ingenieur, eine mathematische Formulierung der Dauerschwingfestigkeit von Metallen, der sog. **Wöhler-Kurve**, formuliert. Hierzu hat er Materialproben zyklischen Belastungen ausgesetzt und die Tests bis zur Zerstörung gefahren. Aus diesen Messungen konnte August Wöhler in einem Diagramm die mechanische Spannung über der erreichten Schwingspielzahl auftragen und eine Kennlinie des getesteten Materials ableiten.

Ergänzend zu diesen grundlegenden Beobachtungen wurde der Einfluss der **Mittelspannung**, d. h. einer konstanten mechanischen Vorspannung, auf die Ermüdungslebensdauer von Gerber und Goodman näher untersucht. Sie haben Berechnungsmethoden entwickelt, um den Mittelspannungseinfluss in den Berechnungen zu berücksichtigen.

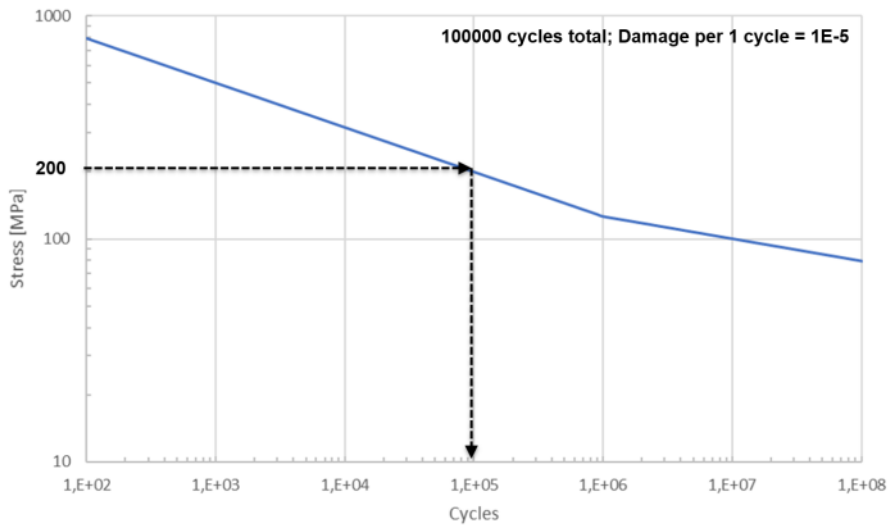
Die Hypothese der linearen **Schadenakkumulation** wurde 1924 von A. Palmgren aufgestellt und von M.A. Miner im Jahr 1945 veröffentlicht. Sie besagt, dass bei einer Gesamtschädigung von $D = 1$ das betrachtete Bauteil versagt.

$$D = \sum_{i=1}^{k} \frac{n_i}{N_i}$$

Hier beschreiben N_i die maximal Anzahl an Schwingzyklen bis zum Bauteilversagen und n_i die bereits vom Bauteil erfahrene Anzahl von Schwingzyklen. Das Verhältnis n_i/N_i wird als Teilschädigung bezeichnet und betrachtet nur die Schädigung, die durch eine feste Schwingbreite verursacht wird. Die Summe aller Teilschädigungen über alle auftretenden Schwingbreiten ist dann die Gesamtschädigung D. Der Index i läuft entsprechend über k Klassen, welche jeweils eine Schwingbreite definiert.

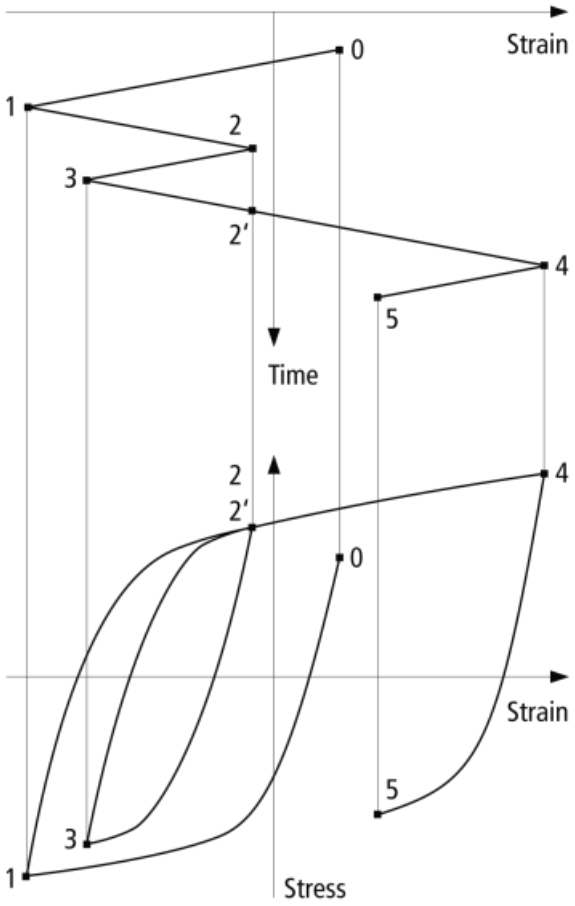
Zur Definition von Mittelspannung und Schwingbreite: Angenommen ein Material wird mit einer konstanten Kraft vorgespannt und dann zusätzlich mit einer harmonischen Schwingung mit Amplitude y_p beaufschlagt. Die mechanische Spannung im Material setzt sich dann zusammen aus der Mittelspannung y_m und einer harmonischen Schwingung mit Schwingbreite von $2*y_p$.

Im folgenden Beispiel wird ein Bauteil mit einer Schwingbreite von 200 MPa beansprucht. Die Wöhler-Kurve definiert zu dieser Beanspruchung eine Schwingspielzahl von 100.000 (1E5) Zyklen. Ein Zyklus „verbraucht“ somit 1/100.000 der Ermüdungslebensdauer. Die Wähler-Kurve ist doppelt-logarithmisch aufgetragen.



Belastungen an realen Bauteilen haben, im Gegensatz zu Werkstoffprüfungen, fast beliebige Signalverläufe. Es ist demnach notwendig, das zeitliche Beanspruchungssignal mit einem geeigneten Verfahren in Klassen zusammenzufassen, welche dann nach obigem Verfahren nach Miner gezählt werden können.

M. Matsuishi und Tatsuo Endo haben 1968 mit der **Rainflow Zyklenzählung** ein Verfahren vorgestellt, welches die Zeitsignale auf einer um 90° gedrehten Zeitachse darstellt. Die Ähnlichkeit mit einem Pagodendach, über das Regen an den Umkehrpunkten abläuft, gab dem Verfahren den Namen. Es werden zunächst Halbzyklen, z. B. in Zugrichtung, gezählt und mit passenden Halbzyklen in gegengesetzter Beanspruchungsrichtung, z. B. Druckrichtung, zu geschlossenen Zyklen kombiniert. Dieses Verfahren ist in der ASTM E 1049-85 (Standard Practices for Cycle Counting in Fatigue Analysis) genormt und weit verbreitet.



Ermittlung und Bedeutung von Wöhler-Kurven

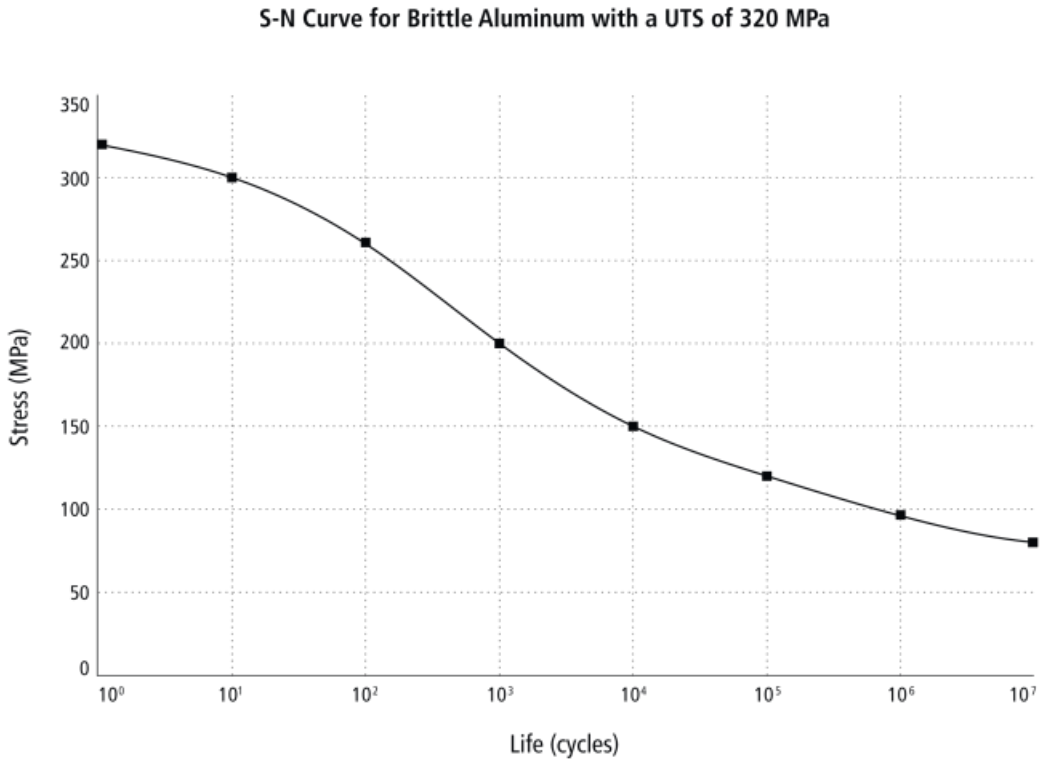
Zur Ermittlung der Wöhler-Kurve werden Materialproben mit einer definierten Probengeometrie (häufig eine Rundprobe mit 10 mm Durchmesser) in eine Prüfmaschine eingespannt. Diese leitet zyklisch Kräfte in die Probe ein, meist symmetrisch um den Nullwert (Mittelspannungsfrei) mit einer konstanten Maximalkraft schwingend. Es sind aber auch Abwandlungen im Druck- oder Zugschwellbereich möglich, um Materialparameter bei anderen Mittelspannungswerten zu ermitteln. Der Versuch wird bei einer konstanten Maximalamplitude bis zur Zerstörung (= Ermüdungsbruch) oder bis zum technischen Anriss gefahren und die erreichte Zyklenzahl für diese Laststufe im Diagramm eingetragen.

Ein Material reagiert bei einer äußeren Belastung, also der eingeleiteten Kraft *F*, aufgrund seiner Geometrie mit Querschnitt *A* mit einer Beanspruchung, die als mechanische Spannung definiert ist. Bei bekanntem (initialem) Probendurchmesser und bekannter einwirkender Kraft kann somit die resultierende Bauteilbeanspruchung als Quotient aus *F* und *A* ermittelt werden. Sie wird üblicherweise in der Einheit Megapascal (MPa) angegeben.

Entsprechend wird also ein mechanischer Spannungswert in der Wöhler-Kurve über die erreichbare Zyklenzahl aufgetragen. Um die natürliche Streuung der Tests zu ermitteln, werden pro Laststufe stets mehrere Wiederholungen mit gleichwertigem Material und gleicher Probengeometrie durchgeführt.

Die Durchführung von Schwingfestigkeitsversuchen ist u. a. in der DIN 50100 normiert. Auch die ASTM hat mit der ASTM E466-15 ein Standardverfahren für die Ermüdungsversuche mit konstanter Amplitude veröffentlicht.

Exemplarisch ist im Folgenden eine Wöhler-Kurve für Aluminium in einfach logarithmischer Darstellung abgebildet.



Die Wöhler-Kurve lässt sich in drei Bereiche einteilen:

Kurzzeitfestigkeit (Low Cycle Fatigue, LCF): Bereich hoher Beanspruchungsamplituden (N = 1 … ca. 10^4 Zyklen). In untenstehender Grafik Gelb dargestellt.

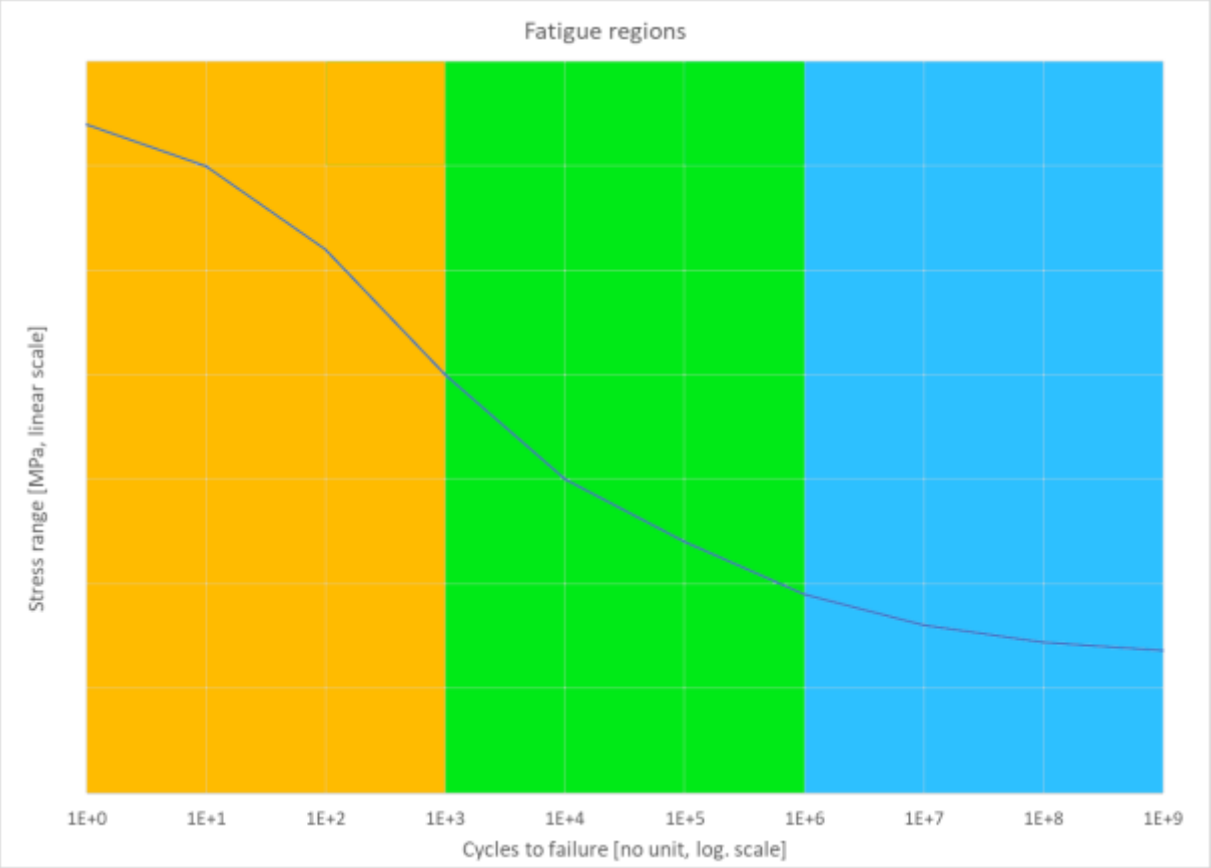
- In diesem Bereich findet eine starke Beanspruchung statt, bei der sich das Material plastisch stark verformt. Für diesen Bereich bilden dehnungsgesteuerte Materialversuche das Verhalten deutlich besser ab; es werden sog. Dehnungs-Wöhlerlinien verwendet.
- Nur wenige Bauteile werden in der Praxis im Bereich der Kurzzeitfestigkeit ausgelegt.

Zeitfestigkeit/Betriebsfestigkeit (High Cycle Fatigue, HCF): Bereich konstanter Steigung wenn die Wöhler-Kurve in x- und y-Achse logarithmisch aufgetragen wird ([N = ca. 10^4 … 10^6). In untenstehender Grafik Grün dargestellt.

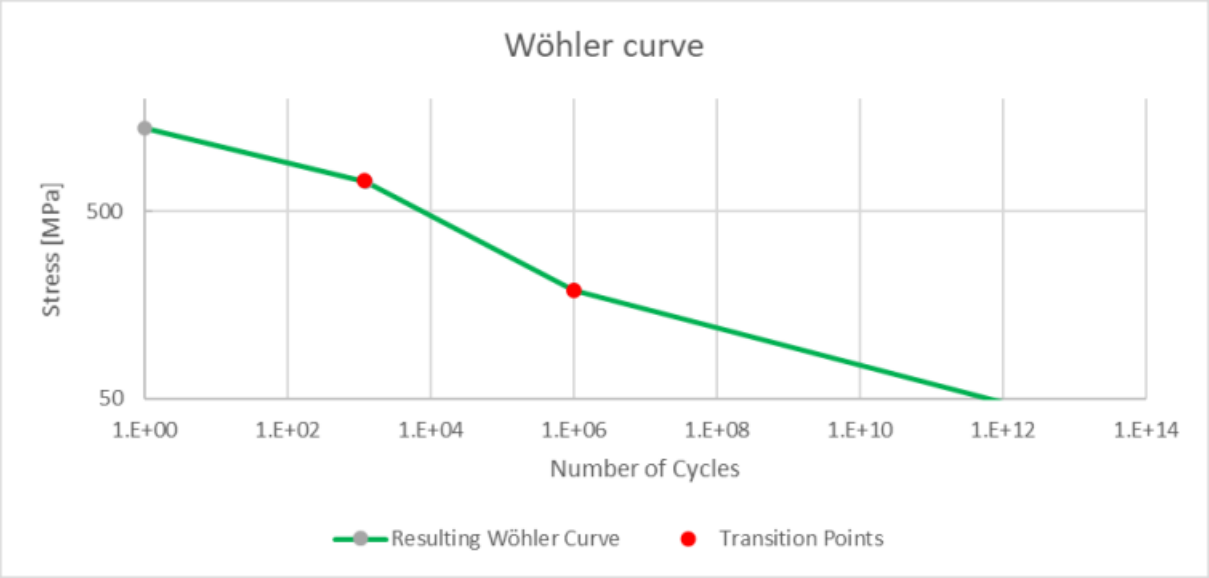
- Klassischer Bereich der betriebsfesten Bauteilauslegung.
- In Industrie und Forschung weit verbreitet und in vielen Anwendungen eingesetzt.
- Viele Materialparameter in der Literatur verfügbar.

Dauerfestigkeit (Very High Cycle Fatigue, VHCF): Bereich ohne signifikanten Abfall der Wöhler-Kurve ab einer bestimmten Amplitude [N > ca. 10^6]. In untenstehender Grafik Blau dargestellt.

- Theoretische Beanspruchungsgrenze, unterhalb der keine Ermüdung beobachtet wird (durch Korrosion und erhöhte Temperaturen treten dennoch Ermüdungserscheinungen auf).



Eine gebräuchliche Approximation einer Wöhler-Kurve in Anlehnung an obige drei Bereiche ist im Folgenden dargestellt. Dazu wird die Wöhler-Kurve in doppelt-logarithmische Darstellung umgesetzt und diese dann in drei lineare Abschnitte unterteilt. Diese Darstellung wird häufig genutzt und kann aus wenigen Materialparametern erstellt werden.

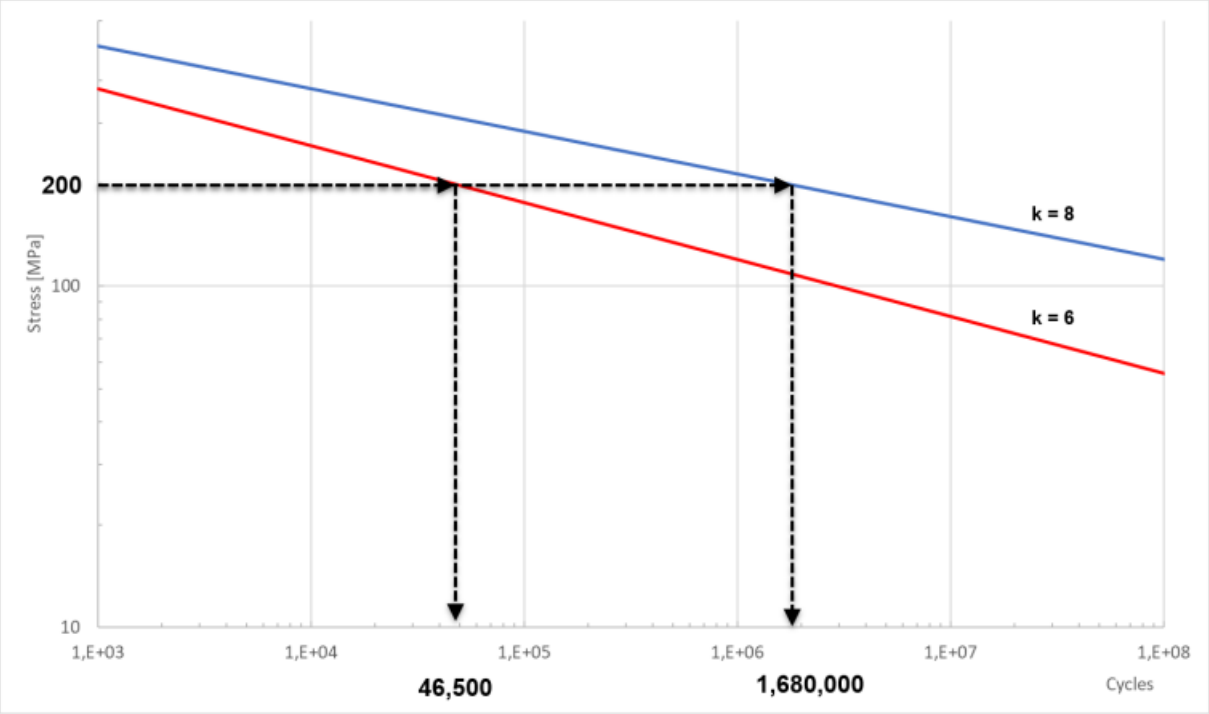


Übliche Werte für die Steigung k der Wöhler-Kurve sind:

- Geschweißte Bauteile: k = 3
- Elektronische Komponenten: k = 4
- Bauteile ohne Spannungskonzentrationen: k = ~10

Je flacher die Steigung einer Wöhler-Kurve ist (hohe k Werte), desto unempfindlicher ist das Material aus Sicht des Ermüdungsverhaltens. Kleinere Schwingbreiten schädigen das Material also weniger stark, d.h. die ertragbare Schwingspielzahl zu einer bestimmten Schwingbreite ist größer als bei einem Material mit einem steileren Verlauf der Wöhler-Kurve.

Ein steiler Verlauf eignet sich für Bauteile, die auch bei kleineren Schwingbreiten noch ermüdungsempfindlich sind: Geschweißte Konstruktionen, Bauteile mit Spannungskonzentrationen oder auch elektronische Bauteile. Im folgenden Beispiel sind bei gleichem Spannungsschnittpunkt zwei Wöhler-Kurven mit Steigungen k=8 und k=6 aufgetragen. Die ertragbaren Schwingspielzahlen unterscheiden sich deutlich (46.500 zu 1,68*10^6).



Praktische Hinweise zur Deutung der Wöhler-Kurve

Arbeitet man im Themenfeld der Materialermüdung, sollte man jederzeit bedenken, dass die Berechnungen stets eine statistische Natur besitzen. Häufig werden Wöhler-Kurven über 50 % Überlebenswahrscheinlichkeit definiert. Dies bedeutet, dass beim angegebenen Punkt einer p50-Wöhler-Kurve bereits 50 % der getesteten Prüflinge zerstört wurden. Wenn eine höhere Überlebenswahrscheinlichkeit gefordert ist, sind aus den Materialversuchen Kurven mit entsprechender Überlebenswahrscheinlichkeit abzuleiten.

Nicht nur die Ermittlung der Materialparameter abstrahiert die streuenden Messergebnisse in einige wenige einfache Kurvenparameter, sondern es unterscheiden sich auch Materialproben deutlich von realen Bauteilen. Insbesondere die reduzierte Lebensdauer von Bauteilen durch Spannungskonzentrationen spielt eine wichtige Rolle.

Geometrie:

- Probenkörper weisen keine scharfen Kanten auf und sind geschliffen (keine Kerben, Lunker etc.)
- Scharfe Kanten und Ausstanzungen im Bauteil führen lokalen Spannungsüberhöhungen und Rissbildung

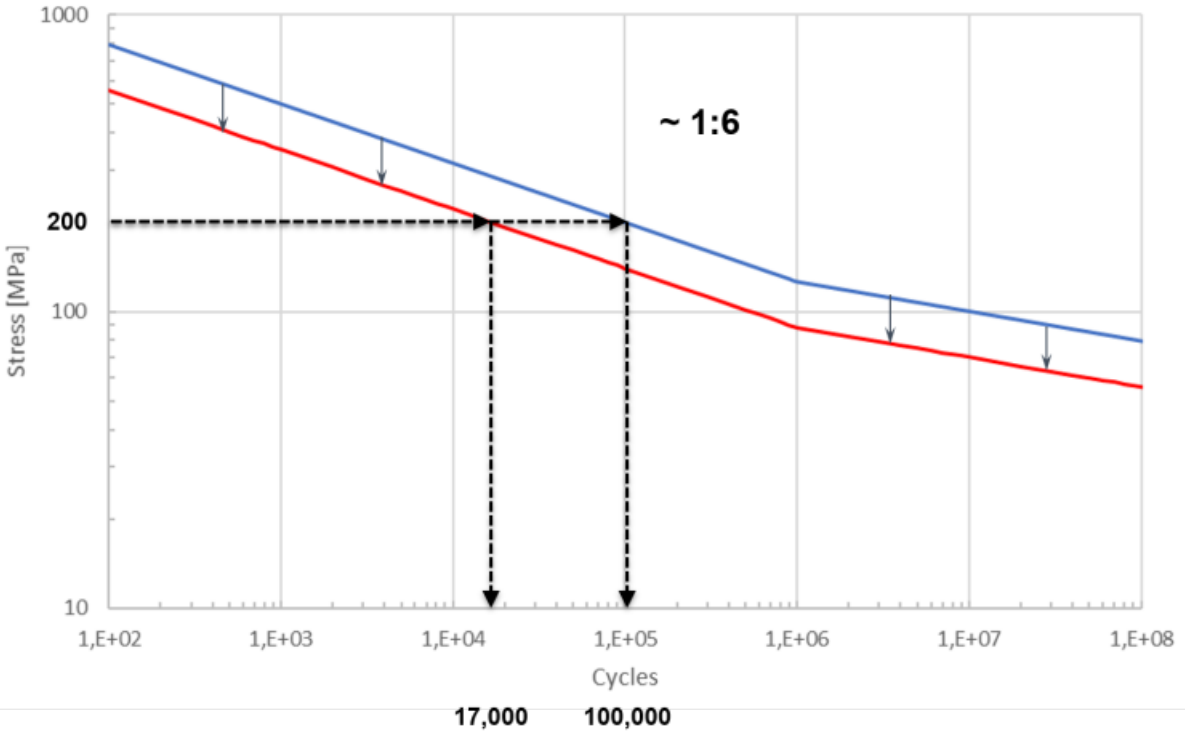
Materialqualität:

- Probenkörper besitzen eine hohe Materialqualität
- Reale Materialien in der Produktion weisen oft schwankende Qualität auf (Lunker, Variation der Herstellungsprozesse, Oxidation, Elektrokorrosion, Beschädigungen durch Verformung etc.)

Bearbeitungsprozesse:

- Schweißverbindungen in realen Bauteilen
- Durch Bearbeitungsprozesse entstehen scharfe Kanten, Ausfräsungen und Kerben. Eine Nachbearbeitung der Oberfläche oder eine Konstruktionsänderung kann diese Spannungskonzentrationen reduzieren.

Spannungskonzentrationen führen zu deutlich früherem Materialversagen. Im Folgenden ist exemplarisch der Einfluss einer Kerbwirkung (*stress concentration factor*) auf die Wöhler-Kurve dargestellt, der zu einer Absenkung der Kurve führt und die ertragbare Schwingspielzahl um Faktor 6 reduziert.



Wird für ein Bauteil/eine Komponente eine präzise Aussage zur Schwingfestigkeit gefordert, besteht die Möglichkeit – je nach Bauteilkosten, Größe und Verfügbarkeit von Bauteil und Prüfequipment – eine sog. „Bauteil-Wöhler-Kurve“ zu ermitteln. Nachteilig ist hierbei, dass neben den hohen Kosten für das Testen die Aussagekraft zwar für das aktuelle Bauteil hoch ist, dies aber eingeschränkt nutzbar sind, wenn sich die Bauteilkonstruktion ändert.

Überwachungsstrategie: Ermüdungs- bzw. Lebensdaueranalyse

Die Ermittlung der Lastdaten erfolgt direkt am Prüfling, z. B. durch Messung mit Dehnungs-, Kraft- oder Drucksensoren. Nach der Zyklenzählung mit dem Rainflow-Algorithmus stehen damit aus Sicht der Belastungen alle notwendigen Informationen zur Verfügung. Hinsichtlich der Ermüdungsbewertung nach Miner sind nun die Materialeigenschaften (Wöhler-Kurve) hinzuzuziehen. Die Festlegung der Ermüdungseigenschaften des verwendeten Materials sind häufig schwierig, oft stehen nur wenige oder keine Materialkennwerte zur Verfügung oder müssen abgeschätzt werden.

Praktisch helfen folgende Überlegungen:

- Beschränkung auf einen relativen Vergleich von baugleichen Komponenten
- Heranziehen von Literaturdaten und entsprechende Umrechnung in eine approximierte Wöhler-Kurve

Relativer Vergleich von Komponenten

Insbesondere für relative Vergleiche zwischen baugleichen Komponenten, z. B. Maschinenteilen im Serienmaschinenbau, eignen sich generische Wöhler-Kurven mit definierten Steigungen. Die absolute Schadenssumme ist hier nicht aussagekräftig, jedoch können Aussagen wie „Maschinenteil B ist um 40 % stärker geschädigt als Maschinenteil A“ abgeleitet werden und damit Wartungsentscheidungen unterstützt werden.

Nutzung von Literaturquellen

Für die Ermittlung von absoluten Schadenswerten werden üblicherweise Wöhler-Kurven aus realen Materialproben in Prüfmaschinen ermittelt. Es stehen aber mit dem Eurocode 3 und der FKM-Richtlinie ebenfalls Quellen für Materialparameter von Stahl zur Verfügung, aus denen sich Wöhler-Kurven berechnen lassen.

Im Folgenden werden immer Schwingbreiten angegeben und nicht wie in einigen Quellen auch üblich Peak-Amplituden. Die Schwingbreite ist doppelt so groß wie die Peak-Amplitude.

Die **FKM Richtlinie** (Forschungskuratorium Maschinenbau, Verbund im VDMA) gibt konservative Materialkennwerte für verschiedene Werkstoffsorten und -zustände an. Ein Beispiel auf der FKM Richtlinie (weitere siehe Abschnitt Anlagen [▶ 62]).

Sorte	UTS [Schwing-breite, MPa]	Schwingbreite bei Dauerfestig-keit [MPa]	k	NC	SRI [MPa]
S185	620	280	5	1E+06	4438

Übersetzt in die Materialparameter für die Funktion F_CM_CalculateWoehlerCurve [▶ 288] ergeben sich:

```
fSRI : LREAL := 4438;
fUTS : LREAL := 620;
bUseUTSCorrection : BOOL := TRUE;
fK1 : LREAL := 5;
fK2 : LREAL := 0;
nNC1 : ULINT := 18790;
nNC2 : ULINT := 1E+06;
```

`fSRI` und `fUTS` können also direkt übernommen werden. Die Steigung beträgt wie angegeben 5 und bezieht sich auf den Bereich zwischen der Transition von der UTS Korrektur bis zur Dauerfestigkeitsgrenze. Die Dauerfestigkeitsgrenze ist bei NC definiert, welche als `nNC2` eingetragen wird. Die Steigung im Dauerfestigkeitsbereich ist natürlich `fK2 = 0`. Nur als Eingabe in die Funktion F_CM_CalculateWoehlerCurve [▶ 288] muss zusätzlich der Transitionspunkt von der UTS Korrektur zum Bereich mit Steigung 5 über die Gleichung

$$NC1 = \left(\frac{UTS}{SRI}\right)^{-k}$$

berechnet werden.

Der **Eurocode 3** (BS EN1993-1-9: 2005) definiert für Stahl in unterschiedlichen Konstruktionsdetails, insbesondere gerollte/extrudierte Produkte, Schweiß- und Schraubverbindungen und unterschiedliche Geometrien, wie T-Träger, unterschiedliche Wöhler-Kurven. Alle Wöhler-Kurven haben diese drei Abschnittsdefinitionen gemeinsam:

- $N \leq 5*10^6$ mit k = 3
- $5*10^6 \leq N \leq 10^8$ mit k = 5
- $N \geq 10^8$ mit k=0 (dauerfest)

Ein Beispiel aus dem Eurocode 3 (weitere siehe Abschnitt Anlagen [▶ 62]).

Konstruktionsdetail	Δσc [Schwingbreite, MPa]	k1	k2	N bei Δσc	NC1	ND (Dauerfestigkeitsgrenze)	SRI bei N = 1 [MPa]	ΔσNC1 bei NC1 [MPa]	ΔσI bei ND [MPa]
160	160	3	5	2E+06	5E+06	1E+08	20159	118	65

Als Konstruktionsdetail 160 ist im Eurocode ein gerolltes oder extrudiertes Produkt definiert, ausgeführt als Platte, Hohlzylinder oder ähnliche Formen.

Übersetzt in die Materialparameter aus dem bereitgestellten Beispielcode [▶ 370] ergeben sich:

```
fSRI : LREAL := 2580;
fUTS : LREAL := 20159;
bUseUTSCorrection : BOOL := TRUE;
fK1 : LREAL := 5;
fK2 : LREAL := 0;
nNC1 : ULINT := 5E+06;
nNC2 : ULINT := 1E+08;
```

Die Parametrierung wurde hier so umgerechnet, dass sich die Forderung nach drei definierten Zonen mit Steigung 3, 5, und 0 bei Verwendung der Funktion F_CM_CalculateWoehlerCurve [▶ 288] ergibt. Die UTS-Korrektur wird hier so genutzt, dass sich im ersten Abschnitt eine Steigung von 3 ergibt. Aus diesem Grund ist `fUTS` auch größer als `fSRI`. Der Wert für `fSRI` ergibt sich mit k_2 = 5 zu

$$\mathrm{fSRI} = \Delta\sigma_{NC1} NC1^{(1/k_2)}$$

`fUTS` wird dann zu SRI aus dem Eurocode gesetzt. Damit ergibt sich im ersten Abschnitt eine Steigung von 3. Durch Setzen von `fK1` = k2 und `fK2` = 0 ergeben sich die weiteren zwei Sektionen.

Zusammenfassung

Die **Eingangsgrößen** einer Lebensdaueranalyse lassen sich in drei Gruppen einteilen:

- Materialeigenschaften beschreiben die Ermüdungsfestigkeit des Werkstoffes, meist als Wöhler-Kurve, ergänzt um Oberflächenbeschaffenheit und -bearbeitung.
- Geometriefaktoren dienen zur Anpassung der Materialparameter an lokale Geometrieeigenschaften, die die Ermüdungslebensdauer beeinflussen (z. B. Kerben oder scharfe Kanten).
- Lasten sind auf das Bauteil einwirkende Signale, die zu einer Beanspruchung führen. Sie werden häufig mit Dehnungsmessstreifen, Kraft- oder Drehmomentaufnehmern erfasst.

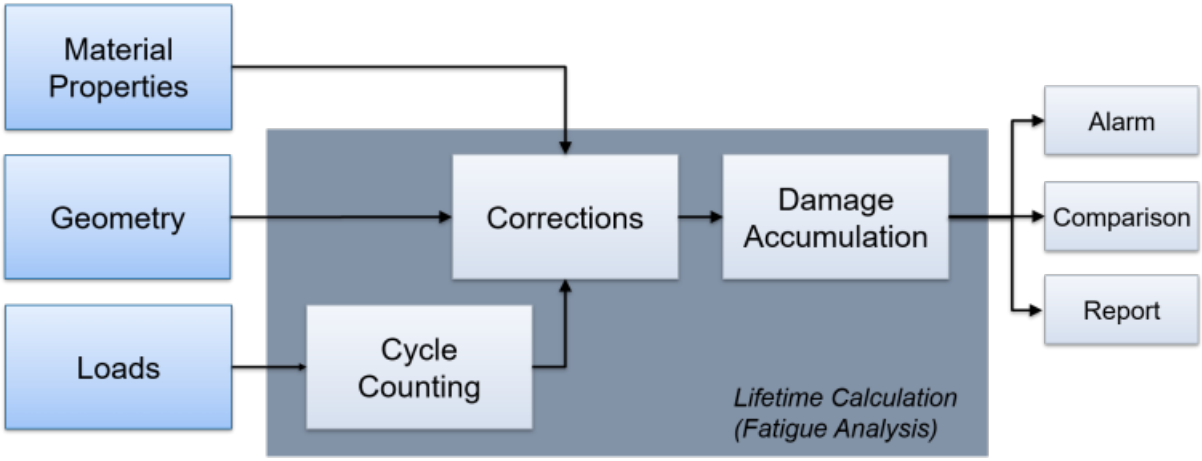
Im **Analyseteil** befinden sich je nach Komplexitätsgrad folgende Komponenten:

- Korrekturen der Materialparameter und Verwendung von Geometriefaktoren
- Zyklenzählung zur Überführung der Zeitsignale in Klassen (Mittelspannung, Schwingbreite) für die Verarbeitung in der Schadenakkumulation
- Mittelspannungskorrekturen, um die Reduzierung der Ermüdungslebensdauer durch Mittelspannungseffekte zu berücksichtigen

- Schadenakkumulation für die Summierung der einzelnen Schädigungsanteile aus der Zyklenmatrix, Materialparametern und Korrekturfaktoren

Als Ergebnis stehen u. a. die Schadenssumme für weitergehende Vergleiche, beispielsweise zu baugleichen Komponenten, oder als Grundlage für Warnungs-/Alarmmeldungen zur Verfügung. Der monoton steigende Verlauf der Schadenssumme eignet sich gut für eine Extrapolation zu definierten Grenzen, um kritisch beanspruchte Komponenten zu identifizieren und Wartungseinsätze besser planen zu können.

Dieses Verfahren eignet sich nicht nur für die klassische Ermüdungsanalyse zur Berechnung einer absoluten Schädigungszahl, sondern auch für relative Vergleiche der Schädigungsintensität: Kraft- oder Drucksignale, Temperaturzyklen oder andere schwingende Signalformen können verwendet werden, um statische Verfahren (Extremwerte, Crest-Faktor, Kurtosis usw.) um eine aussagekräftige Information zu erweitern.



Die drei in obiger Grafik zentralen Elemente „Cycle Counting“, „Corrections“ und „Damage Accumulation“ werden in der Function TwinCAT 3 Condition Monitoring umgesetzt mit FB_CMA_RainflowCounting [▶ 211], FB_CMA_MeanStressCorrection [▶ 176] und FB_CMA_MinersRule [▶ 179].

Anlage: Eurocode und FKM Richtline

Eurocode 3 (EN 1993-1-9:2005 - Design of steel structures)

Konstrukti-onsdetail	Δσc [Schwing-breite, MPa]	k1	k2	N bei Δσc	NC1	ND (Dauer-festigkeits-grenze)	SRI bei N = 1 [MPa]	ΔσNC1 bei NC1 [MPa]	Δσl bei ND [MPa]
160	160	3	5	2E+06	5E+06	1E+08	20159	118	65
140	140	3	5	2E+06	5E+06	1E+08	17639	103	57
125	125	3	5	2E+06	5E+06	1E+08	15749	92	51
112	112	3	5	2E+06	5E+06	1E+08	14111	83	45
100	100	3	5	2E+06	5E+06	1E+08	12599	74	40
90	90	3	5	2E+06	5E+06	1E+08	11339	66	36
80	80	3	5	2E+06	5E+06	1E+08	10079	59	32
71	71	3	5	2E+06	5E+06	1E+08	8945	52	29
63	63	3	5	2E+06	5E+06	1E+08	7938	46	25
56	56	3	5	2E+06	5E+06	1E+08	7056	41	23
50	50	3	5	2E+06	5E+06	1E+08	6300	37	20
45	45	3	5	2E+06	5E+06	1E+08	5670	33	18
40	40	3	5	2E+06	5E+06	1E+08	5040	29	16
36	36	3	5	2E+06	5E+06	1E+08	4536	27	15

FKM Richtlinie

Die FKM Richtlinie gibt konservative Materialkennwerte für verschiedene Werkstoffsorten und -zustände an. Die folgenden Werte für unlegierten Baustahl, unlegierten Feinkornbaustahl, Vergütungsstahl und Gusseisen gelten für eine Überlebenswahrscheinlichkeit von 97,5 %.

Unlegierter Baustahl

Sorte	UTS [Schwing-breite, MPa]	Schwingbreite bei Dauerfestig-keit [MPa]	k	NC	SRI [MPa]
S185	620	280	5	1E+06	4438
S235	720	320	5	1E+06	5072
S275	860	390	5	1E+06	6181
S355	1020	460	5	1E+06	7291
S450	1100	500	5	1E+06	7924
E295	980	440	5	1E+06	6974
E335	1180	530	5	1E+06	8400
E360	1380	620	5	1E+06	9826

Vergütungsstahl, vergüteter Zustand

Sorte	UTS [Schwing-breite, MPa]	Schwingbreite bei Dauerfestig-keit [MPa]	k	NC	SRI [MPa]
C22 E/R	1000	450	5	1E+06	7132
C35 E/R/-	1260	570	5	1E+06	9034
C40 E/R/-	1300	590	5	1E+06	9351
C45 E/R/-	1400	630	5	1E+06	9985
C50 E/R/-	1500	680	5	1E+06	10777
C55 E/R/-	1600	720	5	1E+06	11411
C60 E/R/-	1700	770	5	1E+06	12204
28Mn6	1600	720	5	1E+06	11411
38Cr2	1600	720	5	1E+06	11411
46Cr2	1800	810	5	1E+06	12838
34Cr4	1800	810	5	1E+06	12838
37Cr4	1900	860	5	1E+06	13630
41Cr4	2000	900	5	1E+06	14264
25CrMo4	1800	810	5	1E+06	12838
34CrMo4	200	450	5	1E+06	7132
42CrMo4	2200	990	5	1E+06	15690
50CrMo4	2200	990	5	1E+06	15690
34CrNiMo8	2400	1080	5	1E+06	17117
30CrNiMo8	2500	1130	5	1E+06	17909
35NiCr6	1760	790	5	1E+06	12521
36NiCrMo16	2500	1130	5	1E+06	17909
39NiCrMo3	1960	880	5	1E+06	13947
30NiCrMo16-6	2160	970	5	1E+06	15373
51CrV4	2200	990	5	1E+06	15690

2.3 Literaturhinweise

Im Folgenden werden Hinweise - keine Empfehlungen - auf Sekundärliteratur gegeben. Die Liste ist nicht allumfassend, sondern zeigt nur eine kleine Teilmenge der themenbezogenen Literatur auf.

Digitale Signalverarbeitung, Fourier-Analyse, Fensterung (Deutsch)

- A.V. Oppenheim, R.W. Schafer, J.R. Buck: Zeitdiskrete Signalverarbeitung. Pearson Studium, 2004. ISBN 3-8273-7077-9
- K.-D. Kammeyer, K. Kroschel: Digitale Signalverarbeitung – Filterung und Spektralanalyse mit MATLAB-Übungen. Teubner, 2002. ISBN 3-519-46122-6

Discrete-Time signal processing, Fourier-analysis, windowing (English)

- A.V. Oppenheim, R.W. Schafer, J.R. Buck: Discrete-Time Signal Processing. Pearson Education, 2009. ISBN 987-0131988422
- J.G. Proakis, D.K. Manolakis: Digital Signal Processing. Pearson Education, 2013. ISBN 978-0131988422

Zustandsüberwachung (Deutsch)

- J. Kolerus, J. Wassermann: Zustandsüberwachung von Maschinen. Expert Verlag, 2008. ISBN: 978-3-8169-2597-2
- DIN ISO 10816, Mechanische Schwingungen – Bewertung der Schwingungen von Maschinen durch Messung an nicht-rotierenden Teilen (vorher VDI-Richtlinie 2056). Die Norm besteht aus mehreren Bestandteilen
  - DIN ISO 10816-3 bezieht sich auf industrielle Maschinen mit einer Nennleistung über 15 kW und Nenndrehzahlen zwischen 120 U/min und 15000 U/min bei Messung am Aufstellungsort.
  - DIN ISO 10816-7 bezieht sich auf Kreiselpumpen für den industriellen Einsatz
  - DIN ISO 10816-21 Windenergieanlagen mit horizontaler Drehachse und Getriebe beziehen
- DIN ISO 7919, Mechanische Schwingungen - Bewertung der Schwingungen von Maschinen durch Messungen an rotierenden Wellen. Die Norm besteht aus mehreren Teilen
  - DIN ISO 7919-3 bezieht sich auf Gekuppelte industrielle Maschinen
  - DIN ISO 7919-2 bezieht sich auf Stationäre Dampfturbinen und Generatoren über 50 MW mit Nenn-Betriebsdrehzahlen von 1500 min^{-1}, 1800 min^{-1}, 3000 min^{-1} und 3600 min^{-1}
- DIN ISO 20816-1, Mechanische Schwingungen – Messung und Bewertung der Schwingungen von Maschinen. Zusammenfassung von DIN ISO 7919-1 und DIN ISO 10816-1.
- DIN ISO 13373-1, Zustandsüberwachung und -diagnostik von Maschinen - Schwingungs-Zustandsüberwachung - Teil 1: Allgemeine Anleitungen
- DIN ISO 13373-2, Zustandsüberwachung und -diagnostik von Maschinen - Schwingungs-Zustandsüberwachung - Teil 2: Verarbeitung, Analyse und Darstellung von Schwingungsmesswerten
- DIN ISO 17359, Zustandsüberwachung und -diagnostik von Maschinen - Allgemeine Anleitungen

Condition Monitoring (English)

- R.B. Randall: Vibration-based Condition Monitoring. Wiley, 2011. ISBN: 978-0-470-7485-8
- ISO 10816, Mechanical vibration -- Evaluation of machine vibration by measurements on non-rotating parts.
  - ISO 10816-3 Industrial machines with nominal power above 15 kW and nominal speeds between 120 U/min and 15000 U/min when measured in situ.
  - ISO 10816-7 Rotodynamic pumps for industrial applications, including measurements on rotating shafts
  - DIN ISO 10816-21 Horizontal axis wind turbines with gearbox
- ISO 7919, Mechanical vibration -- Evaluation of machine vibration by measurements on rotating shafts.
  - ISO 7919-3 Coupled industrial machines
  - ISO 7919-2 Land-based steam turbines and generators in excess of 50 MW with normal operating speeds of 1 500 r/min, 1 800 r/min, 3 000 r/min and 3 600 r/min

- ISO 13373-1, Condition monitoring and diagnostics of machines - Vibration condition monitoring -Part 1: General procedures
- ISO 13373-2, Condition monitoring and diagnostics of machines - Vibration condition monitoring - Part 2: Processing, analysis and presentation of vibration data
- ISO 17359:2011, Condition monitoring and diagnostics of machines - General guidelines

3 Installation

3.1 Systemvoraussetzungen

Der folgende Artikel beschreibt die Mindestsystemvoraussetzungen für das Condition Monitoring Produkt für Engineering und/oder Runtime Systeme. Die Condition Monitoring Installation muss auf dem Engineering und dem Runtime System durchgeführt werden.

Versionsabhängiger Funktionsumfang

Der volle, beschriebene Funktionsumfang (siehe Abschnitt SPS API) ist nur mit der Installation der aktuellen Versionen der Treiber und SPS Bibliotheken gewährleistet. Die Verwendung vorangegangener Versionen ist mit eingeschränktem Funktionsumfang möglich. Eine detaillierte Übersicht der mit Version CM3.2 erweiterten Funktionalität ist im Abschnitt Kompatibilität [▶ 66] zu finden.

In der Übersicht [▶ 10] ist ferner eine vom Produktlevel abhängige Auflistung der Funktionalitäten der Bibliothek zu finden.

Engineering

Ein Engineering-System beschreibt einen Rechner der für die Entwicklung von Programm-Code genutzt wird und keinen Programm-Code ausführt. Ein Engineering-System muss die folgenden Voraussetzungen erfüllen:

- TwinCAT 3.1 XAE (Engineering Installation) Build 4022.25 oder höher*

Runtime

Ein Runtime-System beschreibt einen Industrie- oder Embedded-PC auf dem Programm-Code ausgeführt wird. Ein Runtime-System muss die folgenden Voraussetzungen erfüllen:

- TwinCAT 3.1 XAR (Runtime Installation) Build 4022.25 oder höher*
- Betriebssysteme: Win 7, Win 10, Windows Embedded Standard 7
- Eine Lizenz für TC1200 PLC und für TF360x Condition Monitoring

Für Testzwecke kann eine 7-Tage Trial-Lizenz wiederholbar aktiviert werden

Engineering und Runtime auf dem gleichen System

Für den Fall, dass auf einem System Engineering und Runtime genutzt werden sollen, müssen folgende Systemvoraussetzungen erfüllt sein:

- TwinCAT 3.1 XAE (Engineering Installation) Build 4022.25 oder höher*
- Eine Lizenz für TC1200 PLC und für TF360x Condition Monitoring

Für Testzwecke kann eine 7-Tage Trial-Lizenz wiederholbar aktiviert werden

*: Nutzung mit eingeschränktem Funktionsumfang möglich ab TwinCAT 3.1 Build 4018.

3.2 Kompatibilität

Die TwinCAT Condition Monitoring Bibliothek ist seit vielen Jahren am Markt. Dabei wurde die CM-Version 3.1.x in vielen Applikationen erfolgreich eingesetzt. Um den neusten Anforderungen und Möglichkeiten im Bereich der Algorithmik und TwinCAT gerecht zu werden ist eine neue Version 3.2.x verfügbar. Bei der

Entwicklung haben wir höchstes Augenmerk auf die Kompatibilität zu bestehenden Applikationen gelegt. So ist es möglich, dass Sie in sehr wenigen und speziellen Fällen Ihre bestehende Applikation für die Nutzung der Version 3.2.x anpassen müssen.

- Die in den InitPars-Strukturen verwendeten `Enums` sind `externalTypes` und können nur ohne Bibliotheks-Namespace verwendet werden.
- Die Verwendung von CM `Enum`-Werten, in durch den Anwender implementierten Bibliotheken, darf nicht ohne Typ-Qualifier erfolgen.

Übersicht über freigegebene Version der Condition Monitoring Bibliothek

TcCM 3.1.x	3.1.16	3.1.17	3.1.18	
TcCM 3.2.x				3.2.20

Änderungen mit Version CM 3.2 (CM Setup Version)

Die im Weiteren erläuterten Änderungen/Erweiterungen der Condition Monitoring Bibliothek setzen folgende Mindestversionen der SPS Bibliotheken und Treiber voraus:

Tc3_CM	Tc3_CM_Base	Tc3_MultiArray	TcCM.sys	TcMultiArray.sys
2.0.30.0	2.1.18.0	2.0.14.0	3.4.17.0	3.4.17.0

Grundlegende Änderungen

- Mehrkanalfähigkeit der Algorithmen, d.h. sequenzielle Verarbeitung mehrerer Kanäle mittels einer einzelnen Instanz. Dies vereinfacht die Implementierung von SPS Applikationen.
- Erweiterung der Auswahl an Fensterfunktionen um `eCM_BartlettWindow`, `eCM_KaiserWindow`, `eCM_FlatTopWindow` sowie die Möglichkeit der freien Einstellung der Überlappung bei Verwendung der Welch-Methode. Siehe hierzu den Abschnitt Fensterfunktionen [▶ 20].
- Nutzung des TC3 EventLoggers zur Fehler- und Informationsausgabe. Siehe hierzu den Abschnitt Online View [▶ 85].
- Ergebnisse an den meisten Funktionsbausteinen sind nun direkt im Tc3 Scope View zu betrachten. Siehe hierzu den Abschnitt Online View [▶ 85].
- Übergreifende Steigerung der Performance und Verbesserung der Numerik.

Anwendungsorientierte Algorithmen

- Schwingungsbeurteilung [▶ 35] in Anlehnung an ISO 10816-3 zur vibrationsbasierten Klassifizierung des Maschinenzustandes mit dem Funktionsbaustein FB_CMA_VibrationAssessment [▶ 258].

Erweiterungen bestehender Funktionsbausteine

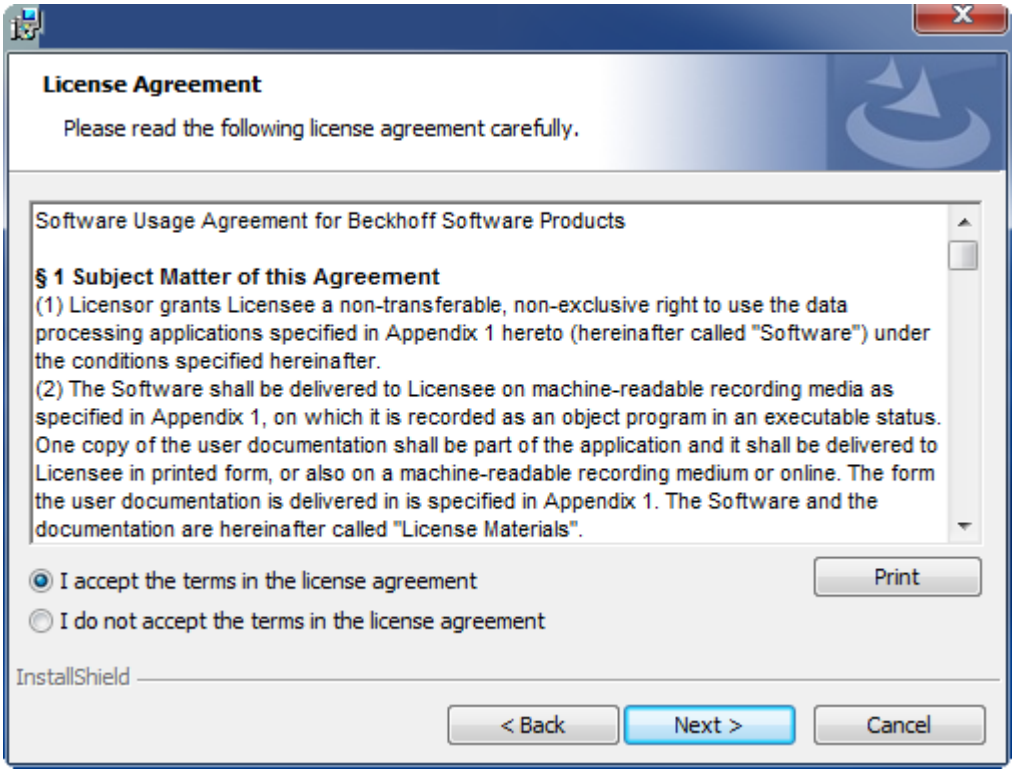
- FB_CMA_Source [▶ 245]: Vereinfachte Methoden für die Einspeisung von ein- und mehrkanaligen Eingangsdaten. Der Funktionsbaustein bietet zudem die Möglichkeit einen Reset der vollständigen Analysekette auszulösen.
- FB_CMA_Sink [▶ 238]: Methode für Ausgabe von 3D Daten, benötigt für die Bausteine FB_CMA_ArgSort [▶ 93] und FB_CMA_IntegratedRMS [▶ 167], sowie vereinfachte Varianten der Methoden für die Ausgabe von Ergebnissen.
- FB_CMA_IntegratedRMS [▶ 167]: Berechnung (optional) integrierter RMS Werte auf konfigurierbaren Frequenzbändern.
- FB_CMA_MomentCoefficients [▶ 183]: Möglichkeit der Berechnung des Exzesses in zwei Varianten: Ist `bExcessKurtosis := TRUE` wird die Kurtosis um drei reduziert.

3.3 Installation

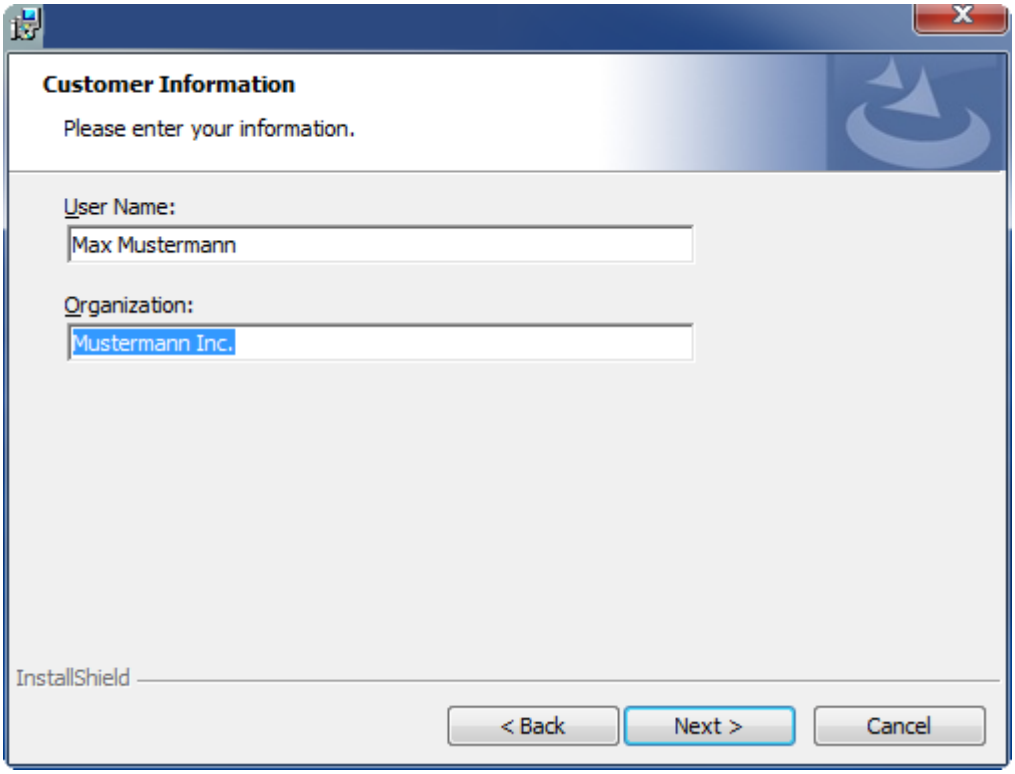
Nachfolgend wird beschrieben, wie die TwinCAT 3 Function für Windows-basierte Betriebssysteme installiert wird.

✓ Die Setup-Datei der TwinCAT 3 Function wurde von der Beckhoff-Homepage heruntergeladen.

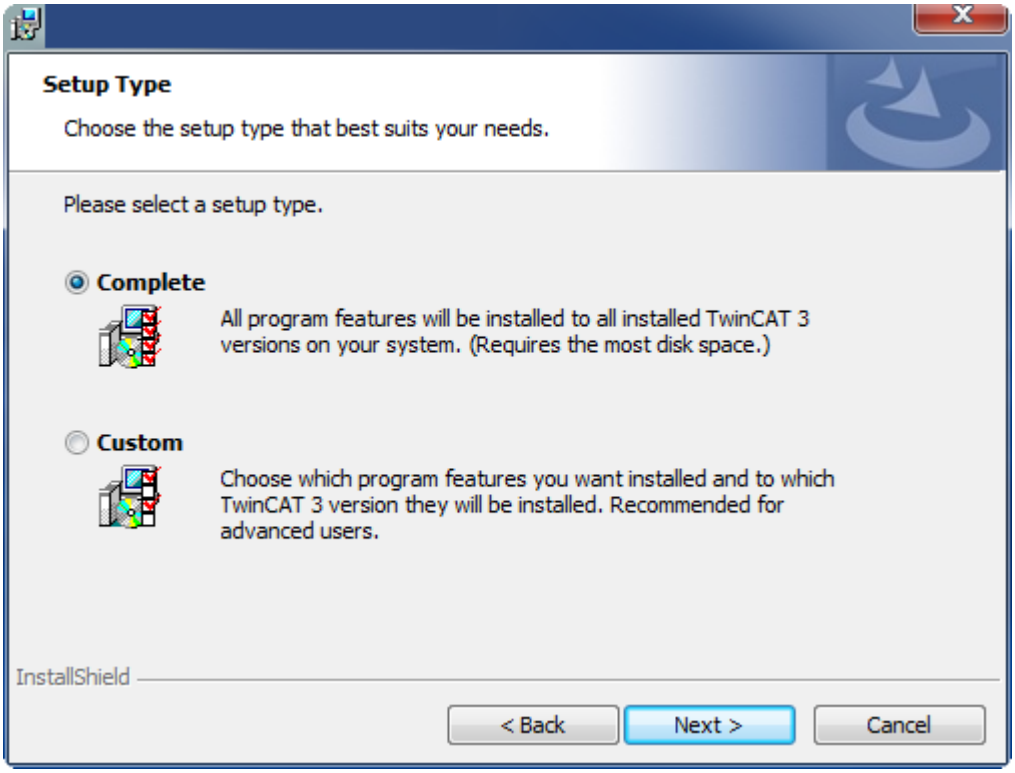
1. Führen Sie die Setup-Datei als Administrator aus. Wählen Sie dazu im Kontextmenü der Datei den Befehl **Als Administrator ausführen**.
   ⇨ Der Installationsdialog öffnet sich.
2. Akzeptieren Sie die Endbenutzerbedingungen und klicken Sie auf **Next**.



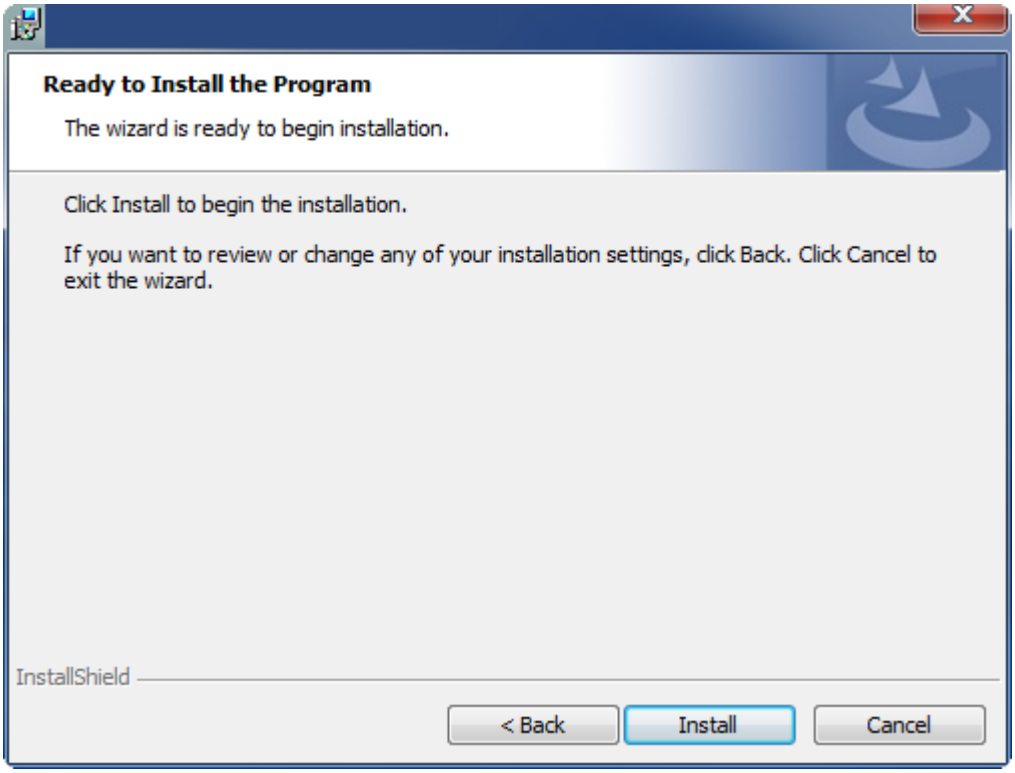
3. Geben Sie Ihre Benutzerdaten ein.



4. Wenn Sie die TwinCAT 3 Function vollständig installieren möchten, wählen Sie **Complete** als Installationstyp. Wenn Sie die Komponenten der TwinCAT 3 Function separat installieren möchten, wählen Sie **Custom**.

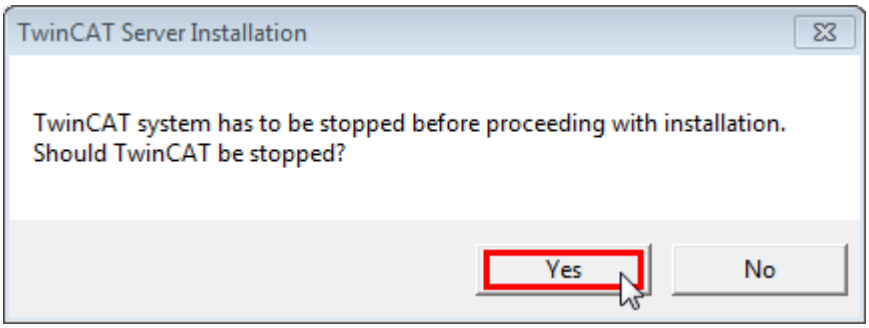


5. Wählen Sie **Next** und anschließend **Install**, um die Installation zu beginnen.

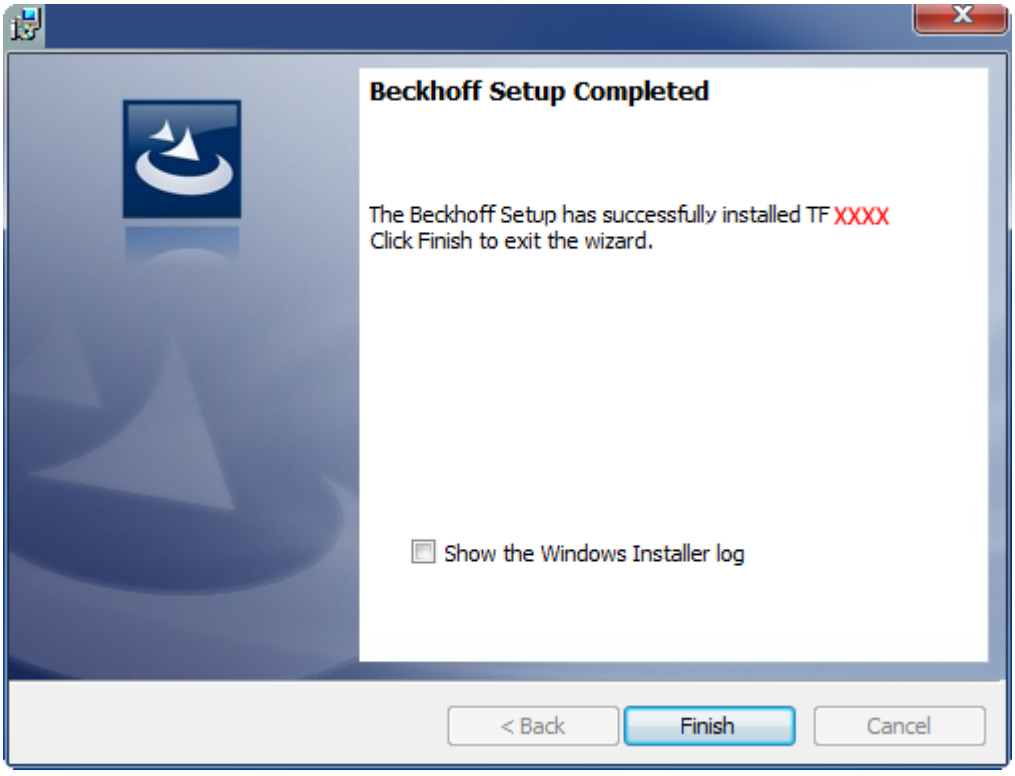


⇨ Ein Dialog weist Sie darauf hin, dass das TwinCAT-System für die weitere Installation gestoppt werden muss.

6. Bestätigen Sie den Dialog mit **Yes.**



7. Wählen Sie **Finish**, um das Setup zu beenden.



⇨ Die TwinCAT 3 Function wurde erfolgreich installiert.

3.4 Installation TwinCAT 3.1. Build 4026

TwinCAT Package Manager: Installation (TwinCAT 3.1 Build 4026)

Eine ausführliche Anleitung zur Installation von Produkten finden Sie im Kapitel Workloads installieren in der Installationsanleitung TwinCAT 3.1 Build 4026.

Installieren Sie den folgenden Workload, um das Produkt nutzen zu können:

TwinCAT Package Manager UI: TF3600 | TwinCAT 3 Condition Monitoring

TwinCAT Package Manager CLI: tcpkg install TF3600.ConditionMonitoring.XAE

3.5 Lizenzierung

Die TwinCAT 3 Function ist als Vollversion oder als 7-Tage-Testversion freischaltbar. Beide Lizenztypen sind über die TwinCAT-3-Entwicklungsumgebung (XAE) aktivierbar.

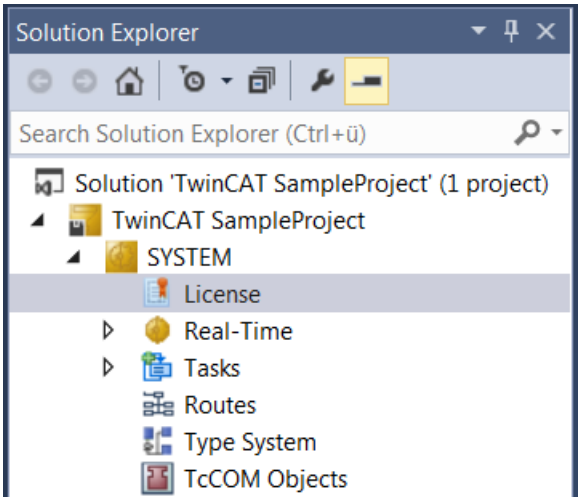
Lizenzierung der Vollversion einer TwinCAT 3 Function

Die Beschreibung der Lizenzierung einer Vollversion finden Sie im Beckhoff Information System in der Dokumentation „TwinCAT-3-Lizenzierung“.

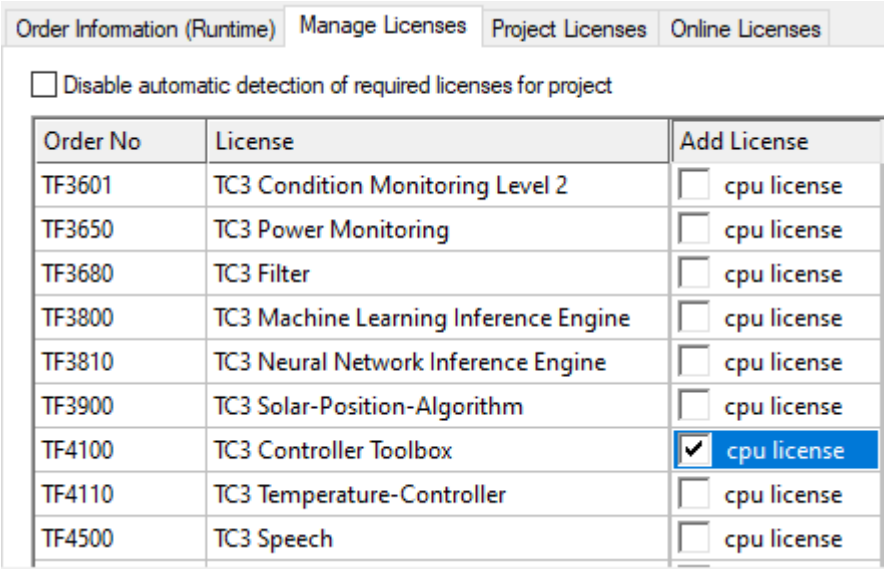
Lizenzierung der 7-Tage-Testversion einer TwinCAT 3 Function

Eine 7-Tage-Testversion kann nicht für einen TwinCAT-3-Lizenz-Dongle freigeschaltet werden.

1. Starten Sie die TwinCAT-3-Entwicklungsumgebung (XAE).
2. Öffnen Sie ein bestehendes TwinCAT-3-Projekt oder legen Sie ein neues Projekt an.
3. Wenn Sie die Lizenz für ein Remote-Gerät aktivieren wollen, stellen Sie das gewünschte Zielsystem ein. Wählen Sie dazu in der Symbolleiste in der Drop-down-Liste **Choose Target System** das Zielsystem aus.
   ⇨ Die Lizenzierungseinstellungen beziehen sich immer auf das eingestellte Zielsystem. Mit der Aktivierung des Projekts auf dem Zielsystem werden automatisch auch die zugehörigen TwinCAT-3-Lizenzen auf dieses System kopiert.
4. Klicken Sie im **Solution Explorer** im Teilbaum **SYSTEM** doppelt auf **License.**

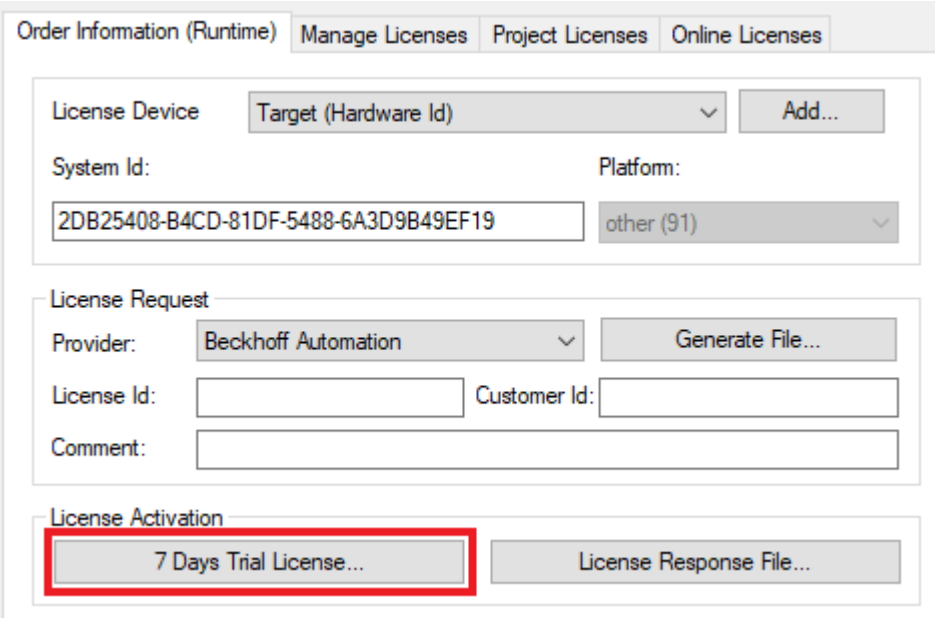


   ⇨ Der TwinCAT-3-Lizenzmanager öffnet sich.
5. Öffnen Sie die Registerkarte **Manage Licenses.** Aktivieren Sie in der Spalte **Add License** das Auswahlkästchen für die Lizenz, die Sie Ihrem Projekt hinzufügen möchten (z. B. „TF4100 TC3 Controller Toolbox“).

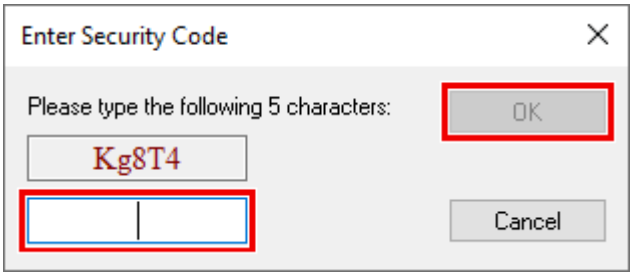


Order No	License	Add License
TF3601	TC3 Condition Monitoring Level 2	cpu license
TF3650	TC3 Power Monitoring	cpu license
TF3680	TC3 Filter	cpu license
TF3800	TC3 Machine Learning Inference Engine	cpu license
TF3810	TC3 Neural Network Inference Engine	cpu license
TF3900	TC3 Solar-Position-Algorithm	cpu license
TF4100	TC3 Controller Toolbox	cpu license
TF4110	TC3 Temperature-Controller	cpu license
TF4500	TC3 Speech	cpu license

6. Öffnen Sie die Registerkarte **Order Information (Runtime).**
   ⇨ In der tabellarischen Übersicht der Lizenzen wird die zuvor ausgewählte Lizenz mit dem Status „missing“ angezeigt**.**
7. Klicken Sie auf **7 Days Trial License...**, um die 7-Tage-Testlizenz zu aktivieren.



   ⇨ Es öffnet sich ein Dialog, der Sie auffordert, den im Dialog angezeigten Sicherheitscode einzugeben.



8. Geben Sie den Code genauso ein, wie er angezeigt wird, und bestätigen Sie ihn.
9. Bestätigen Sie den nachfolgenden Dialog, der Sie auf die erfolgreiche Aktivierung hinweist.
   ⇨ In der tabellarischen Übersicht der Lizenzen gibt der Lizenzstatus nun das Ablaufdatum der Lizenz an.
10. Starten Sie das TwinCAT-System neu.

⇨ Die 7-Tage-Testversion ist freigeschaltet.

4 Technische Einführung

4.1 Speicherverwaltung

Die Condition Monitoring Bibliothek nutzt intern TcCOM Objekte welche von den installierten Treibern zur Verfügung gestellt werden. Die Instanzen werden dynamisch im TwinCAT AMS Routerspeicher angelegt.

Notwendigkeit dynamischer Speicherverwaltung

Alle Speicheranforderungen und Initialisierungen werden innerhalb der Initialisierungsphase durchgeführt. Da die Anzahl der Elemente der Eingangsdaten und der internen Strukturen von der Konfiguration der jeweiligen Bausteine abhängen, wird der Speicherplatz für diese grundsätzlich dynamisch angelegt. Dies geschieht bei Verwendung der SPS Condition Monitoring Library automatisch.

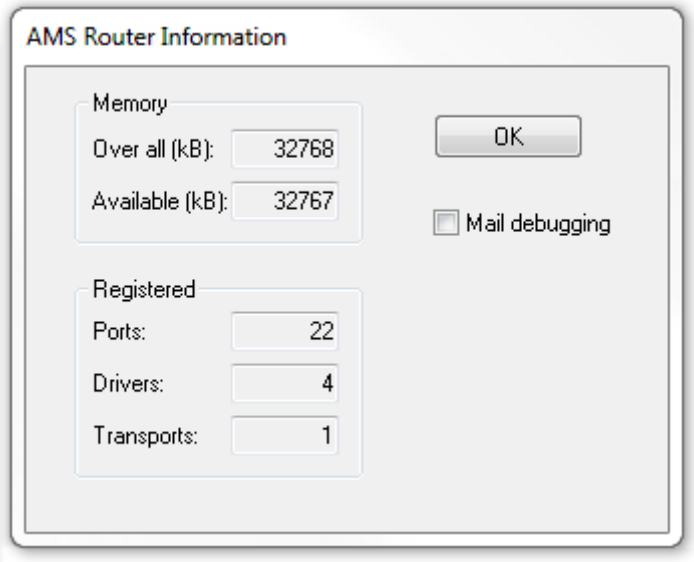
Da alle Speicherbelegungen bei der Initialisierung erfolgen und somit die Initialisierung von Bausteinen unter Umständen eine relativ große Speichermenge beansprucht, kann sie an dieser Stelle - nicht jedoch später- auch aufgrund Speichermangels fehlschlagen.
Der belegte Speicher wird wieder freigegeben, wenn das Objekt gelöscht wird.

TwinCAT Routerspeicher für dynamisch erzeugte Objekte

Die Puffer, welche die TwinCAT 3 Condition Monitoring Library reserviert, werden bei der Initialisierung von Funktionsbausteinen im TwinCAT AMS Routerspeicher angelegt, so dass sie für eine Ausführung unter Echtzeitbedingungen zur Verfügung stehen. Bestimmte Funktionen wie z.B. hochauflösende Histogramme und Quantile, aber auch die Berechnung von Spektren mit sehr hoher Auflösung, erfordern wesentlich mehr Routerspeicher als herkömmliche Steuerungsprogramme. Deswegen ist es möglicherweise notwendig, den Routerspeicher zu vergrößern.

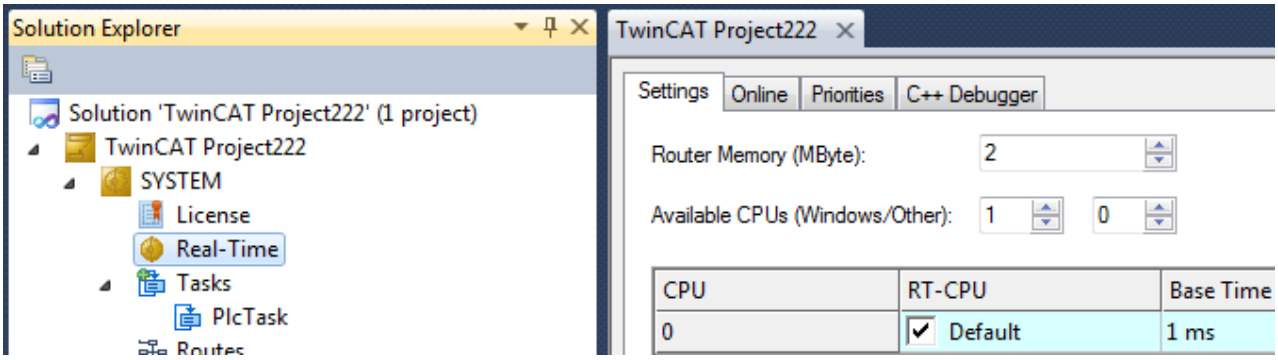
Routerspeicher anpassen

Die Standardgröße ist 32 MB (2 MB bis TwinCAT 3.1.4016). Die aktuelle Einstellung kann mit dem Dialogfenster AMS Router Information angezeigt werden.



Um den Routerspeicher zu vergrößern wird in der TwinCAT Konfiguration unter System\ Real-Time\ Settings ein Wert in MB eingetragen und die Konfiguration aktiviert.

Die Anpassung des Routerspeichers erforderte bis TwinCAT 3.1.4022.4 einen Reboot des Zielgerätes!



4.2 Task Einstellung

Applikationen mit mehreren Echtzeit-Tasks

Eine Condition Monitoring Analysekette wird gebildet aus der Datensammlung, meist mehreren Algorithmen und der Ergebnisbereitstellung. Die Weiterverarbeitung der Ergebnisse sowie programmatische Reaktionen auf diese sind applikationsabhängig.

Da der Umfang der Eingangsdaten, z.B. die Länge von Eingangsvektoren, stark von der jeweiligen Anwendung abhängt, benötigt Software zur Signalverarbeitung Arrays mit unterschiedlicher Länge und verschiedenen Elementtypen. Die TwinCAT 3 Condition Monitoring Library verwendet daher durchgängig eine flexible Datenstruktur für numerische Arrays. Diese erlaubt es, numerische Daten effizient blockweise zu speichern, zu übertragen und auszuwerten. Sie kann neben ein- auch mehrdimensionale Daten darstellen.

Die Condition Monitoring Algorithmen sind je nach Konfiguration sehr rechenaufwändig. Die Algorithmen werden deshalb bevorzugt in eine separate Task ausgelagert. Die Analysekette erstreckt sich in dem Fall über mehrere Tasks. Die damit verbundenen Schwierigkeiten des synchronen Datenaustausches und der Threadsicherheit werden von den Bibliotheksbausteinen intern gekapselt, so dass flexibel handzuhabende Analyseketten ermöglicht werden.

Mehr Informationen zum Datenaustausch finden sich im Kapitel „Parallelverarbeitung [▶ 77]“.

Tipp: Selbstverständlich kann das Programm auch als Anwendung einer einzelnen Task implementiert werden. Dies wird empfohlen, wenn die benötigten Algorithmen, in Abhängigkeit der CPU und der Taskzykluszeit, schnell genug abgearbeitet werden können.

Taskzykluszeiten

Die Analyseschritte und die entsprechenden Puffergrößen stellen eine Bedingung für die Taskzykluszeit dar. Die Berechnung muss häufig genug ausgeführt werden, um alle Eingangsdaten verarbeiten zu können.

Beispiel: Die Datensammlung füllt Puffer, deren Größe bei der Deklaration auf 1600 Elemente festgelegt wurde. Mit einer Oversampling-Rate von 10 werden 160 Zyklen für die Füllung eines Puffers benötigt. Wenn die Signalsammlung von einer 1 ms Task ausgelöst wird, muss die Berechnung von einer Task mit einer Zykluszeit unter 160 ms ausgelöst werden.

Es wird empfohlen die Berechnungszykluszeit kleiner einzustellen, um eine schnellere Reaktion zu realisieren (mindestens Faktor 0,5). Auf der anderen Seite hängt die kleinstmögliche Berechnungszykluszeit von der Komplexität der zu berechnenden Algorithmen und von der Leistungsfähigkeit der genutzten CPU ab.

Richtwert für obere Schranke der Berechnungszykluszeit

Berechnungszyklenzeit < 0,5 * Signalsammlungszyklenzeit * Puffergröße / Oversampling-Rate

Die meisten Algorithmenbausteine (Spectrum, Cepstrum, …) umfassen rechenintensive mathematische Operationen. Sie sollten in einem Taskkontext mit ausreichender Zykluszeit aufgerufen werden. Die erforderliche Ausführungszeit hängt zudem von der Hardwareplattform ab. Die obige Gleichung stellt einen oberen Richtwert für die Berechnungszyklenzeit dar. Für die Abschätzung eines unteren Richtwerts wird

bspw. für jeden Funktionsbaustein ein Profiler zur Verfügung gestellt, der im Verlauf der Online-Überwachung aktiviert werden kann. Sie finden diesen Profiler in der Instanz des Funktionsbausteins unter `fbImplementation → fbExecutionTimeMonitoring`. Durch **manuelles** Setzen von `bMeasureMaxExecTime` aktivieren Sie den Profiler. Es soll auf interne Variablen eines Funktionsbausteins wie gewohnt programmatisch nicht zugegriffen werden.

RealFFT_Sample.RealFFT.MAIN_CM

Expression	Type	Value
nCntResults	ULINT	278
stInitParsResultBuffer	ST_MA_MultiArr...	
fbImplementation	FB_CMA_Imple...	
nDiscardFirstResults	USINT	0
fbExecTimeMonitoring	FB_CMA_ExecTi...	
bMeasureMaxExecTime	BOOL	TRUE
fbProfiler	Profiler	
tMaxExecTime	LTIME	LTIME#656us600ns
tMaxElapsedTimeout	LTIME	LTIME#15us800ns
bValueInitialized	BOOL	TRUE
bObjectsInitialized	BOOL	TRUE
bStreamInitialized	BOOL	TRUE
bStreamsAllocated	BOOL	TRUE

Die angezeigten Werte sind maximale Ausführungszeiten. Die Taskeinstellungen sollten eine kleine Reserve für mögliche Kombinationen von Parametern und Eingangswerten, die zu längeren Ausführungszeiten führen könnten, vorsehen.

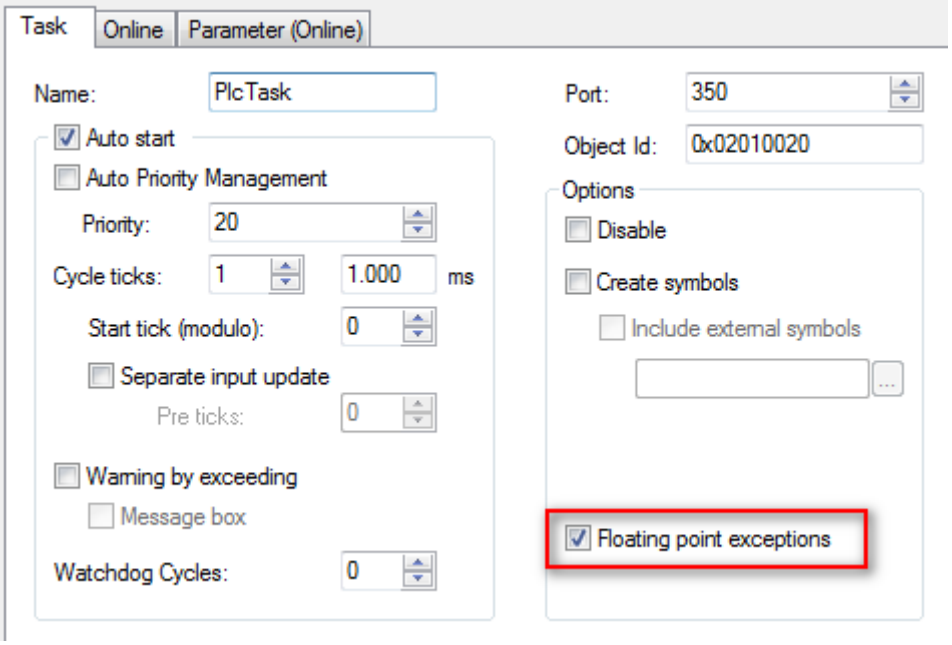
Ausnahmen zu den obigen Betrachtungen bilden manche Statistik-Bausteine (Quantile, Histogramme, …). Diese Bausteine fügen in der Regel zunächst lediglich Daten für mehrere Taskzyklen dem internen Speicher hinzu. Nur die anschließende Berechnung (nach *N* Zyklen Daten sammeln) benötigt Zeit. Die entsprechende Taskzykluszeit kann dem einfachen Aufruf ohne Berechnung angepasst werden. Dies führt zwar zur Überschreitung der Zykluszeit im Falle eines Aufrufs mit Berechnung, sorgt aber für schnelle Reaktionszeiten. Dies ist ein Sonderfall für die SPS-Programmierung. Normalerweise sollte eine Taskzykluszeit niemals überschritten werden.

Zykluszeit beachten

Die Zykluszeit von Tasks, die ausschließlich Condition Monitoring Algorithmen aufrufen, können auf solch eine Weise angepasst werden, dass die Zykluszeit selten überschritten wird. Programmbausteine, die von dieser Task aufgerufen werden, sollten keinen anderen Programmcode enthalten! Und selbstverständlich muss die Priorität dieser langsamen Tasks niedriger sein, als die von anderen Tasks.

Floating Point Exceptions

Diese Exceptions können getrennt für jede Task deaktiviert werden. Sie sind standardmäßig aktiviert.



Einige Algorithmenaufrufe können ein NaN (Not a Number) als Ergebnis ausgeben. Wenn NaNs in der Anwendung abgearbeitet werden sollen, müssen die FP Exceptions für diese Task deaktiviert werden. Anschließend müssen Sie sich vergewissern, dass der gesamte Programmcode und alle verwendeten Funktionen NaNs verarbeiten können.
Weitere Hinweise bezüglich des Umgangs mit NaN Werten finden Sie im getrennten Kapitel „NaN Werte [▶ 76]“.

⚠ VORSICHT

Ausführungsstopp

Floating Point Exceptions sind standardmäßig aktiv. Vergleiche mit NaN (Not a Number) können eine solche Exception verursachen, die zu einem Ausführungsstopp führt, und möglicherweise einen Maschinenschaden verursachen. Es wird strengstens empfohlen, das Ergebnis für NaN zu überprüfen, bevor es verarbeitet wird. (siehe Kapitel „NaN Werte“)

4.3 NaN-Werte

In einigen Fällen ist die Fehlerbehandlung durch Fehlercodes [▶ 386] nicht die beste Wahl, insbesondere wenn Operationen aufgrund besonderer, grundsätzlich jedoch möglicher Eingangsdaten undefinierte Werte liefern, oder wenn Werte aus der Verarbeitung ausgeklammert werden sollen.

Die Norm IEEE 754 definiert für diese Zwecke symbolische Werte der Kategorie NaN (Not a Number) . In der TwinCAT 3 Condition Monitoring Library werden diese in den folgenden Situationen erzeugt bzw. berücksichtigt:

- Sind für eine statistische Auswertung noch nicht genügend gültige Werte vorhanden, wird das Ergebnis mit NaN ausgegeben.
- Sollen bei einer statistischen Auswertung bestimmte Werte ausgenommen werden, so wird dies erreicht indem der Baustein NaN-Werte am Eingang ignoriert.
- Falls der Eingangsvektor der Frequenzanalyse einer Zeitreihe einen oder mehrere NaN-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Diese Eigenschaft kann genutzt werden, um Ergebnisse als nicht definiert zu kennzeichnen für den Fall, dass eine Lücke im Eingangssignal zu Sprüngen in der Zeitreihe führt. Denn es ist nicht möglich korrekte Spektren anhand zerstückelter Zeitreihen zu berechnen.

Wenn Bausteine NaN-Werte erzeugen können, ist dies in der Beschreibung des jeweiligen Bausteins vermerkt.

Zu den wesentlichen Eigenschaften von NaN-Werten zählen die folgenden Punkte:

- Alle arithmetischen Operationen, die NaN als Eingangsdaten verwenden, liefern wiederum NaN als Ergebnis.
- Alle relationalen Operatoren =, !=, > < >= <= liefern stets den Wert False, wenn mindestens einer der Operanden NaN ist.
- Die Standard-C-Funktionen `isnan()` bzw. `_isnan()` oder die SPS Funktionen LrealIsNaN() und RealIsNaN() (Tc2_Utilities Bibliothek) liefern den Wert True, wenn das Argument den Wert NaN hat.
- Der Ausdruck `isnan(a)` ist äquivalent dem Ausdruck `!(a == a)` bzw. `NOT(a = a)`.

Die Tatsache, dass NaN-Werte sich bei der Verwendung in weiteren Berechnungen fortpflanzen, hat den Vorteil, dass ungültige Werte nicht übersehen werden können.

⚠ VORSICHT

Fehlfunktionen der Software

NaN-Werte dürfen in SPS-Bibliotheken, insbesondere als Stellwerte in Funktionen für Motion Control und zur Antriebssteuerung, nur verwendet werden, wenn sie ausdrücklich zugelassen sind! Anderenfalls können NaN-Werte zu potenziell gefährlichen Fehlfunktionen der betreffenden Software führen!

⚠ VORSICHT

Floating Point Exceptions

Falls NaNs in der Applikation verwendet und verarbeitet werden sollen, müssen die FP Exceptions ausgeschaltet werden. Andernfalls können Vergleiche mit NaN zu einer Exception führen, welche einen Stopp der Laufzeit und möglichen Maschinenschaden nach sich zieht.

Weitere Erläuterungen zur Möglichkeit die FP Exceptions aus- und anzuschalten finden Sie im Kapitel Task Einstellungen [▶ 74].

4.4 Parallelverarbeitung mit Transfer Tray

Der folgende Abschnitt behandelt die **thread-sichere** und **mehrkernfähige Datenübermittlung**, welche die TwinCAT 3 Condition Monitoring Library bietet.

Asynchrone Kommunikation und parallele Ausführung rechenintensiver Schritte

Condition-Monitoring-Anwendungen erfordern häufig mehrere Megabyte große Datensätze, welche die Anforderungen an Rechenzeit und Rechenleistung erhöhen. Die maximal zulässige Rechenzeit orientiert sich an der Zykluszeit, die bspw. für Antriebssteuerungen niemals überschritten werden darf. Aufgrund dessen werden im Falle von rechenintensiven Algorithmen Multitask-Software-Architekturen für TwinCAT 3 Condition-Monitoring-Anwendungen empfohlen. Siehe Kapitel "Task Einstellungen [▶ 74]".

Idee des Transfer Tray

Hierzu sind thread-sichere Implementierungen der Algorithmen erforderlich. Die TwinCAT 3 Condition Monitoring Library bietet einen sehr effizienten und einfach zu verwendenden Kommunikationsmechanismus, der die typischen Probleme mit Sperren und Entsperren von Daten so weit wie möglich beseitigt. Die Bibliothek bietet einen sehr effizienten Mechanismus zur Parallelverarbeitung von Daten z.B. mit unterschiedlichen Datenraten. Damit können Array-Daten fehlerfrei zwischen mehreren Tasks für exklusiven synchronisierten Zugriff übermittelt werden - unter Verwendung von Warteschlangen (*queues*) mittels des Transfer Tray. Dies ermöglicht auch die Verwendung von Mehrkern-CPUs ohne Synchronisationsprobleme und verhindert schwer zu diagnostizierende Fehler wie Blockaden und Inkonsistenzen, die von nicht synchronisierten Überschreibungen numerischer Daten verursacht werden.

Die Bibliotheksfunktionsbausteine dürfen nicht als globale Instanzen in der Liste der globalen Variablen deklariert werden, weil paralleler Schreibzugriff auf *MultiArray Puffer* (vgl. Abschnitt Umgang mit MultiArray [▶ 79]) und die parallele Ausführung der gleichen Funktionsbausteine ausdrücklich verboten sind.

Beispiel für die Notwendigkeit von Zykluszeit-Transitionen

Unter manchen Umständen ist ein sequentielles Konzept nicht ausreichend. Das ist immer dann der Fall, wenn die Verarbeitung eines Datensatzes mehr Zeit in Anspruch nimmt, als die Zykluszeit einer Steuerungstask zulässt.

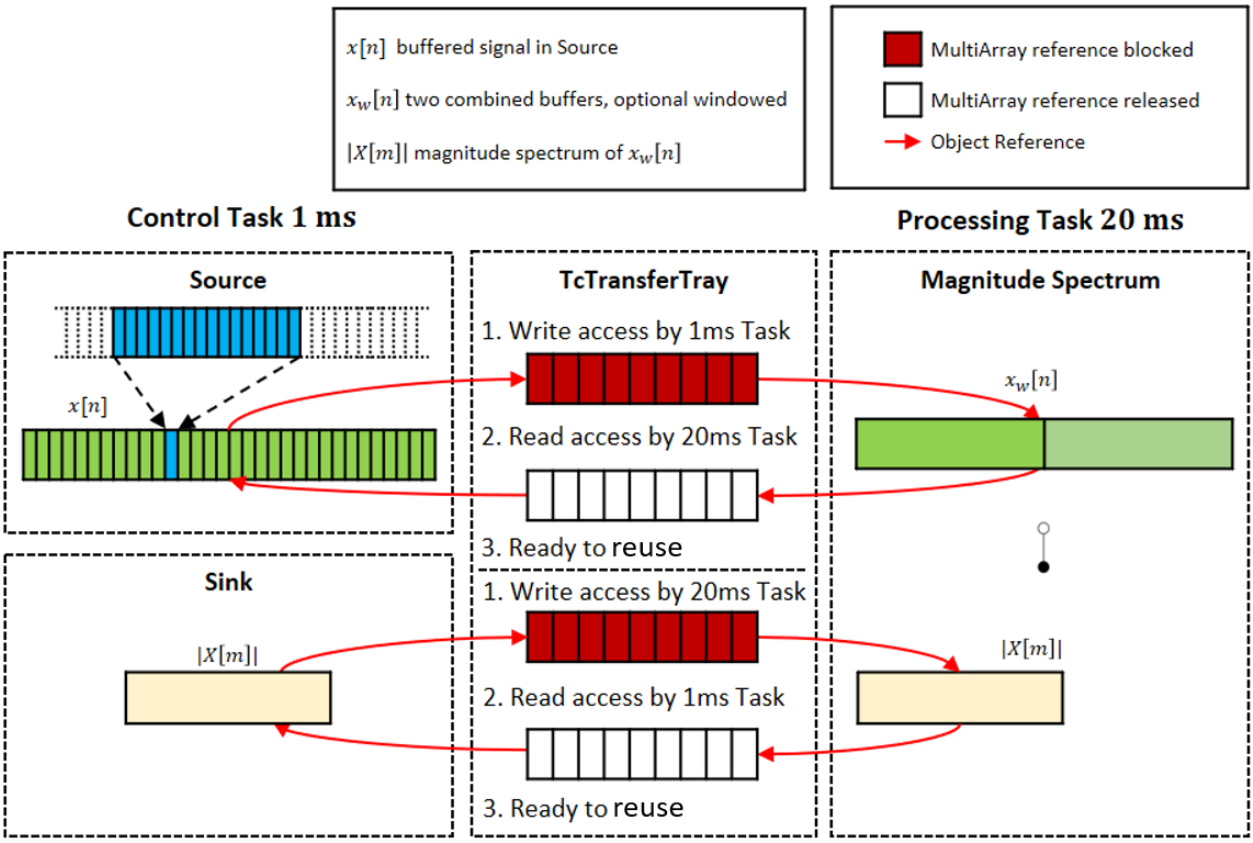
Die Steuerungstask hat z.B. eine Zykluszeit von 1 Millisekunde und ein Oversampling von Daten von 20 Abtastungen pro Zyklus (entspricht einer Abtastrate von 20 kHz). Es wird für die Signalverarbeitung eine Frequenzauflösung von 0,16 Hz gefordert, was z.B. für die Analyse von großen Rollenlagern erforderlich sein könnte, damit zwischen Mängeln am Innen- und Außenlaufring, die mit nur geringfügig verschiedenen Geschwindigkeiten laufen, unterschieden werden kann.

Die Beziehung zwischen FFT-Länge N, Frequenzauflösung Δf und Abtastrate f_s ist: $N = f_s / \Delta f$ (zur Vereinfachung sei hier ein Rechteckfenster angenommen). Es ergibt sich eine FFT-Länge von $N = 125000$. Darüber hinaus muss die FFT-Länge N' eine Potenz von zwei sein, daraus folgt mit $\log_2(125000) = 16.93$, dass das Signal der Länge N auf $N' = 2^{17} = 131072$ mit Nullen aufgefüllt wird.

Die erforderliche Rechenzeit hängt von der Leistung der CPU ab, aber die Berechnung in der Steuerungstask ist definitiv nicht möglich. Die erforderliche Eingangsdatenmenge entspricht einem Signalabschnitt von mehreren Sekunden, so dass die Berechnung infolgedessen nur selten durchzuführen ist.

Lösungskonzept mit dem Transfer Tray

Die von der Condition Monitoring Library bereitgestellte hochleistungsfähige Lösung wird in untenstehender Abbildung gezeigt. Die Steuerungstask (Control Task) sammelt Daten in "Paketen“ von 20 Samples über die Oversampling-Klemme (in Abbildung blau dargestellt). Diese werden in einen Puffer gespeichert, dessen Größe der Länge des Eingangspuffers des Magnituden Spektrum Funktionsbausteins entspricht (125000 / 20 = 6250, in Abbildung grün dargestellt). Sobald der Puffer gefüllt ist, also nach 3125 Zyklen der Steuerungstask, wird dieser, genauer gesagt, dessen Objektreferenz, mit Hilfe eines asynchronen Kommunikationsmechanismus an eine zweite Task (Processing Task) übergeben (FIFO Prinzip), die mit 20 Millisekunden eine viel größere Zykluszeit aufweist. Nach der in Task Einstellung [▶ 74] beschriebenen Daumenregel ist eine Zykluszeit für die berechnende Task von maximal 1562,5 ms zulässig. Mit den 20 ms wird diese Forderung deutlich erfüllt.



Dieser Kommunikationsmechanismus garantiert mit Hilfe von hardware-gesicherten, sogenannten *atomaren Operationen*, dass nur eine der Tasks Zugriff auf den entsprechenden Puffer (im folgenden auch *MultiArray* gennant) zur gleichen Zeit hat. Dies ist vergleichbar mit einer Übergabeschublade (*transfer tray*) am Bankschalter, die sicherstellt, dass entweder der Kunde oder der Kassierer auf deren Inhalt zugreifen kann.

Reaktionslatenz

Für die Warteschlangen gilt das Prinzip des FIFO. Deswegen und wegen der asynchronen Kommunikation ist das Ergebnis nicht sofort verfügbar. Reaktionen mit variabler Latenz sind möglich.

Das Rechenergebnis (das Magnituden Spektrum) wird über eine weitere Warteschlange mit dem gleichen Kommunikationsmechanismus an die Steuerungstask zurückgegeben, die dieses dann weiter auswerten kann, natürlich ist auch die Kommunikation an eine andere dritte Task sowie die Bereitstellung des Ergebnisses in der berechnenden Task selbst möglich.

Im Allgemeinen sind der berechnenden Task, im Vergleich zu Motion-Anwendungen, keine harten Echtzeit-Bedingungen auferlegt und sie kann demzufolge mit einer geringeren Priorität ausgeführt werden, als die Steuerungstask. Das Taskmanagement des TwinCAT 3 Systems sorgt dafür, dass die Task mit der höchsten Priorität immer zuerst ausgeführt wird, so dass diese echtzeitlichen Bedingungen selbst mit komplexen Berechnungen erfüllt werden können.

Das vorgestellte Konzept kann sowohl auf Einkern-, als auch auf Mehrkern-CPUs verwendet werden. Die Verteilung auf vielen Kernen, ohne die zentralen Sperren, die Engpässe verursachen, ist möglich.

Timeout

Die internen Kommunikationsbefehle zum Transfer Tray können in seltenen Fällen fehlschlagen, z.B. in Abhängigkeit der Eigenschaften der Hardware. Es befindet sich z.B. ein leerer Puffer in der Warteschlange, der trotzdem nicht herausgenommen werden kann, weil eine andere Task gerade auf ihn zugreift. Ein synchrones Timeout ist spezifiziert und es kann infolge ein Timeout-Fehler auftreten. Das Programm muss also stets auf den möglichen Fehlerzustand, dass ein für die Stetigkeit der Signaldaten erforderlicher Puffer nicht verfügbar ist, gefasst sein. Folgefehler wie Datenüberlauf und Diskontinuitäten von analysierten Zeitserien müssen auf konsistente Weise aufgearbeitet werden. So lange die Eingangssignaldaten einer Analysekette fehlerfrei gesammelt werden können, treten keine Diskontinuitäten auf. Wenn ein einzelnes Timeout bei einem folgenden Algorithmenbaustein auftrat oder für den folgenden Algorithmenbaustein kein Ergebnis-MultiArray-Puffer verfügbar war, gehen weder Eingangsdaten noch Ergebnisdaten verloren. Sie werden beim nächsten Aufruf übergeben.

Funktionsweise des Transfer Tray

Der Transfer Tray selber wird mit Hilfe eines von der Tc3_CM Bibliothek bereitgestellten internen Funktionsbausteins dargestellt. Dieser Baustein wird mit Anfangsparametern initialisiert, die in der globalen Strukturinstanz definiert sind.

Die typische Verwendung von Warteschlangen geschieht so, dass Puffer von genau einer Task der Warteschlange mit einer festen Datenstromkennung hinzugefügt werden und dass diese Puffer im Gegenzug von einer bestimmten anderen Task zwecks Verarbeitung entfernt werden. Diese Puffer werden dann über eine weitere Warteschlange mit einer anderen Datenflusskennung zurückgeschickt und erneut verwendet. Allerdings stellt es auch kein Problem dar, wenn mehrere Tasks Lese- oder Schreibzugriff auf die gleichen Warteschlangen haben, wenn z.B. statistische Daten analysiert werden.

Die MultiArray Puffer

Um Daten über das Transfer Tray von einer Task zur nächsten zu kommunizieren werden sogenannte MultiArray Puffer verwendet. Diese werden im Kapitel "Umgang mit MultiArray [▶ 79]" erläutert.

4.5 Umgang mit MultiArray

Ein MultiArray ist ein **mehrdimensionaler Datenpuffer**, welcher in der Condition Monitoring Library in Kombination mit dem Transfer Tray genutzt wird. Er ermöglicht einer Anwendung mehrdimensionale Daten problemlos zwischen mehreren SPS-Tasks auszutauschen. Während der Kommunikation zwischen den Tasks wird kein Speicher kopiert, sondern es werden lediglich Verweise auf die Datenpuffer übergeben, wodurch die Kommunikation extrem effizient wird. Die Kommunikation erfordert lediglich ein sehr geringes Overhead mit Ausführungszeiten im Mikrosekundenbereich.

Der MultiArray-Kommunikationsring

Das Füllen (Schreiben von Inhalt) und Senden (Weitergabe der Zugriffsrechte) von MultiArrays für Eingangs- oder Ergebnisdatenströme haben zur Folge, dass ständig „freie“ MultiArrays gefordert werden. Deswegen werden die ausgewerteten MultiArrays als „leere“ Datenbehälter an diejenige Task zurückgegeben, welche sie auch gefüllt hat. So entsteht ein stetiger Kreislauf von MultiArrays, vgl. Abbildung im Abschnitt Parallelverarbeitung mit Transfer Tray [▶ 77].

Normalerweise sind mindestens drei MultiArrays pro Kreislauf notwendig: Das erste MultiArray „gehört“ der Steuerungstask und ist dabei, mit neuen Daten gefüllt zu werden. Auf das zweite MultiArray greift die Prozesstask zu und verarbeitet ihn. Ein drittes MultiArray muss in Reserve gehalten werden, so dass es verfügbar ist, falls die Steuerungstask das aktuelle MultiArray gefüllt hat, jedoch verbleibende Oversampling-Daten in genau diesem Zyklus auch noch in ein nächstes MultiArray geschrieben werden müssen. Deswegen ist die Mindestanzahl gleich drei.

Anzahl von MultiArrays

Zur Sicherheit wird als Worst-Case-Abschätzung eine Anzahl von vier MultiArrays pro Kreislauf empfohlen. Greift mehr als ein Algorithmus auf die Daten eines MultiArrays zu wird empfohlen, für jeden weiteren zugreifenden Algorithmus jeweils ein zusätzliches MultiArray bereitzustellen.

Die Anzahl von bereitgestellten MultiArrays wird über die Eingangsparameter `nResultBuffers` der Funktionsbausteine der Condition Monitoring Library eingestellt, default-Wert ist entsprechend 4.

Anzahl von MultiArrays im Kommunikationsring

Mehr als vier MultiArrays sind nur dann erforderlich, wenn die Ergebnispuffer (= MultiArrays) direkt von mehreren Algorithmen abgearbeitet werden sollen. Also nehmen mehr als zwei Analysebausteine im Kommunikationsring für diese Ergebnisse teil. Es wird empfohlen, die Anzahl Ergebnispuffer mit jedem zusätzlichen Baustein um eins zu erhöhen. Die Anzahl der in einem asynchronen Kommunikationsring verwendeten MultiArray Puffer kann in jedem Analysefunktionsbaustein konfiguriert werden.

Diese zusätzlichen Puffer werden intern erzeugt und verwaltet. Sie erfordern eine bestimmte Größe an zusätzlichem Speicher im AMS Router.

Grundsätzlich kann die Dimension eines MultiArrays getrennt im Hinblick auf Länge, Größe und sogar Datentyp konfiguriert werden. Die Parameter definieren zusammen die sogenannte Shape (Form) des MultiArrays für dessen gesamten Lebenszyklus.

Es ist zu beachten, dass die interne Struktur des MultiArrays automatisch verwaltet wird und keinerlei Programmierung erfordert. Die Lebensdauer des MultiArrays ist die gleiche, wie diejenige der Anwendung, d.h. ab SPS Start bis zum SPS Stopp, wobei die MultiArrays von der einen zur anderen Task mit Hilfe des sogenannten Transfer Tray übergeben werden.

Das Konzept ist sehr flexibel. Das Ändern und Neuverteilen der Berechnung auf andere Tasks und/oder CPUs ist einfach und unkompliziert.

Konfiguration von MultiArrays

MultiArrays werden mit der ST_MA_MultiArray_InitPars [▶ 322] Struktur konfiguriert. Diese ist Teil der Tc3_MultiArray Bibliothek, welche mit dem Condition Monitoring Setup installiert wird.

Beispielkonfiguration eines MultiArray:

```
cInitSource : ST_MA_MultiArray_InitPars:= ( eTypeCode := eMA_TypeCode_LREAL,
                          nDims := 2,
                          aDimSizes := [cChannels, cBufferLength]);
```

Wird das MultiArray mit dem `FB_CMA_Source` Funktionsbaustein verwendet, dann wird eine konfigurierte MultiArray Instanz (oder mehrere) von der Source-Instanz `fbSource` gefordert. Das oben beschriebene MultiArray hat 2 Dimensionen (`nDims = 2`, erlaubt ist ebenfalls `nDims = 1`), wobei die Größe der Dimensionen mit `aDimSizes` beschrieben wird. Entsprechend ist das beschriebene MultiArray von der Dimension `cChannelsxcBufferLength` mit dem Datentyp LREAL für jedes Element.
Beispiel für die Nutzung von MultiArrays mit dem `FB_CMA_Source`:

```
fbSource : FB_CMA_Source := ( stInitPars := cInitSource,
                  nOwnId := eID_Source,
                  aDestIDs := [eID_Rms],
                  nResultBuffers := 4);
```

MultiArrays sind hinsichtlich der Verwaltung des Datenspeichers flexibel. So sind z.B. im obigen Fall die Zeilen und Spalten vollständig austauschbar, und bei korrekter Zuweisung/Identifizierung der Dimensionen (wie im Beispiel unten gezeigt), hat dies keinerlei Einfluss auf die Ergebnisse.

Erweiterte Konfigurationsoptionen

Wie Sie unten im Beispiel sehen, bietet FB_CMA_Source [▶ 245] (oder FB_CMA_Sink [▶ 238], FB_CMA_BufferConverting [▶ 96]) Parameter wie `nWorkDim`, `pStartIndex` oder `nElementsDim`. Diese Parameter können verwendet werden um:

- Ein bestimmtes Segment des MultiArray zu beschreiben/auszulesen
- Ab einer bestimmten Stelle zu schreiben/lesen/kopieren
- Eine bestimmte Anzahl Elemente ab einer bestimmten Stelle zu kopieren

Eine Kombination dieser Parameter garantiert nicht nur Speicheroptimierung, sondern garantiert auch die Selektivität in mehrkanaligen, Multi-Task Anwendungen. Siehe Beispiel unten.

Anwendungsszenario

Dieses Anwendungsszenario hat lediglich innerhalb des TwinCAT Condition Monitoring Anwendungsbereichs Gültigkeit. Wie bereits erwähnt, werden die MultiArrays automatisch verwaltet, aber sie müssen zunächst initialisiert werden. Dies geschieht in der SPS Deklaration mit Hilfe von `ST_MA_MultiArray_InitPars` und wird an die `FB_CMA_Source`-Instanz übergeben.

Jeder Algorithmen-Funktionsbaustein übergibt seine Ergebnisse mit Hilfe der mit stInitPars konfigurierten MultiArrays. Deren Formen werden mit den Initialisierungsparametern (siehe jeweilige Erläuterungen zu den Funktionsbausteinen), mit Ausnahme des `FB_CMA_Sink`, definiert. Es ist zudem möglich nur einen Teil des MultiArray in ein SPS Array zwecks weiterer Bearbeitung oder Auswertung zu kopieren. Dies wird mit Hilfe von `FB_CMA_BufferConverting` erledigt.

Die Funktionsbausteine verfügen über Methoden, mit denen SPS Variablen in MultiArrays geschrieben oder ausgelesen werden können. Weitere Erläuterungen zu den Methoden und deren Parametern finden Sie in den Beschreibungen der Funktionsbausteine.

Der `FB_CMA_Sink` Funktionsbaustein erfordert keinerlei Initialisierung eines MultiArray. Die Shape der von `FB_CMA_Sink` verwendeten MultiArrays wird intern spezifiziert.

Jede Dimension eines MultiArray, genannt Arbeitsdimension (WorkDim), hat einen mit 0 beginnenden Index.

Bei zweidimensionalen MultiArrays wird in der Condition Monitoring Library die Arbeitsdimension 0 normalerweise mit der Anzahl von Kanälen verknüpft (vgl. „Beispielkonfiguration eines MultiArray“ im obenstehenden Text)

Beispiele zur Handhabung von MultiArrays

Für das bessere Verständnis bezüglich der Verwendung eines MultiArray in einer Condition Monitoring Anwendung, betrachten wir folgendes Fallbeispiel.

Es werden, z.B. mit zwei EL3632, drei Signale von einem Beschleunigungssensor mit einem Oversampling-Faktor von 10 aufgenommen. Die Eingangsdaten werden in einem MultiArray mit der Länge 1000 gesammelt und an einen Funktionsbaustein übergeben. In diesem Fall ist es der Baustein für die Berechnung der Moment Coefficients [▶ 183]. Der `FB_CMA_MomentCoefficients` berechnet für jeden Kanal, je nach Konfiguration, verschiedene statistische Kenngrößen der Eingangsdaten. Unser Ziel ist es nun das MultiArray am Ausgang des `FB_CMA_MomentCoefficients` so zu konfigurieren, dass nur ein gewisser Teil des Ergebnisses, zum Beispiel der Mittelwert und die Standardabweichung, ausgegeben werden.

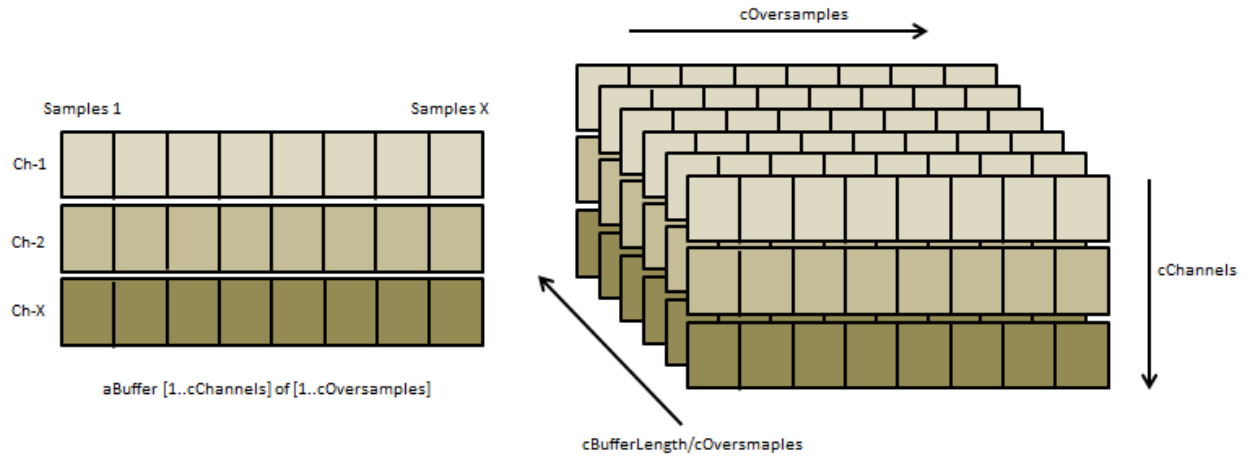
Die Ein- und Ausgangsvariablen werden folgendermaßen deklariert und initialisiert:

```
cInitSource : ST_CM_MultiArray_InitPars := (eTypeCode := eMA_TypeCode_LREAL,
                        nDims := 2,
                        aDimSizes := [3,1000]);
```

```
aBuffer  : ARRAY [1..3] OF ARRAY [1..cOverSamples] OF LREAL;
fbSource : FB_CMA_Source := (stInitPars := cInitSource,
                 nOwnID := eID_Source,
                 aDestIDs := [eID_MomentCoeffs]);

// MultiArray indices begin with 0, not 1!
// aStartIndex := [0,0],[0,1],[0,2],[1,0],[1,1],[1,2],[2,0],...
aStartIndex : ARRAY [1..2] OF UDINT := [0, 1];

// Select channels := 1: one, 2: one and two, 3: one, two and three and so on
// Select moments := 0: count, 1: mean, 2: standard deviation, 3: skew, 4: kurtosis
aMomentCoef : ARRAY [1..3, 1..2] OF LREAL;
```

Wie oben gezeigt, erhält der `fbSource` ein MultiArray mit 2 Dimensionen und soll die Daten von `aBuffer` nach entsprechender Aufpufferung den `FB_CMA_MomentCoefficients` übergeben. In Funktion der Initialisierungsparameter, kann man entweder die Daten speichern:

- Indem die Kanäle über die Zeilen und die Abtastungen über die Spalten gespeichert werden,
- oder, indem die Abtastungen über die Zeilen und die Kanäle über die Spalten gespeichert werden.



Weil das **MultiArray zweidimensional** ist, wird dies mit dem Aufruf der **Input2D() Methode** erledigt.

```
fbSource.Input2D(pDataIn := ADR(aBuffer),
         nDataInSize := SIZEOF(aBuffer),
         eElementType := eMA_TypeCode_LREAL,
         nWorkDim0 := 0, (* aBuffer stores channels across first dim*)
         nWorkDim1 := 1, (* aBuffer stores samples across second dim*)
         pStartIndex := 0,
         nOptionPars := 0 );
```

Gehen wir diesen Aufruf der Methode Schritt für Schritt durch:

- Die lokale SPS Variable `aBuffer` wird als Referenz übergeben.
- Es wird der Datentyp spezifiziert, welcher übergeben wird.
- Die Methode weist hier die erste Arbeitsdimension des MultiArray der ersten Dimension von `aBuffer` (`cChannels`) und die zweite Arbeitsdimension den Abtastwerten (`cOversamples`) zu. Alternativ könnte z. B. die Variable `aBuffer : ARRAY [1.. cOversamples] OF ARRAY [1.. 3] OF LREAL` deklariert sein, und die notwendige Transponation durch `nWorkDim0 =1` und `nWorkDim1 =0` realisiert werden.
- `pStartIndex=0` kopiert hier den gesamten `aBuffer` in das MultiArray, was der Standardeinstellung entspricht. Wie nur Teile eines Arrays kopiert werden, wird anhand des `FB_CMA_Sink` nachfolgend gezeigt.

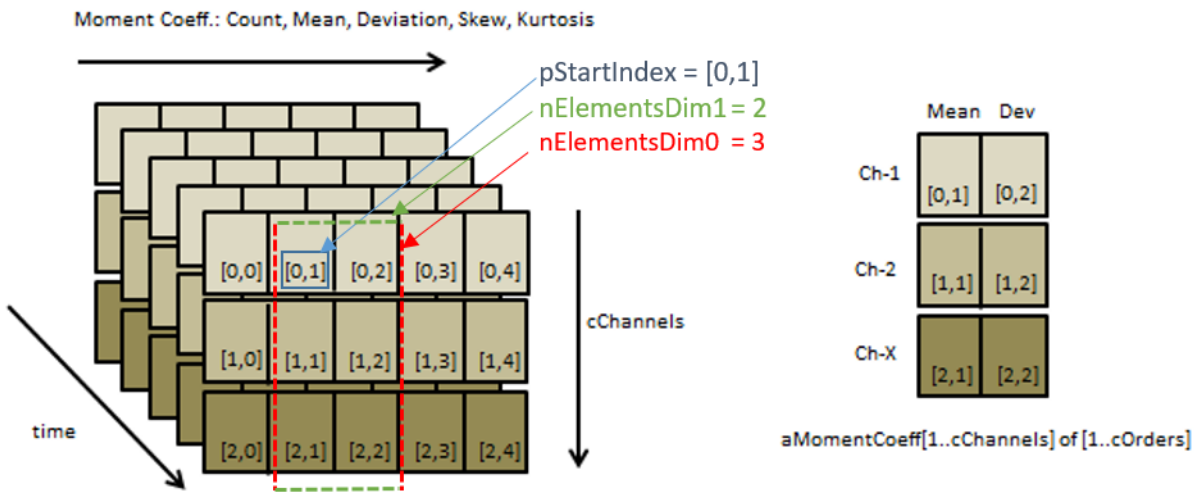
Alle obigen Einstellungen konfigurieren das MultiArray vollständig, um die Kanäle entlang seiner ersten Dimension (Zeilen) und die Abtastwerte entlang seiner zweiten Dimension (Spalten) bis zur Länge `cBufferLength` zu speichern.

Auf ähnliche Weise kann eine `FB_CMA_Sink` Instanz die Inhalte des MultiArray in die lokale SPS-Variable `aMomentCoef` schreiben.

```
fbSink.Output2D(pDataOut := ADR(aMomentCoef),
           nDataOutSize := SIZEOF(aMomentCoef),
           eElementType := E_MA_ElementTypeCode.eMA_TypeCode_LREAL,
           nWorkDim0 := 0,         (* aMomentCoef stores channels across first dim *)
           nWorkDim1 := 1,         (* aMomentCoef stores moments across second dim *)
           nElementsDim0 := 3,     (* aMomentCoef stores all 3 channels *)
           nElementsDim1 := 2,     (* aMomentCoef stores mean and deviation*)
           pStartIndex := ADR(aStartIndex),
           nOptionPars := 0);
```

Gehen wir auch diesen Aufruf der Methode Schritt für Schritt durch:

- Die lokale SPS Variable `aMomentCoef` (in welche nun geschrieben werden soll) wird als Referenz übergeben.
- Es wird der Datentyp spezifiziert.
- Es wird die erste Arbeitsdimension des MultiArray der ersten Arbeitsdimension des Variable `aMomentCoef` zugewiesen, also den Kanälen. Die zweite Dimension wird analog übergeben und entspricht im Beispiel den statistischen Parametern *count, mean, deviation, skew, kurtosis*.
- Mit den Parametern `nElementsDim0` und `nElementsDim1` wird spezifiziert wie viele Elemente des MultiArray in WorkDim0-Richtung und WorkDim1-Richtung kopiert werden sollen. In diesem Fall also 3 Elemente in WorkDim0-Richtung (alle drei Kanäle) und 2 Elemente in WorkDim1-Richtung.
- Der Paramater `pStartIndex` definiert das erste Element in dem 2x3 großen Rechteck, welches kopiert werden soll. Der Paramater ist ein Pointer auf ein 2D-Array (hier `aStartIndex`).



In der dargestellten Konfiguration wird die Output2D() Methode lediglich ein Segment des MultiArrays in die SPS Variable `aMomentCoef` kopieren. Das zu kopierende Segment wird, wie oben näher erläutert, mit den Parametern `nWorkDim0`, `nWorkDim1`, `nElementsDim0`, `nElementsDim1`, und `pStartIndex` konfiguriert.

5 SPS-API

Die TwinCAT3 Condition Monitoring Library bietet Analysemöglichkeiten in einer TwinCAT SPS Anwendung. Lesen Sie bitte hierzu unsere Produktbeschreibung [▶ 10] und die technischen Einführungen, um sich einen Überblick zu verschaffen und wichtiges Hintergrundwissen über das Produkt anzueignen.

Die SPS API setzt sich aus drei **SPS Bibliotheken** zusammen. Diese Bibliotheken müssen in einem Condition Monitoring SPS Projekt eingebunden werden:

- Tc3_CM
- Tc3_CM_Base
- Tc3_MultiArray

Condition Monitoring Analyse

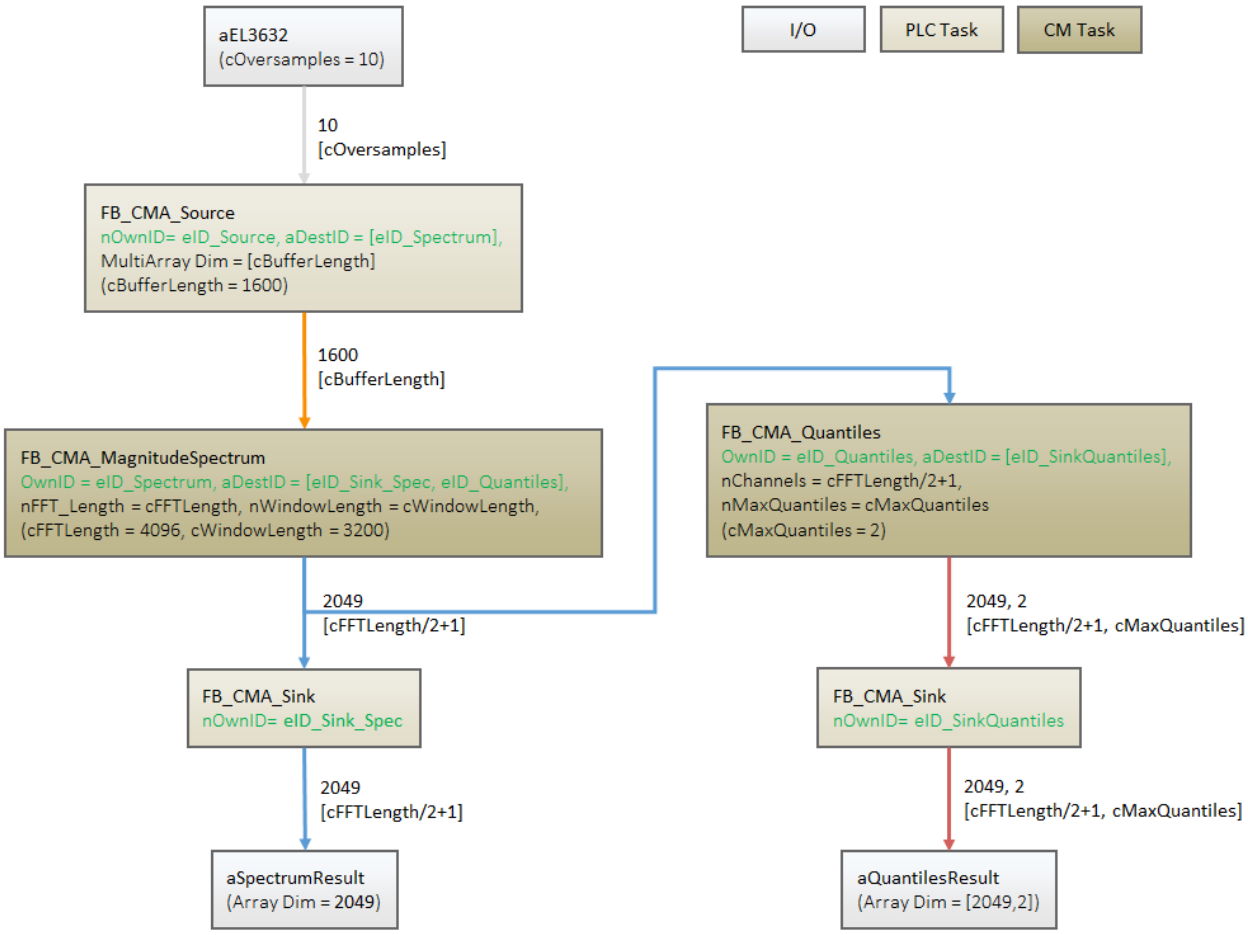
Neben der Programmierung, welche die Aufnahme der Messdaten, die Verarbeitung mittels unterschiedlichster Algorithmen und die Auswertung der Ergebnisse umfasst, nimmt bei jeder Signalverarbeitung eine den Anforderungen entsprechende Analysekette einen großen Stellenwert ein. Deshalb unterstützt die TwinCAT 3 Condition Monitoring Library Sie mit Funktionsbausteinen, welche die Implementierung der geplanten Analysekette nahezu zu reiner Parametrierungsarbeit machen.

Analysekette als Diagramm

Es ist sinnvoll, ein Diagramm (Beispiel siehe unten) bezüglich der Analyseschritte zu erstellen, bevor die Condition-Monitoring-Anwendung programmiert wird!
Es umfasst eine Darstellung von jedem SPS-Funktionsbaustein. In der Regel werden mindestens zwei Tasks verwendet, eine Task für das reguläre Steuerungsprogramm und eine weitere (langsamere und niederpriore) Task für die rechenintensiven Operationen des Condition Monitoring.

Jeder Analysefunktionsbaustein verwendet eine besondere Art für die Kommunikation untereinander. Diese interne Implementierung ermöglicht auch eine Kreuzkommunikation über mehrere Tasks. Intern werden ein TransferTray Objekt und mehrere MultiArrays verwendet (siehe Kapitel Parallelverarbeitung [▶ 77]). Ein Baustein oder dessen Methoden dürfen jedoch nur aus einem Task-Kontext in der Applikation aufgerufen werden!
Die Analysefunktionsbausteine können in verschiedenen Task-Kontexten platziert werden. Die Reihenfolge der Analyseschritte wird mittels Transfer-IDs (grüne Werte in Abbildung unten) zugewiesen. Jeder Baustein erhält seine eigene beliebige ID und die Ziel-ID(s), an welche die Ergebnisse zu senden sind. Am besten werden die Transfer-IDs als Aufzählung/Enumeration definiert.

Die Abbildung unten zeigt vier verschiedene Datenpuffer: grau, orange, blau und rot. Die Form aller entsprechenden Puffer (SPS-Arrays, MultiArrays) und die Algorithmusparameter müssen zu diesen Puffergrößen passen.



Zyklischer Aufruf

Solange die Funktionalität von FB_CMA_Source aufgerufen und Signaldaten an einen Zielbaustein übermittelt werden, müssen alle anderen Bausteine der Analysekette zyklisch aufgerufen werden. Siehe Beschreibung des internen Kommunikationsprinzips in Kapitel Parallelverarbeitung. Wenn nicht alle Zielbausteine während einer bestimmten Phase abgearbeitet werden sollen, ist ihr Aufruf zwar dennoch notwendig, aber es kann die PassInputs() Methode dazu verwendet werden, lediglich die Eingangspuffer zu übergeben, ohne dass Ergebnisse erzeugt werden.

Folgefehler beachten

Ein zyklisch wiederkehrender Fehler bei einem Analysebaustein kann Folgefehler in der Analysekette verursachen.

Online View

Die Funktionsbausteine der Condition Monitoring Bibliothek halten die signifikanten Informationen über den aktuellen Zustand des Bausteins im Online View bereit. Diese können für die Programmierung der Applikation (Initialisierung und Konfiguration der Bausteine, Anpassung der Puffergrößen in der Analysekette, etc.) sowie zur Analyse und Auswertung (Fehleranalyse, grafische Darstellung) verwendet werden.

Die nachfolgende Grafik zeigt den Online View des Funktionsbausteins FB_CMA_MagnitudeSpectrum. Im Folgenden werden einige Knoten und deren Verwendungsmöglichkeiten näher erläutert.

Expression	Type	Value
fbAlgorithm	FB_CMA_MagnitudeSpectrum	
bError	BOOL	FALSE
hrErrorCode	HRESULT	16#00000000
ipErrorMessage	I_TcMessage	16#FFFFBB8691299868
nCntResults	ULINT	0
nOwnID	UDINT	2
aDestIDs	ARRAY [1..cCMA_MaxDest] OF UDINT	
nResultBuffers	UDINT	4
tTransferTimeout	LTIME	LTIME#500us
stInitPars	ST_CM_MagnitudeSpectrum_InitPars	
stInitParsResultBuffer	ST_MA_MultiArray_InitPars	
fbImplementation	FB_CMA_Implementation	
fbBaseAlgorithm	FB_CM_MagnitudeSpectrum	
eTraceLevel	TCEVENTSEVERITY	Critical

Rückgabewerte

Im Fehlerfall wird `bError := TRUE` gesetzt und der Rückgabewert `hrErrorCode` enthält den Fehlercode, siehe hierzu Fehlercodes Übersicht [▶ 386].

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Das Auslesen der Fehlerliste und eine exemplarische Behandlung ist im Beispiel Mehrkanal-Magnitudenspektrum [▶ 332] zu finden.

Fehlerbeschreibung und Information

Unter `ipErrorMessage` befinden sich nähere Informationen zum aktuellen Rückgabewert. Der `sEventText` zeigt die textuelle Beschreibung eines aufgetretenen Fehlers. Diese Nachricht wird, je nach eingestellter TcEventSeverity (Initialparameter `eTraceLevel`, siehe auch Einstellung Param [▶ 324]), über den TC3 EventLogger ausgegeben.

ipErrorMessage	I_TcMessage	16#FFFFBB8691299868
eSeverity	TCEVENTSEVERITY	Error
ipSourceInfo	I_TcSourceInfo	16#FFFFBB86912996B0
nEventId	UDINT	8230
sEventClassName	STRING(255)	'TcMultiArrayEventClass'
sEventText	STRING(255)	'Input data has illegal shape.'

Initialparameter

Die Initialparameter des Bausteins, welche im Zuge der Initialisierungsphase der SPS übernommen werden, werden unter dem Knoten `stInitPars` abgelegt.

Für alle Funktionsbausteine, die über eine einstellbare Überlappung (Parameter `nOverlap`) verfügen, kann der berechnete empfohlene Wert (im Fall `nOverlap := cCM_OverlapRecommended`) nach dem Einloggen der SPS im genannten Knoten ausgelesen werden. Dieser kann anschließend für die Anpassung der Datenpuffer (Input/Output Shape) in der Analysekette verwendet werden. Ein Starten der Applikation ist hierfür nicht erforderlich.

ADS-Zugriff auf Ergebnisdaten

Jeder Analyse-Funktionsbaustein besitzt den Unterknoten `pResultData`, welcher nur im TwinCAT Target Browser sichtbar ist. Damit können die versendeten Ergebnisse (MultiArrays) zwischen verschiedenen Bausteinen einer Analysekette über ADS ausgelesen werden. Dies ermöglicht die einfache grafische Darstellung der Ergebnisse eines Bausteins mittels TwinCAT 3 Scope View ohne die Ergebnisdaten zuerst über einen Sink-Baustein von einem Multi-Array in ein SPS-Array zu überführen.

Die beschriebene Funktionalität erfordert TC3 Scope View ab Version 1.4.3141 (verfügbar unter anderem mit TwinCAT ab Version 3.1.4024.7).

Name	Type	Size	Subitems
pResultData	POINTER TO ARRAY [1..<aResultDataLen>] OF ARRAY [1..0] OF LREAL	8	0
pResultData^	ARRAY [1..<aResultDataLen>] OF ARRAY [1..0] OF LREAL	41000	5
pResultData^[1]	ARRAY [1..0] OF LREAL	8200	1025
pResultData^[2]	ARRAY [1..0] OF LREAL	8200	1025
pResultData^[3]	ARRAY [1..0] OF LREAL	8200	1025
pResultData^[4]	ARRAY [1..0] OF LREAL	8200	1025
pResultData^[5]	ARRAY [1..0] OF LREAL	8200	1025
aResultDataLen	ARRAY [0..1] OF UDINT	8	2
aResultDataLen[0]	UDINT	4	0
aResultDataLen[1]	UDINT	4	0
hrResultDataSymbol	HRESULT	4	0

5.1 Funktionsbausteine

Nachfolgend werden die zur Verfügung stehenden Funktionsbausteine zur leichteren Auffindbarkeit nach unterschiedlichen Schemata sortiert.

Gesamtheit der Condition Monitoring Bibliothek

Algorithmen und Werkzeuge zur Verarbeitung von Signalen im Bereich:

Signalverarbeitung	Statistik	Klassifikation	Puffer-handling
FB_CMA_AnalyticSignal [▶ 89]	FB_CMA_HistArray [▶ 154]	FB_CMA_DiscreteClassification [▶ 121]	FB_CMA_BufferConverting [▶ 96]
FB_CMA_ArgSort [▶ 93]	FB_CMA_MinersRule [▶ 179]	FB_CMA_RainflowCounting [▶ 211]	FB_CMA_ComplexDataHandling [▶ 108]
FB_CMA_ComplexFFT [▶ 112]	FB_CMA_MomentCoefficients [▶ 183]	FB_CMA_VibrationAssessment [▶ 258]	FB_CMA_Sink [▶ 238]
FB_CMA_Correlation [▶ 115]	FB_CMA_Quantiles [▶ 206]	FB_CMA_WatchUpperThresholds [▶ 262]	FB_CMA_Source [▶ 245]
FB_CMA_CrestFactor [▶ 99]	FB_CMA_EmpiricalMean [▶ 131]		FB_CMA_Normalization
FB_CMA_CrestFactorPlus [▶ 104]	FB_CMA_EmpiricalStandardDeviation [▶ 141]		
FB_CMA_Downsampling [▶ 125]	FB_CMA_EmpiricalSkew [▶ 136]		
FB_CMA_Envelope [▶ 146]	FB_CMA_EmpiricalExcess [▶ 126]		
FB_CMA_EnvelopeSpectrum [▶ 150]	FB_CMA_MovingMean [▶ 270]		
FB_CMA_InstantaneousFrequency [▶ 160]	FB_CMA_MovingStandardDeviation [▶ 274]		
FB_CMA_InstantaneousPhase [▶ 164]	FB_CMA_MovingSkew [▶ 278]		
FB_CMA_IntegratedRMS [▶ 167]	FB_CMA_MovingExcess [▶ 283]		
FB_CMA_MagnitudeSpectrum [▶ 172]			
FB_CMA_MultiBandRMS [▶ 189]			
FB_CMA_PowerCepstrum [▶ 198]			
FB_CMA_PowerSpectrum [▶ 202]			
FB_CMA_OrderPowerSpectrum [▶ 194]			
FB_CMA_RealFFT [▶ 216]			
FB_CMA_RMS [▶ 220]			
FB_CMA_SlidingDFT [▶ 224]			
FB_CMA_SparseSpectrum [▶ 229]			

FB_CMA_SpikeEnergySpectrum [▶ 233]
FB_CMA_ZoomFFT [▶ 266]

Thematische Gliederung zur Signalverarbeitung

Algorithmen zur Analyse von Signalen im:

Zeitbereich	Frequenzbereich	Zeit-Frequenzbereich	Weitere
FB_CMA_AnalyticSignal [▶ 89]	FB_CMA_ComplexFFT [▶ 112]	FB_CMA_InstantaneousFrequency [▶ 160]	FB_CMA_ArgSort [▶ 93]
FB_CMA_CrestFactor [▶ 99]	FB_CMA_EnvelopeSpectrum [▶ 150]	FB_CMA_SpikeEnergySpectrum [▶ 233]	FB_CMA_ComplexDataHandling [▶ 108]
FB_CMA_CrestFactorPlus [▶ 104]	FB_CMA_IntegratedRMS [▶ 167]		FB_CMA_Downsampling [▶ 125]
FB_CMA_Envelope [▶ 146]	FB_CMA_MagnitudeSpectrum [▶ 172]		FB_CMA_OrderPowerSpectrum [▶ 194]
FB_CMA_InstantaneousPhase [▶ 164]	FB_CMA_MultiBandRMS [▶ 189]		
FB_CMA_RMS [▶ 220]	FB_CMA_PowerSpectrum [▶ 202]		
FB_CMA_PowerCepstrum [▶ 198]	FB_CMA_RealFFT [▶ 216]		
FB_CMA_Correlation [▶ 115]	FB_CMA_SlidingDFT [▶ 224]		
FB_CMA_MovingMean [▶ 270]	FB_CMA_SparseSpectrum [▶ 229]		
FB_CMA_MovingStandardDeviation [▶ 274]	FB_CMA_ZoomFFT [▶ 266]		
FB_CMA_MovingSkew [▶ 278]			
FB_CMA_MovingExcess [▶ 283]			
FB_CMA_MomentCoefficients [▶ 183]			

5.1.1 FB_CMA_AnalyticSignal

Berechnung des Analytischen Signals einer Zeitreihe.

Das Analytische Signal ist die komplexwertige Ergänzung des eingehenden reellen Signals, wobei der Imaginärteil gegenüber dem unveränderten Realteil um 90° phasenverschoben ist. Der Imaginärteil wird über die Hilbert-Transformierte des eingehenden reellen Signals gebildet. In zeitkontinuierlicher Darstellung ist das analytische Signal $x_{analytic}(t)$ des reellen Signal $x(t)$ beschrieben durch

$$x_{\text{analytic}}(t) = x(t) + \mathrm{i}\mathcal{H}[x(t)]$$

Der Funktionsbaustein berechnet das analytische Signal mittels einer diskreten Hilbert-Transformation im Frequenzbereich. Das Ergebnis ist ein komplexwertiger Vektor der Länge `nWindowLength/2`.

Der Eingabevektor wird mit einem 50% überlappenden vorhergehenden Eingabevektor mit der Welch-Methode kombiniert. Anschließend wird er mit einer Fensterfunktion multipliziert (verwendeter Fenstertyp: `eCM_HannWindow`). Danach wird eine FFT für reelle Eingangswerte angewandt. Im Frequenzbereich wird auf

das Signal die Hilbert-Transformation angewandt, welche ein komplexwertiges Ergebnis liefert. Anschließend wird das Ergebnis mit einer inversen FFT zurück in den Zeitbereich transformiert. Das so erhaltene Zeitsignal wird mit Hilfe der Overlap-Add Methode überlappend aufaddiert. Bei der Signalsynthese wird erneut eine Fensterung durchgeführt (verwendeter Fenstertyp: `eCM_HannWindow`). Es ergibt sich somit ein kontinuierliches Ausgangssignal ohne Sprünge.

Fensterlänge beachten

Der Wert von `nWindowLength` muss kleiner oder gleich dem Wert von `nFFT_Length` sein. Die Länge der FFT kann sich an der benötigten Frequenzauflösung orientieren. Typischerweise wird oft ein Wert von ca. 4/5 der FFT-Länge als Fensterlänge verwendet.

Wenn `nFFT_Length` größer ist als `nWindowLength`, wird die Frequenzauflösung der FFT (und damit auch die Länge des Vektors der Rückgabewerte) vergrößert. Die Differenz der Länge wird vor der Fouriertransformation mit Nullen aufgefüllt. Dies kann sinnvoll sein, um eine höhere Frequenzauflösung zu erreichen oder, z.B. bei der Berechnung mit Rücktransformation in den Zeitbereich, zirkuläres Aliasing zu vermeiden. Das Ergebnis enthält trotz der höheren Frequenzauflösung allerdings nicht mehr Information.

Gedächtniseigenschaften

Aufgrund der Verwendung der Overlap-Add-Methode werden jeweils der aktuelle Eingangspuffer zusammen mit den zwei zuletzt übergebenen Puffern zur Berechnung genutzt.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1, nWindowLength/2`	`LCOMPLEX, 1, nWindowLength/2`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2, nChannels x nWindowLength/2`	`LCOMPLEX, 2, nChannels x nWindowLength/2`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_AnalyticSignal_InitPars;   // init parameter
    nOwnID           : UDINT;                           // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                      // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_AnalyticSignal_InitPars [▶ 297]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.

- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars      : ST_CM_AnalyticSignal_InitPars;    // init parameter
    nOwnID          : UDINT;                            // ID for this FB instance
    aDestIDs        : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers  : UDINT := 4;                       // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_AnalyticSignal_InitPars [▶ 297]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_Envelope [▶ 146] berechnet die Einhüllende einer Zeitreihe.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.2 FB_CMA_ArgSort

Sortiert die eingehenden Argumente

Die eingehenden Argumente werden wahlweise aufsteigend oder absteigend sortiert. Hierbei werden etwaig auftretende NaN-Werte in den Eingangsdaten an das Ende der Ausgangsdaten geschoben. Man liefert ein eindimensionales Array z.B. den Ausgang von einem Power Spektrum als Eingangsdatenstrom. Als Ausgangsdaten erhält man ein zweidimensionales Array. In der ersten Dimension die Amplitude und in der zweiten den Index wo diese Amplitude im Eingangs-Array zu finden ist. Über einen Skalierungsfaktor kann anstelle des Index z.B. direkt die Frequenz angezeigt werden. Der Baustein rechnet intern mit „0“-basierten Arrays. Dies ist bei der Auswertung zu berücksichtigen.

NaN Vorkommnis

Mögliche NaN-Werte in den Eingangsdaten werden an das Ende der Ausgangsdaten sortiert, um ein stets gültiges Ergebnis der Sortierung zu erhalten.

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle

Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1,` `nInLength`	`LREAL, 2,` `nInLength x 2`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2,` `nChannels x nInLength`	`LREAL, 3,` `nChannels x nInLength x 2`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_ArgSort_InitPars;           // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_ArgSort_InitPars [▶ 297]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars      : ST_CM_ArgSort_InitPars;           // init parameter
    nOwnID          : UDINT;                            // ID for this FB instance
    aDestIDs        : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers  : UDINT := 4;                       // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_ArgSort_InitPars [▶ 297]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Keine.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.3 FB_CMA_BufferConverting

Kopiert Daten von einem MultiArray zu einem anderen MultiArray.

Falls der definierte Eingangspuffer eines Algorithmus-Funktionsbausteins nicht mit dem Ausgangspuffer des vorhergehenden Funktionsbausteins der Analysekette übereinstimmt, kann mit dieser Funktionalität die Übergabe erreicht werden. Eine unterschiedliche Anzahl von Dimensionen kann entsprechend konvertiert werden.

Ebenfalls bietet sich die Möglichkeit nur eine Teilmenge der Daten weiterzuverarbeiten, um beispielsweise nur relevante Frequenzbereiche eines Spektrums zu berücksichtigen.

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

MultiArray im	Elementtyp	Dimensionen	Dimensionsgrößen
output stream	eTypeCode	nDims	aDimSizes

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_MA_MultiArray_InitPars;       // init parameter
    nOwnID           : UDINT;                           // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                      // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_MA_MultiArray_InitPars [▶ 322]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Copy1D():

Kopiert eindimensionale Daten von einem MultiArray zu einem anderen MultiArray.

```
METHOD Copy1D : HRESULT
VAR_INPUT
    nWorkDimIn     : UDINT;                // It designates the dimension in the input MultiArray be
ing processed.
    nWorkDimOut    : UDINT;                // It designates the dimension in the output MultiArray b
eing processed.
    nElements      : UDINT;                // optional: default:0-
>full copy; It designates the number of elements to be copied out of the MultiArray.
    pStartIndexIn  : POINTER TO UDINT;    (* optional: default:0-
>internally handled as [0,0,..]; It designates the index of the first element to be copied out of th
e MultiArray.
```

```
                        If allocated it must point to a onedimensional array of UDINT with so many
elements as dimensions of the MultiArray. *)
    pStartIndexOut : POINTER TO UDINT;    (* optional: default:0-
>internally handled as [0,0,..]; It designates the index of the first MultiArray element to be copie
d.
                        If allocated it must point to a onedimensional array of UDINT with so many
elements as dimensions of the MultiArray. *)
    nOptionPars    : DWORD;               // option mask
END_VAR
VAR_OUTPUT
    bNewResult     : BOOL;                // TRUE every time when outgoing MultiArray was calculated
 and sent to transfer tray.
    bError         : BOOL;                // TRUE if an error occurs.
    hrErrorCode    : HRESULT;             // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `nWorkDimIn`: Falls das Eingangs-Multiarray mehrdimensional ist, kann die Dimension gewählt werden, dessen Daten kopiert werden sollen. Hierbei wäre die erste Dimension mit 0 anzugeben (0 basierend).
- `nWorkDimOut`: Falls das Ausgangs-Multiarray mehrdimensional ist, kann die Dimension gewählt werden, auf welche die Daten kopiert werden sollen. Hierbei wäre die erste Dimension mit 0 anzugeben (0 basierend).
- `nElements`: Um die vollständigen Daten einer Multiarray-Dimension zu kopieren, kann dieser Parameter auf 0 gesetzt werden. Die Gesamtanzahl wird in dem Fall intern ermittelt. Alternativ kann hier die Anzahl der zu kopierenden Elemente angegeben werden.
- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Transpose():

Transponiert zwei Dimensionen, sodass der Ausgangspuffer das Transponierte des Eingangspuffers ist. Diese Operation ist nur für ein- und zweidimensionale Eingangspuffer möglich.

```
METHOD Transpose : HRESULT
VAR_INPUT
    nOptionPars : DWORD;   // option mask
END_VAR
VAR_OUTPUT
    bNewResult  : BOOL;    // TRUE every time when outgoing MultiArray was calculated and sent to
transfer tray.
    bError      : BOOL;    // TRUE if an error occurs.
    hrErrorCode : HRESULT; // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bNewResult`: Der Ausgang ist `TRUE`, jedes Mal, wenn ein Ausgangs-MultiArray berechnet und an das TransferTray gesendet wurde.
- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_MA_MultiArray_InitPars;       // init parameter
    nOwnID         : UDINT;                           // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_MA_MultiArray_InitPars [▶ 322]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

5.1.4 FB_CMA_CrestFactor

Berechnet für ein- und mehrkanalige Zeitreihen den Crest-Faktor für jeden Kanal.

Dieser ist definiert als das Verhältnis zwischen dem Betrag des Spitzenwerts eines Signals und dem RMS Wert.

$$C_{\mathrm{dB}} = 20\log_{10}\left(\frac{\max\{|x(t)|\}}{\mathrm{rms}\{x(t)\}}\right)$$

Der Crest-Faktor wird in der logarithmischen Einheit Dezibel berechnet. Eine Sinuswelle hat beispielsweise einen Crest-Faktor von 3.01 dB (=1.414). Er erlaubt z.B. Rückschlüsse auf den Zustand von Wälzlagern. Generell steigt der Crest Faktor bei einer beginnenden Schädigung eines Wälzlagers an und kann bei fortschreitendem Verlauf wieder zurückgehen.

Zur Vermeidung von Wertebereichsfehlern wird jeder berechnete Wert vor der Anwendung des Logarithmus mit einem Minimalwert verglichen und, sofern er kleiner als dieser ist, durch diesen Minimalwert ersetzt.

Da der Crest-Faktor durch das Verhältnis von Spitzenwert zum RMS-Mittelwert definiert ist, bedeutet dies, dass das Ergebnis stark durch einzelne Maxima bestimmt wird, was zu unerwarteten Resultaten führen kann.

Gedächtniseigenschaften

Für die Berechnung des RMS Wertes werden intern `nBufferLength` Werte der Zeitreihe je Kanal/ Unterkanal gespeichert. Bei einem Aufruf mit kleinerer Eingangspuffergröße können weniger Werte übergeben werden. In diesem Fall wird der Pufferinhalt verschoben und die Signallänge mit der geeigneten Zahl an neu übergebenen Werten aufgefüllt. Ist die Eingangspuffergröße größer als der interne Puffer, so wird dieser mit den jüngsten Werten für die Berechnung gefüllt.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Der Ausgangsvektor wird so lange mit NaN gefüllt, bis der interne Puffer komplett mit neuen gültigen Werten gefüllt wurde.

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Crest Faktor [▶ 360].

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante für mehrere Datensätze (`nSubChannels = 0`)	`LREAL, 2, nChannels x` *not specified**	`LREAL, 1, nChannels`
Mehrkanalige Variante für mehrere Datensätze (`nSubChannels > 0`)	`LREAL, 3, nChannels x nSubChannels x` *not specified**	`LREAL, 2, nChannels x nSubChannels`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorrausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_CrestFactor_InitPars;      // init parameter
    nOwnID           : UDINT;                           // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                      // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_CrestFactor_InitPars [▶ 298]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_CrestFactor_InitPars;      // init parameter
    nOwnID         : UDINT;                           // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_CrestFactor_InitPars [▶ 298]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_HistArray [▶ 154] berechnet die Histogramme von Verteilungen von Eingangswerten.

Der Baustein FB_CMA_Quantiles [▶ 206] berechnet die Quantile einer empirischen Verteilung, die ebenfalls erlauben, die Häufigkeit von Ausreißern zu beurteilen.

Der Baustein FB_CMA_MomentCoefficients [▶ 183] stellt mit der Kurtosis ein alternatives Maß für die Spitzenhaltigkeit eines Signals zur Verfügung, das weniger empfindlich gegenüber Ausreißern ist.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.5 FB_CMA_CrestFactorPlus

Berechnet für ein und mehrkanalige Zeitreihen den Crest-Faktor Plus für jeden Kanal.

Dieser ist definiert als Linearkombination des Betrages des Spitzenwerts eines Signals, dem RMS Wert und dem Verhältnis zwischen den beiden Größen:

$$C^{+} = c_1 \max\{|x(t)|\} + c_2 \operatorname{rms}\{x(t)\} + c_3 \frac{\max\{|x(t)|\}}{\operatorname{rms}\{x(t)\}}, \qquad C^{+}_{\mathrm{dB}} = 20 \log_{10}\left(C^{+}\right), \qquad c_1, c_2, c_3 \in \mathbb{R}$$

Die Standardwerte der Konsanten sind so gewählt, dass der Crest-Faktor Plus mit dem Crest-Faktor ohne eine gesonderte Konfiguration der Parameter übereinstimmt.

Im Unterschied zum Crest-Faktor, steigt der Wert bei passender Konfiguration im fortschreitenden Verlauf einer Schädigung eines Wälzlagers an. Somit ist eine rückwirkende Kenntnis des Lagerzustandes nicht erforderlich. Vergleiche hierzu FB_CMA_CrestFactor [▶ 99].

Zur Vermeidung von Wertebereichsfehlern in der Berechnung wird jeder berechnete Wert vor der Anwendung des Logarithmus mit einem Minimalwert verglichen und, sofern er kleiner als dieser ist, durch diesen Minimalwert ersetzt.

Da der Crest-Faktor Plus von dem Verhältnis von Spitzenwert zum RMS-Mittelwert abhängig ist, kann das Ergebnis stark durch einzelne Maxima bestimmt werden. Dies kann zu unerwarteten Resultaten führen.

Gedächtniseigenschaften

Für die Berechnung des RMS Wertes werden intern `nBufferLength` Werte der Zeitreihe je Kanal/ Unterkanal gespeichert. Bei einem Aufruf mit kleinerer Eingangspuffergröße können weniger Werte übergeben werden. In diesem Fall wird der Pufferinhalt verschoben und die Signallänge mit der geeigneten Zahl an neu übergebenen Werten aufgefüllt. Ist die Eingangspuffergröße größer als der interne Puffer, so wird dieser mit den jüngsten Werten für die Berechnung gefüllt.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Der Ausgangsvektor wird so lange mit NaN gefüllt, bis der interne Puffer komplett mit neuen gültigen Werten gefüllt wurde.

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante für mehrere Datensätze (`nSubChannels = 0`)	`LREAL, 2,` `nChannels x` *not specified**	`LREAL, 1,` `nChannels`
Mehrkanalige Variante für mehrere Datensätze (`nSubChannels > 0`)	`LREAL, 3,` `nChannels x nSubChannels x` *not specified**	`LREAL, 2,` `nChannels x nSubChannels`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorrausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_CrestFactorPlus_InitPars;      // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_CrestFactorPlus_InitPars [▶ 298]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError          : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.

- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Configure() :

Mit dem Aufruf dieser Methode können die Gewichte des Betrages des Spitzenwerts, dem RMS Wert und dem Verhältnis zwischen den beiden Größen neu konfiguriert werden. Das entsprechende SPS Array muss wie folgt definiert sein.

- **Rückgabewert:** Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : POINTER TO LREAL;  // pointer to 2-dimensional array (LREAL) of arguments
    nArgSize : UDINT;             // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`. Die drei zu konfigurierenden Parameter je Kanal und Unterkanal sind `c1`; `c2`; `c3`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle und Unterkanäle	`LREAL, 1, 3`
Kanalspezifische Konfiguration (`nSubChannels = 0`)	`LREAL, 2, nChannels x 3`
Unterkanalspezifische Konfiguration (`nSubChannels > 0`)	`LREAL, 2, nSubChannels x 3`
Kanal- und unterspezifische Konfiguration (`nSubChannels > 0`)	`LREAL, 3, nChannels x nSubChannels x 3`

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_CrestFactorPlus_InitPars;  // init parameter
    nOwnID         : UDINT;                           // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_CrestFactorPlus_InitPars [▶ 298]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Funktionsbaustein FB_CMA_CrestFactor [▶ 99] berechnet die Standardvariante des Crest-Faktors.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.6 FB_CMA_ComplexDataHandling

Konvertierung von komplexen Eingangsdaten in reelle Ausgangsdaten

Der Baustein FB_CMA_ComplexDataHandling extrahiert aus komplexen Eingangsdaten entweder den Real- oder Imaginärteil oder berechnet den Absolutbetrag oder die Phase.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, werden die zugehörigen Elemente im Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN-Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LCOMPLEX, 1, nInLength`	`LREAL, 1, nInLength`
Mehrkanalige Variante Spektrum (`nChannels > 1`)	`LCOMPLEX, 2, nChannels x nInLength`	`LREAL, 2, nChannels x nInLength`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_ComplexDataHandling_InitPars;    // init parameter
    nOwnID           : UDINT;                                 // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;       // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                            // number of MultiArrays which should
be initialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;                  // timeout checking off during access
to inter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_ComplexDataHandling_InitPars [▶ 299]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Configure():

Mit dem Aufruf dieser Methode kann die verwendete Methode zur Behandlung der komplexen Eingangsdaten zur Laufzeit verändert werden. Für die Auswahl der Methode kann der Datentyp E_CM_ComplexDataHandling [▶ 289] verwendet werden.

Das entsprechende SPS Array muss wie folgt definiert sein. Für eine erneute Konfiguration mit einem anderen Satz an Argumenten, kann die `Configure()` Methode ebenfalls genutzt werden.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : PVOID;    // pointer to array of arguments
    nArgSize : UDINT;    // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle	`UDINT, 1, 1`
Kanalspezifische Konfiguration (`nChannels > 1`)	`UDINT, 2, nChannels x 1`

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars      : ST_CM_ComplexDataHandling_InitPars;  // init parameter
    nOwnID          : UDINT;                               // ID for this FB instance
    aDestIDs        : ARRAY[1..cCMA_MaxDest] OF UDINT;     // IDs of destinations for output
```

```
    nResultBuffers : UDINT := 4;                          // number of MultiArrays which should be i
nitialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_ComplexDataHandling_InitPars [▶ 299]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_BufferConverting [▶ 96] ermöglicht das Transponieren oder Extrahieren niederdimensionaler Daten aus mehrdimensionalen Eingangsdaten.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.7 FB_CMA_ComplexFFT

Berechnung der schnellen Fouriertransformation (FFT) für komplex-wertige Eingangssignale.

Der Baustein FB_CMA_ComplexFFT berechnet die Fourier-Transformierte des am Baustein anliegenden komplex-wertigen Eingangssignals *x*[n]. Dabei wird ein performanter FFT-Algorithmus verwendet. Es ist möglich sowohl die Hin- als auch die Rücktransformation zu berechnen. Die Einstellung erfolgt über den Input `stInitPars` (siehe Ein- und Ausgänge).

Definition der Fourier-Hintransformation in DFT-Notation:

$$X[k] = \sum_{n=0}^{N-1} x[n]\, \mathrm{e}^{-\mathrm{i}2\pi nk/N}$$

Definition der Fourier-Rücktransformation in DFT-Notation:

$$x[n] = \frac{1}{N} \sum_{n=0}^{N-1} X[k]\, \mathrm{e}^{\mathrm{i}2\pi nk/N}$$

Die höchste Frequenz einer Komponente im Eingangssignal sollte maximal bei der halben Abtastrate des Eingangssignals liegen, damit Aliasing-Effekte vermieden werden.

Die FFT ist als Transformierte eines zyklisch fortgesetzten Signals definiert. Dies kann zur Feststellung von Sprüngen führen, sobald das zyklisch fortgesetzte Signal nicht exakt kontinuierlich, also am Anfang und am Ende gleich ist. Der Baustein FB_CMA_PowerSpectrum [▶ 202] sowie FB_CMA_MagnitudeSpectrum [▶ 172] vermeiden diese Schwierigkeiten durch eine Analyse in überlappenden, mit einer Fensterfunktion multiplizierten Abschnitten.

Skalierung

Zur messtechnisch quantitativen Auswertung des Spektrums ist das berechnete Spektrum mit `1/nFFT_Length` für den Gleichanteil, also dem ersten Array-Element des Outputs, und mit `2/nFFT_Length` für alle anderen Elemente des Outputs zu gewichten. Der Baustein skaliert bei der Hin-Transformation und der Rücktransformation derart, dass bei aufeinanderfolgender Hin- und Rücktransformation direkt das ursprüngliche Eingangssignal wieder am Ausgang berechnet wird.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: FFT mit komplex-wertigem Eingangssignal [▶ 327].

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LCOMPLEX, 1, nFFT_Length`	`LCOMPLEX, 1, nFFT_Length`
Mehrkanalige Variante (`nChannels > 1`)	`LCOMPLEX, 2, nChannels x nFFT_Length`	`LCOMPLEX, 2, nChannels x nFFT_Length`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_ComplexFFT_InitPars;       // init parameter
    nOwnID           : UDINT;                           // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                      // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;            // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_ComplexFFT_InitPars [▶ 299]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult    : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError        : BOOL;       // TRUE if an error occurs.
    hrErrorCode   : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars      : ST_CM_ComplexFFT_InitPars;       // init parameter
    nOwnID          : UDINT;                           // ID for this FB instance
    aDestIDs        : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers  : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_ComplexFFT_InitPars [▶ 299]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise

sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_RealFFT [▶ 216] berechnet die Fouriertransformation eines reellwertigen Signals.

Der Baustein FB_CMA_PowerSpectrum [▶ 202] berechnet das Leistungsspektrum eines fortlaufenden Zeitsignals.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.8 FB_CMA_Correlation

Berechnung der Korrelationsfunktion für ein- und mehrkanalige reell-wertige Zeitreihen.

Der Funktionsbaustein FB_CMA_Correlation berechnet die Korrelationsfunktion reell-wertiger Zeitreihen.

Die Korrelationsfunktion zweier Signale beschreibt deren Ähnlichkeit bei unterschiedlichen zeitlichen Verschiebungen gegeneinander. Die zeitdiskrete Verschiebung wird im Folgenden mit *m* bezeichnet.

Die Eingangsdaten bestehen aus einem Tupel aus einem (oder mehreren) Testsignalen $x[n]$ sowie einem (oder mehreren) Referenzsignalen $y[n]$. Die Eingangsdaten werden über den Baustein FB_CMA_SourcePaired [▶ 252] in die Analysekette eingebracht. Hierbei ist zu beachten, dass bei der Angabe eines einzelnen Referenzsignals alle Kanäle der Testsignale mit dem gleichen Referenzsignal korreliert werden. Hingegen bei der Angabe mehrerer Referenzsignale paarweise das Referenzsignal von Kanal *i* mit dem Testsignal von Kanal *i* korreliert wird. Die Anzahl der Kanäle `nChannels` muss entsprechend übereinstimmen.

Die Anwendungsmöglichkeiten der Korrelationsfunktion sind vielfältig. So ist es beispielsweise möglich eine Signalreferenz eines Prozessschritts zu erfassen und diese dann fortwährend mit Wiederholungen dieses Prozessschritts zu vergleichen. Das Maß der Ähnlichkeit (Wert des Maximums der Korrelationsfunktion) kann dann zur Überwachung des Prozessschritts herangezogen werden. Ebenso ist die Korrelationsfunktion zur Bestimmung von zeitlichen Verschiebungen ein gutes Werkzeug. Hier wird nicht der Wert des Maximums, sondern dessen Position auf der Verschiebeachse *m* gesucht. Dies kann beispielsweise zur Ermittlung von Laufzeitunterschieden von Signalen genutzt werden, bspw. bei der Ortung von Leckagen mittels mehrerer Mikrofone. Ganz allgemein lässt sich die Autokorrelation, also die Korrelation eines Signales mit sich selbst, zur Signalerkennung in stark verrauschten Signalen nutzen. Ist das Rauschen zufällig und das Nutzsignal ein deterministisches Signal, beinhaltet das Ergebnis der Autokorrelation die Rauschkomponente nur noch bei *m*=0 und das deterministische Signal wird sichtbar.

Abtasttheorem: Wie der diskreten Fourier-Transformation ist das Abtasttheorem in gleicher Weise gültig. Der zeitkontinuierliche Verlauf einer Autokorrelationsfunktion kann bei diskreten Zeitreihen rekonstruiert werden, wenn die Abtastfrequenz mindestens doppelt so groß ist wie die höchste im Signal vorkommende Frequenz.

Die Berechnung der Korrelationsfunktion erfolgt auf Basis einer der nachfolgenden, konfigurierbaren Definitionen:

Base

$$C_{xy}[m] = \sum_{n=0}^{N-1} x[n-m]\, y[n]$$

Normed

$$\hat{C}_{xy}[m] = \frac{1}{N} \sum_{n=0}^{N-1} x[n-m]\, y[n]$$

Covariance

$$cov_{xy}[m] = \left(\frac{1}{N} \sum_{n=0}^{N-1} x[n-m]\, y[n] \right) - \left(\frac{1}{N} \sum_{n=0}^{N-1} x[n-m] \right) \left(\frac{1}{N} \sum_{n=0}^{N-1} y[n] \right)$$

Covariance (Bessel Korrektur)

$$\widetilde{cov}_{xy}[m] = \frac{N}{N-1}\, cov_{xy}[m]$$

Pearson

$$\rho_{xy}[m] = \frac{cov_{xy}[m]}{\sigma_x[m]\, \sigma_y[m]}, \qquad \sigma_x^2[m] = cov_{xx}[m]$$

Die Wahl der oben genannten Definitionen erfolgt über den Initialparameter `eCorrelationMode` vom Typ E_CM_CorrelationMode [▶ 289]. Bei quantitativer Bewertung der Ähnlichkeit von Signalen wird „Pearson“ empfohlen. Hingegen ist bei Suche nach einer Verschiebezeit die Normierung unerheblich.

Die Berechnung erfolgt über einem gewählten Zeitfenster, welches durch `eWindowMode` vom Typ E_CM_WindowMode [▶ 291] eingestellt wird. Hierbei stehen drei Varianten zur Auswahl: ein gleitendes Fenster, eine festes Fenster sowie die Berechnung auf Grundlage aller gesammelten Daten seit dem letzten `ResetData()`.

Konfiguration

Mit den Initialisierungsparametern werden die Berechnungsvariante sowie das Fenster festgelegt. Diese können mit der `Configure()` Methode für jeden Kanal individuell angepasst werden.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Korrelationsfunktion [▶ 381]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`. Für die Verarbeitung eines einzelnen Kanals (`nChannels = 1`) gilt:

MultiArray im	Elementtyp	Dimensionen	Dimensionsgrößen
input stream A	LREAL	1	*not specified**
input stream B	LREAL	1	*not specified**
output stream	LREAL	1	`1+(nPositiveShift+nNegativeShift)/nStepsize`

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) gilt:

MultiArray im	Elementtyp	Dimensionen	Dimensionsgrößen
input stream A	LREAL	2	`nChannels x` *not specified**
input stream B	LREAL	1	*not specified**
output stream	LREAL	2	`nChannels x (1+(nPositiveShift+nNegativeShift)/nStepsize)`

oder für die paarweise Berechnung der Korrelationskoeffizienten:

MultiArray im	Elementtyp	Dimensionen	Dimensionsgrößen
input stream A	LREAL	2	`nChannels x` *not specified**
input stream B	LREAL	2	`nChannels x` *not specified**
output stream	LREAL	2	`nChannels x (1+(nPositiveShift+nNegativeShift)/nStepsize)`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_Correlation_InitPars;         // init parameter
    nOwnID           : UDINT;                              // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                         // number of MultiArrays which should be
initialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;               // timeout checking off during access to
inter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_Correlation_InitPars [▶ 300]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.

- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Configure():

Mit dem Aufruf dieser Methode können die Skalierung der Korrelationskoeffizienten (Initialparameter `eCorrelationMode`) sowie das zu Grunde liegende Berechnungsfenster (Initialparameter `eWindowMode`) zur Laufzeit verändert werden.

Das entsprechende SPS Array muss wie folgt definiert sein. Für eine erneute Konfiguration mit einem anderen Satz an Argumenten kann die `Configure()` Methode ebenfalls genutzt werden.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : PVOID;     // pointer to array of arguments
    nArgSize : UDINT;     // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle	`UDINT, 1, 2`
Kanalspezifische Konfiguration (`nChannels > 1`)	`UDINT, 2, nChannels x 2`

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars      : ST_CM_Correlation_InitPars;         // init parameter
    nOwnID          UDINT;                                // ID for this FB instance
    aDestIDs        : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers : UDINT := 4;                          // number of MultiArrays which should be in
itialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_Correlation_InitPars [▶ 300]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.

- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.9 FB_CMA_DiscreteClassification

Klassifikation von mehrkanaligen Daten anhand konfigurierbarer Schwellwerte

Der Baustein ordnet die einzelnen Kanäle eines mehrkanaligen Signals anhand konfigurierbarer Schwellwerte einer festen Zahl von diskreten Kategorien zu. Die Zahl der Kanäle und die Anzahl der Kategorien werden bei der Instanziierung festgelegt. Der Baustein lässt sich zur Laufzeit konfigurieren, indem für jeden Kanal und jede Schwellwert-Kategorie die Größe des Schwellwerts angegeben wird.

In der Operationsphase wird für jeden Zeitschritt ein Eingangsvektor angenommen und die Nummer der zutreffenden Kategorie für jeden Kanal berechnet. Rückgabewert ist ein eindimensionales Array, das für jeden Eingangskanal einen vorzeichenbehafteten Integer-Wert enthält, den Index der zugeordneten Kategorie.

Wenn der Eingangswert kleiner ist als der Schwellwert der ersten Kategorie, wird für diesen Kanal der Wert -1 zurückgegeben. Wenn ein Eingangswert größer gleich dem Schwellwert für eine Kategorie ist, wird der null-basierte Index dieser Kategorie zurückgegeben. Falls mehrere Schwellwerte gleich konfiguriert sind, wird der Wert mit dem größten Index genommen.

Konfiguration

Der Funktionsbaustein muss mittels Parameter wie der Anzahl der Klassifikationsklassen konfiguriert werden. Die Klassifikationsschwellwerte jedes Kanals können individuell vergeben werden. Diese Schwellwerte müssen monoton steigend (aber nicht streng monoton) sein. Dementsprechend darf kein Schwellwert kleiner als der vorhergehende Wert sein.

Gedächtniseigenschaften

Der Baustein berücksichtigt nur die bei Konfiguration und Training gespeicherten Werte. Die während der Klassifikation übergebenen Werte haben keinen Einfluss auf spätere Aufrufe.

NaN Vorkommnis

Bei einem Eingangswert von NaN ist das Ergebnis -2. Am Ausgang sind keine NaN Werte zu erwarten.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Beispielimplementierung

Zwei exemplarische Implementierungen sind unter folgenden Links verfügbar: Schwellwertbetrachtung gemittelte Betragsspektren [▶ 359] sowie Condition Monitoring mit Frequenzanalyse [▶ 353].

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nSubChannels = 0`)	`LREAL, 1, nChannels`	`DINT (32bit), 1, nChannels`
Mehrkanalige Variante (`nSubChannels > 0`)	`LREAL, 2, nChannels x nSubChannels`	`DINT (32bit), 2, nChannels x nSubChannels`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_DiscreteClassification_InitPars;  // init parameter
    nOwnID           : UDINT;                                  // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;        // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                             // number of MultiArrays which should
 be initialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;                   // timeout checking off during access
 to inter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_DiscreteClassification_InitPars [▶ 300]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Configure():

Mit dem Aufruf dieser Methode müssen die Klassifikationsargumente zu Beginn konfiguriert werden. Das entsprechende SPS Array muss wie folgt definiert sein. Für eine erneute Konfiguration mit einem anderen Satz an Argumenten kann die `Configure()` Methode ebenfalls genutzt werden.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : POINTER TO LREAL; // pointer to array (LREAL) of arguments
    nArgSize : UDINT;            // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle und Unterkanäle	`LREAL, 1, nMaxClasses`
Kanalspezifische Konfiguration (`nSubChannels = 0`)	`LREAL, 2, nChannels x nMaxClasses`
Unterkanalspezifische Konfiguration (`nSubChannels > 0`)	`LREAL, 2, nSubChannels x nMaxClasses`
Kanal- und unterspezifische Konfiguration (`nSubChannels > 0`)	`LREAL, 3, nChannels x nSubChannels x nMaxClasses`

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_DiscreteClassification_InitPars;   // init parameter
    nOwnID         : UDINT;                                   // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;        // IDs of destinations for output
    nResultBuffers : UDINT := 4;                              // number of MultiArrays which should b
e initialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_DiscreteClassification_InitPars [▶ 300]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_WatchUpperThresholds [▶ 262] berechnet die höchste Kategorie mehrkanaliger Eingangsdaten und gibt diese, sowie die Nummer des zugehörigen Kanals zurück.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.10 FB_CMA_Downsampling

Downsampling von Signaldaten durch Kopieren der Signaldaten von einem SPS Puffers zu einem anderen SPS Puffer (Array).

Ein Signal, als Puffer vorliegend (z.B. Oversampling Array), kann um einen individuellen Teiler verlangsamt abgetastet werden. Downsampling ist eine Möglichkeit niedrigere Frequenzen zu analysieren ohne die FFT Länge vergrößern zu müssen, um eine hohe Auflösung beizubehalten.

Üblicherweise wird ein Downsampling Block in der Condition Monitoring Analysekette vor einem FB_CMA_Source [▶ 245] angefügt.

Ein- und Ausgänge

Eingangsparameter

```
VAR_INPUT
    nDivider   : UDINT := 1;    // given input signal is sampled down by the specified divider. (1 =
 no downsampling)
    nChannels  : UDINT;         // number of channels in data buffer (=1:onedimensional dataset, >1:
twodimensional dataset)
END_VAR
```

- `nDivider`: Gibt den Downsampling-Faktor als ganzzahligen Divisor an. Eine Samplerate von 10 KHz kann beispielsweise mit `nDivider=5` zu einer Samplerate von 2 KHz gewandelt werden.
- `nChannels`: Zum Downsampling eines mehrkanaligen Datensatzes wird die Anzahl der Kanäle am Eingang `nChannels` angegeben.

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

SchreibtDaten des Eingangsdatenpuffers in den Ausgangsdatenpuffer. Der Ausgangsdatenpuffer ist vollständig gefüllt sobald bNewResult gesetzt ist.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_INPUT
    pDataIn        : POINTER TO BYTE;  // address of data buffer (e.g. oversampling data)
```

```
    nDataInSize    : UDINT;            // size of data buffer in bytes
    pDataOut       : POINTER TO BYTE;  // address of result buffer
    nDataOutSize   : UDINT;            // size of data buffer in bytes
    nElementSize   : UDINT;            // Size of element type in bytes
END_VAR
VAR_OUTPUT
    bNewResult     : BOOL;             // TRUE every time when outgoing data buffer was calculated.
    bError         : BOOL;             // TRUE if an error occurs.
    hrErrorCode    : HRESULT;          // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataIn`: Der zugewiesene Eingangspuffer.
- `nDataInSize`: Gibt die Größe des SPS Eingangspuffers an und muss folgende Bedingung erfüllen: `nChannels * nElementSize *` „Elementanzahl pro Kanal“.
- `pDataOut`: Der zugewiesene Ausgangspuffer muss unverändert bleiben bis der Ausgang `bNewResult` gesetzt wird. Üblicherweise werden Ein- und Ausgangspuffer immer beibehalten.
- `nDataOutSize`: Gibt die Größe des SPS Ausgangspuffers an und muss folgende Bedingung einhalten: `nDataOutSize = n * nDataInSize`. Falls der Quotient ohne Rest durch den Parameter `nDivider` teilbar ist, ist alternativ folgende Bedingung anwendbar: `nDataOutSize = n * (nDataInSize/nDivider)`.
- `nElementSize`: Gibt die Größe eines Elements in Bytes an. Die Größe ist beispielsweise 8 für ein Array von LREAL Elementen.
- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ HRESULT ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Falls ein Fehler auftritt, wird keine Kopieraktion ausgeführt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.11 FB_CMA_EmpiricalExcess

Berechnet den Exzess für ein- und mehrkanalige reell-wertige Zeitreihen.

Der Baustein behandelt das Eingangssignal als Zeitreihe mit ggf. mehreren unabhängigen Kanälen. Für jeden Kanal wird der empirische Exzess nach Gleichung

$$E = \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{x[n]-\overline{x}}{s}\right)^4 - 3$$

berechnet, wobei *s* die empirische Standardabweichung ist. Der Exzess ist der um den Wert 3 geminderte Wert der empirischen Wölbung (auch Kurtosis genannt), wobei der 3 gerade der Wölbung einer Normalverteilung entspricht. Daraus ergibt sich in der Interpretation des Exzess:

Exzess > 0: Verteilung spitzer als Normalverteilung; deutet auf häufige Peaks in der Stichprobe hin

Exzess < 0: Im Vergleich zur Normalverteilung abgeflachte Verteilung

Der Exzess bietet z.B. die Möglichkeit zur Beurteilung von Stößen im Vibrationssignal, wie sie beim Überrollen von lokalen Schädigungen im Wälzlager entstehen.

Es kann sowohl in jedem Zyklus ein einzelnes Sample pro Kanal (siehe Ein- und Ausgänge, erste Tabelle) als auch in einem Zyklus mehrere Samples pro Kanal zur Stichprobenmenge hinzugefügt werden (siehe Ein- und Ausgänge, zweite Tabelle).

Weitere Anmerkungen

Zur Berechnung eines ersten Ergebnisses müssen 4 Werte vorliegen. Des Weiteren darf die Standardabweichung nicht Null sein. Ergebnisse können unter Umständen ungenau werden, wenn die Eingangswerte viele Größenordnungen auseinanderliegen.

Gedächtniseigenschaften

Die Stichprobenmenge *N*, welche zur Berechnung des aktuellen Ergebnisses genutzt wird, erhöht sich automatisch bei jedem neuen eingehenden Datensatz, d.h. der Baustein verwendet alle Eingangswerte seit dessen Instanziierung. Das Zurücksetzen der Stichprobenmenge auf Null (löschen des internen Speichers des FBs) ist über eine `ResetData()` Methode oder, bei Verwendung der `CallEx()` Methode über die Variable `bResetData` vorgesehen.

NaN Vorkommnis

Falls für einen Kanal noch keine ausreichende Anzahl Eingangswerte für die Berechnung eines Ergebnisses vorliegt oder die Varianz Null beträgt, wird für diesen Kanal der Wert NaN (Not A Number) nach IEEE 754 zurückgegeben. Das Vorliegen dieses Fehlerwertes kann mit der Funktion LrealIsNaN() überprüft werden. Ursache dafür kann entweder sein, dass noch nicht genug Eingangsdaten übergeben wurden oder dass für einzelne Kanäle bisher lediglich NaNs als Eingangswerte übergeben wurden. Eine Varianz von Null kann insbesondere auftreten, wenn die Zeitreihe der Werte konstant ist, etwa wenn aufgrund von Kabelbruch oder Schaltfehlern keine Sensordaten übertragen werden.

Enthält ein Satz von Eingangswerten die spezielle Konstante NaN, so wird der Statistik für diesen Kanal bei diesem Zeitschritt kein Wert hinzugefügt, d.h. sie wird als Indikator für fehlende Werte behandelt.

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Statistische Methoden [▶ 345]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen folgender Definition (Shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nSubChannels = 0`)	`LREAL, 1, nChannels`	`LREAL, 1, nChannels`
Standardvariante für mehrere Datensätze (`nSubChannels = 0`)	`LREAL, 2, nChannels x` *not specified**	`LREAL, 1, nChannels`
Mehrkanalige Variante (`nSubChannels > 0`)	`LREAL, 2, nChannels x nSubChannels`	`LREAL, 2, nChannels x nSubChannels`
Mehrkanalige Variante für mehrere Datensätze (`nSubChannels > 0`)	`LREAL, 3, nChannels x nSubChannels x` *not specified**	`LREAL, 2, nChannels x nSubChannels`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorrausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_EmpiricalMoments_InitPars;  // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;           // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_EmpiricalMoments_InitPars [▶ 301]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError          : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
```

```
    nCntResults     : ULINT;                          // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult    : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError        : BOOL;        // TRUE if an error occurs.
    hrErrorCode   : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

CallEx():

Die Methode wird jeden Zyklus aufgerufen um den internen Speicher aus dem Eingangssignal zu aktualisieren. Ein Ergebnis wird nur alle `nAppendData` Zyklen ausgegeben. Eine alternative Methode ist `Call()`.

Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD CallEx : HRESULT
VAR_INPUT
    nAppendData   : UDINT;       // count of data buffers which are appended until calculation (1= cal
culate always)
    bResetData    : BOOL;        // automatic reset of dataset buffer after each calculation
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError        : BOOL;         // TRUE if an error occurs.
    hrErrorCode   : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `nAppendData`: Legt fest wie viele Eingangsdatenpuffer gesammelt werden sollen bevor eine Berechnung ausgeführt wird, denn vorzugsweise werden mehrere Datenblöcke angefügt um ein präzises Ergebnis zu erreichen.
- `bResetData`: Falls gesetzt, wird der interne Datenpuffer nach Berechnung vollständig gelöscht.
- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.

- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_EmpiricalMoments_InitPars;  // init parameter
    nOwnID         : UDINT;                              // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers: UDINT := 4;                          // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_EmpiricalMoments_InitPars [▶ 301]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

Alternativ kann der automatische Reset in der Methode `CallEx()` verwendet werden.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am

Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_EmpiricalMean [▶ 131] berechnet den empirischen Mittelwert von Eingangswerten.

Der Baustein FB_CMA_EmpiricalStandardDeviation [▶ 141] berechnet die empirische Standardabweichung von Eingangswerten.

Der Baustein FB_CMA_EmpiricalSkew [▶ 136] berechnet die empirische Schiefe von Eingangswerten.

Der Baustein FB_CMA_MomentCoefficients [▶ 183] berechnet je nach Parametrierung, sowohl den empirischen Mittelwert, also auch Standardabweichung, Schiefe und Exzess.

Der Baustein FB_CMA_HistArray [▶ 154] berechnet die Histogramme von Verteilungen von Eingangswerten.

Der Baustein FB_CMA_Quantiles [▶ 206] berechnet die Quantile einer empirischen Verteilung, die ebenfalls erlauben die Häufigkeit von Ausreißern zu beurteilen.

Der Baustein FB_CMA_CrestFactor [▶ 99] berechnet ein zur Kurtosis alternatives Maß für die Spitzenhaltigkeit (Crest Faktor) eines Signals, das allerdings empfindlicher gegenüber Ausreißern ist.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.12 FB_CMA_EmpiricalMean

Berechnet den Mittelwert für ein- und mehrkanalige reell-wertige Zeitreihen.

Der Baustein behandelt das Eingangssignal als Zeitreihe mit ggf. mehreren unabhängigen Kanälen. Für jeden Kanal wird der empirische (arithmetische) Mittelwert nach Gleichung

$$\overline{x} = \frac{1}{N}\sum_{n=0}^{N-1} x[n]$$

berechnet. Es kann sowohl in jedem Zyklus ein einzelnes Sample pro Kanal (siehe Ein- und Ausgänge, erste Tabelle) als auch in einem Zyklus mehrere Samples pro Kanal zur Stichprobenmenge hinzugefügt werden (siehe Ein- und Ausgänge, zweite Tabelle).

Gedächtniseigenschaften

Die Stichprobenmenge *N*, welche zur Berechnung des aktuellen Ergebnisses genutzt wird, erhöht sich automatisch bei jedem neuen eingehenden Datensatz, d.h. der Baustein verwendet alle Eingangswerte seit dessen Instanziierung. Das Zurücksetzen der Stichprobenmenge auf Null (löschen des internen Speichers des FBs) ist über eine `ResetData()` Methode oder, bei Verwendung der `CallEx()` Methode über die Variable `bResetData` vorgesehen.

NaN Vorkommnis

Falls für einen Kanal noch keine ausreichende Anzahl Eingangswerte für die Berechnung eines Ergebnisses vorliegt oder die Varianz Null beträgt, wird für diesen Kanal der Wert NaN (Not A Number) nach IEEE 754 zurückgegeben. Das Vorliegen dieses Fehlerwertes kann mit der Funktion LrealIsNaN() überprüft werden. Ursache dafür kann entweder sein, dass noch nicht genug Eingangsdaten übergeben wurden oder dass für einzelne Kanäle bisher lediglich NaNs als Eingangswerte übergeben wurden. Eine Varianz von Null kann insbesondere auftreten, wenn die Zeitreihe der Werte konstant ist, etwa wenn aufgrund von Kabelbruch oder Schaltfehlern keine Sensordaten übertragen werden.

Enthält ein Satz von Eingangswerten die spezielle Konstante NaN, so wird der Statistik für diesen Kanal bei diesem Zeitschritt kein Wert hinzugefügt, d.h. sie wird als Indikator für fehlende Werte behandelt.

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Statistische Methoden [▶ 345]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen folgender Definition (Shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nSubChannels = 0`)	`LREAL, 1,` `nChannels`	`LREAL, 1,` `nChannels`
Standardvariante für mehrere Datensätze (`nSubChannels = 0`)	`LREAL, 2,` `nChannels x` *not specified**	`LREAL, 1,` `nChannels`
Mehrkanalige Variante (`nSubChannels > 0`)	`LREAL, 2,` `nChannels x nSubChannels`	`LREAL, 2,` `nChannels x nSubChannels`

Mehrkanalige Variante für mehrere Datensätze
(`nSubChannels > 0`)
	`LREAL, 3,`
	`nChannels x nSubChannels x`
	*not specified**
		`LREAL, 2,`
		`nChannels x nSubChannels`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorrausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_EmpiricalMoments_InitPars;  // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;           // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_EmpiricalMoments_InitPars [▶ 301]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError          : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

CallEx():

Die Methode wird jeden Zyklus aufgerufen um den internen Speicher aus dem Eingangssignal zu aktualisieren. Ein Ergebnis wird nur alle `nAppendData` Zyklen ausgegeben. Eine alternative Methode ist `Call()`.

Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD CallEx : HRESULT
VAR_INPUT
    nAppendData  : UDINT;      // count of data buffers which are appended until calculation (1= cal
culate always)
    bResetData   : BOOL;       // automatic reset of dataset buffer after each calculation
END_VAR
VAR_OUTPUT
    bNewResult   : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError       : BOOL;        // TRUE if an error occurs.
    hrErrorCode  : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `nAppendData`: Legt fest wie viele Eingangsdatenpuffer gesammelt werden sollen bevor eine Berechnung ausgeführt wird, denn vorzugsweise werden mehrere Datenblöcke angefügt um ein präzises Ergebnis zu erreichen.
- `bResetData`: Falls gesetzt, wird der interne Datenpuffer nach Berechnung vollständig gelöscht.
- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_EmpiricalMoments_InitPars;  // init parameter
    nOwnID         : UDINT;                              // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers: UDINT := 4;                          // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_EmpiricalMoments_InitPars [▶ 301]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

Alternativ kann der automatische Reset in der Methode `CallEx()` verwendet werden.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_EmpiricalStandardDeviation [▶ 141] berechnet die empirische Standardabweichung von Eingangswerten.

Der Baustein FB_CMA_EmpiricalSkew [▶ 136] berechnet die empirische Schiefe von Eingangswerten.

Der Baustein FB_CMA_EmpiricalExcess [▶ 126] berechnet den empirischen Exzess von Eingangswerten.

Der Baustein FB_CMA_MomentCoefficients [▶ 183] berechnet je nach Parametrierung, sowohl den empirischen Mittelwert, also auch Standardabweichung, Schiefe und Exzess.

Der Baustein FB_CMA_HistArray [▶ 154] berechnet die Histogramme von Verteilungen von Eingangswerten.

Der Baustein FB_CMA_Quantiles [▶ 206] berechnet die Quantile einer empirischen Verteilung, die ebenfalls erlauben die Häufigkeit von Ausreißern zu beurteilen.

Der Baustein FB_CMA_CrestFactor [▶ 99] berechnet ein zur Kurtosis alternatives Maß für die Spitzenhaltigkeit (Crest Faktor) eines Signals, das allerdings empfindlicher gegenüber Ausreißern ist.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.13 FB_CMA_EmpiricalSkew

Berechnet die empirische Schiefe für ein- und mehrkanalige reell-wertige Zeitreihen.

Der Baustein behandelt das Eingangssignal als Zeitreihe mit ggf. mehreren unabhängigen Kanälen. Für jeden Kanal wird die empirische Schiefe nach Gleichung

$$v = \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{x[n]-\overline{x}}{s}\right)^3$$

berechnet, wobei *s* die empirische Standardabweichung ist. Die Schiefe quantifiziert die Asymmetrie einer Stichprobe. Sie bietet eine zum Exzess ergänzende Möglichkeit zur Beurteilung von Stößen (z.B. durch Überrollen von lokalen Schäden im Wälzlager) in einem Vibrationssignal. Die Schiefe ist als Kriterium robuster zu berechnen als die Kurtosis/der Exzess, jedoch sind Verteilungen von Signalen aufgrund von lokalen Schädigungen nicht immer asymmetrisch.

Es kann sowohl in jedem Zyklus ein einzelnes Sample pro Kanal (siehe Ein- und Ausgänge, erste Tabelle) als auch in einem Zyklus mehrere Samples pro Kanal zur Stichprobenmenge hinzugefügt werden (siehe Ein- und Ausgänge, zweite Tabelle).

Weitere Anmerkungen

Zur Berechnung eines ersten Ergebnisses müssen 3 Werte vorliegen. Des Weiteren darf die Standardabweichung nicht Null sein. Ergebnisse können unter Umständen ungenau werden, wenn die Eingangswerte viele Größenordnungen auseinanderliegen.

Gedächtniseigenschaften

Die Stichprobenmenge *N*, welche zur Berechnung des aktuellen Ergebnisses genutzt wird, erhöht sich automatisch bei jedem neuen eingehenden Datensatz, d.h. der Baustein verwendet alle Eingangswerte seit dessen Instanziierung. Das Zurücksetzen der Stichprobenmenge auf Null (löschen des internen Speichers des FBs) ist über eine `ResetData()` Methode oder, bei Verwendung der `CallEx()` Methode über die Variable `bResetData` vorgesehen.

NaN Vorkommnis

Falls für einen Kanal noch keine ausreichende Anzahl Eingangswerte für die Berechnung eines Ergebnisses vorliegt oder die Varianz Null beträgt, wird für diesen Kanal der Wert NaN (Not A Number) nach IEEE 754 zurückgegeben. Das Vorliegen dieses Fehlerwertes kann mit der Funktion LrealIsNaN() überprüft werden. Ursache dafür kann entweder sein, dass noch nicht genug Eingangsdaten übergeben wurden oder dass für einzelne Kanäle bisher lediglich NaNs als Eingangswerte übergeben wurden. Eine Varianz von Null kann insbesondere auftreten, wenn die Zeitreihe der Werte konstant ist, etwa wenn aufgrund von Kabelbruch oder Schaltfehlern keine Sensordaten übertragen werden.

Enthält ein Satz von Eingangswerten die spezielle Konstante NaN, so wird der Statistik für diesen Kanal bei diesem Zeitschritt kein Wert hinzugefügt, d.h. sie wird als Indikator für fehlende Werte behandelt.

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Statistische Methoden [▶ 345]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen folgender Definition (Shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nSubChannels = 0`)	`LREAL, 1,` `nChannels`	`LREAL, 1,` `nChannels`
Standardvariante für mehrere Datensätze (`nSubChannels = 0`)	`LREAL, 2,` `nChannels x` *not specified**	`LREAL, 1,` `nChannels`
Mehrkanalige Variante (`nSubChannels > 0`)	`LREAL, 2,` `nChannels x nSubChannels`	`LREAL, 2,` `nChannels x nSubChannels`

Mehrkanalige Variante für mehrere Datensätze
(`nSubChannels > 0`)
	`LREAL, 3,`
	`nChannels x nSubChannels x`
	*not specified**
		`LREAL, 2,`
		`nChannels x nSubChannels`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorrausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_EmpiricalMoments_InitPars;  // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;           // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_EmpiricalMoments_InitPars [▶ 301]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError          : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
```

```
    nCntResults     : ULINT;                          // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult    : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError        : BOOL;       // TRUE if an error occurs.
    hrErrorCode   : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

CallEx():

Die Methode wird jeden Zyklus aufgerufen um den internen Speicher aus dem Eingangssignal zu aktualisieren. Ein Ergebnis wird nur alle `nAppendData` Zyklen ausgegeben. Eine alternative Methode ist `Call()`.

Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD CallEx : HRESULT
VAR_INPUT
    nAppendData   : UDINT;      // count of data buffers which are appended until calculation (1= cal
culate always)
    bResetData    : BOOL;       // automatic reset of dataset buffer after each calculation
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError        : BOOL;        // TRUE if an error occurs.
    hrErrorCode   : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `nAppendData`: Legt fest wie viele Eingangsdatenpuffer gesammelt werden sollen bevor eine Berechnung ausgeführt wird, denn vorzugsweise werden mehrere Datenblöcke angefügt um ein präzises Ergebnis zu erreichen.
- `bResetData`: Falls gesetzt, wird der interne Datenpuffer nach Berechnung vollständig gelöscht.
- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.

- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_EmpiricalMoments_InitPars;  // init parameter
    nOwnID         : UDINT;                              // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers: UDINT := 4;                          // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_EmpiricalMoments_InitPars [▶ 301]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

Alternativ kann der automatische Reset in der Methode `CallEx()` verwendet werden.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am

Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_EmpiricalMean [▶ 131] berechnet den empirischen Mittelwert von Eingangswerten.

Der Baustein FB_CMA_EmpiricalStandardDeviation [▶ 141] berechnet die empirische Standardabweichung von Eingangswerten.

Der Baustein FB_CMA_EmpiricalExcess [▶ 126] berechnet den empirischen Exzess von Eingangswerten.

Der Baustein FB_CMA_MomentCoefficients [▶ 183] berechnet je nach Parametrierung, sowohl den empirischen Mittelwert, also auch Standardabweichung, Schiefe und Exzess.

Der Baustein FB_CMA_HistArray [▶ 154] berechnet die Histogramme von Verteilungen von Eingangswerten.

Der Baustein FB_CMA_Quantiles [▶ 206] berechnet die Quantile einer empirischen Verteilung, die ebenfalls erlauben die Häufigkeit von Ausreißern zu beurteilen.

Der Baustein FB_CMA_CrestFactor [▶ 99] berechnet ein zur Kurtosis alternatives Maß für die Spitzenhaltigkeit (Crest Faktor) eines Signals, das allerdings empfindlicher gegenüber Ausreißern ist.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.14 FB_CMA_EmpiricalStandardDeviation

Berechnet die empirische Standardabweichung für ein- und mehrkanalige reell-wertige Zeitreihen.

Der Baustein behandelt das Eingangssignal als Zeitreihe mit ggf. mehreren unabhängigen Kanälen. Für jeden Kanal wird die empirische Standardabweichung nach Gleichung

$$s = \sqrt{\frac{1}{N-1}\sum_{n=0}^{N-1}(x[n]-\overline{x})^2}$$

berechnet. Die Bessel'sche Korrektur wird, im Gegensatz zum FB_CMA_MomentCoefficients [▶ 183], immer angewandt. Es kann sowohl in jedem Zyklus ein einzelnes Sample pro Kanal (siehe Ein- und Ausgänge, erste Tabelle) als auch in einem Zyklus mehrere Samples pro Kanal zur Stichprobenmenge hinzugefügt werden (siehe Ein- und Ausgänge, zweite Tabelle).

Weitere Anmerkungen

Zur Berechnung eines ersten Ergebnisses müssen 2 Werte vorliegen. Ergebnisse können unter Umständen ungenau werden, wenn die Eingangswerte viele Größenordnungen auseinanderliegen.

Gedächtniseigenschaften

Die Stichprobenmenge *N*, welche zur Berechnung des aktuellen Ergebnisses genutzt wird, erhöht sich automatisch bei jedem neuen eingehenden Datensatz, d.h. der Baustein verwendet alle Eingangswerte seit dessen Instanziierung. Das Zurücksetzen der Stichprobenmenge auf Null (löschen des internen Speichers des FBs) ist über eine `ResetData()` Methode oder, bei Verwendung der `CallEx()` Methode über die Variable `bResetData` vorgesehen.

NaN Vorkommnis

Falls für einen Kanal noch keine ausreichende Anzahl Eingangswerte für die Berechnung eines Ergebnisses vorliegt oder die Varianz Null beträgt, wird für diesen Kanal der Wert NaN (Not A Number) nach IEEE 754 zurückgegeben. Das Vorliegen dieses Fehlerwertes kann mit der Funktion LrealIsNaN() überprüft werden. Ursache dafür kann entweder sein, dass noch nicht genug Eingangsdaten übergeben wurden oder dass für einzelne Kanäle bisher lediglich NaNs als Eingangswerte übergeben wurden. Eine Varianz von Null kann insbesondere auftreten, wenn die Zeitreihe der Werte konstant ist, etwa wenn aufgrund von Kabelbruch oder Schaltfehlern keine Sensordaten übertragen werden.

Enthält ein Satz von Eingangswerten die spezielle Konstante NaN, so wird der Statistik für diesen Kanal bei diesem Zeitschritt kein Wert hinzugefügt, d.h. sie wird als Indikator für fehlende Werte behandelt.

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Statistische Methoden [▶ 345]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen folgender Definition (Shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nSubChannels = 0`)	`LREAL, 1,` `nChannels`	`LREAL, 1,` `nChannels`
Standardvariante für mehrere Datensätze (`nSubChannels = 0`)	`LREAL, 2,` `nChannels x` *not specified**	`LREAL, 1,` `nChannels`
Mehrkanalige Variante (`nSubChannels > 0`)	`LREAL, 2,` `nChannels x nSubChannels`	`LREAL, 2,` `nChannels x nSubChannels`
Mehrkanalige Variante für mehrere Datensätze (`nSubChannels > 0`)	`LREAL, 3,` `nChannels x nSubChannels x` *not specified**	`LREAL, 2,` `nChannels x nSubChannels`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorrausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_EmpiricalMoments_InitPars;  // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;           // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_EmpiricalMoments_InitPars [▶ 301]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError          : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
```

```
    nCntResults     : ULINT;                          // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult    : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError        : BOOL;        // TRUE if an error occurs.
    hrErrorCode   : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

CallEx():

Die Methode wird jeden Zyklus aufgerufen um den internen Speicher aus dem Eingangssignal zu aktualisieren. Ein Ergebnis wird nur alle `nAppendData` Zyklen ausgegeben. Eine alternative Methode ist `Call()`.

Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD CallEx : HRESULT
VAR_INPUT
    nAppendData   : UDINT;       // count of data buffers which are appended until calculation (1= cal
culate always)
    bResetData    : BOOL;        // automatic reset of dataset buffer after each calculation
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError        : BOOL;         // TRUE if an error occurs.
    hrErrorCode   : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `nAppendData`: Legt fest wie viele Eingangsdatenpuffer gesammelt werden sollen bevor eine Berechnung ausgeführt wird, denn vorzugsweise werden mehrere Datenblöcke angefügt um ein präzises Ergebnis zu erreichen.
- `bResetData`: Falls gesetzt, wird der interne Datenpuffer nach Berechnung vollständig gelöscht.
- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.

- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_EmpiricalMoments_InitPars;  // init parameter
    nOwnID         : UDINT;                              // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers: UDINT := 4;                          // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_EmpiricalMoments_InitPars [▶ 301]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

Alternativ kann der automatische Reset in der Methode `CallEx()` verwendet werden.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am

Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_EmpiricalMean [▶ 131] berechnet den empirischen Mittelwert von Eingangswerten.

Der Baustein FB_CMA_EmpiricalSkew [▶ 136] berechnet die empirische Schiefe von Eingangswerten.

Der Baustein FB_CMA_EmpiricalExcess [▶ 126] berechnet den empirischen Exzess von Eingangswerten.

Der Baustein FB_CMA_MomentCoefficients [▶ 183] berechnet je nach Parametrierung, sowohl den empirischen Mittelwert, also auch Standardabweichung, Schiefe und Exzess.

Der Baustein FB_CMA_HistArray [▶ 154] berechnet die Histogramme von Verteilungen von Eingangswerten.

Der Baustein FB_CMA_Quantiles [▶ 206] berechnet die Quantile einer empirischen Verteilung, die ebenfalls erlauben die Häufigkeit von Ausreißern zu beurteilen.

Der Baustein FB_CMA_CrestFactor [▶ 99] berechnet ein zur Kurtosis alternatives Maß für die Spitzenhaltigkeit (Crest Faktor) eines Signals, das allerdings empfindlicher gegenüber Ausreißern ist.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.15 FB_CMA_Envelope

Berechnet die Einhüllende eines Zeitsignals.

Die Einhüllende ist im mathematischen Sinn definiert als Absolutbetrag des analytischen Signals, siehe FB_CMA_AnalyticSignal [▶ 89]. In der zeitkontinuierlichen Darstellung ist die Einhüllende $x_{env}(t)$ des Signals $x(t)$ definiert als:

$$x_{\mathrm{env}}(t) = |x_{\mathrm{analytic}}(t)| \quad , \quad x_{\mathrm{analytic}}(t) = x(t) + \mathrm{i}\mathcal{H}[x(t)]$$

Die Einhüllende kann z.B. als amplitudenmodulierter Anteil des Signals x(t) interpretiert werden

$$x(t) = x_{\mathrm{env}}(t) \cos \varphi(t)$$

Der phasenmodulierte Anteil φ(t) kann ebenfalls berechnet werden, siehe FB_CMA_InstantaneousPhase [▶ 164]. Mit der Einhüllenden können z. B. Aufschwing- bzw. Abklingvorgänge effizient bewertet werden.

Die diskrete Berechnung der Einhüllenden mit dem Funktionsbaustein erfolgt hier effizient im Frequenzbereich. Der Eingabevektor wird zunächst mit dem unmittelbar vorhergehenden Vektor überlappend kombiniert und mit einer Fensterfunktion multipliziert. Anschließend wird eine FFT für reelle Eingangswerte angewandt. Im Frequenzbereich wird auf das Signal die Hilbert-Transformation angewandt und das Ergebnis in den Zeitbereich zurück transformiert. Von den resultierenden komplexen Werten wird der Absolutbetrag berechnet. Das Zeitsignal wird mit Hilfe der Overlap-Add Methode überlappend aufaddiert. Durch die Auswahl geeigneter Fensterfunktionen ergibt sich ein kontinuierliches Ausgangssignal ohne Sprünge.

Die Einhüllende liefert nur gültige Ergebnisse für mittelwertfreie Signale. Sollte ein mittelwertbehaftetes Signal analysiert werden, ist der Signalmittelwert vorher zu subtrahieren und auf das Ergebnis des Bausteins mit dem zuvor subtrahierten Wert wieder zu addieren.

Fensterlänge beachten

Der Wert von `nWindowLength` muss kleiner oder gleich dem Wert von `nFFT_Length` sein. Die Länge der FFT kann sich an der benötigten Frequenzauflösung orientieren. Typischerweise wird oft ein Wert von ca. 4/5 der FFT-Länge als Fensterlänge verwendet.

Wenn `nFFT_Length` größer ist als `nWindowLength`, wird die Frequenzauflösung der FFT (und damit auch die Länge des Vektors der Rückgabewerte) vergrößert. Die Differenz der Länge wird vor der Fouriertransformation mit Nullen aufgefüllt. Dies kann sinnvoll sein, um eine höhere Frequenzauflösung zu erreichen oder, z.B. bei der Berechnung mit Rücktransformation in den Zeitbereich, zirkuläres Aliasing zu vermeiden. Das Ergebnis enthält trotz der höheren Frequenzauflösung allerdings nicht mehr Information.

Gedächtniseigenschaften

Aufgrund der Verwendung der Overlap-Add-Methode werden jeweils der aktuelle Eingangspuffer zusammen mit den zwei zuletzt übergebenen Puffern zur Berechnung genutzt.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1, nWindowLength/2`	`LREAL, 1, nWindowLength/2`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2, nChannels x nWindowLength/2`	`LREAL, 2, nChannels x nWindowLength/2`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_Envelope_InitPars;          // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_Envelope_InitPars [▶ 301]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult    : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError        : BOOL;       // TRUE if an error occurs.
    hrErrorCode   : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_Envelope_InitPars;          // init parameter
    nOwnID         : UDINT;                            // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;                       // number of MultiArrays which should be init
ialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_Envelope_InitPars [▶ 301]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_AnalyticSignal [▶ 89] berechnet das analytische Signal einer Zeitreihe.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.16 FB_CMA_EnvelopeSpectrum

Berechnet das Einhüllendenspektrum eines Zeitsignals.

Das Einhüllendenspektrum ist ein kombinierter Baustein aus FB_CMA_Envelope [▶ 146] und FB_CMA_PowerSpectrum [▶ 202]. Entsprechend wird zunächst die Signaleinhüllende eines Zeitsignals berechnet und dann das Leistungsspektrum gebildet. Der Baustein hat große Bedeutung bei der frequenzaufgelösten Analyse von Wälzlagerschädigungen, siehe Wälzlagerüberwachung [▶ 41].

Die intern verwendete Fensterlänge und Länge der FFT für die Berechnung der Signaleinhüllenden berechnet sich in Abhängigkeit der gewählten Fensterfunktion und zugehöriger Überlappung (`nOverlap`), beziehend auf die Berechnung des Leistungsspektrum. Hierbei gilt für die Fensterlänge `L = 2*(nWindowLength-nOverlap)`. Die FFT Länge `N` entspricht der nächst größeren Potenz von 2, für die gilt: `N >= 4*L/3`.

Fensterlänge beachten

Der Wert von `nWindowLength` muss kleiner oder gleich dem Wert von `nFFT_Length` sein. Die Länge der FFT kann sich an der benötigten Frequenzauflösung orientieren. Typischerweise wird oft ein Wert von ca. 4/5 der FFT-Länge als Fensterlänge verwendet.

Wenn `nFFT_Length` größer ist als `nWindowLength`, wird die Frequenzauflösung der FFT (und damit auch die Länge des Vektors der Rückgabewerte) vergrößert. Die Differenz der Länge wird vor der Fouriertransformation mit Nullen aufgefüllt. Dies kann sinnvoll sein, um eine höhere Frequenzauflösung zu erreichen oder, z.B. bei der Berechnung mit Rücktransformation in den Zeitbereich, zirkuläres Aliasing zu vermeiden. Das Ergebnis enthält trotz der höheren Frequenzauflösung allerdings nicht mehr Information.

Gedächtniseigenschaften

Aufgrund der Verwendung der Overlap-Add-Methode werden jeweils der aktuelle Eingangspuffer zusammen mit den zwei zuletzt übergebenen Puffern zur Berechnung genutzt.

Dieser Delay Effekt vergrößert sich gegebenen falls bei der Verwendung einiger Fensterfunktionen mit weiter Überlappung (`nOverlap`) bei der Berechnung des Leistungsspektrums.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Einhüllendenspektrum [▶ 363].

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1, nWindowLength-nOverlap`	`LREAL, 1, nFFT_Length/2+1`

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2, nChannels x nWindowLength-nOverlap`	`LREAL, 2, nChannels x nFFT_Length/2+1`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_EnvelopeSpectrum_InitPars;    // init parameter
    nOwnID           : UDINT;                              // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                         // number of MultiArrays which should be
initialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;               // timeout checking off during access to
inter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_EnvelopeSpectrum_InitPars [▶ 302]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                            // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                         // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;   // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                           // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_EnvelopeSpectrum_InitPars;  // init parameter
    nOwnID    : UDINT;                                 // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;                        // number of MultiArrays which should be init
ialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_EnvelopeSpectrum_InitPars [▶ 302]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_Envelope [▶ 146] berechnet die Einhüllende einer Zeitreihe.

Der Baustein FB_CMA_PowerSpectrum [▶ 202] berechnet das Leistungsspektrum mit einer Quadrierung der Werte im letzten Schritt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.17 FB_CMA_HistArray

Berechnet die Häufigkeitsverteilung für ein- und mehrkanalige reell-wertige Zeitreihen.

Der Baustein FB_CMA_HistArray berechnet die Häufigkeitsverteilung (in der graphischen Darstellung Histogramm genannt) von ein- und auch mehrkanaligen reell-wertigen Eingangsdaten. Jeder Kanal wird dabei unabhängig voneinander betrachtet. Für jeden einzelnen Kanal wird die Häufigkeitsverteilung der zyklisch eingehenden Datenpuffer berechnet, wobei als Eingangspuffer sowohl Einzelwerte oder auch Arrays zugelassen sind.

Zur Parametrierung werden der untere und obere Grenzwert sowie die Anzahl der Klassen (auch Bins genannt) übergeben. Die einzelnen Klassengrenzen werden dann im dadurch definierten Gesamtbereich in gleich großen Intervallen verteilt, vgl. Histogramm [▶ 29]. Werte, die unterhalb der unteren Grenze, sowie Werte, die oberhalb der oberen Grenze liegen, werden jeweils in zwei zusätzlichen Bins gezählt.

Der Rückgabewert ist ein zweidimensionales Array mit unsigned 64-Bit Integer-Werten. Der erste Index ist die Nummer des Kanals und der zweite Index ist die Nummer des betreffenden Histogramm-Bins. Dabei sind die Zählungen der Elemente mit einem Wert unterhalb des unteren Grenzwerts, bzw. oberhalb des oberen Grenzwerts jeweils im ersten bzw. letzten Bin enthalten.

Wenn ein Histogrammzähler einen Wert von 2 hoch 64, ungefähr 1.8E19, überschreitet, läuft in der aktuellen Implementation der Zähler über, ohne dass eine Fehlermeldung erzeugt wird. Bei einem Zählschritt alle 100 Mikrosekunden geschähe dies frühestens nach 59 Millionen Jahren.

Gedächtniseigenschaften

Die Stichprobenmenge *N*, welche zur Berechnung des aktuellen Ergebnisses genutzt wird, erhöht sich automatisch bei jedem neuen eingehenden Datensatz, d.h. der Baustein verwendet alle Eingangswerte seit dessen Instanziierung. Das Zurücksetzen der Stichprobenmenge auf Null (löschen des internen Speichers des FBs) ist über eine `ResetData()` Methode oder, bei Verwendung der `CallEx()` Methode über die Variable `bResetData` vorgesehen.

Konfiguration

Mit den Initialisierungsparametern werden bereits die Grenzwerte, für die Samples in den regulären Histogramm-Bins gezählt werden, festgelegt. Diese können mit der `Configure()` Methode für jeden Kanal individuell angepasst werden.

NaN Vorkommnis

Enthält ein Satz von Eingangswerten die spezielle Konstante NaN, so wird der Statistik für diesen Kanal bei diesem Zeitschritt kein Wert hinzugefügt, d.h. sie wird als Indikator für fehlende Werte behandelt. Am Ausgang sind keine NaN Werte zu erwarten.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Histogramm [▶ 343].

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen folgender Definition (Shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nSubChannels = 0`)	`LREAL, 1,` `nChannels`	`ULINT, 2,` `nChannels x nBins+2`
Standardvariante für mehrere Datensätze (`nSubChannels = 0`)	`LREAL, 2,` `nChannels x` *not specified**	`ULINT, 2,` `nChannels x nBins+2`
Mehrkanalige Variante (`nSubChannels > 0`)	`LREAL, 2,` `nChannels x nSubChannels`	`ULINT, 3,` `nChannels x nSubChannels x nBins+2`
Mehrkanalige Variante für mehrere Datensätze (`nSubChannels > 0`)	`LREAL, 3,` `nChannels x nSubChannels x` *not specified**	`ULINT, 3,` `nChannels x nSubChannels x nBins+2`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorrausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_HistArray_InitPars;         // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_HistArray_InitPars [▶ 303]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError          : BOOL;                            // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                         // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                           // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

CallEx() :

Die Methode wird jeden Zyklus aufgerufen um das Histogramm aus dem Eingangssignal zu berechnen. Eine alternative Methode ist `Call()`.
Die Auswertung des Histogramms ist in der Regel erheblich rechenaufwendiger als die Registrierung neuer Eingangswerte. Deswegen kann eine Benutzung der Methode `CallEx()` die Laufzeit je nach den konfigurierten Parametern erheblich verkürzen, indem statistische Ergebnisse erst berechnet werden, wenn sie benötigt werden.

Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD CallEx : HRESULT
VAR_INPUT
    nAppendData  : UDINT;      // count of data buffers which are appended until calculation (1= cal
culate always)
    bResetData   : BOOL;       // automatic reset of dataset buffer after each calculation
END_VAR
VAR_OUTPUT
    bNewResult   : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError       : BOOL;        // TRUE if an error occurs.
    hrErrorCode  : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `nAppendData`: Legt fest wie viele Eingangsdatenpuffer gesammelt werden sollen bevor eine Berechnung ausgeführt wird, denn vorzugsweise werden mehrere Datenblöcke angefügt um ein präzises Ergebnis zu erreichen.
- `bResetData`: Falls gesetzt, wird der interne Datenpuffer nach Berechnung vollständig gelöscht.
- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Configure() :

Mit dem Aufruf dieser Methode können die Histogrammargumente neu konfiguriert werden. Dies ermöglicht eine feine Einstellung der Parameter `fMinBinned` und `fMaxBinned` während der Laufzeit. Das entsprechende SPS Array muss wie folgt definiert sein.

- **Rückgabewert:** Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : POINTER TO LREAL;  // pointer to 2-dimensional array (LREAL) of arguments
    nArgSize : UDINT;             // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`. Die zwei zu konfigurierenden Parameter je Kanal und Unterkanal sind `[fMinBinned, fMaxBinned]`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle und Unterkanäle	`LREAL, 1, 2`
Kanalspezifische Konfiguration (`nSubChannels = 0`)	`LREAL, 2, nChannels x 2`
Unterkanalspezifische Konfiguration (`nSubChannels > 0`)	`LREAL, 2, nSubChannels x 2`
Kanal- und unterspezifische Konfiguration (`nSubChannels > 0`)	`LREAL, 3, nChannels x nSubChannels x 2`

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_HistArray_InitPars;         // init parameter
    nOwnID         : UDINT;       // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;  // number of MultiArrays which should be initialized for results (
0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_HistArray_InitPars [▶ 303]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.

- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

Alternativ kann der automatische Reset in der Methode `CallEx()` verwendet werden.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_Quantiles [▶ 206] berechnet die Quantile von Verteilungen von Eingangswerten.

Der Baustein FB_CMA_MomentCoefficients [▶ 183] berechnet die statistischen Momentenkoeffizienten : Mittelwert, Standardabweichung, Schiefe (Skew) und Kurtosis.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.18 FB_CMA_InstantaneousFrequency

Berechnung der Momentanfrequenz eines Zeitsignals

Die Momentanfrequenz bzw. Instantfrequenz ist im mathematischen Sinn definiert als zeitliche Ableitung der Momentanphase, siehe FB_CMA_InstantaneousPhase [▶ 164]. In der zeitkontinuierlichen Darstellung ist die Momentanfrequenz ω(t) des Signals x(t) definiert als:

$$\omega(t) = \frac{\mathrm{d}}{\mathrm{d}t}\varphi(t) = \frac{\mathrm{d}}{\mathrm{d}t}\arctan\frac{\mathcal{H}[x(t)]}{x(t)} \quad , \quad x_{\mathrm{analytic}} = x(t) + \mathrm{i}\mathcal{H}[x(t)]$$

Die Momentanfrequenz kann z.B. als frequenzmodulierter Anteil des Signals x(t) interpretiert werden

$$x(t) = x_{\mathrm{env}}(t)\cos\varphi(t) = x_{\mathrm{env}}(t)\cos\int\omega(t)\mathrm{d}t$$

Somit lässt sich das Signal x(t) durch die Berechnung der Momentanfrequenz und der Einhüllenden [▶ 146] in die amplituden- und frequenzmodulierte Darstellung transformieren.

Die Funktionsbausteine Momentanphase und Momentanfrequenz liefern nur gültige Ergebnisse für mittelwertfreie Signale. Sollte ein mittelwertbehaftetes Signal analysiert werden, ist der Signalmittelwert vorher zu subtrahieren.

Die Momentanfrequenz eignet sich gut zur Analyse von Drehschwingungen (Torsionsschwingungen) an einer Kurbelwelle. Torsionsschwingungen entstehen u.a. durch ein fluktuierendes Drehmoment und resultieren in einer Frequenzmodulation auf einer ansonsten gleichförmigen Drehzahl.

Fensterlänge beachten

Der Wert von `nWindowLength` muss kleiner oder gleich dem Wert von `nFFT_Length` sein. Die Länge der FFT kann sich an der benötigten Frequenzauflösung orientieren. Typischerweise wird oft ein Wert von ca. 4/5 der FFT-Länge als Fensterlänge verwendet.

Wenn `nFFT_Length` größer ist als `nWindowLength`, wird die Frequenzauflösung der FFT (und damit auch die Länge des Vektors der Rückgabewerte) vergrößert. Die Differenz der Länge wird vor der Fouriertransformation mit Nullen aufgefüllt. Dies kann sinnvoll sein, um eine höhere Frequenzauflösung zu erreichen oder, z.B. bei der Berechnung mit Rücktransformation in den Zeitbereich, zirkuläres Aliasing zu vermeiden. Das Ergebnis enthält trotz der höheren Frequenzauflösung allerdings nicht mehr Information.

Gedächtniseigenschaften

Aufgrund der Verwendung der Overlap-Add-Methode werden jeweils der aktuelle Eingangspuffer zusammen mit den zwei zuletzt übergebenen Puffern zur Berechnung genutzt.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1, nWindowLength/2`	`LREAL, 1, nWindowLength/2`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2, nChannels x nWindowLength/2`	`LREAL, 2, nChannels x nWindowLength/2`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_InstantaneousFrequency_InitPars;  // init parameter
    nOwnID           : UDINT;                                  // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;        // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                             // number of MultiArrays which should
 be initialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;                   // timeout checking off during access
 to inter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_InstantaneousFrequency_InitPars [▶ 303]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_InstantaneousFrequency_InitPars;       // init parameter
    nOwnID         : UDINT;                                   // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_InstantaneousFrequency_InitPars [▶ 303]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.

- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_InstantaneousPhase [▶ 164] berechnet die Momentanphase eines Zeitsignals.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.19 FB_CMA_InstantaneousPhase

Berechnung der Momentanphase eines Zeitsignals

Die Momentanphase oder Instantanphase $\varphi(t)$ eines Signals $x(t)$ ist über die Phase des analytischen Signals, siehe FB_CMA_AnalyticSignal [▶ 89], definiert:

$$\varphi(t) = \arctan \frac{\mathcal{H}[x(t)]}{x(t)} \quad , \quad x_{\text{analytic}} = x(t) + \mathrm{i}\mathcal{H}[x(t)]$$

Die Momentanphase kann als phasenmodulierter Anteil des Signals x(t) interpretiert werden:

$$x(t) = x_{\text{env}}(t) \cos \varphi(t)$$

Der Amplitudenmodulierte Anteil (Einhüllende) des Signals kann ebenfalls bestimmt werden, siehe FB_CMA_Envelope [▶ 146].

Die Funktionsbausteine Momentanphase und Momentanfrequenz liefern nur gültige Ergebnisse für mittelwertfreie Signale. Sollte ein mittelwertbehaftetes Signal analysiert werden, ist der Signalmittelwert vorher zu subtrahieren.

Fensterlänge beachten

Der Wert von `nWindowLength` muss kleiner oder gleich dem Wert von `nFFT_Length` sein. Die Länge der FFT kann sich an der benötigten Frequenzauflösung orientieren. Typischerweise wird oft ein Wert von ca. 4/5 der FFT-Länge als Fensterlänge verwendet.

Wenn `nFFT_Length` größer ist als `nWindowLength`, wird die Frequenzauflösung der FFT (und damit auch die Länge des Vektors der Rückgabewerte) vergrößert. Die Differenz der Länge wird vor der Fouriertransformation mit Nullen aufgefüllt. Dies kann sinnvoll sein, um eine höhere Frequenzauflösung zu erreichen oder, z.B. bei der Berechnung mit Rücktransformation in den Zeitbereich, zirkuläres Aliasing zu vermeiden. Das Ergebnis enthält trotz der höheren Frequenzauflösung allerdings nicht mehr Information.

Gedächtniseigenschaften

Aufgrund der Verwendung der Overlap-Add-Methode werden jeweils der aktuelle Eingangspuffer zusammen mit den zwei zuletzt übergebenen Puffern zur Berechnung genutzt.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1, nWindowLength/2`	`LREAL, 1, nWindowLength/2`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2, nChannels x nWindowLength/2`	`LREAL, 2, nChannels x nWindowLength/2`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_InstantaneousPhase_InitPars;  // init parameter
    nOwnID           : UDINT;                              // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                         // number of MultiArrays which should be
initialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;               // timeout checking off during access to
inter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_InstantaneousPhase_InitPars [▶ 304]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult    : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError        : BOOL;       // TRUE if an error occurs.
    hrErrorCode   : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_InstantaneousPhase_InitPars;       // init parameter
    nOwnID         : UDINT;                                 // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_InstantaneousPhase_InitPars [▶ 304]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_InstantaneousFrequency [▶ 160] berechnet die Momentanfrequenz eines Zeitsignals.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.20 FB_CMA_IntegratedRMS

Berechnet (optional integrierte) RMS Werte für ein- und mehrkanalige reell-wertige Zeitreihen.

Berechnet für ein- und mehrkanalige Zeitreihen den RMS Wert, wobei sowohl der genutzte Frequenzbereich als auch die Integrationsordnung der Zeitreihe definiert werden kann. Für ein Beschleunigungssignal ergeben sich somit die Effektivwerte der Schwingbeschleunigung, Schwinggeschwindigkeit und der Schwingungsamplitude, jeweils in einem definierten Frequenzbereich.

Der Baustein behandelt das Eingangssignal als Signal mit mehreren unabhängigen Kanälen. Für jeden Kanal werden im Frequenzbereich die Werte für bis zu drei verschiedene Integrationsordnungen über einem definierten Frequenzintervall integriert und anschließend die Effektivwerte berechnet. Der Baustein eignet sich für die Schwingungsbeurteilung nach DIN ISO 10816 und DIN ISO 7919 bzw. DIN ISO 20816, siehe Schwingungsbeurteilung [▶ 35].

Die Abtastrate und die Grenzen des integrierten Intervalls sind parametrierbar. Um eine nachvollziehbare Skalierung zu erhalten, müssen die Eingangssignale sowie die Frequenzen in SI-Einheiten skaliert übergeben werden, also 1 m/(sec)² für Beschleunigungswerte und 1/sec = 1 Hz für Frequenzen. Die maximale Ordnung der Integration ist konfigurierbar zwischen Null und Zwei. Die Anzahl der zu berechnenden integrierten RMS Werte ist mittels (`nOrder+1`) anzugeben. Das Ergebnis wird als Array dieser Werte mit den entsprechenden Indizes weitergegeben.

Das zugrundeliegende Kurzzeit-Leistungsspektrum ist in vielen Fällen kein guter statistischer Schätzer für das Spektrum eines Signals, so dass die Werte trotz Mittelung über Frequenzen dennoch fluktuieren können. Daher empfiehlt sich die Verwendung einer ausreichend großen Fensterlänge. In vielen Fällen ist es möglicherweise zusätzlich sinnvoll, die Fluktuation durch Bildung des geometrischen Mittels über mehrere aufeinanderfolgende Werte weiter zu verringern.

Gedächtniseigenschaften

Aufgrund der Verwendung der Welch-Methode wird jeweils der aktuelle Eingangsdatenpuffer zusammen mit den zuletzt übergebenen Puffern zur Berechnung genutzt. Die Anzahl der einfließenden Puffer ist abhängig von der gewählten Überlappung (`nOverlap`).
Die Frequenzanalyse berücksichtigt Sprünge in der Zeitreihe. Um ein korrektes Ergebnis zu erzielen, müssen sich deswegen die verwendeten Eingangsdatenpuffer lückenlos und ohne Sprünge aneinanderreihen.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nMaxBands = 1`)	`LREAL, 2,` `nChannels x nWindowLength-nOverlap`	`LREAL, 2,` `nChannels x nOrder+1`

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Variante für mehrere Frequenzbänder (`nMaxBands > 1`)	`LREAL, 2, nChannels x nWindowLength-nOverlap`	`LREAL, 3, nChannels x nOrder+1 x nMaxBands`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_IntegratedRMS_InitPars;     // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_IntegratedRMS_InitPars [▶ 304]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                            // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                         // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;   // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                           // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Configure():

Mit dem Aufruf dieser Methode müssen die Frequenzbänder zu Beginn konfiguriert werden. Das entsprechende SPS Array muss wie folgt definiert sein. Für eine erneute Konfiguration mit einem anderen Satz an Argumenten kann die `Configure()` Methode ebenfalls genutzt werden.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : POINTER TO LREAL; // pointer to array (LREAL) of arguments
    nArgSize : UDINT;            // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`. Die zwei zu konfigurierenden Parameter je Frequenzband und Kanal sind `[fLowerFrequencyLimit, fUpperFrequencyLimit]`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle	`LREAL, 1, 2`
Frequenzbandspezifische Konfiguration (`nMaxBands >= 1`)	`LREAL, 2, nMaxBands x 2`
Kanalspezifische Konfiguration (`nMaxBands = 1`)	`LREAL, 2, nChannels x 2`
Kanal- und frequenzbandspezifische Konfiguration	`LREAL, 3, nChannels x nMaxBands x 2`

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars      : ST_CM_IntegratedRMS_InitPars;     // init parameter
    nOwnID          : UDINT;                            // ID for this FB instance
    aDestIDs        : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers  : UDINT := 4;                       // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_IntegratedRMS_InitPars [▶ 304]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_MultiBandRMS [▶ 189] berechnet RMS Werte für ein- und mehrkanalige reell-wertige Zeitreihen für konfigurierbarer Frequenzbänder.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.21 FB_CMA_MagnitudeSpectrum

Berechnet das Magnitudenspektrum (auch Betragsspektrum oder Amplitudenspektrum genannt) eines reell-wertigen Eingangssignals.

Der Baustein FB_CMA_MagnitudeSpectrum berechnet aus einem reell-wertigem Eingangssignal das Betragsspektrum. Der Baustein übernimmt dabei mehrere Funktionen, siehe Analyse von Daten-Streams [▶ 18] und Frequenzanalyse [▶ 37]:

Der Eingabedatenpuffer wird zunächst mit den unmittelbar vorhergehenden Puffern überlappend kombiniert und mit einer Fensterfunktion multipliziert. Wenn der Wert des Parameters `nFFT_Length` größer ist als der des Parameter `nWindowLength`, wird das gefensterte Zeitsignal am Anfang und am Ende mit der gleichen Anzahl Nullen aufgefüllt, um die angeforderte FFT Eingangslänge zu erreichen (Zero Padding).
Anschließend wird eine FFT für reelle Werte angewandt und von den resultierenden komplexen Werten wird der Absolutbetrag berechnet. Wenn der Parameter `bTransformToDecibel` den Wert `TRUE` hat werden die Werte zu Dezibel -Werten transformiert. Diese Dezibel-Werte sind für Magnituden- und Leistungsspektrum gleich, d.h. der Einfluss der Quadrierung wird bei der Berechnung des Dezibel-Wertes durch einen Faktor zwei für das Magnitudenspektrum berücksichtigt. Des Weiteren ist eine Skalierung des Betragsspektrums über den Parameter `eScalingType` durchführbar, siehe dazu Skalierung von Spektren [▶ 26].

Der Baustein FB_CMA_MagnitudeSpectrum verhält sich gleichwertig zum FB_CMA_PowerSpectrum [▶ 202]. Der Unterschied ist die Quadrierung des Ergebnisses beim FB_CMA_PowerSpectrum [▶ 202].

Das Kurzzeit-Spektrum ist in vielen Fällen kein guter statistischer Schätzer für das Spektrum eines Signals. In vielen Fällen ist es angebracht, die Fluktuation der Schätzwerte durch Mittelung über mehrere Frequenzen oder über aufeinanderfolgende Spektren zu verringern.

Skalierung

Die Skalierung der erhaltenen Ergebniswerte, also z. B. der Beschleunigungsdichten (Acceleration Spectral Densities) entspricht standardmäßig der Definition der FFT. Dies bedeutet, dass der Einfluss der Fensterlänge und der Fensterfunktion herausgerechnet werden.
Zur rechnerischen Skalierung absoluter Messungen können tabellierte Parameter verwendet werden, die im Abschnitt "Optionen der Spektrumsskalierung [▶ 390]“ beschrieben sind.

Gedächtniseigenschaften

Aufgrund der Verwendung der Welch-Methode wird jeweils der aktuelle Eingangsdatenpuffer zusammen mit den zuletzt übergebenen Puffern zur Berechnung genutzt. Die Anzahl der einfließenden Puffer ist abhängig von der gewählten Überlappung (`nOverlap`).
Die Frequenzanalyse berücksichtigt Sprünge in der Zeitreihe. Um ein korrektes Ergebnis zu erzielen, müssen sich deswegen die verwendeten Eingangsdatenpuffer lückenlos und ohne Sprünge aneinanderreihen.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Magnitudenspektrum [▶ 329].

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1, nWindowLength-nOverlap`	`LREAL, 1, nFFT_Length/2+1`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2, nChannels x nWindowLength-nOverlap`	`LREAL, 2, nChannels x nFFT_Length/2+1`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_MagnitudeSpectrum_InitPars;  // init parameter
    nOwnID           : UDINT;                             // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;   // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                        // number of MultiArrays which should be i
nitialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;              // timeout checking off during access to i
nter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MagnitudeSpectrum_InitPars [▶ 306]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.

- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_MagnitudeSpectrum_InitPars; // init parameter
    nOwnID         : UDINT;                            // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;                       // number of MultiArrays which should be init
ialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MagnitudeSpectrum_InitPars [▶ 306]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CM_PowerSpectrum [▶ 202] berechnet das Leistungsspektrum mit einer Quadrierung der Werte im letzten Schritt.

Der Baustein FB_CMA_PowerCepstrum [▶ 198] berechnet eine Transformation, die harmonische Oberschwingungen hervorhebt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.22 FB_CMA_MeanStressCorrection

Berechnet Mittelspannungsfreie Umkehrpunkte aus klassierten Umkehrpunkten.

Der Baustein FB_CMA_MeanStressCorrection implementiert die Mittelspannungskorrekturen nach Goodman und Gerber zur Umrechnung von Range-Mean Zählungen in Mittelwertfreie Range-Mean Zählungen. Die Zählungen werden in Form von mehrdimensionalen MultiArrays verarbeitet.

Der Ermüdungsschaden eines Bauteils kann anhand der durch den Baustein FB_CMA_RainflowCounting [▶ 211] gezählten Halbschleifen mit dem Baustein FB_CMA_MinersRule [▶ 179] berechnet werden. Bei der Schädigungsüberwachung kann mit Hilfe des Bausteins FB_CMA_MeanStressCorrection [▶ 176] eine Mittelspannungskorrektur des Zählergebnisses durchgeführt werden.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Schädigungsüberwachung [▶ 370]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`UDINT, 2, (nBins+2) x (nBinsMean+2)`	`UDINT, 2, (nBins+2) x (nBinsMean+2)`
Mehrkanalige Variante (`nChannels > 1`)	`UDINT, 3, nChannels x (nBins+2) x (nBinsMean+2)`	`UDINT, 3, nChannels x (nBins+2) x (nBinsMean+2)`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_MeanStressCorrection_InitPars;  // init parameter
    nOwnID           : UDINT;                                // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;      // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                           // number of MultiArrays which should b
e initialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;                 // timeout checking off during access t
o inter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MeanStressCorrection_InitPars [▶ 307]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
```

```
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Configure():

Mit dem Aufruf dieser Methode kann die Methode zur Mittelspannungskorrektur konfiguriert werden. Die Auswahl erfolgt über Werte vom Typ E_CM_MeanStressCorrection [▶ 290]. Nähere Informationen sind im Abschnitt Lebensdaueranalyse und Schädigungsrechnung [▶ 53] zu finden.

- **Rückgabewert:** Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg      : POINTER TO UDINT;  // pointer to 1-dimensional array (UDINT) of arguments
    nArgSize  : UDINT;             // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle und Unterkanäle	`UDINT, 1, 1`
Kanalspezifische Konfiguration	`UDINT, 2, nChannels x 1`

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars      : ST_CM_MeanStressCorrection_InitPars;  // init parameter
    nOwnID          : UDINT;                                // ID for this FB instance
    aDestIDs        : ARRAY[1..cCMA_MaxDest] OF UDINT;      // IDs of destinations for output
    nResultBuffers : UDINT := 4;                            // number of MultiArrays which should be
initialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MeanStressCorrection_InitPars [▶ 307]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.

- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_RainflowCounting [▶ 211] verwendet berechnet auf Basis der Dreipunkt-Regel eine Zählmatrix aus Belastungs- und Mittelwerten.

Der Algorithmus des Bausteins FB_CMA_MinersRule [▶ 179] kann verwendet werden um den Ermüdungsschaden von Bauteilen auf Basis der gezählten Halbzyklen zu berechnen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.23 FB_CMA_MinersRule

Berechnet den Ermüdungsschaden eines Bauteils.

Der Baustein FB_CMA_MinersRule berechnet den Ermüdungsschaden von Bauteilen auf Basis von Belastungs- und Mittelwerten sowie die Anzahl der gezählten Halbschleifen.

Das Ergebnis ist ein eindimensionales Array `aResult : ARRAY[0..1] OF LREAL`, welches die berechnete Gesamtschädigung (siehe Größe *D* in Hintergundinformationen [▶ 53] zur Ermüdungsrechnung) und die zugrunde liegende Anzahl von Halbschleifen enthält, d.h. `[{fDamage},{fCycles}]`.

Der Ermüdungsschaden eines Bauteils kann anhand der durch den Baustein FB_CMA_RainflowCounting [▶ 211] gezählten Halbschleifen mit dem Baustein FB_CMA_MinersRule [▶ 179] berechnet werden. Bei der Schädigungsüberwachung kann mit Hilfe des Bausteins FB_CMA_MeanStressCorrection [▶ 176] eine Mittelspannungskorrektur des Zählergebnisses durchgeführt werden.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Schädigungsüberwachung [▶ 370]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`UDINT, 2, (nBins+2) x (nBinsMean+2)`	`LREAL, 1, 2`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2, nChannels x (nBins+2) x (nBinsMean+2)`	`LREAL, 2, nChannels x 2`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_MinersRule_InitPars;        // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MinersRule_InitPars [▶ 307]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Configure():

Mit dem Aufruf dieser Methode können die Anzahlen der Halbschleifen bis zum Identifizieren eines Fehlers je definiertem Bin konfiguriert werden.

- **Rückgabewert:** Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg      : POINTER TO LREAL;  // pointer to 1-dimensional array (LREAL) of arguments
    nArgSize  : UDINT;             // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle und Unterkanäle	`LREAL, 1, nBins+2`

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Kanalspezifische Konfiguration	`LREAL, 2, nChannels x (nBins+2)`

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_RainflowCounting_InitPars;  // init parameter
    nOwnID         : UDINT;                            // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;                       // number of MultiArrays which should be init
ialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MinersRule_InitPars [▶ 307]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_RainflowCounting [▶ 211] verwendet berechnet auf Basis der Dreipunkt-Regel eine Zählmatrix aus Belastungs- und Mittelwerten.

Der Baustein FB_CMA_MeanStressCorrection [▶ 176] ermöglicht die Durchführung einer Mittelspannungskorrektur bezüglich der Zugfestigkeit des überwachten Bauteils.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.24 FB_CMA_MomentCoefficients

Berechnet den Mittelwert, die empirische Standardabweichung, die Schiefe und den Exzess für ein- und mehrkanalige reell-wertige Zeitreihen.

Der Baustein behandelt das Eingangssignal als Zeitreihe mit mehreren unabhängigen Kanälen. Für jeden Kanal werden die Momentenkoeffizienten optional bis zur Ordnung vier berechnet. Die maximale Ordnung der zu berechnenden Momente ist konfigurierbar mittels des Parameters `nOrder`. Eine spezifische Enumeration zur Verwendung der Momentenkoeffizienten ist ebenfalls verfügbar: E_CM_MCoefOrder [▶ 290]. Das Ergebnis wird als Array dieser Koeffizienten mit entsprechenden Indizes weitergegeben.

Standardmäßig wird für die Berechnung von empirischer Standardabweichung, Schiefe und Exzess keine Bessel'sche Korrektur durchgeführt. In den Initialisierungsparametern kann optional die Korrektur eingeschaltet werden, siehe `bPopulationEstimates`. Um erwartungstreue Ergebnisse zu erhalten, ist der Parameter auf `TRUE` zu setzen. Der Einfluss der Bessel'schen Korrektur wird mit Vergrößerung der Stichprobenmenge kleiner. Die relative Abweichung von korrigierter und nicht korrigierter empirischer Standardabweichung lässt sich eindeutig bestimmen. Anhaltspunkte liefert folgende Tabelle:

Stichprobenumfang *N*	**Relative Abweichung / %**
10	-5,13
100	-0,501
1000	-0,05001
10000	-0,0050001

Vom Baustein ausgegeben werden: Der Stichprobenumfang *N* (für alle `nOrder`), der arithmetische Mittelwert (`nOrder = 1`), die empirische Standardabweichung (`nOrder = 2`), die Schiefe (`nOrder = 3`), der Exzess (`nOrder = 4`).

Definition der empirisch berechneten Momente

Der arithmetische Mittelwert

$$\overline{x} = \frac{1}{N}\sum_{n=0}^{N-1} x[n]$$

Die empirische Standardabweichung, ohne Bessel'sche Korrektur

$$s' = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}(x[n]-\overline{x})^2}$$

Die empirische Standardabweichung, mit Bessel'scher Korrektur

$$s = \sqrt{\frac{1}{N-1}\sum_{n=0}^{N-1}(x[n]-\overline{x})^2}$$

Die empirische Schiefe (ohne Bessel'sche Korrektur *v'* sowie mit Korrektur *v*)

$$v' = \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{x[n]-\overline{x}}{s'}\right)^3 \qquad v = \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{x[n]-\overline{x}}{s}\right)^3$$

Der empirische Exzess (ohne Bessel'sche Korrektur *E'* sowie mit Korrektur *E*)

$$K' = \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{x[n]-\overline{x}}{s'}\right)^4 \qquad K = \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{x[n]-\overline{x}}{s}\right)^4$$

$$E' = K' - 3 \qquad E = K - 3$$

Der Exzess *E* ist demnach die Differenz von Wölbung *K* und dem Wert 3, dies entspricht der Wölbung der Normalverteilung. Sie beschreibt also die Bewertung der berechneten Wölbung hinsichtlich einer Normalverteilung.

Es kann sowohl in jedem Zyklus ein einzelnes Sample pro Kanal (siehe Ein- und Ausgänge, erste Tabelle) als auch in einem Zyklus mehrere Samples pro Kanal zur Stichprobenmenge hinzugefügt werden (siehe Ein- und Ausgänge, zweite Tabelle).

Weitere Anmerkungen

Die Berechnung von Standardabweichung und höheren Momenten erfordert, dass eine Mindestzahl von Stichprobenwerten vorliegt. Ist die Bessel'sche Korrektur inaktiv wird der Mittelwert sowie die Standardabweichung bereits für einen Stichprobenumfang von 1 berechnet. Für die Berechnung der Schiefe und den Exzess werden 2 Werte benötigt. Ist die Bessel'sche Korrektur hingegen aktiv entspricht der minimal benötigte Stichprobenumfang der Ordnung (Mittelwert – 1, Standardabweichung -2, Schiefe – 3, Exzess - 4). Zusätzlich darf für die Berechnung von Schiefe und Exzess die Varianz nicht Null sein.

Ergebnisse für höhere Momente können unter Umständen ungenau werden, wenn die Eingangswerte viele Größenordnungen auseinanderliegen.

Gedächtniseigenschaften

Die Stichprobenmenge *N*, welche zur Berechnung des aktuellen Ergebnisses genutzt wird, erhöht sich automatisch bei jedem neuen eingehenden Datensatz, d.h. der Baustein verwendet alle Eingangswerte seit dessen Instanziierung. Das Zurücksetzen der Stichprobenmenge auf Null (löschen des internen Speichers des FBs) ist über eine `ResetData()` Methode oder, bei Verwendung der `CallEx()` Methode über die Variable `bResetData` vorgesehen.

NaN Vorkommnis

Falls für einen Kanal noch keine ausreichende Anzahl Eingangswerte für die Berechnung eines Ergebnisses vorliegt oder die Varianz Null beträgt, wird für diesen Kanal der Wert NaN (Not A Number) nach IEEE 754 zurückgegeben. Das Vorliegen dieses Fehlerwertes kann mit der Funktion LrealIsNaN() überprüft werden. Ursache dafür kann entweder sein, dass noch nicht genug Eingangsdaten übergeben wurden oder dass für einzelne Kanäle bisher lediglich NaNs als Eingangswerte übergeben wurden. Eine Varianz von Null kann insbesondere auftreten, wenn die Zeitreihe der Werte konstant ist, etwa wenn aufgrund von Kabelbruch oder Schaltfehlern keine Sensordaten übertragen werden.

Enthält ein Satz von Eingangswerten die spezielle Konstante NaN, so wird der Statistik für diesen Kanal bei diesem Zeitschritt kein Wert hinzugefügt, d.h. sie wird als Indikator für fehlende Werte behandelt.

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen folgender Definition (Shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nSubChannels = 0`)	`LREAL, 1,` `nChannels`	`LREAL, 2,` `nChannels x nOrder+1`
Standardvariante für mehrere Datensätze (`nSubChannels = 0`)	`LREAL, 2,` `nChannels x` *not specified**	`LREAL, 2,` `nChannels x nOrder+1`

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Mehrkanalige Variante (`nSubChannels > 0`)	`LREAL, 2, nChannels x nSubChannels`	`LREAL, 3, nChannels x nSubChannels x nOrder+1`
Mehrkanalige Variante für mehrere Datensätze (`nSubChannels > 0`)	`LREAL, 3, nChannels x nSubChannels x` *not specified**	`LREAL, 3, nChannels x nSubChannels x nOrder+1`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorrausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_MomentCoefficients_InitPars;  // init parameter
    nOwnID           : UDINT;                              // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                         // number of MultiArrays which should be
initialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;               // timeout checking off during access to
inter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MomentCoefficients_InitPars [▶ 308]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult    : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError        : BOOL;        // TRUE if an error occurs.
    hrErrorCode   : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

CallEx():

Die Methode wird jeden Zyklus aufgerufen um den internen Speicher aus dem Eingangssignal zu aktualisieren. Ein Ergebnis wird nur alle `nAppendData` Zyklen ausgegeben. Eine alternative Methode ist `Call()`.

Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD CallEx : HRESULT
VAR_INPUT
    nAppendData   : UDINT;       // count of data buffers which are appended until calculation (1= cal
culate always)
    bResetData    : BOOL;        // automatic reset of dataset buffer after each calculation
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;         // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError        : BOOL;         // TRUE if an error occurs.
    hrErrorCode   : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `nAppendData`: Legt fest wie viele Eingangsdatenpuffer gesammelt werden sollen bevor eine Berechnung ausgeführt wird, denn vorzugsweise werden mehrere Datenblöcke angefügt um ein präzises Ergebnis zu erreichen.
- `bResetData`: Falls gesetzt, wird der interne Datenpuffer nach Berechnung vollständig gelöscht.
- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars    : ST_CM_MomentCoefficients_InitPars;  // init parameter
    nOwnID        : UDINT;                              // ID for this FB instance
    aDestIDs      : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers: UDINT := 4;                         // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MomentCoefficients_InitPars [▶ 308]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

Alternativ kann der automatische Reset in der Methode `CallEx()` verwendet werden.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_EmpiricalMean [▶ 131] berechnet den empirischen Mittelwert von Eingangswerten.

Der Baustein FB_CMA_EmpiricalStandardDeviation [▶ 141] berechnet die empirische Standardabweichung von Eingangswerten.

Der Baustein FB_CMA_EmpiricalSkew [▶ 136] berechnet die empirische Schiefe von Eingangswerten.

Der Baustein FB_CMA_EmpiricalExcess [▶ 126] berechnet den empirischen Exzess von Eingangswerten.

Der Baustein FB_CMA_HistArray [▶ 154] berechnet die Histogramme von Verteilungen von Eingangswerten.

Der Baustein FB_CMA_Quantiles [▶ 206] berechnet die Quantile einer empirischen Verteilung, die ebenfalls erlauben die Häufigkeit von Ausreißern zu beurteilen.

Der Baustein FB_CMA_CrestFactor [▶ 99] berechnet ein zur Kurtosis alternatives Maß für die Spitzenhaltigkeit (Crest Faktor) eines Signals, das allerdings empfindlicher gegenüber Ausreißern ist.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.25 FB_CMA_MultiBandRMS

Berechnet RMS Werte für ein- und mehrkanalige reell-wertige Zeitreihen für konfigurierbarer Frequenzbänder

Der Baustein FB_CMA_MultiBandRMS berechnet für ein- und mehrkanalige Zeitreihen bezogen auf individuell konfigurierbare Frequenzbänder die RMS Werte der Signale.

Die Anzahl der Kanäle wird über den Input-Stream beschrieben. Die maximale Anzahl der für einen Kanal konfigurierten Frequenzbänder sowie die Parameter der internen Fourier-Transformation werden über die ST_CM_MultiBandRMS_InitPars [▶ 309] übergeben. Die Konfiguration der Frequenzbänder erfolgt durch den Aufruf der Configure() Methode.

Der Baustein kann beispielsweise bei der Überwachung von Lagerschadfrequenzen eingesetzt werden.

Abgrenzung zum FB_CMA_IntegratedRMS [▶ 167]:

Der Baustein FB_CMA_IntegratedRMS hat die zusätzliche Funktionalität, dass die Eingangszeitreihen vor der Frequenzband-limitierten RMS Berechnung, optional mit bis zur Ordnung 2 zeitlich integriert werden können. Damit kann dieser Baustein z. B. direkt den RMS Wert für Schwingbeschleunigung, Schwinggeschwindigkeit und Schwingweg für ein definiertes Frequenzband berechnen.

Konfiguration

Als Konfigurationsparameter wird der Configure() Methode des Bausteins ein dreidimensionales Array mit Werten übergeben (oder optional zweidimensional im Falle eines einzigen Eingangskanals, ansonsten werden alle Kanäle identisch konfiguriert). In diesem wird die untere sowie obere Grenze eines Frequenzbandes angegeben. Der Funktionsbaustein berechnet dann basierend auf den Eingangsdaten die RMS Werte für diese Frequenzbänder eines jeden Kanals.

Gedächtniseigenschaften

Aufgrund der Verwendung der Welch-Methode wird jeweils der aktuelle Eingangsdatenpuffer zusammen mit den zuletzt übergebenen Puffern zur Berechnung genutzt. Die Anzahl der einfließenden Puffer ist abhängig von der gewählten Überlappung (`nOverlap`).
Die Frequenzanalyse berücksichtigt Sprünge in der Zeitreihe. Um ein korrektes Ergebnis zu erzielen, müssen sich deswegen die verwendeten Eingangsdatenpuffer lückenlos und ohne Sprünge aneinanderreihen.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

> **Behandlung von NaN-Werten**
> Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/3394505867.zip.

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`. Für die Verarbeitung einer beliebigen Anzahl von Kanälen (`nChannels >= 1`) gilt:

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nMaxBands >= 1`)	`LREAL, 2, nChannels x nWindowLength-nOverlap`	`LREAL, 2, nChannels x nMaxBands`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_MultiBandRMS_InitPars;      // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MultiBandRMS_InitPars [▶ 309]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                            // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                         // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;   // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                           // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Configure():

Mit dem Aufruf dieser Methode müssen die Frequenzbänder zu Beginn konfiguriert werden. Das entsprechende SPS Array muss wie folgt definiert sein. Für eine erneute Konfiguration mit einem anderen Satz an Argumenten kann die `Configure()` Methode ebenfalls genutzt werden.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : POINTER TO LREAL; // pointer to array (LREAL) of arguments
    nArgSize : UDINT;            // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`. Die zwei zu konfigurierenden Parameter je Frequenzband und Kanal sind `[fLowerFrequencyLimit, fUpperFrequencyLimit]`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle	`LREAL, 1, 2`
Frequenzbandspezifische Konfiguration (`nMaxBands >= 1`)	`LREAL, 2, nMaxBands x 2`
Kanalspezifische Konfiguration (`nMaxBands = 1`)	`LREAL, 2, nChannels x 2`
Kanal- und frequenzbandspezifische Konfiguration	`LREAL, 3, nChannels x nMaxBands x 2`

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_MultiBandRMS_InitPars;      // init parameter
    nOwnID         : UDINT;        // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;   // number of MultiArrays which should be initialized for results
(0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MultiBandRMS_InitPars [▶ 309]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.

- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_IntegratedRMS [▶ 167] hat die zusätzliche Funktionalität, dass die Eingangszeitreihen vor der Frequenzband-limitierten RMS Berechnung, optional mit bis zur Ordnung 2 zeitlich integriert werden können.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.26 FB_CMA_OrderPowerSpectrum

Berechnung des Ordnungsspektrums reell-wertiger Eingangssignale.

Die Ordnungsanalyse dient der Analyse von Schwingungen an rotierenden Maschinen. Besonderheit ist, dass die Ordnungsanalyse auch verlässlich interpretierbare Ergebnisse liefert, wenn die Drehgeschwindigkeit der rotierenden Komponente während der Analyse nicht gleichförmig ist, vgl. z.B. Windkraftanlagen oder rampende Motoren.

Bei der Frequenzanalyse, z.B. mit dem `FB_CMA_PowerSpectrum`, wird ein Zeitsignal (z.B. ein Vibrationssignal) in den Frequenzbereich transformiert. Die Darstellung der Spektralwerte erfolgt dann über die Frequenz. Hingegen wird bei der Ordnungsanalyse das Zeitsignal vor der Transformation mit Hilfe eines Encodersignals so transformiert und interpoliert, dass das Vibrationssignal räumlich äquidistante Abtastpunkte entlang eines Wellenumlaufs aufweist. Folgende Überlegung hilft beim Verständnis der Grundidee. Das Vibrationssignal und das Positionssignal werden zeitlich äquidistant gesampled. Ist die Wellengeschwindigkeit konstant, so kann das Vibrationssignal einfach auf äquidistante Positionen entlang eines Wellenumlaufs projizieren werden. Verändert sich die Geschwindigkeit während einer Umdrehung, so liegen die Abtastungen entlang eines Umlaufs nicht mehr äquidistant auseinander; wird z.B. die Welle schneller, vergrößert sich der Abstand zwischen zwei Abtastungen, da sich die Welle zwischen zwei zeitlichen Abtastungen weitergedreht hat als bei kleinerer Wellengeschwindigkeit. Durch Einbeziehung eines Encodersignals kann dieser Effekt durch entsprechende Interpolation kompensiert werden. Anschließend wird das Leistungsspektrum, vgl. `FB_CMA_PowerSpectrum`, berechnet. Das Ordnungsspektrum wird dann nicht über die Frequenz aufgetragen, sondern über die Ordnung.

Die Frequenzachse wird mit der maximalen Drehzahl `fMaxRPM` (in Umdrehungen pro Minute) der beobachteten Welle skaliert. Daraus ergibt sich für die Ordnungs-Achse.

Für die Frequenzachse gilt:

```
fSampleRate     := cOversamples * (1000.0 / fTaskCycleTime);
fResolutionFreq := fSampleRate / cFFTLength;
fNyquistFreq    := fSampleRate / 2;
```

Für die Ordnungsachse gilt:

```
fMaxOrder        := fNyquistFreq / (fMaxRPM / 60);
fResolutionOrder := fResolutionFreq / (fMaxRPM / 60);
```

Die Parametrierung erfolgt über die Struktur ST_CM_OrderPowerSpectrum_InitPars [▶ 311].

Der Baustein `FB_CMA_OrderPowerSpectrum` erwartet pro Kanal ein Vibrations- und ein Positionssignal. Diese werden durch den Baustein FB_CMA_SourcePaired [▶ 252] als Tupel in die Analysekette eingebracht.

Gedächtniseigenschaften

Aufgrund der Verwendung der Welch-Methode wird jeweils der aktuelle Eingangsdatenpuffer zusammen mit den zuletzt übergebenen Puffern zur Berechnung genutzt. Die Anzahl der einfließenden Puffer ist abhängig von der gewählten Überlappung (`nOverlap`).
Die Frequenzanalyse berücksichtigt Sprünge in der Zeitreihe. Um ein korrektes Ergebnis zu erzielen, müssen sich deswegen die verwendeten Eingangsdatenpuffer lückenlos und ohne Sprünge aneinanderreihen.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Ordnungsanalyse [▶ 368]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Bei der Verarbeitung eines einzelnen Kanals (`nChannels = 1`) gilt:

MultiArray im	Elementtyp	Dimensionen	Dimensionsgrößen
input stream A (Vibration)	LREAL	1	`nWindowLength - nOverlap`
input stream B (Position)	LREAL	1	`(nWindowLength - nOverlap) / (fSampleRateSignal / fSampleRatePosition)`
output stream	LREAL	1	`nFFT_Length/2+1`

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) gilt:

MultiArray im	Elementtyp	Dimensionen	Dimensionsgrößen
input stream A (Vibration)	LREAL	2	`nChannels x (nWindowLength - nOverlap)`
input stream B (Postion)	LREAL	1	`(nWindowLength - nOverlap) / (fSampleRateSignal / fSampleRatePosition)`
output stream	LREAL	2	`nChannels x nFFT_Length/2+1`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_OrderPowerSpectrum_InitPars;  // init parameter
    nOwnID           : UDINT;                              // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                         // number of MultiArrays which should be
initialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;               // timeout checking off during access to
inter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_OrderPowerSpectrum_InitPars [▶ 311]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.

- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars      : ST_CM_OrderPowerSpectrum_InitPars;  // init parameter
    nOwnID          UDINT;                                // ID for this FB instance
    aDestIDs        : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers : UDINT := 4;                          // number of MultiArrays which should be in
itialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_OrderPowerSpectrum_InitPars [▶ 311]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Der Baustein FB_CMA_PowerSpectrum [▶ 202] berechnet das Leistungsspektrum für reell-wertige Eingangsdaten.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.27 FB_CMA_PowerCepstrum

Der Baustein berechnet das Leistungscepstrum oder Powercepstrum eines reell-wertigen Eingangssignals.

Das Leistungscepstrum $C_p(\tau)$ ist wie folgt, hier in zeitkontinuierlicher Darstellung, definiert:

$$C_{\mathrm{p}}(\tau) = X^{-1}\left\{\log(|X(\omega)|^2)\right\} = X^{-1}\left\{2\log|X(\omega)|\right\}, \qquad \text{Quefrency } \tau$$

Es ist entsprechend als inverse Fouriertransformation des logarithmierten Leistungsspektrums (siehe FB_CMA_PowerSpectrum [▶ 202]) definiert. Durch Hin- und Rücktransformation ist das Ergebnis wieder im Zeitbereich.

Der Baustein ist für das Monitoring von Getrieben hilfreich, siehe Getriebeüberwachung [▶ 48].

In der numerischen Implementierung wird zunächst das PowerSpectrum berechnet. Der Eingangsdatenpuffer wird dazu mit den unmittelbar vorhergehenden Puffern überlappend kombiniert und mit einer Fensterfunktion multipliziert. Wenn der Wert des Parameters `nFFT_Length` größer ist als der des Parameter `nWindowLength`, wird das gefensterte Zeitsignal am Anfang und am Ende mit der gleichen Anzahl Nullen aufgefüllt, um die angeforderte FFT Eingangslänge zu erreichen (Zero Padding).
Anschließend wird eine FFT für reelle Werte angewandt und von den resultierenden komplexen Werten wird der Absolutbetrag sowie das Quadrat der Werte berechnet. Die Werte werden dann logarithmiert. Vor der Logarithmierung wird das Argument mit dem Wert des Parameters `fLogThreshold` verglichen. Wenn sie kleiner sind werden sie auf diesen Wert gesetzt, um Wertbereichsfehler oder den Versuch, den Logarithmus von Null zu berechnen, zu vermeiden. Anschließend wird erneut eine inverse Fouriertransformation vorgenommen. Das Ergebnis ist dann ein Array mit komplexen Werten.

Auswertung des komplex-wertigen Ergebnisses

In der Praxis wird, je nach Applikation, der Absolutbetrag, der quadrierte Absolutbetrag oder auch nur der Realteil des komplex-wertigen Leistungscepstrums ausgewertet. Dies ist vom Anwender entsprechend zu implementieren.

Zum Leistungscepstrum existieren jeweils leicht unterschiedliche Definitionen. Die hier verwendete Definition orientiert sich an gängigen Definitionen von J. Korelus sowie Robert B. Randall, siehe Literaturhinweise [▶ 64].

Abgrenzung zum komplexen Cepstrum:

Das Leistungscepstrum ist zu unterscheiden vom komplexen Cepstrum, welches als logarithmierte Fourier-Rücktransformation des komplexen Spektrums eines Signals definiert ist. Aufgrund der Betragsbildung ist das Leistungscepstrum im Vergleich zum komplexen Cepstrum weniger empfindlich auf die Eigenschaften der Phasenlage des Signals. Des Weiteren liefert das komplexe Cepstrum direkt ein reell-wertiges Ergebnis.

Gedächtniseigenschaften

Aufgrund der Verwendung der Welch-Methode wird jeweils der aktuelle Eingangsdatenpuffer zusammen mit den zuletzt übergebenen Puffern zur Berechnung genutzt. Die Anzahl der einfließenden Puffer ist abhängig von der gewählten Überlappung (`nOverlap`).
Die Frequenzanalyse berücksichtigt Sprünge in der Zeitreihe. Um ein korrektes Ergebnis zu erzielen, müssen sich deswegen die verwendeten Eingangsdatenpuffer lückenlos und ohne Sprünge aneinanderreihen.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Leistungscepstrum [▶ 365].

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1, nWindowLength-nOverlap`	`LCOMPLEX, 1, nFFT_Length/2+1`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2, nChannels x nWindowLength-nOverlap`	`LCOMPLEX, 2, nChannels x nFFT_Length/2+1`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_PowerCepstrum_InitPars;     // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_PowerCepstrum_InitPars [▶ 312]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.

- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_PowerCepstrum_InitPars;     // init parameter
    nOwnID         : UDINT;          // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;     // number of MultiArrays which should be initialized for results
 (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_PowerCepstrum_InitPars [▶ 312]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_Envelope [▶ 146] eignet sich ebenfalls zur Analyse impulshafter Anregungen mit linearen und nichtlinearen Systembestandteilen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.28 FB_CMA_PowerSpectrum

Berechnung des Leistungsspektrums eines reell-wertigen Eingangssignals sowie optionale Skalierung zu Dezibel.

Der Baustein berechnet das Leistungsspektrum (auch Powerspektrum, Korrelogramm und Periodogramm genannt) eines reell-wertigen Eingangssignals. Der Baustein übernimmt dabei mehrere Funktionen, siehe Analyse von Daten-Streams [▶ 18] und Frequenzanalyse [▶ 37]:

Der Eingabedatenpuffer wird zunächst mit den unmittelbar vorhergehenden Puffern überlappend kombiniert und mit einer Fensterfunktion multipliziert. Wenn der Wert des Parameters `nFFT_Length` größer ist als der des Parameter `nWindowLength`, wird das gefensterte Zeitsignal am Anfang und am Ende mit der gleichen Anzahl Nullen aufgefüllt, um die angeforderte FFT Eingangslänge zu erreichen (Zero Padding).
Anschließend wird eine FFT für reelle Werte angewandt und von den resultierenden komplexen Werten wird der Absolutbetrag berechnet. Wenn der Parameter `bTransformToDecibel` den Wert `TRUE` hat werden die Werte zu Dezibel -Werten transformiert. Diese Dezibel-Werte sind für Magnituden- und Leistungsspektrum gleich, d.h. der Einfluss der Quadrierung wird bei der Berechnung des Dezibel-Wertes durch einen Faktor zwei für das Magnitudenspektrum berücksichtigt. Des Weiteren ist eine Skalierung des Betragsspektrums über den Parameter `eScalingType` durchführbar, siehe dazu Skalierung von Spektren [▶ 26].

Der Baustein FB_CMA_PowerSpectrum verhält sich gleichwertig zum FB_CMA_MagnitudeSpectrum [▶ 172]. Der Unterschied ist die Quadrierung des Ergebnisses beim FB_CMA_PowerSpectrum [▶ 202].

Das Kurzzeit-Leistungsspektrum ist in vielen Fällen kein guter statistischer Schätzer für das Spektrum eines Signals. In vielen Fällen ist die Fluktuation der Schätzwerte durch Mittelung über mehrere Frequenzen oder über aufeinanderfolgende Spektren zu verringern.

Skalierung

Die Skalierung der erhaltenen Ergebniswerte, also z. B. der Beschleunigungsdichten (Acceleration Spectral Densities) entspricht standardmäßig der Definition der FFT. Dies bedeutet, dass der Einfluss der Fensterlänge und der Fensterfunktion herausgerechnet werden.
Zur rechnerischen Skalierung absoluter Messungen können tabellierte Parameter verwendet werden, die im Abschnitt "Optionen der Spektrumsskalierung [▶ 390]“ beschrieben sind.

Gedächtniseigenschaften

Aufgrund der Verwendung der Welch-Methode wird jeweils der aktuelle Eingangsdatenpuffer zusammen mit den zuletzt übergebenen Puffern zur Berechnung genutzt. Die Anzahl der einfließenden Puffer ist abhängig von der gewählten Überlappung (`nOverlap`).
Die Frequenzanalyse berücksichtigt Sprünge in der Zeitreihe. Um ein korrektes Ergebnis zu erzielen, müssen sich deswegen die verwendeten Eingangsdatenpuffer lückenlos und ohne Sprünge aneinanderreihen.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1, nWindowLength-nOverlap`	`LREAL, 1, nFFT_Length/2+1`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2, nChannels x nWindowLength-nOverlap`	`LREAL, 2, nChannels x nFFT_Length/2+1`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars        : ST_CM_PowerSpectrum_InitPars;     // init parameter
    nOwnID            : UDINT;                            // ID for this FB instance
    aDestIDs          : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers    : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;              // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_PowerSpectrum_InitPars [▶ 313]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_PowerSpectrum_InitPars;    // init parameter
    nOwnID         : UDINT;                               // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_PowerSpectrum_InitPars [▶ 313]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_MagnitudeSpectrum [▶ 172] berechnet das Magnitudenspektrum ohne Quadrierung der Werte im letzten Schritt.

Der Baustein FB_CMA_PowerCepstrum [▶ 198] berechnet eine Transformation, die harmonische Oberschwingungen hervorhebt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.29 FB_CMA_Quantiles

Berechnet für ein- und mehrkanalige reell-wertige Signale die Quantile der Verteilung der Eingangswerte.

Der Baustein FB_CMA_Quantiles berechnet p-Quantile von ein- und mehrkanaligen reell-wertigen Eingangsdaten. Jeder Kanal wird dabei unabhängig voneinander betrachtet.

Der Baustein basiert zunächst auf der Berechnung einer Häufigkeitsverteilung, siehe FB_CMA_HistArray [▶ 154]. Zur Parametrierung werden der untere und obere Grenzwert sowie die Anzahl der Klassen (auch Bins genannt) der Häufigkeitsverteilung übergeben. Die einzelnen Klassengrenzen werden dann im dadurch definierten Gesamtbereich in gleich großen Intervallen verteilt, siehe Histogramm [▶ 29]. Nachfolgende wird die Summenhäufigkeitsverteilung und daraus die konfigurierten Quantile berechnet, siehe Statistische Auswertung [▶ 29]. Ein zusätzlicher Konfigurationsparameter ist also die Zahl der zu berechnenden Quantile für jeden Kanal.

Das Ergebnis ist ein zweidimensionales Array mit reellen Werten. Der erste Index ist die Nummer des Kanals und der zweite Index ist die Nummer des betreffenden Quantils.

Werte die unterhalb des unteren Grenzwertes liegen, sowie Werte die oberhalb des oberen Grenzwertes bezüglich der Klasseneinteilung liegen, werden für die Quantils-Berechnung nicht berücksichtigt. Innerhalb eines Intervalls werden die Quantilwerte interpoliert. Wenn die empirische Summenhäufigkeitsverteilung abschnittsweise konstant ist, wird der kleinste passende Wert verwendet.

Gedächtniseigenschaften

Die Stichprobenmenge *N*, welche zur Berechnung des aktuellen Ergebnisses genutzt wird, erhöht sich automatisch bei jedem neuen eingehenden Datensatz, d.h. der Baustein verwendet alle Eingangswerte seit dessen Instanziierung. Das Zurücksetzen der Stichprobenmenge auf Null (löschen des internen Speichers des FBs) ist über eine `ResetData()` Methode oder, bei Verwendung der `CallEx()` Methode über die Variable `bResetData` vorgesehen.

Konfiguration

Als Konfigurationsparameter wird der `Configure()` Methode des Bausteins ein zweidimensionales Array mit Werten übergeben. Jeder Wert stellt die relative Häufigkeit für einen Kanal und ein zu berechnendes Quantil dar. Der Baustein berechnet dann für die eingegangenen Daten jeweils kanalweise die Quantile zu diesen Häufigkeiten. Vorgegeben ist die Häufigkeit 0.5, die dem Median entspricht. Alternativ kann ein eindimensionales Array mit Werten für zu berechnende Quantile übergeben werden, welche für alle Kanäle angewendet werden.

NaN Vorkommnis

Falls für einen Kanal noch keine Ergebniswerte vorliegen, wird für diesen Kanal der Wert NaN (Not A Number) zurückgegeben. Ursache dafür kann entweder sein, dass noch keine Eingangsdaten übergeben wurden, dass alle bisher übergebenen Daten außerhalb des Intervalls zwischen `fMinBinned` und `fMaxBinned` liegen, oder dass für einzelne Kanäle lediglich NaNs als Eingangswerte übergeben wurden. Enthält ein Satz von Eingangswerten die spezielle Konstante NaN, so wird der Statistik für diesen Kanal bei diesem Zeitschritt kein Wert hinzugefügt, d.h. sie wird als Indikator für fehlende Werte behandelt.

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Routerspeicher

Die Quantilberechnung ist eine statistische Berechnung auf Basis von Histogrammen welche viel Speicher benötigen. Die belegte Speichermenge hängt von den Parametern `nChannels`, `nBins` und `nMaxQuantiles` ab. Es wird empfohlen diese Parameter so klein wie nötig zu halten.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Condition Monitoring mit Frequenzanalyse [▶ 353].

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen folgender Definition (Shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nSubChannels = 0`)	`LREAL, 1,` `nChannels`	`LREAL, 2,` `nChannels x nMaxQuantiles`
Standardvariante für mehrere Datensätze (`nSubChannels = 0`)	`LREAL, 2,` `nChannels x` *not specified**	`LREAL, 2,` `nChannels x nMaxQuantiles`
Mehrkanalige Variante (`nSubChannels > 0`)	`LREAL, 2,` `nChannels x nSubChannels`	`LREAL, 3,` `nChannels x nSubChannels x nMaxQuantiles`
Mehrkanalige Variante für mehrere Datensätze (`nSubChannels > 0`)	`LREAL, 3,` `nChannels x nSubChannels x` *not specified**	`LREAL, 3,` `nChannels x nSubChannels x nMaxQuantiles`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorrausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_Quantiles_InitPars;         // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_Quantiles_InitPars [▶ 314]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                           // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                        // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                          // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.

- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

CallEx() :

Die Methode wird jeden Zyklus aufgerufen um Quantile aus dem Eingangssignal zu berechnen. Eine alternative Methode ist `Call()`.
Die Auswertung der Quantile ist in der Regel erheblich rechenaufwendiger als die Registrierung neuer Eingangswerte. Deswegen kann eine Benutzung der Methode `CallEx()` die Laufzeit je nach den konfigurierten Parametern erheblich verkürzen, indem statistische Ergebnisse erst berechnet werden, wenn sie benötigt werden.

Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD CallEx : HRESULT
VAR_INPUT
    nAppendData  : UDINT;      // count of data buffers which are appended until calculation (1= cal
culate always)
    bResetData   : BOOL;       // automatic reset of dataset buffer after each calculation
END_VAR
VAR_OUTPUT
    bNewResult   : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent
to transfer tray.
    bError       : BOOL;        // TRUE if an error occurs.
    hrErrorCode  : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `nAppendData`: Legt fest wie viele Eingangsdatenpuffer gesammelt werden sollen bevor eine Berechnung ausgeführt wird, denn vorzugsweise werden mehrere Datenblöcke angefügt um ein präzises Ergebnis zu erreichen.
- `bResetData`: Falls gesetzt, wird der interne Datenpuffer nach Berechnung vollständig gelöscht.
- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Configure() :

Mit dem Aufruf dieser Methode müssen die Quantilargumente zu Beginn konfiguriert werden. Das entsprechende SPS Array muss wie folgt definiert sein. Für eine erneute Konfiguration mit einem anderen Satz an Argumenten kann die `Configure()` Methode ebenfalls genutzt werden.

- **Rückgabewert:** Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : POINTER TO LREAL;  // pointer to 2-dimensional array (LREAL) of arguments
    nArgSize : UDINT;             // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle und Unterkanäle	`LREAL, 1, nMaxQuantiles`
Kanalspezifische Konfiguration (`nSubChannels = 0`)	`LREAL, 2, nChannels x nMaxQuantiles`
Unterkanalspezifische Konfiguration (`nSubChannels > 0`)	`LREAL, 2, nSubChannels x nMaxQuantiles`
Kanal- und unterspezifische Konfiguration (`nSubChannels > 0`)	`LREAL, 3, nChannels x nSubChannels x nMaxQuantiles`

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_Quantiles_InitPars;  // init parameter
    nOwnID         : UDINT;                     // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_Quantiles_InitPars [▶ 314]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

Alternativ kann der automatische Reset in der Methode `CallEx()` verwendet werden.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_HistArray [▶ 154] berechnet die Histogramme von Verteilungen von Eingangswerten.

Der Baustein FB_CMA_MomentCoefficients [▶ 183] berechnet die statistischen Momentenkoeffizienten : Mittelwert, Standardabweichung, Schiefe (Skew) und Kurtosis .

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.30 FB_CMA_RainflowCounting

Berechnet klassierte Umkehrpunkte in einer Beanspruchungs-Zeit-Folge.

Der Baustein FB_CMA_RainflowCounting implementiert den drei-Punkt Algorithmus an Anlehnung an ASTM-E-1049-85, angepasst auf die Verarbeitung von Datenströmen. Die gezählten Halbzyklen werden in Form eines zweidimensionalen MultiArrays (*mean stress und stress range*) zurückgegeben.

Der Ermüdungsschaden eines Bauteils kann anhand der gezählten Halbschleife mit dem Baustein FB_CMA_MinersRule [▶ 179] berechnet werden. Bei der Schädigungsüberwachung kann mit Hilfe des Bausteins FB_CMA_MeanStressCorrection [▶ 176] eine Mittelspannungskorrektur des Zählergebnisses durchgeführt werden.

Gedächtniseigenschaften

Die Stichprobenmenge *N*, welche zur Berechnung des aktuellen Ergebnisses genutzt wird, erhöht sich automatisch bei jedem neuen eingehenden Datensatz, d.h. der Baustein verwendet alle Eingangswerte seit dessen Instanziierung. Das Zurücksetzen der Stichprobenmenge auf Null (löschen des internen Speichers des FBs) ist über eine `ResetData()` Methode vorgesehen.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Schädigungsüberwachung [▶ 370]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1,` *not specified**	`UDINT, 2,` `(nBins+2) x (nBinsMean+2)`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2,` `nChannels x` *not specified**	`UDINT, 3,` `nChannels x (nBins+2) x (nBinsMean+2)`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_RainflowCounting_InitPars;  // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_RainflowCounting_InitPars [▶ 315]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError          : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult    : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError        : BOOL;       // TRUE if an error occurs.
    hrErrorCode   : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars      : ST_CM_RainflowCounting_InitPars;  // init parameter
    nOwnID          : UDINT;                            // ID for this FB instance
    aDestIDs        : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
```

```
    nResultBuffers : UDINT := 4;                        // number of MultiArrays which should be init
ialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_RainflowCounting_InitPars [▶ 315]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_MeanStressCorrection [▶ 176] ermöglicht die Durchführung einer Mittelspannungskorrektur bezüglich der Zugfestigkeit des überwachten Bauteils.

Der Algorithmus des Bausteins FB_CMA_MinersRule [▶ 179] kann verwendet werden um den Ermüdungsschaden von Bauteilen auf Basis der gezählten Halbzyklen zu berechnen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.31 FB_CMA_ReadCsvFile

Auslesen einer CSV Datei und sequenzielle Ausgabe der Daten in einen SPS Puffer (Array).

Mit dem TwinCAT 3 Scope können Daten aufgezeichnet, angezeigt und abgespeichert werden. Ein mögliches Speicherformat ist CSV. So können Signaldaten in einer CSV Datei abgelegt werden und später mit dem FB_CMA_ReadCsvFile erneut eingelesen werden. Die hiermit eingelesenen Signaldaten können als alternative Quelle von Eingangsdaten einer Instanz von FB_CMA_Source [▶ 245] und somit der Condition Monitoring Analysekette übergeben werden.

Ein Beispiel zur Verwendung des Funktionsbausteins finden Sie hier: Schädigungsüberwachung [▶ 370].

Ein weiteres Beispiel finden Sie am Ende dieser Seite.

Ein- und Ausgänge

Eingangsparameter

```
VAR_INPUT
    bExecute        : BOOL := TRUE;       // starts execution (on rising edge)
    bAbort          : BOOL;               // aborts execution (after finishing current ADS communica
tion)
    sNetId          : T_AmsNetId := '';   // TwinCAT system network address (could be kept empty for
 local host)
    tAdsTimeout     : TIME := T#1S;       // ADS timeout (only 1 second as condition for a fast data
 transmission)
    sFileName       : T_MaxString := '';  // CSV source file path and name (located on the system wi
th the specified Net Id)
    sSeparator      : STRING(2) := '$T';  // CSV field separator [e.g. tab '$T' or comma ',' or semi
colon ';' or colon ':' or blank ' ']
    nHeaderLines    : UDINT := 0;         // Number of lines in the CSV file which belongs to the he
ader. Data start after header lines.
    bColumn         : BOOL := TRUE;       // select whether the data is placed as column (e.g. TcSco
pe CSV file) or as line
END_VAR
VAR_IN_OUT
    aBuffer         : ARRAY[*] OF LREAL;  // buffer with individual length (do not switch the buffer
 during one execution)
END_VAR
```

- `bExecute`: Mit einer positiven Flanke am Eingang `bExecute` wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
- `bAbort`: Wird dieser Eingang gesetzt, so wird die Ausführung abgebrochen. Der Funktionsbaustein muss noch so lange aufgerufen werden, bis der Ausgang `bBusy=FALSE` ist, so dass eine intern bereits begonnene ADS Kommunikation beendet werden kann.
- `sNetId`: Um die Operation auf dem lokalen Gerät durchzuführen, bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderen Computer zu richten, kann hier dessen AMS Net Id (vom Typ T_AmsNetId) angegeben werden.
- `tAdsTimeout`: Gibt eine maximale Zeitdauer für die Ausführung der internen ADS Kommunikation an.
- `sFileName`: Gibt den Dateipfad der zu lesenden CSV Datei an.
- `sSeparator`: Gibt den CSV Feld Separator an, der in der CSV Datei verwendet ist. Einzelne Werte werden hiermit voneinander separiert.

- `nHeaderLines`: Gibt die Anzahl der Zeilen an, in denen sich nur Header Informationen befinden. Diese werden beim Auslesen übersprungen. Eine mit dem TwinCAT 3 Scope View abgespeicherte CSV Datei kann unterschiedlich umfangreiche Header Informationen beinhalten.
- `bColumn`: Gibt an, ob die Daten sich hintereinander in einer Zeile befinden (`bColumn=FALSE`) oder untereinander als Spalte im CSV vorliegen (`bColumn=TRUE`). Letzteres ist bei CSV Dateien der Fall, die mit dem TwinCAT 3 Scope View abgespeichert wurden.
- `aBuffer`: Gibt den Ausgangsdatenpuffer an, in welchen die gelesenen Werte geschrieben werden. Dabei muss es sich um ein SPS Array vom Typ LREAL handeln mit beliebiger Länge. Wird der `FB_CMA_ReadCsvFile` als Alternative zum Hardware Sensorsignal verwendet, so entspricht die Länge typischerweise dem Oversamplingfaktor. Der angegebene Puffer muss über die Dauer der Operation beibehalten werden. Während der Operation wird der Puffer `aBuffer` mehrfach mit neuen Datensätzen gefüllt und dies über die Ausgangsvariabe `bBufferFull` signalisiert.

Ausgangsparameter

```
VAR_OUTPUT
    bBufferFull     : BOOL;     // signals that aBuffer is fully filled
    nReadRecords    : UDINT;    // shows the number of records saved to aBuffer
    bBusy           : BOOL;     // TRUE if execution is active
    bError          : BOOL;     // TRUE if an error occurred
    hrErrorCode     : HRESULT;  // '< 0' = error; '> 0' = info; '0' = no error/info
    ipErrorMessage  : I_TcMessage := fbErrorMessage;    // shows detailed information about occurred
 errors,warnings and more
END_VAR
```

- `bBufferFull`: Der Ausgang ist `TRUE`, falls der Puffer `aBuffer` vollständig gefüllt wurde. Dies wird über die Dauer der Operation vielfach der Fall sein, so lange bis die CSV Datei vollständig ausgelesen wurde. Ist ein neuer Datensatz im Puffer `aBuffer` vorhanden, sollte dieser sofort weiterverarbeitet werden bevor der Funktionsbaustein erneut aufgerufen wird. Typischerweise liegen nach jedem Aufruf des Funktionsbausteins neue Daten im Puffer bereit. Dennoch sollte der Ausgang `bBufferFull` geprüft werden, um einen korrekten Daten-Stream sicherzustellen.
- `nReadRecords`: Der Ausgang gibt an, wieviele Werte der gesamte gelesene Datensatz enthalten hat.
- `bBusy`: Der Ausgang ist `TRUE` so lange der Funktionsbaustein aktiv ist. Der Ausgang wird `FALSE`, wenn die CSV Datei vollständig gelesen wurde oder ein Fehler auftrat.
- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Beispiel

Das Beispiel zeigt die Verwendung des FB_CMA_ReadCsvFile. Hierbei wird eine große Pufferlänge des bei aBuffer angegebenen Puffers angenommen, was die Auslagerung in eine separate Task begünstigt. Der Puffer wird über das Prozessabbild an die eigentliche SPS Task übergeben und steht dort als Prozessabbildeingang, analog zu einer Oversampling-Klemme, zur Verfügung.

Download des Sample: https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/17670679435.zip

5.1.32 FB_CMA_RealFFT

Berechnung der schnellen Fouriertransformation (FFT) für reell-wertige Eingangssignale.

Der Baustein FB_CMA_RealFFT berechnet die Fourier-Transformierte des am Baustein anliegenden reell-wertigen Eingangssignals *x*[n]. Dabei wird ein performanter FFT-Algorithmus verwendet. Es ist möglich die Hin- und die Rücktransformation zu berechnen. Die Einstellung erfolgt über den Input `stInitPars` (siehe Ein- und Ausgänge).

Definition der Fourier-Hintransformation in DFT-Notation:

$$X[k] = \sum_{n=0}^{N-1} x[n]\, \mathrm{e}^{-\mathrm{i}2\pi nk/N}$$

Definition der Fourier-Rücktransformation in DFT-Notation:

$$x[n] = \frac{1}{N} \sum_{n=0}^{N-1} X[k]\, \mathrm{e}^{\mathrm{i}2\pi nk/N}$$

Die höchste Frequenz einer Komponente im Eingangssignal sollte maximal bei der halben Abtastrate des Eingangssignals liegen, damit Aliasing-Effekte vermieden werden.

Die FFT ist als Transformierte eines zyklisch fortgesetzten Signals definiert. Dies kann zur Feststellung von Sprüngen führen, sobald das zyklisch fortgesetzte Signal nicht exakt kontinuierlich, also am Anfang und am Ende gleich ist. Der Baustein FB_CMA_PowerSpectrum [▶ 202] sowie FB_CMA_MagnitudeSpectrum [▶ 172] vermeiden diese Schwierigkeiten durch eine Analyse in überlappenden, mit einer Fensterfunktion multiplizierten Abschnitten.

Skalierung

Zur messtechnisch quantitativen Auswertung des Spektrums ist das berechnete Spektrum mit `1/nFFT_Length` für den Gleichanteil, also dem ersten Array-Element des Outputs, und mit `2/nFFT_Length` für alle anderen Elemente des Outputs zu gewichten. Der Baustein skaliert bei der Hin-Transformation und der Rücktransformation derart, dass bei aufeinanderfolgender Hin- und Rücktransformation direkt das ursprüngliche Eingangssignal wieder am Ausgang berechnet wird.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: FFT mit reell-wertigem Eingangssignal [▶ 325].

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1, nFFT_Length`	`LCOMPLEX, 1, nFFT_Length/2+1`
Variante mit vollem Spektrum* (`nChannels = 1`)	`LREAL, 1, nFFT_Length`	`LCOMPLEX, 1, nFFT_Length`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2, nChannels x nFFT_Length`	`LCOMPLEX, 2, nChannels x nFFT_Length/2+1`
Mehrkanalige Variante mit vollem Spektrum* (`nChannels > 1`)	`LREAL, 2, nChannels x nFFT_Length`	`LCOMPLEX, 2, nChannels x nFFT_Length`

*: Falls der Ausgangspuffer das volle Spektrum ausgeben soll, so kann dies mit Negieren des Parameters `bHalfSpec (:=FALSE)` erreicht werden.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_RealFFT_InitPars;           // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_RealFFT_InitPars [▶ 315]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                            // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                         // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;   // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                           // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_RealFFT_InitPars;           // init parameter
    nOwnID         : UDINT;                            // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;                       // number of MultiArrays which should be init
ialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_RealFFT_InitPars [▶ 315]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_ComplexFFT [▶ 112] berechnet die Fouriertransformation eines komplexwertigen Eingangssignals.

Der Baustein FB_CMA_PowerSpectrum [▶ 202] berechnet das Leistungsspektrum eines reellwertigen Eingangssignals.

Der Baustein FB_CMA_MagnitudeSpectrum [▶ 172] berechnet das Magnitudenspektrum (Betragsspektrum) eines reellwertigen Eingangssignals.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.33 FB_CMA_RMS

Berechnet für ein- und mehrkanalige reell-wertige Signale den zeitlichen RMS Wert.

Dieser Baustein berechnet den zeitlichen RMS eines oder mehrerer Eingangskanäle. Der RMS wird blockweise über eine interne Pufferlänge *M* gerechnet.

$$x_{\mathrm{RMS}} = \sqrt{\frac{1}{M}\sum_{m=0}^{M-1} x^2[m]}$$

Ist dieser interne Puffer voll, werden die ältesten Werte durch die aktuellen ersetzt und ein neues Ergebnis ausgegeben. Die Menge der zu ersetzenden Eingangswerte hängt von der am Source-Baustein (FB_CMA_Source [▶ 245]) eingestellten MultiArray-Größe ab.

Gedächtniseigenschaften

Für die Berechnung des RMS Wertes werden intern `nBufferLength` Werte der Zeitreihe je Kanal/ Unterkanal gespeichert. Bei einem Aufruf mit kleinerer Eingangspuffergröße können weniger Werte übergeben werden. In diesem Fall wird der Pufferinhalt verschoben und die Signallänge mit der geeigneten Zahl an neu übergebenen Werten aufgefüllt. Ist die Eingangspuffergröße größer als der interne Puffer, so wird dieser mit den jüngsten Werten für die Berechnung gefüllt.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Der Ausgangsvektor wird so lange mit NaN gefüllt, bis der interne Puffer komplett mit neuen gültigen Werten gefüllt wurde.

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Zeitbasierter RMS [▶ 339].

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante für mehrere Datensätze (`nSubChannels = 0`)	`LREAL, 2,` `nChannels x` *not specified**	`LREAL, 1,` `nChannels`
Mehrkanalige Variante für mehrere Datensätze (`nSubChannels > 0`)	`LREAL, 3,` `nChannels x nSubChannels x` *not specified**	`LREAL, 2,` `nChannels x nSubChannels`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorrausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_RMS_InitPars;                // init parameter
    nOwnID           : UDINT;                             // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;   // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                        // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;              // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_RMS_InitPars [▶ 316]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars      : ST_CM_RMS_InitPars;  // init parameter
    nOwnID          : UDINT;               // ID for this FB instance
    aDestIDs        : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;  // number of MultiArrays which should be initialized for results (
0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_RMS_InitPars [▶ 316]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_CrestFactor [▶ 99] berechnet für ein- und mehrkanalige Zeitreihen den Crest-Faktor für jeden Kanal.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.34 FB_CMA_SlidingDFT

Online Berechnung von Spektralwerten für reell-wertige Eingangssignale

Der Baustein FB_CMA_SlidingDFT liefert eine konfigurierbare Anzahl einzelner Spektralwerte, welche konform zu dem Baustein FB_CMA_RealFFT [▶ 216] sind. Die Berechnung der DFT Koeffizienten

$$X_n(k) = \sum_{m=0}^{N-1} x(q+m)e^{-i2\pi km/N}, \qquad q = n - N + 1$$

erfolgt näherungsweise, mit dem Dämpfungsparameter *r*, *0<r<1*, mittels der Rekursionsvorschrift

$$\tilde{X}_n(k) = \sum_{m=0}^{N-1} x(q+m) r^{N-m} e^{-i2\pi km/N} = re^{i2\pi k/N} \left(\tilde{X}_{n-1}(k) - r^N x(n-N) + x(n) \right)$$

Die Berechnung erfolgt durch die Verwendung eines gleitenden Fensters, sodass die Spektralwerte mit jedem neuen Sample aktualisiert werden. Die Vorteile der online Berechnung ausgewählter Spektralwerte gegenüber der FFT besteht zum einen aus der konstanten Auslastung des Prozessors (keine Blockverarbeitung) sowie der Unabhängigkeit der Fensterlänge bezüglich Zweierpotenzen. Die höchste Frequenz einer Komponente im Eingangssignal sollte maximal bei der halben Abtastrate des Eingangssignals liegen, damit Aliasing-Effekte vermieden werden.

Gedächtniseigenschaften

Aufgrund der Verwendung der letzten `nWindowLength` (`N`) Samples für die Berechnung der Spektralwerte können bei der Frequenzanalyse Sprünge in der Zeitreihe zu nicht korrekten Ergebnissen im Spektralbereich führen. Bei einem Aufruf mit kleinerer Eingangspuffergröße können weniger Werte übergeben werden. In diesem Fall werden die konfigurierten Spektralwerte für alle neu übergebenen Werte aktualisiert.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

> **Behandlung von NaN-Werten**
>
> Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bespielimplementierung

Eine beispielhafte Implementierung, welche die Verwendungsmöglichkeiten des Bausteins und dessen Konfiguration zeigt, ist unter folgendem Link verfügbar: Sliding DFT [▶ 378]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1,` *not specified**	`LCOMPLEX, 1,` `nBins`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2,` `nChannels x` *not specified**	`LCOMPLEX, 2,` `nChannels x nBins`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_SlidingDFT_InitPars;        // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;              // timeout checking off during access to i
nter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_SlidingDFT_InitPars [▶ 316]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.

- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Configure():

Mit dem Aufruf dieser Methode müssen die Spektralwerte zu Beginn konfiguriert werden. Hierbei berechnet sich der DFT Index zur Frequenz `f` zu `k = f * fSampleRate / nWindowLength`. Ist `f` kein ganzzahliges Vielfaches der Frequenzauflösung `fSampleRate / nWindowLength` sind die Spektralanteile auf zwei aufeinander folgende Spektralwerte aufgeteilt.

Das entsprechende SPS Array muss wie folgt definiert sein. Für eine erneute Konfiguration mit einem anderen Satz an Argumenten kann die `Configure()` Methode ebenfalls genutzt werden.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : PVOID;    // pointer to array of arguments
    nArgSize : UDINT;    // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle	`UDINT, 1, nBins`
Kanalspezifische Konfiguration (`nChannels > 1`)	`UDINT, 2, nChannels x nBins`

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_SlidingDFT_InitPars;        // init parameter
    nOwnID         : UDINT;                            // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;                       // number of MultiArrays which should be init
ialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_SlidingDFT_InitPars [▶ 316]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.

- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_RealFFT [▶ 216] berechnet die schnelle Fouriertransformation für reell-wertige Eingangssignale.

Der Baustein ST_CM_SparseSpectrum_InitPars [▶ 317] berechnet einzelne, konfigurierbare Magnituden- und Spektralwerte.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.35 FB_CMA_SparseSpectrum

Berechnung einzelner Spektralwerte für reell-wertige Eingangssignale

Der Baustein FB_CMA_SparseSpectrum liefert eine konfigurierbare Anzahl einzelner Spektralwerte, welche konform zu den Bausteinen FB_CMA_RealFFT [▶ 216], FB_CMA_MagnitudeSpectrum [▶ 172] bzw. FB_CMA_PowerSpectrum [▶ 202] skaliert sind. Die Berechnung der DFT Koeffizienten

$$X[k] = \sum_{n=0}^{N-1} x[n]\, \mathrm{e}^{-\mathrm{i}2\pi nk/N}$$

erfolgt mit dem Goertzel-Algorithmus. Dieser ist effizienter als eine radix-2 FFT, falls für die Anzahl *M* der zu berechnenden Koeffizienten bei einer Fensterlänge *L* gilt:

$$M \leq \frac{5N}{6L} \log_2(N), \quad N = 2^m$$

Hierbei ist *N* die nächste größere Potenz von 2 bzgl. der Fensterlänge *L*. Werden nur wenige/einzelne Spektralwerte benötigt, können diese mit dem Funktionsbaustein ggf. direkt in der (schnellen) Sampling-Task berechnet werden, sodass auf spontane Veränderungen des Spektrums schneller reagiert werden kann.

Die höchste Frequenz einer Komponente im Eingangssignal sollte maximal bei der halben Abtastrate des Eingangssignals liegen, damit Aliasing-Effekte vermieden werden.

Der Baustein übernimmt mehrere Funktionen, siehe Analyse von Daten-Streams [▶ 18] und Frequenzanalyse [▶ 37].

Der Eingabedatenpuffer wird zunächst mit den unmittelbar vorhergehenden Puffern überlappend kombiniert und mit einer Fensterfunktion multipliziert. Anschließend werden die DFT Koeffizienten über den Goertzel-Algorithmus berechnet. Über den Parameter `eSpectrumType` wird definiert ob von den resultierenden komplexen Werten der Absolutbetrag oder dessen Quadrat berechnet werden. Wenn der Parameter `bTransformToDecibel` den Wert `TRUE` hat werden die Werte zu Dezibel -Werten transformiert, falls Magnituden- oder Leistungswerte berechnet wurden. Diese Dezibel-Werte sind für beide Spektralwerte gleich, d.h. der Einfluss der Quadrierung bei den Leistungswerten wird bei der Berechnung der Dezibel-Werte durch einen Faktor zwei für die Magnitudenwerte berücksichtigt. Des Weiteren ist eine Skalierung der Ergebnisse über den Parameter `eScalingType` durchführbar, siehe dazu Skalierung von Spektren [▶ 26].

Skalierung

Die Skalierung der erhaltenen Ergebniswerte, also z. B. der Beschleunigungsdichten (Acceleration Spectral Densities) entspricht standardmäßig der Definition der FFT. Dies bedeutet, dass der Einfluss der Fensterlänge und der Fensterfunktion herausgerechnet werden.
Zur rechnerischen Skalierung absoluter Messungen können tabellierte Parameter verwendet werden, die im Abschnitt "Optionen der Spektrumsskalierung [▶ 390]“ beschrieben sind.

Gedächtniseigenschaften

Aufgrund der Verwendung der Welch-Methode wird jeweils der aktuelle Eingangsdatenpuffer zusammen mit den zuletzt übergebenen Puffern zur Berechnung genutzt. Die Anzahl der einfließenden Puffer ist abhängig von der gewählten Überlappung (`nOverlap`).
Die Frequenzanalyse berücksichtigt Sprünge in der Zeitreihe. Um ein korrektes Ergebnis zu erzielen, müssen sich deswegen die verwendeten Eingangsdatenpuffer lückenlos und ohne Sprünge aneinanderreihen.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bespielimplementierung

Eine beispielhafte Implementierung, welche die Verwendungsmöglichkeiten des Bausteins und dessen Konfiguration zeigt, ist unter folgendem Link verfügbar: Berechnung einzelner Spektralwerte [▶ 333]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante DFT (`nChannels = 1`, `eSpectrumType = eCM_DFT`)	`LREAL, 1, nWindowLength-nOverlap`	`LCOMPLEX, 1, nBins`
Standardvariante Spektrum (`nChannels = 1`, `eSpectrumType <> eCM_DFT`)	`LREAL, 1, nWindowLength-nOverlap`	`LREAL, 1, nBins`
Mehrkanalige Variante DFT (`nChannels > 1`, `eSpectrumType = eCM_DFT`)	`LREAL, 2, nChannels x nWindowLength-nOverlap`	`LCOMPLEX, 2, nChannels x nBins`
Mehrkanalige Variante Spektrum (`nChannels > 1`, `eSpectrumType <> eCM_DFT`)	`LREAL, 2, nChannels x nWindowLength-nOverlap`	`LREAL, 2, nChannels x nBins`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_SparseSpectrum_InitPars;     // init parameter
    nOwnID           : UDINT;                             // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;   // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                        // number of MultiArrays which should be i
nitialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;              // timeout checking off during access to i
nter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_SparseSpectrum_InitPars [▶ 317]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.

- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Configure():

Mit dem Aufruf dieser Methode müssen die Spektralwerte zu Beginn konfiguriert werden. Hierbei berechnet sich der DFT Index zur Frequenz `f` zu `k = f * fSampleRate / nWindowLength`. Ist `f` kein ganzzahliges Vielfaches der Frequenzauflösung `fSampleRate / nWindowLength` sind die Spektralanteile auf zwei aufeinander folgende Spektralwerte aufgeteilt.

Das entsprechende SPS Array muss wie folgt definiert sein. Für eine erneute Konfiguration mit einem anderen Satz an Argumenten kann die `Configure()` Methode ebenfalls genutzt werden.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : PVOID;    // pointer to array of arguments
    nArgSize : UDINT;    // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle	`UDINT, 1, nBins`
Kanalspezifische Konfiguration (`nChannels > 1`)	`UDINT, 2, nChannels x nBins`

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_SparseSpectrum_InitPars;    // init parameter
    nOwnID         : UDINT;                            // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;                       // number of MultiArrays which should be init
ialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_SparseSpectrum_InitPars [▶ 317]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_RealFFT [▶ 216] berechnet die schnelle Fouriertransformation für reell-wertige Eingangssignale.

Der Baustein FB_CMA_MagnitudeSpectrum [▶ 172] berechnet das Magnitudenspektrum eines reell-wertigen Eingangssignals.

Der Baustein FB_CMA_PowerSpectrum [▶ 202] berechnet das Leistungsspektrum mit einer Quadrierung der Werte im letzten Schritt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.36 FB_CMA_SpikeEnergySpectrum

Analyse von Spitzenenergie hochfrequenter Signalanteile

Der Baustein FB_CMA_SpikeEnergySpectrum realisiert eine weit verbreitete Methode zur Vibrationsübermachung an Maschinenelementen. Die Methode analysiert hochfrequente Signalanteile, insbesondere Frequenzen jenseits des linearen Übertragungsbereich eines Beschleunigungssensors, und ist dadurch sehr sensitiv bei der frühen Erkennung von Schadensbildern.

Die Methode basiert auf der Auswertung eines Beschleunigungssignals, wobei Signalanteile betrachtet werden, die jenseits der Resonanzfrequenz des Beschleunigungssensors liegen. Entsprechend erscheinen hochfrequente impulsförmige Störungen, entstanden z.B. beim Überrollen einer kleinen Schädigung im Wälzlager, als Modulation auf einem Trägersignal. Das Trägersignal wird durch die Resonanzfrequenz des

montierten Beschleunigungssensors bestimmt. Die Spike Energy Methode filtert das Trägersignal heraus und bildet eine Einhüllende über die hochfrequenten impulsförmigen Störungen. Damit lassen sie die Schadfrequenzen einer Komponente, z.B. eines Wälzlagers, prägnanter im Spektrum erkennen.

Da die Methode stark von der Montage, dem Ort und dem Sensortyp abhängt ist die Methode ausschließlich für eine Trendüberwachung geeignet, d.h. eine Übertragung von Schwellwerten auf andere Messstellen ist nicht möglich.

Der Eingabedatenpuffer des Bausteins FB_CMA_SpikeEnergySpectrum wird zunächst mit den unmittelbar vorhergehenden Puffern überlappend kombiniert und mit einer Fensterfunktion multipliziert. Anschließend werden die DFT Koeffizienten über den Goertzel-Algorithmus berechnet. Über den Parameter `eSpectrumType` wird definiert ob von den resultierenden komplexen Werten der Absolutbetrag oder dessen Quadrat berechnet werden. Wenn der Parameter `bTransformToDecibel` den Wert `TRUE` hat werden die Werte zu Dezibel -Werten transformiert. Diese Dezibel-Werte sind für beide Spektralwerte (Betrag oder dessen Amplitude) gleich, d.h. der Einfluss der Quadrierung bei den Leistungswerten wird bei der Berechnung der Dezibel-Werte durch einen Faktor zwei für die Magnitudenwerte berücksichtigt. Des Weiteren ist eine Skalierung der Ergebnisse über den Parameter `eScalingType` durchführbar, siehe dazu Skalierung von Spektren [▶ 26].

Skalierung

Die Skalierung der erhaltenen Ergebniswerte, also z. B. der Beschleunigungsdichten (Acceleration Spectral Densities) entspricht standardmäßig der Definition der FFT. Dies bedeutet, dass der Einfluss der Fensterlänge und der Fensterfunktion herausgerechnet werden.
Zur rechnerischen Skalierung absoluter Messungen können tabellierte Parameter verwendet werden, die im Abschnitt "Optionen der Spektrumsskalierung [▶ 390]“ beschrieben sind.

Gedächtniseigenschaften

Aufgrund der Verwendung der Welch-Methode wird jeweils der aktuelle Eingangsdatenpuffer zusammen mit den zuletzt übergebenen Puffern zur Berechnung genutzt. Die Anzahl der einfließenden Puffer ist abhängig von der gewählten Überlappung (`nOverlap`).
Die Frequenzanalyse berücksichtigt Sprünge in der Zeitreihe. Um ein korrektes Ergebnis zu erzielen, müssen sich deswegen die verwendeten Eingangsdatenpuffer lückenlos und ohne Sprünge aneinanderreihen.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bespielimplementierung

Eine beispielhafte Implementierung, welche die Verwendungsmöglichkeiten des Bausteins und dessen Konfiguration zeigt, ist unter folgendem Link verfügbar: Spitzenwert Spektrum [▶ 373]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1, nWindowLength-nOverlap`	`LREAL, 1, nFFT_Length/2+1`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2, nChannels x nWindowLength-nOverlap`	`LREAL, 2, nChannels x nFFT_Length/2+1`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_SpikeEnergySpectrum_InitPars;  // init parameter
    nOwnID           : UDINT;                               // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;     // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                          // number of MultiArrays which should be
 initialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;                // timeout checking off during access to
 inter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_SpikeEnergySpectrum_InitPars [▶ 318]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Configure():

Mit dem Aufruf dieser Methode können die Abklingzeit (`fDecayTime`) sowie das betrachtete Frequenzband (via `fLowerFrequencyLimit`, `fUpperFrequencyLimit`) zur Laufzeit angepasst werden. Die Ablinkzeit sollte im optimalen Fall so gewählt werden, dass die Spitzenenergie vollständig abklingen kann, d.h. `fDecayTime > 1/f_fault`.

Das entsprechende SPS Array muss wie folgt definiert sein: [`fDecayTime, fLowerFrequencyLimit, fUpperFrequencyLimit`]. Für eine erneute Konfiguration mit einem anderen Satz an Argumenten kann die `Configure()` Methode ebenfalls genutzt werden.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : PVOID;    // pointer to array of arguments
    nArgSize : UDINT;    // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle	`LREAL, 1, 3`
Kanalspezifische Konfiguration (`nChannels > 1`)	`LREAL, 2, nChannels x 3`

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_SpikeEnergySpectrum_InitPars;  // init parameter
    nOwnID         : UDINT;                               // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;     // IDs of destinations for output
    nResultBuffers : UDINT := 4;                          // number of MultiArrays which should be i
nitialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_SpikeEnergySpectrum_InitPars [▶ 318]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_PowerCepstrum [▶ 198] berechnet das Leistungscepstrum eines reell-wertigen Eingangssignals.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.37 FB_CMA_Sink

Dieser Funktionsbaustein schreibt Daten aus einem MultiArray Puffer in einen externen SPS Datenpuffer.

Er erhält alle MultiArrays, die an die angegebene Analyse-ID übergeben werden. Je nach Analysekette können die Ausgangsergebnisse NaN Werte enthalten.

HINWEIS

Exception

Vergleiche mit NaN (Not a Number) können eine Exception verursachen, die zu einem Ausführungsstopp führt und möglicherweise einen Maschinenschaden verursachen. Es wird strengstens empfohlen, das Ergebnis für NaN zu überprüfen, bevor es verarbeitet wird. Oder wenn NaNs in der Anwendung abgearbeitet werden sollen, müssen die Floating Point Exceptions für diese Task deaktiviert werden.

Ein- und Ausgänge

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    nOwnID           : UDINT;                // ID for this FB instance
    tTransferTimeout : LTIME := LTIME#40US;  // timeout checking off during access to inter-
task FIFOs
END_VAR
```

- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.

- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Eigenschaften

Folgende Eigenschaften geben Informationen zu den zur letzten Ergebnisausgabe zugehörigen Eingangswerten der Analysekette. Siehe hierzu auch die mögliche Parametrierung am FB_CMA_Source [▶ 245].

Name	Typ	Zugriff	Beschreibung
nSourceTimestampNewest	ULINT [T_DCTIME64; 1ns steps since 1.1.2000]	Get	Zugehörig zur letzten Ergebnisausgabe, gibt dies den Zeitstempel des neuesten Eingangswertes der Analysekette aus. Es handelt sich um die Obergrenze (exklusive) der Zeitspanne.
nSourceTimestampOldest	ULINT [T_DCTIME64; 1ns steps since 1.1.2000]	Get	Zugehörig zur letzten Ergebnisausgabe, gibt dies den Zeitstempel des ältesten Eingangswertes der Analysekette aus. Es handelt sich um die Untergrenze (inklusive) der Zeitspanne.
nSourceValues	ULINT	Get	Zugehörig zur letzten Ergebnisausgabe, gibt dies die Anzahl der Eingangswerte (Signalwerte) der Analysekette aus.
tSourceTimespan	LTIME	Get	Zugehörig zur letzten Ergebnisausgabe, gibt dies die Zeitspanne der Eingangswerte (Signalwerte) der Analysekette aus. tSourceTimespan := nSourceValues * tSamplePeriod;

Methoden

Output1D():

Schreibt Daten von einem MultiArray in einen externen eindimensionalen Datenpuffer. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes erläutert.

```
METHOD Output1D : HRESULT
VAR_INPUT
    pDataOut       : POINTER TO BYTE;       // address of data buffer
    nDataOutSize   : UDINT;                 // size of data buffer in bytes
    eElementType   : E_MA_ElementTypeCode;
    nWorkDim       : UDINT:=0;              // It designates the dimension in the MultiArray being pro
cessed.
    nElements      : UDINT:=0;              // optional: default:0->full
```

```
 copy; It designates the number of elements to be copied out of the MultiArray.
    pStartIndex    : POINTER TO UDINT;     (* optional: default:0->internally handled as [0,0,..];
                                            It designates the index of the first element to be copied
out of the MultiArray.
                                            If allocated it must point to a onedimensional array of UD
INT with so many elements as dimensions of the MultiArray. *)
    nOptionPars    : DWORD;                // option mask
END_VAR
VAR_OUTPUT
    bNewResult     : BOOL;                 // TRUE every time when data was written from MultiArray t
o data buffer.
    bError         : BOOL;                 // TRUE if an error occurs.
    hrErrorCode    : HRESULT;              // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `eElementType`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode. Der Elementtyp des gegebenen MultiArray Puffers muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.
- `nWorkDim`: Gibt die Dimension des zu verarbeitenden MultiArrays an. Diesen Daten werden in den angegebenen externen Datenpuffer kopiert. Im Allgemeinen ist das MultiArray auch eindimensional und `nWorkDim:=0`, aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann aber nicht kopiert werden dürfen.
- `nElements`: Gibt die Anzahl Elemente an, die aus dem MultiArray zu kopieren sind. `nElements:=0` ist einzustellen, um alles zu kopieren. Wenn Sie allerdings nur an einer bestimmten Bandbreite Ihres Ergebnisses interessiert sind, dann ist es nicht notwendig, die gesamte Datenmenge zu kopieren. Dies verringert ebenfalls die notwendige Größe Ihres angegebenen externen Datenpuffers.
- `pStartIndex`: Dies ist ein optionaler Parameter, der nützlich ist, wenn das MultiArray mehr als eine Dimension hat oder wenn nicht alle Elemente kopiert werden sollen. Gibt den Index des ersten Elements an, das aus dem MultiArray zu kopieren ist. Wenn zugewiesen, dann muss es auf ein eindimensionales Array von `UDINT` zeigen, das so viele Elemente hat, wie das MultiArray Dimensionen hat.
- `bNewResult`: Dieser Ausgang ist jedes Mal `TRUE`, wenn ein neues Ergebnis erfolgreich in den Datenpuffer geschrieben wurde.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.
- `hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ `HRESULT` angezeigt. Mögliche Werte sind in der Liste der Fehlercodes definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.
  Tipp: Wenn ein Timeout auftritt, gehen die Eingangsdaten nicht verloren. Sie werden beim nächsten Aufruf verarbeitet.

Output1DStd():

Schreibt Daten von einem MultiArray in einen externen eindimensionalen Datenpuffer. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

Diese Methode verwendet Standardwerte für die Parameter `nWorkDim:=0`, `nElements:=0` und `pStartIndex:=0`.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes erläutert.

```
METHOD Output1DStd : HRESULT
VAR_INPUT
    pDataOut       : POINTER TO BYTE;      // address of data buffer
    nDataOutSize   : UDINT;                // size of data buffer in bytes
    eElementType   : E_MA_ElementTypeCode;
    nOptionPars    : DWORD;                // option mask
END_VAR
VAR_OUTPUT
    bNewResult     : BOOL;                 // TRUE every time when data was written from MultiArray t
o data buffer.
    bError         : BOOL;                 // TRUE if an error occurs.
    hrErrorCode    : HRESULT;              // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `eElementType`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode. Der Elementtyp des gegebenen MultiArray Puffers muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.

- `bNewResult`: Dieser Ausgang ist jedes Mal `TRUE`, wenn ein neues Ergebnis erfolgreich in den Datenpuffer geschrieben wurde.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.
- `hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ `HRESULT` angezeigt. Mögliche Werte sind in der Liste der Fehlercodes definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.
  Tipp: Wenn ein Timeout auftritt, gehen die Eingangsdaten nicht verloren. Sie werden beim nächsten Aufruf verarbeitet.

Output2D():

Schreibt Daten von einem MultiArray in einen externen zweidimensionalen Datenpuffer. Wird die Methode weder mit einem Hinweis auf neues Ergebnis, noch mit einem Fehler zurückgegeben, dann wartet das Objekt auf Eingangsdaten. Das ist ein reguläres Verhalten in der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Output2D : HRESULT
VAR_INPUT
    pDataOut      : POINTER TO BYTE;  // address of data buffer
    nDataOutSize  : UDINT;            // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode;
    nWorkDim0     : UDINT:=0;         // It designates the first dimension in the MultiArray being p
rocessed.
    nWorkDim1     : UDINT:=1;         // It designates the second dimension in the MultiArray being
processed.
    nElementsDim0 : UDINT:=0;         // optional: default:0-
>full copy; It designates the number of elements to be copied out of WorkDim0 of the MultiArray.
    nElementsDim1 : UDINT:=0;         // optional: default:0-
>full copy; It designates the number of elements to be copied out of WorkDim1 of the MultiArray.
    pStartIndex   : POINTER TO UDINT; (* optional: default:0->internally handled as [0,0,..];
                                      It designates the index of the first element to be copied out
of the MultiArray.
                                      If allocated it must point to a onedimensional array of UDINT
with so many elements as dimensions of the MultiArray. *)
    nOptionPars   : DWORD;           // option mask
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;            // TRUE every time when data was written from MultiArray to dat
a buffer.
    bError        : BOOL;            // TRUE if an error occurs.
    hrErrorCode   : HRESULT;         // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataOut`: Adresse des externen zweidimensionalen Datenpuffers.
- `eElementType`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode [▶ 292]. Der Elementtyp des gegebenen MultiArray Puffers muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.
- `nWorkDim0`: Gibt die erste Dimension des zu verarbeitenden MultiArray an. Diesen Daten werden in die erste Dimension des angegebenen externen Datenpuffers kopiert. Im Allgemeinen ist das MultiArray auch zweidimensional und `nWorkDim0:=0`, aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann aber nicht kopiert werden dürfen.
  Z.B.: Wenn die zweite Dimension in den ersten Index des Zieldatenpuffers kopiert werden soll, dann setzen Sie `nWorkDim0:=1`.
- `nWorkDim1`: Gibt die zweite Dimension des zu verarbeitenden MultiArray an. Diesen Daten werden in die zweite Dimension des angegebenen externen Datenpuffers kopiert. Im Allgemeinen ist das MultiArray auch zweidimensional und `nWorkDim1:=1`, aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann aber nicht kopiert werden dürfen.
  Z.B.: Wenn die erste Dimension in den zweiten Index des Zieldatenpuffers kopiert werden soll, dann setzen Sie `nWorkDim1:=0`.
- `nElementsDim0`: Gibt die Anzahl Elemente an, die aus `nWorkDim0` des MultiArrays zu kopieren sind. `nElementsDim0:=0` ist einzustellen, um alles zu kopieren. Wenn Sie allerdings nur an einer bestimmten Bandbreite Ihres Ergebnisses interessiert sind, dann ist es nicht notwendig, die gesamte Datenmenge zu kopieren. Dies verringert ebenfalls die notwendige Größe Ihres angegebenen externen Datenpuffers.

- `nElementsDim1`: Gibt die Anzahl Elemente an, die aus `nWorkDim0` des MultiArrays zu kopieren sind. `nElementsDim1:=0` ist einzustellen, um alles zu kopieren. Wenn Sie allerdings nur an einer bestimmten Bandbreite Ihres Ergebnisses interessiert sind, dann ist es nicht notwendig, die gesamte Datenmenge zu kopieren. Dies verringert ebenfalls die notwendige Größe Ihres angegebenen externen Datenpuffers.
- `pStartIndex`: Dies ist ein optionaler Parameter, der nützlich ist, wenn das MultiArray mehr als zwei Dimensionen hat oder wenn nicht alle Elemente kopiert werden sollen. Gibt den Index des ersten Elements an, das aus dem MultiArray zu kopieren ist. Wenn zugewiesen, dann muss es auf ein eindimensionales Array von `UDINT` zeigen, das so viele Elemente hat, wie das MultiArray Dimensionen hat.
- `bNewResult`: Dieser Ausgang ist jedes Mal `TRUE`, wenn ein neues Ergebnis erfolgreich in den Datenpuffer geschrieben wurde.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.
- `hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ `HRESULT` angezeigt. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.
  Tipp: Wenn ein Timeout auftritt, gehen die Eingangsdaten nicht verloren. Sie werden beim nächsten Aufruf verarbeitet.

Output2DStd():

Schreibt Daten von einem MultiArray in einen externen zweidimensionalen Datenpuffer. Wird die Methode weder mit einem Hinweis auf neues Ergebnis, noch mit einem Fehler zurückgegeben, dann wartet das Objekt auf Eingangsdaten. Das ist ein reguläres Verhalten in der Analysekette.

Diese Methode verwendet Standardwerte für die Parameter `nWorkDim0:=0`, `nWorkDim1:=1`, `nElementsDim0:=0`, `nElementsDim1:=0` und `pStartIndex:=0`.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Output2DStd : HRESULT
VAR_INPUT
    pDataOut       : POINTER TO BYTE;  // address of data buffer
    nDataOutSize   : UDINT;            // size of data buffer in bytes
    eElementType   : E_MA_ElementTypeCode;
    nOptionPars    : DWORD;           // option mask
END_VAR
VAR_OUTPUT
    bNewResult     : BOOL;            // TRUE every time when data was written from MultiArray to dat
a buffer.
    bError         : BOOL;            // TRUE if an error occurs.
    hrErrorCode    : HRESULT;         // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataOut`: Adresse des externen zweidimensionalen Datenpuffers.
- `eElementType`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode [▶ 292]. Der Elementtyp des gegebenen MultiArray Puffers muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.
- `bNewResult`: Dieser Ausgang ist jedes Mal `TRUE`, wenn ein neues Ergebnis erfolgreich in den Datenpuffer geschrieben wurde.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.
- `hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ `HRESULT` angezeigt. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.
  Tipp: Wenn ein Timeout auftritt, gehen die Eingangsdaten nicht verloren. Sie werden beim nächsten Aufruf verarbeitet.

Output3D():

Schreibt Daten von einem MultiArray in einen externen dreidimensionalen Datenpuffer. Wird die Methode weder mit einem Hinweis auf neues Ergebnis, noch mit einem Fehler zurückgegeben, dann wartet das Objekt auf Eingangsdaten. Das ist ein reguläres Verhalten in der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Output3D : HRESULT
VAR_INPUT
    pDataOut      : POINTER TO BYTE;  // address of data buffer
    nDataOutSize  : UDINT;            // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode;
    nWorkDim0     : UDINT:=0;         // It designates the first dimension in the MultiArray being p
rocessed.
    nWorkDim1     : UDINT:=1;         // It designates the second dimension in the MultiArray being
processed.
    nWorkDim2     : UDINT:=1;         // It designates the third dimension in the MultiArray being p
rocessed.
    nElementsDim0 : UDINT:=0;         // optional: default:0-
>full copy; It designates the number of elements to be copied out of WorkDim0 of the MultiArray.
    nElementsDim1 : UDINT:=0;         // optional: default:0-
>full copy; It designates the number of elements to be copied out of WorkDim1 of the MultiArray.
    nElementsDim2 : UDINT:=0;         // optional: default:0-
>full copy; It designates the number of elements to be copied out of WorkDim2 of the MultiArray.
    pStartIndex   : POINTER TO UDINT; (* optional: default:0->internally handled as [0,0,..];
                                      It designates the index of the first element to be copied out
of the MultiArray.
                                      If allocated it must point to a onedimensional array of UDINT
with so many elements as dimensions of the MultiArray. *)
    nOptionPars   : DWORD;           // option mask
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;            // TRUE every time when data was written from MultiArray to dat
a buffer.
    bError        : BOOL;            // TRUE if an error occurs.
    hrErrorCode   : HRESULT;         // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataOut`: Adresse des externen zweidimensionalen Datenpuffers.
- `eElementType`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode [▶ 292]. Der Elementtyp des gegebenen MultiArray Puffers muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.
- `nWorkDim0`: Gibt die erste Dimension des zu verarbeitenden MultiArray an. Diese Daten werden in die erste Dimension des angegebenen externen Datenpuffers kopiert. Im Allgemeinen ist das MultiArray auch zweidimensional und `nWorkDim0:=0`, aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann aber nicht kopiert werden dürfen.
  Z.B.: Wenn die zweite Dimension in den ersten Index des Zieldatenpuffers kopiert werden soll, dann setzen Sie `nWorkDim0:=1`.
- `nWorkDim1`: Gibt die zweite Dimension des zu verarbeitenden MultiArray an. Diese Daten werden in die zweite Dimension des angegebenen externen Datenpuffers kopiert. Im Allgemeinen ist das MultiArray auch zweidimensional und `nWorkDim1:=1`, aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann aber nicht kopiert werden dürfen.
  Z.B.: Wenn die erste Dimension in den zweiten Index des Zieldatenpuffers kopiert werden soll, dann setzen Sie `nWorkDim1:=0`.
- `nWorkDim2`: Gibt die dritte Dimension des zu verarbeitenden MultiArray an. Diese Daten werden in die dritte Dimension des angegebenen externen Datenpuffers kopiert. Im Allgemeinen ist das MultiArray auch dreidimensional und `nWorkDim2:=2`, aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann aber nicht kopiert werden dürfen.
  Z.B.: Wenn die erste Dimension in den zweiten Index des Zieldatenpuffers kopiert werden soll, dann setzen Sie `nWorkDim2:=0`.
- `nElementsDim0`: Gibt die Anzahl Elemente an, die aus `nWorkDim0` des MultiArrays zu kopieren sind. `nElementsDim0:=0` ist einzustellen, um alles zu kopieren. Wenn Sie allerdings nur an einer bestimmten Bandbreite Ihres Ergebnisses interessiert sind, dann ist es nicht notwendig, die gesamte Datenmenge zu kopieren. Dies verringert ebenfalls die notwendige Größe Ihres angegebenen externen Datenpuffers.
- `nElementsDim1`: Gibt die Anzahl Elemente an, die aus `nWorkDim1` des MultiArrays zu kopieren sind. `nElementsDim1:=0` ist einzustellen, um alles zu kopieren. Wenn Sie allerdings nur an einer bestimmten Bandbreite Ihres Ergebnisses interessiert sind, dann ist es nicht notwendig, die gesamte Datenmenge zu kopieren. Dies verringert ebenfalls die notwendige Größe Ihres angegebenen externen Datenpuffers.

- `nElementsDim2`: Gibt die Anzahl Elemente an, die aus `nWorkDim2` des MultiArrays zu kopieren sind. `nElementsDim2:=0` ist einzustellen, um alles zu kopieren. Wenn Sie allerdings nur an einer bestimmten Bandbreite Ihres Ergebnisses interessiert sind, dann ist es nicht notwendig, die gesamte Datenmenge zu kopieren. Dies verringert ebenfalls die notwendige Größe Ihres angegebenen externen Datenpuffers.
- `pStartIndex`: Dies ist ein optionaler Parameter, der nützlich ist, wenn das MultiArray mehr als zwei Dimensionen hat oder wenn nicht alle Elemente kopiert werden sollen. Gibt den Index des ersten Elements an, das aus dem MultiArray zu kopieren ist. Wenn zugewiesen, dann muss es auf ein eindimensionales Array von `UDINT` zeigen, das so viele Elemente hat, wie das MultiArray Dimensionen hat.
- `bNewResult`: Dieser Ausgang ist jedes Mal `TRUE`, wenn ein neues Ergebnis erfolgreich in den Datenpuffer geschrieben wurde.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.
- `hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ `HRESULT` angezeigt. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.
  Tipp: Wenn ein Timeout auftritt, gehen die Eingangsdaten nicht verloren. Sie werden beim nächsten Aufruf verarbeitet.

Output3DStd():

Schreibt Daten von einem MultiArray in einen externen dreidimensionalen Datenpuffer. Wird die Methode weder mit einem Hinweis auf neues Ergebnis, noch mit einem Fehler zurückgegeben, dann wartet das Objekt auf Eingangsdaten. Das ist ein reguläres Verhalten in der Analysekette.

Diese Methode verwendet Standardwerte für die Parameter `nWorkDim0:=0`, `nWorkDim1:=1`, `nWorkDim2:=2`, `nElementsDim0:=0`, `nElementsDim1:=0`, `nElementsDim2:=0` und `pStartIndex:=0`.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Output3DStd : HRESULT
VAR_INPUT
    pDataOut       : POINTER TO BYTE;  // address of data buffer
    nDataOutSize   : UDINT;             // size of data buffer in bytes
    eElementType   : E_MA_ElementTypeCode;
    nOptionPars    : DWORD;            // option mask
END_VAR
VAR_OUTPUT
    bNewResult     : BOOL;             // TRUE every time when data was written from MultiArray to dat
a buffer.
    bError         : BOOL;             // TRUE if an error occurs.
    hrErrorCode    : HRESULT;          // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataOut`: Adresse des externen zweidimensionalen Datenpuffers.
- `eElementType`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode [▶ 292]. Der Elementtyp des gegebenen MultiArray Puffers muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.
- `bNewResult`: Dieser Ausgang ist jedes Mal `TRUE`, wenn ein neues Ergebnis erfolgreich in den Datenpuffer geschrieben wurde.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.
- `hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ `HRESULT` angezeigt. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.
  Tipp: Wenn ein Timeout auftritt, gehen die Eingangsdaten nicht verloren. Sie werden beim nächsten Aufruf verarbeitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    nOwnID: UDINT; // ID for this FB instance
END_VAR
```

- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

5.1.38 FB_CMA_Source

Dieser Funktionsbaustein schreibt Daten aus einem externen SPS Datenpuffer in einen MultiArray Puffer.

Er häuft ununterbrochen Eingangsdaten an, bis die Größe des MultiArray erreicht ist. Wenn das MultiArray komplett gefüllt ist, wird es an die Zielanalyse-ID übergeben.

Eine Instanz von FB_CMA_Source darf nicht als Ziel für irgendeinen anderen Analysebaustein verwendet werden. Sie bietet ausschließlich Quellfunktionalitäten.

Eine Zeitreihensammlung kann im Falle eines Fehlerauftritts unterbrochen werden. Verlorene Signaldaten können zu einem unerwarteten Ergebnis der Analysekette führen, je nach Konfiguration der Algorithmen.

Ein- und Ausgänge

Die Ausgangspuffer entsprechen folgender Definition (Shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

MultiArray im	Elementtyp	Dimensionen	Dimensionsgrößen
output stream	eTypeCode	`nDims`	`aDimSizes`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_MA_MultiArray_InitPars;       // init parameter
    nOwnID           : UDINT;                           // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                      // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#40US;             // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- `stInitPars`: Funktionsbausteinspezifische Struktur der Initialisierungsparameter vom Typ ST_MA_MultiArray_InitPars [▶ 322]. MultiArray Puffer werden für die Ergebnispuffer spezifiziert. Diese Parameter müssen mit der obigen Definition der Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Eigenschaften

Folgende Eigenschaften können optional verwendet werden, um den Zeitstempel der Eingangswerte zugehörig zum nächsten Input-Aufruf zu spezifizieren. Werden die Eigenschaften nicht gesetzt, so wird intern automatisch ein aktueller Zeitstempel und eine typische Zeitdifferenz zwischen zwei Eingangswerten ermittelt.
Am FB_CMA_Sink [▶ 238] werden Zeitspanne und Zeitstempel der Eingangswerte der Analysekette zugeordnet zum Ergebnis der Analyse bereitgestellt.

Name	Typ	Zugriff	Beschreibung
nTimestamp	ULINT [T_DCTIME64; 1ns steps since 1.1.2000]	Set	Angabe des Zeitstempels zugehörig zum ältesten Wert, welcher mit dem nächsten Input-Aufruf übergeben wird.

Name	Typ	Zugriff	Beschreibung
			(Beispiel: StartTimeNextLatch -CycleTime)
tSamplePeriod	LTIME	Set	Angabe der Zeitdifferenz zwischen zwei Eingangswerten. (Beispiel: CycleTime / cOversamples). Diese Angabe ist nur einmalig nötig, da es sich um einen konstanten Wert handelt.

Methoden

Input1D():

Schreibt Daten von einem externen eindimensionalen Datenpuffer in ein MultiArray. Jedes Mal wenn neue Eingangsabtastungen vorliegen, muss diese Methode aufgerufen werden, normalerweise zyklisch.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Input1D : HRESULT
VAR_INPUT
    pDataIn       : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSize   : UDINT;                // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode;
    nWorkDim      : UDINT;                // It designates the dimension in the multi array being pr
ocessed.
    pStartIndex   : POINTER TO UDINT;     (* optional: default:0-
>internally handled; It designates the index of the first MultiArray element to be copied.
                                             If allocated it must point to a onedimensional array of
 UDINT with so many elements as dimensions of the MultiArray.
                                             Upon successful completion of the copy, corresponding S
tartIndex is incremented by the number of copied elements. *)
    nOptionPars   : DWORD;                // option mask
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;                 // TRUE every time when outgoing MultiArray was calculated
 and sent to transfer tray.
    bError        : BOOL;                 // TRUE if an error occurs.
    hrErrorCode   : HRESULT;              // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataIn`: Der Datenpuffer muss die Daten von allen Kanälen enthalten.
- `eElementType`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode [▶ 292]. Der Elementtyp des spezifizierten MultiArray Puffers (Initialisierungsparameter) muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.
- `nWorkDim`: Definiert die Dimension, in welcher die Daten angesammelt werden. Im Allgemeinen ist das MultiArray auch eindimensional und `nWorkDim := 0`, aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann jedoch nicht verarbeitet werden dürfen.
- `pStartIndex`: Dies ist ein optionaler Parameter, der dann nützlich ist, wenn das MultiArray mehr als eine Dimension erhalten hat. Gibt den Index des ersten zu kopierenden MultiArray Elements an. Wenn zugewiesen, dann muss es auf ein eindimensionales Array von `UDINT` zeigen, das so viele Elemente hat, wie das MultiArray Dimensionen hat. Im Anschluss an einen erfolgreichen Kopiervorgang wird der entsprechende Start-Index um die Anzahl der kopierten Elemente inkrementiert.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.
- `hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ HRESULT angezeigt. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.

Input1DStd():

Schreibt Daten von einem externen eindimensionalen Datenpuffer in ein MultiArray. Jedes Mal wenn neue Eingangsabtastungen vorliegen, muss diese Methode aufgerufen werden, normalerweise zyklisch.

Diese Methode verwendet Standardwerte für die Parameter `nWorkDim := 0` und `pStartIndex := 0`.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Input1DStd : HRESULT
VAR_INPUT
    pDataIn       : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSize   : UDINT;                // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode;
    nOptionPars   : DWORD;                // option mask
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;                 // TRUE every time when outgoing MultiArray was calculated
 and sent to transfer tray.
    bError        : BOOL;                 // TRUE if an error occurs.
    hrErrorCode   : HRESULT;              // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataIn`: Der Datenpuffer muss die Daten von allen Kanälen enthalten.
- `eElementType`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode [▶ 292]. Der Elementtyp des spezifizierten MultiArray Puffers (Initialisierungsparameter) muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.
- `hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ HRESULT angezeigt. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.

Input2D():

Schreibt Daten von einem externen zweidimensionalen Datenpuffer in ein MultiArray. Jedes Mal wenn neue Eingangsabtastungen vorliegen, muss diese Methode aufgerufen werden, normalerweise zyklisch.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Input2D : HRESULT
VAR_INPUT
    pDataIn       : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as twod
imensional array (e.g.[1..channels,1..oversamples] )
    nDataInSize   : UDINT;                // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode;
    nWorkDim0     : UDINT:=0;             // It designates the first dimension in the MultiArray bei
ng processed. (e.g. 1..channels)
    nWorkDim1     : UDINT:=1;             // It designates the second dimension in the MultiArray be
ing processed.
    pStartIndex   : POINTER TO UDINT;     (* optional: default:0->
internally handled; It designates the index of the first MultiArray element to be copied.
                                             If allocated it must point to a onedimensional array of
 UDINT with so many elements as dimensions of the MultiArray.
                                             Upon successful completion of the copy, corresponding S
tartIndex is incremented by the number of copied elements. *)
    nOptionPars   : DWORD;                // option mask
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;                 // TRUE every time when outgoing MultiArray was calculated
 and sent to transfer tray.
    bError        : BOOL;                 // TRUE if an error occurs.
    hrErrorCode   : HRESULT;              // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataIn`: Der Datenpuffer muss die Daten von allen Kanälen enthalten.
- `eElementType`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode [▶ 292]. Der Elementtyp des spezifizierten MultiArray Puffers (Initialisierungsparameter) muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.

- `nWorkDim0`: Definiert die Dimension, welche der Anzahl Kanäle entspricht. Im Allgemeinen ist das MultiArray auch zweidimensional und `nWorkDim0 := 0`, aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann jedoch nicht verarbeitet werden dürfen.
  Z.B.: Wenn der erste Index des angegebenen Datenpuffers für die Kanäle steht, aber die zweite Dimension des MultiArray die Kanäle aufzählt, dann setzen Sie `nWorkDim0:=1`.
- `nWorkDim1`: Definiert die Dimension, in welcher die Daten angesammelt werden. Im Allgemeinen ist das MultiArray auch zweidimensional und `nWorkDim1 := 1`, aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann aber nicht verarbeitet werden dürfen.
  Z.B.: Wenn der zweite Index des angegebenen Datenpuffers für die gesammelten Daten steht, aber die erste Dimension des MultiArray die Daten sammelt, dann setzen Sie `nWorkDim1 := 0`.
- `pStartIndex`: Dies ist ein optionaler Parameter, der dann nützlich ist, wenn das MultiArray mehr als zwei Dimensionen erhalten hat. Gibt den Index des ersten zu kopierenden MultiArray Elements an. Wenn zugewiesen, dann muss es auf ein eindimensionales Array von `UDINT` zeigen, das so viele Elemente hat, wie das MultiArray Dimensionen hat. Im Anschluss an einen erfolgreichen Kopiervorgang wird der entsprechende Start-Index um die Anzahl der kopierten Elemente inkrementiert.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.
- `hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ `HRESULT` angezeigt. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.

Input2DStd():

Schreibt Daten von einem externen zweidimensionalen Datenpuffer in ein MultiArray. Jedes Mal wenn neue Eingangsabtastungen vorliegen, muss diese Methode aufgerufen werden, normalerweise zyklisch.

Diese Methode verwendet Standardwerte für die Parameter `nWorkDim0 := 0`, `nWorkDim1 := 1` und `pStartIndex := 0`.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Input2DStd : HRESULT
VAR_INPUT
    pDataIn        : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as twod
imensional array (e.g.[1..channels,1..oversamples] )
    nDataInSize    : UDINT;                // size of data buffer in bytes
    eElementType   : E_MA_ElementTypeCode;
    nOptionPars    : DWORD;                // option mask
END_VAR
VAR_OUTPUT
    bNewResult     : BOOL;                 // TRUE every time when outgoing MultiArray was calculated
 and sent to transfer tray.
    bError         : BOOL;                 // TRUE if an error occurs.
    hrErrorCode    : HRESULT;              // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataIn`: Der Datenpuffer muss die Daten von allen Kanälen enthalten.
- `eElementType`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode [▶ 292]. Der Elementtyp des spezifizierten MultiArray Puffers (Initialisierungsparameter) muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.
- `hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ `HRESULT` angezeigt. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.

Input3D():

Schreibt Daten von einem externen dreidimensionalen Datenpuffer in ein MultiArray. Jedes Mal wenn neue Eingangsabtastungen vorliegen, muss diese Methode aufgerufen werden, normalerweise zyklisch.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Input2D : HRESULT
VAR_INPUT
    pDataIn       : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as twod
imensional array (e.g.[1..channels,1..oversamples] )
    nDataInSize   : UDINT;                // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode;
    nWorkDim0     : UDINT:=0;             // It designates the first dimension in the MultiArray bei
ng processed. (e.g. 1..channels)
    nWorkDim1     : UDINT:=1;             // It designates the second dimension in the MultiArray be
ing processed.
    nWorkDim2     : UDINT:=2;             // It designates the third dimension in the MultiArray bei
ng processed.
    pStartIndex   : POINTER TO UDINT;     (* optional: default:0->
internally handled; It designates the index of the first MultiArray element to be copied.
                                            If allocated it must point to a onedimensional array of
 UDINT with so many elements as dimensions of the MultiArray.
                                            Upon successful completion of the copy, corresponding S
tartIndex is incremented by the number of copied elements. *)
    nOptionPars   : DWORD;                // option mask
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;                 // TRUE every time when outgoing MultiArray was calculated
 and sent to transfer tray.
    bError        : BOOL;                 // TRUE if an error occurs.
    hrErrorCode   : HRESULT;              // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataIn`: Der Datenpuffer muss die Daten von allen Kanälen enthalten.
- `eElementType`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode [▶ 292]. Der Elementtyp des spezifizierten MultiArray Puffers (Initialisierungsparameter) muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.
- `nWorkDim0`: Definiert die Dimension, welche der Anzahl Kanäle entspricht. Im Allgemeinen ist das MultiArray auch zweidimensional und `nWorkDim0 := 0`, aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann jedoch nicht verarbeitet werden dürfen.
  Z.B.: Wenn der erste Index des angegebenen Datenpuffers für die Kanäle steht, aber die zweite Dimension des MultiArray die Kanäle aufzählt, dann setzen Sie `nWorkDim0 := 1`.
- `nWorkDim1`: Definiert die Dimension, welche der Anzahl Unterkanäle entspricht. Im Allgemeinen ist das MultiArray auch dreidimensional und `nWorkDim0 := 0` und `nWorkDim1 := 1`, aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann jedoch nicht verarbeitet werden dürfen.
  Z.B.: Wenn der zweite Index des angegebenen Datenpuffers für die Unterkanäle steht, aber die dritte Dimension des MultiArray die Unterkanäle aufzählt, dann setzen Sie `nWorkDim1 := 2`.
- `nWorkDim2`: Definiert die Dimension, in welcher die Daten angesammelt werden. Im Allgemeinen ist das MultiArray auch zweidimensional und `nWorkDim1 := 1`, aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann aber nicht verarbeitet werden dürfen.
  Z.B.: Wenn der zweite Index des angegebenen Datenpuffers für die gesammelten Daten steht, aber die erste Dimension des MultiArray die Daten sammelt, dann setzen Sie `nWorkDim1 := 0`.
- `pStartIndex`: Dies ist ein optionaler Parameter, der dann nützlich ist, wenn das MultiArray mehr als zwei Dimensionen erhalten hat. Gibt den Index des ersten zu kopierenden MultiArray Elements an. Wenn zugewiesen, dann muss es auf ein eindimensionales Array von `UDINT` zeigen, das so viele Elemente hat, wie das MultiArray Dimensionen hat. Im Anschluss an einen erfolgreichen Kopiervorgang wird der entsprechende Start-Index um die Anzahl der kopierten Elemente inkrementiert.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.

`hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ `HRESULT` angezeigt. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.

Input3DStd():

Schreibt Daten von einem externen dreidimensionalen Datenpuffer in ein MultiArray. Jedes Mal wenn neue Eingangsabtastungen vorliegen, muss diese Methode aufgerufen werden, normalerweise zyklisch.

Diese Methode verwendet Standardwerte für die Parameter `nWorkDim0 := 0`, `nWorkDim1 := 1`, `nWorkDim2 := 2` und `pStartIndex := 0`.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Input2DStd : HRESULT
VAR_INPUT
    pDataIn       : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as twod
imensional array (e.g.[1..channels,1..oversamples] )
    nDataInSize   : UDINT;                // size of data buffer in bytes
    eElementType  : E_MA_ElementTypeCode;
    nOptionPars   : DWORD;                // option mask
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;                 // TRUE every time when outgoing MultiArray was calculated
 and sent to transfer tray.
    bError        : BOOL;                      // TRUE if an error occurs.
    hrErrorCode   : HRESULT;              // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataIn`: Der Datenpuffer muss die Daten von allen Kanälen und Unterkanälen enthalten.
- `eElementType`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode [▶ 292]. Der Elementtyp des spezifizierten MultiArray Puffers (Initialisierungsparameter) muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.

`hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ `HRESULT` angezeigt. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_MA_MultiArray_InitPars;       // init parameter
    nOwnID         : UDINT;                           // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Funktionsbausteinspezifische Struktur der Initialisierungsparameter vom Typ ST_MA_MultiArray_InitPars [▶ 322]. MultiArray Puffer werden für die Ergebnispuffer spezifiziert. Diese Parameter müssen mit der obigen Definition der Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, um den aktuellen Ausgangspuffer (MultiArray) erneut von Beginn an befüllen zu können. Sollten beim Befüllen externe Indizes (`pStartIndex` Parameter) genutzt werden, so müssen diese explizit zurückgesetzt werden.

- **Rückgabewert** : Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

ResetAnalysisChain():

Durch Aufruf dieser Methode wird automatisch ein Reset aller Algorithmen der vollständigen Analysekette durchgeführt. Intern wird jeweils ein ResetData() vor der Übernahme des neuen Datensatzes ausgeführt. Folglich impliziert dies auch ein ResetData() am FB_CMA_Source.

Soll die Analysekette nur für einen bestimmten Zeitraum aktiv sein, so bietet diese Methode die Möglichkeit alle Algorithmen vor der nächsten Ausführung zurückzusetzen.

Es können Fehler beim Aufruf einer Input Methode auftreten und Unterbrechungen der Zeitreihensammlung verursachen. Falls die folgenden Algorithmen der Analysekette Spektren berechnen, so kann im Falle eines Fehlers beim Aufruf einer Input Methode die ResetAnalysisChain() Methode aufgerufen werden. Denn es ist nicht möglich korrekte Spektren anhand zerstückelter Zeitreihen zu berechnen.

- **Rückgabewert** : Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes erläutert.

```
METHOD ResetAnalysisChain : HRESULT
VAR_INPUT
END_VAR
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

5.1.39 FB_CMA_SourcePaired

Dieser Funktionsbaustein schreibt Daten aus externen SPS Datenpuffern in ein Paar von MultiArray Puffern.

Er häuft ununterbrochen Eingangsdaten an, bis die Größen der MultiArrays erreicht ist. Wenn diese komplett gefüllt sind, wird es an die Zielanalyse-ID übergeben.

Synchronisation der Daten

Die Zuordnung der Daten des ersten Streams zum zweiten ist direkt, d.h. der erste Wert aus Stream A steht in Beziehung zum ersten Wert aus Stream B. Messdaten, z.B. Positions- und Vibrationswerte müssen vorab synchronisiert werden.

Eine Instanz von FB_CMA_Source darf nicht als Ziel für irgendeinen anderen Analysebaustein verwendet werden. Sie bietet ausschließlich Quellfunktionalitäten.

Eine Zeitreihensammlung kann im Falle eines Fehlerauftritts unterbrochen werden. Verlorene Signaldaten können zu einem unerwarteten Ergebnis der Analysekette führen, je nach Konfiguration der Algorithmen.

Ein- und Ausgänge

Die Ausgangspuffer entsprechen folgender Definition (Shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

MultiArray im	Elementtyp	Dimensionen	Dimensionsgrößen
output stream A	eTypeCode	`nDims`	`aDimSizes`
output stream B	eTypeCode	`nDims`	`aDimSizes`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitParsA      : ST_MA_MultiArray_InitPars;       // init parameters for first MultiArray
    stInitParsB      : ST_MA_MultiArray_InitPars;       // init parameters for second MultiArray
    nOwnID           : UDINT;                           // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                      // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#40US;             // timeout checking off during access to int
er-task FIFOs
END_VAR
```

- `stInitParsA`: Funktionsbausteinspezifische Struktur der Initialisierungsparameter vom Typ ST_MA_MultiArray_InitPars [▶ 322]. MultiArray Puffer werden für die Ergebnispuffer spezifiziert. Diese Parameter müssen mit der obigen Definition der Ausgangspuffer übereinstimmen.
- `stInitParsB`: Funktionsbausteinspezifische Struktur der Initialisierungsparameter vom Typ ST_MA_MultiArray_InitPars [▶ 322]. MultiArray Puffer werden für die Ergebnispuffer spezifiziert. Diese Parameter müssen mit der obigen Definition der Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Eigenschaften

Folgende Eigenschaften können optional verwendet werden, um den Zeitstempel der Eingangswerte zugehörig zum nächsten Input-Aufruf zu spezifizieren. Werden die Eigenschaften nicht gesetzt, so wird intern automatisch ein aktueller Zeitstempel und eine typische Zeitdifferenz zwischen zwei Eingangswerten ermittelt.
Am FB_CMA_Sink [▶ 238] werden Zeitspanne und Zeitstempel der Eingangswerte der Analysekette zugeordnet zum Ergebnis der Analyse bereitgestellt.

Name	Typ	Zugriff	Beschreibung
nTimestamp	ULINT [T_DCTIME64; 1ns steps since 1.1.2000]	Set	Angabe des Zeitstempels zugehörig zum ältesten Wert, welcher mit dem nächsten Input-Aufruf

Name	Typ	Zugriff	Beschreibung
			übergeben wird. (Beispiel: StartTimeNextLatch -CycleTime)
tSamplePeriod	LTIME	Set	Angabe der Zeitdifferenz zwischen zwei Eingangswerten. (Beispiel: CycleTime / cOversamples). Diese Angabe ist nur einmalig nötig, da es sich um einen konstanten Wert handelt.

Methoden

Input1D():

Schreibt Daten von externen eindimensionalen Datenpuffern in ein Paar von MultiArrays. Jedes Mal wenn neue Eingangsabtastungen vorliegen, muss diese Methode aufgerufen werden, normalerweise zyklisch.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Input1D : HRESULT
VAR_INPUT
    pDataInA      : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeA  : UDINT;                // size of data buffer in bytes
    pDataInB      : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeB  : UDINT;                // size of data buffer in bytes
    eElementTypeA : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    eElementTypeB : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    nWorkDimA     : UDINT;                // It designates the dimension in the first multi array be
ing processed.
    nWorkDimB     : UDINT;                // It designates the dimension in the second multi array b
eing processed.
    pStartIndexA  : POINTER TO UDINT;    (* optional: default:0-
>internally handled; It designates the index of the first MultiArray element to be copied.
                          If allocated it must point to a onedimensional array of UDINT with so many
elements as dimensions of the MultiArray.
                          Upon successful completion of the copy, corresponding StartIndex is increme
nted by the number of copied elements. *)
    pStartIndexB  : POINTER TO UDINT;    // see pStartIndexA
    nOptionPars   : DWORD;               // option mask
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;                // TRUE every time when outgoing MultiArray was calculated
and sent to transfer tray.
    bError        : BOOL;                // TRUE if an error occurs.
    hrErrorCode   : HRESULT;             // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataInA`, `pDataInB`: Der Datenpuffer muss die Daten von allen Kanälen enthalten.
- `eElementTypeA`, `eElementTypeB`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode [▶ 292]. Der Elementtyp des spezifizierten MultiArray Puffers (Initialisierungsparameter) muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.
- `nWorkDimA`, `nWorkDimB`: Definiert die Dimension, in welcher die Daten angesammelt werden. Im Allgemeinen ist das MultiArray auch eindimensional und `nWorkDimA := 0`, bzw. `nWorkDimB := 0` aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann jedoch nicht verarbeitet werden dürfen.
- `pStartIndexA`, `pStartIndexB`: Dies ist ein optionaler Parameter, der dann nützlich ist, wenn das MultiArray mehr als eine Dimension erhalten hat. Gibt den Index des ersten zu kopierenden MultiArray Elements an. Wenn zugewiesen, dann muss es auf ein eindimensionales Array von `UDINT` zeigen, das

so viele Elemente hat, wie das MultiArray Dimensionen hat. Im Anschluss an einen erfolgreichen Kopiervorgang wird der entsprechende Start-Index um die Anzahl der kopierten Elemente inkrementiert.

- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.
- `hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ HRESULT angezeigt. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.

Input1DStd():

Schreibt Daten von externen eindimensionalen Datenpuffern in ein Paar von MultiArrays. Jedes Mal wenn neue Eingangsabtastungen vorliegen, muss diese Methode aufgerufen werden, normalerweise zyklisch.

Diese Methode verwendet Standardwerte für die Parameter `nWorkDimA := 0`, `nWorkDimB := 0` und `pStartIndexA := 0`, `pStartIndexB := 0`.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Input1DStd : HRESULT
VAR_INPUT
    pDataInA      : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeA  : UDINT;                // size of data buffer in bytes
    pDataInB      : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeB  : UDINT;                // size of data buffer in bytes
    eElementTypeA : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    eElementTypeB : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    nOptionPars   : DWORD;               // option mask
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;                // TRUE every time when outgoing MultiArray was calculated
and sent to transfer tray.
    bError        : BOOL;                // TRUE if an error occurs.
    hrErrorCode   : HRESULT;             // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataInA`, `pDataInB`: Der Datenpuffer muss die Daten von allen Kanälen enthalten.
- `eElementTypeA`, `eElementTypeB`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode [▶ 292]. Der Elementtyp des spezifizierten MultiArray Puffers (Initialisierungsparameter) muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.
- `hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ HRESULT angezeigt. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.

Input2D():

Schreibt Daten von externen zweidimensionalen Datenpuffern in ein Paar von MultiArrays. Jedes Mal wenn neue Eingangsabtastungen vorliegen, muss diese Methode aufgerufen werden, normalerweise zyklisch.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Input2D : HRESULT
VAR_INPUT
    pDataInA      : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeA  : UDINT;                // size of data buffer in bytes
    pDataInB      : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeB  : UDINT;                // size of data buffer in bytes
    eElementTypeA : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    eElementTypeB : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    nWorkDimA0    : UDINT;                // It designates the first dimension in the first multi ar
ray being processed.
    nWorkDimA1    : UDINT;                // It designates the second dimension in the first multi a
rray being processed.
    nWorkDimB0    : UDINT;                // It designates the first dimension in the second multi a
rray being processed.
```

```
    nWorkDimB1     : UDINT;                // It designates the second dimension in the second multi
array being processed.
    pStartIndexA   : POINTER TO UDINT;    (* optional: default:0-
>internally handled; It designates the index of the first MultiArray element to be copied.
                           If allocated it must point to a onedimensional array of UDINT with so many
elements as dimensions of the MultiArray.
                           Upon successful completion of the copy, corresponding StartIndex is increme
nted by the number of copied elements. *)
    pStartIndexB   : POINTER TO UDINT;    // see pStartIndexA
    nOptionPars    : DWORD;               // option mask
END_VAR
VAR_OUTPUT
    bNewResult     : BOOL;                // TRUE every time when outgoing MultiArray was calculated
and sent to transfer tray.
    bError         : BOOL;                // TRUE if an error occurs.
    hrErrorCode    : HRESULT;             // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataInA`, `pDataInB`: Der Datenpuffer muss die Daten von allen Kanälen enthalten.
- `eElementTypeA`, `eElementTypeB`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode [▶ 292]. Der Elementtyp des spezifizierten MultiArray Puffers (Initialisierungsparameter) muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.
- `nWorkDimA0`, `nWorkDimB0`: Definiert die Dimension, welche der Anzahl Kanäle entspricht. Im Allgemeinen ist das MultiArray auch zweidimensional und `nWorkDimA0 := 0`, bzw. `nWorkDimB0 := 0`, aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann jedoch nicht verarbeitet werden dürfen.
  Z.B.: Wenn der erste Index des angegebenen Datenpuffers für die Kanäle steht, aber die zweite Dimension des MultiArray die Kanäle aufzählt, dann setzen Sie `nWorkDimA0 := 1`, bzw. `nWorkDimB0 := 1`.
- `nWorkDimA1`, `nWorkDimB1`: Definiert die Dimension, in welcher die Daten angesammelt werden. Im Allgemeinen ist das MultiArray auch zweidimensional und `nWorkDimA1 := 1`, bzw. `nWorkDimB1 := 1`, aber das MultiArray kann auch zusätzliche Dimensionen haben, die dann aber nicht verarbeitet werden dürfen.
  Z.B.: Wenn der zweite Index des angegebenen Datenpuffers für die gesammelten Daten steht, aber die erste Dimension des MultiArray die Daten sammelt, dann setzen Sie `nWorkDimB1 := 0`, bzw. `nWorkDimB1 := 0`.
- `pStartIndexA`, `pStartIndexB`: Dies ist ein optionaler Parameter, der dann nützlich ist, wenn das MultiArray mehr als eine Dimension erhalten hat. Gibt den Index des ersten zu kopierenden MultiArray Elements an. Wenn zugewiesen, dann muss es auf ein eindimensionales Array von `UDINT` zeigen, das so viele Elemente hat, wie das MultiArray Dimensionen hat. Im Anschluss an einen erfolgreichen Kopiervorgang wird der entsprechende Start-Index um die Anzahl der kopierten Elemente inkrementiert.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.
- `hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ HRESULT angezeigt. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.

Input2DStd():

Schreibt Daten von externen zweidimensionalen Datenpuffern in ein Paar von MultiArrays. Jedes Mal wenn neue Eingangsabtastungen vorliegen, muss diese Methode aufgerufen werden, normalerweise zyklisch.

Diese Methode verwendet Standardwerte für die Parameter `nWorkDimA0 := 0`, `nWorkDimA1 := 1`, `nWorkDimB0 := 0`, `nWorkDimB1 := 1` und `pStartIndexA := 0`, `pStartIndexB := 0`.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Input2D : HRESULT
VAR_INPUT
    pDataInA       : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeA   : UDINT;                // size of data buffer in bytes
    pDataInB       : POINTER TO BYTE;      // address of data buffer (e.g. oversampling data) as one-
dimensional array
    nDataInSizeB   : UDINT;                // size of data buffer in bytes
    eElementTypeA  : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
```

```
    eElementTypeB : E_MA_ElementTypeCode; // valid types: LREAL, INT32, UINT64, LCOMPLEX
    nOptionPars   : DWORD;                // option mask
END_VAR
VAR_OUTPUT
    bNewResult    : BOOL;                 // TRUE every time when outgoing MultiArray was calculated
and sent to transfer tray.
    bError        : BOOL;                 // TRUE if an error occurs.
    hrErrorCode   : HRESULT;              // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `pDataInA`, `pDataInB`: Der Datenpuffer muss die Daten von allen Kanälen enthalten.
- `eElementTypeA`, `eElementTypeB`: Dieser Eingang ist vom Typ E_MA_ElementTypeCode [▶ 292]. Der Elementtyp des spezifizierten MultiArray Puffers (Initialisierungsparameter) muss mit dem Elementtyp des angegebenen externen Datenpuffers übereinstimmen.
- `bError`: Dieser Ausgang ist `TRUE`, wenn ein Fehler auftritt.
- `hrErrorCode`: Tritt ein Fehler auf, dann wird ein beschreibender Fehlercode vom Typ HRESULT angezeigt. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] definiert. Dieser Ausgang ist gleich dem Rückgabewert der Methode.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitParsA    : ST_MA_MultiArray_InitPars;       // init parameters for first MultiArray
    stInitParsB    : ST_MA_MultiArray_InitPars;       // init parameters for second MultiArray
    nOwnID         : UDINT;                           // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT; // IDs of destinations for output
    nResultBuffers : UDINT := 4;                      // number of MultiArrays which should be initi
alized for results (0 for no initialization)
END_VAR
```

- `stInitParsA`: Funktionsbausteinspezifische Struktur der Initialisierungsparameter vom Typ ST_MA_MultiArray_InitPars [▶ 322]. MultiArray Puffer werden für die Ergebnispuffer spezifiziert. Diese Parameter müssen mit der obigen Definition der Ausgangspuffer übereinstimmen.
- `stInitParsB`: Funktionsbausteinspezifische Struktur der Initialisierungsparameter vom Typ ST_MA_MultiArray_InitPars [▶ 322]. MultiArray Puffer werden für die Ergebnispuffer spezifiziert. Diese Parameter müssen mit der obigen Definition der Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, um die aktuellen Ausgangspuffer (MultiArray) erneut von Beginn an befüllen zu können. Sollten beim Befüllen externe Indizes (`pStartIndex` Parameter) genutzt werden, so müssen diese explizit zurückgesetzt werden.

- **Rückgabewert** : Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

5.1.40 FB_CMA_VibrationAssessment

Schwingungsbeurteilung reell-wertiger Eingangssignale.

Der Baustein FB_CMA_VibrationAssessment führt eine Schwinungsbeurteilung reell-wertiger Eingangssignale in Anlehnung an ISO 10816-3 durch. Diese ist im Abschnitt Anwendungskonzepte näher erläutert, siehe Schwingungsbeurteilung [▶ 35]. Der Baustein kombiniert die Berechnung integrierter RMS Werte auf konfigurierbaren Frequenzbändern und deren Klassifikation für ein- und mehrkanalige Eingangsdaten.

Das Ergebnis ist ein eindimensionales Array, welches für jedes Frequenzband drei Werte bereithält, die höchste berechnete Klassifikation (im Bereich `-1..nMaxClasses`) sowie die zugehörige Integrationsordnung (im Bereich `0..nOrder`) und dem Kanal (im Bereich `1..nChannels`). Es empfiehlt sich, das Ergebnis der Klassifikation am Sink in einem zweidimensionalen Array der Form `aResult : ARRAY[1..nMaxBands] OF ARRAY [1..3] OF DINT` zu speichern. Zu jedem Band liegen die Daten dann wie folgt vor: `[{class},{order},{channel}]`.

Die erhaltenen Daten können verwendet werden um eine Aussage/Bewertung bezüglich des Maschinenzustandes auf Basis von Vibrationsmessungen zu treffen. Die mögliche Konfiguration mehrerer Frequenzbänder erleichtert die Bewertung; am Beispiel der ISO 10816-3 kann der Zustand für verschiedene Drehzahlen der Maschine simultan bestimmt werden.

Fensterlänge beachten

Der Wert von `nWindowLength` muss kleiner oder gleich dem Wert von `nFFT_Length` sein. Die Länge der FFT kann sich an der benötigten Frequenzauflösung orientieren. Typischerweise wird oft ein Wert von ca. 4/5 der FFT-Länge als Fensterlänge verwendet.

Wenn `nFFT_Length` größer ist als `nWindowLength`, wird die Frequenzauflösung der FFT (und damit auch die Länge des Vektors der Rückgabewerte) vergrößert. Die Differenz der Länge wird vor der Fouriertransformation mit Nullen aufgefüllt. Dies kann sinnvoll sein, um eine höhere Frequenzauflösung zu erreichen oder, z.B. bei der Berechnung mit Rücktransformation in den Zeitbereich, zirkuläres Aliasing zu vermeiden. Das Ergebnis enthält trotz der höheren Frequenzauflösung allerdings nicht mehr Information.

Gedächtniseigenschaften

Aufgrund der Verwendung der Welch-Methode wird jeweils der aktuelle Eingangsdatenpuffer zusammen mit den zuletzt übergebenen Puffern zur Berechnung genutzt. Die Anzahl der einfließenden Puffer ist abhängig von der gewählten Überlappung (`nOverlap`).
Die Frequenzanalyse berücksichtigt Sprünge in der Zeitreihe. Um ein korrektes Ergebnis zu erzielen, müssen sich deswegen die verwendeten Eingangsdatenpuffer lückenlos und ohne Sprünge aneinanderreihen.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Schwingungsbeurteilung nach ISO 10816-3 (kompakt) [▶ 349].

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1, nWindowLength-nOverlap`	`DINT, 1, nMaxBands*3`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2, nChannels x nWindowLength-nOverlap`	`DINT, 1, nMaxBands*3`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_VibrationAssessment_InitPars;  // init parameter
    nOwnID           : UDINT;                               // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;     // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                          // number of MultiArrays which should be
 initialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;                // timeout checking off during access to
 inter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_VibrationAssessment_InitPars [▶ 319]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError          : BOOL;                            // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                         // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                           // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.

- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_VibrationAssessment_InitPars;  // init parameter
    nOwnID         : UDINT;                               // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;     // IDs of destinations for output
    nResultBuffers : UDINT := 4;                          // number of MultiArrays which should be i
nitialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_VibrationAssessment_InitPars [▶ 319]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

Configure():

Mit dem Aufruf dieser Methode müssen die Klassifikationsargumente und Frequenzbänder zu Beginn konfiguriert werden. Die entsprechenden SPS Arrays müssen wie folgt definiert sein. Für eine erneute Konfiguration mit einem anderen Satz an Argumenten kann die `Configure()` Methode ebenfalls genutzt werden.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg1     : POINTER TO LREAL; // pointer to 2-dimensional
array (LREAL) of arguments for classification
    nArgSize1 : UDINT;            // size of arguments buffer in bytes
    pArg2     : POINTER TO LREAL; // pointer to 2-dimensional
array (LREAL) of arguments for frequency bands
    nArgSize2 : UDINT;            // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`. Die zwei zu konfigurierenden Parameter je Frequenzband sind `[fLowerFrequencyLimit, fUpperFrequencyLimit]` (Eingangspuffer 2).

Varianten	Eingangspuffer 1 (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Eingangspuffer 2 (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle (`nChannels >= 1`)	`LREAL, 2,` `nOrder+1 x nMaxClasses`	`LREAL, 2,` `nMaxBands x 2`

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_IntegratedRMS [▶ 167] berechnet (optional) integrierte RMS Werte für ein- und mehrkanalige reell-wertige Zeitreihen.

Der Baustein FB_CMA_WatchUpperThresholds [▶ 262] führt eine konfigurierbare Schwellwertüberwachung von mehrkanaligen Daten durch.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.41 FB_CMA_WatchUpperThresholds

Konfigurierbare Schwellwertüberwachung von mehrkanaligen Daten

Der Baustein ordnet, ähnlich wie der Baustein FB_CMA_DiscreteClassification [▶ 121] die einzelnen Kanäle eines mehrkanaligen Signals anhand konfigurierbarer Schwellwerte einer festen Zahl von konfigurierbaren diskreten Kategorien zu. Nach der Konfiguration berechnet der Baustein für jeden Eingangsvektor ein eindimensionales Array mit genau zwei Werten. Der Typ der beiden Elemente ist eine vorzeichenbehaftete 32-Bit Integer-Zahl. Der erste Wert des Ergebnisses bezeichnet die Nummer der höchsten ermittelten Kategorie und der zweite Wert die Nummer des Kanals mit der höchsten Kategorie. Dabei beginnt die Nummerierung in beiden Fällen mit dem Wert Null. Entspricht kein Eingangswert eines Kanals dem jeweiligen Schwellwert für die niedrigste Kategorie, ist der Wert -1 das Ergebnis. Ist ein Eingangswert genau gleich dem Schwellwert einer Kategorie, wird er als zu dieser Kategorie gehörend gewertet. Wird mehreren Kanälen die höchste Kategorie zugeordnet, so wird die Nummer des Kanals mit der niedrigeren Nummer zurückgegeben.

Konfiguration

Der Baustein lässt sich zur Laufzeit konfigurieren, indem für jeden Kanal und jede Schwellwert-Kategorie die Größe des Schwellwerts angegeben wird.

Gedächtniseigenschaften

Je nach Konfiguration des Bausteins mit `bMemorize` wird die Nummer der höchsten Schwellwertkategorie sowie die Nummer des auslösenden Kanals so lange gespeichert, bis die Methode `ResetData()` aufgerufen wird, oder die Werte werden bei jedem Schritt neu berechnet.

NaN Vorkommnis

Bei einem Eingangswert von NaN ist das Ergebnis der Klassifikation `-2`. Am Ausgang sind keine NaN Werte zu erwarten.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle

Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nSubChannels = 0`)	`LREAL, 1, nChannels`	`DINT (32bit), 1, 2`
Mehrkanalige Variante (`nSubChannels > 0`)	`LREAL, 2, nChannels x nSubChannels`	`DINT (32bit), 2, nChannels x 2`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_WatchUpperThresholds_InitPars;  // init parameter
    nOwnID           : UDINT;                                // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;      // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                           // number of MultiArrays which should be
 initialized for results (0 for no initialization)
    tTransferTimeout: LTIME := LTIME#500US;                  // timeout checking off during access to
 inter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_WatchUpperThresholds_InitPars [▶ 321]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError          : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
```

```
    hrErrorCode     : HRESULT;                          // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                            // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult    : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError        : BOOL;       // TRUE if an error occurs.
    hrErrorCode   : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_WatchUpperThresholds_InitPars [▶ 321]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.

- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

Configure():

Mit dem Aufruf dieser Methode müssen die Klassifikationsargumente zu Beginn konfiguriert werden. Das entsprechende SPS Array muss wie folgt definiert sein. Für eine erneute Konfiguration mit einem anderen Satz an Argumenten kann die `Configure()` Methode ebenfalls genutzt werden.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : POINTER TO LREAL; // pointer to array (LREAL) of arguments
    nArgSize : UDINT;            // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle und Unterkanäle	`LREAL, 1, nMaxClasses`
Kanalspezifische Konfiguration (`nSubChannels = 0`)	`LREAL, 2, nChannels x nMaxClasses`
Unterkanalspezifische Konfiguration (`nSubChannels > 0`)	`LREAL, 2, nSubChannels x nMaxClasses`
Kanal- und unterspezifische Konfiguration (`nSubChannels > 0`)	`LREAL, 3, nChannels x nSubChannels x nMaxClasses`

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

Alternativ kann für einen automatischen Reset `bMemorize=FALSE` in der Initialisierungstruktur gesetzt werden.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_DiscreteClassification [▶ 121] klassifiziert mehrkanalige Eingangsdaten.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.42 FB_CMA_ZoomFFT

Effiziente Berechnung von Spektralwerten über ein definiertes Frequenzband

Der Baustein FB_CMA_ZoomFFT berechnet einen Ausschnitt (Frequenzband) der Fourier-Transformierten des am Baustein anliegenden reell-wertigen Eingangssignals *x*[n]. Das Ergebnis des Funktionsbausteins ist für das betrachtete Frequenzband identisch mit dem Ergebnis aus FB_CMA_RealFFT, jedoch bei entsprechend geringerem Rechenaufwand.

Das zu berechnende Frequenzband (der „Zoom“) ist definiert über eine Zentrierungsfrequenz `fCenter` f_c und einen Dezimierungsfaktor `nDecimationFactor` D. Der Dezimierungsfaktor muss eine Potenz von 2 sein (2, 4, 8, 16, …). Das Frequenzband ist dann definiert als:

$$\left[f_c - \frac{D}{2}\Delta f, f_c + \frac{D}{2}\Delta f\right]$$

Die Frequenzauflösung Δf im Frequenzband bleibt wie man sie bei einer normalen FFT über das gesamte Spektrum erwartet. Es gilt:

$$\Delta f = \frac{f_s}{N} = \frac{f_s/D}{N/D} = \frac{\tilde{f}_s}{\tilde{N}} = \Delta\tilde{f}, \qquad \tilde{f}_s = \frac{f_s}{D}, \quad \tilde{N} = \frac{N}{D}$$

Abtasttheorem: Die höchste Frequenz einer Komponente im Eingangssignal sollte maximal bei der halben Abtastrate des Eingangssignals liegen, damit Aliasing-Effekte vermieden werden.

NaN Vorkommnis

Falls der Eingangsvektor einen oder mehrere NaN (Not a Number)-Werte beinhaltet, wird der gesamte Ausgangsvektor mit NaN gefüllt. Siehe separates Kapitel für weitere Information über NaN Werte [▶ 76].

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Zoom FFT [▶ 376]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen einer der folgenden Definitionen (input / output shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nChannels = 1`)	`LREAL, 1, nFFT_Length`	`LCOMPLEX, 1, nFFT_Length/ (2*nDecimationFactor+1)`
Standardvariante (`nChannels = 1`)	`LCOMPLEX, 1, nFFT_Length`	`LCOMPLEX, 1, nFFT_Length/ (2*nDecimationFactor+1)`
Mehrkanalige Variante (`nChannels > 1`)	`LREAL, 2, nChannels x nFFT_Length`	`LCOMPLEX, 2, nChannels x nFFT_Length/ (2*nDecimationFactor+1)`
Mehrkanalige Variante (`nChannels > 1`)	`LCOMPLEX, 2, nChannels x nFFT_Length`	`LCOMPLEX, 2, nChannels x nFFT_Length/ (2*nDecimationFactor+1)`

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_ZoomFFT_InitPars;           // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs         : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_ZoomFFT_InitPars [▶ 321]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.

- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Methoden

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Configure():

Mit dem Aufruf dieser Methode kann der Parameter `fCenter` zur Laufzeit angepasst werden. Hierbei berechnet sich der FFT Index zur Frequenz `f` zu `k = f * fSampleRate / nFFT_Length`. Ist `f` kein ganzzahliges Vielfaches der Frequenzauflösung `fSampleRate / nFFT_Length` sind die Spektralanteile auf zwei aufeinander folgende Spektralwerte aufgeteilt.

Das entsprechende SPS Array muss wie folgt definiert sein. Für eine erneute Konfiguration mit einem anderen Satz an Argumenten kann die `Configure()` Methode ebenfalls genutzt werden.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Configure : HRESULT
VAR_INPUT
    pArg     : PVOID;     // pointer to array of arguments
    nArgSize : UDINT;     // size of arguments buffer in bytes
END_VAR
```

Die Eingangspuffer entsprechen einer der folgenden Definitionen (input shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Identische Konfiguration aller Kanäle	`LREAL, 1, 1`
Kanalspezifische Konfiguration (`nChannels > 1`)	`LREAL, 1, nChannels`

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_ZoomFFT_InitPars;           // init parameter
    nOwnID         : UDINT;                            // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;  // IDs of destinations for output
    nResultBuffers : UDINT := 4;                       // number of MultiArrays which should be init
ialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_ZoomFFT_InitPars [▶ 321]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_RealFFT [▶ 216] berechnet die schnelle Fouriertransformation für reell-wertige Eingangssignale.

Der Baustein ST_CM_SparseSpectrum_InitPars [▶ 317] berechnet einzelne, konfigurierbare Magnituden- und Spektralwerte.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.1.43 FB_CMA_MovingMean

Berechnet den gleitenden Mittelwert für ein- und mehrkanalige reell-wertige Zeitreihen.

Der Baustein behandelt das Eingangssignal als Zeitreihe mit ggf. mehreren unabhängigen Kanälen. Für jeden Kanal wird der gleitende (arithmetische) Mittelwert nach Gleichung

$$\overline{x} = \frac{1}{N}\sum_{n=0}^{N-1} x[n]$$

berechnet. Es können in jedem Zyklus ein feste Anzahl (`nInLength`) Samples pro Kanal zur Stichprobenmenge hinzugefügt werden (siehe Ein- und Ausgänge). Die Berechnung der Mittelwerte erfolgt hierbei stets Sample für Sample.

Gedächtniseigenschaften

Die Stichprobenmenge *N*, welche zur Berechnung des aktuellen Ergebnisses genutzt wird, erhöht sich automatisch bei jedem neuen eingehenden Datensatz, d.h. der Baustein verwendet alle Eingangswerte seit dessen Instanziierung. Das Zurücksetzen der Stichprobenmenge auf Null (löschen des internen Speichers des FBs) ist über eine `ResetData()` Methode oder, bei Verwendung der `CallEx()` Methode über die Variable `bResetData` vorgesehen.

NaN Vorkommnis

Falls für einen Kanal noch keine ausreichende Anzahl Eingangswerte für die Berechnung eines Ergebnisses vorliegt oder die Varianz Null beträgt, wird für diesen Kanal der Wert NaN (Not A Number) nach IEEE 754 zurückgegeben. Das Vorliegen dieses Fehlerwertes kann mit der Funktion LrealIsNaN() überprüft werden. Ursache dafür kann entweder sein, dass noch nicht genug Eingangsdaten übergeben wurden oder dass für

einzelne Kanäle bisher lediglich NaNs als Eingangswerte übergeben wurden. Eine Varianz von Null kann insbesondere auftreten, wenn die Zeitreihe der Werte konstant ist, etwa wenn aufgrund von Kabelbruch oder Schaltfehlern keine Sensordaten übertragen werden.

Enthält ein Satz von Eingangswerten die spezielle Konstante NaN, so wird der Statistik für diesen Kanal bei diesem Zeitschritt kein Wert hinzugefügt, d.h. sie wird als Indikator für fehlende Werte behandelt.

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Gleitender Mittelwert [▶ 383]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen folgender Definition (Shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nSubChannels = 0`)	`LREAL, 1, nInLength`	`LREAL, 1, nInLength`
Standardvariante für mehrere Datensätze (`nSubChannels = 0`)	`LREAL, 2, nChannels x nInLength`	`LREAL, 2, nChannels x nInLength`
Mehrkanalige Variante (`nSubChannels > 0`)	`LREAL, 2, nChannels x nSubChannels`	`LREAL, 2, nChannels x nSubChannels`
Mehrkanalige Variante für mehrere Datensätze (`nSubChannels > 0`)	`LREAL, 3, nChannels x nSubChannels x nInLength`	`LREAL, 3, nChannels x nSubChannels x nInLength`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorrausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_MovingMoments_InitPars;     // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;           // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MovingMoments_InitPars [▶ 309]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError          : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_MovingMoments_InitPars;       // init parameter
    nOwnID         : UDINT;                              // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers: UDINT := 4;                          // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MovingMoments_InitPars [▶ 309]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_MovingStandardDeviation [▶ 274] berechnet die gleitende Standardabweichung von Eingangswerten.

Der Baustein FB_CMA_MovingSkew [▶ 278] berechnet die gleitende Schiefe von Eingangswerten.

Der Baustein FB_CMA_MovingExcess [▶ 283] berechnet den gleitenden Exzess von Eingangswerten.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.44 FB_CMA_MovingStandardDeviation

Berechnet die gleitende Standardabweichung für ein- und mehrkanalige reell-wertige Zeitreihen.

Der Baustein behandelt das Eingangssignal als Zeitreihe mit ggf. mehreren unabhängigen Kanälen. Für jeden Kanal wird die gleitende Standardabweichung nach Gleichung

$$s = \sqrt{\frac{1}{N-1}\sum_{n=0}^{N-1}(x[n]-\overline{x})^2}$$

berechnet. Es können in jedem Zyklus ein feste Anzahl (`nInLength`) Samples pro Kanal zur Stichprobenmenge hinzugefügt werden (siehe Ein- und Ausgänge). Die Berechnung der Standardabweichung erfolgt hierbei stets Sample für Sample.

Weitere Anmerkungen

Zur Berechnung eines ersten Ergebnisses müssen zwei Werte vorliegen. Ergebnisse können unter Umständen ungenau werden, wenn die Eingangswerte viele Größenordnungen auseinanderliegen.

Gedächtniseigenschaften

Die Stichprobenmenge *N*, welche zur Berechnung des aktuellen Ergebnisses genutzt wird, erhöht sich automatisch bei jedem neuen eingehenden Datensatz, d.h. der Baustein verwendet alle Eingangswerte seit dessen Instanziierung. Das Zurücksetzen der Stichprobenmenge auf Null (löschen des internen Speichers des FBs) ist über eine `ResetData()` Methode oder, bei Verwendung der `CallEx()` Methode über die Variable `bResetData` vorgesehen.

NaN Vorkommnis

Falls für einen Kanal noch keine ausreichende Anzahl Eingangswerte für die Berechnung eines Ergebnisses vorliegt oder die Varianz Null beträgt, wird für diesen Kanal der Wert NaN (Not A Number) nach IEEE 754 zurückgegeben. Das Vorliegen dieses Fehlerwertes kann mit der Funktion LrealIsNaN() überprüft werden. Ursache dafür kann entweder sein, dass noch nicht genug Eingangsdaten übergeben wurden oder dass für einzelne Kanäle bisher lediglich NaNs als Eingangswerte übergeben wurden. Eine Varianz von Null kann insbesondere auftreten, wenn die Zeitreihe der Werte konstant ist, etwa wenn aufgrund von Kabelbruch oder Schaltfehlern keine Sensordaten übertragen werden.

Enthält ein Satz von Eingangswerten die spezielle Konstante NaN, so wird der Statistik für diesen Kanal bei diesem Zeitschritt kein Wert hinzugefügt, d.h. sie wird als Indikator für fehlende Werte behandelt.

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Gleitender Mittelwert [▶ 383]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen folgender Definition (Shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nSubChannels = 0`)	`LREAL, 1, nInLength`	`LREAL, 1, nInLength`

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante für mehrere Datensätze (`nSubChannels = 0`)	`LREAL, 2, nChannels x nInLength`	`LREAL, 2, nChannels x nInLength`
Mehrkanalige Variante (`nSubChannels > 0`)	`LREAL, 2, nChannels x nSubChannels`	`LREAL, 2, nChannels x nSubChannels`
Mehrkanalige Variante für mehrere Datensätze (`nSubChannels > 0`)	`LREAL, 3, nChannels x nSubChannels x nInLength`	`LREAL, 3, nChannels x nSubChannels x nInLength`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorrausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_MovingMoments_InitPars;     // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;           // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MovingMoments_InitPars [▶ 309]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                          // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                       // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage; // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                         // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult    : BOOL;        // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError        : BOOL;        // TRUE if an error occurs.
    hrErrorCode   : HRESULT;     // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_MovingMoments_InitPars;       // init parameter
    nOwnID         : UDINT;                              // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers: UDINT := 4;                          // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MovingMoments_InitPars [▶ 309]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_MovingMean [▶ 270] berechnet den gleitenden Mittelwert von Eingangswerten.

Der Baustein FB_CMA_MovingSkew [▶ 278] berechnet die gleitende Schiefe von Eingangswerten.

Der Baustein FB_CMA_MovingExcess [▶ 283] berechnet den gleitenden Exzess von Eingangswerten.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.45 FB_CMA_MovingSkew

Berechnet die gleitende Schiefe für ein- und mehrkanalige reell-wertige Zeitreihen.

Der Baustein behandelt das Eingangssignal als Zeitreihe mit ggf. mehreren unabhängigen Kanälen. Für jeden Kanal wird die empirische Schiefe nach Gleichung

$$v = \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{x[n]-\overline{x}}{s}\right)^3$$

berechnet, wobei *s* die gleitende Standardabweichung ist. Die Schiefe quantifiziert die Asymmetrie einer Stichprobe. Sie bietet eine zum Exzess ergänzende Möglichkeit zur Beurteilung von Stößen (z.B. durch Überrollen von lokalen Schäden im Wälzlager) in einem Vibrationssignal. Die Schiefe ist als Kriterium robuster zu berechnen als die Kurtosis/der Exzess, jedoch sind Verteilungen von Signalen aufgrund von lokalen Schädigungen nicht immer asymmetrisch.

Es können in jedem Zyklus ein feste Anzahl (`nInLength`) Samples pro Kanal zur Stichprobenmenge hinzugefügt werden (siehe Ein- und Ausgänge). Die Berechnung der Mittelwerte erfolgt hierbei stets Sample für Sample.

Weitere Anmerkungen

Zur Berechnung eines ersten Ergebnisses müssen 3 Werte vorliegen. Des Weiteren darf die Standardabweichung nicht Null sein. Ergebnisse können unter Umständen ungenau werden, wenn die Eingangswerte viele Größenordnungen auseinanderliegen.

Gedächtniseigenschaften

Die Stichprobenmenge *N*, welche zur Berechnung des aktuellen Ergebnisses genutzt wird, erhöht sich automatisch bei jedem neuen eingehenden Datensatz, d.h. der Baustein verwendet alle Eingangswerte seit dessen Instanziierung. Das Zurücksetzen der Stichprobenmenge auf Null (löschen des internen Speichers des FBs) ist über eine `ResetData()` Methode oder, bei Verwendung der `CallEx()` Methode über die Variable `bResetData` vorgesehen.

NaN Vorkommnis

Falls für einen Kanal noch keine ausreichende Anzahl Eingangswerte für die Berechnung eines Ergebnisses vorliegt oder die Varianz Null beträgt, wird für diesen Kanal der Wert NaN (Not A Number) nach IEEE 754 zurückgegeben. Das Vorliegen dieses Fehlerwertes kann mit der Funktion LrealIsNaN() überprüft werden. Ursache dafür kann entweder sein, dass noch nicht genug Eingangsdaten übergeben wurden oder dass für einzelne Kanäle bisher lediglich NaNs als Eingangswerte übergeben wurden. Eine Varianz von Null kann insbesondere auftreten, wenn die Zeitreihe der Werte konstant ist, etwa wenn aufgrund von Kabelbruch oder Schaltfehlern keine Sensordaten übertragen werden.

Enthält ein Satz von Eingangswerten die spezielle Konstante NaN, so wird der Statistik für diesen Kanal bei diesem Zeitschritt kein Wert hinzugefügt, d.h. sie wird als Indikator für fehlende Werte behandelt.

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Gleitender Mittelwert [▶ 383]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen folgender Definition (Shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nSubChannels = 0`)	`LREAL, 1,` `nInLength`	`LREAL, 1,` `nInLength`
Standardvariante für mehrere Datensätze (`nSubChannels = 0`)	`LREAL, 2,` `nChannels` x `nInLength`	`LREAL, 2,` `nChannels x nInLength`
Mehrkanalige Variante (`nSubChannels > 0`)	`LREAL, 2,` `nChannels x nSubChannels`	`LREAL, 2,` `nChannels x nSubChannels`
Mehrkanalige Variante für mehrere Datensätze (`nSubChannels > 0`)	`LREAL, 3,` `nChannels x nSubChannels x nInLength`	`LREAL, 3,` `nChannels x nSubChannels x nInLength`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorrausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars       : ST_CM_MovingMoments_InitPars;     // init parameter
    nOwnID           : UDINT;                            // ID for this FB instance
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;           // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MovingMoments_InitPars [▶ 309]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError          : BOOL;                             // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode     : HRESULT;                          // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults     : ULINT;                            // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult    : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError        : BOOL;       // TRUE if an error occurs.
    hrErrorCode   : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_MovingMoments_InitPars;       // init parameter
    nOwnID         : UDINT;                              // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers: UDINT := 4;                          // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MovingMoments_InitPars [▶ 309]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.

- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_MovingMean [▶ 270] berechnet den gleitenden Mittelwert von Eingangswerten.

Der Baustein FB_CMA_MovingStandardDeviation [▶ 274] berechnet die gleitende Standardabweichung von Eingangswerten.

Der Baustein FB_CMA_MovingExcess [▶ 283] berechnet den gleitenden Exzess von Eingangswerten.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.1.46 FB_CMA_MovingExcess

Berechnet den gleitenden Exzess für ein- und mehrkanalige reell-wertige Zeitreihen.

Der Baustein behandelt das Eingangssignal als Zeitreihe mit ggf. mehreren unabhängigen Kanälen. Für jeden Kanal wird der empirische Exzess nach Gleichung

$$E = \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{x[n]-\overline{x}}{s}\right)^4 - 3$$

berechnet, wobei *s* die gleitende Standardabweichung ist. Der Exzess ist der um den Wert 3 geminderte Wert der empirischen Wölbung (auch Kurtosis genannt), wobei der 3 gerade der Wölbung einer Normalverteilung entspricht. Daraus ergibt sich in der Interpretation des Exzess:

Exzess > 0: Verteilung spitzer als Normalverteilung; deutet auf häufige Peaks in der Stichprobe hin

Exzess < 0: Im Vergleich zur Normalverteilung abgeflachte Verteilung

Der Exzess bietet z.B. die Möglichkeit zur Beurteilung von Stößen im Vibrationssignal, wie sie beim Überrollen von lokalen Schädigungen im Wälzlager entstehen.

Es können in jedem Zyklus ein feste Anzahl (`nInLength`) Samples pro Kanal zur Stichprobenmenge hinzugefügt werden (siehe Ein- und Ausgänge). Die Berechnung der Mittelwerte erfolgt hierbei stets Sample für Sample.

Weitere Anmerkungen

Zur Berechnung eines ersten Ergebnisses müssen 4 Werte vorliegen. Des Weiteren darf die Standardabweichung nicht Null sein. Ergebnisse können unter Umständen ungenau werden, wenn die Eingangswerte viele Größenordnungen auseinanderliegen.

Gedächtniseigenschaften

Die Stichprobenmenge *N*, welche zur Berechnung des aktuellen Ergebnisses genutzt wird, erhöht sich automatisch bei jedem neuen eingehenden Datensatz, d.h. der Baustein verwendet alle Eingangswerte seit dessen Instanziierung. Das Zurücksetzen der Stichprobenmenge auf Null (löschen des internen Speichers des FBs) ist über eine `ResetData()` Methode oder, bei Verwendung der `CallEx()` Methode über die Variable `bResetData` vorgesehen.

NaN Vorkommnis

Falls für einen Kanal noch keine ausreichende Anzahl Eingangswerte für die Berechnung eines Ergebnisses vorliegt oder die Varianz Null beträgt, wird für diesen Kanal der Wert NaN (Not A Number) nach IEEE 754 zurückgegeben. Das Vorliegen dieses Fehlerwertes kann mit der Funktion LrealIsNaN() überprüft werden. Ursache dafür kann entweder sein, dass noch nicht genug Eingangsdaten übergeben wurden oder dass für einzelne Kanäle bisher lediglich NaNs als Eingangswerte übergeben wurden. Eine Varianz von Null kann insbesondere auftreten, wenn die Zeitreihe der Werte konstant ist, etwa wenn aufgrund von Kabelbruch oder Schaltfehlern keine Sensordaten übertragen werden.

Enthält ein Satz von Eingangswerten die spezielle Konstante NaN, so wird der Statistik für diesen Kanal bei diesem Zeitschritt kein Wert hinzugefügt, d.h. sie wird als Indikator für fehlende Werte behandelt.

Behandlung von NaN-Werten

Falls die oben beschriebenen Situationen, welche zu NaN-Werten führen, nicht ausgeschlossen oder sicher vernachlässigt werden können, muss das Anwendungsprogramm diese Fehlerwerte behandeln.

Verhalten bei der Verarbeitung mehrkanaliger Eingangsdaten

Bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`) besteht die Möglichkeit unterschiedlicher Rückgabewerte je Kanal. In diesem Fall können gesonderte Rückgabewerte am Funktionsbaustein abgefragt werden. Sind die Ergebnisse von einem oder mehreren Kanälen unzulässig, jedoch nicht alle Kanäle, dann entspricht der Wert am Baustein `eCM_InfRTime_AmbiguousChannelResults`. Sind die Ergebnisse aller Kanäle unzulässig, dann entspricht der Wert am Baustein `eCM_ErrRTime_ErrornousChannelResults`.

Die Abfrage einer Liste der Rückgabewerte aller Kanäle kann über die Methode `GetChannelErrors()` erfolgen.

Bei der Verarbeitung mehrerer Unterkanäle (`nSubChannels > 0`) ist die Formatierung der Eingangs- und Ausgangsdaten gesondert zu beachten. Bestehen die Eingangsdaten aus einem mehrkanaligen Ergebnis eines vorangestellten Funktionsbausteins, muss der Wert von `nChannels` übernommen werden, eine weitere Konfiguration erfolgt in diesem Fall über den Parameter `nSubChannels`.

Beispiel: Bei der statistischen Betrachtung (z.B. durch FB_CMA_Quantiles) der Frequenzkanäle eines mehrkanaligen Spektrums (z.B. FB_CMA_MagnitudeSpectrum) ist der Wert von `nChannels` identisch zur Anzahl der Eingangssignale zu wählen, die Anzahl der Unterkanäle `nSubChannels` entspricht der Länge des Spektrums.

Beispielimplementierung

Eine exemplarische Implementierung ist unter folgendem Link verfügbar: Gleitender Mittelwert [▶ 383]

Ein- und Ausgänge

Die Ein- und Ausgangspuffer entsprechen folgender Definition (Shape). Die variablen Parameter sind Teil des Bausteineingangs `stInitPars`.

Varianten	Eingangspuffer (MultiArray input stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen	Ausgangspuffer (MultiArray output stream) Elementtyp, Dimensionsanzahl, Dimensionsgrößen
Standardvariante (`nSubChannels = 0`)	`LREAL, 1,` `nInLength`	`LREAL, 1,` `nInLength`
Standardvariante für mehrere Datensätze (`nSubChannels = 0`)	`LREAL, 2,` `nChannels x nInLength`	`LREAL, 2,` `nChannels x nInLength`
Mehrkanalige Variante (`nSubChannels > 0`)	`LREAL, 2,` `nChannels x nSubChannels`	`LREAL, 2,` `nChannels x nSubChannels`
Mehrkanalige Variante für mehrere Datensätze (`nSubChannels > 0`)	`LREAL, 3,` `nChannels x nSubChannels x nInLength`	`LREAL, 3,` `nChannels x nSubChannels x nInLength`

*: Die Länge dieser Dimension kann beliebig gewählt werden und sich so der Applikation bzw. dem Ausgangspuffer des vorrausgehenden Algorithmus anpassen.

Eingangsparameter

Die Eingangsparameter dieses Bausteins repräsentieren Initialisierungsparameter und müssen bereits bei der Deklaration der FB Instanz zugewiesen werden! (Alternativ: Init()-Methode). Sie dürfen nur einmalig zugewiesen werden. Eine Änderung zur Laufzeit ist nicht möglich.

```
VAR_INPUT
    stInitPars        : ST_CM_MovingMoments_InitPars;     // init parameter
    nOwnID            : UDINT;                            // ID for this FB instance
```

```
    aDestIDs: ARRAY[1..cCMA_MaxDest] OF UDINT;           // IDs of destinations for output
    nResultBuffers   : UDINT := 4;                       // number of MultiArrays which should be in
itialized for results (0 for no initialization)
    tTransferTimeout : LTIME := LTIME#500US;             // timeout checking off during access to in
ter-task FIFOs
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MovingMoments_InitPars [▶ 309]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.
- `tTransferTimeout`: Einstellung des synchronen Timeout für interne MultiArray Weiterleitungen. Siehe Kapitel Parallelverarbeitung [▶ 77].

Ausgangsparameter

```
VAR_OUTPUT
    bError         : BOOL;                           // TRUE if an error occurs. Reset by next metho
d call.
    hrErrorCode    : HRESULT;                        // '< 0' = error; '> 0' = info; '0' = no error/
info
    ipErrorMessage : I_TcMessage := fbErrorMessage;  // Shows detailed information about occurred er
rors, warnings and more.
    nCntResults    : ULINT;                          // Counts outgoing results (MultiArrays were ca
lculated and sent to transfer tray).
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.
- `ipErrorMessage`: Beinhaltet nähere Informationen zum aktuellen Rückgabewert. Siehe hierzu den Abschnitt Fehlerbeschreibung und Information [▶ 86]. Für diesen speziellen Schnittstellenzeiger ist intern sichergestellt, dass er immer gültig/zugewiesen ist.

Call():

Die Methode wird jeden Zyklus aufgerufen, um den Algorithmus auf die aktuellen Eingangsdaten anzuwenden. Der Baustein wartet auf Eingangsdaten, sofern die Methode weder neue Ergebnisse noch einen Fehler angibt. Dies ist ein reguläres Verhalten im Ablauf der Analysekette.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Call : HRESULT
VAR_OUTPUT
    bNewResult   : BOOL;       // TRUE every time when outgoing MultiArray was calculated and sent t
o transfer tray.
    bError       : BOOL;       // TRUE if an error occurs.
    hrErrorCode  : HRESULT;    // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

- `bError`: Der Ausgang ist `TRUE`, falls ein Fehler auftritt.
- `hrErrorCode`: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert. Dieser Ausgang ist identisch zum Rückgabewert der Methode.

> Falls ein Timeout eintritt oder kein MultiArray Puffer für das Ergebnis verfügbar ist, so gehen weder die Eingangsdaten noch die Ergebnisdaten verloren. Sie werden beim nächsten Aufruf weitergeleitet.

Init():

Üblicherweise ist diese Methode in einer Condition Monitoring Applikation nicht notwendig. Sie bietet eine Alternative zur Bausteininitialisierung. Die `Init()` Methode darf nur während der Initialisierungsphase der SPS aufgerufen werden. Sie kann nicht während der Laufzeit verwendet werden. Es sei auf die Verwendung von einer `FB_init` Methode oder dem Attribut `'call_after_init'` hingewiesen (siehe TwinCAT SPS Referenz). Hiermit wird zudem die Bausteinkapselung erleichtert.

Die Eingangsparameter der Bausteininstanz dürfen nicht bei der Deklaration zugewiesen werden, falls die Initialisierung mit der Init() Methode erfolgen soll.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD Init : HRESULT
VAR_INPUT
    stInitPars     : ST_CM_MovingMoments_InitPars;       // init parameter
    nOwnID         : UDINT;                              // ID for this FB instance
    aDestIDs       : ARRAY[1..cCMA_MaxDest] OF UDINT;    // IDs of destinations for output
    nResultBuffers: UDINT := 4;                          // number of MultiArrays which should be ini
tialized for results (0 for no initialization)
END_VAR
```

- `stInitPars`: Baustein-spezifische Struktur mit Initialisierungsparametern vom Typ ST_CM_MovingMoments_InitPars [▶ 309]. Die Parameter müssen mit der obigen Definition der Ein- und Ausgangspuffer übereinstimmen.
- `nOwnID`: Identifiziert die Bausteininstanz mit einer eindeutigen ID. Diese muss immer größer als null sein. Eine bewährte Vorgehensweise ist die Definition einer Enumeration für diesen Zweck.
- `aDestIDs`: Legt die Ziele durch Angabe derer IDs fest, zu denen die Ergebnisse weitergeleitet werden sollen. Die Definition des Ausgangspuffers (wie oben beschrieben) muss mit der Definition des Eingangspuffers jedes gewählten Zieles übereinstimmen.
- `nResultBuffers`: Der Funktionsbaustein initialisiert einen Transfer Tray Stream mit der gegebenen Anzahl an MultiArray Puffern. Der Defaultwert ist vier.

ResetData():

Die Methode löscht alle bereits hinzugefügten Datensätze, vgl. Gedächtniseigenschaft des Funktionsbausteins. Wird nach einem `ResetData()` die `Call()`-Methode wieder aufgerufen, muss entsprechend erst der interne Speicher wieder aufgefüllt werden um ein gültiges Ergebnis zu berechnen.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD ResetData : HRESULT
VAR_INPUT
END_VAR
```

PassInputs():

Solange eine `FB_CMA_Source` Instanz aufgerufen wird und somit Signaldaten zu einem Zielblock übertragen werden, müssen wie in der API SPS Referenz [▶ 84] erläutert alle weiteren Blöcke der Analysekette zyklisch aufgerufen werden.
Manchmal ist es sinnvoll, einen Algorithmus für eine bestimmte Zeit nicht auszuführen. Beispielsweise sollten manche Algorithmen nur nach vorherigem Training bzw. Konfiguration ausgeführt werden. Zwar muss der Funktionsbaustein dennoch zyklisch aufgerufen werden, aber es ist ausreichend, wenn die am Baustein ankommenden Daten im Kommunikationsring [▶ 77] weitergeleitet werden. Dies geschieht mittels der Methode `PassInputs()` anstelle der Methode `Call()`. Hierbei wird der Algorithmus selbst nicht aufgerufen und entsprechend kein Ergebnis berechnet sowie kein Ausgangspuffer generiert.

- **Rückgabewert**: Falls ein Fehler auftritt, wird ein entsprechender Fehlercode vom Typ `HRESULT` ausgegeben. Mögliche Werte sind in der Liste der Fehlercodes [▶ 386] erläutert.

```
METHOD PassInputs : HRESULT
VAR_INPUT
END_VAR
```

GetChannelErrors():

Die Methode ermöglicht die Abfrage einer Liste der kanalspezifischen Rückgabewerte bei der Verarbeitung mehrerer Kanäle (`nChannels > 1`). Ein Aufruf ist sinnvoll für den Fall, dass der Rückgabewert des Bausteins einem der Werte `eCM_InfRTime_AmbiguousChannelResults` oder `eCM_ErrRTime_ErrornousChannelResults` entspricht.

- **Rückgabewert**: Information über den Ausleseprozess der Liste der Fehlercodes. Der Wert wird auf `TRUE` gesetzt, falls die Abfrage erfolgreich war, `FALSE` anderenfalls.

```
METHOD GetChannelErrors : BOOL
VAR_IN_OUT
    aChannelErrors : ARRAY[*] OF HRESULT;
END_VAR
```

- `aChannelErrors`: Fehlerliste vom Typ `HRESULT` der Länge `nChannels`.

Ähnliche Funktionsbausteine

Der Baustein FB_CMA_MovingMean [▶ 270] berechnet den gleitenden Mittelwert non Eingangswerten.

Der Baustein FB_CMA_MovingStandardDeviation [▶ 274] berechnet die gleitende Standardabweichung von Eingangswerten.

Der Baustein FB_CMA_MovingSkew [▶ 278] berechnet die gleitende Schiefe von Eingangswerten.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

5.2 Funktionen

5.2.1 F_MA_IsNAN

Diese Funktion testet, ob ein Wert NaN (Not-a-Number) ist. Ist der Rückgabewert TRUE, dann ist der Wert NaN.

```
FUNCTION F_MA_IsNAN : BOOL
VAR_INPUT
     fValue : LREAL;
END_VAR
```

Für weitere Informationen lesen Sie das Kapitel Handling von NaN Werten [▶ 76].

Die Funktion ist überholt.

Bitte verwenden Sie stattdessen die Funktion LrealIsNaN() aus der Tc2_Utilities Bibliothek.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_MultiArray

5.2.2 F_CM_CalculateRecommendedOverlap

Diese Funktion berechnet eine empfohlene Überlappung basierend auf der *amplitude flatness* und der *overlap correlation* der Fensterfunktion. Siehe Abschnitt: Berechnung der empfohlenen Überlappung [▶ 22].

```
FUNCTION F_CM_CalculateRecommendedOverlap : UDINT
VAR_INPUT
    eWindowType         : E_CM_WindowType;          (* Window type. *)
    aWindowParameters   : T_CM_WindowParameters;    (* Window parameters for specific windows, e.g. F
latTop. *)
    nWindowLength       : UDINT;                    (* Length of analysis window. *)
END_VAR
```

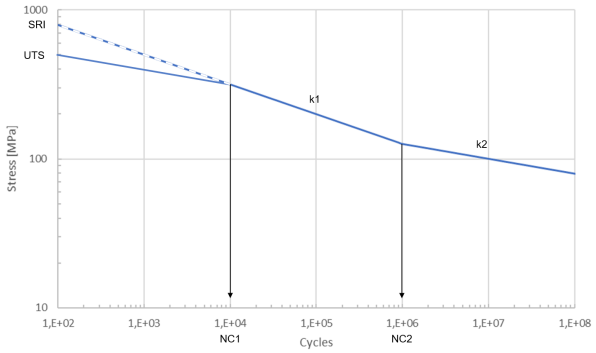
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.2.3 F_CM_CalculateWoehlerCurve

Diese Funktion berechnet aus Materialparametern als Input eine Wöhler-Kurve.

```
FUNCTION F_CM_CalculateWoehlerCurve : HRESULT
VAR_INPUT
    fSRI                : LREAL;                        (* Stress Range Intercept. *)
    fUTS                : LREAL;                        (* Ultimate Tensile Strength OF the material.
 *)
    bUseUTSCorrection   : BOOL;                         (* Apply UTS correction. *)
    fK1                 : LREAL;                        (* Gradient of Woehler Curve for region N = 1
 to nNC1. *)
    fK2                 : LREAL;                        (* Gradient of Woehler Curve starting from nN
C2. *)
    nNC1                : ULINT;                        (* Transition Point for UTS correction. *)
    nNC2                : ULINT;                        (* Fatigue Transition Point between fK1 and f
K2. *)
    fRRatio             : LREAL;                        (* R-Ratio of the material test. *)
    fStepsize           : LREAL;                        (* Width of stress bins. *)
    nCurveLength        : UDINT;                        (* Length of curve. *)
    pCurve              : POINTER TO POINTER TO LREAL;  (* Pointer to array data. *)
END_VAR
```



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.2.4 F_CM_CalculateWindow

Diese Funktion berechnet Fensterfunktionen zu gegebenen Eingangsparametern. Siehe Abschnitt: Fensterfunktionen [▶ 20].

```
FUNCTION F_CM_CalculateWindow : HRESULT
VAR_INPUT
    eWindowType        : E_CM_WindowType;              (* Window type. *)
    aWindowParameters : T_CM_WindowParameters;         (* Window parameters for specific windows, e.g
. FlatTop. *)
    nWindowLength      : UDINT;                        (* Length of analysis window. *)
    pWindow            : POINTER TO POINTER TO LREAL;  (* Pointer to array data. *)
END_VAR
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.2.5 F_CM_ApplySpectralScaling

Diese Funktion führt eine Skalierung von Spektralwerten durch. Siehe Abschnitt: Skalierung von Spektren [▶ 26]

```
FUNCTION F_CM_ApplySpectralScaling : HRESULT
VAR_INPUT
    eScalingType        : E_CM_ScalingType;             (* Scaling type. *)
    nFFT_Length         : UDINT;                        (* Length of FFT. *)
    nOrder              : UDINT;                        (* Order of spectrum: 1 = magnitude, 2 = powe
r. *)
    fSampleRate         : LREAL;                        (* Sample rate in Hertz. *)
    bTransformToDecibel : BOOL;                         (* Transform to decibel. *)
    fDecibelThreshold   : LREAL;                        (* Minimum argument of decadic logarithm for
64- bit IEEE 754 arithmetic. *)
    nWindowLength       : UDINT;                        (* Length of analysis window. *)
    pWindow             : POINTER TO POINTER TO LREAL; (* Pointer to window data. *)
    nResultLength       : UDINT;                        (* Length of spectrum, i.e. nFFT_Length or nF
FT_Length / 2 + 1. *)
    pSpectrum           : POINTER TO POINTER TO LREAL; (* Pointer to spectrum. *)
END_VAR
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3 Datentypen

5.3.1 E_CM_ComplexDataHandling

```
TYPE E_CM_ComplexDataHandling : (
    eCM_CplxReal           := 0,   (* Real part of complex data. *)
    eCM_CplxImag           := 1,   (* Imaginary part of complex data. *)
    eCM_CplxAbs            := 2,   (* Absolute value of complex data. *)
    eCM_CplxPhase          := 3,   (* Phase/angle of complex data. *)
    eCM_CplxUnwrappedPhase := 4    (* Unwrapped Phase/angle of complex data. *)
) UDINT;
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.2 E_CM_CorrelationMode

```
TYPE E_CM_CorrelationMode :
    eCM_Base               := 0,    (* The correlation is calculate as the convolution of the two sig
nals. *)
    eCM_Normed             := 1,    (* The correlation is calculate as the of the two signals divided
 by the number of elements added. *)
    eCM_Covariance         := 2,    (* The covariance with bias (divided by N instead N-1) is calcula
ted. *)
    eCM_CovarianceBessel   := 3,    (* The covariance is calculated. *)
    eCM_Pearson            := 4     (* The correlation is calculate as the Pearson correlation coeffi
cient. *)
) UDINT;
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.3 E_CM_MCoefOrder

```
TYPE E_CM_MCoefOrder :
(
    eCM_N        := 0,    (* Count of included cases. *)
    eCM_Mean     := 1,    (* Mean value. *)
    eCM_StDev    := 2,    (* Standard deviation. *)
    eCM_Skew     := 3,    (* Skew value (third moment). *)
    eCM_Kurtosis := 4     (* Excess Kurtosis value. *)
) UDINT;
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.4 E_CM_MeanStressCorrection

```
TYPE E_CM_MeanStressCorrection : (
    eCM_NoCorrection := 0,    (* No correction of the stress matrix. *)
    eCM_Goodman      := 1,    (* Goodman correction of the stress matrix. *)
    eCM_Gerber       := 2,    (* Gerber correction of the stress matrix.  *)
) UDINT;
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.5 E_CM_ScalingType

Weitere Details finden Sie im Kapitel Spektrumsskalierung [▶ 390].

```
TYPE E_CM_ScalingType :
(
    eCM_NoScaling            := 0,
    eCM_DiracScaling         := 1,
    eCM_PeakAmplitude        := 2,
    eCM_RootPowerSum         := 3,
    eCM_RMS                  := 4,
    eCM_GainCorrection       := 5,
    eCM_PowerSpectralDensity := 6,
    eCM_UnitaryScaling       := 7
) UDINT;
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_CM_Base

5.3.6 E_CM_SlidingDFTType

```
TYPE E_CM_SlidingDFTType : (
    eCM_SDFT            := 0,    (* Plain (damped) SDFT. *)
    eCM_mSDFT           := 1,    (* Modulated SDFT. *)
    eCM_SlidingGoertzel := 2,    (* Sliding Goertzel transform. *)
    eCM_DouglasAndSoh   := 3     (* Douglas and Soh algorithm. *)
) UDINT;
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.7 E_CM_SpectrumType

```
TYPE E_CM_SpectrumType : (
    eCM_DFT       := 0,     (* Plain DFT. *)
    eCM_Magnitude := 1,     (* Magnitude spectrum. *)
    eCM_Power     := 2      (* Power spectrum. *)
) UDINT;
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.8 E_CM_UnwrapMethod

```
TYPE E_CM_UnwrapMethod :
(
    eCM_NoUnwrapping        := 0,
    eCM_ThresholdUnwrapping := 1
) UDINT;
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.9 E_CM_WindowMode

```
TYPE E_CM_WindowMode : (
    eCM_SlidingWindow := 0,     (* The analysis is done over the last cycles in each cycle. *)
    eCM_FixWindow     := 1,     (* The analysis is done over an interval of fix size. If one interval
 is passed another interval begins. *)
    eCM_Continuous    := 2      (* The analysis is done for the whole time since starting of the anal
ysis. *)
) UDINT;
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.10 E_CM_WindowType

Weitere Details finden Sie im Kapitel Fensterfunktionen [▶ 20].

```
TYPE E_CM_WindowType :
(
    eCM_HannWindow        := 16#05300901,
    eCM_RectangularWindow := 16#05300902,
    eCM_BartlettWindow    := 16#05300905,
    eCM_HammingWindow     := 16#05300906,
    eCM_KaiserWindow      := 16#05300907,
    eCM_FlatTopWindow     := 16#05300917
) UDINT;
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.11 E_MA_ElementTypeCode

```
TYPE E_MA_ElementTypeCode :
(
    eMA_TypeCode_NONE      := 0,     (* Internally used. *)
    eMA_TypeCode_BYTE      := 2,
    eMA_TypeCode_CHAR      := 3,
    eMA_TypeCode_WCHAR     := 4,
    eMA_TypeCode_BOOL      := 5,     (* Boolean type. *)
    eMA_TypeCode_INT16     := 6,
    eMA_TypeCode_UINT16    := 7,
    eMA_TypeCode_INT32     := 8,     (* Used e.g. for classification results. *)
    eMA_TypeCode_UINT32    := 9,
    eMA_TypeCode_INT64     := 10,
    eMA_TypeCode_UINT64    := 11,    (* 64-bit long unsigned. Use  for statistical counters. *)
    eMA_TypeCode_REAL      := 12,    (* Unsupported: 32-bit floating point type. *)
    eMA_TypeCode_LREAL     := 13,    (* Standard floating-point type. *)
    eMA_TypeCode_LCOMPLEX  := 15,    (* Standard 128-bit complex type (real part first). *)
    eMA_TypeCode_SPUNKNOWN := 22,    (* Used for TCOM Pointers. *)
) UDINT;
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.12 T_CM_WindowParameters

```
TYPE T_CM_WindowParameters : ARRAY [0..4] OF LREAL;
END_TYPE
```

Verwendung von `T_CM_WindowParameters`

Die Angabe freier Parameter zur Konfiguration von Fensterfunktionen erfolgt über ein allgemeines Array der Länge 5. Bei der Wahl `eCM_KaiserWindow` wird nur der erste Wert, bei `eCM_FlatTopWindow` werden alle Werte dieses Arrays verwendet. Für alle anderen Fenstertypen sind die Parameter nicht relevant.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.13 Fehlercodes

5.3.13.1 E_CM_ErrorCode

Code (HRESULT)	Symbol	Beschreibung / Lösung
0	eCM_OK	Kein Fehler, alles ist OK.
Logikfehler		
16#9851_0100	eCM_ErrLogic	Allgemeiner Logikfehler
16#9851_0102	eCM_ErrLogic_NotImplemented	Funktion ist noch nicht implementiert.
16#9851_0110	eCM_ErrLogic_LackOfInitialization	Algorithmus ist nicht korrekt initialisiert.
16#9851_0125	eCM_ErrLogic_InvalidObjectState	Operation ist für den Objektzustand ungültig.
Konfigurationsfehler		
16#9851_1000	eCM_ErrConfig	Allgemeiner Konfigurationsfehler

Code (HRESULT)	Symbol	Beschreibung / Lösung
Abbildungen an ADSERR_DEVICE_NOMEMORY		
16#9851_1100	eCM_ErrConfig_OutOfMemory	Speicherallokation ist fehlgeschlagen. => Routerspeicher erhöhen (siehe Kapitel Speicherverwaltung [▶ 73])
alle nachstehenden Fehler führen zu einem HRESULT von ADS_E_INVALIDPARM		
16#9851_1800	eCM_ErrConfig_IllegalParameter	Konfigurationsparameter ist nicht gültig.
16#9851_1900	eCM_ErrConfig_ParameterOutOfRange	Konfigurationsparameter ist außerhalb des Bereichs.
16#9851_1901	eCM_ErrConfig_ParameterOutOfRange_NoPowerOfTwo	Parameter ist keine Zweierpotenz wie gefordert.
16#9851_1902	eCM_ErrConfig_ParameterOutOfRange_FFT_length_Zero	Die FFT-Länge ist null.
16#9851_1903	eCM_ErrConfig_ParameterOutOfRange_DecibelThreshold_too_small	Dezibelschwelle ist zu niedrig, was Unterlauf verursachen könnte.
16#9851_1904	eCM_ErrConfig_ParameterOutOfRange_LogThreshold_too_small	Logarithmenschwelle ist zu niedrig, was Unterlauf verursachen könnte.
16#9851_1905	eCM_ErrConfig_ParameterOutOfRange_nInLength_Minimum_two	Eingangslänge ist zu klein. Wert muss mindestens zwei sein.
16#9851_190D	eCM_ErrConfig_ParameterOutOfRange_nChannels_smaller_one	Anzahl der Kanäle ist null.
16#9851_190E	eCM_ErrConfig_ParameterOutOfRange_nBins_smaller_one	Anzahl der Behälter ist null.
16#9851_190F	eCM_ErrConfig_ParameterOutOfRange_invalid_limit_interval	Untere Grenze ist nicht kleiner als obere Grenze.
16#9851_1910	eCM_ErrConfig_ParameterOutOfRange_unknown_scaling_type	Skalierungstyp ist nicht bekannt.
16#9851_1911	eCM_ErrConfig_ParameterOutOfRange_illegal_quantile_argument	Quantilargument liegt außerhalb [0 .. 1].
16#9851_1912	eCM_ErrConfig_ParameterOutOfRange_illegal_threshold_order	Schwellwert Reihenfolge ist ungültig. Die Schwellwerte müssen in aufsteigender Reihenfolge sein.
16#9851_1913	eCM_ErrConfig_ParameterOutOfRange_threshold_number_toolarge	Mehr Schwellwerte angegeben als konfiguriert.
16#9851_1914	eCM_ErrConfig_ParameterOutOfRange_Integration_limit_too_low	Integrationsgrenze ist zu niedrig.
16#9851_1915	eCM_ErrConfig_ParameterOutOfRange_Integration_limit_too_high	Integrationsgrenze ist zu hoch.
16#9851_1916	eCM_ErrConfig_ParameterOutOfRange_Integration_limits_inconsistent	Integrationsgrenzen sind inkonsistent.
16#9851_1917	eCM_ErrConfig_ParameterOutOfRange_Samplerate_not_positive	Abtastrate ist null oder negativ.
16#9851_191A	eCM_ErrConfig_ParameterOutOfRange_TimeConstant_too_small	Übergebene Zeitkonstante ist zu klein.
16#9851_192C	eCM_ErrConfig_ParameterOutOfRange_fScaleFactor_invalid	Skalierungsfaktor ist ungültig.
16#9851_192D	eCM_ErrConfig_ParameterOutOfRange_DivThreshold_too_small	Divisionsschwelle ist zu niedrig, was Unterlauf verursachen könnte.
16#9851_192E	eCM_ErrConfig_ParameterOutOfRange_nMaxBands_zero	Anzahl konfigurierbarer Bänder ist null.
16#9851_192F	eCM_ErrConfig_ParameterOutOfRange_nOrder_invalid	Ordnung ist unzulässig.
16#9851_1930	eCM_ErrConfig_ParameterOutOfRange_fDecayTime_invalid	fDecayTime ist nicht zulässig.
16#9851_1B00	eCM_ErrConfig_ParameterMismatch	Parameterabhängigkeit ist nicht erfüllt.

Code (HRESULT)	Symbol	Beschreibung / Lösung
16#9851_1B01	eCM_ErrConfig_ParameterMismatch_WindowLength_larger_FFT_length	Fensterlänge ist größer als FFT-Länge.
16#9851_1B06	eCM_ErrConfig_ParameterMismatch_overlap_larger_BufferLength	Überlappung ist größer als Pufferlänge.
Laufzeitfehler (während der Datenverarbeitung)		
diese Fehler führen zu einem HRESULT von ADS_E_INVALIDPARM		
16#9851_2000	eCM_ErrRTime	Allgemeiner Laufzeitfehler
16#9851_2011	eCM_ErrRTime_IllegalPointer	Unzulässiger (Schnittstellen-) Zeiger oder Speicheradresse
16#9851_2015	eCM_ErrRTime_NonMonotonousInputData	Eingangsdaten sind nicht monoton wie verlangt.
16#9851_2016	eCM_ErrRTime_ErrornousChannelResults	Ergebnisse auf mindestens einem Kanal sind fehlerhaft.
16#9851_2021	eCM_ErrRTime_IllegalBuffer	Unzulässiger Datenpuffer
illegaler Eingangspufferparameter (kann in ADS-Aufrufen mit festem Puffer vorkommen)		
16#9851_2023	eCM_ErrRTime_IllegalInput	Unzulässige Eingangspufferparameter
16#9851_2025	eCM_ErrRTime_IllegalInputDimensionNumber	Eingangspuffer hat unzulässige Anzahl an Dimensionen.
16#9851_2026	eCM_ErrRTime_IllegalInputShape	Eingangspuffer hat unzulässige Form.
16#9851_2028	eCM_ErrRTime_IllegalInputDataType	Unzulässiger Elementtyp des Eingangspuffers
16#9851_202A	eCM_ErrRTime_IllegalInputNoArray	Kein Multiarray als Eingangsparameter übergeben.
illegaler Ausgangspufferparameter (kann in ADS-Aufrufen mit festem Puffer vorkommen)		
16#9851_2030	eCM_ErrRTime_IllegalOutput	Unzulässige Ausgangspufferparameter
16#9851_2032	eCM_ErrRTime_IllegalOutputDimensionNumber	Ausgangspuffer hat unzulässige Anzahl an Dimensionen.
16#9851_2033	eCM_ErrRTime_IllegalOutputShape	Ausgangspuffer hat unzulässige Form.
16#9851_2034	eCM_ErrRTime_IllegalOutputDataType	Unzulässiger Elementtyp des Ausgangspuffers
16#9851_2035	eCM_ErrRTime_IllegalOutputNoArray	Kein Multiarray als Ausgangsparameter übergeben.
Interpolationsfehler		
16#9851_2060	eCM_ErrRTime_Interpolation	Allgemeiner Interpolationsfehler
16#9851_2063	eCM_ErrRTime_Interpolation_OutOfBounds	Eingangsdaten sind außerhalb der Grenzen.
16#9851_2064	eCM_ErrRTime_Interpolation_InvalidDimension	Array mit Eingangsdaten hat unzulässige Größe für verwendete Methode.
16#9851_2065	eCM_ErrRTime_Interpolation_InvalidConstraints	Beschränkungen an die Ableitungen an den Endpunkten können nicht eingehalten werden.
Info Codes		
16#9851_B103	eCM_InfRTime_InsufficientInputData	Unzureichende Eingangsdaten für ein repräsentatives Ergebnis
16#9851_B104	eCM_InfRTime_InvalidInputData	Eingangsdaten sind unzulässig.
16#9851_B105	eCM_InfRTime_AmbiguousChannelResults	Ergebnisse auf mindestens einem Kanal sind mehrdeutig.
16#9851_B106	eCM_InfRTime_nBufferLength_too_small	Konfigurierte Pufferlänge ist zu klein für Eingangsdaten.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.13.2 E_CMA_ErrorCode

Diese Fehlercodes sind lediglich im Echtzeitkontext erforderlich. Beachten Sie, dass die Analysefunktionsbausteine in der SPS Deklaration korrekt zugewiesen werden müssen.
Die Konfigurationsfehler müssen zuerst behoben werden, anschließend die Initialisierungsfehler.
Beispiel: Wenn eine Instanz den Fehler eCMA_ErrConfig_InvalidOwnID ausgibt, muss dieser zuerst behoben werden. Laufzeitfehler in anderen Funktionsblöcken können Folgefehler sein.

Code (HRESULT)	Symbol	Beschreibung / Lösung
0	eCMA_OK	Kein Fehler, alles ist OK
Konfigurationsfehler		
16#9852_0101	eCMA_ErrConfig_InvalidOwnID	ungültiger Transfer eigene ID wurde zugewiesen
16#9852_0102	eCMA_ErrConfig_InvalidDestID	ungültige Transferziel-IDs wurden zugewiesen
16#9852_0103	eCMA_ErrConfig_InvalidBufferNumber	ungültige Nummer von MultiArrays, die für Ergebnisse initialisiert werden sollte
16#9852_0104	eCMA_ErrConfig_InvalidTimeout	ungültige Timeout-Bedingung: 0us << tTransferTimeout << Aufgabenzykluszeit
Initialisierungsfehler		
16#9852_0201	eCMA_ErrInit_IllegalInitContext	Initialisierung nicht möglich. Illegaler Initialisierungskontext oder interne Mitglieder nicht initialisiert.
16#9852_0202	eCMA_ErrInit_InitTransferTrayFailed	Initialisierung der Übergabeablage fehlgeschlagen. TcCOM Objektzustände und Routerspeicher überprüfen (siehe Speicherverwaltung [▶ 73]). Installierte TwinCAT-Version überprüfen (siehe Systemanforderungen [▶ 66]).
16#9852_0203	eCMA_ErrInit_NoStreamAllocated	Die Analysekette ist inkorrekt. Alle OwnIDs und DestIDs überprüfen.
16#9852_0204	eCMA_ErrInit_StreamOverrun	Nicht genügend Streams verfügbar. ST_CM_TransferTray_InitPars anpassen
Laufzeitfehler		
16#9852_0301	eCMA_ErrRTime_InvalidPointer	NULL-Zeiger wurde zugewiesen
16#9852_0302	eCMA_ErrRTime_InvalidDataBufferSize	ungültige Größe des Datenpuffers wurde zugewiesen
16#9852_0303	eCMA_ErrRTime_InvalidElementType	ungültiger Elementtyp wurde zugewiesen
16#9852_0304	eCMA_ErrRTime_InvalidElementCnt	Elementzählung stimmt nicht überein. (Anzahl der Elemente, MultiArray Puffergröße und Startindex überprüfen)
16#9852_0305	eCMA_ErrRTime_InvalidStartIndex	ungültiger pStartIndex wurde zugewiesen (Puffergrößen überprüfen)
16#9852_0311	eCMA_ErrRTime_MissingConfiguration	Argument nicht konfiguriert. Zuerst Methode Configure() aufrufen.
16#9852_0321	eCMA_ErrRTime_NoMultiArrayAvailable	kein Multiarray für Ergebnis verfügbar. Analysekette, Aufgabenzykluszeiten und die Anzahl der MultiArrays (normalerweise mindestens 3 in jedem Ring) überprüfen

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4013	PC oder CX (x86, x64)	Tc3_CM

5.3.13.3 E_MA_ErrorCode

Code (HRESULT)	Symbol	Beschreibung / Lösung
0	eMA_OK	Kein Fehler, alles ist OK.
Logikfehler		
16#9871_0100	eMA_ErrLogic	Allgemeiner Logikfehler
16#9871_0102	eMA_ErrLogic_NotImplemented	Funktion ist noch nicht implementiert.
16#9871_0110	eMA_ErrLogic_LackOfInitialization	Algorithmus ist nicht korrekt initialisiert.
16#9871_0126	eMA_ErrLogic_ObjectCreationFailed	Objekterstellung fehlgeschlagen. Vermutlich fehlt der Treiber oder ist veraltet.
Konfigurationsfehler		
16#9871_1000	eMA_ErrConfig	Allgemeiner Konfigurationsfehler
Abbildungen an ADSERR_DEVICE_NOMEMORY		
16#9871_1100	eMA_ErrConfig_OutOfMemory	Speicherallokation ist fehlgeschlagen. => Routerspeicher erhöhen (siehe Kapitel Speicherverwaltung [▶ 73])
alle nachstehenden Fehler führen zu einem HRESULT von ADS_E_INVALIDPARM		
16#9871_1800	eMA_ErrConfig_IllegalParameter	Konfigurationsparameter ist nicht gültig.
16#9871_1900	eMA_ErrConfig_ParameterOutOfRange	Konfigurationsparameter ist außerhalb des Bereichs.
Laufzeitfehler (während der Datenverarbeitung)		
diese Fehler führen zu einem HRESULT von ADS_E_INVALIDPARM		
16#9871_2000	eMA_ErrRTime	Allgemeiner Laufzeitfehler
16#9871_2011	eMA_ErrRTime_IllegalPointer	Unzulässiger (Schnittstellen-) Zeiger oder Speicheradresse
16#9871_2012	eMA_ErrRTime_EmptyArray	MultiArray hat keine Daten (Produkt der Dimensionsgrößen ist null).
16#9871_2013	eMA_ErrRTime_InstanceExists	Es existiert bereits eine allokierte Instanz.
16#9871_2014	eMA_ErrRTime_NoInstanceExists	Es existiert keine allokierte Instanz.
16#9871_2021	eMA_ErrRTime_IllegalBuffer	Unzulässiger Datenpuffer
16#9871_2022	eMA_ErrRTime_IllegalSubarraySize	Unzulässige Größe des Teilarrays
16#9871_2029	eMA_ErrRTime_IllegalPermutation	Angeforderte Permutation der Dimensionen ist nicht zulässig.
illegaler Eingangspufferparameter		
16#9871_2023	eMA_ErrRTime_IllegalInput	Unzulässige Eingangspufferparameter
16#9871_2025	eMA_ErrRTime_IllegalInputDimensionNumber	Eingangspuffer hat unzulässige Anzahl an Dimensionen.
16#9871_2026	eMA_ErrRTime_IllegalInputShape	Eingangspuffer hat unzulässige Form.
16#9871_2028	eMA_ErrRTime_IllegalInputDataType	Unzulässiger Elementtyp des Eingangspuffers
16#9871_202A	eMA_ErrRTime_IllegalInputNoArray	Kein Multiarray als Eingangsparameter übergeben.
illegaler Ausgangspufferparameter		
16#9871_2030	eMA_ErrRTime_IllegalOutput	Unzulässige Ausgangspufferparameter

Code (HRESULT)	Symbol	Beschreibung / Lösung
16#9871_2035	eMA_ErrRTime_IllegalOutputNoArray	Kein Multiarray als Ausgangsparameter übergeben.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14 InitPars Strukturen

5.3.14.1 ST_CM_AnalyticSignal_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_AnalyticSignal_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length     : UDINT := 512;     (* Length of FFT. *)
    nWindowLength   : UDINT := 400;     (* Length of analysis window. *)
    nChannels       : UDINT := 1;       (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Artefakte vermeiden

Der Wert von `nFFT_Length` muss mindestens gleich dem Wert von `nWindowLength` sein. Um Artefakte bei der Berechnung zu vermeiden, sollte `nFFT_Length` jedoch darüber hinaus mindestens 25% größer sein als `nWindowLength`. Eine Erhöhung der FFT -Länge gegenüber der Fensterlänge ist bei diesem Baustein sinnvoll, um Zirkuläres Aliasing zu vermeiden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.2 ST_CM_ArgSort_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_ArgSort_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nInLength         : UDINT := 256;      (* Length of input data array.  *)
    bSortDownward     : BOOL  := FALSE;    (* If true, sort in descending order (largest values first
). *)
    fScaleFactor      : LREAL := 1.0;      (* Scaling factor to transform index values, for example t
o frequency values. *)
    nChannels         : UDINT := 1;        (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nInLength` ist die Länge des Eingangs-Arrays.
- `bSortDownward` ist ein Flag, mit dem gewählt werden kann, ob die Daten aufsteigend oder absteigend sortiert werden sollen. Ist `bSortDownward` TRUE, so werden die größten Werte nach vorne gestellt.

- `fScaleFactor` kann genutzt werden, um sich anstelle der Index-Position (fScaleFactor = 1) beispielsweise direkt die Amplituden mit zugehöriger Frequenzen anzeigen zu lassen.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.3 ST_CM_CrestFactor_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_CrestFactor_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels          : UDINT := 32;                  (* Number of channels. *)
    nSubChannels       : UDINT := 0;                   (* Number of subchannels. *)
    nBufferLength      : UDINT := 250;                 (* Buffer length. *)
    fDecibelThreshold  : LREAL := cCM_MinArgLog10;     (* Minimum argument of decadic logarithm for 6
4-bit IEEE 754 arithmetic. *)
END_STRUCT
END_TYPE
```

- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.
- `nSubChannels` definiert die Anzahl von unabhängigen Unterkanälen. Diese muss größer oder gleich Null sein.
- `nBufferLength` ist die Zahl der im internen Puffer gehaltenen Eingangswerte je Kanal.
- `fDecibelThreshold` ist ein sehr kleiner Fließkommawert größer als Null. Werte, die kleiner als diese Zahl sind, werden vor einer Transformation in die Dezibel-Skala durch diesen Wert ersetzt. (Zweck ist die Vermeidung von Wertbereichsfehlern. Der Logarithmus von Null ist nicht definiert und strebt für den Grenzwert kleiner Argumente gegen minus unendlich. Entsprechendes gilt für das Argument der Zahl Null, arg(0). Der kleinste mögliche Wert ist 2.3e-308, dies entspricht der Konstanten `cCM_MinArgLog10`.)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.4 ST_CM_CrestFactorPlus_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_CrestFactorPlus_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels           : UDINT := 32;              (* Number of channels. *)
    nSubChannels        : UDINT := 0;               (* Number of subchannels. *)
    nBufferLength       : UDINT := 250;             (* Buffer length. *)
    bTransformToDecibel : BOOL := TRUE;             (* Transform to Decibel. *)
    fDecibelThreshold   : LREAL := cCM_MinArgLog10;(* Minimum argument of decadic logarithm for 64-
bit IEEE 754 arithmetic. *)
END_STRUCT
END_TYPE
```

- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.
- `nSubChannels` definiert die Anzahl von unabhängigen Unterkanälen. Diese muss größer oder gleich Null sein.
- `nBufferLength` ist die Zahl der im internen Puffer gehaltenen Eingangswerte je Kanal.
- `bTransformToDecibel` ist ein boolescher Wert, der angibt, ob das Ergebnis der FFT in die Dezibel-Skala transformiert werden soll, entsprechend der Transformation x → 20 * log10(x).

- `fDecibelThreshold` ist ein sehr kleiner Fließkommawert größer als Null. Werte, die kleiner als diese Zahl sind, werden vor einer Transformation in die Dezibel-Skala durch diesen Wert ersetzt. (Zweck ist die Vermeidung von Wertbereichsfehlern. Der Logarithmus von Null ist nicht definiert und strebt für den Grenzwert kleiner Argumente gegen minus unendlich. Entsprechendes gilt für das Argument der Zahl Null, arg(0). Der kleinste mögliche Wert ist 2.3e-308, dies entspricht der Konstanten `cCM_MinArgLog10`.)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.5 ST_CM_ComplexDataHandling_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_ComplexDataHandling_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    eComplexDataHandling : E_CM_ComplexDataHandling := E_CM_ComplexDataHandling.eCM_CplxReal;    (*
Extraction of real or imaginary part or calculation of absolute value  or phase of complex data. *)
    nInLength            : UDINT := 256;                                                          (*
Length of input data array. *)
    fPhaseThreshold      : LREAL := 3.14159265358979;                                             (*
Threshold value for phase unwrapping. *)
    nChannels            : UDINT := 1;                                                            (*
Number of channels. *)
END_STRUCT
END_TYPE
```

- `eComplexDataHandling` definiert die Aktion, welche mittels Call() ausgeführt wird, z.B. das extrahieren von Real- oder Imaginärteil, Berechnung des Betrages oder der Phase der komplexen Eingangsdaten.
- `nInLength` ist die Länge des Eingangs-Arrays.
- `fPhaseThreshold` definiert den Grenzwert zum Ausrollen der Phase. In der Regel wird der Wert Pi verwendet.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.6 ST_CM_ComplexFFT_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_ComplexFFT_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length  : UDINT := 256;     (* Length of FFT. *)
    bForward     : BOOL  := TRUE;    (* Flag indicating forward FFT. *)
    nChannels    : UDINT := 1;       (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `bForward` ist ein boolescher Parameter, der die Richtung der FFT angibt. Ist der Wert `TRUE`, so wird die normale FFT berechnet. Andernfalls wird die inverse FFT verwendet.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.7 ST_CM_Correlation_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_Correlation_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nNegativeShift   : UDINT := 0;                                          (* Maximum negative
 time shift in samples. *)
    nPositiveShift   : UDINT := 0;                                          (* Maximum positive
 time shift in samples. *)
    nStepsize        : UDINT := 0;                                          (* Stepsize in Samp
les. *)
    eCorrelationMode : E_CM_CorrelationMode := E_CM_CorrelationMode.eCM_Normed;  (* Calculation mode
 of correlation. *)
    eWindowMode      : E_CM_WindowMode := E_CM_WindowMode.eCM_FixWindow;    (* Window handling,
 i.e. sliding, fixed, continuous. *)
    nWindowLength    : UDINT := 1000;                                       (* Length of analys
is window. *)
    nChannels        : UDINT := 1;                                          (* Number of channe
ls. *)
END_STRUCT
END_TYPE
```

- `nNegativeShift` gibt die maximale zeitliche Verschiebung in negativer Richtung in Samples an.
- `nPositiveShift` gibt die maximale zeitliche Verschiebung in positiver Richtung in Samples an.
- `nStepsize` definiert die Schrittweite in Samples in welcher die Korrelationskoeffizienten berechnet werden.
- `eCorrelationMode` definiert die Berechnungsvariante für die Korrelationsparameter.
- `eWindowMode` definiert das der Berechnung zugrundeliegende Fenster. Die Länge des Fensters wird durch den Parameter `nWindowLength` bestimmt.
- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.8 ST_CM_DiscreteClassification_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_DiscreteClassification_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels     : UDINT := 10;    (* Number of channels. *)
    nSubChannels  : UDINT := 0;     (* Number of subchannels. *)
    nMaxClasses   : UDINT := 3;     (* Number of configurable threshold classes. *)
END_STRUCT
END_TYPE
```

- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.
- `nSubChannels` definiert die Anzahl von unabhängigen Unterkanälen. Diese muss größer oder gleich Null sein.
- `nMaxClasses` ist die maximale Zahl der Klassen, die konfiguriert werden. Diese muss mindestens Eins sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.9 ST_CM_EmpiricalMoments_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_EmpiricalMoments_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels              : UDINT := 512;     (* Number of channels. *)
    nSubChannels           : UDINT := 0;       (* Number of subchannels. *)
    bPopulationEstimates   : BOOL := TRUE;     (* Apply Bessel's correction to results. *)
END_STRUCT
END_TYPE
```

- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.
- `nSubChannels` definiert die Anzahl von unabhängigen Unterkanälen. Diese muss größer oder gleich Null sein.
- `bPopulationEstimates` ist ein boolescher Parameter, der angibt, ob eine Bessel-Korrektur durchgeführt werden soll.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.10 ST_CM_Envelope_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_Envelope_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length     : UDINT := 512;     (* Length of FFT. *)
    nWindowLength   : UDINT := 400;     (* Length of analysis window. *)
    nChannels       : UDINT := 1;       (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Artefakte vermeiden

Der Wert von `nFFT_Length` muss mindestens gleich dem Wert von `nWindowLength` sein. Um Artefakte bei der Berechnung zu vermeiden, sollte `nFFT_Length` jedoch darüber hinaus mindestens 25% größer sein als `nWindowLength`. Eine Erhöhung der FFT -Länge gegenüber der Fensterlänge ist bei diesem Baustein sinnvoll, um Zirkuläres Aliasing zu vermeiden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.11 ST_CM_EnvelopeSpectrum_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_EnvelopeSpectrum_InitPars EXTENDS ST_CM_InitPars :
STRUCT
    nFFT_Length          : UDINT := 512;                            (* Length of FFT. *)
    nWindowLength        : UDINT := 400;                            (* Length of analysis window. *)
    bTransformToDecibel  : BOOL  := TRUE;                           (* Transform to decibel. *)
    fDecibelThreshold    : LREAL := cCM_MinArgLog10;                (* Minimum
argument of decadic logarithm for 64-bit IEEE 754 arithmetic. *)
    eWindowType          : E_CM_WindowType := eCM_HannWindow;       (* Window type. *)
    aWindowParameters    : T_CM_WindowParameters := [2.5,1,1,1,1];  (* Window parameters for specifi
c windows, e.g. FlatTop. *)
    nOverlap             : UDINT := -1;                             (* Size of overlap in samples. *
)
    eScalingType         : E_CM_ScalingType := eCM_DiracScaling;    (* Scaling type. *)
    nChannels            : UDINT := 1;                              (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT des Spektrums. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `bTransformToDecibel` ist ein boolescher Wert, der angibt, ob das Ergebnis der FFT in die Dezibel-Skala transformiert werden soll, entsprechend der Transformation x → 20 * log10(x).
- `fDecibelThreshold` ist ein sehr kleiner Fließkommawert größer als Null. Werte, die kleiner als diese Zahl sind, werden vor einer Transformation in die Dezibel-Skala durch diesen Wert ersetzt. (Zweck ist die Vermeidung von Wertbereichsfehlern. Der Logarithmus von Null ist nicht definiert und strebt für den Grenzwert kleiner Argumente gegen minus unendlich. Entsprechendes gilt für das Argument der Zahl Null, arg(0). Der kleinste mögliche Wert ist 2.3e-308, dies entspricht der Konstanten `cCM_MinArgLog10`.)
- `eWindowType` definiert die verwendete Fensterfunktion (vom Typ E_CM_WindowType [▶ 291]). Ein guter Standardwert ist der Fenstertyp `eCM_HannWindow`.
- `eWindowType` definiert die verwendete Fensterfunktion (vom Typ E_CM_WindowType [▶ 291]). Ein guter Standardwert ist der Fenstertyp `eCM_HannWindow`.
- `aWindowParameters` beinhaltet die freien Parameter ausgewählter Fensterfunktionen. Bei der Verwendung von `eCM_KaiserWindow` definiert der erste Eintrag den Parameter beta; wird das `eCM_FlatTopWindow` verwendet, werden alle Parameter genutzt. Siehe Abschnitt Fensterfunktionen [▶ 20].
- `nOverlap` definiert die Anzahl der überlappenden Samples. Diese muss größer oder gleich Null sein. Wird der Wert `cCM_OverlapRecommended` gewählt, so wird eine empfohlene Überlappung intern berechnet (siehe F_CM_CalculateRecommendedOverlap [▶ 287]); der Wert `cCM_OverlapInactive` deaktiviert den intern verwendeten Puffer und setzt den Wert Null.
- `eScalingType` ermöglicht eine Auswahl der verwendeten Skalierung (vom Typ E_CM_ScalingType [▶ 290]), falls eine absolute Skalierung benötigt wird. Standardwert ist `eCM_DiracScaling`. Bei der Auswahl der Skalierung sollte die Art des Signals, entweder deterministische Signale oder breitbandige Signale mit stochastischen Anteil, berücksichtigt werden; Beide Arten erfordern unterschiedliche Skalierungen.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Artefakte vermeiden

Der Wert von `nFFT_Length` muss mindestens gleich dem Wert von `nWindowLength` sein. Um Artefakte bei der Berechnung zu vermeiden, sollte `nFFT_Length` jedoch darüber hinaus mindestens 25% größer sein als `nWindowLength`. Eine Erhöhung der FFT -Länge gegenüber der Fensterlänge ist bei diesem Baustein sinnvoll, um Zirkuläres Aliasing zu vermeiden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.12 ST_CM_HistArray_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_HistArray_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels      : UDINT :=  512;   (* Number of channels. *)
    nSubChannels   : UDINT :=  0;     (* Number of subchannels. *)
    nBins          : UDINT :=  100;   (* Number of bins. *)
    fMinBinned     : LREAL := -120;   (* Minimum binned value. *)
    fMaxBinned     : LREAL :=  100;   (* Maximum binned value. *)
END_STRUCT
END_TYPE
```

- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.
- `nSubChannels` definiert die Anzahl von unabhängigen Unterkanälen. Diese muss größer oder gleich Null sein.
- `nBins` ist die Zahl der Bins des Histogramms. Sie muss mindestens Eins sein, in vielen Fällen sind Werte zwischen zehn und zwanzig eine sinnvolle Wahl. Die beiden speziellen Bins für Werte, die unterhalb von `fMinBinned` bzw. oberhalb von `fMaxBinned` liegen, sind in diesem Wert nicht mit enthalten.
- `fMinBinned` ist der untere Grenzwert, für den Samples in den regulären Histogramm-Bins gezählt werden.
- `fMaxBinned` ist der obere Grenzwert, für den Samples in den regulären Histogramm-Bins gezählt werden. `fMaxBinned` muss größer sein als `fMinBinned`.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.13 ST_CM_InstantaneousFrequency_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_InstantaneousFrequency_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length           : UDINT := 512;              (* Length of FFT. *)
    nWindowLength         : UDINT := 400;              (* Length of analysis window. *)
    fMagnitudeThreshold   : LREAL := cCM_MinArgDiv;    (* Minimum value for the numerical calculabili
ty. *)
    fSampleRate           : LREAL := 50000;            (* Sample rate in Hertz. *)
    nChannels             : UDINT := 1;                (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `fMagnitudeThreshold` definiert den Grenzwert für die numerische Berechenbarkeit der Momentanfrequenz. Der Grenzwert bezieht sich auf den Wert

  $$\mathrm{conj}\{\mathcal{H}\{x[n]\}\}\mathcal{H}\{x[n+1]\} < \mathrm{cCM_MinArgDiv}$$

- `fSampleRate` Abtastrate des eingehenden Zeitsignals. Der Wert wird genutzt zur Skalierung des Ergebnisses in Hz.

- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.14 ST_CM_InstantaneousPhase_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_InstantaneousPhase_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length      : UDINT := 512;                                    (* Length of FFT. *)
    nWindowLength    : UDINT := 400;                                    (* Length of analysis window
. *)
    eUnwrapMethod    : E_CM_UnwrapMethod := eCM_ThresholdUnwrapping;    (* Unwrap method for phase v
alues. *)
    fPhaseThreshold  : LREAL := cCM_MinArgDiv;                          (* Minimum value for calcula
ting the instantaneous phase. *)
    nChannels        : UDINT := 1;                                      (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `eUnwrapMethod` definiert die verwendete Methode zum *phase-unwrapping* bezüglich der Phase in Vielfachen von `2 PI` (siehe E_CM_UnwrapMethod [▶ 291]).
- `fPhaseThreshold` Grenzwert zur Berechnung der Momentanphase. Der Wert bezieht sich auf die Einhüllende des Signals. Interpretation: Bei zu geringem Signalpegel ist die Berechnung der Phase numerisch zu unsicher und nicht verlässlich auswertbar. Als Phase wird dann eine 0 ausgegeben.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.15 ST_CM_IntegratedRMS_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_IntegratedRMS_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length           : UDINT := 512;                              (* Length of FFT. *)
    nWindowLength         : UDINT := 400;                              (* Length of analysis window.
 *)
    fSampleRate           : LREAL := 20000;                            (* Sample rate in Hertz. *)
    fLowerFrequencyLimit  : LREAL := 20.0;                             (* Lower limit of frequency b
and in Hertz. *)
    fUpperFrequencyLimit  : LREAL := 1000.0;                           (* Upper limit of frequency b
and in Hertz. *)
    nMaxBands             : UDINT := 1;                                (* Maximum number of frequenc
y bands. *)
    nOrder                : UDINT := 2;                                (* Maximum order of integrati
on: 0 = acceleration, 1 = velocity, 2 = place. *)
    nChannels             : UDINT := 2;                                (* Number of channels. *)
    eWindowType           : E_CM_WindowType := eCM_HannWindow;         (* Window type. *)
    aWindowParameters     : T_CM_WindowParameters := [2.5,1,1,1,1];    (* Window parameters for spec
ific windows, e.g. FlatTop. *)
    nOverlap              : UDINT := -1;                               (* Size of overlap in samples
. *)
    bTransformToDecibel   : BOOL := TRUE;                              (* Transform to decibel. *)
    fDecibelThreshold     : LREAL := cCM_MinArgLog10;                  (* Minimum argument of decadi
```

```
c logarithm for 64- bit IEEE 754 arithmetic. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `fSampleRate` Abtastrate des eingehenden Zeitsignals. Der Wert wird genutzt zur Skalierung des Ergebnisses in Hz.
- `fLowerFrequencyLimit` Untere Grenze des berücksichtigten Frequenzintervalls. Die untere Grenzfrequenz muss mindestens so groß sein wie die Abtastrate geteilt durch die FFT-Länge.
- `fUpperFrequencyLimit` Obere Grenze des berücksichtigten Frequenzintervalls. Die obere Grenzfrequenz darf höchstens so groß sein wie die halbe Abtastrate und muss größer als die untere Grenzfrequenz sein.
- `nMaxBands` gibt die maximale Anzahl an Frequenzbändern an, für die der RMS-Wert berechnet wird.
- `nOrder` ist die maximale Ordnung der Integration. Dies muss eine ganze Zahl zwischen Null und Zwei sein. Die Zahl der ermittelten Werte pro Kanal ist (nOrder+1).
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.
- `eWindowType` definiert die verwendete Fensterfunktion (vom Typ E_CM_WindowType [▶ 291]). Ein guter Standardwert ist der Fenstertyp `eCM_HannWindow`.
- `aWindowParameters` beinhaltet die freien Parameter ausgewählter Fensterfunktionen. Bei der Verwendung von `eCM_KaiserWindow` definiert der erste Eintrag den Parameter beta; wird das `eCM_FlatTopWindow` verwendet, werden alle Parameter genutzt. Siehe Abschnitt Fensterfunktionen [▶ 20].
- `nOverlap` definiert die Anzahl der überlappenden Samples. Diese muss größer oder gleich Null sein. Wird der Wert `cCM_OverlapRecommended` gewählt, so wird eine empfohlene Überlappung intern berechnet (siehe F_CM_CalculateRecommendedOverlap [▶ 287]); der Wert `cCM_OverlapInactive` deaktiviert den intern verwendeten Puffer und setzt den Wert Null.
- `bTransformToDecibel` ist ein boolescher Wert, der angibt, ob das Ergebnis der FFT in die Dezibel-Skala transformiert werden soll, entsprechend der Transformation x → 20 * log10(x).
- `fDecibelThreshold` ist ein sehr kleiner Fließkommawert größer als Null. Werte, die kleiner als diese Zahl sind, werden vor einer Transformation in die Dezibel-Skala durch diesen Wert ersetzt. (Zweck ist die Vermeidung von Wertbereichsfehlern. Der Logarithmus von Null ist nicht definiert und strebt für den Grenzwert kleiner Argumente gegen minus unendlich. Entsprechendes gilt für das Argument der Zahl Null, arg(0). Der kleinste mögliche Wert ist 2.3e-308, dies entspricht der Konstanten `cCM_MinArgLog10`.)

Fensterlänge beachten

Der Wert von `nWindowLength` muss kleiner oder gleich dem Wert von `nFFT_Length` sein. Die Länge der FFT kann sich an der benötigten Frequenzauflösung orientieren. Typischerweise wird oft ein Wert von ca. 4/5 der FFT-Länge als Fensterlänge verwendet.

Wenn `nFFT_Length` größer ist als `nWindowLength`, wird die Frequenzauflösung der FFT (und damit auch die Länge des Vektors der Rückgabewerte) vergrößert. Die Differenz der Länge wird vor der Fouriertransformation mit Nullen aufgefüllt. Dies kann sinnvoll sein, um eine höhere Frequenzauflösung zu erreichen oder, z.B. bei der Berechnung mit Rücktransformation in den Zeitbereich, zirkuläres Aliasing zu vermeiden. Das Ergebnis enthält trotz der höheren Frequenzauflösung allerdings nicht mehr Information.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.16 ST_CM_MagnitudeSpectrum_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_MagnitudeSpectrum_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length          : UDINT := 512;                              (* Length of FFT. *)
    nWindowLength        : UDINT := 400;                              (* Length of analysis window.
*)
    fDecibelThreshold    : LREAL := cCM_MinArgLog10;                  (* Minimum argument of decadic
 logarithm for 64- bit IEEE 754 arithmetic. *)
    bTransformToDecibel  : BOOL := TRUE;                              (* Transform to decibel. *)
    eWindowType          : E_CM_WindowType := eCM_HannWindow;         (* Window type. *)
    aWindowParameters    : T_CM_WindowParameters := [2.5,1,1,1,1];    (* Window parameters for speci
fic windows, e.g. FlatTop. *)
    nOverlap             : UDINT := -1;                               (* Size of overlap in samples.
 *)
    eScalingType         : E_CM_ScalingType := eCM_DiracScaling;      (* Scaling type. *)
    nChannels            : UDINT := 1;                                (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `fDecibelThreshold` ist ein sehr kleiner Fließkommawert größer als Null. Werte, die kleiner als diese Zahl sind, werden vor einer Transformation in die Dezibel-Skala durch diesen Wert ersetzt. (Zweck ist die Vermeidung von Wertbereichsfehlern. Der Logarithmus von Null ist nicht definiert und strebt für den Grenzwert kleiner Argumente gegen minus unendlich. Entsprechendes gilt für das Argument der Zahl Null, arg(0). Der kleinste mögliche Wert ist 2.3e-308, dies entspricht der Konstanten `cCM_MinArgLog10`.)
- `bTransformToDecibel` ist ein boolescher Wert, der angibt, ob das Ergebnis der FFT in die Dezibel-Skala transformiert werden soll, entsprechend der Transformation x → 20 * log10(x).
- `eWindowType` definiert die verwendete Fensterfunktion (vom Typ E_CM_WindowType [▶ 291]). Ein guter Standardwert ist der Fenstertyp `eCM_HannWindow`.
- `aWindowParameters` beinhaltet die freien Parameter ausgewählter Fensterfunktionen. Bei der Verwendung von `eCM_KaiserWindow` definiert der erste Eintrag den Parameter beta; wird das `eCM_FlatTopWindow` verwendet, werden alle Parameter genutzt. Siehe Abschnitt Fensterfunktionen [▶ 20].
- `nOverlap` definiert die Anzahl der überlappenden Samples. Diese muss größer oder gleich Null sein. Wird der Wert `cCM_OverlapRecommended` gewählt, so wird eine empfohlene Überlappung intern berechnet (siehe F_CM_CalculateRecommendedOverlap [▶ 287]); der Wert `cCM_OverlapInactive` deaktiviert den intern verwendeten Puffer und setzt den Wert Null.
- `eScalingType` ermöglicht eine Auswahl der verwendeten Skalierung (vom Typ E_CM_ScalingType [▶ 290]), falls eine absolute Skalierung benötigt wird. Standardwert ist `eCM_DiracScaling`. Bei der Auswahl der Skalierung sollte die Art des Signals, entweder deterministische Signale oder breitbandige Signale mit stochastischen Anteil, berücksichtigt werden; Beide Arten erfordern unterschiedliche Skalierungen.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Fensterlänge beachten

Der Wert von `nWindowLength` muss kleiner oder gleich dem Wert von `nFFT_Length` sein. Die Länge der FFT kann sich an der benötigten Frequenzauflösung orientieren. Typischerweise wird oft ein Wert von ca. 4/5 der FFT-Länge als Fensterlänge verwendet.

Wenn `nFFT_Length` größer ist als `nWindowLength`, wird die Frequenzauflösung der FFT (und damit auch die Länge des Vektors der Rückgabewerte) vergrößert. Die Differenz der Länge wird vor der Fouriertransformation mit Nullen aufgefüllt. Dies kann sinnvoll sein, um eine höhere Frequenzauflösung zu erreichen oder, z.B. bei der Berechnung mit Rücktransformation in den Zeitbereich, zirkuläres Aliasing zu vermeiden. Das Ergebnis enthält trotz der höheren Frequenzauflösung allerdings nicht mehr Information.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.17 ST_CM_MeanStressCorrection_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_MeanStressCorrection_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    eMeanStressCorrection : E_CM_MeanStressCorrection
                            := E_CM_MeanStressCorrection.eCM_Goodman;  (* Mean stress correction typ
e, e.g. Goodman or Gerber correction. *)
    nBins                 : UDINT := 1;                                (* Number of bins for the str
ess levels. *)
    fBinWidth             : LREAL := 1.0;                              (* Width of the stress bins.
*)
    nBinsMean             : UDINT := 1;                                (* Number of bins for the mea
n levels. *)
    fBinWidthMean         : LREAL := 1.0;                              (* Width of the mean stress b
ins. *)
    fUTS                  : LREAL := 700.0;                            (* Ultimate tensile strength
of material. *)
    nChannels             : UDINT := 1;                                (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `eMeanStressCorrection` definiert den zu Verwendenden Korrekturmodus. Es stehen die Korrektur nach Goodman und Gerber zur Verfügung sowie die Möglichkeit keine Korrektur durch zu führen.
- `nBins` definiert die Anzahl der Bins für das Stresslevel. Die Gesamtzahl der Bins ist `nBins + 2`, der Wert muss mindestens eins sein. Das erste Bin jeder Zeile enthält die Stresslevels kleiner oder gleich dem definierten Minimum, analog fallen alle Werte größer als dem definierten Maximum in das letzte Bin jeder Zeile.
- `fBinWidth` definiert die Breite der Behälter für das Stresslevel. Der Parameter ist passend zu der Dimension der eingehenden Zählmatrix zu wählen, d.h. im Falle der Korrektur eines Outputs aus dem Baustein FB_CMA_RainflowCounting ist hier der Wert `(fMaxStress - fMinStress)/nBins` zu wählen.
- `nBinsMean` definiert die Anzahl der Bins für die Mittelwerte. Der Wert muss mindestens eins sein. Zwei gesonderte Bins werden in der ersten/letzten Spalte der *Halfcycle Count Matrix* hinzugefügt, in der Mittelwerte kleiner oder gleich bzw. größer als die definierten Schranken gehalten werden.
- `fBinWidthMean` definiert die Breite der Behälter für die Mittelwerte der Stresszyklen. Der Parameter ist passend zu der Dimension der eingehenden Zählmatrix zu wählen, d.h. im Falle der Korrektur eines Outputs aus dem Baustein FB_CMA_RainflowCounting ist hier der Wert `(fMaxMean - fMinMean)/nBinsMean` zu wählen.
- `fUTS` definiert die Zugfestigkeit des überwachten Materials
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.18 ST_CM_MinersRule_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_MinersRule_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nBins       : UDINT := 64;   (* Number of bins for the stress levels. *)
    nBinsMean   : UDINT := 64;   (* Number of bins for the mean levels. *)
    fDamage     : LREAL := 0.0;  (* Additional constant damage *)
    nCycles     : ULINT := 0;    (* Initial number of cycles counted *)
```

```
    nChannels   : UDINT := 1;     (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nBins` definiert die Anzahl der Bins für das Stresslevel. Die Gesamtzahl der Bins ist `nBins + 2`, der Wert muss mindestens eins sein. Das erste Bin jeder Zeile enthält die Stresslevels kleiner oder gleich dem definierten Minimum, analog fallen alle Werte größer als dem definierten Maximum in das letzte Bin jeder Zeile.
- `nBinsMean` definiert die Anzahl der Bins für die Mittelwerte. Der Wert muss mindestens eins sein. Zwei gesonderte Bins werden in der ersten/letzten Spalte der *Halfcycle Count Matrix* hinzugefügt, in der Mittelwerte kleiner oder gleich bzw. größer als die definierten Schranken gehalten werden.
- `fDamage` definiert den konstanten Schaden ab Beginn. Der Gesamtschaden berechnet sich somit aus der Summe aus `fDamage` und dem akkumulierten Schaden bezüglich der konfigurierten Wöhlerkurve.
- `nCycles` definiert den initialen Wert der gezählten Halbzyklen zu Beginn. Die gesamte Anzahl an Zyklen berechnet sich aus der Summe aus `nCycles` und der aktuellen Anzahlt der Eingangsdaten.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.19 ST_CM_MomentCoefficients_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_MomentCoefficients_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels             : UDINT := 512;                                        (* Number of channel
s. *)
    nSubChannels          : UDINT := 0;                                          (* Number of subchan
nels. *)
    nOrder                : E_CM_MCoefOrder := E_CM_MCoefOrder.eCM_Kurtosis;     (* Maximum order of
the moment coefficients that are calculated. *)
    bPopulationEstimates  : BOOL := FALSE;                                       (* Apply Bessel's co
rrection to results. *)
    bKurtosisExcess       : BOOL := TRUE;                                        (* Flag if kurtosis
is reduced by 3. *)
END_STRUCT
END_TYPE
```

- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.
- `nSubChannels` definiert die Anzahl von unabhängigen Unterkanälen. Diese muss größer oder gleich Null sein.
- `nOrder` ist die maximale Ordnung der Momentenkoeffizienten (E_CM_MCoefOrder [▶ 290]), die berechnet werden. Dies muss eine ganze Zahl zwischen Eins und Vier sein. Die Ordnungsnummern sind: 0 = Zähler, 1 = Mittelwert, 2 = Standardabweichung, 3 = Schiefe, 4 = Exzess Kurtosis. Die Zahl der ermittelten Koeffizienten ist (nOrder+1).
- `bPopulationEstimates` ist ein boolescher Wert der angibt, ob die jeweils Bessel'sche Korrektur zu Sample Variance (Stichproben-Standardabweichung), Schiefe und Exzess angewandt wird.
- `bKurtosisExcess` ist ein boolscher Wert der angibt, ob der Exzess um den Wert der Wölbung einer Normalverteilung reduziert wird.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.20 ST_CM_MovingMoments_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_MovingMoments_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels             : UDINT := 1;      (* Number of channels. *)
    nSubChannels          : UDINT := 0;      (* Number of subchannels. *)
    nInLength             : UDINT := 100;    (* Length of input data array. *)
    nPopulation           : UDINT := 100;    (* Size of the population. *)
    bPopulationEstimates  : BOOL := TRUE;    (* Apply Bessel's correction to results. *)
END_STRUCT
END_TYPE
```

- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.
- `nSubChannels` definiert die Anzahl von unabhängigen Unterkanälen. Diese muss größer oder gleich Null sein.
- `nInLength` definiert die Länge des Eingangsdatenarrays.
- `nPopulation` definiert die Größe des internen Datenpuffers für die Berechnung des gleitenden Moments.
- `bPopulationEstimates` ist ein boolescher Wert der angibt, ob die jeweils Bessel'sche Korrektur zu Sample Variance (Stichproben-Standardabweichung), Schiefe und Exzess angewandt wird.
- `bKurtosisExcess` ist ein boolscher Wert der angibt, ob der Exzess um den Wert der Wölbung einer Normalverteilung reduziert wird.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.21 ST_CM_MultiBandRMS_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_MultiBandRMS_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length           : UDINT := 512;                              (* Length of FFT. *)
    nWindowLength         : UDINT := 400;                              (* Length of analysis window.
 *)
    fSampleRate           : LREAL := 20000;                            (* Sample rate in Hertz. *)
    nMaxBands             : UDINT := 10;                               (* Maximum number of frequenc
y bands. *)
    nChannels             : UDINT := 10;                               (* Number of channels. *)
    eWindowType           : E_CM_WindowType := eCM_HannWindow;         (* Window type. *)
    aWindowParameters     : T_CM_WindowParameters := [2.5,1,1,1,1];    (* Window parameters for spec
ific windows, e.g. FlatTop. *)
    nOverlap              : UDINT := -1;                               (* Size of overlap in samples
. *)
    bTransformToDecibel   : BOOL := TRUE;                              (* Transform to decibel. *)
    fDecibelThreshold     : LREAL := cCM_MinArgLog10;                  (* Minimum argument of decadi
c logarithm for 64- bit IEEE 754 arithmetic. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `fSampleRate` Abtastrate des eingehenden Zeitsignals. Der Wert wird genutzt zur Skalierung des Ergebnisses in Hz.
- `nMaxBands` gibt die maximale Anzahl an Frequenzbändern an, für die der RMS-Wert berechnet wird.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

- `eWindowType` definiert die verwendete Fensterfunktion (vom Typ E_CM_WindowType [▶ 291]). Ein guter Standardwert ist der Fenstertyp `eCM_HannWindow`.
- `aWindowParameters` beinhaltet die freien Parameter ausgewählter Fensterfunktionen. Bei der Verwendung von `eCM_KaiserWindow` definiert der erste Eintrag den Parameter beta; wird das `eCM_FlatTopWindow` verwendet, werden alle Parameter genutzt. Siehe Abschnitt Fensterfunktionen [▶ 20].
- `nOverlap` definiert die Anzahl der überlappenden Samples. Diese muss größer oder gleich Null sein. Wird der Wert `cCM_OverlapRecommended` gewählt, so wird eine empfohlene Überlappung intern berechnet (siehe F_CM_CalculateRecommendedOverlap [▶ 287]); der Wert `cCM_OverlapInactive` deaktiviert den intern verwendeten Puffer und setzt den Wert Null.
- `bTransformToDecibel` ist ein boolescher Wert, der angibt, ob das Ergebnis der FFT in die Dezibel-Skala transformiert werden soll, entsprechend der Transformation x → 20 * log10(x).
- `fDecibelThreshold` ist ein sehr kleiner Fließkommawert größer als Null. Werte, die kleiner als diese Zahl sind, werden vor einer Transformation in die Dezibel-Skala durch diesen Wert ersetzt. (Zweck ist die Vermeidung von Wertbereichsfehlern. Der Logarithmus von Null ist nicht definiert und strebt für den Grenzwert kleiner Argumente gegen minus unendlich. Entsprechendes gilt für das Argument der Zahl Null, arg(0). Der kleinste mögliche Wert ist 2.3e-308, dies entspricht der Konstanten `cCM_MinArgLog10`.)

Fensterlänge beachten

Der Wert von `nWindowLength` muss kleiner oder gleich dem Wert von `nFFT_Length` sein. Die Länge der FFT kann sich an der benötigten Frequenzauflösung orientieren. Typischerweise wird oft ein Wert von ca. 4/5 der FFT-Länge als Fensterlänge verwendet.

Wenn `nFFT_Length` größer ist als `nWindowLength`, wird die Frequenzauflösung der FFT (und damit auch die Länge des Vektors der Rückgabewerte) vergrößert. Die Differenz der Länge wird vor der Fouriertransformation mit Nullen aufgefüllt. Dies kann sinnvoll sein, um eine höhere Frequenzauflösung zu erreichen oder, z.B. bei der Berechnung mit Rücktransformation in den Zeitbereich, zirkuläres Aliasing zu vermeiden. Das Ergebnis enthält trotz der höheren Frequenzauflösung allerdings nicht mehr Information.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.22 ST_CM_Normalization_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_Normalization_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels         : UDINT := 1;    (* Number of channels. *)
    nSubChannels      : UDINT := 0;    (* Number of subchannels. *)
    nInLength         : UDINT := 256;  (* Length of input data array. *)
    fParameter_s      : LREAL := 1.0;  (* Parameter s for normalization. *)
    nParameter_p      : UDINT := 1;    (* Parameter p for normalization. *)
    nParameter_q      : UDINT := 1;    (* Parameter q for normalization. *)
    nWorkingDimension : UDINT := 0;    (* Dimension index of MultiArray for normalization. *)
    bAbsoluteValues   : BOOL := TRUE;  (* Use absolute values in calculation. *)
END_STRUCT
END_TYPE
```

- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.
- `nSubChannels` definiert die Anzahl von unabhängigen Unterkanälen. Diese muss größer oder gleich Null sein.
- `nInLength` ist die Länge des Eingangs-Arrays.
- `fParameter_s` definiert den Skalierungsfaktor der Normalisierung.
- `fParameter_p` definiert den Exponenten der Summanden in der Normalisierung.
- `fParameter_q` definiert den Quotienten des Äußeren Exponenten in der Normalisierung.

- `nWorkingDimension` gibt die Null basierte Dimension des Eingangsmultiarrays an, über welche summiert/normiert wird.
- `bAbsoluteValues` ist ein Flag mit dem gewählt werden kann, ob die einzelnen Werte im Betrag aufsummiert werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

5.3.14.23 ST_CM_OrderPowerSpectrum_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_OrderPowerSpectrum_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    fSampleRateSignal     : LREAL := 1000;                              (* Sample rate in Hertz of vib
ration signal. *)
    fSampleRatePosition   : LREAL := 1000;                              (* Sample rate in Hertz of pos
itions. *)
    fPositionScaling      : LREAL := 1;                                 (* One rotation needs to be re
presented by 1. *)
    fMaxRPM               : LREAL := 1800;                              (* Max. rotations per minute.
*)
    nFFT_Length           : UDINT := 512;                               (* Length of FFT. *)
    nWindowLength         : UDINT := 400;                               (* Length of analysis window.
*)
    fDecibelThreshold     : LREAL := cCM_MinArgLog10;                   (* Minimum argument of decadic
 logarithm for 64- bit IEEE 754 arithmetic. *)
    bTransformToDecibel   : BOOL := TRUE;                               (* Transform to decibel. *)
    eWindowType           : E_CM_WindowType := eCM_HannWindow;          (* Window type. *)
    aWindowParameters     : T_CM_WindowParameters := [2.5,1,1,1,1];     (* Window parameters for speci
fic windows, e.g. FlatTop. *)
    nOverlap              : UDINT := -1;                                (* Size of overlap in samples.
 *)
    eScalingType          : E_CM_ScalingType := eCM_DiracScaling;       (* Scaling type. *)
    nChannels             : UDINT := 1;                                 (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `fSampleRateSignal` definiert die Abtastrate in Hertz des eingehenden Vibrationssignals.
- `fSampleRatePosition` definiert die Abtastrate in Hertz des eingehenden Positionssignals. Diese kann ungleich `fSampleRateSignal` sein.
- `fPositionScaling` ist der Skalierungsfaktor für das Positionssignal. Eine vollständige Rotation muss durch diesen Wert repräsentiert werden. Ist das Positionssignal in Grad skaliert, so ist `fPositionScaling = 360` zu setzen. Ist das Positionssignal so skaliert, dass eine volle Umdrehung der Welle den Wert 2*pi ergibt, ist `fPositionScaling = 2 * pi`.
- `fMaxRPM` ist die maximale Anzahl an Rotationen pro Minute. Dieser Parameter definiert mit der `fSampleRateSignal` die maximal auflösbare Ordnung des Ordnungsspektrums.

Alle weiteren Parameter erfolgen analog zur Parametrierung des Power Spektrums, vgl. ST_CM_PowerSpectrum_InitPars [▶ 313].

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `fDecibelThreshold` ist ein sehr kleiner Fließkommawert größer als Null. Werte, die kleiner als diese Zahl sind, werden vor einer Transformation in die Dezibel-Skala durch diesen Wert ersetzt. (Zweck ist die Vermeidung von Wertbereichsfehlern. Der Logarithmus von Null ist nicht definiert und strebt für den Grenzwert kleiner Argumente gegen minus unendlich. Entsprechendes gilt für das Argument der Zahl Null, arg(0). Der kleinste mögliche Wert ist 2.3e-308, dies entspricht der Konstanten `cCM_MinArgLog10`.)

- `bTransformToDecibel` ist ein boolescher Wert, der angibt, ob das Ergebnis der FFT in die Dezibel-Skala transformiert werden soll, entsprechend der Transformation x → 20 * log10(x).
- `eWindowType` definiert die verwendete Fensterfunktion (vom Typ E_CM_WindowType [▶ 291]). Ein guter Standardwert ist der Fenstertyp `eCM_HannWindow`.
- `aWindowParameters` beinhaltet die freien Parameter ausgewählter Fensterfunktionen. Bei der Verwendung von `eCM_KaiserWindow` definiert der erste Eintrag den Parameter beta; wird das `eCM_FlatTopWindow` verwendet, werden alle Parameter genutzt. Siehe Abschnitt Fensterfunktionen [▶ 20].
- `nOverlap` definiert die Anzahl der überlappenden Samples. Diese muss größer oder gleich Null sein. Wird der Wert `cCM_OverlapRecommended` gewählt, so wird eine empfohlene Überlappung intern berechnet (siehe F_CM_CalculateRecommendedOverlap [▶ 287]); der Wert `cCM_OverlapInactive` deaktiviert den intern verwendeten Puffer und setzt den Wert Null.
- `eScalingType` ermöglicht eine Auswahl der verwendeten Skalierung (vom Typ E_CM_ScalingType [▶ 290]), falls eine absolute Skalierung benötigt wird. Standardwert ist `eCM_DiracScaling`. Bei der Auswahl der Skalierung sollte die Art des Signals, entweder deterministische Signale oder breitbandige Signale mit stochastischen Anteil, berücksichtigt werden; Beide Arten erfordern unterschiedliche Skalierungen.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.24 ST_CM_PowerCepstrum_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_PowerCepstrum_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length         : UDINT := 512;                                (* Length of FFT. *)
    nWindowLength       : UDINT := 400;                                (* Length of analysis window. *)
    eWindowType         : E_CM_WindowType := eCM_HannWindow;           (* Window type. *)
    aWindowParameters   : T_CM_WindowParameters := [2.5,1,1,1,1];      (* Window parameters for specifi
c windows, e.g. FlatTop. *)
    nOverlap            : UDINT := -1;                                 (* Size of overlap in samples. *
)
    fLogThreshold       : LREAL := cCM_MinArgLogN;                     (* Minimum argument of decadic l
ogarithm for 64- bit IEEE 754 arithmetic
    eScalingType        : E_CM_ScalingType := eCM_DiracScaling;        (* Scaling type. *)
    nChannels           : UDINT := 1;                                  (* Number of channels. *)
 *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `eWindowType` definiert die verwendete Fensterfunktion (vom Typ E_CM_WindowType [▶ 291]). Ein guter Standardwert ist der Fenstertyp `eCM_HannWindow`.
- `aWindowParameters` beinhaltet die freien Parameter ausgewählter Fensterfunktionen. Bei der Verwendung von `eCM_KaiserWindow` definiert der erste Eintrag den Parameter beta; wird das `eCM_FlatTopWindow` verwendet, werden alle Parameter genutzt. Siehe Abschnitt Fensterfunktionen [▶ 20].
- `nOverlap` definiert die Anzahl der überlappenden Samples. Diese muss größer oder gleich Null sein. Wird der Wert `cCM_OverlapRecommended` gewählt, so wird eine empfohlene Überlappung intern berechnet (siehe F_CM_CalculateRecommendedOverlap [▶ 287]); der Wert `cCM_OverlapInactive` deaktiviert den intern verwendeten Puffer und setzt den Wert Null.

- `fLogThreshold` ist ein sehr kleiner Fließkommawert größer als Null. Der kleinste mögliche Wert ist 3.75e-324, dies entspricht der Konstanten `cCM_MinArgLogN` .
  Spektralwerte, deren Absolutbetrag kleiner als diese Zahl ist, werden vor der Logarithmierung des Spektrums durch diesen Wert ersetzt. Zweck ist die Vermeidung von Wertbereichsfehlern. Der Logarithmus von Null ist nicht definiert und strebt für den Grenzwert kleiner Argumente gegen minus unendlich. Entsprechendes gilt für das Argument der Zahl Null, arg(0).
- `eScalingType` ermöglicht eine Auswahl der verwendeten Skalierung (vom Typ E_CM_ScalingType [▶ 290]), falls eine absolute Skalierung benötigt wird. Standardwert ist `eCM_DiracScaling`. Bei der Auswahl der Skalierung sollte die Art des Signals, entweder deterministische Signale oder breitbandige Signale mit stochastischen Anteil, berücksichtigt werden; Beide Arten erfordern unterschiedliche Skalierungen.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Fensterlänge beachten

Der Wert von `nWindowLength` muss kleiner oder gleich dem Wert von `nFFT_Length` sein. Die Länge der FFT kann sich an der benötigten Frequenzauflösung orientieren. Typischerweise wird oft ein Wert von ca. 3/4 der FFT-Länge als Fensterlänge verwendet.

Wenn `nFFT_Length` größer ist als `nWindowLength`, wird die Frequenzauflösung der FFT (und damit auch die Länge des Vektors der Rückgabewerte) vergrößert. Die Differenz der Länge wird vor der Fouriertransformation mit Nullen aufgefüllt. Dies kann sinnvoll sein, um eine höhere Frequenzauflösung zu erreichen oder, z.B. bei der Berechnung mit Rücktransformation in den Zeitbereich, zirkuläres Aliasing zu vermeiden. Das Ergebnis enthält trotz der höheren Frequenzauflösung allerdings nicht mehr Information.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.25 ST_CM_PowerSpectrum_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_PowerSpectrum_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length          : UDINT := 512;                              (* Length of FFT. *)
    nWindowLength        : UDINT := 400;                              (* Length of analysis window.
*)
    fDecibelThreshold    : LREAL := cCM_MinArgLog10;                  (* Minimum argument of decadic
 logarithm for 64- bit IEEE 754 arithmetic. *)
    bTransformToDecibel  : BOOL := TRUE;                              (* Transform to decibel. *)
    eWindowType          : E_CM_WindowType := eCM_HannWindow;         (* Window type. *)
    aWindowParameters    : T_CM_WindowParameters := [2.5,1,1,1,1];    (* Window parameters for speci
fic windows, e.g. FlatTop. *)
    nOverlap             : UDINT := -1;                               (* Size of overlap in samples.
 *)
    eScalingType         : E_CM_ScalingType := eCM_DiracScaling;      (* Scaling type. *)
    nChannels            : UDINT := 1;                                (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `fDecibelThreshold` ist ein sehr kleiner Fließkommawert größer als Null. Werte, die kleiner als diese Zahl sind, werden vor einer Transformation in die Dezibel-Skala durch diesen Wert ersetzt. (Zweck ist die Vermeidung von Wertbereichsfehlern. Der Logarithmus von Null ist nicht definiert und strebt für den Grenzwert kleiner Argumente gegen minus unendlich. Entsprechendes gilt für das Argument der Zahl Null, arg(0). Der kleinste mögliche Wert ist 2.3e-308, dies entspricht der Konstanten `cCM_MinArgLog10`.)

- `bTransformToDecibel` ist ein boolescher Wert, der angibt, ob das Ergebnis der FFT in die Dezibel-Skala transformiert werden soll, entsprechend der Transformation x → 20 * log10(x).
- `eWindowType` definiert die verwendete Fensterfunktion (vom Typ E_CM_WindowType [▶ 291]). Ein guter Standardwert ist der Fenstertyp `eCM_HannWindow`.
- `aWindowParameters` beinhaltet die freien Parameter ausgewählter Fensterfunktionen. Bei der Verwendung von `eCM_KaiserWindow` definiert der erste Eintrag den Parameter beta; wird das `eCM_FlatTopWindow` verwendet, werden alle Parameter genutzt. Siehe Abschnitt Fensterfunktionen [▶ 20].
- `nOverlap` definiert die Anzahl der überlappenden Samples. Diese muss größer oder gleich Null sein. Wird der Wert `cCM_OverlapRecommended` gewählt, so wird eine empfohlene Überlappung intern berechnet (siehe F_CM_CalculateRecommendedOverlap [▶ 287]); der Wert `cCM_OverlapInactive` deaktiviert den intern verwendeten Puffer und setzt den Wert Null.
- `eScalingType` ermöglicht eine Auswahl der verwendeten Skalierung (vom Typ E_CM_ScalingType [▶ 290]), falls eine absolute Skalierung benötigt wird. Standardwert ist `eCM_DiracScaling`. Bei der Auswahl der Skalierung sollte die Art des Signals, entweder deterministische Signale oder breitbandige Signale mit stochastischen Anteil, berücksichtigt werden; Beide Arten erfordern unterschiedliche Skalierungen.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Fensterlänge beachten

Der Wert von `nWindowLength` muss kleiner oder gleich dem Wert von `nFFT_Length` sein. Die Länge der FFT kann sich an der benötigten Frequenzauflösung orientieren. Typischerweise wird oft ein Wert von ca. 4/5 der FFT-Länge als Fensterlänge verwendet.

Wenn `nFFT_Length` größer ist als `nWindowLength`, wird die Frequenzauflösung der FFT (und damit auch die Länge des Vektors der Rückgabewerte) vergrößert. Die Differenz der Länge wird vor der Fouriertransformation mit Nullen aufgefüllt. Dies kann sinnvoll sein, um eine höhere Frequenzauflösung zu erreichen oder, z.B. bei der Berechnung mit Rücktransformation in den Zeitbereich, zirkuläres Aliasing zu vermeiden. Das Ergebnis enthält trotz der höheren Frequenzauflösung allerdings nicht mehr Information.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.26 ST_CM_Quantiles_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_Quantiles_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels       : UDINT := 512;     (* Number of channels. *)
    nSubChannels    : UDINT := 0;       (* Number of subchannels. *)
    fMinBinned      : LREAL := -120;    (* Minimum binned value. *)
    fMaxBinned      : LREAL := 100;     (* Maximum binned value. *)
    nBins           : UDINT := 100;     (* Number of bins. *)
    nMaxQuantiles   : UDINT := 10;      (* Maximum number of quantiles. *)
END_STRUCT
END_TYPE
```

- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.
- `nSubChannels` definiert die Anzahl von unabhängigen Unterkanälen. Diese muss größer oder gleich Null sein.
- `fMinBinned` ist der untere Grenzwert, für den Samples in den regulären Histogramm-Bins gezählt werden.
- `fMaxBinned` ist der obere Grenzwert, für den Samples in den regulären Histogramm-Bins gezählt werden. `fMaxBinned` muss größer sein als `fMinBinned`.

- `nBins` ist die Zahl der Bins des Histogramms. Sie muss mindestens Eins sein, in vielen Fällen sind Werte zwischen zehn und zwanzig eine sinnvolle Wahl. Die beiden speziellen Bins für Werte, die unterhalb von `fMinBinned` bzw. oberhalb von `fMaxBinned` liegen, sind in diesem Wert nicht mit enthalten.
- `nMaxQuantiles` ist die Zahl der zu berechnenden Quantile für jeden Kanal. Dies muss eine ganze Zahl größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.27 ST_CM_RainflowCounting_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_RainflowCounting_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nBins       : UDINT := 64;       (* Number of bins for the stress levels. *)
    fminStress  : LREAL := -40.0;    (* Lower limit of the stress bins. *)
    fmaxStress  : LREAL := 40.0;     (* Upper limit of the stress bins. *)
    nBinsMean   : UDINT := 64;       (* Number of bins for the mean levels. *)
    fminMean    : LREAL := -40.0;    (* Lower limit of bins for mean stress values. *)
    fmaxMean    : LREAL := 40.0;     (* Upper limit of bins for mean stress values. *)
    nChannels   : UDINT := 1;        (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nBins` definiert die Anzahl der Bins für das Stresslevel. Die Gesamtzahl der Bins ist `nBins + 2`, der Wert muss mindestens eins sein. Das erste Bin jeder Zeile enthält die Stresslevels kleiner oder gleich dem definierten Minimum, analog fallen alle Werte größer als dem definierten Maximum in das letzte Bin jeder Zeile.
- `fminStress` definiert die untere Schranke des Stresslevels. Alle Werte, die kleiner oder gleich diesem Wert sind, werden in einem Bin akkumuliert.
- `fmaxStress` definiert die obere Schranke des Stresslevels. Werte, die das Maximum übersteigen, werden in das letzte Bin gezählt.
- `nBinsMean` definiert die Anzahl der Bins für die Mittelwerte. Der Wert muss mindestens eins sein. Zwei gesonderte Bins werden in der ersten/letzten Spalte der *Halfcycle Count Matrix* hinzugefügt, in der Mittelwerte kleiner oder gleich bzw. größer als die definierten Schranken gehalten werden.
- `fminMean` definiert die untere Schranke der Mittelwerte. Alle Werte, die kleiner oder gleich diesem Wert sind, werden in einem Bin akkumuliert.
- `fmaxMean` definiert die obere Schranke der Mittelwerte. Werte, die das Maximum übersteigen, werden in das letzte Bin gezählt.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.28 ST_CM_RealFFT_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_RealFFT_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length  : UDINT := 512;     (* Length of FFT. *)
    bForward     : BOOL := TRUE;     (* Flag indicating forward FFT. *)
    bHalfSpec    : BOOL := TRUE;     (* Flag indicating length of output; if TRUE, the algorithm outp
uts half the spectrum (nFFT_Length/2 + 1). *)
```

```
    nChannels     : UDINT := 1;       (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `bForward` ist ein boolescher Parameter, der die Richtung der FFT angibt. Ist der Wert `TRUE`, so wird die normale FFT berechnet. Andernfalls wird die inverse FFT verwendet.
- `bHalfSpec` ist ein boolescher Parameter, der die Größe des Ergebnispuffers angibt. Ist der Wert `TRUE`, so gibt der Algorithmus das halbe Spektrum aus (`nFFT_Length`/2 + 1).
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.29 ST_CM_RMS_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_RMS_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels            : UDINT := 4;                  (* Number of channels. *)
    nSubChannels         : UDINT := 0;                  (* Number of subchannels. *)
    nBufferLength        : UDINT := 2000;               (* Buffer length. *)
    fDecibelThreshold    : LREAL := cCM_MinArgLog10;    (* Minimum argument of decadic logarithm for
 64-bit IEEE 754 arithmetic. *)
    bTransformToDecibel  : BOOL := TRUE;                (* Transform to decibel. *)
END_STRUCT
END_TYPE
```

- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.
- `nSubChannels` definiert die Anzahl von unabhängigen Unterkanälen. Diese muss größer oder gleich Null sein.
- `nBufferLength` ist die Zahl der im internen Puffer gehaltenen Eingangswerte je Kanal.
- `fDecibelThreshold` ist ein sehr kleiner Fließkommawert größer als Null. Werte, die kleiner als diese Zahl sind, werden vor einer Transformation in die Dezibel-Skala durch diesen Wert ersetzt. (Zweck ist die Vermeidung von Wertbereichsfehlern. Der Logarithmus von Null ist nicht definiert und strebt für den Grenzwert kleiner Argumente gegen minus unendlich. Entsprechendes gilt für das Argument der Zahl Null, arg(0). Der kleinste mögliche Wert ist 2.3e-308, dies entspricht der Konstanten `cCM_MinArgLog10`.)
- `bTransformToDecibel` ist ein boolescher Wert, der angibt, ob das Ergebnis der FFT in die Dezibel-Skala transformiert werden soll, entsprechend der Transformation x → 20 * log10(x).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.30 ST_CM_SlidingDFT_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_SlidingDFT_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nWindowLength  : UDINT := 1000;      (* Length of analysis window. *)
    nBins          : UDINT := 1;         (* Number of spectral bins. *)
    fDampingFactor : LREAL := 0.995;     (* Damping factor for SDFT. *)
    nChannels      : UDINT := 1;         (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `nBins` ist die Anzahl der zu berechnenden Spektralwerte. Die zugehörigen Indizes (`k := f / fSampleRate / nWindowLength`) müssen über die `Configure()` Methode nach der Initialisierung konfiguriert werden.
- `fDampingFactor` definiert den Wert des Dämpfungsparameters der Rekursionsformel.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.31 ST_CM_SparseSpectrum_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_SparseSpectrum_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nWindowLength        : UDINT := 1000;                                        (* Length of analy
sis window. *)
    fSampleRate          : LREAL := 10000;                                       (* Sample rate in
Hertz. *)
    nBins                : UDINT := 1;                                           (* Number of spect
ral bins. *)
    eSpectrumType        : E_CM_SpectrumType := E_CM_SpectrumType.eCM_Magnitude;  (* Type of spectru
m, i.e. plain DFT, magnitude, power. *)
    eWindowType          : E_CM_WindowType := eCM_HannWindow;                    (* Window type. *)
    aWindowParameters    : T_CM_WindowParameters := [2.5,1,1,1,1];               (* Window paramete
rs for specific windows, e.g. FlatTop. *)
    nOverlap             : UDINT := -1;                                          (* Size of overlap
 in samples. *)
    eScalingType         : E_CM_ScalingType := eCM_DiracScaling;                 (* Scaling type. *
)
    bTransformToDecibel  : BOOL := FALSE;                                        (* Transform to de
cibel. *)
    fDecibelThreshold    : LREAL := cCM_MinArgLog10;                             (* Minimum argumen
t of decadic logarithm for 64- bit IEEE 754 arithmetic. *)
    nChannels            : UDINT := 1;                                           (* Number of chann
els. *)
END_STRUCT
END_TYPE
```

- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `fSampleRate` Abtastrate des eingehenden Zeitsignals. Der Wert wird genutzt zur Skalierung des Ergebnisses in Hz.
- `nBins` ist die Anzahl der zu berechnenden Spektralwerte. Die zugehörigen Indizes (`k := f / fSampleRate / nWindowLength`) müssen über die `Configure()` Methode nach der Initialisierung konfiguriert werden.
- `eSpectrumType` definiert die Art der zu berechnenden Spektralwerte (vom Typ E_CM_SpectrumType [▶ 291]). Die Skalierung der Werte ist auf den jeweiligen Typ angepasst, sodass die Werte mit den Ergebnissen der Bausteine FB_CMA_RealFFT [▶ 216], FB_CMA_MagnitudeSpectrum [▶ 172] bzw. FB_CMA_PowerSpectrum [▶ 202] übereinstimmen.
- `eWindowType` definiert die verwendete Fensterfunktion (vom Typ E_CM_WindowType [▶ 291]). Ein guter Standardwert ist der Fenstertyp `eCM_HannWindow`.
- `aWindowParameters` beinhaltet die freien Parameter ausgewählter Fensterfunktionen. Bei der Verwendung von `eCM_KaiserWindow` definiert der erste Eintrag den Parameter beta; wird das `eCM_FlatTopWindow` verwendet, werden alle Parameter genutzt. Siehe Abschnitt Fensterfunktionen [▶ 20].

- `nOverlap` definiert die Anzahl der überlappenden Samples. Diese muss größer oder gleich Null sein. Wird der Wert `cCM_OverlapRecommended` gewählt, so wird eine empfohlene Überlappung intern berechnet (siehe F_CM_CalculateRecommendedOverlap [▶ 287]); der Wert `cCM_OverlapInactive` deaktiviert den intern verwendeten Puffer und setzt den Wert Null.
- `eScalingType` ermöglicht eine Auswahl der verwendeten Skalierung (vom Typ E_CM_ScalingType [▶ 290]), falls eine absolute Skalierung benötigt wird. Standardwert ist `eCM_DiracScaling`. Bei der Auswahl der Skalierung sollte die Art des Signals, entweder deterministische Signale oder breitbandige Signale mit stochastischen Anteil, berücksichtigt werden; Beide Arten erfordern unterschiedliche Skalierungen.
- `bTransformToDecibel` ist ein boolescher Wert, der angibt, ob das Ergebnis der FFT in die Dezibel-Skala transformiert werden soll, entsprechend der Transformation x → 20 * log10(x).
- `fDecibelThreshold` ist ein sehr kleiner Fließkommawert größer als Null. Werte, die kleiner als diese Zahl sind, werden vor einer Transformation in die Dezibel-Skala durch diesen Wert ersetzt. (Zweck ist die Vermeidung von Wertbereichsfehlern. Der Logarithmus von Null ist nicht definiert und strebt für den Grenzwert kleiner Argumente gegen minus unendlich. Entsprechendes gilt für das Argument der Zahl Null, arg(0). Der kleinste mögliche Wert ist 2.3e-308, dies entspricht der Konstanten `cCM_MinArgLog10`.)
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.32 ST_CM_SpikeEnergySpectrum_InitPars

Baustein-
spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_SpikeEnergySpectrum_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length           : UDINT := 1024;                                          (* Length of FFT
. *)
    nWindowLength         : UDINT := 800;                                           (* Length of ana
lysis window. *)
    fSampleRate           : LREAL := 10000.0;                                       (* Sample rate i
n Hertz. *)
    fDecayTime            : LREAL := 0.08;                                          (* Decay time co
nstant in seconds. *)
    fLowerFrequencyLimit  : LREAL := 500.0;                                         (* Lower limit o
f frequency band in Hertz. *)
    fUpperFrequencyLimit  : LREAL := 4950.0;                                        (* Upper limit o
f frequency band in Hertz. *)
    nOrder                : UDINT := 4;                                             (* Order of band
pass filter in range [1 .. 10]. *)
    eWindowType           : E_CM_WindowType := E_CM_WindowType.eCM_HannWindow;      (* Window type.
*)
    aWindowParameters     : T_CM_WindowParameters := [2.5, 1.0, 1.0, 1.0, 1.0];     (* Window parame
ters for specific windows, e.g. FlatTop. *)
    nOverlap              : UDINT := 4294967295;                                    (* Size of overl
ap in samples. *)
    eScalingType          : E_CM_ScalingType := E_CM_ScalingType.eCM_RootPowerSum;  (* Scaling type.
 *)
    bTransformToDecibel   : BOOL := FALSE;                                          (* Transform to
decibel. *)
    fDecibelThreshold     : LREAL := 2.3e-308;                                      (* Minimum argum
ent of decadic logarithm for 64- bit IEEE 754 arithmetic. *)
    nChannels             : UDINT := 1;                                             (* Number of cha
nnels. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.

- `fSampleRate` Abtastrate des eingehenden Zeitsignals. Der Wert wird genutzt zur Skalierung des Ergebnisses in Hz.
- `fDecayTime` definiert die Abklingzeit der Zeit-Wellen-Form (Spike-Funktion). Der Wert kann über die `Configure()` Methode konfiguriert werden um die Genauigkeit der berechneten Spikes zu verbessern.
- `fLowerFrequencyLimit` ist die untere Grenze des intern verwendeten Bandpasses (Bandbreiten Filters). Der Wert sollte oberhalb der erwarteten Maschinenvibration liegen.
- `fUpperFrequencyLimit` ist die obere Grenze des intern verwendeten Bandpasses (Bandbreiten Filters). Der Wert sollte möglichst hoch gewählt werden um hochfrequente Stöße identifizieren zu können.
- `nOrder` definiert die Ordnung des verwendeten IIR Filters.
- `eWindowType` definiert die verwendete Fensterfunktion (vom Typ E_CM_WindowType [▶ 291]). Ein guter Standardwert ist der Fenstertyp `eCM_HannWindow`.
- `aWindowParameters` beinhaltet die freien Parameter ausgewählter Fensterfunktionen. Bei der Verwendung von `eCM_KaiserWindow` definiert der erste Eintrag den Parameter beta; wird das `eCM_FlatTopWindow` verwendet, werden alle Parameter genutzt. Siehe Abschnitt Fensterfunktionen [▶ 20].
- `nOverlap` definiert die Anzahl der überlappenden Samples. Diese muss größer oder gleich Null sein. Wird der Wert `cCM_OverlapRecommended` gewählt, so wird eine empfohlene Überlappung intern berechnet (siehe F_CM_CalculateRecommendedOverlap [▶ 287]); der Wert `cCM_OverlapInactive` deaktiviert den intern verwendeten Puffer und setzt den Wert Null.
- `eScalingType` ermöglicht eine Auswahl der verwendeten Skalierung (vom Typ E_CM_ScalingType [▶ 290]), falls eine absolute Skalierung benötigt wird. Standardwert ist `eCM_DiracScaling`. Bei der Auswahl der Skalierung sollte die Art des Signals, entweder deterministische Signale oder breitbandige Signale mit stochastischen Anteil, berücksichtigt werden; Beide Arten erfordern unterschiedliche Skalierungen.
- `bTransformToDecibel` ist ein boolescher Wert, der angibt, ob das Ergebnis der FFT in die Dezibel-Skala transformiert werden soll, entsprechend der Transformation x → 20 * log10(x).
- `fDecibelThreshold` ist ein sehr kleiner Fließkommawert größer als Null. Werte, die kleiner als diese Zahl sind, werden vor einer Transformation in die Dezibel-Skala durch diesen Wert ersetzt. (Zweck ist die Vermeidung von Wertbereichsfehlern. Der Logarithmus von Null ist nicht definiert und strebt für den Grenzwert kleiner Argumente gegen minus unendlich. Entsprechendes gilt für das Argument der Zahl Null, arg(0). Der kleinste mögliche Wert ist 2.3e-308, dies entspricht der Konstanten `cCM_MinArgLog10`.)
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.33 ST_CM_VibrationAssessment_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_VibrationAssessment_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length           : UDINT := 512;                              (* Length of FFT. *)
    nWindowLength         : UDINT := 400;                              (* Length of analysis window.
 *)
    fSampleRate           : LREAL := 20000;                            (* Sample rate in Hertz. *)
    nChannels             : UDINT := 2;                                (* Number of channels. *)
    nOrder                : UDINT := 2;                                (* Maximum order of integrati
on: 0 = acceleration, 1 = velocity, 2 = place. *)
    nMaxBands             : UDINT := 1;                                (* Maximum number of frequenc
y bands. *)
    nMaxClasses           : UDINT := 3;                                (* Number of configurable thr
eshold classes. *)
```

```
    bMemorize              : BOOL :=TRUE;                                (* Flag if results are memori
zed. *)
    eWindowType            : E_CM_WindowType := eCM_HannWindow;          (* Window type. *)
    aWindowParameters      : T_CM_WindowParameters := [2.5,1,1,1,1];     (* Window parameters for spec
ific windows, e.g. FlatTop. *)
    nOverlap               : UDINT := -1;                                (* Size of overlap in samples
. *)
    bTransformToDecibel    : BOOL := TRUE;                               (* Transform to decibel. *)
    fDecibelThreshold      : LREAL := cCM_MinArgLog10;                   (* Minimum argument of decadi
c logarithm for 64- bit IEEE 754 arithmetic. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `nWindowLength` ist die Länge des Analysefensters in Samples. Die Länge muss größer als Eins und eine gerade Zahl sein.
- `fSampleRate` Abtastrate des eingehenden Zeitsignals. Der Wert wird genutzt zur Skalierung des Ergebnisses in Hz.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.
- `nOrder` ist die maximale Ordnung der Integration. Dies muss eine ganze Zahl zwischen Null und Zwei sein. Die Zahl der ermittelten Werte pro Kanal ist (nOrder+1).
- `nMaxBands` gibt die maximale Anzahl an Frequenzbändern an, für die der RMS-Wert berechnet wird.
- `nMaxClasses` ist die maximale Zahl der Klassen, die konfiguriert werden. Diese muss mindestens Eins sein.
- `bMemorize` ist eine boolesche Variable. Wenn sie den Wert `FALSE` hat, berechnet der Baustein die Nummer der höchsten Kategorie und des entsprechenden Kanals bei jedem Schritt neu. Wenn der Wert `TRUE` ist, werden die Ergebniswerte bei Überschreitung eines Grenzwerts so lange gespeichert, bis die ResetData()-Methode ausgeführt wird oder ein Kanal eine höhere Kategorie erreicht. Standardmäßig ist der Wert `TRUE`.
- `eWindowType` definiert die verwendete Fensterfunktion (vom Typ E_CM_WindowType [▶ 291]). Ein guter Standardwert ist der Fenstertyp `eCM_HannWindow`.
- `aWindowParameters` beinhaltet die freien Parameter ausgewählter Fensterfunktionen. Bei der Verwendung von `eCM_KaiserWindow` definiert der erste Eintrag den Parameter beta; wird das `eCM_FlatTopWindow` verwendet, werden alle Parameter genutzt. Siehe Abschnitt Fensterfunktionen [▶ 20].
- `nOverlap` definiert die Anzahl der überlappenden Samples. Diese muss größer oder gleich Null sein. Wird der Wert `cCM_OverlapRecommended` gewählt, so wird eine empfohlene Überlappung intern berechnet (siehe F_CM_CalculateRecommendedOverlap [▶ 287]); der Wert `cCM_OverlapInactive` deaktiviert den intern verwendeten Puffer und setzt den Wert Null.
- `bTransformToDecibel` ist ein boolescher Wert, der angibt, ob das Ergebnis der FFT in die Dezibel-Skala transformiert werden soll, entsprechend der Transformation x → 20 * log10(x).
- `fDecibelThreshold` ist ein sehr kleiner Fließkommawert größer als Null. Werte, die kleiner als diese Zahl sind, werden vor einer Transformation in die Dezibel-Skala durch diesen Wert ersetzt. (Zweck ist die Vermeidung von Wertbereichsfehlern. Der Logarithmus von Null ist nicht definiert und strebt für den Grenzwert kleiner Argumente gegen minus unendlich. Entsprechendes gilt für das Argument der Zahl Null, arg(0). Der kleinste mögliche Wert ist 2.3e-308, dies entspricht der Konstanten `cCM_MinArgLog10`.)

Fensterlänge beachten

Der Wert von `nWindowLength` muss kleiner oder gleich dem Wert von `nFFT_Length` sein. Die Länge der FFT kann sich an der benötigten Frequenzauflösung orientieren. Typischerweise wird oft ein Wert von ca. 4/5 der FFT-Länge als Fensterlänge verwendet.

Wenn `nFFT_Length` größer ist als `nWindowLength`, wird die Frequenzauflösung der FFT (und damit auch die Länge des Vektors der Rückgabewerte) vergrößert. Die Differenz der Länge wird vor der Fouriertransformation mit Nullen aufgefüllt. Dies kann sinnvoll sein, um eine höhere Frequenzauflösung zu erreichen oder, z.B. bei der Berechnung mit Rücktransformation in den Zeitbereich, zirkuläres Aliasing zu vermeiden. Das Ergebnis enthält trotz der höheren Frequenzauflösung allerdings nicht mehr Information.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.34 ST_CM_WatchUpperThresholds_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_WatchUpperThresholds_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nChannels     : UDINT := 10;    (* Number of channels. *)
    nSubChannels  : UDINT := 0;     (* Number of subchannels. *)
    nMaxClasses   : UDINT := 3;     (* Number of configurable threshold classes. *)
    bMemorize     : BOOL :=TRUE;    (* Flag if results are memorized. *)
END_STRUCT
END_TYPE
```

- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.
- `nSubChannels` definiert die Anzahl von unabhängigen Unterkanälen. Diese muss größer oder gleich Null sein.
- `nMaxClasses` ist die maximale Zahl der Klassen, die konfiguriert werden. Diese muss mindestens Eins sein.
- `bMemorize` ist eine boolesche Variable. Wenn sie den Wert `FALSE` hat, berechnet der Baustein die Nummer der höchsten Kategorie und des entsprechenden Kanals bei jedem Schritt neu. Wenn der Wert `TRUE` ist, werden die Ergebniswerte bei Überschreitung eines Grenzwerts so lange gespeichert, bis die ResetData()-Methode ausgeführt wird oder ein Kanal eine höhere Kategorie erreicht. Standardmäßig ist der Wert `TRUE`.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.35 ST_CM_ZoomFFT_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_CM_ZoomFFT_InitPars EXTENDS ST_CM_Object_InitPars :
STRUCT
    nFFT_Length       : UDINT := 2048;       (* Length of FFT. *)
    fSampleRate       : LREAL := 10000.0;    (* Sample rate in Hertz. *)
    fCenter           : LREAL := 200.0;      (* Center frequency of band in Hertz. *)
    nDecimationFactor : UDINT := 16;         (* FFT decimation factor. *)
    nOrder            : UDINT := 4;          (* Order of lowpass filter in range [1 .. 20]. *)
    nChannels         : UDINT := 1;          (* Number of channels. *)
END_STRUCT
END_TYPE
```

- `nFFT_Length` ist die Länge der FFT. Sie muss größer als Eins und eine ganzzahlige Potenz von Zwei sein.
- `fSampleRate` Abtastrate des eingehenden Zeitsignals. Der Wert wird genutzt zur Skalierung des Ergebnisses in Hz.
- `fCenter` ist die Mitte des zu verwendenden Frequenzbandes. Die Bandbreite ist abhängig von dem Dezimierungsfaktor.
- `nDecimationFactor` ist der Dezimierungsfaktor des angewendeten Tiefpassfilters nach der Zentrierung des Eingangssignals. Der Wert muss eine Potenz von zwei sein um eine ganzzahlige Division beim Downsampling zu gewährleisten.
- `nOrder` definiert die Ordnung des verwendeten IIR Filters.
- `nChannels` definiert die Anzahl von unabhängigen Kanälen. Diese muss größer als Null sein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.3.14.36 ST_MA_MultiArray_InitPars

Baustein-spezifische Struktur mit Initialisierungsparametern, die bei der Initialisierung des Bausteins ausgewertet werden.

```
TYPE ST_MA_MultiArray_InitPars :
STRUCT
    eTypeCode   : E_MA_ElementTypeCode := eMA_TypeCode_LREAL;                    (* Used datatype. *
)
    nDims       : UDINT := 1;                                                    (* Number of dimens
ions the array will have. *)
    aDimSizes   : ARRAY[0.. 15] OF UDINT := [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1];    (* Size for each di
mension. *)
END_STRUCT
END_TYPE
```

- `eTypeCode` Der Parameter legt den Elementypen (E_MA_ElementTypeCode [▶ 292]) der MultiArray Pufferelemente fest.
- `nDims` Der Parameter legt die Anzahl der Dimensionen des MultiArray Puffers fest.
- `aDimSizes` Die Größe jeder Dimension wird durch dieses Array angegeben. Ist das Shape des Eingangspuffers eines folgenden Algorithmus mit 'm x n' (im input stream) definiert, so muss der MultiArray Puffer mit aDimSizes := [m,n] angelegt werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4013	PC or CX (x86, x64)	Tc3_MultiArray

5.4 Globale Konstanten

5.4.1 GVL_CM

Analyse-Baustein Konstanten:

```
cCMA_MaxDest  : UDINT := 20;
cCMA_MaxID    : UDINT := 1000;
```

- `cCMA_MaxDest`: Maximale Anzahl der Destinations für einen Analyse-Baustein.
- `cCMA_MaxID`: Maximale ID die für einen Analyse-Baustein vergeben werden kann (entspricht der maximalen Anzahl möglicher Analyse-Bausteine).

Transfer Tray Parameter:

Das interne Transfer Tray wird für den Daten-Transfer zwischen den Analyse-Bausteinen mit den folgenden Konstanten initialisiert.

```
cCMA_InitParsTransferTray  : ST_MA_TransferTray_InitPars := (
                                  nStreams      := 2048,
                                  nMaxEntries   := 10,
                                  nQueueSize    := 64,
                                  bLockFree     := TRUE,
                                  nUpdatePeriod := 2 );
```

- `nStreams` Dieser Parameter gibt an, wie viele unabhängig voneinander funktionierende Queues bereitgestellt werden. Für jeden taskübergreifenden Datenstrom sollte es eine separate Queue geben. Zusätzliche Queues benötigen keine System-Ressourcen.

- `nMaxEntries` Dieser Parameter gibt an, wie viele Elemente die Queues maximal beinhalten können. Für die Kommunikation von Datenpuffern ist es meistens sinnvoll, wenn alle in Frage kommenden Puffer in einer Queue Platz haben, so dass keine Buffer-Overrun-Bedingungen eintreten können. Auch ein Wert von Eins ist auswählbar.
- `nQueueSize` Die reservierte Länge der Queues. Dieser Wert muss größer als `nMaxEntries` und außerdem eine ganzzahlige Potenz von Zwei sein.
- `bLockFree` Wenn dieser Parameter `TRUE` ist, wird eine moderne lock-freie Implementation für die Queues verwendet. Dies ist die Voreinstellung. Anderenfalls wird eine klassische Implementation mit Interrupt-Sperren verwendet. Die lock-freie Implementation kann ein besseres Zeitverhalten des Gesamtsystems erreichen, führt aber bei extrem hoher Auslastung unter Umständen zu größeren Latenzen.
- `nUpdatePeriod` Dieser Parameter gibt an, wie oft interne Zwischenergebnisse aufgefrischt werden. Durch einen Wert größer Eins kann die Häufigkeit aufwendiger Operationen leicht reduziert werden. Werte von Zwei (voreingestellt) oder Drei sind in der Regel sinnvoll.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

5.4.2 GVL_CM_Base

Grenzwert-Konstanten:

```
cCM_MinArgLog10 : LREAL :=  2.3E-308;   (* approximate minimum argument of decadic logarithm *)
cCM_MinArgLogN  : LREAL :=  2.3E-308;   (* approximate minimum argument of natural logarithm *)
cCM_MinArgDiv   : LREAL :=  2.3E-308;   (* minimum argument of division *)
```

Diese Konstanten dienen dazu Wertebereichsfehler zu vermeiden. Der Logarithmus reeller Zahlen ist bei negativen Zahlen und Null nicht definiert. Aus diesem Grund sind Konstanten mit Werten gegen Null definiert. Das Gleiche gilt für die Division durch Null.

Konstanten zur empfohlenen Überlappung:

```
cCM_OverlapRecommended  : UDINT := TO_UDINT(-1);    (* setting for recommended overlap (e.g. HannWin
dow: 50% overlapping) *)
cCM_OverlapInactive     : UDINT := TO_UDINT(-2);    (* setting in order to deactivate overlap *)
```

Diese Konstanten dienen dazu, die Größe des internen Datenpuffers bei der Verwendung von Fensterfunktionen einzustellen. Sie dienen in den InitPars Strukturen [▶ 297] der jeweiligen Funktionsbausteine als Werte für den Parameter nOverlap [▶ 302].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

5.4.3 Global_Version

Diese globale Konstante beinhaltet die Bibliotheksversion.

Alle Bibliotheken haben eine spezifische Versionsnummer. Diese Versionsnummer wird auch im SPS Library Repository angezeigt.

```
VAR_GLOBAL CONSTANT
    stLibVersion_Tc3_CM : ST_LibVersion;
END_VAR
```

Typdefinition der konstanten Struktur: ST_LibVersion

Zum Vergleichen der existierenden Versionsnummer mit der geforderten Versionsnummer, kann die Funktion F_CmpLibVersion (in Tc2_System library) genutzt werden.

5.4.4 Param

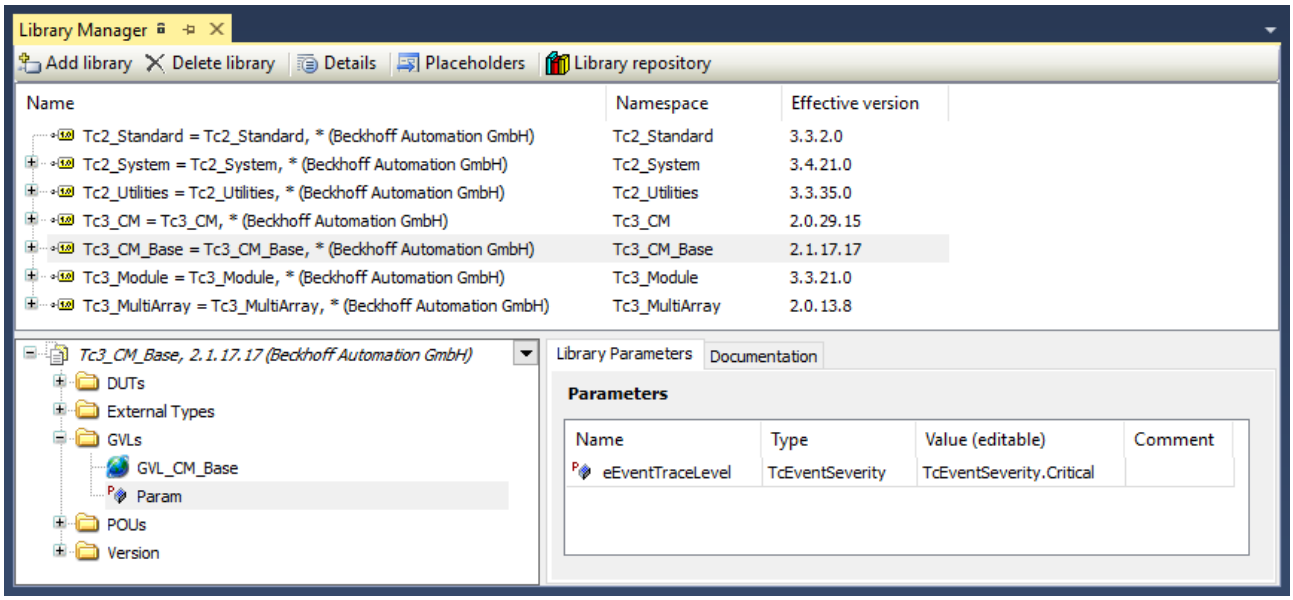
Parameterliste der vor der Laufzeit einstellbaren Parameter

```
VAR_GLOBAL CONSTANT
    eEventTraceLevel  : TcEventSeverity := TcEventSeverity.Critical;
END_VAR
```

- `eEventTraceLevel`: Der Parameter ermöglicht die zentrale Einstellung der TcEventSeverity der Ereignisse für die Nutzung des TcEventLogger (siehe Übersicht) innerhalb der Condition Monitoring Bibliothek.

Einstellung

Änderungen der Parameter erfolgen über den Library Manager: In der Tc3_CM_Base Bibliothek im Ordner GVLs unter Param in der Spalte „Value (editable)“.



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM_Base

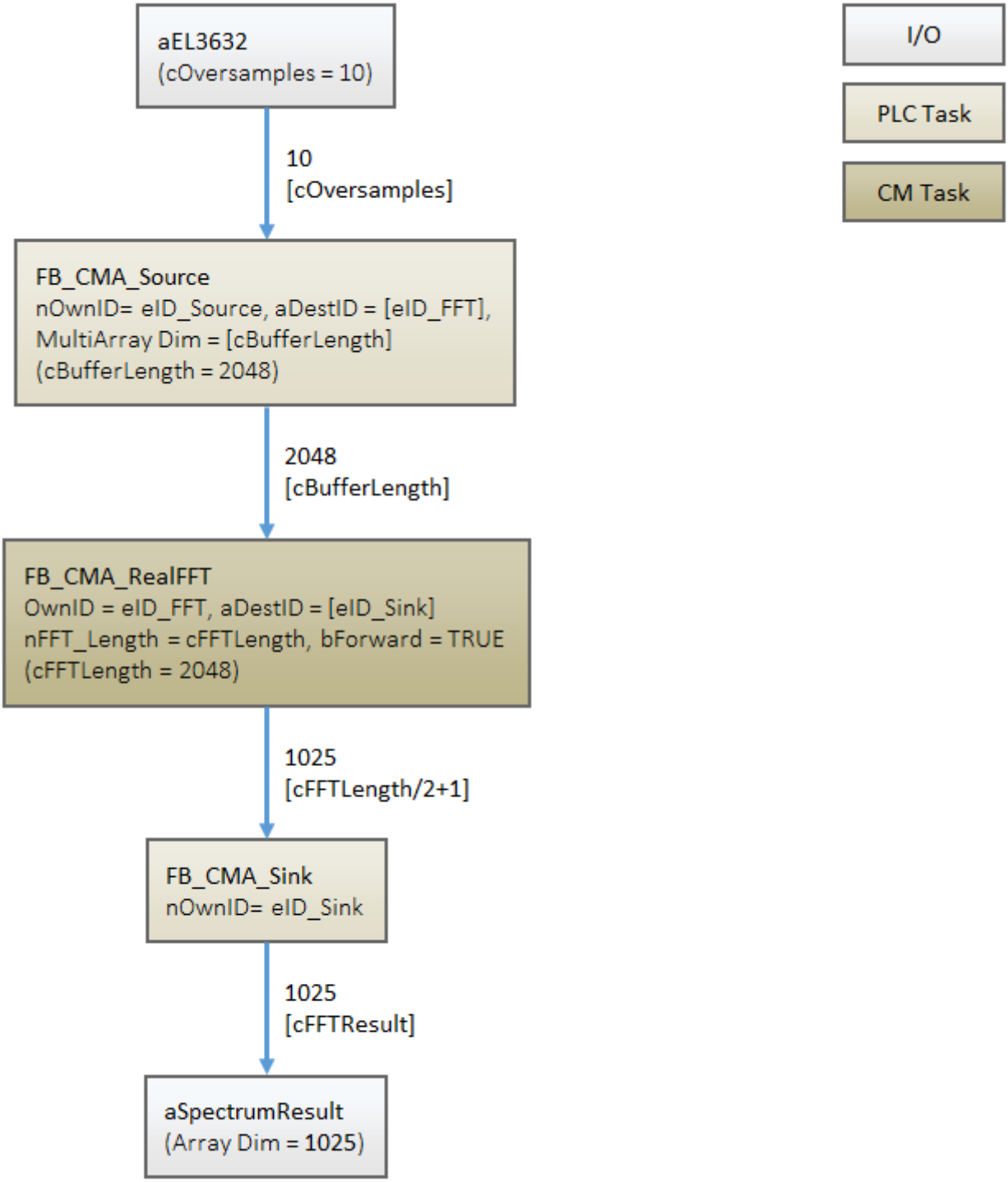
6 Beispiele

6.1 FFT mit reell-wertigem Eingangssignal

Das Beispiel zeigt eine exemplarische Implementierung einer Spektrums-Berechnung mit dem Baustein FB_CMA_RealFFT [▶ 216].

Das Beispiel können Sie hier herunterladen:
https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/3394507531.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration des RealFFT-Funktionsbausteins.

FFT Länge	2048
Forwärtsberechnung	TRUE

Voraussetzungen

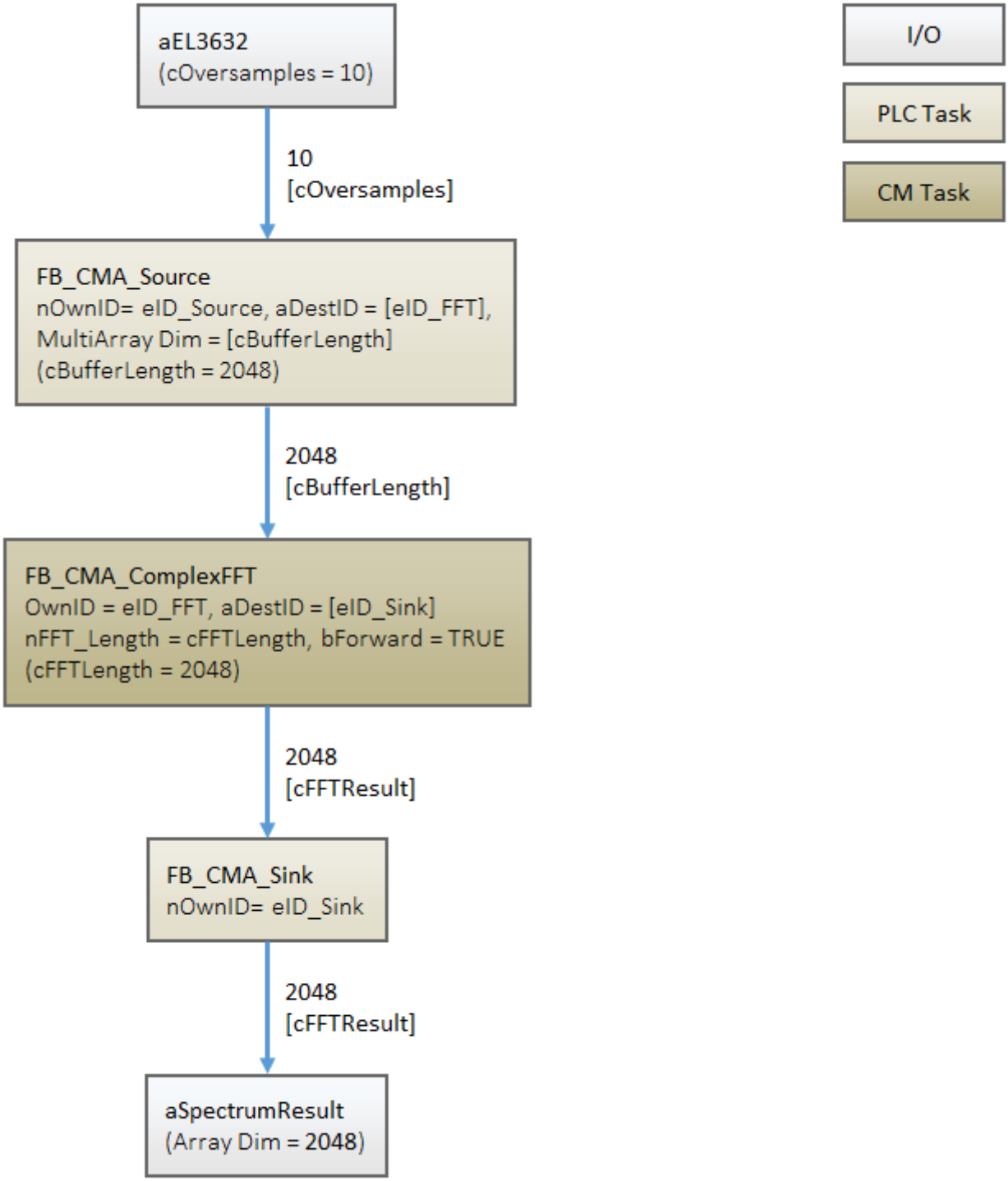
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4018	PC oder CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.2 FFT mit komplex-wertigem Eingangssignal

Das Beispiel zeigt eine exemplarische Implementierung einer Spektrums-Berechnung mit dem Baustein FB_CMA_ComplexFFT [▶ 112]. Im Vergleich zur Verwendung des Bausteins FB_CMA_RealFFT [▶ 216] wird für die benötigten MultiArrays der Datentyp LCOMPLEX verwendet.

Das Beispiel können Sie hier herunterladen:
https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/3394479755.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration des ComplexFFT-Funktionsbausteins.

Typ-Code	eMA_TypeCode_LCOMPLEX
FFT Länge	2048

Forwärtsberechnung	TRUE

Voraussetzungen

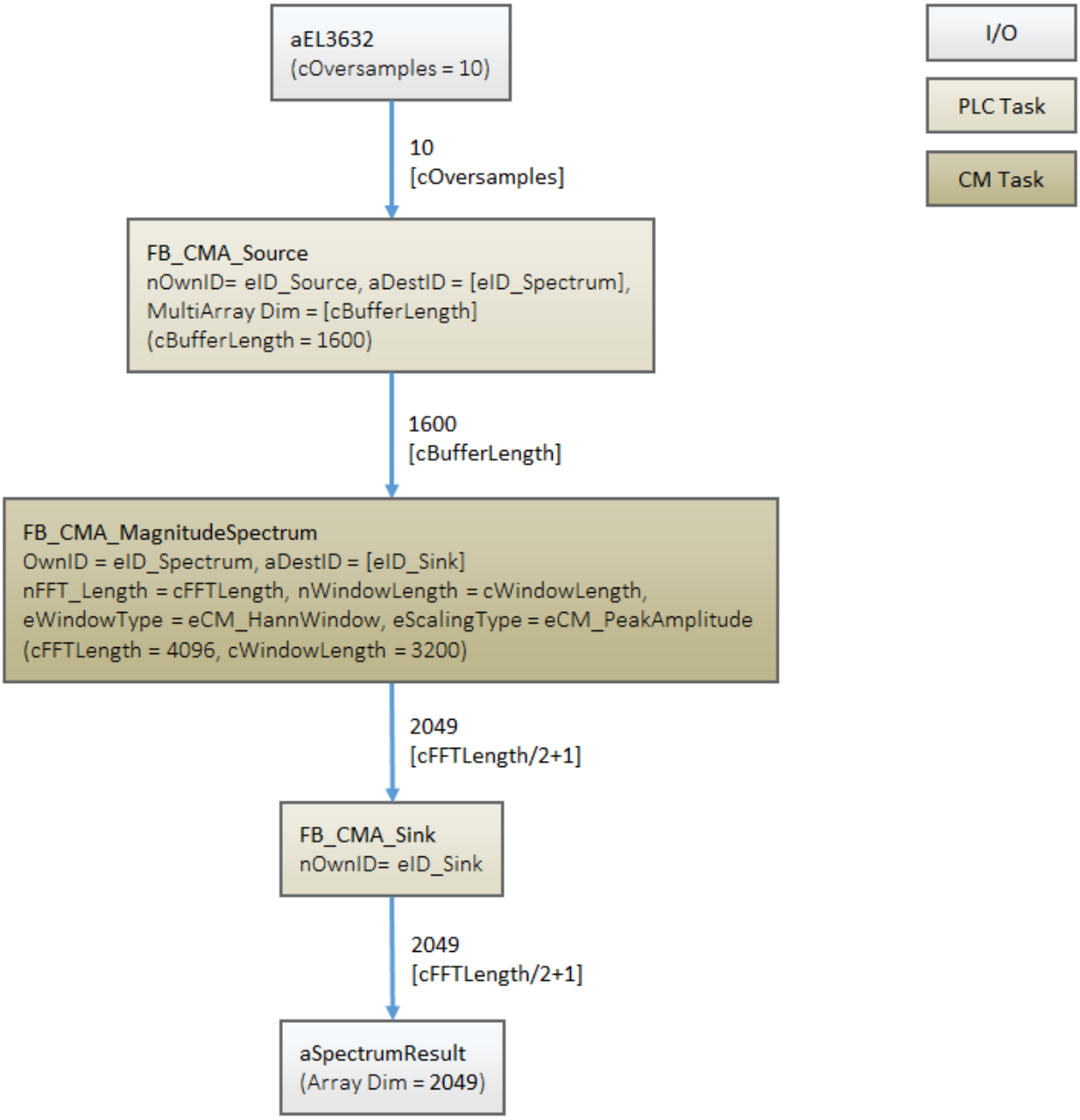
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4018	PC oder CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.3 Magnitudenspektrum

Dieses Beispiel implementiert ein Einzelkanal-Magnitudenspektrum. Der Code ist in zwei Tasks aufgeteilt; einer Steuerungstask, welche das diskrete Eingangssignal von einem Hardwaremodul sammelt, z.B. EL3632, und einer CM-Task, die das Spektrum berechnet. Das Blockdiagramm unten zeigt die im Beispiel implementierte Analysekette.

Den Quellcode für das Beispiel können Sie hier herunterladen:
https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/3394574859.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration des Magnitudenspektrum-Funktionsbausteins.

FFT-Länge	4096
Fenstergröße	3200
Puffergröße	1600
Fenster Typ	eCM_HannWindow
Skalierungsart	eCM_ROOT_POWER_SUM
Skalierung in Dezibel (dB)	FALSE

Globale Konstanten

Diese Parameter werden in der Liste der globalen Variablen als Konstanten definiert.

```
VAR_GLOBAL CONSTANT
    cOversamples       : UDINT := 10;      // oversampling factor
    cBufferLength      : UDINT := 1600;    // size of buffer for spectrum
    cWindowLength      : UDINT := 3200;   // size of window
    cFFTResult         : UDINT := 2049;   // size of spectrum result
    cFFTLength         : UDINT := 4096;   // spectrum lines
END_VAR
```

Code für Steuerungstask

Der folgende Code-Ausschnitt zeigt die Deklaration im MAIN Programm:

```
PROGRAM MAIN

VAR CONSTANT
    cInitSource   : ST_MA_MultiArray_InitPars := ( eTypeCode := eMA_TypeCode_LREAL, nDims := 1, aDim
Sizes := [cBufferLength]);
END_VAR

VAR
     nInputSelection   : UDINT := 1; // Switch between hardware and function generator
     nSample  : UDINT;
     aEl3632 AT %I*    : ARRAY[1..cOversamples] OF INT; // Input from hardware e.g. EL3632
     aBuffer  : ARRAY[1..cOversamples] OF LREAL;

     fbSource      : FB_CMA_Source :=( stInitPars := cInitSource, nOwnID := eID_Source, aDestIDs :=
[eID_Spectrum]); // Initialize source buffers
     fbSink        : FB_CMA_Sink := (nOwnID := eID_Sink);
     aSpectrumResult   : ARRAY[1..cFFTResult] OF LREAL; // Copy result
END_VAR
```

Methode ruft MAIN Programm auf:

```
fbSource.Input1D(pDataIn := ADR(aBuffer),
               nDataInSize := SIZEOF(aBuffer),
               eElementType := eMA_TypeCode_LREAL,
               nWorkDim := 0,
               pStartIndex := 0,
               nOptionPars := 0);

fbSink.Output1D(pDataOut := ADR(aSpectrumResult),
               nDataOutSize := SIZEOF(aSpectrumResult),
               eElementType := eMA_TypeCode_LREAL,
               nWorkDim := 0,
               nElements := 0,
               pStartIndex := 0,
               nOptionPars := 0,
               bNewResult => bCalculate);
```

Code für CM-Task

Deklaration im MAIN_CM Programm:

```
PROGRAM MAIN_CM

VAR CONSTANT
     cInitSpectrum : ST_CM_MagnitudeSpectrum_InitPars := (nFFT_Length := cFFTLength,
                                    nWindowLength := cWindowLength,
                                    bTransformToDecibel:= FALSE,
                                    eWindowType := eCM_HannWindow,
                                    eScalingType := eCM_RMS);
END_VAR
VAR
     fbSpectrum : FB_CMA_MagnitudeSpectrum :=(stInitPars := cInitSpectrum,
                                   nOwnID := eID_Spectrum,
                                   aDestIDs := [eID_Sink]);
END_VAR
```

Methode ruft MAIN_CM Programm auf:

```
fbSpectrum.Call();
```

Das Ergebnis des Beispielcodes kann für ein sinusförmiges Signal beliebiger Amplitude und Frequenz als Eingangssignal getestet werden. Die Variable fRmsValue oben sollte genau dem Wert von Amplitude/ SQRT(2) entsprechen.

Jede Frequenz kann dem entsprechenden Array-Index des Spektrumergebnisses zugeordnet werden.
Rechenformel:
Abtastrate = Oversampling-Faktor / Abtasttaskzykluszeit
Index = Frequenz * (FFT Länge / Abtastrate)

Voraussetzungen

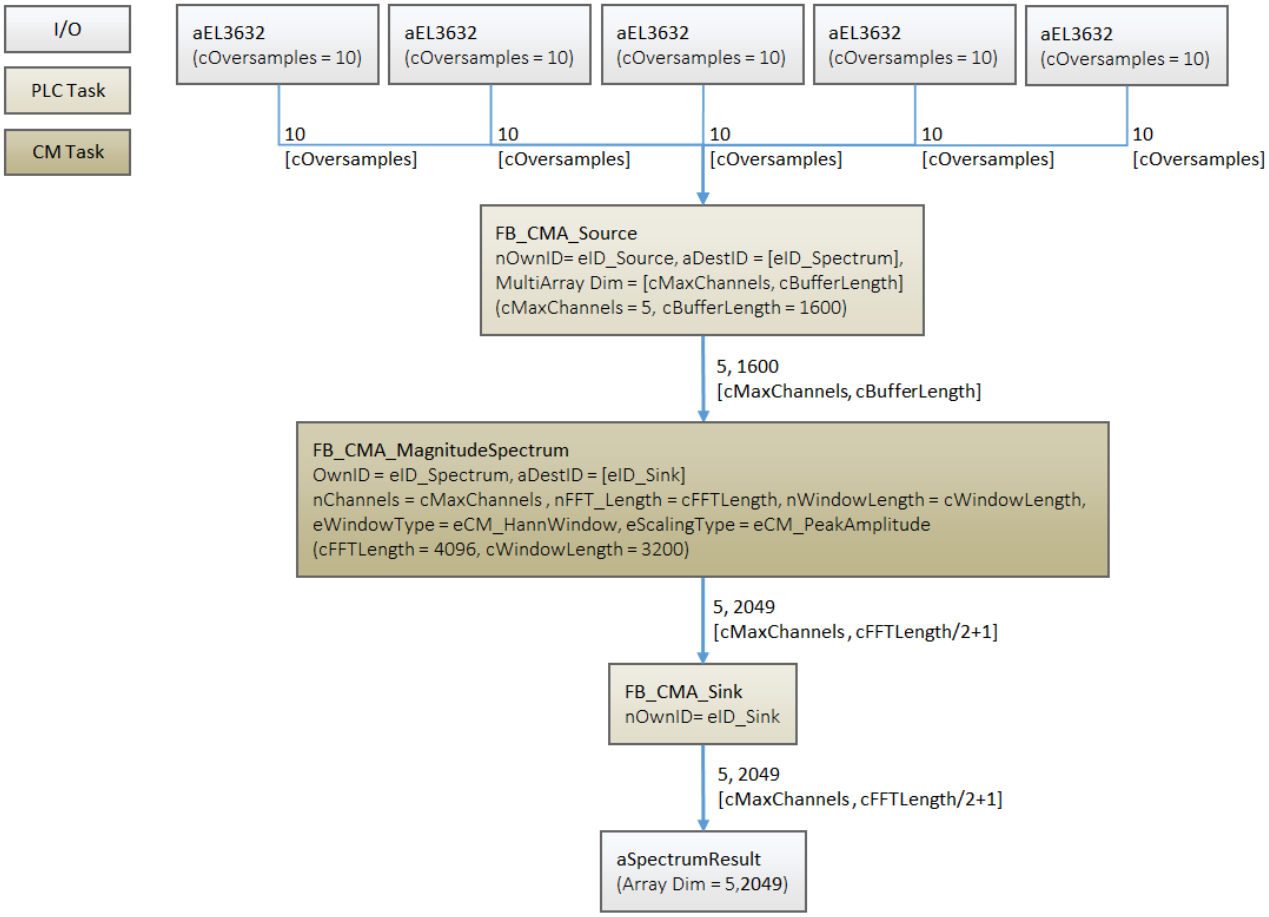
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4013	PC oder CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.4 Mehrkanal-Magnitudenspektrum

Dieses Beispiel implementiert das Magnitudenspektrum für 5 Eingangskanäle gleichzeitig. Der Code ist in zwei Tasks aufgeteilt; einer Steuerungstask, welche die Eingangsabtastungen von einem Hardwaremodul sammelt, z.B. EL3632, und einer CM-Task, die das Spektrum errechnet. Das Blockdiagramm unten zeigt die im Programm implementierte Analysekette.

Den Quellcode für das Beispiel können Sie hier herunterladen:
https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/3394576523.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration der Magnitudenspektrum-Funktionsbausteine.

Kanäle	5
FFT-Länge	4096
Fenstergröße	3200
Puffergröße	1600
Window Typ	eCM_HannWindow
Skalierungsart	eCM_RootPowerSum
Umwandeln in Dezibel	FALSE

Globale Konstanten

Diese Parameter werden in der Liste der globalen Variablen als Konstanten definiert.

```
VAR_GLOBAL CONSTANT
    cOversamples  : UDINT := 20;                                      // number of oversamples
    cMaxChannels  : UDINT := 5;                                       // Number of data channels
    cWindowType   : E_CM_WindowType := E_CM_WindowType.eCM_HannWindow; // window type for analysis
    cWindowLength : UDINT := 3200;                                    // length of signal window.
    cOverlap      : UDINT := 1600;                                    // recommended buffer overlap
    cBufferLength : UDINT := cWindowLength -
 cOverlap;                 // internal buffer size with 50% overlapping
    cFFTLength    : UDINT := 4096;                                    // length of FFT for mag.
 spectrum
    cFFTResult    : UDINT := 2049;                                    // result of mag. spectrum
(cFFTLength/2+1)
END_VAR
```

Globale Variablen

Diese Parameter werden in der Liste der globalen Variablen definiert.

```
VAR_GLOBAL
    bInvalidateCh4 : BOOL := FALSE; // Invalidate input signal on channel 4
END_VAR
```

Erläuterungen

Das Ergebnis des Beispielcodes kann für ein sinusförmiges Signal beliebiger Amplitude und Frequenz getestet werden. Die Effektivwerte werden im Array `aRmsValue` gemäß der entsprechenden Kanalnummer gespeichert. Das Ergebnis muss genau der Spitzenamplitude von jedem sinusförmigen Signal geteilt durch `SQRT(2)` entsprechen. Der Beispielcode kann für mehr als 5 Kanäle erweitert werden, je nach Anforderungen und Ressourcen des Zielsystems.

Das Setzen der globalen Variablen `bInvalidateCh4 := TRUE` demonstriert ein mögliches Fehlerhandling bei unzulässigen Eingangsdaten.

Voraussetzungen

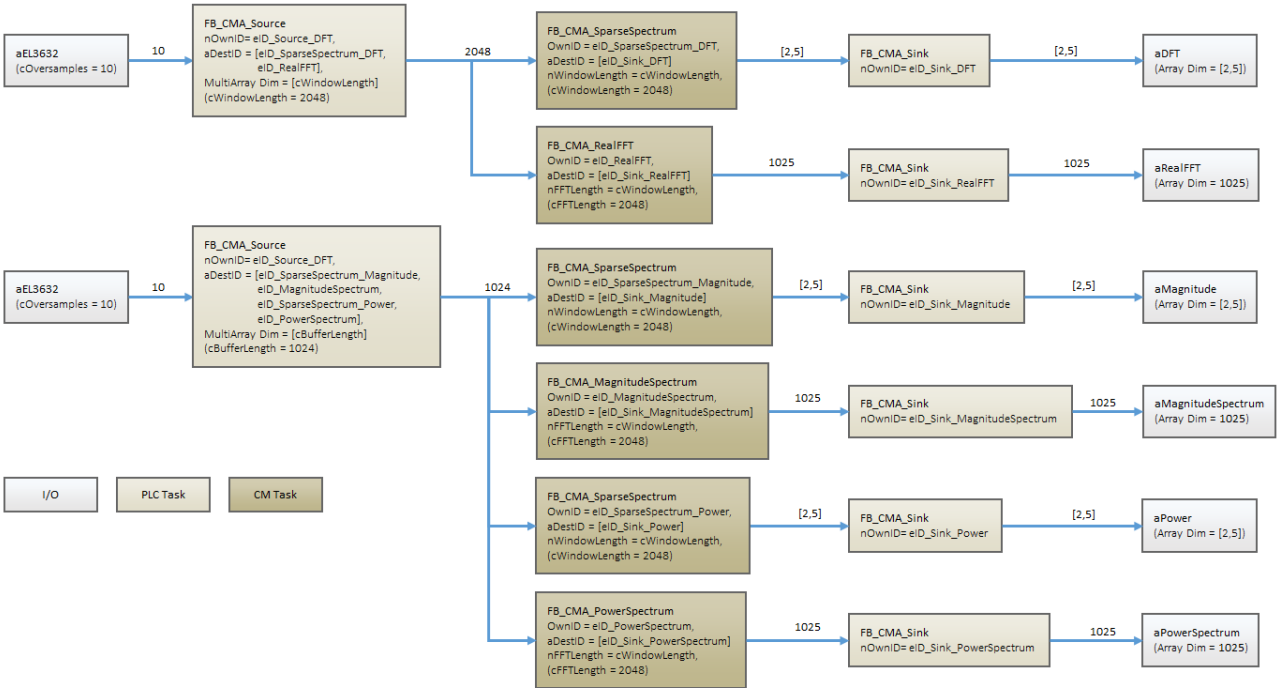
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.5 Berechnung einzelner Spektralwerte

Dieses Beispiel implementiert exemplarisch die Verwendungsmöglichkeiten des Funktionsbausteins FB_CMA_SparseSpectrum. Es werden verschiedene Fensterfunktionen und Skalierungen für die möglichen Berechnungen von DFT-, Magnituden- und Power-Werten vorgestellt. Numerische Effekte bei der Detektion werden in zwei Frequenzanteilen eines generierten Signals aufgezeigt: Die erste Frequenz entspricht einem Vielfachen der numerischen Auflösung im Spektralbereich, die Zweite liegt zwischen zwei solcher Werte.

Den Quellcode für das Beispiel können Sie hier herunterladen: https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/9066662027.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit den wichtigen Parametern für die Konfiguration der Funktionsbausteine.

FFT-Länge	2048 / 2048 / 2048
Fenstergröße	2048 / 2048 / 2048
Puffergröße	2048 / 1024 / 1024
Fenster Typ	eCM_RectangularWindow / eCM_HannWindow / eCM_HannWindow
Spaklierungsart	eCM_NoScaling / eCM_PeakAmplitude / eCM_PeakAmplitude
Typ der Spektralwerte	eCM_DFT / eCM_Magnitude / eCM_Power

Konfiguration der Frequenzbänder

In der GVL_Constants werden die zentralen Parameter für die Initialisierung des Algorithmus sowie die Eigenschaften des generierten Signals definiert.

```
VAR_GLOBAL CONSTANT
    cSampleRate   : LREAL := 10000;                                    // Sample rate of input s
ignal.
    cWindowLength : UDINT := 2048;                                     // Internal buffer size w
ith 50% overlapping
    cResolution   : LREAL := cSamplerate / cWindowLength;              // Frequency resolution
    cBands        : UDINT := 2;                                        // Number of bands
    cSetFrequency : ARRAY[1..cBands] OF LREAL := [ 41*cResolution, 413 ];  // Frequency in Hz; [ exa
ct, intermediate ]
    cSetAmplitude : ARRAY[1..cBands] OF LREAL := [ 1.0, 2.0 ];         // Peak amplitudes of sin
e signals
    cBandWidth    : UDINT := 5;                                        // Computed bins per
frequency.
    cDFTBins      : UDINT := cBandWidth * cBands;                      // Number of spectral bin
s, cBandWidth bins per frequency
END_VAR
```

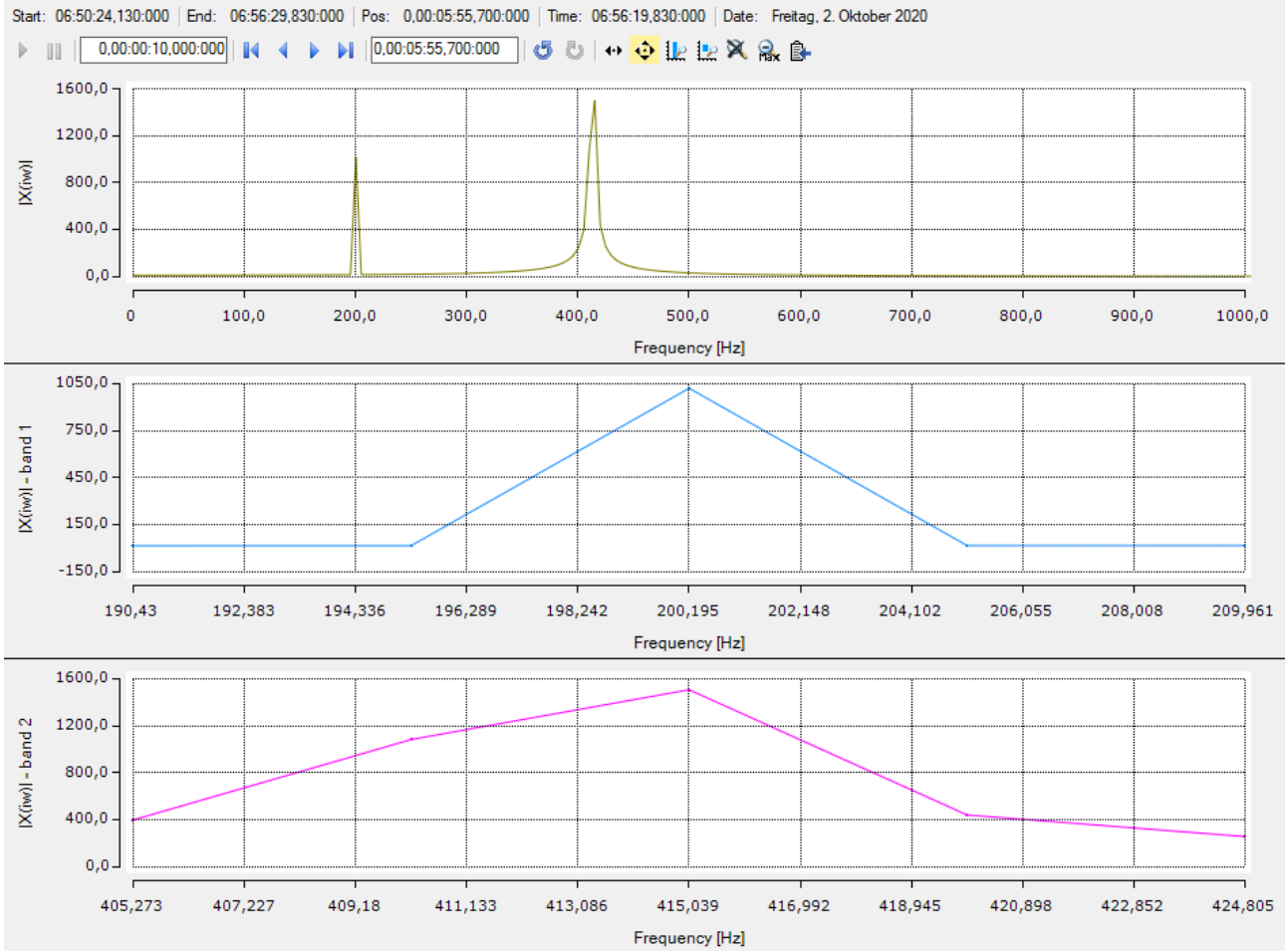
Das generierte Signal (`MAIN.aBuffer`) besteht aus zwei Frequenzkomponenten. Eine ist bezüglich der numerischen Auflösung gewählt, d.h. sie ist ein Vielfaches von `f = 10000 Hz/2048 = 4,8828125 Hz`. Die zweite ist so gewählt, dass der Peak zwischen zwei Spektralwerten liegt und somit nicht exakt dargestellt werden kann. Um die numerischen Effekte veranschaulichen zu können, werden um die jeweiligen gesuchten Spektralwerte weitere vier Werte berechnet. Die Konfiguration erfolgt in der `MAIN_CM`.

```
// Compute parameters, adjust if cDFTBins is changed.
FOR i := 1 TO cBands DO
    k := LREAL_TO_DINT(cSetFrequency[i] / cResolution);
    aDFTBins[(i-1) * cBandWidth + 1] := DINT_TO_UDINT(MAX(k-2,1));
    aDFTBins[(i-1) * cBandWidth + 2] := DINT_TO_UDINT(MAX(k-1,1));
    aDFTBins[(i-1) * cBandWidth + 3] := DINT_TO_UDINT(MIN(k+0,nyquist));
    aDFTBins[(i-1) * cBandWidth + 4] := DINT_TO_UDINT(MIN(k+1,nyquist));
    aDFTBins[(i-1) * cBandWidth + 5] := DINT_TO_UDINT(MIN(k+2,nyquist));
END_FOR
```

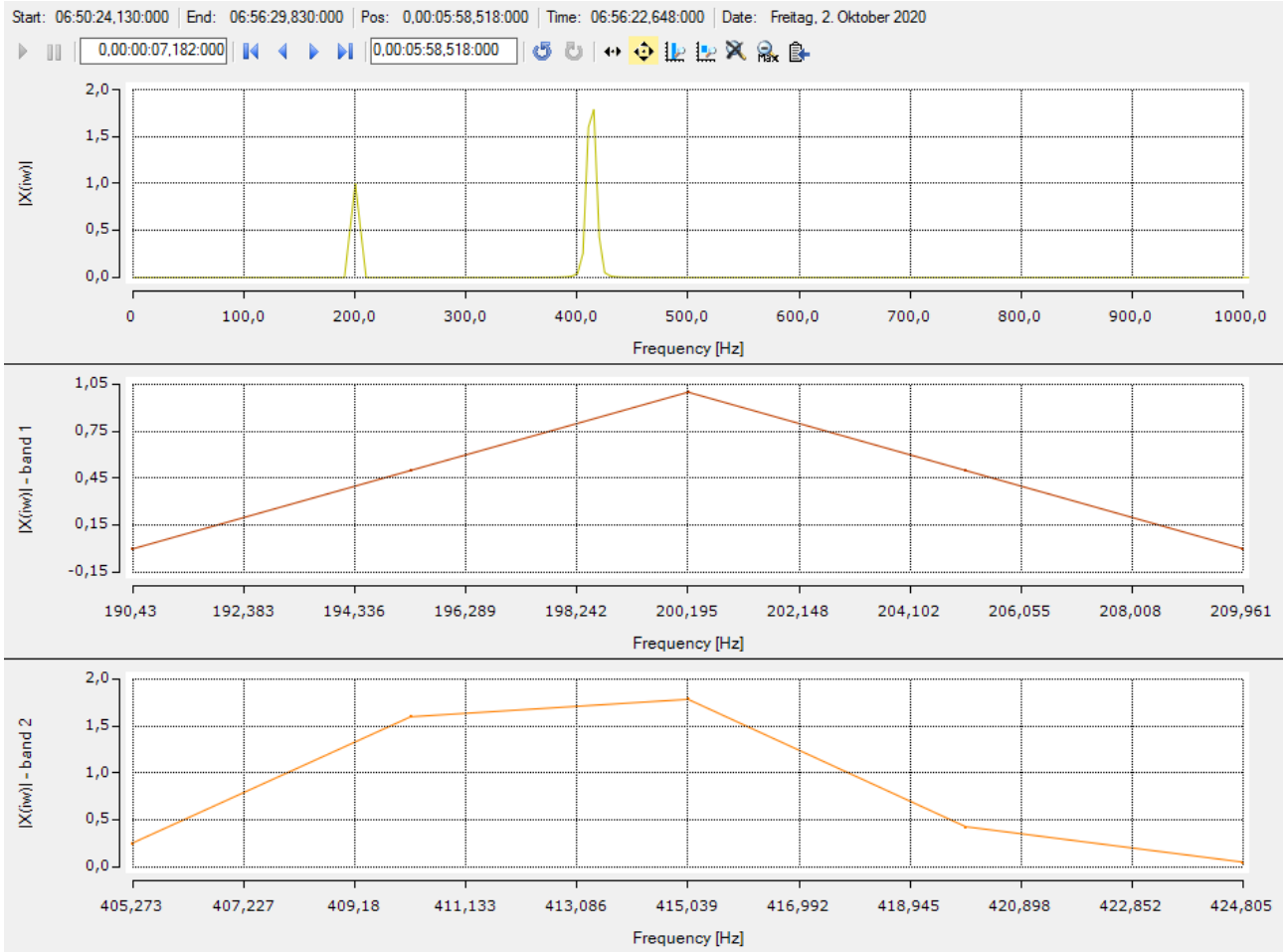
Visualisierung der Ergebnisse

Das Beispiel beinhaltet ein umfangreiches TwinCAT Measurement Projekt, in dem die berechneten Spektralwerte aus dem Baustein FB_CMA_SparseSpectrum den Referenzalgorithmen (FB_CMA_RealFFT, FB_CMA_MagnitudeSpectrum, FB_CMA_PowerSpectrum) gegenübergestellt sind. Es wird der Unterschied zwischen einer Anregung als vielfaches der numerischen Frequenzauflösung und einem Wert zwischen zwei solcher Werte verdeutlicht. Die visualisierten Frequenzbänder können als „Zoom“ auf den entspechenden Bereich angesehen werden.

Die folgenden Abbildungen zeigen die Ergebnisse für die Gegenüberstellung zu den Bausteinen FB_CMA_RealFFT und FB_CMA_MagnitudeSpectrum.



Spektrum des FB_RealFFT (oben) sowie die Spektralwerte der Bänder 1 (mitte) und 2 (unten).



Magnituden Spektrum des Bausteins FB_MagnitudeSpectrum (oben) sowie die Spektralwerte der Bänder 1 (mitte) und 2 (unten).

Voraussetzungen

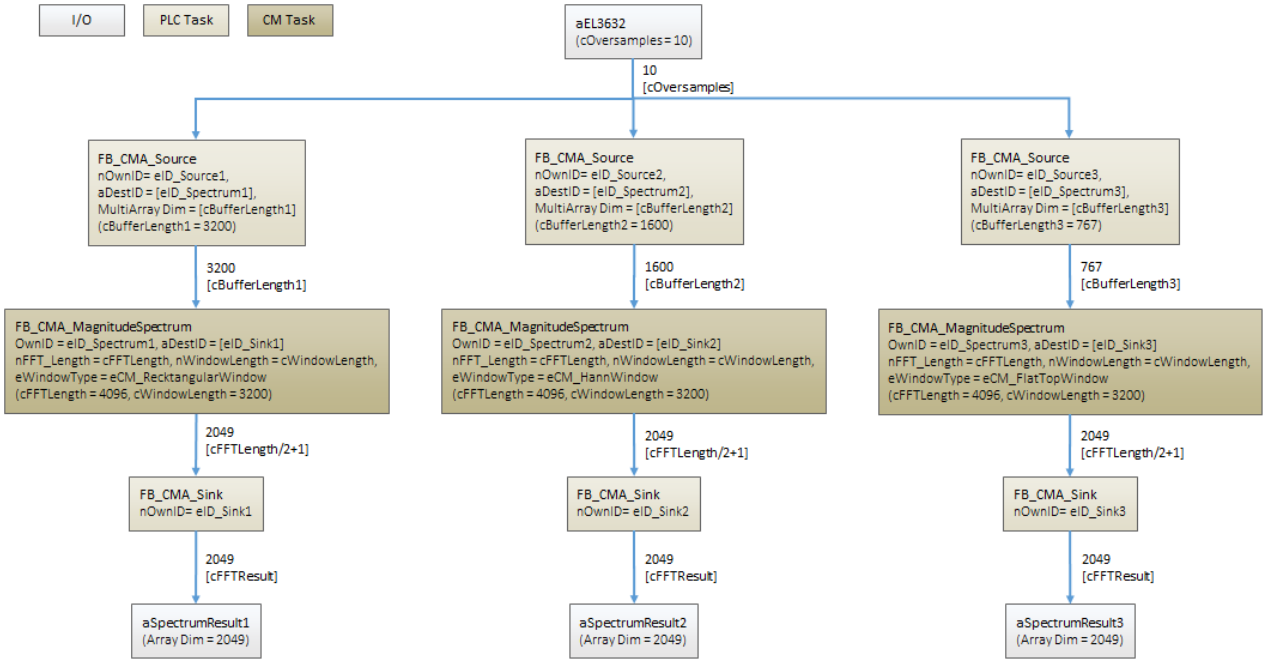
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.6 Fensterfunktionen

Dieses Beispiel implementiert exemplarisch mehrere Einzelkanal-Magnitudenspektren und vergleicht die Verwendung verschiedener Fensterfunktionen. Dabei werden drei Varianten vorgestellt: Eine mit deaktiviertem Overlap-Mechanismus, eine automatische Berechnung der empfohlenen Überlappung und eine manuelle Konfiguration unter Verwendung einer geeigneten Firmware-Funktion.

Den Quellcode für das Beispiel können Sie hier herunterladen: https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/5261536139.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration der Magnitudenspektrum-Funktionsbausteine.

FFT Länge	4096
Fenstergröße	3200
Puffergröße	3200 / 1600 / 767
Fenster Typ	`eCM_RectangularWindow` / `eCM_HannWindow` / `eCM_FlatTopWindow`
Dezibel-Transformation	`False`

Berechnung/Festlegen der Überlappung

Variante 1: Verwendung der Funktion F_CM_CalculateRecommendedOverlap [▶ 287]

Mittels der genannten Funktion kann die empfohlene Überlappung zur Laufzeit ausgerechnet werden. Nach Download/Login, werden die benötigten Werte in der `MAIN_CM` im ersten Zyklus aufgerufen.

```
IF bCalculateOverlap THEN
    // recommended overlap for window 1
    nOverlap1 := F_CM_CalculateRecommendedOverlap(eWindowType := cWindowType1, aWindowParameters :=
cWindowParameters1, nWindowLength := cWindowLength);

    // recommended overlap for window  2
    nOverlap2 := F_CM_CalculateRecommendedOverlap(eWindowType := cWindowType2, aWindowParameters :=
cWindowParameters2, nWindowLength := cWindowLength);

    // recommended overlap for window  3
    nOverlap3 := F_CM_CalculateRecommendedOverlap(eWindowType := cWindowType3, aWindowParameters :=
cWindowParameters3, nWindowLength := cWindowLength);

bCalculateOverlap := FALSE;
END_IF
```

Die so erhaltenen Längen können anschließend in der Konfiguration der Analyseketten eingepflegt werden (Prameter `cOverlap1`, `cOverlap2`, `cOverlap3` in der `GVL_Constants`.). Zu beachten sind hierbei die Abhängigkeiten in der Pufferlänge! Nach einem erneuten Download/Login ist die Applikation hinsichtlich der Überlappung konfiguriert.

Variante 2: Auslesen der Initialparameter im Online View [▶ 85]

Wird bei der Initialisierungs-Struktur für die Überlappung der Wert `nOverlap := cCM_OverlapRecommended` verwendet, so wird in der Init-Phase des Funktionsbausteins die optimale Überlappung in Abhängigkeit der Fensterparameter berechnet. Dieser Wert kann nach dem Download und Login im Online View des Bausteins in dem Knoten `stInitPars` ausgelesen werden. Bei dieser Variante ist kein Starten der Applikation nötig!

Expression	Type	Value
stInitPars	ST_CM_MagnitudeS...	
eObjState	TCOM_STATE	TCOM_STATE_OP
sObjName	STRING(127)	'CMA_MagnitudeSpectrum'
eTraceLevel	TCEVENTSEVERITY	Critical
nFFT_Length	UDINT	4096
nWindowLength	UDINT	3200
eWindowType	E_CM_WINDOWTYPE	eCM_HannWindow
aWindowParameters	T_CM_WindowPara...	
nOverlap	UDINT	1600
eScalingType	E_CM_SCALINGTYPE	eCM_DiracScaling
bTransformToDecibel	BOOL	FALSE
fDecibelThreshold	LREAL	2.3E-308
nChannels	UDINT	1

Online View zu Spektrum 2 (Hann-Fenster)

Voraussetzungen

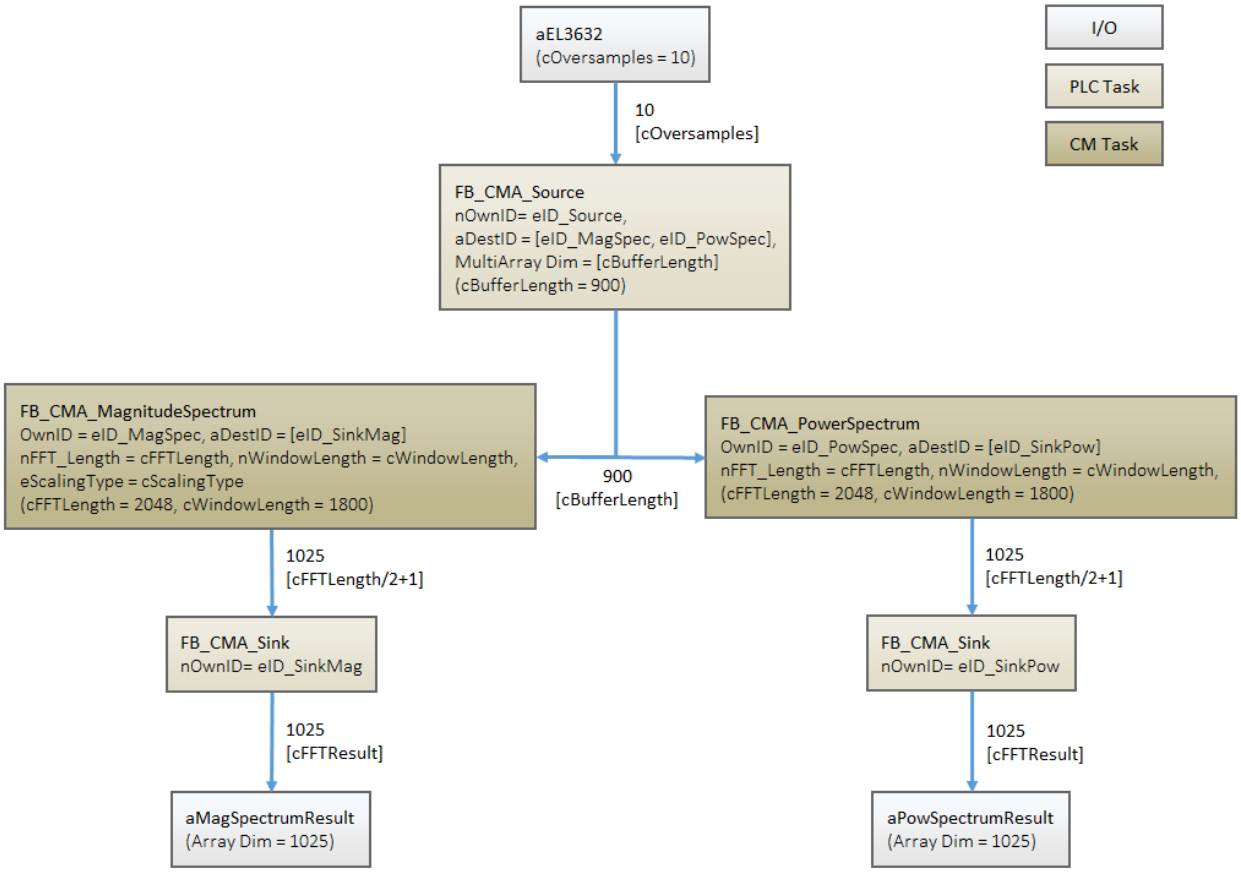
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.7 Skalierung von Spektren

Wie unter Skalierung von Spektren [▶ 26] beschrieben, bietet die Condition Monitoring Bibliothek eine Reihe von verschiedenen Möglichkeiten zur Skalierung von Spektren. Dieses Tutorial bietet die Möglichkeit anhand einer einfachen Sinusschwingung verschiedene vorbereitete Skalierungen zu betrachten und das theoretische Wissen zu vertiefen. Die Scopes sind auf den Bereich von 0 Hz bis 400 Hz eingeschränkt um die Unterschiede deutlicher darstellen zu können.

Das Tutorial können Sie hier herunterladen:
https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/3394510859.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration der verwendeten Funktionsbausteine FB_CMA_MagnitudeSpectrum [▶ 172] und FB_CMA_PowerSpectrum [▶ 202].

In der GVL_Constant finden Sie drei Szenarien die sich durch Ein- und Auskommentieren der markierten Code-Segmente sowie Aktivieren der Konfiguration testen können. Das erwartete Verhalten der Szenarien ist in der GVL als Kommentar dokumentiert.

FFT Länge	2048
Fenstergröße	1800
Umwandeln in Dezibel	TRUE / FALSE
Fenster Typ	eCM_HannWindow
Skalierungsart	eCM_PeakAmplitude / eCM_RMS

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4018	PC oder CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

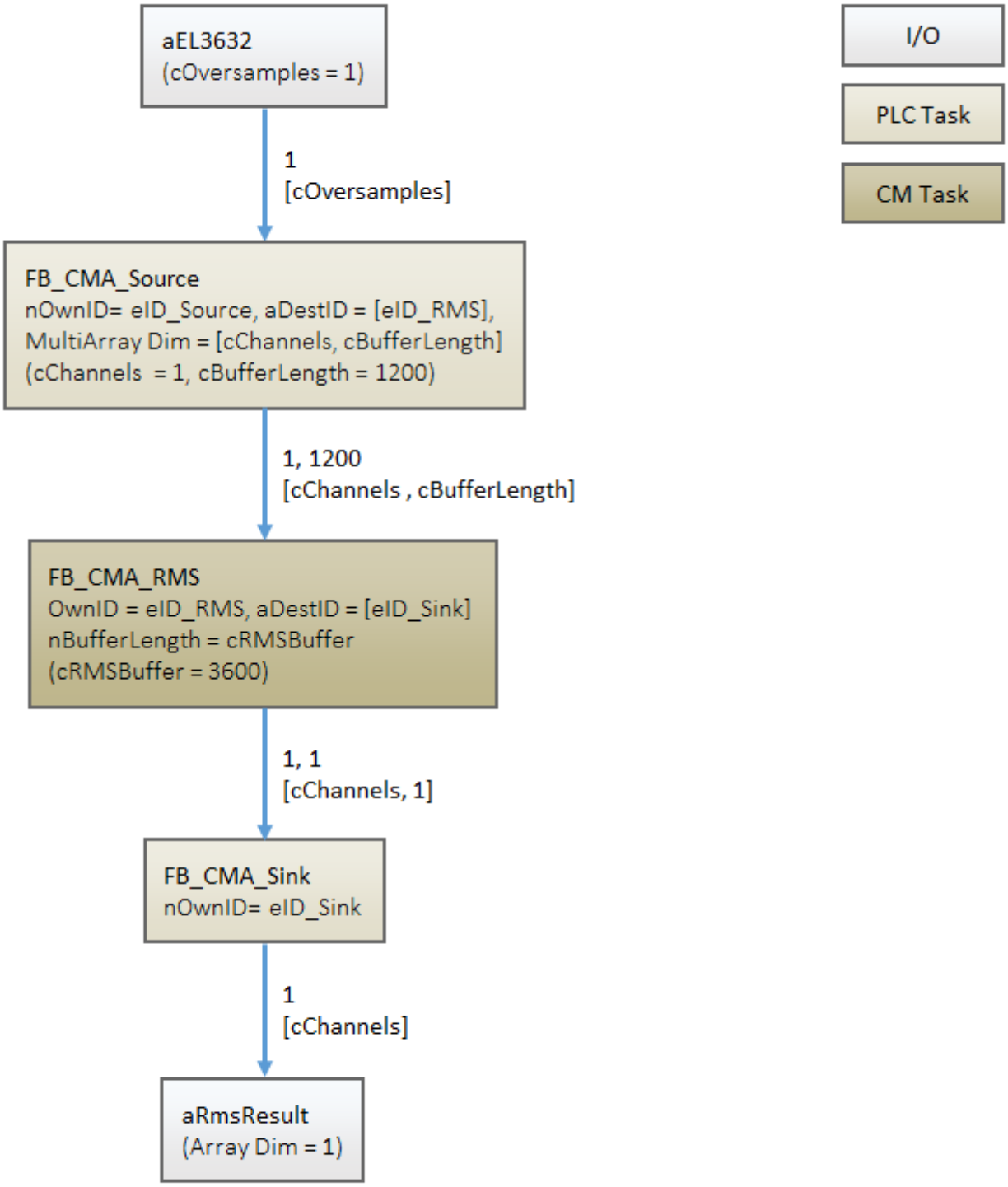
6.8 Zeitbasierter RMS

Das Beispiel zeigt eine exemplarische Implementierung einer Berechnung des zeitbasierten RMS eines Signals mit dem Baustein FB_CMA_RMS [▶ 220].

Das Beispiel können Sie hier herunterladen:

https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/3394509195.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration des Programmbausteins zur Berechnung des zeitbasierten RMS eines Signals.

Kanäle	1
Puffergröße	1200
Umwandeln in Dezibel	FALSE

Voraussetzungen

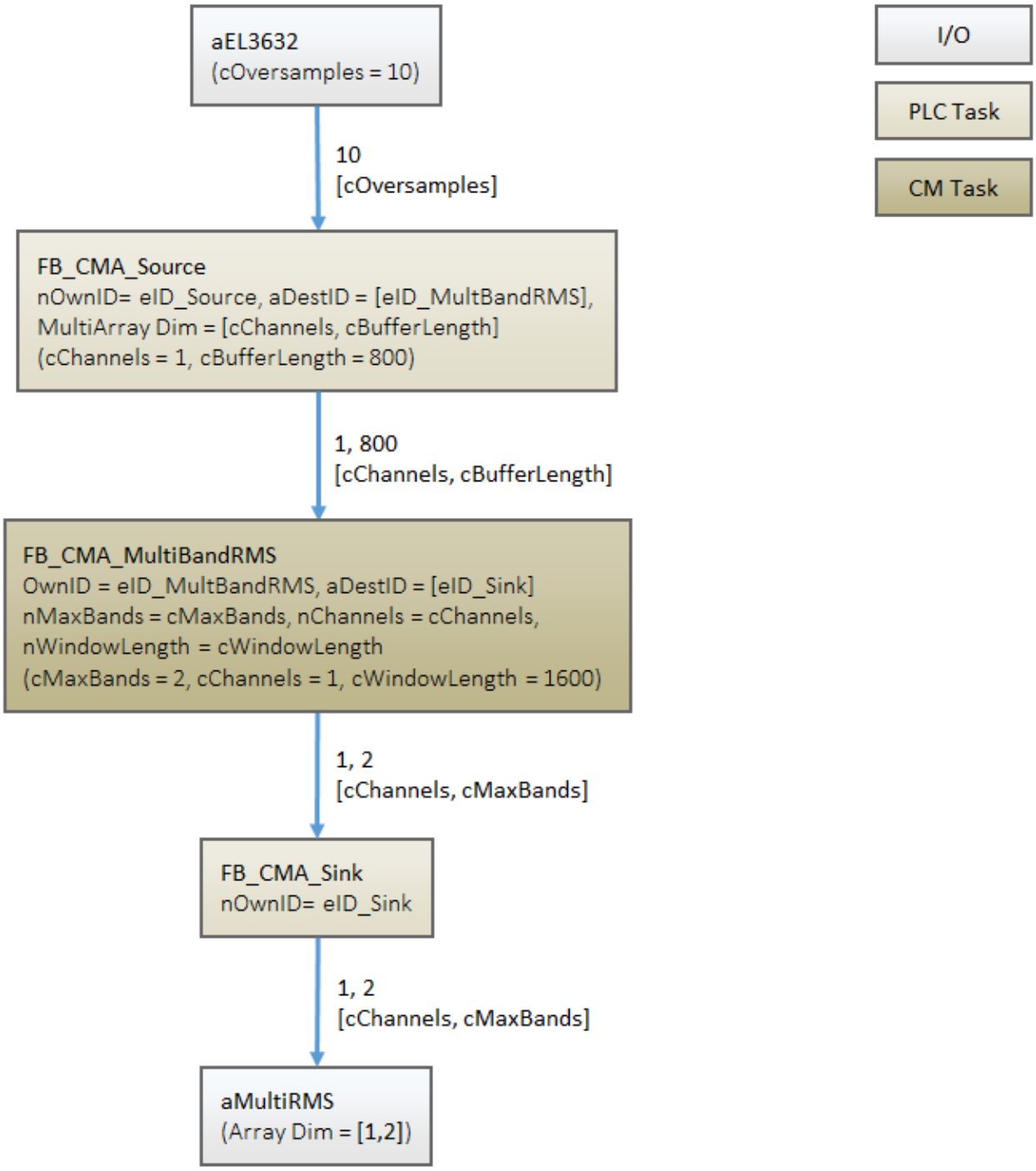
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4016.12	PC oder CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.9 Multiband-RMS

Das Beispiel zeigt eine exemplarische Implementierung zur Berechnung mehrerer frequenzband-limitierter RMS-Werte eines Signals mit dem Baustein FB_CMA_MultiBandRMS [▶ 189].

Das Beispiel können Sie hier herunterladen:
https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/3394505867.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration des Funktionsbausteins zur Berechnung mehrerer frequenzband-limitierter RMS-Werte eines Signals

Größe der FFT	2048
Fenstergröße	1600
Sampling-Rate	10000
Frequenzbänder	2
Kanäle	1

Fenster Typ	eCM_HannWindow
Umwandeln in Dezibel	FALSE

Voraussetzungen

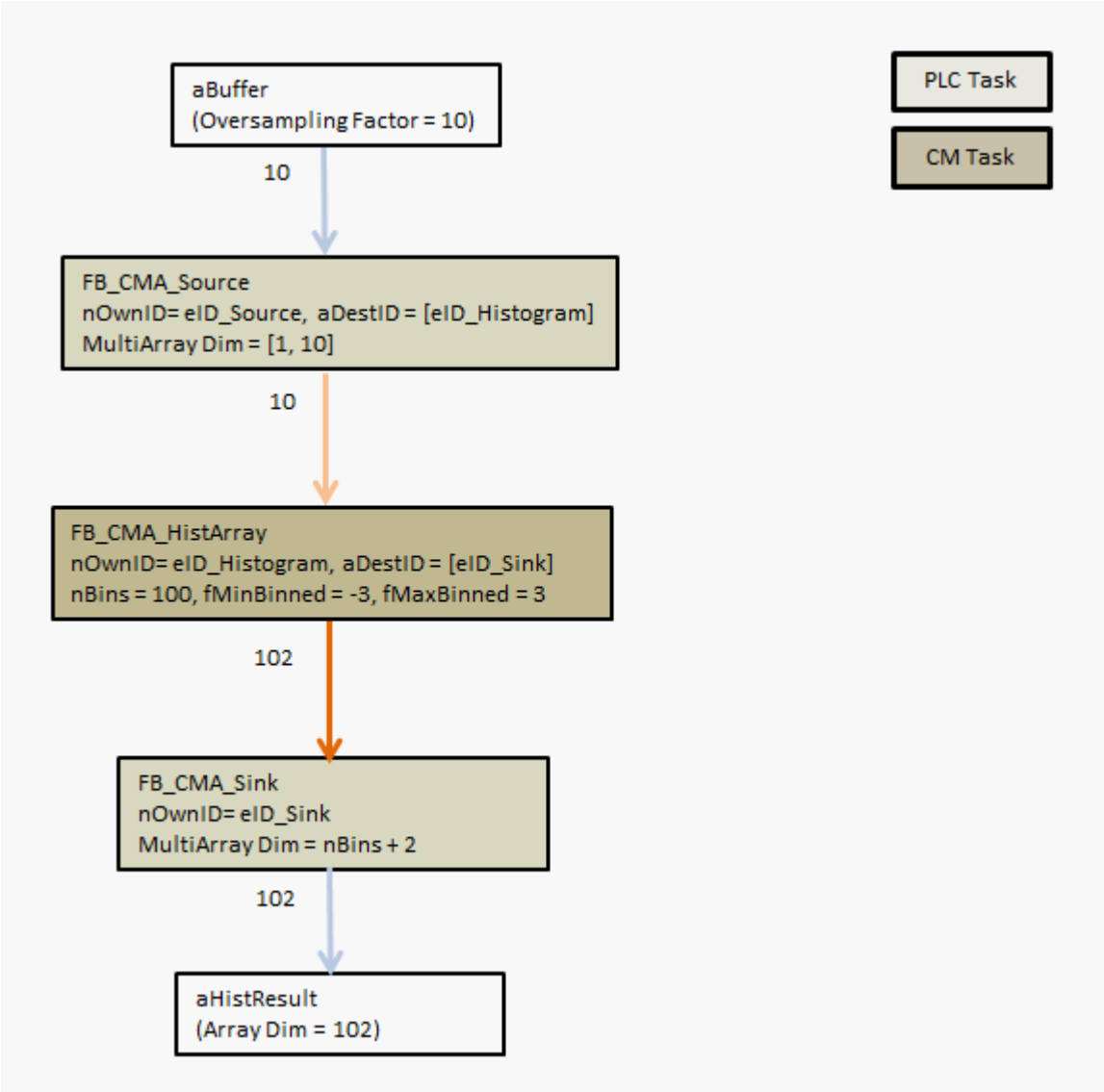
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4016.12	PC oder CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.10 Histogramm

Dieses Beispiel implementiert ein Histogramm. Der Code ist in zwei Tasks aufgeteilt; eine Steuerungstask welche die Eingangsdaten sammelt, z.B. aus EL3632, und eine sogenannten CM-Task, die das Histogramm errechnet. Das Blockdiagramm unten zeigt die Analysekette.

Den Quellcode für das Beispiel können Sie hier herunterladen:
https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/3394573195.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt die wichtigsten Parameter, um den Histogramm-Baustein zu konfigurieren:

Histogram Bins	100
Appended Datasets	10
Oversamples	10
Max. Bin Limit	+3 or +5
Min. Bin Limit	-3 or -5
Channels	1
Buffer Length	100

Globale Konstanten

Die oben angegebenen Parameter lassen sich als globale Konstanten definieren:

```
VAR_GLOBAL CONSTANT
    cBufferLength : UDINT := 100;
    cChannels     : UDINT := 1;
    cOversamples  : UDINT := 10;
    cMaxBins      : UDINT := 100;
    cAppendedData : UDINT := 10;
    cBinLimit_1   : LREAL := 3;
    cBinLimit_2   : LREAL := 5;
END_VAR
```

Code für die MAIN Task

Der folgende Code-Ausschnitt zeigt die Deklaration im MAIN Programm:

```
PROGRAM MAIN
VAR CONSTANT
    cInitSource    : ST_MA_MultiArray_InitPars
    := (eTypeCode := eMA_TypeCode_LREAL, nDims := 2, aDimSizes := [1, cBufferLength]);
END_VAR
VAR
    nInputSelection : UDINT := 1;
    nSample         : UDINT;
    aEl3632 AT %I* : ARRAY [1..cOversamples] OF INT;
    aBuffer         : ARRAY [1..cOversamples] OF LREAL;
    fbSource        : FB_CMA_Source := (stInitPars := cInitSource, nOwnId
:= eID_Source, aDestIDs := [eID_Histogram]);
    fbSink          : FB_CMA_Sink := (nOwnID := eID_Sink);
    aHistReulst     : ARRAY [1..cMaxBins+2];
END_VAR
```

Der folgende Code-Ausschnitt zeigt die Methodenaufrufe im MAIN Programm:

```
fbSource.Input2D(pDataIn := ADR(aBuffer),
                aDataInSize  := SIZEOF(aBuffer),
                eElementType := eMA_TypeCode_LREAL,
                nWorkDim0    := 0,
                nWorkDim1    := 1,
                pStartIndex  := 0,
                nOptionPars  := 0);

fbSink(pDataOut := ADR(aHistResult),
        nDataOutSize := SIZEOF(aHistResult),
        eElementType := eMA_TypeCode_UINT64,
        nWorkDim0    := 0,
        nWorkDim1    := 1,
        nElements    := 0,
        pStartIndex  := 0,
        nOptionPars  := 0);
```

Code für die CM Task

Die Variablen-Deklaration im MAIN_CM Programm:

```
VAR CONSTANT
    cInitHistArray : ST_CM_HistArray_InitPars := (nChannels := cChannels, nBins := cMaxBins,
fMinBinnded := -cBinLimit_1, fMaxBinned := cBinLimit_1);
END_VAR
```

Die Methodenaufrufe im MAIN_CM Programm:

```
fbHistArray.CallEx(nAppendData := cAppendData, bReset := );

IF bConfig then
    fbHistArray.Configure(pArg := ADR(aHisArrayConfig), nArgSize := SIZEOF(aHistArrayConfig)
END_IF
```

Die Configure Methode ist optional, ermöglicht aber eine feine Einstellung der Parameter fMinBinned und fMaxBinned während der Laufzeit.

Random Number Generator

Ein Histogramm wird sehr oft als eine visuelle Hilfe verwendet, um die zugrunde liegende Verteilung aller Messwerte zu verstehen, z.B. die Spitzen im Schwingungssignal. Der im Samplecode enthaltene Funktionsgenerator ist für diesen Zweck erweitert. Der Funktionsgenerator kann die üblichen und praxisnahen Zufallszahlen und deren Verteilungen simulieren. Mit der Variablen E_DistributionType können Sie eine Verteilung wie Exponentielle, Normale (oder Gaußsche), Chi-Squared oder Gamma auswählen. Standardmäßig werden die Zufallszahlen aus einer Gleichverteilung erzeugt.

Beachten Sie, dass jede Verteilung ein oder mehrere Parameter erfordert, um die Ausbreitung der Zufallszahlen oder deren Bereich festzustellen. Dies kann über die Eingangsvariable aRange erfolgen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4016.12	PC oder CX (x86, x64)	Tc3_CM (v1.0.19), Tc3_CM_Base, Tc3_MultiArray

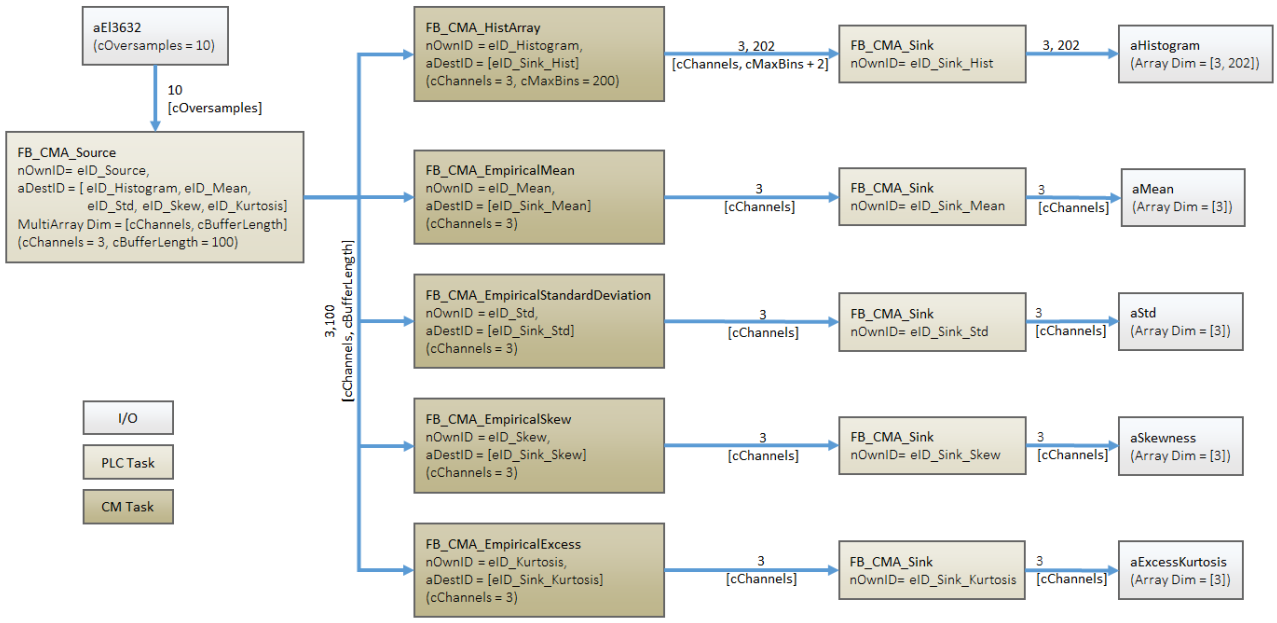
6.11 Statistische Methoden

In diesem Beispiel werden die Möglichkeiten zur statistischen Auswertung von Daten der Condition Monitoring Bibliothek präsentiert. Statistische Auswertungen für Standard Normal und Gamma verteilte Signaldaten sowie eines Sinussignals werden verarbeitet. Die Funktionsbausteine FB_CMA_HistArray [▶ 154], FB_CMA_EmpiricalMean [▶ 131], FB_CMA_EmpiricalStandardDeviation [▶ 141], FB_CMA_EmpiricalSkew [▶ 136] und FB_CMA_EmpiricalExcess [▶ 126] werden verwendet.

Das Beispiel können Sie hier herunterladen:

https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/5261532811.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration der verwendeten Funktionsbausteine.

Puffergröße	100
Kanäle	3
Frequenz-Bins	200

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022	PC oder CX (x86, x64)	Tc3_CM (>= 1.0.22), Tc3_CM_Base (>= 1.1.10), Tc3_MultiArray

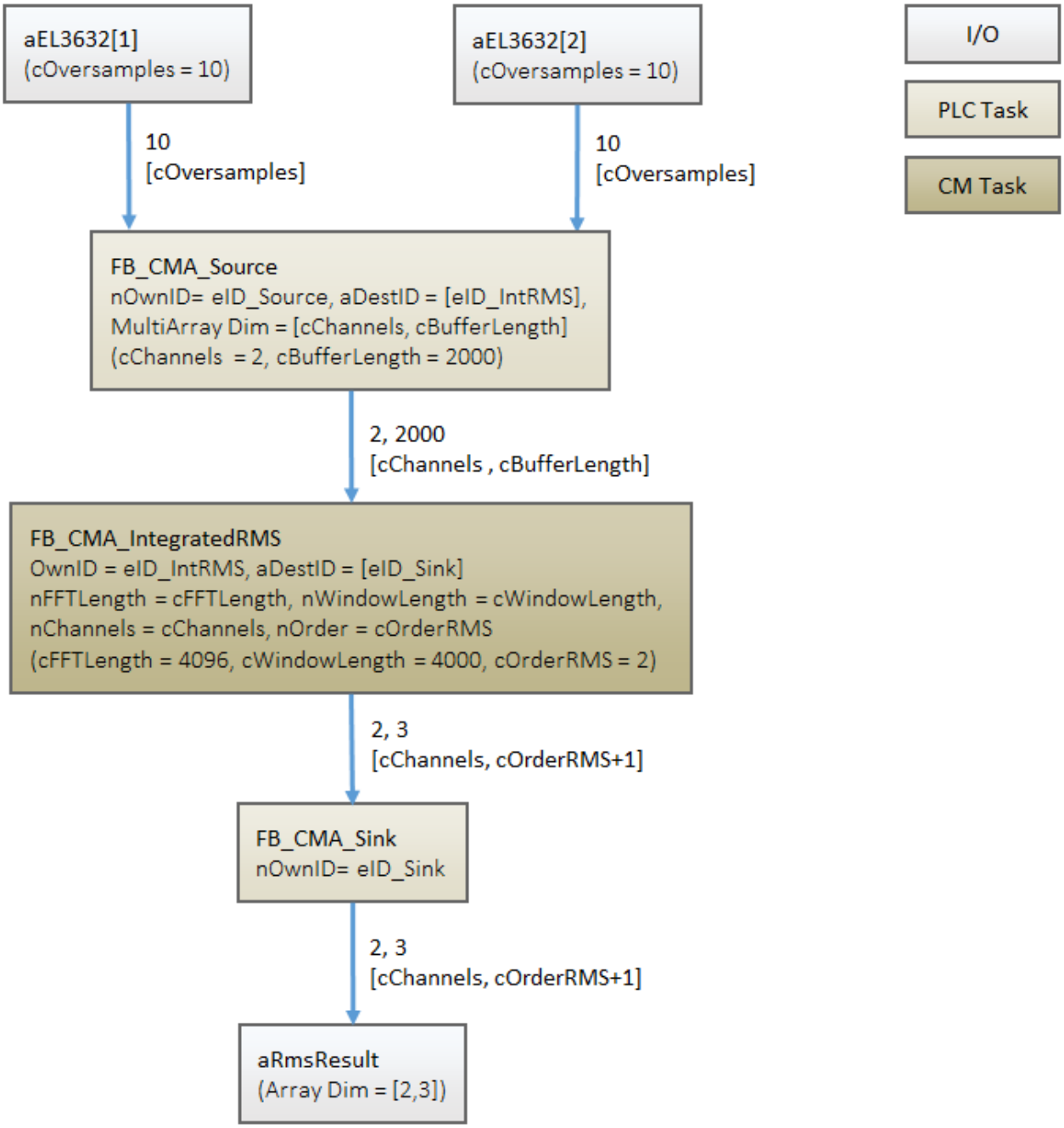
6.12 Schwingungsbeurteilung nach ISO 10816-3

Die Schwingungsbeurteilung in Anlehnung an ISO 10816-3 ist im Abschnitt Anwendungskonzepte näher erläutert, siehe Schwingungsbeurteilung [▶ 35]. Die Klassifizierung anhand der berechneten RMS Werte erfolgt direkt im MAIN Programm. Alternativ ließe sich dies mit den Funktionsbausteinen FB_CMA_WatchUpperThresholds [▶ 262] bzw. FB_CMA_DiscreteClassification [▶ 121].

Das Beispiel können Sie hier herunterladen:
https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/3394512523.zip

Eine alternative Umsetzung finden Sie im Beispiel Schwingungsbeurteilung nach ISO 10816-3 (kompakt) [▶ 349] und im Beispiel Schwingungsbeurteilung nach ISO 10816-3 (erweitert) [▶ 351].

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration der verwendeten Funktionsbausteine.

Puffergröße	2000
Kanäle	2
FFT Länge	4096
Fenstergröße	4000
Sampling-Rate	10000
Untere Frequenzschranke	10

Obere Frequenzschranke	1000
Ordnung (RMS)	2
Fenster Typ	eCM_HannWindow
Umwandeln in Dezibel	FALSE

Daten-Input

In dem Sample ist das Oversampling auf 10 und die mit den I/Os verknüpfte Task (PlcTask) auf 1 ms eingestellt. Daraus ergibt sich für den Daten-Input eine Abtastrate von 10 kHz. Entsprechend können nach dem Abtasttheorem nun Signale im Spektrum bis zu 5 kHz korrekt analysiert werden, vorausgesetzt der Anti-Aliasing-Filter in der I/O-Klemme ist korrekt eingestellt (siehe Digitalisierung [▶ 13]).

Puffern des Daten-Streams

Die Eingangsdaten der zwei Kanäle werden in der MAIN Routine mit einem Source-Baustein gepuffert. Entsprechend wird hier ein 2-Dimensionales Array der Größe [cChannels, cBufferLength] aufgebaut. Nach DIN ISO 10816-3 ist für eine Rotationsgeschwindigkeit über 600 min^{-1} ein Frequenzbereich von 10 Hz bis 1000 Hz auszuwerten. Die Frequenzauflösung der Frequenzanalyse (wird intern im IntegratedRMS Baustein gerechnet) sollte demnach deutlich unter 10 Hz liegen. Wird bei einer Abtastrate von 10 kHz ein Puffer von 4000 Samples genutzt, ergibt sich eine Frequenzauflösung von 2,5 Hz. Durch Nutzung des Hann-Fensters verringert diese sich formal auf 2,5 Hz * 1,5 = 3,75 Hz. Hinzu kommt, dass die FFT-Länge eine Potenz von 2 darstellen und größer sein muss als die WindowLength. Die BufferLength ergibt sich dann über die 50%ig überlappenden Fenster. Die Parametrisierung hinsichtlich der internen FFT wird entsprechend in der GVL_Constants wie folgt definiert:

```
cFFTLength     : UDINT  := 4096;            // length of FFT
cWindowLength  : UDINT  := 4000;            // 96 samples Zero padding
cBufferLength  : UDINT  := cWindowLength/2; // buffer due to 50% overlap
```

Entsprechend ergibt sich, wie im Schaubild oben gekennzeichnet, ein Array der Größe [2,2000] zur Übergabe an den FB_CMA_IntegratedRMS Baustein.

Frequenzselektive RMS-Wert Berechnung

Im Baustein FB_CMA_IntegratedRMS wird nun eine FFT berechnet und der RMS-Wert für den übergebenen Frequenzbereich (hier 10 Hz bis 1000 Hz) berechnet (formal können auch mehrere Bereiche angegeben werden). Der Baustein berechnet neben den RMS-Werten des direkten Eingangssignals (i.d.R. bei Anschluss eines Beschleunigungssensors ein Beschleunigungssignal) auch die jeweilig integrierten Größen, also RMS-Wert der Schwinggeschwindigkeit und RMS-Wert des Schwingungswegs. Der Ausgang des Bausteins ist entsprechend ein 2-dimensionales Array mit [2,3] (2 Kanäle, 3 RMS-Werte pro Kanal).

```
// define frequency interval according to ISO 10816-3
// e.g. 10 .. 1000 Hz for rotating speed over 600r/min
cfLowerFrequencyLimit : UDINT := 10;
cfUpperFrequencyLimit : UDINT := 1000;

// Parameters for RMS calculation
cOrderRMS   : UDINT := 2;              // acceleration, velocity, and displacement
cChannels   : UDINT := 2;              // ISO 10816-3 says 2 orthogonal sensors
cResult_Length : UDINT := cOrderRMS+1; // nOrder+1 (see InfoSys)
```

In den oben genannten Einstellungen benötigt der Source Baustein 2000/10=200 Zyklen mit 1 ms zum Füllen eines Puffers. Die Zykluszeit der PlcTask_CM sollte damit kleiner 0,5 * 200 ms liegen, siehe Task Einstellung [▶ 74]. Da der Baustein nur wenig Rechenzeit beansprucht, wird die Zykluszeit der PlcTask_CM zu 10 ms gewählt. Die Übergabe der Daten vom Source Baustein an den FB_CMA_IntegratedRMS über die Task-Grenzen hinweg wird von der Condition Monitoring Bibliothek intern behandelt.

Ergebnis Auswerten

Die Ergebnisse der RMS-Wert Berechnung werden durch einen Sink-Baustein wieder in die schnelle PlcTask mit 1 ms übergeben. Dazu muss lediglich in der MAIN Routine ein Array angegeben werden, welches zu der Größe des Arrays am Ausgang des FB_CMA_IntegratedRMS passt, siehe Variable `aRMSResult`.

Der Sink-Baustein setzt eine Flag auf TRUE, sobald ein gültiges Ergebnis berechnet worden ist, siehe Variable `bCalcuate`.

```
(* Push results to sink *)
fbSink.Output2D( pDataOut       := ADR(aRMSResult),
                 nDataOutSize   := SIZEOF(aRMSResult),
                 eElementType   := eMA_TypeCode_LREAL,
                 nWorkDim0      := 0,
                 nWorkDim1      := 1,
                 nElementsDim0  := 0,
                 nElementsDim1  := 0,
                 pStartIndex    := 0,
                 nOptionPars    := 0,
                 bNewResult     => bCalculate );
```

Dieses Flag kann entsprechend genutzt werden, um dann innerhalb der MAIN Routine mit dem Ergebnis der RMS-Wert Berechnung zu arbeiten. In diesem Fall werden die RMS-Werte der Schwinggeschwindigkeit und des Schwingwegs hinsichtlich der in der ISO definierten Grenzwerte überprüft – um das Sample einfach zu halten hier durch einfache IF-Abfragen.

Es wird für jeden der zwei Kanäle die Klasse nach ISO 10816-3 ermittelt und in den Variablen `ISOClassIs_Vel` (für die Klassifikation bezüglich der Schwinggeschwindigkeit) sowie `ISOClassIs_Displ` (für die Klassifikation bezüglich des Schwingwegs) abgespeichert. In diesem Beispiel entstehen so vier Einstufungen in Klassen. Nach ISO 10816-3 sollte bei orthogonal zueinander angeordneten Sensoren der größere der beiden Werte verwendet werden. Des Weiteren soll bei Verwendung von Schwingweg und Schwinggeschwindigkeit die strengere der Beurteilungen genutzt werden. Entsprechend wird im Quellcode der „worst case“ der vier Beurteilungen gesucht und als Output-Variable `ISO_10816_Classification` definiert.

Interaktion und Anmerkungen zum Sample

Im Sample sind zwei harmonische Schwingungen mit gleicher Amplitude (4 m/s^2) aber unterschiedlicher Frequenz (einmal 400 Hz und einmal 35 Hz) als Eingangsgrößen definiert. Während diese Beschleunigungsamplitude für eine Frequenz von 400 Hz für die Auswertung nach Schwingweg und Schwinggeschwindigkeit eine Einstufung in Klasse A bedeutet, ist diese Amplitude für eine Schwingung mit 35 Hz bei Auswertung des Schwingwegs in Klasse C und bei Auswertung der Schwinggeschwindigkeit sogar in Klasse D einzuordnen. Die Ausgangsvariable `ISO_10816_Classification` entspricht also dann Klasse D.

Verändern Sie die Amplitude der Schwingung mit 35 Hz auf 1 m/s^2, verringert sich die Klassifizierung auf B (für Schwinggeschwindigkeit) und A (für Schwingweg). Entsprechend wird die Variable `ISO_10816_Classification` auf B gesetzt.

Alternativ können Sie die Amplitude auf 4 m/s^2 belassen und die Frequenz anheben auf z. B. 800 Hz. Dadurch werden dann alle Einzel-Klassifikationen auf A eingestuft und somit die Variable `ISO_10816_Classification` auf A gesetzt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4013	PC oder CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

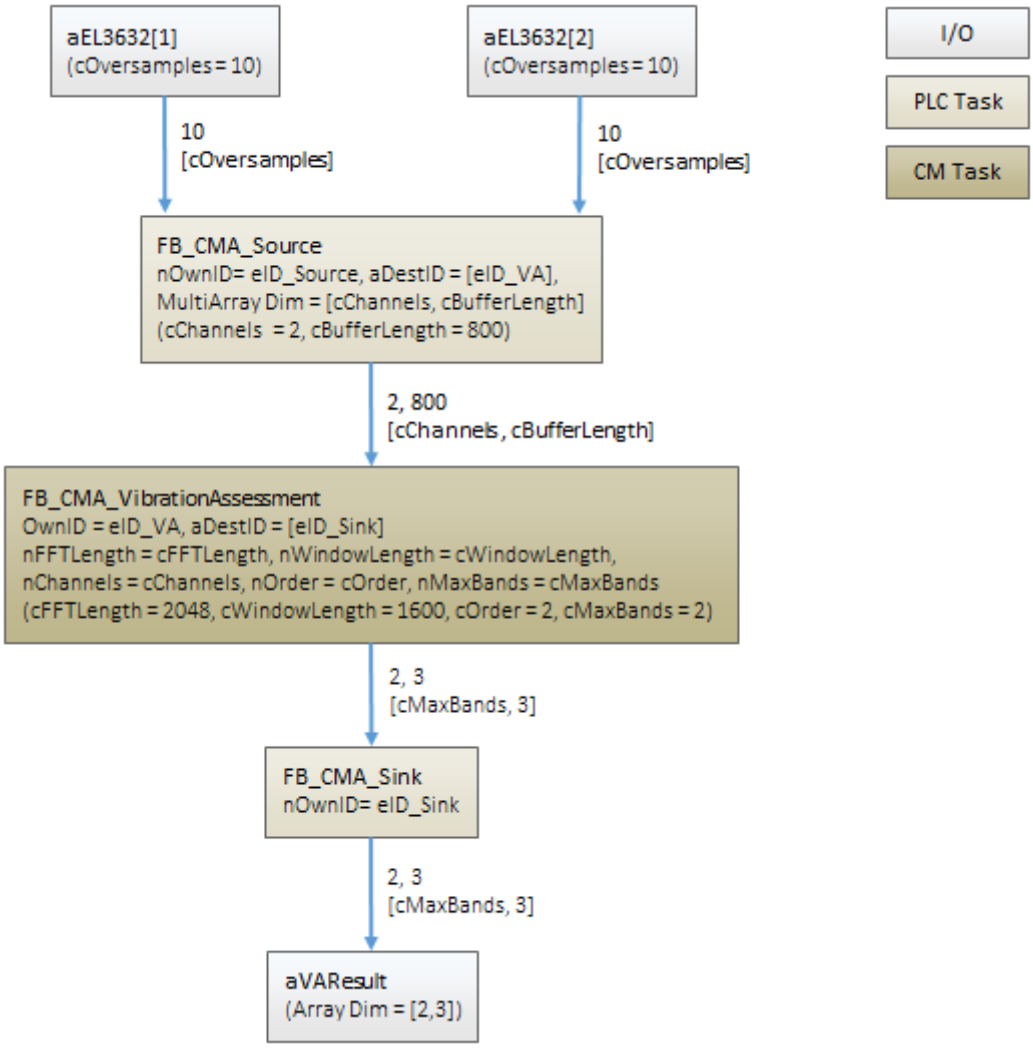
6.13 Schwingungsbeurteilung nach ISO 10816-3 (kompakt)

Die Schwingungsbeurteilung in Anlehnung an ISO 10816-3 ist im Abschnitt Anwendungskonzepte näher erläutert, siehe Schwingungsbeurteilung [▶ 35]. Die Klassifizierung erfolgt direkt durch den Funktionsbaustein FB_CMA_VibrationAssessment [▶ 258].

Das Beispiel können Sie hier herunterladen: https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/8236478987.zip

Eine alternative Umsetzung finden Sie im Beispiel Schwingungsbeurteilung nach ISO 10816-3 [▶ 346] und im Beispiel Schwingungsbeurteilung nach ISO 10816-3 (erweitert) [▶ 351].

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration der verwendeten Funktionsbausteine.

Puffergröße	800
Kanäle	2
FFT Länge	2048
Fenstergröße	1600
Sampling-Rate	10000
Anzahl der Frequenzbänder	2
Untere Frequenzschranke	[10, 200]
Obere Frequenzschranke	[1000, 2000]
Ordnung (RMS)	2
Fenster Typ	eCM_HannWindow
Umwandeln in Dezibel	FALSE

Konfiguration

Für die Maschinenüberwachung nach ISO 10816-3 werden die Schnelle und Auslenkung der Messdaten verwendet. Eine Klassifikation anhand der Beschleunigungsdaten wird nicht vorgenommen. Aus diesem Grund wird die Klassifikation implizit deaktiviert, indem die Grenzwerte auf einen ausreichend hohen Wert gesetzt werden (siehe `GVL_Constants`):

```
cISOClassDef_Vibration : ARRAY[1..cMaxClasses] OF LREAL := [1E6, 1E6, 1E6];
cISOClassDef_Velocity  : ARRAY[1..cMaxClasses] OF LREAL := [2.3E-3, 4.5E-3, 7.1E-3];
cISOClassDef_Displ     : ARRAY[1..cMaxClasses] OF LREAL := [29E-6, 57E-6, 90E-6];
```

Auswertung

Die Definition der Klassifizierung in Anlehnung an ISO 10816-3 basierend auf der Schnelle und Auslenkung für alle Kanäle erfolgt in der Steuerungstask. Ist ein neues Ergebnis verfügbar, wird dieses wie folgt ausgewertet:

```
IF bCalculate THEN
    // Highscore in classification according to ISO 10816-3.
    ISO_10816_HighscoreClass    := aVAResult[ISO_10816_nSelectedBand][1];  // class
    ISO_10816_HighscoreOrder    := aVAResult[ISO_10816_nSelectedBand][2];  // order
    ISO_10816_HighscoreChannel  := aVAResult[ISO_10816_nSelectedBand][3];  // channel

    IF NOT (ISO_10816_HighscoreClass = E_IsoClass.Error) THEN
        nCountResults := fbSink.nCntResults;
        // ToDo: if succeeded
    ELSE
        // ToDo: if error; RMS result is NaN. Code here what to do.
    END_IF
END_IF
```

Die Ergebnisdaten `aVAResult` beinhalten für alle konfigurierten Frequenzbänder folgende Informationen:

- `ISO_10816_HighscoreClass`: Die Klassifizierung (`A-D`) des Maschinenzustandes anhand der konfigurierten Grenzwerte.
- `ISO_10816_HighscoreOrder`: Die Integrationsordnung für die berechnete Bewertung, d.h. `0` für die Beschleunigung, `1` für die Schnelle und `2` für die Auslenkung.
- `ISO_10816_HighscoreChannel`: Der zugrundeliegende Kanal für die berechnete Bewertung.

In diesem Beispiel erfolgt die Auswahl des Frequenzbandes für die Bewertung über die Variable `ISO_10816_nSelectedBand`.

Gedächtniseigenschaft

Über den Initialparameter `bMemorize` kann die Gedächtniseigenschaft des Algorithmus geändert werden. Wird der Parameter auf `TRUE` gesetzt, wird die höchste Klassifikation nicht wieder durch eine niedrigere Bewertung überschrieben, bis die Methode `ResetData()` aufgerufen wird. Vergleiche hierzu das Verhalten des Funktionsbausteins FB_CMA_WatchUpperThresholds [▶ 262].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

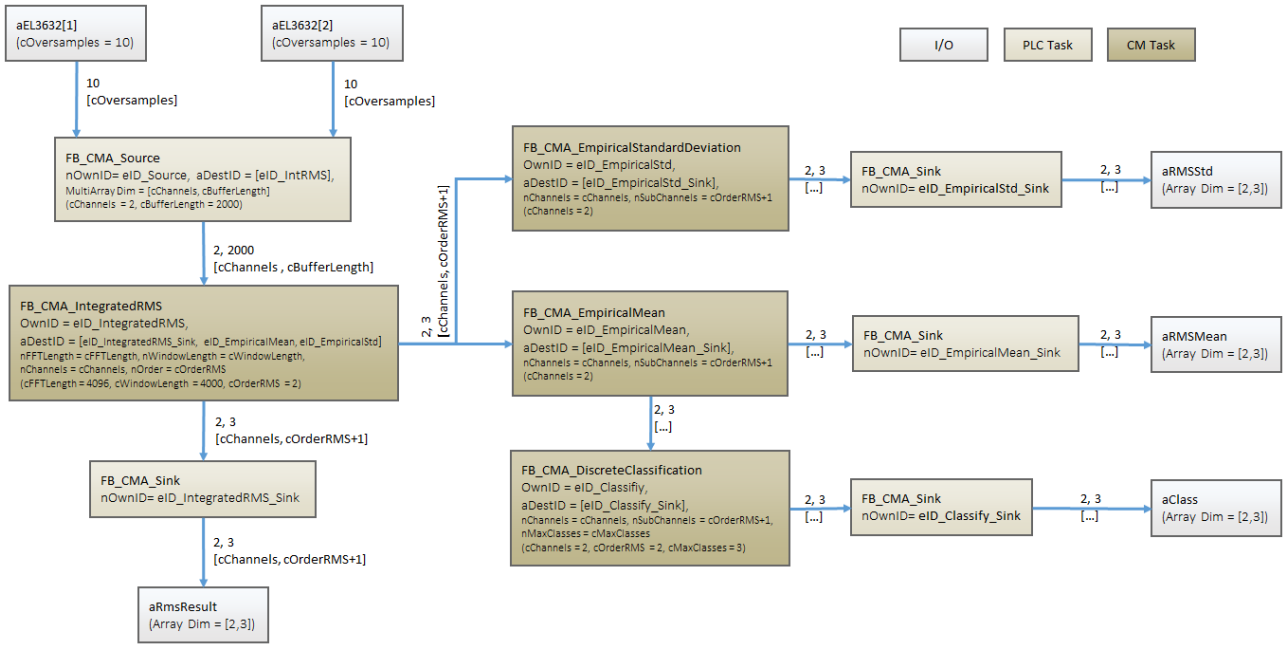
6.14 Schwingungsbeurteilung nach ISO 10816-3 (erweitert)

Die Schwingungsbeurteilung in Anlehnung an ISO 10816-3 ist im Abschnitt Anwendungskonzepte näher erläutert, siehe Schwingungsbeurteilung [▶ 35].

Im Vergleich zum Beispiel Schwingungsbeurteilung nach ISO 10816-3 [▶ 346] werden die berechneten (integrierten) RMS Werte zunächst mit Hilfe des Funktionsbausteins FB_CMA_EmpiricalMean [▶ 131] gemittelt. Die Klassifikation erfolgt hier auf Basis der Mittelwerte durch eine Instanz von FB_CMA_DiscreteClassification [▶ 121]. Durch die Verwendung von statistischen Daten ist die Maschinenbewertung stabiler als bei der direkten Verarbeitung der RMS Werte. Der gleiche Effekt könnte auch im oben genannten Beispiel erreicht werden, indem die Fensterlänge entsprechend angepasst wird, jedoch bei einem deutlich höheren Rechenaufwand.

Das Beispiel können Sie hier herunterladen: https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/5261534475.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration der verwendeten Funktionsbausteine.

Puffergröße	2000
Kanäle	2
FFT Länge	4096
Fenstergröße	4000
Sampling-Rate	10000
Untere Frequenzschranke	10
Obere Frequenzschranke	1000
Ordnung (RMS)	2
Klassen (Klassifizierung)	3
Fenster Typ	eCM_HannWindow
Datensätze (Statistik)	100
Reset nach Berechnung (Statistik)	`TRUE`

Konfiguration

Für die Maschinenüberwachung nach ISO 10816-3 werden die Schnelle und Auslenkung der Messdaten verwendet. Eine Klassifikation anhand der Beschleunigungsdaten wird nicht vorgenommen, wird aber in verwandten Normen verwendet, z.B. ISO 10816-21 für Windkraftanlagen. Die hier verwendeten Grenzwerte sind in der `GVL_Constants` definiert:

```
cISOClassDef_Acc        : ARRAY[1..3] OF LREAL := [1E6, 1E6, 1E6];
cISOClassDef_Velocity   : ARRAY[1..3] OF LREAL := [2.3E-3, 4.5E-3, 7.1E-3];
cISOClassDef_Displ      : ARRAY[1..3] OF LREAL := [29E-6, 57E-6, 90E-6];
```

Auswertung

Die berechneten RMS Werte für Schnelle und Auslenkung sowie die zugehörigen Mittelwerte und Streuung (Standardabweichungen) werden kanalweise in Instanzen von `ST_Channel` gespeichert. Für die Klassifikation werden ausschließlich die Mittelwerte herangezogen. Die Streuung kann als Kenngröße für die Vertrauenswürdigkeit für den Mittelwert gewertet werden. Die Das Ergebnis der Klassifikation wird anhand der konfigurierten Grenzwerte wird mittels `E_IsoClass` dargestellt. Die Ergebnisdaten werden über ein Scope Projekt visualisiert.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

Eingeschränkter Funktionsumfang bereits mit CM 3.1 verfügbar. Siehe Abschnitt Kompatibilität [▶ 66].

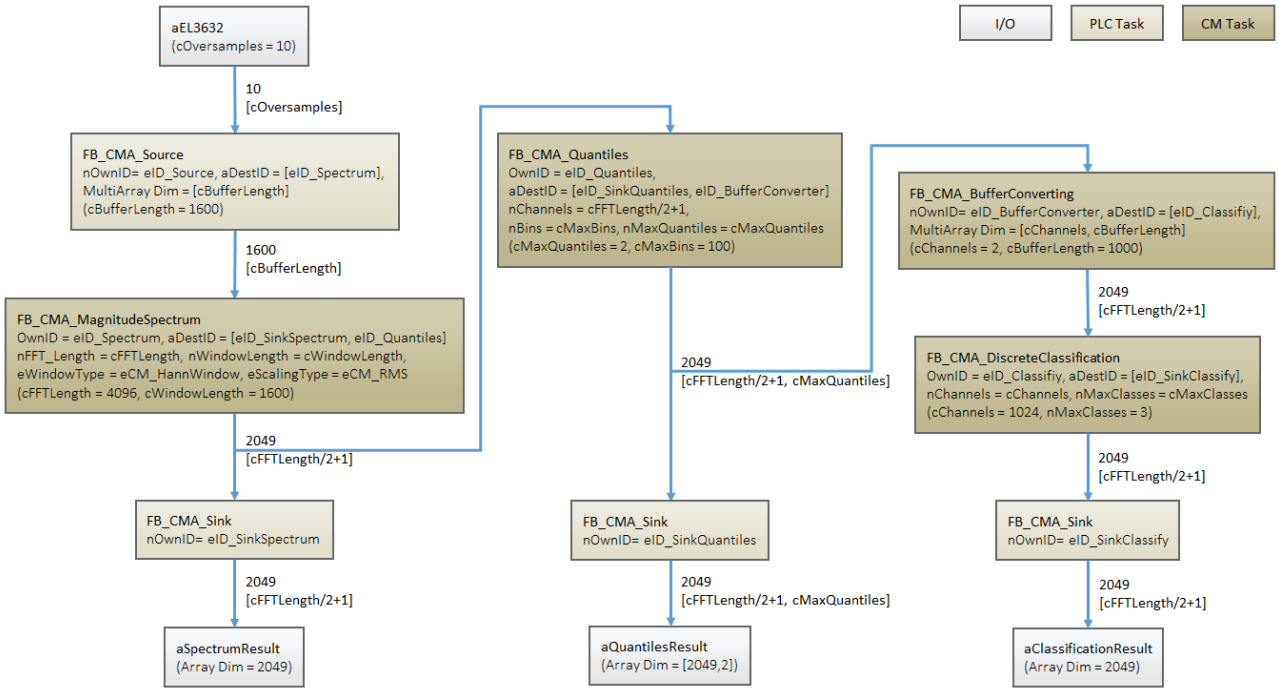
6.15 Condition Monitoring mit Frequenzanalyse

Dieses Lernprogramm baut eine komplette Überwachungsanwendung auf der Grundlage von TwinCAT 3 Condition Monitoring API auf. Von der Datensammlung bis zum Hinzufügen von leistungsstarken Analysealgorithmen hilft es dabei, einen Arbeitsfluss für Condition-Monitoring-Anwendungen zu erstellen. Das Blockdiagramm unten zeigt die schematische Anordnung der Anwendung. Zum besseren Verständnis der Programmierung ist das Dokument weiter in Designschritten unterteilt.

Das Beispiel können Sie hier herunterladen:
https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/3394571531.zip

Sie können es Ihren Wünschen entsprechend umgestalten und erweitern.

Blockdiagramm



Schritt 1: Anwendungsspezifikation

Der erste Schritt bei der Konzeption einer Zustandsüberwachungsanwendung besteht darin, die wichtigsten Ziele der Anwendung zu bestimmen, z.B. automatische Warnmeldung bei übermäßigen Vibrationen oder bei Erkennung einer Fehlfunktion im Lager mit Hilfe der Frequenzanalyse. Wichtig ist auch, andere technische Faktoren wie Messsensoren, Abtastrate des Reglers und erwartete Genauigkeit zu berücksichtigen. Bei diesem Lernprogramm besteht das Ziel darin, kleine und große Fehler im Eingangssignal mit Hilfe des Magnitudenspektrums und seiner Quantilenverteilung zu erkennen. Darüber hinaus wird ein Klassifizierer verwendet, um den allgemeinen Zustand als „Normalzustand“, „Warnzustand“ oder „Alarmzustand“ vorauszusagen. Die Tabelle unten zeigt eine Liste der in diesem Lernprogramm verwendeten Funktionsbausteine.

Funktionsbaustein
FB_CMA_Souce
FB_CMA_Sink
FB_CMA_MagnitudeSpectrum
FB_CMA_Quantiles
FB_CMA_DiscreteClassification

Für eine ausführlichere Beschreibung zwecks Auswahl des richtigen Algorithmus für spezifische Probleme, wie Lageranalyse [▶ 41], Getriebeanalyse [▶ 48] oder Frequenzanalyse [▶ 37] wird auf die an anderer Stelle beschriebenen Konzepte verwiesen. Da das Ziel des Lernprogramms in der Erkennung allgemeiner Veränderungen im Eingangssignal besteht, ist ein Magnitudenspektrum mit einer Auflösung von 4096 Linien ausreichend. Es werden die 50% und 90% Quantile der Spektralwerte berechnet und das Ergebnis gemäß der drei Zustände klassifiziert in "Normalzustand", „Warnzustand“ und "Alarmzustand“.

Schritt 2: Konfiguration der SPS-Tasks

Da die Zustandsüberwachung und –analyse sich aus einer Datenerfassungsstufe, Berechnungsstufe und einer Auswertungsstufe zusammensetzt, ist die Task entsprechend den Rechenanforderungen von jeder Stufe aufzugliedern. Unter Task Einstellung [▶ 74] finden Sie zusätzliche Informationen zu diesem Thema. In diesem Lernprogramm soll das Magnitudenspektrum von 4096 Linien berechnet werden, dazu sind ca. 3200 Datenerfassungen notwendig. Das bedeutet, in der Datensammlungsstufe muss eine Quellen-MultiArray

unter Berücksichtigung von Überlappung 1600 Datenerfassungen sammeln. Mit einer 10x Oversampling werden 160 Zyklen benötigt, oder 160ms bei 1ms Triggerung, um ein einziges Multiarray zu füllen. Aus diesem Grunde wird folgende Einstellung für die Berechnungstask empfohlen:

Berechnungszyklenzeit < (Datensammlungszyklenzeit * Puffergröße / Oversampling-Faktor)*0,5

Für das Lernprogramm ist die Berechnungszykluszeit auf 10ms festgelegt. Bei der realen Anwendung ist es wichtig, die Rechenlast zu berücksichtigen, die von anderen gleichzeitig auf derselben Steuerung ausgeführten Aufgaben wie Visualisierung, Netzwerkkommunikation, beeinflusst wird. Weitere Informationen zu Taskeinstellungen finden Sie im Abschnitt Taskzykluszeit [▶ 74]. Beachten Sie zudem, ausreichenden Routerspeicherplatz [▶ 73] vor dem Start des Aufbaus einer Condition-Monitoring-Anwendung zu reservieren. Dieses Lernprogramm wurde für das Arbeiten mit einem Routerspeicherplatz von 32 MB eingestellt.

Schritt 3: Konfiguration der Funktionsbausteine

Bei diesem Schritt werden die oben aufgeführten Funktionsbausteine entsprechend den Anwendungsanforderungen konfiguriert. Wie bereits erwähnt sammelt das Quell-MultiArray 1600 Datenerfassungen, um ein Spektrum zu berechnen. Darum wird der aDimSizes Parameter auf 1600 gesetzt. Da das Lernprogramm nur einen Kanal in Betracht zieht, wird nDims auf 1 gesetzt.

```
PROGRAM CM_Worker

VAR CONSTANT
      cInitSourceSpectrum       : ST_MA_MultiArray_InitPars := ( eTypeCode := eMA_TypeCode_LREAL, nD
ims := 1, aDimSizes := [1600]);
END_VAR
```

In der Berechnungstask wird das Magnitudenspektrum für die Berechnung eines Spektrums von 4096 Linien, angezeigt mittels cFFTLength, konfiguriert. Weil die Spektrumsberechnung mit periodischer Abarbeitung von diskreten Segmenten eines Dauersignals verbunden ist, wird eine sogenannte Fensterfunktion verwendet. Eine korrekt ausgewählte Fensterfunktion verbessert die Signaltransformationseffizienz, verringert die Schwankungen dank der overlapp-add Methode und verbessert die spektrale Auflösung. Zudem verringert die Fensterfunktion in praktischen Anwendungen den Leakage Effekt in der Nähe von kritischen Frequenzen. In dem Lernprogramm wurde das Hann Fenster gewählt. Der Magnitudenspektrum-Funktionsbaustein bietet viele Skalierungsoptionen [▶ 390], aus denen der Effektivwert (RMS) ausgewählt wurde. Der Grund hierfür ist, dass im Falle von in der Zeit veränderlicher physikalischer Signale ein Effektivwert ein bevorzugter Indikator der mittleren Leistung des Signals darstellt, im Vergleich z.B. zum Spitzenwert des Signals. Im Schwingungsbeschleunigungsspektrum zeigen einzelne Linien die Effektivwerte der Schwingungen bei entsprechender Frequenz an und können direkt in den entsprechenden Einheiten wie z.B. mm/s² oder g ausgedrückt werden.

```
PROGRAM MAIN_CM

VAR CONSTANT
     cInitSpectrum : ST_CM_MagnitudeSpectrum_InitPars := (      nFFT_Length := 4096,
                                    nWindowLength := 2*1600,
                                    bTransformToDecibel:= FALSE,
                                    eWindowType := eCM_HannWindow,
                                    eScalingType := eCM_RMS);
END_VAR
```

Das Ergebnis des Magnitudenspektrums wird über einen Sink Funktionsbaustein, mit vorgegebener Länge des Arrays auf nFFT_Length/2+1, in ein Array kopiert. Beim nächsten Schritt in der Analysekette wird ein Quantilfunktionsbaustein für die Berechnung der 50% und 90% Quantile der Spektralwerte konfiguriert. Die Spektralwerte sind häufig hochgradig fluktuierend, so dass eine Auswertung im Falle zu niedriger oder zu hoher Werte schwierig ist. Mit Hilfe der Quantilen kann das Maximum, Minimum oder sogar der Mittelwert über ein spezifiziertes Zeitintervall ermittelt werden. Diese Art der bereichsbasierten Auswertung ist häufig zuverlässiger und einfacher zu handhaben. Ein 50% Quantilwert $Q_{0.5}$ ist ein Wert, für den fast 50% der Werte einer Verteilung kleiner als $Q_{0.5}$ sind. Er wird auch als Medianwert bezeichnet. Ähnlich gilt, dass eine 90% Quantile $Q_{0.9}$ einen Wert anzeigt, für den 90% der Werte einer Verteilung kleiner als $Q_{0.9}$ sind.

```
VAR CONSTANT
     cInitQuantiles : ST_CM_Quantiles_InitPars := ( nChannels := (4096/2+1),
                              fMinBinned := -10,
                              fMaxBinned := 10,
                              nBins := 100,
                              nMaxQuantiles := 2);
END_VAR
```

Im Programm werden die Quantile folgendermaßen konfiguriert:

```
(*--------- Configure quantile args ---------*)
IF bConfigureQuantile THEN
     FOR nChannel := 1 TO (cFFTLength/2+1) DO
          aQuantilesArg[nChannel,1]:= 0.50; // 50% quantile
          aQuantilesArg[nChannel,2]:= 0.90; // 90% quantile
     END_FOR
     fbQuantiles.Configure( pArg := ADR(aQuantilesArg), nArgSize := SIZEOF(aQuantilesArg));
     bConfigureQuantile := FALSE;
END_IF
```

Hier finden Sie die ausführliche Beschreibung des Funktionsbausteins FB_CMA_Quantiles [▶ 206]. Beachten Sie, dass die Parameter fMinBinned und fMaxBinned den erwarteten Bereich des Eingangssignals definieren und nBins die Anzahl der Bins darstellt, in die der Bereich des Signals unterteilt wird. Diese Parameter hängen vom jeweiligen Eingangssignal ab. Der Zustand des Signals wird auf der Grundlage der Quantileninformation klassifiziert. Der diskrete Funktionsbaustein kann mehrere Kanäle gleichzeitig abarbeiten, deswegen wird der Ausgang der Quantilen ihm direkt zugeführt. Der Klassifizierer ist darauf eingestellt, zwischen drei Zuständen zu unterscheiden, und den entsprechenden Zustand über den nMaxClasses Parameter anzuzeigen.

```
VAR CONSTANT
     cInitClassification    : ST_CM_DiscreteClassification_InitPars := (nChannels:= (4096/2+1),
                                        nMaxClasses := 3);
END_VAR
```

Anmerkung: Der Ausgang des Quantilenfunktionsbausteins ist ein 2D-Array, das in diesem Fall die Anzahl der Spektrallinien über die Anzahl Quantilen ist. Aber der diskrete Klassifizierer erlaubt lediglich ein eindimensionales Array, das die Anzahl der Eingangskanäle enthält. Um einen Konflikt der Dimension zu vermeiden, ist der Pufferumwandler von FB_CMA_BufferConverting zu verwenden. Dieser Funktionsbaustein wandelt ein zweidimensionales Multiarray in ein eindimensionales Array ohne Datenverlust um. Der Code-Ausschnitt beschreibt die entsprechende Verwendung.

```
VAR CONSTANT
     cInitBuffer       : ST_MA_MultiArray_InitPars := ( eTypeCode := eMA_TypeCode_LREAL,
                                   nDims := 1,
                                   aDimSizes := [(4096/2+1)]);
END_VAR
VAR
     fbBufferConverter : FB_CMA_BufferConverting := (stInitPars := cInitBuffer, nOwnID := eID_Buffer
Converter, aDestIDs := [eID_Classify]);
END_VAR
```

Der Pufferumwandler ruft eine Methode auf:

```
fbBufferConverter.Copy1D(nWorkDimIn := 0,
               nWorkDimOut := 0,
               nElements := 0,
               pStartIndexIn := 0,
               pStartIndexOut := 0,
               nOptionPars := 0);
```

Weitere Informationen zu diesem Funktionsbaustein finden Sie unter FB_CMA_BufferConverting [▶ 96]. Zum Abschluss der Konfiguration des Funktionsbausteins muss jeder Sink Funktionsbaustein mit SPS-Arrays mit den korrekten Dimensionen verbunden werden.

Schritt 4: Feinabstimmung der Anwendungsparameter

Vor Beginn der Analyse, ist es wichtig den diskreten Klassifizierer bezüglich seiner Grenzwerte zu konfigurieren. Eine Klassifizierungsgrenze oder ein Schwellenwert ermöglicht es dem diskreten Klassifizierer, eingehende Kanäle ständig zu überwachen und festzustellen, ob einer der Eingangskanäle diesen Schwellenwert überschreitet. Die Schwellenwerte hängen von der jeweiligen Anwendung, den Genauigkeitsanforderungen, den zulässigen Detektionstoleranzen, usw. ab. In diesem Lernprogramm besteht das Ziel darin, kleine Fehler, vergleichbar mit Zufallsrauschen, und auch große Fehler, die bei einer bestimmten Frequenz (z.B. 200 Hz) auftreten, zu erkennen. Es werden die Schwelllwerte fWarning und fAlarm bestimmt. Überschreitet die Amplitude der Eingangskanäle fWarning, wechselt der allgemeine Zustand in den Warnzustand, wird fAlarm überschritten, erfolgt eine Alarmmeldung. Werden die Schwellwerte nicht überschritten befindet sich der Kanalzustand im normalen Bereich.

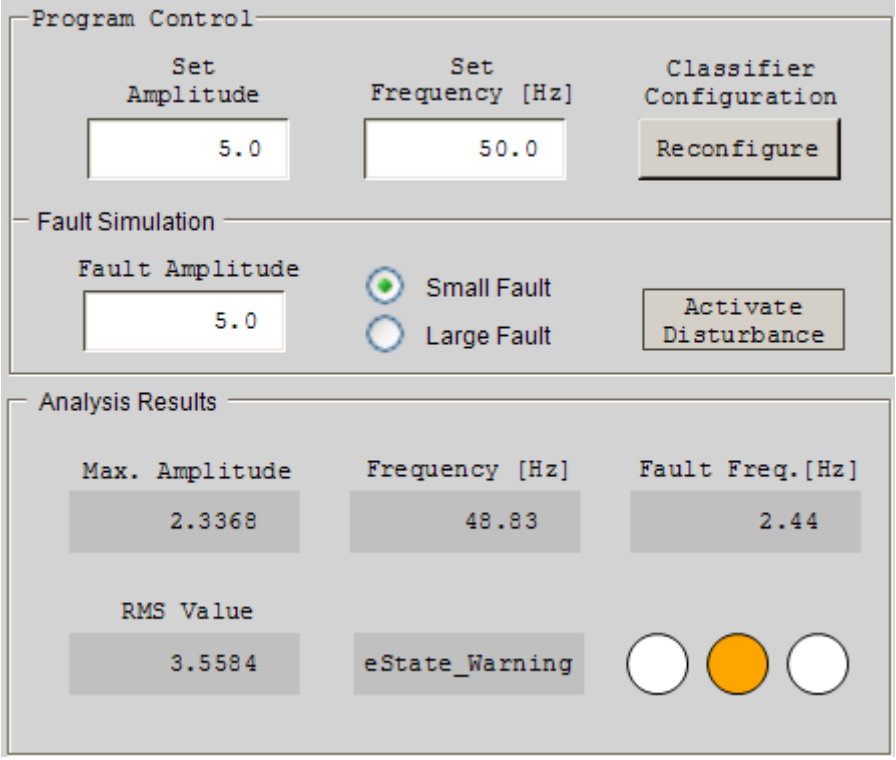
```
(*--------- Configure classifier args ---------*)
IF bConfigureClassifier THEN

   fWarning := (fMonitoringLevel/100)*1.5;
   fAlarm := (fMonitoringLevel/100)*2.5;
```

```
   fbTeachTimer(IN := TRUE, PT := T#15S);
   IF fbTeachTimer.Q THEN

      fbTeachTimer(IN := FALSE);

      FOR nChannel := 1 TO (cFFTLength/2+1) DO
     aClassArgs[nChannel, 1] := (fMonitoringLevel/100)*aQuantilesCopy[nChannel,1];
     aClassArgs[nChannel, 2] := fWarning*aQuantilesCopy[nChannel,1];
     aClassArgs[nChannel, 3] := fAlarm*aQuantilesCopy[nChannel,1];
      END_FOR

      fbClassification.Configure(pArg := ADR(aClassArgs), nArgSize := SIZEOF(aClassArgs));
      bConfigureClassifier := FALSE;

   END_IF

END_IF
```

Der Code-Ausschnitt oben beschreibt die Konfiguration von diesem diskreten Klassifizierer, sodass ein Zeitgeberblock ein normales Betriebsfenster in die Lage versetzt, eine sogenannte Lernphase, in welcher der diskrete Klassifizierer konfiguriert wird, zu durchlaufen. Es wird vorausgesetzt, dass sich das Eingangssignal im Verlauf dieser Zeit normal verhält, d.h. innerhalb des zulässigen Bereichs verbleibt. Der Warnschwellenwert liegt bei 150% der “normalen” 50% Quantile und die Alarmschwelle bei 250% der normalen 50% Quantile. Da die 50% Quantile das durchschnittliche Verhalten beschreibt, eignet sich diese Schwellenwertkonfiguration für Anwendungen, deren Eingänge nur wenige Ausreißer aufweisen. Es kann auch die 90% Quantile als Schwellenwert bestimmt werden, wenn angenommen wird, dass das Eingangssignal wahrscheinlich stark schwankt. Es kann ebenfalls eine andere Variable, fMonitoringLevel, konfiguriert werden, dank derer ein bestimmtes Toleranzband um die zulässigen Werte gelegt wird, um die Anzahl von falschen Alarmen zu kontrollieren. Mit Hilfe dieses Parameters können die Schwellenwerte genauer abstimmt werden. Beachten Sie, dass die Schwellenwerte für den diskreten Klassifizierer individuell für alle Eingangskanäle festgelegt werden können.

Schritt 5: Anwendung starten

Kompilieren Sie den Code, laden Sie ihn auf das Zielsystem herunter und starten Sie die SPS, um das Lernprogramm auszuführen. Im Solution Explorer unter dem Knoten von VISU befindet sich eine kleine vorbereitete Visualisierung, genannt Dashboard die für einen Schnelltest verwendet werden kann. Für die Simulation wird das Eingangssignal mit einem Funktionsgenerator verbunden, der für die Erzeugung eines sinusförmigen 50Hz Signals mit einer Amplitude von 5 konfiguriert wurde. Sie können am Eingang andere verfügbare Signale, wie ein Impuls-, Dreieck- oder Sägezahnsignal oder aber auch ein Hardwaremodul wie EL3632 anlegen. Nach dem Start der Anwendung werden in den Displayfeldern die maximale Amplitude, die RMS-Amplitude und die Frequenz bei der maximalen Amplitude von der SPS in Echtzeit angezeigt.



In der Abbildung sehen Sie, dass der Zustand der Anwendung im entsprechenden Displayfeld angezeigt wird. Sie können einen geringfügigen Fehler simulieren, indem Sie auf die Add Fault Taste drücken. Sie können beobachten, wie der RMS-Wert des Eingangssignals langsam über den Schwellenwert ansteigt und der Zustand sich entsprechend ändert. Um einen großen Fehler zu simulieren, drücken Sie auf Small/Large Taste. Ähnlich wie im Falle des vorherigen Fehlers wird der RMS-Wert ansteigen, aber diesmal wird im „Fault Frequency“ Feld die Frequenz des Fehlersignals, die in diesem Falle 200Hz beträgt, angezeigt.

Schritt 6: Monitoring

Nach dem Start der SPS befinden sich in den Anzeigefeldern die aktuellen Werte. Anfangs erscheint die Reconfigure Taste als gedrückt und das Signal in der rechten Ecke ist deaktiviert. Das bedeutet, dass die Grenzwerte gerade für den diskreten Klassifizierer festgelegt werden. Nachdem die Konfiguration abgeschlossen ist, wird die Reconfigure Taste sich selbt zurücksetzen und der Zustand der Maschine wird als „Normalzustand“ angezeigt. Das Signal wechselt zu grün, was die gleiche Bedeutung hat.

Um einen Fehler zu simulieren, belassen Sie das Optionsfeld auf „Small Fault“ und drücken die Activate Disturbance (Störung aktivieren) Taste. Die Maschine wird in Folge zwischen „Normal-“ und „Warnzustand“ wechseln, und das Signal wird abwechselnd grün und orange dargestellt. Wird ein großer Fehler simuliert, indem das Optionsfeld umgeschaltet wird, dann wechselt der Maschinenzustand zum „Alarmzustand“ und das Signal zeigt rot an. Um die Störung zu unterbinden, lassen Sie die Activate Disturbance Taste los. Der Signalzustand kehrt wieder zu grün zurück. Beachten Sie, dass eine Änderung der Signalamplitude ebenfalls einen Fehlerzustand zur Folge hat. Wenn dies unerwünscht ist, dann drücken Sie die Reconfigure Taste erneut, um den diskreten Klassifizierer an diesen neuen Signalzustand anzupassen.

Voraussetzungen

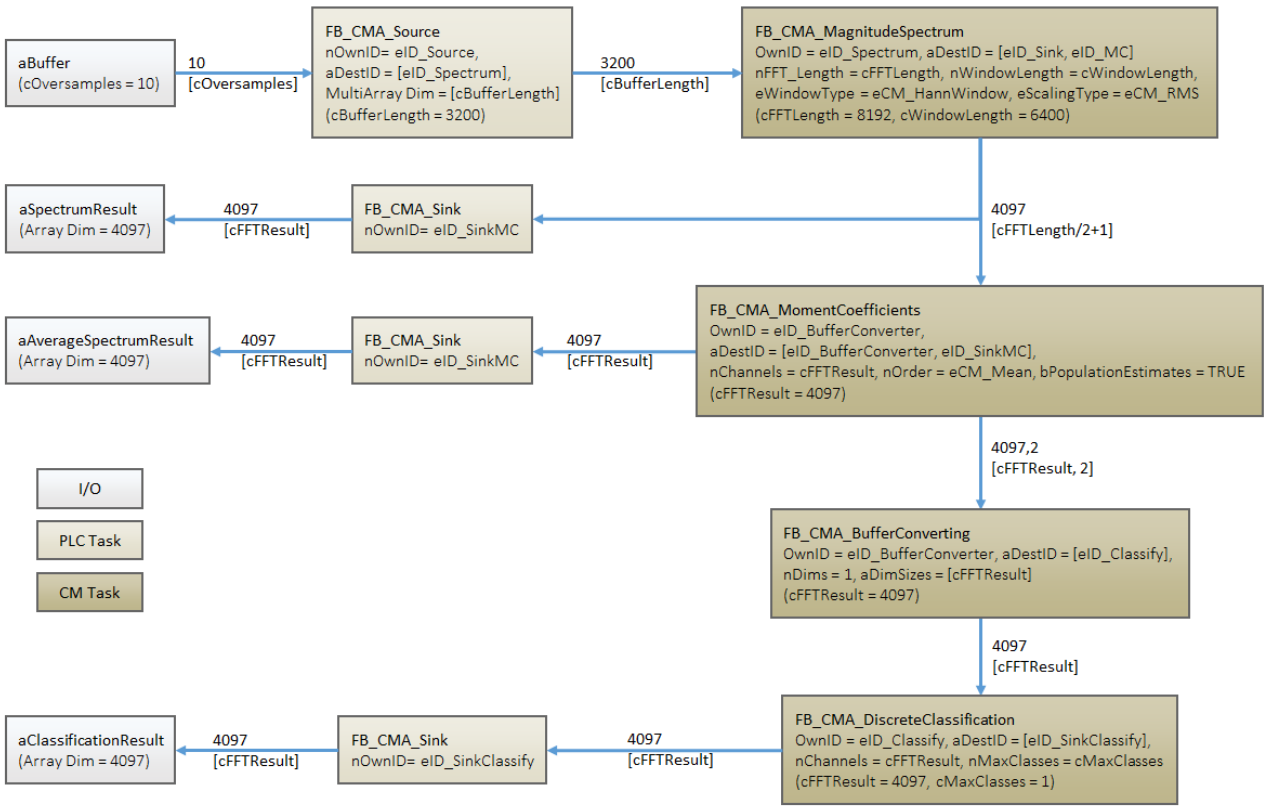
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4013	PC oder CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.16 Schwellwertbetrachtung gemittelte Betragsspektren

In diesem Beispiel wird, wie im Anwendungskonzept Frequenzanalyse [▶ 37] näher erläutert, eine Analysekette zur Schwellwertbetrachtung betrachtet. Die Analysekette implementiert dabei die Berechnung eines Betragsspektrums, Mittelung von mehreren Betragsspektren und anschließende Schwellwertbetrachtung für exemplarische Frequenzbänder. Zur besseren Veranschaulichung der Schwellwertbetrachtung um 50 Hz ist der Scope auf den Frequenzbereich von 0 Hz bis 100 Hz beschränkt.

Das Beispiel können Sie hier herunterladen:
https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/3394495883.zip

Blockdiagramm der Analysekette:



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration der verwendeten Funktionsbausteine.

FFT Länge	8192
Fenstergröße	6400
Puffergröße	3200
Fenster Typ	eCM_HannWindow
Skalierungsart	eCM_RMS
Koeffizientenordnung	eCM_Mean
Maximale Anzahl Klassen	1

Voraussetzungen

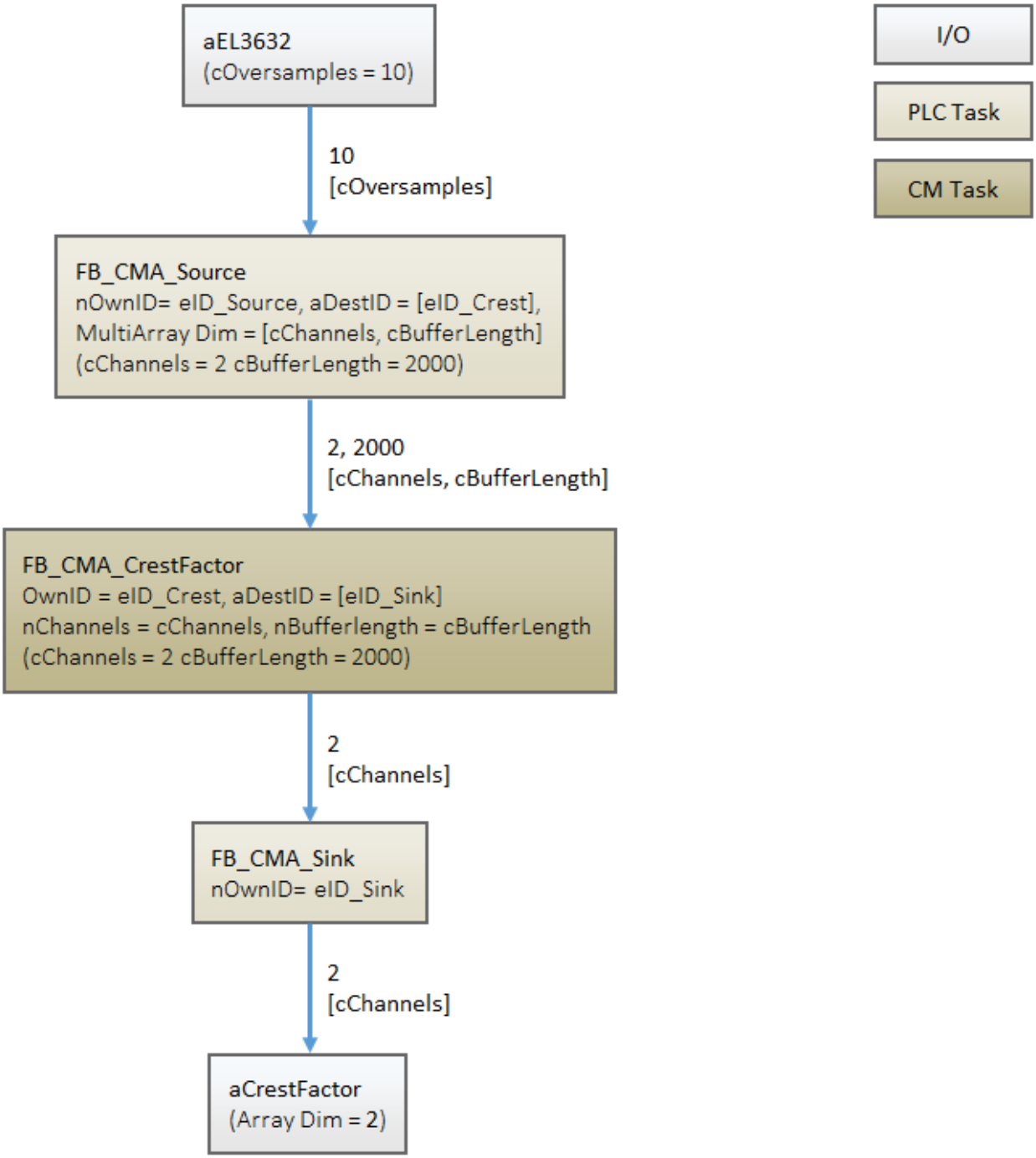
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4013	PC oder CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.17 Crest Faktor

Dieses Beispiel berechnet den Crest Faktor eines Eingangssignals. Auch wenn der Funktionsbaustein FB_CMA_CrestFactor [▶ 99] in der Lage ist, mehrere Kanäle zu verarbeiten, wird zwecks Veranschaulichung nur ein Einzelkanal betrachtet. Das Blockdiagramm unten zeigt die im Programm implementierte Analysekette.

Den Quellcode für das Bespiel können Sie hier herunterladen:
https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/3394544267.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration des Bausteins zur Berechnung des Scheitelfaktors.

Kanäle	2
Puffergröße	1600

Globale Konstanten

Diese Parameter werden in der Liste der globalen Variablen als Konstanten definiert.

```
VAR_GLOBAL CONSTANT
    cOversamples       : UDINT := 10;    // oversampling factor
    cChannels          : UDINT := 2;     // number of channels
    cBufferLength      : UDINT := 2000;  // size of buffer
END_VAR
```

Code für Steuerungstask

Der folgende Code-Ausschnitt zeigt die Deklaration im MAIN Programm:

```
VAR CONSTANT
     cInitSource       : ST_MA_MultiArray_InitPars := ( eTypeCode := eMA_TypeCode_LREAL, nDims := 2,
 aDimSizes := [cChannels, cBufferLength]);
END_VAR

VAR
     nInputSelection   : UDINT := 1;
     aCrestFactor      : ARRAY[1..cChannels] OF LREAL;
     nSampleIdx        : UDINT;
     nChannelIdx       : UDINT;
     aEl3632 AT %I*    : ARRAY[1..cChannels] OF ARRAY[1..cOversamples] OF INT; // input from hardwar
e e.g. EL3632
     aBuffer           : ARRAY[1..cChannels] OF ARRAY[1..cOversamples] OF LREAL;
     fbSource          : FB_CMA_Source := (stInitPars := cInitSource, nOwnID := eID_Source, aDestIDs
 := [eID_Crest]); // Initialize source
     fbSink            : FB_CMA_Sink := (nOwnID := eID_Sink);
END_VAR
```

Methode ruft Main Programm auf:

```
// Collect data in a source
fbSource.Input2D(pDataIn := ADR(aBuffer),
          nDataInSize := SIZEOF(aBuffer),
          eElementType := eMA_TypeCode_LREAL,
          nWorkDim0 := 0,
          nWorkDim1 := 1,
          pStartIndex := 0,
          nOptionPars := 0 );

// Push results to sink
fbSink.Output1D(pDataOut := ADR(aCrestFactor),
          nDataOutSize := SIZEOF(aCrestFactor),
          eElementType := eMA_TypeCode_LREAL,
          nWorkDim := 0,
          nElements := 0,
          pStartIndex := 0,
          nOptionPars := 0,
          bNewResult => bNewResult);
```

Code für CM-Task

Deklaration im MAIN_CM Programm:

```
VAR CONSTANT
     cInitCrest : ST_CM_CrestFactor_InitPars := ( nChannels := cChannels, nBufferLength := cBufferLe
ngth );
END_VAR

VAR
     fbCrest : FB_CMA_CrestFactor := (stInitPars := cInitCrest, nOwnID:= eID_Crest, aDestIDs:= [eID_
Sink]); // Initialize crest
END_VAR
```

Methode ruft MAIN_CM Programm auf:

```
fbCrest.Call();
```

Das Ergebnis des Beispielcodes kann für ein sinusförmiges Signal beliebiger Amplitude und Frequenz als Eingangssignal getestet werden. Der Scheitelfaktor, in diesem Falle das erste Element von aCrestFactor, muss gleich 3,01 dB sein.

Voraussetzungen

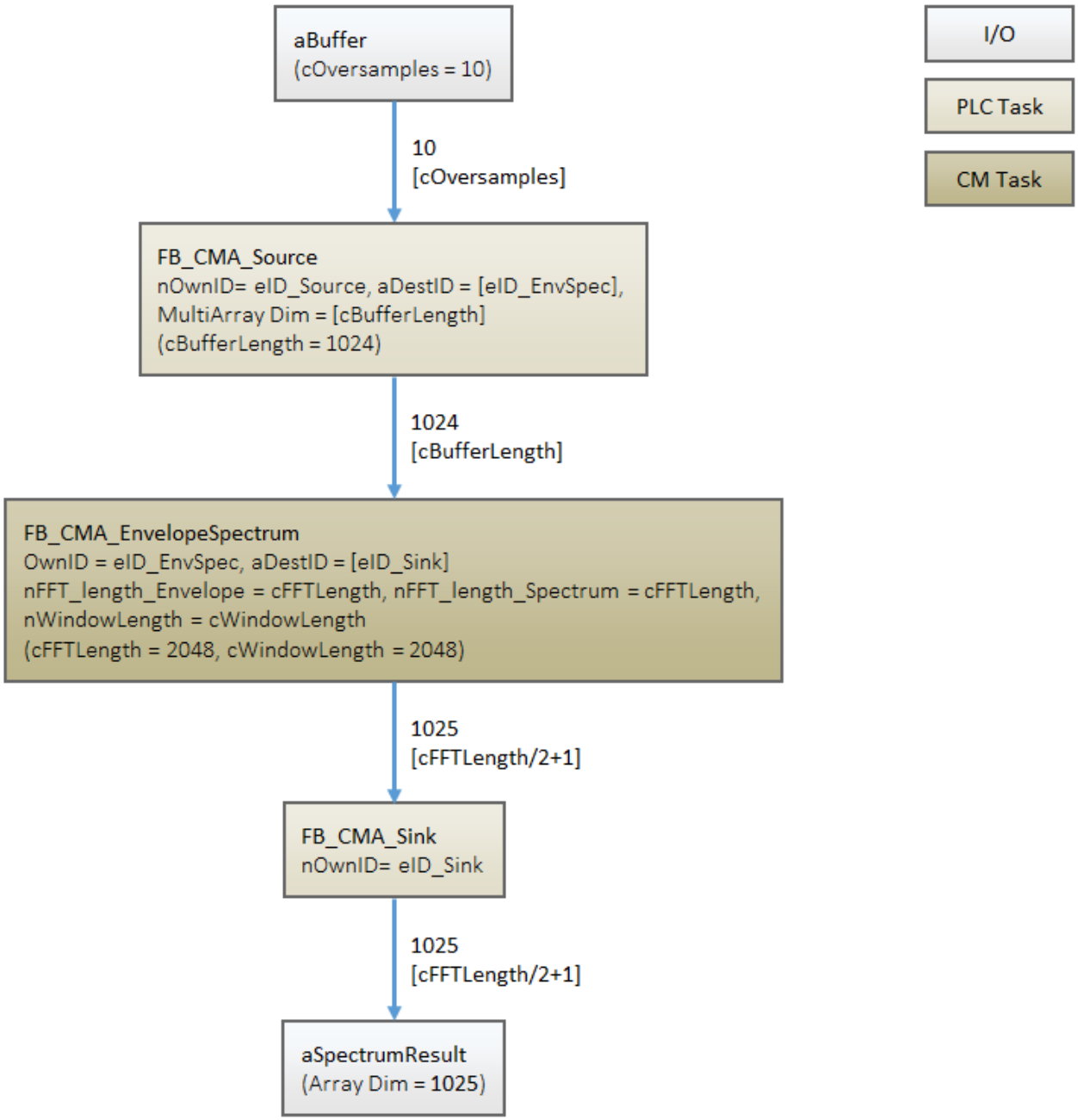
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4013	PC oder CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.18 Einhüllendenspektrum

Das Beispiel zeigt eine exemplarische Implementierung der Berechnung eines Einhüllendenspektrums mit dem Baustein FB_CMA_EnvelopeSpectrum [▶ 150]. Das Eingangssignal wird mittels eines Funktionengenerators generiert und entspricht der Überlagerung zweier Sinus-Schwingungen mit 120 Hz bzw. 230 Hz. Zur besseren Veranschaulichung des Ergebnisses ist der Scope auf den Frequenzbereich von 0 Hz bis 300 Hz beschränkt.

Das Beispiel können Sie hier herunterladen:
https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/3394494219.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration des Funktionsbausteins zur Berechnung des Einhüllendenspektrums.

FFT Länge Einhüllende	2048
FFT Länge Spektrum	2048
Fenstergröße	2048
Umwandeln in Dezibel	FALSE
Fenster Typ	eCM_HannWindow
Skalierungsart	eCM_RMS

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4013	PC oder CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

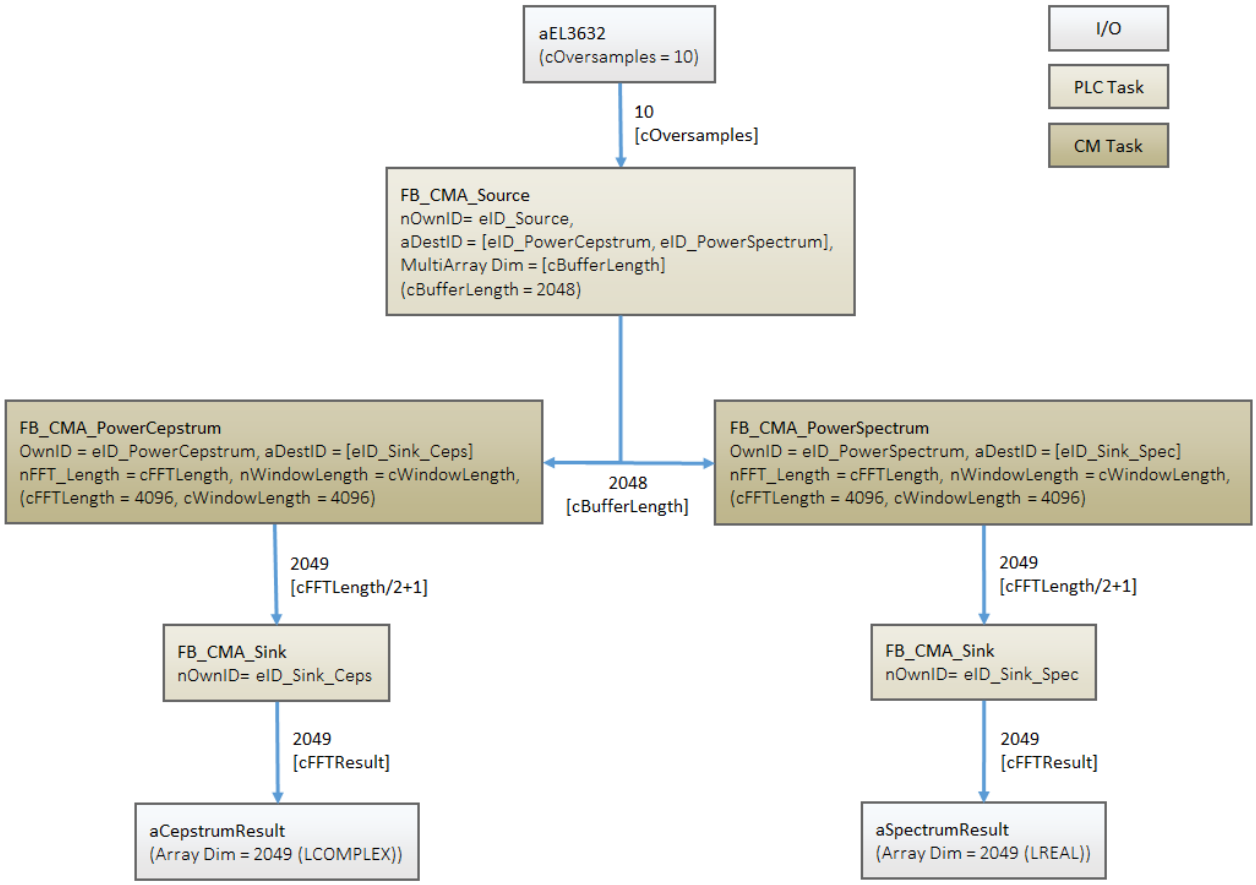
6.19 Leistungscepstrum

Dieses Beispiel implementiert exemplarisch die Berechnung von Leistungscepstrum und Leistungsspektrums. Das betrachtete Signal wird basierend auf zwei Sinunsschwingungen, einer Trägerfrequenz und einer Modulationsfrequenz, mittels Amplitudenmodulation erzeugt.

Den Quellcode für das Beispiel können Sie hier herunterladen:

https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/5261531147.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration der verwendeten Funktionsbausteine.

FFT Länge	4096
Fenstergröße	4096
Puffergröße	2048

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4018	PC oder CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

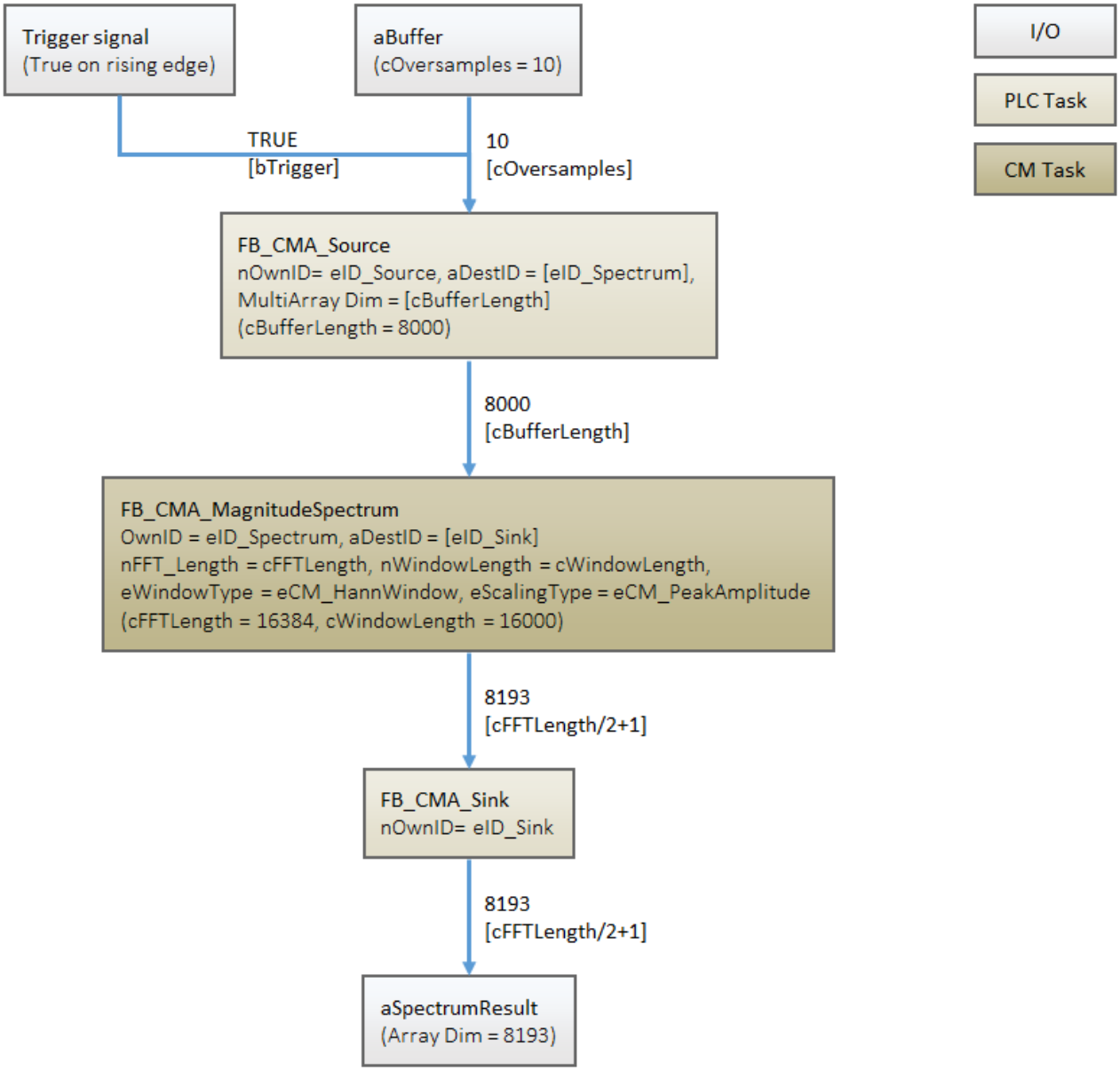
6.20 Eventbasierte Frequenzanalyse

Dieses Beispiel implementiert eine eventbasierte Frequenzanalyse. Das generierte Signal besteht aus einem im 2 s Takt wechselnden verrauschten Sinussignal mit einer Frequenz von 200 Hz und reinem Rauschen. Wird im (generierten) Eingangssignal eine steigende Flanke detektiert, beginnt die Pufferung des Signals. Die gesammelten Daten werden anschließend über FB_CMA_Source [▶ 245] an den Funktionsbaustein FB_CMA_MagnitudeSpectrum [▶ 172] weitergeleitet.

Den Quellcode für das Beispiel können Sie hier herunterladen:

https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/5261425419.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration des Magnitudenspektrum-Funktionsbausteins.

FFT Länge	16384
Fenstergröße	16000
Puffergröße	8000
Fenster Typ	eCM_HannWindow
Skalierungsart	eCM_ PeakAmplitude

Eventbasierte Pufferung des Eingangssignals

Der Programmbaustein `CollectData` steuert die Eventbasierte Aufnahme des Eingangssignals. Die Input-Parameter sind wie folgt definiert:

```
PROGRAM CollectData
VAR_INPUT
    bTrigger     : BOOL;     // Trigger signal, start with rising edge
END_VAR
VAR_IN_OUT
    aInputSignal : ARRAY[1..cOversamples] OF LREAL;  // input time signal
END_VAR
```

Die Umkehrung des Triggersignals `bTrigger_` sowie der aktuelle Zustand des Puffers werden lokal abgelegt.

```
VAR
    bTrigger_           : BOOL := FALSE;
    nSourceState        : UINT := 0;
    nActualBuffersSent  : ULINT := 0;
    nBuffersToSent      : ULINT := 2;

    // ...

END_VAR
```

Die Steuerung der eventgesteuerten Aufnahme des Signals erfolgt, falls das Triggersignal eine steigende Flanke aufweist und der Puffer bereit ist, d.h. Zustand `0`.

```
IF (bTrigger AND NOT bTrigger_) AND nSourceState = 0 THEN
    nActualBuffersSent := fbSource.nCntResults;  // check number of sent MultiArrays from fbSource
    fbSourceState := 1;
END_IF
bTrigger_ := bTrigger;
```

Der folgende Code zeigt die tatsächliche eventbasierte Pufferung des Signals über den Source Baustein.

```
CASE nSourceState OF

    1: // if <nBuffersToSent> MultiArrays has been sent, stop buffering

       fbSource.Input1D( pDataIn      := ADR(aInputSignal),
                         nDataInSize  := SIZEOF(aInputSignal),
                         eElementType := eMA_TypeCode_LREAL,
                         nWorkDim     := 0,
                         pStartIndex  := 0,
                         nOptionPars  := 0);

       IF (fbSource.nCntResults-nActualBuffersSent) = nBuffersToSent THEN
           nSourceState := 2;
       END_IF

    2: // reset Source Buffer and wait for next trigger hit

       fbSource.ResetData();
       nSourceState := 0;

END_CASE;
```

Im Folgenden werden die gepufferten Signaldaten an den Magnitudenspektrum-Funktionsbaustein weitergeleitet. Die Verarbeitung des gepufferten Signals verläuft analog wie im Beispiel Magnitudenspektrum [▶ 329].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4018	PC oder CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

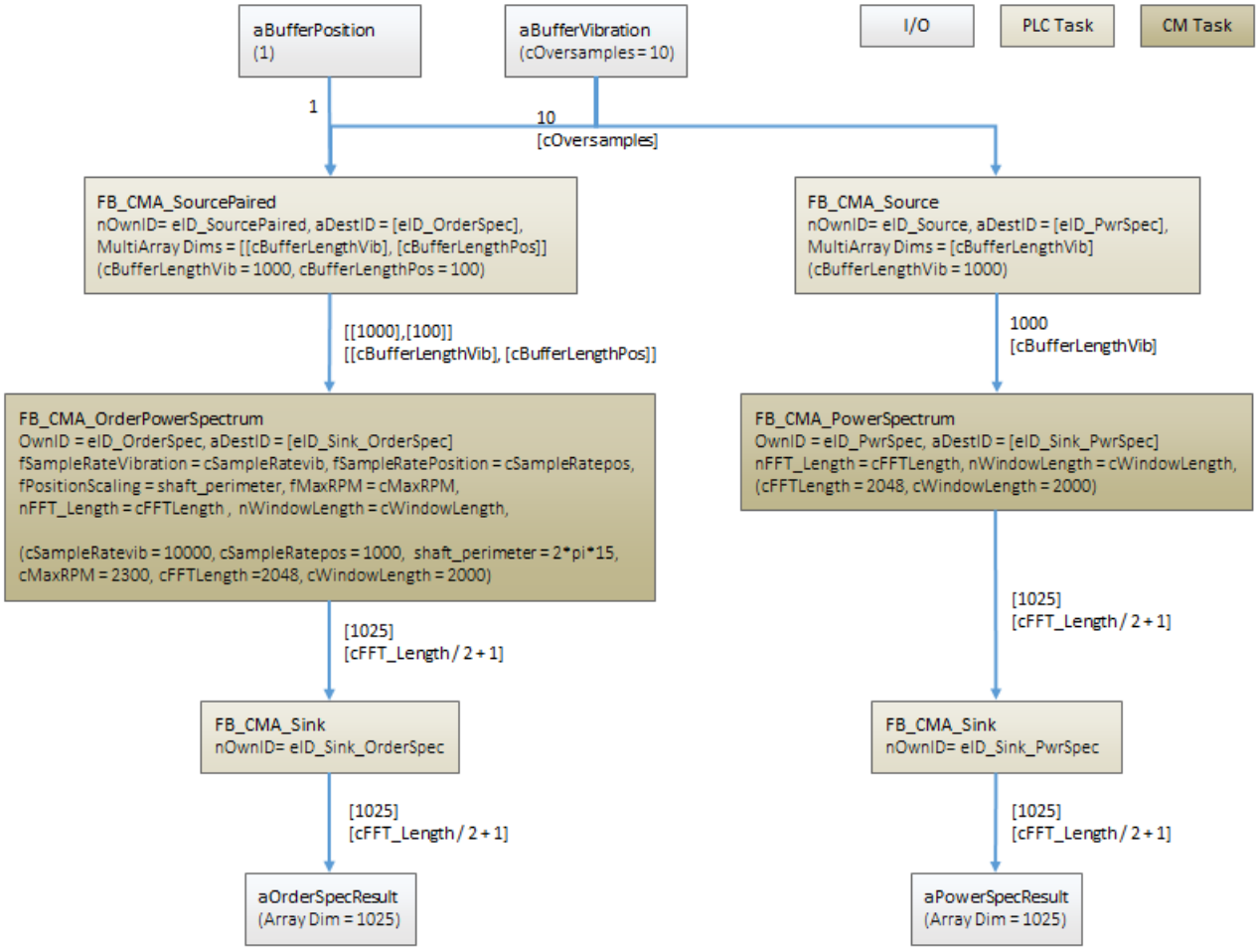
6.21 Ordnungsanalyse

In dieser Beispielimplementierung des FB_CMA_OrderPowerSpectrum [▶ 194] wird eine simulierte NC Achse sowie ein synthetisches Vibrationssignal genutzt, um die Parametrierung des OrderPowerSpectrum exemplarisch zu verdeutlichen sowie den Unterschied zum gewöhnlichen PowerSpectrum aufzuzeigen.

Die TwinCAT Solution nutzt eine 500 µs Zykluszeit für die NC. Es ist daher vorzugsweise als Zielsystem ein Beckhoff IPC oder embedded PC zu nutzen. Schalten Sie vorzugsweise, vor allem auf PCs dritter, Autostart Boot Project aus.

Der Code kann hier heruntergeladen werden: https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/9846479115.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit den wichtigen Parametern für die Konfiguration des Ordnungsspektrums, vgl. GVL_constants.

Das künstliche Vibrationssignal wird erzeugt in der PLC Task (1 ms Zykluszeit) mit einem Oversampling-Faktor von 10. Dies entspricht einer Abtastrate von 10 kHz, vgl. `cSampleRateVib`. Das Positionssignal wird aus der PLC Task heraus mit 1 ms von der NC über das Axis-Interface abgefragt. Dies entspricht einer Abtastrate von 1 kHz, vgl. `cSampleRatePos`. Als maximale Drehgeschwindigkeit wird `cMaxRPM` angesetzt.

Damit ergibt sich eine maximal auflösbare Ordnung 130,4.

Die Auflösung der Ordnungsachse wird über die FFT-Länge bestimmt, hier 2048, wodurch sich eine Auflösung von 0.1274 ergibt. Dieser Wert ist im Beispielprogramm im TwinCAT Scope auf dem Symbol aOrderSpecResult unter Scalefactor (i) zu finden, um die Darstellung des Arrays von Bins auf die Ordnung zu skalieren.

Die Fensterlänge des Vibrationssignal `cWindowLength` beschreibt die Signallänge, welche zur Transformation in den Frequenzbereich genutzt wird. Sie ist zu 2000 Samples gewählt worden. Die FFT-Länge größer als die Fensterlänge und als power-of-2 zu wählen. Für die Transformation ist ein Hanning-Fenster mit standard overlap konfiguriert, sodass der Eingangspuffer als `cWindowLength/2`, also 1000, definiert ist. Der Positions-Puffer beinhaltet, in der Zeit in der der Vibrations-Puffer 1000 Elemente aufweist, 100 Elemente.

`cSampleRateVib`	10000
`cSampleRatePos`	1000
`cShaftPerimeter`	2 * PI * 15
`cFFTLength`	2048
`cWindowLength`	2000
`cBufferLengthVib`	1000
`cBufferLengthPos`	100
`cMaxRPM`	2300

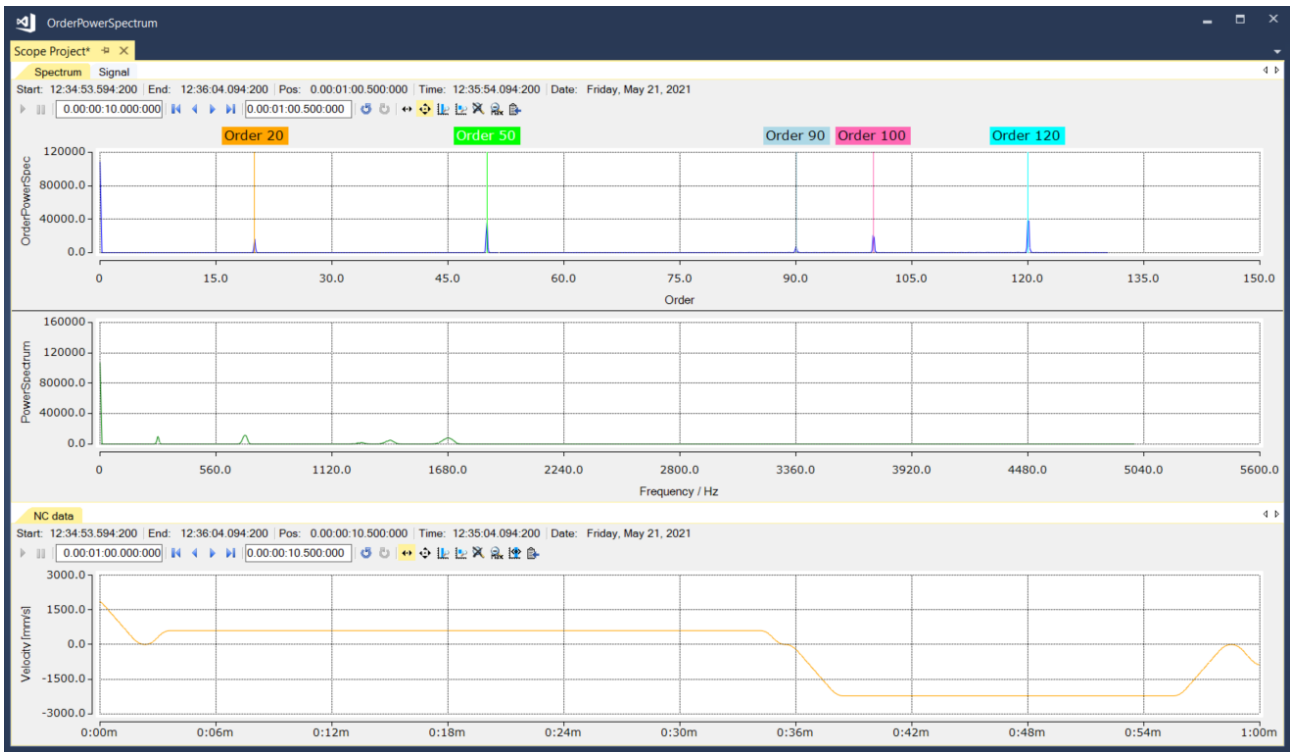
Konfiguration

Der Source Baustein wird nur dann aufgerufen, wenn die Geschwindigkeit der Welle zwischen einem minimalem und einem maximalen Wert liegt. Darunter und darüber wird die Analysekette zurückgesetzt und darauf gewartet, bis die Wellengeschwindigkeit wieder in einen gültigen Bereich rückt.

```
IF ABS(motorspeed) >= cMinRPM / 60 AND ABS(motorspeed) <= cMaxRPM / 60 THEN
```

Es wird ein synthetisches Signal erzeugt, welches an definierte Ordnungen eine definierte Amplitude aufweist. Die Ordnungen aus dem Array werden im Scope mit Markern zur Orientierung angezeigt.

Die virtuelle Achse verändert automatisch ihr Drehgeschwindigkeit, vgl. `nmovestate` in der MAIN (PRG). Im TwinCAT Scope ist das Ergebnis der OrderPowerSpectrum sowie des gewöhnlichen PowerSpectrum übereinander aufgezeigt. Darunter ist zur Orientierung der zeitliche Verlauf der Drehgeschwindigkeit aufgetragen. Wie in untenstehender Grafik gezeigt, sind die synthetisch erzeugten Amplituden genau an den vorgesehenen Ordnungen (vgl. MAIN.Orders) im Ordnungsspektrum als scharfe Peaks zu sehen. Im PowerSpectrum sind die Amplituden je nach Drehzahl der Welle an einer anderen Frequenz und im Bereich sich verändernder Drehzahl auch nur als unscharfe Peaks zu erkennen.



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base, Tc3_MultiArray

6.22 Schädigungsüberwachung

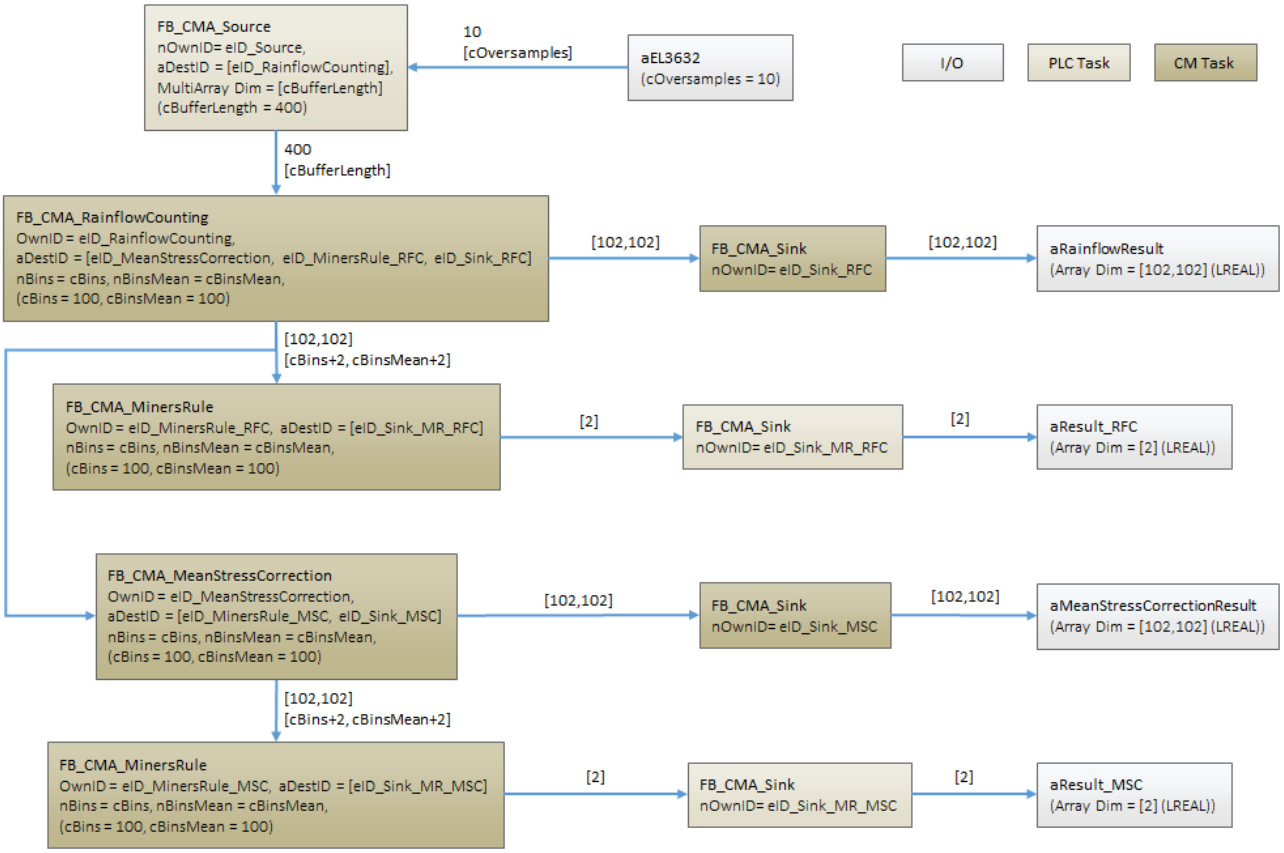
Dieses Beispiel erläutert die Funktionsweise und Anwendungsmöglichkeit der FunktionsbausteineFB_CMA_RainflowCounting [▶ 211], FB_CMA_MeanStressCorrection [▶ 176] und FB_CMA_MinersRule [▶ 179] zur Abschätzung des Ermüdungsprozesses von überwachten Bauteilen.

Für die Berechnung der Wöhlerkurve wird die Funktion F_CM_CalculateWoehlerCurve [▶ 288] auf Basis von fiktiven Materialparametern durchgeführt.

Nähere Erläuterungen sind unter dem Punkt Anwendungskonzepte [▶ 34] im Abschnitt Lebensdaueranalyse und Schädigungsrechnung [▶ 53] zu finden.

Den Quellcode für dieses Beispiel können Sie hier herunterladen: https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/9785658251.zip

Blockdiagramm



Programmparameter

Die Tabelle unten zeigt eine Liste mit den wichtigen Parametern (GVL_Constants) für die Konfiguration der Funktionsbausteine.

Zunächst ist der voraussichtlich zu erwartende Messbereich für mechanische Spannungen, definiert über `cStressMin` und `cStressMax`. Es ist wichtig, dass alle während der Überwachung auftretenden Spannungen in diesem Intervall liegen. Die Auflösung mit der die Schwingbreiten der Spannungszyklen dann gezählt werden wird durch `cRfRangeBins` und `cRfMeanBins` definiert. Standardmäßig wäre hier minimal 64 x 64 anzusetzen, gewöhnlich wäre 128 x 128. Weitere Konstanten zur Parametrierung der Funktionsbausteine definieren sich dann aus diesen vier Werten.

Im Beispielcode ist der Einfluss der Mittelspannungskorrektur berücksichtigt und kann vergleichend gegen die Berechnung ohne Mittelspannungskorrektur im TwinCAT Scope beobachtet werden. Das Beispiel nutzt die Korrektur nach Goodman, kann aber über `cMSCType` umgestellt auf die Korrektur nach Gerber oder auch deaktiviert werden.

Des Weiteren werden Materialparameter für die Definition der verwendeten materialspezifischen Wöhler Kurve angegeben. Die hier verwendet Paramater sind fiktiv und so gewählt, dass nach kurzer Zeit ein visueller Effekt im TwinCAT Scope zu beobachten ist.

`cStressMin`	-50 MPa	Minimal auftretende Spannung
`cStressMax`	50 MPa	Maximal auftretende Spannung
`cRfRangeBins`	100	Anzahl der Bins in der Stress-Range-Achse
`cRfMeanBins`	100	Anzahl der Bins in der Mittelspannungs-Achse
`cMSCType`	eCM_Goodman	Mittelspannungskorrektur-Typ
`cSRI`	350	Stress Range Intercept (Spannungsschnittpunkt bei Zyklenzahl 1)
`cUTS`	700	Ultimate Tensile Strength (dt. Zugfestigkeit)
`cK1`	3	Steigung der Wöhlerkurve zwischen N = 1 und cNC1
`cK2`	5	Steigung der Wöhlerkurve beginnend ab Punkt cNC2

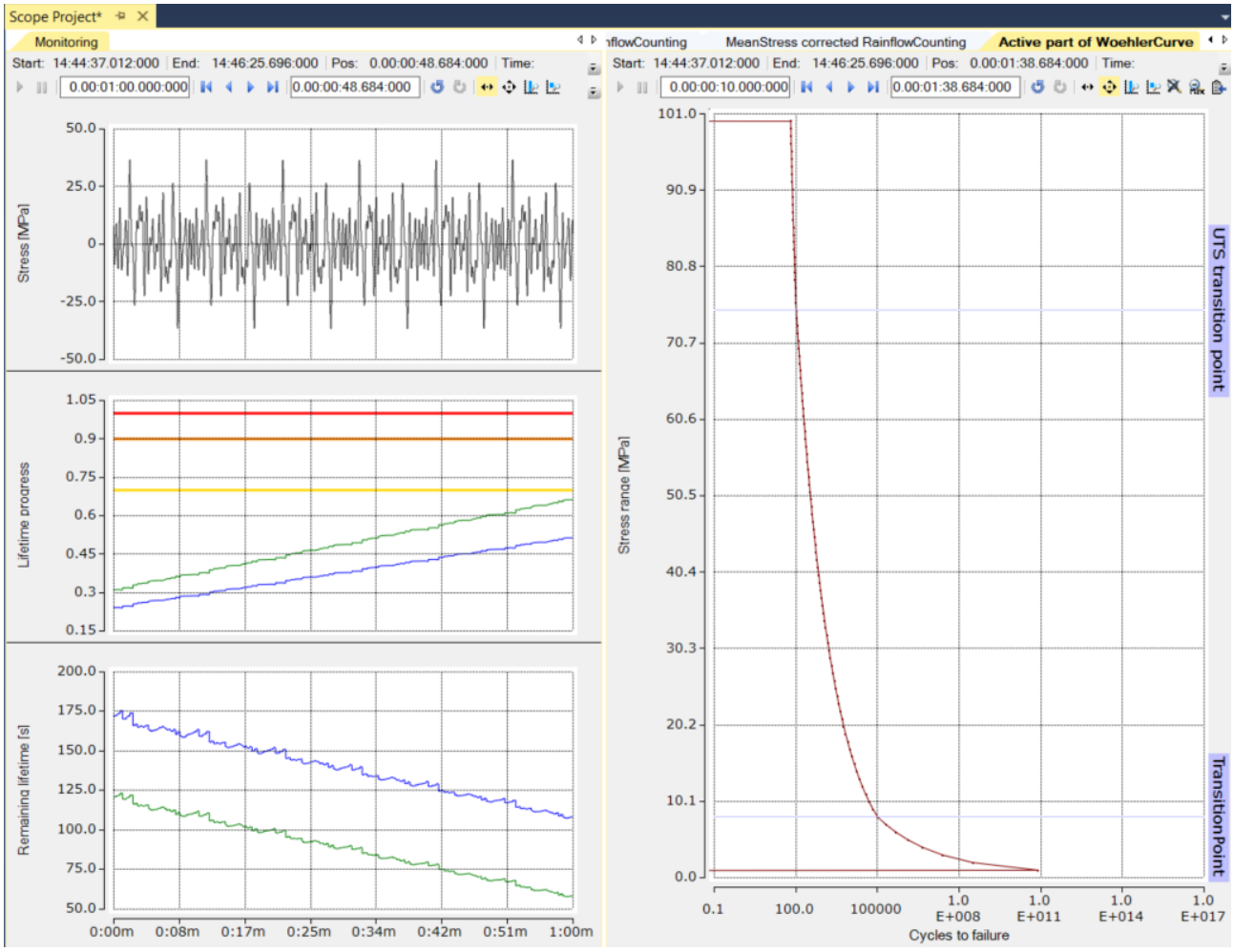
cNC1	100	Transitionspunkt für UTS Korrektur
cNC2	100000	Fatigue Transition Point

Erläuterungen

Nach Aktivierung der TwinCAT Konfiguration und Start des TwinCAT Scope sind mehrere Darstellungen zu sehen.

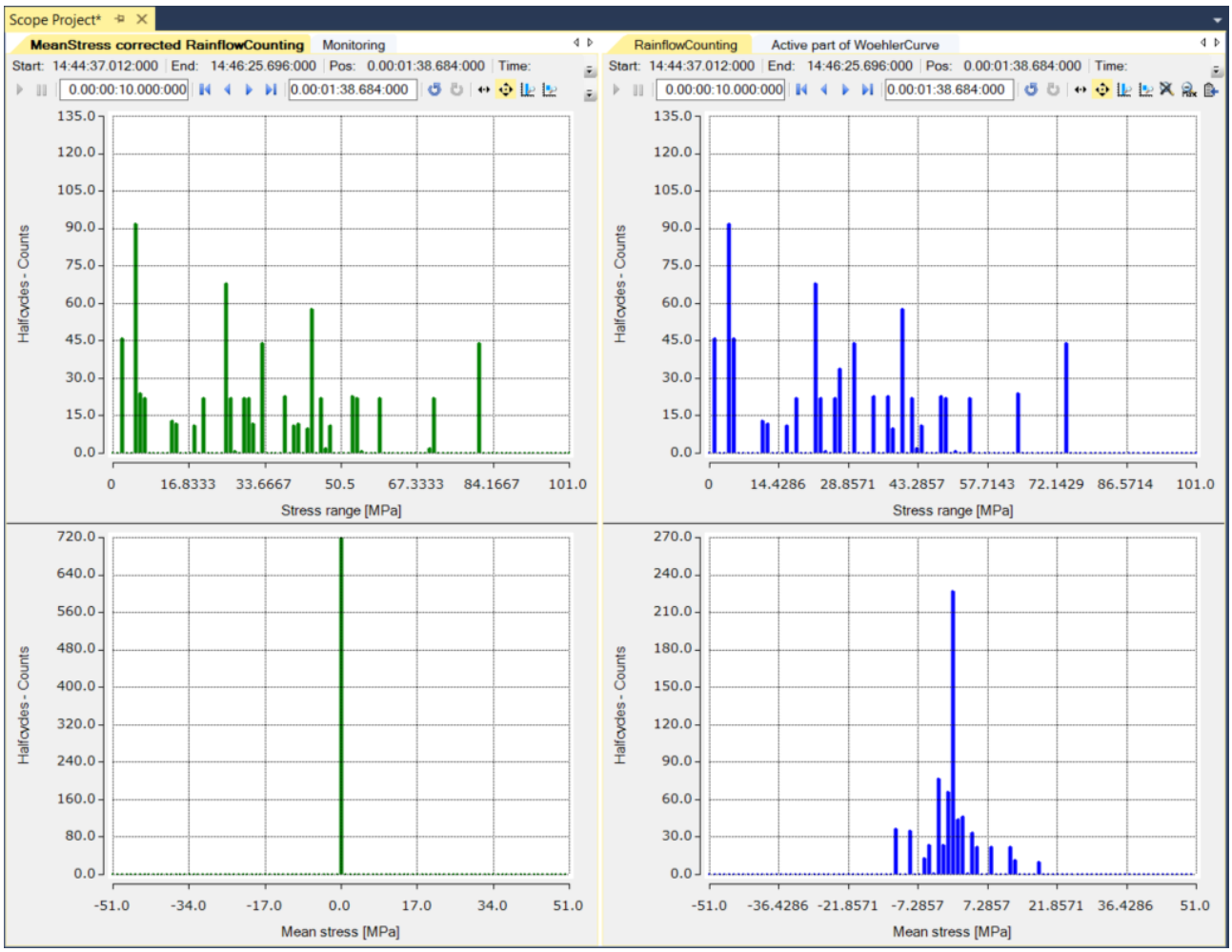
Unten ist auf der rechten Seite eine statische Grafik zu sehen, welche die Wöhler Kurve darstellt. Diese Darstellung erfolgt aus der Parametrierung in der GVL_Constant.

Auf der linken Seite in Schwarz ist das synthetisch erzeugte Spannungssignal über die Zeit aufgetragen. Die Spannung wird in diesem Beispiel immer in Megapascal (MPa) angegeben. In der Grafik darunter wird der Fortschritt der Summenschädigung angezeigt, wobei Grün jeweils für die Schadenssumme mit aktivierter Mittelspannungskorrektur steht und Blau für die Schadenssumme mit deaktivier Mittelspannungskorrektur. Der Fortschritt der Schädigung wächst von Null an, wobei Eins die maximale Summenschädigung (Punkt des rechnerischen Bauteilversagens) unter Verwendung der bereitgestellten Materialparameter darstellt. Entsprechend sind 3 horizontale Linien bei 70%, 90% und 100% eingezeichnet, welche als Warn- und Alarmschwellen genutzt werden könnten. In der untersten Grafik ist die geschätzte Restlebensdauer in Sekunden angegeben. Entsprechend fallen diese beiden Kurven mit Fortlaufen der Simulation.



Ebenso sind die bereits gezählten Halbzyklen des Rainflow Counting in Array Bar Charts dargestellt, wiederum in Grün mit Mittelspannungskorrektur und in Blau ohne Mittelspannungskorrektur. Die höhere Schädigung durch die Korrektur der Schwingbreiten bei aktivierter Mittelspannungskorrektur ist deutlich zu beobachten.

Im linken Teil der Grafik ist zu sehen, dass die Mittelspannungskorrektur die Mean-Stress-Achse obsolet macht (auf Null setzt) und als Korrektur auf die Stress Range aufschlägt. Aus diesem Grund sind Halbzyklen-Zählungen auf der linken Seite etwas nach rechts auf der Stress-Range-Achse versetzt.



Voraussetzungen

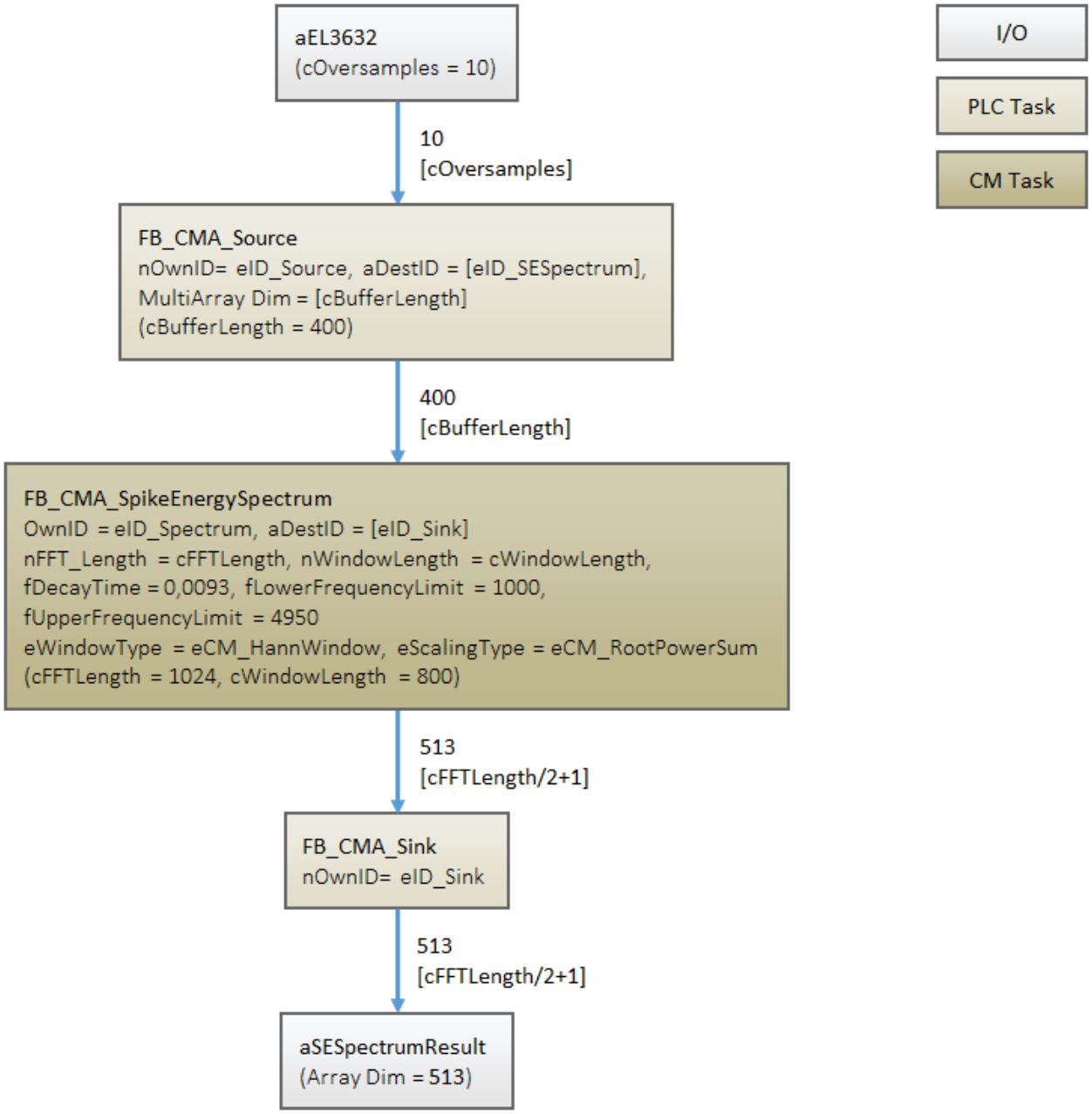
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

6.23 Spitzenwert Spektrum

Dieses Beispiel implementiert eine Möglichkeit zu Analyse von Lagerschäden durch die Betrachtung von hochfrequenten Anteilen in den Vibrationsdaten. Aufgrund der Sensitivität der Spitzenenergie gegenüber einem realen Aufbau wurde die Energie der simulierten Stöße hinreichend groß gewählt. Im Einsatz sollten hierbei Trends im Spitzenwert Spektrum im Laufe der Zeit betrachtet werden.

Den Quellcode für dieses Beispiel können Sie hier herunterladen: https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/9783190923.zip

Blockdiagramm



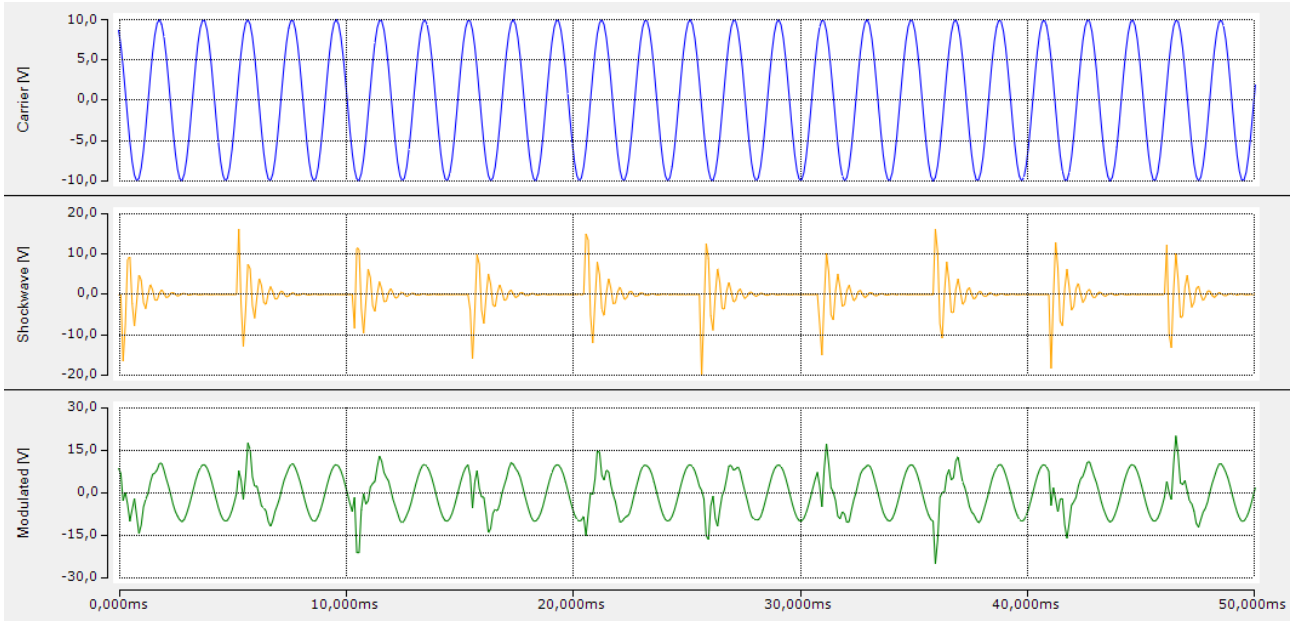
Programmparameter

Die Tabelle unten zeigt eine Liste mit den wichtigen Parametern für die Konfiguration der Funktionsbausteine.

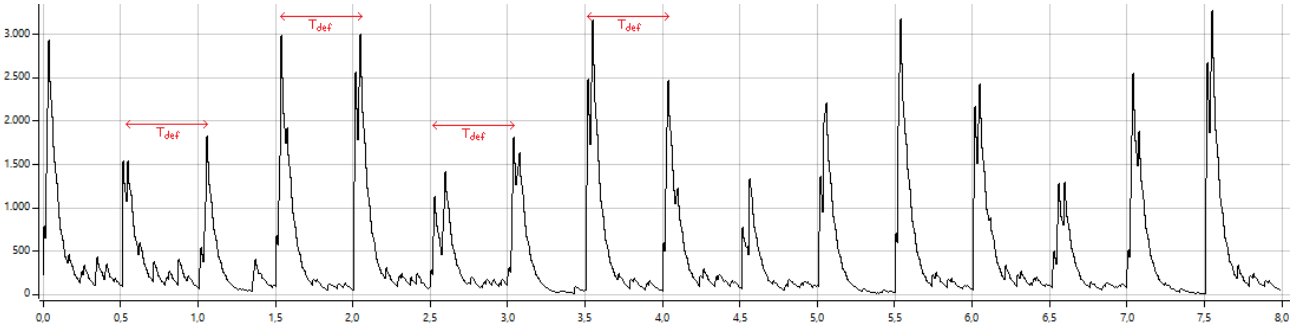
FFT-Länge	1024
Fenstergröße	800
Puffergröße	400
Länge des Spitzenwert Spektrums	513
Untere Grenze des Bandpassfilters [Hz]	1000
Grenzfrequenz des Bandpassfilters [Hz]	4950

Erläuterungen

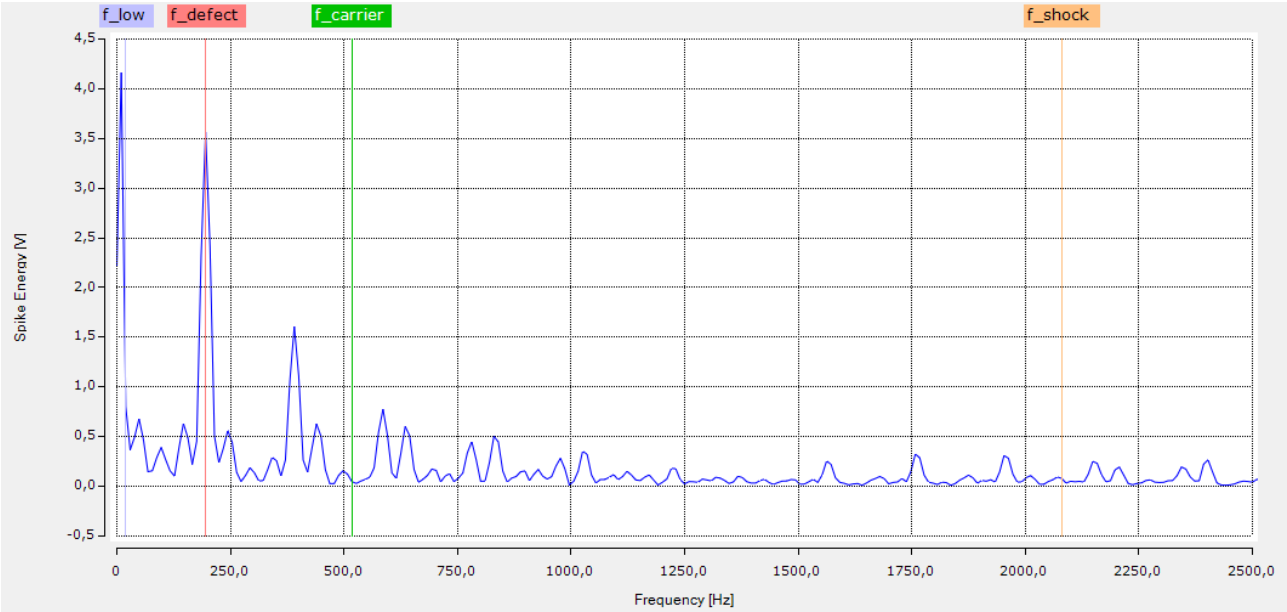
Das generierte amplitudenmodulierte Eingangssignal setzt sich wie folgt zusammen: Auf eine Grundschwingung (blau) wird mit einer Frequenz von 200 Hz ein Schock (gelb) moduliert. Das Eingangssignal des Bausteins entspricht somit der Kurve unten (grün).



Die Eingangsdaten werden wie folgt verarbeitet: Die für die Analyse nicht relevanten Signalanteile (unterhalb von 1 kHz) werden mittels eines IIR Bandpassfilters herausgefiltert. Dies entspräche hier im Fall exakter Arithmetik dem Schock (gelb). Das gefilterte Signal wird nun in eine Zeit-Wellen-Form transformiert, in welcher die Energiespitzen in Abhängigkeit der konfigurierten Abklingzeit (`fDecayTime`) dargestellt sind („peak to peak“). Das nachfolgende Bild zeigt die Zeit-Wellen-Form für die oben beschriebenen Eingangsdaten.



Bereits in dieser Darstellung ist die Frequenz der auftretenden Spitzenwerte, d.h. die Wiederholrate hochfrequenter Frequenzanteile in Form von Stößen im Ursprungssignal, erkennbar. Der Abstand zwischen zwei aufeinander folgenden Spitzen liegt bei 0,5 ms, d.h. 1 / 0,005 s = 200 Hz. Diese Periodizität wird durch die Berechnung eines Magnitudenspektrums dargestellt. Dies führt zu nachfolgender Visualisierung im Scope Projekt des Beispiels:



Im resultierenden Spitzenwert Spektrum ist der Anteil der Fehlerfrequenz (`f_defect`) bei 200 Hz sowie die zugehörigen Harmonischen deutlich erkennbar. Ferner sind die Frequenzanteile der Grundschwingung (`f_carrier`) sowie jene der Schocks (`f_shock`) durch die Bandpassfilterung und Transformation in die Zeit-Wellen form eliminiert.

Der Spitzenwert unterhalb von `f_low` ist hierbei nicht aussagekräftig, da er unterhalb der Auflösungsgrenze von `cMinFrequency = cSampleRate/cWindowLength = 12,5` Hz liegt.

Voraussetzungen

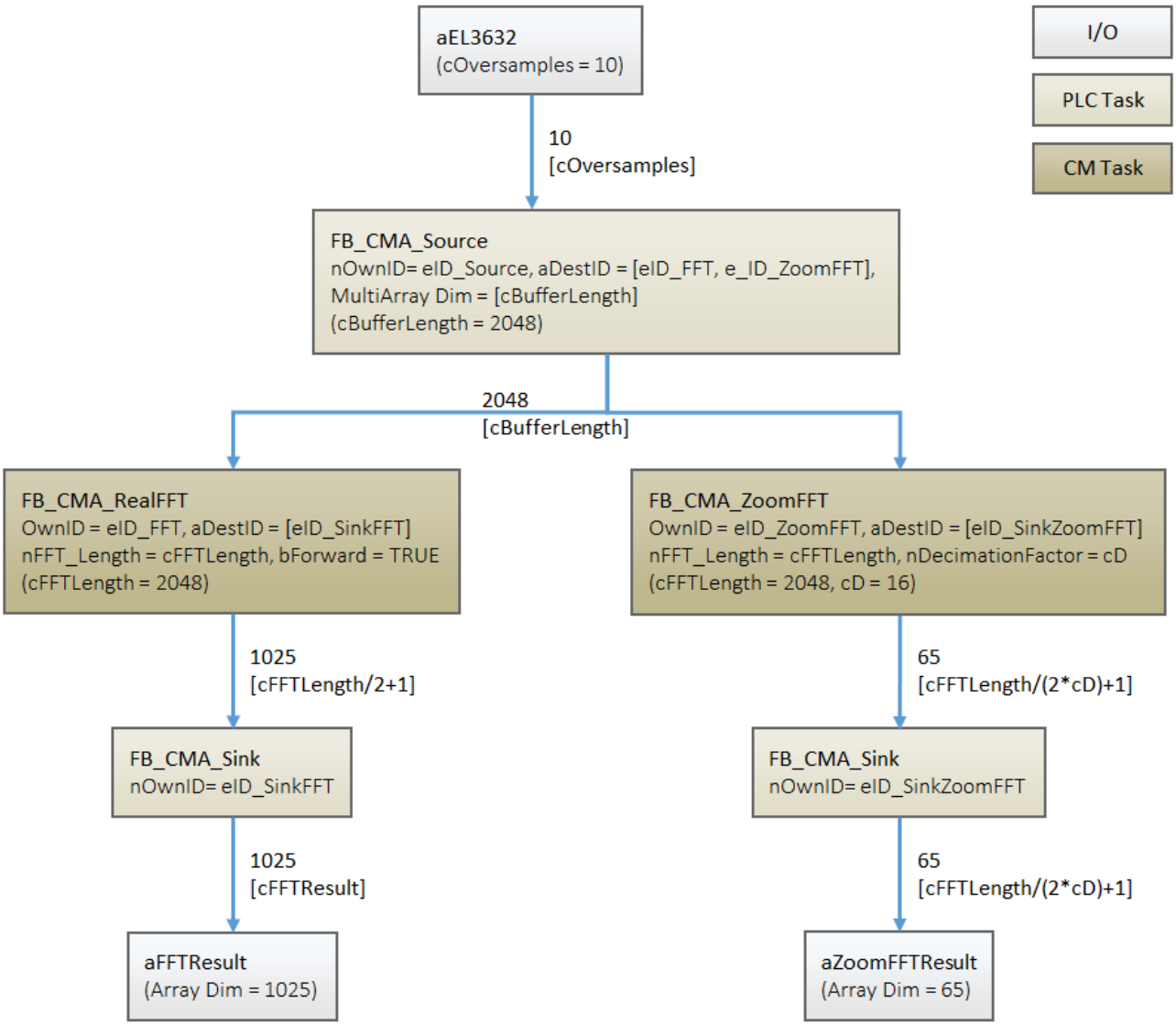
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

6.24 Zoom FFT

Dieses Beispiel zeigt exemplarisch die Verwendungsmöglichkeiten des Bausteins FB_CMA_ZoomFFT. Dabei wird mittels des genannten Funktionsbausteins der Ausschnitt eines mit dem Baustein FB_CMA_RealFFT ebenfalls berechneten Spektrums berechnet.

Den Quellcode für dieses Beispiel können Sie hier herunterladen: https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/9783192587.zip

Blockdiagramm



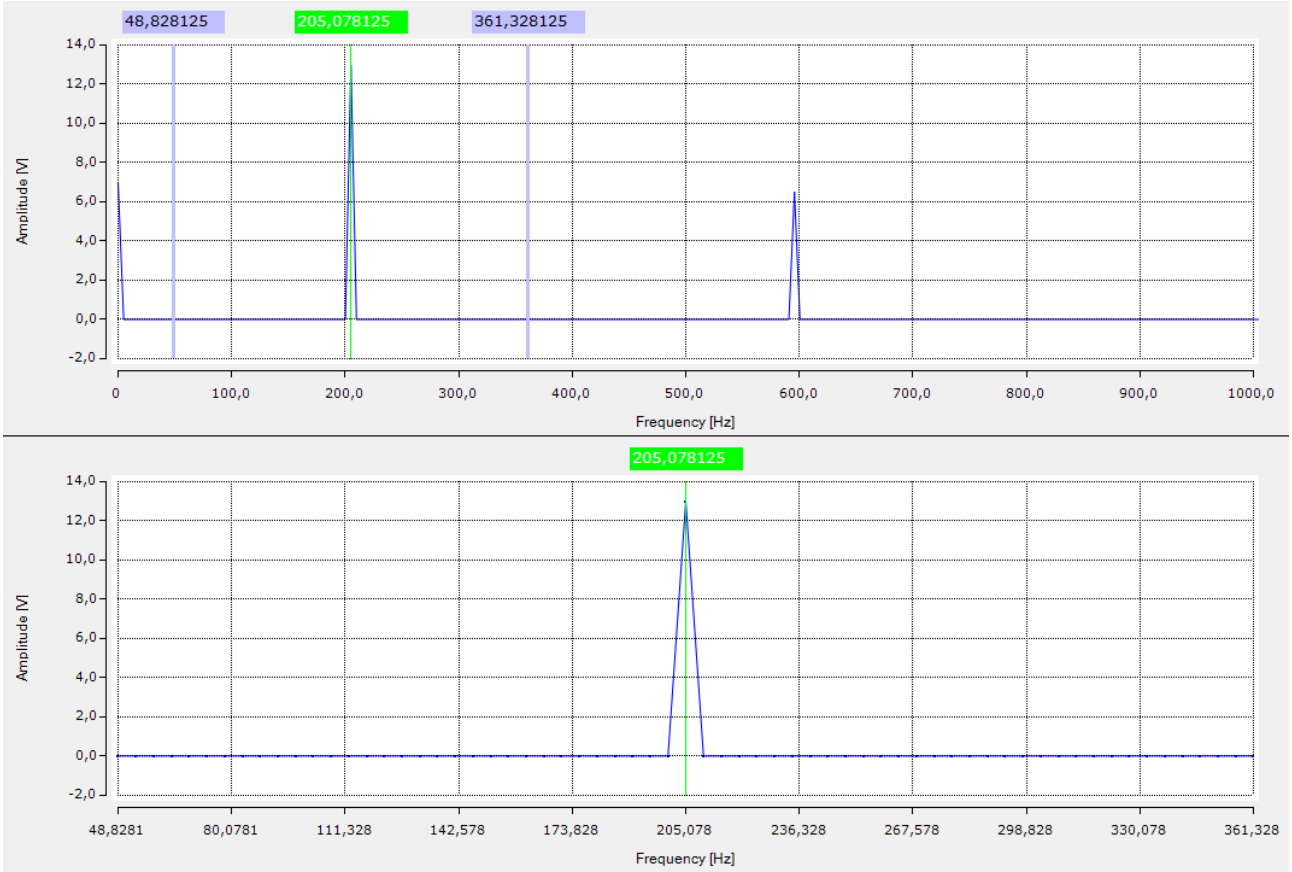
Programmparameter

Die Tabelle unten zeigt eine Liste mit den wichtigen Parametern für die Konfiguration der Funktionsbausteine.

FFT-Länge	2048 / 2048
Puffergröße	2048 / 2048
Decimierungsfaktor	16 / -
Länge des Berechneten Spektrums	65 / 1025

Erläuterungen

Das in dem Beispielcode enthaltene Scope Projekt visualisiert ein mittels des Bausteins FB_CMA_RealFFT berechnetes Spektrum im Bereich von 0 bis 1kHz sowie einen Ausschnitt im Bereich 48,83 Hz und 361,33 Hz welcher mit dem Baustein FB_CMA_ZoomFFT berechnet wurde. Hierbei ist die Auflösung im Frequenzbereich identisch (4,8828125 Hz), wobei für das gesamte Spektrum eine FFT der Länge 2048 und in dem Ausschnitt über die Zoom FFT eine FFT der Länge 128 auf dem dezimierten Zeitsignal gerechnet wird.



Voraussetzungen

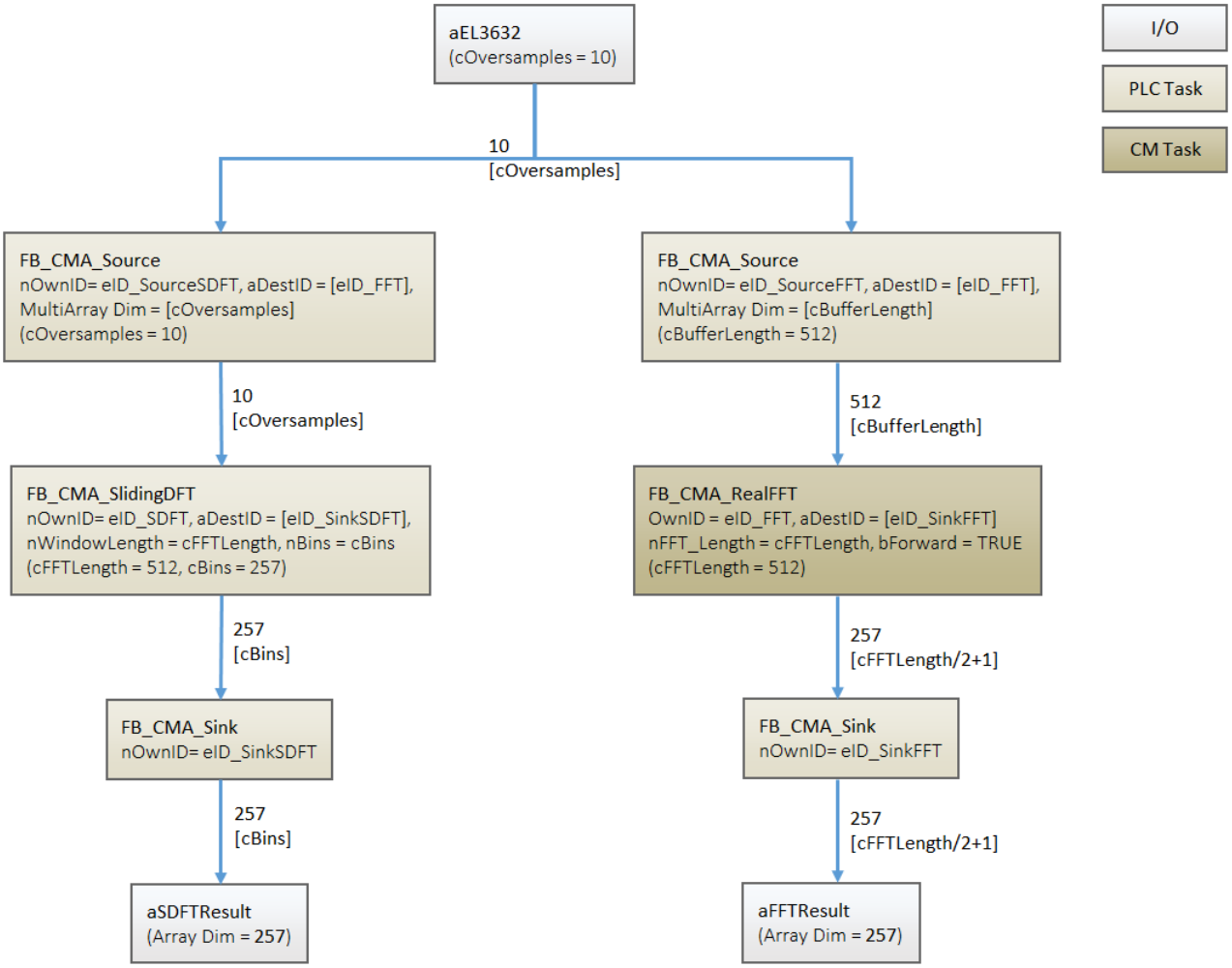
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

6.25 Sliding DFT

Dieses Beispiel implementiert exemplarisch die Verwendungsmöglichkeiten des Funktionsbausteins FB_CMA_SlidingDFT. Hierin wird der Baustein so konfiguriert, dass alle Spektralwerte berechnet und die Ergebnisse werden denen aus der Berechnung mittels des Bausteins FB_CMA_RealFFT gegenübergestellt. Ferner wird zur Verbesserung des Berechneten Spektrums ein Hanning-Fenster im Spektralbereich angewendet.

Den Quellcode für dieses Beispiel können Sie hier herunterladen: https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/9783189259.zip

Blockdiagramm



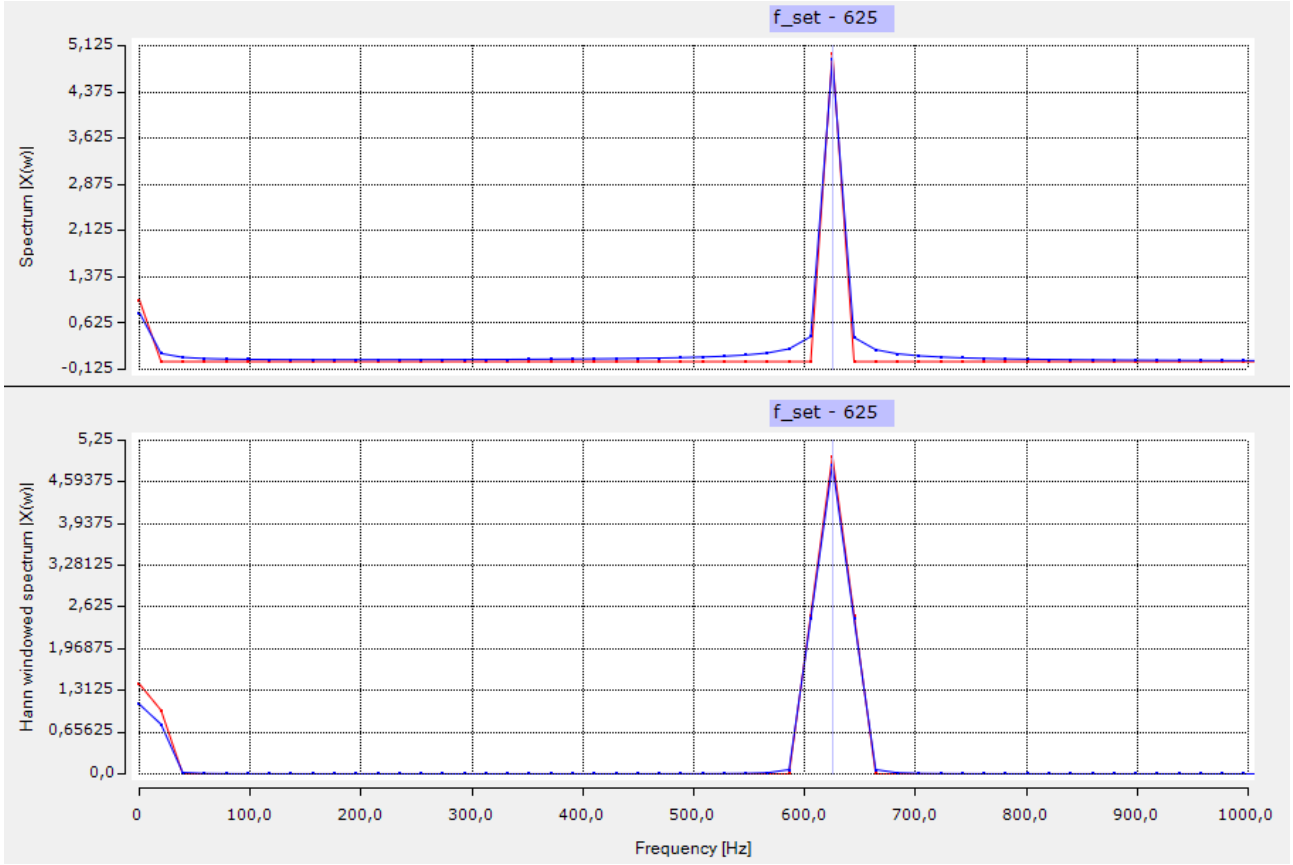
Programmparameter

Die Tabelle unten zeigt eine Liste mit den wichtigen Parametern für die Konfiguration der Funktionsbausteine.

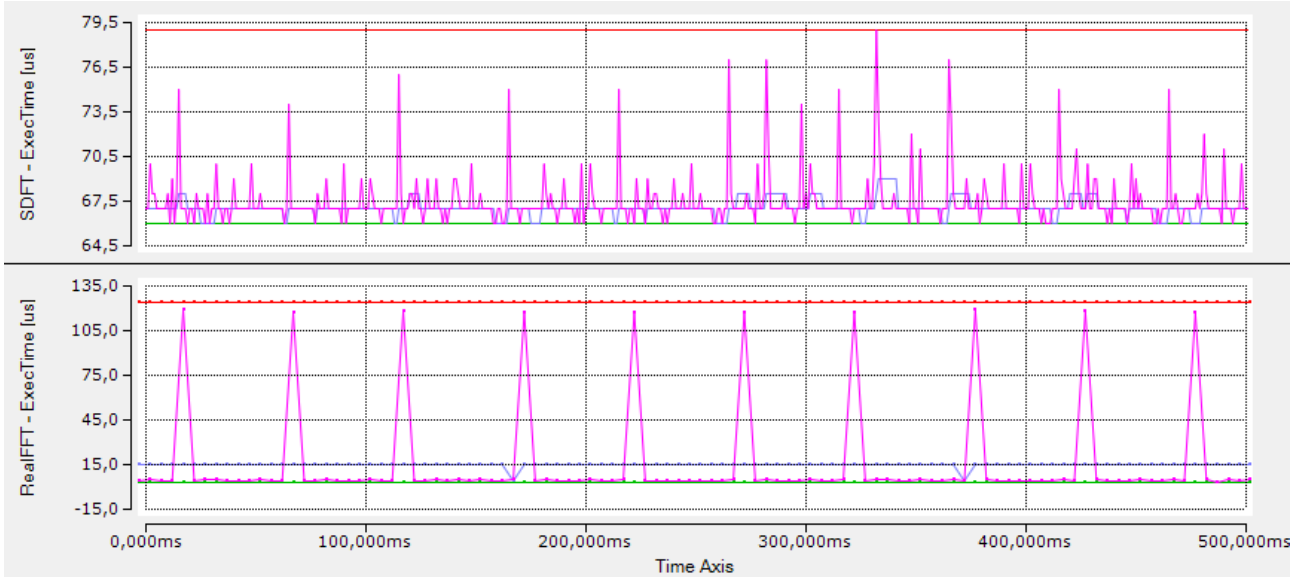
FFT-Länge	512 / 512
Puffergröße	10 / 512
FFT Ergebnislänge	257 / 257

Erläuterungen

Das im Sample enthaltene Scope Projekt visualisiert die berechneten Spektren über ihren Betrag im Bereich von 0 bis 1 kHz. Hierbei fällt zum einen der Einfluss der Sidelobes im Spektrum des Bausteins FB_CMA_SlidingDFT (blaue Linie) auf. Die Abweichungen sind wesentlich von dem gewählten Dämpfungsparameter `fDampingFactor` abhängig. Die Abweichungen werden mit Werten oberhalb von `0.995` kleiner, jedoch wird die Berechnung bei einer zu geringen Dämpfung (`>0.999)` instabil. Auf Grund der Definition der Rekursionsvorschrift für die Spektrallinie `k = 0`, ist die Berechnung für den DC Anteil hingegen genauer. Eine Verbesserung der der Berechneten Werte kann durch die Anwendung des Hanning Fensters im Spektralbereich erreicht werden (Bild unten).



Ein wesentlicher Unterschied zwischen den beiden Bausteinen ist die Auslastung in der jeweiligen Task. Dies wird in der folgenden Grafik verdeutlicht.



Im Gegensatz zu den Auslastungspeaks der RealFFT (Bild unten) ist die Auslastung der SDFT (Bild oben) konstanter. Letzterer Wert liegt im Mittel zwar deutlich höher, unterliegt aber geringeren Schwankungen.

Voraussetzungen

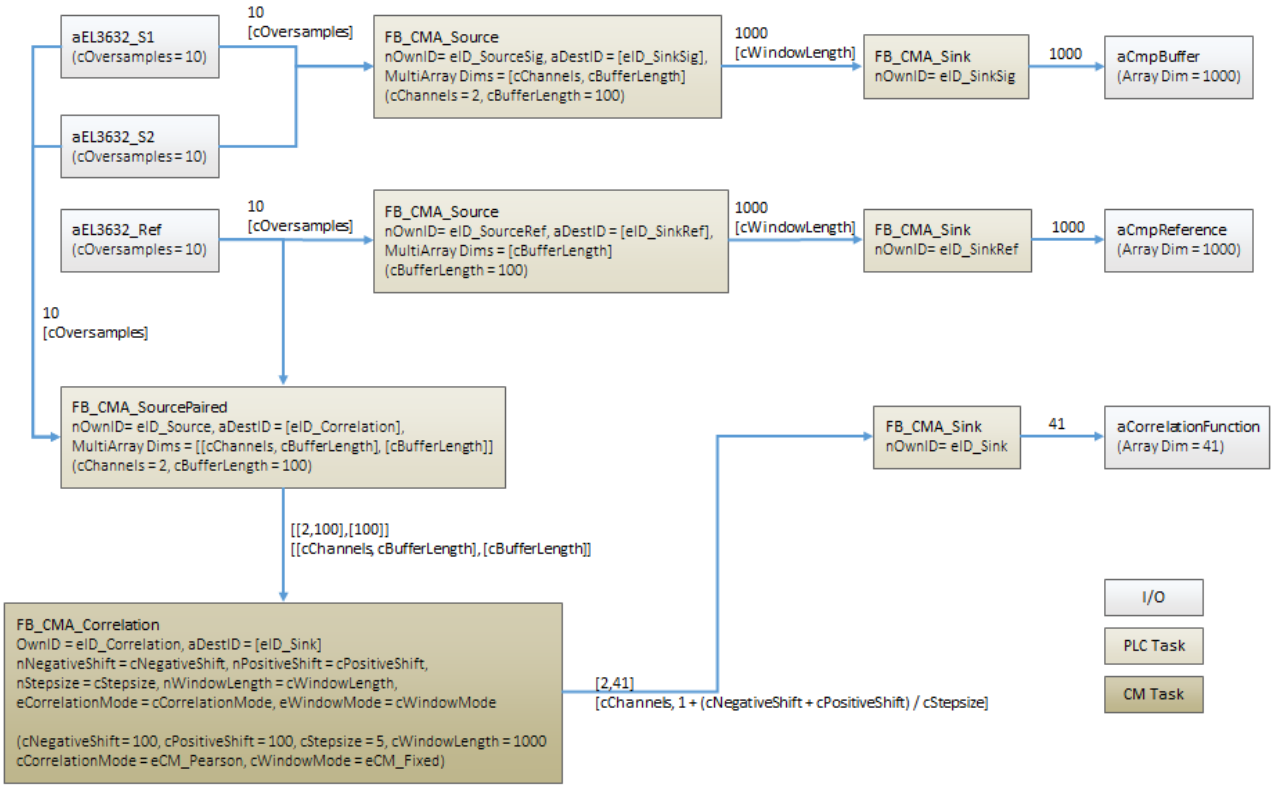
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

6.26 Korrelationsfunktion

Dieses Beispiel implementiert exemplarisch die Verwendungsmöglichkeiten des Funktionsbausteins FB_CMA_Correlation. Hierbei werden die Korrelationskoeffizienten für einen zeitlichen Versatz von +/- 100 Samples bezüglich eines Referenzsignals berechnet.

Den Quellcode für dieses Beispiel können Sie hier herunterladen: https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/9833229195.zip

Blockdiagramm



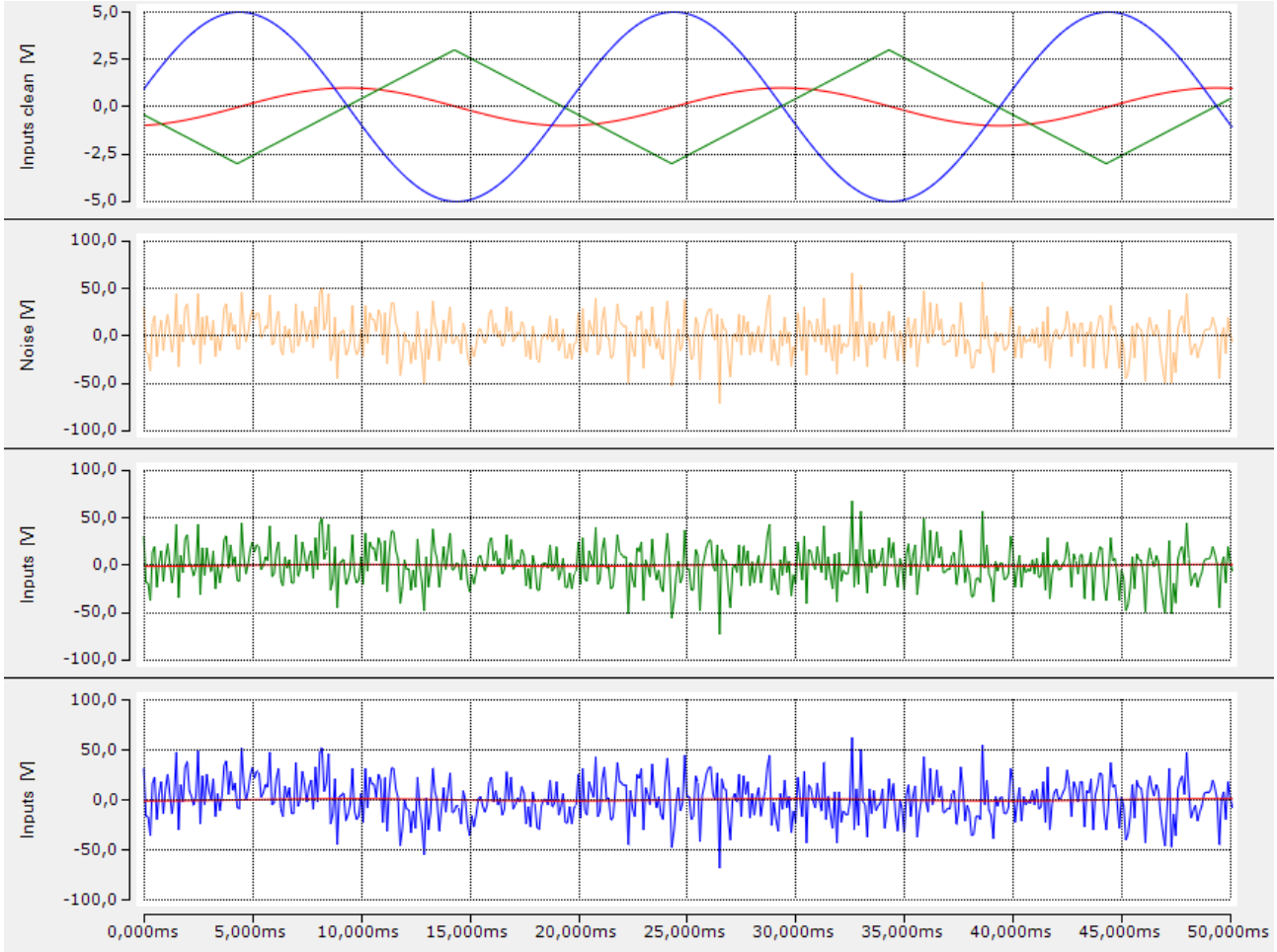
Programmparameter

Die Tabelle unten zeigt eine Liste mit den wichtigen Parametern für die Konfiguration der Funktionsbausteine.

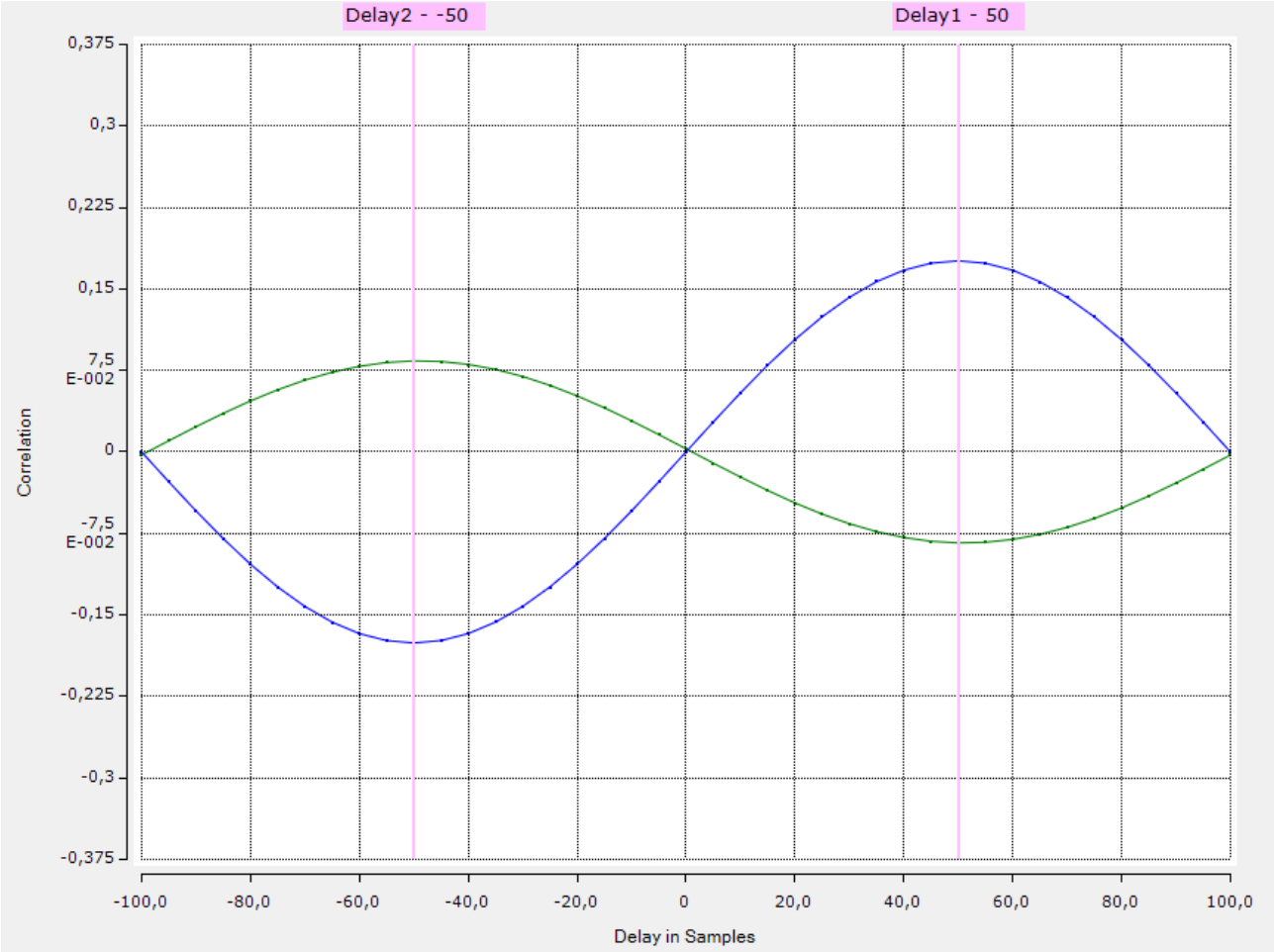
nWindowLength	1000
nNegativeShift	100
nPositiveShift	100
nStepsize	5
eCorrelationMode	eCM_Pearson
eWindowMode	eCM_Fixed

Erläuterungen

In diesem Beispiel werden die Korrelationsparameter zwischen einem Sinus mit Amplitude 1 (rot) sowie einem verstärkten um +5ms (50 Samples bei 10 kHz Abtastrate) Sinus (blau) und einem Dreiecksimpuls (grün) mit identischer Frequenz (50 Hz) und einer Verzögerung von -5ms berechnet. Die Testsignale sind mit einem Normalverteiten Rauschen überlagert. Die nachfolgende Grafik zeigt die Ausgangssignale (oben) und die verrauschten Signale (unten)



Das Rauschen ist so stark gewählt, dass die Ausgangs im Zeitbereich nicht mehr identifiziert werden können. Die Berechnung der Korrelationskoeffizienten ermöglich jedoch die Einordnung in Bezug auf das Referenzsignal:



Die höchste Korrelation des verstärkten Sinus (blau) liegt bei einer Verschiebung von 50 Samples zur Referenz, d.h. das Signal ist um 5ms verzögert. Analog hierzu ist der Dreiecksimpuls (grün) um 50 Samples verschoben, d.h. es geht der Referenz 5ms voraus.

Voraussetzungen

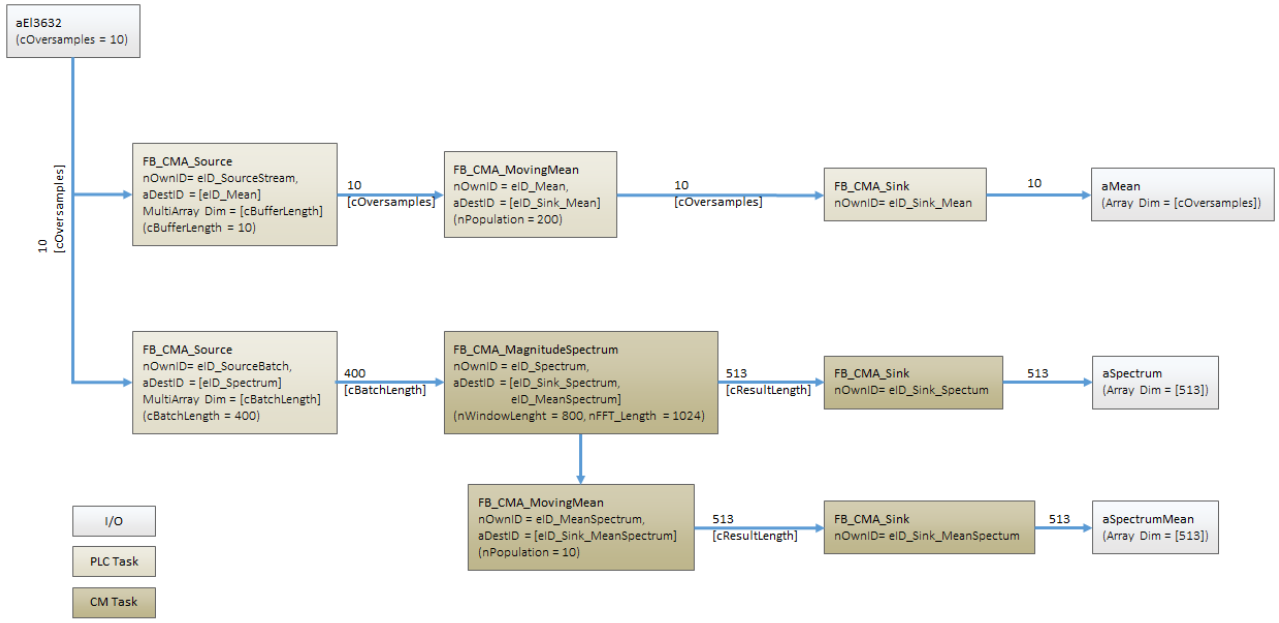
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

6.27 Gleitender Mittelwert

In diesem Beispiel wird die Verwendung der Condition Monitoring Bibliothek exemplarisch an der Berechnung eines gleitenden Mittelwertes präsentiert. Hierbei wird der Baustein FB_CMA_MovingMean [▶ 270] verwendet. In Analogie lassen sich ebenfalls eine gleitende Standardabweichung, Schiefe und Exzess mit Hilfe der Funktionsbausteine FB_CMA_MovingStandardDeviation [▶ 274], FB_CMA_MovingSkew [▶ 278] und FB_CMA_MovingExcess [▶ 283] berechnen.

Den Quellcode für das Beispiel können Sie hier herunterladen: https://infosys.beckhoff.com/content/1031/TF3600_TC3_Condition_Monitoring/Resources/16593242507.zip

Blockdiagramm



Programmparameter

Diese Tabelle enthält eine Liste mit den wichtigen Parametern für die Konfiguration der Funktionsbausteine.

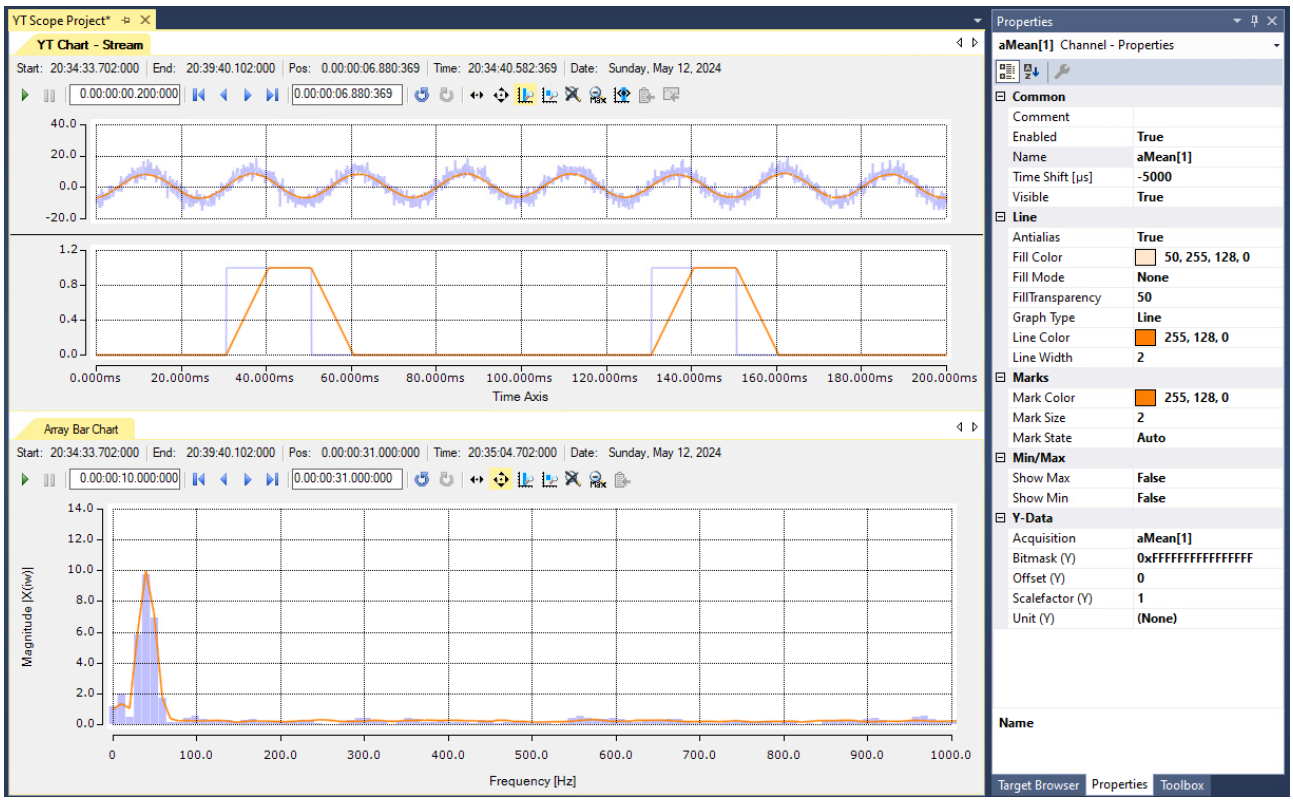
Oversamples / Pufferlänge	10
Population (Stream)	100
Stapellänge (Spektrum)	400
Fensterlänge	800
FFT Länge	1024
FFT Ergebnislänge	513

Erläuterungen

Das in diesem Sample enthaltene Scope-Projekt visualisiert in einem YT-Chart die berechneten gleitenden Mittelwerte eines Sinus sowie eines Impulses im Streaming-Kontext. Für eine korrekte Verarbeitung der Ausgangsdaten ist es wichtig, den zeitlichen Einfluss der Mittelung zu betrachten. Dieser wird im Folgenden erläutert.

- **Sinus**: Der berechnete Mittelwert wurde hier mit einem zeitlichen Versatz von -5000 us visualisiert. Dieser Wert ergibt sich aus der halben internen Pufferlänge (`nPopulation`) unter Beachtung der Zykluszeit von 1 ms (siehe *FastTask*) sowie dem Oversampling Faktors.
- **Impuls**: Die Mittelung der Impulsfunktion visualisiert die Trägheit des gleitenden Mittelwertes. Die Impulslänge ist hier auf 20 ms eingestellt. Respektive der Konfiguration (Zykluszeit, Oversampling) erreicht der berechnete gleitende Mittelwert nach jeweils 10 ms einen stabilen Wert.

Des Weiteren beinhaltet das Projekt die Darstellung eines berechneten gemittelten Spektrums in einem separaten Array Bar Chart. Dieses zeigt ein Magnitudenspektrum für den Bereich von 0 bis 1 kHz. Das jeweils aktuelle Spektrum wird als Balkendiagramm visualisiert, die gleitende Mittelung als durchgehende Linie.



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.25	PC or CX (x86, x64)	Tc3_CM, Tc3_CM_Base

7 Anhang

7.1 Fehlercodes Übersicht

Fehler Codes werden vom Typ HRESULT zurückgegeben. Ein Test auf Nonzero Values ist bei Werten vom Typ HRESULT nicht ausreichend.

declaration	error	ok	ok but with info	check functions
hrErrorCode :HRESULT;	< 0	>= 0	> 0	SUCCEEDED(), FAILED()

Die folgenden Fehlercodes können auftreten.

16#9811_0000 - 16#9811_FFFF	gelistet in TwinCAT (ADS) Error Codes [▶ 387] (dort ohne höherwertiges WORD). Weitere Hinweise unten auf der Seite.
16#9851_0000 - 16#9851_FFFF	Condition Monitoring Error Codes sind aufgeführt unter E_CM_ErrorCode [▶ 292]
16#9852_0000 - 16#9852_0FFF	Condition Monitoring Analysis Error Codes stehen in E_CMA_ErrorCodes [▶ 295]
16#9871_0000 - 16#9871_FFFF	MultiArray Error Codes findet man in E_MA_ErrorCode [▶ 296]

Wenn ein Fehler während der Initialisierung auftritt, kann der Funktionsbaustein nicht genutzt werden.

Weitere Hinweise für Standard TwinCAT Error Codes:

error value	symbol	Fehlerbeschreibung	Behebungsmöglichkeit
16#9811_070A	NOMEMORY	Keinen Speicher	Falsche Speicher-Einstellungen => Router Speicher erhöhen (siehe Kapitel Speicherverwaltung [▶ 73])
16#9811_0719	TIMEOUT	Gerät hat einen Timeout	Ein Timeout kann während des Transfers von Pufferspeichern auftreten. Für die CM Analysekette ist dies meistens nicht kritisch. Es kommt auf die Art des Algorithmus an und auf den Ort, wo genau der Fehler aufgetreten ist, wie auf den Fehler reagiert werden muss. Der Eingang Timeout sollte nur dann angehoben werden, wenn dies zur Zykluszeit der Task passt. Siehe Kapitel Parallelverarbeitung [▶ 77].

In manchen Fällen ist das Error Handling mit Error Codes nicht die beste Wahl, insbesondere dann wenn die Aktionen einen undefinierten Wert ausgeben, in Bezug auf unübliche, aber mögliche Eingangsdaten. Oder wenn Werte aus dem Prozess ausgeschlossen wurden. In diesem Fall können fehlende Werte und teilweise undefinierte Ergebnisse durch die spezielle Konstante NaN beschrieben werden (siehe dazu: Kapitel NaN values [▶ 76]). Dies wird im Falle von Fehlern genutzt dessen Erscheinung nicht von der Programmlogik abhängig ist, aber von bestimmten Input Daten.

7.2 ADS Return Codes

Gruppierung der Fehlercodes:
Globale Fehlercodes: 0x0000 [▶ 387]... (0x9811_0000 ...)
Router Fehlercodes: 0x0500 [▶ 387]... (0x9811_0500 ...)
Allgemeine ADS Fehler: 0x0700 [▶ 388]... (0x9811_0700 ...)
RTime Fehlercodes: 0x1000 [▶ 389]... (0x9811_1000 ...)

Globale Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x0	0	0x98110000	ERR_NOERROR	Kein Fehler.
0x1	1	0x98110001	ERR_INTERNAL	Interner Fehler.
0x2	2	0x98110002	ERR_NORTIME	Keine Echtzeit.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Zuweisung gesperrt - Speicherfehler.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Postfach voll – Es konnte die ADS Nachricht nicht versendet werden. Reduzieren der Anzahl der ADS Nachrichten pro Zyklus bringt Abhilfe.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Falsches HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Ziel-Port nicht gefunden – ADS Server ist nicht gestartet, nicht erreichbar oder nicht installiert.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Zielrechner nicht gefunden – AMS Route wurde nicht gefunden.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unbekannte Befehl-ID.
0x9	9	0x98110009	ERR_BADTASKID	Ungültige Task-ID.
0xA	10	0x9811000A	ERR_NOIO	Kein IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unbekannter AMS-Befehl.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 Fehler.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port nicht verbunden.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Ungültige AMS-Länge.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Ungültige AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installations-Level ist zu niedrig –TwinCAT 2 Lizenzfehler.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	Kein Debugging verfügbar.
0x12	18	0x98110012	ERR_PORTDISABLED	Port deaktiviert – TwinCAT System Service nicht gestartet.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port bereits verbunden.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 Fehler.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync Fehler.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	Keine Index-Map für AMS Sync vorhanden.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Ungültiger AMS-Port.
0x19	25	0x98110019	ERR_NOMEMORY	Kein Speicher.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP Sendefehler.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host nicht erreichbar.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Ungültiges AMS Fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS Sendefehler – Secure ADS Verbindung fehlgeschlagen.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Zugriff Verweigert – Secure ADS Zugriff verweigert.

Router Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	Das Debug Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.

Hex	Dec	HRESULT	Name	Beschreibung
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	Die Portnummer ist bereits vergeben.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	Die maximale Portanzahl ist erreicht.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	Das Postfach hat die maximale Anzahl für fragmentierte Nachrichten erreicht.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	Fragment Timeout aufgetreten.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	Port wird entfernt.

Allgemeine ADS Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	Allgemeiner Gerätefehler.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Lesen oder Schreiben nicht gestattet. Mehrere Ursachen sind möglich. Beispielsweise beim Anlegen von Routen, dass ein falsches Passwort angegeben wurde.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Ungültige Daten-Werte.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Gerät nicht betriebsbereit.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Gerät beschäftigt.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext vom Betriebssystem - Kann durch Verwendung von ADS Bausteinen in unterschiedlichen Tasks auftreten. Abhilfe kann die Multitasking-Syncronisation in der SPS geben.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig – Kann durch einen Online-Change auftreten. Erzeuge einen neuen Handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Gerät (Server) ist im ungültigen Zustand.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	Keine weiteren Handles verfügbar.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Anforderung steht aus.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal-Warnung.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv.

Hex	Dec	HRESULT	Name	Beschreibung
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert. Mehrere Ursachen sind möglich. Beispielsweise, dass eine Unidirectionale ADS Route in die umgekehrte Richtung verwendet wird.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	Lizenzproblem: System-ID ist ungültig.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Zertifikat ungültig.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public Key vom OEM nicht bekannt.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	Lizenz nicht gültig für diese System.ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo-Lizenz untersagt.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Funktions-ID ungültig.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Außerhalb des gültigen Bereiches.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Ungültiges Alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Ungültiger Plattform Level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Kontext – Weiterleitung zum Passiv-Level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Kontext – Weiterleitung zum Dispatch-Level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Kontext – Weiterleitung zur Echtzeit.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Clientfehler.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten – Die Gegenstelle antwortet nicht im vorgegebenen ADS Timeout. Die Routeneinstellung der Gegenstelle kann falsch konfiguriert sein.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port nicht geöffnet.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	Keine AMS Adresse.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Interner Fehler in Ads-Sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Überlauf der Hash-Tabelle.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Schlüssel in der Tabelle nicht gefunden.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	Keine Symbole im Cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	Die Anfrage wurde abgebrochen.

RTime Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x1000	4096	0x98111000	RTERR_INTERNAL	Interner Fehler im Echtzeit-System.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer-Wert nicht gültig.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack-Pointer hat den ungültigen Wert 0 (null).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	Die Request Task Priority ist bereits vergeben.
0x1005	4101	0x98111005	RTERR_NOMORETCB	Kein freier TCB (Task Control Block) verfügbar. Maximale Anzahl von TCBs beträgt 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.

Hex	Dec	HRESULT	Name	Beschreibung
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	Kein externer Sync-Interrupt angewandt.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs-Interrupts ist fehlgeschlagen.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

Spezifische positive HRESULT Return Codes:

HRESULT	Name	Beschreibung
0x0000_0000	S_OK	Kein Fehler.
0x0000_0001	S_FALSE	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch ein negatives oder unvollständiges Ergebnis erzielt wurde.
0x0000_0203	S_PENDING	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch noch kein Ergebnis vorliegt.
0x0000_0256	S_WATCHDOG_TIMEOUT	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch eine Zeitüberschreitung eintrat.

TCP Winsock-Fehlercodes

Hex	Dec	Name	Beschreibung
0x274C	10060	WSAETIMEDOUT	Verbindungs Timeout aufgetreten - Fehler beim Herstellen der Verbindung, da die Gegenstelle nach einer bestimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung konnte nicht aufrecht erhalten werden, da der verbundene Host nicht reagiert hat.
0x274D	10061	WSAECONNREFUSED	Verbindung abgelehnt - Es konnte keine Verbindung hergestellt werden, da der Zielcomputer dies explizit abgelehnt hat. Dieser Fehler resultiert normalerweise aus dem Versuch, eine Verbindung mit einem Dienst herzustellen, der auf dem fremden Host inaktiv ist—das heißt, einem Dienst, für den keine Serveranwendung ausgeführt wird.
0x2751	10065	WSAEHOSTUNREACH	Keine Route zum Host - Ein Socketvorgang bezog sich auf einen nicht verfügbaren Host.
Weitere Winsock-Fehlercodes: Win32-Fehlercodes			

7.3 Optionen der Spektrumsskalierung

Diese Seite bietet eine Übersicht der Skalierungsoptionen für Spektralberechnungen. Die folgende Tabelle zeigt Symbole und wichtige Parameter für die Skalierung.

Symbol	Baustein-Parameter	Bedeutung
N	nFFT_Length	Zahl der Eingangswerte der FFT
F_s		Abtastfrequenz oder Samplingrate
Σw_n	eWindowFunction, nWindowLength	Summe der Werte der Fensterfunktion
Σw_n^2	eWindowFunction, nWindowLength	Summe der quadrierten Werte der Fensterfunktion
SQRT(x)		Quadratwurzel von X
$MAX(\|X_n\|)$		Maximum der Spektralwerte X_n
$RMS(x_n) = SQRT([\Sigma (x_n)^2] / N)$		Root Mean Square Wert oder Effektivwert eines Signals
$PSD(X_n)$		Power Spectral Density
$LSD(X_n)$		Linear Spectral Density

Symbol	Baustein-Parameter	Bedeutung
A		Amplitude eines Referenz-Sinussignals

Die folgende Tabelle listet voreingestellte Skalierungsoptionen auf (vom Typ E_CM_ScalingType [▶ 290]), die beispielsweise bei den Bausteinen FB_CMA_PowerSpectrum [▶ 202] und FB_CMA_MagnitudeSpectrum [▶ 172] und davon abgeleiteten Bausteinen ausgewählt werden können. Die resultierenden Faktoren müssen nicht vom Anwender ausgewertet werden. Sie sind in der zweiten Spalte angegeben, um bei Bedarf weitere Parameter einbeziehen zu können. Die Werte x_n bezeichnen die Eingangswerte des Bausteins und die Werte X_k den mit der Skalierung resultierenden Spektralwert für den Frequenzkanal *k*.

Skalierungsoption	Enthaltener Faktor	Beschreibung
Deterministische Signale		
`eCM_PeakAmplitude`	$2 / \Sigma w_n$	Diese Skalierung passt die Magnitudenwerte so an, dass ein Eingangs-Sinussignal mit der Amplitude A einen Maximalwert von A erreicht. Das Resultat ist unabhängig von der Art der Fensterfunktion. Die Einheit der Magnitudenwerte ist gleich der Einheit des Eingangssignals. $MAX(\|X_k\|) = A$ Die Maximalwerte des Spektrums ermöglichen allerdings keine robuste Abschätzung der Amplitude, da sogenannte Scalloping Losses auftreten können.
`eCM_RootPowerSum`	$2 / SQRT(N * \Sigma w_n^{\ 2})$	Die Skalierung passt die Spektralwerte so an, dass für ein Eingangs-Sinussignal mit der Amplitude A, die Wurzel aus der Summe der Power-Werte den Wert A hat. Entsprechend kann auch die Wurzel aus der Summe der Quadrate der Magnitudenwerte verwendet werden. Somit ist das Ergebnis gleich dem Effektivwert des Eingangssignals multipliziert mit SQRT(2). $SQRT(\Sigma\|X_k\|^2) = A$ Die Skalierung ist zur Bewertung schmalbandiger Signale geeignet. Da die Summierung über benachbarte Frequenzbänder Scalloping Losses reduziert, ist sie deutlich robuster als `eCM_PeakAmplitude`.
`eCM_RMS`	$SQRT(2/N * \Sigma w_n^{\ 2})$	Diese Skalierung resultiert in Power-Werten, deren Summe mit nachfolgendem Ziehen der Quadratwurzel gleich dem Effektivwert des Eingangssignals ist. Ein Sinussignal mit der Amplitude A resultiert in einen Wert von A/SQRT(2): $SQRT(\Sigma\|X(k)\|^2) = RMS(x_n) = A * SQRT(1/2)$ Auch diese Skalierung ist wie `eCM_ROOT_POWER_SUM` robust und zur Bewertung schmalbandiger Signale geeignet. Zusätzlich ist der RMS Wert auch für breitbandige Signale wohldefiniert.
Stochastische und breitbandige Signale		
`eCM_PowerSpectralDensity`	$SQRT(2 / \Sigma w_n^2)$	Diese Skalierung ermittelt die Power Spectral Density (PSD). Für breitbandige und stochastische Signale ist diese unabhängig von den Parametern der FFT und Fensterfunktion. $PSD(X_k) = \|X_k\|^2/F_S$ Um eine physikalisch korrekte Spektrale Leistungsdichte zu ermitteln, muss das Ergebnis noch durch die Abtastrate des Eingangssignals in Hertz geteilt werden. Wenn das Eingangssignal die Einheit Volt hat, erhält man somit für die Magnitude

Skalierungsoption	Enthaltener Faktor	Beschreibung
		die Einheit 1 V/ Hz und für die Leistungsdichte die Einheit 1 V^2/Hz. Für die Linear Spectral Density muss durch die *Wurzel der Abtastrate* geteilt werden, die Einheit ist dann 1 V/(1 Hz)$^{1/2}$: $LSD(X_k) = \|X_k\| / SQRT(F_S)$
Elementar		
`eCM_DiracScaling`	$sqrt(N / \Sigma w_n^{\ 2})$	Diese Skalierung normalisiert das PowerSpektrum so, dass das breitbandige Signal gleich der unskalierten FFT (mit der oben angegebenen Definition) ist. Der Einfluss von Fenstertyp und Fensterlänge wird also eliminiert. Der Einfluss der FFT-Länge N ist jedoch genauso wie bei der unskalierten FFT vorhanden.
`eCM_NoScaling`	1	Keine Skalierung. Das Resultat besteht aus der Anwendung der Fensterfunktion (die konventionsgemäß stets ein Maximum von Eins hat) gefolgt von der FFT.

7.4 Support und Service

Beckhoff und seine weltweiten Partnerfirmen bieten einen umfassenden Support und Service, der eine schnelle und kompetente Unterstützung bei allen Fragen zu Beckhoff Produkten und Systemlösungen zur Verfügung stellt.

Downloadfinder

Unser Downloadfinder beinhaltet alle Dateien, die wir Ihnen zum Herunterladen anbieten. Sie finden dort Applikationsberichte, technische Dokumentationen, technische Zeichnungen, Konfigurationsdateien und vieles mehr.

Die Downloads sind in verschiedenen Formaten erhältlich.

Beckhoff Niederlassungen und Vertretungen

Wenden Sie sich bitte an Ihre Beckhoff Niederlassung oder Ihre Vertretung für den lokalen Support und Service zu Beckhoff Produkten!

Die Adressen der weltweiten Beckhoff Niederlassungen und Vertretungen entnehmen Sie bitte unserer Internetseite: www.beckhoff.com

Dort finden Sie auch weitere Dokumentationen zu Beckhoff Komponenten.

Beckhoff Support

Der Support bietet Ihnen einen umfangreichen technischen Support, der Sie nicht nur bei dem Einsatz einzelner Beckhoff Produkte, sondern auch bei weiteren umfassenden Dienstleistungen unterstützt:

- Support
- Planung, Programmierung und Inbetriebnahme komplexer Automatisierungssysteme
- umfangreiches Schulungsprogramm für Beckhoff Systemkomponenten

Hotline: +49 5246 963-157
E-Mail: support@beckhoff.com

Beckhoff Service

Das Beckhoff Service-Center unterstützt Sie rund um den After-Sales-Service:

- Vor-Ort-Service

- Reparaturservice
- Ersatzteilservice
- Hotline-Service

Hotline:	+49 5246 963-460
E-Mail:	service@beckhoff.com

Beckhoff Unternehmenszentrale

Beckhoff Automation GmbH & Co. KG

Hülshorstweg 20
33415 Verl
Deutschland

Telefon:	+49 5246 963-0
E-Mail:	info@beckhoff.com
Internet:	www.beckhoff.com

Glossar

Abtastfrequenz

Abtastfrequenz ist die Frequenz, mit der das analoge Signal ursprünglich abgetastet und in digitale Werte umgewandelt wird. Diese Umwandlung geschieht in Schritten mit konstanter zeitlicher Länge, der Abtastperiode. Der Kehrwert der Abtastperiode heißt Abtastfrequenz, sie wird in Hertz angegeben. Siehe auch "Abtasttheorem."

Aliasing

Aliasing ist ein Fehler, der auftritt, wenn in einem Signal Frequenzen auftreten, die höher sind als die halbe Abtastrate. In diesem Fall kann das Signal aus der Abtastung nicht mehr eindeutig rekonstruiert werden (Nyquist-Theorem). Im Spektrum schlagen sich solche Frequenzen als sogenannte Spiegelfrequenzen nieder.

Artefakte

Unerwünschte Veränderungen im Signal, die aus Fehlern bei der Verarbeitung resultieren, beispielsweise aufgrund von Aliasing.

Bessel'sche Korrektur

Korrektur, welche die Zahl der Freiheitsgrade beim Schätzen von statistischen Momentenkoeffizienten aus einer Datenreihe berücksichtigt. Konkret wird z.B. die Standardabweichung durch Multiplikation mit dem Faktor sqrt(n/(n-1)) korrigiert, die Schiefe durch sqrt(n*(n-1)/(n-2)) und so weiter. Für größere n ist der Faktor im Allgemeinen vernachlässigbar.

Bin

Bezeichnet einen Kanal eines mehrkanaligen Signalausgangs. Die Bezeichnung wird besonders bei Transformationen, welche Signale umwandeln, verwendet, wie der FFT oder der Bildung des Histogramms.

Cepstrum

Das ist eine Transformation, welche auf der Frequenzanalyse aufbaut und periodische Elemente im Spektrum aufgrund von Oberschwingungen oder Amplitudenmodulationen hervorhebt. Dabei wird zwischen dem Power Cepstrum und dem Komplexen Cepstrum unterschieden.

Crest Faktor

auf Deutsch auch Scheitelfaktor oder manchmal Spitzenhaltigkeit: Verhältnis zwischen Maximum des Betrags und Effektivwert eines Signals, normalerweise ausgedrückt in Dezibel.

Dezibel oder dB

Logarithmische Skala zur Bewertung der Intensität von Schwingungen oder von Intensitätsverhältnissen. Ein Dezibel (Symbol dB) ist definiert als ein Zehntel der Hilfsmaßeinheit Bel. Wenn x ein Leistungswert ist, ist y = 20 * log10(x/x0) der Wert in Dezibel. Dabei wird für x0 der Wert Eins oder ein festgelegter Referenzwert verwendet.

Fensterfunktionen

Funktionen, welche in Verbindung mit z.B. einer Frequenzanalyse (Welch-Methode) dazu verwendet werden, lange Eingangssignale ohne Hinzufügen künstlicher Sprünge zu zerlegen. Standardmäßig kann in fast allen Fällen das Hann-Fenster verwendet werden. Die Wahl der Fensterfunktion hat Auswirkungen auf die Frequenz- und Zeitauflösung der Frequenzanalyse.

Fensterung (Windowing)

bezeichnet den Rechenschritt der Multiplikation mit einer Fensterfunktion.

FFT

FFT oder Fast Fouriertransformation: Schnelle Fouriertransformation, ein Berechnungsverfahren zur Berechnung der Diskreten Fouriertransformation. Genaugenommen existieren mehrere solcher Rechenverfahren, wobei die gängigen Implementationen nur Zweierpotenzen als Eingangslänge zulassen (Cooley-Tukey Algorithmus). Gemeinsames Merkmal ist eine Komplexität der Ordnung O(n * nlog(n)), d.h. die Berechnung einer FFT mit 2048 Punkten ist etwas mehr als vier mal so aufwendig wie für 512 Punkte.

Fouriertransformation

Transformation, welche es ermöglicht, ein Zeitsignal in unterschiedliche Frequenzanteile zu zerlegen und damit die Grundlage für viele Methoden der Frequenzanalyse bildet. Hierfür wird in der Praxis anstelle der kontinuierlichen Fouriertransformation, die eine kontinuierliche Funktion eines unendlichen Signals darstellt, in der Regel die diskrete Fouriertransformation (DFT) verwendet, die für ein diskretes, periodisches Signal definiert ist. Eine effiziente Implementation der Diskreten Fouriertransformation, die große praktische Bedeutung hat, ist die Fast Fouriertransformation (FFT) oder Schnelle Fouriertransformation.

Frequenzbereich

oder Frequenzraum bezeichnet die Darstellung eines Signals anhand der Werte der FFT. Da das komplexe Fourier-Spektrum jedes Signal eindeutig darstellen kann und sich ohne Verlust wieder in ein äquivalentes Zeitsignal zurück transformieren lässt, repräsentieren Frequenzbereich und Zeitbereich (als sogenannte "orthonormale Basen" im Funktionenraum) äquivalente Darstellungen desselben Signals. Viele Operationen zur Signalanalyse sind im Frequenzbereich einfacher und effizienter vorzunehmen als im Zeitbereich.

Hilbert-Transformation

Transformation, welche aus einem Schwingungssignal effizient das um neunzig Grad phasenverschobene Signal ermittelt. Die Hilbert-Transformation dient beispielsweise zur Berechnung des Analytischen Signals.

Komplexität

Hier: Angabe der benötigten Ressourcen eines Algorithmus (Rechenzeit und gegebenenfalls Speicherplatz). Funktionen zum Condition Monitoring werden mit stark unterschiedlichen Datenmengen aufgerufen; während eine Kurzzeit-FFT eventuell nur mit 32 Werten aufgerufen wird, kann es ohne weiteres sinnvoll sein, z.B. ein Cepstrum für 16000 Werte zu berechnen. Deswegen wird bei einer variablen Zahl von Eingangsdaten n betrachtet, wie sich der Algorithmus mit ansteigender Datenmenge verhält; dies wird in der Informatik üblicherweise durch die Schreibweise O(f(n)) beschrieben (auch "Landau-Notation" genannt). Diese Notation besagt, dass die Komplexität mit zunehmendem n nicht wesentlich schneller wächst als eine Funktion f(n). Ein Algorithmus mit der Rechenzeit-Komplexität O(n) benötigt also z.B. bei achtfacher Datenmenge n die achtfache Rechenzeit, ein Algorithmus der Komplexität O(n2) bereits die vierundsechzigfache. Eine FFT der Komplexität O(n * log2 n) benötigt hingegen für n=16384 die 112-fache Rechenleistung gegenüber n=256. Bei geringer Datenmenge dominiert in der Regel ein von der Anzahl der Eingangsdaten unabhängiger Anteil den Rechenaufwand.

Kontaktwinkel

Winkel zwischen der Linie, auf der die Kugeln eines Kugellagers die Lauffläche berühren und jener Ebene, welche senkrecht zur Achse des Lagers ist. Während der Kontaktwinkel bei Lagern, die ausschließlich für radiale Belastungen konstruiert sind, stets nahe bei Null ist, kann er bei Lagern, die auch axiale Belastungen aufnehmen, wesentlich größer sein. Er hängt also sowohl von der Geometrie als auch von der aktuellen Belastung des Lagers ab und hat aufgrund der Auswirkungen auf den Rollkreisdurchmesser Einfluss auf die beobachtbaren Schadfrequenzen. Diese sind bei Lagern für axiale Belastungen also nicht konstant.

Kurtosis

(manchmal auch Curtosis oder Wölbung): Maß für die "Breitschultrigkeit" bzw. "Spitzenhaltigkeit" einer statistischen Verteilung von Werten, ermittelt aus dem vierten zentralen statistischen Moment. Zur besseren Beurteilung von Verteilungen wird häufig die Abweichung zwischen Kurtosis einer gemessenen Verteilung und Kurtosis der Normalverteilung (besitzt den Wert 3) genutzt. Diese wird als Exzess Kurtosis (auch nur Exzess) bezeichnet. Eine Gaussverteilung hat entsprechend die Exzess Kurtosis Null, eine Verteilung mit vielen Ausreißern erzielt einen Wert deutlich über Null.

Maschinenschutz

bezeichnet Verfahren, die eine möglichst schnelle automatische Abschaltung einer Anlage anstreben, falls Überwachungsparameter eine kritische Schwelle überschreiten. Auf diese Weise können Unfälle und Folgeschäden vermieden werden.

Momentenkoeffizienten

Sammelbegriff für statistische Werte wie Mittelwert, Standardabweichung, Schiefe (Skew) und Kurtosis von statistischen Variablen. Sie heißen so, weil sie aus den zentralen statistischen Momenten der Wahrscheinlichkeitsverteilungen oder Histogramme dieser Variablen berechnet werden können.

NaN (Not a Number)

symbolische Konstante, welche nach dem Standard IEEE 754 ungültige oder fehlende Werte markiert. Zu den wesentlichen Eigenschaften von NaN Werten zählen die folgenden Punkte: Alle arithmetischen Operationen, die NaN als Eingangsdaten verwenden, liefern wiederum NaN als Ergebnis. Alle relationalen Operatoren =, !=, > < >= <= liefern stets den Wert False, wenn mindestens einer der Operanden NaN ist. Die Standard-Funktion isnan bzw. _isnan liefert den Wert True, wenn das Argument den Wert NaN hat. Der Ausdruck isnan(a) ist äquivalent dem Ausdruck !(a == a) bzw. NOT(a = a). Die Tatsache, dass NaN-Werte sich bei der Verwendung in weiteren Berechnungen fortpflanzen, hat den Vorteil, dass ungültige Werte nicht übersehen werden können.

Nyquist-Theorem oder Abtasttheorem

Theorem der Nachrichtentechnik und Signalverarbeitung, welches leicht vereinfacht besagt, dass ein kontinuierliches Signal mit einer Frequenz größer als dem doppelten der höchsten enthaltenen Frequenz abgetastet werden muss, damit man aus dem so erhaltenen zeitdiskreten Signal das Ursprungssignal ohne Informationsverlust oder Mehrdeutigkeiten rekonstruieren kann. Diese Maximalfrequenz wird als Nyquist-Frequenz bezeichnet. In der Praxis werden in den meisten D/A Wandlern Eingangsfilter eingesetzt, welche die maximale Frequenz des Eingangssignals auf einen Wert kleiner als die halbe Abtastrate beschränken.

Oberschwingungen

(Oft auch Harmonische) sind Schwingungen, die als ganzzahlige Vielfache einer Grundfrequenz auftreten. Sie sind charakteristisch für impulshafte Anregungen und nichtlineare Effekte am Ort der Schwingungsentstehung und in diesem Fall typischerweise an Liniengruppen im Spektrum zu erkennen, die voneinander einen konstanten Abstand haben.

Overlap-Add Methode

Verfahren, das es ermöglicht, ein Signal ohne Informationsverlust zunächst in Kurzzeitspektren zu zerlegen, dann im Frequenzbereich weiter zu verarbeiten (z.B. zu filtern) und dann wieder als kontinuierliches Zeitsignal zu rekonstruieren. Zu ausführlicheren Informationen siehe die Beschreibung der Overlap-Add Methode im einführenden Teil.

Quantil oder Perzentil

Bezeichnung eines Wertes, der aus einer statistischen Größe ermittelt wird. Zunächst wird deren empirische Häufigkeitsverteilung (Dichtefunktion) bestimmt und aus dieser die Summenhäufigkeitsverteilung (auch kumulative Verteilungsfunktion genannt) berechnet. Der Wert des Perzentils q ist der Wert, den die Zufallsgröße in q Prozent aller Fälle maximal erreicht, aber nicht überschreitet. Dieser Wert wird durch Bildung der Umkehrfunktion der Summenhäufigkeitsverteilung bestimmt. Quantile verwenden als einzigen Unterschied zu Perzentilen statt Prozentzahlen den entsprechenden Dezimalbruch. Der Wert des 50-Prozent-Perzentils wird auch als Median bezeichnet.

Quefrency (deutsch auch "Quefrenz")

Bezeichnung für die Zeitachse, die sich bei der Berechnung des Cepstrums ergibt. Der Name deutet als "verwürfelte" Umkehrung des Begriffs "Frequency" die für das Cepstrum charakteristischen Operationen "Umkehrung" und "Neusortierung" an. Durch die Aufeinanderfolge von zwei Fourier-Transformationen ergibt sich zunächst eine Transformation in den Frequenzbereich, mit der zugeordneten Einheit 1 Hertz. Die zweite Transformation führt wiederum in einen Zeitbereich, bei dem jedoch nicht mehr die absolute Zeit auf der Achse liegt, sondern die durch das Cepstrum ermittelten Periodendauern. Die Einheit der Quefrency ist eine Sekunde.

RCFA oder Root Cause Failure Analysis

Bezeichnung für die Analyse zur Ermittlung von primären Ursachen für Schäden. Dies hat eine besondere Bedeutung bei Wälzlagern, da primäre Schäden zu auffälligeren Folgeschäden führen. Eine Ermittlung der Ursachen ermöglicht es, die Entstehung von Schäden wirkungsvoll zu reduzieren.

Scalloping

Effekt, dass der genaue Spektralwert schmalbandiger Signale (etwa eines Sinussignals oder eines Kalibrators) davon abhängt, an welcher Stelle des FFT-Kanals die Frequenz des Kanals liegt. Das Ausmaß des Effekts hängt von der Fensterfunktion ab. Eine Erörterung des Effekts findet sich z.B. im Artikel von Frederic Harris von 1978 auf Seite 57. Literatur: Frederic J. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform", Proceedings of the IEEE, Vol 66(1), Jan. 1978.

Schadfrequenzen

Charakteristische Frequenzen, die bei einem Schaden bestimmter Maschinenelemente auftreten. Beispielsweise sind Schäden von Wälzelementen, Innenring, Außenring und Käfig in Wälzlagern bestimmte Frequenzen zugeordnet, die – abhängig vom Kontaktwinkel– in einem bestimmten Verhältnis zur Umdrehungsgeschwindigkeit der Achse stehen.

Schiefe (Skew)

Maß für die Asymmetrie einer statistischen Verteilung, ermittelt aus dem dritten zentralen statistischen Moment. Eine symmetrische Verteilung hat die Schiefe Null.

Spektrale Beschleunigungsdichte

Oder Acceleration Spectral Density (ASD) ist die Bezeichnung für die physikalische Größe, welche die Ausgangswerte der Fouriertransformation repräsentieren, wenn das Eingangssignal ein Beschleunigungssignal ist, wie es beispielsweise ein piezoelektrischer Schwingungsaufnehmer misst. Die Beschleunigungsdichte integriert über einen Frequenzintervall ergibt hierbei — ganz analog wie dieLeistungs-

dichte — eine frequenzspezifische Beschleunigung. Die gängige Einheit ist 1 Millimeter pro Sekundenquadrat pro Hertz = 1 mm/s2/Hz.

Zahneingriffsfrequenz

Bezeichnet die Frequenz, mit der die Zahnpaare in einem Getriebe sich berühren. Dieser Kontakt verursacht die sogenannte Zahneingriffsschwingung

Zeitbereich

Bezeichnet die Darstellung eines Signals anhand der zeitlich abgetasteten Werte, wie es nach einer Messung ursprünglich vorliegt. Da auch das Fourier-Spektrum jedes Signal eindeutig darstellen kann und sich ohne Verlust wieder in ein äquivalentes Zeitsignal zurück transformieren lässt, repräsentieren Zeitbereich und Frequenzbereich (als sogenannte "orthonormale Basen" im Funktionenraum) äquivalente Darstellungen desselben Signals.

Zero Padding

Bezeichnet einen Verarbeitungsschritt, der angewandt wird, wenn eine FFT mit einer bestimmten Länge aus einer kleineren Anzahl von Samples berechnet werden soll. Hierzu werden die Werte der Zeitreihe vorne und hinten mit Nullen aufgefüllt, bis die gewünschte Zahl der Werte erreicht ist. Dies setzt in der Regel die Fensterung des Signals z.B. nach der Welch-Methode voraus, damit keine unechten Sprünge in der Zeitreihe entstehen. Zero Padding erhöht die Frequenzauflösung einer FFT, welche gleich der Abtastrate geteilt durch die Zahl der FFT Punkte ist, der Informationsgehalt des ursprünglichen Signals wird jedoch natürlich nicht erhöht. Bei spektralen Modifikationen des Signals kann Zero Padding notwendig sein, um zirkuläres Aliasing zu unterdrücken.

Zirkuläres Aliasing

Artefakt, das auftreten kann, wenn Signale im Frequenzbereich modifiziert werden und dann mit einer inversen FFT wieder in den Zeitbereich zurück transformiert werden (Overlap-Add Methode). Die Modifikation lässt sich als eine Multiplikation im Frequenzbereich beschreiben, die ganz allgemein einer Filterung im Zeitbereich entspricht. Diese ist äquivalent einer zyklisch definierten Faltung mit der Impulsantwort der Filterung. Wenn die Impulsantwort zu lang ist, so treten Signalanteile, welche dem Anfang des Zeitabschnitts zugehören, am Ende des Abschnitts in Erscheinung und umgekehrt. Dies ist begründet in der zyklischen Definition der Diskreten Fourier-Transformation. Starke Modifikationen im Frequenzbereich können also zu Artefakten führen. Als Gegenmaßnahme kann man das Zeitsignal vor der Verarbeitung durch Nullen ergänzen (Zero Padding), so dass eine Reserve für die Verlängerung des Signals entsteht.

Mehr Informationen:
www.beckhoff.com/tf3600

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