

Meilhaus Electronic Handbuch
Meilhaus Intelligent Driver System
ME-iDS 2.0D



Intelligentes Treibersystem für
Windows 2000/XP/Vista, Windows 7 und Linux

Impressum

Handbuch Meilhaus Intelligent Driver System (ME-iDS)

Revision 2.0D

Ausgabedatum: 2. August 2012

Meilhaus Electronic GmbH

Fischerstraße 2

D-82178 Puchheim bei München

Germany

<http://www.meilhaus.de>

© Copyright 2012 Meilhaus Electronic GmbH

Alle Rechte vorbehalten. Kein Teil dieses Handbuches darf in irgendeiner Form (Fotokopie, Druck, Mikrofilm oder in einem anderen Verfahren) ohne ausdrückliche schriftliche Genehmigung der Meilhaus Electronic GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Wichtiger Hinweis:

Alle in diesem Handbuch enthaltenen Informationen wurden mit größter Sorgfalt und nach bestem Wissen zusammengestellt. Dennoch sind Fehler nicht ganz auszuschließen.

Aus diesem Grund sieht sich die Firma Meilhaus Electronic GmbH dazu veranlaßt, darauf hinzuweisen, daß sie weder eine Garantie (abgesehen von den im Garantieschein vereinbarten Garantieansprüchen) noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernehmen kann.

Für die Mitteilung eventueller Fehler sind wir jederzeit dankbar.

Delphi ist ein Warenzeichen von Embarcadero Technologies, Inc.

Visual C++ und VisualBASIC sind Warenzeichen von Microsoft.

VEE Pro und VEE Express sind Warenzeichen von Agilent Technologies.

ME-VEC und ME-FoXX sind Warenzeichen von Meilhaus Electronic.

Weitere der im Text erwähnten Firmen- und Produktnamen sind eingetragene Warenzeichen der jeweiligen Firmen.

Inhalt

1	Einführung	7
1.1	Unterstützte Hardware	9
1.2	Systemanforderungen.....	10
1.3	Nomenklatur	11
1.4	Dokumentation	12
2	Installation.....	13
2.1	Installation unter Windows	13
2.2	Installation unter Linux.....	13
2.3	Konfigurationsprogramm (ME-iDC)	14
2.3.1	Subdevice-Konfiguration	16
2.3.2	Firmware-Konfiguration	16
2.3.3	Remote-Gerät anmelden	16
2.4	Einstellen der IP-Adresse.....	18
3	Programmierung	21
3.1	Architektur des Treibersystems	21
3.1.1	Bibliotheksdateien	22
3.2	Sprachen-Unterstützung	23
3.2.1	Hochsprachen-Unterstützung	23
3.2.2	Graphische Programmierwerkzeuge	23
3.3	Konzept der Bibliothek	24
3.3.1	Hierarchie-Ebenen	24
3.3.2	Properties	26
3.3.2.1	Property-Pfade	27
3.3.2.2	Abkürzungen für Property-Pfade	28
3.3.2.3	Property-Funktionen	28
3.3.2.3.1	Lesen von Property-Werten	29
3.3.2.3.2	Schreiben von Property-Werten	29
3.3.2.4	Attribute.....	30
3.3.2.5	Property-Typen	32
3.3.2.6	Zugriffstyp von Properties.....	33
3.3.2.7	Systemattribute	34
3.3.2.8	Allgemeine Device-Properties.....	34
3.3.2.9	Subdevice-Properties	35
3.3.2.10	Properties von Konfigurations-Containern	35

3.3.3	Subdevices	37
3.3.3.1	Analoge Ein-/Ausgabe.....	37
3.3.3.2	Digitale Ein-/Ausgabe.....	37
3.3.3.3	Frequenz-Ein-/Ausgabe	38
3.3.3.4	Zähler	38
3.3.3.5	Interrupt.....	38
3.3.3.6	„FPGA“	39
3.3.4	Struktur der API	39
3.3.4.1	Query-Funktionen.....	39
3.3.4.2	Property-Funktionen	39
3.3.4.3	Ein-/Ausgabe-Funktionen.....	40
3.3.4.4	Hilfs-Funktionen	40
3.3.5	Grundsätzliche Vorgehensweise	41
3.3.5.1	Initialisierung	41
3.3.5.2	Zugriffsschutz	41
3.3.5.3	Fehlerbehandlung	43
3.4	Betriebsarten	45
3.4.1	Single-Betrieb	45
3.4.1.1	Operation starten/Trigger-Optionen	46
3.4.1.2	Analoge Ein-/Ausgabe	48
3.4.1.3	Digitale Ein-/Ausgabe	49
3.4.1.4	Frequenz-Ein-/Ausgabe	50
3.4.1.4.1	Frequenzmessung	51
3.4.1.4.2	Impulsgenerator	53
3.4.1.5	Zählerbetrieb	55
3.4.1.5.1	Modus 0: Zustandsänderung bei Nulldurchgang	57
3.4.1.5.2	Modus 1	58
3.4.1.5.3	Modus 2	58
3.4.1.5.4	Modus 3	58
3.4.1.5.5	Modus 4	59
3.4.1.5.6	Modus 5	59
3.4.1.5.7	Modus „Pulsweiten-Modulation“	60
3.4.2	Streaming-Betrieb	62
3.4.2.1	Hardware-Ressourcen ermitteln	63
3.4.2.2	Hardware konfigurieren	63
3.4.2.3	Kanalliste	63
3.4.2.4	Triggerstruktur	64
3.4.2.4.1	Timing Stream-Timer	69
3.4.2.4.2	Timing Stream-Trigger-Sample.....	71
3.4.2.4.3	Timing Stream-Trigger-List	73

3.4.2.5	Daten lesen	75
3.4.2.5.1	Vorgehensweise Daten lesen	75
3.4.2.5.2	Lesen ohne Callback-Funktion	77
3.4.2.5.3	Lesen mit Callback-Funktion	78
3.4.2.6	Daten schreiben	79
3.4.2.6.1	Vorgehensweise Daten schreiben	80
3.4.2.6.2	Schreiben ohne Callback-Funktion	82
3.4.2.6.3	Schreiben mit Callback-Funktion	83
3.4.2.6.4	Wraparound-Option	84
3.4.2.7	Streaming-Betrieb stoppen	84
3.4.3	Spezielle Features	86
3.4.3.1	Sample & Hold	86
3.4.3.2	Bitmuster-Ausgabe der ME-4680	86
3.4.3.3	Synchronstart	88
3.4.3.4	Offset-Einstellung	89
3.4.4	Interrupt-Betrieb	90
4	Funktionsreferenz	93
4.1	Allgemeine Hinweise	93
4.2	Beschreibung der API-Funktionen	94
4.2.1	Query-Funktionen	97
4.2.2	Property-Funktionen.....	122
4.2.3	Ein-/Ausgabe-Funktionen	135
4.2.4	Hilfs-Funktionen	197
Anhang	221
A	Technische Fragen	221
A1	Fax-Hotline	221
A2	Serviceadresse	221
A3	Treiber-Update.....	221
B	Spezielle Betriebsarten	222
B1	Betriebsarten 8254	222
B2	Pulsweiten-Modulation	226
B3	Bitmuster-Ausgabe der ME-4680	228
B4	MEphisto Scope	230
B5	ME-MultiSig-Steuerung	231
C	Subdevice-Caps	233
C1	„Caps“ in meQuerySubdeviceCaps()	233
C2	„Caps“ in meQuerySubdeviceCapsArgs()	236

D	Properties.....	237
E	Fehlercodes	238
F	Index.....	239

1 Einführung

Sehr geehrte Kundin, sehr geehrter Kunde,

Mit dem Meilhaus Intelligent Driver System (ME-iDS) können Sie die Programmierung aller unterstützten Meilhaus-Geräte vereinheitlichen und vereinfachen. Es wurde mit dem Ziel entwickelt, dem Programmierer eine geräte- und betriebssystemübergreifend einheitliche Programmierschnittstelle (API) zu bieten. Vereinfacht ausgedrückt beruht das Konzept auf einem „Frage-und-Antwort“-Spiel zwischen Software und Hardware. Die Software kann alle unterstützten Geräte, die sog. Devices, nach deren Komponenten bzw. Funktionsumfang abfragen. Mit dieser Information kann dann im nächsten Schritt auf geeignete funktionelle Einheiten (im Folgenden als „Subdevices“ bezeichnet) zugegriffen werden.

ME-iDS Generation 2.0

Die ME-iDS-API wurde mit Version 2.0 um sog. „Properties“ erweitert. „Properties“ sind charakteristische Eigenschaften eines Gerätes (Device), einer Funktionsgruppe (Subdevice) oder eines Kanals (Channel) usw. die über den sog. „Property-Baum“ ermittelt und gegebenenfalls auch gesetzt werden können (siehe auch Kap. 3.3.2 Properties ab Seite 26). Die neuen Property-Funktionen erlauben den vollen Zugriff auf die Funktionalität Ihres Gerätes. Dies bedeutet für manche Geräte eine deutliche Erweiterung und elegantere Zugriffsmöglichkeit auf die Features der Hardware. Die Konfiguration der Standardfunktionalität Ihrer Geräte - auch der neuen ME-5000-Serie - bleibt davon unberührt und ist auch mit der bekannten Vorgehensweise möglich.

Alle „Properties“ sind auch über das ME-iDC Konfigurations-Tool zugänglich welches ebenfalls erheblich erweitert wurde.

Hinweis:

Die Property-Funktionen stehen ab ME-iDS 2.0 zur Verfügung und sind unter Windows in vollem Umfang für die ME-5000-Serie implementiert. Für die anderen Devices werden augenblicklich nur die allgemeinen Properties unterstützt. In zukünftigen Versionen des ME-iDS werden diese aber weiter ausgebaut und vervollständigt. Die Unterstützung der Property-Funktionen unter Linux ist in Vorbereitung.

Im Kapitel „Programmierung“ finden Sie grundlegende Informationen zur Vorgehensweise bei der Programmierung. Im Kapitel „Funktionsreferenz“ ab Seite 93 sind die Funktionen detailliert beschrieben.

1.1 Unterstützte Hardware

Unterstützte Geräte	Windows	Linux	Bemerkungen
ME-Synapse USB/LAN	✓	✓	
ME-Axon USB/LAN	✓	✓	
ME-94/95/96 cPCI/PCI	✓	✓	Interrupt nicht unterstützt
ME-630 cPCI/PCI/PCle	✓	✓	
ME-1000 Serie cPCI/PCI	✓	✓	
ME-1400 Serie cPCI/PCI	✓	✓	OSC-Ausgang nur unter Linux unterstützt
ME-1600 Serie cPCI/PCI	✓	✓	
ME-4600 Serie cPCI/PCI/PCle	✓	✓	
ME-5001	✓	–	Aufsteckkarte für ME-5000-Serie
ME-5002	✓	–	Aufsteckkarte ME-5810
ME-5004	✓	–	Aufsteckkarte für ME-5000-Serie
ME-5100 cPCI/PCle	✓	–	
ME-5810(/S) cPCI/PCle	✓	–	
ME-6000 Serie cPCI/PCI	✓	✓	
ME-8100 Serie cPCI/PCI	✓	✓	
ME-8200 Serie cPCI/PCI/PCle	✓	✓	
MEphisto-Digi (ME-1400 USB)	✓	–	*
MEphisto-Opto (ME-8200 USB)	✓	–	*
MEphisto-Scope (UM202, UM203)	✓	–	eingeschränkte Funktionalität
MEphisto-Switch (ME-630 USB)	✓	–	*

Tabelle 1: Unterstützte Hardware

*MEphisto-Digi, MEphisto-Opto und MEphisto-Switch werden nicht unter 64 bit Betriebssystemen unterstützt.

1.2 Systemanforderungen

Das Meilhaus Intelligent Driver System unterstützt Windows 2000/XP/Vista und Windows 7 (32 bit und 64 bit* Versionen verfügbar) sowie Linux 2.6 (x86 and x86_64). Das Treibersystem unterstützt Multiprozessor-Systeme.

Hinweis: Linux-Unterstützung für ME-5000-Serie in Vorbereitung. Bitte fragen Sie unseren Vertrieb (sales@meilhaus.com).

*MEphisto-Digi, MEphisto-Opto und MEphisto-Switch werden nicht unter 64 bit Betriebssystemen unterstützt.

1.3 Nomenklatur

Die API-Funktionen der ME-iDS Funktionsbibliothek gelten für die in Tabelle 1 auf Seite 9 gelistete Hardware, sofern Funktionalität vom jeweiligen Device unterstützt wird. Es wird einheitlich das Präfix “*me*” in den Funktionsnamen verwendet. Der Funktionsname besteht aus dem Präfix und mehreren Bestandteilen, die die jeweilige Funktion näher beschreiben und weitgehend „selbstredend“ sind (z. B. “*IO*” für Ein-/Ausgabe-Funktion).

Für die Funktionsbeschreibung gelten folgende Vereinbarungen:

<i>Funktionsnamen</i>	werden im Fließtext kursiv geschrieben z. B. <i>meIOStreamRead()</i> .
<Parameter>	Parameter folgen der ungarischen Notation und werden in spitzen Klammern in der Schriftart Courier geschrieben. p – pointer i – integer f – float d – double c – char (string)
[eckige Klammern]	werden zur Kennzeichnung physikalischer Einheiten verwendet.
main (...)	Programmausschnitte sind in der Schriftart Courier geschrieben.

1.4 Dokumentation

Die 3 Säulen der Dokumentation im Kontext des ME-iDS:

1. Dieses ME-iDS-Handbuch bietet eine weitgehend allgemeine Beschreibung des Konzepts und der Programmierung sowie eine umfassende Funktionsreferenz (Schnellzugriff über ME-iDS-System-Tray oder Windows-Start-Menü).
2. Die ME-iDS-Hilfe (CHM-Dateiformat) wird automatisch mitinstalliert (Schnellzugriff über ME-iDS-System-Tray oder Windows-Start-Menü) und umfaßt folgende Punkte:
 - Hardwarespezifische Aspekte der Programmierung
 - Beispielcode
 - Konstanten-Definition (siehe auch *medefines.h*)
 - Liste der Fehlermeldungen
 - Liste der Properties
 - Häufige Fragen (FAQs)
 - Versionsgeschichte
3. Hardware-Handbücher beschreiben die jeweilige Hardware (Einstellungen, Steckerbelegung, Spezifikationen).

2 Installation

2.1 Installation unter Windows

Hinweise zur Installation unter Windows entnehmen Sie bitte dem Dokument „FirstSteps.pdf“ und den Readme-Dateien im ME-iDS Paket.

Bitte beachten!



Sollten Sie bereits eine ME-iDS Version 1.2.x oder älter auf Ihrem Rechner installiert haben, deinstallieren Sie diese mit dem Programm „meIDSWinRemove.exe“ bevor Sie ME-iDS Version 1.3.0 oder neuer installieren. Sie finden „meIDSWinRemove.exe“ standardmäßig im Pfad „C:\Meilhaus\ME-iDS\install“.

Bitte beachten Sie unbedingt, dass die ME-iDS Treibersoftware vor Einbau der Hardware installiert wird. Dies gilt insbesondere für eine Erstinstallation unter Windows 7. Ansonsten können eine ordnungsgemäße Installation und Betrieb nicht gewährleistet werden.

2.2 Installation unter Linux

Das ME-iDS ist ein Open-Source-Projekt (GPL-Lizenz). Es wird im Source Code verteilt. Der Anwender muss den Code kompilieren und installieren. Details finden Sie in der Datei *installation.html* in der Distribution.

Das ME-iDC erhalten Sie als RPM-Paket, das mit dem Paketmanager der Distribution installiert werden kann.

Hinweis: Linux-Unterstützung für ME-5000-Serie in Vorbereitung. Bitte fragen Sie unseren Vertrieb (sales@meilhaus.com).

2.3 Konfigurationsprogramm (ME-iDC)

Die Verwendung des Meilhaus Intelligent Device Configuration Utility (ME-iDC) erleichtert die Übersicht und bietet komfortable Konfigurationsmöglichkeiten. Die Kommunikation zwischen ME-iDS und ME-iDC erfolgt über die Konfigurationsdatei *meconfig.xml*.

Das Utility-Programm wird unter Windows automatisch mitinstalliert und kann entweder über das ME-System-Tray-Icon im Info-Bereich der Taskleiste oder über den Eintrag „Meilhaus ME-iDS“ im Windows Start-Menü gestartet werden.

Hinweis: Unter Linux muss die Datei *meIDC.py* zunächst kompiliert werden – dies setzt eine Python-Installation voraus. Die ME-5000-Serie wird gegenwärtig nicht unterstützt.

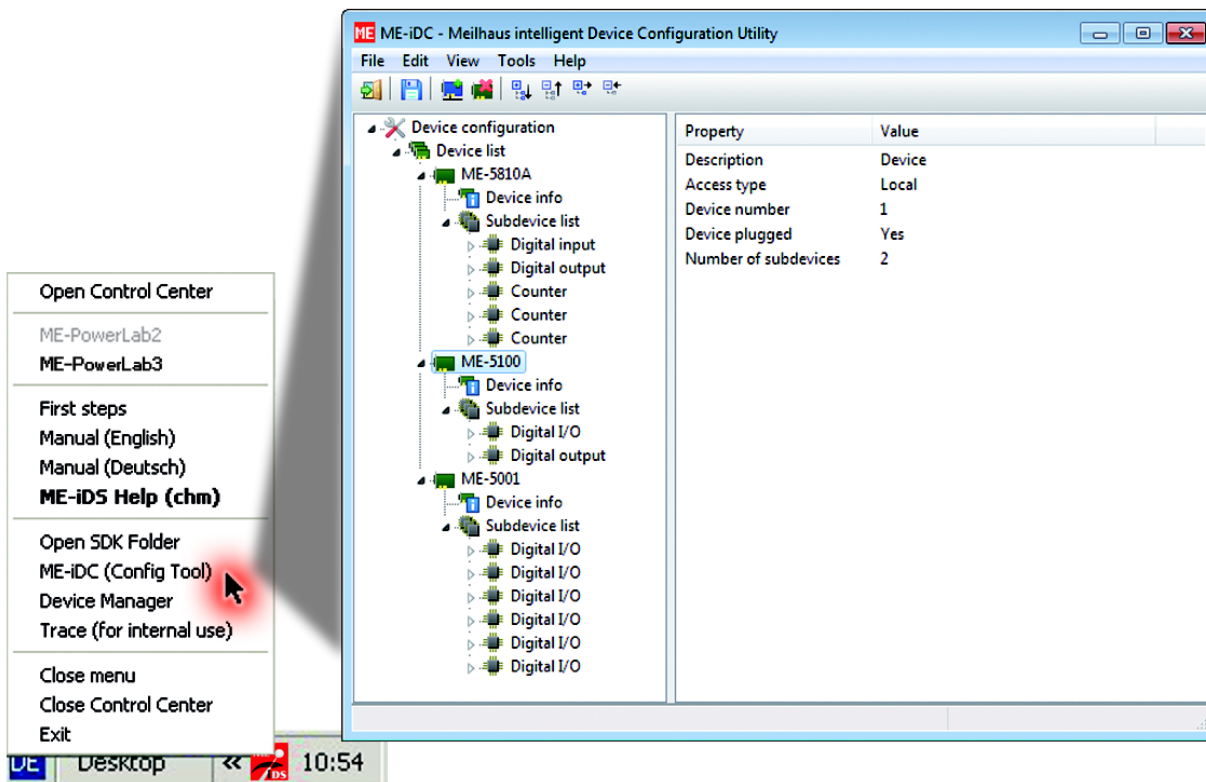


Abbildung 1: ME-iDC

Die wichtigsten Funktionen des ME-iDC:

- Geräte-Informationen können bequem abgefragt werden. Die Struktur der installierten Geräte wird dargestellt.
- Properties können ermittelt und gegebenenfalls gesetzt werden. Die Struktur des Property-Baumes kann dargestellt werden.
- Der „Device“-Index, über den ein Gerät angesprochen wird (Parameter <iDevice>) kann geändert werden.
- Zubehör kann beim Treibersystem angemeldet werden. Die Programmierung wird dadurch vereinfacht.
- „Merkfunktion“ für Geräte, die temporär aus dem System entfernt werden.
- Löschen von Geräten, die aus dem System entfernt wurden.
- Subdevice-Konfiguration ändern
- Laden einer neuen Firmware
- Registrieren von Remote-Geräten wie z. B. ME-Synapse LAN.

Eine ausführliche Beschreibung finden Sie in der Hilfe des ME-iDC.

2.3.1 Subdevice-Konfiguration

Bei manchen Modellen kann die Standard-Funktionalität von Subdevices vom Anwender durch Auswahl einer Alternativ-Konfiguration geändert werden. Die gewünschte Konfiguration wird mit dem Konfigurations-Tool ME-iDC vor Start Ihrer Anwendung aktiviert. Mit der Standardkonfiguration (ID 0) ist das Subdevice sofort betriebsbereit.

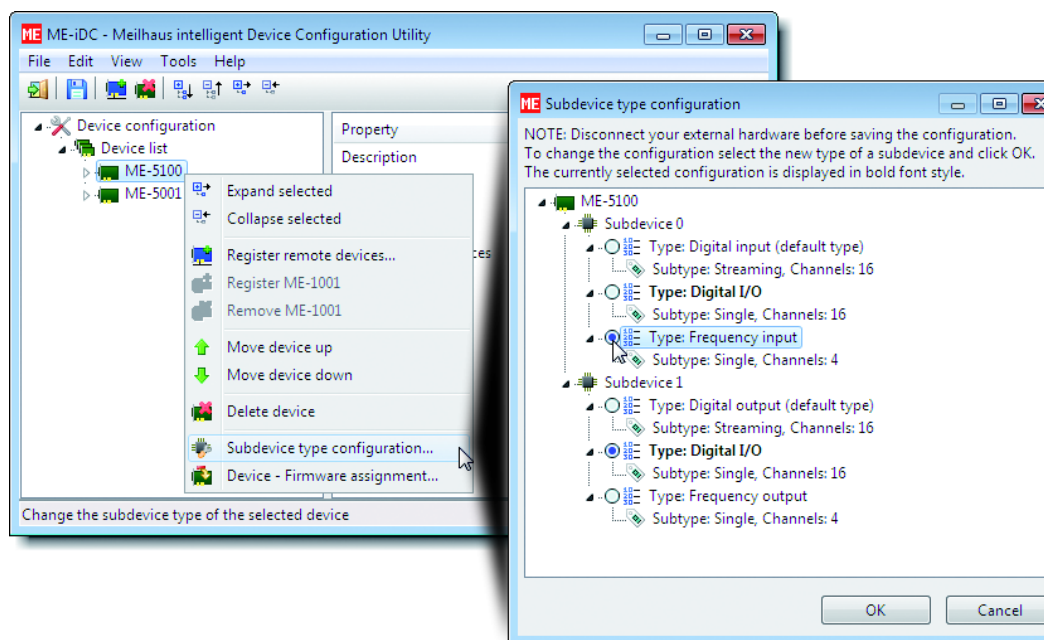


Abbildung 2: Subdevice-Konfiguration

2.3.2 Firmware-Konfiguration

Bei manchen Modellen kann die Standard-Funktionalität des Devices vom Anwender durch Auswahl einer alternativen Firmware geändert werden. Vorgehensweise siehe ME-iDC-Hilfe.

2.3.3 Remote-Gerät anmelden

Der Zugriff auf ein Remote-Device (z. B. ME-Synapse LAN) unter Windows via ME-iDS API setzt die Installation einer sog. ONC RPC-Client-Software für Windows voraus, die den Kauf eines Lizenzschlüssels erfordert. Die Abkürzung ONC RPC steht für „Open Network Computing Remote Procedure Call“.

Zum Einsatz kommt die „ONC RPC/XDR for C/C++ Client Runtime“ Software von Distinct, welche den Sun Microsystems RPC-Standard auf Ihrem Windows-Rechner implementiert. Über sog. „Remote Procedure Calls“ (RPC) kann eine Applikation über ein TCP/IP-basierendes Netz-

werk mit ONC RPC Clients kommunizieren, da für den Datentransfer das „External Data Representation“ (XDR) Format verwendet wird, welches unabhängig von Prozessor und Betriebssystem ist.

Zur Installation der RPC-Software wählen Sie in der ME-iDS-Installationsroutine die Option „Netzwerk Installation“. Es wird ein eigenständiges Installationsprogramm gestartet, in deren Verlauf die Eingabe eines Lizenzschlüssels erforderlich ist.

Beachten Sie, dass für die 32 bit und 64 bit Version der RPC-Software getrennte Lizenzschlüssel erforderlich sind. Sofern Sie keinen Lizenzschlüssel im Rahmen Ihres Produktpaketes erhalten haben, können Sie diesen bei Meilhaus Electronic erwerben. Wenden Sie sich dazu an unsere Vertriebs-Abteilung (sales@meilhaus.com).

Abschließend müssen Sie das Remote-Gerät mit seiner IP-Adresse beim Treibersystem registrieren.

Gehen Sie folgendermaßen vor:

- Installieren Sie das ME-iDS in der Installationsart „Netzwerk Installation“ und geben Sie in dessen Verlauf den Lizenzschlüssel ein.
- Rufen Sie das ME-iDC (Config Tool) via ME-iDS-System-Tray oder über das Windows Start-Menü auf.
- Geben Sie dem Treibersystem die IP-Adresse des Remote-Geräts bekannt (siehe auch Config-Tool-Hilfe) indem Sie auf das blaue Icon „Register remote devices“ klicken.
- Öffnen Sie den Dialog und geben Sie die aktuelle IP-Adresse ein.

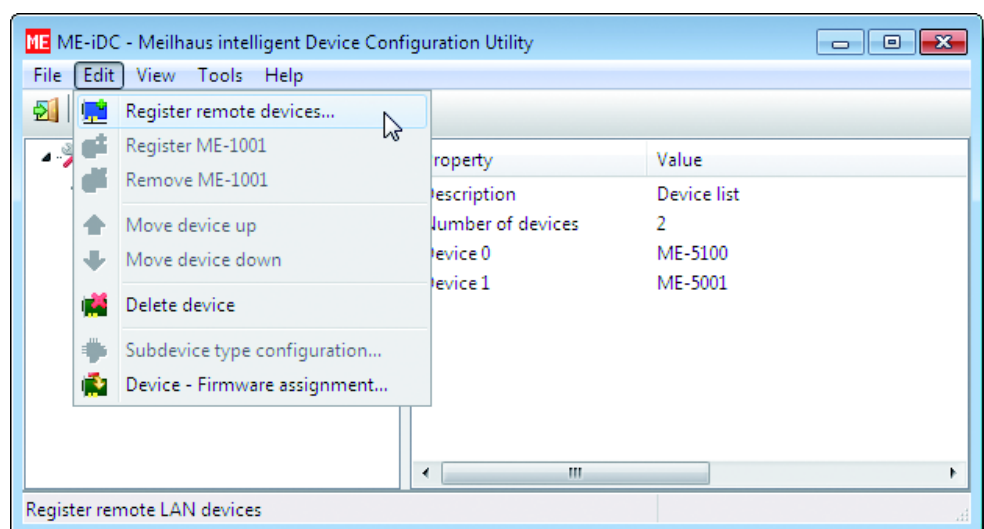


Abbildung 3: IP-Adresse anmelden

2.4 Einstellen der IP-Adresse

Fragen Sie bei Bedarf Ihren Netzwerk-Administrator welche IP-Adresse verwendet werden soll. Um die IP-Adresse der ME-Synapse LAN und ME-Axon-LAN zu ändern, starten Sie einen Web-Browser und geben Sie die werkseitig voreingestellte (oder die von Ihnen selbst zuletzt eingestellte) Adresse ein. Im Fall der voreingestellten Adresse ist dies **http://192.168.20.228** (siehe auch Handbuch des Geräts). Es wird die IP-Konfigurations-Seite angezeigt.

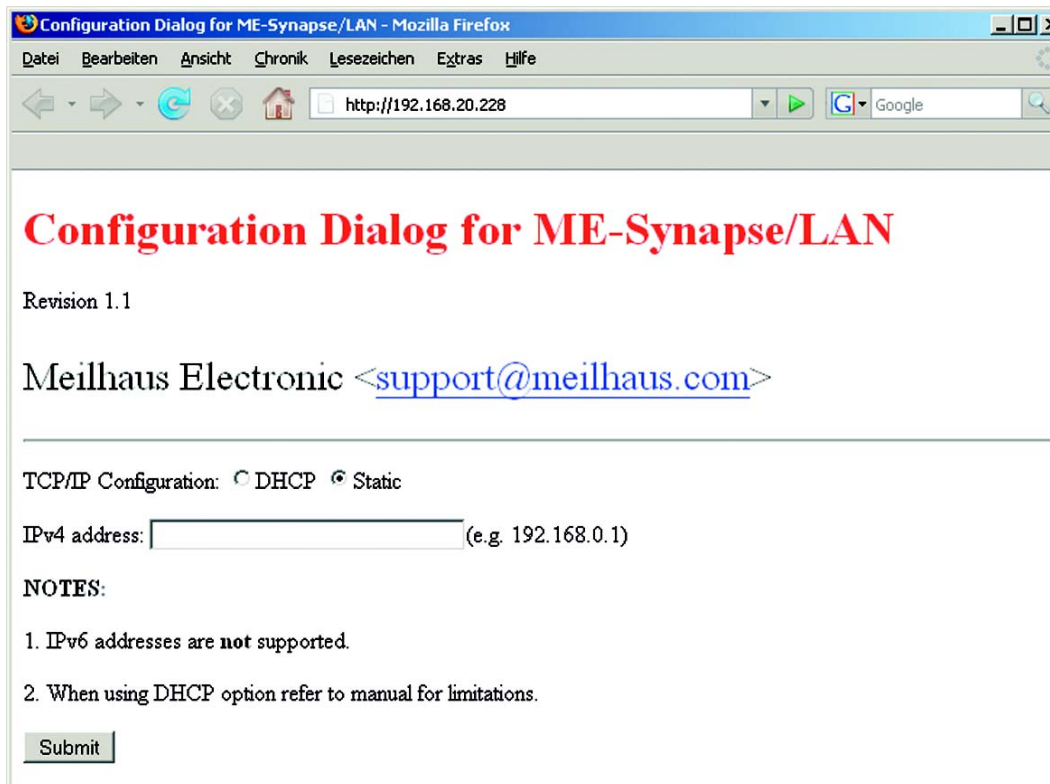


Abbildung 4: Einstellen der IP-Adresse

Statisch

Wählen Sie „Static“ und geben Sie im Feld „IPv4 address“ die gewünschte Adresse ein. Notieren Sie diese Adresse (mit Netzwerk-Kenntnissen können Sie die Adresse später auch mit dem Kommando „ipconfig“ unter Windows „Ausführen...“ wieder ermitteln). Geben Sie dann die IP-Adresse im ME-iDC dem Treibersystem bekannt.

Dynamisch

ME-Synapse LAN und ME-Axon LAN können nur in Netzwerken betrieben werden, in denen die IP-Adressen zwar dynamisch vergeben werden, dann den Geräten jedoch dauerhaft und fest zugewiesen bleiben.

Zur Einstellung wählen Sie „DHCP“. Sprechen Sie mit Ihrem Netzwerk-Administrator, um z. B. mit einem „Ping“-Befehl die zugewiesene Adresse zu ermitteln. Geben Sie dann die Adresse im ME-iDC dem Treibersystem bekannt.

3 Programmierung

3.1 Architektur des Treibersystems

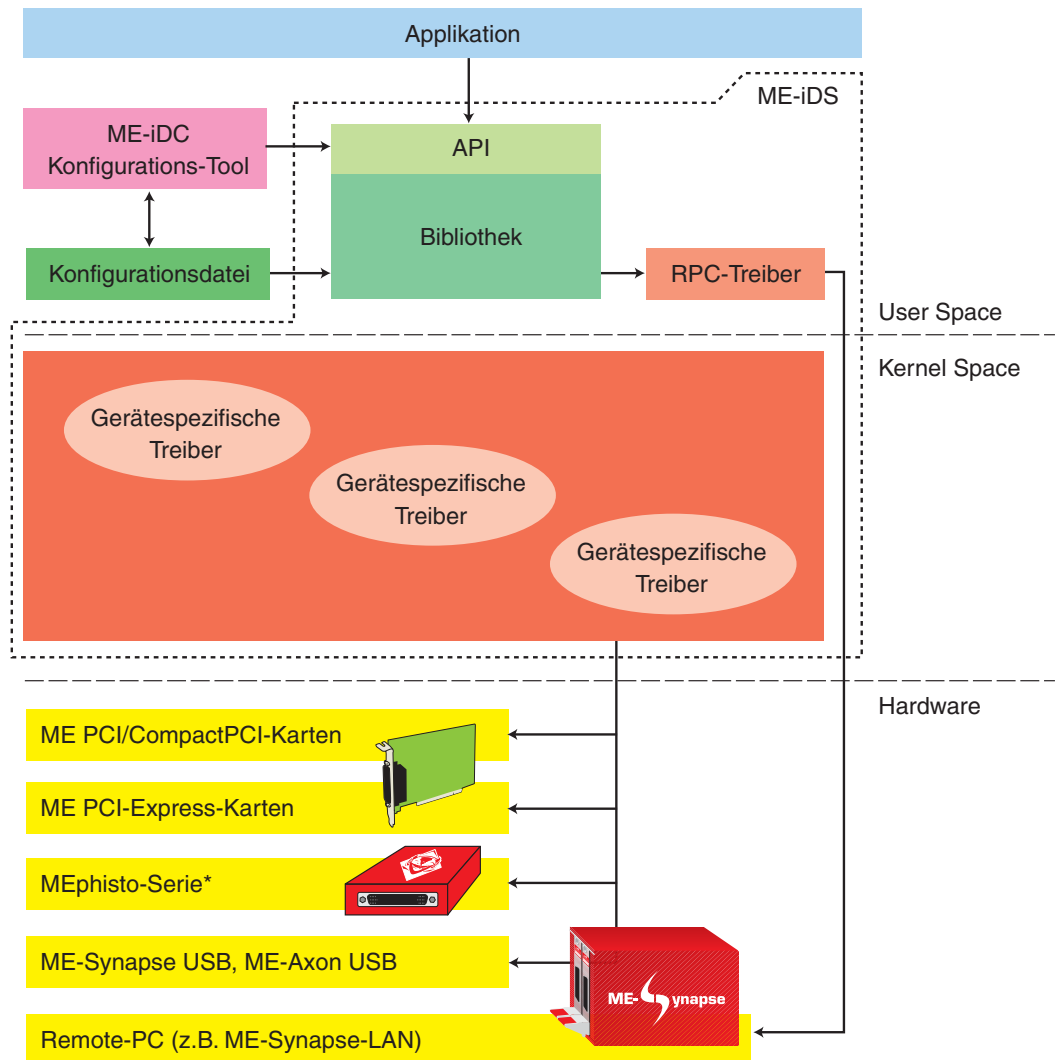


Abbildung 5: Treiber-Struktur

*MEphisto-Digi, MEphisto-Opto und MEphisto-Switch werden nicht unter 64 bit Betriebssystemen unterstützt.

Das Meilhaus Intelligent Driver System (ME-iDS) bietet eine geräte- und betriebssystemübergreifende, einheitliche Programmierschnittstelle (API). Es gliedert sich in die Ebene „User Space“ mit der Funktionsbibliothek, die Ebene „Kernel Space“ mit dem Haupttreiber und gerätespezifischen Treibermodulen und schließlich die Hardwareebene. Für den Fernzugriff über ein Netzwerk benötigen Sie unter Windows eine ONC RPC-Software von Distinct mit Lizenz-Schlüssel (siehe auch

Kap. 2.3.3 auf Seite 16), die nicht standardmäßig im ME-iDS-Paket enthalten ist. Für Linux wird keine extra Lizenz benötigt.

3.1.1 Bibliotheksdateien

Die Bibliotheksdateien gibt es in Versionen ohne (lokal) und mit RPC-Unterstützung für den Fernzugriff via Netzwerk.

Windows

Unter Windows wird standardmäßig die lokale Version installiert. Der Anwender kann die RPC-Unterstützung optional wählen (bitte nur wählen, falls gültige RPC-Lizenz vorhanden ist).

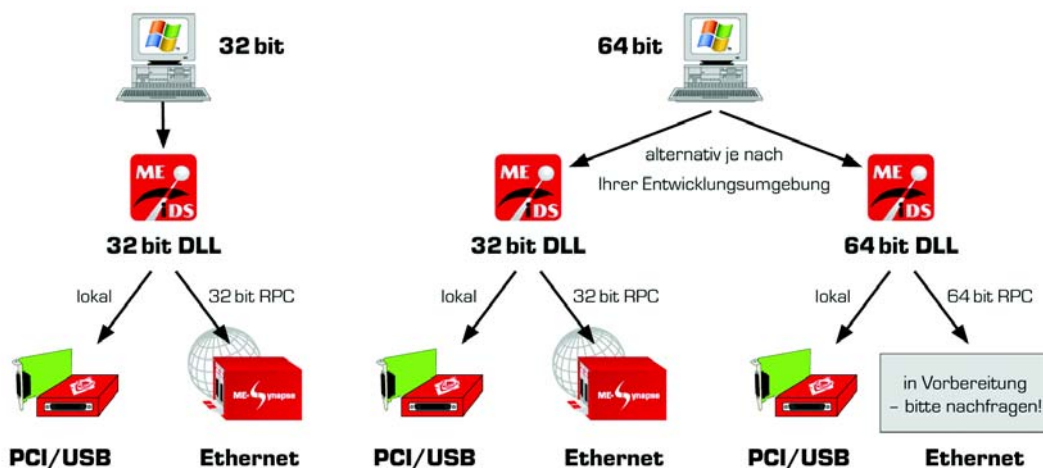


Abbildung 6: Bibliotheksdateien

Linux

In Linux müssen während der Installation die Bibliotheksdateien entsprechend „gemappt“ werden. Die Verlinkung erfolgt automatisch durch die Installationsroutine. Die folgenden Versionen sind im Moment verfügbar:

- Lokal (nur PCI und USB, keine RPC-Unterstützung)
- „Remote“ (nur RPC, kein PCI oder USB)
- Universal (PCI, USB und RPC-Unterstützung)

Hinweis: Die lokale Version ist kleiner und schneller. Die Zeit um die Verbindung zu einem entfernten Gerät herzustellen kann sehr lang sein, besonders dann, wenn das Gerät nicht aktiv ist.

3.2 Sprachen-Unterstützung

Im Folgenden finden Sie eine Übersicht der Programmiersprachen und Entwicklungsumgebungen, die standardmäßig vom ME-iDS unterstützt werden.

Sie finden das ME-iDS auf der mitgelieferten CD/DVD oder unter www.meilhaus.de/download.

3.2.1 Hochsprachen-Unterstützung

Um Ihnen die Hochsprachenprogrammierung zu erleichtern werden einfache Beispiele und kleine Projekte im Source-Code der jeweiligen Sprache mitgeliefert. Die Programmierbeispiele werden als ZIP-Datei mitgeliefert und zum Download angeboten (siehe: www.meilhaus.de/download).

Bitte beachten Sie auch die Hinweise in den entsprechenden README-Dateien.

	Windows	Linux
C/C++	Visual C++, C++ Builder	✓
Visual Basic	Visual Basic 6.0, Visual Basic .NET	–
Visual C#	✓	with MONO
Delphi	✓	–

Tabelle 2: Hochsprachen-Unterstützung

3.2.2 Graphische Programmierwerkzeuge

Zur Programmierung mit graphischen Programmierwerkzeugen wie Agilent VEE oder LabVIEW™ erhalten Sie vordefinierte Objekte („User Objects“ bzw. „Virtual Instruments“) und Demos, die Sie mit der Maus „verdrahten“ können.

	Windows	Linux
Agilent VEE	✓	–
LabVIEW	✓	✓

Tabelle 3: Graphische Programmierwerkzeuge

3.3 Konzept der Bibliothek

Das Konzept des ME-iDS (Meilhaus Intelligent Driver System) kann man als hierarchischen Baum mit einer Wurzel beschreiben. Sämtliche Hardware ist gemäß folgender Hierarchie organisiert.

3.3.1 Hierarchie-Ebenen

- „**Driver**“: Der Treiber repräsentiert das gesamte System als Menge an Geräten. Wurzel des Hierarchiebaumes.
- „**Device**“: Ein Device repräsentiert eine vorhandene Hardware-Einheit mit einem oder mehreren sog. Subdevices. Jedes Device hat seine eigene unveränderbare Seriennummer. Jedem Gerät wird ein Index zugeordnet, mit dem es bei einem Funktionsaufruf durch die ME-iDS API angesprochen wird (Parameter `<iDevice>`).

Wichtig: Es gibt keine strenge Bindung zwischen dem physikalischen Gerät (Hardware) und dem Device-Index, der zur Adressierung verwendet wird. Device-Indizes werden während der Initialisierung dynamisch zugewiesen (beim Aufruf der Funktion *meOpen()*) und können von einem Applikationsstart zum nächsten variieren. Deshalb sollten die *meQuery...()* Funktionen verwendet werden, um den korrekten Device-Index für ein bestimmtes physikalisches Gerät zu ermitteln. Beachten Sie hierzu auch die Beispielpprogramme im ME-iDS Software Developer Kit (SDK).

Hinweis: Informationen, wie man einem physikalischen Gerät einen Device-Index fest zuweisen kann finden Sie in der Hilfe des ME-iDC (Config Tool).

- „**Subdevice**“: Ein sog. Subdevice repräsentiert eine funktionale Einheit (z. B. analoge Erfassung). Jedem Subdevice wird ein Index zugewiesen, welcher in den Funktionen der ME-iDS API übergeben wird (Parameter `<iSubdevice>`) um das jeweilige „Subdevice“ anzusprechen. Subdevice-Indizes beginnen mit '0' und werden statisch zugewiesen. Zwei Geräte des gleichen Typs enthalten die gleichen Subdevices in der gleichen Reihenfolge.

Wichtig: Jedes Modell hat seine eigene Zählreihenfolge der Subdevices. Zwei Modelle der gleichen Familie können eine unterschiedliche Gruppierung der Subdevices aufweisen, so dass ein bestimmtes Subdevice möglicherweise verschiedene Subdevice-Indizes auf zwei unterschiedlichen Modellen aufweist. Zum Beispiel: der externe Interrupt (EXT_IRQ) auf der ME-1400A hat den Subdevice-Index '6' aber auf der ME-1400B den Subdevice-Index '12'. Daher sollten mit Hilfe der *meQuery...()* Funktionen die korrekten Subdevice-Indizes für ein bestimmtes Subdevice ermittelt werden, so wie es auch in den Programmierbeispielen im ME-iDS SDK gezeigt wird.

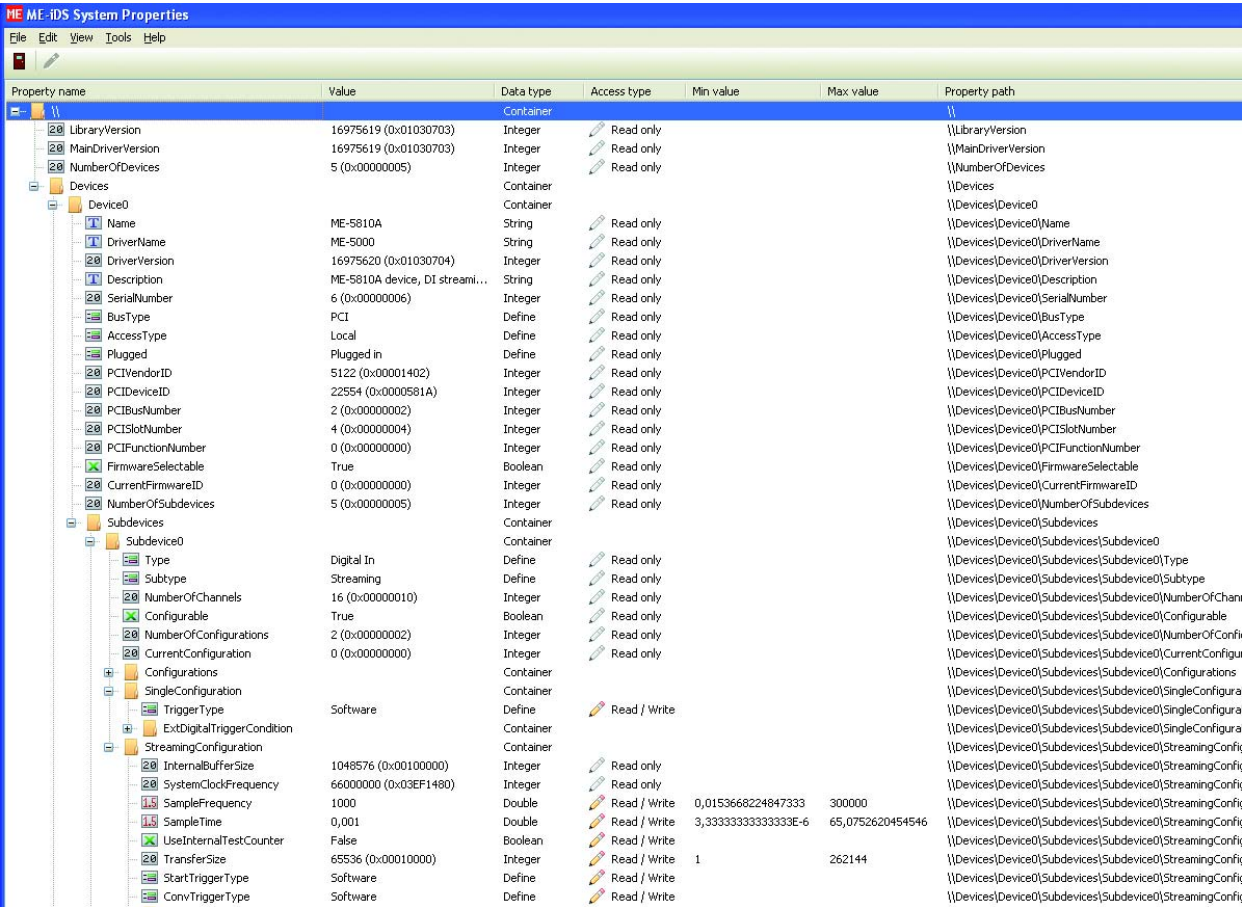
- „**Channel**“: Die unterste Ebene der Hardwarekomponenten repräsentiert einen einzelnen Datenkanal, z. B. einen Eingang oder Ausgang (digital, analog, etc.). Jeder Kanal kann mehrere Parameter haben wie: Bereich, Referenz, Polarität (unipolar, bipolar), etc.

3.3.2 Properties

Die ME-iDS-API wurde mit Version 2.0 um sog. „Properties“ erweitert. „Properties“ sind charakteristische Eigenschaften eines Devices, eines Subdevices oder eines Kanals usw. die in Form einer Baumstruktur organisiert sind. Über den sog. Property-Pfad können Sie auf alle Properties und Attribute zugreifen, deren Wert ermitteln und gegebenenfalls auch setzen (lesen/schreiben). Die Namen der Properties sind vordefiniert.

Es gibt Properties vom Zugriffstyp „Read only“, die nur gelesen werden können, wie z. B. der Typ eines Subdevices (DI, DO, DIO, AI etc.) und solche vom Zugriffstyp „Read/Write“ wie z. B. die Richtung eines bidirektionalen Ports die sowohl abgefragt (gelesen) als auch gesetzt (geschrieben) werden kann.

In der folgenden Abbildung sehen Sie eine typischen Property-Baum wie Sie ihn via ME-iDC darstellen können:



Property name	Value	Data type	Access type	Min value	Max value	Property path
Library/Version	16975619 (0x01030703)	Integer	Read only			\\Library/Version
MainDriverVersion	16975619 (0x01030703)	Integer	Read only			\\MainDriverVersion
NumberOfDevices	5 (0x00000005)	Integer	Read only			\\NumberOfDevices
Devices	Container	Container				\\Devices
Device0	Container	Container				\\Devices\\Device0
Name	ME-5810A	String	Read only			\\Devices\\Device0\\Name
DriverName	ME-5000	String	Read only			\\Devices\\Device0\\DriverName
DriverVersion	16975620 (0x01030704)	Integer	Read only			\\Devices\\Device0\\DriverVersion
Description	ME-5810A device, DI streami...	String	Read only			\\Devices\\Device0\\Description
SerialNumber	6 (0x00000006)	Integer	Read only			\\Devices\\Device0\\SerialNumber
BusType	PCI	Define	Read only			\\Devices\\Device0\\BusType
AccessType	Local	Define	Read only			\\Devices\\Device0\\AccessType
Plugged	Plugged in	Define	Read only			\\Devices\\Device0\\Plugged
PCIVendorID	5122 (0x00001402)	Integer	Read only			\\Devices\\Device0\\PCIVendorID
PCIDeviceID	22554 (0x0000581A)	Integer	Read only			\\Devices\\Device0\\PCIDeviceID
PCIBusNumber	2 (0x00000002)	Integer	Read only			\\Devices\\Device0\\PCIBusNumber
PCISlotNumber	4 (0x00000004)	Integer	Read only			\\Devices\\Device0\\PCISlotNumber
PCIFunctionNumber	0 (0x00000000)	Integer	Read only			\\Devices\\Device0\\PCIFunctionNumber
FirmwareSelectable	True	Boolean	Read only			\\Devices\\Device0\\FirmwareSelectable
CurrentFirmwareID	0 (0x00000000)	Integer	Read only			\\Devices\\Device0\\CurrentFirmwareID
NumberOfSubdevices	5 (0x00000005)	Integer	Read only			\\Devices\\Device0\\NumberOfSubdevices
Subdevices	Container	Container				\\Devices\\Device0\\Subdevices
Subdevice0	Container	Container				\\Devices\\Device0\\Subdevices\\Subdevice0
Type	Digital In	Define	Read only			\\Devices\\Device0\\Subdevices\\Subdevice0\\Type
Subtype	Streaming	Define	Read only			\\Devices\\Device0\\Subdevices\\Subdevice0\\Subtype
NumberOfChannels	16 (0x00000010)	Integer	Read only			\\Devices\\Device0\\Subdevices\\Subdevice0\\NumberOfChani
Configurable	True	Boolean	Read only			\\Devices\\Device0\\Subdevices\\Subdevice0\\Configurable
NumberOfConfigurations	2 (0x00000002)	Integer	Read only			\\Devices\\Device0\\Subdevices\\Subdevice0\\NumberOfConfig
CurrentConfiguration	0 (0x00000000)	Integer	Read only			\\Devices\\Device0\\Subdevices\\Subdevice0\\CurrentConfigu
Configurations	Container	Container				\\Devices\\Device0\\Subdevices\\Subdevice0\\Configurations
SingleConfiguration	Container	Container				\\Devices\\Device0\\Subdevices\\Subdevice0\\SingleConfigura
TriggerType	Software	Define	Read / Write			\\Devices\\Device0\\Subdevices\\Subdevice0\\SingleConfigura
ExtDigitalTriggerCondition	Container	Container				\\Devices\\Device0\\Subdevices\\Subdevice0\\SingleConfigura
StreamingConfiguration	Container	Container				\\Devices\\Device0\\Subdevices\\Subdevice0\\StreamingConfir
InternalBufferSize	1048576 (0x00100000)	Integer	Read only			\\Devices\\Device0\\Subdevices\\Subdevice0\\StreamingConfir
SystemClockFrequency	66000000 (0x03EF1480)	Integer	Read only			\\Devices\\Device0\\Subdevices\\Subdevice0\\StreamingConfir
SampleFrequency	1000	Double	Read / Write	0,015368224847333	300000	\\Devices\\Device0\\Subdevices\\Subdevice0\\StreamingConfir
SampleTime	0,001	Double	Read / Write	3,33333333333333E-6	65,0752620454546	\\Devices\\Device0\\Subdevices\\Subdevice0\\StreamingConfir
UseInternalTestCounter	False	Boolean	Read / Write			\\Devices\\Device0\\Subdevices\\Subdevice0\\StreamingConfir
TransferSize	65536 (0x00100000)	Integer	Read / Write	1	262144	\\Devices\\Device0\\Subdevices\\Subdevice0\\StreamingConfir
StartTriggerType	Software	Define	Read / Write			\\Devices\\Device0\\Subdevices\\Subdevice0\\StreamingConfir
ConvTriggerType	Software	Define	Read / Write			\\Devices\\Device0\\Subdevices\\Subdevice0\\StreamingConfir

Abbildung 7: Property-Baum

Hinweis:

Die Property-Funktionen stehen ab ME-iDS 2.0 zur Verfügung und sind unter Windows in vollem Umfang für die ME-5000-Serie implementiert. Für die anderen Devices werden augenblicklich nur die allgemeinen Properties unterstützt. In zukünftigen Versionen des ME-iDS werden diese aber weiter ausgebaut und vervollständigt. Die Unterstützung der Property-Funktionen unter Linux ist in Vorbereitung.

3.3.2.1 Property-Pfade

Properties werden grundsätzlich in sog. „Containern“ zusammengefasst. Ein Container enthält Properties und/oder weitere Container. Auf diese Weise werden alle Properties des Systems hierarchisch, einer Baumstruktur gleich, organisiert.

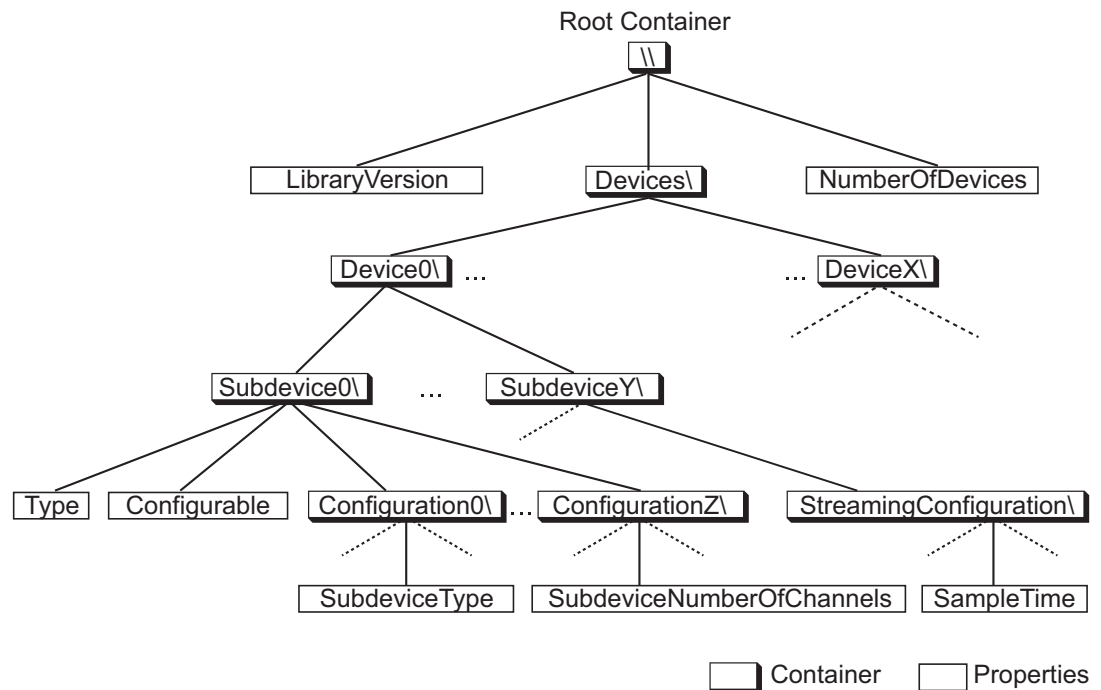


Abbildung 8: Baumstruktur

Die Wurzel (Root) des Property-Baumes ist immer ein Container, der durch zwei aufeinanderfolgende Backslashes "\\" beschrieben wird. Die einzelnen Elemente eines Property-Pfades werden durch einen Backslash "\" getrennt.

Um auf eine Eigenschaft zuzugreifen muss der gesamte Property-Pfad von der Wurzel "\\" über alle betroffenen Container, getrennt durch einen Backslash, bis zum Namen der Eigenschaft angegeben werden.

Um z. B. auf den Subdevice-Typ von Subdevice 0 von Device 0 zuzugreifen lautet der Pfad:

```
"\\Devices\\Device0\\Subdevices\\Subdevice0\\Type"
```

Am Ende des Pfades darf **kein** „\\“ angehängt werden.

Hinweis: Je nach Programmiersprache wird der Backslash als „Escape“-Zeichen verwendet. In diesem Fall müssen im Quellcode (z. B. C, C++, C#) zwei Backslashes verwendet werden um einen Backslash in einem Property-String darzustellen. Zum Beispiel:

```
"\\\\Devices\\\\Device1"
```

3.3.2.2 Abkürzungen für Property-Pfade

Zur Vereinfachung wurde eine verkürzte Schreibweise für den Property-Pfad eingeführt. Folgende Properties bzw. Container können abgekürzt werden:

- **Devices:**
 "\\Devices\\Device1" kann abgekürzt werden:
 "\\Device1"
- **Subdevices:**
 "\\Device1\\Subdevices\\Subdevice0" kann abgekürzt werden:
 "\\Device1\\Subdevice0"
- **Channels:**
 "\\Device1\\Subdevice0\\Channels\\Channel7" kann abgekürzt werden:
 "\\Device1\\Subdevice0\\Channel7"
- **Configurations:**
 "\\Device1\\Subdevice0\\Configurations\\Configuration0" kann abgekürzt werden:
 "\\Device1\\Subdevice0\\Configuration0"

3.3.2.3 Property-Funktionen

Die Property-Funktionen sind als ANSI- und Unicode (UTF-16)-Version implementiert. Die ANSI-Version (mit Suffix „A“) verwendet NULL-terminierte ANSI-Strings (char*). Die Unicode-Version (mit Suffix „W“) nutzt „Wide Character“ Strings (wchar_t*).

In Abhängigkeit vom Property-Typ (siehe Kap. 3.3.2.5 auf Seite 32) muss die entsprechende Funktion zum Lesen bzw. Schreiben des Wertes verwendet werden (siehe auch Funktionsreferenz ab Seite 122).

Hinweis: Im Folgenden werden stellvertretend die ANSI-Funktionen verwendet. Alle Beispiele gelten jedoch analog auch für Unicode-Funktionen.

3.3.2.3.1 Lesen von Property-Werten

Folgende Funktionen stehen für **lesenden Zugriff** zur Verfügung:

ANSI:

int mePropertyGetIntA(char pcPropertyPath, int* piValue)*

Liest eine Eigenschaft als vorzeichenbehafteten 32 bit Integer-Wert.

int mePropertyGetDoubleA(char pcPropertyPath, double* pdValue)*

Liest eine Eigenschaft im Fließkomma-Format als 64 bit Double-Wert.

int mePropertyGetStringA(char pcPropertyPath, char* pcValue, int iBufferLength)*

Liest eine String-Eigenschaft als NULL-terminiertes ANSI-Zeichen-Array.

Unicode:

int mePropertyGetIntW(wchar_t pcPropertyPath, int* piValue)*

Liest eine Eigenschaft als vorzeichenbehafteten 32 bit Integer-Wert.

int mePropertyGetDoubleW(wchar_t pcPropertyPath, double* pdValue)*

Liest eine Eigenschaft im Fließkomma-Format als 64 bit Double-Wert.

int mePropertyGetStringW(wchar_t pcPropertyPath, wchar_t* pcValue, int iBufferLength)*

Liest eine String-Eigenschaft als NULL-terminiertes Unicode-Zeichen-Array.

3.3.2.3.2 Schreiben von Property-Werten

Folgende Funktionen stehen für **schreibenden Zugriff** zur Verfügung:

ANSI:

int mePropertySetIntA(char pcPropertyPath, int iValue)*

Setzt eine Eigenschaft als vorzeichenbehafteten 32 bit Integer-Wert.

int mePropertySetDoubleA(char pcPropertyPath, double dValue)*

Setzt eine Eigenschaft im Fließkomma-Format als 64 bit Double-Wert.

int mePropertySetStringA(char pcPropertyPath, char* pcValue)*

Setzt eine String-Eigenschaft als NULL-terminiertes ANSI-Zeichen-Array.

Unicode:

int mePropertySetIntW(wchar_t pcPropertyPath, int iValue)*

Setzt eine Eigenschaft als vorzeichenbehafteten 32 bit Integer-Wert.

int mePropertySetDoubleW(wchar_t pcPropertyPath, double dValue)*

Setzt eine Eigenschaft im Fließkomma-Format als 64 bit Double-Wert.

int mePropertySetStringW(wchar_t pcPropertyPath, wchar_t* pcValue)*

Setzt eine String-Eigenschaft als NULL-terminiertes Unicode-Zeichen-Array.

3.3.2.4

Attribute

Properties und Container haben sog. „Attribute“ auf welche ebenfalls mit den Property-Funktionen zugegriffen werden kann. Fügen Sie einfach einen weiteren Backslash „\“ an den Property-Pfad an, gefolgt von dem Attributnamen. Attribute liefern zusätzliche Informationen über Properties und/oder Container, z. B. den Datentyp der Property, die Zugriffsart (nur lesend oder auch schreibend), die Anzahl der Elemente in einem Container etc.

Beispiele für Attribute:

Name	Definiert für...	Typ	Mögliche Werte
NumberOfElements	Container	Integer	-
PropertyName	Properties & Container	String	-
PropertyType	Properties & Container	Define	ME_PROPERTY_TYPE_CONTAINER, ME_PROPERTY_TYPE_BOOL, ME_PROPERTY_TYPE_INT, ME_PROPERTY_TYPE_DOUBLE, ME_PROPERTY_TYPE_STRING, ME_PROPERTY_TYPE_DEFINE
PropertyAccess	Properties	Define	ME_PROPERTY_ACCESS_READ_ONLY, ME_PROPERTY_ACCESS_READ_WRITE

Tabelle 4: Attribute (beispielhaft)

Sie finden alle Attribute mit einer kurzen Beschreibung in der ME-iDS Hilfe-Datei (*.chm) unter „Related Pages“ - „Properties“.

Das Attribut „NumberOfElements“ ist z. B. für jeden Container definiert (aber nicht für Properties). Der Wert ist vom Typ Integer und kann mit der Funktion *mePropertyGetIntA()* gelesen werden. Um herauszufinden wieviele Elemente (Container und Properties) sich im

Root-Container befinden können Sie folgenden Funktionsaufruf verwenden (<piValue> gibt die Anzahl der Elemente zurück):

```
mePropertyGetIntA("\\NumberOfElements", int* piValue);
```

Sobald Sie wissen wieviele Elemente ein Container enthält, können Sie die Elemente über den Index ansprechen ohne zu wissen, wie die Elemente heißen. Wenn Sie z. B. festgestellt haben, dass der Root-Container 7 Elemente enthält, können Sie diese mit "\\0", "\\1", ... "\\6" ansprechen.

Mit dem Attribut „PropertyName“ können Sie den Namen aller Elemente im Container abfragen. Um z. B. den Namen des ersten Elements (mit Index 0) im Root-Container herauszufinden können Sie folgenden Funktionsaufruf nutzen:

```
mePropertyGetStringA("\\0\\PropertyName, char* pcValue, int iBufferLength);
```

Der Eigenschaftsname wird über den Parameter <pcValue> zurückgegeben.

Sobald Sie den Namen der Eigenschaft kennen, können Sie zur Adressierung den Namen anstatt dem Index verwenden. Angenommen das erste Element im Root-Container sei „LibraryVersion“, dann können Sie den Typ dieser Eigenschaft folgendermaßen ermitteln:

```
mePropertyGetIntA("\\0\\PropertyType", int* piValue); oder...
```

```
mePropertyGetIntA("\\LibraryVersion\\PropertyType", int* piValue);
```

Das Ergebnis ist das Gleiche.

Auf diesem Weg können Sie Name und Typ aller Elemente im Root-Container ermitteln. Einige dieser Elemente werden selbst Container sein. Zur Ermittlung von Property-Namen und -Typ können Sie die selbe Vorgehensweise wählen.

Der Root-Container enthält stets einen Container namens „Devices“. Um herauszufinden wieviele Elemente in diesem Container sind, können Sie folgenden Funktionsaufruf verwenden:

```
mePropertyGetIntA("\\Devices\\NumberOfElements", int* piValue);
```

Im nächsten Schritt können Sie den Namen des ersten Elements (mit Index 0) ermitteln usw.

```
mePropertyGetStringA("\\Devices\\0\\PropertyName, char* pcValue,  
int iBufferLength);
```

Auf diese Weise können Sie rekursiv den gesamten Eigenschaftsbaum abarbeiten. Das Beispielprogramm `Con_meIDSSystemProperties` im ME-iDS SDK zeigt, wie ein einfaches Konsolenprogramm in „C“-Code aussieht.

3.3.2.5 Property-Typen

Das Attribut „PropertyType“ ist für alle Properties und Container definiert. Angenommen „P“ sei der Pfad zu einer Property oder einem Container gilt stets folgender Funktionsaufruf:

```
mePropertyGetIntA("P\PropertyType", int* piValue)
```

Der Typ der Property wird über den Parameter `<piValue>` zurückgegeben und ist von einem der folgenden, vordefinierten Typen:

- **ME_PROPERTY_TYPE_CONTAINER:**
"P" ist ein Container welcher selbst weitere Properties und Container enthält.
- **ME_PROPERTY_TYPE_BOOL:**
"P" ist eine zweiwertige Eigenschaft z. B. Aus/Ein, False/True, Nein/Ja etc. Der Wert kann mit der Funktion *mePropertyGetIntA()* abgefragt und in manchen Fällen mit der Funktion *mePropertySetIntA()* gesetzt werden. Er kann 0 für FALSE oder 1 für TRUE sein und kann auch mit der Funktion *mePropertyGetStringA()* abgefragt werden. Es wird eine „lesbare“ Version des Wertes zurückgegeben, z. B. für Anzeigezwecke.
- **ME_PROPERTY_TYPE_INT:**
"P" ist eine Eigenschaft vom Typ „Integer“. Der vorzeichenbehaftete 32 bit Integer-Wert kann mit der Funktion *mePropertyGetIntA()* abgefragt und in manchen Fällen mit der Funktion *mePropertySetIntA()* gesetzt werden. Die Eigenschaft kann auch mit der Funktion *mePropertyGetStringA()* abgefragt werden. Es wird eine „lesbare“ Version des Wertes zurückgegeben, z. B. für Anzeigezwecke.
- **ME_PROPERTY_TYPE_DOUBLE:**
"P" ist eine Eigenschaft vom Typ „Double“. Der 64 bit Fließkomma-Wert kann mit der Funktion *mePropertyGetDoubleA()* abgefragt und in manchen Fällen mit der Funktion *mePropertySetDoubleA()* gesetzt werden. Der Wert kann auch mit der Funktion *mePropertyGetStringA()* abgefragt werden. Es wird eine „lesbare“ Version des Wertes zurückgegeben, z. B. für Anzeigezwecke.
- **ME_PROPERTY_TYPE_STRING:**
"P" ist eine Eigenschaft vom Typ „String“ in Form einer Zeichenkette, die mit NULL abgeschlossen ist. Der Wert kann mit der Funktion

mePropertyGetStringA() abgefragt und in manchen Fällen mit der Funktion *mePropertySetStringA()* gesetzt werden.

- **ME_PROPERTY_TYPE_DEFINE:**
"P" ist eine Eigenschaft vom Typ "Define". Diese kann den Wert vordefinierter Konstanten annehmen. Der Wert kann mit der Funktion *mePropertyGetIntA()* abgefragt und in manchen Fällen mit der Funktion *mePropertySetIntA()* gesetzt werden. Die Eigenschaft kann auch mit der Funktion *mePropertyGetStringA()* abgefragt werden. Es wird eine „lesbare“ Version der Konstante zurückgegeben, z. B. für Anzeigezwecke.

Der Wert einer Eigenschaft kann mittels Property-Pfad ohne Attribut in der entsprechenden *mePropertyGet...* Funktion (siehe oben) verwendet werden. Um zum Beispiel den Wert der Eigenschaft „LibraryVersion“ in obigem Root-Container abzufragen können Sie entweder die Funktion

```
mePropertyGetIntA("\\LibraryVersion", int* piValue)
```

...aufrufen, welche die Bibliotheksversion als Integer im Parameter <piValue> zurückgibt oder die Funktion:

```
mePropertyGetStringA("\\LibraryVersion", char* pcValue, int iBufferLength)
```

...aufrufen, um den Wert im Parameter <pcValue> als String zu erhalten.

3.3.2.6 Zugriffstyp von Properties

Das Attribut "PropertyAccess" gibt an ob der Zugriffstyp einer Property vom Typ „Read only“ oder vom Typ „Read/Write“ ist. Der Zugriffstyp ist für alle Properties fest definiert, nicht jedoch für Container.

Beispiel für Pfad der Property „ClockSource“. Der Device-Index sei 1, das Subdevice sei vom Typ Zähler mit Index 2. Der Property-Pfad „P“ lautet dann folgendermaßen:

```
"\\Devices\\Device1\\Subdevices\\Subdevice2\\ClockSource"
```

Es gilt folgender Funktionsaufruf:

```
mePropertyGetIntA("P\\PropertyAccess", int* piValue)
```

Im Parameter <piValue> wird eine der beiden folgenden Konstanten zurückgegeben:

- **ME_PROPERTY_ACCESS_READ_ONLY:**
Der Wert der Property „P“ kann nur gelesen werden.
- **ME_PROPERTY_ACCESS_READ_WRITE:**
Der Wert der Eigenschaft „P“ kann gelesen und geschrieben werden.

3.3.2.7 Systemattribute

Die folgenden Properties sind Elemente des Root-Containers "\\\" und können über den Pfad "\\<Property name>" angesprochen werden.

Name	Typ	Zugriffstyp
LibraryVersion	Integer	Read only
MainDriverVersion	Integer	Read only
NumberOfDevices	Integer	Read only
Devices	Container	--

Tabelle 5: Systemattribute

3.3.2.8 Allgemeine Device-Properties

Die folgenden Properties stehen für jedes Device im ME-iDS zur Verfügung. Der Pfad zu einem Device mit dem Index X lautet:

"\\Devices\\X\\<Property name>" oder...

"\\Devices\\DeviceX\\<Property name>"

oder abgekürzt...

"\\DeviceX\\<Property name>"

Beispiel: Property "Plugged" für Device mit Index 0

"\\Devices\\Device0\\Plugged" oder... "\\Device0\\Plugged"

Name	Typ	Zugriffstyp	Mögliche Werte
Name	String	Read only	-
DriverName	String	Read only	-
DriverVersion	Integer	Read only	-
Description	String	Read only	-
SerialNumber	Integer	Read only	-
BusType	Define	Read only	ME_BUS_TYPE_PCI, ME_BUS_TYPE_USB, ME_BUS_TYPE_LAN_PCI, ME_BUS_TYPE_LAN_USB
AccessType	Define	Read only	ME_ACCESS_TYPE_LOCAL, ME_ACCESS_TYPE_REMOTE
Plugged	Boolean	Read only	-
NumberOfSubdevices	Integer	Read only	-
Subdevices	Container	--	-

Tabelle 6: Allgemeine Device-Properties

3.3.2.9 Subdevice-Properties

Die folgenden Properties stehen für jedes Subdevice im ME-iDS zur Verfügung. Der Pfad zu einer Subdevice-Eigenschaft von Subdevice Y auf Device X lautet:

"\\Devices\\DeviceX\\Subdevices\\Y\\<Property name>" oder...

"\\Devices\\DeviceX\\Subdevices\\SubdeviceY\\<Property name>"

oder abgekürzt....

"\\DeviceX\\SubdeviceY\\<Property name>"

Name	Typ	Zugriffstyp
Type	Define	Read only
Subtype	Define	Read only
NumberOfChannels	Integer	Read only
Configurable	Boolean	Read only

Tabelle 7: Allgemeine Subdevice-Properties

Falls die boolsche Eigenschaft „Configurable“ gesetzt ist, gibt es folgende zusätzliche Subdevice-Properties:

Name	Type	Zugriffstyp
NumberOfConfigurations	Integer	Read only
CurrentConfiguration	Integer	Read only
Configurations	Container	

Tabelle 8: Properties für konfigurierbare Subdevices

3.3.2.10 Properties von Konfigurations-Containern

Der Container „Configurations“ enthält „NumberOfConfigurations“ Elemente, welche die verfügbaren Subdevice-Konfigurationen beschreiben:

Der Pfad zu einer Eigenschaft von Konfiguration Z des Subdevice Y auf Device X lautet:

"\\Devices\\DeviceX\\Subdevices\\SubdeviceY\\Configurations\\Z\\<Property name>" oder...

"\\DeviceX\\SubdeviceY\\Configurations\\ConfigurationZ\\<Property name>"

oder abgekürzt....

"\\DeviceZ\\SubdeviceY\\ConfigurationZ\\<Property name>"

Name	Typ	Zugriffstyp
SubdeviceType	Define	Read only
SubdeviceSubtype	Define	Read only
NumberOfChannels	Integer	Read only

Tabelle 9: Properties von Konfigurations-Containern

3.3.3 Subdevices

Das ME-iDS kennt folgende Subdevices:

- Analoge Eingabe (ME_TYPE_AI)
- Analoge Ausgabe (ME_TYPE_AO)
- Digitale Ein-/Ausgabe (ME_TYPE_DIO)
- Digitale Eingabe (ME_TYPE_DI)
- Digitale Ausgabe (ME_TYPE_DO)
- Frequenz-Ein-/Ausgabe (ME_TYPE_FIO)
- Frequenz-Eingabe (ME_TYPE_FI)
- Frequenz-Ausgabe (ME_TYPE_FO)
- Zähler vom Typ 8254 (ME_TYPE_CTR)
- Externer Interrupt (ME_TYPE_EXT_IRQ)
- FPGA (ME_TYPE_FPGA) - geplant

3.3.3.1 Analoge Ein-/Ausgabe

Subdevice-Typen für analoge Ports:

- Analoge Eingabe (ME_TYPE_AI)
- Analoge Ausgabe (ME_TYPE_AO)

meQueryNumberChannels() gibt die Anzahl verfügbarer Ein- oder Ausgangskanäle zurück.

3.3.3.2 Digitale Ein-/Ausgabe

Subdevice-Typen für digitale Ports:

- Digitale Eingabe (ME_TYPE_DI)
- Digitale Ausgabe (ME_TYPE_DO)
- Digitale Ein-/Ausgabe (ME_TYPE_DIO)

meQueryNumberChannels() gibt die Größe (Breite) in Bits eines digitalen Subdevices zurück.

3.3.3.3 Frequenz-Ein-/Ausgabe

Subdevice-Typen für Frequenz-Ein-/Ausgabe:

- Frequenz-Eingabe (ME_TYPE_FI)
- Frequenz-Ausgabe (ME_TYPE_FO)
- Frequenz-Ein-/Ausgabe (ME_TYPE_FIO)

Hinweis: Obige Subdevices werden nur in der Betriebsart Single unterstützt.

3.3.3.4 Zähler

Subdevice-Typ für Zähler vom Typ 8254:

- Zähler vom Typ 8254 (ME_TYPE_CTR)

Zähler-Subdevices haben stets einen Kanal (mit Index 0). Der korrespondierende Parameter der ME-iDS API-Funktionen (<i>iChannel</i>)> sollte daher immer auf '0' gesetzt werden. Die Abfrage mit der Funktion *meQueryNumberChannels()* gibt immer '1' zurück.

Triggeroptionen sind nicht möglich.

3.3.3.5 Interrupt

Subdevice-Typ für externe Interrupt-Eingänge:

- Externer Interrupt (ME_TYPE_EXT_IRQ)

meQueryNumberChannels() gibt die Anzahl der Interruptkanäle zurück.

Ein Interrupt-Subdevice ist stets vom Untertyp ME_SUBTYPE_SINGLE. Das Interrupt-Handling erfolgt mit folgenden Funktionen:

- *meIOIrqStart()*: Konfiguration und Start eines Interrupt-Subdevice.
- *meIOIrqStop()*: Interrupt-Subdevice stoppen.
- *meIOIrqWait()*: „Event-Listener“ wartet auf Interrupt.
- *meIOIrqSetCallback()*: Mit dieser Funktion können Sie eine benutzerdefinierte Callback-Funktion installieren, die im Hintergrund auf einen Interrupt wartet.

Hinweis: Der aktuelle Zustand eines Interrupt-Eingangs kann nicht mit der Funktion *meIOSingle()* gelesen werden.

3.3.3.6 „FPGA“

Geplant - alle Angaben vorläufig! Subdevice-Typ ME_TYPE_FPGA (FPGA = „Free Programmable Gate Array“) ist für Subdevices, deren Funktionalität durch den Anwender mit entsprechendem Fachwissen entworfen werden kann. Voraussetzung ist eine Entwicklungs-Software zum Design der Firmware für den entsprechenden FPGA-Baustein.

3.3.4 Struktur der API

Das ME-iDS-Konzept kann als „Frage-und-Antwort“-Spiel zur Abfrage der Einstellungen beschrieben werden. Die Software kann bzw. muss Ihr System nach verfügbaren Devices, deren Subdevices und Eigenschaften abfragen. Mit dieser Information wird im nächsten Schritt die Hardware konfiguriert damit schließlich darauf zugegriffen werden kann.

Die gesamte ME-iDS API kann in vier Gruppen unterteilt werden:

- Query-Funktionen
- Property-Funktionen
- Ein-/Ausgabe-Funktionen
- Hilfs-Funktionen

3.3.4.1 Query-Funktionen

Durch Verwendung der sog. „Query“-Funktionen können die Fähigkeiten der aktuellen ME-iDS-Konfiguration abgefragt werden.

- System abfragen
- Device abfragen
- Subdevice abfragen
- Bereich abfragen

3.3.4.2 Property-Funktionen

Mit den sog. „Property“-Funktionen haben Sie die Möglichkeit, alle allgemeinen und hardwarespezifischen Properties und Attribute zu ermitteln und gegebenenfalls auch zu setzen. Die Gesamtheit der Properties gleicht einer Baumstruktur. Über den sog. Property-Pfad können Sie damit auf ein Device, ein Subdevice, einen Kanal (Channel) oder einen Messbereich (Range) usw. zugreifen. Siehe auch Kap. 3.3.2 ab Seite 26.

Hinweis:

Die Property-Funktionen stehen ab ME-iDS 2.0 zur Verfügung und sind unter Windows in vollem Umfang für die ME-5000-Serie implementiert. Für die anderen Devices werden augenblicklich nur die allgemeinen Properties unterstützt. In zukünftigen Versionen des ME-iDS werden diese aber weiter ausgebaut und vervollständigt. Die Unterstützung der Property-Funktionen unter Linux ist in Vorbereitung.

Alle Property-Funktionen sind als ANSI- und Unicode (UTF-16)-Version implementiert. Zur Unterscheidung wird an die Funktionsnamen das Suffix „A“ für ANSI bzw. „W“ für „Wide character“ (Unicode) angehängt.

- Properties lesen
- Properties setzen

3.3.4.3 Ein-/Ausgabe-Funktionen

Vorbereitung und Ausführung von Ein-/Ausgabe-Operationen.

- Reset-Funktionen
- Interrupt-Betrieb
- Single-Betrieb
- Streaming-Betrieb

3.3.4.4 Hilfs-Funktionen

- Treiber-Initialisierung
- Zugriffsschutz
- Fehlerbehandlung
- Utility-Funktionen

3.3.5 Grundsätzliche Vorgehensweise

3.3.5.1 Initialisierung

Vor dem Aufruf von Funktionen, die auf Devices des ME-iDS zugreifen, muss das ME-iDS einmalig initialisiert werden. Die Funktion *meOpen()* initialisiert das gesamte Treibersystem:

- Ermitteln der physikalisch vorhandenen Devices und Abgleich mit den Devices in der Device-Configuration, die mit dem ME-iDC abgespeichert wurde
- Umschalten einer geänderten Subdevice-Konfiguration
- Verbindung zu den Treibermodulen herstellen

Um einen sauberen Programmabschluss zu gewährleisten, sollte am Ende eines Programmes *meClose()* verwendet werden. Diese Funktion schließt das ME-iDS-Treibersystem und gibt alle verwendeten Ressourcen frei.

Eine Applikation kann *meOpen()* und *meClose()* auch mehrfach aufrufen um Informationen über angeschlossene Geräte aufzufrischen. Um eine Neuinitialisierung des ME-iDS-Treibersystems durchzuführen, muss vor einem Aufruf von *meOpen()* zuerst *meClose()* aufgerufen werden.

3.3.5.2 Zugriffsschutz

Um einen gleichzeitigen Zugriff verschiedener Applikationen auf die gleichen Ressourcen zu vermeiden, empfehlen wir die Verwendung der *meLock...()* Funktionen im ME-iDS.

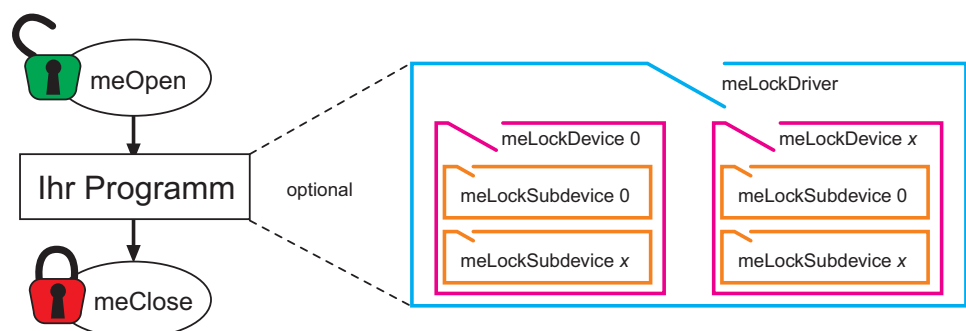


Abbildung 9: Zugriffsschutz-Hierarchie

Es gibt 3 Ebenen um den Zugriff auf Ressourcen zu sperren:

- Driver
 - Device
 - Subdevice

Eine Ressource kann nicht gesperrt werden falls:

... die Ressource selbst oder eine ihrer Unter-Ressourcen bereits durch eine andere Applikation gesperrt sind. Zum Beispiel können Sie ein Gerät nicht sperren, falls eines seiner Subdevices bereits von einer anderen Applikation gesperrt wurde.

... die Ressource selbst oder eine ihrer Unter-Ressourcen sich nicht im Ruhezustand befindet. Zum Beispiel können Sie kein Gerät sperren, falls eines seiner Subdevices auf einen externen Triggerimpuls wartet.

Wenn eine Ressource von einer Applikation gesperrt wurde und eine zweite Applikation versucht mit einer I/O-Funktion auf diese Ressource oder eine seiner Unter-Ressourcen zuzugreifen, gibt die Funktion den Fehlercode `ME_ERRNO_LOCKED` zurück. Wenn anderen Applikationen der Zugriff auf gesperrte Ressourcen wieder gestattet werden soll, muss die Anwendung, die die Ressourcen zuvor gesperrt hat, diese wieder mit den *meLock...()* Funktionen freigeben.

Sperren wirken sich nicht auf *meQuery...()* und *meUtility...()* Funktionen aus. Diese Funktionen können jederzeit durch beliebige Applikationen aufgerufen werden, sofern zuvor *meOpen()* aufgerufen wurde.

Wichtig: Sperren verhindern nur den gleichzeitigen Zugriff verschiedener Applikationen (Prozesse). Wenn Sie den gleichzeitigen Zugriff verschiedener Threads, die im gleichen Prozeß laufen, verhindern wollen, müssen Sie Methoden zur Synchronisation verwenden, wie sie von Ihrem Betriebssystem zur Verfügung gestellt werden (z. B. Mutex, Semaphore, etc.).

Wenn Sie sicher sein können, dass zur gleichen Zeit nur eine einzige Applikation auf eine Ressource zugreift (z. B. wenn Sie nur eine Applikation haben, die das ME-iDS nutzt) brauchen die *meLock...()* Funktionen nicht verwendet werden.

3.3.5.3 Fehlerbehandlung

Alle ME-iDS Funktionen geben den Fehler-Code als Integerwert zurück.

Eine Liste aller Fehler-Codes finden Sie in der ME-iDS Hilfedatei.

Bei Erfolg wird ME_ERRNO_SUCCESS zurückgegeben (entspricht 0). Andernfalls wird ein Fehlercode ungleich ME_ERRNO_SUCCESS zurückgegeben und die interne Variable `<iErrno>` wird entsprechend gesetzt.

Als Erleichterung bietet das ME-iDS die Möglichkeit mit der Funktion *meErrorGetMessage()* Fehlercodes in einen Text als Fehlerbeschreibung umzuwandeln.

Das ME-iDS bietet zwei Funktionen um den zuletzt aufgetretenen Fehler zu ermitteln:

- *meErrorGetLast()* - liefert den letzten Fehlercode zurück
- *meErrorGetLastMessage()* - liefert die Fehlerbeschreibung des letzten Fehlers zurück

Hinweis: Falls eine Funktion versucht einen Zustand herzustellen der bereits zutrifft, gibt die Funktion ME_ERRNO_SUCCESS zurück. Zum Beispiel, falls eine der *meLock...()* Funktionen aufgerufen wird um eine Ressource zu entsperren, die bereits entsperrt ist, so wird ME_ERRNO_SUCCESS zurückgegeben.

Allgemeine Fehler:

- ME_ERRNO_NOT_OPEN: ME-iDS wurde nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: die übergebenen Zeiger sind NULL.
- ME_ERRNO_INVALID_DEVICE: es kann kein Gerät dem angefragten Device-Index zugeordnet werden.
- ME_ERRNO_INVALID_SUBDEVICE: auf dem angesprochenen Gerät kann keinem Subdevice der angefragte Subdevice-Index zugeordnet werden.
- ME_ERRNO_DEVICE_UNPLUGGED: Gerät ist physikalisch gegenwärtig nicht verfügbar.

- **ME_ERRNO_NOT_SUPPORTED:** Funktion wird vom Subdevice nicht unterstützt (z. B.: *meIOStreamConfig()* wird auf ein Subdevice angewandt, das nur für Single-Betrieb ist.

Funktionsspezifische Fehler finden Sie in der Funktionsreferenz.

Falls nötig, kann der Anwender die Aufzeichnung der Fehler aktivieren:

- *meErrorSetDefaultProc()*:
 - **Windows:** Es wird eine Message-Box mit Fehlercode und Fehlerbeschreibung angezeigt. Durch drücken von „OK“ wird die Message-Box einfach geschlossen, durch drücken von „Cancel“ wird die Messagebox geschlossen und weitere Fehlermeldungen unterdrückt.
 - **Linux:** Die Standard-Fehlerausgabe wird benutzt. Normalerweise wird der Fehler in die Datei `/var/log/messages` und zur Konsole geschrieben. Dieses Verhalten kann vom Nutzer modifiziert werden.
- *meErrorSetUserProc()*: Anwenderdefinierbare Logging-Funktion aufrufen.

3.4 Betriebsarten

3.4.1 Single-Betrieb

Diese Betriebsart dient sowohl dem Lesen als auch dem Schreiben einzelner Werte. Bevor jegliche Ein-/Ausgabe ausgeführt werden kann muss das Subdevice mit der Funktion *meIOSingleConfig()* konfiguriert werden. Anschließend wird die Funktion *meIOSingle()* aufgerufen um die Operation selbst auszuführen. Sobald konfiguriert, kann ein Subdevice öfters lesen und schreiben. Mehrere Single-Operationen können in eine Liste geschrieben und durch Aufruf der Funktion *meIOSingle()* nacheinander abgearbeitet werden.

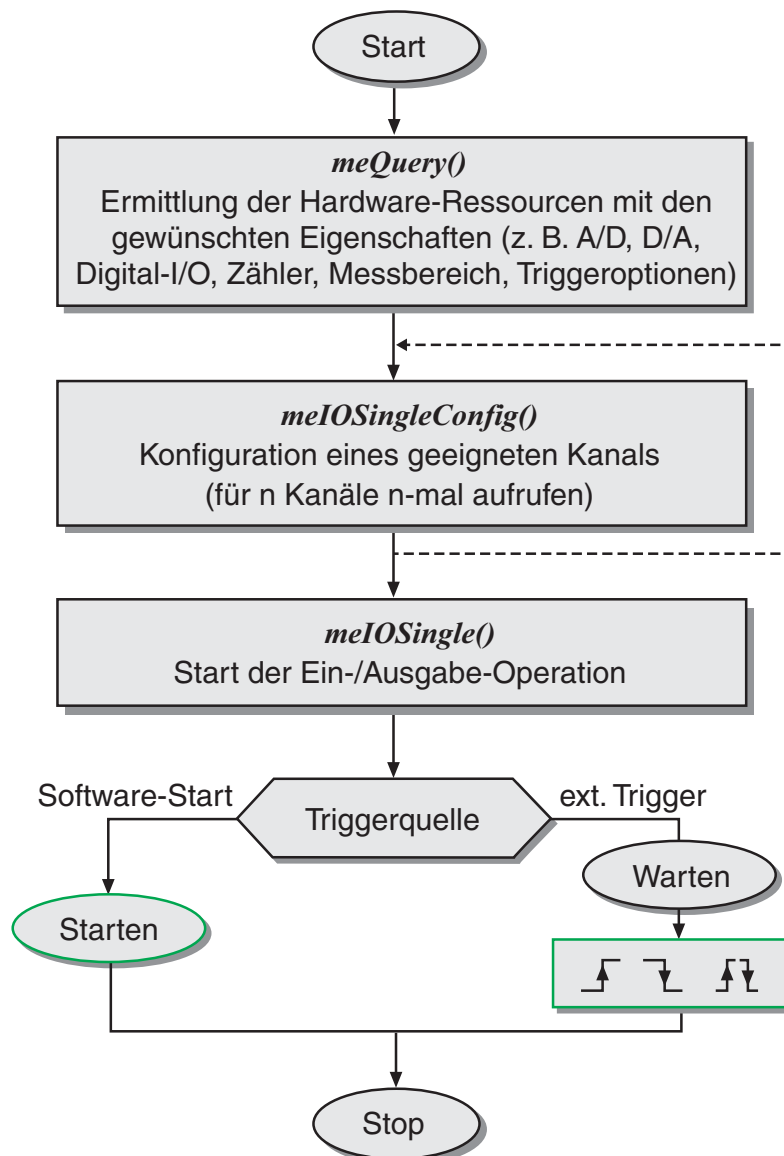


Abbildung 10: Single-Betrieb

Wichtig: Operationen in einer Liste vom Typ `meIOSingle_t` werden in der Reihenfolge des Eintrags ausgeführt. Falls eine Operation warten muss (z. B. auf ein externes Triggersignal), müssen auch alle Folgenden warten. Die Abarbeitung der Liste wird gestoppt, sobald eine Operation fehlschlägt.

3.4.1.1 Operation starten/Trigger-Optionen

Eine Single-Operation kann auf vielfältige Weise gestartet werden. Der Parameter `<iTrigType>` der Funktion `meIOSingleConfig()` legt die Triggerart fest. Die hier verfügbaren Optionen hängen von dem betreffenden Subdevice ab. Eine Auflistung aller gegenwärtig verfügbaren Trigger-Optionen finden Sie hier:

- **Kein Trigger verfügbar** (`ME_TRIG_TYPE_NONE`):
Einige Subdevices (wie Zähler) bieten keine Triggeroptionen. Die Operation wird ausgeführt sobald `meIOSingle()` aufgerufen wird. Es ist kein Time-Out einstellbar.
- **Software-Trigger** (`ME_TRIG_TYPE_SW`):
Subdevice wird unmittelbar nach Aufruf der Funktion `meIOSingle()` gelesen bzw. geschrieben oder via „Synchronliste“ (siehe Seite 88).
- **Externer Digital-Trigger** (`ME_TRIG_TYPE_EXT_DIGITAL`):
Subdevice kann durch ein externes Triggersignal gestartet werden. Der Digital-Trigger kann auch zum Start der „Synchronliste“ (siehe Seite 88) verwendet werden.
- **Externer Analog-Trigger** (`ME_TRIG_TYPE_EXT_ANALOG`):
Subdevice kann durch ein externes Analogsignal gestartet werden, das von einem internen Komparator ausgewertet wird (see folgende Abbildung).

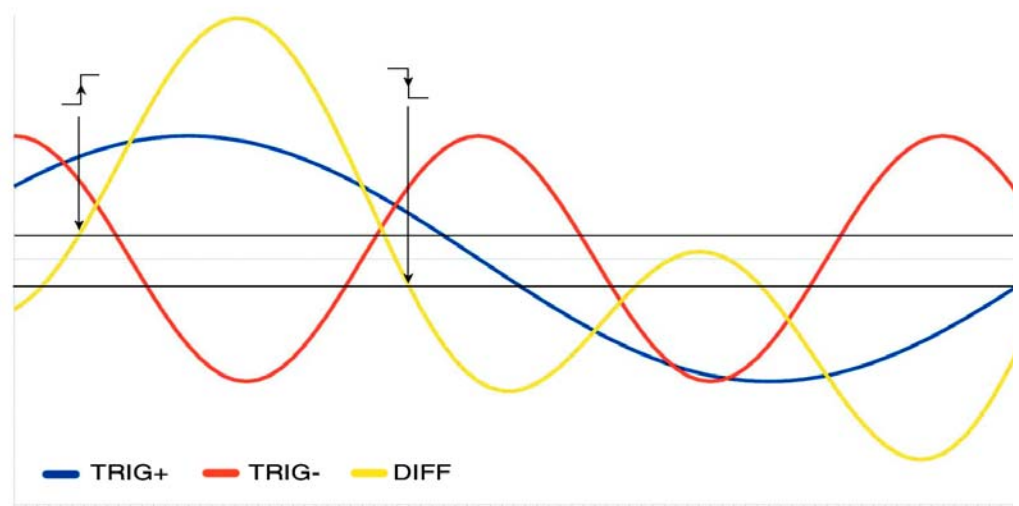


Abbildung 11: Analog-Trigger

Hinweis: Der Analog-Trigger hat ein bestimmtes Hysterese-Verhalten. Details finden Sie im entsprechenden Hardware-Handbuch.

Die Triggerflanken können mit dem Parameter `<iTrigEdge>` so konfiguriert werden, dass sie auf steigende („RISING“), fallende („FALLING“) oder beliebige Flanke („ANY“) triggern.



Abbildung 12: Triggerflanken

Parameter `<iTrigChan>` wird verwendet um Einträge einer „Synchronliste“ hinzuzufügen. Details finden Sie im Kapitel siehe „Synchronstart“ auf Seite 88.

Hinweis: Bitte beachten Sie das Hardware-Handbuch bzgl. möglicher Trigger-Modi.

Wichtig: Parameter wie `<iChannel>`, die in `meIOSingleConfig()` und `meIOSingle()`, übergeben werden, haben unterschiedliche Bedeutung in Abhängigkeit des Subdevice-Typs. Genaue Angaben finden Sie in der Hilfedatei des ME-iDS-Treibersystems.

3.4.1.2 Analoge Ein-/Ausgabe

Werte werden als Integer gelesen und geschrieben. Sie können mit der Funktion *meUtilityDigitalToPhysical()* in die jeweilige physikalische Einheit umgerechnet werden. Die Funktion *meUtilityPhysicalToDigital()* erlaubt die umgekehrte Umrechnung. Die Parameter *<dMin>*, *<dMax>* und *<iMaxData>* der Umrechnungsfunktionen müssen mit den Parametern *<pdMin>*, *<pdMax>* und *<piMaxData>* (siehe *meQueryRangeInfo()*) jenes Bereichs korrespondieren, welcher im Parameter *<iSingleConfig>* der Funktion *meIOSingleConfig()* verwendet wird.

Das ME-iDS bietet mehrere Funktionen, die dem Anwender helfen, den am besten passenden Bereich für eine bestimmte Ein- oder Ausgabe-Operation zu ermitteln.

- *meQueryNumberRanges()*: gibt die Anzahl der verfügbaren Bereiche für ein bestimmtes Subdevice zurück.
- *meQueryRangeInfo()*: gibt die Parameter zurück, die den gewählten Bereich definieren:
 - *<piUnit>* Physikalische Einheit (z. B.: ME_UNIT_VOLT für Volt).
 - *<pdMin>* Untere Bereichsgrenze in der gewählten physikalischen Einheit.
 - *<pdMax>* Obere Bereichsgrenze in der gewählten physikalischen Einheit.
 - *<piMaxData>* Integer-Wert, der die digitale Auflösung repräsentiert (z. B.: 65535 für 16 bit Auflösung). Die untere Bereichsgrenze ist immer 0.
- *meQueryRangeByMinMax()*: gibt den Index des Bereichs zurück, der am besten zu Ihren Anforderungen passt. Es wird stets der kleinstmögliche Bereichsindex zurückgegeben, der die vorgegebenen Minimal- und Maximalwerte einschließt.

Der Massebezug für analoge Messungen wird mit dem Parameter *<iRef>* konfiguriert. Mit differentiellen Messungen können nur bipolare Eingangsbereiche kombiniert werden (siehe auch ME-iDS-Hilfdatei und Hardware-Handbuch der jeweiligen Karte).

3.4.1.3 Digitale Ein-/Ausgabe

Grundsätzlich müssen digitale Ports mit der Funktion *meIO-SingleConfig()* für Eingabe (Lesen) oder Ausgabe (Schreiben) konfiguriert werden. Die Pins der bidirektionalen Subdevices werden nach einem Reset auf Eingang geschaltet, damit keine unerwünschten Signalpegel ausgegeben werden. Da die Konfiguration - je nach Hardware - in unterschiedlicher Bitbreite (Bit, Byte, Word, usw.) erfolgen kann, wird dafür der Begriff "Block" eingeführt. Der Parameter *<iChannel>* bezeichnet den Index des Blocks, der konfiguriert werden soll (siehe auch Funktion *meQuerySubdeviceCaps()* auf Seite 112).

Die Funktion *meQuerySubdeviceCaps()* ermittelt die Breite eines Blocks. Zum Beispiel: ME_CAPS_DIO_DIR_BYTE bedeutet, dass der Port mit einer Blockgröße von einem Byte konfiguriert werden muss. Es ist nicht möglich, Bit 0 als Eingang und die anderen 7 Bits als Ausgang zu konfigurieren. Im Vergleich dazu bedeutet ME_CAPS_DIO_DIR_BIT, dass die Blockgröße ein Bit ist und daher jedes Bit einzeln konfiguriert werden kann. Mit dem Flag ME_IO_SINGLE_CONFIG_NO_FLAGS werden immer alle Kanäle des Subdevices verwendet.

Der Parameter *<iSingleConfig>* definiert die Richtung eines Blocks. Abhängig von der Hardware-Implementation sind verschiedene Werte zulässig. Falls nur Standard-Ein-/Ausgabe-Operationen möglich sind, verwenden Sie ME_SINGLE_CONFIG_DIO_INPUT oder ME_SINGLE_CONFIG_DIO_OUTPUT. Ansonsten beachten Sie bitte die ME-iDS-Hilfdatei und das Hardware-Handbuch bzgl. Optionen und der Standard-Einstellung.

Zum Beispiel sind die optoisolierten Karten ME-5810 und ME-8100 ausgangsseitig mit Leistungstreibern ausgestattet, die zusätzliche Optionen bieten:

- Ausgang mit Sink-Treiber
(ME_SINGLE_CONFIG_DIO_SINK)
- Ausgang mit Source-Treiber
(ME_SINGLE_CONFIG_DIO_SOURCE)
- Hochohmiger Zustand
(ME_SINGLE_CONFIG_DIO_HIGH_IMPEDANCE)

Eine Lese- oder Schreib-Operation mit der Funktion *meIOSingle()* benötigt die Angabe der Portbreite im Parameter *<iFlags>* für jeden Eintrag (ME_IO_SINGLE_TYPE_DIO...). Parameter *<iChannel>* gibt den

Index des Blocks an auf den zugegriffen werden soll. Mit dem Flag `ME_IO_SINGLE_TYPE_NO_FLAGS` werden immer alle Kanäle des Subdevices verwendet.

Falls Sie mit der ME-4680 timergesteuert ein Bitmuster auf einen (oder mehrere) Digital-Ports ausgeben möchten, übergeben Sie `ME_SINGLE_CONFIG_DIO_BIT_PATTERN` im Parameter `<iSingleConfig>` der Funktion *meIOSingleConfig()* und setzen Sie Parameter `<iRef>` auf `ME_REF_DIO_FIFO_LOW` bzw. `ME_REF_DIO_FIFO_HIGH`. Ansonsten verwenden Sie `ME_REF_NONE`. Details finden Sie im Kapitel 3.4.3.2 Bitmuster-Ausgabe der ME-4680 auf Seite 86 und Anhang B3 ab Seite 228.

3.4.1.4 Frequenz-Ein-/Ausgabe

Ein Subdevice zur Frequenzmessung (Eingabe) bzw. Frequenzausgabe (Impulsgenerator) kann ein oder mehrere Kanäle (Channels) enthalten. Die Zuordnung der Kanäle zu den Subdevices entnehmen Sie bitte der ME-iDS Hilfe-Datei.

Grundsätzlich muss ein Subdevice vom Typ „`ME_TYPE_FI`“, „`ME_TYPE_FO`“ oder „`ME_TYPE_FIO`“ in der Betriebsart Single mit der Funktion *meIOSingleConfig()* konfiguriert werden. Die Anzahl der Kanäle kann mit der Funktion *meQueryNumberChannels()* ermittelt werden. Im Parameter `<iChannel>` übergeben Sie den Index des Kanals.

Zur Beschreibung des Rechtecksignals wurden zwei Variablen eingeführt, die für Ein- und Ausgabe gleichermaßen gelten. Der eine Wert gibt die Periodendauer T an, der andere Wert die Impulsdauer der ersten Phase der Periode t_{1p} . Bei der Frequenzmessung startet die Messung mit der ersten positiven Flanke und endet mit der darauffolgenden positiven Flanke. Die dazwischen liegende fallende Flanke definiert das Ende der ersten Phase. Im Impulsgenerator-Betrieb startet die Ausgabe standardmäßig mit High-Pegel und wechselt nach Ablauf der ersten Phase nach Low-Pegel.

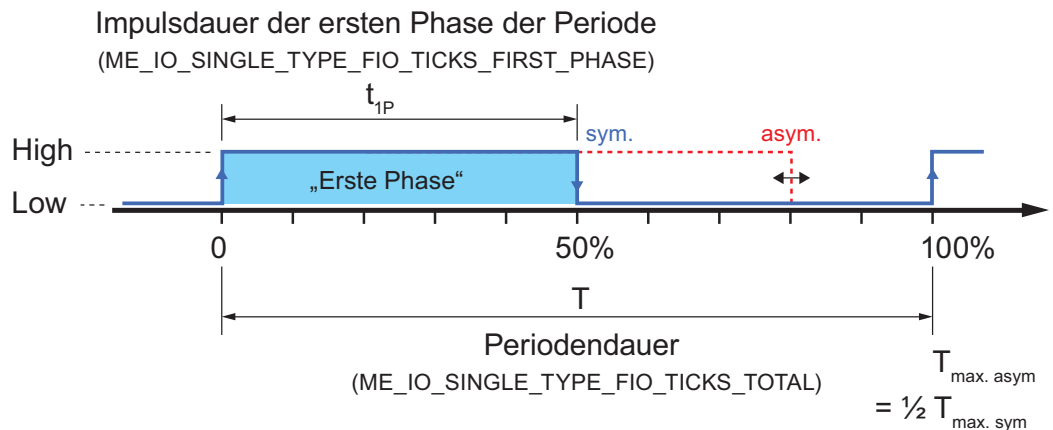


Abbildung 13: Signaldefinition

Beachten Sie, dass je nach Systemtakt unterschiedliche Werte für die maximale Periodendauer $T_{\max.}$ in Abhängigkeit vom Tastverhältnis gelten. Es wird zwischen Rechtecksignalen mit asymmetrischem Tastverhältnis $T_{\max. \text{ asym.}}$ und symmetrischem Tastverhältnis $T_{\max. \text{ sym.}}$ unterschieden. In der Regel gilt: $T_{\max. \text{ asym.}} = \frac{1}{2} T_{\max. \text{ sym.}}$.

3.4.1.4.1 Frequenzmessung

Mit der Betriebsart Frequenzmessung (FI=“Frequency Input“) können Sie Periodendauer und Tastverhältnis von Rechtecksignalen ermitteln. Jeder Frequenz-Messkanal (Eingang) wird derzeit als Funktionsgruppe vom Typ ME_TYPE_FI, Untertyp ME_SUBTYPE_SINGLE angesprochen. Es sind die Funktionen *meQuery...()*, *meIOSingleConfig()*, *meIOSingle()* und *meIOSingleTicksToTime()* relevant:

- Subdevice mit *meQuery...()* Funktionen ermitteln.
- Konfiguration des Subdevices mit der Funktion *meIOSingleConfig()*:
 - In Parameter <iChannel> den Kanal-Index übergeben.
 - In Parameter <iSingleConfig> müssen Sie ME_SINGLE_CONFIG_FIO_INPUT übergeben
 - Die Parameter <iRef>, <iTrigChan>, <iTrigType> und <iTrigEdge> werden hier nicht benötigt. Bitte übergeben Sie ME_VALUE_NOT_USED.
 - In Parameter <iFlags> übergeben Sie ME_IO_SINGLE_CONFIG_FI_SINGLE_MODE.

- Um Periodendauer und Dauer der ersten Phase der Periode einlesen zu können, müssen Sie die Funktion *meIOSingle()* zweimal aufrufen. Je nach Option im Parameter *<iFlags>* liefert *<iValue>* entweder die gesamte Periodendauer (in Ticks) oder die Dauer der ersten Phase der Periode (in Ticks) zurück.

Im Standardbetrieb empfehlen wir folgende Vorgehensweise:

1. Flags für Einlesen der Periodendauer bitweise verodern:
ME_IO_SINGLE_TYPE_FIO_TICKS_TOTAL „or“
ME_IO_SINGLE_TYPE_NONBLOCKING
2. Flags für Einlesen der Dauer der ersten Phase bitweise verodern:
ME_IO_SINGLE_TYPE_FIO_TICKS_FIRST_PHASE
„or“ ME_IO_SINGLE_TYPE_NONBLOCKING.

Zur wiederholten Erfassung langsamer Frequenzen ist folgende Variante möglich (siehe auch Parameter *<iFlags>* der Funktion *meIOSingleConfig()*):

1. Flags für Einlesen der Periodendauer bitweise verodern:
ME_IO_SINGLE_TYPE_FIO_TICKS_TOTAL „or“
ME_IO_SINGLE_TYPE_FI_LAST_VALUE „or“
ME_IO_SINGLE_TYPE_NONBLOCKING
2. Flags für Einlesen der Dauer der ersten Phase bitweise verodern:
ME_IO_SINGLE_TYPE_FIO_TICKS_FIRST_PHASE
„or“ ME_IO_SINGLE_TYPE_FI_LAST_VALUE „or“
ME_IO_SINGLE_TYPE_NONBLOCKING.

- Zur einfachen Umrechnung von Ticks in Sekunden können Sie die Funktion *meIOSingleTicksToTime()* verwenden. Sie müssen die Funktion für Periodendauer und Dauer der ersten Phase der Periode getrennt aufrufen. Beachten Sie, dass die Option für *<iTimer>* mit *<iFlags>* in der Funktion *meIOSingle()* korrespondiert.

In Parameter *<iTicksLow>* werden die Ticks von *<iValue>* (siehe oben) übergeben. Parameter *<pdTime>* gibt schließlich einen Zeiger auf den errechneten Wert in Sekunden zurück.

Hinweis: Wenn Sie die Größen Frequenz und Tastverhältnis benötigen können Sie diese leicht aus den Rückgabewerten von *<pdTime>* berechnen. Es gilt:

Frequenz [Hz] = 1/Periodendauer [s]

Tastverhältnis [%] = („Dauer der ersten Phase der Periode“ [s] / Periodendauer [s]) × 100

Der Parameter <iTicksHigh> ist reserviert für zukünftige Erweiterungen. Bitte übergeben Sie 0.

3.4.1.4.2 Impulsgenerator

Mit der Betriebsart Impulsgenerator (FO=“Frequency Output“) können Sie Rechtecksignale mit variablem Tastverhältnis mit einer Auflösung von einem Tick ausgeben. Jeder Impulsgenerator-Kanal (Ausgang) wird derzeit als Funktionsgruppe vom Typ ME_TYPE_FO, Untertyp ME_SUBTYPE_SINGLE angesprochen. Es sind die Funktionen *meQuery...()*, *meIOSingleConfig()*, *meIOSingle()* und *meIOSingleTimeToTicks()* relevant:

- Subdevice mit *meQuery...()* Funktionen ermitteln.
- Konfiguration des Subdevices mit der Funktion *meIOSingleConfig()*:
 - In Parameter <iChannel> den Kanal-Index übergeben.
 - In Parameter <iSingleConfig> müssen Sie ME_SINGLE_CONFIG_FIO_OUTPUT übergeben.
 - In Parameter <iTrigChan> übergeben Sie entweder ME_TRIG_CHAN_DEFAULT oder ME_TRIG_CHAN_SYNCHRONOUS für den Synchron-Start mehrerer Generatoren (siehe „Synchronstart“ auf Seite 88).
 - Die Parameter <iRef>, <iTrigType> und <iTrigEdge> werden hier nicht benötigt. Bitte übergeben Sie ME_VALUE_NOT_USED.
 - In Parameter <iFlags> übergeben Sie ME_IO_SINGLE_CONFIG_NO_FLAGS.
- Zur einfachen Umrechnung des auszugebenden Signals von Sekunden in Ticks dient die Funktion *meIOSingleTimeToTicks()*. Sie müssen die Funktion für Periodendauer und Dauer der ersten Phase der Periode getrennt aufrufen.
 - Wählen Sie dazu in Parameter <iTimer> zunächst ME_TIMER_FIO_TOTAL für die Periodendauer und dann ME_TIMER_FIO_FIRST_PHASE für die Dauer der ersten

Phase der Periode. Übergeben Sie in Parameter `<pdTime>` jeweils den gewünschten Wert in Sekunden.

- `<piTicksLow>` liefert jeweils einen Zeiger mit den Ticks zurück, die im nächsten Schritt an Parameter `<iValue>` der Funktion *meIOSingle()* korrespondierend zu `<iFlags>` übergeben werden.
- Zur Übergabe von Periodendauer und der Dauer der ersten Phase muss die Funktion *meIOSingle()* zweimal aufgerufen werden.

Im Standardbetrieb empfehlen wir folgende Vorgehensweise:

1. Flags für Übergabe der Periodendauer bitweise verodern:
`ME_IO_SINGLE_TYPE_FIO_TICKS_TOTAL` „or“
`ME_IO_SINGLE_TYPE_FO_UPDATE_ONLY`

2. Flags für Übergabe der Dauer der ersten Phase bitweise verodern:
`ME_IO_SINGLE_TYPE_FIO_TICKS_FIRST_PHASE`
 „or“ `ME_IO_SINGLE_TYPE_FO_START_SOFT`.

- Übergeben Sie `ME_DIR_OUTPUT` im Parameter `<iDir>`. Mit `ME_DIR_INPUT` auch rücklesbar.
- In Parameter `<iValue>` werden die Ticks von `<piTicksLow>` (siehe oben) übergeben. Beachten Sie, dass die Funktion *meIOSingle()* zweimal aufgerufen werden muss und dass der Wert mit `<iFlags>` korrespondiert.
- Der Start der Ausgabe kann durch geeignete Kombination der Flags in Parameter `<iFlags>` gesteuert werden. Verodern Sie dazu `ME_IO_SINGLE_TYPE_FIO_TICKS_TOTAL` bzw. `ME_IO_SINGLE_TYPE_FIO_TICKS_FIRST_PHASE` bitweise mit einer oder mehreren der folgenden Optionen:

`ME_IO_SINGLE_TYPE_FO_UPDATE_ONLY`

Der Ausgabewert soll geändert aber nicht sofort ausgegeben werden. Keine Verknüpfung mit weiteren Flags möglich. Standard: der neue Wert wird sofort ausgegeben.

`ME_IO_SINGLE_TYPE_FO_START_SOFT`

Der Wert wird erst mit Ende der aktuellen Periode (falls bereits gestartet) ausgegeben. Kann mit `ME_IO_SINGLE_TYPE_FO_START_LOW` bitweise verodert werden. Standard: der neue Wert wird sofort ausgegeben.

ME_IO_SINGLE_TYPE_TRIG_SYNCHRONOUS

Alle Subdevices, die mit Parameter `<iTrigChan>` der Funktion *meIOSingleConfig()* in die Synchronliste eingetragen wurden, werden simultan gestartet (siehe auch Kap. 3.4.3.3 auf Seite 88). Standard: Subdevice startet unabhängig.

ME_IO_SINGLE_TYPE_FO_START_LOW

Standardmäßig hat die erste Phase des Rechtecksignals High-Pegel. Wird dieses Flag gesetzt, startet die Ausgabe mit Low-Pegel. Kann mit `ME_IO_SINGLE_TYPE_FO_START_SOFT` bitweise verodert werden.

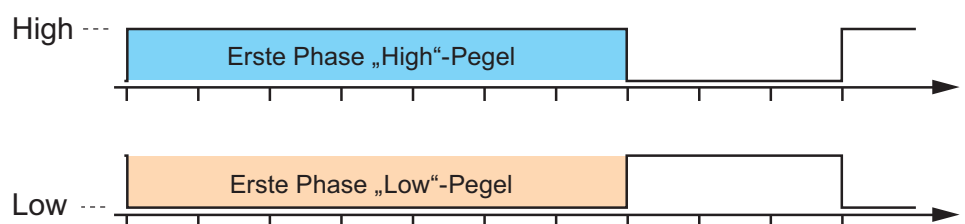


Abbildung 14: Negierte Impulsausgabe

3.4.1.5 Zählerbetrieb

Grundsätzlich kann das ME-iDS verschiedenste Zählertypen unterstützen. Im Augenblick ist der Standard-Zählerbaustein vom Typ 8254 implementiert, der über drei 16 bit Zähler verfügt.

Jeder Zähler wird als Subdevice vom Typ `ME_TYPE_CTR`, Untertyp `ME_SUBTYPE_CTR_8254` angesprochen. Für Zähler-Subdevices hat der Parameter `<iChannel>` keine Bedeutung, bitte übergeben Sie '0'. *meQueryNumberChannels()* gibt stets '1' zurück.

Hinweis: Es gibt keine „STOP“-Funktion für Zähler. Verwenden Sie stattdessen *meIOResetSubdevice()*. Trigger-Optionen sind für Zähler nicht verfügbar.

Parameter `<iRef>` definiert die Taktquelle für die Zähler:

- Externer Taktgenerator als Taktquelle (`ME_REF_CTR_EXTERNAL`)
- Ausgang des vorigen Zählers als Taktquelle (`ME_REF_CTR_PREVIOUS`)
- Interner 1 MHz Taktgenerator als Taktquelle (`ME_REF_CTR_INTERNAL_1MHZ`)

- Interner 10 MHz Taktgenerator als Taktquelle
(ME_REF_CTR_INTERNAL_10MHZ)

Parameter <iSingleConfig> bezieht sich auf die ausgewählte Betriebsart. Die 3 Zähler eines Bausteins können unabhängig voneinander für eine der folgenden 6 Betriebsarten konfiguriert werden (siehe auch Anhang B1 ab Seite 222):

- Modus 0: Zustandsänderung bei Nulldurchgang
(ME_SINGLE_CONFIG_CTR_8254_MODE_0)
- Modus 1: Retriggerbarer „One Shot“
(ME_SINGLE_CONFIG_CTR_8254_MODE_1)
- Modus 2: Asymmetrischer Teiler
(ME_SINGLE_CONFIG_CTR_8254_MODE_2)
- Modus 3: Symmetrischer Teiler
(ME_SINGLE_CONFIG_CTR_8254_MODE_3)
- Modus 4: Zählerstart durch Softwaretrigger
(ME_SINGLE_CONFIG_CTR_8254_MODE_4)
- Modus 5: Zählerstart durch Hardwaretrigger
(ME_SINGLE_CONFIG_CTR_8254_MODE_5)

Hinweis: Der tatsächliche Spannungspegel an den Ein-/Ausgängen der Zähler hängt von der jeweiligen Hardware ab. Zum Beispiel entspricht bei optoisolierten Varianten der ME-4600-Serie ein High-Pegel am Ausgang dem Zustand „hochohmig“ und ein Low-Pegel dem Zustand „leitend“. Bitte beachten Sie das entsprechende Hardware-Handbuch. Die Logik-Pegel in der folgenden Beschreibung gelten für den Zählerbaustein ohne weitere Beschaltung.

Die folgende Abbildung soll den Programmablauf kurz beschreiben:

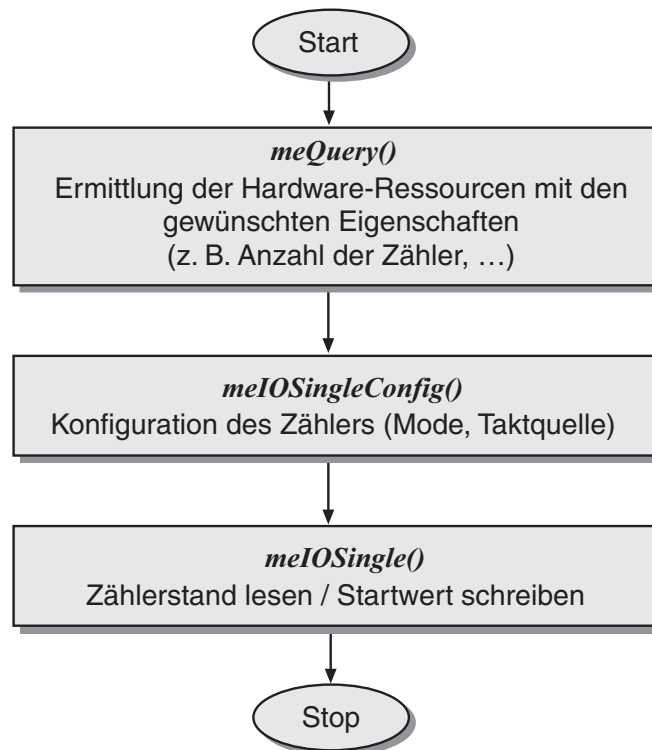


Abbildung 15: Programmierung der Zähler

3.4.1.5.1 Modus 0: Zustandsänderung bei Nulldurchgang

Diese Betriebsart ist z. B. zur Signalisierung eines Interrupts bei Nulldurchgang geeignet. Der Zähler-Ausgang (OUT_0...2) geht in den Low-Zustand, sobald der Zähler initialisiert wird oder ein neuer Startwert in den Zähler geladen wird. Zur Freigabe des Zählers muss der GATE-Eingang mit High-Pegel beschaltet werden. Sobald der Zähler geladen und freigegeben wurde, beginnt er abwärts zu zählen während der Ausgang im Low-Zustand bleibt.

Bei Erreichen des Nulldurchganges geht der Ausgang in den High-Zustand und bleibt dort, bis der Zähler neu initialisiert wird oder ein neuer Startwert geladen wird. Auch nach Erreichen des Nulldurchganges wird weiter abwärts gezählt. Sollte während des Zählvorganges ein Zählerregisters erneut geladen werden, hat dies zur Folge, dass:

1. beim Schreiben des ersten Bytes der momentane Zählvorgang gestoppt wird
2. beim Schreiben des zweiten Bytes der neue Zählvorgang gestartet wird.

3.4.1.5.2 **Modus 1: Retriggerbarer „One-Shot“**

Der Zähler-Ausgang (OUT_0...2) geht in den High-Zustand, sobald der Zähler initialisiert wird. Nachdem ein Startwert in den Zähler geladen wurde geht der Ausgang mit dem auf den ersten Triggerimpuls am GATE-Eingang folgenden Takt in den Low-Zustand. Nach Ablauf des Zählers geht der Ausgang wieder in den High-Zustand.

Mit einer geeigneten Flanke am GATE-Eingang kann der Zähler jederzeit auf den Startwert zurückgesetzt („retriggered“) werden. Der Ausgang bleibt solange im Low-Zustand bis der Zähler den Nulldurchgang erreicht.

Der Zählerstand kann jederzeit ohne Auswirkung auf den momentanen Zählvorgang, ausgelesen werden.

3.4.1.5.3 **Modus 2: Asymmetrischer Teiler**

In diesem Modus arbeitet der Zähler als Frequenzteiler. Der Zähler-Ausgang (OUT_0...2) geht nach der Initialisierung in den High-Zustand. Nach Freigabe des Zählers durch geeignete Beschaltung des GATE-Eingangs wird abwärts gezählt, während der Ausgang noch im High-Zustand verbleibt. Sobald der Zähler den Wert 0001Hex erreicht hat, geht der Ausgang für die Dauer einer Taktperiode in den Low-Zustand. Dieser Ablauf wiederholt sich periodisch solange der GATE-Eingang freigegeben ist, ansonsten geht der Ausgang sofort in den High-Zustand.

Wird das Zählerregister zwischen zwei Ausgangs-Pulsen erneut geladen, so beeinflusst dies den momentanen Zählvorgang nicht, die nächste Periode arbeitet jedoch mit den neuen Werten.

3.4.1.5.4 **Modus 3: Symmetrischer Teiler**

Dieser Modus arbeitet ähnlich wie Modus 2 mit dem Unterschied, dass der geteilte Takt ein symmetrisches Tastverhältnis besitzt (nur für geradzählige Zählerwerte geeignet). Der Zähler-Ausgang (OUT_0...2) geht nach der Initialisierung in den High-Zustand. Nach Freigabe des Zählers durch geeignete Beschaltung des GATE-Eingangs wird in 2er-Schritten abwärts gezählt. Nun wechselt der Ausgang, mit der Anzahl Perioden des halben Startwertes bezogen auf den Eingangstakt (beginnend mit High-Pegel). Solange der Gate-Eingang freigegeben ist, wiederholt sich dieser Ablauf periodisch, ansonsten geht der Ausgang sofort in den High-Zustand.

Wird das Zählerregister zwischen zwei Ausgangs-Pulsen erneut geladen, so beeinflusst dies den momentanen Zählvorgang nicht, die nächste Periode arbeitet jedoch mit den neuen Werten.

3.4.1.5.5 **Modus 4: Zählerstart durch Softwaretrigger**

Der Zähler-Ausgang (OUT_0...2) geht in den High-Zustand, sobald der Zähler initialisiert wird. Zur Freigabe des Zählers muss der Gate-Eingang geeignet beschaltet werden. Sobald der Zähler geladen (Software-Trigger) und freigegeben wurde, beginnt er abwärts zu zählen, während der Ausgang noch im High-Zustand bleibt.

Bei Erreichen des Nulldurchganges geht der Ausgang für die Dauer einer Takt-Periode in den Low-Zustand. Danach geht der Ausgang wieder in den High-Zustand und bleibt dort bis der Zähler initialisiert und ein neuer Startwert geladen wird.

Wird das Zählerregister während eines Zählvorganges erneut geladen, so wird der neue Startwert mit dem nächsten Takt geladen.

3.4.1.5.6 **Modus 5: Zählerstart durch Hardwaretrigger**

Der Zähler-Ausgang (OUT_0...2) geht in den High-Zustand, sobald der Zähler initialisiert wird. Nachdem ein Startwert in den Zähler geladen wurde beginnt der Zählvorgang mit dem auf den ersten Triggerimpuls am GATE-Eingang folgenden Takt. Bei Erreichen des Nulldurchganges geht der Ausgang für die Dauer einer Takt-Periode in den Low-Zustand. Danach geht der Ausgang wieder in den High-Zustand und bleibt dort bis ein erneuter Triggerimpuls ausgelöst wird.

Wird das Zählerregister zwischen zwei Triggerimpulsen erneut geladen, so wird der neue Startwert erst nach dem nächsten Triggerimpuls berücksichtigt.

Mit einer positiven Flanke am Gate-Eingang kann der Zähler jederzeit auf den Startwert zurückgesetzt („retriggered“) werden. Der Ausgang bleibt solange im High-Zustand bis der Zähler den Nulldurchgang erreicht.

3.4.1.5.7 Modus „Pulsweiten-Modulation“

Ein spezieller Anwendungsfall der Zählerbausteine vom Typ 8254 ist die Ausgabe eines Rechtecksignals mit variablem Tastverhältnis („PWM“-Modus). Damit können Sie an OUT_2 ein Rechteck-Signal von max. 50 kHz bei variablem Tastverhältnis ausgeben.

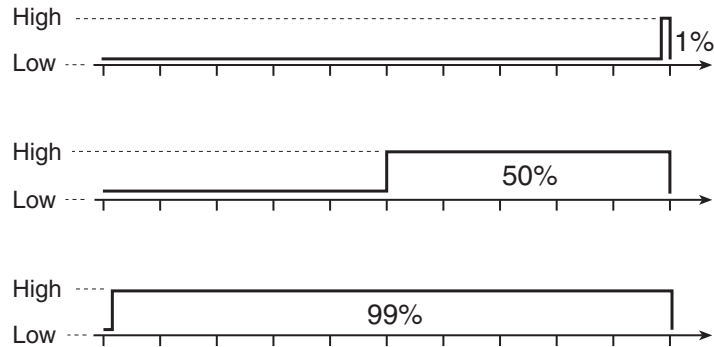


Abbildung 16: PWM-Signal

Voraussetzung ist die geeignete Beschaltung der Ein- und Ausgänge (CLK, GATE, OUT) durch externe Verdrahtung. Lesen Sie bitte das entsprechende Kapitel zur PWM-Beschaltung - insbesondere bei opto-isolierten Zählern - im Hardware-Handbuch.

Grundsätzlich gilt folgende Beschaltung:

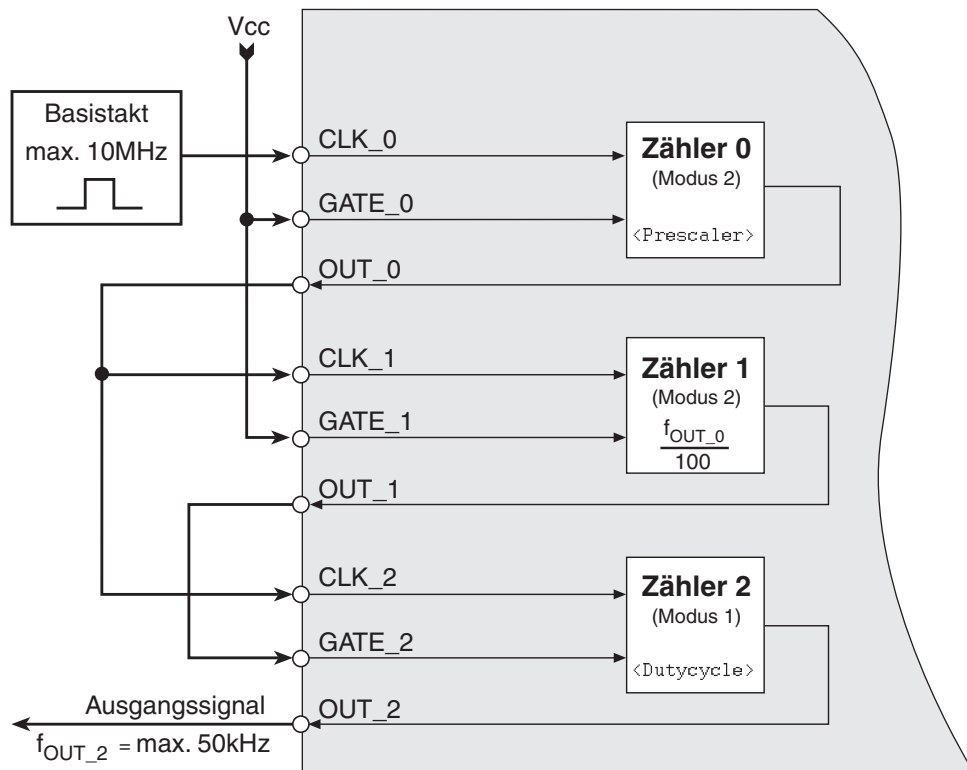


Abbildung 17: Beschaltung Pulsweiten-Modulation

Zähler 0 wird als Vorteiler für den extern eingespeisten Basistakt verwendet. Über den Parameter `<iPrescaler>` können Sie die Frequenz f_{OUT_2} folgendermaßen einstellen:

$$f_{\text{OUT}_2} = \frac{\text{Basistakt}}{\text{<iPrescaler>} \cdot 100} \quad (\text{mit } \text{<iPrescaler>} = 2 \dots (2^{16} - 1))$$

Mit dem Parameter `<iDutyCycle>` kann das Tastverhältnis zwischen 1...99% in Schritten von 1% eingestellt werden (siehe Abb. 16 auf Seite 60). Die Ausgabe wird unmittelbar durch Aufruf der Funktion *meUiltiyPWMStart()* gestartet und mit *meUiltiyPWMStop()* beendet. Es ist keine weitere Programmierung der Zähler erforderlich.

Beachten Sie, dass bei optoisolierten Geräten der Ausgang OUT_2 in der Regel als „Open Collector“ Ausgang ausgeführt ist. D. h. logisch „1“ bedeutet, dass der Ausgang leitend ist und logisch 0, dass der Ausgang hochohmig ist (siehe auch Hardware-Handbuch).

3.4.2 Streaming-Betrieb

Der Begriff „Streaming“ steht für alle Operationen, in denen Daten via FIFO übertragen werden. Die Zeitsteuerung übernehmen wahlweise ein Timer und/oder externe Trigger-Signale. Das entsprechende Subdevice muss vom Untertyp ME_SUBTYPE_STREAMING sein. Dies kann mit der Funktion *meQuerySubdeviceType()* (Parameter <piSubtype>) überprüft werden.

Sofern von der Hardware unterstützt, betrifft dies nicht nur analoge Ein-/Ausgabe sondern auch digitale Ein-/Ausgabe. Eine detaillierte Beschreibung der Bitmuster-Ausgabe der ME-4680 finden Sie in Anhang B3 ab Seite 228.

Gehen Sie folgendermaßen vor:

1. Abfrage der Fähigkeiten mit den *meQuery...* Funktionen bzw. Ermitteln der Properties mit den *meProperty...* Funktionen.
2. Konfigurieren des Subdevices mit der Funktion *meIOStreamConfig()* und *meIOSetChannelOffset()* sofern es sich um ein Subdevice mit einstellbarem Offset handelt.
3. Bei einer Ausgabe-Operation muss zuerst der Pufferspeicher mit der Funktion *meIOStreamWrite()* vorgeladen werden. Übergeben Sie im Parameter <iWriteMode> die Option ME_WRITE_MODE_PRELOAD zu benutzen.
4. Starten der Streaming-Operation mit der Funktion *meIOStreamStart()*.
5. Lesen oder Schreiben der Daten (*meIOStreamRead()* bzw. *meIOStreamWrite()*).
6. Stoppen der Streaming-Operation mit der Funktion *meIOStreamStop()* oder *meIOReset...()*.

Hinweis: Ein Streaming-Betrieb ist für Subdevices vom Typ Zähler (ME_TYPE_CTR), Frequenz-Ein-/Ausgabe (ME_TYPE_FIO/ME_TYPE_FI/ME_TYPE_FO) und Interrupt (ME_TYPE_IRQ) nicht möglich.

3.4.2.1 Hardware-Ressourcen ermitteln

Zur Ermittlung der Hardware-Ressourcen können Sie wahlweise die Query-Funktionen (*meQuery...*) oder die Property-Funktionen (*meProperty...*) verwenden. Mit den Query-Funktionen können Sie z. B. ein Device auf das Vorhandensein bestimmter Subdevices oder bestimmter Fähigkeiten (Caps) abfragen (z. B.: „Hat das Device ein Subdevice vom Typ: Analog-Eingabe?“). Einen etwas anderen Ansatz verfolgen die Property-Funktionen. Einer Baumstruktur gleich können Sie Properties und Attribute über deren Property-Pfad ermitteln und gegebenenfalls auch setzen.

3.4.2.2 Hardware konfigurieren

Um ein Subdevice für den Streaming-Betrieb zu konfigurieren, müssen immer zwei Datenstrukturen vorbereitet werden.

1. Die **Kanalliste** definiert, welche Digital-Ports bzw. Analog-Kanäle - und gegebenenfalls weitere kanalspezifischen Parameter - eingestellt werden sollen. Siehe auch Seite 84.
2. Die **Triggerstruktur** definiert die Bedingungen wie eine Streaming-Operation gestartet und Werte ein- oder ausgegeben werden sollen. Siehe auch Seite 64. Wählen Sie aus einer der Betriebsarten:
 - **Stream-Timer** (siehe S. 69)
 - **Stream-Trigger-Sample** (siehe S. 71)
 - **Stream-Trigger-List** (siehe S. 73)

3.4.2.3 Kanalliste

Die Kanalliste (*meIOStreamConfig_t*) definiert, welche Digital-Ports bzw. Analog-Kanäle - und gegebenenfalls weitere kanalspezifischen Parameter - eingestellt werden sollen. Auch wenn keine kanalspezifischen Parameter auf der betreffenden Hardware eingestellt werden können, muss eine Kanalliste mit einem Standard-Eintrag geschrieben werden. Weisen Sie dazu einen Puffer definierter Größe zu, in das die Kanallisten-Einträge geschrieben werden. Die maximale Anzahl der Kanallisten-Einträge hängt von der Hardware ab (z. B. max. 1024 Einträge für die ME-4600-Serie). Dies kann mit Hilfe der *meQuery...* Funktionen abgefragt werden.

Typische Einstellungen in der Kanalliste:

- Kanalindex für Analog-Kanäle bzw. Subdevice-Index für Digital-Ein-/Ausgänge.
- Messbereich für analoge Kanäle.
- Massebezug: z. B. single-ended oder differentiell.

3.4.2.4 Triggerstruktur

Die Triggerstruktur (`meIOStreamTrigger_t`) definiert Bedingungen zum Starten und Stoppen einer Streaming-Operation.

Sie können damit z. B. Triggertyp (Software oder ext. Analog- bzw. Digitaltrigger) und Triggerflanke definieren. Je nach Gerät sind unterschiedliche Triggerbedingungen möglich.

Es gibt 3 hierarchische Triggerebenen und verschiedene Triggerbedingungen, die im Folgenden kurz erläutert werden:

Ebene 1: Betrifft die Operation als Ganzes (`iAcq...`)

Ebene 2: Abarbeitung einer Liste wie z. B. der Kanalliste bei analoger Erfassung (`iScan...`)

Ebene 3: Die Wandlung eines einzelnen Wertes (`iConv...`)

Start der gesamten Operation (<iAcqStart...>)		
	Start einer Listenabarbeitung (Scan) <iScanStart...>	
		Start einer einzelnen Abtastung/Wandlung <iConvStart...>
	Stop einer Listenabarbeitung (Scan) <iScanStop...>	
Stop der gesamten Operation (<iAcqStop...>)		

Tabelle 10: Triggerstruktur

Tip: Initialisieren Sie die Triggerstruktur zunächst mit 0. Damit brauchen Sie sich nur um die von Ihnen benötigten Parameter kümmern und zugleich werden nicht zutreffende bzw. nicht benötigte Parameter automatisch korrekt übergeben.

a. **Triggertyp** <`iAcqStartTrigType`>

Dieser Parameter definiert den Triggertyp für den Start der gesamten Operation. In Abhängigkeit von der verwendeten Hardware können Sie zwischen den Optionen Software-Start sowie externer Analog- oder

Digital-Trigger wählen (siehe auch Funktionsbeschreibung auf Seite 167)

b. Triggerflanke <iAcqStartTrigEdge>

Dieser Parameter definiert die Triggerflanke für den Start einer einzelnen Wandlung. In Abhängigkeit von Triggertyp und verwendeter Hardware können Sie aus verschiedenen Optionen für die Triggerflanke (steigend, fallend, etc.) wählen (siehe auch Funktionsbeschreibung auf Seite 168).

c. Triggerkanal <iAcqStartTrigChan>

Mit diesem Parameter können Sie wählen, ob die Triggerung für jeden Kanal getrennt erfolgen soll (Standard) oder ob ein Kanal synchron mit anderen Kanälen gestartet werden soll, z. B. für analoge Erfassung mit Sample & Hold-Option oder Synchron-Start mehrerer D/A-Kanäle (siehe auch Funktionsbeschreibung auf Seite 169).

d. Offsetzeit <iAcqStartTicksLow>, <iAcqStartTicksHigh>

Anzahl der Ticks zwischen Start der Operation und der ersten Wandlung. Beachten Sie, dass die Einschwingzeit des AI-Teils nicht unterschritten werden darf. Für Standardanwendungen empfehlen wir die Offsetzeit mit dem minimalen Chan-Intervall der betreffenden Hardware gleichzusetzen (AcqStartTicks = min. ConvStartTicks). Siehe auch Funktionsbeschreibung auf Seite 169)

Sofern von der Hardware unterstützt, sind durch Kombination von „AcqStartTicksLow“ und „AcqStartTicksHigh“ bis zu 64 bit breite Werte möglich (hardwareabhängig). Es gilt:

$$\text{AcqStartTicks} = (\text{AcqStartTicksHigh} \ll 32) \vee \text{AcqStartTicksLow}.$$

e. AcqStartArgs <iAcqStartArgs>

Dieser Parameter wird für erweiterte Triggeroptionen des MEphistoScope wie z. B. Fenstertrigger oder Trigger auf Flankensteilheit (Slope) verwendet. Siehe auch Funktionsbeschreibung auf Seite 170.

f. Triggertyp <iScanStartTrigType>

Dieser Parameter definiert den Triggertyp für den Start eines Scans. In Abhängigkeit von der verwendeten Hardware können Sie aus zahlreichen Optionen wie externer Analog- oder Digital-Trigger, timer-gesteuert, u. a. wählen. Siehe auch Funktionsbeschreibung auf Seite 170.

g. Scan-Intervall

`<iScanStartTicksLow>, <iScanStartTicksHigh>`

Dieser Parameter bestimmt den Zeitintervall zwischen dem Start zwei aufeinander folgender Scans (= Kanallistenabarbeitungen). Die Verwendung ist optional. Siehe auch Funktionsbeschreibung auf Seite 171.

Sofern von der Hardware unterstützt, sind durch Kombination von „ScanStartTicksLow“ und „ScanStartTicksHigh“ bis zu 64 bit breite Werte möglich (hardwareabhängig). Es gilt:

$\text{ScanStartTicks} = (\text{ScanStartTicksHigh} \ll 32) \vee \text{ScanStartTicksLow}$.

Beachten Sie bei der Berechnung des Scan-Intervalls folgenden Zusammenhang (siehe auch Timing-Diagramme ab Seite 69):

$\text{ScanStartTicks} = (\text{Anzahl der Kanallisteneinträge} \times \text{ConvStartTicks}) + \text{„Pause“ [Ticks]}$

h. ScanStartArgs <iScanStartArgs>

Dieser Parameter ist reserviert für zukünftige Erweiterungen.

i. Triggertyp <iConvStartTrigType>

Dieser Parameter definiert den Triggertyp für den Start einer einzelnen Wandlung. In Abhängigkeit von der verwendeten Hardware können Sie zwischen den Optionen externer Analog- oder Digital-Trigger, timer-gesteuert, u. a. wählen. Siehe auch Funktionsbeschreibung auf Seite 171.

j. Chan-Intervall

`<iConvStartTicksLow>, <iConvStartTicksHigh>`

Dieser Parameter bestimmt den Chan-Intervall in Anzahl der Ticks zwischen zwei Wandlungen (Abtast- bzw. Ausgaberate). Siehe auch Funktionsbeschreibung auf Seite 171.

Sofern von der Hardware unterstützt, sind durch Kombination von „ConvStartTicksLow“ und „ConvStartTicksHigh“ bis zu 64 bit breite Werte möglich (hardwareabhängig). Es gilt:

$\text{ConvStartTicks} = (\text{ConvStartTicksHigh} \ll 32) \vee \text{ConvStartTicksLow}$.

k. ConvStartArgs <iConvStartArgs>

Dieser Parameter ist reserviert für zukünftige Erweiterungen.

l. Triggertyp <iScanStopTrigType>

Dieser Parameter definiert den Triggertyp für das Ende eines Scans. In Abhängigkeit von der verwendeten Hardware können Sie z. B. nach der in <iScanStopCount> definierten Gesamtzahl an Wandlungen den Scan-Betrieb beenden. Siehe auch Funktionsbeschreibung auf Seite 172.

m. Anzahl Wandlungen <iScanStopCount>

Dieser Parameter bestimmt die Gesamtzahl der Wandlungen nach denen der Scan-Betrieb beendet wird und damit auch die Operation als Ganzes. Wenn Sie die Operation auf unbestimmte Zeit laufen lassen wollen übergeben Sie hier 0. Siehe auch Funktionsbeschreibung auf Seite 172.

n. ScanStopArgs <iScanStopArgs>

Dieser Parameter wird für erweiterte Triggeroptionen des MEphisto-Scope verwendet. Siehe auch Funktionsbeschreibung auf Seite 172.

o. Triggertyp <iAcqStopTrigType>

Mit diesem Parameter können Sie bei Bedarf den Triggertyp für das Ende der gesamten Operation definieren. Folgende Optionen stehen zur Verfügung. Siehe auch Funktionsbeschreibung auf Seite 172:

- Die Operation wird beendet, nach der in <iAcqStopCount> definierten Anzahl an „Scans“ (Kanallisten-Abarbeitungen).
- Die Operation wird beendet, sobald die in <iScanStopCount> vorgegebene Anzahl an Wandlungen erreicht wurde.

p. Anzahl Scans <iAcqStopCount>

Dieser Parameter bestimmt die Anzahl der „Scans“ (Kanallisten-Abarbeitungen), nach denen die gesamte Operation beendet wird.

Wenn Sie die Operation auf unbestimmte Zeit laufen lassen wollen übergeben Sie hier 0. Siehe auch Funktionsbeschreibung auf Seite 173.

q. AcqStopArgs <iAcqStopArgs>

Dieser Parameter ist reserviert für zukünftige Erweiterungen.

Hinweise:

In den folgenden Abbildungen (Seite 69 bis 73) ist nur die steigende Triggerflanke aktiv sofern der externe Trigger verwendet wird.

Siehe auch Kapitel „Synchronstart“ auf Seite 88.

Die Parameter <iAcqStartTrigType>, <iScanStartTrigType> und <iConvStartTrigType> definieren alle Start-Bedingungen, die möglich sind.

Für die Übergabe der konkreten Werte für das betreffende Subdevice schauen Sie bitte in der ME-iDS Hilfedatei nach.

Wählen Sie aus einer der Betriebsarten und beachten Sie die Abbildungen auf den folgenden Seiten:

- **Stream-Timer** (siehe S. 69)
- **Stream-Trigger-Sample** (siehe S. 71)
- **Stream-Trigger-List** (siehe S. 73)

3.4.2.4.1 Timing Stream-Timer

Timergesteuerter Streaming-Betrieb (mit/ohne Scan-Timer). Start per Software oder externem Triggerimpuls nach Aufruf der Funktion *meIOStreamStart()* - alle weiteren Triggerimpulse werden ignoriert.

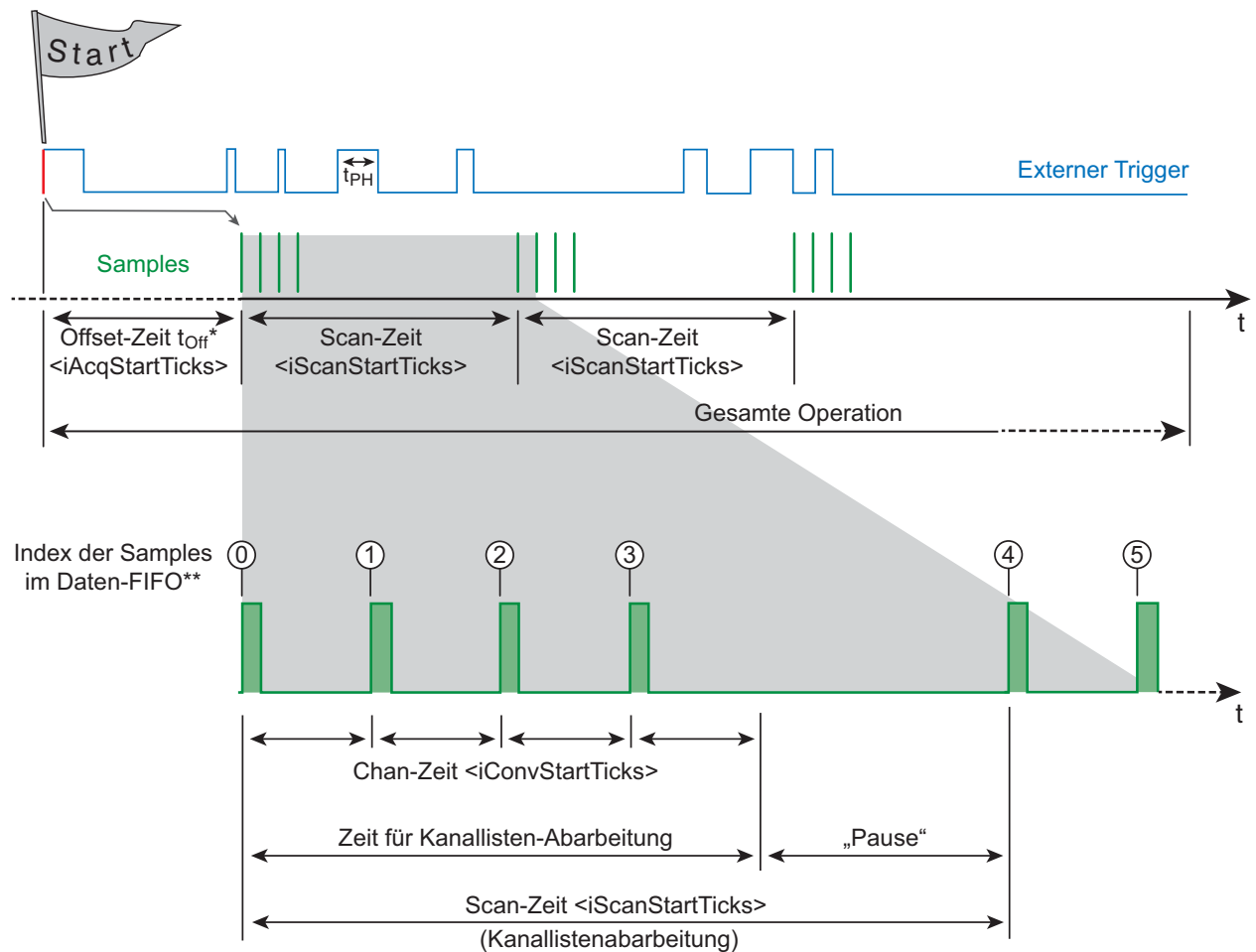


Abbildung 18: Timing-Diagramm Stream-Timer

- * Beachten Sie, dass die Offset-Zeit t_{Off} die Einschwingzeit der Hardware berücksichtigen muss! Für Standardanwendungen empfehlen wir die Offset-Zeit mit der minimalen Chan-Zeit der betreffenden Hardware gleichzusetzen ($\langle iAcqStartTicks \rangle = \min. \langle iConvStartTicks \rangle$).
- ** Index der Samples im Daten-FIFO. Die Werte werden stets gemäß der Reihenfolge der Kanäle in der Kanalliste gesampelt. Unabhängig davon, ob kanalspezifische Parameter auf der betreffenden Hardware eingestellt werden können, muss eine Kanalliste mit einem Standard-Eintrag geschrieben werden. Siehe Funktion *meIOStreamConfig()*.

Parameter	Software-Trigger	Ext. Trigger
<i>...ohne Scan-Pause</i>		
<iAcqStartTrigType>	ME_TRIG_TYPE_SW	ME_TRIG_TYPE_EXT_XX
<iAcqStartTrigEdge>	ME_TRIG_EDGE_NONE	Flanke erforderlich
<iScanStartTrigType>	ME_TRIG_TYPE_FOLLOW	ME_TRIG_TYPE_FOLLOW
<iScanStartTicks>	0	0
<iConvStartTrigType>	ME_TRIG_TYPE_TIMER	ME_TRIG_TYPE_TIMER
<i>...mit Scan-Pause</i>		
<iAcqStartTrigType>*	ME_TRIG_TYPE_SW	ME_TRIG_TYPE_EXT_XX
<iAcqStartTrigEdge>	ME_TRIG_EDGE_NONE	Flanke erforderlich
<iScanStartTrigType>	ME_TRIG_TYPE_TIMER	ME_TRIG_TYPE_TIMER
<iScanStartTicks>	Zeit für Kanallisten-Abarbeitung + Pause	
<iConvStartTrigType>	ME_TRIG_TYPE_TIMER	ME_TRIG_TYPE_TIMER

Tabelle 11: Timing Stream-Timer

Hinweise zu Stream-Timer:

Um einen ext. Triggerimpuls zuverlässig zu erfassen, muss die High-Phase des ext. Triggerimpulses t_{PH} mindestens 1 Tick betragen (hardware-abhängig). Siehe auch Spezifikation Ihres Gerätes.

Je nach Gerät sind weitere Triggerbedingungen zum Start der Operation möglich. Siehe auch Parameter <iAcqStartTrigType> der Funktion *meIOStreamConfig()*.

3.4.2.4.2 Timing Stream-Trigger-Sample

Ereignisgesteuerter Streaming-Betrieb. Start per Software oder externem Triggerimpuls nach Aufruf der Funktion *meIOStreamStart()* - je Triggerereignis wird ein Wert gesampelt.

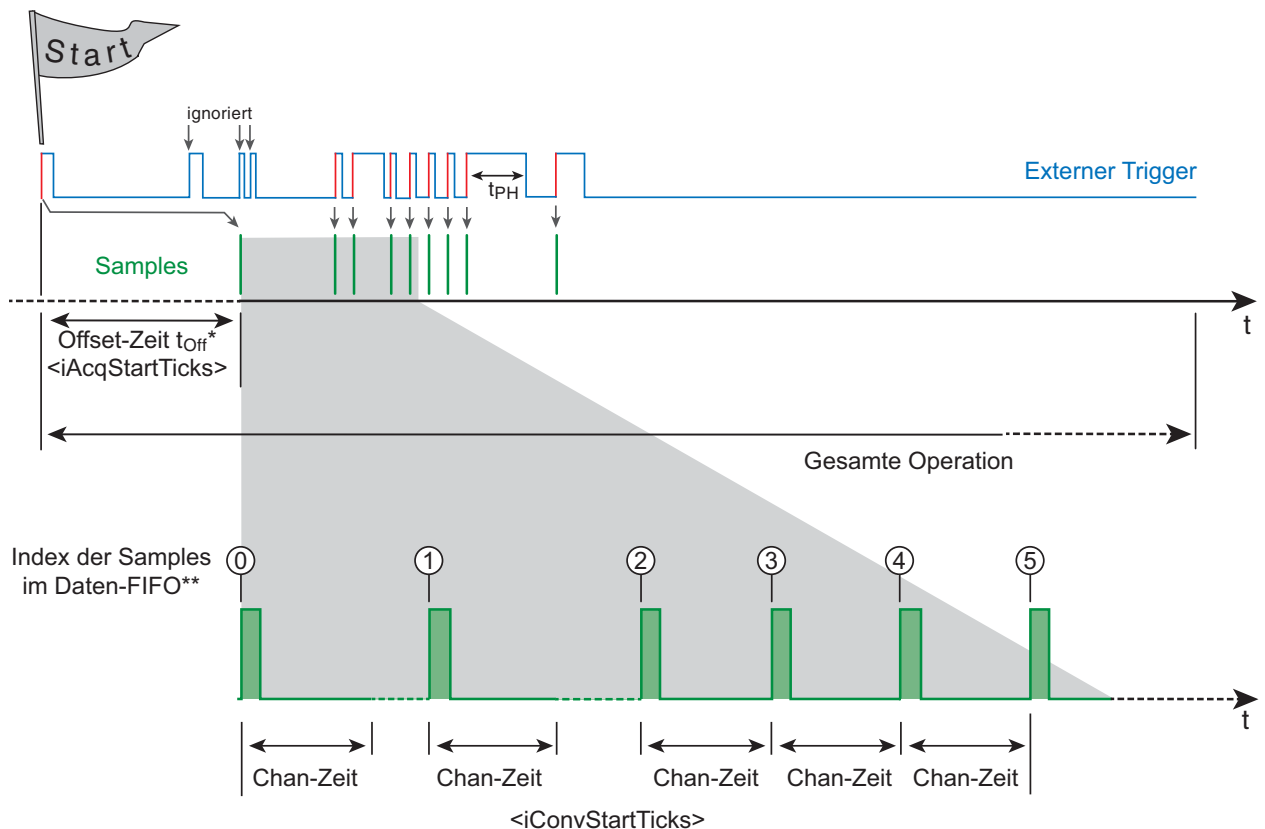


Abbildung 19: Timing-Diagramm Stream-Trigger-Sample

- * Beachten Sie, dass die Offset-Zeit t_{Off} die Einschwingzeit der Hardware berücksichtigen muss! Für Standardanwendungen empfehlen wir die Offset-Zeit mit der minimalen Chan-Zeit der betreffenden Hardware gleichzusetzen ($\text{<iAcqStartTicks>} = \min. \text{<iConvStartTicks>}$).
- ** Index der Samples im Daten-FIFO. Die Werte werden stets gemäß der Reihenfolge der Kanäle in der Kanalliste gesampelt. Unabhängig davon, ob kanalspezifische Parameter auf der betreffenden Hardware eingestellt werden können, muss eine Kanalliste mit einem Standard-Eintrag geschrieben werden. Siehe Funktion *meIOStreamConfig()*.

Parameter	Software-Trigger	Ext. Trigger
<iAcqStartTrigType>	ME_TRIG_TYPE_SW	ME_TRIG_TYPE_EXT_XX
<iAcqStartTrigEdge>	ME_TRIG_EDGE_NONE	Flanke erforderlich
<iScanStartTrigType>	ME_TRIG_TYPE_SW	<iAcqStartTrigType>
<iScanStartTrigEdge>	ME_TRIG_EDGE_NONE	<iAcqStartTrigEdge>
<iConvStartTrigType>	ME_TRIG_TYPE_SW	<iAcqStartTrigType>
<iConvStartTrigEdge>	ME_TRIG_EDGE_NONE	<iAcqStartTrigEdge>

Tabelle 12: Timing Stream-Trigger-Sample

Hinweise zu Stream-Trigger-Sample:

Um einen ext. Triggerimpuls zuverlässig zu erfassen, muss die High-Phase des ext. Triggerimpulses t_{PH} mindestens 1 Tick betragen (hardware-abhängig). Siehe auch Spezifikation Ihres Gerätes.

Je nach Gerät sind weitere Triggerbedingungen zum Start der Operation möglich. Siehe auch Parameter <iAcqStartTrigType> der Funktion *meIOStreamConfig()*.

In dieser Betriebsart wird eine einstellbare Chan-Zeit von manchen Geräten nicht unterstützt. Details entnehmen Sie bitte der ME-iDS-Hilfedatei und dem Hardware-Handbuch.

3.4.2.4.3 Timing Stream-Trigger-List

Ereignisgesteuerter Streaming-Betrieb. Start per Software oder externem Triggerimpuls nach Aufruf der Funktion *meIOStreamStart()* - je Triggerimpuls wird einmal die Kanalliste gesampelt.

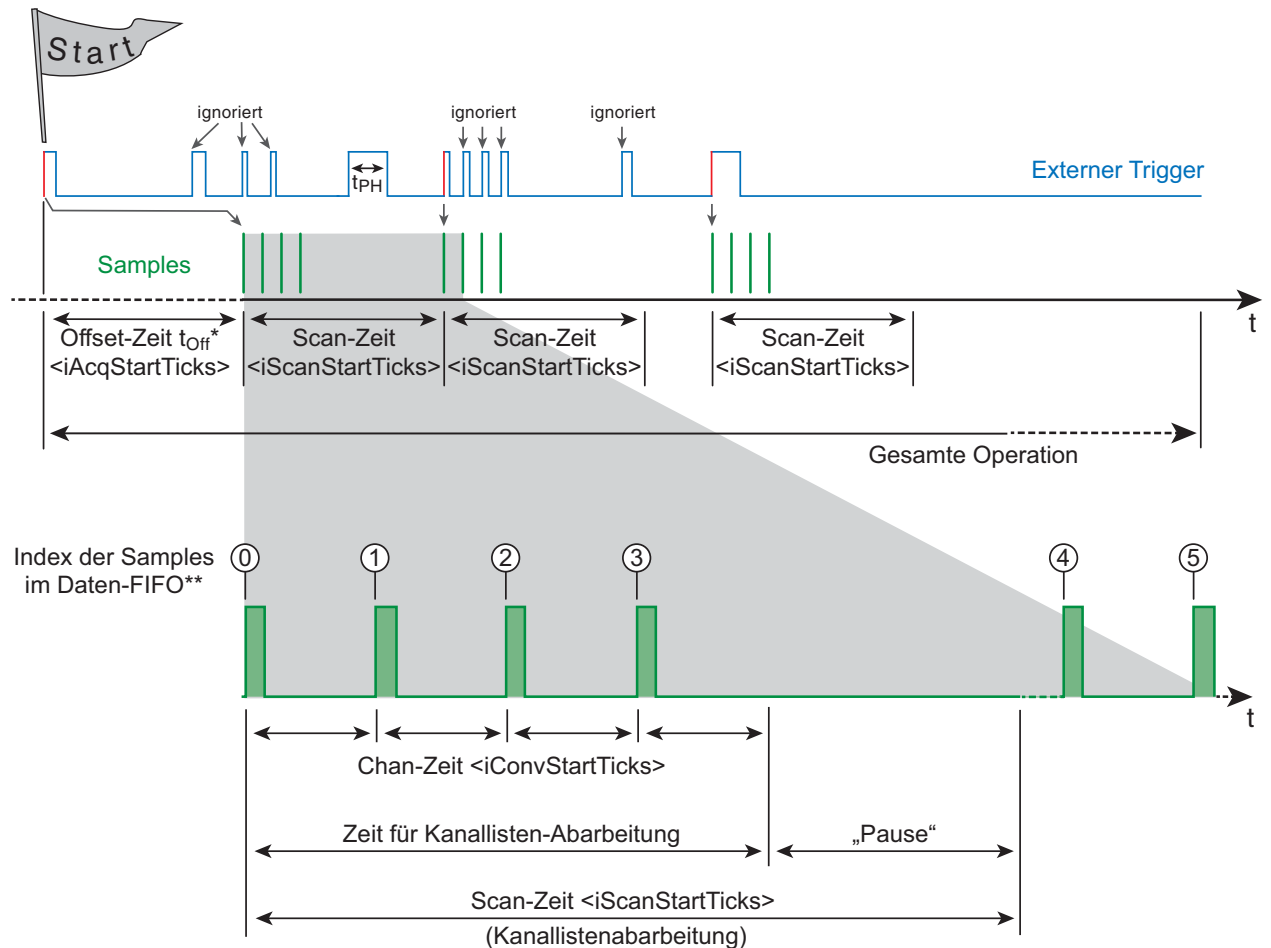


Abbildung 20: Timing-Diagramm Stream-Trigger-List

- * Beachten Sie, dass die Offset-Zeit t_{Off} die Einschwingzeit der Hardware berücksichtigen muss! Für Standardanwendungen empfehlen wir die Offset-Zeit mit der minimalen Chan-Zeit der betreffenden Hardware gleichzusetzen ($\langle iAcqStartTicks \rangle = \min. \langle iConvStartTicks \rangle$).
- ** Index der Samples im Daten-FIFO. Die Werte werden stets gemäß der Reihenfolge der Kanäle in der Kanalliste gesampelt. Unabhängig davon, ob kanalspezifische Parameter auf der betreffenden Hardware eingestellt werden können, muss eine Kanalliste mit einem Standard-Eintrag geschrieben werden. Siehe Funktion *meIOStreamConfig()*.

Parameter	Software-Trigger	Ext. Trigger
<iAcqStartTrigType>	ME_TRIG_TYPE_SW	ME_TRIG_TYPE_EXT_XX
<iAcqStartTrigEdge>	ME_TRIG_EDGE_NONE	Flanke erforderlich
<iScanStartTrigType>	ME_TRIG_TYPE_SW	ME_TRIG_TYPE_EXT_XX
<iScanStartTrigEdge>	ME_TRIG_EDGE_NONE	<iAcqStartTrigEdge>
<iConvStartTrigType>	ME_TRIG_TYPE_TIMER	ME_TRIG_TYPE_TIMER
<iConvStartTrigEdge>	ME_TRIG_EDGE_NONE	ME_TRIG_EDGE_NONE

Tabelle 13: Timing „Stream-Trigger-List“

Hinweise zu Stream-Trigger-List:

Um einen ext. Triggerimpuls zuverlässig zu erfassen, muss die High-Phase des ext. Triggerimpulses t_{PH} mindestens 1 Tick betragen (hardware-abhängig). Siehe auch Spezifikation Ihres Gerätes.

Je nach Gerät sind weitere Triggerbedingungen zum Start der Operation möglich. Siehe auch Parameter <iAcqStartTrigType> der Funktion *meIOStreamConfig()*.

In dieser Betriebsart wird die „Pause“ zwischen den Kanallisten-Abarbeitungen von manchen Geräten nicht unterstützt. Details entnehmen Sie bitte dem Hardware-Handbuch.

3.4.2.5 Daten lesen

Einlesen der Daten durch wiederholtes Aufrufen der Funktion *meIOStreamRead()* (siehe Seite 185). Die folgende Abbildung gilt für analoge Erfassung. Die Verfügbarkeit der gezeigten Blöcke ist hardware-abhängig.

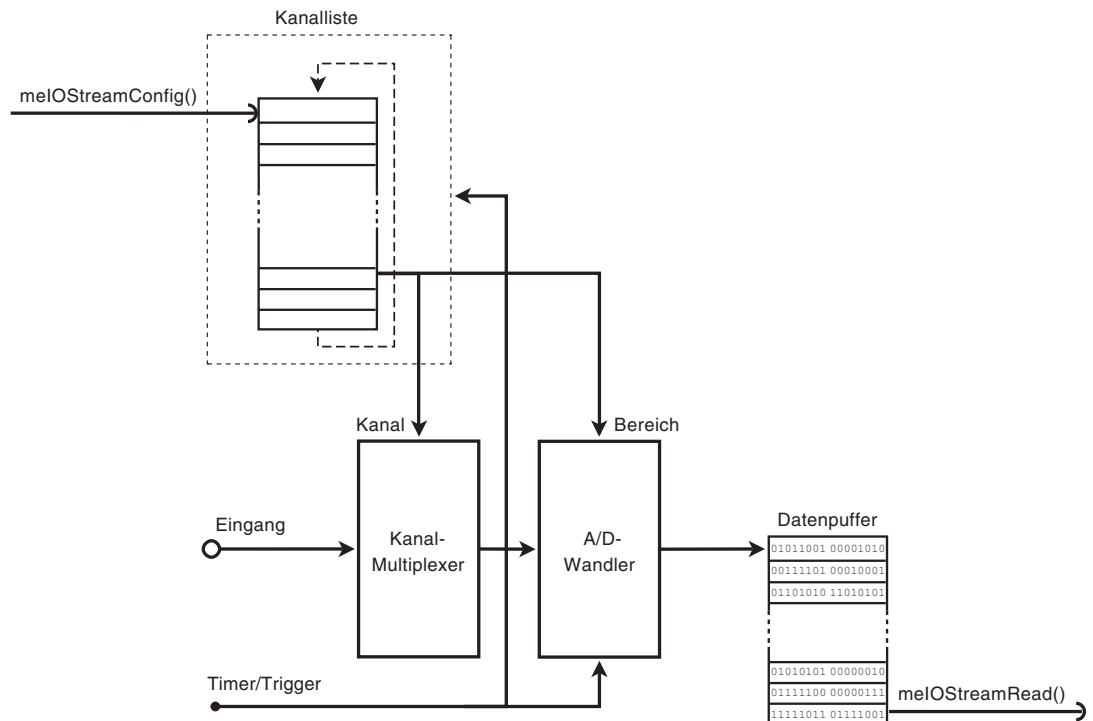


Abbildung 21: Daten lesen

3.4.2.5.1 Vorgehensweise Daten lesen

Das „Abholen“ der Daten erfolgt durch wiederholten Aufruf der Funktion *meIOStreamRead()* (siehe Seite 179).

Die Diagramme auf den nächsten Seiten beschreiben den Programm-Fluss unter folgenden Bedingungen:

- Das „Abholen“ der Daten ohne Callback-Funktion (BLOCKING oder NONBLOCKING).
Siehe Abbildung 23 auf Seite .77.
- Das „Abholen“ der Daten erfolgt im Hintergrund im Rahmen einer benutzerdefinierten Callback-Funktion. Bei Bedarf können Sie mit der Funktion *meIOStreamSetCallbacks()* drei verschiedene Callback-Funktionen installieren, die beim Start, während oder nach der Erfassung aufgerufen werden. Siehe Abbildung 24 auf Seite .78.

Beachten Sie auch die Programmierbeispiele im ME-iDS SDK.

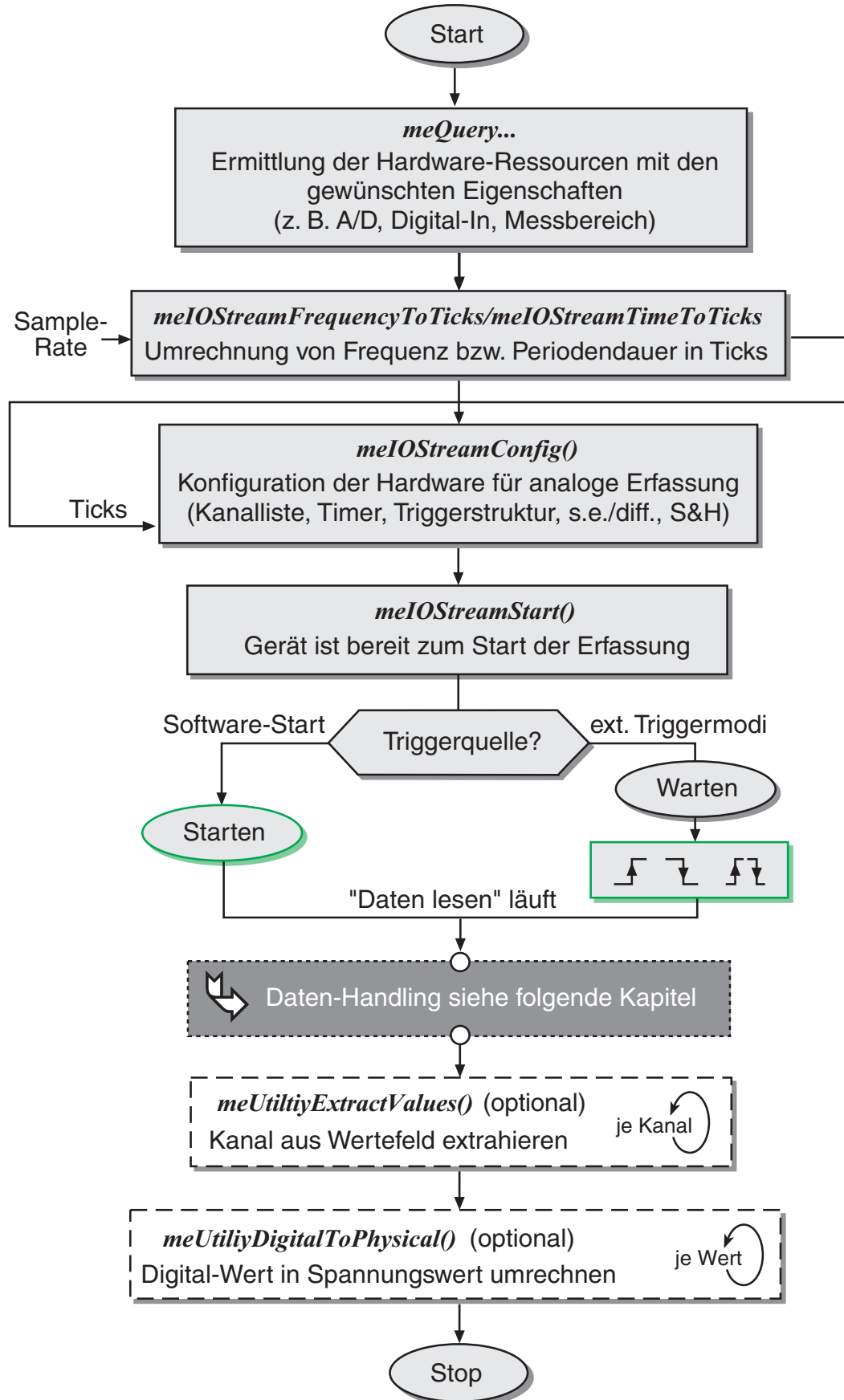


Abb. 22: Vorgehensweise Programmierung Daten lesen

3.4.2.5.2 Lesen ohne Callback-Funktion

Daten lesen durch wiederholten Aufruf der Funktion *meIOStreamRead()* (<iReadMode>: BLOCKING oder NONBLOCKING):

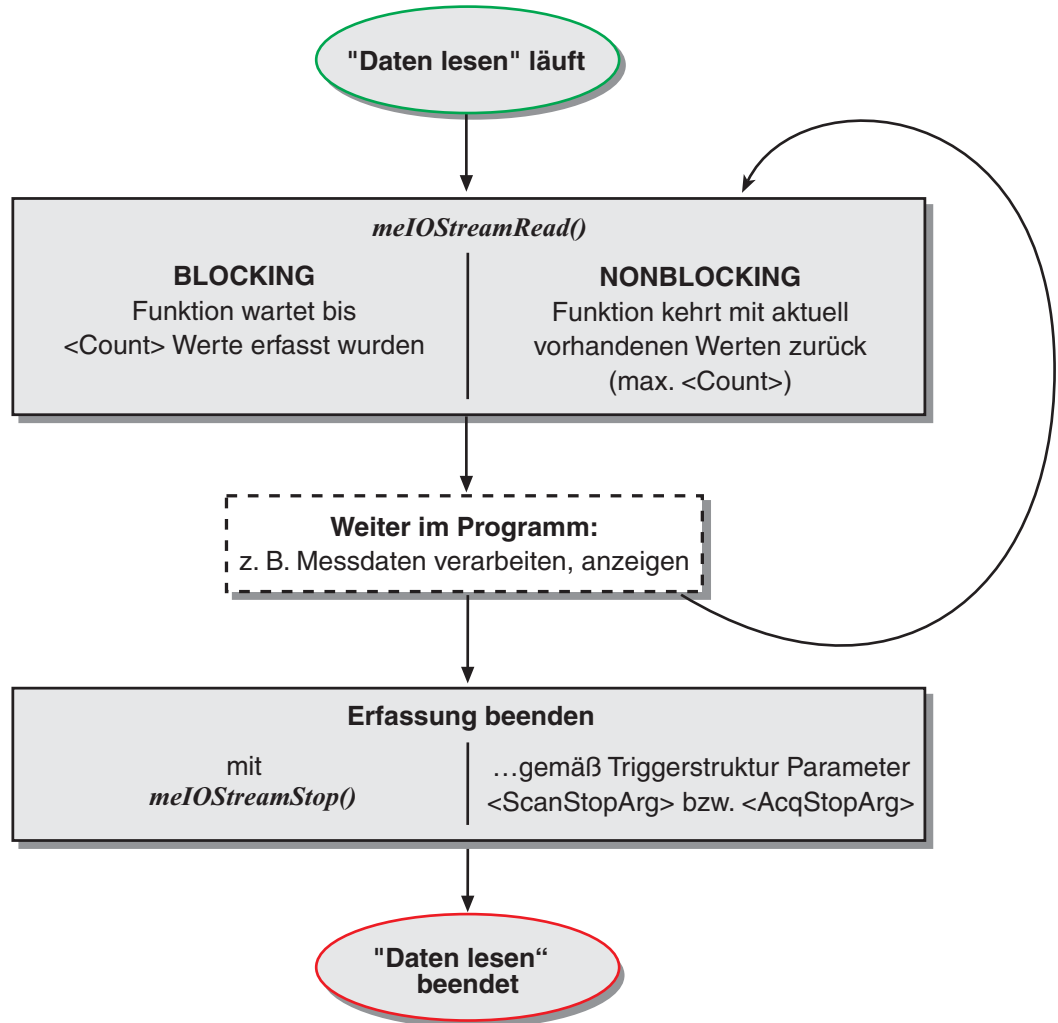


Abb. 23: Daten lesen ohne Callback-Funktion

3.4.2.5.3 Lesen mit Callback-Funktion

Daten lesen mit Hilfe einer benutzerdefinierten Callback-Funktion:

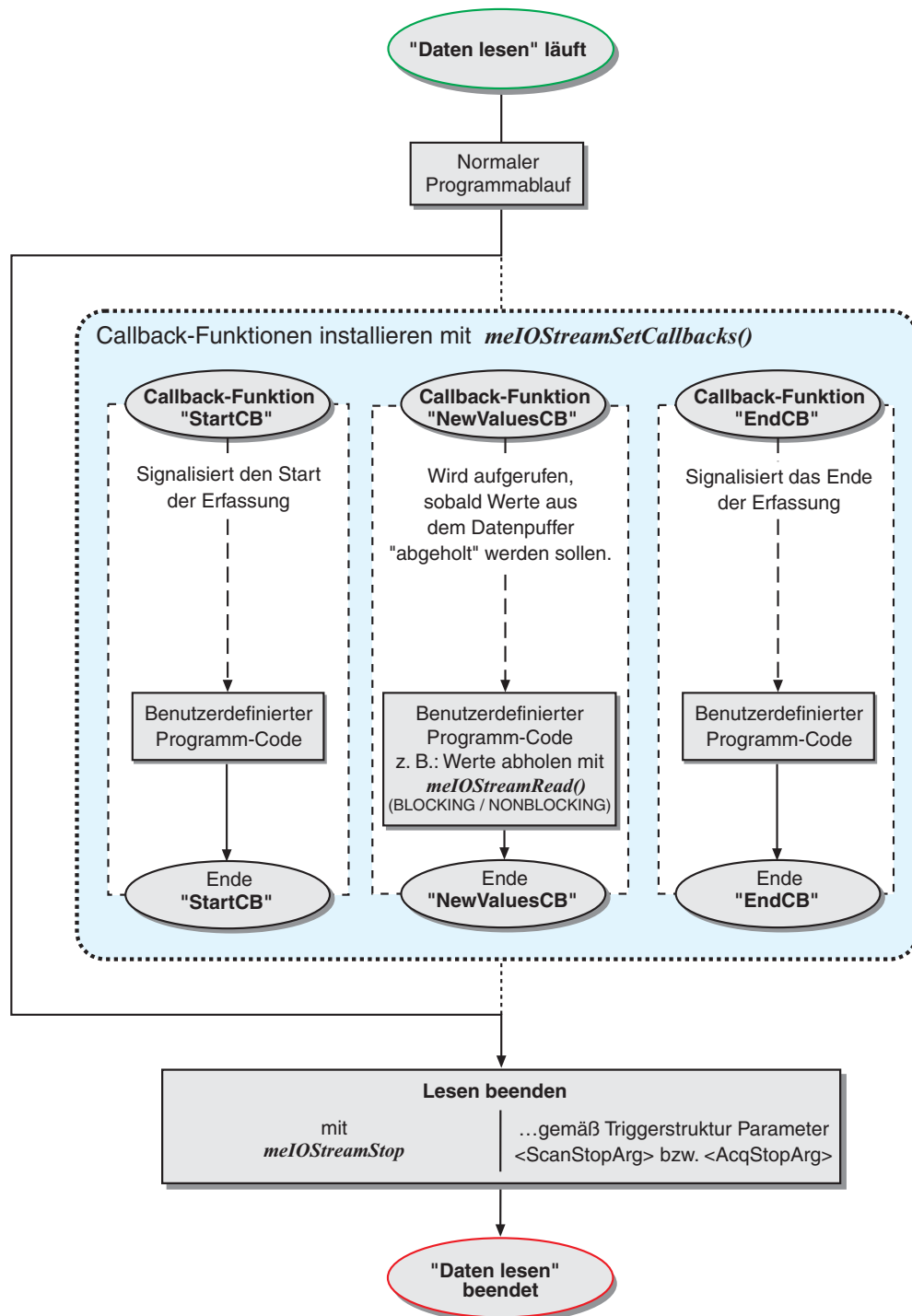


Abb. 24: Daten lesen mit Callback-Funktion

Bei Rückgabe eines Wertes ungleich 0 durch die Callback-Funktion wird der Streaming-Betrieb beendet. Sie können den Fehlercode der Callback-Funktion aus dem Parameter `<iErrorCode>` zurückgeben.

3.4.2.6 Daten schreiben

Schreiben der Daten durch wiederholtes Aufrufen der Funktion *meIOStreamWrite()* (siehe Seite 188). Die folgende Abbildung gilt für analoge Ausgabe. Die Verfügbarkeit der gezeigten Blöcke ist hardware-abhängig.

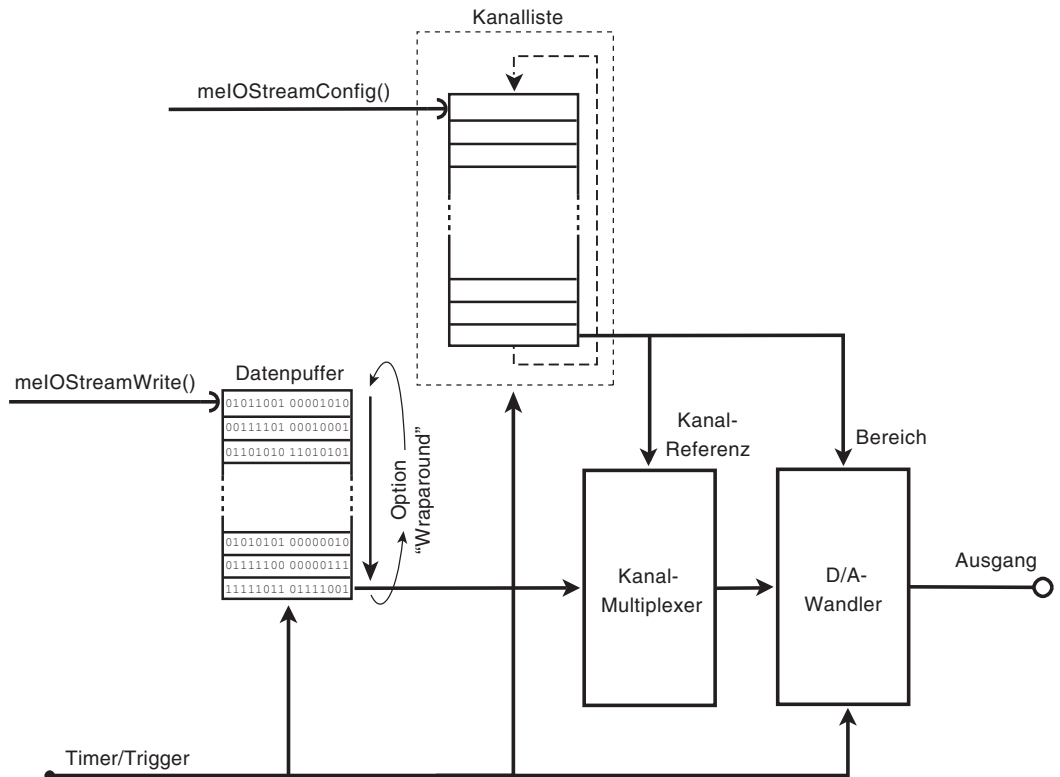


Abbildung 25: Daten schreiben

Grundsätzlich werden die Daten kontinuierlich in ein Subdevice vom Untertyp `ME_SUBTYPE_STREAMING` geschrieben. Die Applikation ist verantwortlich für das Nachladen des Datenpuffers mit neuen Werten mit Hilfe der Funktion *meIOStreamWrite()*. Dies bietet Ihnen die Möglichkeit, die Ausgabewerte während des Betriebs fortwährend zu ändern.

Allokieren sie einen Datenpuffer definierter Größe für die auszugebenden Werte. Vor Beginn der Ausgabe-Operation müssen Sie mit der Funktion *meIOStreamWrite()* das erste Wertepaket unter Verwendung der Option `ME_IO_WRITE_MODE_PRELOAD` im Parameter `<iWriteMode>` in den Datenpuffer schreiben.

Der Chan-Timer gibt ein festes Zeitraster für die Ausgabe der Werte vor. Er muss mit der Funktion *meIOStreamConfig()* vor Start der Ausgabe konfiguriert werden.

Siehe auch **Wraparound-Option** auf Seite 84.

Hinweis:

Zu den Besonderheiten der timergesteuerten Bitmuster-Ausgabe der ME-4680 beachten Sie bitte die detaillierte Beschreibung in Anhang B3 auf Seite 228.

3.4.2.6.1 Vorgehensweise Daten schreiben

Die Diagramme auf den nächsten Seiten beschreiben den Programm-Fluss unter folgenden Bedingungen:

- a. Die Ausgabe erfolgt ohne Callback-Funktion
Im Modus „NONBLOCKING“ wird die Anzahl an Werten „nachgefüllt“, die aktuell im Datenpuffer Platz finden. Im Modus „BLOCKING“ wartet (blockiert) die Funktion bis die Anzahl der im Parameter `<piCount>` spezifizierten Werte, nachgeladen werden konnte. Siehe Abbildung 27 auf Seite .82.
- b. Die Ausgabe erfolgt im Hintergrund im Rahmen einer benutzerdefinierten Callback-Funktion. Bei Bedarf können Sie mit der Funktion `meIOStreamSetCallbacks()` drei verschiedene Callback-Funktionen installieren, die beim Start, während oder nach der Ausgabe aufgerufen werden. Siehe Abbildung 28 auf Seite .83.

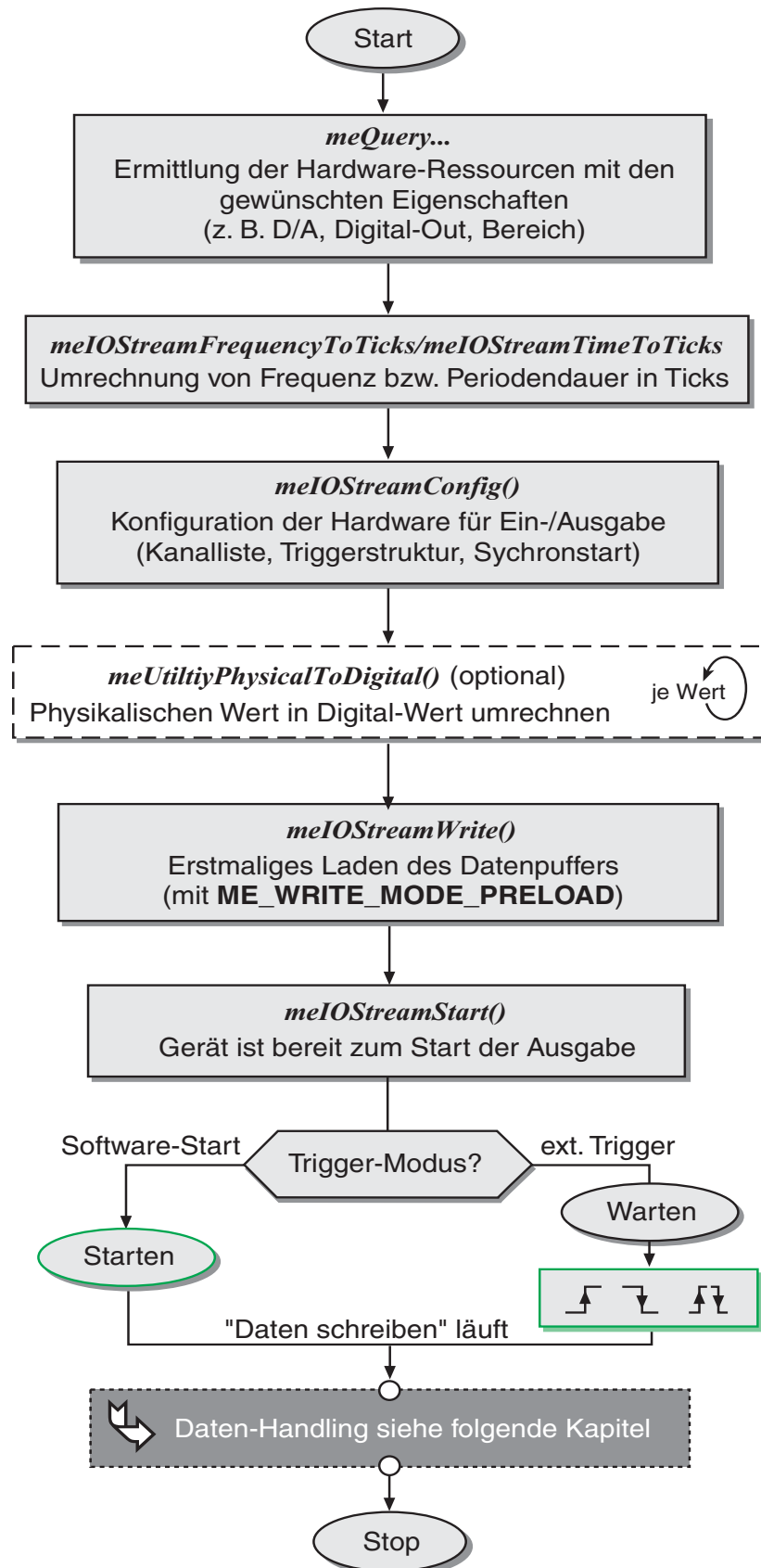


Abb. 26: Vorgehensweise Programmierung Daten schreiben

3.4.2.6.2 Schreiben ohne Callback-Funktion

Daten schreiben durch wiederholten Aufruf der Funktion *meIOStreamWrite* (<iWriteMode>: BLOCKING oder NONBLOCKING):

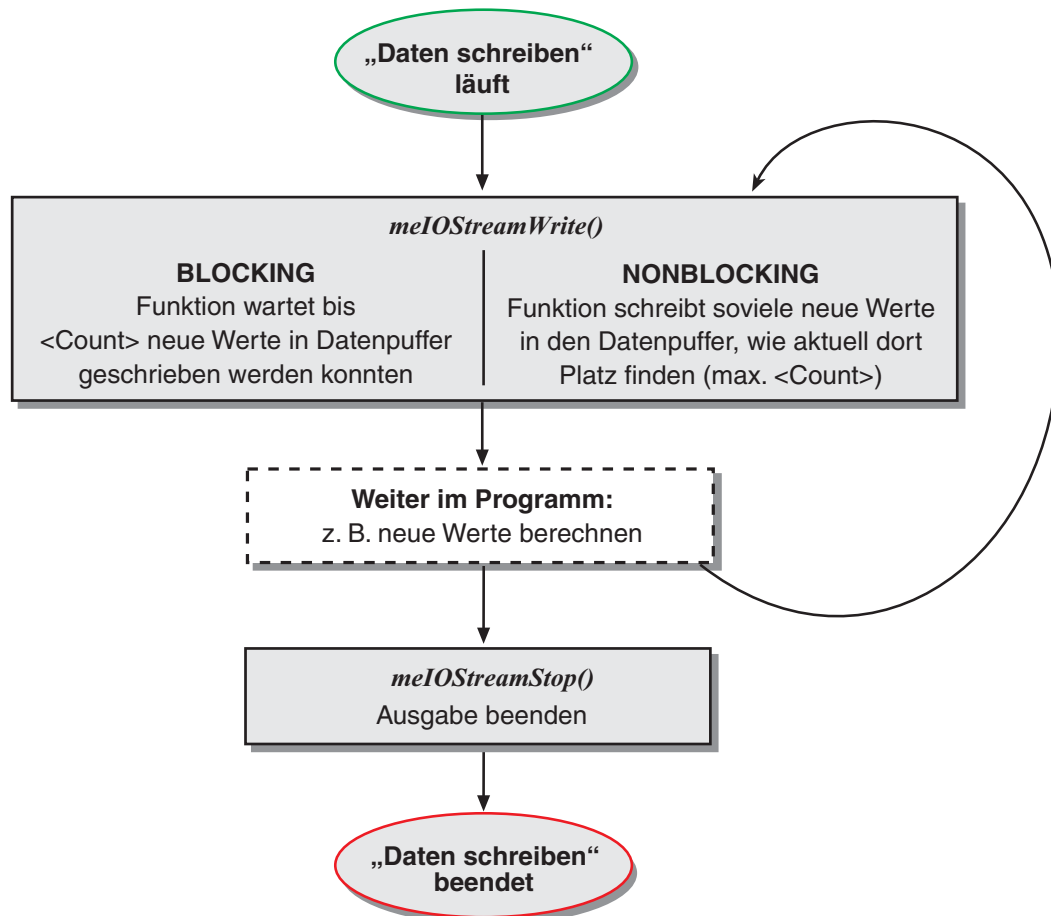


Abb. 27: Daten schreiben ohne Callback-Funktion

3.4.2.6.3 Schreiben mit Callback-Funktion

Daten schreiben mit Hilfe einer benutzerdefinierten Callback-Funktion:

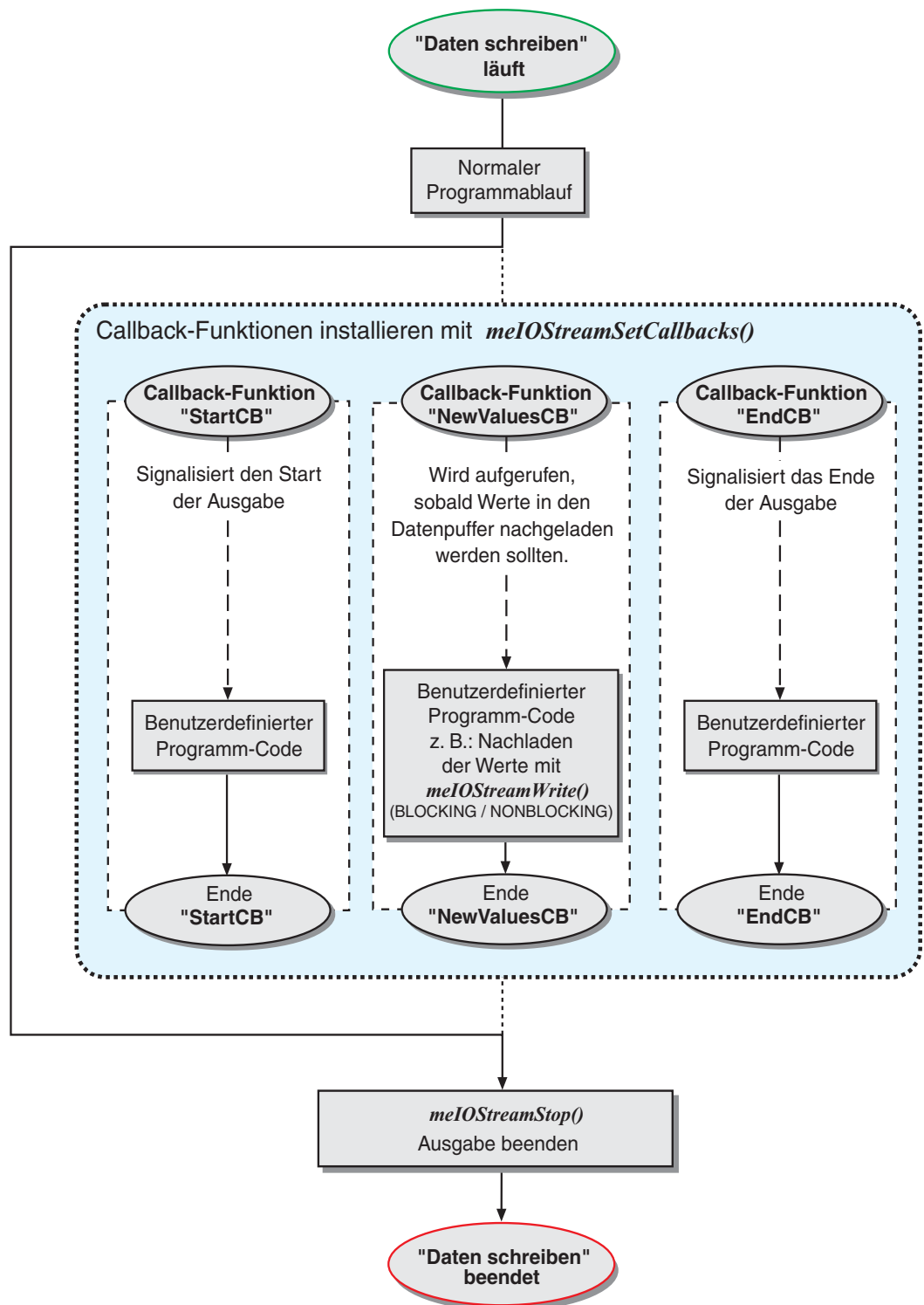


Abb. 28: Daten schreiben mit Callback-Funktion

Bei Rückgabe eines Wertes ungleich 0 durch die Callback-Funktion wird der Streaming-Betrieb beendet. Im einfachsten Fall geben Sie den

Fehlercode der Callback-Funktion aus dem Parameter `<iErrorCode>` wieder zurück.

3.4.2.6.4 Wraparound-Option

Mit der „Wraparound“-Option werden die Ausgabe-Werte einmalig in den Datenpuffer geschrieben und periodisch ausgegeben. Es besteht keine Möglichkeit die Werte zu aktualisieren, nachdem die Operation mit *meIOStreamStart()* gestartet wurde.

In Abhängigkeit von verschiedenen hardware-spezifischen Parametern (z. B.: FIFO-Größe, Abtastrate) wird die Operation auf Firmware-Ebene ausgeführt ohne Belastung für den Host-Computer.

Bevor die Ausgabe startet, muss das erste Datenpaket in den Datenpuffer geschrieben werden. Verwenden sie dazu die Option `ME_IO_WRITE_MODE_PRELOAD` im Parameter `<iWriteMode>` der Funktion *meIOStreamWrite()*.

3.4.2.7 Streaming-Betrieb stoppen

Die Operation wird beendet sobald eine der folgenden Bedingungen erfüllt ist:

1. Abbruch durch den Anwender mit *meIOStreamStop()* oder *meIOReset...()*
2. Beendet, nachdem die Anzahl der Werte ausgegeben wurde
3. Funktion *meIOStreamConfig()*: Die Triggerstruktur `meIOStreamTrigger_t` enthält mehrere Parameter zum Beenden einer Streaming-Operation:
 - a. Manueller Stop (im Falle eines unendlichen Betriebs)
 - `<iScanStopTrigType> = ME_TRIG_TYPE_NONE`
 - `<iScanStopCount> = 0`
 - `<iAcqStopTrigType> = ME_TRIG_TYPE_NONE`
 - `<iAcqStopCount> = 0`

- b. Stop nach einer definierten Anzahl an Wandlungen
 - `<iScanStopTrigType> = ME_TRIG_TYPE_COUNT`
 - `<iAcqStopTrigType> = ME_TRIG_TYPE_FOLLOW`
 - `<iAcqStopCount> = 0`
 - c. Stop nach einer definierten Anzahl an Kanallisten-Abarbeitungen
 - `<iScanStopTrigType> = ME_TRIG_TYPE_NONE`
 - `<iScanStopCount> = 0`
 - `<iAcqStopTrigType> = ME_TRIG_TYPE_COUNT`
4. Fehler aufgetreten: kein Platz im Datenpuffer zum Nachladen bzw. keine Werte im Datenpuffer zum Abholen.

3.4.3 Spezielle Features

3.4.3.1 Sample & Hold

Die „Sample & Hold“-Option wird verwendet, wenn mehrere Kanäle gleichzeitig abgetastet werden sollen. Die „Sample & Hold“-Kanäle werden durch ein gemeinsames Triggersignal (per Software oder durch externes Triggersignal) auf der Hardware simultan „eingefroren“. Anschließend können die Werte sequentiell eingelesen werden.

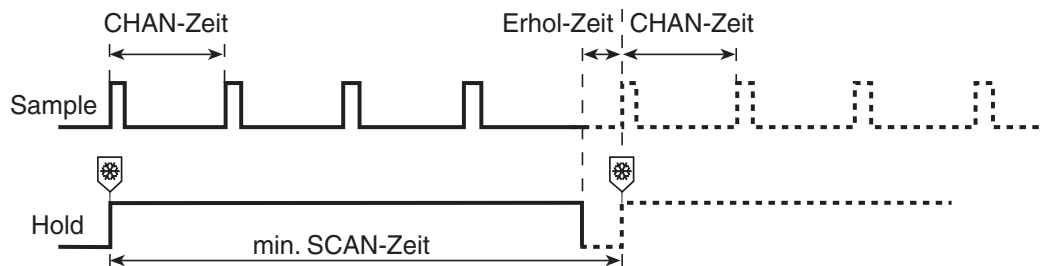


Abbildung 29: Sample & Hold

Hinweis: „Sample & Hold“-Betrieb benötigt eine gewisse Erholzeit, sodass die Abarbeitung der Kanalliste unmittelbar nach der Vorherigen nicht möglich ist. Diese Option ist nur für analoge Eingangskanäle verfügbar.

Falls die „Sample & Hold“-Option von der Hardware unterstützt wird gibt die Funktion `meQuerySubdeviceCaps()` `ME_CAPS_AI_SAMPLE_HOLD` zurück.

Weitere Details und Einschränkungen finden Sie in der ME-iDS-Hilfdatei und im jeweiligen Hardware-Handbuch.

3.4.3.2 Bitmuster-Ausgabe der ME-4680

Eine Sonderfunktion ist die Umlenkung eines D/A-FIFOs vom D/A-Wandler (DAC) auf digitale Ports, wie es für die „timergesteuerte Bitmuster-Ausgabe“ der ME-4680 erforderlich ist.

Einige der Daten-FIFOs für analoge Ausgabe können auf andere Subdevices (mit digitalem Ausgangsport) umgelenkt werden um einen digitalen Datenstrom auszugeben.

Die Programmierung erfolgt in 2 Schritten:

- **Abtrennung des DAC:** Dies wird in einem Subdevice für analoge Ausgabe vom Untertyp ME_SUBTYPE_STREAMING programmiert. Die Verbindung zum D/A-Wandler wird getrennt, wenn im Parameter `<iFlags>` der Funktion `meIOStreamConfig()` die Konstante ME_IO_STREAM_CONFIG_BIT_PATTERN verwendet wird.

Hinweis: Nach Trennung des D/A-Wandlers vom FIFO hält der analoge Ausgang seinen Spannungswert.

- **Umlenkung des FIFO-Ausgangs:** Dies wird in einem Subdevice vom Typ ME_TYPE_DIO oder ME_TYPE_DO programmiert. Angenommen das D/A-FIFO ist 16 bit breit, dann kann das Datenwort getrennt als „Low-Byte“ und „High-Byte“ behandelt werden. Die Werte können über den Parameter `<iRef>` (ME_REF_FIFO_LOW für Bit 7...0 und ME_REF_FIFO_HIGH für Bit 15...8 in der Funktion `meIOSingleConfig()` zugewiesen werden. Im Parameter `<iSingleConfig>` muss ME_SINGLE_CONFIG_DIO_BIT_PATTERN übergeben werden.

Siehe auch Abb. 33 in Anhang B3 ab Seite 228.

Hinweise:

- Ports, die für „Bitmuster-Ausgabe“ verwendet werden sollen, müssen als Ausgang konfiguriert werden.
- Es besteht die Möglichkeit, die gleiche Datenquelle mit mehreren digitalen Ports zu verbinden.
- Nach Programmierung einer Umlenkung ist ein normaler Zugriff auf diese digitalen Ports (Lesen/Schreiben via `meIOSingle()`) nicht mehr möglich.
- Die Programmierung einer Umlenkung wird sofort aktiv.
- Falls nur eine Umlenkung programmiert wird (ohne Abtrennung des DACs), arbeitet der analoge Ausgang ganz normal mit korrespondierenden Werten zu denen am digitalen Ausgang.

3.4.3.3 Synchronstart

Einige Meilhaus-Karten (z. B.: ME-6000) bieten die Möglichkeit den Betrieb verschiedener Subdevices zu synchronisieren. Für diesen Zweck muss eine sog. Synchronliste generiert werden.

Hinweise:

- Es kann mehr als eine Synchronliste für ein einzelnes Device geben.
- Eine Synchronliste kann Subdevices verschiedenen Typs beinhalten.

Details finden Sie in der ME-iDS-Hilfedatei und im Hardware-Handbuch.

Standardmäßig läuft jedes Subdevice vollständig unabhängig. Übergeben Sie in diesem Fall die Konstante `ME_TRIG_CHAN_DEFAULT...` Um ein Subdevice der Synchronliste hinzuzufügen muss dieser Parameter mit `ME_TRIG_CHAN_SYNCHRONOUS` übergeben werden...

... für Single-Betrieb:

im Parameter `<iTrigChan>` der Funktion `meIOSingleConfig()`.

Beispiel: Die Impulsausgabe mehrerer Subdevices synchron starten:

- Konfigurieren Sie jedes Subdevice in der Funktion `meIOSingleConfig()` mit dem Flag `ME_TRIG_CHAN_SYNCHRONOUS`.
- In der Funktion `meIOSingle()` für jedes Subdevice den Wert für `...FIO_TICKS_TOTAL` und `...FIO_TICKS_FIRST_PHASE` ohne gesetztem „Synchron“-Flag schreiben mit Ausnahme des letzten, der mit dem Flag `ME_IO_SINGLE_TYPE_TRIG_SYNCHRONOUS` übergeben werden muss. Der letzte Schreibbefehl mit gesetztem „Synchron“-Flag startet die Ausgabe an allen Subdevices, die vorher mit `ME_TRIG_CHAN_SYNCHRONOUS` konfiguriert worden sind, sofort.

... für Streaming-Betrieb:

im Parameter `<iAcqStartTrigChan>` in der Triggerstruktur der Funktion `meIOStreamConfig()`.

Wählen Sie eine der folgenden Triggerquellen für die Synchronliste:

- **Externer Trigger:** Aktiver externer Trigger auf beliebigem Subdevice, das in der Synchronliste enthalten ist.

- **Software-Trigger:** *meIOSingle()* oder *meIOStreamStart()* mit dem Flag `..._TRIG_SYNCHRONOUS` wird als Software-Trigger bezeichnet. Subdevices, die einen Softwaretrigger generieren, müssen nicht in der Synchronliste enthalten sein.
- Funktion *meIOSingle()*, Parameter `<iFlags>`:
`ME_IO_SINGLE_TYPE_TRIG_SYNCHRONOUS`
- Funktion *meIOStreamStart()* Start-Eintrag
`pStartList[x].iFlags:`
`ME_IO_STREAM_START_TYPE_TRIG_SYNCHRO-`
`NOUS`

Wichtig: Ein externer Trigger kann eine Synchronliste nur starten, wenn das zugehörige Subdevice mit enthalten ist. Ein Software-Trigger startet die Liste auch wenn das Subdevice nicht Bestandteil der Synchronliste ist.

3.4.3.4 Offset-Einstellung

Sofern ein Eingangskanal (z. B. MEphisto-Scope) die Möglichkeit bietet den Messbereich mit einem Offset zu verschieben, müssen Sie vor Start der Operation die Funktion *meIOSetChannelOffset()* aufrufen. Beachten Sie, dass diese Funktionalität nur im Streaming-Betrieb möglich ist. Nach einem Reset mit der Funktion *meIOResetSubdevice()* werden alle Kanäle und Bereiche auf 0 gesetzt.

Beachten Sie auch die Beschreibung der Funktion *meIOSetChannelOffset()* im Kapitel Funktionsreferenz ab Seite 147.

3.4.4 Interrupt-Betrieb

Für Subdevices vom Typ „externer Interrupt“ (ME_TYPE_EXT_IRQ) ist kein Single-Betrieb ausführbar. Bei Bedarf können Sie den Interrupt-Betrieb mit der Funktion *meIOIrqStart()* aktivieren. Je nach Hardware können Sie via Parameter `<iIrqSource>` zwischen verschiedenen Interruptquellen wählen:

- ME_IRQ_SOURCE_DIO_LINE
Interruptquelle ist ein dedizierter, externer Interrupt-Eingang.
- ME_IRQ_SOURCE_DIO_PATTERN
Betriebsart „Bitmuster-Vergleich“ (z. B.: ME-5810/8100/8200):
Bei Bitmuster-Gleichheit wird ein Interrupt ausgelöst.
- ME_IRQ_SOURCE_DIO_MASK
Betriebsart „Bitmuster-Änderung“ (z. B.: ME-5100/5810, ME-8100/ 8200):
Bei Zustandsänderung von mindestens einem Bit, das als „sensitiv“ maskiert wurde, wird ein Interrupt ausgelöst.
- ME_IRQ_SOURCE_DIO_OVER_TEMP
Bei Überhitzung eines Treiberbausteins wird ein Interrupt ausgelöst (z. B.: ME-5810/8200).
- ME_IRQ_SOURCE_DIO_NORMAL_TEMP
Interrupt wird ausgelöst sobald ein überhitzter Treiberbaustein wieder seine normale Temperatur erreicht hat.
- ME_IRQ_SOURCE_DIO_CHANGE_TEMP
Interrupt wird ausgelöst sowohl bei Überhitzung als auch bei Abkühlung auf Normaltemperatur eines Treiberbaustein.

Verwenden Sie die Funktion *meIOIrqWait()* um die verschiedenen Interruptquellen (je nach Hardware) auszuwerten. Zum Beenden des Interrupt-Betriebs verwenden Sie die Funktion *meIOIrqStop()*.

Beachten Sie auch die Beschreibung der *meIOIrq...* Funktionen im Kapitel Funktionsreferenz ab Seite 144.

Bei Bedarf kann eine benutzerdefinierte Callback-Funktion aufgerufen werden. Durch Rückgabe eines Wertes ungleich 0 von der Callback-Funktion kann der Interrupt-Betrieb beendet werden.

Siehe auch Abbildung auf der nächsten Seite.

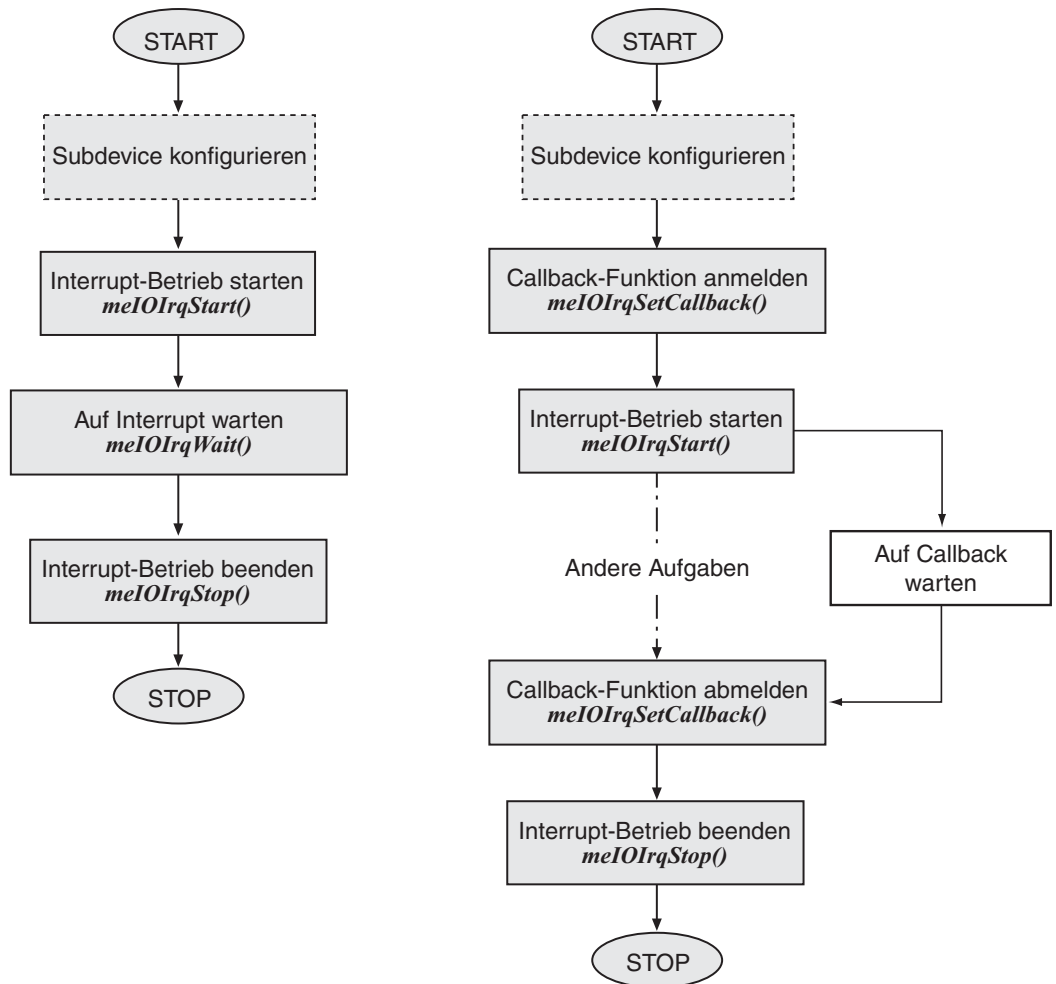


Abbildung 30: Interrupt-Betrieb ohne (links) und mit Callback-Funktion (rechts)

4 Funktionsreferenz

4.1 Allgemeine Hinweise

- **Funktionsprototypen:**
In der folgenden Funktionsbeschreibung werden die generischen Funktionsprototypen für ANSI C verwendet. Die Definitionen für andere unterstützte Programmiersprachen entnehmen Sie bitte den entsprechenden Definitions- bzw. Header-Dateien. Die unterschiedlichen Schreibweisen für Zeiger wie „int * piName“ und „int *piName“ sind als äquivalent zu betrachten.
- **Ausführungsmodus BLOCKING:**
Beachten Sie, dass es in Verbindung mit langen Abtast-Intervallen oder nicht bzw. nur spät eintreffenden externen Triggersignalen zu einer längeren Blockierung des Prozesses kommen kann.
- **Callback-Funktionen:**
Unter Agilent VEE, LabVIEW, bei älteren Visual Basic-Dialekten und Python können keine Callback-Funktionen verwendet werden.

Bei Rückgabe eines Wertes ungleich 0 durch eine Callback-Funktion kann der laufende Betrieb beendet werden.
- **Externer Trigger mit Time-Out:**
Bei Funktionen mit externem Trigger ist es möglich, ein Zeitintervall anzugeben, in dem der erste Triggerimpuls eintreffen muss, ansonsten wird die Operation abgebrochen (Parameter <iTimeOut>). Das Ausbleiben weiterer Triggerimpulse z. B. während der Erfassung in der Betriebsart Streaming wird dadurch nicht abgefangen. Berücksichtigen Sie dies gegebenenfalls bei der Programmierung.
- **Verwendung von Flags:**
Sofern sinnvoll, können Konstanten in den Parametern <iFlags> mit „ODER“-verknüpft werden.
- **Indizes:**
Indizes der Parameter <iDevice>, <iSubdevice>, <iChannel> und <iRange> beginnen immer bei 0.

4.2 Beschreibung der API-Funktionen

Die Funktionen sind nach Zusammengehörigkeit geordnet:

4.2.1 Query-Funktionen ab Seite 97

4.2.2 Property-Funktionen ab Seite 122

4.2.3 Ein-/Ausgabe-Funktionen ab Seite 135

4.2.4 Hilfs-Funktionen ab Seite 197

Funktion	Kurzbeschreibung	Seite
<i>Query-Funktionen</i>		
meQueryInfoDevice	Ermittelt Informationen wie Device-ID, Seriennummer, Bus-Typ...	97
meQueryNameDeviceDriver	Ermittelt die Treiberbezeichnung	99
meQueryNameDevice	Ermittelt die Typbezeichnung des Geräts	100
meQueryDescriptionDevice	Ermittelt String, der Gerät beschreibt	101
meQueryVersionLibrary	Ermittelt Version der Funktionsbibliothek	102
meQueryVersionMainDriver	Ermittelt Version des Haupttreibers	103
meQueryVersionDeviceDriver	Ermittelt Version des Gerätetreibers	104
meQueryNumberDevices	Ermittelt die Anzahl der vom Treibersystem erkannten Geräte	105
meQueryNumberSubdevices	Ermittelt die Anzahl der Subdevices eines Geräts	106
meQuerySubdeviceType	Ermittelt Typ eines Subdevice	107
meQueryNumberChannels	Ermittelt die Anzahl der Kanäle eines Subdevice	108
meQueryNumberRanges	Ermittelt die Anzahl der Messbereiche eines analogen Subdevice	109
meQueryRangeInfo	Ermittelt die Bereichsgrenzen eines analogen Messbereichs	110
meQuerySubdeviceCaps	Ermittelt besondere Fähigkeiten eines Subdevice	112
meQuerySubdeviceCapsArgs	Ermittelt Details der besonderen Fähigkeiten eines Subdevice	116
meQuerySubdeviceByType	Ermittelt Index eines Subdevice vom gewünschten Typ	118
meQueryRangeByMinMax	Ermittelt den passenden analogen Messbereich durch Übergabe der Bereichsgrenzen	120

Tabelle 14: Übersicht der Bibliotheksfunktionen

Funktion	Kurzbeschreibung	Seite
<i>Property-Funktionen</i>		
mePropertyGetInt(A/W)	Properties vom Typ Integer ermitteln	123
mePropertyGetDouble(A/W)	Properties vom Typ Double ermitteln	125
mePropertyGetString(A/W)	Properties vom Typ String ermitteln	127
mePropertySetInt(A/W)	Properties vom Typ Integer setzen	129
mePropertySetDouble(A/W)	Properties vom Typ Double setzen	131
mePropertySetString(A/W)	Properties vom Typ String setzen	133
<i>Ein-/Ausgabe-Funktionen</i>		
meIOResetDevice	Das gesamte Gerät wird rückgesetzt	135
meIOResetSubdevice	Das Subdevice wird rückgesetzt	136
meIOIrqStart	Interrupt-Betrieb aktivieren	137
meIOIrqStop	Interrupt-Betrieb deaktivieren	141
meIOIrqWait	Interrupt-Ereignis auswerten	141
meIOIrqSetCallback	Callback-Funktion für IRQ installieren	144
meIOSetChannelOffset	Offset für analoge Eingänge verschieben (z.Zt. nur im Streaming-Betrieb möglich)	147
meIOSingleConfig	Ein Kanal wird für Ein-/Ausgabe eines einzelnen Wertes konfiguriert	149
meIOSingle	Einzelwert-Ein-/Ausgabe	155
meIOSingleTicksToTime	Ticks in Zeit [s] umrechnen	160
meIOSingleTimeToTicks	Zeit [s] in Ticks umrechnen	162
meIOStreamConfig	Kontinuierliche Ein-/Ausgabe wird vorbereitet	164
meIOStreamTimeToTicks	Periodendauer [s] in Ticks umrechnen	175
meIOStreamFrequencyToTicks	Frequenz [Hz] in Ticks umrechnen	177
meIOStreamStart	Ein-/Ausgabe wird gestartet	179
meIOStreamStop	Ein-/Ausgabe wird gestoppt	182
meIOStreamRead	Timergesteuerte Erfassung	185
meIOStreamWrite	Timergesteuerte Ausgabe	188
meIOStreamStatus	Status-Abfrage während Ein-/Ausgabe	191
meIOStreamNewValues	Status-Abfrage im Streaming-Betrieb	193
meIOStreamSetCallbacks	Callback-Routinen installieren	194

Tabelle 14: Übersicht der Bibliotheksfunktionen

Funktion	Kurzbeschreibung	Seite
<i>Hilfs-Funktionen</i>		
meOpen	ME-iDS wird initialisiert	197
meClose	ME-iDS wird geschlossen	198
meLockDriver	Treiber reservieren	199
meLockDevice	Gerät reservieren	200
meLockSubdevice	Subdevice reservieren	201
meErrorGetLast	Gibt den letzten Fehlercode zurück	202
meErrorGetLastMessage	Zuletzt aufgetretenem Fehler einen Fehlerstring zuweisen	203
meErrorGetMessage	Fehlernummer einen Fehlerstring zuweisen.	204
meErrorSetDefaultProc	Vordefinierte, globale Fehlerroutine installieren	205
meErrorSetUserProc	Benutzerdefinierte, globale Fehlerroutine installieren	206
meUtilityDigitalToPhysical	Rechnet normierten Digital-Wert in physikalischen Wert um	207
meUtilityDigitalToPhysicalV	Wie meUtilityDigitalToPhysical, jedoch für Wertefeld anwendbar	211
meUtilityPhysicalToDigital	Rechnet physikalischen Wert in normierten Digital-Wert um	212
meUtilityPhysicalToDigitalV	Wie meUtilityPhysicalToDigital, jedoch für Wertefeld anwendbar	213
meUtilityExtractValues	Extrahiert die Werte für einen Kanal aus dem Datenpuffer	214
meUtilityPWMStart	Startet PWM-Betrieb für 8254	207
meUtilityPWMStop	Beendet PWM-Betrieb	207
meUtilityPWMRestart	PWM-Operation erneut starten ohne Reset	219

Tabelle 14: Übersicht der Bibliotheksfunktionen

4.2.1 Query-Funktionen

meQueryInfoDevice

Beschreibung

Mit dieser Funktion erhalten Sie detaillierte Informationen über das spezifizierte Gerät. Dies ist eine PCI-orientierte Funktion. Einige Parameter haben keine Bedeutung z. B. für ME-Synapse USB oder ME-Synapse LAN.

● Funktions-Deklaration

```
int meQueryInfoDevice(int iDevice, int *piVendorId, int *piDeviceId, int
    *piSerialNo, int *piBusType, int *piBusNo, int *piDevNo, int
    *piFuncNo, int *piPlugged);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<piVendorId> (r)

Zeiger gibt Hersteller-Kennung (Vendor-ID) des Geräts zurück.

- 1402Hex für Meilhaus PCI-Karten
- 1B04Hex für USB-Geräte wie z. B. ME-1 (ME-Synapse USB).

<piDeviceId> (r)

Zeiger gibt Kennung (Device-ID) des Gerätetyps zurück.

<piSerialNo> (r)

Zeiger gibt Seriennummer des Geräts zurück.

<piBusType> (r)

Zeiger gibt den Bustyp zurück über den das Gerät mit dem PC verbunden ist (PCI/cPCI, USB).

- ME_BUS_TYPE_INVALID Ungültiger Rückgabewert
- ME_BUS_TYPE_PCI PCI/cPCI-Bus
- ME_BUS_TYPE_USB Universal Serial Bus (USB)

<piBusNo> (r)

- PCI: Zeiger gibt die PCI-Busnummer zurück, falls mehrere PCI-Busse im System vorhanden sind (bei einem Bus immer 0).

<piDevNo> (r)

- PCI: Slot-Nr. in dem die angesprochene Karte steckt.

<piFuncNo> (r)

- PCI: Funktionsnummer.

<piPlugged> (r)

Zeiger gibt an, ob Gerät physikalisch vorhanden ist.

- ME_PLUGGED_INVALID Ungültiger Rückgabewert
- ME_PLUGGED_IN
Gerät physikalisch vorhanden.
- ME_PLUGGED_OUT
Gerät ist im ME-Config-Tool angemeldet aber physikalisch nicht vorhanden.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebene Zeiger sind NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.

meQueryNameDeviceDriver

Beschreibung

Funktion ermittelt den Namen des gerätespezifischen Treibermoduls. Z. B.: “me6000PCI” (Linux) oder “ME-6000” (Windows).

● Funktions-Deklaration

```
int meQueryNameDeviceDriver(int iDevice, char *pcName, int iCount);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<pcName>

Zeiger auf einen Puffer, der nach erfolgreichem Aufruf der Funktion den Namen des Treibermoduls enthält, einschließlich terminierendem Nullzeichen.

<iCount>

Puffergröße in Bytes für Treiberbezeichnung. Empfohlener Wert: ME_DEVICE_DRIVER_NAME_MAX_COUNT.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_USER_BUFFER_SIZE: Puffergröße von <pcName> zu klein.

meQueryNameDevice

Beschreibung

Funktion ermittelt die Typenbezeichnung des Geräts.

Z. B.: „ME-6000ISLE/16“

● Funktions-Deklaration

```
int meQueryNameDevice(int iDevice, char *pcName, int iCount);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<pcName>

Zeiger auf einen Puffer, der nach erfolgreichem Aufruf der Funktion die Typenbezeichnung des Geräts enthält einschließlich terminierendem Nullzeichen.

<iCount>

Puffergröße in Bytes für Typbezeichnung. Empfohlen:
ME_DEVICE_NAME_MAX_COUNT.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_USER_BUFFER_SIZE: Puffergröße von <pcName> zu klein.

meQueryDescriptionDevice

Beschreibung

Funktion liefert Zeiger auf Gerätebeschreibung. Z. B.: “ME-6000ISLE/4 isle device, 4 analog outputs”.

● Funktions-Deklaration

```
int meQueryDescriptionDevice(int iDevice, char *pcDescription, int  
                             iCount);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<pcDescription>

Zeiger auf einen Puffer, der nach erfolgreichem Aufruf der Funktion die Gerätebeschreibung enthält, einschließlich terminierendem Nullzeichen.

<iCount>

Puffergröße in Bytes für Gerätebeschreibung. Verwenden Sie die Konstante ME_DEVICE_DESCRIPTION_MAX_COUNT.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_USER_BUFFER_SIZE: Puffergröße von <pcDescription> zu klein.

meQueryVersionLibrary

Beschreibung

Funktion ermittelt die Versionsnummer der Funktionsbibliothek.

● Funktions-Deklaration

```
int meQueryVersionLibrary(int *piVersion);
```

<piVersion> (r)

Gibt Versionsnummer der Bibliothek zurück (hexadezimal).

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.

meQueryVersionMainDriver

Beschreibung

Funktion ermittelt die Versionsnummer des Haupttreibers.

● Funktions-Deklaration

```
int meQueryVersionMainDriver(int *piVersion);
```

<piVersion> (r)

Versionsnummer des Haupttreibers (hexadezimal). Die höherwertigen 2 Bytes (Hauptversion, Unterversion) müssen mit der des gerätespezifischen Treibermoduls (siehe *meQueryVersionDeviceDriver()*) identisch sein. Die niederwertigen Bytes (Build-Nummer) können unterschiedlich sein.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.

meQueryVersionDeviceDriver

Beschreibung

Funktion ermittelt die Versionsnummer des gerätespezifischen Treiber-Moduls.

● Funktions-Deklaration

```
int meQueryVersionDeviceDriver(int iDevice, int *piVersion);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<piVersion> (r)

Versionnummer des gerätespezifischen Treiber-Moduls (hexadezimal).
Die höherwertigen 2 Bytes (Hauptversion, Unterversion) müssen mit der des Haupttreibers (siehe *meQueryVersionMainDriver()*) identisch sein.
Die niederwertigen Bytes (Build-Nummer) können unterschiedlich sein.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.

meQueryNumberDevices

Beschreibung

Funktion ermittelt die Anzahl der vom ME-iDS erkannten Geräte.

● Funktions-Deklaration

```
int meQueryNumberDevices(int *piNumber);
```

<piNumber> (r)

Anzahl der erkannten Geräte.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.

meQueryNumberSubdevices

Beschreibung

Funktion ermittelt die Anzahl der Funktionsgruppen (Subdevices) eines Geräts.

● Funktions-Deklaration

```
int meQueryNumberSubdevices(int iDevice, int *piNumber);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<piNumber> (r)

Anzahl der Funktionsgruppen (Subdevices) auf dem Gerät.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.

meQuerySubdeviceType

Beschreibung

Funktion ermittelt Typ und Untertyp der spezifizierten Funktionsgruppe.

● Funktions-Deklaration

```
int meQuerySubdeviceType(int iDevice, int iSubdevice, int *piType, int
                        *piSubtype);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des Subdevice, welches abgefragt werden soll.

<piType> (r)

Gibt Typ des Subdevice zurück:

- ME_TYPE_AI Analoge Erfassung
- ME_TYPE_AO Analoge Ausgabe
- ME_TYPE_DIO Digitale Ein-/Ausgabe (bidir.)
- ME_TYPE_DI Digitale Eingabe
- ME_TYPE_DO Digitale Ausgabe
- ME_TYPE_FIO Frequenz-Ein-/Ausgabe
- ME_TYPE_FI Frequenz-Eingabe
- ME_TYPE_FO Frequenz-Ausgabe
- ME_TYPE_CTR Zähler
- ME_TYPE_EXT_IRQ Externer Interrupt

<piSubtype> (r)

Gibt den Untertyp des Subdevice zurück:

- ME_SUBTYPE_SINGLE
Subdevice kann einzelne Werte erfassen bzw. ausgeben.
- ME_SUBTYPE_STREAMING
Das Subdevice kann kontinuierlich Werte erfassen bzw. einen Datenstrom ausgeben.
- ME_SUBTYPE_CTR_8254
Subdevice mit einem Zähler vom Typ 8254.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.

- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.

meQueryNumberChannels

Beschreibung

Funktion ermittelt die Anzahl der Kanäle eines Subdevice.

● Funktions-Deklaration

```
int meQueryNumberChannels(int iDevice, int iSubdevice, int *piNumber);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des Subdevice, welches abgefragt werden soll.

<piNumber> (r)

Gibt die Anzahl der Kanäle des spezifizierten Subdevices zurück.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.

meQueryNumberRanges

Beschreibung

Funktion ermittelt die Anzahl der Messbereiche eines Subdevices, welche die spezifizierte Messeinheit unterstützen. Der Parameter `<iUnit>` ermöglicht eine Eingrenzung der Abfrage auf eine bestimmte physikalische Einheit.

● Funktions-Deklaration

```
int meQueryNumberRanges(int iDevice, int iSubdevice, int iUnit, int
                        *piNumber);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des Subdevice, welches abgefragt werden soll.

<iUnit>

Messbereiche mit der hier spezifizierten physikalischen Einheit sollen in die Abfrage einbezogen werden (siehe auch *meQueryRangeByMinMax()*).

- ME_UNIT_VOLT Nur Spannungsbereiche
- ME_UNIT_AMPERE Nur Strombereiche
- ME_UNIT_ANY Alle Bereiche

<piNumber> (r)

Gibt die Anzahl der Bereiche zurück, welche die spezifizierte Einheit unterstützen.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_NOT_SUPPORTED: Funktion wird vom Subdevice nicht unterstützt.

meQueryRangeInfo

Beschreibung

Funktion ermittelt Details des spezifizierten Messbereiches wie: Bereichsgrenzen, Auflösung und physikalische Einheit.

● Funktions-Deklaration

```
int meQueryRangeInfo(int iDevice, int iSubdevice, int iRange, int *piUnit,
    double *pdMin, double *pdMax, int *piMaxData);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des Subdevice, welches abgefragt werden soll.

<iRange>

Index des Messbereiches, der abgefragt werden soll.

<piUnit> (r)

Zeiger, der die physikalische Einheit des spezifizierten Messbereichs zurückgibt.

- ME_UNIT_VOLT Spannungsbereich
- ME_UNIT_AMPERE Strombereich
- ME_UNIT_INVALID Ungültiger Rückgabewert

<pdMin> (r)

Gibt die untere Bereichsgrenze zurück. Es gilt die unter <piUnit> ermittelte physikalische Einheit.

<pdMax> (r)

Gibt die obere Bereichsgrenze zurück. Es gilt die unter <piUnit> ermittelte physikalische Einheit.

<piMaxData> (r)

Gibt die max. Auflösung des Messbereichs zurück (z. B. der Wert 65535 (0xFFFF) für 16 Bit Auflösung).

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.

- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_RANGE: Auf dem angefragten Subdevice ist kein gewünschter Bereich verfügbar.
- ME_ERRNO_NOT_SUPPORTED: Funktion wird vom Subdevice nicht unterstützt

meQuerySubdeviceCaps

Beschreibung

Funktion ermittelt die besonderen Fähigkeiten, sog. „Caps“ eines Subdevices, welche im Parameter `<piCaps>` der Funktion `meQuerySubdeviceCaps()` zurückgegeben werden. Weitere Details können mit der Funktion `meQuerySubdeviceCapsArgs()` ermittelt werden.

● Funktions-Deklaration

```
int meQuerySubdeviceCaps(int iDevice, int iSubdevice, int *piCaps);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des Subdevice, welches abgefragt werden soll.

<piCaps> (r)

Zeiger auf einen bit-kodierten Integerwert, der die speziellen Fähigkeiten des spezifizierten Subdevices zurückgibt. Ein gesetztes Bit zeigt an, dass das Subdevice die entsprechende Fähigkeit besitzt. Sofern mehrere „Caps“ zutreffen, werden die Werte bitweise ODER-verknüpft.

Beispiel: ein Subdevice verfügt über einen digitalen Triggereingang, der wahlweise auf steigende, fallende oder beliebige (d. h. steigende oder fallende) Flanke triggert. Der entsprechende Rückgabewert ist: 0x000E8000.

Hinweis: Eine Tabelle aller „Caps“, die hier abgefragt werden können, finden Sie in Anhang C1 auf Seite 233.

- ME_CAPS_NONE (0x00000001)
Subdevice hat keine besonderen Fähigkeiten

Fortsetzung auf nächster Seite...

Für Subdevices vom Typ *ME_TYPE_AI*, *ME_TYPE_AO*, *ME_TYPE_DI*, *ME_TYPE_DO* und *ME_TYPE_DIO* gilt:

Ersetzen Sie *xx* je nach Subdevice-Typ mit **AI**, **AO** oder **DIO** (siehe auch Tabelle 15 ab Seite 233).

- ME_CAPS_XX_TRIG_DIGITAL (0x00008000)
Subdevice hat digitalen Triggereingang.
- ME_CAPS_XX_TRIG_ANALOG (0x00010000)
Subdevice hat analogen Triggereingang.
- ME_CAPS_XX_TRIG_EDGE_RISING (0x00020000)
Subdevice kann gezielt auf steigende Flanke triggern.
- ME_CAPS_XX_TRIG_EDGE_FALLING (0x00040000)
Subdevice kann gezielt auf fallende Flanke triggern.
- ME_CAPS_XX_TRIG_EDGE_ANY (0x00080000)
Subdevice kann auf steigende oder fallende Flanke triggern.

Für Subdevices vom Typ *ME_TYPE_AI* gilt:

- ME_CAPS_AI_TRIG_SYNCHRONOUS (0x00000001)
Subdevice kann synchron gestartet werden.
- ME_CAPS_AI_FIFO (0x00000002)
Subdevice verfügt über FIFO für erfasste Werte.
- ME_CAPS_AI_FIFO_THRESHOLD (0x00000004)
Möglichkeit die Schwelle (Anzahl der Messwerte) einzustellen, bei der die Werte im A/D-FIFO „abgeholt“ werden sollen.
Kann gegebenenfalls mit Parameter `<i>FIFOIrqThreshold> der Funktion meIOStreamConfig() eingestellt werden.`
- ME_CAPS_AI_SAMPLE_HOLD (0x00000008)
Subdevice verfügt über Sample & Hold-Stufe für simultane Erfassung.

Für Subdevices vom Typ *ME_TYPE_AO* gilt:

- ME_CAPS_AO_TRIG_SYNCHRONOUS (0x00000001)
Subdevice bietet Möglichkeit zum Synchron-Start der Ausgabe.
- ME_CAPS_AO_FIFO (0x00000002)
Subdevice verfügt über FIFO für Ausgabewerte.
- ME_CAPS_AO_FIFO_THRESHOLD (0x00000004)
Möglichkeit die Schwelle (Anzahl der Ausgabewerte) einzustellen, bei der das D/A-FIFO nachgeladen werden soll.
Kann gegebenenfalls mit Parameter `<i>FIFOIrqThreshold> der Funktion meIOStreamConfig() eingestellt werden.`

Für Subdevices vom Typ ME_TYPE_DI, ME_TYPE_DO und ME_TYPE_DIO gilt:

- ME_CAPS_DIO_DIR_BIT (0x00000001)
Richtung je Bit (1bit Block) konfigurierbar.
- ME_CAPS_DIO_DIR_BYTE (0x00000002)
Richtung je Byte (8bit Block) konfigurierbar.
- ME_CAPS_DIO_DIR_WORD (0x00000004)
Richtung je Wort (16bit Block) konfigurierbar.
- ME_CAPS_DIO_DIR_DWORD (0x00000008)
Richtung je Langwort (32bit Block) konfigurierbar.
- ME_CAPS_DIO_SINK_SOURCE (0x00000010)
Ausgangstreiber des Subdevices können zwischen „Sink“- und „Source“-Betrieb umgeschaltet werden.
- ME_CAPS_DIO_BIT_PATTERN_IRQ (0x00000020)
Bei Bitmuster-Gleichheit kann Interrupt ausgelöst werden.
- ME_CAPS_DIO_BIT_MASK_IRQ_EDGE_RISING (0x00000040)
Bei steigender Flanke von mindestens einem der aktiven Bits kann Interrupt ausgelöst werden.
- ME_CAPS_DIO_BIT_MASK_IRQ_EDGE_FALLING (0x00000080)
Bei fallender Flanke von mindestens einem der aktiven Bits kann Interrupt ausgelöst werden.
- ME_CAPS_DIO_BIT_MASK_IRQ_EDGE_ANY (0x00000100)
Bei steigender oder fallender Flanke von mindestens einem der aktiven Bits kann Interrupt ausgelöst werden.
- ME_CAPS_DIO_OVER_TEMP_IRQ (0x00000200)
Bei Überhitzung des Treiberbausteins kann Interrupt ausgelöst werden (z. B. ME-5810/8200).

Für Subdevices vom Typ ME_TYPE_CTR gilt:

- ME_CAPS_CTR_CLK_PREVIOUS (0x00000001)
Möglichkeit den Takteingang (CLK) eines Zählers mit dem Zählerausgang (OUT) des vorherigen Zählers zu verknüpfen.
- ME_CAPS_CTR_CLK_INTERNAL_1MHZ (0x00000002)
Möglichkeit zur Speisung des Zählers mit internem 1 MHz Takt.
- ME_CAPS_CTR_CLK_INTERNAL_10MHZ (0x00000004)
Möglichkeit zur Speisung des Zählers mit internem 10 MHz Takt.
- ME_CAPS_CTR_CLK_EXTERNAL (0x00000008)
Möglichkeit zur Speisung des Zählers mit externer Taktquelle.

Für Subdevices vom Typ ME_TYPE_EXT_IRQ gilt:

- ME_CAPS_EXT_IRQ_EDGE_RISING (0x00000001)
Interrupt mit steigender Flanke am IRQ-Eingang auslösen.
- ME_CAPS_EXT_IRQ_EDGE_FALLING (0x00000002)
Interrupt mit fallende Flanke am IRQ-Eingang auslösen.
- ME_CAPS_EXT_IRQ_EDGE_ANY (0x00000004)
Interrupt mit beliebiger Flanke (steigend oder fallend) am IRQ-Eingang auslösen.

Für Subdevices vom Typ ME_TYPE_FO gilt:

- ME_CAPS_FIO_SINK_SOURCE (0x00000010)
Ausgangstreiber des Subdevices können zwischen „Sink“- und „Source“-Betrieb umgeschaltet werden.

Siehe auch Anhang C1 auf Seite 233 für weitere Informationen.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.

meQuerySubdeviceCapsArgs

Beschreibung

Diese Funktion liefert detaillierte Informationen über eine bestimmte Fähigkeit eines Subdevices. Siehe auch Funktion *meQuerySubdeviceCaps()*.

● Funktions-Deklaration

```
int meQuerySubdeviceCapsArgs(int iDevice, int iSubdevice, int iCap, int
    *piArgs, int iCount);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des Subdevice, welches abgefragt werden soll.

<iCap>

Auswahl welche „Cap“ abgefragt werden soll (Auswahl):

- ME_CAP_AI_FIFO_SIZE (0x001D0000)
Größe (Anzahl der Werte) des A/D-FIFOs abfragen.
- ME_CAP_AO_FIFO_SIZE (0x001F0000)
Größe (Anzahl der Werte) des D/A-FIFOs abfragen.
- ME_CAP_CTR_WIDTH (0x00200000)
Breite des Zählers in Bits abfragen.

Siehe Anhang C2 auf Seite 236 für eine komplette Liste der „Caps“, die hier abgefragt werden können.

<piArgs> (r)

Zeiger auf Feld vom Typ Integer, das die abgefragten Werte zurückgibt.

<iCount>

Anzahl der Werte im Parameter <piArgs>. In der Regel „1“, sofern die zurückgegebenen Werte in einen Integer passen.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.

- ME_ERRNO_INVALID_CAP: Übergebener Code ist ungültig oder nicht unterstützt durch Subdevice.
- ME_ERRNO_INVALID_CAP_ARG_COUNT: Parameter `<iCount>` passt nicht zum Argument der angefragten "Cap".

meQuerySubdeviceByType

Beschreibung

Funktion ermittelt den Index des ersten verfügbaren Subdevices, das dem gesuchten Typ entspricht. Die Suche beginnt stets bei dem Subdevice mit dem Index <iStartSubdevice>.

Hinweis

Intern genutzte Hardware-Ressourcen werden nicht gemeldet.

● Funktions-Deklaration

```
int meQuerySubdeviceByType(int iDevice, int iStartSubdevice, int iType,  
    int iSubtype, int *piSubdevice);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iStartSubdevice>

Index des Subdevices mit dem die Suche beginnen soll.

<iType>

Typ des gesuchten Subdevices:

- | | |
|-------------------|---|
| • ME_TYPE_AI | Analoge Erfassung |
| • ME_TYPE_AO | Analoge Ausgabe |
| • ME_TYPE_DIO | Digitale Ein-/Ausgabe (bidir.) |
| • ME_TYPE_DI | Digitale Eingabe |
| • ME_TYPE_DO | Digitale Ausgabe |
| • ME_TYPE_FIO | Frequenz-Ein-/Ausgabe |
| • ME_TYPE_FI | Frequenz-Eingabe |
| • ME_TYPE_FO | Frequenz-Ausgabe |
| • ME_TYPE_CTR | Zähler |
| • ME_TYPE_EXT_IRQ | Externer Interrupt |
| • ME_TYPE_FPGA | „FPGA“ (geplant)
- nur für fortgeschrittene Anwender |

<iSubtype>

Mit diesem Parameter kann die Art des gesuchten Subdevices noch näher spezifiziert werden:

- ME_SUBTYPE_ANY
Alle Untertypen berücksichtigen.
- ME_SUBTYPE_SINGLE
Erfassung bzw. Ausgabe eines einzelnen Wertes.
- ME_SUBTYPE_STREAMING
Kontinuierliche Erfassung bzw. Ausgabe von Werten unter Nutzung spezieller Hardware-Eigenschaften (z. B. FIFOs).
- ME_SUBTYPE_CTR_8254
Subdevice mit einem Zähler vom Typ 8254.

<piSubdevice> (r)

Gibt den Index des ersten, passenden Subdevices zurück.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_NO_MORE_SUBDEVICE_TYPE: Passendes Subdevice nicht gefunden.

meQueryRangeByMinMax

Beschreibung

Funktion ermittelt den passenden Messbereich durch Vorgabe der gewünschten Bereichsgrenzen.

● Funktions-Deklaration

```
int meQueryRangeByMinMax(int iDevice, int iSubdevice, int iUnit, double
    *pdMin, double *pdMax, int *piMaxData, int *piRange);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des Subdevice, welches abgefragt werden soll.

<iUnit>

Bereichsabfrage auf die hier gewählte physikalische Einheit beschränken.

- ME_UNIT_VOLT Nur Spannungsbereiche
- ME_UNIT_AMPERE Nur Strombereiche
- ME_UNIT_ANY Alle Bereiche

<pdMin> (r/w)

(w) : Vorgabe der gewünschten unteren Bereichsgrenze.

(r) : Gibt die untere Bereichsgrenze für den ermittelten Bereich zurück (siehe Parameter <piRange>).

<pdMax> (r/w)

(w) : Vorgabe der gewünschten oberen Bereichsgrenze.

(r) : Gibt die obere Bereichsgrenze für den ermittelten Bereich zurück (siehe Parameter <piRange>)

<piMaxData> (r)

Gibt die max. Auflösung für den ermittelten Bereich zurück (z. B. 65535 für 16 bit Auflösung).

<piRange> (r)

Index des am besten passenden Messbereichs wird zurückgegeben. Es wird immer der kleinste Bereich gewählt, der die gesuchten Bereichsgrenzen umfasst.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.

- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_UNIT: Bereichstyp vom Subdevice nicht unterstützt.
- ME_ERRNO_INVALID_MIN_MAX: Untere Grenze ist höher als obere Bereichsgrenze.
- ME_ERRNO_NO_RANGE: Passender Bereich nicht gefunden.

4.2.2 Property-Funktionen

Mit den sog. Property-Funktionen haben Sie die Möglichkeit alle allgemeinen und hardware-spezifischen Properties und Attribute zu ermitteln und gegebenenfalls auch zu setzen. Die Gesamtheit der Properties stellt eine Baumstruktur dar. Über den sog. Property-Pfad können Sie damit auf ein Device, ein Subdevice, einen Kanal (Channel) oder einen Messbereich (Range) usw. zugreifen.

Alle Property-Funktionen sind als ANSI- und Unicode (UTF-16)-Version implementiert. Die ANSI-Version (mit Suffix „A“) verwendet NULL-terminierte ANSI-Strings (char*). Die Unicode-Version (mit Suffix „W“) nutzt „Wide Character“ Strings (wchar_t*).

In Abhängigkeit vom zu übergebenden Wert (Integer, Double, String), muss die entsprechende Funktion zum Lesen bzw. Schreiben des Wertes verwendet werden (siehe auch Parameter <piValue> der Funktion *mePropertyGetIntA()* auf Seite 123).

Hinweis:

Die Property-Funktionen stehen ab ME-iDS 2.0 zur Verfügung und sind unter Windows in vollem Umfang für die ME-5000-Serie implementiert. Für die anderen Devices werden augenblicklich nur die allgemeinen Properties unterstützt. In zukünftigen Versionen des ME-iDS werden diese aber weiter ausgebaut und vervollständigt. Die Unterstützung der Property-Funktionen unter Linux ist in Vorbereitung.

mePropertyGetIntA mePropertyGetIntW

Beschreibung

Mit dieser Funktion können Sie Properties vom Typ ME_PROPERTY_TYPE_INT über den sog. Property-Pfad abfragen. Sie können diese Funktion auch dazu verwenden um den unbekannten Typ einer Property (*PropertyType*) zu ermitteln.

Um mehr über die möglichen Properties der von Ihnen eingesetzten Hardware zu erfahren, beachten Sie bitte die ME-iDS Hilfe-Datei (*.chm), welche Sie über das „ME-iDS Control Center“ oder das Windows-Startmenü aufrufen können.

Hinweis

Diese Funktion steht ab ME-iDS Rev. 2.0 zur Verfügung.

● Funktions-Deklaration

```
int mePropertyGetIntA(char* pcPropertyPath, int* piValue);
int mePropertyGetIntW(wchar_t* pcPropertyPath, int* piValue);
```

<pcPropertyPath> (w)

Zeiger auf Property-Pfad auf den zugegriffen werden soll.

<piValue> (r)

Zeiger auf Puffer, der die Property als Integer-Wert zurückgibt.

Neben den normalen Rückgabewerten wird bei Abfrage der Property *PropertyType* eine der folgenden Konstanten zurückgegeben:

- ME_PROPERTY_TYPE_CONTAINER: Enthält weitere Properties.
- ME_PROPERTY_TYPE_BOOL: Abgefragte Property ist ein bool'scher Wert 0 (FALSE) oder „1“ (TRUE) für Aus/Ein bzw. inaktiv/aktiv.
- ME_PROPERTY_TYPE_INT: Abgefragte Property ist vom Typ „Integer“.
- ME_PROPERTY_TYPE_DOUBLE: Abgefragte Property ist vom Typ „Double“.
- ME_PROPERTY_TYPE_STRING: Abgefragte Property ist vom Typ „String“.
- ME_PROPERTY_TYPE_DEFINE: Abgefragte Property gibt eine vordefinierte Konstante als Integer-Wert zurück. Siehe auch Funktion *mePropertyGetString...* auf Seite 127.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_PROPERTY_CONTAINER: Es wurde versucht, einen Wert von einem Container zu lesen.
- ME_ERRNO_PROPERTY_DATA_TYPE: Der Wert im angegebenen Property-Pfad kann nicht als Integer gelesen werden.
- ME_ERRNO_PROPERTY_PATH: Die Angabe des Property-Pfades ist nicht gültig.
- ME_ERRNO_PROPERTY_INDEX: Der angegebene Index im Property-Pfad ist außerhalb des zulässigen Bereichs.
- ME_ERRNO_PROPERTY_UNKNOWN: Die angegebene Property ist unbekannt.
- ME_ERRNO_PROPERTY_ATTRIBUTE_UNKNOWN: Das angegebene Attribut ist unbekannt.
- ME_ERRNO_PROPERTY_ATTRIBUTE_UNSUPPORTED: Das angegebene Attribut wird nicht unterstützt.

mePropertyGetDoubleA mePropertyGetDoubleW

Beschreibung

Mit dieser Funktion können Sie Properties vom Typ ME_PROPERTY_TYPE_DOUBLE über den sog. Property-Pfad abfragen. Verwenden Sie zunächst die Funktion *mePropertyGetIntA()*, falls Sie den unbekannten Typ einer Property (*PropertyType*) ermitteln möchten.

Um mehr über die möglichen Properties der von Ihnen eingesetzten Hardware zu erfahren, beachten Sie bitte die ME-iDS Hilfe-Datei (*.chm), welche Sie über das „ME-iDS Control Center“ oder das Windows-Startmenü aufrufen können.

Hinweis

Diese Funktion steht ab ME-iDS Rev. 2.0 zur Verfügung.

● Funktions-Deklaration

```
int mePropertyGetDoubleA(char* pcPropertyPath, double* pdValue);
int mePropertyGetDoubleW(wchar_t* pcPropertyPath, double* pdValue);
```

<pcPropertyPath> (w)

Zeiger auf Property-Pfad auf den zugegriffen werden soll.

<pdValue> (r)

Zeiger auf Puffer, der die Property als Double-Wert zurückgibt.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_PROPERTY_CONTAINER: Es wurde versucht, einen Wert von einem Container zu lesen.
- ME_ERRNO_PROPERTY_DATA_TYPE: Der Wert im angegebenen Property-Pfad kann nicht als Double gelesen werden.
- ME_ERRNO_PROPERTY_PATH: Die Angabe des Property-Pfades ist nicht gültig.
- ME_ERRNO_PROPERTY_INDEX: Der angegebene Index im Property-Pfad ist außerhalb des zulässigen Bereichs.
- ME_ERRNO_PROPERTY_UNKNOWN: Die angegebene Property ist unbekannt.

- ME_ERRNO_PROPERTY_ATTRIBUTE_UNKNOWN: Das angegebene Attribut ist unbekannt.
- ME_ERRNO_PROPERTY_ATTRIBUTE_UNSUPPORTED: Das angegebene Attribut wird nicht unterstützt.

mePropertyGetStringA mePropertyGetStringW

Beschreibung

Mit dieser Funktion können Sie Properties vom Typ ME_PROPERTY_TYPE_STRING über den sog. Property-Pfad abfragen. Verwenden Sie zunächst die Funktion *mePropertyGetIntA()*, falls Sie den unbekannten Typ einer Property (*PropertyType*) ermitteln möchten.

Um mehr über die möglichen Properties der von Ihnen eingesetzten Hardware zu erfahren, beachten Sie bitte die ME-iDS Hilfe-Datei (*.chm), welche Sie über das „ME-iDS Control Center“ oder das Windows-Startmenü aufrufen können.

Hinweis

Diese Funktion steht ab ME-iDS Rev. 2.0 zur Verfügung.

● Funktions-Deklaration

```
int mePropertyGetStringA(char* pcPropertyPath, char* pcValue, int
    iBufferLength);
```

```
int mePropertyGetStringW(wchar_t* pcPropertyPath, wchar_t* pcValue,
    int iBufferLength);
```

<pcPropertyPath> (w)

Zeiger auf Property-Pfad auf den zugegriffen werden soll.

<pcValue> (r)

Zeiger auf einen Puffer, der nach erfolgreichem Aufruf der Funktion den Wert als String enthält, einschließlich terminierendem Nullzeichen.

<iBufferLength>

Pufferlänge in Zeichen (nicht Bytes), einschließlich terminierendem Nullzeichen. Die erforderliche Länge kann mit dem Attribut *Length* abgefragt werden.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_PROPERTY_CONTAINER: Es wurde versucht, einen Wert von einem Container zu lesen.
- ME_ERRNO_PROPERTY_DATA_TYPE: Der Wert im angegebenen Property-Pfad kann nicht als String gelesen werden.

- ME_ERRNO_PROPERTY_PATH: Die Angabe des Property-Pfades ist nicht gültig.
- ME_ERRNO_PROPERTY_INDEX: Der angegebene Index im Property-Pfad ist außerhalb des zulässigen Bereichs.
- ME_ERRNO_PROPERTY_BUFFER_TOO_SMALL: Die Größe des angegebenen Puffers ist zu klein, um den String aufzunehmen
- ME_ERRNO_PROPERTY_UNKNOWN: Die angegebene Property ist unbekannt.
- ME_ERRNO_PROPERTY_ATTRIBUTE_UNKNOWN: Das angegebene Attribut ist unbekannt.
- ME_ERRNO_PROPERTY_ATTRIBUTE_UNSUPPORTED: Das angegebene Attribut wird nicht unterstützt.

mePropertySetIntA mePropertySetIntW

Beschreibung

Mit dieser Funktion können Sie Properties vom Typ ME_PROPERTY_TYPE_INT über den sog. Property-Pfad ändern. Verwenden Sie zunächst die Funktion *mePropertyGetIntA()*, falls Sie den unbekannten Typ einer Property (*PropertyType*) ermitteln möchten.

Um mehr über die möglichen Properties der von Ihnen eingesetzten Hardware zu erfahren, beachten Sie bitte die ME-iDS Hilfe-Datei (*.chm), welche Sie über das „ME-iDS Control Center“ oder das Windows-Startmenü aufrufen können.

Hinweis

Diese Funktion steht ab ME-iDS Rev. 2.0 zur Verfügung.

● Funktions-Deklaration

```
int mePropertySetIntA(char* pcPropertyPath, int iValue);
int mePropertySetIntW(wchar_t* pcPropertyPath, int iValue);
```

<pcPropertyPath> (w)

Zeiger auf Property-Pfad auf den zugegriffen werden soll.

<iValue>

Die Property wird als Integer-Wert übergeben.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_PROPERTY_CONTAINER: Es wurde versucht, einen Wert auf einen Container zu schreiben.
- ME_ERRNO_PROPERTY_DATA_TYPE: Der Wert im angegebenen Property-Pfad kann nicht als Integer geschrieben werden.
- ME_ERRNO_PROPERTY_PATH: Die Angabe des Property-Pfades ist nicht gültig.
- ME_ERRNO_PROPERTY_INDEX: Der angegebene Index im Property-Pfad ist außerhalb des zulässigen Bereichs.

- ME_ERRNO_PROPERTY_UNKNOWN: Die angegebene Property ist unbekannt.
- ME_ERRNO_PROPERTY_SELECTION_INVALID: Der angegebene Wert ist nicht Teil der gültigen Defines
- ME_ERRNO_PROPERTY_VALUE_INVALID: Der Wert liegt außerhalb des zulässigen Bereiches. Mit den Attributen *MinValue* und *MaxValue* können die zulässigen Minimal- und Maximalwerte ermittelt werden.
- ME_ERRNO_PROPERTY_READ_ONLY: Die angegebene Property ist nur lesbar

mePropertySetDoubleA mePropertySetDoubleW

Beschreibung

Mit dieser Funktion können Sie Properties vom Typ ME_PROPERTY_TYPE_DOUBLE über den sog. Property-Pfad ändern. Verwenden Sie zunächst die Funktion *mePropertyGetIntA()*, falls Sie den unbekannten Typ einer Property (*PropertyType*) ermitteln möchten.

Um mehr über die möglichen Properties der von Ihnen eingesetzten Hardware zu erfahren, beachten Sie bitte die ME-iDS Hilfe-Datei (*.chm), welche Sie über das „ME-iDS Control Center“ oder das Windows-Startmenü aufrufen können.

Hinweis

Diese Funktion steht ab ME-iDS Rev. 2.0 zur Verfügung.

● Funktions-Deklaration

```
int mePropertySetDoubleA(char* pcPropertyPath, double dValue);
int mePropertySetDoubleW(wchar_t* pcPropertyPath, double dValue);
```

<pcPropertyPath> (w)

Zeiger auf Property-Pfad auf den zugegriffen werden soll.

<dValue>

Die Property wird als Double-Wert übergeben.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_PROPERTY_CONTAINER: Es wurde versucht, einen Wert auf einen Container zu schreiben.
- ME_ERRNO_PROPERTY_DATA_TYPE: Der Wert im angegebenen Property-Pfad kann nicht als Double geschrieben werden.
- ME_ERRNO_PROPERTY_PATH: Die Angabe des Property-Pfades ist nicht gültig.
- ME_ERRNO_PROPERTY_INDEX: Der angegebene Index im Property-Pfad ist außerhalb des zulässigen Bereichs.

- ME_ERRNO_PROPERTY_UNKNOWN: Die angegebene Property ist unbekannt.
- ME_ERRNO_PROPERTY_VALUE_INVALID: Der Wert liegt außerhalb des zulässigen Bereiches. Mit den Attributen *MinValue* und *MaxValue* können die zulässigen Minimal- und Maximalwerte ermittelt werden.
- ME_ERRNO_PROPERTY_READ_ONLY: Die angegebene Property ist nur lesbar

mePropertySetStringA **mePropertySetStringW**

Beschreibung

Mit dieser Funktion können Sie Properties vom Typ ME_PROPERTY_TYPE_STRING über den sog. Property-Pfad ändern. Verwenden Sie zunächst die Funktion *mePropertyGetIntA()*, falls Sie den unbekannten Typ einer Property (*PropertyType*) ermitteln möchten.

Um mehr über die möglichen Properties der von Ihnen eingesetzten Hardware zu erfahren, beachten Sie bitte die ME-iDS Hilfe-Datei (*.chm), welche Sie über das „ME-iDS Control Center“ oder das Windows-Startmenü aufrufen können.

Hinweis

Diese Funktion steht ab ME-iDS Rev. 2.0 zur Verfügung.

● Funktions-Deklaration

```
int mePropertySetStringA(char* pcPropertyPath, char* pcValue);  
int mePropertySetStringW(wchar_t* pcPropertyPath, wchar_t* pcValue);
```

<pcPropertyPath> (w)

Zeiger auf Property-Pfad auf den zugegriffen werden soll.

<pcValue> (w)

Es wird ein Zeiger auf die Property als String mit terminierendem Nullzeichen übergeben.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_PROPERTY_CONTAINER: Es wurde versucht, einen Wert auf einen Container zu schreiben.
- ME_ERRNO_PROPERTY_DATA_TYPE: Der Wert im angegebenen Property-Pfad kann nicht als String geschrieben werden.
- ME_ERRNO_PROPERTY_PATH: Die Angabe des Property-Pfades ist nicht gültig.
- ME_ERRNO_PROPERTY_INDEX: Der angegebene Index im Property-Pfad ist außerhalb des zulässigen Bereichs.

- ME_ERRNO_PROPERTY_UNKNOWN: Die angegebene Property ist unbekannt.
- ME_ERRNO_PROPERTY_VALUE_INVALID: Der String ist zu lang. Mit dem Attribut *MaxLength* kann die erforderliche Länge des Puffers in Zeichen (nicht Bytes), ermittelt werden, einschließlich des terminierenden Nullzeichens.
- ME_ERRNO_PROPERTY_READ_ONLY: Die angegebene Property ist nur lesbar

4.2.3 Ein-/Ausgabe-Funktionen

meIOResetDevice

Beschreibung

Das Gerät wird zurückgesetzt. Alle laufenden Operationen des spezifizierten Geräts (Device) werden abgebrochen:

- Alle Hardware-Aktionen werden gestoppt
- Die Hardware wird in den Grundzustand versetzt
- Interne Zustände werden rückgesetzt
- Pufferspeicher werden geleert
- Interrupt-Zähler werden auf Null gesetzt

● Funktions-Deklaration

```
int meIOResetDevice(int iDevice, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iFlags>

- ME_IO_RESET_DEVICE_NO_FLAGS

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_LOCKED: Gerät oder einige Subdevices sind gesperrt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt

meIOResetSubdevice

Beschreibung

Das Subdevice wird zurückgesetzt. Alle laufenden Operationen des spezifizierten Subdevices werden abgebrochen:

- Alle Hardware-Aktionen werden gestoppt
- Die Hardware wird in den Grundzustand versetzt
- Interne Zustände werden rückgesetzt
- Pufferspeicher werden geleert
- Interrupt-Zähler werden auf Null gesetzt

● Funktions-Deklaration

```
int meIOResetSubdevice(int iDevice, int iSubdevice, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iFlags>

- ME_IO_RESET_SUBDEVICE_NO_FLAGS

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_LOCKED: Subdevice ist gesperrt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt.

meIOIrqStart

Beschreibung

Diese Funktion startet den Interrupt-Handler für das gewünschte Subdevice. Sie können Interruptquelle, Triggerflanke, Referenz-Bitmuster, etc. auswählen.

Bei Bedarf können Sie mit der Funktion *meIOIrqSetCallback()* eine benutzerdefinierte Callback-Funktion installieren, die bei jedem Interrupt aufgerufen wird.

● Funktions-Deklaration

```
int meIOIrqStart(int iDevice, int iSubdevice, int iChannel, int iIrqSource,
                int iIrqEdge, int iIrqArg, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iChannel>

Index des Interrupt-Kanals innerhalb des gewählten Subdevices. Falls nicht zutreffend auf 0 setzen.

<iIrqSource>

Auswahl der Interruptquelle:

- ME_IRQ_SOURCE_DIO_LINE
Interruptquelle ist ein dedizierter, externer Interrupt-Eingang.
<iIrqArg> ist ohne Bedeutung und muss 0 sein. <iIrqEdge> kann verwendet werden.
- ME_IRQ_SOURCE_DIO_PATTERN (nur für digitale Eingangsports) Betriebsart „Bitmuster-Vergleich“ (z. B.: ME-5810, ME-8100/8200):
Wenn das aktuelle Bitmuster am Digitalport mit dem Referenz-Bitmuster, das in <iIrqArg> übergeben werden muss, übereinstimmt, wird ein Interrupt ausgelöst. <iIrqEdge> ist ohne Bedeutung und muss auf ME_IRQ_EDGE_NOT_USED gesetzt werden.
- ME_IRQ_SOURCE_DIO_MASK (nur für digitale Eingangsports) Betriebsart „Bitmuster-Änderung“ (z. B.: ME-5100/5810, ME-8100/8200):
Bei Zustandsänderung von mindestens einem als „sensitiv“ maskierten Bit wird ein Interrupt ausgelöst. Das Referenz-Bitmuster wird mit Parameter <iIrqArg> übergeben. <iIrqEdge> kann verwendet werden.

- ME_IRQ_SOURCE_DIO_OVER_TEMP
Bei Überhitzung des Treiberbausteins wird ein Interrupt ausgelöst (z. B.: ME-5810/8200). `<iIrqEdge>` ist ohne Bedeutung und muss auf ME_IRQ_EDGE_NOT_USED gesetzt werden.
`<iIrqArg>` ist ohne Bedeutung und muss auf 0 gesetzt werden.

`<iIrqEdge>`

Auswahl der Flanke mit der ein Interrupt ausgelöst werden soll:

- ME_VALUE_NOT_USED
Auswahl einer Flanke nicht unterstützt.
- ME_IRQ_EDGE_RISING
Interrupt durch steigende Flanke.
- ME_IRQ_EDGE_FALLING
Interrupt durch fallende Flanke.
- ME_IRQ_EDGE_ANY
Interrupt durch steigende oder fallende Flanke.

`<iIrqArg>`

Argument zur Konfiguration der Modi „Bitmuster-Vergleich“ (ME_IRQ_SOURCE_DIO_PATTERN) und „Bitmuster-Änderung“ (ME_IRQ_SOURCE_DIO_MASK). In allen anderen Fällen übergeben Sie hier 0.

Falls Sie im Parameter `<iIrqSource>` die Konstante ME_IRQ_SOURCE_DIO_LINE gewählt haben, ist dieser Parameter nicht relevant. Übergeben Sie in diesem Fall ME_VALUE_NOT_USED.

Beispielsweise gilt für die Karten vom Typ ME-8100/8200: Je nach Auswahl der Interruptquelle im Parameter `<iIrqSource>` wird ein Referenz-Bitmuster in das entsprechende Register geschrieben. Die Breite des Bitmusters wird durch den Parameter `<iFlags>` bestimmt.

- „Interrupt bei Bitmuster-Gleichheit“:
Das Argument (Referenz-Bitmuster) wird ins Vergleichsregister geschrieben. Falls das aktuelle Bitmuster am Digitalport dem Referenz-Bitmuster entspricht wird ein Interrupt ausgelöst.
- „Interrupt bei Bitmuster-Änderung“:
Das Argument (Referenz-Bitmuster) wird ins Maskenregister geschrieben. Bei Zustandsänderung ($0 \rightarrow 1$ oder $1 \rightarrow 0$) von mindestens einem Bit, das im Maskenregister auf „1“ gesetzt wurde, wird ein Interrupt ausgelöst. Das Bitmuster am Digital-Port, welcher den Interrupt ausgelöst hat, kann über den Parameter `<piValue>` der Funktion `meIOIrqWait()` („BLOCKING“-Mode) oder mit Hilfe der Callback-Funktion `meIOIrqSetCallback()` abgefragt werden.

<iFlags>

- ME_IO_IRQ_START_NO_FLAGS
Gültig sofern keine der folgenden Konstanten gewählt wird. Grundeinstellung wird verwendet.
- ME_IO_IRQ_START_DIO_BIT
Das Referenz-Bitmuster ist ein Bit breit.
- ME_IO_IRQ_START_DIO_BYTE
Das Referenz-Bitmuster ist ein Byte breit (8 Bit).
- ME_IO_IRQ_START_DIO_WORD
Das Referenz-Bitmuster ist ein Wort breit (16 Bit).
- ME_IO_IRQ_START_DIO_DWORD
Das Referenz-Bitmuster ist ein Langwort breit (32 Bit).
- ME_IO_IRQ_START_PATTERN_FILTERING (nur für „Bitmuster-Vergleich“) Erlaubt eine Filterung des Ergebnisses.
Der Modus „Bitmuster-Vergleich“ kann unter Umständen viele falsche Interrupts generieren. Theoretisch kann sich der Zustand von mehreren Bits eines „Multibit-Ports“ gleichzeitig ändern. In der Praxis gibt es jedoch immer eine (mehr oder weniger große) Verschiebung zwischen den sich ändernden Bits. Da der Bitmuster-Vergleich asynchron und sehr schnell arbeitet, können auch Zwischenzustände gespeichert werden.
Beispiel: Wenn sich zwei Bits von 00b nach 11b ändern, können auch die Zwischenzustände 01b und 10b vom Komparator erkannt werden. Falls der Filter aktiviert ist, werden in der Interrupt-Routine der aktuelle Portzustand mit dem Referenz-Bitmuster verglichen.
- ME_IO_IRQ_START_EXTENDED_STATUS (nur für „Bitmuster-Änderung“): Verwende das erweiterte IRQ-Status-Format. Siehe auch Funktion *meIOIrqWait()*.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_CHANNEL: Kein Kanal auf Subdevice verfügbar.
- ME_ERRNO_INVALID_IRQ_SOURCE: Falscher Modus oder nicht vom Subdevice unterstützt.
- ME_ERRNO_INVALID_IRQ_EDGE: Flanke wird nicht unterstützt.

- ME_ERRNO_INVALID_IRQ_ARG: Falsches Konfigurations-Argument / Argument nicht unterstützt.
- ME_ERRNO_LOCKED: Subdevice ist gesperrt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt.
- ME_ERRNO_START_THREAD: Callback-Thread generieren nicht möglich (nur Windows)

meIOIrqStop

Beschreibung

Mit dieser Funktion wird der Interrupt-Handler beendet.

- Jegliche mit Interrupt in Verbindung stehende Aktion wird abgebrochen
- Interrupts werden deaktiviert (in Hardware und Betriebssystem)

● Funktions-Deklaration

```
int meIOIrqStop(int iDevice, int iSubdevice, int iChannel, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iChannel>

Index des Interrupt-Kanals innerhalb des gewählten Subdevices. Falls nicht zutreffend auf 0 setzen.

<iFlags>

- ME_IO_IRQ_STOP_NO_FLAGS

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_CHANNEL: Kein Kanal auf Subdevice verfügbar.
- ME_ERRNO_LOCKED: Subdevice ist gesperrt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt

meIOIrqWait

Beschreibung

Diese Funktion „wartet“ (blockiert) solange bis ein Interrupt eintrifft und dient der Auswertung eines Interrupt-Ereignisses. Dies erfolgt völlig unabhängig von *meIOIrqStart()*. In „Multi-Threading“-Applikationen besteht keine Notwendigkeit zur Synchronisation. *meIOIrqWait()* kann vor oder nach *meIOIrqStart()* aufgerufen werden und wartet auf das erste Interrupt-Event. Falls ein Interrupt eintrifft, bevor *meIOIrqWait()* aufgerufen wurde, kehrt die Funktion sofort zurück und meldet dies.

Sie müssen sich vor Aufruf der Funktion *meIOIrqStart()* entscheiden, ob Sie mit *meIOIrqSetCallback()* eine Callback-Funktion installieren möchten.

● Funktions-Deklaration

```
int meIOIrqWait(int iDevice, int iSubdevice, int iChannel, int *piIrqCount,
               int *piValue, int iTimeOut, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des Interrupt-Subdevices.

<iChannel>

Index des Interrupt-Kanals innerhalb des gewählten Subdevices vom Typ ME_TYPE_EXT_IRQ. Falls nicht zutreffend auf 0 setzen.

<piIrqCount> (r)

Parameter gibt die Anzahl der Interrupts vom spezifizierten Interrupt-Kanal seit Start. *meIOResetDevice()* und *meIOResetSubdevice()* löschen diesen Zähler.

<piValue> (r)

Parameter gibt den Interrupt-Status zurück. Es gibt zwei Formate (die im Parameter <iFlags> konfiguriert werden):

- „Einfaches Format“: Ein Statusbit je Interruptleitung. Das Statusbit wird gesetzt, sobald die korrespondierende Leitung einen Interrupt ausgelöst hat. Es können auch mehrere Statusbits (b15...0) gesetzt sein, z. B. in den Betriebsarten Bitmuster-Vergleich und Bitmuster-Änderung.
- „Erweitertes Format“: Zwei Statusbits je Interruptleitung. Ein Statusbit für eine steigende Flanke (0 → 1) und eines für eine fallende Flanke (1 → 0). Die Bits für die fallenden Flanken sind dem niederwertigen „Word“ (b15...0) zugeordnet, die steigenden Flanken dem höherwertigen „Word“ (b31...16). Interessant z. B. in der Betriebsart Bitmuster-Änderung.

<iTimeout>

Zeit in Millisekunden innerhalb der das Interruptereignis eintreffen muss, ansonsten wird die Funktion beendet. Falls Sie kein Time-Out nutzen, übergeben Sie hier den Wert 0.

<iFlags>

- ME_IO_IRQ_SET_CALLBACK_NO_FLAGS
Keine Flags – Standardeinstellung wird verwendet
- ME_IO_IRQ_WAIT_NORMAL_STATUS
„Einfaches Format“ für den Interrupt-Status verwenden (siehe <piValue>)
- ME_IO_IRQ_WAIT_EXTENDED_STATUS
„Erweitertes Format“ für den Interrupt-Status verwenden (siehe <piValue>)

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebene Zeiger sind NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_CHANNEL: Kein Kanal auf Subdevice verfügbar.
- ME_ERRNO_LOCKED: Subdevice ist gesperrt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt.
- ME_ERRNO_CANCELLED: Subdevice wurde zurückgesetzt.
- ME_ERRNO_SIGNAL: Treiber ist nicht mehr geladen.
- ME_ERRNO_PREVIOUS_CONFIG: Subdevice wurde falsch konfiguriert

meIOIrqSetCallback

Beschreibung

Mit dieser Funktion können Sie eine Callback-Funktion installieren, die im Hintergrund auf einen Interrupt wartet.

- **Windows:**
 - Callback-Funktion ist abhängig von *meIOIrqStart()* und *meIOIrqStop()*.
 - Die Hintergrund-Task wird durch Ausführung der Funktion *meIOIrqStart()* generiert und durch die Funktion *meIOIrqStop()* beendet. Deshalb kann *meIOIrqSetCallback()* nur vor *meIOIrqStart()* aufgerufen werden.
 - Callback-Funktion arbeitet genauso wie *meIOIrqWait()* jedoch ohne Flags (ME_IO_IRQ_WAIT_NORMAL_STATUS und ME_IO_IRQ_WAIT_EXTENDED_STATUS werden nicht unterstützt).
 - Je Subdevice kann nur eine Callback-Funktion installiert werden.
- **Linux:**
 - Hintergrund-Task wird durch Ausführung der Funktion *meIOIrqSetCallback()* generiert.
 - Callback-Funktion ist vollkommen unabhängig von *meIOIrqStart()* und *meIOIrqStop()*.
 - *meIOIrqSetCallback()* kann vor oder nach *meIOIrqStart()* aufgerufen werden.
 - Callback-Funktion arbeitet genauso wie *meIOIrqWait()*. ME_IO_IRQ_WAIT_NORMAL_STATUS und ME_IO_IRQ_WAIT_EXTENDED_STATUS werden unterstützt.
 - Je Subdevice können mehrere Callback-Funktionen installiert werden.

Zum Deinstallieren der Callback-Funktion (alle registrierten Instanzen) für das gewählte Subdevice rufen Sie die Funktion *meIOIrqSetCallback()* auf und übergeben Sie im Parameter `<pCallback>` den Wert NULL.

● Funktions-Deklaration

```
int meIOIrqSetCallback(int iDevice, int iSubdevice, meIOIrqCB_t
    pCallback, void *pCallbackContext, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des Interrupt-Subdevices.

<pCallback>

Zeiger auf benutzerdefinierte Callback-Funktion, die aufgerufen wird, sobald ein Interrupt eingetroffen ist. Falls die Funktion mit einem Rückgabewert ungleich ME_ERNNO_SUCCESS (0x00) beendet wird, wird die Interruptverarbeitung umgehend gestoppt (*meIOIrqStop()* wird ausgeführt).

<pCallbackContext> (w)

Benutzerdefinierter Zeiger, der an die Callback-Funktion übergeben wird. Falls Sie diesen Parameter nicht verwenden wollen übergeben Sie NULL.

<iFlags>

- ME_IO_IRQ_SET_CALLBACK_NO_FLAGS
Keine Flags – Standardeinstellung wird verwendet
- ME_IO_IRQ_WAIT_NORMAL_STATUS (nur Linux)
„Einfaches Format“ für Interrupt-Status verwenden (siehe Parameter <iValue>)
- ME_IO_IRQ_WAIT_EXTENDED_STATUS (nur Linux)
„Erweitertes Format“ für Interrupt-Status verwenden (siehe Parameter <iValue>)

→ Typ-Definition meIOIrqCB_t

```
typedef int (*meIOIrqCB_t) (
    int iDevice,
    int iSubdevice,
    int iChannel,
    int iIrqCount,
    int iValue,
    void *pvContext,
    int iErrorCode);
```

<iDevice>

Index des spezifizierten Geräts.

<iSubdevice>

Index des Interrupt-Subdevices.

<iChannel>

Index des Interruptkanals innerhalb des gewählten Subdevices vom Typ ME_TYPE_EXT_IRQ, ansonsten muss 0 übergeben werden.

<iIrqCount>

Parameter gibt die Anzahl der Interrupts vom spezifizierten Interruptkanal seit dem Starten zurück. Siehe auch Parameter <piIrqCount> der Funktion *meIOIrqWait()*.

<iValue>

Parameter gibt den Interrupt-Status zurück. Es gibt zwei Formate (die im Parameter **<iFlags>** konfiguriert werden):

- „Einfaches Format“: Ein Statusbit je Interruptleitung. Das Statusbit wird gesetzt, sobald die korrespondierende Leitung einen Interrupt ausgelöst hat. Es können auch mehrere Statusbits (b15...0) gesetzt sein, z. B. in den Betriebsarten Bitmuster-Vergleich und Bitmuster-Änderung.
- „Erweitertes Format“: Zwei Statusbits je Interruptleitung. Ein Statusbit für eine steigende Flanke (0 → 1) und eines für eine fallende Flanke (1 → 0). Die Bits für die fallenden Flanken sind dem niederwertigen „Word“ (b15...0) zugeordnet, die steigenden Flanken dem höherwertigen „Word“ (b31...16). Interessant z. B. in der Betriebsart Bitmuster-Änderung.

<pvContext> (w)

Benutzerdefinierter Zeiger **<pCallbackContext>**. Falls Sie diesen Parameter nicht verwenden wollen übergeben Sie NULL.

<iErrorCode>

Fehlercode: siehe Fehlermeldung durch *meIOIrqWait()*.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebene Zeiger sind NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_CHANNEL: Kein Kanal auf Subdevice v verfügbar.
- ME_ERRNO_LOCKED: Subdevice ist gesperrt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt.
- ME_ERRNO_START_THREAD: Callback-Thread generieren nicht möglich. (nur Linux)
- -ENOMEM: Nicht genügend Speicher verfügbar. (nur Linux)
- ME_ERRNO_THREAD_RUNNING: Callback-Thread läuft bereits. (nur Windows)

meIOSetChannelOffset

Beschreibung

Durch Verwendung der Funktion *meIOSetChannelOffset()* kann der analoge Eingangsbereich verschoben werden. Falls Sie die Funktion *meIOSetChannelOffset()* zur Offset-Einstellung verwenden, müssen Sie das Endergebnis durch Addition des entsprechenden Offsetwertes korrigieren.

Hinweis!

Die Offset-Einstellung ist derzeit nur im Streaming-Betrieb möglich. Im Single-Betrieb wird stets der Offset 0,0 Volt verwendet!

Spezifische Informationen zum Betrieb des MEphisto Scope finden Sie im Kapitel B4 auf Seite 230.

● Funktions-Deklaration

```
int meIOSetChannelOffset(int iDevice, int iSubdevice, int iChannel, int
                        iRange, double *pdOffset, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iChannel>

Index des Kanals dessen Offset eingestellt werden soll.

<iRange>

Index (0...6) des Messbereichs, der für die Messung verwendet werden soll. Siehe auch Funktionen *meQueryNumberRanges()*, *meQueryRangeByMinMax()* und *meQueryRangeInfo()*.

<pdOffset> (r/w)

(w) : Zeiger auf einen Double-Wert, der den gewünschten Offset [V] enthält.

(r) : Falls der gewünschte Offsetwert durch die Hardware nicht exakt umgesetzt werden kann, wird der tatsächlich eingestellte Wert zurückgegeben.

<iFlags>

Flag für erweiterte Funktionen:

- ME_IO_SET_CHANNEL_OFFSET_NO_FLAGS
Keine Flags – Standardeinstellung wird verwendet

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.

- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_CHANNEL: Kein Kanal auf Subdevice verfügbar.
- ME_ERRNO_LOCKED: Subdevice ist gesperrt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt

meIOSingleConfig

Beschreibung

Diese Funktion bereitet ein Subdevice (A/D-, D/A-, Digital-I/O-, Frequenz-Ein-/Ausgabe oder Zähler) für eine „Einzelooperation“ vor. Der Start erfolgt grundsätzlich nach Aufruf der Funktion *meIOSingle()*, gemäß den in dieser Funktion beschriebenen Triggerbedingungen.

Hinweis!

Bei differentieller Messung können nur bipolare Eingangsbereiche verwendet werden!

Selbstverständlich stehen die vielfältigen Triggerarten nur zur Verfügung, sofern die Hardware die entsprechende Properties bietet.

● Funktions-Deklaration

```
int meIOSingleConfig(int iDevice, int iSubdevice, int iChannel, int
                    iSingleConfig, int iRef, int iTrigChan, int iTrigType, int iTrigEdge,
                    int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iChannel>

Kanal-Index. Siehe Kapitel Single-Betrieb ab Seite 45 für Details.

<iSingleConfig>

Grundlegende Konfiguration von Kanälen für die Betriebsart Single. Siehe auch Kapitel 3.4.1 ab Seite 45.

- *Analog-Eingangsbereich*. Übergeben Sie den Bereich, der von den Query-Funktionen (*meQuery...*) zurückgegeben wird.
- *Analog-Ausgangsbereich*. Übergeben Sie den Bereich, der von den Query-Funktionen (*meQuery...*) zurückgegeben wird.

Konfiguration der Digital-Ports sofern durch jeweilige Hardware unterstützt (siehe Hardware-Handbuch):

- ME_SINGLE_CONFIG_DIO_INPUT
Spezifizierter Digital-Port wird als Eingang konfiguriert.
- ME_SINGLE_CONFIG_DIO_OUTPUT
Spezifizierter Digital-Port wird als Ausgang konfiguriert.

Hinweis: Für die ME-5810/8100 müssen Sie statt obiger Konstanten eine der folgenden Konstanten wählen:

- ME_SINGLE_CONFIG_DIO_HIGH_IMPEDANCE
Spezifizierter Digital-Ausgangsport wird hochohmig geschaltet (z. B.: ME-5810/8100).
- ME_SINGLE_CONFIG_DIO_SINK
Aktivieren der „Sink“-Treiber (low-aktiv) für spezifizierten Digital-Ausgangsport (z. B.: ME-5810/8100).
- ME_SINGLE_CONFIG_DIO_SOURCE
Aktivieren der „Source“-Treiber (high-aktiv) für spezifizierten Digital-Ausgangsport (z. B.: ME-5810/8100).
- ME_SINGLE_CONFIG_DIO_BIT_PATTERN
Spezifizierter Digital-Ausgangsport (Subdevice vom Typ „ME_TYPE_DO“ oder „ME_TYPE_DIO“) wird hiermit für die zeitgesteuerte Bitmuster-Ausgabe konfiguriert. Siehe auch Parameter <iRef> dieser Funktion und Parameter <iFlags> der Funktion *meIOStreamConfig()*. Eine detaillierte Beschreibung finden Sie im Anhang B3 auf Seite 228.

Die folgenden beiden Konstanten werden nur benötigt, falls das *ME-MultiSig-System* nicht im ME-iDC angemeldet werden soll (in Vorbereitung):

- ME_SINGLE_CONFIG_DIO_MUX32M
Spezifizierter Digital-Ausgangsport wird für zeitgesteuerten MUX-Betrieb (Streaming-Betrieb) des ME-MultiSig-Systems in Verbindung mit der ME-4680 verwendet. Siehe auch Parameter <iRef> dieser Funktion.
- ME_SINGLE_CONFIG_DIO_DEMUX32
Spezifizierter Digital-Ausgangsport wird für zeitgesteuerten DEMUX-Betrieb (Streaming-Betrieb) des ME-MultiSig-Systems in Verbindung mit der ME-4680 verwendet. Siehe auch Parameter <iRef> dieser Funktion.

Konfiguration für Frequenz-Ein-/Ausgabe :

- ME_SINGLE_CONFIG_FIO_INPUT
Spezifizierter Kanal (Subdevice vom Typ „ME_TYPE_FI“ oder „ME_TYPE_FIO“) soll als Eingang für Frequenzmessung verwendet werden.
- ME_SINGLE_CONFIG_FIO_OUTPUT
Spezifizierter Kanal (Subdevice vom Typ „ME_TYPE_FO“ oder „ME_TYPE_FIO“) soll als Impulsgenerator-Ausgang verwendet werden.

Betriebsart für *Zähler vom Typ 8254* (eine detaillierte Beschreibung der Modi finden Sie ab Seite 222):

- ME_SINGLE_CONFIG_CTR_8254_MODE_0
"Zustandsänderung bei Nulldurchgang"
- ME_SINGLE_CONFIG_CTR_8254_MODE_1
"Retriggerbarer One-Shot"
- ME_SINGLE_CONFIG_CTR_8254_MODE_2
"Asymmetrischer Teiler"
- ME_SINGLE_CONFIG_CTR_8254_MODE_3
"Symmetrischer Teiler"
- ME_SINGLE_CONFIG_CTR_8254_MODE_4
"Zählerstart durch Softwaretrigger"
- ME_SINGLE_CONFIG_CTR_8254_MODE_5
"Zählerstart durch Hardwaretrigger"

<iRef>

Für manche Kanäle muss der Masse-Bezug explizit definiert werden: (siehe auch Kapitel Single-Betrieb auf Seite 45 für Details):

- ME_REF_NONE
Standardeinstellung (z. B. für digitale Standard-Ein-/Ausgabe und Frequenz-Ein-/Ausgabe)
- ME_REF_AI_GROUND
Single-ended Messung mit Bezug zur Masse des A/D-Teils.
- ME_REF_AI_DIFFERENTIAL
Differentielle Messung ohne direkten Masse-Bezug.
- ME_REF_AO_GROUND
Analoge Ausgabe mit Bezug auf Masse des D/A-Teils.
- ME_REF_AO_DIFFERENTIAL
Differentielle Ausgabe ohne direkten Masse-Bezug.

Folgende Konstanten definieren die Taktquelle der Zähler:

- ME_REF_CTR_PREVIOUS
Taktquelle ist der Ausgang des vorherigen Zählers.
- ME_REF_CTR_INTERNAL_1MHZ
Taktquelle ist der interne 1 MHz Quarzoszillator.
- ME_REF_CTR_INTERNAL_10MHZ
Taktquelle ist der interne 10 MHz Quarzoszillator.
- ME_REF_CTR_EXTERNAL
Taktquelle ist ein externer Taktgenerator.

Die folgenden beiden Konstanten sind relevant für Bitmuster-Ausgabe und ME-MultiSig-Betrieb, falls das System nicht im ME-iDC angemeldet werden soll (in Vorbereitung). Sie dienen der Zuordnung von „Low-Byte“ und „High-Byte“ der 16 Bit breiten FIFO-Werte zu den 8 Bit breiten Digital-Ports der ME-4680 (nur für Ausgangsports). Siehe auch Parameter `<iSingleConfig>`.

- ME_REF_DIO_FIFO_LOW
Low-Byte des FIFOs (Bit 7...0).
- ME_REF_DIO_FIFO_HIGH
High-Byte des FIFOs (Bit 15...8).

<iTrigChan>

Triggerkanal sofern vom Subdevice unterstützt, ansonsten ME_TRIG_CHAN_DEFAULT übergeben.

- ME_TRIG_CHAN_DEFAULT
Die Triggerung erfolgt für jeden Kanal getrennt.
- ME_TRIG_CHAN_SYNCHRONOUS
Dieser Kanal wird in die „Synchronliste“ einbezogen. Alle Kanäle werden in Abhängigkeit weiterer Triggeroptionen (z. B. Software-Start oder ext. Trigger) synchron gestartet. Siehe auch Kapitel 3.4.3.3 „Synchronstart“ auf Seite 88.

<iTrigType>

Triggertyp für Start der Ein-/Ausgabe (sofern vom Subdevice unterstützt, ansonsten ME_TRIG_TYPE_SW übergeben). Grundsätzlich wird die Wandlung nach Aufruf der Funktion *meIOSingle()*, gemäß den in dieser Funktion (*meIOSingleConfig()*) definierten Triggerbedingungen gestartet.

- ME_TRIG_TYPE_SW
Start unmittelbar nach Aufruf der Funktion *meIOSingle()*.
- ME_TRIG_TYPE_EXT_DIGITAL
Start durch externes, digitales Trigger-Signal.
- ME_TRIG_TYPE_EXT_ANALOG
Start durch externes, analoges Trigger-Signal.

<iTrigEdge>

Auswahl der passenden Triggerflanke (sofern vom Subdevice unterstützt, ansonsten übergeben Sie hier ME_VALUE_NOT_USED):

- ME_TRIG_EDGE_ABOVE
Start falls sich Pegel oberhalb Schwellwert befindet.
- ME_TRIG_EDGE_UNDER
Start falls sich Pegel unterhalb Schwellwert befindet.
- ME_TRIG_EDGE_ENTRY
Start, falls Wert in ein definiertes Fenster eintritt.
- ME_TRIG_EDGE_EXIT
Start, falls Wert aus einem definierten Fenster austritt.
- ME_TRIG_EDGE_RISING
Start durch steigende Flanke.
- ME_TRIG_EDGE_FALLING
Start durch fallende Flanke.
- ME_TRIG_EDGE_ANY
Start durch steigende oder fallende Flanke.

<iFlags>

Flag für erweiterte Funktionen:

- ME_IO_SINGLE_CONFIG_NO_FLAGS
Keine Flags – Standardeinstellung wird verwendet
- ME_IO_SINGLE_CONFIG_DIO_BIT
Die digitale Ein-/Ausgabe erfolgt bitweise.
- ME_IO_SINGLE_CONFIG_DIO_BYTE
Die digitale Ein-/Ausgabe erfolgt als Byte (8 Bit).
- ME_IO_SINGLE_CONFIG_DIO_WORD
Die digitale Ein-/Ausgabe erfolgt als Wort (16 Bit).
- ME_IO_SINGLE_CONFIG_DIO_DWORD
Die digitale Ein-/Ausgabe erfolgt als Langwort (32 Bit).
- ME_IO_SINGLE_CONFIG_CONTINUE
„Helfer“-Flag zur Übernahme der in dieser Funktion gemachten Einstellungen für den im Parameter <iChannel> übergebenen Kanal und alle darüber liegenden Kanäle. Z. B.: ein Subdevice für analoge Erfassung habe 32 Kanäle. In <iChannel> wird Index 4 übergeben und dieses Flag gesetzt, d. h. die Kanäle 4...31 werden identisch mitkonfiguriert. Dies hat den Vorteil, dass deutlich weniger Schreibzugriffe auf das Gerät nötig sind (insbesondere bei USB-Geräten).
- ME_IO_SINGLE_CONFIG_MULTIPIN (nur Linux)
Konfiguration des Mehrzweck-Pins der ME-1400 als Oszillator-Ausgang (OSC).

- ME_IO_SINGLE_CONFIG_FI_SINGLE_MODE
Frequenzmessung wird einmal ausgeführt. Standardeinstellung:
Frequenzmessung wird wiederholt ausgeführt (nicht empfohlen).
Siehe auch Kap. 3.4.1.4.1 auf Seite 51.

Die beiden folgenden Konstanten dienen dem Ein-/Ausschalten der Adress-LED auf MUX-Basiskarten (ME-MUX32-M/S) – siehe auch ME-MultiSig-Handbuch. Im Parameter `<iChannel>` muss ein beliebiger Kanal jener Basiskarte gewählt werden, deren Adress-LED geschaltet werden soll:

- ME_IO_SINGLE_CONFIG_MULTISIG_LED_ON
Adress-LED einschalten.
- ME_IO_SINGLE_CONFIG_MULTISIG_LED_OFF
Adress-LED ausschalten.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_CHANNEL: Kein Kanal auf Subdevice verfügbar.
- ME_ERRNO_LOCKED: Subdevice ist gesperrt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt.
- ME_ERRNO_INVALID_REF: Parameter `<iRef>` is not correct.
- ME_ERRNO_INVALID_TRIG_CHAN: Parameter `<iTrigChan>` ist nicht korrekt.
- ME_ERRNO_INVALID_TRIG_TYPE: Parameter `<iTrigType>` ist nicht korrekt.
- ME_ERRNO_INVALID_TRIG_EDGE: Parameter `<iTrigEdge>` ist nicht korrekt.
- ME_ERRNO_INVALID_SINGLE_CONFIG: Parameter `<iSingleConfig>` ist nicht korrekt.

meIOSingle

Beschreibung

Mit dieser Funktion können ein oder mehrere Schreib-/Lese-Operationen in Form einer Liste übergeben werden. Siehe auch Kapitel 3.4.1 Single-Betrieb ab Seite 45 für Details.

Hinweis!

Sofern für ein oder mehrere Einzeloperationen eine externe Triggerquelle gewählt wurde, wartet die Funktion bis das entsprechende Triggersignal eintrifft. D. h. falls Operation 1 mit externem Trigger im Blocking-Modus ausgeführt wird, wartet Operation 2 bis das externe Triggersignal für Operation 1 eingetroffen ist.

● Funktions-Deklaration

```
int meIOSingle(meIOSingle_t *pSingleList, int iCount, int iFlags);
```

<pSingleList> (r/w)

Zeiger auf eine Liste vom Typ `meIOSingle_t`. Jeder Eintrag steht für eine einzelne Schreib-Lese-Operation. Sofern für einen oder mehrere Listeneinträge eine externe Triggerquelle gewählt wurde, wartet ("blockiert") die Funktion bis das entsprechende Triggersignal eintrifft.

<iCount>

Anzahl der Einträge in `<pSingleList>`.

<iFlags>

- **ME_IO_SINGLE_NO_FLAGS**
Keine weiteren Optionen – Standardeinstellungen werden verwendet. Die Ausführung wird mit dem ersten Fehler beendet. Der Rückgabewert korrespondiert mit dem Array `<iErrno>` des zuletzt ausgeführten Eintrags von `<pSingleList>`.
- **ME_IO_SINGLE_NONBLOCKING** (nur Linux)
Die gesamte Liste ausführen. Sobald dieses Flag gesetzt wird, wird die Liste abgearbeitet auch wenn ein Fehler für einen oder mehrere Einträge aufgetreten sein sollte. Die Funktion gibt `ME_ERRNO_SUCCESS` zurück, sofern kein globaler Fehler auftrat. Das Array `<iErrno>` muss ausgewertet werden.

→ Typ-Definition meIOSingle

```
typedef struct meIOSingle {
    int iDevice;
    int iSubdevice;
    int iChannel;
    int iDir;
```

```

        int iValue;
        int iTimeout;
        int iFlags;
        int iErrno;
    } meIOSingle_t;

```

<iDevice> (w)

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice> (w)

Index des anzusprechenden Subdevices.

<iChannel> (w)

Kanal-Index bzw. Index der Gruppe. Die Größe einer Gruppe hängt vom verwendeten Flag im Parameter <iFlags> ab.

In Verbindung mit dem ME-MultiSig-System werden über diesen Parameter die MUX- bzw. DEMUX-Kanäle ausgewählt.

Falls Sie ein Subdevice mit nur einem Kanal verwenden, übergeben Sie bitte 0.

Beachten Sie, dass die Anzahl der Kanäle von Digital-I/O-Ports vom Parameter <iFlags> dieser Funktion abhängt. D. h. für einen 32 bit breiten Port gilt:

- Bit-Zugriff (...DIO_BIT): Kanal-Index 0...31
- Byte-Zugriff (...DIO_BYTE): Kanal-Index 0...3
- Wort-Zugriff (...DIO_WORD): Kanal-Index 0...1
- Langwort-Zugriff (...DIO_DWORD): Kanal-Index 0

Beispiel: Falls Sie für ein digitales Subdevice mit 32 Bits das Flag ME_IO_SINGLE_TYPE_DIO_BYTE übergeben können Sie auf vier Kanäle mit der Breite eines Bytes (8 Bits) zugreifen.

<iDir> (w)

- ME_DIR_INPUT Lesen/Erfassen
- ME_DIR_OUTPUT Schreiben/Ausgeben
- ME_DIR_SET_OFFSET Offset für Stromkanäle bestimmen
(siehe auch ME-Axon Handbuch)

<iValue> (r/w)

- **A/D:** Messwert wird als normierter Digitalwert zurückgegeben. Verwenden Sie die Funktion *meUtilityDigitalToPhysical()* zur Umrechnung des digitalen Wertes in die korrekte physikalische Einheit.
- **D/A:** Spannung/Strom als normierten Digitalwert übergeben. Verwenden Sie die Funktion *meUtilityPhysicalToDigital()* zur Umrechnung von Spannung bzw. Strom in den korrekten Digitalwert.
- **Digital-Ein-/Ausgabe:** 32 bit Digital-Wert wird eingelesen bzw ausgegeben. Je nach Portbreite sind stets die niederwertigen Bits relevant.

- **Frequenz-Ein-/Ausgabe:** Um Periodendauer und Dauer der ersten Phase der Periode einlesen bzw. ausgeben zu können, müssen Sie die Funktion *meIOSingle()* zweimal aufrufen. Je nach Wert im Parameter *<iFlags>* liefert bzw. übergibt *<iValue>* entweder die gesamte Periodendauer in Ticks (ME_IO_SINGLE_TYPE_FIO_TICKS_TOTAL) oder die Dauer der ersten Phase der Periode in Ticks (ME_IO_SINGLE_TYPE_FIO_TICKS_FIRST_PHASE) (siehe auch Kap. 3.4.1.4.1 auf Seite 51). Verwenden Sie die Funktionen *meIOSingleTicksToTime()* und *meIOSingleTimeToTicks()* zur bequemen Umrechnung von Ticks in Sekunden und umgekehrt.
- **Zähler:** Startwert schreiben bzw. Zählerstand einlesen.

<iTimeOut> (w)

Optional können Sie hier ein Zeitintervall in Millisekunden angeben innerhalb dessen der erste Triggerimpuls eintreffen muss. Ansonsten wird die Messung abgebrochen. Falls Sie ohne ext. Trigger arbeiten oder kein Time-Out nutzen möchten, übergeben Sie hier ME_VALUE_NOT_USED.

<iFlags> (w)

Erweiterte Einstellungen:

- ME_IO_SINGLE_TYPE_NO_FLAGS
Keine weiteren Optionen – Standardeinstellungen werden verwendet. Für Digital-Ports wird die „natürliche“ Breite verwendet.
- ME_IO_SINGLE_TYPE_DIO_BIT
Die digitale Ein-/Ausgabe erfolgt bitweise.
- ME_IO_SINGLE_TYPE_DIO_BYTE
Die digitale Ein-/Ausgabe erfolgt als Byte (8 bit).
- ME_IO_SINGLE_TYPE_DIO_WORD
Die digitale Ein-/Ausgabe erfolgt als Wort (16 bit).
- ME_IO_SINGLE_TYPE_DIO_DWORD
Die digitale Ein-/Ausgabe erfolgt als Langwort (32 bit).
- ME_IO_SINGLE_TYPE_TRIG_SYNCHRONOUS
Synchron-Trigger per Software-Start. Sinnvoll für letzten Kanal der *<pSingleList>* um alle Kanäle, die in der Synchronliste eingetragen wurden mit Aufruf dieser Funktion zu starten. Siehe Option ME_TRIG_CHAN_SYNCHRONOUS im Parameter *<iTrigChan>* der Funktion *meIOSingleConfig()*.
- ME_IO_SINGLE_TYPE_NONBLOCKING
(alias: ME_IO_SINGLE_TYPE_WRITE_NONBLOCKING)
Operation läuft im Hintergrund. D. h. die Funktion wartet nicht auf das Ende der Ausführung.
Hinweis: Nicht jedes Subdevice unterstützt den NONBLOCKING-Modus.

- ME_IO_SINGLE_TYPE_FIO_TICKS_TOTAL
Die Periodendauer in <iValue> einlesen bzw. ausgeben.
- ME_IO_SINGLE_TYPE_FIO_TICKS_FIRST_PHASE
Die Dauer der ersten Phase der Periode in <iValue> einlesen bzw. ausgeben.

Die *Frequenz-Messung* (Frequenzzähler) kann durch geeignete Kombination folgender Flags gesteuert werden. Verodern Sie dazu ME_IO_SINGLE_TYPE_FIO_TICKS_TOTAL bzw. ME_IO_SINGLE_TYPE_FIO_TICKS_FIRST_PHASE mit folgenden Optionen:

- ME_IO_SINGLE_TYPE_FI_LAST_VALUE
Flag zur wiederholten Erfassung langsamer Frequenzen (siehe auch Parameter <iFlags> der Funktion *meIOSingleConfig()*). Muß zusätzlich mit dem Flag ME_IO_SINGLE_TYPE_NONBLOCKING bitweise verodert werden.

Der Start der *Frequenz-Ausgabe* (Impulsgenerator) kann durch geeignete Kombination folgender Flags gesteuert werden. Verodern Sie dazu ME_IO_SINGLE_TYPE_FIO_TICKS_TOTAL bzw. ME_IO_SINGLE_TYPE_FIO_TICKS_FIRST_PHASE bitweise mit einer oder mehreren der folgenden Optionen:

- ME_IO_SINGLE_TYPE_FO_UPDATE_ONLY
Der Ausgabewert soll geändert aber nicht sofort ausgegeben werden. Keine Verknüpfung mit weiteren Flags möglich. Standard: der neue Wert wird sofort ausgegeben.
- ME_IO_SINGLE_TYPE_FO_START_SOFT
Der Wert wird erst mit Ende der aktuellen Periode (falls bereits gestartet) ausgegeben. Verknüpfung mit ME_IO_SINGLE_TYPE_FO_START_LOW möglich. Standard: das neue Signal wird sofort ausgegeben.
- ME_IO_SINGLE_TYPE_FO_START_LOW
Standardmäßig ist die erste Phase des Rechtecksignals „High“. Wird dieses Flag gesetzt, startet die Ausgabe mit „Low“-Pegel. Verknüpfung mit ME_IO_SINGLE_TYPE_FO_START_SOFT möglich.
- ME_IO_SINGLE_TYPE_TRIG_SYNCHRONOUS
Alle Subdevices, die mit Parameter <iTrigChan> der Funktion *meIOSingleConfig()* in die Synchronliste eingetragen wurden, werden simultan gestartet (siehe auch Kap. 3.4.3.3 auf Seite 88). Standard: Subdevice startet unabhängig.

<iErrno>

Fehlercode korrespondiert mit einem bestimmten Eintrag.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_CHANNEL: Kein Kanal auf Subdevice verfügbar.
- ME_ERRNO_LOCKED: Subdevice ist gesperrt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt.
- ME_ERRNO_TIMEOUT: Timeout-Bedingung eingetreten.
- ME_ERRNO_PREVIOUS_CONFIG: Subdevice wurde nicht für angeforderte Operation konfiguriert.
- ME_ERRNO_SUBDEVICE_BUSY: Subdevice führt andere Operation aus.

meIOSingleTicksToTime

Beschreibung

Konvertiert die Anzahl der „Ticks“ in die gewünschte Zeit, z. B. Periodendauer [s] zur bequemen Weiterverarbeitung in Ihrer Applikation (z. B. Frequenzmessung). In Abhängigkeit des Parameters `<iTimer>` dieser Funktion können die Rückgabewerte von `<piTicksLow>` bzw. `<piTicksHigh>` an den Parameter `<iValue>` der Funktion *meIOSingle()* übergeben werden.

Hinweis

Die Umrechnung und der erlaubte Wertebereich sind abhängig vom jeweiligen Subdevice und dessen Properties. Falls Hardwaregrenzen überschritten werden, werden stets die Grenzwerte zurückgegeben.

Tip: Wenn Sie die Größen Frequenz und Tastverhältnis benötigen, können Sie diese leicht aus den Rückgabewerten von `<pdTime>` berechnen. Es gilt:

- Frequenz [Hz] = 1/Periodendauer [s]
- Tastverhältnis [%] = („Dauer der ersten Phase der Periode“ [s] / Periodendauer [s]) × 100

● Funktions-Deklaration

```
int meIOSingleTicksToTime(int iDevice, int iSubdevice, int iTimer, int
    iTicksLow, int iTicksHigh, double *pdTime, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iTimer>

Die Zeit wird in Abhängigkeit vom Subdevice-Typ und des hier spezifizierten Timers berechnet.

- ME_TIMER_FIO_TOTAL
Periodendauer wird in Sekunden umgerechnet.
- ME_TIMER_FIO_FIRST_PHASE
Die Dauer der ersten Phase der Periode wird in Sekunden umgerechnet.

<iTicksLow>

Die Anzahl der Ticks (niederwertiger Teil, Bits 31...0) von Parameter `<iValue>` der Funktion *meIOSingle()* werden hier übergeben.

<iTicksHigh>

Die Anzahl der Ticks (höherwertiger Teil, Bits 63...32) von Parameter <iValue> der Funktion *meIOSingle()* werden hier übergeben. Dieser Parameter ist für zukünftige Erweiterungen reserviert.

<pdTime> (r)

(r) : Zeiger auf einen Double-Wert, der die errechnete Zeit in Sekunden, z. B. Periodendauer [s] zurückgibt.

<iFlags>

- ME_IO_SINGLE_TIME_TO_TICKS_NO_FLAGS (Standardeinstellung)

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebene Zeiger sind NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_TIMER: Timer nicht unterstützt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt

meIOSingleTimeToTicks

Beschreibung

Konvertiert die gewünschte Zeit, z. B. Periodendauer [s] in die Anzahl der „Ticks“ zur Übergabe an die Funktion *meIOSingle()* z.B. für Frequenzausgabe (Impulsgenerator). In Abhängigkeit des Parameters `<iTimer>` dieser Funktion können die Rückgabewerte von `<piTicksLow>` bzw. `<piTicksHigh>` an den Parameter `<iValue>` der Funktion *meIOSingle()* übergeben werden.

Hinweis

Die Umrechnung und der erlaubte Wertebereich sind abhängig vom jeweiligen Subdevice und dessen Properties. Falls Hardwaregrenzen überschritten werden, wird eine entsprechende Fehlermeldung zurückgegeben.

● Funktions-Deklaration

```
int meIOSingleTimeToTicks(int iDevice, int iSubdevice, int iTimer, double
    *pdTime, int *piTicksLow, int *piTicksHigh, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iTimer>

Ticks werden in Abhängigkeit vom Subdevice-Typ und des hier spezifizierten Timers berechnet. Die Werte `<piTicksLow>` und `<piTicksHigh>` können anschließend an Parameter `<iValue>` der Funktion *meIOSingle()* übergeben werden.

- ME_TIMER_FIO_TOTAL
Periodendauer wird in Ticks umgerechnet.
- ME_TIMER_FIO_FIRST_PHASE
Die Dauer der ersten Phase der Periode wird in Ticks umgerechnet.

<pdTime> (r/w)

(w) : Zeiger auf einen Double-Wert, der die gewünschte Zeit in Sekunden, z. B. Periodendauer [s] zur Umrechnung in Ticks übergibt. Bei Übergabe ungültiger Werte wird eine entsprechende Fehlermeldung zurückgegeben.

(r) : Falls die gewünschte Zeit durch die Hardware nicht exakt eingestellt werden kann, wird der nächstliegende Wert zurückgegeben. Die korrespondierenden Ticks werden in den Parametern `<piTicksLow>` und `<piTicksHigh>` zurückgegeben.

<piTicksLow> (r)

Zeiger auf einen Integerwert, der die niederwertigen 32 Bits (31...0) der errechneten Ticks enthält. Wird an den Parameter <iValue> der Funktion *meIOSingle()* übergeben.

<piTicksHigh> (r)

Zeiger auf einen Integerwert, der die höherwertigen 32 Bits (63...32) der errechneten Ticks enthält. Dieser Parameter ist für zukünftige Erweiterungen reserviert.

<iFlags>

- ME_IO_SINGLE_TIME_TO_TICKS_NO_FLAGS (Standardeinstellung)

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebene Zeiger sind NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_TIMER: Timer nicht unterstützt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt

meIOStreamConfig

Beschreibung

Diese Funktion konfiguriert die Hardware für eine timergesteuerte Operation. Siehe auch Kapitel 3.4.2 Streaming-Betrieb ab Seite 62.

Zur Konfiguration analoger und digitaler (Bitmuster-Ausgabe) Streaming-Operationen müssen Sie eine Kanalliste generieren, die für jeden Kanal einen Eintrag (Kanal-Nr., Messbereich...) vom Typ `meIOStreamConfig_t` enthält.

Daneben benötigen Sie eine Triggerstruktur (`meIOStreamTrigger_t`), die zahlreiche Einstellungen wie Start/Stop-Bedingungen, Timer-Einstellungen, Triggerquellen und Triggerflanken definiert, die für die gesamte Operation gelten.

Gestartet wird die Operation stets mit der Funktion `meIOStreamStart()` entweder sofort (Software-Start) oder gemäß den in der Funktion `meIOStreamConfig()` definierten Start-Bedingungen. Beendet wird die Operation entweder gemäß den in der Triggerstruktur definierten Stop-Bedingungen oder mit der Funktion `meIOStreamStop()`.

Hinweis:

Verwenden Sie die Funktionen `meIOStreamFrequencyToTicks()` und `meIOStreamTimeToTicks()` (siehe Seite 177ff) zur bequemen Umrechnung von Frequenz bzw. Periodendauer in Ticks zur Übergabe an die Timer.

Beachten Sie bitte Anhang B5 auf Seite 231 bei Einsatz des ME-MultiSig-Systems.

Tip: Initialisieren Sie die Triggerstruktur `meIOStreamTrigger_t` zunächst mit 0. Damit brauchen Sie sich nur um die von Ihnen benötigten Parameter kümmern und zugleich werden nicht zutreffende bzw. nicht benötigte Parameter automatisch korrekt übergeben.

● Funktions-Deklaration

```
int meIOStreamConfig(int iDevice, int iSubdevice, meIOStreamConfig_t
    *pConfigList, int iCount, meIOStreamTrigger_t *pTrigger, int
    iFIFOIrqThreshold, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<pConfigList>

Zeiger auf Liste vom Typ `meIOStreamConfig_t` (siehe unten).

<iCount>

Anzahl der Einträge in `<pConfigList>`.

<pTrigger>

Zeiger auf Struktur vom Typ `meIOStreamTrigger_t` (siehe unten).

<iFIFOIrqThreshold>

Anzahl der Werte, die als ein Paket gelesen bzw. geschrieben (nachgeladen) werden sollen. Durch Aktualisierung kann das Hardware-puffer-Event zur Benutzer/Hardware-Synchronisation verwendet werden. Falls dieses nicht gesetzt ist, wird das Hardware-FIFO gelesen bzw. geschrieben, sobald das „HALF FIFO“-Flag erkannt wird.

<iFlags>

- `ME_IO_STREAM_CONFIG_NO_FLAGS` (Standardeinstellung)
Zeiger auf eine Liste mit `<iCount>` Einträgen vom Typ `meIOStreamConfig_t` mit der Konfiguration der einzelnen Kanäle
 - Dieses Flag wird u.a. für den kontinuierlichen Streaming-Betrieb verwendet. Siehe Kapitel 3.4.2.6 auf Seite 79.
 - MEphisto Scope: Analoge Erfassung
 - `ME_IO_STREAM_CONFIG_BIT_PATTERN`
 - MEphisto Scope: Logikanalysator-Mode, der Parameter `<iCount>` wird ignoriert.
 - Sonderfunktionen Bitmuster-Ausgabe und FIFO-Umlenkung (z. B. zur Steuerung des ME-MultiSig-Systems). Die Verbindung vom D/A-Wandler zum FIFO wird getrennt. Eine detaillierte Beschreibung finden Sie im Kapitel 3.4.3.2 auf Seite 86.
- Hinweis:* Auch die Digital-Ports müssen entsprechend konfiguriert werden damit die Umlenkung wirksam wird (mit `ME_REF_DIO_FIFO_LOW` oder `ME_REF_DIO_FIFO_HIGH`).
- `ME_IO_STREAM_CONFIG_WRAPAROUND`
Flag für den „Wraparound-Modus“ (periodische Ausgabe). Siehe Kapitel 3.4.2.6.4 auf Seite 84.
 - `ME_IO_STREAM_CONFIG_HARDWARE_ONLY` (nur Linux)
Softwarepuffer wird deaktiviert. Verwendung nur in Verbindung mit `ME_IO_STREAM_CONFIG_WRAPAROUND` sinnvoll.
 - `ME_IO_STREAM_CONFIG_SAMPLE_AND_HOLD`
Aktiviert „Sample&Hold“-Option. Eine detaillierte Beschreibung finden Sie im Kapitel 3.4.3.1 „Sample & Hold“ auf Seite 86.
- ! Zur periodischen Bitmuster-Ausgabe und im periodischen DEMUX-Betrieb muss die Konstante `ME_IO_STREAM_CONFIG_WRAPAROUND` mit der Konstante `ME_IO_STREAM_CONFIG_BIT_PATTERN` „ODER“-verknüpft werden.

→ Typ-Definition meIOStreamConfig

```
typedef struct meIOStreamConfig {
    int iChannel;
    int iStreamConfig;
    int iRef;
    int iFlags;
} meIOStreamConfig_t;
```

<iChannel> (w)

Kanal-Index. Je nach Subdevice können dies analoge Ein- oder Ausgänge sein sowie eine Gruppe aus Digital-I/Os (z. B.: in der Betriebsart Bitmuster-Ausgabe).

<iStreamConfig> (w)

Wahl des Bereichs für den Kanallisten-Eintrag (siehe Kapitel 3.4.2 Streaming-Betrieb auf Seite 62):

- Index für Analog-Eingangsbereich. Übergeben Sie den mit den Query-Funktionen (*meQuery...*) ermittelten Index für den gewünschten Bereich.
- Index für Analog-Ausgangsbereich. Übergeben Sie den mit den Query-Funktionen (*meQuery...*) ermittelten Index für den gewünschten Bereich..
- Bei Bitmuster-Ausgabe übergeben Sie ME_VALUE_NOT_USED.

<iRef> (w)

Definiert Masse-Bezug für analoge Ein- und Ausgänge (siehe Kapitel 3.4.2 Streaming-Betrieb auf Seite 62):

- ME_REF_NONE
Standardeinstellung (z. B. für Bitmuster-Ausgabe)
- ME_REF_AI_GROUND
Single-ended Messung mit Bezug zur Masse des A/D-Teils.
- ME_REF_AI_DIFFERENTIAL
Differenzielle Messung ohne direkten Masse-Bezug.
- ME_REF_AO_GROUND
Ausgabe mit Bezug zur Masse des D/A-Teils. Verwenden Sie diese Konstante für analoge Ausgabe.
- ME_REF_AO_DIFFERENTIAL
Differenzielle Ausgabe ohne direkten Masse-Bezug.

<iFlags>

- ME_IO_STREAM_CONFIG_TYPE_NO_FLAGS (Standard)
(Keine Flags definiert)

→ Typ-Definition meIOStreamTrigger

```
typedef struct meIOStreamTrigger {
    int iAcqStartTrigType;
    int iAcqStartTrigEdge;
    int iAcqStartTrigChan;
    int iAcqStartTicksLow;
    int iAcqStartTicksHigh;
    int iAcqStartArgs[10];
    int iScanStartTrigType;
    int iScanStartTicksLow;
    int iScanStartTicksHigh;
    int iScanStartArgs[10];
    int iConvStartTrigType;
    int iConvStartTicksLow;
    int iConvStartTicksHigh;
    int iConvStartArgs[10];
    int iScanStopTrigType;
    int iScanStopCount;
    int iScanStopArgs[10];
    int iAcqStopTrigType;
    int iAcqStopCount;
    int iAcqStopArgs[10];
    int iFlags;
} meIOStreamTrigger_t;
```

! Beachten Sie auch die Beschreibung der Triggerstruktur ab Seite 64.

<iAcqStartTrigType>

Dieser Parameter definiert den Triggertyp für den Start der gesamten Operation. In Abhängigkeit von der verwendeten Hardware können Sie zwischen folgenden Optionen wählen:

- ME_TRIG_TYPE_SW
Start unmittelbar nach Aufruf der Funktion *meIOStreamStart()*.
Übergeben Sie 0 in <iAcqStartTrigEdge>.
- ME_TRIG_TYPE_EXT_ANALOG
Start der Operation durch geeignetes Signal am externen, analogen Triggereingang.
- ME_TRIG_TYPE_THRESHOLD
Start der Operation bei Über- oder Unterschreiten einer vorgegebenen

Schwelle am analogen Triggerkanal (siehe <iAcqStartArgs[0]>).

- ME_TRIG_TYPE_WINDOW
Start der Operation wenn das Signal am analogen Triggerkanal in das definierte Fenster eintritt oder dieses verläßt (siehe <iAcqStartArgs[0]> und <iAcqStartArgs[1]>).
- ME_TRIG_TYPE_EDGE
Start der Operation bei einer steigenden oder fallenden Flanke die einen vorgegebenen Pegel am Triggerkanal kreuzt (siehe <iAcqStartArgs[0]>) . Z. B. um ein stehendes Bild bei Wechselsignalen zu erhalten.
- ME_TRIG_TYPE_SLOPE
Start der Operation wenn das Signal am Triggerkanal schneller ansteigt oder abfällt als in <iAcqStartArgs[0]> definiert.
- ME_TRIG_TYPE_EXT_DIGITAL
Start der Operation durch geeignetes Signal am externen, digitalen Triggereingang.
- ME_TRIG_TYPE_PATTERN
Start der Operation wenn das Bitmuster am Trigger-Port dem definierten Referenz-Bitmuster (siehe <iAcqStartTrigChan>) entspricht. Übergeben Sie 0 in <iAcqStartTrigEdge>.

<iAcqStartTrigEdge>

Dieser Parameter definiert die Flanke für den Start einer einzelnen Wandlung durch ein externes Triggersignal. In Abhängigkeit von Triggertyp und verwendeter Hardware können Sie aus folgenden Optionen wählen:

- ME_TRIG_EDGE_NONE
...falls Sie im Parameter <iAcqStartTrigType> die Option Software-Start (ME_TRIG_TYPE_SW) gewählt haben.
- ME_TRIG_EDGE_RISING
Start durch steigende Flanke.
- ME_TRIG_EDGE_FALLING
Start durch fallende Flanke.
- ME_TRIG_EDGE_ANY
Start entweder durch steigende oder fallende Flanke.

...in Kombination mit ME_TRIG_TYPE_THRESHOLD in
<iAcqStartTrigType>:

- ME_TRIG_EDGE_ABOVE
Start sobald Pegel über der Triggerschwelle liegt.
- ME_TRIG_EDGE_BELOW
Start sobald Pegel unter der Triggerschwelle liegt.

...in Kombination mit ME_TRIG_TYPE_WINDOW in
<iAcqStartTrigType>:

- ME_TRIG_EDGE_ENTRY
Start sobald Pegel in definiertes Fenster eintritt.
- ME_TRIG_EDGE_EXIT
Start sobald Pegel definiertes Fenster verläßt.

<iAcqStartTrigChan>

Mit diesem Parameter können Sie wählen, ob die Triggerung für jeden Kanal getrennt erfolgen soll (Standard) oder ob ein Kanal synchron mit anderen Kanälen gestartet werden soll (z. B. für analoge Erfassung mit Sample & Hold-Option oder Synchron-Start mehrerer D/A-Kanäle).

- ME_TRIG_CHAN_DEFAULT
Unabhängiger Start (Standardeinstellung).
- ME_TRIG_CHAN_SYNCHRONOUS
Dieser Kanal wird in den Synchron-Start einbezogen
- MEphisto Scope:
 - Auswahl des analogen Triggerkanals (0, 1) in den Modi „Threshold“, „Window“, „Edge“ und „Slope“.
 - Übergabe des Referenz-Bitmusters im Triggermodus „Pattern“.
 - Ansonsten übergeben Sie hier 0.

<iAcqStartTicksLow>

Offset-Zeit in Anzahl der Ticks zwischen „Start“ der Messung und der ersten Wandlung. Beachten Sie, dass die Einschwingzeit des AI-Teils nicht unterschritten werden darf.

Sofern von der Hardware unterstützt, sind durch Kombination von <iAcqStartTicksLow> und <iAcqStartTicksHigh> bis zu 64 Bit breite Werte möglich. Es gilt:

$$AcqStartTicks = (AcqStartTicksHigh \ll 32) \vee AcqStartTicksLow.$$

Für Standardanwendungen empfehlen wir die Offset-Zeit mit der minimalen Chan-Zeit der betreffenden Hardware gleichzusetzen (<iAcqStartTicks> = min. <iConvStartTicks>).

<iAcqStartTicksHigh>

Höherwertiger Teil (Bits 63...32) der Offset-Zeit, siehe <iAcqStartTicksLow>. Wenn Sie den Parameter nicht nutzen wollen, übergeben Sie 0.

<iAcqStartArgs [0]> (r/w)

- Falls in <iAcqStartTrigType> ME_TRIG_TYPE_THRESHOLD übergeben wurde, definieren Sie hier den Schwellwert in [μV].
- Falls in <iAcqStartTrigType> ME_TRIG_TYPE_WINDOW übergeben wurde, definieren Sie hier den oberen Schwellwert des Fensters in [μV].
- Falls in <iAcqStartTrigType> ME_TRIG_TYPE_EDGE übergeben wurde, definieren Sie hier den Schwellwert in [μV].
- Falls in <iAcqStartTrigType> ME_TRIG_TYPE_SLOPE übergeben wurde, definieren Sie hier die Flankensteilheit in [μV/Sample].

(r): In obigen Fällen gibt dieser Parameter den aktuell verwendeten Triggerwert bei Rückkehr der Funktion *meIOStreamConfig()* zurück. Für andere Werte von <iAcqStartTrigType> wird dieser Parameter nicht genutzt und sollte 0 sein.

<iAcqStartArgs [1]> (r/w)

- Falls in <iAcqStartTrigType> ME_TRIG_TYPE_WINDOW übergeben wurde, definieren Sie hier den unteren Schwellwert des Fensters in [μV].

(r): In obigen Fällen gibt dieser Parameter den aktuell verwendeten Triggerwert bei Rückkehr der Funktion *meIOStreamConfig()* zurück. Für andere Werte von <iAcqStartTrigType> wird dieser Parameter nicht genutzt und sollte 0 sein.

<iScanStartTrigType>

Dieser Parameter definiert den Triggertyp für den Start eines Scans. In Abhängigkeit von der verwendeten Hardware können Sie aus folgenden Optionen wählen:

- ME_TRIG_TYPE_TIMER
Start des Scans durch Scan-Timer (z. B. Kanallistenabarbeitung).
- ME_TRIG_TYPE_FOLLOW
Der Start wird automatisch durch die Wandlung des letzten Kanallisteneintrags ausgelöst. Der Scan-Timer wird deaktiviert.
- ME_TRIG_TYPE_EXT_DIGITAL
Start des Scans durch geeignetes Signal am externen, digitalen Triggeringang.
- ME_TRIG_TYPE_EXT_ANALOG
Start des Scans durch geeignetes Signal am externen, analogen Triggeringang.

<iScanStartTicksLow>

Zeitintervall in Ticks zwischen dem Start zwei aufeinander folgender Scans (= Kanallistenabarbeitungen). Die Verwendung ist optional. Wenn Sie den Scan-Timer nicht nutzen möchten, übergeben Sie 0.

Beachten Sie bei der Berechnung des Scan-Intervalls folgenden Zusammenhang (siehe auch Abbildungen ab Seite 64):

$$\text{ScanStartTicks} = (\text{Anzahl der Kanallisteneinträge} \times \text{ConvStartTicks}) + \text{„Pause“ [Ticks]}$$

Sofern von der Hardware unterstützt, sind durch Kombination von <iScanStartTicksLow> und <iScanStartTicksHigh> bis zu 64 bit breite Werte möglich. Es gilt:

$$\text{ScanStartTicks} = (\text{ScanStartTicksHigh} \ll 32) \vee \text{ScanStartTicksLow}.$$

<iScanStartTicksHigh>

Höherwertiger Teil (Bits 63...32) der SCAN-Zeit, siehe <iScanStartTicksLow>. Wenn Sie den Parameter nicht nutzen wollen, übergeben Sie 0.

<iScanStartArgs [10]>

Reserviert für zukünftige Erweiterungen. Übergeben Sie 0.

<iConvStartTrigType>

Dieser Parameter definiert den Triggertyp für den Start einer einzelnen Wandlung. In Abhängigkeit von der verwendeten Hardware können Sie aus folgenden Optionen wählen:

- ME_TRIG_TYPE_TIMER
Start der Wandlung durch Chan-Timer.
- ME_TRIG_TYPE_EXT_DIGITAL
Start der Wandlung durch geeignetes Signal am externen, digitalen Triggereingang.
- ME_TRIG_TYPE_EXT_ANALOG
Start der Wandlung durch geeignetes Signal am externen, analogen Triggereingang.

<iConvStartTicksLow>

Chan-Intervall in Anzahl der Ticks zwischen zwei Wandlungen (Abtast- bzw. Ausgaberate). Der Wertebereich für die ME-4600 liegt zwischen 66 (42Hex) und $2^{32}-1$ (FFFFFFFFHex) Ticks.

Sofern von der Hardware unterstützt, sind durch Kombination von <iConvStartTicksLow> und <iConvStartTicksHigh> bis zu 64 bit breite Werte möglich. Es gilt:

$$\text{ConvStartTicks} = (\text{ConvStartTicksHigh} \ll 32) \vee \text{ConvStartTicksLow}.$$

<iConvStartTicksHigh>

Höherwertiger Teil (Bits 63...32) des Chan-Intervalls, siehe <iConvStartTicksLow>. Wenn Sie den Parameter nicht nutzen wollen, übergeben Sie 0.

<iConvStartArgs [10] >

Reserviert für zukünftige Erweiterungen. Übergeben Sie 0.

<iScanStopTrigType>

Dieser Parameter definiert den Triggertyp für das Ende eines Scans. In Abhängigkeit von der verwendeten Hardware können Sie aus folgenden Optionen wählen:

- ME_TRIG_TYPE_NONE
Keine Triggerquelle angegeben.
 - ME_TRIG_TYPE_COUNT
Erfassung/Ausgabe wird nach der unter <iScanStopCount> definierten Gesamtzahl an Wandlungen beendet.
- ! Verwenden Sie ME_TRIG_TYPE_COUNT nur alternativ entweder in <iScanStopTrigType> oder <iAcqStopTrigType>.

<iScanStopCount>

Gesamtzahl an Wandlungen nach denen die Scans beendet werden und damit auch die Operation als Ganzes. Wenn Sie die Operation auf unbestimmte Zeit laufen lassen wollen, übergeben Sie hier 0.

! Verwenden Sie diesen Parameter nur alternativ entweder in <iScanStopCount> oder <iAcqStopCount>.

<iScanStopArgs [0] >

- MEphisto Scope im Oszilloskop-Modus: Triggerpunkt in Prozent zwischen 0%...100%. Bei Rückkehr der Funktion *meIOStreamConfig()* wird hier der aktuelle Triggerpunkt zurückgegeben. Falls das MEphisto Scope im Datenlogger-Modus verwendet wird, wird dieser Parameter nicht benötigt und sollte 0 sein.

<iAcqStopTrigType>

Mit diesem Parameter können Sie bei Bedarf den Triggertyp für das Ende der gesamten Operation definieren. Folgende Optionen stehen zur Verfügung:

- ME_TRIG_TYPE_NONE
Keine Triggerquelle angegeben.
 - ME_TRIG_TYPE_COUNT
Die Operation wird nach der unter <iAcqStopCount> definierten Anzahl an „Scans“ (Kanallisten-Abarbeitungen) beendet.
- ! Verwenden Sie ME_TRIG_TYPE_COUNT nur alternativ entweder in <iScanStopTrigType> oder <iAcqStopTrigType>.
- ME_TRIG_TYPE_FOLLOW
Die Messung wird beendet, sobald die in <iScanStopCount> vorgegebene Anzahl an Wandlungen erreicht wurde.

<iAcqStopCount>

Anzahl an „Scans“ (Kanallisten-Abarbeitungen) nach denen die gesamte Operation beendet werden soll. Wenn Sie die Operation auf unbestimmte Zeit laufen lassen wollen, übergeben Sie 0.

! Verwenden Sie diesen Parameter nur alternativ entweder in **<iScanStopCount>** oder **<iAcqStopCount>**.

<iAcqStopArgs[10]>

Reserviert für zukünftige Erweiterungen. Übergeben Sie 0.

<iFlags>

- ME_IO_STREAM_TRIGGER_TYPE_NO_FLAGS (Standard)
(derzeit keine Flags definiert)

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_CHANNEL: Kein Kanal auf Subdevice verfügbar.
- ME_ERRNO_LOCKED: Subdevice ist gesperrt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt.
- ME_ERRNO_INVALID_REF: Parameter **<iRef>** ist nicht korrekt.
- ME_ERRNO_INVALID_ACQ_START_TRIG_CHAN: Parameter **<iAcqStartTrigChan>** ist nicht korrekt.
- ME_ERRNO_INVALID_ACQ_START_TRIG_EDGE: Parameter **<iAcqStartTrigEdge>** ist nicht korrekt.
- ME_ERRNO_INVALID_STREAM_CONFIG: Parameter **<iStreamConfig>** ist nicht korrekt.
- ME_ERRNO_TIMEOUT: Time-Out-Bedingung ist eingetreten.
- ME_ERRNO_PREVIOUS_CONFIG: Subdevice wurde nicht für die gewünschte Operation konfiguriert.
- ME_ERRNO_SUBDEVICE_BUSY: Subdevice führt andere Operation aus.

- ME_ERRNO_INVALID_FIFO_IRQ_THRESHOLD: Parameter <iFifoIrqThreshold> ist nicht gültig (zu groß).
- ME_ERRNO_INVALID_CONFIG_LIST_COUNT: Falsche Anzahl <iCount> in Konfigurationsliste.
- ME_ERRNO_INVALID_ACQ_START_TRIG_TYPE: Parameter <iAcqStartTrigType> ist nicht korrekt.
- ME_ERRNO_INVALID_ACQ_START_ARG: Zeitintervall <iAcqStartTicks> ist nicht korrekt.
- ME_ERRNO_INVALID_SCAN_START_ARG: Zeitintervall <iScanStartTicks> ist nicht korrekt.
- ME_ERRNO_INVALID_CONV_START_ARG: Zeitintervall <iConvStartTicks> ist nicht korrekt.
- ME_ERRNO_INVALID_ACQ_STOP_TRIG_TYPE: Parameter <iConvStartTrigType> ist nicht korrekt.
- ME_ERRNO_INVALID_SCAN_STOP_TRIG_TYPE: Parameter <iConvStartTrigType> ist nicht korrekt.
- ME_ERRNO_INVALID_ACQ_STOP_ARG: Parameter <iAcqStopCount> ist nicht korrekt.
- ME_ERRNO_INVALID_SCAN_STOP_ARG: Parameter <iScan-StopCount> ist nicht korrekt

meIOStreamTimeToTicks

Beschreibung

Konvertiert die gewünschte Periodendauer [s] in die Anzahl der „Ticks“ zur Übergabe an die Timer in der Funktion *meIOStreamConfig()*.

Hinweis

Die Umrechnung und der erlaubte Wertebereich sind abhängig vom jeweiligen Subdevice und dessen Timer. Falls Hardwaregrenzen überschritten werden, werden stets die Grenzwerte zurückgegeben.

Tip: Durch Übergabe von "0" im Parameter <pdTime> dieser Funktion wird die minimal erlaubte Periodendauer zurückgegeben. In Abhängigkeit des Parameters <iTimer> dieser Funktion können die Rückgabewerte von <piTicksLow> und <piTicksHigh> an die entsprechenden Parameter <iAcqStartTicks...>, <iConvStartTicks...> und <iScanStartTicks...> der Triggerstruktur von Funktion *meIOStreamConfig()* übergeben werden.

● Funktions-Deklaration

```
int meIOStreamTimeToTicks(int iDevice, int iSubdevice, int iTimer,
    double *pdTime, int *piTicksLow, int *piTicksHigh, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iTimer>

Ticks werden in Abhängigkeit vom Subdevice und des hier spezifizierten Timers berechnet. Die Werte <piTicksLow> und <piTicksHigh> werden in der Triggerstruktur der Funktion *meIOStreamConfig()* übergeben (siehe Seite 64ff).

- ME_TIMER_ACQ_START
<iAcqStartTicks> zur Übergabe an den gleichnamigen Parameter sollen berechnet werden.
- ME_TIMER_SCAN_START
<iScanStartTicks> zur Übergabe an den gleichnamigen Parameter sollen berechnet werden.
- ME_TIMER_CONV_START
<iConvStartTicks> zur Übergabe an den gleichnamigen Parameter sollen berechnet werden.

<pdTime> (r/w)

(w): Zeiger auf einen Double-Wert, der die gewünschte Periodendauer [s] zur Umrechnung in Ticks übergibt. Bei Übergabe von 0 wird die minimal mögliche Periodendauer zurückgegeben.

(r): Falls die gewünschte Periodendauer durch die Hardware nicht exakt umgesetzt werden kann, wird als Annäherung die nächstniedrigere Periodendauer zurückgegeben. Die korrespondierenden Ticks werden in den Parametern <piTicksLow> und <piTicksHigh> zurückgegeben.

<piTicksLow> (r)

Zeiger auf einen Integerwert, der die niederwertigen 32 Bits (31...0) der errechneten Ticks enthält. Wird an die entsprechenden Parameter <...StartTicksLow> der Funktion *meIOStreamConfig()* übergeben.

<piTicksHigh> (r)

Zeiger auf einen Integerwert, der die höherwertigen 32 Bits (63...32) der errechneten Ticks enthält. Wird an die entsprechenden Parameter <...StartTicksHigh> der Funktion *meIOStreamConfig()* übergeben.

<iFlags>

- ME_IO_TIME_TO_TICKS_NO_FLAGS (Standardeinstellung)
MEphisto Scope: Datenlogger-Modus
- ME_IO_TIME_TO_TICKS_MEPHISTO_SCOPE_OSCILLOSCOPE
MEphisto Scope: Oszilloskop-Modus

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebene Zeiger sind NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_TIMER: Timer nicht unterstützt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt

meIOStreamFrequencyToTicks

Beschreibung

Konvertiert die gewünschte Frequenz [Hz] in die Anzahl der „Ticks“ zur Übergabe an die Timer in der Funktion *meIOStreamConfig()*.

Hinweis

Die Umrechnung und der erlaubte Wertebereich sind abhängig von der jeweiligen Funktionsgruppe und deren Timer. Falls Hardwaregrenzen überschritten werden, werden stets die Grenzwerte zurückgegeben.

Tip: Durch Übergabe von "0" im Parameter *<pdFrequency>* dieser Funktion wird die maximal erlaubte Frequenz zurückgegeben. In Abhängigkeit des Parameters *<iTimer>* dieser Funktion können die Rückgabewerte von *<piTicksLow>* und *<piTicksHigh>* an die entsprechenden Parameter *<iAcqStartTicks...>*, *<iConvStartTicks...>* und *<iScanStartTicks...>* der Triggerstruktur von Funktion *meIOStreamConfig()* übergeben werden.

● Funktions-Deklaration

```
int meIOStreamFrequencyToTicks(int iDevice, int iSubdevice, int iTimer,
                               double *pdFrequency, int *piTicksLow, int *piTicksHigh, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iTimer>

Ticks werden in Abhängigkeit vom Subdevice des hier spezifizierten Timers berechnet. Die Werte *<TicksLow>* und *<TicksHigh>* werden in der Triggerstruktur der Funktion *meIOStreamConfig()* übergeben (siehe Seite 64ff).

- ME_TIMER_ACQ_START
<iAcqStartTicks> zur Übergabe an den gleichnamigen Parameter sollen berechnet werden.
- ME_TIMER_SCAN_START
<iScanStartTicks> zur Übergabe an den gleichnamigen Parameter sollen berechnet werden.
- ME_TIMER_CONV_START
<iConvStartTicks> zur Übergabe an den gleichnamigen Parameter sollen berechnet werden.

<pdFrequency> (r/w)

Zeiger auf einen Double-Wert, der die gewünschte Frequenz [Hz] zur Umrechnung in Ticks übergibt. Bei Übergabe von 0 wird die maximal mögliche Frequenz zurückgegeben.

Falls die gewünschte Frequenz durch die Hardware nicht exakt umgesetzt werden kann, wird als Annäherung die nächsthöhere Frequenz zurückgegeben. Die korrespondierenden Ticks werden in den Parametern <piTicksLow> und <piTicksHigh> zurückgegeben.

<piTicksLow> (r)

Zeiger auf einen Integerwert, der die niederwertigen 32 Bits (31...0) der errechneten Ticks enthält. Wird an die entsprechenden Parameter <...StartTicksLow> der Funktion *meIOStreamConfig()* übergeben.

<piTicksHigh> (r)

Zeiger auf einen Integerwert, der die höherwertigen 32 Bits (63...32) der errechneten Ticks enthält. Wird an die entsprechenden Parameter <...StartTicksHigh> der Funktion *meIOStreamConfig()* übergeben.

<iFlags>

- ME_IO_FREQUENCY_TO_TICKS_NO_FLAGS (Standardeinstellung)
MEphisto Scope: Datenlogger-Modus
- ME_IO_FREQUENCY_TO_TICKS_MEPHISTO_SCOPE_OSCILLOSCOPE
MEphisto Scope: Oszilloskop-Modus

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebene Zeiger sind NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_TIMER: Timer nicht unterstützt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt

meIOStreamStart

Beschreibung

Funktion startet Streaming-Operation. Entweder sofort (Software-Start) oder gemäß den in der Funktion *meIOStreamConfig()* definierten Start-Bedingungen.

Beendet wird eine Streaming-Operation entweder gemäß den in der Funktion *meIOStreamConfig()* definierten Stop-Bedingungen oder mit der Funktion *meIOStreamStop()*.

Sofern eine Streaming-Operation nicht mit der Funktion *meIOResetDevice()* beendet wurde (Hardware-Konfiguration wird gelöscht), kann durch Aufruf dieser Funktion eine neue Operation gestartet werden, ohne dass vorher neu konfiguriert werden muss.

Hinweis!

Das Rückkehr-Verhalten der Funktion hängt vom `<iStartmode>` (BLOCKING oder NONBLOCKING) und den in der Funktion *meIOStreamConfig()* definierten Triggerbedingungen ab.

● Funktions-Deklaration

```
int meIOStreamStart(meIOStreamStart_t *pStartList, int iCount, int
                    iFlags);
```

<pStartList>

Zeiger auf eine Liste vom Typ `meIOStreamStart_t`, mit der ein oder mehrere Streaming-Operationen gestartet werden können. Der Start erfolgt unmittelbar nach Funktionsaufruf gemäß den Start-Bedingungen. Sofern für einen oder mehrere Listeneinträge der `<iStartMode>` `ME_START_MODE_BLOCKING` und eine externe Triggerquelle (siehe *meIOStreamConfig()*) gewählt wurde, wartet („blockiert“) die Funktion bis das Triggersignal eintrifft.

<iCount>

Anzahl der Einträge in `<pStartList>`.

<iFlags>

- `ME_IO_STREAM_START_NO_FLAGS`
Standard-Einstellungen. Liste wird abgearbeitet solange kein Fehler auftritt. Im Fehlerfall korrespondiert der zurückgegebene Wert mit dem ersten Eintrag im Array `<iErrno>`, welcher nicht 0 ist.
- `ME_IO_STREAM_START_NONBLOCKING` (nur Linux)
Führe die gesamte Startliste aus. Sobald dieses Flag gesetzt ist, wird die Liste abgearbeitet auch wenn für ein oder mehrere Einträge ein Fehler auftrat. Die Funktion gibt `ME_ERRNO_SUCCESS` zurück, sofern

kein globaler Fehler entdeckt wurde. Das Array <iErrno> kann anschließend überprüft werden.

→ Typ-Definition meIOStreamStart

```
typedef struct meIOStreamStart {
    int iDevice;
    int iSubdevice;
    int iStartMode;
    int iTimeout;
    int iFlags;
    int iErrno;
} meIOStreamStart_t;
```

<iDevice> (w)

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice> (w)

Index des anzusprechenden Subdevices.

<iStartMode> (w)

- ME_START_MODE_BLOCKING
Bei Verwendung eines externen Triggers, wartet (blockiert) die Funktion bis das entsprechende Triggersignal eingetroffen ist.
- ME_START_MODE_NONBLOCKING
Die Funktion kehrt sofort zurück. Der Start der Hardware erfolgt im Hintergrund. <pStartList> wird unabhängig vom Eintreffen externer Triggersignale abgearbeitet. Die Funktion gibt ME_ERRNO_SUCCESS zurück sofern kein globaler Fehler aufgetreten ist.

<iTimeout> (w)

Optional können Sie hier ein Zeitintervall in Millisekunden angeben, innerhalb dessen der erste Triggerimpuls gemäß den in der Funktion *meIOStreamConfig()* definierten Bedingungen eintreffen muss. Ansonsten wird die Messung abgebrochen. Falls Sie ohne ext. Trigger arbeiten oder kein Time-Out nutzen möchten, übergeben Sie hier 0.

<iFlags> (w)

- ME_IO_STREAM_START_TYPE_NO_FLAGS
Gilt, sofern keine andere Konstante gewählt wurde.
- ME_IO_STREAM_START_TYPE_TRIG_SYNCHRONOUS
Synchronstart per „Synchronstart-Liste“.

<iErrno>

Im Fehlerfall wird Fehlercode zurückgegeben.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_FLAGS: Nicht unterstütztes Flag verwendet.
- ME_ERRNO_INVALID_TIMEOUT: Time-Out kleiner als 0.
- ME_ERRNO_INVALID_START_MODE: Start-Modus nicht unterstützt.
- ME_ERRNO_PREVIOUS_CONFIG: Subdevice ist nicht korrekt für Streaming-Operation konfiguriert.
- ME_STATUS_ERROR: Vorherige Operation wurde mit einem Fehler beendet. Um den Zustand zu zurückzusetzen ist ein Reset erforderlich.
- ME_ERRNO_LACK_OF_RESOURCES: (nur Ausgabe) Ausgabe kann nicht fortfahren. Puffer sind leer. (nur Linux)
- ME_ERRNO_TIMEOUT: Time-Out. Operation startete nicht rechtzeitig.
- ME_ERRNO_START_THREAD: Callback-Thread generieren nicht möglich (nur Windows)

meIOStreamStop

Beschreibung

Mit dieser Funktion können Sie eine „Endlos“-Operation wahlweise sofort beenden oder definiert „anhalten“ (siehe Parameter `<iStopMode>`). Damit haben Sie z. B. die Möglichkeit eine Ausgabe mit dem letzten Eintrag im FIFO und somit einem bekannten Wert zu stoppen.

Falls in den Parametern `<iAcqStopCount>` bzw. `<iScanStopCount>` der Funktion `meIOStreamConfig()` Stop-Bedingungen definiert wurden, ist ein Aufruf der Funktion `meIOStreamStop()` nicht notwendig.

Die Konfiguration des Subdevices bleibt erhalten (Kanalliste, Timer, etc.), sodass ein erneutes Starten mit der Funktion `meIOStreamStart()` ohne Neukonfiguration möglich ist.

Im Gegensatz dazu wird bei Verwendung der Funktion `meIOResetDevice()` die gesamte Konfiguration des Geräts gelöscht.

● Funktions-Deklaration

```
int meIOStreamStop(meIOStreamStop_t *pStopList, int iCount, int iFlags);
```

<pStopList>

Zeiger auf eine Liste vom Typ `meIOStreamStop_t`, mit der ein oder mehrere Ein-/Ausgabe-Operationen beendet werden können. Der Stop erfolgt gemäß Parameter `<iStopMode>`.

<iCount>

Anzahl der Einträge in `<pStopList>`.

<iFlags>

- **ME_IO_STREAM_STOP_NO_FLAGS**
Standard-Einstellungen. Liste wird abgearbeitet solange kein Fehler auftritt. Im Fehlerfall korrespondiert der zurückgegebene Wert mit dem ersten Eintrag im Array `<iErrno>`, welcher nicht 0 ist.
- **ME_IO_STREAM_STOP_NONBLOCKING** (nur Linux)
Führe die gesamte Stopliste aus. Sobald dieses Flag gesetzt ist, wird die Liste abgearbeitet auch wenn für ein oder mehrere Einträge ein Fehler auftrat. Die Funktion gibt `ME_ERRNO_SUCCESS` zurück, sofern kein globaler Fehler entdeckt wurde. Das Array `<iErrno>` kann anschließend überprüft werden.

→ Typ-Definition meIOStreamStop

```
typedef struct meIOStreamStop {  
    int iDevice;  
    int iSubdevice;  
    int iStopMode;  
    int iFlags;  
    int iErrno;  
} meIOStreamStop_t;
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iStopMode>

- ME_STOP_MODE_IMMEDIATE
Streaming-Operation wird sofort beendet. Im Falle einer analogen Ausgabe liegt anschließend 0V am Pin an.
- ME_STOP_MODE_LAST_VALUE
 - Ausgabe-Subdevice: Operation wird mit dem letzten Eintrag im D/A-FIFO gestoppt, welcher ein bekannter, definierter Wert ist.
 - Eingabe-Subdevice: Operation wird mit dem letzten Eintrag der Kanalliste gestoppt.

<iFlags>

- ME_IO_STREAM_STOP_TYPE_NO_FLAGS
Standardeinstellungen
 - Ausgabe-Subdevice: alle Puffer werden gelöscht.
 - Eingabe-Subdevice: Hardwarepuffer wird gelöscht. Keinerlei Synchronisation. Im Modus ME_STOP_MODE_IMMEDIATE können Daten verlorengehen.
- ME_IO_STREAM_STOP_TYPE_PRESERVE_BUFFERS (nur Linux)
 - Ausgabe-Subdevice: alle Puffer werden erhalten. "Streaming"-Betrieb kann fortgesetzt werden.
 - Eingabe-Subdevice: Synchronisation erfolgt bevor Hardwarepuffer gelöscht wird. Kein Datenverlust.

<iErrno>

Im Fehlerfall wird ein Fehlercode zurückgegeben.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_FLAGS: Nicht unterstütztes Flag verwendet.
- ME_ERRNO_INVALID_STOP_MODE: Nicht unterstützter Stop-Modus verwendet.
- ME_ERRNO_PREVIOUS_CONFIG: Subdevice ist nicht korrekt für Streaming-Operation konfiguriert.
- ME_STATUS_ERROR: Vorherige Operation wurde mit einem Fehler beendet. Um den Zustand zu zurückzusetzen ist ein Reset erforderlich.

meIOStreamRead

Beschreibung

Mit dieser Funktion können während einer Streaming-Operation (timergesteuerte Erfassung) Werte aus dem Datenpuffer gelesen werden.

Der Benutzer muss einen Datenpuffer zuweisen, in dem die Messwerte abgelegt werden. Im Ausführungsmodus „BLOCKING“ kehrt die Funktion *meIOStreamRead()* erst nach Lesen des letzten Wertes zurück. Im Modus „NONBLOCKING“ kehrt die Funktion sofort mit den verfügbaren Messwerten zurück.

Für das Einlesen der Daten können auch Callback-Funktionen verwendet werden (zur Installation der Callback-Funktionen siehe Funktion *meIOStreamSetCallbacks()* auf Seite 194).

Hinweis:

Weitere Details finden Sie im Kapitel Streaming-Operation ab Seite 62.

● Funktions-Deklaration

```
int meIOStreamRead(int iDevice, int iSubdevice, int iReadMode, int
    *piValues, int *piCount, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iReadMode>

- ME_READ_MODE_BLOCKING
Die Funktion wartet, bis die im Parameter <piCount> spezifizierte Anzahl der Messwerte erfasst wurde.
Beachte: Aufruf kann dauerhaft blockieren!
- ME_READ_MODE_NONBLOCKING
Die Funktion kehrt sofort zurück, entweder mit der Anzahl der Messwerte, die zum Zeitpunkt des Aufrufs vorhanden waren oder mit der in <piCount> spezifizierten Anzahl. Es gilt jeweils der kleinere Wert.

<piValues> (r)

Zeiger auf Datenpuffer (Datenstrom) der laufenden Erfassung. Verwenden Sie die Funktion *meUtilityDigitalToPhysical()* zur einfachen Umrechnung in Spannungswerte.

<piCount> (r/w)

(w) : Größe des Datenpuffers in Anzahl der Messwerte, die eingelesen werden sollen.

(r) : Der Zeiger gibt die Anzahl der tatsächlich aus dem Datenpuffer gelesenen Werte zurück. Falls im BLOCKING-Modus die Erfassung abgebrochen wurde, kann der Rückgabewert auch kleiner als der Übergabewert sein.

- BLOCKING-Modus:

Anzahl der einzulesenden Werte - in der Regel ein Vielfaches der Kanallistenlänge, dies ist aber nicht zwingend (siehe auch *meUtilityExtractValues()*, Seite 214).

- NONBLOCKING-Modus:

Anzahl der einzulesenden Werte - falls Sie ein Vielfaches der Kanallistenlänge einlesen möchten, verwenden Sie im Parameter **<iFlags>** die Konstante ME_IO_STREAM_READ_FRAMES.

<iFlags>

- ME_IO_STREAM_READ_NO_FLAGS

Standardeinstellungen.

- ME_IO_STREAM_READ_FRAMES

Verwenden Sie dieses Flag, falls Sie im NONBLOCKING-Modus ein Vielfaches der Kanallistenlänge einlesen möchten.

Beispiel: Die Kanalliste hat 5 Einträge und im Parameter **<piCount>** wird der Wert „14“ übergeben:

- ME_IO_STREAM_READ_NO_FLAGS: 14 Werte werden zurückgegeben.
- ME_IO_STREAM_READ_FRAMES: 10 Werte werden zurückgegeben.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebene Zeiger sind NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_FLAGS: Nicht unterstütztes Flag verwendet.
- ME_ERRNO_INVALID_VALUE_COUNT: **<piCount>** ist kleiner als Null.

- ME_ERRNO_INVALID_READ_MODE: Nicht unterstützter Lese-Modus verwendet.
- ME_ERRNO_SUBDEVICE_NOT_RUNNING: Puffer ist leer und Subdevice befindet sich nicht im Streaming-Betrieb.
- ME_ERRNO_HARDWARE_FIFO_OVERFLOW: Fehler während der Erfassung. Das Lesen der Daten vom FIFO war zu langsam.
- ME_ERRNO_RING_BUFFER_OVERFLOW: Kein Platz im Speicher für neue Daten. Das Lesen der Daten vom FIFO war zu langsam.

meIOStreamWrite

Beschreibung

Diese Funktion dient dem Laden des Datenpuffers für timergesteuerte, analoge bzw. digitale Ausgabe in der Betriebsart Streaming. Weisen sie für jedes Subdevice, das verwendet werden soll, einen Datenpuffer definierter Größe zu für die auszugebenden Werten.

In Abhängigkeit von Parameter `<iFlags>` der Funktion `meIOStreamConfig()` können Sie zwischen folgenden Optionen wählen:

- a. Mit der Konstante `ME_IO_STREAM_CONFIG_NO_FLAGS` können Sie beliebige analoge Signale kontinuierlich ausgeben. Der Datenpuffer muss regelmäßig mit neuen Werten nachgeladen werden, die sich nach Beginn der Ausgabe auch ändern können. Verwenden Sie beim erstmaligen Laden die Option `ME_WRITE_MODE_PRELOAD` im Parameter `<iWriteMode>`.
- b. Die Konstante `ME_IO_STREAM_CONFIG_BIT_PATTERN` dient der Bitmuster-Ausgabe und der Zeitsteuerung des ME-MultiSig-Systems, falls es nicht im ME-iDC angemeldet werden soll (in Vorbereitung). Mit dieser Konstante verknüpfen Sie die Zeitsteuerung mit den Digital-Ports der ME-4680. Siehe auch Parameter `<iSingleConfig>` und `<iRef>` der Funktion `meIOSingleConfig()`.
- c. Mit der Konstante `ME_IO_STREAM_CONFIG_WRAPAROUND` können Sie sowohl analoge Signale als auch digitale Bitmuster *periodisch* ausgeben. Der Datenpuffer muss einmalig mit den auszugebenden Werten beladen werden. Verwenden Sie dazu die Option `ME_WRITE_MODE_PRELOAD` im Parameter `<iWriteMode>`. Sofern die Anzahl der Werte im Datenpuffer die FIFO-Größe (hardwareabhängig) nicht übersteigt, läuft die Ausgabe auf Firmware-Ebene, d. h. der Host-Rechner wird nicht belastet! (ME-4680: 4096 Werte, ME-6100/6300: 8192 Werte). Zur periodischen Bitmuster-Ausgabe und im periodischen DEMUX-Betrieb muss diese Konstante mit der Konstante `ME_IO_STREAM_CONFIG_BIT_PATTERN` „ODER“-verknüpft werden

Unter Linux siehe auch `ME_IO_STREAM_CONFIG_HARDWARE_ONLY` im Parameter `<iFlags>` der Funktion `meIOStreamConfig()`.

Hinweise

- > Beachten Sie, dass D/A-Kanäle mit Daten-FIFO, die für timergesteuerte Ausgabe geeignet sind (`ME_SUBTYPE_STREAMING`) als eigenständige Subdevices angesprochen werden.

Siehe auch Kapitel Streaming-Betrieb ab Seite 62.

● Funktions-Deklaration

```
int meIOStreamWrite(int iDevice, int iSubdevice, int iWriteMode, int
    *piValues, int *piCount, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iWriteMode>

- ME_WRITE_MODE_BLOCKING

Die Funktion wartet bis die in <piCount> spezifizierte Anzahl an Werten in den Datenpuffer geschrieben werden konnte.

Beachte: Aufruf kann dauerhaft blockieren!

- ME_WRITE_MODE_NONBLOCKING

Mit dieser Option schreibt die Funktion so viele Werte in den Datenpuffer, wie zum Zeitpunkt des Funktionsaufrufs Platz haben (max. die in <piCount> spezifizierte Anzahl).

- ME_WRITE_MODE_PRELOAD

Erstmaliges Vorladen des Datenpuffers. Daten werden direkt in den Hardwarepuffer geschrieben. Falls es mehr Daten sind, als in den FIFO-Speicher passen, wird der Rest im internen Puffer gespeichert.

<piValues> (w)

Zeiger auf Datenpuffer (Datenstrom) der auszugebenden Werte bzw. Bitmuster. Verwenden Sie die Funktion *meUtilityPhysicalToDigital()* zur einfachen Umrechnung von physikalischen Werten (z. B. Spannung) in Digital-Werte.

<piCount> (r/w)

(w): Anzahl der Werte, die in den Datenpuffer geladen werden sollen.

(r): Zeiger gibt die Anzahl der Werte zurück, die tatsächlich in den Datenpuffer geschrieben werden konnten. Falls nicht so viele neuen Werte Platz hatten und im Parameter <iWriteMode> die Konstante ME_WRITE_MODE_NONBLOCKING übergeben wurde, kann der Rückgabewert auch kleiner als der Übergabewert sein.

<iFlags>

- ME_IO_STREAM_WRITE_NO_FLAGS

Standardeinstellungen - keine Flags

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebene Zeiger sind NULL.

- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_FLAGS: Nicht unterstütztes Flag verwendet.
- ME_ERRNO_INVALID_VALUE_COUNT: <piCount> ist kleiner als Null.

Hinweis: Wenn <piCount> auf Null gesetzt wird, wird ME_ERRNO_SUCCESS zurückgegeben.

- ME_ERRNO_INVALID_WRITE_MODE: Nicht unterstützter Schreib-Modus verwendet.
- ME_ERRNO_PREVIOUS_CONFIG: Device ist für den Single-Betrieb konfiguriert.
- ME_ERRNO_HARDWARE_FIFO_UNDERFLOW:
 - Allgemein: Fehler während Streaming-Betrieb. Das Schreiben der Daten ins FIFO war zu langsam.
 - Hardware-Wraparound: Mehr Daten als Platz im FIFO.
- ME_ERRNO_RING_BUFFER_UNDERFLOW: Datenpuffer ist leer. Das Schreiben der Daten in den Puffer war zu langsam.
- ME_ERRNO_SUBDEVICE_NOT_RUNNING: Die interne Ablaufsteuerung der Hardware wurde gestoppt aber nach logischem Status sollte sie arbeiten. Keine Daten im FIFO aber der Software-Puffer ist nicht leer.

meIOStreamStatus

Beschreibung

Status der Streaming-Operation abfragen. Je nach Art der Operation (Ein- oder Ausgabe) dient diese Funktion der Abfrage ob bereits alle Messwerte erfasst wurden oder ob eine Ausgabe-Operation noch läuft.

Im Parameter `<iWait>` können Sie steuern, ob die Funktion den aktuellen Status sofort zurückgeben soll, oder ob Sie warten möchten bis die Erfassung bzw. Ausgabe beendet ist.

● Funktions-Deklaration

```
int meIOStreamStatus(int iDevice, int iSubdevice, int iWait, int *piStatus,
                    int *piCount, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iWait>

Rückkehr-Verhalten dieser Funktion:

- ME_WAIT_NONE
Aktuellen Status abfragen. Funktion gibt den aktuellen Betriebszustand im Parameter `<piStatus>` sofort zurück.
- ME_WAIT_IDLE
Im Falle einer Ausgabe-Operation kehrt die Funktion erst zurück, nachdem alle Werte ausgegeben wurden. In diesem Fall gibt der Parameter `<piStatus>` stets den Wert ME_STATUS_IDLE zurück.
Beachte: Aufruf kann dauerhaft blockieren!
- ME_WAIT_BUSY (nur Linux)
Die Funktion blockiert solange der Status ME_STATUS_BUSY ist
Beachte: Aufruf kann dauerhaft blockieren!

<piStatus> (r)

Zeiger, der den aktuellen Betriebszustand des spezifizierten Subdevices zurückgibt:

- ME_STATUS_IDLE
Streaming-Operation ist beendet.
- ME_STATUS_BUSY
Streaming-Operation läuft noch.
- ME_STATUS_ERROR
Fehler aufgetreten, z. B. Datenstrom wurde unterbrochen.

<piCount> (r)

- Eingabe-Subdevice: Anzahl der Werte, die eingelesen werden können
- Ausgabe-Subdevice: Freier Speicher im Ausgabepuffer (in Anzahl der Werte)

<iFlags>

- ME_IO_STREAM_STATUS_NO_FLAGS
Keine Flags – Standardeinstellung.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebene Zeiger sind NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_FLAGS: Nicht unterstütztes Flag verwendet.
- ME_ERRNO_INVALID_WAIT: Nicht unterstütztes Rückkehr-Verhalten verwendet.

meIOStreamNewValues

Beschreibung

Status einer Streaming-Operation abfragen. Die Funktion kehrt zurück wenn:

- Eingabe-Subdevice: Einige Werte im Puffer.
- Ausgabe-Subdevice: Freier Speicher im Puffer.

Mit dem Parameter `<iTimeOut>` können Sie verhindern, dass die Funktion dauerhaft blockiert.

● Funktions-Deklaration

```
int meIOStreamNewValues(int iDevice, int iSubdevice, int iTimeOut, int
    *piCount, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iTimeOut>

Time-Out in Millisekunden. Die Funktion kehrt innerhalb einer definierten Zeit zurück, falls der Pufferstatus nicht ermittelt werden kann. Falls Sie kein Time-Out nutzen möchten, übergeben Sie hier 0.

<piCount> (r)

- Eingabe-Subdevice: Freier Speicher im Puffer (Anzahl Werte).
- Ausgabe-Subdevice: Freier Speicher im Puffer (Anzahl Werte).

<iFlags>

- ME_IO_STREAM_NEW_VALUES_NO_FLAGS
Keine Flags – Standardeinstellung

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_FLAGS: Nicht unterstütztes Flag.
- ME_ERRNO_TIMEOUT: Time-Out.

meIOStreamSetCallbacks

Beschreibung

Mit dieser Funktion können Sie verschiedene Callback-Funktionen installieren, die im Hintergrund auf ein Event eines Streaming-Subdevices warten. Die Funktionen können in Abhängigkeit vom Datenstrom aufgerufen werden:

- `<pStartCB>` – wird beim Start der Operation ausgeführt
- `<pNewValuesCB>` – wird ausgeführt, wenn Daten eingelesen (Eingabe-Subdevice) oder geschrieben (Ausgabe-Subdevice) werden können.
- `<pEndCB>` – wird am Ende der Operation ausgeführt

Hinweis

Zum Deinstallieren aller registrierten Callback-Instanzen eines bestimmten Subdevices rufen Sie die Funktion *meIOStreamSetCallbacks()* auf und setzen Sie alle Callback-Zeiger (`<pStartCB>`, `<pNewValuesCB>` und `<pEndCB>`) auf NULL. Dies ist gleichbedeutend mit dem Aufruf der Funktion *meIOStreamStop()*.

● Funktions-Deklaration

```
int meIOStreamSetCallbacks(int iDevice, int iSubdevice, meIOStreamCB_t
    pStartCB, void *pStartCBContext, meIOStreamCB_t
    pNewValuesCB, void *pNewValuesCBContext, meIOStreamCB_t
    pEndCB, void *pEndCBContext, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<pStartCB>

Zeiger auf eine anwenderdefinierte Funktion, die aufgerufen wird, sobald die Streaming-Operation startet. Falls die Funktion mit einem anderen Rückgabewert als `ME_ERNNO_SUCCESS (0x00)` zurückkehrt wird die Streaming-Operation sofort gestoppt (*meIOStreamStop()* wird ausgeführt).

<pStartCBContext>

Anwenderdefinierter Zeiger, der der „Start“-Callback-Funktion übergeben wird. Wenn Sie die Funktionalität nicht verwenden möchten, übergeben Sie NULL.

<pNewValuesCB>

Zeiger auf eine anwenderdefinierte Funktion, die aufgerufen wird, sobald sich der Pufferstatus ändert. Falls die Funktion mit einem anderen Rückgabewert als ME_ERNNO_SUCCESS (0x00) zurückkehrt wird die Streaming-Operation sofort gestoppt (*meIOStreamStop()* wird ausgeführt).

<pNewValuesCBContext>

Anwenderdefinierter Zeiger, der der „Neue Werte“-Callback-Funktion übergeben wird. Wenn Sie die Funktionalität nicht verwenden möchten, übergeben Sie NULL.

<pEndCB>

Zeiger auf eine anwenderdefinierte Funktion, die aufgerufen wird, sobald die Streaming-Operation beendet wird.

<pEndCBContext>

Anwenderdefinierter Zeiger, der der „Stop“-Callback-Funktion übergeben wird. Wenn Sie die Funktionalität nicht verwenden möchten, übergeben Sie NULL.

<iFlags>

- ME_IO_STREAM_SET_CALLBACKS_NO_FLAGS
(z. Zt. keine Flags vorhanden)

→ Typ-Definition `meIOStreamCB_t`

```
typedef int (*meIOStreamCB_t) (
    int iDevice,
    int iSubdevice,
    int iCount,
    void *pvContext,
    int iErrorCode);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iCount>

- Eingabe-Subdevice: Anzahl der Werte, die gelesen werden können
- Ausgabe-Subdevice: Freier Speicher im Ausgabe-Puffer.

<pvContext> (w)

Benutzerdefinierter Zeiger auf genau den Wert, der zuvor im Parameter **<p...CBContext>** dieser Funktion übergeben wurde. Falls Sie diesen Parameter nicht verwenden wollen übergeben Sie NULL.

<iErrorCode>

Im Fehlerfall wird ein Fehlercode zurückgegeben.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_POINTER: Übergebene Zeiger sind NULL.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Device zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_LOCKED: Subdevice ist gesperrt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt.
- ME_ERRNO_START_THREAD: Callback-Thread generieren nicht möglich. (nur Linux)
- -ENOMEM: Nicht genügend Speicher verfügbar. (nur Linux)

4.2.4 Hilfs-Funktionen

meOpen

Beschreibung

Diese Funktion initialisiert die Funktionsbibliothek:

- Speicher wird reserviert
- Interne Variablen werden gesetzt
- Verbindung zum Treiber einrichten
- Ermittelte Ressourcen den logischen Strukturen zuordnen.

Ansonsten ist kein Zugriff auf das ME-iDS möglich.

● Funktions-Deklaration

```
int meOpen(int iFlags);
```

<iFlags>

- ME_OPEN_NO_FLAGS

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_OPEN: ME-iDS kann nicht korrekt geöffnet werden. Vermutlich ist der Treiber nicht geladen.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt.

meClose

Beschreibung

Diese Funktion schließt die Verbindung zur Funktionsbibliothek:

- Reservierter Speicher wird freigegeben
- Verbindung zum Treiber wird aufgehoben

● Funktions-Deklaration

```
int meClose(int iFlags);
```

<iFlags>

- ME_CLOSE_NO_FLAGS

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_CLOSE: ME-iDS kann nicht korrekt geschlossen werden.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt

meLockDriver

Beschreibung

Das gesamte Treibersystem (ME-iDS) wird für den Zugriff anderer Prozesse gesperrt bzw. freigegeben. Falls eine andere Anwendung auf das Treibersystem zugreifen möchte führt dies zu einer Fehlermeldung.

● Funktions-Deklaration

```
int meLockDriver(int iLock, int iFlags);
```

<iLock>

- ME_LOCK_SET
Das Treibersystem wird für den Zugriff anderer Prozesse gesperrt.
- ME_LOCK_RELEASE
Das gesperrte Treibersystem wird freigegeben.
- ME_LOCK_CHECK
Aktuellen Status überprüfen.

<iFlags>

- ME_LOCK_DRIVER_NO_FLAGS

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_LOCKED: Einige Ressourcen sind von anderer Anwendung gesperrt.
- ME_ERRNO_USED: Einige Ressourcen sind gerade in Aktion und können daher nicht gesperrt werden.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt

meLockDevice

Beschreibung

Ein Gerät wird als Ganzes gesperrt bzw. freigegeben. Falls ein anderer Prozess auf ein gesperrtes Gerät zugreifen möchte führt dies zu einer Fehlermeldung.

● Funktions-Deklaration

```
int meLockDevice(int iDevice, int iLock, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iLock>

- ME_LOCK_SET
Das Gerät wird für den Zugriff anderer Prozesse gesperrt.
- ME_LOCK_RELEASE
Das gesperrte Gerät wird freigegeben.
- ME_LOCK_CHECK
Aktuellen Status überprüfen.

<iFlags>

- ME_LOCK_DEVICE_NO_FLAGS

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_LOCKED: Einige Ressourcen sind von anderer Anwendung gesperrt.
- ME_ERRNO_USED: Einige Ressourcen sind gerade in Aktion und können daher nicht gesperrt werden.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt

meLockSubdevice

Beschreibung

Ein Subdevice wird gesperrt bzw. freigegeben. Falls ein anderer Prozess auf ein gesperrtes Subdevice zugreifen möchte, führt dies zu einer Fehlermeldung.

● Funktions-Deklaration

```
int meLockSubdevice(int iDevice, int iSubdevice, int iLock, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice>

Index des anzusprechenden Subdevices.

<iLock>

- ME_LOCK_SET
Das Subdevice wird für den Zugriff anderer Prozesse gesperrt.
- ME_LOCK_RELEASE
Das gesperrte Subdevice wird freigegeben.
- ME_LOCK_CHECK
Aktuellen Status überprüfen.

<iFlags>

- ME_LOCK_SUBDEVICE_NO_FLAGS

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_LOCKED: Ressource ist von anderer Anwendung gesperrt.
- ME_ERRNO_USED: Ressource ist gerade in Aktion und kann daher nicht gesperrt werden.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt.

meErrorGetLast

Beschreibung

Diese Funktion gibt den letzten Fehlercode zurück.

● Funktions-Deklaration

```
int meErrorGetLast(int *piErrorCode, int iFlags);
```

<piErrorCode>

Zeiger auf Fehlercode.

<iFlags>

- ME_NO_FLAGS: Standardeinstellung.
- ME_ERRNO_CLEAR_FLAGS: Diesen Fehler nicht erneut melden.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_INVALID_FLAGS: Übergebene Flags werden nicht unterstützt.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.

meErrorGetLastMessage

Beschreibung

Diese Funktion gibt den letzten, von einer API-Funktion verursachten, Fehler zurück. Ein entsprechender Fehlertext kann angezeigt werden.

● Funktions-Deklaration

```
int meErrorGetLastMessage(char *pcErrorMsg, int iCount);
```

<pcErrorMsg>

Zeiger auf den Fehlerbeschreibungstext.

<iCount>

Puffergröße in Bytes für Fehlerbeschreibungstext. Verwenden Sie die Konstante ME_ERROR_MSG_MAX_COUNT.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_INVALID_ERROR_MSG_COUNT: Reservierter Puffer ist zu klein für Beschreibung.

Hinweis: Dieser Fehler kann in einigen Fällen ignoriert werden. Der gesamte verfügbare Puffer wird mit einem Beschreibungsstring gefüllt.

- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.

meErrorMessage

Beschreibung

Diese Funktion kann dazu verwendet werden um einen Fehlercode, der von einer API-Funktion zurückgegeben wurde in lesbaren Text umzuwandeln.

● Funktions-Deklaration

```
int meErrorMessage(int iErrorCode, char *pcErrorMsg, int iCount)
```

<iErrorCode>

Fehlercode, der API-Funktion.

<pcErrorMsg>

Zeiger auf Fehlerbeschreibungstext.

<iCount>

Puffergröße in Bytes für Fehlerbeschreibungstext. Verwenden Sie die Konstante ME_ERROR_MSG_MAX_COUNT.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_INVALID_ERROR_NUMBER: Angegebener Fehlercode ist im ME-iDS nicht gültig.
- ME_ERRNO_INVALID_ERROR_MSG_COUNT: Reservierter Puffer ist zu klein für Beschreibung.

Hinweis: Dieser Fehler kann in einigen Fällen ignoriert werden. Der gesamte verfügbare Puffer wird mit einem Beschreibungsstring gefüllt.

- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.

meErrorSetDefaultProc

Beschreibung

Diese Funktion dient dazu eine (vordefinierte) globale Standard-Fehlerroutine für das gesamte ME-iDS zu installieren. Die globale Fehlerroutine wird automatisch aufgerufen, sobald eine Funktion einen Fehler zurückgibt. Sie erhalten folgende Infos:

- Name der Funktion, die den Fehler verursacht hat
- Kurze Fehler-Beschreibung
- Fehlercode

Hinweis:

Es kann stets nur eine globale Fehlerroutine installiert sein (...*ErrorSetDefaultProc* oder ...*ErrorSetUserProc*).

● Funktions-Deklaration

```
int meErrorSetDefaultProc(int iSwitch);
```

<iSwitch>

- ME_SWITCH_ENABLE
Installieren der vordefinierten Fehlerroutine für globale Fehleraufzeichnung.
- ME_SWITCH_DISABLE
Deinstallieren der vordefinierten Fehlerroutine.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_INVALID_SWITCH: Übergebener Action-Code wird nicht unterstützt.

meErrorSetUserProc

Beschreibung

Diese Funktion dient dazu, eine benutzerdefinierte, globale Fehlerroutine für das ME-iDS zu installieren. Diese Funktion wird automatisch aufgerufen, sobald eine Funktion einen Fehler zurückgibt. Sie erhalten folgende Infos:

- Name der Funktion, die den Fehler verursacht
- Fehlercode

Verwenden Sie die Funktion `...ErrorGetMessage()` um dem Fehlercode eine Fehlerbeschreibung zuzuordnen.

Hinweis

Es kann stets nur eine globale Fehlerroutine installiert sein (`...ErrorSetDefaultProc` oder `...ErrorSetUserProc`).

● Funktions-Deklaration

```
int meErrorSetUserProc(meErrorCB_t pErrorProc);
```

<pErrorProc>

Zeiger auf eine benutzerdefinierte Fehlerroutine. Es wird der Name der fehlerhaften Funktion und der Fehlercode an die hier „installierte“ Callback-Funktion übergeben. Durch Übergabe von NULL wird eine bereits installierte Fehlerroutine wieder deinstalliert.

→ Typ-Definition `meErrorCB_t`

```
typedef int (*meErrorCB_t)
    (char *pcFunctionName,
     int iErrorCode);
```

<pcFunctionName>

String mit dem Namen der Funktion, die den Fehler zurückgegeben hat.

<iErrorCode>

Fehlercode.

◀ Rückgabewert

Kein Fehler möglich.

meUtilityDigitalToPhysical

Beschreibung

Diese Funktion erlaubt die einfache Umrechnung der normierten Digital-Werte in die jeweilige physikalische Dimension (Spannung, Strom oder Temperatur). Die Verwendung dieser Funktion ist optional.

Wenn Sie Daten von einem Eingabe-Subdevice im Streaming-Betrieb eingelesen haben, sollten Sie vor Aufruf dieser Funktion, die Funktion *meUtilityExtractValues()* auf das erfasste Wertefeld anwenden. Nur so ist gewährleistet, dass unterschiedliche Verstärkungsfaktoren und eine Bestückung mit unterschiedlichen Aufsteck-Modulen in Verbindung mit dem ME-MultiSig-System bei der Berechnung berücksichtigt werden können!

Die Berechnung der Temperatur:

...für Widerstandssensoren erfolgt nach DIN EN 60751,

...für Thermoelemente erfolgt nach DIN EN 60584.

Wenn Sie die Funktion auf eine ganzes Wertefeld anwenden möchten, empfehlen wir die Funktion *meUtilityDigitalToPhysicalV()*.

Hinweis

Die Parameter `<dMin>` und `<dMax>` müssen mit den Bereichsgrenzen des in den Funktionen *meIOSingleConfig()* bzw. *meIOStreamConfig()* gewählten Messbereichs korrespondieren.

Die Parameter `<dMin>`, `<dMax>` und `<pdPhysical>` müssen stets in der gleichen Zehnerpotenz der jeweiligen Basiseinheit angegeben werden (z.B. entweder „mV“ oder „V“).

Die physikalische Einheit ist für die Berechnung nicht relevant.

● Funktions-Deklaration

```
int meUtilityDigitalToPhysical(double dMin, double dMax, int iMaxData,
    int iData, int iModuleType, double dRefValue, double *pdPhysical);
```

<dMin>

Untere Bereichsgrenze (von *meQueryRangeInfo()*), z. B. -10[V]. Siehe auch obiger Hinweis.

<dMax>

Obere Bereichsgrenze (von *meQueryRangeInfo()*), z. B. +10[V]. Siehe auch obiger Hinweis.

<iMaxData>

Die max. Auflösung des verwendeten Bereichs (von *meQueryRangeInfo()*), z. B. 65535 (0xFFFF) bei 16 bit Auflösung.

<iData>

Digital-Wert (0...65535) zur Konvertierung.

<iModuleType>

Hinweis: Die ME-MultiSig-Unterstützung ist in Vorbereitung. Bitte fragen Sie unser Vertriebsteam nach weiteren Details.

Falls Sie das ME-MultiSig-System mit einem Aufsteckmodul zur Signalkonditionierung einsetzen, wählen Sie hier den Modul-Typ. Übergeben Sie im Parameter <dMin> „-10“, im Parameter <dMax> „+10“ und im Parameter <iMaxData> „65535“. Dies ist wichtig, damit der Messwert richtig berechnet werden kann. Falls für die aktuelle Berechnung kein Aufsteckmodul berücksichtigt werden muss, übergeben Sie die Konstante:

- ME_MODULE_TYPE_MULTISIG_NONE
Kein Aufsteckmodul verwendet (Standard)
- ME_MODULE_TYPE_MULTISIG_DIFF16_10V
Aufsteckmodul ME-Diff16 mit Eingangsbereich 10V
- ME_MODULE_TYPE_MULTISIG_DIFF16_20V
Aufsteckmodul ME-Diff16 mit Eingangsbereich 20V
- ME_MODULE_TYPE_MULTISIG_DIFF16_50V
Aufsteckmodul ME-Diff16 mit Eingangsbereich 50V
- ME_MODULE_TYPE_MULTISIG_CURRENT16_0_20MA
Aufsteckmodul ME-Current16 mit Eingangsbereich 0...20mA
- ME_MODULE_TYPE_MULTISIG_RTD8_PT100
Aufsteckmodul ME-RTD8 für RTDs vom Typ Pt100 (0,4Ω/K)
- ME_MODULE_TYPE_MULTISIG_RTD8_PT500
Aufsteckmodul ME-RTD8 für RTDs vom Typ Pt500 (2,0Ω/K)
- ME_MODULE_TYPE_MULTISIG_RTD8_PT1000
Aufsteckmodul ME-RTD8 für RTDs vom Typ Pt1000 (4,0Ω/K)
- ME_MODULE_TYPE_MULTISIG_TE8_TYPE_B
Aufsteckmodul ME-TE8, Kanal mit Thermoelement Typ B
- ME_MODULE_TYPE_MULTISIG_TE8_TYPE_E
Aufsteckmodul ME-TE8, Kanal mit Thermoelement Typ E
- ME_MODULE_TYPE_MULTISIG_TE8_TYPE_J
Aufsteckmodul ME-TE8, Kanal mit Thermoelement Typ J
- ME_MODULE_TYPE_MULTISIG_TE8_TYPE_K
Aufsteckmodul ME-TE8, Kanal mit Thermoelement Typ K
- ME_MODULE_TYPE_MULTISIG_TE8_TYPE_N
Aufsteckmodul ME-TE8, Kanal mit Thermoelement Typ N
- ME_MODULE_TYPE_MULTISIG_TE8_TYPE_R
Aufsteckmodul ME-TE8, Kanal mit Thermoelement Typ R
- ME_MODULE_TYPE_MULTISIG_TE8_TYPE_S
Aufsteckmodul ME-TE8, Kanal mit Thermoelement Typ S

- ME_MODULE_TYPE_MULTISIG_TE8_TYPE_T
Aufsteckmodul ME-TE8, Kanal mit Thermoelement Typ T
- ME_MODULE_TYPE_MULTISIG_TE8_TEMP_SENSOR
Aufsteckmodul ME-TE8, Kanal für Vergleichsstellentemperatur an Klemmleiste des Moduls
- ME_MODULE_TYPE_MULTISIG_BA4_DMS120
Aufsteckmodul ME-BA4, Kanal mit 120Ω DMS-Nennwiderstand.
- ME_MODULE_TYPE_MULTISIG_BA4_DMS350
Aufsteckmodul ME-BA4, Kanal mit 350Ω DMS-Nennwiderstand.
- ME_MODULE_TYPE_MULTISIG_BA4_DMS1000
Aufsteckmodul ME-BA4, Kanal mit $1k\Omega$ DMS-Nennwiderstand.

<dRefValue>

- Standardeinstellung: ME_VALUE_NOT_USED.
- Falls Sie im Parameter <iModuleType> ein RTD-Modul ausgewählt haben:
Zur exakten Berechnung der Temperatur müssen Sie hier den Konstant-Meßstrom I_M in Ampere [A] übergeben. Dieser muss zuvor mit einem hochgenauen Amperemeter gemessen werden (siehe Handbuch ME-MultiSig-System).
Falls Sie die Meßtoleranz vernachlässigen möchten, rechnet die Funktion mit einem typischen Konstant-Meßstrom von $I_M = 500 \times 10^{-6}$ A. Übergeben Sie in diesem Fall die Konstante:
ME_REFVALUE_MULTISIG_I_MEASURED_DEFAULT.
- Falls Sie im Parameter <iModuleType> ein Thermoelementen-Modul ausgewählt haben:
Da sich die Berechnung auf eine Vergleichsstellentemperatur von 0°C bezieht, müssen Sie mit dem integrierten Sensor des jeweiligen Aufsteckmoduls vor Beginn der Messreihe die Temperatur an der Klemmleiste STM1 messen (siehe Parameter <ModuleType>). Anschließend wird der ermittelte Wert in diesem Parameter übergeben (in $^\circ\text{C}$). Parameter <Physical> gibt einen Zeiger auf den kompensierten Temperaturwert in $^\circ\text{C}$ zurück.

<pdPhysical>

Ergebnis in der jeweiligen physikalischen Einheit [V], [A], [$^\circ\text{C}$].

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_VALUE_OUT_OF_RANGE: Übergebener Wert ist kleiner als "0" oder größer als <iMaxData>.
- ME_ERRNO_INVALID_MIN_MAX: Übergebener Wert für <iMaxData> ist nicht gültig.

- ME_ERRNO_INVALID_MODULE_TYPE: Übergebener Modultyp wird nicht unterstützt.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL.

meUtilityDigitalToPhysicalV

Beschreibung

Im Gegensatz zur Funktion *meUtilityDigitalToPhysical()* kann diese Funktion auf ein ganzes Wertefeld angewandt werden. Die Funktionsdeklaration wurde dafür um den Parameter `<iCount>` erweitert. Ansonsten gilt die Beschreibung der Funktion *meUtilityDigitalToPhysical()* sinngemäß.

● Funktions-Deklaration

```
int meUtilityDigitalToPhysicalV(double dMin, double dMax, int
    iMaxData, int *piDataBuffer, int iCount, int iModuleType, double
    dRefValue, double *pdPhysicalBuffer);
```

<dMin>

Untere Bereichsgrenze (von *meQueryRangeInfo()*).

<dMax>

Obere Bereichsgrenze (von *meQueryRangeInfo()*).

<iMaxData>

Max. Auflösung des verwendeten Bereichs (von *meQueryRangeInfo()*).

<piDataBuffer> (w)

Zeiger auf Wertefeld mit Digital-Werten, die konvertiert werden sollen.

<iCount>

Anzahl der Werte im Wertefeld.

<iModuleType>

Siehe Funktion *meUtilityDigitalToPhysical()*.

<dRefValue>

Siehe Funktion *meUtilityDigitalToPhysical()*.

<pdPhysicalBuffer> (r)

Zeiger auf Wertefeld mit den errechneten Werten in der jeweiligen physikalischen Einheit [V], [A], [°C].

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_VALUE_OUT_OF_RANGE: Übergebener Wert ist kleiner als "0" oder größer als `<iMaxData>`.
- ME_ERRNO_INVALID_MIN_MAX: Übergebener Wert für `<iMaxData>` ist nicht gültig.
- ME_ERRNO_INVALID_MODULE_TYPE: Übergebener Modultyp wird nicht unterstützt.
- ME_ERRNO_INVALID_POINTER: Übergebene Zeiger sind NULL.

meUtilityPhysicalToDigital

Beschreibung

Diese Funktion erlaubt Ihnen die einfache Umrechnung der auszugebenden Spannungs- oder Stromwerte in normierte Digital-Werte. Die Verwendung dieser Funktion ist optional.

Wenn Sie die Funktion auf ein ganzes Wertefeld anwenden möchten, empfehlen wir die Funktion *meUtilityPhysicalToDigitalV()*.

Hinweis

Die Parameter `<dMin>` und `<dMax>` müssen mit den Bereichsgrenzen des in den Funktionen *meIOSingleConfig()* bzw. *meIOStreamConfig()* gewählten Messbereichs korrespondieren.

Die Parameter `<dMin>`, `<dMax>` und `<pdPhysical>` müssen stets in der gleichen Zehnerpotenz der jeweiligen Basiseinheit angegeben werden (z.B. entweder „mV“ oder „V“).

Die physikalische Einheit ist für die Berechnung nicht relevant.

● Funktions-Deklaration

```
int meUtilityPhysicalToDigital(double dMin, double dMax, int iMaxData,
    double dPhysical, int *piData);
```

<dMin>

Untere Bereichsgrenze (von *meQueryRangeInfo()*), z. B. -10[V]. Siehe auch obiger Hinweis.

<dMax>

Obere Bereichsgrenze (von *meQueryRangeInfo()*), z. B. +10[V]. Siehe auch obiger Hinweis.

<iMaxData>

Die max. Auflösung des verwendeten Bereichs (von *meQueryRangeInfo()*), z. B. 65535 (0xFFFF) bei 16 bit Auflösung.

<dPhysical>

Umzurechnender Spannungs- oder Stromwert, z. B. +0,75[V].

<piData>

Ergebnis ist der auszugebende Digital-Wert.

◀ Rückgabewert

- ME_ERRNO_VALUE_OUT_OF_RANGE: Übergebener Wert ist kleiner als `<dMin>` oder größer als `<dMax>`
- ME_ERRNO_INVALID_MIN_MAX: Übergebener Wert für `<iMaxData>` ist nicht gültig.
- ME_ERRNO_INVALID_POINTER: Übergebener Zeiger ist NULL

meUtilityPhysicalToDigitalV

Beschreibung

Im Gegensatz zur Funktion *meUtilityPhysicalToDigital()* kann diese Funktion auf ein ganzes Wertefeld angewandt werden. Die Funktionsdeklaration wurde dafür um den Parameter `<iCount>` erweitert. Ansonsten gilt die Beschreibung der Funktion *meUtilityPhysicalToDigital()* sinngemäß.

● Funktions-Deklaration

```
int meUtilityPhysicalToDigitalV(double dMin, double dMax, int
                               iMaxData, double *pdPhysicalBuffer, int iCount, int *piDataBuffer);
```

<dMin>

Untere Bereichsgrenze (von *meQueryRangeInfo()*).

<dMax>

Obere Bereichsgrenze (von *meQueryRangeInfo()*).

<iMaxData>

Max. Auflösung des verwendeten Bereichs (von *meQueryRangeInfo()*).

<pdPhysicalBuffer> (w)

Zeiger auf Wertefeld mit den umzurechnenden Spannungs- oder Stromwerten, z. B. +0,75[V].

<iCount>

Anzahl der Werte im Wertefeld.

<piDataBuffer> (r)

Zeiger auf Wertefeld mit den auszugebenden Digital-Werten. Muß groß genug sein um `<iCount>` Werte zu erfassen.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_VALUE_OUT_OF_RANGE: Übergebener Wert ist kleiner als `<dMin>` oder größer als `<dMax>`
- ME_ERRNO_INVALID_MIN_MAX: Übergebener Wert für `<iMaxData>` ist nicht gültig.
- ME_ERRNO_INVALID_POINTER: Übergebene Zeiger sind NULL.

meUtilityExtractValues

Beschreibung

Diese Hilfsfunktion extrahiert aus einem Wertefeld, das von der Funktion *meIOStreamRead()* angelegt wurde die Werte des spezifizierten Kanals unter Berücksichtigung der Kanalliste. Um die Daten für mehrere Kanäle zu extrahieren, muss die Funktion für jeden Kanal getrennt aufgerufen werden.

Hinweis: Falls der Kanal nicht in der Kanalliste ist, gibt die Funktion ME_ERRNO_SUCCESS zurück und <piChanBufferCount> wird auf „0“ gesetzt.

● Funktions-Deklaration

```
int meUtilityExtractValues(int iChannel, int *piAIBuffer, int
                           iAIBufferCount, meIOStreamConfig_t *pConfigList, int
                           iConfigListCount, int *piChanBuffer, int *piChanBufferCount);
```

<iChannel>

Kanal-Index dessen Werte extrahiert werden sollen.

<piAIBuffer> (w)

Zeiger auf den von der Funktion *meIOStreamRead()* angelegten Datenpuffer.

<iAIBufferCount>

Anzahl der Messwerte im Datenpuffer <piAIBuffer>.

<pConfigList> (w)

Zeiger auf Kanalliste, die an die Funktion *meIOStreamConfig()* übergeben wurde.

<iConfigListCount>

Anzahl der Kanallisteneinträge in <pConfigList>.

<piChanBuffer> (r)

Zeiger auf Wertefeld in dem die extrahierten Werte des spezifizierten Kanals abgelegt werden.

<piChanBufferCount> (r/w)

(w) : Übergabe der Größe des Wertefeldes <piChanBuffer> in Anzahl der Werte.

(r) : Die Funktion gibt die Anzahl der tatsächlich in <piChanBuffer> abgelegten Werte zurück.

→ Typ-Definition meIOStreamConfig_t

Typdefinition siehe Funktion *meIOStreamConfig()* ab Seite 164.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_INVALID_POINTER: Übergebene Zeiger sind NULL.

meUtilityPWMStart

Beschreibung

Diese Hilfsfunktion konfiguriert den Zählerbaustein 8254 (ME_SUBTYPE_CTR_8254) für die Betriebsart „Pulsweiten-Modulation“ (PWM) und startet die Ausgabe. Eine anderweitige Nutzung der Zähler 0...2 ist in dieser Betriebsart nicht möglich. Das Signal steht an OUT_2 des spezifizierten Zählerbausteins zur Verfügung. Je nach Gerätetyp muss der Basistakt (max. 10 MHz) entweder von außen zugeführt werden oder es wird (sofern vorhanden) ein Quarzoszillator „on Board“ verwendet. Zähler 0 wird als Vorteiler verwendet. Die Frequenz des Ausgangssignals kann max. 50 kHz betragen und errechnet sich folgendermaßen:

$$f_{\text{OUT}_2} = \frac{\text{Basistakt}}{\langle \text{iPrescaler} \rangle \cdot 100} \quad (\text{mit } \langle \text{iPrescaler} \rangle = 2 \dots (2^{16} - 1))$$

Das Tastverhältnis kann zwischen 1...99% in Schritten von 1% eingestellt werden (siehe Abb. 32 auf Seite 227).

Hinweis!

Die Verwendung dieser Funktion ist nur sinnvoll in Verbindung mit der in Abb. 31 auf Seite 226 gezeigten externen Beschaltung.

● Funktions-Deklaration

```
int meUtilityPWMStart(int iDevice, int iSubdevice1, int iSubdevice2, int
    iSubdevice3, int iRef, int iPrescaler, int iDutyCycle, int iFlags);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice1>

Index von Subdevice „Zähler 0“ (Vorteiler).

<iSubdevice2>

Index von Subdevice „Zähler 1“.

<iSubdevice3>

Index von Subdevice „Zähler 2“.

<iRef>

Definiert die Taktquelle für Zähler 0 (CLK_0):

- ME_REF_CTR_INTERNAL_1MHZ
Taktquelle ist der interne 1 MHz Quarzoszillator.
- ME_REF_CTR_INTERNAL_10MHZ
Taktquelle ist der interne 10 MHz Quarzoszillator.
- ME_REF_CTR_EXTERNAL
Taktquelle ist ein externer Taktgenerator.

<iPrescaler>

Wert für Zähler 0 (Vorteiler) im Bereich 2...65535.

<iDutyCycle>

Tastverhältnis des Ausgangssignals von 1% –99% in 1%-Schritten einstellbar.

<iFlags>

Flag für erweiterte Funktionen:

- ME_PWM_START_NO_FLAGS: Standardeinstellung
- ME_PWM_START_CONNECT_INTERNAL
Falls von der Hardware unterstützt (z. B. ME-1400-Serie), wird OUT_1 mit GATE_2 intern verbunden. Dies reduziert die Anzahl externer Verbindungen.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_INVALID_FLAGS: Nicht unterstütztes Flag verwendet.
- ME_ERRNO_INVALID_REF: Signalquelle ist nicht verfügbar.
- ME_ERRNO_INVALID_DUTY_CYCLE: Wert außerhalb des unterstützten Bereichs.
- ME_ERRNO_NOT_SUPPORTED:
 - Verwendetes Subdevice ist kein Zähler
 - Interne Verbindung nicht möglich

meUtilityPWMStop

Beschreibung

Mit dieser Funktion wird der mit der Funktion *meUtilityPWMStart()* gestartete PWM-Betrieb beendet.

● Funktions-Deklaration

```
int meUtilityPWMStop(int iDevice, int iSubdevice1);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice1>

Index von Subdevice „Zähler 0“ (Vorteiler).

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_NOT_SUPPORTED: Subdevice ist kein Zähler.

meUtilityPWMRestart

Beschreibung

Hilfsfunktion zum erneuten Starten des PWM-Betriebs nach Stop durch *meUtilityPWMStop()*. Der Vorteiler kann mit einem neuen Wert geladen werden um eine andere Frequenz einzustellen. Das Tastverhältnis kann hier nicht geändert werden.

Hinweis

Die Verwendung dieser Funktion ist nur sinnvoll in Verbindung mit der in Abb. 31 auf Seite 226 gezeigten externen Beschaltung.

Die Operation startet dort wo sie angehalten wurde. Kein Reset!

● Funktions-Deklaration

```
int meUtilityPWMRestart(int iDevice, int iSubdevice1, int iRef, int
                        iPrescaler);
```

<iDevice>

Index des Geräts auf das zugegriffen werden soll.

<iSubdevice1>

Index von Subdevice „Zähler 0“ (Vorteiler).

<iRef>

Definiert die Taktquelle für Zähler 0 (CLK_0):

- ME_REF_CTR_INTERNAL_1MHZ
Taktquelle ist der interne 1 MHz Quarzoszillator.
- ME_REF_CTR_INTERNAL_10MHZ
Taktquelle ist der interne 10 MHz Quarzoszillator.
- ME_REF_CTR_EXTERNAL
Taktquelle ist ein externer Taktgenerator.

<iPrescaler>

Wert für Zähler 0 (Vorteiler) im Bereich 2...65535.

◀ Rückgabewert

- ME_ERRNO_SUCCESS: Funktion erfolgreich zurückgekehrt.
- ME_ERRNO_NOT_OPEN: ME-iDS ist nicht korrekt geöffnet.
- ME_ERRNO_INVALID_DEVICE: Dem angefragten Device-Index ist kein Gerät zugeordnet.
- ME_ERRNO_INVALID_SUBDEVICE: Dem angefragten Subdevice-Index ist kein Subdevice zugeordnet.
- ME_ERRNO_NOT_SUPPORTED: Subdevice ist kein Zähler.

- ME_ERRNO_INVALID_REF: Signalquelle ist nicht verfügbar.

Anhang

A Technische Fragen

A1 Fax-Hotline

Sollten Sie technische Fragen oder Probleme haben, die auf ein Meilhaus-Gerät zurückzuführen sind, dann schicken Sie bitte eine ausführliche Problembeschreibung an unsere Hotline:

Fax-Hotline: (++49) (0)89 - 89 01 66-28

eMail: support@meilhaus.de

A2 Serviceadresse

Sollte bei Ihrem Gerät ein technischer Defekt auftreten, wenden Sie sich bitte an:

Meilhaus Electronic GmbH

Abteilung Reparaturen

Fischerstraße 2

D-82178 Puchheim

Falls Sie ein Gerät zur Reparatur an uns zurücksenden wollen, fordern Sie unbedingt zuerst eine RMA-Nummer an und beachten Sie die Hinweise zur Abwicklung des RMA-Verfahrens. Legen Sie bitte eine ausführliche Fehlerbeschreibung bei, inkl. Angaben zu Ihrem Rechner/System und verwendeter Software!

A3 Treiber-Update

Unter www.meilhaus.de/download stehen Ihnen stets die aktuellen Treiber für Meilhaus-Geräte sowie unsere Handbücher im PDF-Format zur Verfügung.

B Spezielle Betriebsarten

B1 Betriebsarten 8254

Das ME-iDS unterstützt den Standard-Zählerbaustein vom Typ 8254, der über drei 16 bit Zähler verfügt (siehe auch Datenblatt des Chip-Herstellers). Jeder Zähler wird als Subdevice vom Typ ME_TYPE_CTR, Untertyp ME_SUBTYPE_SINGLE betrachtet.

- Modus 0: Zustandsänderung bei Nulldurchgang
(ME_SINGLE_CONFIG_CTR_8254_MODE_0)
- Modus 1: Retriggerbarer „One Shot“
(ME_SINGLE_CONFIG_CTR_8254_MODE_1)
- Modus 2: Asymmetrischer Teiler
(ME_SINGLE_CONFIG_CTR_8254_MODE_2)
- Modus 3: Symmetrischer Teiler
(ME_SINGLE_CONFIG_CTR_8254_MODE_3)
- Modus 4: Zählerstart durch Softwaretrigger
(ME_SINGLE_CONFIG_CTR_8254_MODE_4)
- Modus 5: Zählerstart durch Hardwaretrigger
(ME_SINGLE_CONFIG_CTR_8254_MODE_5)

Hinweis: Der tatsächliche Spannungspegel an den Ein-/Ausgängen des Zählers hängt von der jeweiligen Hardware ab. Bei optoisolierten Varianten der ME-4600 Serie z. B. entspricht ein High-Pegel am Ausgang dem Zustand „hochohmig“ und ein Low-Pegel dem Zustand „leitend“. Bitte beachten Sie das entsprechende Hardware-Handbuch. Die Logik-Pegel in der folgenden Beschreibung gelten für den Zählerbaustein ohne weitere Beschaltung.

Modus 0: Zustandsänderung bei Nulldurchgang

Diese Betriebsart ist z. B. zur Signalisierung eines Interrupts bei Nulldurchgang geeignet. Der Zähler-Ausgang (OUT_0...2) geht in den Low-Zustand, sobald der Zähler initialisiert wird oder ein neuer Startwert in den Zähler geladen wird. Zur Freigabe des Zählers muss der GATE-Eingang mit High-Pegel beschaltet werden. Sobald der Zähler geladen und freigegeben wurde, beginnt er abwärts zu zählen während der Ausgang im Low-Zustand bleibt.

Bei Erreichen des Nulldurchganges geht der Ausgang in den High-Zustand und bleibt dort, bis der Zähler neu initialisiert wird oder ein neuer Startwert geladen wird. Auch nach Erreichen des Nulldurchganges wird weiter abwärts gezählt. Sollte während des Zählvorganges ein Zählerregisters erneut geladen werden, hat dies zur Folge, dass:

1. beim Schreiben des ersten Bytes der momentane Zählvorgang gestoppt wird
2. beim Schreiben des zweiten Bytes der neue Zählvorgang gestartet wird.

Modus 1: Retriggerbarer „One-Shot“

Der Zähler-Ausgang (OUT_0...2) geht in den High-Zustand, sobald der Zähler initialisiert wird. Nachdem ein Startwert in den Zähler geladen wurde geht der Ausgang mit dem auf den ersten Triggerimpuls am GATE-Eingang folgenden Takt in den Low-Zustand. Nach Ablauf des Zählers geht der Ausgang wieder in den High-Zustand.

Mit einer geeigneten Flanke am GATE-Eingang kann der Zähler jederzeit auf den Startwert zurückgesetzt („retriggered“) werden. Der Ausgang bleibt solange im Low-Zustand bis der Zähler den Nulldurchgang erreicht.

Der Zählerstand kann jederzeit ohne Auswirkung auf den momentanen Zählvorgang, ausgelesen werden.

Modus 2: Asymmetrischer Teiler

In diesem Modus arbeitet der Zähler als Frequenzteiler. Der Zähler-Ausgang (OUT_0...2) geht nach der Initialisierung in den High-Zustand. Nach Freigabe des Zählers durch geeignete Beschaltung des GATE-Eingangs wird abwärts gezählt, während der Ausgang noch im High-Zustand verbleibt. Sobald der Zähler den Wert 0001Hex erreicht hat, geht der Ausgang für die Dauer einer Taktperiode in den Low-Zustand. Dieser Ablauf wiederholt sich periodisch solange der GATE-Eingang freigegeben ist, ansonsten geht der Ausgang sofort in den High-Zustand.

Wird das Zählerregister zwischen zwei Ausgangs-Pulsen erneut geladen, so beeinflusst dies den momentanen Zählvorgang nicht, die nächste Periode arbeitet jedoch mit den neuen Werten.

Modus 3: Symmetrischer Teiler

Dieser Modus arbeitet ähnlich wie Modus 2 mit dem Unterschied, dass der geteilte Takt ein symmetrisches Tastverhältnis besitzt (nur für geradzählige Zählerwerte geeignet). Der Zähler-Ausgang (OUT_0...2) geht nach der Initialisierung in den High-Zustand. Nach Freigabe des Zählers durch geeignete Beschaltung des GATE-Eingangs wird in 2er-Schritten abwärts gezählt. Nun wechselt der Ausgang, mit der Anzahl Perioden des halben Startwertes bezogen auf den Eingangstakt (beginnend mit High-Pegel). Solange der Gate-Eingang freigegeben ist, wiederholt sich dieser Ablauf periodisch, ansonsten geht der Ausgang sofort in den High-Zustand.

Wird das Zählerregister zwischen zwei Ausgangs-Pulsen erneut geladen, so beeinflusst dies den momentanen Zählvorgang nicht, die nächste Periode arbeitet jedoch mit den neuen Werten.

Modus 4: Zählerstart durch Softwaretrigger

Der Zähler-Ausgang (OUT_0...2) geht in den High-Zustand, sobald der Zähler initialisiert wird. Zur Freigabe des Zählers muss der Gate-Eingang geeignet beschaltet werden. Sobald der Zähler geladen (Software-Trigger) und freigegeben wurde, beginnt er abwärts zu zählen, während der Ausgang noch im High-Zustand bleibt.

Bei Erreichen des Nulldurchganges geht der Ausgang für die Dauer einer Takt-Periode in den Low-Zustand. Danach geht der Ausgang wieder in den High-Zustand und bleibt dort bis der Zähler initialisiert und ein neuer Startwert geladen wird.

Wird das Zählerregister während eines Zählvorganges erneut geladen, so wird der neue Startwert mit dem nächsten Takt geladen.

Modus 5: Zählerstart durch Hardwaretrigger

Der Zähler-Ausgang (OUT_0...2) geht in den High-Zustand, sobald der Zähler initialisiert wird. Nachdem ein Startwert in den Zähler geladen wurde beginnt der Zählvorgang mit dem auf den ersten Triggerimpuls am GATE-Eingang folgenden Takt. Bei Erreichen des Nulldurchganges geht der Ausgang für die Dauer einer Takt-Periode in den Low-Zustand. Danach geht der Ausgang wieder in den High-Zustand und bleibt dort bis ein erneuter Triggerimpuls ausgelöst wird.

Wird das Zählerregister zwischen zwei Triggerimpulsen erneut geladen, so wird der neue Startwert erst nach dem nächsten Triggerimpuls berücksichtigt.

Mit einer positiven Flanke am Gate-Eingang kann der Zähler jederzeit auf den Startwert zurückgesetzt („retriggered“) werden. Der Ausgang bleibt solange im High-Zustand bis der Zähler den Nulldurchgang erreicht.

B2 Pulsweiten-Modulation

Ein spezieller Anwendungsfall der Zählerbausteine vom Typ 8254 ist die Ausgabe eines Rechtecksignals mit variablem Tastverhältnis (Betriebsart „PWM“). Damit können Sie an OUT_2 ein Rechteck-Signal von max. 50 kHz bei variablem Tastverhältnis ausgeben. Voraussetzung ist die geeignete Beschaltung der Ein- und Ausgänge (CLK, GATE, OUT) durch externe Verdrahtung. (siehe Abb. 31 for TTL Ein-/Ausgänge). Für opto-isolierte Zähler lesen Sie bitte das entsprechende Kapitel zur PWM-Beschaltung opto-isolierter Zähler im Hardware-Handbuch.

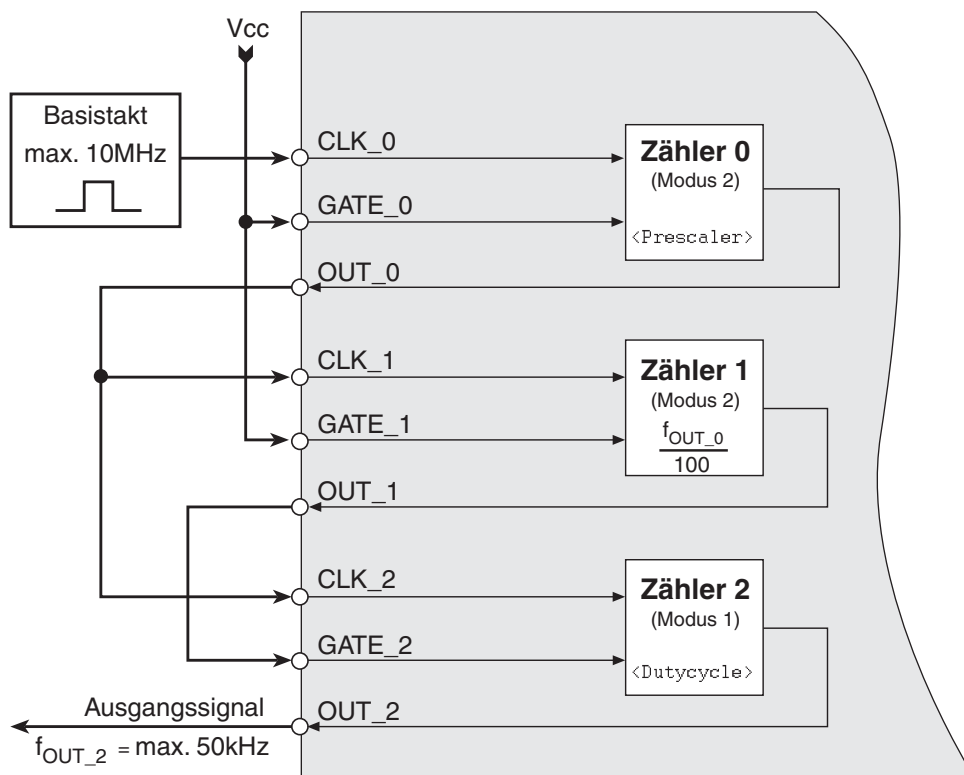


Abbildung 31: Beschaltung Pulsweiten-Modulation

Zähler 0 wird als Vorteiler für den extern eingespeisten Basistakt verwendet. Über den Parameter `<iPrescaler>` können Sie die Frequenz f_{OUT_2} folgendermaßen einstellen:

$$f_{OUT_2} = \frac{\text{Basistakt}}{\langle iPrescaler \rangle \cdot 100} \quad (\text{mit } \langle iPrescaler \rangle = 2 \dots (2^{16} - 1))$$

Mit dem Parameter `<iDutyCycle>` kann das Tastverhältnis zwischen 1...99% in Schritten von 1% eingestellt werden (siehe Abb. 32). Die Ausgabe wird unmittelbar durch Aufruf der Funktion `meUtilityPWMStart()` gestartet und mit `meUtilityPWMStop()` beendet. Es ist keine weitere Programmierung der Zähler erforderlich.

Beachten Sie, dass bei optoisolierten Geräten der Ausgang OUT_2 in der Regel als „Open Collector“ Ausgang ausgeführt ist. D. h. logisch „1“ bedeutet, dass der Ausgang leitend ist und logisch 0, dass der Ausgang hochohmig ist (siehe auch Hardware-Handbuch).

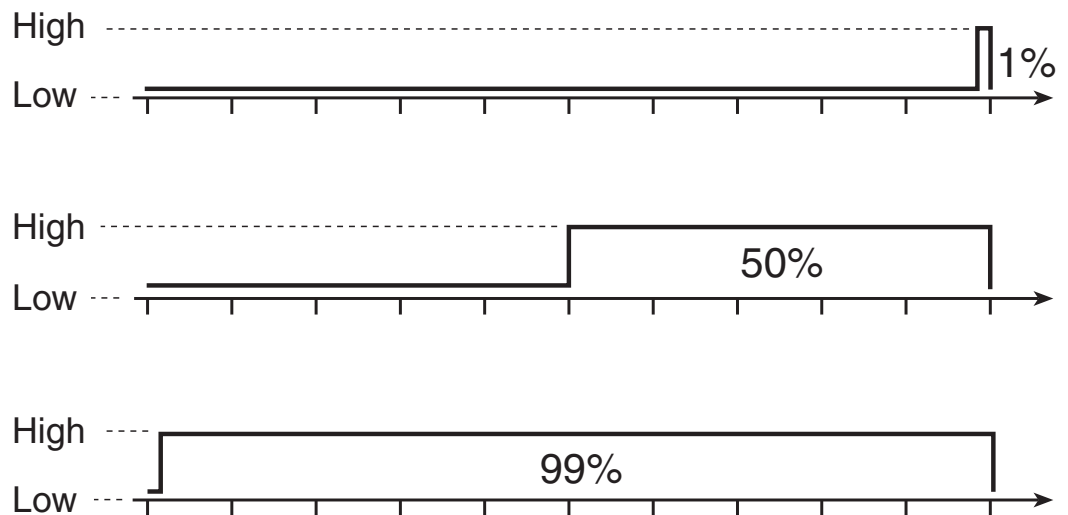


Abbildung 32: Tastverhältnis PWM-Signal

B3 Bitmuster-Ausgabe der ME-4680

Dieses Kapitel beschreibt einige Besonderheiten der timergesteuerten Bitmuster-Ausgabe, wie sie auf der ME-4680 implementiert ist. Hardwaretechnisch wird hierzu das FIFO von D/A-Kanal 3 „zweckentfremdet“. Getrennt nach „Low-Byte“ und „High-Byte“ können die 16 Bit breiten FIFO-Werte (= Bitmuster) byteweise den 8 Bit breiten Digital-Ports (Subdevice 0, 1, 2, 3) zugeordnet werden (siehe Abb. 33 auf Seite 229). Ein für die Bitmuster-Ausgabe verwendeter Port ist automatisch Ausgangs-Port. Eingangsport B (Subdevice 1) der optoisolierten Versionen kann nicht für die Bitmuster-Ausgabe verwendet werden.

Die Programmierung erfolgt in der Betriebsart Streaming. Ein für die Bitmuster-Ausgabe verwendeter Digital-Port muss ein Subdevice vom Typ `ME_TYPE_DO` oder `ME_TYPE_DIO` sein mit Untertyp `ME_SUBTYPE_STREAMING`. Folgende Parameter können mit den Funktionen *meIOSingleConfig()* und *meIOStreamConfig()* konfiguriert werden.

- Konfiguration eines oder mehrerer digitaler Ausgangsports für die timer-gesteuerte Bitmuster-Ausgabe mit der Konstante `ME_SINGLE_CONFIG_DIO_BIT_PATTERN` im Parameter `<iSingleConfig>` der Funktion *meIOSingleConfig()*.
- Zuweisung von Low-Byte und High-Byte des 16 bit breiten FIFO-Wertes zum spezifizierten Digital-Port mit den Konstanten `ME_REF_FIFO_LOW` bzw. `ME_REF_FIFO_HIGH` im Parameter `<iRef>` der Funktion *meIOSingleConfig()*.
- Das Subdevice von D/A-Kanal 3 (Subdevice mit Index 11 vom Typ `ME_TYPE_AO`) wird mit der Konstante `ME_IO_STREAM_CONFIG_BIT_PATTERN` im Parameter `<iFlags>` der Funktion *meIOStreamConfig()* für Bitmuster-Ausgabe konfiguriert.
- Als Referenz muss im Parameter `<iRef>` der Funktion *meIOStreamConfig()* die Konstante `ME_REF_AO_GROUND` angegeben werden. Für die externe Beschaltung gilt jedoch die Masse des Digital-I/O-Teils (`PC_GND` bzw. `DIO_GND`) als Bezugsmasse (nicht die des AO-Teils!).

- Triggerkanal, Triggertyp und Triggerflanke werden über die Triggerstruktur `meIOStreamTrigger` in der Funktion `meIOStreamConfig()` definiert.
- Als Zeitgeber dient ein programmierbarer Zähler, der über die Triggerstruktur `meIOStreamTrigger` konfiguriert wird. Der 32 bit Zähler verwendet einen 33 MHz Takt als Zeitbasis. Daraus ergibt sich eine Periodendauer von $30,30\overline{\text{ns}}$, die als kleinste Zeiteinheit definiert wird und im Folgenden „1 Tick“ genannt wird. Zur einfachen Umrechnung von Frequenz bzw. Periodendauer in Ticks können Sie die Funktionen `meIOStreamFrequencyToTicks()` oder `meIOStreamTimeToTicks()` verwenden.

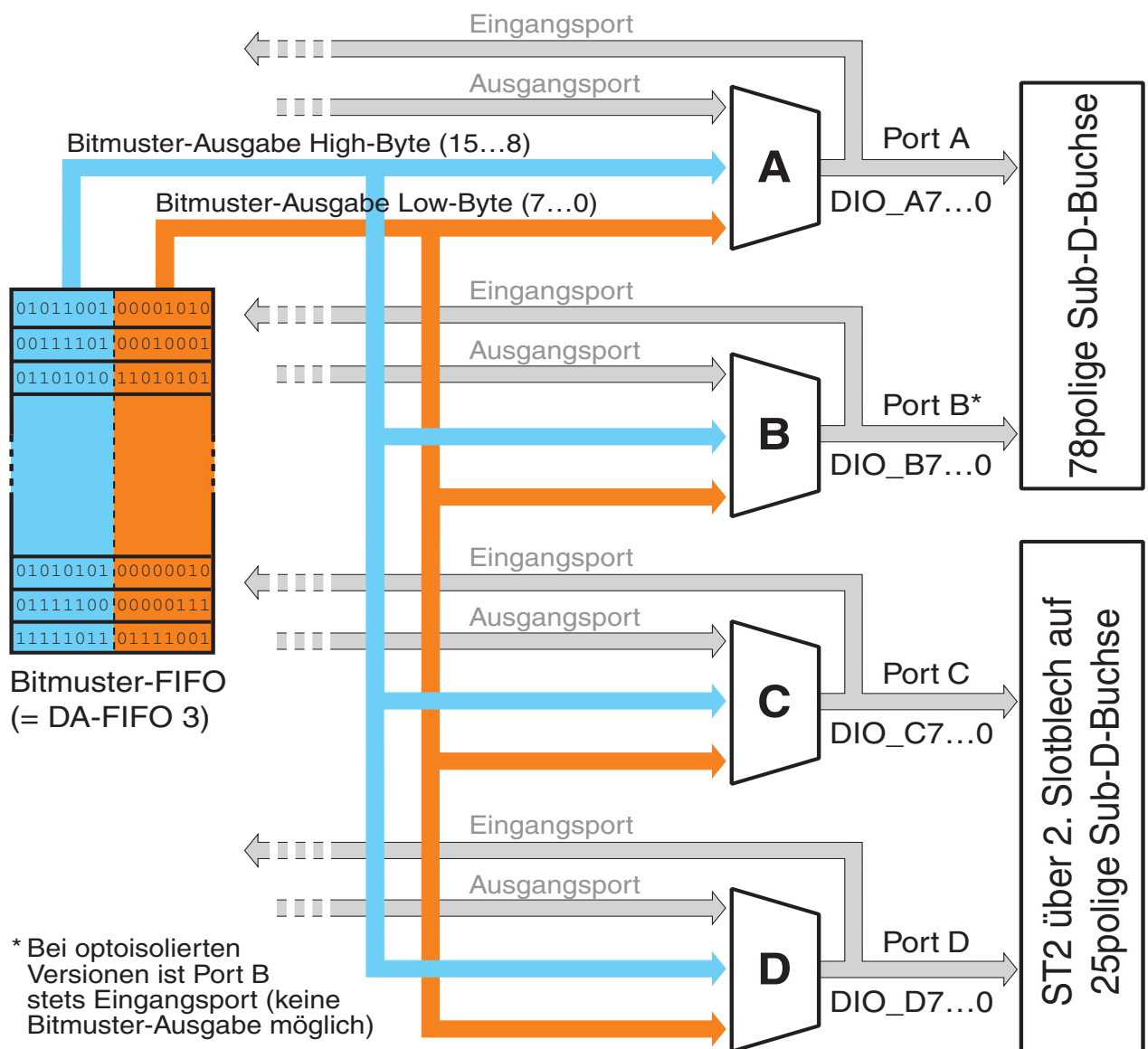


Abbildung 33: Port-Mapping

B4 MEphisto Scope

Spezielle Funktionen und Einschränkungen

Eine Demonstration zur Anwendung der speziellen Funktionen des MEphisto Scope im Streaming-Betrieb ist im SDK enthalten. Das C++ Beispiel-Programm „Con_meIOMephistoScopeStreamRead“ (im Source-Code) finden Sie im Verzeichnis „MS Visual C“ des ME-iDS-SDK.

Durch Verwendung der Funktion *meIOSetChannelOffset()* kann der analoge Eingangsbereich verschoben werden. Dies erlaubt es ein anliegendes Signal sehr genau zu messen, wenn Sie vorher die Minimal- und Maximalwerte in etwa kennen. Die Offset-Einstellung ist derzeit nur im Streaming-Betrieb möglich, im Single-Betrieb wird stets ein Offset von 0,0 Volt verwendet. Falls Sie die Funktion *meIOSetChannelOffset()* zur Offset-Einstellung verwenden, müssen Sie das Endergebnis durch Addition des entsprechenden Offsetwertes korrigieren.

Im Gegensatz zu anderen Geräten im ME-iDS kann das MEphisto Scope (UM202, UM203) nur von einer Applikation zur gleichen Zeit genutzt werden. Falls mehrere MEphisto Scopes in Ihrem System vorhanden sind, „sieht“ die erste Applikation, die gestartet wird, eine Liste aller Geräte. Solange diese Applikation läuft, ist dieses MEphisto Scope für später gestartete Applikationen „unsichtbar“ und kann nicht angesprochen werden. Sobald die erste Applikation beendet wird, wird das MEphisto Scope für die als nächstes gestartete Applikation „sichtbar“.

Das Subdevice 0 „Digitale Ein-/Ausgabe“ und das Subdevice 1 „Analoge Ein-/Ausgabe“ sind nicht völlig unabhängig. Falls eine Streaming-Operation des analogen Subdevices aktiv ist, führt der Versuch auf das Subdevice für digitale Ein-/Ausgabe zuzugreifen zu einem Fehler (sub-device busy, error 33). Sobald auf einem der Subdevices eine Operation läuft, muss jeder andere Thread solange warten, bis die erste Operation beendet ist.

Im Single-Betrieb ist das Ergebnis welches von der Funktion *meIO-Single()* zurückgeben wird, der Mittelwert einer großen Anzahl von Messungen, die über eine Zeitdauer von ungefähr 0,9 s erfaßt wurden. Während dieser Zeit muss jeder andere Thread, der auf eines der beiden Subdevices zugreifen möchte, warten wie im vorigen Absatz beschrieben.

B5 ME-MultiSig-Steuerung

Eine umfangreiche Unterstützung des ME-MultiSig-Systems befindet sich in Vorbereitung (fragen Sie unser Vertriebsteam für weitere Details). Daher ist die Beschreibung in diesem Kapitel als vorläufig zu betrachten.

Zum Verständnis des ME-MultiSig-Systems empfehlen wir dringend das Handbuch des ME-MultiSig-System vorher und vollständig zu lesen! Im Folgenden finden Sie einige Anmerkungen, die die Programmierung in Verbindung mit dem ME-iDS betreffen:

- Im ME-iDS kann die Programmierung des ME-MultiSig-Systems völlig transparent erfolgen. „Transparent“ heißt, dass Sie z. B. eine MUX-Kette bestehend aus einer Master- und einer Slave-Karte einfach als A/D-Funktionsgruppe (Subdevice vom Typ „AI“) mit 64 Kanälen ansprechen können.
- Voraussetzung für die transparente Programmierung ist die „Anmeldung“ der verwendeten Basiskarten (ME-MUX32-M(aster), ME-MUX32-S(lave), ME-DEMUX32) im ME-iDC. Dort legen Sie Anzahl und Typ der Basiskarten fest. Eine MUX-Kette kann maximal aus einem ME-MUX32-M und bis zu sieben ME-MUX32-S oder einem ME-DEMUX32 bestehen.
- Die Hardware der Basiskarten vom Typ ME-MUX32-M/S muss für die Betriebsart „Single-Mux“ konfiguriert sein (siehe Handbuch ME-MultiSig-System).
- Der A/D-Kanal der Karte, der für die „Mux-Kette“ verwendet werden soll, muss im ME-iDC eingestellt werden (Standard: A/D-Kanal 0). Die Kanal-Nummer muss mit der Lötbrücke „A“ auf der Masterkarte (ME-MUX32-M) korrespondieren (siehe Handbuch ME-MultiSig-System).
- Zur Nutzung des vollen Funktionsumfangs im „Mux“-Betrieb wird ein Subdevice vom Typ „ME_TYPE_DIO“ oder „ME_TYPE_DO“ mit min. 16 digitalen Ausgängen benötigt. Diese können aus einem Pool geeigneter Ports im ME-iDC ausgewählt werden. Beachten Sie, dass die Verkabelung damit korrespondieren muss.
- Die Umrechnung der Digital-Werte in die entsprechende physikalische Einheit (Spannung, Strom, Temperatur) und

umgekehrt erfolgt durch die Funktionen *meUtilityDigitalToPhysical()* bzw. *meUtilityPhysicalToDigital()* unter Berücksichtigung der verwendeten Aufsteck-Module.

- Beachten Sie bei der Konfiguration, dass der Verstärkungsfaktor innerhalb einer Kanalgruppe der Basiskarten ME-MUX32-M(aster) und ME-MUX32-S(lave) gleich sein muss (siehe Funktionen *meIO-SingleConfig()* bzw. *meIOStreamConfig()*).
- Die Adress-LED der Basiskarten ME-MUX32-M und ME-MUX32-S kann über den Parameter <iFlags> der Funktion *meIO-SingleConfig()* angesteuert werden.
- Mit Hilfe der Funktion *meIOResetDevice()* bzw. *meIOResetSubdevice()* können Sie alle Master- und Slave-Karten in den Grundzustand setzen (Verstärkung V=1, Adress-LEDs aus).
- Die Verwendung der Betriebsarten „Stream-Input“ bzw. „Stream-Output“ in Verbindung mit dem ME-MultiSig-System ist nur nach „Anmeldung“ der Basiskarten im ME-iDC mit der Option „Use streaming operation mode“ möglich. Sie benötigen für diese Betriebsart eine Multi-I/O-Karte vom Typ ME-4680. D/A-Kanal 3 wird in diesem Fall für timergesteuerte Ausgabe gesperrt.
- Sobald Sie eine MUX-Basiskarte vom Typ ME-MUX32-M im ME-iDC anmelden, können Sie die restlichen A/D-Kanäle der Karte nicht ansprechen. Das „AI“-Subdevice umfaßt in diesem Fall nur die Kanäle der MUX-Kette (max. 256 Kanäle).
- Anschluß und transparente Programmierung einer Basiskarte vom Typ ME-SIG32 ist jederzeit möglich, muss jedoch nicht beim ME-iDC angemeldet werden.
- Falls Sie auf die transparente Programmierung verzichten, können Sie im Rahmen des ME-iDS alle Betriebsarten, wie im ME-MultiSig-Handbuch beschrieben, „zu Fuß“ programmieren.

C Subdevice-Caps

C1 „Caps“ in meQuerySubdeviceCaps()

Die Fähigkeiten des abgefragten Subdevices werden im Parameter `<piCaps>` der Funktion `meQuerySubdeviceCaps()` zurückgegeben. Sofern mehrere „Caps“ zutreffen, werden die Werte bitweise ODER-verknüpft. Z. B.: ein Subdevice verfügt über einen digitalen Triggereingang, der wahlweise auf steigende, fallende oder beliebige (d. h. steigende oder fallende) Flanke triggert. Der entsprechende Rückgabewert ist: `0x000E8000`.

Definition	Beschreibung	Hex-Wert
ME_CAPS_NONE	Keine speziellen Caps	0x00000000
Analoge Erfassung		
ME_CAPS_AI_TRIG_SYNCHRONOUS	Analoge Erfassung kann synchron gestartet werden	0x00000001
ME_CAPS_AI_TRIG_SIMULTANEOUS		
ME_CAPS_AI_FIFO	A/D-FIFO vorhanden	0x00000002
ME_CAPS_AI_FIFO_THRESHOLD	Schwelle einstellbar, bei der A/D-FIFO gelesen wird	0x00000004
ME_CAPS_AI_SAMPLE_HOLD	„Sample & Hold“-Stufe vorhanden	0x00000008
ME_CAPS_AI_TRIG_DIGITAL	Digitaler Triggereingang	0x00008000
ME_CAPS_AI_TRIG_ANALOG	Analoger Triggereingang	0x00010000
ME_CAPS_AI_TRIG_EDGE_RISING	Trigger auf steigende Flanke	0x00020000
ME_CAPS_AI_TRIG_EDGE_FALLING	Trigger auf fallende Flanke	0x00040000
ME_CAPS_AI_TRIG_EDGE_ANY	Trigger auf beliebige Flanke	0x00080000
ME_CAPS_AI_DIFFERENTIAL	Differentielle Messung möglich	0x00000010
Analoge Ausgabe		
ME_CAPS_AO_TRIG_SYNCHRONOUS	Analoge Ausgabe kann synchron gestartet werden	0x00000001
ME_CAPS_AO_TRIG_SIMULTANEOUS		
ME_CAPS_AO_FIFO	D/A-FIFO vorhanden	0x00000002
ME_CAPS_AO_FIFO_THRESHOLD	Schwelle einstellbar, bei der D/A-FIFO nachgeladen wird	0x00000004
ME_CAPS_AO_TRIG_DIGITAL	Digitaler Triggereingang	0x00008000

Tabelle 15: „Caps“ in meQuerySubdeviceCaps

Definition	Beschreibung	Hex-Wert
ME_CAPS_AO_TRIG_ANALOG	Analoger Triggereingang	0x00010000
ME_CAPS_AO_TRIG_EDGE_RISING	Trigger auf steigende Flanke	0x00020000
ME_CAPS_AO_TRIG_EDGE_FALLING	Trigger auf fallende Flanke	0x00040000
ME_CAPS_AO_TRIG_EDGE_ANY	Trigger auf beliebige Flanke	0x00080000
ME_CAPS_AO_DIFFERENTIAL	Differentielle Ausgabe möglich	0x00000010
Digitale Ein-/Ausgabe		
ME_CAPS_DIO_DIR_BIT	Richtung je Bit konfigurierbar	0x00000001
ME_CAPS_DIO_DIR_BYTE	Richtung je Byte (8bit Block) konfigurierbar	0x00000002
ME_CAPS_DIO_DIR_WORD	Richtung je Wort (16bit Block) konfigurierbar	0x00000004
ME_CAPS_DIO_DIR_DWORD	Richtung je Langwort (32bit Block) konfigurierbar	0x00000008
ME_CAPS_DIO_SINK_SOURCE	Sink/Source-Umschaltung	0x00000010
ME_CAPS_DIO_BIT_PATTERN_IRQ	IRQ bei Bitmuster-Gleichheit	0x00000020
ME_CAPS_DIO_BIT_MASK_IRQ_EDGE_RISING	IRQ bei steigender Flanke von mindestens einem aktiven Bit	0x00000040
ME_CAPS_DIO_BIT_MASK_IRQ_EDGE_FALLING	IRQ bei fallender Flanke von mindestens einem aktiven Bit	0x00000080
ME_CAPS_DIO_BIT_MASK_IRQ_EDGE_ANY	IRQ bei beliebiger Flanke von mindestens einem aktiven Bit	0x00000100
ME_CAPS_DIO_OVER_TEMP_IRQ	IRQ bei Überhitzung des Treiberbausteins	0x00000200
ME_CAPS_DIO_NORMAL_TEMP_IRQ	IRQ sobald Normaltemperatur des Treiberbausteins wieder erreicht ist	0x00000400
ME_CAPS_DIO_TRIG_SYNCHRONOUS	Digitale Ein-/Ausgabe kann synchron gestartet werden	0x00004000
ME_CAPS_DIO_TRIG_DIGITAL	Digitaler Triggereingang	0x00008000
ME_CAPS_DIO_TRIG_ANALOG	Analoger Triggereingang	0x00010000
ME_CAPS_DIO_TRIG_EDGE_RISING	Trigger auf steigende Flanke	0x00020000
ME_CAPS_DIO_TRIG_EDGE_FALLING	Trigger auf fallende Flanke	0x00040000
ME_CAPS_DIO_TRIG_EDGE_ANY	Trigger auf beliebige Flanke	0x00080000

Tabelle 15: „Caps“ in meQuerySubdeviceCaps

Definition	Beschreibung	Hex-Wert
Frequenz-Ein-/Ausgabe		
ME_CAPS_FIO_SINK_SOURCE	Sink/Source-Umschaltung	0x00000010
Zähler		
ME_CAPS_CTR_CLK_PREVIOUS	CLK kann mit OUT des vorherigen Zählers gespeist werden	0x00000001
ME_CAPS_CTR_CLK_INTERNAL_1MHZ	Zähler kann intern mit 1MHz Takt gespeist werden	0x00000002
ME_CAPS_CTR_CLK_INTERNAL_10MHZ	Zähler kann intern mit 10MHz Takt gespeist werden	0x00000004
ME_CAPS_CTR_CLK_EXTERNAL	Zähler kann über externen Takteingang gespeist werden	0x00000008
Externer Interrupt		
ME_CAPS_EXT_IRQ_EDGE_RISING	Externer IRQ kann mit steigender Flanke auslösen	0x00000001
ME_CAPS_EXT_IRQ_EDGE_FALLING	Externer IRQ kann mit fallender Flanke auslösen	0x00000002
ME_CAPS_EXT_IRQ_EDGE_ANY	Externer IRQ kann mit beliebiger Flanke auslösen	0x00000004

Tabelle 15: „Caps“ in meQuerySubdeviceCaps

C2 „Caps“ in meQuerySubdeviceCapsArgs()

Der Wert der abgefragten „Cap“ eines Subdevices wird im Parameter <piArgs> der Funktion *meQuerySubdeviceCapsArgs()* zurückgegeben.

Definition	Beschreibung	Hex-Wert
Analoge Erfassung		
ME_CAP_AI_FIFO_SIZE	Größe des A/D-FIFOs abfragen	0x001D0000
ME_CAP_AI_BUFFER_SIZE	Größe des vom Treiber zugewiesenen A/D-Pufferspeichers	0x001D0001
ME_CAP_AI_CHANNEL_LIST_SIZE	Größe der A/D-Kanalliste abfragen	0x001D0002
ME_CAP_AI_MAX_THRESHOLD_SIZE	Maximale Anzahl der Werte im A/D-FIFO	0x001D0003
Analoge Ausgabe		
ME_CAP_AO_FIFO_SIZE	Größe des D/A-FIFOs abfragen	0x001F0000
ME_CAP_AO_BUFFER_SIZE	Größe des vom Treiber zugewiesenen D/A-Pufferspeichers	0x001F0001
ME_CAP_AO_CHANNEL_LIST_SIZE	Größe der D/A-Kanalliste abfragen	0x001F0002
ME_CAP_AO_MAX_THRESHOLD_SIZE	Maximale Anzahl der Werte im D/A-FIFO	0x001F0003
Zähler		
ME_CAP_CTR_WIDTH	Bitbreite des Zählers abfragen	0x00200000

Tabelle 16: „Caps“ in meQuerySubdeviceCapsArgs

D Properties

Eine Übersicht der reservierten Schlüsselwörter zum Zugriff auf die Properties finden Sie in der ME-iDS-Hilfedatei.

Hinweis: Installieren Sie ME-iDS 2.0 oder höher zur Verwendung der Properties.

E Fehlercodes

Eine Tabelle mit allen Fehlercodes finden Sie in der ME-iDS-Hilfedatei.

F Index

A

Agilent VEE 23
 Analoge Ein-/Ausgabe 48
 Anhang 221
 API-Funktionen 94
 Architektur des Treibersystems 21
 Ausgabe-Funktionen 135

B

Betriebsarten 45
 Bitmuster-Ausgabe 228
 Interrupt-Betrieb 90
 ME-MultiSig 231
 Pulsweiten-Modulation 226
 Single-Betrieb 45
 Streaming-Betrieb 62
 Zähler 8254 222
 Bibliotheksdateien 22
 Bitmuster-Änderung 90
 Bitmuster-Ausgabe 86
 Bitmuster-Vergleich 90

D

Daten lesen 75
 Daten schreiben 79
 Digitale Ein-/Ausgabe 49
 Digital-I/O
 Port-Mapping 229
 Dokumentation 12

E

Ein-/Ausgabe-Funktionen
 Allgemein
 meIOResetDevice 135
 meIOResetSubdevice 136
 Interrupt
 meIOIrqSetCallback 144
 meIOIrqStart 137
 meIOIrqStop 141
 meIOIrqWait 142

Single-Betrieb

meIOSingle 155
 meIOSingleConfig 149
 meIOSingleTicksToTime 160
 meIOSingleTimeToTicks 162

Streaming-Betrieb

meIOSetChannelOffset 147
 meIOStreamConfig 164
 meIOStreamFrequencyTo-
 Ticks 177
 meIOStreamNewValues 193
 meIOStreamRead 185
 meIOStreamSetCallbacks 194
 meIOStreamStart 179
 meIOStreamStatus 191
 meIOStreamStop 182
 meIOStreamTimeToTicks 175
 meIOStreamWrite 188

Einführung 7

Eingabe-Funktionen 135

F

Fehlerbehandlung 43
 Fehlercodes 238
 Fehlercodes (Tabelle) 233, 236
 FIFO-Umlenkung 86
 Firmware-Konfiguration 16
 Frequenz-Ein-/Ausgabe 50
 Frequenzmessung 51
 Funktionsreferenz 93

H

Hilfs-Funktionen

Fehler-Behandlung

meErrorGetLast 202
 meErrorGetLastMessage 203
 meErrorGetMessage 204
 meErrorSetDefaultProc 205
 meErrorSetUserProc 206

Hardware spezifisch
 meUtilityPWMRestart 219
 meUtilityPWMStart 216
 meUtilityPWMStop 218
Initialisierung
 meClose 198
 meOpen 197
Konvertierung
 meUtilityDigitalToPhysical 207
 meUtilityDigitalToPhysicalV
 211
 meUtilityExtractValues 214
 meUtilityPhysicalToDigital 212
 meUtilityPhysicalToDigitalV
 213
Zugriffsschutz
 meLockDevice 200
 meLockDriver 199
 meLockSubdevice 201
Hochsprachen-Unterstützung 23
I
Impulsgenerator 53
Initialisierung 41
Interrupt-Betrieb 90
 Bitmuster-Änderung 90
 Bitmuster-Vergleich 90
 Externer Interrupt 90
 Interrupt bei Überhitzung 90
K
Kanalliste 63
Konfigurations-Utility 14
L
LabVIEW 23
M
ME-iDC 14
ME-MultiSig-Steuerung 231
MEphisto Scope 230
O
Offset-Einstellung 89

P
Port-Mapping 229
Programmierung 21
 Funktionsbeschreibung 94
 Hochsprachen 23
 ME-MultiSig-System 231
 Zähler 55
Properties 26
 Abkürzungen 28
 Attribute 30
 Baumstruktur 27
 Property-Funktionen 28
 Property-Pfad 27
 Property-Typen 32
 Zugriffstypen 33
Property-Funktionen
 mePropertyGetDouble(A/W) 125
 mePropertyGetInt(A/W) 123
 mePropertyGetString(A/W) 127
 mePropertySetDouble(A/W) 131
 mePropertySetInt(A/W) 129
 mePropertySetString(A/W) 133
Pulsweiten-Modulation 60, 226
Q
Query-Funktionen
 meQueryDescriptionDevice 101
 meQueryInfoDevice 97
 meQueryNameDevice 100
 meQueryNameDeviceDriver 99
 meQueryNumberChannels 108
 meQueryNumberDevices 105
 meQueryNumberRanges 109
 meQueryNumberSubdevices 106
 meQueryRangeByMinMax 120
 meQueryRangeInfo 110
 meQuerySubdeviceByType 118
 meQuerySubdeviceCaps 112
 meQuerySubdeviceCapsArgs 116
 meQuerySubdeviceType 107
 meQueryVersionDeviceDriver 104

meQueryVersionLibrary 102
meQueryVersionMainDriver 103

S

Sample & Hold 86
Service und Support 221
Single-Betrieb 45
Streaming-Betrieb 62
Subdevice-Konfiguration 16
Synchron-Start 88
Systemanforderungen 10

T

Technische Fragen 221
Treiber allgemein 93
Treiberkonzept 21
Treiber-Update 221
Triggeroptionen 46
Triggerstruktur 64

V

Vorgehensweise 41

W

Wraparound-Option 84

Z

Zähler 55
 Pulsweiten-Modulation 226
Zugriffsschutz 41