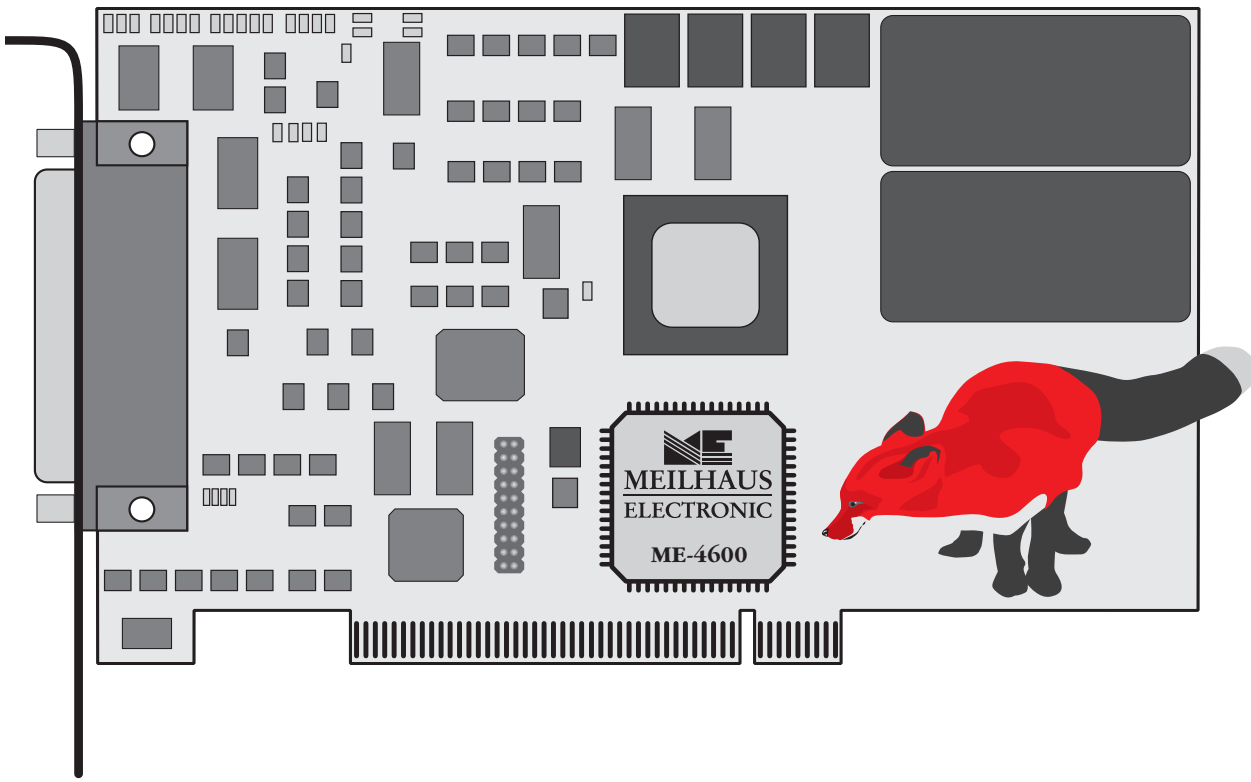
# Meilhaus Electronic Manual

# ME-4600 Series 1.7E

**(ME-4650/4660/4670/4680)**

## The „ME-FoXX®“ Family




**16 Bit Multifunction Board with up to 32 A/D and 4 D/A Channels**
**Optional: Opto Isolation and Sample and Hold Section**

# Imprint

**Manual ME-4650/4660/4670/4680**

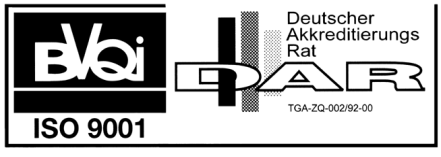
Revision 1.7E
Revised: 22. June 2005

Meilhaus Electronic GmbH
Fischerstraße 2
D-82178 Puchheim/Munich
Germany
http://www.meilhaus.com



**Important note:**
The information contained in this manual has been reviewed with great care and is believed to be complete and accurate. Meilhaus Electronic assumes no responsibility for its use, any infringements of patents or other rights of third parties which may result from use of this manual or the product. Meilhaus Electronic assumes no responsibility for any problems or damage which may result from errors or omissions. Specifications and instructions are subject to change without notice.

Borland Delphi is a trademark of Borland International Inc.
Turbo/Borland C is a trademark of Borland International Inc.
Visual C++ and Visual Basic are trademarks of the Microsoft Corporation.
VEE Pro and VEE OneLab are trademarks of Agilent Technologies.
ME-VEC and ME-FoXX are trademarks of Meilhaus Electronic.
Other company names and product names found in the text of this manual are also trademarks of the companies involved.




# Table of Contents

# 1 Introduction

Valued customer,

Thank you for purchasing a Meilhaus PC plug-in board. You have chosen an innovative high technology product that left our premises in a fully functional and new condition.

Take the time to carefully examine the contents of the package for any loss or damage that may have occurred during shipping. If there are any items missing or if an item is damaged, contact us immediately.

Before you install the board in your computer, read this manual carefully, especially the chapter describing board installation.

## 1.1 Package Contents

We take great care to make sure that the package is complete in every way. We do ask that you take the time to examine the content of the box. Your box should consist of:

- Multifunction board of the ME-4600 series for PCI-bus.
- Manual in PDF format on CD-ROM.
  (optional as printed version)
- Driver software on CD-ROM.
- 78pin D-Sub male connector
- Additional mounting bracket ME-AK-D25F/S
- 25pin D-Sub male connector

# 1.2 Features

| Overview | 16 bit A/D [channels] single-ended/differential | Analog trigger for A/D section | 16 bit D/A [channels] | D/A channels with FIFOs | Digital I/O [channels][1] | Opto-isolation[2] | Sample & Hold [channels][3] | Counters | |
|---|---|---|---|---|---|---|---|---|---|
| **ME-4650** "ME-LittleFoXX" | 16/– | – | – | – | 32 | – | – | – | ME-FoXX Family |
| **ME-4660**(i/s/is)* "ME-RedFoXX" | 16/– | – | 2 | – | 32 | ✔ | 8 | 3 | |
| **ME-4670**(i/s/is)* "ME-SlyFoXX" | 32/16 | ✔ | 4 | – | 32 | ✔ | 8 | 3 | |
| **ME-4680**(i/s/is)* "ME-SylverFoXX" | 32/16 | ✔ | 4 | ✔ | 32 | ✔ | 8 | 3 | |

* Note: not all theoretical variations are available as standard models for immediate delivery (see our website for more details: www.meilhaus.com/me-foxx).

1) Digital ports A and B are available on the standard D-sub 78 female connector, ports C and D are available by an optional mounting bracket with a D-sub 25 female connector.

2) Only for the "i" versions: Optical isolation of the A/D and D/A sections, counters and digital ports A and B (not digital ports C and D)

3) Optionally available with 8 sample and hold channels ("s" versions)

The boards of the **ME-4600 series** are available with 32 A/D channels which can be operated as 32 single ended or 16 differential channels (ME-4650/4660: 16 single ended channels). The input channels are routed through a high impedance input stage to a 16 bit 500 kHz A/D converter. The following input ranges are available: 0…2.5 V, 0…10 V, ±2.5 V and ±10 V.

All models have **32 digital I/O channels** which are organised as 4 bi-directional ports. The opto-isolated version (optional) is fixed with port A as output and port B as input. The ports C and D are not opto-isolated and are available on a flat IDC 20 pin connector. They can also be routed to a D-sub 25 connector on a separate mounting bracket.

With exception of the ME-4650 there are 3 free programmable **16 bit counters** (1 x 8254) available.

The model **ME-4660** has 2 and the model **ME-4670** has 4 high accuracy **16 bit D/A channels**. The output voltage range is ±10 V.

The high end model **ME-4680** has **4 FIFOs** on the D/A channels. The maximum output rate per channel of the D/A output channels is 500 kS/s. In the "AOContinuous" mode values can be reloaded when the output is already running. The "AOWraparound" mode is designed for output of periodic signals.

The **opto-isolated** board versions („i"-versions) allow a complete isolation from the PC ground (exception: ports C and D). This can be very helpful in preventing ground loops and in electrically "noisy" environments.

For simultaneous data acquisition, 8 A/D channels on the „s"-versions are equipped with a **"Sample and Hold"** option.

The software included allows quick and easy integration into all common high level languages under Windows and Linux. Drivers for graphical programming languages like Agilent VEE and LabVIEW™ are also available.

## 1.3 System Requirements

The ME-4600 can be installed into any PC with Intel® Pentium® processor or compatible computers with a free standard PCI slot (32 bit, 33MHz, 5V). The board is supported by all current Windows versions as well as Linux.

## 1.4 Software Support

For the newest versions and latest software releases, please consult the README files.

| | |
|---|---|
| System Drivers | For all common operating systems (see README files) |

ME-Software-Developer-Kit (ME-SDK):

Support for all common programming languages, demos, tools und test programs

Graphical programming tools

Meilhaus VEE Driver System for
HP VEE, HP VEE Lab,
Agilent VEE Pro and
Agilent VEE OneLab

LabVIEW™ Driver

# 2 Installation

Please read your computer manual instructions on how to install new hardware components **before installing the board**. Note the chapter „Hardware Installation“ in this manual (if applicable, e. g. for ISA boards).

- **Installation under Windows (Plug&Play)**

An installation guide describing how to install the driver software can be found in HTML format on CD-ROM. Please read **before installation** and print it on demand!

The following basic procedure should be used:

If you have received the driver software as an archive file please un-pack the software **before installing the board**. First choose a directory on your computer (e. g. C:\Meilhaus).

Then insert the board into your computer and install the driver software. This order of operation is important to guarantee the Plug&Play operation under Windows 95*/98/Me/2000/XP. Windows 95 and NT 4.0 need an analogous order of operation however the installation procedure differs slightly.

**If the Windows version is supported by the appropriate board type (see readme files).*

- **Installation under Linux**

Note the installation instructions included with archive file of the appropriate driver.

## 2.1 Test Program

For simple testing of the board use the appropriate test programs provided with the ME Software Developer Kit (ME-SDK). After un-packing the ME-SDK the test programs can be found in corresponding subdirectories of `C:\MEILHAUS` (Default). Note that the system driver must be installed correctly.

# 3 Hardware

## 3.1 Block Diagram

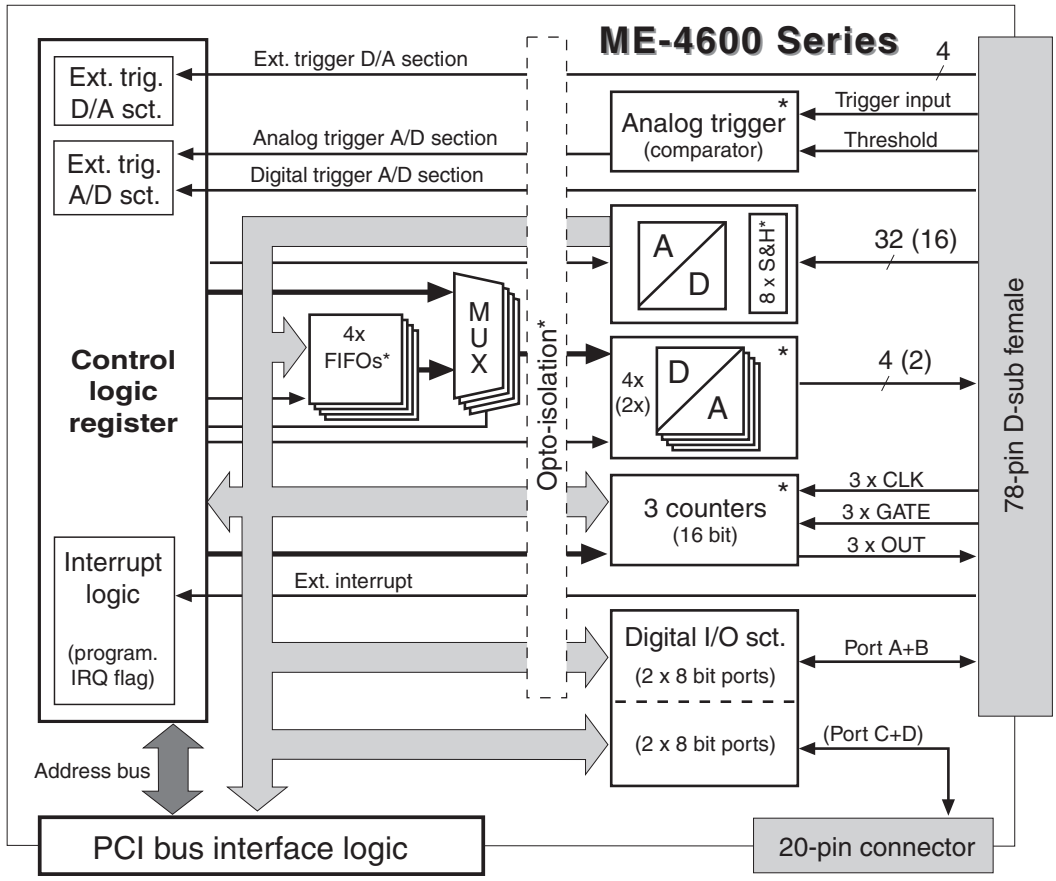



*Diagram 1: Block diagram of ME-4600*

* Depending on the model, not all functional groups shown in the diagram above are available:

**ME-4650**: 16 A/D channels, 32 DIOs

**ME-4660**: 16 A/D channels, 2 D/A channels, 32 DIOs, 3 counters

**ME-4670**: 32 A/D channels, Analog-Trigger, 4 D/A channels, 32 DIOs, 3 counters

**ME-4680**: 32 A/D channels, analog trigger, 4 D/A channels with FIFO, 32 DIOs, 3 counters

**„i"-Option**: with opto-isolation

**„s"-Option**: with 8 sample and hold channels

## 3.2 General Notes

**Attention**: The external connections to the board should only be made or removed in a powered down state.

Ensure that no static discharge occurs when handling the board or when connecting/disconnecting the external cable.

Ensure that the cable is properly connected. It must be seated firmly on the D-Sub connector of the board and must be tightened with the both screws, otherwise proper operation of the board can not be guaranteed!

All unused A/D channels should be grounded. This avoids cross talk between the input lines. We recommend using shielded cables.

The pinout of the 78pin D-Sub connector is shown in Appendix B on page 233.

The following chapters describe the switching and connections of the different functional groups. The operating modes and programming is described in chapter 4 (page 31 and following).

## 3.3 A/D Section

With exception of the models ME-4650 and ME-4660 (16 single ended channels) all models of the ME-4600 series have 32 single ended resp. 16 differential input channels. All channels are isolated with a high impedance input stage:

- Input impedance: $R_{IN}$ = typ. 600MΩ, $C_{IN}$ = typ. 3pF

Boards with sample & hold option (see also chapter 3.3.3) the input impedance of the first 8 channels (AD_0…7) is:

- $R_{IN}$ = typ. 1MΩ, $C_{IN}$ = typ. 5pF (valid independently whether the sample & hold option is used or not).

**The input voltage applied to the analog inputs must not exceed ±15V.**

The user can choose between the unipolar input voltage ranges 0…(2.5V-1LSB) and 0…(10V-1LSB), and the bipolar input voltage ranges -2.5V…(+2,5V-1LSB) and -10V…(+10V-1LSB).

The following (ideal) characteristic curves are valid:

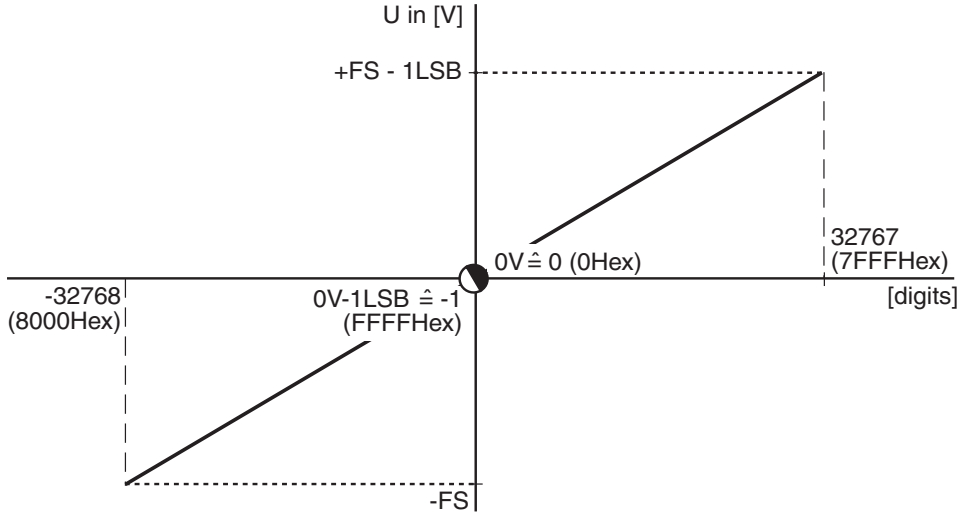



*Diagram 2: Characteristic of bipolar input ranges*

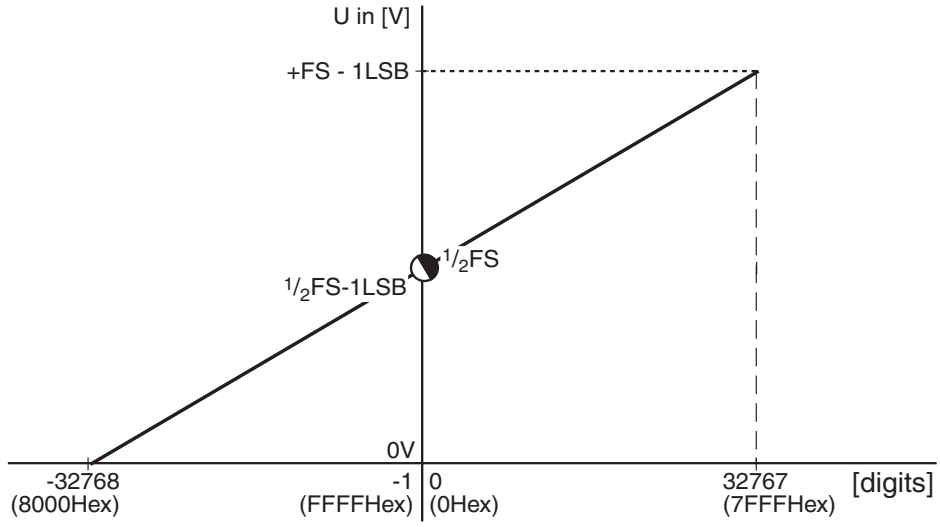



*Diagram 3: Characteristic of unipolar input ranges*

(„FS“ means „Full Scale“ in the appropriate measuring range; „LSB“ means „Least Significant Bit“ of the 16 bit A/D conversion).

**Note:** The theoretical value for full-scale in the appropriate measuring range is only met approximately as a rule (see also specifications on page 227).

**Timer controlled A/D conversion** can be achieved using the 32 bit chan and 36 bit scan timers. The configuration of the A/D section in “AIContinuous” and “AIScan” mode is done using the function …*AIConfig*. The input voltage range for the appropriate channel will be written to a so called channel list. Use the function …*AIMakeChannelListEntry* for creating the channel list

entries (max. 1024 entries). The conversion process can be started (depending on the mode of operation) by software or by one of the many external triggering options.

### 3.3.1 Single Ended Operation

For single ended operation the 32 input channels (ME-4650/ ME-4660: 16 channels) are available in all input ranges. The measurement signal is connected to the desired input channel AD_*x*. Each input channel requires a low resistance connection to the ground of the A/D section (A_GND). It is important that all minus (-) lines have the same potential to avoid cross currents (and therefore measurement errors).

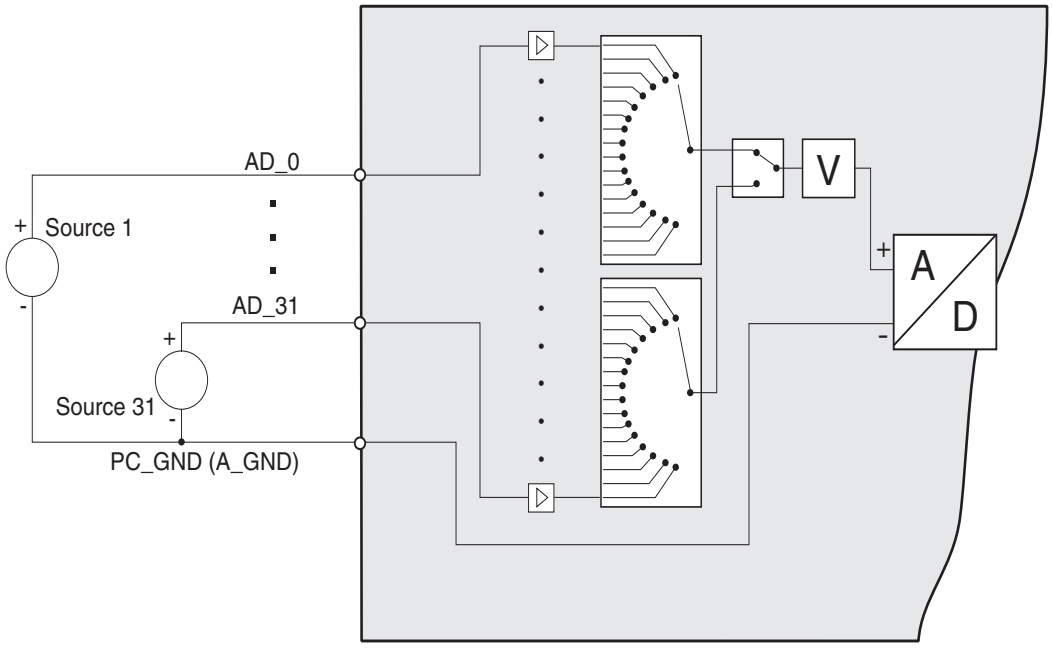



*Diagram 4: Switching in single ended operation*

### 3.3.2 Differential Operation

The advantage of differential operation is that common mode noise is greatly suppressed. You can use up to 16 differential channels operating in the bipolar ranges (±2.5V and ±10V). Each input channel requires a positive and negative input line.

**Note:** the ME-4650/4660 provide only single ended operation.

The following table shows how the differential pin connections are organised:

| Pos. Signal | | Neg. Signal | | Pos. Signal | | Neg. Signal | |
|---|---|---|---|---|---|---|---|
| **Channel** | **Pin** | **Channel** | **Pin** | **Channel** | **Pin** | **Channel** | **Pin** |
| AD_0 | 39 | AD_16 | 15 | AD_8 | 78 | AD_24 | 54 |
| AD_1 | 19 | AD_17 | 34 | AD_9 | 58 | AD_25 | 73 |
| AD_2 | 38 | AD_18 | 14 | AD_10 | 77 | AD_26 | 53 |
| AD_3 | 18 | AD_19 | 33 | AD_11 | 57 | AD_27 | 72 |
| AD_4 | 37 | AD_20 | 13 | AD_12 | 76 | AD_28 | 52 |
| AD_5 | 17 | AD_21 | 32 | AD_13 | 56 | AD_29 | 71 |
| AD_6 | 36 | AD_22 | 12 | AD_14 | 75 | AD_30 | 51 |
| AD_7 | 16 | AD_23 | 31 | AD_15 | 55 | AD_31 | 70 |

*Table 1: Channel assignment in differential operation*

Please note, that also in differential mode, a connection to the analog ground is required. This is achieved by connecting the negative (-) input to the ground of the A/D section (A_GND) via a resistor (approx. 100 kΩ).

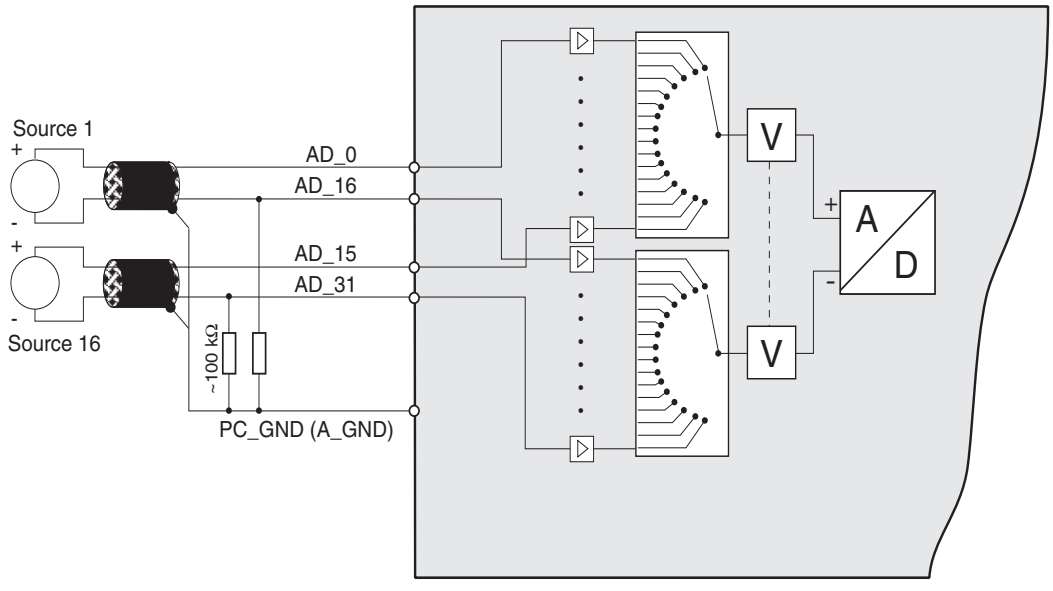



*Diagram 5: Switching in differential operation*

### 3.3.3 Simultaneous Operation

Boards with the Sample and Hold option („s“-versions) allow a simultaneous data acquisition using the first 8 channels to be controlled by software. The input impedance of the sample and hold channels is as follows: $R_{IN}$ = typ. 1MΩ, $C_{IN}$ = typ. 5pF. Valid independently whether the sample & hold option is used or not.

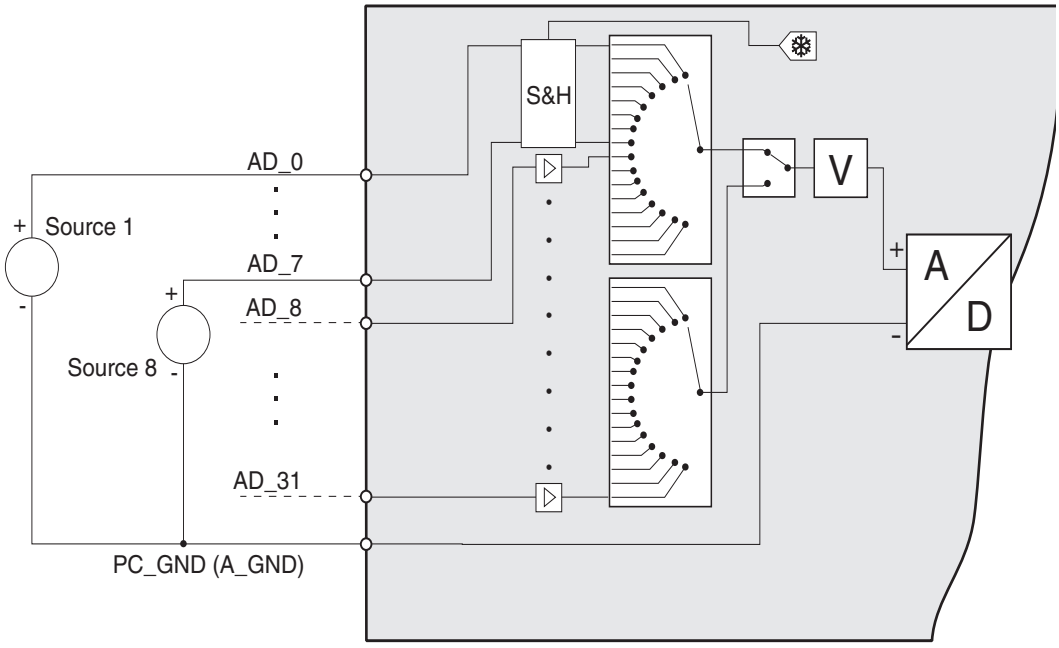



*Diagram 6: Switching in simultaneous operation*

The signals applied to the channels AD_0…7 are “frozen” when a signal from the state machine occurs. The “frozen” values are read in sequentially. The following points should be noted:

- Only single ended mode is possible for the simultaneous operation (for all channel list entries).
- Each sample and hold channel can only be sampled once per channel list processing. This means that channels 0…7 can only appear once in the channel list.
- Useful values for the number of channel list entries are 2…8.
- It is recommended to use the maximum sample rate (2µs) always be used for simultaneous operation, otherwise the „frozen“ voltage value will “melt” at a rate of typ. 0.08µV/µS.

- The minimum time between 2 simultaneous measurements depends on the number of channels being sampled and the recovery time. This must be considered when calculating the SCAN time. The minimum SCAN time can be calculated as follows:

  Min. SCAN time = (Number of channel list entries x CHAN time) + recovery time

**Note** that after the channel list processing is done, a recovery time of at least 1.5 µs is required.

In the following example 4 channels are to be sampled simultaneously. The values should be read as quickly as possible, i. e. the minimum CHAN time should be used (2 µs). The following calculation is valid:

min. SCAN time = (4 x 2 µs) + 1.5 µs = 9.5 µs

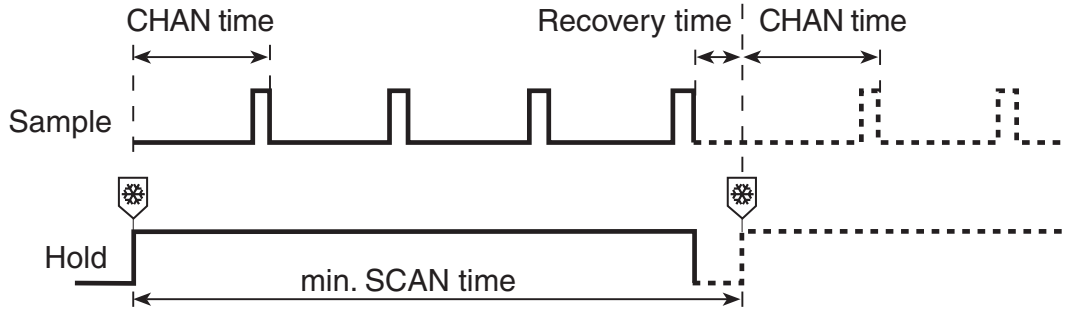



*Diagram 7: Sample and Hold timing*

## 3.3.4 External Trigger A/D Section

All ME-4600 models have a digital A/D trigger input. The models ME-4670 and ME-4680 have an additional analog trigger unit. Depending on the selected option (RISING, FALLING or BOTH) the A/D conversion will be started on the matching edge.


*Diagram 8: Trigger edges*

### 3.3.4.1 Analog Trigger A/D Section

The analog A/D trigger unit uses a comparator to compare the voltage levels on the AD_TRIG_A+ (pin 50) and the AD_TRIG_A- (pin 69) inputs.

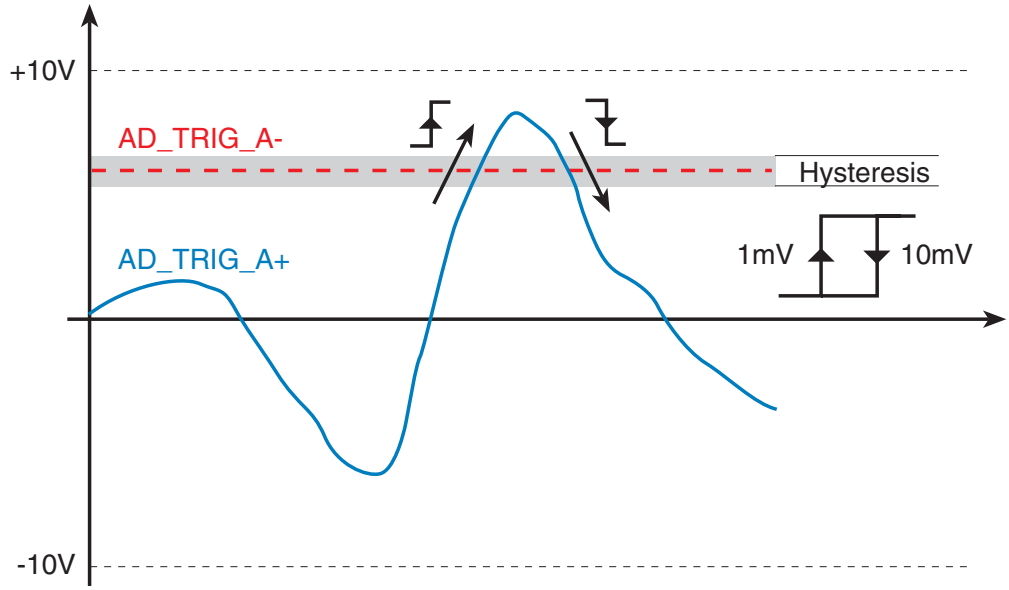



*Diagram 9: Analog Trigger*

We recommend applying a „Threshold voltage“ level on the minus input. This can be done using one of the D/A channels or an external voltage source. The voltage value which serves as the trigger is applied to the positive input. This could be an A/D channel which is connected to the positive input for example (see also diagram 10). When the voltage level on the positive input becomes higher (more positive) than the threshold voltage on the minus input a rising edge is met. The inverse direction means a falling edge.

Dynamic signals of up to 500 kHz at ±10V can be applied. The ground reference of the trigger inputs must be considered. For boards without optical isolation, this is the PC ground (PC_GND). For optically isolated boards, the analog trigger requires a reference to the analog ground (A_GND).

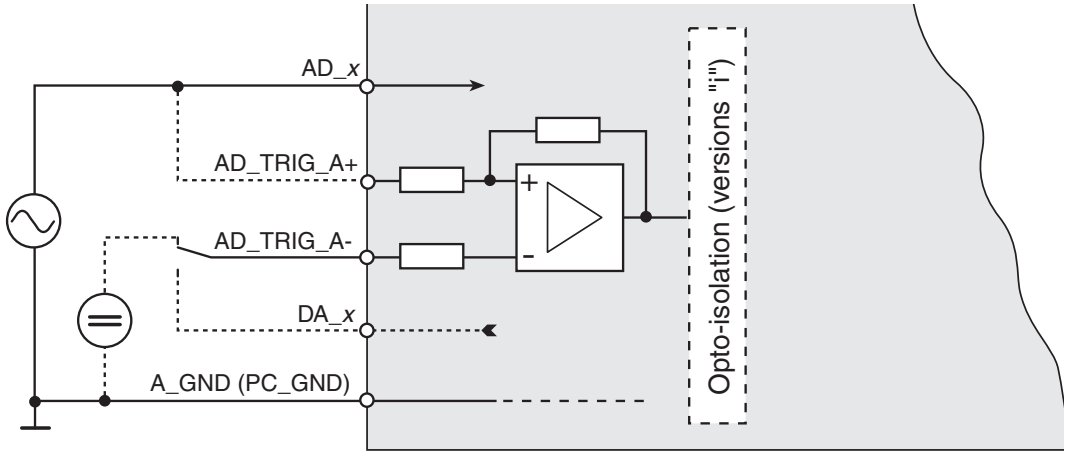



*Diagram 10: Switching analog trigger*

### 3.3.4.2 Digital Trigger A/D Section

The digital trigger input (AD_TRIG_D) requires a high level of +5V. Board versions with optical isolation require a minimum $I_F$ of 7.5 mA to be supplied. The trigger signal requires a reference to ground (PC_GND or DIO_GND).

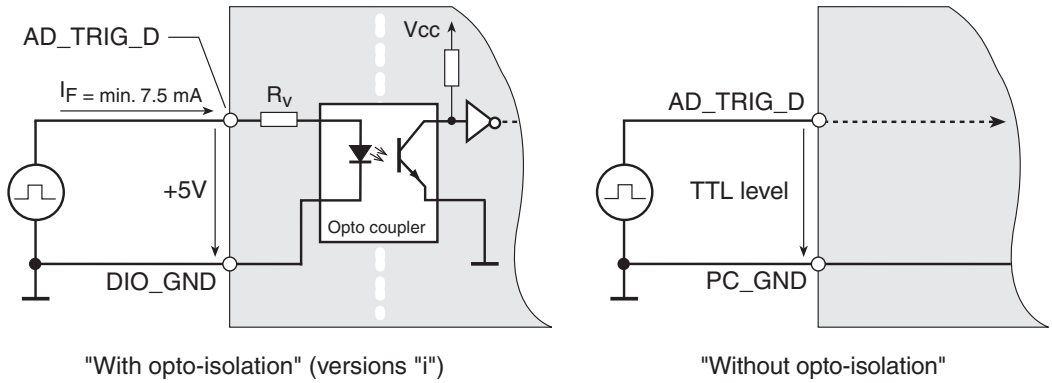



*Diagram 11: Switching digital trigger*

# 3.4 D/A Section

The ME-4660 provides 2 and the ME-4670 and ME-4680 provides 4 analog output channels. Each channel has its own serial 16 bit D/A converter and converts up to 500 kS/s. The output voltage range for each channel is -10V…+10V-1LSB.

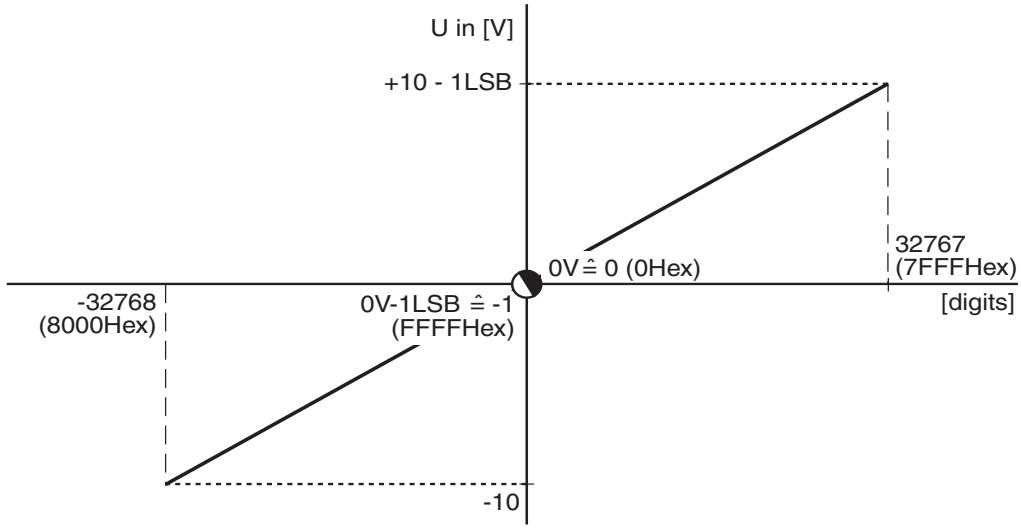



*Diagram 12: Characteristic of the D/A converter*

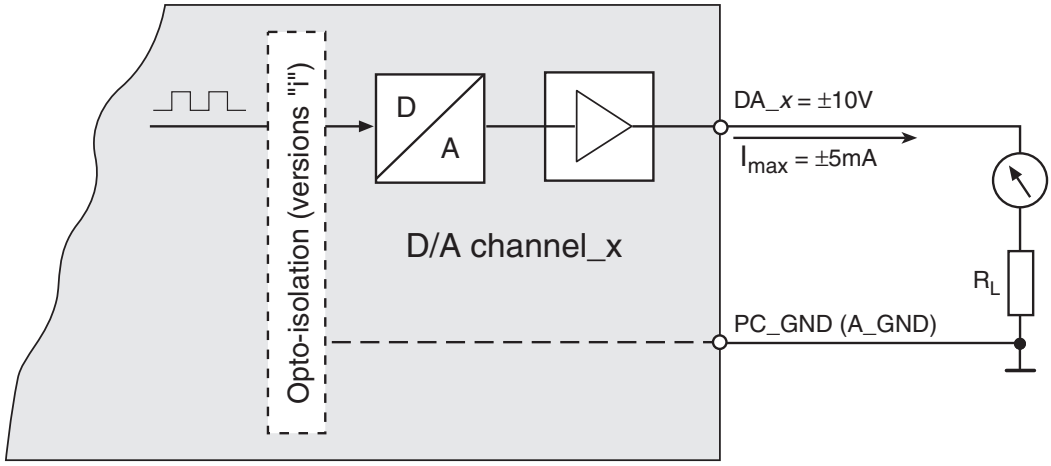



*Diagram 13: Switching of the analog outputs*

**Note:** $I_{max}$ = **±5 mA** per channel must not be exceeded!

The optically isolated models (“i” versions) have D/A channels which are isolated from the PC ground and are all referenced to the analog ground (A_GND).

**Attention**:
After power up the D/A channels outputs are set to -10V. After starting the driver, the outputs are set to 0V. To guarantee a defined power up condition please start your host computer first and do not power up your external switching until the driver started.

## 3.4.1 External Trigger D/A Section

Each D/A channel has its own external trigger input (DA_TRIG_*x*). Depending on the selected option (RISING, FALLING or BOTH) the conversion will be started on the matching edge.


*Diagram 14: Trigger edges*

It is important that the voltage levels of the external trigger input switching be within the specified limits (see specifications on page 227) and that a reference to PC ground (PC_GND) resp. digital ground (DIO_GND) for the “i” board versions be made. The resistor $R_V$ of the opto-isolated trigger inputs is set for a high level of +5V at $I_F$ = 7.5 mA. For not opto-isolated inputs TTL level is valid.

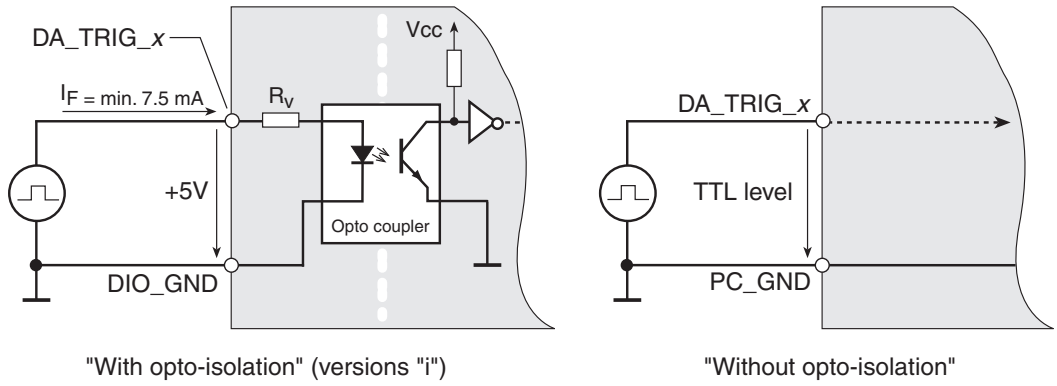



*Diagram 15: Switching of the D/A trigger inputs*

# 3.5 Digital I/O Section

The ME-4600 series boards have four digital ports with 8 bits each. As long as the board is not optically isolated, each port can be configured independently as input or output. Models with optical isolation (“i” versions) are fixed with port A as output and port B as input.

Port C and D are available on the 20pin flat connector ST2 and can be routed to an external mounting bracket (ME-AK-D25F/S) with a D-sub 25 female connector. Port C and D are not isolated, even on the “i” versions. Optionally, the external adapter ME-AA4-3i can be used for optical isolation of ports C and D.

The port direction is set by the software. After power up, all ports are set to input with the exception of port A (output) on the optically isolated models (“i” versions).

For more information about programming the digital I/O section, refer to chapter 4.3 "Digital I/O Section".

## 3.5.1 Digital Inputs

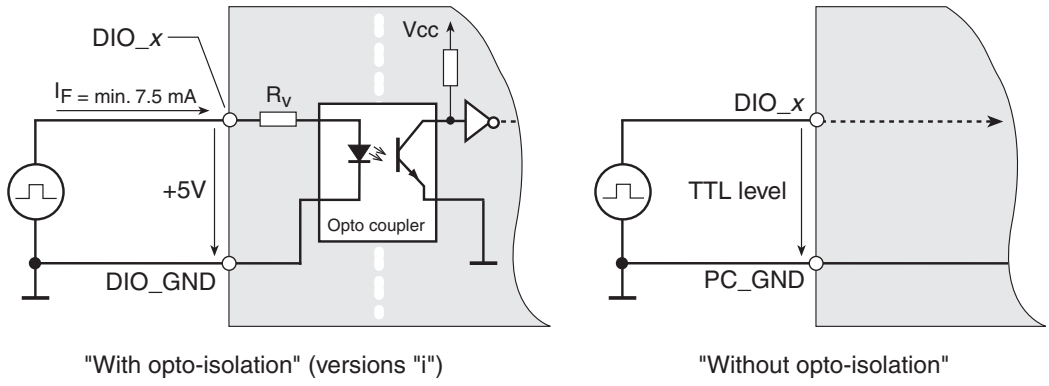



*Diagram 16: Switching of the digital inputs*

It is important that the voltage levels of the digital input switching be within the specified limits (see specifications on page 227) and that a reference to PC ground (PC_GND) resp. digital ground (DIO_GND) for the “i” board versions be made. The resistor $R_V$ of the opto-isolated inputs is set for a high level of +5V at $I_F$ = 7.5 mA. For not opto-isolated inputs TTL level is valid.

## 3.5.2 Digital Outputs

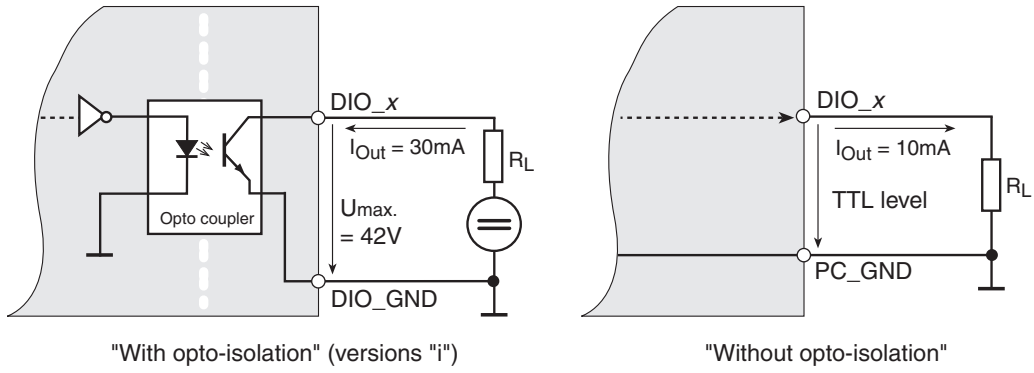



*Diagram 17: Switching of the digital outputs*

It is important that the voltage levels of the digital output switching be within the specified limits (see specifications on page 227) and that a reference to PC ground (PC_GND) resp. digital ground (DIO_GND) for the “i” versions be made. The opto-isolated board versions allow switching of signals up to $U_{max}$ = 42V. The maximum output current for TTL versions is $I_{Out}$ = $I_{OL}$ = $I_{OH}$ = 10mA. The opto-isolated versions provide an $I_{Out}$ of maximum 30mA.

# 3.6 Counter

## 3.6.1 Counter Chip

The **ME-4600 series** boards (not ME-4650) use the standard counter chip of type 82C54. This flexible component has 3 independent 16 bit down counters. All counter signals are available on the external D-sub connector. After the GATE signal has been properly set (TTL: 5V, Opto version: 0V) the counter counts down on every falling edge. The clock (CLK) sourcing the counter must be supplied externally and can have a maximum frequency of 10MHz. The counters can be cascaded by making the proper external connections.

The counter signals of the non optically isolated boards work with TTL level (see Appendix A "Specifications") and require a reference to ground (PC_GND). The maximum output current for low level is $I_{OL}$ = 7.8mA and for high level $I_{OH}$ = 6mA.

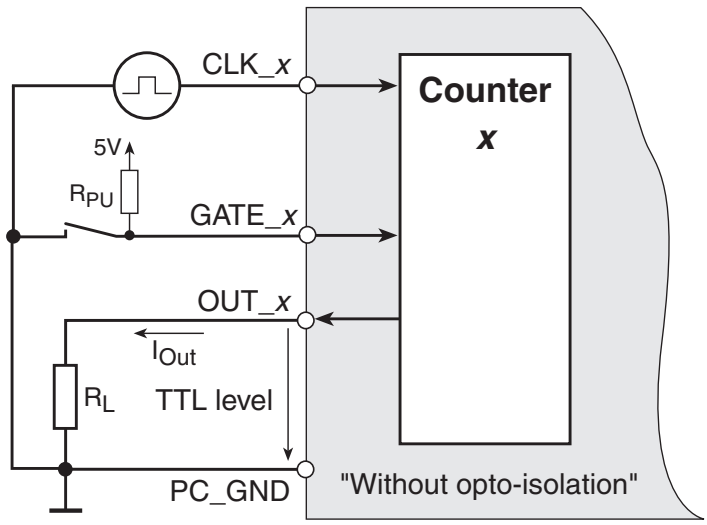
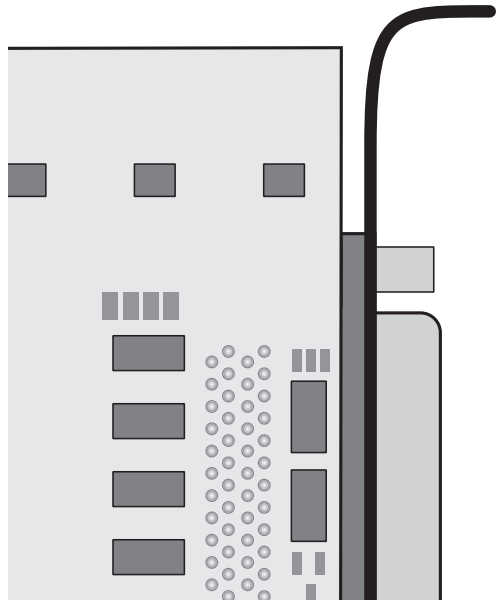



*Diagram 18: Switching of counters without opto-isolation*



An external power supply (+5V, 30mA) connected to pin 1 (CNT_VCC_IN) can be used to supply the opto couplers, or the internal supply (A_VCC) of the analog section. The jumpers J1 and J2 must not be bridged (default) if an external supply is being used. If the internal supply is used the jumpers J1 and J2 must be bridged. Note: the isolation between the analog ground (A_GND) and the counter ground (CNT_GND) is removed if the jumpers are bridged.

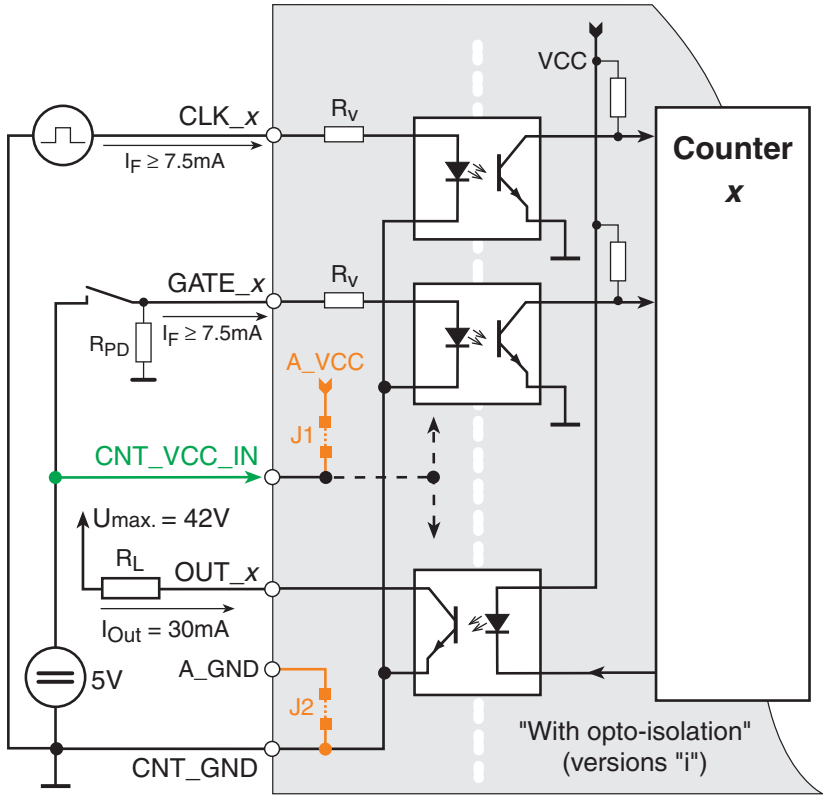



*Diagram 19: Switching of the counters with opto-isolation*

Note, that the polarity of the input signals (CLK_*x* and GATE_*x*) on the optically isolated board versions is inverted by the opto coupler circuitry. All counter signals require a reference to the counter ground (CNT_GND). The voltage $U_{max}$ must not exceed

42V! The maximum output current $I_{Out}$ of the optically isolated versions must not exceed 30mA.

For more information about programming the counters see chapter 4.4.

### 3.6.2 Pulse Width Modulation

With proper external connections the counters 0…2 can be used together to create an output signal with a variable duty cycle. The duty cycle can be set between 1…99% in 1% increments. The prescaler must be sourced by an external base clock of maximum 10MHz. This results in an output signal of maximum 50kHz. By using the connections shown in the diagram below, the functions *me4000CntPWMStart/Stop* can be used which greatly simplify the programming (see page 78 and 174).

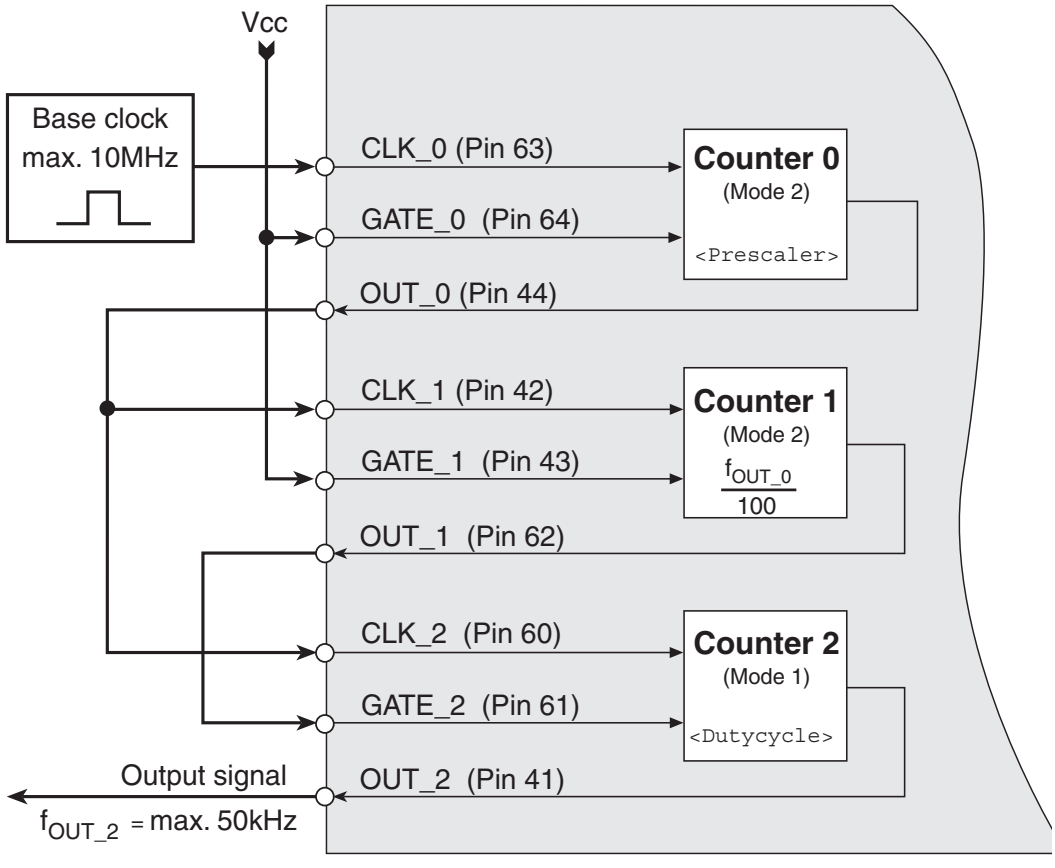



*Diagram 20: Switching for pulse width modulation*

Use the following formula to calculate the frequency $f_{OUT\_2}$:

$$f_{OUT\_2} = \frac{\text{Base clock}}{\texttt{<Prescaler>} \cdot 100} \qquad (\text{with } \texttt{<Prescaler>} = 2\ldots(2^{16}-1))$$

**Tip:** The external switching can be done by the optional connection adapter ME-AA4-3, which also provides a 10MHz crystal oscillator.

## 3.7 External Interrupt

Interrupts can be generated by applying a positive edge to the external interrupt input (EXT_IRQ, pin 48). This interrupt is sent directly to the PCI bus. The external interrupt must first be enabled by calling the function *me4000ExtIrqEnable*.

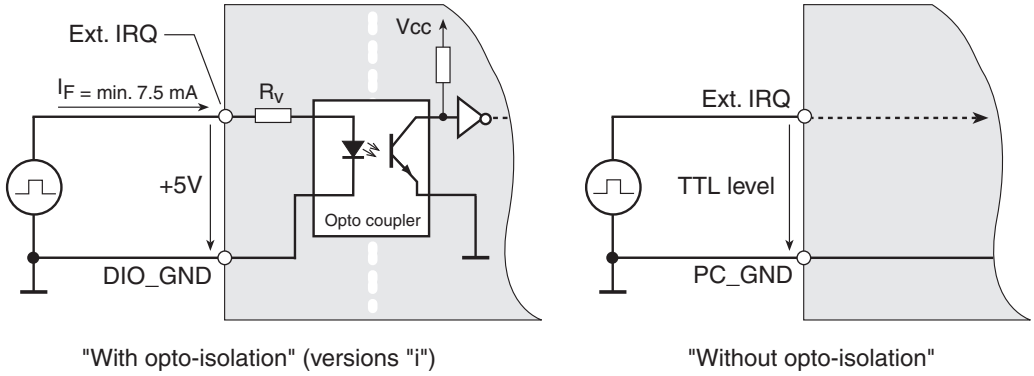



*Diagram 21: Switching of external interrupt input*

The external signal must be within the specified limits (see appendix A, specification) and a reference to the PC ground (versions without optical isolation) or the digital ground (versions with optical isolation) must be made.

# 4 Programming

## 4.1 A/D Section

| Operation Mode | Usage | Trigger | Timing |
|---|---|---|---|
| AISingle (see page 31) | 1 channel, 1 measurement value | Software-Start, ext. digital, analog (opt.) | – |
| AIContinuous (see page 34) | Acquisition of an unknown number of measurement values corresponding to the channel list | Software-Start, ext. digital, analog (opt.) | CHAN Timer, SCAN Timer (…AIConfig) |
| AIScan (see page 34) | Acquisition of a known number of measurement values corresponding to the channel list | Software-Start, ext. digital, analog (opt.) | CHAN Timer, SCAN Timer (…AIConfig) |
| AISimultaneous (see page 32) | Optional for AIScan and AIContinuous: Simultaneous acquisition of multiple channels | | |

*Table 2: A/D operation modes*

### 4.1.1 Operation Modes „AISingle“

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This mode of operation serves to acquire a single value from the selected channel. The following parameters are available:

- Channel number 0…31 (ME-4650/4660: 0…15)
- Input voltage range: 0…2.5V; 0…10V; ±2.5V; ±10V (note: for differential mode, only the bipolar ranges can be used).
- Operation mode can be single ended or differential (ME-4650/4660: single ended only).
- Trigger modes: per software, external digital trigger, or external analog trigger (ME-4670/4680 only)
- External trigger: on falling edge, rising edge, or both.
- Time-Out: in case the external trigger signal does not occur.

No channel list is required in „AISingle“ operation mode.

The following diagram shows the program order of operation:

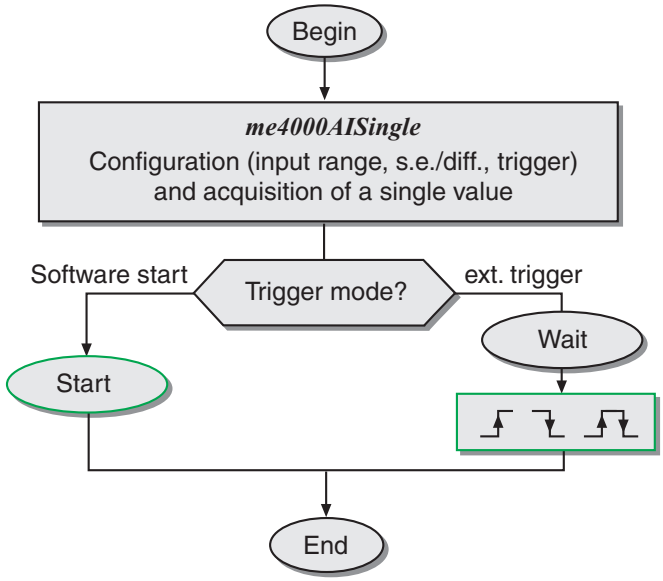



*Diagram 22: Operation mode „AISingle“*

For more information, refer to the sample programs in the ME-SDK and the function description on page 126.

## 4.1.2 Operation Mode „AISimultaneous“

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | nur mit „s“-Option | | |

For boards with the sample and hold option (“s” models) channels 0…7 can be sampled simultaneously (see also chapter 3.3.3).

After a signal from the state machine, the signals on channels AD_0…7 are „frozen“ and then read in sequentially corresponding to the channel list. Note the following items:

- Only single ended mode is possible for the simultaneous operation (for all channel list entries)!
- Each sample and hold channel can only be sampled once per channel list processing. This means that channels 0…7 can only appear once in the channel list.
- Useful number of channel list entries in sample and hold operation is 2…8.

- It is recommended that the maximum sample rate (500 kS/s) always be used for simultaneous operation otherwise the voltage measured will "melt" at a rate of approx. 0.08 µV/µs (max. 0.5 µV/µs).
- The minimum time between 2 simultaneous measurements depends on the number of channels being sampled and the recovery time. This must be considered when calculating the SCAN time.
- The minimum SCAN time can be calculated as follows:

  min. SCAN time = (number of channel list entries x CHAN time) + recovery time

- A minimum recovery time of 1.5 µs is required.

The following example shows how 4 channels are acquired simultaneously. The values are read in as quickly as possible, therefore the CHAN time should be minimal (2 µs). This is shown in the following calculation:

min. SCAN time = (4 x 2 µs) + 1.5 µs = 9.5 µs

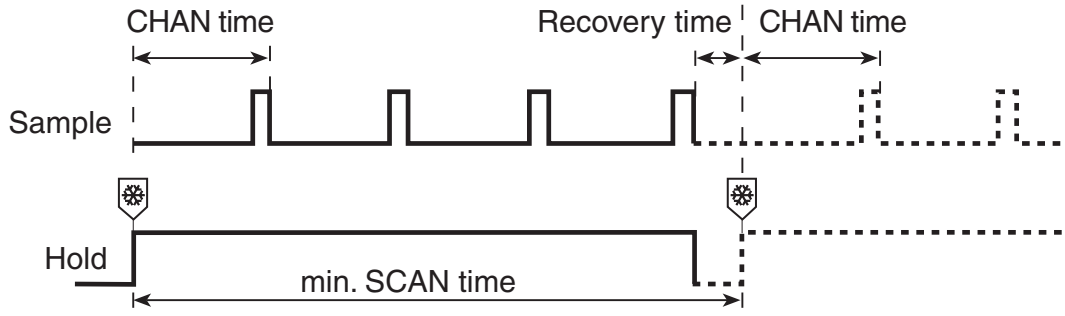



*Diagram 23: Sample & Hold Timing*

### 4.1.3 Timer Controlled „AI Operation Modes“

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

Timer controlled data acquisition requires the following 3 general steps:

1. Create a user defined channel list using the function …*AIMakeChannelListEntry* and configuration of the A/D section with the function …*AIConfig* (see chapter 4.1.3.1)
2. Software preparation with the function …*AIContinuous* or …*AIScan* (see chapter 4.1.3.2)
3. Start the acquisition with …*AIStart* (see chapter 4.1.3.3)

Before the data acquisition is started, it must be decided whether a known number of values (“AIScan”) or continuous sampling („AIContinuous“) is required. This decision determines how the process will be stopped. After configuration of hardware and software the acquisition can be started by software or an external trigger signal.

Diagram 24 shows the general process for the **„AIContinuous“** and **„AIScan“** operation. For more information about configuration, data handling, and execution modes, see the following chapters.

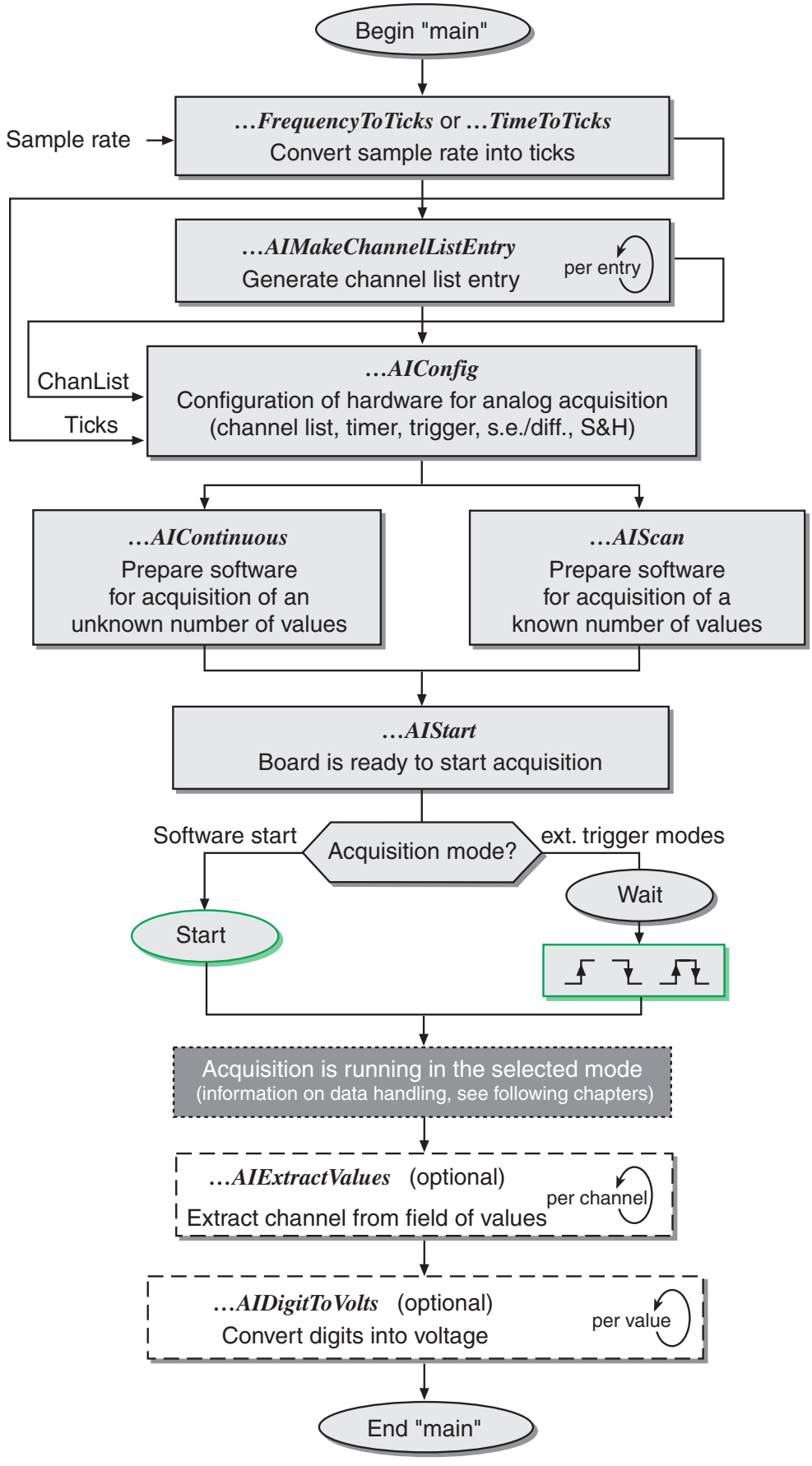



*Diagram 24: Programming of AI section*

#### 4.1.3.1 Configuration of the A/D Section

Before the actual configuration is done, a user defined channel list must be created. This channel list controls the channel numbers and input ranges for the A/D data acquisition. It can have a maximum of 1024 entries. A value field of defined size must be allocated where the single entries will be written to by calling the function *…AIMakeChannelListEntry* repeatedly. The following parameters are required (for a description of the function see page 121):

- Channel number 0…31 (ME-4650/4660: 0…15)
- Input voltage range: 0…2.5V; 0…10V; ±2.5V; ±10V (note: for differential mode, only the bipolar ranges can be used).

After the channel list has been created, the hardware is configured with the function *…AIConfig*. The following parameters are required (for a description of the function see page 110):

- Operation modes single ended or differential are possible (ME-4650/4660: single ended only)
- If wanted: simultaneous acquisition of channels 0…7 (only for boards with sample and hold option, see also chapter 3.3.3)
- Acquisition Modes
  (see also chapter 4.1.4 "External Trigger A/D Section"):

  - ***Acquisition Mode „Software Start***

    The data acquisition is started immediately after calling the function *me4000AIStart*. The processing is done according to the timer settings.

  - ***Acquisition Mode „External Standard“***

    The data acquisition is ready to start after calling the function *…AIStart*. The acquisition starts with the first external trigger pulse and the processing is done according to the timer settings. Any trigger pulses after the first one have no effect on the data acquisition.

- ***Acquisition Mode „External Single Value“***

  The data acquisition is ready to start after calling the function *…AIStart*. Exactly **one value** is converted on each trigger pulse according to the channel list. When the first trigger pulse appears, a single time delay of one CHAN time occurs before the first A/D conversion is done.

- ***Acquisition Mode „External Channel List“***

  The data acquisition is ready to start after calling the function *…AIStart*. The channel list is processed once on each trigger pulse according to the timer settings. The SCAN timer has no effect here!

- External trigger modes: analog or digital trigger source (FALLING, RISING or BOTH edges).

- Two programmable counters serve as timers. There is a 32 bit CHAN timer and a 36 bit SCAN timer. A 33MHz time base is used by both timers. This allows a period time of 30.$\overline{30}$ns, this is the smallest possible time unit and is defined as “one tick” from now on. For convenient conversion, the functions *…FrequencyToTicks* and *…TimeToTicks* can be used.

  - The CHAN timer determines the sample rate within the channel list (time between two consecutive conversions within the channel list). CHAN times between 2 µs…130 s can be set.

  - The SCAN timer determines the time between two consecutive channel list processings. Use of this timer is optional. SCAN times of up to 30 minutes are possible. The SCAN time can be calculated as follows:

    (number of channel list entries x CHAN time) + „pause“

    The “pause” and therefore the SCAN time, can be set in increments of 30.3 ns (1 tick). The pause time must be at least 1 tick.

The following diagram shows the A/D timing for a typical data acquisition using the …*AIScan* operation:

- Number of channel list processings = 2
- Number of channel list entries = 3 (max. 1024 entries)

⇒ Number of measured values = 6

The start is done by software:
(Acquisition mode: ME4000_AI_ACQ_MODE_SOFTWARE).

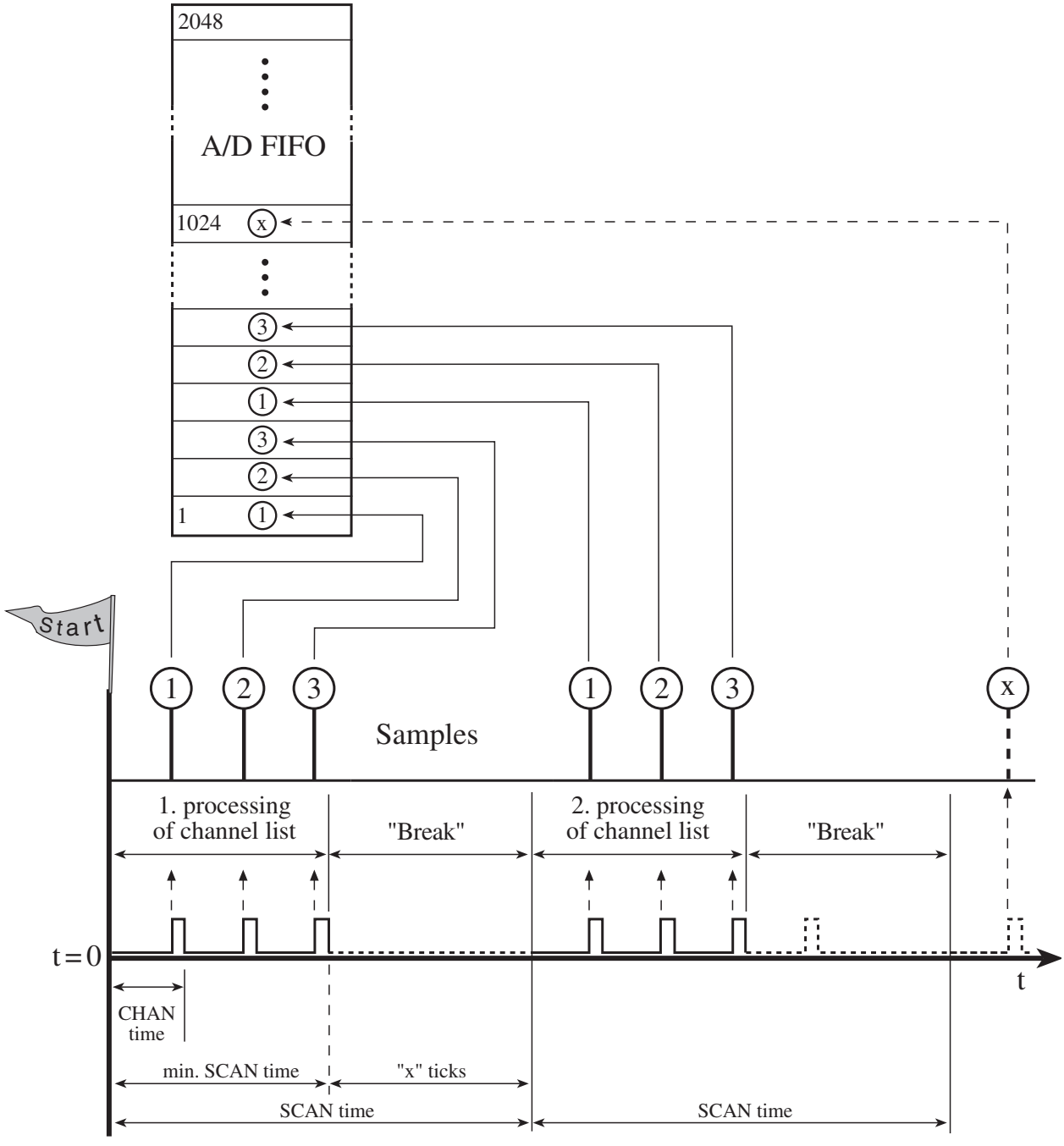



*Diagram 25: A/D Timing*

#### 4.1.3.2 Software Preparation

The next step is to prepare the software for data acquisition. Different options and functions are available depending on whether a known number of samples (see chapter 4.1.3.2.2) or a continuous stream of samples (see chapter 4.1.3.2.1) is required. See the following chapters for a detailed description.

Internally the driver uses a ring buffer to store the measured values. If required, the user can influence the interrupt controlled writing of the measured values to the ring buffer. A “control value” can be passed in the parameter `<RefreshFrequency>`. This value defines the number of channel list processings done before the A/D FIFO is read out. If no value is given, the driver uses an appropriate value by default.

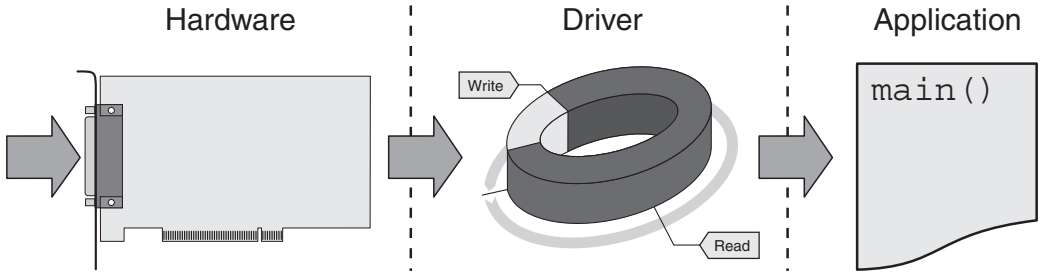



*Diagram 26: Ring buffer AI operation*

#### 4.1.3.2.1 Operation Mode „AIContinuous“

The …*AIContinuous* function serves for continuous acquisition of an unknown number of measured values. The measured values can be read in either by a user defined callback function or repeated calling of the function …*AIGetNewValues*. The data acquisition is started as a background process in this mode. This results in a new thread being created automatically when the function …*AIStart* is called. Parallel to this, other operations (threads) can still be processed.

The diagrams on the following pages show the program flow with the following conditions applied:

a. The data retrieval runs in a background operation (asynchronous) with a user defined callback function.

b. The data retrieval runs in a background operation (asynchronous) using the function …*AIGetNewValues*.

For more information, refer to the sample programs found in the ME-SDK and the function descriptions on page 113.

**Note to a:** Data handling in the **„AIContinuous“** operation mode using a user defined callback function:

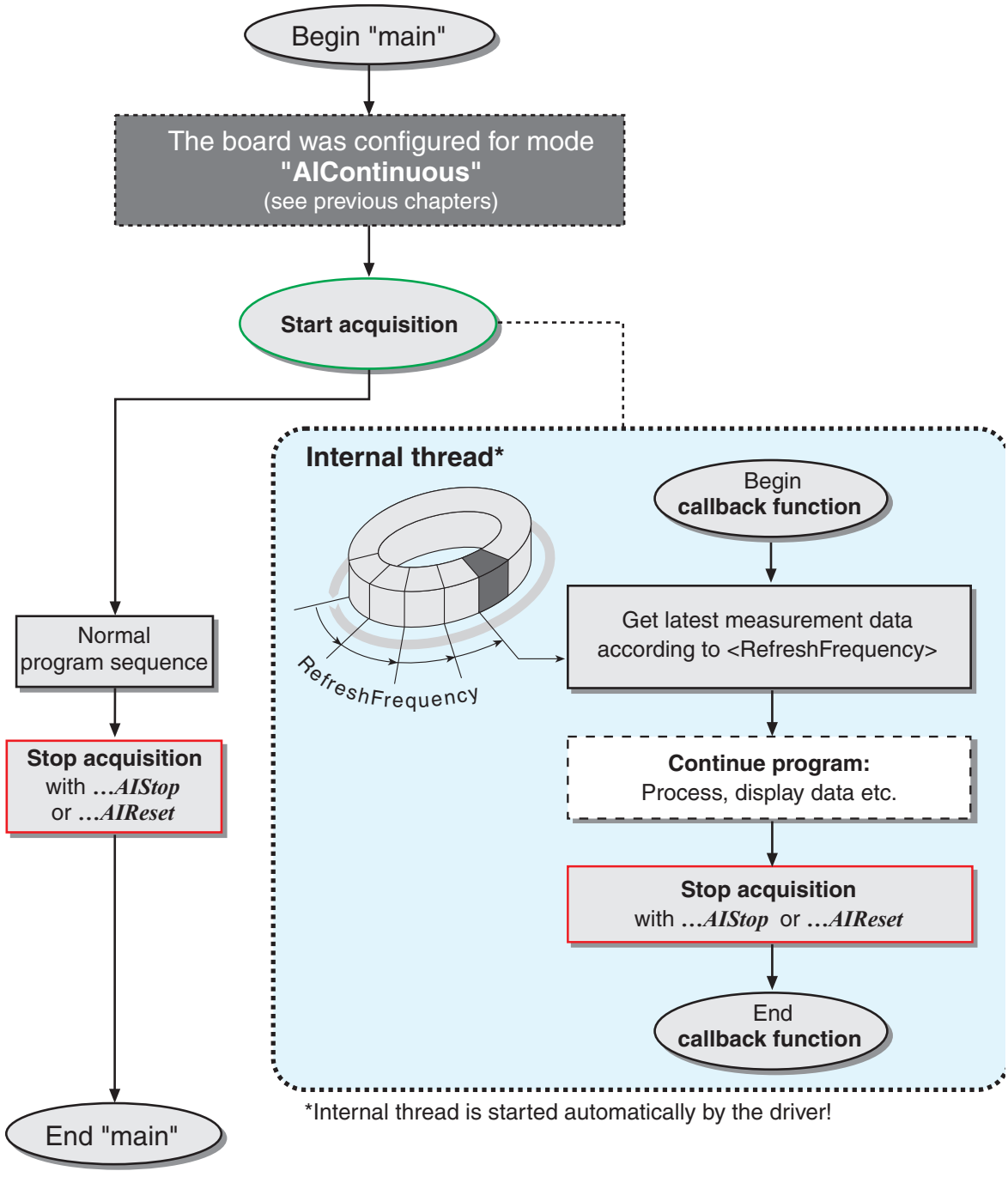



*Diagram 27: Programming „AIContinuous“ with a callback function*

**Note to b:** Data handling in the **„AIContinuous“** operation mode using repeated calls of the function *…AIGetNewValues*. (BLOCKING or NON_BLOCKING):

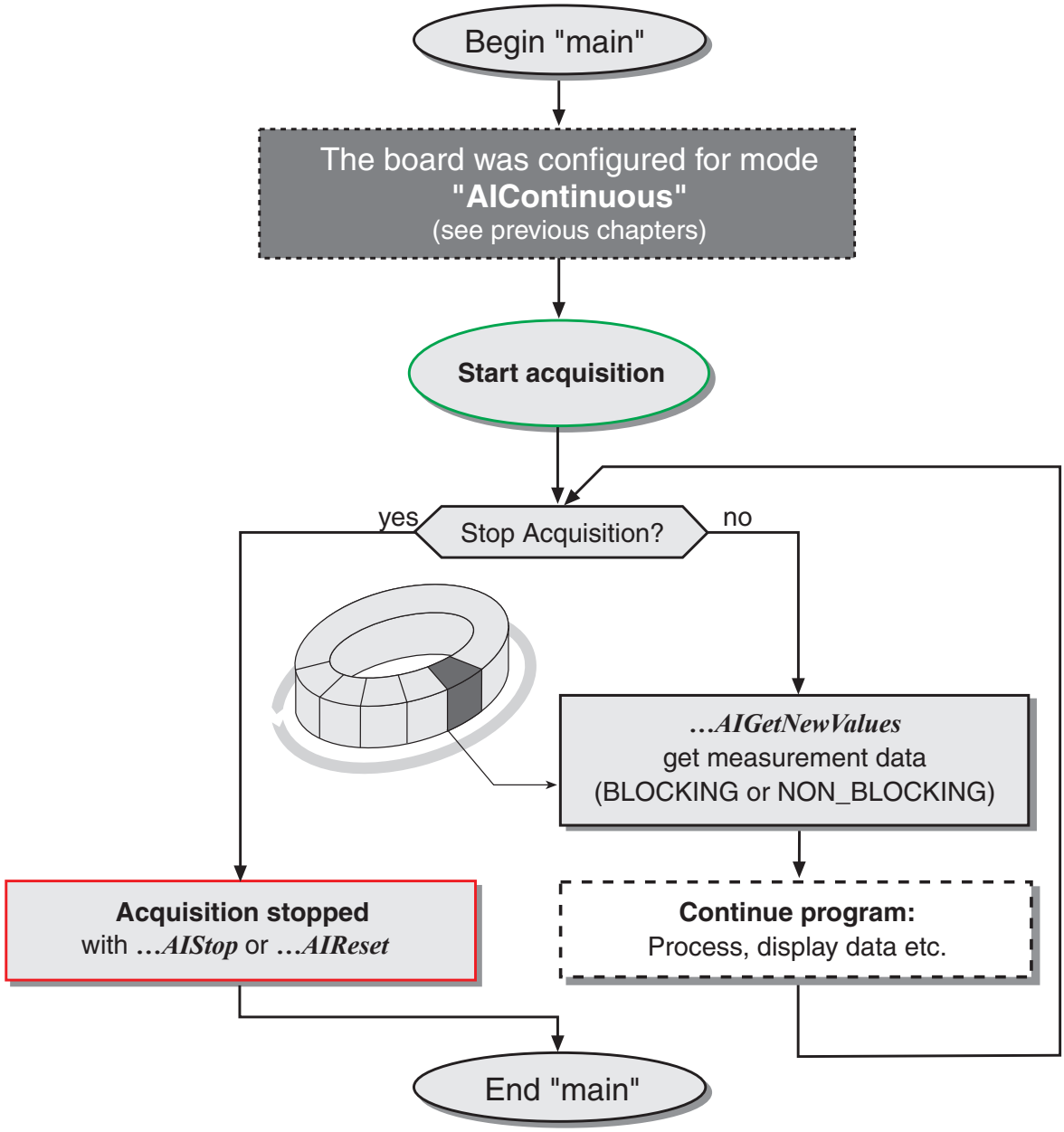



*Diagram 28: Programming „AIContinuous“ with …AIGetNewValues*

#### 4.1.3.2.2 Operation Mode „AIScan“

The function …*AIScan* allows the acquisition of a known number of measured values. The measured values are stored in a user defined data buffer when the data acquisition is completed. In the execution mode “BLOCKING”, the thread in which the …*AIStart* function is called returns when the last value is acquired. In the execution mode “ASYNCHRONOUS” the data acquisition is started as a background process. A new thread is automatically created when the …*AIStart* function is called. Parallel to this, other threads can still be processed. If required (e. g. for a longer lasting scan operation) the user can already „view” the measured values while the acquisition is running. This is done with a user defined callback function or by calling the function …*AIGetNewValues*. A “Terminate” function allows (if desired) a notification to the application that the data acquisition process is completed.

The diagrams on the following pages describe the program flow under the following conditions:

a. The data acquisition blocks the program flow until all values are acquired into the data buffer (execution mode BLOCKING for the …*AIScan* function).

b. The data acquisition runs in a background operation with a user defined callback function (execution mode ASYNCHRONOUS for the …*AIScan* function)

c. „Viewing” the data with the function …*AIGetNewValues* during a running acquisition in background operation (execution mode ASYNCHRONOUS for the …*AIScan* function).

For more information, refer to the sample programs found in the ME-SDK and the function descriptions on page 123.

**Note to a:** Data handling in the **„AIScan“** operation mode using the run mode “BLOCKING”:

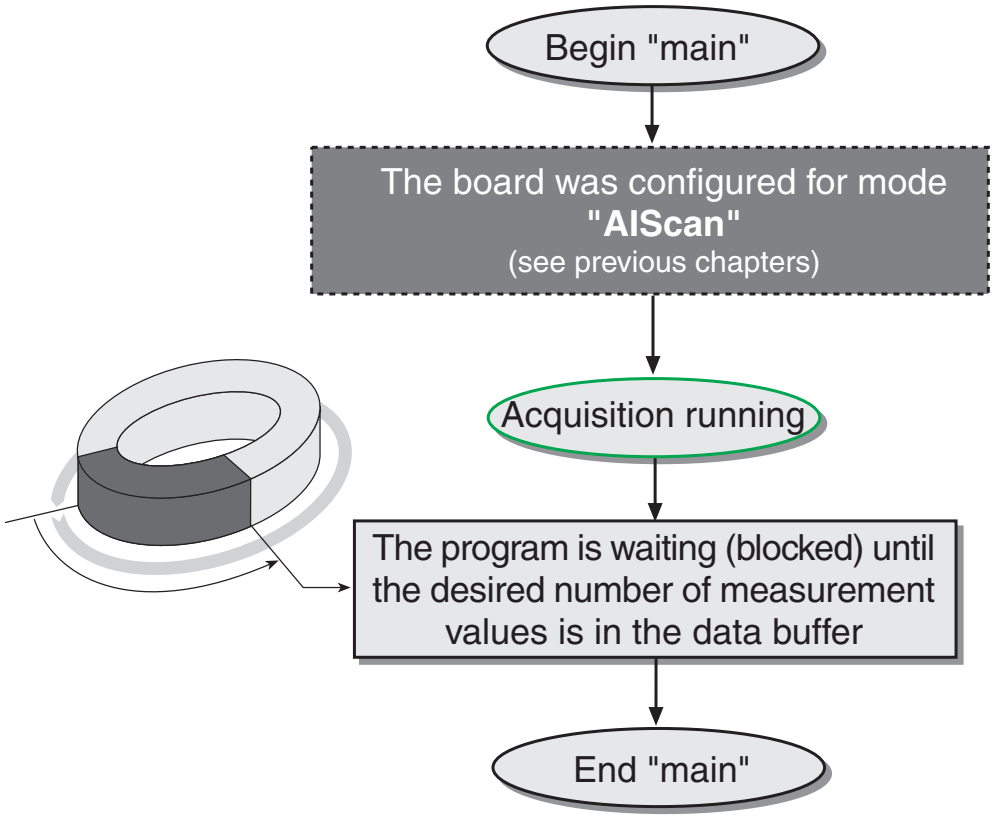



*Diagram 29: Programming „AIScan“ in BLOCKING mode*

**Note to b:** Data handling in the **„AIScan“** operation mode using the run mode “ASYNCHRONOUS” and a user defined callback function.

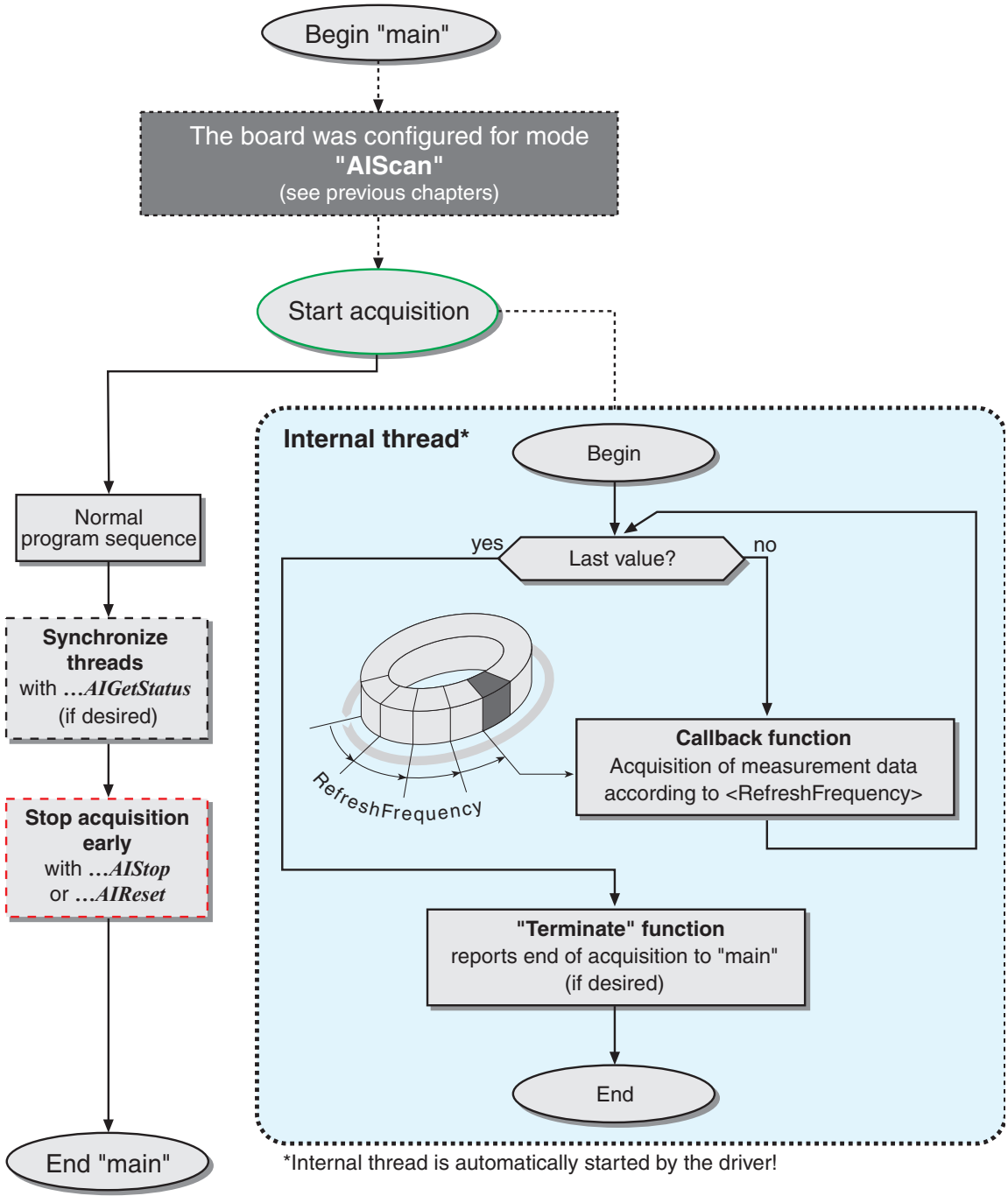



*Diagram 30: Programming „AIScan“ with callback function*

**Note to c:** Data handling in the **„AIScan“** operation mode using the execution mode “ASYNCHRONOUS” and the function *AIGetNewValues* (BLOCKING or NON_BLOCKING)

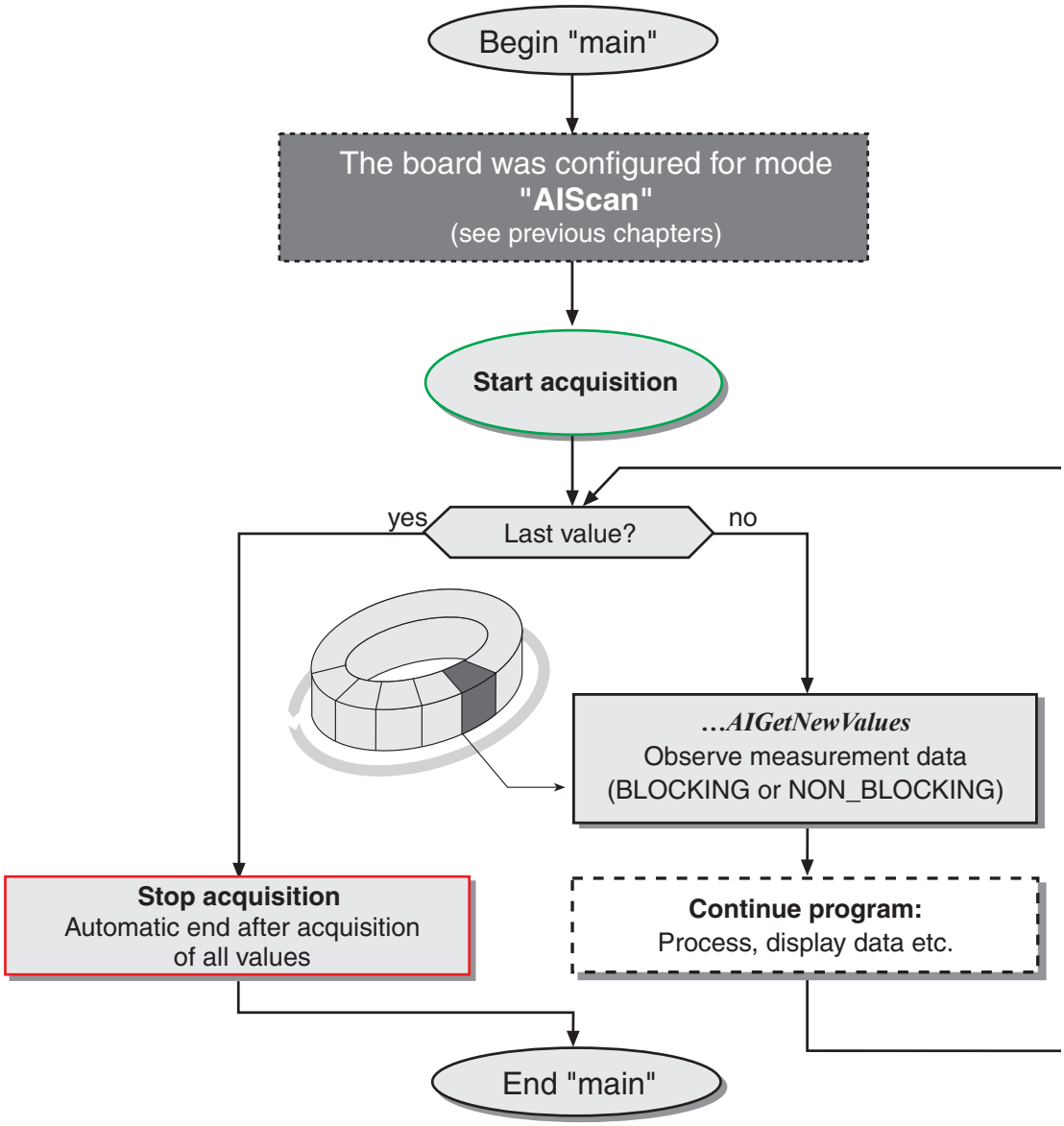



*Diagram 31: Programming „AIScan“ with …AIGetNewValues*

#### 4.1.3.3 Starting the Data Acquisition

As soon as the function *…AIStart* was called the board is ready for running. Depending on the acquisition mode the operation will either start immediately after calling the function (software start) or the board waits for the matching external trigger pulse. If you use an external trigger but the external trigger pulse does not occur, the data acquisition can be cancelled by a useful time-out value. Data acquisition with an external trigger signal is explained in detail in chapter 4.1.4.

#### 4.1.3.4 Ending the Data Acquisition

In the "AIContinuous" mode of operation the data acquisition is ended when the function *…AIStop* is called.

When using the „AIScan" mode of operation, the data acquisition is stopped automatically when the required number of measured values is reached.

The acquisition process can be started again from the beginning any time by calling the function *…AIStart* as long as the mode of operation has not been changed.

## 4.1.4 External Trigger A/D Section

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

Depending on the type of application required, you can choose between different „acquisition modes“ with differing philosophies. The acquisition modes are independent of the trigger mode (analog or digital) and trigger edge (rising, falling or both edges). To enable the external trigger choose one of the acquisition modes described below in the `<AcqMode>` parameter of the function *…AIConfig*. Don´t forget to „arm“ the board for data acquisition by calling the function *…AIStart*.

### 4.1.4.1 Acquisition Mode „External Standard“

(ME4000_AI_ACQ_MODE_EXT)

The board is ready to start the data acquisition after calling the function *…AIStart*. The acquisition begins when the first external trigger pulse is detected. The values are acquired according to the CHAN- and SCAN timer. Any further trigger pulses are ignored. Optional you can define a time-out within the external trigger pulse must occur else the operation will be cancelled (parameter `<TimeOutSeconds>`).

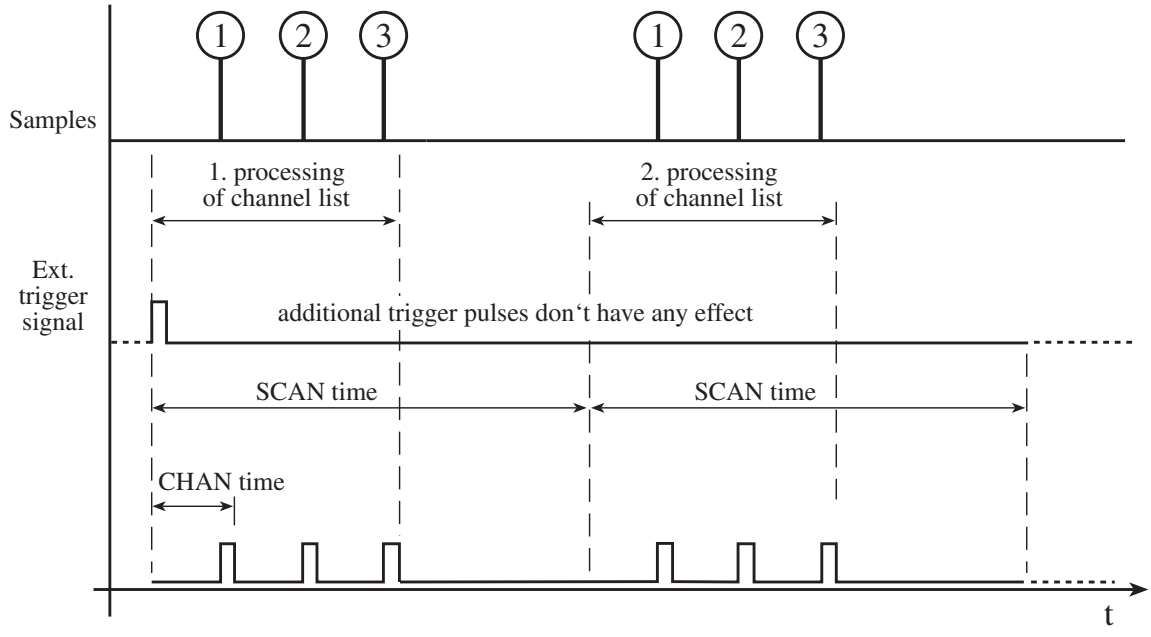



*Diagram 32: Acquisition mode „External Standard“*

#### 4.1.4.2 Acquisition Mode „External Single Value“

(ME4000_AI_ACQ_MODE_EXT_SINGLE_VALUE)

The board is ready to start the data acquisition after calling the function *…AIStart*. On each external trigger pulse the next value is acquired according to the channel list. When the first trigger pulse appears, a single time delay of one CHAN time occurs before the first conversion is done. For further operation the CHAN- and SCAN timer settings are ignored. The period of the external trigger pulse must be at least 2 µs.

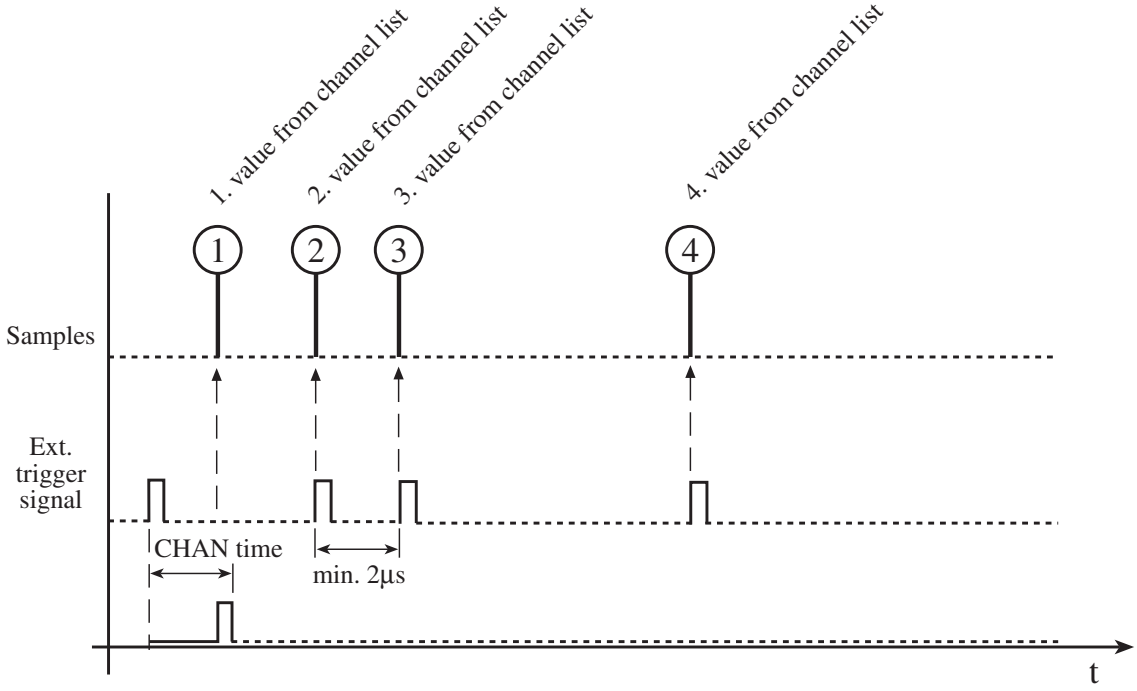



*Diagram 33: Acquisition mode „External Single Value“*

Optional you can define a time-out within the **first** trigger pulse must occur else the operation will be cancelled (parameter `<TimeOutSeconds>`). It is not checked if further trigger signals fail to appear. Note this when choosing the execution mode.

### 4.1.4.3 Acquisition Mode „External Channel List“

(ME4000_AI_ACQMODE_EXT_CHANNELLIST)

The board is ready to start the data acquisition after calling the function *…AIStart*. The channel list is processed once every time a external trigger pulse is detected. The data acquisition is done according to the CHAN timer. When the first trigger pulse appears, a single time delay of one CHAN time occurs before the first conversion is done. The SCAN timer setting is ignored in this operation mode. The minimum „trigger time“, i. e. the minimum period for the external trigger, is:

(Number of channel list entries x CHAN time) + 2µs

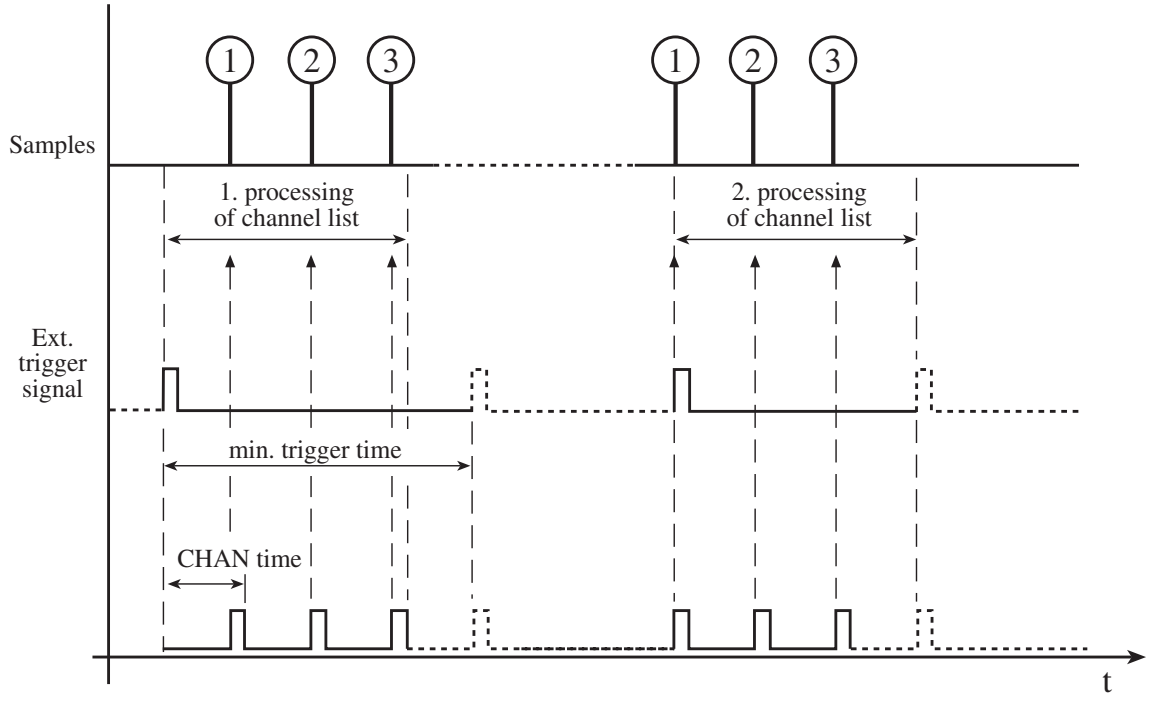



*Diagram 34: Acquisition Mode „External Channel List“*

Optional you can define a time-out within the **first** trigger pulse must occur else the operation will be cancelled (parameter `<TimeOutSeconds>`). It is not checked if further trigger signals fail to appear. Note this when choosing the execution mode.

# 4.2 D/A Section

| Operation Mode | Usage | Trigger | Timing |
|---|---|---|---|
| AOSingle (see page 51) | Output 1 value to 1 channel | Software start, ext. digital | – |
| AOSimultaneous (see page 52) | Update multiple channels simultaneously | Software start, ext. digital | – |
| AOContinuous (see page 56) | Timer controlled output of values changing continuously | Software start, ext. digital | D/A Timer (…AOConfig) |
| AOWraparound (see page 60) | Timer controlled output of values changing periodically | Softwarestart, ext. digital | D/A Timer (…AOConfig) |

*Table 3: D/A operation modes*

## 4.2.1 Operation Mode „AOSingle“

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | ✔ | ✔ | ✔ |

The voltage value to be output is loaded to the D/A converter of the selected channel by the function *…AOSingle*. Depending on the trigger mode the value is output at once (software start) or by an appropriate edge at the corresponding trigger input. No further configuration is required (see also page 137).

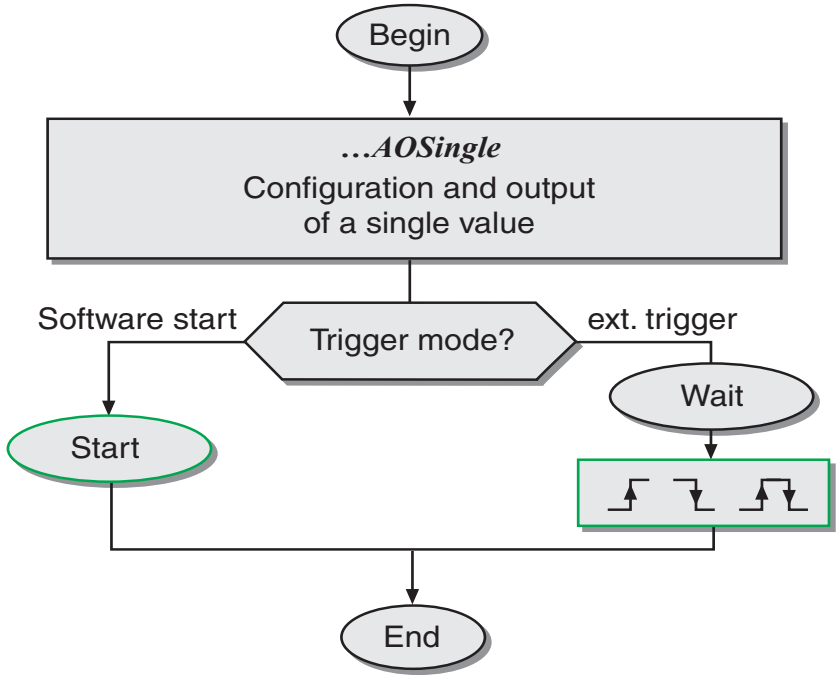



*Diagram 35: Programming „AOSingle“*

### 4.2.2 Operation Mode „AOSimultaneous“

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | ✔ | ✔ | ✔ |

The voltage values to be output are loaded to the appropriate D/A converters individually for every channel to be included in the simultaneous output. Depending on the selected trigger mode the channels are updated at once (software start) or by an appropriate edge at the selected trigger input. You can choose what trigger input (resp. inputs) of the simultaneous channels should start the output.

The following diagram shows the basic order of operation (see also the function descriptions on page 139 and the sample programs in the ME-SDK):

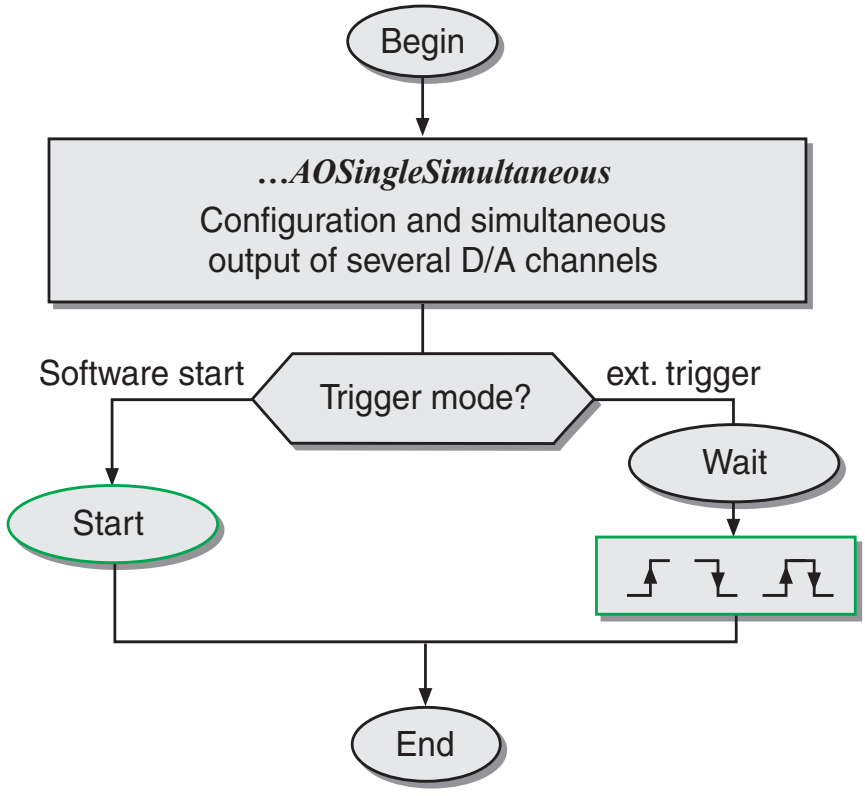



*Diagram 36: Programming „AOSimultaneous“*

### 4.2.3 Timer Controlled „AO Operation Modes“

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

The timer controlled output programming is done in 3 basic steps described below:

1. Hardware configuration for each channel with the function …*AOConfig* (see chapter 4.2.3.1)
2. Software preparation for each channel with the function …*AOContinuous* or …*AOWraparound* (see chapter 4.2.3.2)
3. Start of the analog output with the function …*AOStart* or …*AOStartSynchronous* (see chapter 4.2.3.3)

Before the output begins, you must decide whether the new values should be continuously reloaded while the output operation is running (AOContinuous) or whether the same values should be output periodically (AOWraparound). Once the hardware and software have been configured, the data output can be started by software or by an external trigger signal.

Diagram 37 shows the basic order of operation for the **“AOContinuous”** and **“AOWraparound”** modes. The following chapters describe the configuration, data handling and execution modes in more detail.

#### 4.2.3.1 Configuration of the D/A Section

The first step is to configure the hardware for each channel with the function …*AOConfig* (see function description on page 131).

- Select the wanted D/A channel.
- A 32 bit programmable counter with a 33MHz base frequency is used for the timer. This results in a period of 30.30ns, which is the smallest time unit available. This will be referred to as "1 Tick" in the following sections. The functions …*FrequencyToTicks* and …*TimeToTicks* offer a convenient way to convert the frequency resp. the period in ticks to program the timers. Sample rates between 500 kS/s and 0.5 samples per minute can be set.
- The following trigger modes are available:
  - Software start: the output to the specified channel is started with the function …*AOStart* resp. with the first matching trigger event.
  - Synchronous software start: Synchronous start of multiple channels after the proper configuration with the function …*AOConfig*. The output starts immediately after calling the function …*AOStartSynchronous*.
  - Start by the first external trigger pulse (rising, falling or any edge) on the appropriate input line DA_TRIG_*x*.
  - Synchronous start by the first external trigger pulse (rising, falling or any edge) on one or several input lines (DA_TRIG_*x*).

The diagram below shows the order of operation in the **„AOContinuous“** and **„AOWraparound“** modes.

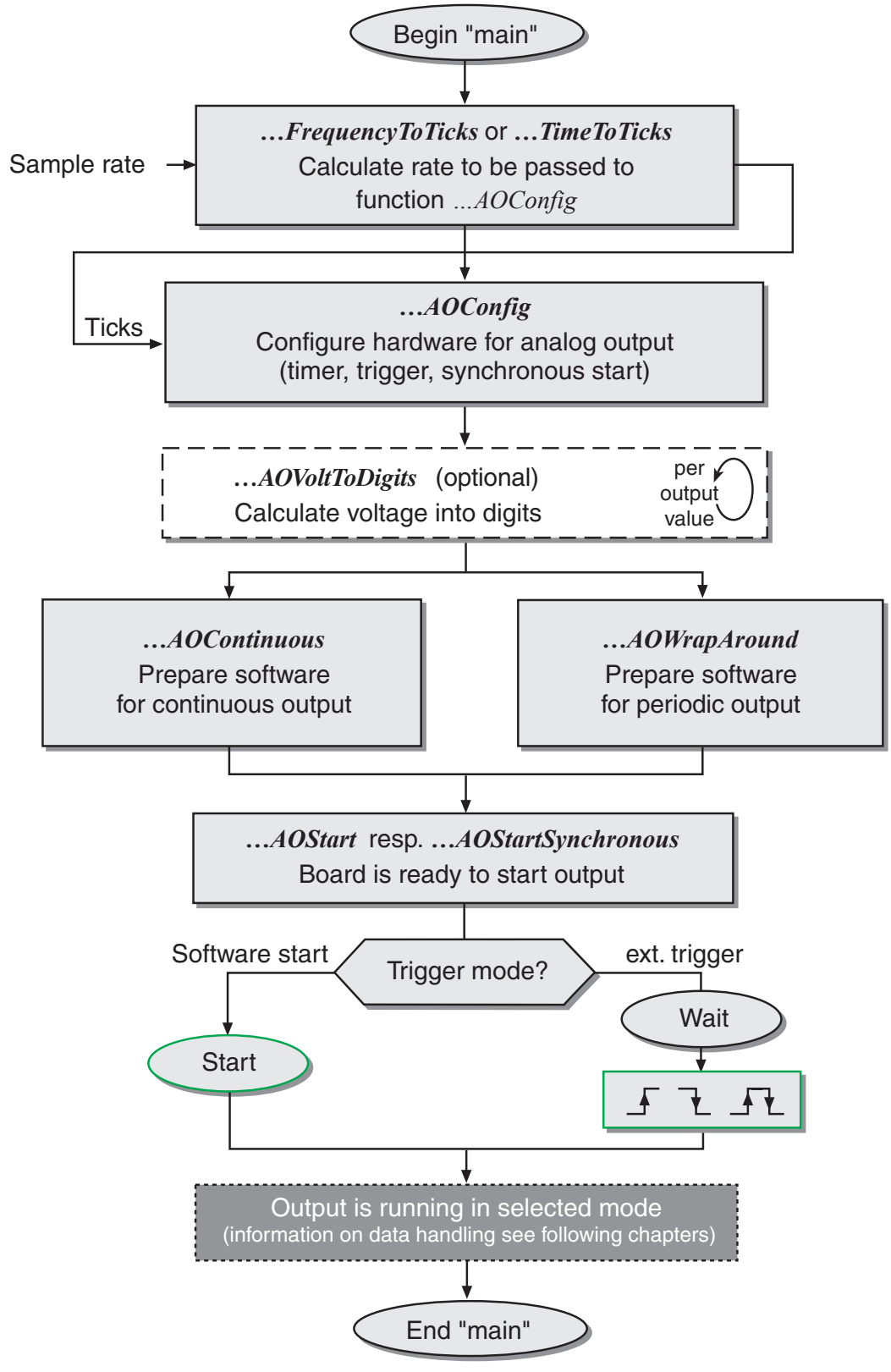



*Diagram 37: Programming AO Section*

#### 4.2.3.2 Software Preparation

The second step is to prepare the software for the analog output. Depending on your decision whether new values should be reloaded continuously (AOContinuous – see chapter 4.2.3.2.1) or whether the same values should be output periodically (AOWraparound – see chapter 4.2.3.2.2) different functions are available. The operation modes are described in the following chapters.

The driver operates internally with a ring buffer used to store the values to be output.

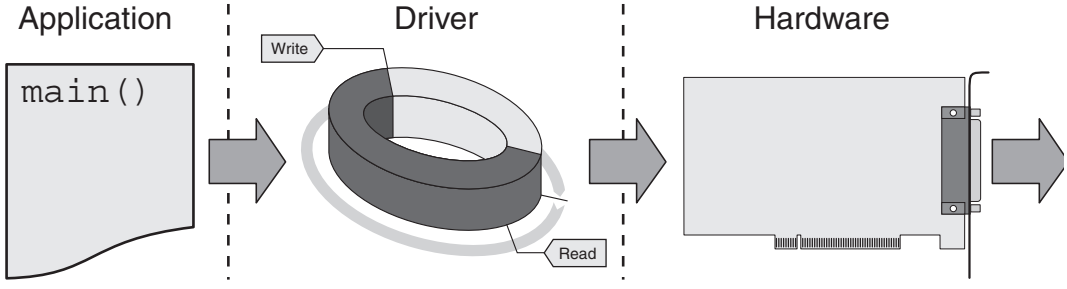



*Diagram 38: Ring buffer AO operation*

##### 4.2.3.2.1 Operation Mode „AOContinuous“

This mode allows the independent output of arbitrary analog signals to D/A channels 0…3. The values can also be changed or newly calculated during operation (opposite to the „AOWraparound“ mode). A data buffer must be allocated for each output channel to be used. The buffer contains the values to be output **first**. Before the output begins, the first data package is written to the ring buffer. The D/A timer defines the sample rate for converting the single values. The timer must be configured with the function …*AOConfig* before the output is started.

Loading new values into the ring buffer is done with the function …*AOAppendNewValues*. In the execution mode NON_BLOCKING only the number of values which have enough space in the ring buffer (NON_BLOCKING) are loaded. The execution mode BLOCKING can not be recommended, because of the internal thread is blocked until all values are loaded. The diagrams on the following pages show the program flow for the following conditions:

a. The output is done as a background operation (asynchronous) with the function …*AOAppendNewValues* and uses a user defined callback function.

b. The output is done as a background operation (asynchronous) with the function …*AOAppendNewValues.*

For more information, refer to the sample programs in the ME-SDK and the function description on page 133.

**Note to a:** Data handling in the operation mode **„AOContinuous“** using the function *…AOAppendNewValues* within a user defined callback function:

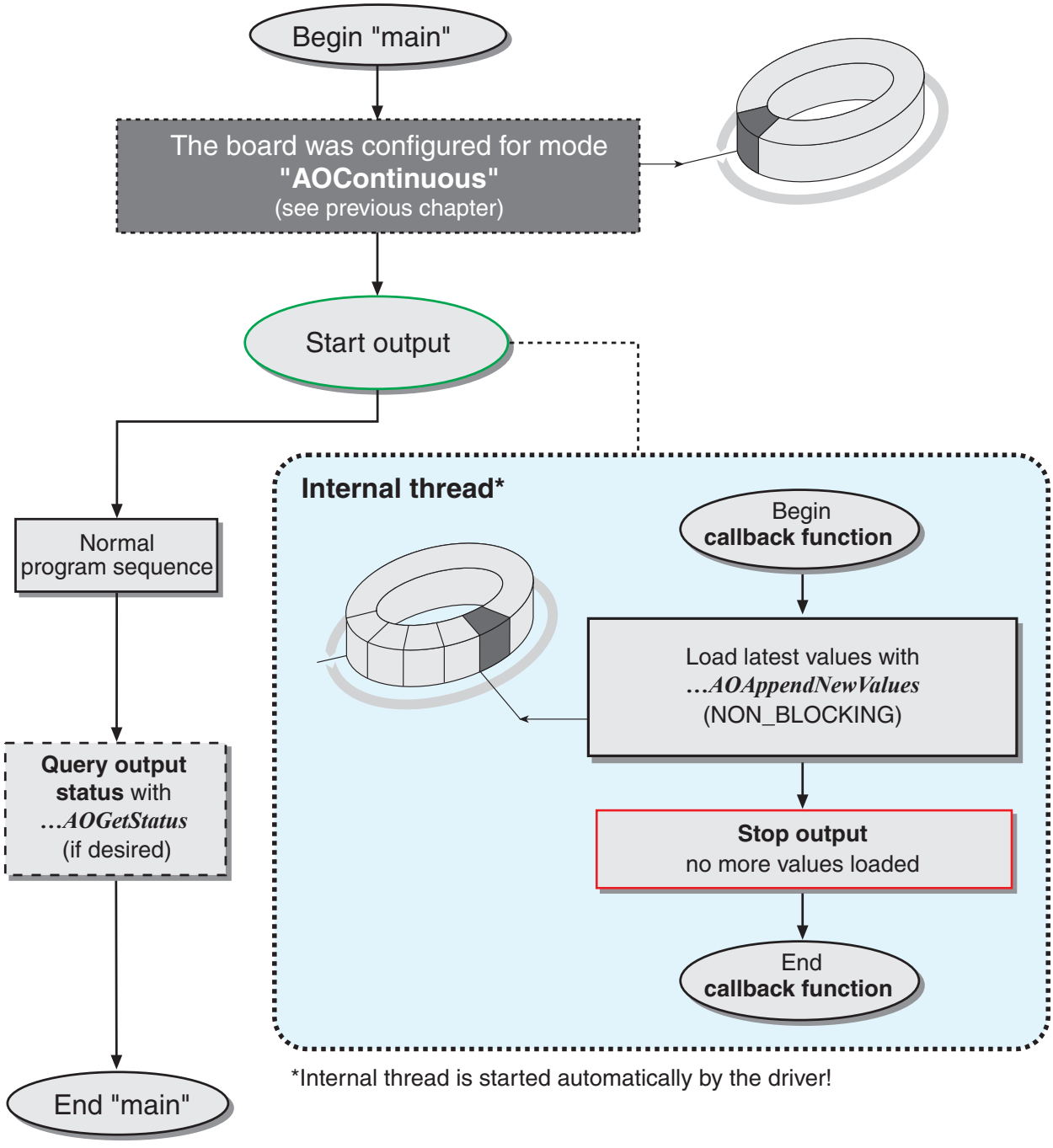



*Diagram 39: Programming „AOContinuous“ with callback function*

**Note to b:** Data handling in the operation mode **„AOContinuous“** using the function *…AOAppendNewValues*:

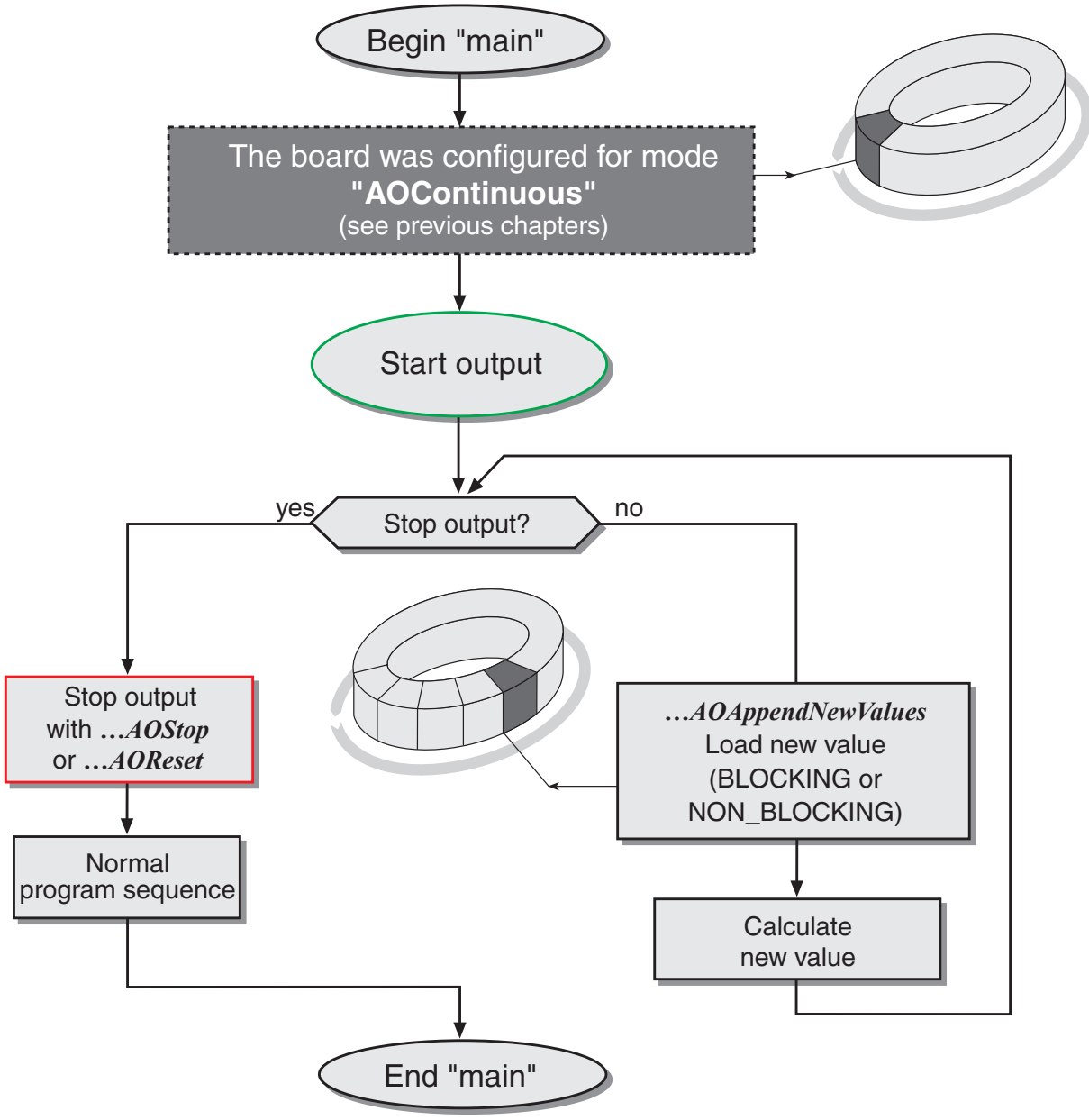



*Diagram 40: Programming „AOContinuous“ without callback function*

#### 4.2.3.2.2 Operation Mode „AOWraparound“

Using the „AOWraparound“ operation mode allows independent output of periodic analog signals to D/A channels 0…3. A data buffer must be allocated for each channel used. This data buffer is loaded with the values to be output repeatedly. Before starting the operation the values are written to the data buffer **once**. The D/A timer defines the sample rate for converting the single values. The timer must be configured with the function …*AOConfig* before the output is started.

**Note:** If the number of values in the data buffer does not exceed 4096 and the output runs in “infinite” mode, the entire operation works on the firmware level and does not load the host PC!

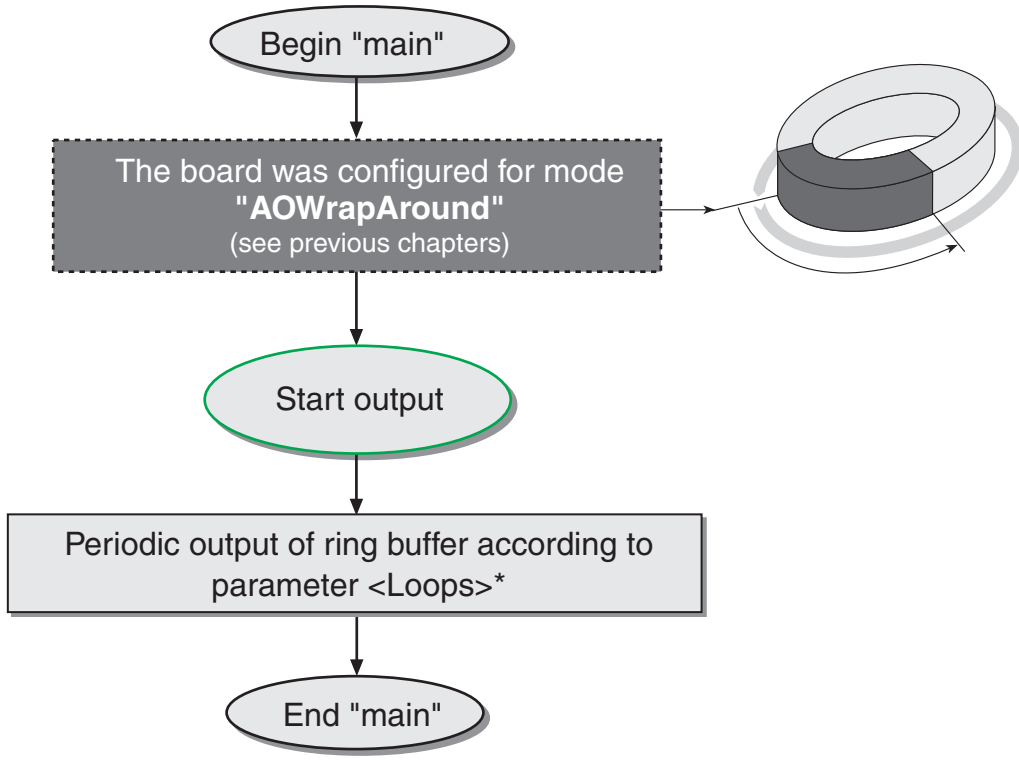
The diagrams on the following pages show the program flow for the following conditions:

a. The program flow is blocked until the output is ended (execution mode BLOCKING). The parameter `<Loops>` defines how often the data buffer is to be output. The value “infinite” is not possible here.

b. The output is done as a background operation (execution mode ASYNCHRONOUS). The parameter `<Loops>` defines how often the data buffer is to be output. Unlike the BLOCKING mode, it is also possible to choose “infinite” for the parameter `<Loops>`. The calling thread is not blocked.

For more information, refer to the sample programs in the ME-SDK and the function description on page 150.

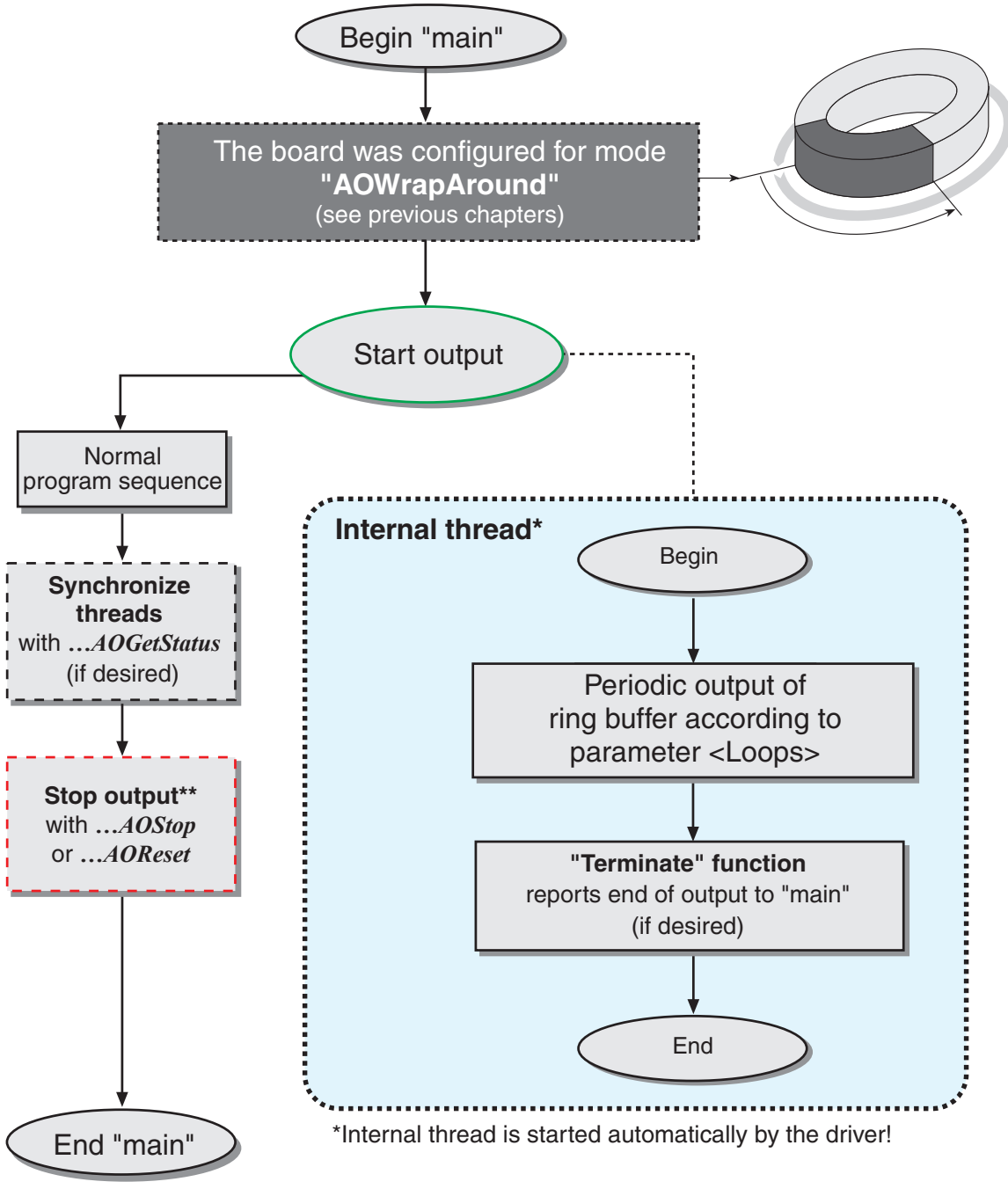
**Note to a:** Data handling in the operation mode **„AOWrap-around“** using the execution mode „BLOCKING“:




* Use a finite value in the parameter `<Loops>` (the program is blocked until the output operation is ended).

*Diagram 41: Programming „AOWraparound“ in „BLOCKING“ mode*

**Note to b:** Programming in the operation mode **„AOWrap-around“** using the execution mode „ASYNCHRONOUS“:




** This call is only required if the output is to be ended before the defined number of loops is completed or if the parameter `<Loops>` is set to “Infinite”.

*Diagram 42: Programming „AOWraparound“ in „ASYNCHRONUOUS“ mode*

#### 4.2.3.3 Starting the Analog Output

Either call the function *…AOStart* (starting a single channel) or the function *…AOStartSynchronous* (synchronous start of multiple channels) to make the board ready for running. Depending on the trigger mode the operation will either start immediately after calling the function (software start) or the board waits for the matching external trigger event. If you use an external trigger but the external trigger pulse does not occur, the output operation can be cancelled by a useful time-out value.

#### 4.2.3.4 Ending the Analog Output

As a rule in the "AOContinuous" mode the output can be ended by not reloading the data buffer values. Calling the function *…AOStop* will end the output immediately. The voltage level will be set to 0V at the appropriate D/A channel.

In the "AOWraparound" mode the output can be either stopped immediately by the function *…AOStop* (and set to 0V) or can be ended with the last value in the buffer. This results in the output staying at a known voltage level when the output operation is stopped.

If no change was made to the operation mode of the channel, the output can be started again from the beginning with the functions *…AOStart* resp. *…AOStartSynchronous* at any time.

# 4.3 Digital I/O Section

All models of the ME-4600 series provide four digital I/O ports, each being 8 bits wide (ports A, B, C and D). The optically isolated board versions have port A fixed as output and port B fixed as input. Ports C and D are not isolated.

For more information about the digital I/O circuits, refer to chapter 3.5 "Digital I/O Section".

## 4.3.1 Digital I/O Standard

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

Each port can be independently configured as input or output (exception: optically isolated versions). On power up, all ports are set to input with the exception of port A on optically isolated models („i“ versions). The function …*DIOConfig* is used to define the direction of the digital I/O ports.

**Note:** Ports defined as output can still be read!

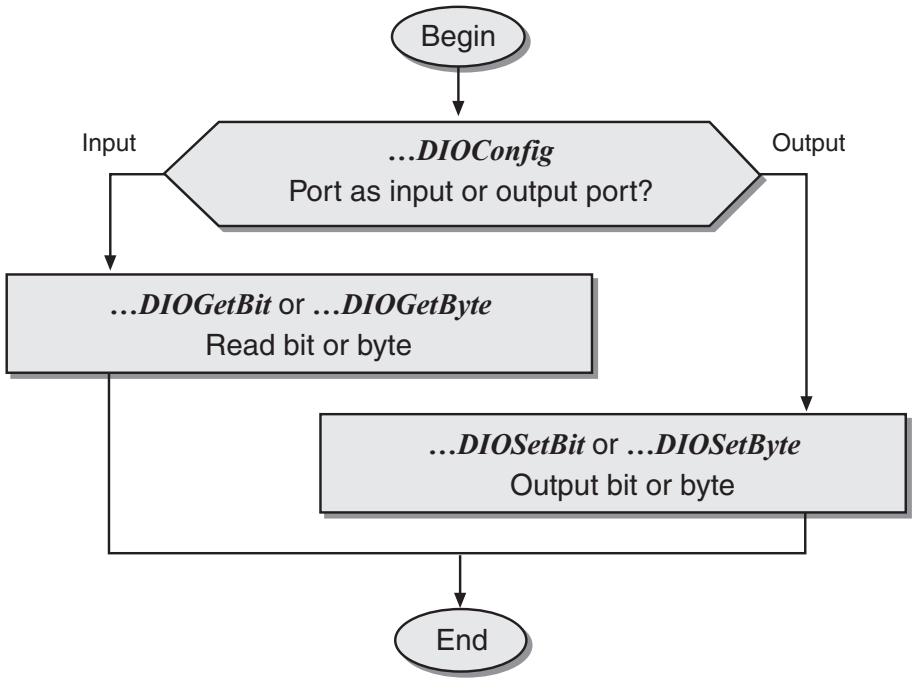



*Diagram 43: Programming „DIO Standard“*

### 4.3.2 Bit Pattern Output

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

A special feature of the ME-4680 is the timer controlled bit pattern output. The FIFO from D/A channel 3 serves a special purpose for doing this. Separated into low byte (FIFO_LOW_BYTE) and high byte (FIFO_HIGH_BYTE), the 16 bit wide FIFO values (bit patterns) can be assigned by bytes to the 8 bit wide digital ports (A, B, C and D). See also the function description for …*DIOBPPortConfig*. The ports used for the bit pattern output are automatically configured as output (it is not necessary to configure the port direction with the function …*DIOPortConfig*). The input port B on optically isolated board versions can not be used for bit pattern output.

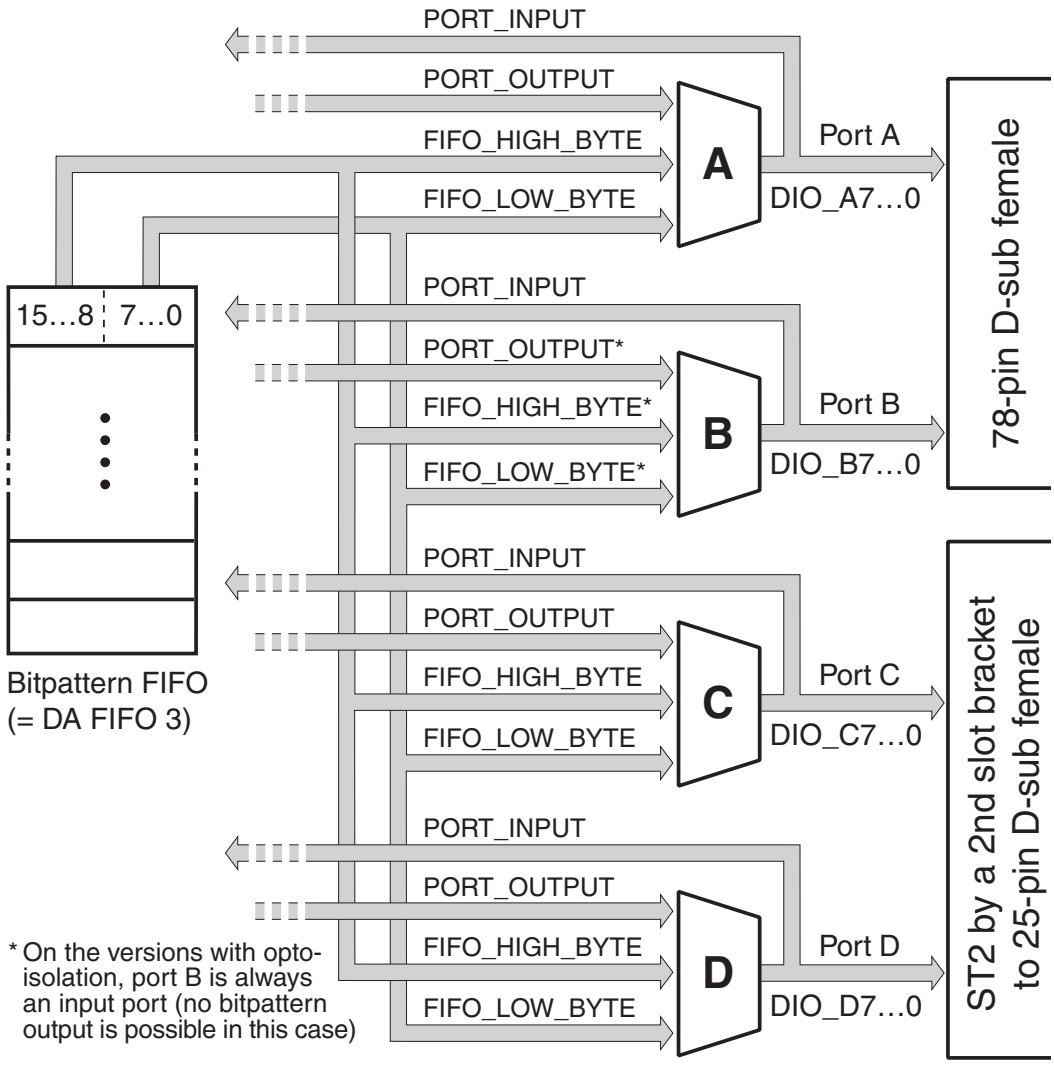



*Diagram 44: Port mapping*

The following diagram shows the order of operation in the operation modes **"BitPattern-Continuous"** and **"BitPattern-Wrap-around"**.

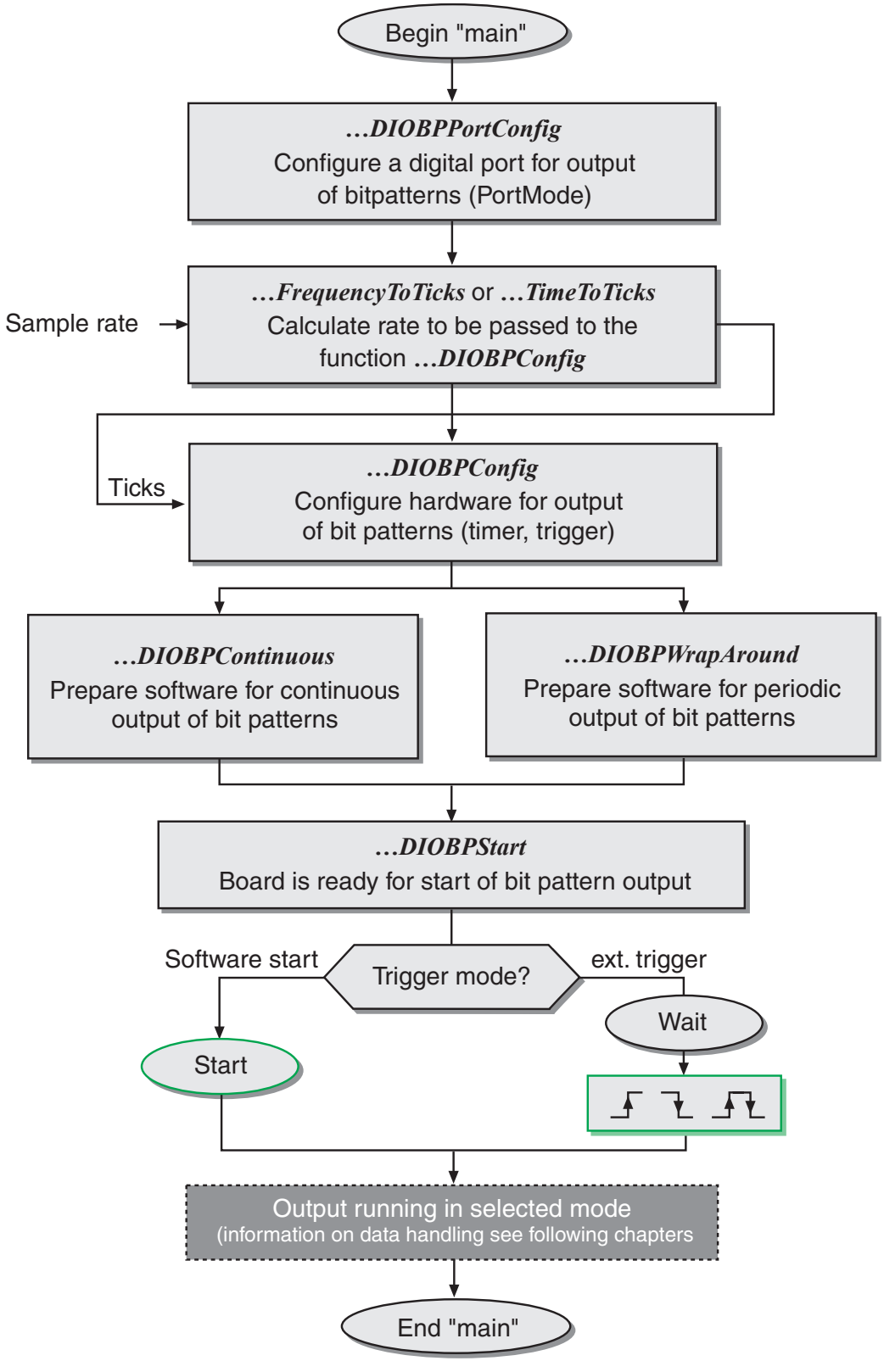



*Diagram 45: Programming bit pattern output*

Programming the timer controlled bit pattern output has 3 basic steps:

1. Configuration of the hardware with the functions *…DIOBPPortConfig* and *…DIOBPConfig* (see chapter 4.2.3.1)
2. Prepare the software with the functions *…DIOBPContinuous* or *…DIOBPWraparound* (see chapter 4.2.3.2)
3. Starting the bit pattern output with the function *…DIOBPStart* (see chapter 4.2.3.3)

Before starting the bit pattern output, you must decide whether the new values should be continuously reloaded while the output operation is running (BitPattern-Continuous) or whether the same values should be output periodically (BitPattern-Wraparound). Once the hardware and software have been configured, the data output can be started by software or by an external trigger signal.

#### 4.3.2.1 Hardware Configuration

The first step is to configure the hardware for the bit pattern output:

- Choose the port(s) to be used for the bit pattern output (see function description *…DIOBPPortConfig* on page 167).
- A 32 bit programmable counter with a 33MHz base frequency is used for the timer. This results in a period of 30.$\overline{30}$ns, which is the smallest time unit available. This will be referred to as "1 Tick" in the following sections (see *…DIOBPConfig* on page 154). The functions *…FrequencyToTicks* and *…TimeToTicks* offer a convenient way to convert the frequency resp. the period in ticks to program the timers.
- The following trigger modes are available (see *…DIOBPConfig* on page 154):
  - Software start: the output is started with the function *…DIOBPStart.*
  - Start by an external trigger pulse (rising, falling or any edge) on the appropriate input line DA_TRIG_3 (Pin 65).

#### 4.3.2.2 Software Preparation

The second step is to prepare the software for the bit pattern output. Depending on your decision whether new values should be reloaded continuously (see chapter 4.3.2.2.1) or whether the same values should be output periodically (see chapter 4.3.2.2.2) different functions are available. The operation modes are described in the following chapters.

Internally the functions …*DIOBPContinuous* and …*DIOBPWraparound* use a ring buffer for storing the bit pattern values to be output (see diagram 38).

##### 4.3.2.2.1 Operation Mode „BitPattern-Continuous“

This mode allows the output of any bit patterns to the digital ports. The values can also be changed or newly calculated during operation (opposite to the „BitPattern-Wraparound“ mode). A data buffer must be allocated which contains the bit patterns to be output **first**. Before the output begins, the first data package is written to the ring buffer. The timer defines the sample rate for output the single bit patterns. The timer must be configured with the function …*DIOBPConfig* before the output is started.

Loading new values into the ring buffer is done with the function …*DIOBPAppendNewValues*. In the execution mode NON_BLOCKING only the number of values which have enough space in the ring buffer (NON_BLOCKING) are loaded. The execution mode BLOCKING can not be recommended, because of the internal thread is blocked until all values are loaded. The diagrams on the following pages show the program flow for the following conditions:

a. The output is done as a background operation (asynchronous) with the function …*DIOBPAppendNewValues* within a user defined callback function.

b. The output is done as a background operation (asynchronous) with the function …*DIOBPAppendNewValues*

For more information, refer to the sample programs in the ME-SDK and the function description on page 157.

**Note to a:** Data handling in the operation mode **„BitPattern-Continuous“** using the function *…DIOBPAppendNewValues* within a user defined callback function:

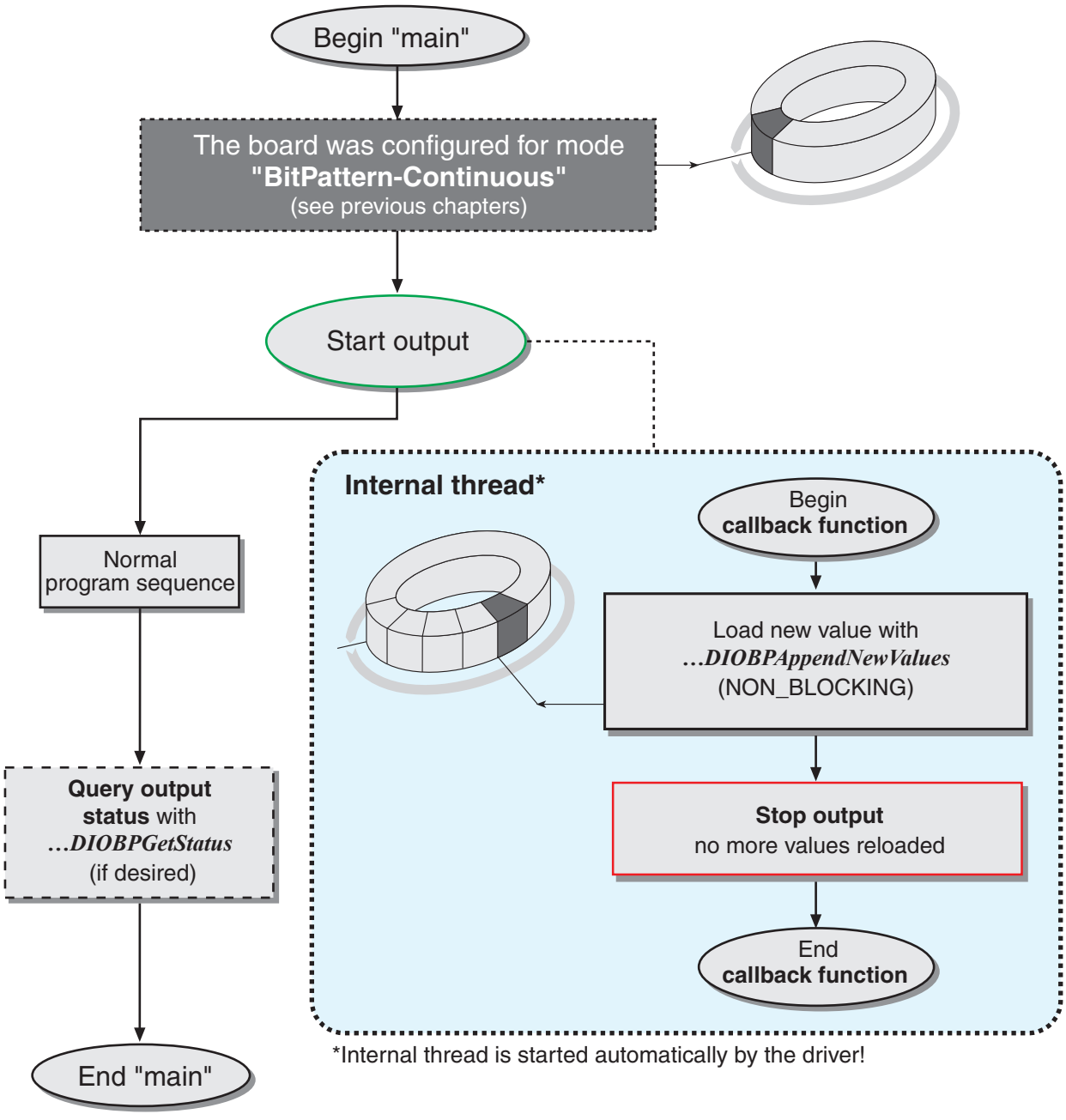



*Diagram 46: Programming „BitPattern-Continuous“ with callback function*

**Note to b:** Data handling in the operation mode **„BitPattern-Continuous“** with the function …*DIOBPAppendNewValues*:

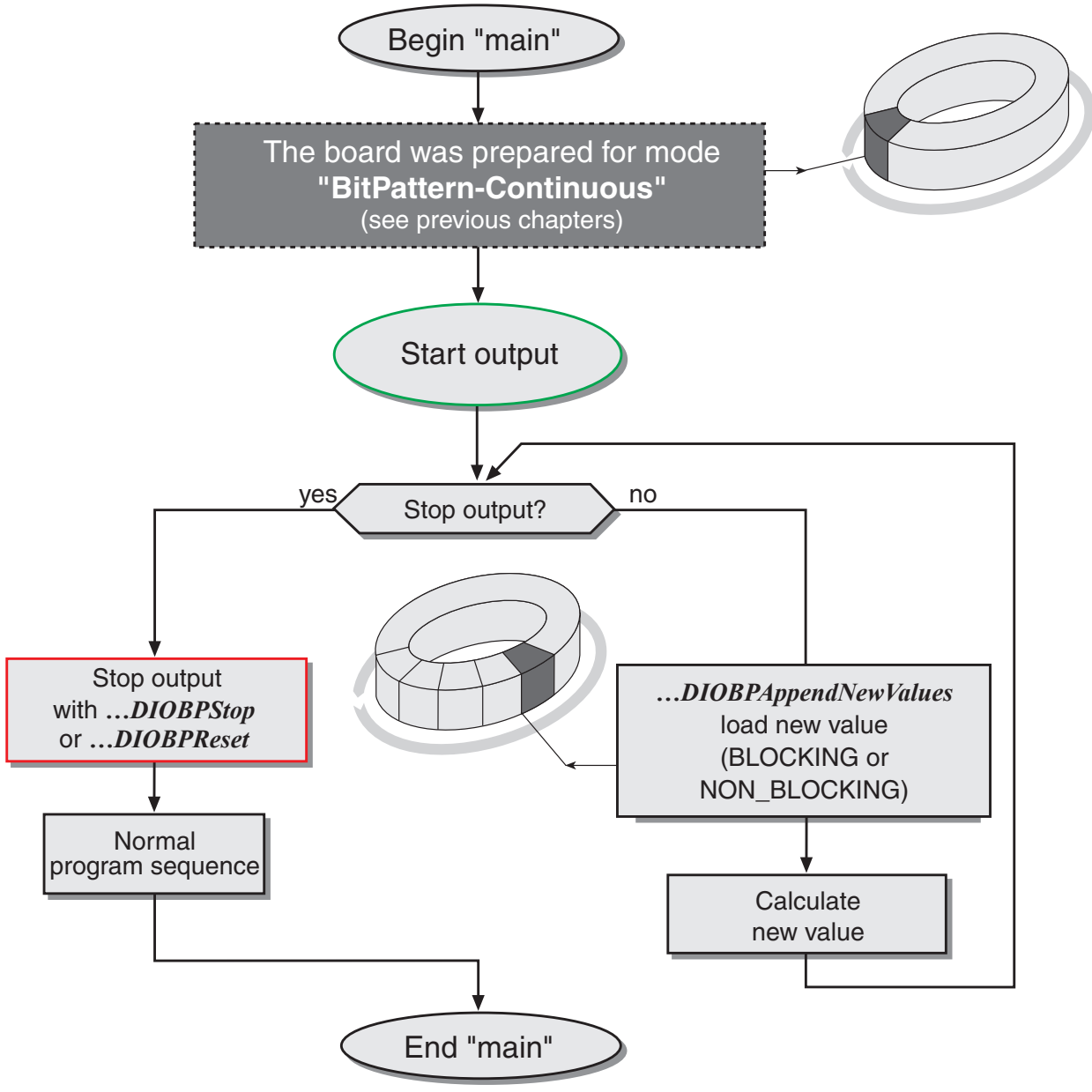



*Diagram 47: Programming „BitPattern-Continuous“ without callback function*

#### 4.3.2.2.2 Operation Mode „BitPattern-Wraparound“

The operation mode „BitPattern-Wraparound” allows periodic output of the same bit pattern values. A data buffer must be allocated with the bit patterns to be output repeatedly. Before starting the operation the bit patterns are written to the data buffer **once**. The timer defines the sample rate for output of the single bit patterns. The configuration is done with the function *…DIOBPConfig*.

**Note:** If the number of values in the data buffer does not exceed 4096 and the output runs in “infinite” mode, the entire operation works on the firmware level and does not load the host PC!

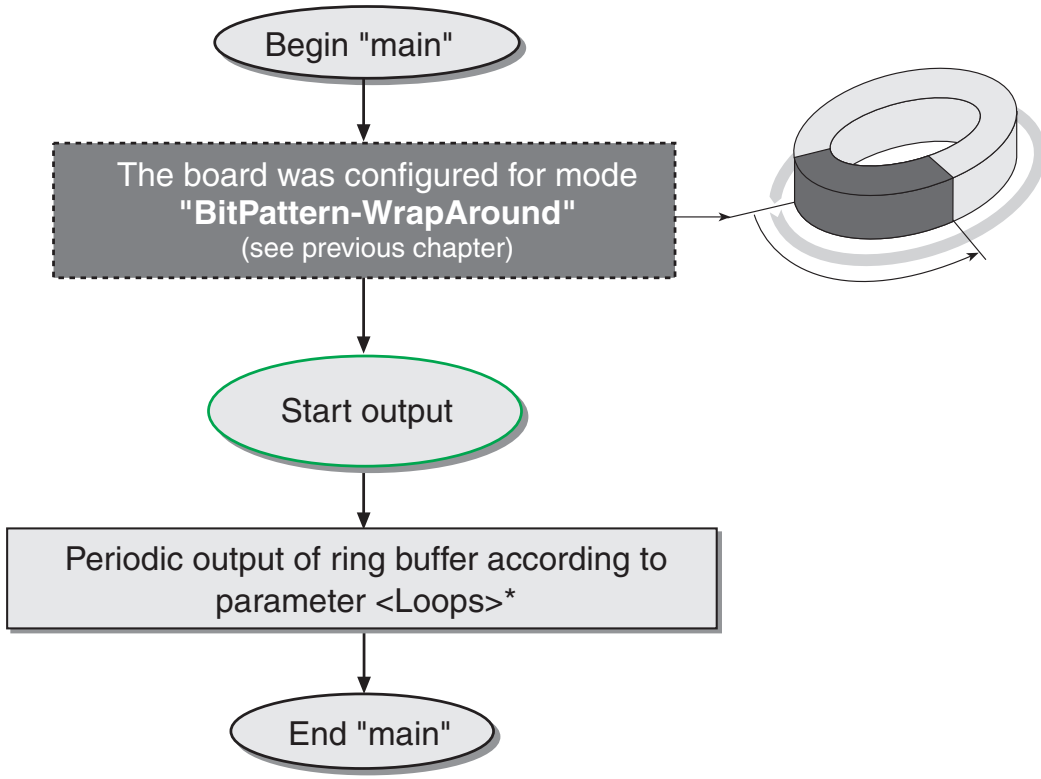
The diagrams on the following pages show the program flow for the following conditions:

a. The program flow is blocked until the output is ended (execution mode BLOCKING). The parameter `<Loops>` defines how often the data buffer is to be output. The value “infinite” is not possible here.

b. The output is done as a background operation (execution mode ASYNCHRONOUS). The parameter `<Loops>` defines how often the data buffer is to be output. Unlike the BLOCKING mode, it is also possible to choose “infinite” for the parameter `<Loops>`. The calling thread is not blocked.

For more information, refer to the sample programs in the ME-SDK and the function description on page 165.

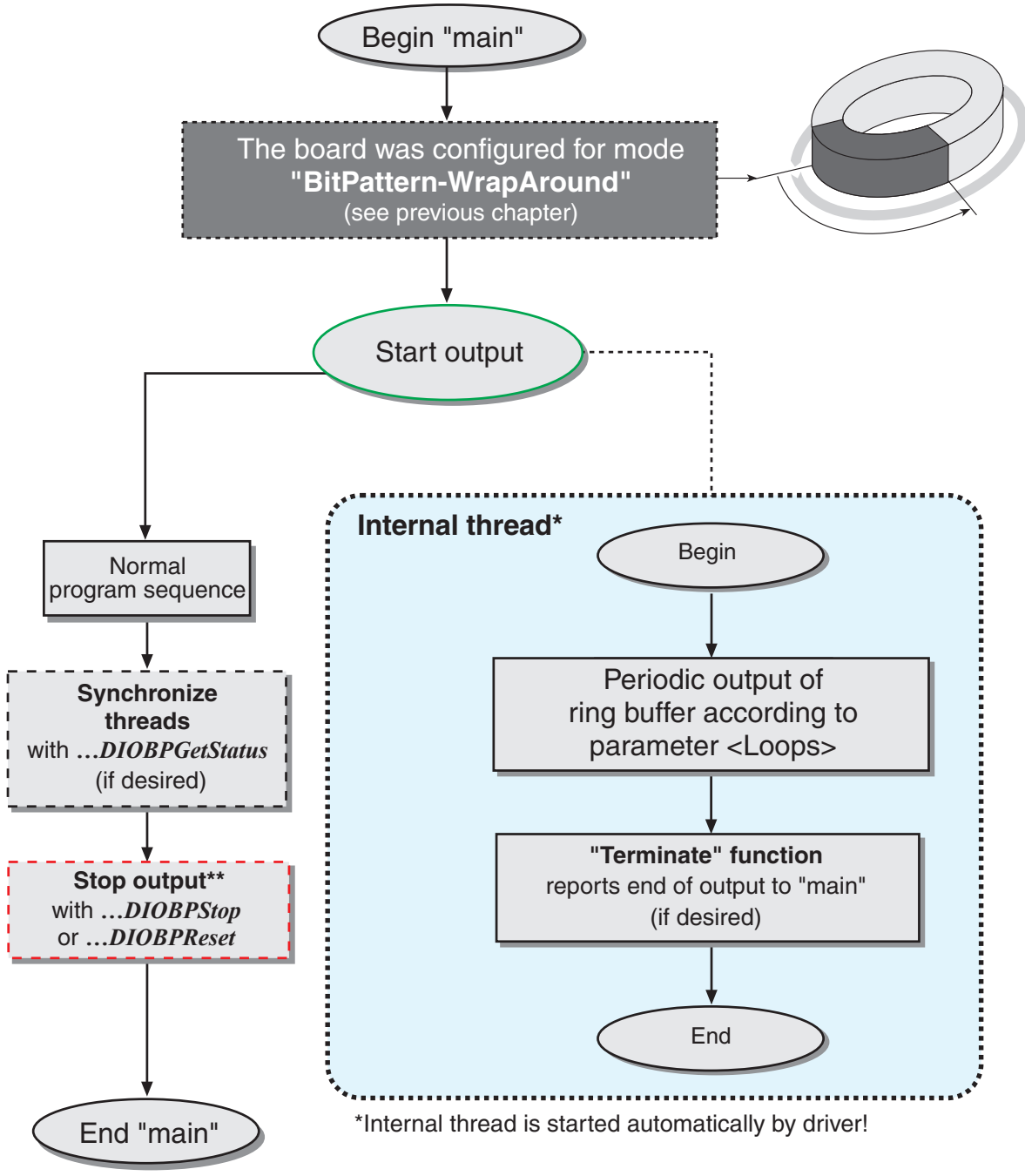
**Note to a:** Data handling in the operation mode **„BitPattern-Wraparound“** in execution mode „BLOCKING“:




* Use a finite value in the parameter `<Loops>` (the program is blocked until the output operation is ended).

*Diagram 48: Programming with …DIOBPWraparound in „BLOCKING“ mode*

**Note to b:** Programming in the operation mode **„BitPattern-Wraparound“** in execution mode „ASYNCHRONOUS“:




** This call is only required if the output is to be ended before the defined number of loops is completed or if the parameter <Loops> is set to “Infinite”.

*Diagram 49: Programming with …DIOBPWraparound in „ASYNCHRONUOUS“ mode*

#### 4.3.2.3 Starting the Bit Pattern Output

As soon as the function …*DIOBPStart* was called the board is clear for running. Depending on the trigger mode the operation will either start immediately after calling the function (software start) or waits for the matching external trigger event. If you use an external trigger but the external trigger pulse does not occur, the output operation can be cancelled by a useful time-out value.

#### 4.3.2.4 Ending the Bit Pattern Output

As a rule in the "BitPattern-Continuous" mode the output can be ended by not reloading the data buffer values. Calling the function …*DIOBPStop* will end the output immediately. The bit pattern will be set to 0000Hex.

In the "BitPattern-Wraparound" mode the output can be either stopped immediately by the function …*DIOBPStop* (and set to 0000Hex) or can be ended with the last value in the buffer. This results in a known bit pattern at the used digital ports when the output operation is stopped.

If no change was made to the operation mode, the output can be started again from the beginning with the function …*DIOBPStart* at any time.

# 4.4 Counter

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | ✔ | ✔ | ✔ |

Each of the three 16 bit counters (1 x 82C54) can be configured by software to function independently in the following 6 operational modes:

- Mode 0: Change state at zero
- Mode 1: Retriggerable „One Shot“
- Mode 2: Asymmetric divider
- Mode 3: Symmetric divider
- Mode 4: Counter start by software trigger
- Mode 5: Counter start by hardware trigger

For programming of the counters use the functions included with the ME-4600 driver software (see chapter 5.3.6 „Counter Functions“ on page 174).

## 4.4.1 Mode 0: Change State at Zero

This mode of operation can be used e.g. to trigger an interrupt when the counter reaches zero. The counter output (OUT_0…2) is set to low (TTL: 0V/Opto: conductive) when the counter is initialised or when a new start value is loaded. To enable counting, the proper voltage level must be applied to the GATE input (TTL: +5V/Opto: 0V). As soon as the start value is loaded and the counter is enabled, the counter begins counting downwards and the output remains low (TTL: 0V/Opto: conductive).

Upon zero axis crossing, the output is set to high (TTL: +5V/Opto: high impedance ) and remains there until the counter is reloaded or initialised again. The counter continues to count down, even after zero is meet. If a counter register is loaded during a count in progress the following occurs:

1. when the first byte is written, the count process is stopped.
2. when the second byte is written, the count process begins again.

### 4.4.2 Mode 1: Retriggerable „One-Shot“

The counter output (OUT_0…2) is set high (TTL: +5V/Opto: high impedance) when the counter is initialised. When a start value is loaded the output becomes low on the next clock following to the first trigger pulse at the GATE input (TTL: positive edge/Opto: negative edge). Upon zero axis crossing, the counter output is set to high (TTL: +5V/Opto: high impedance) again.

By a proper edge at the GATE input (TTL: positive edge/Opto: negative edge), the counter can be reset (re-triggered) to the start value. The output remains at low (TTL: 0V/Opto: conductive) until the counter meets zero.

The counter value can be read at any time without effecting the counting process.

### 4.4.3 Mode 2: Asymmetric Divider

In this mode, the counter functions as a frequency divider. The counter output (OUT_0…2) is set to high (TTL: +5V/Opto: high impedance) after initialisation. When the counter is enabled by applying the proper voltage level to the GATE input (TTL: +5V/Opto: 0V), the counter is counting downwards and the output remains high (TTL: +5V/Opto: high impedance). When the count meets the value 0001Hex, the output becomes low (TTL: 0V/Opto: conductive) for one clock cycle. This process will be repeated periodically as long as the GATE input is enabled (TTL: +5V/Opto: 0V), else the output is set to high (TTL: +5V/Opto: high impedance) immediately.

If the counter is reloaded between two output pulses, the current counter state is not affected. The new value is used on the following period.

### 4.4.4 Mode 3: Symmetric Divider

This mode of operation is similar to mode 2 with the difference that the divided frequency is symmetric (only for even count values). The counter output (OUT_0…2) is set to high (TTL: +5V/Opto: high impedance) after initialisation. When the GATE input is enabled (TTL: 5V/Opto: 0V), the counter is counting downwards in steps of 2. The output will toggle its state on a half

of the start value number of periods referenced to the input clock (starting with high level). This process will be repeated periodically as long as the GATE input is enabled (TTL: +5V/Opto: 0V), else the output is set to high (TTL: +5V/Opto: high impedance) immediately.

If the counter is reloaded between two output pulses, the current counter state is not affected. The new value is used on the following period.

### 4.4.5 Mode 4: Counter Start by Software Trigger

The counter output (OUT_0…2) is set to high (TTL: +5V/Opto: high impedance) when the counter is initialised. To enable the counter the GATE input must be enabled (TTL: +5V/Opto: 0V). When the counter is loaded (software trigger) and enabled, the counter starts counting downwards, while the output remains high (TTL: +5V/Opto: high impedance).

Upon zero axis crossing the output becomes low (TTL: 0V/Opto: conductive) for one clock cycle. Afterwards the output becomes high (TTL: +5V/Opto: high impedance) again and remains there until the counter is initialised and a new start value is loaded.

If the counter is reloaded during a count process, the new start value is used in the next cycle.

### 4.4.6 Mode 5: Counter Start by Hardware Trigger

The counter output (OUT_0…2) is set to high (TTL: +5V/Opto: high impedance) when the counter is initialised. After loading a start value to the counter, counting starts on the next clock following to the first trigger pulse at the GATE input (TTL: positive edge/Opto: negative edge). Upon zero axis crossing, the output becomes low (TTL: 0V/Opto: conductive) for one clock cycle. Afterwards the output becomes high (TTL: +5V/Opto: high impedance) again and remains there until the next trigger pulse occurs.

If the count register is reloaded between two trigger pulses, the new start value is used after the next trigger pulse.

The counter can be reset to the start value (re-triggered) at any time by applying a positive (TTL) resp. negative (Opto) edge to

the GATE input. The output will remains high (TTL: +5V/Opto: high impedance) until zero axis crossing is meet.

### 4.4.7 Pulse Width Modulation

A special application for the counters is the pulse width modulation (PWM). With this operation mode you can output a rectangular signal of maximum 50kHz at OUT_2 (pin 41) by a variable duty cycle. The conditions and the external switching of the counters are described in chapter 3.6.2 on page 28. Counter 0 is used as a prescaler for the externally driven base clock. Using the parameter `<Prescaler>` you can vary the frequency $f_{OUT\_2}$ as follows:

$$f_{OUT\_2} = \frac{\text{Base clock}}{\texttt{<Prescaler>} \cdot 100} \qquad (\text{with } \texttt{<Prescaler>} = 2\ldots(2^{16} - 1))$$

By the parameter `<DutyCycle>` you can set the duty cycle between 1…99% in steps of 1%. The operation is started immediately after calling the function …*CntPWMStart* and stopped by the function …*CntPWMStop*. No further programming of the counters is required.

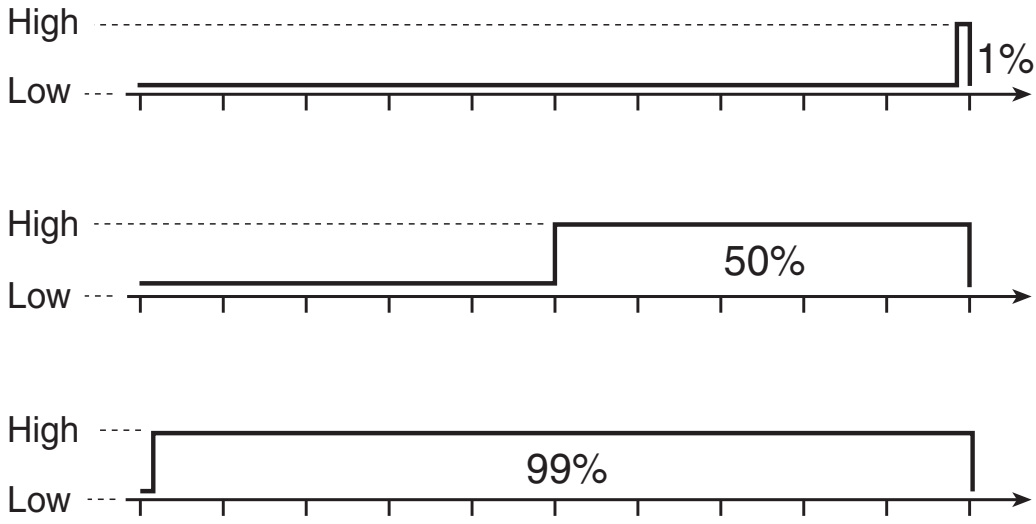



*Diagram 50: Duty cycle PWM signal*

* On opto-isolated boards the output OUT_2 is an open collector output. I. e. „High“ means output is driving and „Low“ output is in a high impedance state (see diagram 19 on page 27).

# 4.5 ME-MultiSig Control

***To understand the ME-MultiSig system and the functional descriptions in the following sections it is strongly recommended to fully read the ME-MultiSig manual!***

## 4.5.1 „Mux“ Operation

| Operation Mode | Usage | Trigger | Timing |
|---|---|---|---|
| MultiSig-AISingle (see page 203) | 1 channel (0…255), 1 measurement value | Software start, ext. digital, analog (opt.) | – |
| MultiSig-AIContinuous (see page 191) | Acquisition of an unknown number of measurement values corresponding to the „Mux“ channel list | Software start, ext. digital, analog (opt.) | CHAN Timer, SCAN Timer (…MultiSigAIConfig) |
| Multisig-AIScan (see page 203) | Acquisition of a known number of measurement values corresponding to the „Mux“ channel list | Software-Start, ext. digital, analog (opt.) | CHAN Timer, SCAN Timer (…MultiSigAIConfig) |

*Table 4: „Mux“ Operation*

In order to utilize the complete functional capability in „Mux“ operation, 2 digital output ports of the ME-4600 are required. When using a non optically isolated board, digital ports A and B are used. If an optically isolated board version is being used, the driver will automatically control ports A and C (port B is an input port on optically isolated boards). In this way, with help of the adapter ME-AA4-3i (optional) the full functional operation of the base boards can be achieved.

Make sure that the ME-MultiSig base boards (ME-MUX32-M/S) are configured for “Single-Mux” operation when using the „MultiSig“ functions of the ME-4600 driver. If a channel other than A/D channel 0 (default channel) is to be used, the solder bridge “A” for the desired A/D channel of the ME-4600 must be set (see the ME-MultSig manual). The channel set on the hardware must be passed to the parameter `<AIChannelNumber>` in the functions *…MultiSigAIConfig* resp. *…MultiSigAISingle*.

#### 4.5.1.1 Configuration of the Base Boards

In order to use the functional operation described in the following sections the configuration mode must be used (…*MultiSigOpen*, …*MultiSigClose*):

The following diagram demonstrates the program flow required:

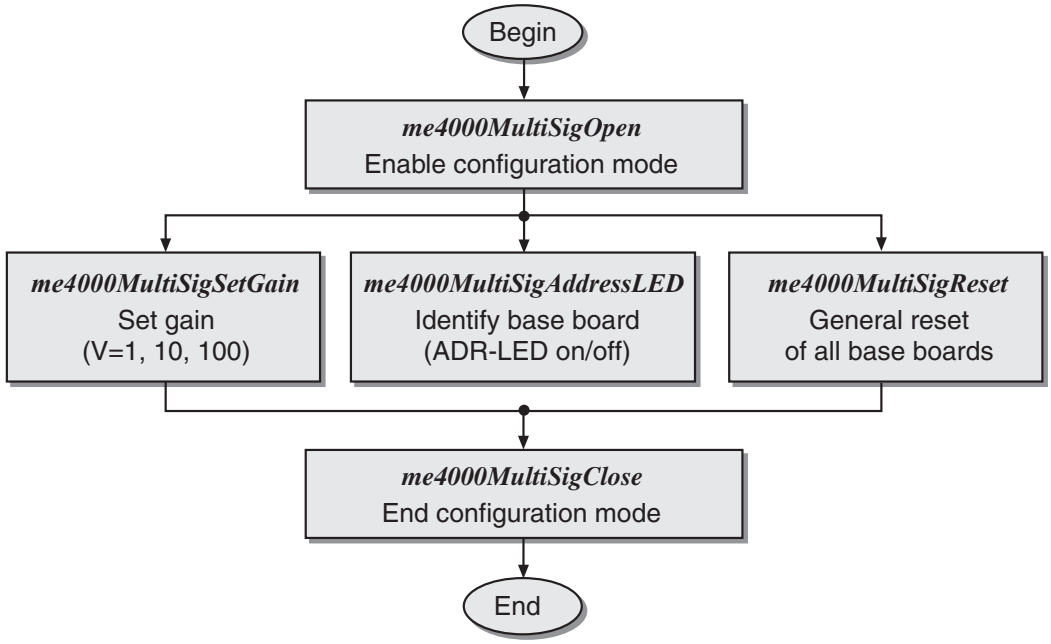



*Diagram 51: Basic configuration „Mux“ operation*

##### 4.5.1.1.1 Setting the Amplification

The amplification factor can be separately set for each channel group of the base boards ME-MUX32-M(aster) and ME-MUX32-S(lave). When using amplification factor V = 1 (standard) no configuration is required.

##### 4.5.1.1.2 Address LED Control

For maintenance purpose etc. the address LED of the base boards ME-MUX32-M and ME-MUX32-S can be directly controlled.

##### 4.5.1.1.3 General Reset

All master and slave boards can be reset to their default state with the function …*MultiSigReset*. The default state is as follows:

- Amplification V = 1.

- Address LEDs are all off

### 4.5.1.2 Operation Mode „MultiSig-AISingle“

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This mode of operation serves to acquire a single value from the selected channel of the Mux-system. The following parameters are available:

- Mux Channel numbers 0…255.
- A/D Channel numbers 0…31 (ME-4650/4660: 0…15).
- Gain factor of the channel group (V=1, 10, 100).
- Trigger modes: per software, external digital trigger, or external analog trigger (ME-4670/4680 only).
- External trigger: falling, rising or both.
- Time out: in case the external trigger is not detected.

The following diagram demonstrates the program flow:

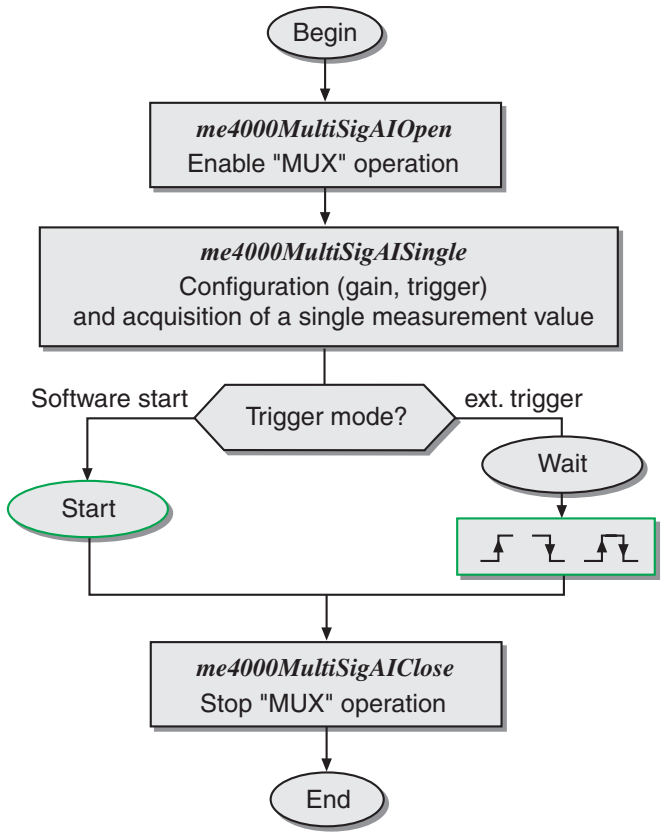



*Diagram 52: Programming „MultiSig-AISingle“*

### 4.5.1.3 Timer Controlled „Mux“ Operation

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This section describes the timer controlled acquisition by the ME-MultiSig system in conjunction with an ME-4600 series board. Base boards suitable for this purpose are the ME-MUX32-M (master) with up to 7 slave boards of type ME-MUX32-S (slave).

The following order of operations applies:

1. Open the „Mux“ operation with the function …*MultiSig-AIOpen* (see page 201)
2. Create a user defined Mux channel list to control the Mux channels (0…255)
3. Configuration of the A/D section of the ME-4600 with the function …*MultiSigAIConfig* (see page 188)
4. Preparation of the software with the function …*MultiSigAI-Continuous* (see page 191) resp. …*MultiSigAIScan* (see page 203)
5. Starting the data acquisition with the function …*MultiSigAI-Start* (see page 208)
6. Close the „Mux“ operation with the function …*MultiSig-AIClose* (see page 187)

Before the acquisition begins, you must decide whether a defined number of values should be acquired („MultiSig-AIScan“) or whether the values are to be acquired continuously („MultiSig-AI-Continuous“) until the process is stopped by the user. Once the hardware and software have been configured, the data acquisition can be started by software or by an external trigger signal.

The following diagram shows an overview of the process for the **„MultiSig-AIContinuous“** and **„MultiSig-AIScan“** modes for acquiring data values. The data handling follows the “normal” analog input operation:

- Data handling „AIContinuous“ see pages 40ff.
- Data handling „AIScan“ see pages 43ff.

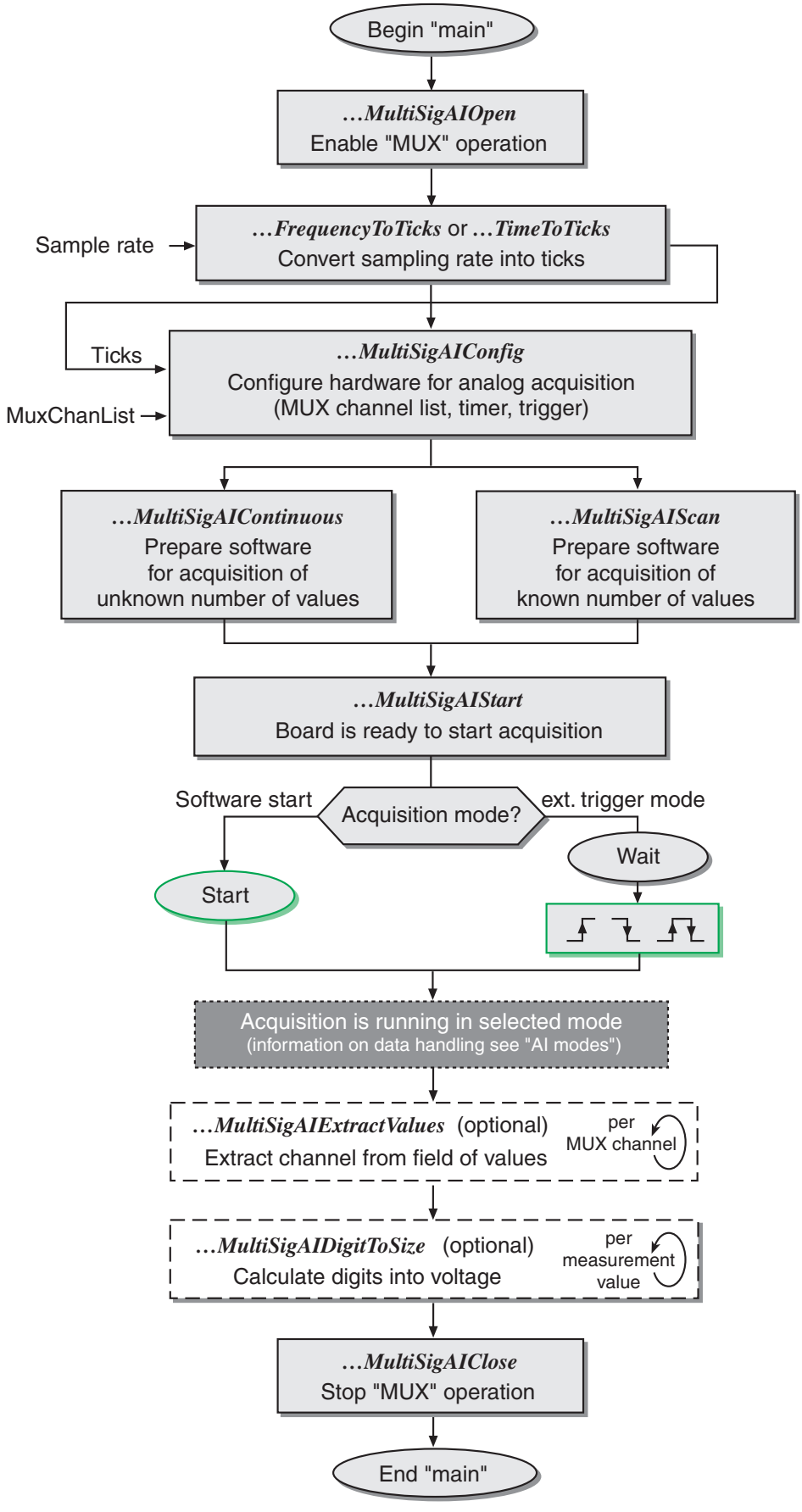



*Diagram 53: „MultiSig-AIContinuous“ and „MultiSig-AIScan“*

## 4.5.2 „Demux“ Operation

| Operation Mode | Usage | Trigger | Timing |
|---|---|---|---|
| MultiSig-AOSingle (see page 212) | 1 value to desired demux channel | Software start | – |
| MultiSig-AOContinuous (see page 214) | Output of values changing continuously | Software start, ext. digital | …MultiSig-AOConfig |
| MultiSig-AOWraparound (see page 223) | Output of values changing periodically | Software start, ext. digital | …MultiSig-AOConfig |

*Table 5: „Demux“ operation modes*

Note the following specifications:

- Digital port A is always used to control the demultiplexers.
- D/A channel 0 (pin 30) of the ME-4600 is always used for the output.

### 4.5.2.1 Operation Mode „MultiSig-AOSingle“

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | ✔ | ✔ | ✔ |

This mode of operation is used to output the desired voltage value on the selected channel of the „Demux“ system (0…31). The following diagram shows the program flow for this mode of operation:

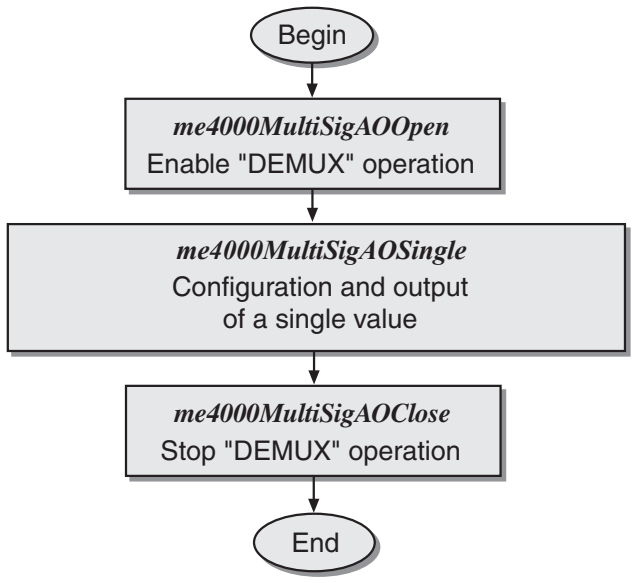



*Diagram 54: Programming „MultiSig-AOSingle“*

### 4.5.2.2 Timer Controlled „Demux“ Operation

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This section describes the timer controlled output operation by the ME-MultiSig system in conjunction with an ME-4600 series board. Use a base board of type ME-DEMUX32. Up to 32 channels can be „Demultiplexed“ in this mode of operation.

The following order of operations applies:

1. Open the „Demux“ operation mode with the function …*MultiSigAOOpen* (see page 217)
2. Create a user defined „Demux“ channel list to control the demux channels (0…31).
3. Configuration of the D/A section of the ME-4600 with the function …*MultiSigAOConfig* (see page 212)
4. Preparation of the software with the function …*MultiSig-AOContinuous* (see page 214) resp. …*MultiSigAOWrap-around* (see page 223)
5. Start the output with the function …*MultiSigAOStart* (see page 220)
6. Close the „Demux“ operation with the function …*MultiSig-AOClose* (see page 211)

Before the output begins, you must decide whether the values for output should be continuously reloaded („MultiSig-AOContinuous“) or whether the same values should be repeatedly output („MultiSig-AOWraparound“) until the process is stopped by the user. Once the hardware and software have been configured, the output can be started by software or by an external trigger signal.

The following diagram shows an overview of the process for the **„MultiSig-AOContinuous“** and **„MultiSig-AOWraparound“** modes for outputting the data values. The data handling follows the “normal” analog output operation:

- Data handling „AOContinuous“ see page 56.
- Data handling „AOWraparound“ see page 60.

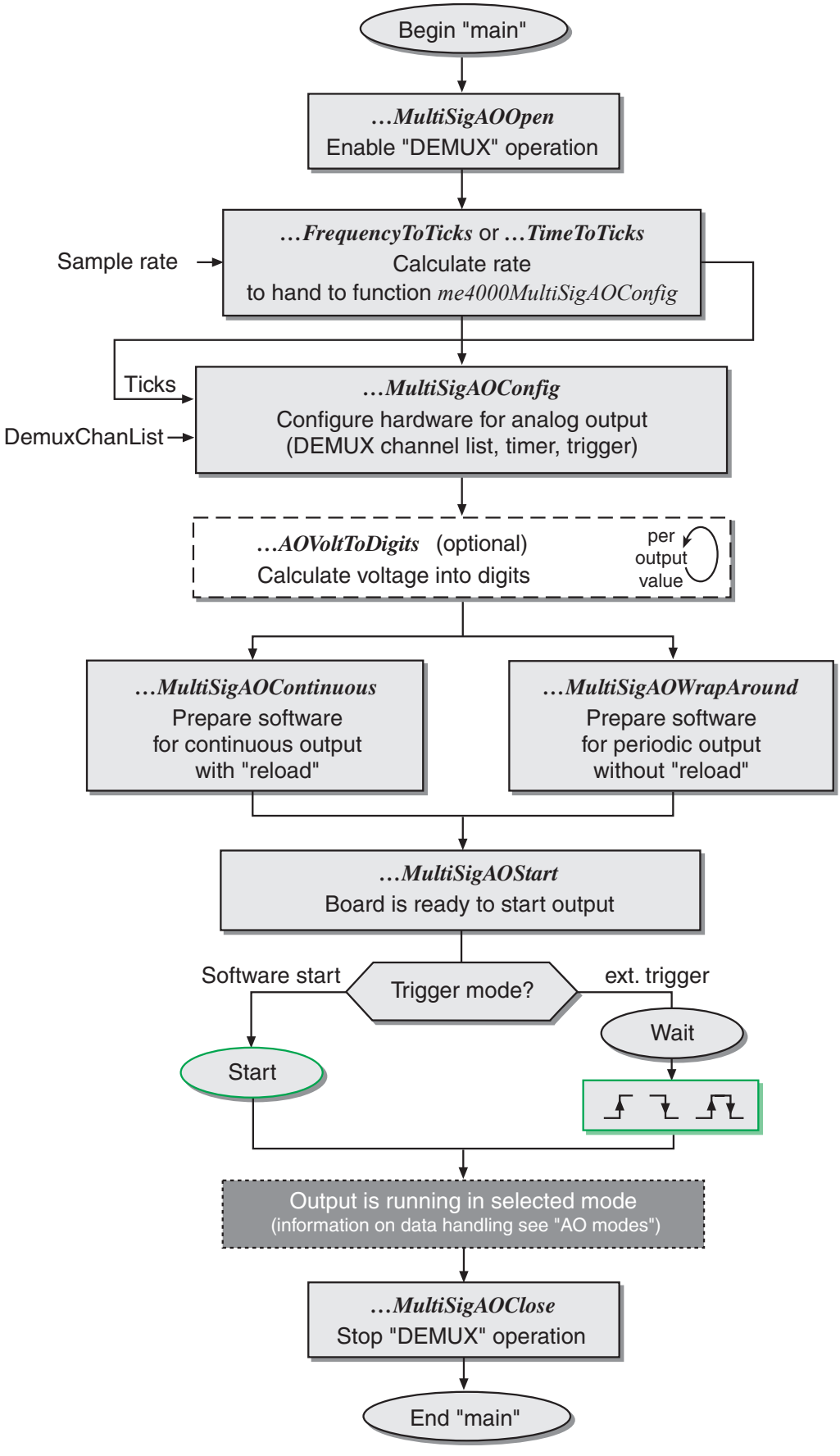



*Diagram 55: „MultiSig-AIContinuous“ and „MultiSig-AIScan“*

# 4.6 Driver Concept

**Important note:** The boards of the ME-4600 series (ME-4610/4650/4660/4670/4680) and the boards of type ME-6000 and ME-6100 are using a common system driver. The unique prefix used for file and function names is „me4000“.

The 32 bit driver was developed for the Windows driver architecture called „Windows Driver Model“ (WDM). The WDM architecture is implemented in Windows 98/Me/2000 and XP. For additional support for Windows NT4.0 a conventional kernel driver is available. The system driver consists of the following components:

- WDM driver (`me4000.sys`) for Windows 98/Me/2000/XP.
- Kernel driver (`me4000.sys`) for Windows NT.
- API-DLL (`me4000.dll`) for Visual C++ and Delphi. These API function start with the prefix *me4000…*
- API-DLL (`me4000Ex.dll`) for Agilent VEE, LabVIEW™ and VisualBasic.

To make it easy for you we provide simple demo programs and small projects with source code to help understanding of the functions and how to include them into your project. These demo programs can be found within the ME Software Developer Kit (ME-SDK), which is installed to the directory `C:Meilhaus\me-sdk` by default. Please read the notes in the appropriate README files.

## 4.6.1 Visual C++

| | | |
|---|---|---|
| **API-DLL** | me4000.dll | System driver |
| **Function prototypes** | me4000dll.h | ME-SDK |
| **Constant definitions** | me4000defs.h | ME-SDK |
| **Function prefix** | me4000… | |

*Table 6: Visual C++*

Visual C++ support for your board is included with the ME-SDK on the „ME-Power-CD“ or under www.meilhaus.de/download.

### 4.6.2 Visual Basic

| | | |
|---|---|---|
| **API-DLL** | me4000Ex.dll | System driver |
| **Function prototypes** | me4000.bas | ME-SDK |
| **Constant definitions** | me4000.bas | ME-SDK |
| **Function prefix** | me4000VB_… | |

*Table 7: Visual Basic*

Visual C++ support for your board is included with the ME-SDK on the „ME-Power-CD“ or under www.meilhaus.de/download.

**Important Notes:** Partly the function prototypes for Visual Basic differ in the number of parameters and the datatype of single parameters. Please note the file `me4000.bas`, included with the ME-SDK. Instead of the standard API `me4000.dll` you have to use the specific API `me4000Ex.dll`. „Missing“ parameters are marked with the symbol „**VB**“ in the function reference.

Because of the threading model was changed in Visual Basic 6.0, the usage of callback functions is not possible there. However it is possible in Visual Basic 5.0.

### 4.6.3 Delphi

| | | |
|---|---|---|
| **API-DLL** | me4000.dll | System driver |
| **Function prototypes** | me4000dll.pas | ME-SDK |
| **Constant definitions** | me4000defs.pas | ME-SDK |
| **Function prefix** | me4000… | |

*Table 8: Delphi*

Delphi support for your board is included with the ME-SDK on the „ME-Power-CD“ or under www.meilhaus.de/download.

### 4.6.4 Agilent VEE

| | | |
|---|---|---|
| **API-DLL** | me4000Ex.dll | System driver |
| **Function prototypes** | me4000VEE.h | VEE driver system |
| **Constant definitions** | me4000Defines.vee | VEE driver system |
| **Function prefix** | me4000VEE_… | |

*Table 9: Agilent VEE*

The Meilhaus VEE driver system is included with the „ME-Power-CD“ or under www.meilhaus.com/download.

For installation of VEE components and for further information please read the documentation included with the VEE driver system. For basics of VEE programming please use your VEE documentation and the VEE online help index.

**Important Notes:** Partly the function prototypes for VEE differ in the number of parameters and the datatype of single parameters. Please note the VEE header file `me4000VEE.h`, which is copied into the root directory of your VEE installation. Instead of the standard API `me4000.dll` you have to use the specific API `me4000Ex.dll`.

Because of VEE does not support callback functions the corresponding parameters are missing in the VEE-specific API (e. g. `<CallbackProc>`). „Missing“ parameters are marked with the symbol „**VEE**“ in the function reference.

The functions *me4000VEE_AIScan* and *me4000VEE_ MultiSigAIScan* automatically call the appropriate start function *…VEE_AIStart* resp. *…VEE_AIMultiSigStart* and return after ending the measurement.

### 4.6.5 LabVIEW

| | | |
|---|---|---|
| **API-DLL** | me4000Ex.dll | System driver |
| **Function prototypes** | me4000LV.h* | LabVIEW driver |
| **Constant definitions** | no central definition file | |
| **Function prefix** | me4000LV(50)_… | see notes in the text |

*Table 10: LabVIEW*

*The file `me4000LV.h` is only for documentation purposes.

The LabVIEW™driver for your board is included with the „ME-Power-CD“ or under www.meilhaus.com/download.

For installation of the LabVIEW components and for further information please read the documentation included with the appropriate LabVIEW driver. For basics of LabVIEW programming please use your LabVIEW documentation and the LabVIEW online help.

**Important Notes:** Partly the function prototypes for LabVIEW differ in the number of parameters and the datatype of single parameters. Please note the header file `me4000LV.h`, which is coming with the LabVIEW driver. The file is only for documentation purposes. Instead of the standard API `me4000.dll` you have to use the specific API `me4000Ex.dll`.

Because of LabVIEW does not support callback functions the corresponding parameters are missing in the LabVIEW-specific API (e. g. `<CallbackProc>`). „Missing“ parameters are marked with the symbol „**LV**“ in the function reference.

With older LabVIEW versions (Rev. 5.0 and older) the special functions *me4000LV50_AIScan* and *me4000LV50_MultiSigAIScan* must be used, which call the appropriate start function *me4000LV_AIStart* resp. *me4000LV_AIMultiSigStart* automatically. The „scan“ function returns after ending the measurement.

### 4.6.6 Python

Python is a text-based, interpreted (no compiler needed) and interactive (input by command line possible) programming language whose source code is free available. It facilitates procedural as well as object based programming.

Python allows easy and quick programming independent of the platform under Windows and Linux. You can write a program on Windows, copy it to a Linux system and run it at once with the interpreter there without compiling or editing the source code.

For the measurement technican Python provides a number of extension modules. Modules for programming of graphical user interfaces, for visualisation of measurement data and for numerical calculations belong to it.

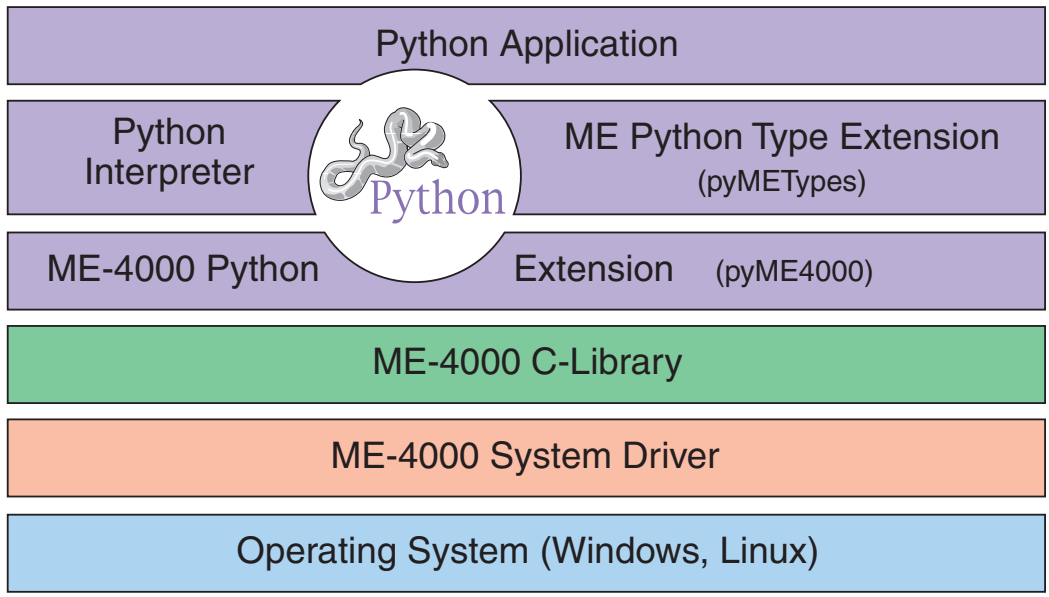



*Diagram 56: Python Programming*

The diagram shows the basic structure of the software system. The Python application on the top level looks at the programming interface consisting of the Python interpreter and the extension modules. The platform dependent part of the software architecture is completely hidden by the Python interpreter and the extension modules.

To use a board of type ME-46x0 or ME-6x00 (only under Windows) with Python along with the system driver and the function library a Python interpreter is required. It is available for free

download under http://www.python.org (already included with common Linux distributions). Additional the ME-4000 extension module is required including all functions and constants for Windows resp. Linux. Both are provided for free by Meilhaus Electronic under:

http://www.sourceforge.net/projects/meilhaus

There you find the packages „pyME4000“ and „pyMETypes“ as a source distribution for Linux and Windows. Beside the sources of the extension modules there are also examples and test programs as well as README files and installation notes included. Additionally installation programs under Windows are provided for the packages „pyME4000“ and „pyMETypes“ (Condition: valid Python installation).

**Note:** In the function names the prefix „me4000“ was leaved because of the import command for the ME-4000 extension modul automatically puts the characters „me4000.“ in front of the function name. Also note, that in Python (like in Linux) for all function groups the programming is opened by the function *…Open* and closed by the function *…Close.*

Example for a simple console program:

```
1 # Python
2 > import me4000
3 > me4000.DIOOpen(0)
4 > value = me4000.DIOGetByte(0, 0)
5 > me4000.DIOClose(0)
6 > print 'Value = 0x%X' % value
7 Value = 0xAA
```

# 5 Function Reference

## 5.1 General Notes

- **Function Prototypes:**
  In the following function description the generic function prototypes for Visual C++ are used. The definitions for other supported programming languages which are partly using different data types can be found in the appropriate definition files included with the ME-SDK resp. the header files of the LabVIEW resp. VEE driver (see also chap. 4.6 on page 87).

- **Parameter „BoardNumber“**
  When using a single board of a board family, the board number is always „0“ (interger value). In systems running several boards from the same board family the computer assigns the board number. Use this number to access the board. Determine the assignment of board numbers after installation of the boards.

  **Important note:** Concerning the software the boards of the ME-4600 series (currently: ME-4610/4650/4660/4670/4680) and the boards of type ME-6000/6100 belong to the same family and use a common driver. This means the boards „share“ the value range of the parameter `<BoardNumber>` from 0…31.

  **Tip:** Verify the assignment of „BoardNumber“ and serial number at the beginning of your program (see function *…GetSerialNumber*).

- **Execution Mode BLOCKING:**
  Note, when using a computer with low performance, slow sample rates or an external trigger which appears slowly or not it can result in a longer lasting blocking of the computer.

  **Note:** Agilent VEE and LabVIEW always use the execution mode (BLOCKING).

- **External Trigger with Time-Out:**
  For functions with external trigger you can define a time-out period within the **first** trigger pulse must occur. Else the operation will be cancelled (parameter `<TimeOutSeconds>`).
  E. g. In the acquisition modes „External Single Value“ and „External Channel List“ it is not checked if further trigger signals fail to appear. Note this when choosing the execution mode.

## 5.2 Naming Conventions

With the API functions of the ME-4000 function library all boards of type ME-4610/4650/4660/4670/4680 as well as the boards of type ME-6000/6100 can be accessed (if function is supported by the respective board). Uniformly the prefix "*me4000...*" will be used in the function names which consist of the prefix and several components representing the respective function as descriptive as possible. (e. g. "AO" for "Analog Output").

For Visual C++ and Delphi no language specific identification is given, e. g. *me4000AOConfig*. However for Agilent VEE the characters„ “*VEE_*“ (e. g. *me4000VEE_AOConfig*), for LabVIEW the characters „“*LV_*“ (e. g. *me4000LV_AOConfig*) and for Visual Basic the characters „*VB_*“ (e. g. *me4000VB_AOConfig*) are inserted.

For Visual C++ and Delphi no language specific identification is given, e. g. *me4000AOConfig*. However for Agilent VEE the characters„ “*VEE_*“ (e. g. *me4000VEE_AOConfig*), for LabVIEW the characters „“*LV_*“ (e. g. *me4000LV_AOConfig*) and for Visual Basic the characters „*VB_*“ (e. g. *me4000VB_AOConfig*) are inserted.

For the description of the functions, the following standards will be used:

| | |
|---|---|
| *function name* | will be italic in body text e. g. *me4000GetBoardVersion*. |
| `<parameters>` | will be in brackets as shown and in font `Courier`. |
| [square brackets] | will indicate physical units. |
| `main(...)` | parts of programs will be in `Courier` type |

# 5.3 Description of the API Functions

The functions will be described by functional groups as listed below. Within each functional group, the individual functions will be described in alphabetical order:



| Function | Short Description | Page |
|---|---|---|
| ***Error Handling*** | | |
| …ErrorGetMessage | Assign error string to a error number | 99 |
| …ErrorGetLastMessage | Assign error string to the last error occured | 100 |
| …ErrorSetDefaultProc | Install predefined global error routine for API | 101 |
| …ErrorSetUserProc | Install user defined global error routine for API | 102 |
| ***General Functions*** | | |
| …FrequencyToTicks | Convert frequency to ticks | 103 |
| …GetBoardVersion | Determine board version | 105 |
| …GetDLLVersion | Determine DLL version number | 106 |
| …GetDriverVersion | Determine driver version number | 106 |
| …GetSerialNumber | Determine serial number of the board | 107 |
| …TimeToTicks | Convert period to ticks | 108 |

*Table 11: Overview library functions*

| Function | Short Description | Page |
|---|---|---|
| ***Analog Input*** | | |
| …AIConfig | Configuration of A/D section | 110 |
| …AIContinuous | Acquisition of a unknown number of measurement values | 113 |
| …AIDigitToVolt | Conversion of the digit value to a voltage value | 115 |
| …AIExtractValues | Extract values for one channel from data buffer | 116 |
| …AIGetNewValues | Asynchronous reading of data | 118 |
| …AIGetStatus | State request for „AIScan“ | 119 |
| …AIMakeChannelListEntry | Create a channel list entry | 121 |
| …AIReset | Reset of a timer controlled acquisition | 122 |
| …AIScan | Acquisition of a known number of measurement values | 123 |
| …AISingle | Single value measurement | 126 |
| …AIStart | Starting a timer controlled acquisition | 128 |
| …AIStop | Stopping a timer controlled acquisition | 129 |
| ***Analog Output*** | | |
| …AOAppendNewValues | Reloading the output buffer | 130 |
| …AOConfig | Configuration of D/A section | 131 |
| …AOContinuous | Continuous output | 133 |
| …AOGetStatus | State request for „AOContinuous“ and „AOWraparound“ | 135 |
| …AOReset | Reset output channel | 136 |
| …AOSingle | Single value output | 137 |
| …AOSingleSimultaneous | Start of a simultaneous output in the operation mode „AOSimultaneous“ | 139 |
| …AOStart | Starting a timer controlled output | 142 |
| …AOStartSynchronous | Synchronous start in the operation modes „AOContinuous“ and „AOWraparound“ | 143 |
| …AOStop | Stopping the output | 146 |
| …AOVoltToDigit | Conversion of the voltage value to be output in a digit value | 147 |

*Table 11: Overview library functions*

| Function | Short Description | Page |
|---|---|---|
| …AOWaveGen | Simple function generator | 148 |
| …AOWraparound | Periodic output | 150 |
| ***Bit Pattern Output*** | | |
| …DIOBPAppendNewValues | Reload data butter | 153 |
| …DIOBPConfig | Configuring the hardware for bit pattern output | 154 |
| …DIOBPPortConfig | | 154 |
| …DIOBPContinuous | Continuous bit pattern output | 157 |
| …DIOBPGetStatus | State request for „BitPattern-Continuous“ and „BitPattern-Wraparound“ | 159 |
| …DIOBPReset | Reset bit pattern output | 162 |
| …DIOBPStart | Starting the bit pattern output | 162 |
| …DIOBPStop | Stopping bit pattern output | 164 |
| …DIOBPWraparound | Periodic bit pattern output | 165 |
| ***Digital-I/O Standard*** | | |
| …DIOConfig | Configuring the digital ports for standard digital I/O operation | 167 |
| …DIOGetBit | Getting one bit | 169 |
| …DIOGetByte | Getting a byte | 170 |
| …DIOResetAll | Reset the digital I/O section | 171 |
| …DIOSetBit | Setting one bit | 172 |
| …DIOSetByte | Setting a byte | 173 |
| ***Counter Functions*** | | |
| …CntPWMStart | Starting PWM output | 174 |
| …CntPWMStop | Ending the PWM output | 175 |
| …CntRead | Reading the counter state | 176 |
| …CntWrite | Configuring and starting a counter | 177 |
| ***Functions for external Interrupt*** | | |
| …ExtIrqDisable | Disable the external IRQ input | 178 |
| …ExtIrqEnable | Enable the external IRQ input | 179 |
| …ExtIrqGetCount | Determine the number of ext. IRQs | 180 |

*Table 11: Overview library functions*

| Function | Short Description | Page |
|---|---|---|
| ***MultiSig Functions*** | | |
| …MultiSigAddressLED | Controlling the Address LED | 182 |
| …MultiSigClose | Close the configuration mode | 183 |
| …MultiSigOpen | Open the configuration mode | 184 |
| …MultiSigReset | Reset all master and slave boards | 185 |
| …MultiSigSetGain | Gain factor per channel group | 186 |
| …MultiSigAIClose | Closing „Mux“ operation mode | 187 |
| …MultiSigAIConfig | Configuring the hardware for analog acquisition | 188 |
| …MultiSigAIContinuous | Acquisition of a unknown number of measurement values | 191 |
| …MultiSigAIDigitToSize | Conversion of the digit value to the appropriate physical size | 193 |
| …MultiSigAIExtractValues | Extract values for one channel from the data buffer | 196 |
| …MultiSigAIGetNewValues | Asynchronous reading of data | 198 |
| …MultiSigAIGetStatus | State request in „Mux“ operation | 199 |
| …MultiSigAIOpen | Opening „Mux“ operation mode | 201 |
| …MultiSigAIReset | Reset a timer controlled acquisition | 202 |
| …MultiSigAIScan | Acquisition of a known number of measurement values | 203 |
| …MultiSigAISingle | Single value measurement | 205 |
| …MultiSigAIStart | Starting a timer controlled acquisition | 208 |
| …MultiSigAIStop | Stopping a timer controlled acquisition | 209 |
| …MultiSigAOAppendNewValues | Reloading the output buffer | 210 |
| …MultiSigAOClose | Closing „Demux“ operation mode | 211 |
| …MultiSigAOConfig | Configuring the hardware for analog output | 212 |
| …MultiSigAOContinuous | Kontinuierliche Ausgabe | 214 |
| …MultiSigAOGetStatus | State request in „Demux“ operation | 216 |
| …MultiSigAOOpen | Opening „Demux“ operation mode | 217 |
| …MultiSigAOReset | Reset output channel | 218 |
| …MultiSigAOSingle | Single value outpu | 219 |
| …MultiSigAOStart | Starting a timer controlled output | 220 |

*Table 11: Overview library functions*

| Function | Short Description | Page |
|---|---|---|
| …MultiSigAOStop | Stopping the output | 221 |
| …MultiSigAOVoltToDigit | Conversion of the voltage value to be output to a digit value | 222 |
| …MultiSigAOWraparound | Periodic output | 223 |

*Table 11: Overview library functions*

## 5.3.1 Error Handling

### me4000ErrorGetMessage

#### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function can be used to determine the error text from an error number returned from the API functions.

#### ● Definitions

VC: me4000ErrorGetMessage(int iErrorCode, char* pcBuffer, unsigned int uiBufferSize);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

#### ➔ Parameters

**`<ErrorCode>`**

The error number caused by the API function.

**`<Buffer>`**

Pointer to the error description text.

**`<BufferSize>`**

Buffer size in bytes for the error description text (max. of 256 characters).

#### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000ErrorGetLastMessage

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function returns the last error caused by a „me4000…“ API function and retrieves the error description text.

### ● Definitions

VC: me4000ErrorGetLastMessage(char* pcBuffer, unsigned int uiBufferSize);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<Buffer>`**

Pointer to the error description text.

**`<BufferSize>`**

Buffer size in bytes for the error description text (max. of 256 characters).

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000ErrorSetDefaultProc

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function can be used to install a predefined global error routine for the entire API. This global error routine is automatically called if an API function call returns an error. The following information is returned in the form of a message box:

- Name of the function that returned an error
- Short error description
- Error code

### ☞ Note

Only one global error routine can be installed (…*ErrorSetDefaultProc* or …*ErrorSetUserProc*)

### ● Definitions

VC: me4000ErrorSetDefaultProc(int iDefaultProcStatus);
LV: me4000LV_… (see `me4000LV.h`)
VB: me4000VB_… (see `me4000.bas`)
VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<DefaultProcStatus>`**

- ME4000_ERROR_DEFAULT_PROC_ENABLE
  Installing the predefined error routine.
- ME4000_ERROR_DEFAULT_PROC_DISABLE
  Uninstall the predefined error routine.

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000ErrorSetUserProc

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function is used to install a global user defined error routine for the API. This function is automatically called when an API function returns an error. The function …*ErrorGetMessage* is used to assign an error description to the error code.

### ☞ Note

Only one global error routine can be installed (…*ErrorSetDefaultProc* or …*ErrorSetUserProc*)

### ● Definitions

Type definition for ME4000_P_ERROR_PROC:

```
typedef void (_stdcall * ME4000_P_ERROR_PROC)
(char* pcFunctionName, int iErrorCode)
```

VC: me4000ErrorSetUserProc(ME4000_P_ERROR_PROC pErrorProc);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<ErrorProc>`**

Pointer to an error routine. The name of the faulty function and the error code will be passed to the function installed there. Passing a NULL will uninstall a previously installed error routine.

### ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## 5.3.2 General Functions

### me4000FrequencyToTicks

#### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function is used to convert the desired frequency (in Hz) into a "Ticks" value that can be passed directly to the timer in the appropriate „*…Config*" function (depending on the operation being done). The range of valid values depends on which timer is being configured. If this parameter is outside of the allowable hardware range, the "*…Config*" function being called will return an error.

**Example:** The max. sample rate of the A/D section of 500kS/s is 66 Ticks (`ChanTicks`).

#### ☞ Note

The number of Ticks is calculated as follows:

**General**:
$$\frac{1}{\text{Frequency[Hz]} \cdot 30.30 \cdot 10^{-9}\text{s}} = \text{Ticks}$$

The period (time) can be set in steps of $30.\overline{30}$ ns within the allowable value range.

**Example:**

The number of Ticks passed to the <`ChanTicks`> parameter for the maximum sample rate of 500 kS/s:

$$\frac{1}{500000\text{Hz} \cdot 30.30 \cdot 10^{-9}\text{s}} = 66 \text{ Ticks (42Hex)}$$

**Note** that the parameter <`TicksHighPart`> is only required when the SCAN frequency < 0.0077Hz (see functions „*…AIConfig*" and „*…MultiSigAIConfig*"). SCAN frequencies up to approx. 0.00048 Hz can be set.

Programming examples are available in the ME-SDK.

## ● Definitions

VC: me4000FrequencyToTicks(double dRequiredFreq, unsigned long* pulTicksLowPart, unsigned long* pulTicksHighPart, double* pdAchievedFreq);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**`<RequiredFreq>`**

This is the required frequency in Hz. It is converted into a Ticks value sent to the timer. If a "0" is passed the parameters `<TicksLowPart>` and `<TicksHighPart>` are set to FFFFFFFFHex. The timer is then set to the minimum frequency.

**`<TicksLowPart>`**

Pointer to the calculated Ticks (lower 32 bits) for passing to the appropriate parameter of the „*…Config*" functions.

**`<TicksHighPart>`**

Pointer to the calculated Ticks (upper bits 32…35 of the 36 bit wide SCAN timer) for passing to the `<ScanTicksHigh>` parameter of the functions *…AIConfig* and *…MultiSigAIConfig*. This value is only required for SCAN frequencies <0.0077 Hz. Otherwise, this parameter always returns "0".

**`<AchievedFreq>`**

Pointer to a double value containing the actual calculated frequency in Hz when the function returns (the next highest frequency is always set). If the parameter is not required, the constant ME4000_POINTER_NOT_USED is passed to the function.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000GetBoardVersion

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function determines the board version for an installed ME-4600 series board.

### ● Definitions

VC: me4000GetBoardVersion(unsigned int uiBoardNumber, unsigned short* pusVersion);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<Version>`**

Pointer to the device ID. Possible values are:

| | | |
|---|---|---|
| 4650Hex: | ME-4650 | 16A/D, without counter |
| 4660Hex: | ME-4660 | 16 A/D, 2 D/A |
| 4661Hex: | ME-4660i | 16 A/D, 2 D/A, opto |
| 4662Hex: | ME-4660s | 16 A/D, 2 D/A, S&H |
| 4663Hex: | ME-4660is | 16 A/D, 2 D/A, opto, S&H |
| 4670Hex: | ME-4670 | 32A/D, 4 D/A |
| 4671Hex: | ME-4670i | 32A/D, 4 D/A, opto |
| 4672Hex: | ME-4670s | 32A/D, 4 D/A, S&H |
| 4673Hex: | ME-4670is | 32A/D, 4 D/A, opto, S&H |
| 4680Hex: | ME-4680 | 32A/D, 4 D/A-FIFO |
| 4681Hex: | ME-4680i | 32A/D, 4 D/A-FIFO, opto |
| 4682Hex: | ME-4680s | 32A/D, 4 D/A-FIFO, S&H |
| 4683Hex: | ME-4680is | 32A/D, 4 D/A-FIFO, opto, S&H |

### ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000GetDLLVersion

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

Determines the version number of the driver DLL.

### ● Definitions

VC: me4000GetDLLVersion(unsigned long* pulVersion);
LV: me4000LV_... (see `me4000LV.h`)
VB: me4000VB_... (see `me4000.bas`)
VEE: me4000VEE_... (see `me4000VEE.h`)

### ➔ Parameters

**`<Version>`**

Version number. The 32 bit value contains the main version (higher 16 bits) and the sub version (lower 16 bits). Example: 0x00020001 is the version 2.01

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000GetDriverVersion

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

Determines the version number of the ME-4600 driver.

### ● Definitions

VC: me4000GetDriverVersion(unsigned long* pulVersion);

### ➔ Parameters

**`<Version>`**

Pointer to a value containing the driver version (hexadecimal coded).

### ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000GetSerialNumber

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

Determines the serial number of the selected ME-4600 board.

### ● Definitions

VC: me4000GetSerialNumber(unsigned int uiBoardNumber, unsigned long* pulSerialNumber);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<SerialNumber>`**

Pointer to a value containing the serial number.

### ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000TimeToTicks

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function is used to convert the desired period (in seconds) into a "Ticks" value that is sent directly to the timer in the appropriate „*…Config*" function (depending on the operation being done). The range of valid values depends on which timer is being configured. If this parameter is outside of the allowable hardware range, the „*…Config*" function being called will return an error.

**Example**:
The min. CHAN time of 2µs results in 66 Ticks (`ChanTicks`)

### ☞ Note

The number of Ticks is calculated as follows:

**General**:
$$\frac{\text{Period[s]}}{30.30 \cdot 10^{-9}\text{s}} = \text{Ticks}$$

The period (time) can be set in steps of $30.\overline{30}$ ns within the allowable value range.

**Example:**

The number of Ticks passed to the <`ChanTicks`> parameter for the minimum period of 2 µs:

$$\frac{(2 \cdot 10^{-6})\text{s}}{30.30 \cdot 10^{-9}\text{s}} = 66 \text{ Ticks (42Hex)}$$

**Note** that the parameter <`TicksHighPart`> is only required when the SCAN time >130s (see function „*…AIConfig*" and „*…MultiSigAI-Config*"). SCAN times up to approx. 34 minutes can be set.

Programming examples are available in the ME-SDK.

### ● Definitions

VC: me4000TimeToTicks(double dRequiredTime, unsigned long* pulTicksLowPart, unsigned long* pulTicksHighPart, double* pdAchievedTime);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameter

**<RequiredTime>**

Period time (in seconds) to be converted to a Tick value. If a “0” is passed, the parameters <TicksLowPart> and <TicksHighPart> return a “0”.

**<TicksLowPart>**

Pointer to the calculated Ticks (lower 32 bits) for passing to the appropriate parameters of the „*…Config*“ functions.

**<TicksHighPart>**

Pointer to the calculated Ticks (upper bits 32…35 of the 36 bit wide SCAN timer) for passing to the <ScanTicksHigh> parameter of the functions *…AIConfig* and *…MultiSigAIConfig*. This value is only required for SCAN times >130s. Otherwise, this parameter always returns “0”.

**<AchievedTime>**

Pointer to a double value containing the actual calculated period time in seconds when the function returns (the next lowest period time is always set). If the parameter is not required, the constant ME4000_POINTER_NOT_USED is passed to the function.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## 5.3.3 Analog Input

**me4000AIConfig**

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function configures the hardware of the A/D section for a timer controlled data acquisition. It configures the timer, passes the channel list, determines the acquisition mode `<AcqMode>`, establishes the external trigger mode and edge, and switches the simultaneous acquisition mode on (optional).

The channel list must be created first using the function *…AIMakeChannelListEntry*.

After the function *…AIScan* or *…AIContinuous* was called, the data acquisition is always started with the function *…AIStart* either immediately (software start) or when an external trigger pulse is detected.

### ☞ Note

A 32 bit CHAN timer and a 36 bit SCAN timer are available for setting the data acquisition timing. The CHAN timer sets the sample rate within the channel list. The max. CHAN time is approx. 130s, the min. CHAN time is 2 µs. The SCAN timer sets the time between the sampling of the first channel list entry of two consecutive channel list processings. Setting the SCAN timer is optional. All timers use a base clock frequency of 33 MHz. This results in a period of 30.30 ns, which is the smallest time unit available and will be referred to as "1 Tick". The period is set as a multiple of a tick and passed to the parameters `<ChanTicks>`, `<ScanTicksLow>` and `<ScanTicksHigh>`. The `<ScanTicksHigh>` parameter is only required for SCAN times >130s.

The functions *…FrequencyToTicks* and *…TimeToTicks* (see page 103) can be used for convenient conversion of frequency resp. period into ticks for passing to the timers.

Programming examples are available in the ME-SDK.

## ● Definitions

VC: me4000AIConfig(unsigned int uiBoardNumber, unsigned char* pucChanList, unsigned int uiChanListCount, int iSDMode, int iSimultaneous, unsigned long ulReserved, unsigned long ulChanTicks, unsigned long ulScanTicksLow, unsigned long ulScanTicksHigh, int iAcqMode, int iExtTriggerMode, int iExtTriggerEdge);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<ChanList>**

Pointer to the beginning of the channel list (see function *…AI-MakeChannelListEntry)*

**<ChanListCount>**

Number of channel list entries.

**<SDMode>**

Single ended or differential measurement:

- ME4000_AI_INPUT_SINGLE_ENDED:
  Single ended measurement
- ME4000_AI_INPUT_DIFFERENTIAL:
  Differential measurement

**<Simultaneous>**

Set the simultaneous data acquisition on channels 0…7 on or off (only for "s" models). See also section 3.3.3 for more details.

- ME4000_AI_SIMULTANEOUS_DISABLE
  Simultaneous operation OFF (Standard)
- ME4000_AI_SIMULTANEOUS_ENABLE
  Simultaneous operation ON

**<Reserved>**

This parameter is reserved. Please pass "0".

**<ChanTicks>**

Number of Ticks for the CHAN timer (32 bit) which sets the sample rate. Possible values are 66 (42Hex) to $2^{32}$-1 (FFFFFFFFHex) Ticks.

**<ScanTicksLow>**

Number of Ticks for the lower part (bits 31…0) of the 36 bit SCAN timer. It determines the time between the sampling of the first channel list entry of two consecutive channel list processings. If the SCAN timer is not used the constant ME4000_VALUE_NOT_USED is passed to this parameter **and** the <ScanTicksHigh> parameter.

The minimum SCAN time is (see also diagram 25):
(number of channel list entries x CHAN time) + „x“ Ticks

The max. allowable SCAN time is 30 minutes, this is 59,400,000,000 Ticks (DD4841200Hex).

**<ScanTicksHigh>**

Number of Ticks for the higher part (bits 35…32) of the 36 bit SCAN timer. This timer is only required for scan times over 130.15s otherwise pass the constant ME4000_VALUE_NOT_USED.

**<AcqMode>**

Acquisition mode for timer controlled acquisition:

- ME4000_AI_ACQ_MODE_SOFTWARE
  The data acquisition is started by calling the function …*AIStart*. Data is acquired according to the timer settings (see diagram 25).
- ME4000_AI_ACQ_MODE_EXT
  The data acquisition is ready for starting after calling the …*AIStart* function. The actual data acquisition starts on the first external trigger pulse. Data is acquired according to the timer settings. Only the first trigger pulse has an effect, all others are ignored (see diagram 32).
- ME4000_AI_ACQ_MODE_EXT_SINGLE_VALUE
  The data acquisition is ready for starting after calling the …*AIStart* function. On every external trigger pulse exactly **one** value is acquired (according to the channel list). A time delay of one CHAN-time is between the first trigger pulse and the first conversion. Otherwise, the timer settings have not effect (see diagram 33).
- ME4000_AI_ACQ_MODE_EXT_SINGLE_CHANLIST
  The data acquisition is ready for starting after calling the …*AIStart* function. Every time a trigger pulse occurs, the channel list is processed once. Data is acquired according to the timer settings (see diagram 34). The SCAN timer has no effect.

**<ExtTriggerMode>**

Selects the external trigger source for the A/D-section. See also parameter <AcqMode>.

- ME4000_AI_TRIGGER_EXT_DIGITAL
  Trigger source is the digital trigger input.
- ME4000_AI_TRIGGER_EXT_ANALOG (not ME-4650/4660)
  Trigger source is the analog trigger unit.
- ME4000_VALUE_NOT_USED
  No external trigger in use.

**<ExtTriggerEdge>**

Selects the trigger edge for the analog or digital trigger input. See parameter <ExtTriggerMode>.

- ME4000_AI_TRIGGER_EXT_EDGE_RISING
  Starts the acquisition on a rising edge.
- ME4000_AI_TRIGGER_EXT_EDGE_FALLING
  Starts the acquisition on a falling edge.
- ME4000_AI_TRIGGER_EXT_EDGE_BOTH
  Starts the acquisition on a rising or falling edge.
- ME4000_VALUE_NOT_USED
  No external trigger in use.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000AIContinuous

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function is used to prepare the software for timer controlled acquisition when the number of measured values is not known before. This function is always executed asynchronously. The measured values are retrieved either with a user defined callback function or repeated calls of the function *…AIGetNewValues*.

The data acquisition is always started with the function …*AIStart*. It starts immediately (software start) or when an external trigger is detected (see …*AIConfig*). If working with an external trigger, and this does not appear, the "Time-Out" value can be used to cancel the data acquisition process. The data acquisition is ended by calling the functions …*AIStop* or …*AIReset*.

## ☞ Note

The flow of operation is described in chapter 4.1.3 "Timer Controlled „AI Operation Modes"" on page 34. Programming examples are available in the ME-SDK.

## ● Definitions

Type definition for ME4000_P_AI_CALLBACK_PROC:

```
typedef void (_stdcall *
ME4000_P_AI_CALLBACK_PROC)(short* psValues,
unsigned int uiNumberOfValues, void* pCall-
backContext, int iLastError);
```

VC: me4000AIContinuous(unsigned int uiBoardNumber, ME4000_P_AI_CALLBACK_PROC pCallbackProc, void* pCallbackContext, unsigned int uiRefreshFrequency, unsigned long ulTimeOutSeconds);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<CallbackProc>** **~~LV~~, ~~VB~~, ~~VEE~~**

This defines the callback function called in fixed intervals during the data acquisition. The function receives a pointer to the new values and the number of new values. If this operation is not required, pass the constant ME4000_POINTER_NOT_USED.

**<CallbackContext>** **~~LV~~, ~~VB~~, ~~VEE~~**

User defined pointer passed to the callback function. If no callback function is used, pass the constant ME4000_POINTER_NOT_USED.

**<RefreshFrequency>**

Defines the number of channel list processings after which the ring buffer should be read. The value passed here serves as a guideline and can be fitted by the driver. If the constant

ME4000_VALUE_NOT_USED is passed, the driver will determine a useful value automatically.

**<TimeOutSeconds>**

Time-out value in seconds. If no external trigger pulse was detected within the defined interval, the data acquisition process is automatically cancelled. If no external trigger is being used or no time-out value is used, pass the constant ME4000_VALUE_NOT_USED.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000AIDigitToVolt

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function allows a convenient conversion of the measured values from digits to volts. The input range used is taken into account by this function. The function can be used for converting single values or an entire array of values. Its use is optional.

The following formulas are used for conversion:

| | |
|---|---|
| ***Bipolar, ±10V:*** $U[Volt] = \frac{10V}{32768} \cdot U[Digits]$ | ***Bipolar, ±2.5V:*** $U[Volt] = \frac{2,5V}{32768} \cdot U[Digits]$ |
| ***Unipolar, 0...10V:*** $U[Volt] = \frac{5V}{32768} \cdot U[Digits] + 5V$ | ***Unipolar, 0...2.5V:*** $U[Volt] = \frac{1,25V}{32768} \cdot U[Digits] + 1,25V$ |

## ● Definitions

VC: me4000AIDigitToVolt(short sDigit, int iRange, double* pdVolt);

LV: me4000LV_... (see `me4000LV.h`)

VB: me4000VB_... (see `me4000.bas`)

VEE: me4000VEE_... (see `me4000VEE.h`)

## ➔ Parameters

**<Digit>**

The “Raw” data value as it appears in the buffer after the data acquisition.

**<Range>**

The input voltage range, possible values are:

- ME4000_AI_RANGE_BIPOLAR_10: ±10V
- ME4000_AI_RANGE_BIPOLAR_2_5: ±2.5V
- ME4000_AI_RANGE_UNIPOLAR_10: 0…10V
- ME4000_AI_RANGE_UNIPOLAR_2_5: 0…2.5V

**<Volt>**

Pointer to the calculated value in volts.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000AIExtractValues

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function is used to extract the values for a specific channel from the array of all measured values corresponding to the channel list. This function must be called separately for each channel whose values are to be extracted from the data buffer.

## ● Definitions

VC: me4000AIExtractValues(unsigned int uiChannelNumber, short* psAIBuffer, unsigned long ulAIDataCount, unsigned char* pucChanList, unsigned int uiChanListCount, short* psChanBuffer, unsigned long ulChanBufferSizeValues, unsigned long* pulChanDataCount);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<ChannelNumber>**

The A/D channel number whose measured values are to be extracted from the data buffer; possible values: 0…31; (ME-4650/4660: 0…15 single ended)

**<AIBuffer>**

Pointer to the data buffer containing all measured values.

**<AIDataCount>**

Number of measured values in the <AIBuffer> data buffer.

**<ChanList>**

Pointer to the channel list which was generated with the function *…AIMakeChannelListEntry* and passed to the function *…AIConfig*.

**<ChanListCount>**

Number of channel list entries (ChanList).

**<ChanBuffer>**

Pointer to a data buffer where the extracted values for the selected channel are stored.

**<ChanBufferSizeValues>**

Size of the array passed in the <ChanBuffer> parameter above.

**<ChanDataCount>**

Pointer to a value containing the actual number of values stored in the array ChanBuffer. This value will never be larger than the <ChanBufferSizeValues> parameter, but could be smaller.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000AIGetNewValues

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function is used to „retrieve“ the measured values and is used in the „AIContinuous“ operation mode. In the “AIScan” operation mode this function can be used to „look at“ the measured values during a data acquisition process that is running as a background operation (asynchronous). A typical use for this function would be to read in and display some measured values during a longer acquisition process.

### ☞ Note

An example of the process flow can be found in the “Programming” section on page 39 and in the example programs provided in the ME-SDK.

### ● Definitions

VC: me4000AIGetNewValues(unsigned int uiBoardNumber, short* psBuffer, unsigned long ulNumberOfValuesToRead, int iExecutionMode, unsigned long* pulNumberOfValuesRead, int* piLastError);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<Buffer>`**

Pointer to a data buffer where the newest data values are stored from the running data acquisition process. The function *…AIDigitToVolt* can be used to convert the raw data into voltage values.

**`<NumberOfValuesToRead>`**

Size of the data buffer given as the number of measured values. The size should be a multiple of the channel list size. If you pass the value „0“ here the parameter `<NumberOfValuesRead>` returns the number of values to be „retrieved“.

**<ExecutionMode>**

Choose the execution mode for this function:

- ME4000_AI_GET_NEW_VALUES_BLOCKING:
  The program flow is blocked until all measured values are retrieved.
- ME4000_AI_GET_NEW_VALUES_NON_BLOCKING:
  Only the currently available measured values are retrieved.

If you passed the value „0“ in parameter `<NumberOfValuesToRead>` this parameter is not relevant.

**<NumberOfValuesRead>**

Pointer which returns the actual number of measured values stored into the `buffer` defined above. This value will never be greater than the `<NumberOfValuesToRead>` parameter, but it can be smaller if not as many measured values are available.

**<LastError>**

This parameter contains the last error which occurred when calling this function. Possible errors are a FIFO overflow, or data buffer overflow. If no error occurred, a “0” (ME4000_NO_ERROR) is returned.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000AIGetStatus

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function is used to check whether a data acquisition process in the operation mode „AIScan“ in the execution mode “ASYNCHRONOUS” is still running or whether all the measured values have been retrieved.

The parameter `<WaitIdle>` can be used to return the status immediately or wait until the data acquisition process is ended.

## ● Definitions

VC: me4000AIGetStatus(unsigned int uiBoardNumber, int iWaitIdle, int* piStatus);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<WaitIdle>`**

This parameter defines how the function returns:

- ME4000_AI_WAIT_NONE
  The status of the operation is returned immediately in the parameter `<Status>`.
- ME4000_AI_WAIT_IDLE
  The function returns when all measured values have been acquired. In that case, the parameter `<Status>` will always have the value ME4000_AI_STATUS_IDLE.

**`<Status>`**

The current status of the data acquisition:

- ME4000_AI_STATUS_IDLE
  The acquisition is ended.
- ME4000_AI_STATUS_BUSY
  The acquisition is still running.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000AIMakeChannelListEntry

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function generates a channel list entry based on the parameters `<ChannelNumber>` and `<Range>` and writes it into an array that is later passed to the function …*AIConfig*. This function must be called separately for each channel list entry.

### ☞ Note

Uni-polar input ranges cannot be combined with the differential mode of operation!

### ● Definitions

VC: me4000AIMakeChannelListEntry(unsigned int uiChannelNumber, int iRange, unsigned char* pucChanListEntry);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<ChannelNumber>`**

A/D channel number. Possible values are in single ended mode: 0…31; in differential mode: 0…15 (ME-4650/4660: 0…15 single ended)

**`<Range>`**

Selection of the input voltage range:

- ME4000_AI_RANGE_BIPOLAR_10: ±10V
- ME4000_AI_RANGE_BIPOLAR_2_5: ±2.5V
- ME4000_AI_RANGE_UNIPOLAR_10: 0…10V
- ME4000_AI_RANGE_UNIPOLAR_2_5: 0…2.5V

**`<ChanListEntry>`**

Pointer to a single value of an array of type “unsigned char” where the channel list entry is stored. The array must be allocated first. In the function …*AIConfig* a pointer to this array must be passed in the parameter `<ChanList>`.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000AIReset

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

The data acquisition process is stopped completely when this function is called. All measured values acquired up until this function is called are lost. The A/D section must be configured again before another data acquisition process can be started (*…AIConfig*, *…AIScan*, *…AIContinuous*).

## ● Definitions

VC: me4000AIReset(unsigned int uiBoardNumber);LV: me4000LV_… (see `me4000LV.h`)

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000AIScan

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function is used to prepare the software for a data acquisition process with a known number of measured values. A user defined data buffer is allocated to which the measured values are stored. In the execution mode “BLOCKING”, the thread where the function …*AIStart* was called does not return until the last measured value is retrieved. In the execution mode „ASYNCHRONOUS“ the process is started as a background operation. A new thread is created when the …*AIStart* function is called. Other threads can be processed parallel to the running data acquisition. If desired (e. g. during a longer running data acquisition process) you can „look at“ the measured values during a running process. This can be done by a user defined callback function or by calling the function …*AIGetNewValues*. When using the “Terminate” callback function it is possible to “report” the end of the data acquisition process to your application.

The function …*AIStart* is always used to start the data acquisition process. It can be started immediately after the function is called (software start) or by an external trigger pulse (see …*AIConfig*). It is possible to set a “Time-Out” when an external trigger is being used in case this does not occur. The data acquisition process is ended automatically when the last data value is acquired.

### ☞ Note

An example of the process flow can be found in chapter 4.1.3 "Timer Controlled „AI Operation Modes“" on page 34 and in the example programs provided in the ME-SDK.

### ● Definitions

Type definition for ME4000_P_AI_CALLBACK_PROC:

```
typedef void (_stdcall *
ME4000_P_AI_CALLBACK_PROC)(short* psValues,
unsigned int uiNumberOfValues, void* pCall-
backContext, int iLastError);
```

Type definition for ME4000_P_AI_TERMINATE_PROC:

```
typedef void (_stdcall *
    ME4000_P_AI_TERMINATE_PROC)(short*psValues,
    unsigned int uiNumberOfValues, void*
    pTerminateContext, int iLastError);
```

VC: me4000AIScan(unsigned int uiBoardNumber, unsigned int uiNumberOfChanLists, short* psBuffer, unsigned long ulBufferSizeValues, int iExecutionMode, ME4000_P_AI_CALLBACK_PROC pCallbackProc, void* pCallbackContext, unsigned int uiRefreshFrequency, ME4000_P_AI_TERMINATE_PROC pTerminateProc, void* pTerminateContext, unsigned long ulTimeOutSeconds);

LV50: me4000LV50_… (siehe `me4000LV.h`)

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<NumberOfChanLists>**

Number of channel list processings.

**<Buffer>**

Pointer to a data buffer where the data values are stored. The function …*AIDigitToVolt* can be used to conveniently convert the measured values into a voltage value.

**<BufferSizeValues>**

The size of the data buffer given as the number of measured values.

**<ExecutionMode>**

Choose the execution mode for this function:

- ME4000_AI_SCAN_BLOCKING:
  The program is blocked until all measured values are acquired.
- ME4000_AI_SCAN_ASYNCHRONOUS:
  The following call of the function …*AIStart* will automatically create a new thread so that the calling thread is not blocked.

**<CallbackProc>** **~~LV~~, ~~VB~~, ~~VEE~~**

This is the callback function that is regularly called during the data acquisition process. This function receives a pointer to the newly acquired data values and the number of data values. If this operation is not required, pass the constant ME4000_POINTER_NOT_USED.

**<CallbackContext>** **~~LV~~, ~~VB~~, ~~VEE~~**

This is a user defined pointer that is passed to the callback function. If no callback function is being used, pass the constant ME4000_POINTER_NOT_USED.

**<RefreshFrequency>**

Defines the number of channel list processings after which the ring buffer should be read. The value passed here serves as a guideline and can be fitted by the driver. If the constant ME4000_VALUE_NOT_USED is passed, the driver will determine a useful value automatically.

**<TerminateProc>** **~~LV~~, ~~VB~~, ~~VEE~~**

This is the callback function that is executed when the data acquisition process is completed. The function is passed a pointer to the start of the data buffer and the number of values. If this operation is not required, pass the constant ME4000_POINTER_NOT_USED.

**<TerminateContext>** **~~LV~~, ~~VB~~, ~~VEE~~**

A user defined pointer that is passed to the “Terminate” function. If the „Terminate“ function is not being used pass the constant ME4000_POINTER_NOT_USED.

**<TimeOutSeconds>**

Time out value in seconds. If no external trigger was detected within the defined interval, the data acquisition process is automatically cancelled. If no external trigger is being used, or no time out is required, pass the constant ME4000_VALUE_NOT_USED.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000AISingle

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function is used to acquire a single value. The conversion start can be either by software or by an external trigger. No further functions for configuration and start are required.

### ☞ Note

If using differential operation, only bi-polar input ranges can be used! This function is always run in "Blocking" mode, therefore the program flow is blocked until the function returns. This is normally only relevant when using an external trigger signal to start the acquisition.

### ● Definitions

VC: me4000AISingle(unsigned int uiBoardNumber, unsigned int uiChannelNumber, int iRange, int iSDMode, int iTriggerMode, int iExtTriggerEdge, unsigned long ulTimeOutSeconds, short* psDigitalValue);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<ChannelNumber>**

A/D channel number; possible values in single ended mode: 0…31; in differential mode 0…15; (ME-4650/4660: 0…15 single ended)

**<Range>**

Choose the input voltage range:

- ME4000_AI_RANGE_BIPOLAR_10: ±10V
- ME4000_AI_RANGE_BIPOLAR_2_5: ±2.5V
- ME4000_AI_RANGE_UNIPOLAR_10: 0…10V
- ME4000_AI_RANGE_UNIPOLAR_2_5: 0…2.5V

**<SDMode>**

Single ended or differential measurement:

- ME4000_AI_INPUT_SINGLE_ENDED:
  Single ended measurement
- ME4000_AI_INPUT_DIFFERENTIAL:
  Differential measurement

**<TriggerMode>**

Trigger event for the A/D-section:

- ME4000_AI_TRIGGER_SOFTWARE
  The conversion is done immediately after this function is called.
- ME4000_AI_TRIGGER_EXT_DIGITAL
  The process is ready for conversion after this function is called. The conversion is started by a digital trigger signal.
- ME4000_AI_TRIGGER_EXT_ANALOG (not ME-4650/4660)
  The process is ready for conversion after this function is called. The conversion is started by an analog trigger signal.

**<ExtTriggerEdge>**

Selects the trigger edge for the external A/D trigger input (analog or digital).

- ME4000_AI_TRIGGER_EXT_EDGE_RISING
  Start when a rising edge is detected.
- ME4000_AI_TRIGGER_EXT_EDGE_FALLING
  Start when a falling edge is detected.
- ME4000_AI_TRIGGER_EXT_EDGE_BOTH
  Start when a falling or rising edge is detected.
- ME4000_VALUE_NOT_USED
  No external trigger in use. See parameter <TriggerMode>.

**<TimeOutSeconds>**

Time-out value in seconds. If no external trigger was detected within this interval, the data acquisition process is automatically cancelled. If no external trigger is being used, or no time-out is required, pass the constant ME4000_VALUE_NOT_USED.

**<DigitalValue>**

Pointer to a signed linearized 16 bit value. The function *…AI-DigitToVolt* can be used to convert the digit value to volts.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000AIStart

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function is used to „arm“ the data acquisition process. Depending on the configuration of the hardware and software, the process may start immediately after calling this function (software start) or it will depend on the selected external trigger event (see chapter 3.3.4 on page 19).

As long as not ended with the function …*AIReset* the data acquisition process can be restarted from the beginning by calling this function again. The A/D section does not need to be configured.

When ending a process in the operation mode „AIContinuous“ or for ending a process started with the “AIScan” function before it is completed, the functions …*AIStop* and …*AIReset* can be used.

### ☞ Note

If the A/D section is already active when …*AIStart* is called, an error message is returned.

**Behaviour of return depending on the trigger mode:**

- Software start: immediately
- ext. trigger without time-out: immediately
- ext. trigger with time-out: when the time-out expired or the external trigger signal occured.

### ● Definitions

VC: me4000AIStart(unsigned int uiBoardNumber);
LV: me4000LV_… (see `me4000LV.h`)
VB: me4000VB_… (see `me4000.bas`)
VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000AIStop

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

The data acquisition is stopped immediately. All measured values acquired since the last "retrieval" are lost. The A/D section configuration is kept (channel list, timer settings, etc). A new data acquisition process can be started with the function …*AIStart* at any time.

### ● Definitions

VC: me4000AIStop(unsigned int uiBoardNumber, int iReserved);
LV: me4000LV_… (see `me4000LV.h`)
VB: me4000VB_… (see `me4000.bas`)
VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<Reserved>`**

This parameter is reserved. Please pass the value „0".

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## 5.3.4 Analog Output

### me4000AOAppendNewValues

#### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used for continuously reloading the D/A FIFO during a running data output process. The process is stopped immediately with the functions *…AOStop* or *…AOReset*.

#### ☞ Note

It is not required that the same data buffer passed in the function *…AOContinuous* must be used.

#### ● Definitions

VC: me4000AOAppendNewValues(unsigned int uiBoardNumber, unsigned int uiChannelNumber, short* psBuffer, unsigned long ulNumberOfValuesToAppend, int iExecutionMode, unsigned long* pulNumberOfValuesAppended);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

#### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<ChannelNumber>`**

D/A channel number 0…3

**`<Buffer>`**

Pointer to the data buffer with the values to be reloaded.

**`<NumberOfValuesToAppend>`**

Number of values in the data buffer. If you pass the value „0“ here the parameter `<NumberOfValuesAppended>` returns the number of values, for which is currently space in the data buffer.

**<ExecutionMode>**

Choose execution mode for this function:

- ME4000_AO_APPEND_NEW_VALUES_BLOCKING:
  The program is blocked until all values are written to the ring buffer.
- ME4000_AO_APPEND_NEW_VALUES_NON_BLOCKING:
  The program only "refills" the number of values which have currently room in the ring buffer.

If you passed the value „0" in parameter `<NumberOfValuesToAppend>` this parameter is not relevant.

**<NumberOfValuesAppended>**

The actual number of values written to the ring buffer. See also parameter `<NumberOfValuesToAppend>`.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000AOConfig

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function configures the hardware of the D/A section for a timer controlled data output in the operation modes „AOContinuous" and „AOWraparound". The output operation is either started by calling the function *…AOStart* (for a single channel) or by the function *…AOStartSynchronous* (synchronous start of multiple channels). You can choose between software start or by an external trigger pulse.

A 32 bit counter serves as the time base. It is supplied by a clock frequency of 33 MHz which gives a period of 30.30ns as the smallest possible time unit. This is referred to as "1 Tick" in the following sections. The sample rate for the analog output must be a multiple of "1 Tick" and is passed in the parameter `<Ticks>`. The sample rate can therefore be set in increments of 30.30ns between the minimum and maximum sample rate. The minimum sample rate is 0.5 samples per minute. The maximum sample rate will depend on the mode of operation and the PC performance an can be up to 500 kS/s per channel.

See also the chapter "Programming" from page 22 on and the sample programs provided in the ME-SDK.

## ☞ Note

The functions …*FrequencyToTicks* and …*TimeToTicks* offer a convenient conversion of frequency resp. period time to "Ticks" which can be passed to the D/A timer (see page 103).

If the D/A section is already active when this function is called, an error message is returned.

## ● Definitions

VC: me4000AOConfig(unsigned int uiBoardNumber, unsigned int uiChannelNumber, unsigned long ulTicks, int iTriggerMode, int iExtTriggerEdge);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<ChannelNumber>`**

D/A channel number 0…3

**`<Ticks>`**

Number of Ticks for the 32 bit D/A timer. This sets the sample rate for the timer controlled data output. The value range is from 66 (42Hex) to $2^{32}$-1 (FFFFFFFF Hex) Ticks.

**`<TriggerMode>`**

Trigger event to start the analog output (if using synchronous start the settings in …*AOStartSynchronous* are valid):

- ME4000_AO_TRIGGER_SOFTWARE
  Start by software after calling the function …*AOStart*.
- ME4000_AO_TRIGGER_EXT_DIGITAL
  Ready to run after calling the function …*AOStart*. The output is started when the external trigger signal is detected.

**<ExtTriggerEdge>**

Selects the trigger edge for the corresponding trigger input DA_TRIG_*x* (if using synchronous start the settings in *…AOStart-Synchronous* are valid):

- ME4000_AO_TRIGGER_EXT_EDGE_RISING
  Starts the output on a rising edge.
- ME4000_AO_TRIGGER_EXT_EDGE_FALLING
  Starts the output on a falling edge.
- ME4000_AO_TRIGGER_EXT_EDGE_BOTH
  Starts the output on a rising or falling edge.
- ME4000_VALUE_NOT_USED
  No external trigger in use. See parameter <TriggerMode>.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000AOContinuous

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to prepare the operation mode “AOContinuous”. Any signal shapes can be output and the signals can also be changed while the output process is running (opposite to the operation mode “AOWraparound”). The D/A timer provides a fixed time base (sample rate) for outputting the single values (see function *…AOConfig*). A data buffer must be allocated for each channel being used and loaded with the initial values for output. Use the function *…AOAppendNewValues* for continuously reloading new data values. This can be done with or without a callback function.

The output is started by calling the function *…AOStart(Synchronous)*. either immediately (software start) or when an external trigger pulse is detected (see function *…AOConfig*). The function *…AOStop* is used to end the data output process at once. If the operation mode was not changed, a new output process can be started again from the beginning by calling the function *…AOStart(Synchronous)*. If the *…AOReset* function is used to end the data output process, the FIFO is also cleared and the process is completely terminated.

See also chapter 4.2 on page 51 and the sample programs provided in the ME-SDK.

## ☞ Note

If the selected D/A channel is already active when this function is called, an error message is returned.

## ● Definitions

Type definition for ME4000_P_AO_CALLBACK_PROC:

```
typedef void (_stdcall *
ME4000_P_AO_CALLBACK_PROC)
(unsigned long ulBufferAvailable,
void* pCallbackContext);
```

VC: me4000AOContinuous(unsigned int uiBoardNumber, unsigned int uiChannelNumber, short* psBuffer, unsigned long ulDataCount, ME4000_P_AO_CALLBACK_PROC pCallbackProc, void* pCallbackContext, unsigned long ulTimeOutSeconds, unsigned long* pulNumberOfValuesWritten);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<ChannelNumber>**

D/A channel number 0…3

**<Buffer>**

Pointer to a user defined data buffer, which is „filled“ with the **initial** values to be output.

**<DataCount>**

Number of values in data buffer

**<CallbackProc>** ~~**LV**~~**, ~~VB~~, ~~VEE~~**

Callback function which is called at regular intervals to reload the data buffer. If this operation is not required, pass the constant ME4000_POINTER_NOT_USED.

**<CallbackContext>** ~~**LV**~~**, ~~VB~~, ~~VEE~~**

This is a user defined pointer that can be passed to the callback function. If no callback function is being used, pass the constant ME4000_POINTER_NOT_USED.

**<TimeOutSeconds>**

Optional you can choose a time interval in seconds which specifies the time within the first trigger pulse must be detected. Otherwise the operation is cancelled. If you don´t want to use external trigger operation or no time-out value is set, pass the constant ME4000_VALUE_NOT_USED.

When using „synchronous start“ the time-out value in the function …*AOStartSynchronous* is valid!

**<NumberOfValuesWritten>**

The number of values actual written into the data buffer.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000AOGetStatus

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to check whether a data output operation in the „AOContinuous“ or „AOWraparound“ modes is still running or whether the FIFO has “run dry”. This can happen when the FIFO is intentionally allowed to run dry by not reloading any new data values to end the output process or when the values could not reloaded quickly enough because the PC performance is too slow.

The parameter `<WaitIdle>` can control whether the function returns the status immediately or whether it waits until the output process is ended.

## ● Definitions

VC: me4000AOGetStatus(unsigned int uiBoardNumber, unsigned int uiChannelNumber, int iWaitIdle, int* piStatus);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<ChannelNumber>**

D/A channel number 0…3

**<WaitIdle>**

This parameter defines the “return behaviour“ of this function:

- ME4000_AO_WAIT_NONE
  The current status is returned immediately in the parameter <Status>.
- ME4000_AO_WAIT_IDLE
  The function returns when the data output process is ended (FIFO is empty). The parameter <Status> always returns the constant ME4000_AO_STATUS_IDLE.

**<Status>**

The current status:

- ME4000_AO_STATUS_IDLE
  The output is ended, i. e. the FIFO is empty.
- ME4000_AO_STATUS_BUSY
  The output is still running.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000AOReset

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function ends the currently running „AOContinuous“ or „AO-Wraparound“ output process for the selected channel number. The output is stopped completely and immediately. The corresponding ring buffer is deleted and the channel output is set to 0V.

## ● Definitions

VC: me4000AOReset(unsigned int uiBoardNumber, unsigned int uiChannelNumber);

LV: me4000LV_... (see `me4000LV.h`)

VB: me4000VB_... (see `me4000.bas`)

VEE: me4000VEE_... (see `me4000VEE.h`)

## ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<ChannelNumber>`**

- Resets the selected channel. The D/A channel number 0…3 is passed in this parameter.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000AOSingle

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | ✔ | ✔ | ✔ |

This function serves for the „transparent“ output of a voltage value to a single channel.

To output a single value („AOSingle“), no further configuration with other functions is required.

For simultaneous output to multiple channels (operation mode „AO-Simultaneous“) please use the function *…AOSingleSimultaneous*.

## ☞ Note

If the selected D/A channel is already active when this function is called, an error message is returned.

**Behaviour of return depending on the trigger mode:**

- Software start: immediately
- ext. trigger without time-out: immediately
- ext. trigger with time-out: when the time-out expired or the external trigger signal occured.

## ● Definitions

VC: me4000AOSingle(unsigned int uiBoardNumber, unsigned int uiChannelNumber, int iTriggerMode, int iExtTriggerEdge, unsigned long ulTimeOutSeconds, short sValue);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<ChannelNumber>`**

D/A channel number 0…3

**`<TriggerMode>`**

Trigger event for the selected D/A-channel:

- ME4000_AO_TRIGGER_SOFTWARE
  Output a single value immediatly after calling this function (software start).
- ME4000_AO_TRIGGER_EXT_DIGITAL
  Ready to run after calling this function. The output starts when an external trigger signal on the appropriate trigger line DA_TRIG_*x* was detected.

**`<ExtTriggerEdge>`**

Selects the trigger edge for the corresponding trigger input (DA_TRIG_*x*):

- ME4000_AO_TRIGGER_EXT_EDGE_RISING
  Starts on a rising trigger edge.
- ME4000_AO_TRIGGER_EXT_EDGE_FALLING
  Starts on a falling trigger edge.
- ME4000_AO_TRIGGER_EXT_EDGE_BOTH
  Starts on a rising or falling trigger edge.
- ME4000_VALUE_NOT_USED
  No external trigger in use. See parameter `<TriggerMode>`.

**`<TimeOutSeconds>`**

Optional you can choose a time interval in seconds which specifies the time within the first trigger pulse must be detected. Otherwise the operation is cancelled. If you don´t want to use external trigger operation or no time-out value is set, pass the constant ME4000_VALUE_NOT_USED.

**<Value>**

16 bit output value. The value range is between -32768 (-10V) and +32767 (+10V - LSB)

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000AOSingleSimultaneous

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | ✔ | ✔ | ✔ |

Function for simultaneous output to multiple D/A channels (operation mode „AO-Simultaneous“). The array `<ChannelNumber>` allows you a targeted selection of the channels to be involved in the simultaneous output. You can choose what trigger input (resp. inputs) of the simultaneous channels should start the output.

## ☞ Note

**Behaviour of return depending on the trigger mode:**

- Software start: immediately
- ext. trigger without time-out: immediately
- ext. trigger with time-out: when the time-out expired or the external trigger signal occured.

To start the synchronous output in the timer controlled operation modes „AOContinuous“ and „AOWraparound“ please use the function …*AOStartSynchronous*.

## ● Definitions

VC: me4000AOSingleSimultaneous(unsigned int uiBoardNumber, unsigned int *puiChannelNumber, unsigned long ulCount, int iTriggerMode, int* piExtTriggerEnable, int* piExtTriggerEdge, unsigned long ulTimeOutSeconds, short* psValue);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<ChannelNumber>**

Array which contains the channel numbers of the channels to be output simultaneously.

**<Count>**

Count of channels listed in the array <ChannelNumber>. Also valid for the parameter <ExtTriggerEnable>, <ExtTriggerEdge> and <Value>.

**<TriggerMode>**

Trigger event for the simultaneous output of the channels listed in array <ChannelNumber>:

- ME4000_AO_TRIGGER_SOFTWARE
  Simultaneous output after calling this function (software start).
- ME4000_AO_TRIGGER_EXT_DIGITAL
  Ready to run after calling this function. The selected channels are updated simultaneously when the external trigger signal is detected (see also the following parameters).

**<ExtTriggerEnable>**

Array to enable a single or multiple trigger inputs (DA_TRIG_*x*). The order of entries corresponds with that one in the array <ChannelNumber>. The output is done when the first matching edge at one of the enabled trigger inputs is detected.

- ME4000_AO_TRIGGER_EXT_DISABLE
  Corresponding trigger input is disabled.
- ME4000_AO_TRIGGER_EXT_ENABLE
  Corresponding trigger input is enabled.

If you use the constant ME4000_AO_TRIGGER_SOFTWARE in parameter <TriggerMode> please pass ME4000_POINTER_NOT_USED here.

**<ExtTriggerEdge>**

Array for selection of the trigger edge. The order of entries corresponds with that one in the array <ChannelNumber>.

- ME4000_AO_TRIGGER_EXT_EDGE_RISING
  Starts on a rising trigger edge.
- ME4000_AO_TRIGGER_EXT_EDGE_FALLING
  Starts on a falling trigger edge.
- ME4000_AO_TRIGGER_EXT_EDGE_BOTH
  Starts on a rising or falling trigger edge.

- ME4000_VALUE_NOT_USED
  Use this constant if the trigger input for the corresponding channel is disabled (`<ExtTriggerEnable>` = ME4000_AO_TRIGGER_ EXT_DISABLE)

If you use the constant ME4000_AO_TRIGGER_SOFTWARE in parameter `<TriggerMode>` please pass ME4000_POINTER_ NOT_USED here.

**`<TimeOutSeconds>`**

Optional you can choose a time interval in seconds which specifies the time within the first trigger pulse must be detected. Otherwise the operation is cancelled. If you don´t want to use external trigger operation or no time-out value is set, pass the constant ME4000_VALUE_NOT_USED.

**`<Value>`**

Array of the values to be output (16 bit). The value range is between -32768 (-10V) and +32767 (+10V - LSB). The order of entries corresponds with that one in the array `<ChannelNumber>`.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000AOStart

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to start the output in the operation modes „AO-Continuous“ and „AOWraparound“.

If you have choosen the external trigger option in parameter `<TriggerMode>` of the function *…AOConfig* the output starts when a matching edge at the trigger input of the channel is detected.

For synchronous start of multiple channels please use the function *…AOStartSynchronous* (see page 143).

The flow of operation is described in the “Programming” section on page 53. Programming examples are available in the ME-SDK.

### ☞ Note

If the selected D/A channel is already active when this function is called, an error message is returned.

**Behaviour of return depending on the trigger mode:**

- Software start: immediately
- ext. trigger without time-out: immediately
- ext. trigger with time-out: when the time-out expired or the external trigger signal occured.

### ● Definitions

VC: me4000AOStart(unsigned int uiBoardNumber, unsigned int uiChannelNumber);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<ChannelNumber>`**

- Start the output to the desired channel. The D/A channel number 0…3 is passed in this parameter.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000AOStartSynchronous

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used for synchronous start of multiple channels in the operation modes „AOContinuous“ and „AOWraparound“.

Before calling this function the sample rate for each channel to be involved in the synchronous output operation must be set by the function *…AOConfig*.

When using the external trigger the settings in this function are valid. Related settings in the function *…AOConfig* are ignored.

The flow of operation is described in the “Programming” section on page 53. Programming examples are available in the ME-SDK.

### ☞ Note

If the selected D/A channel is already active when this function is called, an error message is returned.

**Behaviour of return depending on the trigger mode:**

- Software start: immediately
- ext. trigger without time-out: immediately
- ext. trigger with time-out: when the time-out expired or the external trigger signal occured.

### ● Definitions

VC: me4000AOStartSynchronous(unsigned int uiBoardNumber, unsigned int *puiChannelNumber, unsigned long ulCount, int iTriggerMode, int* piExtTriggerEnable, int* piExtTriggerEdge, unsigned long ulTimeOutSeconds);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<ChannelNumber>**

Array which contains the channel numbers of the channels to be started synchronously.

**<Count>**

Count of channels listed in the array <ChannelNumber>. Also valid for the parameter <ExtTriggerEnable> and <ExtTriggerEdge>.

**<TriggerMode>**

Trigger event for the synchronous start of the channels listed in array <ChannelNumber>:

- ME4000_AO_TRIGGER_SOFTWARE
  Synchronous start by calling this function.
- ME4000_AO_TRIGGER_EXT_DIGITAL
  Ready to run after calling this function. The simultaneous output starts when the external trigger signal is detected (see also the following parameters).

**<ExtTriggerEnable>**

Array to enable a single or multiple trigger inputs (DA_TRIG_*x*). The order of entries corresponds with that one in the array <ChannelNumber>. The output starts when the first matching edge at one of the enabled trigger inputs is detected.

- ME4000_AO_TRIGGER_EXT_DISABLE
  Corresponding trigger input is disabled.
- ME4000_AO_TRIGGER_EXT_ENABLE
  Corresponding trigger input is enabled.

If you use the constant ME4000_AO_TRIGGER_SOFTWARE in parameter <TriggerMode> please pass ME4000_POINTER_NOT_USED here.

**<ExtTriggerEdge>**

Array for selection of the trigger edge. The order of entries corresponds with that one in the array <ChannelNumber>.

- ME4000_AO_TRIGGER_EXT_EDGE_RISING
  Starts on a rising trigger edge.
- ME4000_AO_TRIGGER_EXT_EDGE_FALLING
  Starts on a falling trigger edge.
- ME4000_AO_TRIGGER_EXT_EDGE_BOTH
  Starts on a rising or falling trigger edge.

- ME4000_VALUE_NOT_USED
  Use this constant if the trigger input for the corresponding channel is disabled (`<ExtTriggerEnable>` = ME4000_AO_TRIGGER_ EXT_DISABLE)

If you use the constant ME4000_AO_TRIGGER_SOFTWARE in parameter `<TriggerMode>` please pass ME4000_POINTER_ NOT_USED here.

**`<TimeOutSeconds>`**

Optional you can choose a time interval in seconds which specifies the time within the first trigger pulse must be detected. Otherwise the operation is cancelled. If you don´t want to use external trigger operation or no time-out value is set, pass the constant ME4000_VALUE_NOT_USED.

The value for `<TimeOutSeconds>` passed in the functions *…AOContinuous* or *…AOWraparound* will be ignored.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000AOStop

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to end an analog output operation process started in the „AOContinuous“ or „AOWraparound“ modes of operation. In the „AOContinuous“ mode of operation, the data output is ended completely and immediately. The selected output channel is set to 0V and the ring buffer is deleted. In the „AOWraparound“ mode of operation the parameter `<StopMode>` is used to define whether the operation is stopped immediately or whether to continue until the last value in the ring buffer is output. As long as the operation mode for this channel was not changed, a new output process can be started again by calling the function *…AOStart(Synchronous)*.

### ● Definitions

VC: me4000AOStop(unsigned int uiBoardNumber, unsigned int uiChannelNumber, int iStopMode);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<ChannelNumber>`**

- Stop the output of the desired channel. The D/A channel number 0…3 is passed in this parameter.

**`<StopMode>`**

- ME4000_AO_STOP_MODE_LAST_VALUE
  The output is stopped when the last value in the ring buffer is output (only relevant in the “AOWraparound” mode)
- ME4000_AO_STOP_MODE_IMMEDIATE
  The output is stopped immediately and the output channel is set to 0V.

### ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000AOVoltToDigit

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | ✔ | ✔ |

This function allows the simple conversion of the voltage values [V] to be output into digit values [Digits]. The voltage can be output in steps of 0.3 mV = 1 Digit. Using this function is optional.

For a output voltage range of ±10 V the following formula is valid (characteristic curve of the D/A converter see diagram 12 on page 22):

$$U[\text{Digits}] = \frac{32768}{10\text{V}} \cdot U[\text{V}]$$

### ● Definitions

VC: me4000AOVoltToDigit(double dVolt, short* psDigit);
LV: me4000LV_… (see `me4000LV.h`)
VB: me4000VB_… (see `me4000.bas`)
VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<Volt>`**

Voltage value to be output in volts.

**`<Digit>`**

Pointer to digit value to be output.

### ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000AOWaveGen

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function provides you with a virtual function generator (rectangle, sine, triangle,…) which can be easily programmed. This function handles the entire configuration for the selected channel. The output is started immediately after this function is called and is ended by calling the function *me4000AOStop*. The signal is always output with the max. number of „steps“ (minimum 5) and depends on the selected frequency and shape (rectangle max. 250kHz, others max. 100kHz). Exception: A rectangle signal is always output with 2 steps per period. See also chapter „AOWraparound“ on page 60.

Programming examples are available in the ME-SDK.

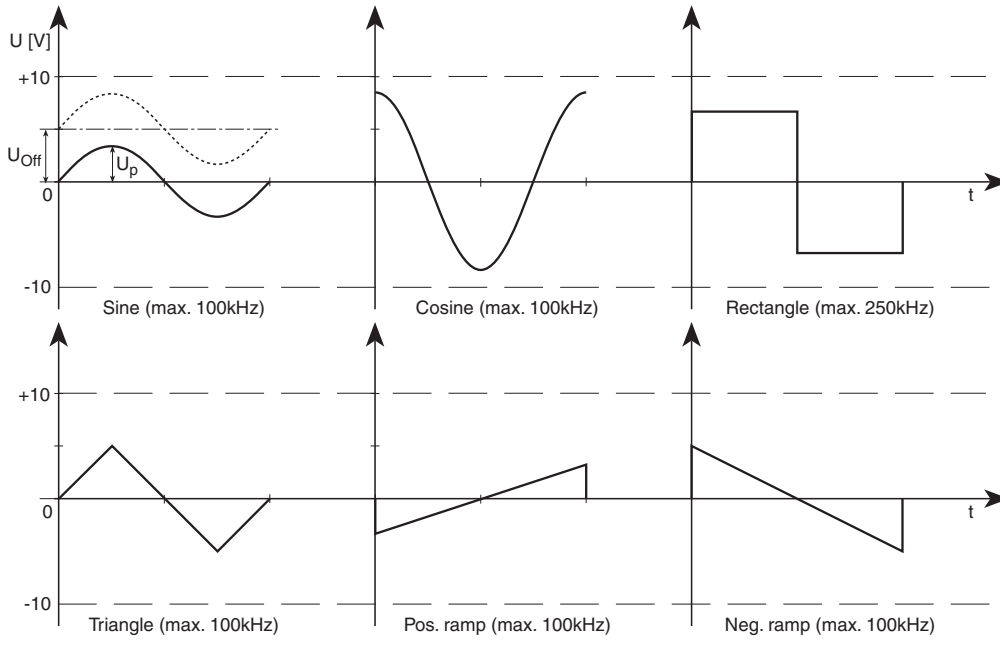



*Diagram 57: Offset, amplitude, shape*

### ☞ Note

**The output operation using this function runs entirely on the firmware level, therefore the host PC is not loaded!**

It is not possible to use an external trigger with this function. If external trigger operation is required, use the function *…AOContinuous* or *…AOWraparound*.

If the parameters used go beyond the hardware limitations of the board, the driver returns and error message. If any of the required hardware resources are active when this function is called, the driver returns an error message.

## ● Definitions

VC: me4000AOWaveGen(unsigned int uiBoardNumber, unsigned int uiChannelNumber, int iShape, double dAmplitude, double dOffset, double dFrequency);

LV: me4000LV_... (see `me4000LV.h`)

VB: me4000VB_... (see `me4000.bas`)

VEE: me4000VEE_... (see `me4000VEE.h`)

## ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<ChannelNumber>`**

D/A channel number 0…3

**`<Shape>`**

Signal curve; possible values:

- ME4000_AO_SHAPE_RECTANGLE Rectangle signal
- ME4000_AO_SHAPE_TRIANGLE Triangle signal
- ME4000_AO_SHAPE_SINUS Sine signal
- ME4000_AO_SHAPE_COSINUS Cosine signal
- ME4000_AO_SHAPE_POS_RAMP Positive ramp
- ME4000_AO_SHAPE_NEG_RAMP Negative ramp

**`<Amplitude>`**

Signal amplitude $U_p$ [V] as a decimal voltage value; the value range is between: 0…+10.00V

**`<Offset>`**

Offset voltage $U_{Off}$ [V] for moving the signal in positive or negative direction; value range: -10.00V…+10.00V

**`<Frequency>`**

Frequency in [Hz] of the signal to be output periodically; value range: 0…100000Hz (rectangle up to 250000Hz)

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000AOWraparound

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to prepare the operation mode "AOWraparound". Periodic signals of all types can be sent to the analog output channels 0…3 using this function. The channels must be loaded once before the output process can start. Create a data buffer of defined size which contains the values to be output for each channel.

The D/A timer provides a fixed time base (sample rate) for the output process (see function *…AOConfig*).

The D/A-timer provides a fixed time base (sample rate) for the output operation (see *…AOConfig*).

The output is started by calling the function *…AOStart(Synchronous)*. The output either starts immediately (software start) or when the external trigger is detected (see function *…AOConfig*).

The function *…AOStop* is used to end the data output process either immediately or it can be stopped with the last value in the FIFO i. e. with a known voltage value on the output. If the operational mode was not changed, a new output process can be started again from the beginning any time by calling the *…AOStart(Synchronous)* function. If the *…AOReset* function is used to end the data output process, the FIFO is also cleared. This terminates the process **completely**.

The flow of operation is described in the "Programming" section on page 53. Programming examples are available in the ME-SDK.

### ☞ Note

If the data buffer is less than 4096 values and the output operation is configured for "infinite", the entire process runs on the firmware level. The host PC is not loaded by the running output process!

If the selected D/A channel is already active when this function is called, an error message is returned.

### ● Definitions

Type definition for ME4000_P_AO_TERMINATE_PROC:

```
typedef void (_stdcall *
ME4000_P_AO_TERMINATE_PROC)
(void* pTerminateContext);
```

VC: me4000AOWraparound(unsigned int uiBoardNumber, unsigned int uiChannelNumber, short* psBuffer, unsigned long ulDataCount, unsigned long ulLoops, int iExecutionMode, ME4000_P_AO_TERMINATE_PROC pTerminateProc, void* pTerminateContext, unsigned long ulTimeOutSeconds);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<ChannelNumber>**

D/A channel number 0…3

**<Buffer>**

Pointer to a user allocated data buffer with the voltage values to be output.

**<DataCount>**

Number of values in data buffer `<Buffer>`.

**<Loops>**

This parameter defines how often the values in the data buffer „Buffer“ are to be output. For a infinite operation, pass the constant: ME4000_AO_WRAPAROUND_INFINITE.

**<ExecutionMode>**

Choose the execution mode for this function:

- ME4000_AO_WRAPAROUND_BLOCKING:
  The program flow is blocked until the data output process is ended. It is not possible to use this mode when the `<Loops>` parameter (described above) is set to “infinite“.
- ME4000_AO_WRAPAROUND_ASYNCHRONOUS:
  The output is handled as a background operation (asynchronous). The program flow is not blocked.

**<TerminateProc>** **~~LV~~, ~~VB~~, ~~VEE~~**

„Terminate“-function that is executed when the output process is completed. If this operation is not required, pass the constant ME4000_POINTER_NOT_USED.

**<TerminateContext>** **~~LV~~, ~~VB~~, ~~VEE~~**

User defined pointer passed to the „Terminate“-function. If the “Terminate”-function is not used, pass the constant ME4000_POINTER_NOT_USED.

**<TimeOutSeconds>**

Optional you can choose a time interval in seconds which specifies the time within the first trigger pulse must be detected. Otherwise the operation is cancelled. If you don´t want to use external trigger operation or no time-out value is set, pass the constant ME4000_VALUE_NOT_USED.

When using synchronous start the time-out value in the function *…AOStartSynchronous* is valid.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## 5.3.5 Digital Input/Output

### 5.3.5.1 Bit Pattern Output

**me4000DIOBPAppendNewValues**

#### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to continuously reload the ring buffer during a running bit pattern output process.

Use the function …*DIOBPStop* or …*DIOBPReset* to end the output process.

#### ☞ Note

It is not required that the same data buffer passed in the function …*DIOBPContinuous* be used.

#### ● Definitions

VC: me4000DIOBPAppendNewValues(unsigned int uiBoardNumber, short* psBuffer, unsigned long ulNumberOfValuesToAppend, int iExecutionMode, unsigned long* pulNumberOfValuesAppended);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

#### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<Buffer>`**

Pointer to the data buffer with the bit pattern stream to be reloaded.

**`<NumberOfValuesToAppend>`**

Number of values in the data buffer. If you pass the value „0“ here the parameter `<NumberOfValuesAppended>` returns the number of values, for which is currently space in the data buffer.

**<ExecutionMode>**

Choose execution mode for this function:

- ME4000_DIOBP_APPEND_NEW_VALUES_BLOCKING:
  The program is blocked until all values are written to the ring buffer.
- ME4000_DIOBP_APPEND_NEW_VALUES_NON_BLOCKING:
  The program only "refills" the number of values which currently have room in the ring buffer.

If you passed the value „0" in parameter `<NumberOfValuesToAppend>` this parameter is not relevant.

**<NumberOfValuesAppended>**

The actual number of values written to the ring buffer. See also parameter `<NumberOfValuesToAppend>`.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000DIOBPConfig

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to configure the board for a timer controlled bit pattern output through the digital ports. The operation modes „Bit-Pattern-Continuous" and „BitPattern-Wraparound" are available. The output process is started by calling the function …*DIOBPStart*. The process either starts immediately (software start) or when the external trigger is detected.

A 32 bit counter serves as the time base. It is supplied by a clock frequency of 33 MHz which gives a period of 30.30ns as the smallest possible time unit. This is referred to as "1 Tick" in the following sections. The sample rate for the bit pattern output must be a multiple of "1 Tick" and is passed in the parameter `<Ticks>`. i. e. the sample rate can be set in increments of 30.30ns between the minimum and maximum sample rate. The minimum sample rate is approx. 0.5 samples per minute, the maximum sampling rate is 500 kS/s for TTL ports and 172 kS/s for the optically isolated port A of the "i"-versions.

**Note:** The function …*DIOBP**Port**Config* must first be called (separately) for each port involved in the bit pattern output (see also diagram 44 on page 65).

## ☞ Note

The Functions …*FrequencyToTicks* and …*TimeToTicks* allow convenient conversion of frequency resp. period time to "Ticks" which can be passed to the timer (see page 103).

During this mode of operation, no timer controlled output on D/A channel 3 is possible. If the required hardware is already active when this function is called, an error message is returned.

## ● Definitions

VC: me4000DIOBPConfig(unsigned int uiBoardNumber, unsigned long ulTicks, int iTriggerMode, int iExtTriggerEdge);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<Ticks>`**

Number of Ticks for the 32 bit timer. This sets the sample rate for the timer controlled bit pattern output. The value range is between 66 (42Hex) for TTL ports resp. 192 (C0Hex) for optoisolated ports and $2^{32}$-1 (FFFFFFFFHex) Ticks.

**`<TriggerMode>`**

Trigger event to start the bit pattern output:

- ME4000_DIOBP_TRIGGER_SOFTWARE
  Start by software after calling the function …*DIOBPStart*.
- ME4000_DIOBP_TRIGGER_EXT_DIGITAL
  Ready to run after calling the function …*DIOBPStart*. The output is started when the external trigger signal is detected.

**<ExtTriggerEdge>**

Selects the trigger edge (trigger input is DA_TRIG_3).

- ME4000_DIOBP_TRIGGER_EXT_EDGE_RISING
  Starts the output on a rising edge.
- ME4000_DIOBP_TRIGGER_EXT_EDGE_FALLING
  Starts the output on a falling edge.
- ME4000_DIOBP_TRIGGER_EXT_EDGE_BOTH
  Starts the output on a rising or falling edge.
- ME4000_VALUE_NOT_USED
  No external trigger in use. See parameter <TriggerMode>.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000DIOBPContinuous

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function prepares for a timer controlled bit pattern output of up to 500 kS/s using the digital ports. Any bit pattern can be output which can also be changed during the output process is running (in opposite to the operation mode "BitPattern-Wraparound"). The timer provides a fixed time base (sample rate) for outputting the single values (see function …*DIOBPConfig*). A data buffer must be allocated for each channel being used and loaded with the initial bit pattern package for output. Use the function …*DIOBPAppendNewValues* for continuously reloading new values. This can be done with or without a callback function.

The output is started by calling the function …*DIOBPStart*. The output either starts immediately by calling this function (software start) or when an external trigger pulse is detected (see function …*DIOBPConfig*). The function …*DIOBPStop* is used to end the output process immediately. If the operation mode was not changed, a new output process can be started again any time by calling the function …*DIOBPStart*. If the …*DIOBPReset* function is used to end the data output process, the FIFO is also cleared and the process is completely terminated.

### ☞ Note

During this mode of operation, no timer controlled output on D/A channel 3 is possible. If the required hardware is already active when this function is called, an error message is returned.

### ● Definitions

Type definition for ME4000_P_DIOBP_CALLBACK_PROC:

```
typedef void (_stdcall *
ME4000_P_DIOBP_CALLBACK_PROC)
(unsigned long ulBufferAvailable,
void* pCallbackContext);
```

VC: me4000DIOBPContinuous(unsigned int uiBoardNumber, short* psBuffer, unsigned long ulDataCount, ME4000_P_DIOBP_CALLBACK_PROC pCallbackProc, void* pCallbackContext, unsigned long ulTimeOutSeconds, unsigned long* pulNumberOfValuesWritten);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<Buffer>**

Pointer to an user defined data buffer, which is „filled“ with the **initial** values to be output.

**<DataCount>**

Number of values in data buffer <Buffer>.

**<CallbackProc>** ~~**LV**~~**, ~~VB~~, ~~VEE~~**

Callback function which is called at regular intervals to reload the data buffer. If this operation is not required, pass the constant ME4000_POINTER_NOT_USED.

**<CallbackContext>** ~~**LV**~~**, ~~VB~~, ~~VEE~~**

This is a user defined pointer that can be passed to the callback function. If no callback function is being used, pass the constant ME4000_POINTER_NOT_USED.

**<TimeOutSeconds>**

Optional you can choose a time interval in seconds which specifies the time within the first trigger pulse must be detected. Otherwise the operation is cancelled. If you don´t want to use external trigger operation or no time-out value is set, pass the constant ME4000_VALUE_NOT_USED.

**<NumberOfValuesWritten>**

The number of values actual written into the data buffer.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000DIOBPGetStatus

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to check whether a bit pattern output in the „Bitpattern-Continuous“ or „BitPattern-Wraparound“ modes is still running or whether the FIFO has “run dry”. This can happen when the FIFO is intentionally allowed to run dry by not reloading any new data values to end the output process or when the values could not reloaded quickly enough because the PC performance is too slow.

The parameter `<WaitIdle>` can control whether the function returns the status immediately or whether it waits until the output process is ended.

### ● Definitions

VC: me4000DIOBPGetStatus(unsigned int uiBoardNumber, int iWaitIdle, int* piStatus);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<WaitIdle>`**

This parameter defines the “return behaviour“ of this function:

- ME4000_DIOBP_WAIT_NONE
  The current status is returned immediately in the parameter `<Status>`.
- ME4000_DIOBP_WAIT_IDLE
  The function returns when the data output process is ended (FIFO is empty). The parameter `<Status>` always returns the constant ME4000_DIOBP_STATUS_IDLE.

**<Status>**

The current status:

- ME4000_DIOBP_STATUS_IDLE
  The output is ended, i. e. the FIFO is empty.
- ME4000_DIOBP_STATUS_BUSY
  The output is still running.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000DIOBPPortConfig

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function defines the ports assignment for the bit pattern output.

The 16 bit wide bit pattern is separated into low byte (ME4000_DIOBP_OUTPUT_BYTE_LOW) and high byte (ME4000_DIOBP_OUTPUT_BYTE_HIGH). The low byte and high byte can be assigned to any of the 8 bit wide digital ports A, B, C and/or D (see diagram 44 on page 65).

This function must be called separately for each port to be used for the bit pattern output. Further configuration is done using the function …*DIOBPConfig*. The function …*DIOConfig* is not required for this operation mode.

## ☞ Note

For the optically isolated board versions („i“-versions) the direction of ports A and B are fixed by the hardware. Port A is fixed as output, port B is fixed as input. Port B can not be used for the bit pattern output, but it can still be used for normal digital input operation. Generally, all ports not used for the bit pattern output are available for use as normal digital I/O ports.

If the required hardware is already active when this function is called, an error message is returned.

## ● Definitions

VC: me4000DIOBPPortConfig(unsigned int uiBoardNumber, unsigned int uiPortNumber, int iOutputMode);

LV: me4000LV_... (see `me4000LV.h`)

VB: me4000VB_... (see `me4000.bas`)

VEE: me4000VEE_... (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<PortNumber>**

Port selection:

- ME4000_DIO_PORT_A: Port A
- ME4000_DIO_PORT_B: Port B
- ME4000_DIO_PORT_C: Port C
- ME4000_DIO_PORT_D: Port D

**<OutputMode>**

Port configuration for bit pattern output (see diagram 44 on page 65):

- ME4000_DIOBP_OUTPUT_BYTE_LOW: Low byte of the FIFO (Bit 7…0)
- ME4000_DIOBP_OUTPUT_BYTE_HIGH: High byte of the FIFO (Bit 15…8)

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000DIOBPReset

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function ends a currently running „BitPattern-Continuous“ or „BitPattern-Wraparound“ output process. The output is stopped completely and immediately. The corresponding ring buffer will be deleted and the digital ports involved are set 0000Hex.

### ● Definitions

VC: me4000DIOBPReset (unsigned int uiBoardNumber);
LV: me4000LV_… (see `me4000LV.h`)
VB: me4000VB_… (see `me4000.bas`)
VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000DIOBPStart

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to start the bit pattern output in the „BitPattern-Continuous“ and „BitPattern-Wraparound“ modes of operation. If the external trigger operation is chosen, the output process starts when a matching trigger edge is detected at pin 65 (DA_TRIG_3).

The flow of operation is described in the “Programming” section on page 65. Programming examples are available in the ME-SDK.

## ☞ Note

If any of the required hardware resources are already active when this function is called, an error message is returned.

**Behaviour of return depending on the trigger mode:**

- Software start: immediately
- ext. trigger without time-out: immediately
- ext. trigger with time-out: when the time-out expired or the external trigger signal occured.

## ● Definitions

VC: me4000DIOBPStart(unsigned int uiBoardNumber);

LV: me4000LV_... (see `me4000LV.h`)

VB: me4000VB_... (see `me4000.bas`)

VEE: me4000VEE_... (see `me4000VEE.h`)

## ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000DIOBPStop

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to end a bit pattern output started in the „Bit-Pattern-Continuous“ or „BitPattern-Wraparound“ modes of operation. In the „BitPattern-Continuous“ mode of operation, the output is ended completely and immediately. The ring buffer is deleted and the bit pattern 0000Hex is output. In the „BitPattern-Wraparound“ mode of operation the parameter `<StopMode>` is used to define whether the operation stopped immediately and the bit pattern 0000Hex is output or whether to continue until the last (known) bit pattern in the ring buffer is output. As long as the operation mode for this channel was not changed, a new output process can be started again from the beginning by calling the function …*DIOBPStart*.

The flow of operation is described in the “Programming” section on page 65. Programming examples are available in the ME-SDK.

### ● Definitions

VC: me4000DIOBPStop(unsigned int uiBoardNumber, int iStopMode);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<StopMode>`**

- ME4000_DIOBP_STOP_MODE_LAST_VALUE
  The output is stopped when the last value in the ring buffer is output (only relevant in the “BitPattern-Wraparound” mode)
- ME4000_DIOBP_STOP_MODE_IMMEDIATE
  The output is stopped immediately and bit pattern 0000Hex is output.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000DIOBPWraparound

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to prepare the operation mode “BitPattern-Wraparound”. Any bit patterns can be output repeatedly in this operation mode. Before the output process can start the ring buffer must be loaded once. Create a data buffer of defined size which contains the bit pattern stream to be output.

The timer provides a fixed time base (sample rate) for sending the bit patterns (see function …*DIOBPConfig*).

The output is started by calling the function …*DIOBPStart*. The output either starts immediately (software start) or when the external trigger is detected (see function …*DIOBPConfig*).

The function …*DIOBPStop* is used to end the data output process either immediately or it can be stopped with the last value in the FIFO i. e. with a known bit pattern on the digital ports. If the operational mode was not changed, a new output process can be started again from the beginning any time by calling the …*DIOBPStart* function. If the …*DIOBPReset* function is used to end the data output process, the FIFO is also cleared and the output process is **completely** terminated.

The flow of operation is described in the “Programming” section on page 65. Programming examples are available in the ME-SDK.

## ☞ Note

During this mode of operation, no timer controlled output on D/A channel 3 is possible. If the required hardware is already active when this function is called, an error message is returned.

If the data buffer is less than 4096 values and the output operation is configured for “infinite”, the entire process runs on the firmware level. The host PC is not loaded by the running output process!

## ● Definitions

Typdefinition für ME4000_P_DIOBP_TERMINATE_PROC:

```
typedef void (_stdcall *
ME4000_P_DIOBP_TERMINATE_PROC)
(void* pTerminateContext);
```

VC: me4000DIOBPWraparound(unsigned int uiBoardNumber, short* psBuffer, unsigned long ulDataCount, unsigned long ulLoops, int iExecutionMode, ME4000_P_DIOBP_TERMINATE_PROC pTerminateProc, void* pTerminateContext, unsigned long ulTimeOutSeconds);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<Buffer>**

Pointer to a user allocated data buffer with the bit patterns to be output.

**<DataCount>**

Number of values in data buffer `<Buffer>`.

**<Loops>**

This parameter defines how often the bit patterns in the data buffer are to be output. For a infinite operation, pass the constant: ME4000_DIOBP_WRAPAROUND_INFINITE.

**<ExecutionMode>**

Choose the execution mode for this function:

- ME4000_DIOBP_WRAPAROUND_BLOCKING:
  The program flow is blocked until the data output is ended. It is not possible to use this mode when the `<Loops>` parameter (described above) is set to “infinite“.
- ME4000_DIOBP_WRAPAROUND_ASYNCHRONOUS:
  The output is handled as a background operation (asynchronous). The program flow is not blocked.

**<TerminateProc>** **~~LV~~, ~~VB~~, ~~VEE~~**

„Terminate“-function that is executed when the output process is completed. The function is passed a pointer to the start of the data buffer and the number of values. If this operation is not required, pass the constant ME4000_POINTER_NOT_USED.

**<TerminateContext>** **~~LV~~, ~~VB~~, ~~VEE~~**

User defined pointer passed to the „Terminate“-function. If the “Terminate”-function is not used, pass the constant ME4000_POINTER_NOT_USED.

**<TimeOutSeconds>**

Optional you can choose a time interval in seconds which specifies the time within the first trigger pulse must be detected. Otherwise the operation is cancelled. If you don´t want to use external trigger operation or no time-out value is set, pass the constant ME4000_VALUE_NOT_USED.

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## 5.3.5.2 Digital-I/O Standard

### me4000DIOConfig

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function is used to set the direction of the digital ports for standard input/output operations. The port direction can be configured separately for each of the 8 bit wide ports (see note below).

This function must be called separately for each port. A port set for output can also be read back.

### ☞ Note

Boards with the optical isolation (“i”-versions) have the directions for port A and B fixed by the hardware. In that case port A is an output port and port B is an input port. Ports C and D can be configured as required.

Configuration of the ports for a bit pattern output is done using the function …*DIOBPPortConfig* (see page 160).

## ● Definitions

VC: me4000DIOConfig(unsigned int uiBoardNumber, unsigned int uiPortNumber, int iPortDirection);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<PortNumber>**

Port selection:

- ME4000_DIO_PORT_A: Port A
- ME4000_DIO_PORT_B: Port B
- ME4000_DIO_PORT_C: Port C
- ME4000_DIO_PORT_D: Port D

**<PortDirection>**

Port direction for standard input/output:

- ME4000_DIO_PORT_INPUT:
  Input port (port B on „i“-versions)
- ME4000_DIO_PORT_OUTPUT:
  Output port (port A on „i“-versions)

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000DIOGetBit

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function returns the state of the specified bit. A port set for output can also be read back with this function.

### ☞ Note

The ports must be configured with the function …*DIOConfig* first.

### ● Definitions

VC: me4000DIOGetBit(unsigned int uiBoardNumber, unsigned int uiPortNumber, unsigned int uiBitNumber, int *piBitValue);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<PortNumber>`**

Port selection:

- ME4000_DIO_PORT_A: Port A
- ME4000_DIO_PORT_B: Port B
- ME4000_DIO_PORT_C: Port C
- ME4000_DIO_PORT_D: Port D

**`<BitNumber>`**

Bit number to be checked; possible values: 0…7

**`<BitValue>`**

Pointer to an integer value which returns the state of the line.

„0“: input line is low

„1“: input line is high

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000DIOGetByte

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function reads a byte from the specified port. A port set for output can also be read back with this function.

### ☞ Note

The ports must be configured with the function …*DIOConfig* first.

### ● Definitions

VC: me4000DIOGetByte(unsigned int uiBoardNumber, unsigned int uiPortNumber, unsigned char *pucByteValue);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<PortNumber>**

Port selection:

- ME4000_DIO_PORT_A: Port A
- ME4000_DIO_PORT_B: Port B
- ME4000_DIO_PORT_C: Port C
- ME4000_DIO_PORT_D: Port D

**<ByteValue>**

Pointer to an „unsigned char“ value, which returns the read byte.

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000DIOResetAll

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function sets all ports to input (exception: port A of boards with optical isolation).

Calling this function does not effect a bit pattern output process or a „ME-MultiSig“ operation. An error message is returned in that case.

### ● Definitions

VC: int me4000DIOResetAll (unsigned int uiBoardNumber);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000DIOSetBit

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function sets the state of the specified bit.

### ☞ Note

The ports must be configured with the function …*DIOConfig* first.

### ● Definitions

VC: me4000DIOSetBit(unsigned int uiBoardNumber, unsigned int uiPortNumber, unsigned int uiBitNumber, int iBitValue);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<PortNumber>`**

Port selection:

- ME4000_DIO_PORT_A: Port A
- ME4000_DIO_PORT_B: Port B
- ME4000_DIO_PORT_C: Port C
- ME4000_DIO_PORT_D: Port D

**`<BitNumber>`**

Number of output line to be set; possible values: 0…7

**`<BitValue>`**

Possible values are:

„0“: Bit is set low level

„1“: Bit is set to high level

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000DIOSetByte

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

Wirtes a byte to an output port.

### ☞ Note

The ports must be configured with the function …*DIOConfig* first.

### ● Definitions

VC: me4000DIOSetByte(unsigned int uiBoardNumber, unsigned int uiPortNumber, unsigned char ucByteValue);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<PortNumber>`**

Port selection:

- ME4000_DIO_PORT_A: Port A
- ME4000_DIO_PORT_B: Port B
- ME4000_DIO_PORT_C: Port C
- ME4000_DIO_PORT_D: Port D

**`<ByteValue>`**

Value to be output; possible values are: 0...255 (00Hex...FFHex).

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## 5.3.6 Counter Functions

### me4000CntPWMStart

#### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | ✔ | ✔ | ✔ |

This function configures the 8254 counter chip for the operation mode „Pulse Width Modulation“ and starts the output. When this operation mode is active, counters 0…2 can not be used for any other operation. Any previous programming of the counters is overwritten. The signal is output on pin 41 (OUT_2) of the D-sub connector. A base clock (max. 10MHz) must be supplied externally. Counter 0 can be used as a prescaler (see diagram 20 on page 28). The maximum frequency of the output signal is 50kHz and is calculated as follows:

$$f_{OUT\_2} = \frac{\text{Basistakt}}{\texttt{<Prescaler>} \cdot 100} \qquad (\text{mit } \texttt{<Prescaler>} = 2\ldots(2^{16} - 1))$$

The duty cycle can be set between 1…99% in increments of 1% (see diagram 50 on page 78).

#### ☞ Note

Using this function is only useful in combination with external switching shown in diagram 20 on page 28.

#### ● Definitions

VC: me4000CntPWMStart(unsigned int uiBoardNumber, int iPrescaler, int iDutyCycle);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

#### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<Prescaler>`**

Value for the prescaler (counter 0) within the range: 2…65535.

**<DutyCycle>**

Duty cycle of the output signal to be set between 1…99% in increments of 1%.

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000CntPWMStop

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | ✔ | ✔ | ✔ |

This function is used to end an operation started with the function *…CntPWMStart*.

### ● Definitions

VC: me4000CntPWMStop(unsigned int uiBoardNumber);LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000CntRead

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | ✔ | ✔ | ✔ |

Buffers the current value of the selected counter and reads the value.

### ● Definitions

VC: me4000CntRead(unsigned int uiBoardNumber, unsigned int uiCounterNumber, unsigned short* pusValue);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<CounterNumber>**

Specifies the counter to be read; possible values: 0, 1, 2

**<Value>**

Counter state as a 16 bit value.

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000CntWrite

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | ✔ | ✔ | ✔ |

This function configures the selected counter with the desired mode of operation and loads a 16 bit start value into the count register. The counter is automatically started when this function is called.

For further information about programming the counter chip 8254, refer to the suppliers data sheets (e. g. NEC, Intel, Harris) which are available in the internet.

### ● Definitions

VC: me4000CntWrite(unsigned int uiBoardNumber, unsigned int uiCounterNumber, int iMode, unsigned short usValue);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameter

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<CounterNumber>`**

Specifies the counter to be configured; possible values: 0, 1, 2

**`<Mode>`**

Operation mode of the counter (see chapter 4.4 on page 75).

- ME4000_CNT_MODE_0
  "Change state at zero"
- ME4000_CNT_MODE_1
  "Retriggerable One-Shot"
- ME4000_CNT_MODE_2
  "Asymmetric divider"
- ME4000_CNT_MODE_3
  "Symmetric divider"
- ME4000_CNT_MODE_4
  "Counter start by software trigger"
- ME4000_CNT_MODE_5
  "Counter start by hardware trigger"

**<Value>**

16 bit start value for specified counter; value range: 0…65535 (0000Hex…FFFFHex)

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## 5.3.7 External Interrupt

### me4000ExtIrqDisable

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function disables the external interrupt function.

### ● Definitions

VC: int me4000ExtIrqDisable (unsigned int uiBoardNumber);
LV: me4000LV_… (see `me4000LV.h`)
VB: me4000VB_… (see `me4000.bas`)
VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000ExtIrqEnable

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function is used to enable the external interrupt function. An incoming interrupt is sent directly to the operating system.

### ● Definitions

Type definition for ME4000_P_EXT_IRQ_PROC:

```
typedef void (_stdcall *
ME4000_P_EXT_IRQ_PROC)
(void* pExtIrqContext);
```

VC: me4000ExtIrqEnable(unsigned int uiBoardNumber, ME4000_P_EXT_IRQ_PROC pExtIrqProc, void* pExtIrqContext);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<ExtIrqProc>** **~~LV~~, ~~VB~~, ~~VEE~~**

This is a pointer to a user defined callback function. If this operation is not required, pass the constant ME4000_POINTER_NOT_USED.

**<ExtIrqContext>** **~~LV~~, ~~VB~~, ~~VEE~~**

This is a user defined pointer that is passed to the callback function. If no callback function is required, pass the constant ME4000_POINTER_NOT_USED.

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000ExtIrqGetCount

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function determines the number of external interrupts which have occurred since the device was started. The purpose of this function is to provide external interrupt operation when using graphical programming languages such as Agilent VEE or LabVIEW™.

The interrupt function must be enabled and disabled using the functions …*ExtIrqEnable* and …*ExtIrqDisable* as usual. By checking the interrupt count parameter `<IrqCount>` and comparing it to a previously read in interrupt count, it is possible to determine whether or not an external interrupt has occurred.

### ● Definitions

VC: me4000ExtIrqGetCount(unsigned int uiBoardNumber, unsigned int *puiIrqCount);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<IrqCount>`**

Number of interrupts since the device was started.

### ☞ Example

```
me4000ExtIrqEnable(0, ME4000_POINTER_NOT_USED,
   ME4000_POINTER_NOT_USED);
me4000ExtIrqGetCount(0, &iIrqCountBefore);

Sleep(1000);           //waiting for interrupts

me4000ExtIrqGetCount(0, &iIrqCountAfter);
me4000ExtIrqDisable(0);
IrqCount = (iIrqCountAfter-iIrqCountBefore);
```

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## 5.3.8 MultiSig Functions

### me4000MultiSigAddressLED

#### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

Identification of the specified base board by setting the respective address LED (e. g. for maintenance purposes).

#### ☞ Note

When this operation is required with an opto-isolated ME-4600 series board, we recommend using the adapter ME-AA4-3i.

#### ● Definitions

VC: me4000MultiSigAddressLED(unsigned int uiBoardNumber, unsigned int uiBase, int iLEDStatus);

LV: me4000LV_... (see `me4000LV.h`)

VB: me4000VB_... (see `me4000.bas`)

VEE: me4000VEE_... (see `me4000VEE.h`)

#### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<Base>`**

Address of the base board; possible values: 0 … 7.
(the master board always uses address „0“).

**`<LEDStatus>`**

Turn the LED on/off for the specified base board.

- ME4000_MULTISIG_LED_OFF
  Switch off the address LED
- ME4000_MULTISIG_LED_ON
  Switch on the address LED

#### ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigClose

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function closes a configuration mode that was opened with the function …*MultiSigOpen*. Hardware resources that were reserved are released again. See also chapter 4.5 „ME-MultiSig Control“ on page 79.

### ● Definitions

VC: int me4000MultiSigClose (unsigned int uiBoardNumber);
LV: me4000LV_… (see `me4000LV.h`)
VB: me4000VB_… (see `me4000.bas`)
VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigOpen

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function opens the configuration mode for the following functional operations of the multiplexer base board ME-MUX32-M/S:

- Setting the gain factor (see …*MultiSigSetGain*)
- Controlling the address LED (see …*MultiSigAddressLED*)
- General reset (see …*MultiSigReset*)

***Note:*** When this operation is required with an opto-isolated ME-4600 series board, we recommend using the adapter ME-AA4-3i.

The following hardware resources will be reserved

- Without opto-isolation: digital port A and B.
- With opto-isolation: digital port A and C.

See also chapter 4.5 „ME-MultiSig Control“ on page 79.

### ● Definitions

VC: int me4000MultiSigOpen (unsigned int uiBoardNumber);
LV: me4000LV_… (see `me4000LV.h`)
VB: me4000VB_… (see `me4000.bas`)
VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigReset

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

Resets all master and slave boards to their default state. The gain is set to V=1 and the address LEDs are turned off.

### ☞ Note

When this operation is required with an optically isolated ME-4600 series board, we recommend using the adapter ME-AA4-3i.

### ● Definitions

VC: me4000MultiSigReset(unsigned int uiBoardNumber);
LV: me4000LV_… (see `me4000LV.h`)
VB: me4000VB_… (see `me4000.bas`)
VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigSetGain

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function is used to set the gain for the specified channel group. Standard setting is to have the gain V=1.

### ☞ Note

When this operation is required with an optically isolated ME-4600 series board, we recommend using the adapter ME-AA4-3i.

### ● Definitions

VC: me4000MultiSigSetGain(unsigned int uiBoardNumber, unsigned int uiBase, int iChannelGroup, int iGain);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<Base>`**

Address of the base board; possible values: 0 … 7.
(the master board always uses address „0“).

**`<ChannelGroup>`**

Selection of channel group:

- ME4000_MULTISIG_GROUP_A:
  Channel group A
- ME4000_MULTISIG_GROUP_B:
  Channel group B

**`<Gain>`**

Setting the gain factor for the specified channel group (a gain factor set by the function …*MultiSigAISingle* will be overwritten):

- ME4000_MULTISIG_GAIN_1
  Gain factor 1 (standard)
- ME4000_MULTISIG_GAIN_10
  Gain factor 10
- ME4000_MULTISIG_GAIN_100
  Gain factor 100

### ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

#### 5.3.8.1 „Mux“ Functions

The following functions are used for analog data acquisition being done with the ME-MultiSig system together with a board from the ME-4600 series (see also “Mux” operation on page 79 and the “ME-MultiSig” system manual).

**me4000MultiSigAIClose**

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function closes a functional operation that was opened with the function *…MultiSigAIOpen*. Hardware resources that were reserved are released again. See also chapter 4.5.1 „„Mux“ Operation“ on page 79.

### ● Definitions

VC: int me4000MultiSigAIClose (unsigned int uiBoardNumber);
LV: me4000LV_… (see `me4000LV.h`)
VB: me4000VB_… (see `me4000.bas`)
VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigAIConfig

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function configures the hardware of the ME-4600 for a timer controlled data acquisition in combination with ME-MultiSig system in the „Single-Mux“ operation mode (see manual of the ME-MultiSig system).

The timer is configured, it passes the „Mux“ channel list, determines the acquisition mode `<AcqMode>` and establishes the external trigger mode and edge. The channel list for configuration of the A/D section of the ME-4600 is automatically generated. The ME-4600 always uses the input voltage range ±10V in single ended mode.

After the function …*MultiSigAIScan* or …*MultiSigAIContinuous* was called, the data acquisition is always started with the function …*MultiSigAIStart* either immediately (software start) or when an external trigger pulse is detected (see function …*MultiSigAIConfig*).

### ☞ Note

A 32 bit CHAN timer and a 36 bit SCAN timer are available for setting the data acquisition timing. The CHAN timer sets the sample rate within the „Mux“ channel list. The max. CHAN time is approx. 130s, the min. CHAN time is 2 µs for not opto-isolated versions resp. 5.8µs for opto-isolated versions. The SCAN timer sets the time between the the first entry in the „Mux“ channel list of two consecutive channel list processings. Setting the SCAN timer is optional. All timers use a base clock frequency of 33 MHz. This results in a period of 30.30 ns, which is the smallest time unit available and will be referred to as “1 Tick”. The period is set as a multiple of one tick and passed to the parameters `<ChanTicks>`, `<ScanTicksLow>` and `<ScanTicksHigh>`. The `<ScanTicksHigh>` parameter is only required for SCAN times >130s.

The functions …*FrequencyToTicks* and …*TimeToTicks* (see page 103) can be used for convenient conversion of frequency resp. period into ticks for passing to the timers.

## ● Definitions

VC: me4000MultiSigAIConfig(unsigned int uiBoardNumber, unsigned int uiAIChannelNumber, unsigned char *pucMuxChanList, unsigned int uiMuxChanListCount, unsigned long ulReserved, unsigned long ulChanTicks, unsigned long ulScanTicksLow, unsigned long ulScanTicksHigh, int iAcqMode, int iExtTriggerMode, int iExtTriggerEdge);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<AIChannelNumber>`**

A/D channel number for which the „Mux“ chain was configured. The channel number must correspond with solder bridge „A“ on the master board; possible values: 0…31 (ME-4650/4660: 0…15)

**`<MuxChanList>`**

Pointer to the beginning of the „Mux“ channel list controlling the multiplexers. A buffer of defined size must be allocated and written to in the desired order for the „Mux“ channel numbers (0…255).

**`<MuxChanListCount>`**

Number of „Mux“ channel list entries.

**`<Reserved>`**

This parameter is reserved. Pass the value „0“.

**`<ChanTicks>`**

Number of Ticks for the CHAN timer (32 bit) which sets the sample rate. The value range is between 66 (42Hex) for not opto-isolated versions resp. 192 (C0Hex) for opto-isolated versions and $2^{32}$-1 (FFFFFFFFHex) Ticks.

**<ScanTicksLow>**

Number of Ticks for the lower part (bits 31…0) of the 36 bit SCAN timer. It determines the time between the sampling of the first „Mux“ channel list entry of two consecutive channel list processings. If the SCAN timer is not used the constant ME4000_VALUE_NOT_USED is passed to this parameter **and** the `<ScanTicksHigh>` parameter.

The minimum SCAN time is (see also diagram 25):
(number of „Mux“ channel list entries x CHAN time) + „x“ Ticks

The max. allowable SCAN time is 30 minutes, this is 59,400,000,000 Ticks (DD4841200Hex).

**<ScanTicksHigh>**

Number of Ticks for the higher part (bits 35…32) of the 36 bit SCAN timer. This timer is only required for scan times over 130.15s else pass the constant ME4000_VALUE_NOT_USED.

**<AcqMode>**

Acquisition mode for timer controlled acquisition:

- ME4000_AI_ACQ_MODE_SOFTWARE
  The data acquisition is started by calling the function …*MultiSigAIStart*. Data is acquired according to the timer settings (see diagram 25).
- ME4000_AI_ACQ_MODE_EXT
  The data acquisition is ready for starting after calling the …*MultiSigAIStart* function. The actual data acquisition starts on the first external trigger pulse. Data is acquired according to the timer settings. Only the first trigger pulse has an effect, all others are ignored (see diagram 32).
- ME4000_AI_ACQ_MODE_EXT_SINGLE_VALUE
  The data acquisition is ready for starting after calling the …*MultiSigAIStart* function. On every external trigger pulse exactly **one** value is acquired (according to the channel list). A time delay of one CHAN-time is between the first trigger pulse and the first conversion. Otherwise, the timer settings have no effect (see diagram 33).
- ME4000_AI_ACQ_MODE_EXT_SINGLE_CHANLIST
  The data acquisition is ready for starting after calling the …*MultiSigAIStart* function. Every time a trigger pulse occurs, the channel list is processed once. Data is acquired according to the timer settings (see diagram 34). The SCAN timer has no effect.

**<ExtTriggerMode>**

Selects the external trigger source for the A/D-section.

- ME4000_AI_TRIGGER_EXT_DIGITAL
  Trigger source is the digital trigger input.
- ME4000_AI_TRIGGER_EXT_ANALOG (not ME-4650/4660)
  Trigger source is the analog trigger unit.
- ME4000_VALUE_NOT_USED
  No external trigger in use. See parameter <AcqMode>.

**<ExtTriggerEdge>**

Selects the trigger edge for the A/D-section.

- ME4000_AI_TRIGGER_EXT_EDGE_RISING
  Starts the acquisition on a rising edge.
- ME4000_AI_TRIGGER_EXT_EDGE_FALLING
  Starts the acquisition on a falling edge.
- ME4000_AI_TRIGGER_EXT_EDGE_BOTH
  Starts the acquisition on a rising or falling edge.
- ME4000_VALUE_NOT_USED
  No external trigger in use. See parameter <AcqMode>.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000MultiSigAIContinuous

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to prepare the software for timer controlled acquisition when the number of measured values is not known. This function is always executed asynchronously. The measured values are retrieved either with a user defined callback function or repeated calls of the function *…MultiSigAIGetNewValues*.

The data acquisition is always started with the function *…MultiSigAIStart*. It starts immediately (software start) or when an external trigger is detected (see *…MultiSigAIConfig*). If working with an external trigger, and this does not appear, the "Time-Out" value can be used to cancel the data acquisition process. The data acquisition is ended by calling the function *…MultiSigAIStop* or *…MultiSigAIReset*.

## ☞ Note

The flow of operation is described in chapter 4.1.3 "Timer Controlled „AI Operation Modes“" on page 34. Programming examples are available in the ME-SDK.

## ● Definitions

Type definition for ME4000_P_AI_CALLBACK_PROC:

```
typedef void (_stdcall *
ME4000_P_AI_CALLBACK_PROC)(short* psValues,
unsigned int uiNumberOfValues, void* pCall-
backContext, int iLastError);
```

VC: me4000MultiSigAIContinuous(unsigned int uiBoardNumber, ME4000_P_AI_CALLBACK_PROC pCallbackProc, void* pCallbackContext, unsigned int uiRefreshFrequency, unsigned long ulTimeOutSeconds);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<CallbackProc>** **~~LV~~, ~~VB~~, ~~VEE~~**

This defines the callback function called in fixed intervals during the data acquisition. The function receives a pointer to the new values and the number of new values. If this operation is not required, pass the constant ME4000_POINTER_NOT_USED.

**<CallbackContext>** **~~LV~~, ~~VB~~, ~~VEE~~**

User defined pointer passed to the callback function. If no callback function is used, pass the constant ME4000_POINTER_NOT_USED.

**<RefreshFrequency>**

Defines the number of channel list processings after which the ring buffer should be read. The value passed here serves as a guideline and can be fitted by the driver. If the constant ME4000_VALUE_NOT_USED is passed, the driver will determine a useful value automatically.

**<TimeOutSeconds>**

Optional you can choose a time interval in seconds which specifies the time within the first trigger pulse must be detected. Otherwise the operation is cancelled. If you don´t want to use external trigger operation or no time-out value is set, pass the constant ME4000_VALUE_NOT_USED.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000MultiSigAIDigitToSize

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function provides a convenient way to convert the raw data values (digits) into the required physical dimension. The gain factor (1, 10, 100) set on the base board or an (optional) expansion module for signal conditioning is taken into account in this conversion. This function can be used to convert a single value or an entire array of values by calling it repeatedly. Its use is optional.

## ☞ Note

For timer controlled „Mux“ operation always use the function …*MultiSigAIExtractValues* before calling this function. This is the only way to guarantee that the proper gain factor for the channel groups and that the different expansion modules can be taken into account for calculation.

This function assumes an input voltage range of ±10V of the ME-4600. If you work with the “MultiSig” functions the ±10V input range is used automatically.

The temperature calculation for RTDs is done according to DIN EN 60751 and that one for thermocouples according to DIN EN 60584. More informations how to calculate the temperature can be found in the „ME-MultiSig“ manual.

## ● Definitions

VC: me4000MultiSigAIDigitToSize(short sDigit, int iGain, int iModuleType, double dRefValue, double* pdSize);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)
VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**`<Digit>`**

The "Raw" data value as it appears in the buffer after the data acquisition.

**`<Gain>`**

Pass the same gain factor as used in the function …*MultiSigSetGain* resp. …*MultiSigAISingle*:

- ME4000_MULTISIG_GAIN_1
  Gain factor 1 (standard)
- ME4000_MULTISIG_GAIN_10
  Gain factor 10
- ME4000_MULTISIG_GAIN_100
  Gain factor 100

**`<ModuleType>`**

If an expansion module for signal conditioning is being used, set the module type in this parameter. The `<Gain>` parameter must be set to 1. This is important and ensures that the measured value is correctly calculated. If no expansion module is being used, pass the first constant:

- ME4000_MULTISIG_MODULE_NONE
  No expansion module is used (standard)

- ME4000_MULTISIG_MODULE_DIFF16_10V
  Expansion module ME-Diff16 with input range 10V
- ME4000_MULTISIG_MODULE_DIFF16_20V
  Expansion module ME-Diff16 with input range 20V
- ME4000_MULTISIG_MODULE_DIFF16_50V
  Expansion module ME-Diff16 with input range 50V

- ME4000_MULTISIG_MODULE_CURRENT16_0_20MA
  Expansion module ME-Current16 with input range 0…20mA

- ME4000_MULTISIG_MODULE_RTD8_PT100
  Expansion module ME-RTD8 for RTDs of type Pt100 (0.4Ω/K)
- ME4000_MULTISIG_MODULE_RTD8_PT500
  Expansion module ME-RTD8 for RTDs of type Pt500 (2.0Ω/K)
- ME4000_MULTISIG_MODULE_RTD8_PT1000
  Expansion module ME-RTD8 for RTDs of type Pt1000 (4.0Ω/K)

- ME4000_MULTISIG_MODULE_TE8_TYPE_B
  Expansion module ME-TE8 for thermocouple type B
- ME4000_MULTISIG_MODULE_TE8_TYPE_E
  Expansion module ME-TE8 for thermocouple type E
- ME4000_MULTISIG_MODULE_TE8_TYPE_J
  Expansion module ME-TE8 for thermocouple type J
- ME4000_MULTISIG_MODULE_TE8_TYPE_K
  Expansion module ME-TE8 for thermocouple type K
- ME4000_MULTISIG_MODULE_TE8_TYPE_N
  Expansion module ME-TE8 for thermocouple type N
- ME4000_MULTISIG_MODULE_TE8_TYPE_R
  Expansion module ME-TE8 for thermocouple type R
- ME4000_MULTISIG_MODULE_TE8_TYPE_S
  Expansion module ME-TE8 for thermocouple type S
- ME4000_MULTISIG_MODULE_TE8_TYPE_T
  Expansion module ME-TE8 for thermocouple type T
- ME4000_MULTISIG_MODULE_TE8_TEMP_SENSOR
  Expansion module ME-TE8, channel used for reference temperature at the sensor connector of the module (cold junction compensation)

**<RefValue>**

This parameter is only relevant if you have chosen an RTD module in the parameter <ModuleType>.

The constant measurement current $I_M$ in amps [A] must be passed here to allow an exact calculation of the temperature. This must be previously measured with a high accuracy amp meter (see the ME-MultiSig system manual).

If the measurement tolerance is to be ignored, the function uses a typical constant measurement current $I_M = 500 \times 10^{-6}$ A. If this is the case, pass the constant ME4000_MULTISIG_I_MEASURED_DEFAULT.

- If you have chosen a RTD module in the parameter <ModuleType>:

  The constant measurement current $I_M$ in amps [A] must be passed here to allow an exact calculation of the temperature. This must be previously measured with a high accuracy amp meter (see the ME-MultiSig system manual).

  If the measurement tolerance is to be ignored, the function uses a typical constant measurement current $I_M = 500 \times 10^{-6}$ A. If this is the case, pass the constant ME4000_MULTISIG_I_MEASURED_DEFAULT.

- If you have chosen a thermocouple module in the parameter <ModuleType>:

  Because of the calculation of the temperature uses a reference to 0°C a cold junction compensation must be done. For this purpose the temperature at the connector of the expansion module must be measured and passed in this parameter (in °C). Please see the ME-MultiSig manual for details of order of operation. Paramter <Size> returns a pointer to the compensated temperature value in °C.
- In all other cases pass the constant: ME4000_VALUE_NOT_USED.

**<Size>**

Pointer to the calculated value in the matching physical dimension: [V], [A], [°C] (depending on the <ModuleType>).

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000MultiSigAIExtractValues

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to extract the values for a specific mux channel from the array of all measured values corresponding to the „Mux“ channel list. This function must be called separately for each channel whose values are to be extracted from the data buffer.

## ● Definitions

VC: me4000MultiSigAIExtractValues(unsigned int uiMuxChannelNumber, short* psAIBuffer, unsigned long ulAIDataCount, unsigned char *pucMuxChanList, unsigned int uiMuxChanListCount, short* psChanBuffer, unsigned long ulChanBufferSizeValues, unsigned long* pulChanDataCount);

LV: me4000LV_… (see me4000LV.h)

VB: me4000VB_… (see me4000.bas)

VEE: me4000VEE_… (see me4000VEE.h)

## ➔ Parameters

**<MuxChannelNumber>**

„Mux“ channel number within the „Mux“ chain whose values should be extracted; possible values: 0…255

**<AIBuffer>**

Pointer to a data buffer with all the acquired values.

**<AIDataCount>**

Number of values in the buffer <AIBuffer>.

**<MuxChanList>**

Pointer to the „Mux“ channel list, which has been passed by the function *…MultiSigAIConfig*.

**<MuxChanListCount>**

Number of channel list entries of <MuxChanList>.

**<ChanBuffer>**

Pointer to an array of values were the extracted values from the specified channel are stored.

**<ChanBufferSizeValues>**

Number of extracted values in the buffer <ChanBuffer>.

**<ChanDataCount>**

Pointer to a value containing the actual number of values stored in the array <ChanBuffer>. This value will never be larger than <ChanBufferSizeValues>, but could be smaller.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigAIGetNewValues

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to „retrieve“ measured values in the „MultiSig-AIContinuous“ operational mode. When the „MultiSig-AIScan“ mode of operation is being used, this function can be used to „look at“ the measured values during a data acquisition process running as a background operation (asynchronous). A typical use for this function would be to read in and display the measured values during a longer data acquisition process.

### ☞ Note

An example of the process flow can be found in the “Programming” section on page 39 and in the example programs provided in the ME-SDK.

### ● Definitions

VC: me4000MultiSigAIGetNewValues(unsigned int uiBoardNumber, short* psBuffer, unsigned long ulNumberOfValuesToRead, int iExecutionMode, unsigned long* pulNumberOfValuesRead, int* piLastError);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<Buffer>`**

Pointer to a data buffer where the newest data values are stored from the running data acquisition process. The function …*MultiSigAIDigitToSize* can be used to convert the raw data into voltage values.

**`<NumberOfValuesToRead>`**

Size of the data buffer given as the number of measured values. The size should be a multiple of the channel list size. If you pass the value „0“ here the parameter `<NumberOfValuesRead>` returns the number of values to be „retrieved“.

**<ExecutionMode>**

Choose the execution mode for this function:

- ME4000_AI_GET_NEW_VALUES_BLOCKING:
  The program flow is blocked until all measured values are retrieved.
- ME4000_AI_GET_NEW_VALUES_NON_BLOCKING:
  Only the currently available measured values are retrieved.

If you passed the value „0“ in parameter `<NumberOfValuesToRead>` this parameter is not relevant.

**<NumberOfValuesRead>**

Pointer which returns the actual number of measured values stored into the `<buffer>` defined above. This value will never be greater than the `<NumberOfValuesToRead>` parameter, but it can be smaller if not so much measured values are available.

**<LastError>**

This parameter contains the last error which occurred when calling this function. Possible errors are a FIFO overflow, or data buffer overflow. If no error occurred, a “0” (ME4000_NO_ERROR) is returned.

### ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigAIGetStatus

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to check whether a data acquisition process in the operation mode „MultiSig-AIScan“ in the execution mode “ASYNCHRONOUS” is still running or whether all the measured values have been retrieved.

The parameter `<WaitIdle>` can be used to return the status immediately or wait until the data acquisition process is ended.

## ● Definitions

VC: me4000MultiSigAIGetStatus(unsigned int uiBoardNumber, int iWaitIdle, int* piStatus);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<WaitIdle>`**

This parameter defines how the function returns:

- ME4000_AI_WAIT_NONE
  The status of the operation is returned immediately in the parameter `<Status>`.
- ME4000_AI_WAIT_IDLE
  The function returns when all measured values have been acquired. In that case, the parameter `<Status>` will always have the value ME4000_AI_STATUS_IDLE.

**`<Status>`**

The current status of the data acquisition:

- ME4000_AI_STATUS_IDLE
  The acquisition is ended.
- ME4000_AI_STATUS_BUSY
  The acquisition is still running.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigAIOpen

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function prepares the „Mux“ operation. The required hardware resources are reserved:

- Without opto-isolation: digital port A and B.
- With opto-isolation: digital port A and C.
- D/A channel 3 is disabled for analog output (only relevant for ME-4680 models).
- The A/D section is disabled for analog input using the „normal AI-functions“ (*me4000AI…*).

See also chapter 4.5 „ME-MultiSig Control“ on page 79 for more information.

### ● Definitions

VC: int me4000MultiSigAIOpen (unsigned int uiBoardNumber);
LV: me4000LV_… (see `me4000LV.h`)
VB: me4000VB_… (see `me4000.bas`)
VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigAIReset

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

The data acquisition process is stopped completely when this function is called. All measured values acquired up until this function is called are lost. The board must be configured again before another data acquisition process can be started (…*MultiSigAIConfig*, …*MultiSigAIScan*, and …*MultiSigAIContinuous*).

### ● Definitions

VC: me4000MultiSigAIReset (unsigned int uiBoardNumber);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigAIScan

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to prepare the software for a data acquisition process with a known number of measured values. A user defined data buffer is allocated where the measured values are stored. In the execution mode “BLOCKING”, the thread where the function …*MultiSigAIStart* was called does not return until the last measured value is retrieved. In the execution mode „ASYNCHRONOUS“ the process is started as a background operation. A new thread is created when the …*MultiSigAIStart* function is called. Other threads can be processed parallel to the running data acquisition. If desired (e. g. during a longer running data acquisition process) you can „look at“ the measured values during a running process. This can be done by a user defined callback function or by calling the function …*MultiSigAIGetNewValues*. When using the “Terminate” callback function it is possible to “report” the end of the data acquisition process to your application.

The function …*MultiSigAIStart* is always used to start the data acquisition process. It can be started immediately after the function is called (software start) or by an external trigger pulse (see …*MultiSigAIConfig*). It is possible to set a “Time-Out” when an external trigger is being used in case this does not occur. The data acquisition process is ended automatically when the last data value is acquired.

### ☞ Note

An example of the process flow can be found in chapter 4.1.3 "Timer Controlled „AI Operation Modes“" on page 34 and in the example programs provided in the ME-SDK.

### ● Definitions

Type definition for ME4000_P_AI_CALLBACK_PROC:

```
typedef void (_stdcall *
ME4000_P_AI_CALLBACK_PROC)(short* psValues,
unsigned int uiNumberOfValues, void* pCall-
backContext, int iLastError);
```

Type definition for ME4000_P_AI_TERMINATE_PROC:

```
typedef void (_stdcall *
    ME4000_P_AI_TERMINATE_PROC)(short*psValues,
    unsigned int uiNumberOfValues, void*
    pTerminateContext, int iLastError);
```

VC: me4000MultiSigAIScan(unsigned int uiBoardNumber, unsigned int uiNumberOfMuxLists, short* psBuffer, unsigned long ulBufferSizeValues, int iExecutionMode, ME4000_P_AI_CALLBACK_PROC pCallbackProc, void* pCallbackContext, unsigned int uiRefreshFrequency, ME4000_P_AI_TERMINATE_PROC pTerminateProc, void* pTerminateContext, unsigned long ulTimeOutSeconds);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<NumberOfMuxLists>**

Number of „Mux“ channel list processings.

**<Buffer>**

Pointer to a data buffer where the measured values are stored. The function …*MultiSigAIDigitToSize* can be used to conveniently convert the digit values to the matching physical size.

**<BufferSizeValues>**

The size of the data buffer given as the number of measured values.

**<ExecutionMode>**

Choose the execution mode for this function:

- ME4000_AI_SCAN_BLOCKING:
  The program is blocked until all measured values are acquired.
- ME4000_AI_SCAN_ASYNCHRONOUS:
  The following call of the function …*MultiSigAIStart* will automatically create a new thread so that the calling thread is not blocked.

**<CallbackProc>** **~~LV~~, ~~VB~~, ~~VEE~~**

This is the callback function that is regularly called during the data acquisition process. This function receives a pointer to the newly acquired data values and the number of data values. If this operation is not required, pass the constant ME4000_POINTER_NOT_USED.

**<CallbackContext>** **~~LV~~, ~~VB~~, ~~VEE~~**

This is a user defined pointer that is passed to the callback function. If no callback function is being used, pass the constant ME4000_POINTER_NOT_USED.

**<RefreshFrequency>**

Defines the number of channel list processings after which the ring buffer should be read. The value passed here serves as a guideline and can be fitted by the driver.

If the constant ME4000_VALUE_NOT_USED is passed, the driver will determine a useful value automatically.

**<TerminateProc>** **~~LV~~, ~~VB~~, ~~VEE~~**

„Terminate“-function that is executed when the data acquisition process is completed. The function is passed a pointer to the start of the data buffer and the number of values. If this operation is not required, pass the constant ME4000_POINTER_NOT_USED.

**<TerminateContext>** **~~LV~~, ~~VB~~, ~~VEE~~**

A user defined pointer that is passed to the “Terminate”-function. If the „Terminate“-function is not being used pass the constant ME4000_POINTER_NOT_USED.

**<TimeOutSeconds>**

Optional you can choose a time interval in seconds which specifies the time within the first trigger pulse must be detected. Otherwise the operation is cancelled. If you don´t want to use external trigger operation or no time-out value is set, pass the constant ME4000_VALUE_NOT_USED.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000MultiSigAISingle

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| ✔ | ✔ | ✔ | ✔ |

This function is used to convert a single value from one of the up to 256 channels in the “Mux” chain. The conversion start can be either by software or by an external trigger (analog or digital). No further functions for configuration and start are required.

## ☞ Note

The ME-4600 board always uses the input range ±10V in single ended operation.

This function is always run in "Blocking" mode, therefore the program flow is blocked until the function returns. This is normally only relevant when using an external trigger signal to start the acquisition.

**Note** that any gain previously set with the function …*MultiSigSetGain* is overwritten by this function.

## ● Definitions

VC: me4000MultiSigAISingle(unsigned int uiBoardNumber, unsigned int uiAIChannelNumber, unsigned int uiMuxChannelNumber, int iGain, int iTriggerMode, int iExtTriggerEdge, unsigned long ulTimeOutSeconds, short* psDigitalValue);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<AIChannelNumber>`**

A/D channel number the „Mux" chain was configured for. The channel number must correspond with solder bridge „A" on the master board; possible values: 0…31 (ME-4650/4660: 0…15)

**`<MuxChannelNumber>`**

„Mux" channel number within the „Mux" chain whose value is to be acquired; possible values: 0…255

**`<Gain>`**

Sets the gain factor for the channel group to which the desired „Mux" channel belongs. The gain factor set here is valid for all channels of the relevant channel group (a gain factor set by the function …*MultiSigSetGain* will be overwritten):

- ME4000_MULTISIG_GAIN_1
  Gain factor 1 (standard)
- ME4000_MULTISIG_GAIN_10
  Gain factor 10
- ME4000_MULTISIG_GAIN_100
  Gain factor 100

**<TriggerMode>**

Select the trigger event for the A/D-section:

- ME4000_AI_TRIGGER_SOFTWARE
  The conversion is done immediately after this function is called.
- ME4000_AI_TRIGGER_EXT_DIGITAL
  The process is ready for conversion after this function is called. The conversion is started by a digital trigger signal.
- ME4000_AI_TRIGGER_EXT_ANALOG (not ME-4650/4660)
  The process is ready for conversion after this function is called. The conversion is started by an analog trigger signal.

**<ExtTriggerEdge>**

Select the trigger edge for the A/D-section:

- ME4000_AI_TRIGGER_EXT_EDGE_RISING
  Start when a rising edge is detected.
- ME4000_AI_TRIGGER_EXT_EDGE_FALLING
  Start when a falling edge is detected.
- ME4000_AI_TRIGGER_EXT_EDGE_BOTH
  Start when a falling or rising edge is detected.
- ME4000_VALUE_NOT_USED
  No external trigger in use. See parameter <TriggerMode>.

**<TimeOutSeconds>**

Optional you can choose a time interval in seconds which specifies the time within the first trigger pulse must be detected. Otherwise the operation is cancelled. If you don´t want to use external trigger operation or no time-out value is set, pass the constant ME4000_VALUE_NOT_USED.

**<DigitalValue>**

Pointer to a signed 16 bit value. The function *…MultiSigAIDigitToSize* can be used to convert the digit value to the matching physical size.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigAIStart

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to „arm“ the data acquisition process. Depending on the configuration of the hardware and software, the process may start immediately after calling this function (software start) or it will wait for the selected type of external trigger (see chapter 3.3.4 on page 19). As long as not ended with the function …*MultiSigAIReset* the data acquisition process can be started again from the beginning by calling this function. The A/D section does not need to be configured.

When ending a process in the operation mode „MultiSig-AIContinuous“ or for ending a process started in the operation mode “MultiSig-AIScan” before it is completed, the functions …*MultiSigAIStop* and …*MultiSigAIReset* can be used.

### ● Definitions

VC: me4000MultiSigAIStart (unsigned int uiBoardNumber);LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigAIStop

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function stops the data acquisition immediately. The measured values acquired since the last "retrieval" are lost. The configuration of the board remains unchanged (the "mux" channel list, the timer settings etc). A new data acquisition process can be started with the function …*MultiSigAIStart* at any time.

### ● Definitions

VC: me4000MultiSigAIStop(unsigned int uiBoardNumber, int iReserved);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<Reserved>`**

This parameter is reserverd. Pass the value „0".

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

### 5.3.8.2 „Demux“ Functions

The following functions are used for analog output being done with the ME-MultiSig system in combination with a board from the ME-4600 series (see also chapter „Demux“ Operation on page 84 and the “ME-MultiSig” system manual).

## me4000MultiSigAOAppendNewValues

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used for continuously reloading the D/A FIFO during a running data output process. The process is stopped immediately with the functions …*MultiSigAOStop* or …*MultiSigAOReset*.

See also chapter 4.5.2 on page 84 and programming examples included with the ME-SDK.

### ☞ Note

It is not required that the same data buffer passed in the function …*MultiSigAOContinuous* must be used. This function always uses D/A channel 0 of the ME-4600.

### ● Definitions

VC: me4000MultiSigAOAppendNewValues(unsigned int uiBoardNumber, short* psBuffer, unsigned long ulNumberOfValuesToAppend, int iExecutionMode, unsigned long* pulNumberOfValuesAppended);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<Buffer>`**

Pointer to the data buffer with the values to be reloaded.

**<NumberOfValuesToAppend>**

Number of values in the data buffer. If you pass the value „0“ here the parameter <NumberOfValuesAppended> returns the number of values, for which is currently space in the data buffer.

**<ExecutionMode>**

Choose execution mode for this function:

- ME4000_AO_APPEND_NEW_VALUES_BLOCKING:
  The program is blocked until all values are written to the ring buffer.
- ME4000_AO_APPEND_NEW_VALUES_NON_BLOCKING:
  The program only “refills” the number of values which have currently room in the ring buffer.

If you passed the value „0“ in parameter <NumberOfValues-ToAppend> this parameter is not relevant.

**<NumberOfValuesAppended>**

The actual number of values written to the ring buffer. See also parameter <NumberOfValuesToAppend>.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000MultiSigAOClose

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | ✔ | ✔ | ✔ |

This function closes an operation that was opened with the function *…MultiSigAOOpen*. Hardware resources that were reserved are released again. See chapter 4.5 „ME-MultiSig Control“ on page 79 for more information.

## ● Definitions

VC: me4000MultiSigAOClose (unsigned int uiBoardNumber);
LV: me4000LV_… (see me4000LV.h)
VB: me4000VB_… (see me4000.bas)
VEE: me4000VEE_… (see me4000VEE.h)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000MultiSigAOConfig

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function configures the hardware for a timer controlled output in the operation modes „MultiSig-AOContinuous“ and „MultiSig-AO-Wraparound“. The output is always started by calling the function *…MultiSigAOStart*. It can start immediately (software start) or by an external trigger pulse.

A 32 bit counter serves as the time base. It is supplied by a clock frequency of 33 MHz which gives a period of 30.30ns as the smallest possible time unit. This is referred to as “1 Tick” in the following sections. The sample rate for the analog output must be a multiple of “1 Tick” and is passed in the parameter <Ticks>. The sample rate can therefore be set in increments of 30.30ns between the minimum and maximum sample rate. The minimum sample rate is 0.5 samples per minute. The maximum sample rate 500 kS/s for not opto-isolated versions and 172 kS/s for opto-isolated versions.

## ☞ Note

The functions *…FrequencyToTicks* and *…TimeToTicks* allow convenient conversion of frequency resp. period time to “Ticks” which can be passed to the timer (see page 103).

The D/A FIFO 0 is always used for the analog values to be output. (see also chapter 4.5 „ME-MultiSig Control“ on page 79).

## ● Definitions

VC: me4000MultiSigAOConfig(unsigned int uiBoardNumber, unsigned char *pucDemuxChanList, unsigned int uiDemuxChanListCount, unsigned long ulTicks, int iTriggerMode, int iExtTriggerEdge);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<DemuxChanList>`**

Pointer to the beginning of the „Demux“ channel list controlling the demultiplexers. Therefore allocate a buffer of defined size to which the „Demux“ channel numbers (0…31) must be written in the desired order.

**`<DemuxChanListCount>`**

Number of „Demux“ channel list entries.

**`<Ticks>`**

Number of Ticks for the D/A timer (32 bit) which sets the sample rate for timer controlled ouput. The value range is between 66 (42Hex) for not opto-isolated versions resp. 192 (C0Hex) for opto-isolated versions and $2^{32}$-1 (FFFFFFFFHex) Ticks.

**`<TriggerMode>`**

Trigger event to start the analog output:

- ME4000_AO_TRIGGER_SOFTWARE
  Start by software after calling the function *…MultiSigAOStart*.
- ME4000_AO_TRIGGER_EXT_DIGITAL
  Ready for output after calling the function *…MultiSigAOStart*. The output is started when the external trigger signal is detected on pin 47 (DA_TRIG_0).

**<ExtTriggerEdge>**

Select the trigger edge for the trigger input DA_TRIG_0.

- ME4000_AO_TRIGGER_EXT_EDGE_RISING
  Starts the output on a rising edge.
- ME4000_AO_TRIGGER_EXT_EDGE_FALLING
  Starts the output on a falling edge.
- ME4000_AO_TRIGGER_EXT_EDGE_BOTH
  Starts the output on a rising or falling edge.
- ME4000_VALUE_NOT_USED
  No external trigger in use. See parameter <TriggerMode>.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000MultiSigAOContinuous

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to prepare the operation mode “MultiSig-AO-Continuous”. Any signal shape can be output and the signals can also be changed while the output process is running (opposite to the operation mode “MultiSig-AOWraparound”). The timer provides a fixed time base (sample rate) for putting out the values (see function *…MultiSigAOConfig*). A data buffer must be allocated and loaded with the initial values for output. Use the function *…MultiSigAOAppendNewValues* for continuously reloading new data values. This can be done with or without a callback function.

The output is started by calling the function *…MultiSigAOStart*. The output either starts immediately by calling this function (software start) or when an external trigger pulse is detected (see function *…MultiSigAOConfig*). The function *…MultiSigAOStop* is used to end the data output process. If the operation mode was not changed, a new output process can be started again from the beginning any time by calling the function *…MultiSigAOStart*. If the *…MultiSigAOReset* function is used to end the data output process, the D/A FIFO is also cleared and the process is completely terminated.

## ☞ Note

The order of values in the output buffer `<Buffer>` must correspond exactly with the order of the channels in the „Demux“ channel list (see function …*MultiSigAOConfig*).

## ● Definitions

Type definition for ME4000_P_AO_CALLBACK_PROC:

```
typedef void (_stdcall *
ME4000_P_AO_CALLBACK_PROC)
(unsigned long ulBufferAvailable,
void* pCallbackContext);
```

VC: me4000MultiSigAOContinuous(unsigned int uiBoardNumber, short* psBuffer, unsigned long ulDataCount, ME4000_P_AO_CALLBACK_PROC pCallbackProc, void* pCallbackContext, unsigned long ulTimeOutSeconds, unsigned long* pulNumberOfValuesWritten);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<Buffer>`**

Pointer to an user defined data buffer, which is „filled“ with the **initial** values to be output.

**`<DataCount>`**

Number of values in data buffer.

**`<CallbackProc>`** **~~LV~~, ~~VB~~, ~~VEE~~**

Callback function which is called at regular intervals to reload the data buffer. If this operation is not required, pass the constant ME4000_POINTER_NOT_USED.

**`<CallbackContext>`** **~~LV~~, ~~VB~~, ~~VEE~~**

This is a user defined pointer that can be passed to the callback function. If no callback function is being used, pass the constant ME4000_POINTER_NOT_USED.

**`<TimeOutSeconds>`**

Optional you can choose a time interval in seconds which specifies the time within the first trigger pulse must be detected. Otherwise the operation is cancelled. If you don´t want to use external trigger operation or no time-out value is set, pass the constant ME4000_VALUE_NOT_USED.

**<NumberOfValuesWritten>**

The number of values actual written into the data buffer.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000MultiSigAOGetStatus

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to check whether a data output operation in the „MultiSig-AOContinuous“ or „MultiSig-AOWraparound“ modes is still running or whether the ring buffer has “run dry”. This can happen when the buffer is intentionally allowed to run dry by not reloading any new data values to end the output process or when the values could not reloaded quickly enough because the PC performance is too slow.

The parameter <WaitIdle> can control whether the function returns the status immediately or whether it waits until the output process is ended.

## ● Definitions

VC: me4000MultiSigAOGetStatus(unsigned int uiBoardNumber, int iWaitIdle, int* piStatus);

LV: me4000LV_… (see me4000LV.h)

VB: me4000VB_… (see me4000.bas)

VEE: me4000VEE_… (see me4000VEE.h)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<WaitIdle>**

This parameter defines the “return behaviour“ of this function:

- ME4000_AO_WAIT_NONE
  The current status is returned immediately in the parameter <Status>.
- ME4000_AO_WAIT_IDLE
  The function returns when the data output process is ended (FIFO is empty). The parameter <Status> always returns the constant ME4000_AO_STATUS_IDLE.

**<Status>**

The current status:

- ME4000_AO_STATUS_IDLE
  The output is ended, i. e. the ring buffer is empty.
- ME4000_AO_STATUS_BUSY
  The output is still running.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000MultiSigAOOpen

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | ✔ | ✔ | ✔ |

This function prepares the “Demux” operation. The required hardware resources are reserved:

- Digital port A
- D/A channel 0 is used for output of the analog values.
- D/A channel 3 is disabled for analog output (only relevant for ME-4680 models).

See also chapter 4.5 „ME-MultiSig Control“ on page 79 for more information.

## ● Definitions

VC: me4000MultiSigAOOpen (unsigned int uiBoardNumber);
LV: me4000LV_… (see me4000LV.h)
VB: me4000VB_… (see me4000.bas)

VEE: me4000VEE_... (see me4000VEE.h)

### ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigAOReset

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function ends the currently running „MultiSigAOContinuous“or „MultiSig-AOWraparound“ output process. The output is stopped completely and immediately. The corresponding ring buffer is deleted and the „Demux“ channel 0 is set to 0V.

### ● Definitions

VC: me4000MultiSigAOReset(unsigned int uiBoardNumber);

LV: me4000LV_... (see me4000LV.h)

VB: me4000VB_... (see me4000.bas)

VEE: me4000VEE_... (see me4000VEE.h)

### ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigAOSingle

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | ✔ | ✔ | ✔ |

This function serves to output a single voltage value to the specified „Demux“ channel. No further configuration with other functions is required.

### ● Definitions

VC: me4000MultiSigAOSingle(unsigned int uiBoardNumber, unsigned int uiDemuxChannelNumber, int iTriggerMode, int iExtTriggerEdge, unsigned long ulTimeOutSeconds, short sValue);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**`<DemuxChannelNumber>`**

„Demux“ channel number to which should be output; possible values: 0…31

**`<TriggerMode>`**

Trigger event to start the analog output:

- ME4000_AO_TRIGGER_SOFTWARE
  A single value is output immediatly after calling this function.
- ME4000_AO_TRIGGER_EXT_DIGITAL
  Ready for output after calling this function. The output starts when the matching trigger signal is detected on pin 47 (DA_TRIG_0).

**<ExtTriggerEdge>**

Select the trigger edge for the trigger input DA_TRIG_0.

- ME4000_AO_TRIGGER_EXT_EDGE_RISING
  Starts on a rising trigger edge.
- ME4000_AO_TRIGGER_EXT_EDGE_FALLING
  Starts on a falling trigger edge.
- ME4000_AO_TRIGGER_EXT_EDGE_BOTH
  Starts the output on a rising or falling edge.
- ME4000_VALUE_NOT_USED
  No external trigger in use. See parameter <TriggerMode>.

**<TimeOutSeconds>**

Optional you can choose a time interval in seconds which specifies the time within the first trigger pulse must be detected. Otherwise the operation is cancelled. If you don´t want to use external trigger operation or no time-out value is set, pass the constant ME4000_VALUE_NOT_USED.

**<Value>**

16 bit output value. The value range is between -32768 (-10V) and +32767 (+10V - LSB)

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000MultiSigAOStart

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to start the output in the „MultiSig-AOContinuous“ and „MultiSig-AOWraparound“ modes of operation. If the external trigger operation is chosen, the output process starts when the selected trigger edge is detected on the external trigger input DA_TRIG_0 (pin 47).

## ● Definitions

VC: me4000MultiSigAOStart (unsigned int uiBoardNumber);
LV: me4000LV_… (see me4000LV.h)
VB: me4000VB_… (see me4000.bas)

VEE: me4000VEE_... (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

### ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

## me4000MultiSigAOStop

### ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to end an analog output operation process started in the „MultiSig-AOContinuous“ or „MultiSig-AOWraparound“ modes of operation. In the „MultiSig-AOContinuous“ mode of operation, the data output is ended completely and immediately. The demultiplexer control stops with the currently set „Demux“ channel and 0V is output. In the „MultiSig-AOWraparound“ mode of operation the parameter `<StopMode>` is used to define whether the operation is stopped immediately (see „MultiSig-AOContinuous“) or whether to continue until the last channel in the „Demux“ channel list is set. As long as the operation mode was not changed, a new output process can be started again from the beginning by calling the function *…MultiSigAOStart*.

### ● Definitions

VC: me4000MultiSigAOStop(unsigned int uiBoardNumber, int iStopMode);

LV: me4000LV_... (see `me4000LV.h`)

VB: me4000VB_... (see `me4000.bas`)

VEE: me4000VEE_... (see `me4000VEE.h`)

### ➔ Parameters

**`<BoardNumber>`**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<StopMode>**

- ME4000_AO_STOP_MODE_LAST_VALUE
  The output process is stopped when the last value in the „Demux“ channel list is output (only relevant in the “MultiSig-AOWraparound” mode).
- ME4000_AO_STOP_MODE_IMMEDIATE
  The output is stopped immediately and the output channel is set to 0V.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000MultiSigAOVoltToDigit

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | ✔ | ✔ | ✔ |

This function allows the simple conversion of the voltage values [V] to be output into digit values [Digits]. The voltage can be output in steps of 0.3 mV = 1 Digit. Using this function is optional.

For a output voltage range of ±10 V the following formula is valid (characteristic curve of the D/A converter see diagram 12 on page 22):

$$U[\text{Digits}] = \frac{32768}{10\text{V}} \cdot U[\text{V}]$$

## ● Definitions

VC: me4000MultiSigAOVoltToDigit(double dVolt, short* psDigit);
LV: me4000LV_... (see `me4000LV.h`)
VB: me4000VB_... (see `me4000.bas`)
VEE: me4000VEE_... (see `me4000VEE.h`)

## ➔ Parameters

**<Volt>**

Voltage value to be output in volts.

**<Digit>**

Pointer to the digit value to be output.

## ‹ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# me4000MultiSigAOWraparound

## ✎ Description

| ME-4650 | ME-4660 | ME-4670 | ME-4680 |
|---|---|---|---|
| – | – | – | ✔ |

This function is used to prepare the operation mode “MultiSig-AO-Wraparound”. Periodic signals of all types can be output using this function. D/A FIFO 0 must be loaded once before the output process can start. Create a data buffer of defined size which contains the values to be output.

The timer provides a fixed time base (sample rate) for the output process (see function *…MultiSigAOConfig*).

The output is started by calling the function *…MultiSigAOStart*. The output either starts immediately (software start) or when the external trigger is detected (see function *…MultiSigAOConfig*). The function *…MultiSigAOStop* is used to end the data output process either immediately or it can be stopped with the last value in the „Demux“ channel list i. e. with a known „Demux“ channel. If the operational mode was not changed, a new output process can be started again from the beginning any time by calling the *…MultiSigAOStart* function. If the *…MultiSigAOReset* function is used to end the data output process, the FIFOs are also cleared. This terminates the process **completely**.

## ☞ Note

The order of the values in the output buffer `<Buffer>` must correspond exactly with the order of the channels in the „Demux“ channel list (see function *…MultiSigAOConfig*).

If the data buffer is less than 4096 values and the output operation is configured for “infinite”, the entire process runs on the firmware level. The host PC is not loaded by the running output process!

## ● Definitions

Type definition for ME4000_P_AO_TERMINATE_PROC:

```
typedef void (_stdcall *
ME4000_P_AO_TERMINATE_PROC)
(void* pTerminateContext);
```

VC: me4000MultiSigAOWraparound(unsigned int uiBoardNumber, short* psBuffer, unsigned long ulDataCount, unsigned long ulLoops, int iExecutionMode, ME4000_P_AO_TERMINATE_PROC pTerminateProc, void* pTerminateContext, unsigned long ulTimeOutSeconds);

LV: me4000LV_… (see `me4000LV.h`)

VB: me4000VB_… (see `me4000.bas`)

VEE: me4000VEE_… (see `me4000VEE.h`)

## ➔ Parameters

**<BoardNumber>**

Number of the board to be accessed of type ME-46xx or ME-6x00 (0…31)

**<Buffer>**

Pointer to a user allocated data buffer with the voltage values to be output.

**<DataCount>**

Number of values in data buffer `<Buffer>`

**<Loops>**

This parameter defines how often the values in the data buffer `<Buffer>` are to be output. For a infinite operation, pass the constant: ME4000_AO_WRAPAROUND_INFINITE.

**<ExecutionMode>**

Choose the execution mode for this function:

- ME4000_AO_WRAPAROUND_BLOCKING:
  The program flow is blocked until the data output process is ended. It is not possible to use this mode when the `<Loops>` parameter (described above) is set to “infinite“.
- ME4000_AO_WRAPAROUND_ASYNCHRONOUS:
  The output is handled as a background operation (asynchronous). The program flow is not blocked.

**<TerminateProc>** **~~LV~~, ~~VB~~, ~~VEE~~**

„Terminate“-function that is executed when the output process is completed. The function is passed a pointer to the start of the data buffer and the number of values. If this operation is not required, pass the constant ME4000_POINTER_NOT_USED.

**<TerminateContext>** **~~LV~~, ~~VB~~, ~~VEE~~**

User defined pointer passed to the „Terminate“-function. If the “Terminate”-function is not used, pass the constant ME4000_POINTER_NOT_USED.

**<TimeOutSeconds>**

Optional you can choose a time interval in seconds which specifies the time within the first trigger pulse must be detected. Otherwise the operation is cancelled. If you don´t want to use external trigger operation or no time-out value is set, pass the constant ME4000_VALUE_NOT_USED.

## ❮ Return value

If the function is successfully executed, a '0' (ME4000_NO_ERROR) is returned. If an error occurs, an error code unequal to '0' is returned. The cause of the error can be determined with the functions for error handling.

# Appendix

## A Specifications

(Ambient temperature 25°C)

### PCI Interface

Standard PCI board (32 Bit, 33MHz, 5V);
PCI Local Bus Specification Version 2.1 compliant;
Resources assigned automatically (Plug&Play)

### Voltage Inputs

| | |
|---|---|
| Total Number of A/D-channels | |
| | ME-4650/4660: 16 single ended<br>ME-4670/4680: 32 single ended/<br>16 differential |
| „Sample&Hold“ channels | optional: 8 single-ended simultanously sampling |
| A/D converter | 500kHz 16 bit A/D converter |
| Input ranges | unipolar: 0…+2.5V - 1LSB (38µV);<br>0V…+10V - 1LSB (152µV);<br>bipolar: -2,5V…+2,5V - 1LSB (76µV);<br>-10V…+10V - 1LSB (305µV) |
| Full scale error | unipolar: 0V+10LSB, +FS-10LSB<br>bipolar: -FS+10LSB, +FS-10LSB |
| Inputs protected up to | ±15 V |
| Input impedance | $R_{IN}$ = typ. 600 MΩ; $C_{IN}$ = typ. 3 pF<br>Channels with „sample & hold“ option:<br>$R_{IN}$ = typ. 1 MΩ; $C_{IN}$ = typ. 5 pF |
| Total sampling rate | 500 kS/s |
| Sampling rate for „Sample&Hold“ operation („s“-versions): | |
| | Number of channel list entries x<br>CHAN time + relaxation time |
| Relaxation time (S&H) | 1.5 µs |
| Total accuracy | typ. ±4 LSB, max. ±10 LSB fullscale in input range ±10 V |
| Opto-isolation („i“-versions) | |
| | A/D and D/A section with common ground (A_GND); isolated from PC ground and the rest of the board |
| A/D FIFO | 2 k values FIFO |
| Channel list | max. 1024 entries (channel number, gain, uni/bipolar, single ended/differential) |

| | |
|---|---|
| Smallest time unit for CHAN and SCAN timer: | |
| | 1 Tick ≙ 30.30 ns ≙ 33 MHz |
| CHAN timer | 32 bit counter determines the time between two consecutive channel list entries: programmable in steps of 30.$\overline{30}$ns from 2µs up to ~130s. |
| SCAN timer | 36 bit counter determines the time between two consecutive channel list processings. Useful SCAN time range (min. 2 channels): programmable in steps of 30.$\overline{30}$ns from 4µs up to ~30 minutes. |
| Operation modes | „AISingle“, „AIContinuous“, „AIScan“ optional: „AISimultaneous“ |
| Acquisition modes | „Software Start“, „External Standard“, „External Single Value“, „External Channel List“ |
| External trigger modes | ext. analog trigger; ext. digital trigger |
| External trigger edges | rising, falling, both |

***External Trigger without Opto-isolation***

| | | |
|---|---|---|
| Reference to ground | | PC ground (PC_GND) |
| Input level | $U_{IL}$: | max. 0.9V at Vcc=4.5V |
| | $U_{IH}$: | min. 3.15V at Vcc=4.5V |
| Delay time | | max. 30ns |

***External Trigger with Opto-isolation***

| | |
|---|---|
| Reference to ground | Digital I/O ground (DIO_GND) |
| Input current $I_F$ | 7.5mA ≤ $I_F$ ≤ 10mA |
| Voltage level | typ. 5V |
| Delay time | typ. 80ns |

## Voltage Outputs (ME-4660, ME-4670, ME-4680)

| | |
|---|---|
| Number of channels | ME-4660: 2;<br>ME-4670/4680: 4 |
| D/A converter | 1 serial converter per channel |
| Resolution | 16 bit |
| Output range | ±10V |
| Output current | max. 5mA per channel |
| Settling time (DAC) | max. 2µs at fullscale (-10V → +10V) |
| Total accuracy: | max. ±10mV |
| Opto-isolation („i“-versions) | |
| | A/D and D/A section with common ground (A_GND); isolated from PC ground and the rest of the board |

| | |
|---|---|
| Operation modes | „AOSingle“: single value output;<br>„AOSimultaneous“: single value to several channels simultaneously |
| Trigger modes | Software start, ext. digital trigger |
| External trigger edges | rising, falling, both |

***Timer controlled Output (only ME-4680)***

| | |
|---|---|
| Channels | 0…3 (independent of each another) |
| D/A FIFOs | 4k values per channel |
| Sample rate | max. 500kS/s |
| D/A Timer | programmable from 2µs up to 130s in steps of $30.\overline{30}$ns |
| Operation modes | „AOContinuous“: continuous output;<br>„AOWraparound“: periodical output |
| Trigger modes | Software start, ext. digital trigger, synchronous start (software/external) |
| External trigger edges | rising, falling, both |

***External Trigger without Opto-isolation***

| | | |
|---|---|---|
| Reference to ground | | PC ground (PC_GND) |
| Input level | $U_{IL}$: | max. 0.9V bei Vcc = 4.5V |
| | $U_{IH}$: | min. 3.15V bei Vcc = 4.5V |
| Delay time | | max. 30ns |

***External Trigger with Opto-isolation***

| | |
|---|---|
| Reference to ground | Digital I/O ground (DIO_GND) |
| Input current $I_F$ | $7.5mA \leq I_F \leq 10mA$ |
| Voltage level | typ. 5V |
| Delay time | typ. 80ns |

## Digital-I/Os

| | |
|---|---|
| Ports | 4 x 8 bit |

***…without opto-isolation***

| | | |
|---|---|---|
| Reference to ground | | PC ground (PC_GND) |
| Port type | | bi-directional TTL ports |
| Output level | $U_{OL}$: | max. 0.5V bei 24mA |
| | $U_{OH}$: | min. 2.4V bei -24mA |
| Input level | $U_{IL}$: | max. 0.8V bei Vcc = 5V |
| | $U_{IH}$: | min. 2V bei Vcc = 5V |
| | Input current: | ±1µA |
| Sample rate | | max. 500kS/s (2µs) |

***...with opto-isolation („i"-versions):***

| | | |
|---|---|---|
| Reference to ground | | Digital I/O ground (DIO_GND); isolated from PC ground and the rest of the board |
| Port type | Port A: | Output port |
| | Port B: | Input port |
| | Port C, D: | Bi-directional TTL ports (refer to the levels „without opto-isolation") |
| Output levels port A, B: | | |
| | $U_{max}$: | 42V (depends on external supply) |
| | $I_{Out}$: | max. 30mA |
| Input levels port A,B: | | |
| | Input current $I_F$: | $7.5mA \leq I_F \leq 10mA$ |
| | $U_{IL}$: | max. 0.8V |
| | $U_{IH}$: | min. 4.5V, max. 5V (optional higher input voltages possible - please contact our support division) |
| Sample rate | | max. 172kS/s (5.8µs) |

## Bit Pattern Output

| | |
|---|---|
| Ports | flexible port mapping to all digital output ports (A, B, C, D) |
| Operation modes | „BitPattern-Continuous", „BitPattern-Wrap-around" |
| Bit pattern FIFO | 4 k values (shared with D/A FIFO 3) |
| Sample rate | |
| TTL port: | max. 500kS/s (2µs) |
| Opto-isolated port: | max. 172kS/s (5.8µs) |
| Bit pattern timer | programmable from 2µs up to 130s in steps of 30.$\overline{30}$ns |
| External trigger | Digital trigger input DA_TRIG_3 |
| Input level: | see external trigger D/A section |
| Delay time: | without opto-isolation: max. 30ns<br>with opto-isolation: typ. 80ns |
| Trigger modes | Software start, ext. digital trigger |
| External trigger edges | rising, falling, both |

## Counter

| | |
|---|---|
| Number | 3 x 16 bit (1 x 82C54) |
| Clock source | external up to 10 MHz |

***...without opto-isolation***

| | | |
|---|---|---|
| Reference to ground | | PC ground (PC_GND) |
| Level for counter output (OUT_*x*) | | |
| | $U_{OL}$: | max. +0.45V ($I_{OL}$=+7.8mA) |
| | $U_{OH}$: | min. +2.4V ($I_{OH}$=-6mA) |

Level for counter inputs (CLK_*x*, GATE_*x*)

| | |
|---|---|
| $U_{IL}$: | -0.5V…+0.8V ($I_{ILmax}$=±10µA) |
| $U_{IH}$: | +2.2V…+6V ($I_{IHmax}$=±10µA) |

***…with opto-isolation („i"-versions):***

| | |
|---|---|
| Reference to ground | Counter ground (CNT_GND); isolated from PC ground and the rest of the board |

External supply for opto-couplers (CNT_VCC_IN)
+5V/30mA

Level for counter outputs (OUT_*x*):

| | |
|---|---|
| $U_{max}$: | 42V |
| $I_{Out}$: | max. 30mA |

Level for counter inputs (CLK_*x*, GATE_*x*):

| | |
|---|---|
| $I_F$: | min. 7.5mA |
| $U_{IL}$: | max. 0.8V |
| $U_{IH}$: | min. 4.5V, max. 5V (optional higher input voltages possible - please contact our support division) |

***Optional***: Supplying the opto-couplers with VCC from analog section (A_VCC). Note that the electrical isolation between analog and counter section will be removed (CNT_GND = A_GND), see diagram 19.

## External Interrupt

| | |
|---|---|
| Ext. interrupt input | directly sent to the system (if enabled by the driver) |
| Reference to ground | „TTL": PC ground (PC_GND); „Opto": Digital I/O ground (DIO_GND) |
| Input level | see digital I/Os |

## General Information

DC/DC converter A/D section ±5V and ±15 V (2 x 3W)
Power consumption (without external load):

| | |
|---|---|
| „Without opto-isolation" | typ. 2.8A |
| „With opto-isolation" | typ. 2.8A |
| Load for VCC_OUT | max. 200mA |
| Physical size | 175mm x 107mm (without mounting bracket and connector) |
| Connectors | 78pin D-Sub female connector (ST1); 20pin IDC connector (ST2) |
| Operating temperature | 0…70°C |
| Storage temperature | -40…100 °C |
| Relative humidity | 20…55% (non condensing) |

**CE Certification**

| | |
|---|---|
| EMC Directive | 89/336/EMC |
| Emission | EN 55022 |
| Noise immunity | EN 50082-2 |

# B Pinout

## Legend for pinouts:

| | |
|---|---|
| AD_*x* | Analog input channels |
| AD_TRIG_D | Digital trigger input for A/D section |
| AD_TRIG_A+ | Analog trigger input for A/D section (positive comparator input) |
| AD_TRIG_A- | Analog trigger input for A/D section (negative comparator input) |
| DA_*x* | Analog output channels |
| DA_TRIG_*x* | Digital trigger input separately for each D/A channel |
| DIO_A*x* | Digital-I/O port A |
| DIO_B*x* | Digital-I/O port B |
| DIO_C*x* | Digital-I/O port C |
| DIO_D*x* | Digital-I/O port D |
| EXT_IRQ | External interrupt input |
| CLK_*x* | Clock input for counter |
| GATE_*x* | Gate input for counter |
| OUT_*x* | Counter output |
| PC_GND | ***Not-optoisolated*** models: Common ground of all functional groups (= PC ground). |
| VCC_OUT | ***Not-optoisolated*** models: $V_{CC}$ output (+5V from PC) max. 200mA load |
| n.c. | Pin not connected |

***Valid for optoisolated models:***

| | |
|---|---|
| A_GND | Ground for A/D and D/A section |
| DIO_GND | Ground for digital-I/O section |
| CNT_GND | Ground for counter section |
| CNT_VCC_IN | Default: Input for external power supply (+5V±10%) for the optocouplers of the counters. |
| A_VCC | Optional (see diagram 19 on page 27): Sourcing the optocouplers of the counters by the analog section (A_VCC). **No external switching to pin 1!** |

# B1 78pin D-Sub Connector ME-4600 (ST1)

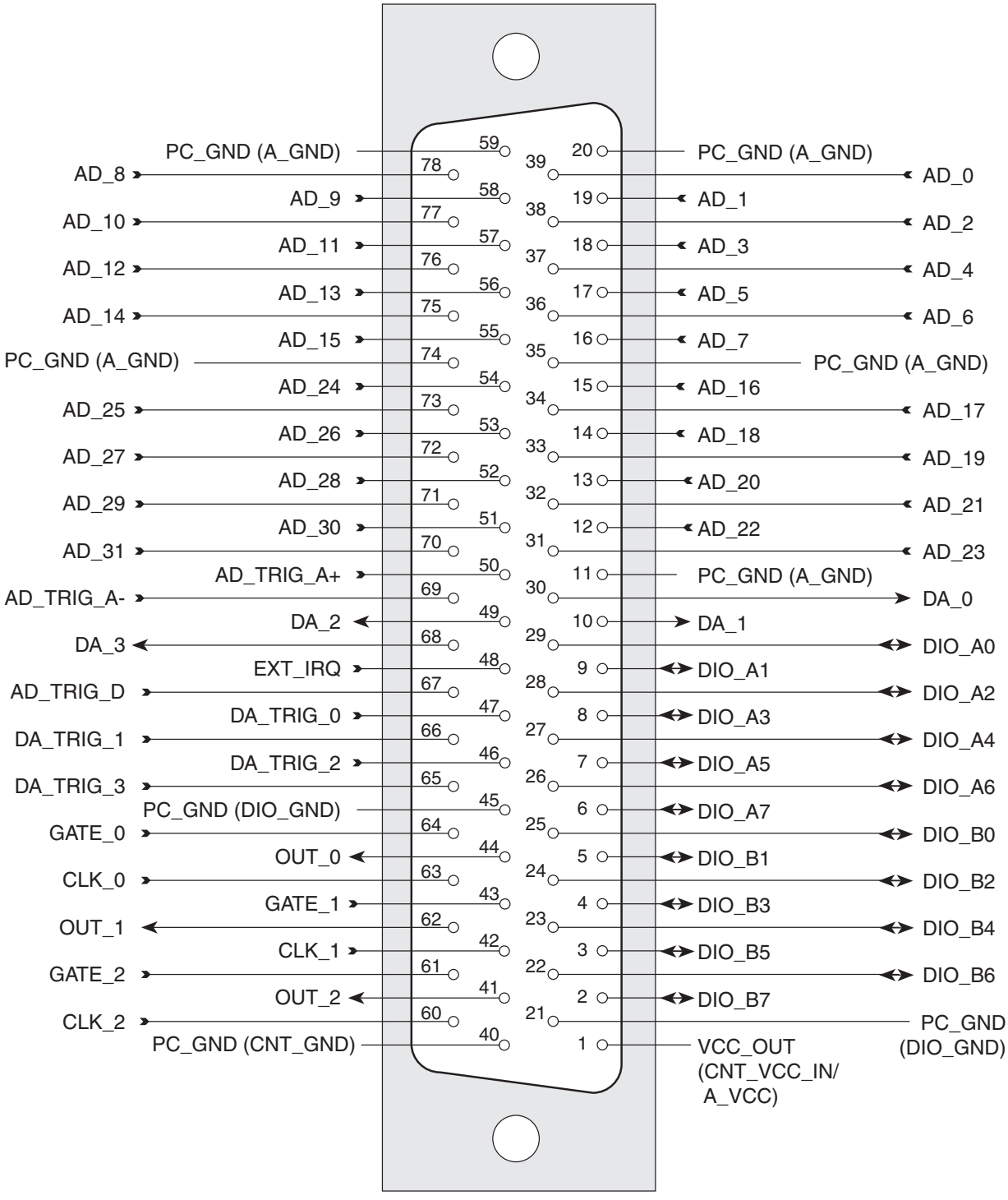



*Diagram 58: Pinout of the 78pin D-Sub female connector (ST1)*

Depending on the model not all pins of the 78pin D-Sub connector are connected. The labels in brackets concern the opto-isolated versions („i“-versions).

## B2 Auxiliary Connector (ST2)

**ME-AK-D25F/S**: Adapter cable from 20pin IDC connector to mounting bracket with 25pin D-Sub female connector (comes with the board).

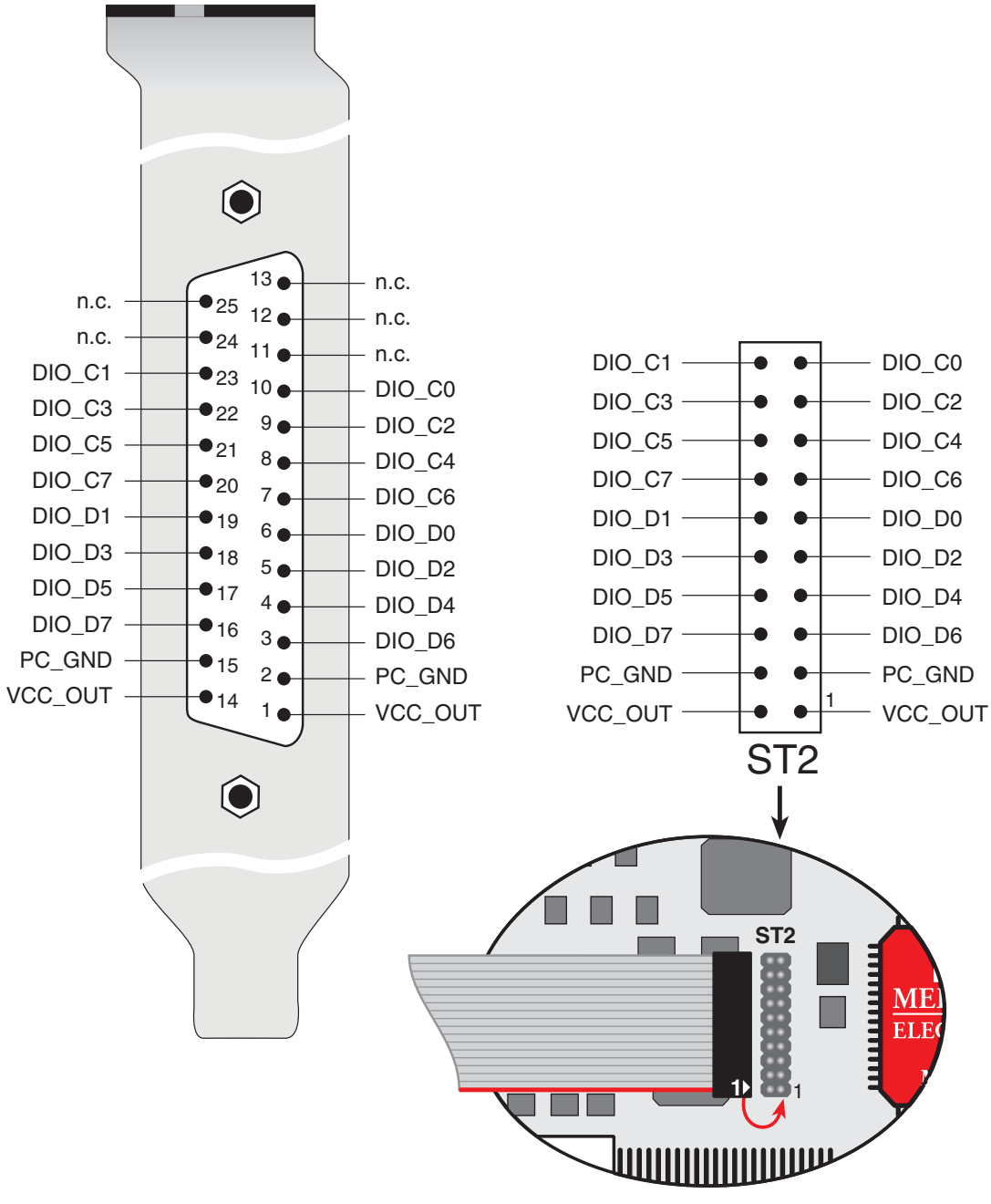



*Diagram 59: Auxiliary connector ST2 for ME-4600 (top view)*

**Note:** Connect the mounting bracket pin 1 of the flat ribbon cable (red marked line) as shown above to the IDC connector ST2.

# C Accessories

We recommend to use high quality connector cables with single shielded lines per channel.

**ME-AK-D78/4000M-F**

Special connector cable (1:1) for ME-4600 series (78pin D-Sub male connector to 78pin D-Sub female connector), length: 1 m

**ME-AB-D78M**

78pin D-Sub connector block (male connector)

**ME-AA4-3(i)**

Connector adapter for adaption of a ME-2x00/3000 application to a board of the ME-4600 series („i"-version for opto-isolated models including opto-isolation for digital port C and D).

**ME-MultiSig System**

Extended multiplex and signal conditioning system:

- Analog multiplexing up to 8192 channels (timer controlled up to 256 channels)
- Analog demultiplexing up to 32 channels
- Signal conditioning (voltage, current, RTDs)

**ME-63Xtend Series**

External relay and digital I/O boards (DIN rail mounting possible). Connection by ST2 with additional mounting bracket ME AK-D25F/S and special connection cable ME AK-D2578/4000.

**ME-UB Series**

Desktop relay and digital I/O boxes. Connection by ST2 with additional mounting bracket ME AK-D25F/S and special connection cable ME AK-D2515/4000.

For further information on accessories please refer to the current Meilhaus catalog.

# D Technical Questions

## D1 Hotline

If you should have any technical questions or problems with the board hardware or the driver software, please send a fax to our hotline:

**Fax hotline**: ++ 49 (0) 89/89 01 66 28
**eMail**: support@meilhaus.de

Please give a full description of the problems and as much information as possible, including operating system information.

## D2 Service address

We hope that your board will never need to be repaired. If this should become necessary please contact us at the following address:

**Meilhaus Electronic GmbH**

*Service Department*
Fischerstraße 2
D-82178 Puchheim/Germany

If you would like to send a board to Meilhaus Electronic for repair, please do not forget to add a full description of the problems and as much information as possible, including operating system information.

## D3 Driver Update

The current driver versions for Meilhaus boards and our manuals in PDF format are available under www.meilhaus.com.

# E Constant Definitions

**Note:** The following constant definitions are valid for Windows. Please note also the current definition file (`me4000defs.h`) included with the Meilhaus Developer Kit (ME-SDK). The Linux driver uses its own constant definitions (see Linux driver).

| Constant | Value |
|---|---|
| ***General*** | |
| ME4000_MAX_DEVICES | 32 |
| ME4000_VALUE_NOT_USED | 0 |
| ME4000_POINTER_NOT_USED | NULL |
| ME4000_NO_ERROR | 0x00000000 |
| ***Error Handling*** | |
| ME4000_ERROR_DEFAULT_PROC_ENABLE | 0x00010101 |
| ME4000_ERROR_DEFAULT_PROC_DISABLE | 0x00010102 |
| ***Analog Input*** | |
| ME4000_AI_ACQ_MODE_SOFTWARE | 0x00020101 |
| ME4000_AI_ACQ_MODE_EXT | 0x00020102 |
| ME4000_AI_ACQ_MODE_EXT_SINGLE_VALUE | 0x00020103 |
| ME4000_AI_ACQ_MODE_EXT_SINGLE_CHANLIST | 0x00020104 |
| ME4000_AI_RANGE_BIPOLAR_10 | 0x00020201 |
| ME4000_AI_RANGE_BIPOLAR_2_5 | 0x00020202 |
| ME4000_AI_RANGE_UNIPOLAR_10 | 0x00020203 |
| ME4000_AI_RANGE_UNIPOLAR_2_5 | 0x00020204 |
| ME4000_AI_INPUT_SINGLE_ENDED | 0x00020301 |
| ME4000_AI_INPUT_DIFFERENTIAL | 0x00020302 |
| ME4000_AI_TRIGGER_SOFTWARE | 0x00020401 |
| ME4000_AI_TRIGGER_EXT_DIGITAL | 0x00020402 |
| ME4000_AI_TRIGGER_EXT_ANALOG | 0x00020403 |
| ME4000_AI_TRIGGER_EXT_EDGE_RISING | 0x00020501 |
| ME4000_AI_TRIGGER_EXT_EDGE_FALLING | 0x00020502 |
| ME4000_AI_TRIGGER_EXT_EDGE_BOTH | 0x00020503 |
| ME4000_AI_SIMULTANEOUS_DISABLE | 0x00020601 |
| ME4000_AI_SIMULTANEOUS_ENABLE | 0x00020602 |
| ME4000_AI_SCAN_BLOCKING | 0x00020701 |
| ME4000_AI_SCAN_ASYNCHRONOUS | 0x00020702 |

*Table 12: Constant definitions*

| Constant | Value |
|---|---|
| ME4000_AI_GET_NEW_VALUES_BLOCKING | 0x00020801 |
| ME4000_AI_GET_NEW_VALUES_NON_BLOCKING | 0x00020802 |
| ME4000_AI_WAIT_IDLE | 0x00020901 |
| ME4000_AI_WAIT_NONE | 0x00020902 |
| ME4000_AI_STATUS_IDLE | 0x00020A01 |
| ME4000_AI_STATUS_BUSY | 0x00020A02 |

***Analog Output (ME-4660/4670/4680)***

| | |
|---|---|
| ME4000_AO_TRIGGER_SOFTWARE | 0x00030101 |
| ME4000_AO_TRIGGER_EXT_DIGITAL | 0x00030103 |
| ME4000_AO_TRIGGER_EXT_EDGE_RISING | 0x00030201 |
| ME4000_AO_TRIGGER_EXT_EDGE_FALLING | 0x00030202 |
| ME4000_AO_TRIGGER_EXT_EDGE_BOTH | 0x00030203 |
| ME4000_AO_WRAPAROUND_BLOCKING | 0x00030301 |
| ME4000_AO_WRAPAROUND_ASYNCHRONOUS | 0x00030302 |
| ME4000_AO_WRAPAROUND_INFINITE | 0x00 |
| ME4000_AO_APPEND_NEW_VALUES_BLOCKING | 0x00030401 |
| ME4000_AO_APPEND_NEW_VALUES_NON_BLOCKING | 0x00030402 |
| ME4000_AO_WAIT_IDLE | 0x00030501 |
| ME4000_AO_WAIT_NONE | 0x00030502 |
| ME4000_AO_STATUS_IDLE | 0x00030601 |
| ME4000_AO_STATUS_BUSY | 0x00030602 |
| ME4000_AO_STOP_MODE_LAST_VALUE | 0x00030701 |
| ME4000_AO_STOP_MODE_IMMEDIATE | 0x00030702 |
| ME4000_AO_SHAPE_RECTANGLE | 0x00030801 |
| ME4000_AO_SHAPE_TRIANGLE | 0x00030802 |
| ME4000_AO_SHAPE_SINUS | 0x00030803 |
| ME4000_AO_SHAPE_COSINUS | 0x00030804 |
| ME4000_AO_SHAPE_POS_RAMP | 0x00030805 |
| ME4000_AO_SHAPE_NEG_RAMP | 0x00030806 |
| ME4000_AO_TRIGGER_EXT_DISABLE | 0x00030901 |
| ME4000_AO_TRIGGER_EXT_ENABLE | 0x00030902 |

***Digital-I/O Standard***

| | |
|---|---|
| ME4000_DIO_PORT_A | 0 |
| ME4000_DIO_PORT_B | 1 |
| ME4000_DIO_PORT_C | 2 |
| ME4000_DIO_PORT_D | 3 |

*Table 12: Constant definitions*

| Constant | Value |
|---|---|
| ME4000_DIO_PORT_INPUT | 0x00040201 |
| ME4000_DIO_PORT_OUTPUT | 0x00040202 |

***Bit Pattern Output (ME-4680)***

| | |
|---|---|
| ME4000_DIOBP_PORT_A | 0 |
| ME4000_DIOBP_PORT_B | 1 |
| ME4000_DIOBP_PORT_C | 2 |
| ME4000_DIOBP_PORT_D | 3 |
| ME4000_DIOBP_OUTPUT_MODE_BYTE_LOW | 0x00060101 |
| ME4000_DIOBP_OUTPUT_MODE_BYTE_HIGH | 0x00060102 |
| ME4000_DIOBP_TRIGGER_SOFTWARE | 0x00060201 |
| ME4000_DIOBP_TRIGGER_EXT_DIGITAL | 0x00060202 |
| ME4000_DIOBP_TRIGGER_EXT_EDGE_RISING | 0x00060301 |
| ME4000_DIOBP_TRIGGER_EXT_EDGE_FALLING | 0x00060302 |
| ME4000_DIOBP_TRIGGER_EXT_EDGE_BOTH | 0x00060303 |
| ME4000_DIOBP_WRAPAROUND_BLOCKING | 0x00060401 |
| ME4000_DIOBP_WRAPAROUND_ASYNCHRONOUS | 0x00060402 |
| ME4000_DIOBP_WRAPAROUND_INFINITE | 0x00 |
| ME4000_DIOBP_APPEND_NEW_VALUES_BLOCKING | 0x00060501 |
| ME4000_DIOBP_APPEND_NEW_VALUES_NON_BLOCKING | 0x00060502 |
| ME4000_DIOBP_WAIT_IDLE | 0x00060601 |
| ME4000_DIOBP_WAIT_NONE | 0x00060602 |
| ME4000_DIOBP_STATUS_IDLE | 0x00060701 |
| ME4000_DIOBP_STATUS_BUSY | 0x00060702 |
| ME4000_DIOBP_STOP_MODE_LAST_VALUE | 0x00060801 |
| ME4000_DIOBP_STOP_MODE_IMMEDIATE | 0x00060802 |

***Counter (ME-4660/4670/4680)***

| | |
|---|---|
| ME4000_CNT_MODE_0 | 0x00050101 |
| ME4000_CNT_MODE_1 | 0x00050102 |
| ME4000_CNT_MODE_2 | 0x00050103 |
| ME4000_CNT_MODE_3 | 0x00050104 |
| ME4000_CNT_MODE_4 | 0x00050105 |
| ME4000_CNT_MODE_5 | 0x00050106 |

*Continuation next page*

*Table 12: Constant definitions*

| Constant | Value |
|---|---|
| ***ME-MultiSig Functions*** | |
| ME4000_MULTISIG_LED_OFF | 0x00070101 |
| ME4000_MULTISIG_LED_ON | 0x00070102 |
| ME4000_MULTISIG_GROUP_A | 0x00070201 |
| ME4000_MULTISIG_GROUP_B | 0x00070202 |
| ME4000_MULTISIG_GAIN_1 | 0x00070301 |
| ME4000_MULTISIG_GAIN_10 | 0x00070302 |
| ME4000_MULTISIG_GAIN_100 | 0x00070303 |
| ME4000_MULTISIG_MODULE_NONE | 0x00070401 |
| ME4000_MULTISIG_MODULE_DIFF16_10V | 0x00070402 |
| ME4000_MULTISIG_MODULE_DIFF16_20V | 0x00070403 |
| ME4000_MULTISIG_MODULE_DIFF16_50V | 0x00070404 |
| ME4000_MULTISIG_MODULE_CURRENT16_0_20MA | 0x00070405 |
| ME4000_MULTISIG_MODULE_RTD8_PT100 | 0x00070406 |
| ME4000_MULTISIG_MODULE_RTD8_PT500 | 0x00070407 |
| ME4000_MULTISIG_MODULE_RTD8_PT1000 | 0x00070408 |
| ME4000_MULTISIG_MODULE_TE8_TYPE_B | 0x00070409 |
| ME4000_MULTISIG_MODULE_TE8_TYPE_E | 0x0007040A |
| ME4000_MULTISIG_MODULE_TE8_TYPE_J | 0x0007040B |
| ME4000_MULTISIG_MODULE_TE8_TYPE_K | 0x0007040C |
| ME4000_MULTISIG_MODULE_TE8_TYPE_N | 0x0007040D |
| ME4000_MULTISIG_MODULE_TE8_TYPE_R | 0x0007040E |
| ME4000_MULTISIG_MODULE_TE8_TYPE_S | 0x0007040F |
| ME4000_MULTISIG_MODULE_TE8_TYPE_T | 0x00070410 |
| ME4000_MULTISIG_MODULE_TE8_TEMP_SENSOR | 0x00070411 |
| ME4000_MULTISIG_I_MEASURED_DEFAULT | 0.0005 |

*Table 12: Constant definitions*

# F Index

## Function Reference

**A**