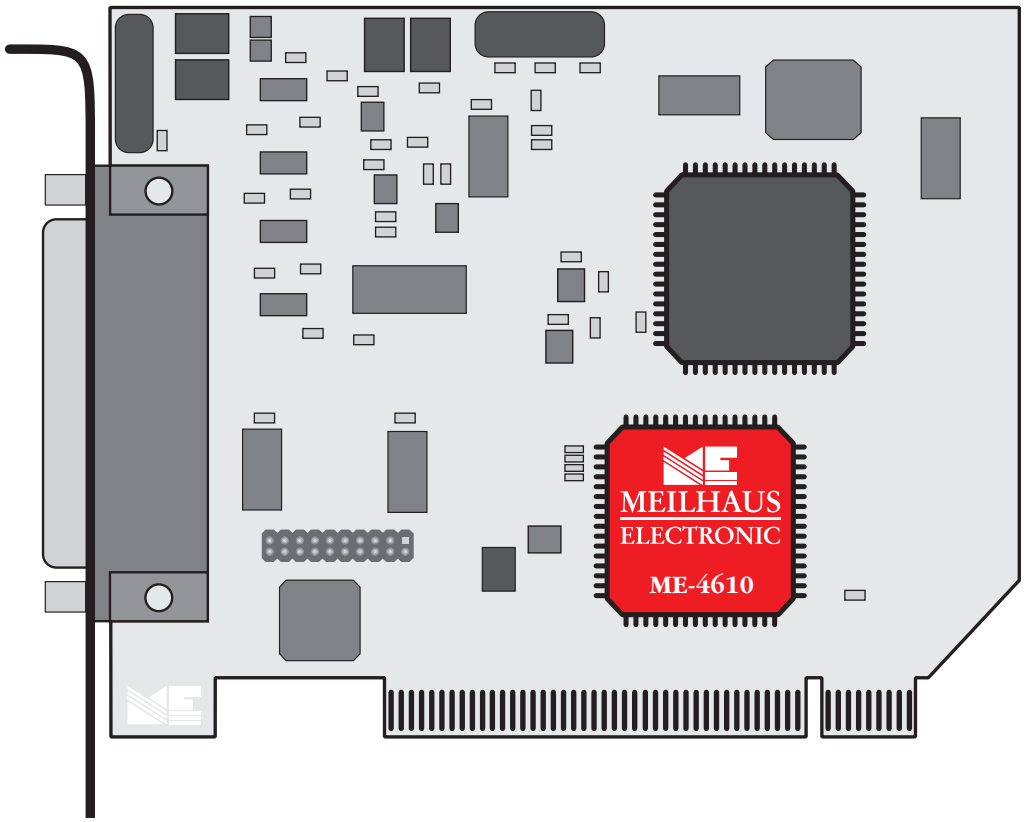
Meilhaus Electronic Manual

ME-4610 1.2E

Typ „ME-Jekyll®“



16 Bit Multifunction Board with 16 A/D Channels

Imprint

Manual ME-4610

Revision 1.2E
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.

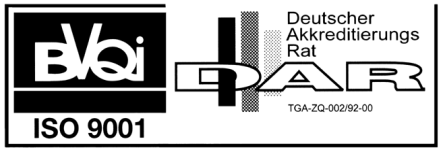


Table of Contents

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 type ME-4610 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

The **ME-4610 series** provide 16 single ended A/D channels. The input channels are routed through a high impedance input stage to a 16 bit 500 kHz A/D converter. The input voltage range is fixed to ±10 V.

The digital I/O section has **32 TTL-I/Os** which are organised as 4 bi-directional ports. The ports C and D are available on a flat

IDC 20 pin connector. They can be routed to a D-sub 25 connector on a separate mounting bracket.

Additional there are 3 free programmable **16 bit counters** (1 x 8254) available on the ME-4610.

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

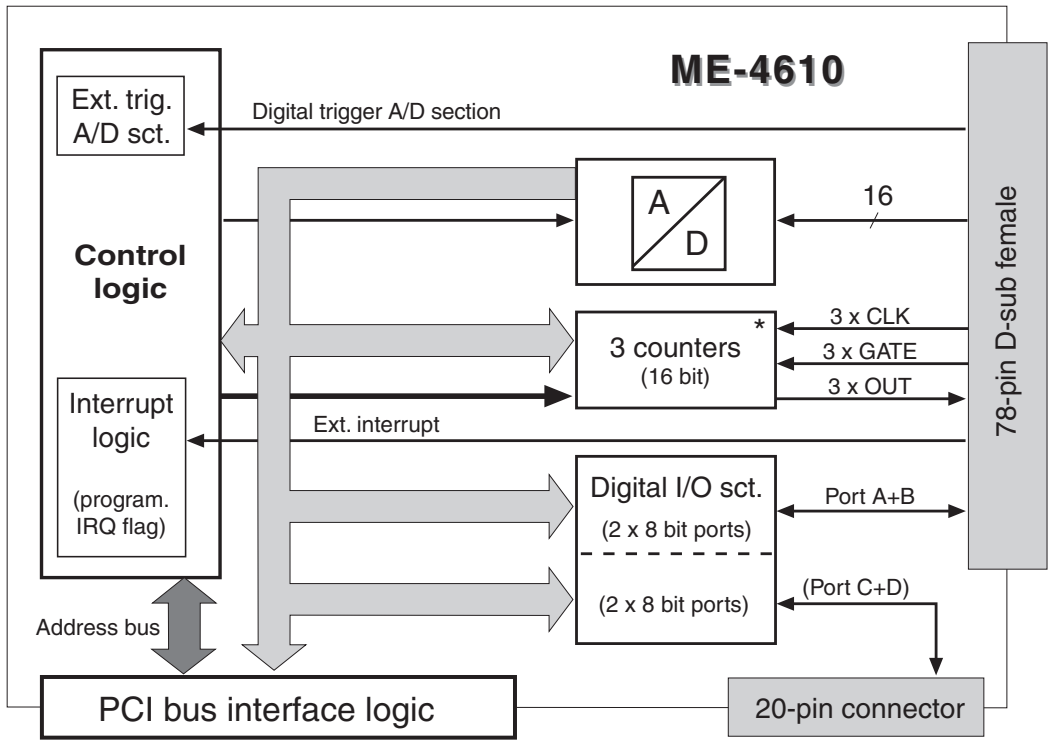


Diagram 1: Block diagram of ME-4610

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 input 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 (see pinout on page 118).

The following chapters describe the switching and connections of the different functional groups. The operating modes and programming is described in chapter 4 (page 19 and following).

3.3 A/D Section

The ME-4610 provides 16 single ended input channels. All channels are isolated with a high impedance input stage:

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

Each input channel (AD_*x*) requires a low resistance connection to PC ground. It is important that all minus (-) lines have the same potential to avoid cross currents and therefore measurement errors.

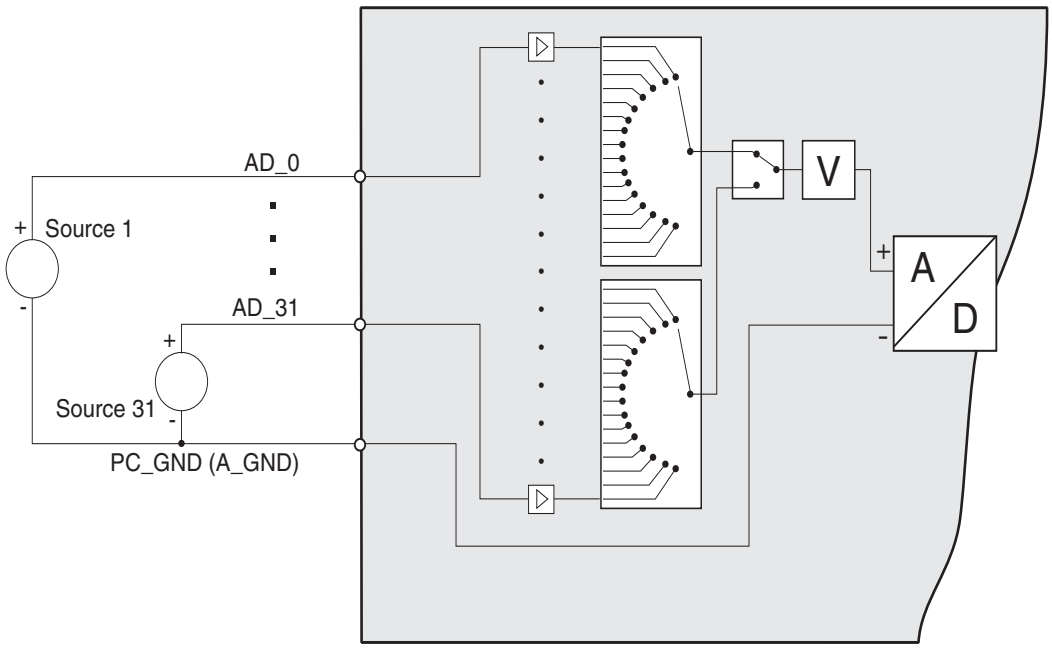


Diagram 2: Switching in single ended operation

The input voltage range is -10V…(+10V-1LSB). **The input voltage applied to the analog inputs must not exceed ±15V.**

The following (ideal) characteristic curves are valid:

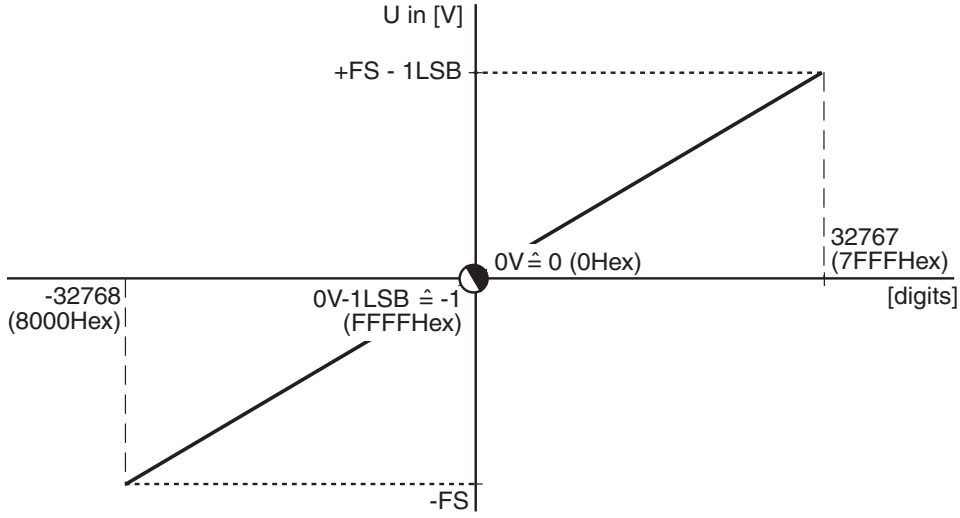


Diagram 3: Characteristic of bipolar input ranges

(„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 is only met approximately as a rule (see also specifications on page 115).

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 trigger options.

3.3.1 External Trigger A/D Section

The ME-4610 provides a digital trigger input for the A/D section (AD_TRIG_D). It is important that the TTL level of the external trigger input switching be within the limits (see specifications on page 115) and that a reference to PC ground (PC_GND) be made.

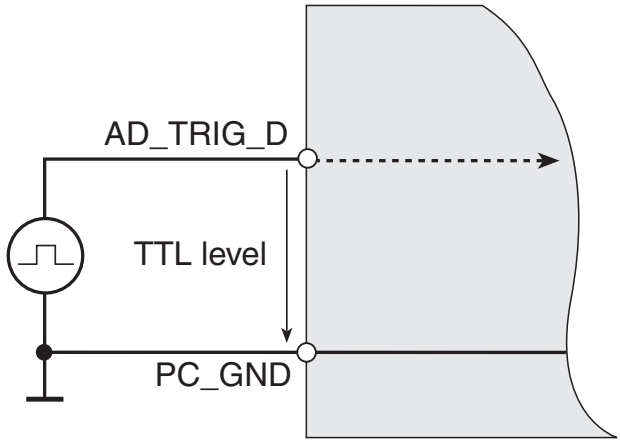


Diagram 4: Switching digital trigger

Depending on the selected option (RISING, FALLING or BOTH) the A/D conversion will be started on the matching edge.

Diagram 5: Trigger edges

3.4 Digital I/O Section

The ME-4610 provides four TTL ports with 8 bits each. Each port can be configured independently as input or output. After power up, all ports are set to input.

Port A and B are available at the 78pin D-Sub connector. 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.

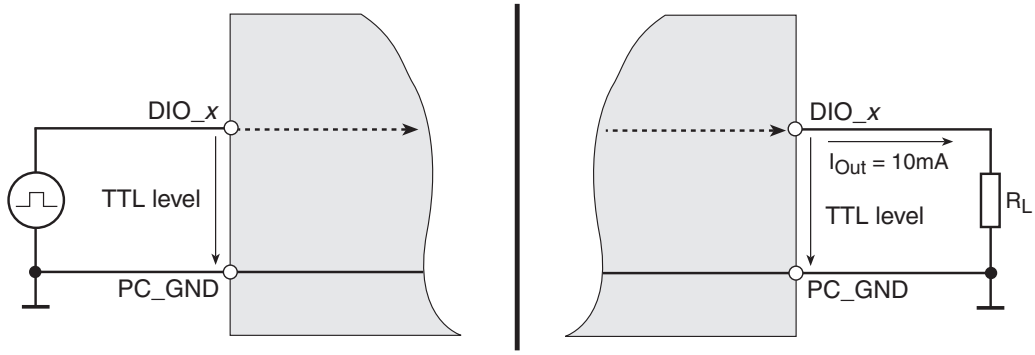


Diagram 6: Switching of the digital inputs

It is important that the TTL level of the inputs and outputs switching be within the limits (see specifications on page 115) and that a reference to PC ground (PC_GND) be made. The maximum output current is $I_{Out} = I_{OL} = I_{OH} = 10mA$.

For more information about programming refer to chapter 4.2 "Digital I/O Section".

3.5 Counter

3.5.1 Counter Chip

The **ME-4610** uses the standard counter chip of type 82C54. This universal component has 3 independent 16 bit down counters. All counter signals are available on the D-sub connector. After the GATE signal has been properly set (TTL: 5V) 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 work with TTL level (see Appendix A "Specifications") and require a reference to PC ground (PC_GND). The maximum output current for low level is I_{OL} = 7.8mA and for high level I_{OH} = 6mA.

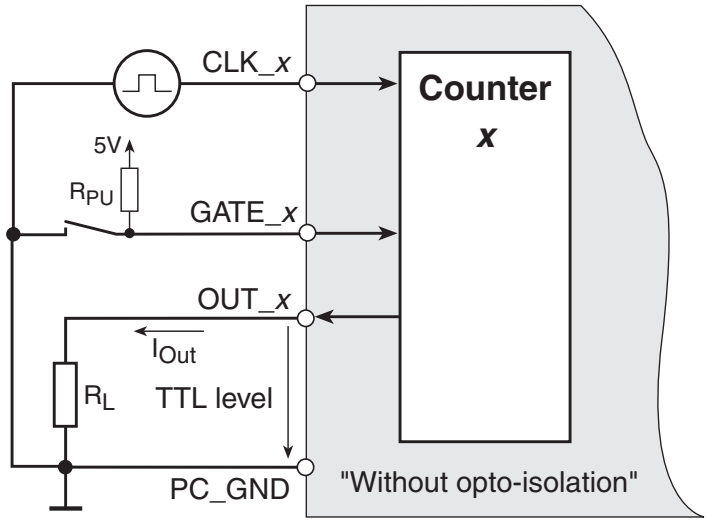


Diagram 7: Switching of counters

For more information about programming the counters see chapter 4.3.

3.5.2 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 42 and 94).

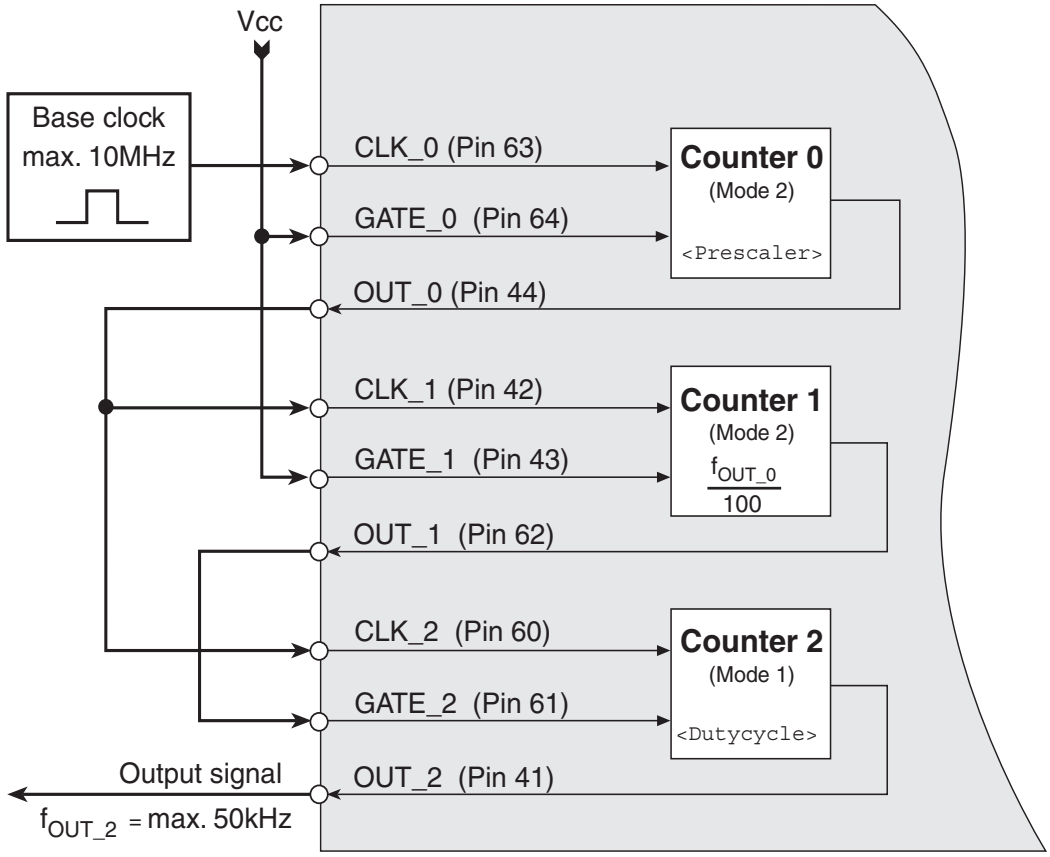


Diagram 8: Switching for pulse width modulation

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(i), which also provides a 10MHz crystal oscillator.

3.6 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*.

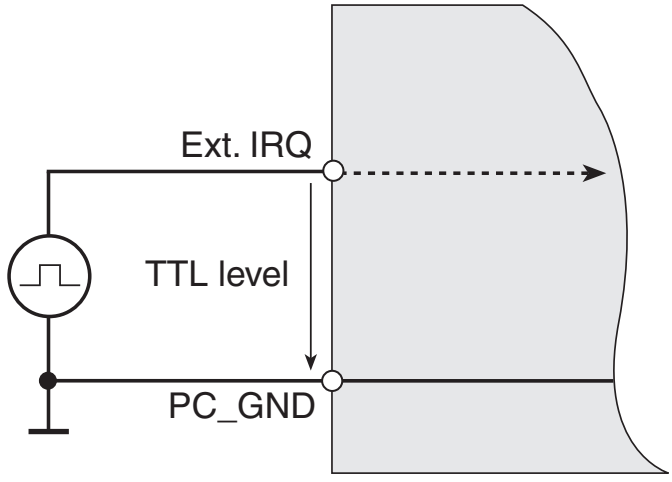


Diagram 9: Switching of external interrupt input

The voltage level at the interrupt input must be within the specified limits (see appendix A, specification) and a reference to PC ground (PC_GND) must be made.

4 Programming

4.1 A/D Section

Operation Mode	Usage	Trigger	Timing
AISingle (see page 19)	1 channel, 1 measurement value	Software-Start, ext. digital	–
AIContinuous (see page 21)	Acquisition of an unknown number of measurement values corresponding to the channel list	Software-Start, ext. digital	CHAN Timer, SCAN Timer (…AIConfig)
AIScan (see page 21)	Acquisition of a known number of measurement values corresponding to the channel list	Software-Start, ext. digital	CHAN Timer, SCAN Timer (…AIConfig)

Table 1: A/D operation modes

4.1.1 Operation Modes „AISingle“

ME-4610			
✔			

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

- Channel number 0…15.
- Input voltage range: always ±10V.
- Operation mode single ended.
- Trigger modes: per software or external digital trigger.
- 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:

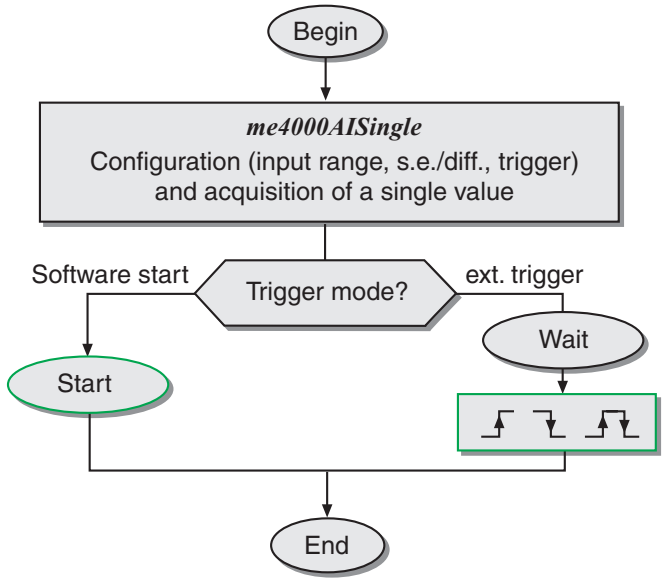


Diagram 10: Operation mode „AISingle“

For more information, refer to the sample programs in the ME-SDK and the function description on page 84.

4.1.2 Timer Controlled „AI Operation Modes“

ME-4610			
✔			

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 4.1.2.1)
2. Software preparation with the function …*AIContinuous* or …*AIScan* (see chapter 4.1.2.2)
3. Start the acquisition with …*AIStart* (see chapter 4.1.2.3)

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

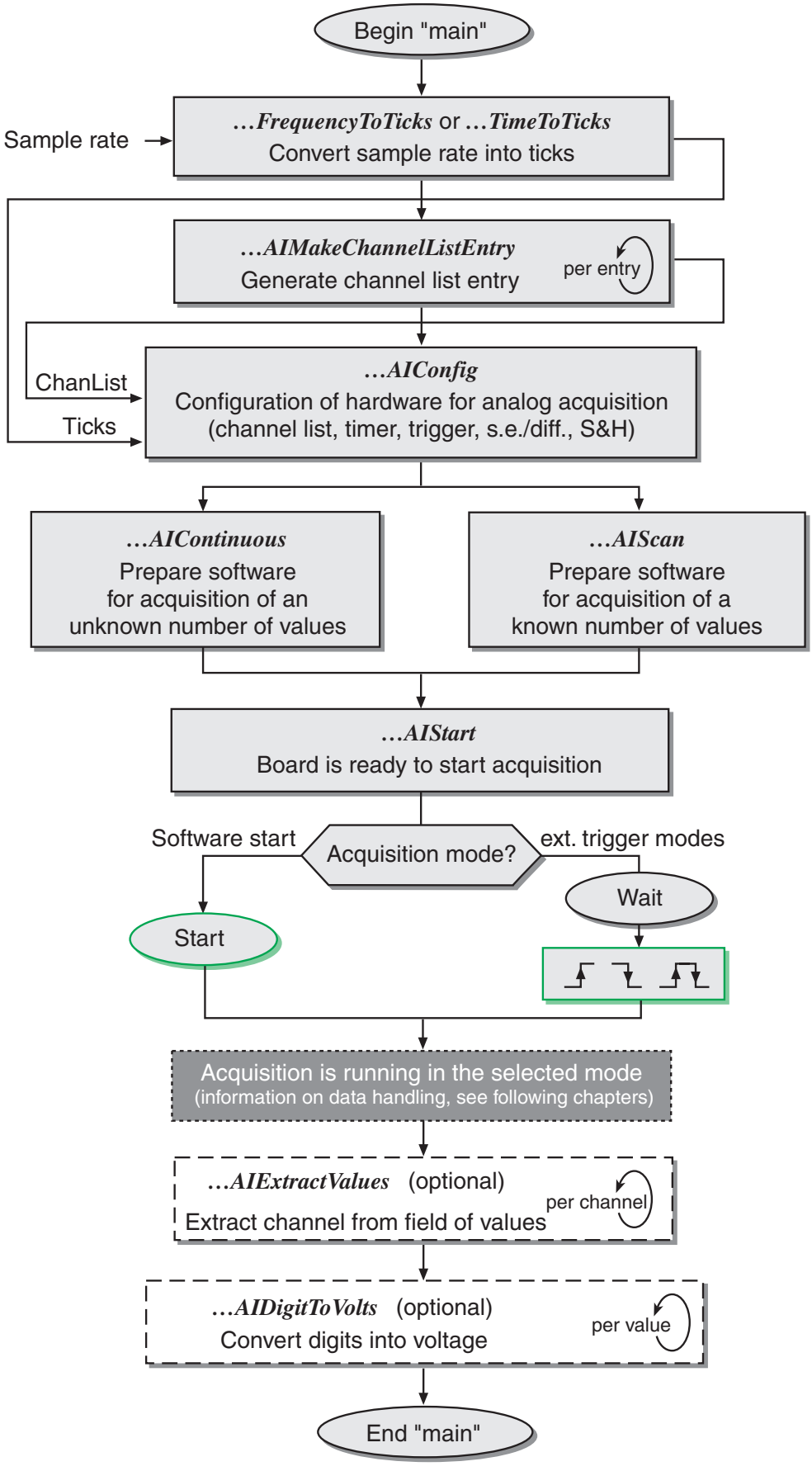


Diagram 11: Programming of AI section

4.1.2.1 Configuration of the A/D Section

Before the actual configuration is done, a user defined channel list to control the channel numbers must be created. 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 79):

- Channel number 0…15.
- Input voltage range: always ±10V.

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 68):

- Operation mode single ended.
- Acquisition modes
  (see also chapter 4.1.3 "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 30.30ns, 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).

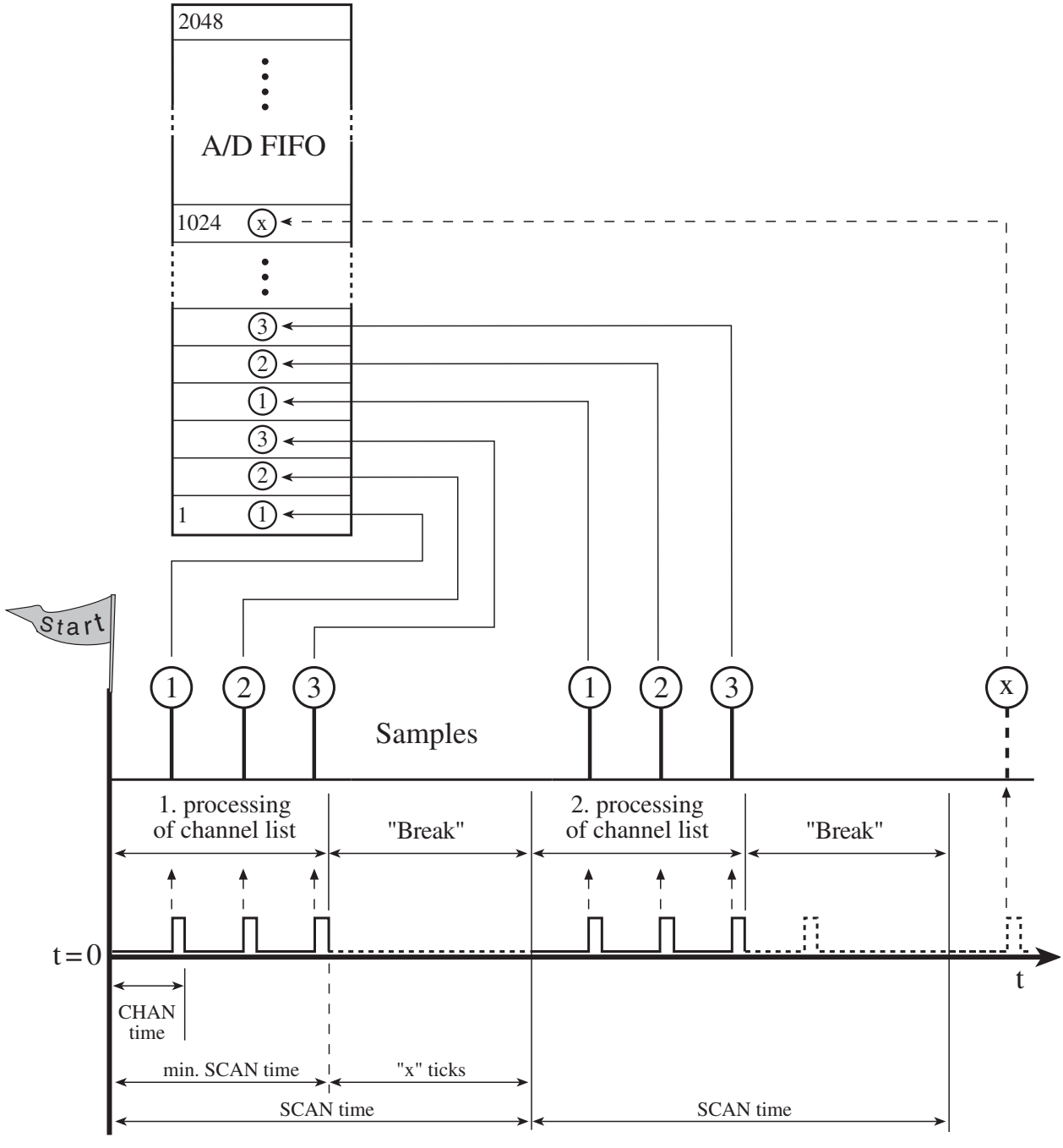


Diagram 12: A/D Timing

4.1.2.2 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 4.1.2.2.2) or a continuous stream of samples (see chapter 4.1.2.2.1) 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.

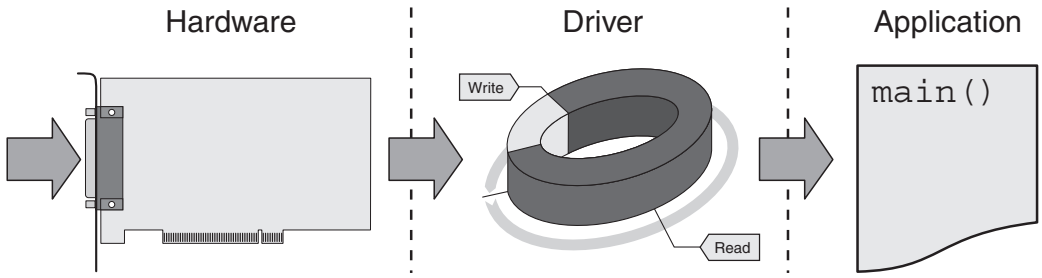


Diagram 13: Ring buffer AI operation

4.1.2.2.1 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 71.

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

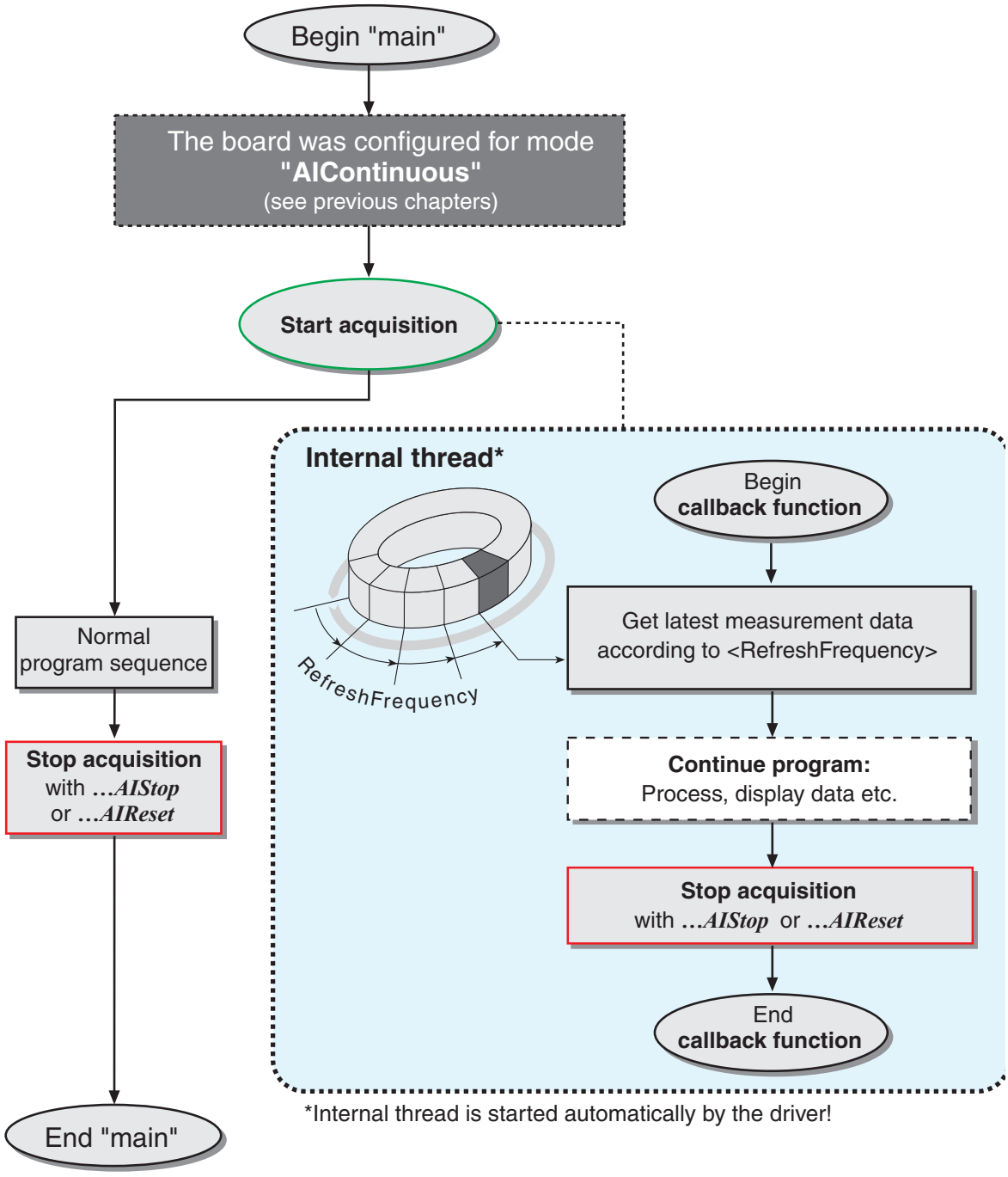


Diagram 14: Programming „AIContinuous“ with a callback function

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

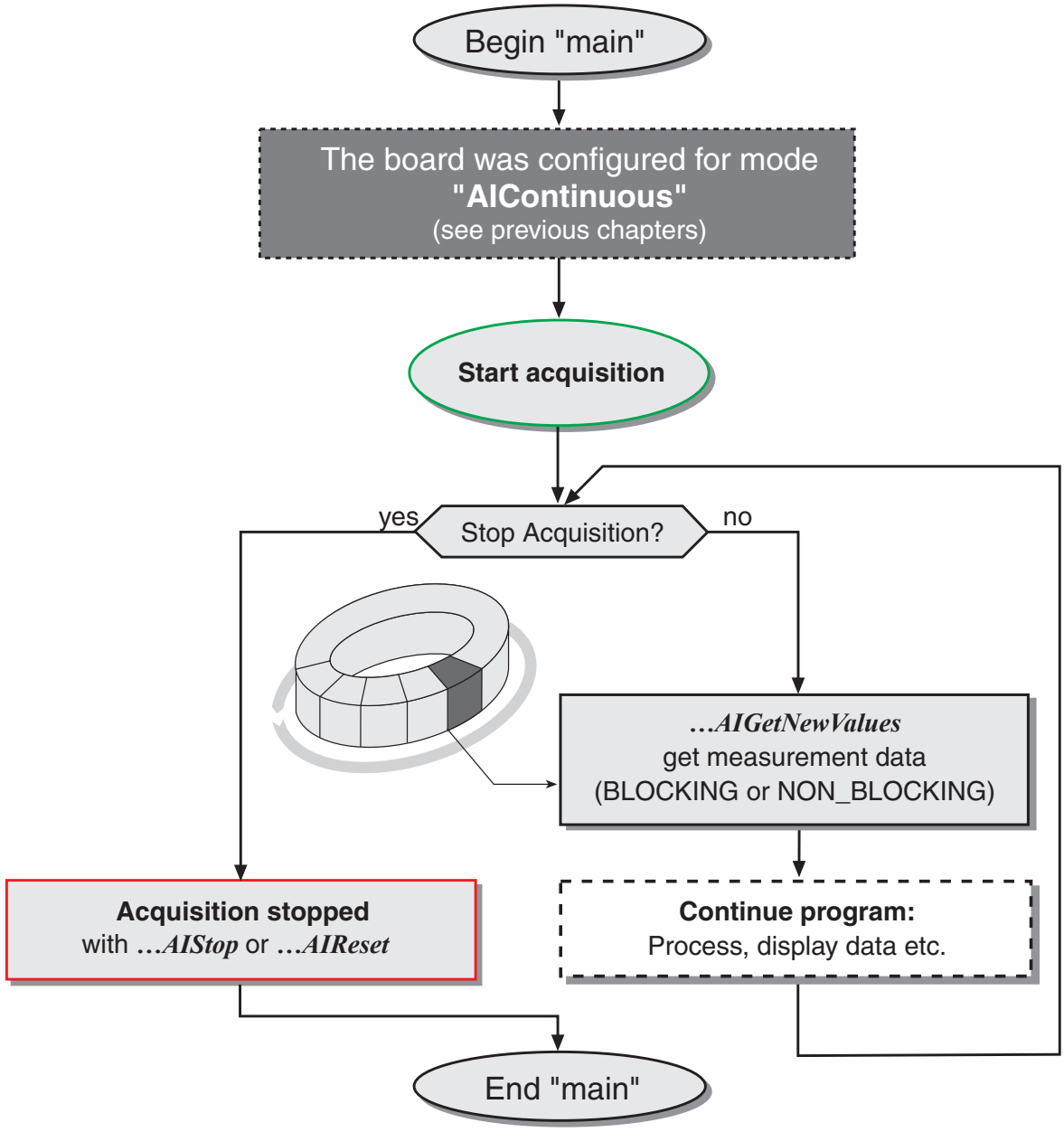


Diagram 15: Programming „AIContinuous“ with …AIGetNewValues

4.1.2.2.2 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 81.

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

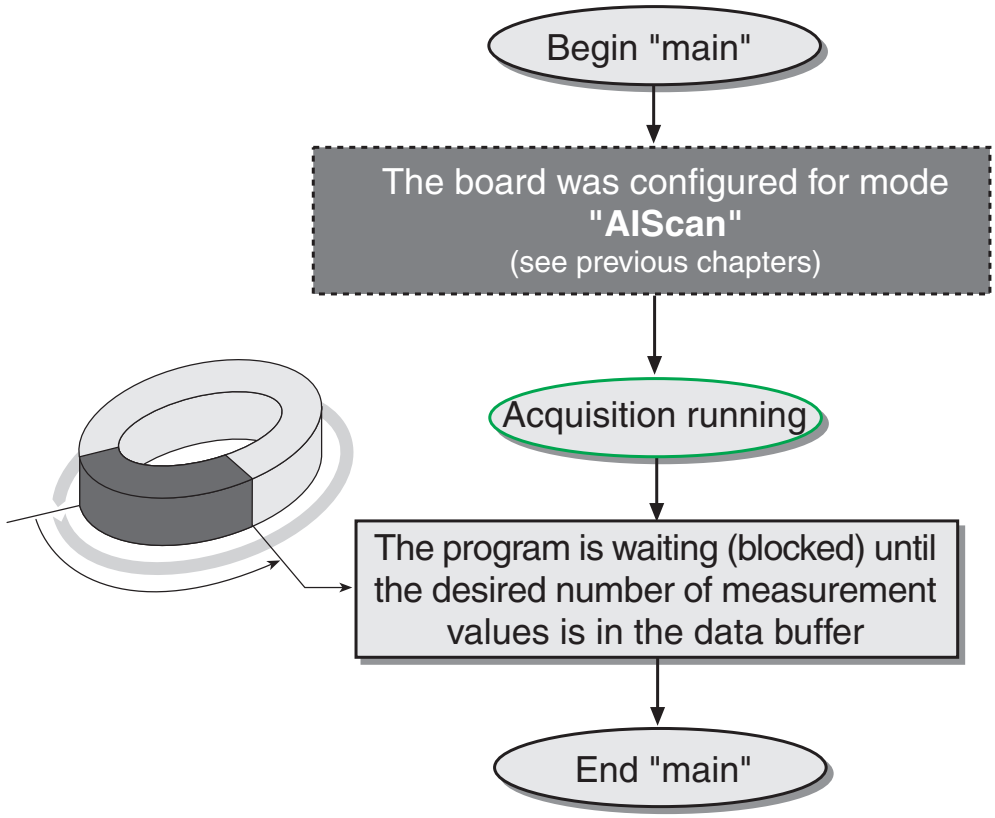


Diagram 16: Programming „AIScan“ in BLOCKING mode

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

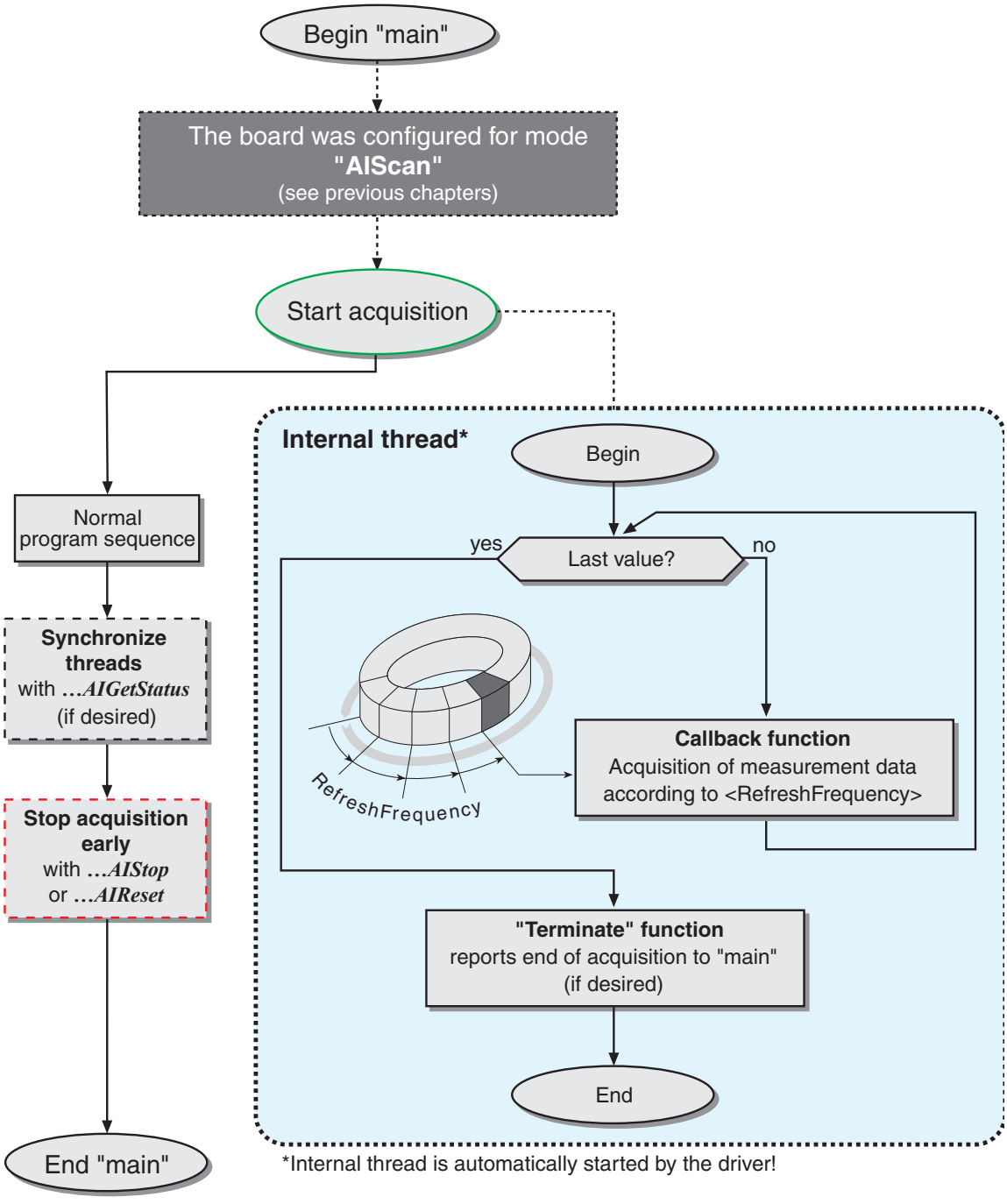


Diagram 17: Programming „AIScan“ with callback function

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

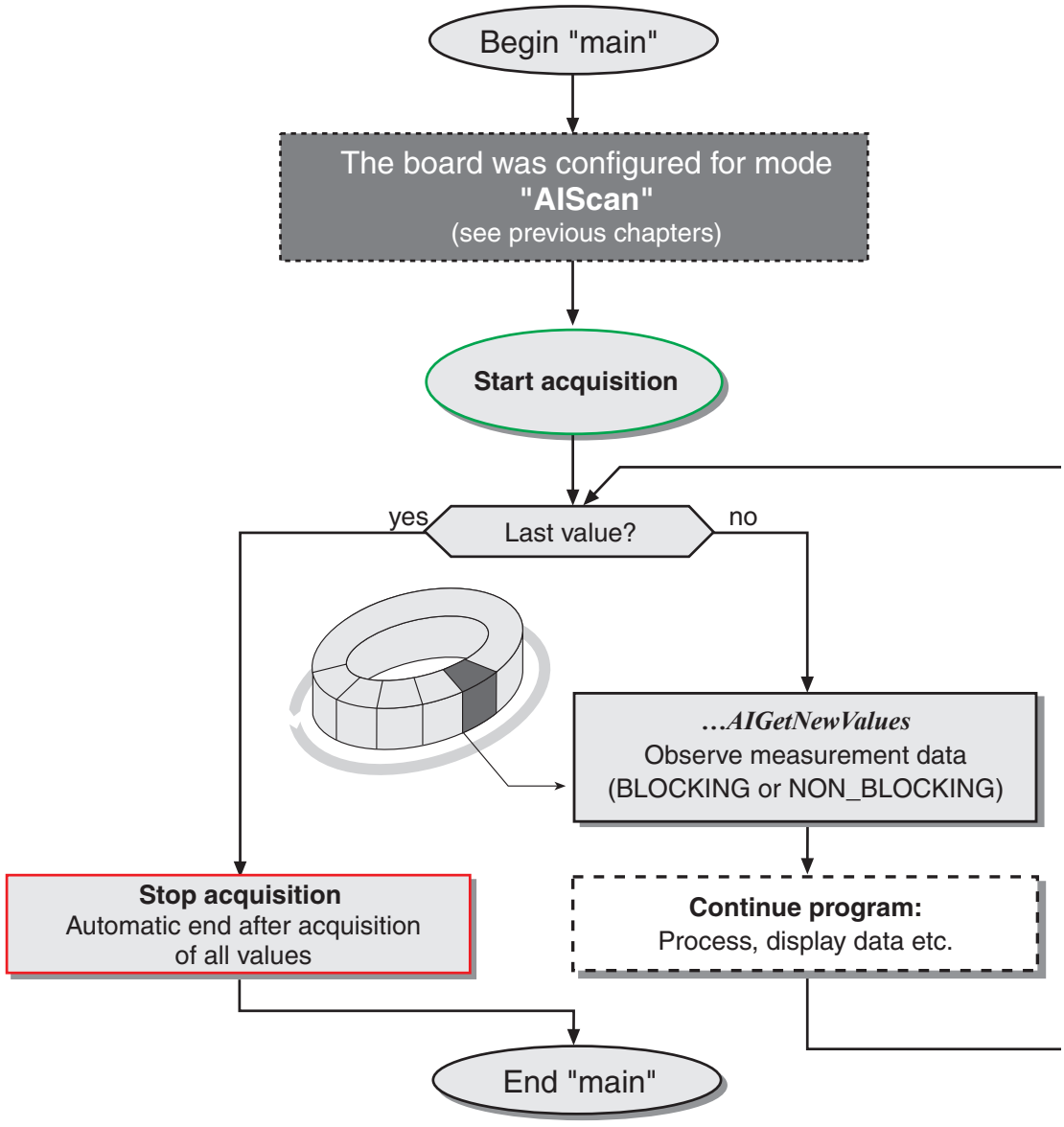


Diagram 18: Programming „AIScan“ with …AIGetNewValues

4.1.2.3 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 4.1.3.

4.1.2.4 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.

4.1.3 External Trigger A/D Section

ME-4610			
✔			

Depending on the type of application required, you can choose between different „acquisition modes“ with differing philosophies. The acquisition modes are independent of trigger mode 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*.

4.1.3.1 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>`).

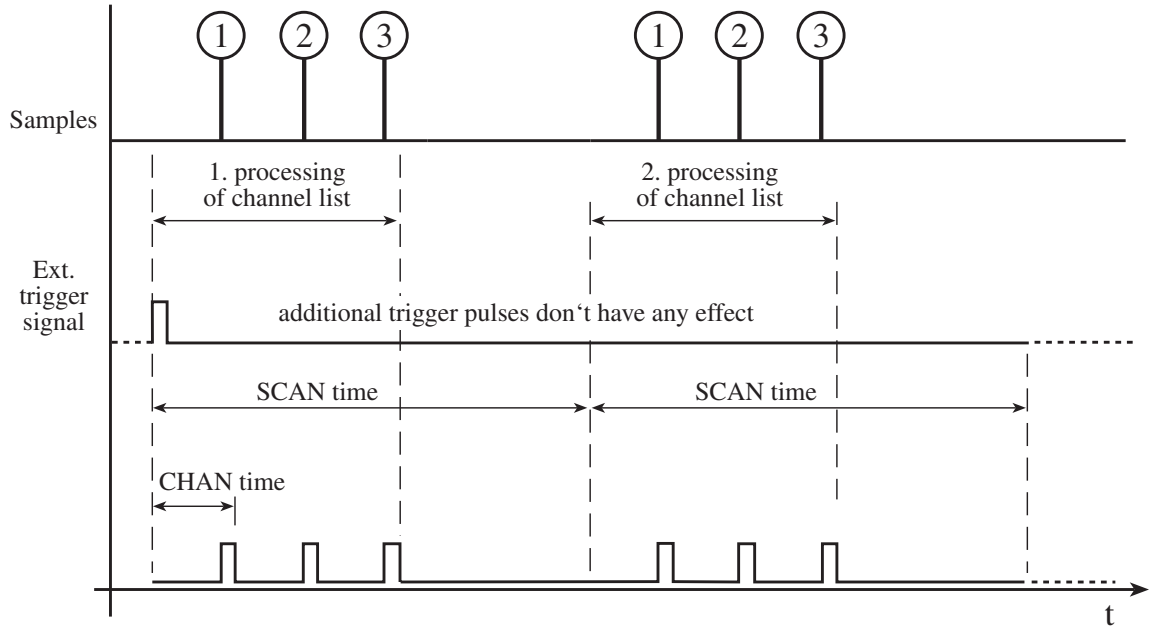


Diagram 19: Acquisition mode „External Standard“

4.1.3.2 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.

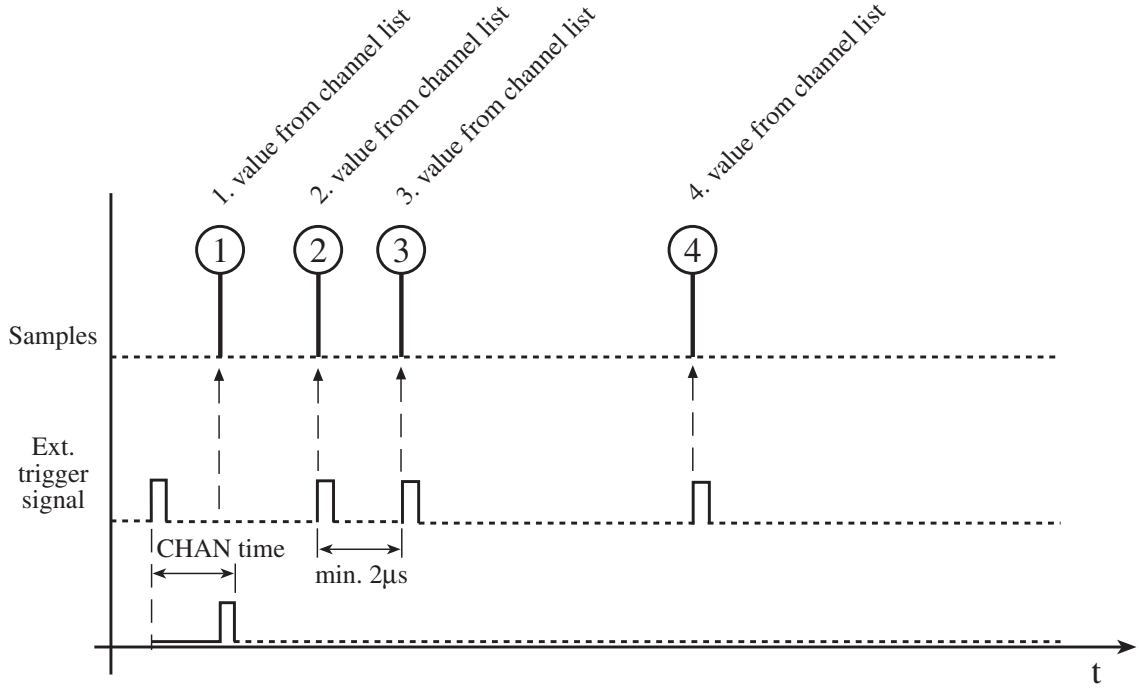


Diagram 20: Acquisition mode „External Single Value“

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.

4.1.3.3 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

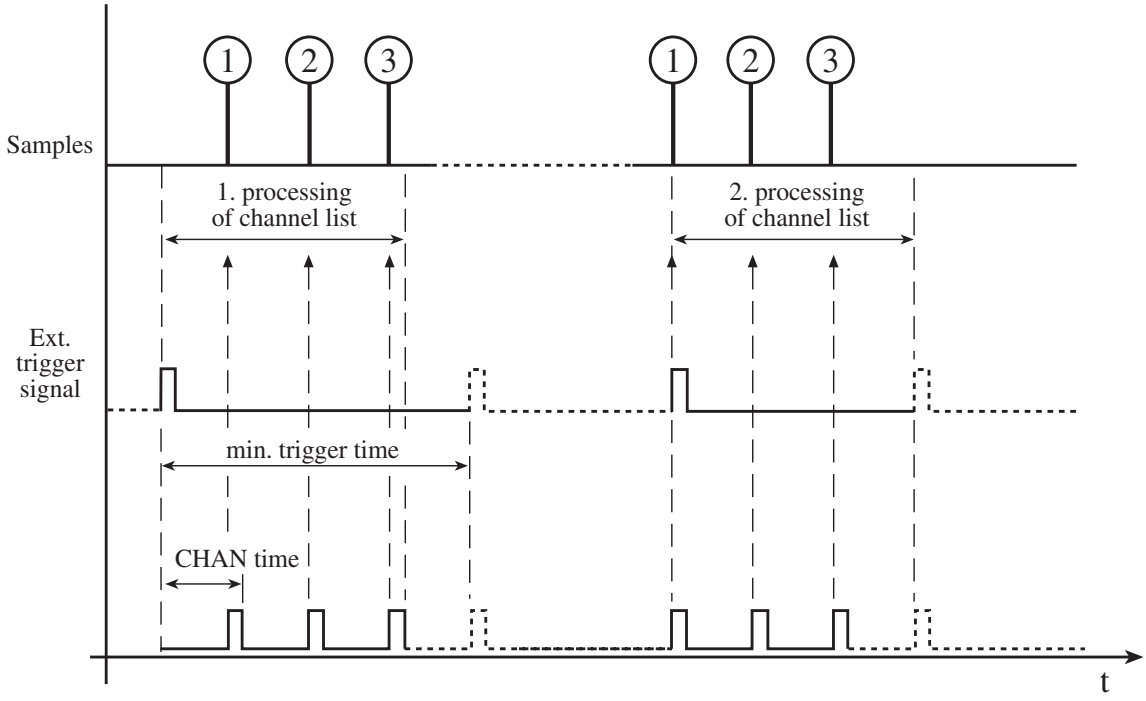


Diagram 21: Acquisition Mode „External Channel List“

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 Digital I/O Section

ME-4610			
✔			

The ME-4610 provides four digital I/O ports, each 8 bits wide (ports A, B, C and D). Each port can be independently configured as input or output. On power up, all ports are set to input. Use the function …*DIOConfig* to define the direction of the ports.

Note: Ports defined as output can still be read!

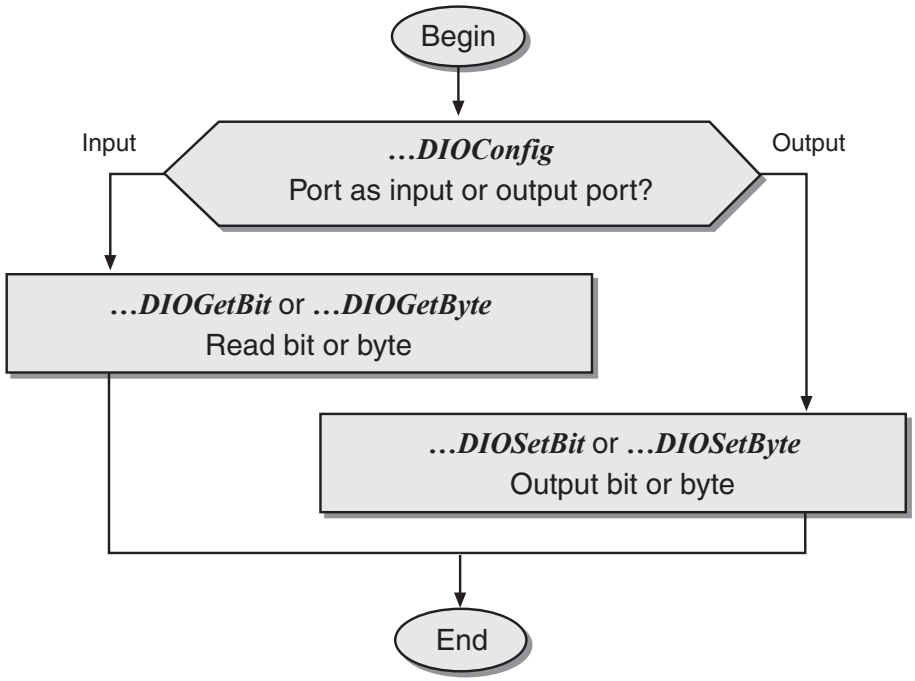


Diagram 22: Programming „DIO Standard“

For more information about the digital I/O section, refer to chapter 3.4 "Digital I/O Section" on page 15.

4.3 Counter

ME-4610			
✔			

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 5.3.5 „Counter Functions“ on page 94).

4.3.1 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 (0V) 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 (+5V). As soon as the start value is loaded and the counter is enabled, the counter begins counting downwards and the output remains low (0V).

Upon zero axis crossing, the output is set to high (+5V) 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.

4.3.2 Mode 1: Retriggerable „One-Shot“

The counter output (OUT_0…2) is set to high (+5V) 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 (positive edge). Upon zero axis crossing, the counter output is set to high (+5V) again.

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

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

4.3.3 Mode 2: Asymmetric Divider

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

4.3.4 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 (+5V) after initialisation. When the GATE input is enabled (+5V), 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 (+5V), else the output is set to high (+5V) 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.

4.3.5 Mode 4: Counter Start by Software Trigger

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

Upon zero axis crossing the output becomes low (0V) for one clock cycle. Afterwards the output becomes high (+5V) 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.

4.3.6 Mode 5: Counter Start by Hardware Trigger

The counter output (OUT_0…2) is set to high (+5V) 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 (positive edge). Upon zero axis crossing, the output becomes low (0V) for one clock cycle. Afterwards the output becomes high (+5V) 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) edge to the GATE input. The output will remains high (+5V) until zero axis crossing is meet.

4.3.7 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 3.5.2 on page 17. 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.

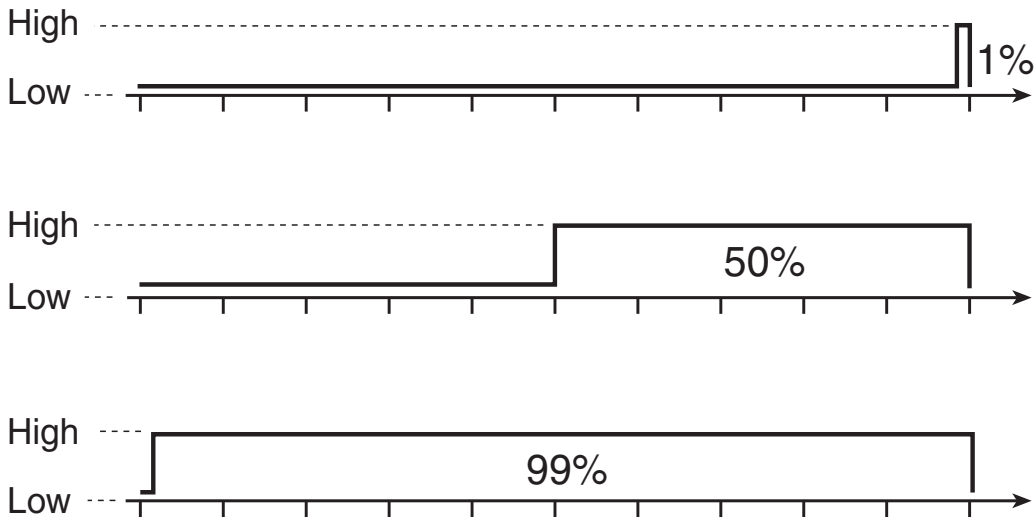


Diagram 23: Duty cycle PWM signal

4.4 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!

4.4.1 „Mux“ Operation

Operation Mode	Usage	Trigger	Timing
MultiSig-AISingle (see page 112)	1 channel (0…255), 1 measurement value	Software start, ext. digital	–

Table 2: „Mux“ Operation

In order to utilize the complete functional capability in „Mux“ operation, the digital ports A and B of the ME-4610 are required.

Make sure that the master board (ME-MUX32-M) is 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 function …*MultiSigAISingle*.

4.4.1.1 Configuration of the Base Boards

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

The following diagram demonstrates the program flow required:

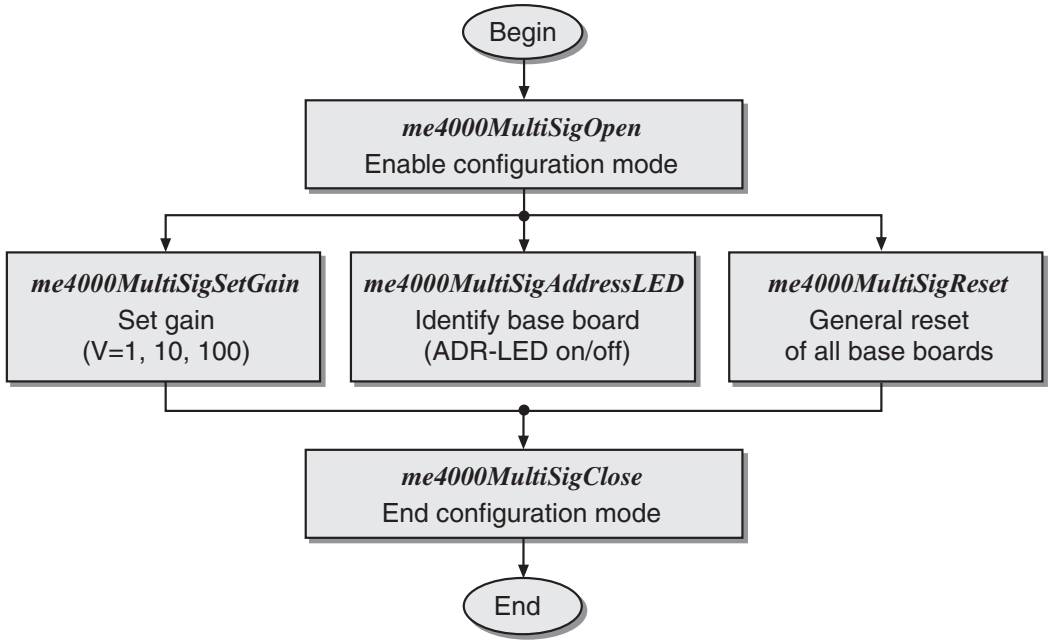


Diagram 24: Basic configuration „Mux“ operation

4.4.1.1.1 Setting the Amplification

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

4.4.1.1.2 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.

4.4.1.1.3 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

4.4.1.2 Operation Mode „MultiSig-AISingle“

ME-4610			
✔			

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…15.
- Gain factor of the channel group (V=1, 10, 100).
- Trigger modes: per software or external digital trigger.
- External trigger: falling, rising or both.
- Time out: in case the external trigger is not detected.

The following diagram demonstrates the program flow:

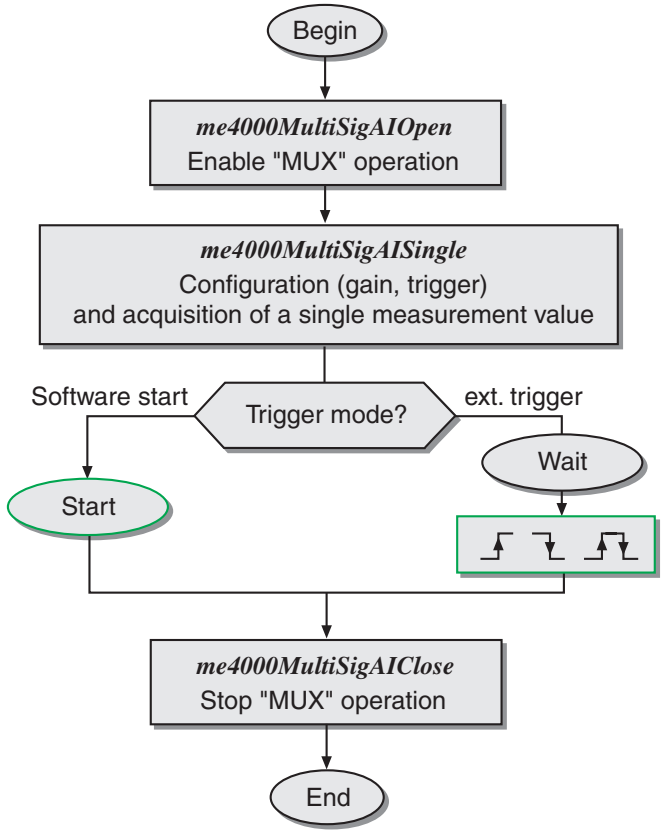


Diagram 25: Programming „MultiSig-AISingle“

4.5 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.

4.5.1 Visual C++

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

Table 3: 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.

4.5.2 Visual Basic

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

Table 4: 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.

4.5.3 Delphi

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

Table 5: Delphi

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

4.5.4 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 6: 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 function *me4000VEE_AIScan* automatically calls the appropriate start function *…VEE_AIStart* and returns after ending the measurement.

4.5.5 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 7: 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 function *me4000LV50_AIScan* must be used, which calls the appropriate start function *me4000LV_AIStart* automatically. The „scan“ function returns after ending the measurement.

4.5.6 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.

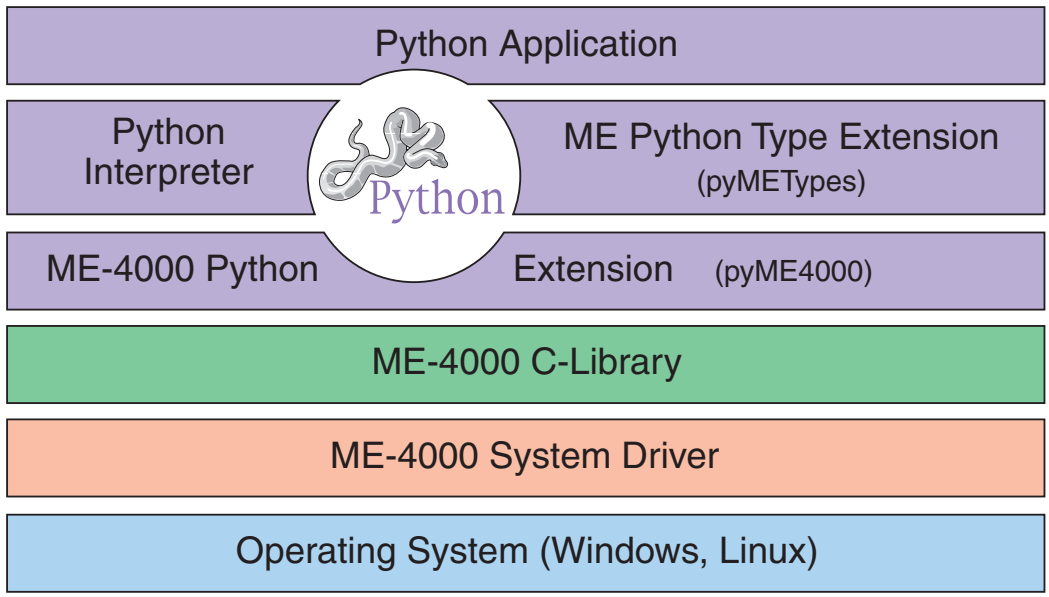


Diagram 26: Python Programming

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. 4.5 on page 46).

- **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 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	57
…ErrorGetLastMessage	Assign error string to the last error occured	58
…ErrorSetDefaultProc	Install predefined global error routine for API	59
…ErrorSetUserProc	Install user defined global error routine for API	60
General Functions		
…FrequencyToTicks	Convert frequency to ticks	61
…GetBoardVersion	Determine board version	63
…GetDLLVersion	Determine DLL version number	64
…GetDriverVersion	Determine driver version number	64
…GetSerialNumber	Determine serial number of the board	65
…TimeToTicks	Convert period to ticks	66

Table 8: Overview library functions

Function	Short Description	Page
Analog Input		
…AIConfig	Configuration of A/D section	68
…AIContinuous	Acquisition of a unknown number of measurement values	71
…AIDigitToVolt	Conversion of the digit value to a voltage value	73
…AIExtractValues	Extract values for one channel from data buffer	74
…AIGetNewValues	Asynchronous reading of data	76
…AIGetStatus	State request for „AIScan“	77
…AIMakeChannelListEntry	Create a channel list entry	79
…AIReset	Reset of a timer controlled acquisition	80
…AIScan	Acquisition of a known number of measurement values	81
…AISingle	Single value measurement	84
…AIStart	Starting a timer controlled acquisition	86
…AIStop	Stopping a timer controlled acquisition	87
Digital-I/O		
…DIOConfig	Configuring the digital ports for standard digital I/O operation	88
…DIOGetBit	Getting one bit	89
…DIOGetByte	Getting a byte	90
…DIOResetAll	Reset the digital I/O section	91
…DIOSetBit	Setting one bit	92
…DIOSetByte	Setting a byte	93
Counter Functions		
…CntPWMStart	Starting PWM output	94
…CntPWMStop	Ending the PWM output	95
…CntRead	Reading the counter state	96
…CntWrite	Configuring and starting a counter	97
Functions for external Interrupt		
…ExtIrqDisable	Disable the external IRQ input	98
…ExtIrqEnable	Enable the external IRQ input	99
…ExtIrqGetCount	Determine the number of ext. IRQs	100

Table 8: Overview library functions

Function	Short Description	Page
MultiSig Functions		
…MultiSigAddressLED	Controlling the Address LED	102
…MultiSigClose	Close the configuration mode	103
…MultiSigOpen	Open the configuration mode	104
…MultiSigReset	Reset all master and slave boards	105
…MultiSigSetGain	Gain factor per channel group	106
…MultiSigAIClose	Closing „Mux“ operation mode	107
…MultiSigAIDigitToSize	Conversion of the digit value to the appropriate physical size	108
…MultiSigAIOpen	Opening „Mux“ operation mode	111
…MultiSigAISingle	Single value measurement	112

Table 8: Overview library functions

5.3.1 Error Handling

me4000ErrorGetMessage

✎ Description

ME-4610			
✔			

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-4610			
✔			

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-4610			
✔			

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-4610			
✔			

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-4610			
✔			

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-4610			
✔			

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:

4610Hex: ME-4610 16 A/D, s. e., 32 DIO, counter

❮ 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-4610			
✔			

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-4610			
✔			

Determines the version number of the ME-4600 driver.

● Definitions

VC: me4000GetDriverVersion(unsigned long* pulVersion);
LV: me4000LV_... (siehe `me4000LV.h`)
VB: me4000VB_... (siehe `me4000.bas`)
VEE: me4000VEE_... (siehe `me4000VEE.h`)

➔ 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-4610			
✔			

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-4610			
✔			

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-4610			
✔			

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>` and establishes the external trigger mode and edge.

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 61) 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 …*AIMakeChannelListEntry*)

<ChanListCount>

Number of channel list entries.

<SDMode>

Please pass the constant for single ended measurement:

- ME4000_AI_INPUT_SINGLE_ENDED

<Simultaneous>

On the ME-4610 the sample & hold option is not available. Please pass the constant:

- ME4000_AI_SIMULTANEOUS_DISABLE

<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 12):
(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 12).
- 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 19).
- 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 20).
- 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 21). 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_VALUE_NOT_USED
  No external trigger in use.

<ExtTriggerEdge>

Selects the trigger edge for the digital trigger input.
See also 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-4610			
✔			

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 4.1.2 "Timer Controlled „AI Operation Modes“" on page 21. 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-4610			
✔			

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:

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

● 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 ME-4610 works in the input voltage range ±10V. Please use the constant:

- ME4000_AI_RANGE_BIPOLAR_10

<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-4610			
✔			

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…15

<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 of 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-4610			
✔			

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 26 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-4610			
✔			

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-4610			
✔			

This function generates a channel list entry based on the parameters `<ChannelNumber>` and `<Range>` and writes it into an array which is passed to the function …*AIConfig* later. 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: 0…15

`<Range>`

The ME-4610 works in the voltage input range ±10V. Please pass the constant:

- ME4000_AI_RANGE_BIPOLAR_10

`<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-4610			
✔			

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-4610			
✔			

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 4.1.2 "Timer Controlled „AI Operation Modes“" on page 21 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-4610			
✔			

This function is used to acquire a single value. The conversion can be started 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: 0…15

<Range>

The ME-4610 works in the input voltage range ±10V. Please pass the constant:

- ME4000_AI_RANGE_BIPOLAR_10

<SDMode>

Please pass the constant for single ended measurement:

- ME4000_AI_INPUT_SINGLE_ENDED

<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.

<ExtTriggerEdge>

Selects the trigger edge for the external A/D trigger input.

- 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-4610			
✔			

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 3.3.1 on page 14).

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-4610			
✔			

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 Digital Input/Output

me4000DIOConfig

✎ Description

ME-4610			
✔			

This function is used to set the direction of the ports for digital input/output operation. The direction can be configured separately for each of the 8 bit wide ports.

This function must be called separately for each port. A port set for output can also be read back.

● 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 digital input/output:

- ME4000_DIO_PORT_INPUT:
  Input port
- ME4000_DIO_PORT_OUTPUT:
  Output port

‹ 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-4610			
✔			

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-4610			
✔			

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-4610			
✔			

This function sets all ports to input.

Calling this function does not effect 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-4610			
✔			

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-4610			
✔			

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.

5.3.5 Counter Functions

me4000CntPWMStart

✎ Description

ME-4610			
✔			

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 cannot 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 8 on page 17). The maximum frequency of the output signal is 50kHz and is calculated as follows:

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

The duty cycle can be set between 1…99% in increments of 1% (see diagram 23 on page 42).

☞ Note

Using this function is only useful in combination with external switching shown in diagram 8 on page 17.

● 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 outputsignal 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-4610			
✔			

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-4610			
✔			

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-4610			
✔			

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 4.3 on page 39).

- 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.

5.3.6 External Interrupt

me4000ExtIrqDisable

✎ Description

ME-4610			
✔			

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-4610			
✔			

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-4610			
✔			

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.

5.3.7 MultiSig Functions

me4000MultiSigAddressLED

✎ Description

ME-4610			
✔			

Identification of the specified base board by setting the respective address LED (e. g. for maintenance purposes).

● 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-4610			
✔			

This function closes a configuration mode that was opened with the function …*MultiSigOpen*. Hardware resources that were reserved are released again. See also chapter 4.4 „ME-MultiSig Control“ on page 43.

● 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-4610			
✔			

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*)

The following hardware resources will be reserved

- Digital port A and B.

See also chapter 4.4 „ME-MultiSig Control“ on page 43.

● 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-4610			
✔			

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-4610			
✔			

This function is used to set the gain for the specified channel group. Standard setting is to have the gain V=1.

● 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.

5.3.7.1 „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 43 and the “ME-MultiSig” system manual).

me4000MultiSigAIClose

✎ Description

ME-4610			
✔			

This function closes a functional operation that was opened with the function …*MultiSigAIOpen*. Hardware resources that were reserved are released again. See also chapter 4.4.1 „„Mux“ Operation“ on page 43.

● 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.

me4000MultiSigAIDigitToSize

✎ Description

ME-4610			
✔			

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 are taken into account.

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.

me4000MultiSigAIOpen

✎ Description

ME-4610			
✔			

This function prepares the „Mux“ operation. The required hardware resources are reserved:

- Digital port A and B.
- The A/D section is disabled for analog input using the „normal AI-functions“ (*me4000AI…*).

See also chapter 4.4 „ME-MultiSig Control“ on page 43 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.

me4000MultiSigAISingle

✎ Description

ME-4610			
✔			

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. 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…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.

<ExtTriggerEdge>

Select the trigger edge for the A/D-section. See also parameter `<TriggerMode>`.

- 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.

<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.

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

Number of A/D-channels	16 single ended
A/D converter	500kHz 16 bit A/D converter
Input ranges	bipolar: -10V…+10V - 1LSB (305µV)
Full scale error	bipolar: -FS+10LSB, +FS-10LSB
Inputs protected up to	±15 V
Input impedance	R_{IN} = typ. 600 MΩ; C_{IN} = typ. 3 pF
Total sampling rate	500 kS/s
Total accuracy	typ. ±4 LSB, max. ±10 LSB fullscale in input range ±10 V
A/D FIFO	2 k values FIFO
Channel list	max. 1024 entries (channel number)
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“
Acquisition modes	„Software Start“, „External Standard“, „External Single Value“, „External Channel List“
External trigger modes	ext. digital trigger
External trigger edges	rising, falling, both

External A/D Trigger

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

Digital-I/Os

Ports		4 x 8 bit
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

Counter

Number		3 x 16 bit (1 x 82C54)
Clock source		external up to 10 MHz
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)

External Interrupt

Ext. interrupt input	directly sent to the system (if enabled by the driver)
Reference to ground	PC ground (PC_GND)
Input level	see digital I/Os

General Information

DC/DC converter A/D section	±5V and ±15 V (2 x 1W)
Power consumption (without external load):	
„Without opto-isolation“	typ. 2.8A
Load for VCC_OUT	max. 200mA
Physical size	136mm 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
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	PC ground
VCC_OUT	V_{CC} output (+5V from PC) max. 200mA load
n.c.	Pin not connected

B1 78pin D-Sub Connector ME-4600 (ST1)

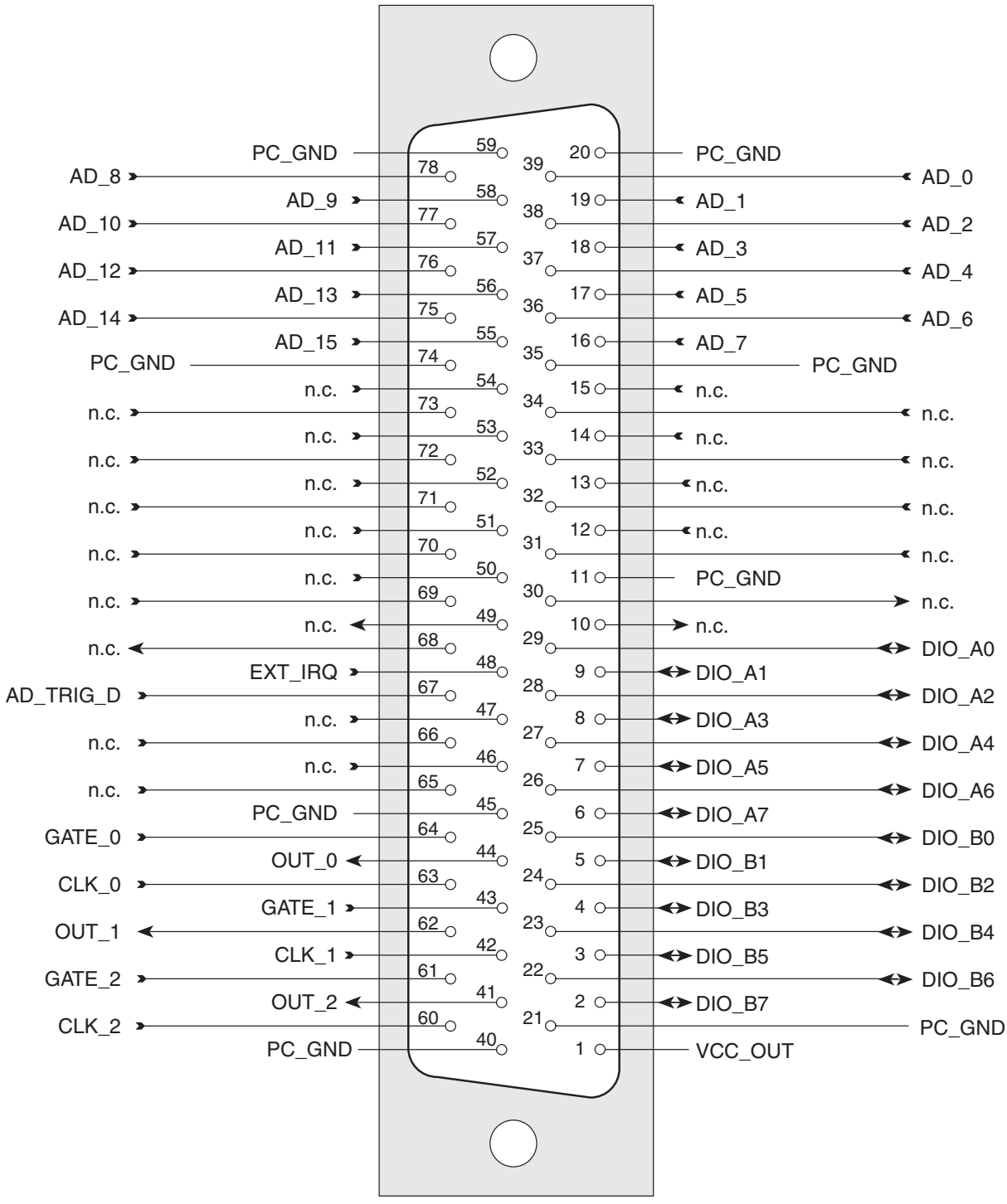


Diagram 27: Pinout of the 78pin D-Sub female connector (ST1)

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).

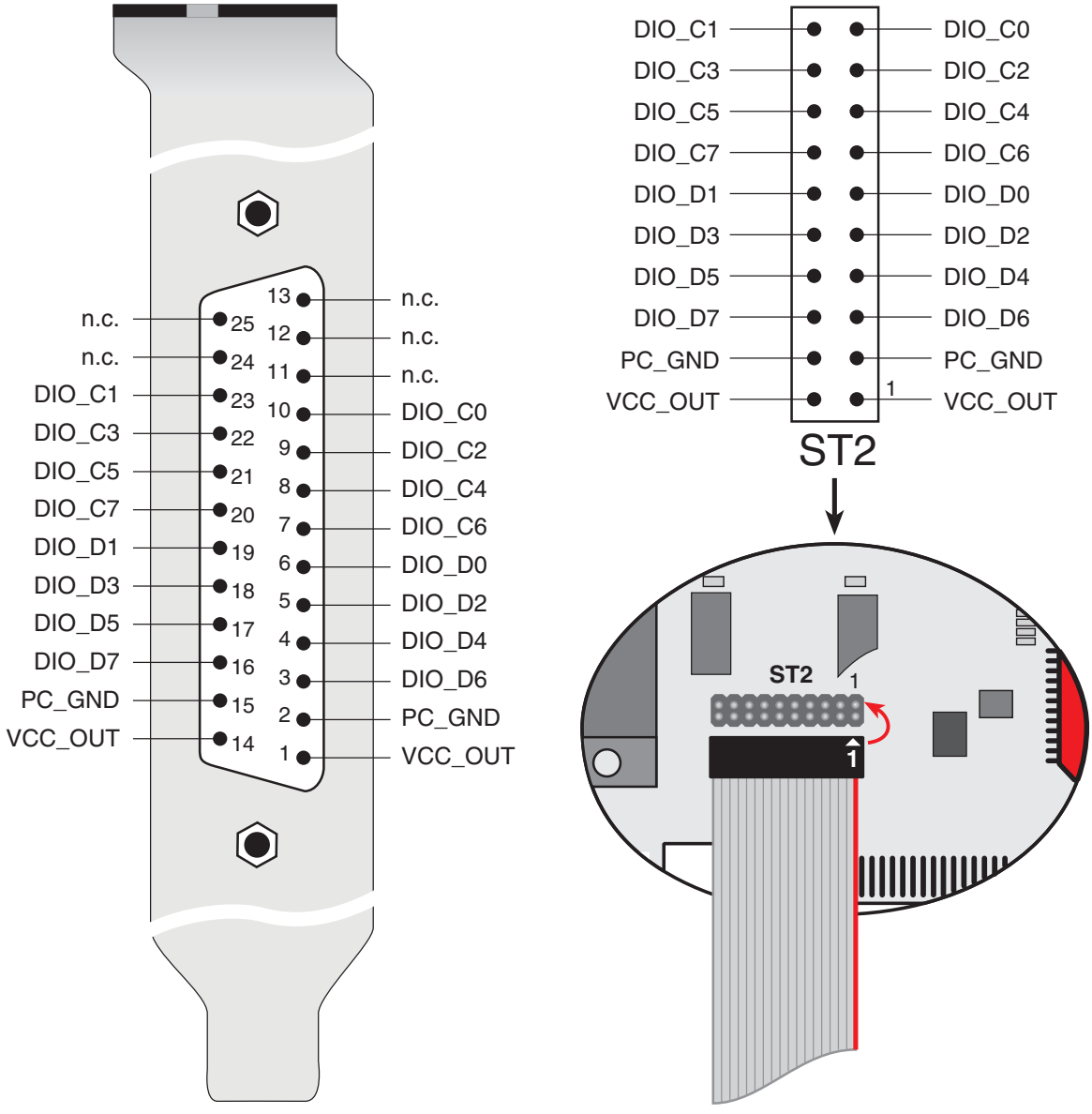


Diagram 28: Auxiliary connector ST2 for ME-4600 (top view)

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

Connector adapter for adaption of a ME-2x00/3000 application to a board of the ME-4600 series.

ME-MultiSig System

Extended multiplex and signal conditioning system:

- Analog multiplexing up to 4096 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 9: 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

Digital-I/O

ME4000_DIO_PORT_A	0
ME4000_DIO_PORT_B	1
ME4000_DIO_PORT_C	2
ME4000_DIO_PORT_D	3
ME4000_DIO_PORT_INPUT	0x00040201
ME4000_DIO_PORT_OUTPUT	0x00040202

Counter

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

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

Table 9: Constant definitions

Constant	Value
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 9: Constant definitions

F Index

Function Reference

A