Programmable Digital Pulse Controller

Firmware Version 1-00

User Manual

Document version 6, created on Jul-06-2021

CGC Instruments
Hübschmannstr. 18 | D–09112 Chemnitz
Tel.: +49 (371) 355 098–55
Fax: +49 (371) 355 098–60
internet: www.cgc-instruments.com
e–mail: info@cgc-instruments.com

Contents

Figure List

Table List

Technical Data

Characteristics

- 1 digital oscillator and 4 digital pulse generators
  resolution: 10 ns, length: 32 bit (maximum delay/period: 42.9 s)
- arbitrary configuration by selectable
  18 trigger and 20 output sources
- control of up to 4 power switches
- up to 7 digital inputs or outputs
  connectors: LEMO
  signal level: TTL, log. 0: 0..0.4 V, log. 1: 2.4..5.0 V
  input impedance: 50 Ω or 50 kΩ pull-up to +5 V
  output impedance: 50 Ω
- monitoring of up to 3 temperature sensors, control of up to 3 fans
- supply voltage supervision
- 16-bit RISC microcontroller running at 16 MHz
- FPGA clocked with 100 MHz
- non-volatile memory (NVM) data space: 16 KB
- maximum number of stored user configurations: 126

Digital Interface

- USB interface according to USB 2.0 standard
  connector: USB plug type B
  data transfer rate: up to 12 MBit/s (*Full Speed*)
  effective data transfer rate: >100 kBit/s

or

- RS-232 interface with all handshake lines
  default data rate: 9600 baud
  maximum data rate: 230.4 kbaud
  communication protocol: 8 data bits, 2 stop bits, even parity

Description

The pulse controller is a module that can be used to control up to four power switches. Its block diagram is shown in Fig. 1.

The device is supplied by a switching-mode power supply unit (PSU in Fig. 1). It has a universal voltage input that accepts AC voltages between 85 V and 264 V. It produces three output voltages: 12 V, 5.0 V, and 3.3 V required for the operation of the device.

The controller contains a 16-bit RISC microcontroller running at 16 MHz (CPU) that is connected to an FPGA (Field Programmable Gate Array). Both components are supplied by the 3.3-V supply rail and exchange data via a serial link. The CPU supervises all supply voltages and measures the temperatures monitored up to three sensors TSens0-2.

The CPU can control up to three fans Fan0-2. The fans are supplied by the 12-V supply rail and are controlled by a PWM (Pulse-Width Modulation) signal generated by the CPU. The CPU measures their rotational speed by monitoring their tachometric outputs and regulates the PWM signals accordingly.

The device status is indicated by an LED controlled by the CPU. The CPU can turn any of the three colors of the RGB-LED on or off, and thus produce 7 different LED colors or turn the LED off.

The CPU further implements communication with a non-volatile memory (NVM). The communication uses a standard serial protocol with a speed of several Mbaud. This allows for fast response times.

For communication with a host computer or another controller, either a USB or an RS-232 interface can be used. The USB interface uses a serial converter and, on the PC side, a virtual port drives is installed so that the communication via both interfaces uses the same serial protocol. As a result, the control utility is independent of the used interface.

The pulse generators and the control of the power switches are implemented in the FPGA. The FPGA is clocked by a quartz oscillator XO, its frequency is internally multiplied to 100 MHz. The value of 100 MHz is the internal time base of the FPGA, this frequency determines the time resolution of 10 ns of the oscillator(s) and the pulse generators.

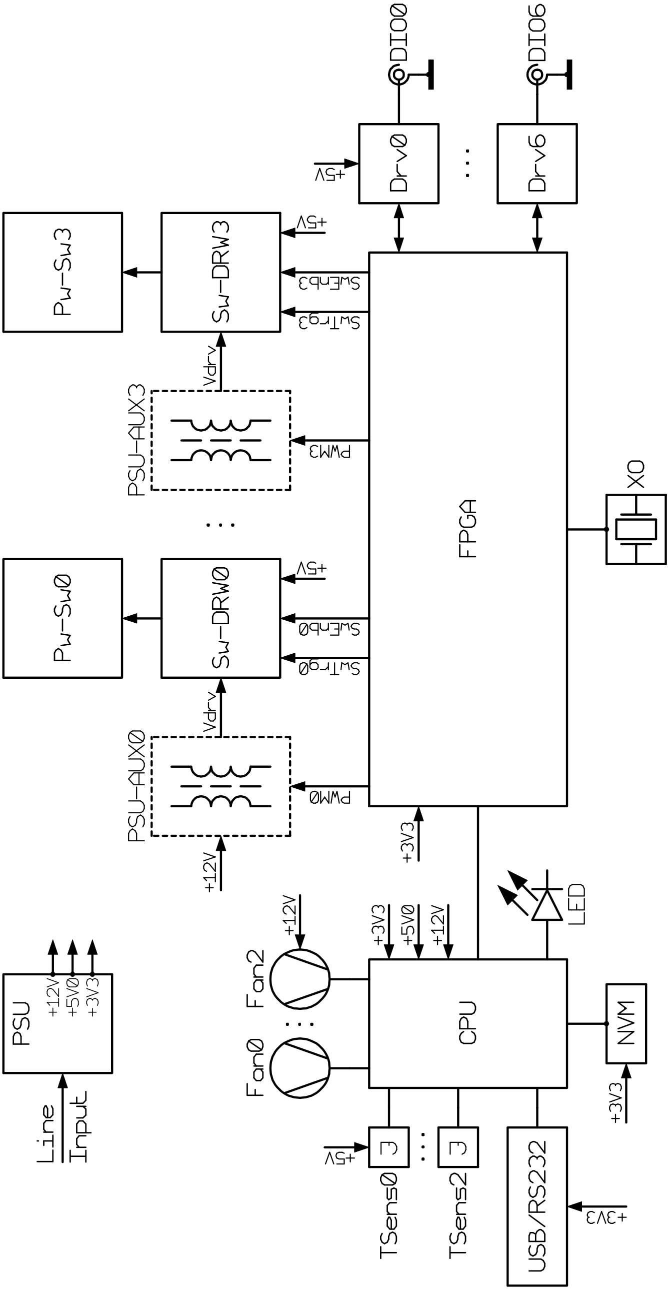

Fig. 1. Block diagram of the device.

The FPGA also produces PWM signals for controlling auxiliary power supply units PSU-AUX0-3. These power supply units provide isolated supply voltages for the switch drivers Sw-DRW0-3. The switch drivers are controlled by two signals each - the trigger and enable signals SwTrg0-3 and SwEnb0-3 (see section "Power Switches").

The FPGA further controls the digital inputs and outputs DIO0-6. They

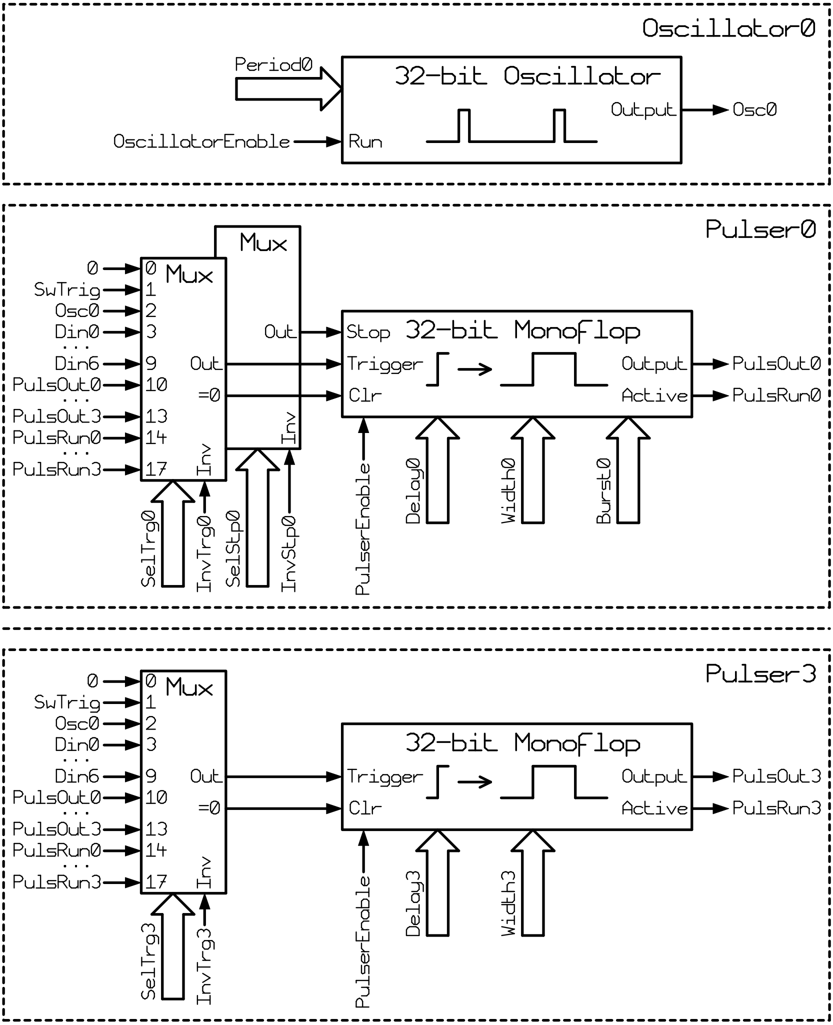

Fig. 2. Block diagram of the pulse controller.

Tab. 1. Assignment of trigger (SelTrg0-3) and stop (SelStp0-3) inputs of the pulse generators.

Select[n]	Value	Explanation
0	0	Logic 0
1	SwTrig	Software trigger
2	Osc0	Output of the oscillator #0
3	Din0	Digital input DIO1
...	...	...
9	Din6	Digital input DIO7
10	PulsOut0	Output of the pulse generator #0
...	...	...
13	PulsOut3	Output of the pulse generator #3
14	PulsRun0	Running state of the pulse generator #0
...	...	...
17	PulsRun3	Running state of the pulse generator #3

are connected to the FPGA via driver circuits Drv0-6 which can be configured as outputs with an internal impedance of 50 Ω, inputs, or inputs with a termination of 50 Ω (see section "Digital Inputs and Outputs").

The FPGA routes the signals from the digital inputs and outputs and the pulse generators to the control signals of the switch drivers via multiplexers. The configuration of the FPGA is freely programmable, this provides a large variety of possible functions of the device. The power switches can be controlled by external signals applied to the digital inputs or by internal signals provided by the pulse generators, whereas the digital outputs can supply signals for controlling external parts of an experimental setup.

Pulse Controller

The controller integrates one digital oscillator and four digital pulse generators, two of them have a burst capability (see Fig. 2). These modules are clocked by 100 MHz, thus providing a time resolution of

10 ns. The maximum pulse delay, pulse width or oscillation period are about 2^{32} clock pulses, i.e. about 42.9 s.

The digital oscillator (module Oscillator0 in Fig. 2) is a free running multivibrator with a period defined by a 32-bit long integer number. The oscillator can be stopped or started at any time by the control signal OscillatorEnable. If enabled, it provides a 1-clock (10 ns) wide positive pulse at its output at the end of the programmed period. The period in clock pulses is given by the 32-bit number Period plus 2. The minimum value of Period is 1, resulting in a period of 3 clock pulses (30 ns).

The pulse generators (modules Pulser0-3 in Fig. 2) are digital monoflops. They are triggered by a rising slope at the trigger input and produce a positive pulse with a specified width (32-bit integer numbers Width0-3) after a specified delay (32-bit integer numbers Delay0-3). The polarity of the trigger signal can be inverted by the control signal InvTrg0-3. The trigger source is selected by an 18-channel multiplexer controlled by the respective 5-bit integer number SelTrg0-3 (see Fig. 2 and Tab. 1). The trigger signal can be inverted by setting the control value InvTrgN to 1.

As a trigger input, the output of the oscillator (signal Osc0), any output of the pulse generators (signals PulsOut0-3), any output reflecting their running state (signals PulsRun0-3), or the external trigger sources, i.e. the digital inputs (signals Din0-6), can be selected. The trigger input can also be set to 0 or to software trigger (signal SwTrig). If the level 0 is selected, the particular pulse generator is stopped by the signal Clr. By inverting the trigger level, i.e. by setting the corresponding signal InvTrgN to 1, the pulse generator is triggered. Note that by setting the signal InvTrgN to 0 again, the pulse generator is stopped immediately.

An alternative way of software triggering is to select the input SwTrig. The signal SwTrig is generated in the software-trigger engine (see Fig. 3). Both input signals SwTrigger and SwPulse are controlled by the software, the latter triggers a monoflop on its 0-to-1 transition. The monoflop produces a 1-clock (10 ns) wide positive pulse that is XORed with the signal SwTrigger, thus inverting the output signal SwTrig for one clock period.

Each pulse generator is a combination of two coupled non-retriggerable monoflops. The first one with a pulse width defined by the numbers Delay0-3 is triggered by the input signal Trigger. After its delay, the second monoflop with a pulse width defined by the numbers Width0-3 is launched and the output Puls0-3 of the particular module is activated. The pulse delay in clock pulses is given by the 32-bit number DelayN plus 3. The minimum value of DelayN is 1, resulting in a delay of 4 clock pulses (40 ns). Similarly, the pulse width in clock pulses is given by the 32-bit number WidthN plus 2. The minimum value of WidthN is 1, resulting in a delay of 3 clock pulses (30 ns). Note that by setting DelayN or WidthN to 0, the pulse generator is stopped.

Two pulse generators (modules Pulser0-1 in Fig. 2) have a burst capability, it is controlled by the 24-bit integer numbers Burst0-1. If the respective value is reset, the pulse generator acts like the conventional module without the burst capability (modules Pulser2-3 in Fig. 2) as described in the previous paragraph. If BurstN is set to any nonzero value, it specifies the number of output pulses that are generated after one trigger event. The first pulse is triggered by the trigger input, the next ones by the trailing edges of the previous output pulses. The maximum pulse number in one burst is limited by the 24-bit number Burst0-1 to about 16.8 million.

The pulse burst can be interrupted by activating the input Stop. The stop input disables the trigger for the next output pulse, i.e. when activated, the current output pulse is completed without any interruption but no more pulses are generated until a new trigger event at the trigger input is received. Similarly to the trigger input, the source of the stop signal is selected by a 18-channel multiplexer controlled by the respective 5-bit integer number SelStp0-1 (see Fig. 2 and Tab. 1). The signal level can be inverted by setting the corresponding signal InvStpN to 1.

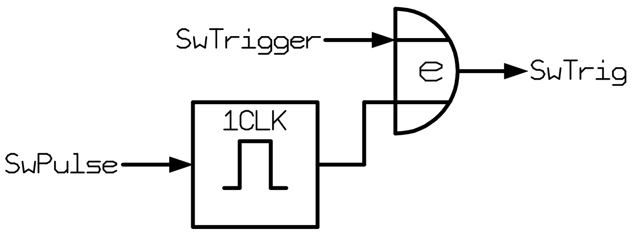

Fig. 3. Software-trigger engine.

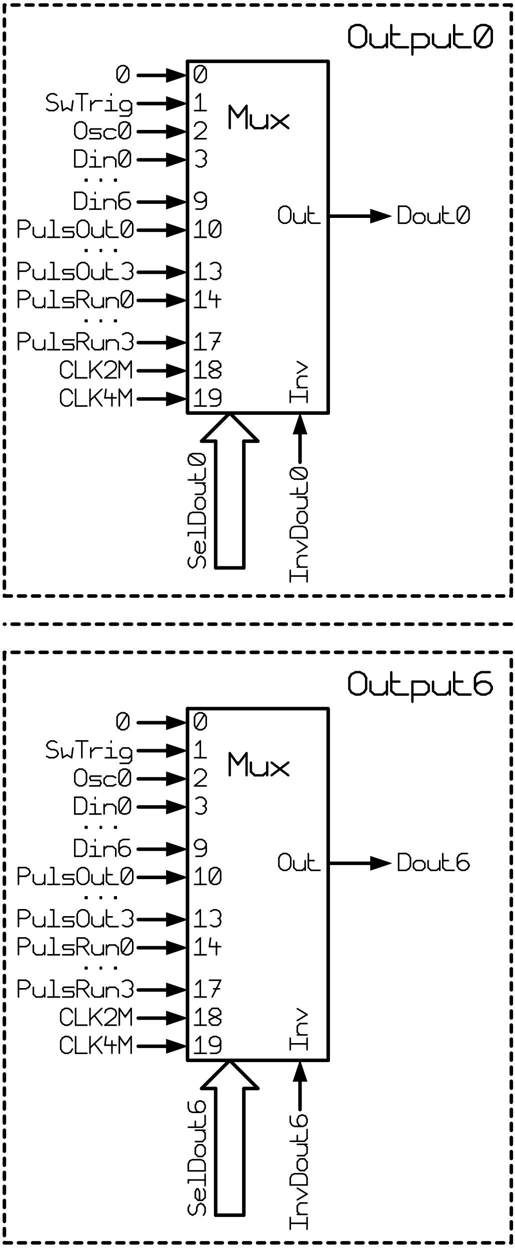

Fig. 4. Output Block Diagram.

Tab. 2. Available signal sources (SelDout0-6) for the digital outputs.

SelDout[n]	Value	Explanation
0	0	Logic 0
1	SwTrig	Software trigger
2	Osc0	Output of the oscillator #0
3	Din0	Digital input DIO1
...	...	...
9	Din6	Digital input DIO7
10	PulsOut0	Output of the pulse generator #0
...	...	...
13	PulsOut3	Output of the pulse generator #3
14	PulsRun0	Running state of the pulse generator #0
...	...	...
17	PulsRun3	Running state of the pulse generator #3
18	CLK2M	Internal quartz clock signal 2 MHz
19	CLK4M	Internal quartz clock signal 4 MHz

Each pulse generator can also be stopped if the signal PulserEnable is reset to 0 or if the control values DelayN or WidthN are set to 0. Note that a combination DelayN = 0 and WidthN ≠ 0 leads to an activated output, i.e. PulsOutN = 1. In all other stopped states, the output is reset, i.e. PulsOutN = 0.

The symmetrical architecture of the pulse generators offers a large variability of configurations. The pulse generators can be chained to produce complex pulse sequences. They can be triggered periodically by the internal oscillator, by an external event, or by the software. Unused channels can be disabled.

Digital Inputs and Outputs

The pulse controller can control up to seven digital outputs (see Fig. 4).

Tab. 3. Function of the digital inputs and outputs DIO1-7.

Configuration Bits		Output	Output State
OutEnb	TermEnb	Dout	
0	0	X	Input with a 50-kΩ pull-up (default)
0	1	X	Input with a 50-Ω termination
1	X	0	Output 0
1	X	1	Output 1

Each digital output can be assigned to any of the available 20 signal sources (see Tab. 2). The first 18 signals are identical to the available trigger and stop inputs of the pulse generators (see Fig. 2 and Tab. 1). However, the digital outputs can provide two additional clock signals (CLK2M and CLK4M) for synchronization purposes. These can be output by the master device or by any other stable clock source and the slave devices receive this clock and synchronize their operation accordingly. This enables an absolutely synchronous operation of many pulse controllers over virtually any time period.

Besides the synchronization feature, each output can be configured in a similar way to the trigger and stop inputs of the pulse generators (see Fig. 2 and Tab. 1). The output may be permanently set to 0, connected to the software-trigger signal SwTrig, to the output of the oscillator (signal Osc0), to the external digital input signals (Din0-7), or to the pulse generator (signals PulsOut0-3 or PulsRun0-3). The polarity of the output signal can be inverted by the control signal InvDoutN.

The digital inputs and outputs are available as coaxial connectors on the front panel of the device labeled as DIO1-7. By default, they function as digital inputs with a weak pull-up resistor (see Tab. 3). When the terminal DION should be used as a signal input, it is advisable to use a signal source with an output impedance of 50 Ω to generate the particular signal, connect it via a 50-Ω coaxial cable to the terminal DION, and activate the 50-Ω termination for the respective digital input DinN-1. This provides the best signal forms but also reduces the output amplitude of the signal source to 50%.

The default configuration without any termination is suitable for short signal cables or for signal sources that are not strong enough to drive the 50-Ω termination resistance in the pulse controller. In such cases,

Tab. 4. Signal monitoring (SignalState bits).

SignalState[n]	Value	Explanation
0	SwTrig	Software trigger
1	Din0	Digital input DIO1
...	...	...
7	Din6	Digital input DIO7
8	PulsOut0	Output of the pulse generator #0
...	...	...
11	PulsOut3	Output of the pulse generator #3
12	PulsRun0	Running state of the pulse generator #0
...	...	...
15	PulsRun3	Running state of the pulse generator #3

the signal form should be checked by an oscilloscope and possible overshooting or ringing that typically occurs after fast signal slopes should be minimized by introducing serial resistors into the signal lines. However, distortions of the signal in the order of 10-20% of its amplitude usually do not influence the functionality of the inputs since they use Schmitt-trigger gates, i.e. they are insensitive to noise levels of up to their hysteresis of 0.5 V minimum.

To check the signal quality seen by the particular digital input, the received signal DinN can be routed to another digital terminal DIO, where it can be monitored by an oscilloscope. To achieve this, the selected terminal has to be configured as an output (see Tab. 3) and assigned to the signal DinN of interest (see Tab. 2).

When a particular terminal is configured as an output, it outputs digital signals with a 5-V amplitude. The internal impedance of the output driver is 50 Ω, thus, with a proper 50-Ω termination, the resulting output amplitude is 2.5 V. This is compatible with TTL signal levels. Without any termination, the output signal level is still TTL compatible, it also corresponds to the 5V-CMOS standard. The absence of the termination at the cable end may lead to distortions of the waveforms (see also the description of unterminated digital inputs in the previous paragraphs) and should only be used with short cables or very long signal lines, where the reflected signals usually do not substantially affect the signal quality.

Tab. 5. Function of dual-level power switches.

Control Signals		Switch State	
SwTrg*N*	SwEnb*N*	Neg-Out	Pos-Out
X	0	off	off
0	1	on	off
1	1	off	on

Please note that the digital inputs are active in any DIO configuration and permanently scan the signal levels at the DIO terminals. If a particular terminal is configured as a digital output, the corresponding signal DinN can be used to monitor the function of the output, i.e. to monitor the signal level at the DIO terminal. Evaluating the signal DinN, for instance, a short circuit, overloading or malfunction of the output can be discovered.

The signal levels of most internal signals of the pulse controller can also be monitored by the software (see Tab. 4). The 16-bit value SignalState reflects the output level of the software-trigger engine (signal SwTrig, see Fig. 3), the levels of the digital terminals DIO1-7 (signals Din0-6), and the outputs as well as the running states of the pulse generators (PulsOut0-3 and PulsRun0-3). Only the oscillator output (signal Osc0) cannot be read by the software. Since it switches to logic 1 for one clock period (10 ns) only, the majority of the read attempts would result in 0, i.e. they would not provide any usable information about the oscillator function.

Power Switches

The pulse controller can control up to four fast dual-level power switches. If the device contains multilevel switches, one switch is equivalent to several dual-level switch channels. For instance, trilevel switches (see section "Trilevel Power Switches") that provide fast switching between three different voltage levels are equivalent to two dual-level switch channels. Thus, one pulse controller can control a maximum of two trilevel switches.

One dual-level switch channel is controlled by two digital signals (see Tab. 5): trigger (SwTrg*N*, *N* = 0-3) and enable (SwEnb*N*). If the signal enable is inactive (logical 0), both switch branches are turned off, i.e. the switch output is in a high-impedance state. The active signal enable (logical 1) turns the switch on and the trigger input determines which branch is conductive. A low level at the trigger input (logical 0) provides a negative output voltage (Vout = Vneg), a high level (logical 1) a positive output voltage (Vout = Vpos).

The signals for controlling the switches (SwTrg*N* and SwEnb*N*) are generated in three modules (see Fig. 5). The first one selects the specific signal that should be used for the control and provides the signals CtrlTrg*N* and CtrlEnb*N* (see Fig. 6). In the second module, the mapping engine (see Fig. 7), these control signals are modified and the signals MapTrg*N* and MapEnb*N* are created. In the third module, optional delays for a precise timing are introduced and the final control signals SwTrg*N* and SwEnb*N* are generated (see Fig. 8 and Tab. 5).

Similarly to the trigger inputs of the pulse generators (see Fig. 2 and Tab. 1), the switch control signals CtrlTrg*N* and CtrlEnb*N* are selected by an 18-channel multiplexer controlled by the respective 5-bit integer number SelSwTrg*N* and SelSwEnb*N* (see Fig. 6 and Tab. 7). The control signals can be inverted by setting the control values InvSwTrg*N* or InvSwEnb*N* to 1. For instance, a switch can be enabled permanently if its enable input is assigned to an inverted logic 0, i.e. by selecting SelSwEnb*N* = 0 and InvSwEnb*N* = 1. Similarly, a certain switch can be enabled by an external signal connected to one of the digital inputs DIO*N* by selecting SelSwEnb*N* = 3-9.

The enable inputs CtrlEnb*N* are additionally anded with the bit Enb of the status register (see Tab. 9). This causes the switches to be disabled when the enable bit Enb is reset - independently of the selected control signals.

For more complex pulse sequences, e.g. for controlling a trilevel switch (see section "Trilevel Power Switches") that should supply an electrodynamic multipole with a square-wave signal for trapping ions and then switch the output to a constant voltage in order to extract the ions, a mapping engine is available (see Fig. 7 and Tab. 6). The function of this module is controlled by the signals MapTrgEnb and MapEnbEnb (see Fig. 5), i.e. the control signal Enable in Fig. 7.

If the module is disabled, i.e. Enable = 0, the mapping engine simply replicates its input signals at the output, i.e. it ensures that MapTrg*N* = CtrlTrg*N* and MapEnb*N* = CtrlEnb*N*. If enabled, i.e. Enable = 1, the mapping engine converts the four control signals (either SwTrg*N* or SwEnb*N*) into different signal levels MapTrg*N* and MapEnb*N* that are determined by the values State*N* (*N* = 0-4).

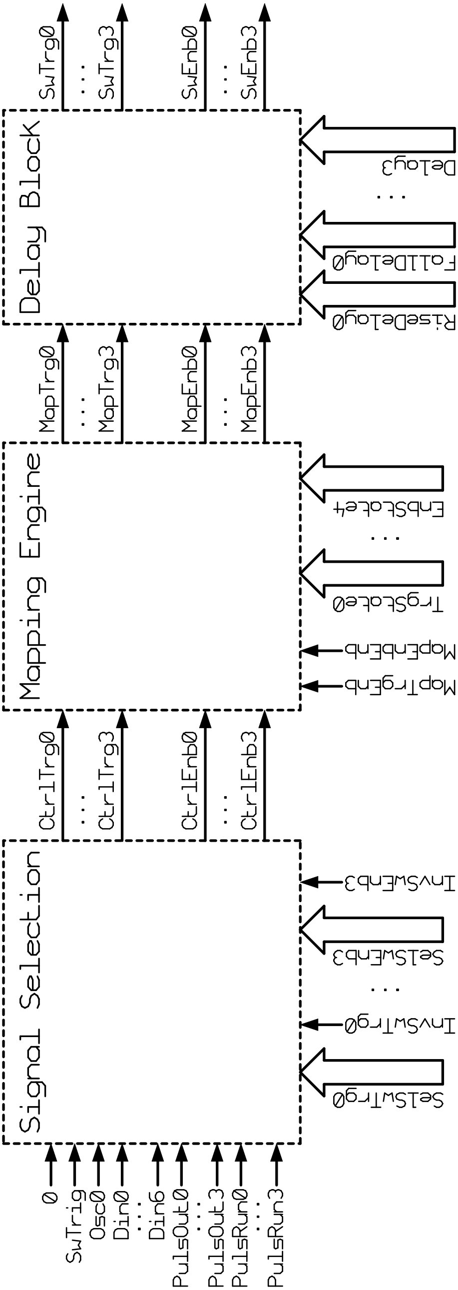

Fig. 5. Block diagram of the switch control.

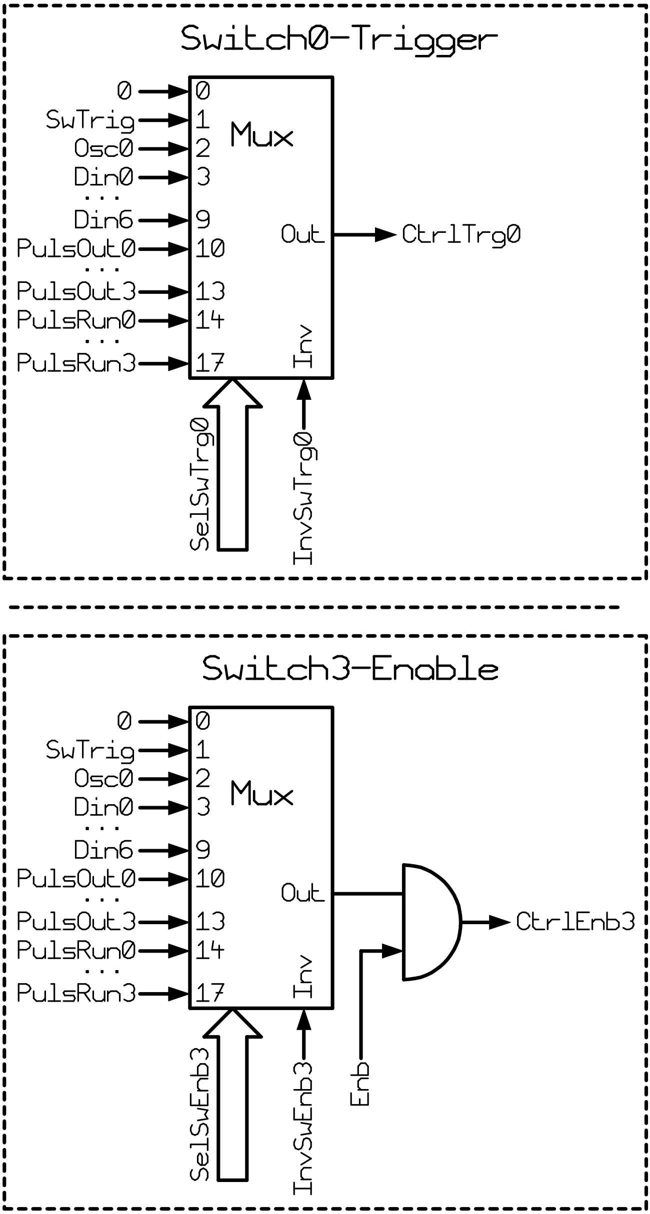

Fig. 6. Block diagram of the signal selection for the switch control.

Tab. 6. Function of the mapping engine.

Input Bits				Output State
In3	In2	In1	In0	
0	0	0	0	State0
X	X	X	1	State1
X	X	1	0	State2
X	1	0	0	State3
1	0	0	0	State4

For the example mentioned above, the square-wave signal can be generated by one pulse generator running in burst mode, e.g. the module Pulser0. Its negated running state (signal PulsRun0) will be routed to the trigger control CtrlTrg0 (SelSwTrg0 = 14 and InvSwTrg0 = 1) and its output (signal PulsOut0) to the next trigger control CtrlTrg1 (SelSwTrg1 = 10 and InvSwTrg1 = 0). The remaining trigger control signals (CtrlTrg2-3) will be kept at logic 0 (SelSwTrg2,3 = 0 and InvSwTrg2,3 = 0).

These signals CtrlTrg*N* correspond to the bits In0-3 in Fig. 7 and Tab. 6 respectively. When the pulse generator is not running, its negated running state (signal PulsRun0 = In0 in Fig. 7 and Tab. 6) is 1 and the trigger control signals for the switches are determined by the value State1 (value TrgState1 in Fig. 5). When the pulse generator is running, the trigger control signals for the switches toggle between the

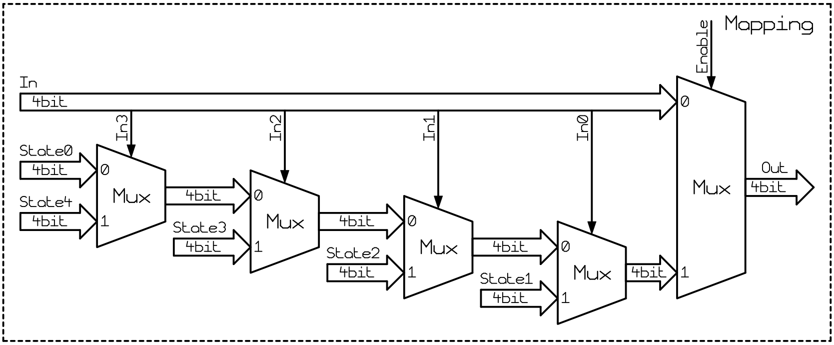

Fig. 7. Mapping engine for the switch control.

Tab. 7. Assignment of trigger (SelSwTrg0-3) and enable (SelSwEnb0-3) inputs of the power switches.

Select[n]	Value	Explanation
0	0	Logic 0
1	SwTrig	Software trigger
2	Osc0	Output of the oscillator #0
3	Din0	Digital input DIO1
...	...	...
9	Din6	Digital input DIO7
10	PulsOut0	Output of the pulse generator #0
...	...	...
13	PulsOut3	Output of the pulse generator #3
14	PulsRun0	Running state of the pulse generator #0
...	...	...
17	PulsRun3	Running state of the pulse generator #3

values State0 (value TrgState0 in Fig. 5) and State2 (value TrgState2 in Fig. 5). Thus, by defining the values State0-2, any signal waveform can be defined for controlling the switches.

For fine-tuning the output signals of the switches, variable delays can be inserted into the control signals of the switches (see Fig. 8). Depending on the values RiseDelay*N*, FallDelay*N*, and Delay*N* (*N* = 0-3), the trigger and the enable signals (CtrlTrg*N* and CtrlEnb*N*) can be delayed by about 0-15 ns with a resolution of approximately 1 ns. The resulting signals (SwTrg*N* and SwEnb*N*) are directly fed to the drivers of the power switches. For the trigger signals, each signal slope can be delayed independently from the opposite one (values RiseDelay*N* amd FallDelay*N*). In opposite, for the enable signals, both signal slopes are delayed by the same value (Delay*N*),

The independent delays of both signal slopes of the trigger signals are helpful, for instance, if two or more switches have to generate symmetrical synchronous signals, i.e. with slopes occurring at the same time. Using the variable delays, each signal slope can be shifted and different propagation delays in the driving circuitry, the power switch, and the cables can be compensated.

Trilevel Power Switches

The control scheme of a trilevel switch is more complicated (see Tab. 8). It is assigned to two switch channels (either 0 and 1 or 2 and 3), the first one controls the negative and the second one the positive branch, respectively.

For the normal operation of the switch, both enable inputs have to be set to a high level (SwEnb0/2 = SwEnb1/3 = 1). Conversely, if the switch should be disabled, both enable inputs have to be set to a low level (SwEnb0/2 = SwEnb1/3 = 0). These preferred operating conditions are shown in Tab. 8 in bold.

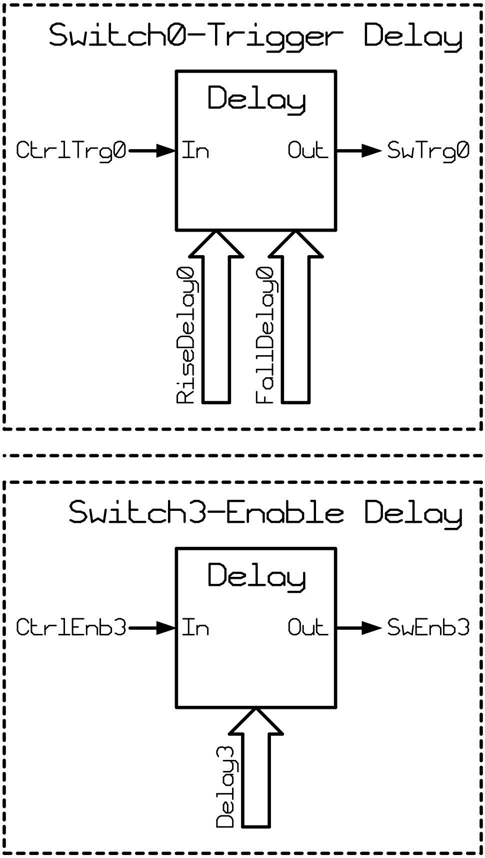

Fig. 8. Block diagram of the delay block for switch control.

Tab. 8. Function of trilevel power switches.

Control Signals				Switch State		
Switch 0/2		Switch 1/3				
SwTrg	SwEnb	SwTrg	SwEnb	Neg-Out	Mid-Out	Pos-Out
X	**0**	**X**	**0**	**off**	**off**	**off**
X	0	0	1	off	off	off
X	0	1	1	off	off	on
0	1	X	0	on	off	off
0	**1**	**0**	**1**	**on**	**off**	**off**
0	1	1	1	off	off	on
1	1	X	0	off	off	off
1	**1**	**0**	**1**	**off**	**on**	**off**
1	**1**	**1**	**1**	**off**	**off**	**on**

When the switch is enabled, the output voltage is negative (Vout = Vneg) if both trigger inputs have a low level (SwTrg0/2 = SwTrg1/3 = 0). The positive output voltage (Vout = Vpos) is set if the trigger inputs of the second switch channel have a high level (SwTrg1/3 = 1); in this case, the level of the trigger input of the first switch channel (SwTrg0/2) does not matter. If the output should, for instance, be switched between Vneg and Vpos, both trigger inputs should be toggled between 0 and 1 synchronously.

The middle output voltage (Vout = Vmid) is reached if the trigger input of the first switch channel has a high level (SwTrg0/2 = 1) and the trigger input of the second switch channel has a low level (SwTrg1/3 = 0).

The remaining rows in Tab. 8 show non-standard operation conditions. In most of them, only one switch branch is activated. This can be useful in special switching modes, please contact the manufacturer for further information.

The direct routing of the control signals to the switch control inputs is sufficient for most operations using the dual-level power switches. A typical example is the control of a switch pair driving an electrodynamic multipole for guiding or trapping ions or charged microparticles.

The control signal can be generated either by the oscillator and one pulse generator or by a pulse generator running in the burst mode. The first switch is driven by the control signal directly, the second switch by inverted control signal.

Controller Configuration

The function of the pulse controller can be modified by changing its configuration (see Tab. 9). The bits 0-6 can be set by the user to control the device (see function **COM_HVAMX4ED_SetControllerConfig**). The bits 8-10 are read-only, they can be obtained when the status is read out (see function **COM_HVAMX4ED_GetControllerState**).

The control logic of the device is shown in Fig. 9. The signal CLRn is the module reset signal active at 0. If the bit is 0, all device modules are reset and not operable. The reset state (CLRn = 0) is induced by the hardware reset signal HwResetN or by a low level of the master enable signal Enable. The hardware reset signal HwResetN is automatically generated by the hardware during power-on or restart (see function **COM_HVAMX4ED_Restart**).

To activate the master enable signal Enable, the hardware enable signal HwEnable must be active. The signal HwEnable is controlled by the supervising CPU, it is only enabled if no error has occurred. It is disabled if the temperature sensors have detected a temperature too low or too high or if any supply voltage has failed.

If no error condition is detected (i.e. HwEnable = 1) and the hardware reset has completed (i.e. HwResetN = 1), the module reset signal CLRn replicates the state of the control signal Enb, provided that the control signal PreventDis is 0. The control signal Enb enables the switches (see Fig. 5) and if the control signal PreventDis is reset (i.e. 0), the module reset signal CLRn is also cleared, i.e. set to 1. This enables the operation of all modules, i.e. the oscillator(s) and the pulse generators. If the signal PreventDis is set (i.e. 1), the inactive control signal Enb does not generate the module reset signal CLRn and the modules continue to operate.

The bits EnbOsc and EnbPulser of the configuration (see Tab. 9) control the oscillator(s) and the pulse generators. If the bits are set (i.e. 1) the respective module is enabled, if they are reset (i.e. 0), the respective module is stopped.

Tab. 9. Assignment of the control and status bits.

Status[n]	Value	Explanation
0	Enb	Enables the switches. When cleared, the oscillator(s) and the pulse generators are stopped. If PreventDis is cleared, the PSUs are also disabled.
1	EnbOsc	Enable input of the oscillator(s).
2	EnbPulser	Enable input of all pulse generators.
3	SW_Trig	Trigger input of the software-trigger engine.
4	SW_Pulse	Pulse input of the software-trigger engine.
5	PreventDis	Prevents disabling the device. When set, CLRn cannot be cleared, i.e. only the switches can be disabled, but not their PSUs.
6	DisDither	Disable the dithering for the PSUs.
8	Enable	Master enable. When cleared, CLRn is reset; read-only.
9	SW_Trig_Out	Output of the software-trigger engine; read-only.
10	CLRn	Clear (CLRn) of all modules (active low); read-only.

The bits SwTrig and SwPulse control the software-trigger engine (see Fig. 3). Its output signal SwTrigOut can be used for triggering the pulse generators or controlling other modules, its logic level is also available in the status (see Tab. 9).

The controller also provides control signals for auxiliary power supplies of the power switches. The function of these signals cannot be configured. The user can only decide whether the control signals are modulated using a dithering technique to reduce the spectral noise amplitude at the switch outputs (signal DisDither in Tab. 9). If the bit DisDither is set (i.e. 1), the auxiliary power supplies run at a constant frequency of typically 125 kHz. If the bit DisDither is reset (i.e. 0), the operating frequency of the power supplies fluctuates around the central frequency of 125 kHz (about ±10%). If a particular application can better tolerate a coherent noise, set the bit DisDither, this will produce a narrow frequency peak at 125 kHz and its harmonics. In contrast to that, if the bit DisDither is reset, the noise at the switch output will have a broad frequency spectrum with wide peaks, but with a much lower spectral amplitude compared to the operation at a constant frequency.

Configuration Management

The configuration of the pulse controller is controlled by software. The current configuration is stored in a non-volatile memory (NVM) and is automatically restored when the device is started. The user can define and save up to 126 additional configurations in the non-volatile memory, they can easily be applied by a software command. Besides the numbering, each configuration can be labeled by a unique name or description text. By applying the stored configurations, the pulse controller can be rapidly reconfigured for a new application or a different measurement procedure. For more details, refer to the section "Configuration Management" which describes the usage of the control utility `COM-HVAMX4ED-Control`.

Remote Control

The pulse controller is equipped with a USB or an RS-232 data interface which allows the user to transfer configuration data to or from the device and remotely control it. The data interface is galvanically connected to the case of the device. When establishing a connection to a

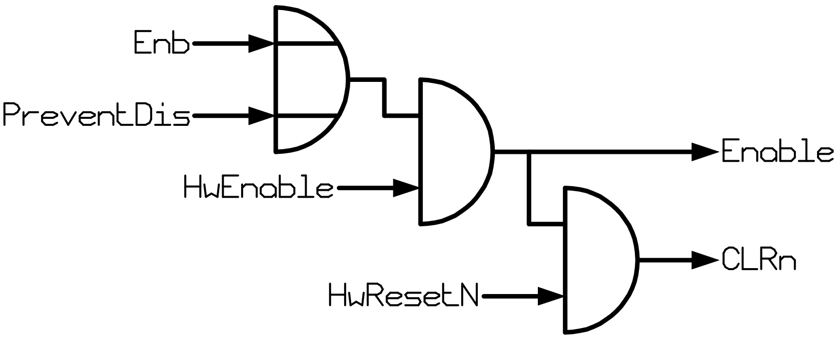

Fig. 9. Device control logic.

host computer, a large ground loop is created that can influence the performance of the experimental setup. Note that an improperly grounded power switch may produce strong electromagnetic interference (EMI) and disrupt the operation of the connected control computer or other devices.

The software package (see sections "Software Utilities") contains a utility for controlling the device and uploading or downloading data (see section "Utility COM-HVAMX4ED-Control"). Furthermore, a utility for upgrading the firmware is available (see section "Utility Flash-Loader").

Quick Setup Guide

The device is shipped with several sample configurations in the non-volatile memory (NVM). To setup the device, follow these steps:

- Connect the mains cable and power the device on. After the startup sequence, the power LED should light up green or yellow. If it blinks, an error has occurred. You should continue with establishing communication to be able to obtain the detailed device status and find the reason for the error.
- Connect a PC to the USB or RS-232 connector and install the port driver if necessary (see the section "Driver Installation").
- Copy the software to a suitable directory on your computer and start the utility `COM-HVAMX4ED-Control.exe` with a command-line parameter indicating the number of the port to which the device is connected (see section "Utility COM-HVAMX4ED-Control" for more details).
- Check the communication with the device by pressing 'p', 'n', 'v', and 'd'. The utility should show the product identification, the product number, the firmware version, and the firmware date. See section "System Monitoring" for more details.
- Check the current configuration of the device by pressing 'o', 'Pp', 'Dd', and 'Rp'. The utility should show the setting of the oscillator, the pulse generators, the digital inputs and outputs, and the switches. If desired, modify the settings. Refer to sections "Pulse Generator Control", "DIO Settings", and "Switch Settings" for more details.
- Examine the main device status by pressing 'i' (see section "Device Monitoring"). If you cannot see the LED on the front panel of the device directly, you can press a lowercase 'l' to query its state once or a capital 'L' to monitor it periodically.
- If the device status indicates an error and the LED is blinking, press 's' to obtain the detailed device status (see section "Device Monitoring"). The detailed status should help you find the reason for the error.
- If the device is disabled and the LED is glowing yellow, you need to enable the device to activate the integrated pulse generators and the power switches. Press 'W' and enter 7 to enable all components of the device (see section "Controller Settings" for more details). If

the LED on the front panel does not start to glow green repeat the previous two points and look for the errors.

- If the device is enabled, i.e. if the power LED is glowing green, the device should be active. Depending on the current configuration (see above), you can start to test the function of the digital inputs and outputs, and the power switches. You can apply the supply voltage to the switches and check the output signal(s).
- If the device does no behave as desired, save the current configuration to a file by pressing 'Ks' and entering the filename. Open the file in an ASCII editor and change the entries to adjust the device function. Refer to section "Configuration Management" for more details. Load the new configuration from the edited file by pressing 'Kl' and entering the filename. Repeat this sequence until the desired response is achieved.
- If the device remains disabled after the new configuration has been loaded, even though the entry "DeviceEnable" is set to "Y", check the enable state of the device controller by pressing 'j' (see section "Configuration Management"). If the enable state is off, turn it on by pressing 'Jy'. With this setting, the device should be activated in accordance with the entry "DeviceEnable". If the enable state is off, you will need to activate the device manually by pressing 'W' and entering 7 (see above).
- If desired, create further device configurations and upload them (for details see section "Configuration Management").

If you encounter any problems, carefully read the corresponding section in this manual. If you cannot resolve the issue, contact the manufacturer.

Software Utilities

The software utilities can be found in the directory "Program" of the enclosed software package. Before using them with a device with a USB interface, the virtual USB port driver must be installed (see section "Driver Installation"). The utilities do not require any additional installation, you only need to copy them to a suitable directory on your computer.

Before starting the utilities, you need to obtain the number of the port to which the device is connected. Devices with an RS-232 interface have to be connected to a physical serial port (for instance COM1). In Windows™ systems, you can find out its number using the device manager. Devices with a USB interface use a virtual serial port for communication. The details are described in the section "Driver Installation".

Utility COM-HVAMX4ED-Control

`COM-HVAMX4ED-Control` is a Windows™ program that runs in text mode. It enables you to control and monitor the pulse controller, manage its configurations, and backup and restore its data. Executing the utility `COM-HVAMX4ED-Control.exe` in a Windows™ command shell[†] without any additional parameters displays a help text with the list of all available commands:

```
COM-HVAMX4ED-Control
```

To start the program without any error messages, at least the number of the COM port must be given as a parameter:

```
COM-HVAMX4ED-Control 6
```

This command starts the utility `COM-HVAMX4ED-Control` and assumes that the device is connected to the (virtual) port COM6. When successful, the utility displays the following message:

[†] Press the Windows key + R to open the "Run" dialog box. Type "cmd" and press Enter, this opens a window with a command prompt. Then change the directory to the one containing the program files using the command "cd". Finally, execute the given command by copying & pasting and pressing "Enter". A better and more comfortable alternative to the Windows™ command shell are utilities such as "Total Commander", "File Commander/W", or "File and archive manager (FAR)". Please use an internet search engine to find out how to obtain these applications.

```
COM6 opened for the communication port 0
Press '?' for help
```

The utility enters the interactive mode and waits for command input.

In case of any problems, check whether the port number matches the system settings and whether the connected device is powered on and working properly. If an error occurs, please consult the section "Error Codes". Tables 22 and 23 explain the possible error messages; they should help you to locate the reason for the software failure.

To check the communication, press the 'p' key to obtain the product identification text. The device should respond as follows:

```
Product identification: HV-AMX-CTRL-4ED, Rev.2-10
```

If the device responds properly, you can enter other program commands. Press '?' to obtain the help listing of all available commands[‡].

In practice, you may prefer to use the command line mode instead of the interactive mode. For instance, the former allows you to save complete commands in batch files for repeated usage.

Tables 10-21 summarize all allowable command line parameters of the utility `COM-HVAMX4ED-Control`. Besides the parameters, the tables also list the functions of the software interface (see section "Software Interface") that are called by the utility when the respective parameter is executed. Refer to the descriptions of the functions for more details. Note that several command line parameters, e.g. most of the commands for managing the configurations, do not have any equivalent functions. They either call several functions or they use functions not intended to be called by the user directly.

The parameters are processed from left to right. When encountering an error in the command line, the program stops with an error text indicating the command line parameter in which the error occurred and displaying the help text with the list of all available commands.

In several cases, the command line parameters can be specified as either a lowercase or an uppercase character. Use capitals, i.e. uppercase characters, if a continuous operation should be initiated. In

[‡] Note that keyboard layouts different to the US one may cause issues when evaluating several characters. We recommend to switch to the US keyboard layout when using the utility COM-HVAMX4ED-Control in the interactive mode.

Tab. 10. Command line parameters of the program `COM-HVAMX4ED-Control` - General control.

Parameter	Explanation
`-b, -B`	get the device buffer status (see function **`COM_HVAMX4ED_GetBufferState`**)
`-z, -Z`	purge the communication (see function **`COM_HVAMX4ED_DevicePurge`**)
`-t, -T`	terminate the program
`-q, -Q`	quiet mode
`-g`	debug mode
`-G`	debug mode with output into `Debug.txt`
`-$`*`speed`*	set the communication speed to the value *`speed`* (see function **`COM_HVAMX4ED_SetBaudRate`**)
`-!`	restart the device (see function **`COM_HVAMX4ED_Restart`**)
`-?`	show the help text with the list of all available commands

the continuous mode, an operation is repeated until the user interrupts it by pressing a certain key - usually Esc.

Numerical integer values can be specified either in decimal code or in hexadecimal or binary ones when characters `h` or `b` are appended. This implies that, for example, `16`, `10h` or `10000b` all specify the same value, namely sixteen.

If you wish to specify a name parameter containing spaces or special characters, such as a pathname of a file containing spaces, use the conventions valid for your operating system. In Windows™ systems, for instance, enclose the names in quotation marks.

General Control

The command line parameters for the general control are listed in Tab. 10. They are intended to control the function of the utility.

The communication speed defaults to 9600 baud (see section "Software Interface"). This is usually sufficient for any short data transfer but may be too slow when the device data are queried periodically or when large data amounts should be transferred. In such cases, the communication speed can be set to a higher value using the parameter `-$`. To set the communication speed to the maximum supported value of 230.4 kbaud, enter the following command:

```
COM-HVAMX4ED-Control 6 -$230400
```

Note that the parameter `-$` is processed the same as any other parameter, thus the changed communication speed is only valid for all subsequent commands, i.e. for all parameters following the parameter `-$`. The communication speed can be changed several times within the command line. This makes it possible to increase the communication speed for data-intensive transmissions and later reset it to the default value for other commands. Also note that the communication speed is automatically reset to the default value of 9600 baud when the utility stops. For more details, see section "Software Interface" and function **`COM_HVAMX4ED_SetBaudRate`**.

If the command line includes the parameter `-t`, the program terminates without processing any following parameters. If you do not specify the parameter `-t` at all, the program does not stop and enters the interactive mode after having processed the complete command line.

The quiet mode activated by the parameter `-q` reduces the text output of the program. In contrast to that, the debug mode provides a detailed output for error analysis. It can be activated by the parameter `-g`. The parameter `-G` activates the debug mode and additionally creates a protocol file `Debug.txt` in which it saves the time of occurrence and the descriptions of all communication issues. In case of any problems that you cannot resolve yourself, describe the issues in detail and send this to the manufacturer together with the debug protocol `Debug.txt`.

To restart the device, the parameter `-!` has to be specified. The utility waits until the device completes its restart and responds again. During the restart, the complete configuration is stored in the non-volatile memory (NVM). Thus, a restart can be used to store data and prevent an accidental data loss.

Tab. 11. Command line parameters of the program `COM-HVAMX4ED-Control` - System monitoring.

Parameter	Explanation
`-v`	get the device firmware version (see function **`COM_HVAMX4ED_GetFWVersion`**)
`-V`	get the device hardware version (see functions **`COM_HVAMX4ED_GetHWType`** and **`COM_HVAMX4ED_GetHWVersion`**)
`-d`	get the device firmware date (see function **`COM_HVAMX4ED_GetFWDate`**)
`-p`	get the product identification text (see function **`COM_HVAMX4ED_GetProductID`**)
`-n, -N`	get the product number (see function **`COM_HVAMX4ED_GetProductNo`**)
`-u`	get the device uptime (see functions **`COM_HVAMX4ED_GetUptime`** and **`COM_HVAMX4ED_GetTotalTime`**)
`-U`	get the device uptime periodically
`-c`	get the CPU data (see function **`COM_HVAMX4ED_GetCPUData`**)
`-C`	get the CPU data periodically
`-h`	get the device housekeeping data (see function **`COM_HVAMX4ED_GetHousekeeping`**)
`-H`	get the device housekeeping data periodically

System Monitoring

The command line parameters for system monitoring are listed in Tab. 11. They are useful in case of a malfunction to collect the data necessary for describing the issues.

Using the parameter `-u` or `-U`, you can monitor the uptime of the device:

```
COM-HVAMX4ED-Control 6 -u -t
```

The device should respond as follows:

```
Device uptime: 7.16 sec.,   total: 47:47:05 sec.,
  Operation time: 7 sec.,   total: 32:44:13 sec.
```

The value `Device uptime` shows the time elapsed since the last (re)start of the device. Similarly, the value `Operation time` shows the time during which the device was active, i.e. not disabled. The values `total` are the respective cumulated values counted since the device was manufactured.

To obtain the housekeeping data of the device, use the parameters `-h` or `-H`:

```
COM-HVAMX4ED-Control 6 -h -t
```

The program output contains information about the supply voltages of the device and about the CPU temperature:

```
Device housekeeping:    12V: 12.079V,   5V0:
  4.988V,    3V3: 3.287V,    CPU: 24.98degC
```

The voltage values should match the corresponding label, i.e. 12 V, 5 V, and 3.3 V, respectively. Under normal conditions, the CPU temperature should not exceed 40˚C.

When contacting the manufacturer in case of any issues, please execute the following command:

```
COM-HVAMX4ED-Control 6 -q -V -v -d -p -n -u -h -t >
  Monitor.txt
```

The command output is redirected to the file Monitor.txt, please submit it together with the description of the observed issues. It should contain the following output:

```
Firmware version: 1-01
Hardware type: 9204h, hardware version: 1.00
Firmware date: Jun 18 2021
Product identification: HV-AMX-CTRL-4ED, Rev.2-10
Product number: 110401
Device uptime: 1:15:20.82 sec.,   total: 46:39:08
  sec., Operation time: 1:15:20 sec.,   total:
  31:36:16 sec.
Device housekeeping:    12V: 12.079V,     5V0:
  4.988V,   3V3: 3.287V,      CPU: 24.99degC
```

Tab. 12. Command line parameters of the program `COM-HVAMX4ED-Control` - Device monitoring.

Parameter	Explanation
`-l`	get the LED data (see function **`COM_HVAMX4ED_GetLEDData`**)
`-L`	get the LED data periodically
`-f`	get the fan data (see function **`COM_HVAMX4ED_GetFanData`**)
`-F`	get the fan data periodically
`-m`	get the sensor data (see function **`COM_HVAMX4ED_GetSensorData`**)
`-M`	get the sensor data periodically
`-i`	get the main device status (see function **`COM_HVAMX4ED_GetMainState`**)
`-I`	get the main device status periodically
`-s`	get the device status (see function **`COM_HVAMX4ED_GetDeviceState`**)
`-S`	get the device status periodically
`-x`	log the monitoring data
`-X`	create test protocol

Device Monitoring

The command line parameters for monitoring the device are listed in Tab. 12. They are useful in case of a malfunction when you are searching for the reason of unwanted issues.

Using the parameter `-l` or `-L`, you can monitor the state of the LED on the device's front panel:

```
COM-HVAMX4ED-Control 6 -l -t
```

The device should respond as follows:

```
LED data:    Red:  No,    Green: Yes,    Blue:  No
```

The shown values indicate whether the respective LED color is active or not.

To obtain the data of the device's fans, use the parameters `-f` or `-F`:

```
COM-HVAMX4ED-Control 6 -f -t
```

The utility displays the following summary:

```
Fan data:
  fan #1: enabled: Yes,   failed:  No,    set:
   0rpm,   measured:    0rpm,       PWM:   0.00%
  fan #2: enabled: Yes,   failed:  No,    set:
   0rpm,   measured:    0rpm,       PWM:   0.00%
  fan #3: enabled:  No,   failed:  No,    set:
   0rpm,   measured:    0rpm,       PWM:   0.00%
```

Similarly, data of the device's temperature sensors can be obtained by entering the parameters `-m` or `-M`:

```
COM-HVAMX4ED-Control 6 -m -t
```

The utility responds with the following output:

```
Sensor data:    #1:  1.36degC   #2: 22.55degC   #3:
  16.82degC
```

Check these values if the device is operated at a high output power and runs hot or even overheats.

The main status of the device can be queried by the parameters `-i` or `-I`:

```
COM-HVAMX4ED-Control 6 -i -t
```

The utility output shows the decoded status:

```
State:  Device active
```

Note that the status is also indicated by the LED. For the color codes, refer to the device manual.

The detailed device status can be obtained by the parameters `-s` or `-S`:

```
COM-HVAMX4ED-Control 6 -s -t
```

The utility output shows the status data and decodes the values of the relevant bits:

```
Device status: 00000000h =
    00000000000000000000000000000000b
        Controller disabled:  No
        Voltages failed:        CPU:  No,
    Fan:  No
        Fans failed:    #1:  No,        #2:  No,
    #3:  No
        Sensors hot:    #1:  No,        #2:  No,
    #3:  No
        Sensors cold:   #1:  No,        #2:  No,
    #3:  No
```

In case of any errors that have deactivated the device, the device status may help to identify the reason.

For long-time monitoring purposes, the parameter `-x` is intended. The command

```
COM-HVAMX4ED-Control 6 -x LogFile.txt -t
```

periodically queries the device data and saves then into the file `LogFile.txt`. The created file contains the values measured by the temperature sensors, the housekeeping data, and the rotational speed of the fans. The data is saved in ASCII format as a table with tabulator-separated values. It can be directly opened by spreadsheet programs like Microsoft Excel.

In case of issues that you cannot resolve, you may consider to create a test protocol:

```
COM-HVAMX4ED-Control 6 -X Protocol.txt -t
```

The command creates a text file `Protocol.txt` that you can submit to the manufacturer together with a detailed description of the observed issues.

Pulse Generator Control

The command line parameters for controlling the pulse generator are listed in Tab. 13. By using them, you can adjust or query the operating frequency of the built-in oscillator and the delay, burst size, and trigger values of the pulse generators (see section "Description" for more details).

Tab. 13. Command line parameters of the program `COM-HVAMX4ED-Control` - Pulse generator control.

Parameter	Explanation
`-o` `-O`	get the oscillator period (see function **`COM_HVAMX4ED_GetOscillatorPeriod`**)
`-o`*`Period`* `-O`*`Period`*	set the oscillator period to the value *`Period`* (see function **`COM_HVAMX4ED_SetOscillatorPeriod`**)
`-P`*`u`*`d`	get the pulse delay of the pulse generator with the number *`u`* (see function **`COM_HVAMX4ED_GetPulserDelay`**)
`-P`*`u`*`D`*`delay`*	set the pulse delay of the pulse generator with the number *`u`* to the value *`delay`* (see function **`COM_HVAMX4ED_SetPulserDelay`**)
`-P`*`u`*`w`	get the pulse width of the pulse generator with the number *`u`* (see function **`COM_HVAMX4ED_GetPulserWidth`**)
`-P`*`u`*`W`*`width`*	set the pulse width of the pulse generator with the number *`u`* to the value *`width`* (see function **`COM_HVAMX4ED_SetPulserWidth`**)
`-P`*`u`*`b`	get the burst size of the pulse generator with the number *`u`* (see function **`COM_HVAMX4ED_GetPulserBurst`**)
`-P`*`u`*`B`*`size`*	set the burst size of the pulse generator with the number *`u`* to the value *`size`* (see function **`COM_HVAMX4ED_SetPulserBurst`**)
`-P`*`u`*`c`	get the configuration of the pulse generator with the number *`u`* (see function **`COM_HVAMX4ED_GetPulserConfig`**)
`-P`*`u`*`C`*`config`*	set the configuration of the pulse generator with the number *`u`* to the value *`config`* (see function **`COM_HVAMX4ED_SetPulserConfig`**)
`-P`	get setting of all pulse generators

The oscillator period (parameters `-o` and `-O`) corresponds to the value `Period0` (see module `Oscillator0` in Fig. 2). Thus, the period in clock pulses is given by this number plus 2, i.e. the constant `COM_HVAMX4ED_OSC_OFFSET` (see the declaration file `COM-HVAMX4ED.h`).

Example: To set the oscillator period to 1 ms, execute the following command:

```
COM-HVAMX4ED-Control 6 -O99998 -t
```

The utility output shows the set value and calculates the corresponding period and frequency values:

```
Oscillator period set to 99998 (1.000000E-03s,
   1.000000E+03Hz)
```

The pulse delay and pulse width of the pulse generators (parameters `-Pud`, `-Puw`, `-PuD`, and `-PuW`) correspond to the values `Delay0-3` and `Width0-3` (see modules `Pulser0-3` in Fig. 2). The pulse delay in clock pulses is given by the values `Delay0-3` plus 3, i.e. the constant `COM_HVAMX4ED_PULSER_DELAY_OFFSET` (see the declaration file `COM-HVAMX4ED.h`). Similarly, the pulse width equals the values `Width0-3` plus 2, i.e. the constant `COM_HVAMX4ED_PULSER_WIDTH_OFFSET`.

Example: To set the pulse generator 1 to a pulse delay of 10 µs and a pulse width of 50 µs, execute the following command:

```
COM-HVAMX4ED-Control 6 -P1D997 -P1W4998 -t
```

The output shows the set values and calculates the corresponding timing here as well:

```
Pulse delay of pulser #1 set to 997 (1.000000E-05s)
Pulse width of pulser #1 set to 4998 (5.000000E-
   05s)
```

The burst size of the pulse generators (parameters `-Pub`, and `-PuB`) correspond to the values `Burst0-1` (see modules `Pulser0-1` in Fig. 2).

Example: To set the burst size of the pulse generator 0 to 500, execute the following command:

```
COM-HVAMX4ED-Control 6 -P0B500 -t
```

Tab. 14. Configuration values for the pulse generators.

Bit	7	6	5	4	3	2	1	0
Value	–	–	InvN	SelectN				

The command produces this output:

```
Burst size of pulser #0 set to 500
```

The trigger and stop selections of the pulse generators (parameters `-Puc` and `-PuC`) correspond to the values `SelTrg0-3` and `InvTrg0-3`, as well as `SelStp0-1` and `InvStp0-1` (see modules `Pulser0-3` in Fig. 2). The configuration value is a bit combination of the values `SelectN` and `InvN` (see Tab. 1 and Tab. 14). Note that the two most significant bits are not used for the configuration values. For more details and for the numbering of the configuration parameters, see function **`COM_HVAMX4ED_GetPulserConfig`** and Tab. 24.

Example: To set the trigger of the pulse generator 1 to the inverted (`Inv1` = 1) oscillator output (`Select1` = 2, see Tab. 1), execute the following command:

```
COM-HVAMX4ED-Control 6 -P2C22h -t
```

The command output summarizes the set value and displays the configuration in a readable format:

```
Configuration of pulser #2 set to 34=22h (negated
   oscillator 0)
```

Note that for the "pulser" numbering, Tab. 24 has to be considered.

DIO Settings

Several commands are available for configuring the digital input and output terminals DIO (see Tab. 15).

To query the settings of all terminals DIO, use the parameter `-D`, i.e. execute the following command:

```
COM-HVAMX4ED-Control 6 -D -t
```

The command produces this output:

```
DIO configuration:
```

Tab. 15. Command line parameters of the program `COM-HVAMX4ED-Control` - DIO settings.

Parameter	Explanation
`-D`	get the settings of all terminals DIO
`-Dni`	get the configuration of the input terminal DIO*n* (see function **`COM_HVAMX4ED_GetInputConfig`**)
`-DnIo` `-DnIO`	set the terminal DIO*n* as output (see function **`COM_HVAMX4ED_SetInputConfig`**)
`-DnIi` `-DnII`	set the terminal DIO*n* as input (see function **`COM_HVAMX4ED_SetInputConfig`**)
`-DnIt` `-DnIT`	set the terminal DIO*n* as input with termination (see function **`COM_HVAMX4ED_SetInputConfig`**)
`-Dno`	get the configuration of the output terminal DIO*n* (see function **`COM_HVAMX4ED_GetOutputConfig`**)
`-DnOconf`	set the configuration of the output terminal DIO*n* to the value *conf* (see function **`COM_HVAMX4ED_SetOutputConfig`**)

```
        DIO0:   output, configuration: 10=0Ah
(pulser 1 output)
        DIO1:   output, configuration: 10=0Ah
(pulser 1 output)
        DIO2:   output, configuration: 10=0Ah
(pulser 1 output)
        DIO3:   output, configuration: 10=0Ah
(pulser 1 output)
        DIO4:   input,  configuration: 0=00h (logic
0)
        DIO5:   input,  configuration: 0=00h (logic
0)
        DIO6:   input,  configuration: 0=00h (logic
0)
```

Note that the utility uses the logical numbering starting at 0, whereas the terminals are labeled on the front panel by numbers starting from 1, i.e. DIO1, DIO2, etc.

The terminal configuration can be set to either output, input, or input with termination. For instance, to set DIO0 to output, DIO3 to input, and DIO2 to input with termination, execute the following command:

```
COM-HVAMX4ED-Control 6 -D0Io -D3Ii -D2It -t
```

The response of the utility confirms the setting:

```
DIO0 configuration set to output
DIO3 configuration set to input
DIO2 configuration set to terminated input
```

Similarly, the following command shows the setting of the input terminal DIO1:

```
COM-HVAMX4ED-Control 6 -D1i -t
```

It provides the following information:

```
DIO1 configuration: output
```

If a terminal is configured as output, the parameter `-Dno` can be used to query or to set the signal that controls the terminal. The control is similar to selecting the trigger and stop signal sources of the pulse generators (see section "Pulse Generator Control" and Tab. 14, as well as section "Digital Inputs and Outputs", Fig. 4 and Tab. 2).

Example: To control DIO0 via the non-inverted (`Inv1` = 0) output of the pulse generator 1 (signal `PulsOut1`, i.e. `SelDout0` = 11 = 0Bh, see Tab. 2), execute the following command:

```
COM-HVAMX4ED-Control 6 -D0O0Bh -t
```

The command outputs the set value and shows the configuration in a readable format:

```
Output DIO0 configuration set to 11=0Bh (pulser 2
   output)
```

Analogously, the following command queries the setting of the output terminal DIO1:

```
COM-HVAMX4ED-Control 6 -D1o -t
```

It provides the following information:

```
Output DIO1 configuration: 10=0Ah (pulser 1 output)
```

Switch Settings

The commands for the switch settings are intended for configuring or querying the switch control signals, adjusting the delays of their edges, or managing the mapping configurations (see Tab. 16).

The switch configuration can be queried by the command

```
COM-HVAMX4ED-Control 6 -R -t
```

It displays the following summary:

```
Switch configurations:
        0 trigger: 0=00h (logic 0),     0 enable:
   0=00h (logic 0)
        1 trigger: 10=0Ah (pulser 1 output),    1
   enable:  32=20h (negated logic 0)
        2 trigger: 0=00h (logic 0),     2 enable:
   0=00h (logic 0)
        3 trigger: 0=00h (logic 0),     3 enable:
   0=00h (logic 0)
Switch delays:
        0 trigger rise: 0 (0.0ns),      0 trigger
   fall: 0 (0.0ns),      0 enable: 0 (0.0ns)
        1 trigger rise: 0 (0.0ns),      1 trigger
   fall: 0 (0.0ns),      1 enable: 0 (0.0ns)
        2 trigger rise: 0 (0.0ns),      2 trigger
   fall: 0 (0.0ns),      2 enable: 0 (0.0ns)
        3 trigger rise: 0 (0.0ns),      3 trigger
   fall: 0 (0.0ns),      3 enable: 0 (0.0ns)
Switch mappings:
        0 trigger: 0=00h,       0 enable:  0=00h
        1 trigger: 0=00h,       1 enable:  0=00h
        2 trigger: 0=00h,       2 enable:  0=00h
        3 trigger: 0=00h,       3 enable:  0=00h
        4 trigger: 0=00h,       4 enable:  0=00h
        trigger: disabled,      enable:  disabled
```

The configuration of the switch trigger and enable inputs can be set by the parameters `-R`*n*`T` and `-R`*n*`E`.

Tab. 16. Command line parameters of the program `COM-HVAMX4ED-Control` - Switch settings.

Parameter	Explanation
`-R`	get the setting of all switches
`-R`*n*`t`	get the trigger configuration of the switch with the number *n* (see function **`COM_HVAMX4ED_GetSwitchTriggerConfig`**)
`-R`*n*`T`*config*	set the trigger configuration of the switch with the number *n* to the value *config* (see function **`COM_HVAMX4ED_SetSwitchTriggerConfig`**)
`-R`*n*`e`	get the enable configuration of the switch with the number *n* (see function **`COM_HVAMX4ED_GetSwitchEnableConfig`**)
`-R`*n*`E`*config*	set the enable configuration of the switch with the number *n* to the value *config* (see function **`COM_HVAMX4ED_SetSwitchEnableConfig`**)
`-R`*n*`r`	get the rise delay of the trigger input of the switch with the number *n* (see function **`COM_HVAMX4ED_GetSwitchTriggerDelay`**)
`-R`*n*`R`*delay*	set the rise delay of the trigger input of the switch with the number *n* to the value *delay* (see function **`COM_HVAMX4ED_SetSwitchTriggerDelay`**)
`-R`*n*`f`	get the fall delay of the trigger input of the switch with the number *n* (see function **`COM_HVAMX4ED_GetSwitchTriggerDelay`**)
`-R`*n*`F`*delay*	set the fall delay of the trigger input of the switch with the number *n* to the value *delay* (see function **`COM_HVAMX4ED_SetSwitchTriggerDelay`**)
`-R`*n*`d`	get the delay of the enable input of the switch with the number *n* (see function **`COM_HVAMX4ED_GetSwitchEnableDelay`**)
`-R`*n*`D`*delay*	set the delay of the enable input of the switch with the number *n* to the value *delay* (see function **`COM_HVAMX4ED_SetSwitchEnableDelay`**)

Tab. 16 continued.

Parameter	Explanation
`-R`*n*`m`	get the mapping with the number *n* of the trigger inputs of the switches (see function **`COM_HVAMX4ED_GetSwitchTriggerMapping`**)
`-R`*n*`M`*map*	set the mapping with the number *n* of the trigger inputs of the switches to the value *map* (see function **`COM_HVAMX4ED_SetSwitchTriggerMapping`**)
`-R`*n*`n`	get the mapping with the number *n* of the enable inputs of the switches (see function **`COM_HVAMX4ED_GetSwitchEnableMapping`**)
`-R`*n*`N`*map*	set the mapping with the number *n* of the enable inputs of the switches to the value *map* (see function **`COM_HVAMX4ED_SetSwitchEnableMapping`**)

`-RYM`
`-RYm`
`-RyM`
`-Rym`
	enable the switch trigger mapping (see function **`COM_HVAMX4ED_SetSwitchTriggerMapping`**)

`-RNM`
`-RNm`
`-RnM`
`-Rnm`
	disable the switch trigger mapping (see function **`COM_HVAMX4ED_SetSwitchTriggerMapping`**)

`-RYN`
`-RYn`
`-RyN`
`-Ryn`
	enable the switch enable mapping (see function **`COM_HVAMX4ED_SetSwitchEnableMapping`**)

`-RNN`
`-RNn`
`-RnN`
`-Rnn`
	disable the switch enable mapping (see function **`COM_HVAMX4ED_SetSwitchEnableMapping`**)

The parameters `-R`*n*`t`/`-R`*n*`T` and `-R`*n*`e`/`-R`*n*`E` can be used to query or to set the signals that control the switch trigger and enable inputs. The control is similar to selecting the trigger and stop signal sources of the pulse generators (see section "Pulse Generator Control" and Tab. 14, as well as section "Power Switches", Fig. 5 and Tab. 7).

Example: To control the trigger input of switch 1 via the inverted (`Inv1 = 1`) output of pulse generator 3 (signal `PulsOut3`, i.e. `SelDout1` = 13 = 0Dh, see Tab. 7), execute the following command:

```
COM-HVAMX4ED-Control 6 -R1T2Dh -t
```

The command outputs the set value and shows the configuration in a readable format:

```
Configuration of switch #1 trigger set to 45=2Dh
   (negated pulser 4 output)
```

Analogously, the following command queries the setting of the trigger input of switch 2:

```
COM-HVAMX4ED-Control 6 -R2t -t
```

It provides the following information:

```
Configuration of switch #2 trigger: 0=00h (logic 0)
```

In a similar way, the delay of the control signals of the switches can be queried or set by the parameters `-R`*n*`r`/`-R`*n*`R`, `-R`*n*`f`/`-R`*n*`F`, and `-R`*n*`d`/`-R`*n*`D`.

Example: To set the delay of the rising slope of the trigger input of switch 2 to a value of 3 and the delay of the falling slope of the same signal to a value of 15, execute the following command:

```
COM-HVAMX4ED-Control 6 -R2R3 -R2F15 -t
```

The command displays the set values and also shows the approximate values of the delays:

```
Rise delay of switch #2 trigger set to 3 (1.5ns)
Fall delay of switch #2 trigger set to 15 (7.5ns)
```

To configure and query the mapping of the control signals of the switches, the parameters `-R`*X*`m`/`-R`*X*`M` and `-R`*X*`n`/`-R`*X*`N` can be used.

Example: To set the enable mapping #0 to 3, #1 to 2, and #4 to 0, and to enable it, execute the command

```
COM-HVAMX4ED-Control 6 -R0M3 -R1M2 -R4M0 -RYM -t
```

The command displays the following values:

```
Mapping #0 of switch trigger set to 3=03h
Mapping #1 of switch trigger set to 2=02h
```

Tab. 17. Command line parameters of the program `COM-HVAMX4ED-Control` - Controller settings.

Parameter	Explanation
`-a`	issue a positive software trigger pulse
`-A`	issue a negative software trigger pulse
`-a`*`TtPp`* `-A`*`TtPp`*	issue a software trigger pulse with defined parameters, *`TtPp`* are 4 Boolean arguments [0,1]
`-w`	get controller state (see function **`COM_HVAMX4ED_GetControllerState`**)
`-W`	get controller state periodically
`-w`*`Config`* `-W`*`Config`*	set controller configuration to the value *`Config`* (see function **`COM_HVAMX4ED_SetControllerConfig`**)

```
Mapping #4 of switch trigger set to 0=00h
Switch trigger mapping set to enabled
```

Controller Settings

Several commands are available for the controller setting. The device can be triggered by software or its configuration can be adjusted (see Tab. 17).

To control the software trigger engine (see Fig. 3), use the parameters `-a` or `-A`. The signals `SwTrigger` and `SwPulse` can be controlled by 4 Boolean arguments ('0' or '1') "*`TtPp`*". They control the device state in two cycles. The values "*`T`*" and "*`t`*" are the values `SwTrigger` in the first and second cycle, respectively. Similarly, the values "*`P`*" and "*`p`*" are the values `SwPulse` in the first and second cycle, respectively.

Example: The execution of the command

```
COM-HVAMX4ED-Control 6 -a1000 -t
```

produces a positive pulse at the signal `SwTrigger`, while `SwPulse` is kept at 0. This results in a positive pulse at the engine output `SwTrig` as indicated by the command output:

```
Controller configuration set to 0507h->050Fh->0507h
```

```
  software trigger: 0->1->0
  software pulse:   0->0->0
```

Note that the pulse length is defined by the software and may vary with the latency of the used interface.

To produce a precise pulse, the signal `SwPulse` should be used: To produce a positive 1-clock (10 ns) wide pulse at the engine output `SwTrig`, execute the following command:

```
COM-HVAMX4ED-Control 6 -a0000 -a0010 -t
```

The first command sets `SwTrigger` and `SwPulse` to 0, the second one pulses `SwPulse` to 1 and back to 0, this generates the above-mentioned pulse:

```
Controller configuration set to 0507h->0507h->0507h
  software trigger: 0->0->0
  software pulse:   0->0->0
Controller configuration set to 0507h->0517h->0507h
  software trigger: 0->0->0
  software pulse:   0->1->0
```

Note that the command

```
COM-HVAMX4ED-Control 6 -a0010 -t
```

can be abbreviated as follows:

```
COM-HVAMX4ED-Control 6 -a -t
```

Similarly, the command

```
COM-HVAMX4ED-Control 6 -a1110 -t
```

can be shortened as follows:

```
COM-HVAMX4ED-Control 6 -A -t
```

It sets `SwTrigger` to 1 and pulses `SwPulse` to 1 and back to 0, i.e. generates a negative 1-clock (10 ns) wide pulse at the engine output `SwTrig`:

```
Controller configuration set to 0507h->051Fh->050Fh
  software trigger: 0->1->1
  software pulse:   0->1->0
```

The parameters `-w` or `-W` can be used to control and monitor the controller status (see Tab. 9). Note that only the bits 0-7 can be modified,

Tab. 18. Displayed names of the control and status bits.

Status[n]	Value	Displayed name
0	Enb	device enable
1	EnbOsc	oscillator enable
2	EnbPulser	pulser enable
3	SW_Trig	software trigger
4	SW_Pulse	software pulse
5	PreventDis	prevent device disable
6	DisDither	dithering disable
8	Enable	master enable
9	SW_Trig_Out	soft trigger out
10	CLRn	device enabled

the bits 8-15 are read-only. Thus, the value `Config` (see Tab. 17) can have only 8-bit values, i.e. must be an integer between 0 and 255 (FFh).

When the current configuration is reset (see function **`COM_HVAMX4ED_ResetCurrentConfig`** and the command line parameters `-KT` or `-Kt` in Tab. 19), the value `Config` is 0 and the device is disabled. This means that the oscillator and the pulse generators are disabled. To enable them, execute the following command:

```
COM-HVAMX4ED-Control 6 -w7 -t
```

This sets the bits `Enb`, `EnbOsc`, and `EnbPulser` to 1 as shown in the output message:

```
Controller configuration set to 0007h
  device enable:          Y       oscillator
   enable: Y     pulser enable: Y
  prevent device disable: N       dithering
   disable: N
  software trigger:       0       software pulse:
   0
```

For the displayed names of the control bits, refer to Tab. 18.

The query by the command

```
COM-HVAMX4ED-Control 6 -w -t
```

shows the controller state:

```
Controller state: 0507h
        device enable:          Y       oscillator
  enable: Y    pulser enable:    Y
        prevent device disable: N       device
  enabled:    Y    master enable:    Y
        software trigger:       0       software
  pulse:    0    soft trigger out: 0
        dithering disable:      N
```

If no hardware issues such as overheating exist, the bit `Enable` (corresponding text "master enable") is set. This also sets the bit `CLRn` (corresponding text "device enabled"), i.e. the device will be active. This is signalized by a green LED color.

Configuration Management

The device manages several configurations: the current one specifying the present setting of the device and up to 126 user configurations that can be loaded or saved and labeled by a name. All configurations are stored in the non-volatile memory (NVM) of the device. Thus, if the device restarts, the device configuration is reloaded from the NVM so that the state of the device is restored exactly to its previous state immediately before the shutdown.

To manage the configurations, several commands are available (see Tab. 19). It is possible to save and load a specified configuration to or from a data file. The configurations can be copied, deleted, and undeleted; additionally, a list of available configurations can be obtained.

Note that only a small part of the utility commands can be executed by calling a function of the software interface or by a direct command (see section "Software Interface").

Using the parameters `-Ks` or `-Kl`, the current configuration can be saved to or loaded from a text file. The file uses the Windows™ INI file format It begins with a section label `[CurrentConfiguration]` followed by rows in the format of `Entry=Content`, where `Entry` is the name of the partial item and `Content` is its value. For the description of the items, see Tab. 20.

Example: To save the current configuration in the file `Config.ini`, execute the following command:

```
COM-HVAMX4ED-Control 6 -Ks Config.ini -t
```

Note that, if the target file already exists, it will be overwritten without any warning. You may edit and reload it with the following command:

```
COM-HVAMX4ED-Control 6 -Kl Config.ini -t
```

Note that you must not delete any row if you edit the configuration file. It must contain all items, otherwise an error will be reported when you try to load the data from the file.

The loaded configuration may activate the device according to the value of the item `DeviceEnable` (see Tab. 20). However, this value is overridden by the enable state of the device controller that can be set or queried by the parameter `-J`. The device can be enabled only if the enable state is turned on. The enable state can be queried by the following command:

```
COM-HVAMX4ED-Control 6 -J -t
```

It may be turned on by the command

```
COM-HVAMX4ED-Control 6 -Jy -t
```

In the NVM, up to 126 user configurations (see constant `COM_HVAMX4ED_MAX_CONFIG` in declaration file `COM-HVAMX4ED.h`) can be saved, numbered from 1 to 126. Each user configuration can be labeled by a name of up to 52 characters (see constant `COM_HVAMX4ED_CONFIG_NAME_SIZE` in declaration file `COM-HVAMX4ED.h`).

Using the parameters `-KL` or `-KS`, the current configuration can be loaded from or saved to a specified user configuration in the NVM. To save the current configuration to the user configuration with the number 1, execute the following command:

```
COM-HVAMX4ED-Control 6 -KL1 -t
```

The utility displays the name of the configuration beside its number:

Tab. 19. Command line parameters of the program `COM-HVAMX4ED-Control` - Configuration management.

Parameter	Explanation
`-KL`*`n`*	load the current configuration from the user configuration with the number *`n`* (see function **`COM_HVAMX4ED_LoadCurrentConfig`**)
`-KS`*`n`*	save the current configuration to the user configuration with the number *`n`* (see function **`COM_HVAMX4ED_SaveCurrentConfig`**)
`-Kl` *`FileName`*	load the current configuration from a text file with the name *`FileName`*
`-Ks` *`FileName`*	save the current configuration to a text file with the name *`FileName`*
`-Kn`*`N`*	get the name of the user configuration with the number *`N`* (see function **`COM_HVAMX4ED_GetConfigName`**)
`-KN`*`n`* *`Name`*	set the name of the user configuration with the number *`n`* to the text *`Name`* (see function **`COM_HVAMX4ED_SetConfigName`**)
`-Kr`*`n`* *`FileName`*	load the data of the user configuration with the number *`n`* from a text file with the name *`FileName`*
`-Kw`*`n`* *`FileName`*	save the data of the user configuration with the number *`n`* to a text file with the name *`FileName`*
`-KR` *`FileName`*	load the data of all available user configurations from a text file with the name *`FileName`*
`-KW` *`FileName`*	save the data of all available user configurations to a text file with the name *`FileName`*
`-KT`, `-Kt`	reset the current configuration (see function **`COM_HVAMX4ED_ResetCurrentConfig`**)
`-Kc`, `-KC`	list the active user configurations
`-Kd`, `-KD`	list the deleted user configurations

Tab. 19 continued.

Parameter	Explanation
`-KX`*n*	delete the user configuration *n*
`-Kx`*n*	undelete the user configuration *n*
`-Kp`*n*	wipe out the user configuration *n*
`-KA`	delete all user configurations
`-Ka`	undelete all user configurations
`-KP`	wipe out all user configurations
`-J`	show the enable state of the device controller (see function **`COM_HVAMX4ED_GetDeviceEnable`**)
`-Jn` `-JN`	turn off the enable state of the device controller (see function **`COM_HVAMX4ED_SetDeviceEnable`**)
`-Jy` `-JY`	turn on the enable state of the device controller (see function **`COM_HVAMX4ED_SetDeviceEnable`**)

```
Current configuration loaded from NVM set #1
   'Standby'
```

By executing the command

```
COM-HVAMX4ED-Control 6 -KN1 "Configuration Name" -t
```

the user configuration can be labeled with the name "`Configuration Name`".

You may list the available user configuration by executing the following command:

```
COM-HVAMX4ED-Control 6 -Kc -t
```

With the following command, the user configuration with the number 1 can be deleted:

```
COM-HVAMX4ED-Control 6 -KX1 -t
```

You will be asked for confirmation before the specified user configuration is deleted. If the deletion occurs unintentionally, you may restore it by executing the command

```
COM-HVAMX4ED-Control 6 -Kx1 -t
```

To permanently delete the user configuration with the number 1, use the command

```
COM-HVAMX4ED-Control 6 -Kp1 -t
```

It deletes the configuration in the NVM and overwrites its data with zeros. Thus, the data of the configuration is lost and can no longer be restored. The program asks for confirmation if the specified user configuration contains valid data.

Similarly to the current configuration, the user configurations can also be saved to or loaded from a text file. The file uses the same format as the files of the current configuration (see Tab. 20), the only difference is the section label that reads `[Configuration`*N*`]` where *N* = 1-126 is the number of the user configuration.

Example: To save the data of the user configuration with the number 10 in the file `ConfigUser10.ini`, execute the following command:

```
COM-HVAMX4ED-Control 6 -Kw10 ConfigUser10.ini -t
```

To save all user configurations in the file `ConfigAll.ini`, execute the following command:

```
COM-HVAMX4ED-Control 6 -KW ConfigAll.ini -t
```

Note that in this case, the configuration file contains several section labels corresponding to all available user configurations. Note that if the target file already exists, it will be overwritten without any warning here as well.

With the reciprocal command

```
COM-HVAMX4ED-Control 6 -KR ConfigAll.ini -t
```

the previously saved user configurations can be stored in the NVM. Note that the existent user configurations in the NVM will be overwritten without any warning. Thus, it is recommended to save the current content of the NVM to a data file by using the previously mentioned parameter `-KW`.

Tab. 20. Items of configuration files.

Entry	Value
Oscillator	Period0 (see Fig. 2)
Pulser*N* (*N* = 0-3)	Delay*N*, Width*N* (see Fig. 2), SelTrg*N* (see Fig. 2 and Tab. 1), InvTrg*N* ("Pos" or "Neg", see Fig. 2)
Burst*N* (*N* = 0-1)	Burst*N* (see Fig. 2), SelStp*N* (see Fig. 2 and Tab. 1), InvStp*N* ("Pos" or "Neg", see Fig. 2)
DIO*N* (*N* = 0-6)	OutEnb*N* + TermEnb*N* ("In", "Term", or "Out", see Tab. 3), SelDout*N* (see Fig. 4 and Tab. 2), InvDout*N* ("Pos" or "Neg", see Fig. 4)
SwitchTrigger*N* (*N* = 0-3)	SelSwTrg*N* (see Fig. 5, Fig. 6, and Tab. 7), InvSwTrg*N* ("Pos" or "Neg", see Fig. 5 and Fig. 6 TriggerRiseDelay, TriggerFallDelay, see Fig. 5 and Fig. 8
SwitchEnable*N* (*N* = 0-3)	SelSwEnb*N* (see Fig. 5 and Tab. 7), InvSwEnb*N* ("Pos" or "Neg", see Fig. 5 EnableDelay, see Fig. 5 and Fig. 8
MappingTrigger	MapTrgEnb (see Fig. 5 and Fig. 7), MappingTriggerState*N* (*N* = 0-4, see Fig. 5, Fig. 7 and Tab. 6)
MappingEnable	MapEnbEnb (see Fig. 5 and Fig. 7), MappingEnableState*N* (*N* = 0-4, see Fig. 5, Fig. 7 and Tab. 6)
DeviceEnable	Bit Enb in Tab. 9 and Fig. 9 ("Y"/"N")
OscillatorEnable	Bit EnbOsc in Tab. 9 ("Y"/"N")
PulserEnable	Bit EnbPulser in Tab. 9 ("Y"/"N")
SoftwareTrigger	Bit SW_Trig in Tab. 9 and Fig. 3 ("Y"/"N")
SoftwarePulse	Bit SW_Pulse in Tab. 9 and Fig. 3 ("Y"/"N")
PreventDeviceDisable	Bit PreventDis in Tab. 9 and Fig. 9 ("Y"/"N")
DisableDithering	Bit DisDither in Tab. 9 ("Y"/"N")

Tab. 21. Command line parameters of the program `COM-HVAMX4ED-Control` - Backup and restore.

Parameter	Explanation
`-y` *`FileName`*	backup the NVM data into a text file with the name *`FileName`*
`-Y` *`FileName`*	restore the NVM data from a text file with the name *`FileName`*

Backing Up and Restoring the Data

All data stored in the device's NVM, i.e. the device settings including the current configuration as well as the user configurations can be backed up or restored (see Tab. 21). A restore procedure rolls back the device exactly to the state it was in at the time of the backup. Thus, if any adjusting of the settings is planned, it is recommended to first create a backup with which the original state can be restored.

Since the back up and restore procedures transfer a large amount of data, it is recommended to increase the communication speed by specifying the parameter `-$` (see section "General Control"). To back up the system memory into a data file `Memory.txt` at a communication speed of 230.4 kbaud, execute the following command:

```
COM-HVAMX4ED-Control 6 -$230400 -y MemoryData.txt
   -t
```

This command downloads the memory data from the device into the file `Memory.txt`.

To restore the data at the same communication speed, execute the following command:

```
COM-HVAMX4ED-Control 6 -$230400 -Y MemoryData.txt
   -t
```

This command uploads the memory data from the file `Memory.txt` to the device. Since all data in the NVM will be overwritten, the utility asks for a confirmation twice.

Utility FlashLoader

FlashLoader is a simple Windows™ program running in text mode. It enables you to upgrade the firmware of the pulse controller. You should perform an upgrade when you have received or downloaded a new firmware file from the device manufacturer. Launching the utility `FlashLoader.exe` without any parameters displays a simple help text with the expected syntax of the command line.

FlashLoader is a universal utility for many different devices. The pulse controller uses a variable data rate for communication, thus a utility with the revision number 1-20 or later must be used and it must be launched with the command line parameter `-$`.

Before upgrading the firmware, you should first test the device and the communication by verifying the current firmware version. To do so, execute the following command:

```
FlashLoader 6 Firmware.txt -$ -v
```

where `Firmware.txt` is the file containing the current firmware and the number 6 indicates the port COM6 to which the device is connected. The program should produce the following output:

```
Code file Firmware.txt from 06/18/2021, 12:00:00
Flash Loader 2.00
Verifying code file Firmware.txt
Verification completed on Mon, 06/21/2021, 12:34:56
23293 (5AFDh) bytes processed, 24064 (5E00h) bytes
verified
Resetting the target
Program completed successfully
```

For the verifying procedure, a flash-loader utility on the device is activated. When the verification is completed without any errors, the device is restarted.

! **Attention:** To ensure that the device cannot activate the attached switches or other peripherals and produce erratic signals while FlashLoader is active, disconnect the power cables of the switches or turn off the power supply units providing the supply voltages for the switches.

If any error occurs, do not proceed with the firmware upgrade. If you cannot resolve the issues, contact the manufacturer. Note that even if the verification fails and the flash loader on the device remains active, it is safe to power the device off to restart it. However, a safer and more comfortable alternative is to execute the following command:

```
FlashLoader 6 -$ -i -f
```

This prevents the utility on the host computer from initializing the flash loader utility on the microcontroller and sends the reset command to the device.

! **Attention:** If the flash loader on the device is still active and you do not specify the command line parameter `-i`, the programming utility sends data to the device at a wrong data rate during initialization. This data stream cannot be received properly, the device ends up in an undefined state and does not respond anymore. It must be powered off and on to restart the controller. Since there is no other way to stop the flash loader on the device and resume normal operation, be sure to exactly follow the instructions in this section.

If the verification has succeeded, you may start the firmware upgrade by entering the command:

```
FlashLoader 6 Firmware.txt -$
```

The parameter `Firmware.txt` is the file with the new firmware. The program should produce the following output:

```
Code file Firmware.txt from 06/18/2021, 12:00:00
Flash Loader 2.00
Programming code file Firmware.txt
Programming completed on Mon, 06/21/2021, 12:34:56
23293 (5AFDh) bytes processed, 24064 (5E00h) bytes
programmed
Resetting the target
Program completed successfully
```

For the programming procedure, a flash loader utility on the device is activated as well. When the programming is completed, the device is restarted with the new firmware. You can recognize this by the startup sequence of the LED on the front panel of the device.

If an error occurs, the flash loader utility on the microcontroller may remain active. This is the case if the device did not restart. In this case, you should reattempt the upgrade with the command line parameter `-i`:

```
FlashLoader 6 Firmware.txt -$ -i
```

This will prevent the utility on the host computer from initializing the flash loader utility on the microcontroller and it will only try to reprogram the file `Firmware.txt`. If the error persists, contact the manufacturer.

! **Attention:** You must not power down the device if the firmware upgrade has not succeeded. Otherwise, the device will not operate properly or it might not even restart at all. If this were to happen, it would be necessary to reprogram the device in the factory.

Driver Installation

Installation of the Virtual Port for the USB Interface

The virtual port driver is required for the operation of the device with a USB interface. If your operating system is Windows™, please note the following:

- Please use the update function of the operating system at the host computer or download the most recent driver from the homepage of the manufacturer of the USB adapter. The drivers are located at the following address: http://www.ftdichip.com/Drivers/VCP.htm. Please choose the correct driver version according to your operating system.
- To install the driver, administrative rights are required.
- The installation is described in detail in the "Installation Guides" available at the abovementioned address. Please read this description carefully before starting the installation.
- After the installation, the number of the virtual port can be set. You can change the settings in the device manager by opening the settings of the device *USB Serial Port (COMx)*. To modify the settings, administrative rights are required. The settings are applied immediately, you do not need to reboot the PC to activate them.

The software can also be used on computers running Linux. You can run it using the Windows™ emulator Wine (see http://www.winehq.org/).

Starting with Linux Kernel 3.0.0-19, all FTDI devices are already supported without the need to compile any additional kernel modules. For more details, consult the homepage of the manufacturer of the USB adapter: http://www.ftdichip.com/Drivers/VCP.htm.

The system has to be configured in the following way:

- Use a program such as 'dmesg' to find out which USB port the device is connected to: Look for a line similar to "FTDI USB Serial Device converter now attached to ttyUSB0"
- Link the Linux device to the virtual COM port of wine: `ln -s /dev/ttyUSB0 ~ /.wine/dosdevices/com6` This assumes that the device is attached to ttyUSB0 and will be linked with COM6.

Software Interface

The software interface for the device consists of a 32- or 64-bit dynamic link library `COM-HVAMX4ED.dll`. Both versions are located in the directory "Program" of the enclosed software package. The software interface is a stand-alone software package, it does not require any additional library or driver, except for the virtual port driver (see section "Driver Installation").

The user functions in the dynamic link library `COM-HVAMX4ED.dll` can be called from any conventional programming language. For the details, please consult the user manual of your compiler. The definition of the interface functions is located in the declaration file `COM-HVAMX4ED.h` written in C/C++. If your compiler cannot create an import library from the dynamic link library `COM-HVAMX4ED.dll`, please link the library `COM-HVAMX4ED.lib` instead of the dynamic link library to your project.

Functionality of the Software Interface

The software interface typically controls up to 16 communication channels for the data transfer to and from the device (see the constant `COM_HVAMX4ED_MAX_PORT` in the declaration file `COM-HVAMX4ED.h`). This means it can control up to 16 devices at a time.

Before utilizing any function from the dynamic link library `COM-HVAMX4ED.dll`, the software should check the version of the library by calling the function **`COM_HVAMX4ED_GetSWVersion`**. Note that a library with a different version number can contain different implementations of the functions and calling them may lead to unpredictable results; in most cases, the software will crash.

Each communication channel must be opened before starting the communication. The opening procedure (function **`COM_HVAMX4ED_Open`**) configures the used physical (RS-232) or virtual (USB) serial port and clears the port buffers.

The communication channel should be closed at the end of the program. If this does not happen, the software interface automatically does it for you when the dynamic link library `COM-HVAMX4ED.dll` is unloaded from the system memory.

The communication speed of the serial port defaults to 9600 baud but can be changed to any value up to 230.4 kbaud (see function **`COM_HVAMX4ED_SetBaudRate`**). Please note that the highest speed can be used with any USB connection.

When using the RS-232 interface, check the host hardware for the maximum available communication speed. Most simple serial ports in PCs only support communication speeds up to 115.2 or 128 kbaud. They may also lose data characters when receiving the data from the device. If you encounter such issues, replace the serial-port hardware by a high-speed adapter card with a dedicated system driver. USB adapters providing one or more serial ports usually do not cause any issues either. However, due to the latency of the USB protocol, they are significantly slower than serial ports placed directly on a PC main-board or than serial-port cards.

If a real-time control of the hardware with a temporal resolution of 1 ms or better should be implemented, a high-speed adapter card with a serial port (RS-232) is the recommended solution. When run at the communication speed of 230.4 kbaud, a typical command consisting of 2-3 ASCII characters can be sent within only 0.1-0.2 ms. In contrast to that, a typical USB transmission may take up to 50 ms. This means that, usually, only less than 10 bidirectional transfers per second can be achieved when USB is used.

The communication uses the handshake lines of the serial port. Thus, when using the RS-232 interface, be sure that you use a connection cable that connects all pins in the DE-09 (D-Sub-9) connector. The device emulates a null-modem, i.e. the line DTR is replicated by the hardware as DSR and DCD and the line RTS is replicated by the firmware as the signal CTS. If time-consuming tasks like bulk operations with the NVM are performed, the device deasserts the signal CTS and indicates that it is not ready to respond. If the host deasserts the lines DTR and/or RTS, the input communication buffer of the device is cleared. This can be used to repair the communication if the response becomes erratic (see function **`COM_HVAMX4ED_Purge`**). Moreover, if the lines DTR and/or RTS are deasserted for more than 100 ms, the communication interface is reset and the communication speed is set to the default value of 9600 baud. This happens automatically when the software stops using the serial port, thus the communication with the device always starts at the default speed of 9600 baud.

As a first parameter, most of the interface functions require the variable `PortNumber`, which is the number of the communication channel that should be used for the operation in question. This number is an unsigned integer that must be lower than `COM_HVAMX4ED_MAX_PORT`, i.e. it must be in the range from 0 to 15.

The return value of most functions contains a number indicating the success of the operation. The return value is a signed 32-bit number (`int`). The last return value can be reloaded by the function **`COM_HVAMX4ED_GetInterfaceState`**. Table 22 summarizes the possible return values together with the error messages, which can also be obtained by the function **`COM_HVAMX4ED_GetErrorMessage`**. If a data transfer failure has occurred, you can find the reason by calling the functions **`COM_HVAMX4ED_GetIOState`** and **`COM_HVAMX4ED_GetIOErrorMessage`**. The former returns the last I/O error, the latter the corresponding error message (see Tab. 23).

If you encounter any problems with the error messages or with establishing the communication, please contact the manufacturer of the device.

Direct Command Control

As an alternative, direct control by a terminal program or another software is possible. When using self-written software instead of the supplied dynamic link library `COM-HVAMX4ED.dll`, the programmer must make sure to collect the data response after issuing a command and to handle the possible communication errors.

The hardware communication protocol uses 8 data bits, 2 stop bits, and even parity. The commands use ASCII characters only. Each data transfer consists of a command character, optional control data, and a termination character.

The termination character is the Carriage-Return code (CR, 0D hexadecimal; in the following text, it is referred to as the symbol ↵). The termination character initiates the command execution in the device, which then responses with ASCII characters that are similar to the command.

Most parameters are transferred as a sequence of hexadecimal digits in uppercase; the most significant digit is transferred first. In several commands, Boolean parameters are used. They use the uppercase characters 'Y' for true and 'N' for false. An ASCII character string is transferred as is but without the null-termination. Instead, the termination character of the command indicates the end of the string.

If a transmitted command is misspelled or unknown or if it contains invalid parameters, the device will not process it. The command and its data are cleared from the input buffer and the device does not provide any response. This implies that the communication control should specify a certain timeout and if no response is received within this timeout, the command has to be considered erroneous. The timeout depends on the interface latency; with USB, the recommended value is 100 ms. With the RS-232 interface, significantly shorter timeouts can be used. However, the proper timeout value may also be influenced by the operating system of the host and the serial-port hardware. Thus, it should be determined experimentally. If the error detection is not time-critical in the application in question, the value recommended for the USB interface (i.e. 100 ms) can also be used for the RS-232 interface.

If the device does not respond or if the response is invalid, use the handshake lines of the serial port to clear the input device buffer (see section "Functionality of the Software Interface" and function **COM_HVAMX4ED_Purge**) before reattempting the operation. The clearing procedure should start with deasserting the lines DTR and/or RTS and waiting for an inactive signal CTS which indicates that the device is not ready and that it has recognized the clear command. Then, the abovementioned handshake lines should be reasserted and the device should respond with activating the signal CTS to indicate that it is ready to receive further commands. Please ensure that the software controlling the communication does not start to send any data to the device while the signal CTS is still inactive, i.e. when the device is not yet ready. The device would either not receive the data at all or the first characters of the sent sequence may be lost.

If you use a communication speed different from the default value of 9600 baud, be sure to deassert the lines DTR and/or RTS for a short time only. It is recommended to assert the handshake lines immediately after the device has responded with an inactive signal CTS. If the handshake lines were deasserted for a longer time period (see section "Functionality of the Software Interface"), the communication speed would be set to the default value and it would need to be adjusted again by the function **COM_HVAMX4ED_SetBaudRate**.

The direct control is summarized in the following sections by the items "Command" and "Response" below the declaration of a particular function. Note that several functions like **`COM_HVAMX4ED_Open`** or **`COM_HVAMX4ED_Close`** do not send any data to the device and thus do not have any equivalent direct control commands.

Error Codes

Tab. 22. Return values of the interface functions

<table>
<tr><th>Return value</th><th>Error message</th><th>Description</th></tr>
<tr><td>0</td><td>No error</td><td>The data transfer was completed successfully.</td></tr>
<tr><td>-1</td><td>PortNumber out of range</td><td>The parameter PortNumber specified when calling the function is out of range.</td></tr>
<tr><td>-2</td><td>Error opening the port</td><td>The port could not be opened. For possible reasons, see Tab. 23.</td></tr>
<tr><td>-3</td><td>Error closing the port</td><td>The port could not be closed. For possible reasons, see Tab. 23.</td></tr>
<tr><td>-4</td><td>Error purging the port</td><td>The port buffers could not be cleared.</td></tr>
<tr><td>-5</td><td>Error setting the port control lines</td><td>The port control lines could not be set.</td></tr>
<tr><td>-6</td><td>Error reading the port status lines</td><td>The port status lines could not be read.</td></tr>
<tr><td>-7</td><td>Error sending command</td><td rowspan="3">The data transfer to the device failed. For possible reasons, see Tab. 23.</td></tr>
<tr><td>-8</td><td>Error sending data</td></tr>
<tr><td>-9</td><td>Error sending termination character</td></tr>
<tr><td>-10</td><td>Error receiving command</td><td rowspan="3">The data transfer from the device failed. For possible reasons, see Tab. 23.</td></tr>
<tr><td>-11</td><td>Error receiving data</td></tr>
<tr><td>-12</td><td>Error receiving termination character</td></tr>
<tr><td>-13</td><td>Wrong command received</td><td rowspan="2">The device sent an unexpected response.</td></tr>
<tr><td>-14</td><td>Wrong argument received</td></tr>
<tr><td>-15</td><td>Wrong argument passed to the function</td><td>One of the arguments passed to the function was out of the allowable range.</td></tr>
</table>

Return value	Error message	Description
-16	Error setting the baud rate	The data rate could not be set. Check if the host is able to communicate at the desired speed.
-100	Device not connected	The port status lines indicate that the device is not connected.
-101	Device not ready	The port status lines indicate that the device is not ready. Communication with the device is only possible if it is not executing any process.
-102	Device state could not be set to not ready	The device did not react properly. Try to reset the communication or restart the device by powering it off and on.
-400	Error opening the file for debugging output	The file for the debugging output cannot be opened for writing. Check if you have permission to perform this action or if the file already exists and is in use by another application.
-401	Error closing the file for debugging output	The file for the debugging output cannot be closed. Check if access to the file is still possible.

Tab. 23. I/O errors

Return value	Error message	Description
0	No error	The data transfer was completed successfully.
1	Port has not been opened yet	You attempted to use the communication channel before having opened it.
2	Cannot open the port	The specified port could not be opened. Either the port does not exist or it is currently being used by another program.
3	Cannot obtain communication error	The system could not get the last communication error of the port.
4	Cannot get the state of the port	The system could not get the state of the port.
5	Cannot set the state of the port	The system could not set the state of the port.
6	Break timing error	The system could not issue a break with the proper timing.
7	Cannot place the transmission line in a break state	The system could not issue a break. Check the data sheet of the used communication port.
8	Cannot place the transmission line in a nonbreak state	The system could not clear the break state. Check the data sheet of the used communication port.
9	Cannot set the timeouts for the port	The system could not set the timeouts for the port.
10	Cannot clear the port	The system could not clear the port buffers.
11	Error reading data from the port	The system could not read data from the port. Most probably, no data is available because the device is either disconnected or does not respond.
12	Error writing data to the port	The system could not write data to the port.

Return value	Error message	Description
13	Wrong data amount written to the port	The system could not write the proper amount of data to the port.
14	Error setting the control lines of the port	The system could not set the state of the port control lines.
15	Error reading the status lines of the port	The system could not get the state of the port status lines.
16	Device is busy	The system could not access the device since a background process is active. Wait until it finishes.

Communication Control

Function COM_HVAMX4ED_Open

```
int COM_HVAMX4ED_Open (WORD PortNumber,
  WORD COMNumber);
```

Opens the communication channel and returns an error code according to Tab. 22.

The function opens the channel with the number `PortNumber` and attaches it to the serial port with the number `COMNumber`. The variable `PortNumber` must be lower than the number of implemented channels `COM_HVAMX4ED_MAX_PORT` (see the declaration file `COM-HVAMX4ED.h`).

The serial port number `COMNumber` must point to a valid serial port to which the controller is attached. Note that this number is the number of the COM port, i.e. `COMNumber = 1` points to the port COM1. The function accepts all numbers that are supported by the operating system, i.e. any port between COM1 and COM255 can be used for the communication.

You must call the function **`COM_HVAMX4ED_Open`** prior to any other communication function. If the function returns an error, the communication channel remains closed and no data communication is possible.

Function COM_HVAMX4ED_Close

```
int COM_HVAMX4ED_Close (WORD PortNumber);
```

Closes the communication channel and returns an error code according to Tab. 22.

You can use this function to free the used port for another application.

If an application that has exclusively used the software interface `COM-HVAMX4ED.dll` finishes, the opened communication channel closes automatically. Thus, the programmer does not need to call the function **`COM_HVAMX4ED_Close`** explicitly.

Function COM_HVAMX4ED_SetBaudRate

```
int COM_HVAMX4ED_SetBaudRate
  (WORD PortNumber, unsigned & BaudRate);
```

Command: *$BBBBB*↵

Response: *$BBBBB*↵

Sets the communication speed to the baud rate given by the variable `BaudRate`, modifies it to the set value (see below) and returns an error code according to Tab. 22.

The device is optimized for communication speed of 9600 baud and its multiples, i.e., for instance, 115.2 or 230.4 kbaud (see function **COM_HVAMX4ED_GetCPUData**). If the communication speed is set to another value such as 128 kbaud, the device will use a slightly different speed. This value is returned in the variable `BaudRate`. Note that if the difference between the communication speed set in the host and in the device is less than about 5%, the communication can still be established without any errors.

If communicating with the device is no longer possible, call the function **COM_HVAMX4ED_Purge** in order to reset the communication speed to the default value of 9600 baud. You can also interrupt the communication, this deactivates the handshake lines and resets communication speed as well.

The direct command (*$BBBBB*↵) contains 5 hexadecimal digits (*BBBBB*) for the variable `BaudRate`. The response has the same format; if the command is successful, it returns the value of the baud rate set by the device. The response is still transmitted at the old data rate. After it has been received, the data rate must be set to the new value in the host UART to enable communication with the device.

Function COM_HVAMX4ED_Purge

```
int COM_HVAMX4ED_Purge (WORD PortNumber);
```

Clears the port data buffers and returns an error code according to Tab. 22.

This function can be used to repair a disturbed communication. In case of a user program crash, this function should be called to erase data incorrectly received from the device. It also deactivates the handshake lines to clear the communication buffer of the device and to reset the communication speed to the default value of 9600 baud.

Note that the function **COM_HVAMX4ED_Purge** is automatically called by the function **COM_HVAMX4ED_Open** to establish a clean communication start independent of the previous data transfers.

Function COM_HVAMX4ED_GetBufferState

```
int COM_HVAMX4ED_GetBufferState
    (WORD PortNumber, BOOL & Empty);
```

Command: Z↵

Response: Z*E*↵

Saves the state of the device's input data buffer in the variable `Empty` and returns an error code according to Tab. 22.

When a large amount of data should be transferred to the device, this function can be used to ensure that the input data buffer contains enough free space. If the return value of the variable `Empty` is false, the input buffer is not empty and there is no guarantee that the device will be able to receive the data. This situation can occur if the device has just received a large amount of data and has not yet finished processing it. In such a case, the call to the function **COM_HVAMX4ED_GetBufferState** should be repeated after several milliseconds until the return value becomes `true`.

The response to the direct command (`Z↵`) contains one Boolean character (*E*) for the variable `Empty`.

Function COM_HVAMX4ED_DevicePurge

```
int COM_HVAMX4ED_DevicePurge
    (WORD PortNumber, BOOL & Empty);
```

Command: z↵

Response: z*E*↵

Clears the device's output data buffer and saves the state of the device's input data buffer in the variable `Empty` like the function **COM_HVAMX4ED_GetBufferState**. The return value is an error code according to Tab. 22.

This function can be used to repair a disturbed communication. If the device does not respond properly, the function **COM_HVAMX4ED_DevicePurge** should be called repeatedly until it returns the value `true` in the variable `Empty`.

The response to the direct command (`z`⏎) contains one Boolean character (*E*) for the variable `Empty`.

Device Control

Function COM_HVAMX4ED_GetMainState

```
int COM_HVAMX4ED_GetMainState
  (WORD PortNumber, WORD & State);
```

Command: `M↵`

Response: `M`*`SSSS`*`↵`

Saves the main device status in the variable `State` and returns an error code according to Tab. 22.

The possible values of the variable `State` are given by the constants `COM_HVAMX4ED_XXX` (see the declaration file `COM-HVAMX4ED.h`). If the device is working properly, the variable `State` returns the value `COM_HVAMX4ED_STATE_ON`. Values higher or equal to `COM_HVAMX4ED_STATE_ERROR` indicate an error. Note that the detected errors are also indicated by the LED on the front panel.

The response to the direct command (`M↵`) contains 4 hexadecimal digits (*`SSSS`*) for the variable `State`.

Function COM_HVAMX4ED_GetDeviceState

```
int COM_HVAMX4ED_GetDeviceState
  (WORD PortNumber, DWORD & DeviceState);
```

Command: `S↵`

Response: `S`*`DDDDDDDD`*`↵`

Saves the detailed state of the device in the variable `DeviceState` and returns an error code according to Tab. 22.

The variable `DeviceState` is a bit combination of the constants `COM_HVAMX4ED_DEVST_XXX` (see the declaration file `COM-HVAMX4ED.h`). If the device is working properly, the variable `DeviceState` returns the value `COM_HVAMX4ED_DEVST_OK` (i.e. zero), nonzero values indicate an error. The errors detected by the firmware set the state of the device (see function **`COM_HVAMX4ED_GetMainState`**).

The response to the direct command (`S↵`) contains 8 hexadecimal digits (*`DDDDDDDD`*) for the variable `DeviceState`.

Function COM_HVAMX4ED_GetHousekeeping

```
int COM_HVAMX4ED_GetHousekeeping
   (WORD PortNumber, double & Volt12V,
   double & Volt5V0, double & Volt3V3,
   double & TempCPU);
```

Command: `H↵`

Response: `H`*`FFFFDDDDCCCCTTTT`*`↵`

Saves the housekeeping data in the variables `Volt12V`, `Volt5V0`, `Volt3V3`, and `TempCPU`, and returns an error code according to Tab. 22.

The return values in the variables `Volt12V`, `Volt5V0`, and `Volt3V3` are supply voltages in V, their nominal values are 12 V, 5.0 V, and 3.3 V. The return value in the variable `TempCPU` is the temperature of the CPU in ºC.

The response to the direct command (`H↵`) contains 3 x 4 hexadecimal digits for the variables `Volt12V` (*`FFFF`*), `Volt5V0` (*`DDDD`*), and `Volt3V3` (*`CCCC`*), followed by 4 hexadecimal digits (*`TTTT`*) for the variable `TempCPU`. The voltage values are in mV, i.e. in order to scale them to the abovementioned values, they have to be divided by 1000. The temperature variable `TempCPU` is received in units of 10 mK, i.e. it has to be corrected by subtracting the offset of 27315 and dividing by 100 to get the value in ºC.

Function COM_HVAMX4ED_GetSensorData

```
int COM_HVAMX4ED_GetSensorData
   (WORD PortNumber, double
   Temperature [COM_HVAMX4ED_SEN_COUNT]);
```

Command: `T↵`

Response: `T`*`AAAA`*`..`*`ZZZZ`*`↵`

Saves the temperature-sensor data in the array `Temperature` and returns an error code according to Tab. 22.

The return values in the array `Temperature` are temperatures in ºC measured by the temperature sensors.

The size of the array `Temperature` must be sufficient for storing data from all sensors, i.e. it must be equal to the constant `COM_HVAMX4ED_SEN_COUNT` (see the declaration file `COM-HVAMX4ED.h`).

The response to the direct command (`T`⏎) contains 4 hexadecimal digits for each sensor temperature (*AAAA*`..`*ZZZZ*). For the temperature units, see the function **`COM_HVAMX4ED_GetHousekeeping`**.

Function COM_HVAMX4ED_GetFanData

```
int COM_HVAMX4ED_GetFanData (WORD PortNumber,
  BOOL Enabled [COM_HVAMX4ED_FAN_COUNT],
  BOOL Failed [COM_HVAMX4ED_FAN_COUNT],
  WORD SetRPM [COM_HVAMX4ED_FAN_COUNT],
  WORD MeasuredRPM [COM_HVAMX4ED_FAN_COUNT],
  WORD PWM [COM_HVAMX4ED_FAN_COUNT]);
```

Command: B⏎

Response: B*EFSSSSMMMMPPPP*.. *EFSSSSMMMMPPPP*⏎

Saves the fan data in the arrays `Enabled`, `Failed`, `SetRPM`, `MeasuredRPM`, and `PWM`, and returns an error code according to Tab. 22.

The return values in the arrays `Enabled` and `Failed` are Boolean values showing whether the respective fan is enabled or has failed. The fans are enabled according to the device setting stored in the CPU ROM, these values cannot be changed by the user. A fan has failed if it should rotate but does not start to spin within a predefined time. Fans that are disabled or that should not spin cannot be detected as failed.

The return values `SetRPM` and `MeasuredRPM` contain the set and the measured rotational speed of the respective fan in RPM (Revolutions Per Minute). In an ideal case, both values should be equal; under real conditions, the measured speed fluctuates around the set value.

The return value `PWM` is the duty cycle of the fan control signal that determines the rotational speed of the respective fan. It can vary between 0 (= 0%) when the fan is stopped and 1 (= 100%) when the fan should run at its maximum speed.

The size of all array variables must be sufficient for storing data from all fans, i.e. it must be equal to the constant `COM_HVAMX4ED_FAN_COUNT` (see the declaration file `COM-HVAMX4ED.h`).

Note that in devices without fans or with fewer fans than the constant `COM_HVAMX4ED_FAN_COUNT`, the data of fans that are not installed does not have any influence on the function of the device.

For each fan, the response to the direct command (`B`⏎) contains one Boolean character for the variables `Enabled` (*E*) and `Failed` (*F*), respectively, 4 hexadecimal digits (*SSSS*) for the variable `SetRPM`, 4 hexadecimal digits (*MMMM*) for the variable `MeasuredRPM`, and 4 hexadecimal digits (*PPPP*) for the variable `PWM`.

Function COM_HVAMX4ED_GetLEDData

```
int COM_HVAMX4ED_GetLEDData (WORD PortNumber,
  BOOL & Red, BOOL & Green, BOOL & Blue);
```

Command: L⏎

Response: L*RGB*⏎

Saves the colors of the LED on the front panel in the variables `Red`, `Green`, and `Blue`, and returns an error code according to Tab. 22.

The colors in the variables `Red`, `Green`, and `Blue` are Boolean values, i.e. they indicate if each color is turned on or off.

The response to the direct command (`L`⏎) contains three Boolean characters for each color value in the variables `Red` (*R*), `Green` (*G*), and `Blue` (*B*).

Pulse Generator Management

Function COM_HVAMX4ED_GetOscillatorPeriod

```
int COM_HVAMX4ED_GetOscillatorPeriod
   (WORD PortNumber, DWORD & Period);
```

Command: s↵

Response: s*PPPPPPPP*↵

Saves the oscillator period in the variable `Period` and returns an error code according to Tab. 22.

To obtain the oscillator period in seconds, the constants `COM_HVAMX4ED_CLOCK` and `COM_HVAMX4ED_OSC_OFFSET` can be used (see the declaration file `COM-HVAMX4ED.h`). The former defines the clock frequency of the generator system (typically 100 MHz), the latter the offset of the period value due to the hardware. For more details, see the description of the number Period0 in Fig. 2 and in section "Pulse Controller".

The response to the direct command (`s↵`) contains 8 hexadecimal digits (*PPPPPPPP*) for the variable `Period`.

Function COM_HVAMX4ED_SetOscillatorPeriod

```
int COM_HVAMX4ED_SetOscillatorPeriod
   (WORD PortNumber, DWORD Period);
```

Command: s*PPPPPPPP*↵

Response: s*PPPPPPPP*↵

Sets the oscillator period to the value given by the variable `Period` and returns an error code according to Tab. 22.

For more details, see function **`COM_HVAMX4ED_GetOscillatorPeriod`**.

The direct command (`s`*PPPPPPPP*↵) contains 8 hexadecimal digits (*PPPPPPPP*) for the variable `Period`. If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetPulserDelay

```
int COM_HVAMX4ED_GetPulserDelay
  (WORD PortNumber, unsigned PulserNo,
  DWORD & Delay);
```

Command: **d*N*⏎**

Response: **d*NDDDDDDDD*⏎**

Saves the pulse delay of the pulse generator with the number `PulserNo` in the variable `Delay` and returns an error code according to Tab. 22.

The pulse generators are numbered from 0 to `COM_HVAMX4ED_PULSER_NUM-1` (see the declaration file `COM-HVAMX4ED.h`).

To obtain the pulse delay in seconds, the constants `COM_HVAMX4ED_CLOCK` (see function **`COM_HVAMX4ED_GetOscillatorPeriod`**) and `COM_HVAMX4ED_PULSER_DELAY_OFFSET` (see the declaration file `COM-HVAMX4ED.h`) can be used. The latter defines the offset of the delay value due to the hardware. For more details, see the description of the numbers Delay0-3 in Fig. 2 and in section "Pulse Controller".

The direct command (d*N*⏎) contains one hexadecimal digit (*N*) for the variable `PulserNo`. If the command is successful, the device response (d*NDDDDDDDD*⏎) has the same beginning as the command and additionally returns 8 hexadecimal digits (*DDDDDDDD*) for the variable `Delay`.

Function COM_HVAMX4ED_SetPulserDelay

```
int COM_HVAMX4ED_SetPulserDelay
  (WORD PortNumber, unsigned PulserNo,
  DWORD Delay);
```

Command: **d*NDDDDDDDD*⏎**

Response: **d*NDDDDDDDD*⏎**

Sets the pulse delay of the pulse generator with the number `PulserNo` to the value given by the variable `Delay` and returns an error code according to Tab. 22.

For more details, see function **COM_HVAMX4ED_GetPulserDelay**.

The direct command (d*NDDDDDDDD*⏎) contains one hexadecimal digit (*N*) for the variable PulserNo and 8 hexadecimal digits (*DDDDDDDD*) for the variable Delay. If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetPulserWidth

```
int COM_HVAMX4ED_GetPulserWidth
  (WORD PortNumber, unsigned PulserNo,
  DWORD & Width);
```

Command: w*N*⏎

Response: w*NWWWWWWWW*⏎

Saves the pulse width of the pulse generator with the number PulserNo in the variable Width and returns an error code according to Tab. 22.

For the number of the pulse generator in the variable PulserNo, see function **COM_HVAMX4ED_GetPulserDelay**.

To obtain the pulse width in seconds, the constants COM_HVAMX4ED_CLOCK (see function **COM_HVAMX4ED_GetOscillatorPeriod**) and COM_HVAMX4ED_PULSER_WIDTH_OFFSET (see the declaration file COM-HVAMX4ED.h) can be used. The latter defines the offset of the width value due to the hardware. For more details, see the description of the numbers Width0-3 in Fig. 2 and in section "Pulse Controller".

The direct command (w*N*⏎) contains one hexadecimal digit (*N*) for the variable PulserNo. If the command is successful, the device response (w*NWWWWWWWW*⏎) has the same beginning as the command and additionally returns 8 hexadecimal digits (*WWWWWWWW*) for the variable Width.

Function COM_HVAMX4ED_SetPulserWidth

```
int COM_HVAMX4ED_SetPulserWidth
  (WORD PortNumber, unsigned PulserNo,
  DWORD Width);
```

Command: **w*NDDDDDDDD*↵**

Response: **w*NDDDDDDDD*↵**

Sets the pulse width of the pulse generator with the number `PulserNo` to the value given by the variable `Width` and returns an error code according to Tab. 22.

For more details, see function **COM_HVAMX4ED_GetPulserWidth**.

The direct command (w*NWWWWWWWW*↵) contains one hexadecimal digit (*N*) for the variable `PulserNo` and 8 hexadecimal digits (*WWWWWWWW*) for the variable `Width`. If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetPulserBurst

```
int COM_HVAMX4ED_GetPulserBurst
  (WORD PortNumber, unsigned PulserNo,
  DWORD & Burst);
```

Command: **b*N*↵**

Response: **b*NBBBBBB*↵**

Saves the burst size of the pulse generator with the number `PulserNo` in the variable `Burst` and returns an error code according to Tab. 22.

The pulse generators with the burst functionality are numbered from 0 to `COM_HVAMX4ED_PULSER_BURST_NUM-1` (see the declaration file `COM-HVAMX4ED.h`).

The burst size, i.e. the value in the variable `Burst`, can be any unsigned 24-bit number, i.e. any value between 0 and `COM_HVAMX4ED_MAX_BURST-1` (see the declaration file `COM-HVAMX4ED.h`).

The direct command (`b`*N*⏎) contains one hexadecimal digit (*N*) for the variable `PulserNo`. If the command is successful, the device response (`b`*NBBBBBB*⏎) has the same beginning as the command and additionally returns 6 hexadecimal digits (*BBBBBB*) for the variable `Burst`.

Function COM_HVAMX4ED_SetPulserBurst

```
int COM_HVAMX4ED_SetPulserBurst
   (WORD PortNumber, unsigned PulserNo,
   DWORD Burst);
```

Command: b*NBBBBBB*⏎

Response: b*NBBBBBB*⏎

Sets the burst size of the pulse generator with the number `PulserNo` to the value given by the variable `Burst` and returns an error code according to Tab. 22.

For more details, see function **`COM_HVAMX4ED_GetPulserBurst`**.

The direct command (`b`*NBBBBBB*⏎) contains one hexadecimal digit (*N*) for the variable `PulserNo` and 6 hexadecimal digits (*BBBBBB*) for the variable `Burst`. If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetPulserConfig

```
int COM_HVAMX4ED_GetPulserConfig
   (WORD PortNumber, unsigned PulserNo,
   BYTE & Config);
```

Command: p*N*⏎

Response: p*NCC*⏎

Saves the configuration value of the pulse generator with the number `PulserNo` in the variable `Config` and returns an error code according to Tab. 22.

The configuration value, i.e. the value in the variable `Config`, is a bit combination of the values `COM_HVAMX4ED_CFG_XXX` (see the declaration file `COM-HVAMX4ED.h`) and the

Tab. 24. Assignment of the variable PulserNo of the function **COM_HVAMX4ED_GetPulserConfig**.

PulserNo	Assignment
0	SelTrg0 and InvTrg0
1	SelStp0 and InvStp0
2	SelTrg1 and InvTrg1
3	SelStp1 and InvStp1
4	SelTrg2 and InvTrg2
5	SelTrg3 and InvTrg3

bit COM_HVAMX4ED_CFG_INVERT. The values COM_HVAMX4ED_CFG_XXX are numbers between 0 and COM_HVAMX4ED_PULSER_INPUT_MAX-1, they correspond to the numbers SelTrg0-3 and SelStp0-1 (see Fig. 2 and Tab. 1).

The bit value COM_HVAMX4ED_CFG_INVERT corresponds to the inversion bits InvTrg0-3 and InvStp0-1 (see Fig. 2 and Tab. 1).

The configuration number in the variable PulserNo ranges from 0 to COM_HVAMX4ED_PULSER_CFG_NUM-1 (see the declaration file COM-HVAMX4ED.h). For devices with two pulse generators with the burst functionality and two pulse generators without the burst functionality, the assignment of the variable PulserNo is summarized in Tab. 24.

The direct command (p*N*↵) contains one hexadecimal digit (*N*) for the variable PulserNo. If the command is successful, the device response (p*NCC*↵) has the same beginning as the command and additionally returns 2 hexadecimal digits (*CC*) for the variable Config.

Function COM_HVAMX4ED_SetPulserConfig

```
int COM_HVAMX4ED_SetPulserConfig
  (WORD PortNumber, unsigned PulserNo,
  BYTE Config);
```

Command: p*NCC*↵

Response: p*NCC*↵

Sets the configuration value of the pulse generator with the number `PulserNo` to the value given by the variable `Config` and returns an error code according to Tab. 22.

For more details, see function **`COM_HVAMX4ED_GetPulserConfig`**.

The direct command (p*NCC*↵) contains one hexadecimal digit (*N*) for the variable `PulserNo` and 2 hexadecimal digits (*CC*) for the variable `Config`. If the command is successful, the device responds with the same characters as the command.

Switch Configuration

Function COM_HVAMX4ED_GetSwitchTriggerConfig

```
int COM_HVAMX4ED_GetSwitchTriggerConfig
  (WORD PortNumber, unsigned SwitchNo,
  BYTE & Config);
```

Command: e*N*⏎

Response: e*NCC*⏎

Saves the configuration value of the trigger input of the switch with the number `SwitchNo` in the variable `Config` and returns an error code according to Tab. 22.

The configuration number in the variable `SwitchNo` ranges from 0 to `COM_HVAMX4ED_SWITCH_NUM-1` (see the declaration file `COM-HVAMX4ED.h` and section "Power Switches").

For the description of the configuration value, see Fig. 5, Tab. 7, and function **`COM_HVAMX4ED_GetPulserConfig`**.

The direct command (`e`*N*⏎) contains one hexadecimal digit (*N*) for the variable `SwitchNo`. If the command is successful, the device response (`e`*NCC*⏎) has the same beginning as the command and additionally returns 2 hexadecimal digits (*CC*) for the variable `Config`.

Function COM_HVAMX4ED_SetSwitchTriggerConfig

```
int COM_HVAMX4ED_SetSwitchTriggerConfig
  (WORD PortNumber, unsigned SwitchNo,
  BYTE Config);
```

Command: e*NCC*⏎

Response: e*NCC*⏎

Sets the configuration value of the trigger input of the switch with the number `SwitchNo` to the value given by the variable `Config` and returns an error code according to Tab. 22.

For more details, see function **`COM_HVAMX4ED_GetSwitchTriggerConfig`**.

The direct command (`eNCC`⮰) contains one hexadecimal digit (*N*) for the variable `SwitchNo` and 2 hexadecimal digits (*CC*) for the variable `Config`. If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetSwitchEnableConfig

```
int COM_HVAMX4ED_GetSwitchEnableConfig
  (WORD PortNumber, unsigned SwitchNo,
  BYTE & Config);
```

Command: f*N*⮰

Response: f*NCC*⮰

Saves the configuration value of the enable input of the switch with the number `SwitchNo` in the variable `Config` and returns an error code according to Tab. 22.

For more details, see function **`COM_HVAMX4ED_GetSwitchTriggerConfig`**.

The direct command (`fN`⮰) contains one hexadecimal digit (*N*) for the variable `SwitchNo`. If the command is successful, the device response (`fNCC`⮰) has the same beginning as the command and additionally returns 2 hexadecimal digits (*CC*) for the variable `Config`.

Function COM_HVAMX4ED_SetSwitchEnableConfig

```
int COM_HVAMX4ED_SetSwitchEnableConfig
  (WORD PortNumber, unsigned SwitchNo,
  BYTE Config);
```

Command: f*NCC*⮰

Response: f*NCC*⮰

Sets the configuration value of the enable input of the switch with the number `SwitchNo` to the value given by the variable `Config` and returns an error code according to Tab. 22.

For more details, see function **`COM_HVAMX4ED_GetSwitchTriggerConfig`**.

The direct command (`f`*NCC*⏎) contains one hexadecimal digit (*N*) for the variable `SwitchNo` and 2 hexadecimal digits (*CC*) for the variable `Config`. If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetSwitchTriggerDelay

```
int COM_HVAMX4ED_GetSwitchTriggerDelay
  (WORD PortNumber, unsigned SwitchNo,
  BYTE & RiseDelay, BYTE & FallDelay);
```

Command: g*N*⏎

Response: g*NFR*⏎

Saves the delay values of the trigger input of the switch with the number `SwitchNo` in the variables `RiseDelay` and `FallDelay`, and returns an error code according to Tab. 22.

For the description of the configuration number in the variable `SwitchNo`, see function **COM_HVAMX4ED_GetSwitchTriggerConfig**.

The values in the variables `RiseDelay` and `FallDelay` determine the delay of the respective signal slope controlling the trigger input of the switch. They can have any integer value between 0 and `COM_HVAMX4ED_SWITCH_DELAY_MAX-1` (see the declaration file `COM-HVAMX4ED.h`). The step of the delay is given by the hardware implementation and has a typical value of 0.5-1 ns.

The direct command (`g`*N*⏎) contains one hexadecimal digit (*N*) for the variable `SwitchNo`. If the command is successful, the device response (`g`*NFR*⏎) has the same beginning as the command and additionally returns 2 hexadecimal digits (*FR*) - the first for the variable `FallDelay` (*F*) and the second for the variable `RiseDelay` (*R*).

Function COM_HVAMX4ED_SetSwitchTriggerDelay

```
int COM_HVAMX4ED_SetSwitchTriggerDelay
  (WORD PortNumber, unsigned SwitchNo,
  BYTE RiseDelay, BYTE FallDelay);
```

Command: g*NFR*↵

Response: g*NFR*↵

Sets the delay values of the trigger input of the switch with the number `SwitchNo` to the value given by the variables `RiseDelay` and `FallDelay`, and returns an error code according to Tab. 22.

For more details, see function **COM_HVAMX4ED_GetSwitchTriggerDelay**.

The direct command (g*NFR*↵) contains one hexadecimal digit (*N*) for the variable `SwitchNo`, one for the variable `FallDelay` (*F*), and one for the variable `RiseDelay` (*R*). If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetSwitchEnableDelay

```
int COM_HVAMX4ED_GetSwitchEnableDelay
  (WORD PortNumber, unsigned SwitchNo,
  BYTE & Delay);
```

Command: h*N*↵

Response: h*ND*↵

Saves the delay value of the enable input of the switch with the number `SwitchNo` in the variable `Delay` and returns an error code according to Tab. 22.

The value in the variable `Delay` determines the delay of both signal slopes controlling the enable input of the switch. For more details about the delay values, see function **COM_HVAMX4ED_GetSwitchTriggerDelay**.

The direct command (h*N*↵) contains one hexadecimal digit (*N*) for the variable `SwitchNo`. If the command is successful, the device response (h*ND*↵) has the same beginning as the command and additionally returns one hexadecimal digit (*D*) for the variable `Delay`.

Function COM_HVAMX4ED_SetSwitchEnableDelay

```
int COM_HVAMX4ED_SetSwitchEnableDelay
  (WORD PortNumber, unsigned SwitchNo,
  BYTE Delay);
```

Command: h*ND*↵

Response: h*ND*↵

Sets the delay value of the enable input of the switch with the number `SwitchNo` to the value given by the variable `Delay` and returns an error code according to Tab. 22.

For more details, see function **COM_HVAMX4ED_GetSwitchEnableDelay**.

The direct command (h*ND*↵) contains one hexadecimal digit (*N*) for the variable `SwitchNo` and one for the variable `Delay` (*D*). If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetSwitchTriggerMapping

```
int COM_HVAMX4ED_GetSwitchTriggerMapping
  (WORD PortNumber, unsigned MappingNo,
  BYTE & Mapping);
```

Command: m*N*↵

Response: m*NM*↵

Saves the mapping value with the number `MappingNo` of the trigger inputs of the switch in the variable `Mapping` and returns an error code according to Tab. 22.

The mapping number in the variable `MappingNo` ranges from 0 to `COM_HVAMX4ED_MAPPING_NUM-1`, the mapping value in the variable `Mapping` can be any number from 0 to `COM_HVAMX4ED_MAPPING_MAX-1` (see the declaration file `COM-HVAMX4ED.h`, Fig. 7 and Tab. 6).

The direct command (m*N*↵) contains one hexadecimal digit (*N*) for the variable `MappingNo`. If the command is successful, the device response (`m`*NM*↵) has the same beginning as the command and additionally returns one hexadecimal digit (*M*) for the variable `Mapping`.

Function COM_HVAMX4ED_SetSwitchTriggerMapping

```
int COM_HVAMX4ED_SetSwitchTriggerMapping
  (WORD PortNumber, unsigned MappingNo,
  BYTE Mapping);
```

Command: m*NM*↵

Response: m*NM*↵

Sets the mapping value with the number `MappingNo` of the trigger inputs of the switch to the value given by the variable `Mapping` and returns an error code according to Tab. 22.

For more details, see function **`COM_HVAMX4ED_GetSwitchTriggerMapping`**.

The direct command (`m`*NM*↵) contains one hexadecimal digit (*N*) for the variable `MappingNo` and one for the variable `Mapping` (*M*). If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetSwitchEnableMapping

```
int COM_HVAMX4ED_GetSwitchEnableMapping
  (WORD PortNumber, unsigned MappingNo,
  BYTE & Mapping);
```

Command: n*N*↵

Response: n*NM*↵

Saves the mapping value with the number `MappingNo` of the enable inputs of the switch in the variable `Mapping` and returns an error code according to Tab. 22.

For more details, see function **`COM_HVAMX4ED_GetSwitchTriggerMapping`**.

The direct command (`n`*N*↵) contains one hexadecimal digit (*N*) for the variable `MappingNo`. If the command is successful, the device response (`n`*NM*↵) has the same beginning as the command and additionally returns one hexadecimal digit (*M*) for the variable `Mapping`.

Function COM_HVAMX4ED_SetSwitchEnableMapping

int COM_HVAMX4ED_SetSwitchEnableMapping (WORD PortNumber, unsigned MappingNo, BYTE Mapping);

Command: n*NM*↵

Response: n*NM*↵

Sets the mapping value with the number `MappingNo` of the enable inputs of the switch to the value given by the variable `Mapping` and returns an error code according to Tab. 22.

For more details, see function **COM_HVAMX4ED_GetSwitchTriggerMapping**.

The direct command (`n`*NM*↵) contains one hexadecimal digit (*N*) for the variable `MappingNo` and one for the variable `Mapping` (*M*). If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetSwitchTriggerMappingEnable

int COM_HVAMX4ED_GetSwitchTriggerMappingEnable (WORD PortNumber, bool & Enable);

Command: k↵

Response: k*E*↵

Saves the enable bit of the switch trigger mapping in the variable `Enable` and returns an error code according to Tab. 22.

For more details, see section "Power Switches".

The response to the direct command (`k`↵) contains one Boolean character (*E*) for the variable `Enable`.

Function COM_HVAMX4ED_SetSwitchTriggerMappingEnable

```
int
  COM_HVAMX4ED_SetSwitchTriggerMappingEnable
   (WORD PortNumber, bool Enable);
```

Command: kE↵

Response: kE↵

Sets the enable bit of the switch trigger mapping to the value given by the variable `Enable` and returns an error code according to Tab. 22.

For more details, see section "Power Switches".

The direct command (`kE↵`) contains one Boolean character for the variable `Enable` (*E*). If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetSwitchEnableMappingEnable

```
int COM_HVAMX4ED_GetSwitchEnableMappingEnable
   (WORD PortNumber, bool & Enable);
```

Command: l↵

Response: lE↵

Saves the enable bit of the switch enable mapping in the variable `Enable` and returns an error code according to Tab. 22.

For more details, see section "Power Switches".

The response to the direct command (`l↵`) contains one Boolean character (*E*) for the variable `Enable`.

Function COM_HVAMX4ED_SetSwitchEnableMappingEnable

```
int COM_HVAMX4ED_SetSwitchEnableMappingEnable
   (WORD PortNumber, bool Enable);
```

Command: lE↵

Response: lE↵

Sets the enable bit of the switch enable mapping to the value given by the variable `Enable` and returns an error code according to Tab. 22.

For more details, see section "Power Switches".

The direct command (`l`*E*⏎) contains one Boolean character for the variable `Enable` (*E*). If the command is successful, the device responds with the same characters as the command.

Configuration of Digital Terminals

Function COM_HVAMX4ED_GetInputConfig

```
int COM_HVAMX4ED_GetInputConfig
  (WORD PortNumber, BYTE & OutputEnable,
  BYTE & TerminationEnable);
```

Command: `i`⮰

Response: `i`*TTOO*⮰

Saves the configuration values of the digital terminals in the variables `OutputEnable` and `TerminationEnable`, and returns an error code according to Tab. 22.

The configuration values in the variables `OutputEnable` and `TerminationEnable` are bit arrays containing the values for all available digital terminals. The lowest bit corresponds to the first terminal (DIO1). A particular value is enabled if the respective bit is set (see Tab. 3).

The number of valid bits in the variables `OutputEnable` and `TerminationEnable` is given by the constant `COM_HVAMX4ED_DIO_NUM` (see the declaration file `COM-HVAMX4ED.h`). It corresponds to the maximum possible number of implemented digital terminals. If a lower number of terminals is implemented, the settings of the unused terminal do not have any influence on the function of the device.

The response to the direct command (`i`⮰) contains four hexadecimal digits (*TTOO*) - two (*TT*) for the variable `TerminationEnable` and two (*OO*) for the variable `OutputEnable`.

Function COM_HVAMX4ED_SetInputConfig

```
int COM_HVAMX4ED_SetInputConfig
  (WORD PortNumber, BYTE OutputEnable,
  BYTE TerminationEnable);
```

Command: **i*TTOO*↵**

Response: **i*TTOO*↵**

Sets the configuration values of the digital terminals to the values given by the variables `OutputEnable` and `TerminationEnable`, and returns an error code according to Tab. 22.

For more details, see function **`COM_HVAMX4ED_GetInputConfig`**.

The direct command (i*TTOO*↵) contains four hexadecimal digits - the first two (*TT*) for the variable `TerminationEnable` and the last two (*OO*) for the variable `OutputEnable`. If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetOutputConfig

```
int COM_HVAMX4ED_GetOutputConfig
  (WORD PortNumber, unsigned OutputNo,
  BYTE & Configuration);
```

Command: **o*N*↵**

Response: **o*NCC*↵**

Saves the configuration value of the digital output with the number `OutputNo` in the variable `Configuration` and returns an error code according to Tab. 22.

The configuration number in the variable `OutputNo` ranges from 0 to `COM_HVAMX4ED_DIO_INPUT_MAX-1` (see the declaration file `COM-HVAMX4ED.h` and Tab. 2).

For the description of the configuration value, see Fig. 4, Tab. 2, and function **`COM_HVAMX4ED_GetPulserConfig`**.

The direct command (o*N*↵) contains one hexadecimal digit (*N*) for the variable `OutputNo`. If the command is successful, the device response (o*NCC*↵) has the same beginning as the command and additionally returns two hexadecimal digits (*CC*) for the variable `Configuration`.

Function COM_HVAMX4ED_SetOutputConfig

```
int COM_HVAMX4ED_SetOutputConfig
  (WORD PortNumber, unsigned OutputNo,
  BYTE Configuration);
```

Command: o*NCC*↵

Response: o*NCC*↵

Sets the configuration value of the digital output with the number `OutputNo` to the value given by the variable `Configuration` and returns an error code according to Tab. 22.

For more details, see function **`COM_HVAMX4ED_GetOutputConfig`**.

The direct command (o*NOO*↵) contains one hexadecimal digit (*N*) for the variable `OutputNo` and two for the variable `Configuration` (*OO*). If the command is successful, the device responds with the same characters as the command.

Device Configuration

Function COM_HVAMX4ED_GetControllerState

```
int COM_HVAMX4ED_GetControllerState
    (WORD PortNumber, WORD & State);
```

Command: c↵

Response: c*SSSS*↵

Saves the configuration value of the device in the variable `State` and returns an error code according to Tab. 22.

The possible values of the variable `State` are given by the constants `COM_HVAMX4ED_ENB` through `COM_HVAMX4ED_CLRN` (see the declaration file `COM-HVAMX4ED.h` and Tab. 9).

The response to the direct command (`c↵`) contains four hexadecimal digits (*SSSS*) for the variable `State`.

Function COM_HVAMX4ED_SetControllerConfig

```
int COM_HVAMX4ED_SetControllerConfig
    (WORD PortNumber, BYTE Config);
```

Command: c*CC*↵

Response: c*CC*↵

Sets the configuration value of the device to the value given by the variable `Config` and returns an error code according to Tab. 22.

The possible values of the variable `Config` are given by the constants `COM_HVAMX4ED_ENB` through `COM_HVAMX4ED_DIS_DITHER` (see the declaration file `COM-HVAMX4ED.h` and Tab. 9).

The direct command (`c`*CC*↵) contains two hexadecimal digits (*CC*) for the variable `Config`. If the command is successful, the device responds with the same characters as the command.

Configuration Management

Function COM_HVAMX4ED_GetDeviceEnable

```
int COM_HVAMX4ED_GetDeviceEnable
  (WORD PortNumber, BOOL & Enable);
```

Command: E↵

Response: E*B*↵

Saves the enable state of the device in the variable `Enable` and returns an error code according to Tab. 22.

The value in the variable `Enable` determines the behavior of the device on startup or when loading new configuration data. If it is true, i.e. nonzero, the new configuration data may set the enable bit Enb in Tab. 9, i.e. the constant `COM_HVAMX4ED_ENB` in the declaration file `COM-HVAMX4ED.h`. This makes it possible to enable the device automatically on startup or when a new configuration is loaded. When the value of the variable `Enable` is false, i.e. zero, the device remains disabled since the bit Enb is kept reset and it must be enabled by a separate call to the function **`COM_HVAMX4ED_SetControllerConfig`**.

The response to the direct command (`E↵`) contains one Boolean character (*B*) for the variable `Enable`.

Function COM_HVAMX4ED_SetDeviceEnable

```
int COM_HVAMX4ED_SetDeviceEnable
  (WORD PortNumber, BOOL Enable);
```

Command: E*B*↵

Response: E*B*↵

Sets the enable state of the device to the value given by the variable `Enable` and returns an error code according to Tab. 22.

For more details, see function **`COM_HVAMX4ED_GetDeviceEnable`**.

The direct command (`E`*B*↵) contains one Boolean character (*B*) for the variable `Enable`. If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_ResetCurrentConfig

```
int COM_HVAMX4ED_ResetCurrentConfig
  (WORD PortNumber);
```

Command: `*↵`

Response: `*↵`

Resets the current configuration and returns an error code according to Tab. 22.

This function resets all register values of the pulse controller. This disables all oscillators, pulse generators, and power switches. It is recommended to call this function before applying a new configuration.

Function COM_HVAMX4ED_SaveCurrentConfig

```
int COM_HVAMX4ED_SaveCurrentConfig
  (WORD PortNumber, unsigned ConfigNumber);
```

Command: *JNN*↵

Response: *JNN*↵

Saves the current device configuration to the configuration in the NVM with the number given by the variable `ConfigNumber` and returns an error code according to Tab. 22.

The configuration with the number given by the variable `ConfigNumber` can be any integer between 0 and `COM_HVAMX4ED_MAX_CONFIG-1` (see the declaration file `COM-HVAMX4ED.h`).

Note that the call to the function **COM_HVAMX4ED_SaveCurrentConfig** overwrites the previously saved configuration data without any warning.

Any configuration stored in the NVM can be restored, i.e. loaded as the current device configuration by the function **COM_HVAMX4ED_LoadCurrentConfig**.

The direct command (*JNN*↵) contains two hexadecimal digits (*NN*) for the variable `ConfigNumber`. If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_LoadCurrentConfig

int COM_HVAMX4ED_LoadCurrentConfig (WORD PortNumber, unsigned ConfigNumber);

Command: j*NN*↵

Response: j*NN*↵

Loads the current device configuration from the configuration in the NVM with the number given by the variable `ConfigNumber` and returns an error code according to Tab. 22.

There must be a saved configuration in the NVM from a prior call to the function **COM_HVAMX4ED_SaveCurrentConfig**, otherwise the call to the function **COM_HVAMX4ED_LoadCurrentConfig** fails.

For more details, see function **COM_HVAMX4ED_SaveCurrentConfig**.

The direct command (j*NN*↵) contains two hexadecimal digits (*NN*) for the variable `ConfigNumber`. If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetConfigName

int COM_HVAMX4ED_GetConfigName (WORD PortNumber, unsigned ConfigNumber, char Name [COM_HVAMX4ED_CONFIG_NAME_SIZE]);

Command: A*NN*↵

Response: A*NNS*..*S*↵

Saves the name of the configuration in the NVM with the number given by the variable `ConfigNumber` in the variable `Name` and returns an error code according to Tab. 22.

For the configuration number in the variable `ConfigNumber`, see function **COM_HVAMX4ED_SaveCurrentConfig**.

The variable `Name` passed to the function must be created before calling the function. Its size must match the value of the constant `COM_HVAMX4ED_CONFIG_NAME_SIZE` (see the declaration file `COM-HVAMX4ED.h`).

Note that the return value is a copy of the data from the NVM, thus there is no guarantee that the value is a null-terminated character string. To generate this string format, allocate the variable `Name` with one additional character and set the value of the last character of the array to 0 after calling the function **`COM_HVAMX4ED_GetConfigName`**.

The call to the function **`COM_HVAMX4ED_SaveCurrentConfig`** does not modify the configuration name, this is only possible with the function **`COM_HVAMX4ED_SetConfigName`**.

The direct command (`A`*`NN`*↵) contains two hexadecimal digits (*`NN`*) for the variable `ConfigNumber`. If the command is successful, the device response (`A`*`NNS`*`..`*`S`*↵) has the same beginning as the command and additionally returns `2*COM_HVAMX4ED_CONFIG_NAME_SIZE` hexadecimal digits (*`S`*`..`*`S`*) for the variable `Name`. Note that in this way, the variable `Name` is transferred in a binary format and thus can be used to store any general character, not only ASCII ones.

Function COM_HVAMX4ED_SetConfigName

```
int COM_HVAMX4ED_SetConfigName
  (WORD PortNumber, unsigned ConfigNumber,
  const char
  Name [COM_HVAMX4ED_CONFIG_NAME_SIZE]);
```

Command: A*NNS*..*S*↵

Response: A*NNS*..*S*↵

Sets the name of the configuration in the NVM with the number given by the variable `ConfigNumber` to the value given by the variable `Name` and returns an error code according to Tab. 22.

For more details, see function **`COM_HVAMX4ED_GetConfigName`**.

The direct command (`A`*`NNS`*`..`*`S`*↵) contains two hexadecimal digits (*`NN`*) for the variable `ConfigNumber` and `2*COM_HVAMX4ED_CONFIG_NAME_SIZE` hexadecimal digits (*`S`*`..`*`S`*) for the variable `Name`. If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetConfigFlags

```
int COM_HVAMX4ED_GetConfigFlags
  (WORD PortNumber, unsigned ConfigNumber,
  bool & Active, bool & Valid);
```

Command: ***XNN*⏎**

Response: ***XNNF*⏎**

Saves the flags of the configuration in the NVM with the number given by the variable `ConfigNumber` in the variables `Active` and `Valid`, and returns an error code according to Tab. 22.

For the configuration number in the variable `ConfigNumber`, see function **`COM_HVAMX4ED_SaveCurrentConfig`**.

The variables `Active` and `Valid` define the properties of a particular configuration in the NVM. The value `Valid` defines whether the configuration is handled as a valid one. If the value is reset, the configuration is considered to be empty.

The value `Active` determines if the configuration is active (value set) or deleted (value reset). When this flag is flipped by calling the function **`COM_HVAMX4ED_SetConfigFlags`**, the particular configuration can be deleted or undeleted.

Note that only a valid active configuration can be loaded successfully by the function **`COM_HVAMX4ED_LoadCurrentConfig`**.

The direct command (X*NN*⏎) contains two hexadecimal digits (*NN*) for the variable `ConfigNumber`. If the command is successful, the device response (X*NNF*⏎) has the same beginning as the command and additionally returns one hexadecimal digit (*F*) for the configuration flags. The configuration flags are a bit array, the least significant bit (bit 0) corresponds to the variable `Active` and the next bit (bit 1) to the variable `Valid`. This means that an empty configuration returns a flag value of 0, an active one 3, and a deleted one a value of 2.

Function COM_HVAMX4ED_SetConfigFlags

```
int COM_HVAMX4ED_SetConfigFlags
  (WORD PortNumber, unsigned ConfigNumber,
  bool Active, bool Valid);
```

Command: *XNNF*⏎

Response: *XNNF*⏎

Sets the flags of the configuration in the NVM with the number given by the variable `ConfigNumber` to the values given by the variables `Active` and `Valid`, and returns an error code according to Tab. 22.

For more details, see function **COM_HVAMX4ED_GetConfigFlags**.

The direct command (X*NNF*⏎) contains two hexadecimal digits (*NN*) for the variable `ConfigNumber` and one hexadecimal digit (*F*) for the configuration flags (for more details, see function **COM_HVAMX4ED_GetConfigFlags**). If the command is successful, the device responds with the same characters as the command.

Function COM_HVAMX4ED_GetConfigList

```
int COM_HVAMX4ED_GetConfigList
  (WORD PortNumber,
  bool Active [COM_HVAMX4ED_MAX_CONFIG],
  bool Valid [COM_HVAMX4ED_MAX_CONFIG]);
```

Command: W⏎

Response: W*F*..*F*⏎

Saves the flags of all configurations in the NVM in the variables `Active` and `Valid`, and returns an error code according to Tab. 22.

The variables `Active` and `Valid` passed to the function are Boolean arrays that must be created before calling the function. Their size must match the value of the constant `COM_HVAMX4ED_CONFIG_NAME_SIZE` (see the declaration file `COM-HVAMX4ED.h`).

For the detailed description of the values `Active` and `Valid`, see function **COM_HVAMX4ED_GetConfigFlags**.

The function **COM_HVAMX4ED_GetConfigList** can be used to obtain a list of valid configurations without the necessity of periodically calling the function **COM_HVAMX4ED_GetConfigFlags**. This would take a long time, especially with devices using a USB interface with a long latency.

The response to the direct command (`W`⏎) contains `[(COM_HVAMX4ED_MAX_CONFIG*2)/4]` hexadecimal digits (*F*..*F*), where the brackets `[]` denote the integer part of the enclosed argument. The received digits are coded as bit arrays. Analogously to the flag coding for the function **COM_HVAMX4ED_GetConfigFlags**, the least significant bit (bit 0) of the first digit corresponds to the variable `Active` and the next bit (bit 1) to the variable `Valid` of the first configuration (i.e. `ConfigNumber = 0`). The following bits (bit 2 and bit 3) of the digit correspond to the variables `Active` and `Valid`, respectively, of the second configuration (i.e. `ConfigNumber = 1`). The next hexadecimal digit is composed in the same way and contains the data of the next configuration pair (i.e. `ConfigNumber = 2` and `ConfigNumber = 3`), etc.

Various Functions

Function COM_HVAMX4ED_GetSWVersion

```
WORD COM_HVAMX4ED_GetSWVersion();
```

Returns the version of the software interface (the dynamic link library `COM-HVAMX4ED.dll`).

This function should be used to check whether a software interface with the correct version is being used. The function should be called prior to any other function of the software interface. It does not have any influence on the communication and can be called at any time.

The return value is an unsigned 16-bit integer (WORD). The higher byte contains the main version number, the lower byte the subversion, i.e. the version order within the main version. If the version numbers of a library `COM-HVAMX4ED.dll` are different, check the change list provided by the manufacturer. There is no guarantee that a library `COM-HVAMX4ED.dll` with different version numbers can be used without any changes. In most cases, the user software has to be recompiled or modified to match the new definition of the software interface.

Function COM_HVAMX4ED_GetHWType

```
int COM_HVAMX4ED_GetHWType (WORD PortNumber,
  WORD & HWType);
```

Command: t↵

Response: t*HHHH*↵

Saves the device's hardware version in the variable `HWType` and returns an error code according to Tab. 22.

The return value in the variable `HWType` can be used to identify the hardware. i.e. to ensure that the connected device is the desired type. Please contact the manufacturer for further details.

The response to the direct command (`t↵`) contains 4 hexadecimal digits (*HHHH*) for the variable `HWType`.

Function COM_HVAMX4ED_GetHWVersion

```
int COM_HVAMX4ED_GetHWVersion
  (WORD PortNumber, WORD & Version);
```

Command: v↵

Response: v*VVVV*↵

Saves the device's hardware version in the variable `Version` and returns an error code according to Tab. 22.

The return value in the variable `Version` should be used to check whether the hardware is an appropriate version.

The return value is similar to the return value of the function **COM_HVAMX4ED_GetSWVersion**. It is an unsigned 16-bit integer (WORD) containing the main version and the subversion numbers. Check the change list provided by the manufacturer to learn whether the software interface is compatible with the hardware.

The response to the direct command (v↵) contains 4 hexadecimal digits (*VVVV*) for the variable `Version`.

Function COM_HVAMX4ED_GetFWVersion

```
int COM_HVAMX4ED_GetFWVersion
  (WORD PortNumber, WORD & Version);
```

Command: V↵

Response: V*vvvv*↵

Saves the device's firmware version in the variable `Version` and returns an error code according to Tab. 22.

The return value in the variable `Version` should be used to check whether the firmware is an appropriate version.

The return value is similar to the return value of the function **COM_HVAMX4ED_GetSWVersion**. It is an unsigned 16-bit integer (WORD) containing the main version and the subversion numbers. Check the change list provided by the manufacturer to learn whether the software interface is compatible with the firmware.

The response to the direct command (`V↵`) contains 4 hexadecimal digits (*vvvv*) for the variable `Version`.

Function COM_HVAMX4ED_GetFWDate

int COM_HVAMX4ED_GetFWDate (WORD PortNumber, char * DateString);

Command: D↵

Response: D*dd..d*↵

Saves the device's firmware date in the variable `DateString` and returns an error code according to Tab. 22.

The return value in the variable `DateString` is a null-terminated character string with the firmware compilation date. The buffer passed to the function must be created before the function call, it must be at least 16 bytes long.

The response to the direct command (`D↵`) usually contains 11 characters (`dd..d`) for the variable `DateString`.

Function COM_HVAMX4ED_GetProductNo

int COM_HVAMX4ED_GetProductNo (WORD PortNumber, DWORD & Number);

Command: N↵

Response: N*nnnnnnnn*↵

Saves the device's product number in the variable `Number` and returns an error code according to Tab. 22.

The response to the direct command (`N↵`) contains 8 hexadecimal digits (*nnnnnnnn*) for the variable `Number`.

Function COM_HVAMX4ED_GetProductID

```
int COM_HVAMX4ED_GetProductID
  (WORD PortNumber, char * Identification);
```

Command: `P↵`

Response: `P`*`ii..i`*`↵`

Saves the device's product identification in the variable `Identification` and returns an error code according to Tab. 22.

The return value in the variable `Identification` is a null-terminated character string with the product identification. The buffer passed to the function must be created before the function call; it should be 60 characters long.

The response to the direct command (`P↵`) contains a variable number of characters (*`ii..i`*) for the variable `Identification`. Note that the termination character of the string is not transmitted.

Function COM_HVAMX4ED_GetUptime

```
int COM_HVAMX4ED_GetUptime (WORD PortNumber,
  DWORD & Seconds, WORD & Milliseconds,
  DWORD & Optime);
```

Command: `U↵`

Response: `U`*`SSSSSSSSMMOOOOOOOO`*`↵`

Saves the device uptime in the variables `Seconds` and `Milliseconds`, and the operating time in the variable `Optime`. The function return value is an error code according to Tab. 22.

The device uptime is the time elapsed from the last (re)start of the device, the operating time is the portion of the uptime in which the device was activated. The function **`COM_HVAMX4ED_GetUptime`** can be used, for instance, to see whether the device has restarted unexpectedly or to check how long it has been operating.

The response to the direct command (`U↵`) contains 8 hexadecimal digits (*`SSSSSSSS`*) for the variable `Seconds`, 2 hexadecimal digits (*`MM`*) for the variable `Milliseconds`, and 8 hexadecimal digits (*`OOOOOOOO`*) for the variable `Optime`. To scale the value of the variable `Milliseconds` to the abovementioned value, i.e. ms, it has to be multiplied by 3.90625, i.e. divided by 256=100h and multiplied by 1000.

Function COM_HVAMX4ED_GetTotalTime

int COM_HVAMX4ED_GetTotalTime (WORD PortNumber, DWORD & Uptime, DWORD & Optime);

Command: u↵

Response: u*UUUUUUUUOOOOOOOO*↵

Saves the total uptime of the device in the variable `Uptime` and the total operating time in the variable `Optime`. The function return value is an error code according to Tab. 22.

The total uptime and the total operating time are the sum of all uptimes and operating times, respectively (see function **COM_HVAMX4ED_GetUptime**), since the device has been manufactured.

The response to the direct command (`u↵`) contains 8 hexadecimal digits (*UUUUUUUU*) for the variable `Uptime` and 8 hexadecimal digits (*OOOOOOOO*) for the variable `Optime`.

Function COM_HVAMX4ED_GetCPUData

int COM_HVAMX4ED_GetCPUData (WORD PortNumber, double & Load, double & Frequency);

Command: C↵

Response: C*LLLFFFF*↵

Saves the load of the device's CPU in the variable `Load` and its operating frequency in the variable `Frequency`. The function return value is an error code according to Tab. 22.

The CPU load is a value between 0 (= 0%) and 1 (= 100%). Under normal conditions, the CPU load should not exceed 10%. Large data transfers or controlling many fans may increase the load to higher values.

The CPU operating frequency is a value stabilized by a frequency locking loop to about 15.7 MHz. The frequency is also used as a time base for the communication interface. The theoretical frequency value equals 15.6672 MHz, which is a multiple of the maximum communication speed of 230.4 kbaud (see the function **`COM_HVAMX4ED_SetBaudRate`**).

The response to the direct command (`C`⏎) contains 3 hexadecimal digits (*`LLL`*) for the variable `Load` and 4 hexadecimal digits (*`FFFF`*) for the variable `Frequency`. To scale the values to the abovementioned values, the variable `Load` has to be divided by 1000 and the variable `Frequency` multiplied by 1024.

Function COM_HVAMX4ED_Restart

`int COM_HVAMX4ED_Restart (WORD PortNumber);`

`Command: #`⏎

`Response: #`⏎

Restarts the device and returns an error code according to Tab. 22.

The function issues a reboot of the device's CPU and waits until the device responds again after the restart. This may take several seconds.

Note that after the restart, the communication speed is restored to the default value of 9600 baud. If a higher communication speed should be used, the function **`COM_HVAMX4ED_SetBaudRate`** must be called again.

The response to the direct command (`#`⏎) occurs at the currently selected data rate. When the response has been received, the data rate must be restored in the host UART to the default value of 9600 baud to be able to communicate with the device.

Error Handling

Function COM_HVAMX4ED_GetInterfaceState

```
int COM_HVAMX4ED_GetInterfaceState
  (WORD PortNumber);
```

Returns the state of the software interface according to Tab. 22.

This function can be used to obtain the last return value of an interface function. It does not have any influence on the communication and can be called at any time.

Function COM_HVAMX4ED_GetErrorMessage

```
const char * COM_HVAMX4ED_GetErrorMessage
  (WORD PortNumber);
```

Returns the error message corresponding to the state of the software interface (see function **COM_HVAMX4ED_GetInterfaceState**). The return value is a pointer to a null-terminated character string according to Tab. 22.

This function does not have any influence on the communication and can be called at any time.

Function COM_HVAMX4ED_GetIOState

```
int COM_HVAMX4ED_GetIOState
  (WORD PortNumber);
```

Returns the interface state of the serial port according to Tab. 23.

This function can be used to obtain the result of the last I/O operation at the port. It does not have any influence on the communication and can be called at any time.

Function COM_HVAMX4ED_GetIOErrorMessage

```
const char * COM_HVAMX4ED_GetIOErrorMessage
  (WORD PortNumber);
```

Returns the error message corresponding to the interface state of the serial port (see function **COM_HVAMX4ED_GetIOState**). The return value is a pointer to a null-terminated character string according to Tab. 23.

This function does not have any influence on the communication and can be called at any time.