Programmable Digital Pulse Controller

Firmware Version 1-00

User Manual

Document version 2, created on Jun-30-2021

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

- 4 digital pulse generators, 1 digital oscillator
  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
- non-volatile memory (NVM) data space: 16 KB
- maximum number of stored 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

Description

The pulse controller produces digital signals that can be used to control up to four power switches and/or seven digital inputs/outputs. The digital inputs can be used as trigger sources or to synchronize several devices.

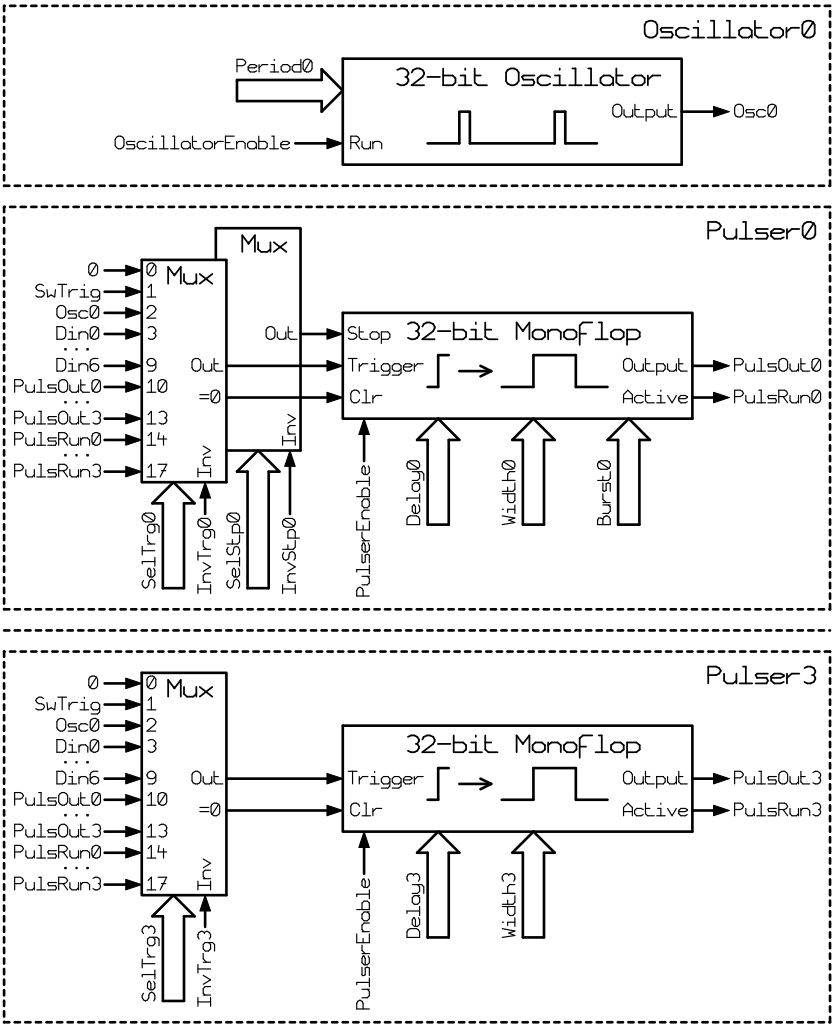

Fig. 1. Block diagram of the pulse controller.

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

Pulse Controller

The controller integrates one digital oscillator and four digital pulse generators, two of them have a burst capability (see Fig. 1). 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. 1) 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. 1) 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. 1 and Tab. 1). The trigger signal can be inverted by setting the control value InvTrigN 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 InvTrigN to 1, the pulse generator is triggered. Note that by setting the signal InvTrigN 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. 2). 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

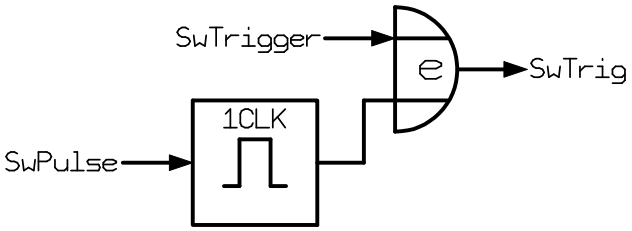

Fig. 2. Software-trigger engine.

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. 1) 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. 1) as described in the previous paragraph. If BurstN is set to any non-zero 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. 1 and Tab. 1). The signal level can be inverted by setting the corresponding signal InvStpN to 1.

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. 3).

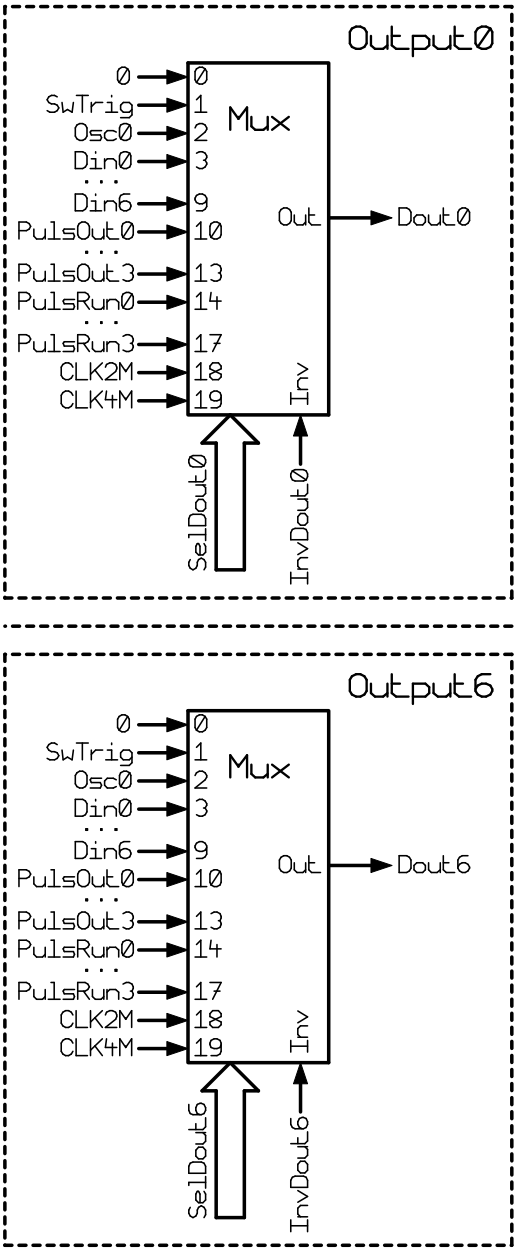

Fig. 3. Output Block Diagram.

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

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
18	CLK2M	Internal quartz clock signal 2 MHz
19	CLK4M	Internal quartz clock signal 4 MHz

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. 1 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. 1 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.

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

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, 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. With-

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

out 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.

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. 2), 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 at-

tempts 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 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 (SwTrig0-3) and enable (SwEnb0-3). 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), respectively.

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

Control Signals		Switch State	
SwTrigN	SwEnbN	Neg-Out	Pos-Out
X	0	off	off
0	1	on	off
1	1	off	on

Similarly to the trigger inputs of the pulse generators (see Fig. 1 and Tab. 1), the switch control signals are selected by an 18-channel multiplexer controlled by the respective 5-bit integer number SelSwTrg0-3 and SelSwEnb0-3 (see Fig. 4 and Tab. 6). The control signals can be inverted by setting the control values InvSwTrgN or InvSwEnbN 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 SelSwEnbN = 0 and InvSwEnbN = 1. Similarly, a certain switch can be enabled by an external signal connected to one of the digital inputs DION by selecting SelSwEnbN = 3-9.

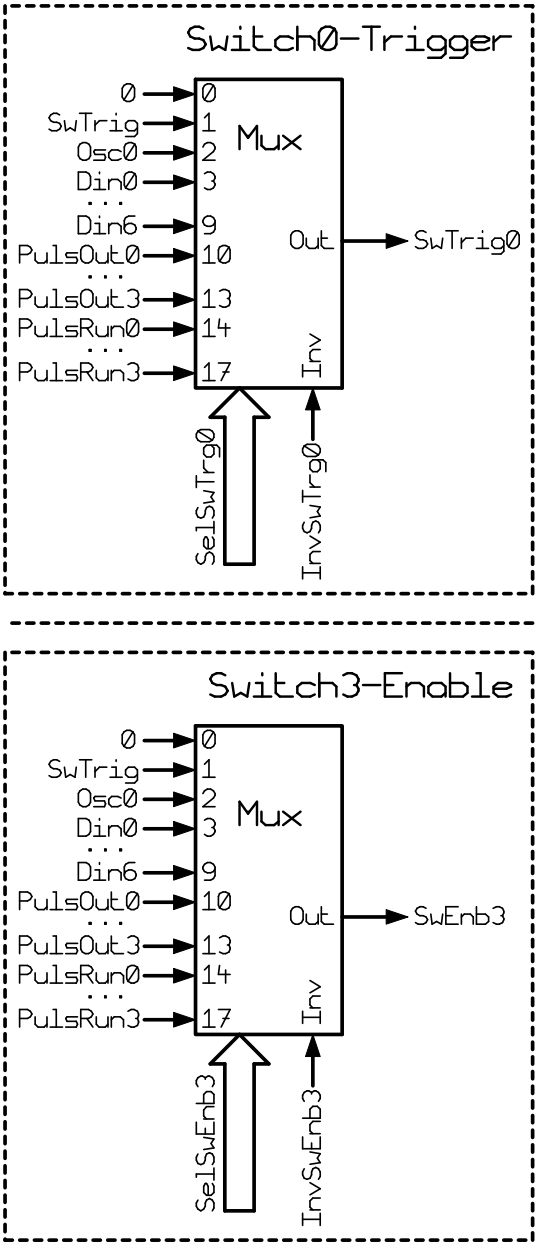

Fig. 4. Block diagram of the switch control.

Tab. 6. 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

The control scheme of a trilevel switch is more complicated (see Tab. 7). 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. 7 in bold.

When the switch is enabled, the output voltage is negative (Vout = Vneg) if both trigger inputs have a low level (SwTrig0/2 = SwTrig1/3 = 0). The positive output voltage (Vout = Vpos) is set if the trigger inputs of the second switch channel have a high level (SwTrig1/3 = 1); in this case, the level of the trigger input of the first switch channel (SwTrig0/2) 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.

Tab. 7. Function of trilevel power switches.

Control Signals				Switch State		
Switch 0/2		Switch 1/3				
SwTrig	SwEnb	SwTrig	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**

The middle output voltage (Vout = Vmid) is reached if the trigger input of the first switch channel has a high level (SwTrig0/2 = 1) and the trigger input of the second switch channel has a low level (SwTrig1/3 = 0).

The remaining rows in Tab. 7 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.

For fine-tuning the output signals of the switches, variable delays can be inserted into the control signals of the switches. The enable signals can be delayed by about 0-15 ns with a resolution of approximately 1 ns. For the trigger signals, each signal slope can be delayed independently from the opposite one. Both delays can be set to similar values as the enable signals, i.e. to 16 different values between roughly 0 ns and 15 ns. This is 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 and the power switch can be compensated.

The direct routing of the control signals to the switch control inputs is sufficient for most operations using the dual-level power switches. A

Tab. 8. 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

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.

For more complex pulse sequences, e.g. for controlling a trilevel switch 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. 5 and Tab. 8).

If enabled, the mapping engine converts the four control signals (either SwTrig0-3 or SwEnb0-3) into different signal levels that are determined by the values Idle and State0-3. For the example mentioned

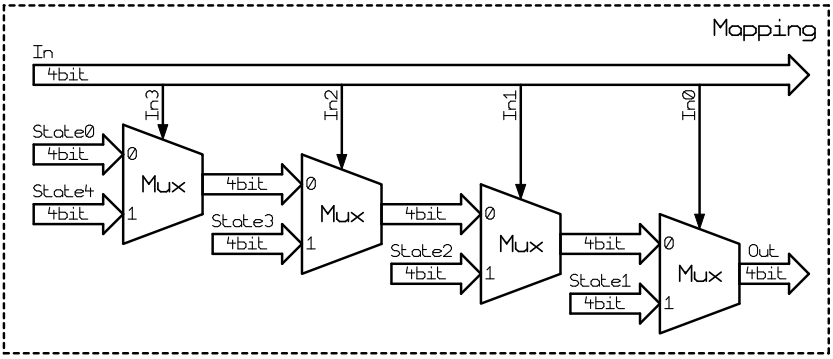

Fig. 5. Mapping engine for the switch control.

Tab. 9. Assignment of 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, also the PSUs are 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, not their PSUs
6	DisDither	Disable the dithering for the PSUs
8	Enable	Master enable, when cleared, CLRn is reset, the oscillator(s) and the pulse generators are stopped, read only
9	SW_Trig_Out	Output of the software-trigger engine, read only
10	CLRn	Clear (CLRn), i.e. device enable output, active low, read only

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 SwTrig0 (SelSwTrg0 = 14 and InvSwTrg0 = 1) and its output (signal PulsOut0) to the next trigger control SwTrig1 (SelSwTrg1 = 10 and InvSwTrg1 = 0). The remaining trigger control signals (SwTrig2-3) will be kept at logic 0 (SelSwTrg2,3 = 0 and InvSwTrg2,3 = 0).

These signals correspond to the bits In0-3 in Fig. 5 and Tab. 8 respectively. When the pulse generator is not running, its negated running state (signal PulsRun0 = In0 in Fig. 5 and Tab. 8) is 1 and the trigger control signals for the switches are determined by the value State1.

When the pulse generator is running, the trigger control signals for the switches toggle between the values State0 and State2. Thus, by defining the values State0-2, any signal waveform can be defined for controlling the switches.

Controller Configuration

The function of the pulse controller can be influenced by changing its configuration (see Tab. 9).

The controller also provides control signals for auxiliary power supplies of the signal switches (PSUs in Tab. 9). 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.

Configuration Management

The configuration of the pulse controller is controlled by software. The current configuration is stored in a non-volatile memory 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 be easily applied by a software command. Beside 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.

The pulse controller is equipped with a USB or RS-232 data interface that allows to transfer configuration data to or from the device and remotely control it. The data interface is galvanically connected to the device's case. When making a connection to a host computer, a large ground loop is created that can influence the performance of the experimental setup.

Tab. 10. Assignment of switch control outputs.

Signal	Output	Explanation
Dout0	DIO1	Digital output #1
...	...	...
Dout6	DIO7	Digital output #7
Ctrl0	Ctrl0	Control input of the power switch #1
...	...	...
Ctrl3	Ctrl3	Control input of the power switch #4
Enb0	Enb0	Enable input of the power switch #1
...	...	...
Enb3	Enb3	Enable input of the power switch #4

The software package for controlling the device contains utilities for uploading or downloading data (see sections "Software Utilities" and "Backing Up and Restoring the Data") and upgrading the firmware (see the section "Utility FlashLoader").

Tab. 11. Assignment of trigger inputs.

Signal	Input	Explanation
Trig0	DIO1-In1	Input 1 of the digital I/O module #1
...	...	...
Trig4	DIO5-In1	Input 1 of the digital I/O module #5

Software Utilities

The software utilities can be found in the directory "Program" of the enclosed software package. Before using them with device with the USB interface, the virtual USB port driver must be installed (see sections "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 (virtual) port number to which the device is attached, as described in the section "Driver Installation".

Utility COM-HVAMX4ED-Control

The `COM-HVAMX4ED-Control` is a simple Windows™ program that runs in text mode. It enables you to control and monitor the pulse controller as well as backup and restore its data. Launching the utility `COM-HVAMX4ED-Control.exe` in a Windows™ command shell[†] without any parameters displays a simple help text:

```
COM-HVAMX4ED-Control
```

To start the program without any error message, at least the number of the COM port must be given:

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

This command starts the utility `COM-HVAMX4ED-Control` and assumes that the device is connected to the virtual port COM6. On success, the utility reports the following message:

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

and waits for command input.

In case of any problem, check whether the port number matches the system settings (see sections "Driver Installation") and whether the connected device is powered on and working properly. If an error oc-

[†] Press the Windows key and the R key at the same time to open the "Run" dialog box and type "cmd". Then change the directory to that with 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 like "Total Commander", "File Commander/W", or "File and archive manager (FAR)". Please use the usual search utilities to find out how to obtain these applications.

Tab. 19. Command line parameters of the program `COM-HVAMX4ED-Control` - Backup and restore.

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

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

This command uploads the memory data from the file `Memory.txt` to the device.

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 Flash-Loader 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 mainboard 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 20 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. 21).

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. 20. Return values of the interface functions

Return value	Error message	Description
0	No error	The data transfer was completed successfully.
-1	PortNumber out of range	The parameter PortNumber specified when calling the function is out of range.
-2	Error opening the port	The port could not be opened. For possible reasons, see Tab. 21.
-3	Error closing the port	The port could not be closed. For possible reasons, see Tab. 21.
-4	Error purging the port	The port buffers could not be cleared.
-5	Error setting the port control lines	The port control lines could not be set.
-6	Error reading the port status lines	The port status lines could not be read.
-7	Error sending command	The data transfer to the device failed. For possible reasons, see Tab. 21.
-8	Error sending data	
-9	Error sending termination character	
-10	Error receiving command	The data transfer from the device failed. For possible reasons, see Tab. 21.
-11	Error receiving data	
-12	Error receiving termination character	
-13	Wrong command received	The device sent an unexpected response.
-14	Wrong argument received	
-15	Wrong argument passed to the function	One of the arguments passed to the function was out of the allowable range.

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. 21. 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. 20.

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. 20.

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. 20.

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. 20.

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: ZE↵

Saves the state of the device's input data buffer in the variable `Empty` and returns an error code according to Tab. 20.

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: zE↵

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. 20.

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: MSSSS↵

Saves the main device status in the variable `State` and returns an error code according to Tab. 20.

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: SDDDDDDDD↵

Saves the detailed state of the device in the variable `DeviceState` and returns an error code according to Tab. 20.

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: HFFFFDDDDCCCCTTTT↵

Saves the housekeeping data in the variables Volt12V, Volt5V0, Volt3V3, and TempCPU, and returns an error code according to Tab. 20.

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: TAAAA..ZZZZ↵

Saves the temperature-sensor data in the array Temperature and returns an error code according to Tab. 20.

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: BEFSSSSMMMMPPPP.. EFSSSSMMMMPPPP↵

Saves the fan data in the arrays `Enabled`, `Failed`, `SetRPM`, `MeasuredRPM`, and `PWM`, and returns an error code according to Tab. 20.

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 be-

tween 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: LRGB↵

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. 20.

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: sPPPPPPPP↵

Saves the oscillator period in the variable `Period` and returns an error code according to Tab. 20.

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. 1 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: sPPPPPPPP↵

Response: sPPPPPPPP↵

Sets the oscillator period to the value given by the variable `Period` and returns an error code according to Tab. 20.

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

The direct command (`sPPPPPPPP↵`) 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: dN↵

Response: dNDDDDDDDD↵

Saves the pulse delay of the pulse generator with the number `PulserNo` in the variable `Delay` and returns an error code according to Tab. 20.

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. 1 and in section "Pulse Controller".

The direct command (`dN↵`) contains one hexadecimal digit (`N`) for the variable `PulserNo`. If the command is successful, the device response (`dNDDDDDDDD↵`) 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: dNDDDDDDDD↵

Response: dNDDDDDDDD↵

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. 20.

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

The direct command (dNDDDDDDDD↵) 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: wN↵

Response: wNWWWWWWWW↵

Saves the pulse width of the pulse generator with the number PulserNo in the variable Width and returns an error code according to Tab. 20.

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. 1 and in section "Pulse Controller".

The direct command (wN↵) contains one hexadecimal digit (N) for the variable PulserNo. If the command is successful, the device response (wNWWWWWWWW↵) 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: wNDDDDDDDD↵

Response: wNDDDDDDDD↵

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. 20.

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

The direct command (`wNWWWWWWWW↵`) 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: bN↵

Response: bNBBBBBB↵

Saves the burst size of the pulse generator with the number `PulserNo` in the variable `Burst` and returns an error code according to Tab. 20.

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 (bN↵) contains one hexadecimal digit (N) for the variable PulserNo. If the command is successful, the device response (bNBBBBBB↵) 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: bNBBBBBB↵

Response: bNBBBBBB↵
```

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. 20.

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

The direct command (bNBBBBBB↵) 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: pN↵

Response: pNCC↵
```

Saves the configuration value of the pulse generator with the number PulserNo in the variable Config and returns an error code according to Tab. 20.

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. 22. 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. 1 and Tab. 1).

The bit value `COM_HVAMX4ED_CFG_INVERT` corresponds to the inversion bits InvTrg0-3 and InvStp0-1 (see Fig. 1 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. 22.

The direct command (`pN↵`) contains one hexadecimal digit (`N`) for the variable `PulserNo`. If the command is successful, the device response (`pNCC↵`) 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: pNCC↵

Response: pNCC↵

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. 20.

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

The direct command (`pNCC↵`) 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: eN↵

Response: eNCC↵

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. 20.

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. 4, Tab. 6, and function **COM_HVAMX4ED_GetPulserConfig**.

The direct command (`eN↵`) contains one hexadecimal digit (`N`) for the variable `SwitchNo`. If the command is successful, the device response (`eNCC↵`) 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: eNCC↵

Response: eNCC↵

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. 20.

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: fN↵

Response: fNCC↵

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. 20.

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: fNCC↵

Response: fNCC↵

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. 20.

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

The direct command (fNCC↵) 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: gN↵

Response: gNFR↵

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. 20.

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 (gN↵) contains one hexadecimal digit (N) for the variable SwitchNo. If the command is successful, the device response (gNFR↵) 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: gNFR↵

Response: gNFR↵

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. 20.

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

The direct command (`gNFR↵`) 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: hN↵

Response: hND↵

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. 20.

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 (`hN↵`) contains one hexadecimal digit (`N`) for the variable `SwitchNo`. If the command is successful, the device response (`hND↵`) 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: hND↵

Response: hND↵

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. 20.

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

The direct command (`hND↵`) 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: mN↵

Response: mNM↵

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. 20.

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. 5 and Tab. 8).

The direct command (`mN↵`) contains one hexadecimal digit (`N`) for the variable `MappingNo`. If the command is successful, the device re-

sponse (mNM↵) 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: mNM↵

Response: mNM↵

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. 20.

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

The direct command (mNM↵) 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: nN↵

Response: nNM↵

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. 20.

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

The direct command (nN↵) contains one hexadecimal digit (N) for the variable MappingNo. If the command is successful, the device re-

sponse (nNM↵) 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: nNM↵

Response: nNM↵

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. 20.

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

The direct command (nNM↵) 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: kE↵

Saves the enable bit of the switch trigger mapping in the variable Enable and returns an error code according to Tab. 20.

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. 20.

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. 20.

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. 20.

For more details, see section "Power Switches".

The direct command (`lE↵`) 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: iTTOO↵

Saves the configuration values of the digital terminals in the variables `OutputEnable` and `TerminationEnable`, and returns an error code according to Tab. 20.

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: iTTOO↵

Response: iTTOO↵

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. 20.

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

The direct command (`iTTOO↵`) 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: oN↵

Response: oNCC↵

Saves the configuration value of the digital output with the number `OutputNo` in the variable `Configuration` and returns an error code according to Tab. 20.

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. 3, Tab. 2, and function **COM_HVAMX4ED_GetPulserConfig**.

The direct command (`oN↵`) contains one hexadecimal digit (`N`) for the variable `OutputNo`. If the command is successful, the device response (`oNCC↵`) 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: oNCC↵

Response: oNCC↵

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. 20.

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

The direct command (oNOO↵) 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: cSSSS↵

Saves the configuration value of the device in the variable `State` and returns an error code according to Tab. 20.

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 for the variable `State`.

Function COM_HVAMX4ED_SetControllerConfig

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

Command: cCC↵

Response: cCC↵

Sets the configuration value of the device to the value given by the variable `Config` and returns an error code according to Tab. 20.

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 (`cCC↵`) 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: EB↵
```

Saves the enable state of the device in the variable `Enable` and returns an error code according to Tab. 20.

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: EB↵

Response: EB↵
```

Sets the enable state of the device to the value given by the variable `Enable` and returns an error code according to Tab. 20.

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

The direct command (`EB↵`) 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. 20.

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. 20.

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: jNN↵

Response: jNN↵

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. 20.

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 (`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_GetConfigName

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

Command: ANN↵

Response: ANNS..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. 20.

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 (`ANN`↵) contains two hexadecimal digits (`NN`) for the variable `ConfigNumber`. If the command is successful, the device response (`ANNS..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: ANNS..S`↵

`Response: ANNS..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. 20.

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

The direct command (`ANNS..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. 20.

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 (`XNN↵`) contains two hexadecimal digits (`NN`) for the variable `ConfigNumber`. If the command is successful, the device response (`XNNF↵`) 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. 20.

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

The direct command (`XNNF↵`) 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: WF..F↵

Saves the flags of all configurations in the NVM in the variables `Active` and `Valid`, and returns an error code according to Tab. 20.

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 re-compiled 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: tHHHH↵

Saves the device's hardware version in the variable `HWType` and returns an error code according to Tab. 20.

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: vVVVV↵

Saves the device's hardware version in the variable `Version` and returns an error code according to Tab. 20.

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: Vvvvv↵

Saves the device's firmware version in the variable `Version` and returns an error code according to Tab. 20.

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: Ddd..d↵
```

Saves the device's firmware date in the variable DateString and returns an error code according to Tab. 20.

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: Nnnnnnnnn↵
```

Saves the device's product number in the variable Number and returns an error code according to Tab. 20.

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: Pii..i↵

Saves the device's product identification in the variable `Identification` and returns an error code according to Tab. 20.

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: USSSSSSSSMMOOOOOOOO↵

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. 20.

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 vari-

able `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: uUUUUUUUUOOOOOOOO↵

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. 20.

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: CLLLFFFF↵

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. 20.

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. 20.

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. 20.

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. 20.

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. 21.

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. 21.

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