# Table of Contents

1 Copyright...........................................................................................................................................4
2 History................................................................................................................................................7
3 Safety...................................................................................................................................................9
4 Overview............................................................................................................................................10
   4.1 Features.......................................................................................................................................10
       4.1.1 E1701D XY2-100 Digital Laser Scanner Controller Baseboard..............................................10
       4.1.2 E1701A Analogue Laser Scanner Controller Baseboard............................................................11
       4.1.3 E1701 LP8 Extension Board.......................................................................................................11
       4.1.4 E1701 Digi I/O Extension Board................................................................................................11
       4.1.5 E1701 Secondary Head Extension Board.....................................................................................11
5 Position Within The System................................................................................................................12
6 Boards And Connectors.......................................................................................................................13
   6.1 E1701D XY2-100 Digital Laser Scanner Controller Baseboard.........................................................13
       6.1.1 Ethernet.....................................................................................................................................13
           6.1.1.1 Ethernet Configuration With Windows.................................................................................14
           6.1.1.2 Ethernet Configuration With Linux......................................................................................14
       6.1.2 USB........................................................................................................................................15
       6.1.3 Power......................................................................................................................................15
       6.1.4 Power LED..............................................................................................................................16
       6.1.5 User LEDs..............................................................................................................................16
       6.1.6 Laser LED..............................................................................................................................16
       6.1.7 Reset-Button............................................................................................................................17
       6.1.8 microSD-Card ..........................................................................................................................17
           6.1.8.1 Firmware Update................................................................................................................22
       6.1.9 Laser/Scanner Signals..............................................................................................................22
           6.1.9.1 XY2-100 Connection Cable.................................................................................................23
       6.1.10 Extension Connectors............................................................................................................24
       6.1.11 Stand-Alone Operation...........................................................................................................25
           6.1.11.1 Create Stand-Alone Data with BeamConstruct..............................................................25
           6.1.11.2 Stand-Alone Configuration Parameters.............................................................................26
           6.1.11.3 Stand-Alone Control..........................................................................................................28
   6.2 E1701A Analogue Laser Scanner Controller Baseboard.................................................................29
       6.2.1 Ethernet.....................................................................................................................................29
       6.2.2 USB........................................................................................................................................29
       6.2.3 Power......................................................................................................................................30
       6.2.4 Power LED..............................................................................................................................30
       6.2.5 User LEDs..............................................................................................................................30
       6.2.6 Laser LED................................................................................................................................30
       6.2.7 microSD-Card..........................................................................................................................30
       6.2.8 Laser/Scanner Signals..............................................................................................................30
       6.2.9 Extension Connectors...............................................................................................................31
       6.2.10 DAC Heatsinks.......................................................................................................................32
       6.2.11 Stand-Alone Operation...........................................................................................................32
   6.3 E1701 LP8 Extension Board...............................................................................................................33
       6.3.1 MO LED...................................................................................................................................33
       6.3.2 Laser Signals............................................................................................................................33
       6.3.3 Extension Connectors...............................................................................................................34
   6.4 E1701 Digi I/O Extension Board......................................................................................................35
       6.4.1 Digi I/O....................................................................................................................................35
           6.4.1.1 Marking On-The-Fly Signals..............................................................................................37
       6.4.2 Opto-Configuration..................................................................................................................38
       6.4.3 Input State LEDs.....................................................................................................................38
   6.5 E1701 Secondary Head Extension Board..........................................................................................39
       6.5.1 Scanner Signals.......................................................................................................................39
       6.5.2 Extension Connectors...............................................................................................................40
7 Quick Start into E1701A/D....................................................................................................................41
8 Command Interface.............................................................................................................................42
1 Copyright
This document is © by HALaser Systems / OpenAPC Project Group.

E1701 base- and extension boards, their hardware and design are copyright / trademark / legal trademark of HALaser Systems / OpenAPC Project Group.

IPG and other are copyright / trademark / legal trademark of IPG Laser GmbH / IPG Photonics Corporation.

Scanlab, RTC4, RTC5 and other are copyright / trademark / legal trademark of Scanlab AG.

SCAPS, USC1, USC2 and other are copyright / trademark / legal trademark of SCAPS GmbH.

Raylase, SP-ICE and other are copyright / trademark / legal trademark of Raylase AG.

Sunny, CSC-USB and other are copyright / trademark / legal trademark of Beijing Century Sunny Technology CO., LTD

All other names / trademarks are copyright / trademark / legal trademark of their respective owners.

Portions of the E1701 firmware are based on lwIP 1.4.0 (or newer):

Copyright (c) 2001, 2002 Swedish Institute of Computer Science.  
All rights reserved.  
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Portions of the E1701 firmware are based on FatFS R0.10a (or newer):

FatFs module is an open source software to implement FAT file system to small embedded systems. This is a free software and is opened for education, research and commercial developments under license policy of following terms.

Copyright (C) 2014, ChaN, all right reserved.

- The FatFs module is a free software and there is NO WARRANTY.
- No restriction on use. You can use, modify and redistribute it for personal, non-profit or commercial product UNDER YOUR RESPONSIBILITY.
- Redistributions of source code must retain the above copyright notice.
Portions of the E1701 firmware are based on StarterWare 2.0 (or newer):

Copyright (C) 2010 Texas Instruments Incorporated - http://www.ti.com/

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (c) 2008-2010 Texas Instruments Incorporated. All rights reserved.

Software License Agreement

Texas Instruments (TI) is supplying this software for use solely and exclusively on TI’s microcontroller products. The software is owned by TI and/or its suppliers, and is protected under applicable copyright laws. You may not combine this software with “viral” open-source software in order to form a larger program.

THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

This is part of AM1808 Sitaraware USB Library and reused from revision 6288 of the Stellaris USB Library.

Portions of the E1701 firmware are based on libzint-backend 2.0 (or newer):

libzint - the open source barcode library, Copyright (C) 2008-2017 Robin Stuart <rstuart114@gmail.com>

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
## 2 History

<table>
<thead>
<tr>
<th>Date</th>
<th>Changes in document</th>
</tr>
</thead>
<tbody>
<tr>
<td>07/2019</td>
<td>Extended description of card state flags</td>
</tr>
<tr>
<td>02/2019</td>
<td>Added &quot;petdh&quot; configuration parameter</td>
</tr>
<tr>
<td>12/2018</td>
<td>Added &quot;haltedloopbuffer&quot; configuration parameter</td>
</tr>
<tr>
<td>10/2018</td>
<td>Added commands <code>cscor</code> and <code>cgor</code></td>
</tr>
<tr>
<td>10/2018</td>
<td>Added &quot;haltedlooptimeout&quot; configuration parameter</td>
</tr>
<tr>
<td>09/2018</td>
<td>New tune-flag added</td>
</tr>
<tr>
<td>06/2018</td>
<td>Added description of flags of function <code>E1701_set_xy_correction3()</code></td>
</tr>
<tr>
<td>04/2018</td>
<td>Description of new parameters &quot;diginit&quot; and &quot;digimask&quot; added</td>
</tr>
<tr>
<td>04/2018</td>
<td>Description of new tune-flags added</td>
</tr>
<tr>
<td>04/2018</td>
<td>Added description of command <code>E1701_digi_pulse()</code></td>
</tr>
<tr>
<td>02/2018</td>
<td>New &quot;d&quot;-command 0x45 to download new firmware</td>
</tr>
<tr>
<td>12/2017</td>
<td>Added description for matrix-&quot;d&quot;-commands 0x40 and 0x41</td>
</tr>
<tr>
<td>10/2017</td>
<td>Added description for <code>E1701_set_scanner_delays2()</code></td>
</tr>
<tr>
<td>08/2017</td>
<td>Added description for config parameters <code>wetout</code> and <code>mipout</code></td>
</tr>
<tr>
<td>08/2017</td>
<td>Description for <code>E1701_set_sync()</code>/<code>E1701_get_sync()</code> added</td>
</tr>
<tr>
<td>07/2017</td>
<td>Description of USB license retrieval clarified</td>
</tr>
<tr>
<td>07/2017</td>
<td>TrueType-support in stand-alone mode</td>
</tr>
<tr>
<td>04/2017</td>
<td>Wiring scheme for IPG YLR types added</td>
</tr>
<tr>
<td>04/2017</td>
<td>Description of stand-alone programming via API added</td>
</tr>
<tr>
<td>04/2017</td>
<td>Pinout for D-SUB15 connector added</td>
</tr>
<tr>
<td>03/2017</td>
<td>Added new API function <code>E1701_set_xy_correction3()</code> and <code>E1701_set_matrix2()</code></td>
</tr>
<tr>
<td>02/2017</td>
<td>Images updated</td>
</tr>
<tr>
<td>01/2017</td>
<td>Added wiring scheme for IPG type E lasers with APD index mode</td>
</tr>
<tr>
<td>12/2016</td>
<td>Added &quot;iohaltedloop&quot; stand-alone mode</td>
</tr>
<tr>
<td>11/2016</td>
<td>Description of <code>corrtable0</code> parameter corrected</td>
</tr>
<tr>
<td>11/2016</td>
<td>API function descriptions added and clarified</td>
</tr>
<tr>
<td>11/2016</td>
<td>Added &quot;haltedloop&quot; stand-alone mode</td>
</tr>
<tr>
<td>10/2016</td>
<td>JPT fiber laser wiring scheme added</td>
</tr>
<tr>
<td>10/2016</td>
<td>Added new API function <code>E1701_set_xy_correction2()</code></td>
</tr>
<tr>
<td>10/2016</td>
<td>Added new API function <code>E1701_set_z_correction()</code></td>
</tr>
<tr>
<td>09/2016</td>
<td>Added description of prebuffering function for <code>ioselect</code> stand-alone mode</td>
</tr>
<tr>
<td>08/2016</td>
<td>Board dimension drawings added</td>
</tr>
<tr>
<td>06/2016</td>
<td>Added description for second marking on-the-fly encoder input</td>
</tr>
<tr>
<td>05/2016</td>
<td>Added standalone commands &quot;cmsor&quot; and &quot;cjisor&quot;</td>
</tr>
<tr>
<td>02/2016</td>
<td>Added wiring schematics for Rayc&quot; fiber laser</td>
</tr>
<tr>
<td>01/2016</td>
<td>Clarified usage of <code>E1701_release_trigger_point()</code></td>
</tr>
<tr>
<td>01/2016</td>
<td>Added protocol description for <code>E1701D</code></td>
</tr>
<tr>
<td>12/2015</td>
<td>Parameter descriptions for Laser On Delay corrected</td>
</tr>
<tr>
<td>10/2015</td>
<td><code>E1701A</code> specification added</td>
</tr>
<tr>
<td>10/2015</td>
<td>Firmware version 21 functions, parameters and commands added</td>
</tr>
<tr>
<td>09/2015</td>
<td>Firmware version 20 function descriptions added</td>
</tr>
<tr>
<td>09/2015</td>
<td>Error in API description/wobble function corrected</td>
</tr>
<tr>
<td>09/2015</td>
<td>Laser wiring description corrected</td>
</tr>
<tr>
<td>06/2015</td>
<td>Stand-alone description extended, <code>E1701_write()</code> added</td>
</tr>
<tr>
<td>04/2015</td>
<td>New I/O and halt- function, new tune parameters</td>
</tr>
<tr>
<td>04/2015</td>
<td>SPI G4 wiring schemes extended</td>
</tr>
<tr>
<td>03/2015</td>
<td>Description of &quot;d&quot;-commands for sending marking data via USB/Ethernet</td>
</tr>
<tr>
<td>02/2015</td>
<td>Pin-out of <code>XY2-100</code> connector added, SNTP and stand-alone feature description added, ASCII-command description added</td>
</tr>
<tr>
<td>01/2015</td>
<td>Specification of default states of digital outputs added</td>
</tr>
<tr>
<td>12/2014</td>
<td>Description for <code>E1701_mark_pixelline()</code> added</td>
</tr>
<tr>
<td>10/2014</td>
<td>Output current values added, description for Secondary Head Extension added</td>
</tr>
<tr>
<td>Date</td>
<td>Notes</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>09/2014</td>
<td>Minor corrections in layout and text</td>
</tr>
<tr>
<td>09/2014</td>
<td>Initial version</td>
</tr>
</tbody>
</table>
3 Safety

The hardware described within this document is designed to control a laser scanner system. Laser radiation may effect a person's health or may otherwise cause damage. Prior to installation and operation compliance with all relevant safety regulations including additional hardware-controlled safety measures has to be secured. The client shall solely be responsible to strictly comply with all applicable and relevant safety regulations regarding installation and operation of the system at any time.

Beside of that some laser equipment can be damaged in case it is controlled with wrong signals or signals outside a given specification. Thus it is highly recommended to check the output generated by this device using e.g. an oscilloscope to avoid problems caused by wrong configurations. This should be done prior to putting a system into operation for the first time, whenever some parameters have been changed or whenever any kind of software update was installed.

The hardware described here is shipped without any cover and without prefabricated equipment for electric installation. It is intended to be integrated in machines or other equipment. It is not for use "as is". Prior to operation compliance with all relevant electric / electromagnetic safety regulations including additional hardware-controlled safety measures has to be secured. The client shall solely be responsible to strictly comply with all applicable and relevant regulations regarding installation and operation of the system at any time.

The hardware described here is an electrostatic sensitive device. This means it can be damaged by common static charges which build up on people, tools and other non-conductors or semiconductors. To avoid such a damage, it has to be handled with care and including all relevant procedures (like proper grounding of people handling the devices, shielding/covering to not to let a person touch the device unwanted, proper packaging in ESD-bags, ...). For more information please refer to related regulations and standards regarding handling of ESD devices.

This document describes the E1701-hardware but may contain errors or may be changed without further notice.
4 Overview

This document describes the E1701 modular scanner controller board family, their electrical characteristics and usage. They consist of E1701D XY2-100 scanner controller baseboard plus optional extension boards and of E1701A analogue scanner controller baseboard plus optional extension boards. Special variant E1701M is no scanner controller and therefore not covered by this document.

The E1701 scanner controller boards are designed for controlling galvanometric scanner systems with two or three axes. Depending on the used extension boards (which are optional) they also supply extensive signals for laser and external control. The communication between the host system and the controller boards is done via Ethernet or USB.

When using E1701 scanner controller boards, there is always one baseboard required for proper operation. This baseboard can be used together with different extension boards that provide additional signals for controlling the laser marking process. These extension boards are optional and have to be used only in environments where the additional signals processed by these boards are required. So depending on used type of laser and requirements, the minimal solution to control a laser marking system may consist of the baseboard only.

Normally extension boards can be combined with any baseboard and all other extension boards freely, there are no restrictions for usage. In case some specific extension board types can’t be operated with other boards, this is stated in description of the related boards below.

Normally an E1701 baseboard can be combined with several extension boards of different types but not with more than one board of same type. In case of special extension boards where more than one board of the same type can be used, this is stated in description of the related board below.

4.1 Features

Following the features of available base- and extension boards are described

4.1.1 E1701D XY2-100 Digital Laser Scanner Controller Baseboard

This baseboard can be used to control 2D or 3D scanheads that come with a XY2-100 or XY2-100-E interface. It can be combined with extension boards without any restrictions. E1701D offers following features:

- XY2-100 interface to scanhead with X, Y and Z channel
- XY2-100-E interface to scanhead with X and Y channel
- 100 Mbit Ethernet connection
- USB 2.0 connection
- online XYZ grid correction with support for several correction table file formats (like SCAPS(tm).ucf, Scanlab(tm).ctb and .ct5, Raylase(tm).gcd)
- high-definition online XYZ grid correction with BeamConstruct HD correction files (.bco)
- fast switching between up to 16 preloaded grid correction tables
- 10 microseconds vector cycle time and resolution (microstep period)
- command execution time down to 0.5 microseconds
- realtime processing of laser and scanner signals
- 26 bit internal resolution (for better quality also with 16 or 18 bit hardware output)
- can control nearly every laser type (this may require extension boards as described below)
- two laser CMOS digital outputs for usage with YAG, CO2, IPG(tm) and compatible laser types (outputs can provide PWM frequency, Q-Switch, FPK-pulse, CW/continuously running frequency, stand-by frequency) running with frequencies of up to 20 MHz
- 512 MByte DDR3 RAM
- 1 GHz CPU clock
- support for microSD and microSDHC cards
- internal command and vector data list with more than 17 million entries
- continuous list concept, no need to swap between buffers
- BeamConstruct PRO license included
- open source compatibility library that emulates existing programming interface for fast and easy usage with existing software (contains e.g. Scanlab(tm) RTC4(tm), SCAPS(tm) USC(TM)/SCI and other
4.1.2 E1701A Analogue Laser Scanner Controller Baseboard

This baseboard can be used to control 2D scanheads that come with an analogue control interface. It can be combined with all extension boards except Secondary Head Extension. E1701A offers following features:

- +5V / -10V bit analogue interface with 16 bit resolution each to control scanheads with X and Y channel
- 3x +5V analogue output with 12 bit resolution each
- 100 Mbit Ethernet connection
- USB 2.0 connection
- online XY grid correction with support for several correction table file formats (like SCAPS(tm) .ucf, Scanlab(tm) .ctb and .ct5, Raylase(tm) .gcd)
- high-definition online XY grid correction with BeamConstruct HD correction files (.bco)
- fast switching between up to 16 preloaded grid correction tables
- 10 microseconds vector cycle time and resolution (microstep period)
- command execution time down to 0.5 microseconds
- realtime processing of laser and scanner signals
- 26 bit internal resolution (for better quality also with 16 bit hardware output)
- can control nearly every laser type (this may require extension boards as described below)
- 512 MByte DDR3 RAM
- 1 GHz CPU clock
- support for microSD and microSDHC cards
- internal command and vector data list with more than 17 million entries
- continuous list concept, no need to swap between buffers
- BeamConstruct PRO license included
- open source compatibility library that emulates existing programming interface for fast and easy usage with existing software (contains e.g. Scanlab(tm) RTC4(tm), SCAPS(tm) USC(TM)/SCI and other compatible interfaces)

4.1.3 E1701 LP8 Extension Board

This board can be used to provide signals for controlling a wide range of laser types. It offers following features:

- LP8 8 bit CMOS level parallel digital output e.g. for controlling laser power
- LP8 latch CMOS level digital output for usage with IPG(tm) and compatible laser types
- Master Oscillator CMOS level digital output for usage with IPG(tm) and compatible laser types
- 8 bit 0..5V analogue output e.g. for controlling laser power (this output is a slave of LP8 outputs)
- two laser CMOS level digital outputs for usage with YAG, CO2, IPG(tm), SPI(tm) and compatible laser types (outputs can provide PWM frequency, Q-Switch, FPK-pulse, CW/continuously running frequency, stand-by frequency) running with frequencies of up to 20 MHz

4.1.4 E1701 Digi I/O Extension Board

This board provides additional digital in- and outputs for synchronisation and communication with external equipment. It offers following features:

- 8 freely usable digital outputs providing either CMOS level or electrically insulated outputs via external power supply
- 8 freely usable digital inputs expecting either CMOS level or electrically insulated inputs via external power supply
- 2 digital inputs usable for quadrature encoder signals for marking on-the-fly applications

4.1.5 E1701 Secondary Head Extension Board

Using boards of this type additional heads can be connected which then work in parallel to the first scanhead of E1701D baseboard. So as only output it provides an additional XY2-100 connection. This extension can't be used with E1701A baseboard.
5 Position Within The System

The E1701 scanner controller system can be connected to the host via Ethernet or USB to receive laser marking data from BeamConstruct laser marking application, from ControlRoom process control software or from any other application which makes use of one of the provided programming possibilities (as described below). When using Ethernet connection, it optionally can be connected via USB too. In this case USB connection is used to retrieve BeamConstruct PRO license from the board:

Since 100 Mbit Ethernet provides much faster data transfer than USB 2.0, this connection type is preferred. Especially in case complex marking data with many short lines that result in many separate jump and mark commands are used, Ethernet connection is more responsive. When using USB connection with such data, time from sending data to the card until marking operation can be started may be longer (up to several seconds in worst case) caused by slower USB data transfer:

In both cases the board itself is connected with the scanhead to submit 2D or 3D position information to it. Beside of that it is connected to a laser to submit motion-synchronous laser data. Additional communication channels between the E1701 scanner controller board and a connected machine can be done via separate IOs of an extension board.
6 Boards And Connectors

6.1 E1701D XY2-100 Digital Laser Scanner Controller Baseboard

The E1701D Digital Laser Scanner Controller Baseboard provides following connectors and interfaces:

1. Ethernet – for communication with the host system, marking information are submitted via this path
2. USB – via miniUSB connector for providing BeamConstruct PRO license to host system and optionally for submitting marking data from host to E1701D card (in case Ethernet is not used)
3. Power – connect with power jack 5V DC
4. Power LED – lights when power is available
5. User LEDs – show operational and error states of card
6. Laser LED – shows modulation state of laser
7. Reset-button – on-board button to restart the board completely
8. microSD-card (on bottom side) – storage place for firmware and extended configuration file, can be used to upgrade firmware, to change the card’s IP and other things more
9. Laser/Scanner signals – white 26 pin laser and scanner output connector which provides XY2-100 / XY2-100-E scanner signals as well as laser and marking control IOs
10. Extension connectors – extension boards can be placed here in order to add some more functionality and hardware interfaces to the board

6.1.1 Ethernet

This is a standard RJ45 Ethernet plug for connection of the board with the host system. The controller board is accessed via this connection, all scanner and laser data are sent via Ethernet. Thus it is recommended for security reasons to have a separate 1:1 connection from the host to the scanner controller card by using a separate Ethernet port. In case this is not possible at least an own, physically separated sub-net for all scanner controller cards should be set up. This network of course should be separated from normal network completely. Ethernet connection is initialised during start-up, thus Ethernet cable connecting E1701 board and host system needs to be plugged before the board is powered up.

By default the E1701 baseboard is using IP 192.168.2.254, thus the Ethernet network the card is connected with needs to belong to subnet 192.168.2.0/24.

PLEASE NOTE: For security reasons it is highly recommended to not to mix a standard communication network with an E1701 network or to connect the scanner controller card with a standard network. Here it may be possible someone else in that network (accidentally) connects to that scanner controller and causes laser emission.

The IP of the scanner controller can be changed. This is necessary e.g. in case an other subnet has to be used or in case the E1701 board has to be operated in multi-head environments where more than one card will be accessed at the same time. The IP can be configured using e1701.cfg configuration file that is placed on microSD-card. To change the IP please perform the following steps:

1. disconnect E1701 board from power and USB
2. remove microSD-card
3. put microSD-card into a desktop computer, this may require a microSD- to SD-card-adapter
4. open the drive that is assigned to the card
5. open file e1701.cfg using a text editor like Notepad or kwrite
6. add a line or edit an existing line "ip1=", here the desired IP has to be appended (as example: when you want to configure IP 192.168.2.13 the line has to be "ip1=192.168.2.13" – without any quotation signs
7. save the file
8. eject the drive the card is assigned to
9. place the microSD-card in E1701 board (place without the use of force, notice correct orientation with connectors of microSD-card to bottom!)
10. power up card

When User LEDs do not light up as described below, please check if microSD-card is placed in board correctly.

6.1.1.1 Ethernet Configuration With Windows

When E1701D scanner controller is accessed via Ethernet, it is recommended to have a 1:1 connection to the host PC for security reasons. Since the controller is working with a static IP (default is 192.168.2.254) the Ethernet port on host PC has to be configured with an IP of same subnet in order to allow access to it. For Windows 7 (and similar) this configuration has to be done using following steps:

1. select "Start"-button and choose entry "Control Panel"
2. Select "Network and Sharing Center"
3. Select "Change adapter settings" in upper left corner
4. find the network interface E1701D has to be connected with and select it
5. find entry "Internet Protocol Version 4 (TCP/IP4)"
6. select it and press button "Properties"
7. now a window opens that has to be configured as follows:

![Internet Protocol Version 4 (TCP/IP4) Properties](image)

There you can specify an IP for your host PC. It has to belong to network 192.168.2.xxx and can be any number except than 192.168.2.254 (this is already the IP of the scanner card), 192.168.2.0 or 192.168.2.255.

6.1.1.2 Ethernet Configuration With Linux

When E1701D scanner controller is accessed via Ethernet, it is recommended to have a 1:1 connection to the host PC for security reasons. Since the controller is working with a static IP (default is 192.168.2.254) the Ethernet port on host PC has to be configured with an IP of same subnet in order to allow access to it. For Linux (with NetworkManager) this configuration has to be done using following steps:

1. right-click the network-symbol in taskbar
2. click "Edit Connections..."
3. select the "Wired" network interface the scanner card is connected with and press button "Edit"
4. go to tab-pane "IPv4 Settings" and configure it as shown below:

![IPv4 Settings Configuration](image)

There you can specify an IP for your host PC. It has to belong to network 192.168.2.xxx and can be any number except than 192.168.2.254 (this is already the IP of the scanner card), 192.168.2.0 or 192.168.2.255.

### 6.1.2 USB

This is a standard miniUSB-connector for connection of the board with the host system. It is used to retrieve BeamConstruct PRO license and optionally – when Ethernet is not connected – to send marking data to the card.

**PLEASE NOTE:** USB 2.0 is much slower than a standard 100 Mbit Ethernet connection, so expect slower execution in case of complex marking data!

Required device driver is installed together with OpenAPC-setup (Windows) or comes with operating system by default (Linux). E1701 card appears as COM-interface on Windows using any free number for the port. With Linux it appears as /dev/ttyACMx where “x” is any number. These numbers are provided by the operating system automatically.

By default USB provides 5V power supply too. So whenever card has to be stopped, both USB and power have to be disconnected in order to shut it down completely. It is not recommended to use USB as power supply, an additional, external power should be connected in order to operate E1701 controller correctly. Nevertheless it might be possible E1701 card can be operated on USB power only. Since this highly depends on the capabilities of used host system, it has to be evaluated for every particular case.

When the controller is connected via USB, a BeamConstruct PRO license is provided via this interface automatically. This is done without the need to configure anything, and as long as following conditions are true:

- physical USB connection from controller to host PC exists
- the COM-port (Windows) has a number smaller than COM20
- the controller is working and the Alive-LED in blinking

It is also possible to have the USB-connection for license retrieval only and to use the Ethernet-connection to transfer marking data to the controller, both can exist beside each other.

### 6.1.3 Power

Power supply for E1701 scanner controller board is done via power jack right beside Ethernet port. Power can
be supplied via a 2.1 mm x 5.5 mm centre connector when connected to a positive power supply rated at 5V DC +/- 0.1V and 2.5A (smoothed, positive pole on inner contact). Do not apply voltages in excess of 5V to the DC input. The DC power supply must be grounded.

To avoid high frequency interferences from other electrical equipment or from within the power supply, it is recommended to place a ferrite bead at the cable close to the board. Please also check for correct shielding in respect to the equipment the E1701 card is used within.

6.1.4 Power LED

This led is lit as soon as the board is on some power. This means it may be functional and could emit any signals as soon as this LED is on but it does not necessarily need to work properly since firmware may not be started at this point. Please refer section below for LEDs that show functional state of the board.

6.1.5 User LEDs

The real operational state of the card is shown by four additional LEDs described here from inner to outer position:

1. Boot- and Alive-LED – this LED is turned on permanently as soon as the card was powered up and the firmware boots properly. When it is not turned on after some seconds please check if the microSD-card is placed properly and if it contains a working firmware file (for details please refer below). After boot process has completed successfully, it starts blinking slowly. This is an alive-notification, as long as it blinks, the board is working and ready for operation. During marking operations the blink frequency may go down. Only in case it does not blink for more than 10 seconds, the board has died for some reason and should be restarted.
2. Marking Active LED – this LED is turned on as long as a marking operation is running. This LED does not correspond to the laser gate signal, comparing to it it's also enabled during jumps when laser is turned off but marking operation itself is active.
3. Stop LED – this LED is lit as long as a valid external stop signal is detected.
4. Error-LED – this LED is turned on in case a fatal error occurs that normally should never happen. When it is on, in most cases board can't continue with operation until the reason for error is removed and the board is restarted. In case this LED is turned on please:
   - check if you are using exactly one baseboard
   - check if you are using E1701 extension boards only (and no other 3rd party hardware)
   - check if you are using latest firmware and host software
   - check all connections and cables
   - undo your latest changes in hardware and configuration
If these steps do not help, please contact HALaser Systems for further assistance.

6.1.6 Laser LED

This LED shows modulation state of the laser and signal of laser gate output. It is turned on as long as the laser is turned on and the laser gate is high. This LED does NOT signal the same like the marking active LED described above since it will be turned off during jumps.
6.1.7 Reset-Button

When this button is pressed for at least 20 milliseconds, it restarts the card completely, a current marking operation is cancelled, all signals are disabled and all remaining marking data are dropped. After releasing this button, the board is rebooted and firmware is started again.

6.1.8 microSD-Card

The microSD card is storage place for firmware and configuration files. Here SD and SDHC cards with storage space of up to 32 GB are supported.

To remove the microSD-card, first disconnect all power from the E1701 board completely (including USB, Power LED has to go off). Next press microSD card gently into the board until you can hear a click-noise. Then you can pull it out of the board. To place a microSD card, the same has to be done in reverse order: place it into the E1701 board’s card slot and press it gently until a noise signals locking of the card. Now the board can be powered.

E1701 baseboard is shipped with a card containing firmware and configuration files:

- E1701.fwi - firmware file that is used to operate the board, to be replaced when a firmware update is provided
- E1701.cfg - configuration text file, can be edited using a text editor in order to modify cards configuration
- E1701.dat – additional data file that is used to operate the board, to be replaced when a firmware update is provided

To use an other microSD card than the one shipped with the board, following conditions have to be met:

- maximum total size of 32 GB (SD or SDHC card)
- FAT32 formatted
- using only one partition
- BOOT-flag is set
- E1701.fwi and e170.dat file available on card

An additional file E1701.cfg can be placed on the card too. It contains plain ASCII text, acts as configuration file and can contain several parameters and its values which are separated by an equal-sign. Every of the possible parameter/value pairs has to be located in an own line. Following configuration parameters are possible within this file:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip1</td>
<td>Configures IP of Ethernet port. Here only IPs in xxx.xxx.xxx.xxx notation are allowed but no host or domain names.</td>
<td>ip1=192.168.2.100 specifies IP 192.168.2.100 to be used for Ethernet interface on next startup</td>
</tr>
<tr>
<td>corrttable0</td>
<td>Specifies a correction table file in .ctb, .ct5, .ucf, .gcd or .bco format to be loaded on start-up. When this parameter is set, the specified correction table is used exclusively and all correction data possibly sent from the host are ignored. The correction file itself has to be located on microSD-card too. This method has also to be used when running the controller in stand-alone mode with .EPR files that require such a correction. When the Error-LED is turned on after a correction table file was configured, E1701 baseboard was not able to load it for some reason.</td>
<td>corrttable0=0:/D2_200.ctb use file D2_200.ctb as correction file and ignore all correction tables possibly sent from host application</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>corrtype&lt;idx&gt;</td>
<td>Specifies one of up to 16 correction table file in .ctb, .ct5, .ucf, .gcd or .bco format to be loaded on start-up. When this parameter is set, the specified correction table is used exclusively and all correction data possibly sent from the host are ignored. The correction file itself has to be located on microSD-card too. This method has also to be used when running the controller in stand-alone mode with .EPR files that require such a correction. When the Error-LED is turned on after a correction table file was configured, E1701 baseboard was not able to load it for some reason. &lt;idx&gt; can be any value in range 0..15 and specifies the storage location index of the correction file to be loaded. Later the related correction file can be used via command cscor. When &lt;idx&gt; has to be set to values greater than 0, a firmware version 33 or newer is needed.</td>
<td>corrtype?=0:/200_200.bco use file 200_200.bco as correction file at index position 7 and ignore all correction tables possibly sent from host application</td>
</tr>
<tr>
<td>passwd</td>
<td>Specifies an access password that is checked when card is controlled via Ethernet connection. This password corresponds to password specified with function E1701_set_password(), please refer below for a detailed description. When a client computer connects to the card without sending the correct password, Ethernet connection to this host is closed immediately. PLEASE NOTE: this password does not replace any network security mechanisms and does not give the possibility to operate E1701 controller via insecure networks or Internet! It is transferred unencrypted and therefore can be &quot;hacked&quot; easily. Intention of this password is to avoid collisions between several E1701 cards that operate in same network and are accessed by several software instances. Maximum allowed length of the password is 48 characters. It is recommended to not to use any language-specific characters.</td>
<td>passwd=myCardPwd set a password &quot;myCardPwd&quot;</td>
</tr>
<tr>
<td>standalone</td>
<td>This command can be used to disable or enable a specific stand-alone operation mode. For a detailed description of possible parameters, operation modes and usage please refer related section below.</td>
<td></td>
</tr>
<tr>
<td>haltedlooptimeout</td>
<td>This parameter is used in stand-alone modes &quot;haltedloop&quot; and &quot;iohaltedloop&quot; (please refer to section &quot;6.1.11 Stand-Alone Operation&quot; for detailed information). It defines a timeout for the laser in unit seconds. If the current operation is active for a longer time, the laser is turned off. It then can be turned on only by toggling the enable-input (ExtStart) again. This parameter requires firmware version 33 or newer.</td>
<td>haltedlooptimeout=5 sets the laser timeout to 5 seconds</td>
</tr>
<tr>
<td>haltedloopbuffer</td>
<td>This parameter is used in stand-alone modes &quot;haltedloop&quot; and &quot;iohaltedloop&quot; (please refer to section &quot;6.1.11 Stand-Alone Operation&quot; for detailed information). It defines a maximum buffer size for the marking data. The buffer size should have a size of 17000000 at max. The minimum size depends on the specific application, in fact, when it is set to some too small values, drop-outs in marking operation may occur. Data which are already buffered in this marking mode can’t be modified any longer. So any change on marking speed, power or similar (done e.g. by commands &quot;cjsor&quot;,&quot;cmsgor&quot; or &quot;cpwor&quot;) will apply only to data which are not yet buffered. And as bigger as this buffer is, as longer it takes until the first new data after change of any of these parameters can be emitted. This parameter requires firmware version 33 or newer.</td>
<td>haltedloopbuffer=10000 set the buffer to a maximum size of 100000 commands which is similar to data for about 1 second marking time</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>autofile</td>
<td>Loads a special .EPR stand-alone file from SD-card in some specific stand-alone modes. For a detailed description of possible parameters, operation modes and usage please refer related section below.</td>
<td>autofile=0:/markdata.epr loads a file markdata.epr from disk; here 0:/ specifies the SD-card to be used. The .EPR-file itself can be generated within BeamConstruct out of a normal .BEAMP project file.</td>
</tr>
<tr>
<td>iobuf</td>
<td>Preloads one or more .EPR files to the RAM of the controller to allow faster switching in &quot;ioselect&quot; stand-alone mode. It can not be used to load file &quot;0.epr&quot;. This parameter requires firmware version 27 or newer.</td>
<td>iobuf=1 preload file 1.EPR on board start-up</td>
</tr>
<tr>
<td>mipout</td>
<td>Configure a Digi I/O output pin to be used as &quot;mark in progress&quot;-signal by default; here an output bit number in range 0..7 has to be configured which will be set to HIGH as long as a marking operation is in progress, the value given here can be overwritten by API-function E1701_digi_set_mip_output(); this parameter requires firmware version 30 or newer</td>
<td>mipout=1 use DOut1 for mark-in-progress signal</td>
</tr>
<tr>
<td>wetout</td>
<td>Configure a Digi I/O output pin to be used as &quot;wait for external trigger&quot;-signal by default; here an output bit number in range 0..7 has to be configured which will be set to HIGH as long as a marking operation is in progress and the controller is waiting for an external trigger signal to arrive at ExtStart input, the value given here can be overwritten by API-function E1701_digi_set_wet_output(); this parameter requires firmware version 30 or newer</td>
<td>wetout=0 use DOut0 for mark-in-progress signal</td>
</tr>
<tr>
<td>digiinit</td>
<td>Initialises the digital outputs on firmware start-up with the given defaults. This overrides the hardware defaults. The default digital values set here are NOT available on power up but a few seconds later after firmware has been loaded and started. This function requires firmware version 32 or newer.</td>
<td>digiinit=2 set DOut1 to HIGH initially and all other outputs to LOW</td>
</tr>
<tr>
<td>digimask</td>
<td>Masks the digital inputs and specifies which inputs can be read. All input bits which are ignored by this command by setting the related value to 0, are no longer read. This may be useful for applications where encoder inputs are used together with a IOSelect stand-alone operation and where the random state of the encoder has to be masked out. This function requires firmware version 32 or newer.</td>
<td>digimask=253 use only DIn2..DIn7 as input and ignore DIn0 and DIn1</td>
</tr>
<tr>
<td>digidebc</td>
<td>Sets a debouncing time / filter time for the digital inputs of the Digi IO extension board in order to not to let the inputs react on noise or bouncing of mechanical inputs. The debouncing value is given in time-units where every time-unit is equal to 31 usec. By default 7 time-units are set.</td>
<td>digidebc=10 set the debounce-time to 310 usec</td>
</tr>
<tr>
<td>fastdebc</td>
<td>Sets a debouncing time / filter time for the ExtStart and ExtStop inputs in order to not to let the inputs react on noise or bouncing of mechanical inputs. The debouncing value is given in time-units where every time-unit is equal to 31 usec. By default 7 time-units are set.</td>
<td>fastdebc=10 set the debounce-time to 310 usec</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| tune<sup>1</sup> | Enables special functions and features that are not activated by default. As parameter a number can be handed over that specifies the functions to be enabled. Following numbers can be concatenated by adding them: 1 – use DIn7 of Digi I/O Extension Board as external trigger, this disables ExtStart input of E1701 Baseboard and LP8 Extension Board 2 – use additional marking encoder inputs on DIn2 and DIn3 for 2D on-the-fly operations (requires firmware version 25 or newer) 4 – enable storage of serial number count values to microSD card; this option is useful in case of stand-alone operation mode when dynamic data with serial number counting is used. When it is set, the current count value of all used serial numbers is stored and reloaded on next power up. Thus their values are not get lost when power was turned off. The values are stored in a file with the same name like the "autofile" or the currently loaded .epr file but with extension ".ser". **ATTENTION:** The file is saved on the FatFS formatted microSD card. FatFS is NOT fault-proof, means it can be corrupted when power is turned off during writing. So when this option is enabled, user has to ensure power is NOT turned of while the card writes to disk. Writing of serial number states is always done in case they have changed, then it is started when state LED of E1701D board is switched off. Write operation is finished when this LED is turned back on the next time. So to ensure data are written successfully, it is recommended to let this LED blink two times after last mark operation has been finished. In automated environment this can be ensured by following procedure: 1. stop all marking operations 2. ensure no new marking operation is triggered 3. wait for 2 seconds 4. turn power off **ATTENTION:** due to this limitation it is not recommended to work with this option but to save the state of the serial numbers by sending ASCII command "cssta" instead (please refer below for details)! This value requires firmware version 12 or later. 8 – invert LaserGate output to work as active HIGH signal; when this option is set, logic of LaserGate-LED changes too, it is on as long as laser is turned off and it is off as long as laser is on (requires firmware version 16 or newer) 16 – invert LaserA output to work as active HIGH signal (requires firmware version 16 or newer) 32 – invert LaserB output to work as active HIGH signal (requires firmware version 16 or newer) 64 – use LaserA output as GPO (general purpose output pin); when this flag is set, LaserA output is no longer able to emit a frequency but can be used as digital output pin; when this value is set, a tune-value of 16 (invert LaserA) is ignored 128 – use LaserB output as GPO (general purpose output pin); | tune=1  
disables ExtStart input and switches over external trigger function to DIn7 input |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>sntp1</td>
<td>Allows to specify the IP of an SNTP time server. This option can be used in case of Ethernet usage to synchronise controller with an external time source. E1701D tries to connect to this server after initialisation of Ethernet interface and – if not successful – a few more times. These additional connection attempts are done whenever the state-LED is switched on. ATTENTION: when this function has to be used, the network or host-computer the controller is connected with needs to be able to route this request. This is a potentially dangerous operation because a connection between encapsulated machine network and open and dangerous Internet has to be established. Since this is NOT RECOMMENDED in general, this option should be used ONLY when it is 100% sure there is no possibility for people from outside to intrude the machine network! Instead of that it is also possible to contact an own, network-internal NTP-server. When this option is used, the gateway and netmask have to be configured for the controllers Ethernet interface This parameter requires firmware version 12 or later.</td>
<td>sntp1=83.170.1.42 – IP of time server at 3.de.pool.ntp.org is used for SNTP time retrieval (not recommended since this requires a connection to potentially dangerous Internet!)</td>
</tr>
<tr>
<td>sntp1offset</td>
<td>This value corresponds to sntp1 parameter above, it is used when system time is retrieved from an external time server to set an offset to the time returned from this server. The offset has to be specified in unit seconds. This parameter requires firmware version 12 or later.</td>
<td>sntp1offset=-3600 – specifies an offset of minus one hour to the time returned from timeserver. So when the time server would return a current time of 11:42:17, the system time of the controller would be set to 10:42:17 with this value</td>
</tr>
<tr>
<td>gw1</td>
<td>Specifies a gateway-address for the scanner controllers Ethernet interface. This option belongs to parameter &quot;ip1&quot; and has to be set in case &quot;sntp1&quot; is used. This parameter requires firmware version 12 or later.</td>
<td>gw1=192.168.2.1 – use 192.168.2.1 as gateway</td>
</tr>
<tr>
<td>nm1</td>
<td>Specifies the netmask for the scanner controllers Ethernet interface. This option belongs to parameter &quot;ip1&quot; and has to be set in case &quot;sntp1&quot; is used. This parameter requires firmware version 12 or later.</td>
<td>nm1=255.255.255.0 – use upper 24 bits of current IP for netmask</td>
</tr>
<tr>
<td>usb</td>
<td>When this parameter is set to 0, USB interface is disabled completely. This means it is no longer possible to connect to E1701 USB serial interface via terminal software or via BeamConstruct and it is also no longer possible to retrieve BeamConstruct PRO license via USB. This option can be used to suppress illegal access to USB and saves some power. This parameter requires firmware 21 or newer.</td>
<td>usb=0 – turn off USB interface</td>
</tr>
<tr>
<td>eth</td>
<td>When this parameter is set to 0, network interface is disabled completely. This means it is no longer possible to connect to E1701 via Telnet or via BeamConstruct. All SNTP-functionalities are disabled too. This option can be used to suppress illegal access to Ethernet, to save several seconds of startup-time and some power. This parameter requires firmware 21 or newer.</td>
<td>eth=0 – turn off Ethernet interface</td>
</tr>
</tbody>
</table>
**Parameter** | **Description** | **Example**
---|---|---
pethd | When Ethernet connection is used, it has to be established on power-up of the controller card as this connection is set-up and configured by the controller only once during boot. There may be situations where the other side of the Ethernet connection can not boot up as fast as E1701. In such cases this parameter can be used. It delays initialisation of Ethernet by the time given as parameter. The time is specified in unit “delayticks” where one “delaytick” is equal to about 0.5 seconds. As long as the controller is halted during initialisation due to this parameter, this is signalled by the Stop-LED (please refer to 6.1.5 User LEDs for details). This feature requires a firmware version 34 or newer. | peth=20 - halt initialisation of the controller for about 10 seconds prior to initialisation of Ethernet interface

### 6.1.8.1 Firmware Update

As described above the firmware is located on microSD-Card and therefore can be updated easily:

1. remove the microSD-Card as described above
2. download a new firmware from [https://openapc.com/download/Firmware/E1701](https://openapc.com/download/Firmware/E1701) (the higher the number in the file name, the newer the firmware is)
3. copy the contents of this ZIP-file to microSD-Card (please take care about e1701.cfg in case it contains a changed configuration)
4. reinsert microSD-Card as described in previous section

### 6.1.9 Laser/Scanner Signals

The white 26 pin connector provides several signals to be used to control up to three galvos of a scanhead and a laser source. It can be connected to an XY2-100 compatible scanner system via an adapter cable which splits XY2-100 and laser/input control signals. The connector is a white one to avoid confusion when a LP8 Extension Board is used too. The connector provides following signals:

<table>
<thead>
<tr>
<th>Upper Row Of Pins</th>
<th>Signal</th>
<th>Voltage</th>
<th>Remarks</th>
<th>Lower Row Of Pins</th>
<th>Signal</th>
<th>Voltage</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CLK-</td>
<td>XY2-100- / XY2-100-E-compatible signals</td>
<td></td>
<td>2</td>
<td>CLK+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SYNC-</td>
<td></td>
<td></td>
<td>4</td>
<td>SYNC+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>X-</td>
<td></td>
<td></td>
<td>6</td>
<td>X+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Y-</td>
<td></td>
<td></td>
<td>8</td>
<td>Y+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Z-</td>
<td></td>
<td></td>
<td>10</td>
<td>Z+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>LaserA</td>
<td>CMOS, 0/5V, max 14 mA</td>
<td>Laser control signals</td>
<td>12</td>
<td>GND</td>
<td>GND</td>
<td>Laser control signals</td>
</tr>
<tr>
<td>13</td>
<td>Laser Gate</td>
<td>CMOS, 0/5V, max 14 mA</td>
<td></td>
<td>14</td>
<td>GND</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>LaserB</td>
<td>CMOS, 0/5V, max 14 mA</td>
<td></td>
<td>16</td>
<td>ExtStart</td>
<td>CMOS, 0/5V</td>
<td>Input control signals</td>
</tr>
<tr>
<td>17</td>
<td>5V</td>
<td>5V</td>
<td></td>
<td>18</td>
<td>ExtStop</td>
<td>CMOS, 0/5V</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>do not connect</td>
<td></td>
<td>20</td>
<td>GND</td>
<td>GND</td>
<td>XY2-100- / XY2-100-E-compatible signals</td>
</tr>
<tr>
<td>21</td>
<td>GND</td>
<td>GND</td>
<td>XY2-100- / XY2-100-E-compatible signals</td>
<td>22</td>
<td>GND</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>do not connect</td>
<td></td>
<td>24</td>
<td></td>
<td></td>
<td>do not connect</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>do not connect</td>
<td></td>
<td>26</td>
<td></td>
<td></td>
<td>do not connect</td>
</tr>
</tbody>
</table>

Laser Gate provides laser modulation signal, turns on the laser during marks and off during jumps.
Laser A usage depends on software configuration and control. It is able to output a pulse-width modulated frequency (e.g. for controlling CO₂ lasers), CW/continuously running frequency (e.g. for fiber lasers) or Q-Switch signal (e.g. for YAG lasers) in range 25 Hz..20 MHz.

Laser B can be used for emitting a FPK pulse (e.g. for YAG lasers).

ExtStart expects a CMOS-level input signal in respect to GND and can be used as external trigger signal to start operations when a HIGH-signal is detected at input pin.

ExtStop expects a CMOS-level input signal in respect to GND and can be used as external stop-signal in order to stop a running marking operation by using a HIGH-signal at input pin.

### 6.1.9.1 XY2-100 Connection Cable

E1701D scanner controller board can operate an XY2-100-compatible scanner system directly. Here an adapter-cable is required that splits XY2-100 signals and additional laser/input control signals. Using such a cable white 26 pin connector described above should be converted to a female, XY2-100-compatible, 25 pin SUB-D connector (to connect with scanhead) and a female 9 pin SUB-D connector (which provides laser signals and start/stop inputs).

Pinout of a D-SUB25 XY2-100 connector should conform to standard with ground-signals (GND) at pins 11, 23 and 24:

```
1 - CLK-
2 - SYNC-
3 - X-
4 - Y-
5 - Z-
11 - GND
14 - CLK+
15 - SYNC+
16 - X+
17 - Y+
18 - Z+
23 - GND
24 - GND
```

XY2-100 status signals from scanhead are not used for E1701D.

Pinout of a limited D-SUB15 XY2-100 connector depends on used scanhead but typically looks like this:

```
1 - CLK-
2 - SYNC-
3 - X-
4 - Y-
5 - Z-
8 - GND
9 - CLK+
10 - SYNC+
11 - X+
12 - Y+
13 - Z+
15 - GND
```

XY2-100 status signals from scanhead are not used for E1701D.

Possible pinout of 9 pin SUB-D laser signal and control connector is shown below:
6.1.10 Extension Connectors

The two extension connectors on each side of the board can be used to place extension boards with additional peripheral interfaces. The extension connectors are designed to place/remove boards from time to time but they are not intended for constant hardware changes. So changing extension boards repeatedly and often e.g. as permanent part of a production process is not recommended.

PLEASE NOTE: when placing a new extension board
1. check correct orientation and position of the key pin which is closed in connector
2. place the pins of the extension boards onto the extension connectors exactly
3. move down the extension board by pressing on its extension connectors gently; DO NOT PRESS THE BOARD ITSELF BUT ONLY THE CONNECTORS!

PLEASE NOTE: When removing an extension board DO NOT pull on the extension connectors but hold both boards on their long side directly at the PCBs edges:
Due to the large number of pins, it is easy to plug in an extension but more difficult to pull it out. So when removing an extension board, it is recommended to be very slow and to carefully pull each side up just a little bit to avoid bending of the pins as they exit.

6.1.11 Stand-Alone Operation

E1701 scanner controller cards can be operated in stand-alone mode. In this mode all marking data are stored on SD-card and the board can operate without direct control of a host-PC that sends the data to be marked. Such stand-alone marking data can be created e.g. in BeamConstruct marking software.

6.1.11.1 Create Stand-Alone Data with BeamConstruct

To use BeamConstruct for generation of stand-alone data for E1701 scanner controllers, the card has to be fully configured (including all scanner, laser and pen-parameters). Next the marking data to be stored on SD-card have to be created. To generate stand-alone data, menu "Processing", sub-menu "Write Marking Data to File" or "Send Named Marking Data" has to be selected.

First one gives the possibility to write the data to microSD card when E1701 is switched off and the microSD card is plugged into host PC. Here it is recommended to use file extension ".EPR" for the file generated by BeamConstruct. Next it is also recommended to always let BeamConstruct write to microSD card directly because sometimes more than only one file is created. Direct write operation to BeamConstruct ensures all files are available on microSD and no data can be forgotten to be copied.

The second variant allows to download the stand-alone data to the controller while it is connected and running. Precondition for sending data to a running controller are:

• no mark operation is in progress (controller is idle)
• no stand-alone project is loaded (please refer to description of ioselect-mode and stand-alone control commands below).
• a valid name is given in style 0:/filename.epr

This operation creates the .EPR-file and all additional files on microSD card of the running controller automatically.

PLEASE NOTE: such an .EPR-stand-alone file can NOT be converted back to vector data that could be edited in BeamConstruct! Creating these files is a one-way-conversion of your projects. Thus it is recommended to save these projects twice – once as normal .BEAMP-File which can be loaded and modified later and once as .EPR-file which has to be used on SD-card. This also means such .EPR-files are protected so that it is possible to give away designs to some end-users which shall not be able to modify them.

E1701 controller supports all static data in stand-alone mode (like all kinds of static geometries, output signals, waiting for input commands, waiting for trigger, all laser- and scanner parameters). But it does not store the vector data using a possibly configured correction table! To get a valid correction for stand-alone operations, the related correction file has to be saved on microSD card and needs to be activated using parameter "corrtaleO" in e1701.cfg configuration file (please refer to description above).

Next E1701 scanner card supports dynamic content when following conditions are met:

• a text element uses one of the laser vector font families "Roman", "Script" or "Times" and it makes use
6.1.11.2 Stand-Alone Configuration Parameters

Within e1701.cfg configuration file of E1701 scanner controller one of the following stand-alone operation modes can be selected via the configuration parameter "standalone":

standalone=off

Stand-alone mode is fully disabled, the card acts as normal slave of a host-PC and all .epr-files on the SD-card are ignored. Digital outputs are not toggled since no stand-alone operational states have to be signalised here (please refer next section).

standalone=auto

Stand-alone mode is enabled, a file specified by and additional parameter "autofile" is loaded and prepared for marking. Marking of this file is started only when an external trigger signal is detected. The file itself has to be specified via additional configuration parameter that gives the filename of the stand-alone file to be loaded. As an example a parameter: "autofile=0:/myfile.epr" would try to load the file "myfile.epr" from SD-card and prepare it for marking. In this mode the digital outputs are toggled as described in next section.

standalone=loop

This is the same like mode "auto" described above, but using "loop" the E1701 controller does NOT wait for an external trigger signal! So when no trigger points are set in stand-alone datafile itself, in this mode marking would be done in an infinite loop, repeating the given "autofile" again and again.

standalone=haltedloop

This is the same like mode "loop" described above, but marking does not start immediately. By default the controller is in state "halt" until the ExtStart input is set to HIGH level. Marking continues only as long this input stays at HIGH. When it goes back to LOW, marking is continued until the laser is turned off the next time (defined by the geometry which is currently loaded) and it is halted again. Next time ExtStart goes to HIGH, marking continues at the position where it was halted before.

Please note: for this mode it is recommended to have marking data with vectors that are interrupted by jumps from time to time. When ExtStart goes to LOW, marking continues until the next jump is found. This means, when the controller processes a complex, long-lasting vector, marking will continue for the whole time this vector needs although ExtStart is already at LOW. So the laser is never interrupted and turned off within a mark operation.

This mode requires firmware version 27 or newer.

In this mode the timeout-parameter "haltedloopertimeout" is used.
This mode is a combination out of "haltedloop" described above and "ioselect" described below (please refer there for usage details). In this mode a project can be selected via digital inputs but it is started immediately and marked in an endless loop as long as ExtStart input is HIGH (so the level at ExtStart is checked, not the rising edge of an applied signal). When a different project is selected by applying a different input pattern at DIn digital inputs, the current project is cancelled and the new one is started in a loop again. This mode requires firmware version 28 or newer.

In this mode the timeout-parameter "haltedlooptimeout" is used.

This mode requires Digi I/O Extension Board (please refer below). Here it is possible to select one of 256 stand-alone marking jobs via the digital inputs. The number that results out of the input pattern of the Digi I/O input lines specifies the filename of the marking job that has to be loaded from SD card:

<table>
<thead>
<tr>
<th>Selected input(s)</th>
<th>Stand-alone file loaded from SD-card</th>
</tr>
</thead>
<tbody>
<tr>
<td>All inputs set to LOW (not recommended to be used)</td>
<td>0.epr</td>
</tr>
<tr>
<td>DIn0 set to HIGH</td>
<td>1.epr</td>
</tr>
<tr>
<td>DIn1 set to HIGH</td>
<td>2.epr</td>
</tr>
<tr>
<td>DIn0 and DIn1 set to HIGH</td>
<td>3.epr</td>
</tr>
<tr>
<td>DIn2 set to HIGH</td>
<td>4.epr</td>
</tr>
<tr>
<td>DIn0 and DIn2 set to HIGH</td>
<td>5.epr</td>
</tr>
<tr>
<td>DIn1 and DIn2 set to HIGH</td>
<td>6.epr</td>
</tr>
<tr>
<td>DIn0, DIn1 and DIn2 set to HIGH</td>
<td>7.epr</td>
</tr>
<tr>
<td>DIn3 set to HIGH</td>
<td>8.epr</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>DIn4 set to HIGH</td>
<td>16.epr</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>DIn5 set to HIGH</td>
<td>32.epr</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>DIn6 set to HIGH</td>
<td>64.epr</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>DIn7 set to HIGH</td>
<td>128.epr</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>All inputs set to high</td>
<td>255.epr</td>
</tr>
</tbody>
</table>

Please note: 0.epr (no inputs set to HIGH) can be used but it is not recommended to do that. This value should be reserved for "no job active" to set the card into an inactive mode also in stand-alone operational mode. This may be necessary e.g. when new project data are downloaded to the controller without removing the SD-card.

Marking of a IO-selected job is started by external trigger signal (ExtStart input). When the input pattern at DIn0..DIn7 changes during marking, the currently running operation is continued and the other stand-alone job is loaded after marking operation has finished. In this mode the digital outputs are toggled as described in next section.

In stand-alone mode "ioselect". EPR-files are loaded from microSD card as soon as a new input pattern is detected at digital inputs. Depending on the size of the EPR file and the speed of the microSD card, this may take a time that is too long for high-speed applications. Thus it is possible to operate such projects from
controller’s RAM completely:

- in e1701.cfg the numbers of the files to be loaded have to be specified with parameter “iobuff”, it can be used up to 20 times and expects the number of the file (so a line “iobuff=3” would be responsible for preloading file “0:/3.epr”), file “0.epr” can not be loaded by this command.

- serial interface/Telnet command “crlbf” can be used to reload such a predefined file during operation, e.g. in case it has been changed from outside (for details please refer to section “8.2 Stand-Alone Control Commands” below)

Now these files are loaded into RAM and switching from one to another is done much faster since toggling between them is done controller-internal and no more disk-operations are necessary for that.

PLEASE NOTE: when too much too large .EPR files are selected for preloading, this may exceed the available memory on card. This is signalled by the Error LED turned on and an appropriate message is stored in log buffer. In such a case a proper operation is no longer guaranteed.

The prebuffer-feature requires firmware version 27 or newer.

### 6.11.3 Stand-Alone Control

The current stand-alone operational state is signalled via digital outputs (requires Digi I/O Extension Board):

**DOut0** – ready for marking – this output goes to HIGH as soon as a stand-alone job could be found on disk, was loaded successfully and is ready for marking. So external start signal should not be given until this output is HIGH. When a new stand-alone file is selected (e.g. via digital inputs in "ioselect"-mode) this output goes to LOW. It is switched back to HIGH only when the new file could be loaded successfully too.

**DOut1** – marking active – as long as this output is HIGH, a marking operation is in progress. When a different stand-alone file is selected (e.g. via digital inputs) as long as this output is HIGH, marking is continued and the new file is NOT loaded. Once the current marking operation is completed, the output goes to LOW. After that the board continues with current marking data (when no new ones have been selected), or it tries to load new ones (when a new file was selected).

Please note: in firmware versions prior to v32 in stand-alone modes “haltedloop” and “iohaltedloop” this signal is not used since the user had full control over the marking process via the ExtStart input. So in case of these modes the logic is: when both, DOut0 and ExtStart are at HIGH, the controller is marking. Starting with firmware version 31 the DOut1 signal is used in same way as for all other stand-alone modes.
6.2 E1701A Analogue Laser Scanner Controller Baseboard

The E1701A Analogue Laser Scanner Controller Baseboard provides following connectors and interfaces:

1. Ethernet – for communication with the host system, marking information are submitted via this path
2. USB – via microUSB connector for providing BeamConstruct PRO license to host system and optionally for submitting marking data from host to E1701A card (in case Ethernet is not used)
3. Power – connect with power supply +- 15V DC
4. Power LED – lights when power is available
5. User LEDs – show operational and error states of card
6. Laser LED – shows modulation state of laser
7. microSD-card (on bottom side) – storage place for firmware and extended configuration file, can be used to upgrade firmware, to change the card’s IP and other things more
8. Laser/Scanner signals – white 26 pin laser and scanner output connector which provides analogue scanner signals as well as laser and marking control IOs
9. Extension connectors – extension boards can be placed below of E1701A Baseboard in order to add some more functionality and hardware interfaces to the board
10. DAC heatsinks

6.2.1 Ethernet

For a description of usage and configuration of Ethernet interface please refer to related section about E1701D's Ethernet interface above.

6.2.2 USB

This is a standard miniUSB-connector for connection of the board with the host system. It is used to retrieve BeamConstruct PRO license and optionally to send marking data to the card. When USB is used for sending all scanner and laser data, Ethernet cable does not need to be connected.

**PLEASE NOTE:** USB 2.0 is much slower than a standard 100 Mbit Ethernet connection, so expect slower execution in case of complex marking data!

Required device driver is installed together with OpenAPC-setup (Windows) or comes with operating system by default (Linux). E1701A card appears as COM-interface on Windows using any free number for the port. With Linux it appears as /dev/ttyACMx where “x” is any number. These numbers are provided by the operating system automatically.

**PLEASE NOTE:** this connection can not be used to power E1701A scanner controller. It may boot up and react on control commands from host PC but it will not emit any signals at analogue output connectors! A separate power supply via +-15V is mandatory in order to operate E1701A controller. For details about this power supply please refer to following section.
6.2.3 Power
Power supply for E1701A scanner controller board is done via 6 pin header on upper left corner of the board. Here pairs of pins belong to same power level. An appropriate fuse for circuit protection must be provided by the external equipment:

Power has to be supplied via this connector by connecting to a bipolar power supply with a maximum voltage of +15V DC, less than +/- 0.1V tolerance and 2.5A (stabilised and smoothed). Do not apply voltages in excess of 15V or with inverted polarity to this input. The DC power supply must be grounded.

To avoid the boards components heat up more than necessary, the power input can be limited:
• when XY outputs operate with +/-5V, voltage of +/-8V is enough for proper operation
• when XY outputs operate with +/-10V, voltage of +/-13V is enough for proper operation

Operating E1701A with a lower voltage than the maximum of +/-15V can not only help to keep the board cooler but also saves power during operation.

To avoid high frequency interferences from other electrical equipment or from within the power supply, it is recommended to place a ferrite bead at the cable close to the board. Please also check for correct shielding in respect to the equipment the E1701A card is used within.

6.2.4 Power LED
For a description of Power LED please refer to related section in description of E1701D controller above.

6.2.5 User LEDs
For a description of User LEDs please refer to related section in description of E1701D controller above.

6.2.6 Laser LED
For a description of Laser LEDs please refer to related section in description of E1701D controller above.

6.2.7 microSD-Card
For a description of microSD card, its usage, configuration possibilities, firmware updates and others please refer to related section in description of E1701D controller above.

6.2.8 Laser/Scanner Signals
The white 26 pin connector provides several signals to be used to control two galvos of a scanhead and a laser source. The connector is a white one to avoid confusion when a LP8 Extension Board is used too. The connector provides following signals:
## Upper Row Of Pins

<table>
<thead>
<tr>
<th>Signal</th>
<th>Voltage</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>X+</strong></td>
<td>±5V / ±10V, max 5 mA</td>
<td></td>
</tr>
<tr>
<td><strong>Y+</strong></td>
<td>±5V / ±10V, max 5 mA</td>
<td></td>
</tr>
<tr>
<td><strong>AOut1</strong></td>
<td>0.5V, max 10 mA</td>
<td></td>
</tr>
<tr>
<td><strong>Static Interlock</strong></td>
<td>Connected with pin 8</td>
<td></td>
</tr>
<tr>
<td><strong>AOut1</strong></td>
<td>0.5V, max 10 mA</td>
<td></td>
</tr>
<tr>
<td><strong>AOut2</strong></td>
<td>0.5V, max 10 mA</td>
<td></td>
</tr>
<tr>
<td><strong>AOut3</strong></td>
<td>0.5V, max 10 mA</td>
<td></td>
</tr>
<tr>
<td><strong>LaserGate</strong></td>
<td>CMOS, 0/5V, max 14 mA</td>
<td></td>
</tr>
<tr>
<td><strong>5V</strong></td>
<td>5V</td>
<td></td>
</tr>
<tr>
<td><strong>LaserGate</strong></td>
<td>CMOS, 0/5V, max 14 mA</td>
<td></td>
</tr>
</tbody>
</table>

## Lower Row Of Pins

<table>
<thead>
<tr>
<th>Signal</th>
<th>Voltage</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>X-</strong></td>
<td>±5V / ±10V, max 5 mA</td>
<td></td>
</tr>
<tr>
<td><strong>Y-</strong></td>
<td>±5V / ±10V, max 5 mA</td>
<td></td>
</tr>
<tr>
<td><strong>GND</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GND</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GND</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ExtStart</strong></td>
<td>CMOS, 0/5V, Input</td>
<td></td>
</tr>
<tr>
<td><strong>ExtStop</strong></td>
<td>CMOS, 0/5V, Input</td>
<td></td>
</tr>
<tr>
<td><strong>GND</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GND</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### LaserGate

LaserGate provides laser modulation signal, turns on the laser during marks and off during jumps. Outputs LaserGate at pin 15 and LaserGate at pin 25 are working fully parallel, so the maximum current that can be pulled out of both together is 14 mA.

### Analogue Outputs

Analogue outputs AOut1..AOut3 provide additional analogue control. Outputs AOut1 at pin 5 and AOut1 at pin 9 are working fully parallel, so the maximum current that can be pulled out of both together is 10 mA.

### ExtStart and ExtStop

ExtStart expects a CMOS-level input signal in respect to GND and can be used as external trigger signal to start operations when a HIGH-signal is detected at input pin.

ExtStop expects a CMOS-level input signal in respect to GND and can be used as external stop-signal in order to stop a running marking operation by using a HIGH-signal at input pin.

### Static Interlock

Static Interlock at pins 7 and 8 is available to be compatible with ILDA standard. Please note: These outputs are connected statically to each other, means there is no condition where connection between pin 7 and 8 is interrupted! Thus this connection can not be used for security locks in order to turn off the laser. Here additional, external security measures are necessary!

### 6.2.9 Extension Connectors

The two extension connectors on each side of the board can be used to place extension boards with additional peripheral interfaces between E1701A Baseboard and its underlying processor board. So comparing to extensions added to E1701D scanner controller (as described in detail above), E1701A baseboard always has to be placed on top. Thus for extending E1701A the baseboard has to be separated from its processor board (as described in detail in related section in E1701D description above), then extension board(s) has/have to be placed on processor board while E1701A baseboard afterwards has to complete this set and needs to be added on top.

For a more detailed description about how to unmount and mount the boards in order to add extensions, please refer to related section in E1701D description above.
PLEASE NOTE: when placing a new extension board or when adding E1701A board on top:
1. check correct orientation and position of the key pin which is closed in connector
2. place the pins of the extension boards onto the extension connectors exactly
3. move down the extension board by pressing on its extension connectors gently; DO NOT PRESS THE BOARD ITSELF BUT ONLY THE CONNECTORS!

6.2.10 DAC Heatsinks

The DAC heatsinks at the output lines of E1701A can become quite hot. Normal operation temperature can be up to 125°C without any damage to the electronics. But for integration of this board it is important to ensure
a) proper air flow and
b) nobody and nothing touches these heatsinks in order to avoid injury or other hardware to be damaged.
To limit temperature at these heatsinks, please refer to section regarding power supply of E1701A above.

6.2.11 Stand-Alone Operation

For a description of Stand-Alone-Operation, it possibilities, usage and configuration please refer to related section in description of E1701D controller above.
6.3 E1701 LP8 Extension Board

The E1701 LP8 Extension Board can be used with E1701D and E1701A baseboards and provides following features:

1. MO LED – shows state of Master Oscillator output
2. Laser signals – black 26 pin laser output connector which provides signals for controlling a laser
3. Extension connectors – more extension boards can be placed here in order to add some more functionality and hardware interfaces to the board, please refer to related section in description of baseboard above

6.3.1 MO LED

This LED is specific to the Master Oscillator output signal described below. As long as the signal is on (HIGH-signal at output pin), the LED is turned on.

6.3.2 Laser Signals

The black 26 pin connector provides several signals for controlling a laser source. It can be used e.g. together with YAG, CO₂, IPG™, fiber and compatible lasers since it provides additional signals and frequencies these laser types may require for proper operation. To avoid confusion with similar connector used on E1701D Base board this connector is black.

The connector provides following signals:
<table>
<thead>
<tr>
<th>Upper Row Of Pins</th>
<th>Signal</th>
<th>Voltage</th>
<th>Remarks</th>
<th>Lower Row Of Pins</th>
<th>Signal</th>
<th>Voltage</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LP8_0</td>
<td>CMOS, 0/5V, max 8 mA</td>
<td></td>
<td>2</td>
<td>GND</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>LP8_1</td>
<td>CMOS, 0/5V, max 8 mA</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>LP8_2</td>
<td>CMOS, 0/5V, max 8 mA</td>
<td></td>
<td>6</td>
<td>5V</td>
<td>5V</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>LP8_3</td>
<td>CMOS, 0/5V, max 8 mA</td>
<td></td>
<td>8</td>
<td>MO</td>
<td>CMOS, 0/5V, max 8 mA</td>
<td>Master Oscillator</td>
</tr>
<tr>
<td>9</td>
<td>LP8_4</td>
<td>CMOS, 0/5V, max 8 mA</td>
<td></td>
<td>10</td>
<td>AOut0</td>
<td>0..5V, max 15 mA</td>
<td>Analogue output</td>
</tr>
<tr>
<td>11</td>
<td>LP8_5</td>
<td>CMOS, 0/5V, max 8 mA</td>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>LP8_6</td>
<td>CMOS, 0/5V, max 8 mA</td>
<td></td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>LP8_7</td>
<td>CMOS, 0/5V, max 8 mA</td>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>LP8 Latch</td>
<td>CMOS, 0/5V, max 8 mA</td>
<td></td>
<td>18</td>
<td>5V</td>
<td>5V</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>LaserB</td>
<td>CMOS, 0/5V, max 14 mA</td>
<td>FPK</td>
<td>20</td>
<td>LaserA</td>
<td>CMOS, 0/5V, max 14 mA</td>
<td>PWM, frequency or Q-Switch</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>Connected to pin 20</td>
<td></td>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>GND</td>
<td>GND</td>
<td></td>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>5V</td>
<td>5V</td>
<td></td>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LP8_0...LP8_7 provide parallel 8 bit output signal (e.g. for power control with IPG(tm)/fiber lasers, waveform selection for SPI(tm) lasers and other).

LP8 Latch pin signals valid output at LP8_0..LP8_7 and AOOut0 by submitting a latch pulse of software-controlled length.

MO can be used to enable master oscillator (e.g. for IPG(tm)/fiber lasers or compatible).

LaserA usage depends on software configuration and control, it is able to output a pulse-width modulated frequency (e.g. for controlling CO₂ lasers), CW/continuously running frequency (e.g. for fiber lasers) or Q-Switch signal (e.g. for YAG lasers) in range 25 Hz..20 MHz.

LaserB can be used for emitting a FPK pulse (e.g. for YAG lasers).

AOOut0 pin provides unipolar analogue output for controlling e.g. laser power or additional equipment. This output is a slave of LP8_0..LP8_7 output, they are electrically connected and therefore can’t have different values and can’t be controlled by software independently. So when LP8 outputs are all LOW, AOOut0 is on 0V. When LP8 outputs are all HIGH, AOOut0 is 5V.

PLEASE NOTE: output of 5V at AOOut0 depends on the used power supply. So in case board is powered via USB and USB power supply delivers less than 5V, maximum output on AOOut0 will be less than 5V too. Here is would be recommended to use the base board with an external power supply that feeds exactly 5V into it.

### 6.3.3 Extension Connectors

The two extension connectors on each side of the board can be used to place extension boards with additional peripheral interfaces. For a description of handling and usage of these connectors please refer above.

---

2 requires hardware-revision 1.1 or newer
6.4 E1701 Digi I/O Extension Board

The E1701 Digi I/O Extension Board can be used with E1701D and E1701A baseboards and provides following features:

1. Digi I/O – electrically insulated digital in- and outputs
2. optional inputs for 90 degree phase shifted encoders to be used with marking on-the-fly operations
3. Opto-Configuration – choose operation mode for Digi I/Os
4. Input state LEDs – displaying of HIGH/LOW state of used inputs

In case more extension boards are used on E1701D, Digi I/O extension always has to be placed on top. For E1701A a specific variant of the Digi I/O extension has to be used which can be stacked and placed below of the E1701A baseboard which always needs to stay on top.

6.4.1 Digi I/O

The 20 pin connector provides 8 lines for input and 8 lines for output of digital signals that can work on CMOS level (non-insulated mode) or via opto-couplers (electrically insulated mode with external power supply) optionally. The operation mode depends on jumper settings described below. The connector is used as follows:
<table>
<thead>
<tr>
<th>Upper Row Of Pins</th>
<th>Signal</th>
<th>Voltage</th>
<th>Remarks</th>
<th>Lower Row Of Pins</th>
<th>Signal</th>
<th>Voltage</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$V_{ext}$</td>
<td>5..24V</td>
<td>Input voltage to be used in opto-insulated mode only</td>
<td>2</td>
<td>$GND_{ext}$</td>
<td>GND</td>
<td>External ground</td>
</tr>
<tr>
<td>3</td>
<td>DOut0</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td>Default level: LOW</td>
<td>4</td>
<td>DIn0</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td>Encoder-input A1 for marking on-the-fly</td>
</tr>
<tr>
<td>5</td>
<td>DOut1</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td>Default level: LOW</td>
<td>6</td>
<td>DIn1</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td>Encoder-input B1 for marking on-the-fly</td>
</tr>
<tr>
<td>7</td>
<td>DOut2</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td>Default level: LOW</td>
<td>8</td>
<td>DIn2</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td>Second encoder-input A2 for marking on-the-fly</td>
</tr>
<tr>
<td>9</td>
<td>DOut3</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td>Default level: LOW</td>
<td>10</td>
<td>DIn3</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td>Second encoder-input B2 for marking on-the-fly</td>
</tr>
<tr>
<td>11</td>
<td>DOut4</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td>Default level: HIGH</td>
<td>12</td>
<td>DIn4</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>DOut5</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td>Default level: HIGH</td>
<td>14</td>
<td>DIn5</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>DOut6</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td>Default level: HIGH</td>
<td>16</td>
<td>DIn6</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>DOut7</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td>Default level: HIGH</td>
<td>18</td>
<td>DIn7</td>
<td>CMOS, 0/5V or 0/$V_{ext}$</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>V</td>
<td>5V</td>
<td>Board voltage, to be used only when not operating in insulated mode</td>
<td>20</td>
<td>GND</td>
<td>GND</td>
<td>Board-internal ground</td>
</tr>
</tbody>
</table>

$V_{ext}$ and $GND_{ext}$ depend on opto-configuration as described below. In opto-insulated mode (opto-configuration jumpers not set) external power supply has to be connected to these inputs. Then DIn0..DIn7 and DOut0..DOut7 work in respect to this external power.

**WARNING:** When no opto-insulated mode is selected (opto-configuration jumpers are set), do NOT FEED ANY POWER into $V_{ext}$, this would cause damage to the E1701 board! In this case $V_{ext}$ is equal to V (5V) of the board and $GND_{ext}$ is connected to boards ground GND.

Maximum current for every output is 15 mA when internally powered (non-insulated mode), here it is recommended to use an external power supply.

Maximum current for outputs DOut0..DOut3 is 50 mA when externally powered ($V_{ext}$ in insulated mode).

Signal output lines DOut0..DOut7 operate in open collector mode and have to be wired as follows:
Here “DOutx” symbolises one of the digital outputs DOut0..DOut7. V+ is either V (5V internal, non-insulated mode) or \( V_{\text{ext}} \) (up to 24V external, insulated mode). GND is either GND (non-insulated mode) or \( GND_{\text{ext}} \) (insulated mode). The internal resistor of the connected device is not allowed to have less than 490 Ohms in order to not exceed the given current limits.

DOut0..DOut3 provide LOW signal level by default, DOut4..DOut7 provide HIGH level by default. These levels are valid immediately on power-up of the card.

### 6.4.1.1 Marking On-The-Fly Signals

Digital inputs 0 and 1 can be used as position encoder signal inputs for marking on-the-fly applications. Here 90 degree phase-shifted input pulses are expected signalling motion direction and position change:

When these pulses are generated from a motion stage that moves the working piece, the resulting position information is used in marking on-the-fly mode to correct the marking positions accordingly. Resulting from that, marking will follow motion as far as available scanhead range and working area allows it. The pulses generated out of the encoder signals have to be multiplied with a factor reflecting the resolution of the used encoder. To set up and adjust a marking on-the-fly-system properly, following steps have to be performed:

1. Connect encoder signals A1 and B1 to DigIn0 and DigIn1 and configure E1701 controller for encoder usage (either from within ControlRoom/BeamConstruct or via programming interface as described below)
2. Mark a square without any encoder signals feed into the controller
3. When the square does not have exact size and/or is distorted, modify correction table and/or gain settings
4. Mark the same square with a slow motion (using encoder pulses)
5. When the square is damaged (means open on one side or compressed) the on-the-fly-factor has to be changed (set to a smaller or higher value)
6. Mark the same square with a fast motion (using encoder pulses)
7. When the square is damaged (means open on one side or compressed) the on-the-fly-factor has to be changed (set to a smaller or higher value)

The on-the-fly-factor controls the strength of compensation and is the relation between speed of external device/encoder pulses and card-internal compensation calculation. When this factor is wrong, the marking results are distorted. For a square (as recommended to be used in calibration steps above) following results are imaginable:

![Diagram of over-compensated and under-compensated systems]

The left drawing shows an over-compensated system, here the internal compensation is too strong, the factor is too big. The right drawing shows an under-compensated set-up, here the factor is too small causing a too weak compensation. Only when marking result is really a square, the on-the-fly-factor is correct.

When "tune"-flag 2 is set, a second encoder can be used for 2D marking on-the-fly applications (requires firmware 25 or newer). In this mode digital inputs 0 and 1 (encoder inputs A1 and B1) correspond to X axis and on-the-fly factor for X direction. Additionally digital inputs 2 and 3 (encoder inputs A2 and B2) correspond to Y axis and on-the-fly factor for Y direction. Operation principle is the same as for 1D on-the-fly described above: the incremental values received from the encoders for X and Y are added to the current X and Y coordinates to be marked. Procedure for adjusting the encoder factor is also the same, here it is recommended to perform this operation for X and Y movements separately and finally try both motion directions together.

**6.4.2 Opto-Configuration**

Using these jumpers the operation mode for digital I/Os 0..7 can be chosen. When they are set, the opto-couplers are powered internally. In this mode it is not working in opto-insulated mode and I/Os are using CMOS level signals.

When they are not set, external power and ground has to be provided at 20 pin connector (as described above) and these digital I/Os are working in electrically insulated, opto-coupled mode.

**6.4.3 Input State LEDs**

These 8 yellow LEDs show the state of corresponding 8 digital inputs. As long as a HIGH signal is detected on an input, the related LED is turned on.
6.5 E1701 Secondary Head Extension Board

The E1701 Secondary Head Extension Board can be used up to three times with the same E1701D Baseboard to have a scanner controller system with up to four heads connected. These additional scanheads work fully parallel to the primary scanhead of E1701D baseboard.

PLEASE NOTE: This board can not be used together with E1701A baseboard!

It provides following features:

1. XY2-100 signals for scanhead
2. Extension connectors – more extension boards can be placed here in order to add some more functionality and hardware interfaces to the board, please refer to related section in description of baseboard above

6.5.1 Scanner Signals

The white 26 pin connector provides XY2-100-compliant signals to be used to control to three galvos of a scanhead. It can be connected to an XY2-100 compatible scanner system directly and without further modifications when a 1:1 connection to a D-SUB25 connector is used. The connector is a white one to avoid confusion when a LP8 Extension Board is used too.

The connector provides following signals:
Comparing to E1701D Baseboard this connector does not provide any laser signals (like LaserGate, LaserA or LaserB). Since this scanner output works completely parallel to the one from E1701D Baseboard it has to be used together with the same laser using beam splitters.

### 6.5.2 Extension Connectors

The two extension connectors on each side of the board can be used to place extension boards with additional peripheral interfaces. For a description of handling and usage of these connectors please refer above.
7 Quick Start into E1701A/D

Following a few steps are described that give users the possibility to quick start into usage of E1701A/D scanner controller. It makes use of BeamConstruct and the (slow) USB connection. For this quick start manual it is assumed correct wiring of the controller is already done according to the description above. For more detailed information about BeamConstruct usage please also refer to quick start manual from https://openapc.com/download/manual_quickstart.pdf and to full user manual which is available at https://openapc.com/download/manual.pdf.

To start with E1701A/D controller:

1. **SECURITY CHECK:** The following steps describe how to set up E1701A/D scanner controller card and how to control laser equipment with them. Thus all laser safety rules and regulations need to be respected, all required technical security mechanisms need to be available and active prior to starting with it.
2. Install latest software version from https://openapc.com/download.php – for Windows this package contains all required drivers, for Linux no separate drivers are needed.
3. Connect E1701A/D controller via USB. For E1701A external +-15V power supply is required (as described above).
4. Now the Alive-LED should light up and then start blinking after some time. When this does not happen, please turn power off, check if the microSD-card is placed correctly and then try again.
5. Evaluate the serial interface the controller is connected with – for Windows the Device Manager (can be found in Control Panel) will list a new COM-port (e.g. “COM3”); for Linux type “dmesg” in console to find out to which interface it was connected with (typically “/dev/ttyACM0”).
7. Go to menu “Project” → “Project Settings...”, then tab-pane “Scanner”.
8. Now you can either select “E1701A” as scanner controller card or can go with “E1701D” which is selected and preconfigured by default. If none of both is selected, please choose the appropriate scanner controller in the related combo box.
9. Press the “Configure”-button to get into the settings dialogue for E1701A/D plug-in.
10. Enter the serial interface name in field “IP/Interface” (e.g. “COM3” or “/dev/ttyACM0”).
11. Leave everything with “OK”.
12. Draw some geometries as described in "BeamConstruct Quick Start Manual".
13. **SECURITY CHECK:** Next the scanner controller card will be accessed for the first time. That means it is opened and initialised and all connected equipment may start working now. Thus it is very important to ensure all security regulations are met and nobody can be injured and no damage can be caused also in case laser output or other motion starts spontaneously and unexpectedly!
14. Press “F2” or go to menu “Process” → “Mark” to open the mark dialogue.
15. Start marking by pressing the yellow button with the laser-symbol.
8 Command Interface

The commands described in this section require firmware version 12 or later. When E1701 scanner card is connected via USB and the USB-connection is NOT used for transmitting marking information, it can be used to send control commands to the card. Some of them are independent of the current operating mode and some of them can be used only in case the controller is operating in stand-alone mode. Starting with firmware version 15 control commands can also be sent via Telnet using Ethernet connection. Here a Telnet-client has to connect to port 23 using the IP of the scanner controller. This Telnet client should work in passive mode. So when E1701 scanner card is connected via Ethernet and the Ethernet-connection is NOT used for transmitting marking information, it can be used to send control commands to the card. Some of them are independent of the current operating mode and some of them can be used only in case the controller is operating in stand-alone mode. Such a control command always consists of ASCII-text. An appropriate client has to connect to the serial port (COMx for Windows and /dev/ttyACMx for Linux where “x” is a number identifying the specific serial interface or TCP/IP port 23). As soon as the connection is established, commands can be sent to the card. All commands come with following structure:

cxxxx <parameter(s)>

The commands always start with character "c". Next four characters identify the command itself. Depending on the command one or more optional or mandatory parameters may follow. The command always returns with an “OK” or with an error.

8.1 General Commands

The following commands can be used in all scenarios, they do not depend on a specific operation mode of the card. Nevertheless it is recommended to not to send commands excessively during card is marking, to not to influence marking operation.

cvers

"version" – return version information of controller card. This command returns a version string specifying version of hard- and software.

cecho <0/1>

"echo" – when typing commands in a serial console communicating with the controller, all the typed characters are echoed, means they are sent back to the host so that a user can see what is typed. This may be an unwanted behaviour when an application communicates with this interface. Using this command the serial echo mode can be turned off (parameter 0, only return values are sent back) or on (parameter 1, all data are sent back). When called with no parameters, the current echo mode value is returned.

Example: cecho 0 – turn off echo mode

cginp

"get input" – get the current state of the digital inputs (in case a Digi I/O extension is available). The input state is returned as a decimal number representing the bitpattern at the inputs. So when e.g. a value “15” is returned, this means the lower four inputs are set to HIGH while the upper ones are at LOW level

csout <value>

"set output" – set the state of the digital outputs (in case a Digi I/O extension is available). The output to be set is specified as a decimal number representing the bitpattern. When no parameter is given, the behaviour is undefined.

Example: csout 128 – set DOut7 at the Digi I/O extension board to HIGH while all others stay at LOW
cgbds
"get boards" – get an identifier value for the connected boards. This command returns a decimal number which depends on the connected extension boards and can be used to identify them. The returned value is a sum consisting of the following numbers:
1 – XY2-100 baseboard (E1701D) is available
2 – analogue baseboard (E1701A) is available
256 – LP8 Extension Board is available
512 – Digi I/O Extension Board is available

cglog
"get logline" – returns a single logging line. This command has to be called repeatedly until an error is returned to get logging information from the controller. On each call of this function one logging line is returned. When "cglog" isn't used for a longer time it may be possible the internal log-buffer has overrun. In this case "cglog" will not return all log information, previous log data may be overwritten.

cgsr
"get board serial number" – returns the serial number of the card. This number is an unique, internal value that is used e.g. to identify a controller on host PC when more than one scanner card is used.

8.2 Stand-Alone Control Commands

Following commands are useful in case scanner controller is operating in stand-alone mode where marking data are loaded from microSD-card using special EPR-fileformat.

cstop
"stop" – stop marking as fast as possible. A running marking operation is stopped and LaserGate is turned off.

chalt <0/1>
"halt" – halts or continues the processing and output of marking data. When given parameter is equal to 1, marking is stopped next time the laser is off but no vector data are dropped. On continue (parameter equal 0) controller continues processing at the point where halt occurred. When marking is stopped with cstop the halt-condition is cleared too, means on next transmission of new marking data they are processed without the need to explicitly continue operation by calling chalt 0.

cstrt
"start" – start marking operation. This command can be called only when no marking operation is running and when a valid project (.epr) file was loaded. In this case the currently loaded project is marked once.

ctrig
"trigger" – send an external trigger signal by software. When scanner card is in state "marking" but waiting for an external trigger, this command releases this trigger. So behaviour is the same like a rising edge on the ExtStart input of the controller card.

cstat
"state" – return the current state of the card. This command returns one of the following texts identifying the operational state:
• marking – card is processing some marking data currently, means either actively outputting them or waiting for an external trigger to start marking
• stand-alone – controller is in stand-alone mode
• idle – card is waiting and not marking
• waiting – a project file was loaded, is ready for execution and waits for a trigger signal (either via
ExtStart input or via command "ctrig"

cgtin

"get trigger inputs" – get the state of the external input signals. This command is not related to digital inputs of Digi I/O extension board but provides information regarding signal state of external start and stop. It returns a value that specifies which of these input signals are currently HIGH:
0 – ExtStart and ExtStop are both LOW
2 – ExtStart is HIGH
4 – ExtStop is HIGH
6 – ExtStart and ExtStop are both HIGH

cscor <idx>

"set correction" - specifies a new index for a previously loaded correction file (see description of configuration parameter corrtable in section ”6.1.8 microSD-Card” above). The parameter idx can be a value in range 0..15 and needs to correspond to a previously loaded correction table. The newly set correction table applies to all vector data which are processed after this call. Thus it is recommended to use it only when marking operation was stopped – elsewhere it is not predictable how many vector data already have been pre-calculated with the previous correction table and starting with which vector data the new correction file is used.
When a idx-value is set which corresponds to no correction file data, no more correction is performed on vector data.
This command requires firmware version 33 or newer.

cgcor

"get correction" - this command is the counterpart of cscor and displays the index number of the currently used correction file
This command requires firmware version 33 or newer.

clepr <path>

"load epr" – loads an EPR stand-alone file from microSD card for outputting it on next marking operation. This command can be executed in stand-alone mode only and expects the path to the file to be loaded as parameter. Since this is the only parameter, no quotes are allowed for the pathname. The pathname itself has to be in format
0:/filename.epr
where 0:/ specifies the microSD-card and .epr is the standard extension of E1701 stand-alone marking data files (this name is a shortcut for "E1701 Processing Data"). During loading the ready-for-marking output signal is turned off and it is turned on only in case the file could be loaded successful (please refer to related section above).
Examples: clepr 0:/test.epr – loads a stand-alone file "test.epr" from microSD card

crlbf <number>

"reload buffered file" - this command can be used in stand-alone mode "ioselect" and for all files that are configured for preloading using configuration parameter "iobuff". When called without any parameter it reloads all files that are configured for prebuffering, when called with a number as parameter it reloads the file with the given number. This command can be useful in cases where a prebuffered .EPR file was changed on disk (e.g. by downloading it via controller to microSD card) and has to be loaded into the RAM without rebooting the card.
This command requires firmware version 27 or newer.

cgepr

"get epr" - returns the name of the currently loaded stand-alone file or an error "no file specified" when no file is loaded.

cjsor <percentage>

44
"jump speed override" – changes the speeds of all jump speed values by the given factor. Here parameter percentage has to be given in unit 1/100%. The override-value specified by this command remains active until it is set back to normal value by calling "cjsor 100000" or until the controller is rebooted. The value given here is active for all processed data including host-controlled marking projects and stand-alone files loaded from microSD card. This command requires firmware version 25 or newer.

cmsor <percentage>

"mark speed override" – changes the speeds of all mark speed values by the given factor. Here parameter percentage has to be given in unit 1/100%. The override-value specified by this command remains active until it is set back to normal value by calling "cmsor 100000" or until the controller is rebooted. The value given here is active for all processed data including host-controlled marking projects and stand-alone files loaded from microSD card. This command requires firmware version 25 or newer.

cpwor <factor>

"power overwrite" - this command modifies the actual power by using the given factor (in unit 1/10000). All operations make use of the changed jump speed until a factor of 10000 is set or until the controller is restarted. This is true for both, stand-alone applications where an .EPR-file is loaded from microSD-card and for host-controlled marking operations (via libE1803D or BeamConstruct). This command influences following methods of setting laser power:

- pulse width, here user has to ensure the resulting pulse width is smaller than the period of the related frequency, elsewhere the output will be a continuous signal
- LP8 laser port
- AOut0 and AOut1 analogue outputs

cstxt <"elementname"> <"text">

"set text" – set a new text value to an element in currently loaded project. The parameters for this command both have to be given with quotes ("."). Setting a text is possible only for dynamic elements like Datamatrix barcodes or texts. Here "elementname" is the name of the element that has to be modified (this is the same name like shown in element tree of BeamConstruct) and the new text to be set. The "text" itself can be a format string as used within BeamConstruct when a serial number input element is involved
Example: cstxt "Barcode 1" "Hello :-)" - sets a new text "Hello :-)" for the element with name "Barcode 1"

cgtxt <"elementname">

"get text" – gets the currently used text value of an element in loaded project. The parameter for this command has to be given with quotes ("."). Getting a text is possible only for dynamic elements like Datamatrix barcodes or texts.
Example: cgtxt "Barcode 1" – gets the text from the element with name "Barcode 1"

csser <"elementname"> <cnt>

"set serial number" – sets a new serial count value to an element in currently loaded project. The element name for this command has to be given with quotes ("."). Setting a new count is possible only for dynamic elements like Datamatrix barcodes or texts that have a serial number input element assigned. Setting the value has to be handled with care, here every value can be specified independent if it fits to possibly exiting beat count values.
Example: csser "Text 2" 42 – set a new serial number count value 42 for element with the name "Text 2"

cgser <"elementname">

"get serial number" – gets the current serial count value from an element in loaded project. The element name for this command has to be given with quotes ("."). Getting the count is possible only for dynamic
elements like Datamatrix barcodes or texts that have a serial number input element assigned.

ciser <"elementname">

"increment serial number" – increments the current serial count value of an element according to its serial number parameters. The element name for this command has to be given with quotes ("."). Incrementing the count is possible only for dynamic elements like Datamatrix barcodes or texts that have a serial number input element assigned. This function is more secure than forced setting of a new count value with "csser" since it can't violate the counting rules.

cdser <"elementname">

"decrement serial number" – decrements the current serial count value of an element according to its serial number parameters. The element name for this command has to be given with quotes ("."). Decrementing the count is possible only for dynamic elements like Datamatrix barcodes or texts that have a serial number input element assigned. This function is more secure than forced setting of a new count value with "csser" since it can't violate the serial number counting rules.

crser <"elementname">

"reset serial number" – resets the current serial count value of an element to its start-value (according to its serial number parameters). The element name for this command has to be given with quotes ("."). Resetting the count is possible only for dynamic elements like Datamatrix barcodes or texts that have a serial number input element assigned. This function is more secure than forced setting of a value with "csser" since it can't violate the predefined serial number parameters and automatically uses the correct reset value.

cstim <seconds>

"set time" – this command sets the system time to the value specified with the parameter. Here the number of seconds have to be specified that have elapsed since 01.01.1970 at 00:00:00. After sending this command the controller card operates at the given time. The time value is lost after next power cycle and has to be set again.
Example: cstim 1420113600 – set the internal time of E1701 controller to 01.01.2015 12:00:00, here 1420113600 represents the number of seconds that have been elapsed between 01.01.1970 00:00:00 and 01.01.2015 12:00:00

crtim

"retrieve time" – this command schedules time retrieval from an SNTP server asynchronously. It always returns with "OK" since the command is scheduled for execution during next working cycles. To use this command, controller has to be configured with IP, netmask, gateway and SNTP server IP correctly and needs to be able to access this SNTP server from its position in network. For details please refer to description of configuration parameters in section about microSD card above.

cgtime

"get time" – returns the current time of the board in number of seconds that have elapsed since 01.01.1970 at 00:00:00. After powering up the board and before a valid time has been set, this value is undefined.

cftime

"get formatted time" – returns the current time of the board as formatted string in style DD.MM.YYYY hh:mm:ss. After powering up the board and before a valid time has been set, this value is undefined.

cstyr <year>

"set time year" – sets the year of the current system time to the value given as parameter. This value has to be in range 1900..2038

cstmo <month>
"set time month" – sets the month of the current system time to the value given as parameter. This value has to be in range 1..12 according to the number of the month.

cstdy <day>
"set time day" – sets the day of the current system time to the value given as parameter. This value has to be in range 1..28, 1..30 or 1..31 according to the length of the current month. To avoid invalid combinations it is recommended to set the month (using command cstmo) before setting the day.

csthr <hour>
"set time hour" – sets the hour of the current system time to the value given as parameter. This value has to be in range 0..23.

cstmi <minute>
"set time minute" – sets the minute of the current system time to the value given as parameter. This value has to be in range 0..59.

cstsc <second>
"set time second" – sets the second of the current system time to the value given as parameter. This value has to be in range 0..59.

cgsta
"get serial state" – this command applies only when working in stand-alone mode with dynamic serial number data that change on every mark operation. It returns information if the state of serial numbers has changed and is not yet saved (in this case "pending" is returned) or if they have been saved and therefore do not get lost when power is turned off now ("saved" is returned in this case).

cssta
"save serial state" - this command applies only when working in stand-alone mode with dynamic serial number data that change on every mark operation. When it is called, a command to save the current state of serial numbers is enqueued and will be processed as soon as controller is able to store these data. So when this command returns with "OK" that doesn't necessarily means the serial number states are saved now. The current save state still has to be checked by calling cgsta after cssta has been issued.

crrrr
"reboot" – perform a warm reboot of the hardware and restart the firmware. Reboot is done immediately, means this command does not return anything but connection to the board will be interrupted as soon as it has been sent. This command requires firmware version 19 or newer.

8.3 Mark Control Commands

The following section describes commands that can be used to send marking data (including vector data and laser/scanner parameters) to the controller. They are available with all firmware version 15 or newer. These commands can be mixed with the commands described above but have a different structure:

- they always start with a character "d"
- the total length of one frame (means one command) is always 14 bytes
- they mustn't be terminated with CR/LF, the end of a frame is determined by its size of 14 bytes
- they contain binary, means not human-readable data and therefore can't be sent manually

PLEASE NOTE: when using Network/Telnet connection and when switching from a Mark Control Command ("d"-command) to a general command ("c"-command as described above) it is recommended to completely
transmit all preceding output before sending a command of other type. These commands always have the following structure:

dCAAAABBBBEEEE

d – marks starting point of a frame and identifies a mark control command with a fixed length of 14 bytes (including this character)
C – 8 bit value that specifies what command has to be executed
AAAA – 32 bit little-endian value, it's meaning and usage depends on "C"
BBBB – 32 bit little-endian value, it's meaning and usage depends on "C"
EEEE – 32 bit little-endian value, it's meaning and usage depends on "C"

It is recommended to collect commands before they are sent to the controller, especially in case Ethernet connection is used. In case of TCP/IP the used payload length of a TCP-frame is 1460 bytes which should be filled as much as possible in order to avoid additional data transfers. So when sending larger amounts of data to the controller, up to 104 command frames should be collected and then sent all together (104 * 14 = 1456 bytes which is close to 1460).

From time to time the controller sends back an answer to give back some state information. In case of Ethernet/Telnet connection this answer is not sent periodically but as response to a complete block of data sent to the card. Since the size of such a block is not specified and depends on the underlying TCP/IP implementation (in case of Ethernet connections), no predictions can be made after what amount of data a response frame is sent. Thus it is recommended to try to receive such a response frame every time some data have been transmitted until at least one frame was received. When host software is idle, it can try to read response frames permanently. To trigger transmission of a new response frame, "ping" control command 0x0A can be used (for details please refer below).

In case of USB/serial connection this response is sent automatically after every 14 byte frame submitted, so it is necessary to always read them in order to avoid overrun of receive buffers. Such a response frame gives back information about the current operational state of the card and comes in following structure:

dRLLLLSSSSIIII

d – marks starting point of a response frame with a fixed length of 14 bytes (including this character), this character can be used to re-synchronise
R – 8 bit value, currently always 0xFF; this value has to be checked for future compatibility, in case it is not 0xFF the frame has to be ignored!
LLLL – 32 bit little-endian value, here the amount of free command buffer space is returned; sending application has to ensure this buffer never overruns, so it is recommended to always leave a space of at least 200 commands (recommended: 1000), new commands should be sent only when there is more space than this left in this buffer
SSSS – 32 bit little-endian value, signalling operational state; this value can consist of following or-concatenated flags:
  • 0x00000001 – card is currently marking
  • 0x00000002 – the external start input is currently HIGH
  • 0x00000004 – the external stop input is currently HIGH
  • 0x00000008 – the external start input was set to HIGH after last response frame, this value is set only once for every rising edge on this input
  • 0x00000010 – the external stop input was set to HIGH after last response frame, this value is set only once for every rising edge on this input
  • 0x00000080 – the controller has received some data which may result in a marking operation; these data are currently processed but marking has not yet started
  • 0x00000400 – card is active but currently waiting for an external trigger to continue operation
  • 0x00000400 – card is active, writing some data to the microSD card
IIII – 32 bit little-endian value, lower 8 bit show the actual state of digital inputs (in case Digi I/O Extension Board is available), the upper 24 bits are reserved for future use.
Currently following mark control commands (identified by the 8 bit hexadecimal value for position "C" in a frame) can be sent to the controller:

Jump to Position
Move to a given coordinate position using the current jump speed and with laser turned off
\[ C = 0x00 \]
AAAA = x-position to move to in range 0..67108863
BBBB = y-position to move to in range 0..67108863
EEEE = z-position to move to in range 0..67108863

Mark to Position
Move to a given coordinate position using the current mark speed and with laser turned on
\[ C = 0x01 \]
AAAA = x-position to move to in range 0..67108863
BBBB = y-position to move to in range 0..67108863
EEEE = z-position to move to in range 0..67108863

Start output
This command has to be called at the end of every marking sequence to ensure marking output really starts. This is important in case only a few vectors are sent to ensure marking is started but it is recommended to always use this command.
\[ C = 0x02 \]
AAAA = unused, set to 0
BBBB = unused, set to 0
EEEE = unused, set to 0

Wait for external trigger
Set a trigger point to current position of stream; emission of output data will stop until an external trigger signal is detected
\[ C = 0x03 \]
AAAA = unused, set to 0
BBBB = unused, set to 0
EEEE = unused, set to 0

Set speed values
Specify the speeds to be used during jump or mark movements (invoked by commands 0x00 and 0x01)
\[ C = 0x04 \]
AAAA = jumpspeed in unit bits per microsecond
BBBB = markspeed in unit bits per microsecond
EEEE = unused, set to 0

Set laser delays
Specify the delays to be used when laser is turned on or off
\[ C = 0x05 \]
AAAA = laser on delay in unit microseconds and in range -10000000..10000000
BBBB = laser off delay in unit microseconds and in range 0..10000000
EEEE = unused, set to 0

Set scanner delays
Specify the delays to be used before and after mark and within a polygon
\[ C = 0x06 \]
AAAA = jumpdelay in unit microseconds
BBBB = markdelay in unit microseconds
EEE = in-polygon delay in unit microseconds

Stop marking
Tries to halt, continue or stop current output depending on the chosen option
C = 0x07
AAAA = stop option:
0 - tries to stop operation as fast as possible and rejects all data that still may be enqueued for execution
1 - marking is stopped next time the laser is off but no vector data are flushed, card is still active
2 - controller continues processing at the point where halt occurred (requires a previously called command 0x07 with stop option 1)
BBBB = unused, set to 0
EEE = unused, set to 0

Set wobble parameters
Specify the wobble settings to be used for next marking operations
C = 0x08
AAAA = wobble amplitude in X-direction using unit bits and with maximum range of 0..10000000 bits
BBBB = wobble amplitude in Y-direction using unit bits and with maximum range of 0..10000000 bits
EEE = wobble frequency in unit Hz*100 and in range 1..2500000

Set LP8 outputs
Set LP0..LP7 output pins on LP8 Extension Board
C = 0x09
AAAA - bitpattern to be set on LP0..LP7 output pins, here only lower 8 bits are used.
BBBB = unused, set to 0
EEE = unused, set to 0

Ping
This command can be used to let the controller send back a state-information. So it can be used to check if the card is still operating or not.
ATTENTION: this command should not be sent repeatedly and without any delay! This could cause E1701D scanner controller to stall because the massive data transfer has to be handled. So it is recommended to have a delay of at least 150 msec between every ping.
C = 0x0A
AAAA = unused, set to 0
BBBB = unused, set to 0
EEE = unused, set to 0

Set digital outputs
Set DOut0..DOut7 output pins on Digi I/O Extension Board
C = 0x0B
AAAA - bitpattern to be set on DOut0..DOut7 output pins, here only lower 8 bits are used.
BBBB = bitmask specifying which of the bits in AAAA have to be set or cleared, all these bits in AAAA are left unchanged, where the corresponding bit in BBBB is 0
EEE = unused, set to 0

Set lasermode
Specify the laser mode the card has to operate with
C = 0x0C
AAAA - flags specifying the laser mode, here following values have to be or-concatenated to specify the behaviour of a laser:
- 0x40000000 - laser frequency on LaserA output is turned on immediately and together with laser gate signal, this flag can't be used together with 0x20000000
- 0x20000000 - laser frequency on LaserA output is turned on after FPK time, this flag can't be used together with 0x40000000
- 0x10000000 - laser supports FPK on LaserB output
- 0x08000000 - laser frequency has to be turned off and switched to standby-frequency
- 0x04000000 - a frequency can be emitted at LaserB permanently, the related frequency can be specified with command 0x15; this flag requires firmware version 19 or newer.

Using these flags following laser types can be configured:
- CO2-laser: 0x40000000 + 0x08000000
- YAG-laser with FPK: 0x40000000 + 0x08000000 + 0x10000000 or 0x20000000 + 0x08000000 + 0x10000000
- laser with continuously running frequency: 0x40000000

BBBB = unused, set to 0
EE EE = unused, set to 0

Set marking on-the-fly parameters
Specify the parameters used for marking on-the-fly applications
C = 0x0D
AAAA = marking on-the-fly resolution in X-direction in unit bits per encoder increment
BBBB = marking on-the-fly resolution in Y-direction in unit bits per encoder increment
EE EE = unused, set to 0

Set laser frequency
Specify the frequency the laser has to be operated with during marks, usage of these parameters depends on the lasermode specified with command 0x0C
C = 0x0E
AAAA = frequency in unit Hz and in range 25..20000000 Hz
BBBB = pulse-width in unit microseconds and in range 1..65530 usec
EE EE = unused, set to 0

Set laser standby frequency
Specify the frequency the laser has to be operated with during jumps, usage of these parameters depends on the lasermode specified with command 0x0C
C = 0x0F
AAAA = frequency in unit Hz and in range 25..20000000 Hz
BBBB = pulse-width in unit microseconds and in range 1..65530 usec
EE EE = unused, set to 0

Set first pulse killer
Specify the pulse width of the FPK signal when laser is turned on, usage of these parameters depends on the lasermode specified with command 0x0C
C = 0x11
AAAA = FPK pulse width in unit microseconds*100
BBBB = the time the laser frequency has to be started after beginning of FPK using unit microseconds*2, this value is used only when lasermode flag 0x20000000 is set
EE EE = unused, set to 0

Switch MO-output
Turns the MO-output of LP8 Extension Bolard on or off
C = 0x12
AAAA = turn MO output on when equal 1, turn it off when 0
BBBB = unused, set to 0
Release external trigger

When card is waiting for an external trigger this command can be sent to release this external trigger by software and to continue execution without the need to receive a real external signal.

C = 0x13
AAAA = unused, set to 0
BBBBB = unused, set to 0
EEEEE = unused, set to 0

Wait for external input signal

Stop execution until a defined input bitpattern is detected at configurable input pins DIn0..DIn7 of Digi I/O Extension Board.

C = 0x14
AAAA = a bitpattern specifying which signals LOW or HIGH have to be detected at digital input pins
BBBBB = a bitpattern specifying which of the digital input pins have to be watched for a signal, these bits that are set to 0 are ignored while these bits, that are set to 1 have to get the state specified in previous parameter in order to let operation of card continue
EEEEE = unused, set to 0

Set LaserB frequency

Specify the frequency LaserB output has to emit; this function can only be used when operating using a laser mode with flag 0x04000000 set (see command 0x0C above).
This command requires firmware version 19 or newer.

C = 0x15
AAAA = frequency in unit Hz and in range 25..20000000 Hz
BBBBB = pulse-width in unit microseconds and in range 1..65530 usec
EEEEE = unused, set to 0

Wait until on-the-fly-increments have been elapsed

This command adds some special kind of delay to the application. It can be used only when marking on-the-fly is enabled (by setting the on-the-fly factors), and halts laser marking not for a given time but for a given distance. Marking is continued only when the given number of increments has elapsed. When no or not enough increments are counted by the controller, operation only can be stopped.
This command requires firmware version 27 or newer.

C = 0x16
AAAA = positive or negative number of increments to wait for until operation continues; here it depends on the used counting direction of the encoder if the given distance-value has to be positive or negative, when sign of the number and counting direction of the encoder do not fit to each other, the controller will halt at this position for a very long time
BBBBB = unused, set to 0
EEEEE = unused, set to 0

Specify output for MIP-signal

Specify a single output pin of Digi I/O Extension Board to be used for "Mark in progress"-signal, this output pin will be HIGH as long as a marking operation is in progress.

C = 0x2A
AAAA = the number (not a bitpattern!) of the digital output pin to be used for MIP-signal (in range 0..7)
BBBBB = unused, set to 0
EEEEE = unused, set to 0

Halt/continue current marking operation

Stops the current marking operation on very next appearance of a jump or continue a previously halted operation.
C = 0x2F
AAAA = 1 to halt marking and 0 to continue a halted operation
BBBB = unused, set to 0
EEEE = unused, set to 0

Specify output for WET-signal
Specify a single output pin of Digi I/O Extension Board to be used for "Wait External Trigger"-signal, this output pin will be HIGH as controller is waiting for an external trigger.
This command requires firmware version 21 or newer.
C = 0x33
AAAA = the number (not a bitpattern!) of the digital output pin to be used for WET-signal (in range 0..7)
BBBB = unused, set to 0
EEEE = unused, set to 0

Set first row of 2x2 output matrix
Specify the elements m11 and m12 of a 2x2 output matrix which is applied to all coordinate values as soon as the second half is applied with command 0x41. This matrix can be used to scale, slant, rotate and mirror the input coordinates in respect to the output positions. For details please check out description of command 0x41 below
This command requires firmware version 31 or newer.
C=0x40
AAAA = the m11 part of the 2x2 matrix multiplied with 1000000
BBBB = the m12 part of the 2x2 matrix multiplied with 1000000
EEEE = unused, set to 0

Set second row of 2x2 output matrix
Specify the elements m21 and m22 of a 2x2 output matrix which is applied to all coordinate values together with the first row of matrix data which has to be set using command 0x40 in a preceding call. This matrix can be used to scale, slant, rotate and mirror the input coordinates in respect to the output positions. Assumed a matrix bases on an 4-elements array, it has following structure:

\[ \{m11, m12, m21, m22\} \]

then these matrix values can be used and even combined with each other by multiplying them:
- rotation: \( \{\cos(\text{angle}), -\sin(\text{angle}), \sin(\text{angle}), \cos(\text{angle})\} \)
- scaling: \( \{\text{factorX}, 0.0, 0.0, \text{factorY}\} \)
- slant X: \( \{1.0, 1.0/\tan(\text{angle}), 0.0, 1.0\} \)
- slant Y: \( \{1.0, 0.0, 1.0/\tan(\text{angle}), 1.0\} \)
- mirror X: \( \{-1.0, 0.0, 0.0, 1.0\} \)
- mirror Y: \( \{1.0, 0.0, 0.0, -1.0\} \)

This command requires firmware version 31 or newer.
C=0x41
AAAA = the m21 part of the 2x2 matrix multiplied with 1000000
BBBB = the m22 part of the 2x2 matrix multiplied with 1000000
EEEE = unused, set to 0

Download new firmware
Download a new firmware file to the controller and write it to the microSD-card so that it can be used after next reboot. The binary data of the new firmware have to be appended directly to this command. This command has to be used in a specific sequence in order to ensure the current firmware file is updated correctly:
- ensure the card is idle (state-flag SSSS is 0)
- send command 0x45 with length of firmware data and checksum
- send binary firmware data directly after this command
- wait until card state returns "active" (by repeatedly sending ping-commands), now in state flag SSSS bit
0x4000 (E1701_CSTATE_WRITING_DATA) is set
• wait until card state returns “idle” again (by repeatedly sending ping-commands), the flag 0x4000 no longer should be set in state flag SSSS
• check if an error occurred: when flags 0x8000 (E1701_CSTATE_WRITING_DATA_ERROR) is set in state flag SSSS, downloading or writing or checksum calculation failed and the original file was not replaced; for debugging in such a case the command “clog” can be called repeatedly until the related error text was found; when this error flag is set it can be reset only by using command 0x45 again
• reboot the controller
• check if the version of the firmware has changed
This command requires firmware 32 or newer.
C = 0x45
AAAA = the length of the firmware file in bytes
BBBB = checksum for verification of the downloaded data, only when this checksum is correct, the old firmware file will be replaced; the checksum can be calculated using following function (C example code):

```c
unsigned int crc32b(const char *buf, size_t len)
{
    int k;
    unsigned int crc=0xFFFFFFFF;

    while (len--)
    {
        crc^=*buf++;
        for (k=0; k<8; k++)
            crc=crc&1 ? (crc>>1) ^ 0x82f63b78 : crc>>1;
    }
    return ~crc;
}
```
EEEE = specifies the file which has to be overwritten by the current data download:
• 0 – overwrite file 0:/version.txt when downloading of data was successful
• 1 – overwrite file 0:/e1701.fwi when downloading of data was successful
• 2 – overwrite file 0:/e1701.dat when downloading of data was successful
• 3 – overwrite file 0:/e1701.cfg when downloading of data was successful

Reset the board
This function performs a warm reboot of the hardware and restarts the firmware. Reboot is done immediately, means this command does not return anything but connection to the board will be interrupted as soon as it has been sent.
This command requires firmware version 19 or newer.
C = 0xFF
AAAA = unused, set to 0
BBBB = unused, set to 0
EEEE = unused, set to 0
9 Programming Interfaces

The e1701.dll / libe1701.so shared library provides an own programming interface that gives the possibility to access and control the E1701 scanner controller card.

Beside of that e1701inter.dll / libe1701inter.so come with some compatibility interfaces for different other scanner cards like Scanlab(tm) RTC4(tm) and SCAPS(tm) USC1/2. They can be used to access E1701 scanner card with existing software easily. To do that, the "e1701inter" shared library just has to be renamed to the library name of the original vendor (like RTC4DLL.DLL, libslrtc4.so or sc_optic.dll). In order to operate E1701 card with different than the default settings, recompilation of code to be reused is necessary, here at the very beginning and before initialisation a call to E1701_set_connection() has to be added in order to specify the connection to the card. Beside of this none of the different programming interfaces should be mixed.

Sources of e1701inter.dll / libe1701inter.so are open and available for free, they can be found in OpenAPC SDK (available for download at http://www.openapc.com/download.php#sdk) or in GIT (available at https://sourceforge.net/p/oapc/code/ci/master/tree/).

9.1 E1701 Easy Interface Functions

These functions belong to the native programming interface of E1701 scanner card and should be used preferential in order to get access to all features and full performance of the scanner card. Functions of E1701 Easy Interface are either stream commands that are executed in the order they are called, or functions that are executed immediately.

The E1701 does NOT use the concept of two or more lists that have to be managed and switched by the calling application. Here all stream commands simply are sent to the card without the need to provide some additional management information. Output of data is started only when E1701_execute() is called or when a card-internal threshold is exceeded. This card-internal triggered output of data can be held back only by calling function E1701_set_trigger_point() as very first so that marking starts only after an external trigger signal was detected by the card. In this case it is necessary to watch the buffer fill level of the card to avoid a buffer-overrun by calling function E1701_get_free_space().

E1701 Easy Interface uses unit "bits" as base for all units and parameters. Since E1701 card internally uses 26 bits resolution for a better accuracy and to minimize round off errors, all calculation is done with these 26 bits. So the working area always has a size of 26 x 26 bits equal to 67108864 x 67108864. Independent from real resolution and output of hardware all calculations have to be done within this 26 bit range.

E1701 Easy Interface provides following functions:

unsigned char E1701_set_connection(const char *address)

This function has to be called as very first. It is used to specify the IP address where the card is accessible at (in case of Ethernet connection) or the serial interface (in case of USB connection, "COMx" for Windows and "/dev/ttyACMx" for Linux where "x" is the number of its interface). By default IP 192.168.2.254 is used. This is the only function that has to be called in case of both, when compatibility functions and when E1701 Easy Interface functions are used.

It returns a board instance number that has to be used with all following functions (this is true for Easy Interface and RTC-compatible functions).

Please note: this function does only set the connection information, it does not yet open the connection to the controller! This happens on first call to E1701_load_correction().

Parameters:
address – a char-array containing the IP in xxx.yyy.zzz.aaa notation or the name of the COM port to be used

Return: the board instance number or 0 in case of an error
void E1701_set_password(const char n, const char *ethPwd)
Sets a password that is used for Ethernet connection of E1701 card. The same password has to be
configured on E1701 configuration file e1701.cfg with parameter "passwd" to add an additional level of
security to an Ethernet controlled card.

PLEASE NOTE: usage of this password does NOT provide enough security to control the card via networks that
are accessible by a larger audience, publicly or via Internet! Also when this password is set, the card always
should operate in secured, separated networks only!
Every card and every connection should use an own, unique password that can consist of up to 48 characters
containing numbers, lower- and uppercase letters and punctuation marks. Due to compatibility reasons no
language-specific special character should be used.
When connected via USB serial interface, this password is ignored. In this case no authentication is done.

Parameters:
ethPwd - the password to be used to authorise at an E1701 card. To reset a local password for connecting to a
card that doesn't has a Ethernet password configured, hand over an empty string "" here

int E1701_load_correction(unsigned char n, const char* filename, unsigned char tableNum)
Loads a correction file to be used during vector data output. In case a previously loaded correction
table has to be flushed and no other correction has to be used, parameter "filename" needs to be empty.
This function has to be called for first time on initialisation and before any vector data are sent to the board. It is
mandatory to call this function at least once since it establishes connection to E1701 card. So when no
correction file has to be used, this function still has to be called but with an empty filename "".
This function supports different correction table file formats directly and without previous conversion:
• Scanlab .ctb and .ct5 files
• SCAPS .ucf files
• Raylase .gcd files
• CTI/GSI .xml files
• BeamConstruct .bco high resolution files
This is not a stream-command, means its data may be applied immediately and independent from current
stream state.

Parameters:
n - the 1-based board instance number as returned by E1701_set_connection()
filename - the full path to the correction file to be loaded from file system, when "" is specified here, a
previously used correction file is flushed and no/neutral correction is used as long as no other correction table
is given
tableNum - the 0-based correction table number these data have to be loaded for; it is possible to download
up to 16 different correction tables and to switch between them during operation using function
E1701_switch_correction()

Return: E1701_OK or an E1701_ERROR - or RTC-compatible return code in case of an error

void E1701_close(unsigned char n)
Closes the connection to a card and releases all related resources. After this function was called, no
more commands can be sent to the card until E1701_set_connection() and
E1701_load_correction() / n_load_correction_file() / load_correction_file() /
ScSCIInitInterface() is called again.

Parameters:
n - the 1-based board instance number as returned by E1701_set_connection()

int E1701_switch_correction(unsigned char n, unsigned char tableNum)
Switches between up to 16 correction tables on the fly. When a table-number is given where no file was
downloaded before using function \texttt{E1701\_load\_correction()}, no correction is performed on all following
vector data.

This is a stream-command, means the new correction is applied to vector data sent to the card after this
command but NOT to already sent but not yet processed data. Thus on-the-fly switching between correction
tables is possible.

Parameters:
- \texttt{n} – the 1-based board instance number as returned by \texttt{E1701\_set\_connection()}
- \texttt{tableNum} – the 0-based table number of the correction that has to be used for all following vector data

Return: \texttt{E1701\_OK} or an \texttt{E1701\_ERROR\_} -return code in case of an error

\begin{verbatim}
int E1701\_set\_debug\_logfile(const unsigned char n, const char *path, const unsigned char flags)

This function can be used during development to check an own application regarding called commands
and their parameters. It lets libe1701 write all function calls into a logfile so that it is possible to evaluate the
real order of commands.

Parameters:
- \texttt{n} – the 1-based board instance number as returned by \texttt{E1701\_set\_connection()}
- \texttt{path} – full path to the file which has to be used as debug log file
- \texttt{flags} – a bunch of OR-concatenated flags which specify what function calls have to be written into or filtered
  from the log output; when 0x00 is specified here, the log file is kept quite small. When 0x01 is set, all motion-
  related function calls are added too, when 0x02 is set, all calls which check the state of the card are added to
  the log file.

Return: \texttt{E1701\_OK} or an \texttt{E1701\_ERROR\_} -return code in case of an error
\end{verbatim}

\begin{verbatim}
int E1701\_set\_xy\_correction3(const unsigned char n, const unsigned int flags, const double gainX, const double gainY, const double rot, const int offsetX, const int offsetY, const double slantX, const double slantY)

Sets size correction factor and offset for X and Y direction of working area as well as a rotation. With
this command a matrix set with \texttt{E1701\_set\_matrix2()} will be overwritten.

This is a stream-command, means its parameters are applied at a point in stream that is relative to the other
stream commands and changed values apply only to these vector data and coordinates, which are sent after
calling this function.

Parameters:
- \texttt{n} – the 1-based board instance number as returned by \texttt{E1701\_set\_connection()}
- \texttt{flags} – following flags require firmware version 32 or newer:
  - \texttt{E1701\_COMMAND\_FLAG\_XYCORR\_MIRRORX} – the output will be mirrored in X-direction
  - \texttt{E1701\_COMMAND\_FLAG\_XYCORR\_MIRRORY} – the output will be mirrored in Y-direction
- \texttt{gainX} – scale factor in x-direction, 1.0 means no scaling
- \texttt{gainY} – scale factor in y-direction, 1.0 means no scaling
- \texttt{rot} – rotation of whole working area in unit degrees
- \texttt{offsetX} – offset in x-direction in unit bits, 0 means no offset
- \texttt{offsetY} – offset in y-direction in unit bits, 0 means no offset
- \texttt{slantX} – trapezoidal correction along X-axis in range -45..45°
- \texttt{slantY} – trapezoidal correction along Y-axis in range -45..45°

Return: \texttt{E1701\_OK} or an \texttt{E1701\_ERROR\_} -return code in case of an error
\end{verbatim}
int E1701_set_z_correction(const unsigned char n, const unsigned int h, const double xy_to_z_ratio, const int res1)

Set additional Z correction parameter. This function may be used in cases where third axis is used excessively and with a large Z working range. Here additional deviation occurs when no F-Theta lens is used caused by the fact that the beam is always sent from the centre of the scanhead – which causes some kind of projection resulting in larger or smaller X and Y positions depending on the real Z height.

This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands and changed values apply only to these vector data and coordinates, which are sent after calling this function.

This function requires firmware version 27 or newer.

Parameters:
- n - the 1-based board instance number as returned by E1701_set_connection()
- h - the vertical height from last mirror of the scanhead to the working area (Z-position 0 of working area) in unit bits
- xy_to_z_ratio - factor specifying the ratio between maximal horizontal working area size and maximal vertical movement size. As an example: when the working area has a size of 100 x 100 mm and the Z-axis has a maximum movement range of -20 mm .. 20 mm, the ratio to be set is 2.5 (100 mm horizontal divided by 40 mm vertical)
- res1 - reserved for future use, set to 0

For more details please refer to the image below:

```
Here "h" is the height from the position where the beam hits the last mirror to the position of the working area at z=0 position (in unit bits). "xy" is the width of the working area to be used together with the "z" range from "z=min" to "z=max" to calculate the xy_to_z_ratio. All working area parameters like its width "xy" and the "z"-range are expected to be the theoretical maximum of the full range, not the – possibly smaller – range used in a specific setup.

Return: E1701_OK or an E1701_ERROR_return code in case of an error
```

int E1701_set_speeds(unsigned char n, double jumpspeed, double markspeed)

Set scanner speed values to be used for all following vector data and until not replaced by other speed values.

This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands and changed values apply only to these vector data and coordinates, which are sent after calling this function.
Parameters:
n – the 1-based board instance number as returned by `E1701_set_connection()`
jumpspeed – scanner movement speed during jumps (movements when laser is off) in unit bits/msec and range 1..4294960000
markspeed – scanner speed during mark (movements when laser is on) in unit bits/msec and range 1..4294960000

Return: `E1701_OK` or an `E1701_ERROR_-return code in case of an error`

```c
int E1701_set_laser_delays(unsigned char n, double ondelay, double offdelay)
Set laser delay values to be used for all following vector data and until not replaced by other delay values.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other
stream commands and changed values apply only to these vector data and coordinates, which are sent after
calling this function.

Parameters:
n – the 1-based board instance number as returned by `E1701_set_connection()
ondelay – laser on delay in unit microseconds, can be a negative or a positive value
offdelay – laser off delay in unit microseconds, must be a positive value

Return: `E1701_OK` or an `E1701_ERROR_-return code in case of an error`
```

```c
int E1701_set_scanner_delays2(const unsigned char n, const unsigned int flags, const double jumpdelay, const double markdelay, const double polydelay)
Set scanner delays in unit microseconds. Smallest possible value and resolution is 0.5 microseconds.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other
stream commands. So values set here apply only to these vector data that are sent after this command.

Parameters:
n – the 1-based board instance number as returned by `E1701_set_connection()
flags – here some flags can be set which add some further functional specifications and features to this
function. At the moment following flags are supported and can be OR-concatenated with each other:
• `E1701_COMMAND_FLAG_SCANNER_VAR_POLYDELAY` – when this flag is set, the value set via
polydelay is not applied statically to every point within a polygon, but it is set dynamically depending
on the angle between two lines; no angle (a straight line) results in no delay while a 180 degree angle
results in a full delay as set by value polydelay; this flag requires firmware version 31 or newer
jumpdelay – the jump delay value in unit microseconds
markdelay – the mark delay value in unit microseconds
polydelay – the in-polygon delay value in unit microseconds

Return: `E1701_OK` or an `E1701_ERROR_-return code in case of an error`
```
int E1701_set_laser_mode(unsigned char n, unsigned int mode)

Sets the laser mode to be used for all following operations, this value influences the signals emitted at
the connectors of the card. This function has to be called prior to setting any other laser parameters (like
frequency, standby-frequency, power).

This is a stream-command, means its parameters are applied at a point in stream that is relative to the other
stream commands.

Parameters:

n - the 1-based board instance number as returned by E1701_set_connection()
mode - the laser mode, here one of the following values is possible:
  • E1701_LASERMODE_CO2 - for controlling CO2 lasers, this mode supports stand-by frequency at
    LaserA output (to be set with function E1701_set_standby()) and PWM-modulated frequencies
during marking and for power control (to be set with function E1701_set_laser_timing())
  • E1701_LASERMODE_YAG1 - for controlling YAG lasers, this mode supports stand-by and Q-Switch
    frequency at LaserA output (to be set with function E1701_set_standby()) and a first pulse killer
    signal at output LaserB that is issued on beginning of a mark together with the Q-Switch frequency (to
    be set with function E1701_set_fpk());

Here Q-Switch signal is started together with laser gate and FPK pulse. At end of mark when laser gate
is turned off stand-by frequency is emitted at LaserA.

  • E1701_LASERMODE_YAG2 - for controlling YAG lasers, this mode supports stand-by and Q-Switch
    frequency at LaserA output (to be set with function E1701_set_standby()) and a first pulse killer
    signal at output LaserB that is issued on beginning followed by Q-Switch frequency that starts when
    FPK pulse has finished:

Here FPK and laser gate are started together. Q-Switch signal is started at end of FPK pulse. At end of
mark when laser gate is turned off, stand-by frequency and pulse-width is emitted at LaserA instead of
Q-Switch frequency.

  • E1701_LASERMODE_YAG3 - for controlling YAG lasers, this mode supports stand-by and Q-Switch
    frequency at LaserA output (to be set with function E1701_set_standby()) and a first pulse killer
    signal at output LaserB that is issued on beginning followed by Q-Switch frequency that starts after a
    freely configurable time period “yag3QTime”:
Here FPK and laser gate are started together. Q-Switch signal is started after yag3QTime has elapsed according to the beginning of FPK pulse. This time value can be set using function E1701_set_fpk(). At end of mark when laser gate is turned off, stand-by frequency and pulse-width is emitted at LaserA instead of Q-Switch frequency.

- E1701_LASERMODE_CRF – for controlling lasers that require a continuously running frequency (like fiber-lasers), this frequency is emitted at LaserA output and can be set and changed by calling function E1701_set_standby().
- E1701_LASERMODE_DUAL – for controlling special lasers that require two frequencies, the second, continuously running frequency is emitted at LaserB output and can be set with function E1701_set_laserb().

Return: E1701_OK or an E1701_ERROR_return code in case of an error

int E1701_set_laser(const unsigned char n, const unsigned int flags, const char on)
Switches the laser on or off independent from many mark or jump commands.

Parameters:
- n – the 1-based board instance number as returned by E1701_set_connection()
- flags – handling flags specifying the behaviour of this command, E1701_COMMAND_FLAG_STREAM to use it as stream command, E1701_COMMAND_FLAG_DIRECT to execute it immediately and independent on current stream and execution state; in case E1701_COMMAND_FLAG_STREAM is used, please ensure this function call is followed by other stream commands, elsewhere the laser is turned off for security reasons as soon as no more data are available to process in order to not to let the laser fire while the card is waiting
- on – set to 1 to turn the laser on or to 0 to turn it off

Return: E1701_OK or an E1701_ERROR_return code in case of an error

int E1701_set_wobble(unsigned char n, unsigned int x, unsigned int y, double freq)
This function gives the possibility to not to let the laser beam follow the given path directly but to rotate around the specified path and lasers current position. Depending on chosen wobble-parameters and marking speed, this results either in a thick or a sinusoidal line. This call sets wobble parameters to be used for all following vector data and until not replaced by other wobble values or by 0 which disables wobble mode. This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
- n – the 1-based board instance number as returned by E1701_set_connection()
- x – wobble amplitude in x direction in units bits and range 1..10000000
- y – wobble amplitude in y direction in units bits and range 1..10000000
- freq – wobble frequency in Hz in range 1..25000

Return: E1701_OK or an E1701_ERROR_return code in case of an error
int E1701_jump_abs(unsigned char n, int x, int y, int z)

Perform a jump (movement with laser turned off) to the given position. This causes a galvo movement from current position to the one specified by this functions parameters using the jump speed and taking the jump delay into account:

When laser was turned on before this function is called, laser is turned off at the beginning with a delay specified by laser off delay (please refer to description of E1701_mark_abs() for a diagram showing laser off delay too).

This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.

Parameters:
- n - the 1-based board instance number as returned by E1701_set_connection()
- x - the x-coordinate in unit bits the scanner has to jump to
- y - the y-coordinate in unit bits the scanner has to jump to
- z - the z-coordinate in unit bits the scanner has to jump to (requires a hardware that is equipped with Z-channel)

Return: E1701_OK or an E1701_ERROR_return code in case of an error

int E1701_mark_abs(unsigned char n, int x, int y, int z)

Perform a mark (movement with laser turned on) to the given position. This causes a galvo movement from current position to the one specified by this functions parameters using the mark speed and taking the mark delay into account. When laser was turned off before this function is called, laser is turned on at the beginning with a delay specified by laser on delay:
This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.

Parameters:
n - the 1-based board instance number as returned by E1701_set_connection()
x - the x-coordinate in unit bits the scanner has to move to
y - the y-coordinate in unit bits the scanner has to move to
z - the z-coordinate in unit bits the scanner has to move to (requires a hardware that is equipped with Z-channel)

Return: E1701_OK or an E1701_ERROR_-return code in case of an error

 HANDLE WITH CARE! This function causes galvo movement to the given position immediately, without respect to any mark or jump speed values, without micro-vectorisation or intermediate steps! This means it can result in a very heavy movement for the galvos and in worst case it may cause some damage! Since the resulting movement speed may be way too high for the used galvos, they may overshoot and need some time until the desired position is reached. So this function is mainly intended to be used for very small position changes in respect to the galvos current position.

This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.

Parameters:
n - the 1-based board instance number as returned by E1701_set_connection()
x - the x-coordinate in unit bits the scanner has to jump to
y - the y-coordinate in unit bits the scanner has to jump to
z - the z-coordinate in unit bits the scanner has to jump to (requires a hardware that is equipped with Z-channel)
laserOn - specifies if the movement has to be done with laser turned on (1) or off (0)

Return: E1701_OK or an E1701_ERROR_-return code in case of an error

Set the operational mode for E1701_mark_pixelline(). This function influences the behaviour when marking a pixel line. This is a stream-command, means its parameters are applied at a point in stream that
is relative to the other stream commands. This function requires firmware version 19 or newer.

Parameters:
- n - the 1-based board instance number as returned by E1701_set_connection()
- mode - pixel marking mode, this parameter can be set to:
  - 0 - default mode, while marking a pixel line the controller tries to perform jumps when power
    is below of the given threshold powerThres to save marking time
  - E1701_PIXELMODE_NO_JUMPS - no jumps are performed, the given power threshold is
    ignored and the full pixel line is done with marking speed; this mode is slower but can result in
    more accurate and more exact images
  - E1701_PIXELMODE_JUMP_N_SHOOT - marking of the line is no longer done with a
    continuous movement but with a sequence "jump to position → shoot → jump to next position
    → shoot → jump to next position → shoot..."; here the shoot-time is equal to the laser-off-delay
    minus laser-on-delay as set with function E1701_set_laser_delays()
- powerThres - this value is used only in default mode, when the marking power for some pixels is below of the
  given value (in unit percent), a jump is performed to save marking time, during this jump the laser is off and no
  marking is done
- res - reserved, set always to 0

Return: E1701_OK or an E1701_ERROR_-return code in case of an error

int E1701_mark_pixelline(unsigned char n, int x, int y, int z, int pixWidth, int pixHeight, int pixDepth, unsigned int pixNum, double *pixels, E1701_power_callback power_callback, void *userData)

This function can be used to mark a single line of a bitmap image. Here horizontal, vertical and even 3D
bitmap lines (going into depth) can be marked. Direction and orientation of the line to be marked can be chosen
freely. A full image can be created by concatenating several lines. Power control during marking of such a
bitmap line is not limited to some specific power outputs, it can be fully customised via a callback function.

Parameters:
- n - the 1-based board instance number as returned by E1701_set_connection()
- x, y, z - the starting coordinates of the line in unit bits
- pixWidth - the width of a single pixel (in unit bits), when this is set to a value greater or smaller than 0 while all
  the others are equal 0, a horizontal line is drawn; the sign of the value specifies the marking direction
- pixHeight - the height of a single pixel (in unit bits), when this is set to a value greater or smaller than 0 while all
  the others are equal 0, a vertical line is drawn; the sign of the value specifies the marking direction
- pixDepth - the depth of a single pixel (in unit bits, requires a 3D-capable scanhead), when this is set to a value
  greater or smaller than 0 while all the others are equal 0, line goes into depth; the sign of the value specifies the
  marking direction
- pixNum - the number of pixel data contained in the array of intensity values handed over with the following
  parameter
- pixels - an array of double-values with a length equal the number of pixels specified with pixNum and with
  an allowed range of 0.0..100.0 specifying the intensity; every entry of this array is equal to one pixel of the
  bitmap, so a greyscale-pixelline with brightness values in range 0..255 has to be converted to values in range
  0.0..100.0
- power_callback - this is a callback function of type

int (*E1701_power_callback)(unsigned char n, double power, void *userData)

which is used to set the power for every pixel. There these E1701_-functions have to be called that belong to
the used laser type and set the power values according to it's hardware capabilities. Within the power callback
function only stream commands are allowed to be called. It is not possible to use external devices that are not
synchronous to E1701 command stream. The power callback has to return with E1701_OK when setting of
power was successful. In case of an error the appropriate error code has to be returned, the pixel marking
function will be cancelled in such a case too and does not finish marking of the line. Parameter n is the 1-based
board instance number specifying the board the power has to be changed for, power is the power to be set in
unit percent and userData are some free to use, custom data that can be handed over on call to E1701_mark_pixelline().

userData - here some custom data can be handed over which are forwarded on and handed over at every call of the power-callback.

Return: E1701_OK or an E1701_ERROR_ - return code in case of an error.

```c
int E1701_set_matrix2(const unsigned char n, const unsigned int flags, const double m11, const double m12, const double m21, const double m22)
```

Specify a 2x2 matrix that contains scaling and rotation corrections for the output. When a given matrix element parameter has a value smaller or equal -1000000 it is ignored and the previous/default value is kept at this position in matrix. With this command any correction set with E1701_set_xy_correction2() will be overwritten.

This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
- n - the 1-based board instance number as returned by E1701_set_connection()
- flags - reserved for future use, set to 0 for compatibility
- m11 - first matrix element in first row
- m12 - second matrix element in first row
- m21 - first matrix element in second row
- m22 - second matrix element in second row

Return: E1701_OK or an E1701_ERROR_ - return code in case of an error.

```c
int E1701_set_trigger_point(unsigned char n)
```

Specifies a point in data stream where execution has to stop until an external trigger signal (mark start) or a manual release of this trigger point is detected. This expects a rising edge on ExtStart input or calling of function E1701_release_trigger_point().

This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.

Parameters:
- n - the 1-based board instance number as returned by E1701_set_connection()

Return: E1701_OK or an E1701_ERROR_ - return code in case of an error.

```c
int E1701_release_trigger_point(unsigned char n)
```

This function should be called only when a call to E1701_set_trigger_point() was done before. It acts like an external trigger signal, releases the waiting condition and lets the controller start processing. So this function provides some kind of software-simulated external start-signal.

ATTENTION: this command will not arrive at the controller when there is no more space left on it, means when all controller-internal buffers are filled. So after a call to E1701_set_trigger_point() and during sending of commands and data to the controller, application has to ensure there is some space left in controller’s buffers. This can be done by calling E1701_get_free_space() with flag E1701_FREE_SPACE_PRIMARY for checking the available space in primary buffer. It is recommended to leave space for at least 10000 elements in primary buffer in order to let a call to E1701_release_trigger_point() work properly.

When the buffers already have been filled completely, this function will no longer work and marking can be started only by applying the ExtStart hardware signal.

This is not a stream-command, it is applied to controller immediately.
Parameters:
 n – the 1-based board instance number as returned by `E1701_set_connection()`

Return: `E1701_OK` or an `E1701_ERROR_`-return code in case of an error

```c
int E1701_set_sync(const unsigned char n, const unsigned int flags, const unsigned int value)
```

This function sends a synchronisation value to the controller. As soon as marking reaches the related position in stream, the value returned by function `E1701_get_sync()` changes to the value given here. This command delays execution of the data by 0.5 usec, so it should not be used excessively. A value of `0xFFFFFFFF` disables this function.

This function requires firmware version 30 or newer.

Parameters:
 n – the 1-based board instance number as returned by `E1701_set_connection()`
 flags – currently unused, set to 0 for future compatibility
 value – the value to be used as sync-identifier, here on every call a different value should be handed over in order to differentiate what is returned by `E1701_get_sync()`.

Return: `E1701_OK` or an `E1701_ERROR_`-return code in case of an error

```c
unsigned int E1701_get_sync(const unsigned char n)
```

Returns a sync-identifier as set by `E1701_set_sync()` as soon as the related position in stream was reached.

This function requires firmware version 30 or newer.

Parameters:
 n – the 1-based board instance number as returned by `E1701_set_connection()`

Return: the last sync-identifier which was identified and processed in stream of commands or `0xFFFFFFFF` when function is not used/turned off

```c
int E1701_execute(unsigned char n)
```

Starts execution of all previously sent commands in case card is not already outputting these data. The E1701 Easy Programming Interface does not use the concept of two or more lists that have to be handled and switched by the calling application. Nevertheless the user has to ensure the card can start marking by calling this function after all vector data have been sent to the card. Here it does not matter if the card is already executing or not, subsequent calls to `E1701_execute()` do not influence marking behaviour. More than this: in case very much data are sent to the card, it starts marking automatically after a defined fill level was reached. Due to this automated, fill level dependent start it would not be necessary to call `E1701_execute()`. But in situations where only very few data are sent to the card, it is necessary to call this function in order to start marking also in these cases where the internal fill threshold is not reached and where the card would not start marking immediately. Thus it is recommended to always call this function after all marking data have been sent. Marking is finished only when STOP (ExtStop signal input or `E1701_stop_execution()` )is invoked or when the internal buffer is empty. When internal buffer runs empty because subsequent data are not sent fast enough, an additional call to `E1701_execute()` is necessary in order to output the remaining data.

This is not a stream command since it controls the already sent stream of commands.

Parameters:
 n – the 1-based board instance number as returned by `E1701_set_connection()`

Return: `E1701_OK` or an `E1701_ERROR_`-return code in case of an error
int E1701_stop_execution(unsigned char n)

Stops the currently running execution as fast as possible and drops all marking data that still may be queued. Calling this function also would drop all laser and scanner parameters that are already sent but not yet processed. Thus after calling this function it may be necessary to set scanner and laser parameters again in order to ensure they are used for following operations.

This is not a stream command since it controls the current stream of commands.

Parameters:

n - the 1-based board instance number as returned by E1701_set_connection()

Return: E1701_OK or an E1701_ERROR_return code in case of an error

int E1701_halt_execution(unsigned char n, unsigned char halt)

Halts or continues the processing and output of marking data. On halt=1 marking is stopped next time the laser is off. Different to a full stop no vector data are flushed. On continue (halt=0) controller continues processing at the point where halt occurred. When marking is stopped with E1701_stop_execution() the halt-condition is cleared too, means on next transmission of new marking data they are processed without the need to explicitly continue last operation. This command requires firmware version 16 or newer.

Parameters:

n - the 1-based board instance number as returned by E1701_set_connection()

halt - 1 to halt operation next time the laser is off, 0 to continue a previously halted operation

Return: E1701_OK or an E1701_ERROR_return code in case of an error

unsigned int E1701_get_startstop_state(unsigned char n)

This function returns a bit pattern that informs about state of the start and stop input pins. This is not a stream command since it returns the current state immediately. Here “current state” means the last known state. When the state changes during this call, it may be possible the previous, no longer actual state is given back since transmission of data from controller to host is done asynchronously and independent from a call to this function.

Parameters:

n - the 1-based board instance number as returned by E1701_set_connection()

Return: a bit pattern specifying the current state:

- bit 0 and 1 (0x00000003) specify if the start input was set after last call of this function, when these bits are set, a rising edge has been detected at this input; calling this function resets the internal state of these bits, means when it is called again and when no new rising edge has been detected meanwhile, these bits will be 0
- bit 2 and 3 (0x0000000C) specify if the stop input was set after last call of this function, when they are set, a rising edge has been detected at this input; calling this function resets the internal state of these bits, means when it is called again and when no new rising edge has been detected at top input meanwhile, these bits will be low
- bit 12 (0x00001000) this bit signals the start input is low, as long as this bit is set no start input signal is detected

int E1701_get_card_state2(const unsigned char n, unsigned int *value)

This function returns a bit pattern that informs about cards current operational state. Here “current state” means the last known state. When the state changes during this call, it may be possible the previous, no
longer actual state is given back since transmission of data from controller to host is done asynchronously and independent from a call to this function.

The card-states are enqueued internally in order to not to lose a "busy"-state which may be available for a very short time only in case of very small and fast marking cycles. So every state change caused by the calling application results in on state change returned by this function. This means for every marking cycle the application has to wait for two state changes: first wait until this function signals "busy" (E1701_CSTATE_MARKING|E1701_CSTATE_PROCESSING), next wait until this function signals "ready" (0).

During transfer of vector data and scanner/laser parameters this function should be called as rarely as possible: every call of `E1701_get_card_state2()` performs a full cycle of transmission and receiving of data to and from the controller. Dependent on the current transmission state this may result in submission of a small block of data which does not uses the full available bandwidth. On excessive use of this function this can slow down the whole transfer of data.

This is not a stream command since it returns the current state immediately.

Parameters:

n – the 1-based board instance number as returned by `E1701_set_connection()`

state – pointer to a variable where the card state has to be written to: a bit pattern of or-concatenated constants specifying the current state:

- E1701_CSTATE_MARKING – card is currently marking
- E1701_CSTATE_PROCESSING – card has received some data that are enqueued for marking
- E1701_CSTATE_WAS_START_PRESSED – the ExtStart input was triggered, this flag is cleared after it has been read and is set again only when ExtStart was triggered again
- E1701_CSTATE_WAS_STOP_PRESSED – the ExtStop input was triggered, this flag is cleared after it has been read and is set again only when ExtStop was triggered again
- E1701_CSTATE_FILE_WRITE_ERROR – this flag is returned only in case stand-alone data are written to the microSD card and in case an file error occurs during this procedure. As writing an EPR file is done as asynchronous stream, errors during this procedure are not announced by the functions which are called but only by this error state. For more information about writing of stand alone data please refer to section “9.1.1 Writing of stand-alone data”
- E1701_CSTATE_WAIT_EXTTRIGGER – the controller is in state "marking" but is not yet processing any data as it is waiting for an external trigger
- E1701_CSTATE_HALTED – the controller is in state "marking" but is not yet processing any data as it is currently halted by function `E1701_halt_execution()`
- E1701_CSTATE_WAIT_INPUT – the controller is in state "marking" but is not yet processing any data as it is waiting for a specific input pattern at the digital inputs

When the function returns an error code instead of E1701_OK, this value is undefined and can't be used.

Return: E1701_OK or an E1701_ERROR_-return code in case of an error

```c
int E1701_delay(unsigned char n, double delay)
```

Pause marking for the given time.

This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.

Parameters:

n – the 1-based board instance number as returned by `E1701_set_connection()`

delay - time to wait until marking continues in unit usec, smallest possible value is 0,500 usecs

Return: E1701_OK or an E1701_ERROR_-return code in case of an error

```c
int E1701_set_laser_timing(unsigned char n, double frequency, double pulse)
```

Set the frequency and pulse-width to be used during marking at LaserA output of LP8 Extension Board or Digital XY2-100 Baseboard.

This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.
Parameters:
- \( n \) – the 1-based board instance number as returned by \( \text{E1701\_set\_connection}() \)
- frequency – emitted frequency in unit Hz and in range 25..20000000 Hz
- pulse – pulse width in usec, this value has to be smaller than period length that results out of frequency

Return: \( \text{E1701\_OK} \) or an \( \text{E1701\_ERROR} \) return code in case of an error

```c
int \text{E1701\_set\_standby}(\text{unsigned \ char \ n}, \text{double \ frequency}, \text{double \ pulse})
```

Set the frequency and pulse-width to be used during jumps, as stand-by frequency or as continuously running frequency at LaserA output of LP8 Extension Board or Digital XY2-100 Baseboard.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
- frequency – emitted frequency in unit Hz and in range 25..20000000 Hz. When a value of 0 is given, the frequency at LaserA output is turned off at end of mark.
- pulse – pulse width in usec, this value has to be smaller than period length that results out of frequency
- \( n \) – the 1-based board instance number as returned by \( \text{E1701\_set\_connection}() \)

Return: \( \text{E1701\_OK} \) or an \( \text{E1701\_ERROR} \) return code in case of an error

```c
int \text{E1701\_set\_laserb}(\text{const \ unsigned \ char \ n}, \text{const \ double \ frequency}, \text{const \ double \ pulse})
```

Set the frequency and pulse-width to be used at LaserB output of LP8 Extension Board or Digital XY2-100 Baseboard. To use LaserB as second frequency output, a laser mode with flag \( \text{E1701\_LASERMODE\_DUAL} \) has to be configured.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.
This function requires firmware version 19 or newer.

Parameters:
- \( n \) – the 1-based board instance number as returned by \( \text{E1701\_set\_connection}() \)
- frequency – emitted frequency in unit Hz and in range 25..20000000 Hz
- pulse – pulse width in usec, this value has to be smaller than period length that results out of frequency

Return: \( \text{E1701\_OK} \) or an \( \text{E1701\_ERROR} \) return code in case of an error

```c
int \text{E1701\_set\_fpk}(\text{unsigned \ char \ n}, \text{double \ fpk}, \text{double \ yag3QTime})
```

Set the parameters for first pulse killer signal that is emitted via LP8 Extension Board or Digital XY2-100 Baseboard whenever the laser is turned on; this applies to YAG-modes only and is emitted as one single pulse at LaserB output.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
- \( n \) – the 1-based board instance number as returned by \( \text{E1701\_set\_connection}() \)
- fpk – the length of the first pulse killer signal in usec
- yag3QTime – the length of the first pulse killer signal in usec, this value is used only when laser mode \( \text{E1701\_LASERMODE\_YAG3} \) is set, elsewhere it is ignored

Return: \( \text{E1701\_OK} \) or an \( \text{E1701\_ERROR} \) return code in case of an error
int E1701_get_free_space(unsigned char n, int buffer)
This function returns the space (in unit “commands”) that is free in one of the buffers of E1701. Here parameter buffer decides which buffer has to be checked.

Parameters:
n - the 1-based board instance number as returned by E1701_set_connection()
buffer - expects a constant which decides what buffer has to be checked, it has to be set to one of the following values:
  • E1701_FREE_SPACE_PRIMARY - return size of the primary buffer; it can be used to avoid memory on host system is filled which may happen when vector data are sent to the controller while it’s internal buffers are already full. In this case these data would have been stored on host side consuming some memory there. Using this function this problem can be avoided by sending commands only in case this function returns a value that is (much) larger than 0.
The primary buffer that can be checked by using this value is one of two available buffers on E1701 controller. The primary one has a size of 900000 and is used to feed the secondary buffer (with a size of 17 million). So when this function returns 900000, this does not mean the buffer is empty and no vector data currently processed – they still may be stored in secondary buffer. So to check the operational state of the controller, only function E1701_get_card_state2() can be used.
This buffer has also to be checked when function E1701_release_trigger_point() is used in order to ensure the command can arrive at the controller. For a detailed description please refer to explanation of E1701_release_trigger_point() above.
  • E1701_FREE_SPACE_SECONDARY - return size of the secondary buffer; this one is filled by data from primary buffer and contains raw commands (like single micro vectors that concatenate to a full vector during output).

Return: -1 in case the function failed or the amount of free space in primary buffer.

void E1701_get_version(unsigned char n, unsigned short *hwVersion, unsigned short *fwVersion)
Get the hardware and software version of the used board. It is recommended to call this function after successful connect always and check if used hardware and firmware version is at least a version that is known to work with own software.
This is not a stream command, it is executed immediately and independent from all other commands.

Parameters:
n - the 1-based board instance number as returned by E1701_set_connection()
hwVersion - pointer to a variable where the hardware revision/version number is written into
fwVersion - pointer to a variable where the revision/version number of the firmware running on the board is written into

const int E1701_get_library_version()
Returns an integer value which is an identifier specifying the version of this shared library. In decimal notation this identifier uses format “Mmmrrr” where “M” is the major version, “m” the minor version number and “r” the release count. The bigger the whole returned number is, the newer the library is.

int E1701_lp8_write2(unsigned char n, unsigned int flags, unsigned char value)
Sets the LP8_0..LP8_7 outputs of 8 bit laser port of LP8 Extension Board without touching the related latch output. Total execution time of this command during processing on controller is 1 usec.
This function does not change the value at the analogue AOut0 output of LP8 Extension Board.
Depending on the value of parameter flags this is either a stream-command (means it is executed at a point in stream that is relative to the other stream commands) or a direct command (means it is executed immediately on calling). This command requires firmware version 16 or newer.
Parameters:

- **n** – the 1-based board instance number as returned by `E1701_set_connection()`
- **flags** – handling flags specifying the behaviour of this command, `E1701_COMMAND_FLAG_STREAM` to use it as stream command, `E1701_COMMAND_FLAG_DIRECT` to execute it immediately and independent on current stream and execution state
- **value** – the 8 bit value to be set at LP8 port

Return: `E1701_OK` or an `E1701_ERROR_`-return code in case of an error

```c
int E1701_lp8_write_latch(unsigned char n, unsigned char on, double delay1, unsigned char value, double delay2, double delay3)
```

Sets the LP8 8 bit laser port of LP8 Extension Board with freely definable delays and toggles the related latch output automatically; calling this function causes the following sequence of commands:

- turn latch bit on/off
- wait for `delay1` usecs
- set LP8
- wait for `delay2` usecs
- turn latch bit off/on
- wait for `delay3` usecs

The whole execution time of this sequence on the controller is is 1.5 usecs for setting LP8 outputs and toggling latch plus `delay1` plus `delay2` plus `delay3`. Depending on the value of parameter "on" this function may or may not set the analogue AOut0 output successfully.

This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.

Parameters:

- **n** – the 1-based board instance number as returned by `E1701_set_connection()`
- **on** – specifies if the latch bit has to be set to HIGH (on=1) or LOW (on=0) on first step, on second step it will toggle to value !=on
- **delay1** – delay to be issued after setting/clearing the latch bit for the first time
- **value** – the 8 bit value to be set at LP8 port
- **delay2** – delay to be issued after setting LP8 output and before clearing/settling the latch bit
- **delay3** – delay to be issued after clearing/settling the latch bit for the second time

Return: `E1701_OK` or an `E1701_ERROR_`-return code in case of an error

```c
int E1701_lp8_write_mo(unsigned char n, unsigned char on)
```

Sets the master oscillator output MO of LP8 Extension Board to be used with e.g. fiber lasers.

This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.

Parameters:

- **n** – the 1-based board instance number as returned by `E1701_set_connection()`
- **on** – the state the MO output has to be switched to

Return: `E1701_OK` or an `E1701_ERROR_`-return code in case of an error

```c
int E1701_lp8_a0_2(unsigned char n, unsigned int flags, unsigned char value)
```

Sets the analogue output AOut0 of LP8 Extension Board. This also changes the state of LP8_0..LP8_7 outputs and toggles the LP8 latch. Total execution time of this command on the controller is 1 usec.

Depending on the value of parameter `flags` this is either a stream-command (means it is executed at a point in stream that is relative to the other stream commands) or a direct command (means it is executed immediately on calling). This command requires firmware version 16 or newer.
Parameters:

- **n** – the 1-based board instance number as returned by `E1701_set_connection()`
- **flags** – handling flags specifying the behaviour of this command, `E1701_COMMAND_FLAG_STREAM` to use it as stream command, `E1701_COMMAND_FLAG_DIRECT` to execute it immediately and independent on current stream and execution state
- **value** – the 8 bit value to be set at analogue output port

Return: `E1701_OK` or an `E1701_ERROR_` return code in case of an error

```c
int E1701_digi_write2(unsigned char n, unsigned int flags, unsigned int value, unsigned int mask)
```

Sets the 8 bit digital output port of Digi I/O Extension Board. Depending on the value of parameter `flags` this is either a stream-command (means it is executed at a point in stream that is relative to the other stream commands) or a direct command (means it is executed immediately on calling). This command requires firmware version 16 or newer.

Parameters:

- **n** – the 1-based board instance number as returned by `E1701_set_connection()`
- **flags** – handling flags specifying the behaviour of this command, `E1701_COMMAND_FLAG_STREAM` to use it as stream command, `E1701_COMMAND_FLAG_DIRECT` to execute it immediately and independent on current stream and execution state
- **mask** – specifies which of the bits in "value" have to be used for setting and clearing output data, only these bits that are set to 1 in `mask` are changed according to the given `value`
- **value** – the 8 bit value to be set at digital out port

Return: `E1701_OK` or an `E1701_ERROR_` return code in case of an error

```c
int E1701_digi_pulse(const unsigned char n, const unsigned int flags, const unsigned int in_value, const unsigned int mask, const unsigned int pulses, const double delayOn, const double delayOff)
```

Send a sequence of pulses to the 8 bit digital output port of Digi I/O Extension Board. This command requires firmware version 16 or newer. When the controller works with a firmware version 32 or later, this operation causes nearly no data transmission load.

This command is available as stream-command only (means it is executed at a point in stream that is relative to the other stream commands).

Parameters:

- **n** – the 1-based board instance number as returned by `E1701_set_connection()`
- **flags** – currently only `E1701_COMMAND_FLAG_STREAM` is supported here
- **mask** – specifies which of the bits in "value" have to be used for setting and clearing output data, only these bits that are set to 1 in `mask` are changed according to the given `value`
- **value** – the 8 bit value to be set at digital out port
- **pulses** – specifies how often the output has to be set/cleared
- **delayOn** – the delay (in unit usec) which has to be issued every time after setting the output, the minimal resolution of this value is 0,5 usec
- **delayOff** – the delay (in unit usec) which has to be issued every time after clearing the output, the minimal resolution of this value is 0,5 usec

Return: `E1701_OK` or an `E1701_ERROR_` return code in case of an error

```c
int E1701_digi_read2(const unsigned char n,unsigned int *value)
```

Reads the 8 bit digital input port of Digi IO Extension Board. This is not a stream-command, means it is executed immediately and returns current state of the digital inputs. When marking on the fly is enabled using function `E1701_digi_set_motf()`, digital inputs 0 and 1
optionally also digital inputs 2 and 3 are used for MOTF-encoder and therefore not available as standard inputs. In this case state of these bits is undefined and does not reflect the current input state caused by the external encoder.

Parameters:
- n: the 1-based board instance number as returned by E1701_set_connection()
- value: pointer to a variable where the current digital input state has to be written into.

When the function returns an error code instead of E1701_OK, this value is undefined and can't be used.

Return: E1701_OK or an E1701_ERROR_return code in case of an error

E1701_digi_wait(unsigned char n, unsigned long value, unsigned long mask)

Stop execution and output of data until the given bitpattern was detected at digital inputs of Digit I/O Extension board. Here parameter mask specifies which of the bits at the input have to be checked, they have to be set to 1. These bits within mask that need to be ignored have to be set to 0. Parameter value itself defines the states of the bits that has to be detected at the input to continue processing of data. All bits of value that correspond to bits of mask, that are 0, are ignored.

Parameters:
- n: the 1-based board instance number as returned by E1701_set_connection()
- value: the expected bitpattern at digital input
- mask: specifies which of the input bits and value bits have to be used for comparison

Return: E1701_OK or an E1701_ERROR_return code in case of an error

int E1701_digi_set_motf(unsigned char n, double motfX, double motfY)

Disables or enables marking on-the-fly functionality and specifies factors for X- and Y-direction. When this function is called with values for motfX or motfY greater than 0, marking on-the-fly is enabled and digital inputs 0 and 1 of DigiIO Extension Board are no longer available as general purpose inputs. Now they are used as decoder inputs for a 90 degree phase shifted encoder signal for marking on-the-fly applications. When both parameters motfX and motfY are set to 0, marking on-the-fly is disabled and inputs 0 and 1 no longer work as encoder inputs. When tune flag “2” is set (requires firmware 25 or higher), the two factors for X and Y are assigned to separate encoder inputs. Here factor for X applies to values received on digital inputs 0 and 1 and factor for Y applies to values received on digital inputs 2 and 3. This is not a stream-command, means it switches states of digital inputs 0 and 1 (plus optionally 2 and 3) and marking on-the-fly functionality immediately.

Parameters:
- n: the 1-based board instance number as returned by E1701_set_connection()
- motfX: marking on-the-fly factor for X-direction in unit bits per encoder increment
- motfY: marking on-the-fly factor for Y-direction in unit bits per encoder increment

Return: E1701_OK or an E1701_ERROR_return code in case of an error

int E1701_digi_set_motf_sim(unsigned char n, double motfX, double motfY)

Disables or enables simulated marking on-the-fly functionality and specifies factors for X- and Y-direction. When this function is called with values for motfX or motfY greater than 0, simulated marking on-the-fly is enabled and internal 100 kHz signal generator is used to create static marking on-the-fly pulses in positive direction. A possibly enabled on-the-fly operation using external signals on digital inputs 0 and 1 of Digio Extension Board is disabled. When both parameters motfX and motfY are set to 0, marking on-the-fly simulation is disabled completely. This is not a stream-command, means it enables simulated marking on-the-fly functionality immediately.

Parameters:
- n: the 1-based board instance number as returned by E1701_set_connection()
- motfX: marking on-the-fly factor for X-direction in unit bits suitable for to be simulated movement-speed on
100 kHz encoder counting frequency

motFY - marking on-the-fly factor for Y-direction in unit bits suitable for to be simulated movement speed on 100 kHz encoder counting frequency

Return: E1701_OK or an E1701_ERROR_ -return code in case of an error

```c
int E1701_digi_wait_motf(const unsigned char n, const unsigned int flags, const double dist)
```

Halts the current marking operation for a given distance of the on-the-fly encoder. Different to E1701_delay() this function does not use a time to wait until marking is continued but a distance specified by parameter `dist` and measured by the connected encoder. To use this function marking on-the-fly has to be enabled by calling E1701_digi_set_motf() or E1701_digi_set_motf_sim() before. This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands. It requires firmware version 23 or newer.

Parameters:
- `n` - the 1-based board instance number as returned by E1701_set_connection()
- `flags` - specifies how the distance value is handed over, with E1701_COMMAND_FLAG_MOTF_WAIT_INCS a value in unit "encoder increments" is expected, with E1701_COMMAND_FLAG_MOTF_WAIT_BITS a distance in unit "bits" is expected. In second case the X-on-the-fly factor of a preceding call to E1701_digi_set_motf() or E1701_digi_set_motf_sim() is used.
- `dist` - the distance to wait for until marking has to be completed, the unit of this value is specified with preceding parameter `flags`

Return: E1701_OK or an E1701_ERROR_ -return code in case of an error

```c
int E1701_digi_set_mip_output(unsigned char n,unsigned int value,unsigned int flags)
```

This function can be used to specify which of the digital outputs has to be used for signalling "marking in progress". When `value` is set to 0xFFFFFFFF, this function is disabled and scanner controller card does not provide this signal automatically. When the number of the digital output (in range 0..7) is given as `value`, the related digital output pin is used for "mark in progress" signal.

PLEASE NOTE: here the number (means the count) of one specific output pin has to be given, not a bitpattern specifying one or more pins! During operation the selected "mark in progress" pin is HIGH as long as the scanner is moving and/or the laser is on and/or a delay is processed and when marking parameter are processed between these operations. It becomes LOW as soon as no more marking data are available and current operation is stopped or when scanner is waiting for an external trigger signal (ExtStart).

This is not a stream-command, when it is called it is applied to current configuration immediately.

Parameters:
- `n` - the 1-based board instance number as returned by E1701_set_connection()
- `value` - the number of the digital output to be used for this signal
- `flags` - currently unused, set always to 0 for compatibility

Return: E1701_OK or an E1701_ERROR_ -return code in case of an error

```c
int E1701_digi_set_wet_output(const unsigned char n,const unsigned int value,const unsigned int flags)
```

This function can be used to specify which of the digital outputs has to be used for signalling "waiting for external trigger". When `value` is set to 0xFFFFFFFF, this function is disabled and scanner controller card does not provide this signal automatically. When the number of the digital output (in range 0..7) is given as `value`, the related digital output pin is used for "waiting for external trigger" signal.

PLEASE NOTE: here the number (means the count) of one specific output pin has to be given, not a bitpattern specifying one or more pins! During operation the selected "waiting for external trigger" pin is HIGH as long as the controller is waiting for an external trigger to be applied at ExtStart input. It becomes LOW as soon as this signal has been detected or
when current operation is stopped.
This is not a stream-command, when it is called, it is applied to current configuration immediately.
This function requires firmware version 21 or newer.

Parameters:
- n - the 1-based board instance number as returned by `E1701_set_connection()`
- value - the number of the digital output to be used for this signal
- flags - currently unused, set always to 0 for compatibility

Return: `E1701_OK` or an `E1701_ERROR_`-return code in case of an error

```c
int E1701_write(unsigned char n, unsigned int flags, unsigned int value)
```

Writes some specific data to outputs at E1701D controller. Here `flags` decides which output to use and `value` specifies what has to be written to this output. Additionally `flags` decides weather this is a stream-command (means it is executed at a point in stream that is relative to the other stream commands) or a direct command (means it is executed immediately on calling). This command requires firmware version 17 or newer.

Parameters:
- n - the 1-based board instance number as returned by `E1701_set_connection()`
- flags - handling flags specifying the behaviour of this command, when `E1701_COMMAND_FLAG_STREAM` is set, it is used as stream command, `E1701_COMMAND_FLAG_DIRECT` specifies to execute it immediately and independent on current stream and execution state. Here exactly one of these two flags can be used, it is not allowed to OR-concatenate them. Additionally exactly one of the following flags has to be set to specify which output need to be used to send the `value` to, this flag has to be OR-concatenated with one of the previously described ones:
  - `E1701_COMMAND_FLAG_WRITE_LP8MO` - set or unset MO-output of LP8 Extension Board to a value of 1 or 0
  - `E1701_COMMAND_FLAG_WRITE_LP8LATCH` - set or unset latch-output of LP8 Extension Board to a value of 1 or 0
  - `E1701_COMMAND_FLAG_WRITE_LASERA_GPO` - set or unset LaserA-output to a value of 1 or 0, this option requires the LaserA output to be configured as GPO-output; for details please refer to description of "tune" parameters
  - `E1701_COMMAND_FLAG_WRITE_LASERB_GPO` - set or unset LaserB-output to a value of 1 or 0, this option requires the LaserB output to be configured as GPO-output; for details please refer to description of "tune" parameters
  - `E1701_COMMAND_FLAG_WRITE_LASERGATE` - set or unset LaserGate-output to a value of 1 or 0, this functions should be used with jump or mark operations only since every switch from jump to mark (or vice versa) still sets the LaserGate output automatically and therefore would overwrite own values set with this function
- value - the value to be sent to the output specified by `flags`

Return: `E1701_OK` or an `E1701_ERROR_`-return code in case of an error

```c
int E1701_ana_a123(const unsigned char n, const unsigned short a1, const unsigned short a2, const unsigned short a3)
```

This function can be used to set the analogue outputs AOut1, AOut2 and AOut3 of the E1701A controller.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
- n - the 1-based board instance number as returned by `E1701_set_connection()`
- a1 - a 16 bit value specifying the output value at the AOut1 analogue output, independent from the real hardware resolution here always full 16 bit range has to be used
- a2 - a 16 bit value specifying the output value at the AOut2 analogue output, independent from the real
9.1.1 Writing of stand-alone data

Using E1701 Easy Programming Interface it is also possible to write stand-alone data which are not marked immediately but are stored either locally or on scanner controller’s micro-SD-card. In this mode sending of vector data, scanner- and laser parameters looks exactly the same as for direct operation mode where data are marked immediately. The difference can be found in initialisation (which tells the software to not to mark these data but to store them for later use) and when dynamic data are created.

E1701 supports two types of writing of stand-alone data:

- sending them to the controller via Ethernet or USB connection where they are written to micro-SD-card and
- writing one or more files to the local file system which later have to be copied to the micro-SD-card of the controller manually.

The general procedure for sending stand-alone data to the controller’s micro-SD-card has to look as follows:

1. The controller needs to be in idle-state, means it should not mark and should not have loaded an already existing .EPR file. This can be ensured by calling stand-alone command clepr with a filename for a file that does not exists on micro-SD-card. For more details please refer to “8.2 Stand-Alone Control Commands”
2. Configure the connection to E1701 controller by calling E1701_set_connection(), the returned board instance number has to be used for all following function calls.
3. Enable stand-alone write mode and specify the filename of the .EPR file to be created on micro-SD-card by calling E1701_set_filepath() with mode E1701_FILEMODE_SEND.
4. Send all laser- and scanner-parameters as well as vector data as usual.
5. Optionally: send information about dynamic contents of the .EPR file to be created by calling E1701_dynamic_data() optionally followed by some vector data followed by an other call to function E1701_dynamic_data() which ends this section of dynamic data (please refer function description below for details).
6. Wait until E1701_get_card_state() returns "busy"
7. Wait until E1701_get_card_state() returns "idle" or an error
8. End data transmission and finish created file by calling E1701_close().

The general procedure for writing stand-alone data to the local filesystem has to look as follows:

1. Since writing of local data does not require a working connection to the controller card, it does not need to be configured and the special board instance number 0 has to be used for all following function calls.
2. Enable stand-alone write mode and specify the filename of the .EPR file to be created by calling E1701_set_filepath() with mode E1701_FILEMODE_LOCAL.
3. Send all laser- and scanner-parameters as well as vector data as usual.
4. Optionally: send information about dynamic contents of the .EPR file to be created by calling E1701_dynamic_data() optionally followed by some vector data followed by an other call to function E1701_dynamic_data() which ends this section of dynamic data (please refer function description below for details).
5. End data transmission and finish created file by calling E1701_close().

The functions which are specific to writing of stand-alone data have to be used as follows:

int E1701_set_filepath(unsigned char n, const char *fname, unsigned int mode)

This function enables operation mode where all following data are not marked immediately but written into an .EPR stand-alone file. This mode stays active until next call of E1701_close(). It has to be called prior to E1701_load_correction(). Valid parameters and their meaning depends on the usage scenario:
• when sending stand-alone data to a connected controller which writes the .EPR file to the micro-SD-card directly:
  n – the 1-based board instance number as returned by E1701_set_connection()
  fname – name of the file as it has to appear on micro-SD-card of the controller in style
  "0:/filename.epr" where "0:" is a fixed prefix specifying the micro-SD-card, "filename" is a free to
  choose name with recommended 8 characters at max and ".epr" is a fixed, mandatory file extension
  specifying an E1701 stand-alone file
  mode – set to E1701_FILEMODE_SEND to specify the data have to be sent to the controller
• when writing stand-alone data to the local filesystem (no controller card directly involved):
  n – board instance number, has to be set to 0 (as well as for all other function calls in this mode)
  fname – name of the file to be written, this has to be a valid path to a location on a local filesystem
  which is writable and needs to have file extension ".epr"
  mode – set to E1701_FILEMODE_LOCAL to specify the data have to be written locally

Return: E1701_OK or an E1701_ERROR_return code in case of an error

```c
int E1701_dynamic_data(unsigned char n, struct oapc_bin_struct_dyn_data *dynData)
{
    This function can be used to write dynamic data such as texts, serial numbers, barcodes which later can
    be changed during operation in stand-alone mode.
    This function always has to be called in fixed sequences:
    1. jump to the start position of the dynamic element by calling E1701_jump_abs()
    2. first call of E1701_dynamic_data(n, dynData) with a valid dynData parameter describing the
       dynamic content and its capabilities
    3. optionally and dependent on type of dynamic data that have to be created: some vector data which
       belong to the dynamic content and are required to build it up
    4. second call of E1701_dynamic_data(n, NULL) with NULL handed over for parameter dynData to
       finish this element

    A stand-alone file can contain up to ten dynamic data elements. So this function can be called up to ten times to
    create a new element on each call.

    When this function is called, beside the .EPR-file an additional .DAT file is created which contains some specific
    data. During operation in stand-alone mode an other file with the same name and with extension .SER may be
    created which contains counting information of an included serial number. All these files belong together and
    deleting one or more of them may lead to unexpected results. When writing the data to local filesystem it also
    has to be ensured both, the .EPR and the .DAT file are copied to the controller later.

    The structure oapc_bin_struct_dyn_data is defined in file "oapc_libio.h" which is part of the OpenSDK.
    The general usage is described in OpenSDK manual, both are available for download at

    For E1701 scanner controller card following specific parameters and features of this structure can be used:

    Independent from what kind of dynamic element has to be created, following members of structure
    oapc_bin_struct_dyn_data always have to be filled with data:

    UID – and unique identifier which can be created out of a plain, human readable text which should be unique
    too and later can be used to access this specific element via stand-alone control commands; this
    identifier has to created out of the 8 bit ASCII character using following CRC-function:
    #define POLY 0x82f63b78
    /* CRC-32 (Ethernet, ZIP, etc.) polynomial in reversed bit order. */
    unsigned int crc32b(const char *buf)
    {
        int k;
        unsigned int crc=0xFFFFFFFF;
        size_t len=strlen(buf);
        
        while (len--)
        {
            
```

```
crc ^= buf++;
for (k=0; k<8; k++)
    crc = crc & 1 ? (crc >> 1)^ POLY : crc >> 1;
return ~crc;
```

uScaleX – scaling factor in X-direction in unit 1/1000000
uScaleY – scaling factor in Y-direction in unit 1/1000000
res1a, res1b, res2, res3, res4, res5, res6, res7 – these members are reserved for later use and all have to be set to 0

Every dynamic element can be a serial number. In such a case the serial number part of structure oapc_bin_struct_dyn_data has to be filled with data:

fmtString – an ASCII text with a maximum length of DYN_DATA_MAX_STRING_LENGTH describing the format of the serial number/date/time in the dynamic element, here the same notation has to be used as it is known from the serial number input element of BeamConstruct (please refer to the related manual)

snBeatCount – specifies how much numbers of mark operations have to elapse before the serial number has to be incremented, here a value of 1 has to be given to increment on every operation

snBeatOffset – specifies a counting offset for the beat count parameter

snIncrement – specifies the step width by which a serial number has to be incremented

snNumericBase – the numeric base of the serial numbers to be displayed, default is 10 for decimal numbers

snResetAt – the value at which the serial number has to be reset to it’s initial value; set to a timestamp (in unit day of week/date/seconds) when it has to be reset at a given time

snFlags – a set of OR-concatenated flags which further specifies handling of the serial number:
0x0002 – reset the serial number at a specific counting value
0x0004 – reset the serial number at a specific day of the week
0x0008 – reset the serial number at a specific date
0x0010 – reset the serial number at a specific time of the day

snStartValue – the initial- and reset-to-value of the serial number

snMinDigits – the minimum number of digits the serial number has to consist of

timeOffset – a static offset (in unit seconds) to be added to the time-part of the current element

Dynamic text elements additionally need to fill following parts of the same structure oapc_bin_struct_dyn_data:

fmtString – an ASCII text with a maximum length of DYN_DATA_MAX_STRING_LENGTH which contains the text to be shown and which can be changed by appropriate stand-alone commands later; when used in combination with serial number data, here a format-string has to be given as described above

type – a number which specifies the font to be used for creating the dynamic texts, here one of following values can be used:
0x00000001 – use "Rect Single" laser font
0x00000002 – use "Rect Double" laser font
0x00000003 – use "Roman Simple" laser font
0x00000004 – use "Roman Double" laser font
0x00000005 – use "Script Simple" laser font
0x00000006 – use "Script Double" laser font
0x00000007 – use "Script Complex" laser font
0x00000008 – use "Times Simple" laser font
0x00000009 – use "Times Bold" laser font
0x0000000A – use "Times Italic" laser font
0x0000000B – use "Times Italic Bold" laser font

flags – some OR-concatenated flags which specify orientation, alignment and style of the text to be generated, here no two flags of same type are allowed to be combined which would conflict with each
Dynamic DataMatrix barcode elements require vector data to be sent between two calls of function E1701_dynamic_data(), these vector data describe the pattern which has to be marked to create one single element (means square) of the DataMatrix barcode. Such an element needs to incorporate all that is needed including laser- and scannerdata as well as vector data for outline and possible hatches. During stand-alone operation the barcode itself is created by combining these single elements at these positions, where a bit (=square) has to be set.

Additionally following data of the structure oapc_bin_struct_dyn_data need to be filled for this type of element:

- **fmtString** – an ASCII text with a maximum length of DYN_DATA_MAX_STRING_LENGTH which contains the text to be encoded as DataMatrix barcode and which can be changed by appropriate stand-alone commands later; when used in combination with serial number data, here a format-string has to be given as described above.
- **type** – set to 71 for DataMatrix barcode.
- **flags** – some OR-concatenated flags which further specify how the barcode has to be created, currently only one flag is supported:
  - 0x0001 – create a square-shaped DataMatrix barcode instead of a rectangular one.
- **param1** – set to 0.
- **param2** – set to -1.
- **param3** – specifies the size to be generated (in range 2..30) and implicitly the error correction level.
- **quietZone** – zone the barcode has to be surrounded with, the value given here is the multiple of the width of a single token multiplied with 1000.

### 9.1.1.1 Example

Following a (simplified) example in some pseudo-code is given which demonstrates the correct usage of the programming interface to write stand-alone data. The laser- and scanner-parameters are dropped in this example since they are not specific to this operation mode and always have to be set.

Example: A serial number in format "000/hh/mm" where "000" is a continuously increased number, "hh" is the current hour and “mm” is the current minute has to be encoded into a DataMatrix barcode which has a size of 25x25 mm and is positioned at -30x30 mm within a 100x100 mm working area that itself is aligned to coordinates -50, 50.

1. not shown here: initialisation of libe1701 (with evaluation of parameter boardIdx), sending of default scanner and laser data as usual.
2. E1701_jump_abs(boardIdx, -20132659, 20132659, 0) // jump to the starting position of the DataMatrix barcode to be created.
3. E1701_dynamic_data(boardIdx, dynData) // initiate the dynamic data sequence, here the members of dynData are set to following values:
   - **UID** = 2340633892 – CRC-value of element name “Barcode 1”
   - **fmtString** = "$S/%I/%M" – display serial number, hour and minute.
**9.1.2 Error Codes**

Most of the functions described above can return an error code in case an operation could not be completed successfully for any reason. So when it does not return with **E1701_OK** the error code informs about the reason for failure:

- **E1701_ERROR_INVALID_CARD** - a wrong or illegal card number was specified with function parameter
- **E1701_ERROR_NO_CONNECTION** - a connection to card could not be established
- **E1701_ERROR_NO_MEMORY** - there is not enough memory available on the host to perform the requested operation
- **E1701_ERROR_UNKNOWN_FW** - card is running an unknown and/or incompatible firmware version
- **E1701_ERROR_TRANSMISSION** - data transmission to card failed
- **E1701_ERROR_FILEOPEN** - opening of a file failed
- **E1701_ERROR_FILEWRITE** - writing of data into a file failed
- **E1701_ERROR_BORD_NA** - a base- or extension board that would be required for a function is not available
- **E1701_ERROR_INVALID_DATA** - data or parameters handed over to a function are invalid, out of range or illegal in current context
- **E1701_ERROR_UNKNOWN_BOARD** - trying to access a controller board that is not a suitable controller
- **E1701_ERROR_FILENAME** - a file name handed over to a function was illegal, it is either too long, has an illegal or too long file extension, comes with too much sub-directories or contains illegal characters
- **E1701_ERROR** - an other, unspecified error occurred
- **E1701_ERROR_NOT_SUPPORTED** - the requested feature or function is not supported by the current firmware version

**9.2 RTC4 Compatibility Functions**

Besides the easy programming interface described above, a bunch of additional functions is provided that are compatible to the ones known from RTC4 scanner controller card. So to use E1701 scanner card with existing code that supports the RTC4 scanner controller, following few steps have to be done:

- Insert a call to **E1701_set_connection()** into existing RTC4-code as very first in order to specify the communication connection for E1701 card (this is the only exception where an E1701 Easy Interface Function should be used together with an RTC Compatibility Function)
- Recompile the existing RTC4-application so that it uses e1701inter.DLL/libe1701inter.so instead of...
RTC4DLL.dll/libslrtc4.so

In case E1701 card has to be operated with default settings, no recompilation is necessary, here e1701inter.dll / libe1701inter.so just has to be renamed to RTC4DLL.dll / libslrtc4.so.

Since most relevant RTC4 functions are already provided, majority of existing applications should work now without any further modifications. Due to the completely different concept of E1701 scanner controller, there are some differences to the original RTC4 programming interface which should be checked in case of some errors:

- all RTC4 functions that exist as list- and non-list-commands are treated like a list command
- list switch commands are ignored since E1701 does not make use of separated lists internally
- output of already sent marking data is started on calls to `n_execute_list()`, `execute_list()`, `n_set_end_of_list()` or `set_end_of_list()`
- some functions are not implemented or always return a default value (please refer below for a list of not implemented RTC4 functions)

Following functions are specific to RTC4 hardware or do not make sense in relation to E1701 scanner card and therefore aren't supported. These functions exist in libe1701inter, but do not provide any functionality:

```c
n_load_z_table()
load_z_table()
n_set_defocus_list()
set_defocus_list()
n_set_offset_list()
set_offset_list()
n_laser_on_list()
laser_on_list()
n_set_list_jump()
set_list_jump()
n_set_input_pointer()
set_input_pointer()
n_list_call()
list_call()
n_list_return()
list_return()
n_z_out_list()
z_out_list()
n_timed_jump_abs()
timed_jump_abs()
n_timed_mark_abs()
timed_mark_abs()
n_timed_jump_rel()
timed_jump_rel()
n_timed_mark_rel()
timed_mark_rel()
n_set_fly_rot()
set_fly_rot()
n_fly_return()
fly_return()
n_calculate_fly()
calculate_fly()
n_select_cor_table_list()
select_cor_table_list()
n_set_wait()
set_wait()
n_simulate_ext_start()
simulate_ext_start()
n_set_pixel_line()
set_pixel_line()
n_set_pixel()
set_pixel()
n_set_extstartpos_list()
set_extstartpos_list()
```
n_laser_signal_on_list()
laser_signal_on_list()
n_laser_signal_off_list()
laser_signal_off_list()
n_set_io_cond_list()
set_io_cond_list()
n_clear_io_cond_list()
clear_io_cond_list()
n_list_jump_cond()
list_jump_cond()
n_list_call_cond()
list_call_cond()
n_save_and_restart_timer()
save_and_restart_timer()
n_set_ext_start_delay_list()
set_ext_start_delay_list()
n_set_trigger()
set_trigger()
n_arc_rel()
arc_rel()
n_arc_abs()
arc_abs()
drilling()
regulation()
flyline()
n_get_input_pointer()
get_input_pointer()
n_get_marking_info()
get_marking_info()
n_auto_change_pos()
auto_change_pos()
aut_change()
n_start_loop()
start_loop()
n_quit_loop()
quit_loop()
n_write_da_2()
write_da_2()
n_set_max_counts()
set_max_counts()
n_set_offset()
set_offset()
n_disable_laser()
disable_laser()
n_enable_laser()
enable_laser()
n_stop_list()
stop_list()
n_restart_list()
restart_list()
n_get_xyz_pos()
get_xyz_pos()
n_get_xy_pos()
get_xy_pos()
n_select_list()
select_list()
n_z_out()
z_out()
n_laser_signal_on()
laser_signal_on()
n_laser_signal_off()
laser_signal_off()
n_set_delay_mode()
set_delay_mode()
n_set_piso_control()
set_piso_control()
n_select_status()
select_status()
n_get_encoder()
get_encoder()
n_select_cor_table()
select_cor_table()
n_execute_at_pointer()
execute_at_pointer()
n_get_head_status()
get_head_status()
n_simulate_encoder()
simulate_encoder()
n_set_hi()
set_hi()
n_release_wait()
release_wait()
n_get_wait_status()
get_wait_status()
n_set_ext_start_delay()
set_ext_start_delay()
n_home_position()
home_position()
n_set_rot_center()
set_rot_center()
n_read_ad_x()
read_ad_x()
n_read_pixel_ad()
read_pixel_ad()
n_get_z_distance()
get_z_distance()
n_get_time()
get_time()
n_set_defocus()
set_defocus()
n_set_softstart_mode()
set_softstart_mode()
n_set_softstart_level()
set_softstart_level()
n_control_command()
control_command()
load_cor()
load_pro()
n_get_serial_number()
get_serial_number()
n_get_serial_number_32()
get_serial_number_32()
get_hi_data()
get_hi_data()
auto_cal()
auto_cal()
n_get_list_space()
get_list_space()
teachin()
n_get_value()
get_value()
set_duty_cycle_table()
move_to()
getmemory()
n_get_waveform()
get_waveform()
9.3 USC1/2 Compatibility Functions (SCI interface)

Beside the easy programming interface described above a bunch of additional functions is provided that are compatible to the ones known from SCI programming interface used for USC1/2 scanner controller card. So to use E1701 scanner card with existing code that supports the USC1 or USC2 scanner controller, following steps have to be done:

- insert a call to `E1701_set_connection()` into existing code as very first in order to specify the communication connection for E1701 card (this is the only exception where an E1701 Easy Interface Function should be used together with an USC1/2 Compatibility Function)
- recompile the existing SCI-application so that it uses e1701inter.DLL instead of sc_optic.dll

In case E1701 card has to be operated with default settings, no recompilation is necessary, here e1701inter.dll just has to be renamed to sc_optic.dll.

Since most relevant SCI functions are already provided, majority of existing applications should work now without any further modifications. Due to the different concept of E1701 scanner controller, there are some differences to the original SCI programming interface which should be checked in case of troubles:

- all USC1/2 functions/types that exist as stream- and non-stream-variants are treated like a stream command
- output of already sent marking data is started on call to `ScSCIFlush()` latest
- some functions are not implemented or always return a default value (please refer below of a list of not implemented SCI functions)

Following functions are specific to USC1/2 hardware or do not make sense in relation to E1701 scanner card and therefore aren’t supported. These functions exist in libe1701inter, but do not provide any functionality:

```c
long ScSCISetContinuousMode()
long ScSCIGetContinuousMode()
long ScSCIDevicePixelLine()
long ScSCIRasterPixelLine()
long ScSCIRasterStart()
long ScSCIRasterEnd()
long ScSCIGetDeviceName()
long ScSCIGetDeviceCaps()
long ScSCIGetDeviceData()
long ScSCISetDeviceData()
long ScSCIGetExternalTrigger()
long ScSCISetExternalTriggerCount()
long ScSCIGetExternalTriggerCount()
long ScSCISetEnableHead()
long ScSCIGetEnableHead()
long ScSCISetZField()
long ScSCIGetZField()
long ScSCIGetZGain()
long ScSCISetZGain()
long ScSCIGetZHomePosition()
long ScSCIGetZHomePosition()
long ScSCISetZHomePosition()
long ScSCIGetZOffset()
long ScSCISetZOffset()
long ScSCIGetZWorkingArea()
long ScSCIGetZWorkingArea()
long ScSCIGetHomeJump()
long ScSCISetHomeJump()
```
Beside the easy programming interface described above, a bunch of additional functions is offered that provide an OLSC v2 compatible programming interface. So to use E1701 scanner card with existing code that supports the Open Laser Show Controller interface, following steps have to be done:

- insert a call to `E1701_set_connection()` into existing code as very first in order to specify the communication connection for E1701 card (this is the only exception where an E1701 Easy Interface Function should be used together with an OLSC Compatibility Function)
- recompile the existing OLSC-application so that it uses e1701inter.DLL

In case E1701 card has to be operated with default IP 192.168.2.254, none of the above steps and no recompilation is necessary, here e1701inter.dll just has to be renamed to the name of the DLL of the target device.

Since most relevant OLSC v2 functions are already provided, majority of existing applications should work now without any further modifications. Due to the different concept of E1701 scanner controller, there are some differences to the original OLSC v2 programming interface which should be checked in case of troubles. So some functions are not implemented and therefore always return a default value (please check behaviour of these functions related to your application).
APPENDIX A – Wiring between E1701 and IPG YLP Series Type B, B1 and B2 fiber laser

PLEASE NOTE: this wiring scheme is a non-binding policy, it may be incorrect due to changes in connected hardware. So in every case please check this table against specification and wiring documentation of the used laser!

Variant using E1701A or E1701D Baseboard, LP8 Extension Board and optional Digi I/O Extension Board for laser alarms and pilot laser.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Board</th>
<th>Connector / Pin</th>
<th>IPG Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP0</td>
<td>LP8 Extension Board</td>
<td>Pin 1</td>
<td>Pin 1</td>
</tr>
<tr>
<td>LP1</td>
<td>LP8 Extension Board</td>
<td>Pin 3</td>
<td>Pin 2</td>
</tr>
<tr>
<td>LP2</td>
<td>LP8 Extension Board</td>
<td>Pin 5</td>
<td>Pin 3</td>
</tr>
<tr>
<td>LP3</td>
<td>LP8 Extension Board</td>
<td>Pin 7</td>
<td>Pin 4</td>
</tr>
<tr>
<td>LP4</td>
<td>LP8 Extension Board</td>
<td>Pin 9</td>
<td>Pin 5</td>
</tr>
<tr>
<td>LP5</td>
<td>LP8 Extension Board</td>
<td>Pin 11</td>
<td>Pin 6</td>
</tr>
<tr>
<td>LP6</td>
<td>LP8 Extension Board</td>
<td>Pin 13</td>
<td>Pin 7</td>
</tr>
<tr>
<td>LP7</td>
<td>LP8 Extension Board</td>
<td>Pin 15</td>
<td>Pin 8</td>
</tr>
<tr>
<td>MO / Master Oscillator</td>
<td>LP8 Extension Board</td>
<td>Pin 8</td>
<td>Pin 18</td>
</tr>
<tr>
<td>LP8 Latch</td>
<td>LP8 Extension Board</td>
<td>Pin 17</td>
<td>Pin 9</td>
</tr>
<tr>
<td>LaserA / Frequency</td>
<td>LP8 Extension Board</td>
<td>Pin 22</td>
<td>Pin 20</td>
</tr>
<tr>
<td>Alarm, one of DIn0…DIn7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 4, 6, 8, 10, 12, 14, 16 or 18</td>
<td>Pin 16</td>
</tr>
<tr>
<td>Alarm, one of DIn0..DIn7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 4, 6, 8, 10, 12, 14, 16 or 18</td>
<td>Pin 21</td>
</tr>
<tr>
<td>Pilot Laser, one of DOut0..DOut7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 3, 5, 7, 9, 11, 13, 15 or 17</td>
<td>Pin 22 *)</td>
</tr>
</tbody>
</table>

*) may require additional power driver since some laser variants consume a current at this input which is higher than the maximum output allowed

In this wiring-scheme no GND-connections are listed, they have to be added in order to get valid and working connections.
APPENDIX B – Wiring between E1701 and IPG YLP Series Type E fiber laser

PLEASE NOTE: this wiring scheme is a non-binding policy, it may be incorrect due to changes in connected hardware. So in every case please check this table against specification and wiring documentation of the used laser!

Variant using E1701A or E1701D Baseboard, LP8 Extension Board and Digi I/O Extension Board for laser alarms and pilot laser with support for APD index setting via DB-25 serial data interface

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Board</th>
<th>Connector / Pin</th>
<th>IPG Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP0</td>
<td>LP8 Extension Board</td>
<td>Pin 1</td>
<td>Pin 1</td>
</tr>
<tr>
<td>LP1</td>
<td>LP8 Extension Board</td>
<td>Pin 3</td>
<td>Pin 2</td>
</tr>
<tr>
<td>LP2</td>
<td>LP8 Extension Board</td>
<td>Pin 5</td>
<td>Pin 3</td>
</tr>
<tr>
<td>LP3</td>
<td>LP8 Extension Board</td>
<td>Pin 7</td>
<td>Pin 4</td>
</tr>
<tr>
<td>LP4</td>
<td>LP8 Extension Board</td>
<td>Pin 9</td>
<td>Pin 5</td>
</tr>
<tr>
<td>LP5</td>
<td>LP8 Extension Board</td>
<td>Pin 11</td>
<td>Pin 6</td>
</tr>
<tr>
<td>LP6</td>
<td>LP8 Extension Board</td>
<td>Pin 13</td>
<td>Pin 7</td>
</tr>
<tr>
<td>LP7</td>
<td>LP8 Extension Board</td>
<td>Pin 15</td>
<td>Pin 8</td>
</tr>
<tr>
<td>MO / Master Oscillator</td>
<td>LP8 Extension Board</td>
<td>Pin 8</td>
<td>Pin 18</td>
</tr>
<tr>
<td>LP8 Latch</td>
<td>LP8 Extension Board</td>
<td>Pin 17</td>
<td>Pin 9</td>
</tr>
<tr>
<td>Laser Gate / Frequency</td>
<td>LP8 Extension Board</td>
<td>Pin 22</td>
<td>Pin 20</td>
</tr>
<tr>
<td>Alarm, one of DIn0..DIn7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 4, 6, 8, 10, 12, 14, 16 or 18</td>
<td>Pin 16</td>
</tr>
<tr>
<td>Alarm, one of DIn0..DIn7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 4, 6, 8, 10, 12, 14, 16 or 18</td>
<td>Pin 21</td>
</tr>
<tr>
<td>Pilot Laser, one of DOut0, DOut1, DOut5..DOut7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 3, 5, 13, 15 or 17</td>
<td>Pin 22 1)</td>
</tr>
<tr>
<td>Serial Enable</td>
<td>Digi I/O Extension Board</td>
<td>Pin 7</td>
<td>Pin 24 2)</td>
</tr>
<tr>
<td>Serial Clock</td>
<td>Digi I/O Extension Board</td>
<td>Pin 9</td>
<td>Pin 13 2)</td>
</tr>
<tr>
<td>Serial Data</td>
<td>Digi I/O Extension Board</td>
<td>Pin 11</td>
<td>Pin 10 2)</td>
</tr>
</tbody>
</table>

1) may require additional power driver since some laser variants consume a current at this input which is higher than the maximum output allowed
2) serial data transmission requires firmware version 28 or newer

In this wiring-scheme no GND-connections are listed, they have to be added in order to get valid and working connections.
APPENDIX C – Wiring between E1701 and IPG YLR Series laser

PLEASE NOTE: this wiring scheme is a non-binding policy, it may be incorrect due to changes in connected hardware. So in every case please check this table against specification and wiring documentation of the used laser!

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Board</th>
<th>Connector / Pin</th>
<th>IPG Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOut0</td>
<td>LP8 Extension Board</td>
<td>Pin 10</td>
<td>Pin 12</td>
</tr>
<tr>
<td>MO / Master Oscillator</td>
<td>LP8 Extension Board</td>
<td>Pin 8</td>
<td>Pin 18</td>
</tr>
<tr>
<td>Laser Gate / Modulation</td>
<td>LP8 Extension Board</td>
<td>Pin 26</td>
<td>Pin 15</td>
</tr>
<tr>
<td>Pilot Laser, one of DOut0..DOut7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 3, 5, 7, 9, 11, 13, 15 or 17</td>
<td>Pin 17</td>
</tr>
</tbody>
</table>

1) maximum analogue output voltage of LP8 extension is limited to 5V while this laser type expects 0..10V range. So this voltage needs to be doubled by some external equipment, elsewhere the laser can be driven with a maximum of 50% power only.

In this wiring-scheme no GND-connections are listed, they have to be added in order to get valid and working connections.
APPENDIX D – Wiring between E1701 and JPT YDFLP series fiber laser (“MOPA”) or IPG YLP Series Type D fiber laser

PLEASE NOTE: this wiring scheme is a non-binding policy, it may be incorrect due to changes in connected hardware. So in every case please check this table against specification and wiring documentation of the used laser!

Variant using E1701A or E1701D Baseboard, LP8 Extension Board and optional Digi I/O Extension Board for laser alarms and pilot laser.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Board</th>
<th>Connector / Pin</th>
<th>JPT D-SUB25 Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP0</td>
<td>LP8 Extension Board</td>
<td>Pin 1</td>
<td>Pin 1</td>
</tr>
<tr>
<td>LP1 / serial data</td>
<td>LP8 Extension Board</td>
<td>Pin 3</td>
<td>Pin 2 ¹)</td>
</tr>
<tr>
<td>LP2 / serial clock</td>
<td>LP8 Extension Board</td>
<td>Pin 5</td>
<td>Pin 3 ²)</td>
</tr>
<tr>
<td>LP3</td>
<td>LP8 Extension Board</td>
<td>Pin 7</td>
<td>Pin 4</td>
</tr>
<tr>
<td>LP4</td>
<td>LP8 Extension Board</td>
<td>Pin 9</td>
<td>Pin 5</td>
</tr>
<tr>
<td>LP5</td>
<td>LP8 Extension Board</td>
<td>Pin 11</td>
<td>Pin 6</td>
</tr>
<tr>
<td>LP6</td>
<td>LP8 Extension Board</td>
<td>Pin 13</td>
<td>Pin 7</td>
</tr>
<tr>
<td>LP7</td>
<td>LP8 Extension Board</td>
<td>Pin 15</td>
<td>Pin 8</td>
</tr>
<tr>
<td>MO / Master Oscillator</td>
<td>LP8 Extension Board</td>
<td>Pin 8</td>
<td>Pin 18</td>
</tr>
<tr>
<td>LaserA / Frequency</td>
<td>LP8 Extension Board</td>
<td>Pin 22</td>
<td>Pin 20</td>
</tr>
<tr>
<td>Laser Gate / Modulation</td>
<td>LP8 Extension Board</td>
<td>Pin 26</td>
<td>Pin 19</td>
</tr>
<tr>
<td>LaserB / serial enable</td>
<td>LP8 Extension Board</td>
<td>Pin 19</td>
<td>Pin 22 ¹) ²)</td>
</tr>
<tr>
<td>Alarm, one of DIn0...DIn7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 4, 6, 8, 10, 12, 14, 16 or 18</td>
<td>Pin 16</td>
</tr>
<tr>
<td>Alarm, one of DIn0...DIn7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 4, 6, 8, 10, 12, 14, 16 or 18</td>
<td>Pin 21</td>
</tr>
<tr>
<td>Pilot Laser, one of DOut0...DOut7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 3, 5, 7, 9, 11, 13, 15 or 17</td>
<td>Pin 22 ¹)</td>
</tr>
</tbody>
</table>

¹) for details regarding double-usage of this pin, please refer to the manual of the laser
²) serial data transmission requires firmware version 28 or newer

In this wiring-scheme no GND-connections are listed, they have to be added in order to get valid and working connections.
APPENDIX E – Wiring between E1701 and SPI G4 Pulsed Fibre Laser series

PLEASE NOTE: this wiring scheme is a non-binding policy, it may be incorrect due to changes in connected hardware. So in every case please check this table against specification and wiring documentation of the used laser!

Variant 1: waveform selected via LP8 outputs of LP8 Extension Board, simmer, power and extended parameter control via laser controller plug in/serial interface:

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Board</th>
<th>Connector / Pin</th>
<th>SPI Laser Connector Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP0</td>
<td>LP8 Extension Board</td>
<td>Pin 1</td>
<td>Pin 17</td>
</tr>
<tr>
<td>LP1</td>
<td>LP8 Extension Board</td>
<td>Pin 3</td>
<td>Pin 18</td>
</tr>
<tr>
<td>LP2</td>
<td>LP8 Extension Board</td>
<td>Pin 5</td>
<td>Pin 19</td>
</tr>
<tr>
<td>LP3</td>
<td>LP8 Extension Board</td>
<td>Pin 7</td>
<td>Pin 20</td>
</tr>
<tr>
<td>LP4</td>
<td>LP8 Extension Board</td>
<td>Pin 9</td>
<td>Pin 51</td>
</tr>
<tr>
<td>LP5</td>
<td>LP8 Extension Board</td>
<td>Pin 11</td>
<td>Pin 52</td>
</tr>
<tr>
<td>LP6</td>
<td>LP8 Extension Board</td>
<td>Pin 13</td>
<td>Pin 53</td>
</tr>
<tr>
<td>LP7</td>
<td>LP8 Extension Board</td>
<td>Pin 15</td>
<td>Pin 54</td>
</tr>
<tr>
<td>MO / Laser Enable</td>
<td>LP8 Extension Board</td>
<td>Pin 8</td>
<td>Pin 7</td>
</tr>
<tr>
<td>LP8 Latch</td>
<td>LP8 Extension Board</td>
<td>Pin 17</td>
<td>Pin 23</td>
</tr>
<tr>
<td>Laser A / Pulse Trigger</td>
<td>LP8 Extension Board</td>
<td>Pin 22</td>
<td>Pin 47</td>
</tr>
<tr>
<td>Alarm, one of DIn0…DIn7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 4, 6, 8, 10, 12, 14, 16 or 18</td>
<td>Pin 9</td>
</tr>
<tr>
<td>Pilot Laser, one of DOut0...DOut7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 3, 5, 7, 9, 11, 13, 15 or 17</td>
<td>Pin 6</td>
</tr>
</tbody>
</table>

Variant 2: waveform selected via digital outputs of Digi I/O Extension Board, simmer, power and extended parameter control via laser controller plug in/serial interface:

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Board</th>
<th>Connector / Pin</th>
<th>SPI Laser Connector Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOut0</td>
<td>Digi I/O Extension Board</td>
<td>Pin 3</td>
<td>Pin 17</td>
</tr>
<tr>
<td>DOut1</td>
<td>Digi I/O Extension Board</td>
<td>Pin 5</td>
<td>Pin 18</td>
</tr>
<tr>
<td>DOut2</td>
<td>Digi I/O Extension Board</td>
<td>Pin 7</td>
<td>Pin 19</td>
</tr>
<tr>
<td>DOut3</td>
<td>Digi I/O Extension Board</td>
<td>Pin 9</td>
<td>Pin 20</td>
</tr>
<tr>
<td>DOut4</td>
<td>Digi I/O Extension Board</td>
<td>Pin 11</td>
<td>Pin 51</td>
</tr>
<tr>
<td>DOut5</td>
<td>Digi I/O Extension Board</td>
<td>Pin 13</td>
<td>Pin 52</td>
</tr>
<tr>
<td>DOut6</td>
<td>Digi I/O Extension Board</td>
<td>Pin 15</td>
<td>Pin 53</td>
</tr>
<tr>
<td>DOut7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 17</td>
<td>Pin 23</td>
</tr>
<tr>
<td>Alarm, one of DIn0…DIn7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 4, 6, 8, 10, 12, 14, 16 or 18</td>
<td>Pin 9</td>
</tr>
<tr>
<td>Pilot Laser, one of LP8_0...P8_7</td>
<td>LP8 Extension Board</td>
<td>Pin 1, 3, 5, 7, 9, 11, 13 or 15</td>
<td>Pin 6</td>
</tr>
<tr>
<td>MO / Laser Enable</td>
<td>LP8 Extension Board</td>
<td>Pin 8</td>
<td>Pin 7</td>
</tr>
<tr>
<td>Laser A / Pulse Trigger</td>
<td>LP8 Extension Board</td>
<td>Pin 22</td>
<td>Pin 47</td>
</tr>
<tr>
<td>Laser Gate / Modulation</td>
<td>LP8 Extension Board</td>
<td>Pin 26</td>
<td>Pin 5</td>
</tr>
</tbody>
</table>
Variant 3: waveform selection, simmer, power and extended parameter control via laser controller plug in/serial interface:

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Board</th>
<th>Connector / Pin</th>
<th>SPI Laser Connector Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>MO / Laser Enable</td>
<td>LP8 Extension Board</td>
<td>Pin 8</td>
<td>Pin 7</td>
</tr>
<tr>
<td>LaserA / Pulse Trigger</td>
<td>LP8 Extension Board</td>
<td>Pin 22</td>
<td>Pin 47</td>
</tr>
<tr>
<td>Laser Gate / Modulation</td>
<td>LP8 Extension Board</td>
<td>Pin 26</td>
<td>Pin 5</td>
</tr>
<tr>
<td>Alarm, one of DIn0…DIn7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 4, 6, 8, 10, 12, 14, 16 or 18</td>
<td>Pin 9</td>
</tr>
<tr>
<td>Pilot Laser, one of LP8_0…LP8_7 or one of DOut0…DOut7</td>
<td>LP8 Extension Board or Digi I/O Extension Board</td>
<td>Pin 1, 3, 5, 7, 9, 11, 13 or 15 or Pin 3, 5, 7, 9, 11, 13, 15 or 17</td>
<td>Pin 6</td>
</tr>
</tbody>
</table>

In these wiring-schemes no GND-connections are listed, they have to be added in order to get valid and working connections.
**APPENDIX F – Wiring between E1701 and Raycus fiber laser**

PLEASE NOTE: this wiring scheme is a non-binding policy, it may be incorrect due to changes in connected hardware. So in every case please check this table against specification and wiring documentation of the used laser!

Variant using E1701A or E1701D Baseboard, LP8 Extension Board and optional Digi I/O Extension Board for laser alarms and pilot laser.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Board</th>
<th>Connector / Pin</th>
<th>Raycus DB25 Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP0</td>
<td>LP8 Extension Board</td>
<td>Pin 1</td>
<td>Pin 1</td>
</tr>
<tr>
<td>LP1</td>
<td>LP8 Extension Board</td>
<td>Pin 3</td>
<td>Pin 2</td>
</tr>
<tr>
<td>LP2</td>
<td>LP8 Extension Board</td>
<td>Pin 5</td>
<td>Pin 3</td>
</tr>
<tr>
<td>LP3</td>
<td>LP8 Extension Board</td>
<td>Pin 7</td>
<td>Pin 4</td>
</tr>
<tr>
<td>LP4</td>
<td>LP8 Extension Board</td>
<td>Pin 9</td>
<td>Pin 5</td>
</tr>
<tr>
<td>LP5</td>
<td>LP8 Extension Board</td>
<td>Pin 11</td>
<td>Pin 6</td>
</tr>
<tr>
<td>LP6</td>
<td>LP8 Extension Board</td>
<td>Pin 13</td>
<td>Pin 7</td>
</tr>
<tr>
<td>LP7</td>
<td>LP8 Extension Board</td>
<td>Pin 15</td>
<td>Pin 8</td>
</tr>
<tr>
<td>MO / Master Oscillator</td>
<td>LP8 Extension Board</td>
<td>Pin 8</td>
<td>Pin 18</td>
</tr>
<tr>
<td>LaserA / Frequency</td>
<td>LP8 Extension Board</td>
<td>Pin 22</td>
<td>Pin 20</td>
</tr>
<tr>
<td>Laser Gate / Modulation</td>
<td>LP8 Extension Board</td>
<td>Pin 26</td>
<td>Pin 19</td>
</tr>
<tr>
<td>Alarm, one of DIn0…DIn7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 4, 6, 8, 10, 12, 14, 16 or 18</td>
<td>Pin 16</td>
</tr>
<tr>
<td>Alarm, one of DIn0…DIn7</td>
<td>Digi I/O Extension Board</td>
<td>Pin 4, 6, 8, 10, 12, 14, 16 or 18</td>
<td>Pin 21</td>
</tr>
</tbody>
</table>

In this wiring-scheme no GND-connections are listed, they have to be added in order to get valid and working connections.
APPENDIX G – IDC connector pin numbering

Pin numbering of the IDC connectors (according to pinout-tables shown in hardware description sections above) can be seen in below image:

The first pin is marked by a small arrow in connector. Second pin is below of it, counting continues column-wise.
APPENDIX H – E1701D XY2-100 protocol description

The data submitted at 26 pin connector of E1701D are conform to XY2-100 specification:

- **CLK+**
- **SYNC+**
- **X+, Y+, Z+ Data**

In standard 16 bit operating mode first three bits are set to 001, then 16 bit position data followed by a parity bit (even parity) are transmitted:

| 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0  | 0  | 1  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | D15..D0 position data | Pe |

In enhanced XY2-100 18 bit operating mode first bit is set to 1, then 18 bit position data followed by a parity bit (odd parity) are transmitted:

| 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | D17..D0 position data | Po |

To use this mode, the related tune-value has to be set in configuration file (please refer to section “6.1.8 microSD-Card”), this mode requires firmware version 24 or newer.
APPENDIX I – Board dimensions

Board dimension drawings (baseboard plus optional extension boards), all values are given in unit mm.

<table>
<thead>
<tr>
<th>Board type</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1701A Baseboard</td>
<td>40 mm</td>
<td>7,3 mm</td>
</tr>
<tr>
<td>E1701D Baseboard</td>
<td>40 mm</td>
<td>7,3 mm</td>
</tr>
<tr>
<td>LP8 Extension Board</td>
<td>40 mm</td>
<td>7,3 mm</td>
</tr>
<tr>
<td>Digi I/O Extension Board</td>
<td>34 mm</td>
<td>10,3 mm</td>
</tr>
<tr>
<td>Secondary Head Extension Board</td>
<td>40 mm</td>
<td>7,3 mm</td>
</tr>
</tbody>
</table>

X – for future compatibility leave additional space of 10 mm at Ethernet connector side of the controller
Index

1
1701_CSTATE_MARKING - 68
18 bit - 20, 94

2
2D marking on-the-fly - 38
2D on-the-fly - 20

A
Alive - 16
analogue output - 11
ASCII - 42
autofile - 19, 26

B
BeamConstruct PRO license - 12, 15
boot - 22
Boot - 16

C
cdser - 46
cecho - 42
cfim - 46
cgbds - 43
cgbser - 43
cgc - 44
cgepr - 44
cginp - 42
cglog - 43
cgser - 45
cgsta - 47
cgtim - 46
cgtin - 44
cgtxt - 45
chalt - 43
ciser - 46
cjsor - 18, 44
clepr - 44, 76
CLK - 23
cmsor - 18, 45
CO2 - 10f., 33f.
corrtable - 18, 44
corrtable0 - 17
cpwor - 18, 45
crlbf - 28, 44
crrrr - 47
crser - 46
crtim - 46
csc - 18, 44
csr - 42
csser - 45
csst - 20, 47
cstat - 43
cstdy - 47
csthr - 47
cstim - 46
cstmi - 47
cstmo - 46
cstop - 43
cstrt - 43
cstsc - 47
cstxt - 45
csty - 46
ctrig - 43
cvers - 42
CW - 10f., 23, 34

cstmo - 46

cstop - 43
cstrt - 43
cstsc - 47
cstxt - 45
csty - 46
ctrig - 43
cvers - 42

D
D-SUB15 - 23
D-SUB25 - 23
DataMatrix - 79
digidebc - 19
digiinit - 19
digimask - 19
digital in- and outputs - 35
dimension drawing - 95
dimensions - 95
Download new firmware - 53
DYN_DATA_MAX_STRING_LENGTH - 78f.

E
E1701_close() - 56, 76
E1701_COMMAND_FLAG_DIRECT - 61, 71f., 75
E1701_COMMAND_FLAG_MOTF_WAIT_BITS - 74
E1701_COMMAND_FLAG_MOTF_WAIT_INCS - 74
E1701_COMMAND_FLAG.Scanner_VAR_POLYDELAY - 59
E1701_COMMAND_FLAG.Scanner_STREAM - 61, 71f., 75
E1701_COMMAND_FLAG.Scanner_WRITE_LASERA_GPO - 75
E1701_COMMAND_FLAG.Scanner_WRITE_LASERB_GPO - 75
E1701_COMMAND_FLAG.Scanner_WRITE_LASER_GATE - 75
E1701_COMMAND_FLAG.Scanner_WRITE_LP8LATCH - 75
E1701_COMMAND_FLAG.Scanner_WRITE_LP8MO - 75
E1701_COMMAND_FLAG.Scanner_XYCORR_MIRRORX - 57
E1701_COMMAND_FLAG.Scanner_XYCORR_MIRRORY - 57
E1701_CSTATE_FILE_WRITE_ERROR - 68
E1701_CSTATE_HALTED - 68
E1701_CSTATE_MARKING - 68
E1701_CSTATE_MARKING|E1701_CSTATE_PROCESSING - 68
E1701_CSTATE_PROCESSING - 68
E1701_CSTATE_WAIT_EXTTRIGGER - 68
E1701_CSTATE_WAIT_INPUT - 68
E1701_CSTATE_WAS_START_PRESSED - 68
E1701_CSTATE_WAS_STOP_PRESSED - 68
E1701_CSTATE_WRITING_DATA - 54
E1701_CSTATE_WRITING_DATA_ERROR - 54
E1701_delay() - 68, 74
E1701_digi_pulse() - 72
E1701_digi_read2() - 72
E1701_digi_set_mip_output() - 19, 74
E1701_digi_set_motf_sim() - 74
E1701_digi_set_motf_sim() - 73
E1701_digi_set_motf() - 73f.
E1701_digi_set_wet_output() - 19, 74
E1701_digi_wait_motf_sim() - 74
E1701_digi_wait() - 73
E1701_digi_write2() - 72
E1701_dynamic_data() - 76f.
E1701_ERROR - 80
E1701_ERROR_BORD_NA - 80
E1701_ERROR_FILENAME - 80
E1701_ERROR_FILEOPEN - 80
E1701_ERROR_FILEWRITE - 80
E1701_ERROR_INVALID_CARD - 80
E1701_ERROR_INVALID_DATA - 80
E1701_ERROR_NO_CONNECTION - 80
E1701_ERROR_NO_MEMORY - 80
E1701_ERROR_NOT_SUPPORTED - 80
E1701_ERROR_TRANSMISSION - 80
E1701_ERROR_UNKNOWN_FW - 80
E1701_execute() - 55, 66
E1701_FILEMODE_LOCAL - 76f.
E1701_FILEMODE_SEND - 76f.
E1701_FREE_SPACE_PRIMARY - 65, 70
E1701_FREE_SPACE_SECONDARY - 70
E1701_get_card_state2() - 67
E1701_get_free_space() - 55, 65, 70
E1701_get_library_version() - 70
E1701_get_startstop_state() - 67
E1701_get_sync() - 66
E1701_get_version() - 70
E1701_halt_execution() - 67f.
E1701_jump_abs() - 62
E1701_LASERMODE_CO2 - 60
E1701_LASERMODE_CRF - 61
E1701_LASERMODE_DUAL - 61, 69
E1701_LASERMODE_YAG1 - 60
E1701_LASERMODE_YAG2 - 60
E1701_LASERMODE_YAG3 - 60, 69
E1701_load_correction() - 55f., 76
E1701_lp8_a0_2() - 71
E1701_lp8_writeLatch() - 71
E1701_lp8_write_mo() - 71
E1701_lp8_write2() - 70
E1701_mark_abs() - 62
E1701_mark_pixelline() - 64f.
E1701_OK - 80
E1701_PIXELMODE_JUMP_N_SHOOT - 64
E1701_PIXELMODE_NO_JUMPS - 64
E1701_power_callback() - 64
E1701_release_trigger_point() - 65, 70
E1701_set_connection() - 55, 63, 75ff.
E1701_set_filepath() - 76
E1701_set_fpk() - 69
E1701_set_laser_delays() - 59, 64
E1701_set_laser_mode() - 60
E1701_set_laser_timing() - 68
E1701_set_laser() - 61
E1701_set_laserb() - 61, 69
E1701_set_matrix2() - 57, 65
E1701_set_password() - 18, 56
E1701_set_pixelmode() - 63
E1701_set_pos() - 63
E1701_set_scanner_delays - 59
E1701_set_speeds() - 58
E1701_set_standby() - 69
E1701_set_sync() - 66
E1701_set_trigger_point() - 55, 65
E1701_set_wobble() - 61
E1701_set_xy_correction2() - 65
E1701_set_xy_correction3() - 57
E1701_set_z_correction() - 58
E1701_stop_execution() - 66
E1701_switch_correction() - 56
E1701_write() - 75
e1701.cfg - 56
E1701.cfg - 17
E1701.dat - 17
E1701.fwi - 17
E1701A - 10, 29
E1701D - 10, 13
E1701M - 10
E1701M_ERROR_UNKNOWN_BOARD - 80
electrically insulated - 35
electrostatic sensitive device - 9
enhanced XY2-100 - 20, 94
Error - 16
ESD - 9
Ethernet - 10f., 13, 17f., 22, 29
extension - 24, 31, 34, 40
ExtStart - 19, 65, 74
ExtStop - 19, 66

F
fastdebc - 19
fiber - 33
fiber laser - 89
fiber lasers - 34
Firmware - 22
FPK - 10f., 34

G
gateway - 21
grid correction - 10f.

H
haltedloop - 18, 26, 28
haltedlooptimeout - 18, 26f.

I
ILDA - 31
in-polygon delay - 59
Interlock - 31
iobuff - 19, 28, 44
iohaltedloop - 18, 27f.
ioselect - 19, 27f., 44
IP - 13, 17
ip1 - 17
IPG - 10f., 33
IPG YLP - 86f., 89
IPG YLR - 88

J
JPT YDFLP - 89
jump delay - 59, 62

L
Laser LED - 13, 16, 29f.
laser off delay - 62
laser on delay - 62
Laser Show - 85
Laser signals - 33
Laser Signals - 33
LaserA - 34
LaserB - 34
Latch - 34
LED - 16, 30
libslrtc4.so - 81
load_correction_file() - 56
loop - 26
LP8 - 34

M
mark delay - 59, 62
Marking Active - 16
marking on-the-fly - 11, 35f.
Marking On-The-Fly - 37
Master Oscillator - 11, 33f.
Micro-SD - 17
Micro-SD-card - 13, 29
Micro-SD-Card - 17, 30
mipout - 19
MO - 34
MO LED - 33
MOPA - 89

N
n_load_correction_file() - 56
netmask - 21
NOTE - 13
NTP - 21

O
OLSC - 85
Opto-Configuration - 35, 38
opto-insulated - 36

P
passwd - 18
pethd - 22
polydelay - 59
Power - 13, 29
Power LED - 13, 16f., 29f.
Power supply - 15, 30
pulse-width modulated frequency - 34
PWM - 34
PWM frequency - 10f.

Q
Q-Switch - 10f, 34
quadrature encoder - 11

R
Raycus - 92
Reset - 13, 17
RTC4 - 80
RTC4DLL.dll - 81
S
ScSCIInitInterface() - 56
SNTP - 21
SPI - 11
SPI G4 - 90
Stand-Alone - 26
Stand-Alone Control - 28
standalone - 18
Stop - 16
SYNC - 23

T
tune - 20
Type B - 86
Type D - 89
Type E - 87

U
USB - 13, 15, 29
USB 2.0 - 10f.
User LEDs - 13, 29

V
debut - 75

W
waveform - 90f.
wetout - 19

X
XY2/100 - 10, 13, 23

Y
YAG - 10f., 33f.

. .
.bco - 56
.ct5 - 56
.ctb - 56
.gcd - 56
.ucf - 56
.xml - 56