E1701C Modular 5-Axis CNC Controller

Users Manual

Table of Contents

1 Copyright

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

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

Portions of the E1701 firmware are based on StarterWare 2.0 (or newer):

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):

The E1701C baseboard firmware bases on motion5 version 1.1 or newer*:

2 History

Date	**Changes in document**
10/2022	Electrical behaviour of digital IOs clarified
06/2022	New commands `cslp8`, `cslgt` and `cslmo` added
02/2022	All references to unused pilot laser signal removed
02/2022	Tune-flag added to invert input logic of ExtStart input
02/2022	Tune-flag now also can read hex-values with 0x prefix
02/2022	Added tune-flags to invert LP8 and MO outputs
07/2021	Inappropriate language and naming removed (“master”, “slave”, ...)
07/2021	Wiring of JPT/MOPA laser with pulse width serial interface clarified
04/2021	Description of new configuration parameter “eth=2” for Ethernet interface polling added
04/2021	Description of Ethernet configuration updated for Windows 10
04/2001	Description of `holdbit` configuration parameter added
03/2021	Added function `E1701C_get_serial_number()`
11/2020	Baseboard pinout description corrected
09/2020	Function description of `E1701C_set_max_speed()` added
07/2020	Described referencing sequence in detail
03/2020	Clarified wiring of Step/Dir outputs
11/2019	Unsupported command `cgcmd` removed
10/2019	Funcition description of `E1701_set_standby2()` added
10/2019	Wiring scheme for MaxPhotonics fiber laser added
10/2019	Example in description of `pethd`-parameter corrected
03/2019	Added description of flag `E1701C_CSTATE_IS_REFERENCING`
02/2019	Added “`pethd`” configuration parameter
11/2018	Initial version

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 hardware 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 within this document is also designed to control motors and other electrically driven tools. Motions caused by these motors and tools 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.

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 a device for use "as is", but a component which is intended to be used as part of a larger device, e.g. for integration in a machine with own housing or within an electrical cabinet. 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 hardware, shielding/covering to not to let a person touch the hardware unwanted, proper packaging in ESD-bags, ...). For more information please refer to related regulations and standards regarding handling of ESD devices. The EMC Directive (2014/30/EU) does not apply to this hardware as it is not intended for an end user (a person without knowledge of EMC) and as it is not otherwise made available on the market.

The Low Voltage Directive (2014/35/EU) does not apply to this hardware as the voltage supply is below the 50V AC / 75V DC limit.

This document describes the E1701C-hardware but may contain errors or may be changed without further notice.

4 Overview

This document describes the E1701C modular CNC controller board family, their electrical characteristics and usage. They consist of E1701C 5-axis CNC controller baseboard plus optional extension board. Special variant E1701M is no CNC controller and therefore not covered by this document.

The E1701C CNC controller boards are designed for controlling motion-stage based stepper motor systems with two to five axes. Depending on the used extension boards (which are optional) they also supply extensive signals for laser or mechanical tools (like a mill) and external control. The communication between the host system and the controller boards is done via Ethernet or USB.

When using E1701C CNC 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.
An E1701C baseboard can be combined with several extension boards of different types but never with more than one board of same type.

4.1 Features

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

4.1.1 E1701C CNC Controller Baseboard

This baseboard can be used to control XY-tables, CNC-mills, XY/XYZ-stages or similar, stepper-motor driven devices. It can be combined with different extension boards without any restrictions as long as only one extension of same type is used at the same time. The E1701C baseboard offers following features:

- up to 5 stepper axes controllable via step and direction signals
- combined CNC motion of 2..5 axes
- can use lasers as well as many other mechanical tools for material processing/milling
- up to 38 kHz maximum step frequency (jitter-free)
- 100 Mbit Ethernet connection
- USB 2.0 connection
- command execution time down to 0,5 microseconds
- realtime processing of laser and motion signals
- can control nearly every laser type (this may require extension boards as described below)
- 512 MByte DDR3 RAM
- 1 GHz CPU main clock plus 3 parallel running, additional CPU cores/MCUs
- 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

4.1.2 E1701C 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
- Main 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 directly connected to 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

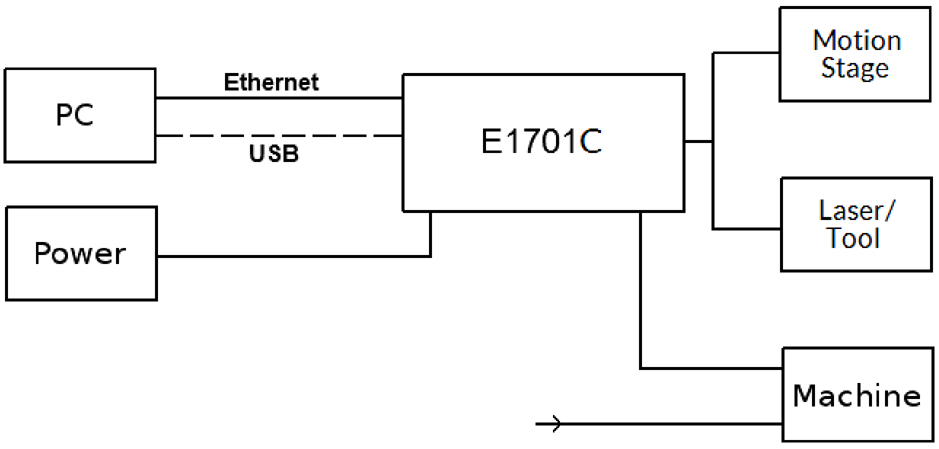
4.1.3 E1701C Digi I/O Extension Board

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

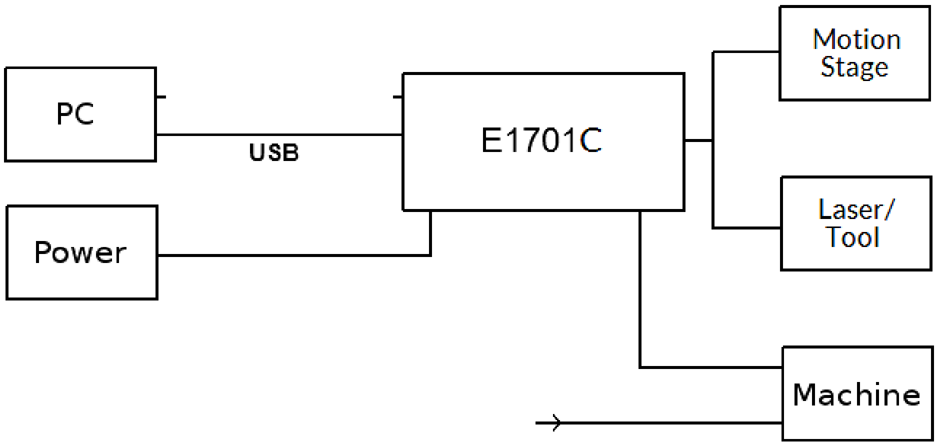
- 8 freely usable digital outputs providing either CMOS level or electrically insulated outputs via external power supply
- 8 freely usable digital inputs expecting either CMOS level or electrically insulated inputs via external power supply

5 Position Within The System

The E1701C CNC controller system can be connected to the host via Ethernet or USB to receive processing 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 possibilties (as described below). When using Ethernet connection, it optionally can be connected via USB too. In this case USB connection is only used to retrieve BeamConstruct PRO license from the board:



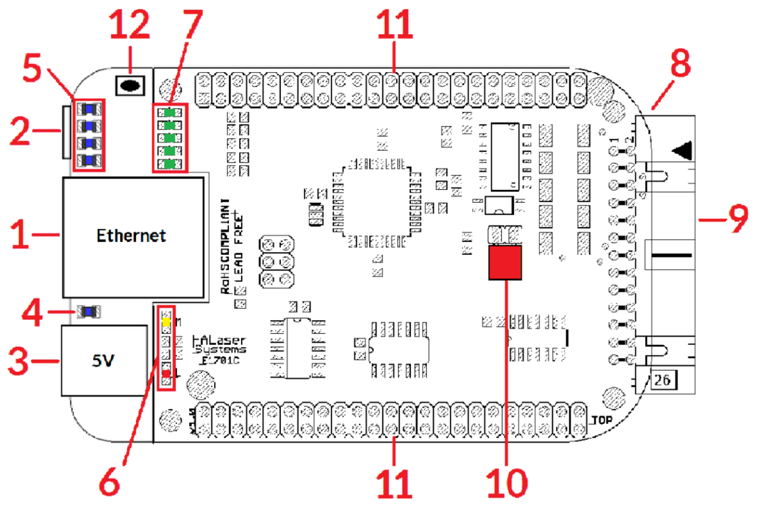
Since 100 Mbit Ethernet provides much faster data transfer than USB 2.0, this connection type is preferred. Especially in case complex processing data with many short lines that result in many separate jump and mark commands are used, Ethernet connection is more responsive.
When using USB connection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



In both cases the board itself is connected with the stepper motors via separate power drivers to submit fully synchronous 2D, 3D or 4..5 axis movement information to it. Beside of that it is connected to a laser or any other tool to submit motion-synchronous data. Additional communication channels between the E1701C CNC controller board and a connected machine can be done via separate IOs of an extension board.

6 Boards And Connectors

6.1 E1701C CNC Controller Baseboard



The E1701C 5-axis CNC Controller Baseboard provides following connectors and interfaces:

1. Ethernet – for communication with the host system, motion and processing information are submitted via this path
2. USB – via miniUSB connector for providing BeamConstruct PRO license to host system and optionally for submitting processing data from host to E1701C 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. State LEDs – show motion and tool/laser modulation states
7. Input state LEDs – 5 LEDs showing current state of limit inputs
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. Stepper motor and laser/tool signals – white 26 pin laser and motion signal output connector
10. Opto-Configuration - choose operation mode for limit-inputs
11. Extension connectors – extension boards can be placed here in order to add some more functionality and hardware interfaces to the board
12. Reset-button – on-board button to restart the board completely

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 motion 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 CNC controller card by using a separate Ethernet port. In case this is not possible, at least an own, physically separated sub-net for all controller cards should be set up. This network of course should be separated from normal network completely.
Ethernet connection is initialised once during start-up, thus Ethernet cable connecting E1701C board and host system needs to be plugged before the board is powered up.
By default the E1701C baseboard is using IP 192.168.2.254, thus the Ethernet network the card is connected to 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 E1701C network or to connect the CNC controller card with a standard network. Here it may be possible someone else in that network (accidentally) connects to that CNC controller and causes motion operations and/or laser emission.
The IP of the controller can be changed. This is necessary e.g. in case an other subnet has to be used or in case the E1701C board has to be operated in multi-controller 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 located on microSD-card. To change the IP please perform the following steps:

1. disconnect E1701C 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, KWrite or MousePad
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 E1701C board (place without the use of force, notice correct orientation with connectors of microSD-card to bottom!)
10. power up card

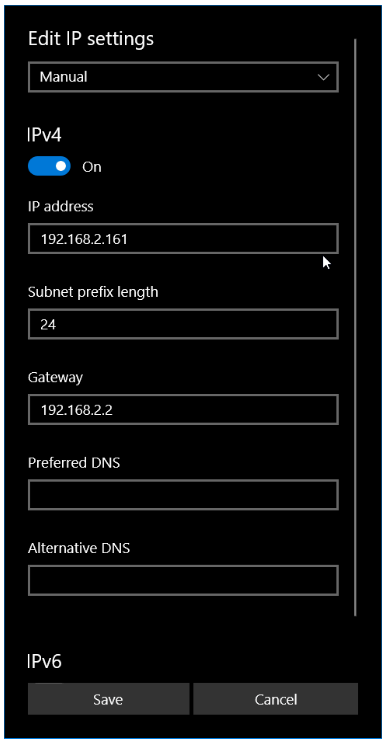
When User LEDs do not light up as described below, please check if microSD-card is placed in board correctly.

6.1.1.1 Ethernet Configuration With Windows 10

When E1701 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 10 (and similar) this configuration has to be done using following steps:

1. right-click the network-symbol in your taskbar
2. Select “Open network and internet settings”
3. Select “Ethernet” on the left
4. find the network interface E1803D has to be connected with and select it
5. Click the “Edit” button in section “IP settings”

6. now a window opens where "IPv4" has to be turned on and that has to be configured as follows:


Edit IP settings

Manual

IPv4

On

IP address: 192.168.2.161

Subnet prefix length: 24

Gateway: 192.168.2.2

Preferred DNS:

Alternative DNS:

IPv6

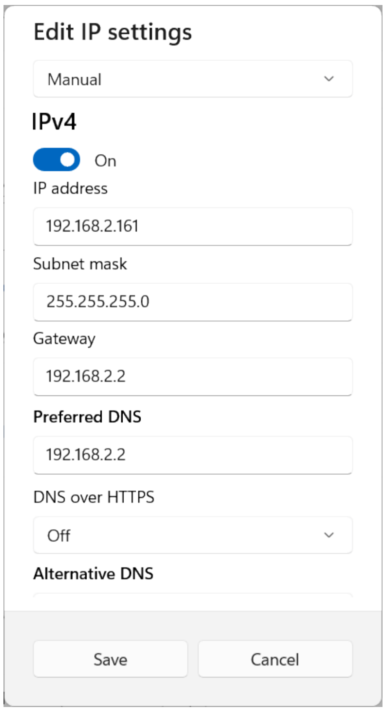
There you can specify an IP for your host PC. It has to belong to network 192.168.2.xxx and can be any number except than 192.168.2.254 (this is already the IP of the scanner card), 192.168.2.0 or 192.168.2.255.

6.1.1.2 Ethernet Configuration With Windows 11

When E1701 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 10 (and similar) this configuration has to be done using following steps:

1. right-click the network-symbol in your taskbar
2. Select "Network and internet settings"
3. Select "Ethernet" in the opened list
4. find the network interface E1701C has to be connected with and select it
5. Click the "Edit" button right beside "IP assignment"
6. now a window opens where "Edit IP Settings" has to be switched from "Automatic (DHCP)" to "Manual"

7. next “IPv4” has to be turned on and the remaining parameters in this window have to be configured as follows:


Edit IP settings

Manual

IPv4

On

IP address: 192.168.2.161

Subnet mask: 255.255.255.0

Gateway: 192.168.2.2

Preferred DNS: 192.168.2.2

DNS over HTTPS: Off

Alternative DNS

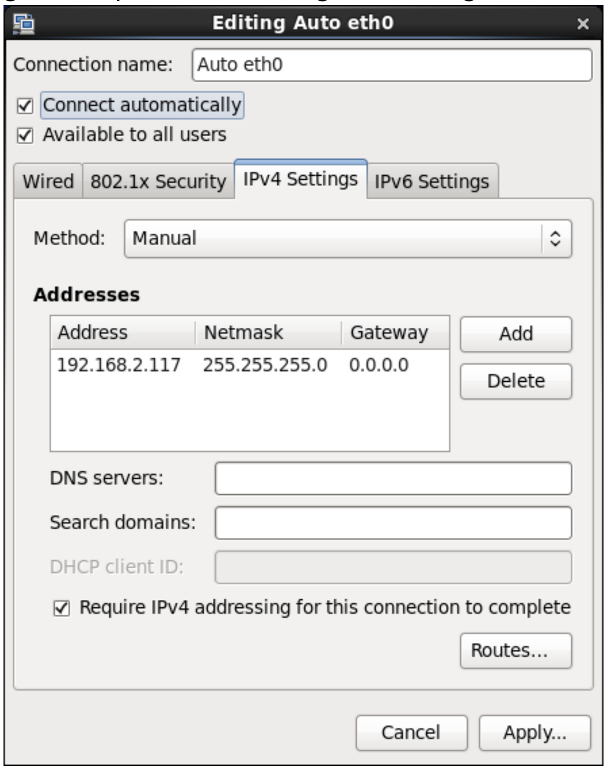
There you can specify an IP for your host PC. It has to belong to network 192.168.2.xxx and can be any number except than 192.168.2.254 (this is already the IP of the scanner card), 192.168.2.0 or 192.168.2.255.

6.1.1.3 Ethernet Configuration With Linux

When E1701C CNC 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 CNC controller card is connected with and press button "Edit"

4. go to tab-pane "IPv4 Settings" and configure it as shown below:


Editing Auto eth0

Connection name: Auto eth0

☑ Connect automatically

☑ Available to all users

Wired | 802.1x Security | IPv4 Settings | IPv6 Settings

Method: Manual

Addresses

Address	Netmask	Gateway
192.168.2.117	255.255.255.0	0.0.0.0

DNS servers:

Search domains:

DHCP client ID:

☑ Require IPv4 addressing for this connection to complete

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 CNC controller 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 processing 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 processing data!
The required device driver is installed automatically during the installation of the HALsetup software package (Windows) or comes with operating system by default (Linux). E1701C 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 E1701C controller correctly. Nevertheless it might be possible E1701C 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 E1701C CNC 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.

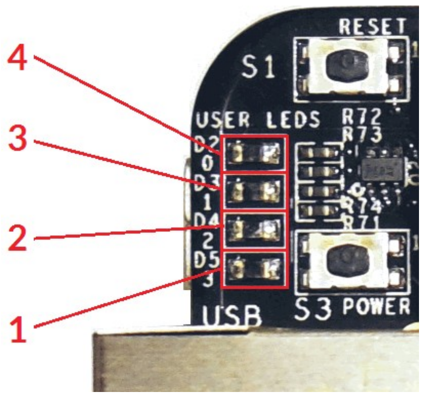
To avoid high frequency interferences from other electrical equipment or from within the power supply, it is recommended to place a ferrite bead at the cable close to the board. Please also check for correct shielding in respect to the equipment the E1701C card is used within.

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 "6.1.5 User LEDs" 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 operation the blink frequency may change. Only in case it does not blink for more than 10 seconds, the board has died for some reason and should be restarted.
2. Processing Active LED – this LED is turned on as long as an operation is running. This LED does not correspond to the tool on-off/laser gate signal, comparing to it it's also enabled during jumps or wait-cycles when laser is turned off but processing 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 E1701C 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.

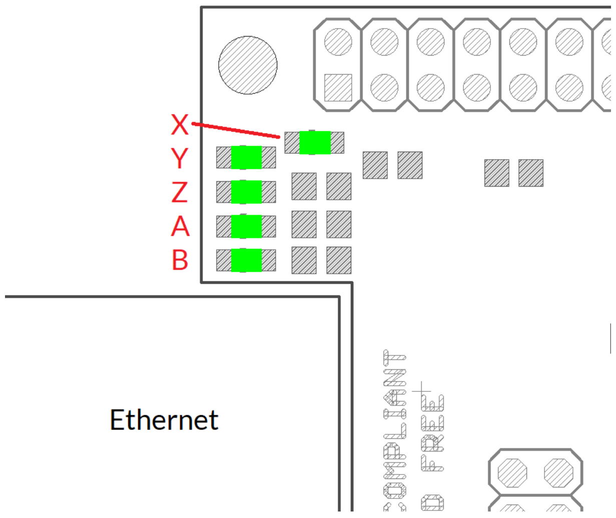
6.1.6 State LEDs

These LEDs show the current motion and marking/milling state of the controller. The inner, yellow LED is turned on as long as a motion is running.
The outer, red LED shows the modulation state of the laser/the activation state of the connected tool and signal of laser gate output. It is turned on as long as the laser/the tool is turned on and the laser gate output high. This

LED does NOT signal the same like the Processing Active LED described above since it will be turned off during jumps.

6.1.7 Input State LEDs

These 5 yellow LEDs show the state of corresponding 5 digital reference inputs. As long as a HIGH signal is detected on an input, the related LED is turned on. These LEDs can be used to check if a reference input "Ref" is at high:



For a description of these inputs, please refer to section "6.1.9 Stepper motor and control signals" below.

6.1.8 microSD-Card

The microSD card is storage place for firmware and configuration files. Here SD and SDHC cards with a maximum capacity of up to 32 GB are supported.
To remove the microSD-card, first disconnect all power from the E1701C board completely (including USB, the 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 E1701C board's card slot and press it gently until a click-noise signals locking of the card. Now the board can be powered.
E1701C 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 e1701.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:

Parameter	Description	Example
ip1	Configures IP of Ethernet port. Here only IPs in xxx.xxx.xxx.xxx notation are allowed but no host or domain names.	`ip1=192.168.2.100` specifies IP 192.168.2.100 to be used for Ethernet interface on next startup
passwd	Specifies an access password that is checked when card is controlled via Ethernet connection. This password corresponds to password specified with function `E1701C_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 <u>not</u> give the possibility to operate E1701C controller via insecure networks or Internet! It is transferred unencrypted and therefore can be "hacked" easily. Intention of this password is to avoid collisions between several E1701C 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 letters.	`passwd=myCardPwd` set a password "myCardPwd"
mipout	Configure a Digi I/O output pin to be used as “mark in progress”-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 an operation is in progress, the value given here can be overwritten by API-function `E1701C_digi_set_mip_output()`	`mipout=1` use DOut1 for mark-in-progress signal
wetout	Configure a Digi I/O output pin to be used as “wait for external trigger”-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 an 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 `E1701C_digi_set_wet_output()`	`wetout=0` use DOut0 for mark-in-progress signal
digiinit	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.	`digiinit=2` set DOut1 to HIGH initially and all other outputs to LOW
digimask	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 function requires firmware version 32 or newer.	`digimask=253` use only DIn2..DIn7 as input and ignore DIn0 and DIn1
digidebc	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.	`digidebc=10` set the debounce-time to 310 usec

Parameter	Description	Example

tune	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. Starting with firmware version 41 the number can also be specified as hexadecimal value when it is prefixed with “0x”. Following numbers can be concatenated by adding them:
	
	1 (0x01) – use DIn7 of Digi I/O Extension Board as external trigger, this disables ExtStart input of E1701C Baseboard and LP8 Extension Board
	
	8 (0x08) – 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
	
	16 (0x10) – invert LaserA output of LP8 extension to work as active HIGH signal
	
	32 (0x20) – invert LaserB output of LP8 extension to work as active HIGH signal
	
	32768 (0x8000) – invert the mark-in-progress signal of Digi I/O extension
	
	65536 (0x10000) – invert the wait-external-trigger signal of Digi I/O extension
	
	4194304 (0x400000) – invert the LP8 signal of LP8 Extension (requires firmware version 41 or newer)
	
	8388608 (0x800000) – invert the MO (main oscillator) signal of LP8 Extension (requires firmware version 41 or newer)
	
	16777216 (0x1000000) – inverts the logic of the ExtStart input. By default, the start-input reacts on a rising edge. When this flag is set, this is inverted and a falling edge is expected to release an external trigger. This also has an effect on the behaviour of tune-flag 0x2000000, it is inverted too.
	This flag requires firmware version 41 or newer.
		`tune=9`
		disables ExtStart input and switches over external trigger function to DIn7 input and inverts the logic of the LaserGate output
		
		`tune=0x10`
		logically invert the LaserA output signal

usb	When this parameter is set to 0, USB interface is disabled completely. This means it is no longer possible to connect to E1701C 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.	`usb=0` turn off USB interface

Parameter	Description	Example
eth	This parameter specifies the behaviour of the Ethernet interface. Here following values can be set: • 0 – Ethernet network interface is disabled completely. This means it is no longer possible to connect to E1701C 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 to save some power. • 1 – this is the default mode which enables the Ethernet interface and checks once at the beginning if some Ethernet hardware is connected to the controller card; when the "eth"-parameter is not specified at all, the resulting behaviour is the same • 2 – this enables Ethernet polling mode; instead of checking for an Ethernet device only once during boot, in this mode the interface is polled regularly until an electrical connection is detected. As long as the controller is polling, the Alive-LED blinks very slow and toggles once in about 20 seconds, when an Ethernet device was detected, the blink frequency changes to normal speed; PLEASE NOTE: when this mode is used, access via USB is limited, so "eth" should be set to "2" only when no communication via USB is intended. The "eth"-value of 2 requires a firmware version 39 or newer	`eth=0` – turn off Ethernet interface completely
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.	`pethd=20` - halt initialisation of the controller for about 10 seconds prior to initialisation of Ethernet interface
holdbit	This parameter specifies a bit (or bitpattern) at the digital inputs which can be used to halt the current operation and to hold all data. This means: • when the specified bitpattern is applied to the digital input, a running motion is stopped using the current acceleration ramps • as long as the bitpattern is applied, the operation is halted • when the bitpattern is removed (and no stop-command or signal was applied in between) the controller continues at the position where the hold-bitpattern was applied This function requires firmware version 39 or newer	`holdbit=128` Use DIn7 as input to hols the current operation

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://halaser.systems/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 Stepper motor and control signals

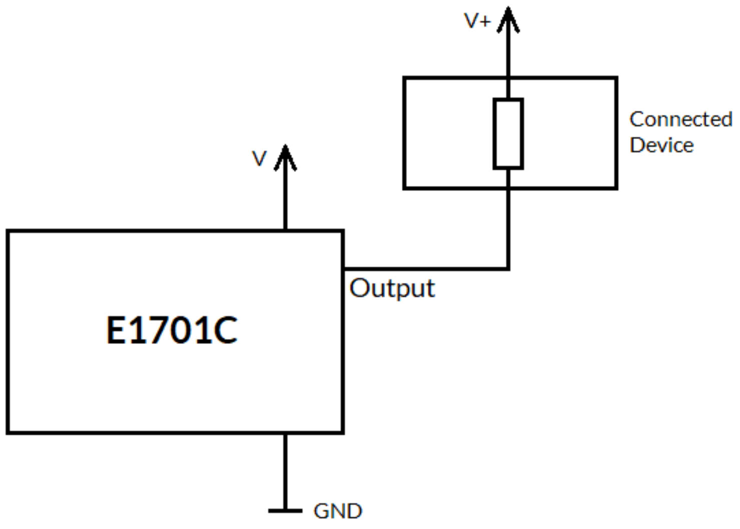
The white 26 pin connector provides several signals to control up to five stepper motor axes and connected tools which can be a laser or any other tool that is able to deal with the related signal. The connector is a white one to avoid confusion when a LP8 Extension Board is used too. This connector provides following signals:

Upper Row Of Pins	Signal	Voltage	Remarks	Lower Row Of Pins	Signal	Voltage	Remarks
1	Unused, do not connect!			**2**	5V	5V	
3	RefX	CMOS, 0/5V or 0/V_{ext}	Reference inputs	**4**	RefY	CMOS, 0/5V or 0/V_{ext}	Reference inputs
5	RefZ	CMOS, 0/5V or 0/V_{ext}		**6**	RefA	CMOS, 0/5V or 0/V_{ext}	
7	RefB	CMOS, 0/5V or 0/V_{ext}		8	Do not connect!		
9	GND_{ext}	GND	External ground	**10**	GND	GND	Board-internal Ground
11	Unused, do not connect!			12	Unused, do not connect!		
13	ExtStop	5V	Input control signal	**14**	ExtStart	5V	Input control signal
15	StepX	5V	Stepper pulse output signals	**16**	DirX	5V	Stepper motor direction output signals
17	StepY	5V		**18**	DirY	5V	
19	StepZ	5V		**20**	DirZ	5V	
21	StepA	5V		**22**	DirA	5V	
23	StepB	5V		**24**	DirB	5V	
25	LaserGate	5V		**26**	PowerRamp	5V PWM	

GND_{ext} depends on opto-configuration as described below. In opto-insulated mode (opto-configuration jumper not set) external ground has to be connected to this input. Then RefX..RefB work in respect to this external power add van be driven with an V_{ext} of up to 24 V. This is true for the reference inputs only, all other inputs remain with 0/5V logic levels and can't be driven with any external power.
WARNING: When no opto-insulated mode is selected (opto-configuration jumper is set), do NOT FEED ANY EXTERNAL POWER into Ref-inputs except the one from 5V output (pin 2), otherwise this would cause damage to the E1701C board!

The pins 15..24 provide the stepper motor control signals for axes 0..5 (step/direction signals to be used with a separate, external power driver).

Pins 15..26 all operate in open collector mode and have to be wired as follows:



Here V+ is either V (5V internal, non-insulated mode) or V_{ext} (up to 24V external, insulated mode). GND is either GND (non-insulated mode) or GND_{ext} (insulated mode). The internal resistor of the connected device is not allowed to have less than 530 Ohms (at 24V) or 110 Ohms (at 5V) in order to not exceed the given current limits as specified below.

The pins 3 to 7 are input pins for axes 0..5 to be used with the reference/homing position.

Laser Gate provides laser modulation signal, turns on the laser during marks and off during jumps.

PowerRamp is a signal which can be used for power-ramping applications: it provides a 9,75 kHz PWM signal which is proportional to the current nominal speed (according to the target maximum speed) and typically varies during acceleration and deceleration phases. It can be used either directly as PWM signal or with an additional capacity as analogue signal to modulate the power of a laser. For a variable PWM frequency the LaserA output of the LP8 extension board ("6.2 E1701C LP8 Extension Board") has to be used.

Maximum current to be pulled out of each of the outputs is 45 mA when wired in open collector mode and when operated in insulated mode with external power supply as shown above.
When working with internal power (non-insulated mode), the maximum allowed current is at 15 mA.

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 Referencing sequence

As the E1701C CNC controller makes use of external stepper motors which can't persist and provide the current position, prior to first use (after power-up) or when the motion position was changed manually and without the controller involved, all axes should be referenced in order to find a defined starting point. For this a referencing sequence has to be started either via the related API function call (please refer to section "9.1 E1701C Easy Interface Functions") or via suitable software. When started, a referencing sequence consists of the following steps:

1. move to the limit switch until it is hit (can be signalled either by LOW or HIGH input level, dependent on current configuration) using the first referencing speed
2. leave the limit switch until the leave distance has elapsed and using a lower speed; when the switch can't be left any more within a reasonable time, referencing fails and is cancelled at this point
3. move again to the limit switch until it is hit (can be signalled either by LOW or HIGH input level, dependent on current configuration) using the second referencing speed
4. leave the limit switch until the leave distance has elapsed and using a lower speed; when the switch can't be left any more within a reasonable time, referencing fails

When the referencing cycle has completed successfully, the controller sets the related axis position to value -1 (which can be changed to any other, suitable value by the controlling software). Now all movement operations can be done in relation to this fixed, defined position.
When the referencing cycle could not be completed successfully, the axis positions are undefined and should not be used for any motion operations!

6.1.10 Opto-Configuration

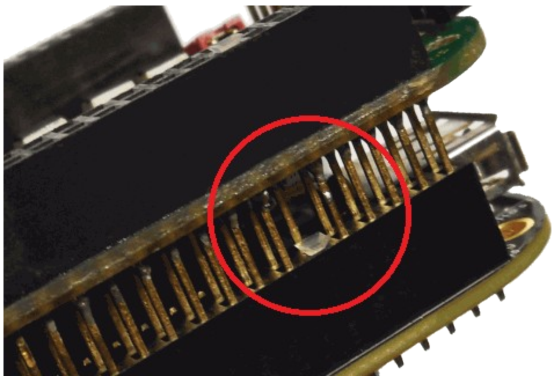
Using this jumper the operation mode for reference inputs RefX..RefB can be chosen. When is is set, the opto-couplers are powered internally. In this mode it is not working in opto-insulated mode and I/Os are using CMOS level signals.

When it is not set, external ground has to be provided at GND_{ext} pin of the 26 pin connector (as described above) and the reference inputs are working in electrically insulated, opto-coupled mode with input signal levels in range 5V..24V.

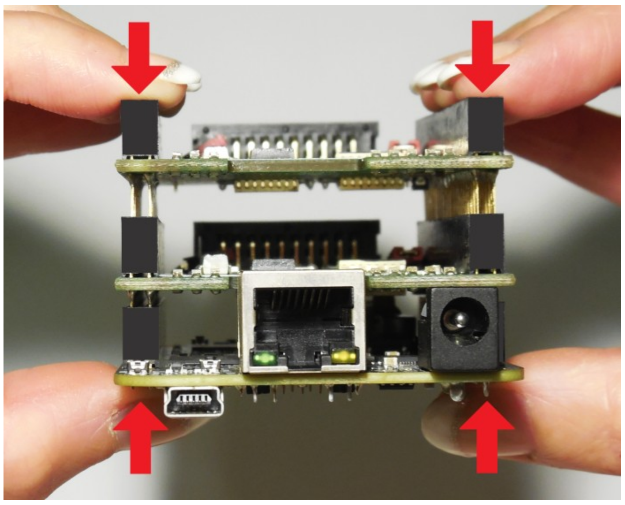
This opto-insulated mode applies ONLY to the reference inputs, all other signals including step/direction signals to stepper motor driver are not separated and need to be operated with an external galvanic separation when this is required.

6.1.11 Extension Connectors

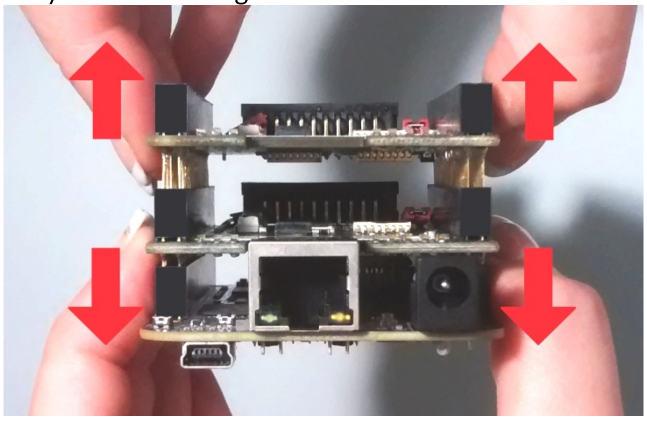
The two extension connectors on each side of the board can be used to place extension boards with additional peripheral interfaces. The extension connectors are designed to place/remove boards from time to time but they are not intended for constant hardware changes. So changing extension boards repeatedly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


Key pin closed on lower connector and missing in upper board to ensure correct orientation

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:



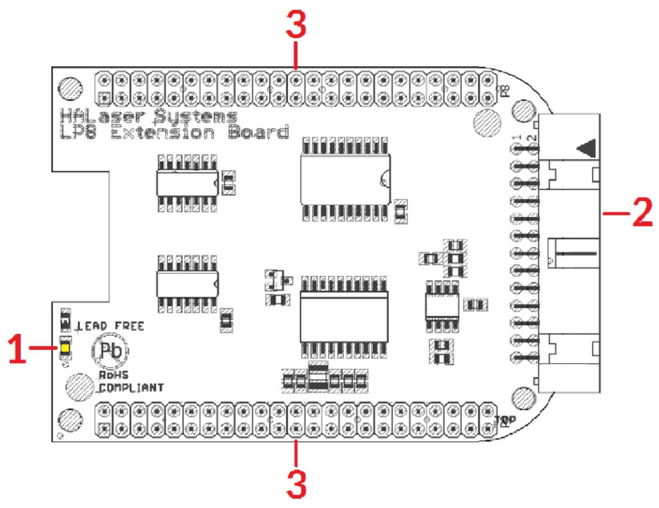
Due to of the large number of pins, it is easy to plug in an extension but more difficult to pull it out. So when removing an extension board, it is recommended to be very slow and to carefully pull each side up just a little bit to avoid bending of the pins as they exit.

6.1.12 Reset-Button

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

6.2 E1701C LP8 Extension Board

The E1701C LP8 Extension Board provides following features:



1. MO LED – shows state of Main 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

6.2.1 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.

6.2.2 Laser Signals

The black 26 pin connector provides several signals for controlling a laser source. It can be used e.g. together with YAG, CO_2, 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 E1701C Base board this connector is black.

This connector provides the following signals:

Upper Row Of Pins	Signal	Voltage	Remarks	Lower Row Of Pins	Signal	Voltage	Remarks
1	LP8_0	CMOS, 0/5V, max 8 mA		**2**	GND	GND	
3	LP8_1	CMOS, 0/5V, max 8 mA		4			
5	LP8_2	CMOS, 0/5V, max 8 mA		**6**	5V	5V	
7	LP8_3	CMOS, 0/5V, max 8 mA		**8**	MO	CMOS, 0/5V, max 8 mA	Master Oscillator
9	LP8_4	CMOS, 0/5V, max 8 mA		**10**	AOut0	0..5V, max 15 mA	Analogue output
11	LP8_5	CMOS, 0/5V, max 8 mA		12			
13	LP8_6	CMOS, 0/5V, max 8 mA		14			
15	LP8_7	CMOS, 0/5V, max 8 mA		16			
17	LP8 Latch	CMOS, 0/5V, max 8 mA		**18**	5V	5V	
19	LaserB	CMOS, 0/5V, max 14 mA	FPK	**20**			Connected to pin 21
21			Connected to pin 20	**22**	LaserA	CMOS, 0/5V, max 14 mA	PWM, frequency or Q-Switch
23	GND	GND		24			
25	5V	5V		**26**	Laser Gate[1]	CMOS, 0/5V, max 14 mA	

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 AOut0 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 CO_2 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).

AOut0 pin provides unipolar analogue output for controlling e.g. laser power or additional equipment. This output depends on LP8_0..LP8_7 outputs, 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, AOut0 is on 0V. When LP8 outputs are all HIGH, AOut0 is 5V.
PLEASE NOTE: output of 5V at AOut0 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 AOut0 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.

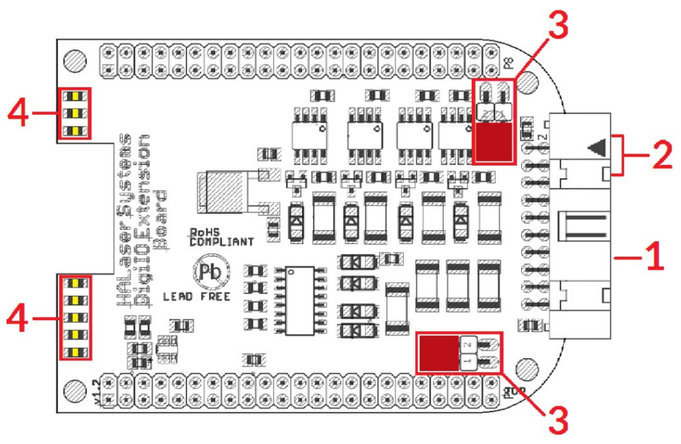
6.2.3 Extension Connectors

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.

1 requires hardware-revision 1.1 or newer

6.3 E1701C Digi I/O Extension Board

The E1701C Digi I/O Extension Board 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 E1701C, Digi I/O extension always has to be placed on top.

6.3.1 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:

Upper Row Of Pins	Signal	Voltage	Remarks	Lower Row Of Pins	Signal	Voltage	Remarks
1	V_{ext}	5..24V	Input voltage to be used in opto-insulated mode only	2	GND_{ext}	GND	External ground
3	DOut0	CMOS, 0/5V or 0/V_{ext}	Default level: LOW [1)]	4	DIn0	CMOS, 0/5V or 0/V_{ext}	Encoder-input A1 for marking on-the-fly
5	DOut1	CMOS, 0/5V or 0/V_{ext}	Default level: LOW [1)]	6	DIn1	CMOS, 0/5V or 0/V_{ext}	Encoder-input B1 for marking on-the-fly
7	DOut2	CMOS, 0/5V or 0/V_{ext}	Default level: LOW [1)]	8	DIn2	CMOS, 0/5V or 0/V_{ext}	Second encoder-input A2 for marking on-the-fly
9	DOut3	CMOS, 0/5V or 0/V_{ext}	Default level: LOW [1)]	10	DIn3	CMOS, 0/5V or 0/V_{ext}	Second encoder-input B2 for marking on-the-fly
11	DOut4	CMOS, 0/5V or 0/V_{ext}	Default level: HIGH [1)]	12	DIn4	CMOS, 0/5V or 0/V_{ext}	
13	DOut5	CMOS, 0/5V or 0/V_{ext}	Default level: HIGH [1)]	14	DIn5	CMOS, 0/5V or 0/V_{ext}	
15	DOut6	CMOS, 0/5V or 0/V_{ext}	Default level: HIGH [1)]	16	DIn6	CMOS, 0/5V or 0/V_{ext}	
17	DOut7	CMOS, 0/5V or 0/V_{ext}	Default level: HIGH [1)]	18	DIn7	CMOS, 0/5V or 0/V_{ext}	
19	V	5V	Board voltage, to be used only when not operating in insulated mode	20	GND	GND	Board-internal ground

[1)] Please note the wiring scheme and the resulting, inverted logic below: a level of LOW means, the output is pulled to GND and a load that is connected from V to this pin is turned on. An level of HIGH means, the output is pulled to V and a properly wired load if turned off.

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 E1701C board! In this case V_{ext} is equal to V (5V) of the board and GND_{ext} is connected to boards ground GND.

6.3.2 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.

6.4 E1701dock Extension Board

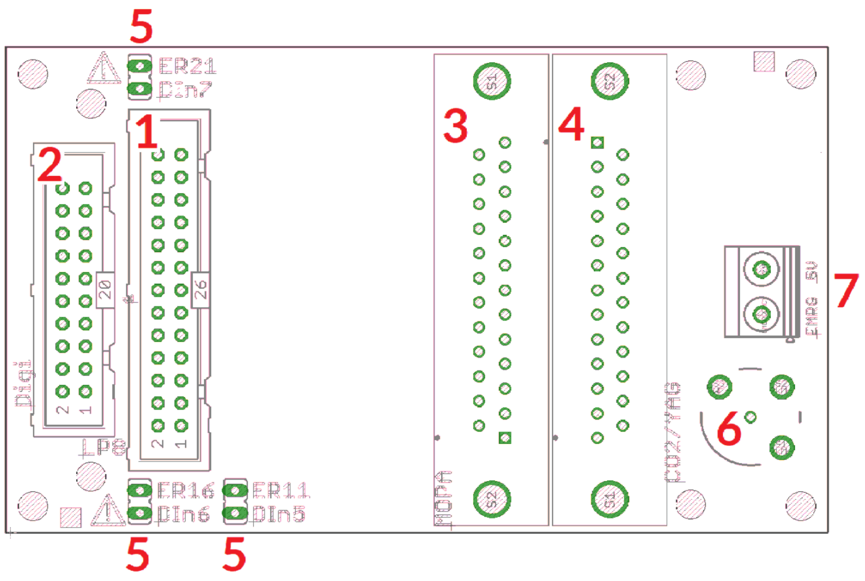
The E1701 dock is an expansion board which either can be mounted on very top of the E1701C (using hex-stands or distance bolts) or which can be mounted near to it (e.g. by using a second E1701base). It has to be wired with simple 1:1 flat-belt cables to the extension board(s) of the E1701C controller card. Dependent on which extension boards are available, the E1701dock provides some easy to use interfaces for connecting external equipment without the need to have own, complex wiring. Or in other words: when E1701dock is used, it acts as some kind of breakout-board, lasers can be connected quite easy, only a few 1:1 cables need to be plugged.

PLEASE NOTE: prior to using the E1701dock, please ensure the pinout of the connectors (as described below) really fit to your specific variant of your used laser. Elsewhere irreversible damage are possible at laser and/or scanner controller card.

PLEASE NOT: when using the laser and/or digital signals (from LP8 or Digi IO extension board) for some other tasks in parallel, you need to ensure they are not used twice, out of both connectors. This may otherwise result in an undefined and unwanted behaviour as they work parallel and can't be switched separately. When a parallel usage is intended, please double-check the current consumption of connected devices as the total current that can be provided by these lines does not double but is the same total value for two connections then. Exceeding the total maximum allowed current of a E1701C-signal may cause an irreversible damage to the controller card otherwise.

6.4.1 E1701dock Connectors

On top side the E1701dock provides the following connectors:



1. **E1701C LP8 connector**
   This connector has to be wired with a 1:1 connection to the LP8 extension boards interface (as described in section "6.2.2 Laser Signals"); in case additional signals of the LP8 extension need to be used elsewhere, a flat-belt cable can be used that comes with three IDC-plugs on it. Then one plug can be connected to the LP8 extension board, one to the E1701dock and the third one to some additional hardware making use of this interfaces signals. In this case please ensure no signals are double-used and no short-circuits are caused on input lines, otherwise the controller card, the E1701dock and/or the connected hardware may be damaged irreversibly!

2. **E1701C Digi I/O connector**
   This connector has to be wired with a 1:1 connection to the Digi IO extension boards interface (as described in section "6.3.1 Digi I/O"); in case additional signals of the Digi I/O extension need to be used elsewhere, a flat-belt cable can be used that comes with three IDC-plugs on it. Then one plug can be

connected to the Digi IO extension board, one to the E1701dock and one to some additional hardware making use of this interface. In this case please ensure no signals are double-used and no short-circuits are caused on input lines, otherwise the controller card, the E1701dock and/or the connected hardware may be damaged irreversibly!

The Digi I/O connector needs to be wired only in case
- a MOPA laser is connected and at least one of the alarm feedback signals is used or
- A YAG/CO_2 laser is connected and the shutter control signal is used

PLEASE NOTE: depending on the connected laser and the jumpers (8), some of the digital inputs are used from within the E1701dock! In this case the related signals should not be used for anything else, otherwise the controller card, the E1701dock and/or the connected hardware may be damaged irreversibly by a short-circuit! For more details please refer to the pinout of the laser connectors (5 and 6) and the description of the jumpers (8) below.

3. **MOPA laser connector**
This connector of the E1701dock can be used with most MOPA/fiber lasers such as IPG YLP interface types E (without APD indexing), D, D1, B, B1, MaxPhotonics MFP, JPT YDFLP, Raycus RFL and compatible laser types. Connection between E1701dock and laser is done mainly via a 1:1 D-SUB25 cable. The connector provides the following pinout:

Pin	Description	Pin	Description
1	LP8_0 laser power signal, CMOS, max 8mA	14	GND
2	LP8_1 laser power signal, CMOS, max 8mA	15	Not connected
3	LP8_2 laser power signal, CMOS, max 8mA	16	DIn6 laser alarm pin 16, available at DIn6 of Digi I/O extension only when jumper „DIn6/ER16“ is set (refer to description below)
4	LP8_3 laser power signal, CMOS, max 8mA	17	Not connected
5	LP8_4 laser power signal, CMOS, max 8mA	18	MO main oscillator signal, CMOS, max 8mA
6	LP8_5 laser power signal, CMOS, max 8mA	19	LaserGate power amplifier signal, CMOS, max 14mA
7	LP8_6 laser power signal, CMOS, max 8mA	20	LaserA pulse repetitive signal, CMOS, max 14mA
8	LP8_7 laser power signal, CMOS, max 8mA	21	DIn7 laser alarm pin 21, available at DIn7 of Digi I/O extension only when jumper „DIn7/ER21“ is set (refer to description below)
9	LP8 Latch laser power latch, CMOS, max 8mA	22	Pilot laser, unused for E1701C
10	GND	23	Emergency input, connected to screw connector (7), has to be pulled to HIGH for different laser types to enable operation
11	DIn5 laser alarm pin 11, available at DIn5 of Digi I/O extension only when jumper „DIn5/ER11“ is set (refer to description below)	24	Not connected
12	Not connected	25	Not connected
13	Not connected		

The signals on this connector are available only when LP8 extension and optionally Digi I/O extension are used.

4. **YAG/CO_2 laser connector**
This is a connector for direct connection to laser. A connection can be established using a standard 1:1 D-SUB-cable. This connector provides the following pinout which is typically to most common YAG or CO_2 lasers:

Pin	Description	Pin	Description
1	Not connected	14	Not connected
2	Not connected	15	Not connected

3	GND	16	Not connected
4	Emergency input, connected to screw connector (7) and has to be pulled to HIGH for different laser types to enable operation	17	GND
5	GND	18	GND
6	GND	19	GND
7	Shutter control, this pin is connected to DOut7 of the Digi IO extension	20	LaserB first pulse killer signal (FPK/QKILL), CMOS, max 14mA
8	GND	21	AOut1 power control, analogue signal in range 0..10V, max 15mA
9	AOut0 frequency control, analogue signal in range 0..5V, max 15mA; this signal is hardware-divided by 2 comparing to the original AOut0	22	LaserA TTL output, CMOS, max 14mA
10	GND	23	Pilot laser, unused for E1701C
11	GND	24	LaserGate non-inverted/high-active laser-on signal (LaserO+), max 14mA
12	LaserGate inverted/low-active laser-on signal (LaserO-), CMOS, max 40mA	25	LaserA negative output of differential PWM signal (PWM-, also connected to shield of BNC-connector/6)
13	LaserA positive output of differential PWM signal (PWM+, also connected to core of BNC-connector/6)		

The signals on this connector are available only when LP8 extension and optionally Digi I/O extension are used.

5. **MOPA alarm signal jumpers**
These jumpers can be set when a MOPA laser is connected via (3) and the alarm feedback signals of this laser have to be used. For each of the lasers alarm lines that is intended to be read back, the related jumper has to be set (for details please refer to the pinout of connector 3 above and the manual of your laser).
PLEASE NOTE: when a jumper is set here, the related digital input of the Digi IO extension can’t be used as input any more! Doing so can cause a short-circuit and may damage the scanner controller card, the E1701dock and/or the external equipment feeding a signal into DIn5/DIn6/DIn7.
Reading back the laser alarm signals requires the Digi IO extension

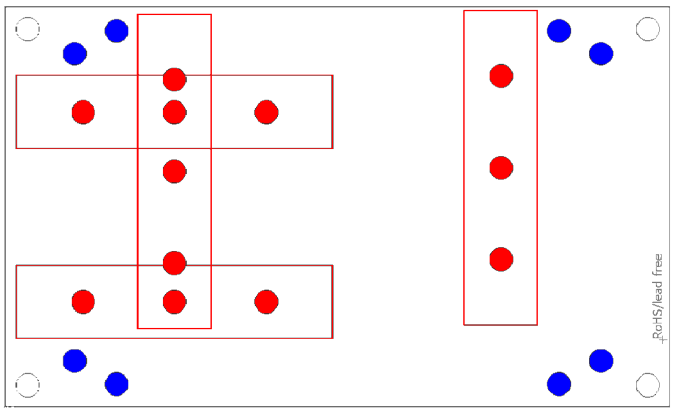
6. **BNC-connector for PWM/tickle-signal**
It provides a differential LaserA signal which is also available as PWM+/PWM- at the D-SUB25 connector (pins 13 and 25 of connector 4) and which can be used for all PWM-controlled lasers such as YAG or CO_2

7. **Enable signal connector**
This connector can be used to apply an enable signal to a connected laser. It has following pinout:

Pin	**Name**	**Description**
2	5V	5V output from scanner controller card, can be used e.g. to feed EMRG input
1	EMRG	Emergency/enable input for lasers, this pin is directly connected to pin 4 of YAG/CO_2 connector (4) and pin 23 of MOPA connector (3)

6.5 E1701base

The E1701base extension is a mounting help for easy installation on DIN rails/C45 rails and other possibilities of mechanical integration into machines:



RED – mounting positions for DIN/C45 rail locks/DIN/C45 rail adapters (bottom side). Pairs of locks can be mounted in one of 2 possible orientations. Here locks of type Phoenix Contact 1201578 or similar can be used. With these locks the board then can be clamped on a DIN/C45 rail.

BLUE – mounting holes for the E1701C CNC controller card on top of the E1701base in one of two possible orientations. These holes are symmetrically arranged so that the board can be mounted by 180 degrees rotated. Here Hex stands/distance bolts can be screwed in where the controller card is mounted on top.

Mounting procedure for E1701base:

1. Identify suitable positions (RED) for two DIN/C45 rail locks and mount them on bottom side (two or three screws from top side into the lock on bottom)
2. Mount hex-stands or distance bolts in at least four of the given mounting holes (BLUE).
3. Mount E1701A or E1701D on top of these hex-stands/distance bolts
4. Clamp the board on your DIN/C45 rail

Without the DIN/C45 rail clamps the board also can be used as top-cover for the E1701A or E1701D.

7 Quick Start into E1701C

Following a few steps are described that give users the possibility to quick start into usage of E1701C CNC 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://halaser.systems/download/manual_quickstart.pdf and to full user manual which is available at https://halaser.systems/download/manual.pdf.

To start with E1701C controller:

1. **SECURITY CHECK:** The following steps describe how to set up E1701C CNC controller card and how to control laser equipment and motors with it. 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://halaser.systems/download.php – for Windows this package contains all required drivers, for Linux no separate drivers are needed.
3. Connect E1701C controller via USB. Power supply via power jack is recommended.
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").
6. Start BeamConstruct laser marking software.
7. Go to menu "Project" → "Project Settings...", then tab-pane "Scanner".
8. Now you can select "E1701C" as controller card.
9. Press the "Configure"-button to get into the settings dialogue for E1701C 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 CNC 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

When E1701C CNC card is connected via USB and the USB-connection is NOT explicitly used for transmitting marking information, it can be used to send control commands to the card. Alternatively 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 CNC controller. This Telnet client should work in passive mode. So when E1701C CNC 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.
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`
"**vers**ion" – 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

`cgbds`
"**g**et **b**oar**ds**" – 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:
4 – CNC baseboard (E1701C) is available
256 – LP8 Extension Board is available
512 – Digi I/O Extension Board is available

`cglog`
"**g**et **log**line" – 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.

`cgbsr`
"**g**et **b**oard **ser**ial 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 controller card is used.

`crrrr`
"**r**eboot" – 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.

8.2 Hardware Commands

These commands can be used to access hardware signals directly. When these hardware outputs are set or unset while a marking operation is running, they may have no effect as they may be overridden immediately. Thus it is recommended to execute them only when the controller card is idle and no other operations are in progress. But also in this case, when a hardware output is set to a specific state, any operation (especially marking cycle) that is executed afterwards, may override that specific state-changes. Following hardware-specific commands are supported:

`cginp`
"**g**et **inp**ut" – 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

`cslgt <value>`
"**s**et **L**aser**G**a**t**e" – set the state of the LaserGate output either to HIGH (value is set to 1) or to LOW (value is set to 0).
This command requires firmware version 42 or newer.

`cslmo <value>`
"**s**et **MO**" – set the state of the main oscillator output either to HIGH (value is set to 1) or to LOW (value is set to 0).
This command requires firmware version 42 or newer.

`cslp8 <value>`
"**s**et **LP8**" – set the state of the LP8 output port to the value given as parameter. Here value is allowed to be in range 0..255, the related bits of the LP8 output are set according to the bitpattern of the specified number.
This command requires firmware version 41 or newer.

9 Programming Interfaces

The e1701c.dll / libe1701c.so shared library provides an own programming interface that gives the possibility to access and control the E1701C CNC controller card.

9.1 E1701C Easy Interface Functions

These functions belong to the native programming interface of E1701C CNC controller card and should be used preferential in order to get access to all features and full performance of the card. Functions of E1701C Easy Interface are either stream commands that are executed in the order they are called, or functions that are executed immediately.

The E1701C 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 `E1701C_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 `E1701C_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 `E1701C_get_free_space()`.

E1701C Easy Interface uses unit “mm” as base for all distance-units and -parameters.

E1701C Easy Interface provides following functions:

`unsigned char E1701C_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).
It returns a board instance number that has to be used with all following 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 `E1701C_open_connection()`.
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 E1701C_set_password(const unsigned char n,const char *ethPwd)`
Sets a password that is used for Ethernet connection of E1701C card. The same password has to be configured on E1701C 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 E1701C 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 E1701C_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 libe1701c write all function calls into a logfile so that it is possible to evaluate the real order of commands.
Parameters:

`n` – the 1-based board instance number as returned by `E1701C_set_connection()`
`path` – full path to the file which has to be used as debug log file
`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: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_write_debug_logfile(const unsigned char n,const char *format,…)`
This function correpsonds to `E1701C_set_debug_logfile()`, it gives the user the possibility to write own log information into a logfile.
Parameters:
`n` – the 1-based board instance number as returned by `E1701C_set_connection()`
`format` – format specifier as it is used e.g. by printf()
… - data according to the format specifier
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_open_connection(const unsigned char n)`
Opens the connection to the CNC controller card.
Parameters:
`n` – the 1-based board instance number as returned by `E1701C_set_connection()`
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`void E1701C_close(const 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 `E1701C_set_connection()` and `E1701C_open_connection()` is called again.
Parameters:
`n` – the 1-based board instance number as returned by `E1701C_set_connection()`

`int E1701C_set_xy_correction(const unsigned char n,const unsigned int flags,const double gainX, const double gainY, const double rot,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 and slant. With this command a matrix set with `E1701C_set_matrix()` 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:
`n` – the 1-based board instance number as returned by `E1701C_set_connection()`
`flags`:

- `E1701C_COMMAND_FLAG_XYCORR_MIRRORX` – the output will be mirrored in X-direction
- `E1701C_COMMAND_FLAG_XYCORR_MIRRORY` – the output will be mirrored in Y-direction

`gainX` – scale factor in x-direction, 1.0 means no scaling
`gainY` – scale factor in y-direction, 1.0 means no scaling
`rot` – rotation of whole working area in unit degrees
`offsetX` – offset in x-direction in unit bits, 0 means no offset
`offsetY` – offset in y-direction in unit bits, 0 means no offset
`slantX` – trapezoidal correction along X-axis in range -45..45°
`slantY` – trapezoidal correction along Y-axis in range -45..45°
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

int E1701C_set_matrix(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 -10000000 it is ignored and the previous/default value is kept at this position in matrix. With this command any correction set with `E1701C_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 `E1701C_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: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

int E1701C_set_speeds(const unsigned char n,double jumpspeed,double markspeed)
Set axis motion speed values to be used for all following vector data and until not replaced by other speed values. This command sets a combined speed value for all axes together. To specify a smaller maximum speed for a single axis, `E1701C_set_max_speed()` has to be called afterwards.
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 `E1701C_set_connection()`
`jumpspeed` – movement speed resulting out of movement of all axes during jumps (movements when laser/tool is off) in unit mm/sec
`markspeed` – movement speed resulting out of movement of all axes during mark (movements when laser/tool is on) in unit mm/msec
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

int E1701C_set_max_speed(const unsigned char n,const unsigned int axes,const double maxSpeed)
Specifies a maximum speed aone or more single axes are allowed to be moved with. Comparing to `E1701C_set_speeds()`, which sets the overall speeds to be used for operation, this command can be used to limit the speed for single axes. This typically is the case for the Z axis which in most cases has to be moved slower than X and Y. To restore the combined axis speeds and to reset the speed limit set with this function, `E1701C_set_speeds()` has to be called.
This function requires firmware version 38 or newer.

Parameters:
`n` – the 1-based board instance number as returned by `E1701C_set_connection()`
`axes` – a set of OR-concatenated `E1701C_COMMAND_FLAG_AXIS_x`-flags which specifies for which axis or axes the speed limit has to be set.
`maxSpeed` – the maximum speed in unit mm/sec the axis/axes (specified by parameter `axes`) have to be moved with
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

int E1701C_set_laser_delays(const unsigned char n,const double ondelay,const double offdelay)
Set laser delay values to be used for all following vector data and until not replaced by other delay values, please also refer to description of function `E1701C_mark_abs()` below.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other

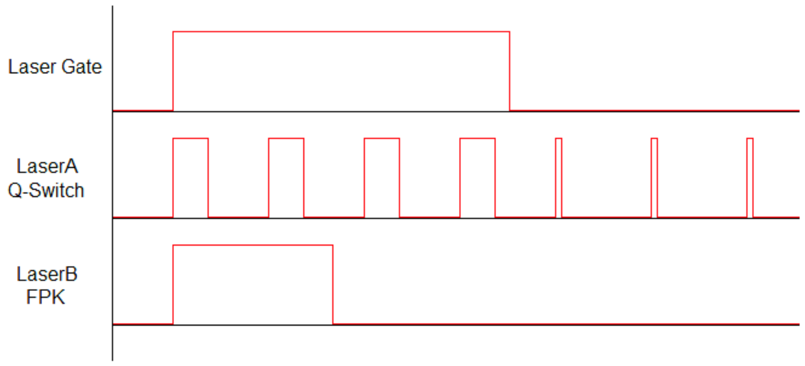
stream commands and changed values apply only to these vector data and coordinates, which are sent after calling this function.

Parameters:
`n` – the 1-based board instance number as returned by `E1701C_set_connection()`
`ondelay` – laser on delay in unit microseconds, currently only negative values are supported; the delay given here is used as pierce-time, means it specifies the time, the laser is turned on, before marking movement starts.
`offdelay` – currently not supported

Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

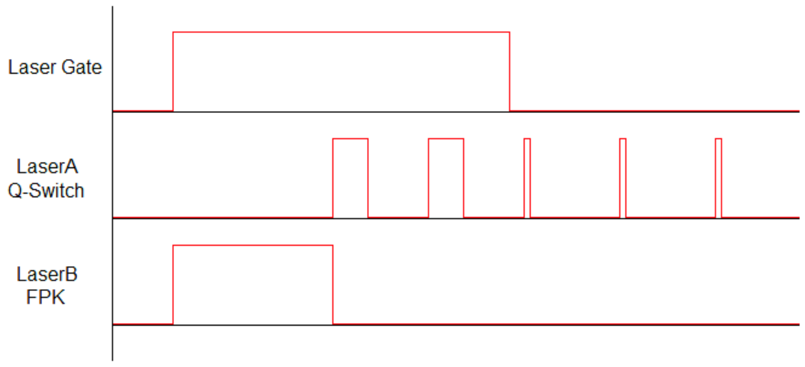
`int E1701C_set_laser_mode(const unsigned char n,const  unsigned int mode)`
Sets the laser mode to be used for all following operations, this value influences the signals emitted at the connectors of the LP8 extension card. This function has to be called prior to setting any other laser parameters (like 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 `E1701C_set_connection()`
`mode` - the laser mode, here one of the following values is possible:

- `E1701C_LASERMODE_CO2` – for controlling CO2 lasers, this mode supports stand-by frequency at LaserA output (to be set with function `E1701C_set_standby2()`) and PWM-modulated frequencies during marking and for power control (to be set with function `E1701C_set_laser_timing()`)
- `E1701C_LASERMODE_YAG1` – for controlling YAG lasers, this mode supports stand-by and Q-Switch frequency at LaserA output (to be set with function `E1701C_set_standby2()`) 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 `E1701C_set_fpk()`):



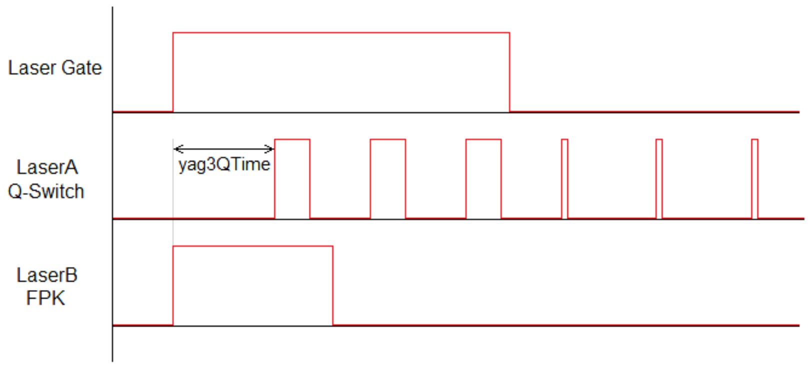
Here Q-Switch signal is started together with laser gate and FPK pulse. At end of mark when laser gate is turned off stand-by frequency is emitted at LaserA.

- `E1701C_LASERMODE_YAG2` - for controlling YAG lasers, this mode supports stand-by and Q-Switch frequency at LaserA output (to be set with function `E1701C_set_standby2()`) 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.

- `E1701C_LASERMODE_YAG3` – for controlling YAG lasers, this mode supports stand-by and Q-Switch frequency at LaserA output (to be set with function `E1701C_set_standby2()`) 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 `E1701C_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.

- `E1701C_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 `E1701C_set_standby2()`.
- `E1701C_LASERMODE_MOPA` – for fiber lasers which are driven by a main oscillator and power amplifier and that are power-controlled via LP8 digital port and latch bit

Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_set_laser(const unsigned char n,const unsigned int flags,const char on)`

Switches the laser on or off independent from any mark or jump commands.
Parameters:
`n` – the 1-based board instance number as returned by `E1701C_set_connection()`
`flags` – handling flags specifying the behaviour of this command, `E1701C_COMMAND_FLAG_STREAM` to use it as stream command, `E1701C_COMMAND_FLAG_DIRECT` to execute it immediately and independent on current stream and execution state; in case `E1701C_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: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_jump_abs(const unsigned char n,const int flags,const double x,const double y,const double z,const double a,const double b)`

Perform a jump (movement with laser/tool turned off) to the given position. This causes a motion from current position to the one specified by this functions parameters using the jump speed.
When laser/tool was turned on before this function is called, it is turned off at the beginning with a delay specified by laser off delay (please refer to description of `E1701C_mark_abs()` for a diagram showing laser off delay too).
This function does not guarantee a movement which describes a straight line from current coordinate position to the jump-target-coordinates. Instead of this always the fastest way to the new target position is used.
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 `E1701C_set_connection()`

`x` – the x-coordinate in unit mm the tool has to jump to
`y` – the y-coordinate in unit mm the tool has to jump to
`z` – the z-coordinate in unit mm the tool has to jump to
`a` – the a-coordinate in unit mm the tool has to jump to
`b` – the b-coordinate in unit mm the tool has to jump to
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_mark_abs(const unsigned char n,const int flags,const double x,const double y,const double z,const double a,const double b)`
Perform a mark (movement with laser/tool turned on) to the given position. This causes a movement from current position to the one specified by this functions parameters using the mark speed. Different to any call to `E1701C_jump_abs()` this function also guarantees a straight line movement is performed from current axis coordinate position to the one specified by this function call. 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 `E1701C_set_connection()`
`x` – the x-coordinate in unit mm the tool has to move to
`y` – the y-coordinate in unit mm the tool has to move to
`z` – the z-coordinate in unit mm the tool has to move to
a – the a-coordinate in unit mm the tool has to move to
`b` – the b-coordinate in unit mm the tool has to move to
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_set_trigger_point(const 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 `E1701C_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 `E1701C_set_connection()`
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_release_trigger_point(const unsigned char n)`
This function should be called only when a call to `E1701C_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 `E1701C_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 `E1701C_get_free_space()` with flag `E1701C_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 `E1701C_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 `E1701C_set_connection()`
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_execute(unsigned char n)`
Starts execution of all previously sent commands in case card is not already outputting these data. The E1701C 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 `E1701C_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 `E1701C_execute()`. But in situations where only very few data are sent to the card, it is necessary to call this function in order to ensure start marking also in these cases where the internal fill threshold is not reached and where the card would not start the operation automatically. 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 `E1701C_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 `E1701C_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 `E1701C_set_connection()`
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_stop_execution(unsigned char n)`
Stops the currently running execution as fast as possible and drops all marking data that still may be queued. A running motion operation is not cancelled but stopped with the defined deceleration rate so that no motion steps are lost on the connected motor and no referencing is necessary after such a stop.
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 `E1701C_set_connection()`
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_halt_execution(unsigned char n,unsigned char halt)`
Halts or continues the processing and output of marking data. On `halt=1` marking is tried to be stopped next time the laser/tool would be turned off. This stop is not guaranteed to take place immediately, so also when this functio nwas called, current process may still continue for some time. 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 `E1701C_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.
Parameters:
`n` – the 1-based board instance number as returned by `E1701C_set_connection()`
`halt` – 1 to halt operation next time the laser is off, 0 to continue a previously halted operation

Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`unsigned int E1701C_get_startstop_state(const 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 `E1701C_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 E1701C_get_card_state(const unsigned char n, unsigned int *state)`
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 any state which may be available for a very short time only in case of very small and fast marking cycles. So every state change on the controller (which itself always is 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" (`E1701C_CSTATE_MARKING|E1701C_CSTATE_PROCESSING`), next wait until this function signals "ready" (0).
Same for a referencing operation: first wait until this function signals busy with the `E1701C_CSTATE_IS_REFERENCING` flag set, next wait until this flag is cleared.
During transfer of vector data and motion/laser parameters this function should be called as rarely as possible: every call of `E1701C_get_card_state()` 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 `E1701C_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:

- `E1701C_CSTATE_MARKING` - card is currently marking
- `E1701C_CSTATE_PROCESSING` - card has received some data that are enqueued for marking
- `E1701C_CSTATE_ERROR` – a fatal operational error happened, such as a failed reference run
- `E1701C_CSTATE_IS_REFERENCING` – the controller is still in referencing mode which may include movement of axes that are signalled by flag `E1701C_CSTATE_MARKING` additionally.

When the function returns an error code instead of E1701C_OK, this value is undefined and can't be used.
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_delay(unsigned char n,const 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 `E1701C_set_connection()`
`delay` - time to wait until marking continues in unit usec, smallest possible value is 0,5 usecs
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_set_laser_timing(const unsigned char n,const double frequency,const double pulse)`
Set the frequency and pulse-width to be used during marking at LaserA output of LP8 Extension Board. This command also defines values for power-ramping. For laser types (like CO_2 or YAG) that perform power control via PWM, the pulse-value specifies the maximum power which has to be issued when axes are at their maximum speed. During ramping the real pulsewidth is smaller according to the current axis speed.
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 `E1701C_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: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_set_standby2(const unsigned char n,const double frequency,const double pulse,const bool force)`
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.
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 `E1701C_set_connection()`
`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`
`force` – when set to true, the new stand-by frequency is not applied the next time the laser is turned off, but immediately
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_set_fpk(const unsigned char n,const double fpk,const double yag3QTime)`
Set the parameters for first pulse killer signal that is emitted via LP8 Extension Board 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 `E1701C_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 `E1701_LASERMODE_YAG3` is set, elsewhere it is ignored
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_get_free_space(const unsigned char n,const int buffer)`
This function returns the space (in unit "commands") that is free in one of the buffers of E1701C. Here parameter `buffer` decides which buffer has to be checked.
Parameters:
`n` – the 1-based board instance number as returned by `E1701C_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:

- `E1701C_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 E1701C 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 `E1701C_get_card_state()` can be used.
  This buffer has also to be checked when function `E1701C_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 `E1701C_release_trigger_point()` above.

- `E1701C_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 E1701C_get_version(const 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 `E1701C_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

`int E1701C_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 E1701C_get_serial_number(const unsigned char n,char *serial,const int length)`

Reads the serial number of the used board and returns it as 7 bit ASCII data.
This is not a stream command, it is executed immediately and independent from all other commands.
This function requires a firmware version 39 or newer.

Parameters:
`n` – the 1-based board instance number as returned by `E1701_set_connection()`
`serial` – pointer to a char-array where the serial number has to be stored into, this memory area needs to have a size of at least 40 bytes
`length` – available length of the memory area where `serial` points to

Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_motion_set_steps(const unsigned char n, const unsigned int flags,const double steps)`

Set the factor which defines the relation between steps (increments) of the used stepper motor and the distance that it travels. This value needs to be specified prior to all other operations in order to allow correct calculation of all distances and speeds as expected by the other functions as described below. For the E1701C CNC controller no default value exists, so if no factor is set, motion operations are done with an undefined, random value which may lead to unexpected results.
The E1701C API always makes use of real distances (in unit mm) and does not expect the calling application to do the conversion from increments to mm.
Parameters:
`n` – the 1-based board instance number as returned by `E1701C_set_connection()`
`flags` – command flags specifying the type of function call (`E1701C_COMMAND_FLAG_STREAM` or `E1701C_COMMAND_FLAG_DIRECT`) and for which axes the given values have to be applied (`E1701C_COMMAND_FLAG_AXIS_0`, `E1701C_COMMAND_FLAG_AXIS_1`,

E1701C_COMMAND_FLAG_AXIS_2, E1701C_COMMAND_FLAG_AXIS_3,
E1701C_COMMAND_FLAG_AXIS_4)
steps – factor which defines relation between stepper motor steps and travel distance (in unit increments/mm)
Return: E1701C_OK or an E1701C_ERROR_-return code in case of an error

int E1701C_motion_set_limits(const unsigned char n,const unsigned int flags,const double llimit,const double hlimit)
Set motion limits for axis operations. When any follow-up command tries to set values beyond these limits, these values are clipped to the allowed range set with this function.

Parameters:
n – the 1-based board instance number as returned by E1701C_set_connection()
flags – command flags specifying the type of function call (E1701C_COMMAND_FLAG_STREAM or E1701C_COMMAND_FLAG_DIRECT) and for which axes the given values have to be applied (E1701C_COMMAND_FLAG_AXIS_0, E1701C_COMMAND_FLAG_AXIS_1, E1701C_COMMAND_FLAG_AXIS_2, E1701C_COMMAND_FLAG_AXIS_3, E1701C_COMMAND_FLAG_AXIS_4)
llimit – lower motion limit (in unit mm)
hlimit – upper motion limit (in unit mm)
Return: E1701C_OK or an E1701C_ERROR_-return code in case of an error

int E1701C_motion_set_accel(const unsigned char n,const unsigned int flags,const double accel)
Set the acceleration to be used for start and stop for all motion operations and for the specified axes.
Parameters:
n – the 1-based board instance number as returned by E1701C_set_connection()
flags – command flags specifying the type of function call (E1701C_COMMAND_FLAG_STREAM or E1701C_COMMAND_FLAG_DIRECT) and for which axes the given values have to be applied (E1701C_COMMAND_FLAG_AXIS_0, E1701C_COMMAND_FLAG_AXIS_1, E1701C_COMMAND_FLAG_AXIS_2, E1701C_COMMAND_FLAG_AXIS_3, E1701C_COMMAND_FLAG_AXIS_4)
accel – acceleration (in unit mm/sec^2)
Return: E1701C_OK or an E1701C_ERROR_-return code in case of an error

int E1701C_motion_reference(const unsigned char n,const unsigned int axis,const unsigned int mode,const double leaveDist,double speedStep0,double speedStep1)
Starts a referencing operation (=homing sequence) to have a defined position for the axis. The referencing sequence consists of following steps:

- move to reference switch (connected to reference-input) with first referencing speed speedStep0
- leave the reference switch by the given distance leaveDist – when the switch has not been released after the distance specified by leaveDist or after a reasonable time, referencing operation is set as "failed"
- move to reference switch (connected to reference-input) with second referencing speed speedStep1
- leave the reference switch by the given distance leaveDist – when the switch has not been released after the distance specified by leaveDist or after a reasonable time, referencing operation is set as "failed"
- set the position of the referenced axis to -1 – when referencing fails for some reason (because it was interrupted or because the limit switch could not be left within a reasonable time), the position of the axis will be undefined and E1701C_get_card_state() will return an error E1701C_CSTATE_ERROR.

This function returns only in case referencing was finished or cancelled due to an error. It can be interrupted by calling E1701C_stop_execution().
Parameters:

`n` – the 1-based board instance number as returned by `E1701C_set_connection()`
`axis` – specifies which axis has to be referenced, here a value in range 0..4 is expected
`mode` – specifies how referencing has to be done exactly, here a bunch of OR-concatenated flags can be handed over: one of the flags `E1701C_MOTION_REFSTEP_N` (to search for the reference input in negative direction) or `E1701C_MOTION_REFSTEP_P` (to search for the reference input in positive direction) which optionally can be combined with flag `E1701C_MOTION_REFSTEP_INV_SWITCH` to have inverted logic on the reference input
`leaveDist` – distance (in unit mm or degrees) to move off the reference switch after the switch was found for the first time
`speedStep0` – referencing speed (in unit mm/sec or degrees/sec) to find the reference switch for the first time (this value can be larger than `speedStep1` but should be small enough to not to overrun the switch)
`speedStep1` – referencing speed (in unit mm/sec or degrees/sec) to find the reference switch for the second time (this value should be smaller than `speedStep0` and is responsible for the accuracy of the referenced position)
Return: `E1701C_OK` when operation could be completed successfully, `E1701C_ERROR_REFERENCING` when referencing has failed for some reason or `E1701C_ERROR_`-return code in case of an other error

`int E1701C_motion_set_pos(const unsigned char n,const unsigned int flags,const double pos)`
This function does not cause any movement but resets the current axis position(s) to a new value. It can be used e.g. after successful referencing to set the initial positions to some own values. All following movement operations then are done in respect to the position values given here.
`n` – the 1-based board instance number as returned by `E1701C_set_connection()`
`flags` – command flags specifying the type of function call (`E1701C_COMMAND_FLAG_STREAM` or `E1701C_COMMAND_FLAG_DIRECT`) and for which axes the given values have to be applied (`E1701C_COMMAND_FLAG_AXIS_0`, `E1701C_COMMAND_FLAG_AXIS_1`, `E1701C_COMMAND_FLAG_AXIS_2`, `E1701C_COMMAND_FLAG_AXIS_3`, `E1701C_COMMAND_FLAG_AXIS_4`)
`pos` – the new position value to be set for the specified axis/axes (in unit mm or degrees)
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_lp8_write(const unsigned char n,const unsigned int flags,const 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).
Parameters:
`n` – the 1-based board instance number as returned by `E1701C_set_connection()`
`flags` – handling flags specifying the behaviour of this command, `E1701C_COMMAND_FLAG_STREAM` to use it as stream command, `E1701C_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: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_lp8_write_latch(const unsigned char n,const unsigned char on,const double delay1,const unsigned char value,const double delay2,const 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 `E1701C_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/setting the latch bit
`delay3` – delay to be issued after clearing/setting the latch bit for the second time
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_lp8_a0(const unsigned char n,const unsigned int flags,const 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).
Parameters:
`n` – the 1-based board instance number as returned by `E1701C_set_connection()`
`flags` – handling flags specifying the behaviour of this command, `E1701C_COMMAND_FLAG_STREAM` to use it as stream command, `E1701C_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: `E1701C_OK` or an `E1701C_ERROR_` return code in case of an error

`int E1701C_lp8_write_mo2(const unsigned char n, const unsigned flags, const unsigned char on)`
Sets the main oscillator output MO of LP8 Extension Board to be used with e.g. fiber lasers.
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).

Parameters:
`n` – the 1-based board instance number as returned by `E1701C_set_connection()`
`flags` – handling flags specifying the behaviour of this command, `E1701C_COMMAND_FLAG_STREAM` to use it as stream command, `E1701C_COMMAND_FLAG_DIRECT` to execute it immediately and independent on current stream and execution state
`on` – the state the MO output has to be switched to; PLEASE NOTE: the main oscillator depends on the current internal state of the laser. Thus turning it on is always possible but turning off the MO is possible only when the controller is not yet handling the laser-off delay, means it is not possible as long as the laser is turned on. In such a case this command is ignored.
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_digi_write(const unsigned char n,const unsigned int flags,unsigned int value,const 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).

Parameters:
`n` – the 1-based board instance number as returned by `E1701C_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: `E1701C_OK` or an `E1701C_ERROR_` return code in case of an error

`int E1701C_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 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 `E1701C_set_connection()`
`flags` – currently only `E1701C_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: `E1701C_OK` or an `E1701C_ERROR_` return code in case of an error

`int E1701C_digi_read(const unsigned char n,const unsigned int flags,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.
Parameters:
`n` - the 1-based board instance number as returned by `E1701C_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 E1701C_OK, this value is undefined and can't be used.
Return: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_digi_wait(const unsigned char n,const unsigned long value,const 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 `E1701C_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: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_digi_set_mip_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 "marking in progress". When `value` is set to 0xFFFFFFFF, this function is disabled and CNC 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 axes are 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 controller is waiting for an external trigger signal (ExtStart).
This is not a stream-commando, when it is called it is applied to current configuration immediately.
Parameters:
`n` – the 1-based board instance number as returned by `E1701C_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: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

`int E1701C_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 CNC 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.
Parameters:
`n` – the 1-based board instance number as returned by `E1701C_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: `E1701C_OK` or an `E1701C_ERROR_`-return code in case of an error

9.1.1 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 `E1701C_OK` the error code informs about the reason for failure:

- `E1701C_ERROR_INVALID_CARD` – a wrong or illegal card number was specified with function parameter `n`
- `E1701C_ERROR_NO_CONNECTION` – a connection to card could not be established
- `E1701C_ERROR_NO_MEMORY` – there is not enough memory available on the host to perform the requested operation
- `E1701C_ERROR_UNKNOWN_FW` – card is running an unknown and/or incompatible firmware version
- `E1701C_ERROR_TRANSMISSION` – data transmission to card failed
- `E1701C_ERROR_FILEOPEN` – opening of a file failed
- `E1701C_ERROR_FILEWRITE` – writing of data into a file failed
- `E1701C_ERROR_BORD_NA` - a base- or extension board that would be required for a function is not available
- `E1701C_ERROR_INVALID_DATA` – data or parameters handed over to a function are invalid, out of range or illegal in current context

- `E1701C_ERROR_UNKNOWN_BOARD` – trying to access a controller board that is not a suitable controller
- `E1701C_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
- `E1701C_ERROR_NOT_SUPPORTED` – the requested feature or function is not supported by the current firmware version
- `E1701C_ERROR_NO_DATA_AVAILABLE` – within a function it was tried to receive some data but there are none available yet
- `E1701C_ERROR_OUT_OF_RANGE` – a function was called with input data that are out of range
- `E1701C_ERROR_REFERENCING` – referencing an axis has failed (e.g. limit switch not found or not left)

APPENDIX A – Wiring between E1701C 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 LP8 Extension Board and optional Digi I/O Extension Board for laser alarms.

Signal Name	**Board**	**Connector / Pin**	**IPG Pin**
LP0	LP8 Extension Board	Pin 1	Pin 1
LP1		Pin 3	Pin 2
LP2		Pin 5	Pin 3
LP3		Pin 7	Pin 4
LP4		Pin 9	Pin 5
LP5		Pin 11	Pin 6
LP6		Pin 13	Pin 7
LP7		Pin 15	Pin 8
MO / Main Oscillator		Pin 8	Pin 18
LP8 Latch		Pin 17	Pin 9
LaserA / Frequency		Pin 22	Pin 20
Laser Gate / Modulation		Pin 26	Pin 19
LaserB		Pin 19	Pin 22 *)
Alarm, one of DIn0...DIn7	Digi I/O Extension Board	Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 16
Alarm, one of DIn0..DIn7		Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 21

⚠ *) 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 E1701C and JPT YDFLP series fiber laser (“MOPA”) or IPG YLP Series Type D fiber laser or Raycus RFL Series 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 LP8 Extension Board and optional Digi I/O Extension Board for laser alarms.

Signal Name	Board	Connector / Pin	JPT D-SUB25 Pin
LP0	LP8 Extension Board	Pin 1	Pin 1
LP1 / serial data		Pin 3	Pin 2
LP2 / serial clock		Pin 5	Pin 3
LP3		Pin 7	Pin 4
LP4		Pin 9	Pin 5
LP5		Pin 11	Pin 6
LP6		Pin 13	Pin 7
LP7		Pin 15	Pin 8
MO / Main Oscillator		Pin 8	Pin 18
LaserA / Frequency		Pin 22	Pin 20
Laser Gate / Modulation		Pin 26	Pin 19
LaserB / serial enable		Pin 19	Pin 22 [1])
Alarm, one of DIn0…DIn7	Digi I/O Extension Board	Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 16
Alarm, one of DIn0…DIn7		Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 21

[1]) for details regarding double-usage of this pin, please refer to the manual of the laser

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 E1701C 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 LP8 Extension Board and Digi I/O Extension Board for laser alarms with support for APD index setting via DB-25 serial data interface

Signal Name	**Board**	**Connector / Pin**	**IPG Pin**
LP0	LP8 Extension Board	Pin 1	Pin 1
LP1		Pin 3	Pin 2
LP2		Pin 5	Pin 3
LP3		Pin 7	Pin 4
LP4		Pin 9	Pin 5
LP5		Pin 11	Pin 6
LP6		Pin 13	Pin 7
LP7		Pin 15	Pin 8
MO / Main Oscillator		Pin 8	Pin 18
LP8 Latch		Pin 17	Pin 9
LaserA / Frequency		Pin 22	Pin 20
Laser Gate / Modulation		Pin 26	Pin 19
LaserB		Pin 19	Pin 22 [1])
Alarm, one of DIn0..DIn7	Digi I/O Extension Board	Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 16
Alarm, one of DIn0..DIn7		Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 21
Serial Enable	LP8 Extension Board	Pin 7	Pin 24 [2])
Serial Clock		Pin 9	Pin 13 [2])
Serial Data		Pin 11	Pin 10 [2])

⚠ [1]) 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 D – Wiring between E1701C and IPG YLP Series Type G 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!

Signal Name	Board	E1803D Connector / Pin	D-SUB25
LP0	LP8 Extension Board	Pin 1	Pin 1
LP1		Pin 3	Pin 2
LP2		Pin 5	Pin 3
LP3		Pin 7	Pin 4
LP4		Pin 9	Pin 5
LP5		Pin 11	Pin 6
LP6		Pin 13	Pin 7
LP7		Pin 15	Pin 8
MO / Main Oscillator		Pin 8	Pin 18
LP8 Latch		Pin 17	Pin 9
LaserA / Frequency		Pin 22	Pin 20
Laser Gate / Modulation		26 pin connector, pin 26	Pin 19
LaserB		Pin 19	Pin 22
GND		Pin 2 or 23	Pin 14
Alarm, one of DIn0...DIn7	Digi IO Extension Board	Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 11
Alarm, one of DIn0...DIn7		Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 16
Alarm, one of DIn0...DIn7		Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 21

APPENDIX E – Wiring between E1701C 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!

Signal Name	Board	Connector / Pin	IPG Pin
AOut0	LP8 Extension Board	Pin 10	Pin 12 [1])
MO / Main Oscillator		Pin 8	Pin 18
Laser Gate / Modulation		Pin 26	Pin 15

⚠ [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 F – Wiring between E1701C and SPI G4 Pulsed Fibre Laser / TRUMPF TruPulse nano 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:

Signal Name	Board	Connector / Pin	SPI Laser Connector Pin
LP0	LP8 Extension Board	Pin 1	Pin 17
LP1		Pin 3	Pin 18
LP2		Pin 5	Pin 19
LP3		Pin 7	Pin 20
LP4		Pin 9	Pin 51
LP5		Pin 11	Pin 52
LP6		Pin 13	Pin 53
LP7		Pin 15	Pin 54
MO / Laser Enable		Pin 8	Pin 7
LP8 Latch		Pin 17	Pin 23
LaserA / Pulse Trigger		Pin 22	Pin 47
Laser Gate / Modulation		Pin 26	Pin 5
Alarm, one of DIn0...DIn7	Digi I/O Extension Board	Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 9

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:

Signal Name	Board	Connector / Pin	SPI Laser Connector Pin
DOut0	Digi I/O Extension Board	Pin 3	Pin 17
DOut1		Pin 5	Pin 18
DOut2		Pin 7	Pin 19
DOut3		Pin 9	Pin 20
DOut4		Pin 11	Pin 51
DOut5		Pin 13	Pin 52
DOut6		Pin 15	Pin 53
DOut7		Pin 17	Pin 23
Alarm, one of DIn0...DIn7		Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 9
MO / Laser Enable	LP8 Extension Board	Pin 8	Pin 7
LaserA / Pulse Trigger		Pin 22	Pin 47
Laser Gate / Modulation		Pin 26	Pin 5

Variant 3: waveform selection, simmer, power and extended parameter control via laser controller plug in/serial interface:

Signal Name	**Board**	**Connector / Pin**	**SPI Laser Connector Pin**
MO / Laser Enable	LP8 Extension Board	Pin 8	Pin 7
LaserA / Pulse Trigger		Pin 22	Pin 47
Laser Gate / Modulation		Pin 26	Pin 5
Alarm, one of DIn0…DIn7	Digi I/O Extension Board	Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 9

In these wiring-schemes no GND-connections are listed, they have to be added in order to get valid and working connections.

APPENDIX G – Wiring between E1701C 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 LP8 Extension Board and optional Digi I/O Extension Board for laser alarms.

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

In this wiring-scheme no GND-connections are listed, they have to be added in order to get valid and working connections.

APPENDIX H – Wiring between E1701C and MaxPhotonics MFP fiber laser

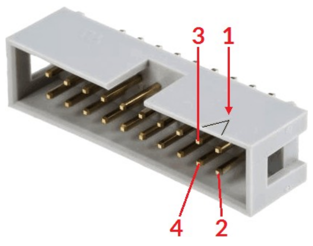
PLEASE NOTE: this wiring scheme is a non-binding policy, it may be incorrect due to changes in connected hardware. So in every case please check this table against specification and wiring documentation of the used laser!

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

Signal Name	**Board**	**Connector / Pin**	**MaxPhotonics DB25 Pin**
LP0	LP8 Extension Board	Pin 1	Pin 1
LP1		Pin 3	Pin 2
LP2		Pin 5	Pin 3
LP3		Pin 7	Pin 4
LP4		Pin 9	Pin 5
LP5		Pin 11	Pin 6
LP6		Pin 13	Pin 7
LP7		Pin 15	Pin 8
LP8 Latch		Pin 17	Pin 9
MO / Main Oscillator		Pin 8	Pin 18
LaserA / Frequency		Pin 22	Pin 20
Laser Gate / Modulation		Pin 26	Pin 19
GND		Pin 2 or 23	Pin 10-15
Alarm, one of DIn0...DIn7	Digi I/O Extension Board	Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 16
Alarm, one of DIn0...DIn7		Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 21

APPENDIX I – 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:

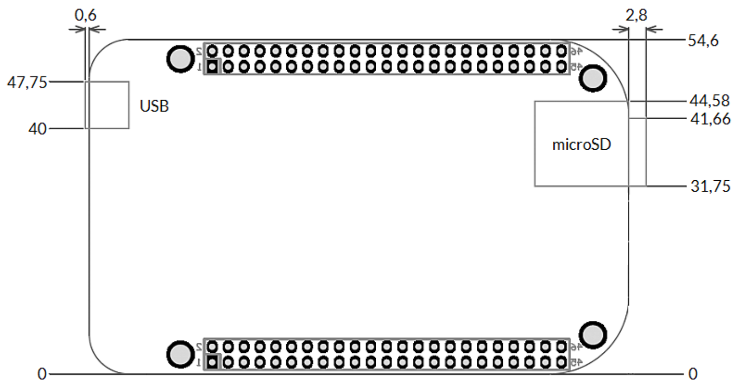


The first pin is marked by a small arrow in connector. Second pin is below of it, counting continues column-wise.

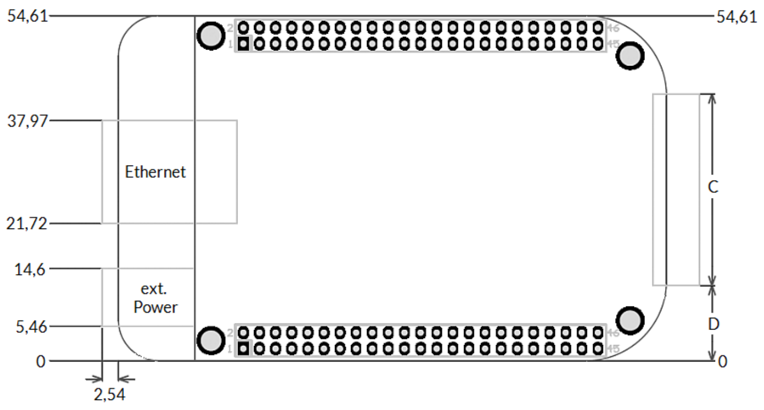
APPENDIX J – Board dimensions

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

Connectors, bottom view:

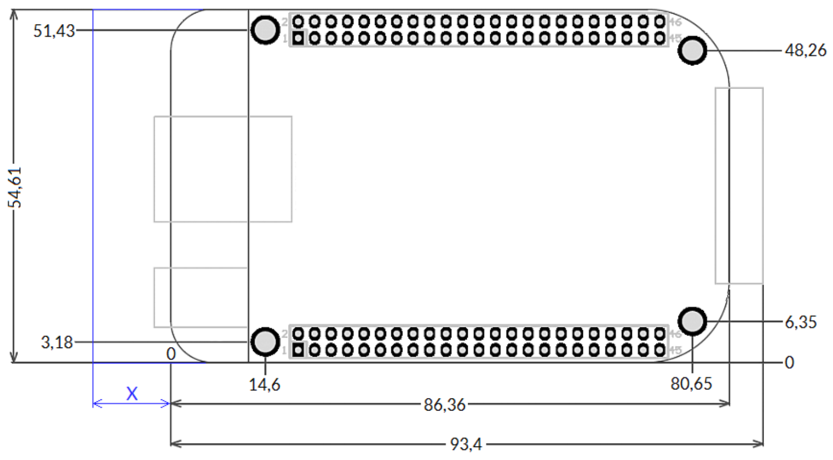


Connectors, top view:



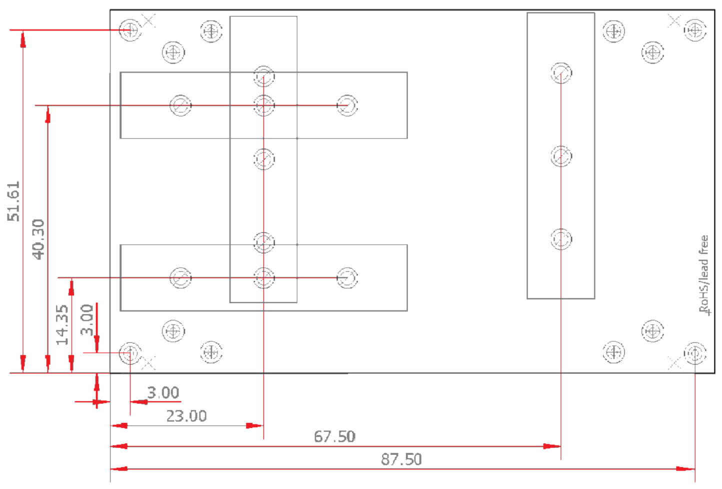
Board type	C	D
E1701C Baseboard	40 mm	7,3 mm
LP8 Extension Board	40 mm	7,3 mm
Digi I/O Extension Board	34 mm	10,3 mm

Dimensions, top view:



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

E1701base dimension drawing, all values are given in unit mm.



Index