E1701P XY Plotter Controller

Users Manual

Table Of Contents

1 Copyright

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

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

Software License Agreement

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

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

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

2 History

Date	Changes in document
08/2017	Added description for config parameter `mipout`
07/2017	Description of USB license retrieval clarified
02/2017	Images updated
11/2016	Description of `E1701P_mark_pixelline2()` added
08/2016	Board dimension drawings added
08/2016	Description of `E1701P_mark/jump_abs2()` added
06/2016	General revision
05/2016	Description of stand-alone functions added
01/2016	New reference flag added
01/2016	Wiring description added, laser wiring schemes corrected
06/2015	Initial version

3 Safety

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

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

The hardware component described within this document is also able to control motors. Motions caused by these motors 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 for use "as is". Prior to operation compliance with all relevant electric / electromagnetic safety regulations including additional hardware-controlled safety measures has to be secured. The client shall solely be responsible to strictly comply with all applicable and relevant regulations regarding installation and operation of the system at any time.

This document describes the E1701P-hardware but may contain errors and/or may be changed without further notice.

4 Overview

This document describes the E1701P XY laser plotter controller board, it's electrical characteristics and usage.

The E1701P controller boards are designed for controlling XY-table based systems with two axes, starting with firmware version 25 it supports three axes X, Y andZ. They also supply extensive signals for laser control. The communication between the host system and the controller boards is done via Ethernet or USB.

E1701P is prepared to control lasers of different types and provides required signals. Nevertheless it is possible to add any other tool instead of a laser that can be controlled via a digital signal (on/off via LaserGate output), via PWM (pulse-width modulated signal as provided by LaserA/PWM output) or via analogue output (0..5V via A0).

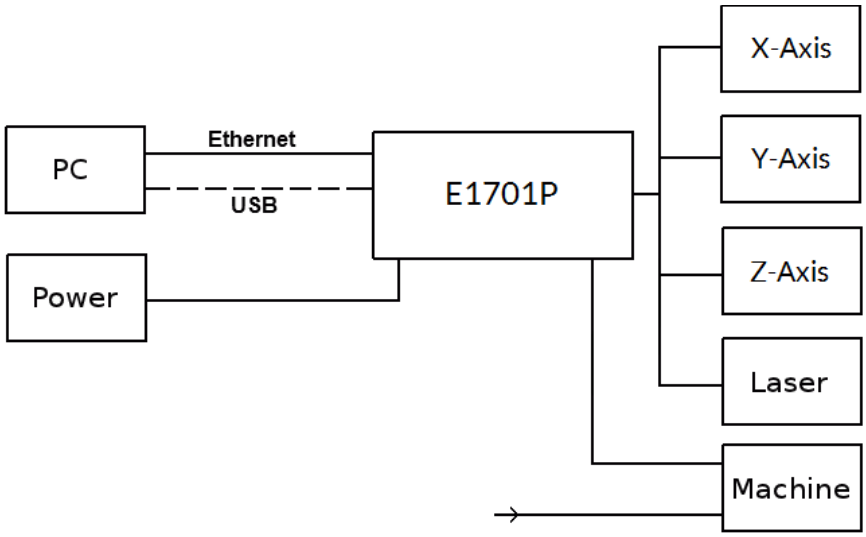
4.1 Features

E1701P plotter controller board provides following features and functionalities:

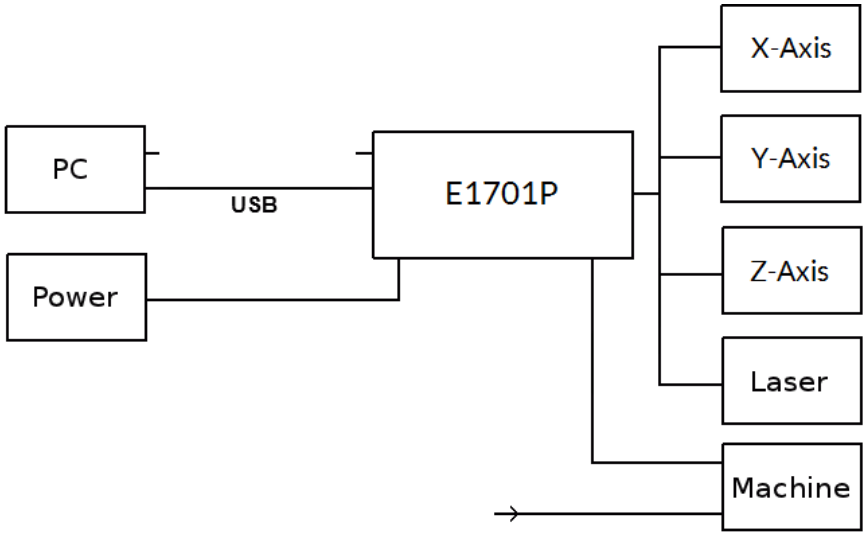
- two opto-insulated step and direction axis output signals for controlling three axes X, Y and Z via stepper motors
- 100 Mbit Ethernet connection
- USB 2.0 connection
- 20 microseconds vector cycle time and resolution (microstep period)
- command execution time down to 1 microseconds
- maximum stepper motor clock of 500 kHz, speed steps 500 kHz, 250kHz, 125kHz, 62,5 kHz, …, 5000Hz, 4950 Hz, 4901 Hz, 4854 Hz, … 1000 Hz, 998 Hz, 996 Hz, 994 Hz, ...
- realtime processing of laser and motion signals
- can control nearly every laser type
- two CMOS digital outputs for usage with YAG, CO2, IPG(tm), SPI(tm) and compatible laser types and tools (outputs can provide PWM frequency, Q-Switch, FPK-pulse, CW/continuously running frequency, stand-by frequency) running with frequencies of up to 20 MHz
- LP8 8 bit CMOS level parallel digital output e.g. for controlling laser power
- LP8 latch CMOS level digital output for usage with IPG(tm) and compatible laser types
- Master Oscillator CMOS level digital output for usage with IPG(tm) and compatible laser types
- 8 bit 0..5V analogue output e.g. for controlling laser power (this output is a slave of LP8 outputs)
- 4 freely usable digital outputs providing either CMOS level or electrically insulated outputs via external power supply
- 4 freely usable digital inputs expecting either CMOS level or electrically insulated inputs via external power supply
- 4 digital inputs usable for quadrature encoder signals to control XY axis position via external encoder
- 512 MByte DDR3 RAM
- 1 GHz CPU clock
- support for microSD and microSDHC cards
- BeamConstruct PRO license included

5 Position Within The System

The E1707P controller 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 used to retrieve BeamConstruct PRO license from the board:



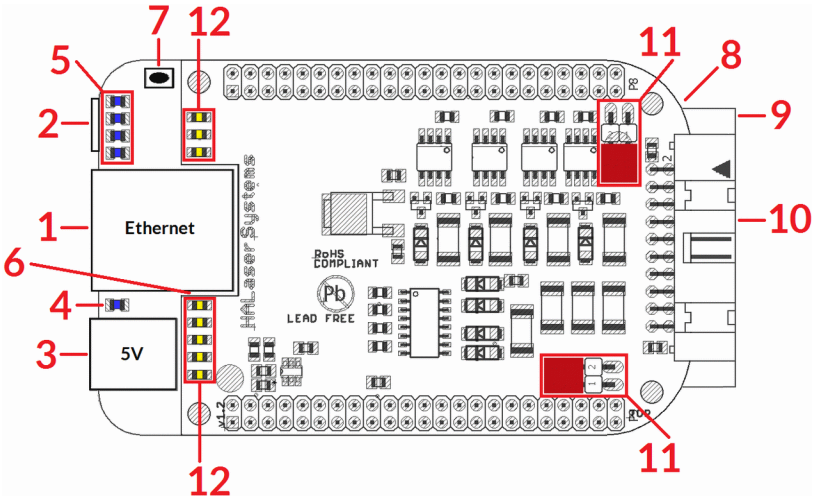
Since 100 MBit Ethernet provides much faster data transfer than USB 2.0, this connection type is preferred. Especially in case complex motion data with many short lines that result in many separate motion steps are used, Ethernet connection is more responsive.
When using USB connection with such data,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



In both cases the board itself is connected with the two axes to submit 2D position information to XY table. Beside of that it is connected to a laser to submit motion-synchronous laser data. Additional communication channels between the E1701P controller board and a connected machine can be done via remaining opto-insulated IOs.

6 Board And Connectors

The E1701P plotter controller board provides two sets of output that are located on two separate, stacked PCBs.



The E1701P XY Plotter Controller provides following connectors and interfaces:

1. Ethernet – for communication with the host system, marking information are submitted via this path
2. USB – via miniUSB connector for providing BeamConstruct PRO license to host system and optionally for submitting marking data from host to E1701P 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. MO LED (on middle board) – shows state of Master Oscillator output
7. Reset-button – on-board button to restart the board completely
8. microSD-card (on bottom side) – storage place for firmware and extended configuration file, can be used to upgrade firmware, to change the card's IP and other things more
9. Laser/control signals (on middle board) – 26 pin laser output connector which provides laser and process control IOs
10. Motor and Digital I/O (on upper board) - electrically insulated digital in- and outputs providing step and direction signals for X, Y and Z axis as well as additional in- and outputs
11. Opto-Configuration – choose operation mode for Digi I/Os
12. Input state LEDs – displaying of HIGH/LOW state of used inputs

6.1 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 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 during start-up, thus Ethernet cable connecting E1701P board and host system needs to be plugged before the board is powered up.
By default the E1701P board is using IP 192.168.2.254, thus the Ethernet network the card is connected with needs to belong to subnet 192.168.2.0/24.
PLEASE NOTE: For security reasons it is highly recommended to not to mix a standard communication network with an E1701P network or to connect the controller card with a standard network. Here it may be possible someone else in that network (accidentally) connects to that controller and causes laser emission or motion.
The IP of the controller can be changed. This is necessary e.g. in case an other subnet has to be used. The IP can be configured using e1701.cfg configuration file that is placed on microSD-card. To change the IP please perform the following steps:

1. disconnect E1701P board from power and USB

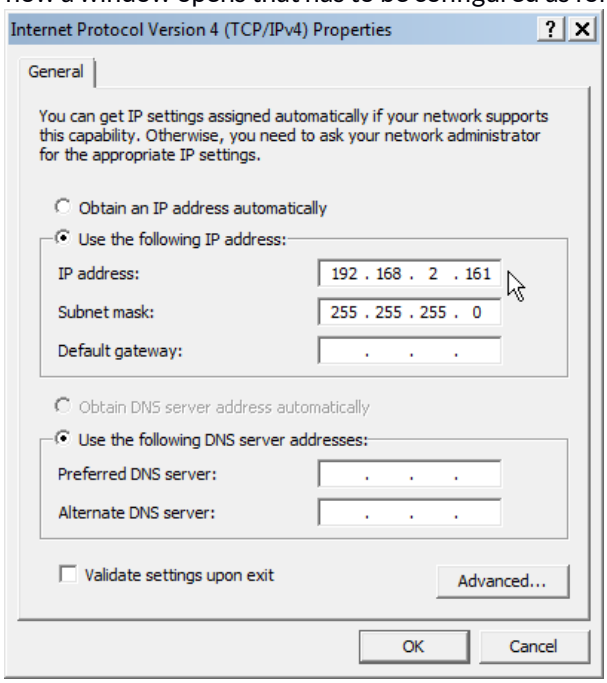
2. remove microSD-card
3. put microSD-card into a desktop computer, this may require a microSD- to SD-card-adapter
4. open the drive that is assigned to the card
5. open file e1701.cfg using a text editor like Notepad or kwrite
6. add a line or edit an existing line "`ip1=`", here the desired IP has to be appended (as example: when you want to configure IP 192.168.1.13 the line has to be "`ip=192.168.1.13`" - without any quotation signs
7. save the file
8. eject the drive the card is assigned to
9. place the microSD-card in E1701P board (place without the use of force, notice correct orientation with connectors of 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 Ethernet Configuration With Windows

When E1701P controller is accessed via Ethernet, it is recommended to have a 1:1 connection to the host PC for security reasons. Since the controller is working with a static IP (default is 192.168.2.254) the Ethernet port on host PC has to be configured with an IP of same subnet in order to allow access to it. For Windows 7 (and similar) this configuration has to be done using following steps:

1. select “Start”-button and choose entry “Control Panel”
2. Select “Network and Sharing Center”
3. Select “Change adapter settings” in upper left corner
4. find the network interface E1701P has to be connected with and select it
5. find entry “Internet Protocol Version 4 (TCP/IP4)”
   select it and press button “Properties”
   now a window opens that has to be configured as follows:

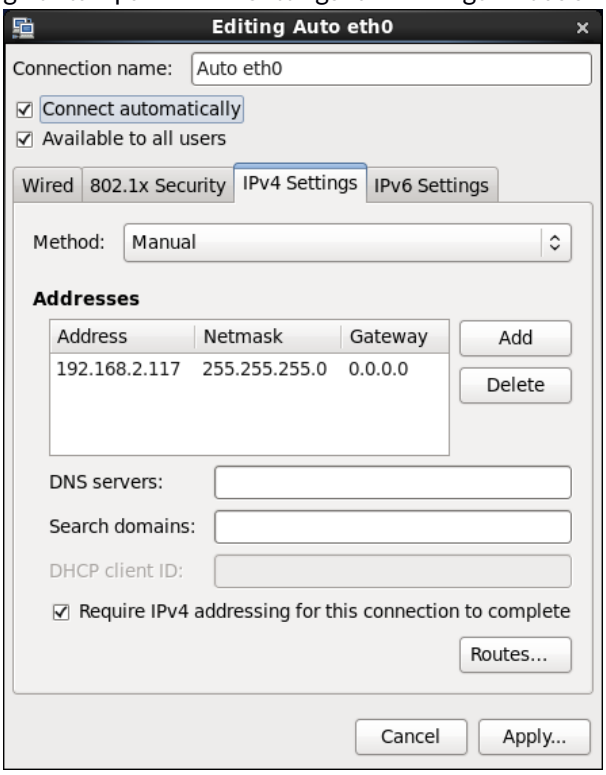


There you can specify an IP for your host PC. It has to belong to network 192.168.2.xxx and can be any number except than 192.168.2.254 (this is already the IP of the controller card), 192.168.2.0 or 192.168.1.255.

6.1.2 Ethernet Configuration With Linux

When E1701P 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 controller card is connected with and press button "Edit"
4. go to tab-pane "IPv4 Settings" and configure it as shown below:



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 controller card), 192.168.2.0 or 192.168.1.255.

6.2 USB

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

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

Required device driver is installed together with OpenAPC-setup (Windows) or comes with operating system by default (Linux). E1701P 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, additional, external power should be connected in order to operate E1701P controller correctly. Nevertheless it might be possible E1701P 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.

6.3 Power

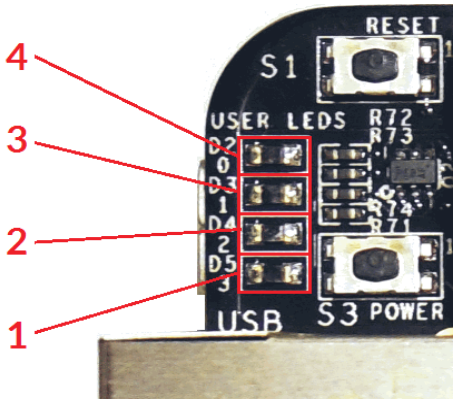
Power supply for E1701P controller board is done via power jack right beside Ethernet port. Power can be supplied via a 2.1 mm x 5.5 mm centre connector when connected to a positive power supply rated at 5V DC +/- 0.1V and 2.5A (smoothed, positive pole on inner contact). Do not apply voltages in excess 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 E1701P card is used within.

6.4 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 is not started at this point. Please refer section below for LEDs that show functional state of the board.

6.5 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 movement operations the blink frequency may go down. Only in case it does not blink for more than 10 seconds, the board has died for some reason and should be restarted.
2. Processing Active LED – this LED is turned on as long as a movement operation is running. This LED does not correspond to the laser gate signal, comparing to it it's also enabled during jumps when laser is turned off but whole operation itself is active.
3. Limit Exceeded LED – this LED is turned on optionally together with the Error-LED as soon as a motion command was detected for at least one axis where speed was beyond the controllers limit. In such cases the resulting polygon line done by the controller may also be damaged because at least one axis could not move fast enough to describe a correct line as desired. For additional details please refer below.
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 latest firmware and host software
   - check all connections and cables
   - check if "Limit Exceeded LED" is turned on too; in this case mark and jump speed of your current project need to be checked and set to lower values the controller can handle; for additional details please refer below

- undo your latest changes in hardware and configuration
If these steps do not help, please contact us for further assistance.

Limit Exceeded LED and Error LED both work in combination. Once they are lit, they are not turned off until next reboot of the controller. Assumed that no previous fatal error occurred that caused the Error LED to be turned on, following is true:

Limit Exceeded LED	Error LED	Description
OFF	OFF	All operations performed successfully until now, all operations done within allowed ranges
ON	OFF	A motion speed was too high during a jump to a new position (laser was turned off). This probably has no influence on the processing result except than for expected total processing time. Nevertheless it is recommended to check the used jump speeds to avoid an undefined behaviour.
ON	ON	A motion speed was too high during a mark operation (laser was turned on). This in most cases has an influence on the processing result because at least one axis could not move as fast as it would be necessary for a given line. In such a case the used marks speeds should be checked.

6.6 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.

6.7 Reset-Button

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

6.8 microSD-Card

The microSD card is storage place for firmware and configuration files. Here SD and SDHC cards with storage space of up to 32 GB are supported.
To remove the microSD-card, first disconnect all power from the E1701P board completely (including USB, Power LED has to go off). Next press microSD card gently into the board until you can hear a click-noise. Then you can pull it out of the board. To place a microSD card, the same has to be done in reverse order: place it into the E1701P board's card slot and press it gently until a noise signals locking of the card. Now the board can be powered.
E1701P board 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.dat – additional file that contains data used by firmware and therefore are required to operate the board
- E1701.cfg – configuration text file, can be edited using a text editor in order to modify cards configuration

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
- E1701.fwi file 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=192.168.1.100` specifies IP 192.168.1.100 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 `E1701P_set_password()`, please refer below for a detailed description. When a client computer connects to the card without sending the correct password, Ethernet connection to this host is closed immediately. PLEASE NOTE: this password does not replace any network security mechanisms and does not give the possibility to operate E1701 controller via insecure networks or Internet! It is transferred unencrypted and therefore can be "hacked" easily. Intention of this password is to avoid collisions between several E1701 cards that operate in same network and are accessed by several software instances. Maximum allowed length of the password is 48 characters. It is recommended to not to use any language-specific characters.	`passwd=myCardPwd` set a password "myCardPwd"
`standalone`	This command can be used to disable or enable a specific stand-alone operation mode. For a detailed description of possible parameters, operation modes and usage please refer related section below.	
`autofile`	Loads a special .EPR stand-alone file from SD-card in some specific stand-alone modes. For a detailed description of possible parameters, operation modes and usage please refer related section below.	`autofile=0:/markdata.epr` loads a file markdata.epr from disk; here 0:/ specifies the SD-card to be used. The .EPR-file itself can be generated within BeamConstruct out of a normal .BEAMP project file.
`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 a marking operation is in progress, the value given here can be overwritten by API-function `E1701P_digi_set_mip_output();` this parameter requires firmware version 30 or newer	`mipout=1` use DOut1 for mark-in-progress signal

6.8.1 Firmware Update

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

1. remove the microSD-Card as described above
2. download a new firmware from https://openapc.com/download/Firmware/E1701/ (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

6.9 Laser Signals

The lower, black 26 pin connector provides several signals to be used to control a laser source or any other tool that is able to handle such signals. 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.

The connector provides 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**	A0	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 /PWM	CMOS, 0/5V, max 14 mA	PWM, frequency or Q-Switch
23	GND	GND		24			
25	5V	5V		**26**	Laser Gate/Tool	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 A0 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/PWM 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 1530 Hz..20 MHz.
LaserB can be used for emitting a FPK pulse (e.g. for YAG lasers).
A0 pin provides unipolar analogue output for controlling e.g. laser power or additional equipment. This output is a slave of LP8_0..LP8_7 output, they are electrically connected an therefore can't have different values and can't be controlled by software independently. So when LP8 outputs are all LOW, A0 is on 0V. When LP8 outputs are all HIGH, A0 is 5V.

PLEASE NOTE: output of 5V at A0 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 A0 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.

6.10 Motor and Digital I/O

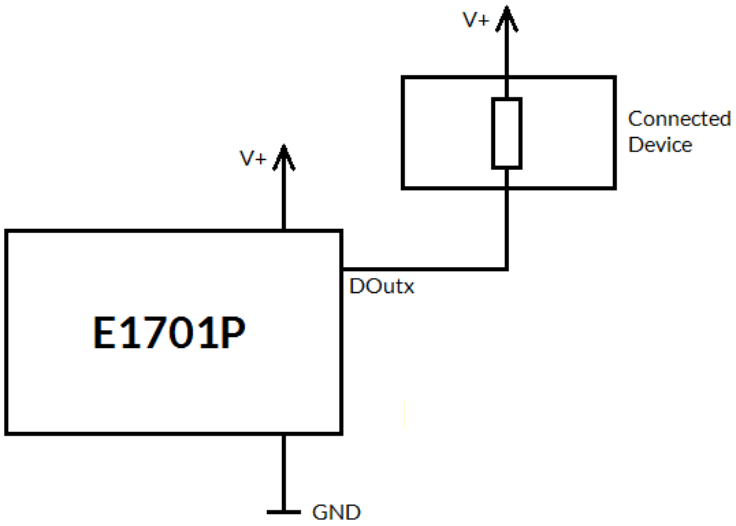
The upper, black 20 pin connector provides 8 lines for input and 8 lines for output of digital signals that can work on CMOS level (non-insulated mode) or via opto-couplers (electrically insulated mode with external

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	StepX	CMOS, 0/5V or 0/V_{ext}	Output of step-signal for X-axis, default level: LOW	**4**	DIn0	CMOS, 0/5V or 0/V_{ext}	Encoder-input 1A
5	StepY	CMOS, 0/5V or 0/V_{ext}	Output of step-signal for Y-axis, default level: LOW	**6**	DIn1	CMOS, 0/5V or 0/V_{ext}	Encoder-input 1B
7	StepZ	CMOS, 0/5V or 0/V_{ext}	Output of step-signal for Z-axis, default level: LOW	**8**	DIn2	CMOS, 0/5V or 0/V_{ext}	Encoder-input 2A
9	DOut3	CMOS, 0/5V or 0/V_{ext}	Default level: LOW	**10**	DIn3	CMOS, 0/5V or 0/V_{ext}	Encoder-input 2B
11	DirX	CMOS, 0/5V or 0/V_{ext}	Output of direction-signal for X-axis, default level: HIGH	**12**	DIn4	CMOS, 0/5V or 0/V_{ext}	
13	DirY	CMOS, 0/5V or 0/V_{ext}	Output of direction-signal for Y-axis, default level: HIGH	**14**	DIn5	CMOS, 0/5V or 0/V_{ext}	
15	DirZ	CMOS, 0/5V or 0/V_{ext}	Output of direction-signal for Z-axis, default level: HIGH	**16**	DIn6	CMOS, 0/5V or 0/V_{ext}	
17	DOut7	CMOS, 0/5V or 0/V_{ext}	Default level: HIGH	**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

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 StepX, StepY, StepZ, DOut3, DirX, DirY, DirZ, DOut7 and DIn0..DIn7 work in respect to this external power. 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 E1701P board! In this case V_{ext} is equal to V (5V) of the board and GND_{ext} is connected to boards ground GND.

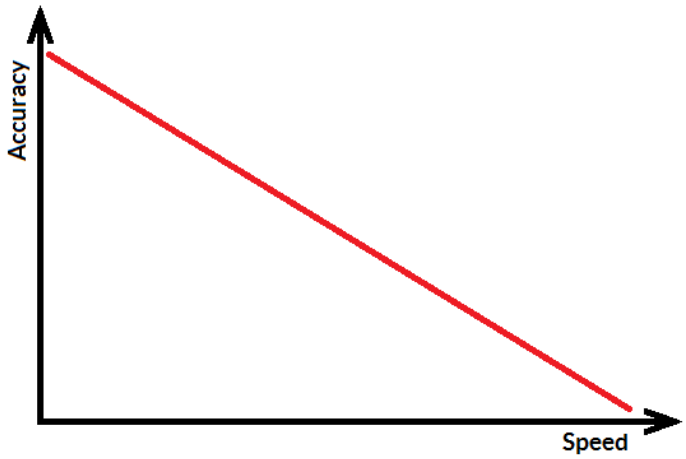
Maximum current for every output is 15 mA when internally powered (non-insulated mode), here it is recommended to use an external power supply.
Maximum current for outputs StepX, StepY, StepZ and DOut3 is 50 mA when externally powered (V_{ext} in insulated mode).
Signal output lines StepX, StepY, StepZ, DirX, DirY, DirZ, DOut3, and DOut7 work in open collector mode and have to be wired as follows:



Here “DOutx” symbolises one of the step-, direction or general outputs, 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 490 Ohms in order to not exceed the given current limits.

StepX, StepY, StepZ and DOut3 provide LOW signal level by default, DirX, DirY, DirZ and DOut7 provide HIGH level by default. These levels are valid immediately on power-up of the card.

Step-signals emitted at StepX, StepY and StepZ work with a base-frequency of 500 kHz. This is also the maximum output frequency to drive a stepper motor with. Lower speeds are calculated by doing a whole-numbered division of this base-frequency which results in possible speed steps 500 kHz, 250kHz, 125kHz, 62,5 kHz, …, 5000Hz, 4950 Hz, 4901 Hz, 4854 Hz, … 1000 Hz, 998 Hz, 996 Hz, 994 Hz, …
So as smaller the output frequency is, the more speed steps are available and the closer the actual output frequency is to the given nominal frequency. Or in other words, the higher the output speed is, the lower is the motion accuracy:



6.11 Opto-Configuration

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

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

6.12 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.

6.13 Stand-Alone Operation

The E1701P controller can be operated in stand-alone mode. In this mode all process data are stored on SD-card and the board can operate without direct control of a host-PC that sends the data to be marked. Such stand-alone marking data can be created e.g. in BeamConstruct marking software.

6.13.1 Create Stand-Alone Data with BeamConstruct

To use BeamConstruct for generation of stand-alone data for E1701P controllers, the card has to be fully configured (including all laser and pen-parameters). Next the marking data to be stored on SD-card have to be created. To generate stand-alone data, menu "Processing" submenu "Write Marking Data to File".

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

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

E1701P controller supports all static data in stand-alone mode (like all kinds of static geometries, output signals, waiting for input commands, waiting for trigger, all laser- and scanner parameters). Dynamic data like serial numbers are generated to static data too, they do not change dynamically once they are converted to .EPR files.

6.13.1.1 Stand-Alone Configuration Parameters

Within e1701.cfg configuration file of E1701 scanner controller the stand-alone operation mode has to be enabled via the configuration parameter "standalone":

```
standalone=auto
```

This means the stand-alone mode is enabled, a file specified by and additional parameter "autofile" is loaded and processed immediately. As an example a parameter: "`autofile=0:/myfile.epr`" would try to load the file "myfile.epr" from SD-card and start marking it. When no file is specified this way, the controller waits for external commands (please refer to section below).

7 Command Interface

When E1701P card is connected via USB and the USB-connection is NOT used for transmitting marking information, it can be used to send control commands to the card. Alternatively the same is true for Ethernet connection: When E1701P card is connected via Ethernet, a Telnet connection can be established to E1701P to send control commands to the card.
Such a control command 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). 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.

7.1 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 a command 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 is an unwanted behaviour when an application communicates with this interface. Using this command the serial echo mode can be turned off (parameter 0, only return values are sent back) or on (parameter 1, all data are sent back). When called with no parameters, the current echo mode value is returned.
Example: `cecho 0` – turn off echo mode

`cginp`
"**g**et **inp**ut" – get the current state of the digital inputs. 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. The output to be set is specified as a decimal number representing the bitpattern. When no parameter is given, the behaviour is undefined.

PLEASE NOTE: this command also influences the state of StepX, StepY, DirX and DirY outputs and therefore may influence a motion operation. Thus it is recommended to use it only for testing purposes and to start referencing again afterwards!
Example: `csout 128` - set DOut7 at the Digi I/O extension board to high while all others stay at low

`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.

`cgbsr`
"**g**et **b**oard **s**er**i**al 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.

`clepr <path>`
"**l**oad **epr**" – loads an EPR stand-alone file from microSD card and executes it immediately. This command can be executed in stand-alone mode only and expects the path to the file to be loaded as parameter.

Since this is the only parameter, no quotes are allowed for the pathname. The pathname itself has to be in format
`0:/filename.epr`
where `0:/` specifies the microSD-card and `.epr` is the standard extension of E1701 stand alone marking data files (this name is a shortcut for "E1701 Processing Data").
Examples: `clepr 0:/test.epr` – loads a stand-alone file "test.epr" from microSD card

`cgepr`
"**g**et **epr**" – returns the name of the currently loaded stand-alone file or an error "no file specified" when no file is loaded.

`cstat`
"**stat**e" – return the current state of the card. This command returns one of the following texts identifying the operational state:

- `marking` – card is processing some marking data currently, means either actively outputting them or waiting for an external trigger to start marking
- `stand-alone` – controller is in stand-alone mode and not yet processing any data
- `idle` - card is waiting and not marking

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

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

8 Programming Interfaces

The e1701p.dll / libe1701p.so shared library provides an own programming interface that gives the possibility to access and control the E1701P controller card.

8.1 E1701P Easy Interface Functions

These functions belong to the native programming interface of E1701 controller card and are either stream commands that are executed in the order they are called or functions that are executed immediately. Here no functions exist that provides same behaviour once as stream command and once as immediate command.

The E1701P 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 `E1701P_execute()` is called or when a card-internal threshold is exceeded.

E1701P Easy Interface uses unit "bits" as base for all units and parameters. Since E1701P card internally uses 26 bits resolution for a better accuracy and to minimize round off errors, all calculation is done with these 26

bits. So the working area always has a size of 26 x 26 bits equal to 67108864 x 67108864. Independent from real resolution and output of hardware all calculations have to be done within this 26 bit range.

E1701P Easy Interface provides following functions:

`unsigned char E1701P_set_connection(const char *address)`

This function has to be called as very first. It is used to specify the IP address where the card is accessible at (in case of Ethernet connection) or the serial interface (in case of USB connection, "COMx" for Windows and "/dev/ttyACMx" for Linux where "x" is the number of its interface). By default IP 192.168.1.254 is used. This is the only function that has to be called in case of both, when compatibility functions and when the E1701P easy function interface is used.
It returns a card index number that has to be used with all following functions.

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 E1701P_set_password(const char n,const char *ethPwd)`

Sets a password that is used for Ethernet connection of E1701 card. The same password should be configured on E1701P 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 E1701P 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 E1701P_set_logfile(unsigned char n,const char *path);`

Using this function a path to a file can be specified where controller log data are written into. These data are uploaded from controller during normal operation cyclically and can be used to find errors in configuration and hardware setup.

Parameters:
`n` – the 1-based board instance number as returned by `E1701P_set_connection()`
path – full pathname to a file where the log data have to be written into, this file does not have to exist, it will be overwritten on every new start-up of software. Write access is required on specified location for current user. When an empty path is specified, a current log file is closed and writing of log data is disabled.

Return: `E1701P_OK` in case function could be completed successfully or an `E1701P_ERROR_`-code otherwise

`void E1701P_close(unsigned char n)`

Closes the connection to a card and releases all related resources. After this function was called, no more commands can be sent to the card until `E1701P_set_connection()` is called again.

Parameters:
`n` – the 1-based board instance number as returned by `E1701P_set_connection()`

int E1701P_open_connection(unsigned char n)

This function has to be called for first time on initialisation and before any vector data are sent to the board. It is mandatory to call this function at least once since it establishes connection to E1701P card.
This is not a stream-command, means its data may be applied immediately and independent from current stream state.

Parameters:
`n` – the 1-based board instance number as returned by `E1701P_set_connection()`

Return: `E1701P_OK` or an `E1701P_ERROR_` in case of an error

int E1701P_set_xy_correction(unsigned char n, double gainX, double gainY,double rot,int offsetX,int offsetY)

Sets size correction factor and offset for X and Y direction of working area as well as a rotation. 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 `E1701P_set_connection()`
`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

Return: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

int E1701P_set_speeds(unsigned char n, double jumpspeed,double markspeed)

Set motion speed values to be used for all following vector data and until not replaced by other speed values.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1701P_set_connection()`
`jumpspeed` – table movement speed during jumps (movements when laser or tool is off) in unit bits/msec and range 1..4294960000
`markspeed` – table speed during mark (movements when laser or tool is on) in unit bits/msec and range 1..4294960000

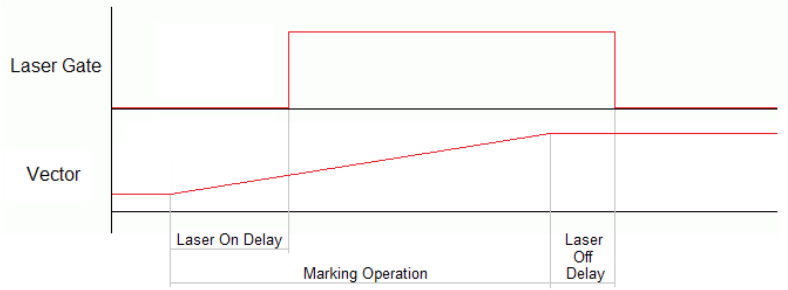
Return: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

int E1701P_set_laser_delays(unsigned char n,double ondelay,double offdelay)

Set laser delay values to be used for all following vector data and until not replaced by other delay values.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1701P_set_connection()`
`ondelay` – laser on delay in unit microseconds

`offdelay` – laser off delay in unit microseconds



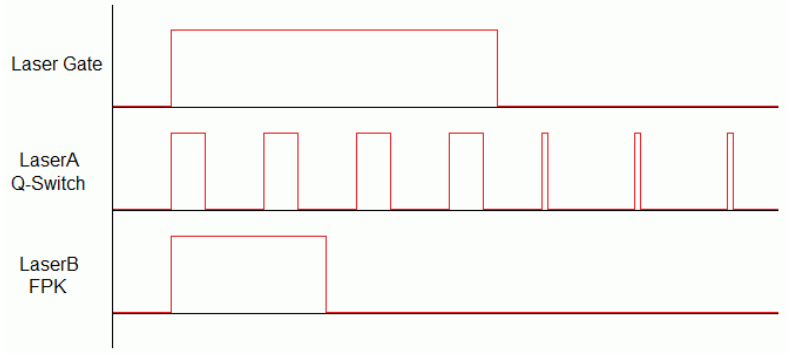
Return: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

`int E1701P_set_laser_mode(unsigned char n, unsigned int mode)`

Sets the laser mode to be used for all following operations, this value influences the signals emitted at the connectors of the card. This function has to be called prior to setting any other laser parameters (like frequency, standby-frequency, power).
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

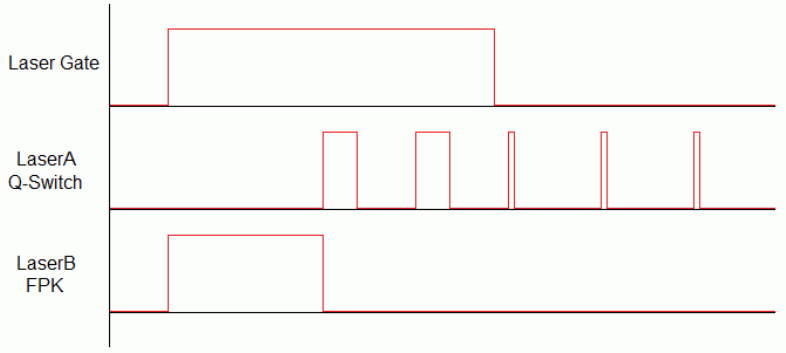
Parameters:
`n` – the 1-based board instance number as returned by `E1701P_set_connection()`
`mode` - the laser mode, here one of the following values is possible:

- `E1701P_LASERMODE_CO2` – for controlling CO2 lasers, this mode supports stand-by frequency at LaserA/PWM output (to be set with function `E1701P_set_standby()`) and PWM-modulated frequencies during marking and for power control (to be set with function `E1701P_set_laser_timing()`); this mode should also be used in case a tool has to be accessed which can be controlled via PWM signal
- `E1701P_LASERMODE_YAG1` – for controlling YAG lasers, this mode supports stand-by and Q-Switch frequency at LaserA output (to be set with function `E1701P_set_standby()`) and a first pulse killer signal at output LaserB that is issued on beginning of a mark together with the Q-Switch frequency (to be set with function `E1701P_set_fpk()`):



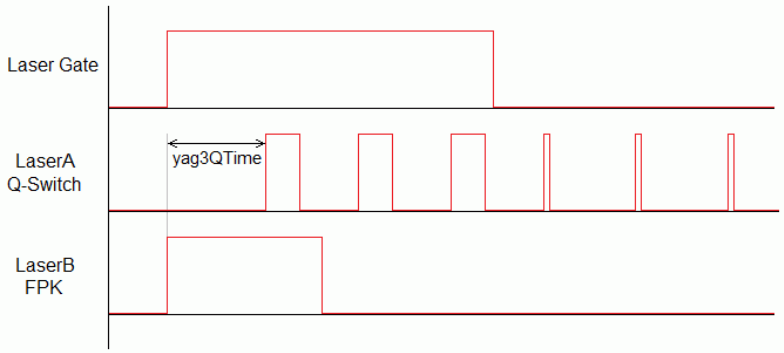
Here Q-Switch signal is started together with laser gate and FPK pulse. At end of mark when laser gate is turned off stand-by frequency is emitted at LaserA.

- `E1701P_LASERMODE_YAG2` – for controlling YAG lasers, this mode supports stand-by and Q-Switch frequency at LaserA output (to be set with function `E1701P_set_standby()`) and a first pulse killer signal at output LaserB that is issued on beginning followed by Q-Switch frequency that starts when FPK pulse has finished:



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

- `E1701P_LASERMODE_YAG3` – for controlling YAG lasers, this mode supports stand-by and Q-Switch frequency at LaserA output (to be set with function `E1701P_set_standby()`) and a first pulse killer signal at output LaserB that is issued on beginning followed by Q-Switch frequency that starts after a freely configurable time period "yag3QTime":



Here FPK and laser gate are started together. Q-Switch signal is started after yag3QTime has elapsed according to the beginning of FPK pulse. This time value can be set using function `E1701P_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.

- `E1701P_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 `E1701P_set_standby()`.

Return: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

`int E1701P_jump_abs2(const unsigned char n,const int x,const int y,const int z)`

Perform a jump (movement with laser or tool turned off) to the given position. This causes a movement from current position to the one specified by this functions parameters using the jump speed. When laser or tool was turned on before this function is called, laser/tool is turned off at the beginning with a delay specified by laser off 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 `E1701P_set_connection()`
`x` – the x-coordinate in unit bits the table has to jump to
`y` – the y-coordinate in unit bits the table has to jump to
`z` – the z-coordinate in unit bits the table has to jump to

Return: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

`int E1701P_mark_abs2(const unsigned char n,const int x,const int y,const int z)`

Perform a mark (movement with laser or 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. When laser or tool was turned off before this function is called, laser/tool 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 E1701P_set_connection()
x – the x-coordinate in unit bits the table has to move to
y – the y-coordinate in unit bits the table has to move to
z – the z-coordinate in unit bits the table has to move to

Return: E1701P_OK or an E1701P_ERROR_-return code in case of an error

int E1701P_mark_pixelline2(const unsigned char n,int x,int y,int z,const int pixWidth,const int pixHeight,const int pixDepth,const unsigned int pixNum,double *pixels,E1701P_power_callback power_callback,void *userData)

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

Parameters:
n – the 1-based board instance number as returned by E1701P_set_connection()
x, y, z – the starting coordinates of the line in unit bits
pixWidth – the width of a single pixel (in unit increments), when this is set to a value greater or smaller than 0 while all the others are equal 0, a horizontal line (X-direction) is drawn; the sign of the value specifies the marking direction
pixHeight – the height of a single pixel (in unit increments), when this is set to a value greater or smaller than 0 while all the others are equal 0, a vertical line (Y-direction) is drawn; the sign of the value specifies the marking direction
pixDepth – the depth of a single pixel (in unit increments), when this is set to a value greater or smaller than 0 while all the others are equal 0, a line into Z-direction is drawn; the sign of the value specifies the marking direction
pixNum – the number of pixel data contained in the array of intensity values handed over with the following parameter
pixels – an array of double-values with a length equal the number of pixels specified with pixNum and with an allowed range of 0.0..100.0 specifying the intensity; every entry of this array is equal to one pixel of the bitmap, so a greyscale-pixelline with brightness values in range 0..255 has to be converted to values in range 0.0..100.0
power_callback – this is a callback function of type

```
int (*E1701P_power_callback)(unsigned char n, double power, void *userData)
```

which is used to set the power for every pixel. There these E1701P_-functions have to be called that belong to the used laser type and set the power values according to it's hardware capabilities. Within the power callback function only stream commands are allowed to be called. It is not possible to use external devices that are not synchronous to E1701P command stream. The power callback has to return with E1701P_OK when setting of power was successful. In case of an error the appropriate error code has to be returned, the pixel marking function will be canceled in such a case too and does not finish marking of the line. Parameter n is the 1-based board instance number specifying the board the power has to be changed for, power is the power to be set in unit percent and userData are some free to use, custom data that can be handed over on call to E1701P_mark_pixelline().

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

Return: E1701P_OK or an E1701P_ERROR_-return code in case of an error

int E1701P_set_matrix(unsigned char n, double m11, double m12, double m21, 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.
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 `E1701P_set_connection()`
`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: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

`int E1701P_execute(unsigned char n)`

Starts execution of all previously sent commands in case card is not already outputting these data. The E1701P 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 `E1701P_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 `E1701P_execute()`. But in situations where only very few data are sent to the card it is necessary to call this function always in order to start marking also in these cases where the internal fill threshold is not reached and where the card would not start marking immediately. Thus it is recommended to always call this function after all marking data have been sent.
Marking is finished only when STOP 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 `E1701P_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 `E1701P_set_connection()`

Return: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

`int E1701P_stop_execution(unsigned char n)`

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

Parameters:
`n` – the 1-based board instance number as returned by `E1701P_set_connection()`

Return: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

`unsigned int E1701P_get_card_state(unsigned char n)`

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

Parameters:
`n` – the 1-based board instance number as returned by `E1701P_set_connection()`

Return: a bit pattern of or-concatenated constants specifying the current state:

- `E1701P_CSTATE_MARKING` - card is currently marking
- `E1701P_CSTATE_PROCESSING` - card has received some data that are enqueued for marking

`int E1701P_delay(unsigned char n, double delay)`

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

Parameters:
`n` – the 1-based board instance number as returned by `E1701P_set_connection()`
`delay` - time to wait until marking continues in unit usec, smallest possible value is 1 usec

Return: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

`int E1701P_set_laser_timing(unsigned char n, double frequency, double pulse)`

Set the frequency and pulse-width to be used during marking at LaserA/PWM 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 `E1701P_set_connection()`
`frequency` – emitted frequency in unit Hz and in range 1530..20000000 Hz
`pulse` – pulse width in usec, this value has to be smaller than period length that results out of frequency

Return: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

`int E1701P_set_laserb(const unsigned char n,const double frequency,const double pulse)`

Set the frequency and pulse-width to be used at LaserB output. To use LaserB as second frequency output, a lasermode with flag `E1701P_LASER_FREQ_DUAL` has to be configured.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.
Parameters:
`n` – the 1-based board instance number as returned by `E1701P_set_connection()`
`frequency` – emitted frequency in unit Hz and in range 1530..20000000 Hz
`pulse` – pulse width in usec, this value has to be smaller than period length that results out of frequency
Return: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

`int E1701P_set_standby(unsigned char n, double frequency, double pulse)`

Set the frequency and pulse-width to be used during jumps, as stand-by frequency or as continuously running frequency at LaserA/PWM 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:
`frequency` – emitted frequency in unit Hz and in range 1530..20000000 Hz
`pulse` – pulse width in usec, this value has to be smaller than period length that results out of `frequency`
`n` – the 1-based board instance number as returned by `E1701P_set_connection()`

Return: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

`int E1701P_set_fpk(unsigned char n, double fpk, double yag3QTime)`

Set the parameters for first pulse killer signal that 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 `E1701P_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 `E1701P_LASERMODE_YAG3` is set, elsewhere it is ignored

Return: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

`void E1701P_get_version(unsigned char n, unsigned short *hwVersion, unsigned short *fwVersion)`

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

Parameters:
`n` – the 1-based board instance number as returned by `E1701P_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 E1701P_lp8_write(unsigned char n, unsigned char value)`

Sets the LP8_0..LP8_7 outputs of 8 bit laser port and toggles the related latch output automatically; calling this function sends following sequence of commands to the card:

- turn latch bit on
- set LP8 outputs
- turn latch bit off

Total execution time of this command is 4 usecs for setting LP8 outputs and toggling latch. Latch pulse width is about 2 usec.
This function also changes the value at the analogue A0 output, it is updated on falling edge of latch pulse.
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 `E1701P_set_connection()`
`value` – the 8 bit value to be set at LP8 port

Return: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

int E1701P_lp8_write_latch(unsigned char n, unsigned char on, double delay1,unsigned char value, double delay2,double delay3)

Sets the LP8 8 bit laser port 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 is 3 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 A0 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 `E1701P_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: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

int E1701P_lp8_write_mo(unsigned char n, unsigned char on)

Sets the master oscillator output MO to be used with e.g. fiber lasers.
This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1701P_set_connection()`
`on` – the state the MO output has to be switched to

Return: `E1701P_OK` or an `E1701P_ERROR_`-return code in case of an error

int E1701P_lp8_a0(unsigned char n,unsigned char value)

This is a convenience function, it sets an 8 bit value to the A0 analogue output which is directly connected with and works in parallel to LP8 output. Because of this setting A0 can't be done independent from LP8_0..LP8_7 and LP8 latch digital outputs.
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 `E1701P_set_connection()`
`value` – the 8 bit value to be set at A0 analogue output

Return: `E1701P_OK` or an `E1701P_ERROR_` return code in case of an error

int E1701P_digi_write2(const unsigned char n,unsigned int value,unsinged int mask)

Sets the digital outputs StepX, StepY, StepZ, DOut3, DirX, DirY, DirZ, DOut7 by giving a bitpattern that represents these values.
PLEASE NOTE: when setting/resetting any of the step- or direction outputs while a motor is connected there, this causes either a movement or a change in direction of running movements! So the related bits should not be touche, means they should be set to 0 in parameter `mask` to avoid unexpected behaviour!
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 `E1701P_set_connection()`
`value` – the 8 bit value to be set at digital out port
`mask` – specifies which of the bits given in `value` have to be changed, only these outputs of `value` are set/re-set where the corresponding mask-bit is set to 1

Return: `E1701P_OK` or an `E1701P_ERROR_` return code in case of an error

unsigned int E1701P_digi_read(unsigned char n)

Reads the 8 bit digital inputs Din0..Din7.
This is not a stream-command, means it is executed immediately and returns current state of the digital inputs. When encoder input is enabled, digital inputs 0 and 1 and/or 2 and 3 are not available as standard inputs. In this case state of these bits is undefined and does not reflect the current input state caused by the external encoder.

Parameters:
`n` - the 1-based board instance number as returned by `E1701P_set_connection()`

Return: the 8 bit value currently set at digital input port or `0xFFFFFFFF` in case of an error

int E1701P_set_accels(unsigned char n, unsigned char axes, double accel, double decel, double stopDecel)

Set acceleration (ramping) values for acceleration, deceleration and in case a stop-event occurs. When one of the values is set to 0, related acceleration/deceleration will not be performed and the axis is started or stopped immediately. This may lead to rough movements or to overshoot with losing its exact position. Acceleration and deceleration is not given in a value that is equal to a physical measurement unit but as a factor that describes strength of acceleration and depends on used acceleration mode.
This command is sent to the card asynchronously, there is no response for it.

Parameters:
`n` – the 1-based board instance number as returned by `E1701P_set_connection()`
`axes` – flags that specify for which axes these parameters have to be set, here OR-concatenated `E1701P_AXIS_x`-flags have to be used; when referencing is running for one or more of the axes specified here, this command is dropped for these axes
`accel` – acceleration of selected axes on start, this value applies when a movement begins
`decel` – deceleration of selected axes on end, this value applies when a movement has to stop regularly; this deceleration is also used on referencing when reference-switch could not be found and axis stops movement due to position-timeout
`stopDecel` – deceleration of selected axes in case of stop condition, this value applies when movement is stopped by a stop-command or because a switch was hit or left; this deceleration is also used on referencing when a reference switch is hit or left

PLEASE NOTE: setting a stop deceleration value greater than 0 lets the related axes continue their movement for the time required for deceleration, means no immediate stop is performed. Depending on situation this may result in run over limit or reference switches which may be unwanted. So deceleration value given here should be as large as possible to have an as small as possible deceleration time and travel distance!

Return: E1701P_OK in case function could be completed successfully or an E1701P_ERROR_-code otherwise

int E1701P_set_accel_modes(unsigned char n, unsigned char axes, unsigned int accelMode, unsigned int decelMode, unsigned int res2)

Sets or changes acceleration and deceleration mode of axes. By default linear mode is selected, using this function an other mode can be chosen.

Parameters:
n – the 1-based board instance number as returned by E1701P_set_connection()
axes – flags that specify for which axes these parameters have to be set, here OR-concatenated E1701P_AXIS_x-flags have to be used; when referencing is running for one or more of the axes specified here, this command is dropped for these axes
accelMode – acceleration mode to be set for specified axes, here one of the following values is possible:

- E1701P_ACCEL_MODE_LIN – linear acceleration mode, here acceleration is constant until nominal speed is reached, this is a smooth mode where speed is reached not very fast
- E1701P_ACCEL_MODE_EXP – exponential acceleration mode, here acceleration increases with speed which lets the axis reach the target speed quite fast but may cause problems when acceleration is set to 0, in some situations here increments may get lost
- E1701P_ACCEL_MODE_SSHAPE - very soft acceleration mode where acceleration itself increases during beginning and decreases before target speed is reached, this mode can be used to have high speeds with inertial masses and without losing any increments but it reaches target speed slower than all other modes

decelMode – deceleration mode to be set for specified axes, here one of the following values can be handed over:

- E1701P_DECEL_MODE_LIN – linear mode, here deceleration is constant until axis is stopped, this is a smooth mode where stopping an axis doesn't happens very fast
- E1701P_DECEL_MODE_EXP – exponential mode, here axis starts with a high deceleration value which decreases over time and stops with a small deceleration. In this mode axis is stopped quite fast but may cause problems at the beginning when using the high deceleration, in some situations here increments may get lost
- E1701P_DECEL_MODE_SSHAPE - very soft deceleration mode where deceleration itself increases during beginning and decreases before axis is stopped, this mode can be used to stop from high speeds with inertial masses and without losing any increments but it stops slower than all other modes

res – unused parameter, set to 0 to be compatible with future versions

Return: E1701P_OK in case function could be completed successfully or an E1701P_ERROR_-code otherwise

int E1701P_set_limits(unsigned char n,unsigned char axes,int llimit,int hlimit,double slimit)

Set maximum movement range and speed limit for all subsequent motion commands. This command is sent to the card asynchronously, there is no response for it.

Parameters:
n – the 1-based board instance number as returned by E1701P_set_connection()
axes – flags that specify for which axes these parameters have to be set, here OR-concatenated E1701P_AXIS_x-flags have to be used; when referencing is running for one or more of the axes specified here, this command is dropped for these axes
llimit – lower movement limit, when a later call specifies a movement position beyond this value, this movement is limited
hlimit – upper movement limit, when a later call specifies a movement position beyond this value, this movement is limited
slimit – speed limit (using unit increments/second); when this value is set to 0, later calls for setting a speed value are used with the given speed and without any limitation. In case a speed limit was specified, all used speed values are limited to the value given here.

Return: E1701P_OK in case function could be completed successfully or an E1701P_ERROR_-code otherwise

int E1701P_set_stopcond(unsigned char n,unsigned char axes,unsigned char stopOnEnter,unsigned char stopOnLeave)

Defines stop-conditions for axes. This function gives the possibility to freely define inputs as limit switches at which a movement has to stop. Here two states for inputs can be defined that are used as stop condition: when a switch is entered (input is set to HIGH) or when a switch is left (input goes to LOW). It is possible to use more than one input for stopping a motion. In this case at least one stopOnEnter input bit has to be set or all stopOnLeave input bits have to be reset to fulfill a stop condition.

PLEASE NOTE: when external encoder 0 or 1 is configured, input bits 0 and 1 or 2 and 3 are not available for checking stop condition.

When all input bits for "stop on enter" and "stop on leave" are set to 0, this function is disabled and motion works independent from input states.

As long as a stop condition is fulfilled, no more motion is possible. In this case the stop condition specified with this function has to be cleared by setting stopOnEnter and stopOnLeave to 0, axis has to be moved in other direction until the switches are in a state where stop condition is no longer fulfilled and previous stop condition values have to be restored. If movement was stopped by such a condition can be checked by calling E1701P_get_axis_state(). Which inputs are high and which are low (to find out which switches caused the stop and in which direction to move next) can be checked by calling E1701P_get_inputs().

This command is sent to the card asynchronously, there is no response for it.

Parameters:

n – the 1-based board instance number as returned by E1701P_set_connection()

axes – flags that specify for which axes the stop conditions have to be set, here OR-concatenated E1701P_AXIS_x-flags have to be used; when referencing is running for one or more of the axes specified here, this command is dropped for these axes

stopOnEnter – bit pattern specifying which input pins stop movement on HIGH-signal

stopOnLeave – a bit pattern specifying which input pins have to be LOW to stop movement

Return: E1701P_OK in case function could be completed successfully or an E1701P_ERROR_-code otherwise

int E1701P_reference(unsigned char n, unsigned char axes, unsigned int mode, unsigned char refSwitch, double speedStep1,double speedStep2,double speedStep3,double speedStep4)

This function starts a reference run for specified axes. This requires one input pin to be used as switch defining the reference position. As long as referencing is active, all other commands sent for this axis (defined by axis flags of other functions) are ignored and dropped. Motion commands for other axes than the referenced ones still can be sent to the controller and will be processed in parallel.

This is a synchronous command, using E1701P_get_axis_state() it has to be checked if referencing has started and finished before any other command can be sent to the card.

Reference movements start with the standard acceleration specified with E1701P_set_accels() and stops with the stop-deceleration specified with E1701P_set_accels().

When a stop-condition is met during referencing by hitting a limit switch for the first time, referencing direction is inversed to auto-search for reference switch.

When a stop-condition is met during referencing by hitting a limit switch for the second time, referencing is canceled. This can be checked via the E1701P_AXIS_STATE_CONDSTOP-flag of function E1701P_get_axis_state().

After referencing has finished successfully, function E1701P_set_pos() can be called to assign a defined position value to the current axis position.

Parameters:

n – the 1-based board instance number as returned by E1701P_set_connection()

axes – flags that specify for which axes referencing has to be started, here OR-concatenated E1701P_AXIS_x-flags have to be used

`mode` – this is a bunch of OR-concatenated `E1701P_REFSTEP_x_y`-flags where `x` is the number of the step and `y` defines the movement that has to be performed during this referencing step. `x` has to be continuous, every number specified for x is allowed to exist only once. Here following flags do exist and can be combined to specify a referencing sequence:

- `E1701P_REFSTEP_1_ENTER_N` – on first step move in negative direction until the reference switch is hit
- `E1701P_REFSTEP_1_ENTER_P` – on first step move in positive direction until the reference switch is hit
- `E1701P_REFSTEP_2_ENTER_N` – on second step move in negative direction until the reference switch is hit
- `E1701P_REFSTEP_2_ENTER_P` – on second step move in positive direction until the reference switch is hit
- `E1701P_REFSTEP_2_LEAVE_N` – on second step move in negative direction until the reference switch is left
- `E1701P_REFSTEP_2_LEAVE_P` – on second step move in positive direction until the reference switch is left
- `E1701P_REFSTEP_3_LEAVE_N` – on third step move in negative direction until the reference switch is left
- `E1701P_REFSTEP_3_LEAVE_P` – on third step move in positive direction until the reference switch is left
- `E1701P_REFSTEP_3_ENTER_N` – on third step move in negative direction until the reference switch is hit
- `E1701P_REFSTEP_3_ENTER_P` – on third step move in positive direction until the reference switch is hit
- `E1701P_REFSTEP_4_ENTER_N` – on fourth step move in negative direction until the reference switch is hit
- `E1701P_REFSTEP_4_ENTER_P` – on fourth step move in positive direction until the reference switch is hit
- `E1701P_REFSTEP_4_LEAVE_N` – on fourth step move in negative direction until the reference switch is left
- `E1701P_REFSTEP_4_LEAVE_P` – on fourth step move in positive direction until the reference switch is left
- `E1701M_REFSTEP_INV_SWITCH` – this is a special flag which has influence on the logic of the input switch; when set, its behaviour is inverted, means reaction on hit/leave as described above changes; this function requires firmware version 23 or newer

Some examples for useful combinations:
`E1701P_REFSTEP_1_ENTER_N|E1701P_REFSTEP_2_LEAVE_P|E1701P_REFSTEP_3_ENTER_N|E1701P_REFSTEP_4_LEAVE_P` – this is for very accurate referencing and requires related speed values becoming slower for every step. Here axis moves in negative direction until reference switch is hit, next it moves in positive direction until it is left. This is repeated, next it again moves in negative direction until reference switch is hit, during last step it moves in positive direction until it is left again. As lower the speed for step 4 is, as more exact the referenced position will be.

`E1701P_REFSTEP_1_ENTER_P|E1701P_REFSTEP_2_ENTER_N|E1701P_REFSTEP_3_LEAVE_N` – this is a special sequence that assumes the reference switch may be hit but traversed in first step because speed is too high or stop-deceleration too slow to fully stop the axis while the switch is held. So after traveling in positive direction until the switch is hit, the axes move back in negative direction until the switch is hit again. Next movement in negative direction is continued until the switch is left. Here optionally a fourth step `E1701P_REFSTEP_4_ENTER_P` could be added to hit the reference switch again

`refSwitch` – bit pattern defining at least one input pin that is used as reference input. In theory it is possible to define more than one input bit for more than one reference switch here. In this case referencing would act on the first reference switch found during motion. In practice multiple-reference-switch-feature should be useful in very few, exotic cases only.

PLEASE NOTE: when external encoder 0 or 1 is configured, input bits 0 and 1 or 2 and 3 are not available for usage as reference switch input.

speedStep1 – speed to be used during first referencing step. When a speed value <=0 is set here although a E1701P_REFSTEP_1_-flag is set, referencing will not be done
speedStep2 – speed to be used during second referencing step. When a speed value <=0 is set here although a E1701P_REFSTEP_2_-flag is set, referencing will be finished after step 1
speedStep3 – movement speed to be used during third referencing step. When a speed value <=0 is set here although a E1701P_REFSTEP_3_-flag is set, referencing will be finished after step 2
speedStep4 – movement speed to be used during fourth referencing step. When a speed value <=0 is set here although a E1701P_REFSTEP_4_-flag is set, this last step will not be done

Return: E1701P_OK in case function could be completed successfully or an E1701P_ERROR_-code otherwise

int E1701P_set_pos(unsigned char n, unsigned char axes,int pos)

Sets a specific position for defined axes. This function does not cause any movements but changes the current position value of the axis. When an axis makes use of the encoder input, the encoder counter is set to a value that corresponds to this position, for all other axes the internal position counter is set to the given value. This command is sent to the card asynchronously, its success can be tested by checking E1701P_AXIS_STATE_SETPOS flag returned by E1701P_get_axis_state()

Parameters:
n – the 1-based board instance number as returned by E1701P_set_connection()
axes – flags that specify for which axes the positions have to be set, here OR-concatenated E1701P_AXIS_x-flags have to be used; when referencing is running for one or more of the axes specified here, this command is dropped for these axes
pos - new position value for the axes specified by flags in parameter axes (using unit increments)

Return: E1701P_OK in case function could be completed successfully or an E1701P_ERROR_-code otherwise

8.1.1 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 E1701P_OK the error code informs about the reason for failure:

- E1701P_ERROR_INVALID_CARD – a wrong or illegal card number was specified with function parameter n
- E1701P_ERROR_NO_CONNECTION – a connection to card could not be established
- E1701P_ERROR_NO_MEMORY – there is not enough memory available on the host
- E1701P_ERROR_UNKNOWN_FW – card is running an unknown and/or incompatible firmware version
- E1701P_ERROR_TRANSMISSION – data transmission to card failed
- E1701P_ERROR_INVALID_DATA – data or parameters handed over to a function are invalid, out of range or illegal in current context
- E1701P_ERROR_UNKNOWN_BOARD – trying to access a controller board that is not a motion controller
- E1701P_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
- E1701P_ERROR – an other, unspecified error occurred
- E1701_ERROR_NOT_SUPPORTED – the requested feature or function is not supported by the current firmware version

APPENDIX A – Wiring between E1701P 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!

Signal Name	Connector / Pin	IPG Pin
LP0	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 / Master Oscillator	Pin 8	Pin 18
LP8 Latch	Pin 17	Pin 9
LaserA / Frequency	Pin 22	Pin 20
Laser Gate / Modulation	Pin 26	Pin 19
Alarm, one of DIn2, DIn3, DIn6, DIn7	Pin 8, 10, 16 or 18	Pin 16
Alarm, one of DIn2, DIn3, DIn6, DIn7	Pin 8, 10, 16 or 18	Pin 21
Pilot Laser, one of DOut2, DOut3, DOut6, DOut7	Pin 7, 9, 15 or 17	Pin 22

APPENDIX B – Wiring between E1701P and SPI G4 Pulsed Fibre Laser Series

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

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

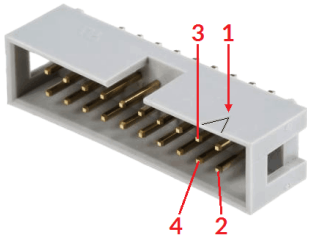
Signal Name	**Connector / Pin**	**SPI Laser Connector Pin**
LP0	Pin 1	Pin 17
LP1	Pin 3	Pin 18
LP2	Pin 5	Pin 19
LP3	Pin 7	Pin 20
LP4	Pin 9	Pin 51
LP5	Pin 11	Pin 52
LP6	Pin 13	Pin 53
LP7	Pin 15	Pin 54
MO / Laser Enable	Pin 8	Pin 7
LP8 Latch	Pin 17	Pin 23
LaserA / Pulse Trigger	Pin 22	Pin 47
Laser Gate / Modulation	26 pin connector, pin 13 or XY2/100 adapter cable, DB9 connector, pin 2	Pin 5
Alarm, one of DIn2, DIn3, DIn6, DIn7	Pin 8, 10, 16 or 18	Pin 9
Pilot Laser, one of DOut2, DOut3, DOut6, DOut7	Pin 7, 9, 15 or 17	Pin 6

Variant 2: waveform selection, simmer, power and extended parameter control via laser controller plug in/serial interface:

Signal Name	**Connector / Pin**	**SPI Laser Connector Pin**
MO / Laser Enable	Pin 8	Pin 7
LaserA / Pulse Trigger	Pin 22	Pin 47
Laser Gate / Modulation	26 pin connector, pin 13 or XY2/100 adapter cable, DB9 connector, pin 2	Pin 5
Alarm, one of DIn2, DIn3, DIn6, DIn7	Pin 8, 10, 16 or 18	Pin 9
Pilot Laser, one of DOut2, DOut3, DOut6, DOut7	Pin 7, 9, 15 or 17	Pin 6

APPENDIX C – 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:

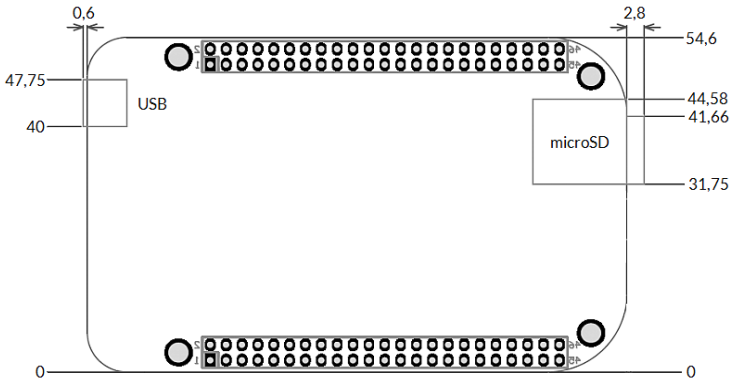


The first pin is marked by a small arrow in connector. Second pin is below of it, counting continues column-wise.

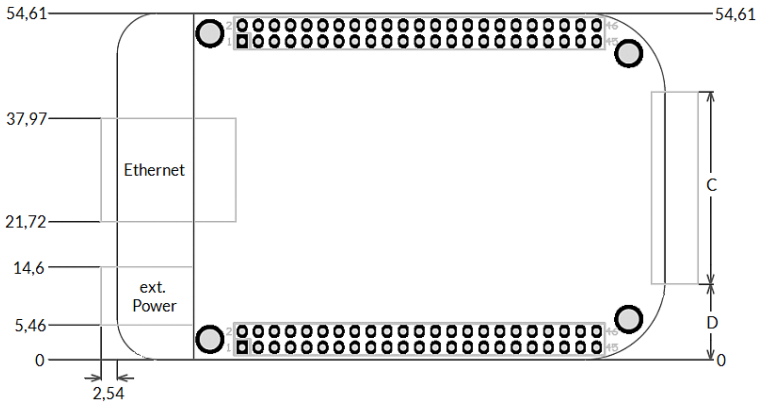
APPENDIX D – Board dimensions

Board dimension drawings, all values are given in unit mm.

Connectors, bottom view:

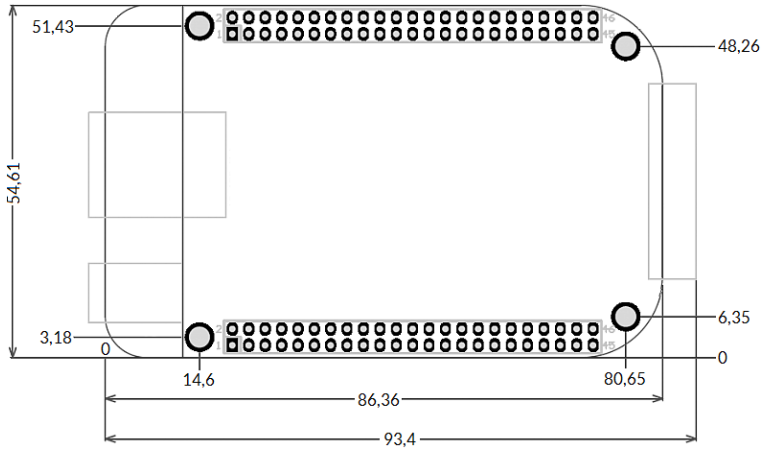


Connectors, top view:



Connector	C	D
Motor and Digital I/O	34 mm	10,3 mm
Laser signal	40 mm	7,3 mm

Dimensions, top view:



Alphabetical Index