Copyright © 2025 Optotune

ICC-1C

Industrial Current Controller Manual

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright ©2025 Optotune Switzerland AG. All Rights Reserved.

The ICC-1C controller's hardware and corresponding software, Optotune Cockpit, shall only be used in connection with the evaluation of the liquid lens product families. Any other use is not permitted without the prior written authorization of Optotune Switzerland AG.

IN NO EVENT SHALL OPTOTUNE BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS CURRENT CONTROLLER AND ITS DOCUMENTATION, EVEN IF OPTOTUNE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

OPTOTUNE SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE CURRENT CONTROLLER AND ACCOMPANYING DOCUMENTATION, IF ANY, PROVIDED HEREUNDER IS PROVIDED "AS IS". OPTOTUNE HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

Contents

1. Overview 5
2. Hardware overview 6
 2.1. Package contents and description 6
 2.2. Connecting to a computer or other system 8
 2.3. Connecting to a lens 10
 2.4. Mounting 11
 2.5. Thermal Management 14
 2.6. Do’s and Don’ts 14
3. Software operation 16
 3.1. Lens related systems (widgets) 17
 3.2. Controller related systems (widgets) 21
4. Software development kits (SDKs) 25
 4.1. C# SDK installation and run 25
 4.2. C# code example 25
 4.3. Python SDK installation 28
 4.4. Python code example 29
5. Firmware 30
 5.1. Firmware content description 30
 5.2. Firmware update 31
6. UART Communication Protocol 32
 6.1. Pro (binary) mode 33
 6.2. Pro mode example 35
7. Ethernet Communication 38
 7.1. IP settings configuration via DHCP 38
 7.2. Static IP address configuration 39
 7.3. Ethernet communication example via Cockpit or a Telnet Client 41
 7.4. Ethernet communication example via the Python SDK 42
8. I2C Communication 44
 8.1. I2C Configuration 44
 8.2. I2C communication protocol description 45
 8.3. I2C Write command structure 46
 8.4. I2C Read command structure 47
 8.5. Changing the I2C register count 48
 8.6. I2C Communication example (Raspberry Pi) 49
 8.7. I2C Communication example (Arduino) 50

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

9. Analog Input Voltage mode 52
10. Input/Output trigger signals 54
11. Troubleshooting and FAQs 57
11.1. Status LEDs 57
11.2. Diagnostics in Optotune Cockpit 58
11.3. Diagnostics with the Device Manager and Terminal software 58
11.4. FAQs 61

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

1. Overview

The ICC-1C Industrial Controller enables seamless control of Optotune's tunable liquid lenses with a wide variety of control interfaces directly from a computer or from an electronics cabinet, offering a streamlined solution for rapid deployment in optical applications.

Key Features:

- **High-Precision Current Control**: Adjustable current range from -500 mA to +500 mA with fine 65 µA current step resolution
- **Communication Interfaces:**
  - **USB**
  - **UART** or **I2C** with automatic protocol detection at device startup
  - **Ethernet** with POE+ capability
  - **Analog Voltage Input Control** in range from 0 to 10 V
  - **GPIO Trigger** for event-based control
- **Advanced Lens Calibration and Compensation**: Enables readout of stored calibration data and real-time temperature from supported lenses for dynamic adjustment in Focal Power Mode
- **Smart Step Feature** to enable faster focal power settling time of the lens by applying a preconditioned coil current waveform
- **User-Friendly Control Options**:
  - Compatible with the **Optotune Cockpit GUI** for USB based control
- **Various Output Connections and Debug Features**:
  - **Hirose Connector** for fast and robust industrial connection
  - **Auxiliary Connectors** to connect an Extension Board providing an option to connect a liquid lens via an FPC flex cable, while also providing various debugging options (external current measurement, triggering, analog voltage control)
- **Developer Support**: Software Development Kits (SDKs) are available for **Python** and **C#**
- **Compliance**: Fully certified with **RoHS**, **REACH** and **CE** standards

The detailed technical specifications can be found in ICC-1C's datasheet.

To access additional resources, like firmware updates, the Optotune Cockpit software or CAD files, please visit Optotune's Download Center (a free registration is required).

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

2. Hardware overview

2.1. Package contents and description

The ICC-1C can be ordered in three different package configurations:

ICC-1C

Figure 1: ICC-1C Controller

Description	Quantity
ICC-1C	1
DIN rail adapter	1

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

As part of the **ICC-1C Controller Kit** (ICC-1C, DIN rail adapter, ICC-1C Extension Board, Power supply, USB-C cable)

Figure 2: ICC-1C Controller Kit

Description	Quantity
ICC-1C	1
DIN rail adapter	1
ICC-1C Extension Board for lenses with FPC flex cable	1
AC/DC Power supply to 24V/1A	1
USB-A to USB-C cable	1

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

ICC-1C PCBA (OEM version)

Figure 3: PCB only ICC-1C Controller

Description	Quantity
ICC-1C Assembled PCB	1

2.2. Connecting to a computer or other system

The ICC-1C provides the following connectivity options:

A **USB-C connector** with an option to use as a power supply to the ICC-1C as well

Please note that while most USB devices can supply sufficient power to the controller driving any of Optotune's lenses under typical conditions (< 500mA), in some cases higher current spikes might occur, so make sure to use a port rated according to the USB-C standard (> 3A power delivery) or use the dedicated barrel supply connector.

An **RJ-45 Ethernet connector** with PoE capability (802.3af or higher)

The **Auxiliary I/O connector** can provide additional communication and power supply options

To use the ICC-1C with **UART** communication, connect the *UART TX* (pin 5), *UART RX* (pin 7) and *GND* (pins 3 and 4) to the system that is controlling the device.

For **I2C** communication, the device uses the same pins as for the UART, with the *I2C SCL* signal being on pin 5, the *I2C SDA* on pin 7 and the *GND* (pins 3 and 4).

This connector can also provide an option of an using an **External power supply**. To use this feature, connect the *External VCC* (pin 13) and *External GND* (pin 14) to the supply rail of the external device.

No representation or warranty, either expressed or implied, is made
as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

To decrease the current consumption of the system, there is also an Active Low Enable signal (*External VCC Enable* – pin 1) on the connector. Pulling this signal down to the *External GND* rail will turn on the ICC-1C and releasing it will disable controller, so that it will draw no power from the system.

The ICC-1C provides an option to map an **Analog Input Voltage** to a specific lens currents or focal powers and thus enables controlling a lens by a value from an external sensor or other analog voltage. This feature is accessible by connecting the external analog signal to the *Analog In* (pin 2) and *GND* (pins 3 and 4) of the auxiliary I/O connector

The controller also has two configurable **GPIO** pins (pin 6 and 8), which can be configured to generate a trigger signal synchronized with the internal signal generator or to wait for an external trigger signal before changing the current or focal power of the lens. Please note, that these pins were meant to be used with 3.3V logic and as such, the related min/max voltages and threshold levels shall be respected.

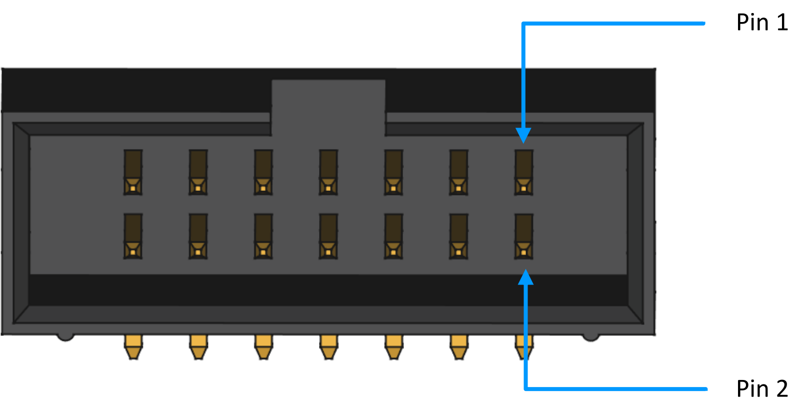


Figure 4: Auxiliary I/O connector pinout

Auxiliary I/O connector pinout		
Position	**Function**	**Description**
1	External VCC Enable	Enable signal for external power supply (connect to Power GND to activate)
2	Analog In	Analog Input Voltage
3	Signal GND	Digital and analog ground
4	Signal GND	Digital and analog ground
5	UART TX/ I2C SCL	Serial interface transmitter line / I2C clock line[1]
6	GPIO1	General purpose digital IO #1, Trigger Input/Output[2]
7	UART RX/ I2C SDA	Serial interface receiver line / I2C data line[1]
8	GPIO0	General purpose digital IO #0, Trigger Input/Output[2]
9	-	Reserved
10	-	Reserved
11	-	Reserved
12	-	Reserved
13	External VCC	External power supply input
14	External GND	External power supply input - ground
		[1] *configurable external serial interface for UART or I2C* [2] *configurable input/output*

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

2.3. Connecting to a lens

The ICC-1C can be connected to a lens by using one of the following two options:

By using an **industrial (Hirose) connector** the lens can be simply and safely connected by a Hirose cable to the plug on the front panel of the ICC-1C, until a "click" sound is heard. The advantage of this connector is that the positions of the pins are unambiguous, so it is safe to connect it even by hot plugging during the operation of the controller.

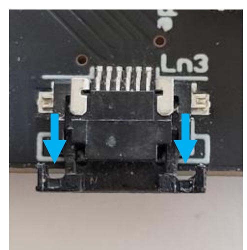
Figure 5: Connecting to a lens with Hirose connector

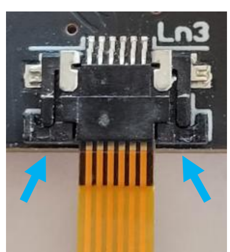
With **an FPC flex cable and the ICC-1C Extension Board**, by connecting the extension board to the two Molex connectors on the front panel of the ICC-1C and then connecting the lens to the FPC connector on the extension board. This option provides the ability for the users to do some measurements via the extension board, but they should be more careful, as the FPC cable is not suitable for hot plugging (it can be inserted under an angle or even upside down), and an incorrect connection can cause damage to the lens and/or the controller. To use the FPC cable connection safely, the power supply always should be removed from the controller first, before manipulating with the flex cable.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

Figure 6: Connecting to a lens with FPC connector





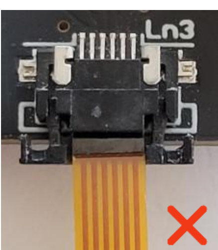


Figure 7: Proper connection to a lens with a flex cable and a wrong connection example with flipped flex cable and not enclosed connector

2.4. Mounting

The ICC-1C is available in a compact metal casing that can be mounted on a standard DIN rail (a suitable adapter is already included in the package with the controller). For more mounting details and dimensions, see the mechanical drawing below:

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

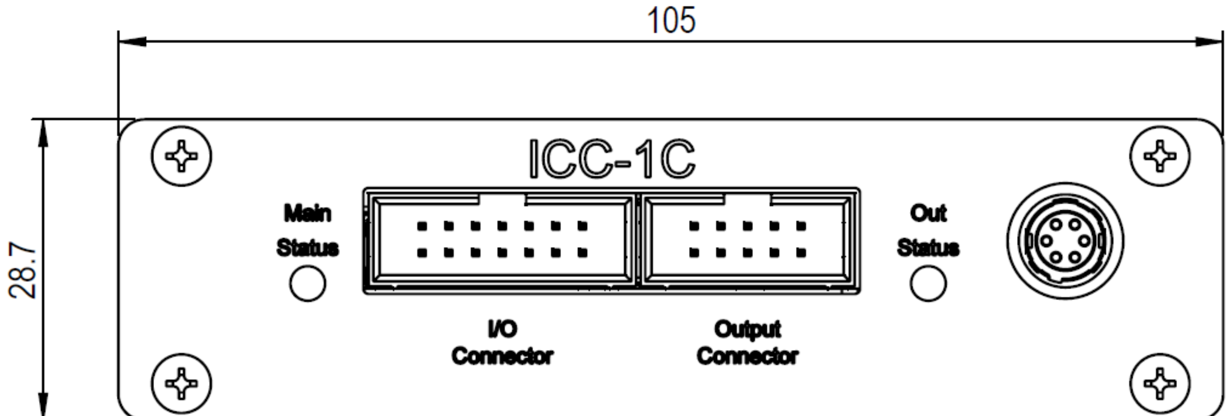


Figure 8: ICC-1C casing front view

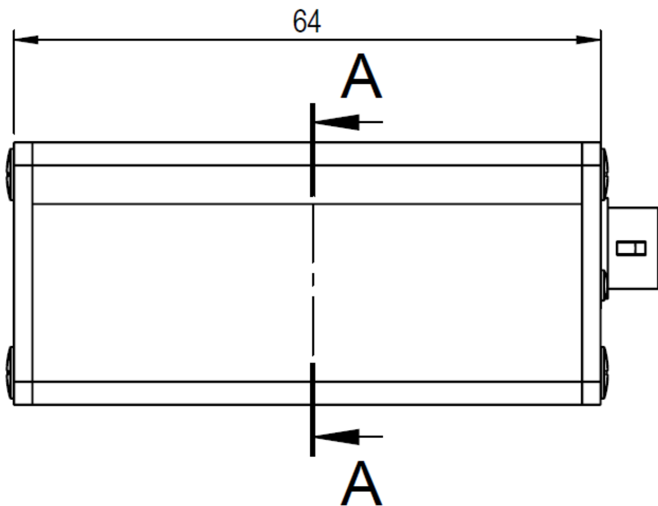


Figure 9: ICC-1C casing side view

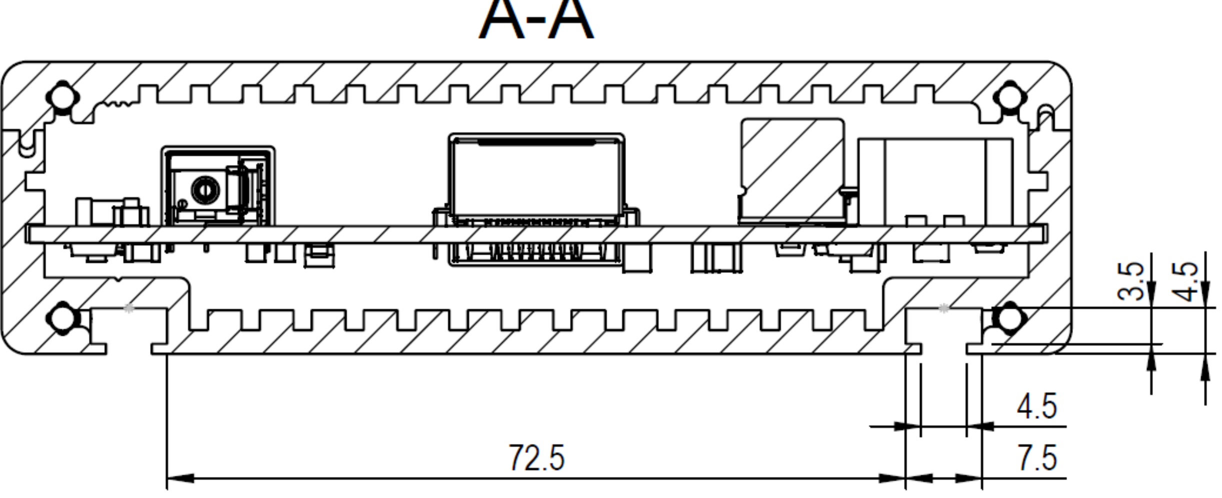


Figure 10: ICC-1C casing cross section A-A with the dimensions for the DIN rail adapter

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

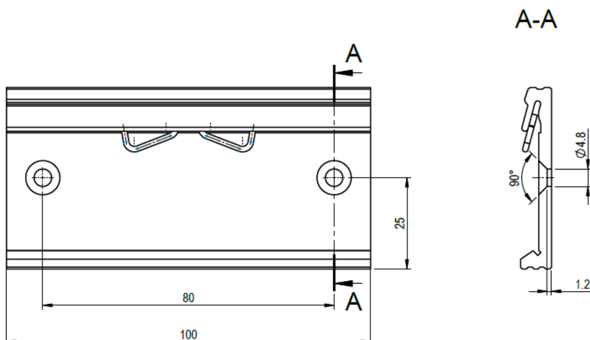


Figure 11: DIN rail adapter for ICC-1C

Alternatively, a PCB-only (OEM) version is also available, in which case the ICC-1C can be mounted by standard M3 screws or standoffs inside the casing of a larger device. For more mounting details and dimensions, see the mechanical drawing below:

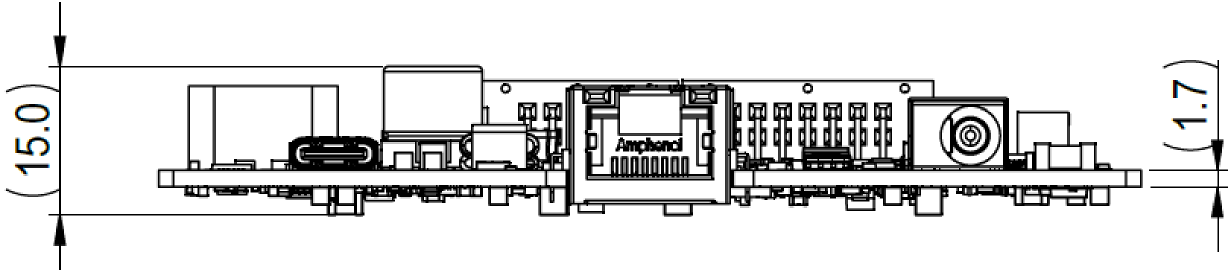


Figure 12: ICC-1C PCBA back view with the dimensions for the heights of the components

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

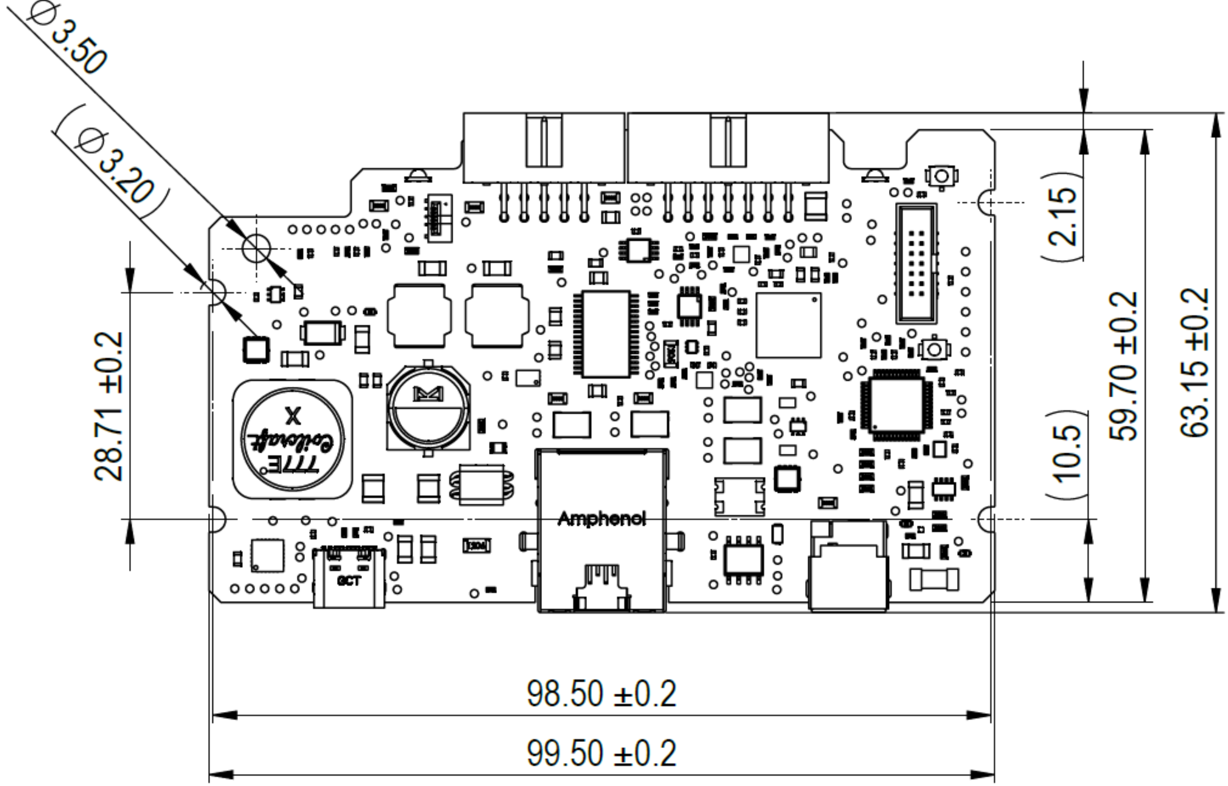


Figure 13: ICC-1C PCBA top view

2.5. Thermal Management

The ICC-1C has a built-in thermal monitoring for both the connected lens and the controller itself, to prevent overheating and potential damage. The temperature limits for both the controller and the connected lens can be programmed in the Optotune Cockpit, or by a custom software or device sending the appropriate command to the controller.

During the placement and mounting of the controller it should be ensured that it's not placed right next to a significant heat source, so that the ambient temperature around the controller will remain in the range defined in the datasheet (max. 65 °C).

2.6. Do's and Don'ts

When using the Extension Board, do not connect or disconnect the FPC flex cable, while the device is powered on. During these manipulations the adjacent pins on the cable can be temporarily shorted and if the device is powered on at the same time, this may lead to EEPROM data corruption and/or permanent damage to the controller or the lens.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

This product must be powered by a certified external SELV power supply with CE marking, compliant with Directive 2014/35/EU (Low Voltage Directive).

This controller and its extensions (for example the Extension Board) can be damaged by ESD. We recommend the device to be handled with appropriate precautions and protections. Failure to observe proper handling and installation procedures might cause damage. ESD damage can range from subtle performance degradation to complete device failure.

Do not use aggressive agents such as acetone or acids to clean the ICC-1C, the Extension Board or any of the other components included with the device.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

3. Software operation

The Optotune Cockpit GUI allows users to establish connections with the controller and the connected lens, monitor device statuses, and perform firmware updates. The menu on the left side contains 3 main sections:

The **Electronics** section is showing the type and the serial number of the connected controllers. Clicking on the "+" button initiates a search for any compatible controller on the available interfaces.

The **Products** section is showing the type and the serial number of the lens connected to the selected controller.

The **Systems** section contains a list of available features for the given controller and lens, and each of them opens a standalone widget in the Dashboard section (right side) containing all the setting related to that feature.

The application supports multiple customizable **Dashboards**, allowing users to rearrange and resize widgets to suit their preferences and workflow.

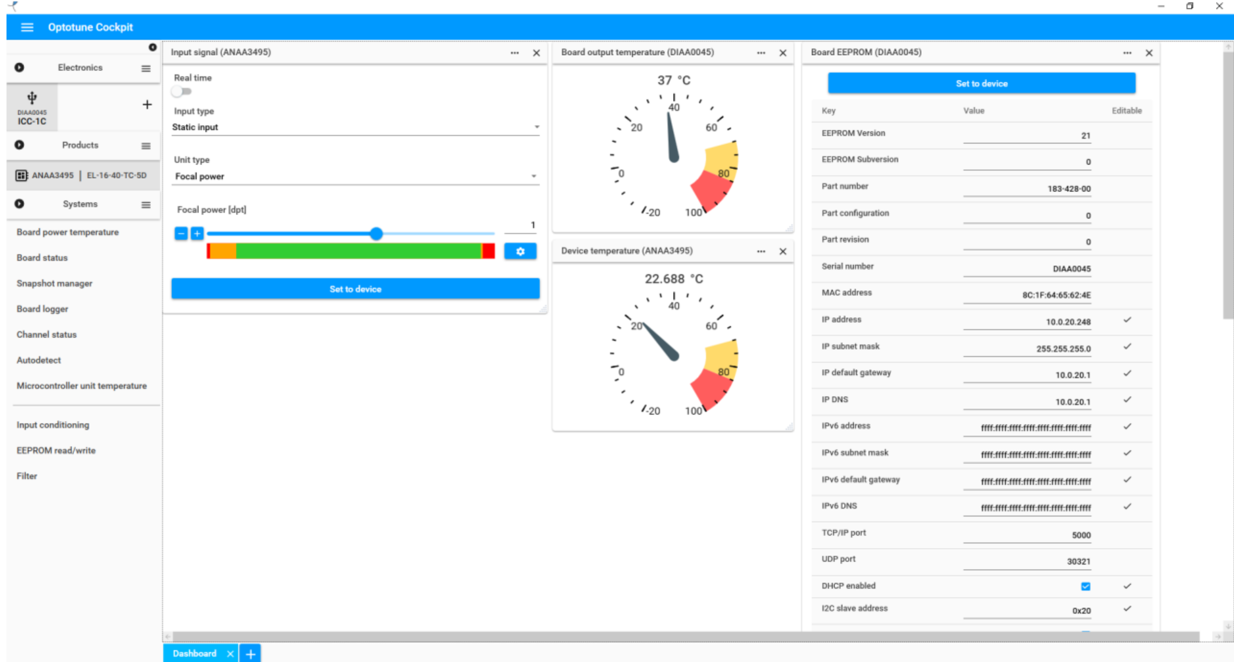


Figure 14: Optotune Cockpit main window overview

The available systems/widgets in the software can be divided into two groups based on whether they contain setting related to the controller or the connected lens.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

3.1. Lens related systems (widgets)

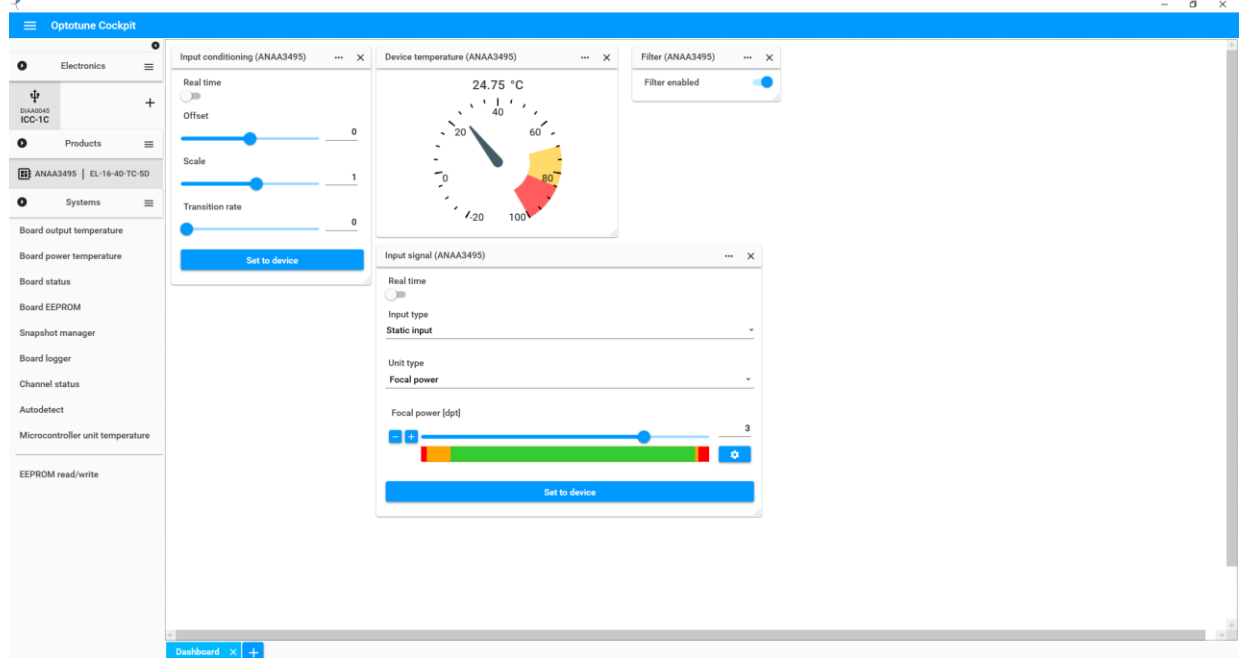


Figure 15: Examples of widgets related to the lens

The widgets related to the lens allow the user to access settings and monitoring features for the various lenses, such as:

Temperature readout and settings for the thermal shutdown temperature. This feature is only available for lenses with a built-in temperature sensor.

Smart Step Filter feature to achieve better settling times during transitions.

EEPROM read/write widget, which allows to read out the serial number, calibration data and other information from the connected lens. It can also allow restoring the EEPROM data in case of issues or data corruption. This feature is only available for lenses with a built-in EEPROM chip.

Input conditioning widget to apply an offset, scaling or transition speed limit to any input value defined in the Input signal widget.

Input signal widget, which allows the user to select the controlled value (current or focal power) as well as multiple options on how the given value should be controlled:

Static input value to maintain a certain output current or focal power value.

A **Signal generator**, where the user can set the output current or focal power to follow a predefined waveform shape from the controller's built-in signal generator (Sinusoidal, Triangular, Square, Sawtooth, Pulse, Staircase or Fast autofocus waveforms are available).

Custom vector, where the user can define multiple output current or focal power values with their respective timings for the lens to go through.

Analog input, where the user can map analog input voltage values (in the range of 0 – 10V) provided via the *Analog In* pin to a specific output currents or focal powers.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

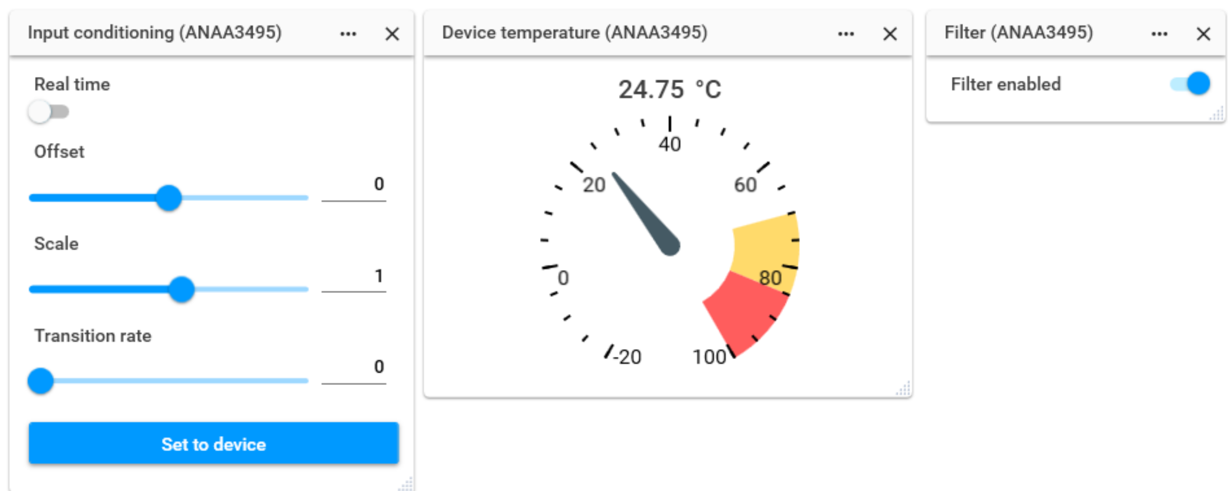
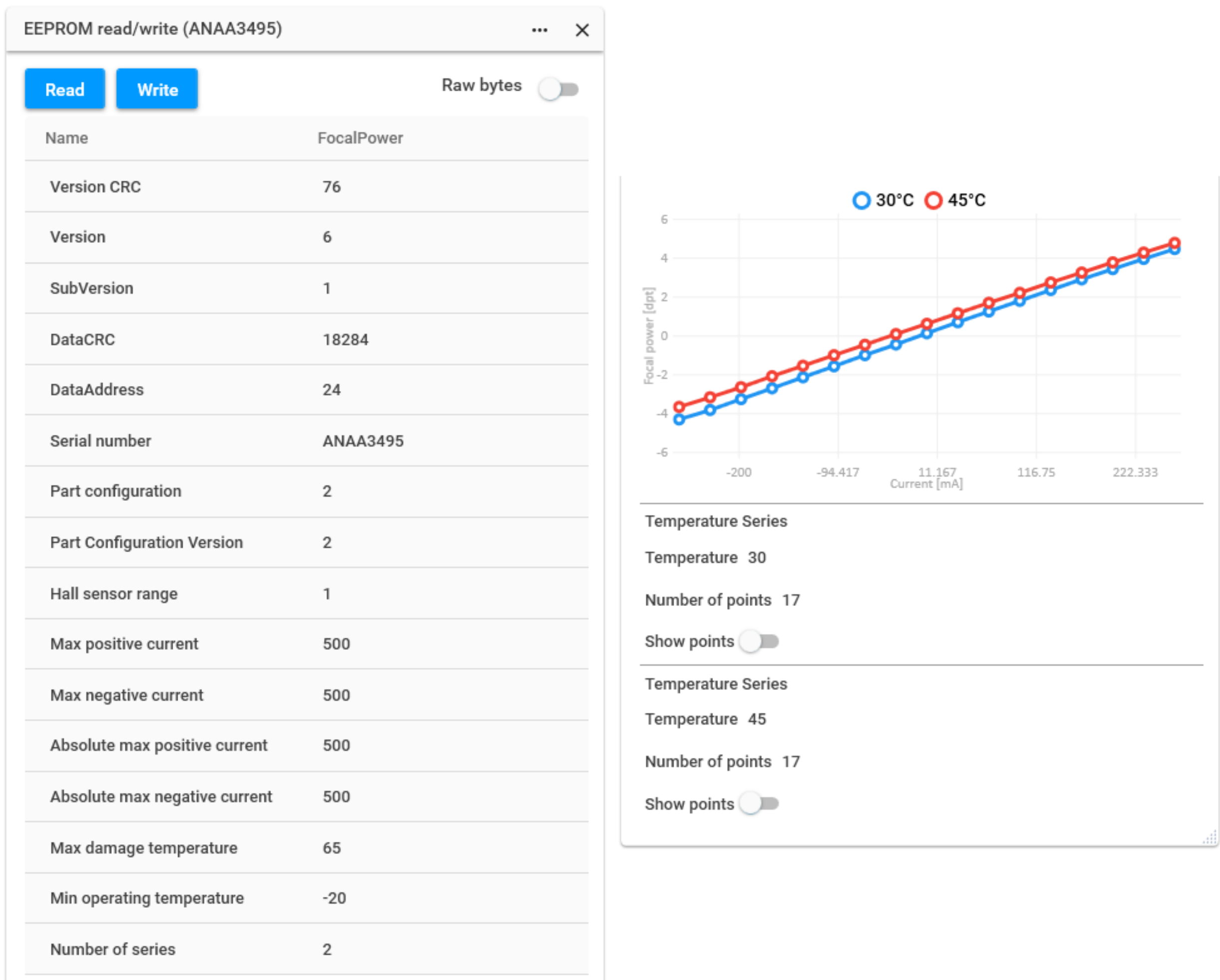


Figure 16: Input conditioning, Lens temperature readout and Smart Step Filter settings widgets



Name	FocalPower
Version CRC	76
Version	6
SubVersion	1
DataCRC	18284
DataAddress	24
Serial number	ANAA3495
Part configuration	2
Part Configuration Version	2
Hall sensor range	1
Max positive current	500
Max negative current	500
Absolute max positive current	500
Absolute max negative current	500
Max damage temperature	65
Min operating temperature	-20
Number of series	2

Figure 17: Lens EEPROM read/write widget

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

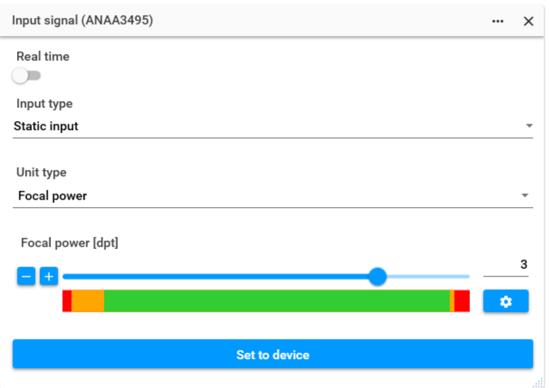


Figure 18: Input signal widget set to Static input

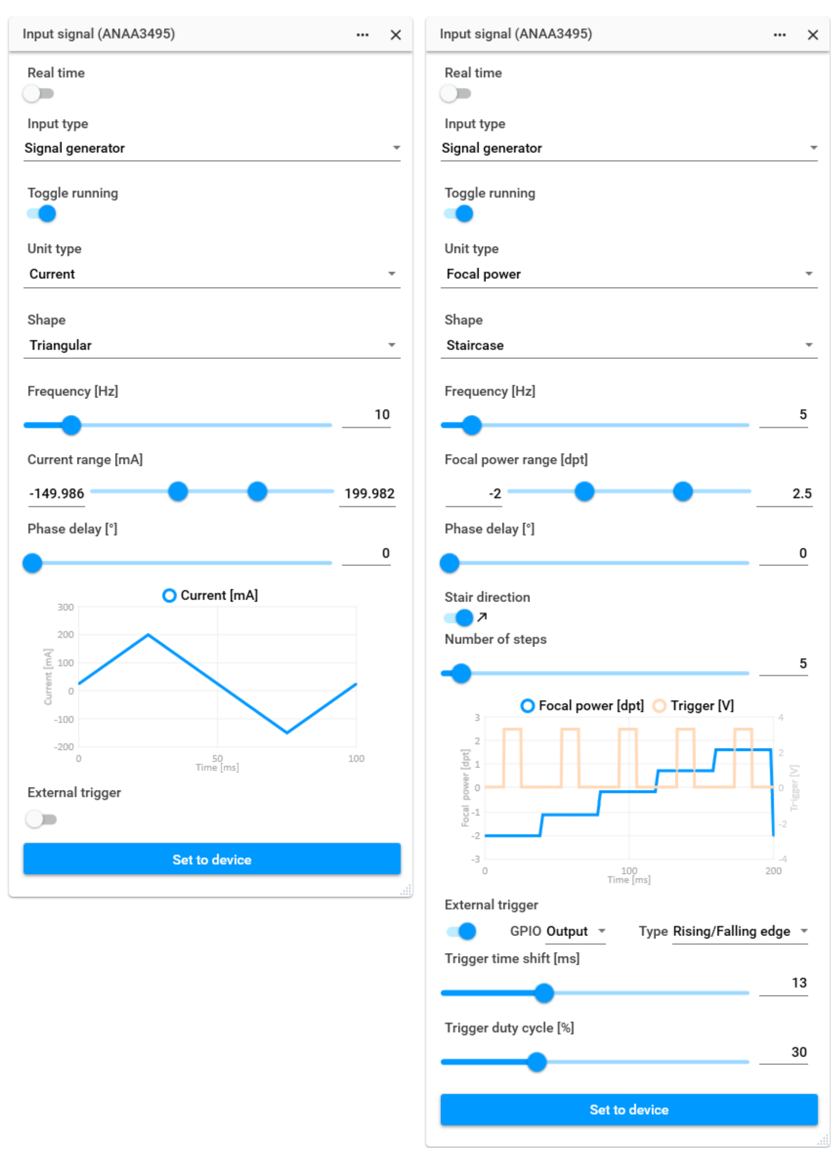


Figure 19: Input signal widget set to various waveforms from the Signal generator

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

Input signal (ANAA3495)

Real time

Input type

Custom vector

Toggle running

Unit type

Focal power

External trigger

Easy/Pro mode

Programmable points

Focal power [dpt]

Time [ms]

Index	Focal power [dpt]	Duration [ms]
0	0	30
1	0.5	40
2	1	50
3	1.5	60

+

Rows per page 5 0-3 / 4

Set to device

Input signal (ANAA3495)

Real time

Input type

Analog input

Analog IN [V] 0.009

Unit type

Focal power

Non-linear analog transition

Analog transition

Focal power [dpt]

Voltage [V]

Constant/Linear extrapolation

Index	Voltage [V]	Focal power [dpt]
0	0	-3.5
1	0.5	-2
2	1	-1
3	1.5	-0.5
4	2	0
5	2.5	0.5
6	3	1
7	3.5	2
8	4	3.5
9	10	5

+

Rows per page 20 0-9 / 10

Set to device

Figure 20: Input signal widget set to Custom vector and Analog input

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

3.2. Controller related systems (widgets)

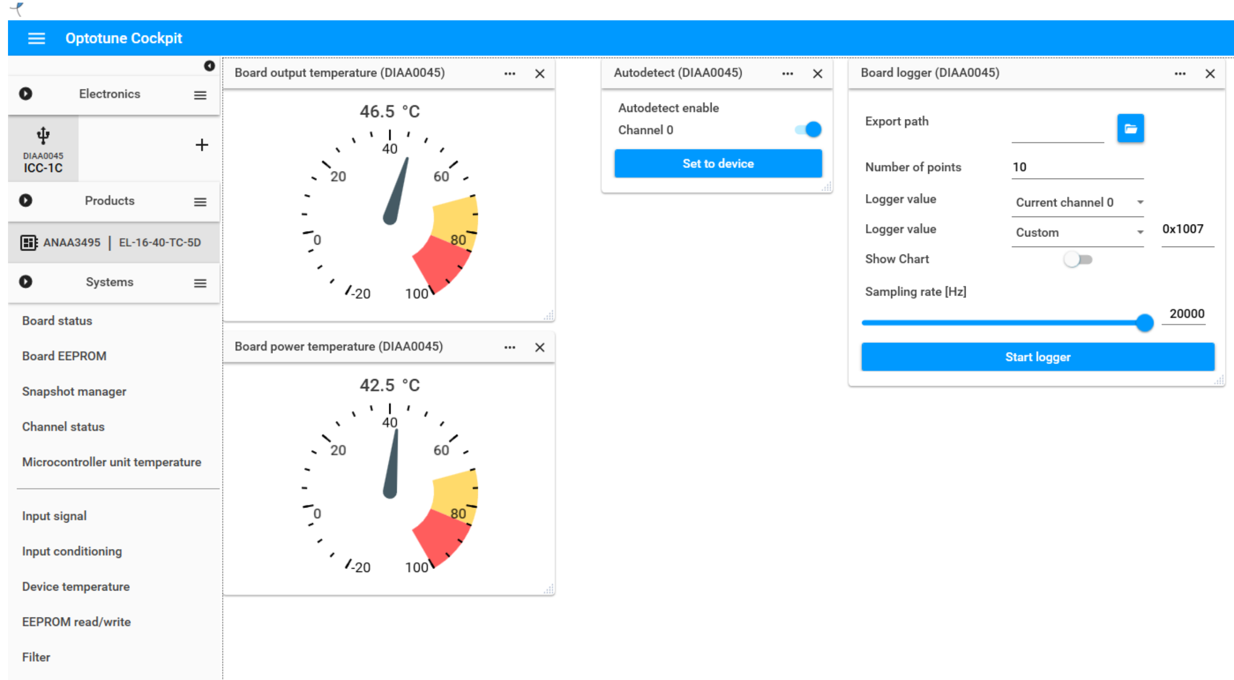


Figure 21: Examples of widgets related to the controller

The widgets related to the controller allow the user to access various settings and monitoring features of the ICC-1C, such as:

Temperature readouts for various subsystems (the input power supply, the output current stage and the MCU) and settings for the thermal shutdown temperatures for these subsystems.

Board status, Channel status and **Autodetect** features to monitor and control values related to the driver, such as the Frontend voltage, which can improve the accuracy of the current control for different sized lens or the Autodetect feature, which allows the user to turn on/off the automatic detection of the lens connected to the controller.

Board EEPROM widget, which contains the information stored in the ICC-1Cs memory, such as the device's serial number or the settings required for Ethernet and I2C communication.

Board logger widget, which allows monitoring and exporting values like output current, focal power and even register values into a .csv file.

Snapshot manager widget, which can store two complete sets of settings for both the controller and the lens. Slot number 1 is read-only and contains the factory defaults, while slot number 2 can by configured and saved by the user.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

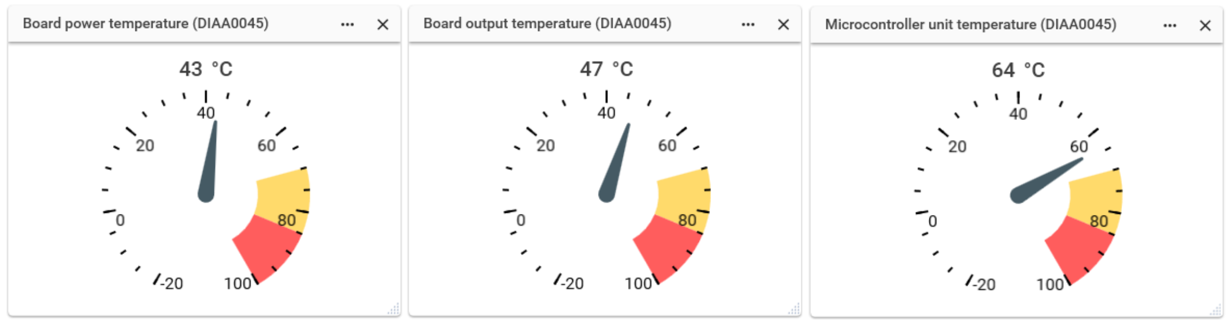


Figure 22: Temperature readout widgets

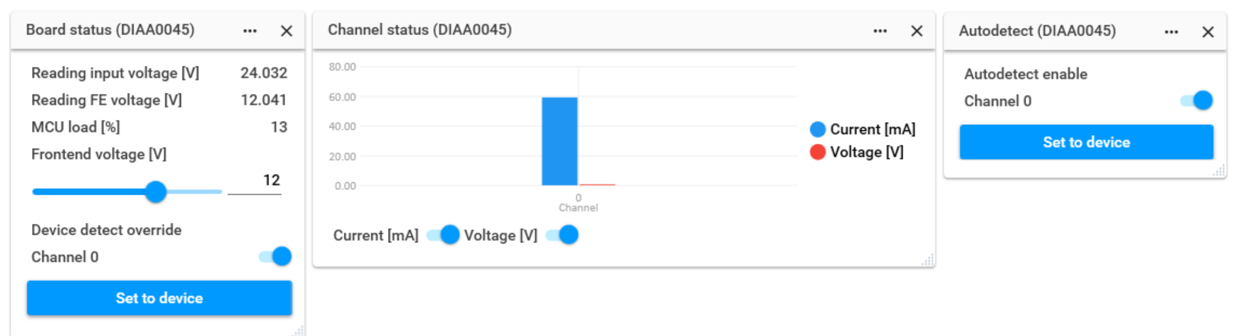


Figure 23: Board status, Channel status and Autodetect widgets

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

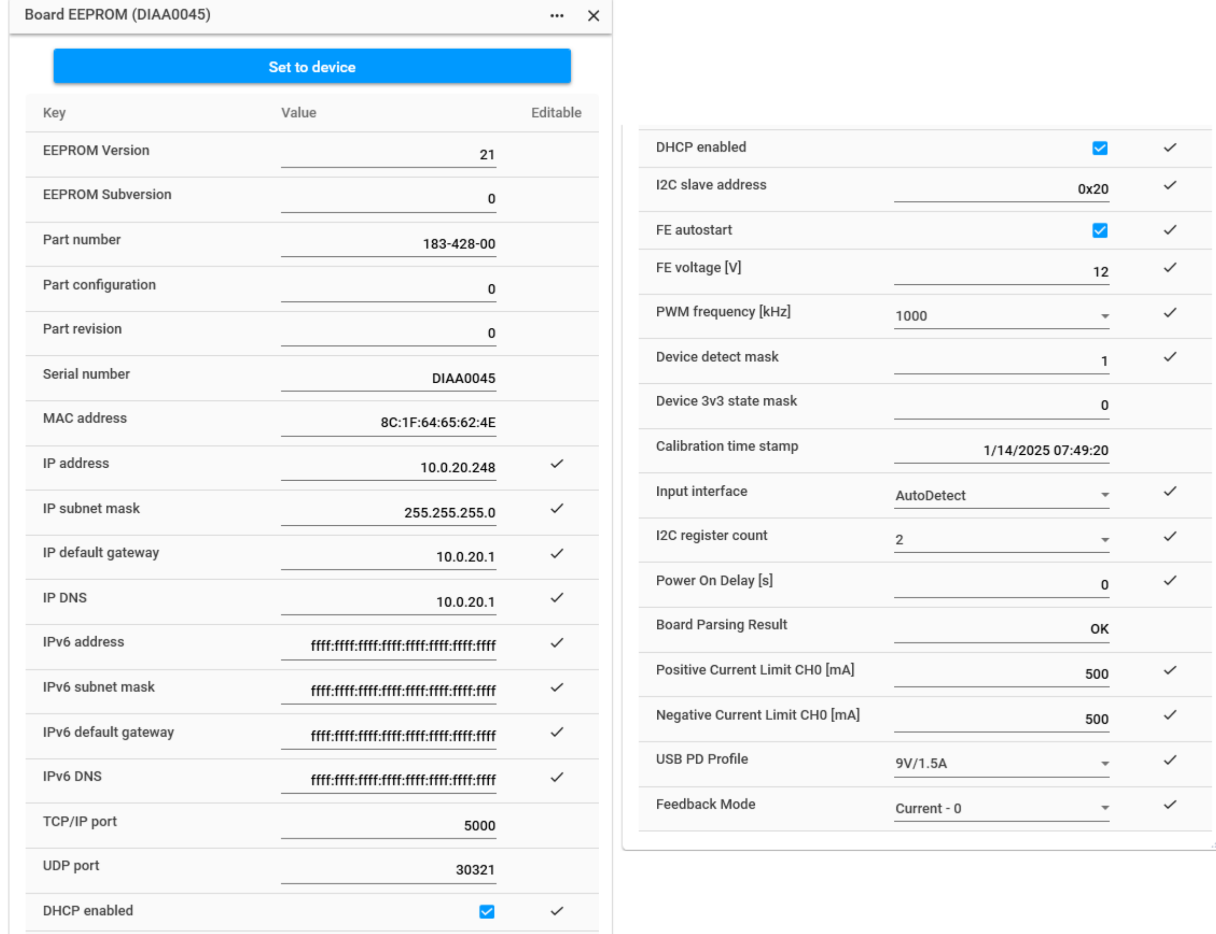


Figure 24: Board EEPROM widget

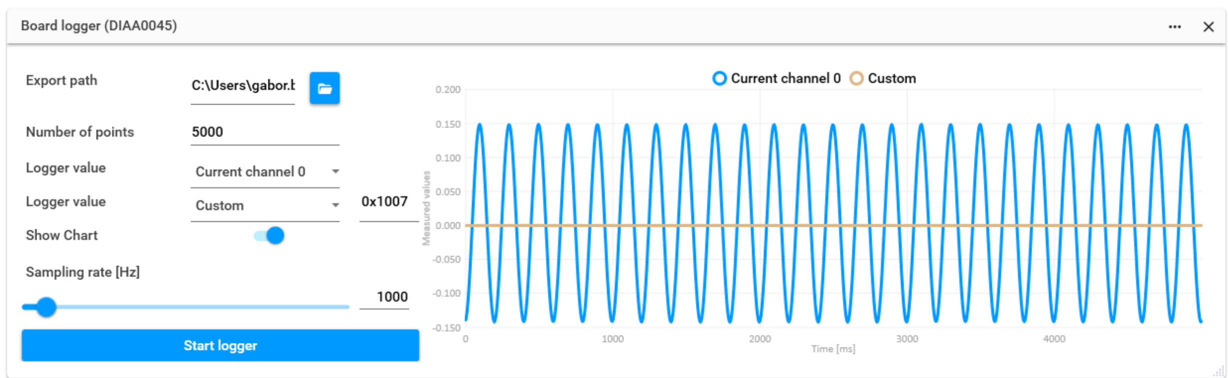


Figure 25: Board logger widget

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

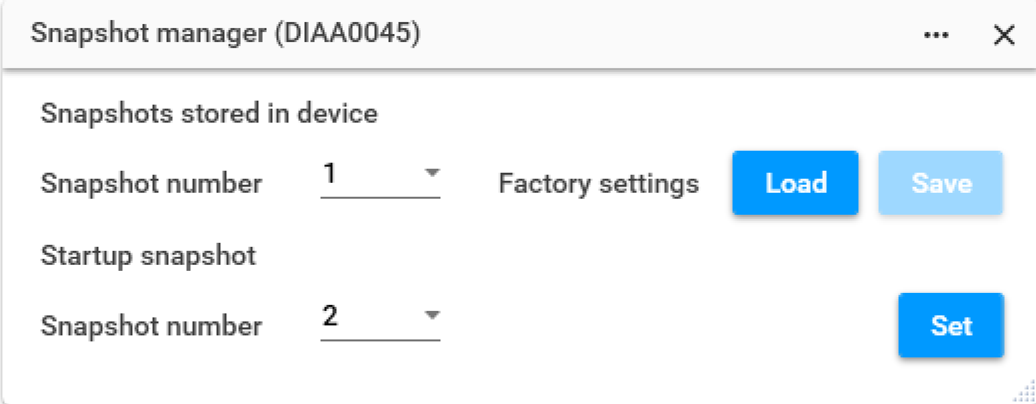


Figure 26: Snapshot manager widget

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

4. Software development kits (SDKs)

There are two different software development kits available for the ICC-1C – one for C# and one for Python. Both are available on the Download Center website (a free registration is required). They contain the necessary libraries to work with the ICC-1C as well as some example codes.

4.1. C# SDK installation and run

To install and run the C# SDK examples, the following steps should be performed:

1. Download the ICC-4C/ECC-1C/ICC-1C C# SDK from Optotune's website.
2. Extract the .zip file.
3. This C# SDK requires to have Visual Studio and the .NET framework 4.6.1/4.6.2 installed, so please make sure you have them on your PC.
4. Open the *Icc4cExample.sln* solution.
5. Build all and Run the appropriate project.
6. To create a new solution, in the downloaded C# SDK there are available .dll files, which could be added into any project's references and directly control Optotune's products.

Additional information about the C# SDK can be found in the detailed SDK documentation (included in the *\html* subfolder of the downloaded SDK – open the *index.html* file).

4.2. C# code example

```
using System;
using System.Threading;
using Icc4cExample;
using ICC4cPwmSdk.Device;
using ICC4cPwmSdk.Dtos;

namespace Icc1cExample
{
    internal class Icc1cExample
    {
        // To run this example, rename AnaInMain to Main and in Example.cs rename Main to
ExMain for example
        // static void Main(string[] args) -> is always an entry point
        static void Main(string[] args)
        {
            Console.WriteLine("Starting basic ICC-1C example...");
            Icc1cSdkDeviceController controller;

            while (true)
            {
                Console.WriteLine("Select connection: C = COM, E = ETHERNET");
                var key = Console.ReadKey();
                Console.WriteLine();
                //COM connection
                if (key.Key == ConsoleKey.C)
                {
                    controller = new Icc1cSdkDeviceController();
                    Console.WriteLine("What is the COM number of the ICC-1c?");
                    var comport = Console.ReadLine();
                    if (int.TryParse(comport, out var comInt))
                    {
                        if (controller.Connect(comInt))
                        {
```


No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

```
                            break;
                        }

                        Console.WriteLine("Connection failed.");
                    }
                }

                // Ethernet connection
                if (key.Key == ConsoleKey.E)
                {

                    controller = new Icc1cSdkDeviceController(true);
                    Console.WriteLine("Static IP or automatic search? A = auto, S = static");
                    key = Console.ReadKey();
                    Console.WriteLine();

                    //Automatic ethernet search (DHCP has to be enabled in the board EEPROM
(default))
                    if (key.Key == ConsoleKey.A)
                    {
                        var devices = EthernetSearcher.SearchEthernetIccBoards();
                        if (devices.Count == 0)
                        {
                            Console.WriteLine("No ethernet boards found.");
                        }
                        else
                        {
                            Console.WriteLine($"Boards found({devices.Count}):");

                            var index = 0;
                            foreach (var device in devices)
                            {
                                Console.WriteLine($"Index: {index}, Serial number: {device.Se-
rialNumber}, IP: {device.Ip}, Port: {device.Port}");
                                index++;
                            }
                            Console.WriteLine("Select index of the device you want to con-
nect.");
                            var selectedIndex = Console.ReadLine();
                            if (int.TryParse(selectedIndex, out var indexInt) && indexInt >= 0
&& indexInt < devices.Count)
                            {
                                if (controller.Connect(devices[indexInt].Ip, devices[in-
dexInt].Port))
                                {
                                    break;
                                }
                                Console.WriteLine("Connection failed.");
                            }
                        }

                    }

                    //Static IP connection (static IP must be configured in the board EEPROM
(using Optotune Cockpit application), also DHCP has to be switched of in the board EEPROM)
                    if (key.Key == ConsoleKey.S)
                    {
                        Console.WriteLine("Specify IP address of the board:");
                        var ipAddress = Console.ReadLine();

                        if (controller.Connect(ipAddress, 5000))
                        {
                            break;
                        }
                        Console.WriteLine("Connection failed.");
                     }
                }
            }

            //Set board to PRO mode
            controller.ChangeModeToPro();

            //Getting board info
```


No representation or warranty, either expressed or implied, is made
as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

```
            Console.WriteLine("Board info");
            var serialNumber = controller.GetElectronicSerialNumber();
            var fwVersion =
                $"{controller.Status.GetBoardFirmwareMajorVersion()}.{controller.Status.Get-
BoardFirmwareMinorVersion()}.{controller.Status.GetBoardFirmwareRevisionVersion()}";

            Console.WriteLine($"Board serial number: {serialNumber}");
            Console.WriteLine($"Board firmware version: {fwVersion}");

            // In order to drive the connected product, device type of the product has to be
obtained (this is not needed if
            // automatic detection is allowed)
            var connectedDevice = controller.GetDeviceType();
            Console.WriteLine();
            Console.WriteLine(
                $"Connected device: {string.Join(Enum.GetName(typeof(EConnectedDeviceType),
connectedDevice))}");
            Console.WriteLine();

            var lensSerialNumber = controller.GetElectronicSerialNumber();
            Console.WriteLine(
                $"Lens {Enum.GetName(typeof(EConnectedDeviceType), connectedDevice)} ({lensSe-
rialNumber}) found");

            var lensTemperature = controller.TemperatureManager.GetDeviceTemperature();

            Console.WriteLine($"Lens temperature: {lensTemperature}°C");

            var minCurrent = controller.DeviceEeprom.GetMaxNegativeCurrent();
            var maxCurrent = controller.DeviceEeprom.GetMaxPositiveCurrent();

            Console.WriteLine($"Minimum current is {minCurrent} mA, maximum current is {maxCur-
rent} mA");

            Console.WriteLine();
            Console.WriteLine("Setting static current");
            //Setting input system to static input
            controller.InputStage.ChangeActiveSystem(EInputSignalStageSystem.StaticInput);

            for (var current = minCurrent; current <= maxCurrent; current += (maxCurrent - min-
Current) / 5)
            {
                //Value has to be converted from mA to A
                var currentInA = current / 1000;
                Console.WriteLine($"Current {currentInA} A");
                controller.InputStage.StaticValue.SetCurrent(currentInA);
                //Diopter can be set similarly
                //controller.InputStage.StaticValue.SetDiopter(lensChannel, diopterInDpt);
                Thread.Sleep(1000);
            }

            Console.WriteLine("Setting static current to 0 A");
            controller.InputStage.StaticValue.SetCurrent(0);

            Console.WriteLine();
            Console.WriteLine("Running signal generator");

            controller.InputStage.ChangeActiveSystem(EInputSignalStageSystem.SignalGenerator);
            controller.InputStage.SignalGenerator.SetUnitType(EICC4cPwmUnitType.Current);
            controller.InputStage.SignalGenerator.SetShape(ESignalGeneratorShape.Sinusoidal);
            controller.InputStage.SignalGenerator.SetAmplitude(0.2f);
            controller.InputStage.SignalGenerator.SetFrequency(5);
            controller.InputStage.SignalGenerator.SwitchRunning(true);

            for (var i = 0; i < 5; i++)
            {
                Console.Write(".");
                Thread.Sleep(1000);
            }

            controller.InputStage.SignalGenerator.SwitchRunning(false);

            Console.WriteLine();
```


No representation or warranty, either expressed or implied, is made
as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

```
            Console.WriteLine("Signal generator stopped");

            Console.WriteLine();
            Console.WriteLine("Device EEPROM");

            var eepromVersion =
                $"{controller.DeviceEeprom.GetEepromVersion()}.{controller.De-
viceEeprom.GetEepromSubVersion()}";
            Console.WriteLine($"EEPROM version: {eepromVersion}");

            var eepromBytes = controller.DeviceEeprom.GetBytes(0, 10);
            var eepromSize = controller.DeviceEeprom.GetDeviceEepromSize();

            Console.WriteLine(
                $"Printing {eepromBytes.Length}/{eepromSize} bytes saved in EEPROM:
{string.Join(", ", eepromBytes)}");

            Console.WriteLine();
            Console.WriteLine("Example finished.");
            Console.ReadLine();
        }
    }
}
```

4.3. Python SDK installation

To install the Python SDK for the ICC-1C, the following steps should be performed:

1. Download appropriate Python SDK from Optotune's website.
2. Extract the .zip file.
3. The Python SDK is installed through *pip*, so make sure you have it installed (usually it's already preinstalled in most Python environments).
4. Open the Command-line interface on your PC or your Python environment and run the following two commands (administrator privileges might be required):
   a. py -m pip install Download_location\ICC-4C_PythonSDK_version\optoKummenberg-version-py3-none-any.whl
   b. py -m pip install Download_location\ICC-4C_PythonSDK_version\optoICC-version-py3-none-any.whl
5. If the installation was successful, the following folders should show up in the *\Lib\site-packages* subfolder of your Python environment:
   - optoICC
   - optoICC-2.0.4922.dist-info
   - optoKummenberg
   - optoKummenberg-1.0.4894.dist-info

* Download_location and version – replace these parts in the commands with the file location and version of the download SDK.

* py -m – might not be required, if the command is run from certain Python environments.

Additional information about the Python SDK can be found in the detailed SDK documentation (included in the *\docs* subfolder of the downloaded SDK – open the *index.html* file).

No representation or warranty, either expressed or implied, is made
as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

4.4. Python code example

```
from time import sleep
from optoKummenberg import UnitType
from optoICC import connectIcc1c, WaveformShape

# Connecting to board.
# Port can be specified like connect(port='COM12')
# Ethernet can be specified like (ip_address='10.0.88.25')
icc1c = connectIcc1c()

#Getting board info
serial_number = icc1c.EEPROM.GetSerialNumber().decode('UTF-8')
fw_version = f"{icc1c.Status.GetFirmwareVersionMajor()}.{icc1c.Status.Get-
FirmwareVersionMinor()}.{icc1c.Status.GetFirmwareVersionRevision()}"

# Getting the lens info
connected_lens = icc1c.MiscFeatures.GetDeviceType(0)
lens_serial_number = icc1c.channel[0].DeviceEEPROM.GetSerialNumber().de-
code('UTF-8')

#Setting input system to static input
icc1c.channel[0].StaticInput.SetAsInput()

min_lens_current = -300
max_lens_current = 300

# Sweep current
for current in range(int(min_lens_current), int(max_lens_current)+1,
int((max_lens_current-min_lens_current)/6)):
    #Value has to be converted from mA to A
    current_in_A = float(current)/1000
    icc1c.channel[0].StaticInput.SetCurrent(current_in_A)
    sleep(1)

icc1c.channel[0].StaticInput.SetCurrent(0.0)

# Running the signal generator
icc1c.channel[0].SignalGenerator.SetAsInput()
icc1c.channel[0].SignalGenerator.SetUnit(UnitType.FP)
icc1c.channel[0].SignalGenerator.SetShape(WaveformShape.SINUSOIDAL)
icc1c.channel[0].SignalGenerator.SetAmplitude(3)
icc1c.channel[0].SignalGenerator.SetFrequency(5)
icc1c.channel[0].SignalGenerator.Run()

for index in range(5):
    sleep(1)

icc1c.channel[0].SignalGenerator.Stop()

sleep(1)

icc1c.disconnect()
```


No representation or warranty, either expressed or implied, is made
as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

5. Firmware

The ICC-1C firmware architecture is organized into multiple Systems, which are modular components of the firmware that collectively provide the device's functionality. Each system may expose a different set of registers and/or vectors, which are user-accessible variables designed for communication and control.

Available systems:

System Name	System ID
STATUS	0x10
BOARD EEPROM	0x20
DEVICE EEPROM	0x21
TEMPERATURE MANAGER	0x22
LOGGER	0x24
MISCELLANEOUS FEATURES	0x25
VECTOR PATTERN MEMORY	0x26
SNAPSHOT MANAGER	0x27
VECTOR ADAPTER	0x28
I2C	0x31
SMART STEP MANAGER	0x3A
SIGNAL FLOW MANAGER	0x40
STATIC INPUT	0x50
ANALOG INPUT	0x58
SIGNAL GENERATOR	0x60
VECTOR PATTERN UNIT	0x68
INPUT CONDITIONING	0x98
OPEN LOOP	0xB0
LENS COMPENSATIONS	0xC8
OUTPUT CONDITIONING	0xD0
OUTPUT CONDITIONING FILTER BASE	0xD8
LINEAR OUTPUT	0xE8

Static Input

Static Input system ID: **0x50**

The static input stage system belongs to the input stage systems of the **Signal Flow** and is the one which is by factory default activated at start up. With this system the user can set constant values as input to the signal flow.

Register Map:

Address	Name	Default Value	Description	Type	Access
0x5000	Current static input	0.0	Static input in amperes. Range is from -0.5 to 0.5	float	read write
0x5001	not used			float	read write
0x5002	not used	0.0		float	read write
0x5003	Active static input type	0	Determined by active control stage system	Unit Type	read only
0x5004	FP static input	0.0	Static input in diopters	float	read write

5.1. Firmware content description

The ICC-1C's firmware uses the following types of registers and vectors for communication with the host PC or system:

- **Registers:**
  - **Definition:** Each register is a 32-bit value used for reading and/or writing data from/to the controller.
  - **Data Transfer:** The register data is always transmitted in big-endian format (the most significant byte is transmitted first).
  - **Data Types:** The specific meaning of the 32-bit value is system-dependent and documented in the system's firmware documentation. The common formats include:
    - IEEE 754 32-bit float.
    - Unsigned integers.
    - Boolean values.
    - Custom-defined formats.
  - **Access Types:**
    - **Read-only:** Used for monitoring or retrieving data from the controller.
    - **Write-only** or **Read/Write:** Used to trigger some operations or update device states.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

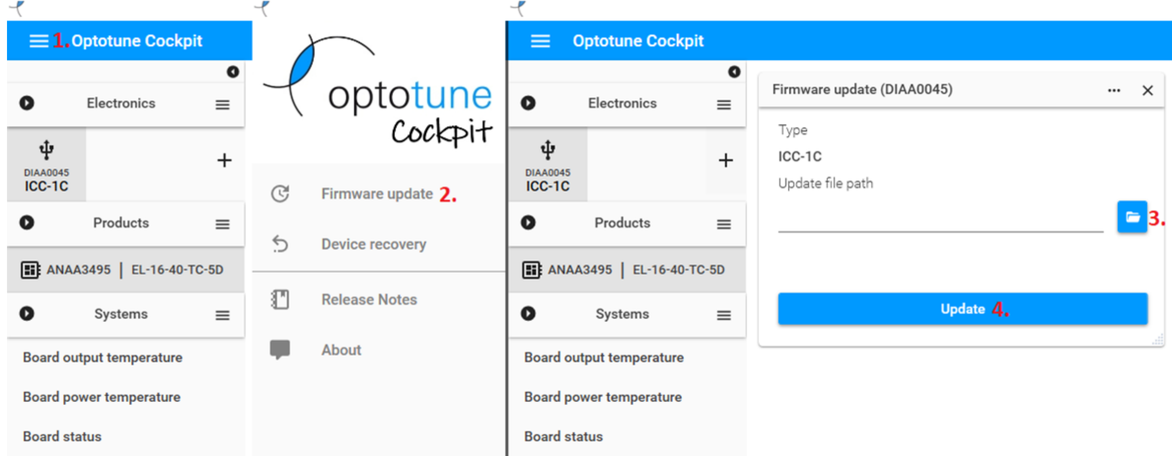
Copyright © 2025 Optotune

- **Vectors:**
  - **Definition:** Vectors are variable-length equivalents of registers, typically used to manage arrays of data.
  - **Structure:**
    - Vectors have defined size and data type, as documented for each system.
    - Byte-addressable in most cases, though some systems may require specific alignment (e.g. 4-byte chunks).
  - **Indexing:** When working with vectors, they require an index to specify the portion of the vector being accessed during the read/write operations.
- **Documentation:**
  - More comprehensive details about the supported registers and vectors, their formats, and uses can be found in the ICC-1C Firmware Documentation. The firmware (.hex file) and its related documentation can be downloaded from Optotune's Download Center, after completing the required registration. The documentation must be first unzipped to show the correct formatting.

5.2. Firmware update

The ICC-1C's firmware can be updated via the Optotune Cockpit software using the USB connection. **Updating the firmware via I2C or Ethernet is not supported.** To perform a firmware update of the controller, the following steps should be performed:

1. Open the menu located in the top-left corner of the Optotune Cockpit interface.
2. Select the "Firmware Update" option. A new widget will appear on the current dashboard for managing the firmware update process.
3. Click on the folder icon within the widget to browse for the appropriate .hex file.
4. After selecting the .hex file, click on the "Update" button to begin the firmware installation process.



5. Once the update is completed, please double-check the firmware version to ensure the update was successfully applied (right-click on the icon of the ICC-1C under the "Electronics" and select "Properties").

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

6. UART Communication Protocol

This chapter describes the protocols supported by the ICC-1Cs UART interface. While it is generally advised to use either of the SDKs presented in chapter 4, the following enables to program ICC-1C on platforms not covered by the Python and C# languages.

The serial communication in the system operates in one of two distinct modes: Simple Mode (ASCII Mode) and Pro Mode (Binary Mode). Simple mode protocol is easier to implement but does not cover all functionalities of the controller.

This mode is active by default upon a system startup. It provides a basic command set (Table 1. List of simple mode commands) based on **RS-232** serial communication for straightforward interactions and quick setup. It's ideal for scenarios where minimal communication complexity is required. It allows interaction with the ICC-1C via commonly available serial terminals like **Termite** or **Putty**.

Interface Details:

- **Baud Rate:** Arbitrary (auto-detected by the device).
- **Data Bits:** 8.
- **Stop Bits:** 1.
- **Parity:** None (N).
- **Baud Rate Auto-Detection:** The device can adapt to baud rate changes during operation, to do so the "START" command must be issued.

Command Structure:

- Communication is **ASCII-based**, using predefined command and response formats.
- Commands and responses are terminated with a **CR, LF** sequence (i.e., 0x0D, 0x0A in hexadecimal).
- **Case Insensitivity:** Commands are not case-sensitive, simplifying their use.
- **Whitespace Handling:** Extra white spaces in commands are ignored, ensuring flexibility in formatting.
- Commands can be issued through a serial terminal interface.
- The ICC-1C responds with ASCII-encoded messages (Table 2. List of simple mode replies), making debugging and testing straightforward.

Table 1. List of simple mode commands

Simple mode command	Description
START[CR][LF]	Answers "OK" if controller is ready to use and a device is detected. Otherwise "ERROR" is received.
STATUS[CR][LF]	Controller answers with a status encoded within 4 bytes of information. Example: "0x00015000[CR][LF]". See the next section for further description of the status bytes.
ACKNOWLEDGE[CR][LF]	Clears history error flags in the status register. Answers "OK".
RESET[CR][LF]	Restarts the controller's firmware. Note: no answer is sent via serial line.
GOTODFU[CR][LF]	Starts the controller's loader for firmware update. Note: no answer is sent via serial line.
GOPRO[CR][LF]	Starts binary protocol-based mode of serial communication. Serial message CRC is not checked.
GOPROCRC[CR][LF]	Starts binary protocol-based mode of serial communication. Serial message CRC is checked.
GETID[CR][LF]	Answers with firmware serial number. Example: "18012600-00-A[CR][LF]".
GETVERSION[CR][LF]	Answers with firmware version number. Example: "1.1.741857[CR][LF]".
GETGITSHA1[CR][LF]	Answers with 40 bytes hexadecimal GIT build identification. Example: "24bb55274a300fe7116aaec5f8f7b023167faa7e[CR][LF]"
GETSN[CR][LF]	Answers with board and device serial number. Example: "Board: DIAA0045, Device: ANAA3495[CR][LF]".

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

DETECTDEVICE[CR][LF]	Runs autodetection of device on active channel, answers with device name. Example: “EL-16-40-TC-5D[CR][LF]”.
GETDEVICESN[CR][LF]	Answers with serial number of a device connected. Example: “DeviceSN: ANAA3495[CR][LF]”
SETCURRENT=%float[CR][LF]	Sets current value. Command supports decimal parameter value in mA units. Current value is limited either by power capabilities of 1CC-1C controller itself or connected device.
GETCURRENT[CR][LF]	Answers with value of active current. Returned value is decimal number in units of milliamperes, Example: “15.6[CR][LF]”
SETFP=%float[CR][LF]	Sets focal power. Supports float value in units of diopters limited to detected lens device capability.
GETFP[CR][LF]	Answers with focal power. Returned value is a float in diopters. If no lens is detected, it returns “NO”.
GETFPMIN[CR][LF]	Answers with focal power lower limit of lens device connected. Returned focal power is decimal value in diopters. If no lens is detected, it returns “NO”.
GETFPMAX[CR][LF]	Answers with focal power upper limit of lens device connected. Returned focal power is a decimal value in diopters. If no lens is detected, it returns “NO”.
GETTEMP[CR][LF]	Answers with actual temperature of device connected. Returned temperature is a decimal value in units of degree Celsius. Example: ”27.54[CR][LF]”.
SETTEMPLIM=%float[CR][LF]	Sets operational temperature limit in degree Celsius.
SETCURLIM=%float;%float[CR][LF]	Set operational current limits in mA. The first argument is the positive limit < 0 to Max current >, the second one is the negative limit < -Max current to 0 >.
GETCURLIMIT[CR][LF]	Answers with the active current limits in mA. Example:”500, -500[CR][LF]”.

Table 2. List of simple mode replies

Simple mode reply	**Description**
OK[CR][LF]	Command accepted and performed without limits.
NO[CR][LF]	Command not accepted, for any reason.
OL[CR][LF]	Command not accepted, because parameter reached lower limit.
OU[CR][LF]	Command not accepted, because parameter reached upper limit.
ERROR[CR][LF]	Command not available.

6.1. Pro (binary) mode

This mode offers an extended and more complex command set for advanced functionality and detailed control. Suitable for applications requiring precise operations, custom workflows, or integration into more complex systems.

Pro mode is a byte-based protocol inspired by HDLC. Individual messages are delimited with the "delimiter" byte, *0x7e*. To avoid the delimiter byte appearing in the message data, an "escape byte", *0x7d*, is also used, and byte stuffing is performed on every message before being sent out (so the receiver should then perform byte destuffing) as follows:

- Every appearance of the bytes *0x7e* or *0x7d* in the data (so in a role other than the delimiter or the escape byte) is replaced by the sequence *0x7d 0x5e* or *0x7d 0x5d*, respectively. In other words, every byte that needs to be escaped by being XOR-ed with *0x20* and prepended with *0x7d*.
- When destuffing, every instance of *0x7d* should be treated as an escape byte – it should not be interpreted as a data byte, it only marks that the following byte should be XOR-ed with *0x20* before being interpreted as another data byte.

Above the delimiting and stuffing layer, every pro mode message has the following format:

No representation or warranty, either expressed or implied, is made
as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

Reserved (slave address)	Command	Data size	Data (payload)	CRC
1 byte	1 byte	1 byte	0 - 50 bytes	2 bytes

The semantics of the individual fields are as follows:

Slave address – this field is currently unused
Command – a 1-byte unique identifier of the selected command (see below)
Data size – the number of bytes in the following data field
Data – the data required by the command or response, limited to 50 bytes
CRC – the CRC checksum, present even if CRC is deactivated (the values can be ignored in that case)

Available pro mode commands:

Command name	Code	Command payload	Response payload
Generic command	0x00	empty	empty
Get firmware identification	0x01	empty	32bits firmware ID
Get status	0x02	empty	32bits status register
Start self-test	0x03	empty	empty
Load configuration	0x04	1 byte snapshot id	empty
Store configuration	0x05	1 byte snapshot id	empty
Set communication mode	0x06	1 byte mode	empty
Set value	0x10	2 bytes register id, 4 bytes value	empty
Get value	0x11	2 bytes register id	4 bytes register value
Set multiple values	0x12	2 bytes value count = CNT, 2xCNT bytes register ids, 4xCNT bytes register values	empty
Get multiple values	0x13	2 bytes value count = CNT, 2xCNT bytes register ids	2 bytes value count = CNT, 4xCNT bytes register values
Set vector	0x14	2 bytes vector id, 2 bytes index, 1 byte length = SIZE, SIZE bytes data	empty
Get vector	0x15	2 bytes vector id, 2 bytes index, 1 byte length = SIZE	SIZE bytes data

Responses:

Responses have the same format as the commands (possibly with different payload data, see above) with the command code of the response corresponding to the command code of the command being responded to. However, error responses are marked by the most significant bit being set in the command code (in other words, an *0x80* is added to the command code), and the payload should consist of a single 4-byte error flag, see the Error Codes section of the firmware documentation.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

6.2. Pro mode example

The section below shows a detailed description of pro mode with example of setting the Signal generator and making a Temperature reading from the output stage of the ICC-1C controller. The example functions are written in Python, but they should be easy to understand and straightforward to implement on any platform.

The serial communication is first set to pro mode with disabled CRC using the "GOPRO" command. After the Pro mode communication has been established, the controller switches to a mode where it expects byte packets in the format below (different from simple commands), with the delimiter *0x7e*.

When Pro mode is activated, none of the simple mode commands will work, until the specific command to change the communication mode back to simple mode is sent. After this command is sent, the simple mode communication is established, and the controller will again accept string-based input commands from the simple serial mode communication section.

Turning on the signal generator on the ICC-1C (channel 0 by default), after all its necessary parameters have been set up:

self.send_cmd (b'7e' + b'00' + b'10' + b'06' + b'6001' + b'00000001' + b'0000' + b'7e')

Where:

0x7e is the delimiter + *0x00* is an unused byte (slave address) + *0x10* is the *Set value* pro mode command + *0x06* is the data size of 6 bytes + *0x6001* is the register ID to run the signal generator for channel 0 + *0x00000001* is the *True* boolean value to be written into the register + *0x0000* are the 2 unused CRC bytes + *0x7e* is the delimiter again.

Command name	Code	Command payload	Response payload	Note
Set value	0x10	2 bytes register id, 4 bytes value	empty	see **Architecture** for explanation of Systems and registers

Address	Name	Default Value	Description	Type	Access
0x6[0-7]01	Run	false	Turn Signal Generator on/off	boolean	read write

Reading the temperature of the output stage – register *0x2202* is done by sending the following command:

self.send_cmd (b'7e' + b'00' + b'11' + b'02' + b'2202' + b'0000' + b'7e')

Where:

0x7e is the delimiter + *0x00* is an unused byte (slave address) + *0x11* is the *Get value* pro mode command + *0x02* is the data size of 2 bytes + *0x2202* is the register ID for the output stage temperature + *0x0000* are the 2 unused CRC bytes + *0x7e* is the delimiter again.

Command name	Code	Command payload	Response payload	Note
Get value	0x11	2 bytes register id	4 bytes register value	see **Architecture** for explanation of Systems and registers

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

Address	Name	Default	Description	Type	Access
0x2202	Output stage board temperature	N/A	Degrees Celsius	float	read only

Switch command mode back to Simple mode:

self.send_cmd (b'7e' + b'00' + b'06' + b'01' + b'00' + b'0000' + b'7e')

Where:

0x7e is the delimiter + *0x00* is an unused byte (slave address) + *0x06* is the *Set communication mode* command + *0x01* is the data size of 1 byte + *0x00* is the *0* value to set the device in simple mode + *0x0000* are the 2 unused CRC bytes + *0x7e* is the delimiter again.

Command name	Code	Command payload	Response payload	Note
Set communication mode	0x06	1 byte mode	empty	0 for simple mode, unchanged otherwise

Example code (written in Python):

```
def float_to_ieee754_hex(x):
    # Pack the float as a 32-bit float in IEEE 754 format
    packed_float = struct.pack('>f', x)
    # Unpack the bytes as an unsigned integer and format as hex
    hex_representation = ''.join(f'{b:02x}' for b in packed_float)
    return hex_representation.encode('ascii')


def activate_signal_generator(self, channel=0):
    self.send_cmd(b'7e001006' + b'4' + str(channel).encode('ascii') + b'00'
+ b'00000060' + b'00007e')


def set_current_unit_type_signal_generator(self, channel=0):
    self.send_cmd(b'7e001006' + b'6' + str(channel).encode('ascii') + b'00'
+ b'00000000' + b'00007e')


def set_fp_unit_type_signal_generator(self, channel=0):
    self.send_cmd(b'7e001006' + b'6' + str(channel).encode('ascii') + b'00'
+ b'00000003' + b'00007e')


def run_signal_generator(self, channel=0):
    self.send_cmd(b'7e001006' + b'6' + str(channel).encode('ascii') + b'01'
+ b'00000001' + b'00007e')


def stop_signal_generator(self, channel=0):
    self.send_cmd(b'7e001006' + b'6' + str(channel).encode('ascii') + b'01'
+ b'00000000' + b'00007e')
```


No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

```
def set_shape_signal_generator(self, channel=0, shape=0):
    self.send_cmd(b'7e001006' + b'6' + str(channel).encode('ascii') + b'02'
+ b'0000000' + str(shape).encode('ascii') + b'00007e')


def set_frequency(self, channel=0, frequency=0.0):
    self.send_cmd(b'7e001006' + b'6' + str(channel).encode('ascii') + b'03'
+ float_to_ieee754_hex(frequency) + b'00007e')


def set_amplitude(self, channel=0, amplitude=0.0):
    self.send_cmd(b'7e001006' + b'6' + str(channel).encode('ascii') + b'04'
+ float_to_ieee754_hex(amplitude) + b'00007e')


def set_offset(self, channel=0, offset=0.0):
    self.send_cmd(b'7e001006' + b'6' + str(channel).encode('ascii') + b'05'
+ float_to_ieee754_hex(offset) + b'00007e')


def set_phase(self, channel=0, phase=0.0):
    self.send_cmd(b'7e001006' + b'6' + str(channel).encode('ascii') + b'06'
+ float_to_ieee754_hex(phase) + b'00007e')


def set_cycles_signal_generator(self, channel=0, cycles=-1):
    #  negative value sets infinite loop
    if cycles == -1:
        self.send_cmd(b'7e001006' + b'6' + str(channel).encode('ascii') +
b'07' + b'ffffffff' + b'00007e')
    # after reaching the number of cycles the signal generator stops
    else:
        self.send_cmd(b'7e001006' + b'6' + str(channel).encode('ascii') +
b'07' + str(cycles).encode('ascii') + b'00007e')


def get_output_stage_temperature(self):
    self.send_cmd(b'7e001102220200007e')


def get_power_supply_temperature(self):
    self.send_cmd(b'7e001102220400007e')
```


No representation or warranty, either expressed or implied, is made
as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

7. Ethernet Communication

Besides UART, the ICC-1C is also capable to communicate via Ethernet, which allows seamless implementation into industrial systems or being accessible from multiple host computers. During the Ethernet communication, by default the device uses TCP/IP communication port number 5000. For IP address detection purposes, the device replies to any broadcast message with the controller's ID and serial number over the UDP port number 30321.

The IP settings of the device can be configured automatically by enabling the DHCP configuration feature, so that the device can obtain an unused IP address from the server (this configuration is active by default). Alternatively, the users can set up manual IP addressing by disabling the DHCP feature and configuring the IP Address, the Subnet Mask and the Default Gateway manually. All the IP configuration settings are stored in the ICC-1Cs Board EEPROM and can be changed using Optotune Cockpit software, the available SDKs (C# or Python) or any other communication interface (UART Pro mode or I2C).

7.1. IP settings configuration via DHCP

The IP settings of the ICC-1C can be configured to obtain an IP address from a nearby DHCP server. This option is selected by factory defaults, but can be changed by the user under the Board EEPROM settings in Optotune Cockpit:

1. Open the Board EEPROM widget and check the "DHCP enabled" checkbox.
2. Press the "Set to device" button and click "Yes" when asked for board reset.

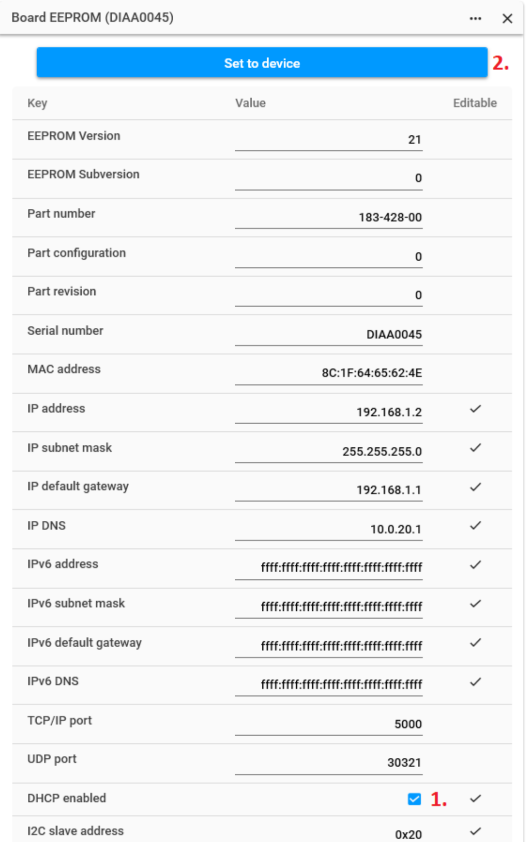


Figure 27: Enabling IP configuration via DHCP in Optotune Cockpit

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

7.2. Static IP address configuration

If the ICC-1C is connected directly to a computer via Ethernet, or if the connected network/router is configured with static IP addressing, the controller can be also configured for static IP addressing. This can be also done in the Optotune Cockpit software:

1. Open the Board EEPROM widget and uncheck the “DHCP enabled” checkbox.
    a. Leaving this box checked will ignore the static IP settings and the device will try to obtain the IP address from a server, even if there isn’t one.
2. Configure the “IP address”, the “IP subnet mask” and the “IP default gateway” fields for the device:
    a. For the “IP address” setting, make sure that the selected address is a usable Host IP address, and not an address for the entire subnetwork or a broadcast address.
    b. The “IP subnet mask” should match the one configured on the host PC or the connected sub-network.
    c. The “IP default gateway" should be configured to the IP address of the host PC, if they are connected directly, or if the controller is connected to a larger network, it usually should be the IP address of the router for the given network.
3. Press the “Set to device” button and click “Yes” when asked for board reset.
4. If a direct connection to a host PC is used, the IP address should also be configured on the PC:
    a. Open the Control Panel and navigate to Network and Sharing Center -> Ethernet -> Properties -> Internet Protocol Version 4 (TCP/IPv4) -> Properties.
    b. Select “Use the following IP address:” option.
    c. The IP address of the PC should match the previously set “IP default gateway” from the ICC-1C.
    d. The Subnet mask should be the same as the one configured on the ICC-1C.
    e. The Default gateway of the PC should match the previously set “IP address” from the ICC-1C.
5. Click on OK in both Properties windows.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

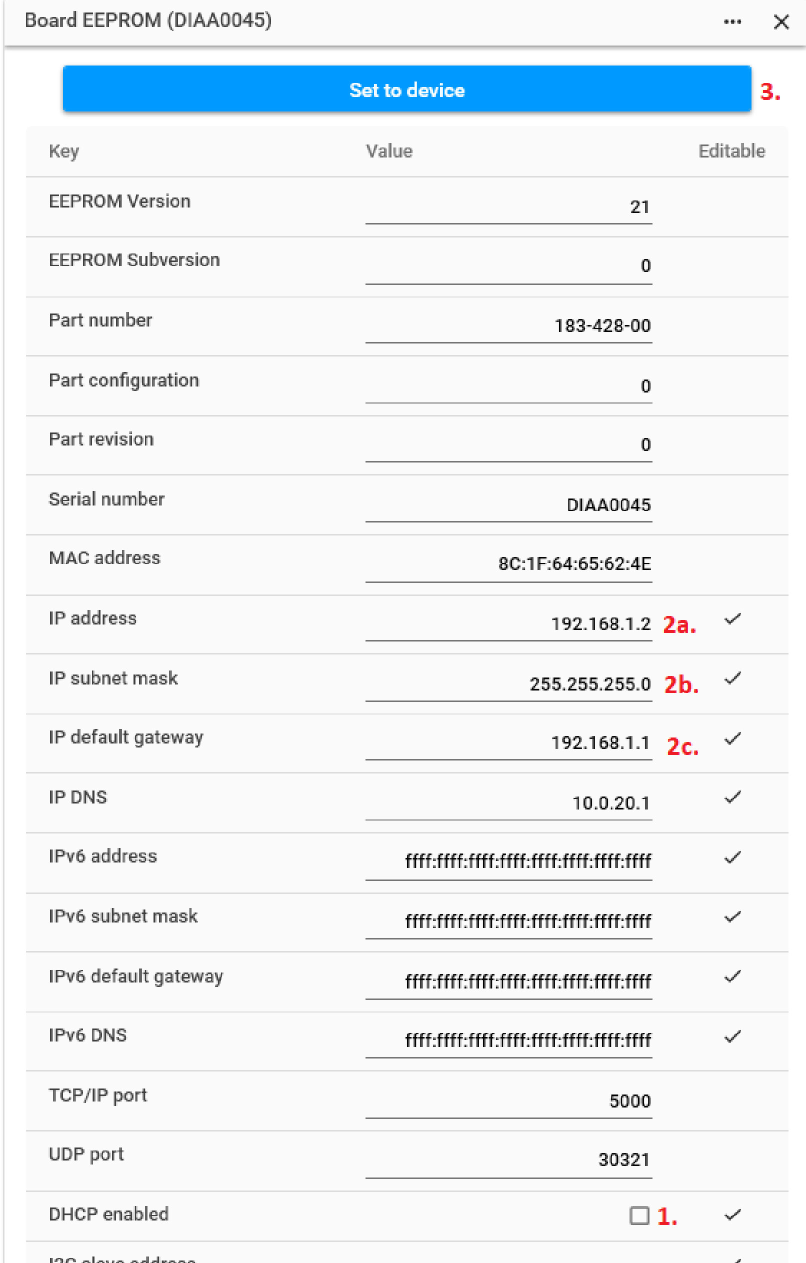


Figure 28: Example of a static IP address configuration in Optotune Cockpit

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

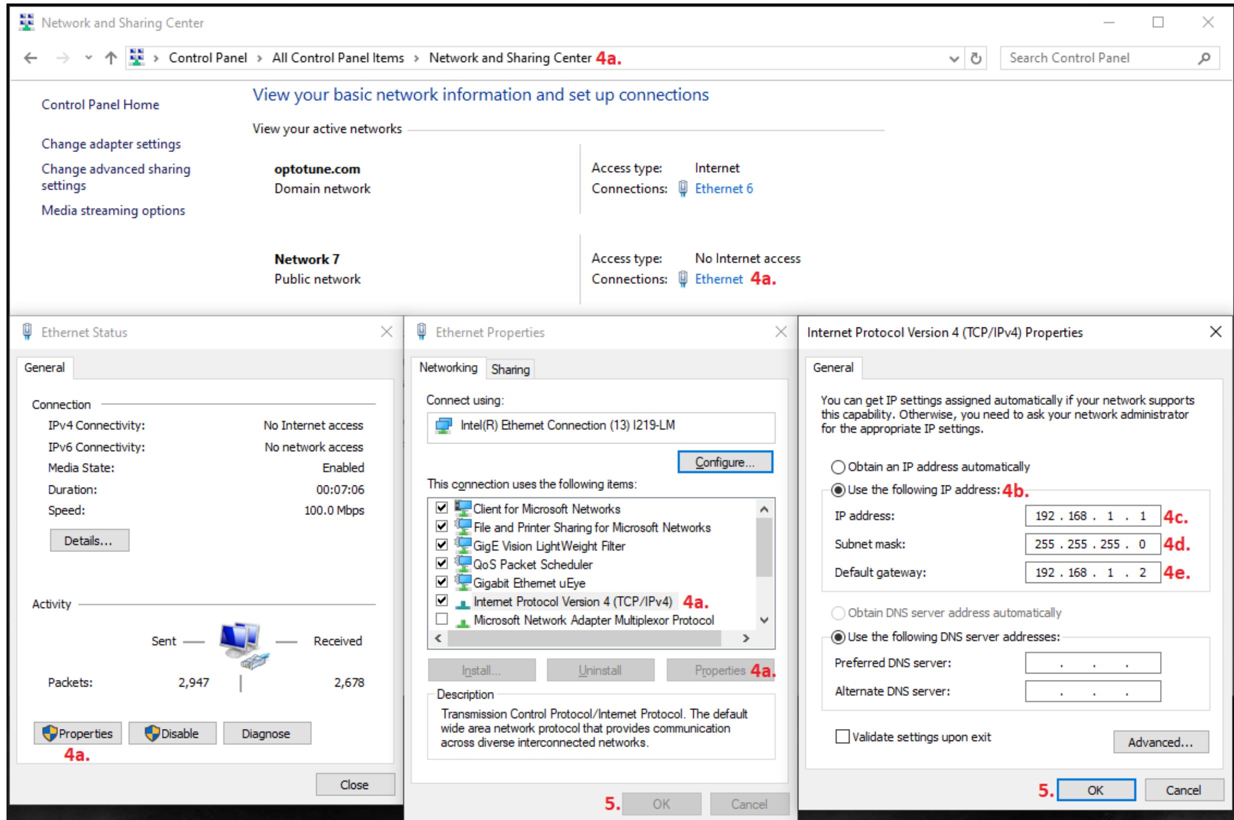


Figure 29: Example of a static IP address configuration in Windows

7.3. Ethernet communication example via Cockpit or a Telnet Client

To connect with the ICC-1C via Ethernet, the users can use the Optotune Cockpit software by selecting the Ethernet tab in the “Add electronics” window, but any Telnet Client software should work (for example Putty or Hercules).

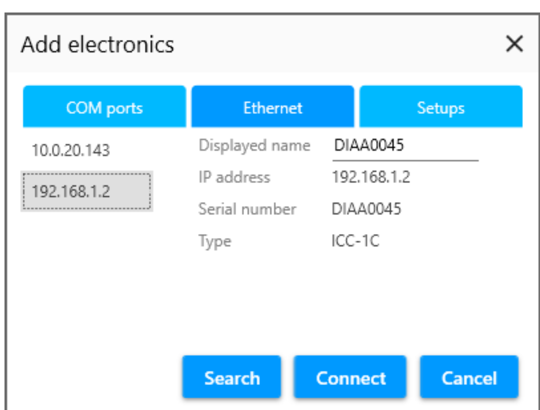


Figure 30: Connecting to the ICC-1C via Ethernet using Optotune Cockpit

To initiate a TCP communication via a Telnet Client, the IP address of the device should be entered to the appropriate field and TCP port number 5000 shall be selected. During the Ethernet communication, the ICC-1C uses the same set of commands and replies as for the UART communication. It is possible to use the simple mode commands

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

terminated with a <CR> <LF> sequence, or switch to Pro mode and sending binary commands to the controller.

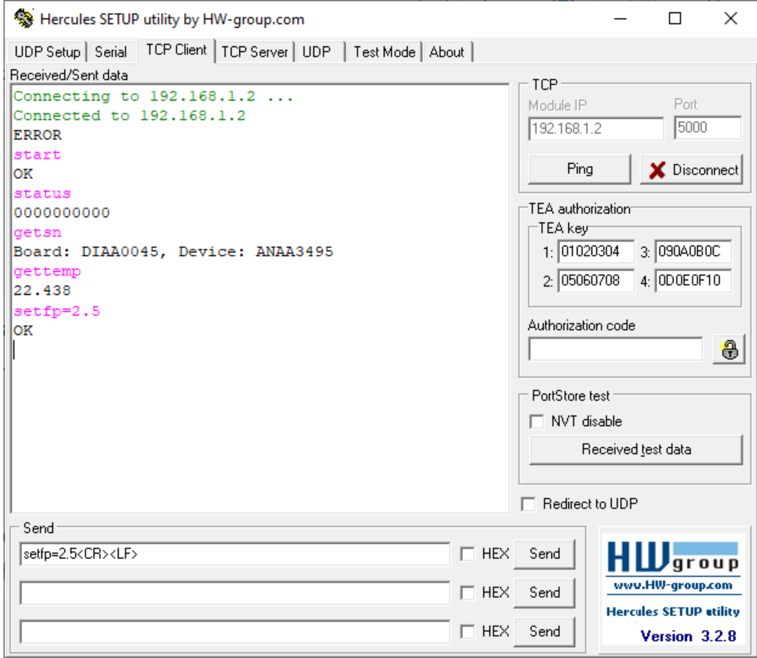


Figure 31: Ethernet communication with the ICC-1C using a Telnet Client

Upon connecting to the ICC-1C, depending on the given client software an “ERROR” message might be received – this is caused by the software sending some identification request, which is not part of the ICC-1C’s instruction set and thus the controller replies with an error. This message can be ignored, and the user could proceed by sending the “START” command to initialize the ICC-1C.

7.4. Ethernet communication example via the Python SDK

The following example shows how to use the Python SDK for communication via Ethernet. The example shows two alternative ways of connecting to the controller, with a static IP address, and with dynamic addressing (DHCP).

```
from optoICC.tools.ethernet_searcher import EthernetSearcher
from optoICC.icc1c import ICC1cBoard
from optoKummenberg import UnitType
from optoICC import WaveformShape
from time import sleep


board = ICC1cBoard()
mode = input("Static IP or automatic search?(A = auto, S = static):
").strip().upper()
if mode == 'A':
    devices = EthernetSearcher.search_ethernet_boards()
    if not devices:
        print("No ethernet boards found.")
        return

    print(f"Boards found ({len(devices)}):")
    for idx, dev in enumerate(devices):
```


No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

```
        print(f"[{idx}] Serial: {dev.serial_number}, IP: {dev.ip}, Port: 
{dev.port}")

    try:
        selection = int(input("Select index of the device you want to con-
nect: "))
        selected = devices[selection]
        board = ICC1cBoard(ip_address=selected.ip)
        print("Board connected successfully!")
    except Exception as e:
        print(f"Connection failed: {e}")

elif mode == 'S':
    ip = input("Specify IP address of the board: ").strip()
    try:
        board = ICC1cBoard(ip_address=ip)
        print("Board connected successfully!")
    except Exception as e:
        print(f"Connection failed: {e}")
else:
    print("Invalid input.")

# Example usage
board.channel[0].SignalGenerator.SetAsInput()
board.channel[0].SignalGenerator.SetUnit(UnitType.CURRENT)
board.channel[0].SignalGenerator.SetShape(WaveformShape.STAIRCASE)
board.channel[0].SignalGenerator.SetAmplitude(0.15)
board.channel[0].SignalGenerator.SetFrequency(2)
board.channel[0].SignalGenerator.Run()
sleep(5)
board.channel[0].SignalGenerator.Stop()
board.disconnect()
```


No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

8. I2C Communication

In addition to UART and Ethernet communication, the ICC-1C also supports I2C communication, making it possible to easily integrate into MCU controlled systems. This is particularly advantageous when the user wants to control the ICC-1C as a part of a larger industrial system and align it with different cameras and actuators used by that system.

8.1. I2C Configuration

To set up an I2C communication with the ICC-1C, three parameters can be configured in the device's Board EEPROM settings:

1. Since the Auxiliary I/O connector has shared pins for I2C and UART (pins 5 and 7), the ICC-1C provides an option to choose the communication type and speed by the "Input interface" setting.
   By default, this setting is configured as "AutoDetect", which means that the device will detect the communication type based on the first data packet, but it can be set to work only as I2C or to UART with a specific Baud rate.
   Warning: Changing this setting will not allow the device to communicate with the ICC-1C by the other communication option (but Ethernet and USB lines will still be available).
2. The "I2C slave address" setting is used to store the I2C address for the ICC-1C.
   The default address is set to *0x20*, but can be changed by the user to any desired value.
3. The "I2C register count" setting determines, how many of the ICC-1C's registers are changed by a single I2C communication, e. g. how many data bytes are following the device address during the I2C communication.
   The default value is set to be 2 registers, but it is possible to set it to 1 or 4 registers at a time as well.
4. To apply the configured settings, press the "Set to device" button and click "Yes" when asked for board reset.

No representation or warranty, either expressed or implied, is made
as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

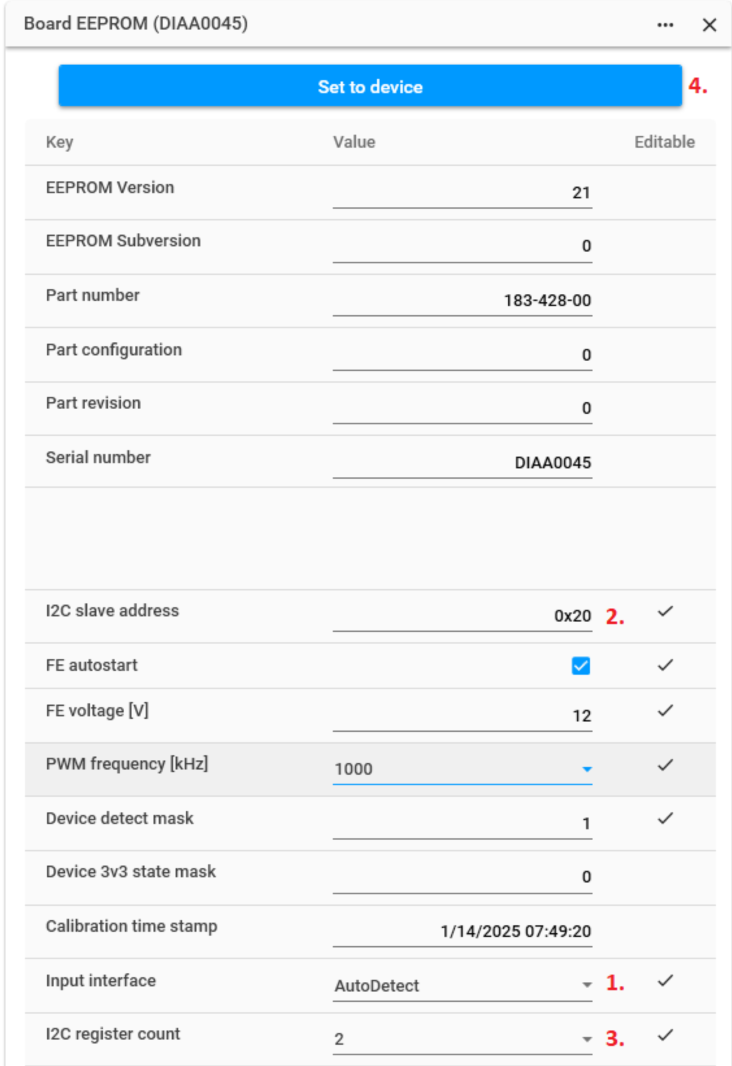


Figure 32: Ethernet communication with the ICC-1C using a Telnet Client

8.2. I2C communication protocol description

Depending on the “I2C register count” setting in the Board EEPROM, the ICC-1C can be configured via I2C communication by setting 1, 2 or 4 registers at a time. Since the device’s registers are defined with a 4-digit hexadecimal address and they are storing a 32-bit data value, changing a single register via I2C requires sending 6 bytes of data (1 byte system ID + 1 byte register address and 4 bytes of data) in a single communication. If the device is set to configure multiple registers, then the final number of bytes sent after the I2C address should be the “I2C register count” setting times 6 bytes (it should be 6, 12 or 24 bytes).

Reading data from the controller is done via 4 read pointer registers at addresses from 0x3100 to 0x3103. Writing a system ID and register address into these registers will copy the content of the given register into this pointer and then it can be read out by the following I2C read command. The device also contains 4 write error response registers (addresses from 0x3104 to 0x3107), which contain a possible error response for the previously initiated write command through I2C and it can be also read out by the following I2C read command.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

Address	Name	Default Value	Description	Type	Access
0x3100	I2C read pointer 0	0x1003	The address of the register that will be used to evaluate the first register in the read request Value is updated at controller frequency.	uint32	read write
0x3101	I2C read pointer 1	0x1004	The address of the register that will be used to evaluate the second register in the read request Value is updated at controller frequency.	uint32	read write
0x3102	I2C read pointer 2	0x1003	The address of the register that will be used to evaluate the first register in the read request Value is updated at controller frequency.	uint32	read write
0x3103	I2C read pointer 3	0x1004	The address of the register that will be used to evaluate the second register in the read request Value is updated at controller frequency.	uint32	read write
0x3104	Write error response 0	0	Error response of the first register of the write request.	uint32	read only
0x3105	Write error response 1	0	Error response of the second register of the write request.	uint32	read only
0x3106	Write error response 2	0	Error response of the third register of the write request.	uint32	read only
0x3107	Write error response 3	0	Error response of the forth register of the write request.	uint32	read only

When the device has been configured to write 2 or 4 registers at the same time, but only fewer registers are required to be changed by the user, the unneeded register write commands can be replaced by configuring the read pointers to copy the value of any system and register ID.

8.3. I2C Write command structure

The following examples show the I2C command structure for setting 2 registers at the same time (default register count setting).

- Set 2.5 dpt focal power as static input (1st register to write):
  - System ID for “STATIC INPUT”: *0x50.*
  - Register address for “FP STATIC INPUT”: *0x04.*
  - The 32-bit float of 2.5 dpt in hexadecimal format corresponds to *0x40200000*, so the data bytes are: *0x40 0x20 0x00 0x00.*
- Set Read pointer 1 for the lens temperature (2nd register to write):
  - System ID for the I2C Read Pointers: *0x31.*
  - Register address for I2C Read Pointer 1: *0x01.*
  - The System ID for “TEMPERATURE MANAGER” is *0x22*, and the register address for “DEVICE TEMPERATURE” is *0x00.*

	I2C slave address	Write to the 1st register						Write to the 2nd register					
Byte#	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12
Meaning	I2C slave address	Sys. ID 1	Reg. address 1	Reg. data 1	Reg. data 1	Reg. data 1	Reg. data 1	Sys. ID 2	Reg. address 2	Empty byte	Empty byte	Sys. ID to read	Reg. address to read
Hex value	0x20	0x50	0x04	0x40	0x20	0x00	0x00	0x31	0x01	0x00	0x00	0x22	0x00
Operation	Select device (write)	Set 2.5 dpt focal power as static input						Set Read pointer 1 for lens temperature register					

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

8.4. I2C Read command structure

Reading of the ICC-1C's registers can be done in two steps. In the first step the read pointers must be configured:

- Set Read pointer 0 for the actual output current (1st register to write):
  - System ID for the I2C Read Pointers: *0x31.*
  - Register address for I2C Read Pointer 0: *0x00.*
  - The System ID for "LINEAR OUTPUT" is *0xE8*, and the register address for "OUTPUT CURRENT" is *0x02.*
- Set Read pointer 1 for the lens temperature (2nd register to write):
  - System ID for the I2C Read Pointers: *0x31.*
  - Register address for I2C Read Pointer 1: *0x01.*
  - The System ID for "TEMPERATURE MANAGER" is *0x22*, and the register address for "DEVICE TEMPERATURE" is *0x00.*

	I2C slave address	Write to the 1st register						Write to the 2nd register					
Byte#	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12
Meaning	I2C slave address	Sys. ID 1	Reg. address 1	Empty byte	Empty byte	Sys. ID to read 1	Reg. address to read 1	Sys. ID 2	Reg. address 2	Empty byte	Empty byte	Sys. ID to read 2	Reg. address to read 2
Hex value	0x20	0x31	0x00	0x00	0x00	0xE8	0x02	0x31	0x01	0x00	0x00	0x22	0x00
Operation	Select device (write)	Set Read pointer 0 for output current register						Set Read pointer 1 for lens temperature register					

In the next step the host can read out the I2C data:

- Data from Read pointer 0 for the actual output current (read from the 1st register):
  - System ID from Read pointer 0: *0xE8.*
  - Register address from Read pointer 0: *0x02.*
  - Data from Read pointer 0: *0x3E0DA1B0*, which in 32-bit float format equals to 0.1383121 A.
- Data from Read pointer 1 for the actual lens temperature (read from the 2nd register):
  - System ID from Read pointer 1: *0x22.*
  - Register address from Read pointer 1: *0x00.*
  - Data from Read pointer 1: *0x41FD0000*, which in 32-bit float format equals to 31.625 °C.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

	I2C slave address	Read from the 1st register						Read from the 2nd register					
Byte#	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12
Meaning	I2C slave address	Sys. ID 1	Reg. address 1	Reg. data 1	Reg. data 1	Reg. data 1	Reg. data 1	Sys. ID 2	Reg. address 2	Reg. data 2	Reg. data 2	Reg. data 2	Reg. data 2
Hex value	0x20	0xE8	0x02	0x3E	0x0D	0xA1	0xB0	0x22	0x00	0x41	0xFD	0x00	0x00
Operation	Select device (read)	Read pointer 0 containing Sys. ID, reg. address and data from the output current register						Read pointer 1 containing Sys. ID, reg. address and data from the lens temperature register					

8.5. Changing the I2C register count

The following examples show changing of the I2C register count command structure for setting 2 registers at the same time (default register count setting).

- Unlock the Board EEPROM (1st register to write):
  - System ID for "BOARD EEPROM": *0x20.*
  - Register address for "LOCK": *0x00.*
  - The 32-bit key value to unlock the Board EEPROM: *0x3F4744F6.*
- Change I2C register count to 1 (2nd register to write):
  - System ID for "BOARD EEPROM": *0x20.*
  - Register address for "SLAVE I2C REGISTER COUNT": *0x17.*
  - The 32-bit integer value for 1 register in hexadecimal format corresponds to *0x00000001*, so the data bytes are: *0x00 0x00 0x00 0x01.*

	I2C slave address	Write to the 1st register						Write to the 2nd register					
Byte#	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12
Meaning	I2C slave address	Sys. ID 1	Reg. address 1	Reg. data 1	Reg. data 1	Reg. data 1	Reg. data 1	Sys. ID 2	Reg. address 2	Empty byte	Empty byte	Empty byte	Reg. data 2
Hex value	0x20	0x20	0x00	0x3F	0x47	0x44	0xF6	0x20	0x17	0x00	0x00	0x00	0x01
Operation	Select device (write)	Unlock the Board EEPROM						Set I2C register count to 1					

After changing the I2C register count settings in the EEPROM, a **Board Reset (power cycling) should be performed** for the changes to take effect.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

8.6. I2C Communication example (Raspberry Pi)

The following example shows sending direct I2C commands to the ICC-1C from a Raspberry Pi's console.

Write Command:

i2ctransfer –y 1 w12@0x20 0x50 0x04 0x40 0x20 0x00 0x00 0x31 0x01 0x00 0x00 0x22 0x00

w12 – Write 12 bytes

0x20 – I2C slave address

0x50 0x04 – Sys ID 1 and Reg. address 1 (Set static input in dpts)

0x40 0x20 0x00 0x00 – Data for register 1 (2.5 dpt encoded in hexadecimal format)

0x31 0x01 – Sys ID 2 and Reg. address 2 (I2C Read pointer 1)

0x00 0x00 0x22 0x00 – Data for register 2 (2 empty bytes + copy lens temperature register into Read pointer 1)

Read Command:

i2ctransfer –y 1 r12@0x20

r12 – Read 12 bytes

0x20 – I2C slave address

Read command – answer from the ICC-1C:

0xe8 0x02 0x3e 0x01 0xf1 0xa0 0x22 0x00 0x41 0xe2 0x80 0x00

0xe8 0x02 – Sys ID 1 and Reg. address 1 (Output current register)

0x3e 0x01 0xf1 0xa0 – Data from register 1 (0.126898 A in hexadecimal format)

0x22 0x00 – Sys ID 2 and Reg. address 2 (Lens temperature register)

0x41 0xe2 0x80 0x00 – Data from register 2 (28.3125 °C in hexadecimal format)

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
pi@raspberrypi:~ $ i2ctransfer -y 1 w12@0x20 0x31 0x00 0x00 0x00 0xE8 0x02 0x31 0x01 0x00 0x00 0x22 0x00
pi@raspberrypi:~ $ i2ctransfer -y 1 r12@0x20
0xe8 0x02 0x00 0x00 0x00 0x00 0x22 0x00 0x41 0xfe 0x80 0x00
pi@raspberrypi:~ $ i2ctransfer -y 1 w12@0x20 0x50 0x04 0x40 0x20 0x00 0x00 0x31 0x01 0x00 0x00 0x22 0x00
pi@raspberrypi:~ $ i2ctransfer -y 1 r12@0x20
0xe8 0x02 0x3e 0x0d 0xa1 0xb0 0x22 0x00 0x41 0xfd 0x00 0x00
pi@raspberrypi:~ $ i2ctransfer -y 1 w12@0x20 0x20 0x00 0x3F 0x47 0x44 0xF6 0x20 0x17 0x00 0x00 0x00 0x01
pi@raspberrypi:~ $ i2ctransfer -y 1 w6@0x20 0x50 0x04 0xC0 0x00 0x00 0x00
pi@raspberrypi:~ $ i2ctransfer -y 1 r6@0x20
0xe8 0x02 0xbe 0x39 0xaa 0x6d
pi@raspberrypi:~ $
```

Figure 33: Ethernet communication with the ICC-1C using a Telnet Client

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

8.7. I2C Communication example (Arduino)

```
#include<Wire.h>
#define slaveAddress 0x20

//Read Device temperature and Output stage temperature
byte dataArray_ReadTempDataIndex[12] = {0x31, 0x00, 0x00, 0x00, 0x22, 0x00,
0x31, 0x01, 0x00, 0x00, 0x22, 0x02};

//Set focal Power and (dummy)Read Output stage temperature
byte dataArray_SetFocalPower25[12] = {0x50, 0x04, 0x40, 0x20, 0x00, 0x00,
0x31, 0x01, 0x00, 0x00, 0x22, 0x02};
byte dataArray_SetFocalPower0[12] = {0x50, 0x04, 0x00, 0x00, 0x00, 0x00,
0x31, 0x01, 0x00, 0x00, 0x22, 0x02};

int start = 0;

//Setup the Arduino
void setup()
{
  Wire.begin(); //initialise I2C
  Serial.begin(9600);
  delay (500);
  Serial.println("Setup done.");
}

//Run loop
void loop()
{
  if (start == 0) //Run loop only if start == 0
  {
    //Write Static Focal power to 2.5
    Wire.beginTransmission(slaveAddress);
    for (int i=0; i<12; i++)
    {
      Wire.write(dataArray_SetFocalPower25[i]); //data bytes are queued in
local buffer
    }
    Wire.endTransmission();
    Serial.println("FP set to 2.5 dpt.");

    delay (1000);

    //Write Static Focal power to 0
    Wire.beginTransmission(slaveAddress);
    for (int i=0; i<12; i++)
    {
      Wire.write(dataArray_SetFocalPower0[i]); //data bytes are queued in
local buffer
    }
    Wire.endTransmission();
    Serial.println("FP set to 0 dpt.");

    delay (1000);

    //Write to I2C Read pointers to device temperature and output stage temp
    Wire.beginTransmission(slaveAddress);
    for (int i=0; i<12; i++)
```


No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

```
    {
      Wire.write(dataArray_ReadTempDataIndex[i]); //data bytes are queued in
local buffer
    }
    Wire.endTransmission();
    delay (200);
    //Read the data from I2C
    Wire.requestFrom(slaveAddress, 12); // request 12 bytes from slave de-
vice
    Serial.print("Temp readings: ");
    while(Wire.available())// slave may send less bytes than requested
    {
      Serial.print(Wire.read(),HEX); //read hex values
      Serial.print(" "); //add space between the hex values
    }
    Serial.println();

    delay (2000);
    //start = 1; //stop the loop after 1 run
  } //End of if
} //End of Loop
```


No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

9. Analog Input Voltage mode

For the ICC-1C it is possible to map an analog input voltage 0-10 V to the driving current or the focal power (if applicable) of the connected lens. Both linear and non-linear mapping are possible. With linear interpolation, the controller linearly interpolates the analog voltage input range (0-10 V) to the corresponding pre-set output current or focal power range. With non-linear analog voltage transition, mapping points can be added manually by the user. For each point, the user can set the input voltage and the corresponding output current or focal power. A toggle button is available to enable constant or linear extrapolation outside of the defined points.

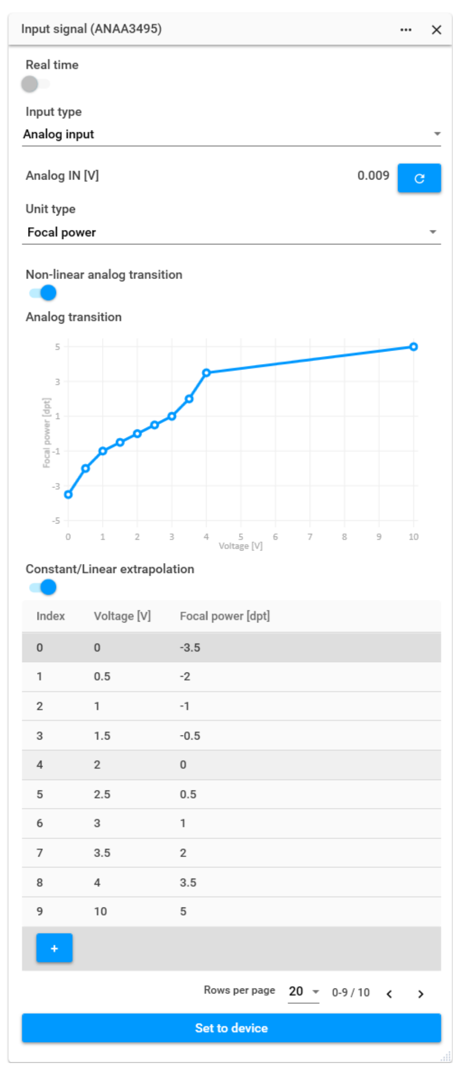


Figure 34: Analog input voltage with non-linear mapping of focal power values

Since the Analog input voltage mode may require to configure multiple points at each startup of the controller, the ICC-1C also offers a Snapshot Manager feature, which allows the user to store the settings in the device's non-volatile memory, so the snapshot with the mapping points will not be lost after powering off the device.

The ICC-1C has two snapshots available, which can be managed by the Snapshot manager widget in the Cockpit software. Snapshot number one is the “Factory settings,” which allows the user to reload the factory default settings on the device (this snapshot cannot be edited or overwritten). Users can create and store a new configuration in the second available snapshot. It is also possible to choose which snapshot should be loaded during the device start-up. For example, a user-defined snapshot can be saved as number 2 by pressing "Save" button. It is

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

recommended to also click on the "Load" button afterwards to check for potential failures. After that the snapshot number can be selected under the “Startup snapshot” section and using the "Set” button it can be configured to be the active setting after a device startup.

Figure 35: Snapshot manager widget

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

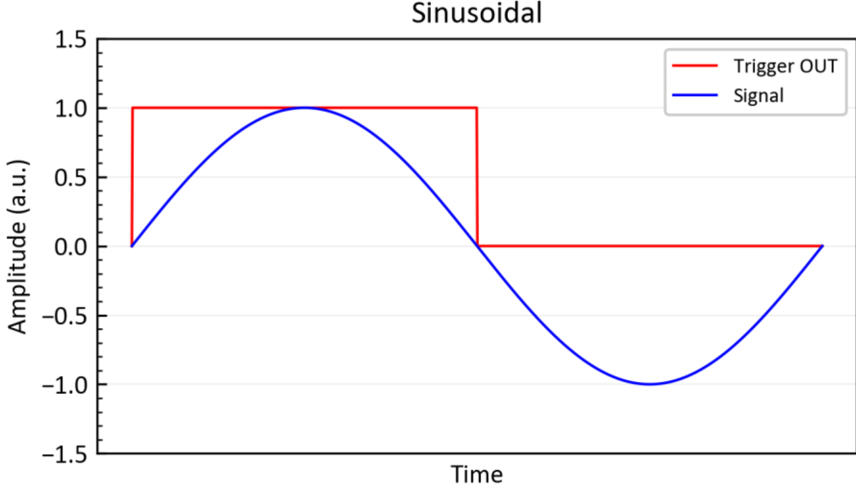
Copyright © 2025 Optotune

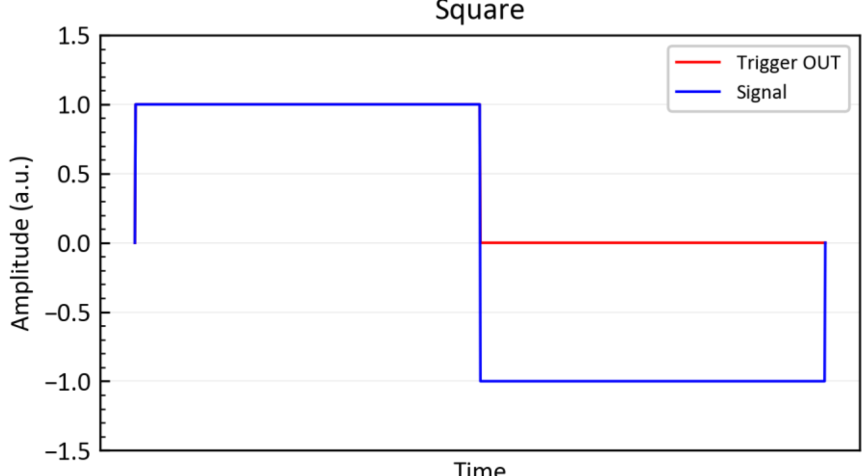
10. Input/Output trigger signals

The ICC-1C can be configured to generate an output trigger signal when changing the focal power/current of the lens, or alternatively, wait for an incoming trigger signal before executing a predefined change on the connected lens. These trigger signals can be synchronized with the controllers built-in signal generator or any custom vector. For the ICC-1C, the following type of waveforms can be defined:

- Sinusoidal
- Triangular
- Square
- Sawtooth
- Pulse
- Staircase
- Fast autofocus waveform
- Any custom vector

By default, the trigger signals are configured as outputs. The trigger signal goes HIGH (3.3V, max. 5 mA) at phase 0° of the selected waveform and goes LOW in the middle of the period. For pulse patterns, the trigger signal reflects the duty cycle.





No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

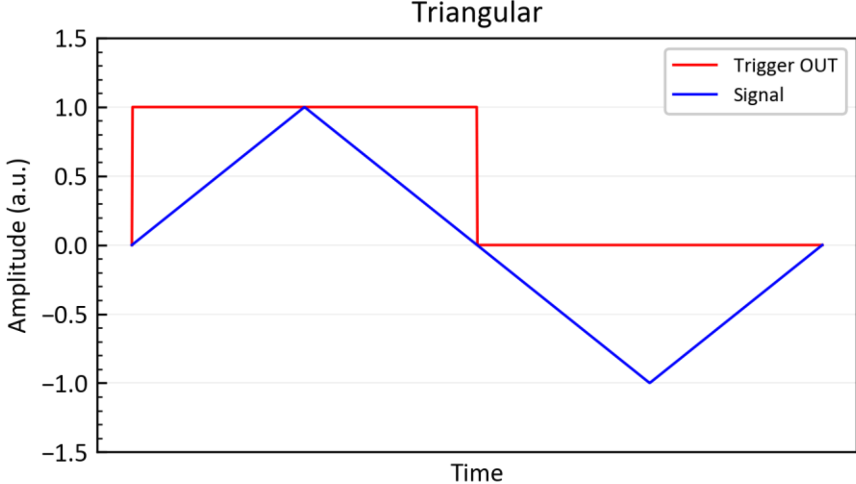
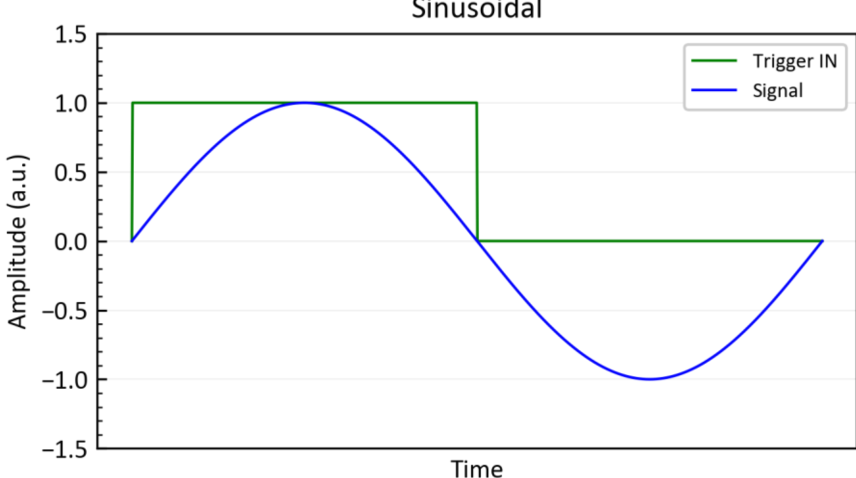


Figure 36: Different waveforms overlapped with the corresponding Trigger OUT signal

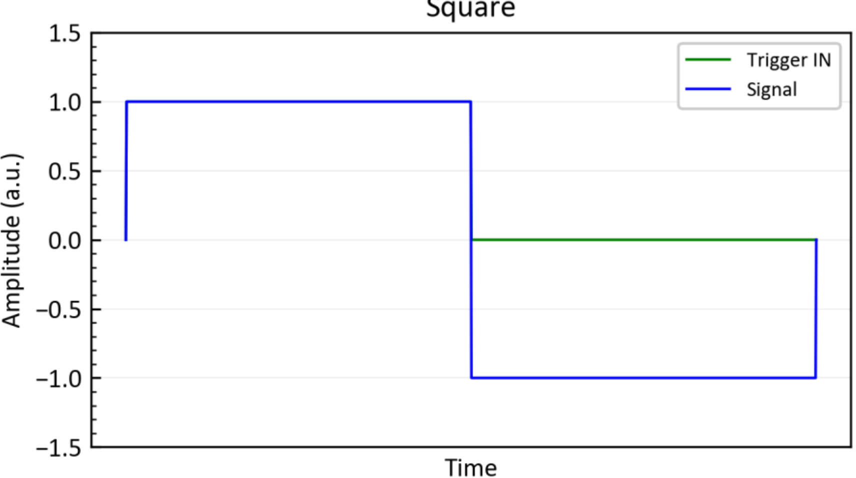
When configured as inputs, the trigger signals can be used to start the built-in signal generator or a preconfigured custom vector. When the trigger input signal goes HIGH (max 3.3V), the selected waveform starts off at phase 0°.



No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune



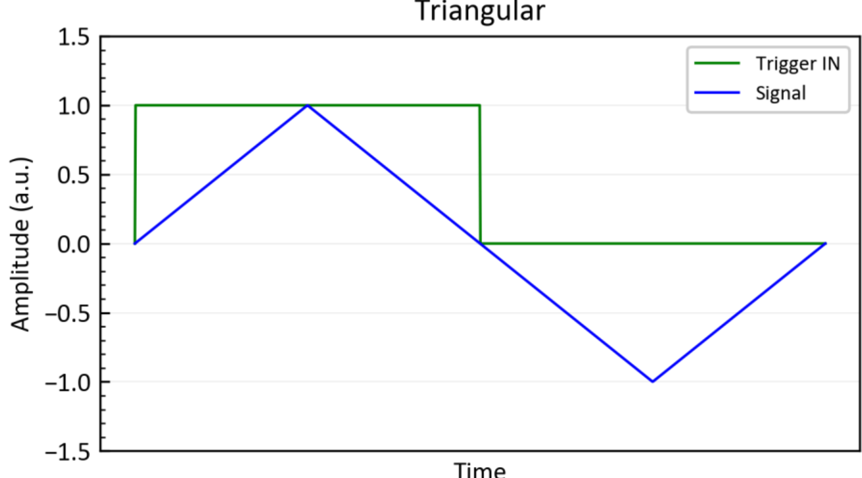


Figure 37: Different waveforms overlapped with an exemplary Trigger IN signal

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

11. Troubleshooting and FAQs

11.1. Status LEDs

The ICC-1C has two status LEDs on its from panel to indicate potential errors/warnings – one is showing the general status of the whole controller (Main Status) and the other one is showing issues related to the output channel (Out Status).

LED	Color	Legend
Main Status	Red	Power on, no connection
	Orange[1]	Operation OK (possible error)
	Green	Operation OK
Out Status	Red	Lens error
	Green	Lens detected; operation OK
	[1] mixed from red and green LEDs. Depending on the viewing angle, one color might be more dominant.	

Both indicators are two-color LEDs (red/green) mounted in the same package, and the firmware is turning on either one of them (Red or Green) or mixing both of them at the same time (Orange). For the Orange color, one of the consisting colors might be more dominant depending on the view angle.

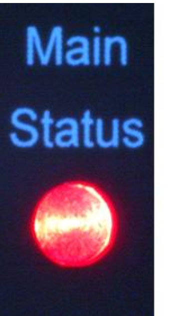


Figure 38: Pictures of the Main Status LED during various operation modes (Red, Orange, Green)

In normal operation, **both LEDs should be glowing Green**, when the device is powered, but below is a list of the most common error cases:

- **Both LEDs OFF** – firmware issue. Potentially can be solved by the “Device recovery” feature in Cockpit or in some cases by updating the firmware.
- **Main status LED is Red** – the device is unable to initiate communication. Check the communication cables (USB, Ethernet).
- **Main status LED is Orange, Out Status is OFF** – lens not connected/not detected. Check the Hirose or the flex cable[1].
- **Out Status is Red** – Lens error (for example overcurrent detected)

Further information can be obtained about the given errors/warnings by the Optotune Cockpit software or by reading the devices status registers.

[1] Always power the device off before handling the flex cable, otherwise EEPORM corruption and/or permanent damage to lens or controller may occur.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

11.2. Diagnostics in Optotune Cockpit

If the device can be connected to a host computer via USB or Ethernet, the Optotune Cockpit software can provide multiple diagnostic features. If some error/warning is present on the controller, a red or orange triangle can show up next to the device's name in the menu under the Electronics section. Clicking on this triangle opens a description of this error/warning and also allows the user to clear the given message.

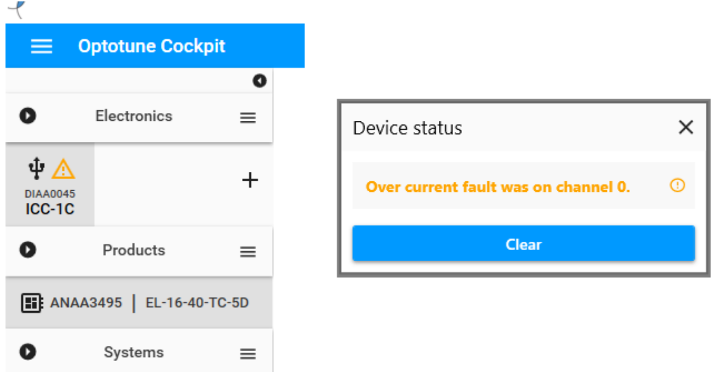


Figure 39: Example of a warning shown in Cockpit

Additional information can be collected by inspecting the data provided by the various **Temperature readout**, the **Board status** and **Channel status** widgets.

The **Board EEPROM** widget can provide useful information about the settings stored in the controller, while the **EEPROM read/write** widget allows to check the data related to the connected lens.

The **Board logger** widget allows to monitor some of the most important values in the controller and also any custom value (register address) over time and export the data into a .csv file.

11.3. Diagnostics with the Device Manager and Terminal software

If the controller cannot be recognized by Cockpit, it might be still possible to run some diagnostics using the **Windows Device Manager** and some Serial terminal software, like **Termite** or **Putty**. Normally the ICC-1C should show up in the Device Manager as a **USB Serial Device** under the COM Ports section, but in case of firmware issues it can be also present itself as a DFU in FS Mode device or some unrecognized USB device.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

Copyright © 2025 Optotune

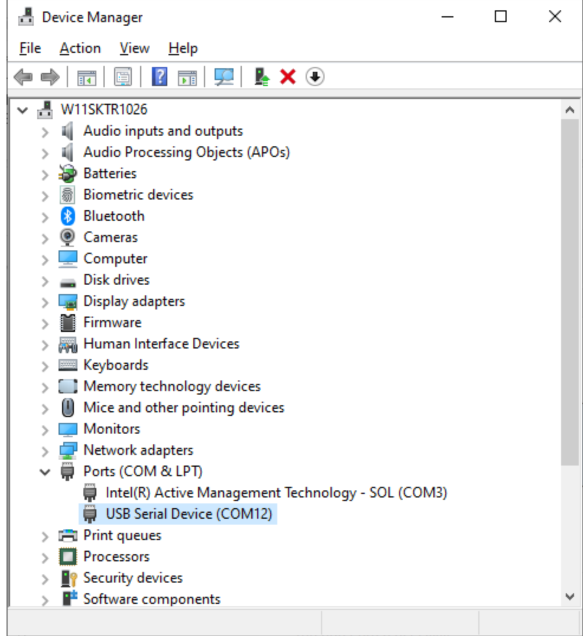


Figure 40: ICC-1C showing up in the Device Manager

If the device can be detected as a USB Serial Device, the user can connect to it by using a terminal software and do some basic debugging. The settings for the software should be the following:

- **COM Port:** The same COM port as the one seen in the Device Manager.
- **Baud Rate:** Any (should be auto-detected by the device, but for debugging purposes Baud rates like 57600 or lower is recommended).
- **Data Bits:** 8
- **Stop Bits:** 1
- **Parity:** None (N)

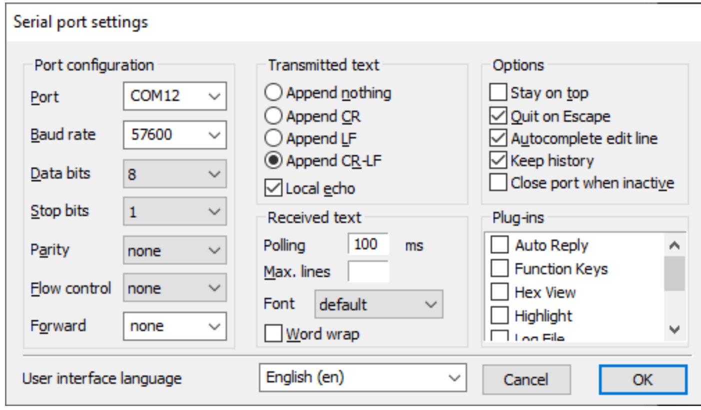


Figure 41: Connection settings for the ICC-1C in Termite

When connected to the COM port, the ICC-1C should accept any simple mode command described in the UART Communication Protocol section of this document. The most useful commands for diagnostics are the following:

- **START[CR][LF]** – Answers “OK” if controller is ready to use and a device is detected. Otherwise “ERROR” is received.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

- **STATUS[CR][LF]** – The controller answers with a status encoded within 4 bytes of information (for example “0x00015000[CR][LF]”), which directly represents the data in the ICC-1C status register in the firmware (register address 0x1007).
- **RESET[CR][LF]** – Restarts the controller’s firmware. No answer is received after this command.
- **GOTODFU[CR][LF]** – Starts the controller’s bootloader for firmware update. No answer is received after this command, but if successful the device should change into a “DFU in FS Mode” device in the Device Manager and can be firmware can be recovered or updated in Cockpit.

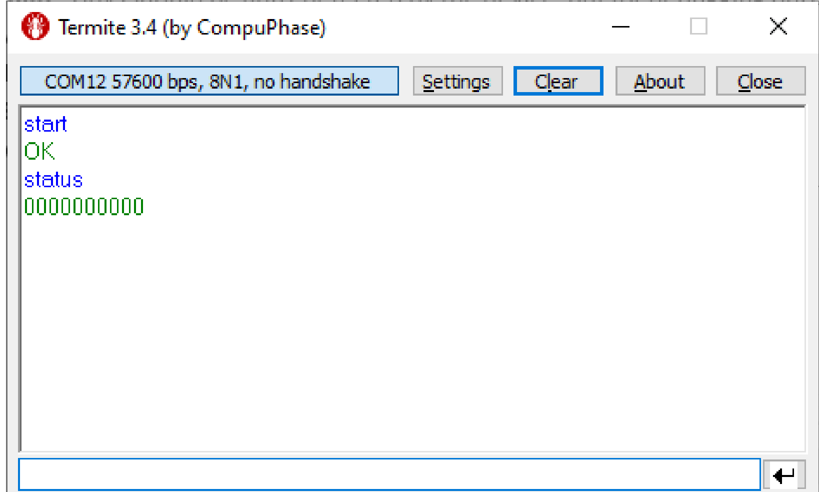
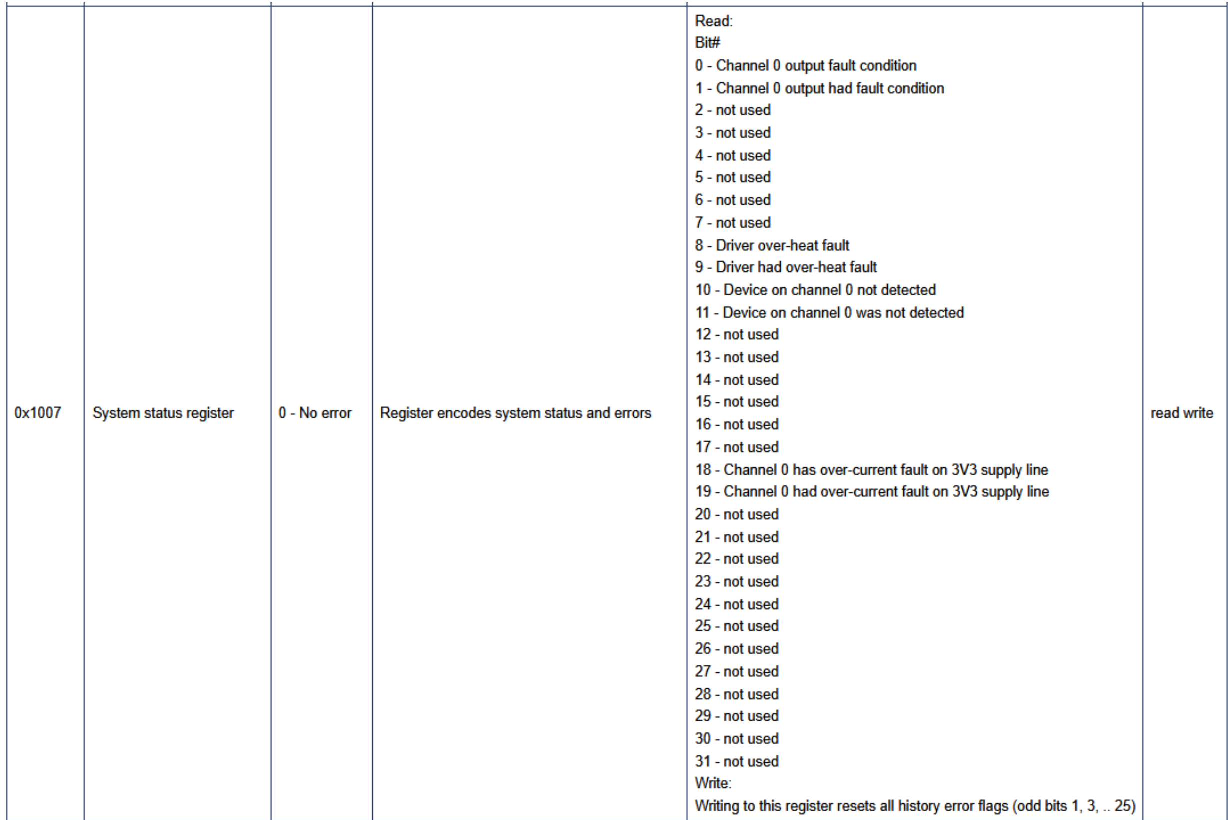


Figure 42: Communicating with the ICC-1C in Termite



0x1007	System status register	0 - No error	Register encodes system status and errors	Read: Bit# 0 - Channel 0 output fault condition 1 - Channel 0 output had fault condition 2 - not used 3 - not used 4 - not used 5 - not used 6 - not used 7 - not used 8 - Driver over-heat fault 9 - Driver had over-heat fault 10 - Device on channel 0 not detected 11 - Device on channel 0 was not detected 12 - not used 13 - not used 14 - not used 15 - not used 16 - not used 17 - not used 18 - Channel 0 has over-current fault on 3V3 supply line 19 - Channel 0 had over-current fault on 3V3 supply line 20 - not used 21 - not used 22 - not used 23 - not used 24 - not used 25 - not used 26 - not used 27 - not used 28 - not used 29 - not used 30 - not used 31 - not used Write: Writing to this register resets all history error flags (odd bits 1, 3, .. 25)	read write

Figure 43: The ICC-1C’s firmware status register (reg. address 0x1007)

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Copyright © 2025 Optotune

11.4. FAQs

Question 1:

The ICC-1C is unable to detect the connected lens or it does not recognize the lens type. What can I do?

Answer 1:

Check the device status register via the Cockpit software or terminal. Also check if the “Autodetect” feature is enabled on the controller. Some lenses do not have EEPROM, so they show up as Unknown devices and can be controlled only in current mode.

Question 2:

My lens has EEPROM, but I still can’t use Focal power mode. What can I do?

Answer 2:

If the lens has EEPROM (check the datasheet), but it’s not accessible, it may indicate data corruption. To restore the data via the Cockpit software, please contact Optotune’s support team for the original lens EEPROM data and further instructions.

Question 3:

The ICC-1C stopped working after an unsuccessful (for example unintentionally interrupted) firmware update and is not recognized by Optotune Cockpit any longer. What can I do?

Answer 3:

In some cases it may be possible to restore the device’s firmware by opening the menu in the upper left corner of our Cockpit software and select “Device recovery”. Select the “Electronic type” and the device you want to restore, and click “Recover”. This solution usually works if the device is stuck in Device Firmware Upgrade (DFU) mode.

If there is no selectable device, then a full hardware reset might still be possible – please contact Optotune’s support team further instructions about this.

Question 4:

I’m unable to change the Board EEPROM settings (for example Ethernet settings or I2C address, etc.). What is the issue?

Answer 4:

This may indicate that the controller’s Board EEPROM data got corrupted. It can be easily verified by opening the Board EEPROM widget, and scrolling down to the “Board Parsing Result” field – if it says “NOK”, please contact Optotune’s support team for the original EEPROM data for the given controller and further instructions.

No representation or warranty, either expressed or implied, is made as to the reliability, completeness or accuracy of this publication.

Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com