optotune
shaping the future of optics

ECC-1C

Embedded Current Controller Manual

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

The ECC-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 EMBEDDED 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 EMBEDDED 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.

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

optotune
shaping the future of optics

Contents

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

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

1. Overview

Optotune's ECC-1C module enables seamless integration of liquid lens control directly from cameras or embedded systems, offering a streamlined and compact solution for rapid deployment in optical applications.

Key Features:

- **Direct System Integration**: Connects effortlessly to cameras and embedded systems for precise lens control. Comes with an adapter to mount to 47 mm diameter Optotune lenses and a variety of third-party lens modules. When ordering from Optotune check for the "-E" option to get the lens with embedded controller.
- **High-Precision Current Control**: Adjustable current range from -300 mA to +300 mA with fine 80 µA resolution.
- **Communication Interfaces:**
  - **UART** and **I2C** with automatic protocol detection.
  - **Analog Voltage Input control** ranging from 0 to 10 V.
  - **GPIO Trigger** for event-based control. Customizable waveforms can be run upon trigger input. Can also be configured as trigger output.
- **Advanced Lens Calibration and Compensation**: Enables readout of calibration data and temperature for dynamic adjustment via "Focal Power Mode."
- **Smart Step Feature** to enable faster lens settling time by applying a preconditioned coil current waveform
- **User-Friendly Control Options**:
  - Compatible with the **Optotune Cockpit GUI** for UART based control.
  - Includes support for a USB-to-UART cable for enhanced connectivity.
- **Developer Support**: Software Development Kits (SDKs) available for **Python** and **C#**.
- **Compliance**: Fully certified with **RoHS**, **REACH**, and **CE** standards.

Key Advantages Over External Drivers:

- **Compact Design**: Reduces system size and complexity.
- **Cost Efficiency**: Eliminates the need for an additional external driver.
- **Simplified Integration**: Minimizes cabling requirements with fewer interfaces and connections.
- **Streamlined Development**: Utilizes a single SDK, rather than managing two separate ones, for faster and simpler implementation.
- **Unlocks Advanced Features**:
  - **Autofocus**: Enables dynamic focus adjustments for enhanced image clarity.
  - **Extended Depth of Field (EDOF)**: Improves focus across a wider range of depths.
  - **Depth from Focus**: Facilitates depth mapping based on focus variation.

Further detailed specifications can be found in ECC-1C's datasheet.

To visit Optotune's Download Center (Firmware, Software, Zemax and CAD files) registration is required.

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

2. Hardware overview

2.1. Package contents and description

The ECC-1C can be used with Optotune´s EL-16-40 or other Optotune lenses, as well as various ELMs (electrical lens modules). It can be ordered separately, together with an Optotune lens (-E option), or as a PCBA.

ECC-1C adapter kit (P/N 150-347-00)

The adapter kit allows for the addition of ECC-1C to EL-16-40, other Optotune threaded lenses and many ELMs (electrical lens modules. The kit consists of:

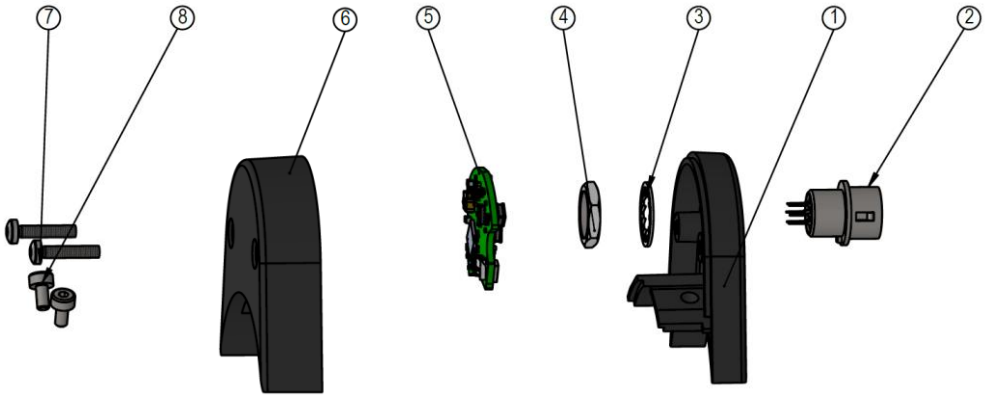


Figure 1: Assembly of ECC-1C

Position	Quantity	Part number	Description
1	1	134-606-00	Housing bottom
2	1	132-281-01	HR10G-7R-6SB (73) connector
3	1	132-281-02	HR10G-7R-6P washer
4	1	132-281-03	HR10G-7R-6P nut
5	1	146-399-00	ECC-1C PCBA
6	1	134-607-00	Housing top
7	2	134-658-00	Screw, K20x10 BN20138
8	2	134-657-00	Screw, M2x4, DIN912 BN610

(Items 1-5 are provided pre-assembled, the rest in plastic zip bags)

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

ECC-1C with lens (EL-16-40)

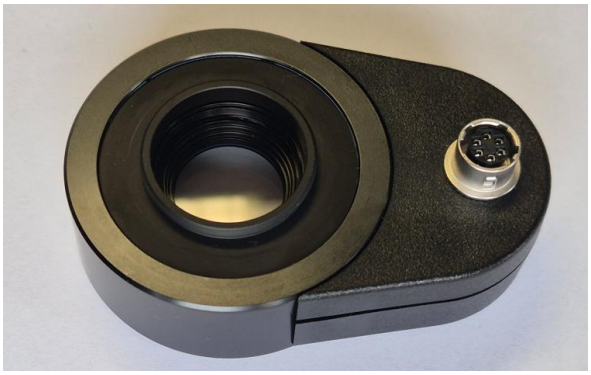


Figure 2: EL-16-40-TC-VIS-5D-C-E

Example: Product name EL-16-40-TC-VIS-5D-C-E contains -> Consists of Electrically focus tunable lens with embedded controller Industrial housing with 6-pin Hirose connector, C-mount threads (male top & female bottom), VIS coated cover glasses (420-950nm) and Focal power range: -2/+3 Diopters

ECC-1C PCBA

Figure 3: ECC-1C PCBA version

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

Optotune provides two cabling options for the ECC-1C:

1. 150-349-00: USB to UART cable, Hirose connectors, 1m. Contains USB type-A integrated UART converter (Silicon Labs CP2102, drivers for manual installation are available here)
2. 152-219-00: CAB-6-100-M-OE - Hirose to open-ended wire cable, 1 m (pinout in Figure 1)

Pin	Cable Color	Function
1	Black/thin	GPIO trigger
2	Red/thin	Analog In
3	Grey/thin	Tx/SCL
4	Green/thin	Rx/SDA
5	Blue/thick	GND
6	Brown/thick	VCC

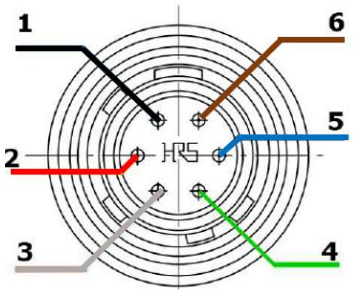
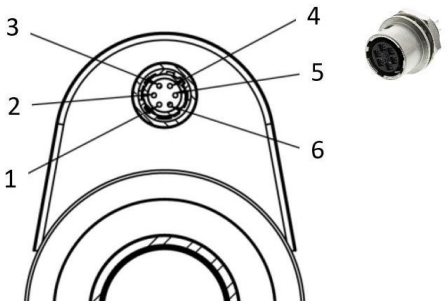


Figure 4: Pinout and color code of Hirose to open-ended wire cable

Figure 5: USB to UART and Hirose to open-ended wire

2.2. Electrical connection

The electrical connection of the ECC-1C consists of a female Hirose connector (HR10G-7R-6SB) mounted on the thermoplastic controller housing:

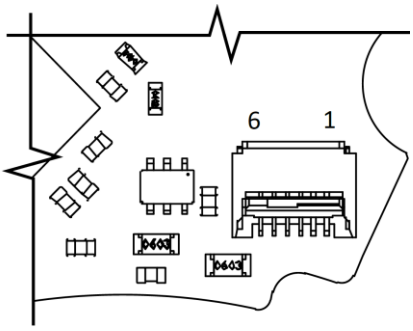


Pin out Hirose connector HR10G-7R-6SB (73)		
Position	**Function**	**Value**
1	GPIO Trigger	3V3 logic
2	Analog In	0-10V
3	UART Tx / I²C SCL	3V3 logic
4	UART Rx / I²C SDA	3V3 logic
5	GND	-
6	VCC input	5-24V (see chapter 2.7)

Figure 6: Connecting to a lens with Hirose connector.

6-pin flex cable connector is the connection between lens and ECC-1C controller. Controller provides current control for lens coil and internal I2C communication with lens' EEPROM and temperature sensor readout. Lens flex cable pins must face upwards when connected.

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



Pin out flex cable connector		
Position	**Function**	**Value**
1	GND	-
2	Lens coil -	-300 to 300 mA
3	Lens coil +	-300 to 300 mA
4	I²C SDA Lens	3.3 V logic
5	I²C SCL Lens	3.3 V logic
6	VCC – Lens supply	3.3 V

Figure 7: Electrical connections of flex cable connector.

The following pictures show the process of connecting a lens with an FPC (Flexible Printed Circuit) connector:

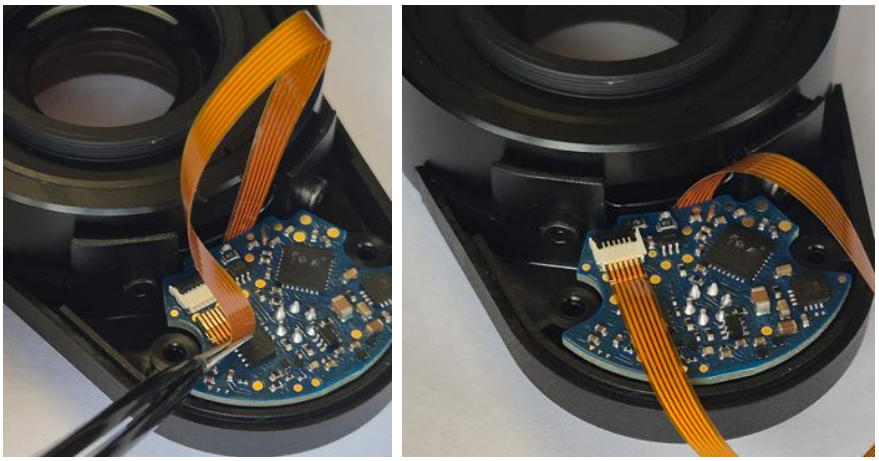
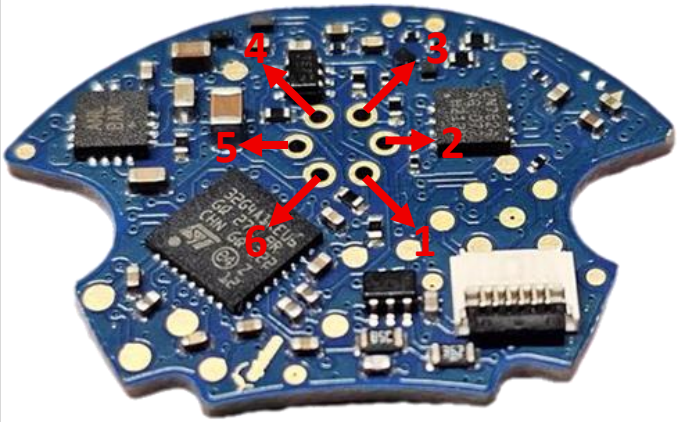


Figure 8: Connecting to a lens with FPC connector

2.3. PCBA mounting - Future OEM

The PCBA version of the board includes all necessary components soldered onto the PCB, enabling full electrical connectivity. When soldering the Hirose connector, pay close attention to the pinout table provided. The table can be seen above. The pins are compact and closely spaced, so ensure precise soldering to avoid shorts or cold joints. Only use recommended components and follow connection guidelines to maintain signal integrity.
Warning: Incorrect soldering or mismatched connections may damage the board or cause malfunction.



Pin out Hirose connector HR10G-7R-6SB (73)		
Position	**Function**	**Value**
1	GPIO Trigger	3V3 logic
2	Analog In	0-10V
3	UART Tx / I²C SCL	3V3 logic
4	UART Rx / I²C SDA	3V3 logic
5	GND	-
6	VCC input	5-24V (see chapter 2.7)

Figure 9: PCBA pinout

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

2.4. Control options

The ECC-1C with a connected lens or ELM can be interfaced with a computer or other device via UART, using a USB-to-UART adapter, provided the host system supports serial communication.

Connection is supported exclusively via a Hirose industrial connector. This connector ensures reliable operation with clearly defined pin positions, allowing for hot plugging even while the controller is powered on.

The ECC-1C is best controlled using the digital interfaces UART or I²C (using pin n.3 and n.4). The respective communication protocol is detected automatically. Both protocols support a pro-mode with a register-based command structure. Additionally, the UART protocol offers a simple mode based on ASCII text commands for easier integration.

Furthermore, the ECC-1C can be pre-configured with a look-up table for analog 0–10 V control (using pin n.2), or with vector-based waveforms that can be triggered via the GPIO pin(using pin n.1), enabling flexible and precise control options. (See Chapter **9. Input/Output trigger signals** for trigger options)

Figure 10: ECC-1C connected to USB interface

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

2.5. Mounting

2.5.1 Mechanical drawing

The ECC-1C is available in a compact reinforced polymer casing that can be mounted on a standardized EL-16-40-TC or other ELs. For more mounting details and dimensions, see the mechanical drawing below:

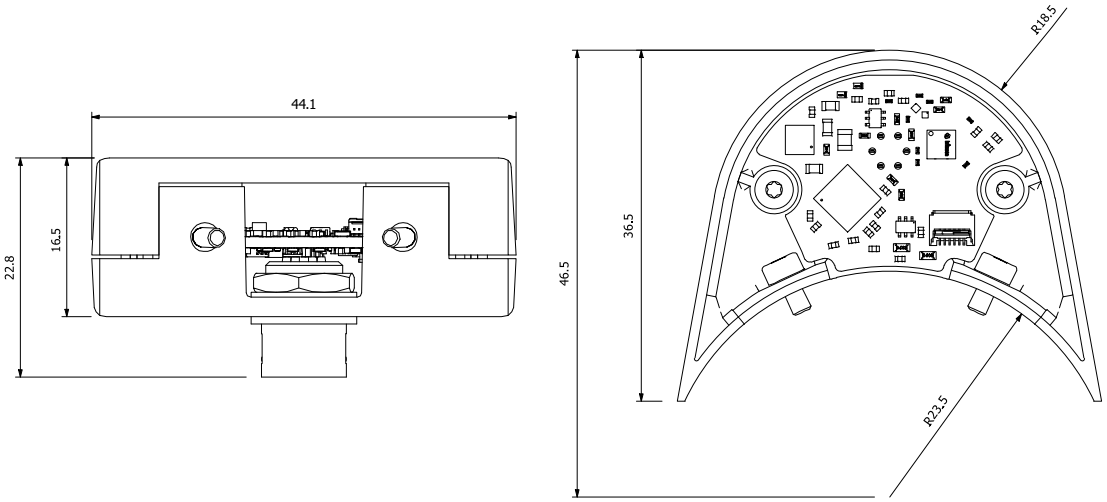


Figure 11: Mechanical drawing of ECC-1C adaptor

Alternatively, a PCBA version is also available. For more mounting details and dimensions, see the mechanical drawing below:

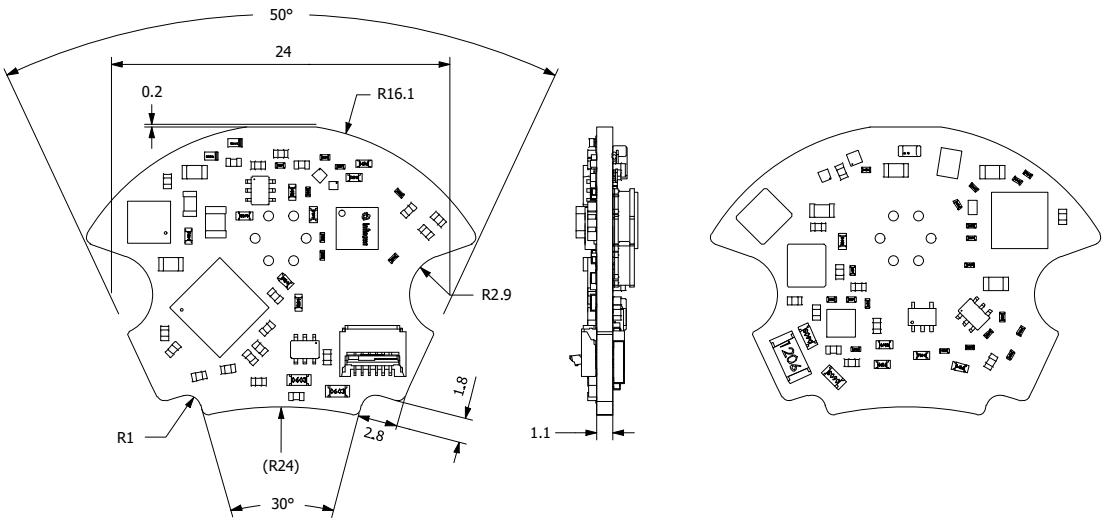


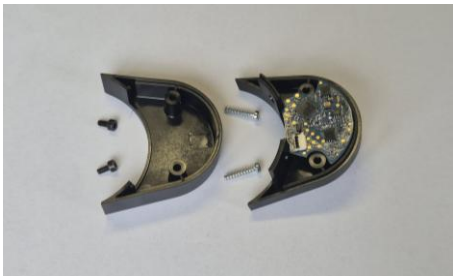
Figure 12: Mechanical drawing of ECC-1C PCBA version

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

2.5.2 ECC-1C adapter kit install on a standard EL-16-40-TC

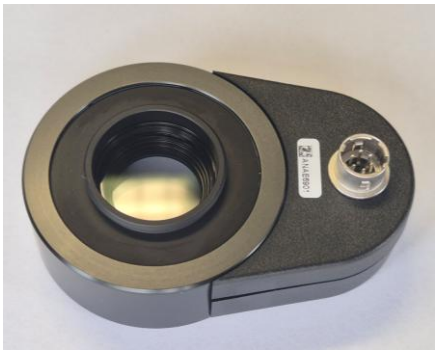
Tools

1.5mm Allen Key and small Torx

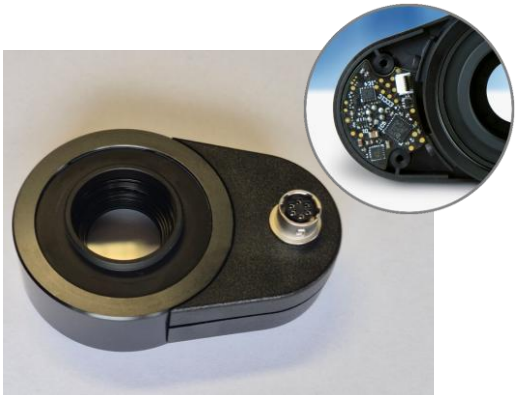


ECC-1C adapter kit (P/N 150-347-00)

Overview



Before: EL-16-40 with FPC-to-Hirose Adaptor
Note that the Hirose connector is male



After: EL-16-40 with ECC-1C and Hirose connector
Note that the Hirose connector is female

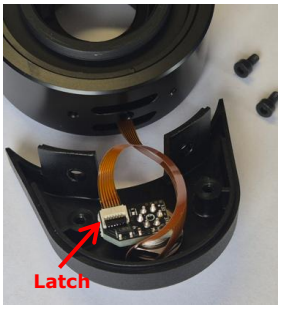
1a. Adapter removal

Unscrew the two screws on the back of the adapter

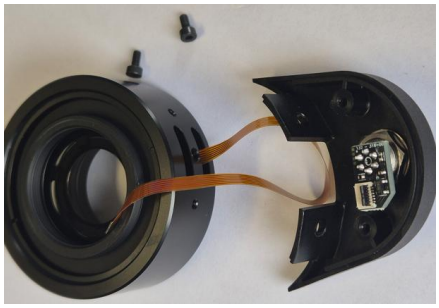
Remove the adapter by lifting it from the side

1b. Adapter removal

Unscrew the bottom adapter part with 1.5mm Allen key



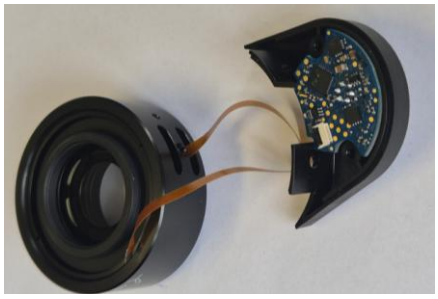
Lift the black latch of the Molex connector to upright position



Remove the flex cable

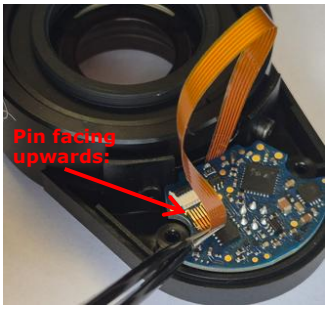
Optotune Switzerland AG | Bernstrasse 388 | CH-8953 Dietikon | Switzerland
Phone +41 58 856 3000 | www.optotune.com | sales@optotune.com

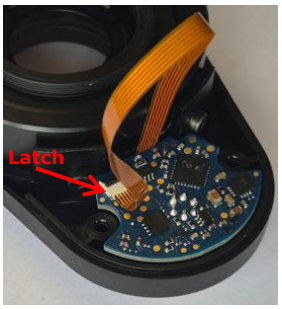
2a. ECC-1C adapter kit installation

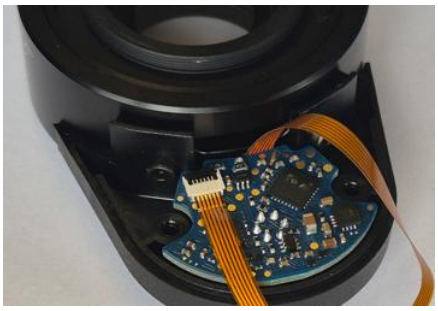


Screw the bottom adapter part
containing ECC-1C with 1.5mm Allen key

2b. ECC-1C adapter kit installation







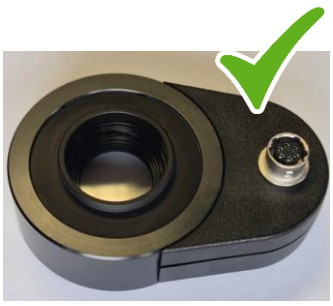
Insert the flex cable in the molex adapter with the
latch being lifed in upright position, then fasten
the latch by closing it in horizontal position

2c. ECC-1C adapter kit installation

Place the adapter by
pushing it down from
the side

Screw back the two screws on the
back of the adapter.

WARNING: MAKE SURE NOT TO
COVER THE HOLES WITH THE FLEX
CABLE. OTHERWISE, CABLE WILL BE
DAMAGED BY THE SCREWS



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

2.6. Thermal Management

The ECC-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 table below.

Thermal specifications

Operating temperature	0 to 65	°C
Storage temperature	-40 to 85	°C

2.7. Available drive current vs. temperature

The guaranteed output current from the ECC-1C driver is influenced by the lens type, input voltage, and ambient temperature. As shown in the graphs:

- At elevated ambient temperatures, the output current is limited due to the maximum operating temperature of the driver. At 65°C, the current drops sharply regardless of the input voltage
- The **input voltage directly affects the maximum achievable output current**:
  - At **5 V**, the output current is significantly lower (e.g., **250 mA**) and begins to drop earlier with rising temperature
  - At **higher voltages (9–24 V)**, the driver can maintain a higher output current (**~250–300 mA**), but this too is clipped as the temperature increases
- **This behaviour is also dependent on the lens used**, as different lenses have different electrical and thermal characteristics. **Additional operating curves for other lenses are available in the ECC-1C datasheet**

This clearly shows **that both ambient temperature and input voltage constrain the current delivered to the lens**, and the **safe operating area narrows** with increasing temperature.

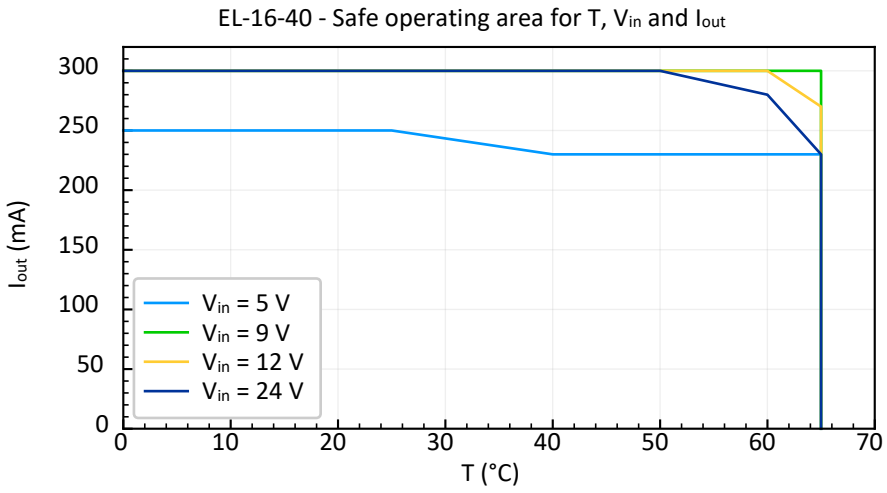


Figure 13: Guaranteed drive current for EL-16-40 lens available from ECC-1C

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

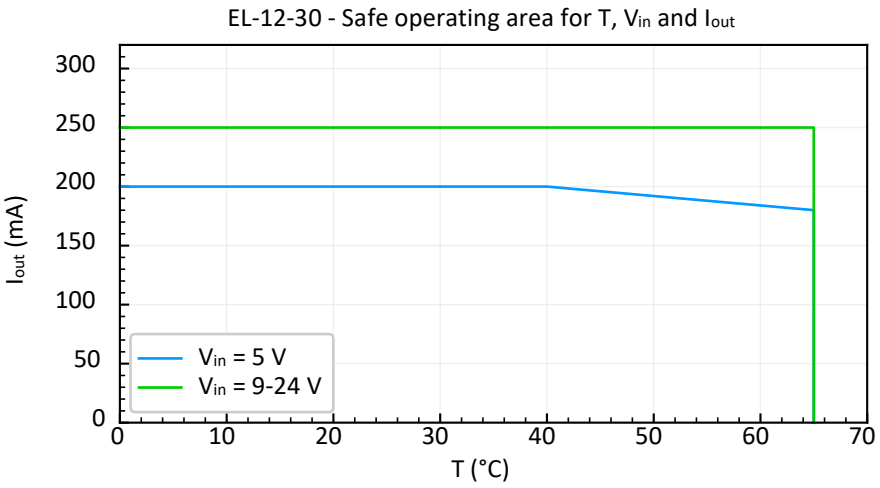


Figure 14: Guaranteed drive current for EL-12-30 lens available from ECC-1C

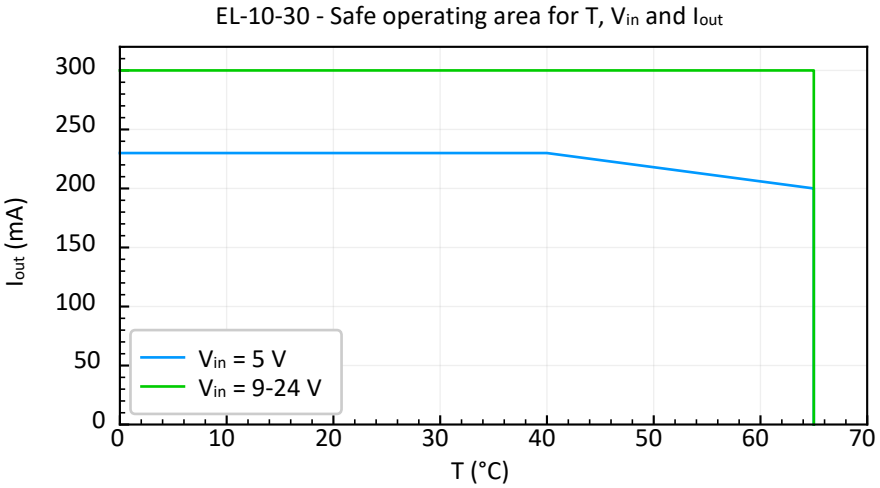


Figure 15: Guaranteed drive current for EL-10-30 lens available from ECC-1C

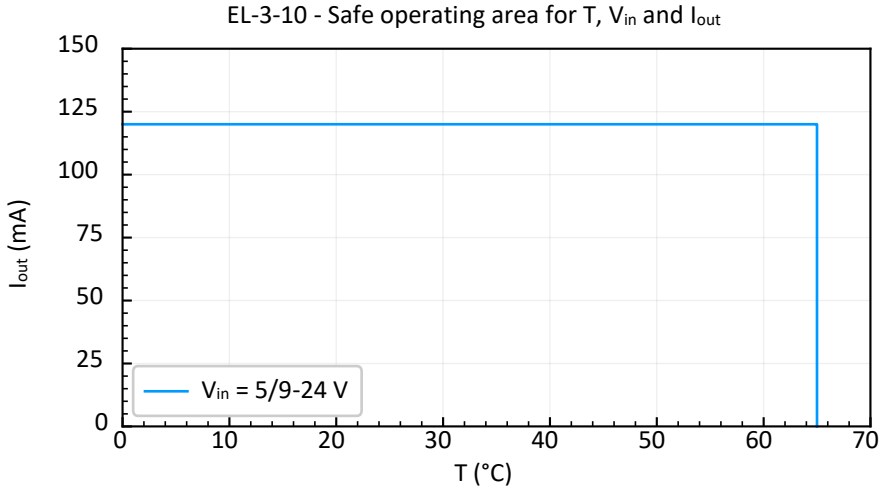


Figure 16: Guaranteed drive current for EL-3-10 lens available from ECC-1C

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

2.8. Do's and Don'ts

Connect the lens with shiny pins facing up. Wrong orientation will cause permanent damage.

The device uses **3.3V logic level** signals. Pins are 5V tolerant. Do not exceed 5.5V, otherwise permanent damage will result. Check pin-out carefully.

Do not plug/unplug the lens when connected to power. This may lead to **EEPROM corruption** and/or **permanent damage** to the lens or controller.

Any handling of these boards is to be minimized. Electrostatic discharge (ESD) may damage the boards, even if the circuit is unpowered.

ECC-1C PCBA: Any mishandling during soldering may void warranty or damage the product. Proceed at your own risk.

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

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

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.

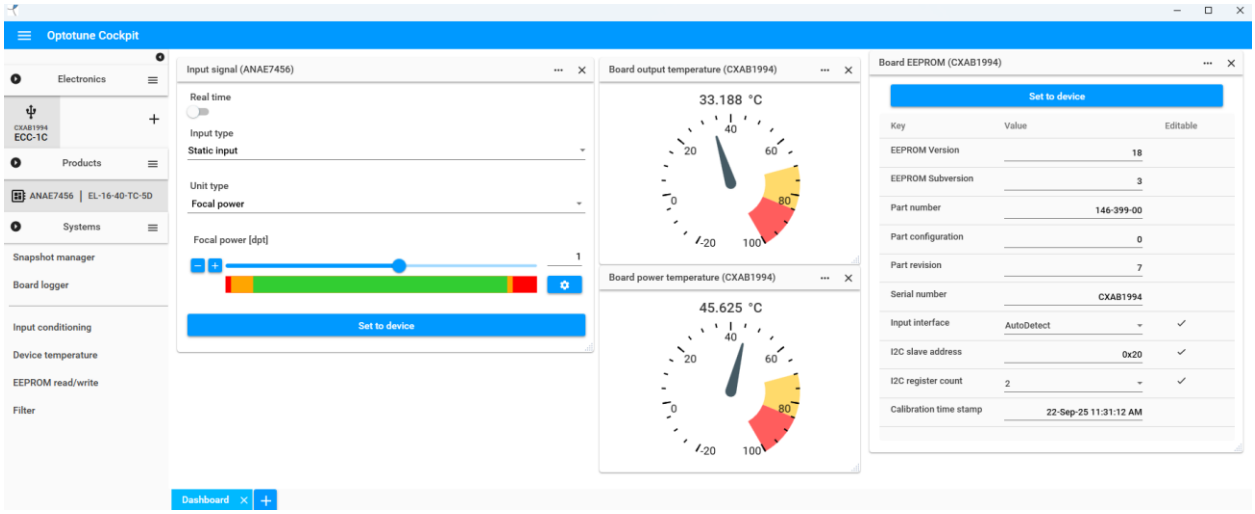


Figure 17: Optotune Cockpit main window overview

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.

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

3.1. Lens-related systems (widgets)

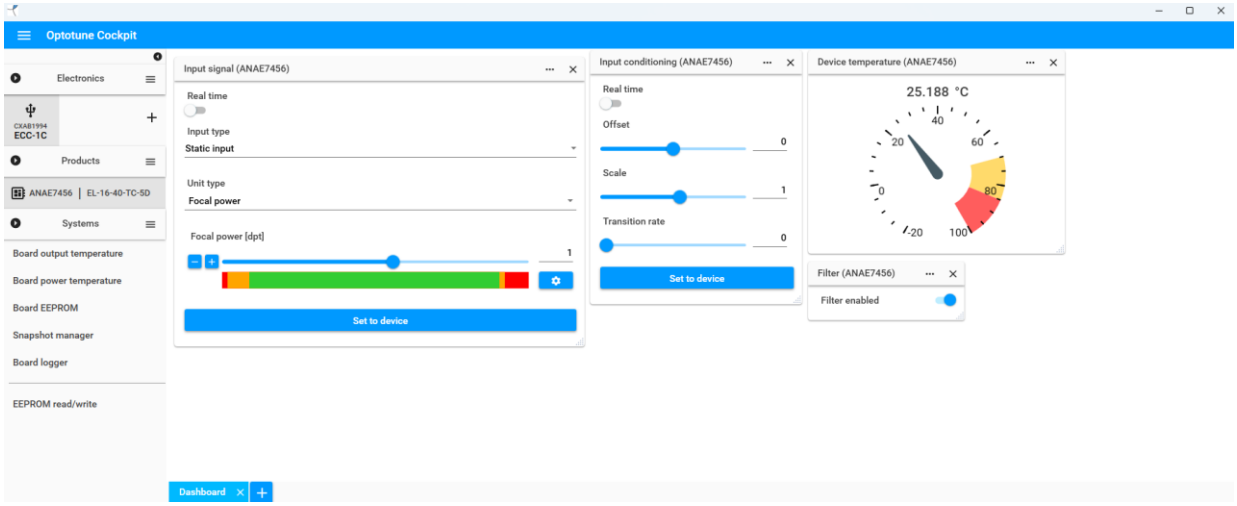


Figure 18: Optotune Cockpit widgets

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.

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

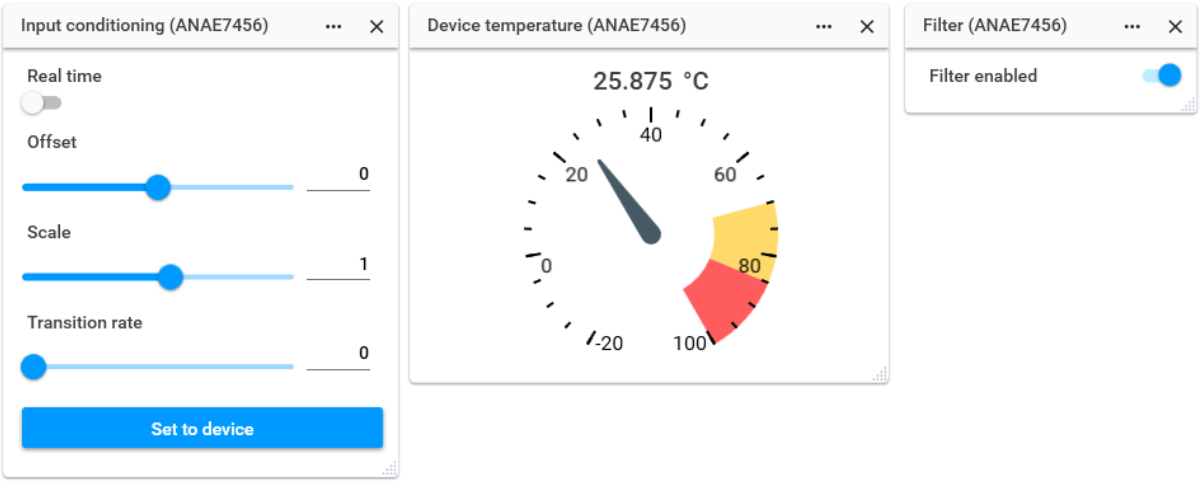


Figure 19: Input conditioning, Lens temperature readout and Smart Step Filter settings widgets

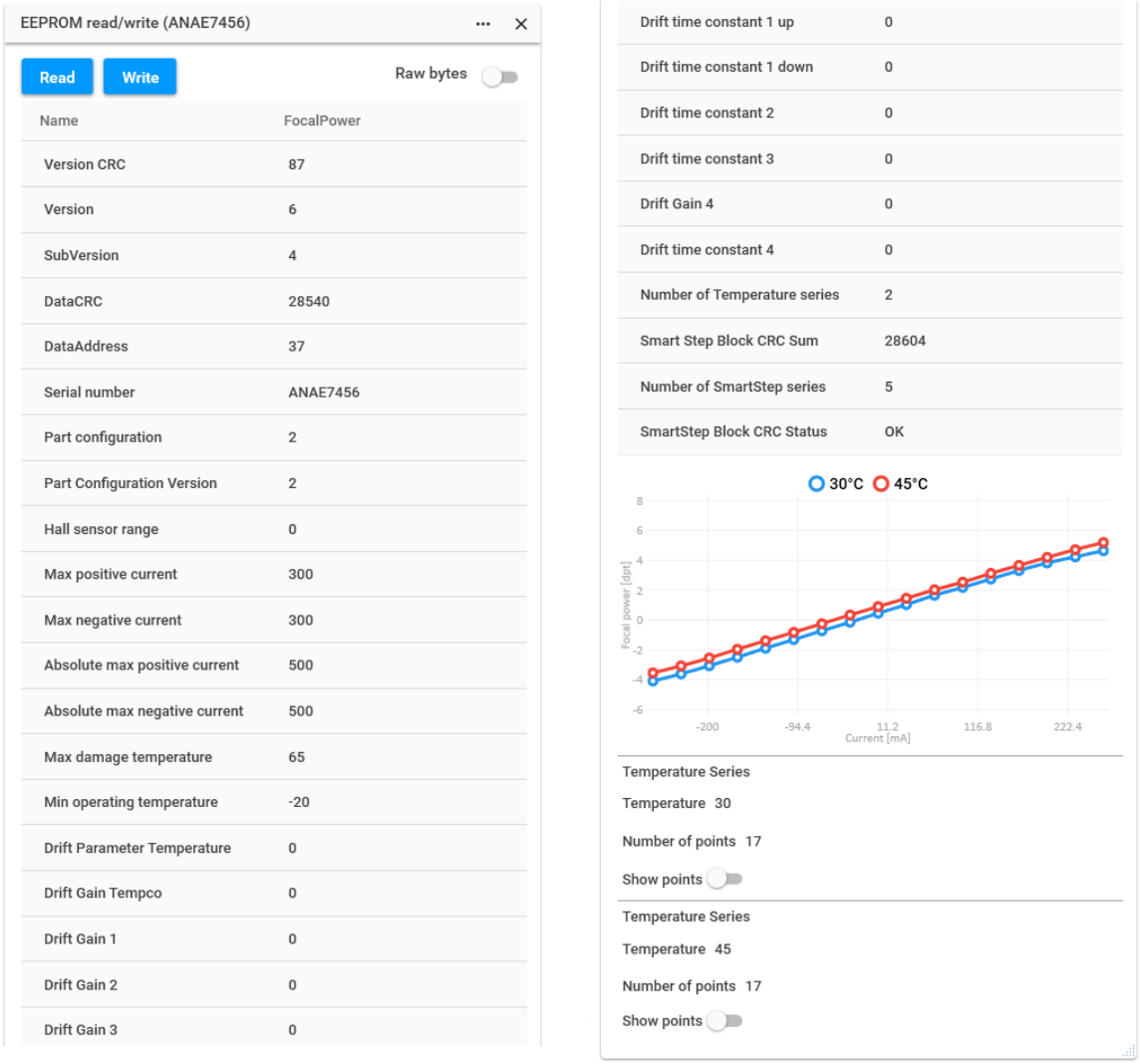


Figure 20: Lens EEPROM read/write widget

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

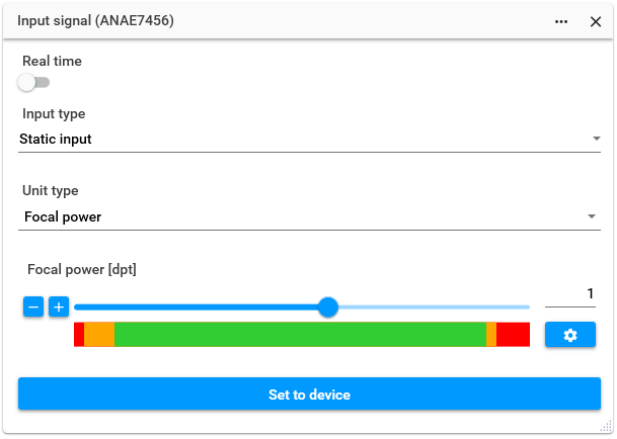


Figure 21: Input signal widget set to static input

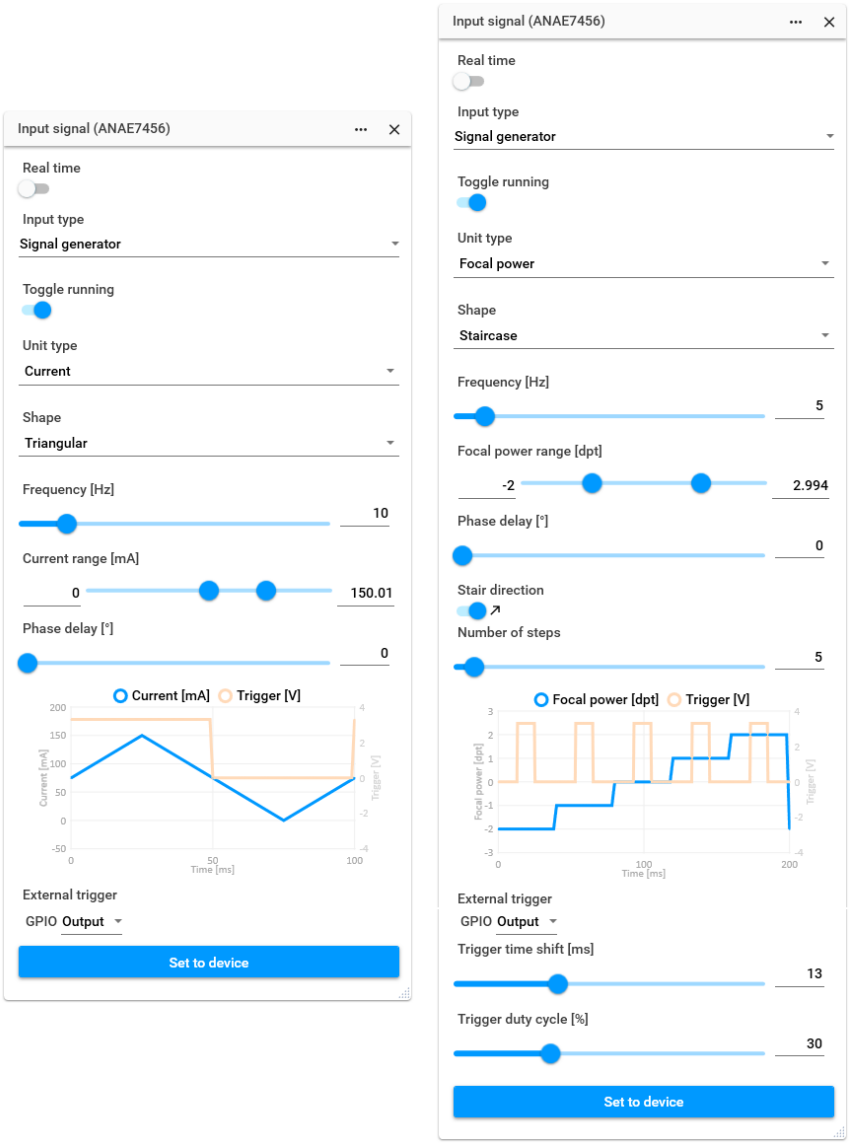


Figure 22: Input signal widget set to various waveforms from the Signal generator

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

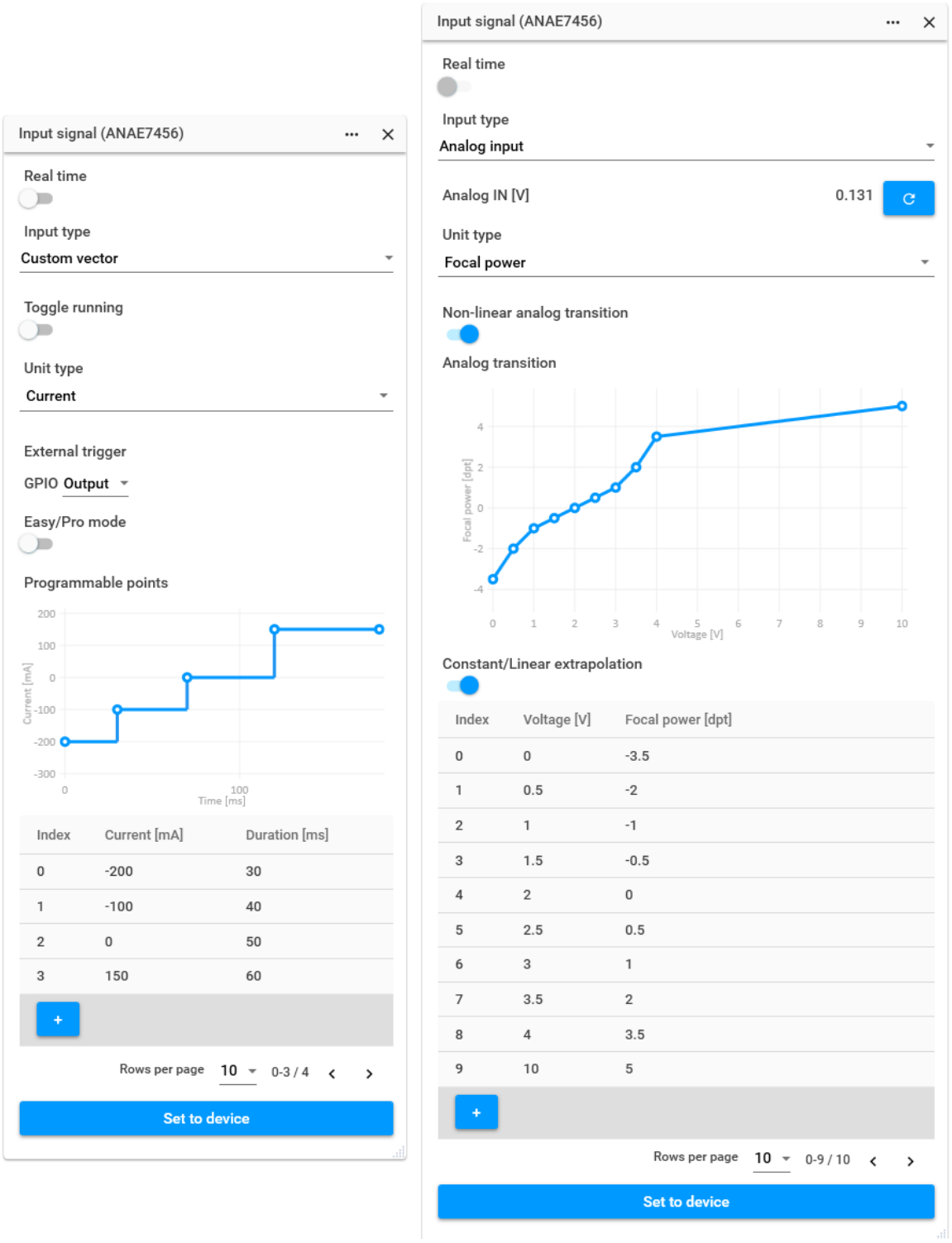


Figure 23: Input signal widget set to Custom vector and Analog input

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

3.2. Controller-related systems (widgets)

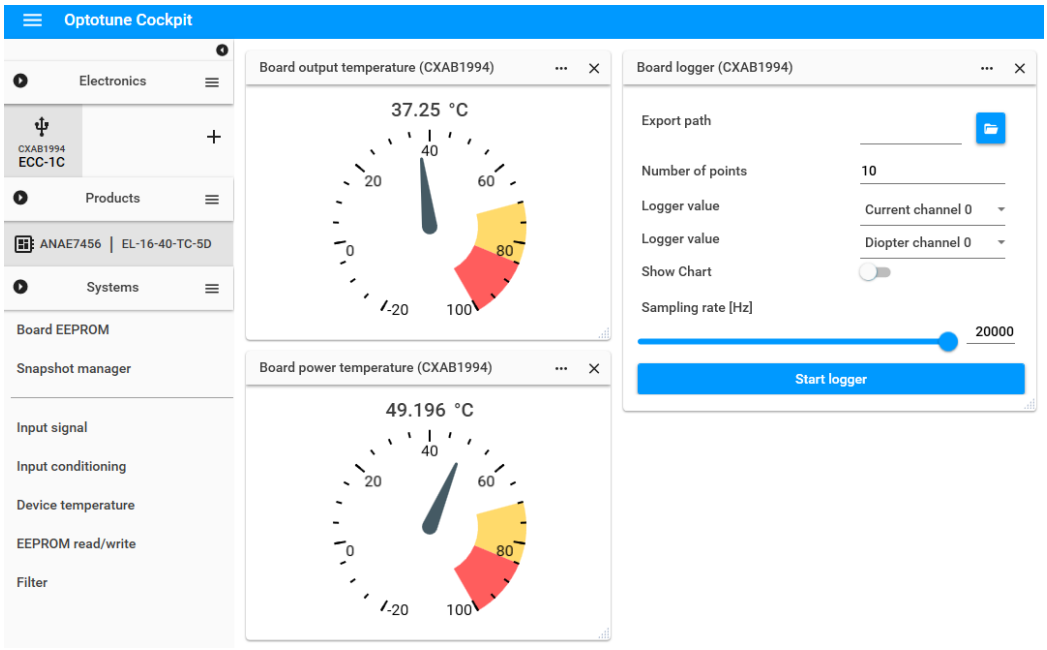


Figure 24: Examples of widgets related to the controller

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

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

Board EEPROM widget, which contains the information stored in the ECC-1Cs memory, such as the device's serial number or the settings required for UART / 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 be configured and saved by the user.

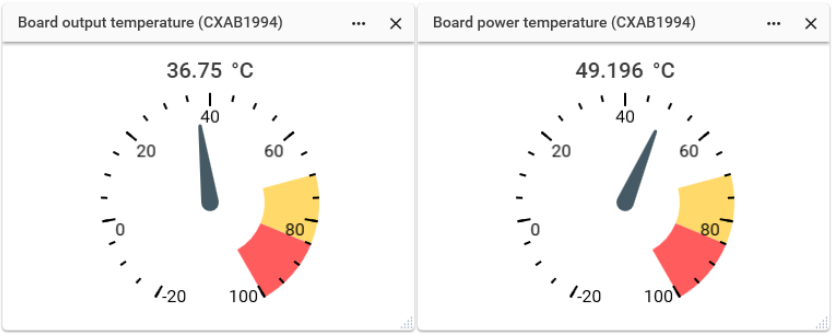


Figure 25: Temperature readout widgets

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

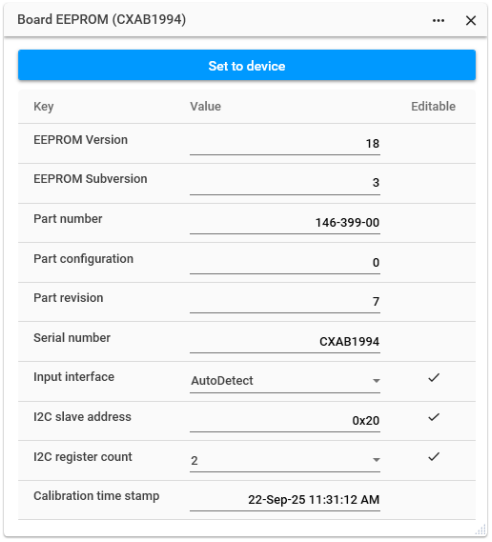
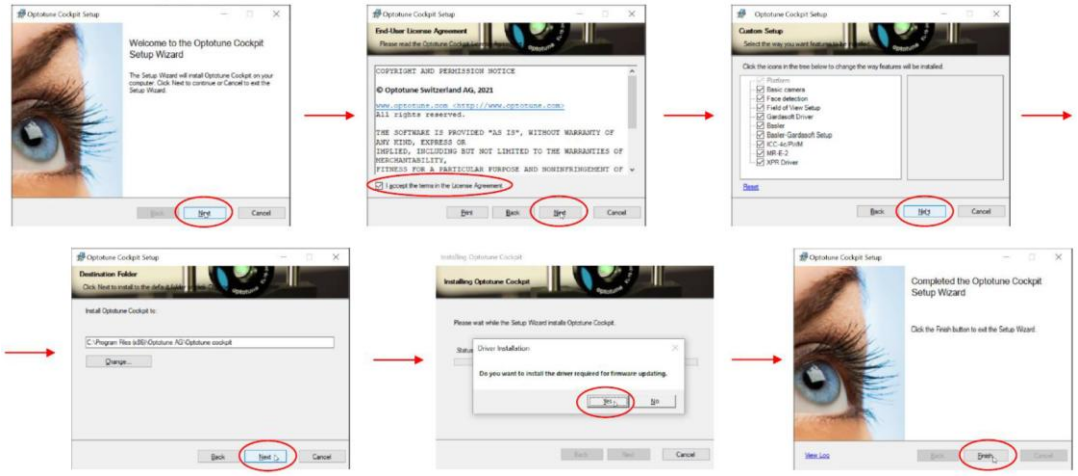


Figure 26: Board EEPROM widget

3.3. How to rewrite EEPROM data using ECC-1C

Step 1: Install Optotune Cockpit software

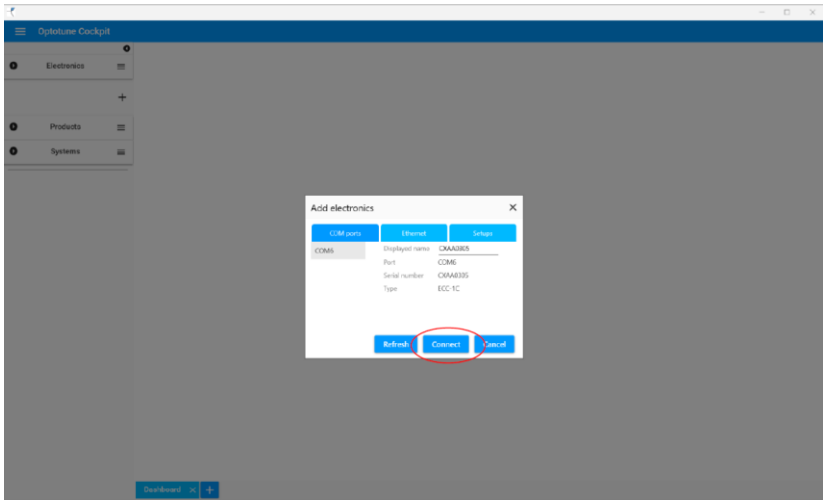
- Download from https://www.optotune.com/software-center
- Follow installation steps:



Step 2: Connect ECC-1C

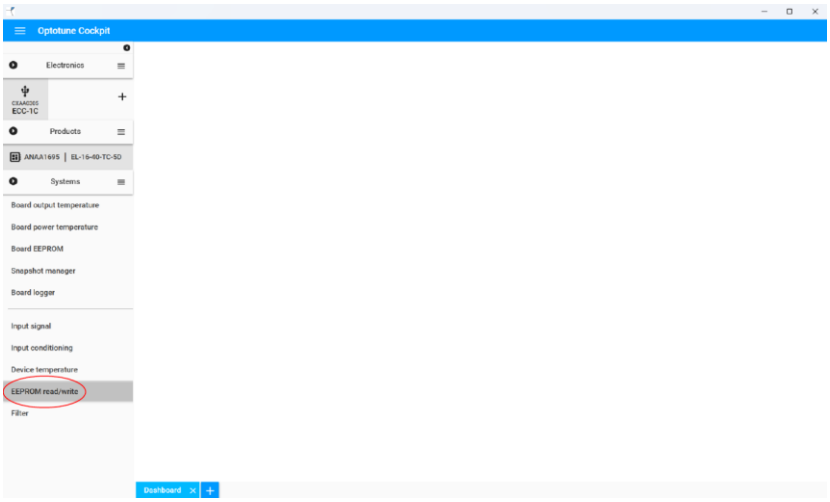
- Connect the lens with E CC-1C to computer using UART-USB cable
- Start OptoCockpit
- Within OptoCockpit connect to ECC-1C

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



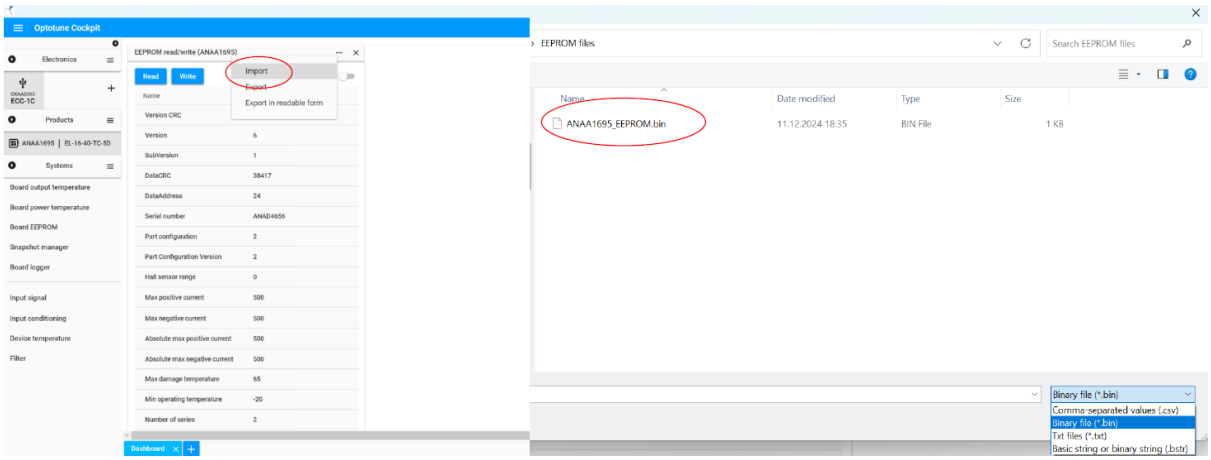
Step 3: EEPROM read/write tab

- Select “EEPROM” read/write” tab



Step 4: Import the EEPROM file

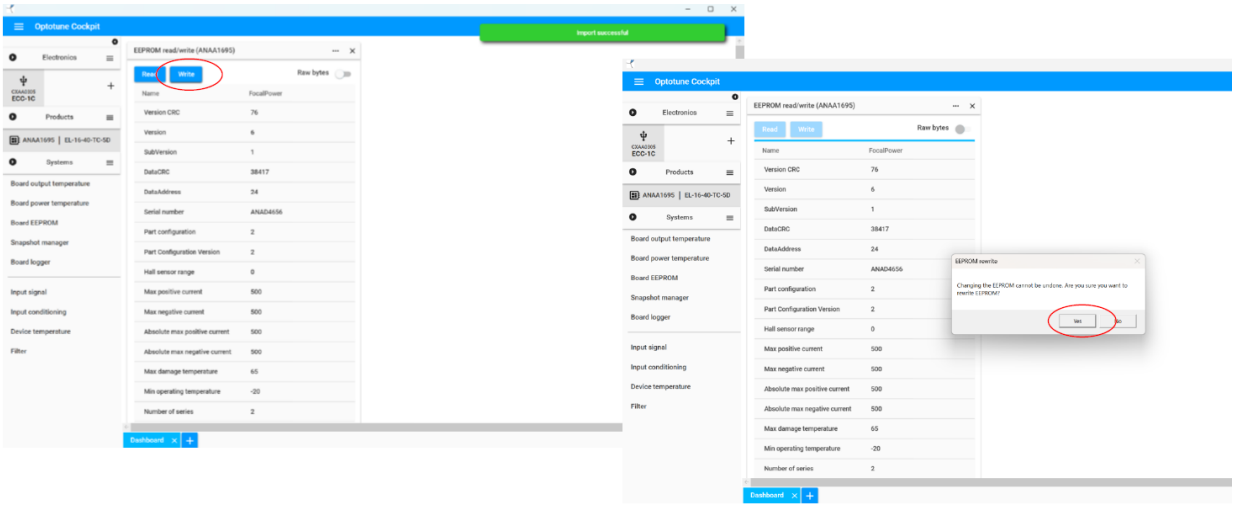
- Click on “…” and then choose “Import”
- Select the file EEPROM file which is provided by Optotune
    - Change the file type depending on the provided EEPROM file (e.g. .bin or .csv)



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

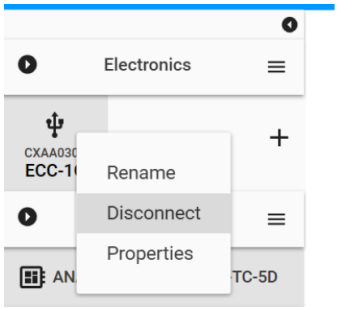
Step 5: Write to EEPROM

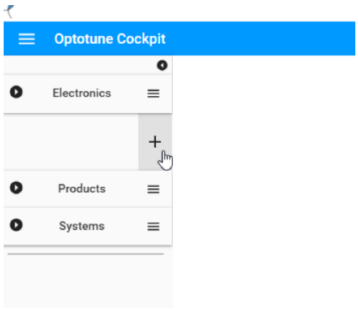
- After successful import, select Write
- Confirm with “Yes”

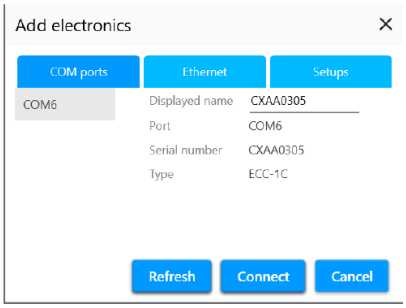


Step 6: Disconnect and power cycle

- Electronics Tab: Right click on connected driver and choose “Disconnect”
- Disconnect the ECC-1C from USB port
- Reconnect to PC and in Optotune Cockpit







Step 7: Validation

- Verify by reading EEPROM content in the EEPROM read/write tab and check serial number of listed lens to be matched with calibration file serial number

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

4. Software development

There are two different software development kits available for the ECC-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 ECC-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).

Describe our SDKs, where to download, how to install, first steps, code examples for download or put them here.

4.2. C# code example

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

namespace Ecc1cExample
{
    internal class Ecc1cExample
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Starting basic ECC-1C example...");
            Ecc1cSdkDeviceController controller;

            while (true)
            {
                // This example could be used also with Icc1c
                // controller = new Icc1cSdkDeviceController();
                controller = new Ecc1cSdkDeviceController();
                Console.WriteLine("What is the COM number of the ECC-1C?");
                var comport = Console.ReadLine();
                if (int.TryParse(comport, out var comInt))
                {
                    //Connecting to driver
                    if (controller.Connect(comInt))
                    {
                        break;
                    }
                    Console.WriteLine("Connection failed.");
                }
            }

            //Set board to PRO mode
```


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

```
35.             controller.ChangeModeToPro();
36.
37.             //Getting board info
38.             Console.WriteLine("Board info");
39.             var serialNumber = controller.GetElectronicSerialNumber();
40.             var fwVersion =
41.                 $"{controller.Status.GetBoardFirmwareMajorVersion()}.{controller.Status.Get-
BoardFirmwareMinorVersion()}.{controller.Status.GetBoardFirmwareRevisionVersion()}";
42.
43.             Console.WriteLine($"Board serial number: {serialNumber}");
44.             Console.WriteLine($"Board firmware version: {fwVersion}");
45.
46.             // In order to drive the connected product, device type of the product has to be
obtained (this is not needed if
47.             // automatic detection is allowed)
48.             var connectedDevice = controller.MiscFeatures.GetDeviceType();
49.             Console.WriteLine();
50.             Console.WriteLine(
51.                 $"Connected device: {Enum.GetName(typeof(EConnectedDeviceType), connectedDe-
vice)}");
52.             Console.WriteLine();
53.
54.             //Supported lens types
55.             var lensesTypes = new[]
56.             {
57.                 EConnectedDeviceType.EL1030C,
58.                 EConnectedDeviceType.EL1030TC,
59.                 EConnectedDeviceType.EL1230TC,
60.                 EConnectedDeviceType.EL1640TC_5D
61.             };
62.
63.
64.             if (!lensesTypes.Contains(connectedDevice))
65.             {
66.                 Console.WriteLine("No lens connected");
67.             }
68.             else
69.             {
70.                 //ECC-1C supports only one channel EChannel.Channel0
71.                 var lensSerialNumber = controller.GetLensSerialNumber();
72.                 Console.WriteLine(
73.                     $"Lens {Enum.GetName(typeof(EConnectedDeviceType), connectedDevice)}
({lensSerialNumber})");
74.
75.                 var lensTemperature = controller.TemperatureManager.GetDeviceTemperature();
76.
77.                 Console.WriteLine($"Lens temperature: {lensTemperature}°C");
78.
79.                 var minCurrent = controller.DeviceEeprom.GetMaxNegativeCurrent();
80.                 var maxCurrent = controller.DeviceEeprom.GetMaxPositiveCurrent();
81.
82.                 Console.WriteLine($"Minimum current is {minCurrent} mA, maximum current is
{maxCurrent} mA");
83.
84.                 Console.WriteLine();
85.                 Console.WriteLine("Setting static current");
86.                 //Setting input system to static input
87.                 controller.InputStage.ChangeActiveSystem(EInputSignalStageSystem.StaticIn-
put);
88.
89.                 for (var current = minCurrent; current <= maxCurrent; current += (maxCurrent
- minCurrent) / 5)
90.                 {
91.                     //Value has to be converted from mA to A
92.                     var currentInA = current / 1000;
93.                     Console.WriteLine($"Current {currentInA} A");
```


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

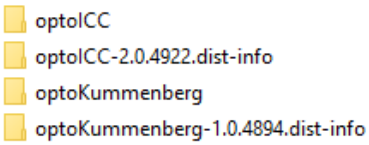
```
94.                     controller.InputStage.StaticValue.SetCurrent(currentInA);
95.                     //Diopter can be set similarly
96.                     //controller.InputStage.StaticValue.SetDiopter(lensChannel, diopter-
InDpt);
97.                     Thread.Sleep(1000);
98.                 }
99.
100.                Console.WriteLine("Setting static current to 0 A");
101.                controller.InputStage.StaticValue.SetCurrent(0);
102.
103.                Console.WriteLine();
104.                Console.WriteLine("Running signal generator");
105.
106.                controller.InputStage.ChangeActiveSystem(EInputSignalStageSystem.SignalGener-
ator);
107.                controller.InputStage.SignalGenerator.SetUnitType(EICC4cPwmUnitType.Current);
108.                controller.InputStage.SignalGenerator.SetShape(ESignalGeneratorShape.Sinusoi-
dal);
109.                controller.InputStage.SignalGenerator.SetAmplitude(0.2f);
110.                controller.InputStage.SignalGenerator.SetFrequency(5);
111.                controller.InputStage.SignalGenerator.SwitchRunning(true);
112.
113.                for (var i = 0; i < 5; i++)
114.                {
115.                    Console.Write(".");
116.                    Thread.Sleep(1000);
117.                }
118.
119.                controller.InputStage.SignalGenerator.SwitchRunning(false);
120.
121.                Console.WriteLine();
122.                Console.WriteLine("Signal generator stopped");
123.
124.                Console.WriteLine();
125.                Console.WriteLine("Device EEPROM");
126.
127.                var eepromVersion =
128.                    $"{controller.DeviceEeprom.GetEepromVersion()}.{controller.De-
viceEeprom.GetEepromSubVersion()}";
129.                Console.WriteLine($"EEPROM version: {eepromVersion}");
130.
131.                var eepromBytes = controller.DeviceEeprom.GetBytes(0, 10);
132.                var eepromSize = controller.DeviceEeprom.GetDeviceEepromSize();
133.
134.                Console.WriteLine(
135.                    $"Printing {eepromBytes.Length}/{eepromSize} bytes saved in EEPROM:
{string.Join(", ", eepromBytes)}");
136.            }
137.
138.            Console.WriteLine();
139.            Console.WriteLine("Example finished.");
140.            Console.ReadLine();
141.        }
142.    }
143. }
```


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

4.3. Python SDK installation

To install the Python SDK for the ECC-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 prein-stalled 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:



* 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).

4.4. Python code example

```
"""
Example how to set analog input signal with linear mapping for -2 to 3 dpts in
0 to 10 V range and save Snapshot on ECC-1C controller.
"""

import optoICC

#Connecting to board. Port can be specified like connect(port='COM4')
ecc1c = optoICC.connectEcc(port='COM4', verbose = True)
print("Connected to ECC-1C.")

lens = ecc1c.Lens

lens.Analog.SetAsInput()
print("Analog Input Activated")
print("-----------------------")

# Use 0 for Current, 3 for Focal Power
lens.Analog.SetLUTtype(3)
print("Analog Input in FP mode")
print("-----------------------")

# Min and Max input  [V]
lens.Analog.set_voltage_LUT([0, 10])
print("Analog Input: Voltage values are saved")
print("-----------------------")

# Min and Max focal power values [dpt]
lens.Analog.set_value_LUT([-2, 3])
```


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

```
print("Analog Input: FP values are saved")
print("-----------------------")

ecc1c.save_snapshot(1)
print("Analog Input: Snapshot saved")
print("-----------------------")

ecc1c.SnapshotManager.SetDefaultSnapShot(1)
print("Analog Input: Default Snapshot Saved")
print("-----------------------")

ecc1c.disconnect()
print("Disconnected")

```

```
"""
Example how to set signal generator with trigger and save Snapshot on ECC-1C controller.
"""

import optoICC

# Setting Waveform parameters
amplitude_FP = 2  # Focal Power amplitude
frequency = 2  # Hz
# offset = 0.5
# phase = 0

#Connecting to board. Port can be specified like connect(port='COM12')
ecc1c = optoICC.connectEcc(port='COM12', baudrate=256000)
print("ECC-1C connected.")

lens = ecc1c.Lens
sign_generator = lens.SignalGenerator

# print(lens.Manager.CheckSignalFlow())  # Signal Flow Manager check

sign_generator.SetAsInput()
sign_generator.SetShape(optoICC.WaveformShape.SINUSOIDAL)
sign_generator.SetFrequency(frequency)
sign_generator.SetAmplitude(amplitude_FP)
# sign_generator.SetOffset(offset)
# sign_generator.SetPhase(phase)

# print(sign_generator.get_register('external_trigger'))  # check of value in 0x6009 register
# Can be synchronised with external signal (0=disabled, 1=rising/falling edge, 2=rising edge trigger)
sign_generator.SetExternalTrigger(2)

# print(sign_generator.GetExtTriggerGPIO())  # check of value in 0x600c register
# Turns GPIOx pin to trigger input source (GPIOx default is trigger output) and disables global external trigger SYNCx pin
sign_generator.SetExtTriggerGPIO(True)

sign_generator.SetCycles(1)  # 1 cycle, period is synchronized with trigger
sign_generator.SetUnit(optoICC.UnitType.FP)  # Unit diopters
# capacity = ecc1c.SnapshotManager.GetSnapShotCapacity()  # the capacity of ECC-1C is 1
# print(capacity)

ecc1c.save_snapshot(1)
ecc1c.SnapshotManager.SetDefaultSnapShot(1)
# ecc1c.load_snapshot(1)  # if it is not set default, it can be loaded

# To reset the driver back to factory settings
# ecc1c.SnapshotManager.SetDefaultSnapShot(0)
```


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

5. Firmware

The ECC-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.

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 -0.318 to 0.318	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 ECC-1C's firmware uses the following types of registers and vectors for communication with the host PC or system:

1. **Registers:**
    a. **Definition:** Each register is a 32-bit value used for reading and/or writing data from/to the controller.
    b. **Data Transfer:** The register data is always transmitted in big-endian format (the most significant byte is transmitted first).
    c. **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:
        i. IEEE 754 32-bit float.
        ii. Unsigned integers.
        iii. Boolean values.
        iv. Custom-defined formats.
    d. Access Types:
        i. **Read-only:** Used for monitoring or retrieving data from the controller.
        ii. **Write-only** or **Read/Write:** Used to trigger some operations or update device states.
2. **Vectors:**
    a. **Definition:** Vectors are variable-length equivalents of registers, typically used to manage arrays of data.
    b. Structure:
        i. Vectors have defined size and data type, as documented for each system.
        ii. Byte-addressable in most cases, though some systems may require specific alignment (e.g. 4-byte chunks).

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

c. **Indexing:** When working with vectors, they require an index to specify the portion of the vector being accessed during the read/write operations.

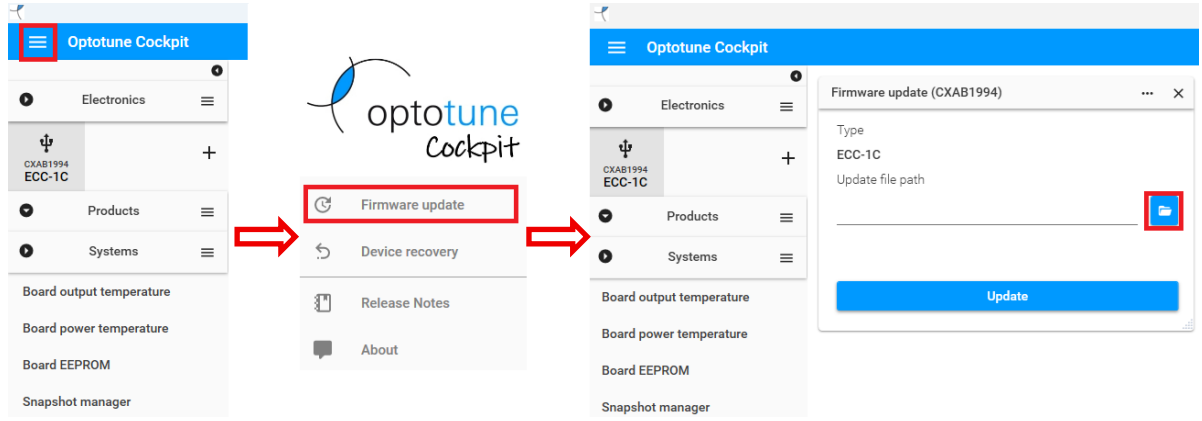
3. **Documentation:**
   a. More comprehensive details about the supported registers and vectors, their formats, and uses can be found in the ECC-1C Firmware Documentation. The firmware (.hex file) and its related documentation can be downloaded from Optotune's Download Center, after completing the required registration. The documentation must be first unzipped to show the correct formatting.

2.1.741648	ECC-1C	Support of EL-7-20. SmartStep support for EL-16-40 (5D and 20D), EL-12-30 and EL-7-20. Fast auto focus waveform and staircase waveform new trigger signal added in signal generator.	Firmware Firmware Documentation

5.2. Firmware update

The ECC-1C's firmware can be updated via the Optotune Cockpit software. **Updating the firmware via I2C 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 ECC-1C under the "Electronics" and select "Properties").

5.3. Firmware update – fix Baud rate mismatch

The older versions of the ECC-1C firmware used a fixed Baud rate during UART communication (Affected versions: v.2.0.741433 and before)

Problem : An ECC-1C with an old firmware is unable to communicate with the new cables manufactured after Feb 2025 using higher Baud rates, Optotune Cockpit is unable to find the device

Some ECC-1Cs in the field might need a firmware update, while communication via Cockpit is not available.

The firmware update procedure is done via STM32CubeProgrammer. In order to update the firmware, download the latest

ECC-1C firmware (v. 2.1.741648 or newer) from the Optotune website. In case you need step by step solution, contact customer support.

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

6. UART Communication Protocol

This chapter describes the protocols supported by the ECC-1Cs UART interface. While it is generally advised to use either of the SDKs presented in Chapter 4 Software development, the following enables to program ECC-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 system startup. Provides a basic command set (**Table 1: List of simple commands**) based on serial communication for straightforward interactions and quick setup. It's ideal for scenarios where minimal communication complexity is required. It allows interaction with the ECC-1C via commonly available serial terminals like **Termite** or **Putty**.

Interface Details:

- **Baud Rate**: Arbitrary (auto-detected by the device).
- **Parity**: None (N).
- **Stop Bits**: 1.
- **Data Bits**: 8.
- **Baud Rate Auto-Detection**: The device adapts to baud rate changes during operation, to do so "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 ECC-1C responds with **ASCII-encoded** messages (**Table 2: List of simple mode replies**), making debugging and testing straightforward.

Simple mode command	Description
START[CR][LF]	Answers "OK" if controller is ready to use and device is detected. Otherwise "ERROR" is received.
STATUS[CR][LF]	Controller answers with status encoded within 4 Bytes of information. Example: "0x00015000[CR][LF]". See 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 controller's firmware. 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: "14352500-00-A[CR][LF]".
GETVERSION[CR][LF]	Answers with firmware version number. Example: "1.0.740706[CR][LF]".
GETSN[CR][LF]	Answers with 40 bytes hexadecimal GIT build identification. Example : "eb8115e6b04814f0c37146bbe3dbc35f3e8992e0[CR][LF]"
GETGITSHA1[CR][LF]	Answers with board and device serial number. Example: "Board: CDAA0057, Device: ANAA1234[CR][LF]".
GOTODFU[CR][LF]	Starts controller's loader for firmware update. Note: no answer is sent via serial line

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

SETCURRENT=%f[CR][LF]	Sets current value. Command supports decimal parameter value in mA units. Current value is limited either by power capabilities of ECC-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=%f[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=%f[CR][LF]	Sets operational temperature limit in degree Celsius.
GETDEVICESN[CR][LF]	Answers with serial number of a device connected. Example: “Device: ANAA1234[CR][LF]”
DETECTDEVICE[CR][LF]	Runs autodetection of device on active channel, answers with device name. Example: “EL-16-40-TC[CR][LF]”.
SETCURLIMIT=%f;f[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 value of active current. Returned value is decimal number in units of milliamperes, Example: “15.6[CR][LF]”

Table 1: List of simple commands

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.

Table 2: List of simple mode replies

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:

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

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

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 / 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 Error Codes section of the firmware documentation.

6.2. Pro mode example

The section below shows very detailed description of pro mode with example of setting the signal generator and temperature reading of the output stage of ECC-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 ECC-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

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

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

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')

```


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

```
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')

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')
```


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

7. I2C Communication Protocol

In addition to UART communication, Optotune's ECC-1C also supports **I2C communication**, making it an excellent choice for direct integration with cameras. This is particularly advantageous when controlling Optotune's liquid lenses directly through a camera system. Focus commands between the host (e.g., PC) and the camera can be managed via the camera SDK or the GeniCam standard, whereas the camera and ECC-1C can interface using I2C.

7.1. I2C Configuration

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

1. Since ECC-1C has shared pins for I2C and UART, it 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 ECC-1C by the other communication option (but USB lines will still be available).
2. The "I2C slave address" setting is used to store the I2C address for the ECC-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 ECC-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.

7.2. I2C communication protocol description

Depending on the "I2C register count" setting in the Board EEPROM, the ECC-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.

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

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

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.

7.3. Reading Protocol

Reading is a two-step process:

- First, let's **write** to the Read Register Pointer (Bytes 1 and 2), specifying which System ID and Register Address we want to read (Bytes 5 and 6).

Byte #	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Meaning	Sys. ID for I2C (always 0x31)	Reg. address of I2C read Pointer (0x00 to 0x03)	0x00	0x00	System ID to be read	Register address to be read

Table 3: I2C Write

- Then, let's **read** the Register's content:

Byte #	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Meaning	System ID to be read	Register address to be read	Data	Data	Data	Data

Table 4: I2C Read

Reading example

In this example (Arduino code in **Section 7.9**), we want to read the **electrical current** being sent to the lens from the controller's output stage, and its **temperature** measured by the temperature sensor.

- 0x31 is the System ID for I2C
- 0xE8 is the System ID for LINEAR OUTPUT BASE
- 0x02 is the address for register OUTPUT CURRENT
- 0x22 is the System ID for TEMPERATURE MANAGER
- 0x00 is the address for register DEVICE TEMPERATURE

	Write to 1st Read Register Pointer						Write to 2nd Read Register Pointer					
Byte #	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	Sys. ID for I2C	Reg. address of 1st I2C read Pointer	0x00	0x00	System ID to be read	Register address to be read	Sys. ID for I2C	Reg. address of 2nd I2C read Pointer	0x00	0x00	System ID to be read	Register address to be read
Hex value	0x31	0x00	0x00	0x00	0xE8	0x02	0x31	0x01	0x00	0x00	0x22	0x00
Operation performed	Request to read OUTPUT CURRENT from LINEAR OUTPUT BASE						Request to read DEVICE TEMPERATURE from TEMPERATURE MANAGER					

Table 5: Write request to read output current and lens temperature

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

After sending this write command, we can now read back from the controller. An exemplary reply from ECC-1C would be:

Byte #	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	System ID to be read	Reg. address to be read	Data	Data	Data	Data	System ID to be read	Reg. address to be read	Data	Data	Data	Data
Hex value	0xE8	0x02	0x00	0x00	0x00	0x00	0x22	0x00	0x41	0xe2	0x80	0x00
Operation performed	Reading OUTPUT CURRENT from LINEAR OUTPUT BASE						Reading DEVICE TEMPERATURE from TEMPERATURE MANAGER					

Table 6: Reading output current and temperature

Converting the hexadecimal bytes into decimal:

- 0x00000000 → Current = 0.0 mA
- 0x41e28000 → Lens Temperature = 28.3125 °C

7.4. Writing protocol

Table 7: I2C Writing Protocol

displays the structure of a write command:

Byte #	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Meaning	Sys. ID	Reg. address	Data	Data	Data	Data

Table 7: I2C Writing Protocol

Writing Example

In this example (Arduino code in **Section 7.10**), we want to set 2.5 dpt as static input. Please note that, even if this were the only command that we want to send to ECC-1C, the last 6 Bytes need to also be sent (since the default number of registers to be written is 2, and therefore the default number of Bytes in the command is 12). For this reason, in the following example we also decide to read the temperature of the liquid lens, following the same protocol explained in section 7.3

Set 2.5 dpt focal power

- The System ID for STATIC INPUT BASE is 0x50
- The address of register “FP STATIC INPUT” is 0x04
- The float 2.5 corresponds to Hex 0x40200000, hence the data bytes: 0x40 0x20 0x00 0x00 in the table.

Read Lens Temperature

- The System ID for I2C is 0x31
- The address of I2C read Pointer 0 is 0x01
- The System ID for TEMPERATURE MANAGER is 0x22
- The register address for Device Temperature is 0x00

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

	Write to 1st register						Write to 2nd register					
Byte #	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	Sys. ID	Reg. address	Data	Data	Data	Data	Sys. ID	Reg. address			Sys. ID	Reg. address
Hex value	0x50	0x04	0x40	0x20	0x00	0x00	0x31	0x01	0x00	0x00	0x22	0x00
Operation performed	Set 2.5 dpt focal power as static input						Read lens temperature					

Table 8: I2C Example

7.5. How to change the default number of registers

In order to change the default number of registers (which is 2) to either 1 or 4, the following Write command should be sent, where:

- 0x20: System ID for Board EEPROM
- 0x00: Address for Lock
- 0x3F 0x47 0x44 0xF6: Key to unlock the register
- 0x01 – 0x02 – 0x04 : desired number of I2C slave registers (1, 2 or 4)

Byte #	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	Sys. ID (Board EEPROM)	Reg. address (Lock)	Data	Data	Data	Data	Sys. ID (Board EEPROM)	Reg. address (I2C slave num of regs)				Data
Hex value	0x20	0x00	0x3F	0x47	0x44	0xF6	0x20	0x17	0x00	0x00	0x00	0x01
Operation performed	Unlock “Lock” Register with 0x3F 0x47 0x44 0xF6						Set I2C slave num of registers to 0x01 = 1					

Table 9: I2C change the default number of registers

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

7.6. How to switch back to Autodetect

In order to switch back to Autodetect, the following Write command should be sent, where:

- 0x20: System ID for Board EEPROM
- 0x00: Address for Lock
- 0x3F 0x47 0x44 0xF6: Key to unlock the register
- 0x20 0x16: Input control setup (configuration of input interface)
- 0x00 0x00 0x00 0x00: Auto detection

Byte #	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	Sys. ID (Board EEPROM)	Reg. address (Lock)	Data	Data	Data	Data	Sys. ID (Board EEPROM)	Reg. address (I2C slave num of regs)				Data
Hex value	0x20	0x00	0x3F	0x47	0x44	0xF6	0x20	0x16	0x00	0x00	0x00	0x00
Operation performed	Unlock “Lock” Register with 0x3F 0x47 0x44 0xF6						Set interface to Autodetect mode					

Table 10: I2C change to Autodetect mode

7.7. I2C Communication example (Raspberry Pi)

The following example shows sending direct I2C commands to the ECC-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

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

Read command – answer from the ECC-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)

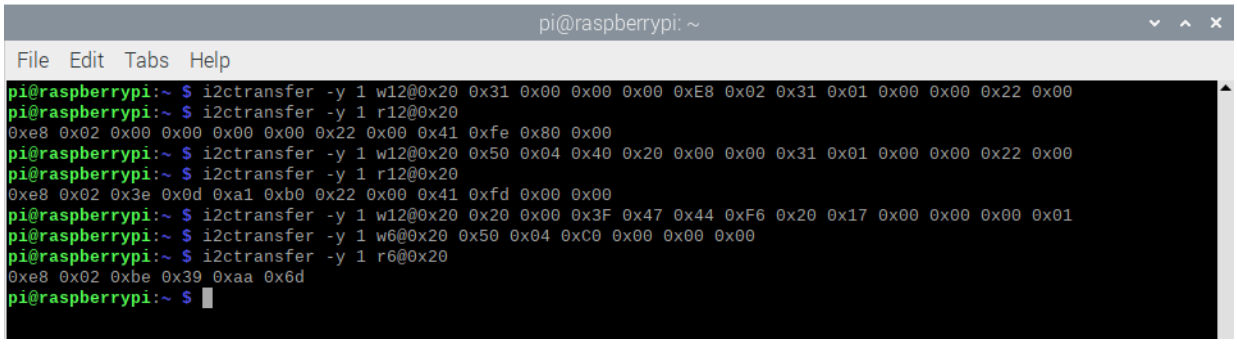
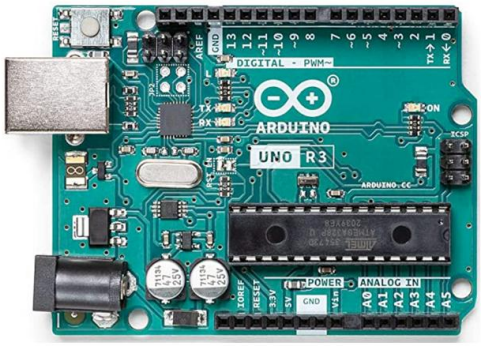
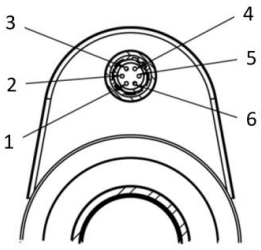


Figure 27: I2C communication with the ECC-1C using a Telnet Client

7.8. I2C Communication example (Arduino)

7.8.1 Pinout with Arduino UNO





Pin out Hirose connector HR10G-7R-6SB(73)

Position	Function	Value
1	GPIO Trigger	3V3 logic
2	Analog In	0-10V
3	UART Tx / I²C SCL	3V3 logic
4	UART Rx / I²C SDA	3V3 logic
5	GND	-
6	VCC input	See electrical specifications on page 1

Arduino	Connection to
Pin A4 (SDA)	ECC-1C Pin 4 (SDA)
Pin A5 (SCL)	ECC-1C Pin 3 (SCL)
GND	ECC-1C Pin 5 (GND)
USB A	PC

ECC-1C	Connection to
Pin 4 (SDA)	Arduino Pin A4 (SDA)
Pin 3 (SCL)	Arduino Pin A5 (SCL)
Pin 5 (GND)	Arduino GND & Power supply GND
Pin 6 (VCC)	5V or 9-24 V Power supply

Figure 28: Pinout for I2C communication between Arduino UNO (Master) and ECC-1C (Slave)

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

If Optotune's 152-219-00: CAB-6-100-M-OE (Hirose to open-ended wire cable, 1m) is used, the following color coding should be used on ECC-1C's side:

Pin	Cable Color	Function
1	Black/thin	GPIO trigger
2	Red/thin	Analog In
3	Grey/thin	Tx/SCL
4	Green/thin	Rx/SDA
5	Blue/thick	GND
6	Brown/thick	VCC

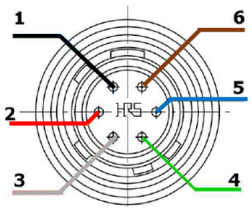


Figure 29: Color coding of CAB-6-100-M-OE

7.9. Arduino: Read Current and Lens Temperature

More detailed explanation on this example can be found in Section 0

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

//Read Output Current and Lens Temperature
byte dataArray_ReadCurTemp[12] = {0x31, 0x00, 0x00, 0x00, 0xE8, 0x02, 0x31, 0x01, 0x00, 0x00, 0x22, 0x00};
dataArray_SetFp_ReadTemp[12] = {0x50, 0x04, 0x40, 0x20, 0x00, 0x00, 0x31, 0x01, 0x00, 0x00, 0x22, 0x00};
int start = 0;

void setup() {
  Wire.begin();    //initialise I2C
  Serial.begin(9600);
}

void loop() {
    if (start == 0){ // only run once
      delay (100);

    Wire.beginTransmission(slaveAddress);
          for (int i=0; i<12; i++)
      {
        Wire.write(dataArray_ReadCurTemp [i]);  //data bytes are queued in local buffer
      } // end of for
    Wire.endTransmission();

    delay (2);

      Wire.requestFrom(slaveAddress, 12);    // request 12 bytes from slave device
      while(Wire.available()){   // slave may send less than requested

        Serial.println(Wire.read(),HEX);

      }

    start = 1;
    } // end of if
} // end of loop

```


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

ECC-1C's reply is:

```
E8
2
0
0
0
0
22
0
41
CF
80
0
```

Where:

- E8 and 2 are System ID and Register address for Linear Output and Output current, respectively
- 0x00000000 = 0.0 A (output current)
- 22 and 00 are System ID and Register address for Temperature Manager and Device Temperature, respectively
- 0x41CF8000 = 25.93 °C (lens temperature)

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

7.10. Arduino: Set Focal Power and Read Lens Temperature

More detailed explanation on this example can be found in Section 0

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

//Set Lens Focal Power and Read Lens Temperature

byte dataArray_SetFp_ReadTemp[12] = {0x50, 0x04, 0x40, 0x20, 0x00, 0x00, 0x31, 0x01, 0x00, 0x00, 0x22, 0x00};
int start = 0;

void setup() {
  Wire.begin();    //initialise I2C
  Serial.begin(9600);
}

void loop() {
    if (start == 0){ // only run once
      delay (100);

    Wire.beginTransmission(slaveAddress);
          for (int i=0; i<12; i++)
      {
        Wire.write(dataArray_SetFp_ReadTemp[i]);  //data bytes are queued in local buffer
      } // end of for
    Wire.endTransmission();

    delay (2);

      Wire.requestFrom(slaveAddress, 12);    // request 12 bytes from slave device
      while(Wire.available()){   // slave may send less than requested

        Serial.println(Wire.read(),HEX);
      }
    start = 1;
    } // end of if
} // end of loop

```

ECC-1C's reply is:

```
E8
2
3E
A
BC
80
22
0
41
D0
0
0
```

Where:

- E8 and 2 are System ID and Register address for Linear Output and Output current, respectively
- 0x3E0ABC80 = 0.135 A (output current)
- 22 and 00 are System ID and Register address for Temperature Manager and Device Temperature, respectively
- 0x41D00000 = 26.00 °C (lens temperature)

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

8. Analog voltage input mode

For the ECC-1C it is possible to map an analog voltage input 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 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 sets the input voltage and the corresponding output current or focal power. A toggle button is available to enable constant or linear extrapolation.

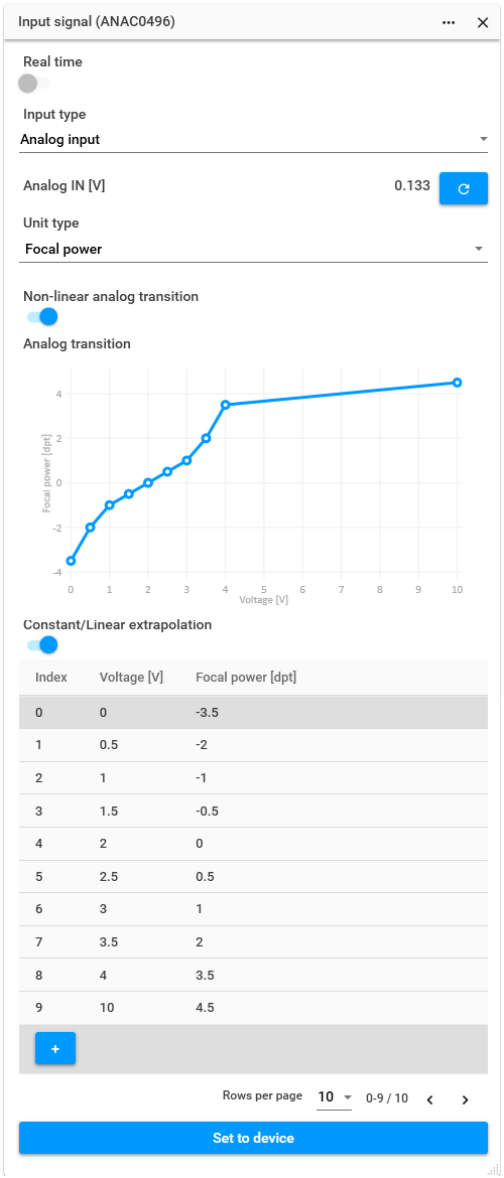


Figure 30: Analog voltage input with non-linear mapping of current values

Since the Analog input voltage mode may require configuring multiple points at each startup of the controller, the ECC-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 ECC-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

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

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 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.

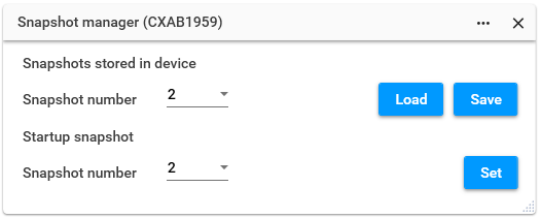


Figure 31: Snapshot manager widget

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

9. Input/Output trigger signals

The ECC-1C can generate an output trigger signal when changing the focal power or current of the lens or wait for an incoming trigger before applying a predefined change. These signals can be synchronized with the built-in signal generator or a custom vector. External trigger input is supported only when using the CAB-6-100-M-OE cable (Hi-rose to open-ended wire). The device also includes a signal generator and vector pattern unit. For the ECC-1C, the following type of waveforms can be defined:

- Sine
- Rectangle
- Triangle
- Sawtooth
- Pulse
- Staircase
- Fast Auto Focus Waveform
- Any custom vector

By default, the controller outputs a trigger signal on the GPIO pin. The trigger signal is HIGH (3.3V, max. 5 mA) at phase 0° of the selected waveform and goes LOW in the middle of the period. For pulse pattern, it reflects the duty cycle.

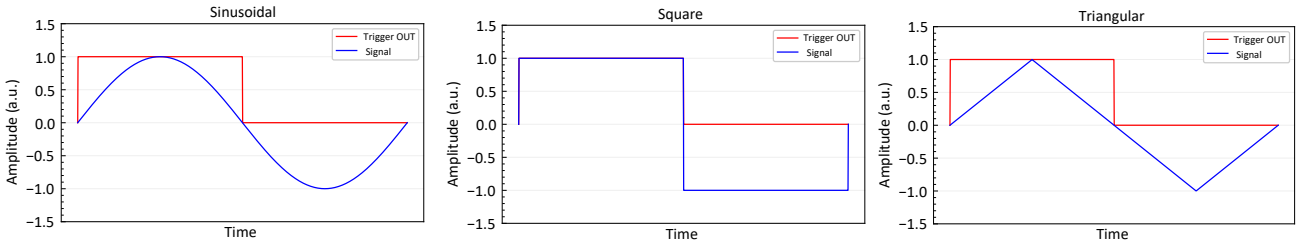


Figure 32: Different waveforms overlapped with the corresponding Trigger OUT signal

The signal generator can also be synchronized with an external input trigger. When the trigger input signal goes HIGH (max 3.3V), the selected waveform starts off at phase 0°.

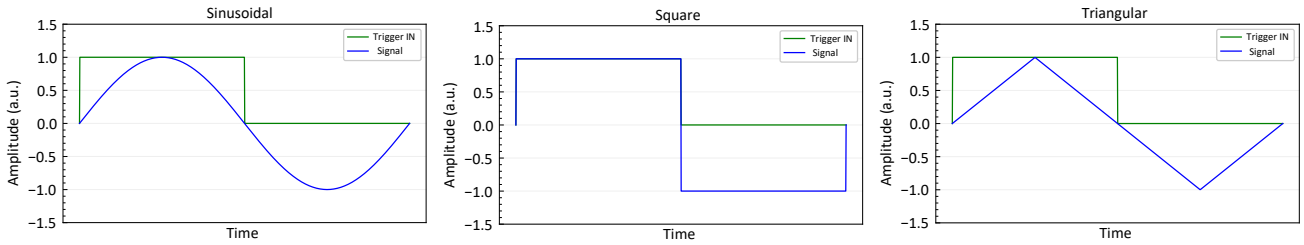


Figure 33: Different waveforms overlapped with an exemplary Trigger IN signal

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

10. Troubleshooting and FAQs

10.1. Diagnostics in Optotune Cockpit

If the device can be connected to a host computer via USB, 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.

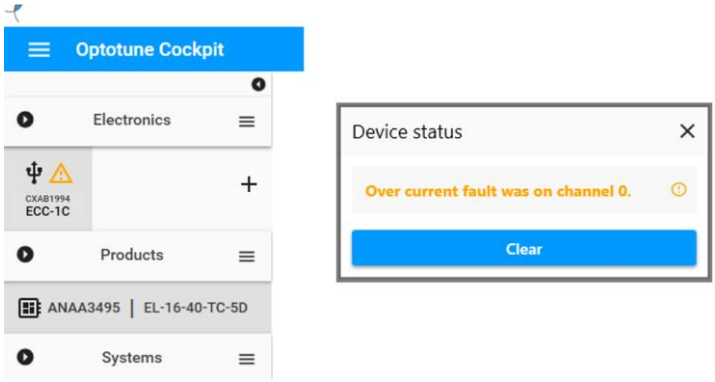


Figure 34: Example of a warning shown in Cockpit

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.

10.2. Diagnostics with Communication Terminal

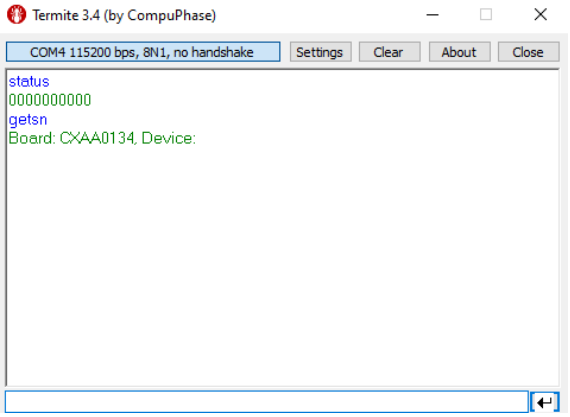
ECC-1C supports Simple Serial Communication Mode, which is based on the serial protocol and allows direct interaction with the device via a serial terminal such as Termite or PuTTY.

This mode uses ASCII-based commands and responses, terminated by carriage return and line feed (CR, LF or 0x0D, 0x0A). The protocol is not case sensitive, and whitespace is ignored, making it easy to use. Baud rate is arbitrary, thanks to ECC-1C's autodetection feature, which adapts to changes on the fly. To correctly initialize communication at a new baud rate, the Start command must be sent first.

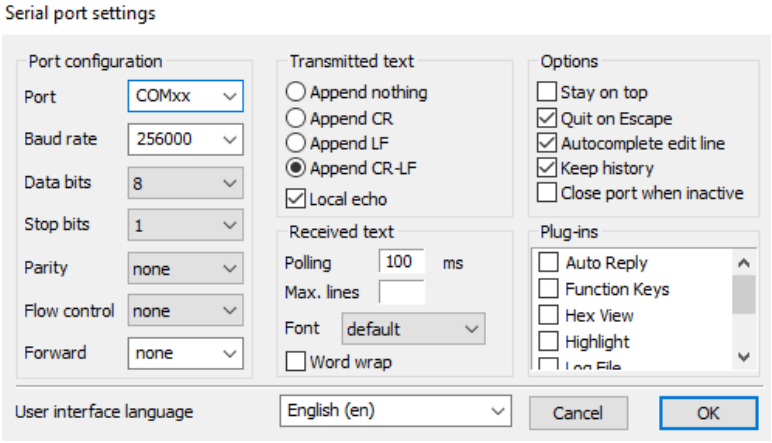
If you have purchased the USB to UART cable with Hirose connector, you can test communication with the ECC-1C using Termite (https://www.compuphase.com/software_termite.htm).

Every important information is mentioned in the datasheet, try connecting to device by command Start and then, use Status command to see the response. Something like this below:

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



Sample setting in Termite:



Focal power limits can be obtained by using the commands below:

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”.

10.3. FAQs

Question 1:

The ECC-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.

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

Question 3:

The ECC-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 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.

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