AS1170 EVK

Quick Start Guide

Evaluation software GUI

Published by ams-OSRAM AG
Tobelbader Strasse 30, 8141 Premstaetten, Austria
Phone +43 3136 500-0
ams-osram.com

Table of contents

1 Introduction 3

2 Launching the GUI 4

2.1 Prerequisites 4

2.2 Running the executable 5

3 Main script overview 5

3.1 Controller class 6

3.2 Main execution 6

4 Landing window overview 6

4.1 Building the GUI 6

5 Interactive window 9

5.1 Classes 10

6 ESP32 commands 15

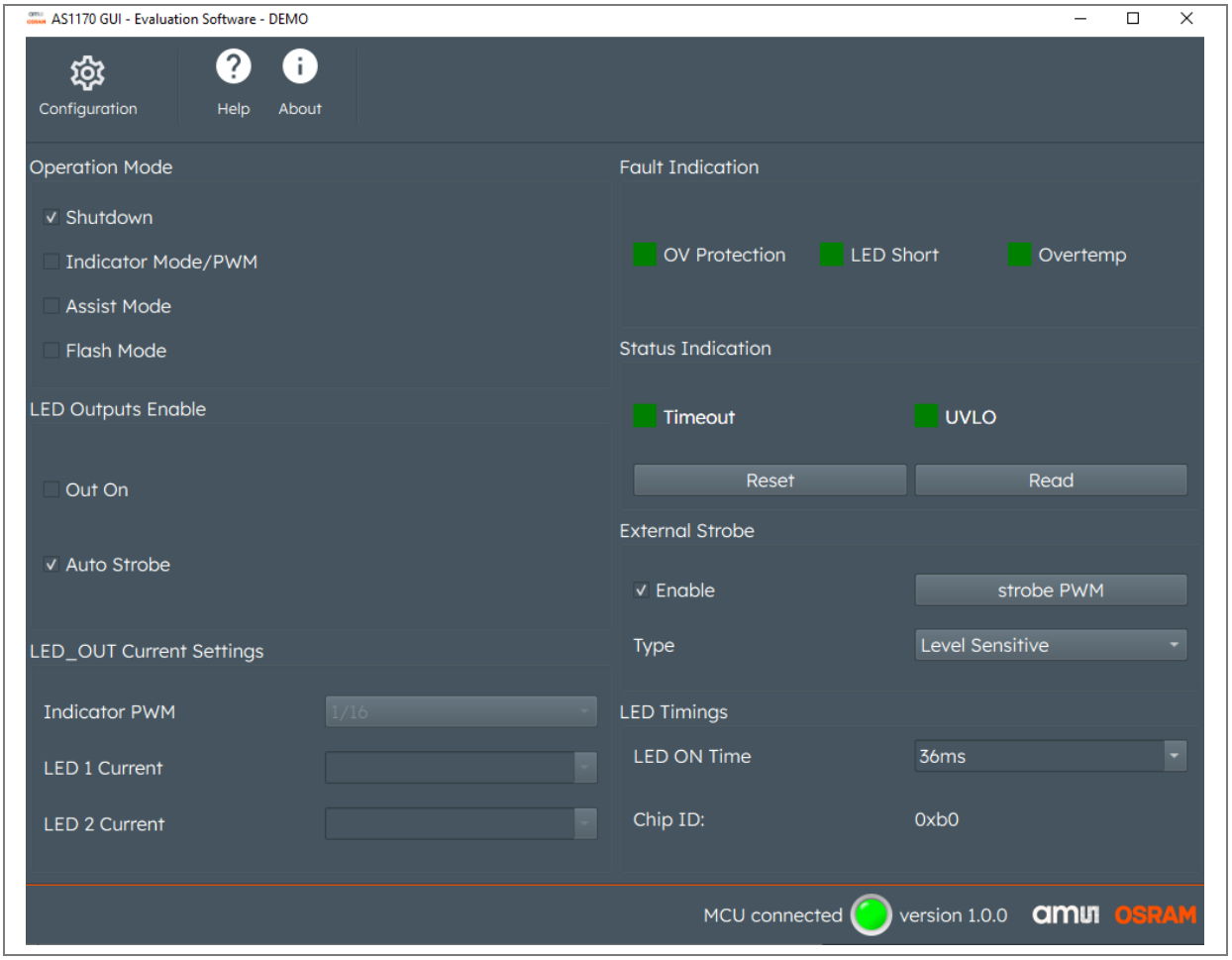
7 Revision information 16

8 Legal information 17

1 Introduction

The AS1170 GUI is an evaluation software designed to interact with the AS1170 hardware. This guide provides detailed instructions on launching and using the GUI, starting from the executable file AS1170_GUI.exe.

Figure 1: AS1170 GUI – evaluation software



2 Launching the GUI

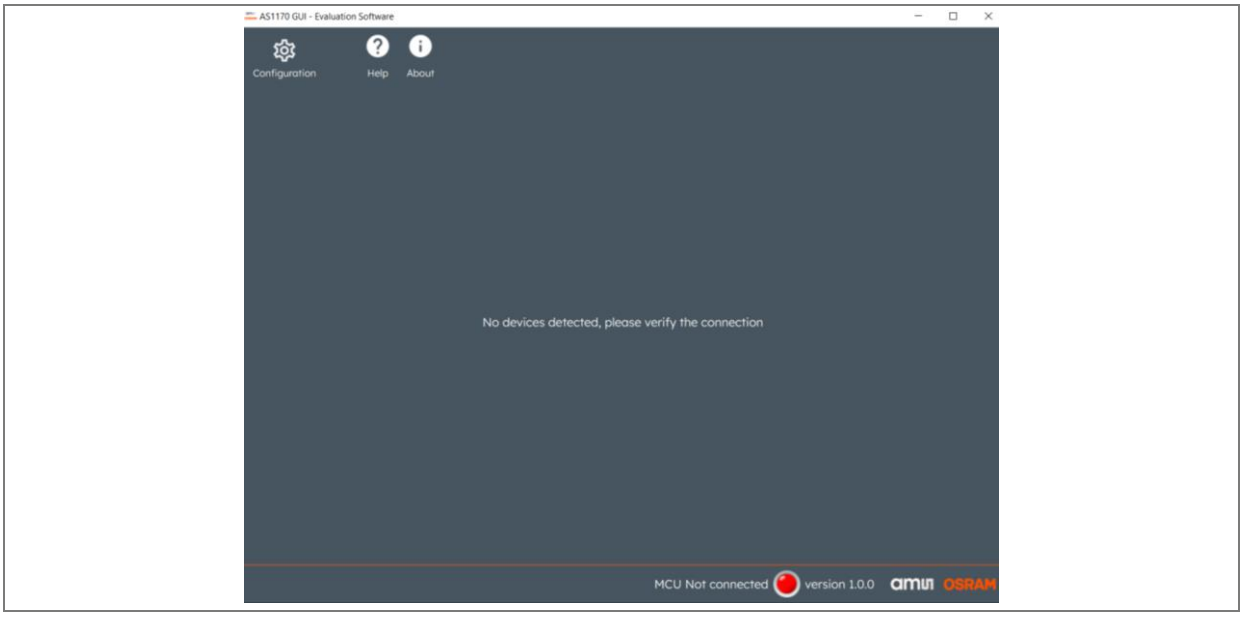
2.1 Prerequisites

Before launching the GUI, ensure the following:

- The AS1170 hardware is connected to your computer.
- The necessary drivers (e.g., USB to UART) are installed.
- The AS1170_GUI.exe file is available on your system.

Unless the result would be the following one:

Figure 2: AS1170 GUI (Status – no devices detected)



2.2 Running the executable

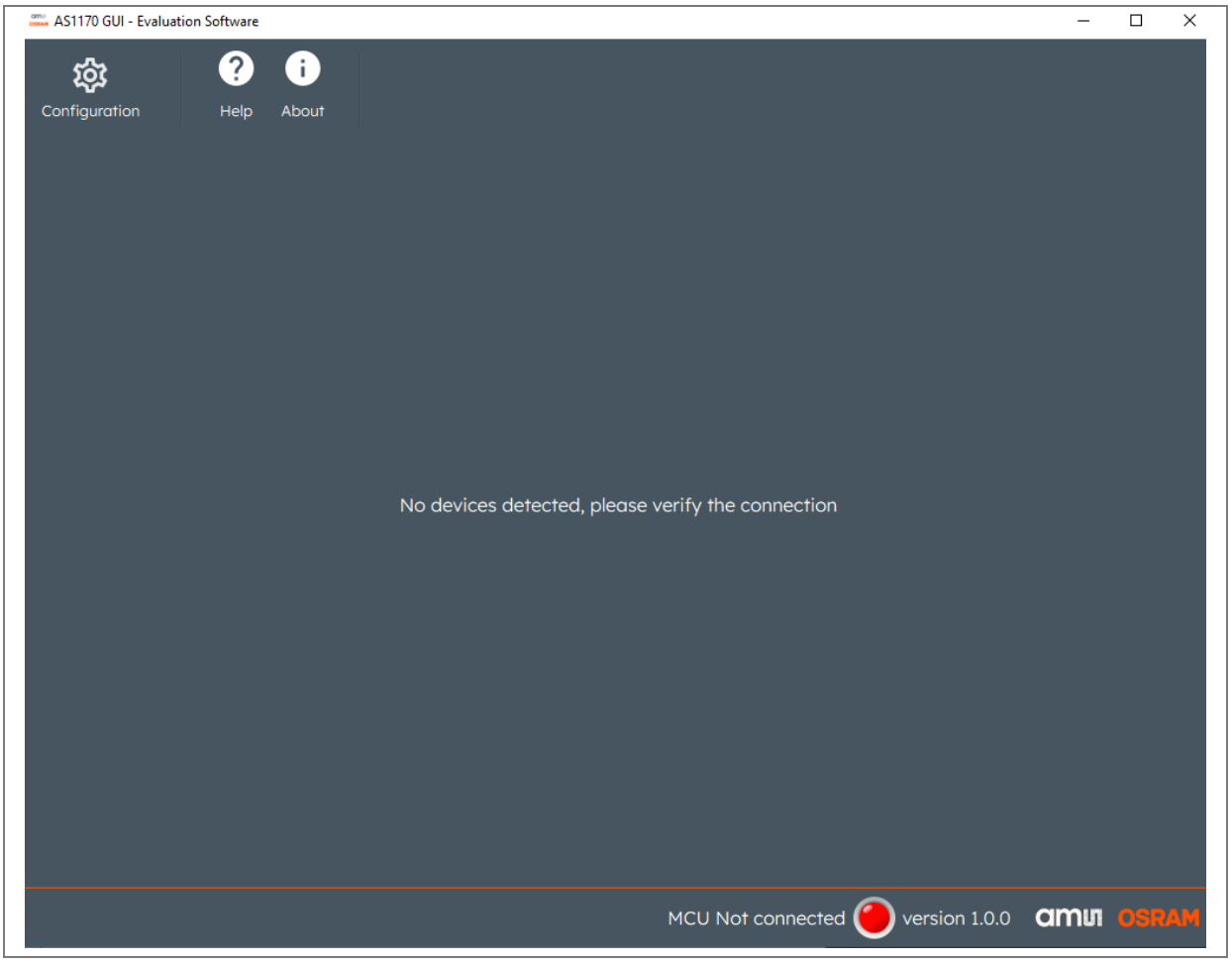
To launch the GUI, follow these steps:

- Locate the Executable: Navigate to the directory where AS1170_GUI.exe is stored.
- Run the Executable: Double-click on AS1170_GUI.exe to start the application.

3 Main script overview

The main script initializes and runs the GUI application. Below is an overview of the main components of the script.

Figure 3: AS1170 GUI – overview of main components



3.1 Controller class

The Controller class manages the overall application. It includes:

- Application Info: Metadata about the application such as name, version, and description.
- Initialization: Sets the root folder and initializes the view.
- Run Method: Starts the GUI.
- New Configuration Method: Sets a new client view.

3.2 Main execution

The script checks if it is being run as the main module and then creates an instance of the Controller class, running the GUI.

4 Landing window overview

The View class in the module view.py initializes the main window and sets up the GUI components.

4.1 Building the GUI

The __build_gui method sets up the main components of the GUI, including the toolbar, status bar, and client view.

4.1.1 Toolbar

The toolbar includes buttons for configuration, help, and about.

Figure 4: Toolbar buttons (Configuration, Help, and About)

4.1.2 Status bar

The status bar displays the connection status and version information. It shows if the right MCU is connected to the USB ports or not.

Figure 5: Status bar (MCU not connected)

If the right device is connected the LED turns green.

Figure 6: Status bar (MCU connected)

4.1.3 Client view

The client view is initially set to an empty view, which displays the connection status of the device.

4.1.4 Updating the GUI

The GUI is updated periodically to reflect the current status of the connected device.

4.1.5 Running the application

The run method starts the application event loop.

4.1.6 Handling configuration

The configuration button allows the user to set up the system settings. If the right device is connected, when the button is pressed, it is shown in the main interactable window. It is imported from the module client_view.py described in the next section.

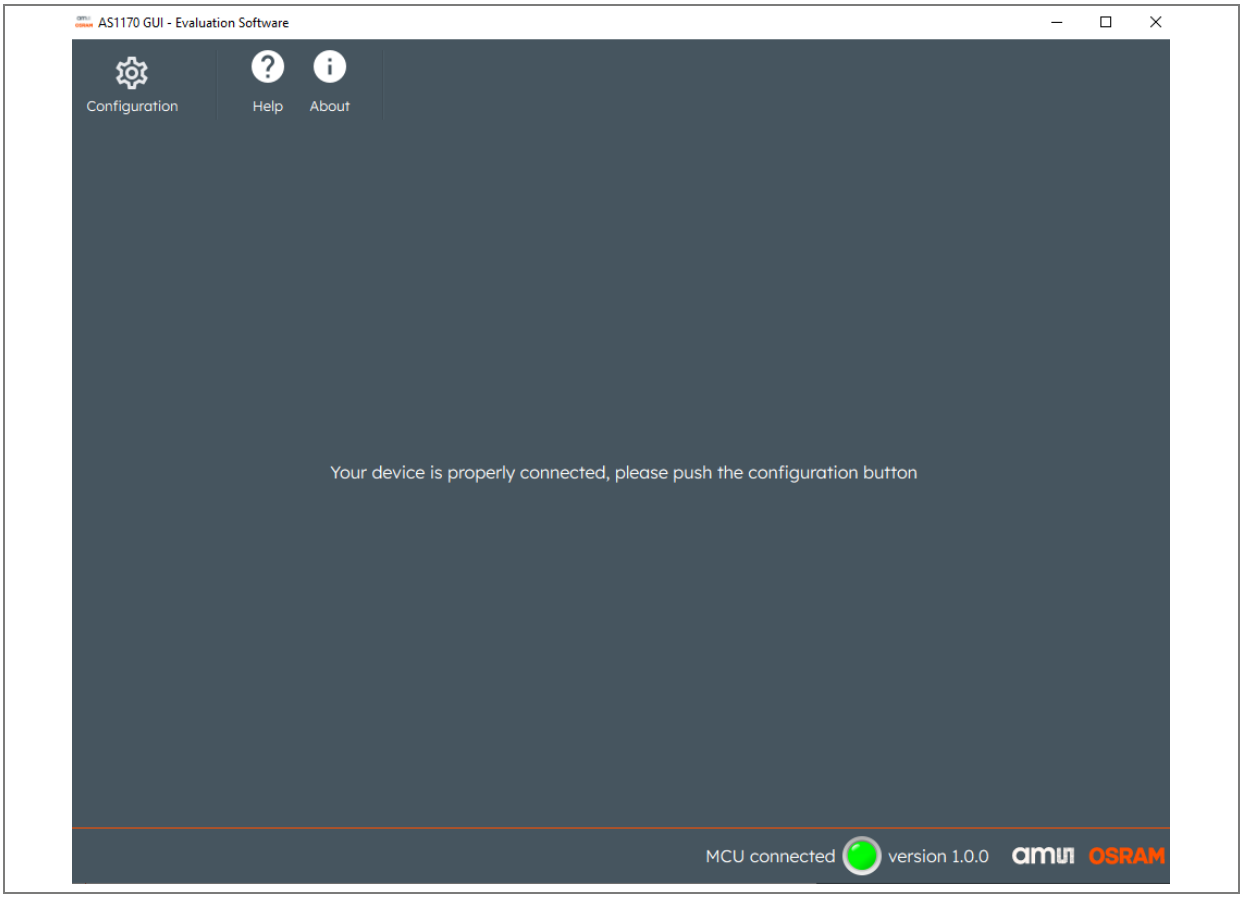
4.1.7 Help and about dialogs

The help and about buttons provide additional information about the application.

4.1.8 Updating the window title

The window title is updated to reflect the current state of the application.

Figure 7: Connection status of the device

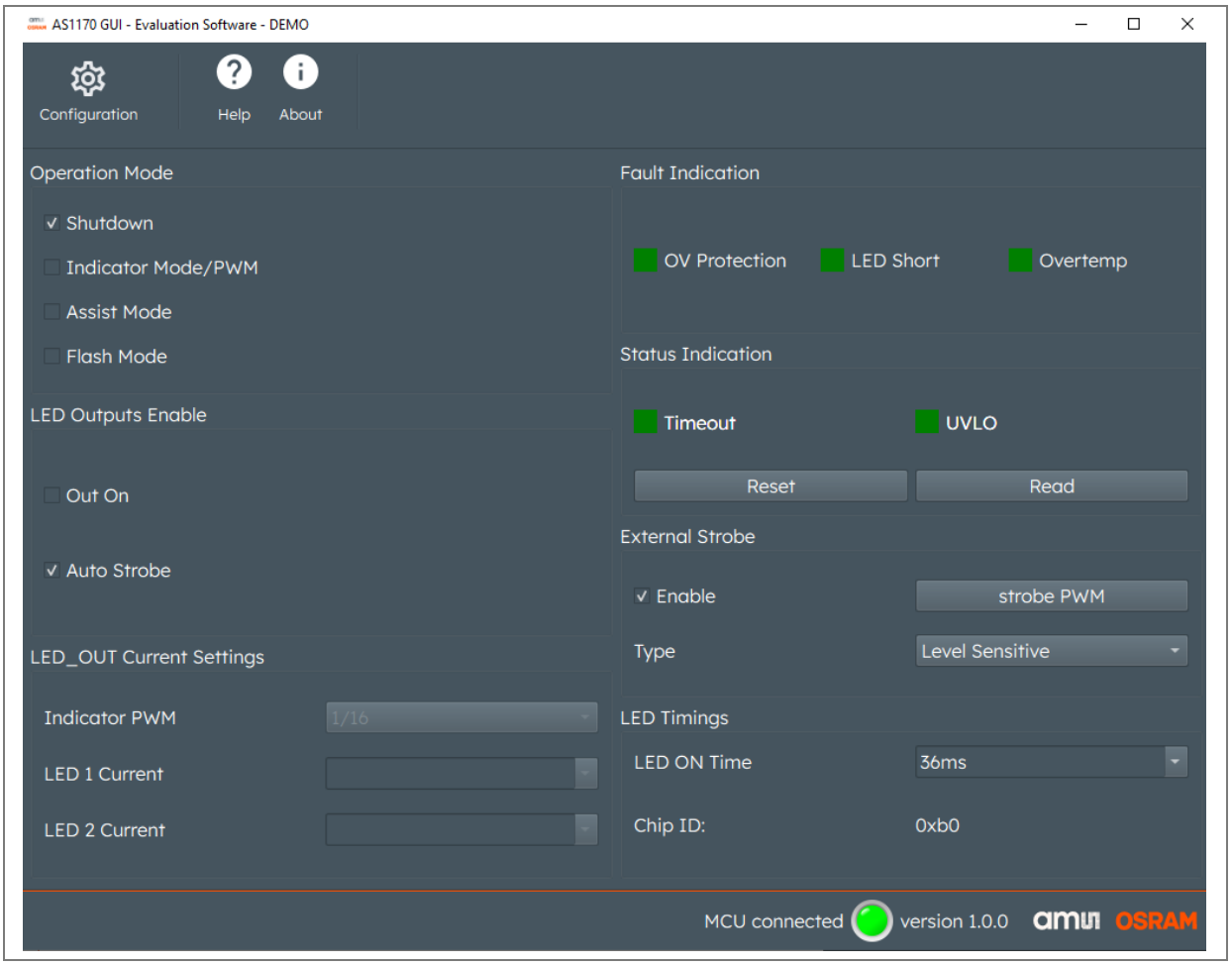


5 Interactive window

The interactable window is contained in the client_view.py module. It is triggered when the configuration button is pressed, and there is a recognized device attached.

This module provides a comprehensive interface for interacting with the AS1170 hardware, allowing users to configure settings, read register values, and monitor the status of the device through a user-friendly GUI.

Figure 8: Interactive window



5.1 Classes

- SerialReaderThread (QThread)
- MyComboBox (QComboBox)
- ClientView (QScrollArea)

5.1.1 SerialReaderThread

The SerialReaderThread class is a QThread that reads data from the serial port in a separate thread, ensuring that the GUI remains responsive.

- Signals: line_read signal is emitted when a line is read from the serial port.
- Initialization: Takes a serial_reader object as a parameter.
- Run Method: Continuously reads lines from the serial port and emits the line_read signal.

5.1.2 MyComboBox

The MyComboBox class extends QComboBox to provide additional functionality, such as auto-completion and validation.

- Initialization: Sets the combo box to be editable and configures the completer.
- focusOutEvent: Ensures that the current text is valid when the combo box loses focus.

5.1.3 ClientView

The ClientView class is the main class for the client view, inheriting from QScrollArea. It sets up the GUI and handles communication with the AS1170 hardware.

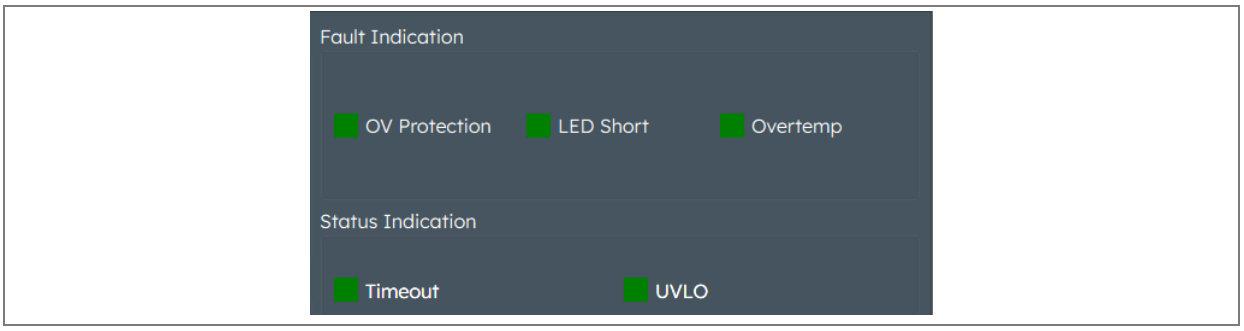
5.1.3.1 Initialization

- Constructor: Initializes the GUI components and starts the serial communication thread.
- Serial Communication: Finds the Silicon Labs port and initializes the Commands object for serial communication.

5.1.3.2 Serial communication

- read_from_serial: Continuously reads data from the serial port and updates the GUI.
- _update_gui: Processes the data read from the serial port and updates the GUI elements based on the register values. Each of the registers value read updates the corresponding field in the GUI. Here there are also the indicators for fault and status.

Figure 9: Indicators for fault and status

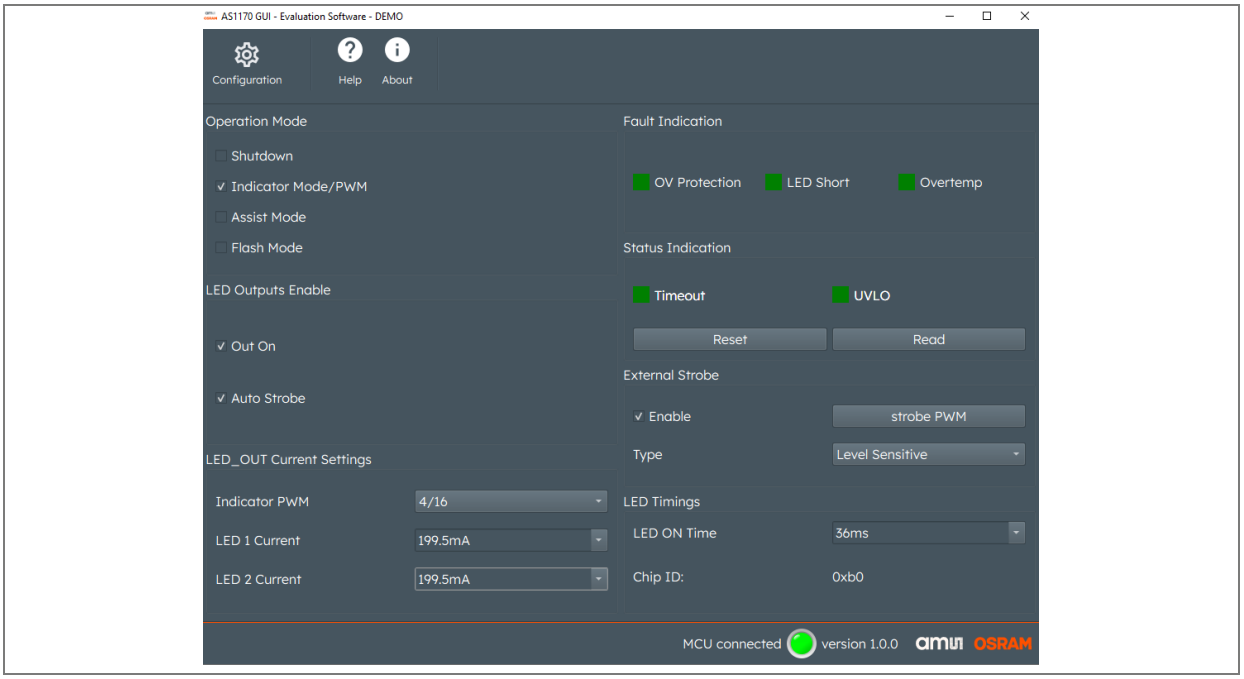


These are read only registers. The green box is updated if and only if there is change in the register. The indicator becomes red if the register value is 0, green if 1.

5.1.3.3 GUI layout

- __init: Sets up the layout of the GUI, including various controls and indicators.
  - Group Boxes: Creates group boxes for different sections such as Operation Mode, LED Outputs Enable, Fault Indication, Status Indication, LED_OUT Current Settings, External Strobe, and LED Timings.
  - CheckBoxes: Adds checkboxes for different modes and settings, and connects their signals to appropriate slots.
  - Indicators: Creates indicators for fault and status indications, and updates their colors based on the register values.
  - ComboBoxes: Adds combo boxes for PWM and current settings, and connects their signals to appropriate slots.
  - Buttons: Adds buttons for reset and read operations, and connects their signals to appropriate slots.

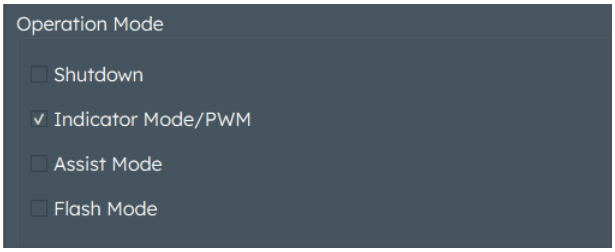
Figure 10: Overview of GUI



5.1.3.4 Event handlers

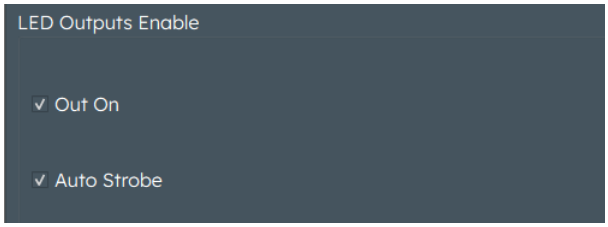
- on_checkbox_clicked_dependent: Handles the state change of dependent checkboxes and updates the register values accordingly.

Figure 11: Operation mode



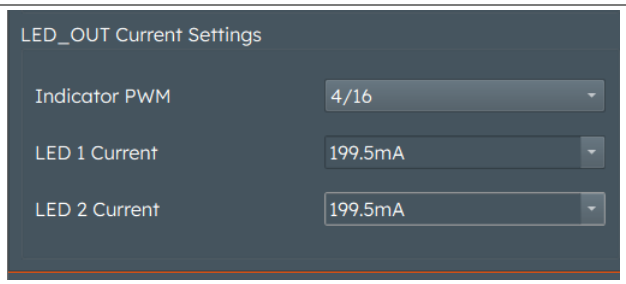
- on_checkbox_clicked_independent_4: Handles the state change of the “Out On” checkbox and updates the register values.
- on_checkbox_clicked_independent_5: Handles the state change of the “Auto Strobe” checkbox and updates the register values.

Figure 12: LED outputs enable



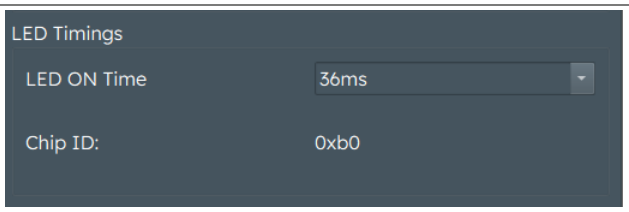
- write_register_PWM: Writes the PWM value to the register based on the selected combo box value. This combobox is activated only if the mode setting is in IndicatorPWM (if in the checkbox_dependent function, checkbox1 is checked).
- write_register_current: Writes the current value to the register based on the selected combo box value. This combobox contains the limit values of each mode setting. If the current mode is shutdown, the combobox is deactivated. If the IndicatorPWM checkbox is checked then values up to 0x3F are admitted. If assist light mode (checkbox2 is clicked) then the values are up to 0x7F (448mA). If flash mode (checkbox3) is active, then the combo box allows the full scale of values (up to 900mA).

Figure 13: LED_OUT current settings



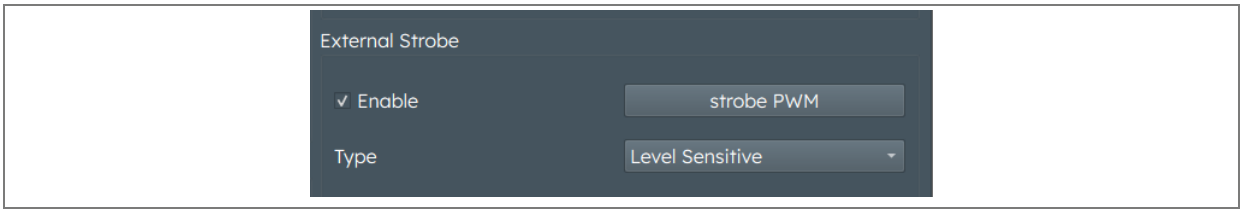
- write_register_flash: Writes the flash value to the register based on the selected combo box value.

Figure 14: LED timings



- write_register_strobe_checkbox: Writes the strobe value to the register based on the state of the checkbox.
- write_register_strobe_combobox: Writes the strobe value to the register based on the selected combo box index. It is enabled only if the previous checkbox is enabled.
- send_command_to_ESP32("strobePWM"): Triggers the PWM signal from the ESP32 GPIO.

Figure 15: External strobe



- updateComboBox: Updates the combo boxes based on the state of the checkboxes.
- on_checkBoxEN_stateChanged: Enables or disables the strobe combo box and button based on the state of the checkbox.
- on_checkBox_stateChanged: Ensures that only one of the dependent checkboxes is checked at a time and updates the combo boxes accordingly.
- reset_registers: Button. Sends a reset command to the ESP32.
- read_registers: Button. Sends a read command to the ESP32.

Figure 16: Reset and read register buttons

- send_command_to_ESP32: Sends a command to the ESP32 via the serial port.

5.1.3.5 Utility

- find_silicon_labs_port: Finds and returns the Silicon Labs port for serial communication.

6 ESP32 commands

The commands.py module is designed to facilitate communication with the ESP32 microcontroller via serial communication. This module includes several key functions that allow for reading and writing data to the ESP32, as well as sending specific commands. Below is an overview of the main components and their functionalities:

- Initialization: The Commands class initializes the serial connection with the specified port and baud rate. If the connection fails, an error message is displayed.
- Reading Data: The read_line method reads a line of data from the serial port, decodes it, and returns the stripped string.
- Reading Registers: The read_registers method sends a command to the ESP32 to read the register values.
- Writing Registers: The write_registers method formats and sends a command to the ESP32 to write a specified binary value to a register. The register address and bit can be specified.
- Sending Commands: The send_command_to_ESP32 method sends a given command to the ESP32 via the serial port.
- Closing Connection: The close method closes the serial connection if it is open.

This module is essential for interacting with the ESP32, allowing for efficient data exchange and command execution.

7 Revision information

Definitions

Draft / Preliminary:
The draft / preliminary status of a document indicates that the content is still under internal review and subject to change without notice. ams-OSRAM AG does not give any warranties as to the accuracy or completeness of information included in a draft / preliminary version of a document and shall have no liability for the consequences of use of such information.

Changes from previous version to current revision v1-00	Page
Initial production version	

- Page and figure numbers for the previous version may differ from page and figure numbers in the current revision.
- Correction of typographical errors is not explicitly mentioned.

8 Legal information

Copyright & disclaimer

Demo Kits, Evaluation Kits and Reference Designs are provided to recipient on an “as is” basis for demonstration and evaluation purposes only and are not considered to be finished end-products intended and fit for general consumer use, commercial applications and applications with special requirements such as but not limited to medical equipment or automotive applications. Demo Kits, Evaluation Kits and Reference Designs have not been tested for compliance with electromagnetic compatibility (EMC) standards and directives, unless otherwise specified. Demo Kits, Evaluation Kits and Reference Designs shall be used by qualified personnel only.

ams-OSRAM AG reserves the right to change functionality and price of Demo Kits, Evaluation Kits and Reference Designs at any time and without notice.

Any express or implied warranties, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose are disclaimed. Any claims and demands and any direct, indirect, incidental, special, exemplary or consequential damages arising from the inadequacy of the provided Demo Kits, Evaluation Kits and Reference Designs or incurred losses of any kind (e.g. loss of use, data or profits or business interruption however caused) as a consequence of their use are excluded.

ams-OSRAM AG shall not be liable to recipient or any third party for any damages, including but not limited to personal injury, property damage, loss of profits, loss of use, interruption of business or indirect, special, incidental or consequential damages, of any kind, in connection with or arising out of the furnishing, performance or use of the technical data herein. No obligation or liability to recipient or any third party shall arise or flow out of ams-OSRAM AG rendering of technical or other services.

Headquarters
ams-OSRAM AG
Tobelbader Strasse 30
8141 Premstaetten
Austria, Europe
Tel: +43 (0) 3136 500 0

Please visit our website at ams-osram.com
For information about our products go to Products
For technical support use our Technical Support Form
For feedback about this document use Document Feedback
For sales offices and branches go to Sales Offices / Branches
For distributors and sales representatives go to Channel Partners