MC110 PDI Builder

Release 6.12.35

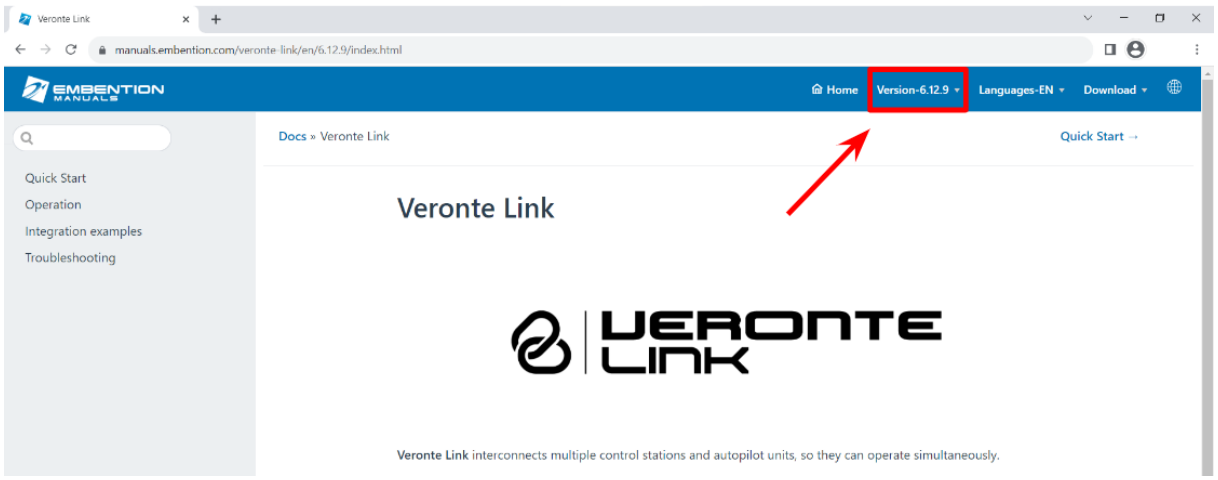
Embention

2024-01-09

CONTENTS

MC110 PDI Builder is an application employed to configure Veronte motor controllers MC110.

Warning: Select your version before reading any user manual for software. The following image shows where to select a version from any Embention user manual.



CHAPTER

ONE

QUICK START

MC110 PDI Builder is the main tool for setting all configurable parameters of the **Veronte Motor Controller MC110**.

Here the user can configure, tune and define the motor, control systems and sensors to be used together with the ESC.

Once MC110 has been detected on Veronte Link, download and install **MC110 PDI Builder**.

1.1 Download

Once a MC110 device has been purchased, a GitHub release should be created for the customer with the application.

To access to the release and download the software, read the Releases section of the **Joint Collaboration Framework** manual.

1.2 Installation

To install **MC110 PDI Builder** on Windows just execute the "Mc110PDIBuilder.exe" file and follow the indications of the *Setup Wizard*. Administrator rights are needed.

> **Warning:** In case of any issue during installation, please disable Windows Defender and Firewall.
>
> To disable Firewall, go to "Control Panel" and "Firewall of windows", then click on **Turn off**.

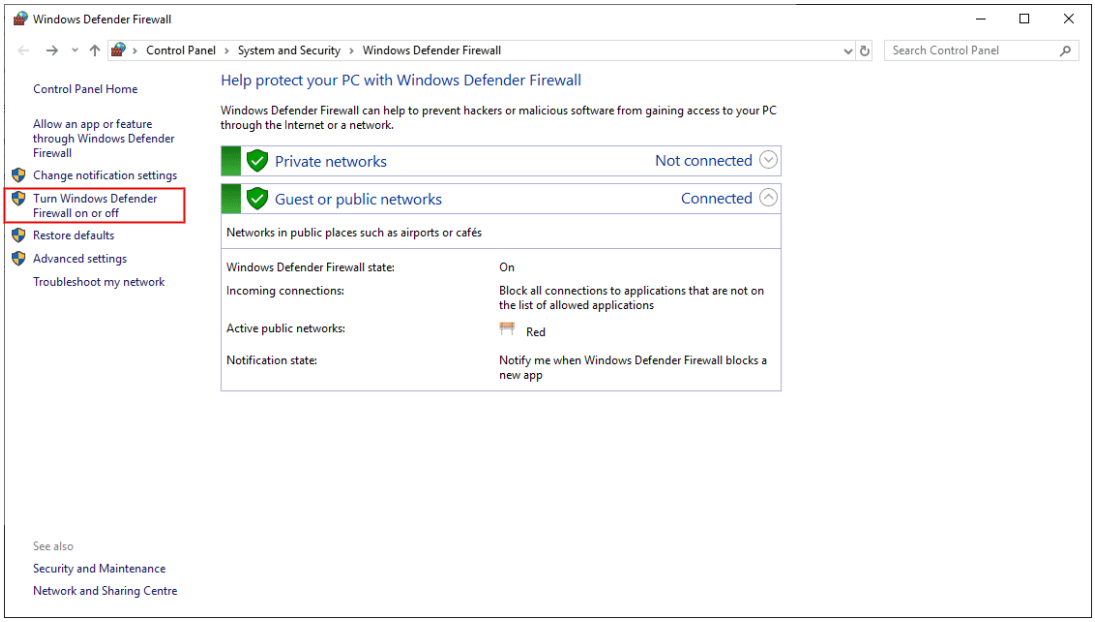


Fig. 1: **Windows Defender Firewall**

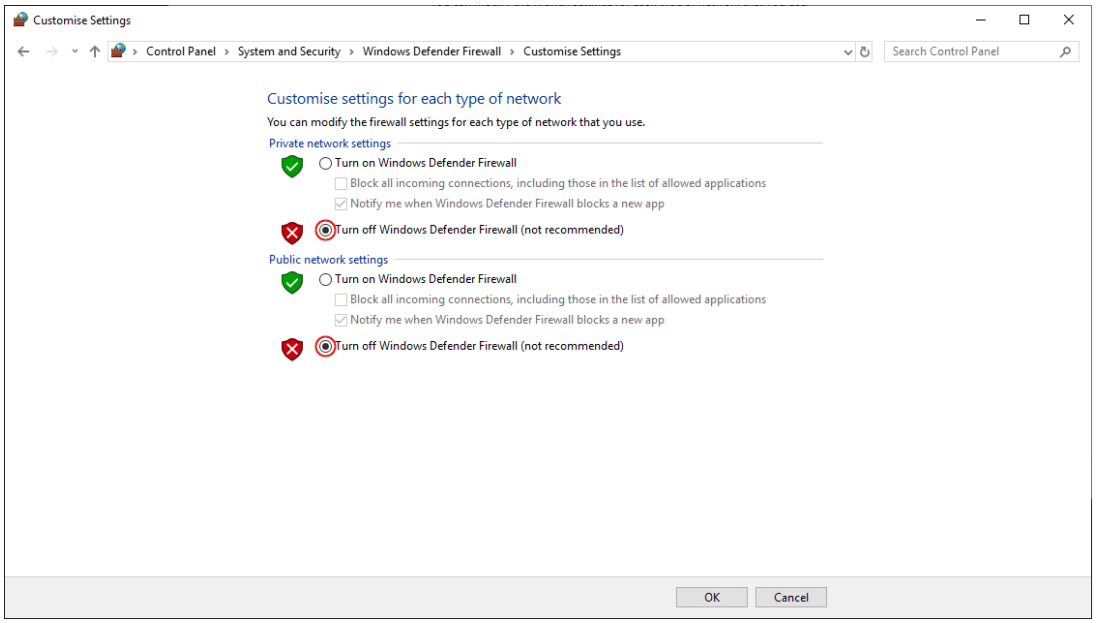


Fig. 2: **Windows Defender Firewall: Settings**

CHAPTER

TWO

CONFIGURATION

This section explains each option and parameter available in **MC110 PDI Builder**.

Once the installation is finished, open **MC110 PDI Builder** and select the unit.

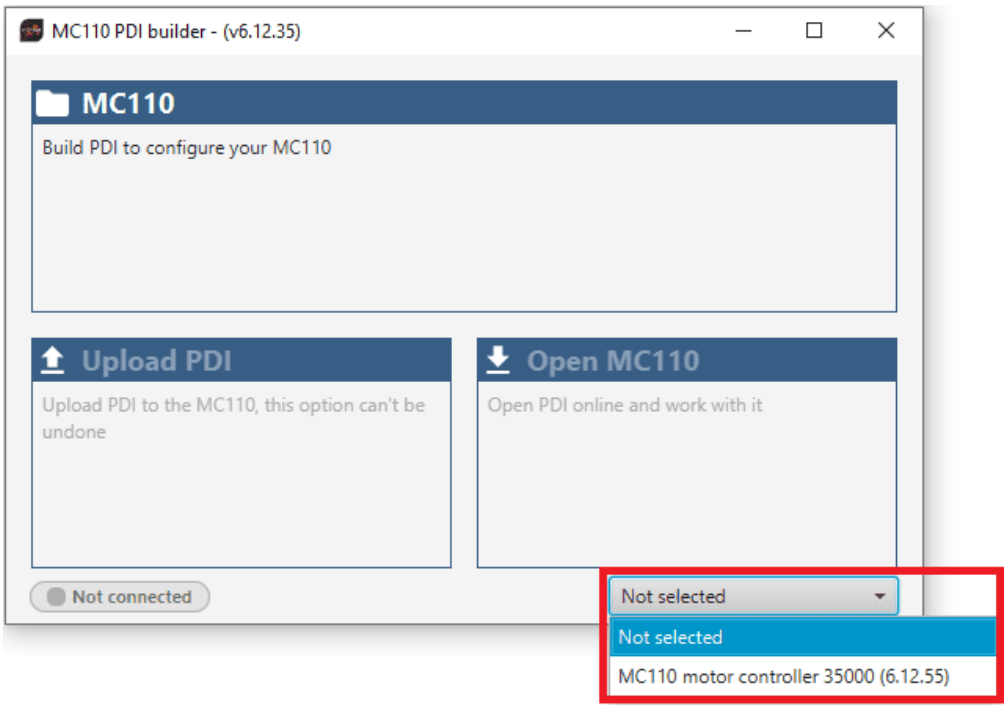


Fig. 1: **MC110 ID**

If the MC110 is correctly connected, **MC110 PDI Builder** will display the **mode** in which the connected unit is. In addition, a **PDI error button**  will appear:

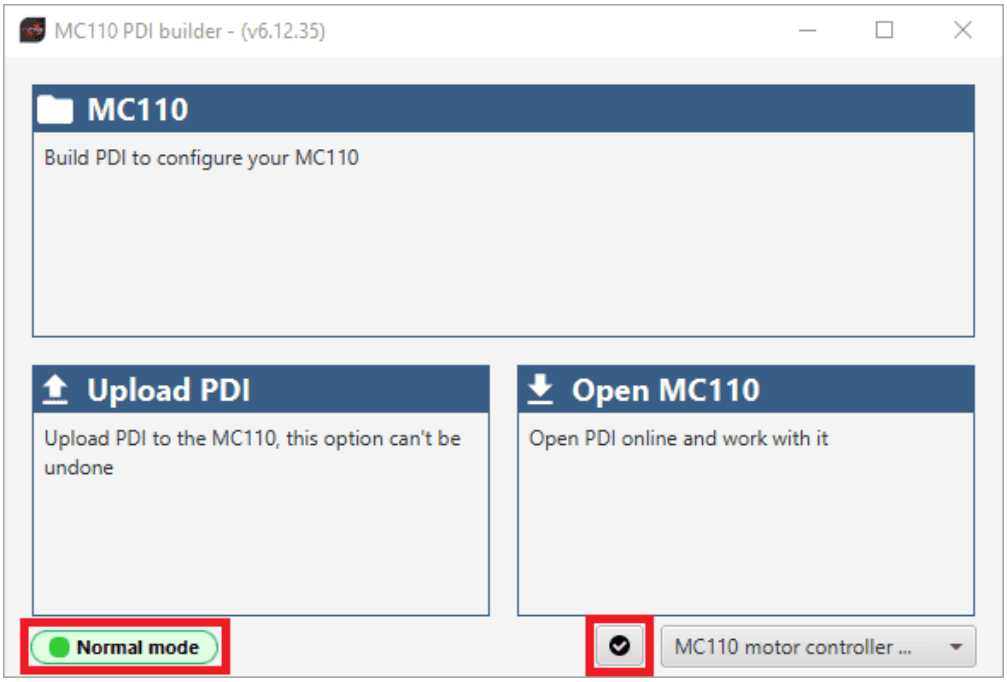


Fig. 2: **MC110 PDI Builder**

- **MC110 mode**: The MC110 unit should appear in **Normal mode**, as shown in the figure above, or **Maintenance mode**.

  It can also appear as **Maintenance mode (loaded with errors)** or **Normal mode - Disconnected**.

  Note: **Maintenance mode (loaded with errors)** appears when something is wrong in the configuration.

-  **PDI Errors** button: The user can check if the connected unit has PDI Errors by simply clicking on it. If there are no errors, the following message appears:

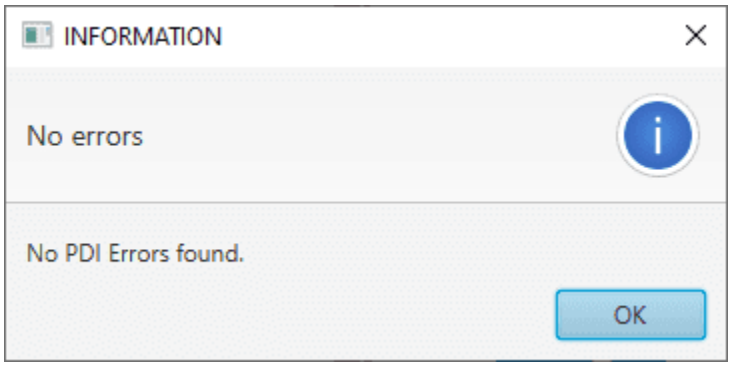


Fig. 3: **MC110 PDI Builder - PDI Errors message**

The user can access now to 3 configuration options:

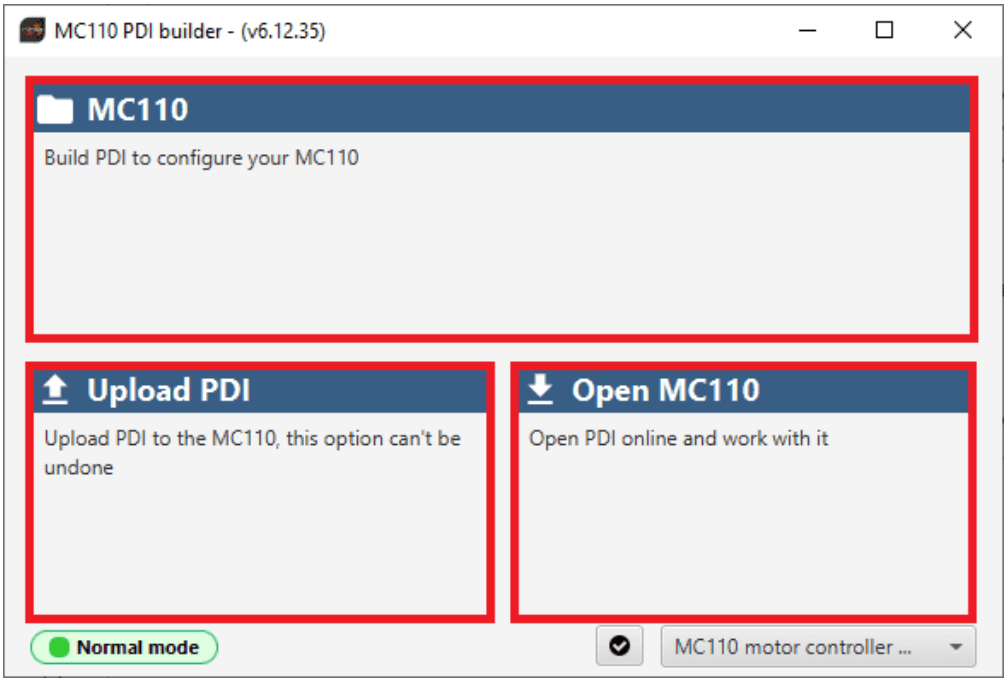


Fig. 4: **MC110 PDI Builder options**

- **MC110**: It allows the user to work with **offline** configurations. A previously exported MC110 PDI configuration can be opened and modified or it is also possible to build a new one from the default configuration.
- **Upload PDI**: A previously exported **MC110 PDI configuration** can be imported to the current **MC110** flash memory.
- **Open MC110**: By clicking on this option, **MC110 PDI Builder** configuration menu opens with the configuration (the PDI files) loaded in the MC110. Then, the user can modify it online.

Note: These PDI files are the files that contain the configuration of the MC110. They are used by modular control with improved version management.

These files are located in a single folder, **setup**, which contains several .xml files, where all parameters and the control system are stored.

Finally, click on **'Open MC110'** to open the configuration and start editing online.

Fig. 5: **Open MC110**

Note: When MC110 PDI is open, the unit changes to **Maintenance mode**.

The different 'buttons' that can be seen in the initial menu of the **MC110 PDI builder** are explained below.

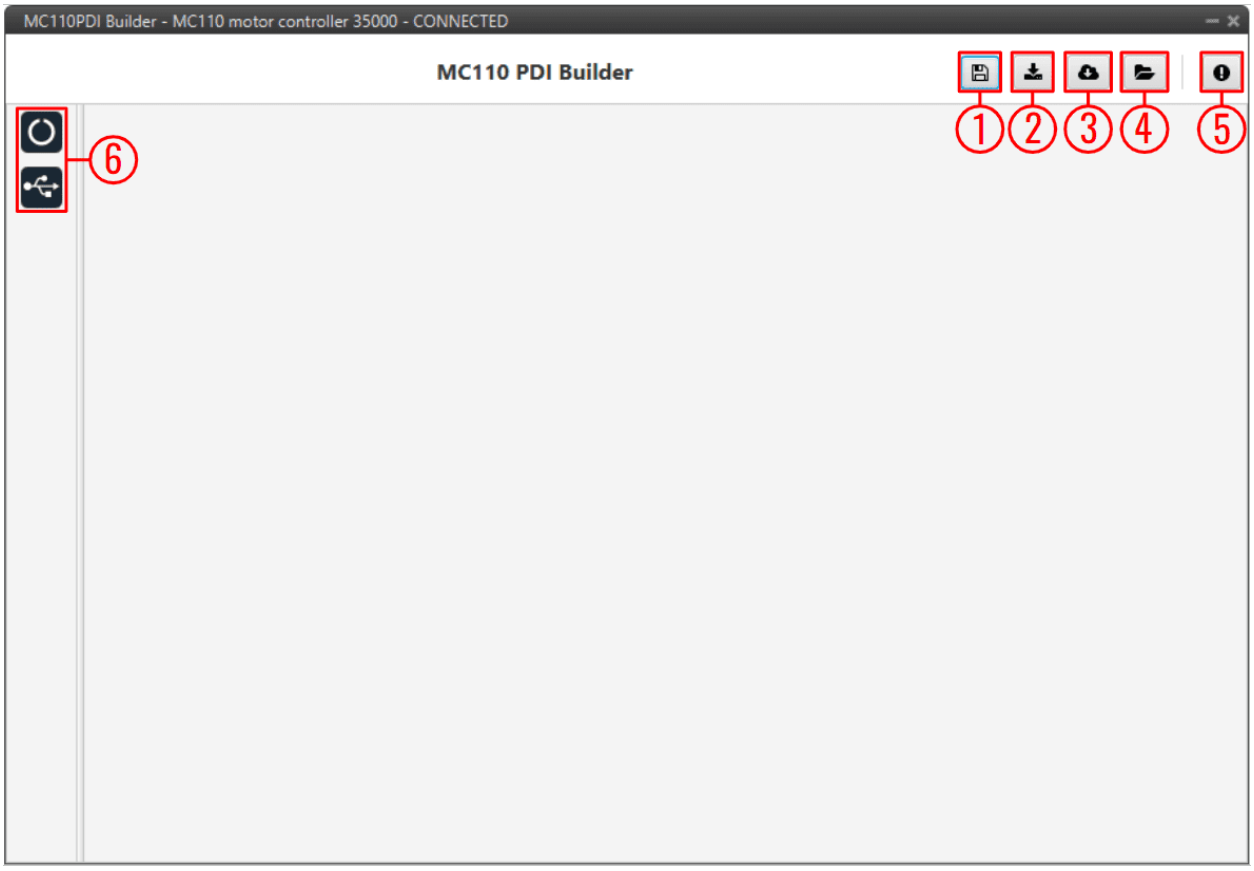


Fig. 6: **Initial menu**

1. **Save PDI**: After changes are done, press on the save button to apply the changes.

Note: This button will only appear if a MC110 is connected, i.e. when working offline this button will not be available.

2. **Export PDI**: After modifying a configuration, press the export button to store the configuration in the local storage.

   Users can store this configuration in an empty folder or in the folder where the previously imported configuration is stored. With the latter option, the "original" configuration will be overwritten by the one with the new changes.

3. **Import PDI from repo**: The user can import a configuration file from the repo and modify it. After that, if the save button is pressed, this configuration will be uploaded to the connected MC110.

4. **Import PDI from local storage**: The user can import a configuration file from the local storage and modify it. After that, if the save button is pressed, this configuration will be loaded into the connected MC110.

5. **Feedback**: Users can report a problem they have encountered by **creating an issue in their own 'Joint Collaboration Framework'**. The '**Download**' button downloads a zipped folder with the current **MC110** configuration and more information needed for Embention to resolve the issue. It is advisable to attach this folder when creating the issue.

Note: The user's '**Joint Collaboration Framework**' is simply a **own Github repository for each customer**.

If the user has any questions about this Joint Collaboration Framework, please see Joint Collaboration Framework user manual or contact sales@embention.com.

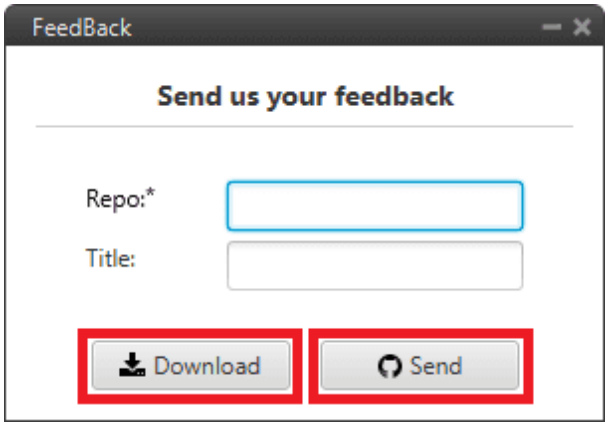


Fig. 7: **Feedback window**

6. These are the different functions of MC110. They are explained in the following menus:

   -  *MC*
   -  *Input/Output*

2.1 MC

2.1.1 Open Loop Startup

If it is not possible to start-up with the low speed PLL observer, it could be done by the open loop procedure:

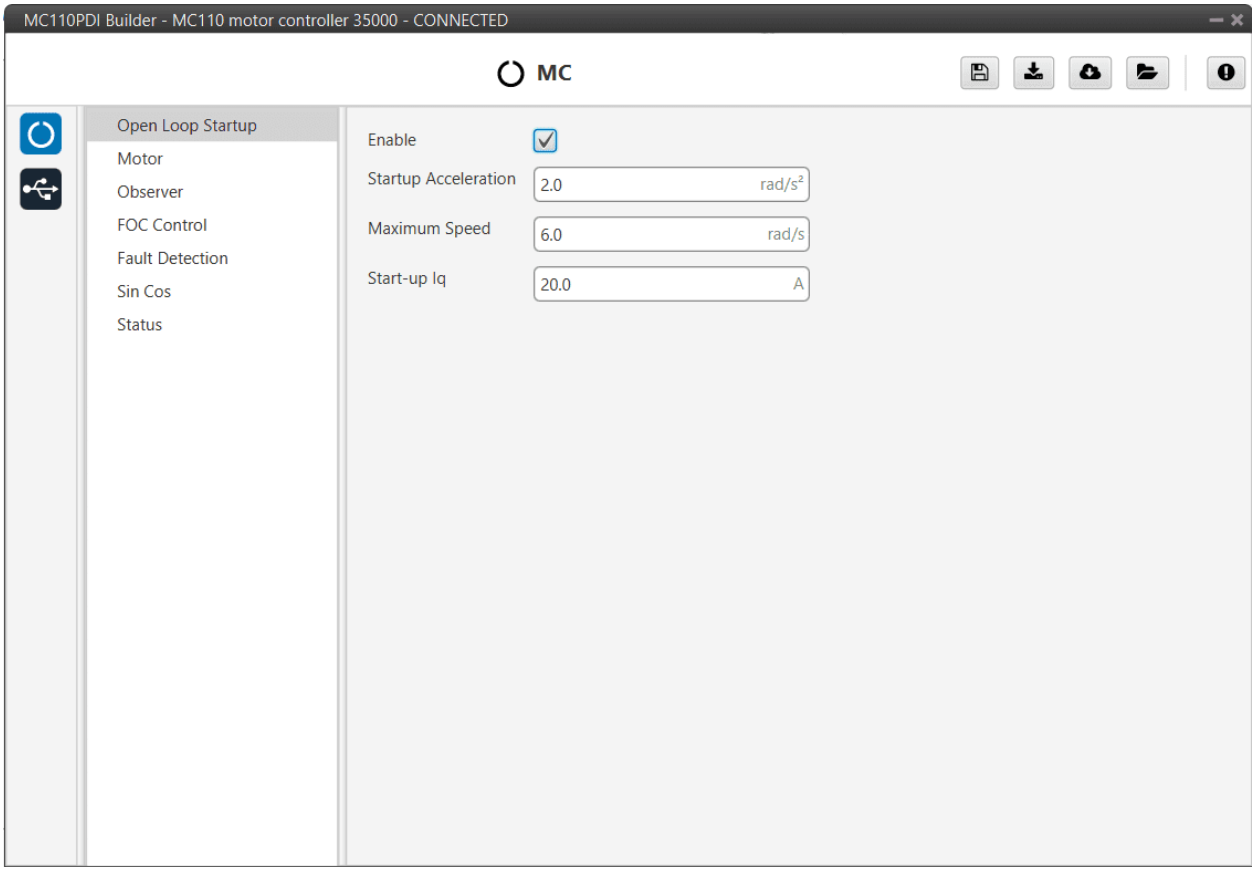


Fig. 8: **Open Loop Startup panel**

In this panel the parameters for open-loop start-up are configured:

- **Enable**: Enables the open-loop start-up.
- **Startup Accelaration**: This is the slope of the start ramp. It is usually a value that **allows for a 1-2 second start-up**.
- **Maximum Speed**: Maximum speed reached on the start ramp. It is usually approximately **10-15%** of the maximum speed of the mechanical motor.
- **Start-up Iq**: Since the speed controller is not working at this point, it is necessary to command a current I_{q0}^* to start the motor. This current is **constant**, and moreover, it is all active current I_q (of torque).

  As the user always wants to maximize the torque, the commanded reactive current I_d^* is **zero**. At constant speed, **the higher this value, the more torque the motor will have at start-up**. It is usually about **10%** of the maximum I_q value.

2.1.2 Motor

This tab sets all the electrical/mechanical parameters of the motor that will define the controller dynamics.

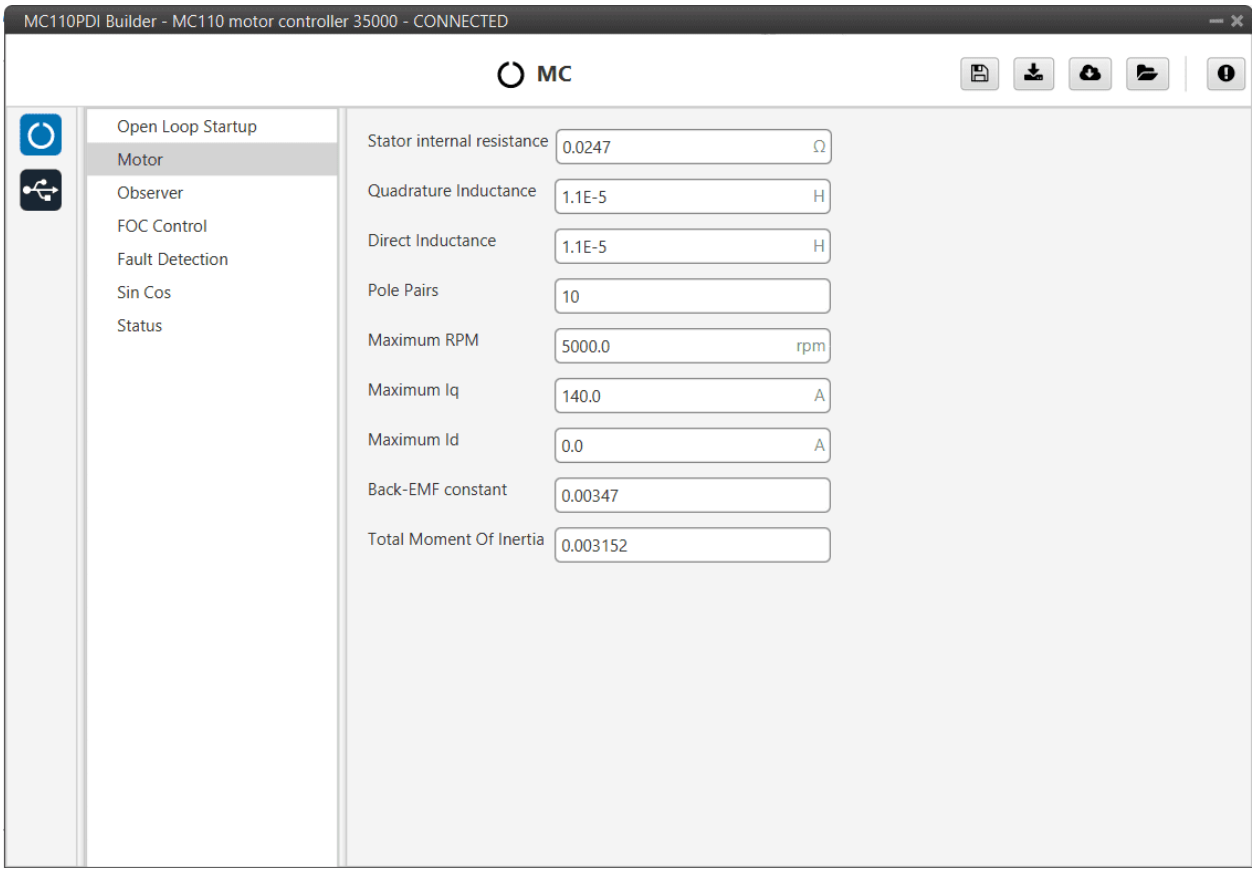


Fig. 9: **Motor panel**

- **Stator internal resistance**: This is the line-to-neutral resistance of the motor. Here it is necessary to take into account the harness resistance if it is not negligible. Expressed in Ohms (Ω).
- **Quadrature/Direct Inductance**: Line-to-neutral motor inductance in q-d axes. Expressed in Henries (H).
- **Pole Pairs**: The number of **pole pairs** shall allow calculation of the mechanical speed of the motor.
- **Maximum RPM**: Maximum desired mechanical speed that the motor can reach. It allows to calculate the input rate in terms of desired speed.

  This parameter is only used to determine the equivalence between the maximum command (PWM or CAN) and the commanded RPM in speed closed loop mode.
- **Maximum Iq**: Maximum permissible quadrature current that the motor can drive.
- **Maximum Id**: Maximum allowable direct current that the motor can conduct. Only used in the MTPA/FW scheme.
- **Back-EMF constant**: This is the line-to-neutral Back-EMF constant. Expressed in $\frac{V}{rad/s}$.
- **Total Moment Of Inertia** (MOI): Total moment of inertia (motor + load). Expressed in $kg \cdot m^2$.

Note: Back-EMF constant is an important parameter for low-speed PLL observer, so a correct characterization of this parameter will allow control in the low-speed range and start-up without open-loop procedure.

2.1.3 Observer

The algorithm is prepared to work with **two observers** (low and high speed observer) working in a predefined speed range.

Moreover, there are **two low-speed observers** that can be selected: **PLL observer** and **Arc-tangent observer** (this should be configured in the *FOC Control panel*). The first one allows starting without open loop while the second is usually selected when there is an open loop procedure.

The observer mixing limits [low,up] are usually approximately **2%** and **15%** of the maximum mechanical speed.

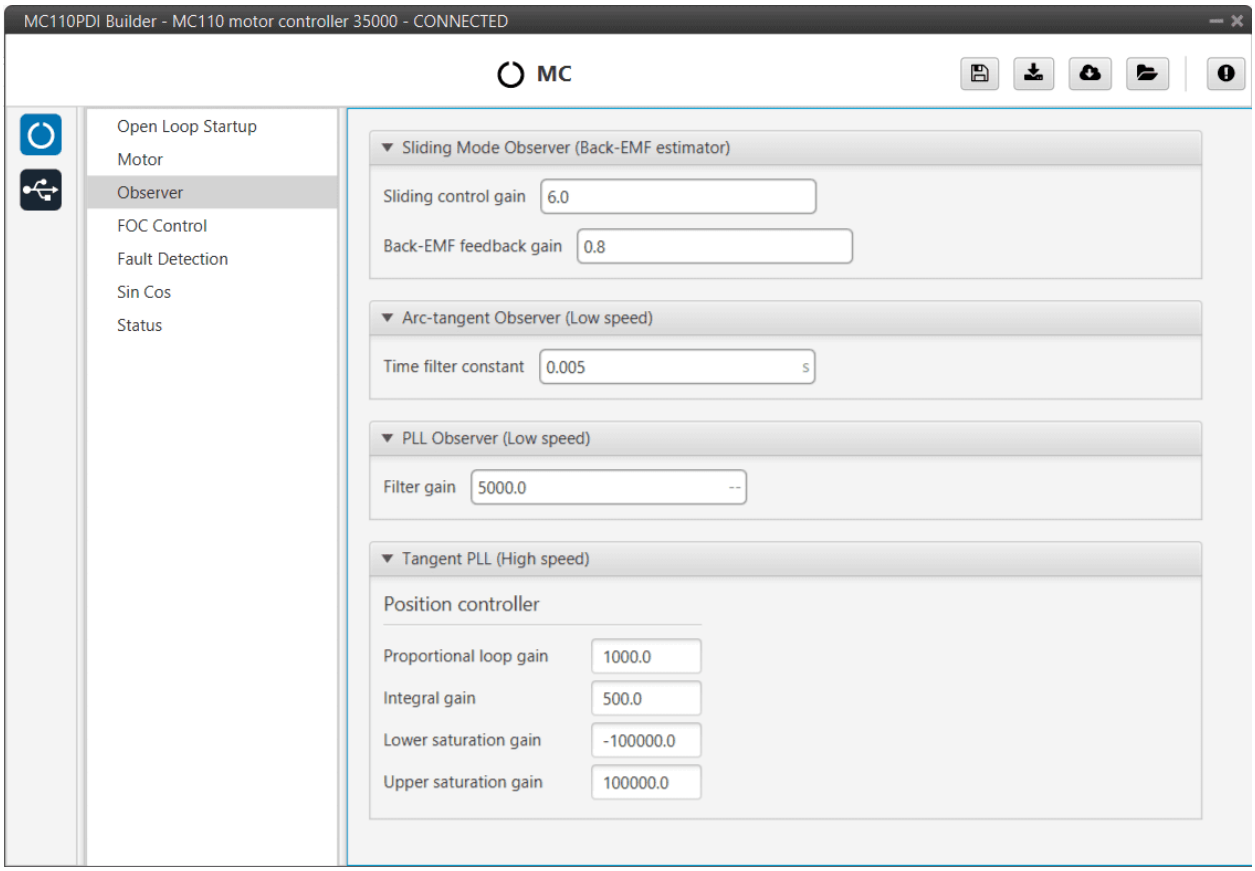


Fig. 10: **Observer panel**

Here the observers parameters must be configured:

- **Sliding Mode Observer (Back-EMF estimator)**: Sensor-less estimator for the Back-EMF.
  - **Sliding Control Gain**: This is the main gain of the observer.

    To ensure the stability of the algorithm, this value must be greater than the absolute value of Back-EMF on the alpha/beta axes.

  - **Back-EMF feedback gain**: This value is to account for unfiltered Back-EMF. It should be between 0 and 1.

- **Arc-tangent Observer (Low speed)**: It is one of the low speed position observers.
  - **Time filter constant**: This is the time constant of the first order filter of the speed calculation. **The lower value, the lower filtering**.

- **PLL Observer (Low speed)**: It is the second one low speed position observer.
  - **Filter gain**: Filter cutoff frequency.

Note: Users can choose one of the two low-speed observers (arc-tangent or PLL) by selecting it in the *low speed observer* parameter of the FOC control menu.

- **Tangent PLL (High speed)**: This is the high speed position observer. It is controlled by a *PI control*.

Warning: It is recommended to tune this estimator even if an external sensor is used for feedback (sensored control).

As it will be described in the next section, the control will automatically jump in case the primary feedback source fails to a sensorless control.

2.1.4 FOC Control

First, the PI (Proportional and Integral) controller is presented.

The form of the PI is the classical parallel form:

$$u = K_p \cdot \left(1 + \frac{1}{T_i} \cdot s\right)$$

Where **integral gain** refers to the quotient $\frac{1}{T_i}$. **Lower** and **upper saturation gain** are the **limits** to which the PI limit its output.

FOC Control is the main control menu, where all parameters related to FOC control are set.

The basic blocks that define the FOC control are three *PI* control loops that should be tuned with the motor characterization.

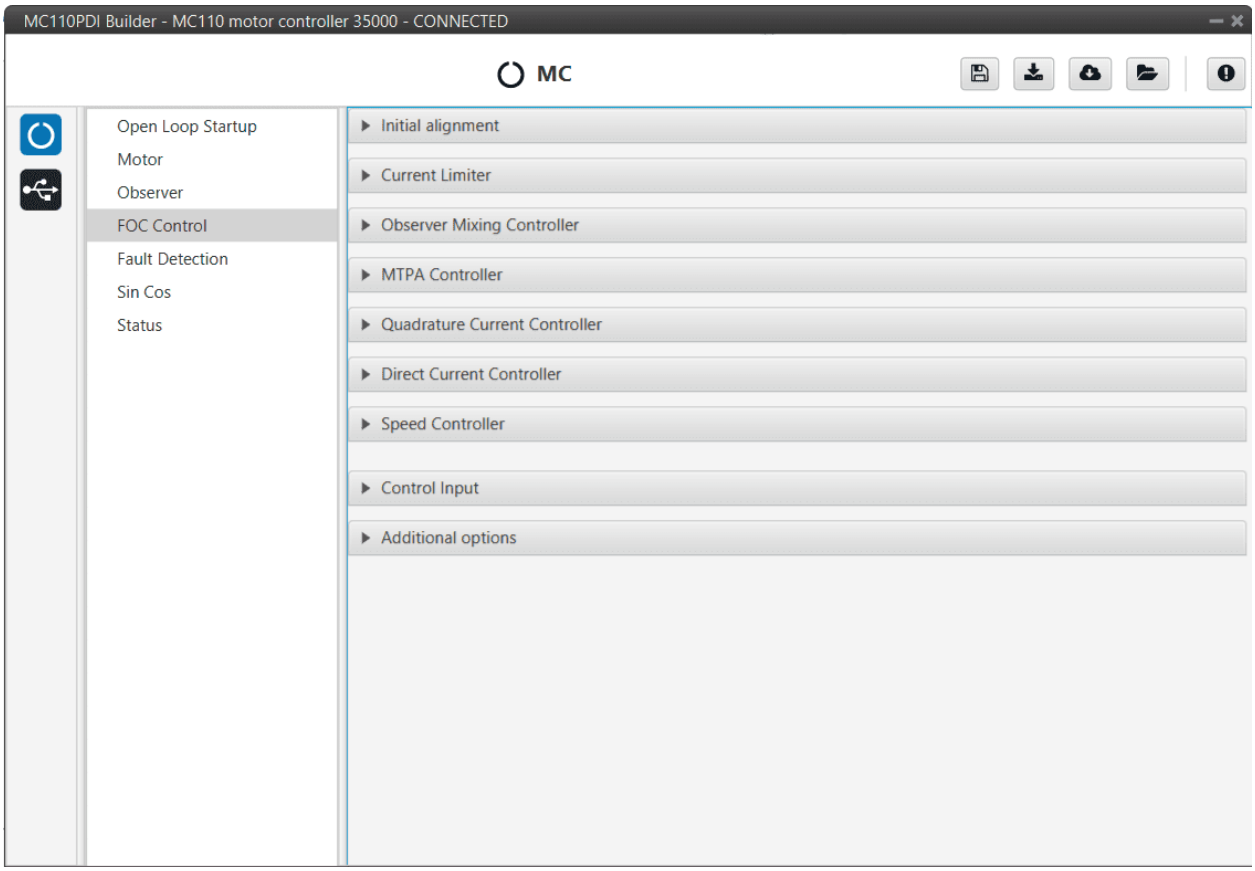


Fig. 11: **FOC Control panel**

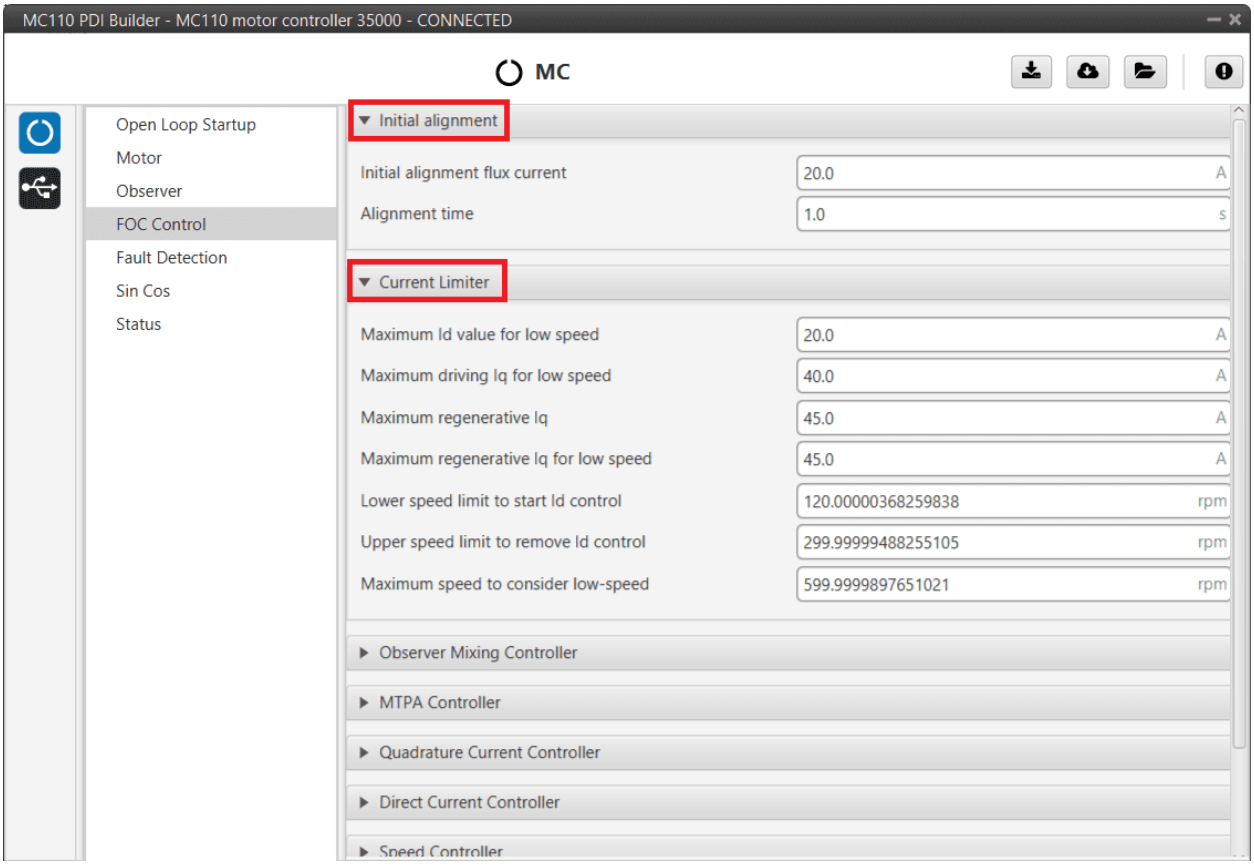


Fig. 12: **FOC Control - Initial alignment and Current Limiter parameters**

Note: These default parameters should work correctly.

- **Initial alignment**: Start-up initial alignment procedure.

  If the detection of the initial rotor position is not available or is difficult to obtain, it is desirable to lock the rotor to a known state (usually zero angle) at start-up. This procedure allows the rotor to always be aligned in the same electrical position.

  Note: This part of the configuration is essential if the open-loop is disabled.

  - **Initial alignment flux current**: This is the desired I_d current to align the rotor. It should be about **15%** of the maximum phase current of the motor.
  - **Alignment time**: This is usually about 1 second.

- **Current Limiter**: Limiter block for the desired currents I_q and I_d.
  - **Maximum Id value for low speed**.

- **Maximum driving Iq for low speed**.
- **Maximum regenerative Iq**.
- **Maximum regenerative Iq for low speed**.
- **Lower speed limit to start Id control**.
- **Upper speed limit to remove Id control**.
- **Maximum speed to consider low-speed**.

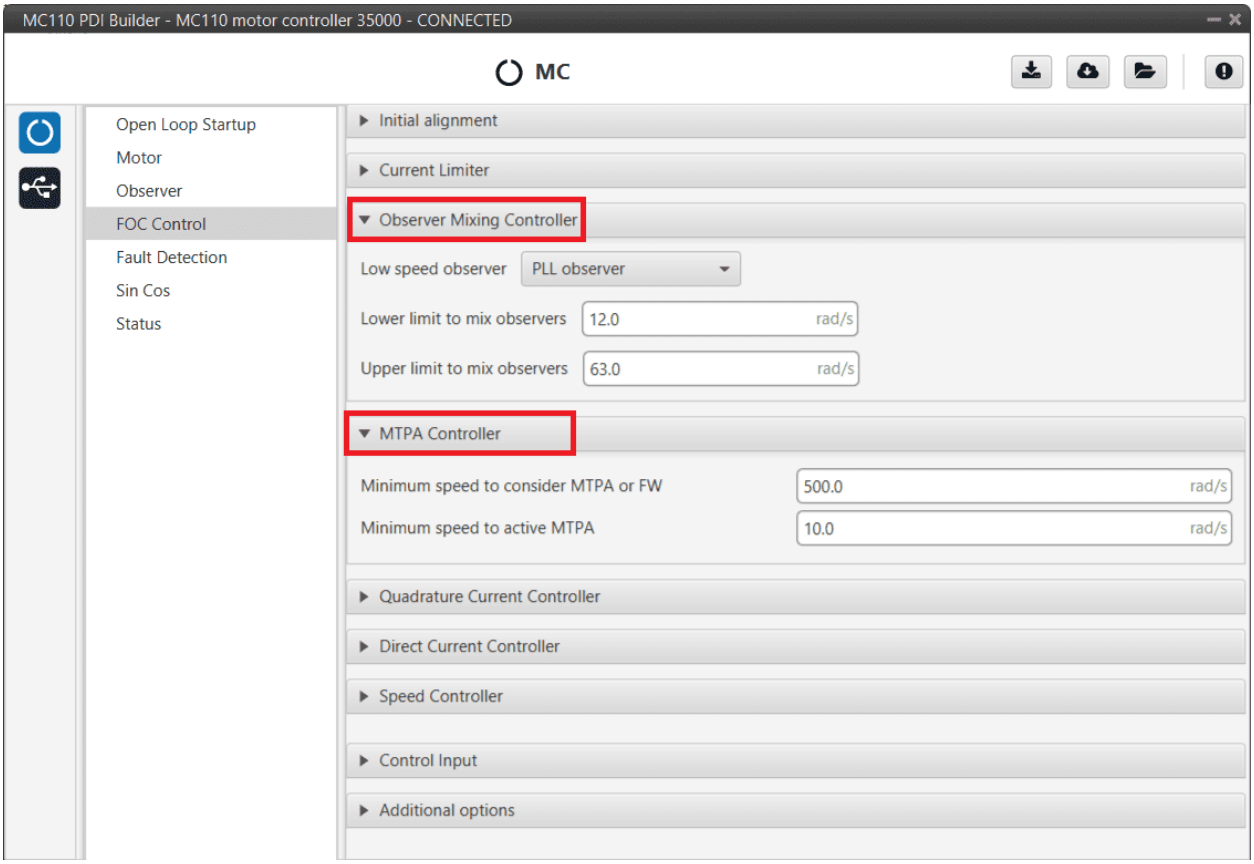


Fig. 13: **FOC Control panel - Observer Mixing Controller and MTPA Controller**

- **Observer Mixing Controller**: Definition of the range of low-high speed observers.
  - **Low speed observer**: Users should select the low-speed observer to be used: arc-tangent or PLL.
  - **Lower limit to mix observers**.
  - **Upper limit to mix observers**.

- **MTPA Controller**: Maximum Torque Per Ampere controller.

  It is used to reach a higher speed than the base speed of the motor. In addition, it can work in a Field Weakening to control with I_d.

  - **Minimum speed to consider MTPA or FW**.
  - **Minimum speed to active MTPA**.

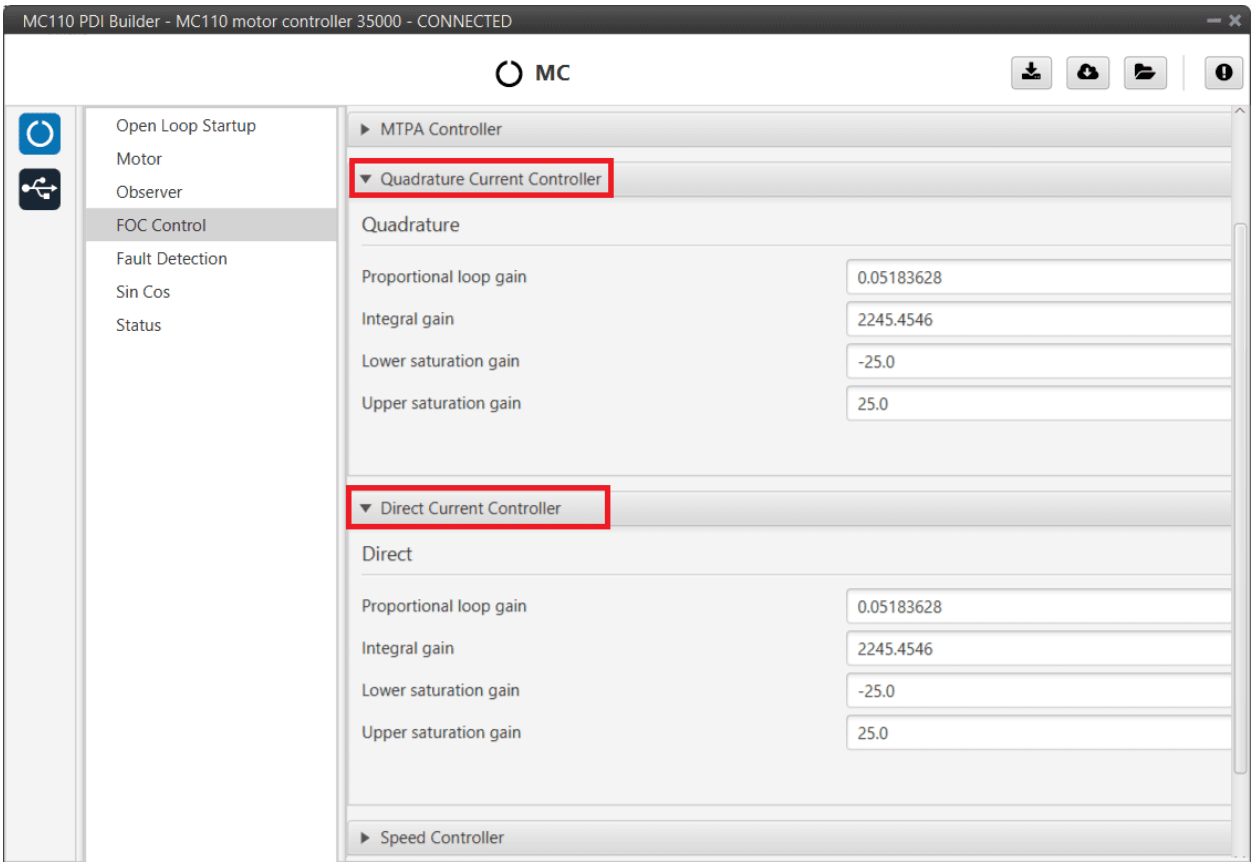


Fig. 14: **FOC Control panel - Quadrature Current Controller and Direct Current Controller**

Both quadrature and direct current PI control must be defined. Also, the quadrature and direct PI gain settings are usually the same, but it will depend on whether the motor parameters are the same on both axes.

- **Quadrature Current Controller**: Torque regulator. It is controlled by a *PI control*.
- **Direct Current Controller**: Flux regulator. This is controlled by a *PI control*.

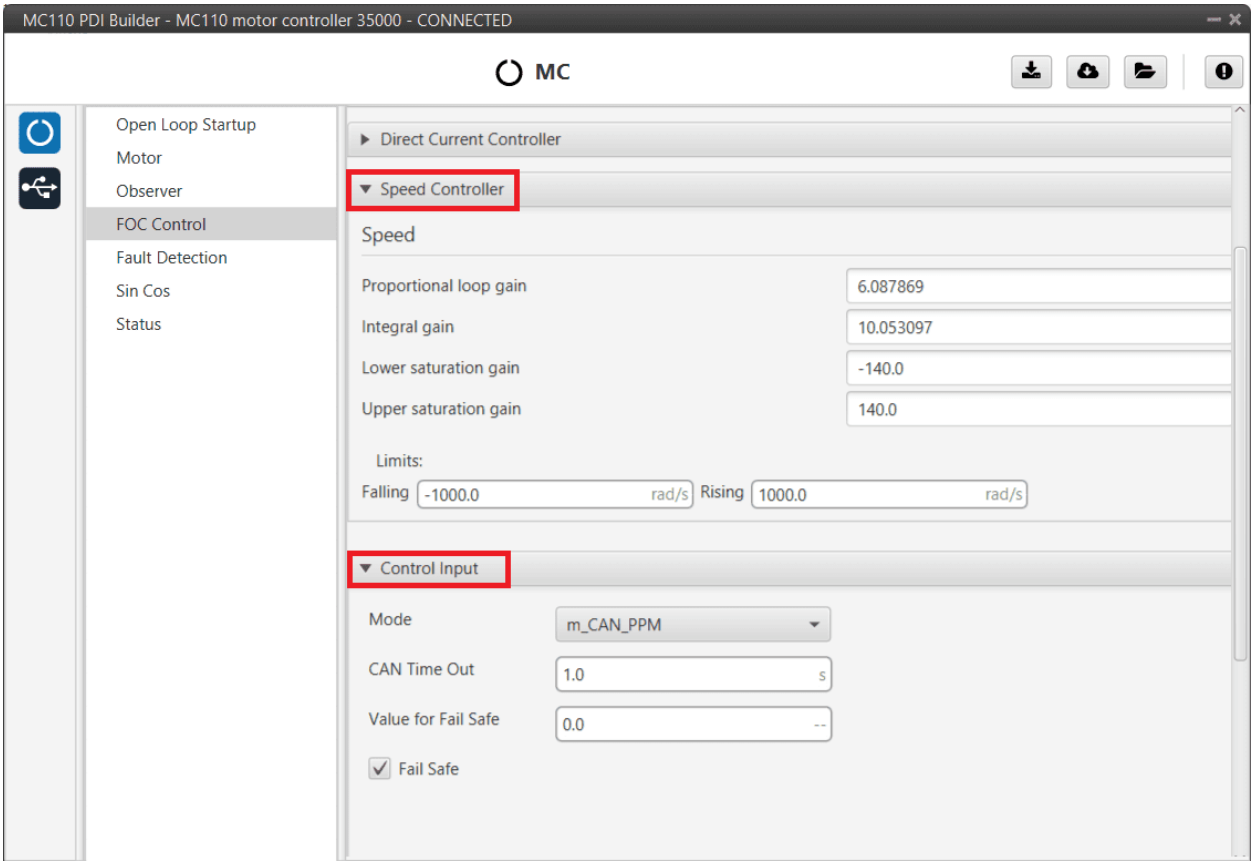


Fig. 15: **FOC Control panel - Speed Controller and Control Input**

- **Speed Controller**: It is controlled by a *PI control*.
  - **Limits**: Maximum desired motor acceleration.
    - Falling: Decceleration rate. Expressed in rad/s^2.
    - Rising: Acceleration rate. Expressed in rad/s^2.

- **Control Input**: Control input allows the user to select the input source and several additional parameters such as:
  - **Mode**: Control mode. The available options are **m_PPM**, **m_CAN** and **m_CAN_PPM**.
  - **CAN Time Out**: Whenever this timeout is met, the control input will switch to the next option if available. If none of them are available, it shall end up in failsafe mode.
  - **Value for Fail Safe**: Value between **0 and 1**. To be written in case the previous mode options are not available.
  - **Fail Safe**: Activates failsafe mode.

  The input flow is the following in case the mode **m_CAN_PPM** is selected:

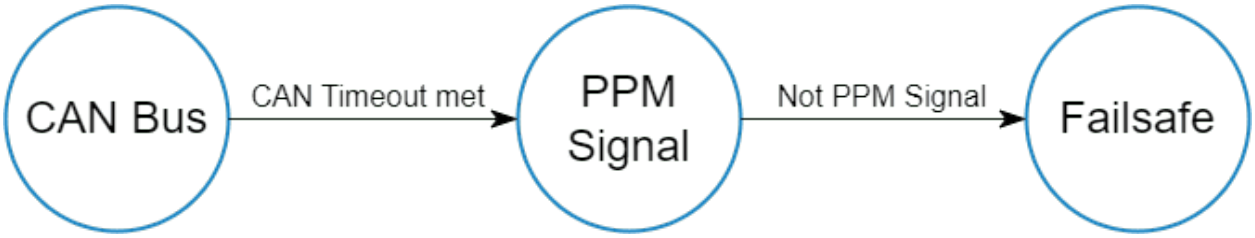


Fig. 16: **CAN_PPM mode diagram**

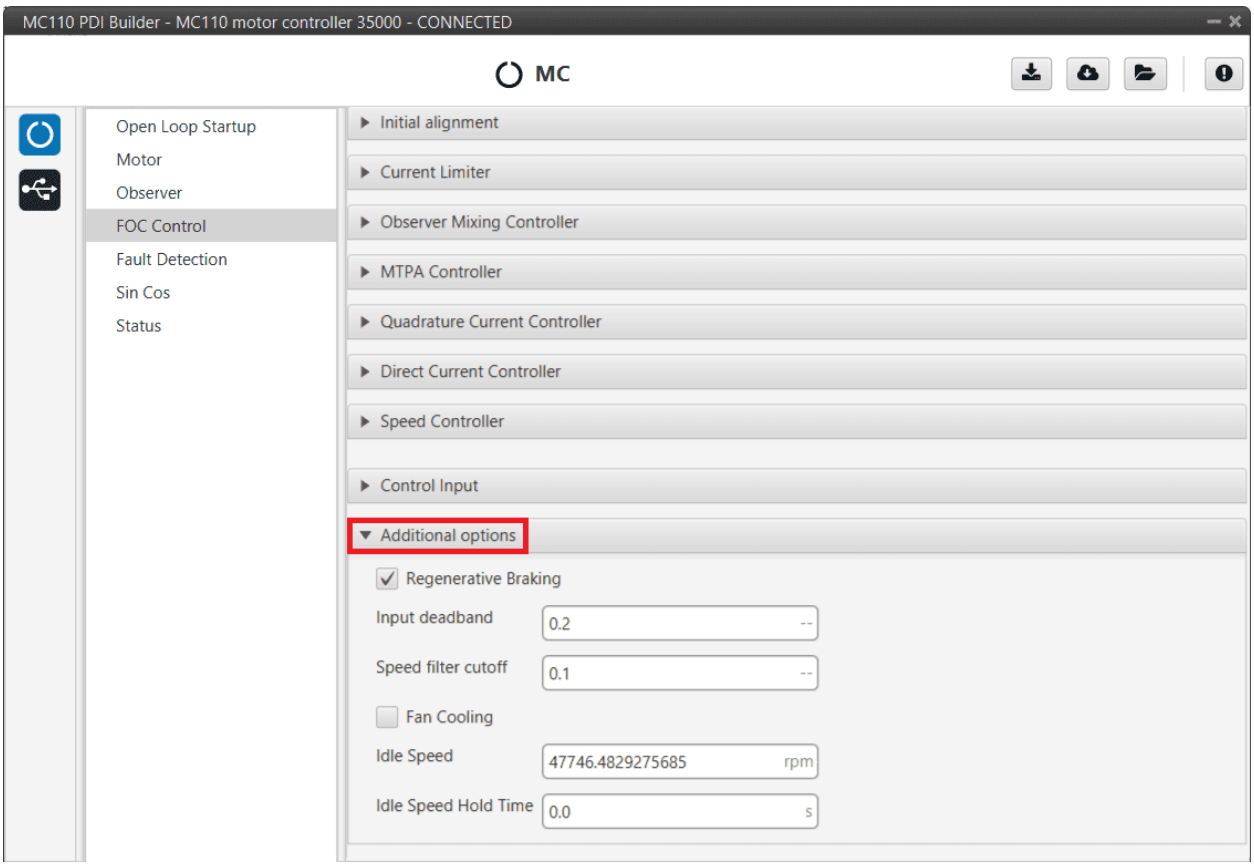


Fig. 17: **FOC Control panel - Additional options**

- **Additional options**:
  - **Regenerative Braking**: Activates regnerative braking feature which is basically consisting of thoughing negative current back to battery whenever the motor is loosing speed (it uses the kinetic energy to charge de battery).

    If disabled, it will only allow slow down due to losses.

> **Warning:** Activating this option is only recommended with batteries (charging them when braking), but not with a power supply, as they are not prepared to receive current (it is necessary to activate the sink mode to work with it).

- **Input deadband**: This is the value of the *input rate* at which the motor will start to move.

  For example, it may be desired to be non-zero in case an RC stick outputs about 0.2 duty cycle by default.

  Note: In case CAN Bus is used to command (see *CAN I/O - Input/Output* section of this manual), this deadband can be calculated as $\frac{RPM^*}{RPM_{max}}$.

- **Speed filter cutoff**: Cutoff frequency (expressed in Hz) to be applied only to **Hall effect sensors** and **SIN/COS sensors** filters.
- **Fan Cooling**: This option activates a PID control on a cooling fan to reach normal operating temperature.
- **Idle Speed**: Minimum motor speed for switching OFF.

  If an *input rate* lower than the *input deadband* is commanded, until this speed is reached (*idle speed*), the MC110 shall not switch to OFF mode.

- **Idle Speed Hold Time**: After the above conditions have been met (speed lower than idle speed) and this time (*idle speed hold time*) has elapsed, the MC110 shall switch to off mode.

  This is done to avoid operator input rate errors.

Finally, the following state machine will be executed:

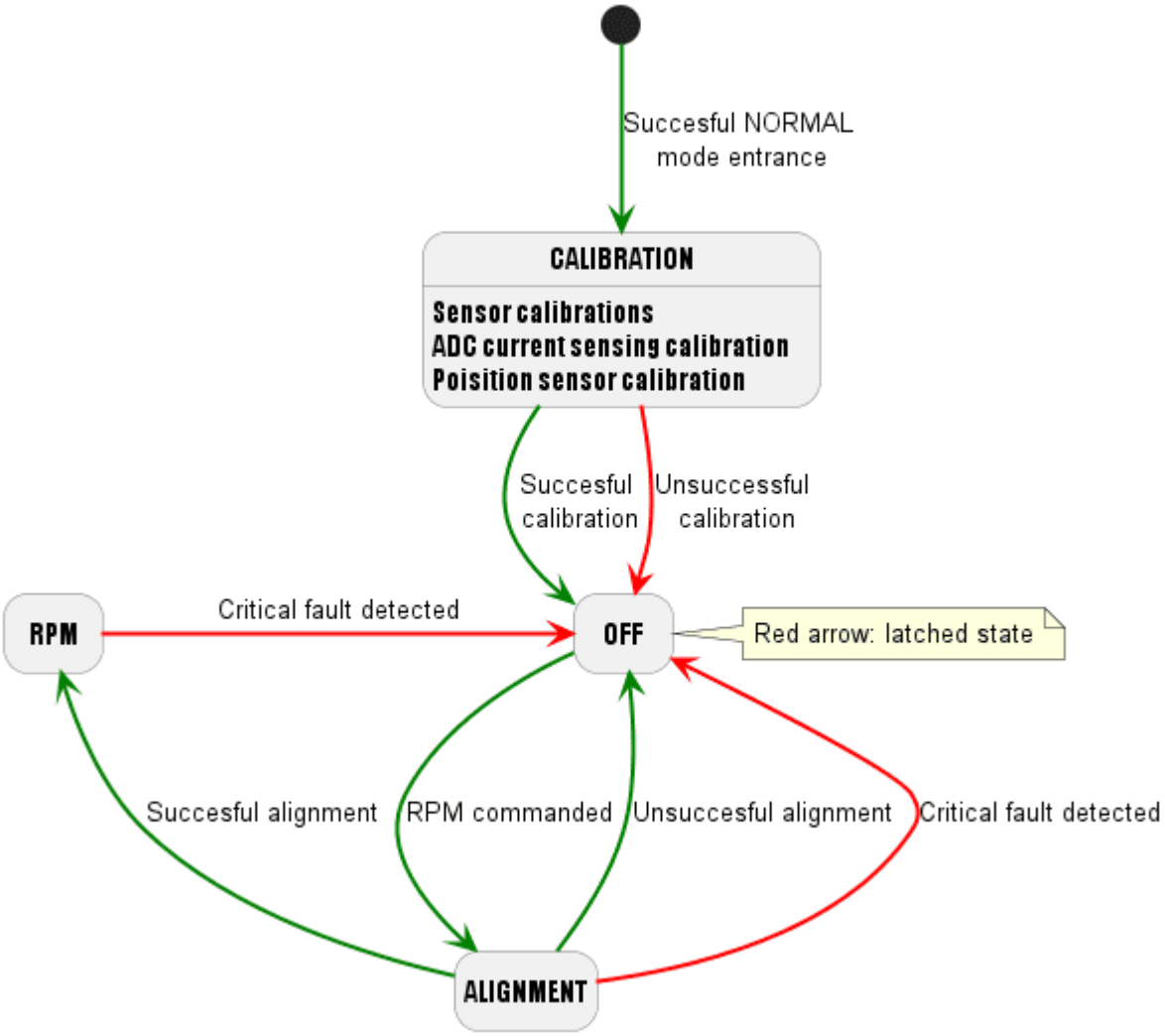


Fig. 18: **State machine diagram**

2.1.5 Fault Detection

Active-passive fault management. There are two types of failures:

- **Critical fault**: It will be followed by a latched OFF state.
- **Non Critical fault**: It will be only followed by a warning.

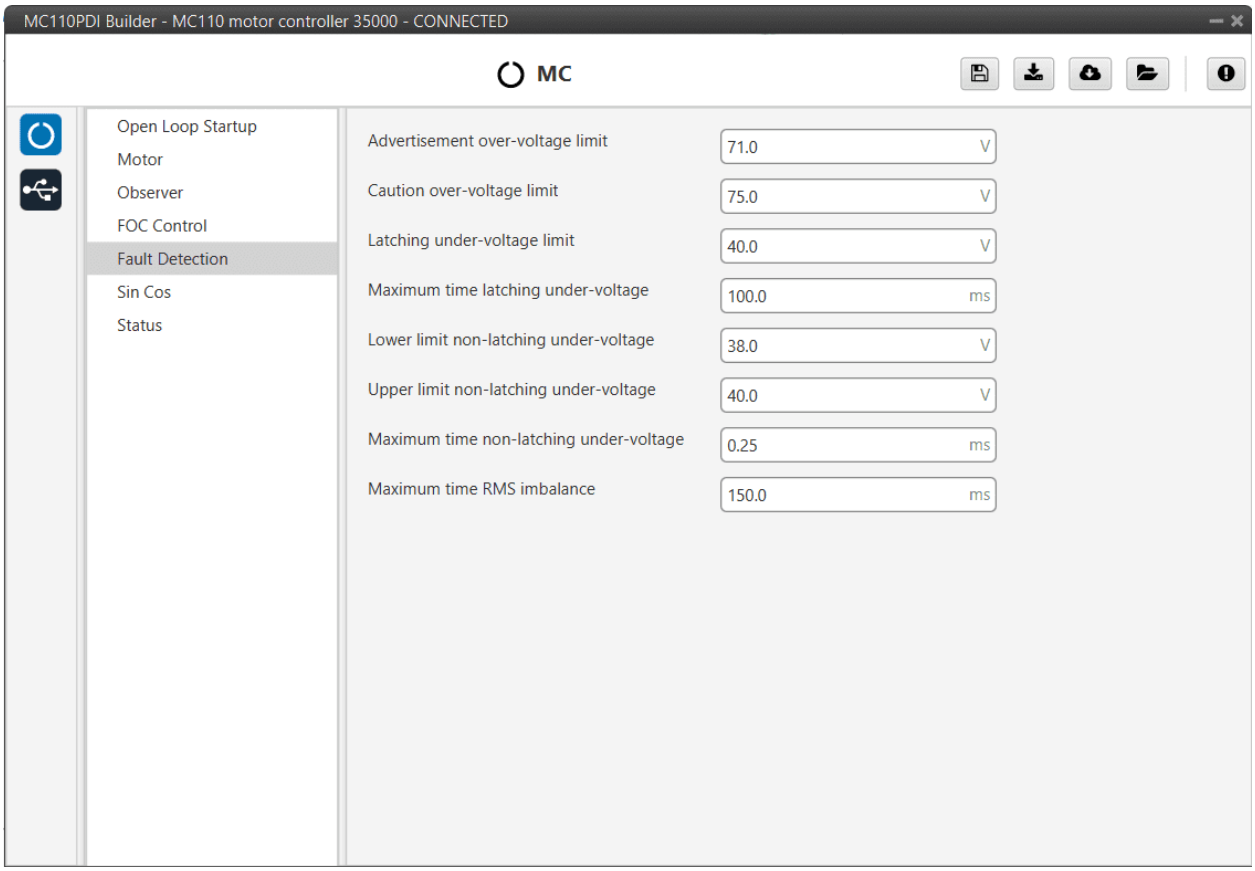


Fig. 19: **Fault Detection panel**

- **Advertisement over-voltage limit**: Non Critical fault.
- **Caution over-voltage limit**: Non Critical fault.
- **Latching under-voltage limit**: Critical fault.
- **Maximum time latching under-voltage**.
- **Lower limit non-latching under-voltage**: Non Critical fault.
- **Upper limit non-latching under-voltage**: Non Critical fault.
- **Maximum time non-latching under-voltage**.
- **Maximum time RMS imbalance**: Critical fault.

Note: There are more fault detections but cannot be configured because it depends on the limits of the ESC.

2.1.6 Sin Cos

In case it is necessary to use an external SIN/COS sensor as a source of electrical angle/speed feedback, the main interface for doing so is via ADC inputs.

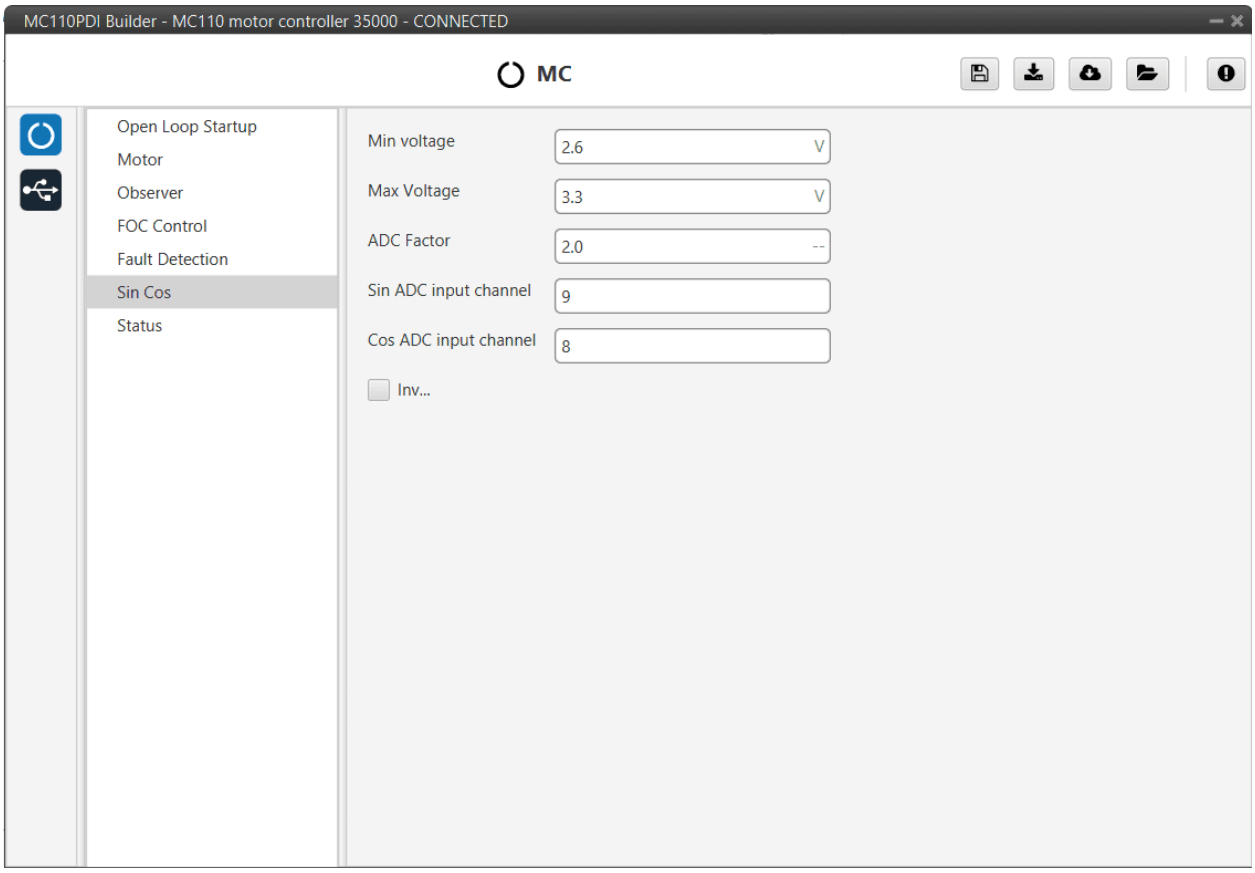


Fig. 20: **Sin Cos panel**

This menu allows users to customize the main characteristics of their sensor:

- **Min Voltage**: Minimum signal voltage. Measured in Volts (V).
- **Max Voltage**: Maximum voltage of the signal. Measured in Volts (V).
- **ADC factor**: Factor that multiplies the ADC port reading, especially useful if a voltage divider (or any other signal level adapter) is installed.
- **Sin ADC input channel**: ADC channel to which the Sin signal is connected.
- **Cos ADC input channel**: ADC channel to which the Cos signal is connected.
- **Invert**: Inverts the resulting electrical angle. Useful for correcting installation inversions.

2.1.7 Status

- **Enable VCP Status Message** enables the periodic sending of the status message that Veronte Link uses to recognise the Veronte Motor Controller MC110.
- **Period**: Enter a desired period to send repeatedly the status message.

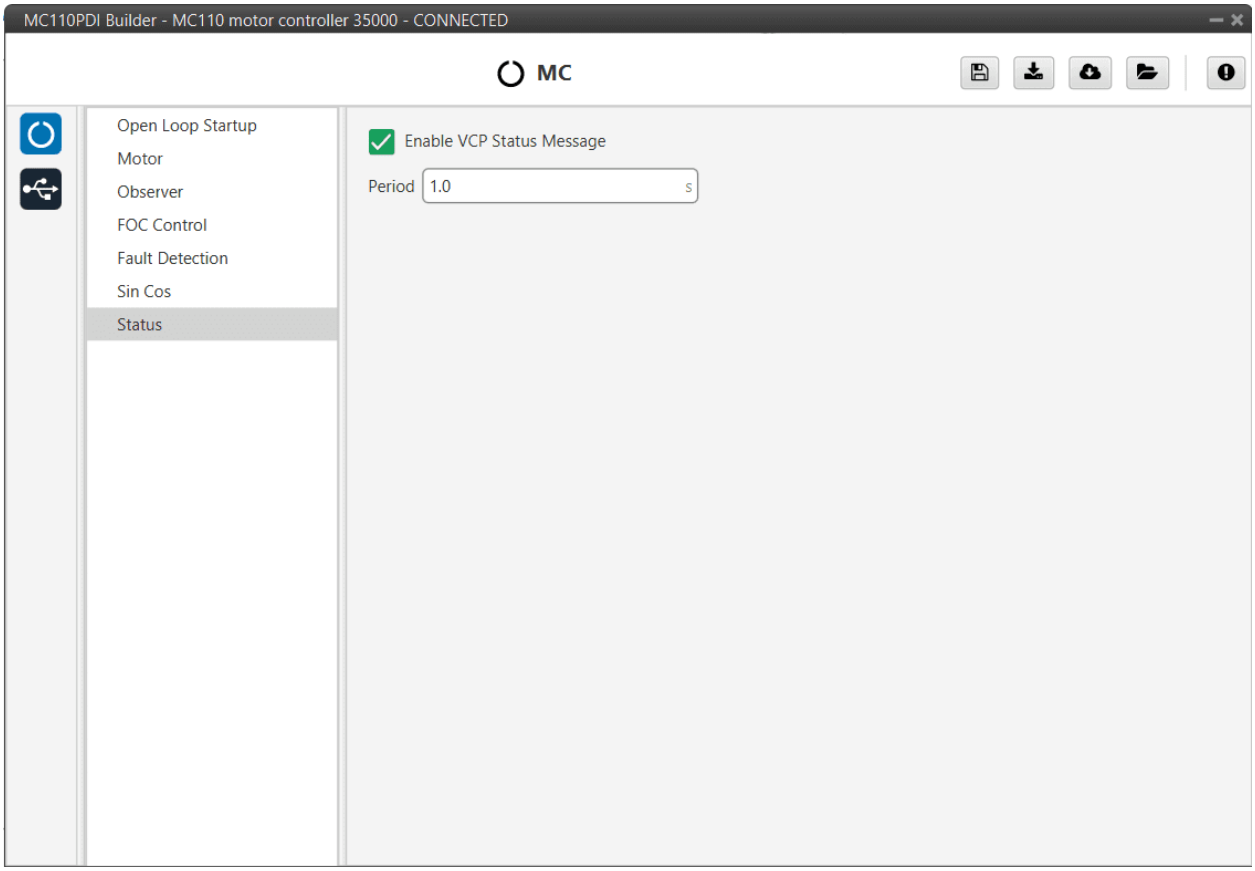


Fig. 21: **Status panel**

Note: VCP is the Veronte Communication Protocol. To know more, read the VCP user manual.

2.2 Input/Output

2.2.1 I/O Setup

In this panel the user can establish the relationship between a determined signal with a I/O port and the duties they are performing. This allows users to configure external sensors, custom messages, etc.

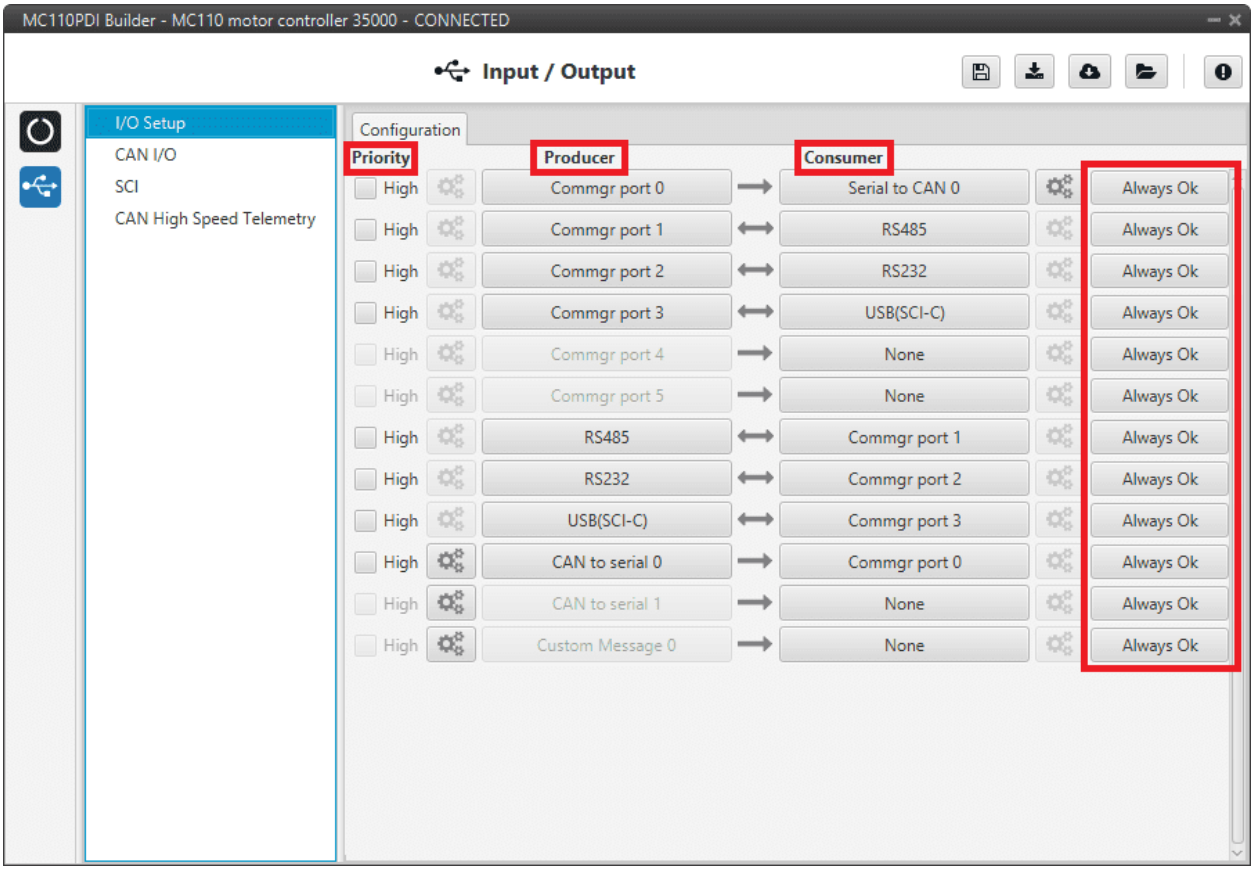


Fig. 22: **I/O Setup panel**

- **Priority**: Connections between I/O ports can be marked with high priority with this checkbox. If **enabled**, they will **run at high frequency: 1000 Hz**.
- **Producer**: Functions for creating and sending messages.
- **Consumer**: Functions for receiving and parsing messages.
- **Bit**: This assigns each connection a bit in a way that allows this connection to be activated/deactivated depending on the status of the selected bit.

  By default, the 'Always Ok' bit is set to all connections so that they are always active.

Firstly, user has to configure the **Producer** selecting the I/O port or information to use. Later, user has to configure the **Consumer** by clicking on an element, a new window will be displayed to select an item. The relationship between them can be unidirectional (Bind $\rightarrow$) or bidirectional (Bind Bidirectional $\leftrightarrow$), the last enables a port to receive or send information.

The following I/O ports are available:

Field	Description
RS232	Serial Port 232 (SCI-B)
RS485	Serial Port 485 (SCI-A)
USB (SCI-C)	USB Port
Commgr port	COM Manager ports send and receive VCP messages. This is the protocol used by Veronte products to communicate. For more information on VCP, read the VCP user manual
Custom Message producer	This allows user to send a serial custom message, see *Serial custom messages* section
CAN to serial / Serial to CAN	**Serial to CAN** sends serial streams over a CAN Bus / **CAN to serial** undoes the transformation 'Serial to CAN'

More information about some elements can be found in the following sections.

2.2.1.1 Serial Custom Messages

Warning:

- MC110 has a **serial limitation** of **64 vectors** (fieldset).

  In addition, there is a limit shared with **CAN Custom Messages**:

  - Maximum number of **vectors** (fieldset): **104**
  - Maximum number of **fields**: **2000**

It is possible to configure the messages sent through the serial port and its conversion to system variables by selecting the option **Custom Message** and configuring the I/O port.

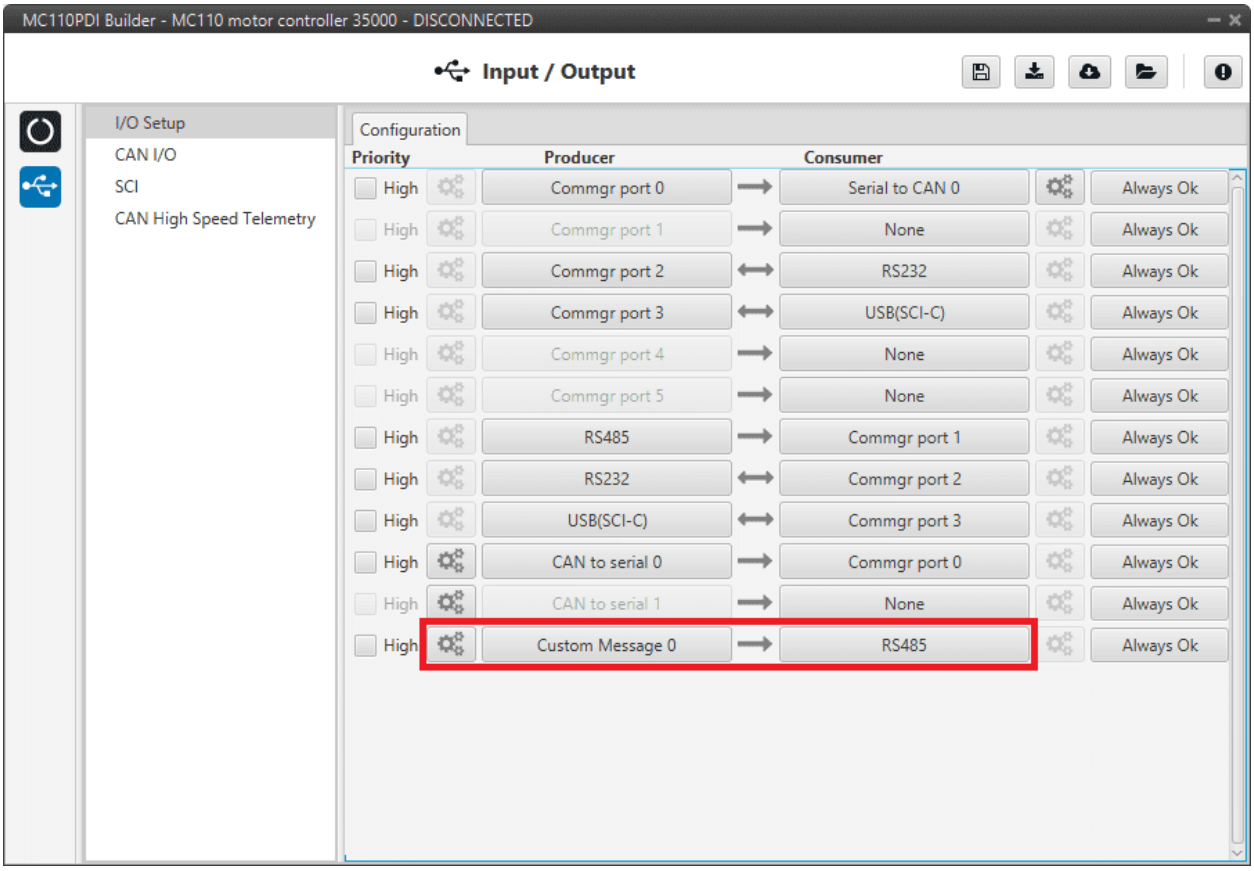


Fig. 23: **Serial Custom Message**

To configure a serial custom message, the user must follow the next steps:

1. Press the **configuration button** ( icon) and a pop-up window will be displayed.

   In this window press the  icon to add a custom message, the user can choose between **System variables**, **ADSB Vehicle** or **External Sensor**.

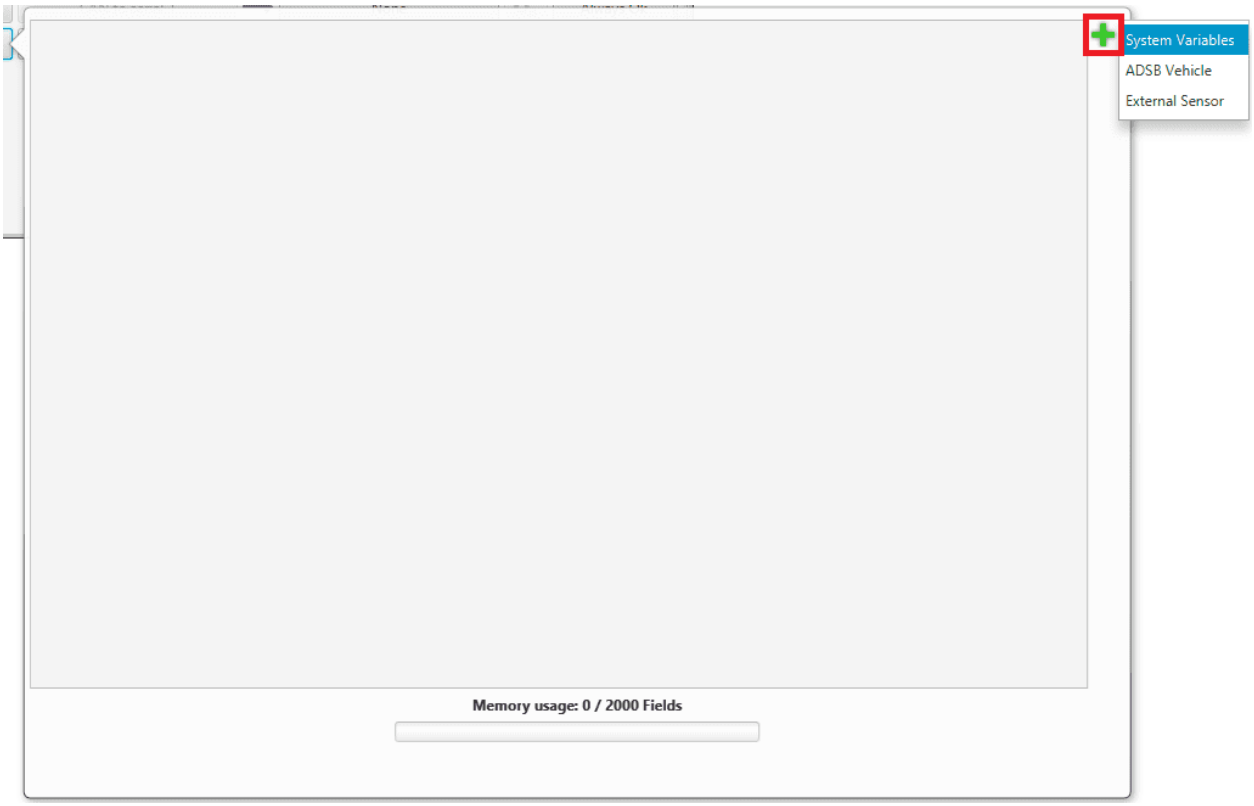


Fig. 24: **Serial Custom Message configuration**

Note: The difference between choosing **System Variables**, **ADSB Vehicle** or **External Sensor** is that when the user selects **Variable** as the custom message type, only system variables will appear when *System Variables* is selected, only ADSB variables when *ADSB Vehicle* is selected and only variables related to external sensors if *External Sensor* is selected.

---

2. When it is already added, the following options are available to configure a custom message:

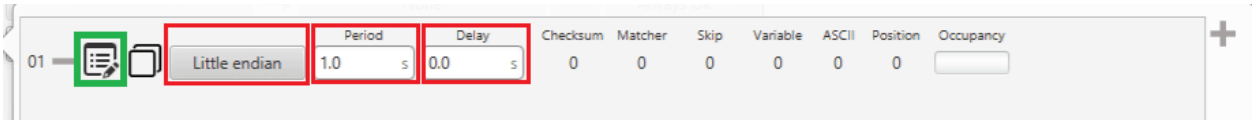


Fig. 25: **Producer Serial Custom Message configuration**

- **Endianness**: Depending on the order in which the device outputs the message, it is possible to select:
  - **Big endian**: Set the value from left to right.
  - **Little endian**: Set the value from right to left.
  - **Mixed endian**: Some devices use this format. If users need to configure it, please contact the support team (create a ticket in the customer's **Joint Collaboration Framework**; for more information, see Tickets section of the **JCF** manual).
- **Period**: It is the inverse of the send frequency.

- **Delay**: It is a delay applied before sending the message. This serves to send messages with the same period without overloading the serial bus.

3. To create the structure of the message, click on the **edit message button** (icon) and then press the icon to add fields to it.

   The following type of messages are available to configure a structure: **Variable**, **Checksum**, **Matcher**, **Skip** and **Parse ASCII**.

   The configuration of each structure is covered in Custom Messages types - Input/Output section of the **1x PDI Builder** user manual.

Warning: Before configuring any message, user has to know the structure it has to have according to the device that is connected to the port. Each device may have a different message structure when it sends or receives information.

2.2.2 CAN I/O

A CAN (Controller Area Network) Bus is a robust standard communication protocol for vehicles widely used in the aviation sector. MC110 has two CAN buses that can be configured independently.

The structure of a CAN message can be seen in the following image:

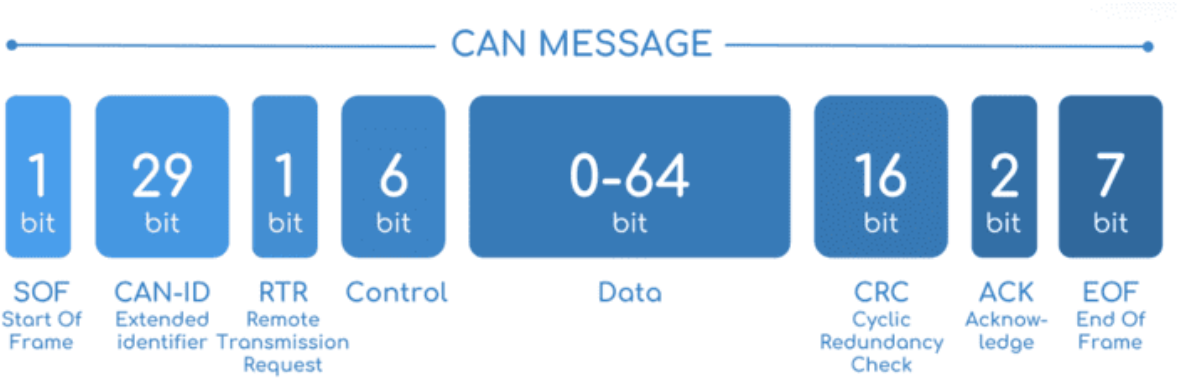


Fig. 26: **CAN message structure**

Only the ID is introduced in the system, the rest of the message layout is already coded. The data field is built by the user to send, and parsed when received.

The baud rate of both CAN buses can be configured in the *Mailboxes panel*.

2.2.2.1 Configuration

This menu allows the configuration of the CAN inputs and outputs.

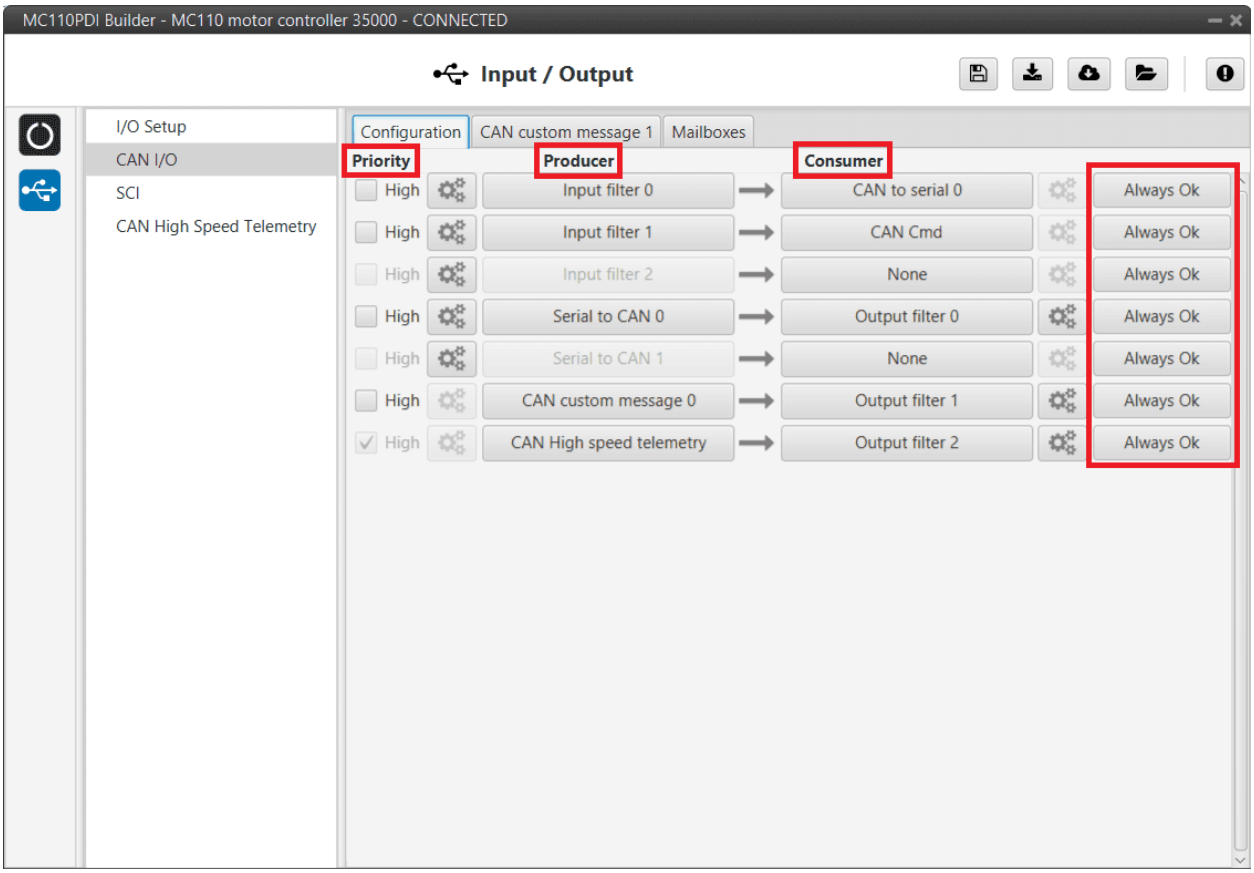


Fig. 27: **CAN configuration panel**

In this menu, the user can find the same 'columns' (**Priority**, **Producer**, **Consumer** and **Bit**) as in the *I/O Setup panel*.

> **Warning:** In CAN, in Low state the specified period is not guaranteed but in High state it is.
>
> However, only those **messages that are critical for external devices** should be set as **high priority**, as this may disrupt the proper functioning of Veronte MC110.

MC110 has the following list of **producers**:

- **Input Filter**: Those CAN messages received in one filter can no longer be received in subsequent filters.

  The following parameters need to be configured by clicking on  :

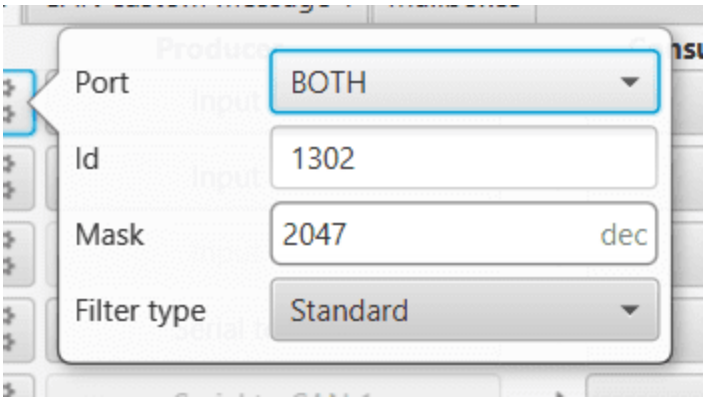


Fig. 28: **Input Filter configuration**

- **Port**: It is required to configure the CAN bus from which it listens, the user can choose between *CAN A*, *CAN B* or *BOTH*.
- **Id**: CAN Id must be set and it is used to identify messages. The value set has to be decimal format.
- **Mask**: A CAN Id mask can be set to filter messages. The mask defines the bits that should match.

  For example, to admit standard Ids (11 bits) from 8 to 11 (100 to 111 in binary) the user should set the mask to binary 11111111100, that is 2044 in decimal.

> **Warning:** Make sure that mask is set properly to be able to receive the desired CAN messages.

- **Filter type**: The available options are *Standard* (frame format with a 11-bit identifier), *Extended* (frame format with a 29-bit identifier) and *Both*.

- **Serial to CAN**: Serial messages through CAN output, it has to be connected to *I/O Setup* **consumer**. It can be configured in  , a pop-up window will appear:

> **Warning:** For correct communication, mark it as **High priority** (with the Priority checkbox).

Fig. 29: **Serial to CAN configuration**

- **Id**: CAN Id must be set and it is used to identify messages. The value set has to be decimal format.
- **Extended**: If it is enabled, the frame format will be 'Extended', i.e. with a **29-bit identifier**. Otherwise, the frame format 'Standard' (11-bit identifier) is set by default.
- **Time out**: This is the threshold time between receptions to consider that it is not being received correctly.

- **CAN custom message 0**: CAN custom messages transmission. They are configured in the next section: *CAN custom message*.
- **CAN High speed telemetry**: Telemetry messages sent via CAN faster than normal. They are configured in the next section: *CAN High Speed Telemetry*.

The **consumers** are explained in the following list:

- **Output Filter**: CAN output filters. The user can choose between **CAN A**, **CAN B** or **BOTH** in  .

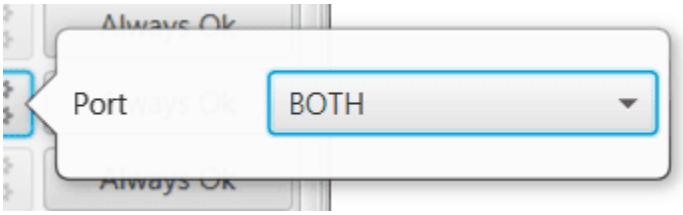


Fig. 30: **Output Filter configuration**

- **CAN to Serial**: This undoes the 'Serial to CAN' action, it has to be connected to *I/O Setup* **producer**.

> **Warning:** For correct communication, mark it as **High priority** (with the Priority checkbox).

- **CAN Cmd**: This is the input ESC expects to command RPM to the motor. It has to be connected to an **Input Filter**.

> **Warning:** This configuration is already done by default. If the user changes this ID, make sure that the ID of the mailboxes also matches.

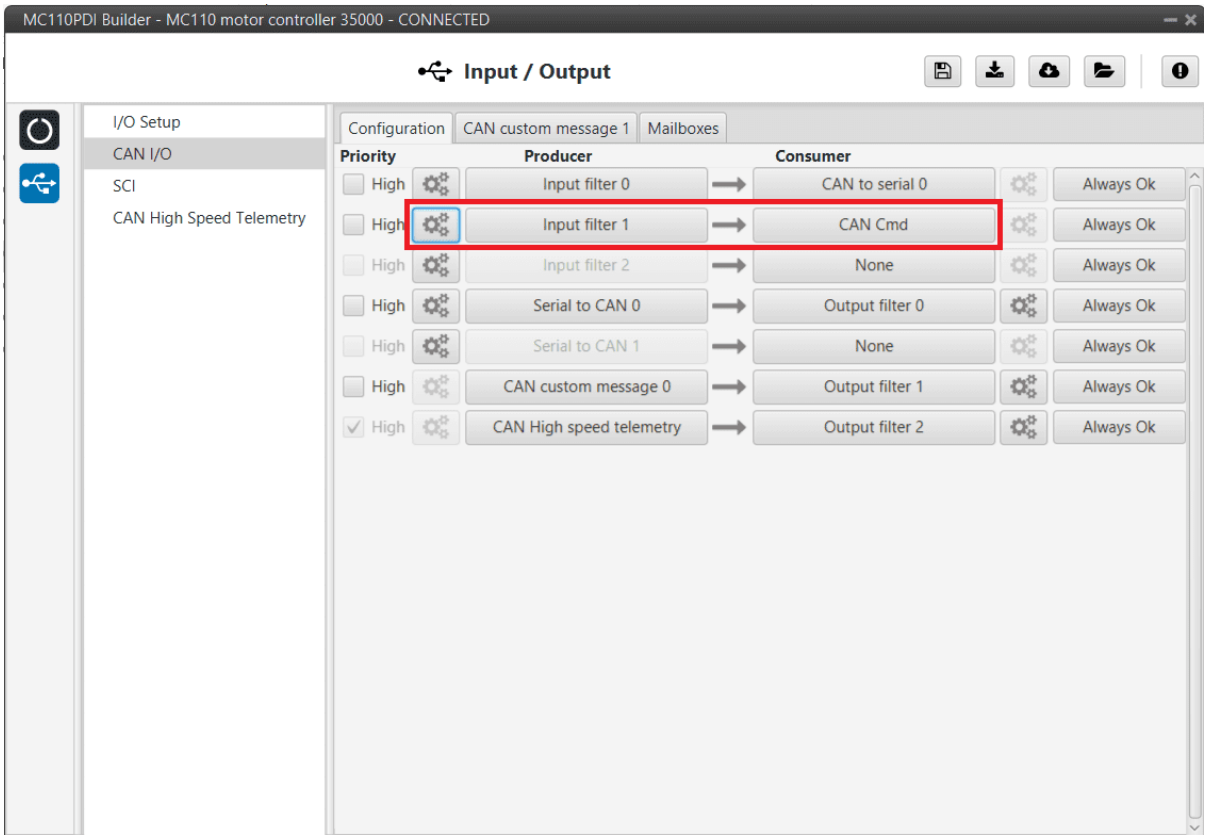


Fig. 31: **CAN configuration - CAN Cmd**

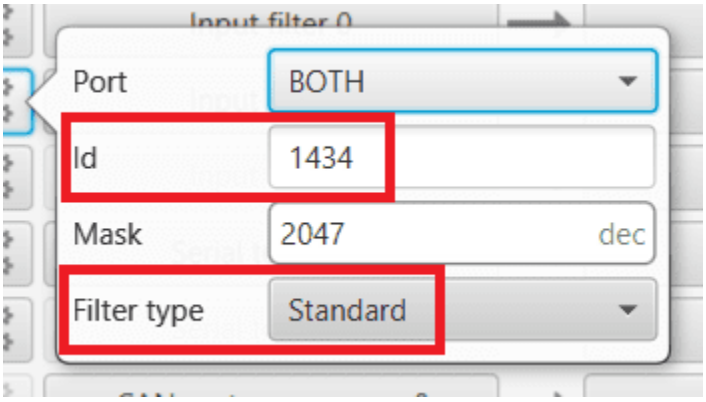


Fig. 32: **CAN Cmd - Input filter configuration**

2.2.2.2 CAN custom message

In the CAN custom message tab, the user chooses the variables to be sent over the CAN buses. The following elements can be configured:

- **TX Ini**: Used to configure transmitted messages that are only sent once at the beginning of the operation (sent when the MC110 boots up). They can be used to initialize some devices.
- **TX**: Used to configure transmitted messages.

Warning:

- The **maximum capacity** of a **CAN message** is **64 bits** (8 bytes), so to send more information it must be divided into several messages.
- MC110 has a **CAN limitation** of **40 TX** messages and **40 TX Ini** messages.

  In addition, there is a limit shared with **Serial Custom Messages**:

  - Maximum number of **vectors** (fieldset): **104**
  - Maximum number of **fields**: **2000**

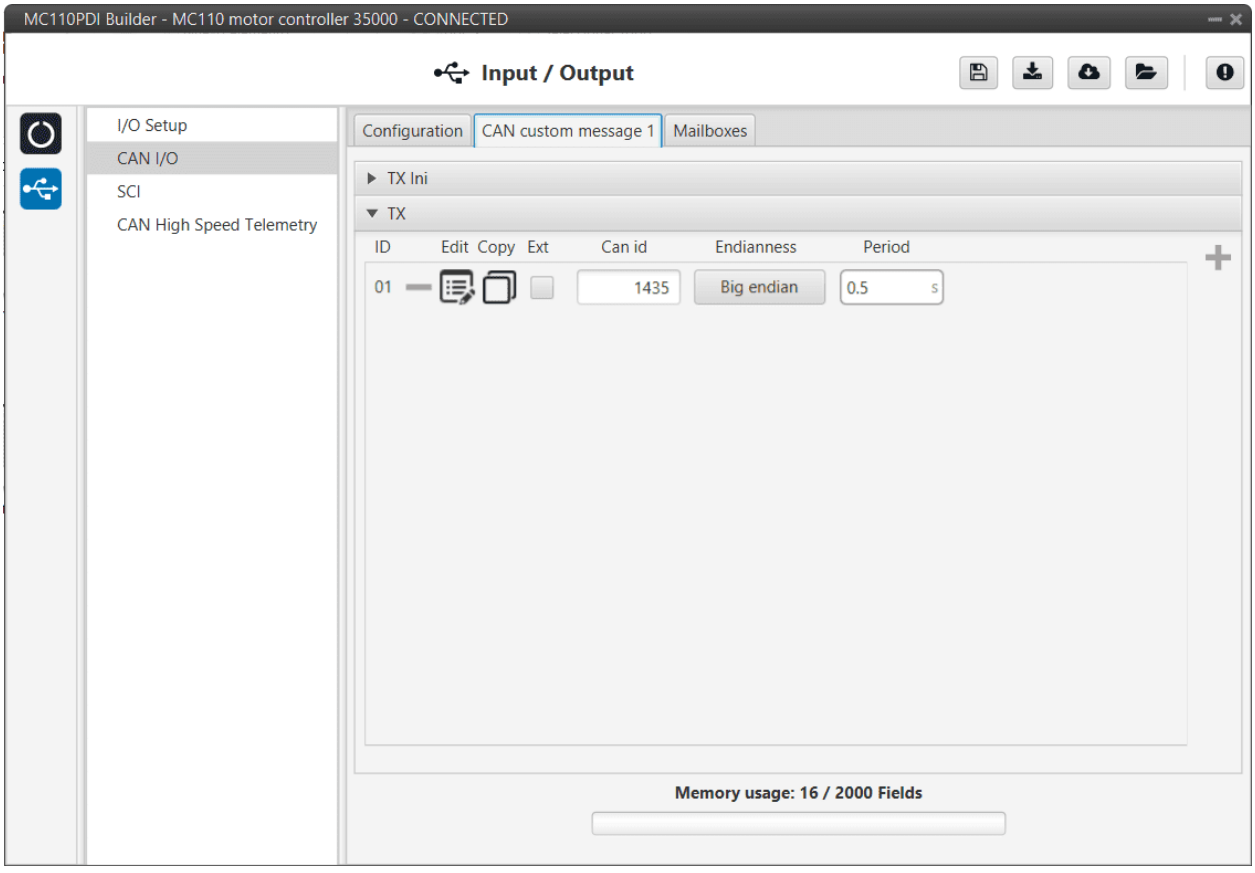


Fig. 33: **CAN custom message panel**

Since this section works in a similar way to the CAN Custom Message configuration in the **1x PDI Builder software**, the explanation to configure the telemetry messages via CAN can be found in the CAN Setup - Input/Output section of

the **1x PDI Builder** user manual.

2.2.2.3 Mailboxes

Main screen to configure baudrate and reception mailboxes of each CAN Bus.

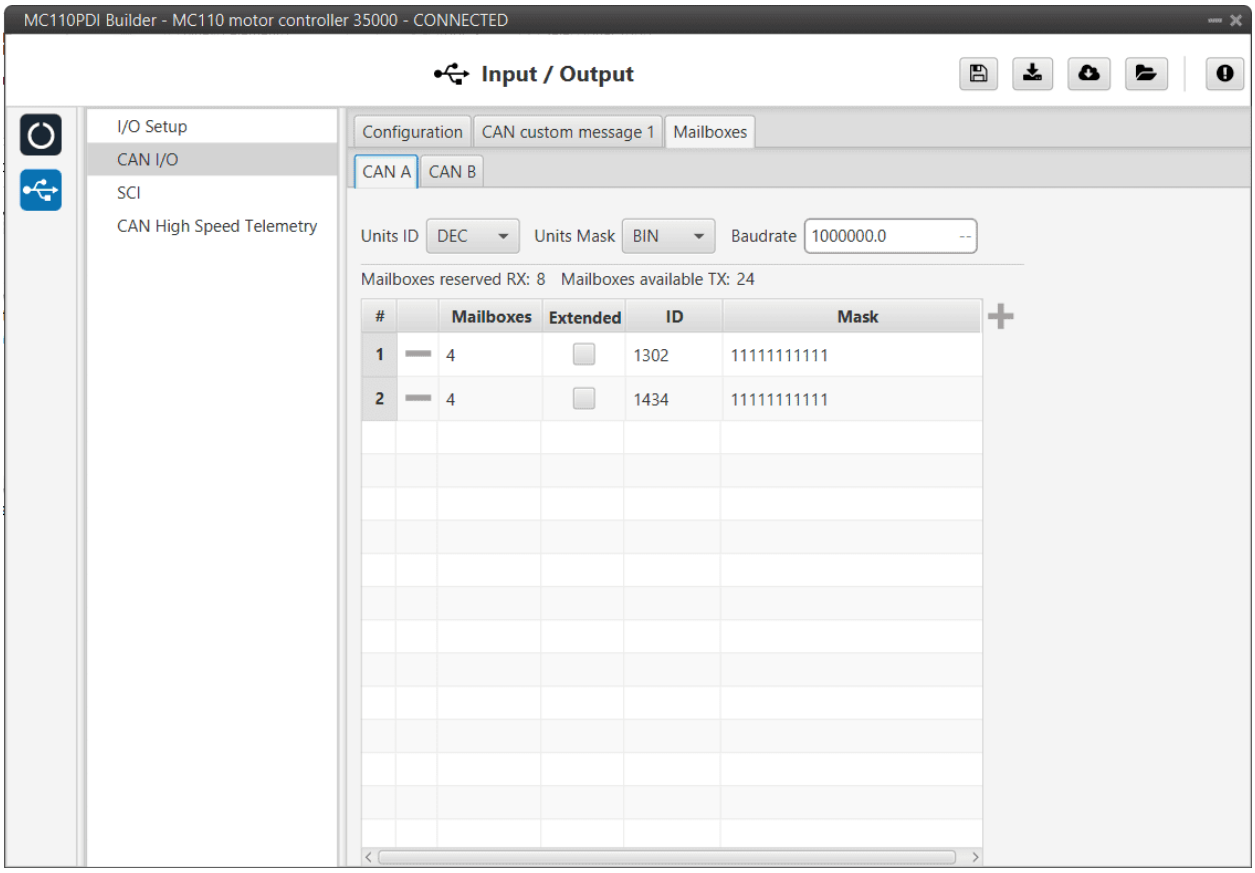


Fig. 34: **Mailboxes panel**

More information about Mailboxes can be found in the Mailboxes - Input/Output section of the **1x PDI Builder** user manual.

Furthermore, in the previous image it can be seen that there is a mailbox already configured with ID **1434**. This is the mailbox that is associated with the **CAN Cmd message**, explained above, and that is **configured by default**.

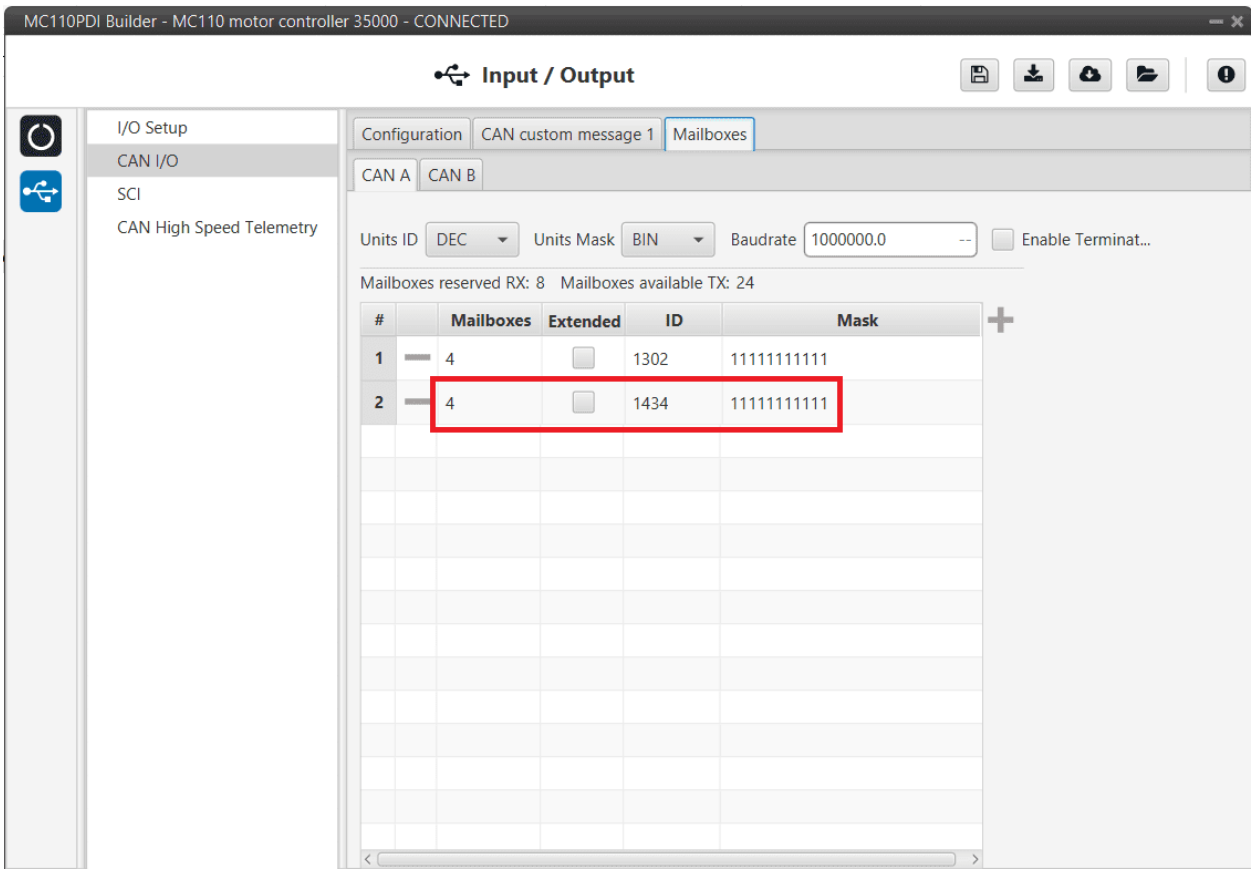


Fig. 35: **Mailboxes - CAN Cmd**

2.2.3 SCI

MC110 can use up to three serial peripherals (SCI A, SCI B and SCI C). Serial ports A, B and C parameters can be edited in this menu to fit the serial protocol requirements.

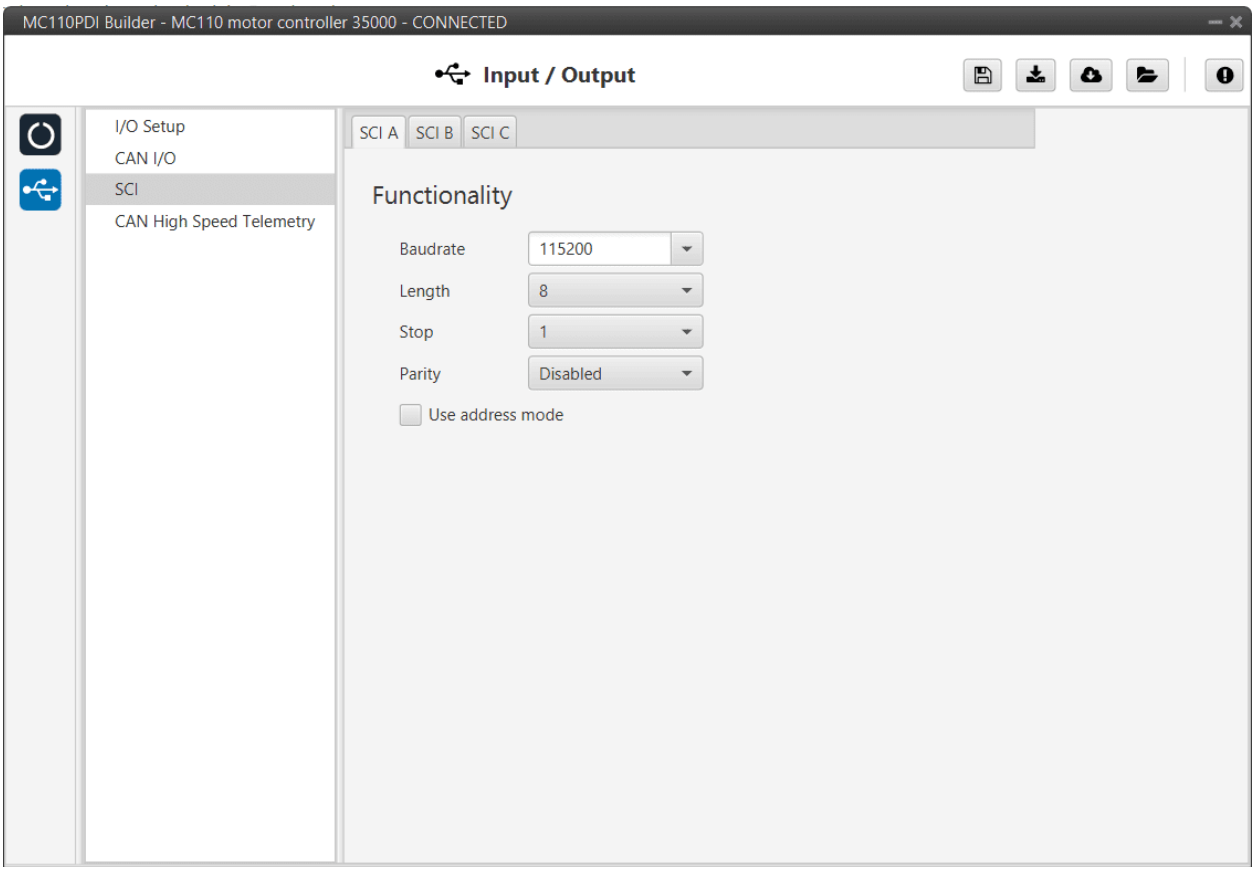


Fig. 36: **SCI panel**

- **Baudrate**: This field specifies how fast data is sent over a serial line.
- **Length**: Defines the number of data bits for each character: 4 to 8 bits.
- **Stop**: Number of stop bits sent at the end of each character: 1, 1.5 or 2.
- **Parity**: Method to detect errors during transmission. When parity is used with a serial port, an extra data bit will be sent with each data character. The bits of each character (including parity bit) will be even or odd according to **parity** mode (**odd**, **even** or **disabled**).
- **Use address mode**: 9-bit data framing uses the bit typically associated with parity error detection to identify address messages. Sent serial data that does not have the address bit set will be ignored (unless the device had previously identified an address message associated with it). This option can be disabled or enabled.

Note: SCI A corresponds to RS485 port, SCI B to RS232 port and SCI C to USB port.

2.2.4 CAN High Speed Telemetry

MC110 is equipped with special CAN telemetry, which is faster than the regular one. This is intended to **monitor all RPM-dependent variables that can be crucial for tuning the user's ESC during the initial stages**.

For instance, seeing the electrical angle can be extremely difficult with a low sampling rate at high RPMs and as a consequence, tuning the observer gain can be difficult. Similarly, monitoring the current being measured in each phase can present the same problem and make PI tuning very time consuming.

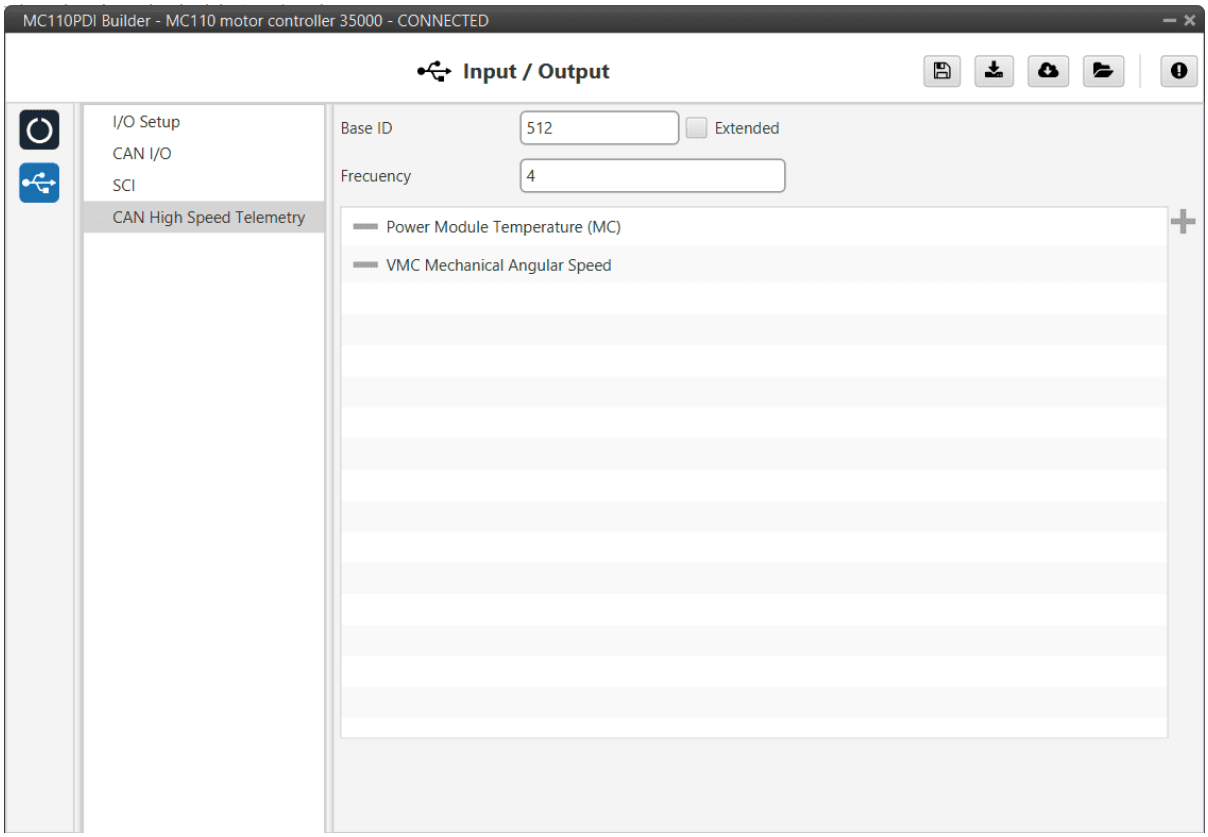


Fig. 37: **CAN High Speed Telemetry panel**

Only two parameters need to be configured here:

- **Base ID**: This is the CAN Id associated with the first variable in the list to be used. Subsequent IDs will be reserved according to the variable list.
  - **Extended**: If enabled, the frame format will be 'Extended', i.e. with a **29-bit identifier**. Otherwise, the frame format 'Standard' (11-bit identifier) is set by default.
- **Frequency** (decimation): The number of clock steps the ESC skips before sending a High Speed CAN telemetry packet.

Note: A separate tool is offered to see MC telemetry ⇒ **CAN Telemetry**, please contact sales@embention.com.