
MEX PDI Builder

Release 6.10.11

Embention

2023-09-29

CONTENTS

1	Quick Start	3
1.1	Download and installation	3
2	Configuration	5
2.1	MEX	10
2.1.1	Mex Base	10
2.1.2	Status	11
2.2	Sensors	11
2.2.1	RPM	11
2.2.2	Magnetometer	13
2.3	Input/Output	14
2.3.1	GPIO	14
2.3.2	PWM	16
2.3.3	I/O Setup	17
2.3.3.1	Tunnel	18
2.3.3.2	Serial custom Messages	19
2.3.3.3	JETI box	22
2.3.4	CAN I/O	26
2.3.4.1	Configuration	27
2.3.4.2	CAN telemetry	29
2.3.5	Digital Input	30
2.3.6	SCI	37
2.3.7	CAN Setup	38
2.3.8	Custom Msg Trigger	39
2.4	Communications	42
2.5	Stick	44
2.5.1	PPM	44
2.5.2	Exponential	46
2.5.3	Trim	47
2.5.4	Output	48
2.6	Devices	50
2.6.1	Jetibox	50
2.6.2	Scorpion tribunus	51
2.7	Arbitration	52
2.7.1	Arbitration	52
2.7.2	Config	53
2.7.3	CAN Setup	54
3	Integration examples	57
3.1	Connection with Autopilot 1x via CAN	57

3.2	CAN Configuration	57
3.2.1	CAN Reception IDs	57
3.2.2	CAN I/O Interconnections	58
3.2.3	CAN Telemetry	60
3.3	Commanding/Reading PWMs	67
3.4	GPIO Command	73
3.5	Jetibox	77
3.6	Reading Arbitration Messages	84
3.7	Reading/Sending RPMs	84
3.8	Scorpion	93
3.9	Serial	95
3.10	Using MEX as external magnetometer Honeywell HMR2300 for Autopilot 1x	101
3.11	CAN Isolator	107
3.11.1	Filtered CAN subnet	107
3.11.2	CAN tunnel	110



QUICK START

MEX PDI Builder is used to set all the configurable parameters of **MEX** (Magnetometer CAN Expander). **MEX** can be adjusted according to different inputs, outputs, CAN communications, sticks, arbitrations and telemetries.

1.1 Download and installation

To configure a **MEX**, first of all download and install **MEX PDI Builder**.

Once the **MEX** 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.

To install **MEX PDI Builder** on Windows just execute the exe file and follow the installer indications. Administrator rights are needed.

Warning: If users have any problems with the installation, please disable the antivirus and the Windows firewall. Disabling the antivirus depends on the antivirus software.

To disable the firewall, go to “Control Panel” → “System and Security” → “Windows Defender Firewall” and then, click on “Turn windows Defender Firewall on or off”.

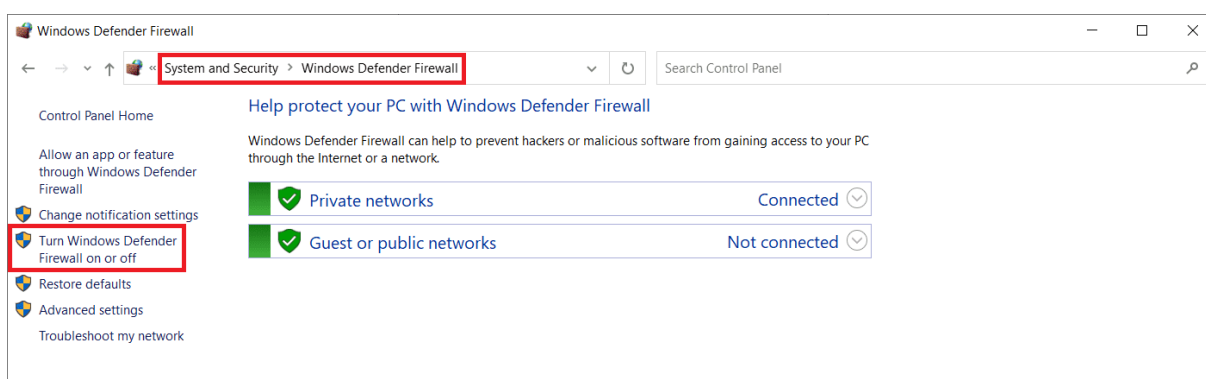


Fig. 1: Windows Defender Firewall

CONFIGURATION

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

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

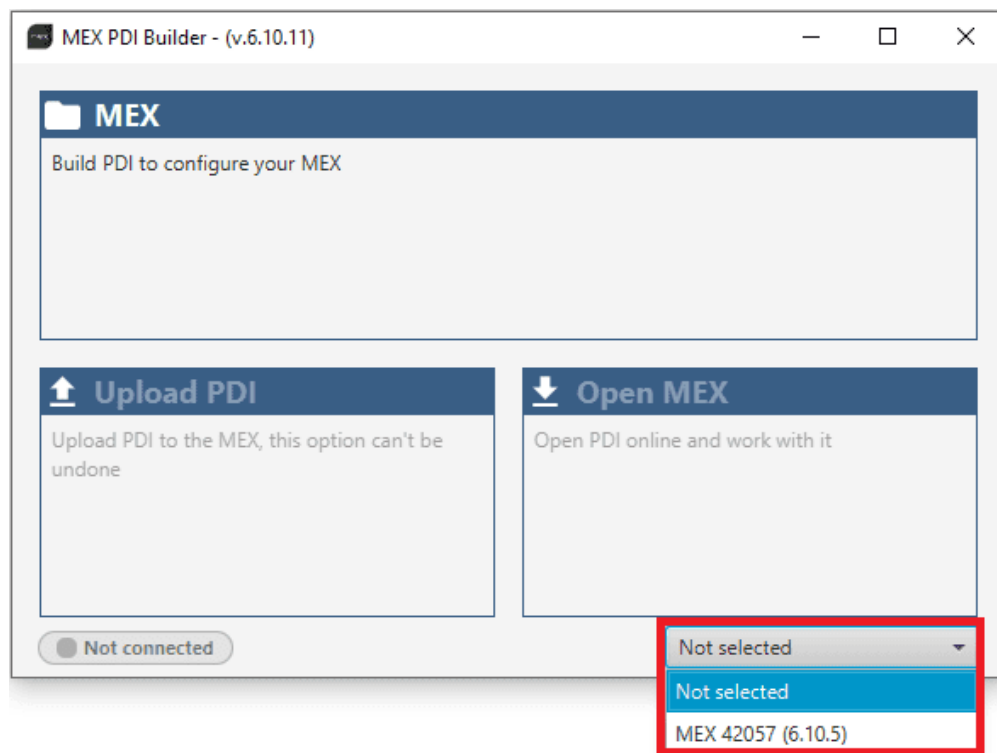


Fig. 1: **MEX ID**

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

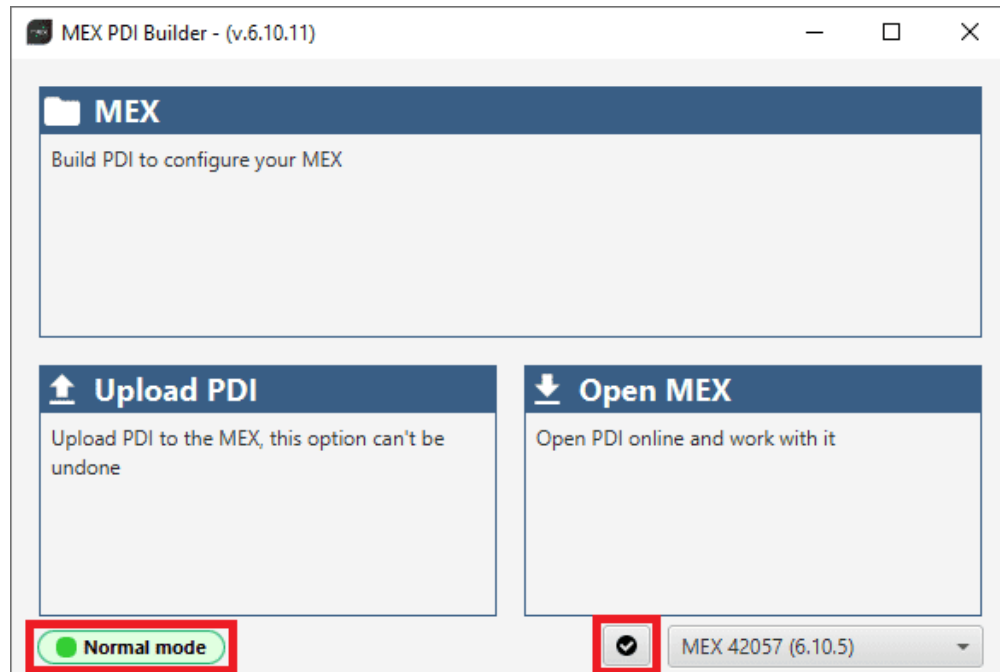



Fig. 2: MEX PDI Builder

- **MEX mode:** The MEX 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 went wrong.

-  **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:

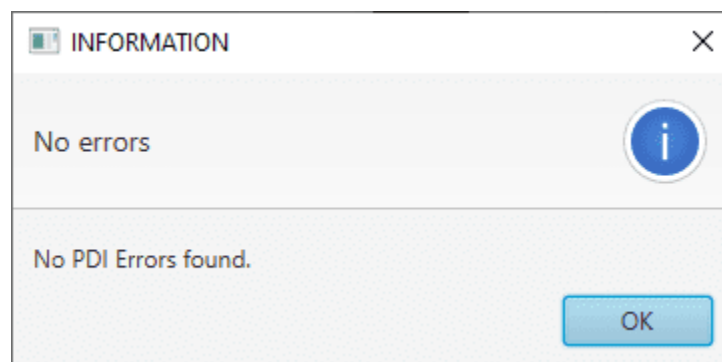


Fig. 3: MEX PDI Builder - PDI Errors message

The user can access now to 3 configuration options:

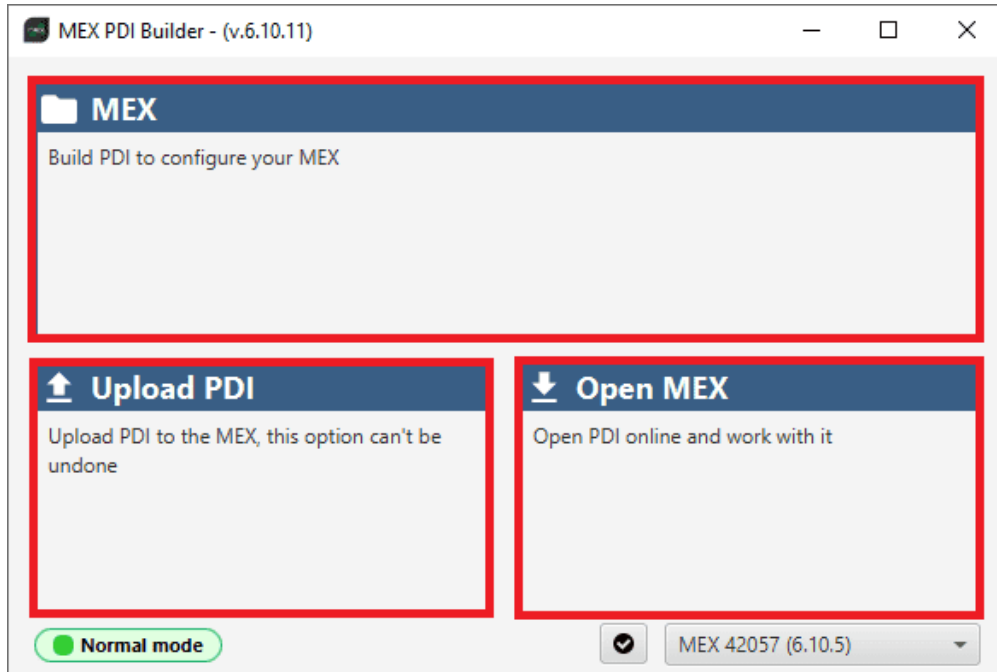


Fig. 4: MEX PDI Builder options

- **MEX**: It allows the user to work with **offline** configurations. A previously exported MEX 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 **MEX PDI configuration** can be imported to the linked **MEX**.
- **Open MEX**: By clicking on this option, **MEX PDI Builder** configuration menu opens with the configuration (the PDI files) loaded in the MEX. Then, the user can modify it online.

Note: PDI files are MEX configuration files. These files are used by modular control with improved version management.

These PDI files are split in 2 folders. Each folder holds several .xml files:

- **mex**: This contains the configuration of the MEX. All control system and parameters are stored here.
- **xsd**: This folder holds .xsd files. An XSD file is a definition file specifying the elements and attributes that can be part of an XML document. This ensures that data is properly interpreted, and errors are caught, resulting in appropriate XML validation. **Users should never delete, replace or modify it.**

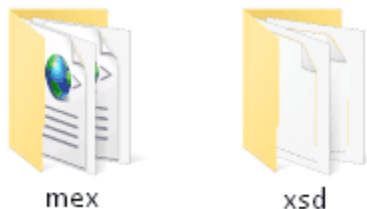


Fig. 5: PDIs files

Finally, click on '**MEX**' or '**Open MEX**' to open the configuration and start editing.

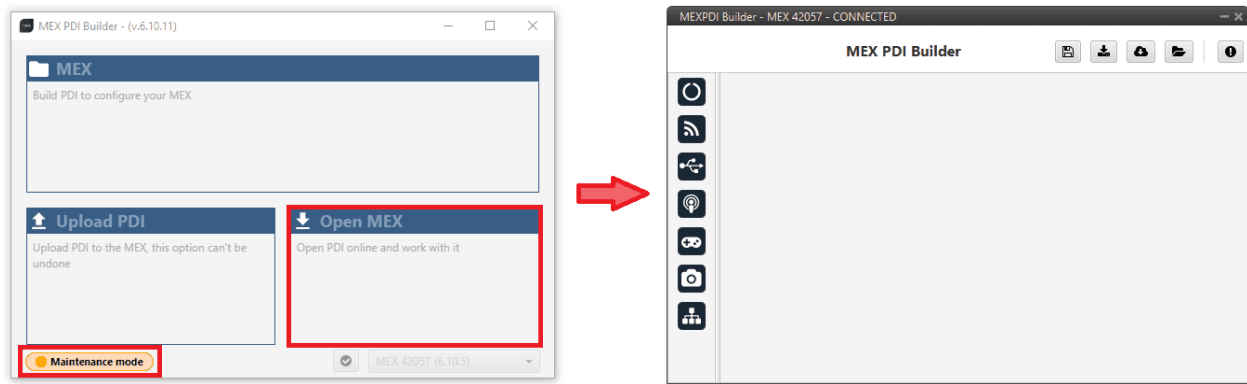


Fig. 6: Open MEX

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

The different ‘buttons’ that can be seen in the initial menu of the **MEX PDI builder** are explained below.

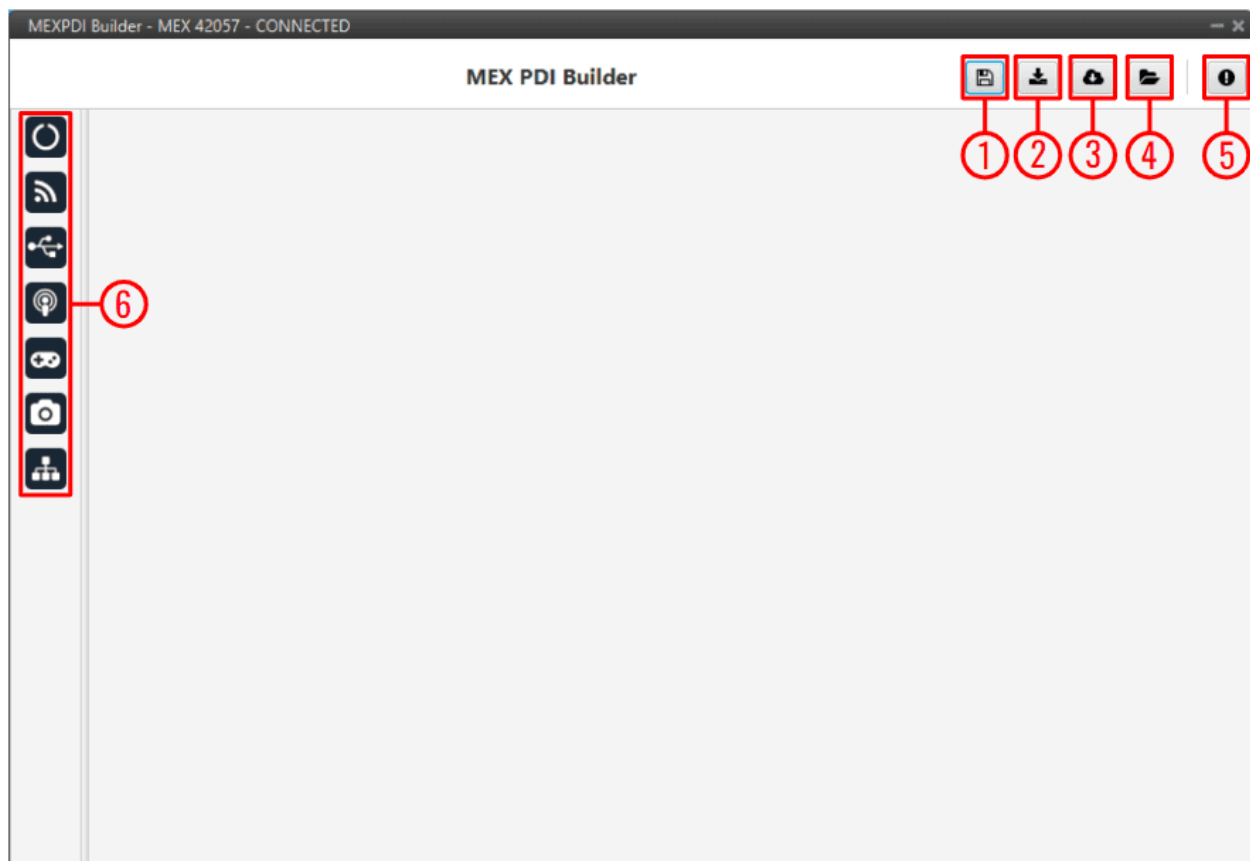


Fig. 7: Initial menu

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

Note: This button will only appear if a MEX 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 MEX.
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 MEX.
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 MEX 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 **Github repository for each customer**.

In case of having any questions about this Joint Collaboration Framework, please see [Joint Collaboration Framework user manual](#) or contact sales@embention.com.

Fig. 8: Feedback window

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

-  *MEX*
-  *Sensors*
-  *Input/Output*

-  *Communications*
-  *Stick*
-  *Devices*
-  *Arbitration*

2.1 MEX

2.1.1 Mex Base

MEX is able to send information about its version and Jeti telemetry from devices connected to MEX, this is the “status message”. It is possible to define the CAN Id used for those messages.

Enable the **Extended** checkbox to use the extended CAN protocol (with a 29-bit identifier), disable it to use the standard protocol (11-bit identifier).

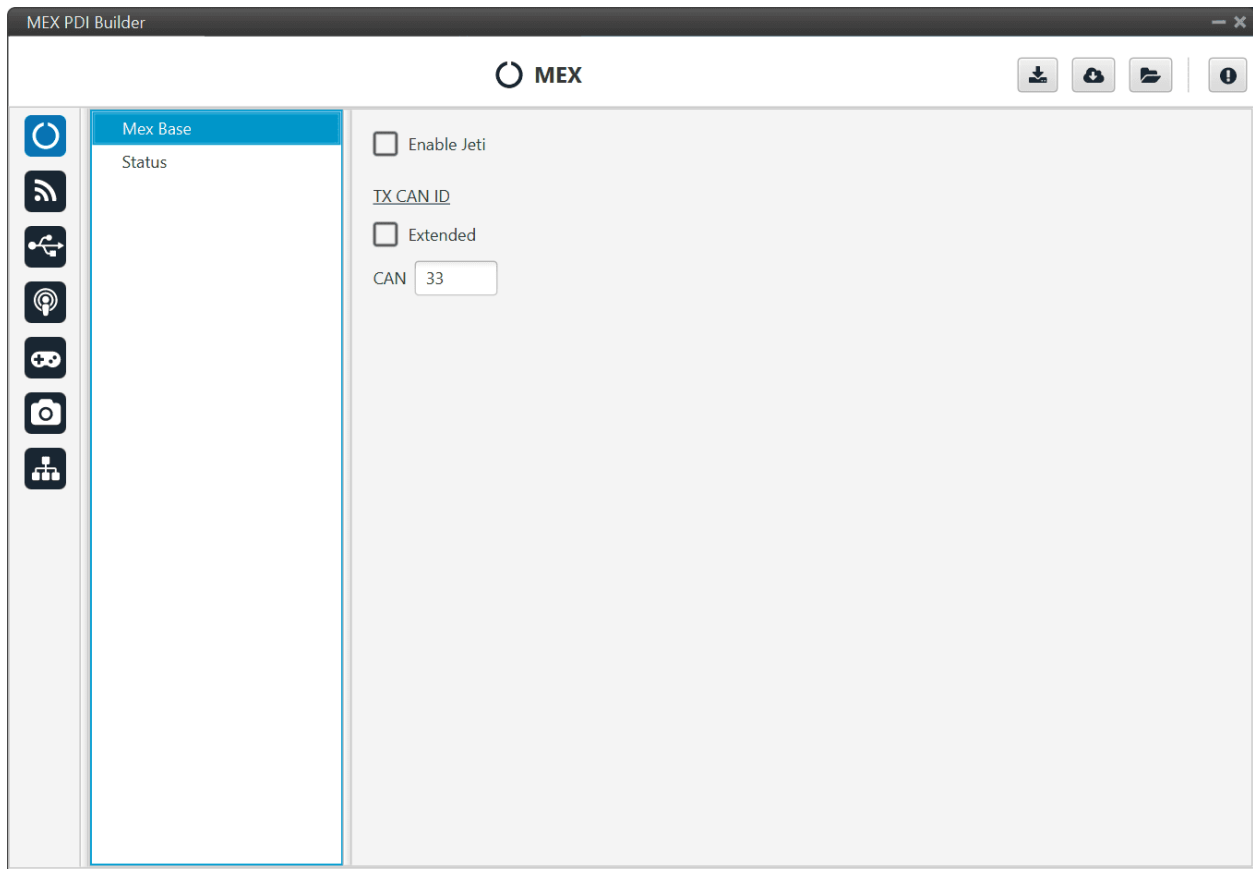


Fig. 9: Mex Base section

2.1.2 Status

Enable VCP Status Message enables the periodic sending of the status message that Veronte Link uses to recognise MEX.

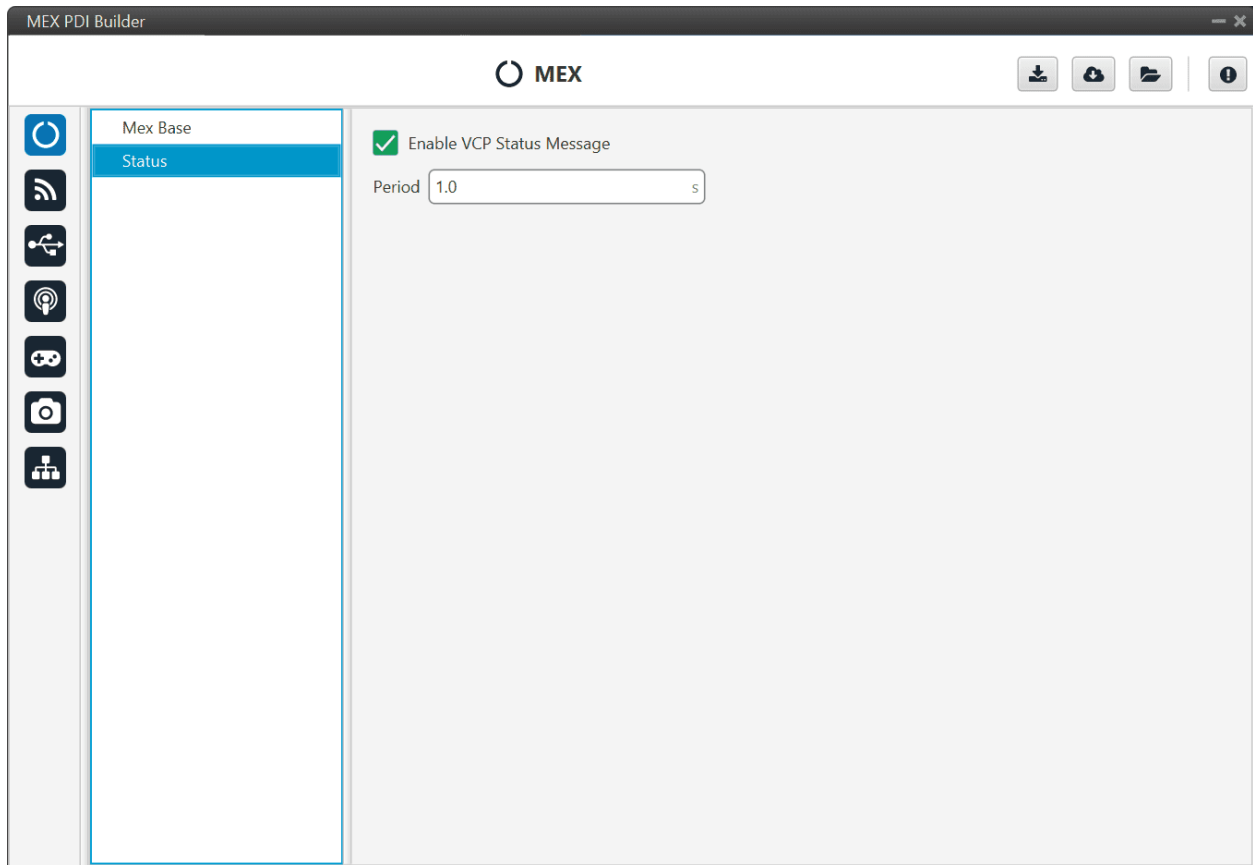


Fig. 10: VCP Status

- **Period:** Enter a desired period to send repeatedly the status message.

2.2 Sensors

2.2.1 RPM

MEX can read RPMs up to four inputs.

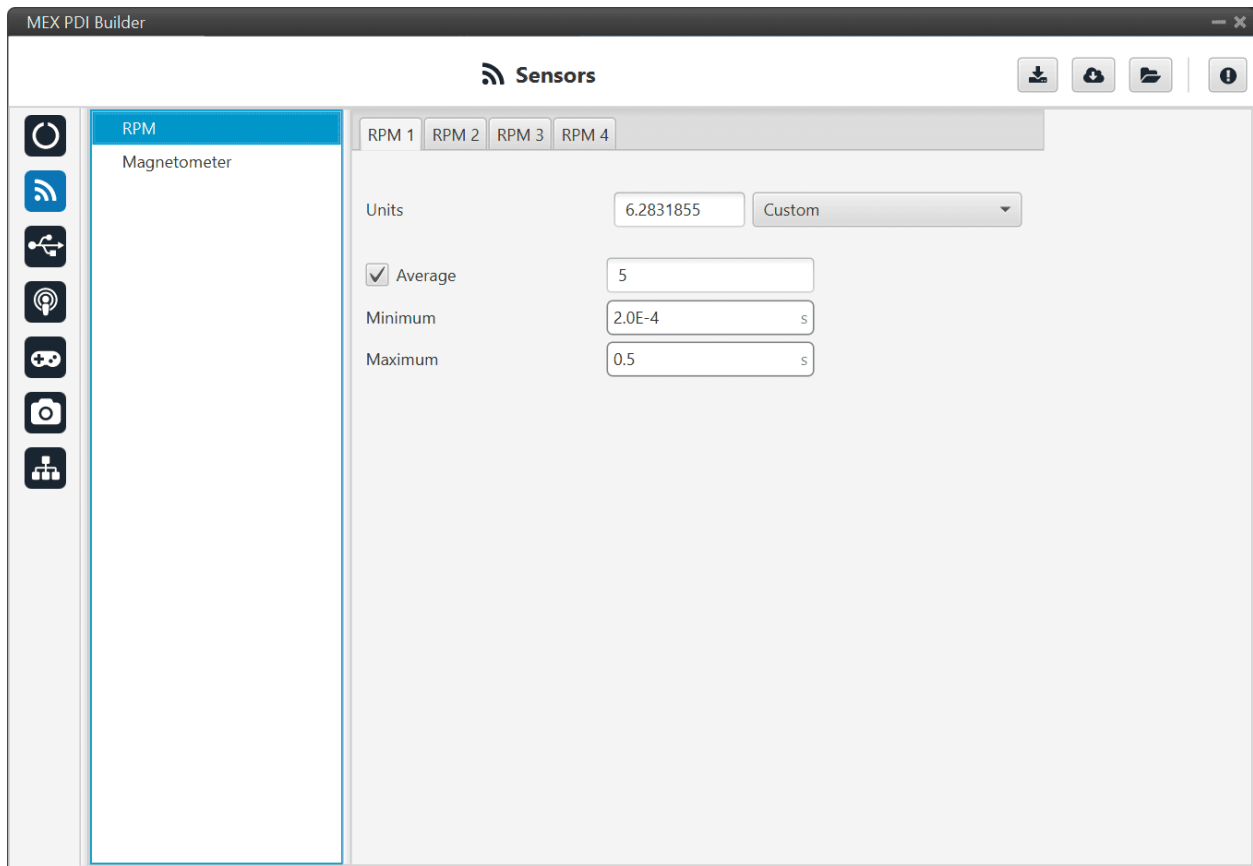


Fig. 11: RPM section

- **Units:** Sensor conversion factor. It can be Custom, Radians per pulse or Pulse per cycle.
- **Average:** Filter to prevent voltage spikes. The readout of the pulse can be filtered as an average output. The amount of measurements to do the average needs to be specified.
- **Minimum:** Here the minimum expected pulse period needs to be specified. This will discard spurious pulses (e.g. induced by EMI) which are smaller than this minimum pulse.
- **Maximum:** The maximum period of time allowed without capturing. If no incoming pulse is received for more than this time, the output RPMs will be 0.

An example about sending RPMs can be found in [Reading/Sending RPMs](#) of the Integration examples section.

2.2.2 Magnetometer

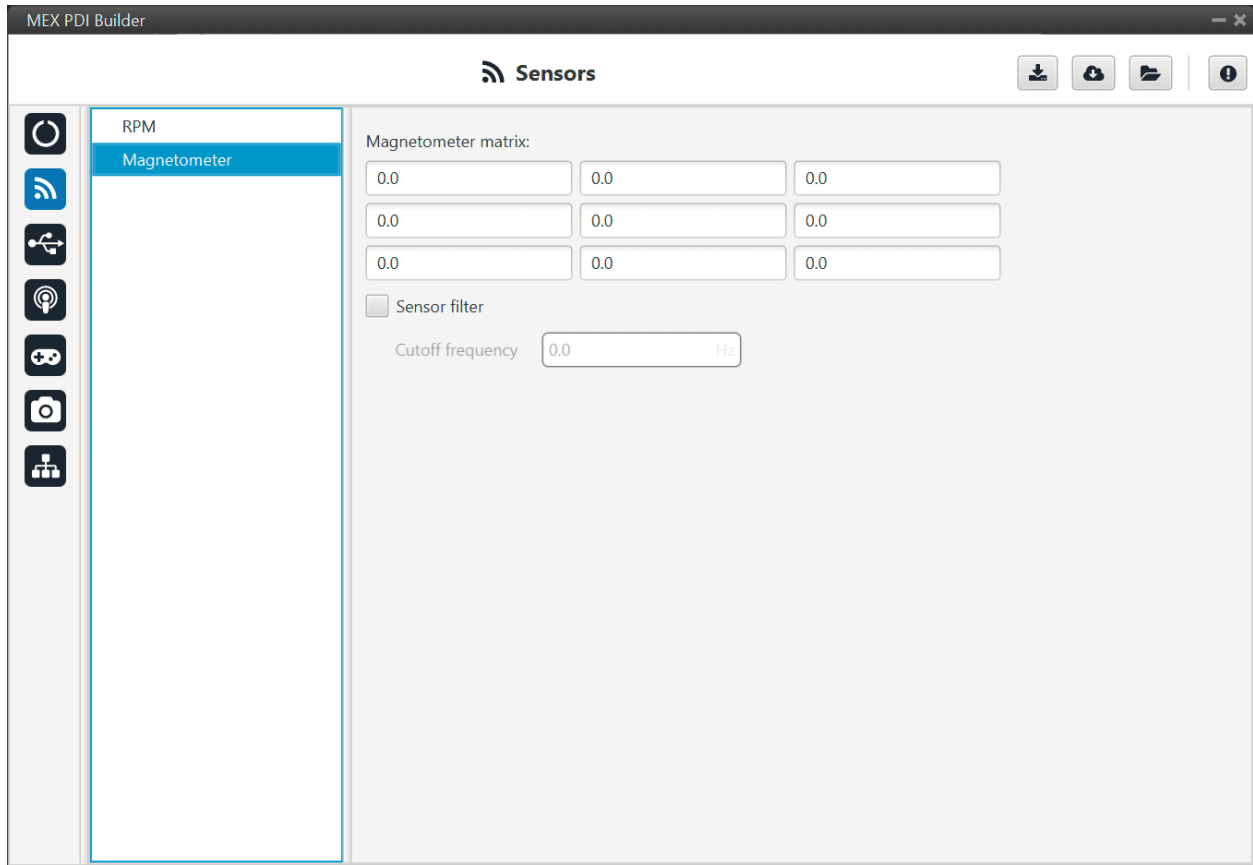


Fig. 12: Magnetometer section

Magnetometer matrix: This rotation matrix represents the rotation of **MEX** coordinates respect to the aircraft body. The axis of **MEX** are physically drawn on the device. The **coordinates axis of the aircraft** are like other Veronte products, which are **defined by the Standard Aeronautical Convention**.

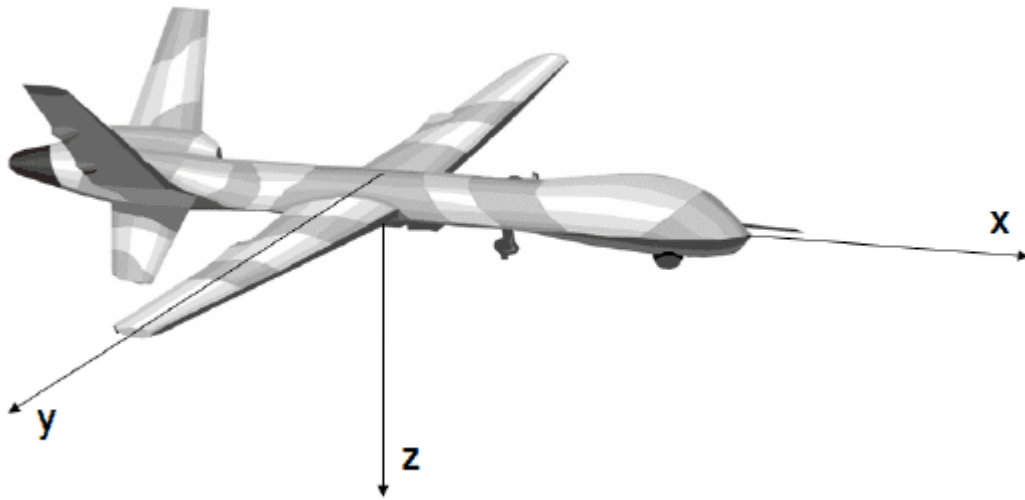


Fig. 13: Aircraft coordinates (Standard Aeronautical Convention)

- **Sensor filter:** The magnetometer measurements may vary with high frequencies and be difficult to read. To solve this problem, **MEX** has a second order low-pass filter to mitigate high frequency variations. The filter can be enabled clicking on the checkbox.
- **Cutoff frequency:** Frequency variations higher than this value will be filtered (if the filter is enabled).

2.3 Input/Output

2.3.1 GPIO

In this window, each individual GPIO (General Purpose Input/Output) behavior can be configured:

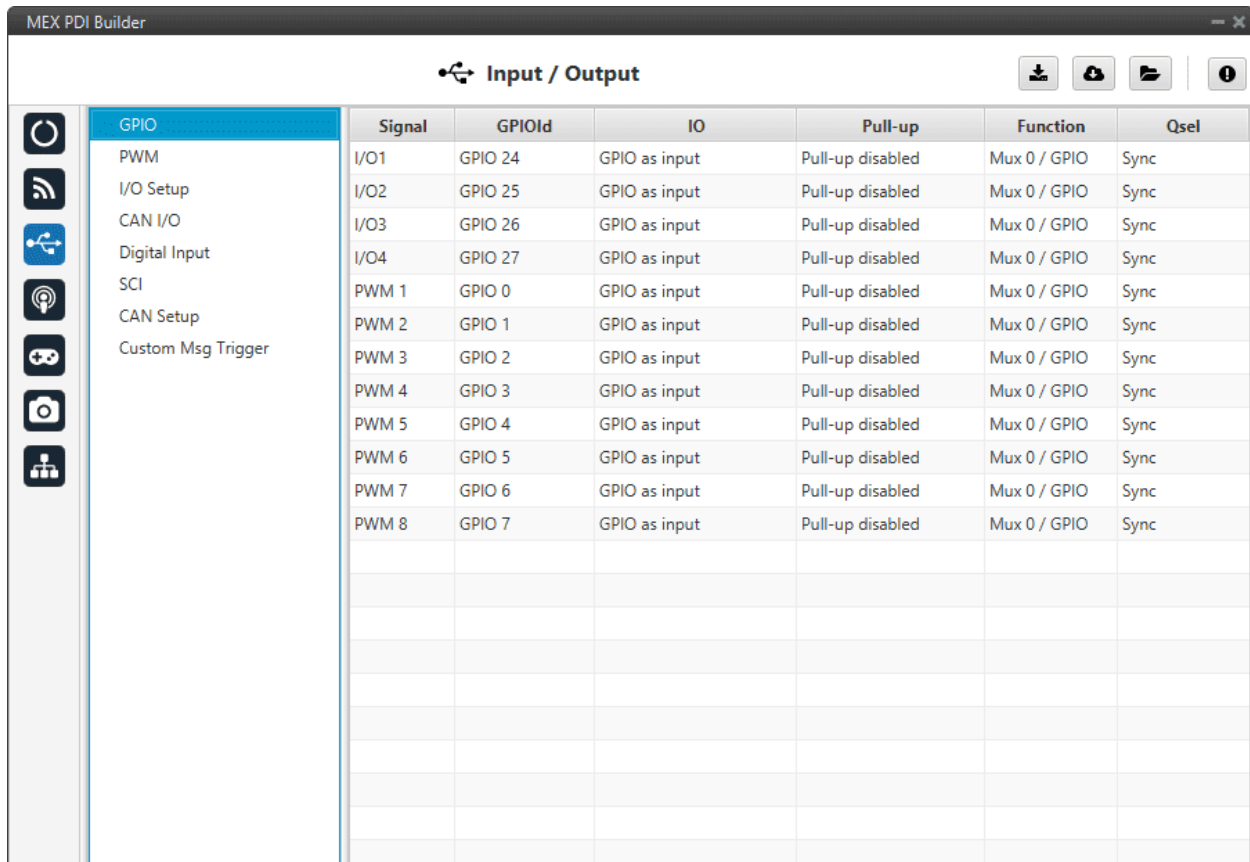


Fig. 14: GPIO section

- **Signal:** Pin ID as described in [Hardware installation -> Pinout](#) section of the **MEX user manual**.
- **GPIOId:** GPIO ID of the microcontroller.
- **IO:** Define GPIO as an input or output.
- **Pull-up:** Enable or disable the pull-up resistance.
- **Function:** Mux 0/GPIO: GPIO, Mux 1: PWM, Mux 2 or Mux 3. These are the different functionalities that the GPIO can have, depending on the multiplexer.
- **Qsel:** This is the “input qualification”, it is used to control how the value of a GPIO is evaluated. The available options are:
 - **Sync:** The value is taken as the time it is checked (synchronously). This is the default mode of all GPIO pins.
 - **3 Samples:** The value is checked 3 times and the value is only changed when the 3 times are the same.
 - **6 Samples:** Same as **3 samples**, but checking 6 times instead of 3.
 - **ASync:** No checks are performed. It is used when it is not used as GPIO.

2.3.2 PWM

In this section each PWM can be configured:

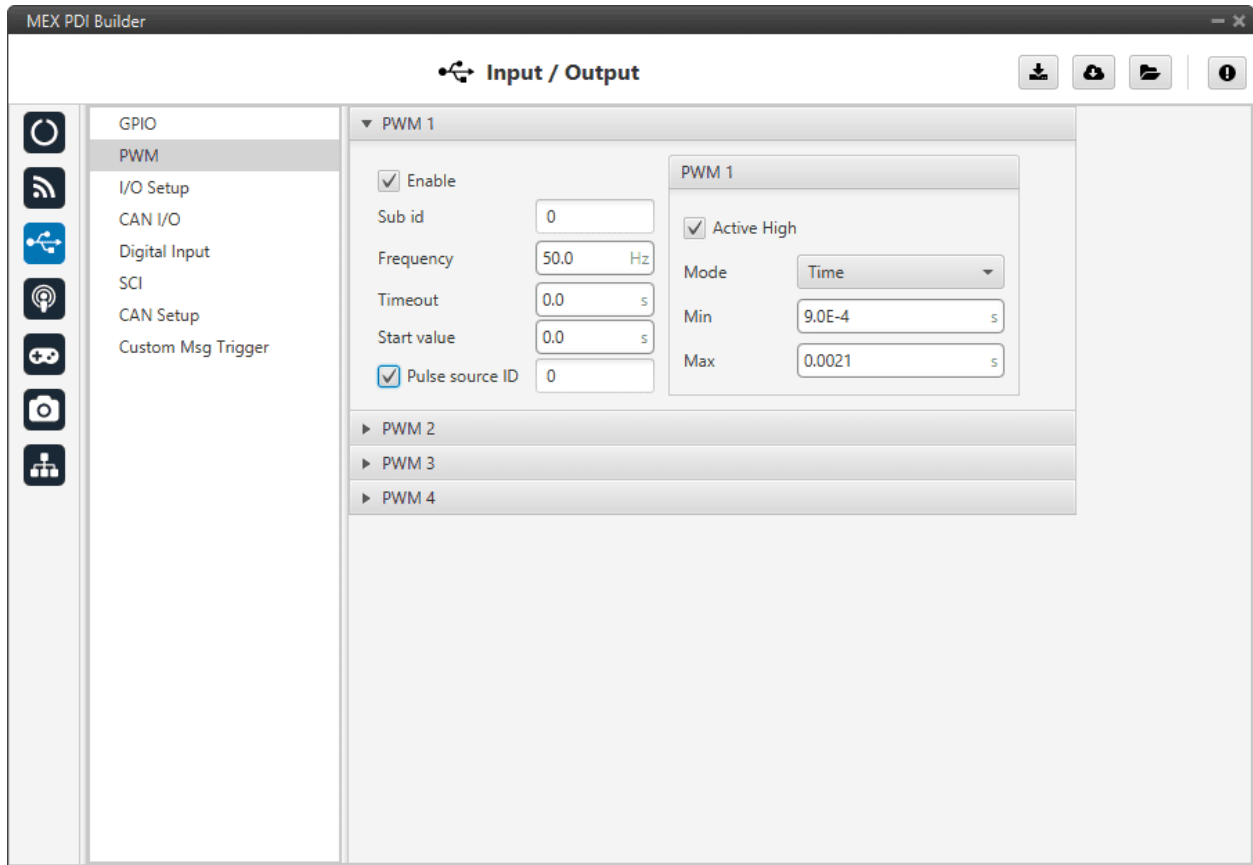


Fig. 15: PWM section

Note: PWMs in MEX work in normalized mode, when the input value is 0 the output value will be the minimum configured, and when the input value is 4095 (12 bits all with ones), the output will be the maximum configured. This approach allows usage of the maximum resolution for the commanded value.

- **Enable:** Define if the PWM is enabled or not.
- **Sub id:** Identifies the sub-id of the PWM and, according to this value, determines the command to be used using one type of CAN message or another.
- **Frequency:** PWM output frequency.
- **Timeout:** If a PWM message is not received in less than this time, the PWM will send as output the start value.
- **Start value:** Value used before any PWM message arrives and on timeout.
- **Pulse source ID:** PWM input ID [0,3], defined in the *Digital Input* section.
- **Active High:** Polarity high or low.
- **Mode:** Time or Duty cycle.
- **Min:** Minimum value. It will be the output when the PWM message specifies 0.

- **Max:** Maximum value. It will be the output when the PWM message specifies 4095.

An example about reading PWM signals can be found in the Integration example section, click [here](#).

2.3.3 I/O Setup

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

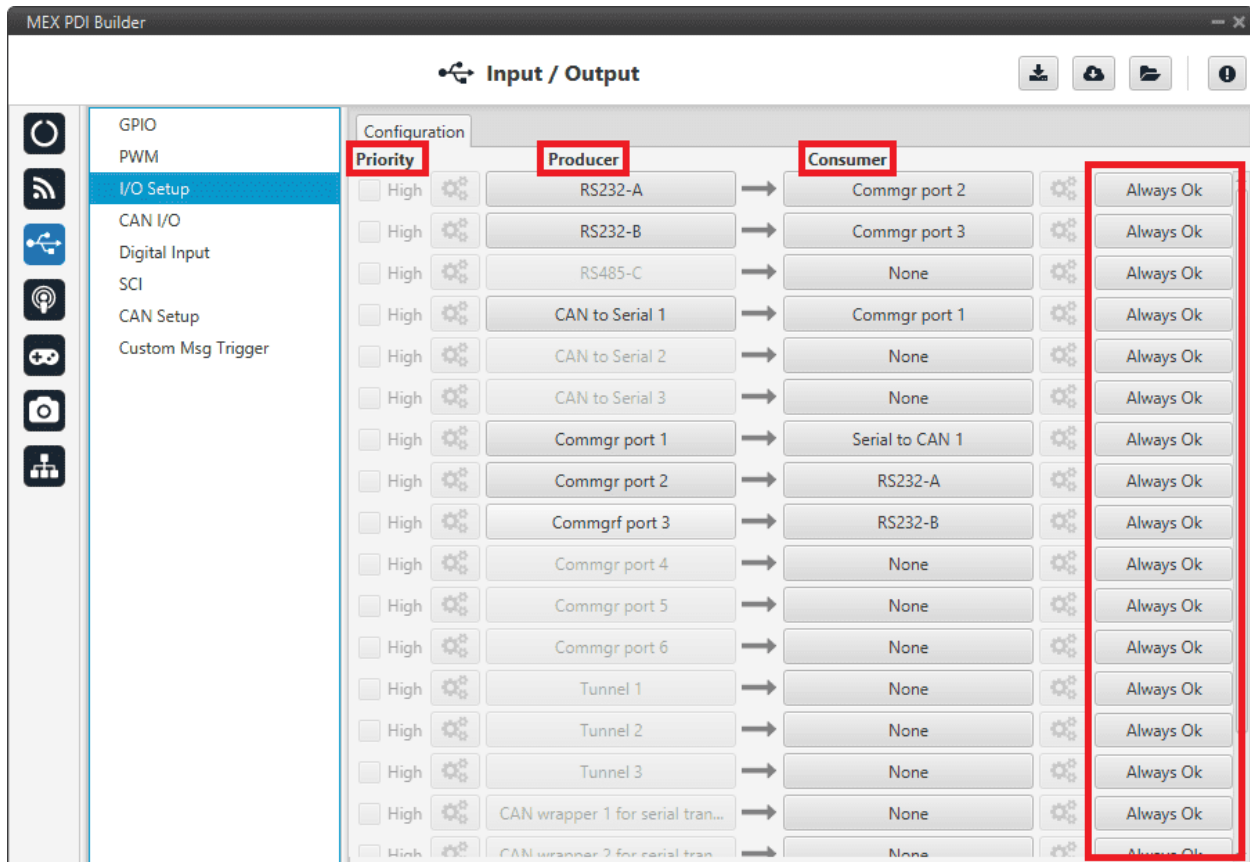


Fig. 16: I/O Setup section

- **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) or bidirectional (Bind Bidirectional), the last enables a port to receive or send information.

The following I/O ports are available:

Field	Description
RS232-A	Serial Port 232 A
RS232-B	Serial Port 232 B
RS485	Serial Port 485
Commgr port	COM Manager Port
Tunnel	Creates a bidirectional bridge between two devices, see Tunnel
Custom message producer/consumer	This allows user to send/receive a serial custom message, see Custom messages
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'
CAN wrapper for serial transmission / Serial CAN unwrapper	CAN wrapper sends CAN streams over a serial Bus / Serial CAN unwrapper undoes this transformation
Tribunus ESC	Reads telemetry data from the Tribunus ESCs by connecting it to one of the serial ports
Lift - MCU	Created for communication with a Lift MCU
JETI box	Simulates a Jetibox to read telemetry from legacy Jeti devices, see JETI box
JETI telemetry	Reads telemetry from Jeti devices

2.3.3.1 Tunnel

A tunnel is a bidirectional bridge between units that communicate each other. The following image shows an example of tunnel configuration:

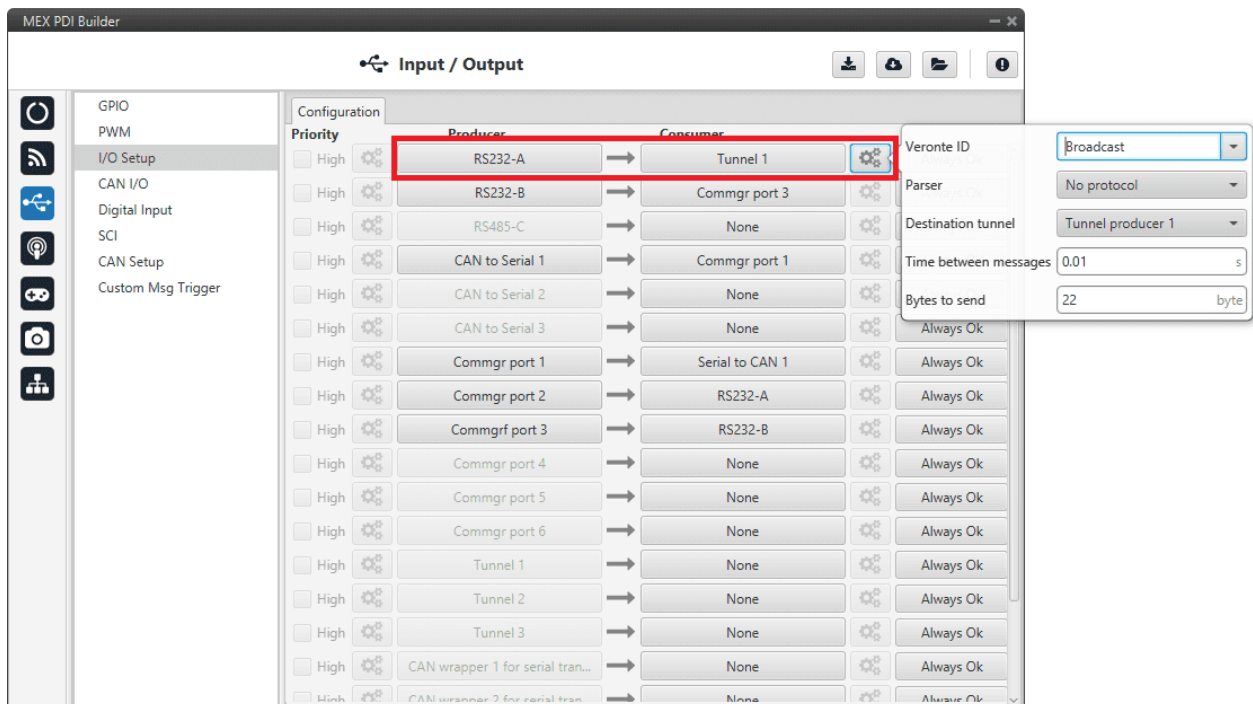


Fig. 17: Tunnel configuration

In the previous image there is a device connected to the **RS232-A (Producer)** and there is a **Tunnel (Consumer)** which sends that information to all units on network (Broadcast). On the other hand, an **Autopilot 1x** unit has to be configured

to receive the signal sent by another device. In this case, the **Producer** will be **Tunnel**, while **Consumer** will be the **port or destination tunnel where the device is connected**.

The options available when configuring **Tunnel** as consumer are:

- **Veronte ID:** Select the address that will receive the information.
 - **App 2:** Veronte Ops address.
 - **Broadcast:** All units on the network. Select this option for a generic configuration.
 - **MEX XXXXXX** Address of a specific unit, it may be a MEX, an autopilot 1x, a 4x, etc.
- **Parser:** The user can choose protocol to parse message data. The options available are:
 - No protocol
 - RTCM3
 - CAN serial
- **Destination tunnel:** Number of port is used to avoid mistakes and identify each Tunnel when using more than one. *Tunnel 1, 2 and 3* are available.
- **Time between messages.**
- **Bytes to send:** Sets the message size to send.

When configuring **Tunnel** as **Producer** (i.e. on the unit that receives the information), no configuration is required. It is only necessary to connect it to a **Consumer**, usually to a serial port.

2.3.3.2 Serial custom Messages

Warning: MEX has the following limitations for serial messages:

- **10** maximum **vectors**.
- **32** maximum **fields** (adding all serial messages).

It is possible to configure messages sent and received through the serial port and its conversion to system variables. To do that, select the option **Custom message producer/consumer** and configure the I/O port.

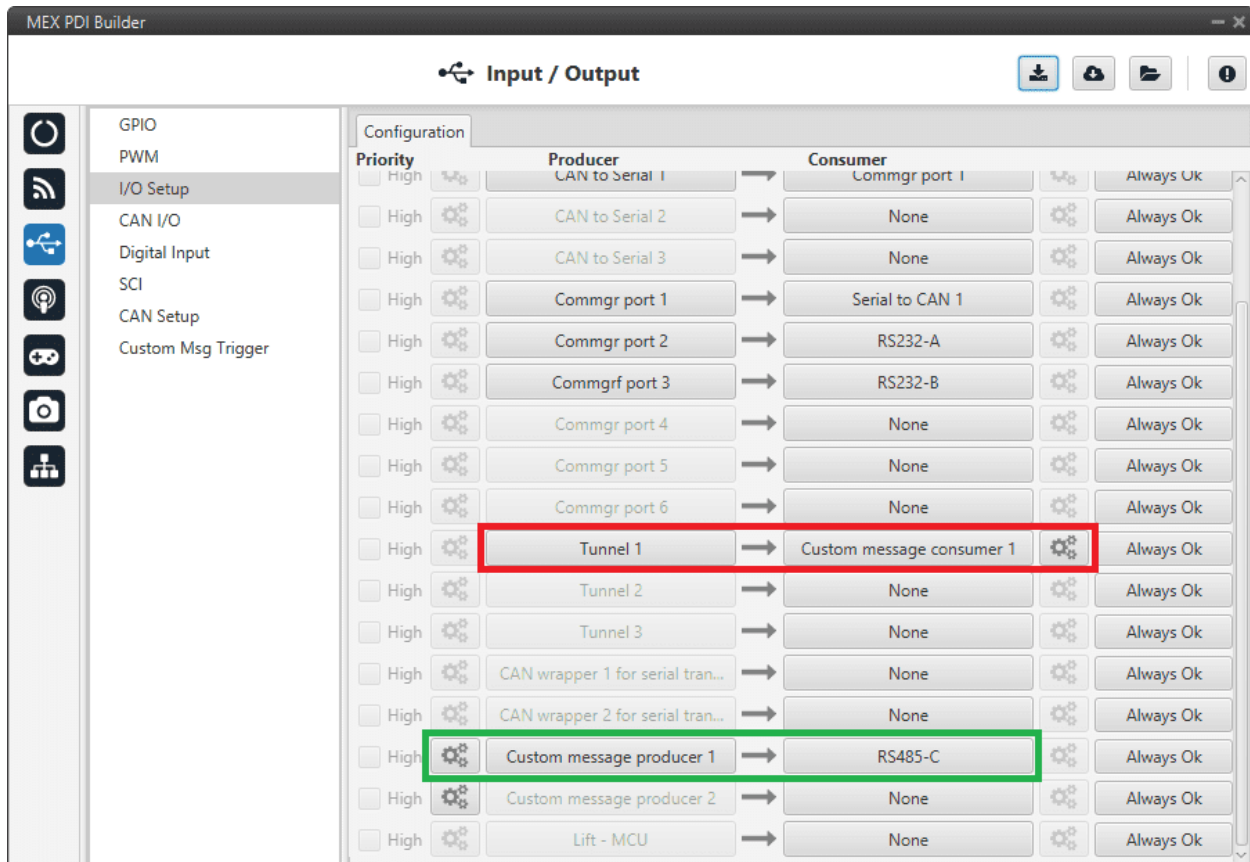


Fig. 18: Serial Custom Messages

The previous image illustrates two possible configurations using a Custom Message.

The red one is configured to receive a message from Tunnel 1 and the green is used to send a Custom Message through a the RS485-C serial port. It is also possible to use the same Custom Message for both tasks if Bind Bidirectional is used (the arrow indicates this).

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

1. Press the **configuration button** ( icon) and another window will be displayed.

In this window press the + icon to add a custom message.

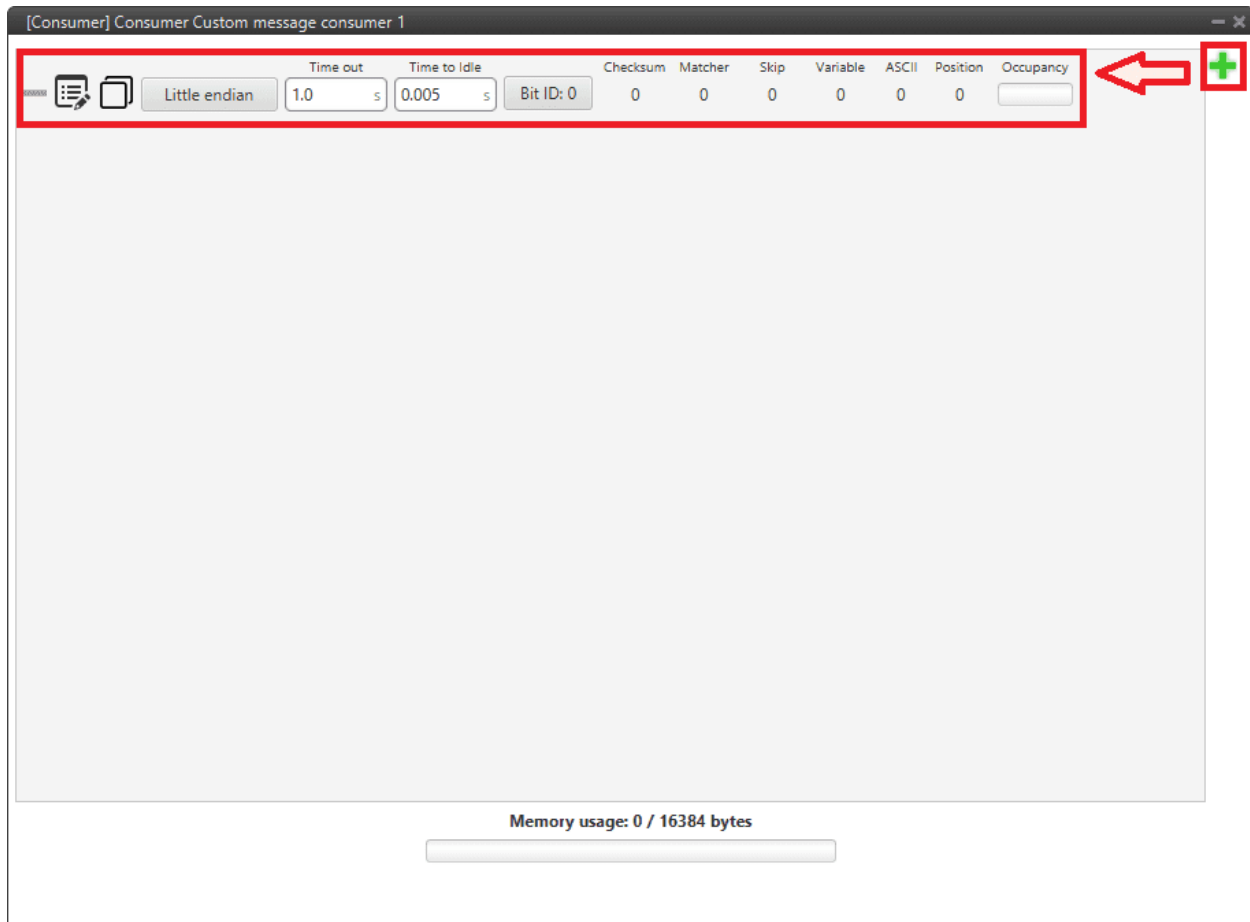


Fig. 19: Serial Custom Message configuration

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

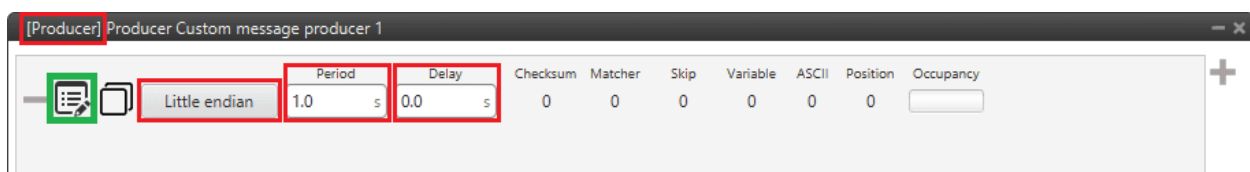


Fig. 20: Custom Message producer configuration




Fig. 21: Custom Message consumer configuration

- **Endianness:** Depending on the order the message is made, it is possible to select:
 - **Big endian:** Set value from left to right.

- **Little endian:** Set 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/Time out:** This option has a dual role depending on whether it is used to transmit or receive data.
 - **Period - Producer:** Time between produced messages.
 - **Time out - Consumer:** Time threshold between receptions to consider that messages are not received correctly.
- **Delay/Time to Idle:** This option has a dual role depending on whether it is used to transmit or receive data.
 - **Delay - Producer:** Delay applied before sending the message. This serves to send messages with the same period without overloading the Serial bus.
 - **Time to Idle - Consumer:** Time that MEX waits before discarding partially parsed bytes.
- **Bit ID:** This option is only available when a message is configured as **Consumer**. The user bit selected in Bit ID box will be true if the message is being received correctly.

Warning: Pay attention that the user bit selected in **Bit ID** is not in use for another task.

3. To create the structure of the message, click on  and then press the “+” icon to add fields to it. The following fields are available to configure a structure: **Variable, Checksum, Matcher, Skip, Parse ASCII** and **Position**.

The configuration of each structure is covered in [Custom Messages section](#) of the **1x PDI Builder manual**.

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

2.3.3.3 JETI box

JETIBOX is a universal communication terminal which can be used with any JETI products.

JETIBOX operates as a two-way terminal, showing all data stored. Employing its four buttons, users browse its menu and set the selected values.



Fig. 22: JETI box device

To simulate it, it is necessary to link the specific JETI box IO consumer to a serial port:

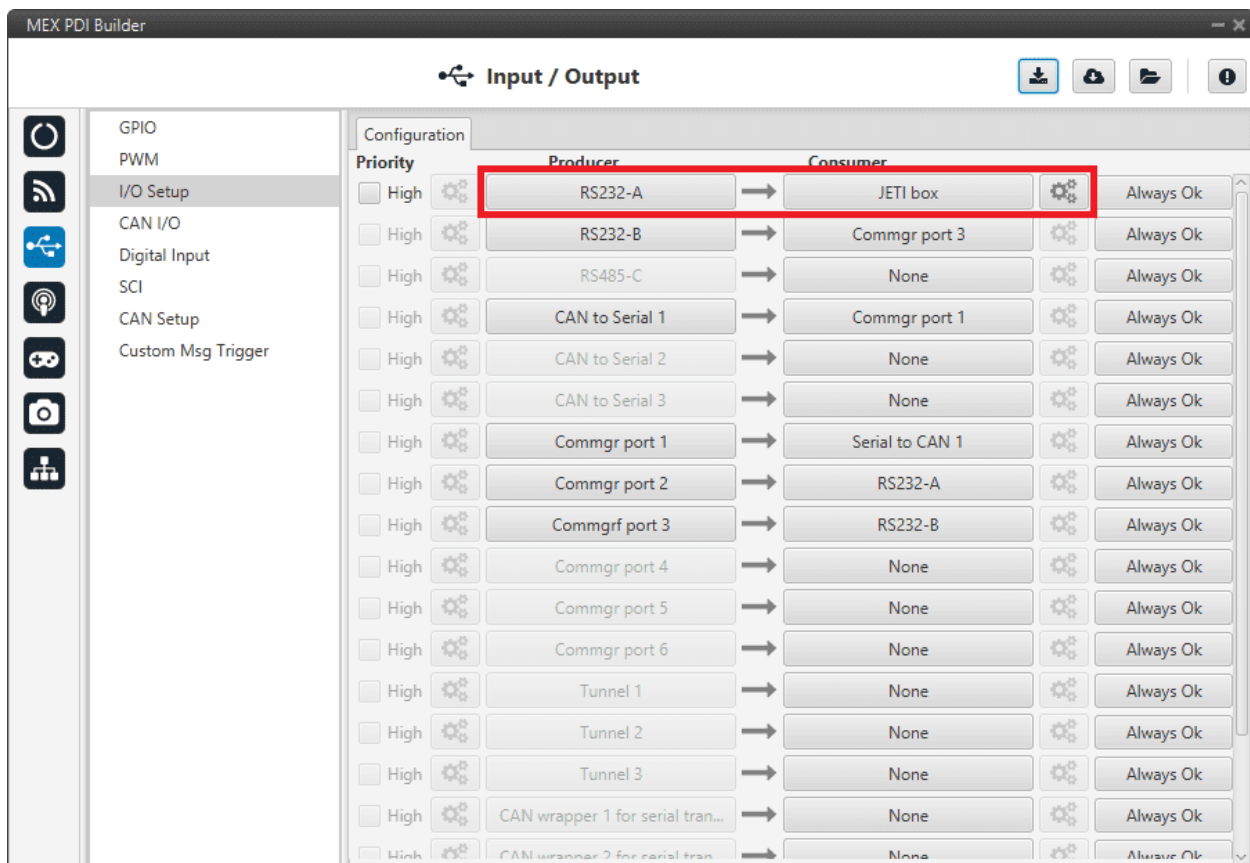



Fig. 23: JETI box Consumer

Then, the sequence to retrieve data shall be configured by clicking on .

To add a Custom Message, click on the + icon:

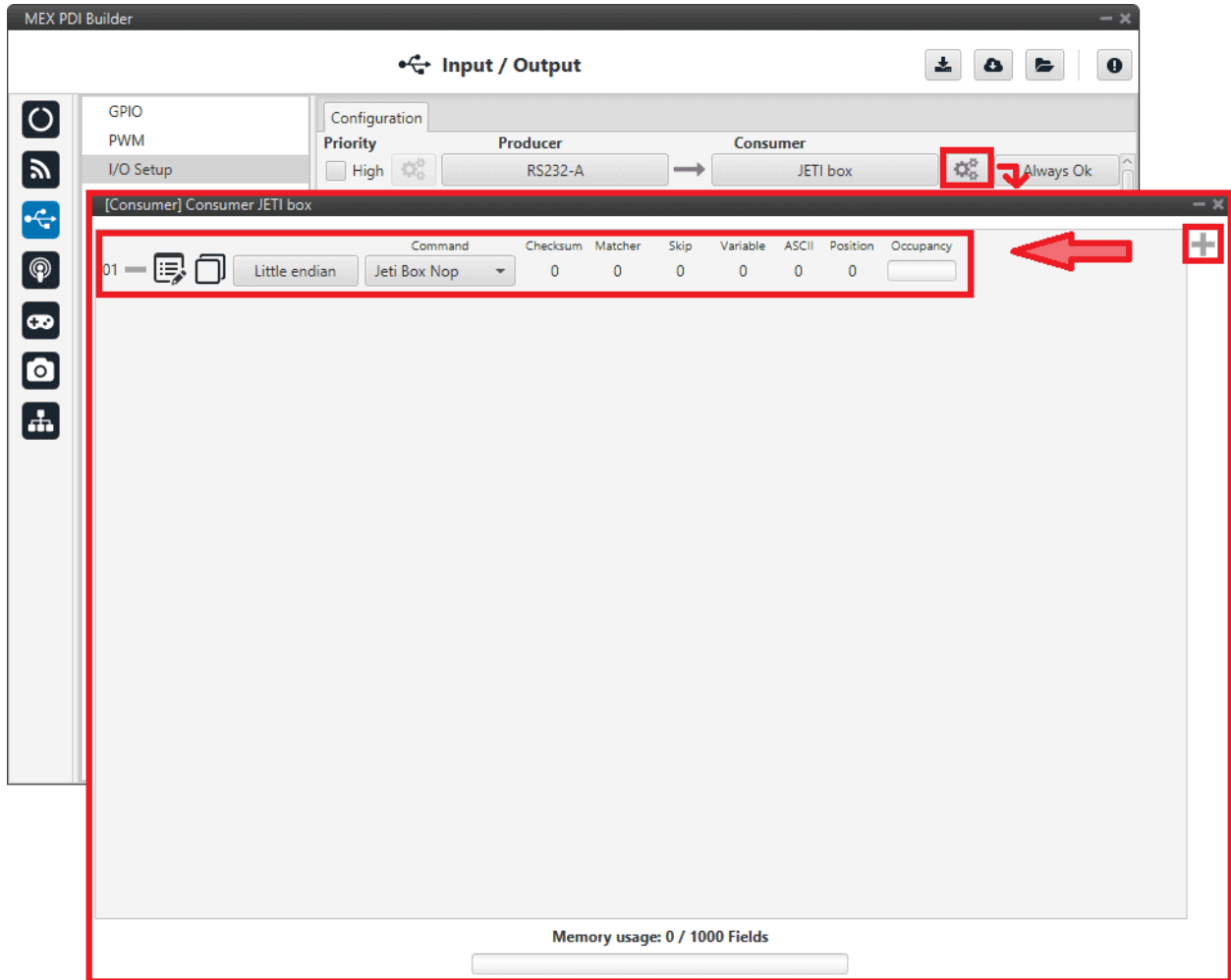


Fig. 24: JETI box Configuration

The following parameters can be configured in the previous pop-up window:

- **Endianness:** Depending on the order in which the device issues the message, it is possible to select:
 - **Big endian:** Sets values from left to right.
 - **Little endian:** Sets values 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).
- **Command:** Here the user can select between **Jeti box Left**, **Jeti box Down**, **Jeti box Up**, **Jeti box Right** or **Jeti box Nop**.

These correspond to the four buttons on the physical JETIBOX (see image above), except the **Jeti box Nop** that is only for simulating a “wait”.

For example, to read the **Actual Voltage** of a **Jeti MasterSpin 220** the Consumer must be configured with a series of custom messages (use **Big endian** for all messages).

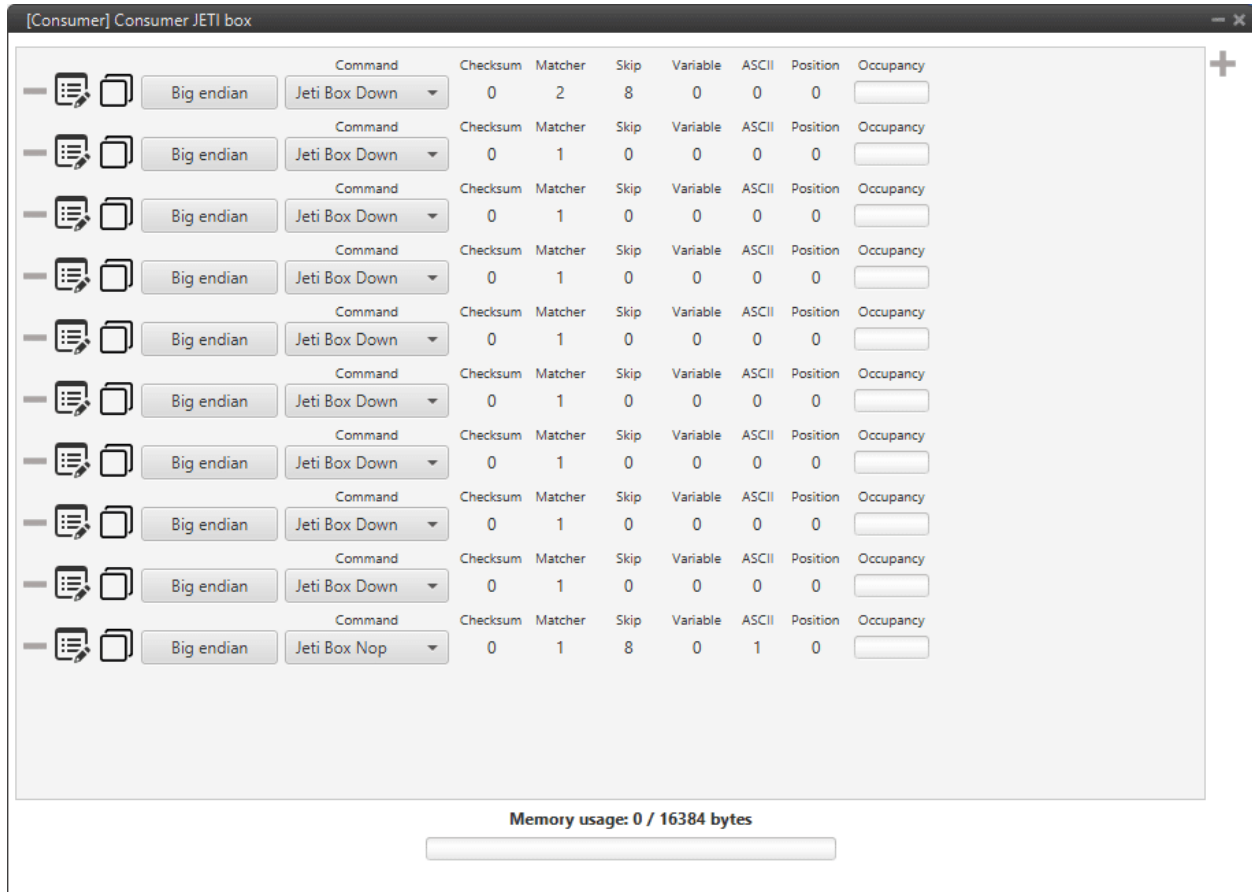


Fig. 25: JETI box example

The following example shows the configuration of one of these messages, the full example can be found in [Jetibox explanation](#) of the **Integration examples** section.

- Expected text: “CONTROLLER TYPE MasterSpin 220~”
- Command: **Jeti box Down**
 - Matcher(32) “CONT” 0x434F4E54 (1129270868)
 - Skip(24*8) 192
 - Matcher(32) “220~” 0x3232307E (842150014)

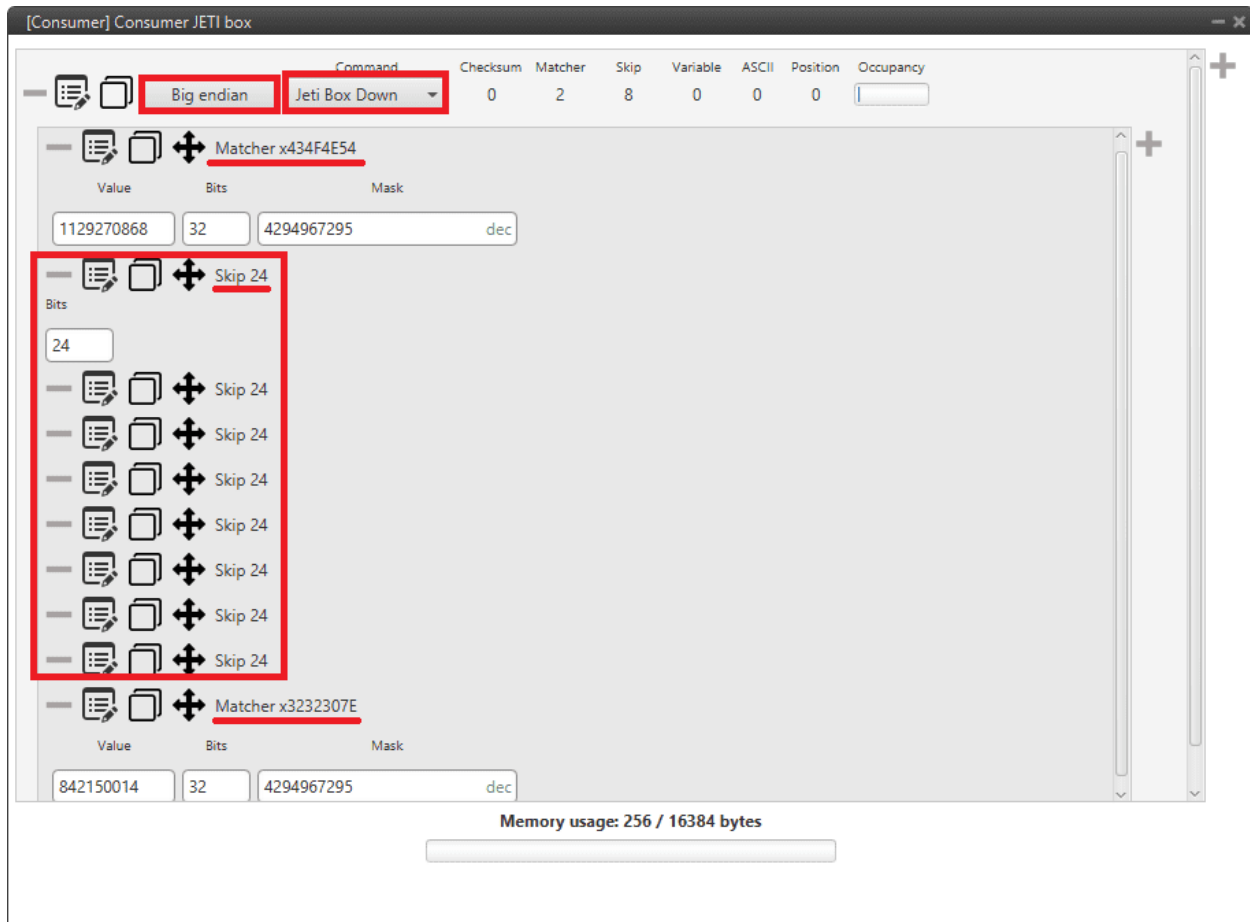


Fig. 26: JETI box custom message example

2.3.4 CAN I/O

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

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

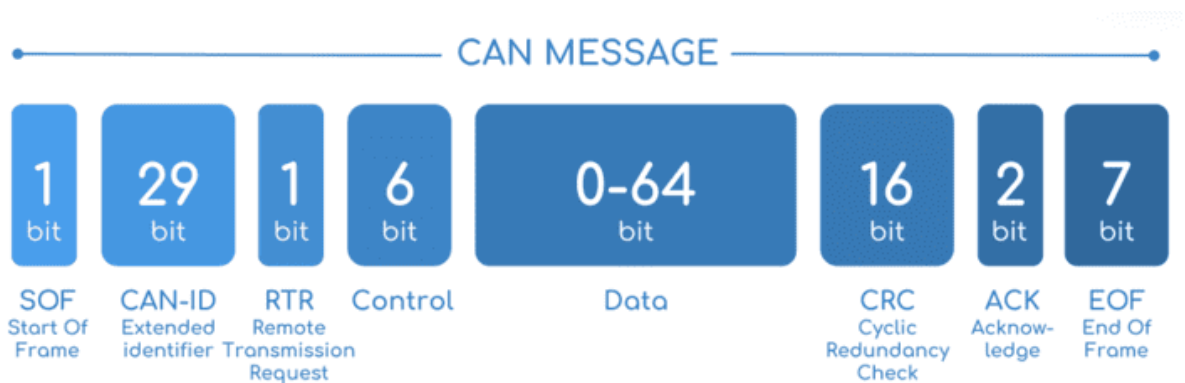


Fig. 27: CAN message structure

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.

For more information on the CAN Bus protocol, see [CAN Bus protocol](#) section of the **MEX Software manual**.

The baud rate of both CAN buses can be configured in the [CAN Setup](#) section.

2.3.4.1 Configuration

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

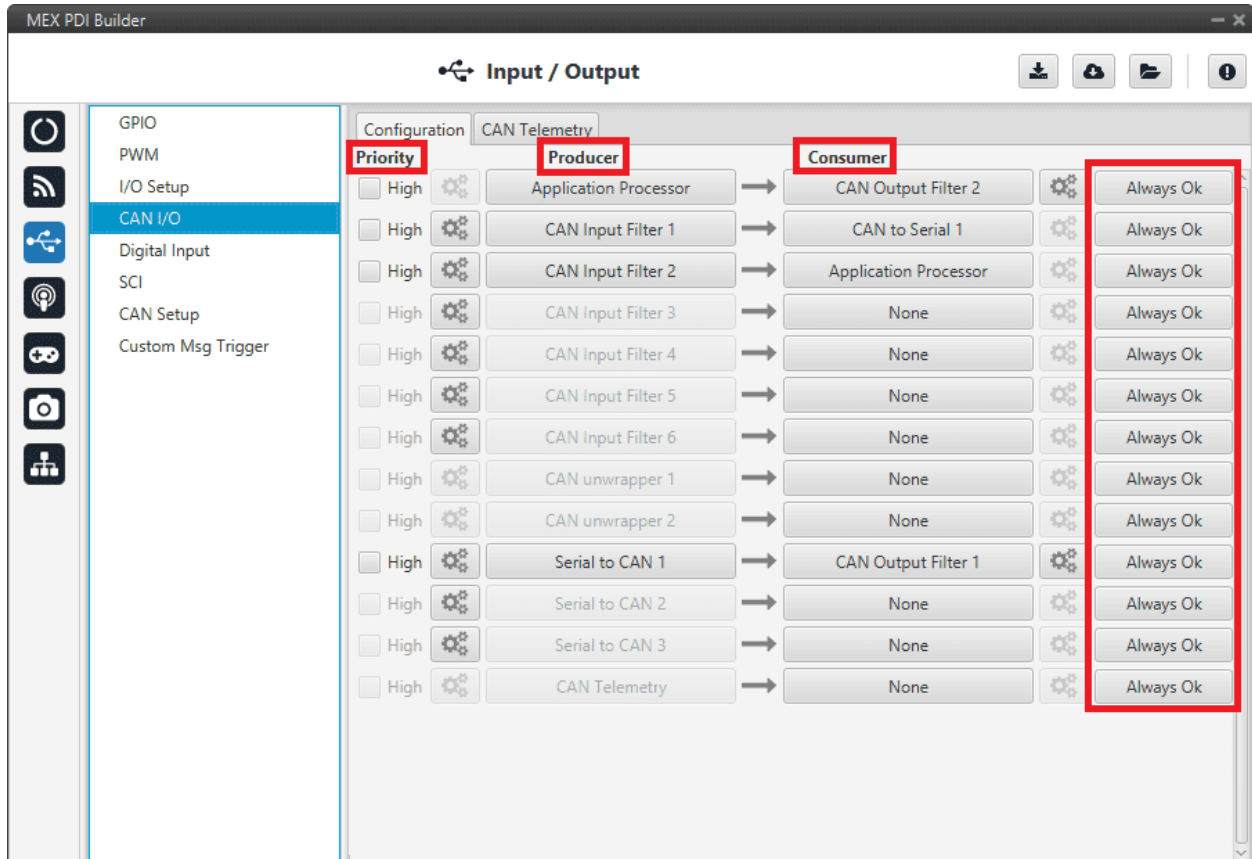



Fig. 28: CAN configuration section

In this menu, the user can find the same 'columns' as in the [I/O Setup](#) menu.

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

- **Application Processor**: Sends a specific set of information, the "status message".
This message is composed as: version (major . minor . revision), address (serial number), system bit error, system power up bit error, PDI bit error, memory allocation bit, file system bit error, CAN A bit error, CAN B bit error, arbiter enabled and arbiter status.
- **CAN 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 :

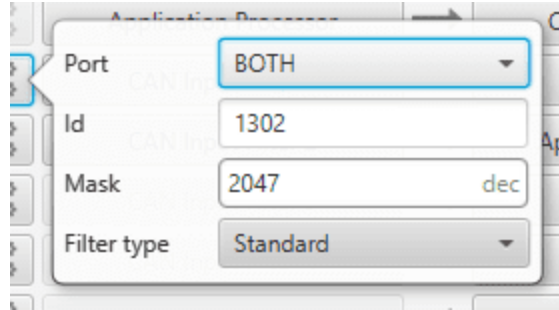



Fig. 29: CAN Input Filter configuration

- **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 1111111100, 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*.
- **CAN unwrapper:** This undoes the ‘CAN serial wrapper’ action, it has to be connected to *I/O Setup consumer* (*Serial CAN unwrapper*).
- **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:

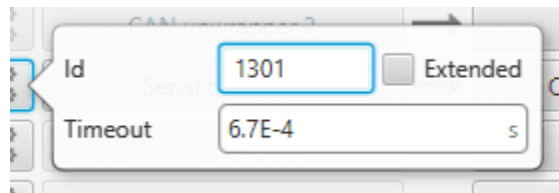



Fig. 30: Serial to CAN configuration

- **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 Telemetry:** Telemetry messages sent via CAN (such as CAN custom messages on Veronte Autopilot 1x to MEX). They are configured in the next section: *CAN Telemetry*.

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

- **Application Processor:** Receives a specific set of information sent by Veronte Autopilot 1x or Arbiter.

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

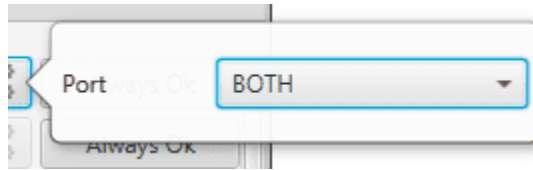


Fig. 31: CAN Output Filter configuration

- **CAN serial wrapper:** CAN messages through serial output, it has to be connected to *I/O Setup* **producer** (*CAN wrapper for serial transmission*).
- **CAN GPIO consumer:** CAN messages from Veronte Autopilot 1x or 4x for GPIO inputs. An example of how to implement it can be found in *Integration Examples*.
- **CAN to Serial:** This undoes the ‘Serial to CAN’ action, it has to be connected to *I/O Setup* **producer**.

2.3.4.2 CAN telemetry

In the CAN Telemetry tab, the user chooses the telemetry to be sent/received through CAN bus. The following element can be configured:

- **TX:** is employed to configure transmitted messages.

Warning: MEX has the following list of limitations:

- The **maximum capacity** of a **CAN message** is **64 bits** (8 bytes), so to send more information it must be divided into several messages.
- CAN limitations:
 - Maximum number of **vectors** (fieldsets): **7**.
 - Maximum number of **fields**: **32** (adding all CAN messages).

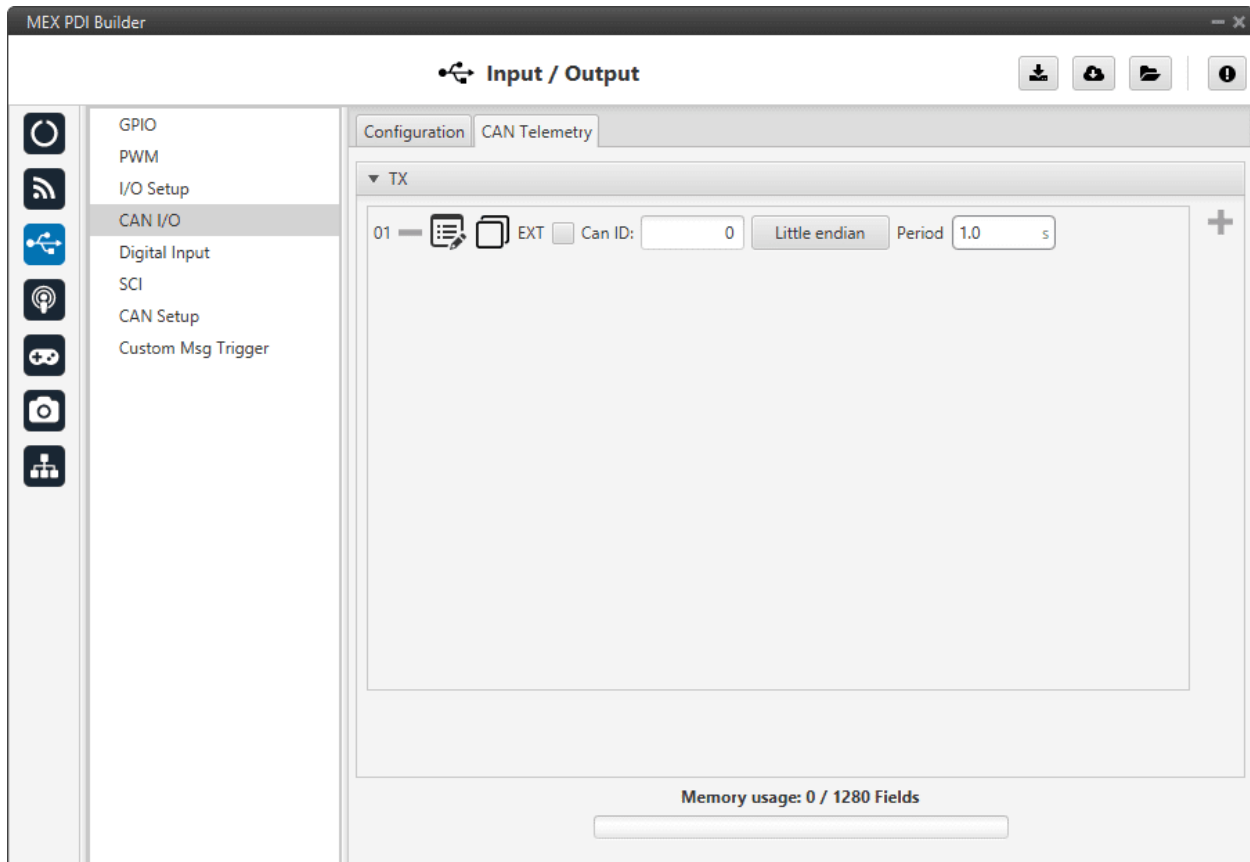


Fig. 32: CAN Telemetry section

Since this section works similar to the **1x PDI Builder** software, the explanation to configure the telemetry messages via CAN is reflected in the [1x PDI Builder manual](#), like the configuration of the CAN custom messages.

An example of the sending of **CAN telemetry messages** can be found in the *Integration example* section.

2.3.5 Digital Input

Digital inputs can be used to measure pulse count, pulse widths and PPM signals from a RC radio. Each source shall be connected to the desired consumer to allow measurements.

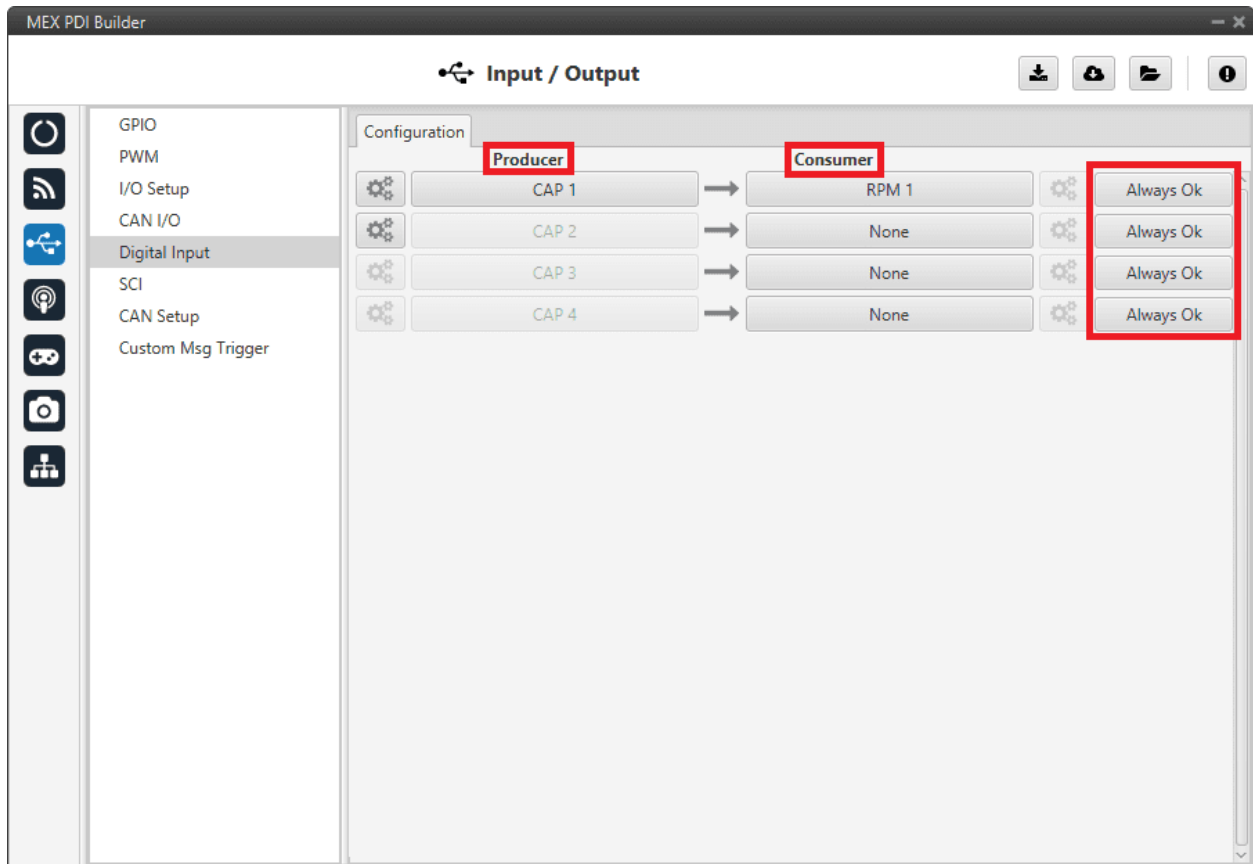


Fig. 33: Digital Input section

In addition, in this menu the user can also find the ‘**Bit** column’ in the same way as in the **I/O Setup** and **CAN I/O** sections:

- **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.

The process to configure a device can be done as follows:

1. Select and configure a **Producer** to read the external signal. There are 4 possible producers: CAP 1 - 4.

Press on the configuration button  and a pop-up window will show.

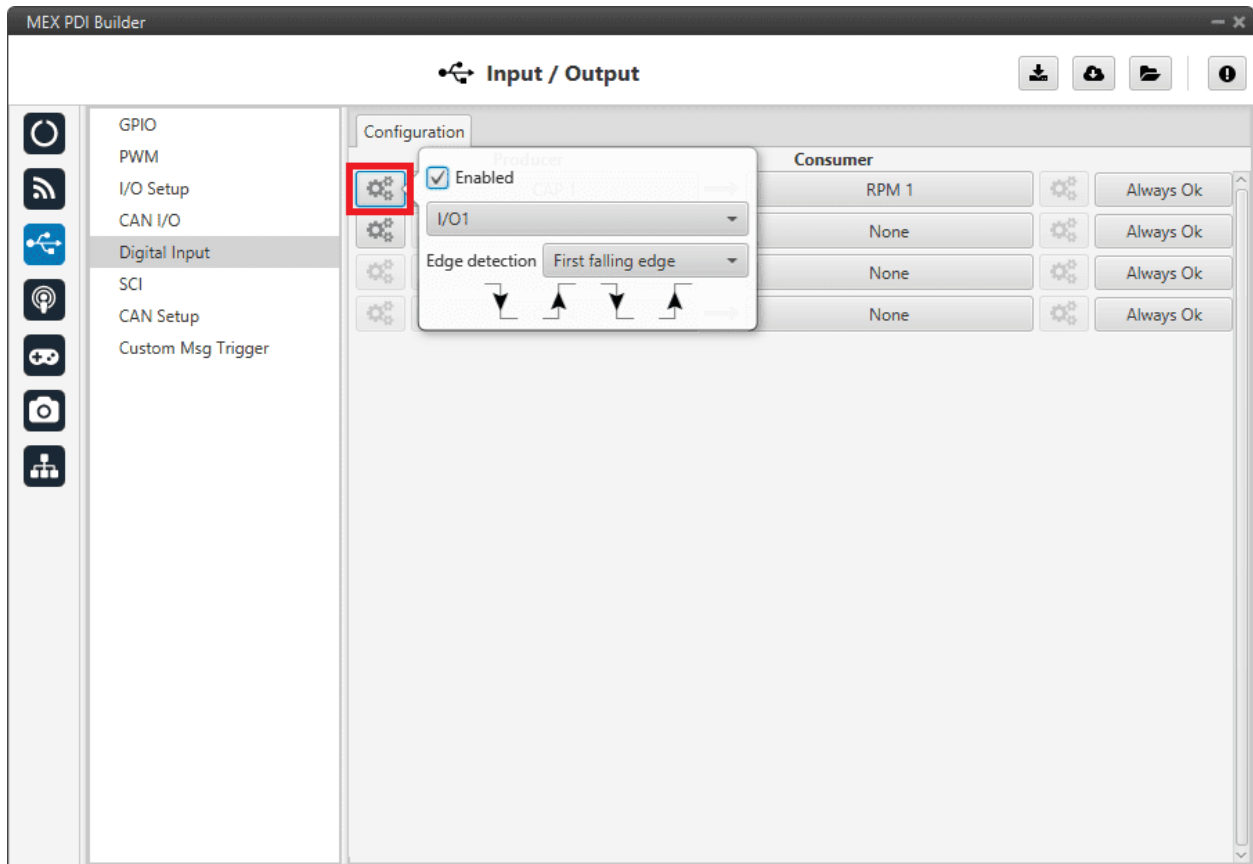


Fig. 34: Digital Input - Producer

In the pop-up window, users can check:

- Whether this producer is **enabled**.
- Which **pin** this CAP is associated to and, therefore, to which device is connected. They are associated in this way: **CAP 1** with **I/O1**, **CAP 2** with **I/O2**, **CAP 3** with **I/O3** and **CAP 4** with **I/O4**.
- How pulses are read and transformed into a digital signal (how they are processed). That can be configured with the **Edge detection** option.
 - **First rising edge**: With this option, when the rise of the pulse is detected, the data will start to be stored. Recommended when consumer is **PPM** or **Pulse**.
 - **First falling edge**: With this option, when the **fall** of the pulse is detected, the data will start to be stored.

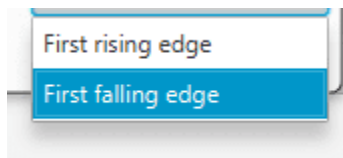


Fig. 35: Digital Input - Edge detection option

Note: By clicking on the marked arrows, it can also be configured as desired. For example, if the user has selected

the 'First rising edge' option, but clicking on the arrows gets the arrow scheme of another defined edge detection option, the name of the edge detection will not be 'First rising edge', but will become that edge detection option's name.

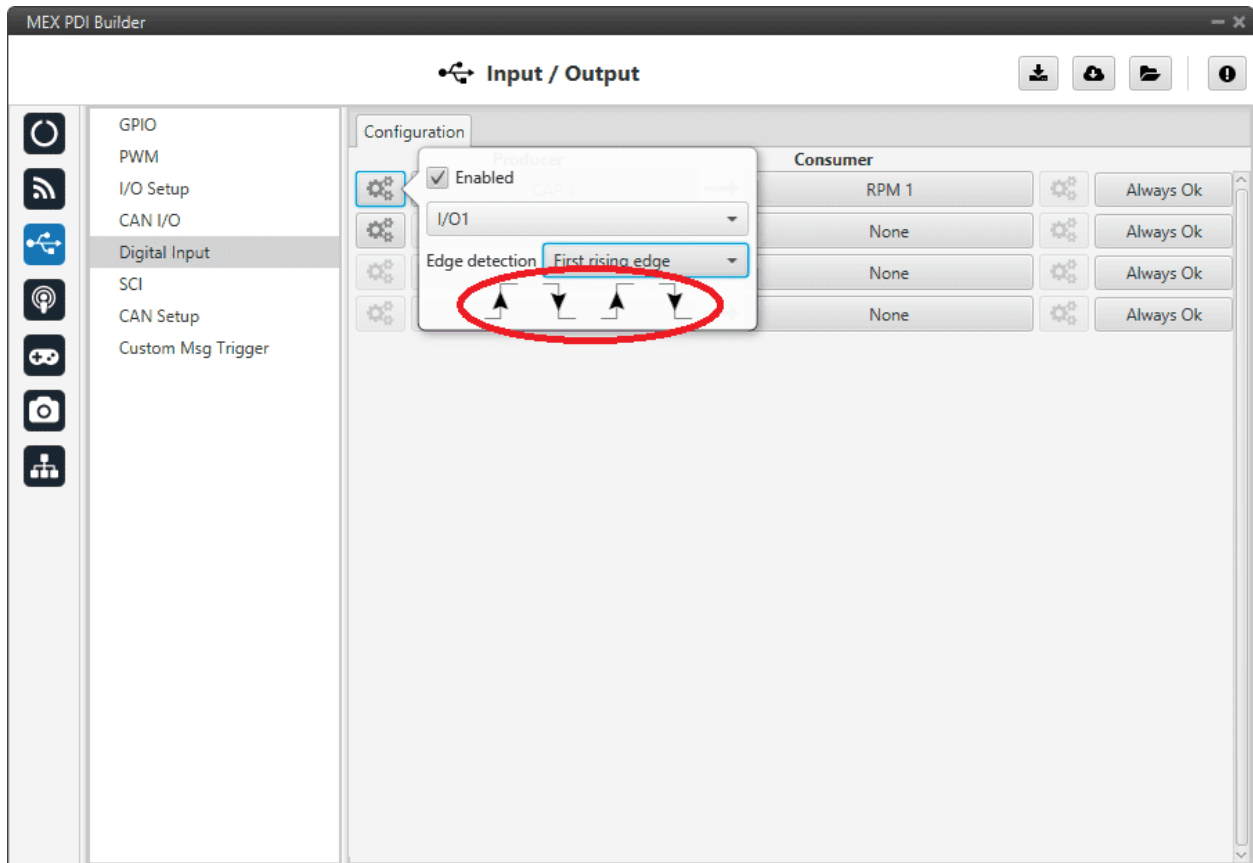


Fig. 36: Digital Input - Edge detection option

2. Click on the **Bind** button to select the type of **Consumer**, it is possible to choose PPM0 (Stick PPM), RPM 1-4 (RPM Sensor) or Pulse 1-4 (Pulse Sensor).

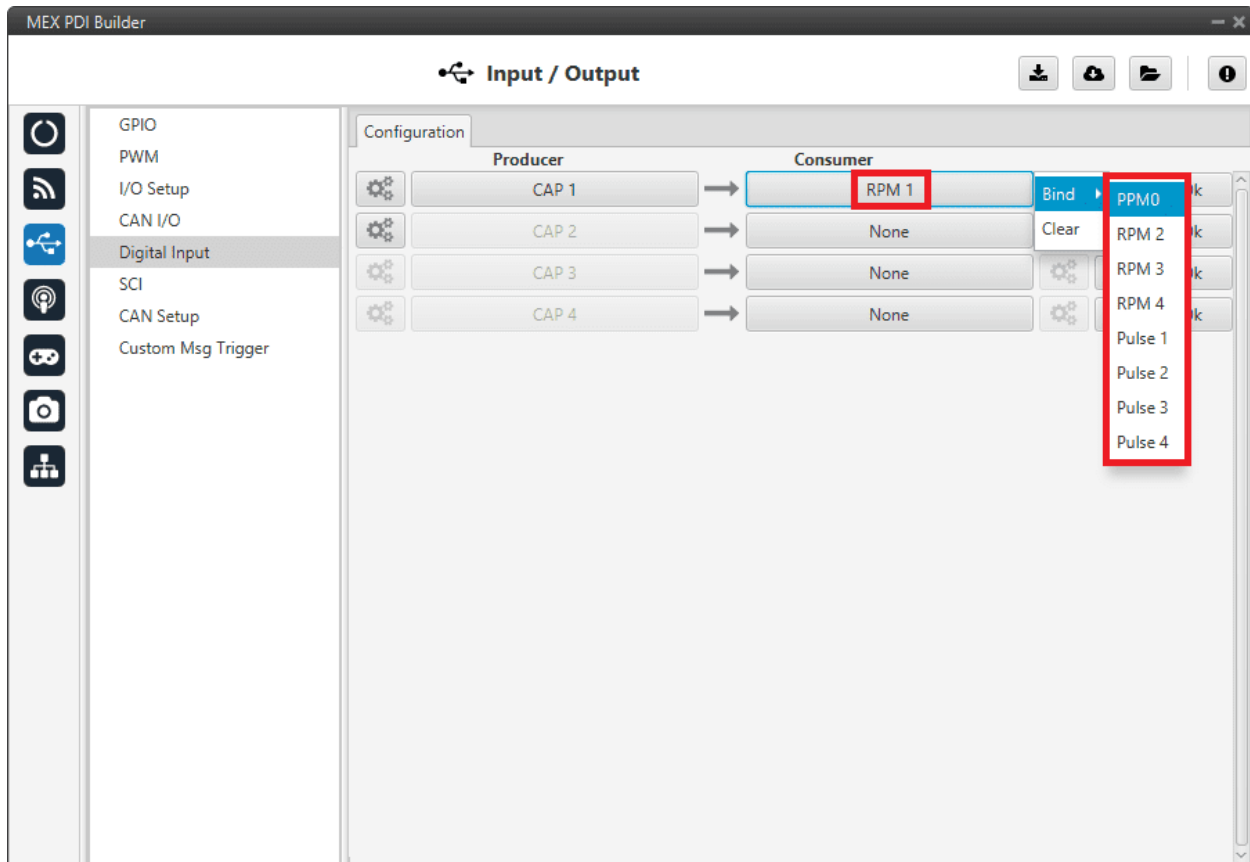



Fig. 37: Digital Input - Consumer

- **PPM 0:** The variable where the information is stored is 'PPM channel 1 output'. Stick PPM is configured in the *Stick* section.
- **RPM 1-4:** The variables where the information read here is stored are 'RPM 1-4'. For more information about RPM configuration, read the [RPM](#) section.
- **Pulse 1-4:** The variables where the information is stored are 'Captured pulse 1-4'. It is possible to configure it clicking on .

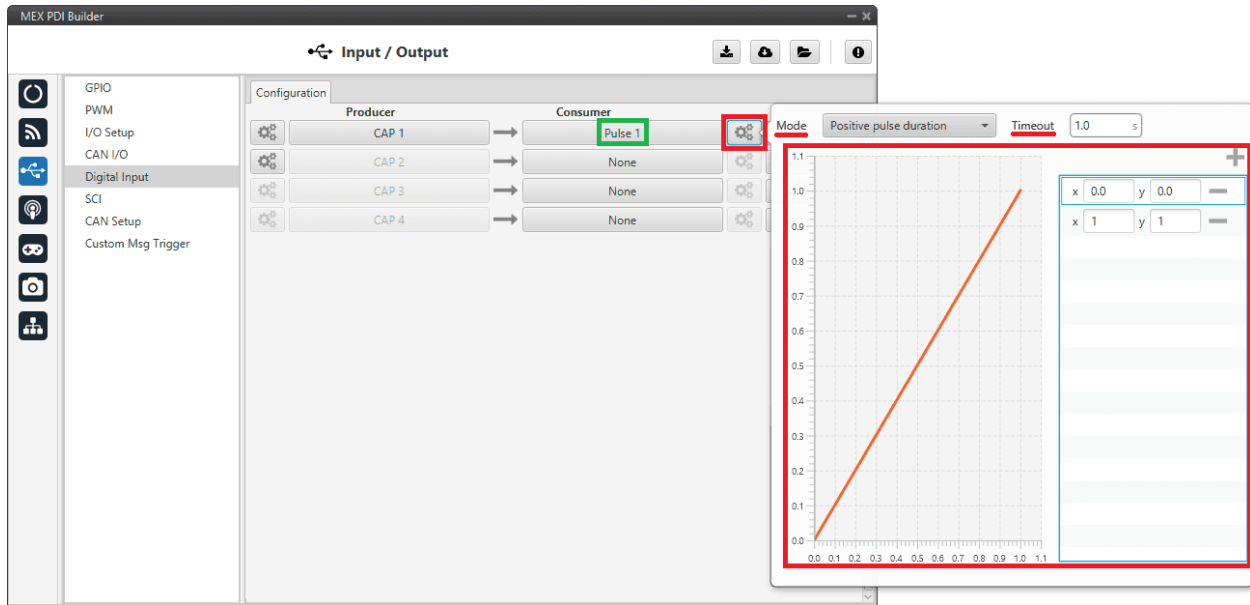


Fig. 38: Digital Input - Pulse

In the pop-up window, users will find the following options:

- Mode:
 - **Positive pulse duration:** The period of the pulse is obtained. It takes the time in 'High' state.
 - **Negative pulse duration:** The period of the pulse is obtained. It takes the time in 'Low' state.



Fig. 39: Positive/Negative pulse duration

- **Positive duty cycle:** The duty cycle of the pulse is obtained. It takes the time in 'High' state.
- **Negative duty cycle:** The duty cycle of the pulse is obtained. It takes the time in 'Low' state.

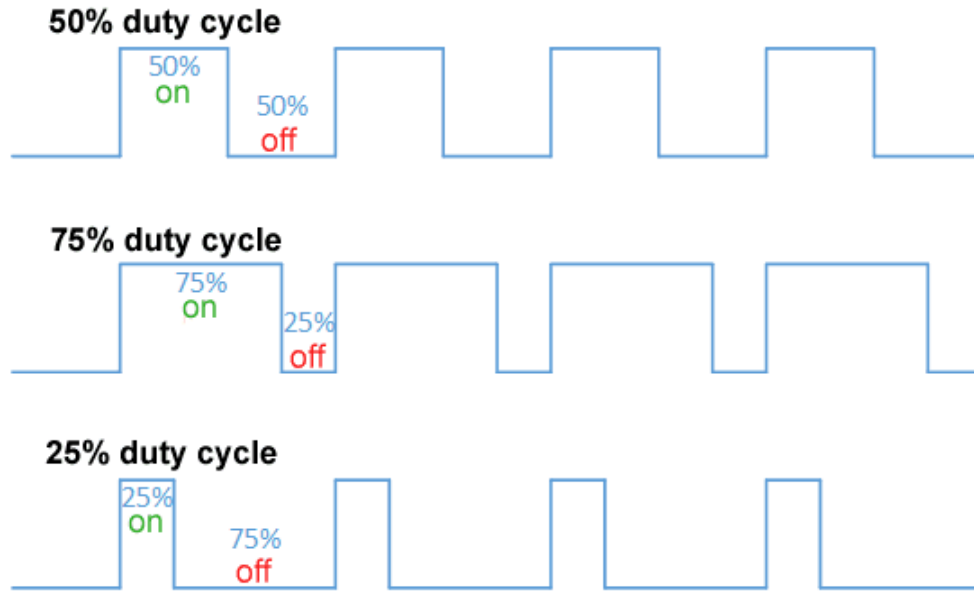


Fig. 40: Positive/Negative duty cycle

- **Time out:** This defines the time to consider that no signal is received.
- **Function:** Here the user can customize a function to handle values. Normally, a function is set with the points [0,0] and [1,1], so no transformation is applied (output = input). However, the user can configure it as desired.

Example

Let selected **First rising edge** as the edge detection option in Producer and the pulse that MEX has to read is a square signal with a period of 2 seconds and a duty cycle of 25% (see the image below).

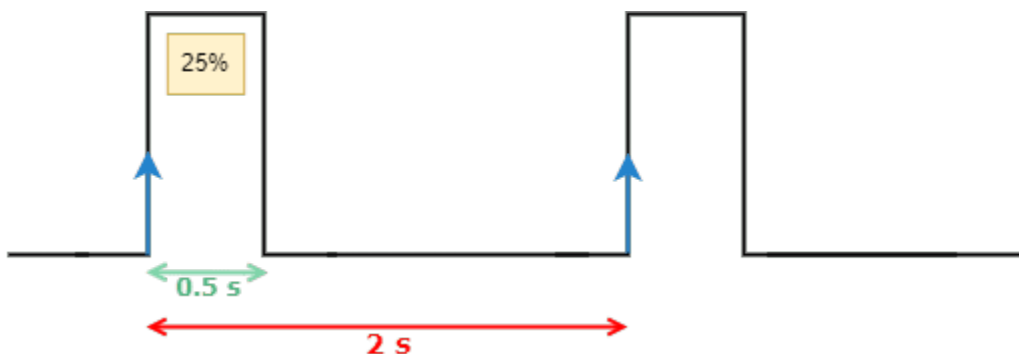


Fig. 41: Signal generated

If **Positive pulse duration** is selected as Consumer and it has been configured as in the previous image (*Digital Input - Pulse*), it will get **0.50 s** as value in the variable **Captured pulse** (*Captured pulse 1* in the following example), since it corresponds to the period of the “positive pulse” of that pulse.

Nonetheless, if **Positive duty cycle** is selected as Consumer, it would have **0.25** as value in **Captured pulse** (*Captured pulse 2* in the following example), since it corresponds to the positive cycle of that pulse.

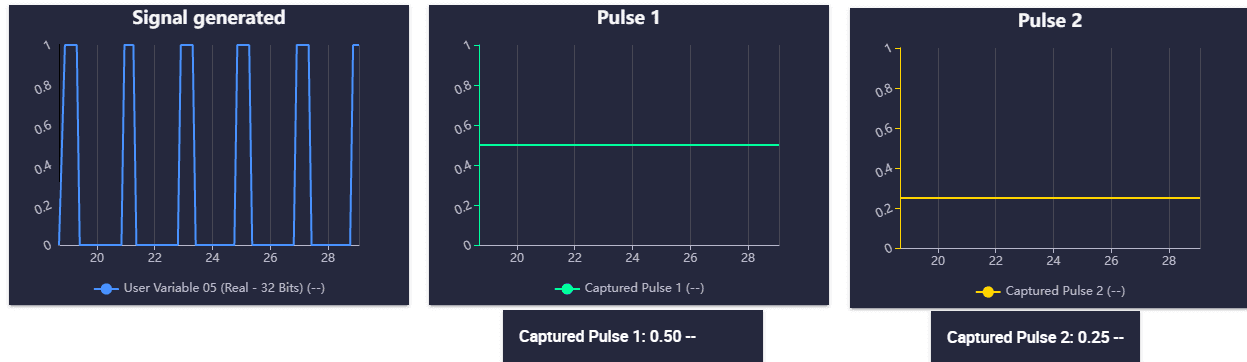


Fig. 42: Digital Input example

2.3.6 SCI

MEX 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.

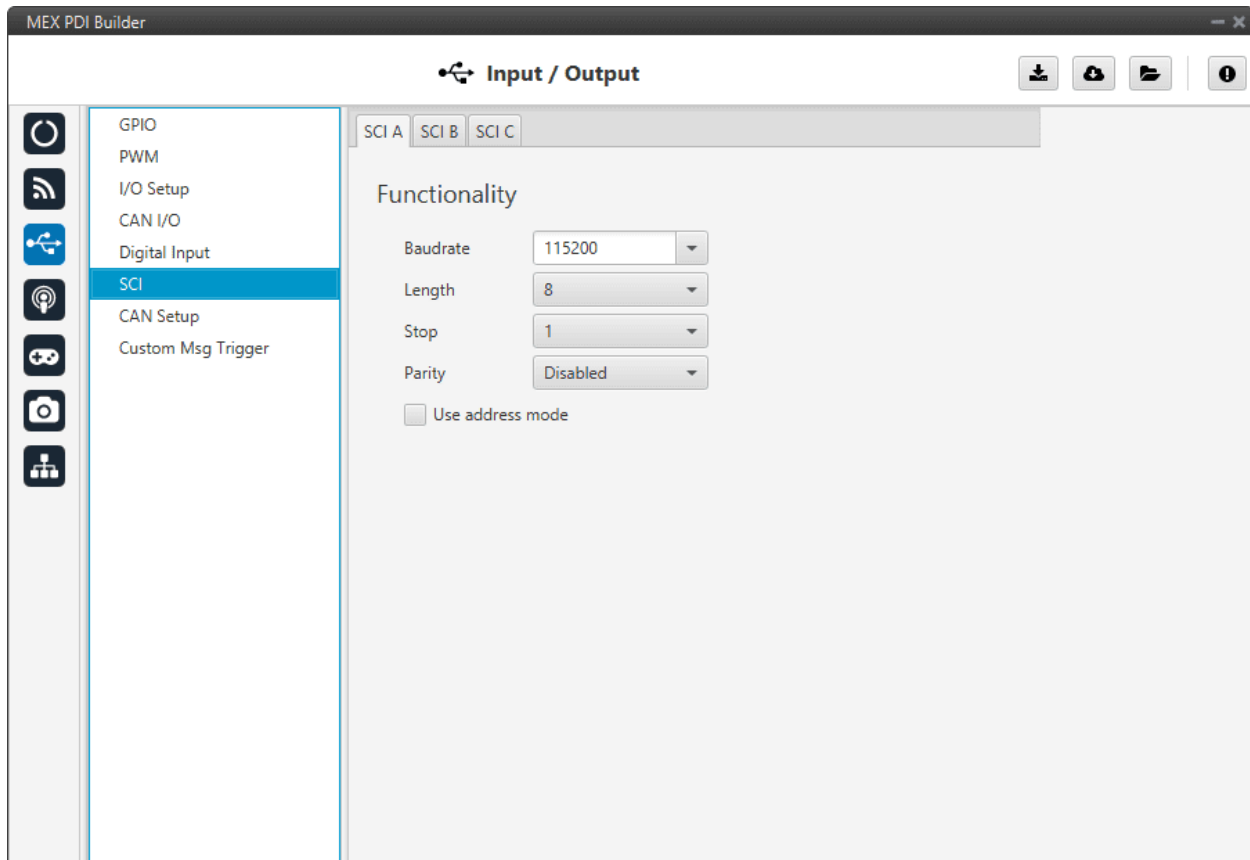


Fig. 43: SCI section

- **Baudrate:** How fast data is sent over a serial line.
- **Length:** 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 port RS232 A, SCI B to port RS232 B and SCI C to port RS485.

2.3.7 CAN Setup

Main screen to enable an internal resistor and configure baudrate and reception mailboxes of each CAN Bus.

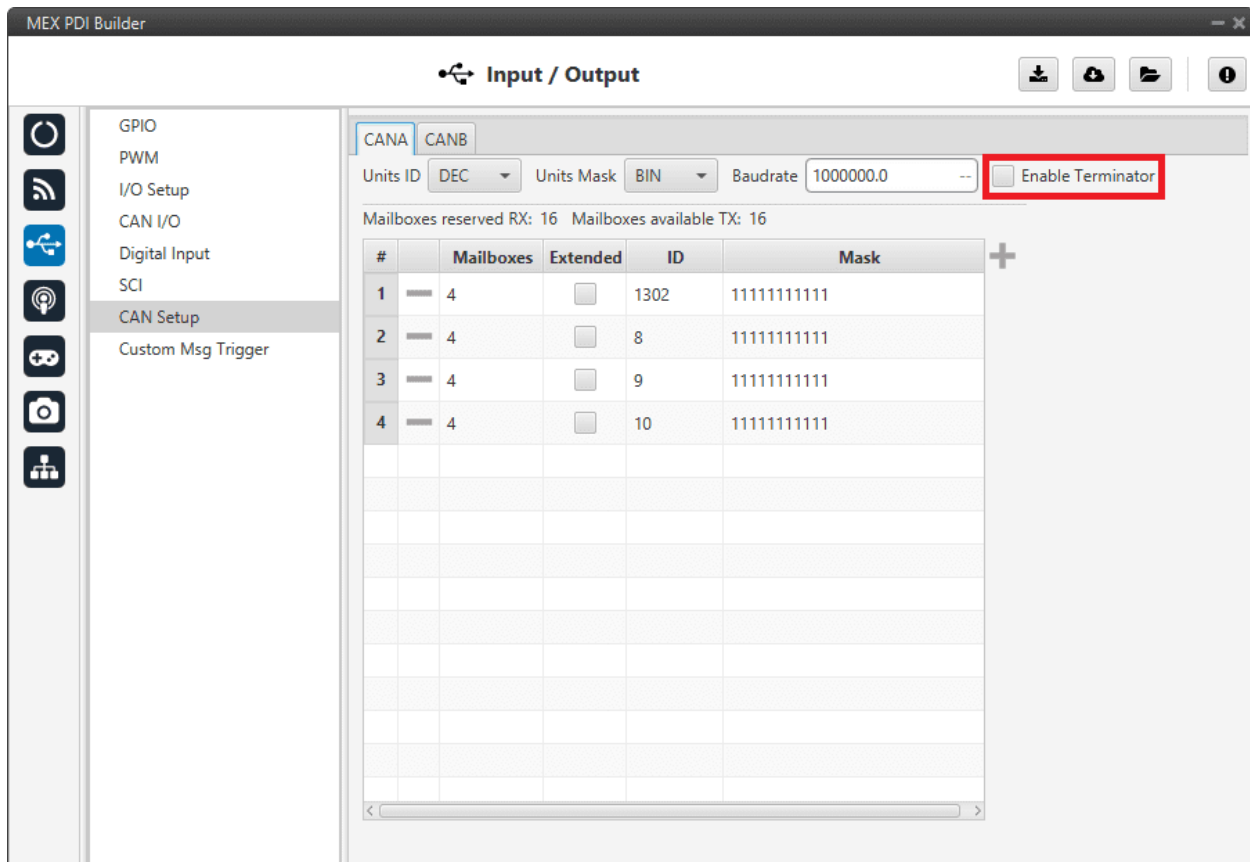


Fig. 44: CAN Setup section

By activating **‘Enable Terminator’**, the internal MEX resistor of 120 Ω is activated. For more information on this resistor, refer to the [Electrical diagram of CAN bus -> Hardware Installation](#) section of the **MEX Hardware Manual**.

More information about Mailboxes can be found in the [Mailboxes section](#) of the **1x PDI Builder** user manual.

2.3.8 Custom Msg Trigger

This menu allows the user to **send** *custom messages* (which have been previously configured) every time there is a **new magnetometer reading/measurement**.

It works similar to the [Custom CAN TX](#) and [Custom Serial TX](#) automations of **1x PDI Builder** software.

In this case, the “automation” will work like this:

- Events: Each new magnetometer reading/measurement.
- Action: Sending of the custom message already configured.

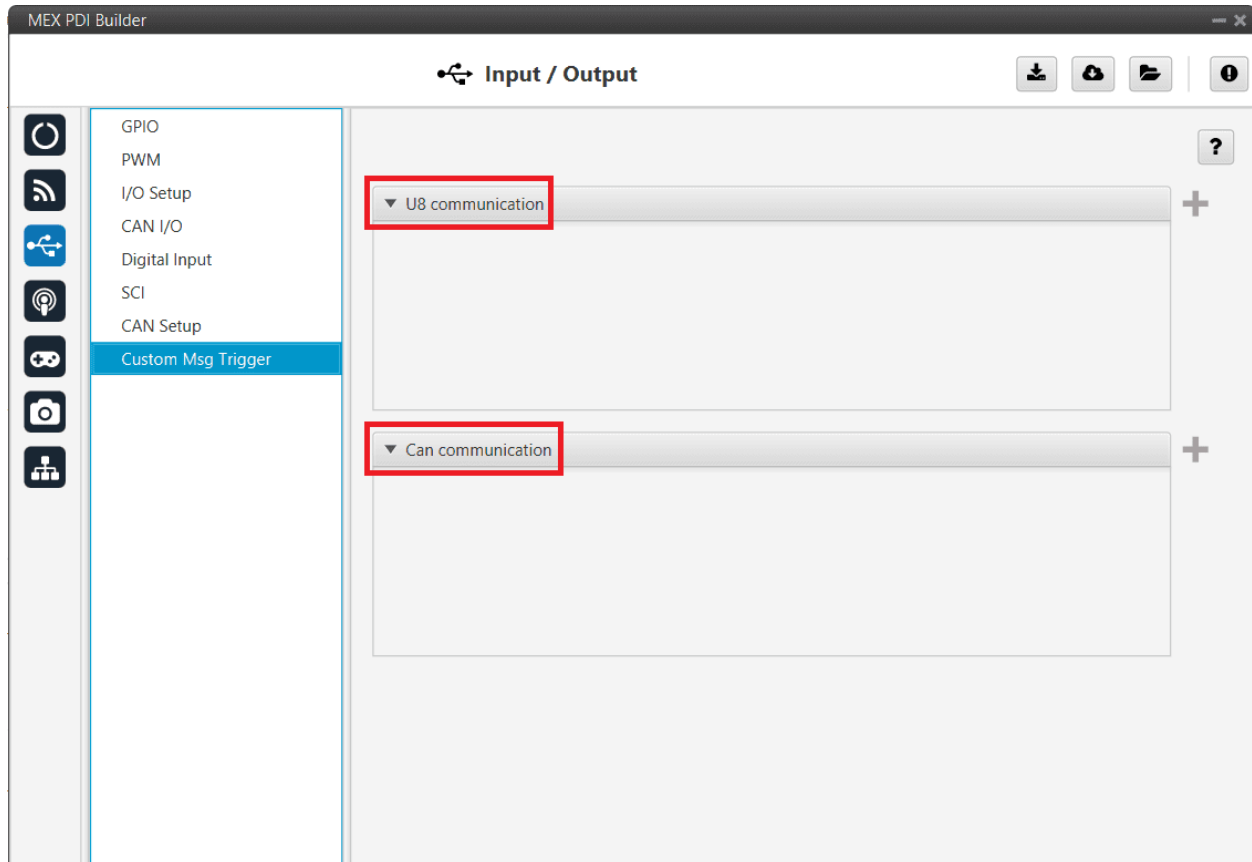


Fig. 45: Custom Msg Trigger section

Users can configure custom messages to be sent via **serial** or **CAN**:

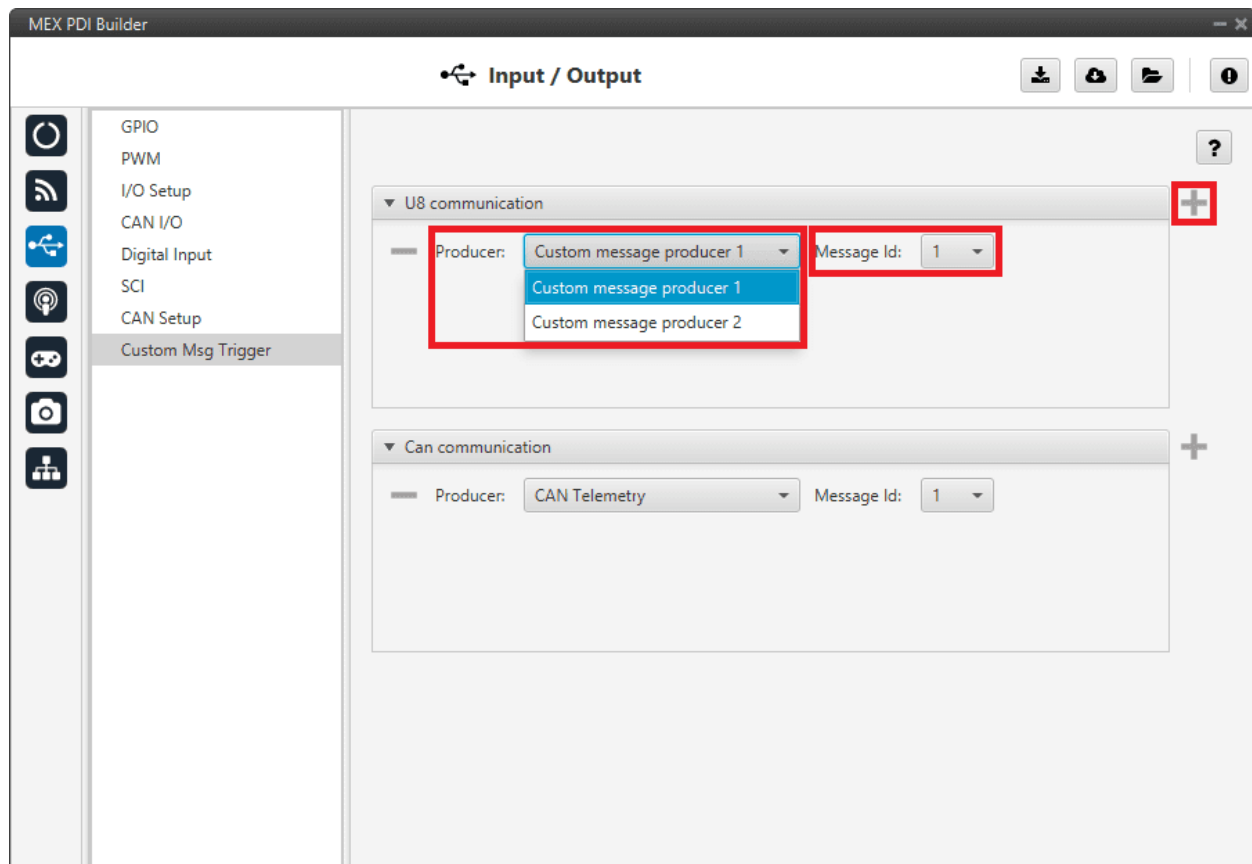


Fig. 46: Custom Msg Trigger section - U8 communication

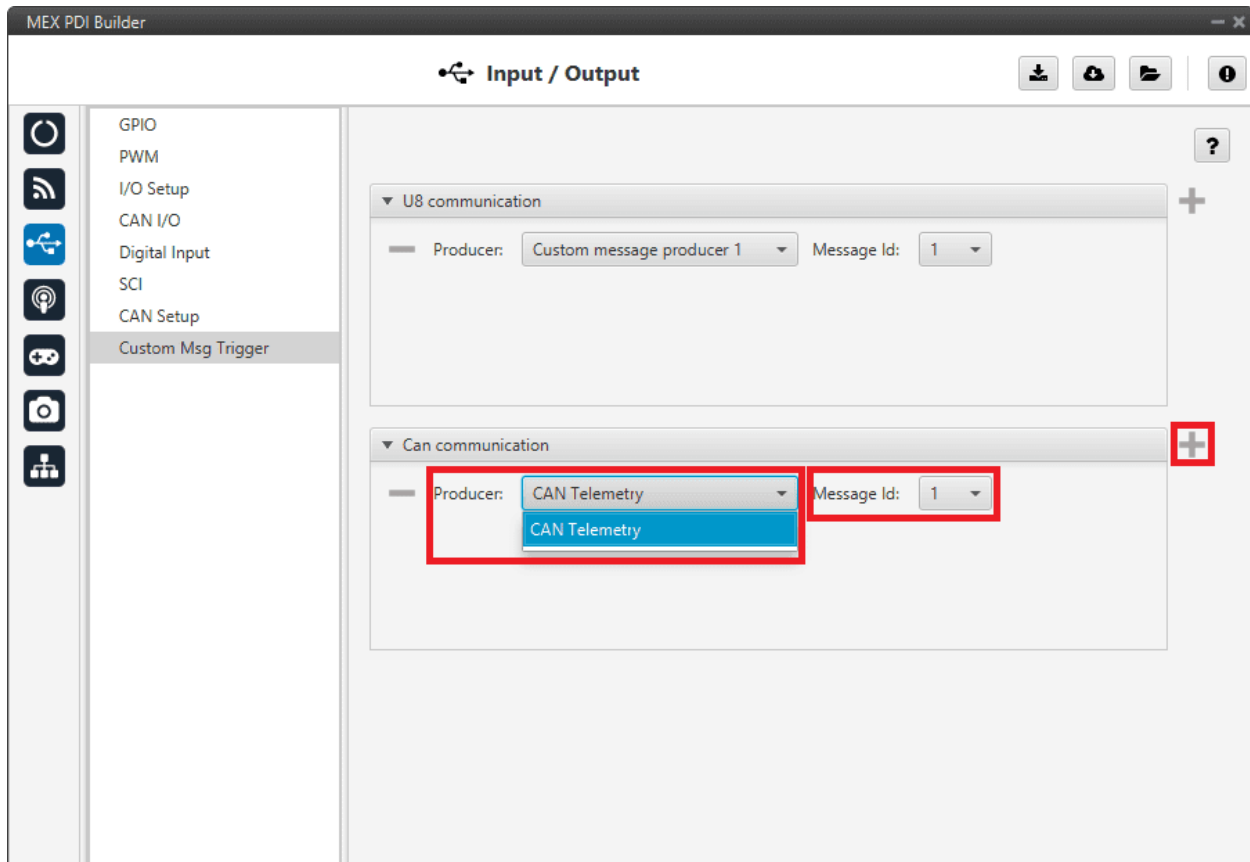


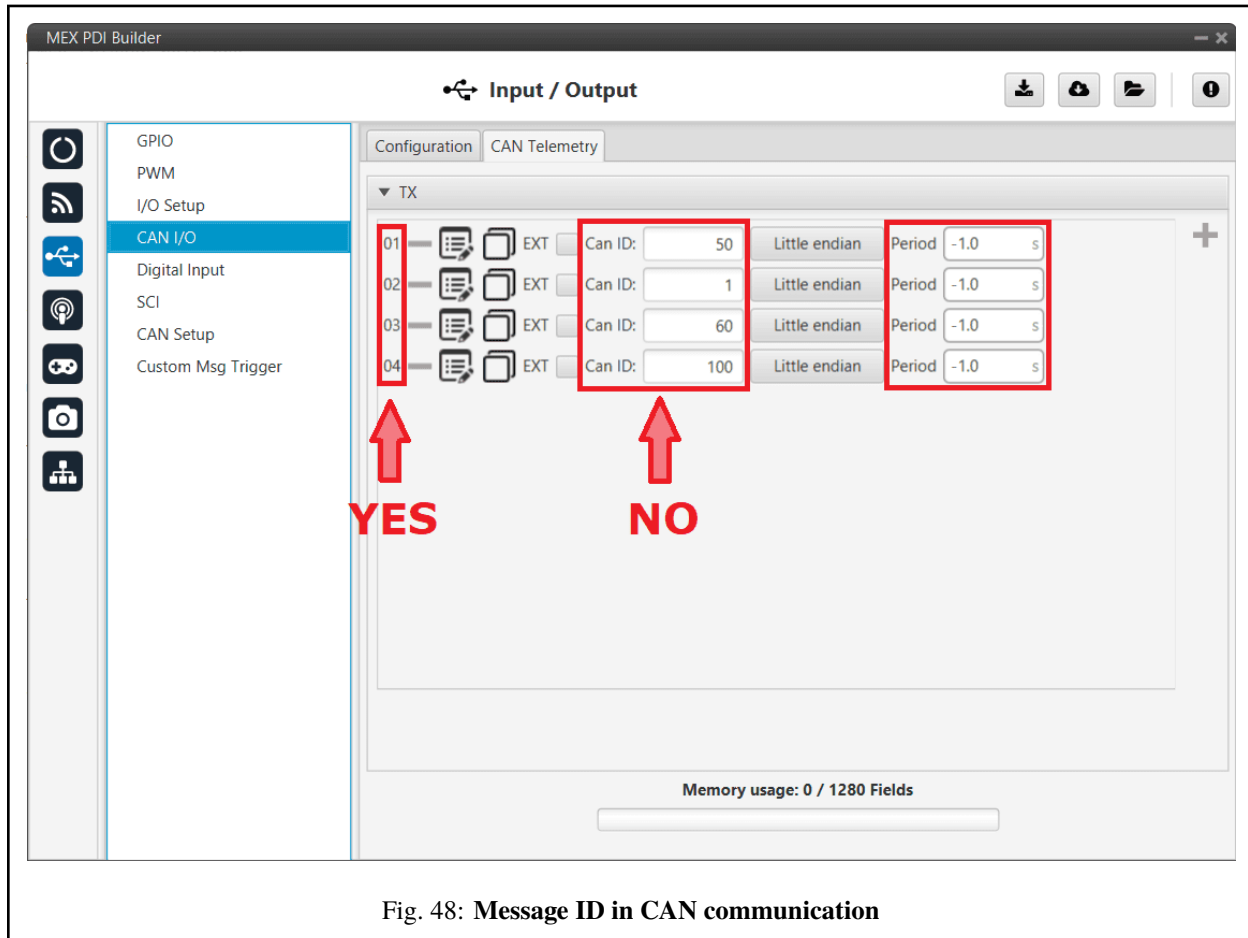
Fig. 47: Custom Msg Trigger section - CAN communication

The two parameters to configure in this menu are:

- **Producer:** The user has to specify where the custom message is located (mailbox):
 - For **serial**: Custom message producer 1 or 2.
 - For **CAN**: CAN Telemetry
- **Message ID:** The number of the custom messages that will be sent.

Warning:

- As it is used to send a single message on demand, in its configuration in **Custom Messages**, the user has to set its **period to -1**. This way, this **message will only be sent when there is a new magnetometer reading/measurement**.
- For CAN communications, do not confuse it with the **CAN ID** of the custom message:



2.4 Communications

Ports configuration allows to configure which communication ports (Commgr Ports in I/O Setup) will be used for communication.

MEX can be configured to **route** VCP messages for an external Veronte device with a known ID through a certain port.

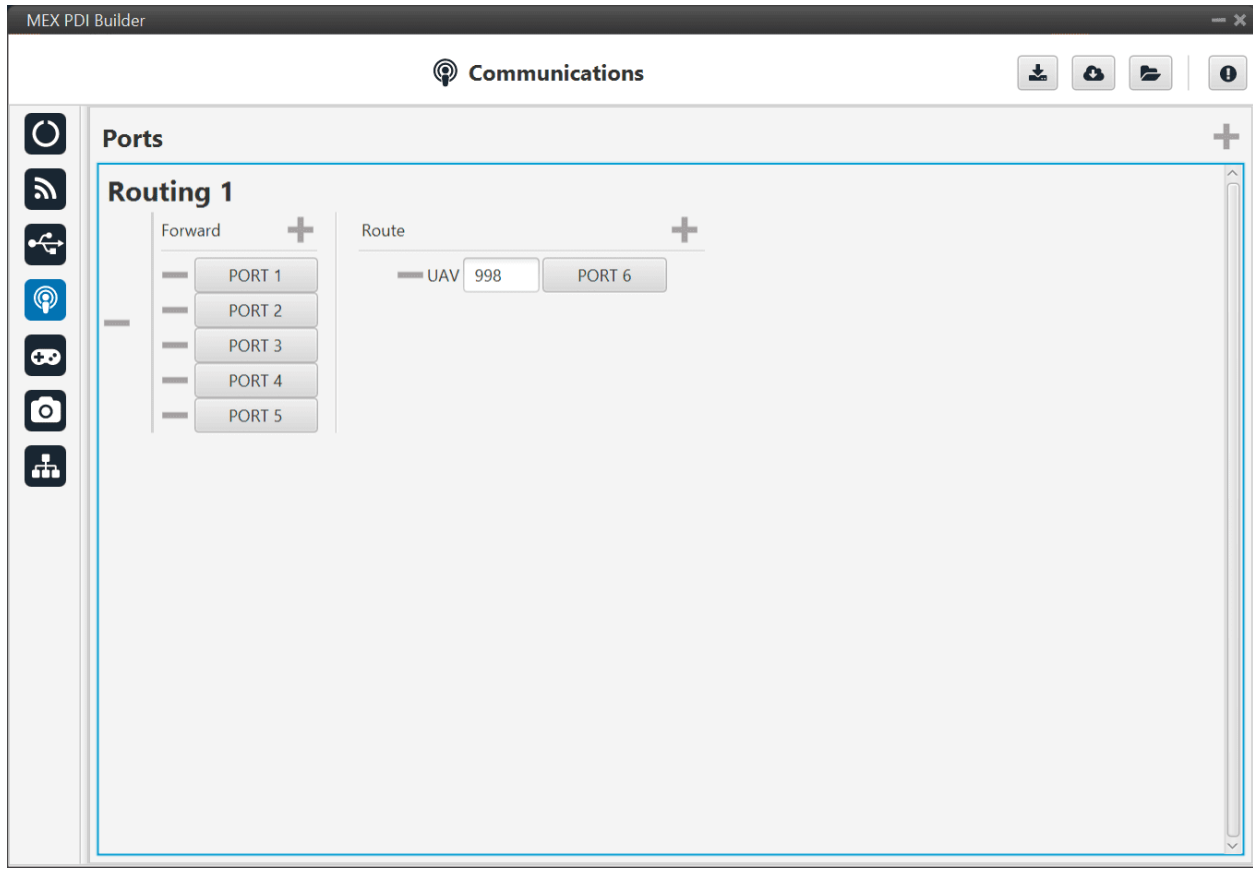


Fig. 49: Communications menu

Each port can be configured as either of the following options:

- **Forward:** Any messages generated by this unit (i.e. Telemetry or response messages to certain commands) will be sent through these ports.
- **Route:** Any messages received at any Commgr Port with the defined address will be re-sent through the defined port. It is possible to route several addresses through the same port, but is not possible to route the same address through several ports. Only the first configured port will be used. Routing also applies to messages generated by the unit for the defined address.

Note: The same port cannot be used as Forward and Route at the same time.

It is possible to define up to 4 routing setups, which can be switched using the [Ports action](#) in the **Automations** menu of the **1x PDI Builder** software.

Routing 1 will always be selected by default when booting MEX.

2.5 Stick

The wired connected transmitter is configured through the following tabs.

2.5.1 PPM

This tab provides the options to configure a Pulse Position Modulation (PPM) radio controller to control the platform fitted with the autopilot.

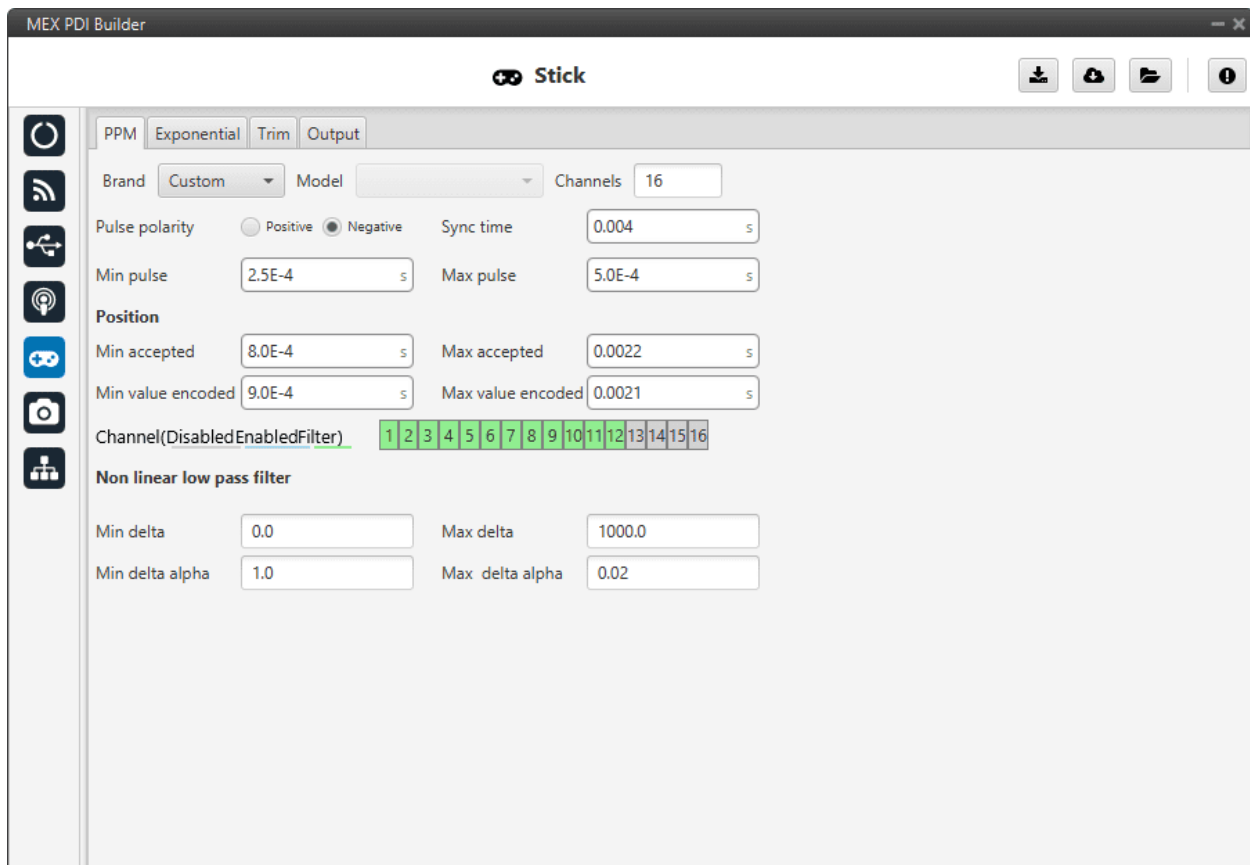


Fig. 50: PPM section

- **Brand, Model and Channels:** MEX PDI Builder has been configured to provide the user with the expected parameters to configure different transmitters models.

Brand	Models	Channels
Futaba	8J/10J/12K/14SG	8
	12K/14SG	12
	T18SZ	8
Jeti	DC 16/DC 24	16
FrSky	Taranis X9D	8
	Horus X12S	8
TBS	Croosfire	8
Embention	Stick Expander	16
Custom	-	-

- Custom: If the user's transmitter is not among those mentioned above, choose this option and replace the parameter values with the appropriate ones.

- **Pulse polarity:** Indicates the pulse polarity.
- **Sync time:** Minimum time on the PPM output till the next frame. It tells the receiver to reset its channel counter.
- **Minimum/Maximum pulse:** Pulse length, it depends on the system and it is a constant value (usually 0.2-0.5 ms).
- **Position**
 - **Minimum/Maximum accepted:** Pulse length accepted for each channel. Standard for R/C servos uses a pulse of 1 ms for the maximum position at one end, 1.5 ms for the midpoint and 2 ms for the maximum position at the opposite end.
 - **Minimum/Maximum encoded:** If there is noise and the signal is varying around the minimum/maximum values accepted, 1x autopilot will encode those values to the ones set here. For instance, a pulse length between 0.8-0.9 ms will be considered as one of 0.9 ms.
 - **Channels:** Sets the number of channels accepted. Besides, it is possible to Disable/Enable/Filter each channel individually.
- **Non linear low pass filter**
 - **Minimum/Maximum delta:** Default parameters are recommended.
 - **Minimum/Maximum delta alpha:** Default parameters are recommended.

The figure below shows the PPM signal that arrives to MEX:

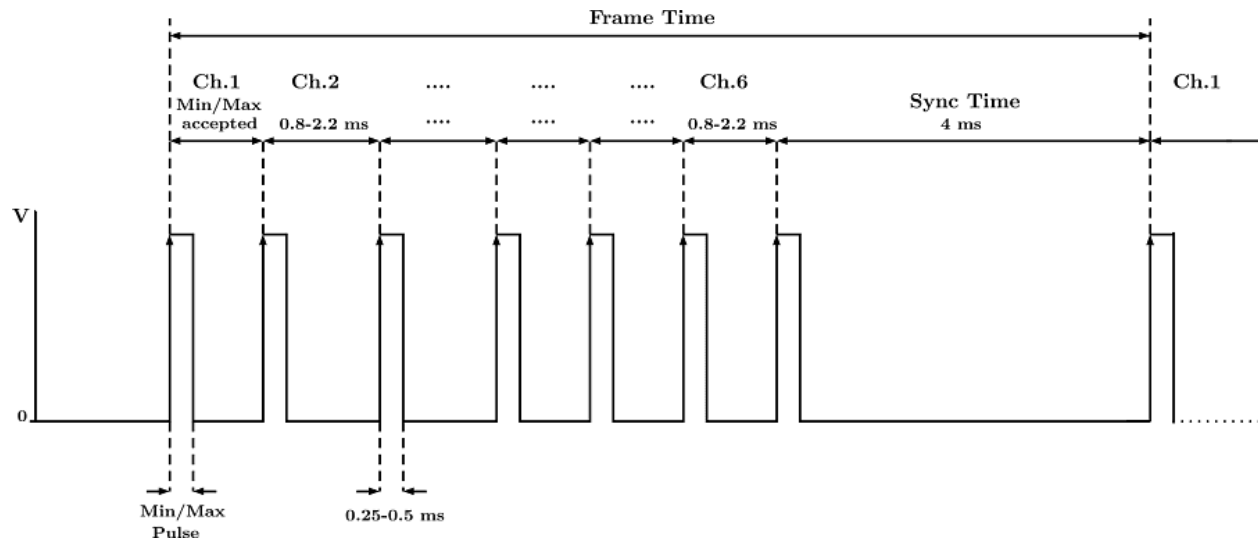


Fig. 51: PPM signal

2.5.2 Exponential

The second tab allows the user to define an exponential stick response for every channel.

The allowed inputs range from 0 to 1 and there is a graph showing the generated response curve, as can be seen in the figure below.

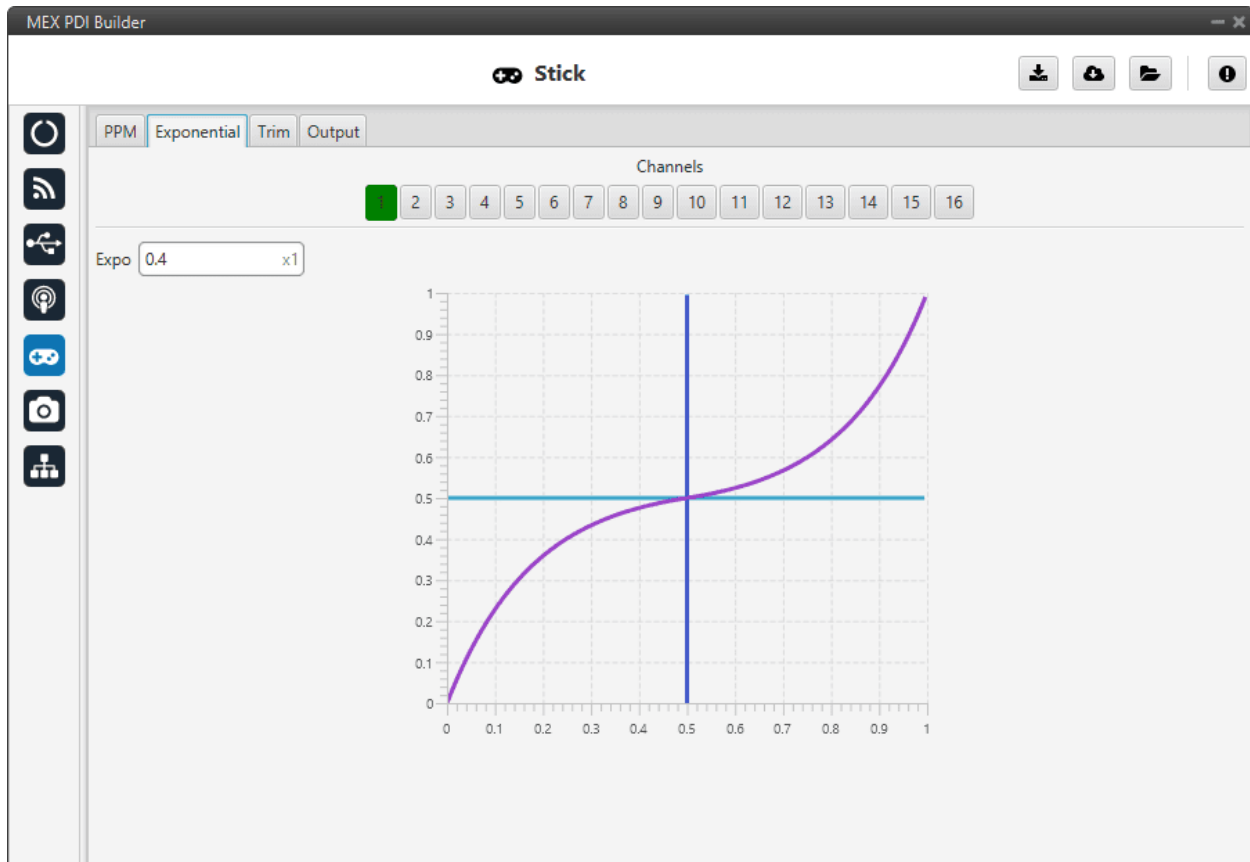


Fig. 52: Exponential section

The **X axis** of the graph corresponds to the stick input and the **Y axis** is the result of applying the exponential function to that stick input.

2.5.3 Trim

The third tab available is the Trim option.

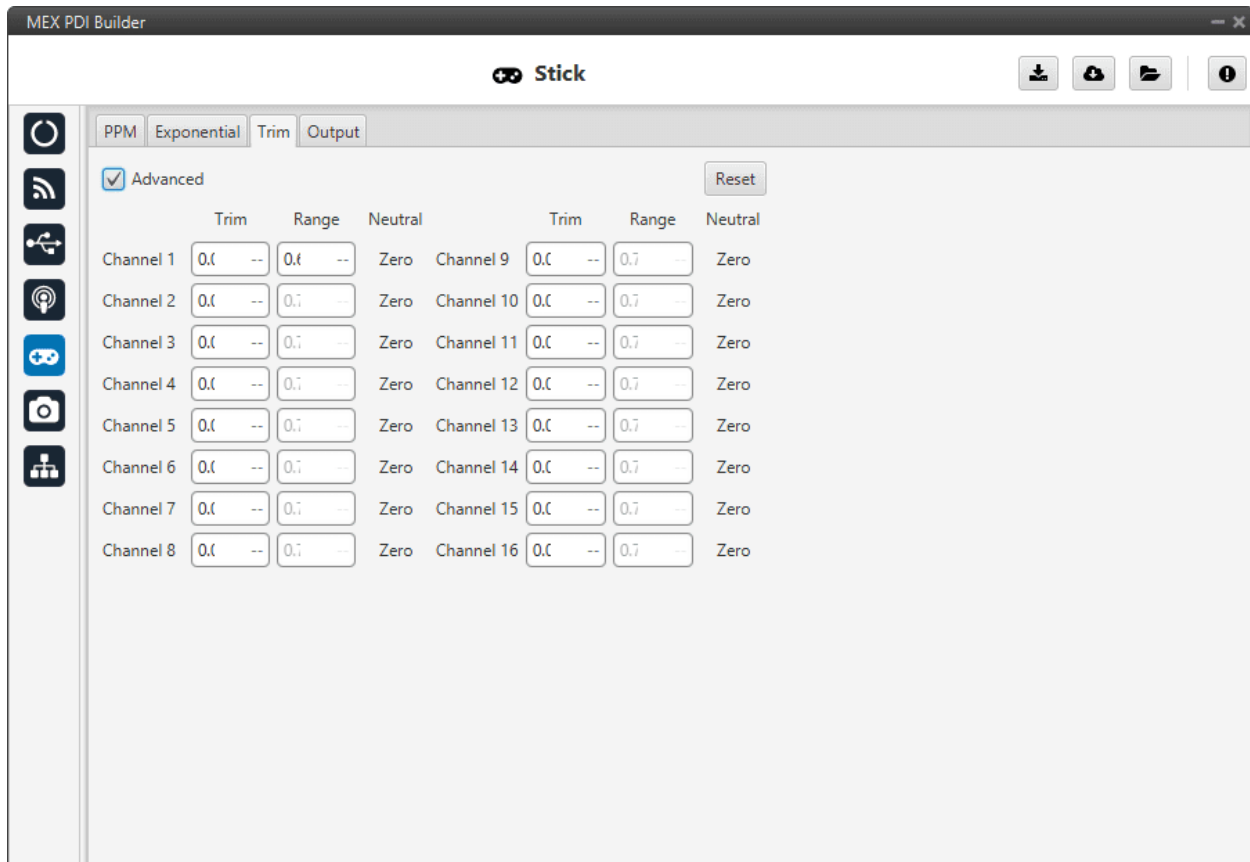


Fig. 53: Trim section

By enabling the **Advanced** option, the user can set the expected trim values manually. The user should have a deep knowledge on its transmitter if this option is selected.

Finally, on the right hand side, the **Reset** button puts every parameter back to 0.

2.5.4 Output

In this menu the user sets the receiving port and process the incoming commands.

Once the stick has been configured, the commands that arrive at the ground autopilot have to be sent to the air unit.

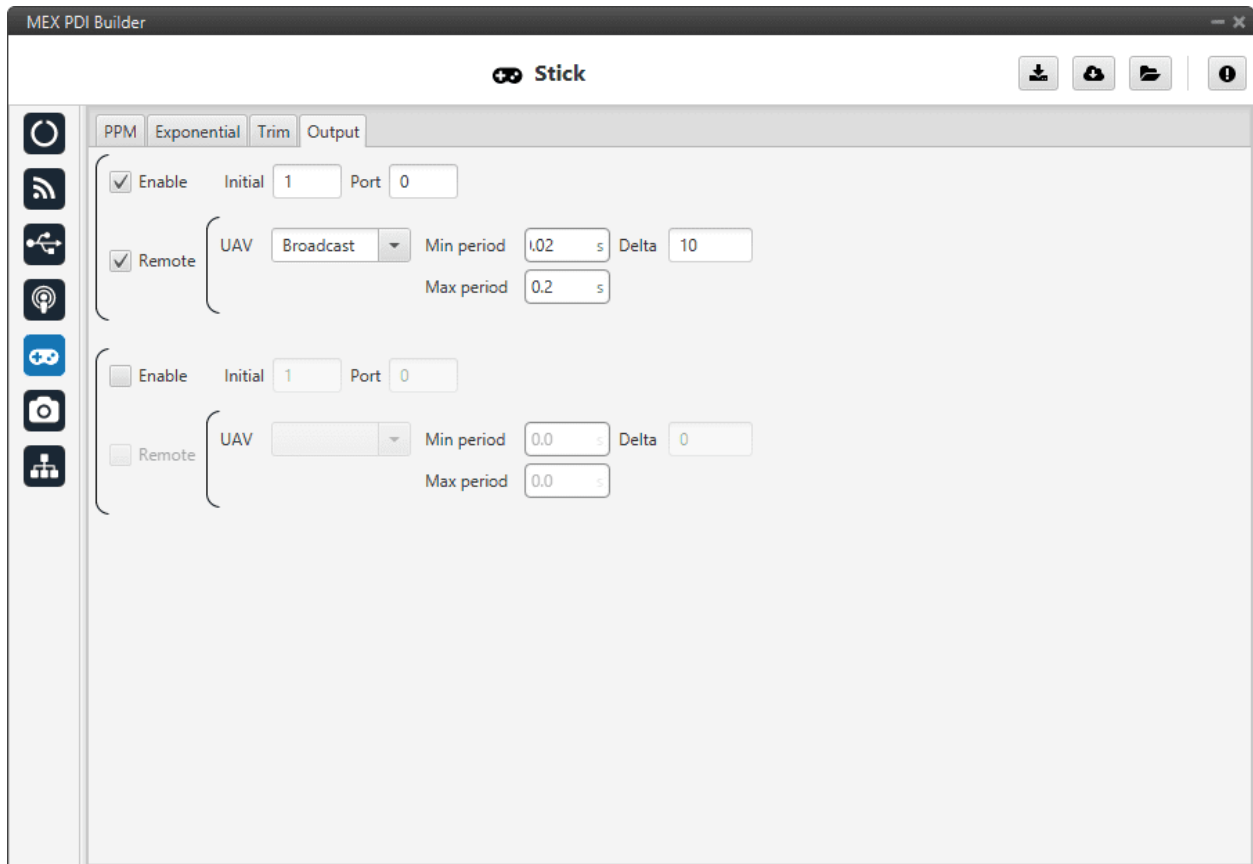


Fig. 54: Output section

In this menu, the following parameters can be configured:

- **Enable.**
- **Initial Channel at destination:** The user indicates to which channel of the air autopilot will be sent the first channel received in the ground unit. The channels arrive at the platform in order and without spaces between them.

For example, if at the GND channels 6,7,8,9 and 10 are enabled, the AIR will receive channels 1,2,3,4 and 5. Therefore channel 6 of the stick will be channel 1 in the AIR configuration.
- **Port:** If more than one transmitter is configured, each transmitter must be configured on a different port. This has to match the port set on the air unit.
- **Remote:** It has to be enabled if the user wants to allow the delivery of the commands to the platform.
 - **UAV:** The address of the UAV that receive the commands has to be indicated. The following options are available:
 - * App 2: Veronte Ops address.
 - * Broadcast: The commands are sent to all units on the network. **We recommend this option.**
 - * 1x v4.X XXXX: The address of a specific air unit.
 - **Min period:** As the period is the inverse of the frequency, this defines the **maximum frequency**. Therefore, to give the pilot more control, this defines the frequency that is set when the **stick is commanding**. We recommend **0.02s**.

- **Max period:** As the period is the inverse of the frequency, this determines the **minimum frequency**. Thus, to free up bandwidth, this defines the frequency that is set when the **stick is idle**. We recommend **0.2s**.
- **Delta:** This parameter determines whether the frequency is set to the minimum or maximum period set above.

If MEX detects a **change above the delta value**, the frequency goes to the **maximum frequency** (minimum period). While if the **changes are less than this value**, it switches to the **minimum frequency** (maximum period). We recommend **10**.

Note: An example of how to configure a stick can be found in the [Stick -> Integration examples](#) section of the **1x PDI Builder** user manual.

2.6 Devices

2.6.1 Jetibox

MEX can simulate a Jetibox to read telemetry from Legacy Jeti devices.

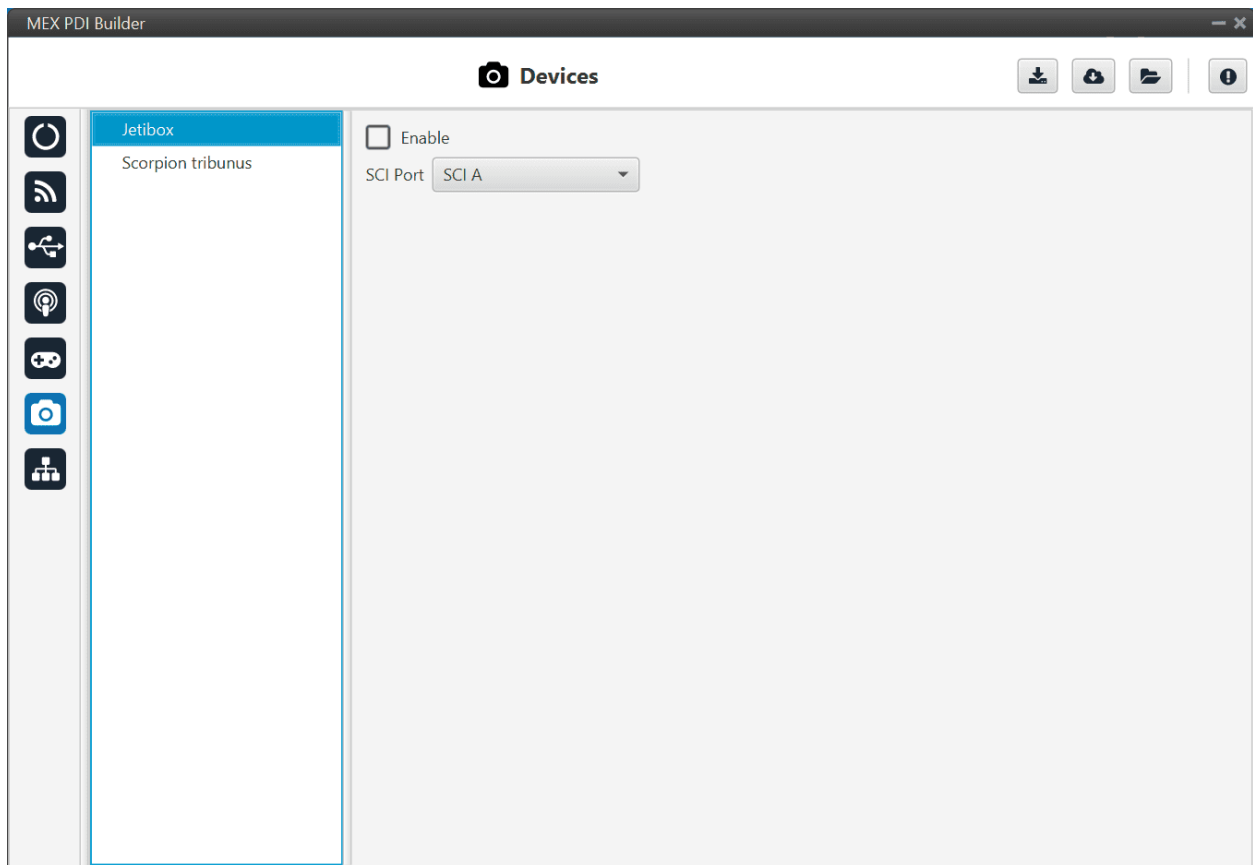


Fig. 55: Jetibox section

- **Enable:** It can be enabled or disabled by the user.

- **SCI Port:** A SCI port must be selected. The available options are: **SCI A**, **SCI B** or **SCI C**.

Besides, a configuration is needed in *SCI* and the *I/O Setup section*.

An example of this can be seen in *Jetibox -> Integration Examples* section.

Note: The serial port will be totally reserved for this, so it will not be usable to other things and the I/O Setup affecting it will be ignored.

2.6.2 Scorpion tribunus

MEX can read telemetry from Tribunus ESC connecting it to a serial port.

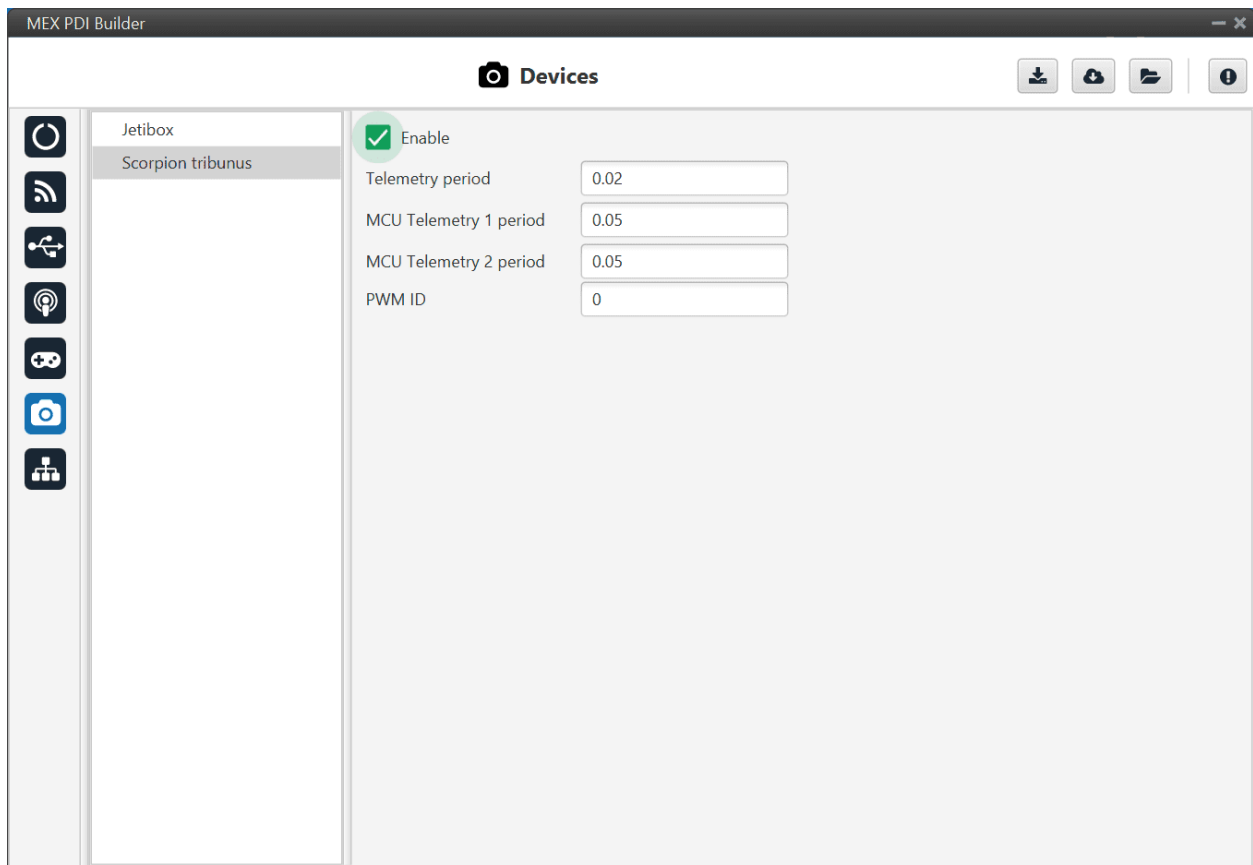


Fig. 56: **Scorpion tribunus** section

The following parameters are configurable:

- **Enable:** It can be enabled or disabled by the user.
- **Telemetry period.**
- **MCU telemetry 1/2 period:** Telemetry period for MCU devices.
- **PWM ID:** PWM ID associated to the Scorpion Tribunus.

The serial port has to be configured in the *I/O Setup section*.

An example is explained in *Scorpion -> Integration Examples* section.

Note: The serial port will be totally reserved for this, so it will not be usable by other things and the I/O setup affecting it will be ignored.

2.7 Arbitration

2.7.1 Arbitration

MEX is able to send PWMs using arbitration in the same way **Veronte Autopilot 4x** does. This functionality requires to be enabled in the **Enable** checkbox.

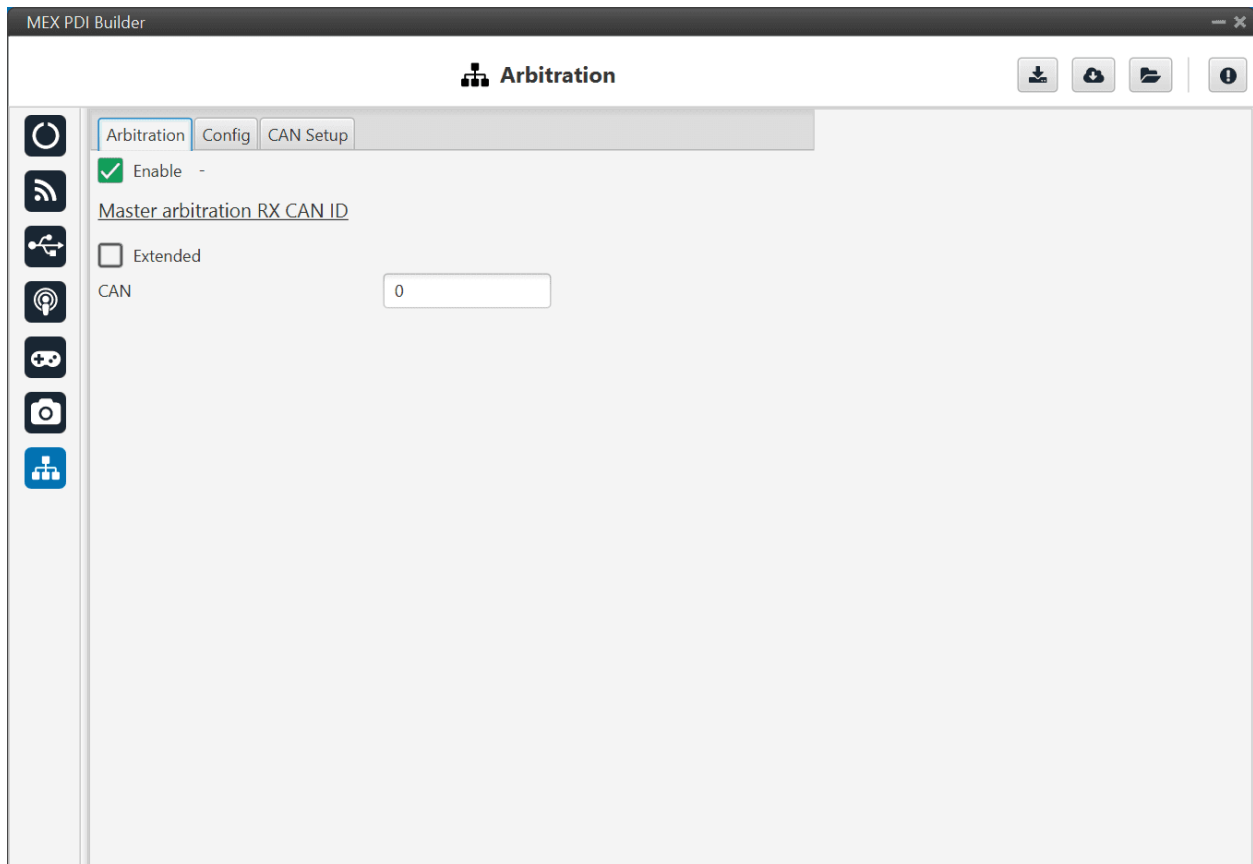


Fig. 57: Arbitration section

Master arbitration RX CAN Id is exclusive for MEX when it is used alongside a Veronte 4x Autopilot. If **enabled**, when an arbitration message is received from the Arbiter of the Veronte 4x Autopilot (**with the CAN Id configured here**), the selected autopilot will be updated according to the data received.

2.7.2 Config

In this tab the parameters of the arbiter algorithm are set.

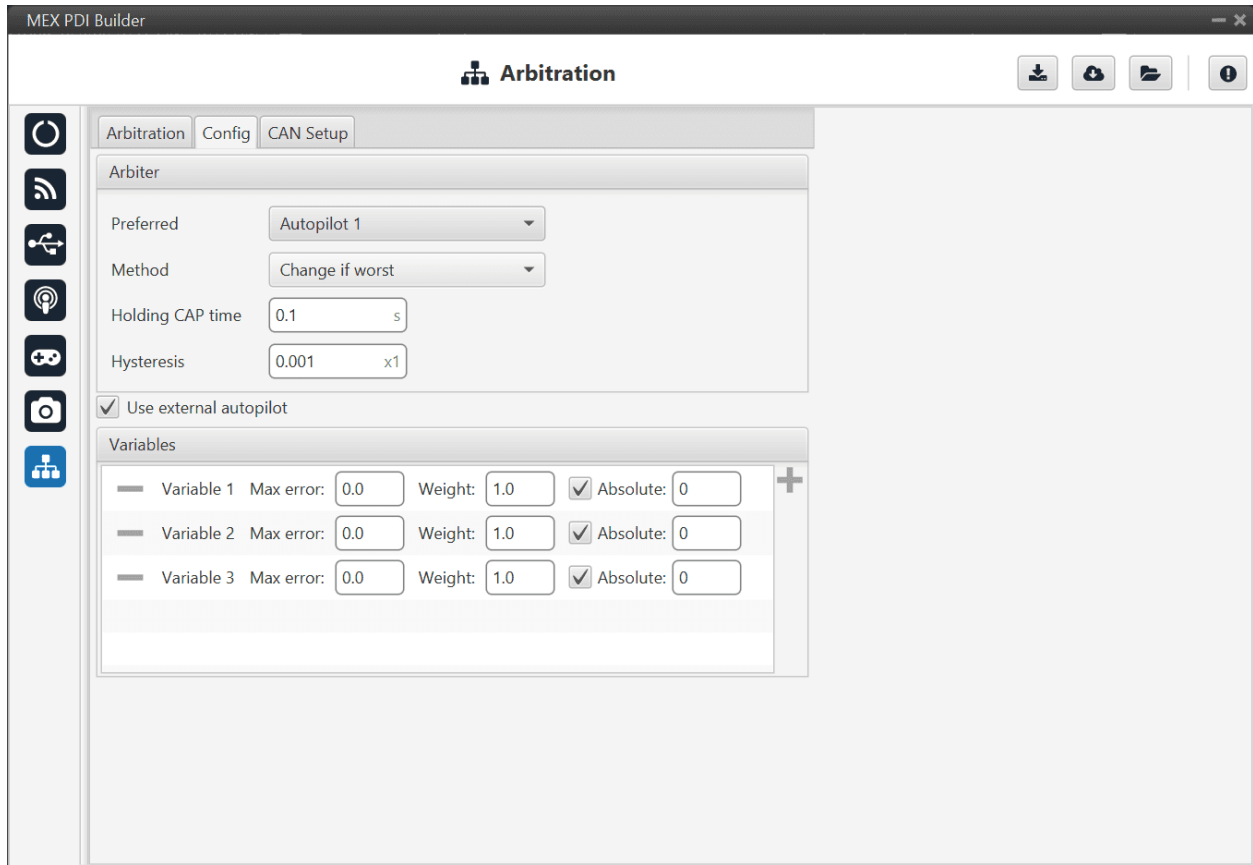




Fig. 58: Configuration section

1. **Preferred.** The preferred autopilot will be chosen in case of a score draw. **Fixed while ok** mode will always select this autopilot first.
2. **Method.** The method of arbitration can be chosen by the user. The available options are:
 - **Always best.** It chooses always the best autopilot.
 - **Change if worst.** The arbiter will only switch if the currently selected autopilot has the worst score. In that case, it will switch to the one with the best score.
 - **Round robin control.** Using the **Holding CAP** time parameter, the **arbiter** will periodically switch between autopilots. This mode is meant for testing purposes only.
 - **Fixed.** Arbitration is disabled and one autopilot is selected. In this mode **Autopilot 4x** will behave as an **Autopilot 1x**.
 - **Fixed 0.** Autopilot 1 is selected.
 - **Fixed 1.** Autopilot 2 is selected.
 - **Fixed 2.** Autopilot 3 is selected.
 - **Fixed while ok.** This mode does not take into account scores. In this mode, the **Preferred** autopilot will be selected by default. A switch will only happen if the current autopilot is considered **Dead**

3. **Holding CAP time.** Amount of time needed from last switch in order to allow a new switch.
4. **Hysteresis.** When comparing scores, the difference between them needs to be bigger than this proportional value, in order to assess scores. The difference is proportional to the score of the selected autopilot.
i.e. If current selected autopilot is the number 1, arbitration mode is **Always best**, hysteresis is **0.5** and score for AP 1 is **0.3**, AP 2 will need a score lower than **0.15** in order to be selected.
5. Enable arbitration of **external autopilot**.
6. **Variable - Max error.** Arbitration maximum error for each variable.
7. **Variable - Weight.** Arbitration weight for each variable.
8. **Variable - Absolute.** If it is enabled, it will indicate that it is an absolute variable. The value set here will be used to compare the variables in order to choose the best autopilot.

Click on  to add variables and  to delete them. Currently, the **maximum** amount of **arbitration variables** supported is **32**.

2.7.3 CAN Setup

This menu allows to configure the receiving CAN Ids for each of the 3 possible **Veronte Autopilots 1x** that are sending data to **MEX**. Therefore, to send data from any of them to MEX, these Ids must be specified.

Note: If arbitration is not enabled, only the configuration for Autopilot 1 will be used.

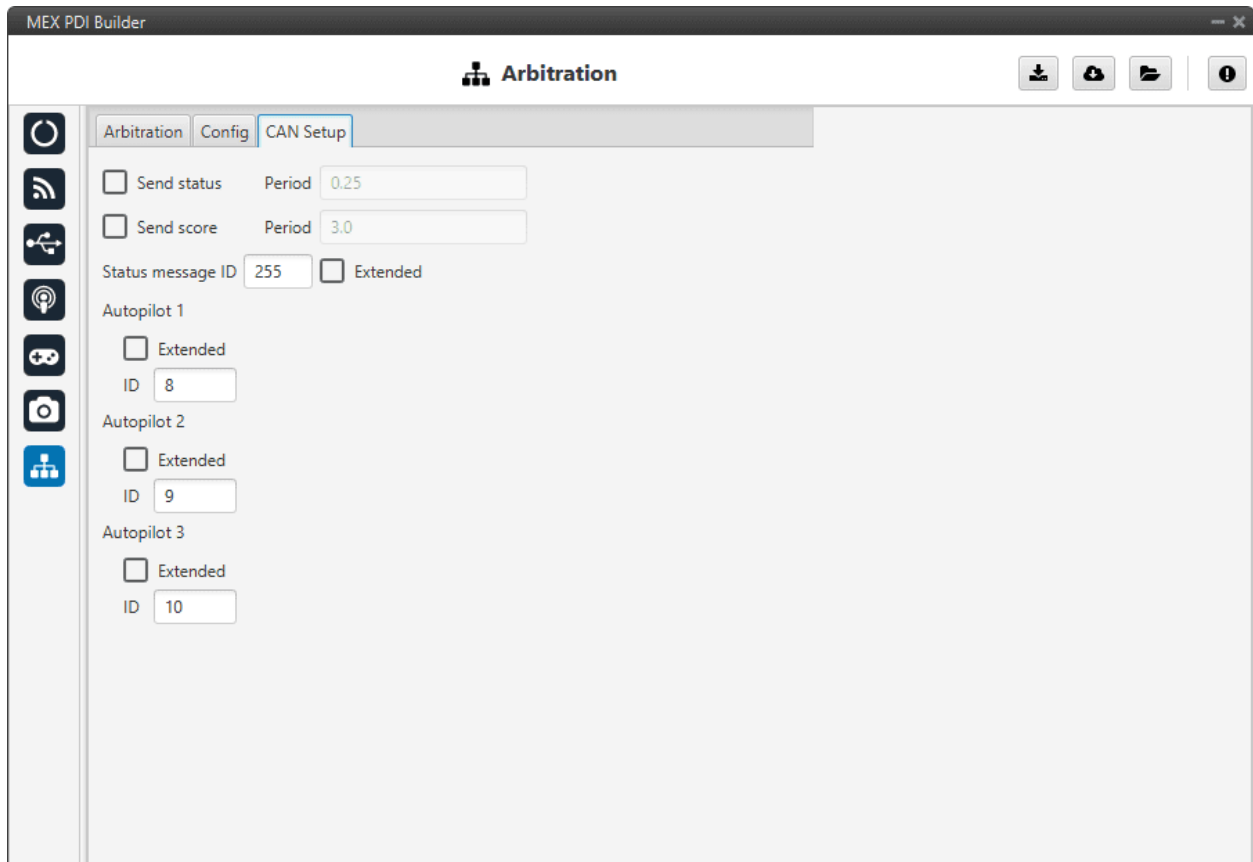


Fig. 59: CAN Setup section

- **Send status and Period:** The user can enable status message sending and set its period.
- **Send score and Period:** Users can enable the score message and define its period.
- **Status message ID:** CAN ID to which Status and Score messages are sent when there is **no External Arbitration** and therefore **MEX** works as **arbiter**.
 - **Extended:** If enabled, the frame format will be '**Extended**' (with a **29-bit identifier**). Otherwise, the frame format '**Standard**' (with a **11-bit identifier**) is set by default.
- **Autopilot 1-3:** CAN IDs used for the reception of autopilot 4x arbitration messages for each autopilot.
 - **Extended:** If enabled, the frame format will be '**Extended**' (with a **29-bit identifier**). Otherwise, the frame format '**Standard**' (with a **11-bit identifier**) is set by default.

INTEGRATION EXAMPLES

3.1 Connection with Autopilot 1x via CAN

No configuration is necessary for MEX, the configuration for communication with Autopilot 1x is set by default.

The configuration required in **1x PDI Builder** to communicate with MEX via CAN is explained in the Integration Examples section of the **1x PDI Builder** manual, click [here](#) to access it.

3.2 CAN Configuration

3.2.1 CAN Reception IDs

First of all, users can setup the receiving CAN Ids for each one of the three possible Veronte Autopilots 1x in the **Arbitration menu**.

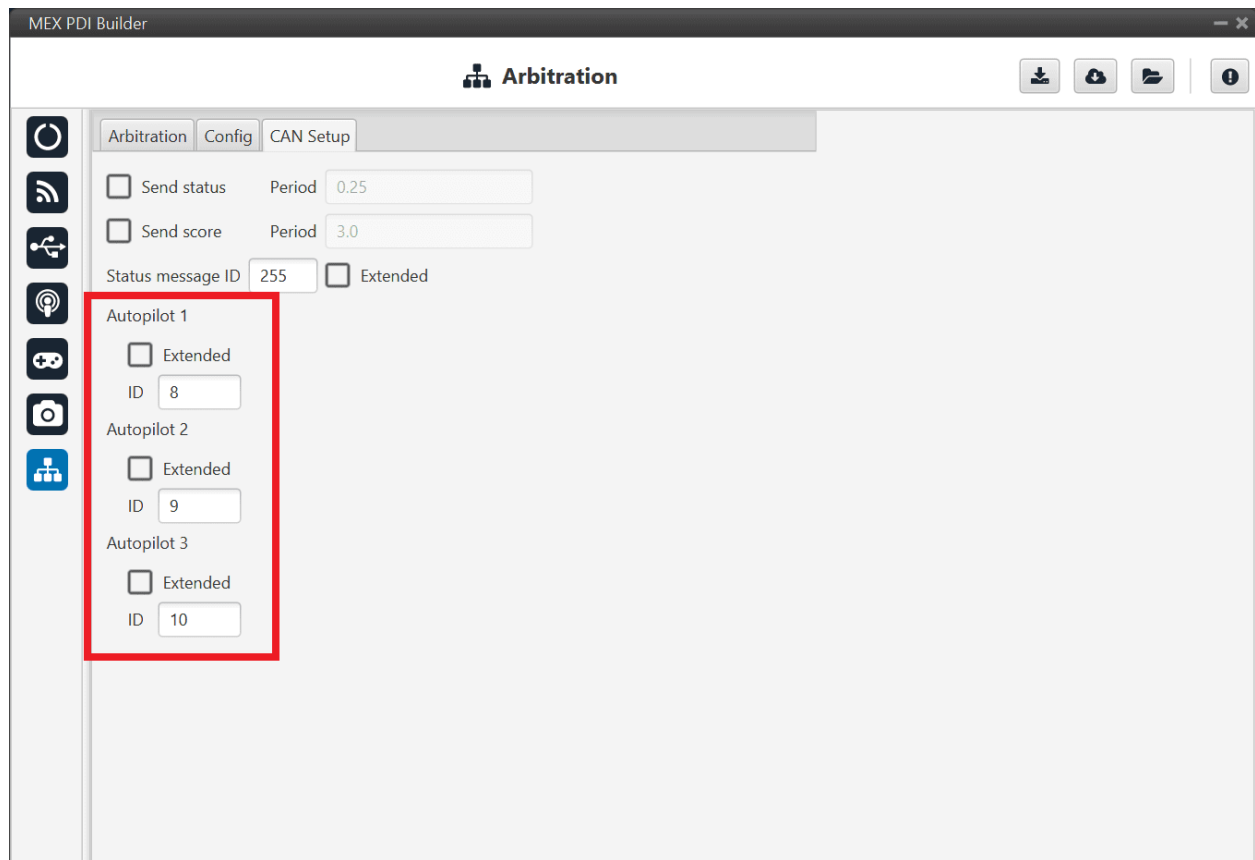


Fig. 1: Arbitration - CAN Ids of autopilots

Note: If **arbitration is not enabled**, and therefore only a 1x autopilot is being used, no CAN Ids need to be configured here.

3.2.2 CAN I/O Interconnections

Once CAN IDs are set, users shall configure:

- **Input Filters** to be used (as communication with MEX has to be **always** through a **Filter**).
- The connection between input filters and data Consumers.

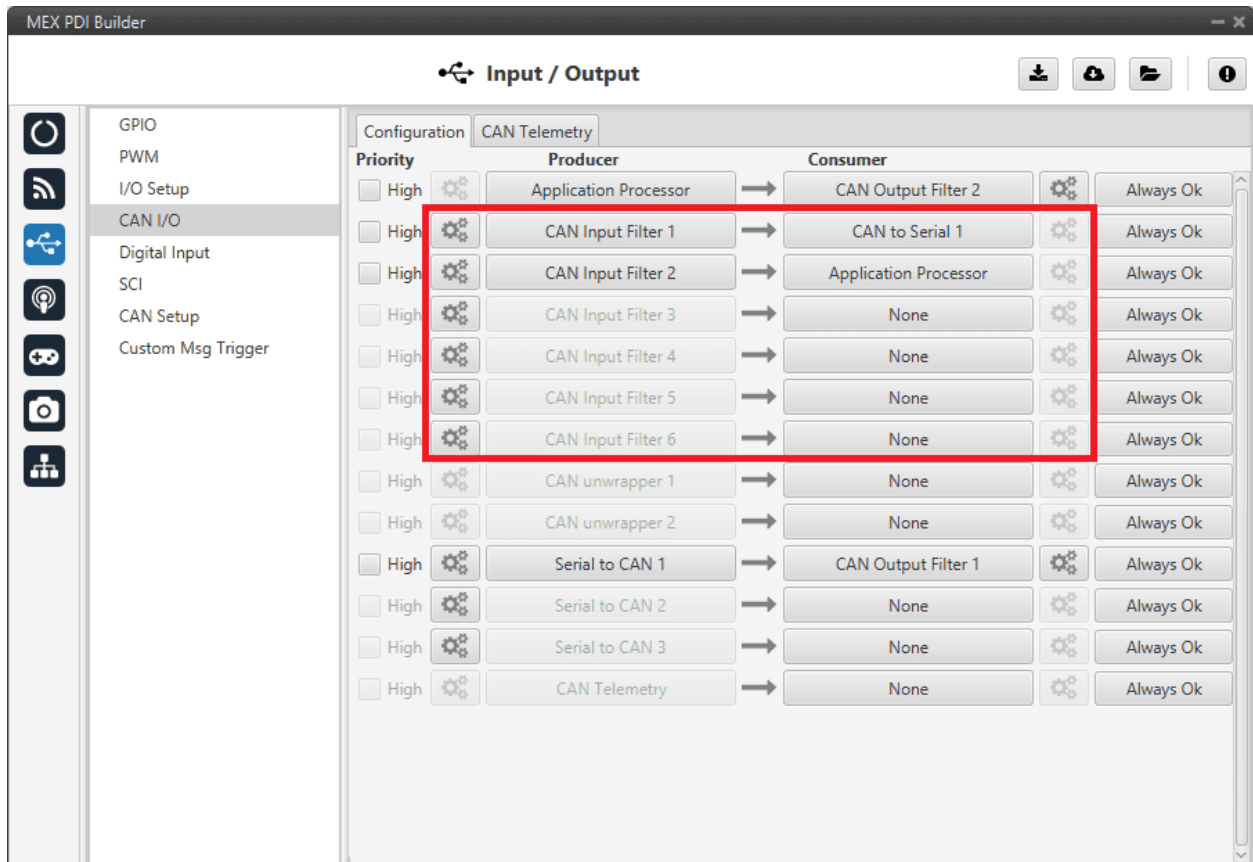


Fig. 2: CAN I/O interconnections

For more information on CAN I/O configuration, see section [CAN I/O](#) of this manual.

Next step is to connect each of the desired data Producers to an **Output Filter**, and configure both the Producer and the Output Filter:

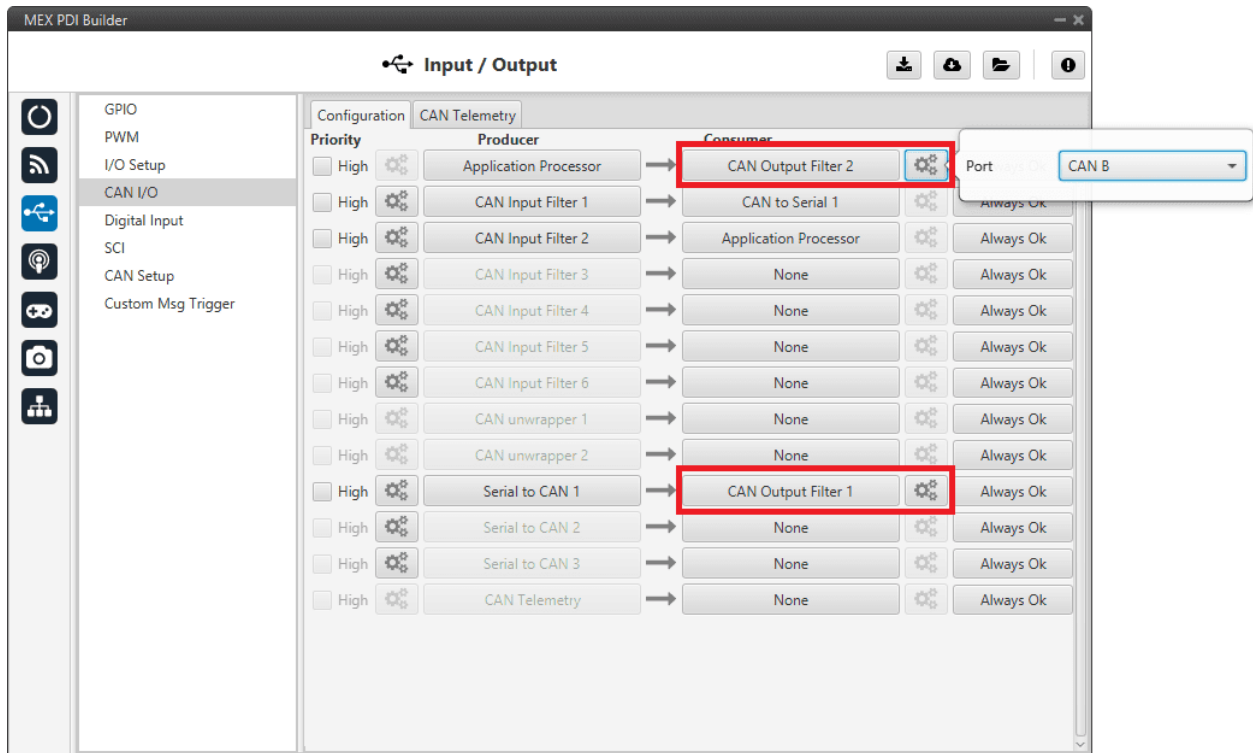


Fig. 3: CAN Output Filters

3.2.3 CAN Telemetry

In addition, MEX can send telemetry via CAN.

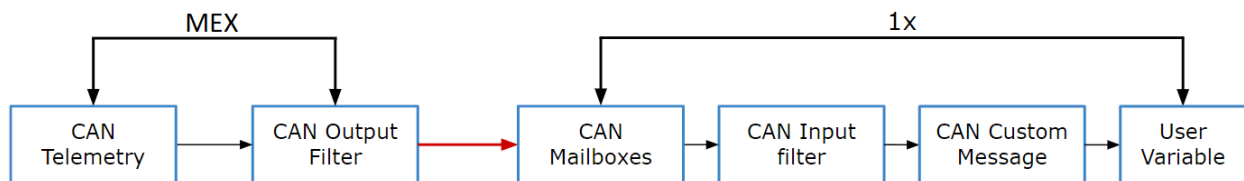


Fig. 4: Communication diagram 1x-MEX

The user must follow the steps below to enable this function and to configure an Autopilot 1x to read its telemetry:

MEX PDI Builder side

1. Go to Input/Output menu ⇒ CAN I/O section ⇒ **Configuration tab**.

Connect **CAN Telemetry** to a **CAN Output Filter** as follows:

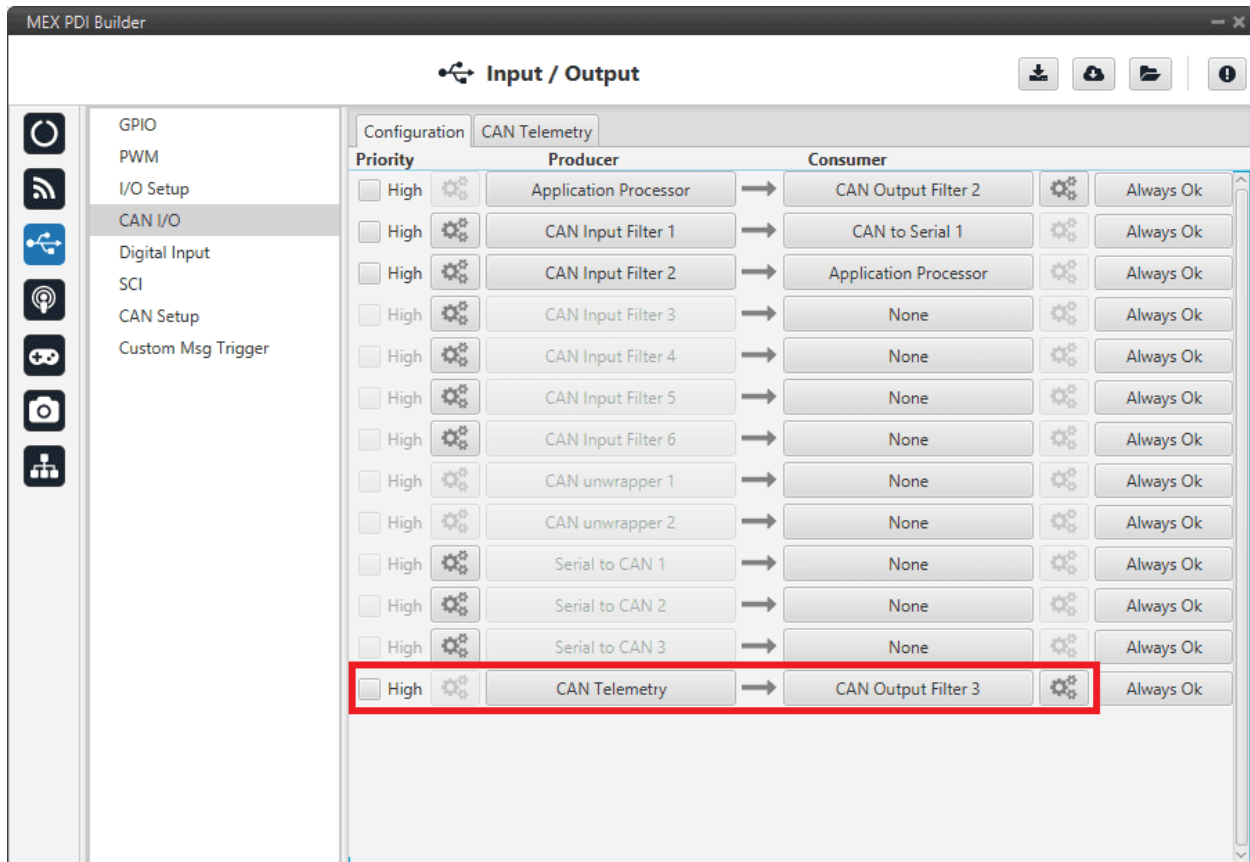


Fig. 5: CAN I/O Custom Telemetry

- Go to the **CAN Telemetry** tab.

Select the fields to send in the **TX section**, as it is a Producer. More information about the configuration of telemetry messages can be found in the [CAN Telemetry section](#).

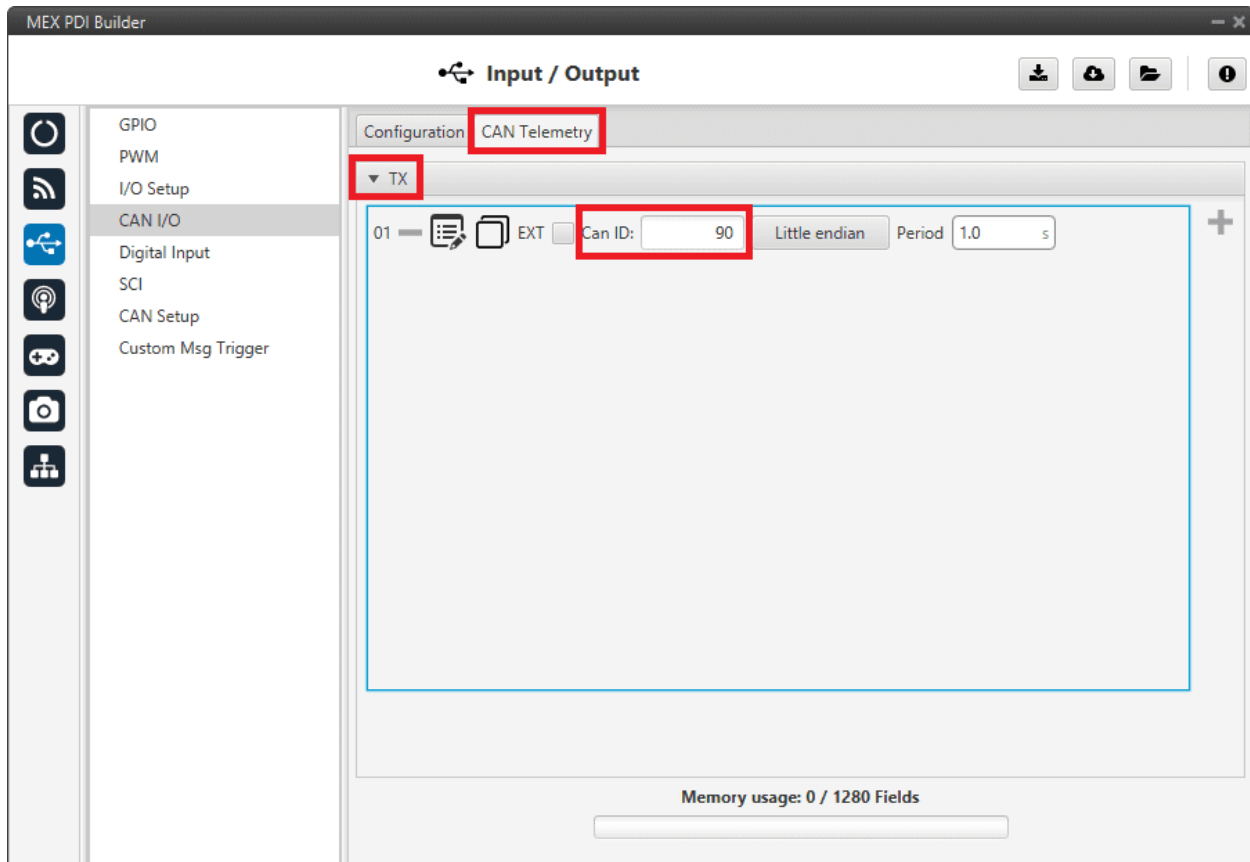


Fig. 6: CAN Custom Telemetry

For example, a telemetry message with a variable set to ID 90:

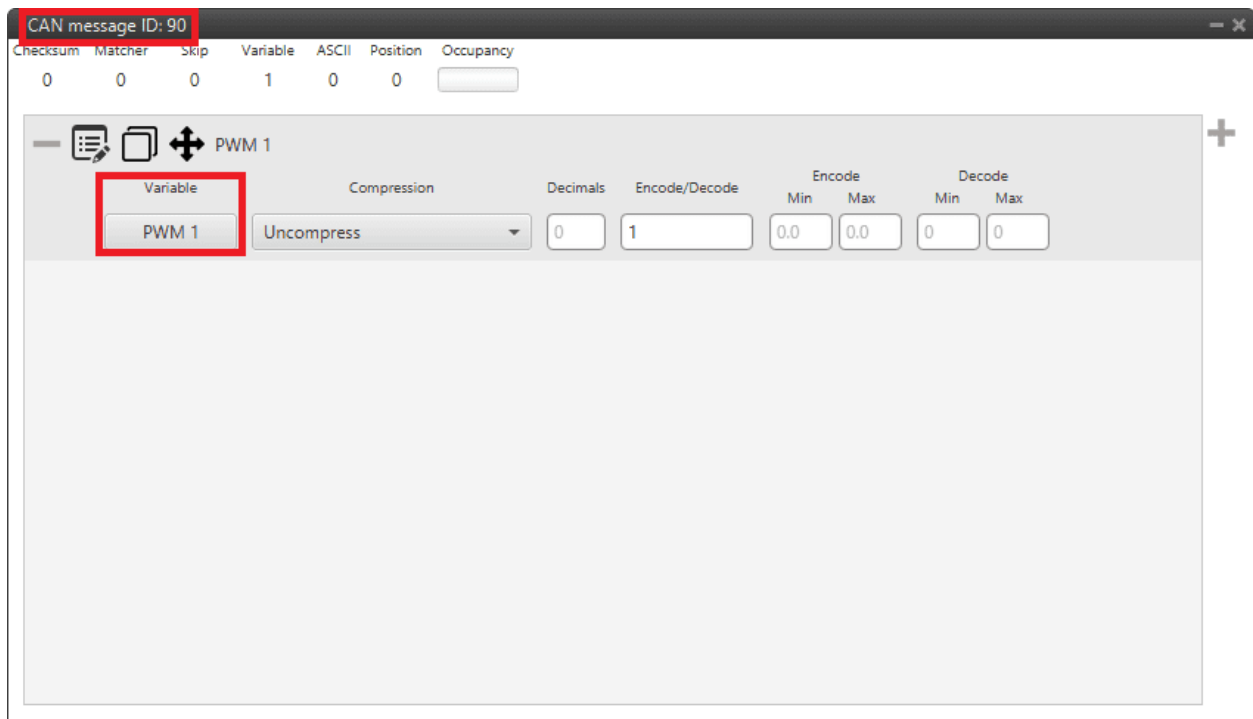


Fig. 7: CAN Custom Telemetry example

1x PDI Builder side

3. Go to UI menu ⇒ Variables section ⇒ **Reals Vars tab**.

Rename a User Variable, that will be used to store the value received from MEX:

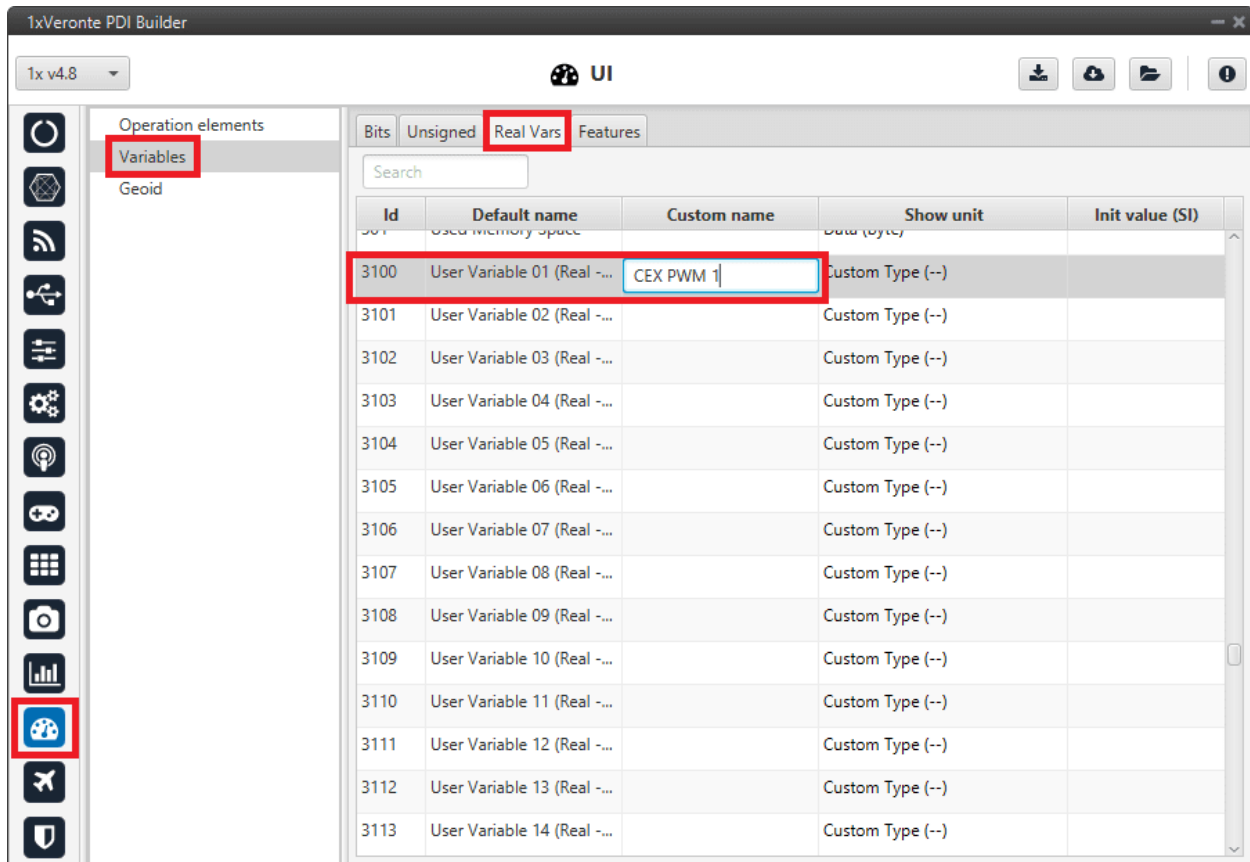


Fig. 8: 1x PDI Builder - User Variable renamed

4. Go to Input/Output menu ⇒ CAN Setup section ⇒ **Mailboxes tab**.

Configure the mailbox to receive the message with ID 90 in **1x PDI Builder**:

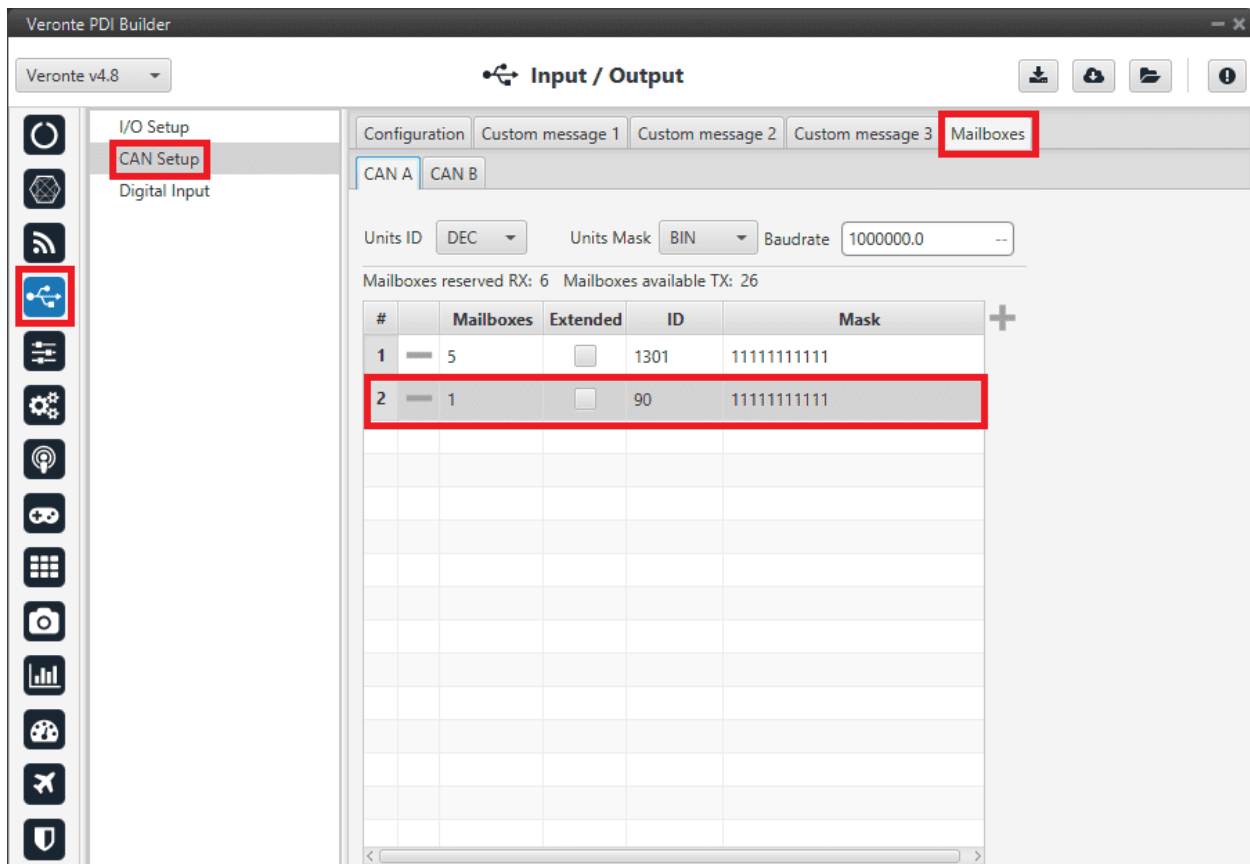


Fig. 9: 1x PDI Builder - Mailbox configuration

5. Go to Input/Output menu ⇒ CAN Setup section ⇒ **Configuration tab**.

Connect an **Input filter** with the **right CAN ID** to a **Custom message consumer**:

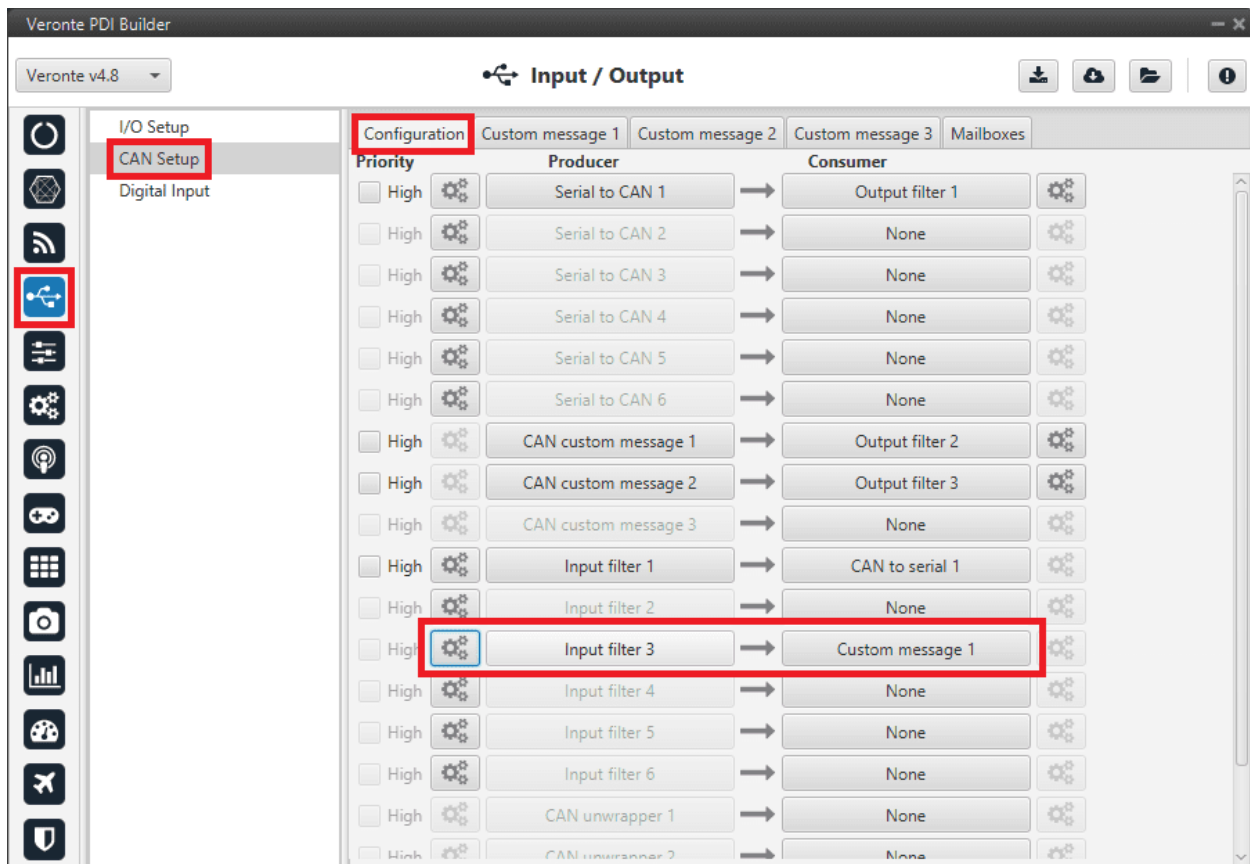


Fig. 10: 1x PDI Builder - Input filter

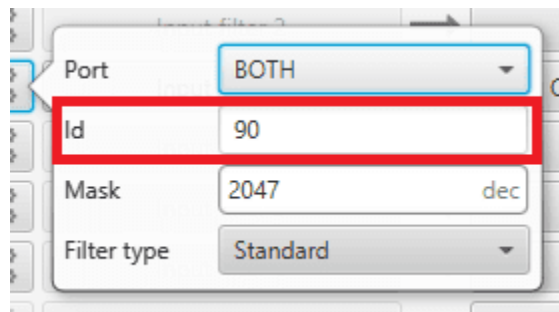


Fig. 11: 1x PDI Builder - Input filter configuration

- Go to Input/Output menu ⇒ CAN Setup section ⇒ **Custom message 1 tab** (as Custom Message 1 has been selected as consumer).

Configure the reading of the message and store the received value in the user variable renamed above:

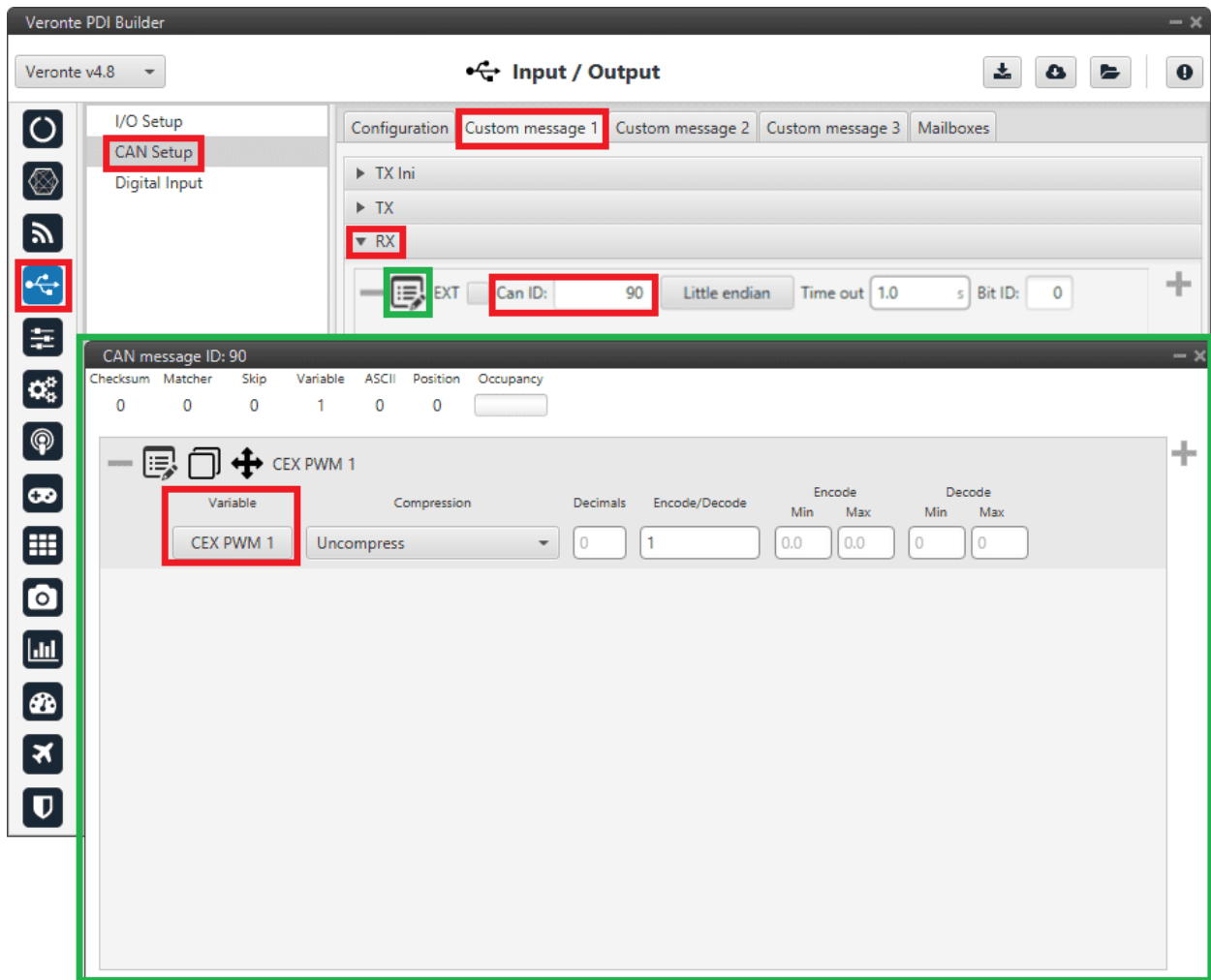


Fig. 12: 1x PDI Builder - Custom Message configuration

3.3 Commanding/Reading PWMs

The appropriate PWM message format is described in [Command PWMs -> CAN Bus protocol](#) section of the **MEX Software manual**.

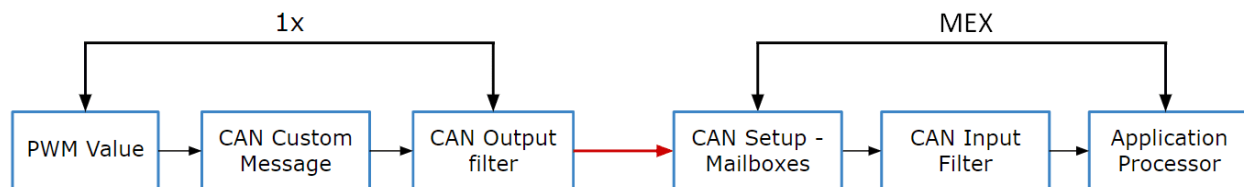


Fig. 13: Communication diagram 1x-MEX

The following steps command PWM from Autopilot 1x to MEX:

1x PDI Builder side

1. Go to Input/Output menu ⇒ **CAN Setup section** ⇒ **Configuration tab**.

Connect a CAN custom message to an Output filter to send the message:

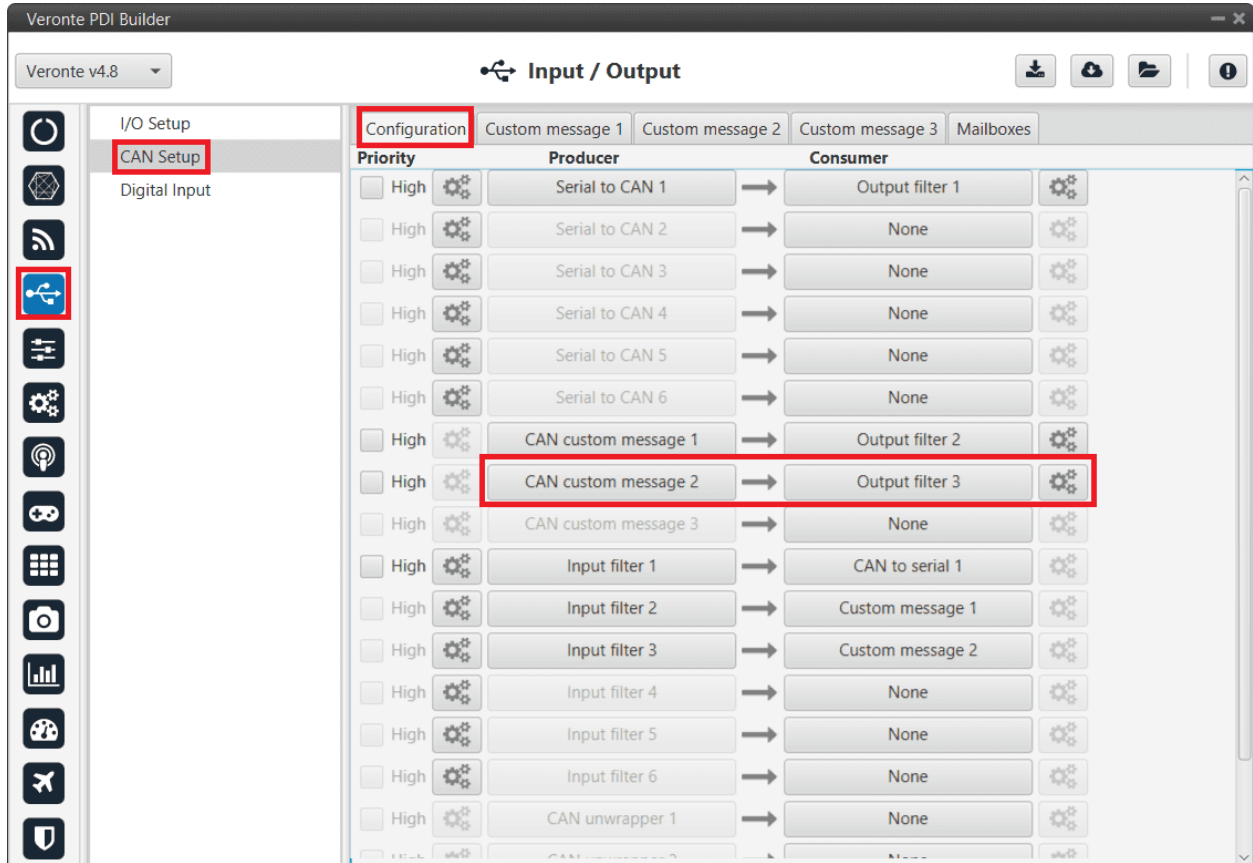


Fig. 14: PWM Command - CAN Custom Message

2. Go to Input/Output menu ⇒ CAN Setup section ⇒ **Custom Message 2 tab** (because Custom Message 2 has been selected as consumer).

Build a CAN custom message using the PWM variable and the appropriate Message format. CAN ID is arbitrary, for this example we will use ID 100:

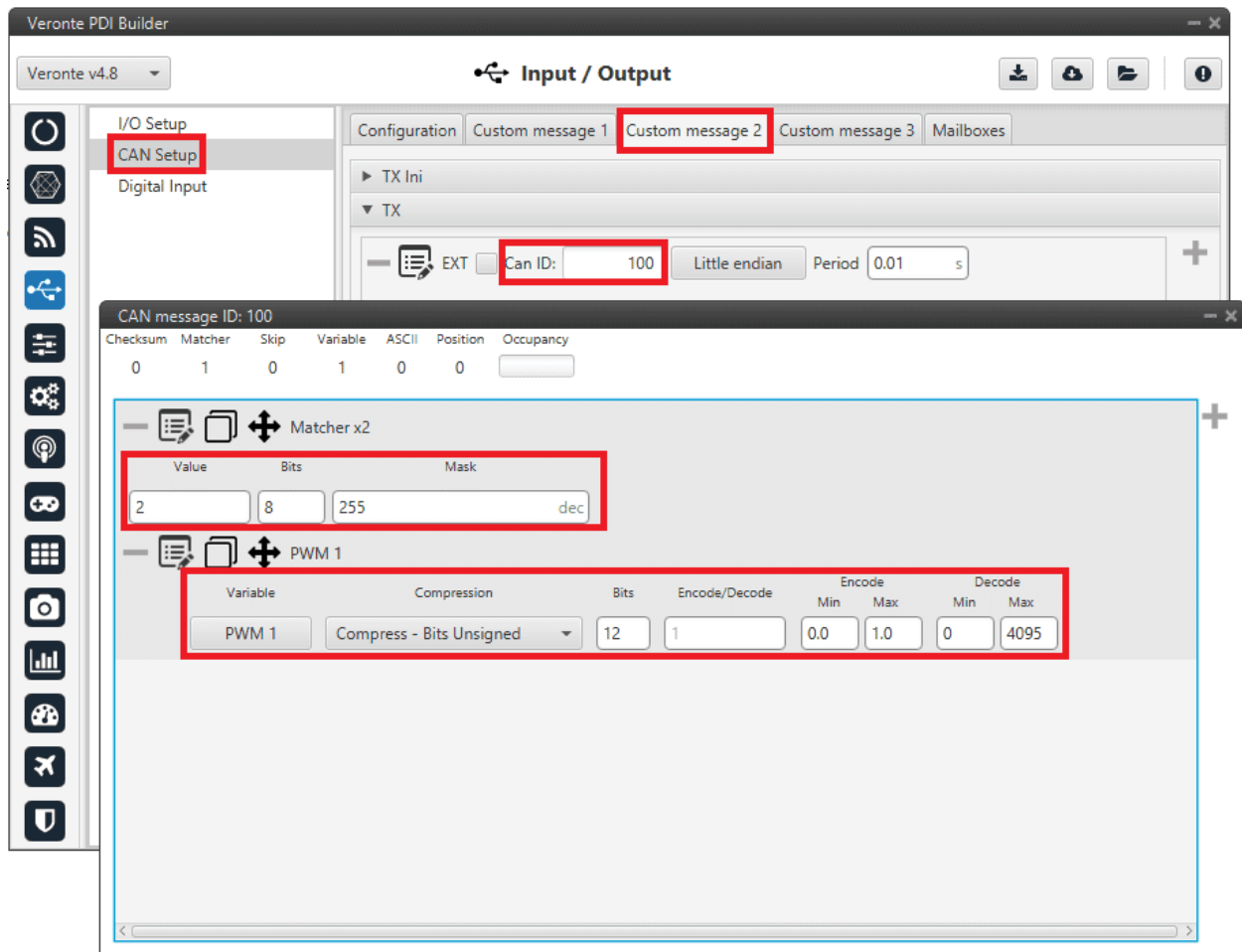


Fig. 15: PWM Command - CAN Custom Message configuration

Message format:

- Matcher (2) [8 bits]
 - PWM 1 [12 bits]
 - PWM 2 [12 bits]
 - PWM 3 [12 bits]
 - PWM 4 [12 bits]
- Matcher (3) [8 bits]
 - PWM 5 [12 bits]
 - PWM 6 [12 bits]
 - PWM 7 [12 bits]
 - PWM 8 [12 bits]
- Each PWM variable has to be set using the following format:
 - Variable: PWM X
 - **Compression:** Compress - Bits Unsigned

- **Bits:** 12
- **Encode:** Min=0.0 / Max=1.0
- **Decode:** Min=0 / Max=4095

MEX PDI Builder side

3. Go to Input/Output menu ⇒ **CAN Setup section.**

Create the mailbox to receive the new message (ID 100):

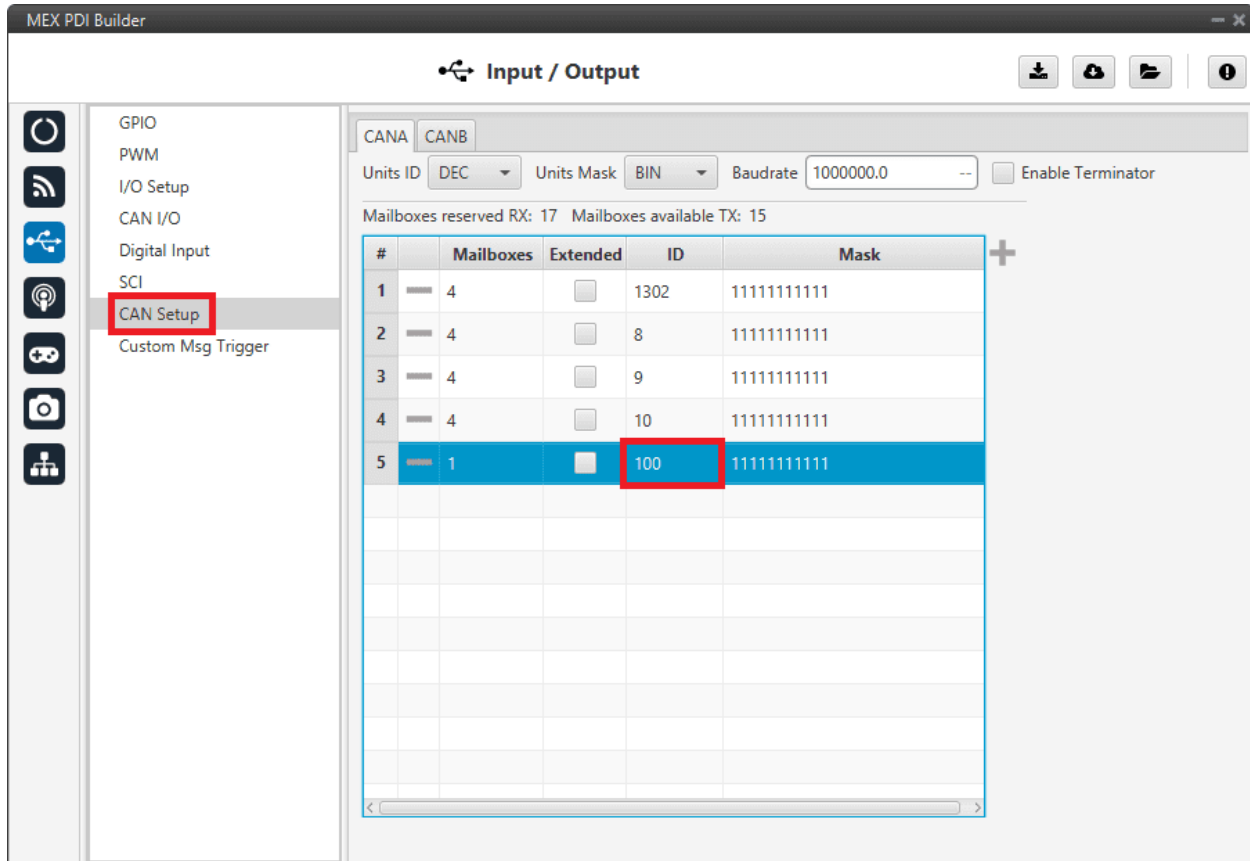


Fig. 16: PWM Command- Mailbox configuration

4. Go to Input/Output menu ⇒ CAN I/O section ⇒ **Configuration tab.**

Connect an **Input filter** with the **right CAN ID** to the **Application Processor consumer**:

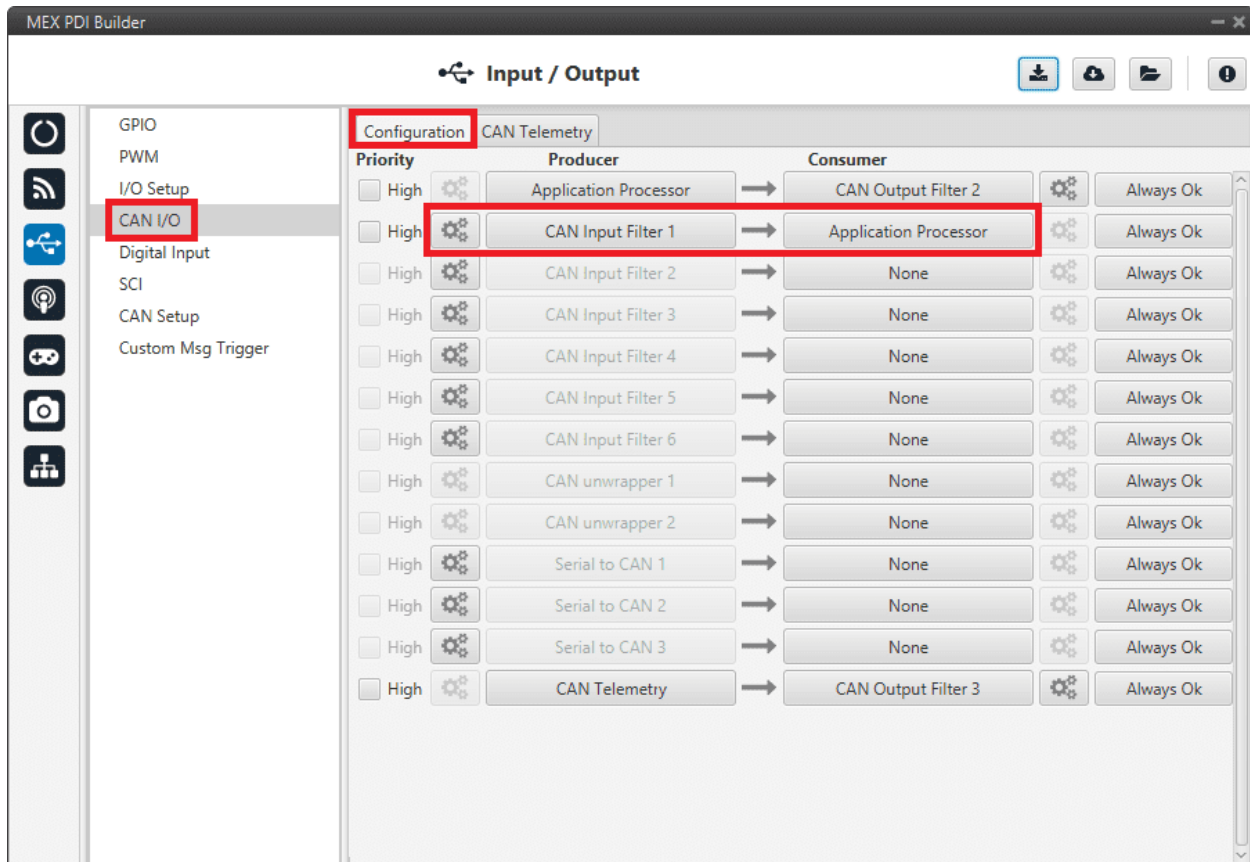


Fig. 17: PWM Command - Input filter

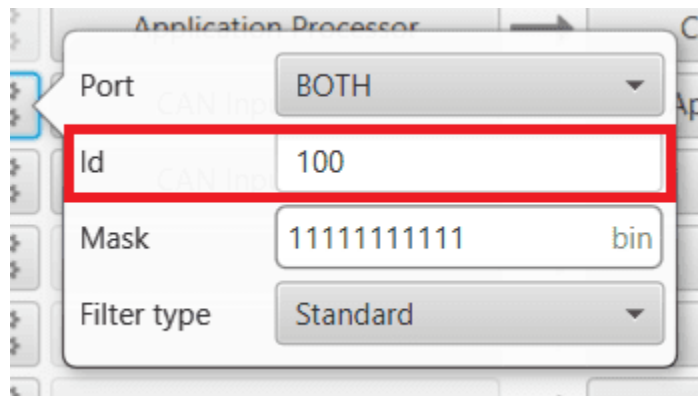


Fig. 18: PWM Command - Input filter configuration

- Go to Input/Output menu ⇒ PWM section ⇒ **PWM 1 tab** (as PWM 1 has been selected in **1x PDI Builder**).

Configure parameters for PWM 1:

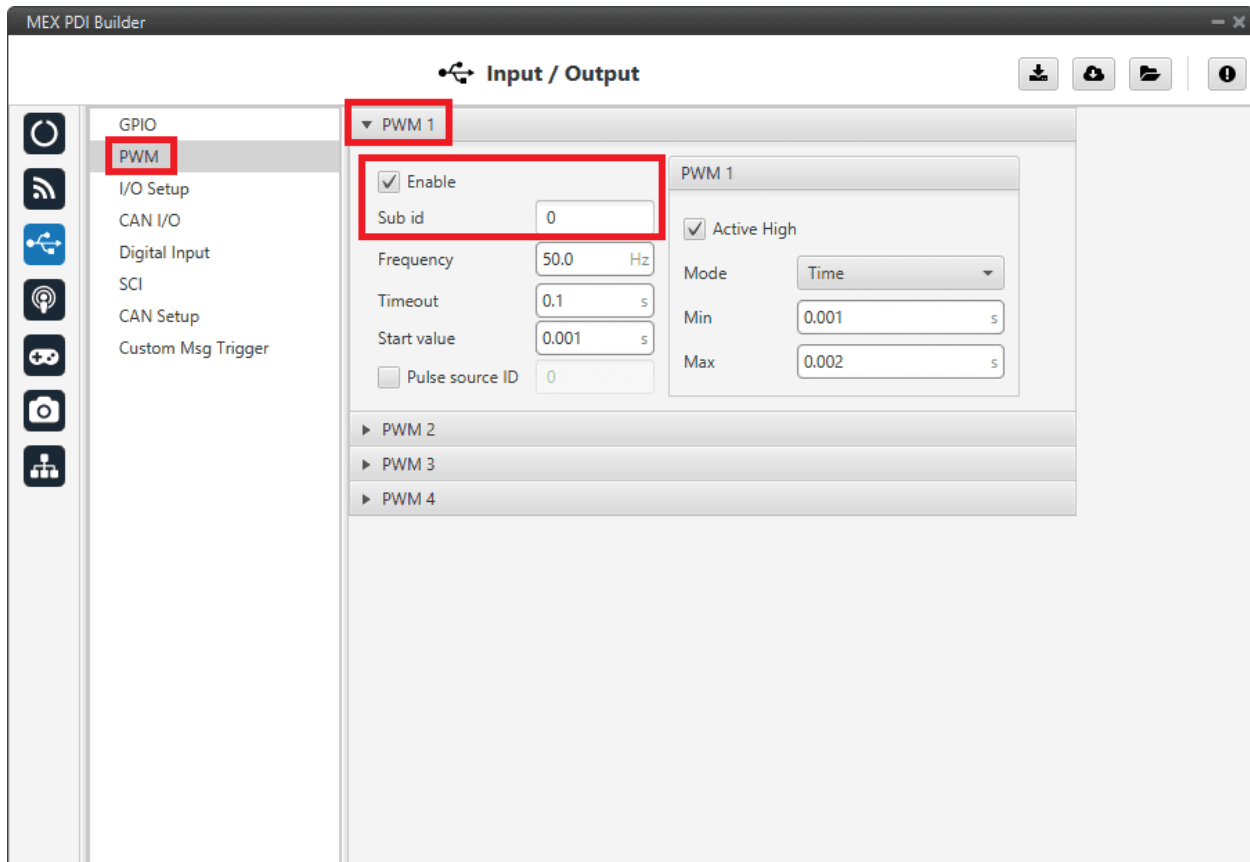


Fig. 19: PWM Command - PWM configuration

6. Go to Input/Output menu ⇒ **GPIO section**.

Check that this PWM pin (**PWM 1** in this case) has correctly switched from GPIO to PWM.

It should look like this:

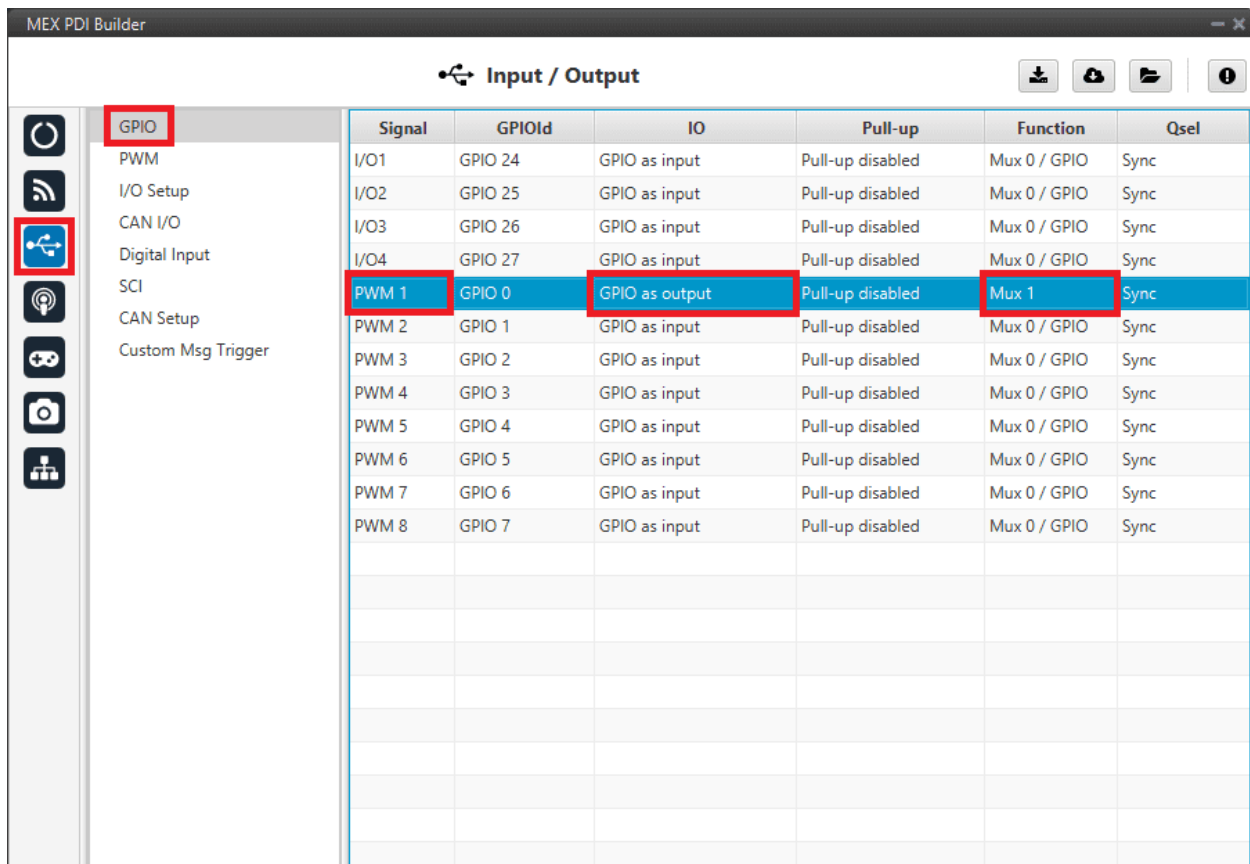


Fig. 20: PWM Command - GPIO configuration

- **IO:** GPIO as **output**.
- **Function:** Mux **1**.

7. Finally, export the configuration and save changes.

3.4 GPIO Command

The following are the steps to send a GPIO command from the Veronte Autopilot 1x, receive it at MEX and process it, so that MEX carries out the command.

GPIO Command is very similar to *PWM command* with a few differences.

1x PDI Builder side

1. Go to Input/Output menu ⇒ CAN Setup section ⇒ **Configuration tab**.

Connect, as Producer, a 'CAN GPIO remote' to an **Output filter**:

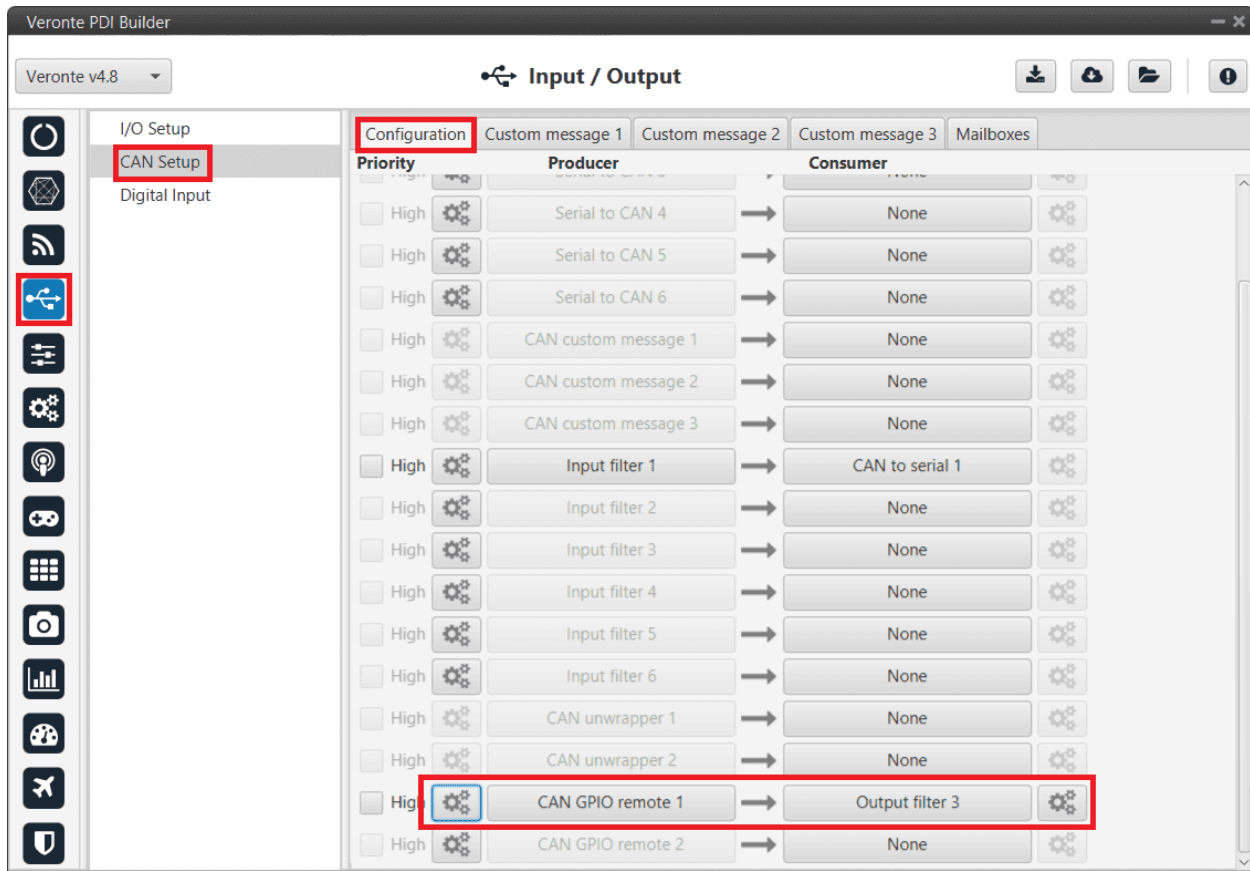


Fig. 21: GPIO Command - 1x CAN Setup

CAN GPIO remote must be configured. For more information about its configuration, read [CAN Setup](#) section of [1x PDI Builder](#) user manual.

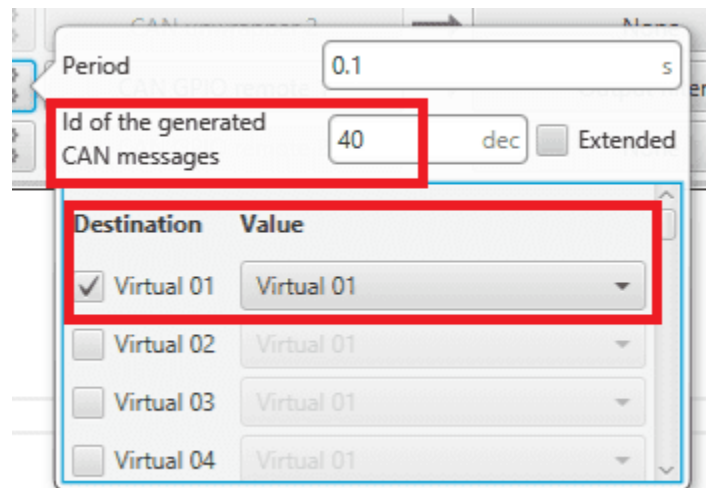


Fig. 22: GPIO Command - 1x CAN Setup configuration

2. Go to **Automations** menu.

GPIO must be activated using an ‘Output action’:

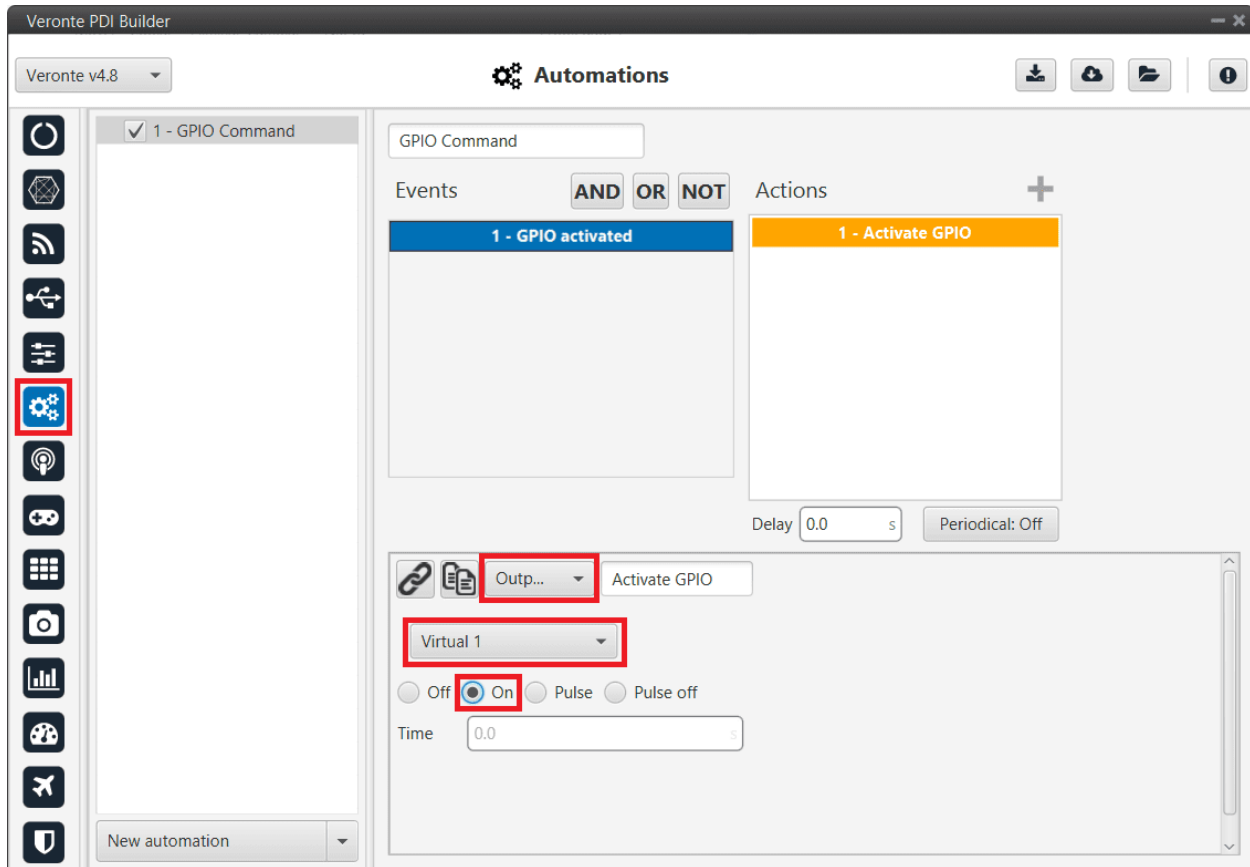


Fig. 23: GPIO Command - 1x Automation configuration

MEX PDI Builder side

- Go to Input/Output menu ⇒ CAN I/O section ⇒ **Configuration tab**.

Finally, connect an **Input filter** to the **CAN GPIO consumer**:

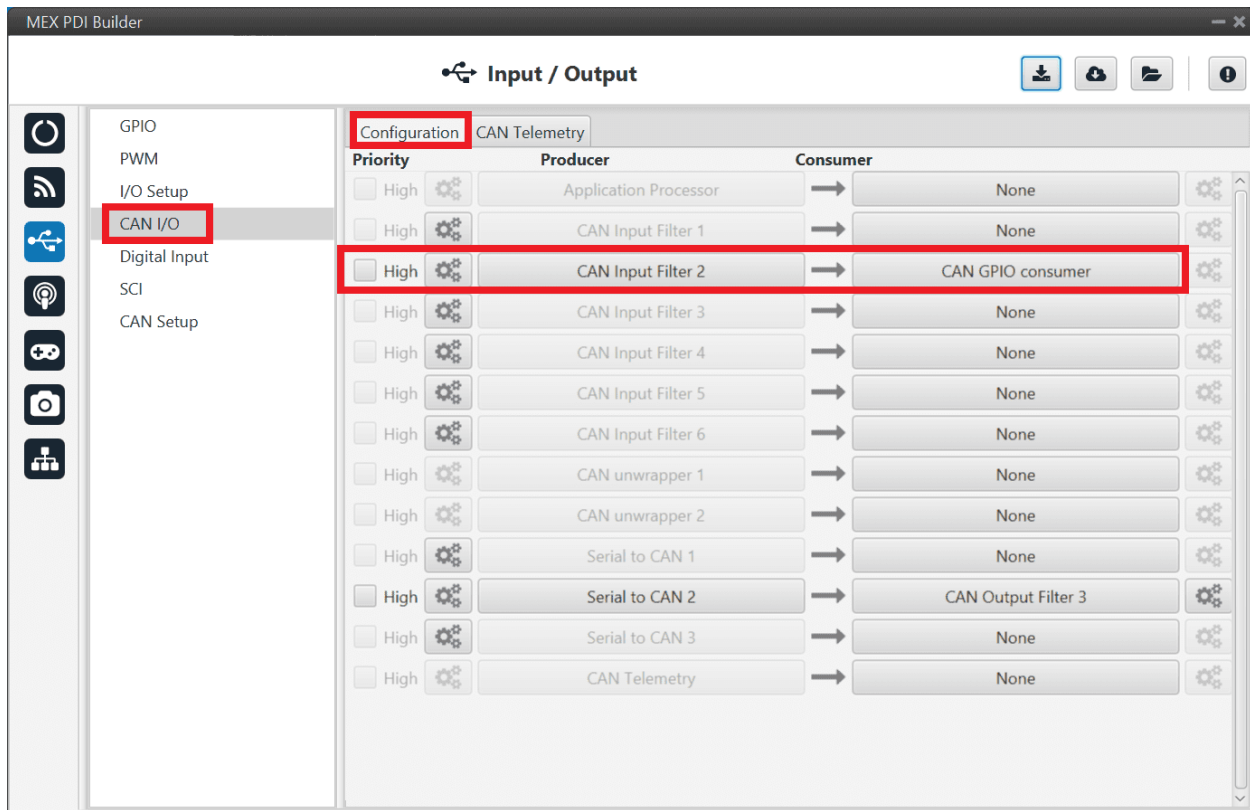


Fig. 24: GPIO Command - CAN I/O

The Id must match the one configured in 1x PDI Builder **Output filter**:

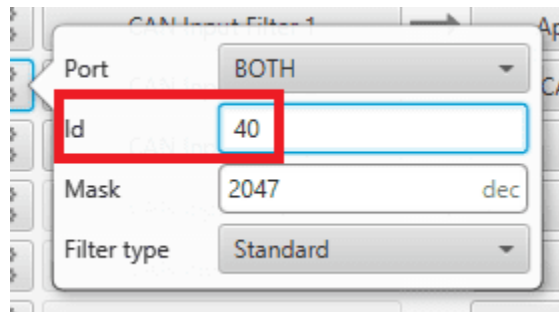


Fig. 25: GPIO Command - Input filter configuration

3.5 Jetibox

MEX is able to simulate a Jetibox to read telemetry from Legacy Jeti devices, the following steps need to be taken:

1. Go to Devices menu ⇒ **Jetibox section**.

Enable it and select a **SCI Port**, in this example SCI A is selected:

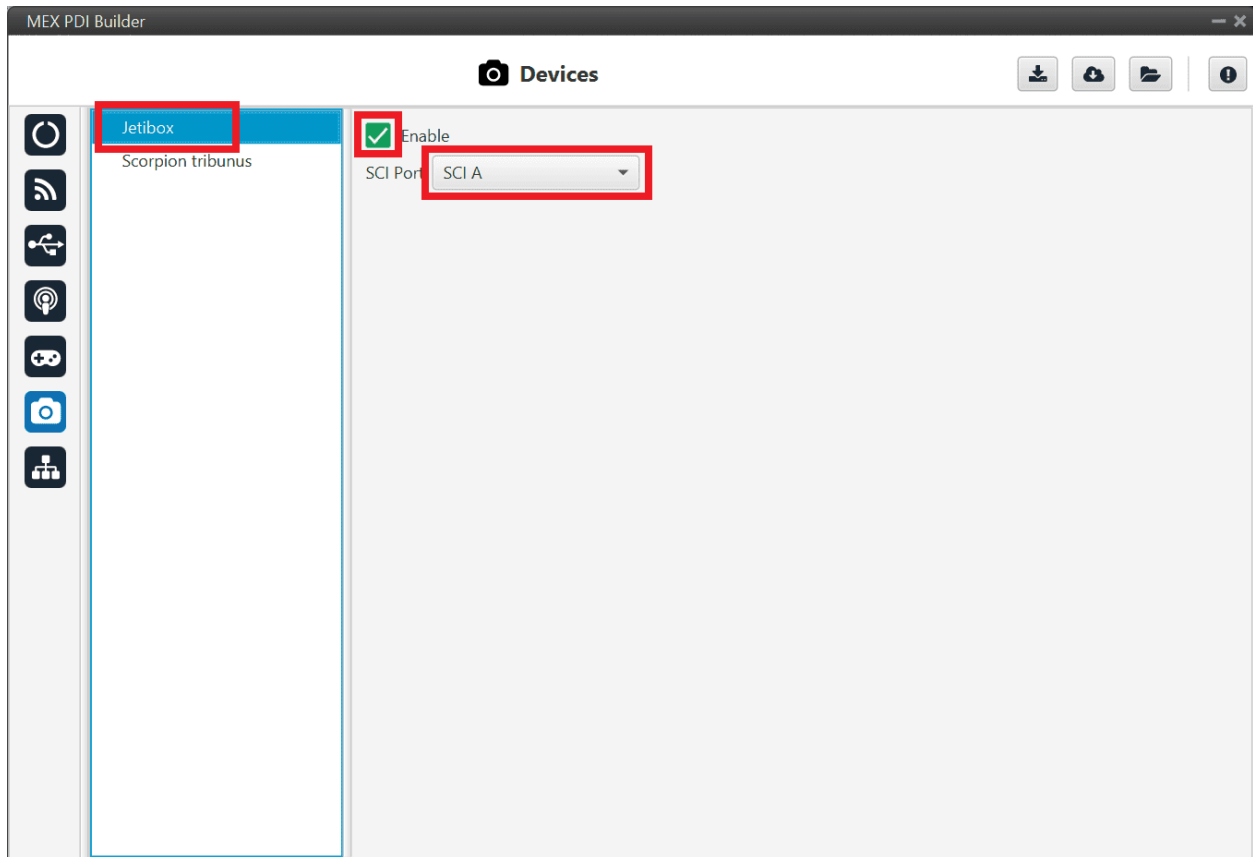


Fig. 26: Jeti box Devices configuration

2. Go to Input/Output menu ⇒ **SCI section**.

Configure the serial port (the one selected in step 1) as follows:

- **Baudrate:** 9800
- **Length:** 8
- **Stop:** 2 stop bits
- **Parity:** Odd
- **Use address mode:** Enabled

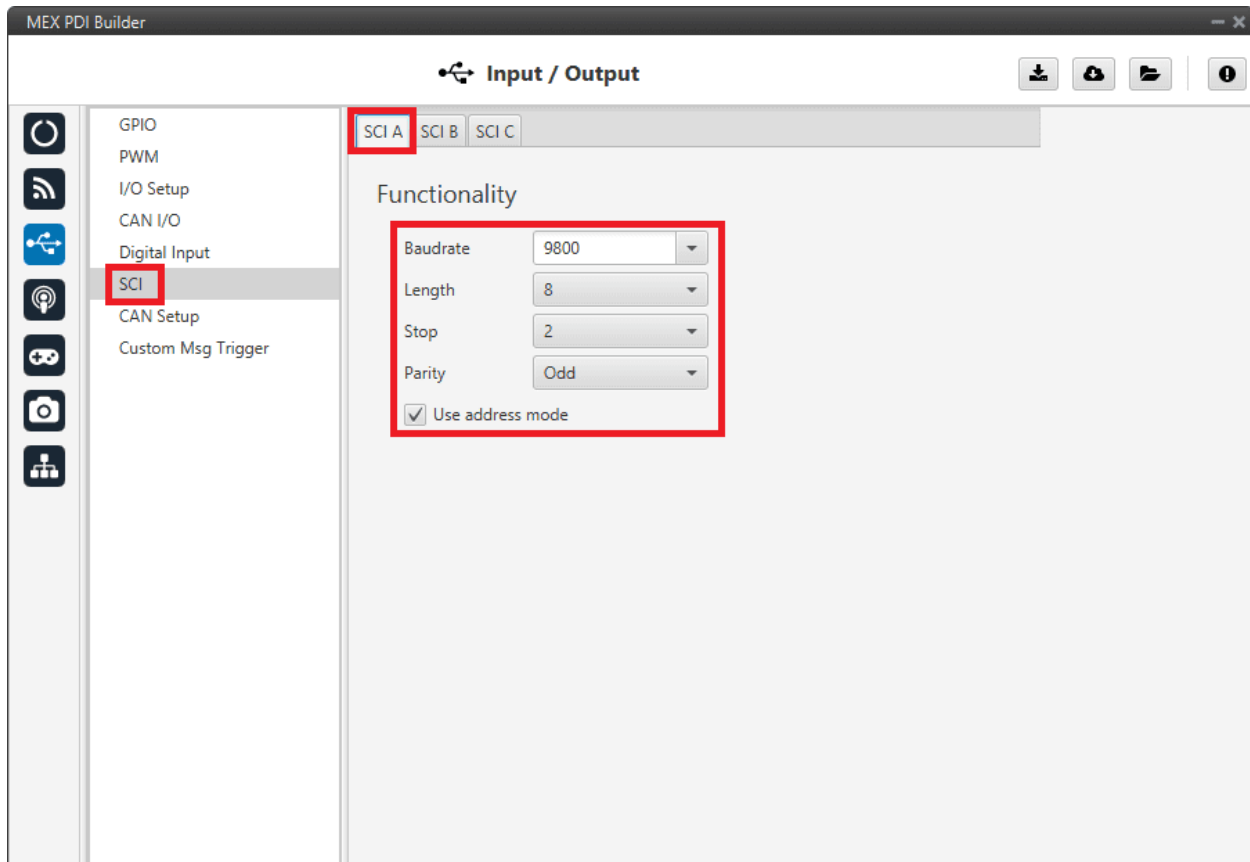


Fig. 27: Jeti box SCI configuration

Note: The serial port will be totally reserved for this, so it will not be usable to other things and the I/O Setup affecting it will be ignored.

3. Go to Input/Output menu ⇒ **I/O Setup section.**

Link the specific Jetibox I/O consumer to a port, that will be configured in step 4:

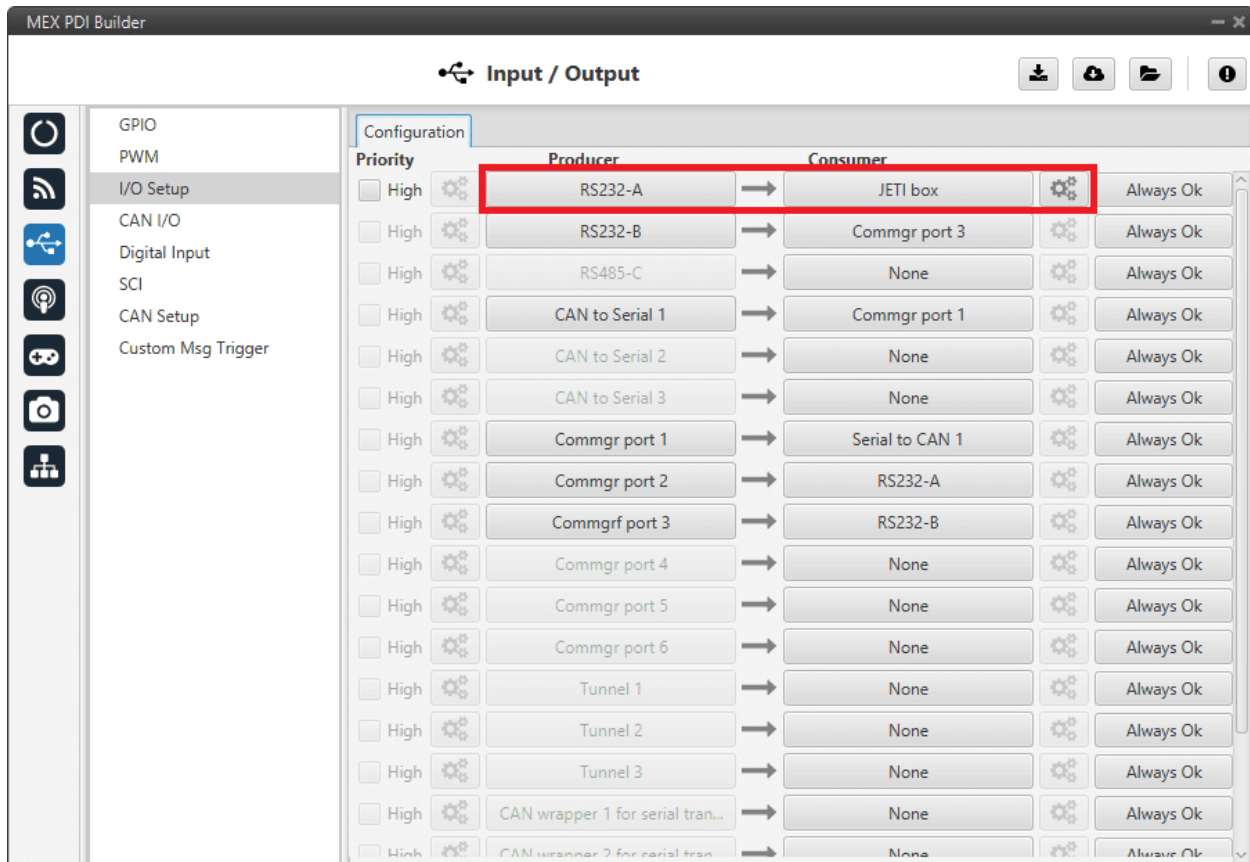



Fig. 28: Jetibox Consumer

4. Configure the Jetibox I/O consumer to retrieve the data. Click on the **configuration button** ( icon):

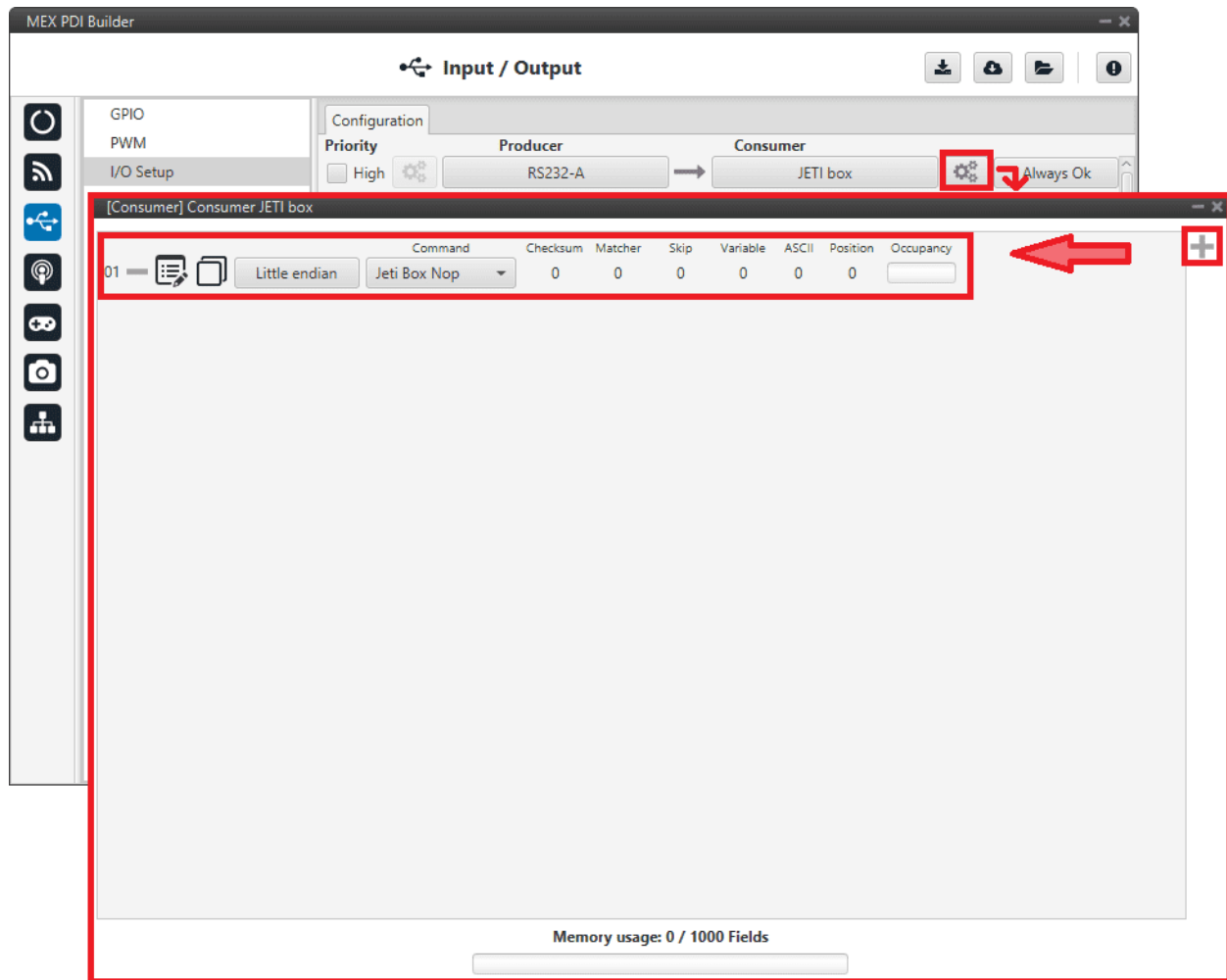


Fig. 29: Jetibox Consumer configuration

Below is an example of the custom messages needed to read the Actual Voltage from a Jeti MasterSpin 220 (use **Big endian** in all messages):

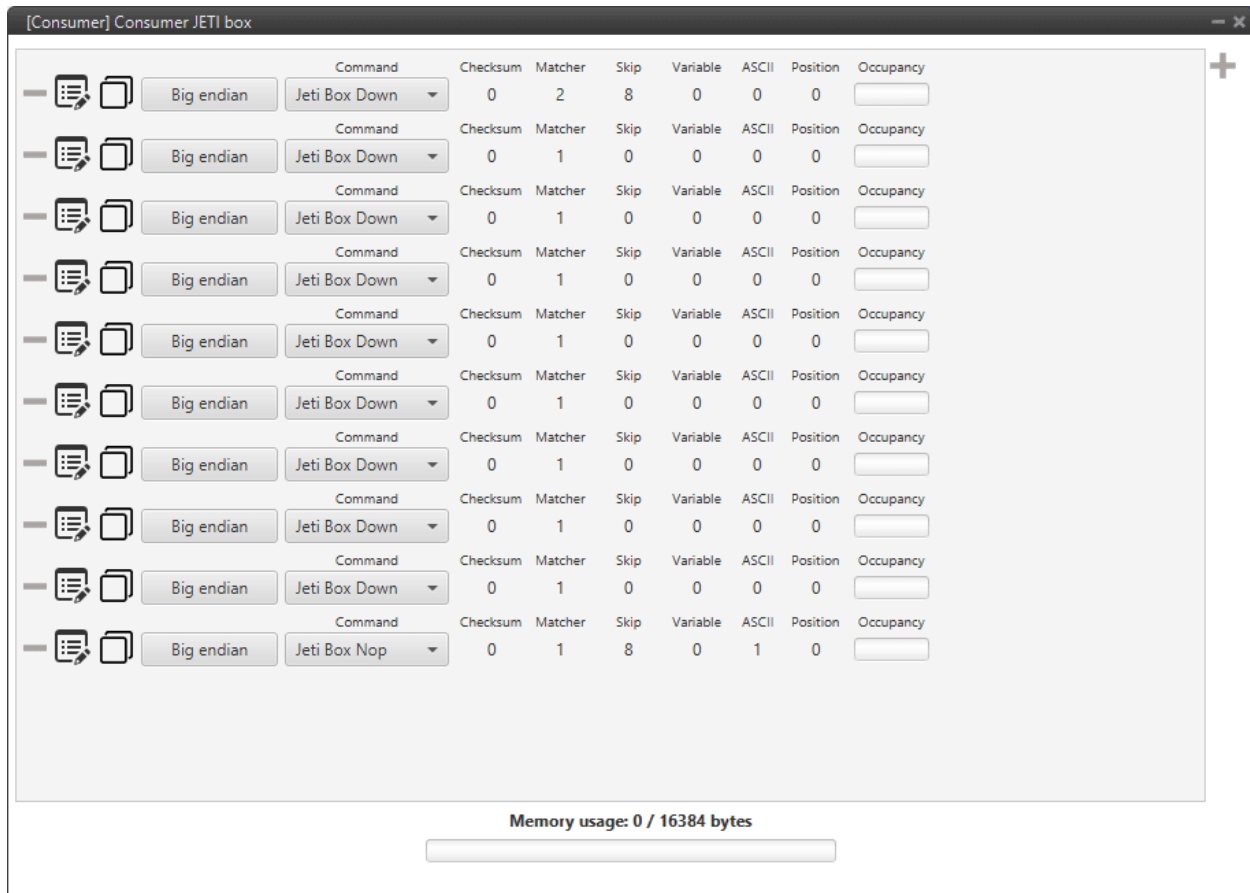


Fig. 30: JETI box example

- Expected text: "CONTROLLER TYPE MasterSpin 220~"
 - Command: **Jeti box Down**
 - Matcher(32) "CONT" 0x434F4E54 (1129270868)
 - Skip(24*8) 192
 - Matcher(32) "220~" 0x3232307E (842150014)

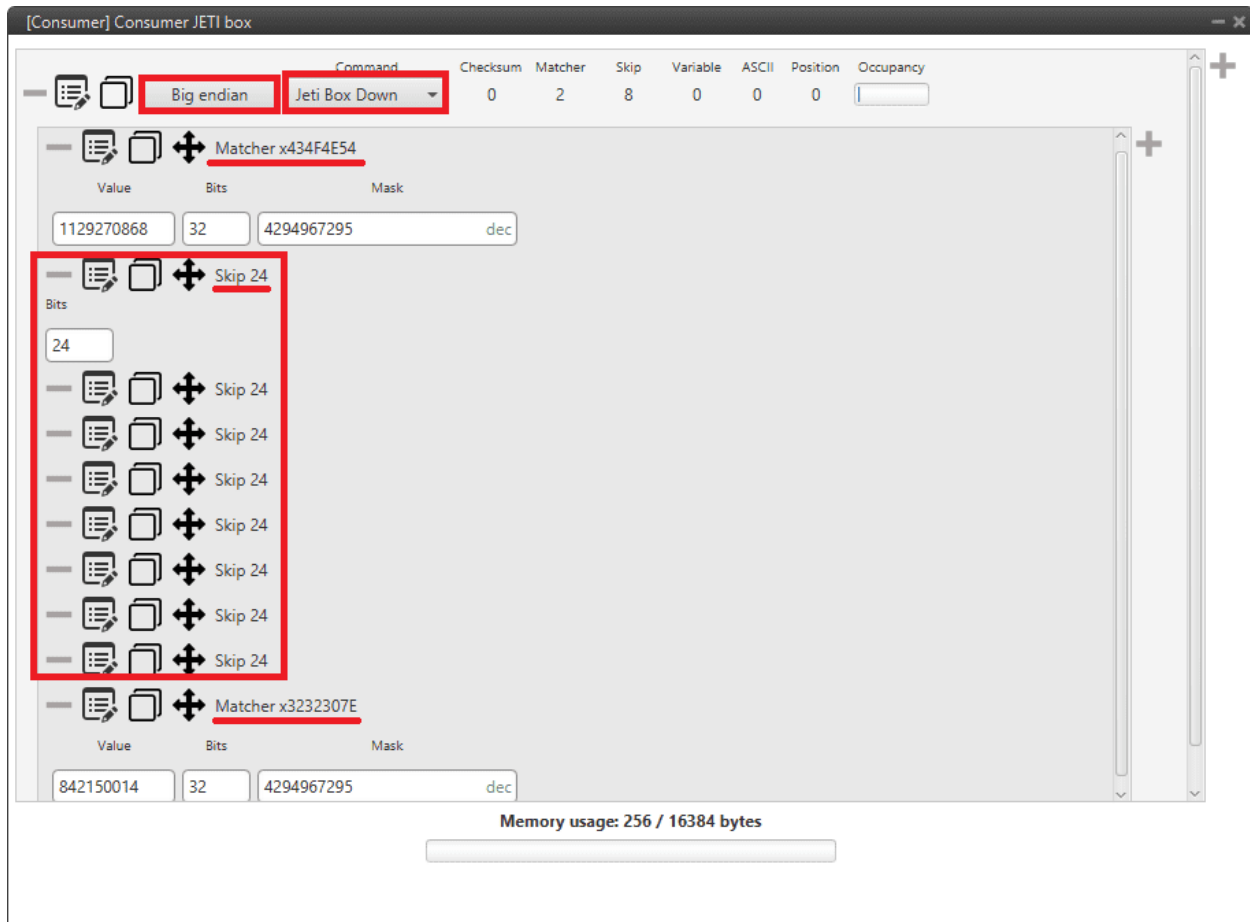


Fig. 31: JETI box custom message example

2. Expected text: "MeasureOrSetting MEASURE ~"
 - Command: **Jeti box Down**
 - Matcher(32) "Meas" 0x4D656173 (1298489715)
3. Expected text: "Max Temperature"...
 - Command: **Jeti box Down**
 - Matcher(32) "Max " 0x4D617820 (1298233376)
4. Expected text: "Min Temperature"...
 - Command: **Jeti box Down**
 - Matcher(32) "Min " 0x4D696E20 (1298755104)
5. Expected text: "Actual Temperature"...
 - Command: **Jeti box Down**
 - Matcher(32) "Actu" 0x41637475 (1097036917)
6. Expected text: "MaxCurrent"...
 - Command: **Jeti box Down**

- Matcher(32) “MaxC” 0x4D617843 (1298233411)
- 7. Expected text: “MinCurrent”...
 - Command: **Jeti box Down**
 - Matcher(32) “MinC” 0x4D696E43 (1298755139)
- 8. Expected text: “Max Voltage”...
 - Command: **Jeti box Down**
 - Matcher(32) “Max ” 0x4D617820 (1298233376)
- 9. Expected text: “Min Voltage”...
 - Command: **Jeti box Down**
 - Matcher(32) “Min ” 0x4D696E20 (1298755104)
- 10. Expected text: “Actual Voltage 11,86 V “
 - Command: **Jeti box Nop**
 - Matcher(32) “Actu” 0x41637475 (1097036917)
 - Skip(12*8) 96
 - Parse ascii: int(2), decimal(2), separator(‘,’)

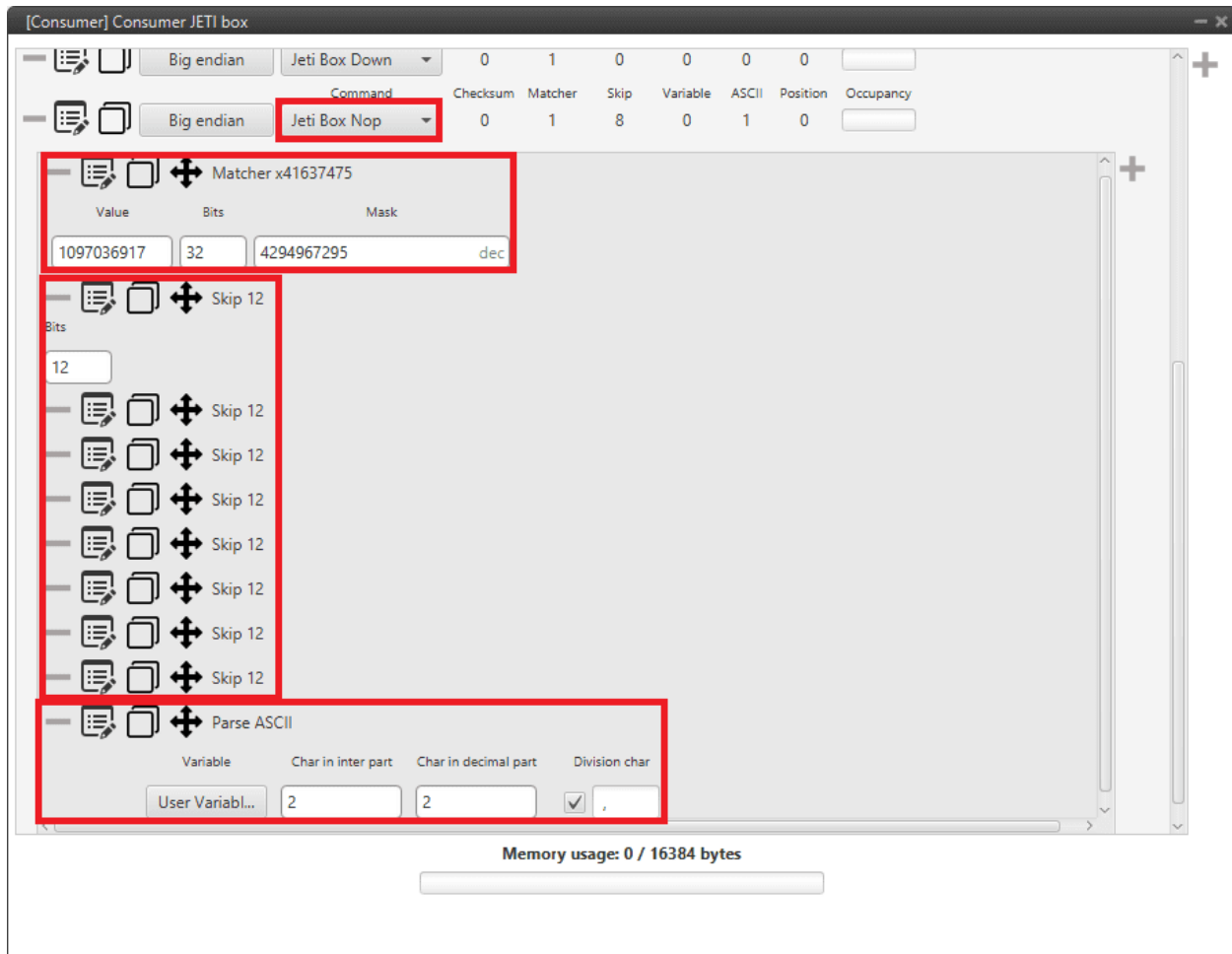


Fig. 32: JETI box last custom message example

3.6 Reading Arbitration Messages

Note that those messages are generated only if arbitration and messages are enabled in MEX.

Arbiter will send its telemetry in little endian format, using its CAN-TX ID.

The appropriate arbitration message format is described in [Arbitration -> CAN Bus protocol section](#) of the **MEX Software manual**.

3.7 Reading/Sending RPMs

This section explains the steps to read RPMs.

MEX PDI Builder side

1. Go to Input/Output menu ⇒ **GPIO section**.

RPM can be read on the available digital inputs I/O 1-4. The chosen pin needs to be configured as “**GPIO as input**”. In the example shown here, I/O1 is chosen.

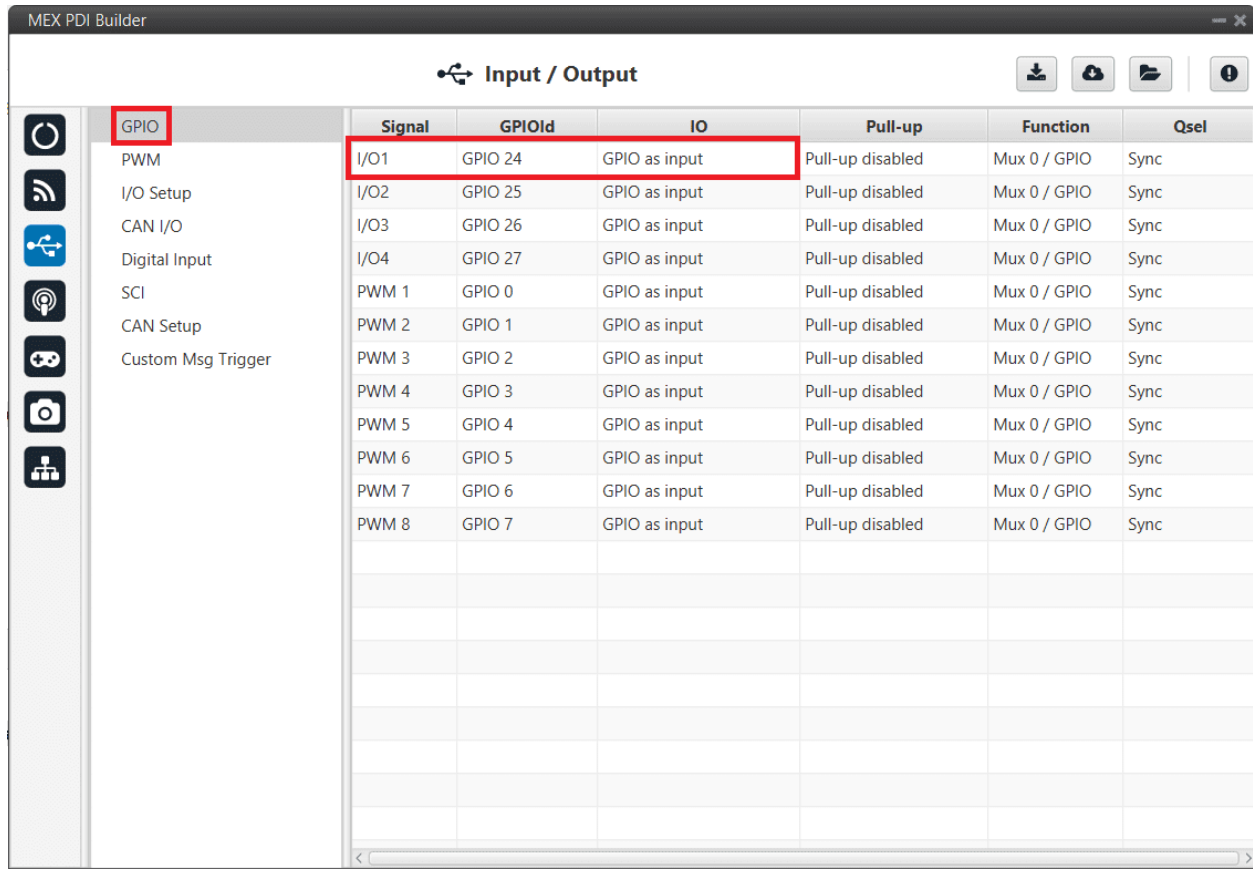


Fig. 33: Reading RPMs - GPIO configuration

2. Go to Input/Output menu ⇒ **Digital Input** section.

There are 4 possible producers: CAP 1-4. One needs to be chosen and linked to one of the RPMs consumers (RPM 1-4). For more information about Digital input configuration, read the [Digital Input](#) section of this manual.

Then, select an **edge detection option**: “first rising edge” or “first falling edge”.

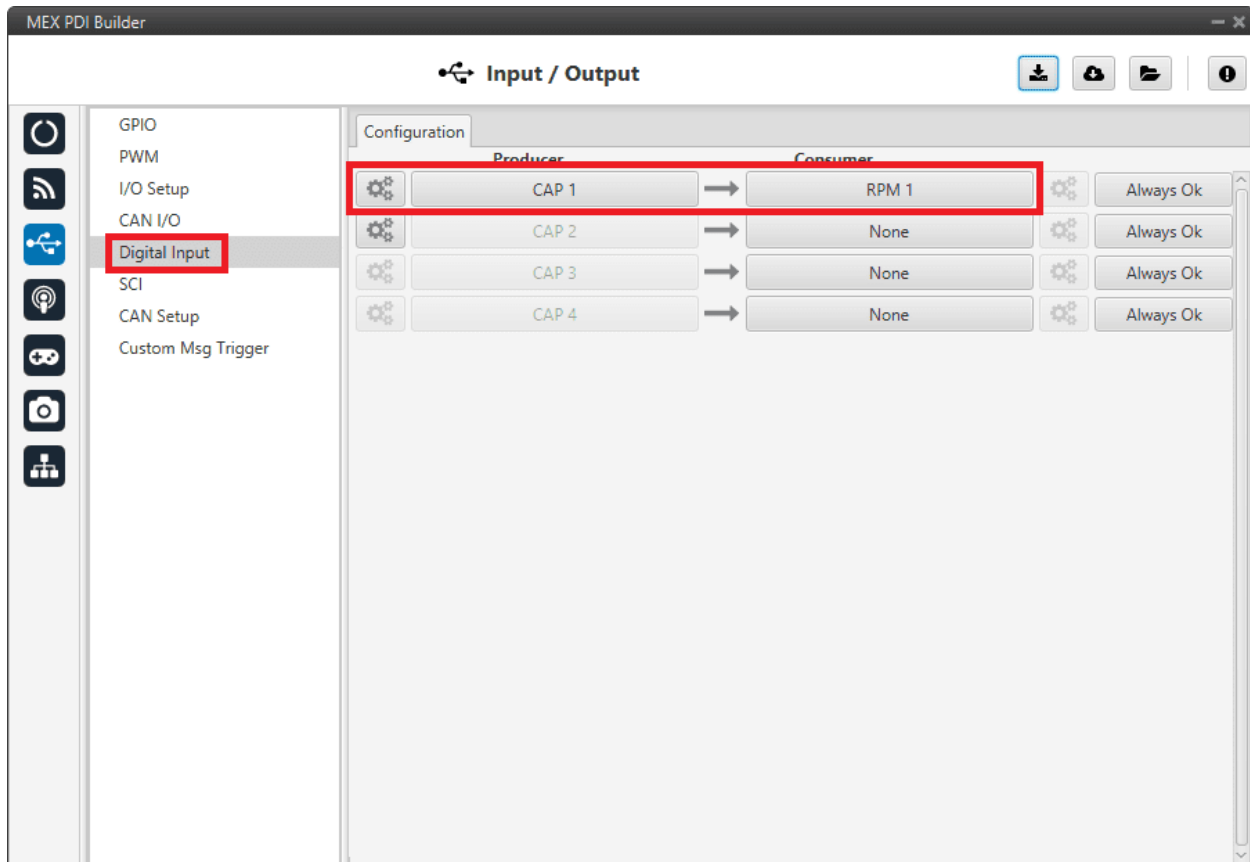


Fig. 34: Reading RPMs - Digital Input



Fig. 35: Reading RPMs - Digital Input configuration

- Go to Sensors menu ⇒ RPM section ⇒ **RPM 1 tab** (since RPM 1 has been selected in the Digital Input section).
Here the expected pulse needs to be defined. For more information about RPM configuration, read section *RPM* of this manual.

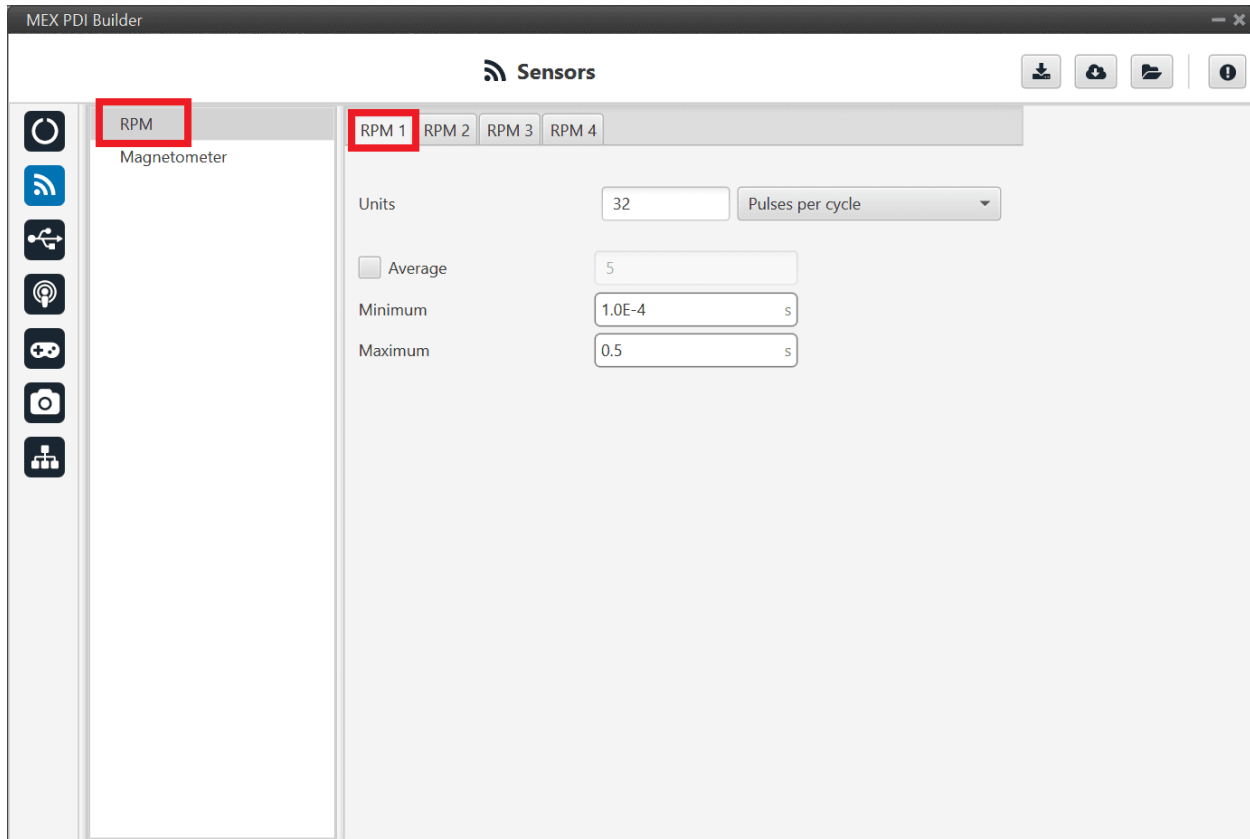


Fig. 36: Reading RPMs - RPM configuration

4. Go to Input/Output menu ⇒ CAN I/O section ⇒ **Configuration tab**.

Connect **CAN Telemetry** to a **CAN Output Filter** as follows (in this example, the **RPM1** variable is sent over CAN B bus of the MEX):

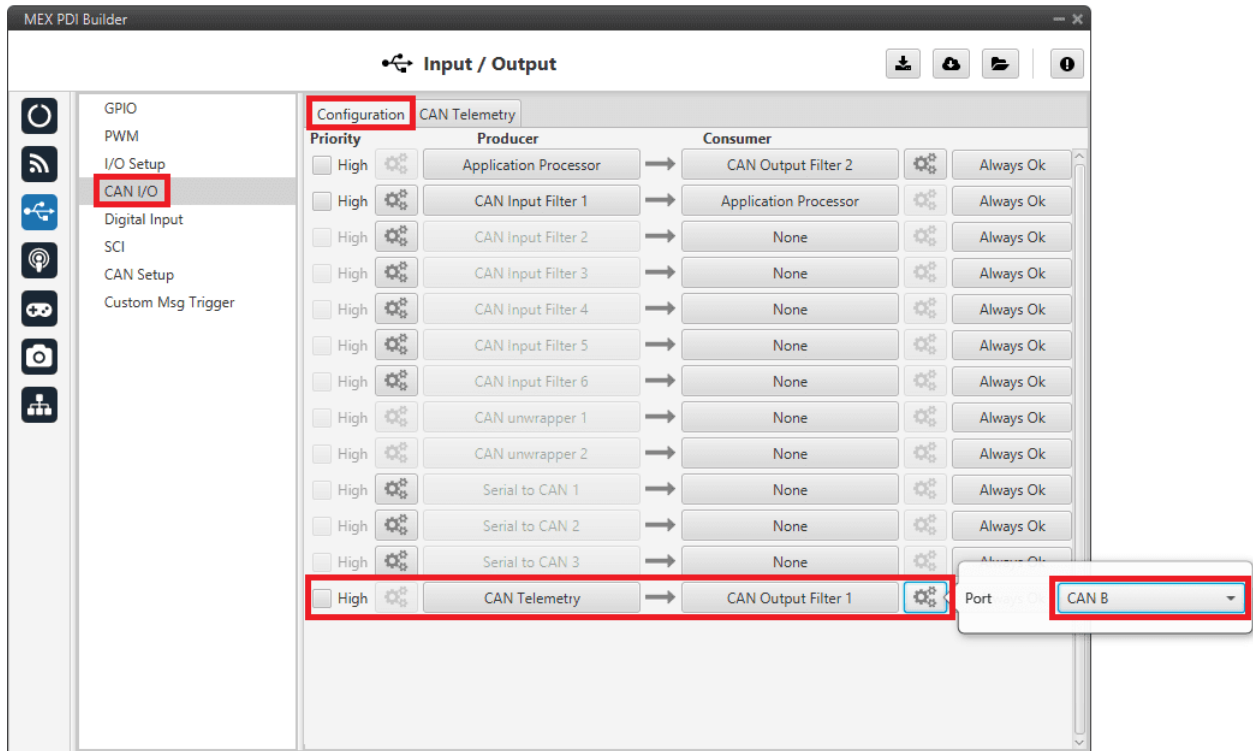


Fig. 37: Reading RPMs - CAN Custom Telemetry

5. Go to the **CAN Telemetry** tab.

A new telemetry message needs to be created with its correspondent ID, endianness and period.

- **Can ID:** 1200.
- **Endianness:** Little.
- **Period:** 0.01 s.

In the telemetry message one of MEX variables needs to be selected. As RPM1 has been chosen as consumer in the **Digital Input**, the variable required to send is **RPM1**.

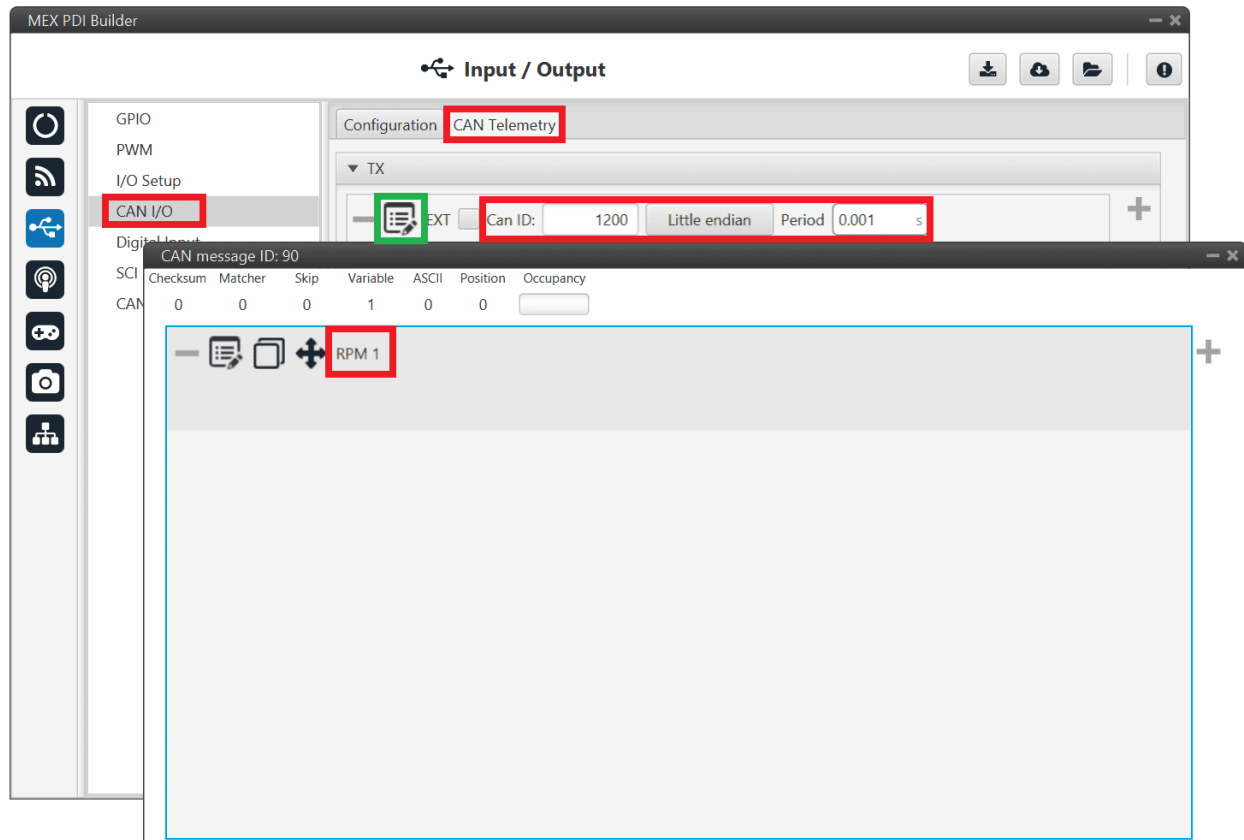


Fig. 38: Reading RPMs - CAN Custom Telemetry configuration

More information on the configuration of Telemetry messages can be found in the [CAN Telemetry section](#).

1x PDI Builder side

6. Go to UI menu ⇒ Variables section ⇒ **Reals Vars** tab.

Rename a Real User Variable (32 bits) that will be used to store the value received from MEX:

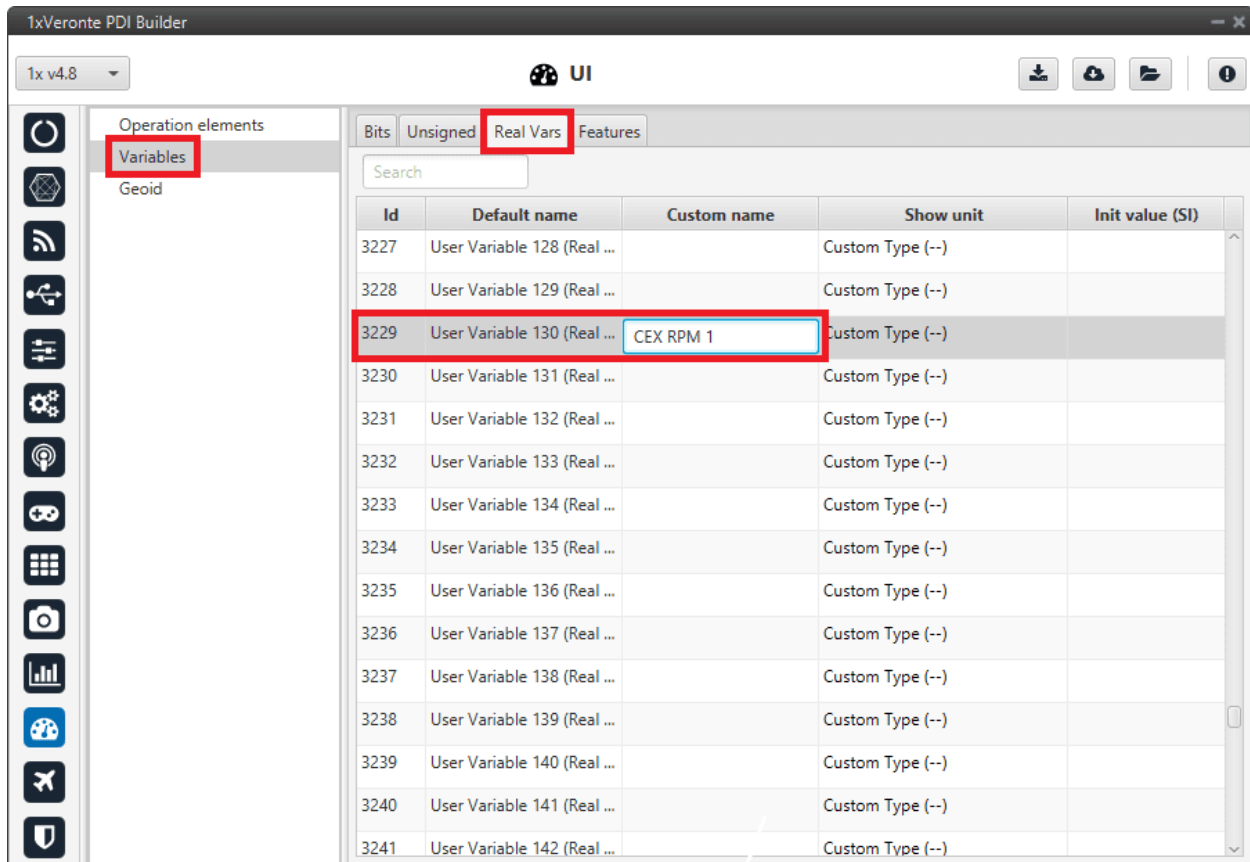


Fig. 39: 1x PDI Builder - User Variable renamed

7. Go to Input/Output menu ⇒ CAN Setup section ⇒ **Mailboxes tab**.

Configure some mailboxes to receive the message with **ID 1200**:

8. Go to Input/Output menu \Rightarrow CAN Setup section \Rightarrow **Configuration tab.**

3.7. Reading/Sending RPMs

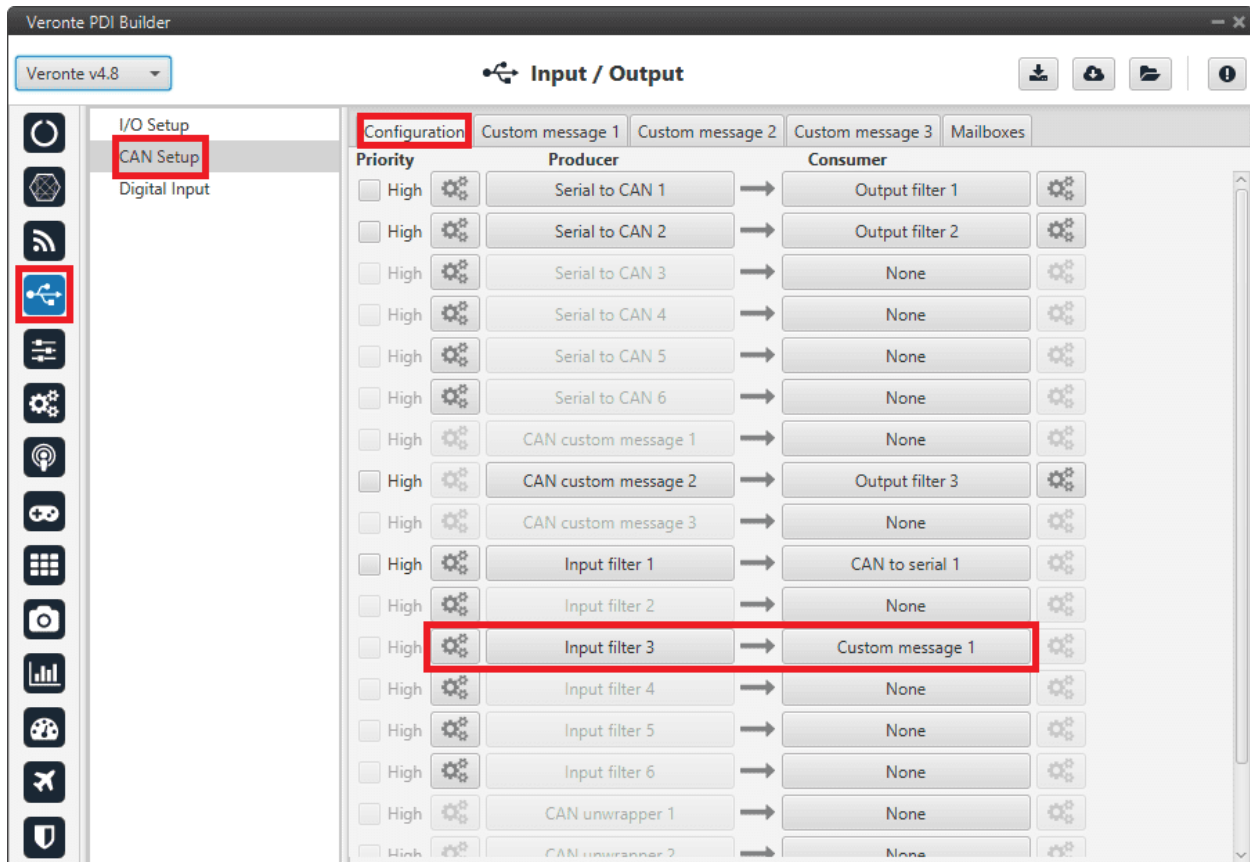


Fig. 41: 1x PDI Builder - Input filter

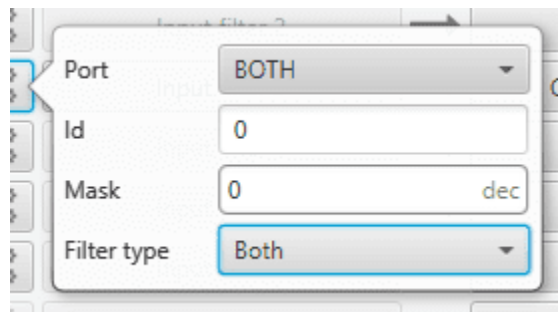


Fig. 42: 1x PDI Builder - Input filter configuration

- Go to Input/Output menu ⇒ CAN Setup section ⇒ **Custom message 1 tab** (as Custom Message 1 has been selected as consumer).

Configure the message reading and store the received value in the user variable renamed above:

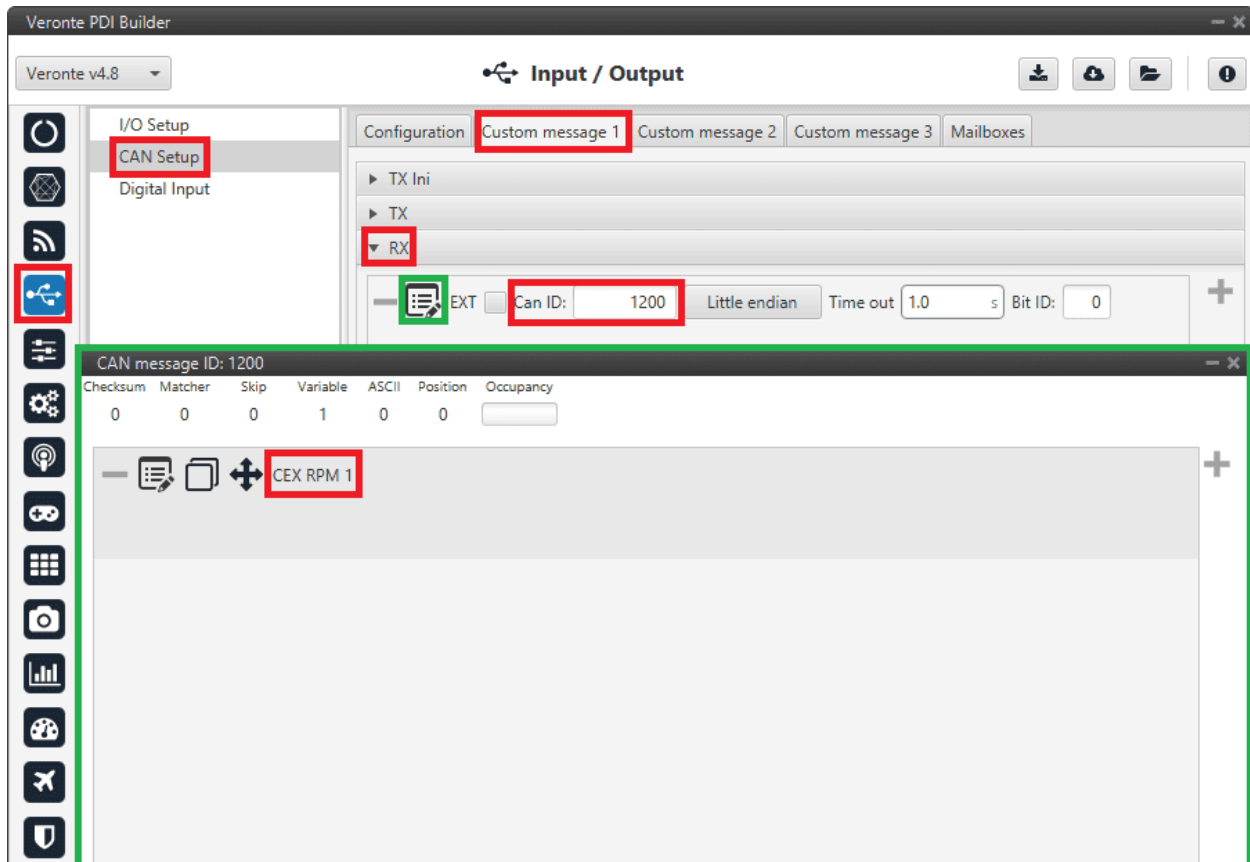


Fig. 43: 1x PDI Builder - Custom Message configuration

3.8 Scorpion

MEX is able to read telemetry from Tribunus ESCs by connecting it to one of its serial ports. The following steps configure MEX to read this telemetry:

1. Go to Devices menu ⇒ **Scorpion tribunus** section

Enable it and configure it as shown:

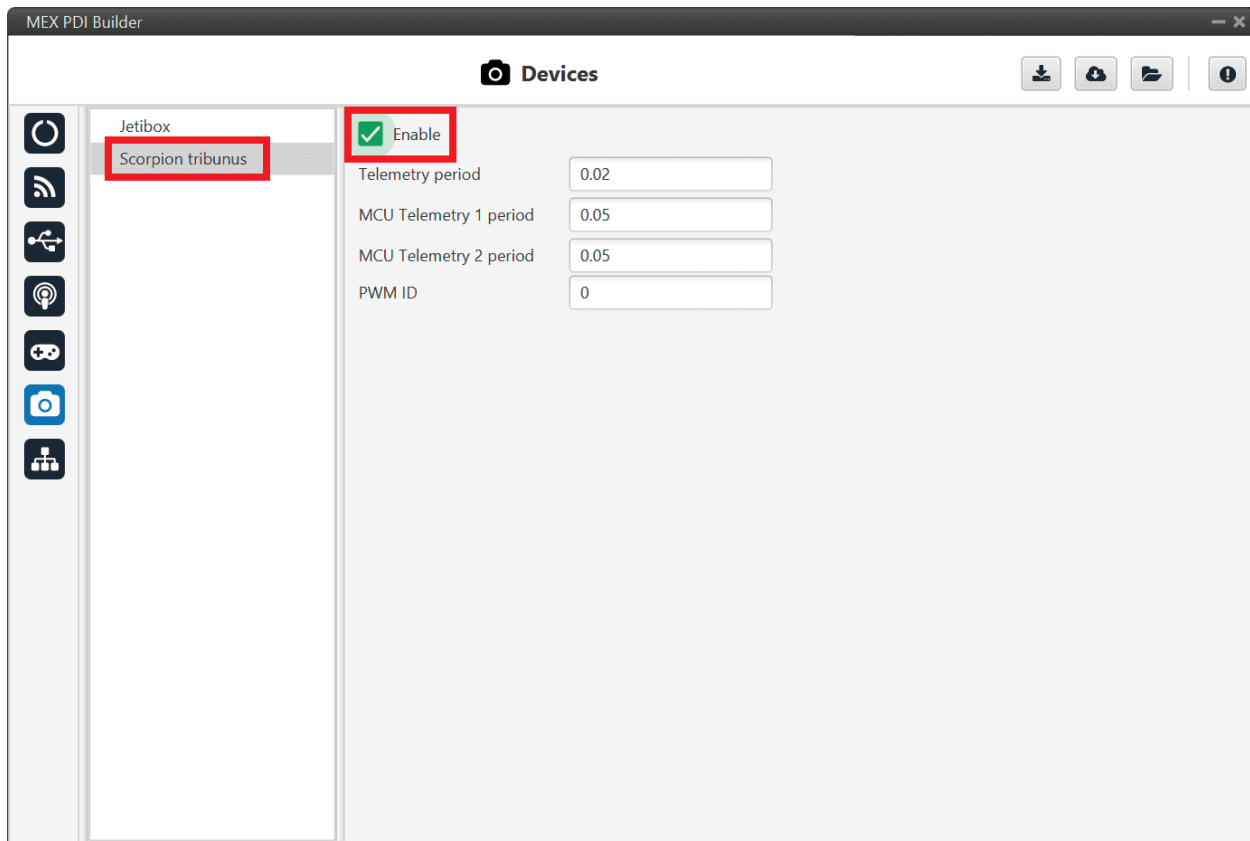


Fig. 44: Scorpion tribunus Devices configuration

2. Go to Input/Output menu ⇒ **I/O Setup section**.

Link the specific Tribunus ESC consumer to the desired port:

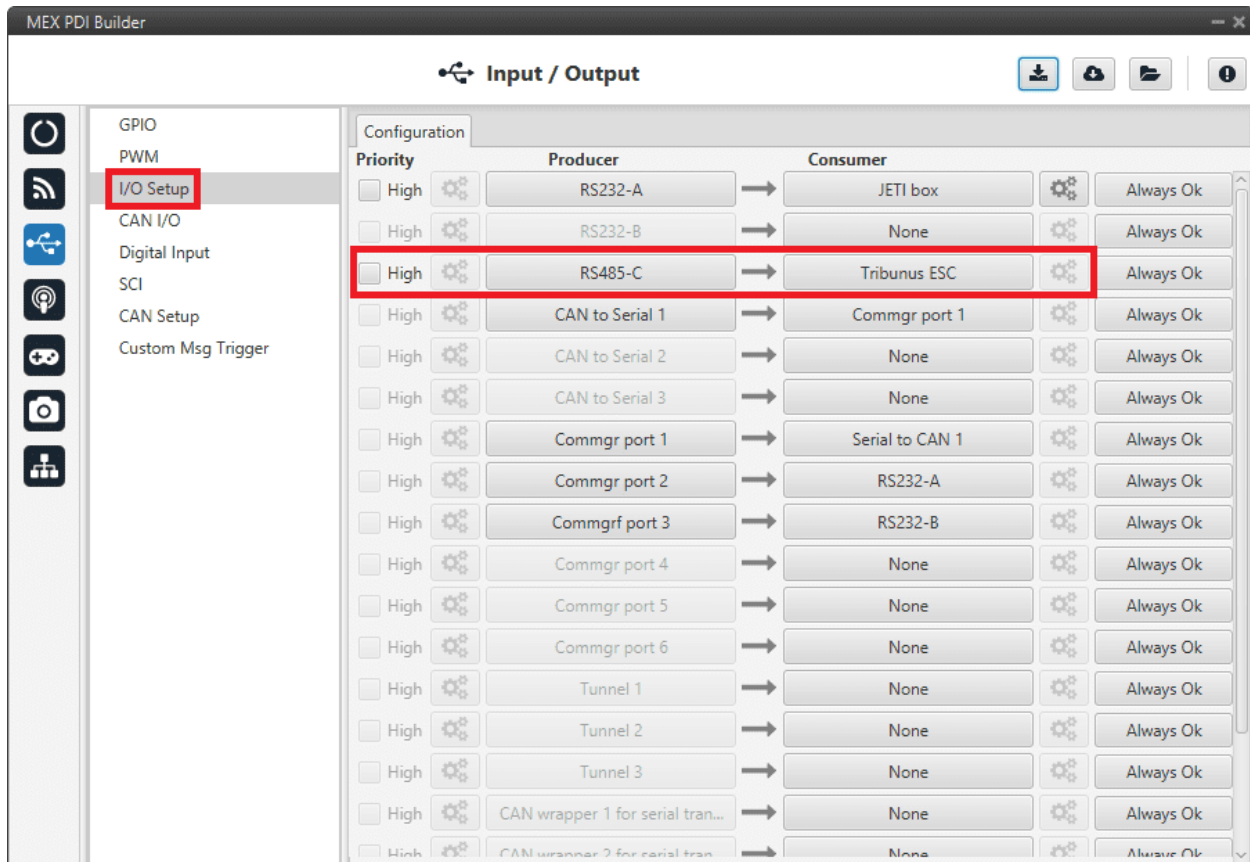


Fig. 45: Scorpion Consumer

3.9 Serial

The user can send information through the serial ports of MEX. For this purpose, please read the following steps:

MEX PDI Builder side

1. I/O Setup configuration

- Reception of serial information on RS232-A (producer) is stored in Serial to CAN 2 (consumer).
- Transmission of serial information is sent using the CAN to Serial 2 (producer) through RS232-A (consumer).

(‘Serial to CAN’ and ‘CAN to Serial’ are explained in the I/O section of this manual, click [here](#) to access it)

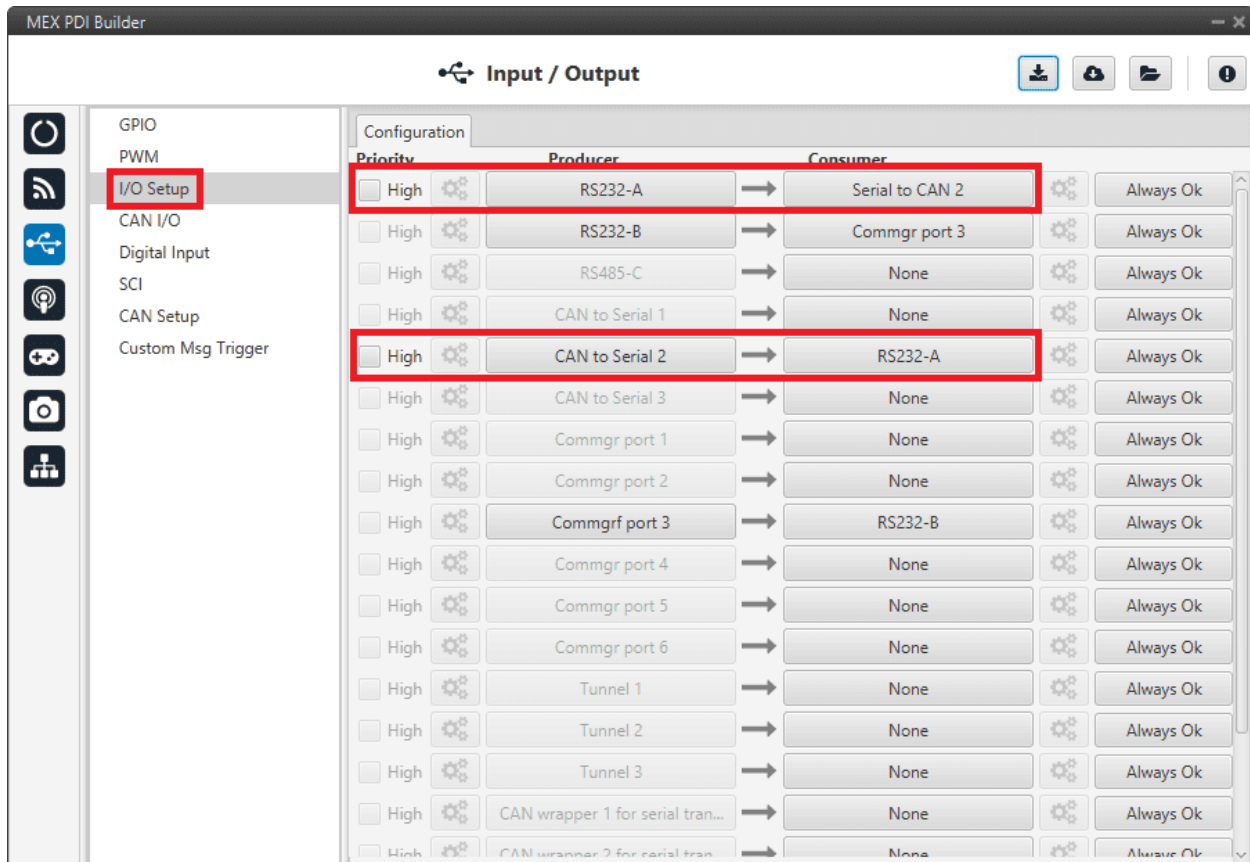


Fig. 46: Serial - I/O Setup

2. CAN I/O configuration

- The information that will be **sent over serial port RS232-A** is going to be received on the MEX over its **CAN B** port. A **mask Id of 51** is added to the **Input filter**.

The incoming information (from Veronte Autopilot 1x) is processed in 'CAN to Serial 2'.

- The information **coming from port RS232-A** and processed as 'Serial to CAN 2' is going to be linked to an **Output filter**.

The information of 'Serial to CAN 2' is going to be sent over **CAN B** with a **mask ID of 50** (to be read by Veronte Autopilot 1x).

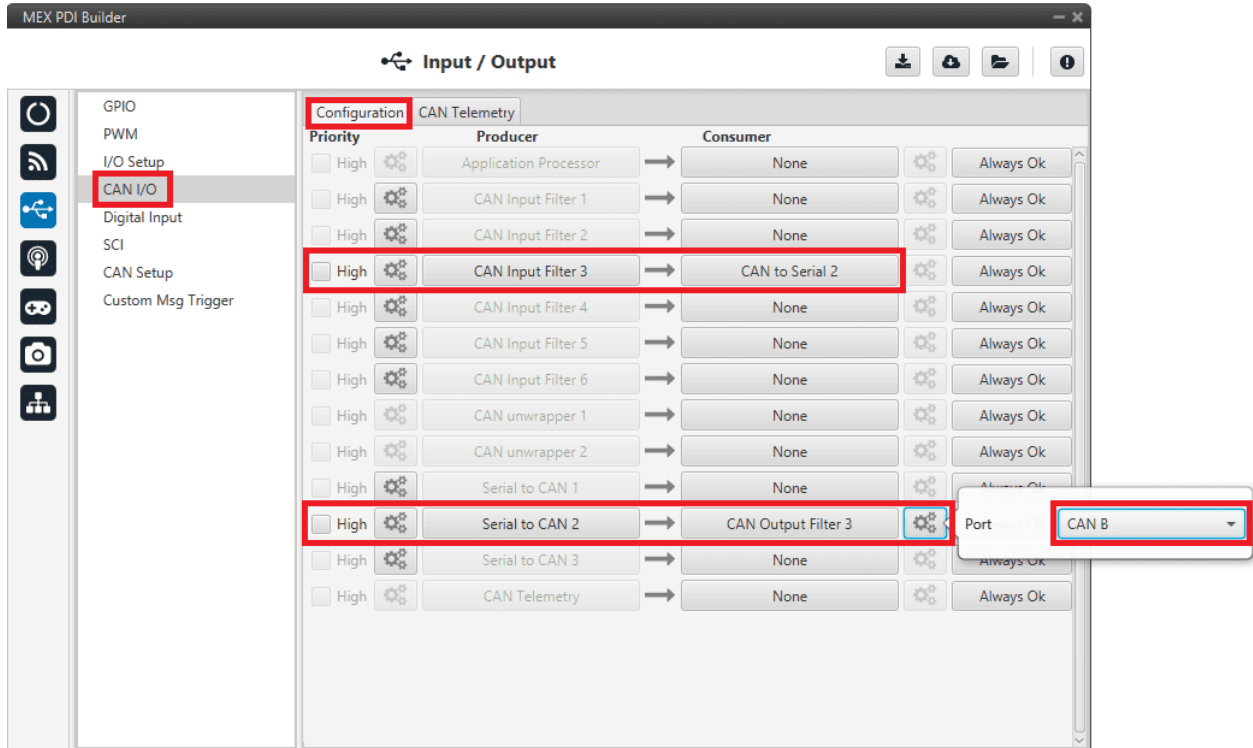


Fig. 47: Serial - CAN I/O

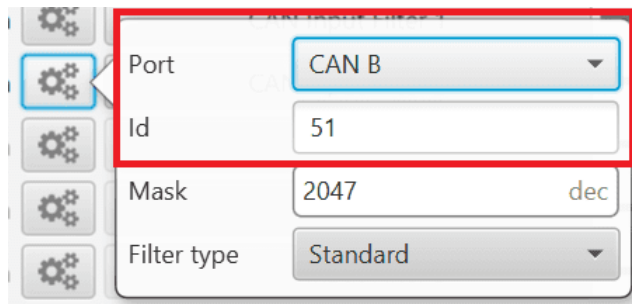


Fig. 48: Serial - Input filter configuration

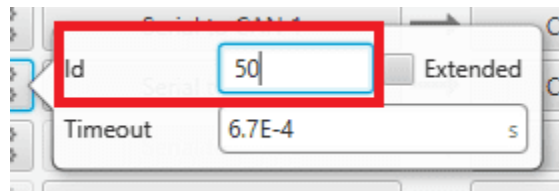


Fig. 49: Serial - Serial to CAN configuration

3. CAN Setup (mailboxes) configuration

Mailboxes need to be defined for the reception of CAN messages. In the example above, mailboxes for **ID 51** need to be added on **CAN B** port.

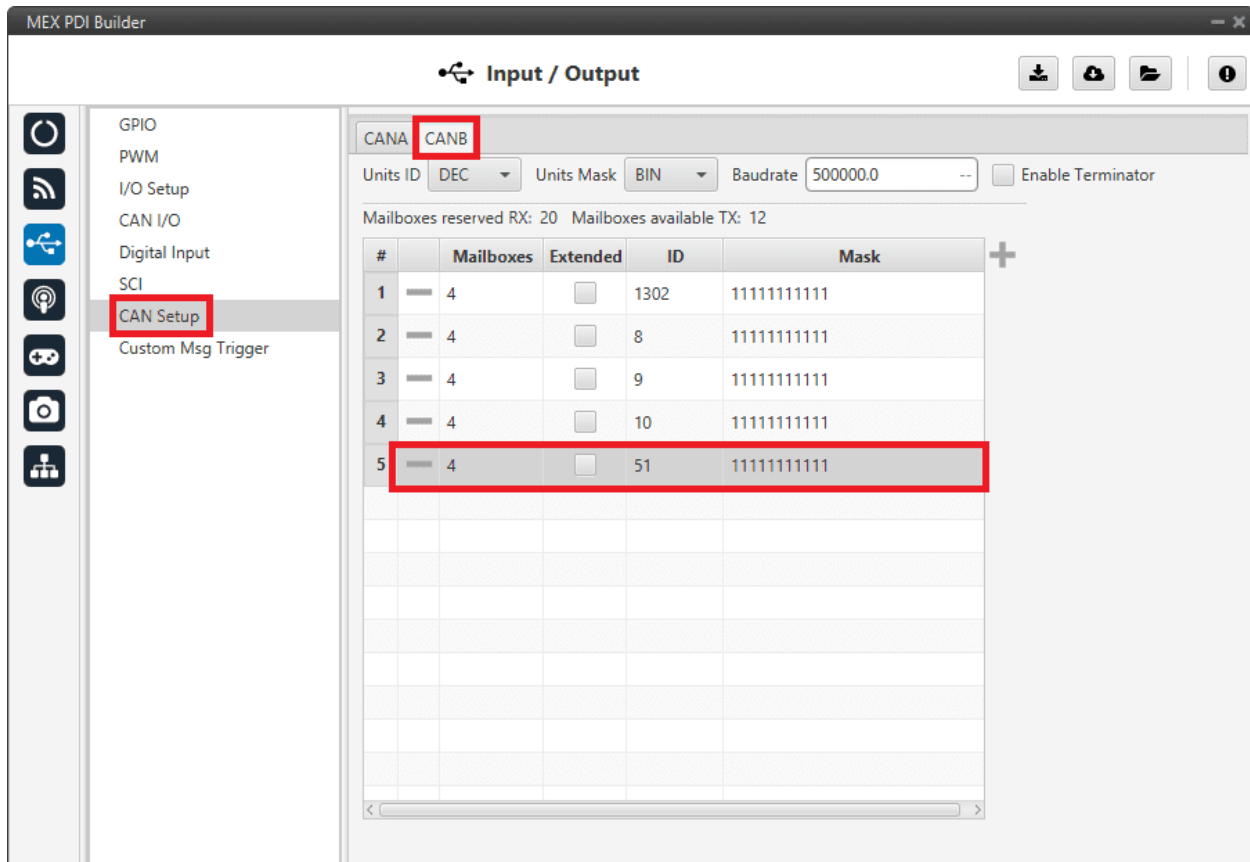


Fig. 50: Serial - CAN Setup configuration

1x PDI Builder side

- On the **I/O Setup**, link an 'RS Custom Message' to a 'Serial to CAN' with the serial data that autopilot is going to send to MEX.

Then, link a 'CAN to Serial' to another 'RS Custom Message' with the expected serial messages that MEX will receive in the selected serial port.

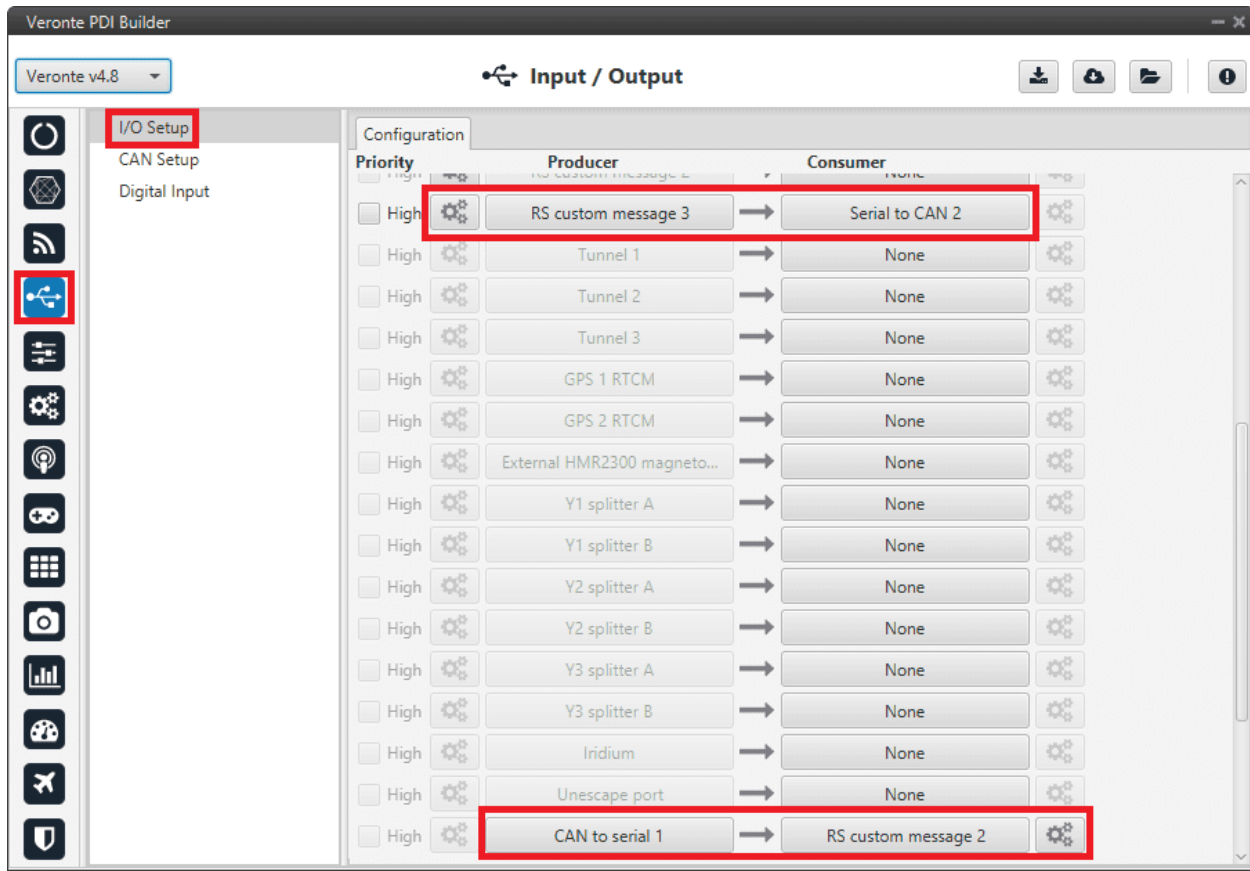


Fig. 51: Serial - 1x I/O Setup configuration

- For the **CAN I/O**, the same IDs employed in MEX for the Input and Output filters are going to be employed on Veronte Autopilot 1x side, but they must be inverted.

Therefore, the **Input filter** linked to the chosen 'CAN to Serial' needs to have **ID 50**. And the **Output filter** linked to the chosen 'Serial to CAN' will have **ID 51**.

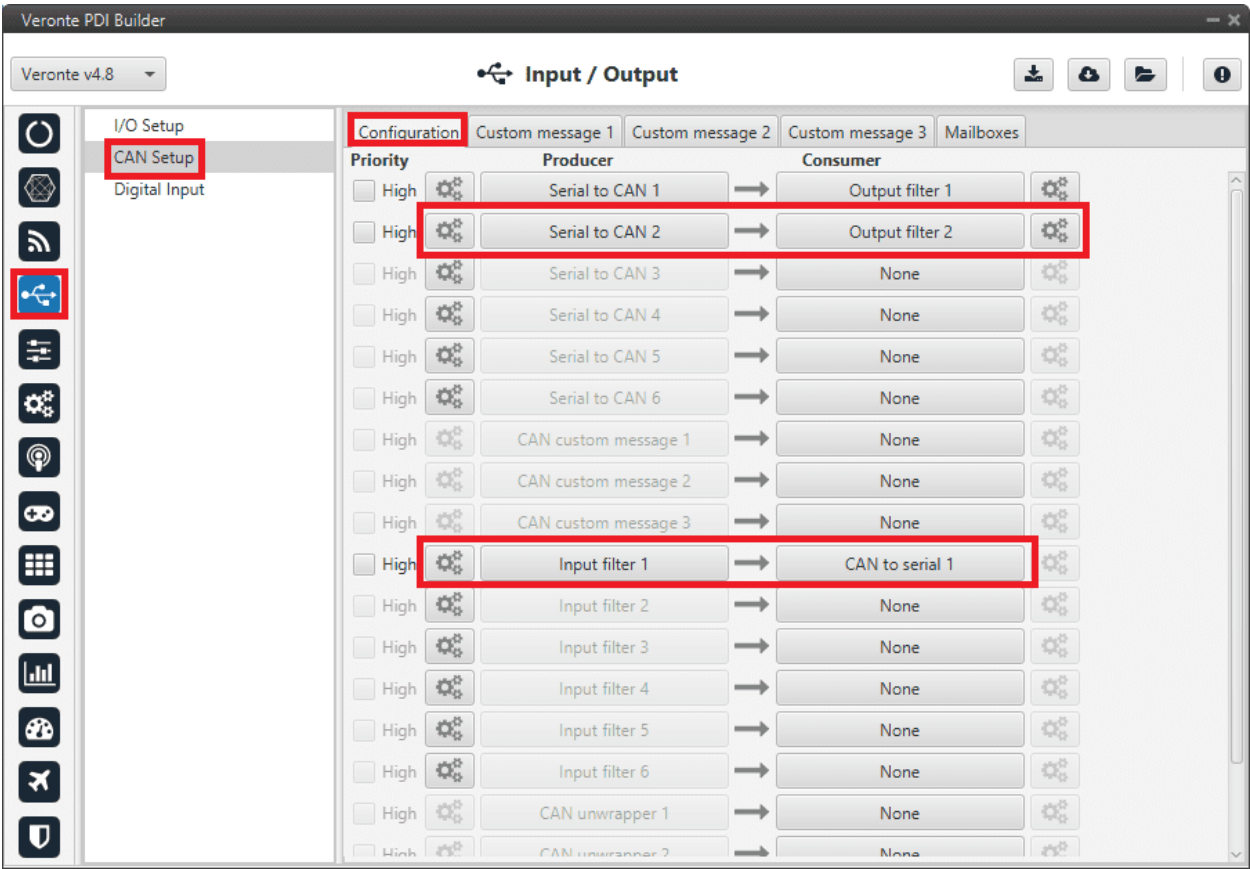


Fig. 52: Serial - 1x CAN Seup

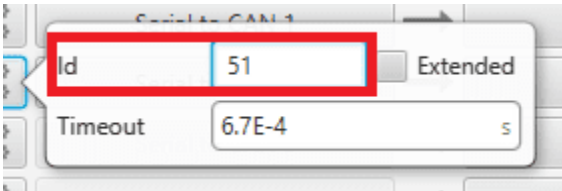


Fig. 53: Serial - 1x Serial to CAN configuration

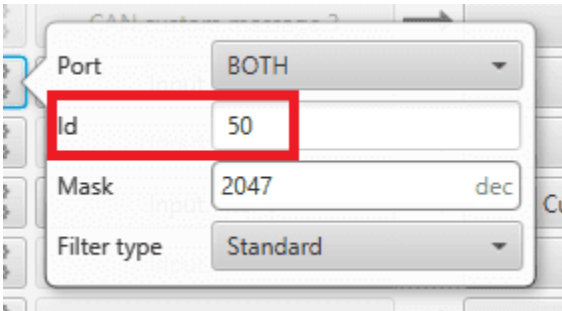


Fig. 54: Serial - 1x Input filter configuration

6. Some **mailboxes** with **ID 50** have to be created on whichever chosen reception CAN bus.

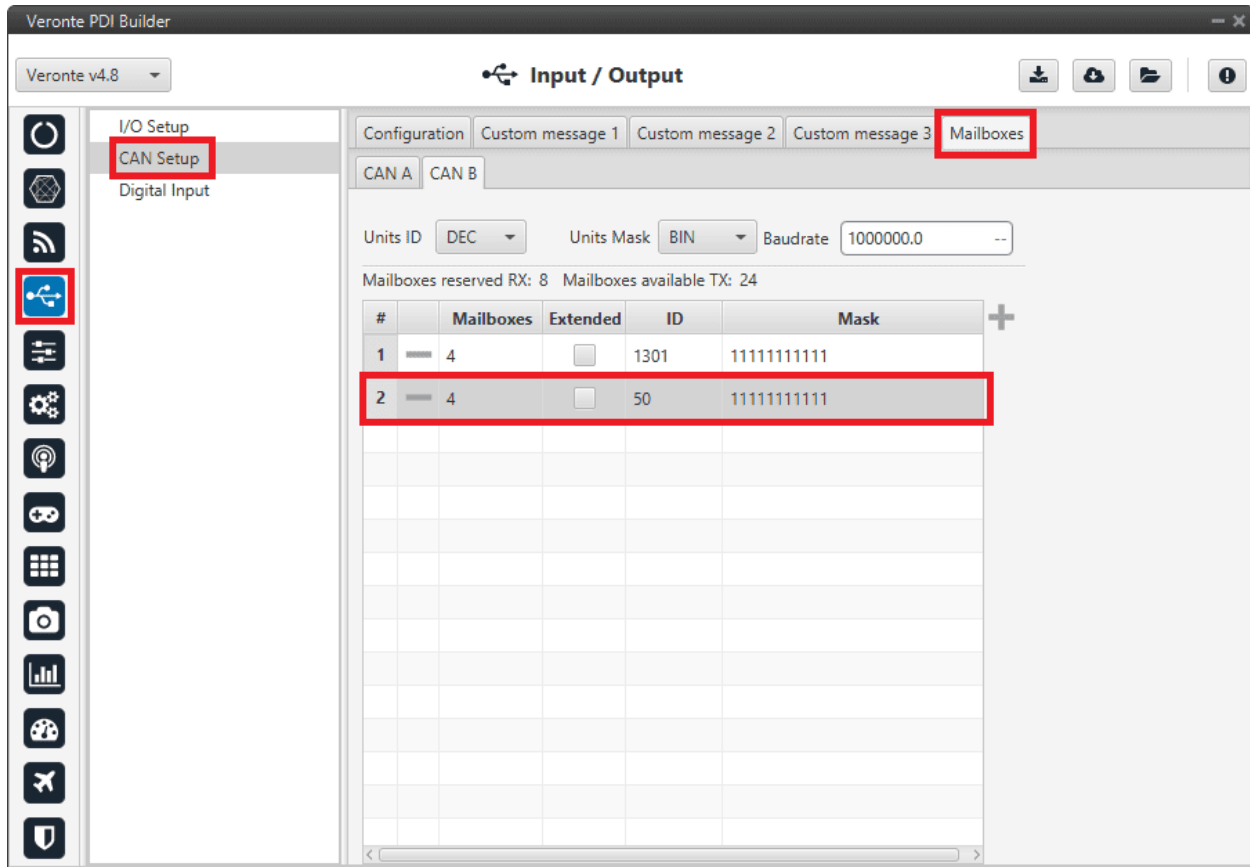


Fig. 55: Serial - 1x Mailboxes configuration

3.10 Using MEX as external magnetometer Honeywell HMR2300 for Autopilot 1x

MEX has to be configured with *MEX PDI Builder* to send the magnetometer measurements by messages, in the same way as **Honeywell HMR2300** does. **Autopilot 1x** also requires a configuration, which is made with *1x PDI Builder*.

MEX PDI Builder side

1. Go to Input/Output menu ⇒ **I/O Setup** section.

Connect the **Custom message producer 1** or **2** to the **RS232-A**, **RS232-B** or **RS485-C** port.

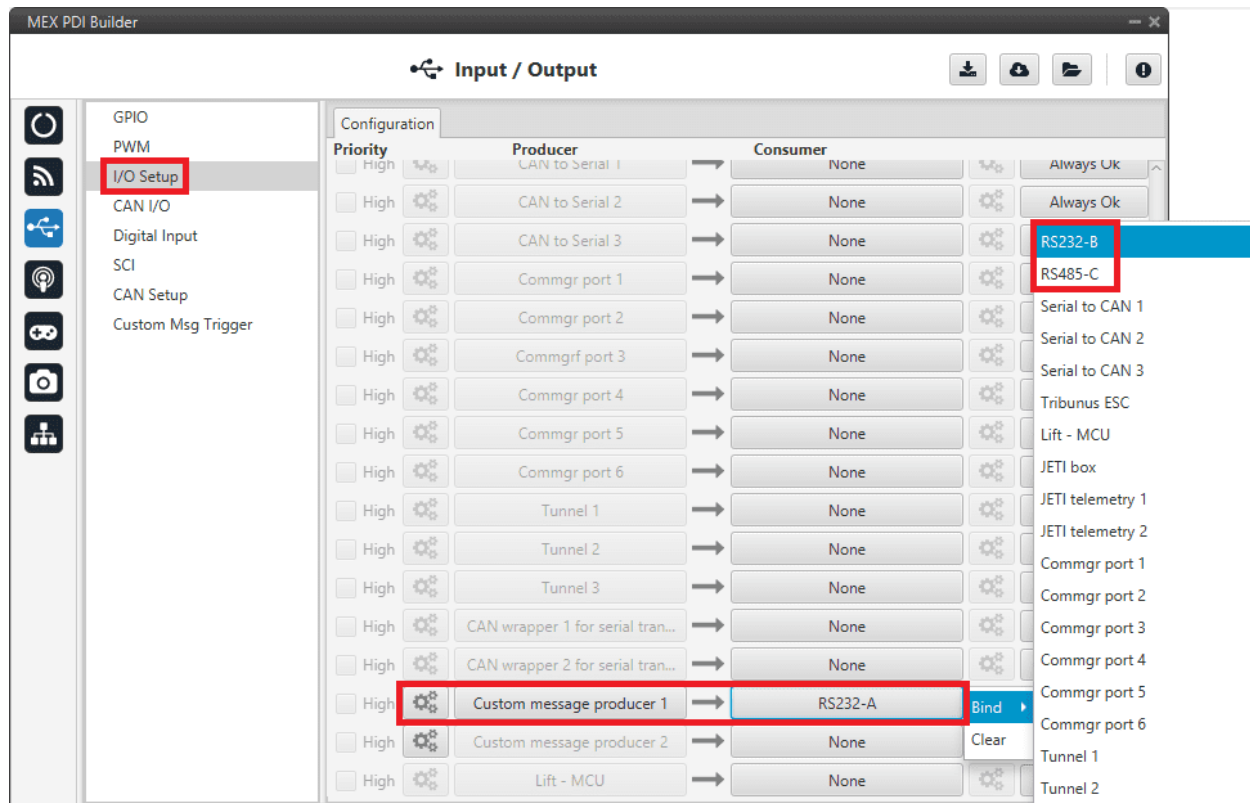




Fig. 56: I/O Setup configuration

- Click on the  button of the chosen **Custom message producer** and add a new message (clicking on ) , then configure it as **Big endian** and set a **Period** of 0.02 s.

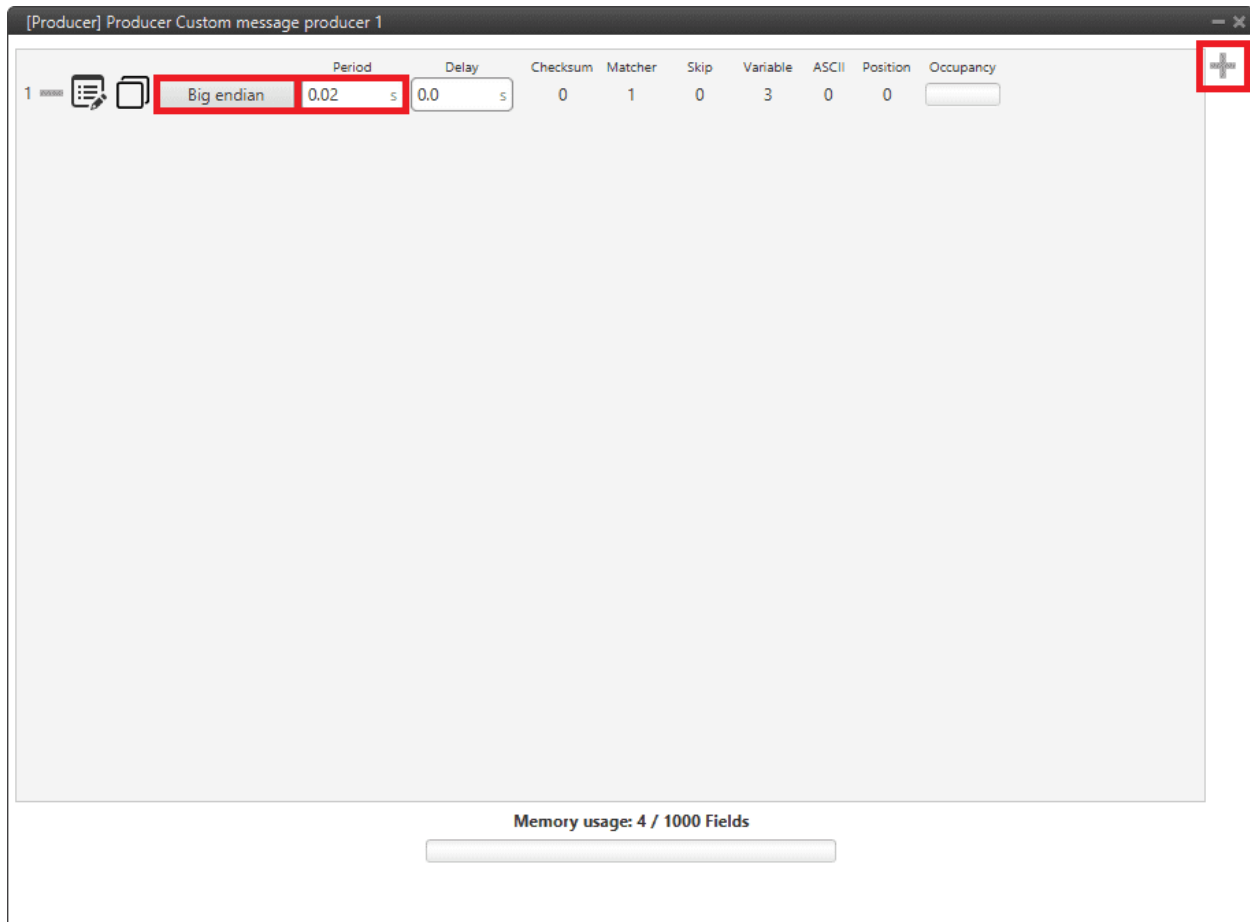


Fig. 57: I/O Setup - Custom message configuration

3. Click on **+** to add the 3 following **Real variables**: **313**, **314** and **315**. Then configure them according to the following table:

Parameter	Configuration
Compression	Compress - Bits Signed
Bits	16
Encode	Min: -2.1845334E-4 / Max: 2.1844667E-4
Decode	Min: -32768 / Max: 32767

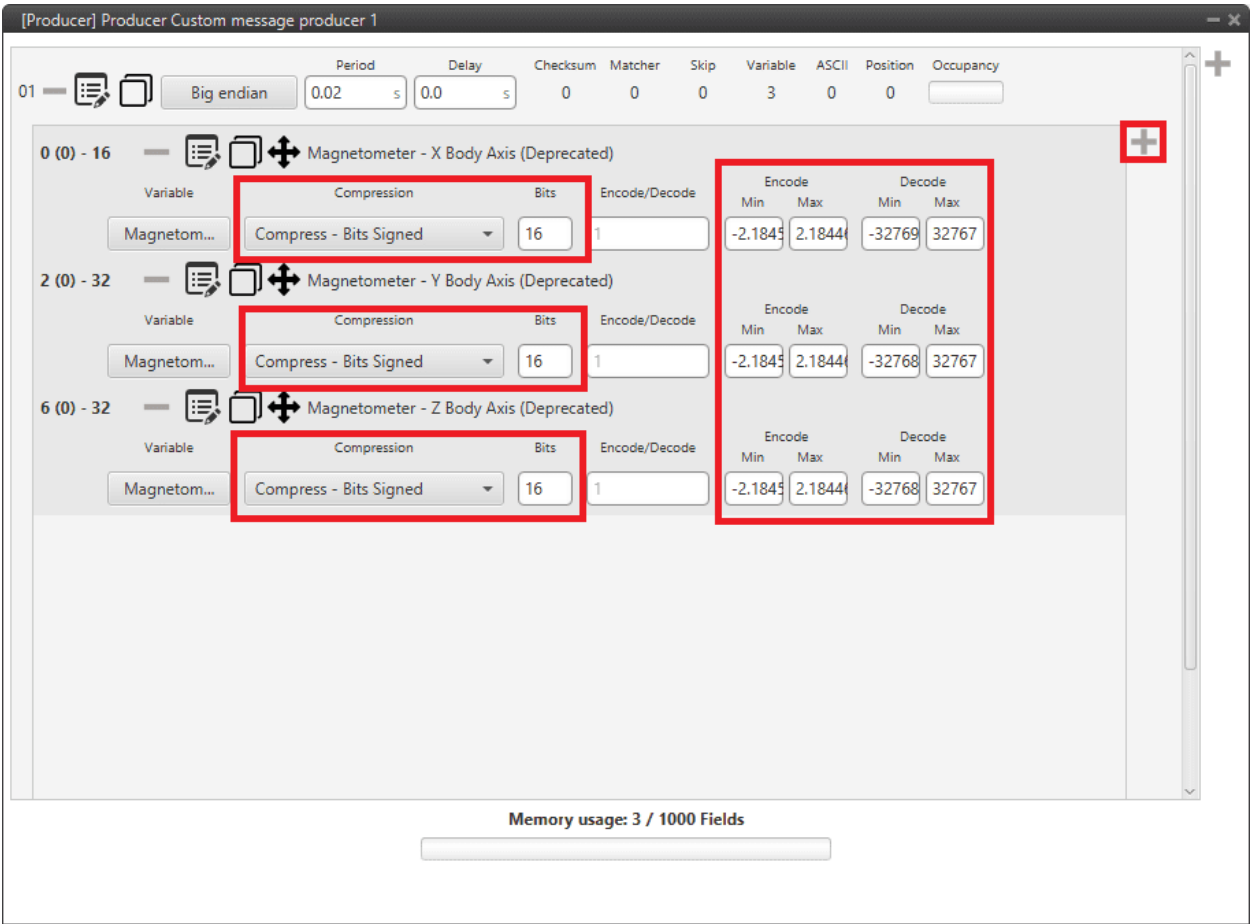


Fig. 58: I/O Setup - Custom message Variables configuration

4. Click on **+** ⇒ **Matcher** to add a matcher to the message and configure it as follows:

Parameter	Configuration
Value	13
Bits	8

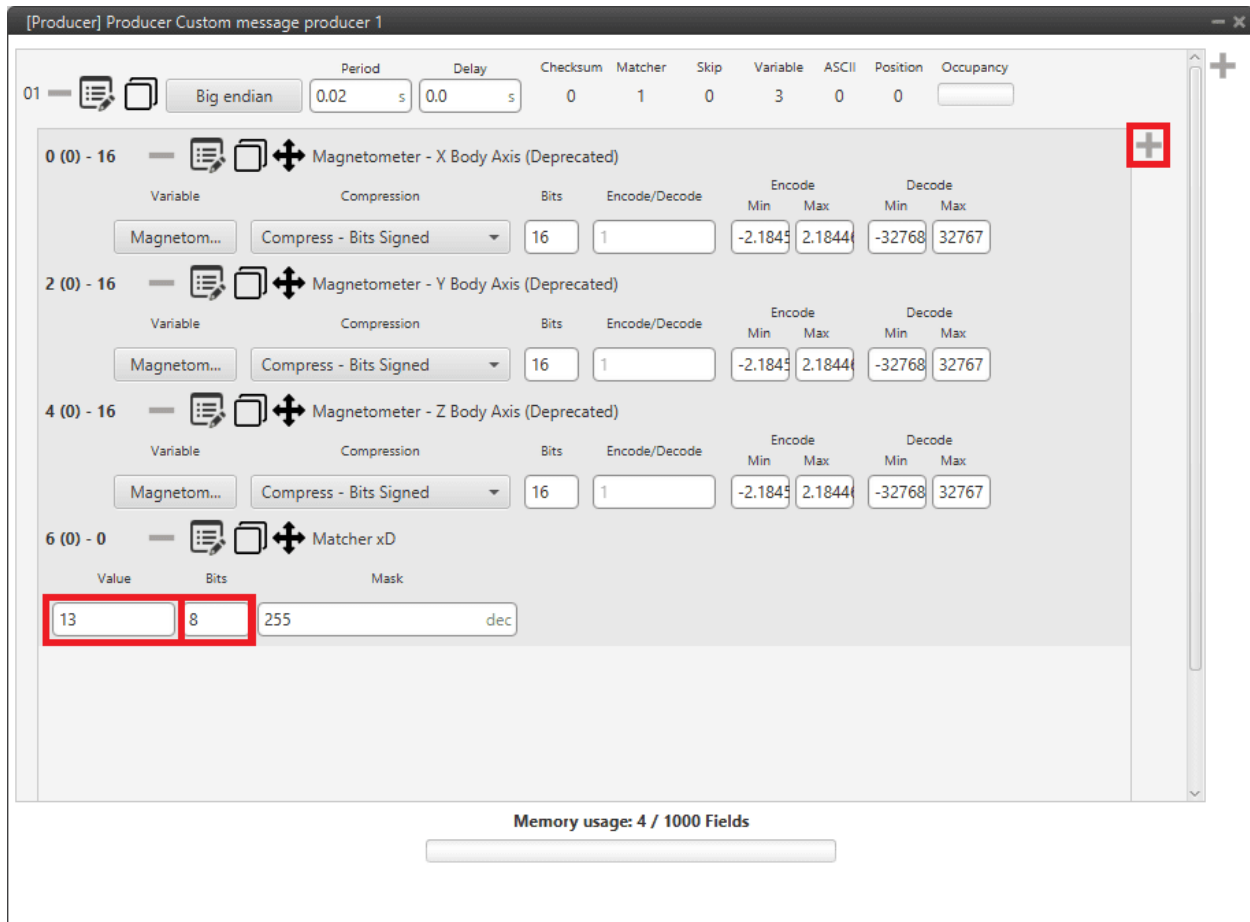


Fig. 59: I/O Setup - Custom message Matcher configuration

Warning: The order of variables and matcher must be the same.

5. Go to **SCI** menu.

Configure **SCI A** tab as follows:

Parameter	Configuration
Baudrate	115200
Length	8
Stop	1
Parity	Disabled
Use adress mode	Disabled

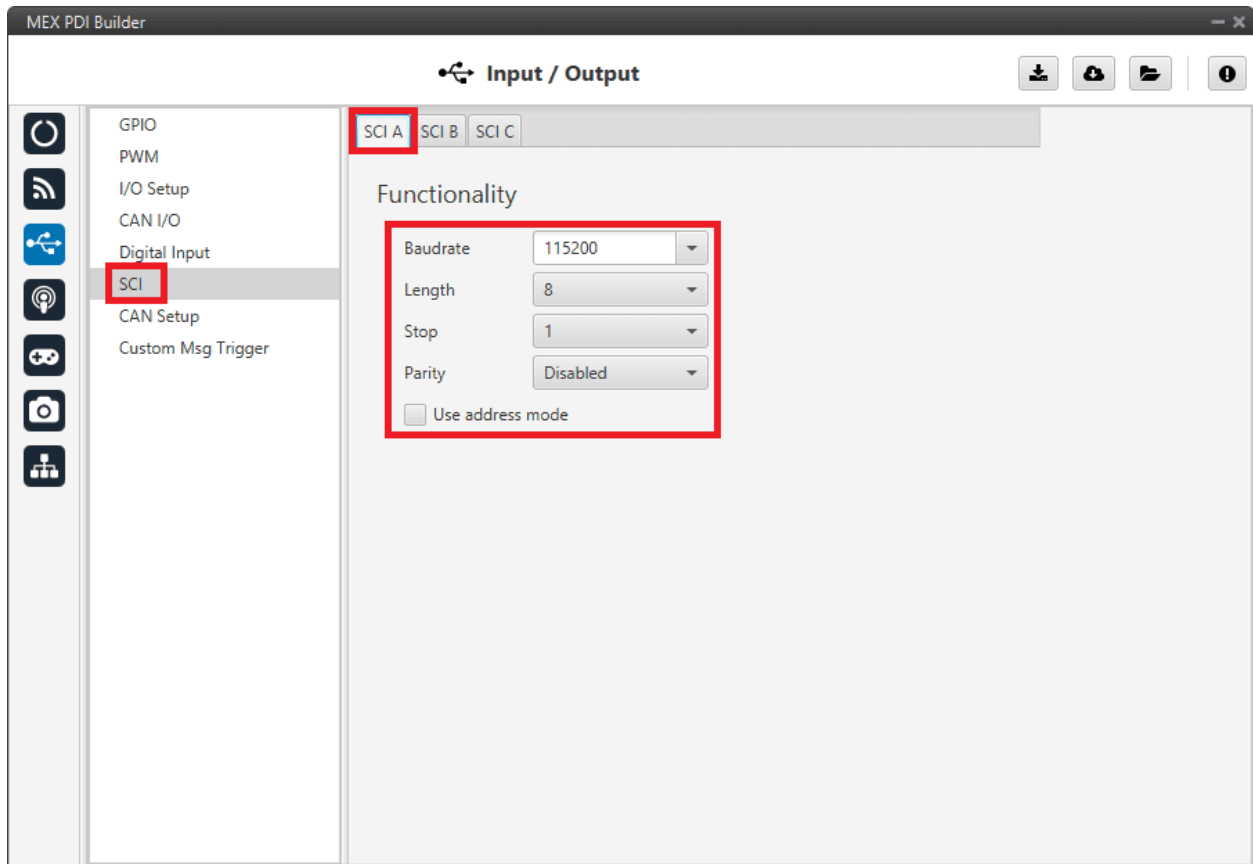



Fig. 60: SCI configuration

6. Once finished the configuration, click on  to write the configuration in **MEX**.

Matcher explanation

The matcher is a binary code employed to discard messages with errors.

The length and the number represented can be edited clicking on  of the matcher. If the receiver does not receive the exact same matcher sequence, the entire message will be discarded, since it will be considered corrupted.

For more information about custom messages and matchers, read [Custom Messages section](#) of the **1x PDI Builder** user manual.

1x PDI Builder side

The configuration for the **Autopilot 1x** is explained in [MEX as Magnetometer Honeywell HMR2300 -> Integration examples section](#) of the **1x PDI Builder** user manual.

3.11 CAN Isolator

MEX can operate as a CAN bus isolator, since it manages both CAN buses as desired. The system has a built-in microcontroller; which manages CAN buses in real-time. Both buses are not electrically connected, in consequence, electrical signals that do not follow the CAN protocol will be isolated.

The functionalities of a CAN isolator are the following:

- *CAN input filter*
- *Filtered CAN subnet*
- *CAN tunnel*

3.11.1 Filtered CAN subnet

With **MEX** it is possible to filter certain messages from one CAN line (Line A) and transmit specific information through the other one (Line B). This is a specific case of *CAN input filter*.

The following diagram summarizes an application example to filter a CAN subnet:

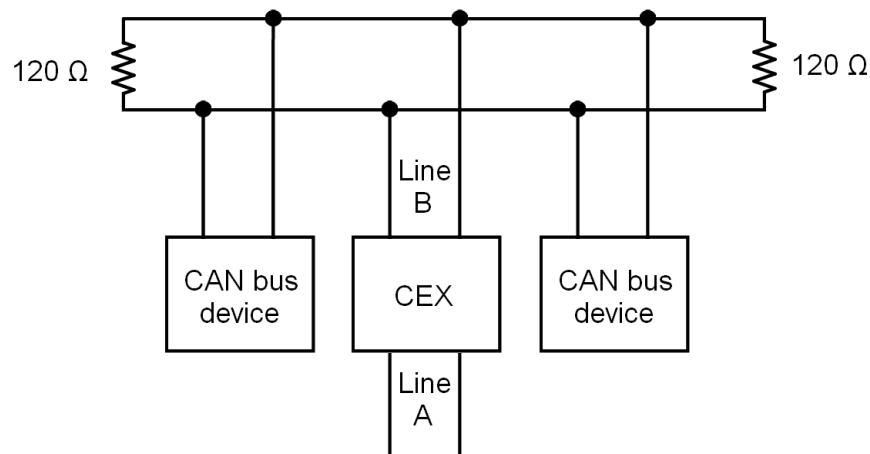
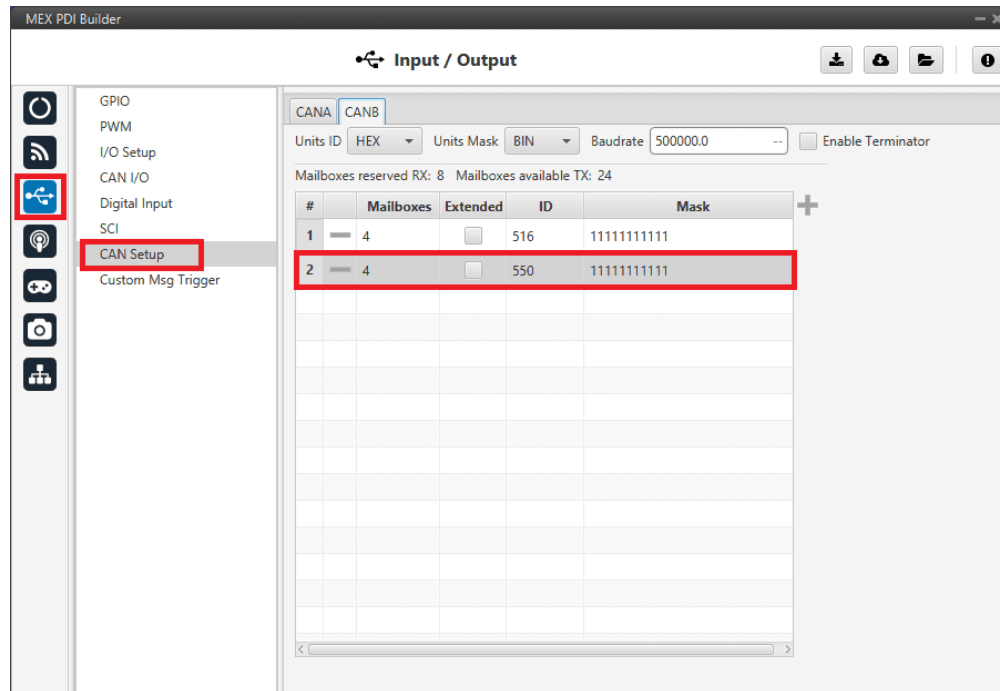


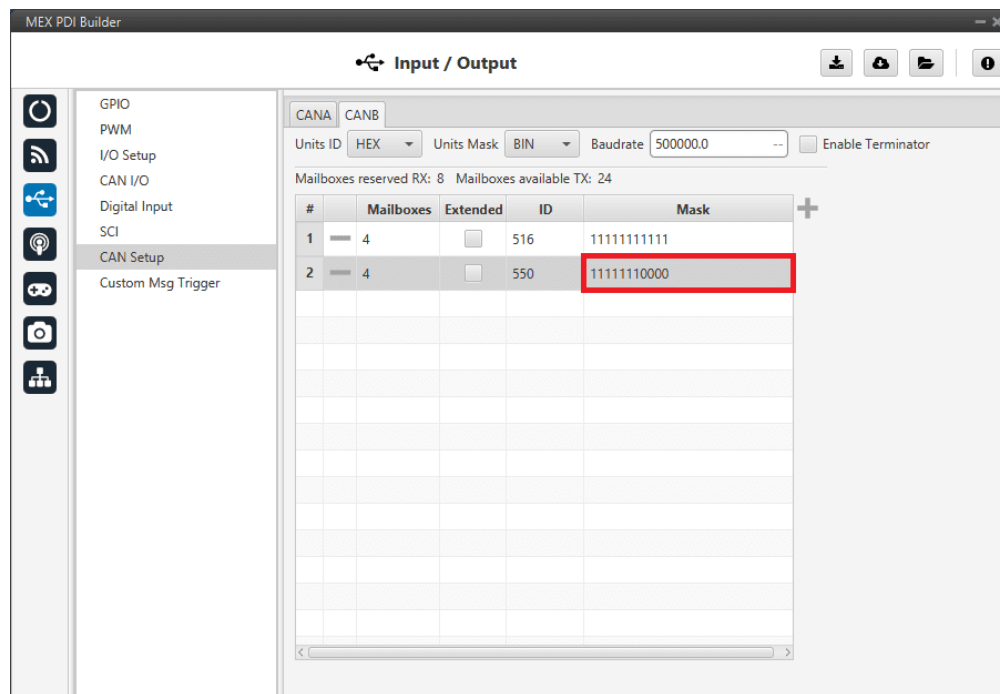
Fig. 61: Sub CAN example diagram

In this example, only a certain range of CAN IDs will be allowed to cross from **Line B** to **Line A**. The allowed range will be from **0x550** to **0x55F**.

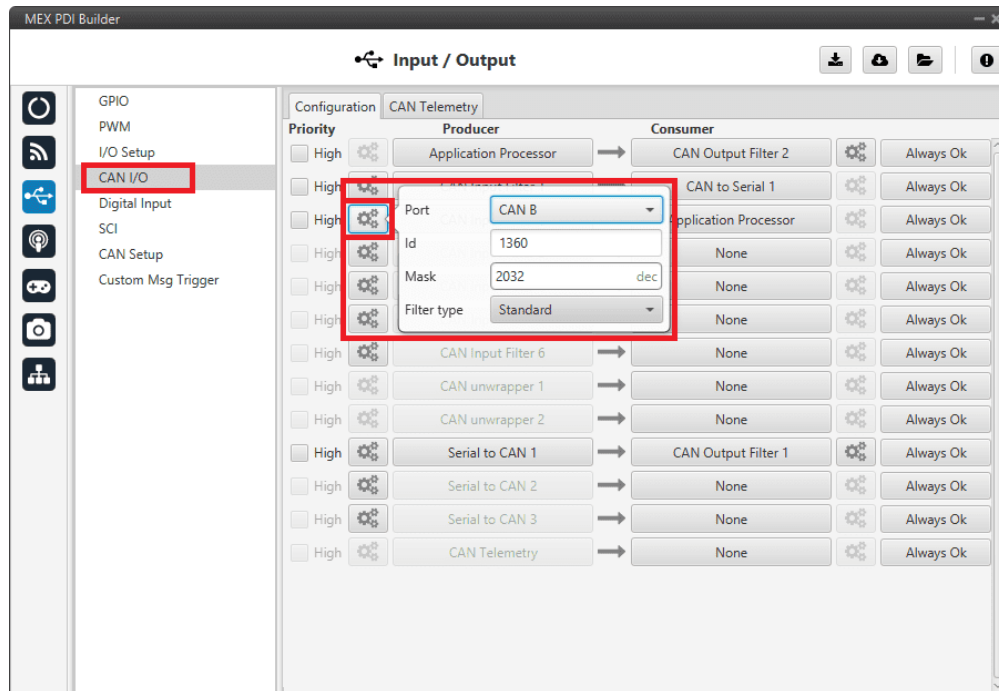
1. Create a new mailbox entry for **Line B**. Assign some of the mailboxes to it and set the ID to **0x550**.



2. Set a Mask which will ignore the last 4 bits.

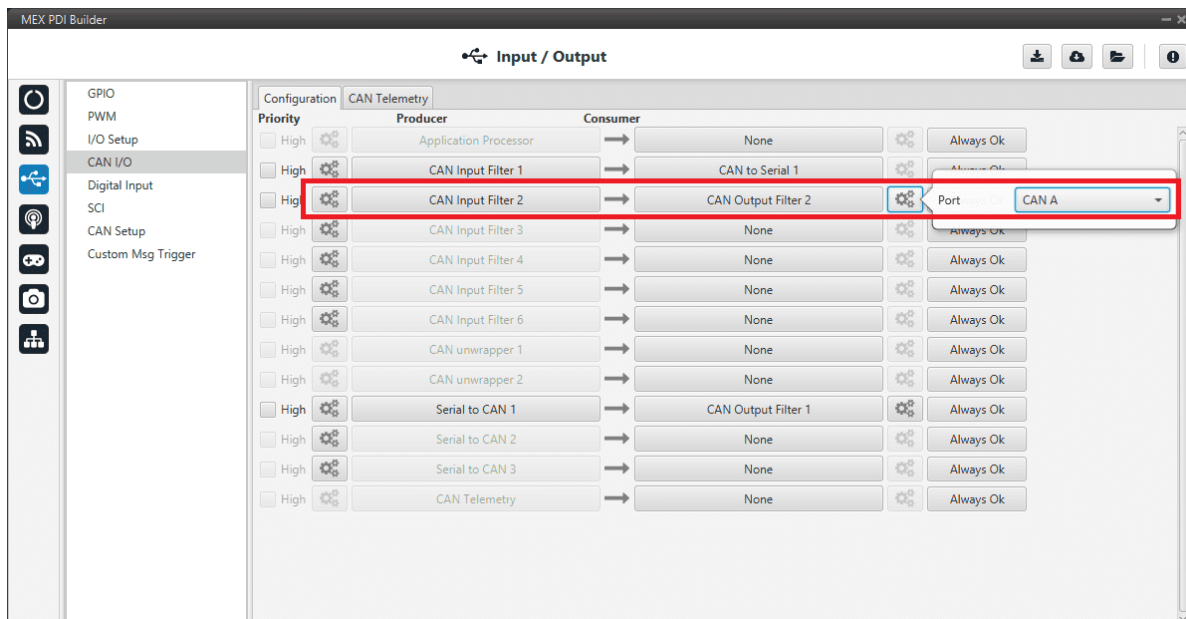


3. Configure **CAN Input Filter 2** on **CAN B**, with the same settings as the **Mailbox**.



Important: This menu only allows decimal numbers. For this example the Id 0x550, is represented as 0d1360.

4. Bind **CAN Output Filter 2** to **CAN Input Filter 2**, configured to **CAN A**.

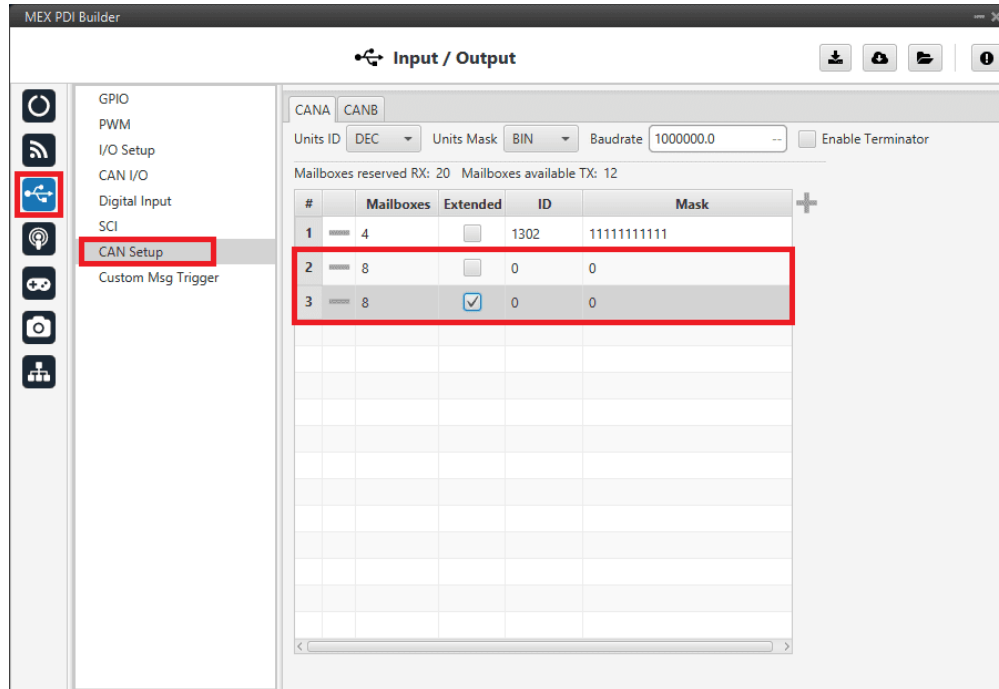


3.11.2 CAN tunnel

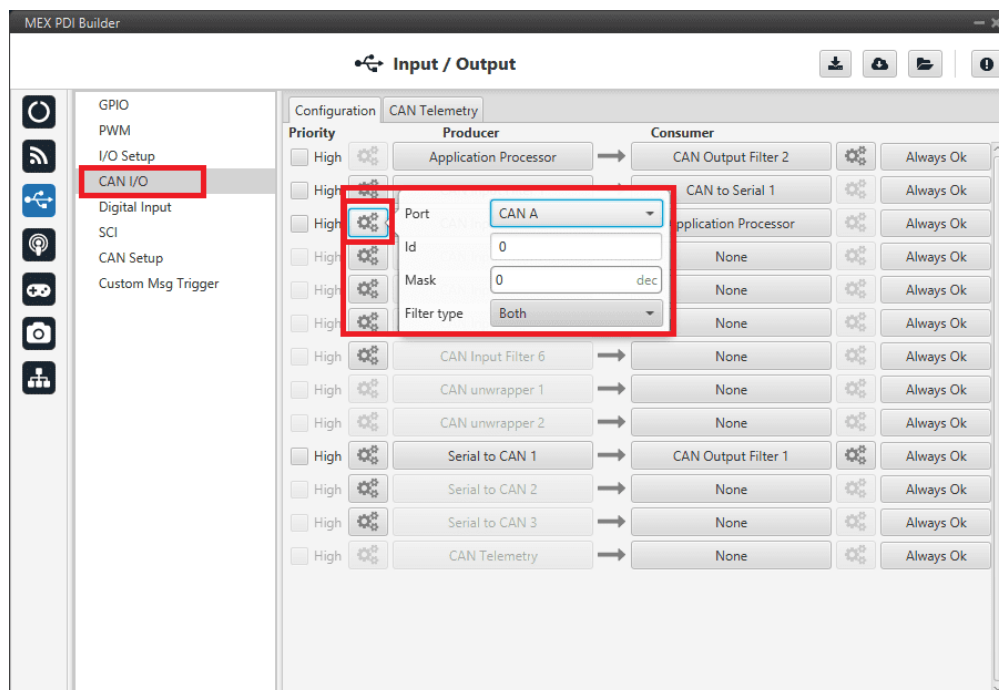
A CAN tunnel is a specific case of *messages tunnel*. The electrical diagram is very similar to *Filtered CAN subnet*.

In this example, a transparent tunnel will be created. So, any messages received on **Interface A** will be sent through **Interface B**. Optionally, the mailboxes can be equally distributed to support both standard and extended CAN IDs.

1. Create a new mailbox entry for **Interface A**. Assign half of the mailboxes to it and set a **Mask** of 0.



2. Configure **CAN Input Filter 2** on **CAN A**, with a **Mask** of 0 and **Both** types.



3. Bind **CAN Output Filter 2** to **CAN Input Filter 2**, configured to **CAN B**.

