CEX PDI Builder

Release 6.12.47

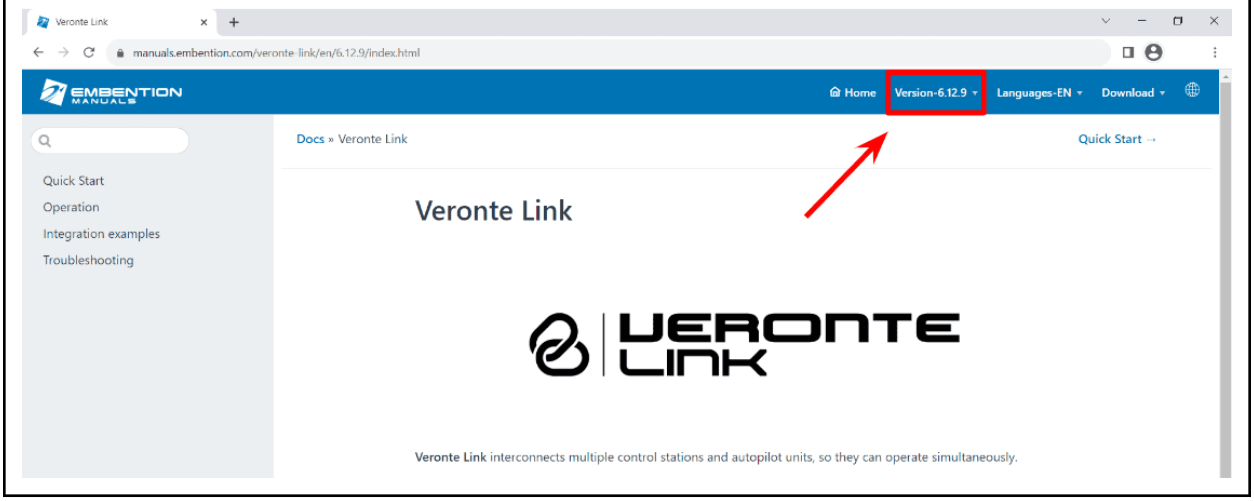
Embention

2024-04-17

CONTENTS

CEX PDI Builder is an application for modifying, generating and uploading CEX PDI files.

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

CEX PDI Builder is the main configuration tool to adapt a **CEX** to be suitable for a specific system, being its main goal to expand the available communication protocols. **CEX PDI Builder** includes:

- Communications: Through general purpose CAN bus, inputs and outputs and PWMs.
- Stick control signal management: Compatible with **Stick Expander**, Futaba, Jeti, FrSky and TBS. It includes custom configuration for other sticks.
- Arbitration: **CEX** is able to send PWM signals using arbitration in the same way **Veronte Autopilot 4x** does.

Once CEX has been detected on Veronte Link, install CEX PDI Builder.

1.1 Download

Once the **CEX** 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 CEX PDI Builder on Windows just execute "CEXPDIBuilder.exe" and follow the *Setup Wizard* instructions.

> **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".

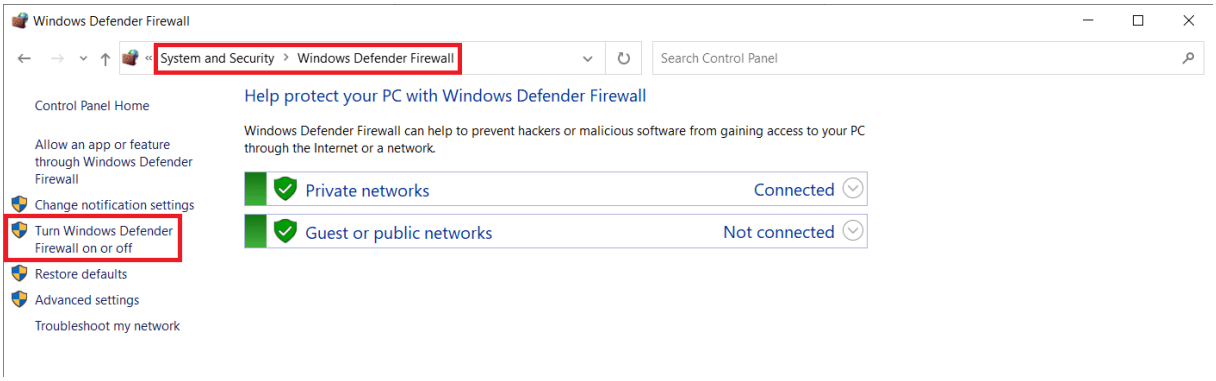


Fig. 1: **Windows Defender Firewall**

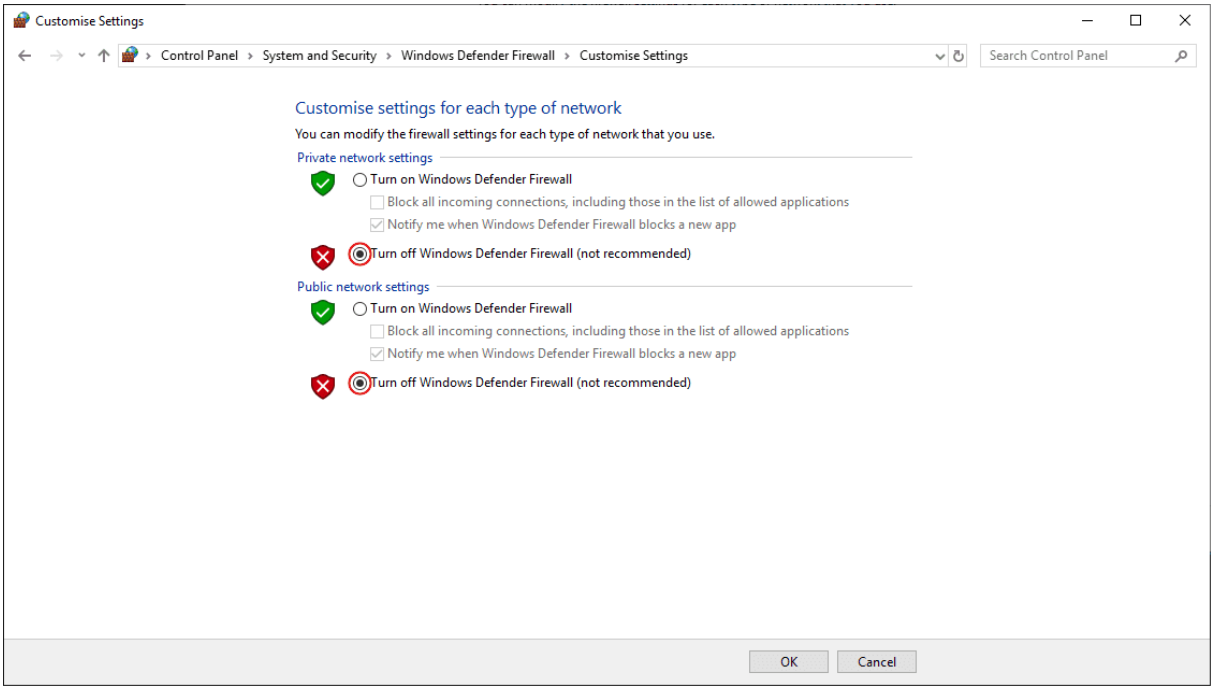


Fig. 2: **Windows Defender Firewall: Settings**

CHAPTER

TWO

CONFIGURATION

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

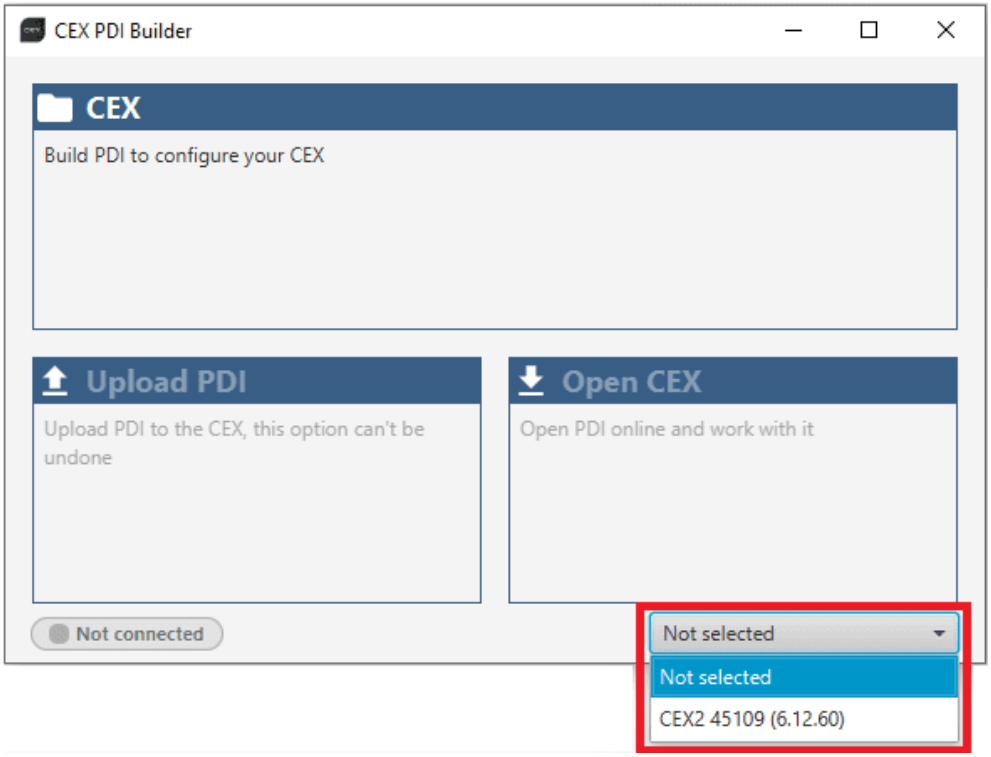


Fig. 1: **CEX ID**

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

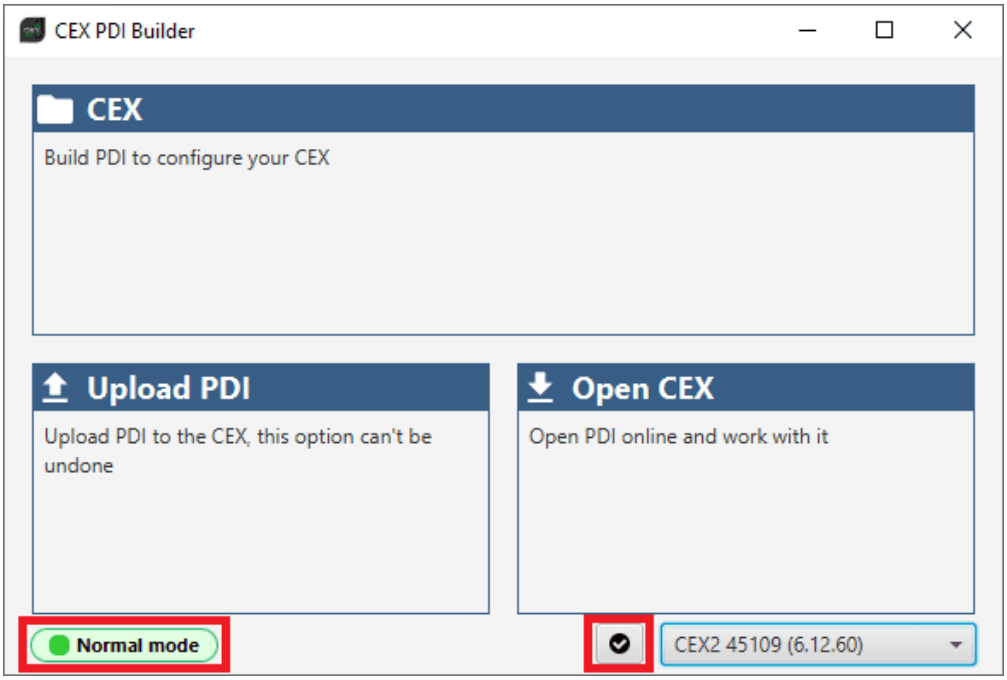


Fig. 2: **CEX PDI Builder**

- **CEX mode**: CEX 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. For more information, see *Maintenance mode (loaded with errors) - Troubleshooting* section of this manual.

-  **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:

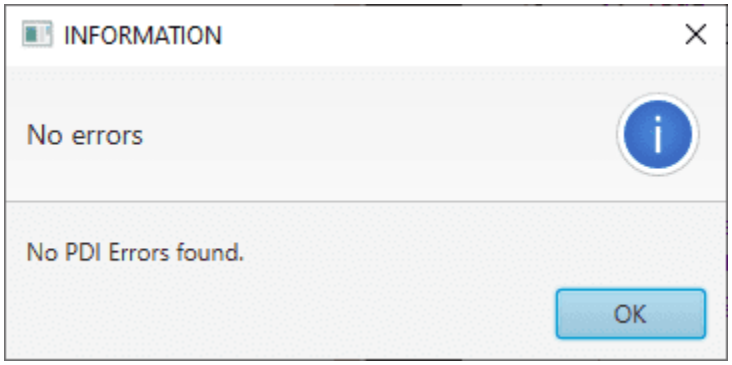


Fig. 3: **CEX PDI Builder - PDI Errors message**

The user can access now to 3 configuration options:

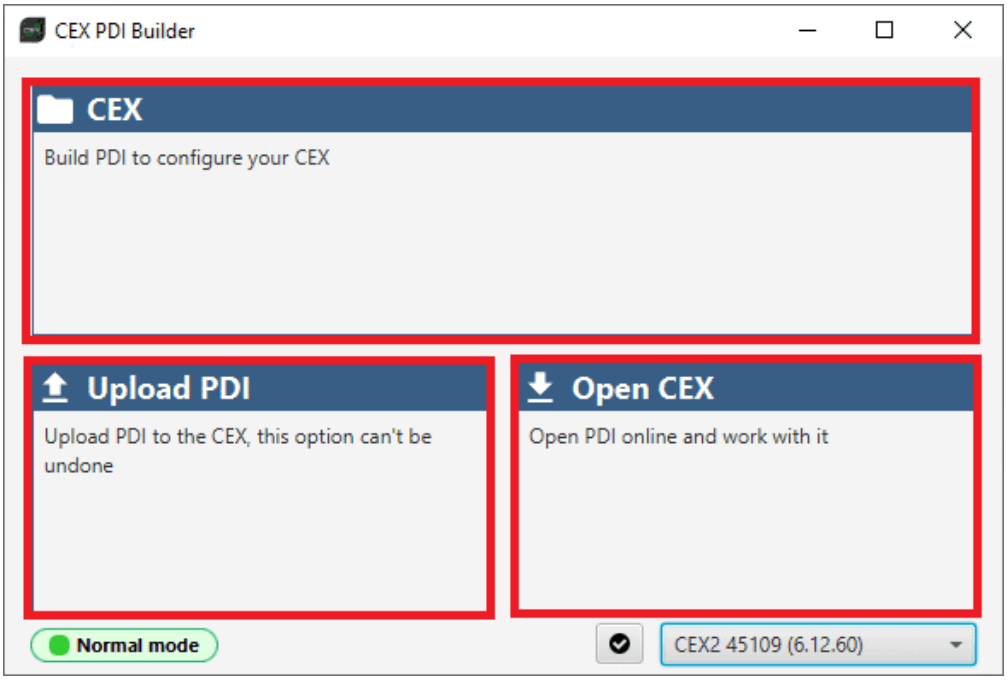


Fig. 4: **CEX PDI Builder Options**

- **CEX**: It allows the user to work with **offline** configurations. A previously exported CEX PDI can be opened and modified or it is possible to build a new one from the default configuration.

Note: When an offline configuration is opened, the user can choose between working with CEX with GPIOs or ARINC.

It is recommended to select the same hardware version that the user's CEX has, so that if the user wishes to load this configuration later to the CEX, there will be no compatibility problems.

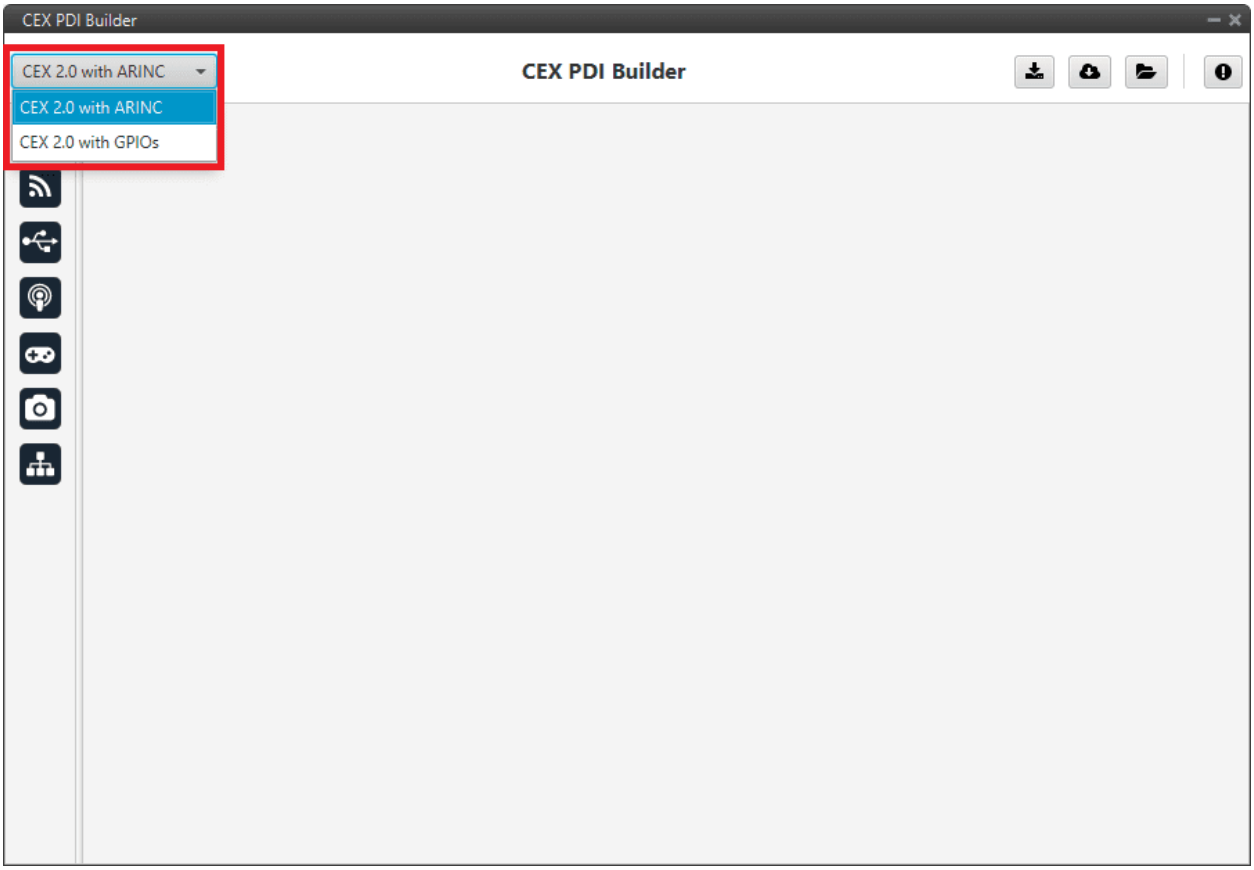


Fig. 5: **Hardware versions**

- **Upload PDI**: A previously exported CEX PDI configuration can be imported to the linked CEX.

> **Warning:** When a configuration is loaded into CEX with a version older than the software version being used, an **automatic migration** from the configuration version to the software version being used will be performed.
>
> For more information on this, consult the *Migrate configuration - Troubleshooting* section of this manual.

- **Open CEX**: By clicking on this option, CEX PDI Builder configuration menu opens with the configuration (the PDI files) loaded in the CEX. Then, the user can modify it **online**.

Note: PDI files are CEX configuration files. These PDI files are stored in the *setup* folder, holding several .xml files which contain all the control system and parameters.

Fig. 6: **PDIs files**

Finally, click on '**CEX**' to edit a configuration offline or '**Open CEX**' to open the configuration and start editing it online.

Note: The CEX unit must enter **Maintenance mode** for the user to start editing, so it is necessary to accept the confirmation panel below.

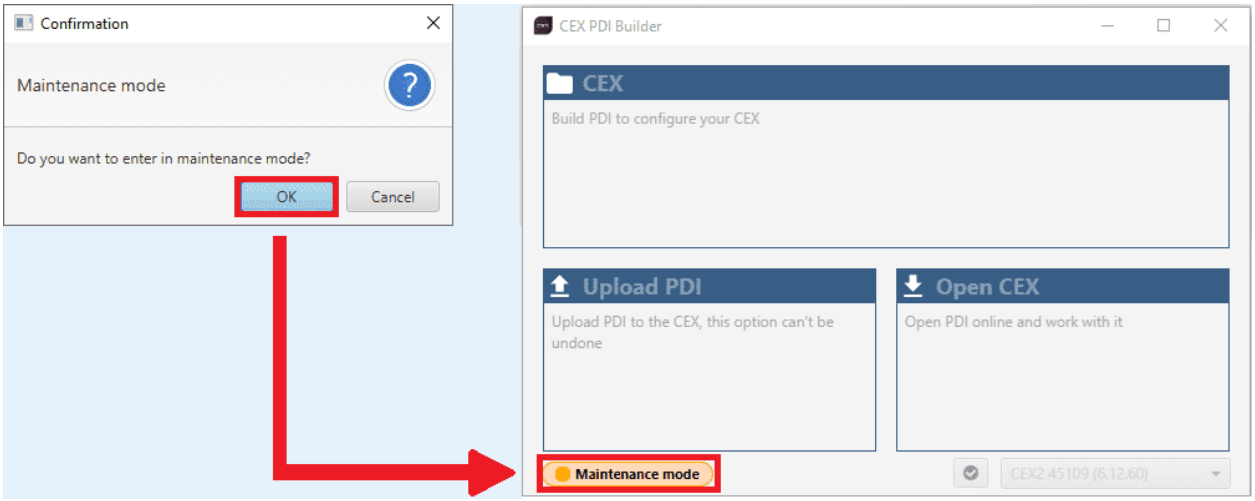


Fig. 7: **Open CEX**

Once in Maintenance mode, the user can access the *Initial menu*. The different 'buttons' that can be seen in this menu of the **CEX PDI Builder** are explained below.

Fig. 8: **Initial menu**

1. **Save and close**: After changes are done, press on the save button to apply the changes in the PDI files. While saving, a percentage of saving process is displayed.

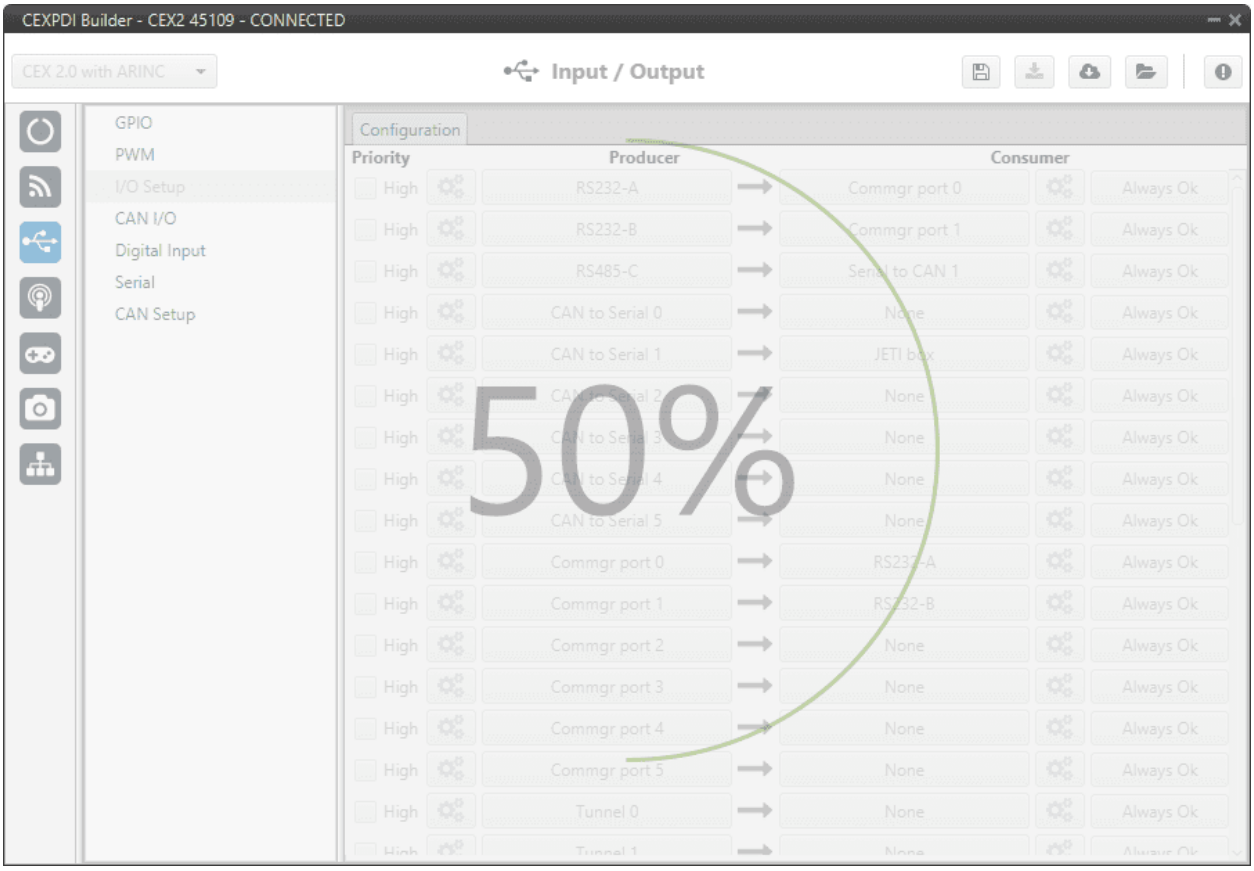


Fig. 9: **Save and close**

In order to save the configuration in the CEX unit it is necessary to **RESET**, therefore the **CEX PDI Builder** software will **close**. For this reason, the user must accept the following panels:

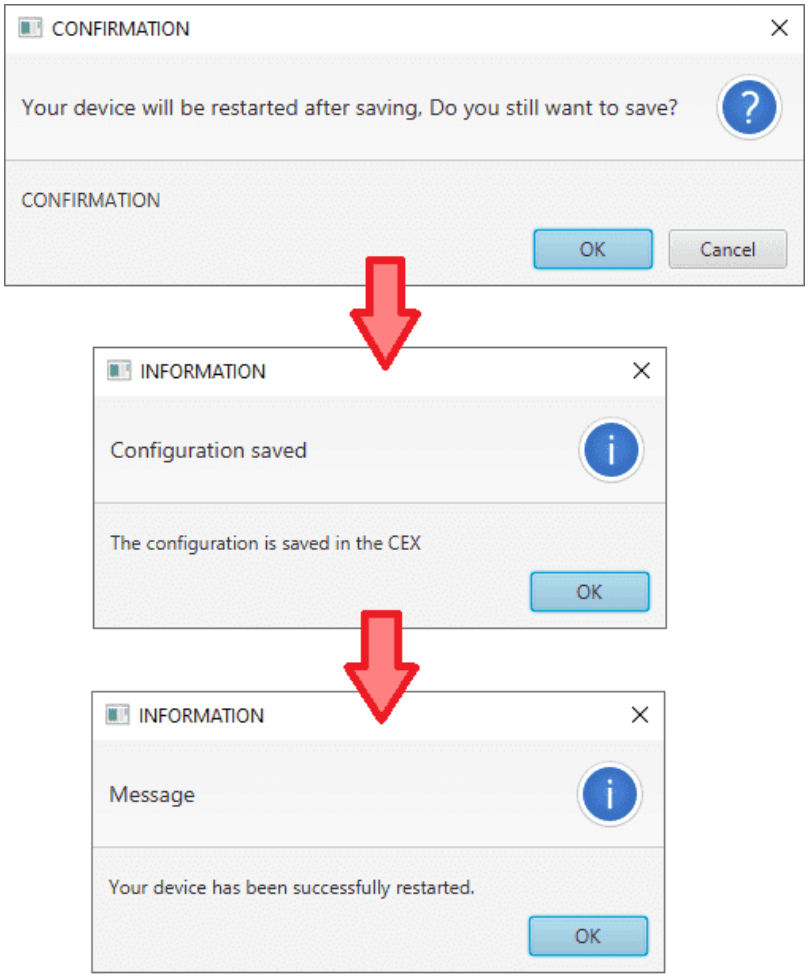


Fig. 10: **Confirmation panel - Save**

Danger: As CEX is **reset**, it is **not advisable to save changes during flight tests**.

Note: This button will only appear if a CEX unit 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 in the CEX.

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 CEX.

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 CEX 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.

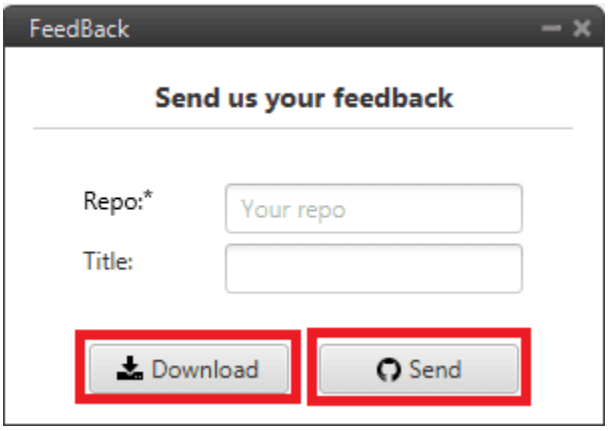


Fig. 11: **Feedback**

6. These are the different functions of CEX. They are explained in the following sections.

-  *CEX*
-  *Sensors*
-  *Input/Output*
-  *Communications*
-  *Stick*
-  *Devices*
-  *Arbitration*

2.1 CEX

2.1.1 CEX base

CEX is able to send information about its version and Jeti telemetry of the devices connected to CEX, this is the “status message”. Users can define the CAN Id used for this message:

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).

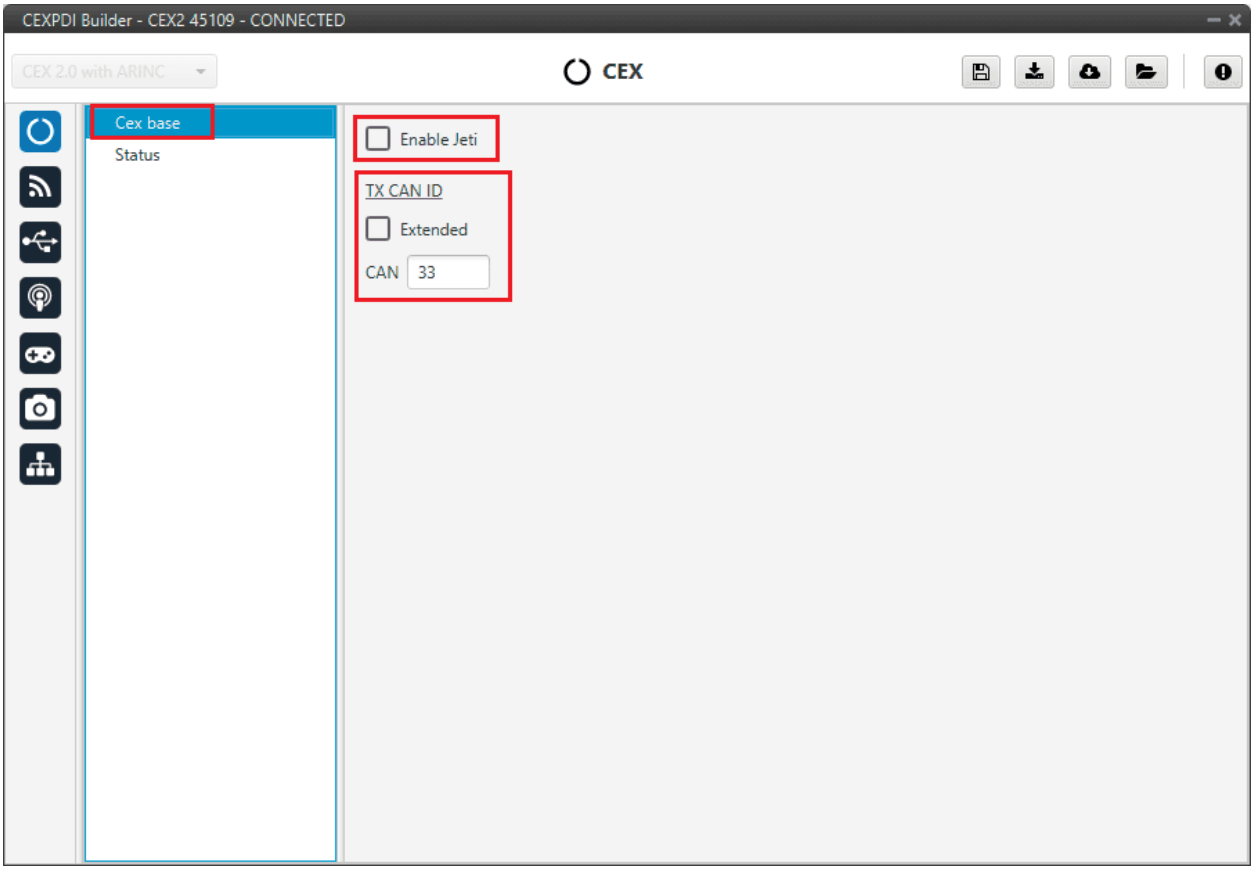


Fig. 12: **CEX base panel**

2.1.2 Status

This option enables the periodic sending of the status message that **Veronte Link** uses to recognize CEX.

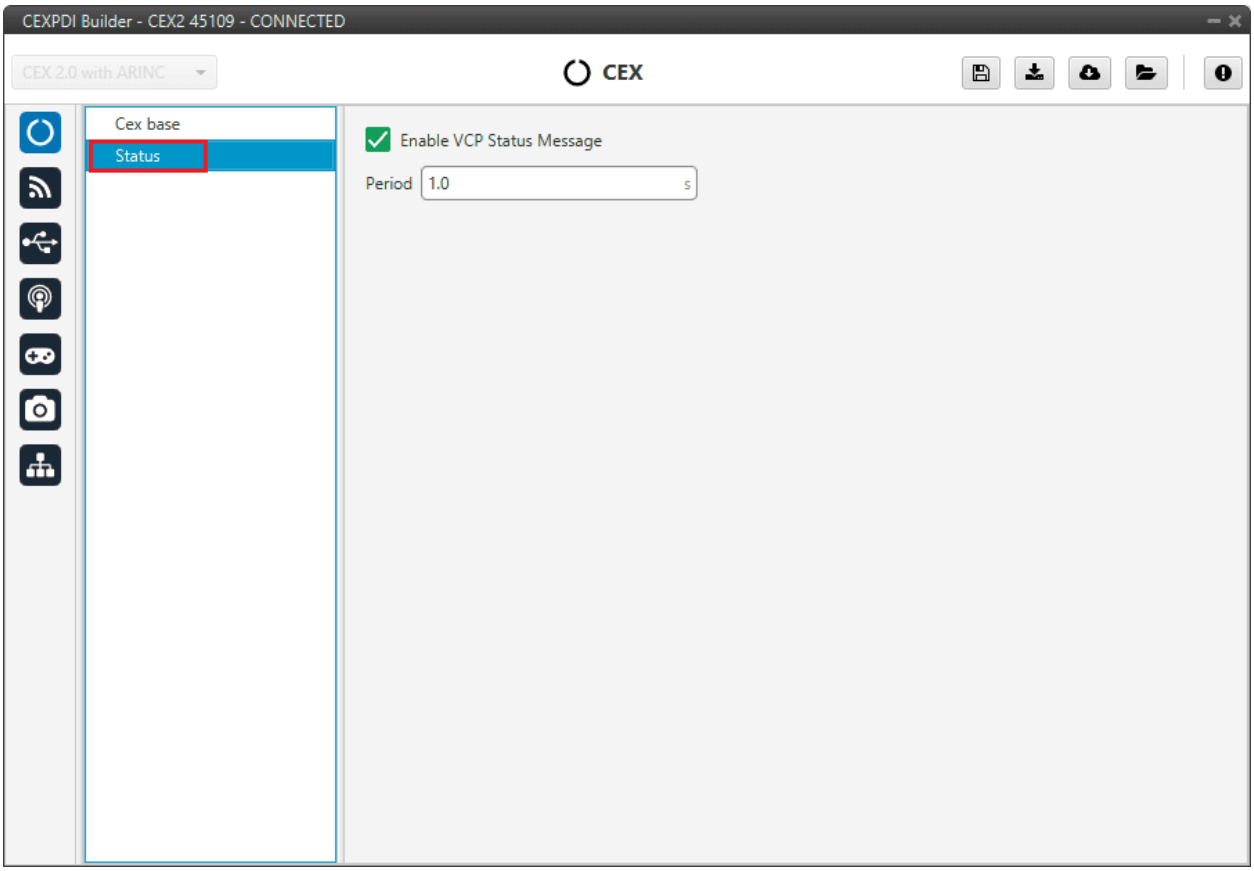


Fig. 13: **Status panel**

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

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

2.2 Sensors

2.2.1 RPM

CEX can measure RPMs by reading from up to four inputs:

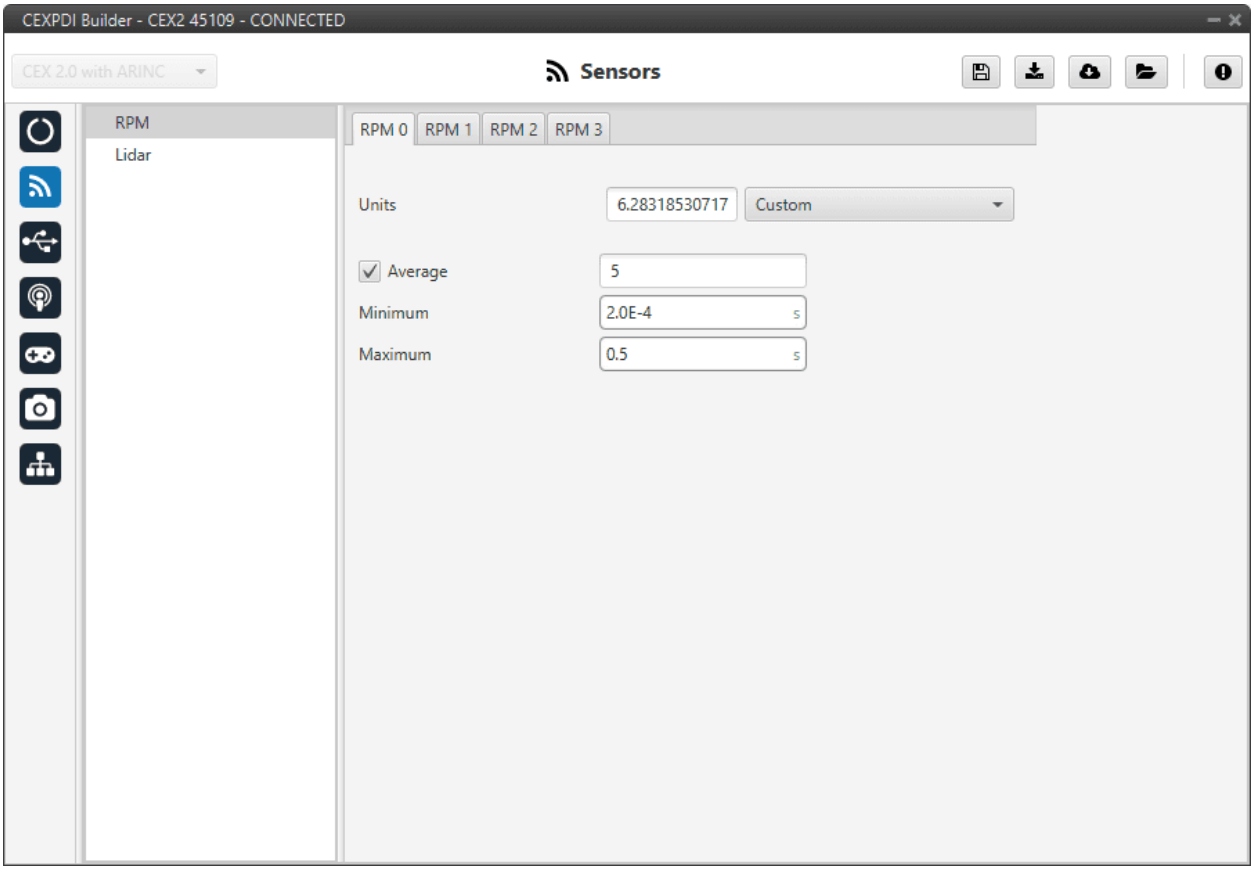


Fig. 14: **RPM panel**

- **Units**: Sensor conversion factor. It can be Custom, Radians per pulse and Pulses per cycle.
- **Average**: It is a filter to avoid voltage spikes. The readout of the pulse can be filtered for the output to be an average. 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 spurius 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.

A configuration example to send RPM can be found in *Reading/Sending RPMs - Integration examples* section of the present manual.

2.2.2 Lidar

The I2C bus allows the connection of several devices with different addresses to the same line via master-slave communication. At this moment, CEX supports the following Lidar devices:

- **Garmin LIDAR-Lite v3**: Optical distance measurement sensor with a range from 5 cm to 40 m.
- **SF11 Lidar**: Long range laser altimeter. Supported SF11/B and SF11/C with a range of maximum 50 m and 0.2 m to 120 m respectively.
- **SF20 Lidar**: OEM laser altimeter module. Supported SF20/C with a range of 0.2 m to 100 m.

CEX allows up to 5 Lidar devices to be connected to the system at the same time. The configuration menu can be seen below:

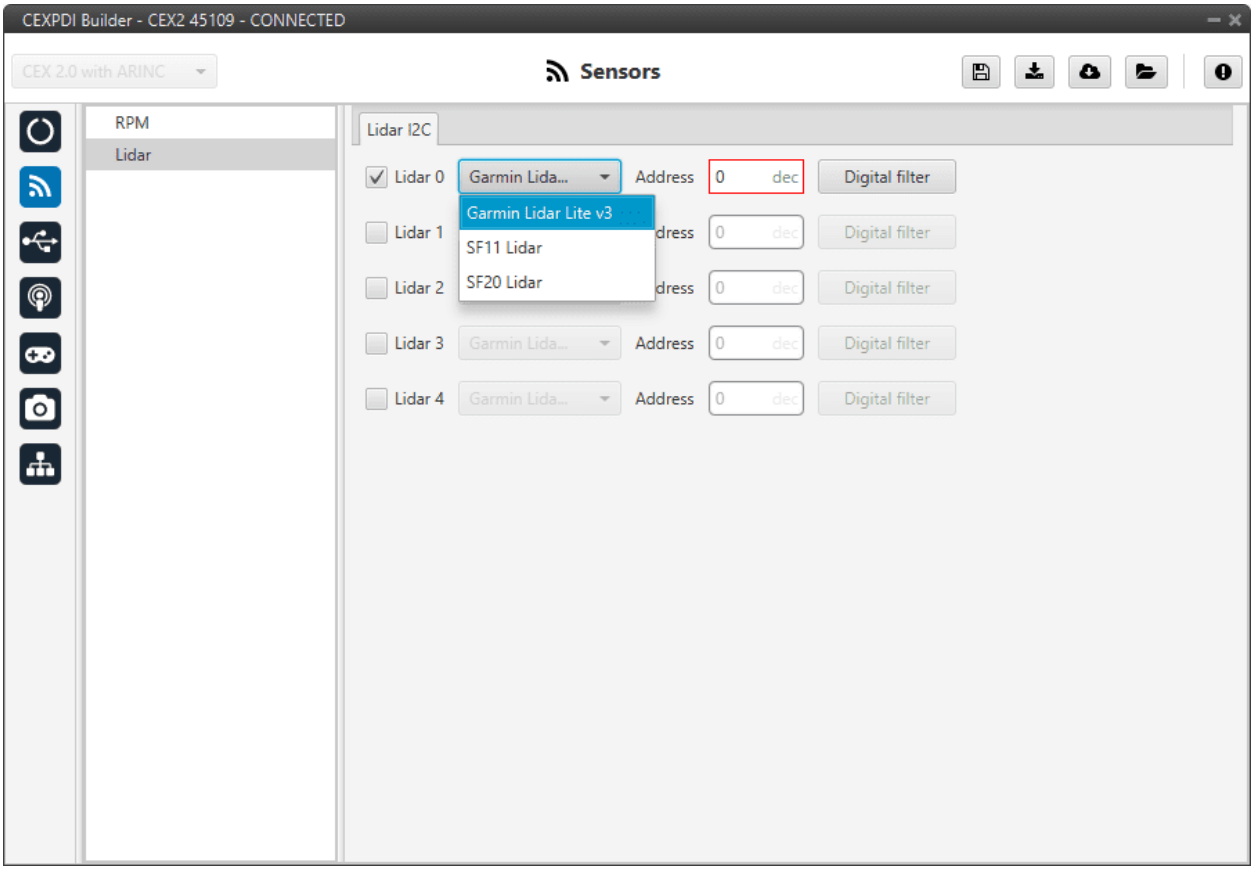


Fig. 15: **Lidar devices**

After enabling the needed number of Lidar devices, configurable parameters are:

- **Type of Lidar**.
- **Address**: With an accepted value between 16 - 239, this is the origin address from the Lidar being configured.
- **Digital filter**: Enables a low pass filter whose cutoff frequency is manually configured, allowing the user to input any desired value in Hz. It is a **software filter**.

Note: I2C address will be different for different devices, make sure to define it properly by checking the manufacturer documentation.

2.3 Input/Output

2.3.1 GPIO

In this tab users can configure each individual GPIO behavior:

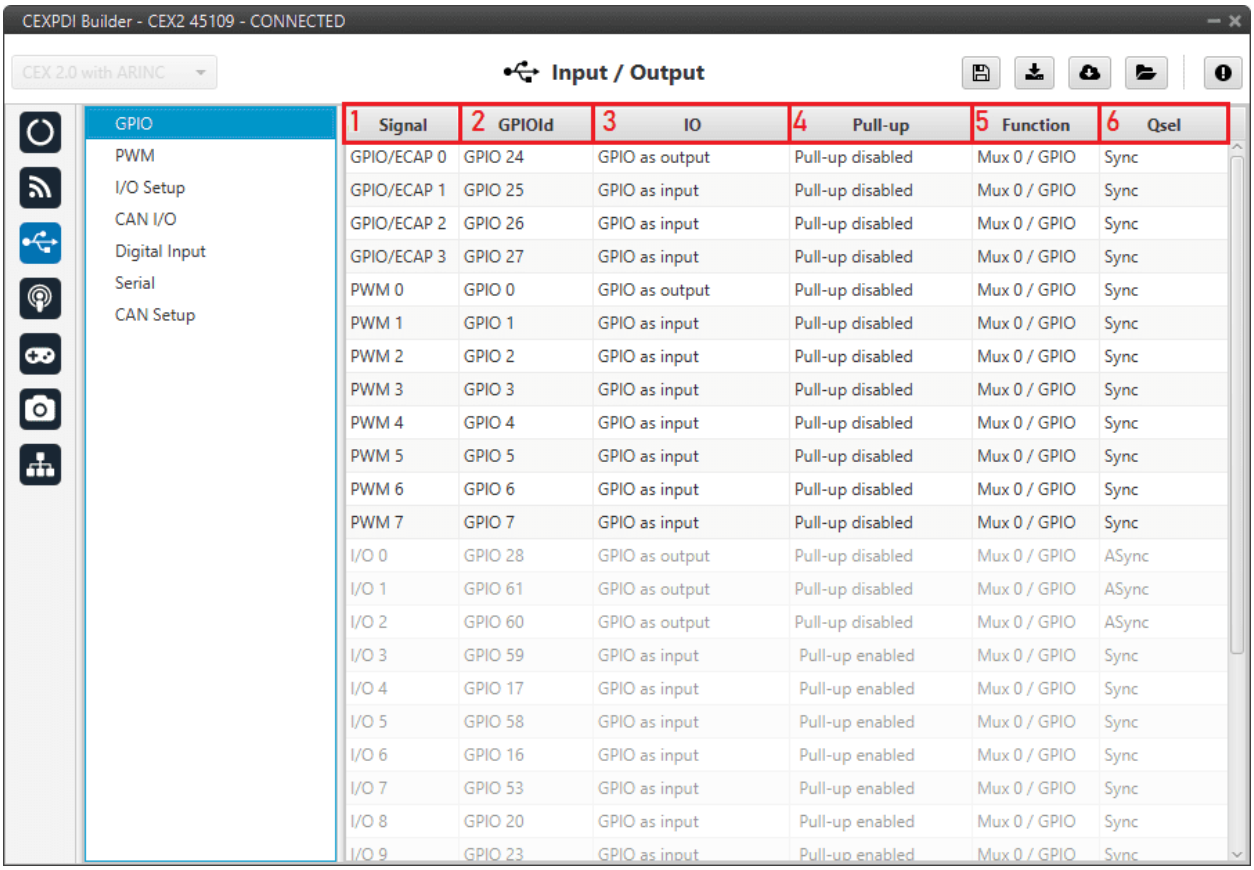


Fig. 16: **GPIO panel**

1. **Signal**: Pin ID as described in Hardware installation - Pinout section of the **CEX Hardware Manual**.
2. **GPIOId**: GPIO ID of the microcontroller.
3. **IO**: Define GPIO as an input or ouput.
4. **Pull-up**: Enable or disable the pull-up resistance.
5. **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.

Note: When users set **Function** to "Mux 1", it indicates that the corresponding pin is disabled as GPIO and enabled as PWM. Consequently, the **Enable** checkbox in the *PWM menu* for that pin should be activated automatically.

Warning: Check that the corresponding **Enable** checkbox in the *PWM menu* is changed.

6. **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 whatever is present at 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 before, 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 panel, each PWM can be configured:

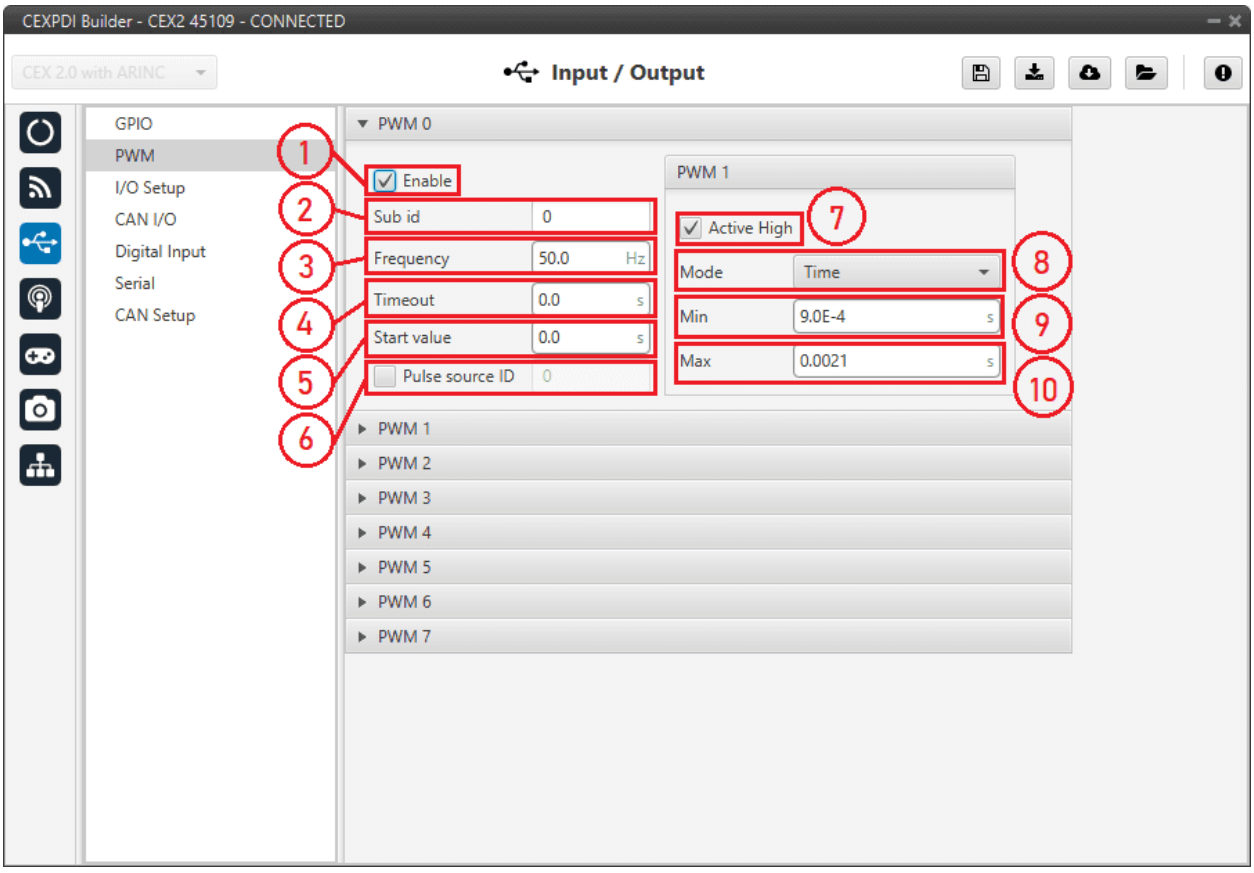


Fig. 17: **PWM panel**

Note: PWMs in CEX work in normalized mode, so 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.

The configuration parameters are:

1. **Enable**: Define if the PWM is enabled or not.

   Note: This checkbox disables the pin as GPIO and enables it as PWM. Hence, in the *GPIO menu* the "Function" parameter shall change to "Mux 1".

2. **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.
3. **Frequency**: PWM output frequency.
4. **Timeout**: If a PWM message is not received in less than this time, the PWM will output the start value.
5. **Start value**: Value used before any PWM message arrives and on timeout.
6. **Pulse source ID**: PWM input ID [0,3], defined in the *Digital Input panel*.
7. **Active High**: Polarity high or low (high if enabled).
8. **Mode**: The available options are **Time** and **Duty cycle**.
   - **Time**: The values indicated in **Min** and **Max** parameters are expressed in time units.

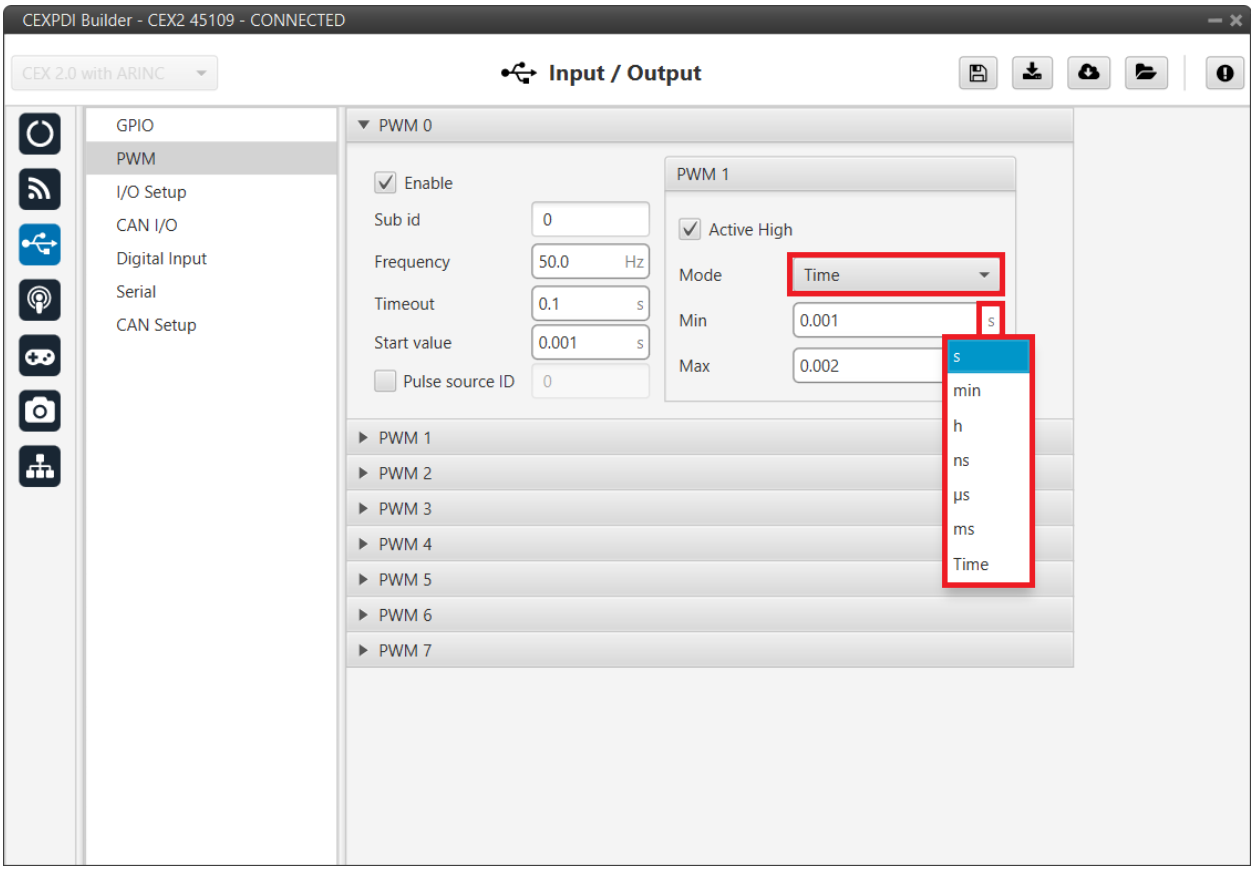


Fig. 18: **Time units**

- **Duty cycle**: This option is a a different way of indicating the pulse width. Now the value indicated in **Min** and **Max** parameters is a percentage which corresponds to the relation between the pulse width over the total period of the sent signal.

  So a 100% duty cycle will correspond to a signal with a constant value of 1, while a 0% duty cycle implies a constant signal with value 0. Between this two extremes, the pulse width can vary as in the examples shown in the following figure.

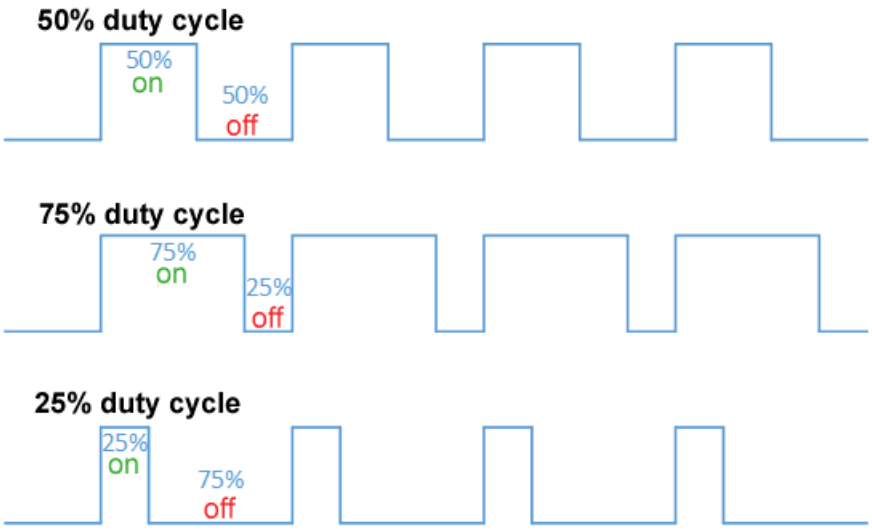


Fig. 19: **Duty cycle**

Note: Duty cycle percentages can be expressed in percent and per unit.

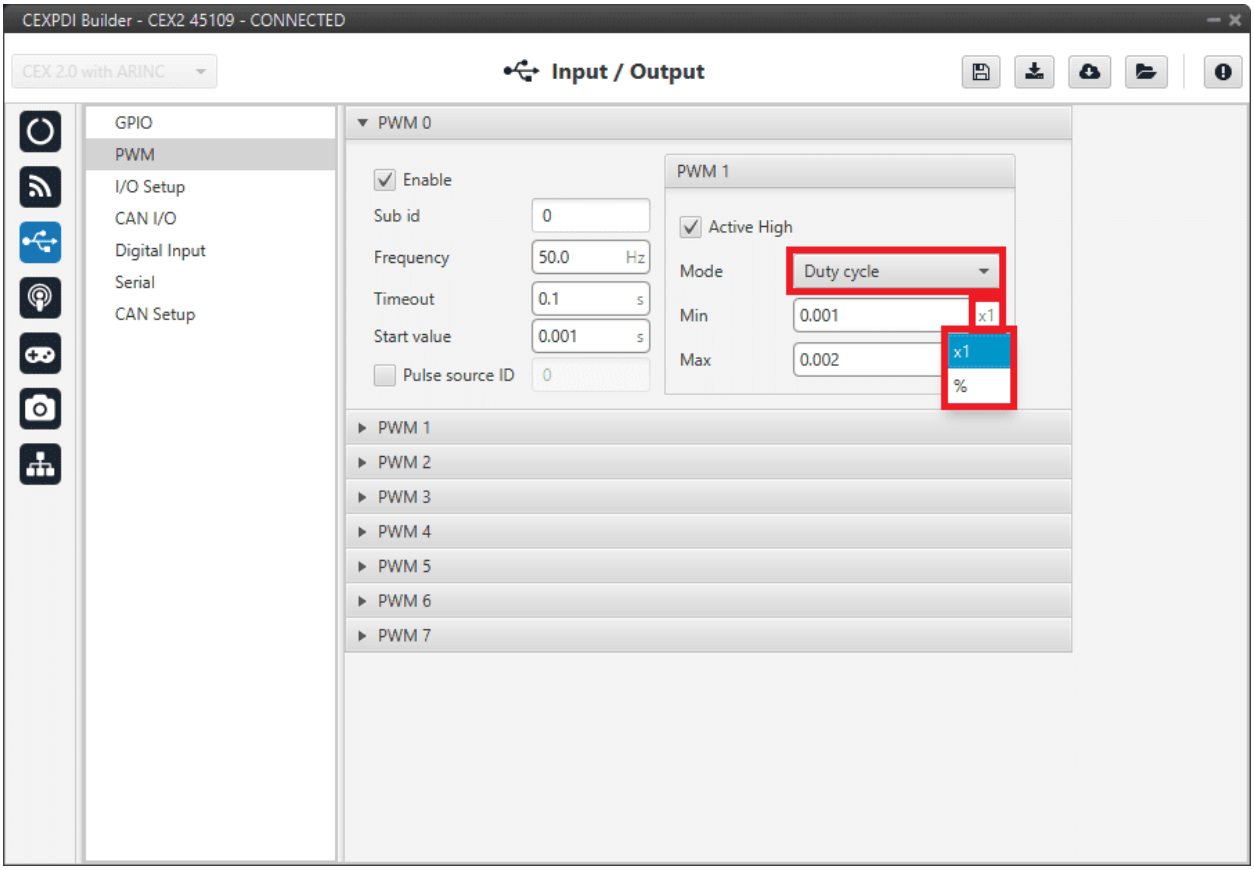


Fig. 20: **Duty cycle units**

9. **Min**: This parameter is the pulse width value that will make the servo/actuator go to its **lowest position**.

   It will be the output when the PWM message specifies **0**.

10. **Max**: This parameter is the pulse width value that will make the servo/actuator go to its **highest position**. It will be the output when the PWM message specifies **4095**.

An example of reading PWM can be found in *Commanding/Reading PWMs - Integration example* section of the present manual.

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 serial inputs and outputs: external sensors, custom messages, etc.

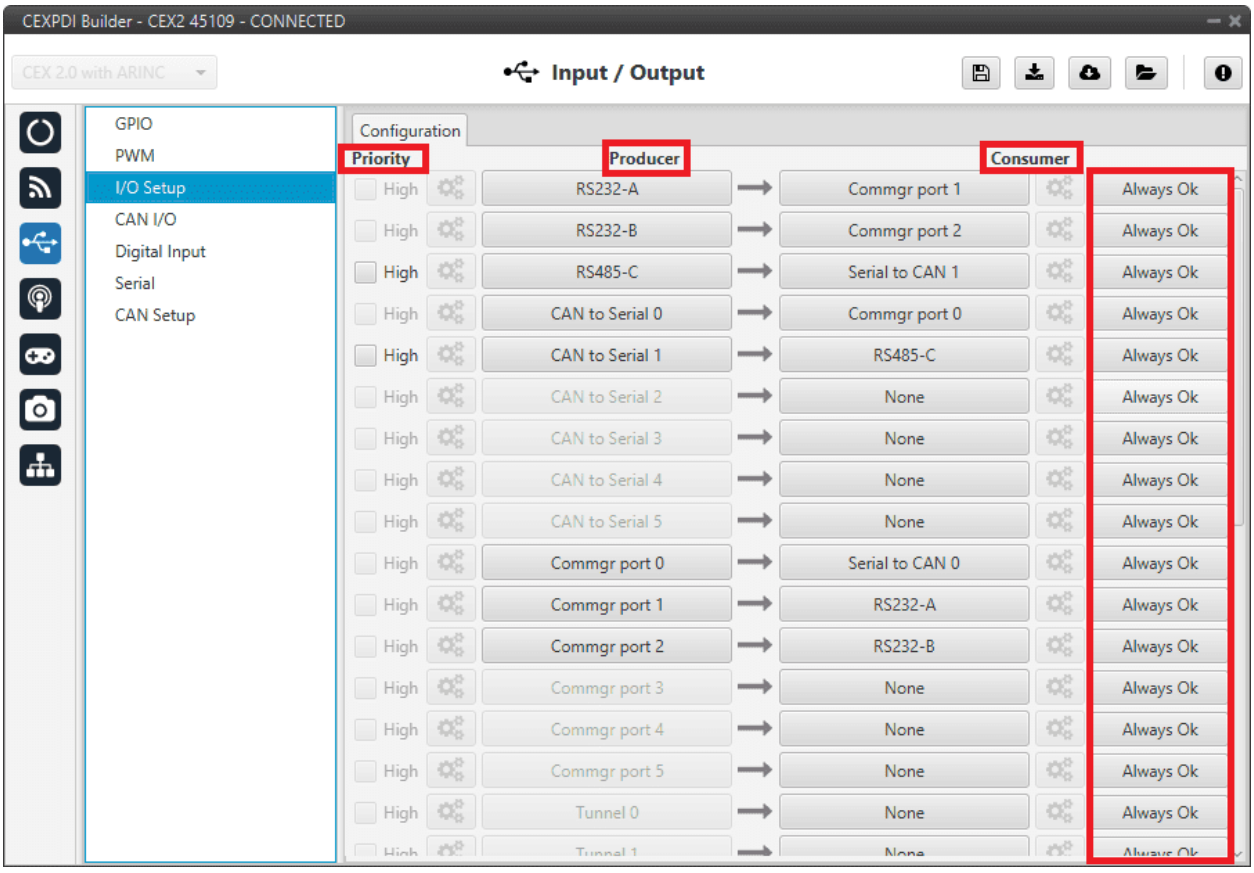


Fig. 21: **I/O Setup panel**

- **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 to a bit. Thus, the connection is 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 following are the steps to setting up reception or transmission between ports:

1. Choose the **Producer** to use.
2. To configure the desired **Consumer** that will be bind to the chosen **Producer**, it is first required to establish the **relationship** between them:
    - **Bind** $\rightarrow$: Unidirectional relationship.
    - **Bind Bidirectional** $\leftrightarrow$: Bidirectional relationship. This enables a port to receive or send information.

Note: Once the **Consumer** has been selected, it is possible to undo the selection by pressing the **Clear** button.

3. Select the desired **Consumer** element.
4. *(optional)* Configure the **Bit** parameter. By pressing on the bit button, the user can select the bit to assign the connection to.

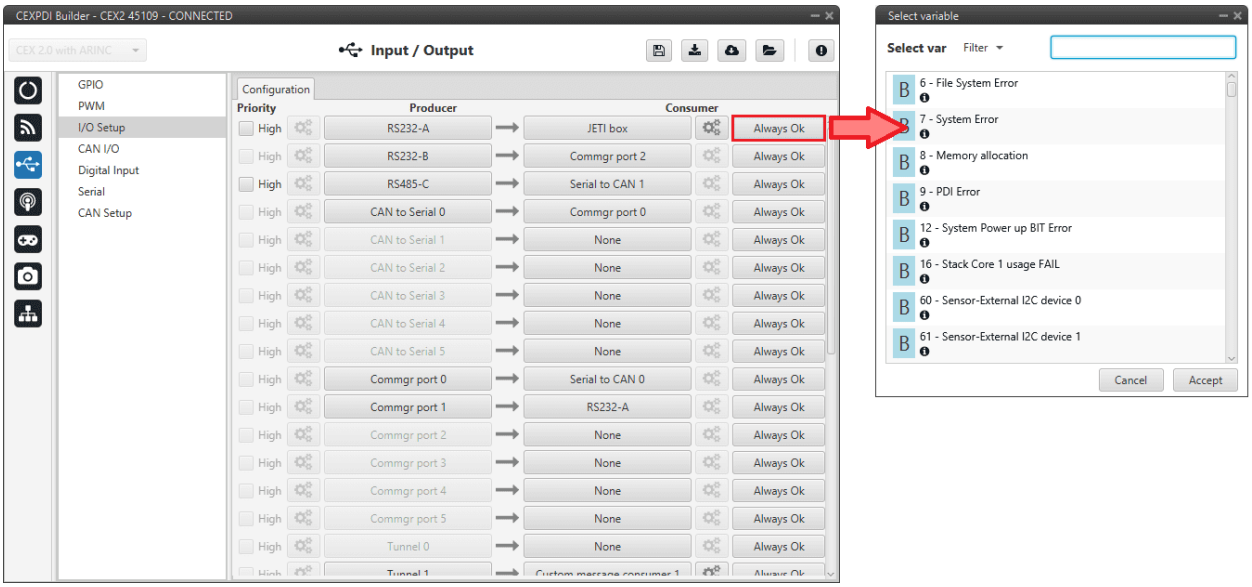


Fig. 22: **Bit - Select a variable**

The following I/O ports are available:

Field	Description
RS232-A	Serial Port 232 A
RS232-B	Serial Port 232 B
RS485-C	Serial Port 485
Commgr port	COM Manager ports send and receive VCP messages. This is the protocol used by Veronte products to communicate. For more information, read the VCP user manual
Tunnel	Creates a bidirectional brigde between two devices, see *Tunnel*
Custom message producer/consumer	This allows user to send/receive a serial custom message, see *Custom message*
Y splitter producer/consumer	Used to split a signal into 2
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 for serial transmission** sends CAN streams over a serial Bus / **Serial CAN unwrapper** undoes this transformation. For more information on these ports, please refer to CAN wrapper/CAN unwrapper - Input/Output section of the **1x PDI Builder** user manual
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
ARINC producer channel/consumer	This allows user to send/receive ARINC 429 messages through HI-3210, see *ARINC communication protocol*

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

2.3.3.1 Tunnel

It is possible to configure a Tunnel which is a bidirectional bridge between units that communicate to each other sharing information about an external device connected to the Serial or Digital port.

Let's consider the following image.

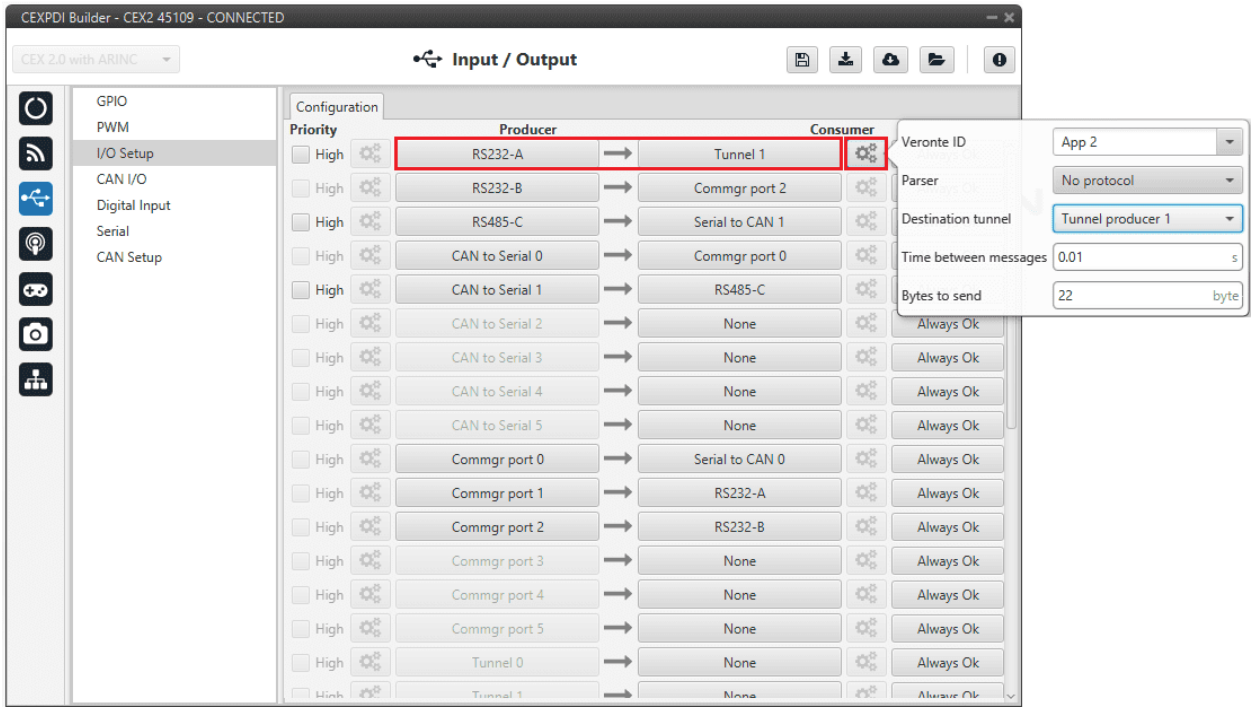


Fig. 23: **Tunnel configuration**

In the image above there is a device connected to the **RS232-A (Producer)** and there is a **Tunnel (Consumer)** which sends that information to Veronte applications (App 2). On the other hand, a Veronte application, for example Veronte Ops, will receive the signal sent by this CEX unit.

The option available when configuring Tunnel as **Consumer** are:

- **Veronte ID**: Enter the address that will receive the information. The following options are the most common:
  - **App 2**: Veronte applications address.
  - **Broadcast**: All units on the network. Select this option for a generic configuration.
  - Address of a specific Veronte unit, it can be a 1x, a 4x, a CEX, etc.

    For more information on the available addresses, see List of addresses section of the **CEX Software Manual**.
- **Parser**: The user can choose protocol to parse message data. The options available are:
  - No protocol
  - RTCM3
  - CANserial
- **Destination tunnel**: Number of port is used to avoid mistakes and identify a 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 Custom Messages

Warning:

- CEX has a **serial limitation** shared with all **Custom Messages**:
  - Maximum number of **vectors** (fieldset): **100**
  - Maximum number of **fields**: **1000**
- Each Custom message **consumer** has a limit of 32 fields.

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

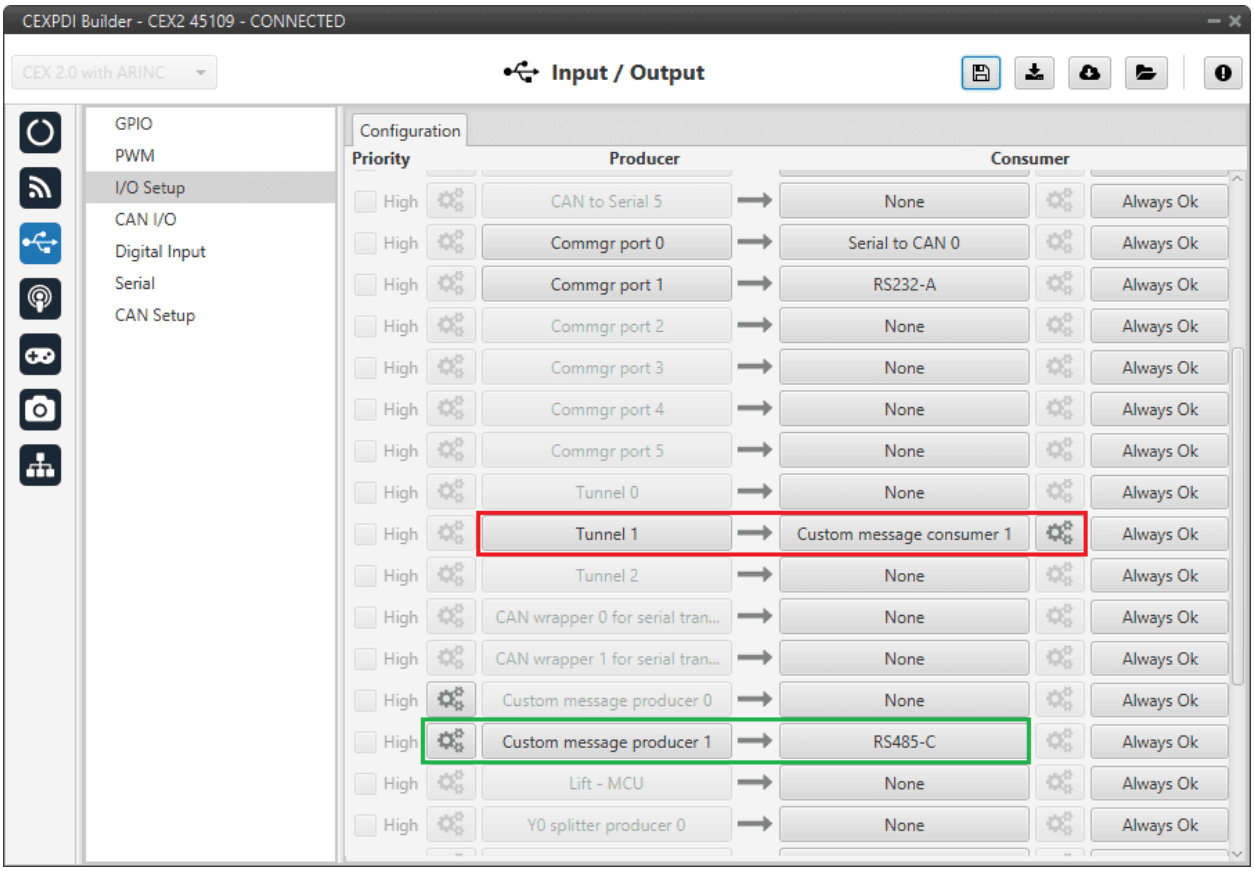


Fig. 24: **Serial Custom Messages**

In the image above can be seen two possible configurations using a Custom Message. The 'red' one is configured to receive a determined message from Tunnel 1 and the 'green' is used to send a Custom Message through a RS-485 serial port. It is also possible to use the same Custom Message for both tasks if the bidirectional relationship is selected (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.

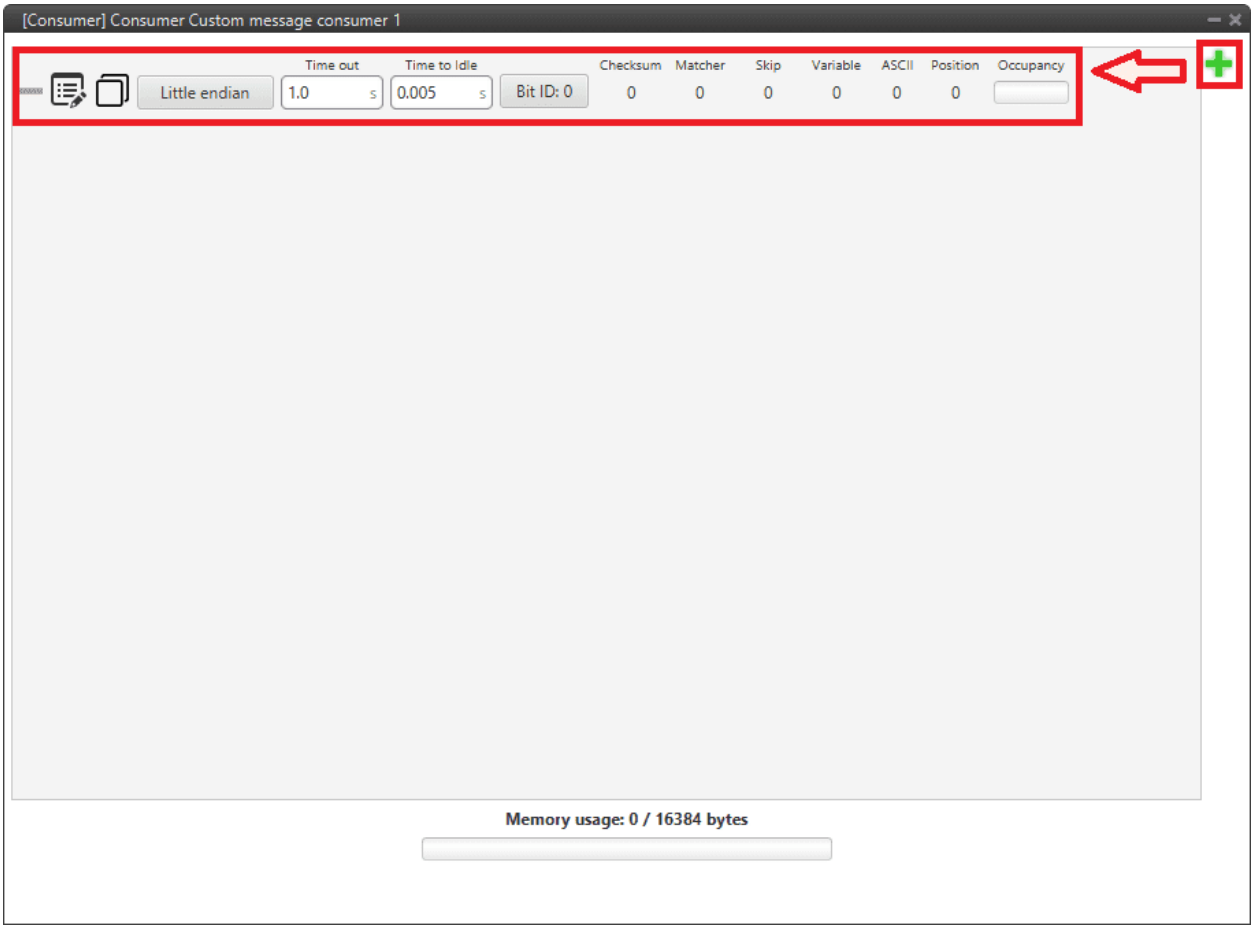


Fig. 25: **Serial Custom Message configuration**

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

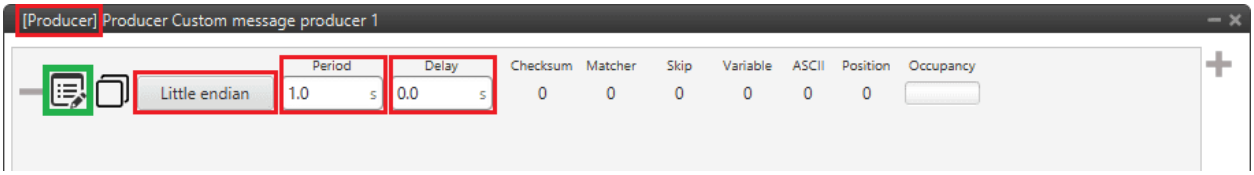


Fig. 26: **Custom Message producer configuration**

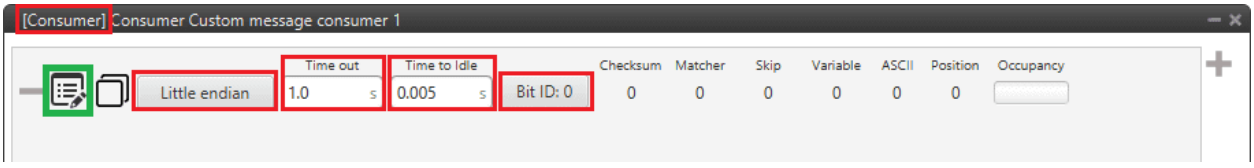


Fig. 27: **Custom Message consumer configuration**

- **Endianness**: Depending on the order in which the device issue 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/Time out**: This option has a dual role depending on if it is used to transmit or receive data.
  - **Period - Producer**: It is the inverse of the send frequency.
  - **Time out - Consumer**: This is the threshold time between receptions to consider that the message is not being received correctly.
- **Delay/Time to Idle**: This option has a dual role depending on if it is used to transmit or receive data.
  - **Delay - Producer**: It is a delay applied before sending the message. This serves to send messages with the same period without overloading the Serial bus.
  - **Time to Idle - Consumer**: This is the time CEX 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 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 the Custom Messages types - Input/Output section of the **1x PDI Builder 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.

2.3.3.3 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 in the JETIBOX Compatible product. With the use of **four buttons**, the user can browse the menu and set the selected values to take advantage of the full capability of JETIBOX Compatible products.

Fig. 28: **JETI box device**

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

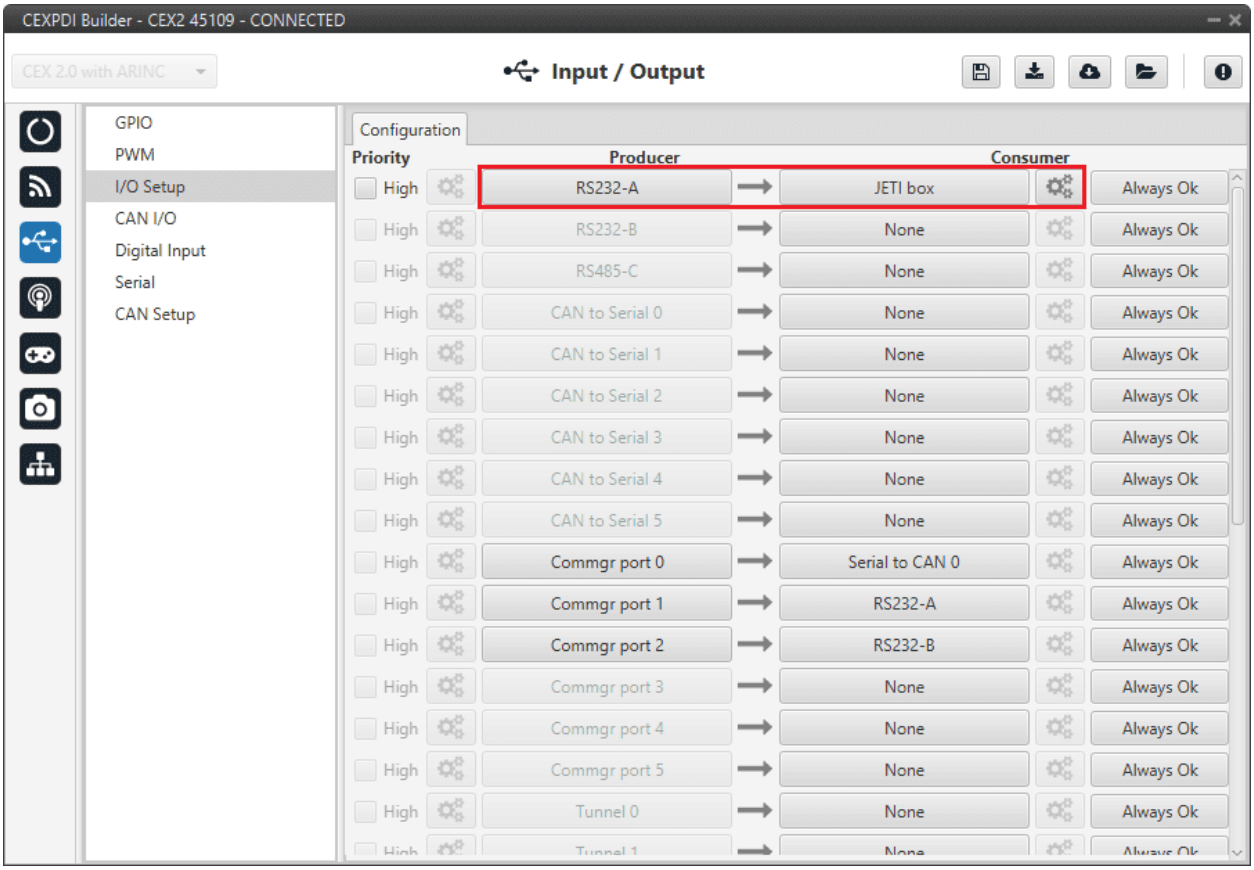


Fig. 29: **JETI box Consumer**

Then, the sequence to retrieve the data shall be configured by clicking on the configuration button ( icon):

- To add a Custom Message, just click on the "**+**" icon:

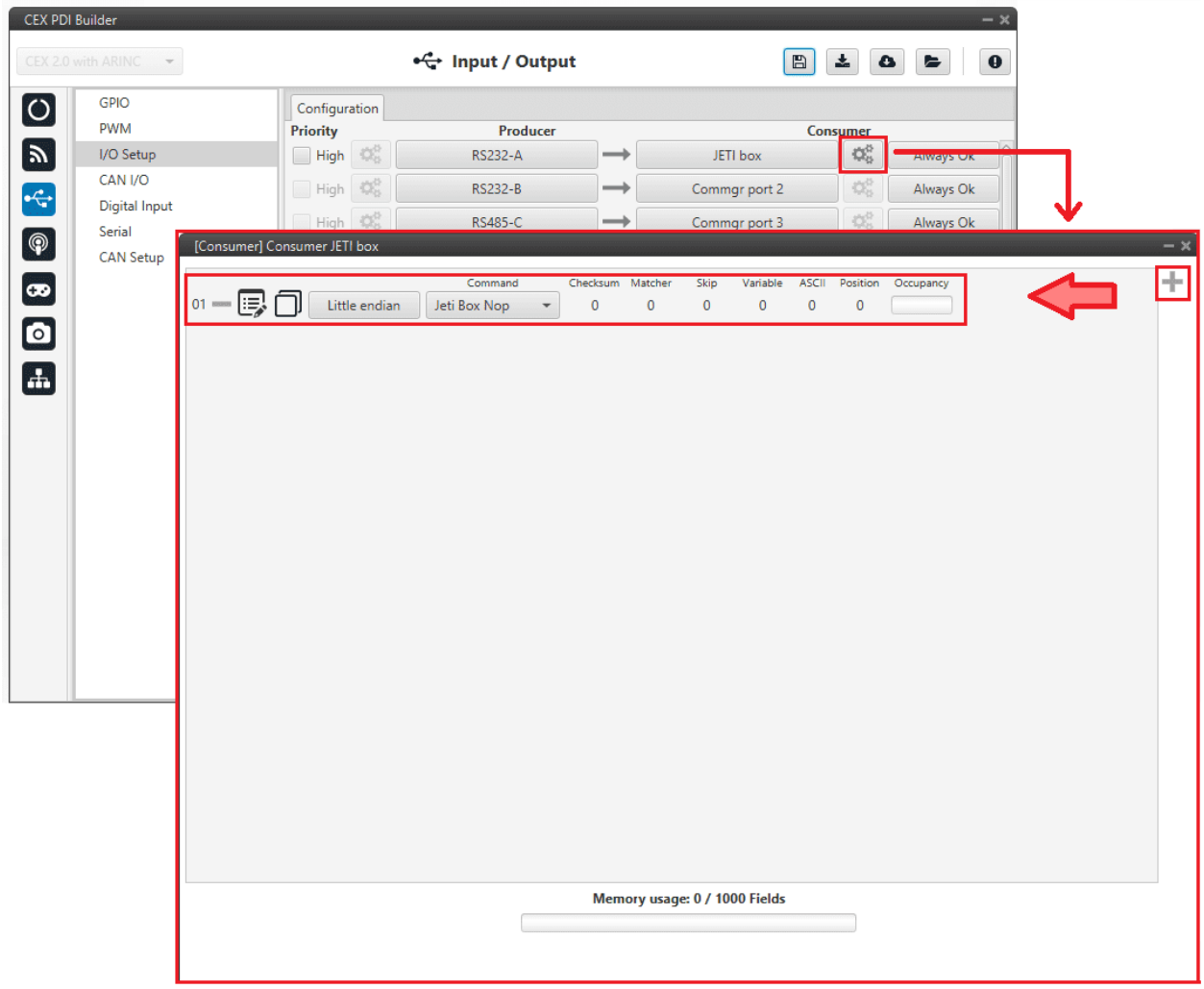


Fig. 30: **JETI box Configuration**

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

- **Endianness**: Depending on the order in which the device issue 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).
- **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** in all messages).

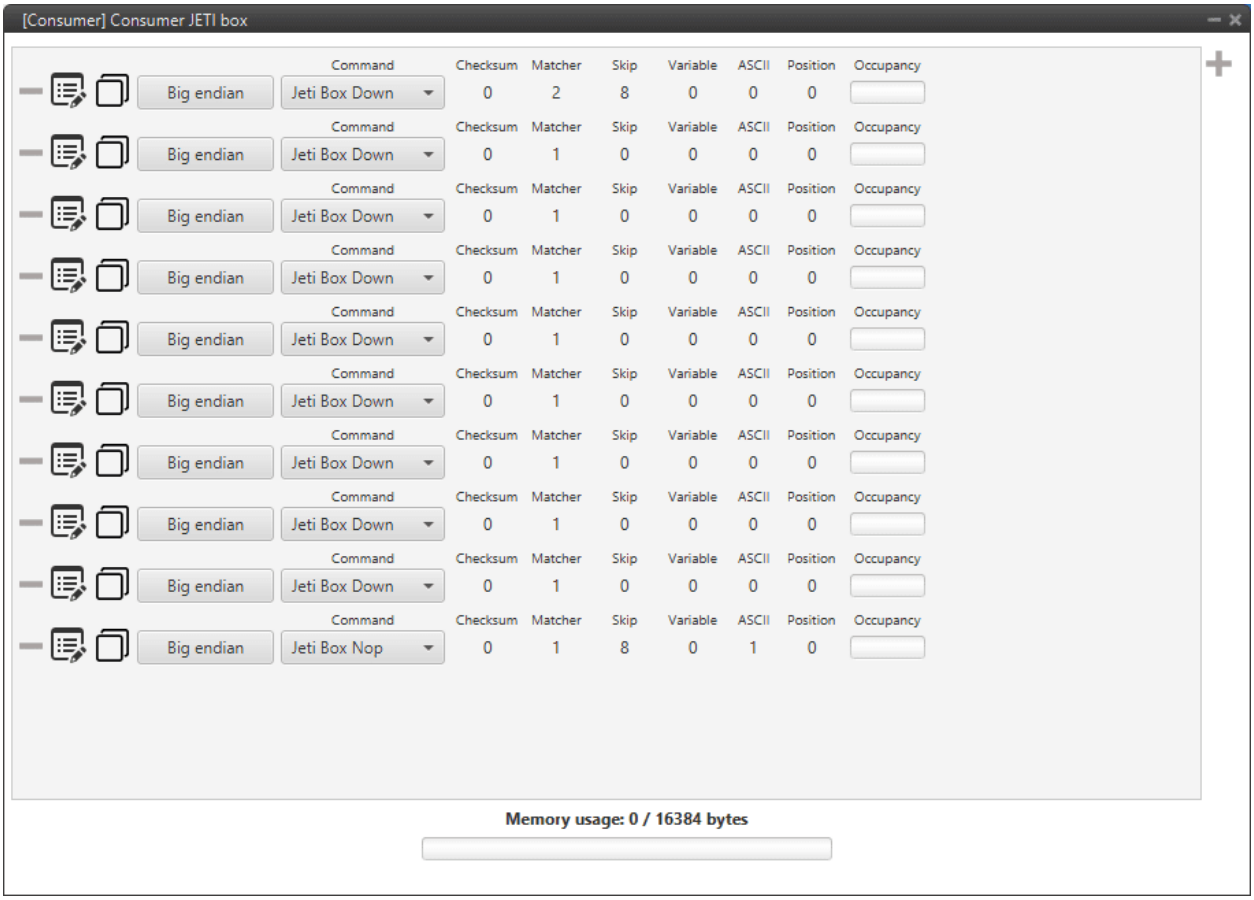


Fig. 31: **JETI box example**

Below is a small example with the configuration of one of these messages, the full example can be found in *Jetibox - Integration examples* section of this manual.

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

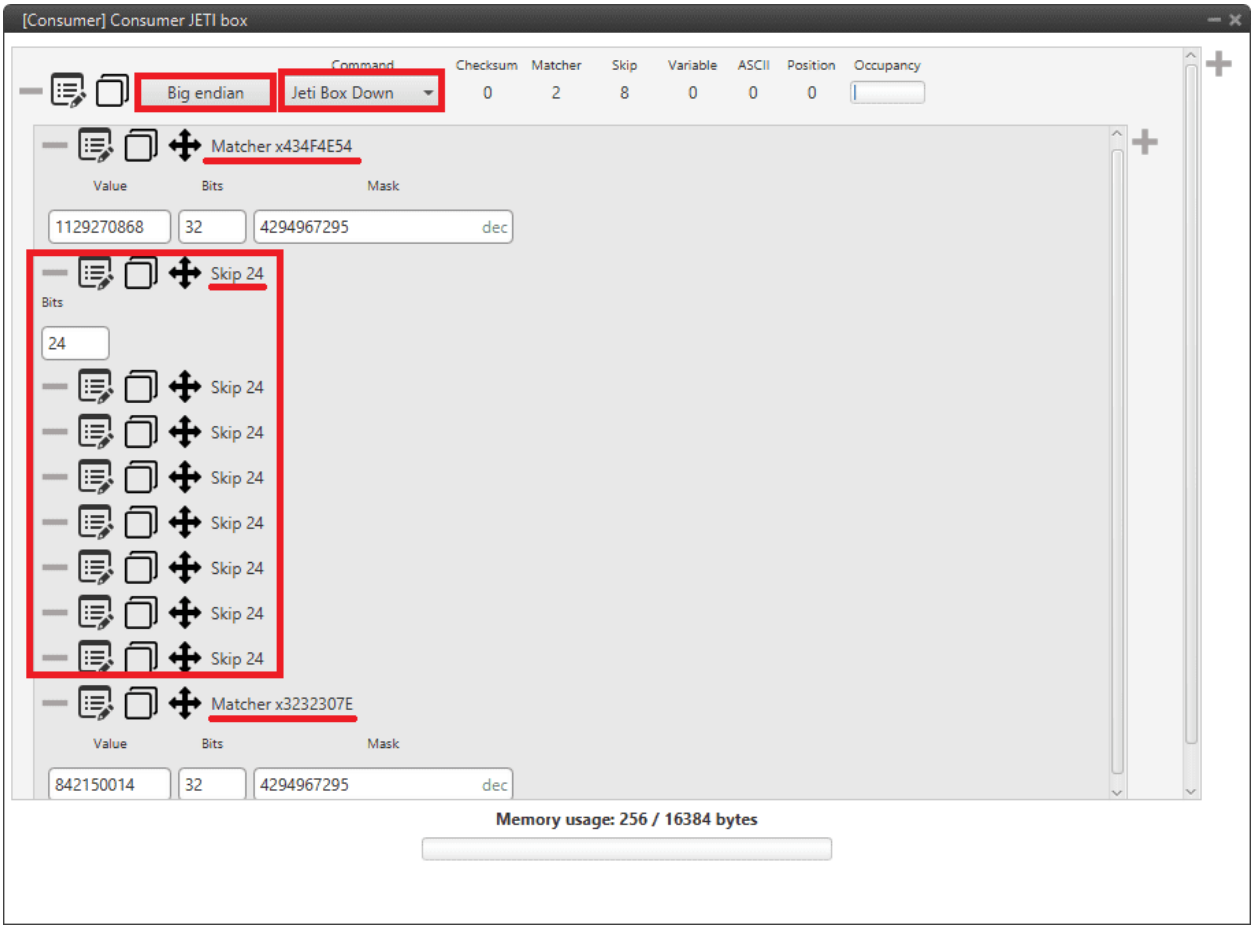


Fig. 32: **JETI box custom message example**

2.3.3.4 ARINC communication protocol

CEX supports the reception and sending of information following the **ARINC 429 communication standard**. ARINC 429 messages consist of a 32-bit sequence as shown below.

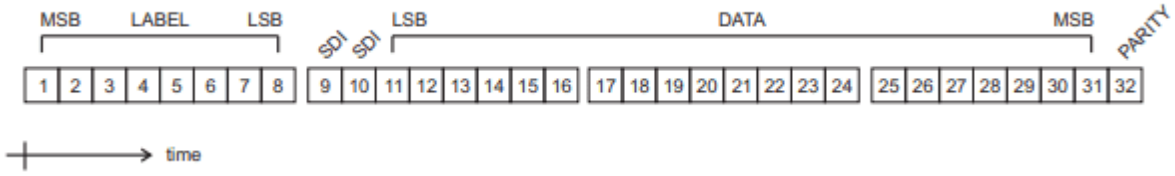


Fig. 33: **ARINC message - Bit sequence**

- **1 - 8**: The label/status byte.
- **9 - 10**: Source Destination Identifier (SDI) bits. SDI function is used when specific messages need to be directed to a specific system of a multi-system installation.
- **11 - 31**: Data bits.
- **32**: Parity bit. Parity is a method of detecting errors in transmission.

The following configuration must be carried out in **CEX PDI Builder** for enabling reception and transmission of ARINC messages.

2.3.3.4.1 ARINC message reception

1. Go to Input/Output → I/O Setup panel → **Configuration tab**.

   Connect an **ARINC producer** to a **Custom message consumer**.

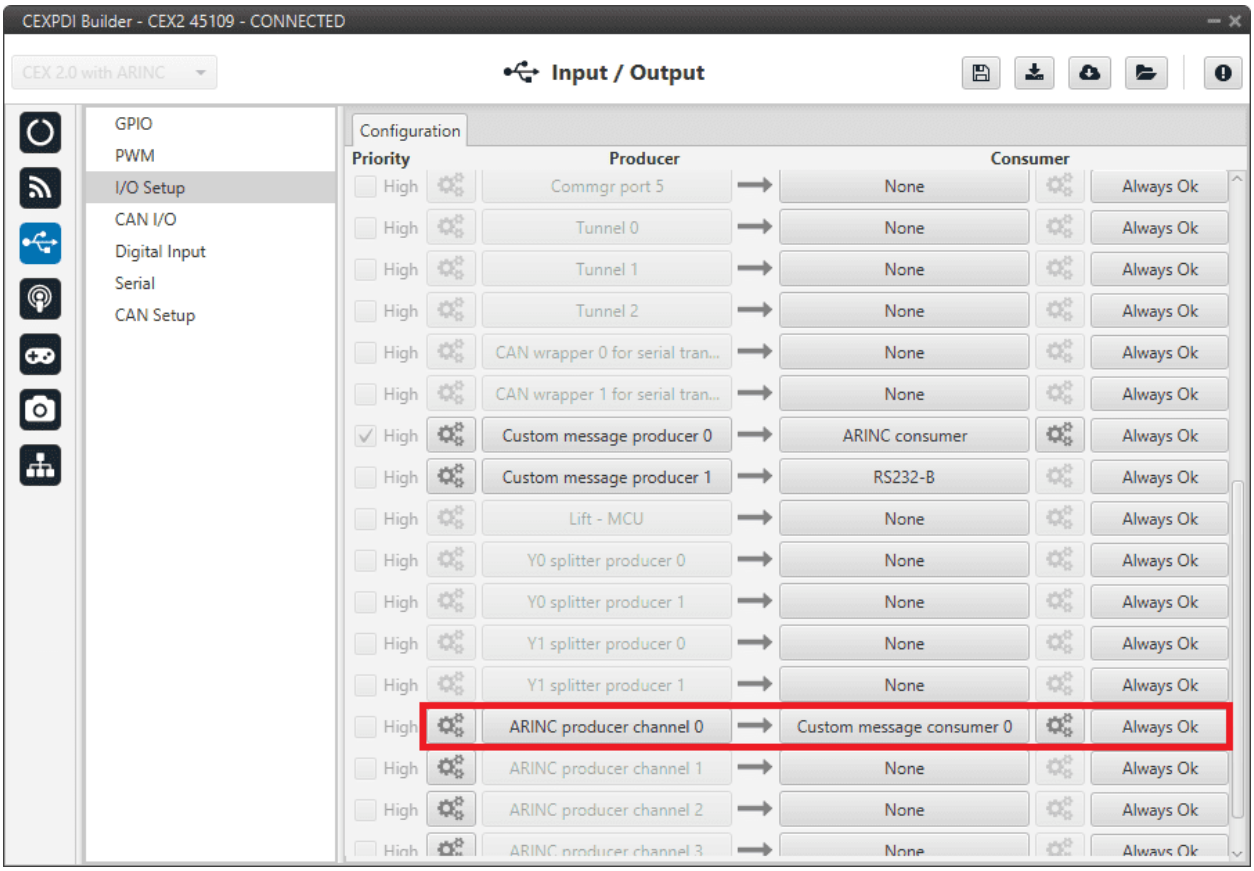


Fig. 34: **I/O Setup - Producer selection**

2. Configure the **ARINC producer** according to the message to read:

   - **Low speed**: Selects the transmission rate for the ARINC 429 transmit channel (**low** if checked, **high** if not).
   - **Parity check**: When this parameter is checked, the 32nd received ARINC bit is overwritten with **a parity flag**. The flag bit is set to a zero when the received ARINC word, including its parity bit, has an odd number of ones. Contrarily, when this **parity check** is not ticked, all 32-bits are received without parity checking.
   - **SDI filter**: Once this parameter is checked, SDI function is enabled. Users must configure the identifier of the ARINC messages to read (0, 1, 2 or 3) corresponding to the decimal value of SDI bits.

   Note: The decimal value is extracted from the binary number formed by bits 10 and 9 **in that order**.

Bit Number		Installation Number
10	9	
0	0	-
0	1	1
1	0	2
1	1	3

Fig. 35: **SDI bits - ID numbers**

- **Enabled labels**: Users can set the labels of the ARINC messages to read, by selecting the corresponding decimal value of Label bits.

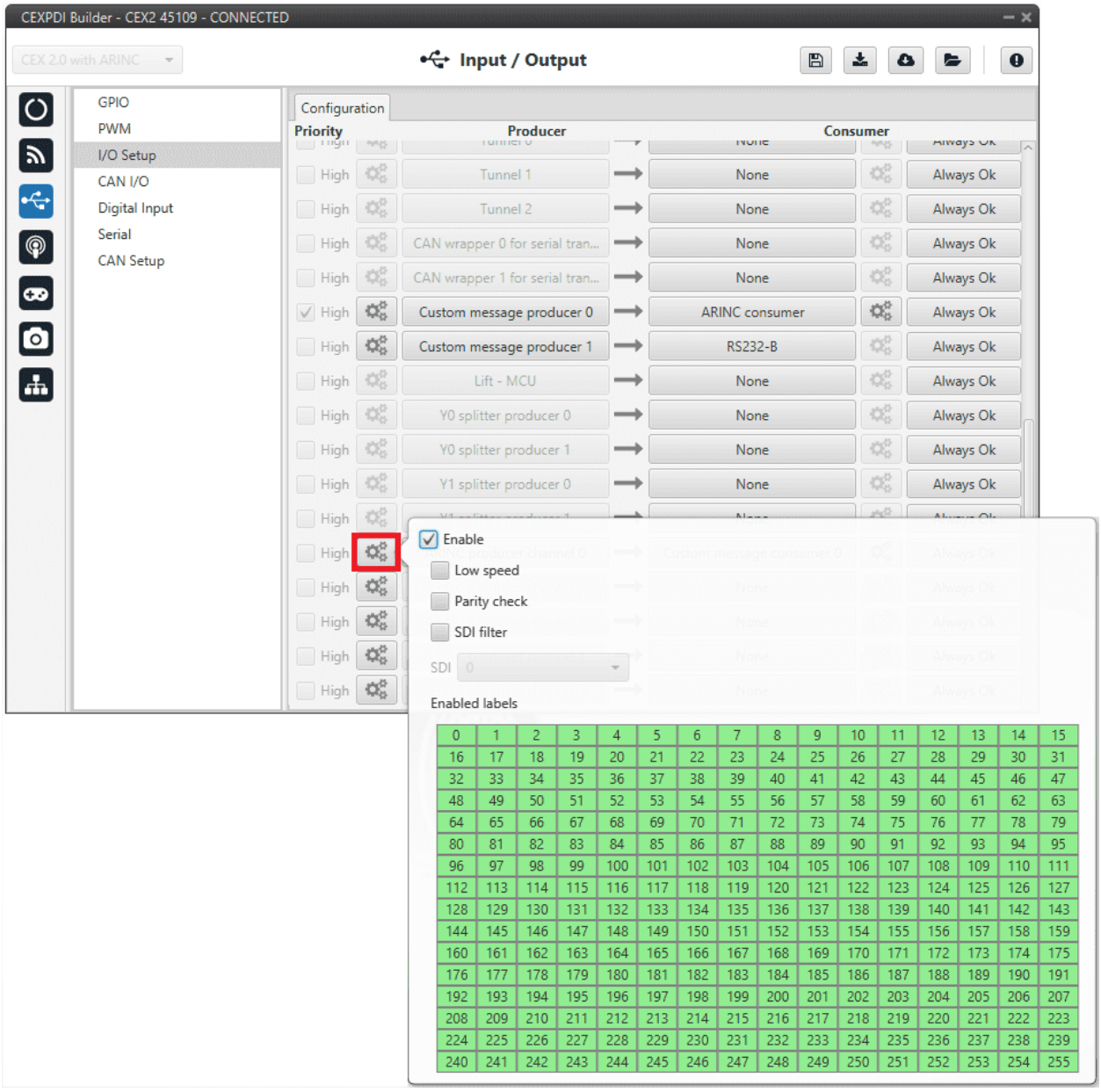


Fig. 36: **I/O Setup - ARINC Producer configuration**

3. Configure the Custom message consumer as follows:

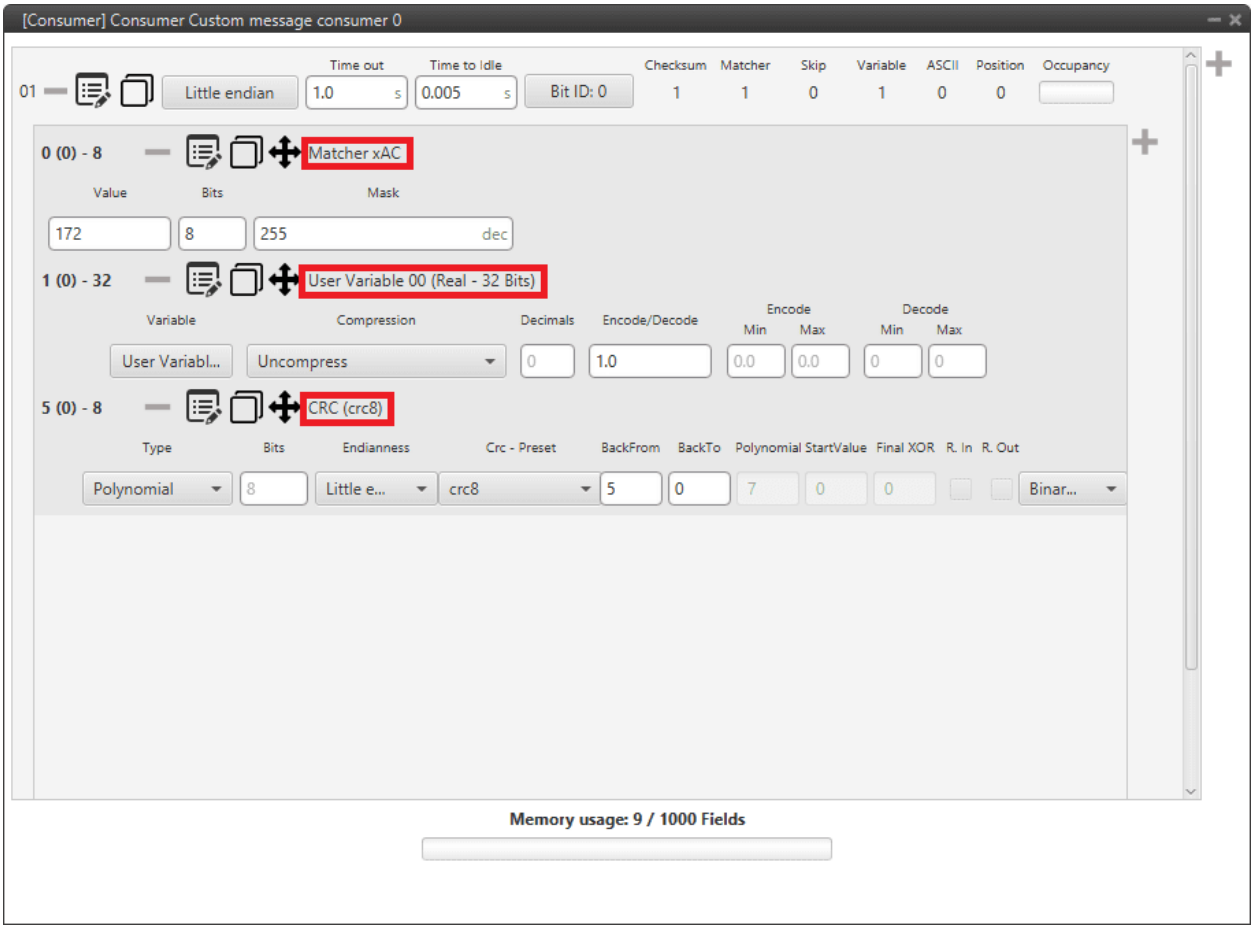


Fig. 37: **I/O Setup - Consumer configuration**

In order to ensure proper communication, it is mandatory to follow the structure above, adding a **Matcher** in the beggining, then a **Variable** (or variables corresponding to the 32 bits of the ARINC message) and a **CRC** at the end of the Custom message with the following values for the parameters:

- **Matcher**:
  - Value: 172
  - Bits: 8
- **Variable** or variables: This must correspond to the 32 bits of the **ARINC message** that users wish to receive. These can be configured in several ways, as long as **all bits are represented and there is no loss of data**. That is, the ARINC message does not necessarily have to be configured as a 32-bit uncompressed variable such as the *User Variable 00* shown in the example.
- **Checksum (CRC)**:
  - Type: Polynomial
  - Endianness: Little endian
  - CRC-Preset: crc8
  - BackFrom: 5
  - BackTo: 0

– Binary mode

2.3.3.4.2 ARINC message transmission

1. Go to Input/Output → I/O Setup panel → **Configuration tab**.

   Connect an **Custom message producer** to a **ARINC consumer**, and configure the consumer according the ARINC message to send.

   - **Low speed**: Selects the transmission rate for the ARINC 429 transmit channel (**low** if checked, **high** if not).
   - **Parity check**: When this parameter is checked, the 32nd transmitted ARINC bit is overwritten with a **parity flag**. Contrarily, when it is not, all 32-bits are transmitted as data.
   - **Even**: When the **Parity check is ticked**, this parameter defines whether the 32nd transmitted bit is set for Even or Odd Parity. Ticking the **Even checkbox** selects **even parity**, if not **odd parity** will be selected.

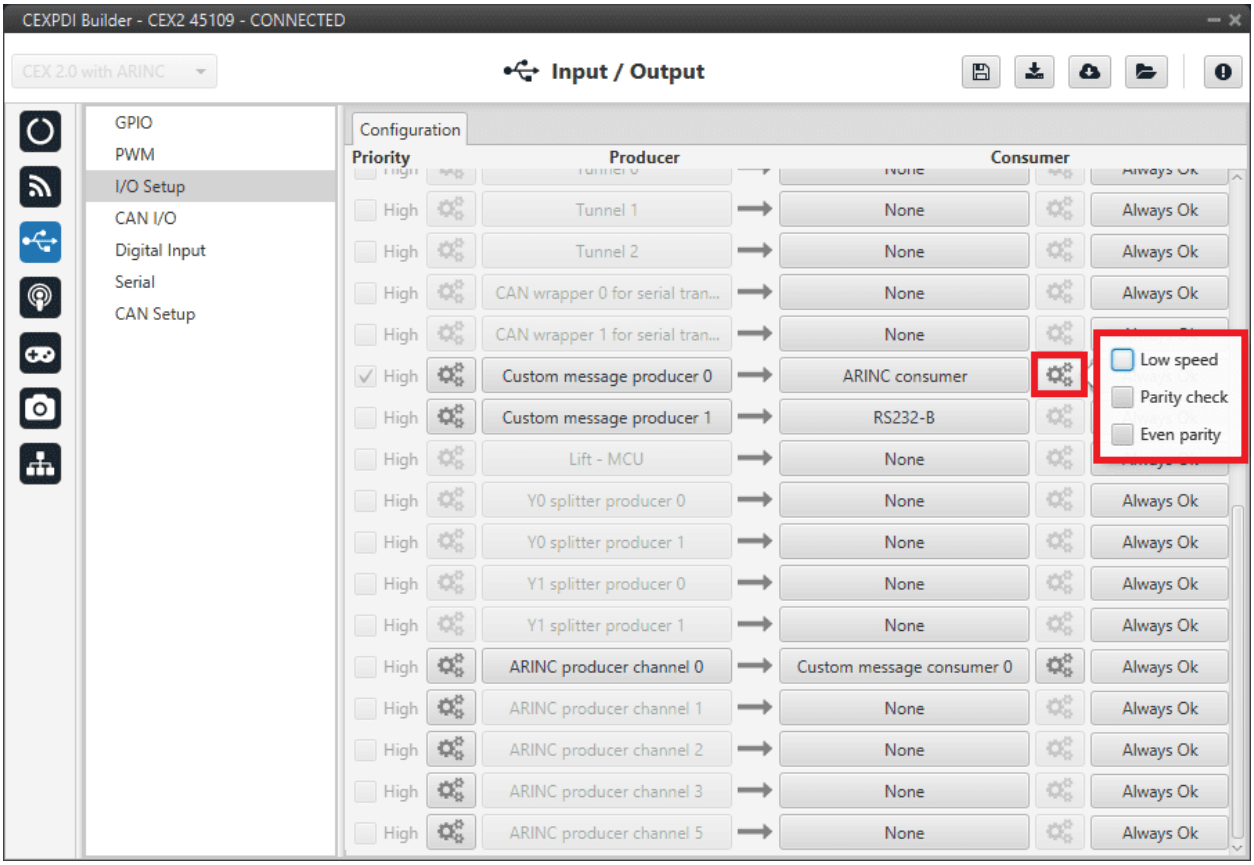


Fig. 38: **I/O Setup - ARINC Consumer configuration**

2. Configure the **Custom message producer** as below. In this example, *Relative Timestamp* variable is being sent.

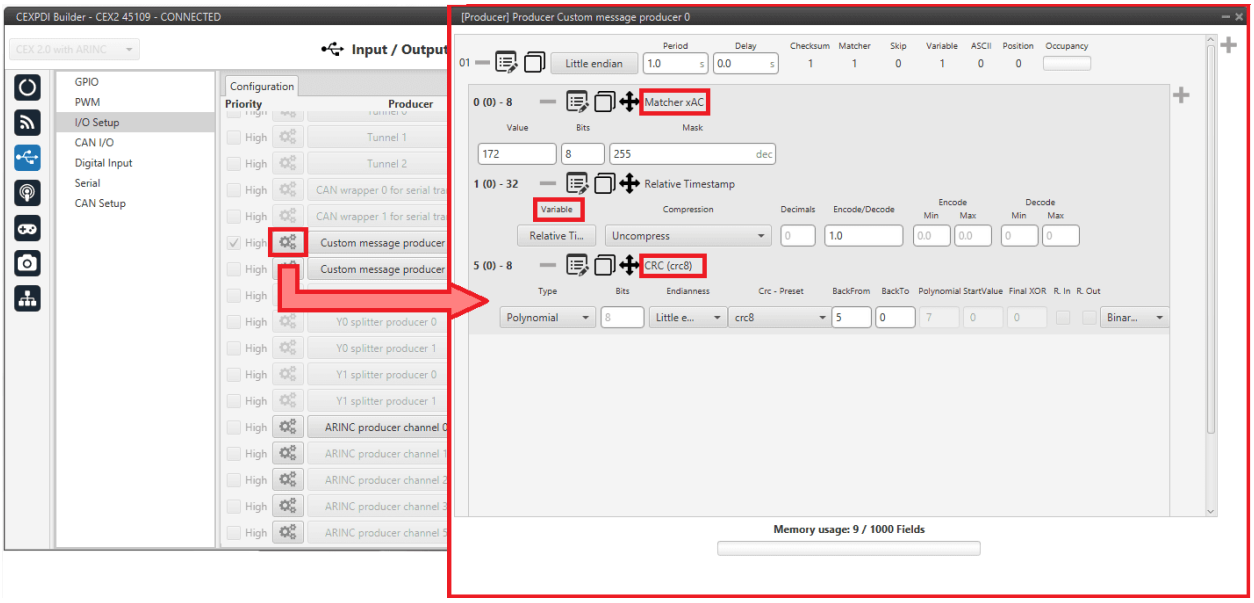


Fig. 39: **I/O Setup - Consumer configuration**

In order to ensure proper communication, it is mandatory to follow the structure above, adding a **Matcher** in the beggining, then a **Variable** (or variables corresponding to the 32 bits of the ARINC message) and a **CRC** at the end of the Custom message with the following values for the parameters:

- **Matcher**:
  - Value: 172
  - Bits: 8
- **Variable** or variables: This must correspond to the 32 bits of the **ARINC message** that users wish to send. These can be configured in several ways, as long as **all bits are represented and there is no loss of data**. That is, the ARINC message does not necessarily have to be configured as a 32-bit uncompressed variable such as the *Relative Timestamp* shown in the example.
- **Checksum (CRC)**:
  - Type: Polynomial
  - Endianness: Little endian
  - CRC-Preset: crc8
  - BackFrom: 5
  - BackTo: 0
  - Binary mode

2.3.4 CAN I/O

Warning: Remember that for the reception of CAN messages, Mailboxes need to be configured accordingly.

A CAN (Controller Area Network) Bus is a robust vehicle bus standard widely used in the aviation sector. CEX is fitted with two CAN buses that can be configured independently.

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

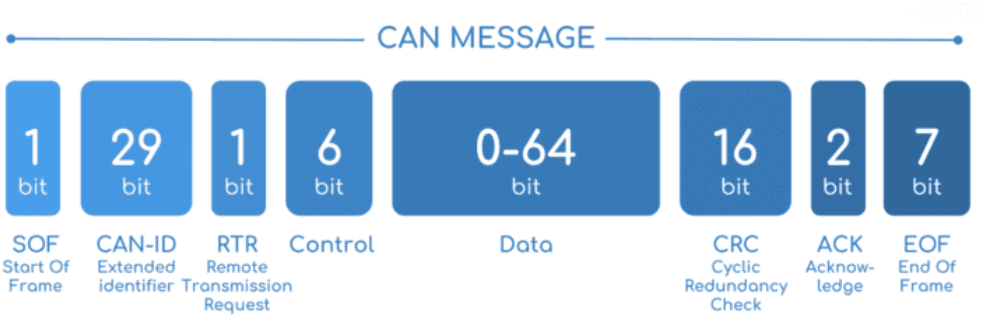


Fig. 40: **CAN message structure**

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

For more information on the CAN Bus protocol, see CAN Bus protocol section of the **CEX Software Manual**.

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

2.3.4.1 Configuration

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

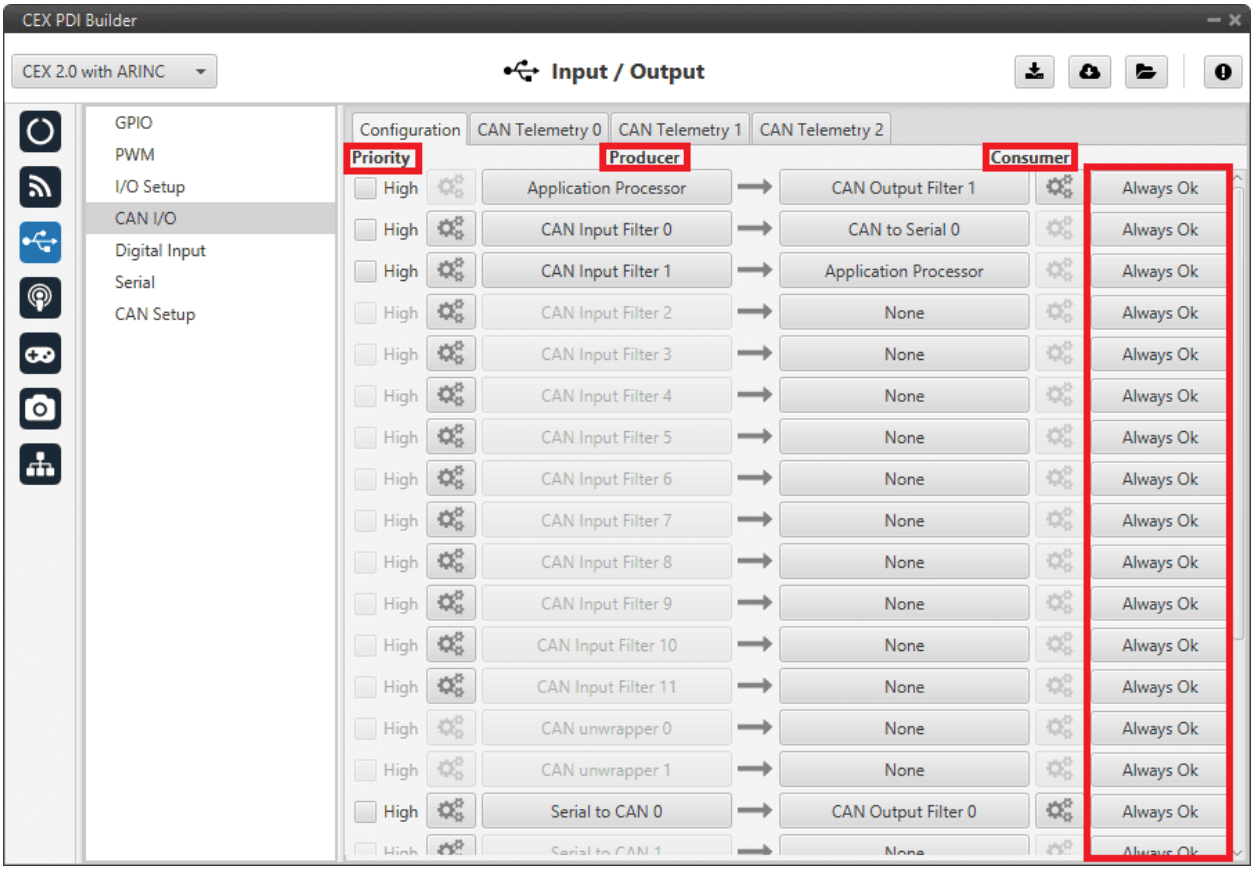


Fig. 41: **CAN I/O - Configuration panel**

In this menu, the user can find the same 'columns' (**Priority**, **Producer**, **Consumer** and **Bit**) as in the *I/O Setup panel*. In addition, the process for configuring producers and consumers is also the same as described in the *I/O Setup - Input/Output* section.

> **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 CEX.

On the one hand, CEX has the **producers** shown below:

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

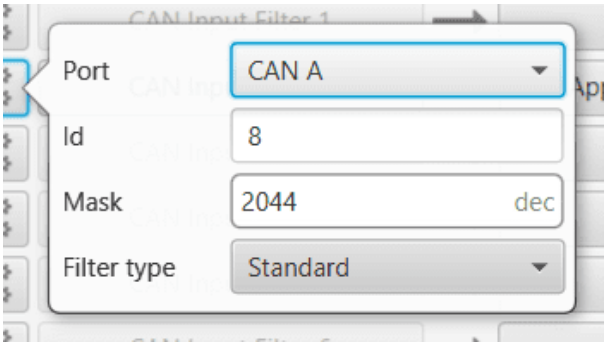


Fig. 42: **CAN Input Filter configuration**

- **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**: Here a CAN Id mask can be set to filter messages. **The mask marks which bits of the message id (in binary) are matched.**

  For example, if standard Ids (11 bits) from 8 to 11 (100 to 111 in binary) want to be admitted, the mask should be set to binary 11111111100, i.e. 2044 in decimal.

  > **Warning:** Make sure that the mask is set properly to be able to receive the desired CAN messages.
  >
  > The mask should be **11 bits for Standard** frame format and **29 bits for Extended** frame format.
  >
  > More information about this can be found in *How to create a mask - FAQ* section of this manual.

- **Filter type**: The options available 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 via CAN output, it has to be connected to *I/O Setup* **consumer**. It can be configured in the **configuration button** ( icon), a pop-up window will appear:

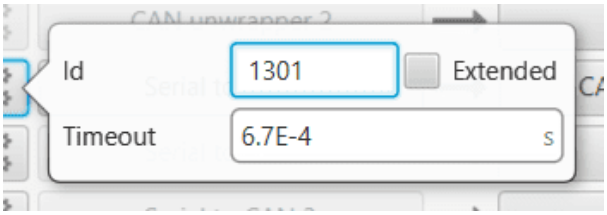


Fig. 43: **Serial to CAN configuration**

- **Id**: CAN Id must be set and it is used to identify messages. The value set has to be **decimal format**.
- **Extended**: If enabled, the frame format will be this, '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 Producer Telemetry**: Telemetry messages sent via CAN (such as CAN custom messages on **Veronte Autopilot 1x**). They are configured in the next section, see *CAN Telemetry*.

On the other hand, the **consumers** are the following:

- **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 the configuration button ( icon).

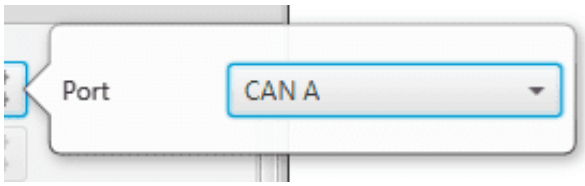


Fig. 44: **CAN Output Filter configuration**

- **CAN serial wrapper**: CAN messages via serial output, it has to be connected to *I/O Setup* **producer** *(CAN wrapper for serial transmission)*.
- **CAN GPIO consumer**: CAN messages from **Autopilot 1x** or **Autopilot 4x** for GPIO inputs. An example of how to implement it can be found in the *GPIO Command - Integration examples* section of the present manual.
- **CAN to Serial**: This undoes the 'Serial to CAN' action, it has to be connected to *I/O Setup* **producer**.
- **CAN Custom Consumer Telemetry**: Telemetry messages received via CAN (such as CAN custom messages on **Veronte Autopilot 1x**). They are configured in the next section, see *CAN Telemetry*.

2.3.4.2 CAN Telemetry

In the CAN Telemetry tabs (there are 3 available), the user chooses the telemetry to be sent/received via 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 CEX boots up). They can be used to initialize some devices.
- **TX**: Used to configure transmitted messages.
- **RX**: Used to configure the reception messages (where they are stored).

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.
- CEX has a **CAN limitation** of **40 TX Ini** messages per Custom, **40 TX** messages per Custom and **80 RX** messages per Custom.

  In addition, there is a limit shared with all Customs:

  - Maximum number of **vectors** (fieldset): **160**
  - Maximum number of **fields**: **1280**

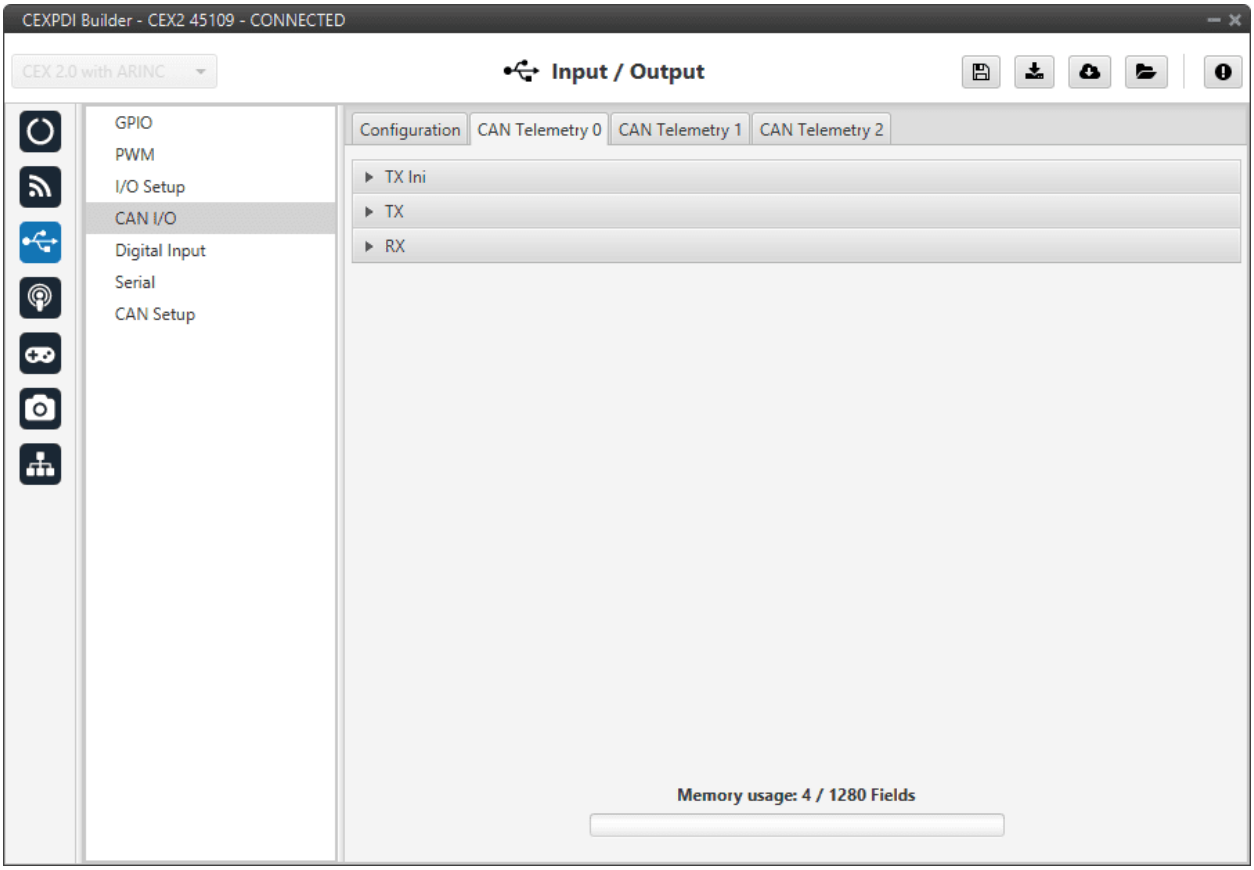


Fig. 45: **CAN I/O - CAN Telemetry panel**

As this section works in exactly the same way as in the **1x PDI Builder** software, the explanation on how to configure the telemetry messages via CAN is reflected in the Custom Messages - Input/Output section of the **1x PDI Builder** user manual

An example of the sending of CAN telemetry messages can be found in the *CAN communication - Integration examples* section of this manual.

2.3.5 Digital Input

CEX digital signal inputs can be used to measure pulse count, pulse widths and PPM signals from a RC radio.

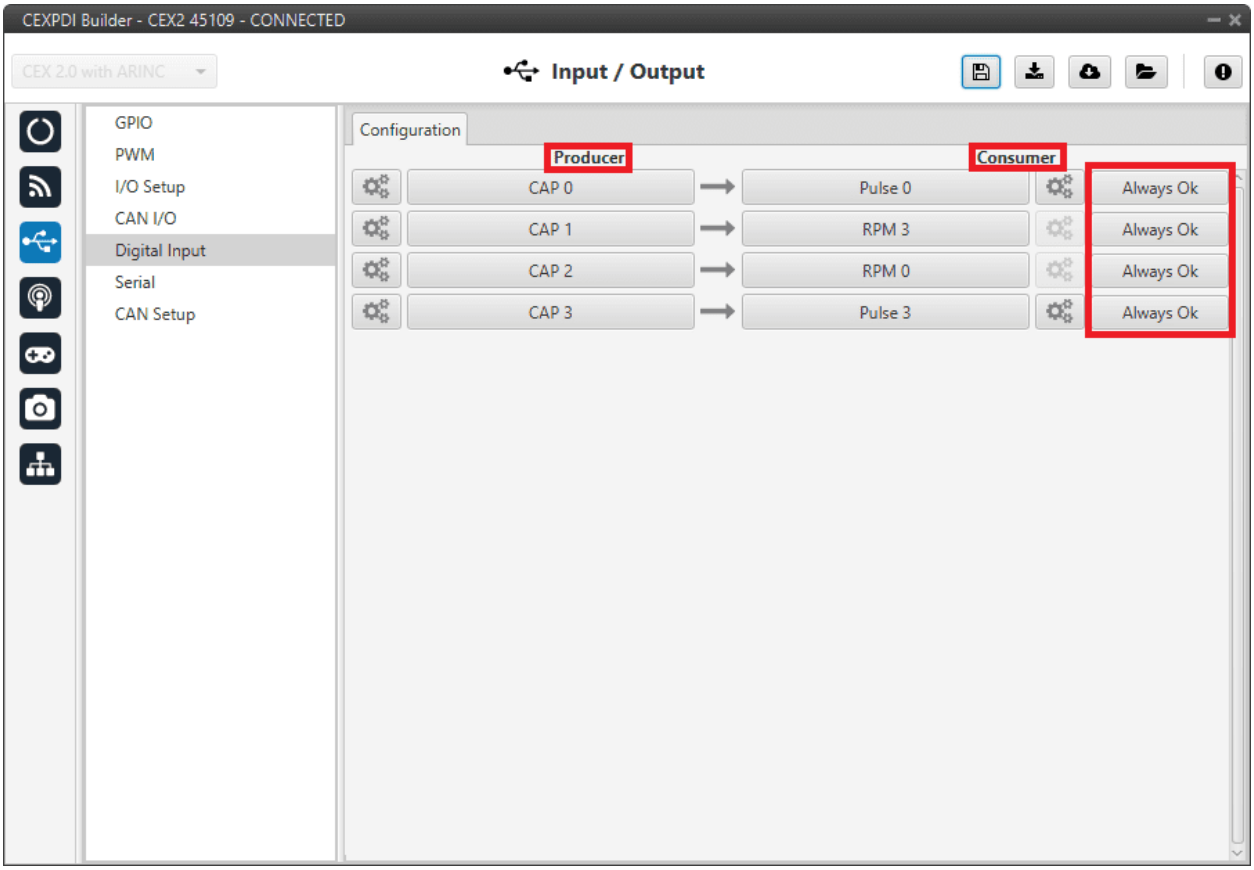


Fig. 46: **Digital Input panel**

In this menu, the user can also find the same 'columns' (**Producer**, **Consumer** and **Bit**) as in the *I/O Setup panel*. In addition, the process for configuring producers and consumers is also the same as described in the *I/O Setup - Input/Output* section.

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

1. Select and configure a **Producer**. There are 4 possible producers: CAP 0 - 3.

   Press on the configuration button ( icon) and a new pop-up window will show.

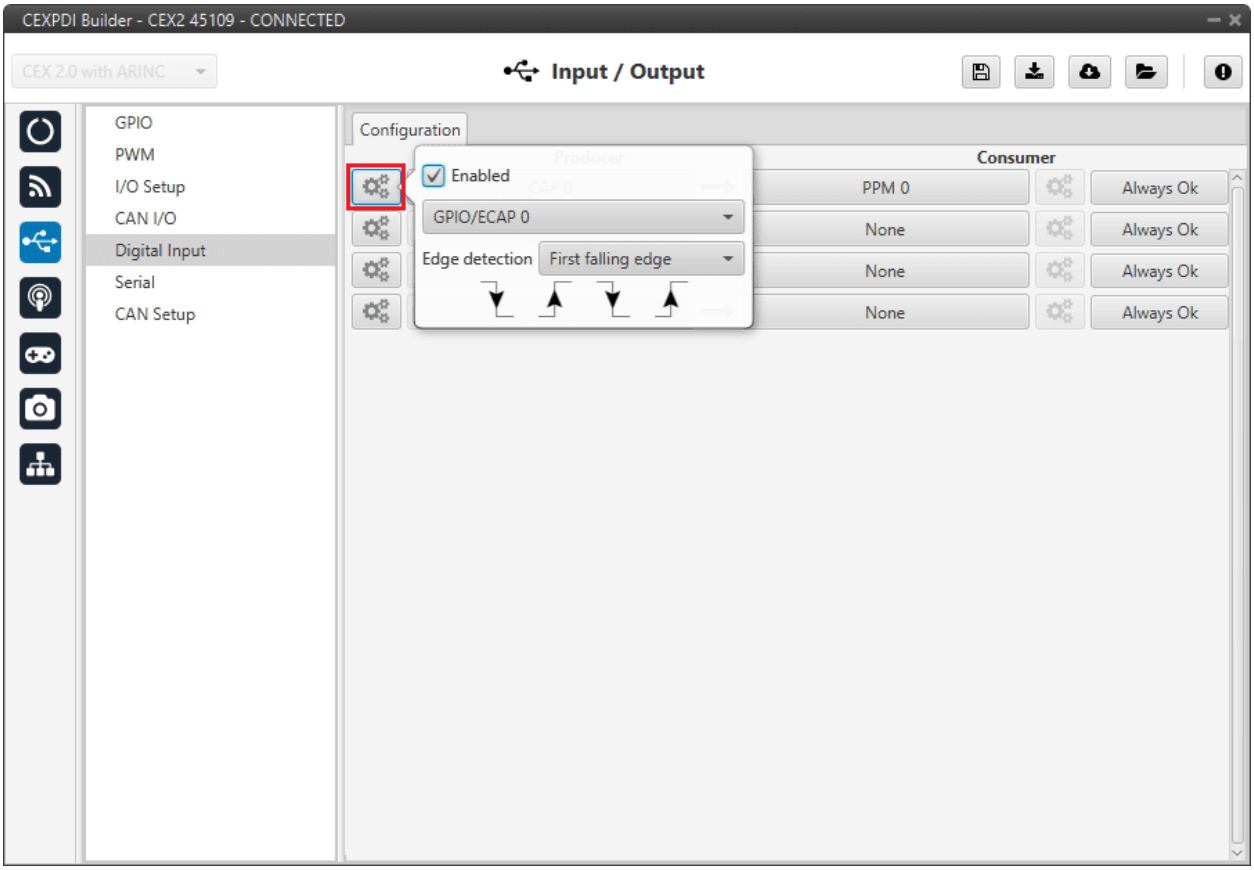


Fig. 47: **Digital Input panel - Producer**

The pop-up window contains the following configurable elements:

- **Enable**: By ticking this checkbox, the corresponding producer is enabled.

Note: When enabling it, in the *GPIO menu* the "Function" parameter corresponding to this pin shall automatically change to "Mux 1".

Warning: Please check that this change has been made.

- **CAP pin entry**: Selects which **pin** this CAP is associated to and, therefore, to which the device is connected to.

Note: Pins are associated in the following way:

- **CAP 0** with **GPIO/ECAP 0**
- **CAP 1** with **GPIO/ECAP 1**
- **CAP 2** with **GPIO/ECAP 2**
- **CAP 3** with **GPIO/ECAP 3**

- **Edge detection**: How the pulses are read and transformed into a digital signal (how they are processed).

  By clicking on the drop-down menu, the following options can be selected:

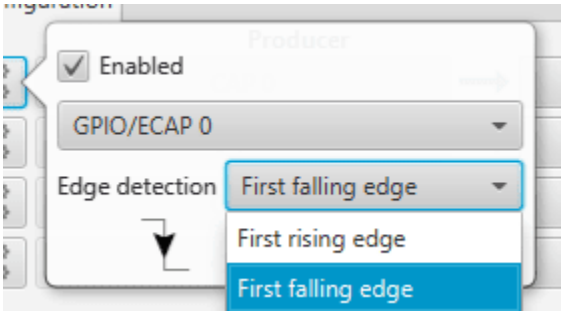


Fig. 48: **Digital Input panel - Edge detection option**

  - **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.

Note: By clicking on the marked arrows, pulse processing can also be configured, getting a customized arrow scheme.

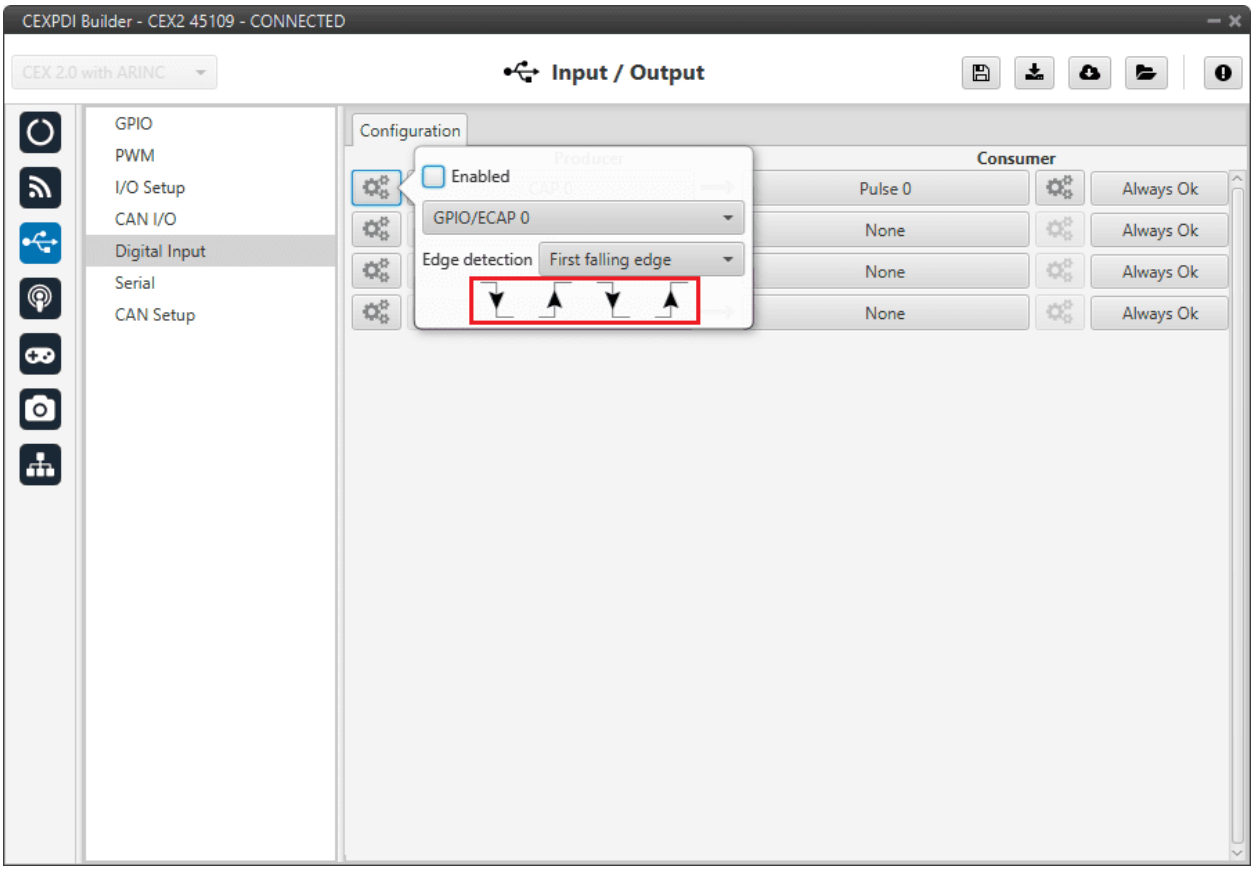


Fig. 49: **Digital Input panel - Edge detection arrows**

2. Click on the **Bind** button to select the type of **Consumer**, it is possible to choose among a PPM 0 (Stick PPM), RPM 0-3 or Pulse 0-3.

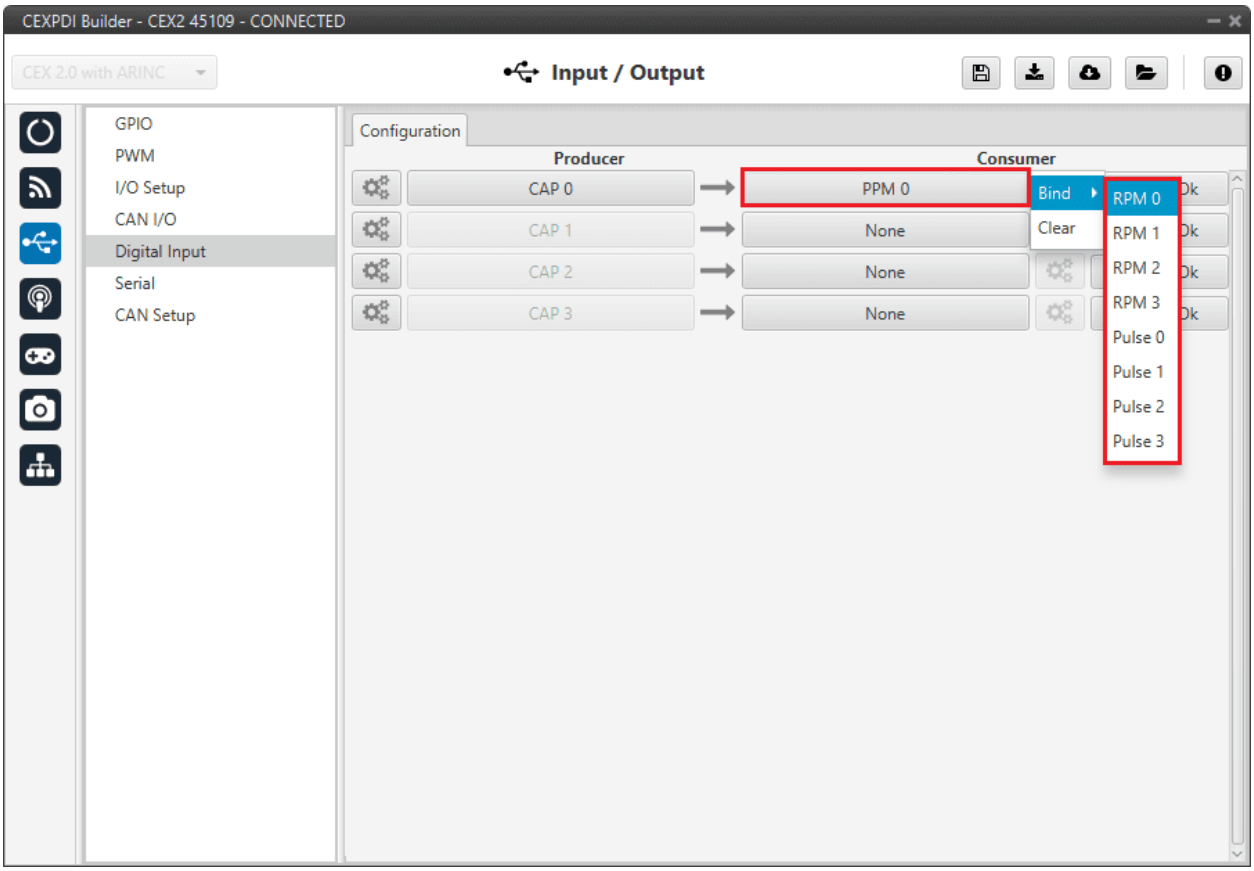


Fig. 50: **Digital Input panel - Consumer**

- **PPM 0** selected: The variable where the information is stored is 'PPM channel 0 output'. Stick PPM is configured in the *Stick* section.
- **RPM 0-3** selected: The variables in which the information read here is stored are '**RPM 0-3**'. For more information on the configuration of RPM, see the *RPM* section.
- **Pulse 0-3** selected: The variables in which the information read here is stored are '**Captured pulse 0-3**'.

  It is possible to configure it cliking on the **configuration button** ( icon):

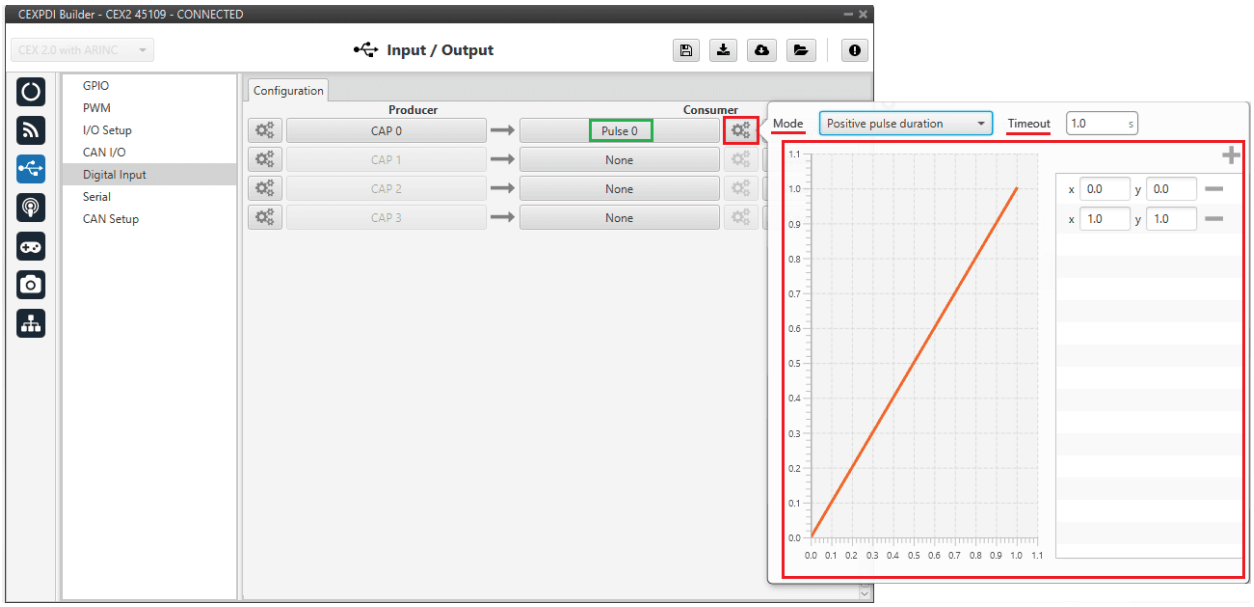


Fig. 51: **Digital Input panel - Pulse**

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

- **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.

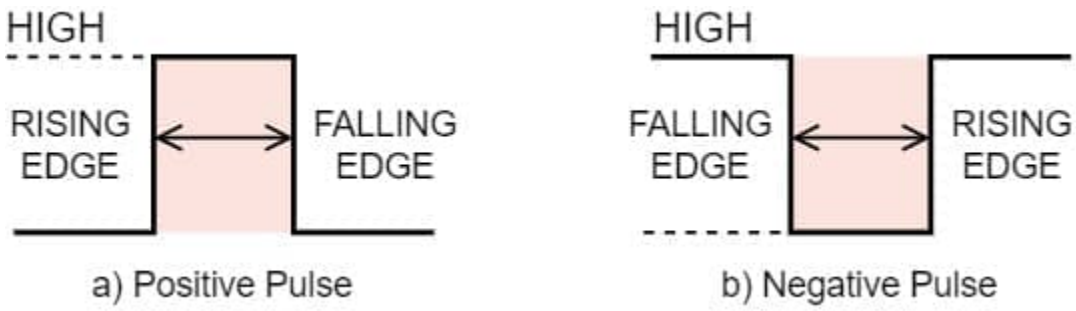


Fig. 52: **Positive/Negative pulse duration**

- **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.

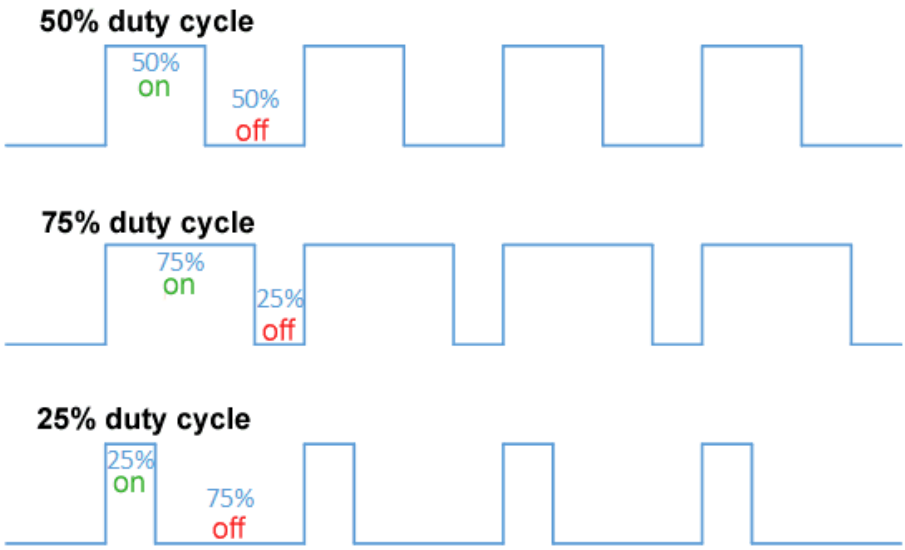


Fig. 53: **Positive/Negative duty cycle**

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

Example

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

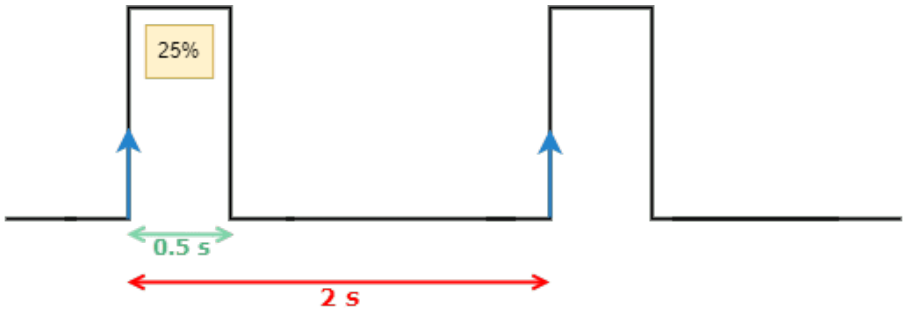


Fig. 54: **Signal generated**

On the other hand, if **Positive pulse duration** is selected as Consumer and it is configured as in the previous image (*Digital Input - Pulse*), the value obtained in the variable **Captured pulse** (*Captured pulse 1* in the following example) will be **0.50s**, this is because it is the period of the "Positive pulse" of that pulse.

However, if **Positive duty cycle** is selected as Consumer, the value obtained in the variable **Captured pulse** (*Captured pulse 2* in the following example) will be **0.25**, this is because it is the positive duty cycle of that pulse.

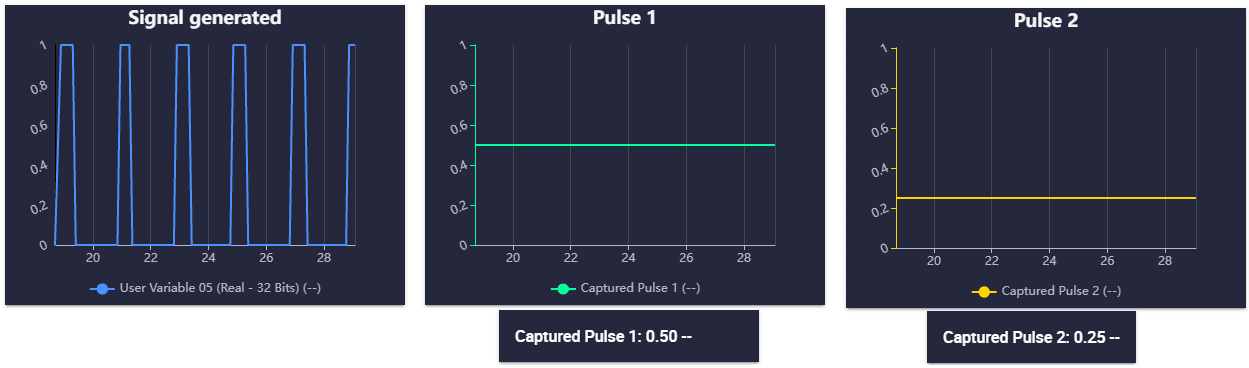


Fig. 55: **Digital Input example**

2.3.6 Serial

CEX 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.

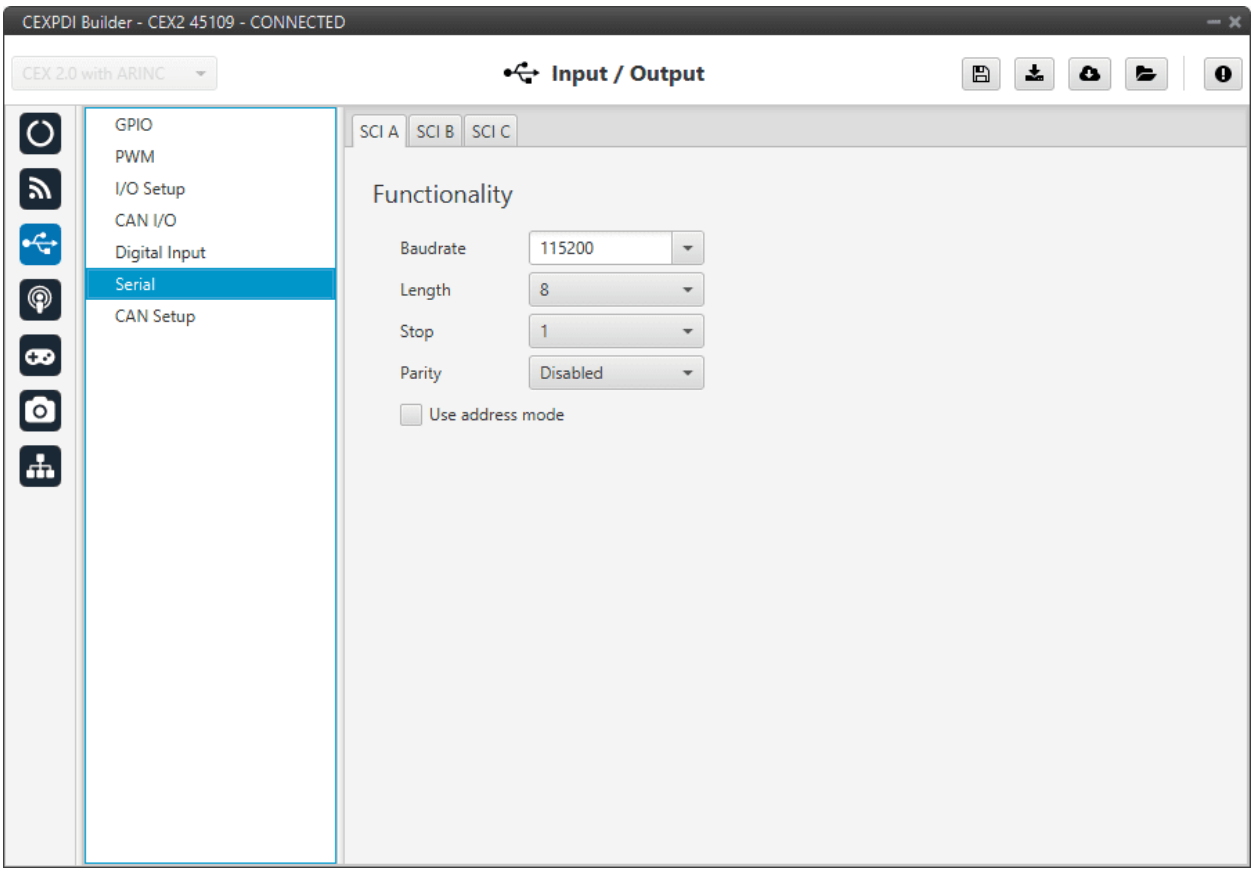


Fig. 56: **Serial panel**

- **Baudrate**: This specifies how fast data is sent over a serial line.
- **Length**: This defines the number of data bits in each character: 4 to 8 bits.

- **Stop**: Number of stop bits sent at the end of every character: 1, 1.5, 2.
- **Parity**: Is a method of detecting errors in transmission. When parity is used with a serial port, an extra data bit is sent with each data character, arranged so that the number of 1 bits in each character, including the parity bit. **Disabled**, **odd** or **even**.
- **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-C.

2.3.7 CAN Setup

In this screen users can configure **the baudrate** and **the reception mailboxes** of each **CAN Bus**.

Since **CEX** is going to receive data on the CAN Bus, it is mandatory to configure a certain number of mailboxes to store that data until CEX reads it.

A mailbox can be configured for multiple CAN message IDs as long as the mask is configured correctly and these messages are **sent spaced out** with enough time between them to allow the high priority core to read each one individually. More information on masks can be found in *How to calculate a mask - FAQ* section of this manual.

Warning: Since **CEX PDI Builder allows up to 32 mailboxes**, users should make sure to **leave at least one mailbox free for transmission (TX)**.

If any mailbox is **full** and another message arrives, the new message is **discarded**.

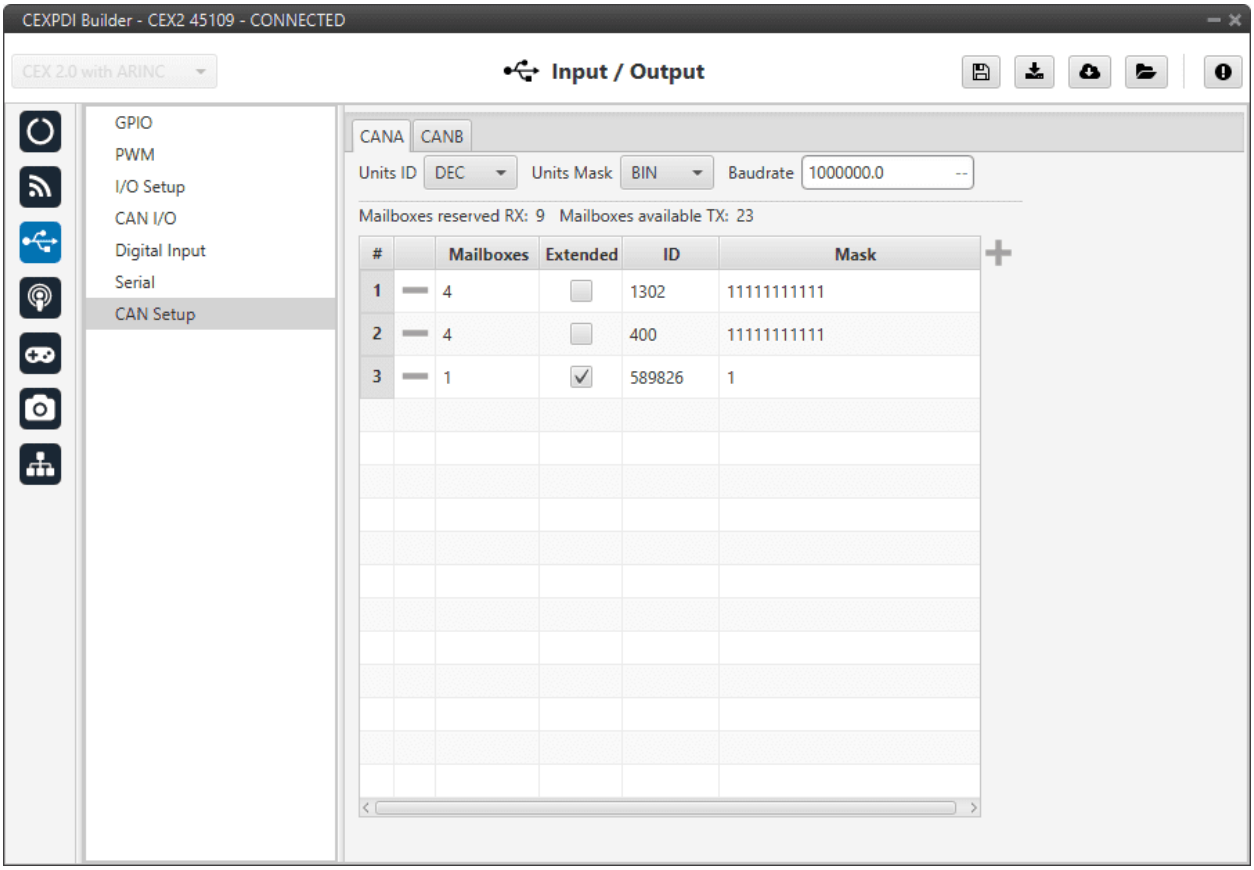


Fig. 57: **CAN Setup panel**

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

2.4 Communications

2.4.1 Ports

Ports configuration allows the user to configure which communication ports (Commgr Ports in *I/O Setup*) will be used for communication. When using the **Route** feature, CEX can be configured to **route** VCP messages for an external Veronte device with a known address (ID) through a given port.

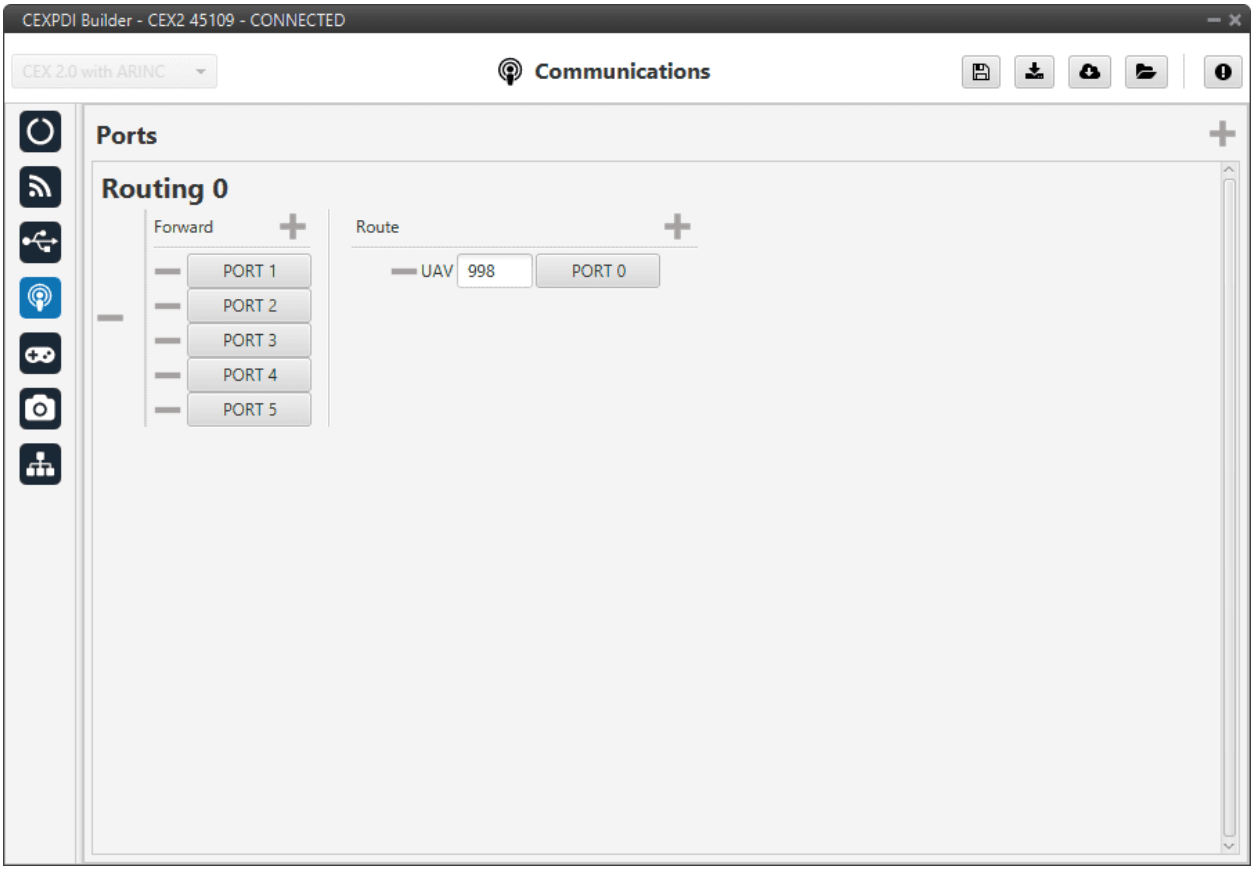


Fig. 58: **Ports menu**

Each port can be configured as either one 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 unsing the Ports action of the Automations menu of **1x PDI Builder**. Routing 0 will always be selected by default when booting CEX.

2.5 Stick

The wired transmitter is configured through the following tabs, which cover the following content:

- Setting of the transmitter's parameters.
- Definition of exponential response-curves for the desired channels.
- Trimming of the channels' neutral position.
- Setting of the data receiving port on the autopilot.

2.5.1 PPM

This tab provides the options to configure CEX to read a PPM radio controller to control the platform.

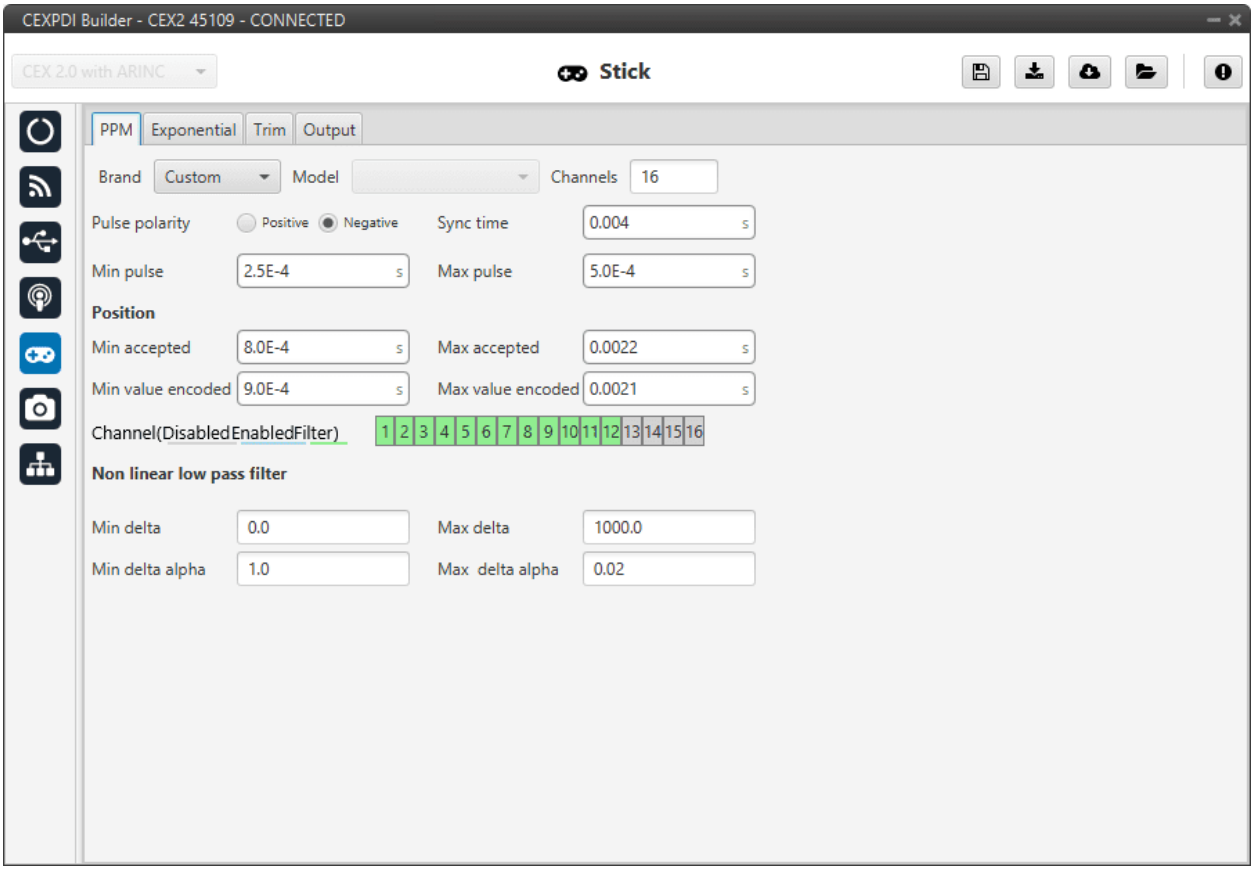


Fig. 59: **Stick - PPM panel**

- **Brand**, **Model** and **Channels**: **CEX 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 (for 8J and 10J)
		12 (for 12K and 14SG)
	T18SZ	8
Jeti	DC 16/DC 24	16
FrSky	Taranis X9D	8
	Horus X12S	8
TBS	Crossfire	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:
  - **Positive**: Default signal is low and goes up to high.
  - **Negative**: Default signal is high and goes down to low.
- **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, CEX 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 shape of the PPM signal.

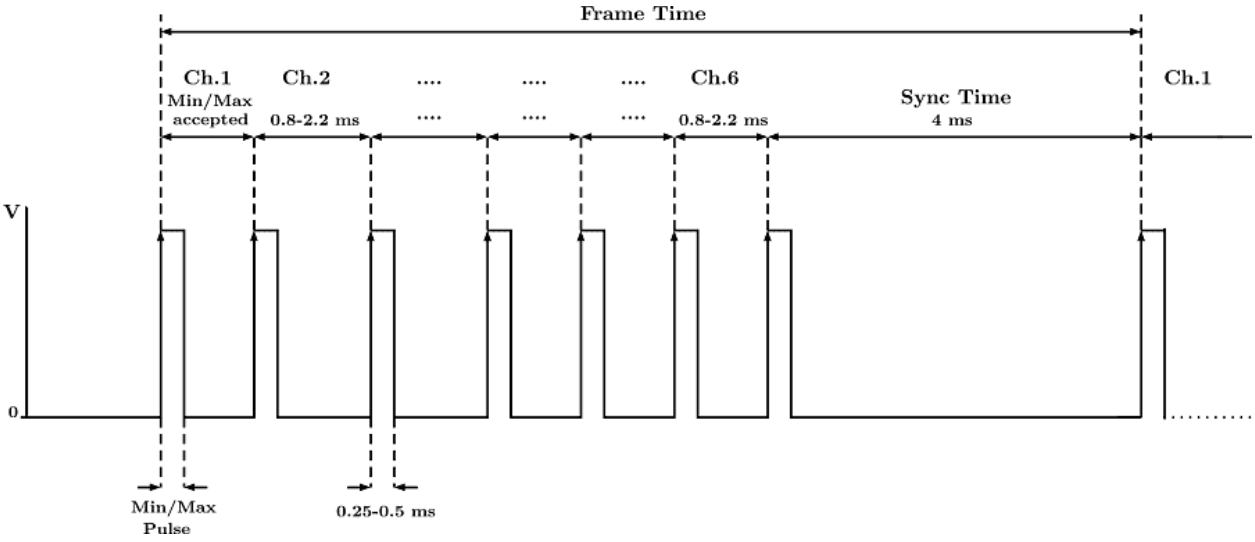


Fig. 60: **PPM signal**

2.5.2 Exponential

The exponential 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.

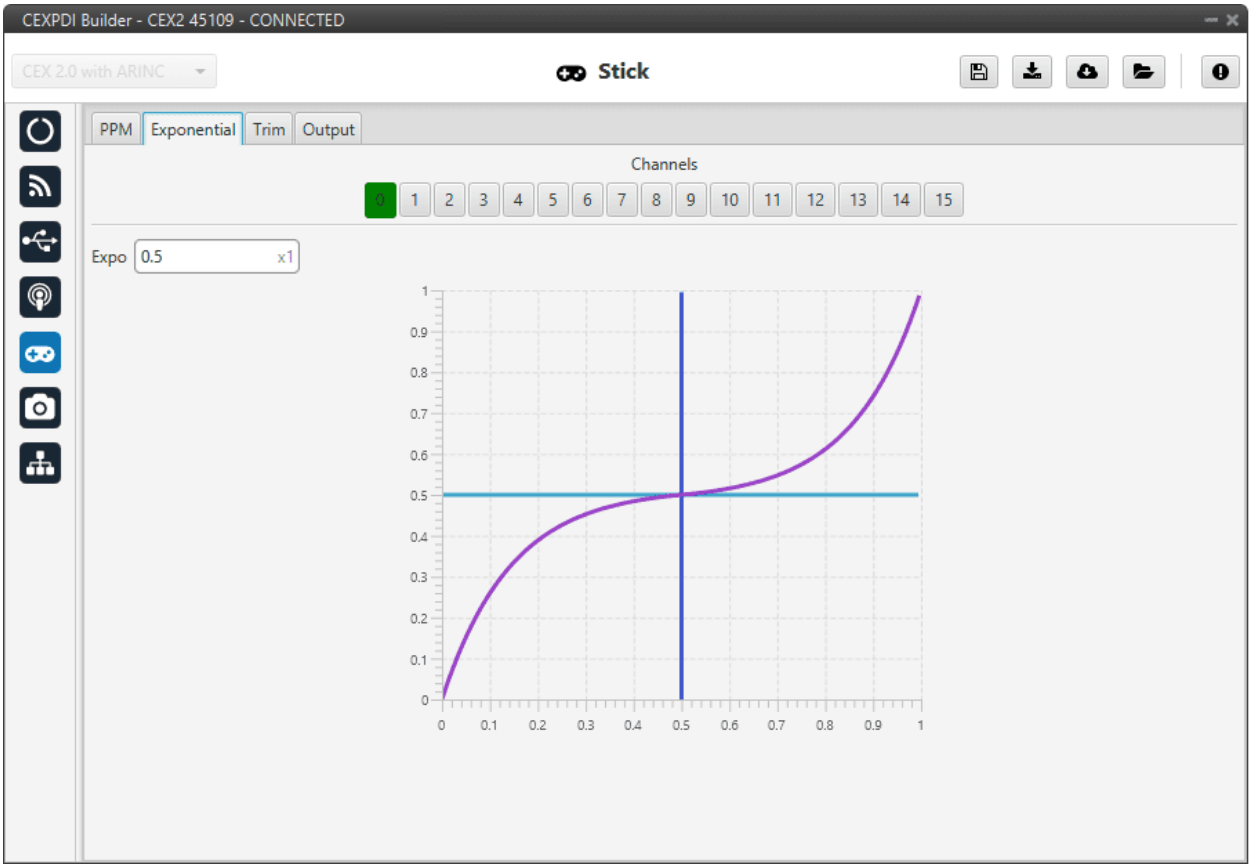


Fig. 61: **Stick - Exponential panel**

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

By enabling the **Avanced** 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.

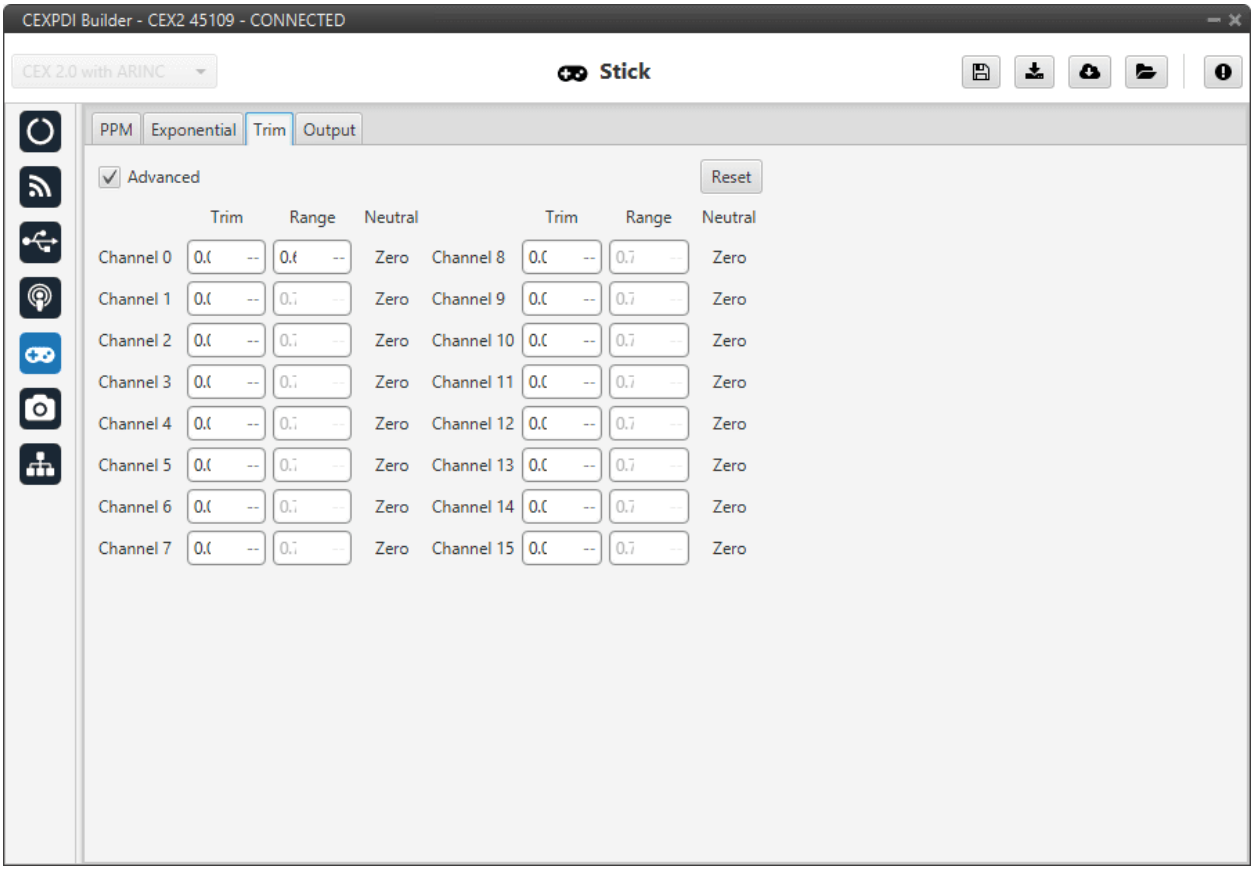


Fig. 62: **Stick - Trim panel**

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 have to be sent to the air unit.

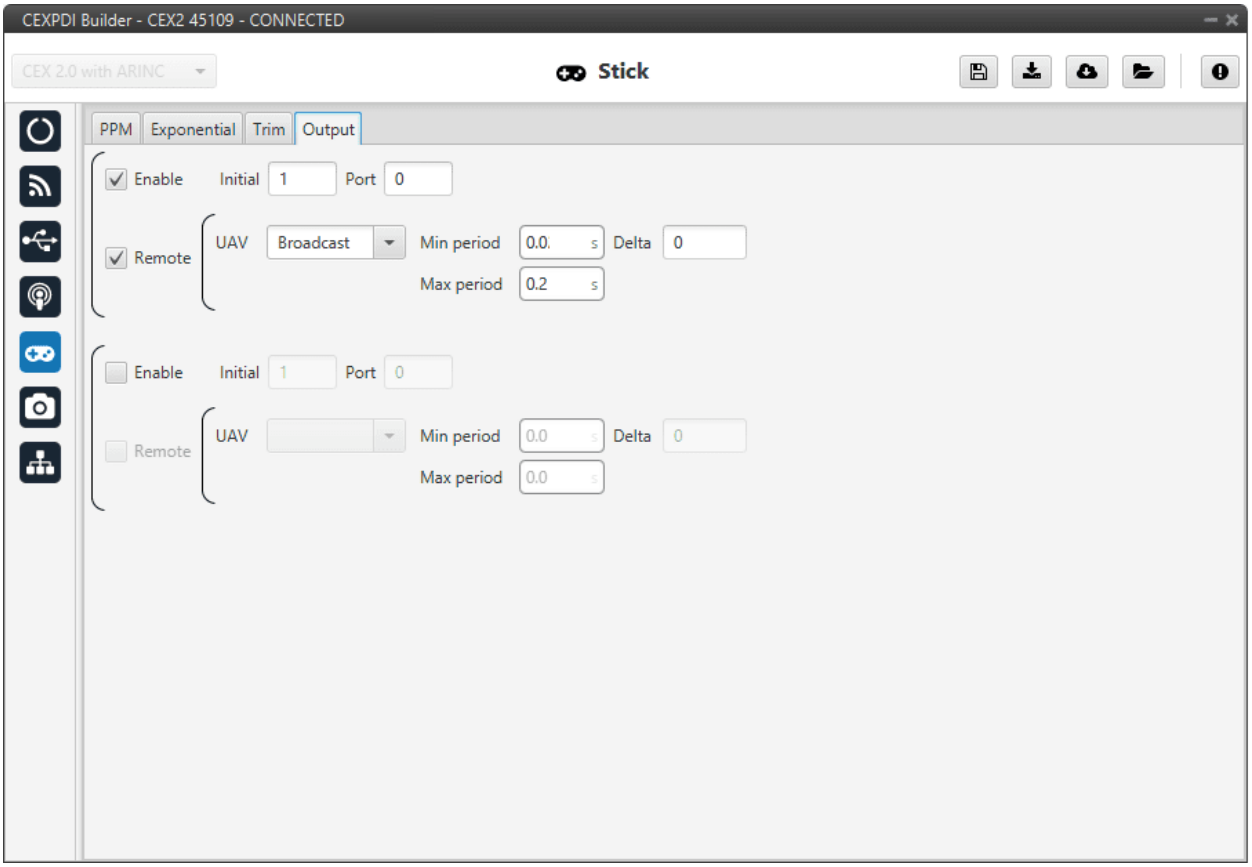


Fig. 63: **Stick - Output panel**

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 the most common:
    - App 2: Veronte applications address.
    - Broadcast: The commands are sent to all units on the network. **This option is recommended**.
    - 1x v4.X XXXX: The address of a specific air unit.

    For more information on the available addresses, see List of addresses section of the **CEX Software Manual**.

- **Min period**: As the period is the inverse of the frequency, this is the **maximum frequency**. Therefore, to give the pilot more control, this is the frequency that is set when the **stick is commanding**. A *Min period* of **0.02s** is recommended.
- **Max period**: As the period is the inverse of the frequency, this is the **minimum frequency**. Thus, to free up bandwidth, this is the frequency that is set when the **stick is idle**. A *Max period* of **0.2s** is recommended.
- **Delta**: This parameter determines whether the frequency is set to the minimum or maximum period set above.

  If CEX 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). **10 Hz** are recommended.

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

CEX can simulate a Jetibox to read telemetry from legacy Jeti devices.

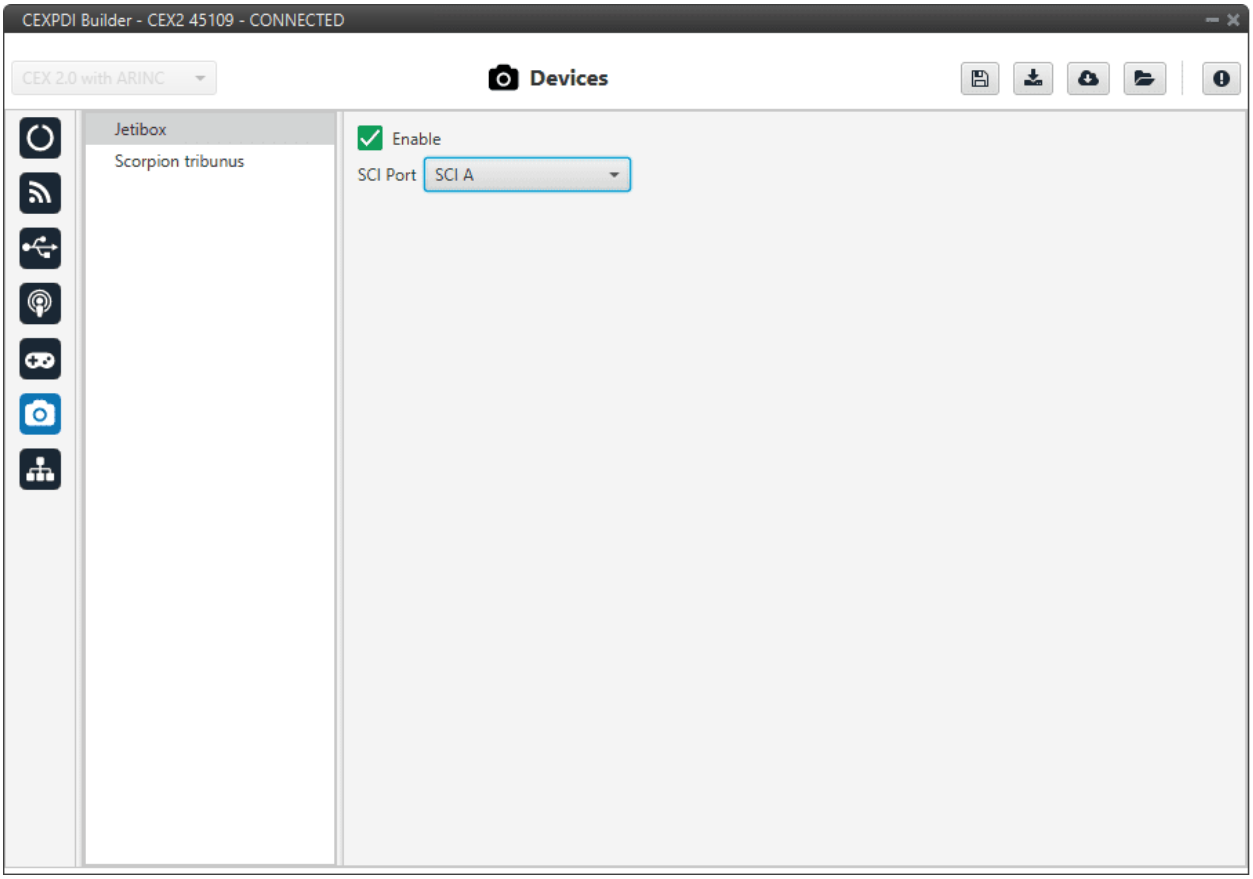


Fig. 64: **Jetibox panel**

- **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 the *Serial* and *I/O Setup panels*.

An example of this can be seen in the *Jetibox - Integration examples* section of the present manual.

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 wil be ignored.

2.6.2 Scorpion tribunus

CEX can read telemetry from Tribunus ESCs by connecting it to one of its serial ports.

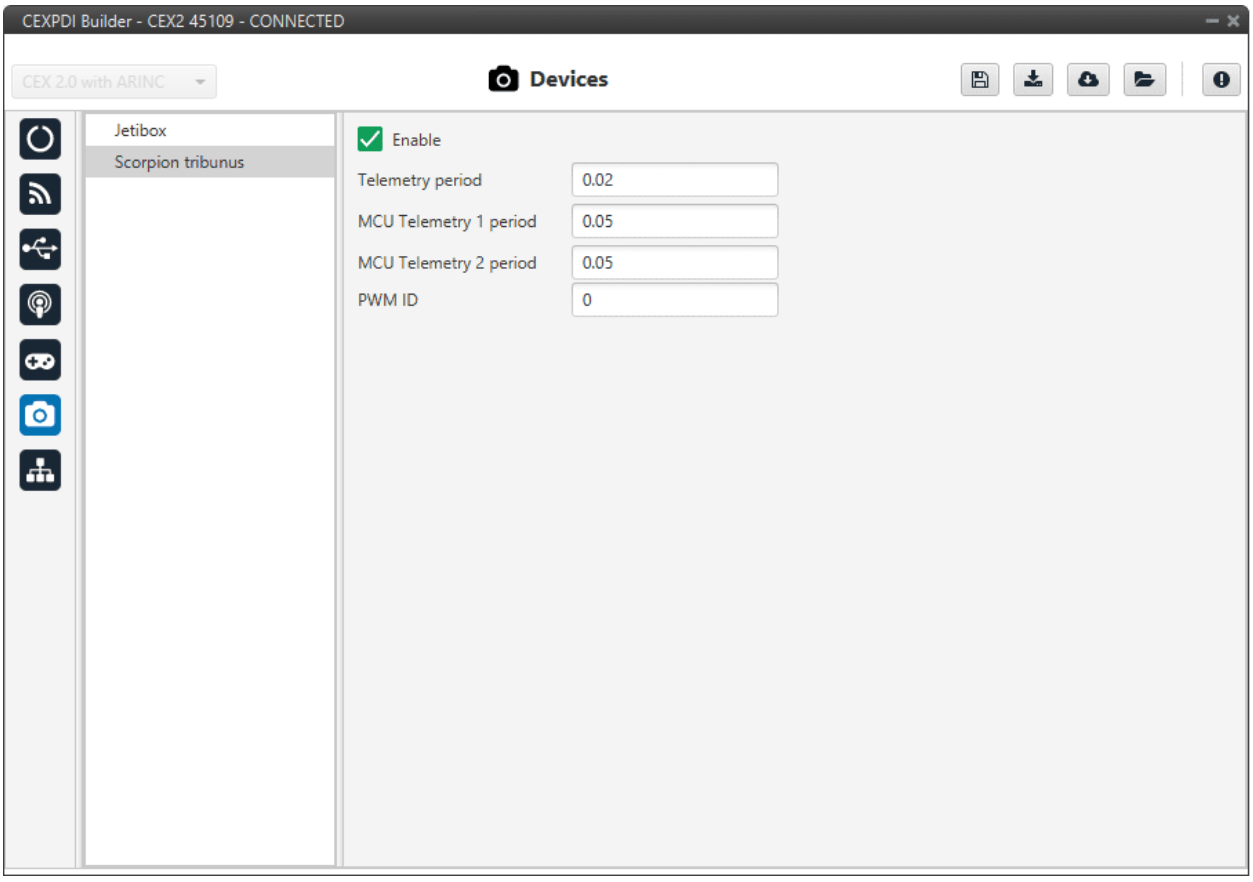


Fig. 65: **Scorpion tribunus panel**

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 of this can be seen in the *Scorpion tribunus - Integration examples* section of this manual.

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 wil be ignored.

2.7 Arbitration

The arbitration algorithm in Veronte is based on a **scoring system**. Each autopilot must send continuously a set of arbitration variables that will be used by the arbiter in order to calculate the score for each unit. Then, based on the scores and the current arbitration mode, the arbiter will choose to keep the current selected autopilot, or switching to one of the other units.

A **Score** is a 32 bit, single precission, floating-point value. This parameter is first computed resulting in a range between **0** and **0xFFFFFFFF**, where **0** is a perfect score (*Score'*). To achieve a better understanding, *Score'* is converted to a value comprehended between 0 an 100 (*Score*), being **100** the best possible score in the end.

Scores are calculated using the **arbitration variables** received from each autopilot at their dedicated addresses. After receiving the value, the following formula is used to compute the score for their respective unit:

$$score'_i = \frac{\sum_j^N min\left[\left(\frac{x_{i,j}-\mu_j}{d_{j,max}}\right)^2, 1\right] \cdot w_j}{\sum_j^N w_j}$$

$$score_i = 100 \cdot (1 - score'_i)$$

Where:

j	Variable index
$x_{i,j}$	j variable from autopilot i
μ_j	j variable reference
$d_{j,max}$	Maximum allowed error
w_j	Weight of j variable
N	Number of variables
$score'_i$	Score' of unit i
$score_i$	Score of unit i

Tip: The Arbitration weights should be used to increase or decrease the relevance that a certain arbitration variable has over the calculation of the score.

Any variable in Veronte can be used as an arbitration variable. Depending on the platform, operation, application, etc. the more relevant variables can be selected for its use as arbitration references.

2.7.1 Arbitration

CEX is able to output PWMs using arbitration in the same way **Veronte Autopilot 4x** does. This functionality has to be enabled as follows:

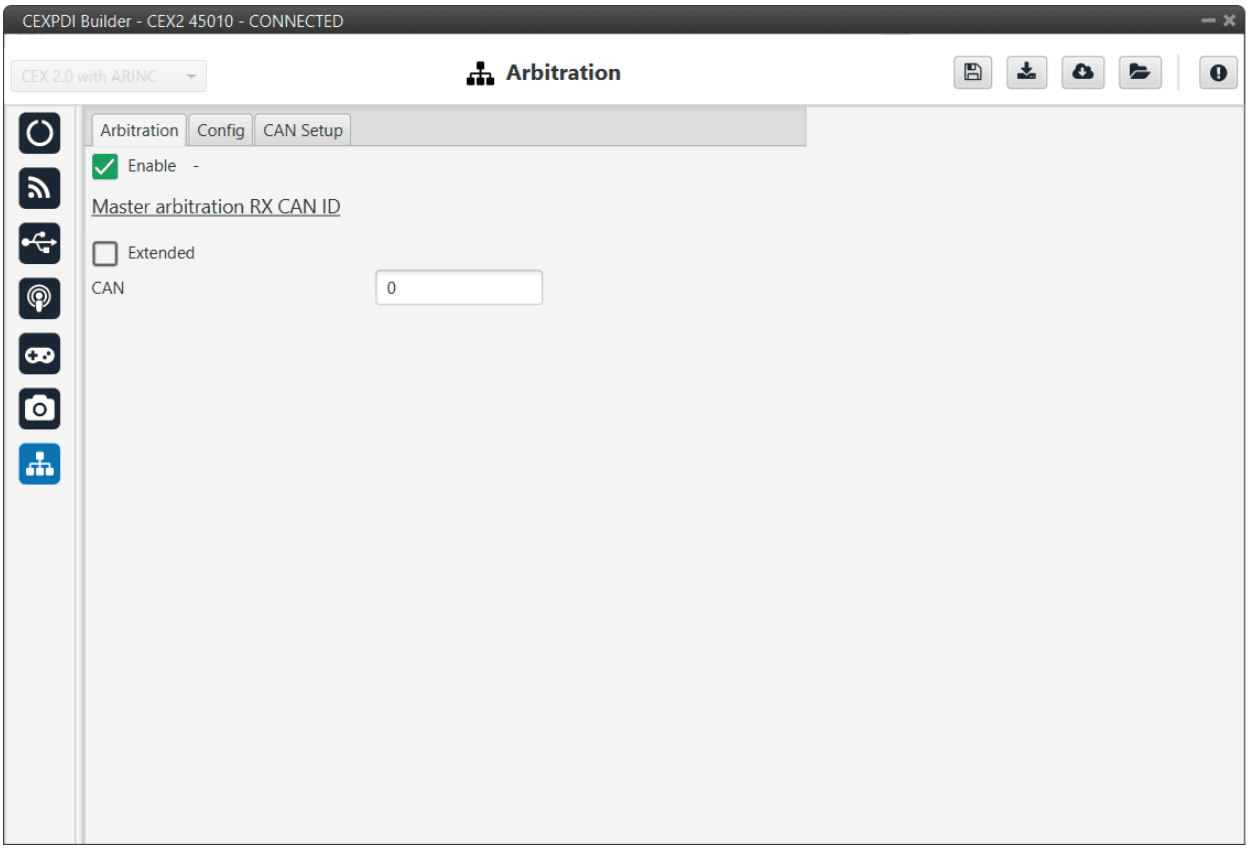


Fig. 66: **Arbitration panel**

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

2.7.2 Config

2.7.2.1 Absolute Arbitration Variables

Absolute arbitration variables are indicators that are **inherently good** or **bad**, and so they are added directly to the score.

Examples of absolute arbitration variables are **Link Quality**, **GNSS accuracy** or warnings such as **Sensors error** or **Position not fixed**.

2.7.2.2 Relative Arbitration Variables

Relative arbitration variables are **not inherently good** or **bad**, and hence need to be compared against the other autopilots in order to calculate its score contribution.

The contribution to the score from a relative arbitration variable will be its **Deviation** from the **Average** of the same variable from each autopilot.

Examples of relative arbitration variables are **Attitude**, **Position**, measurements from sensors, etc.

2.7.2.3 Arbitration Example

Autopilot	Var. Nº	Veronte Variable	Type	$x_{i,j}$	μ_j	w_j	Relative score	$Score'_i$	$Score_i$
1	1	Roll	Relative	0.12	0.096	1	0.054	0.043	95.67
	2	Pitch	Relative	0.30	0.283	1	0.027		
	3	GNSS 1 Accuracy	Absolute	1.7		0.01	0.004		
2	1	Roll	Relative	0.10	0.096	1	0.0011	0.0039	99.60
	2	Pitch	Relative	0.28	0.283	1	0.0011		
	3	GNSS 1 Accuracy	Absolute	1.9		0.01	0.005		
3	1	Roll	Relative	0.07	0.096	1	0.071	0.046	95.39
	2	Pitch	Relative	0.27	0.283	1	0.017		
	3	GNSS 1 Accuracy	Absolute	1.5		0.01	0.0036		

In the above example, **AP2** is considered to be **the best**, since it has the **highest total score**. Even though its **GNSS accuracy** is the worst of all 3, its values for pitch and roll are the ones with the lower deviation from the mean.

2.7.2.4 Config panel

In this panel, the parameters of the arbiter algorithm are set.

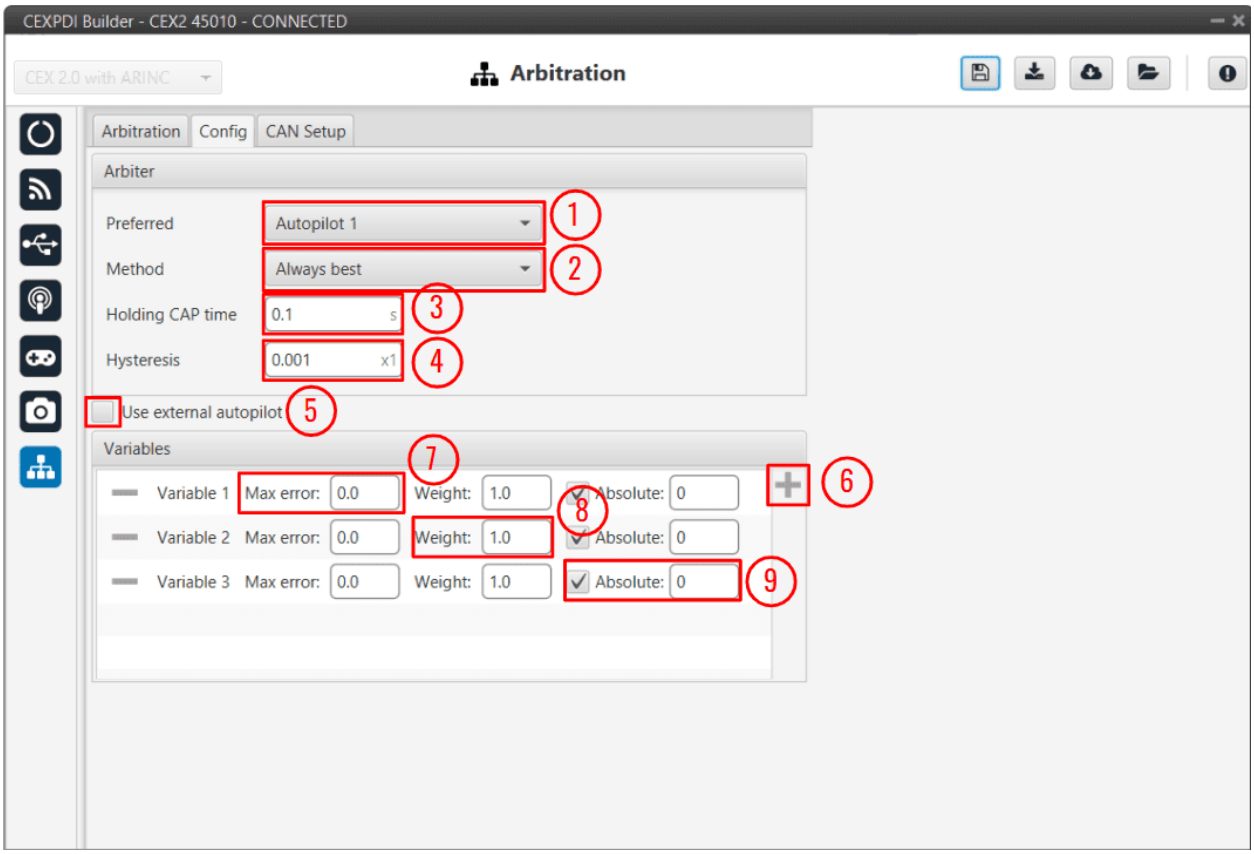


Fig. 67: **Config panel**

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 0 is selected.
     - **Fixed 1**: Autopilot 1 is selected.
     - **Fixed 2**: Autopilot 2 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. **Use external autopilot**: Enables arbitration of **external autopilot**.
6. **Add** button: Adds new arbitration variables.

Arbitration variables have the following configurable parameters:

7. **Max error**: Arbitration maximum error for each variable.
8. **Weight**: Arbitration weight for each variable.
9. **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.

Note: To delete variables click on "**-**".

Currently, the **maximum** amount of **arbitration variables** supported is **32**.

2.7.3 CAN Setup

Here users can configure the receiving CAN Ids for each of the 3 possible **Veronte Autopilots 1x** that are sending data to **CEX**. Therefore, to send data from any of them to the **CEX**, these Ids must be specified.

Note: If arbitration is not enabled, only the configuration of the Autopilot 0 will be used.

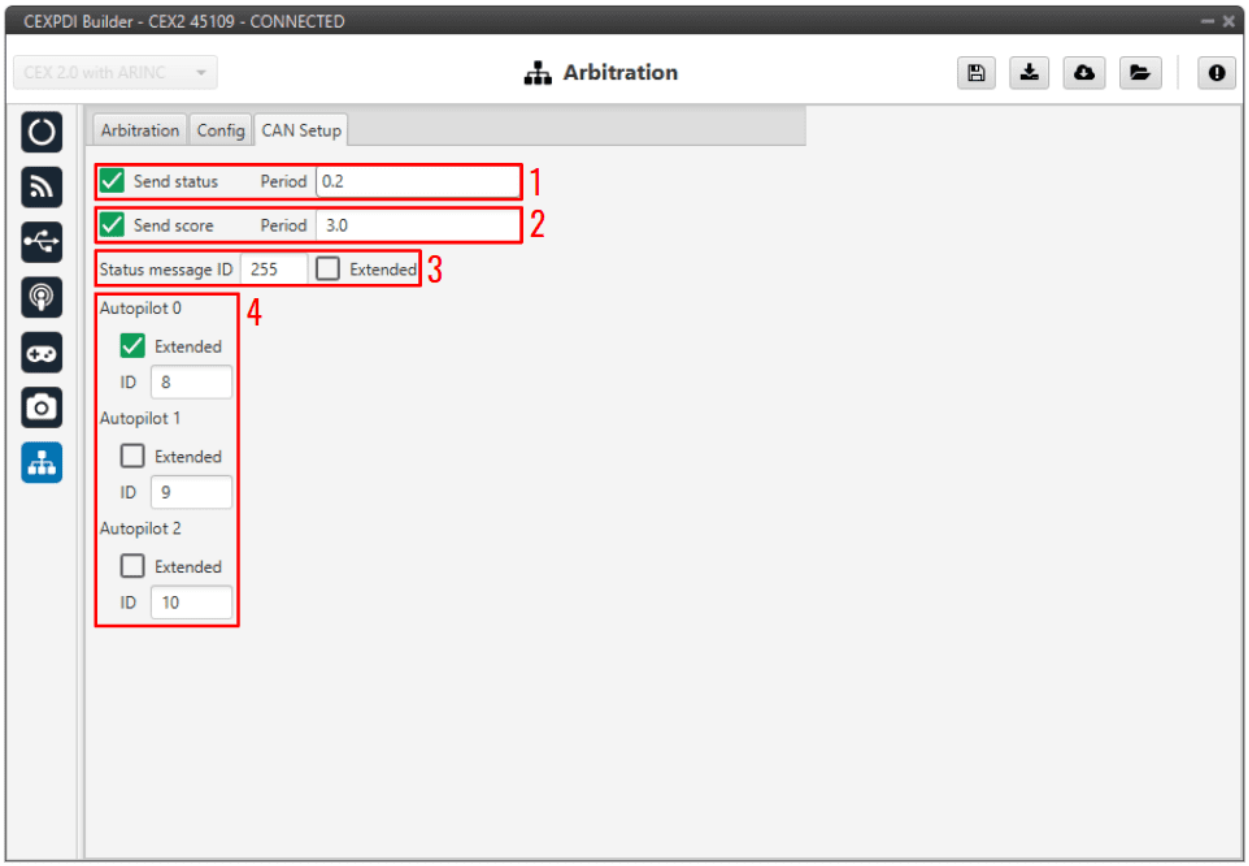


Fig. 68: **CAN Setup panel**

1. **Send status and Period**: The user can enable status sending and set a status message period.

   Note: The *Send Status Period* must be set to **0.2 s**, otherwise, the arbitration algorithm will force the selection of autopilot 0.

   ---

2. **Send score and Period**: Score sending can be enabled and a period of scoring messages can be set.
3. **Status message ID**: CAN ID to which Status and Score messages are sent when there is **no External Arbitration** and therefore **CEX** works as **arbiter**.
   - **Extended**: If enabled, the frame format will be this, '**Extended**', i.e. with a 29-bit identifier. Otherwise, the frame format 'Standard' (11-bit identifier) is set by default.
4. **Autopilot 0-2**: CAN IDs used for the reception of **Autopilot 4x** arbitration messages for each AP.
   - **Extended**: If enabled, the frame format will be this, '**Extended**', i.e. with a 29-bit identifier. Otherwise, the frame format 'Standard' (11-bit identifier) is set by default.

CHAPTER

THREE

INTEGRATION EXAMPLES

3.1 CAN communication

CEX can send and receive messages via CAN.

3.1.1 CAN messages transmission

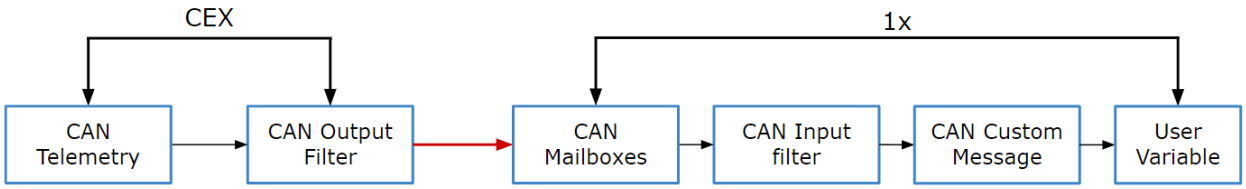


Fig. 1: **CAN messages transmission - Communication diagram CEX → 1x**

Users must follow the steps below to send telemetry via CAN and configure an **Autopilot 1x** to read this telemetry:

1. Go to Input/Output menu → CAN I/O panel → **Configuration tab**.

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

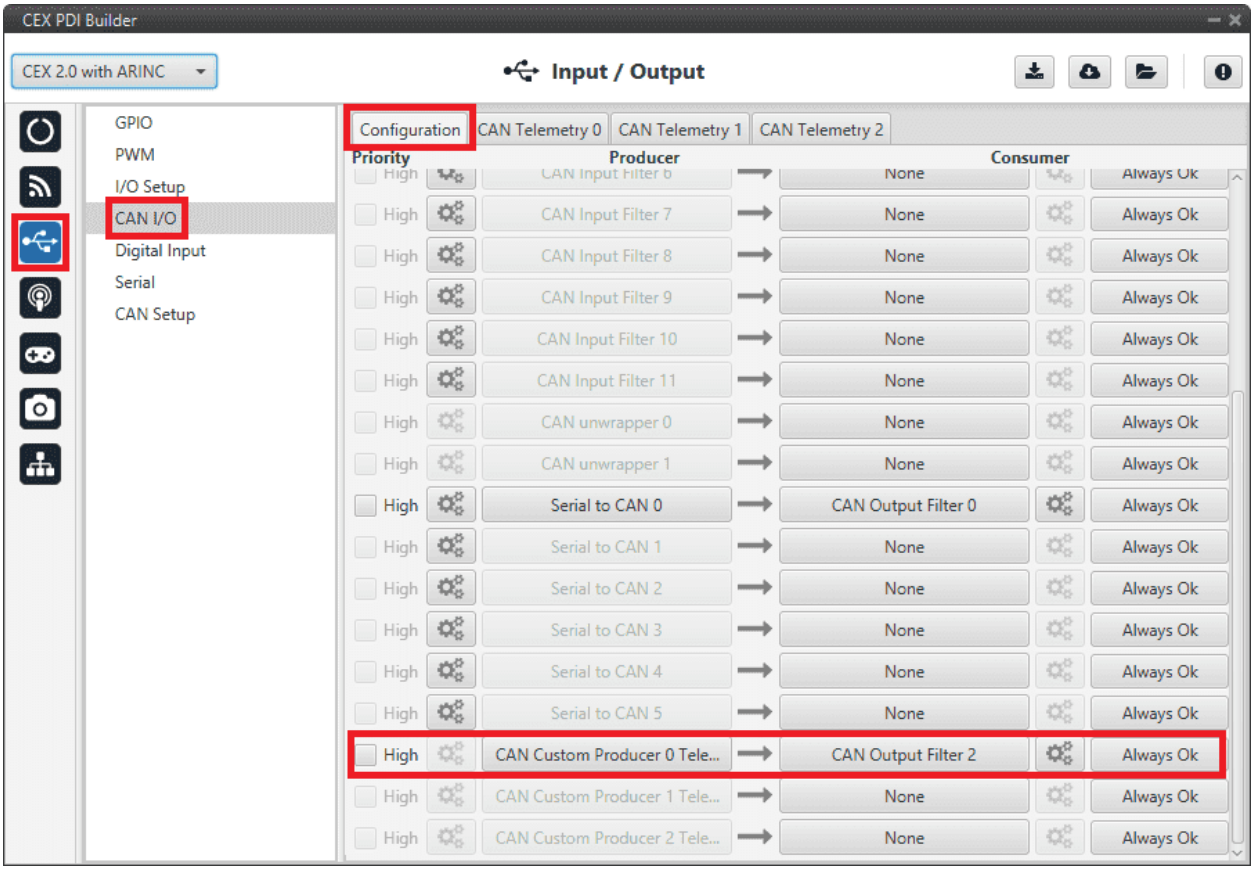


Fig. 2: **CAN messages transmission - CAN I/O configuration**

2. Go to Input/Output menu → CAN I/O panel → **CAN Telemetry tab**.

   This tab must corresponds to the selected CAN Custom Producer Telemetry. In this case CAN Telemetry **0** as the Producer is CAN Custom Producer **0** Telemetry.

   Select the fields to send in **TX** or TX Ini, as it is a Producer. More information on the configuration of telemetry messages can be found in the *CAN Telemetry - Input/Output* section of this manual.

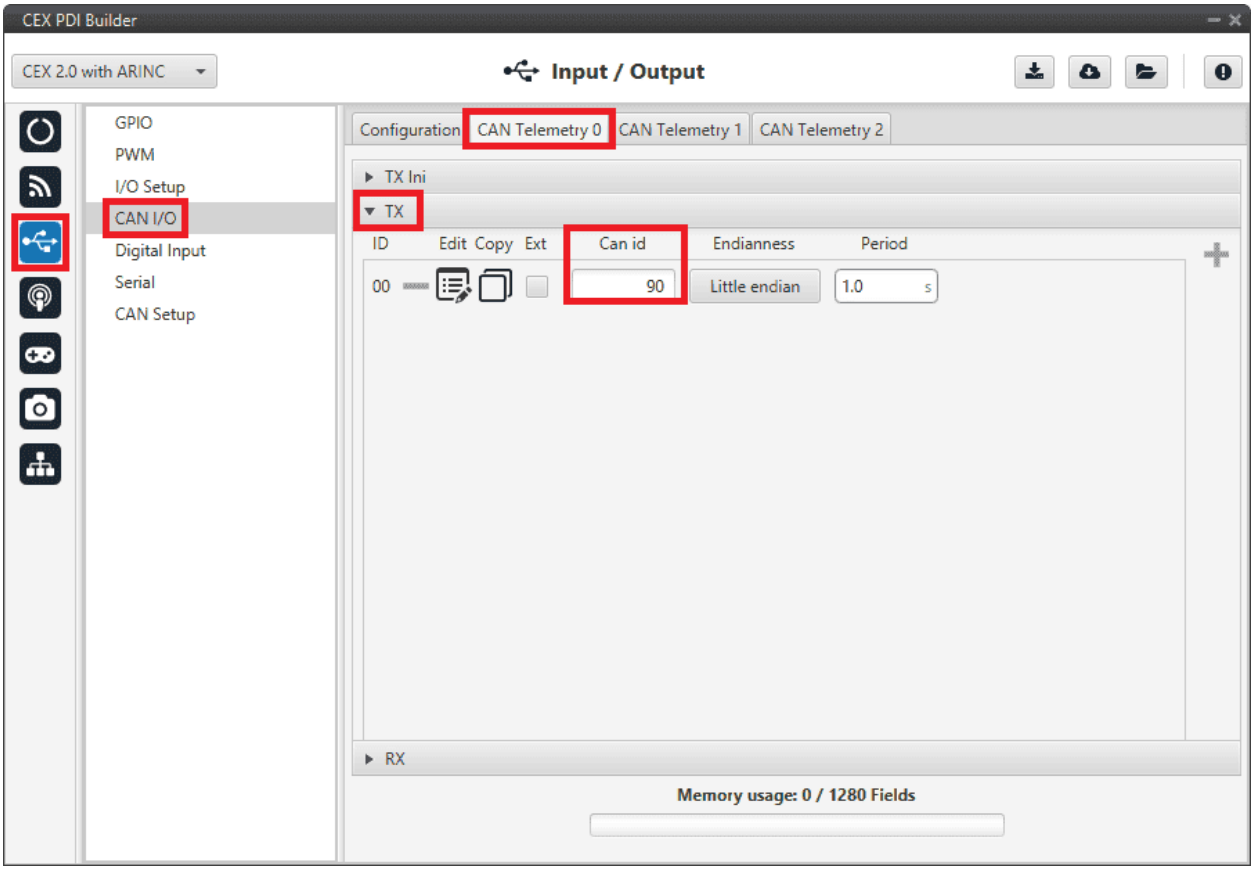


Fig. 3: **CAN messages transmission - CAN Telemetry configuration**

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

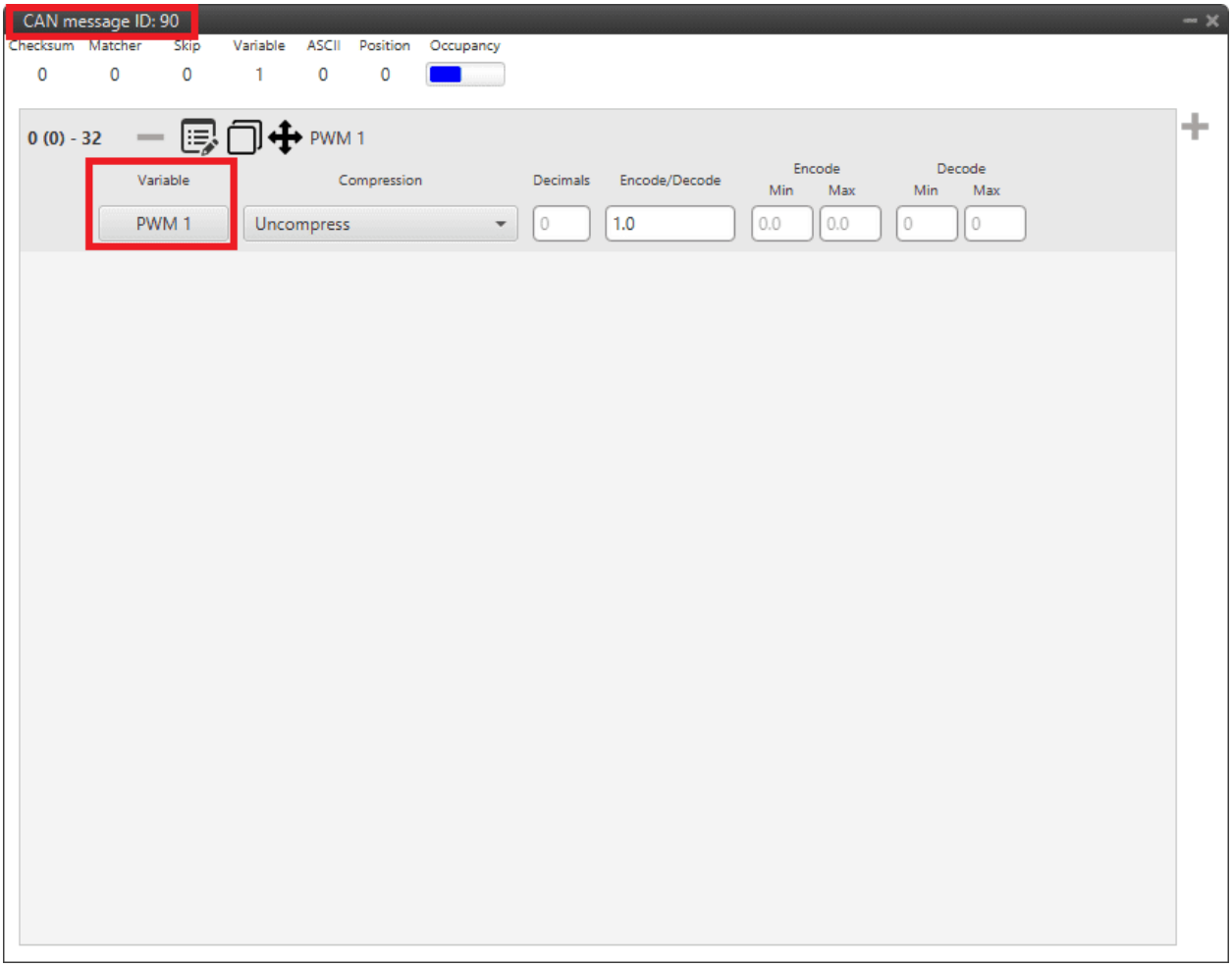


Fig. 4: **CAN messages transmission - CAN Telemetry example**

3. Configure the **Autopilot 1x** to read telemetry from **CEX**, following the CAN messages reception - Integration examples section of the **1x PDI Builder** user manual.

Note: The configured CAN ID must match the one in **CEX**.

3.1.2 CAN messages reception

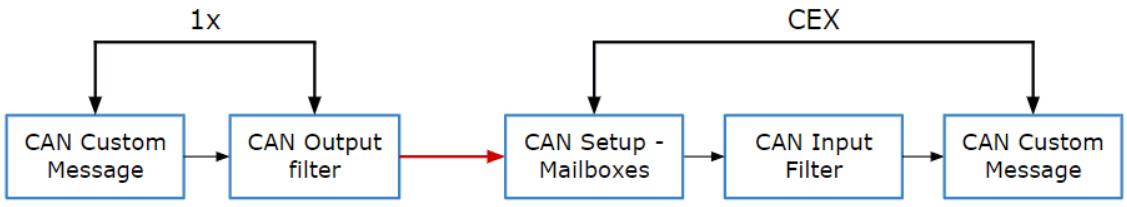


Fig. 5: **CAN messages reception - Communication diagram 1x → CEX**

Users must follow the steps below to read CAN messages via CAN and configure an **Autopilot 1x** to send this telemetry:

1. Configure the **Autopilot 1x** to send telemetry to **CEX**, following the CAN messages transmission - Integration examples section of the **1x PDI Builder** user manual.
2. Go to Input/Output menu → **CAN Setup panel**.

   Configure a mailbox acccording to the message to receive. In this example, a message with ID 90 is received through **CAN A**:

   Note: The configured ID must match the one in **1x**.

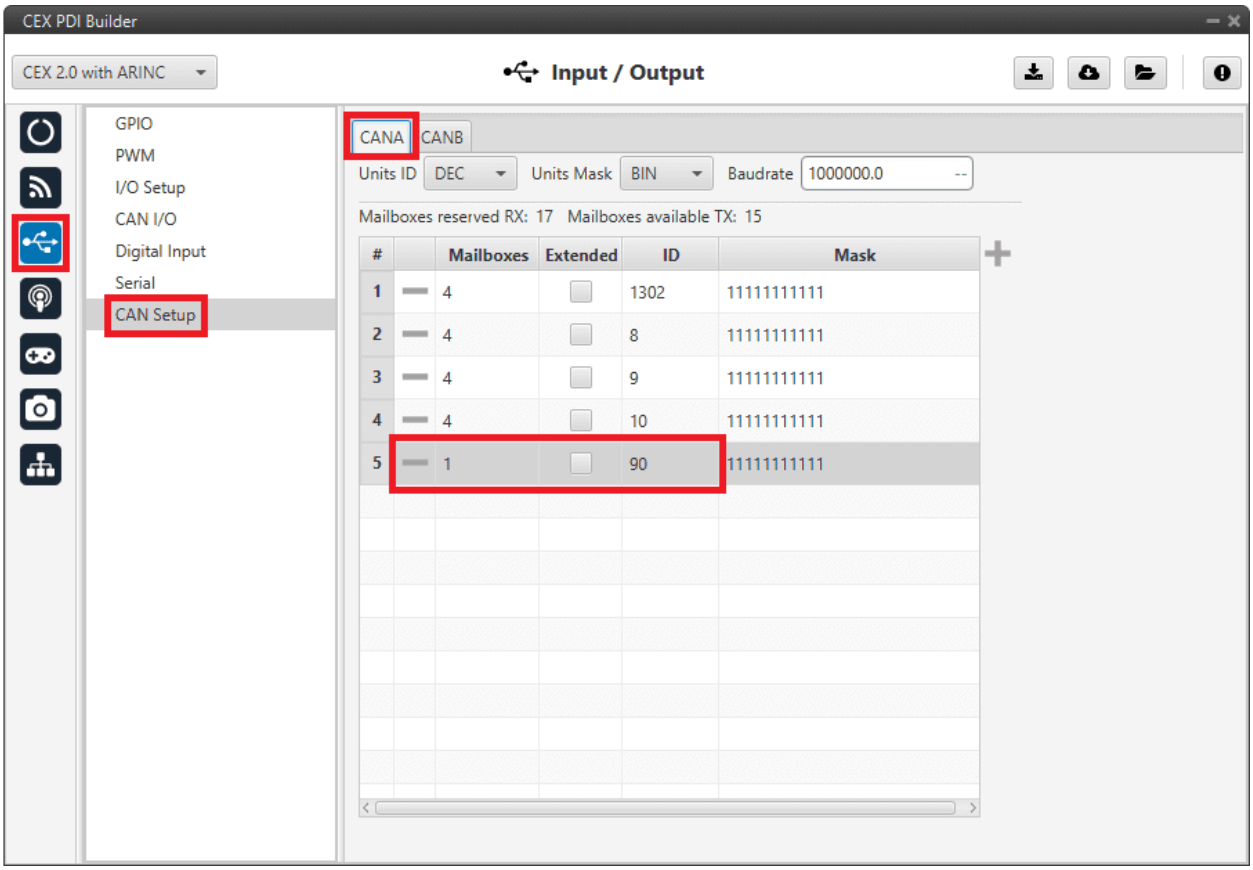


Fig. 6: **CAN messages reception - CAN Setup configuration**

3. Go to Input/Output menu → CAN I/O panel → **Configuration tab**.

   Connect a **CAN Input Filter** to a **CAN Custom Consumer Telemetry** as follows:

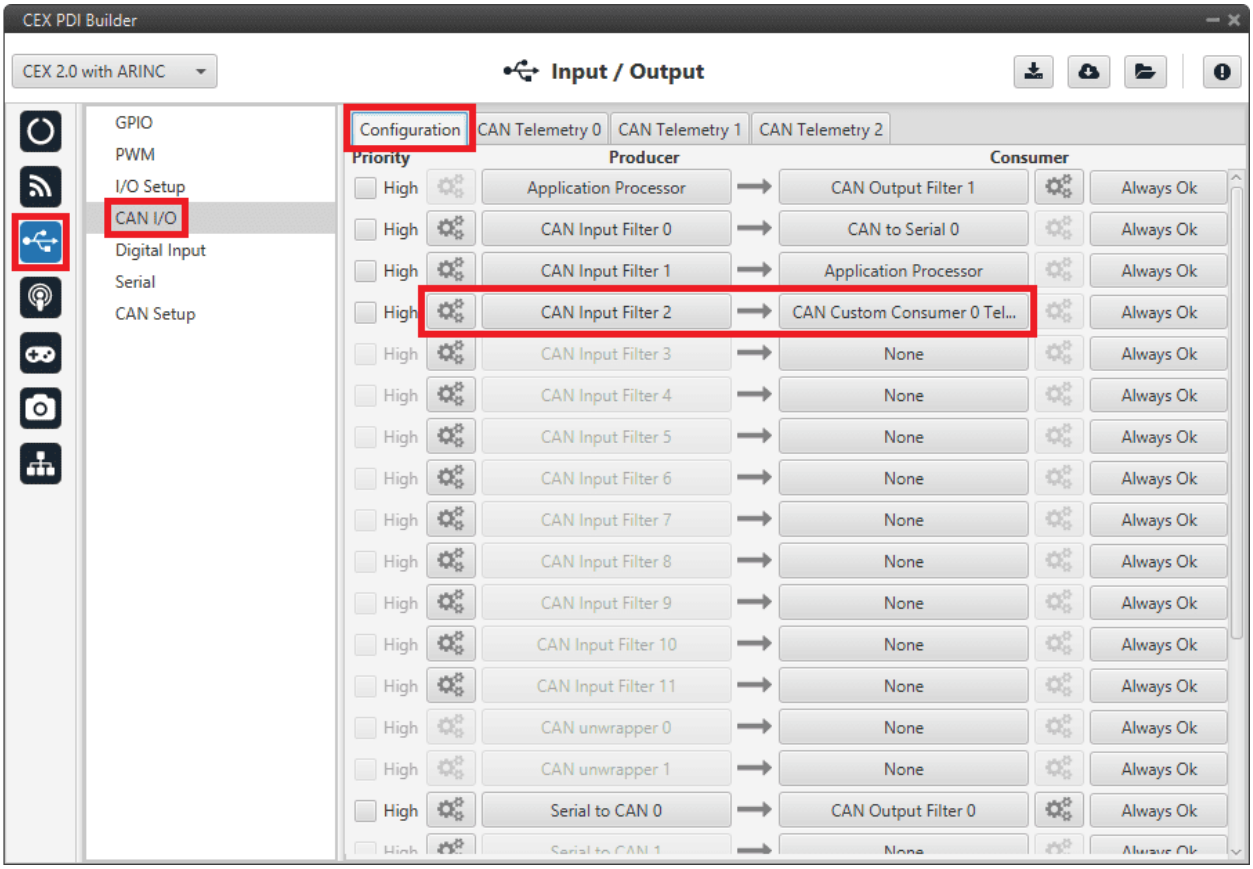


Fig. 7: **CAN messages reception - CAN I/O configuration**

4. Click on the  button of the **CAN Input Filter** and configure the *Port* and *CAN Id* according to the message to receive.

Note: The configured CAN Id must match the one in **1x**.

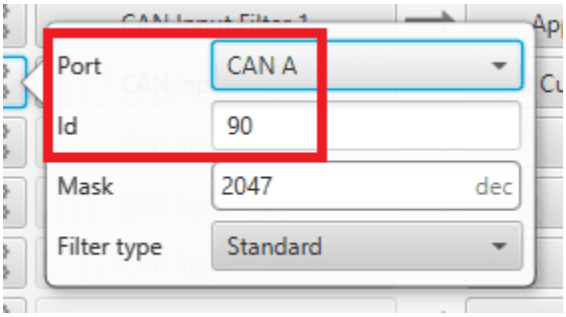


Fig. 8: **CAN messages reception - CAN Input Filter configuration**

5. Go to Input/Output menu → CAN I/O panel → **CAN Telemetry 0 tab** (as CAN Custom Consumer **0** Telemetry has been selected as consumer).

   Add a field to receive the message in **RX**, since it is a Consumer, and configure it to store the received value in a

user variable. More information on the configuration of Telemetry messages can be found in the *CAN Telemetry - Input/Output* section of this manual.

Note: The configured CAN Id must match the one in **1x**.

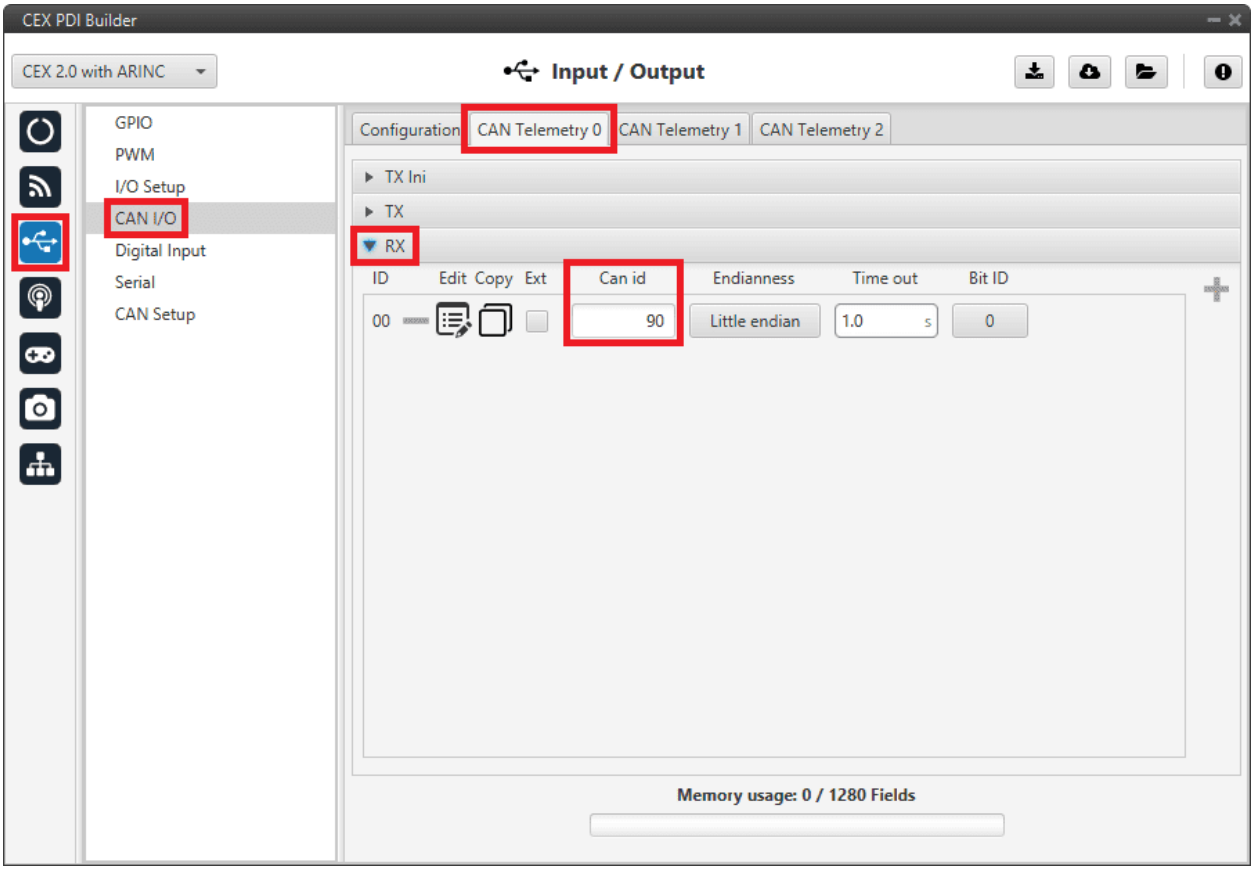


Fig. 9: **CAN messages reception - CAN Telemetry configuration**

3.2 CAN Isolator

CEX 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*

The following diagram summarizes the behavior of CEX as a CAN bus isolator:

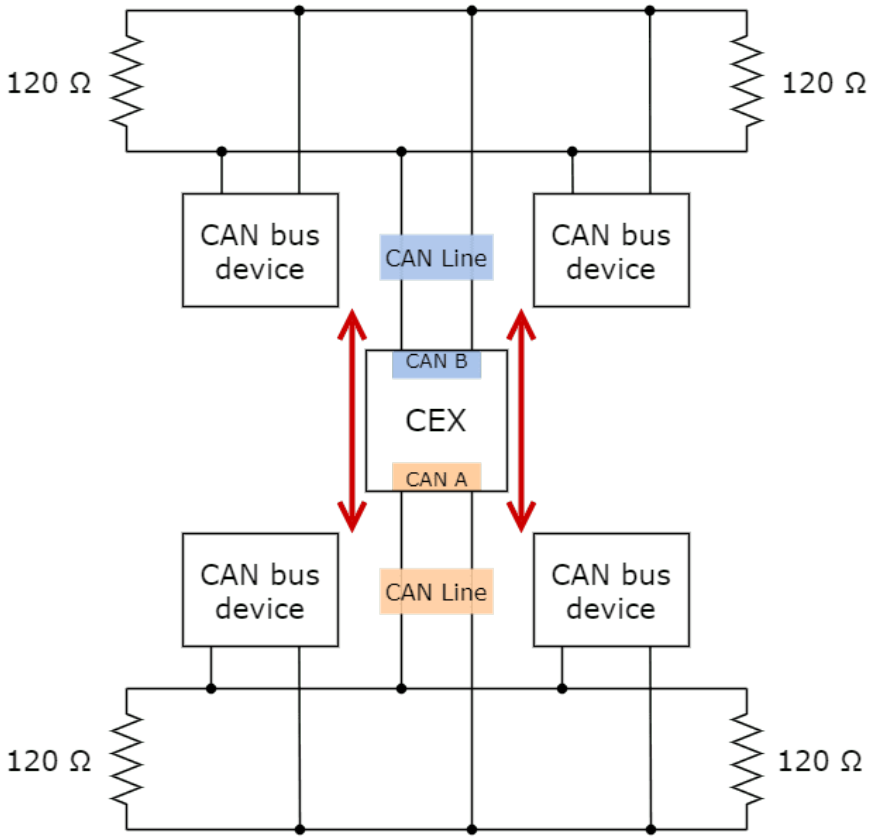


Fig. 10: **Subnets CAN diagram**

3.2.1 Filtered CAN subnet

With **CEX** it is possible to isolate CAN nets, by filtering certain messages from one CAN line and only transmitting specific information through CEX to the other one.

In this example, only a certain range of CAN IDs will be allowed to cross from one CAN line to the other using the two CAN ports of **CEX**. Specifically, information will pass **from CAN B** of CEX **to CAN A** of CEX:

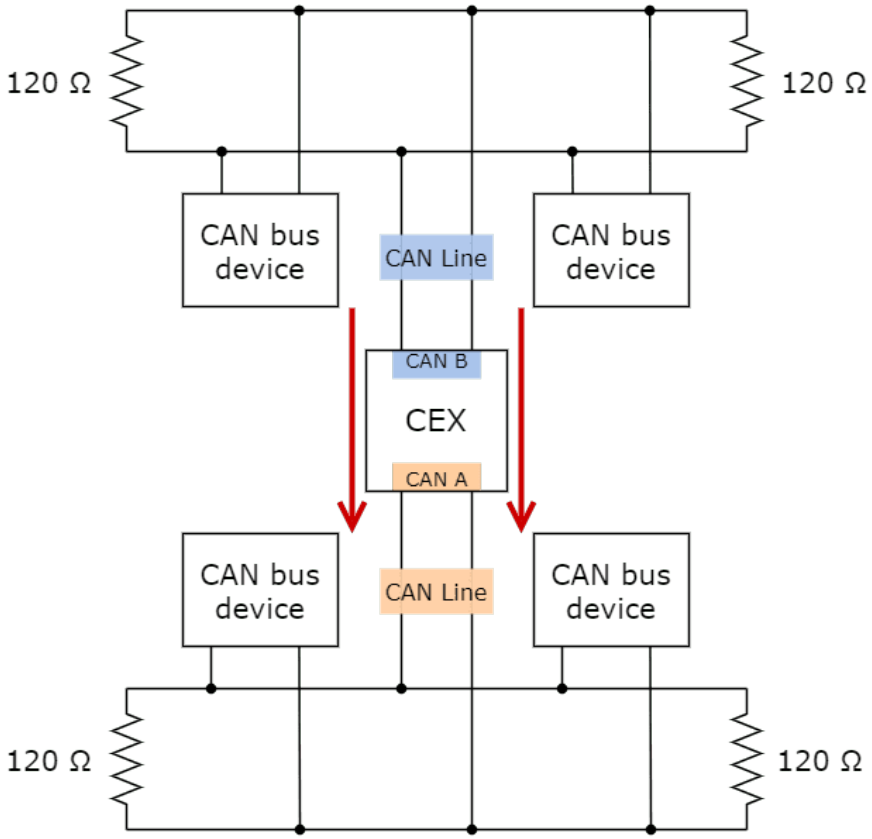


Fig. 11: **Subnets CAN example - From CAN B to CAN A**

The allowed range will be from **0x550** to **0x55F**.

1. Create a new mailbox entry for **CAN B**.

   To do this, go to Input/Output menu → CAN Setup panel → **CAN B tab**.

   Assign some of the mailboxes to it and set the ID to **0x550**. This example uses an ID expressed in **hexadecimal** notation.

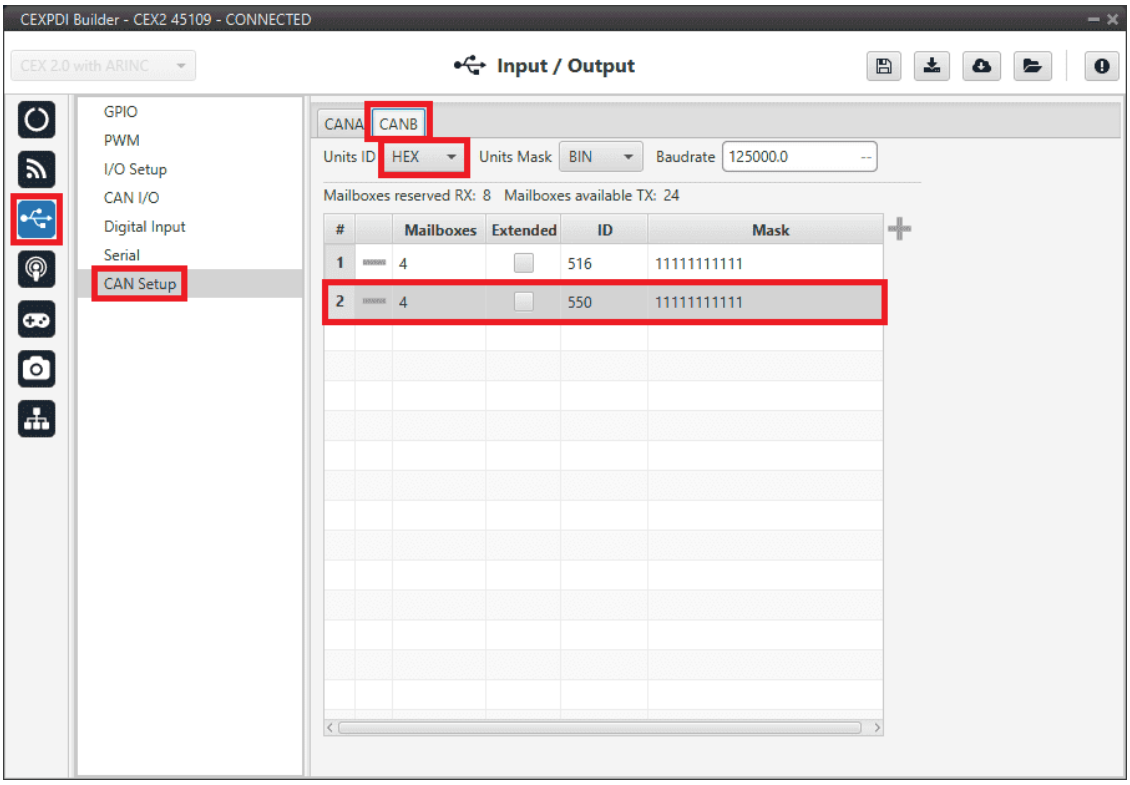


Fig. 12: **Filtered CAN subnet - CAN Setup configuration**

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

Note: Although the ID has been entered in hexadecimal notation, note that the Mask has been set in **binary** format.

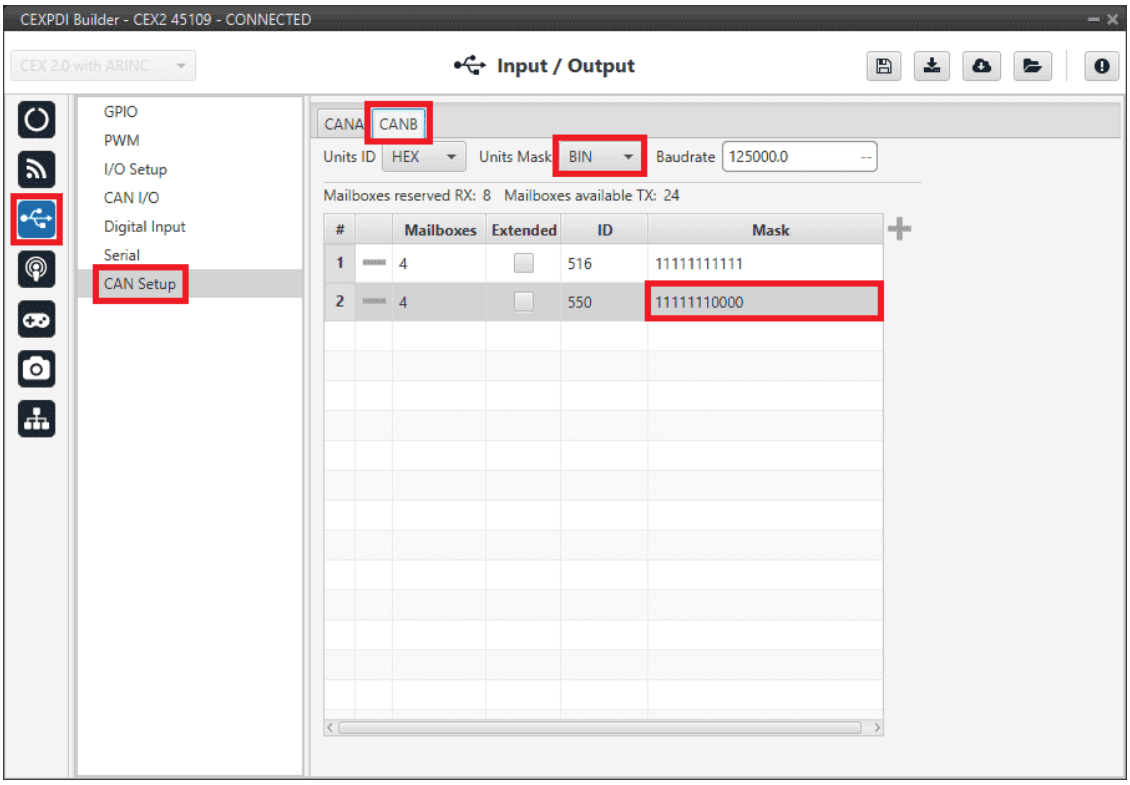


Fig. 13: **Filtered CAN subnet - CAN Setup mask configuration**

3. Go to Input/Output → CAN I/O panel → **Configuration tab**.

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

Important: This menu only allows setting the Id with **decimal numbers**. For this example the Id 0x550 is represented as **1360** in decimal format.

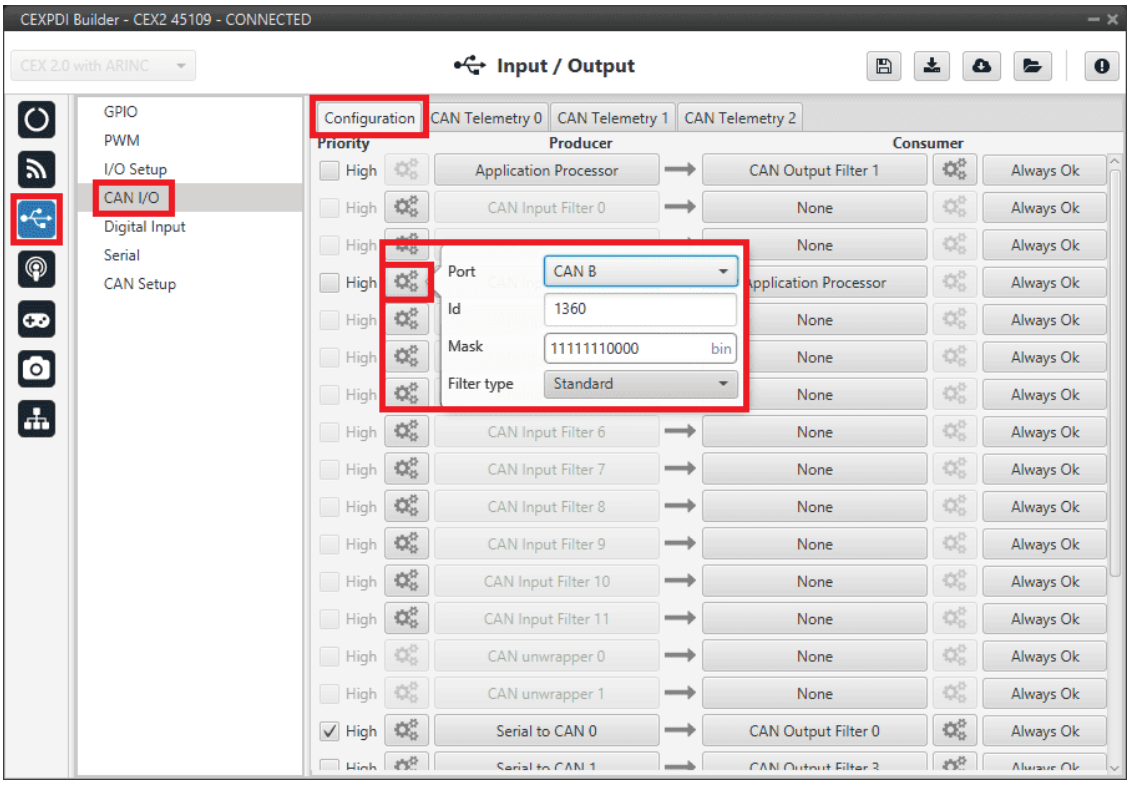


Fig. 14: **Filtered CAN subnet - CAN Input Filter configuration**

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

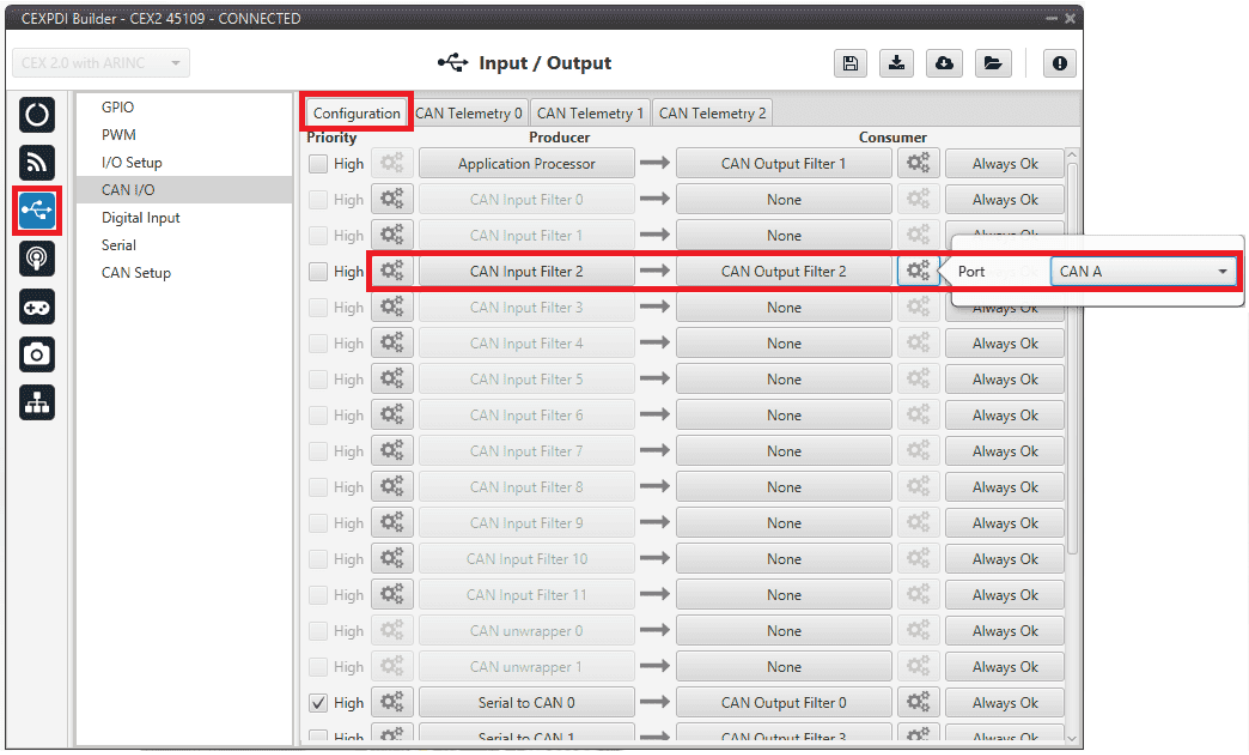


Fig. 15: **Filtered CAN subnet - CAN Output Filter configuration**

3.2.2 CAN tunnel

CAN tunnel is a specific type of *a message tunnel*; where messages are tunneled from one CAN port of CEX to the other.

In this example, a transparent tunnel will be created. So, any messages received on **Interface A** will be sent through **Interface B**.

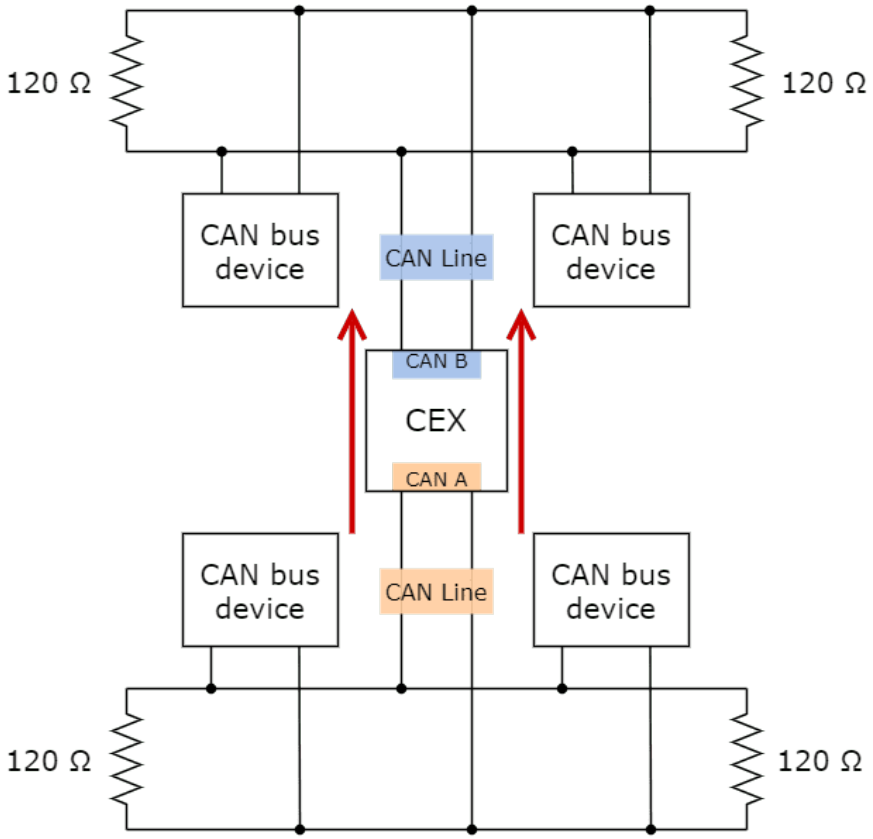


Fig. 16: **Subnets CAN example - From CAN A to CAN B**

Optionally, the mailboxes can be equally distributed to support both standard and extended CAN IDs.

1. Create a new mailbox entry for **Interface A**.

   To do this, go to Input/Output menu → CAN Setup panel → **CAN A tab**.

   Assign half of the mailboxes to it and set a **Mask** of 0.

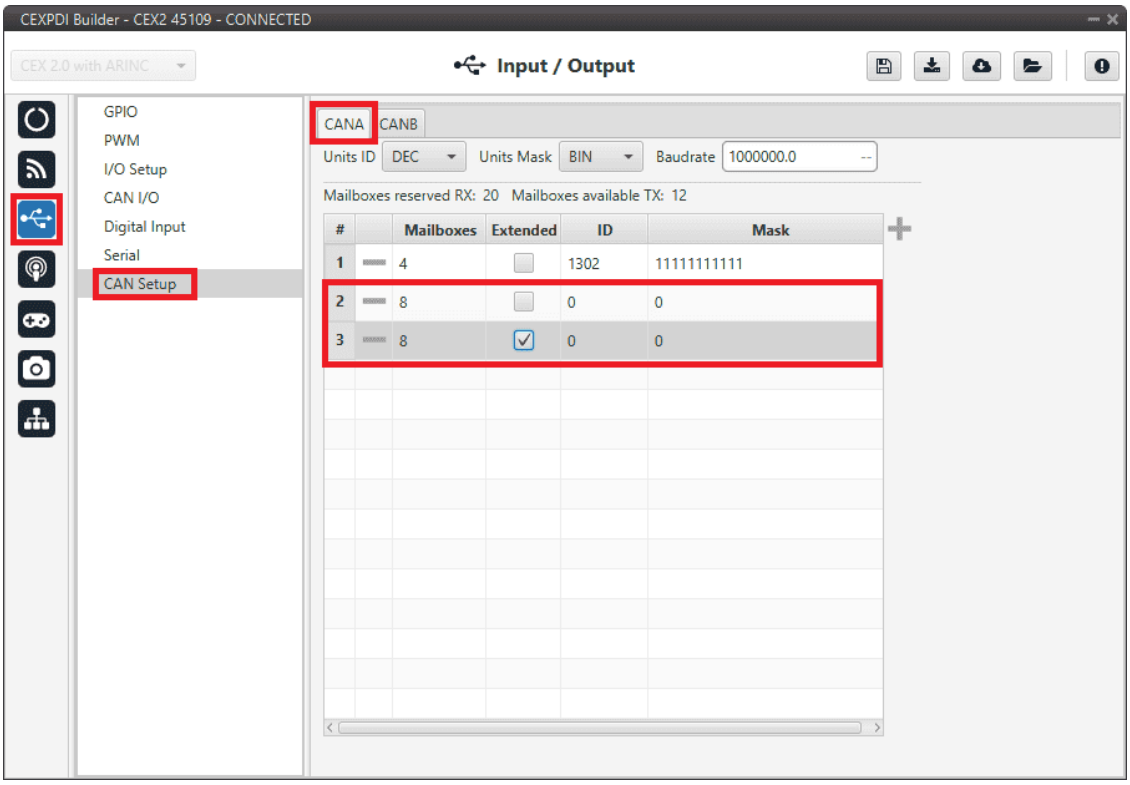


Fig. 17: **CAN tunnel - CAN Setup configuration**

2. Go to Input/Output → CAN I/O panel → **Configuration tab**.

   Configure **CAN Input Filter 2** on **CAN A**, with a **Mask** of 0 and **Both** as frame format type.

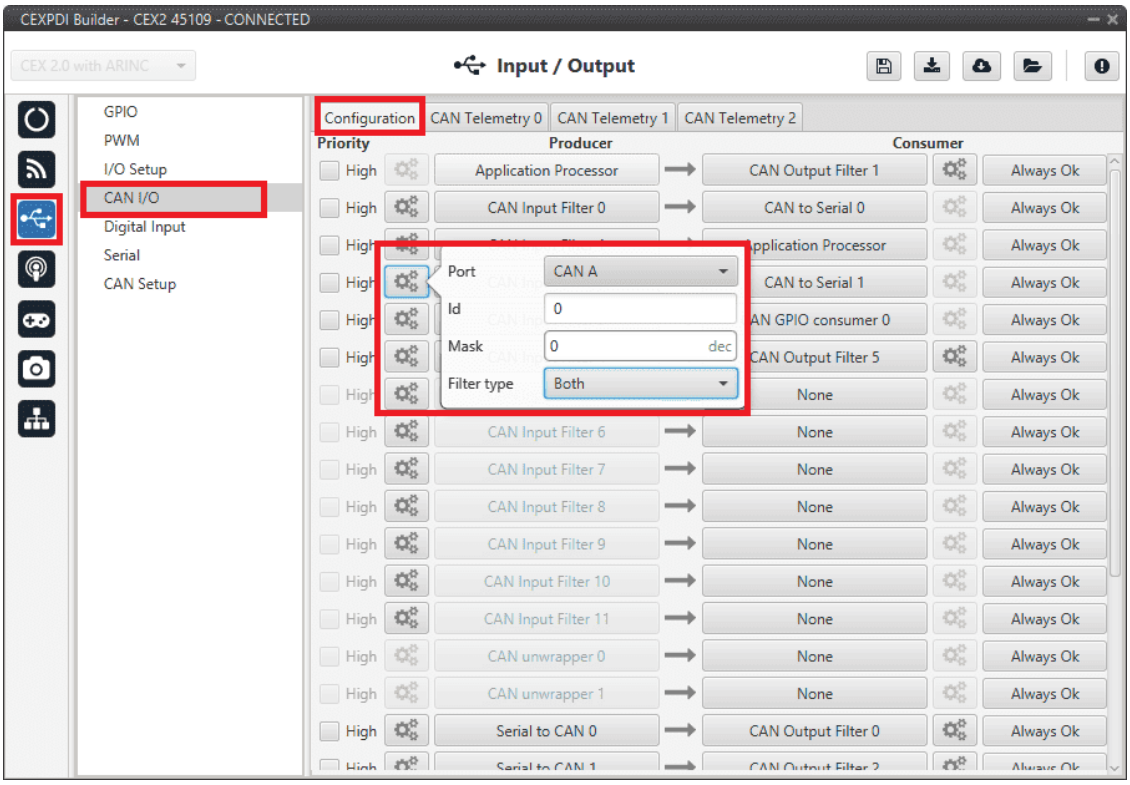


Fig. 18: **CAN tunnel - CAN Input Filter configuration**

3. Bind **CAN Output Filter 2** to **CAN Input Filter 2**, and configure the CAN Output Filter to **CAN B**.

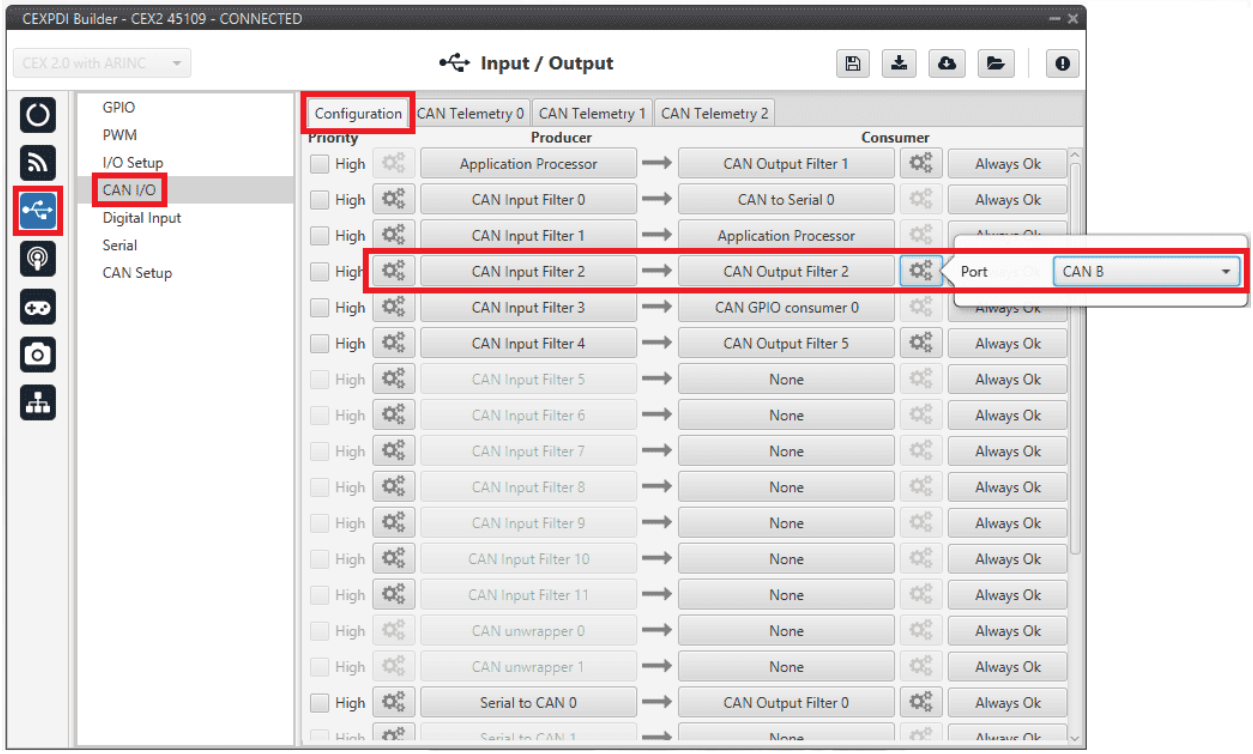


Fig. 19: **CAN tunnel - CAN Output Filter configuration**

3.3 Commanding/Reading PWMs

The appropriate PWM message format is described in the Command PWMs - CAN Bus Protocol section of **CEX Software Manual**.

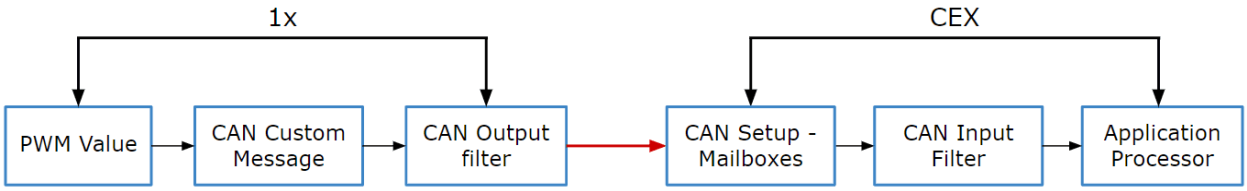


Fig. 20: **PWM Command - Communication diagram 1x → CEX**

The user can follow the following steps to achieve PWM commanding from 1x to CEX:

3.3.1 1x PDI Builder side

1. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

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

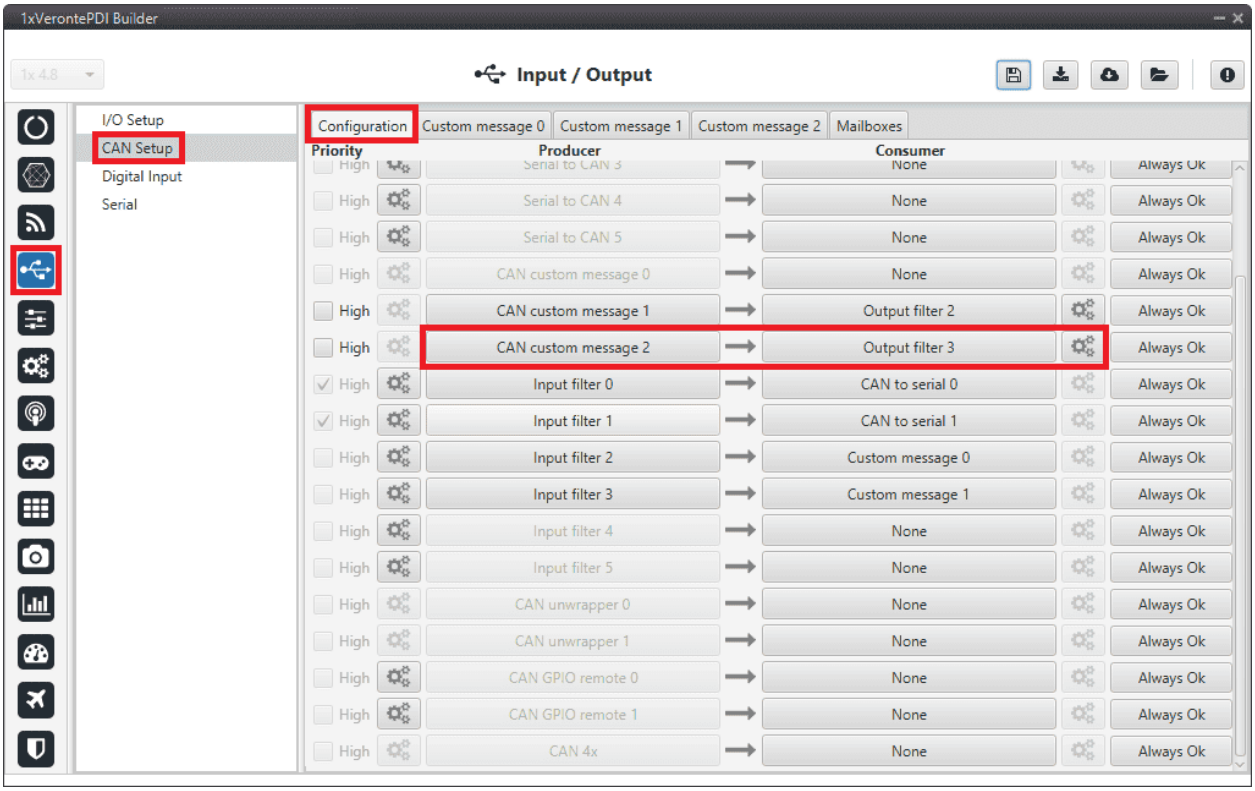


Fig. 21: **1x PDI Builder - CAN Setup configuration**

2. Go to Input/Output menu → CAN Setup panel → **Custom Message 2 tab** (as CAN custom Message **2** has been selected as producer).

   Build a CAN custom message using the PWM variable and the appropriate message format.

   Select the fields to send in the **TX** as it is a Producer. CAN ID is arbitrary, for this example ID 100 will be used:

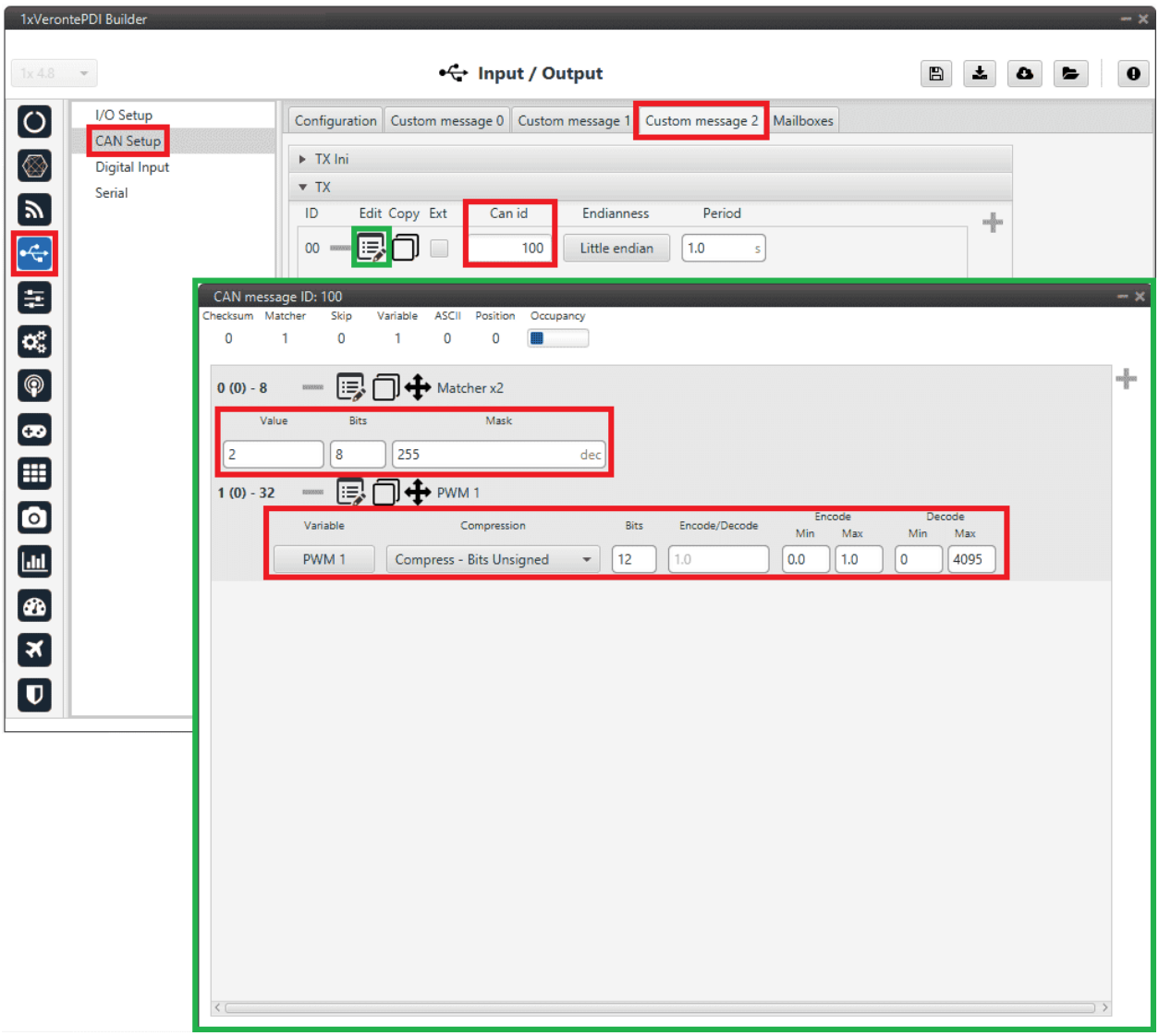


Fig. 22: **1x PDI Builder - CAN Custom message configuration**

Message format:

- Matcher (2) [8 bits]
  - PWM 0 [12 bits]
  - PWM 1 [12 bits]
  - PWM 2 [12 bits]
  - PWM 3 [12 bits]
- Matcher (3) [8 bits]
  - PWM 4 [12 bits]
  - PWM 5 [12 bits]
  - PWM 6 [12 bits]

  - PWM 7 [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

3.3.2 CEX PDI Builder side

3. Go to Input/Output menu → **CAN Setup panel**.

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

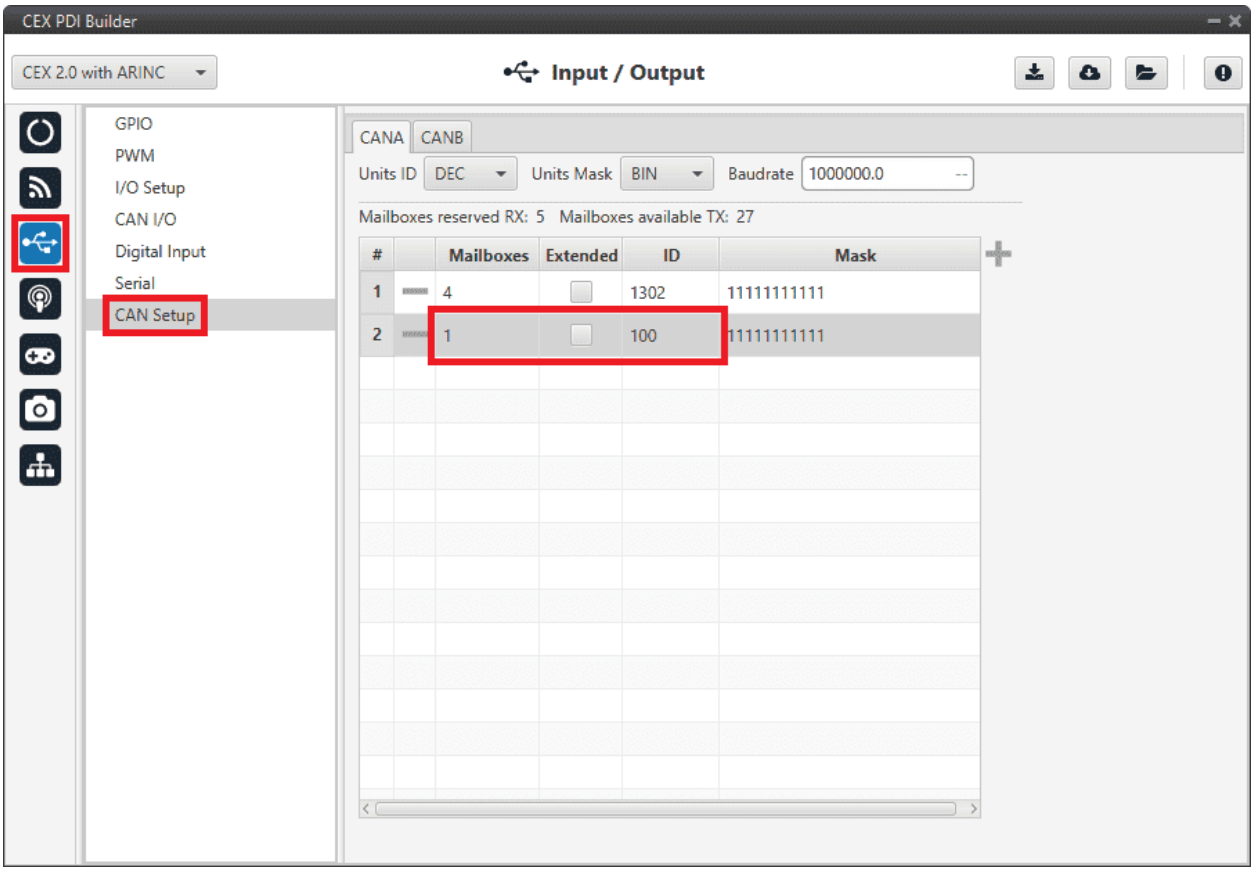


Fig. 23: **CEX PDI Builder - CAN Setup configuration**

4. Go to Input/Output menu → CAN I/O panel → **Configuration tab**.

   Connect a **CAN Input Filter** with the **right CAN ID** to the **Application Processor consumer**:

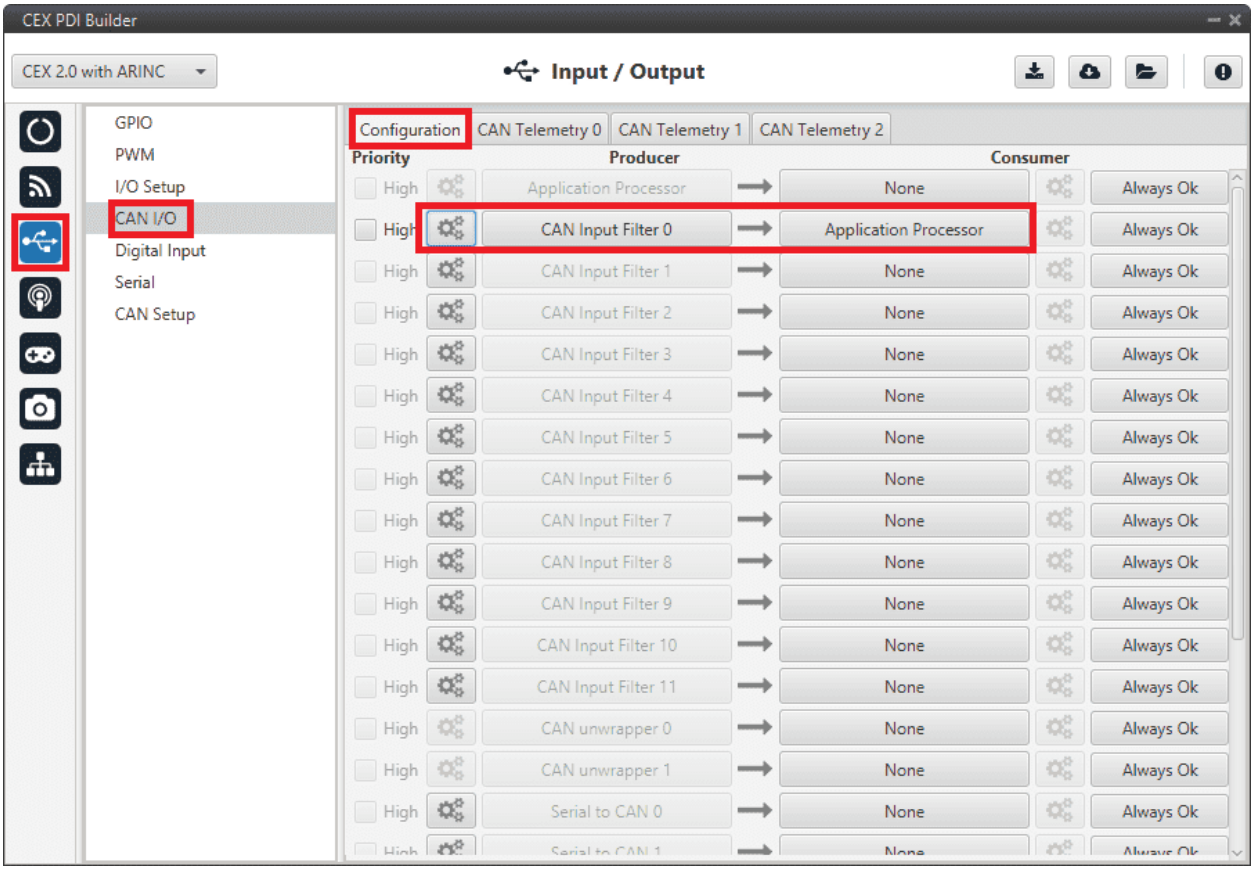


Fig. 24: **CEX PDI Builder - CAN I/O configuration**

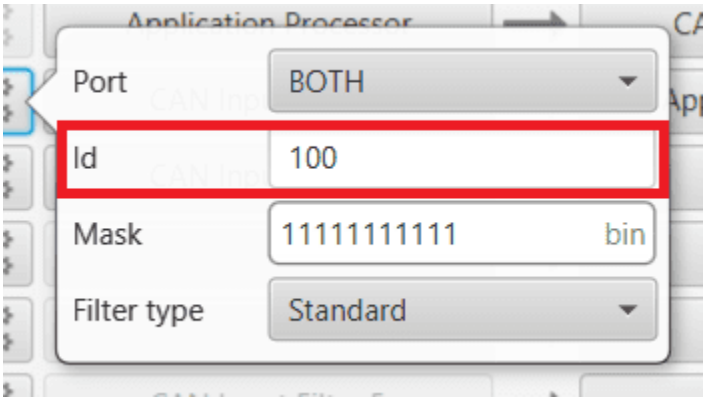


Fig. 25: **CEX PDI Builder - CAN Input Filter configuration**

5. Go to Input/Output menu → PWM panel → **PWM 1 tab** (as PWM **1** has been selected in **1x PDI Builder**). Configure parameters for PWM 1:

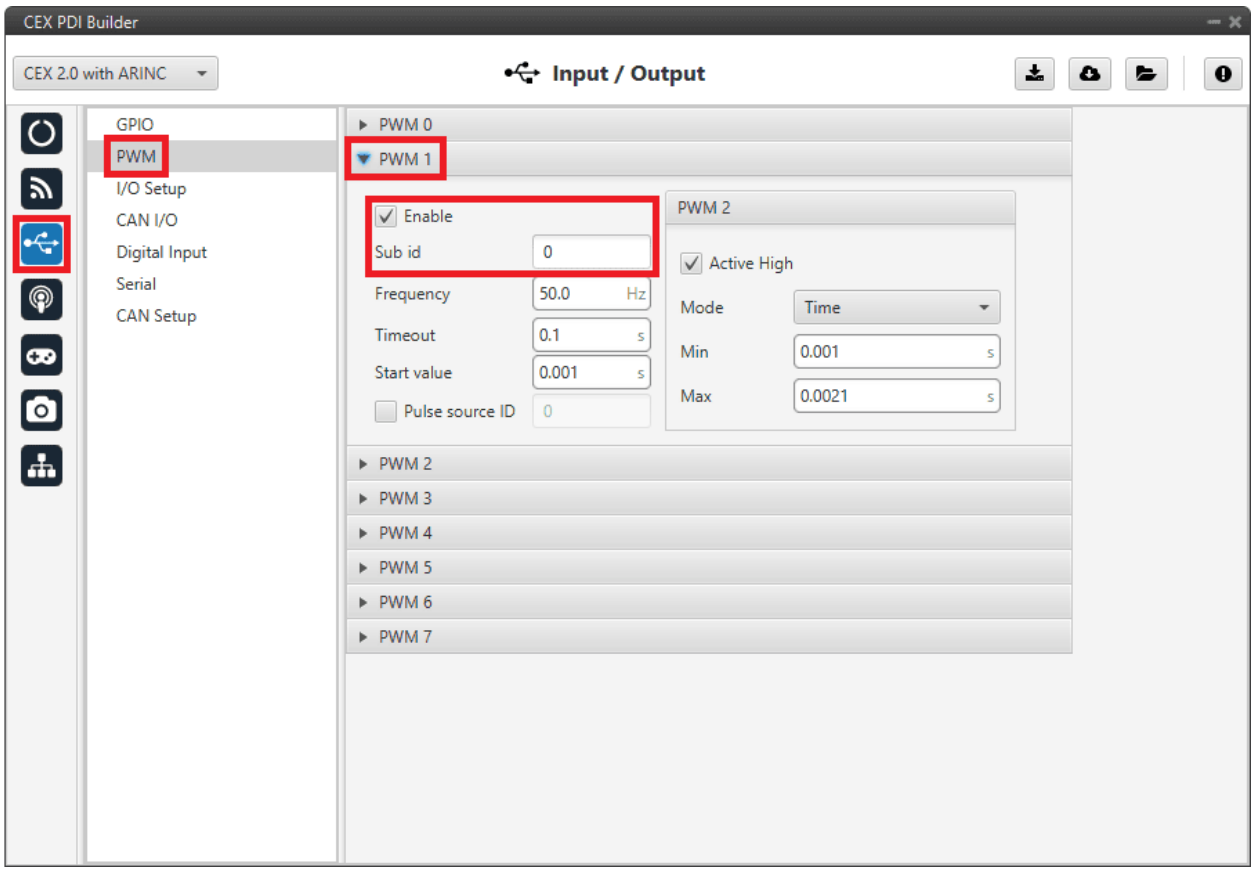


Fig. 26: **CEX PDI Builder - PWM configuration**

6. Go to Input/Output menu → **GPIO panel**.

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

   It should look like this:

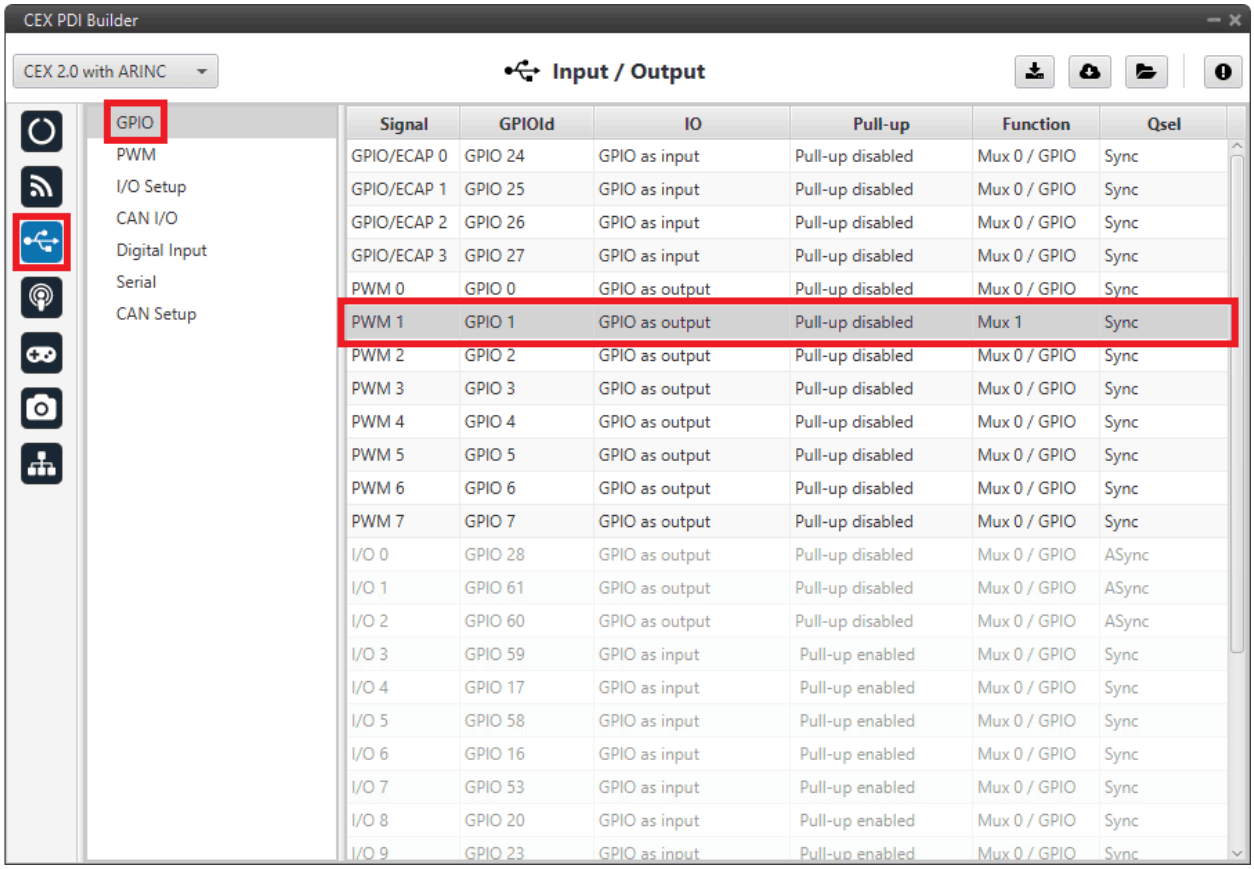


Fig. 27: **CEX PDI Builder - GPIO configuration**

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

7. Finally, save the changes.

3.4 GPIO Command

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

> **Warning:** Remember that for the reception of CAN messages, Mailboxes need to be configured accordingly.

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

3.4.1 1x PDI Builder side

1. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

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

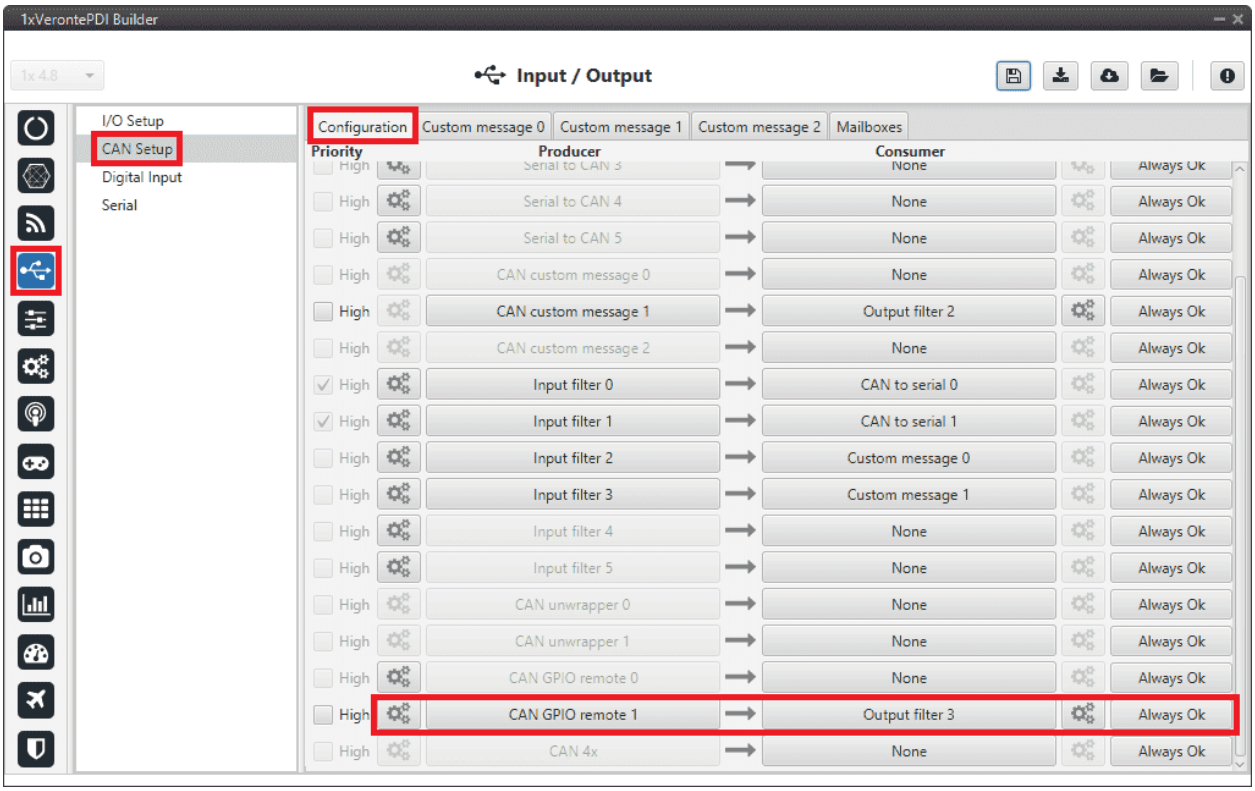


Fig. 28: **1x PDI Builder - CAN Setup configuration**

CAN GPIO remote must be configured. For more information on its configuration, see CAN Setup - Input/Output section of the **1x PDI Builder** user manual.

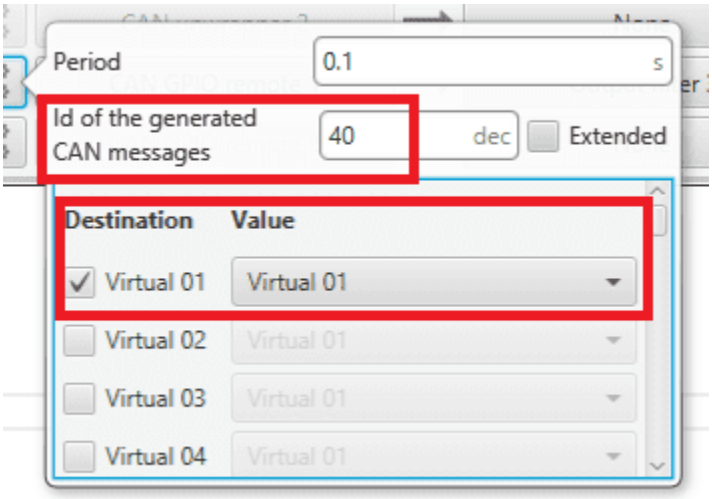


Fig. 29: **1x PDI Builder - CAN GPIO remote configuration**

2. Go to **Automations** menu.

   GPIO must be activated using an '**Output action**':

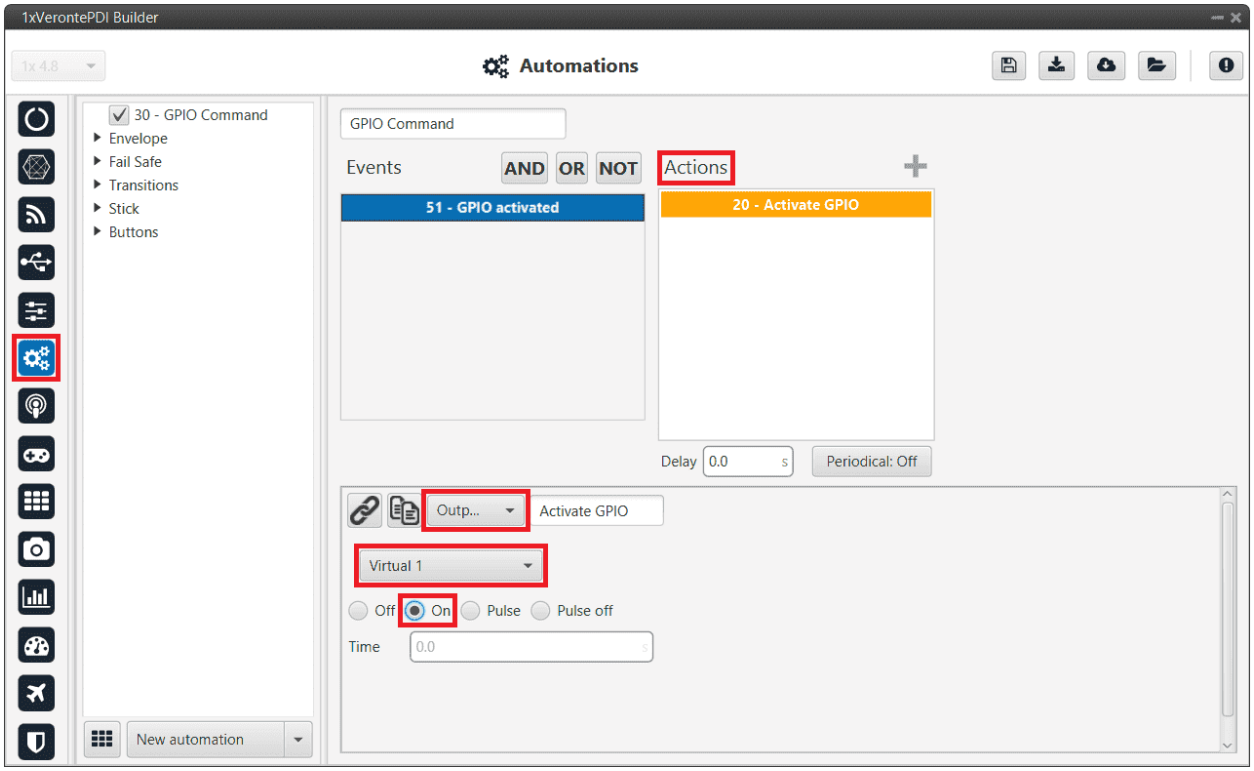


Fig. 30: **1x PDI Builder - Automation configuration**

3.4.2 CEX PDI Builder side

3. Go to Input/Output menu → CAN I/O panel → **Configuration tab**.

   Finally, connect a **CAN Input Filter** to the **CAN GPIO consumer**:

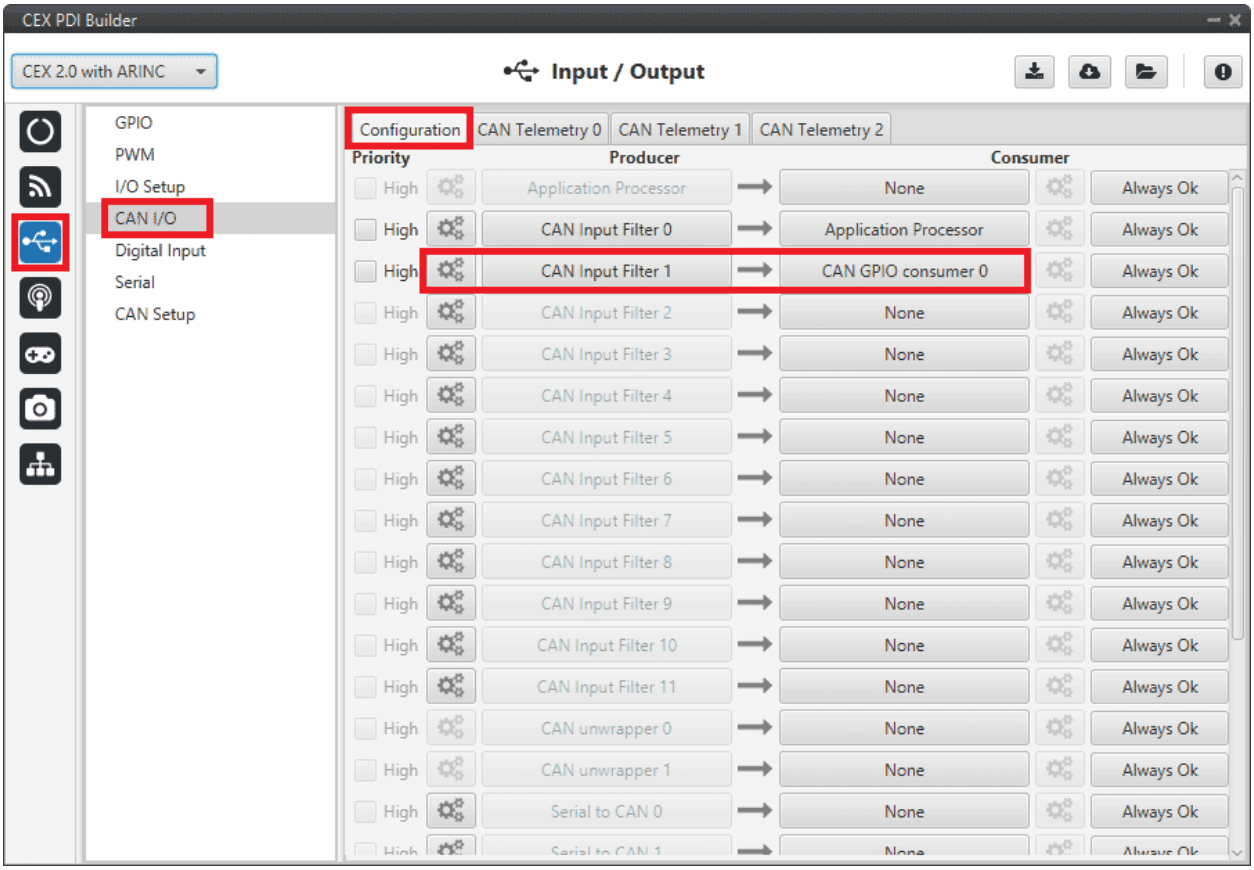


Fig. 31: **CEX PDI Builder - CAN I/O configuration**

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

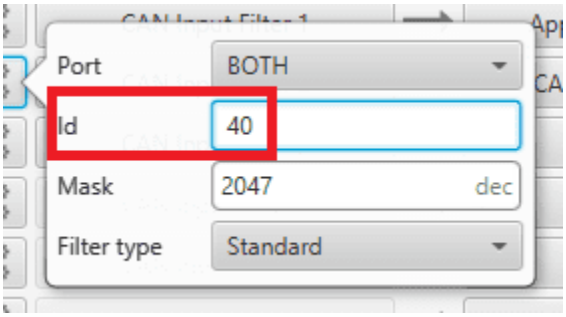


Fig. 32: **CEX PDI Builder - CAN Input Filter configuration**

3.5 Reading/Sending RPMs

This section presents the steps to follow to read RPMs.

3.5.1 CEX PDI Builder side

1. Go to Input/Output menu → **GPIO panel**.

   RPM can be read on the available digital inputs GPIO/ECAP 0-3. The chosen pin needs to be configured as "**GPIO as input**". In the example shown here, GPIO/ECAP 1 is chosen.

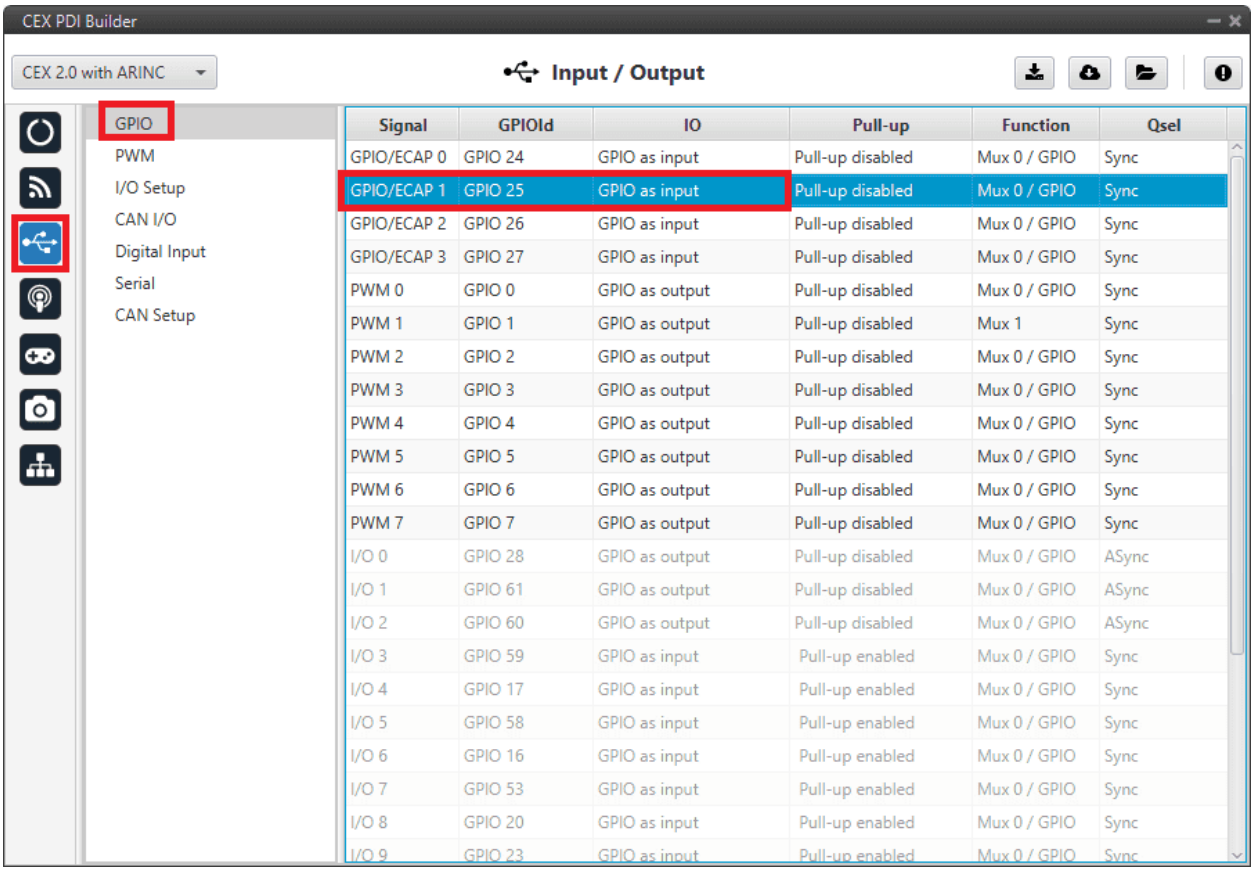


Fig. 33: **CEX PDI Builder - GPIO configuration**

2. Go to Input/Output menu → **Digital Input panel**.

   There are 4 possible producers: CAP 0-3. One needs to be chosen and linked to one of the RPMs consumers (RPMs 0-3). For more information on Digital input configuration, see *Digital Input - Input/Output* section of this manual.

   Then, select an **edge dectection option**: "First rising edge" or "First falling edge".

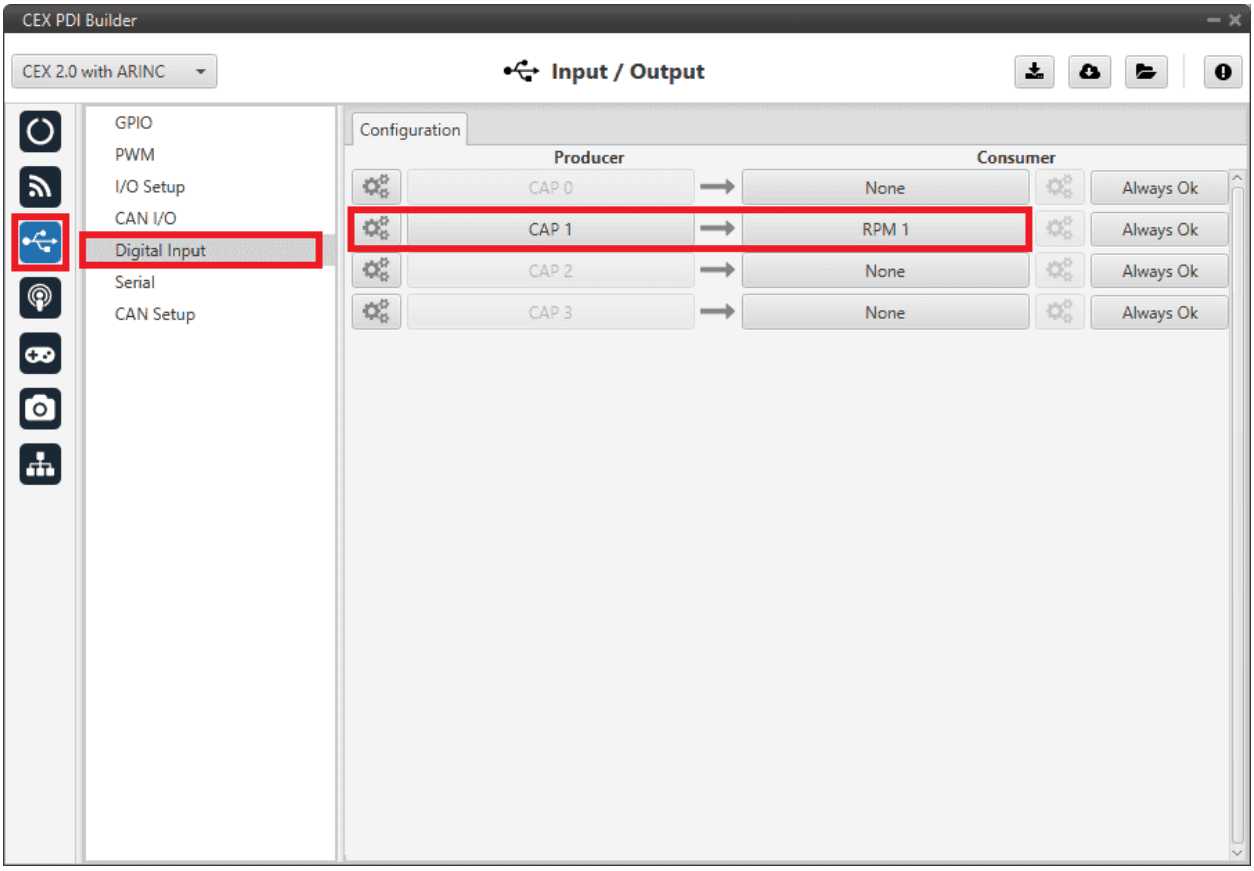


Fig. 34: **CEX PDI Builder - Digital Input configuration**

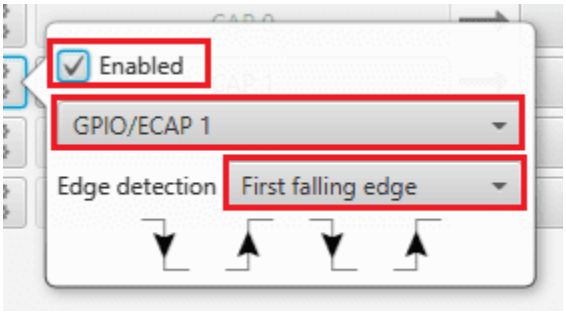


Fig. 35: **CEX PDI Builder - Edge detection configuration**

3. Go to Sensors menu → RPM panel → **RPM 1 tab** (as **RPM 1** has been selected in the Digital Input panel).

   Here the expected pulse needs to be defined. For more information on RPM configuration, see *RPM - Sensors* section of this manual.

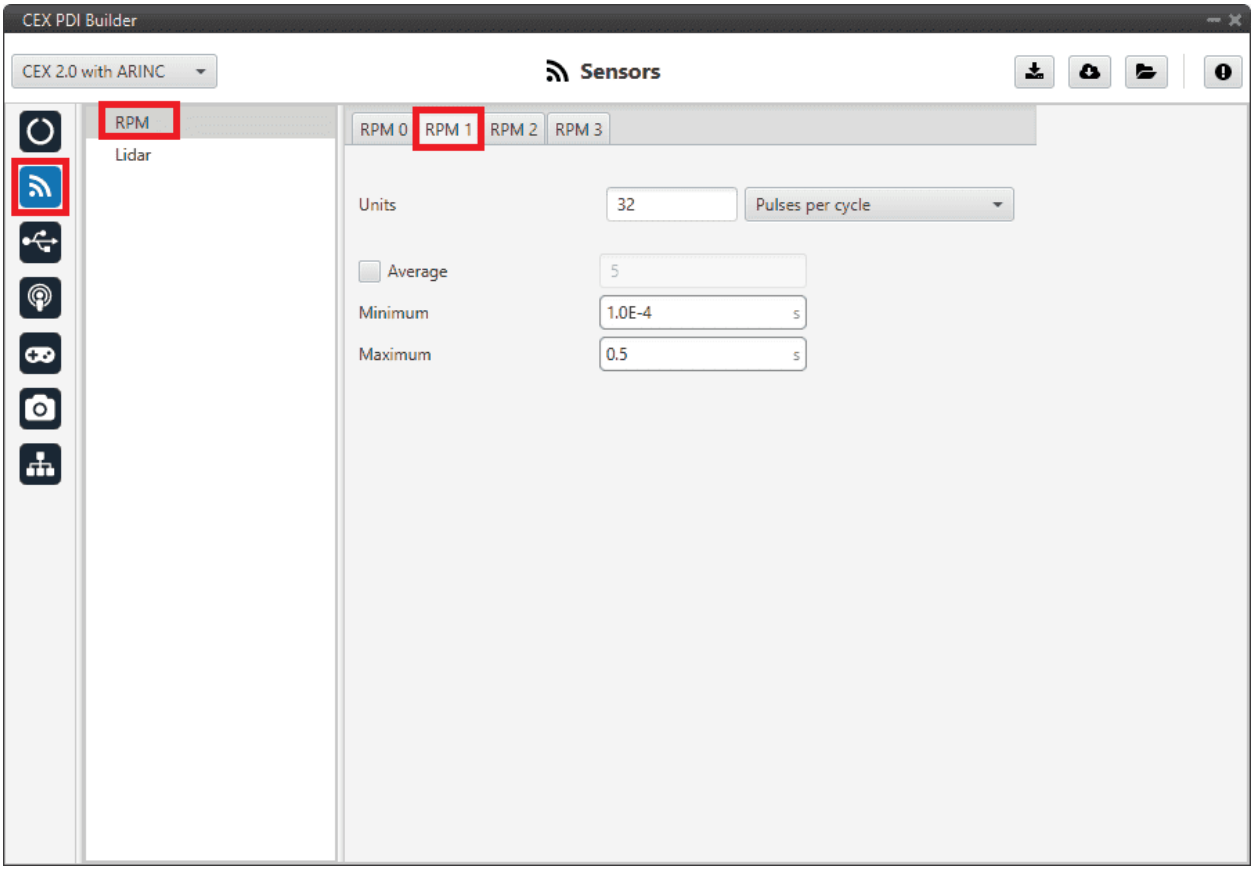


Fig. 36: **CEX PDI Builder - RPM configuration**

4. Go to Input/Output menu → CAN I/O panel → **Configuration tab**.

   Connect a **CAN Custom Producer Telemetry** to a **CAN Output Filter** as follows (in this example, the **RMP1** variable is sent via **CAN B** bus of the **CEX**):

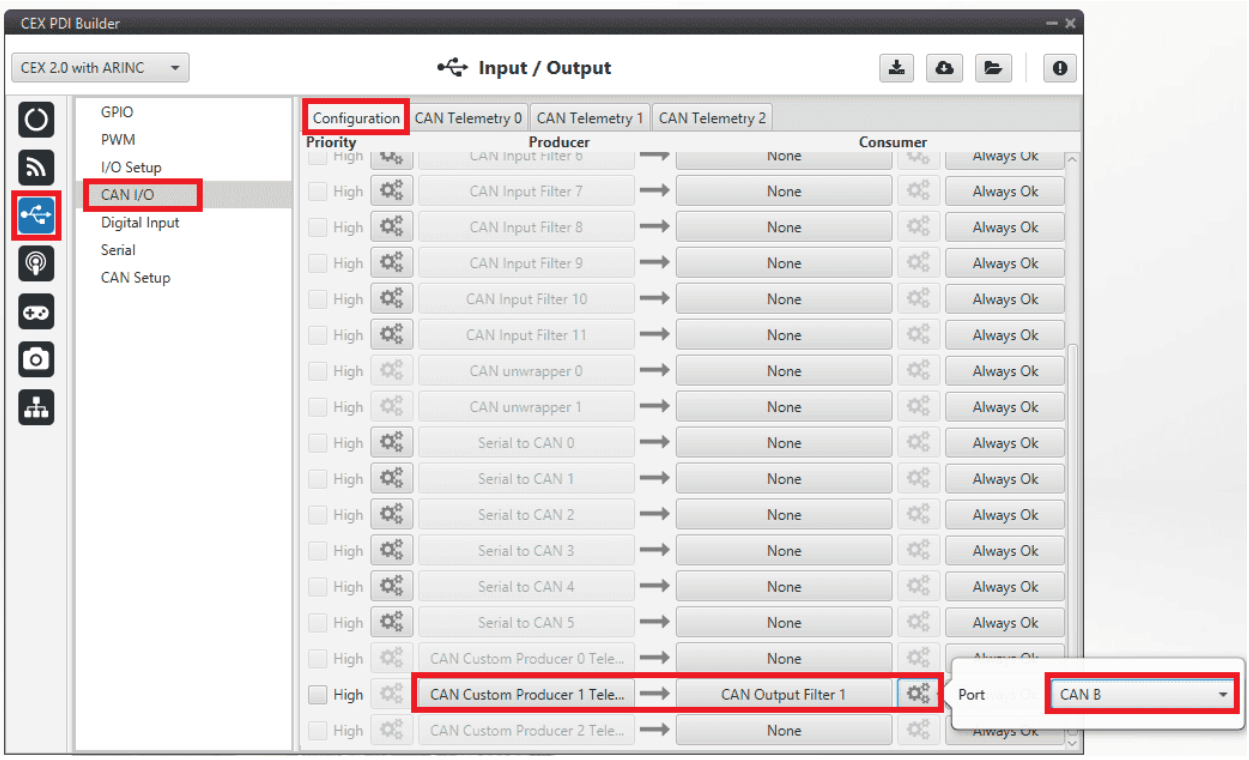


Fig. 37: **CEX PDI Builder - CAN I/O configuration**

5. Go to the **CAN Telemetry tab** that corresponds to the selected CAN Custom Producer Telemetry. In this case CAN Telemetry **1** as the Producer is CAN Custom Producer **1** Telemetry.

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

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

   In the telemetry message one of CEX's variables needs to be selected. As **RPM 1** has been chosen as consumer in the **Digital Input**, the variable to be sent is **RPM1**.

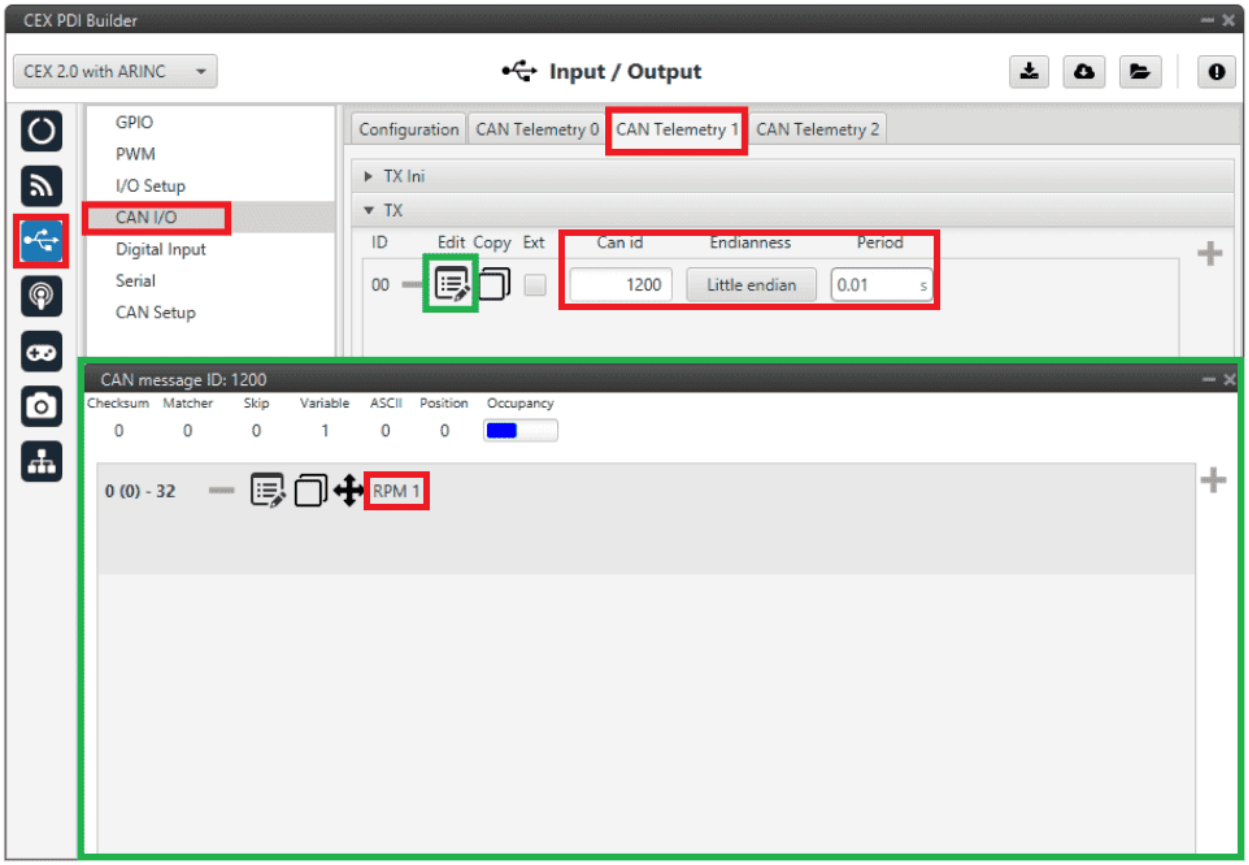


Fig. 38: **Reading RPMs - CAN Telemetry configuration**

More information on the configuration of Telemetry messages can be found in the *CAN Telemetry - Input/Output* section of the present manual.

3.5.2 1x PDI Builder side

6. Go to UI menu → Variables panel → **Real Vars tab**.

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

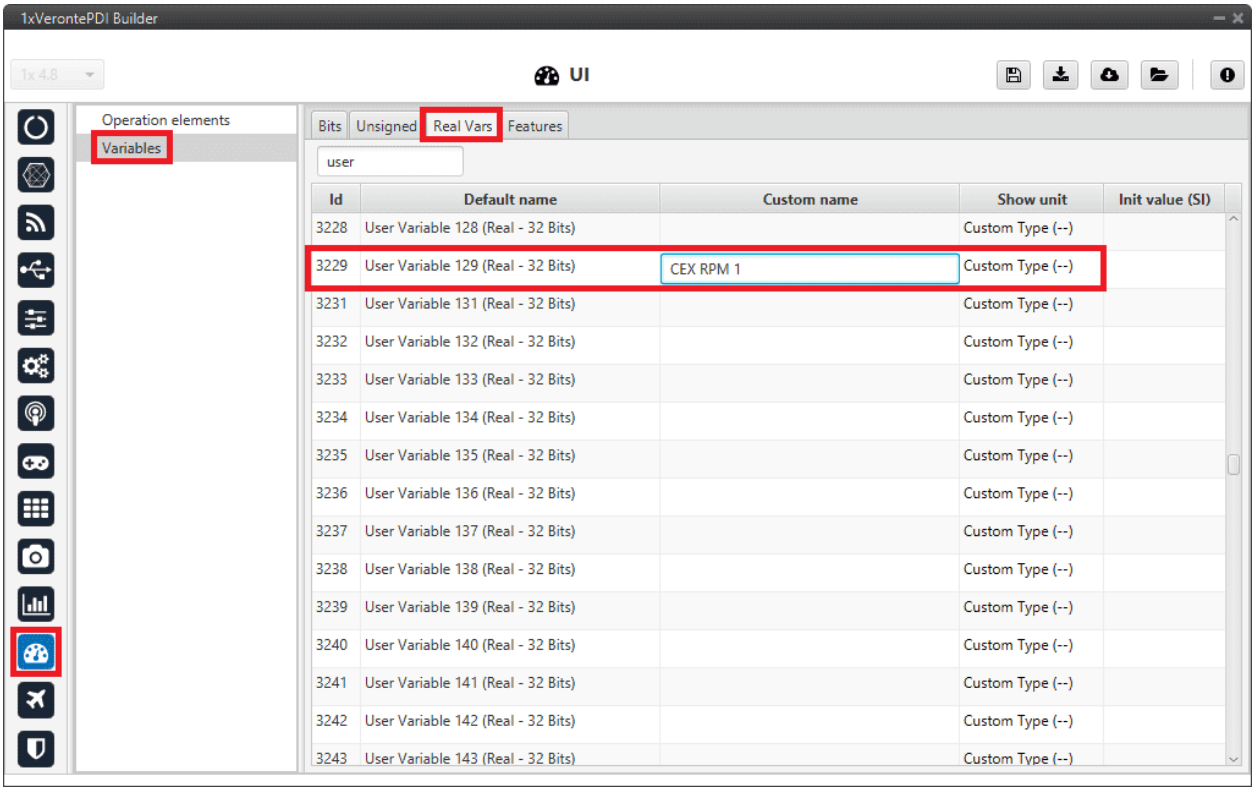


Fig. 39: **1x PDI Builder - User Variable renamed**

7. Go to Input/Output menu → CAN Setup panel → **Mailboxes tab**.

   Configure, in **CAN B**, some mailboxes to receive the message with **ID 1200**:

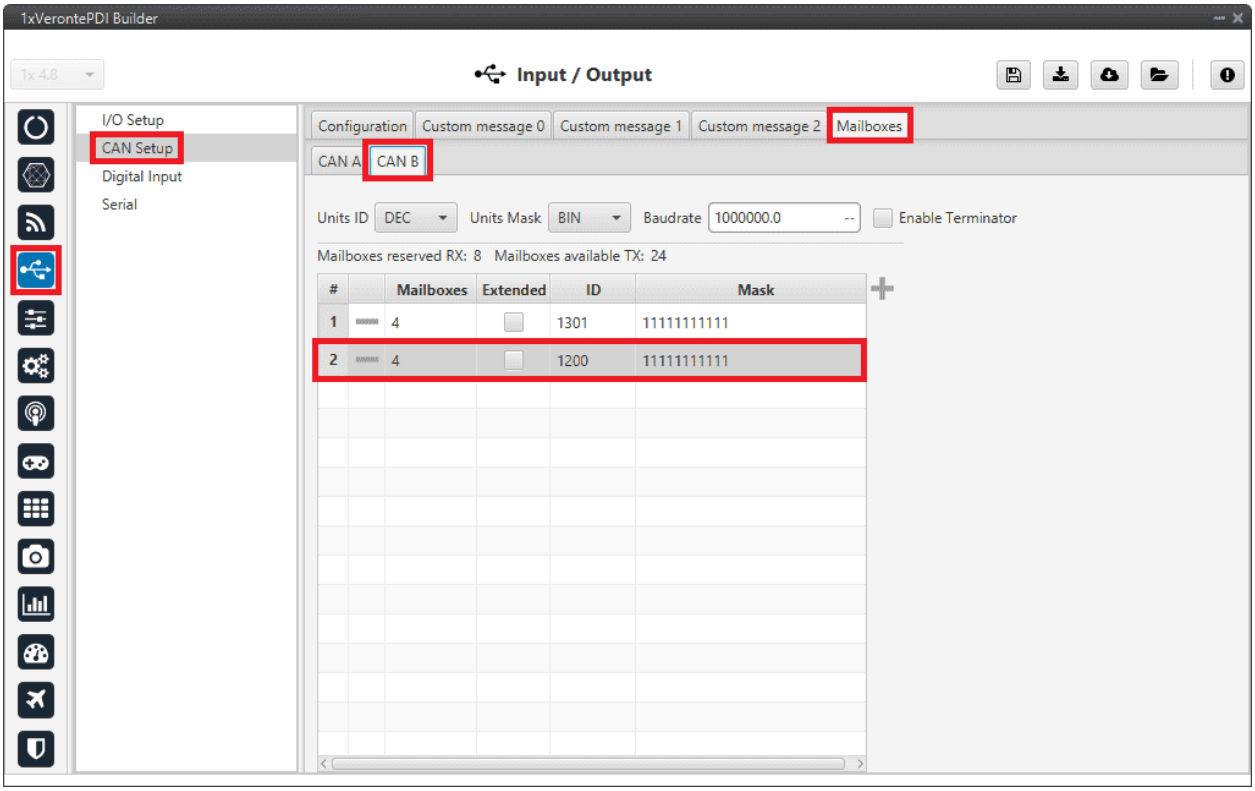


Fig. 40: **1x PDI Builder - Mailbox configuration**

8. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

   Connect an **Input filter** to a **Custom message consumer**. Both CAN buses of the Autopilot 1x can be used, as well as normal IDs and extended IDs:

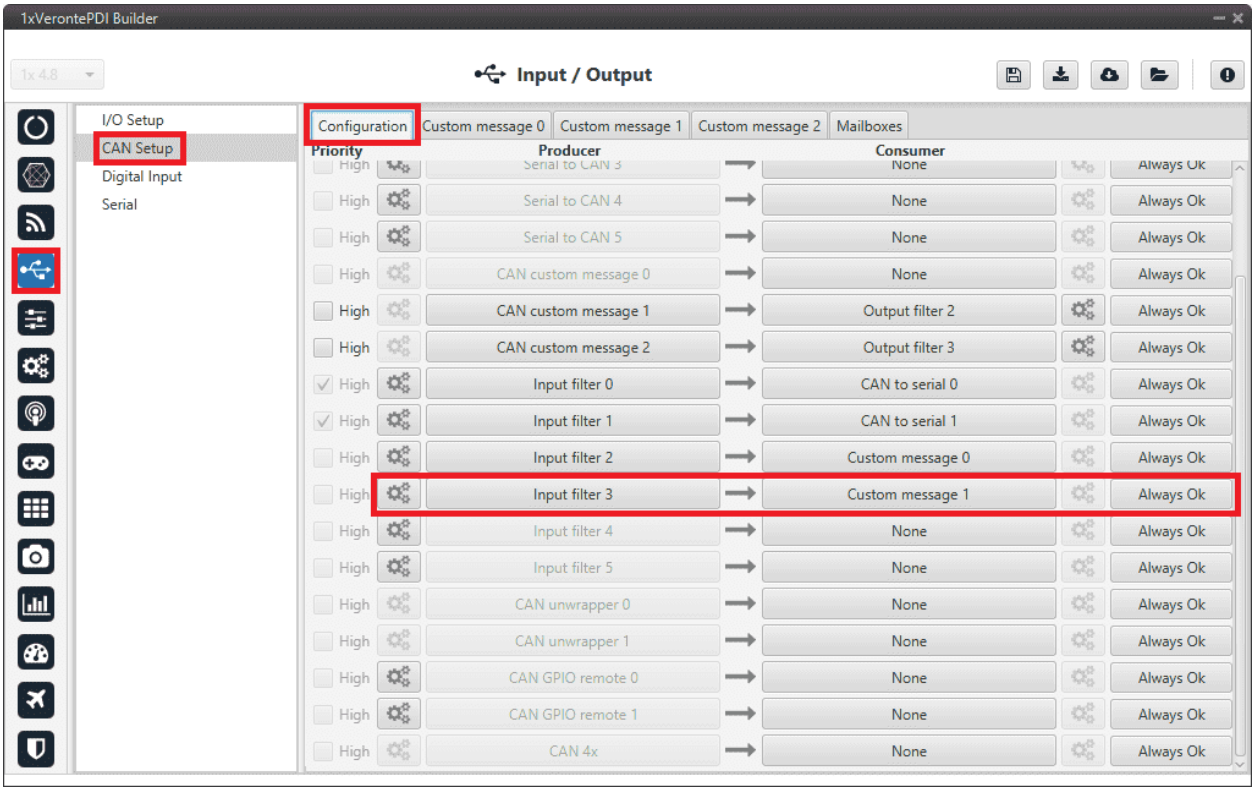


Fig. 41: **1x PDI Builder - CAN Setup configuration**

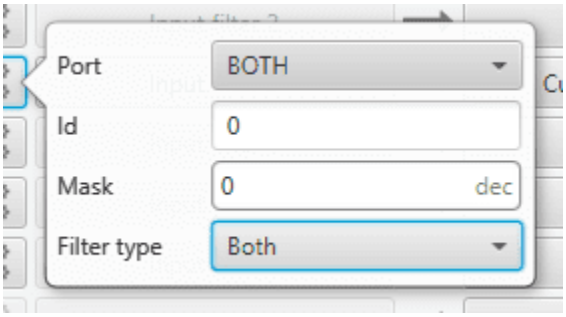


Fig. 42: **1x PDI Builder - Input filter configuration**

9. Go to Input/Output menu → CAN Setup panel → **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:

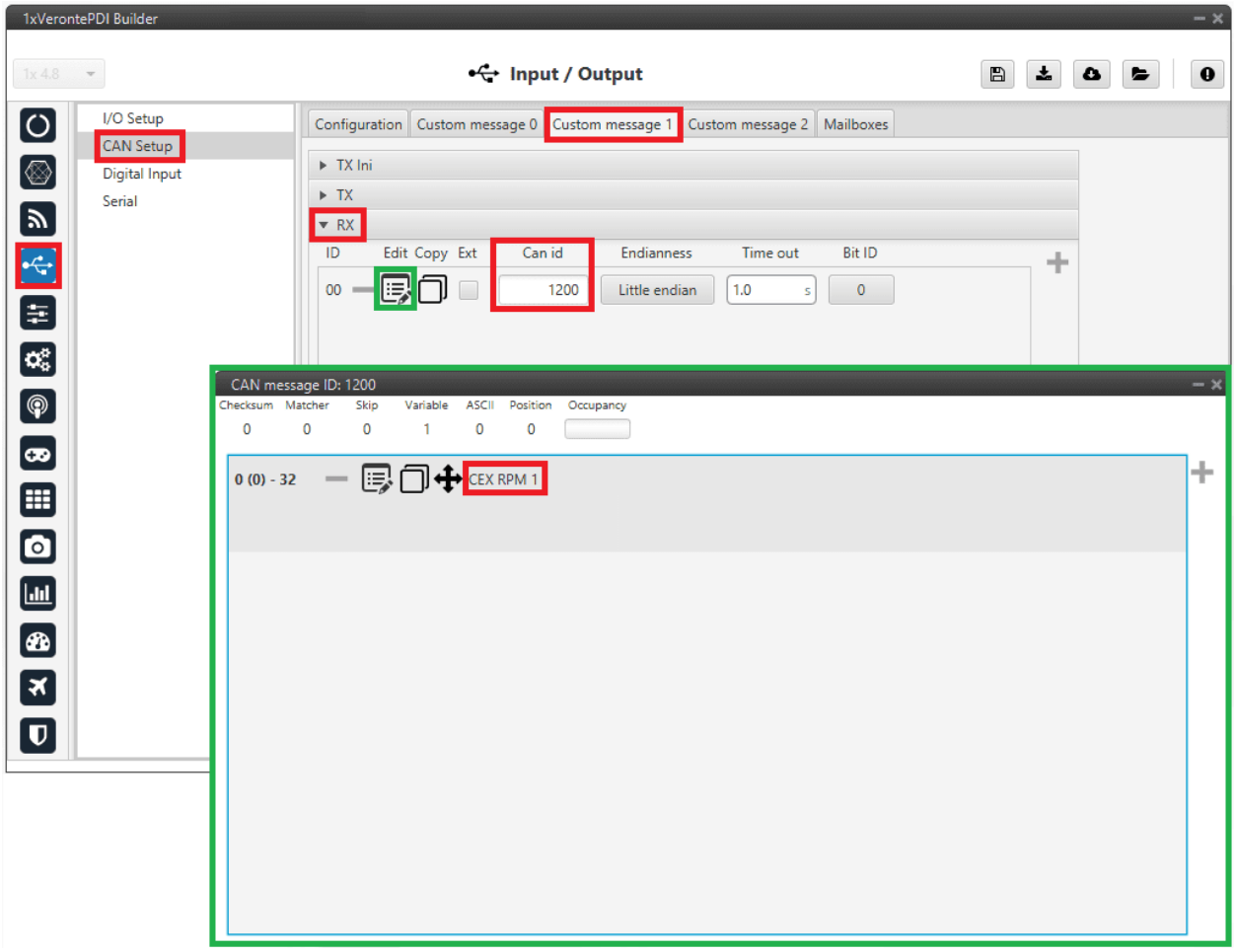


Fig. 43: **1x PDI Builder - Custom Message configuration**

3.6 Serial communication

The user can send information through the serial ports of the **CEX**. For this purpose, please note the following explanation:

3.6.1 CEX 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) via RS232-A (consumer).

   ('Serial to CAN' and 'CAN to Serial' are explained in the *CAN I/O - Input/Output* section of this manual.)

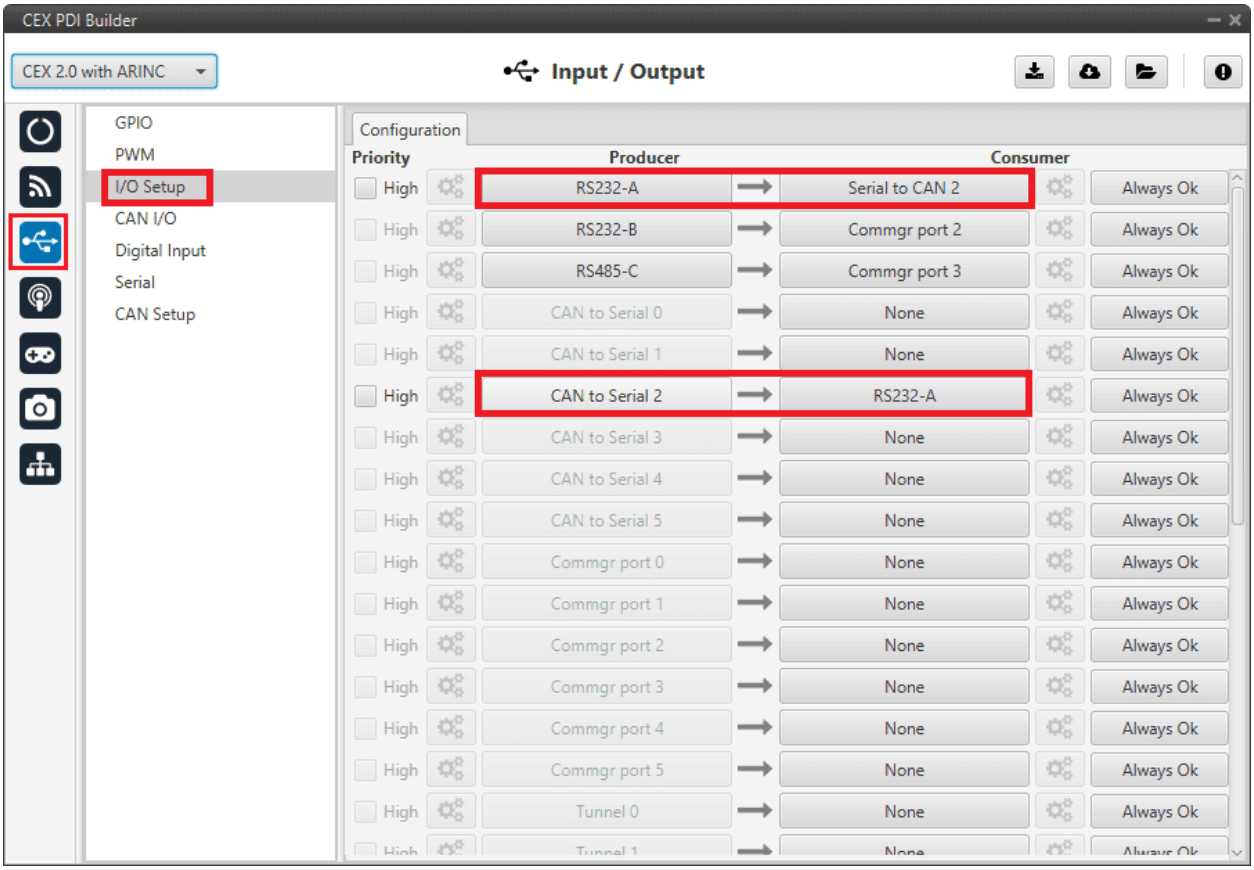


Fig. 44: **CEX PDI Builder - I/O Setup configuration**

2. **CAN I/O configuration**

   - The information that will be **sent via serial port RS232-A** is going to be received on the **CEX** through its **CAN B** port. A **CAN Id of 51** is added to the **CAN 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 a **CAN Output Filter**. The information of '**Serial to CAN 2**' is going to be sent via **CAN B** with a **CAN ID of 50** (to be read by **Veronte Autopilot 1x**).

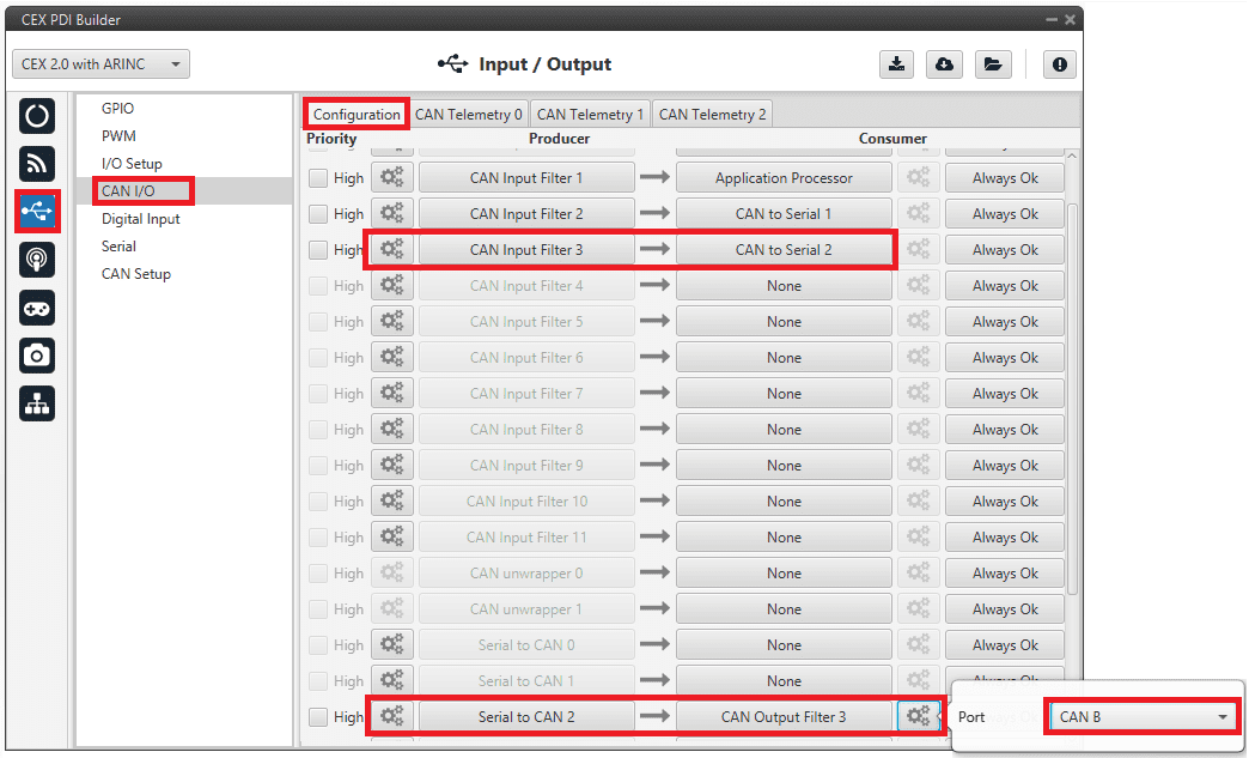


Fig. 45: **CEX PDI Builder - CAN I/O configuration**

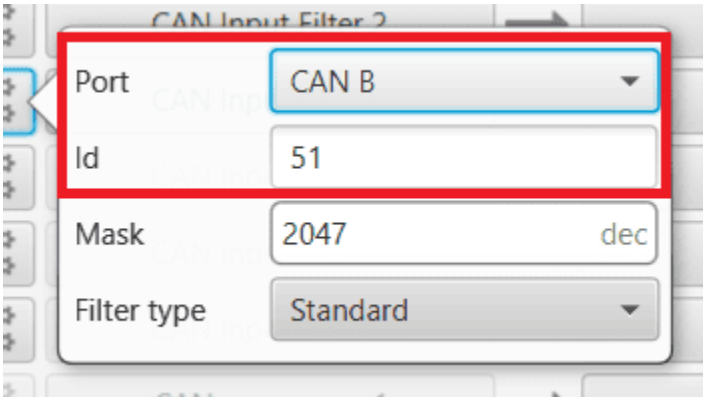


Fig. 46: **CEX PDI Builder - CAN Input Filter configuration**

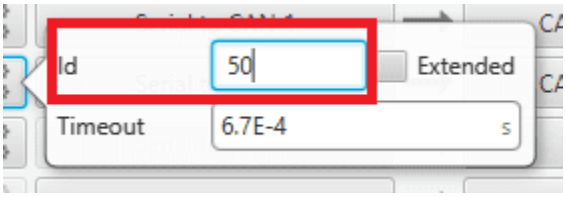


Fig. 47: **CEX PDI Builder - Serial to CAN configuration**

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.

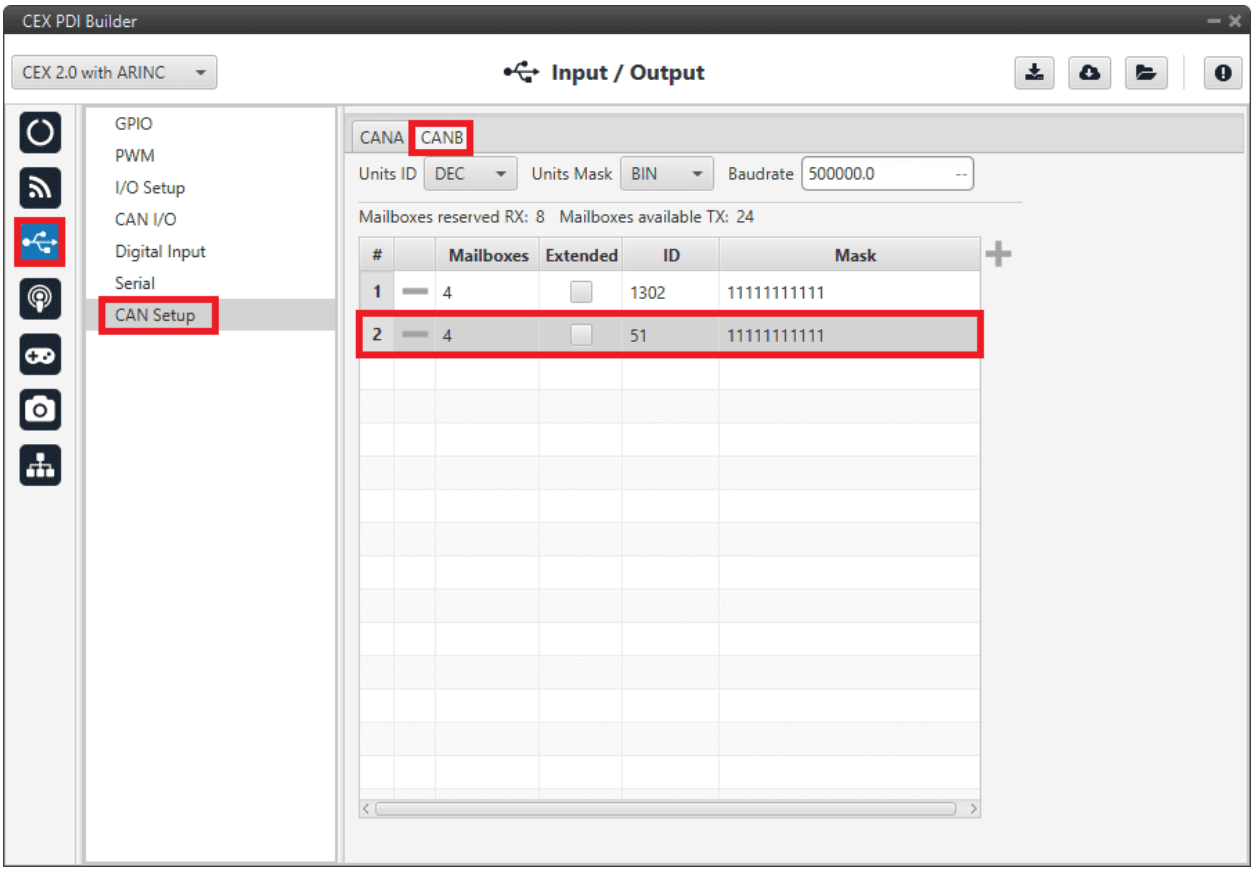


Fig. 48: **CEX PDI Builder - CAN Setup configuration**

3.6.2 1x PDI Builder side

4. **I/O Setup configuration**

   On the **I/O Setup panel**, link an '**RS custom message**' to a '**Serial to CAN**' with the serial data that the **Autopilot 1x** is going to send to **CEX**.

   Then, link a '**CAN to Serial**' to another '**RS custom message**' with the expected serial messages that the **CEX** will receive in the selected serial port.

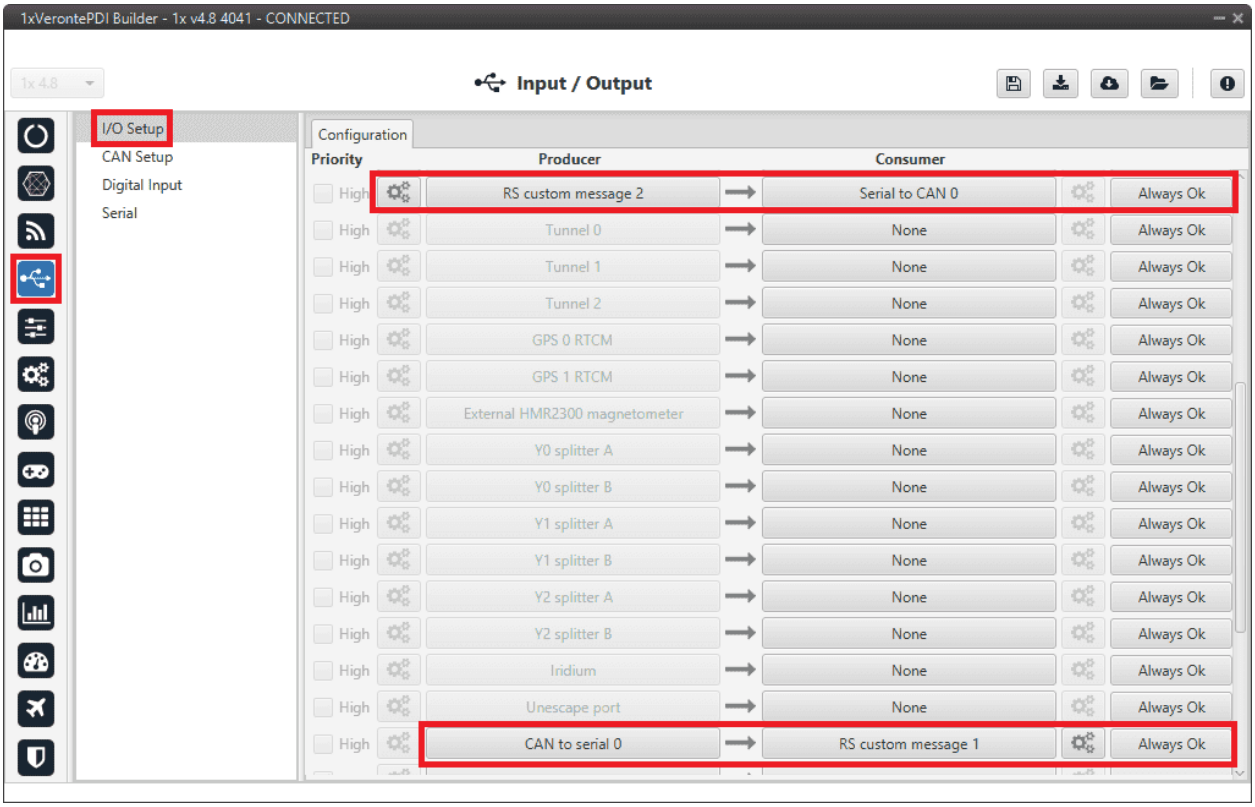


Fig. 49: **1x PDI Builder - I/O Setup configuration**

5. **CAN Setup configuration**

   As for the **CAN I/O**, the same IDs employed in **CEX** for the CAN Input and CAN Output Filters are going to be employed on **Veronte Autopilot 1x** side, but they need to 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**.

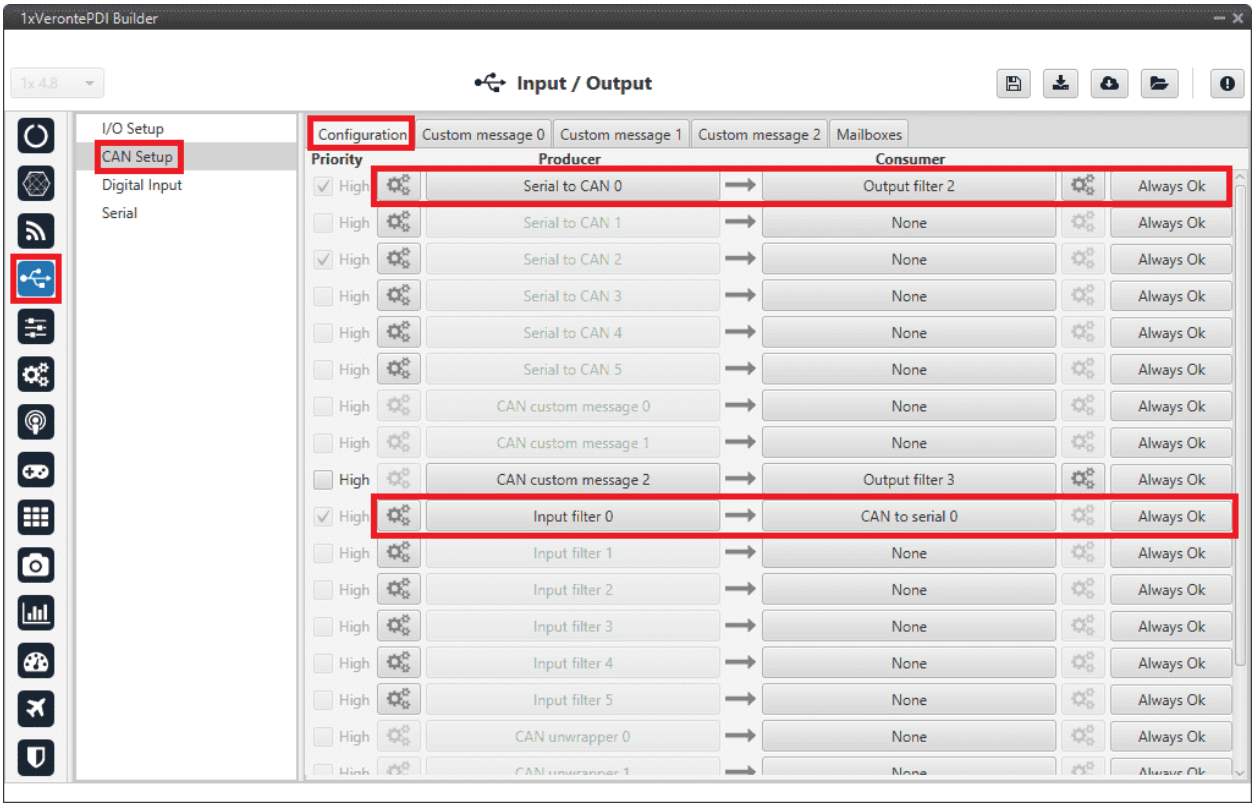


Fig. 50: **1x PDI Builder - CAN Setup configuration**

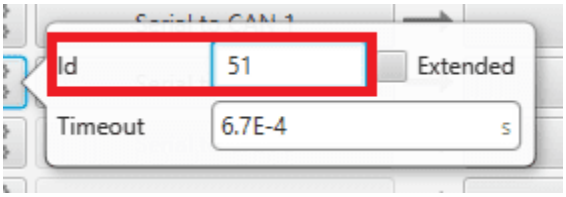


Fig. 51: **1x PDI Builder - Serial to CAN configuration**

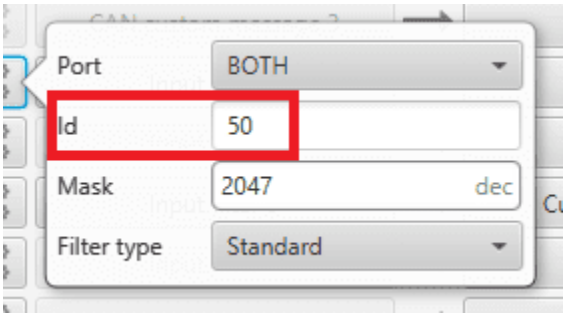


Fig. 52: **1x PDI Builder - Input filter configuration**

6. **Mailboxes configuration**

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

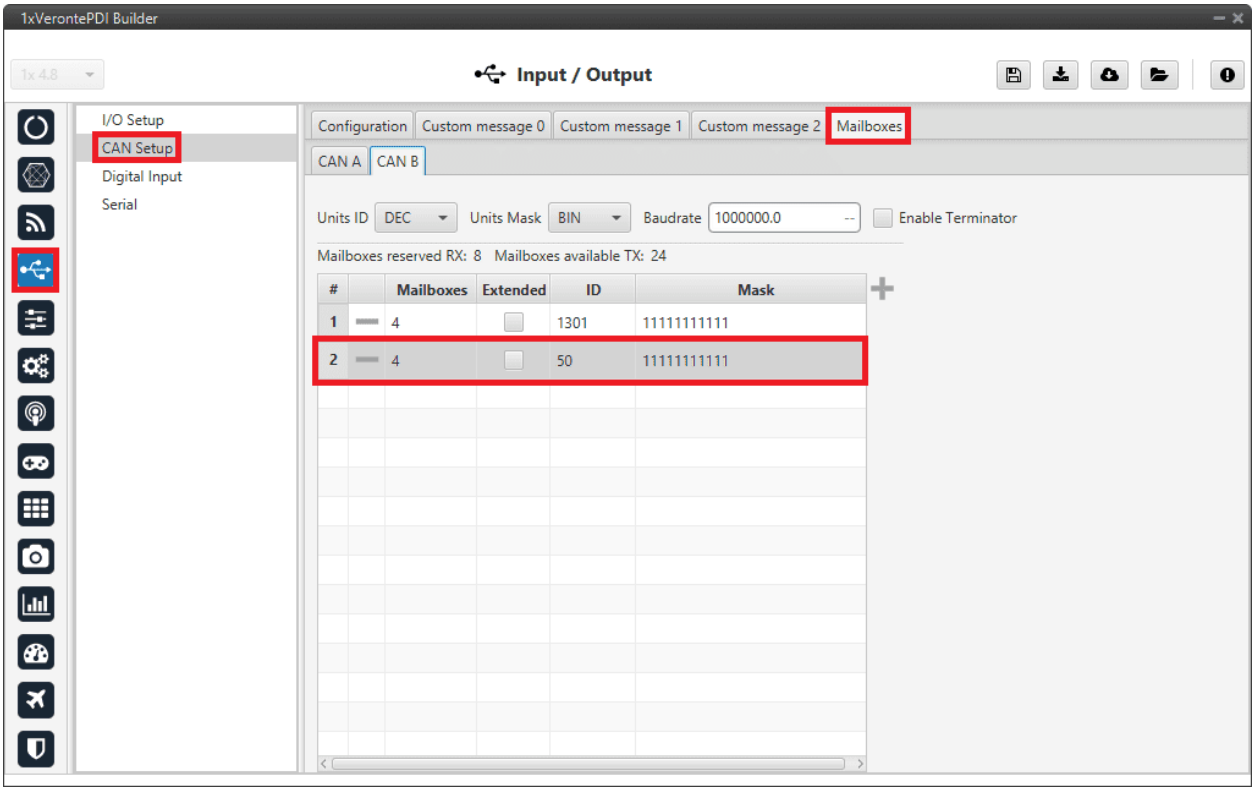


Fig. 53: **1x PDI Builder - Mailboxes configuration**

3.7 External devices

The step-by-step instructions for the following external devices are explained in detail in the following sections:

- *Analog devices*
- *Jetibox*
- *Scorpion tribunus*
- *Veronte products*

3.7.1 Analog devices

CEX is configured to read voltages from *ANALOG 0-7* inputs (**pins 43-50**), automatically storing the received values in **Real variables 1320-1327**.

Note: The correspondence of analog pins and these variables can be consulted in the Real Variables list of **CEX Software Manual**.

For reading information from an analog device, make the following connections:

1. Connect the analog output of the device to one of the designated pins of **CEX** (pins 43-50).
2. Connect the GND of the device to the **ground for analog signals** of **CEX** (pin 42).

Note: For further details on **the pins of CEX**, consult the Pinout - Hardware Installation section of the **CEX Hardware Manual**.

3.7.2 Jetibox

To enable CEX to simulate a Jetibox to read telemetry from legacy Jeti devices, the following steps need to be taken:

1. Go to Devices menu → **Jetibox panel**.

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

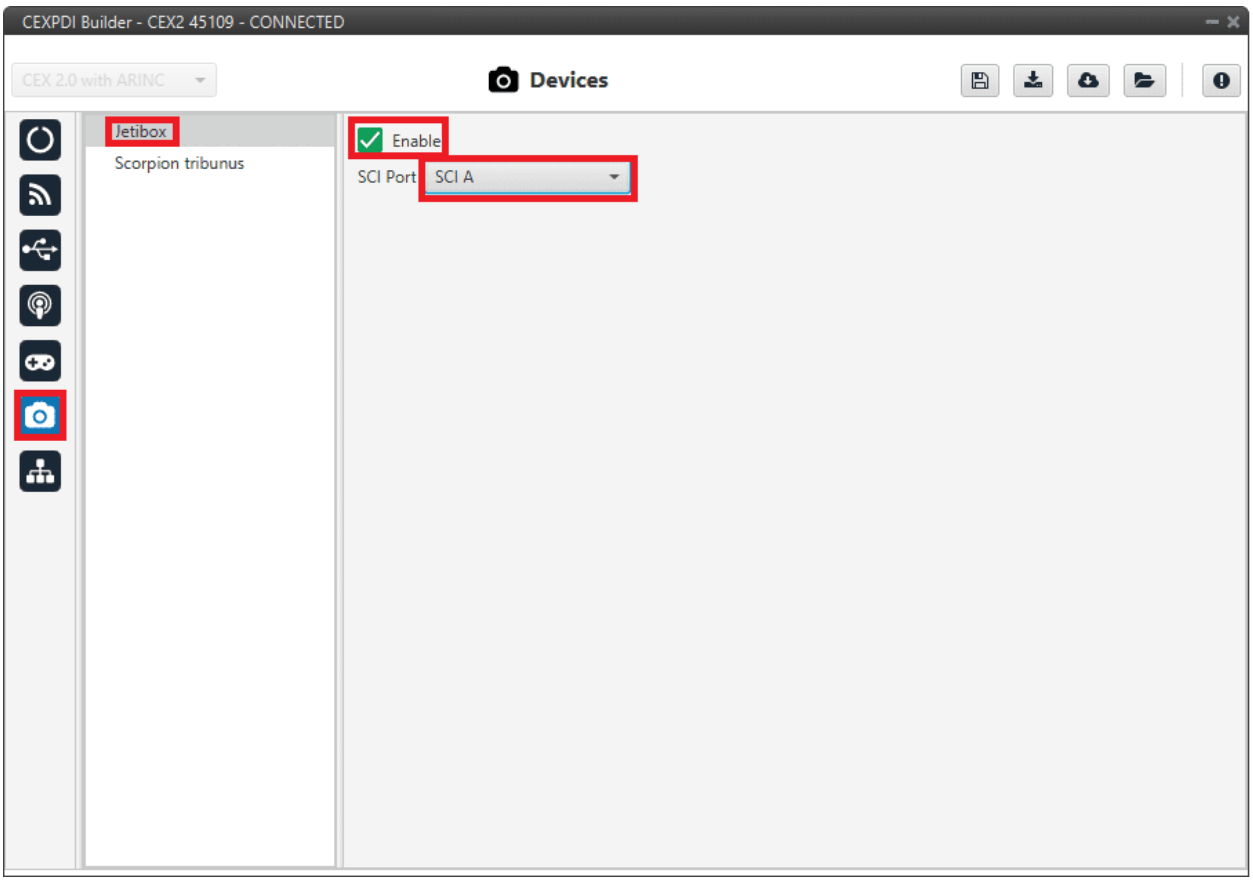


Fig. 54: **Jetibox - Devices configuration**

2. Go to Input/Output menu → **Serial panel**.

   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

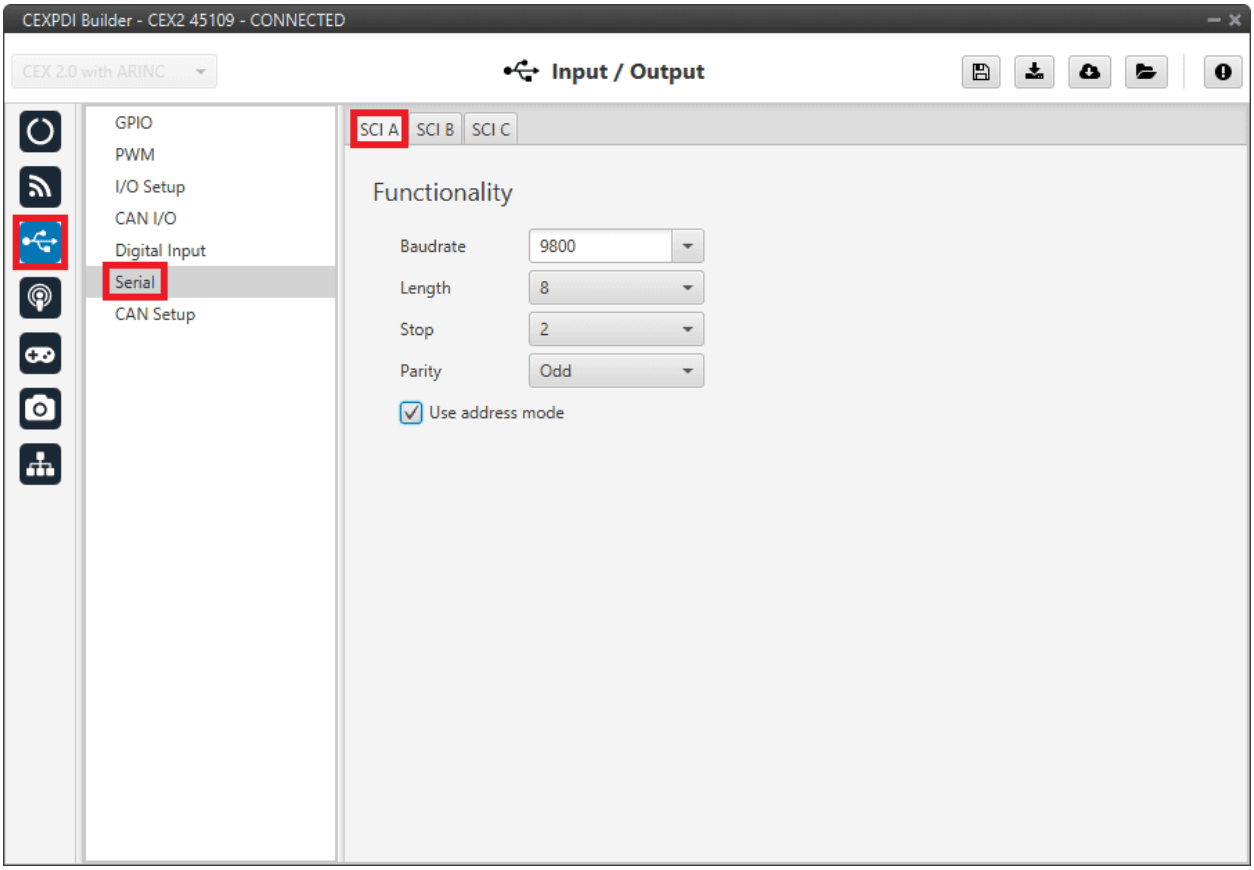


Fig. 55: **Jetibox - Serial configuration**

Note: The serial port will be completely reserved for this, so it cannot be used for any other purpose and the I/O Setup configuration affecting it will be ignored.

---

3. Go to Input/Output menu → **I/O Setup panel**.

   Link the specific **JETI box** consumer to the **serial port** configured in **step 2**:

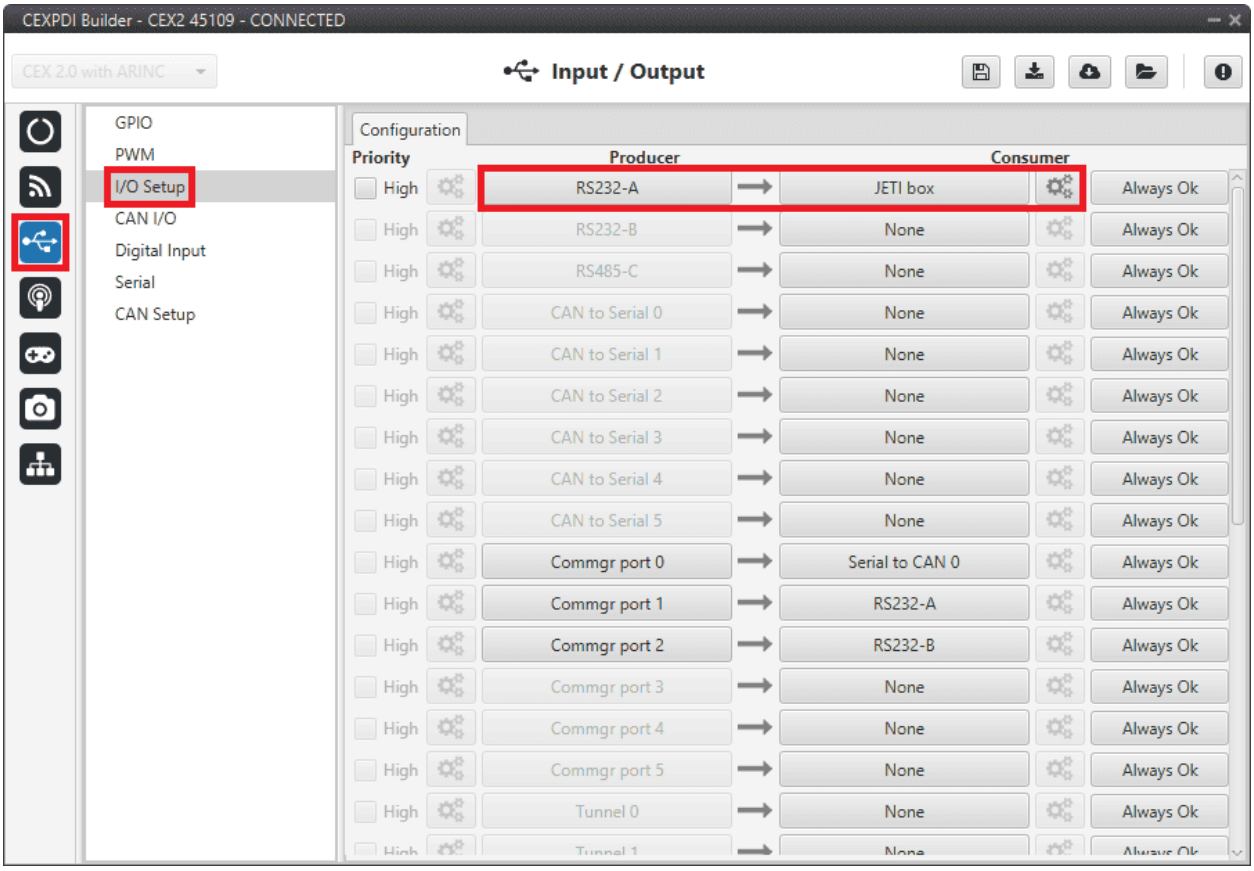


Fig. 56: **Jetibox - I/O Setup configuration**

4. Configure the Jetibox IO consumer to retrieve the data. Click on the **configuration button**  :

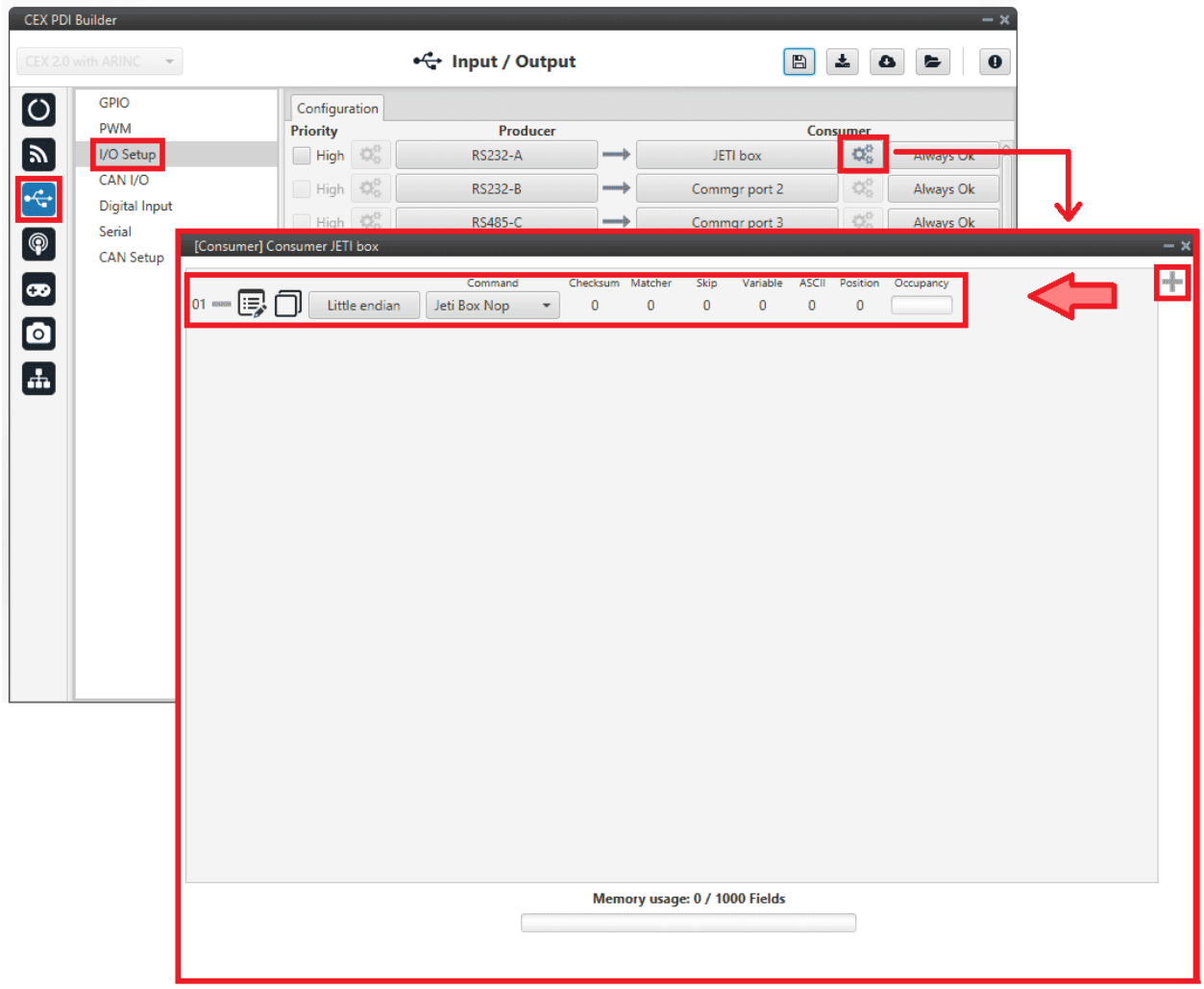


Fig. 57: **Jetibox - JETI box Consumer configuration**

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):

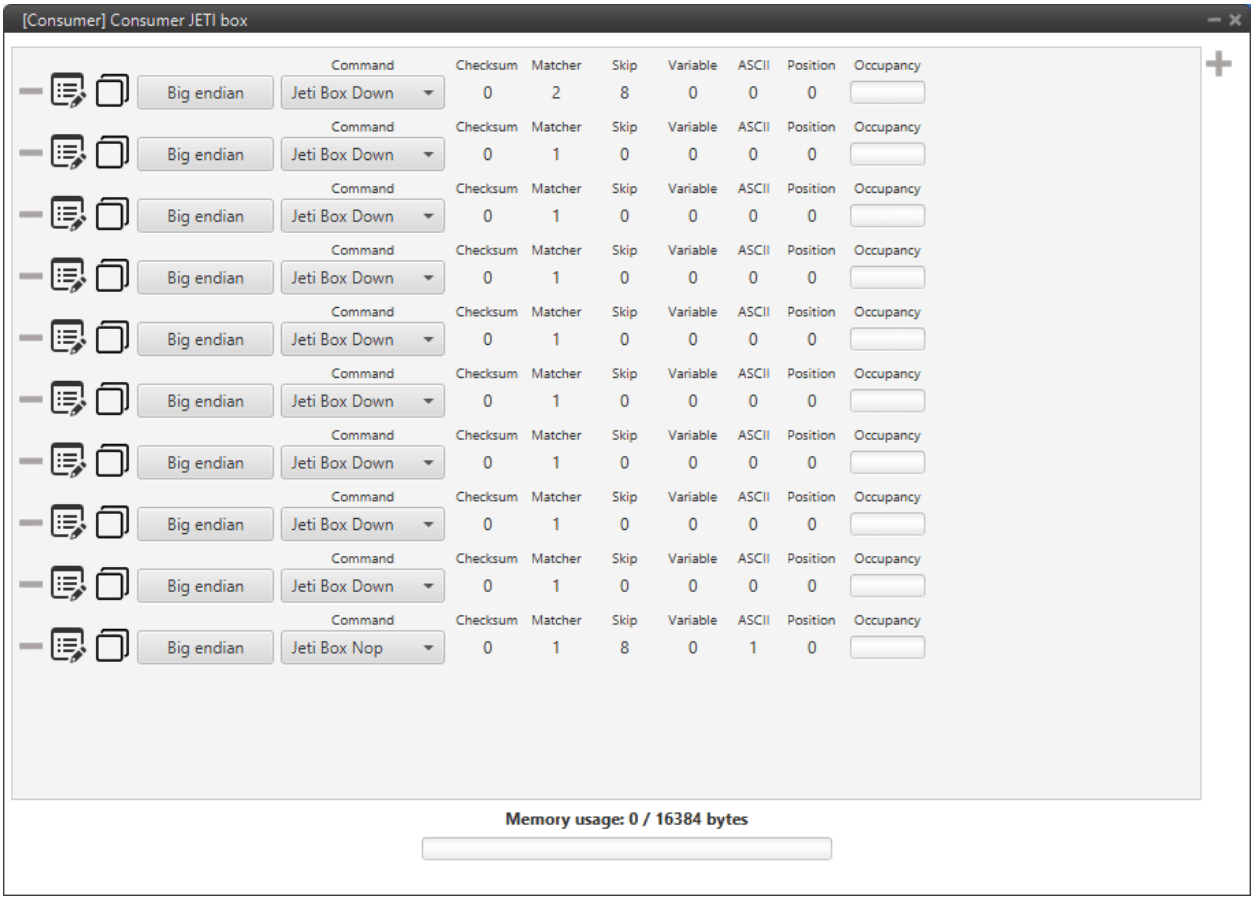


Fig. 58: **JETI box example**

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

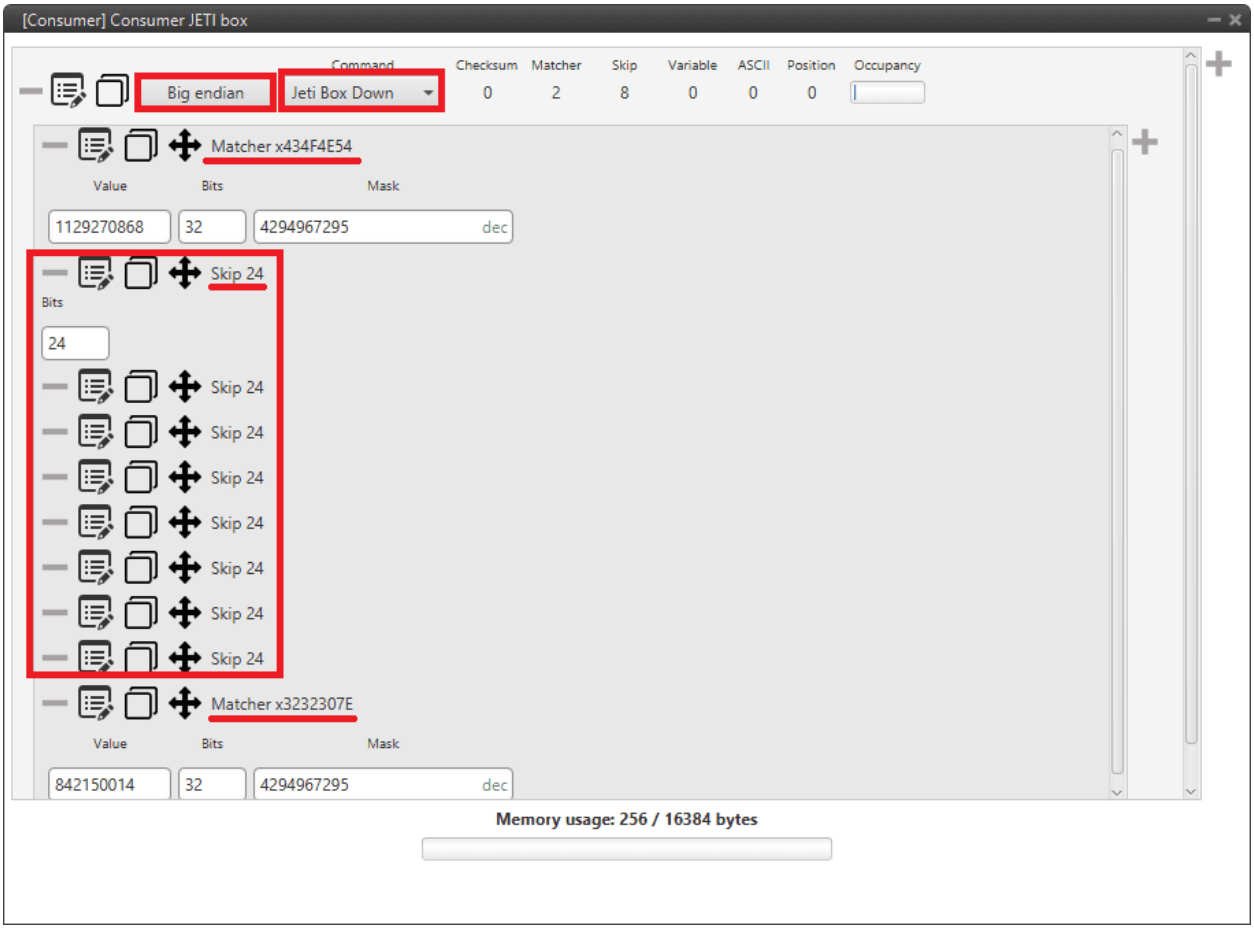


Fig. 59: **JETI box custom message example**

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 Temperatu”…
   - 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), separartor(‘,’)

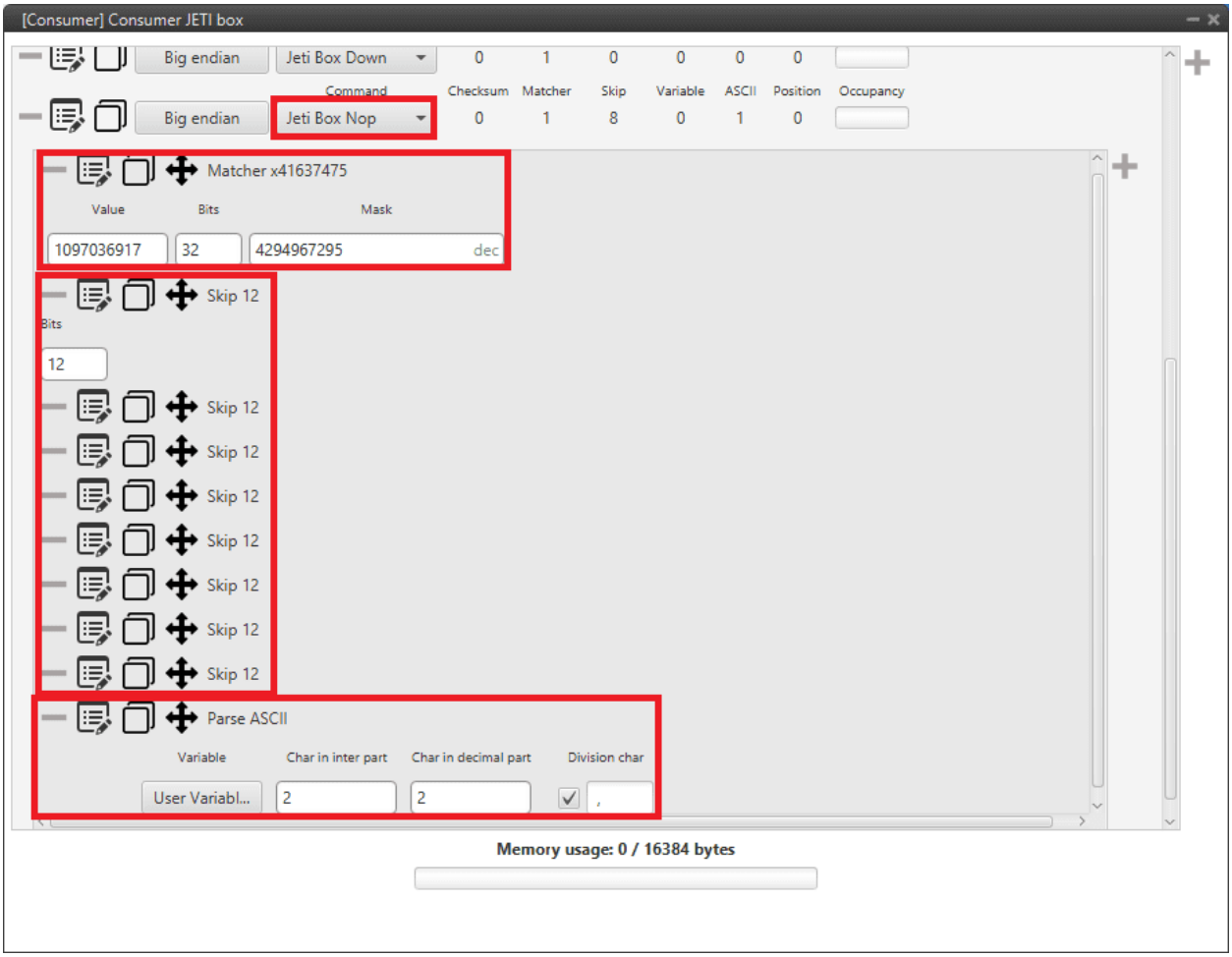


Fig. 60: **JETI box last custom message example**

3.7.3 Scorpion tribunus

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

1. Go to Devices menu → **Scorpion tribunus panel**.

   Enable it and configure it as shown:

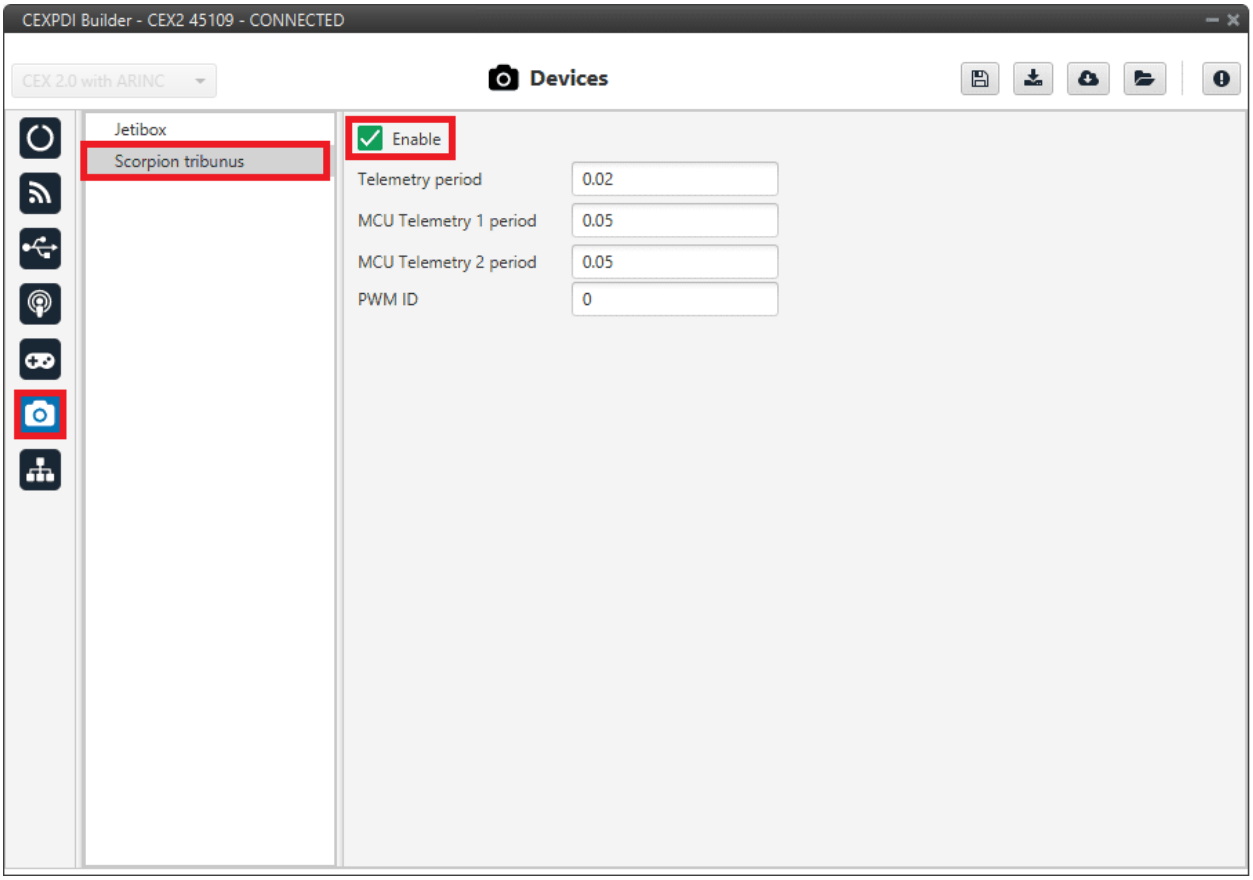


Fig. 61: **Scorpion tribunus - Devices configuration**

2. Go to Input/Output menu → **I/O Setup panel**.

   Link the specific Tribunus ESC consumer to the desired port:

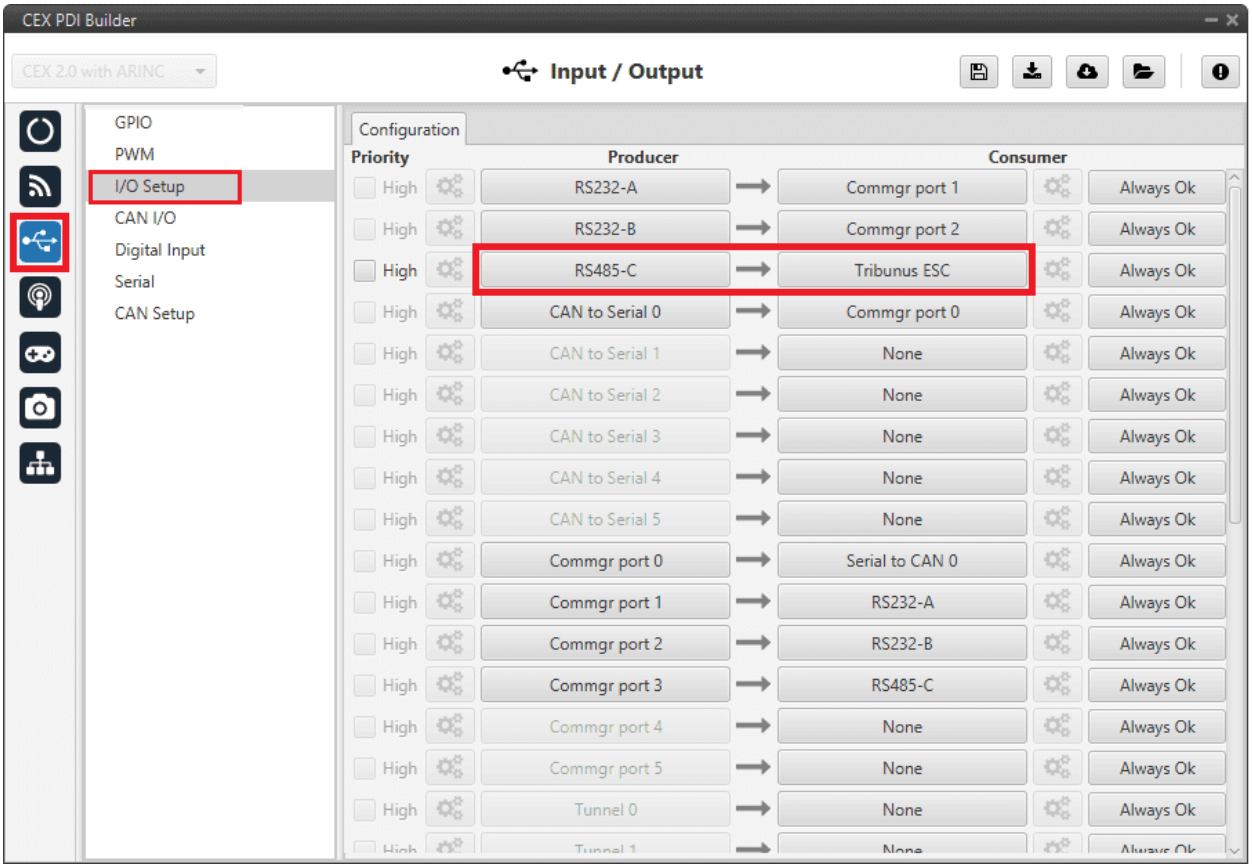


Fig. 62: **Scorpion tribunus - I/O Setup configuration**

3.7.4 Veronte products

3.7.4.1 Connection with Autopilot 1x via CAN

No configuration is necessary in CEX, the configuration for communication with **Autoilot 1x** is set by default.

The configuration required in **1x PDI Builder** to communicate with **CEX** via CAN is explained in the CEX/MEX - Integration examples section of the **1x PDI Builder** user manual.

3.7.4.2 Connection with Autopilot 4x via CAN

3.7.4.2.1 CAN Reception IDs

First, users can setup the receiving CAN Ids for each one of the 3 possible **Veronte Autopilots 1x** sending data to CEX in the **Arbitration menu**.

Go to Arbitration menu → **CAN Setup panel**:

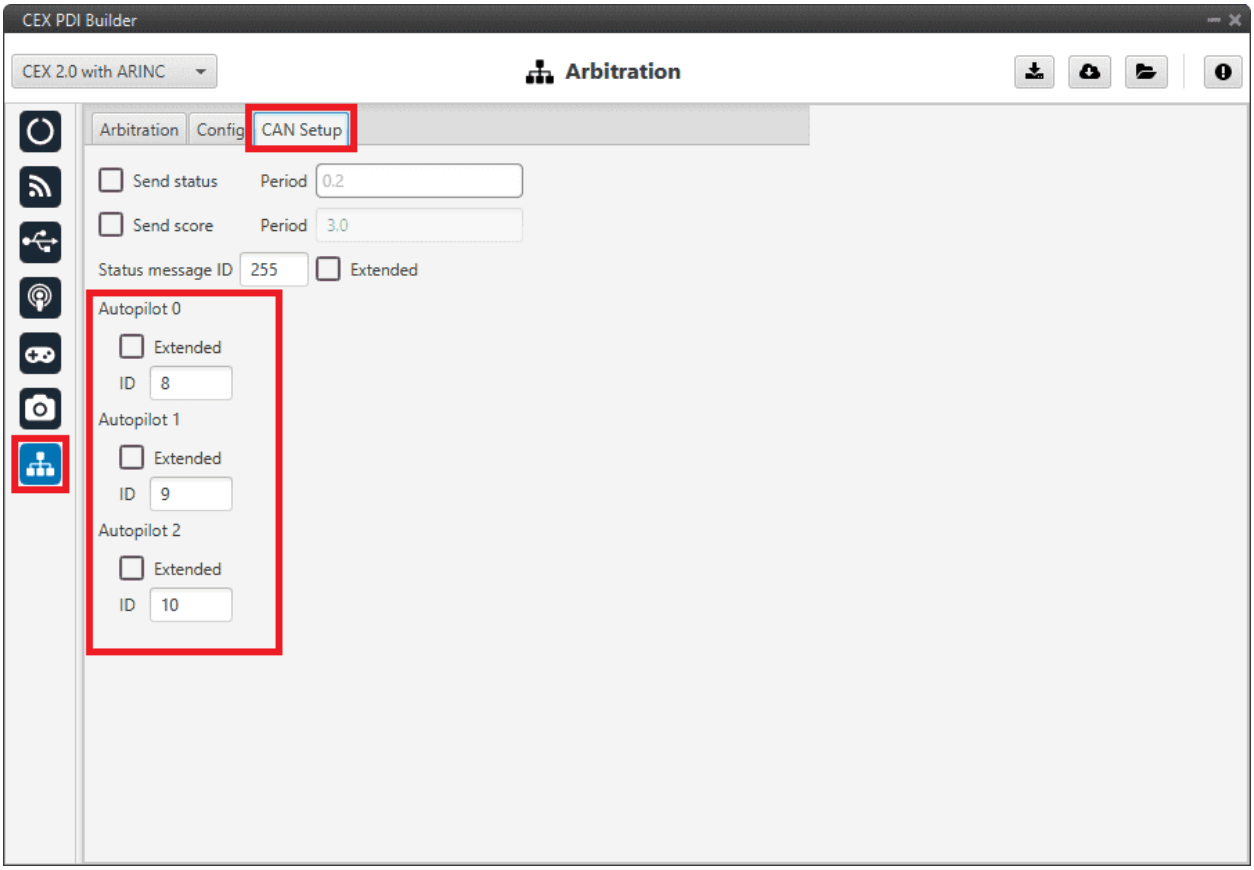


Fig. 63: **Arbitration - CAN Reception IDs**

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

3.7.4.2.2 CAN I/O Interconnections

Once the CAN IDs are set, users shall configure:

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

Go to Input/Output menu → CAN I/O panel → **Configuration tab**:

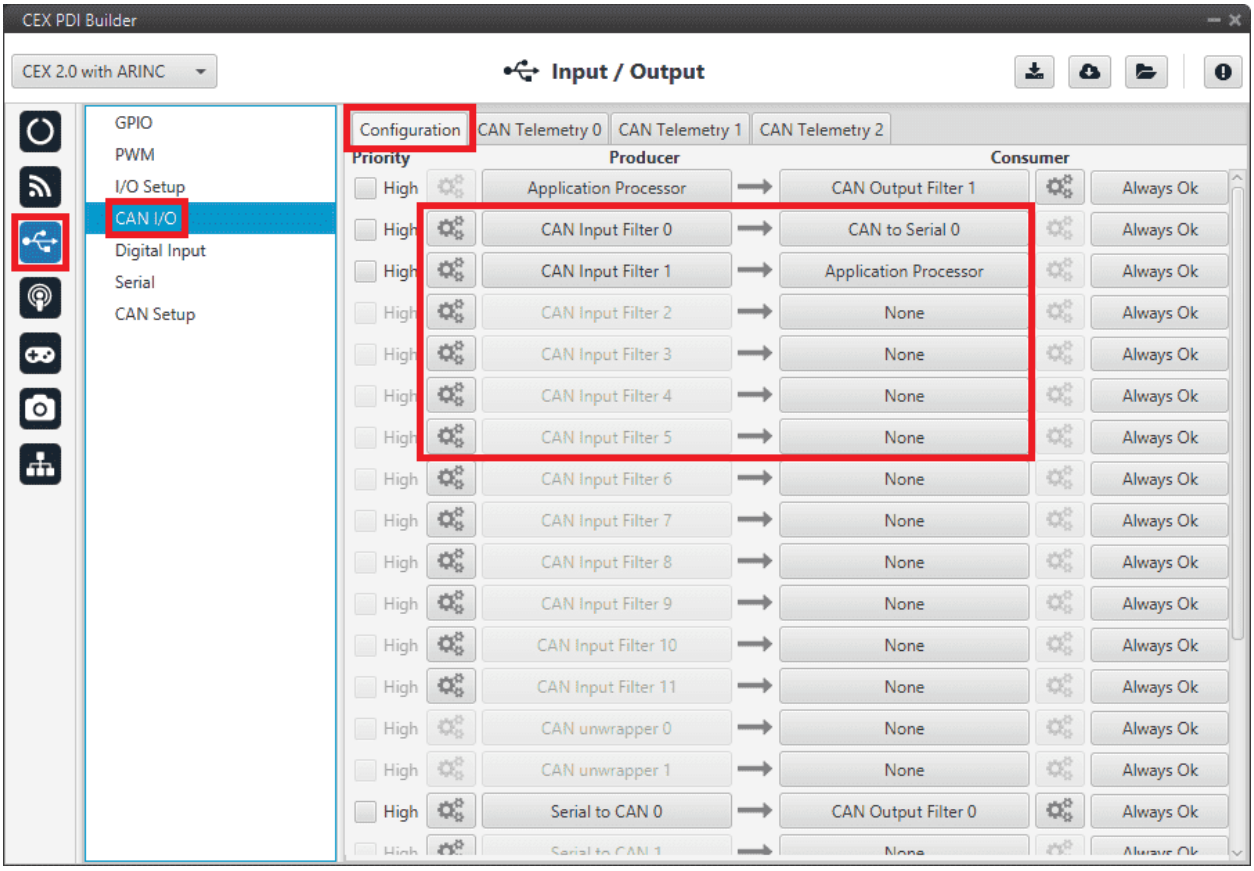


Fig. 64: **Arbitration - CAN Input Filters**

For more information on CAN I/O configuration, see the *CAN I/O - Input/Output* section of this manual.

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

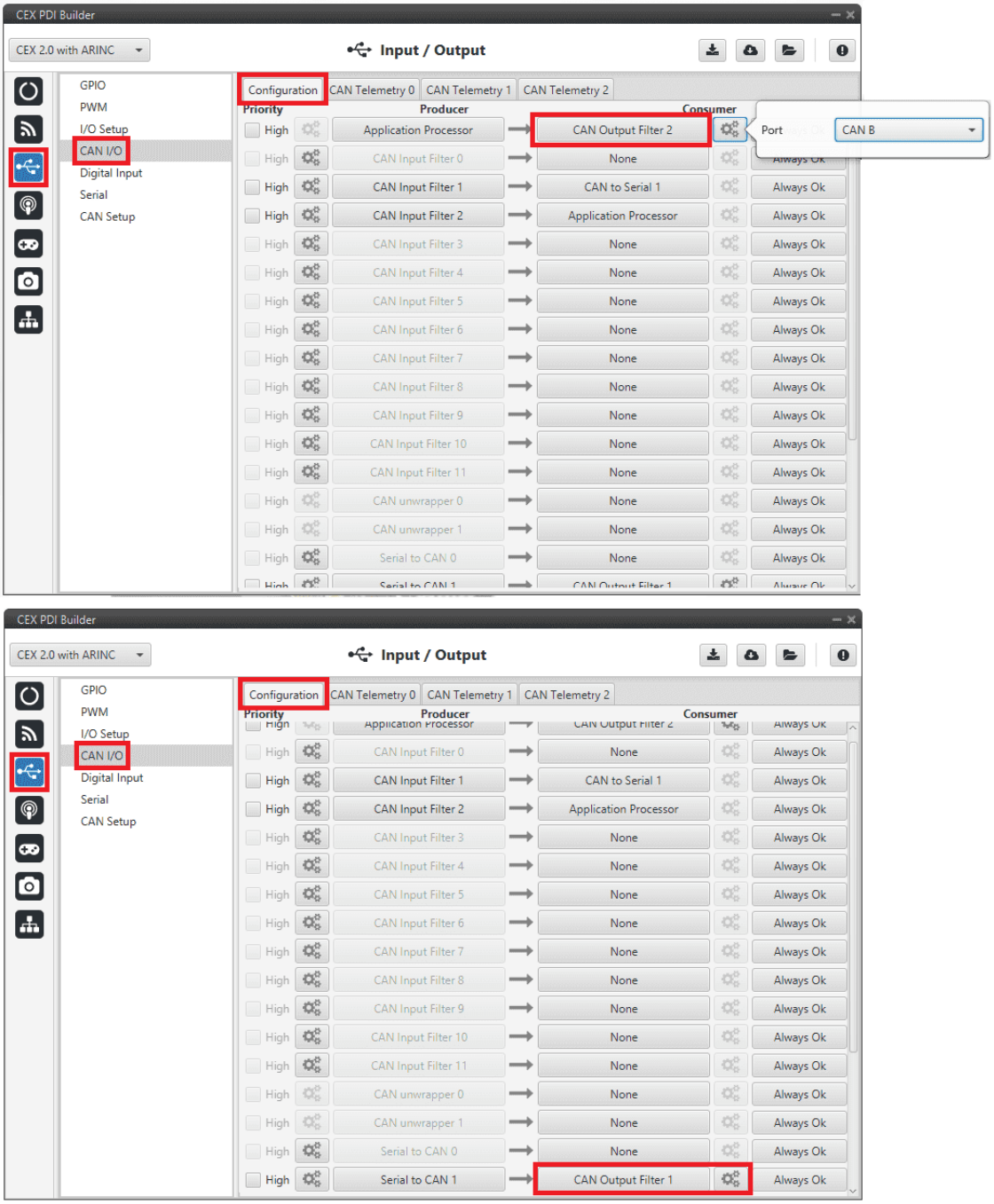


Fig. 65: **Arbitration - CAN Output Filters**

CHAPTER

FOUR

TROUBLESHOOTING

4.1 Maintenance mode (loaded with errors)

The following error message may appear when trying to save a change or import a configuration.

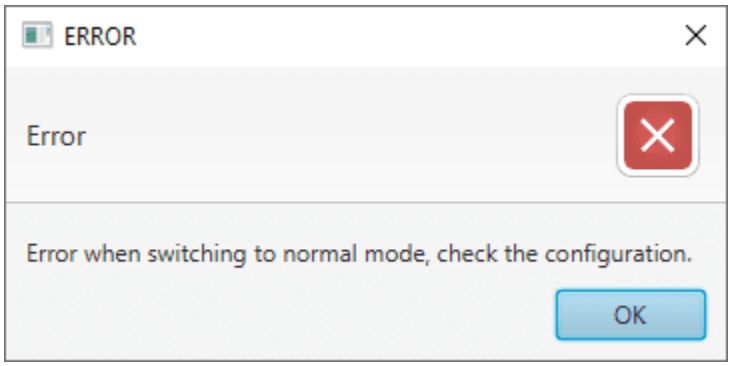


Fig. 1: **Error message**

Therefore, CEX will be in ‘Maintenance mode (loaded with errors)’:

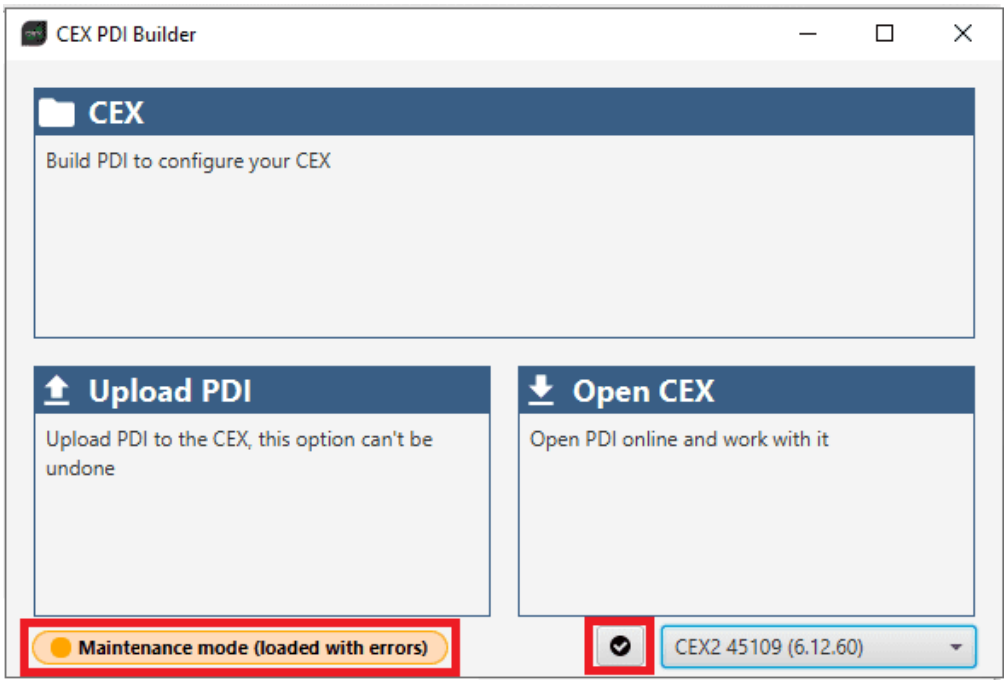


Fig. 2: **Maintenance mode (loaded with errors)**

To check what the source of the problem is, the user can simply click on the **PDI Error button**  , which will show what the PDI Error is:

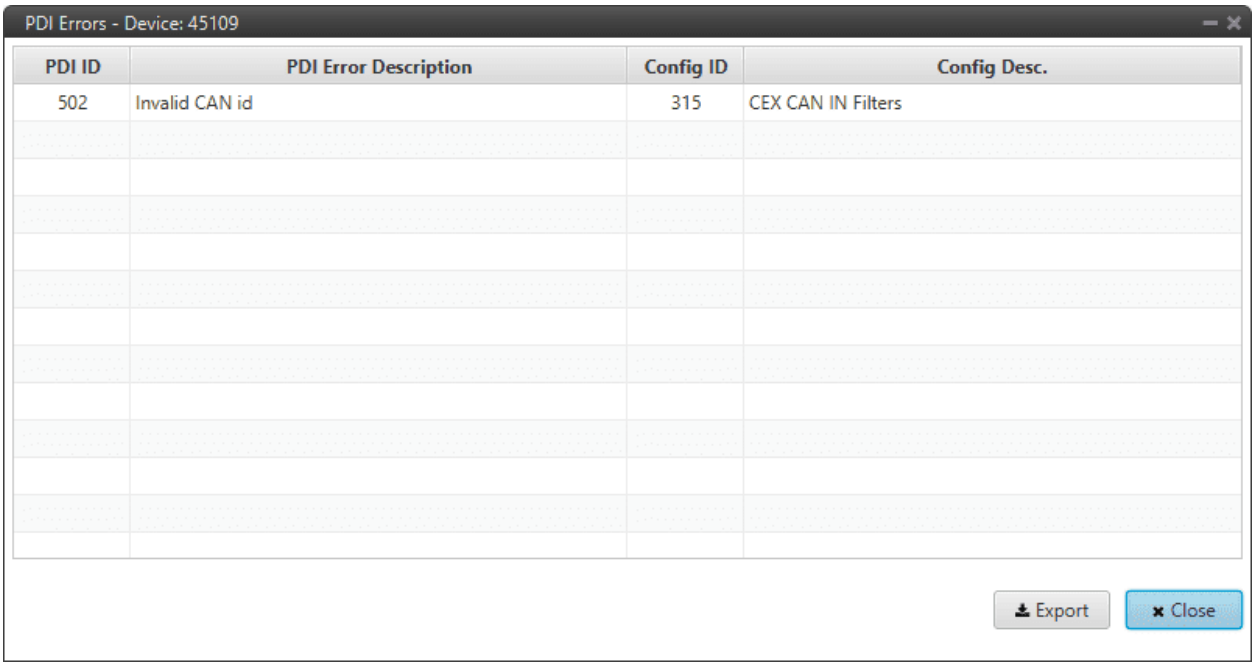


Fig. 3: **Maintenance mode (loaded with errors) - PDI Errors panel**

- **PDI ID**: ID of the PDI Error.
- **PDI Error Description**: Description of this PDI Error. A list of all PDI Errors can also be accessed in the List

of PDI errors section of the **1x Software Manual**.

- **Config ID**: ID of the configurable (.xml file) containing the data in which the PDI Error has been caused.
- **Config Description**: Description of the configurable (.xml file) containing the data in which the PDI Error has been caused.

Clicking the **Export** button will export a `.csv` file with the same information shown in this PDI Errors panel.

Finally, it is possible to access the CEX configuration to fix this error.

4.2 Migrate configuration

Warning: When performing automatic migration from a previous version to the current version of the software, errors may occur.

It is then the responsibility of the user to check the subsequent result.

CHAPTER

FIVE

FAQ

5.1 How to calculate a mask

This section attaches a python program that allows users to easily calculate their mask in standard or extended frame format by simply entering the CAN Ids as a **vector**. In addition, this program also converts each Id entered into binary.

maskCalculator.py

An example of the execution of this program is shown below:

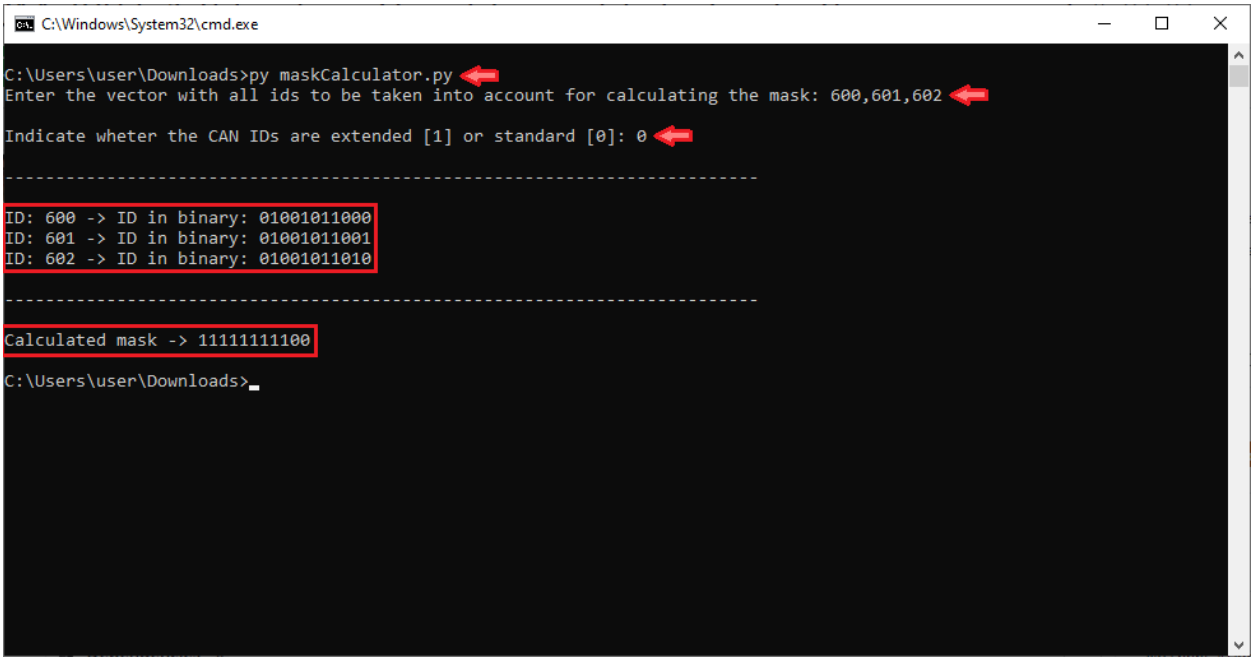


Fig. 1: **Example of maskCalculator program**