CAN out - CAN output mailbox

SD017 | Posted on April 2, 2021 | Updated on June 30, 2025

Stéphane LOVEJOY
Senior Software Developer
imperix • in

Table of Contents

The CAN output mailbox block allows sending CAN messages with up to 8 bytes long payloads. To receive messages with payloads of up to 8 bytes, the CAN input mailbox should be used.

It supports two operating modes:

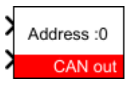
- **On-demand mode**: the user manually triggers the message transmissions.
- **Periodical mode**: the message is sent periodically, whether the data has been changed or not. The user can configure the transmission frequency.

The CAN frame unique identifier also represents the message priority. Therefore, frames with low identifiers will have the highest priority.
Only the CAN base frame format is supported, extended frame format is not. Therefore, the CAN address is limited to 11 bits (0 to 2047).

Simulink block

Signal specification

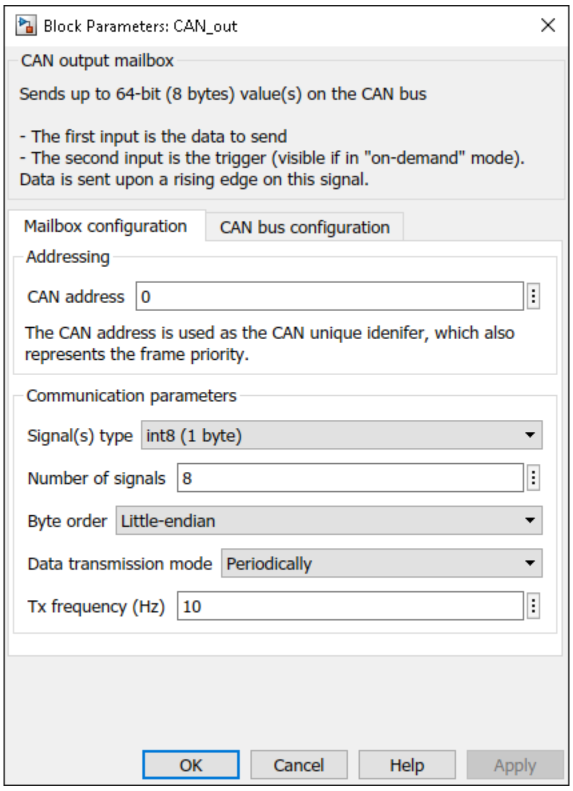
- The data input signal supports a vector of data. The accepted data type is configured by `Signal type` parameter. The vector length can be configured with `Number of signals` parameter.
- The second input is the *send data* signal. It is used to initiate a data transmission when the **on-demand** mode has been selected. Data is sent upon a rising edge on this signal.



Parameters

- `CAN address` : sets the CAN output mailbox address. It also sets the CAN unique identifier of the data frame.
- `Signal Type` : defines the data type accepted in the data input (int8, uint8, int16, uint16, int32, uint32, in64, uint64, float, or double).
- `Number of signals` : sets the vector length of the data output. (from 1 to 8)
- `Byte order` : defines the byte order in which the data will be sent. Either little-endian or big-endian.
- `Tx frequency` : Sets the data transmission frequency if the **periodical mode** has been selected.
- `Baud rate` : configures the baud rate of the CAN bus. Baud rates up to 1 Mbit/s are supported.

The total number of bits sent by CAN output mailbox (resulting from `Signal type` and `Number of signals`) can not exceed 64 bits (8 bytes), which is the maximum payload of a CAN frame.


Block Parameters: CAN_out

CAN output mailbox

Sends up to 64-bit (8 bytes) value(s) on the CAN bus

- The first input is the data to send
- The second input is the trigger (visible if in "on-demand" mode).
Data is sent upon a rising edge on this signal.

Mailbox configuration | CAN bus configuration

Addressing

CAN address: 0

The CAN address is used as the CAN unique idenifer, which also represents the frame priority.

Communication parameters

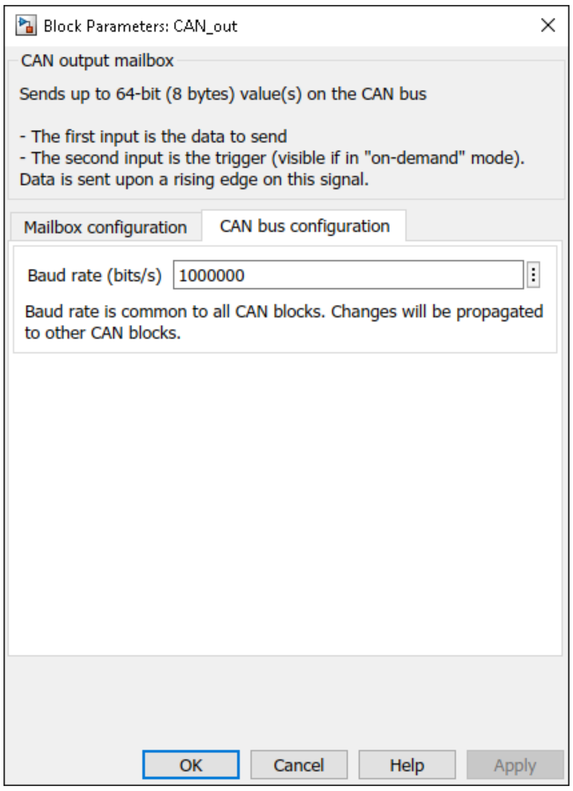
Signal(s) type: int8 (1 byte)

Number of signals: 8

Byte order: Little-endian

Data transmission mode: Periodically

Tx frequency (Hz): 10


Block Parameters: CAN_out

CAN output mailbox

Sends up to 64-bit (8 bytes) value(s) on the CAN bus

- The first input is the data to send
- The second input is the trigger (visible if in "on-demand" mode).
Data is sent upon a rising edge on this signal.

Mailbox configuration | CAN bus configuration

Baud rate (bits/s): 1000000

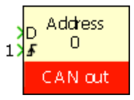
Baud rate is common to all CAN blocks. Changes will be propagated to other CAN blocks.

PLECS block

The CAN baud rate can be configured in the Imperix Controllers' target window (*Coder* → *Coder option* → *Target*).

Signal specification

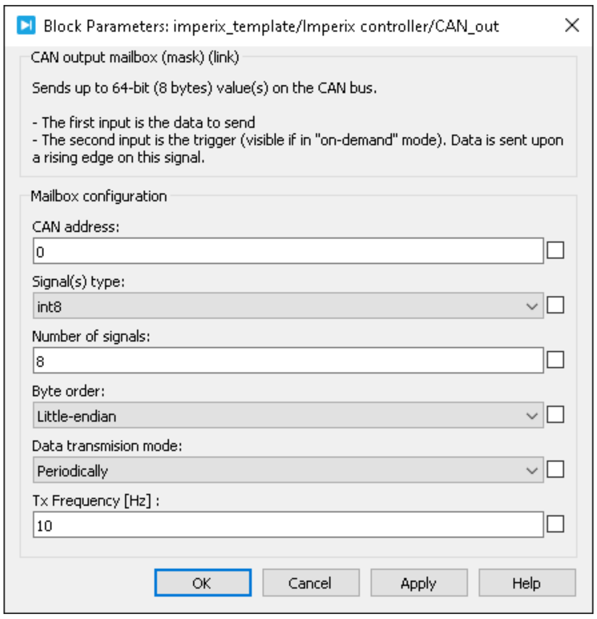
- The data input signal supports a vector of data. The accepted data type is configured by `Signal type` parameter. The vector length can be configured with `Number of signals` parameter.
- The second input is the *send data* signal. It is used to initiate a data transmission when the **on-demand** mode has been selected. Data is sent upon a rising edge on this signal.



Parameters

- `CAN address`: sets the CAN output mailbox address. It also sets the CAN unique identifier of the data frame.
- `Signal Type`: defines the data type accepted in the data input (int8, uint8, int16, uint16, int32, uint32, in64, uint64, float, or double).
- `Number of signals`: sets the vector length of the data output. (from 1 to 8)
- `Byte order`: defines the byte order in which the data will be sent. Either little-endian or big-endian.
- `Tx frequency`: sets the data transmission frequency if the **periodical mode** has been selected.

The total number of bits sent by CAN output mailbox (resulting from `Signal type` and `Number of signals`) can not exceed 64 bits (8 bytes), which is the maximum payload of a CAN frame.



Block Parameters: imperix_template/Imperix controller/CAN_out

CAN output mailbox (mask) (link)

Sends up to 64-bit (8 bytes) value(s) on the CAN bus.

- The first input is the data to send
- The second input is the trigger (visible if in "on-demand" mode). Data is sent upon a rising edge on this signal.

Mailbox configuration

CAN address: 0

Signal(s) type: int8

Number of signals: 8

Byte order: Little-endian

Data transmision mode: Periodically

Tx Frequency [Hz] : 10

C++ functions

`Can_ConfigureCanBus` **— Configure the CAN bus baud rate**

```cpp
void Can_ConfigureCanBus(unsigned int baudrate);
```
`Code language: C++ (cpp)`

Configures the baud rate of the CAN bus. Baud rates up to 1 Mbit/s are supported.

It has to be called in `UserInit()`.

Parameters

- `baudrate`: Baud rate of the CAN bus in bit/s (up to 1'000'000 bit/s)

`Can_ConfigureOutputMailbox` **— Configure a CAN output mailbox**

```cpp
bool Can_ConfigureOutputMailbox(unsigned int maildboxId, unsigned int canAddress, unsigned int dataLength, float m
```

Configures a CAN output mailbox.

It has to be called in `UserInit()`.

Parameters

- `mailboxId`: a unique ID used to distinguish mailboxes from each other. This ID must be unique throughout the code for all ETH and CAN input/output mailboxes.
- `canAddress`: address to whom data will be sent. This parameter is also used as the CAN identifier. Therefore, it also represents the message priority. CAN address range is 0 to 2047 (11 bits).
- `dataLength`: number of bytes of data to transmit (1 to 8 bytes).
- `maxTxFrequency`: maximal frequency at which data can be sent. The frequency must be a multiple of the interrupt frequency. If the requested frequency is not achievable, it will be automatically set to the closest valid frequency.

- `endianness`: defines the frame bytes order. (BIG_ENDIAN or LITTLE_ENDIAN)

Return value

- `bool`: returns false if the configuration fails. The reason may be that too many output mailboxes were created or if `canAddress` is out of range.

`Can_Write` **— Write**

```
// 64-bit (8 bytes) types: uint64, int64 and double
int Can_Write(unsigned int maildboxId, uint64_t data); //8 bytes
int Can_Write(unsigned int maildboxId, int64_t data); //8 bytes
int Can_Write(unsigned int maildboxId, double data); //8 bytesCode language: C++ (cpp)
```

```
// 32-bit (4 bytes) types: int32, uint32 and float
// dataLow  represents the bytes 0, 1, 2 and 3
// dataHigh represents the bytes 4, 5, 6 and 7
int Can_Write(unsigned int maildboxId, unsigned int dataLow, unsigned int dataHigh); //8 bytes
int Can_Write(unsigned int maildboxId, unsigned int dataLow); //4 bytes
int Can_Write(unsigned int maildboxId, int dataLow, int dataHigh); //8 bytes
int Can_Write(unsigned int maildboxId, int dataLow); //4 bytes
int Can_Write(unsigned int maildboxId, float dataLow, float dataHigh); //8 bytes
int Can_Write(unsigned int maildboxId, float dataLow); //4 bytesCode language: C++ (cpp)
```

```
// 16-bit (2 bytes) types: int16 and uint16
// dataLow     represents the bytes 0 and 1
// dataMedLow  represents the bytes 2 and 3
// dataMedHigh represents the bytes 4 and 5
// dataHigh    represents the bytes 6 and 7
int Can_Write(unsigned int maildboxId, uint16_t dataLow, uint16_t dataMedLow, uint16_t dataMedHigh, uint16_t dataH
int Can_Write(unsigned int maildboxId, uint16_t dataLow, uint16_t dataMedLow, uint16_t dataMedHigh); //6 bytes
int Can_Write(unsigned int maildboxId, uint16_t dataLow, uint16_t dataMedLow); //4 bytes
int Can_Write(unsigned int maildboxId, uint16_t dataLow); //2 bytes
int Can_Write(unsigned int maildboxId, int16_t dataLow, int16_t dataMedLow, int16_t dataMedHigh, int16_t dataHigh)
int Can_Write(unsigned int maildboxId, int16_t dataLow, int16_t dataMedLow, int16_t dataMedHigh); //6 bytes
int Can_Write(unsigned int maildboxId, int16_t dataLow, int16_t dataMedLow); //4 bytes
int Can_Write(unsigned int maildboxId, int16_t dataLow); //2 bytesCode language: C++ (cpp)
```

```
// array of uint8
typedef struct {
  uint8_t data[8];
} tCanMsg;

int Can_Write(unsigned int maildboxId, tCanMsg& data); //1 to 8 bytesCode language: C++ (cpp)
```

These functions are used to send data on the CAN bus.

They have to be called during the control interrupt.

Parameters

- `maildboxId`: a unique ID used to distinguish mailboxes from each other. This ID must be unique throughout the code for all ETH and CAN input/output mailboxes.
- `data`: data which will be sent. Several data types and prototypes are available. Note that the prototype used will not affect the length of the data sent, only the `dataLength` in `Can_ConfigureOutputMailbox()` defines it.

Return value

- `int`: returns 1 if the data has successfully been loaded in the write buffer. Returns 0 otherwise.