CAN in - CAN input mailbox

SD016 | Posted on April 2, 2021 | Updated on June 30, 2025

Stéphane LOVEJOY
Senior Software Developer
imperix • in

Table of Contents

- Simulink block
  - Signal specification
  - Parameters
- PLECS block
  - Signal specification
  - Parameters
- C++ functions

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

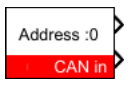
It reads frames from the CAN bus and if the frame address matches the mailbox address, the data is applied to the output port of the block. The CAN input mailbox block features a second output that indicates that new data has been received. The value on the output data port will remain unchanged until new data are received.

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

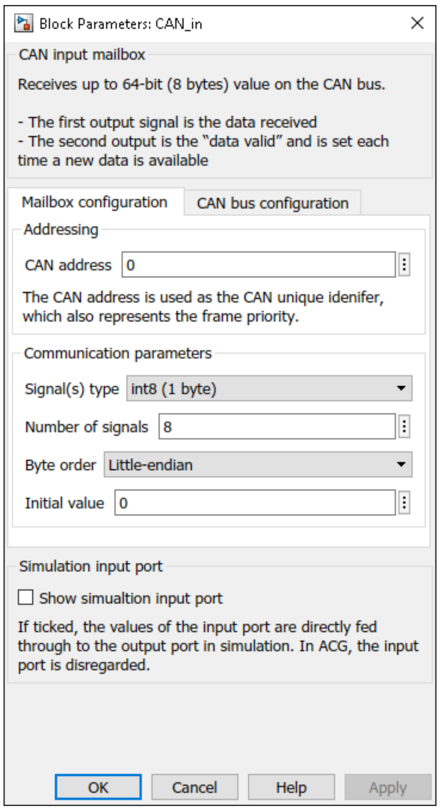
- The data output signal returns a vector containing the data read from the CAN bus. The vector length can be configured with the `Number of signals` parameter. The output data type is configured by the `Signal type` parameter.
- The second signal is the data valid output. It is set to 1 each time new data are available.



Parameters

- `CAN address`: sets the CAN input mailbox address. If a CAN frame with a matching address is received the frame data is applied to the data output port of the block.
- `Signal Type`: defines the type of the data output (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 read. (little-endian or big-endian)
- `Initial value`: sets the initial data value of the data output before any data are received. The value is interpreted as a uint64.
- `Baud rate`: configures the baud rate of the CAN bus. Baud rates up to 1 Mbit/s are supported.

The total number of bits read by a CAN input 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_in

CAN input mailbox

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

- The first output signal is the data received
- The second output is the "data valid" and is set each time a new data is available

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

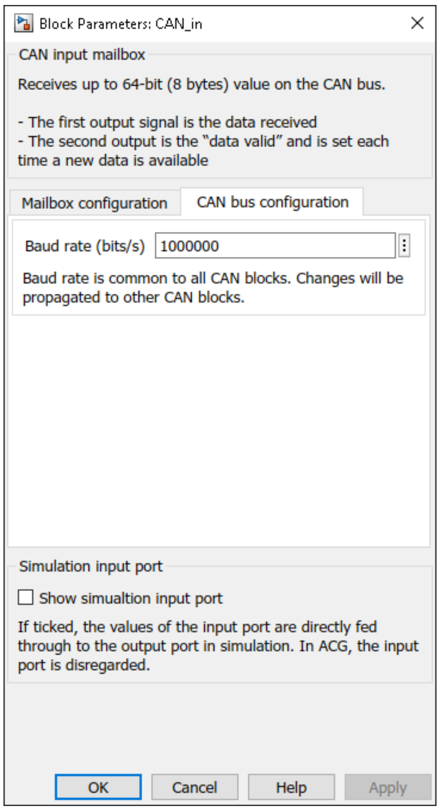
Byte order: Little-endian

Initial value: 0

Simulation input port

☐ Show simualtion input port

If ticked, the values of the input port are directly fed through to the output port in simulation. In ACG, the input port is disregarded.


Block Parameters: CAN_in

CAN input mailbox

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

- The first output signal is the data received
- The second output is the "data valid" and is set each time a new data is available

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.

Simulation input port

☐ Show simualtion input port

If ticked, the values of the input port are directly fed through to the output port in simulation. In ACG, the input port is disregarded.

PLECS block

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

Signal specification

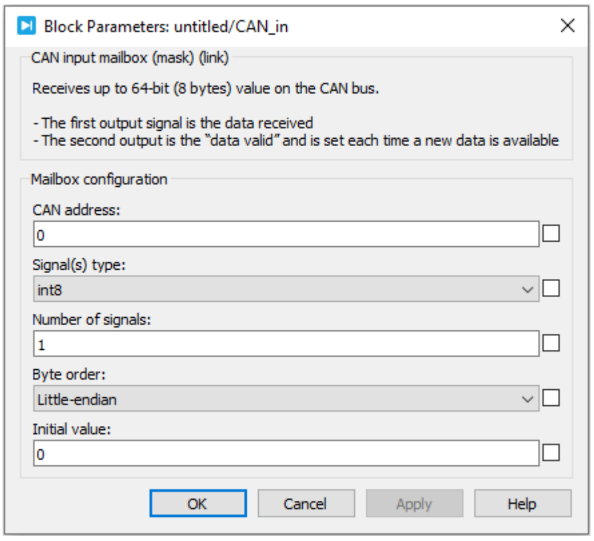
- The data output signal returns a vector containing the data read from the CAN bus. The vector length can be configured with the `Number of signals` parameter. The output data type is configured by the `Signal type` parameter.
- The second signal is the data valid output. It is set to 1 each time new data are available.

Address
0
1
CAN in

Parameters

- `CAN address`: sets the CAN input mailbox address. If a CAN frame with a matching address is received the frame data is applied to the data output port of the block.
- `Signal Type`: defines the type of to the data output (int8, uint8, int16, uint16, int32, uint32, in64, uint64, float, or double).
- `Number of signals`:sets the vector length of the data output.
- `Byte order`: defines the byte order in which the data will be read. (little-endian or big-endian)
- `Initial value`: sets the initial data value of the data output before any data is received. The value is interpreted as a uint64.
- `Baud rate`: configures the baud rate of the CAN bus. Baud rates up to 1 Mbit/s are supported.

The total number of bits read by a CAN input 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: untitled/CAN_in

CAN input mailbox (mask) (link)

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

- The first output signal is the data received
- The second output is the "data valid" and is set each time a new data is available

Mailbox configuration

CAN address: 0

Signal(s) type: int8

Number of signals: 1

Byte order: Little-endian

Initial value: 0

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_ConfigureInputMailbox` **— Configure a CAN input mailbox**

```cpp
bool Can_ConfigureInputMailbox(unsigned int mailboxId, unsigned int canAddress, unsigned int dataLength, tEndianne
```

Configures a CAN input 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`: sets the CAN input mailbox address. If a CAN frame with a matching address is received the frame data is stored in the mailbox and can be read using the `Can_Read()` function. The CAN address range is 0 to 2047 (11 bits).
- `dataLength`: number of bytes of data to read from the CAN bus (1 to 8 bytes).
- `endianness`: defines the bytes order (BIG_ENDIAN or LITTLE_ENDIAN).

Return value

- `bool`: returns false if too many output mailboxes were created or if `canAddress` is out of range.

`Can_ConfigureInputMailboxInitialValue` **— Configure a CAN input mailbox** initial value

```cpp
void Can_ConfigureInputMailboxInitialValue(unsigned int mailboxId, uint64_t data);
```
Code language: C++ (cpp)

Configures the initial value returned by the `Can_Read()` function before any CAN frame is received in the 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.
- `data`: default data which will be returned by any call of `Can_Read()` before data is received on the CAN input mailbox.

`Can_Read` **— Read received value**

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

```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_Read(unsigned int maildboxId, unsigned int& dataLow, unsigned int& dataHigh); //8 bytes
int Can_Read(unsigned int maildboxId, unsigned int& dataLow); //4 bytes
int Can_Read(unsigned int maildboxId, int& dataLow, int& dataHigh); //8 bytes
int Can_Read(unsigned int maildboxId, int& dataLow); //4 bytes
int Can_Read(unsigned int maildboxId, float& dataLow, float& dataHigh); //8 bytes
int Can_Read(unsigned int maildboxId, float& dataLow); //4 bytes
```
Code language: C++ (cpp)

```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_Read(unsigned int maildboxId, uint16_t& dataLow, uint16_t& dataMedLow, uint16_t& dataMedHigh, uint16_t& da
int Can_Read(unsigned int maildboxId, uint16_t& dataLow, uint16_t& dataMedLow, uint16_t& dataMedHigh); //6 bytes
int Can_Read(unsigned int maildboxId, uint16_t& dataLow, uint16_t& dataMedLow); //4 bytes
int Can_Read(unsigned int maildboxId, uint16_t& dataLow); //2 bytes
int Can_Read(unsigned int maildboxId, int16_t& dataLow, int16_t& dataMedLow, int16_t& dataMedHigh, int16_t& dataHi
int Can_Read(unsigned int maildboxId, int16_t& dataLow, int16_t& dataMedLow, int16_t& dataMedHigh); //6 bytes
int Can_Read(unsigned int maildboxId, int16_t& dataLow, int16_t& dataMedLow); //4 bytes
int Can_Read(unsigned int maildboxId, int16_t& dataLow); //2 bytes
```
Code language: C++ (cpp)

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

int Can_Read(unsigned int maildboxId, tCanMsg& canMsg); //1 to 8 bytes
```
Code language: C++ (cpp)

These functions are used to read data received in the input mailbox. The data read will remain unchanged until new data are received.

They have to be called during the control interrupt.

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.
- `data`: data read from the input mailbox. 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_ConfigureInputMailbox`() defines it.

Return value

- `int`: returns 1 if new data is available since the last read. Returns 0 otherwise.