Target Support Package™ 4
Reference Guide

For Use with TI's C5000™

MATLAB®

The MathWorks™
Accelerating the pace of engineering and science

Revision History

September 2009	Online only	New for Version 4.0 (Release 2009b)

Contents

System Requirements

For detailed information about the software and hardware required to use Target Support Package™ software, refer to the Target Support Package system requirements areas on the MathWorks Web site:

- Requirements for Target Support Package: www.mathworks.com/products/target-package/requirements.html
- Requirements for use with TI's C5000™: www.mathworks.com/products/target-package/ti-adaptor/

Block Reference

CAN Message Handling Blocks (canmsglib)

CAN Pack	Pack individual signals into CAN message
CAN Unpack	Unpack individual signals from CAN messages

C5510 DSK (c5510dsk)

C5510 DSK ADC	Configure AIC23 and peripherals to collect data from analog jacks and output digital data
C5510 DSK DAC	Configure AIC23 codec and peripherals to send data stream to output jack

Host Communication (hostcommlib)

Byte Pack	Convert input signals to `uint8` vector
Byte Reversal	Reverse order of bytes in input word
Byte Unpack	Unpack UDP `uint8` input vector into Simulink® data type values
UDP Receive	Receive `uint8` vector as UDP message
UDP Send	Send UDP message

Blocks — Alphabetical List

Byte Pack

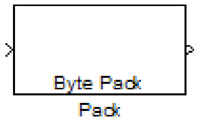
Purpose

Convert input signals to `uint8` vector

Library

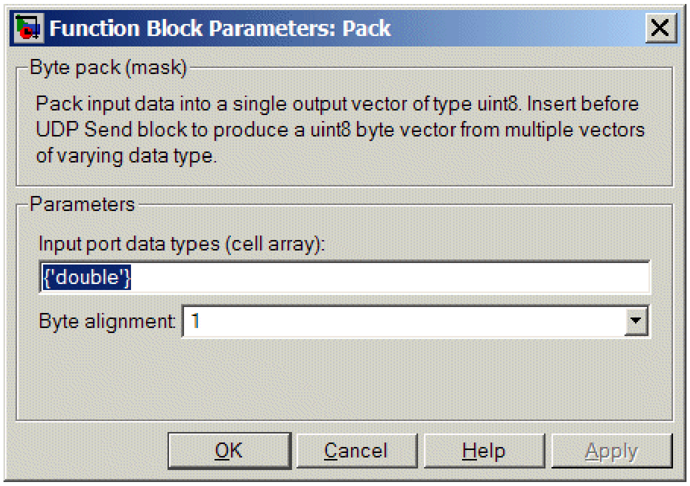
Host Communication (hostcommlib)

Description



Using the input port, the block converts data of one or more data types into a single `uint8` vector for output. With the options available, you specify the input data types and the alignment of the data in the output vector. Because UDP messages are in `uint8` data format, use this block before a UDP Send block to format the data for transmission using the UDP protocol.

Dialog Box



Function Block Parameters: Pack

Byte pack (mask)

Pack input data into a single output vector of type uint8. Insert before UDP Send block to produce a uint8 byte vector from multiple vectors of varying data type.

Parameters

Input port data types (cell array): {'double'}

Byte alignment: 1

Input port data types (cell array)

Specify the data types for the different signals as part of the parameters. The block supports all Simulink data types except characters. Enter the data types as Simulink types in the cell array, such as '`double`' or '`int32`'. The order of the data type entries in the cell array must match the order in which the data arrives at the block input. This block determines the signal sizes automatically. The block always has at least one input port and only one output port.

Byte alignment

This option specifies how to align the data types to form the `uint8` output vector. Select one of the values in bytes from the list.

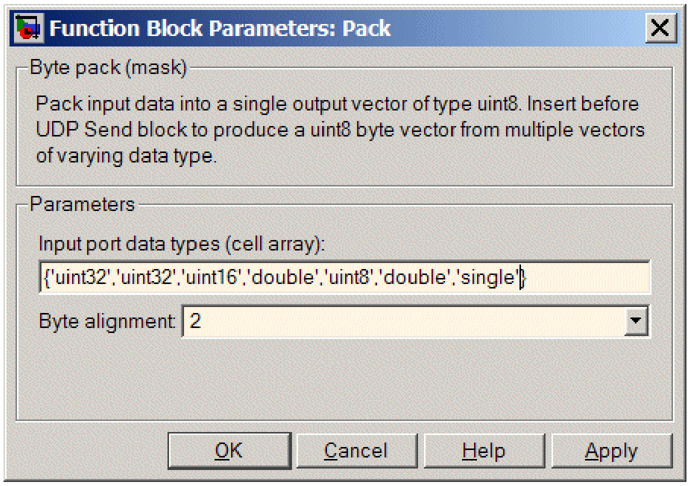
Alignment can occur on 1, 2, 4, or 8-byte boundaries depending on the value you choose. The value defaults to `1`. Given the alignment value, each signal data value begins on multiples of the alignment value. The alignment algorithm ensures that each element in the output vector begins on a byte boundary specified by the alignment value. Byte alignment sets the boundaries relative to the starting point of the vector.

Selecting 1 for **Byte alignment** provides the tightest packing, with no holes between any data types for any combination of data types and signals.

Sometimes, you can have multiple data types of varying lengths. In such cases, specifying a 2-byte alignment can produce 1–byte gaps between uint8 or int8 values and another data type. In the pack implementation, the block copies data to the output data buffer 1 byte at a time. You can specify any of the data alignment options with any of the data types.

Example

Use a cell array to enter input data types in the **Input port data types** parameter. The order of the data types you enter must match the order of the data types at the block input.



Function Block Parameters: Pack

Byte pack (mask)

Pack input data into a single output vector of type uint8. Insert before UDP Send block to produce a uint8 byte vector from multiple vectors of varying data type.

Parameters

Input port data types (cell array): {'uint32','uint32','uint16','double','uint8','double','single'}

Byte alignment: 2

In the cell array, you provide the order in which the block expects to receive data—`uint32`, `uint32`, `uint16`, `double`, `uint8`, `double`, and `single`. With this information, the block automatically provides the proper number of input ports.

Byte alignment equal to `2` specifies that each new value begins 2 bytes from the previous data boundary.

The example shows the following data types:

```
{'uint32','uint32','uint16','double','uint8','double','single'}
```

When the signals are scalar values (no matrices or vectors in this example), the first signal value in the vector starts at 0 bytes. Then, the second signal value starts at 2 bytes, and the third at 4 bytes. Next, the fourth signal value follows at 6 bytes, the fifth at 8 bytes, the sixth at 10 bytes, and the seventh at 12 bytes. As the example shows, the packing algorithm leaves a 1-byte gap between the `uint8` data value and the double value.

See Also

Byte Reversal, Byte Unpack

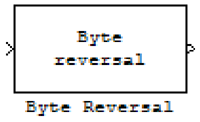
Purpose

Reverse order of bytes in input word

Library

Host Communication (hostcommlib)

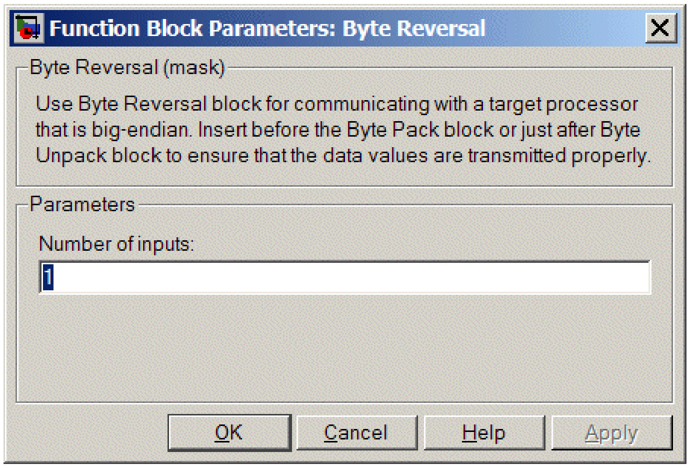
Description



Byte reversal changes the order of the bytes in data you input to the block. Use this block when your process communicates between targets that use different endianness, such as between Intel® processors that are little endian and others that are big endian. Texas Instruments™ processors are little-endian by default.

To exchange data with a processor that has different endianness, place a Byte Reversal block just before the send block and immediately after the receive block.

Dialog Box



Use Byte Reversal block for communicating with a target processor that is big-endian. Insert before the Byte Pack block or just after Byte Unpack block to ensure that the data values are transmitted properly.

Parameters

Number of inputs: 1

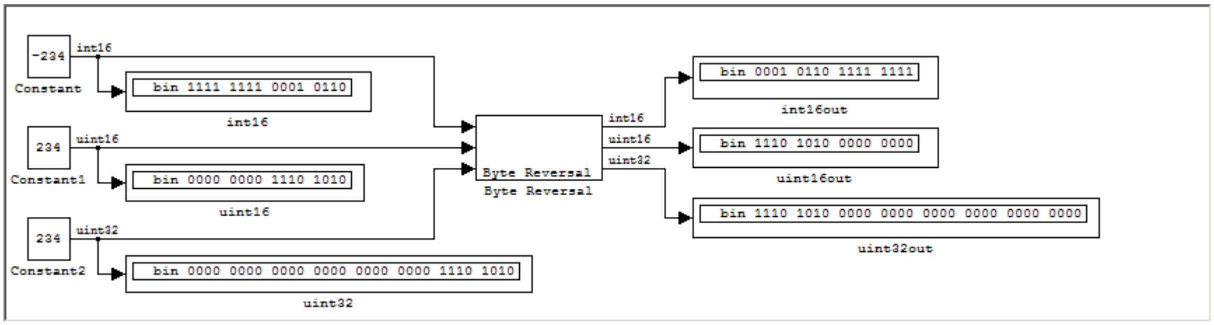
Number of inputs

Specify the number of input ports for the block. The number of input ports adjusts automatically to match value so the number of outputs equals the number of inputs.

When you use more than one input port, each input port maps to the matching output port. Data entering input port 1 leaves through output port 1, and so on.

Reversing the bytes does not change the data type. Input and output retain matching data type.

The following model shows byte reversal in use. In this figure, the input and output ports match for each path.



See Also

Byte Pack, Byte Unpack

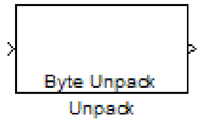
Purpose

Unpack UDP `uint8` input vector into Simulink data type values

Library

Host Communication (hostcommlib)

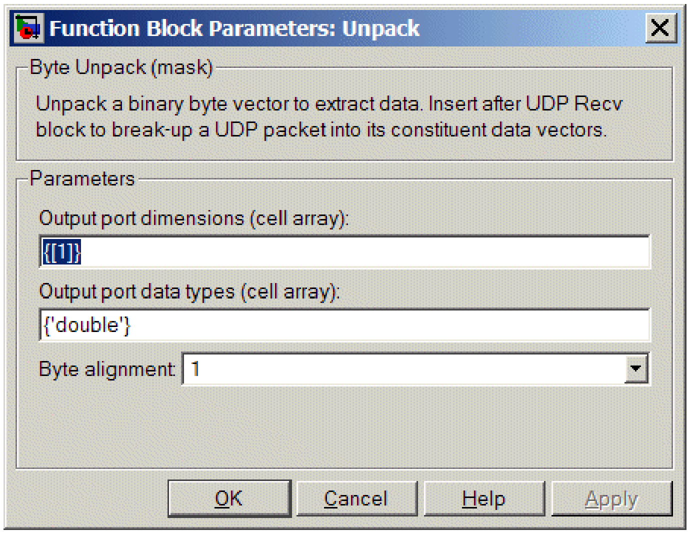
Description



Byte Unpack is the inverse of the Byte Pack block. It takes a UDP message from a UDP receive block as a `uint8` vector, and outputs Simulink data types in various sizes depending on the input vector.

The block supports all Simulink data types.

Dialog Box



Function Block Parameters: Unpack

Byte Unpack (mask)

Unpack a binary byte vector to extract data. Insert after UDP Recv block to break-up a UDP packet into its constituent data vectors.

Parameters

Output port dimensions (cell array): {[1]}

Output port data types (cell array): {'double'}

Byte alignment: 1

Output port dimensions (cell array)

Containing a cell array, each element in the array specifies the dimension that the MATLAB® `size` function returns for the corresponding signal. Usually you use the same dimensions as you set for the corresponding Byte Pack block in the model. Entering one value means that the block applies that dimension to all data types.

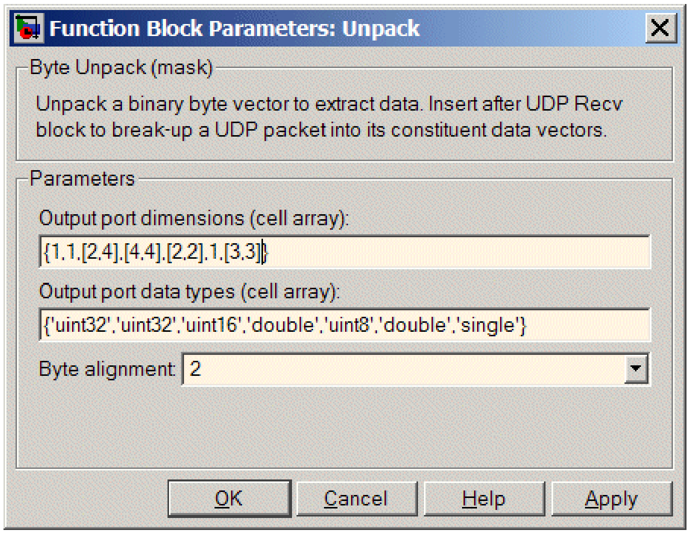
Output port data types (cell array)
Specify the data types for the different input signals to the Pack block. The block supports all Simulink data types—`single`, `double`, `int8`, `uint8`, `int16`, `uint16`, `int32`, and `uint32`, and `Boolean`. The entry here is the same as the Input port data types parameter in the Byte Pack block in the model. You can enter one data type and the block applies that type to all output ports.

Byte Alignment
This option specifies how to align the data types to form the input `uint8` vector. Match this setting with the corresponding Byte Pack block alignment value of `1`, `2`, `4`, or `8` bytes.

Example

This figure shows the Byte Unpack block that corresponds to the example in the Byte Pack example. The **Output port data types (cell array)** entry shown is the same as the **Input port data types (cell array)** entry in the Byte Pack block

```
{'uint32','uint32','uint16','double','uint8','double','single'}.
```


Function Block Parameters: Unpack

Byte Unpack (mask)

Unpack a binary byte vector to extract data. Insert after UDP Recv block to break-up a UDP packet into its constituent data vectors.

Parameters

Output port dimensions (cell array): {1,1,[2,4],[4,4],[2,2],1,[3,3]}

Output port data types (cell array): {'uint32','uint32','uint16','double','uint8','double','single'}

Byte alignment: 2

In addition, the **Byte alignment** setting matches as well. **Output port dimensions (cell array)** now includes scalar values and matrices to demonstrate entering nonscalar values. The example for the Byte Pack block assumed only scalar inputs.

See Also

Byte Pack, Byte Reversal

C5510 DSK ADC

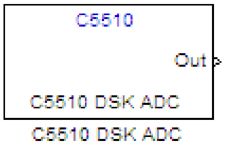
Purpose

Configure AIC23 and peripherals to collect data from analog jacks and output digital data

Library

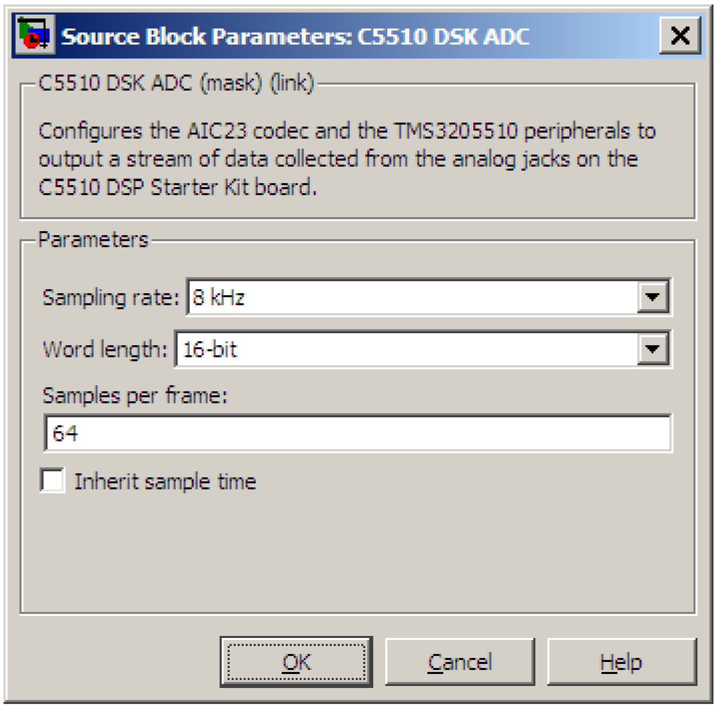
TMS320VC5510 DSP Starter Kit (DSK) (c5510dsk)

Description



Configures the AIC23 codec and the TMS320C5510 peripherals to output a stream of digital data. The block collects this data from the analog jacks on the C5510 DSP Starter Kit board.

Dialog Box



Source Block Parameters: C5510 DSK ADC

C5510 DSK ADC (mask) (link)

Configures the AIC23 codec and the TMS3205510 peripherals to output a stream of data collected from the analog jacks on the C5510 DSP Starter Kit board.

Parameters

Sampling rate: 8 kHz

Word length: 16-bit

Samples per frame: 64

☐ Inherit sample time

Sampling rate
Set the rate at which the analog-to-digital converter samples the analog input. A higher rate increases the resolution of the data the ADC outputs.

Word length
Set the number of data bits the ADC creates for each sample. Increasing the word length increases the accuracy of the data in each sample. If your model also contains a DAC block, set the word length in the DAC block to match that of the ADC block.

Samples per frame
Set the number of samples the ADC buffers internally before it sends the digitized signals, as a frame vector, to the next block in the model. This value defaults to `64` samples per frame. The frame rate depends on the sample rate and frame size. Thus, if you set **Sampling Rate** to 8 kHz, and **Samples per frame** to 32, the resulting frame rate is 250 frames per second (8000/32 = 250).

Inherit sample time
Select whether the block inherits the sample time from the model base rate or from the Simulink base rate. You can locate the Simulink base rate in the Solver options in Configuration Parameters. Selecting Inherit sample time directs the block to use the specified rate in model configuration. Entering -1 configures the block to accept the sample rate from the upstream HWI, Task, or Triggered Task blocks.

See Also

C5510 DSK DAC

C5510 DSK DAC

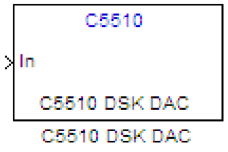
Purpose

Configure AIC23 codec and peripherals to send data stream to output jack

Library

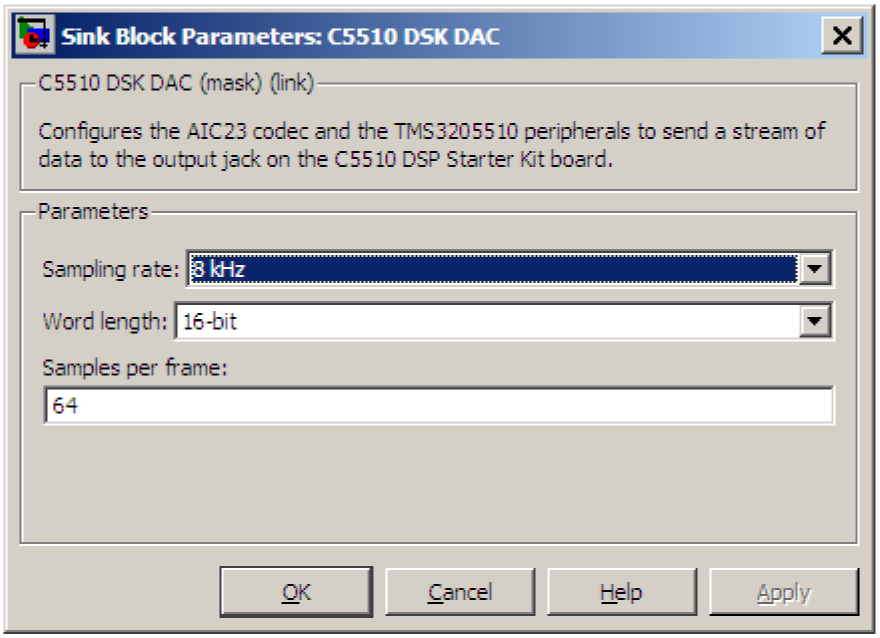
TMS320VC5510 DSP Starter Kit (DSK) (c5510dsk)

Description



Configures the AIC23 codec and the TMS3205510 peripherals to send a stream of data to the output jack on the C5510 DSP Starter Kit board.

Dialog Box



C5510 DSK DAC (mask) (link)

Configures the AIC23 codec and the TMS3205510 peripherals to send a stream of data to the output jack on the C5510 DSP Starter Kit board.

Parameters

Sampling rate: 8 kHz

Word length: 16-bit

Samples per frame: 64

Sampling Rate
Set the rate at which the digital-to-analog converter receives each data sample. If your model contains an ADC block, set this value to match the sampling rate of the ADC block.

Word length
Set the number of bits in each data input sample the DAC. If your model also contains an ADC block, set the word length in the DAC block to match that of the ADC block. If you do not use an accurate setting, the DAC cannot convert the data correctly.

Samples per frame
Set the number of samples per data input frame. Match this value with the value of the block creating the data frames. This value defaults to `64` samples per frame.

See Also

C5510 DSK ADC

CAN Pack

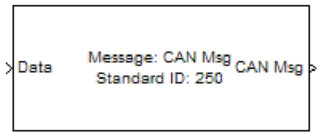
Purpose

Pack individual signals into CAN message

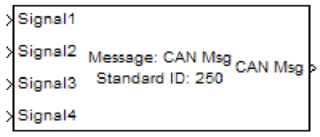
Library

CAN Communication

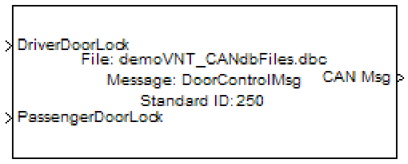
Description



CAN Pack
(With raw data input)



CAN Pack
(With manually specified data input)

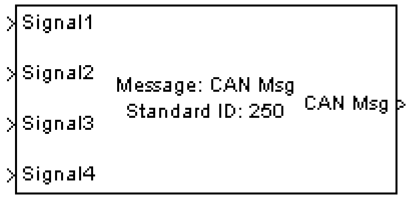


CAN Pack
(With CANdb specified data input)

The CAN Pack block loads signal data into a message at specified intervals during the simulation.

Note To use this block, you also need a license for Simulink software.

CAN Pack block has one input port by default. The number of input ports is dynamic and depends on the number of signals you specify for the block. For example, if your block has four signals, it has four input ports.



CAN Pack

This block has one output port, CAN Msg. The CAN Pack block takes the specified input parameters and packs the signals into a message.

Other Supported Features

The CAN Pack block supports:

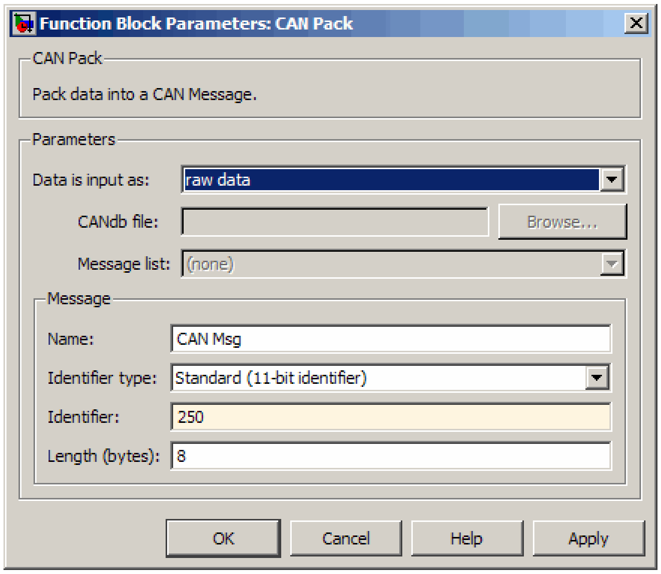
- The use of Simulink® Accelerator™ mode. Using this feature, you can speed up the execution of Simulink models.
- The use of model referencing. Using this feature, your model can include other Simulink models as modular components.
- Code generation using Real-Time Workshop® to deploy models to targets.

Note Code generation is not supported if your signal information consists of signed or unsigned integers greater than 32-bits long.

For more information on these features, see the Simulink documentation.

Dialog Box

Use the Function Block Parameters dialog box to select your CAN Pack block parameters.


Function Block Parameters: CAN Pack

CAN Pack

Pack data into a CAN Message.

Parameters

Data is input as: raw data

CANdb file:

Message list: (none)

Message

Name: CAN Msg

Identifier type: Standard (11-bit identifier)

Identifier: 250

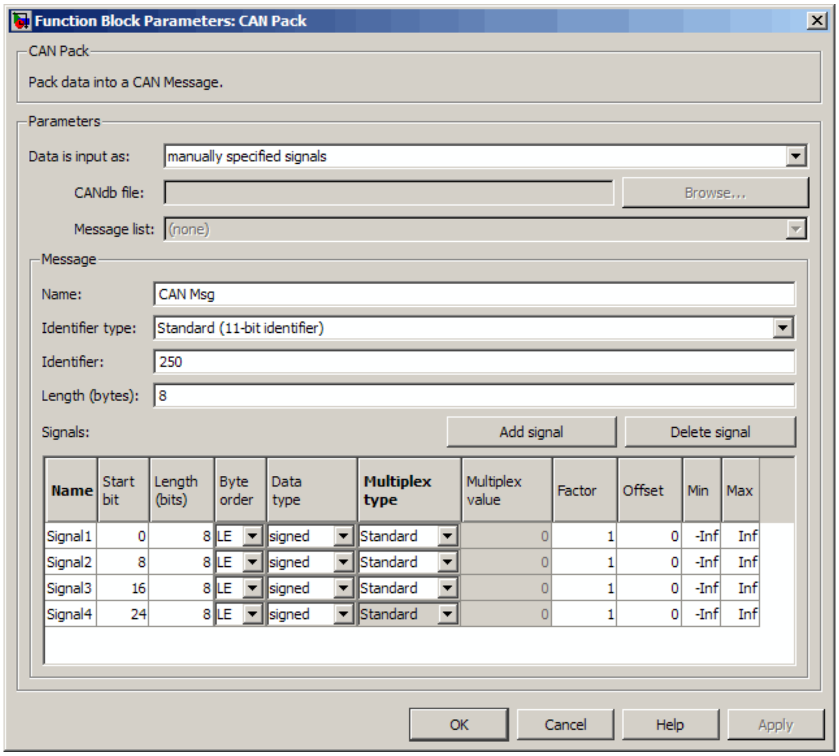
Length (bytes): 8

Parameters

Data is input as

Select your data signal:

- **raw data**: Input data as a uint8 vector array. If you select this option, you only specify the message fields. All other signal parameter fields are unavailable. This option opens only one input port on your block.
- **manually specified signals**: Allows you to specify data signal definitions. If you select this option, use the **Signals** table to create your signals. The number of input ports on your block depends on the number of signals you specify.



CAN Pack

Pack data into a CAN Message.

Parameters

Data is input as: manually specified signals

CANdb file:

Message list: (none)

Message

Name: CAN Msg

Identifier type: Standard (11-bit identifier)

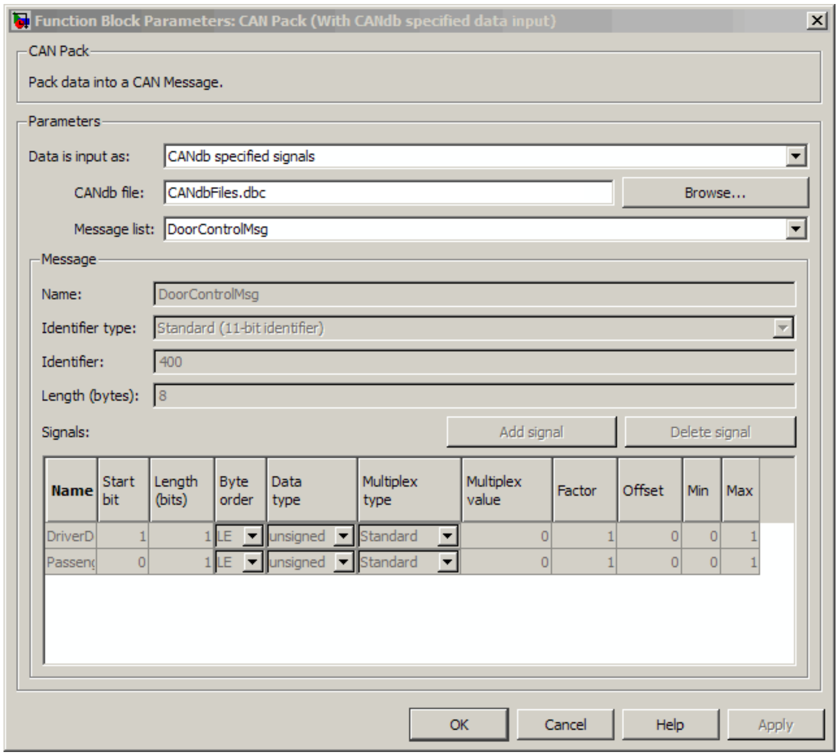
Identifier: 250

Length (bytes): 8

Signals:

Name	Start bit	Length (bits)	Byte order	Data type	Multiplex type	Multiplex value	Factor	Offset	Min	Max
Signal1	0	8	LE	signed	Standard	0	1	0	-Inf	Inf
Signal2	8	8	LE	signed	Standard	0	1	0	-Inf	Inf
Signal3	16	8	LE	signed	Standard	0	1	0	-Inf	Inf
Signal4	24	8	LE	signed	Standard	0	1	0	-Inf	Inf

- **CANdb specified signals**: Allows you to specify a CAN database file that contains message and signal definitions. If you select this option, select a CANdb file. The number of input ports on your block depends on the number of signals specified in the CANdb file for the selected message.



Function Block Parameters: CAN Pack (With CANdb specified data input)

CAN Pack

Pack data into a CAN Message.

Parameters

Data is input as: CANdb specified signals

CANdb file: CANdbFiles.dbc

Message list: DoorControlMsg

Message

Name: DoorControlMsg

Identifier type: Standard (11-bit identifier)

Identifier: 400

Length (bytes): 8

Signals:

Name	Start bit	Length (bits)	Byte order	Data type	Multiplex type	Multiplex value	Factor	Offset	Min	Max
DriverD	1	1	LE	unsigned	Standard	0	1	0	0	1
Passen(	0	1	LE	unsigned	Standard	0	1	0	0	1

CANdb file

This option is available if you specify that your data is input via a CANdb file in the **Data is input as** list. Click **Browse** to find the appropriate CANdb file on your system. The message list specified in the CANdb file populates the **Message** section of the dialog box. The CANdb file also populates the **Signals** table for the selected message.

Message list

This option is available if you specify that your data is input via a CANdb file in the **Data is input as** field and you select a CANdb file in the **CANdb file** field. Select the message to display signal details in the **Signals** table.

Message

Name
: Specify a name for your CAN message. The default is `CAN Msg`. This option is available if you choose to input raw data or manually specify signals. This option in unavailable if you choose to use signals from a CANdb file.

Identifier type
: Specify whether your CAN message identifier is a `Standard` or an `Extended` type. The default is `Standard`. A standard identifier is an 11-bit identifier and an extended identifier is a 29-bit identifier. This option is available if you choose to input raw data or manually specify signals. For `CANdb specified signals`, the **Identifier type** inherits the type from the database.

Identifier
: Specify your CAN message ID. This number must be a positive integer from 0 through 2047 for a standard identifier and from 0 through 536870911 for an extended identifier. You can also specify hexadecimal values using the `hex2dec` function. This option is available if you choose to input raw data or manually specify signals.

Length (bytes)
: Specify the length of your CAN message from 0 to 8 bytes. If you are using `CANdb specified signals` for your data input, the CANdb file defines the length of your message. If not, this field defaults to `8`. This option is available if you choose to input raw data or manually specify signals.

Signals Table

This table appears if you choose to specify signals manually or define signals using a CANdb file.

If you are using a CANdb file, the data in the file populates this table automatically and you cannot edit any fields. To edit signal information, switch to manually specified signals.

If you have selected to specify signals manually, create your signals manually in this table. Each signal you create has the following values:

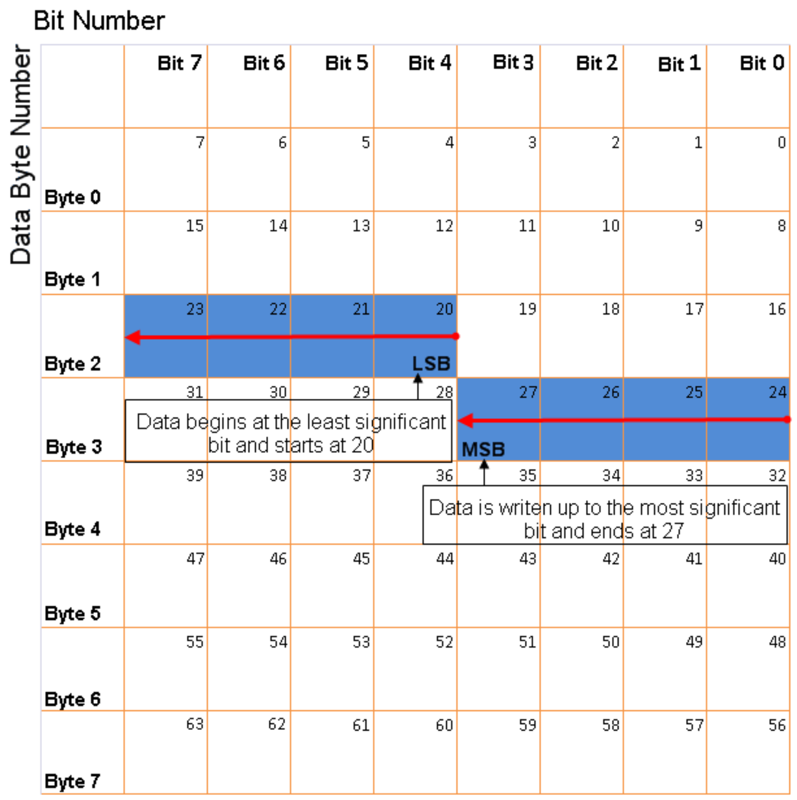
Name
: Specify a descriptive name for your signal. The Simulink block in your model displays this name. The default is `Signal [row number]`.

Start bit
: Specify the start bit of the data. The start bit is the least significant bit counted from the start of the message data. The start bit must be an integer from 0 through 63.

Length (bits)
: Specify the number of bits the signal occupies in the message. The length must be an integer from 1 through 64.

Byte order
: Select either of the following options:

  - `LE`: Where the byte order is in little-endian format (Intel). In this format you count bits from the start, which is the least significant bit, to the most significant bit, which has the highest bit index. For example, if you pack one byte of data in little-endian format, with the start bit at 20, the data bit table resembles this figure.



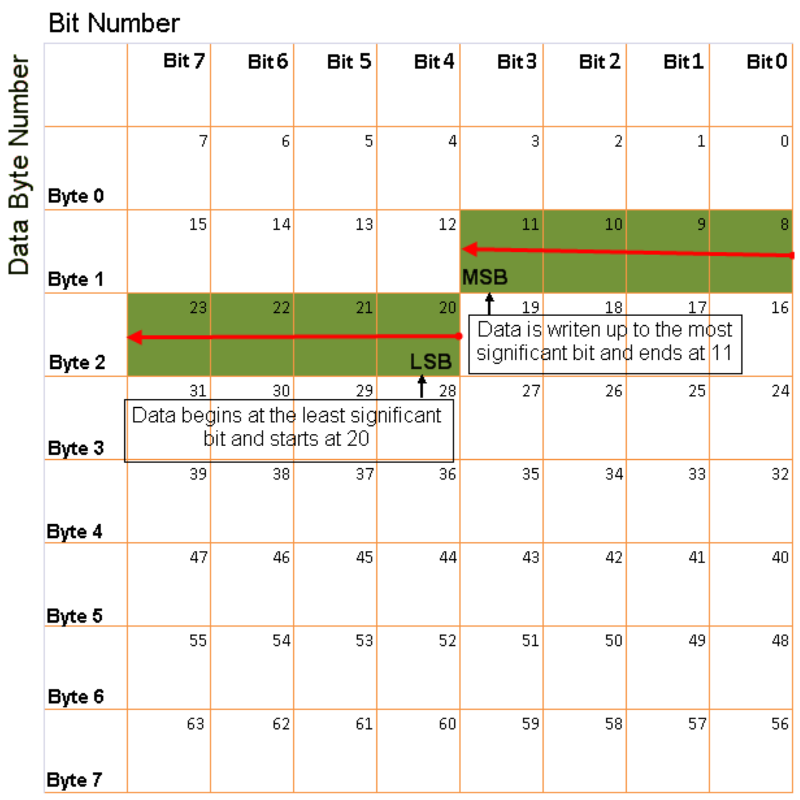
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	7	6	5	4	3	2	1	0
Byte 1	15	14	13	12	11	10	9	8
Byte 2	23	22	21	20	19	18	17	16
Byte 3	31	30	29	28	27	26	25	24
Byte 4	39	38	37	36	35	34	33	32
Byte 5	47	46	45	44	43	42	41	40
Byte 6	55	54	53	52	51	50	49	48
Byte 7	63	62	61	60	59	58	57	56

Data begins at the least significant bit and starts at 20

Data is writen up to the most significant bit and ends at 27

Little Endian Byte Order Counted from the Least Significant Bit to the Highest Address

- BE: Where byte order is in big-endian format (Motorola®). In this format you count bits from the start, which is the least significant bit, to the most significant bit. For example, if you pack one byte of data in big-endian format, with the start bit at 20, the data bit table resembles this figure.



Big Endian Byte Order Counted from the Least Significant Bit to the Lowest Address

Data type

Specify how the signal interprets the data in the allocated bits. Choose from:

- `signed` (default)
- `unsigned`
- `single`
- `double`

Multiplex type

Specify how the block packs the signals into the CAN message at each timestep:

- `Standard`: The signal is always packed at each timestep.
- `Multiplexor`: The `Multiplexor` signal, or the mode signal is always packed. You can specify only one `Multiplexor` signal per message.
- `Multiplexed`: The signal is packed if the value of the `Multiplexor` signal (mode signal) at run time matches the configured **Multiplex value** of this signal.

For example, a message has four signals with the following types and values.

Signal Name	Multiplex Type	Multiplex Value
Signal-A	Standard	N/A
Signal-B	Multiplexed	1
Signal-C	Multiplexed	0
Signal-D	Multiplexor	N/A

In this example

- The block packs Signal-A (Standard signal) and Signal-D (Multiplexor signal) in every timestep.
- If the value of Signal-D is 1 at a particular timestep, then the block packs Signal-B along with Signal-A and Signal-D in that timestep.
- If the value of Signal-D is 0 at a particular timestep, then the block packs Signal-C along with Signal-A and Signal-D in that timestep.
- If the value of Signal-D is not 1 or 0, the block does not pack either of the Multiplexed signals in that timestep.

Multiplex value

This option is available only if you have selected the **Multiplex type** to be `Multiplexed`. The value you provide here must match the `Multiplexor` signal value at run time for the block to pack the `Multiplexed` signal. The **Multiplex value** must be a positive integer or zero.

Factor

Specify the **Factor** value to apply to convert the physical value (signal value) to the raw value packed in the message. See “Conversion Formula” on page 3-24 to understand how physical values are converted to raw values packed into a message.

Offset

Specify the **Offset** value to apply to convert the physical value (signal value) to the raw value packed in the message. See “Conversion Formula” on page 3-24 to understand how physical values are converted to raw values packed into a message.

Min

Specify the minimum physical value of the signal. The default value is `-inf` (negative infinity). You can specify any number for the minimum value. See “Conversion Formula” on page 3-24 to understand how physical values are converted to raw values packed into a message.

Max

Specify the maximum physical value of the signal. The default value is `inf`. You can specify any number for the maximum value. See “Conversion Formula” on page 3-24 to understand how physical values are converted to raw values packed into a message.

Conversion Formula

The conversion formula is

```
raw_value = (physical_value - Offset) / Factor
```

where `physical_value` is the value of the signal after it is saturated using the specified **Min** and **Max** values. `raw_value` is the packed signal value.

See Also

CAN Unpack

CAN Unpack

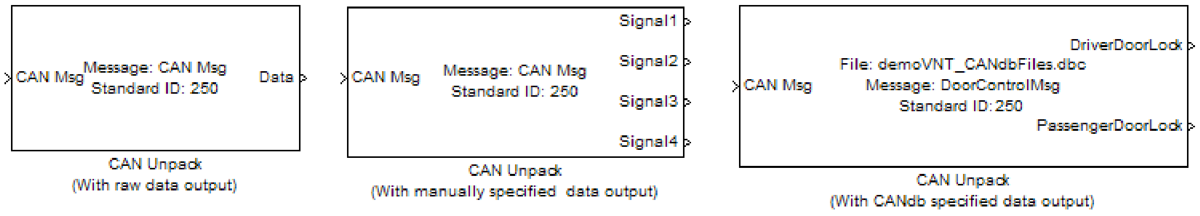
Purpose

Unpack individual signals from CAN messages

Library

CAN Communication

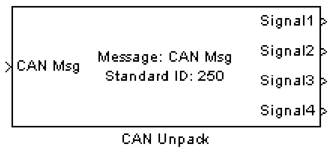
Description



The CAN Unpack block unpacks a CAN message into signal data using the specified output parameters at every timestep. Data is output as individual signals.

Note To use this block, you also need a license for Simulink software.

The CAN Unpack block has one output port by default. The number of output ports is dynamic and depends on the number of signals you specify for the block to output. For example, if your block has four signals, it has four output ports.



Other Supported Features

The CAN Unpack block supports:

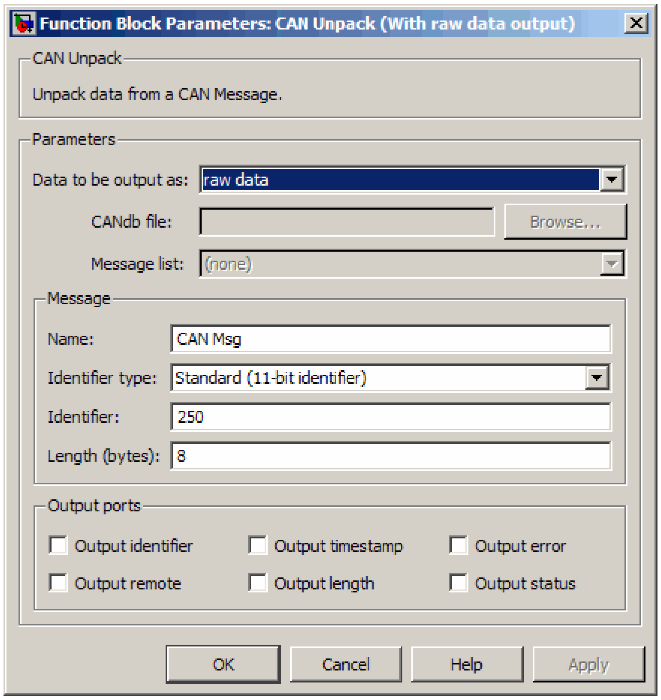
- The use of Simulink Accelerator mode. Using this feature, you can speed up the execution of Simulink models.
- The use of model referencing. Using this feature, your model can include other Simulink models as modular components.
- Code generation using Real-Time Workshop to deploy models to targets.

Note Code generation is not supported if your signal information consists of signed or unsigned integers greater than 32-bits long.

For more information on these features, see the Simulink documentation.

Dialog Box

Use the Function Block Parameters dialog box to select your CAN message unpacking parameters.


Function Block Parameters: CAN Unpack (With raw data output)
CAN Unpack
Browse...
OK
Cancel
Help
Apply

Unpack data from a CAN Message.

Parameters

Data to be output as: raw data

CANdb file:

Message list: (none)

Message

Name: CAN Msg

Identifier type: Standard (11-bit identifier)

Identifier: 250

Length (bytes): 8

Output ports

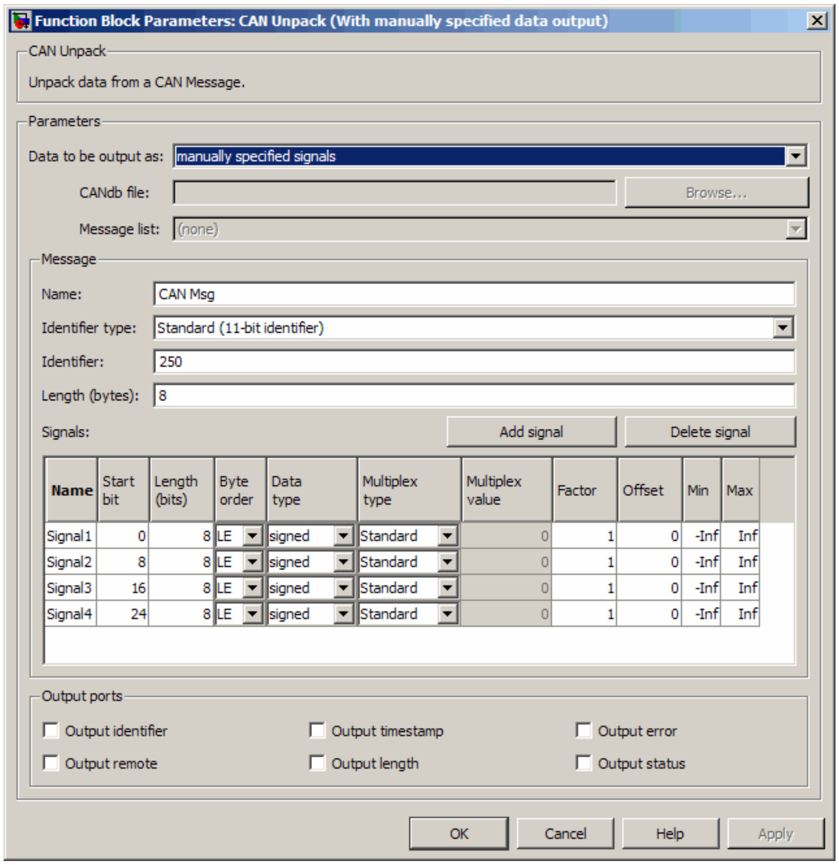
- [ ] Output identifier
- [ ] Output timestamp
- [ ] Output error
- [ ] Output remote
- [ ] Output length
- [ ] Output status

Parameters

Data to be output as

Select your data signal:

- **raw data**: Output data as a uint8 vector array. If you select this option, you only specify the message fields. All other signal parameter fields are unavailable. This option opens only one output port on your block.
- **manually specified signals**: Allows you to specify data signals. If you select this option, use the `Signals` table to create your signals message manually.


Function Block Parameters: CAN Unpack (With manually specified data output)

CAN Unpack

Unpack data from a CAN Message.

Parameters

Data to be output as: manually specified signals

CANdb file:

Message list: (none)

Message

Name: CAN Msg

Identifier type: Standard (11-bit identifier)

Identifier: 250

Length (bytes): 8

Signals:

Name	Start bit	Length (bits)	Byte order	Data type	Multiplex type	Multiplex value	Factor	Offset	Min	Max
Signal1	0	8	LE	signed	Standard	0	1	0	-Inf	Inf
Signal2	8	8	LE	signed	Standard	0	1	0	-Inf	Inf
Signal3	16	8	LE	signed	Standard	0	1	0	-Inf	Inf
Signal4	24	8	LE	signed	Standard	0	1	0	-Inf	Inf

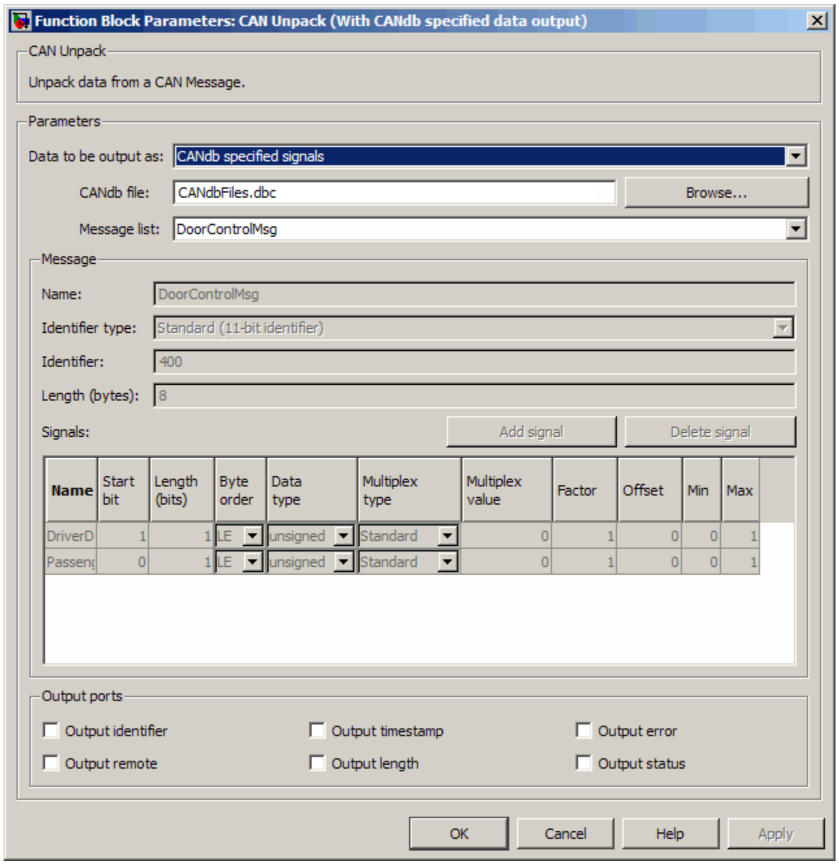
Output ports

☐ Output identifier ☐ Output timestamp ☐ Output error

☐ Output remote ☐ Output length ☐ Output status

The number of output ports on your block depends on the number of signals you specify. For example, if you specify four signals, your block has four output ports.

- **CANdb specified signals**: Allows you to specify a CAN database file that contains data signals. If you select this option, select a CANdb file.



Function Block Parameters: CAN Unpack (With CANdb specified data output)

CAN Unpack

Unpack data from a CAN Message.

Parameters

Data to be output as: CANdb specified signals

CANdb file: CANdbFiles.dbc

Message list: DoorControlMsg

Message

Name: DoorControlMsg

Identifier type: Standard (11-bit identifier)

Identifier: 400

Length (bytes): 8

Signals:

Name	Start bit	Length (bits)	Byte order	Data type	Multiplex type	Multiplex value	Factor	Offset	Min	Max
DriverD	1	1	LE	unsigned	Standard	0	1	0	0	1
Passeng	0	1	LE	unsigned	Standard	0	1	0	0	1

Output ports

- [ ] Output identifier
- [ ] Output timestamp
- [ ] Output error
- [ ] Output remote
- [ ] Output length
- [ ] Output status

The number of output ports on your block depends on the number of signals specified in the CANdb file. For example, if the selected message in the CANdb file has four signals, your block has four output ports.

CANdb file

This option is available if you specify that your data is input via a CANdb file in the **Data to be output as** list. Click **Browse** to find the appropriate CANdb file on your system. The messages and signal definitions specified in the CANdb file populate the

Message section of the dialog box. The signals specified in the CANdb file populate **Signals** table.

Message list
: This option is available if you specify that your data is to be output as a CANdb file in the **Data to be output as** list and you select a CANdb file in the **CANdb file** field. You can select the message that you want to view. The **Signals** table then displays the details of the selected message.

Message

Name
: Specify a name for your CAN message. The default is `CAN Msg`. This option is available if you choose to output raw data or manually specify signals.

Identifier type
: Specify whether your CAN message identifier is a `Standard` or an `Extended` type. The default is `Standard`. A standard identifier is an 11-bit identifier and an extended identifier is a 29-bit identifier. This option is available if you choose to output raw data or manually specify signals. For CANdb-specified signals, the **Identifier type** inherits the type from the database.

Identifier
: Specify your CAN message ID. This number must be a integer from 0 through 2047 for a standard identifier and from 0 through 536870911 for an extended identifier. If you specify `1`, the block unpacks all messages that match the length specified for the message. You can also specify hexadecimal values using the `hex2dec` function. This option is available if you choose to output raw data or manually specify signals.

Length (bytes)
: Specify the length of your CAN message from 0 to 8 bytes. If you are using `CANdb specified signals` for your output data, the CANdb file defines the length of your message. If not, this field

defaults to `8`. This option is available if you choose to output raw data or manually specify signals.

Signals Table

This table appears if you choose to specify signals manually or define signals using a CANdb file.

If you are using a CANdb file, the data in the file populates this table automatically and you cannot edit any fields. To edit signal information, switch to manually specified signals.

If you have selected to specify signals manually, create your signals manually in this table. Each signal you create has the following values:

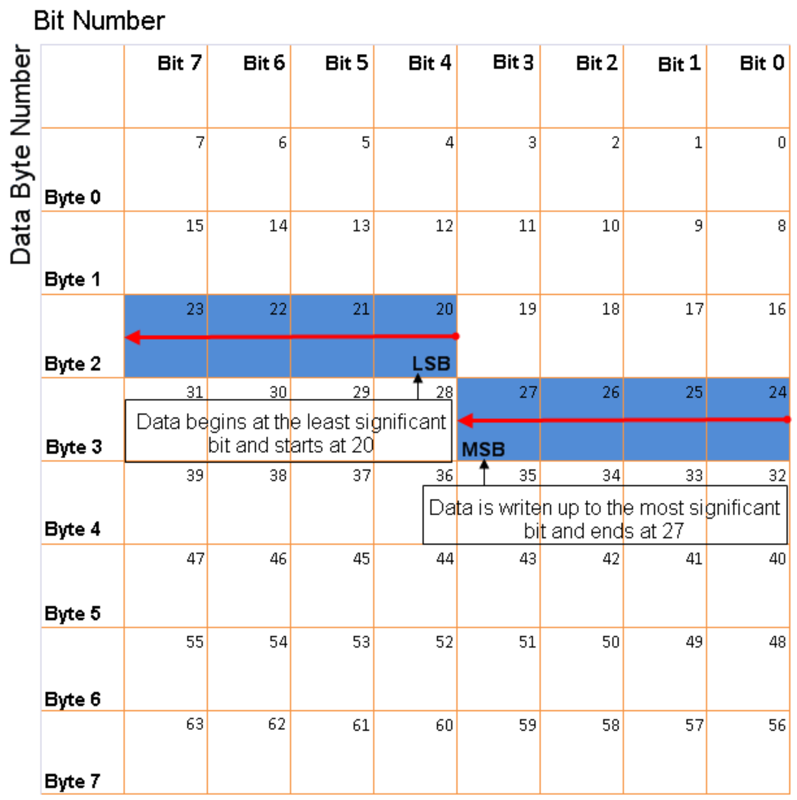
Name
Specify a descriptive name for your signal. The Simulink block in your model displays this name. The default is `Signal [row number]`.

Start bit
Specify the start bit of the data. The start bit is the least significant bit counted from the start of the message. The start bit must be an integer from 0 through 63.

Length (bits)
Specify the number of bits the signal occupies in the message. The length must be an integer from 1 through 64.

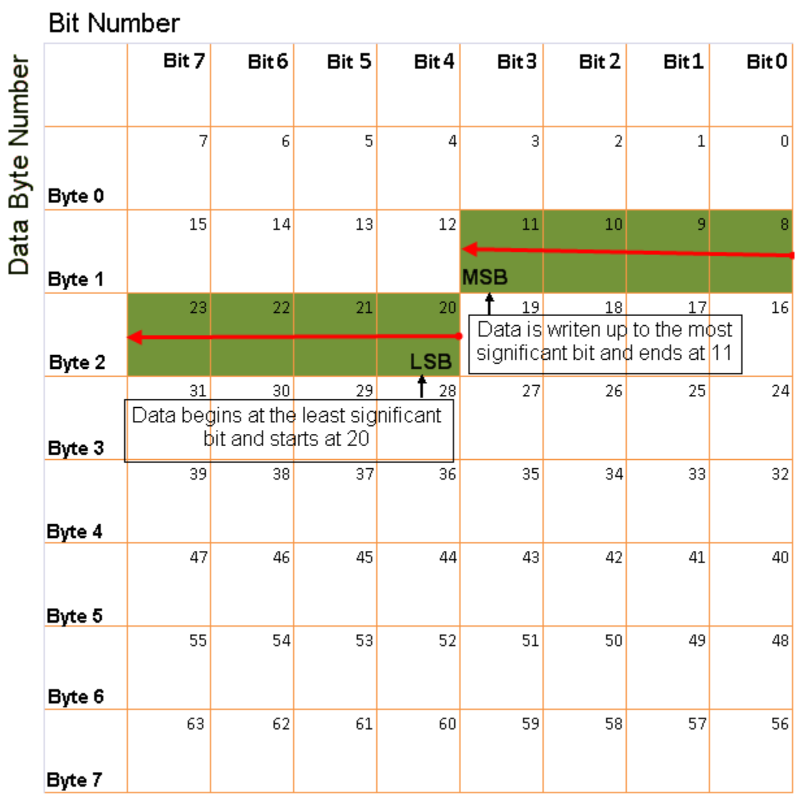
Byte order
Select either of the following options:

- `LE`: Where the byte order is in little-endian format (Intel). In this format you count bits from the start, which is the least significant bit, to the most significant bit, which has the highest bit index. For example, if you pack one byte of data in little-endian format, with the start bit at 20, the data bit table resembles this figure.



Little Endian Byte Order Counted from the Least Significant Bit to the Highest Address

- BE: Where the byte order is in big-endian format (Motorola). In this format you count bits from the start, which is the least significant bit, to the most significant bit. For example, if you pack one byte of data in big-endian format, with the start bit at 20, the data bit table resembles this figure.



Big Endian Byte Order Counted from the Least Significant Bit to the Lowest Address

Data type

Specify how the signal interprets the data in the allocated bits. Choose from:

- `signed` (default)
- `unsigned`
- `single`
- `double`

Multiplex type

Specify how the block unpacks the signals from the CAN message at each timestep:

- `Standard`: The signal is always unpacked at each timestep.
- `Multiplexor`: The `Multiplexor` signal, or the mode signal is always unpacked. You can specify only one `Multiplexor` signal per message.
- `Multiplexed`: The signal is unpacked if the value of the `Multiplexor` signal (mode signal) at run time matches the configured **Multiplex value** of this signal.

For example, if a message has four signals with the following values.

Signal Name	Multiplex Type	Multiplex Value
Signal-A	Standard	N/A
Signal-B	Multiplexed	1
Signal-C	Multiplexed	0
Signal-D	Multiplexor	N/A

In this example

- The block unpacks Signal-A (Standard signal) and Signal-D (Multiplexor signal) in every timestep.
- If the value of Signal-D is 1 at a particular timestep, then the block unpacks Signal-B along with Signal-A and Signal-D in that timestep.
- If the value of Signal-D is 0 at a particular timestep, then the block unpacks Signal-C along with Signal-A and Signal-D in that timestep.
- If the value of Signal-D is not 1 or 0, the block does not unpack either of the Multiplexed signals in that timestep.

Multiplex value

This option is available only if you have selected the **Multiplex type** to be `Multiplexed`. The value you provide here must match the `Multiplexor` signal value at run time for the block to unpack the `Multiplexed` signal. The **Multiplex value** must be a positive integer or zero.

Factor

Specify the **Factor** value applied to convert the unpacked raw value to the physical value (signal value). See “Conversion Formula” on page 3-37 to understand how unpacked raw values are converted to physical values.

Offset

Specify the **Offset** value applied to convert the physical value (signal value) to the unpacked raw value. See “Conversion Formula” on page 3-37 to understand how unpacked raw values are converted to physical values.

Min

Specify the minimum raw value of the signal. The default value is `-inf` (negative infinity). You can specify any number for the minimum value. See “Conversion Formula” on page 3-37 to understand how unpacked raw values are converted to physical values.

Max

Specify the maximum raw value of the signal. The default value is `inf`. You can specify any number for the maximum value. See “Conversion Formula” on page 3-37 to understand how unpacked raw values are converted to physical values.

Output Ports

Selecting an **Output ports** option adds an output port to your block.

Output identifier

Select this option to output a CAN message identifier. The data type of this port is **uint32**.

Output remote
Select this option to output the message remote frame status. This option adds a new output port to the block. The data type of this port is **uint8**.

Output timestamp
Select this option to output the message time stamp. This option adds a new output port to the block. The data type of this port is **double**.

Output length
Select this option to output the length of the message in bytes. This option adds a new output port to the block. The data type of this port is **uint8**.

Output error
Select this option to output the message error status. This option adds a new output port to the block. The data type of this port is **uint8**.

Output status
Select this option to output the message received status. The status is `1` if the block receives new message and `0` if it does not. This option adds a new output port to the block. The data type of this port is **uint8**.

If you do not select any **Output ports** option, the number of output ports on your block depends on the number of signals you specify.

Conversion Formula

The conversion formula is

```
physical_value = raw_value *   Factor + Offset
```

where `raw_value` is the unpacked signal value. `physical_value` is the scaled signal value which is saturated using the specified **Min** and **Max** values.

See Also

CAN Pack

UDP Receive

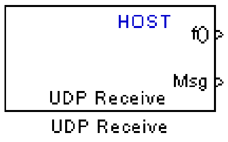
Purpose

Receive `uint8` vector as UDP message

Library

Host Communication (hostcommlib)

Description



A UDP message comes into this block from the transport layer. The block passes the message to the next downstream block. One block output provides the data vector from the message. The second output is a flag that indicates when a new UDP message is available.

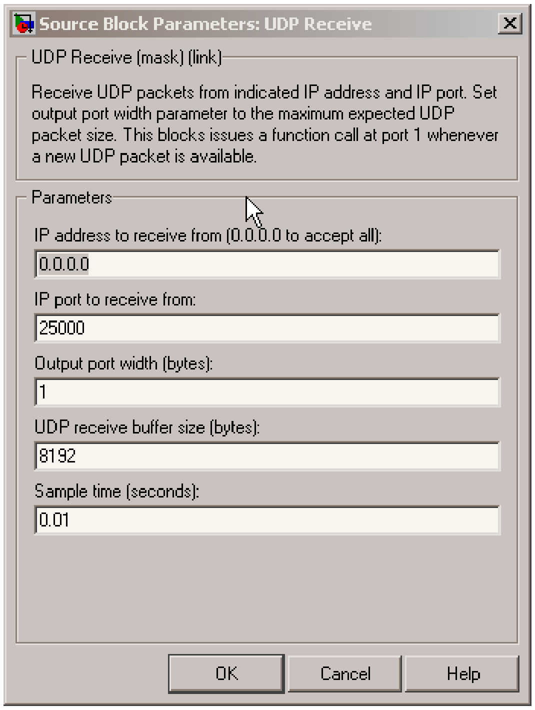
Models can contain only one UDP Receive block.

This block issues a function call from the `fcn` port when a new UDP packet becomes available. At the same time, it updates the signal going out of the `msg`port with the contents of the UDP packet. It reads a single UDP packet every sample hit. It does not attempt to receive multiple UDP packets to fill the output vector.

If the UDP packet size is greater than the output port width parameter, the system truncates the UDP messages at the Msg port. As a result, the system discards the part of the UDP packet that does not fit into the Msg port. The system cannot recover discarded message content.

In some cases, the UDP packet size is smaller than the Msg port width. When this condition occurs, the portion of the output vector that does not fit into the specified size processes as invalid data.

Dialog Box


Source Block Parameters: UDP Receive

UDP Receive (mask) (link)

Receive UDP packets from indicated IP address and IP port. Set output port width parameter to the maximum expected UDP packet size. This blocks issues a function call at port 1 whenever a new UDP packet is available.

Parameters

IP address to receive from (0.0.0.0 to accept all): 0.0.0.0

IP port to receive from: 25000

Output port width (bytes): 1

UDP receive buffer size (bytes): 8192

Sample time (seconds): 0.01

IP address to receive from (0.0.0.0 to accept all)
Specifies the IP address from which the block accepts messages. Setting the address 0.0.0.0 configures the block to accept messages from any IP address. Setting a specific address, instead of the default value, 0.0.0.0, directs the block to accept messages from the specified address only.

IP port to receive from
Specify the port the block accepts messages from on this machine. The other end of the communication, usually a UDP Send block, sends messages to this port. The value defaults to 25000, but the values range from 1–65535.

Output port width (bytes)

Specifies the width of messages that the block accepts. When you design the transmit end of the UDP communication channel, you decide the message width. Set this option to a value as large or larger than any message you expect to receive.

UDP receive buffer size (bytes)

Specify the size of the buffer to which the system stores UDP messages. The default size is 8192 bytes. Make the buffer large enough to store UDP messages that come in while your process reads a message from the buffer or performs other tasks. Specifying the buffer size prevents the receive buffer from overflowing.

Sample time (seconds)

Use this option to specify when the block polls for new messages. Enter a value that is greater than zero. Setting this option to a large value reduces the likelyhood of dropped UDP messages. By default, the sample time is 0.01 s.

See Also

Byte Pack, Byte Reversal, Byte Unpack, UDP Send

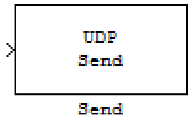
Purpose

Send UDP message

Library

Host Communication (hostcommlib)

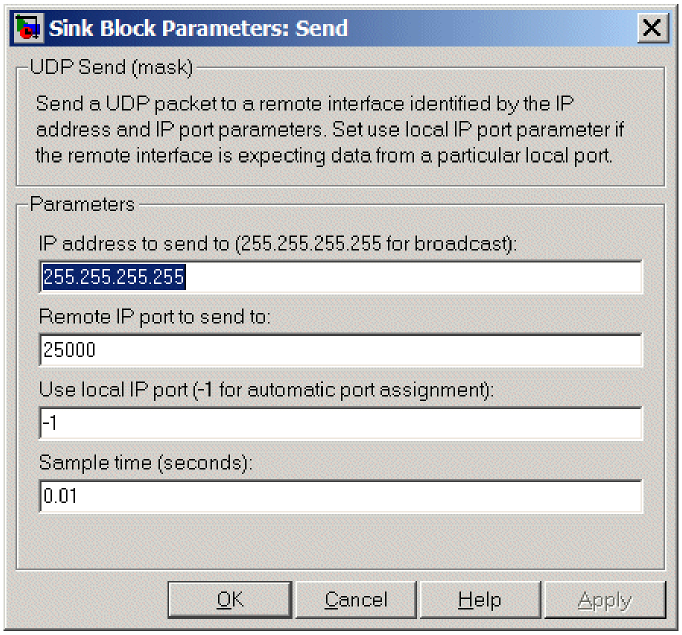
Description



The UDP send block receives a `uint8` vector that it sends as a UDP message to the host. Input must be in the form of a `uint8` vector with UDP format.

Models can contain only one UDP Send block.

Dialog Box



Sink Block Parameters: Send

UDP Send (mask)

Send a UDP packet to a remote interface identified by the IP address and IP port parameters. Set use local IP port parameter if the remote interface is expecting data from a particular local port.

Parameters

IP address to send to (255.255.255.255 for broadcast): 255.255.255.255

Remote IP port to send to: 25000

Use local IP port (-1 for automatic port assignment): -1

Sample time (seconds): 0.01

IP address to send to (255.255.255.255 for broadcast)

Specify the IP address to which the block sends the message. To broadcast the UDP message, retain the default value, 255.255.255.255.

IP port to send to

Specify the port to which the block sends the message. Port numbers range from 1 to 65535. Configure the network port receiving the UPD messages with the same port number.

Use the following local IP port

Specify the local IP port the block sends the message from. Entering `-1` (the default value) for this option allows the network to select automatically the local IP port to use to send the message.

If the address you are sending to expects the message to come from a specific port, enter that port address. If you enter a port number in the UDP Receive block option **IP port to receive from**, enter that port identifier instead of the port address.

Sample time

Sample time tells the block how long to wait before polling for new messages.

See Also

Byte PackByte Reversal, Byte Unpack, UDP Receive

Index